60256010

CONTROL DATA CORPORATION

CDC® CYBER 200 MODEL 203
COMPUTER SYSTEM

Preliminary Edition
HARDWARE REFERENCE MANUAL

COMPUTER INSTRUCTION INDEX

†These instructions have sign control capability.

††Automatic index incrementing takes place on these instructions. (Refer to the individual instruction descriptions.)

†††Delimiters may be used on these instructions; automatic index incrementing also takes place. (Refer to the individual instruction descriptions.)

60256010

CONTROL DATA CORPORATION

CDC® CYBER 200 MODEL 203

COMPUTER SYSTEM

Preliminary Edition

HARDWARE REFERENCE MANUAL

REVISION RECORD

REVISION	DESCRIPTION
01	Preliminary manual.
(04-15-78)	
02	Manual revised; makes miscellaneous technical and editorial corrections. Revised pages are:
(05-15-79)	Front Cover, Inside Front Cover, Title Page, ii through xvi, 1-1 through 1-4, 2-2, 2-5 through
	2-7, 3-1, 3-3, 3-5 through 3-7, 3-9, 3-12 through 3-14, 3-16 through 3-24, 3-30, 3-32, 3-35,
	3-36, 3-49, 3-50, 4-1, 4-2, 4-12, 4-16, 4-21, 4-22, 4-36, 4-37, 5-1, 5-10 through 5-12, 5-39,
	6-10, 6-11, 6-28, 6-60, 6-61, 6-62, 6-92, 6-94, 6-95, 6-99, 6-100, 6-101, 6-107, 6-117, 6-216,
	6-217, 6-221, 6-241, A8, A11, B14, B15, C2, and Comment Sheet. Added pages are: xvii,
	xviii, 6-60.1/6-60.2, 6-100.1/6-100.2, 6-216.1, 6-216.2, and 6-242.1/6-242.2. This manual
	contains preliminary information which is subject to change without notification to manual holders.
Publication No. 60256010	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Printed in the United States of America

Address comments concerning this manual to:
Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Inside Front Cover	-
Title Page	-
ii	02
iii	02
iv	02
v	02
vi	02
vii	02
viii	02
ix	02
x	02
xi	02
xii	02
xiii	02
xiv	02
xv	02
xvi	02
xvii	02
xviii	02
Divider	-
1-1	02
1-2	02
1-3	02
1-4	02
Divider	-
2-1	01
2-2	02
2-3	01
2-4	01
2-5	02
2-6	02
2-7	02
Divider	-
3-1	02
3-2	01
3-3	02
3-4	01
3-5	02
3-6	02
3-7	02
3-8	01
3-9	02
3-10	01
3-11	01
3-12	02
3-13	02
3-14	02
3-15	01
3-16	02
3-17	02
3-18	02
3-19	02
3-20	02
3-21	02
3-22	02
3-23	02
3-24	02
3-25	01
3-26	01
3-27	01
3-28	01
3-29	01
3-30	02
3-31	01
3-32	02
3-33	01
3-34	01
3-35	02
3-36	02
3-37	01
3-38	01
3-39	01
3-40	01
3-41	01
3-42	01
3-43	01
3-44	01
3-45	01
3-46	01
3-47	01
3-48	01
3-49	02
3-50	02
3-51	01
3-52	01
3-53	01
3-54	01
3-55	01
Divider	-
4-1	02
4-2	02
4-3	01
4-4	01
4-5	01
4-6	01
4-7	01
4-8	01
4-9	01
4-10	01
4-11	01
4-12	02
4-13	01
4-14	01
4-15	01
4-16	02
4-17	01
4-18	01
4-19	01
4-20	01
4-21	02
4-22	02
4-23	01
4-24	01
4-25	01
4-26	01
4-27	01
4-28	01
4-29	01
4-30	01
4-31	01
4-32	01
4-33	01
4-34	01
4-35	01
4-36	02
4-37	02
Divider	-
5-1	02
5-2	01
5-3	01
5-4	01
5-5	01
5-6	01
5-7	01
5-8	01
5-9	01
5-10	02
5-11	02
5-12	02
5-13	01
5-14	01
5-15	01
5-16	01
5-17	01
5-18	01
5-19	01
5-20	01
5-21	01
5-22	01
5-23	01
5-24	01
5-25	01
5-26	01
5-27	01
5-28	01
5-29	01
5-30	01
5-31	01
5-32	01
5-33	01
5-34	01
5-35	01
5-36	01
5-37	01
5-38	01
5-39	02
5-40	01
5-41	01
5-42	01
5-43	01
5-44	01
Divider	-
6-1	01
6-2	01
6-3	01
6-4	01
6-5	01
6-6	01
6-7	01
6-8	01
6-9	01
6-10	02
6-11	02
6-12	01
6-13	01
6-14	01
6-15	01
6-16	01
6-17	01
6-18	01
6-19	01
6-20	01
6-21	01
6-22	01
6-23	01
6-24	01
6-25	01
6-26	01
6-27	01
6-28	02
6-29	01
6-30	01
6-31	01
6-32	01
6-33	01
6-34	01
6-35	01
6-36	01
6-37	01
6-38	01
6-39	01
6-40	01
6-41	01
6-42	01
6-43	01
6-44	01
6-45	01
6-46	01
6-47	01
6-48	01
6-49	01
6-50	01
6-51	01
6-52	01
6-53	01
6-54	01
6-55	01
6-56	01
6-57	01
6-58	01
6-59	01
6-60	02
6-60.1/ 6-60.2	02
6-61	02
6-62	02
6-63	01
6-64	01
6-65	01
6-66	01
6-67	01
6-68	01
6-69	01
6-70	01
6-71	01
6-72	01
6-73	01
6-74	01
6-75	01
6-76	01
6-77	01
6-78	01
6-79	01
6-80	01
6-81	01
6-82	01
6-83	01
6-84	01
6-85	01
6-86	01
6-87	01
6-88	01
6-89	01
6-90	01
6-91	01
6-92	02
6-93	01
6-94	02
6-95	02
6-96	01
6-97	01
6-98	01
6-99	02
6-100	02
6-100.1/ 6-100.2	02
6-101	02
6-102	01
6-103	01
6-104	01
6-105	01
6-106	01
6-107	02
6-108	01
6-109	01
6-110	01
6-111	01
6-112	01
6-113	01
6-114	01
6-115	01
6-116	01
6-117	02
6-118	01
6-119	01
6-120	01
6-121	01
6-122	01

PAGE	REV
6-123	01
6-124	01
6-125	01
6-126	01
6-127	01
6-128	01
6-129	01
6-130	01
6-131	01
6-132	01
6-133	01
6-134	01
6-135	01
6-136	01
6-137	01
6-138	01
6-139	01
6-140	01
6-141	01
6-142	01
6-143	01
6-144	01
6-145	01
6-146	01
6-147	01
6-148	01
6-149	01
6-150	01
6-151	01
6-152	01
6-153	01
6-154	01
6-155	01
6-156	01
6-157	01
6-158	01
6-159	01
6-160	01
6-161	01
6-162	01
6-163	01
6-164	01
6-165	01
6-166	01
6-167	01
6-168	01
6-169	01
6-170	01
6-171	01
6-172	01
6-173	01
6-174	01
6-175	01
6-176	01
6-177	01
6-178	01
6-179	01
6-180	01
6-181	01
6-182	01
6-183	01
6-184	01
6-185	01
6-186	01
6-187	01
6-188	01
6-189	01
6-190	01
6-191	01
6-192	01
6-193	01
6-194	01
6-195	01
6-196	01
6-197	01
6-198	01
6-199	01
6-200	01
6-201	01
6-202	01
6-203	01
6-204	01
6-205	01
6-206	01
6-207	01
6-208	01
6-209	01
6-210	01
6-211	01
6-212	01
6-213	01
6-214	01
6-215	01
6-216	02
6-216.1	02
6-216.2	02
6-217	02
6-218	01
6-219	01
6-220	01
6-221	02
6-222	01
6-223	01
6-224	01
6-225	01
6-226	01
6-227	01
6-228	01
6-229	01
6-230	01
6-231	01
6-232	01
6-233	01
6-234	01
6-235	01
6-236	01
6-237	01
6-238	01
6-239	01
6-240	01
6-241	02
6-242	01
6-242.1/ 6-242.2	02
6-243	01
6-244	01
6-245	01
6-246	01
6-247	01
Divider	-
A-1	01
A-2	01
A-3	01
A-4	01
A-5	01
A-6	01
A-7	01
A-8	02
A-9	01
A-10	01
A-11	02
A-12	01
A-13	01
A-14	01
A-15	01
A-16	01
A-17	01
A-18	01
A-19	01
Divider	-
B-1	01
B-2	01
B-3	01
B-4	01
B-5	01
B-6	01
B-7	01
B-8	01
B-9	01
B-10	01
B-11	01
B-12	01
B-13	01
B-14	02
B-15	02
B-16	01
B-17	01
B-18	01
B-19	01
B-20	01
B-21	01
B-22	01
B-23	01
B-24	01
Divider	-
C-1	01
C-2	02
C-3	01
C-4	01
C-5	01
C-6	01
C-7	01
Divider	-
D-1	01
D-2	01
D-3	01
D-4	01
Comment Sheet	02
Back Cover	-

PREFACE

This manual contains hardware reference information for the CDC® CYBER 200 Model 203 Computer System.

RELATED PUBLICATIONS

Other manuals applicable to the CDC CYBER 200 Model 203 Computer System and associated equipment include the following.

Control Data Publication	Publication Number
CYBER 200 Model 203 Computer System Refrigeration System Hardware Maintenance Manual	60329810
INTEBRID® Circuits Customer Engineering Manual	60201000
Motor-Generator Sets Electric Machinery Hardware Maintenance Manual, Volume 1 of 2	60166800
Motor-Generator Sets Electric Machinery Hardware Maintenance Manual, Volume 2 of 2	60423100
Large- and Medium-Scale Computer Systems Site Preparation Manual Section 1 - General Information	60275100
CYBER 200 Model 203 Computer System Site Preparation Manual Section 2 - System Data	60381610
STAR Peripheral Stations Hardware Reference Manual	60405000
STAR Peripheral Stations Hardware Maintenance Manual (General Description, Maintenance, Installation, Cabling, and Power Distribution Diagrams)	60325300

Control Data Publication	Publication Number
STAR Peripheral Stations Hardware Maintenance Manual (Diagrams)	
Station Buffer Unit Chassis 0 (Core Control)	60382000 and 60406700
Station Buffer Unit Chassis 1 (Interfaces)	60382100 and 60406800
Station Control Unit	60362900
Station Display Unit	60382500

These manuals are available on a controlled distribution basis only from:

Control Data Corporation
CYBER 200 Publications Distribution
4290 Fernwood Avenue
St. Paul, Minnesota 55112

DISCLAIMERS

This manual contains preliminary information which is subject to change without notification to manual holders.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or undefined parameters.

CONTENTS

APPENDIXES

FIGURES

TABLES

GENERAL DESCRIPTION 1

GENERAL

The CDC CYBER 200 Model 203 Computer System (central computer) is a large-scale, high-speed computer containing features such as stream processing, virtual addressing, and hardware macro instructions. The central computer uses large-scale integrated (LSI) circuits in a new scalar processor to improve scalar performance. The central computer contains separate scalar and vector processors specifically designed for sequential and parallel operations on single bits, 8-bit bytes, and 32- or 64-bit floating-point operands and vector elements. The central memory is a high-performance semiconductor memory with single error correction double error detection (SECDED) on each 32-bit half-word for high storage integrity. The virtual addressing method employs a high-speed mapping technique to convert a logical address to an absolute storage address.

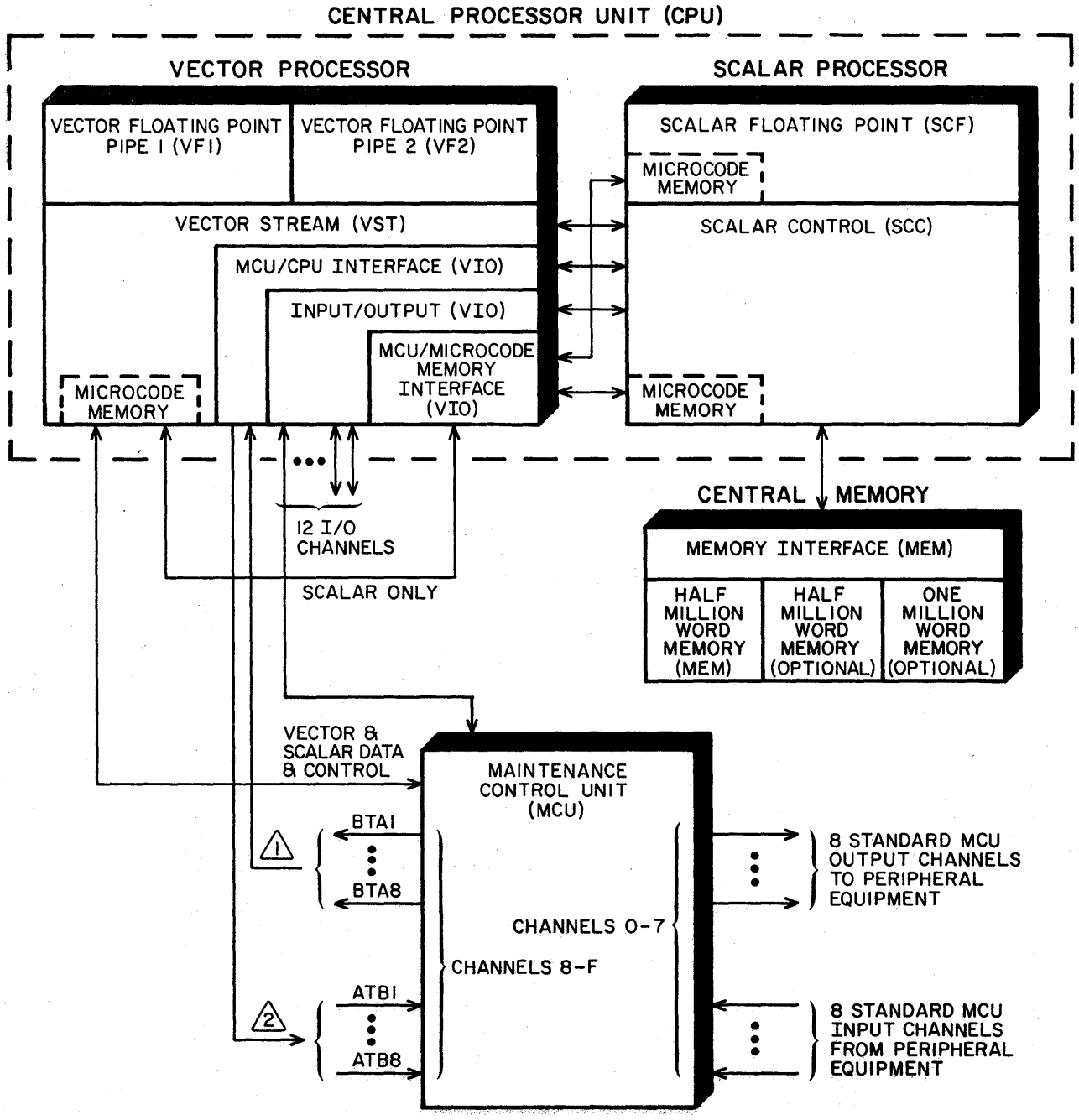
The instruction issue rate is one every 20 nanoseconds.

The basic central computer consists of a central processor unit (CPU), one-half million 64-bit words of central memory, 12 input/output (I/O) channels, and a maintenance control unit (MCU). Figure 1-1 shows the basic central computer configuration.

The central memory is field-expandable to one or two million 64-bit words by adding units of one-half million words.

The CPU contains the vector and scalar processors and the I/O channels.

The vector processor contains a stream unit that peforms vector streaming and instruction control, operand alignment, buffering, and addressing. The stream unit receives vector instructions from the scalar processor, executes instructions not in conflict with the scalar processor, manages the data streams of the vector floating-point pipes, and performs string processing.



NOTES:

1. 8 PULSED NORMAL MCU OUTPUT CHANNELS.
2. 8 PULSED NORMAL MCU INPUT CHANNELS.

Figure 1-1. Basic Central Computer Configuration

The scalar processor receives and decodes instructions from central memory, directs decoded vector/string instructions to the vector processor for execution, provides an orderly buffering and execution of load/store instructions, and controls communication with central memory by the CPU and I/O channels.

Microcode memories in both the scalar and vector processors control setup, interrupt, and termination of instructions.

The I/O channels consist of control units for 16-bit data communications between the scalar processor and the MCU and between the scalar processor and the peripheral stations. Any one of the I/O channels connects to the MCU and the other channels connect to the peripheral stations.

The MCU provides special maintenance control and monitoring capabilities.

Cooling for the basic central computer, including the first one-half million words of memory, consists of two 30-ton water-cooled condensing units. A one-million-word system configuration requires a total of 90 tons of cooling, and a two-million-word configuration requires a total of 120 tons of cooling.

One 250-kVA 400-Hz motor-generator set provides power for a one-half-million-word or one-million-word configuration. A two-million-word configuration has an additional 80-kVA 400-Hz motor-generator set. Each system configuration has a 250-kVA 400-Hz standby motor-generator set.

CENTRAL COMPUTER CHARACTERISTICS

CENTRAL PROCESSOR

- Two's complement arithmetic
- Parallel/dual segmented arithmetic units
- Hardware register file composed of 256 64-bit addressable registers
- Integrated and LSI circuits
- Hardware macro instructions
- Sequential stream processing
- Synchronous internal logic with 40-nanosecond clock period (minor cycle) for vector and 20-nanosecond clock period (minor cycle) for scalar

CENTRAL MEMORY

- 1K, bipolar semiconductor memory chips
- Virtual addressing
- 16 memory stacks of 65,536 32-bit halfwords each obtaining a total storage of 524,288 64-bit words
- Optional one or two million 64-bit words
- Stack arrangement in eight phased banks
- Data transferred to/from memory ports in 32-bit halfwords, 64-bit words, and 512-bit swords
- Two levels of memory degradation

INPUT/OUTPUT

- Four 16-bit I/O channels
- 12 I/O channels
- Highly flexible peripheral stations

CENTRAL MEMORY 2

The central memory is a random-access memory using 1K bipolar semiconductor circuits. A memory word has 78 bits: a 64-bit data word and 14 bits for single error correction double error detection (SECDED). The cycle time for the semiconductor memory is 80 nanoseconds. The memory is directly addressable in monitor mode and via hardware virtual relocation in job mode.

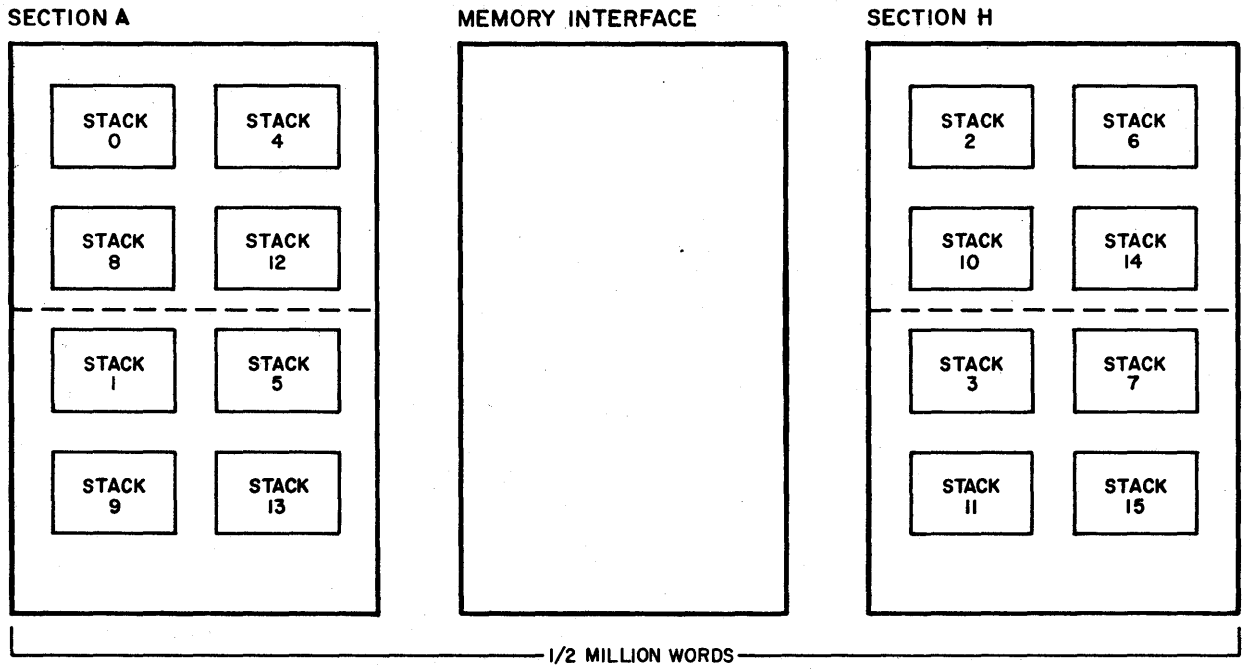
The central memory size is one-half million words with field upgrade options allowing expansion to one or two million words.

Each one-half million words of central memory contains 16 memory stacks, each having 64K 39-bit halfwords (32 data bits plus 7 SECDED bits). Each 64K stack is arranged in eight phased banks. Memory can assign sequential addresses to different banks by using bank phasing. Because the banks are independent, a bank can begin a memory cycle before adjacent banks have completed previously initiated cycles. In streaming mode, a reference is made simultaneously to the same address in each of the 16 memory stacks obtaining a superword (sword) of 512 data bits. Each one-half million words of memory contains 128 phased halfword banks. Figure 2-1 shows the chassis configuration for one-half million words of memory.

MEMORY OPERATION

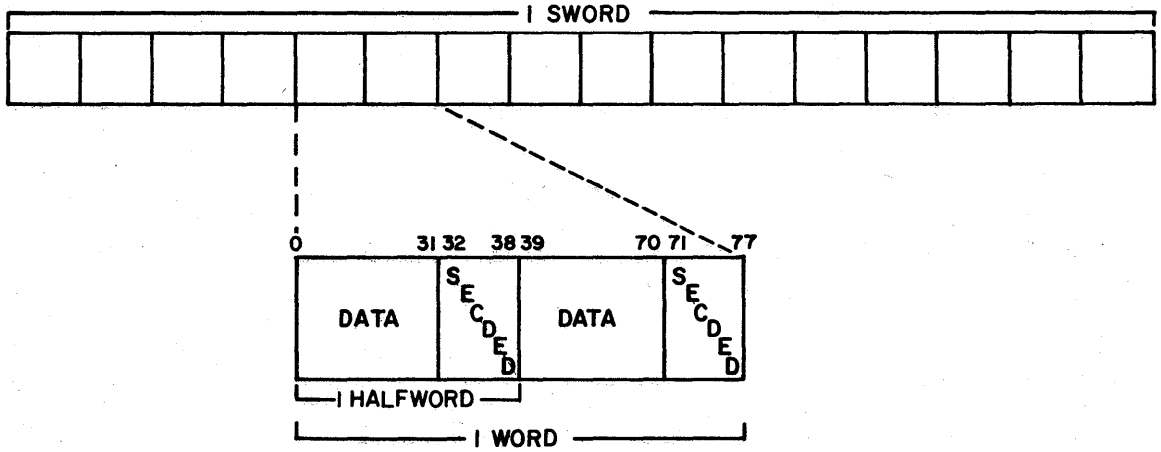
A memory word can be a sword, a word, or a halfword. One sword contains eight 78-bit (64 bits for data and 14 bits for SECDED) words addressed from left to right.

The 624 bits (512 data bits, 112 SECDED bits) of one sword transfer to/from central memory during each write/read operation, although only part of the sword may actually be stored or used. When the memory interface performs a write/read operation on a sword, it addresses each of the 16 memory stacks (figure 2-2). For write/read operations on a word, the memory interface addresses only two of the 16 stacks and for a halfword, the memory interface addresses only one of the 16 stacks. In addition, the memory interface sends a bank address signal that selects only one of the eight banks within a stack. The data signals go through 39-bit data trunks which go to each of the 16 memory stacks. During a write operation, the memory interface sends a write control signal for each halfword (32 bits). Depending on the write control, any or all of the halfwords within the sword may be written into memory.



NOTES:
1. EACH SECTION HAS 8 STACKS.
2. TWO SECTIONS COMPRISE A HALF MILLION WORDS OF MEMORY.

Figure 2-1. Section Configuration



NOTES:
1. EACH SWORD CONTAINS 16 HALFWORDS.
2. EACH HALFWORD CONTAINS 39 BITS (32 DATA BITS, 7 SECDED BITS).
3. MEMORY TRANSFERS MAY BE IN SWORDS, WORDS, OR HALFWORDS.

Figure 2-2. Superword (sword) Configuration

The memory size determines the number of memory stacks available. For the basic one-half million words of memory, 16 memory stacks are available; for one million words of memory, 32 memory stacks are available; and for two-million words of memory, 64 memory stacks are available.

SECDED checking and generating is accomplished in the scalar processor (refer to section 3).

MEMORY ACCESS AND CONTROL

Figure 2-3 shows the control signals sent to each memory stack. All signals except the read data are sent from the memory interface to the stacks. The read data signal is sent back to the memory interface.

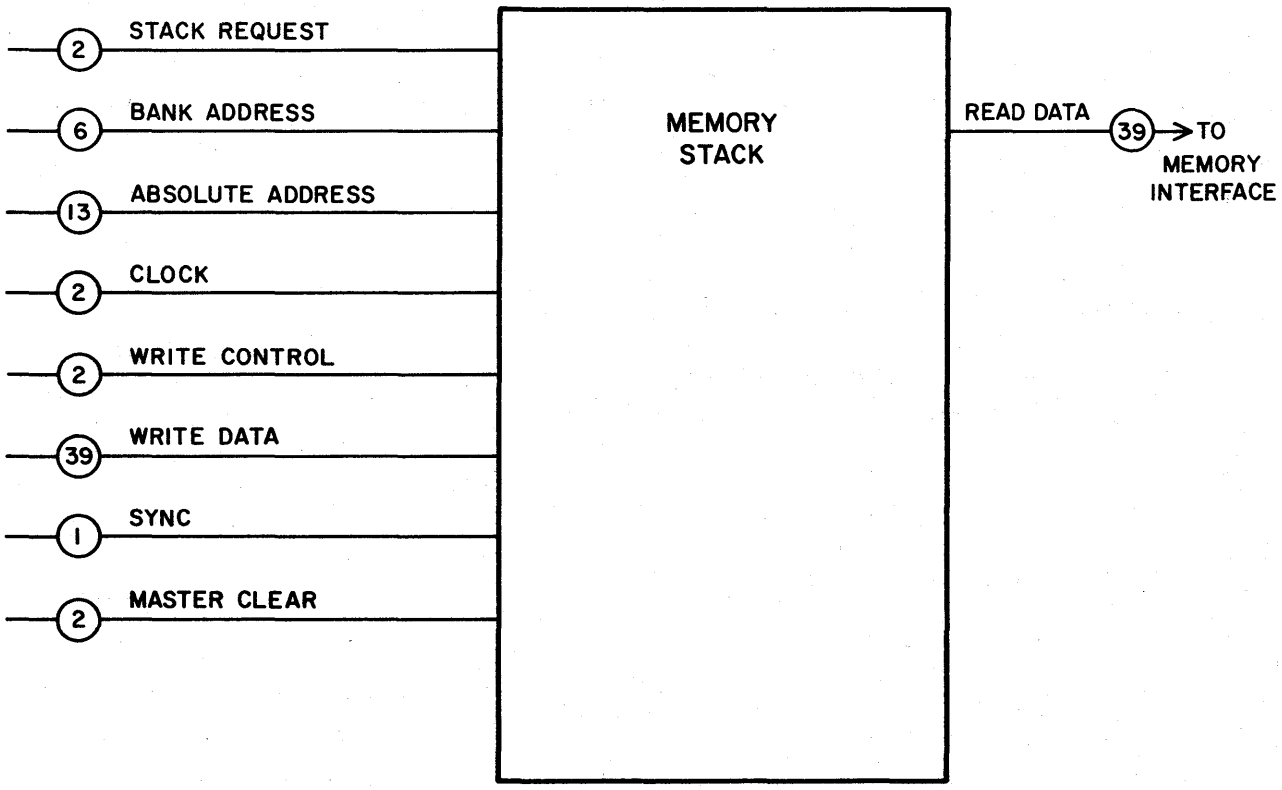


Figure 2-3. Memory Interface/Stack Connections

STACK REQUEST

There are two identical stack request lines for each memory stack. This signal determines which stack has been selected.

BANK ADDRESS

There are six (two sets of three) bank address lines for each memory stack. This signal determines which bank of the eight banks within a stack has been selected.

ABSOLUTE ADDRESS

There are 13 bits that determine the absolute address; three bits determine which of the eight ranks of memory chips has been selected and 10 bits determine the address in memory selected.

CLOCK

There are two identical clock lines for each memory stack. This signal synchronizes the memory stack to the memory interface.

WRITE CONTROL

There are two identical write control lines for each memory stack. This signal informs the memory stack of a write memory cycle.

WRITE DATA

There are 39 write data bit lines for each memory stack: 32 for data and 7 for SECDED.

SYNC

This signal provides a point of reference for maintenance purposes.

MASTER CLEAR

There are two identical master clear lines for each memory stack. The memory interface pulses the master clear signal continuously whenever a master clear is present in the CPU.

READ DATA

The 39 read data bits are obtained from the read data registers on the output, and the information is sent back to the memory interface.

MEMORY INTERFACE

The memory interface provides ports for access to central memory. The scalar processor, vector processor, and I/O channels are connected to central memory through the memory interface as shown in figure 2-4. Data transmissions are controlled by the priority unit in the scalar processor. SECDED for each 32 bits of data on the memory ports is done in the scalar processor. Data can be transferred to and from the memory ports in 32-bit halfwords, 64-bit words, or 512-bit swords (refer to table 2-1).

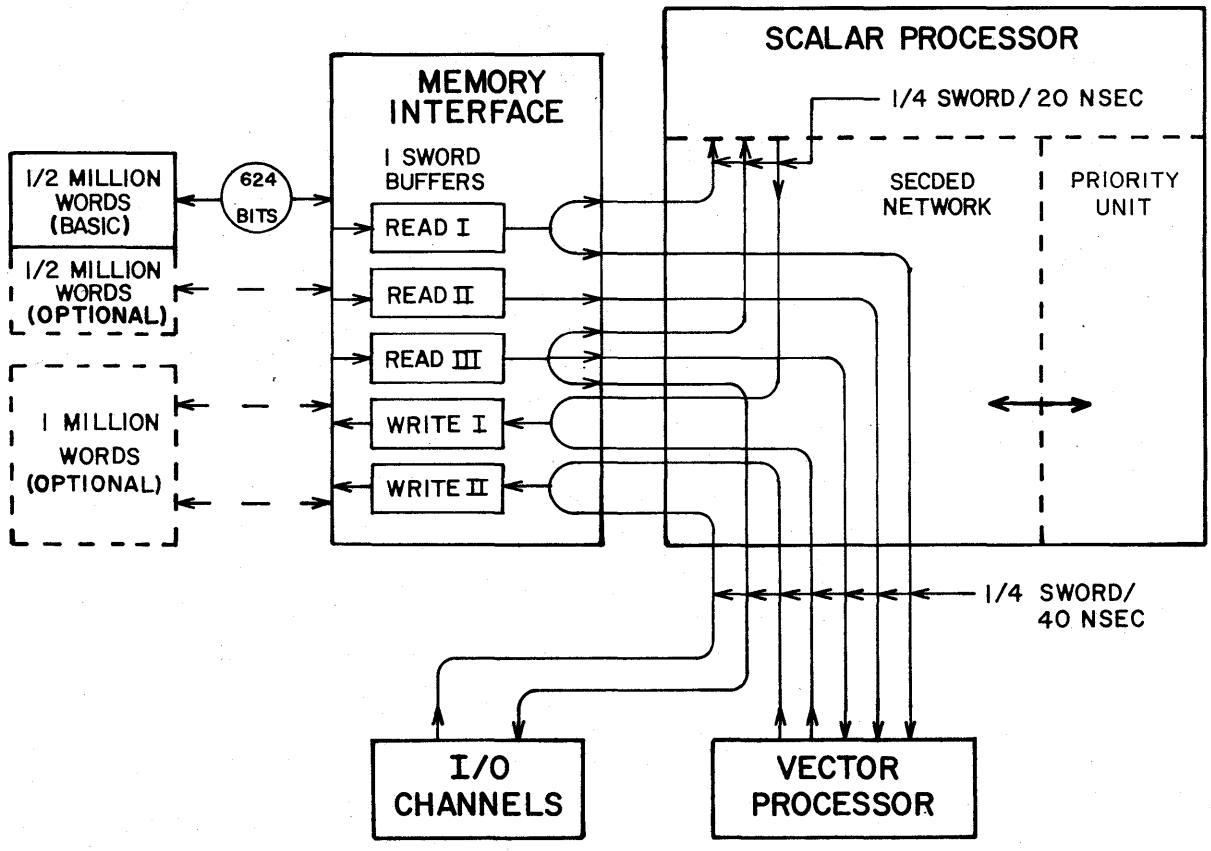


Figure 2-4. Memory Interface Configuration and Connections

Each memory port is connected to memory through a one sword buffer located in the memory interface. If a buffer is shared by multiple ports, the priority unit provides proper port selection to the memory interface selection network. Data is transmitted between the buffers and the processor in quarter swords at a rate of one quarter-word per minor cycle. For the scalar processor, the rate is one quarter-word per 20 nanoseconds; for the vector processor and I/O channels, the rate is one quarter-word per 40 nanoseconds.

TABLE 2-1. MEMORY PORT TRANSFER MODES

Memory Interface Buffer	Memory Port	Transfer Mode
Read 1	Scalar processor	Halfword Word Sword
	Vector processor	Sword
Read 2	Vector processor	Sword
Read 3	Read next sword (RNS) (scalar processor)	Sword
	I/O channels	Sword
	Vector processor	Sword
Write 1	Scalar processor	Halfword Word Sword
	Vector processor	Sword
Write 2	I/O channels	Sword
	Vector processor	Sword

MEMORY DEGRADATION

If more than one-half million words of memory are present, degradation may be selected. Degradation allows the amount of usable memory to be less than the total memory in the system. Three degradation bits from the MCU and a strobe bit control the amount of usable memory. Table 2-2 shows the memory degradation codes and their descriptions.

TABLE 2-2. MEMORY DEGRADATION CODES

System Memory	3-bit Code	Memory Size	Description
0.5 Meg	000	0.5 Meg	
1.0 Meg	001	0.5 Meg	Force section 1→section 0
2.0 Meg	010	0.5 Meg	Force section 2→section 0
2.0 Meg	011	0.5 Meg	Force section 3→section 0
1.0 Meg	100	1.0 Meg	
2.0 Meg	101	1.0 Meg	Force upper Meg→lower Meg
2.0 Meg	110	2.0 Meg	

For example, if the system has one million words of memory, the upper 0.5 Meg words of memory can be forced to the lower 0.5 Meg words of memory by selecting code 001. Figure 2-5 shows the four 0.5 Meg sections comprising a two-million-word memory system.

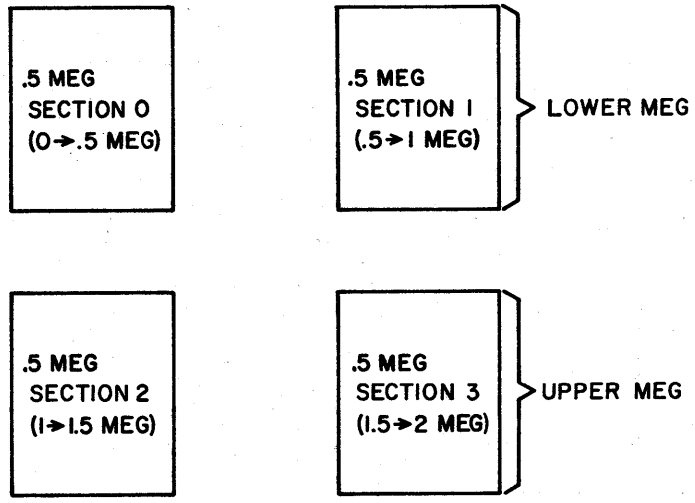


Figure 2-5. Memory Sections Configuration

CENTRAL PROCESSOR UNIT 3

DESCRIPTION

The central processor unit (CPU) consists of the following functional areas:

- Scalar processor
- Vector processor
- Input/output (I/O)

SCALAR PROCESSOR

The scalar processor receives and decodes all instructions from central memory, directs decoded vector/string instructions to the vector processor for execution, and provides orderly buffering and execution of the load and store instructions. To attain high scalar performance, the processor contains independent functional units.

The scalar processor contains the central computer instruction control. The instruction issue pipe receives and decodes instructions from central memory. A semiconductor instruction stack provides buffering for eight virtually addressed swords (512 bits), which can contain up to 128 32-bit instructions, 64 64-bit instructions, or a combination of both. The instruction stack can contain up to six nonadjacent swords with two swords lookahead. The read next sword (RNS) portion of the RNS/branch unit provides the control for loading the instruction stack. The branch portion performs branch condition testing and executes the branch instructions.

The instruction issue pipe is capable of issuing instructions at a rate of one instruction every 20 nanoseconds. The instruction issue pipe decodes instructions and directs decoded vector/string instructions to the vector processor for execution. Therefore, with independent vector and scalar instruction controls operating on a single instruction stream, the scalar processor can execute scalar instructions in parallel with most vector instructions if there are no memory references generated by the scalar instruction.

There are two exceptions to the parallel execution of vector and scalar instructions on a single instruction stream:

- The scalar processor can make memory references to load the instruction stack in parallel with vector operations.
- A vector instruction will not issue until a scalar instruction has placed the results in the register file. A scalar instruction that follows a vector instruction must wait until the vector instruction has completed its use of the register file.

The load/store unit provides orderly buffering and execution of the load and store instructions. The unit acts as a pipeline and is capable of accepting a new request rate of one load every minor cycle or one store every two minor cycles, if a memory busy, access interrupt, or register file write-bus busy does not occur. A circular buffer containing six registers provides buffering for up to six load requests, three store requests, or a combination of loads and stores.

The load/store unit is capable of loading a randomly accessed word of data from central memory into the register file in 300 nanoseconds after reading the base address and item count of the data. This time assumes a memory busy, access interrupt, or register file write-bus busy does not occur. A memory busy would add up to 80 nanoseconds to the load time.

The scalar floating-point contains independent functional units to attain high scalar performance. The following times (in nanoseconds) are required to produce a 32- or 64-bit result in each functional unit. These times correspond to the shortstop times. Shortstop is the process by which a result from any arithmetic unit may be returned directly to either input of any arithmetic unit. This occurs in parallel with the storing of the result in the register file. Shortstop eliminates the time necessary to store the result in the register file and retrieve it for use in the next arithmetic operation.

Unit	Time (nanoseconds)
Add/Subtract pipe	100
Multiply pipe	100
Logical pipe	60
Single cycle pipe	20
Divide/square root/convert unit	1120

The pipe units are segmented and capable of accepting new operands every 20 nanoseconds. The divide/square root/convert unit must complete each operation before a new one can begin. All units are capable of being shortstopped. The scalar processor contains a semiconductor register file providing 256 64-bit registers for use in instruction and operand addressing, indexing, field lengths, and as source and destination registers for scalar instruction operands and results. The register file is capable of two reads and one write every 20 nanoseconds.

The central computer virtual memory feature allows the use of advanced techniques of memory management and user program protection. Some of these features are:

- Key and lock for memory protection and user separation
- Hardware mapping from virtual to physical addresses
- An ordered page table to minimize operating system overhead
- Program overlays at execution time formed by the hardware system transparent to the user's program
- Sharing of user programs or data with other users
- Small page sizes of 512, 2048, and 8192 64-bit words selectable by an operating system software installation parameter. Only one small page size can reside in the associative page table at a time. The default is a small page size of 512 words
- Large page size of 65,536 64-bit words

The associative unit in the scalar processor contains the page table virtual addressing mechanism, which is composed of 16 associative registers and a space table (located in a restricted area of central memory) with sufficient entry space for up to two million words of central memory.

The page table is an ordered list of the associative words necessary to define the pages in absolute memory. The most frequently used associative words are at the top of the table. The space table is an extension of the page table containing the associative words necessary to define pages in memory that have not been in recent use. The associative unit is capable of comparing the associative registers in one minor cycle with the space table entries at the rate of two entries per minor cycle.

For the user, the paging mechanism and the operating system software permit the most active portions (pages) of a user program to reside in the central memory. These program portions can reside in nonadjacent areas of the central memory. The virtual addressing facility, through the page table, makes these areas of memory appear to be adjacent. The paging mechanism ensures that a large number of users can have simultaneous access to the central computer with minimum page swapping overhead.

The scalar processor has synchronous internal logic implemented with LSI circuits. This synchronous internal logic has a clock period of 20 nanoseconds. Figure 3-1 shows the functional components of the scalar processor.

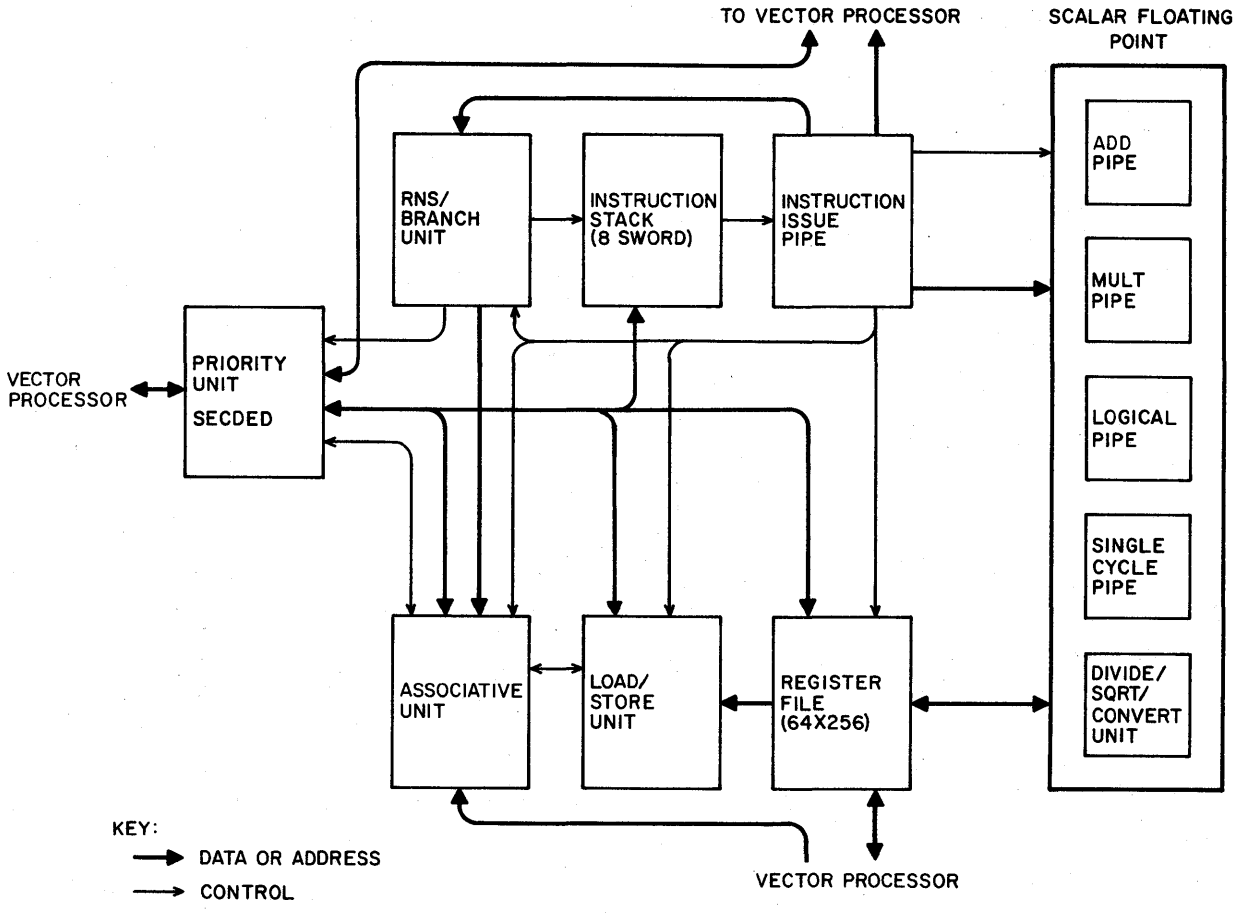


Figure 3-1. Functional Components of Scalar Processor

PRIORITY UNIT

The priority unit receives memory requests from various units within the CPU. There are two categories of requests: scalar processor requests and vector processor-I/O (VST/IO) requests. If simultaneous requests arrive in the priority unit, only the request with the highest priority is processed further.

The bank address (bits 50 through 58) associated with the surviving request is then gated into the priority unit, enabling various bank checks to be performed. Since memory is banked to the halfword level, a sword request causes 16 memory banks to go busy. The bank busies do not begin until the request issues. Once initiated, a memory is busy for four minor cycles. All requests are scalar or vector processor-I/O (VST/IO).

Each VST/IO request has an implied data quantity of one sword. A delay of nine minor cycles occurs between the priority unit's receipt of a VST/IO request and the transmission of the stack request strobe to the memory interface unit (causing the appropriate memory banks to cycle). The VST/IO request is issued when the stack request strobe is transmitted. For a VST/IO write, this delay is necessary to enable the sword buffer in the memory interface to accumulate four quarter-swords of data before the memory is cycled. All VST/IO data is transmitted at a rate of one quarter-sword every 40 nanoseconds.

Each scalar processor request except load/store (L/S) has an implied data quantity of one sword. The L/S request is accompanied by a signal that indicates whether the data quantity is a halfword or word. A delay of two minor cycles occurs between the receipt of the request in the priority unit and the transmission of the stack request strobe. An exception is a scalar processor sword write which is delayed an additional three minor cycles to enable the sword buffer in the memory interface to accumulate four quarter-swords of data. All scalar processor data is transmitted at a rate of one quarter-sword every 20 nanoseconds.

BANK BUSY CHECKS

When the surviving memory request is from the VST/IO, the only bank busy check required is to determine if another VST/IO request with the same sword address has been processed in the three previous minor cycles.

When the surviving memory request is from the scalar processor, the bank busy checks are more complex. Because VST/IO requests do not issue until nine minor cycles after arrival in the priority unit, the sword portion of the bank address associated with a scalar processor request must be different from the bank addresses of any VST/IO request arriving 10, 11, or 12 minor cycles earlier. If it is not, the scalar processor request issues to a busy bank. Also, a check is made to ensure that no VST/IO requests to any bank were honored nine minor cycles earlier. If they were, an attempt to honor the scalar processor request means a conflict on the single address bus from the priority unit to the memory interface due to the issuing of a VST/IO request.

The sword portion of the bank address associated with the scalar processor request must be different from the bank addresses of any VST/IO requests arriving six, seven, or eight minor cycles earlier. If it is not, the issuing VST/IO requests are directed to a busy bank, violating a nonbusy bank guarantee by the VST/IO accept.

Additionally, a check is performed to ensure that the scalar processor request will not encounter bank busies due to other scalar processor requests having been honored during the previous three minor cycles. When checking a scalar processor request against a prior scalar processor request honored one, two, or three minor cycles earlier, the larger of the two data quantities involved determines the number of banks tested for busies. If the larger of the two data quantities is a sword, 16 banks are checked; if the larger is a word, two banks are checked; and if the larger is a halfword, only one bank is checked.

MEMORY INTERFACE BUFFER CHECKS

The memory interface contains three read buffers and two write buffers for assembling and disassembling data. A request is checked to ensure that it does not require the use of an active buffer and does not interfere with the buffer requirements of a previous request not yet issued.

If a request passes all memory and buffer busy checks, an accept signal is returned to the requesting source.† This accept signal guarantees the requested data will be transferred to or from memory.

Requests caused by a space table search or an exchange are not sent unless the source is certain no busies will be encountered. Therefore, accepts are not necessary for these requests.

MEMORY INTERFACE SIGNALS

After an accept signal is honored by the priority unit, appropriate control signals are sent to the memory interface allowing the requested data transfer to be performed. These control signals are grouped as follows:

- Buffer control signals
  Provide the data buffer in the memory interface with information concerning the direction, rate, and quantity of data flow.
- Nine bank address bits
  Define the lowest-numbered memory bank involved in the data transfer.

† If the request requires virtual addressing, a match is also required for an accept to be returned.

- Stack request strobe
  Causes the preselected memory banks to cycle.

The buffer control signals are sent as soon as the signal request is honored in the priority unit. The bank address signals are sent one minor cycle before the request issues. The read buffer control signals and the stack request strobe are sent at issue time.

AUXILIARY PRIORITY UNIT FUNCTIONS

The priority unit also performs a number of auxiliary functions. For example, the absolute address produced by the associative register is stored in the priority unit until the request issues. After a memory read, the priority unit strobes the SECDED checkers as the data is processed. The address of the original request is also saved, so if a SECDED error occurs, the failing address is recorded.

SINGLE ERROR CORRECTION DOUBLE ERROR DETECTION (SECDED)

The central computer has 10 SECDED units within the scalar processor: four write and six read units. The four write or generate units are:

- Write 1 VST
- Write 2 VST
- Write I/O
- Write scalar

The six read or checker units are:

- Read 1 VST
- Read 2 VST
- Read 3 VST
- Read I/O
- Read next instruction (RNI)
- Read scalar

The SECDED error information is stored by the maintenance control unit (MCU). The stored information is the syndrome word, single error, double error, read bus code, and CPU word address bits 37 through 58.

SYNDROME WORD

The error correcting code generates the seven syndrome bits. There are 39 (odd bit) unique syndrome words; only the 32 data bit codes toggle a bit when error correction is enabled. Other odd bit codes stored in SECDED, differing from the 39 unique syndrome words, are flagged by the MCU as multiple add bit errors. Double error syndrome words have an even number of bits.

SINGLE ERROR

Bit 5 of channel ATB8 is the single error bit and sets if there is a single error not preceded by a double error.

DOUBLE ERROR

This MCU display register sets unconditionally on a double error.

READ BUS CODES

The read bus codes are MCU display registers defining the read bus on which the SECDED error occurs.

Code	Read Bus
0	I/O
1	Read 1
2	Read 2
3	Read 3
4	Scalar
5	RNI

The error logging priority for simultaneous SECDED errors on multiple buses is:

1. RNI
2. Scalar
3. R2
4. R1
5. I/O
6. R3

CPU WORD ADDRESS BITS (37 THROUGH 58)

The CPU word address bits are divided into the halfword and word address bits.

The halfword address bits (bits 57 and 58) decode the four 32-bit groups within one quarter-sword. The word address bits (bits 37 through 56) indicate the following:

Bit	Description
37-39	Select 1 of 8 memory chips/bank
40-49	Select 1 of 1024 words/chip
50	1024K select
51	512K select
52-54	Bank select
55-56	Quarter sword select

Address bits 37 through 58 in SECDED are always the CPU word address bits.

Table 3-1 shows the unique syndrome words for single bit failures.

TABLE 3-1. UNIQUE SYNDROME WORDS FOR SINGLE BIT FAILURES

Bit	Data	Syndrome Word
0	Check Bit 0	40
1	Check Bit 1	20
2	Check Bit 2	10
3	Check Bit 3	08
4	Check Bit 4	04
5	Check Bit 5	02
6	Check Bit 6	01
7	80000000	70
8	40000000	68
9	20000000	58
10	10000000	64
11	08000000	54
12	04000000	7C
13	02000000	7A
14	01000000	76
15	00800000	1C
16	00400000	1A

TABLE 3-1. UNIQUE SYNDROME WORDS FOR SINGLE BIT FAILURES (Contd)

Bit	Data	Syndrome Word
17	00200000	16
18	00100000	19
19	00080000	15
20	00040000	1F
21	00020000	5E
22	00010000	5D
23	00008000	07
24	00004000	46
25	00002000	45
26	00001000	26
27	00000800	25
28	00000400	67
29	00000200	57
30	00000100	37
31	00000080	61
32	00000040	51
33	00000020	31
34	00000010	49
35	00000008	29
36	00000004	79
37	00000002	75
38	00000001	6D

The syndrome word is stored (latched) if the bit shown in the data pattern in table 3-1 is in error. For example, if only bit 0 failed on any data pattern, the syndrome word would be 40.

SECDED ERROR LATCHING HARDWARE

The SECDED error latching hardware has two modes of operation: mode 1 and mode 2. Mode selection is accomplished through the MCU/CPU maintenance line called select SECDED error log mode 2.

For simultaneous SECDED errors in both modes, the error latch information to be latched is dependent on the relative priority of the data buses or halfwords containing the errors. It is possible to encounter a single and double error simultaneously and latch the single error; the double error flag sets unconditionally. Therefore, if the

double error flag sets, the syndrome bits must be checked to determine if a single or double error was latched. If the single error flag sets and no double error, the error is a single error.

MODE 1

The first error occurring after a master clear or error clear has its error information latched. The information is correct regardless of subsequent errors. If a double error follows a single error without an error clear, the double error information is lost.

MODE 2

Mode 2 operation is identical to mode 1 except an attempt is made to latch the error information for the first double error encountered regardless if a single error has previously been latched. The double error flag sets unconditionally when a double error is encountered. Other aspects of mode are less certain and conditions which may result are listed.

- If simultaneous errors, mode 2 is the same as mode 1. If the double error flag is set, the syndrome bits must be checked to determine if a single or double error was recorded.
- If the SECDED unit encounters one or more single errors, and the double error flag is absent, the error information is that of the first single error. All information is correct as in mode 1.
- If the SECDED unit encounters a double error followed by other double or single errors, the error information is that of the first double error and the syndrome bits must be checked.
- If the SECDED unit encounters a single error and less than eight minor cycles later a double error is encountered, address bits 37 through 54 for either the single or double error may be latched. Bits 55 and 56 are undeterminable and the remaining error information is that of the double error.
- If the SECDED unit encounters a single error and more than eight minor cycles later a double error is encountered, the double error information is correct. However, the MCU cannot distinguish this case from the previous case.

- If the SECDED unit encounters a double error and one or more minor cycles later encounters a single or double error, the first double error information is latched.

DOUBLE ERROR LOG (MODE 2A)

After a master clear or error clear, the MCU creates a single error using the maintenance function to toggle a check bit. This is not cleared, thereby blocking detection of all subsequent single errors. Therefore, when the MCU detects the double error flag, the error log information is correct for that double error.

SECDED FAULTS

Executing an 06 instruction with bits 9 through 15 of the R designator selected, causes SECDED faults to be generated on all read buses. This allows checking of the read SECDED hardware and also the fault recording hardware for type and address of the fault.

BLOCK WRITE ENABLES

The MCU can enable block write enable if a SECDED error occurs. Depending on the mode, there are two options:

- Mode 1
  The write enable is blocked when SECDED receives its first single or double error.
- Mode 2
  The write enable is blocked when SECDED receives its first double error.

SECDED USAGE

The SECDED design best suited for a system is based on the error rate of the memory.

Mode 1 is a good SECDED latch design for a memory with a low error rate. All error log information is correct, but mode 1 does not latch a double error if it follows a single error within the cycle time of the MCU.

Mode 2 is a good SECDED latch design for a memory with a high error rate. All single errors latched are correct, and all double errors following a single error by more than eight minor cycles are correct. A double error occurring before a single error is also latched correctly.

Mode 2A is a double error logging system used when single errors are to be ignored. This mode misses the double error only if there is a simultaneous single error with a higher latching priority.

ASSOCIATIVE UNIT

The associative unit contains the 16 associative address registers and corresponding control circuits. When the CPU is in job mode, all addresses sent from the stream unit and scalar processor units are virtual addresses. The associative unit compares a virtual address with the virtual address identifier of the associative registers. If a match occurs and one of the four keys compares with the locks of the associative address registers, the virtual address control circuits convert the virtual address into the corresponding absolute memory address from which the reference is made. If a match is not found in the associative address registers, the virtual address control circuits read additional associative words from the space table. The space table is a restricted area of the central memory.

SEARCHING THE PAGE TABLES

The 16 associative registers (ARs), labeled 00 through 15, are loaded from absolute addresses 4000_{16} through $43C0_{16}$ by a load AR (0D) instruction. They can also be stored into the same absolute addresses by a store AR (0C) instruction.

The associative words in the ARs are moved dynamically using the following scheme. Whenever a virtual address is presented for association and a hit is made, the content of the AR containing the hit is moved to AR00. Simultaneously, the content of each AR, from AR00 to (but not including) the hit AR, is moved down one AR (for example, 00 to 01, 01 to 02, 02 to 03, and so on). Thus, the associative words in AR00 through AR15 are in descending order of most recent use. If the end-of-table (END) is contained in the ARs and no hit is made, the contents of the ARs remain unchanged and access interrupt is taken, unless the request is for a read-ahead sword of instructions negated by the branch. Whenever an address is presented, no hit is made, and no END is contained in the ARs, a search through the space table is begun using a ripple method. Each AR from AR00 through 14 is moved down one AR and AR16 is placed in a buffer register. A null is entered into AR00 and then AR00 through AR15 are stored in memory locations 4000_{16} through $43C0_{16}$. The content of the space table is rippled through the ARs. The first associative word of the space table is read and examined; its spot in storage is filled by the old content of the buffer register. If the first word read from the space table is not a hit, the second word is read, is replaced in storage by the first word read, and so on, until a hit is made or an end-of-table is reached.

If during the search a hit is made, the content of the hit address is temporarily stored in the buffer register and is replaced in memory by the associative word which precedes it in the space table. The contents of locations 4000_{16} through $43C0_{16}$ are loaded into the ARs, and the content of the buffer register (content of the hit address) is transferred to AR00. Entries in the space table beyond the hit address are not modified.

If an end-of-table is read before a hit is made, the entire space table, including the sword containing the END, is pushed down by one word position, and a NULL is placed in AR00. However, if the unsuccessful search was initiated by a memory reference in job mode, the NULL may be pushed out of AR00 before the exchange to monitor mode is performed. This unsuccessful search condition and the cause bits are sent to the main control and an access interrupt results.

If a NULL exists in the ARs and no hit is made in the ARs, the space table is not pushed down. A read and compare takes place until a hit is made and the NULL replaces that word in the space table.

If a hit is not made in the ARs and a NULL is encountered in the space table, the operation changes from a ripple to a read only (no push down); if no hit is found, the NULL remains in AR00, as before, If a hit is made deeper in the space table, the NULL replaces it. Thus, only one NULL need exist at any given time in the page table.

If the monitor sets up the page table with one NULL, and it does not add or delete a NULL, the END remains at a fixed address for any given number of associative words in the page table.

At the termination of an unsuccessful space table search, there is a NULL in AR00 if the unsuccessful search was initiated by an OF (load keys, translate address) instruction.

Figure 3-2 is an example of a page table search. The content of the ARs and the contiguous entries in the space table are depicted as P1, P2, and so on, NULL, and END, where P1 represents the associative word for page 1, NULL is a NULL associative word, and END is an end-of-table entry.

The example shows seven consecutive virtual address page references and the resulting page table transfers. Assume that there are 21 associative words in the page table (16 in the associative registers and 5 in the space table) and that no lockout bits are set; the last entry is an end-of-table.

1. The first reference is to page 3. P3 is in AR02 and is moved to AR00; the content of AR00 through AR01 is moved down one word. The space table is not altered.

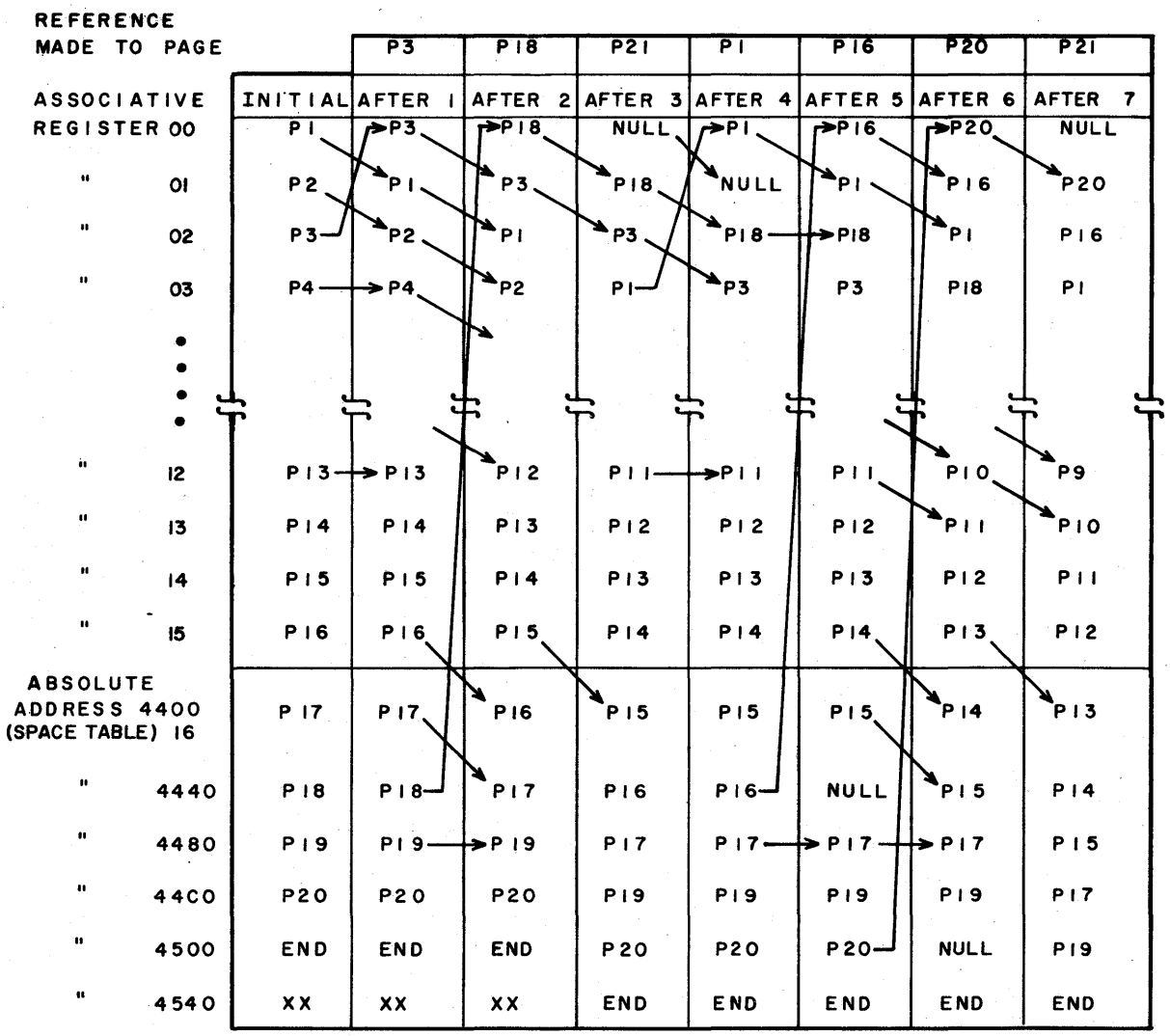


Figure 3-2. Page Table Search Examples

2. The next reference is to page 18. No hit is made in the ARs so the ARs are pushed down one and the content of AR15 (P16) is pushed down into the space table. P17 is read and replaced with P16. Since P17 is not a hit, it is swapped with the next entry in the space table, P18. P18 caused a hit so it is replaced by P17 and moved to AR00.

3. The third reference is to P21, which is not in the page table. The result is that the entire page table, including the END, is examined and pushed down, AR00 is set to a NULL, and an access interrupt is generated.

4. Assume that the access interrupt is properly handled by the monitor program and the page table is not altered. The next storage reference in job mode is to P1. Since P1 is in AR03 when the reference is made, it is moved to AR00, and AR01 through AR02 are moved down one word.

5. The fifth reference is to P16 which is now the second entry of the space table. This time there is a NULL in the ARs. The NULL is moved to AR00 and AR00 is moved down one word. P14 is not moved into the space table and the space table is not pushed down. A read and compare takes place until the hit is found; the NULL then replaces the selected associative word in the space table.

6. The next reference is to P20. Since there is no hit or NULL in the ARs, the page table is pushed down until the NULL is encountered. Push down ceases and read and compare takes place until P20 is read, causing a hit. P20 is moved to AR00 and is replaced by a NULL.

7. The last reference is to P21 which is not in the page table. The page table is pushed down until the NULL is encountered. Push down and searching cease when the END is read.

   AR00 is set to a NULL and an access interrupt is generated.

For page table restrictions and requirements, refer to table 5-5.

MULTIPLE-MATCH FAULT

In the central computer, any given combination of lock and virtual page identifier in an associated word may occur in only one associative word in the page table. Whenever a violation of the rule is detected, a multiple-match fault occurs and the CPU is stopped. When two keys are identical, their lockout bits must be the same. Otherwise, a reference to the differing lockout bits generates a multiple-match fault, resulting in an undefined condition. There are two types of multiple-match faults.

- One virtual address, lock, and key matches more than one register in the associative registers.
- A virtual address makes a successful match with the associative registers, and at least one additional match combination exists, but the reference is locked out by the key lockout bits.

INSTRUCTION ISSUE

The instruction issue unit issues instructions at a rate of one instruction per minor cycle, unless it is blocked by instruction or memory conflicts. The unit must resolve three conflicts:

Source operand conflict	An instruction requiring the result of a previous instruction as an input operand must wait until the operand is available.
Output operand conflict	An instruction result, destined for the same register file location as a previously issued but slower instruction, must wait until the previous instruction stores its result into the register file.
Register file write conflict	An instruction result, arriving at the register file at the same minor cycle as the result of a previously issued but slower instruction, cannot issue.

To resolve these conflicts, 16 result address registers (RARs) hold the register file addresses for the output operands of previously issued instructions. Before an instruction is issued, its source operand addresses are simultaneously checked against all 16 RARs (source operand conflict) and its output operand address is checked against the operand result position timing chain (output operand and register file write conflicts) for possible conflicts. If a conflict exists, the issue is blocked until the conflict is resolved.

Any vector or string instruction modifying the register file (such as an index update, field length, or an operand result) blocks further issues in the scalar unit.

The instruction issue unit allows parallel operation of scalar and vector/string instructions provided there are no register file reference conflicts and no central memory references made by the scalar instruction. However, the scalar processor can make memory references to load the instruction stack in parallel with vector operations.

This parallel load operation requires two separate program address counters: one for vector/string instructions and one for scalar instructions. On interrupt, these counters are stored in the invisible package along with the operation code and G-bits designator of the vector in process. The content of the scalar unit's current instruction register is also stored in the invisible package. This allows for program restart following an interrupt.

The parallel operation of the scalar and vector processors is controlled by the instruction issue unit as follows: when the instruction control unit in the scalar processor decodes a vector instruction and the vector unit is not busy, the scalar unit supplies the vector unit

with the decoded instruction with all the descriptors. As soon as the vector unit is finished with the register file, the scalar unit is free to continue with the next instruction in the instruction control unit while the vector unit completes the vector instruction.

Table 3-2 indicates which instructions are executed in the scalar processor and which in the vector processor. Table 3-3 lists the instructions executed in the vector processor and indicates whether parallel execution is possible.

TABLE 3-2. SCALAR/VECTOR PROCESSOR INSTRUCTION RESPONSIBILITY

		First Digit of Instruction Code															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	S	S	S	S	S	S	S	S	V	V	V	S	V	V	V	V
	1	I	S	S	S	S	S	S	S	V	V	V	S	V	V	V	V
	2	I	S	S	S	S	S	S	S	V	V	V	S	V	I	V	V
	3	S	S	S	V	I	S	S	S	V	V	I	S	V	I	V	V
Second	4	V	V	S	S	S	S	S	S	V	V	V	S	V	V	V	V
Digit	5	S	V	S	S	S	S	S	S	V	V	V	S	V	V	V	V
of	6	S	V	S	S	S	I	S	S	V	V	V	S	V	V	V	V
Instruction	7	I	V	S	V	I	I	S	S	V	V	I	V	V	V	V	V
Code	8	V	V	V	S	S	S	S	S	V	V	V	V	V	V	V	V
	9	V	V	V	V	S	S	S	S	V	V	V	V	V	V	V	V
	A	V	V	S	V	I	S	I	S	I	V	I	V	V	V	V	V
	B	I	V	S	V	S	S	S	S	V	V	V	V	V	V	V	V
	C	S	V	S	S	S	S	S	S	V	V	V	V	I	V	V	V
	D	S	V	S	S	S	S	S	V	I	I	I	V	S	V	V	V
	E	V	V	S	S	S	S	S	S	I	I	I	S	S	V	V	V
	F	S	V	S	S	S	S	S	S	V	I	V	S	V	V	V	V

S Executed within the scalar processor. (Note that data flag information is passed to the data flag register in the vector processor for appropriate instructions.)

V The scalar processor initiates the vector processor to execute portions (or all) of the instructions.

I Illegal instruction (processed by scalar in monitor mode, by vector in job mode).

TABLE 3-3. CENTRAL COMPUTER PARALLEL OPERATIONS

Instruction	Parallel Operation
14	yes
15	yes
16	yes
17	yes
18	yes
19	no, updates index
1A	yes
1B	yes
1C	yes
1D	yes
1E	no, stores count in R.F. at exit
1F	no, stores count in R.F. at exit
28	no, updates index at exit
29	no, updates index at exit
7D	no, always reads/writes R.F.
8X	yes, if no broadcast A+B
9X	yes, if no broadcast A+B
AX	yes, after first pass is completed
B7	no, reads C from R.F.
B8	yes
B9	no, always reads/writes R.F.
BA	no, reads B from R.F.
BB	yes, if no broadcast A+B (C is stored in R.F. at start of instruction.)
BC	no, stores C in R.F. on exit
BD	yes, if no broadcast A+B (C is stored in R.F. at start of instruction.)
C0-C3	no, reg C stored in R.F. on exit
C4-C7	yes, if no broadcast A+B
C8-CB	no, reads B from R.F. after interrupt restart

TABLE 3-3. CENTRAL COMPUTER PARALLEL OPERATIONS (Contd)

Instruction	Parallel Operation
CF	no, stores C in R.F. at exit
D0	yes, if no broadcast A+B
D1	yes
D4	yes, if no broadcast A+B
D5	yes
D6	no, updates index
D7	no, updates index
D8-D9	no, stores B in R.F.
DA-DD	no, result stored in R.F.
DE	no, reads A from R.F. after interrupt restart
DF	no, broadcasts B from R.F. after interrupt restart
E0, E1	yes
E2	no, reads C from R.F. during third pass
E3	yes
E4, E5	yes
E6	no, reads C from R.F. during third pass
E7	yes
E8, E9, EA	yes
EB	no, writes mark pointer in R.F. at end of instruction
EC, ED	yes
EE, EF	no, updates index
F0, F7	yes
F8, F9	no, updates index
FA	yes
FB, FC	yes
FD, FE, FF	no, updates index

REGISTER FILE

The register file of the central computer contains 256 64-bit words. This register file is capable of accomplishing two read operations and one write operation every minor cycle. The register file operations can be exchanged at the rate of two registers in and two registers out every minor cycle.

A scalar result written into the register file can be used by subsequent scalar instructions before the result is available in the register file. An exclusive OR of the read and write addresses is performed, thereby setting all bits of a register if equality exists. The scalar result bypass (shortstop) of the register file may occur at the same time the result is written into the file, if equality exists.

BRANCH/INSTRUCTION STACK

The instruction stack implemented in the central computer accommodates up to eight swords (512 bits per sword), six of which need not be adjacent. To sustain the instruction issue rate, a two-sword lookahead is done by reading the two swords following the sword being executed. Issue of instructions is not blocked if the swords following lookahead are not in the stack.

An address is maintained for each of the eight swords, allowing out-of-the-stack jumps to be taken without voiding the stack. For example, it is possible to call a subroutine of up to three swords (48 instructions of 32 bits each) several times from a three-sword instruction stream and never jump out of the stack.

LOAD/STORE UNIT

The load/store unit (L/S) executes the 12, 13, 32, 5E, 5F, 7E, and 7F instructions. Six address registers in the load/store unit enable requests to be stacked and executed in the proper order. The 12, 5E, and 7E instructions require one register and can be executed (with no memory conflicts) at a rate of one load per minor cycle. The 5F and 7F instructions require two address registers and can be executed at one store per two minor cycles. The 13 and 32 instructions require two address registers which are busy for 17 minor cycles after selection.

The L/S is capable of streaming L/S instructions (other than 13 and 32) at one minor cycle per load and two minor cycles per store assuming no memory busy, access interrupt, or register file write bus busy conflicts exist. For example, a stream of n loads executes in n+14 minor cycles from the issue of the first load until the operand from the last load available in the register file. A stream of n stores executes in $2n + \left\lfloor \frac{n}{3} \right\rfloor$ minor cycles from issue of the first store until issue of the last store.

SCALAR FLOATING-POINT

The central computer has an arithmetic unit dedicated to scalar operations. This unit is divided into five separate functional elements:

- Add/subtract pipe
- Multiply pipe
- Logical pipe
- Single cycle pipe
- Divide/square root/convert pipe

All elements of the arithmetic unit are separately and independently controlled allowing concurrent operation. However, only one operand pair is issued to the arithmetic unit, each minor cycle becoming the limiting factor in determining the result rate from concurrent operations.

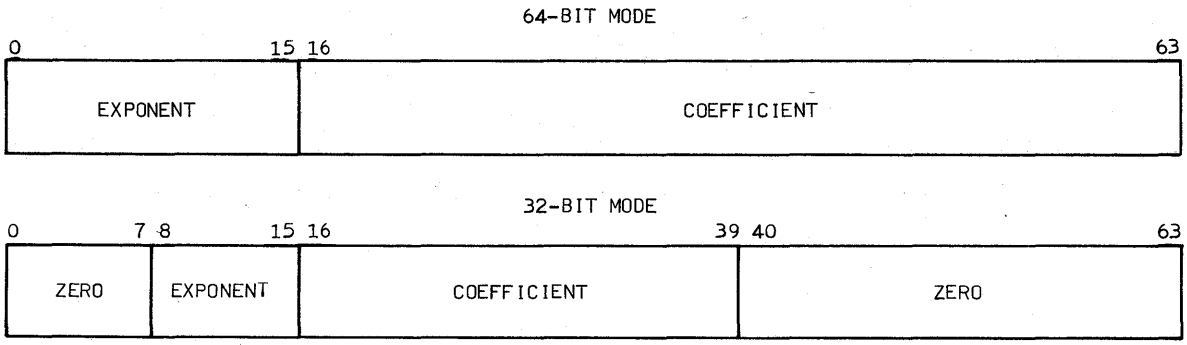
The first four functional elements are segmented pipeline units that accept a new pair of operands every minor cycle and produce a 64- or 32-bit result. The divide/square root/convert element is not segmented and accepts operands only at completion of the previous operation.

SCALAR FLOATING-POINT UNIT CONTROL INTERFACE

There are three input and two output trunks to the scalar floating-point unit. All input operands are 64- or 32-bit floating-point quantities, except where otherwise specified. If an indefinite or machine-zero floating-point operand is received, the coefficient is set with zeros.

A INPUT TRUNK

This 64-bit trunk receives data bits from register location R in the following format:



B INPUT TRUNK

The B input trunk, identical to the A trunk, receives data from register S.

CONTROL TRUNK

The control trunk carries the signals that control the scalar floating-point unit. It is composed of the following signals.

CONTROL ADDRESS

The control address bits select the set of internal control signals for the floating-point instruction being executed. A set of unique codes exists for each instruction (refer to table 3-4). Using the input data to the floating-point unit as a reference, these control bits must arrive at the floating-point logic 1.5 cycles before the data and must be valid for 20 nanoseconds.

MODE CONTROLS

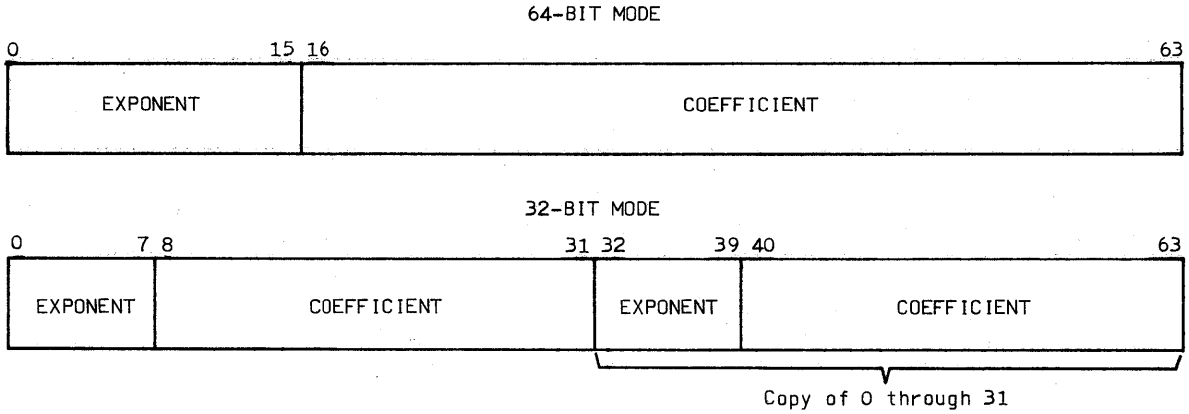
The mode controls are Mode 64 In, Mode 64 Out, G-bit, and Divide. The Mode 64 and G-bit signals must lead the input data by one minor cycle and the Divide signal must lead by 1.5 minor cycles.

ISSUE CONTROLS

The issue controls are S-shortstop, R-shortstop, S-clockgate, R-clockgate, S-shortstop Enable, R-shortstop Enable, and Go. All these controls must be valid one minor cycle before the data. Shortstop is the process by which a result from any arithmetic unit may be returned directly to either input of any arithmetic unit. The shortstop enable signals enable the setting or clearing of the shortstop control flip-flops. The clockgate signals cause data to be clocked into the floating-point input registers. The Go signal allows processing of operands in the input registers.

OUTPUT TRUNK

The output trunk is 64 bits and transmits output data to the stream unit. The formats for the output trunk are as follows. Data remains on this trunk for 20 nanoseconds.



OUTPUT CONTROL TRUNK

The output control trunk transmits control or fault bits associated with results generated by the scalar floating-point unit. These signals come up with data and are held up for 20 nanoseconds. The following signals are transmitted on the output trunk.

Signal	Meaning of a 1 on Signal Line
Branch Condition Met	The operands meet the compare condition. This line is zero when a compare is not being done.
Exit Condition Met	The operands do not meet the compare condition. This line is zero when a compare is not being done.
Divide Timing Pulse	Divide operands follow this timing pulse by 14 cycles.
Divide Unit Busy	The divide unit cannot accept new operands during the time this signal is 1.
Data Flags 39, 41, 42, 43, 45, 46	Refer to appendix D.
Data Flag 58	Logical OR of all the scalar floating-point data flags.

Data flags 39, 41, 42, 43, 45, 46, and 58 are held up for 40 nanoseconds for transmission to (transistor current switch) hardware.

TABLE 3-4. INSTRUCTION CODES

Instruction	M64 In	M64 Out	Control Address	G-Bits	Divide	A Trunk	B Trunk	Output Control
10	1	1	01		1	0	R	DT, DB, DFLG39, 58
11	1	1	02		1	0	R	DT, DB
20	0	0	10		0	R	S	EXCM, BRCM, DFLG46, 58
21	0	0	11		0	R	S	EXCM, BRCM, DFLG46, 58
22	0	0	12		0	R	S	EXCM, BRCM, DFLG46, 58
23	0	0	13		0	R	S	EXCM, BRCM, DFLG46, 58
24	1	0	14		0	R	S	EXCM, BRCM, DFLG46, 58
25	1	0	15		0	R	S	EXCM, BRCM, DFLG46, 58
26	1	0	16		0	R	S	EXCM, BRCM, DFLG46, 58
27	1	0	17		0	R	S	EXCM, BRCM, DFLG46, 58
2A	1	1	18		0	I	R	
2B	1	1	19		0	I	R	
2C	1	1	1A		0	R	S	
2D	1	1	1B		0	R	S	
2E	1	1	1C		0	R	S	
2F	1	1	1D	G2, G3	0	O	T	
30	1	1	1E		0	R	S	
31	1	1	1F		0	R	+1	

TABLE 3-4. INSTRUCTION CODES (Contd)

Instruc-tion	M64 In	M64 Out	Control Address	G-Bits	Div-ide	A Trunk	B Trunk	Output Control
34	1	1	20		0	R	S	
35	1	1	21		0	R	-1	
36	1	1	22		0	CIAR	+20	
38	1	1	23		0	R	T	
3C	0	0	24		0	R	S	
3D	1	1	25		0	R	S	
3E	1	1	26		0	R	I	
3F	1	1	27		0	R	I	
40	0	0	28		0	R	S	DFLG42, 43, 46, 58
41	0	0	29		0	R	S	DFLG42, 43, 46, 58
42	0	0	2A		0	R	S	DFLG42, 43, 46, 58
44	0	0	2B		0	R	S	DFLG42, 43, 46, 58
45	0	0	2C		0	R	S	DFLG42, 43, 46, 58
46	0	0	2D		0	R	S	DFLG42, 43, 46, 58
48	0	0	2E		0	R	S	DFLG42, 43, 46, 58
49	0	0	2F		0	R	S	DFLG42, 43, 46, 58
4B	0	0	30		0	R	S	DFLG42, 43, 46, 58
4C	0	0	31		1	R	S	DFLG41, 42, 43, 46, 5
4D	0	0	32		0	I	0	
4E	0	0	33		0	R	I	
4F	0	0	34		1	R	S	DFLG41, 42, 43, 46, 4
50	0	0	35		0	0	R	DFLG46, 58
51	0	0	36		0	0	R	DFLG46, 58
52	0	0	37		0	0	R	DFLG46, 58

TABLE 3-4. INSTRUCTION CODES (Contd)

Instruction	M64 In	M64 Out	Control Address	G-Bits	Divide	A Trunk	B Trunk	Output Control
53	0	0	38		1	0	R	DF LG43, 45, 46, 58
54	0	0	39		0	S	R	DF LG42, 43, 46, 58
55	0	0	3A		0	S	R	DF LG42, 46, 58
58	0	0	3B		0	R	0	
59	0	0	3C		0	0	R	DF LG42, 43, 46, 58
5A	0	0	3D		0	0	R	
5B	0	0	3E		0	R	S	
5C	0	0	3F		0	0	R	DF LG43, 46, 58
5D	0	1	40		0	0	R	DF LG43, 46, 58
60	1	1	41		0	R	S	DF LG42, 43, 46, 58
61	1	1	42		0	R	S	DF LG42, 43, 46, 58
62	1	1	43		0	R	S	DF LG42, 43, 46, 58
63	1	1	44		0	R	S	
64	1	1	45		0	R	S	DF LG42, 43, 46, 58
65	1	1	46		0	R	S	DF LG42, 43, 46, 58
66	1	1	47		0	R	S	DF LG42, 43, 46, 58
67	1	1	48		0	R	S	
68	1	1	49		0	R	S	DF LG42, 43, 46, 58
69	1	1	4A		0	R	S	DF LG42, 43, 46, 58
6B	1	1	4B		0	R	S	DF LG42, 43, 46, 58
6C	1	1	4C		1	R	S	DF LG41, 42, 43, 56, 58
6D(1)†	1	1	4D		0	R	S	
6D(2)†	1	1	4E		0	T	0	
6E	1	1	4F		0	R	S	

TABLE 3-4. INSTRUCTION CODES (Contd)

Instruction	M64 In	M64 Out	Control Address	G-Bits	Divide	A Trunk	B Trunk	Output Control
6F	1	1	50		1	R	S	DFLG41, 42, 43, 46, 58
70	1	1	51		0	0	R	DFLG54, 58
71	1	1	52		0	0	R	DFLG64, 58
72	1	1	53		0	0	R	DFLG64, 58
73	1	1	54		1	0	R	DFLG43, 45, 46, 58
74	1	1	55		0	S	R	
75	1	1	56		0	S	R	
76	1	1	57		0	0	R	
77	1	0	58		0	0	R	
78	1	0	59		0	R	0	
79	1	1	5A		0	0	R	
7A	1	1	5B		0	R	0	
7B	1	1	5C		0	R	S	
7C	1	1	5D		0	R	0	
B0, G1=0	1	1	60	G1, 2, 3, 4	0	A	X	
B0, G1=1	1	1	70	G1, 2, 3, 4	0	A	X	
B1, G1=0	1	1	61	G1, 2, 3, 4	0	A	X	
B1, G1=1	1	1	71	G1, 2, 3, 4	0	A	X	
B2, G1=0	1	1	62	G1, 2, 3, 4	0	A	X	
B2, G1=1	1	1	72	G1, 2, 3, 4	0	A	X	
B3, G1=0	1	1	63	G1, 2, 3, 4	0	A	X	
B3, G1=1	1	1	73	G1, 2, 3, 4	0	A	X	
B4, G1=0	1	1	64	G1, 2, 3, 4	0	A	X	
B4, G1=1	1	1	74	G1, 2, 3, 4	0	A	X	

TABLE 3-4. INSTRUCTION CODES (Contd)

Instruction	M64 In	M64 Out	Control Address	G-Bits	Divide	A Trunk	B Trunk	Output Control
B5,G1=0	1	1	65	G1, 2, 3, 4	0	A	X	
B5,G1=1	1	1	75	G1, 2, 3, 4	0	A	X	
BE	1	1	76		0	0	I	
BF	1	1	77		0	I	R	
CD	0	0	78		0	0	I	
CE	0	0	79		0	I	R	

† The 6D instruction requires three references to the register file; this takes two minor cycles. The (1) is the first and the (2) is the second.

ABSOLUTE BOUNDS ADDRESS

The absolute bounds address mechanism notifies the MCU of a memory reference (read or write) to a specified block of memory. The block of memory is specified by an upper bounds sword address and a lower bounds sword address. The addresses are absolute physical sword addresses and are transmitted from the MCU on channel BTA5 (refer to section 4). The bounds addresses are not included in the block of memory.

There are two classes referenced: read and/or write requests and CPU and/or I/O requests. Bounds checking is disabled if either (or both) classes have neither possibility selected (channel BTA6, bits 0 through 5).

The checker can selectively test various classes of requests for inbounds conditions. Any combination of classes may be selected (channel BTA6, bits 0 through 5).

If the CPU is stopped by a bounds hit, the hit is cleared by the clear fault signal from the MCU before the CPU restarts. The CPU restarts by setting bit 3 of MCU output channel BTA1. Bit 3 of BTA1, if set, causes the CPU to execute the next instruction in sequence.

A bounds hit (a selected memory reference inside bounds) is sent to the MCU on bit 3 of channel ATB8. To identify a second bounds hit, the MCU must clear the first bounds hit signal via the clear fault signal (bit 7, channel BTA1).

When a bounds hit occurs, the sword address of the request is saved in the bounds hit register until a master clear or fault clear occurs.

The bounds limits and the bounds hit address refer to physical addresses independent of memory degradation modes. (The bounds test is applied to the address after any degradation mode manipulation is applied.)

SCALAR MICROCODE MEMORIES

The scalar microcode consists of five memories: PM00, PM01, HM00, DM00, and GM00. Each memory operates independently during CPU instruction execution and are addressed simultaneously during writing or sweeping operations. The MCU loads the microcode memories via a second block transfer channel.

SCALAR MICROCODE MEMORIES (PM00, PM01)

Scalar microcode (SMIC) is composed of two memories: PM00 and PM01. Both memories operate simultaneously with a cycle time of 20 nanoseconds, and each memory contains 256 120-bit words.

SMIC memory is a read-only memory; writing into SMIC is reserved for loading systems or diagnostic microcode programs. SMIC provides the starting address for SMIC, FMIC, DMIC, and AMIC during the load operation.

The central computer uses microcode memories to start all instructions.

SMIC OPERATION

When the instruction stack has an instruction ready for execution, the function (F) code is sent to the PM00 address register. If the issue unit is ready to execute an instruction, the SMIC output is switched to PM00 and the execution is started.

If the instruction has one cycle of issue, SMIC output remains switched to PM00 and the next instruction begins execution (assuming the instruction stack has the next instruction available).

If the instruction has multicycles of issue, SMIC output is switched to PM01 where the remaining cycles of that instruction are executed. When the remaining cycles are completed, SMIC output switches back to PM00 and the next instruction begins execution (assuming the instruction stack has the next instruction available).

If the instruction has variable cycles of issue (for example, vector processor instructions, some of which execute in the associative unit, and so on), SMIC output is switched to PM01 and the remaining cycles of SMIC control are executed. When PM01 has completed its functions, it waits for the conditions indicating the end of the operation and switches to PM00 to execute the next instruction.

SMIC controls the flow of data from the instruction word to the functional unit. For example, SMIC:

- Selects designators to and from their points of use (register file read address, register file write address, address adder inputs, and so on)
- Selects register file data to functional areas, such as scalar pipeline and address adder

SMIC also controls the operations performed by other functional units. For example, SMIC:

- Provides starting addresses for scalar floating-point microcodes, FMIC and DMIC
- Informs load/store unit which operation to perform

SMIC ADDRESS CONTROL

PM00 addresses are controlled by the instruction stack.

PM01 addresses are controlled by SMIC bits. The next address to be used can be:

- The next sequential address (via incrementer)
- The address contained in the M01 field
- An address made from the M01 field (most significant four bits) and an index based on sense condition status (least significant four bits)

SMIC PARITY

SMIC has five parity bits forming odd parity.

ASSOCIATIVE MICRODE MEMORY (HM00)

The associative microcode (AMIC) is a 256-word by 96-bit memory with a 20-nanosecond cycle time.

AMIC OPERATION

AMIC is active during the following associative operations:

- Space table search
- Load associative registers (OD instruction)
- Store associative registers (OC instruction)

The AMIC memory is initialized into an idle loop and waits for a load, store, space table search, or an exchange operation. The memory supplies control to the associative registers (ARs), branches on conditions from ARs, and returns to the idle loop upon completion of an operation.

AMIC ADDRESS CONTROL

AMIC bits control HM00 addresses. The next address to be used is one of the following:

- The starting address
- The address from the AD1 field
- The address from the space table mode address register

AMIC PARITY

AMIC has four parity bits forming odd parity.

FLOATING-POINT AND DIVIDE MICROCODE MEMORIES (DM00, GM00)

The floating-point and divide microcodes (FMIC and DMIC) control scalar floating-point pipeline segment operations and iterative operations such as divide, square root, and BCD/binary conversion.

The floating point microcode memory (DM00) and the divide microcode memory (GM00) contain 256 48-bit memory words each. Both memories are read only memories. Writing is reserved for loading systems or diagnostic microprograms.

The main functions of FMIC and DMIC are as follows:

FMIC	Selects data paths for operand processing
	Provides constants for exponent correction and coefficient shifting
	Enables hardware checks for end case conditions such as machine zero operands, overflow conditions, and so on
DMIC	Selects data paths for operand processing
	Preconditions logic properly for divide, square root, and BCD/binary convert algorithms
	Notifies machine when unit is processing operands and the results are available

FMIC OPERATION

FMIC receives its address from an 8-bit field (M02) in scalar microcode memory. If an instruction requires the scalar floating-point unit, the issue unit causes one 48-bit microinstruction word to be read from FMIC. This word controls the segments of the floating-point pipeline as the operands are processed. Floating-point operations of differing lengths may cause conflicts within the pipeline segments. It is the issue unit's responsibility to prevent this by issuing instructions at the proper rate.

DMIC OPERATION

If the floating-point operation is divide, square root, or BCD/binary conversion, DMIC microcode memory is used with FMIC to control the iterative segments of the pipeline that perform these operations. The 8-bit field (M02) sent from SMIC to the floating-point unit is used for the starting address. Each iterative operation controlled by DMIC requires the execution of several microinstructions. There is a field in each DMIC microinstruction (GMA) that points to the next microinstruction. This linkage continues until the last microinstruction required is completed. The GMA field of the last microinstruction points to location 0 of DMIC, a one instruction idle loop. DMIC remains in this idle loop until the next divide, square root, or BCD/binary conversion instruction is received, at which time a new starting address is received from SMIC.

VECTOR PROCESSOR

The vector processor contains a stream unit, two floating-point pipelines, and a string unit.

The vector processor instruction and streaming control is contained in the stream unit. The stream unit receives decoded instructions from the scalar processor and executes these instructions while the scalar processor is free to execute scalar instructions not in conflict with the vector operation.

The stream unit manages the data streams between central memory and the vector pipelines.

Pipe 1 (VF1) is used for vector add/subtract and multiply operations. Pipe 2 (VF2) is used for vector add/subtract, multiply, and divide/square root operations. For vector addition, subtraction, and multiplication, the computer contains four 32-bit or two 64-bit pipelines, while for divide and square root, only two 32-bit or one 64-bit pipeline exists.

Each pipeline is segmented allowing a portion of the total operation on the two operands to be done in the first segment. The result is moved to the next segment of the operation, allowing a new set of operands to be moved into the first segment. For example, if an instruction takes six segments or cycles to complete, it takes six cycles for the first result, but each additional result is available every cycle thereafter to the end of the vector operation. Therefore, the upper limit on vector performance becomes the time increment of each segment, while the lower limit is the sum of time increments of all segments.

The maximum operational speed of the pipeline during streaming operations is:

Operation	32-bit Result	64-bit Result
Add/Subtract	100×10^6/seconds	50×10^6/seconds
Multiply	100×10^6/seconds	25×10^6/seconds
Divide/Square root	50×10^6/seconds	12.5×10^6/seconds

The vector processor contains a string unit to process strings of decimal and binary numbers and perform all of the nonregister bit logical and character operations. The results formed in a vector operation are selectable for storage by means of a program-specified bit string called a control vector. The string unit is the processing unit for control vectors during streaming operations. It contains a facility for BCD and binary arithmetic, zoned BCD processing, address arithmetic, and boolean operations.

VECTOR STREAM

The vector stream unit provides basic control for the central computer. Figure 3-3 is a basic block diagram of stream. The stream unit performs the following functions:

- Initiates all storage reference requests for vector instructions and operands.
- Translates vector instructions and transmits control signals to the arithmetic units.
- Provides addressing for all source operands and arithmetic results for vector instructions.
- Buffers and positions all operands and arithmetic results between central storage and the arithmetic units for vector instructions.
- Performs binary and decimal arithmetic operations on byte strings. It also performs other bit or byte string type operations such as edit, pack, unpack, compare, merge, modulo arithmetic, logical, and search with or without delimiter.

The vector stream unit interfaces with vector floating-point pipes 1 and 2 and the scalar processor. It also interfaces with the MCU interface for loading the microcode memory, maintenance, and fault monitoring.

Instruction control receives all vector instructions from the scalar processor via a 64-bit trunk from the scalar instruction issue unit.

ADDRESSING

Addressing is done in stages; that is, the addressing circuits break the address down into groups of bits and send these bits to the various areas of the CPU and memory where they control the selection or shifting of data.

The following are examples of address bits sent to the various areas of the CPU and memory. Address bits 0 through 15 are not used for addressing.

1. Bits 16 through 54 are the virtual sword address. Addressing sends these bits to the load/store unit for comparison with the page table.
2. Bits 55 and 56 select the quarter-sword. These bits are sent to the stream input and buffer control area for selection of operands. They are also sent to the instruction control area for selection of the control vector. Bits 55 and 56 also control the selection of the quarter-sword sent to memory from the write bus 1 output buffer area.

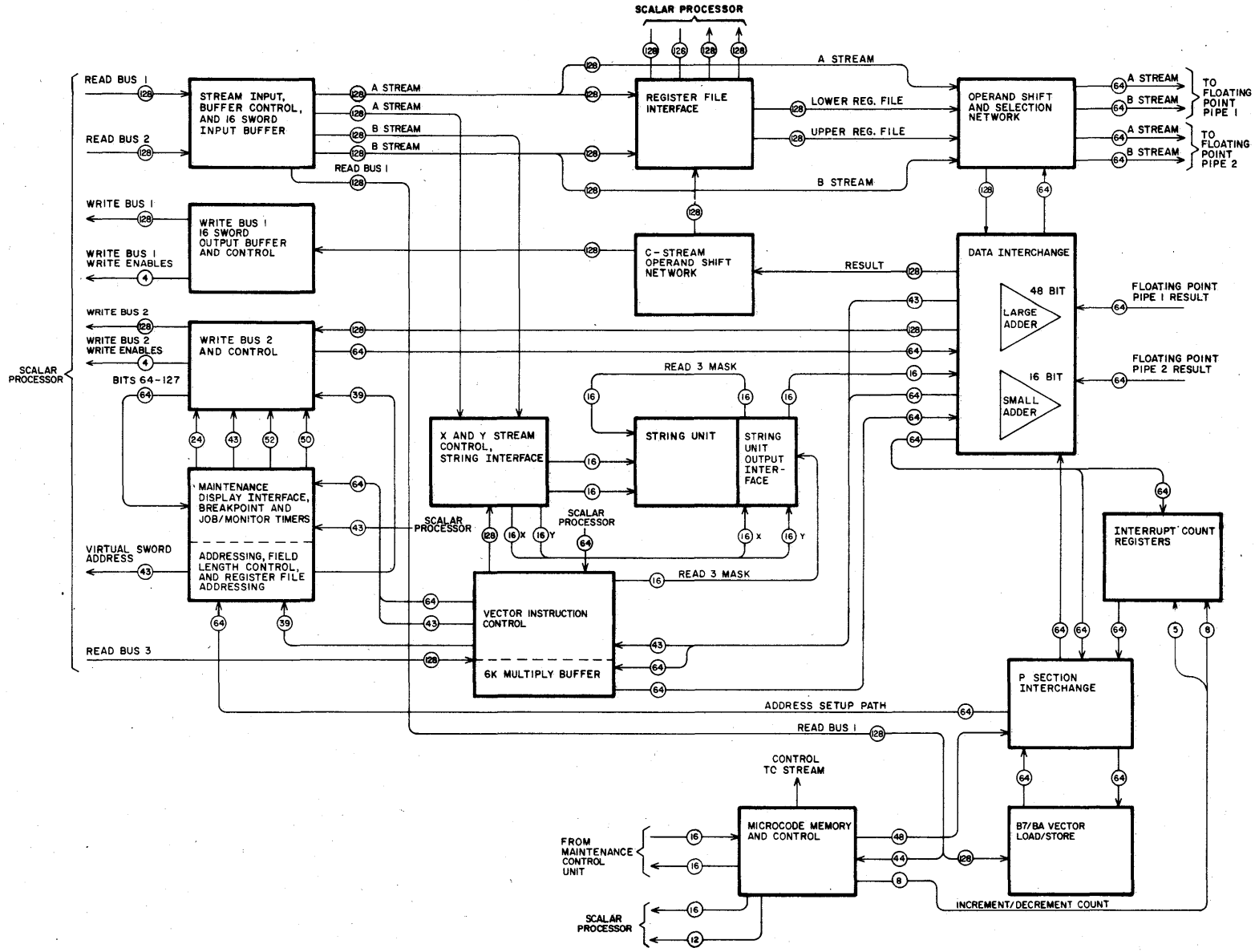


Figure 3-3. Basic Vector Stream Block Diagram

3. Bits 57 and 58 are sent to the operand shift network where they control the operand alignment shift from quarter-sword to word or half-words. Bits 57 and 58 also control the C stream operand shift network where they control the half-word/word to quarter-sword shift of the result.

4. Bits 57 through 63 control the selection and shifting of the A and B stream operands from the quarter-sword level to the byte and bit level in the X and Y stream control and string interface.

5. Bits 55 through 59 select read bus 3 from the sword level to the quarter-word level for the string output interface. This selection takes place in the instruction control area.

6. Bits 60 through 63 control the shifting in the string output interface. If the output goes back into the string unit (read 3 path), the shift is from quarter-word to byte or bit. If the output is to the data interchange, the shift is from bit or byte to quarter-word.

STREAM INPUT AND BUFFER CONTROL

This hardware consists of two 128-bit data paths between memory (read bus 1 and read bus 2) and the quarter-sword to item count addressing interfaces (X and Y stream control, string interface, register file, and operand shift network). This area handles quarter-swords and supplies them to the item count addressing interfaces at a usable rate. There is an 8K buffer (128 bits x 64) which is used to buffer the data reducing the data rate of a sword from memory in some operations and to align the two operand vectors for streaming in other operations.

OPERAND SHIFT AND SELECTION NETWORK

The operand shift network performs the final pairing of the operands before they enter the floating-point pipes. A and B stream buses (128 bits wide) enter the operand shift network from either the scalar processor or the stream input network. The operand shift network is capable of any shifting on 32-bit boundaries. After pairing, the operands are sent to the floating-point pipes via two 64-bit trunks to each pipe.

DATA INTERCHANGE

The data interchange performs the following functions:

- Receives and routes all data from the floating-point pipes, string unit, and the B7/BA unit.
- Routes all data going out write buses 1 and 2.
- Routes all data going to and from the vector setup adders.

C-STREAM OPERAND SHIFT NETWORK

The C-stream operand shift network realigns data to its proper position for writing into memory. The shift network is capable of any shifting on 32-bit boundaries.

WRITE BUS 1 OUTPUT BUFFER AND CONTROL

This hardware consists of one 128-bit data path between the item count to quarter-sword addressing and memory. This area handles quarter-swords (or 64- or 32-bit quantities aligned to the proper quarter-sword bits) and assembles them into swords for storage. There is an 8K buffer (128 bits x 64) which is used to buffer the data increasing the data rate of a sword to memory in some operations and to align the output vector for streaming in other operations.

WRITE BUS 2 AND CONTROL

Write bus 2 and control consists of a 128-bit wide data path into memory (write bus 2) and a large OR gate fed by all the registers which are saved in the invisible package.† These registers feed into their appropriate bit positions for storage in the invisible package. Also, a full 128-bit path from the data interchange is ORed in for storage of the register file in memory during an exchange operation.

X- AND Y-STREAM CONTROL AND STRING INPUT INTERFACE

This hardware consists of three 128-bit wide input data paths (read 1, read 2, and read 3) addressed to the quarter-sword level, and two 16-bit wide output data paths addressed to the bit level. For one type of operation, two inputs (read 1 and read 2)

†Section 5 of this manual contains a description of the invisible package.

supply operands to the string unit via the two output paths. For another operation, one input (read 3) supplies control vector bits via one of the 16-bit outputs to be used as output vector write enables.

STRING UNIT

The string unit (figure 3-4) processes strings of decimal and binary numbers. The X-stream, Y-stream, and data interchange areas of stream perform the bit boundary addressing required for the string instructions.

EDIT CONTROL

The edit control processes strings of numbers in packed binary coded decimal (BCD) format according to the control characters in the pattern field. Source characters are transferred to the result field with commas, decimal point, fill (check suppress) characters, and messages inserted as specified by the pattern field.

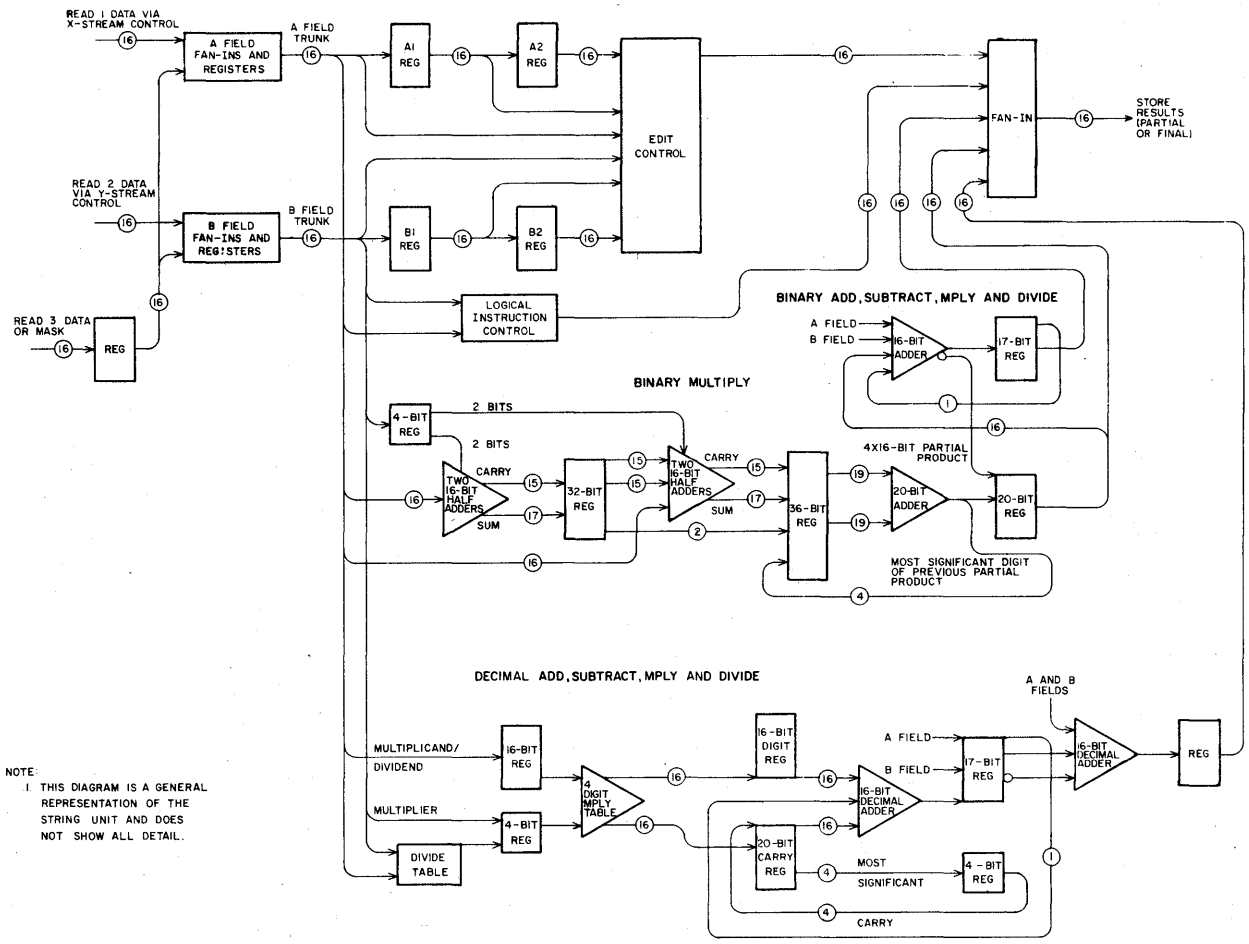
LOGICAL INSTRUCTION CONTROL

This control performs the exclusive OR, AND, inclusive OR, stroke, pierce, implication, inhibit, and equivalence operations on the input data fields.

BINARY ARITHMETIC CONTROL

This control performs the binary add, subtract, multiply, and divide operations on operand strings. The add, subtract, and divide operations are executed in one 16-bit adder. The multiply operation uses four consecutive 16-bit half adders and a 20-bit full adder to generate partial products. The partial product from one pass is added to the partial product of the previous pass in the 16-bit adder.

- Binary Add and Subtract
  The two operand fields are processed through the adder in 16-bit groups from right to left. A register overflow (carry) out of the adder from one 16-bit group is presented as a carry into the adder for the next 16-bit group.
- Binary Divide
  The hardware executes the divide instruction using an algorithm similar to the pencil and paper method of solution. The B field operand is subtracted from the left end of the A field operand



NOTE:
1. THIS DIAGRAM IS A GENERAL REPRESENTATION OF THE STRING UNIT AND DOES NOT SHOW ALL DETAIL.

Figure 3-4. String Block Diagram

generating one bit of quotient and a partial remainder that is stored. The hardware subtracts the two fields in 16-bit groups until the first pass is complete. On the second pass, the B field operand is subtracted from this partial remainder (shifted one bit) to generate a new partial remainder and the second quotient bit. The process continues until the division is complete. The hardware uses a nonrestoring type divide operation.

- Binary Multiply
  The binary multiply is accomplished similar to the pencil and paper method of solution. The A field operand is streamed through in 16-bit groups which are multiplied by the rightmost four bits of the B field operand. The second pass uses the next four bits of the multiplier with the partial results of this pass being added to the partial results of the previous pass. This process continues until the B field is exhausted.

  The multiplication by the 4-bit multiplier occurs in the four half adders, one multiplier bit per half adder. The partial sum and carry bits from the four half adders, together with the upper four carry bits from the previous 4- by 16-bit multiply, are combined in the 20-bit full adder. The lower 16 bits of the partial product are combined with the partial products of the previous passes in the 16-bit binary adder used for binary add, subtract, and divide.

  The binary multiply unit multiplies only positive operands. Negative operands are complemented at the inputs to the various adders. If a negative result is required, the final product is complemented in a separate pass.

DECIMAL ARITHMETIC CONTROL

This control performs the decimal add, subtract, multiply, and divide operations through the use of two 16-bit decimal adders, a divide table, and a 4-digit multiply table. The add and subtract operations are performed in the second adder which also combines the partial results of the successive passes on multiply and divide operations.

- Decimal Multiply
  The A field operand is divided into 4-digit groups which are multiplied by the rightmost digit of the multiplier on the first pass. The multiply lookup table generates a product digit and a carry digit for each digit

of the 4-digit group. The product and carry digits, together with the most significant carry digit from the previous 4-digit group, are combined in the first 4-digit decimal adder and are then stored.

The other multiplier digits are processed on the second and successive passes. The partial products of a pass are combined with the partial products of the previous passes in the second decimal adder.

- Decimal Divide

  The hardware executes the decimal divide instruction by examining the most significant divisor digit and the two most significant dividend digits. The divide table generates the largest quotient digit possible for this input combination. The divisor, divided into 4-digit groups, and the trial quotient digit are multiplied in the multiply table. This product is subtracted from the dividend yielding a partial remainder (similar to the pencil and paper method of solution). Since only one digit of the divisor is examined in determining the quotient, the remainder may be negative (as when 080 is divided by 19 for which a quotient of 8 is generated by the divide table). A negative partial remainder forces the hardware into a correction cycle which adds the divisor to the partial remainder and decreases the value of the trial quotient digit by one. The correction cycle is repeated until the partial remainder is positive.

  The second pass generates the second quotient digit using the divisor and the partial remainder from the first pass (plus the next dividend digit). Additional passes occur until all digit positions of the dividend are processed.

MISCELLANEOUS OPERATIONS

The string unit also performs move, compare, merge, pack, and unpack operations not specifically identified by controls in figure 3-4.

INTERRUPT COUNTERS

The interrupt counters function as follows:

- Hold addresses, delimiters, field lengths, which are necessary to restart vector-type instructions after an interrupt.

- Keeps track of pass counts and termination conditions for multipass instructions.

P SECTION INTERCHANGE

The P section interchange performs the following:

- Receives data from the data interchange, B7/BA unit, interrupt count registers, and microcode memory control registers.
- Routes data to the data interchange, B7/BA unit, and addressing.

B7/BA UNIT

This unit is used to process B7/BA instructions.

MICROCODE (VMIC)

The computer uses microcode (VMIC) to start up and shut down vector type operations. For most other operations microcode is not used. The MCU loads the microcode memory via a second block transfer channel. This channel between the MCU and the microcode is also used to read VMIC memory, VMIC status, and set conditions (switches) in VMIC.

VMIC memory is used as a read-only memory. Writing into VMIC memory is reserved exclusively for loading systems or diagnostic microcode programs.

VMIC memory is composed of two memories: memory 0 and memory 1. Each memory operates on a cycle time of 80 nanoseconds but offset by 40 nanoseconds. Memory 0 leads memory 1 by 40 nanoseconds. Every read from memory 0 is unconditionally followed by a read from memory 1 at the same address, even if the memory 0 word forced a branch.

Each of these memories has 1536 words. Memory 0 has 128 bits (0 through 127) per word and memory 1 has 96 bits (128 through 223) per word. The memory access time of each memory is about 65 nanoseconds.

VMIC OPERATION

When the scalar processor initiates an instruction requiring microcode control, the vector unit sends a function code† and go pulse to the microcode unit. The microcode

† Section 6 of this manual describes the instructions.

go pulse forces the F code into bits 3 through 10 of the microcode program address (P) register (bits 0 through 2 are forced to zero) and starts the memory control timing chain. The F code of the instruction thus forms the starting address of the microcode program for that instruction. An exception to the above startup process occurs if the interrupt flag is set when the microcode unit receives the microcode go pulse. In this case, only the timing chain starts, and the F code does not go to the microcode P register. The microcode P register was set previously with the P address contained in the invisible package.

This type of operation is used when the microcode program is restarted after an interrupt.

After the CPU starts the microcode program, the microcode unit takes control of the startup and termination of the instruction, and in the case of an interrupt, saves all the operands and parameters necessary to resume execution of the instruction after an interrupt. Once initiated, the microcode program continues to execute until the KIL bit is read in a microcode word or until the MCU stops execution by sending a KILL signal.

The microcode program performs the following operations in a typical instruction start-up.

1. Reads the addresses from the register file in the scalar processor according to the instruction designators.
2. Makes the necessary address modifications.
3. Transfers the addresses to the appropriate interrupt count registers.
4. Sets up the usage and mode of operation of the read and write buses to/from main memory.

After startup, the microcode program waits for the conditions indicating the end of the operation and terminates. The program also monitors the external or access interrupt conditions, and if an interrupt occurs during instruction execution, the program saves the information needed to restart the instruction at the point it was interrupted. The microcode program initiates the exchange to monitor mode, sets the interrupt flag, and terminates.

CONTROL LINES

The control lines from VMIC memory, defined for both memory 0 and memory 1, are transmitted in turn. The memory 0 bit is transmitted for 40 nanoseconds and the memory 1 bit is transmitted for the next 40 nanoseconds.

Those bits defined for only one memory are transmitted for the same relative 40 nanosecond period as the doubly defined bits. In some cases, the singly defined bits are transmitted for 80 nanoseconds.

When VMIC is not running, all control lines transmitted to the vector stream unit are forced to conditions not affecting operation. The MCU can disable these control lines with switch 5 (refer to section 4).

VMIC INTERRUPT

When the microcode program senses an interrupt condition, it continues execution until it comes to an appropriate point to stop and allows the interrupt to proceed. At that point, the microcode sets the interrupt flag, initiates the exchange to monitor mode, and stops. During the exchange, pertinent microcode control information is stored into word 3 of the invisible package. This information is used later to restart microcode execution at the point it was stopped.

When the microcode program is restarted, the initial address depends on the state of the interrupt flag as reloaded from the invisible package. If the interrupt flag from the invisible package is set, the P address contained in the invisible package is forced into the VMIC P register. The P address from the invisible package is one plus the address when the KIL bit terminated the microcode control to process the interrupt. If the interrupt flag from the invisible package is clear, the F code is forced into the VMIC P register.

VMIC PARITY

Each 224-bit microcode word has two parity bits forming odd parity: parity bit 0 (PB0) for memory 0 and parity bit 1 (PB1) for memory 1. Software generates the parity bits before loading the word into the microcode memory.

Each microcode memory has hardware which tests the parity as it reads each microcode instruction for execution. A parity fault in either memory stops microcode and CPU instruction execution. Instruction execution stops with the microcode memory P register containing the address 2 words beyond the word causing the parity fault. Bit 1 of MCU channel ATB8 indicates the occurrence of a VMIC memory parity fault stop.

Each VMIC memory also has a separate VMIC memory parity fault status bit available to the MCU via the display register (bits 6 and 7 of display register code 4). The clear faults signal sent from the MCU clears all three VMIC memory parity fault status bits.

There is no VMIC memory parity fault during loading or storing VMIC memory from the MCU.

VMIC PARITY FAULT ISOLATION

Two tools are available to isolate VMIC memory parity faults. The MCU is able to read the contents of VMIC memory and compare it with the data loaded into the VMIC memory. MCU is also able to read the VMIC memory P register. The P register points to the second instruction beyond the VMIC instruction containing the parity fault causing the stop.

CHECKPOINT

The checkpoint bit (CPT field in VMIC memory 1) is a maintenance aid used for microcode program debugging and oscilloscope triggering. During execution of a microcode word, the checkpoint flip-flop sets if the CPT microcode bit in that word is equal to 1. The checkpoint flip-flop is sensed and cleared by the MCU. The MCU senses the checkpoint flip-flop via bit 0 of microcode status word 1 and clears the checkpoint flip-flop via microcode switch bit 0.

MICROCODE WRITE LOCKOUT

A lock and key located on the same chassis as the microcode memory enables or disables the writing of data into microcode memory. If the key is in the disable position, the block transfer channel from the MCU acts as though it made a normal microcode load but no data is written into memory. This protects the microcode program from alteration once the program is loaded.

VECTOR FLOATING-POINT

Floating-point numbers in the computer are two lengths: 32 bits and 64 bits. The 32-bit format has an 8-bit exponent and a 24-bit coefficient (figure 3-5). The 64-bit format has a 16-bit exponent and a 48-bit coefficient. The leftmost bit of each exponent and coefficient is the sign bit. A detailed description of floating arithmetic is presented in the instruction specification.

Pipe 1 and pipe 2 work together to perform all vector arithmetic instructions except divide and square root. Pipe 2 alone performs divide and square root (figures 3-6 and 3-7). This organization of hardware allows optimum performance for both register and vector divide operations. For vector operations common to both pipe 1 and pipe 2,

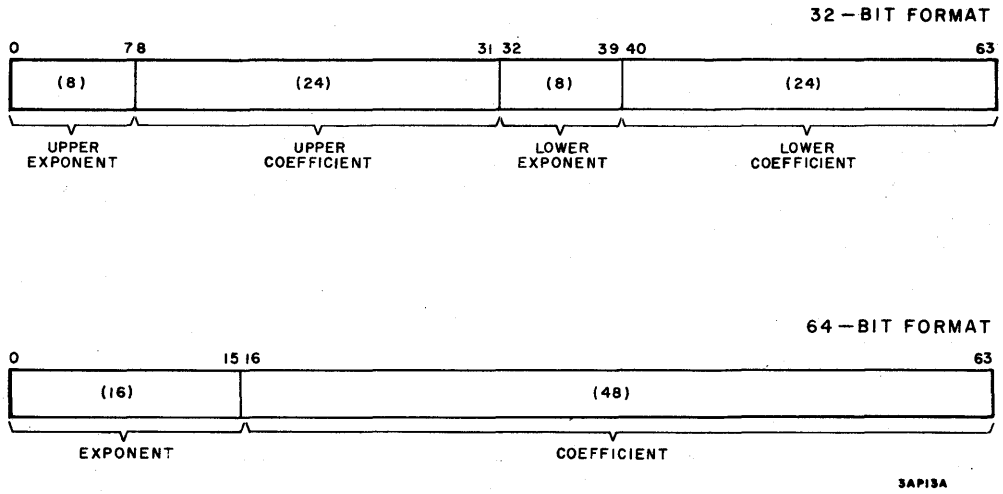


Figure 3-5. Operand Formats

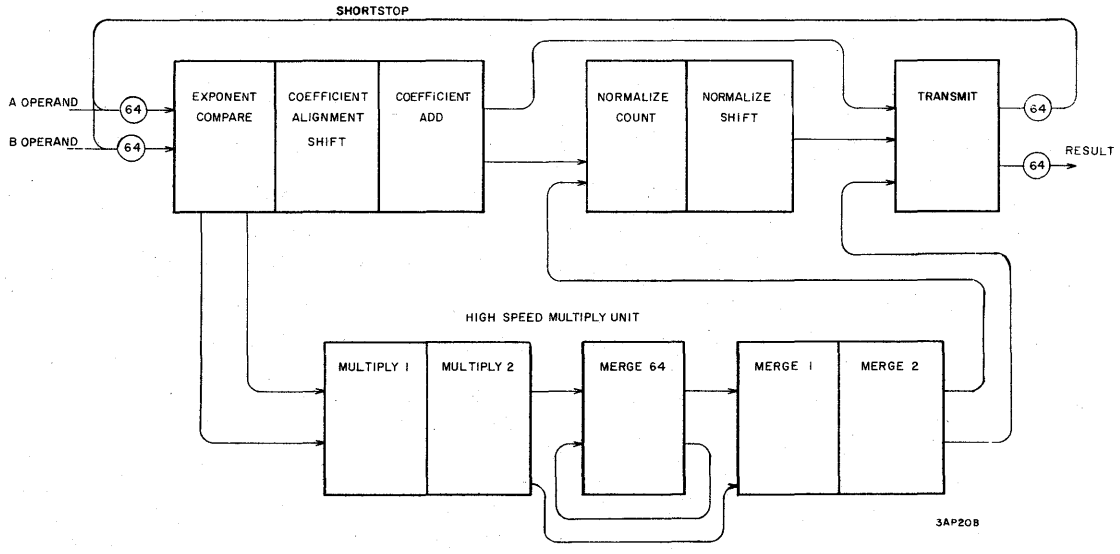


Figure 3-6. Floating-Point Pipe 1

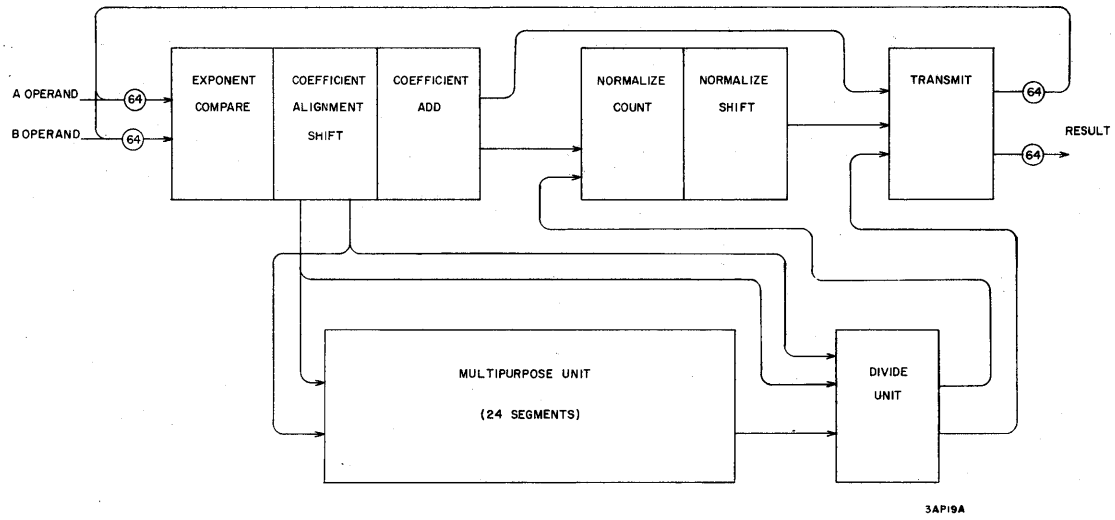


Figure 3-7. Floating-Point Pipe 2

the data is divided in half with every second pair of 64-bit operands going to pipe 2 (first pair, third pair, and so on) and every second pair (second pair, fourth pair, and so on) to pipe 1. In 32-bit mode, each pipe divides in half to become two 32-bit pipes. Therefore, two pair of operands go alternately to each pipe.

PIPE 1

Floating-point pipe 1 receives operands from the vector stream unit, performs the instructed operation, and returns the results to the vector stream unit. Pipe 1 performs arithmetic operations on operands in floating-point format and address operations on nonfloating-point numbers. Arithmetic operations include such operations as add, subtract, multiply, truncate, adjust exponent, contract, extend, and compare. Refer to figure 3-6 for the following description of some basic operations of pipe 1.

For addition and subtraction operations, the input exponents are compared in the exponent compare circuit. The difference in the two exponents is used as a shift count determining the amount the coefficient with the smaller exponent is right-shifted in the coefficient alignment section. The coefficients are added in the add section. If the operation being performed specifies normalization, the result of the add operation is fed to the normalize count. This circuit produces a shift count which controls the normalize shift network and modifies the result exponent. The transmit circuit returns the shifted result to the stream unit.

If normalization is not specified, the result of the add operation is the desired result and is transmitted to stream.

If the instruction is a multiply, the operands are multiplied in the high-speed multiply unit. The result of the multiply is either returned directly to the transmit section or to the normalize count logic for normalization. The normalize count functions only for the multiply significant instructions.

Any result from pipe 1 may be returned directly to either of the inputs of pipe 1 if the result is needed as an input operand. This process is called shortstopping.

PIPE 2

Floating-point pipe 2 (figure 3-7) receives operands from the vector stream unit, performs the instructed operation, and returns the results to the vector stream unit. Pipe 2 performs arithmetic operations on operands in floating-point format and address operations on nonfloating-point numbers. Arithmetic operations include such operations as add, subtract, multiply, divide, truncate, adjust exponent, contract, extend, and compare. Pipe 2 performs only two address type operations. These are the vector add and subtract address instructions (83 and 87 instructions). Pipe 1 and pipe 2 are similar except pipe 2 has a high-speed register divide unit (not used) and a multipurpose unit.

MULTIPURPOSE

The multipurpose unit performs the vector square root, vector divide, and vector multiply instructions. The multipurpose unit contains 24 segments. Each segment performs an add type operation. The segments are arranged in four groups of six segments per group. In 64-bit mode, the operands loop on each group, going through each group twice. In 32-bit mode, the operands proceed from segment to segment going through all of them only once. The multipurpose unit delivers its results to the normalize or transmit portions of pipe 2.

INPUT/OUTPUT CHANNELS

The central computer CPU contains 12 I/O channels. Channel 12 is reserved for the MCU. The MCU provides the interface to the operator for maintenance, system control, and monitoring. The MCU can disable any or all I/O channels from reading or writing into central memory. The peripheral station on a disabled channel can carry on all functions, except the transmission of data to/from central memory, with the I/O channel of the central computer. This feature is useful for maintaining the I/O channels and peripheral stations.

A typical I/O channel connects to a peripheral station. The peripheral station may connect to various peripheral devices or to another second-level peripheral station.

ASSEMBLY/DISASSEMBLY

Each I/O channel contains a 32-bit assembly/disassembly register and address register circuits. Addresses are sent to the channel from the peripheral station. In addition, the I/O channels share a high-density logic (HDL) storage unit. The HDL storage unit has a capability of 32 quarter-swords of data (128 bits each). The I/O buffer is used for assembly, disassembly, and buffer operations. An I/O channel is allocated a quarter, half, or whole sword in the I/O buffer. The amount of I/O buffer space that is allocated to an I/O channel is predetermined and may be altered only by specific contractual arrangement.

The allocation for each I/O channel is:

Channels 1 through 5	Four quarter-swords each
Channels 6 through 10	Two quarter-swords each
Channels 11 and 12	One quarter-sword each

The data trunk between the assembly/disassembly buffer (ADB) and central memory is 128 bits wide. The data trunk between the ADB and the channel assembly/disassembly registers is 32 bits wide. The data trunks between the peripheral stations and the assembly/disassembly registers are 16 bits wide.

I/O DATA

Figure 3-8 shows that in I/O write operations, each 32-bit half-word consists of two successive 16-bit transmissions from the peripheral station. The two 16-bit portions are assembled in the assembly/disassembly register for transmission to the I/O buffer.

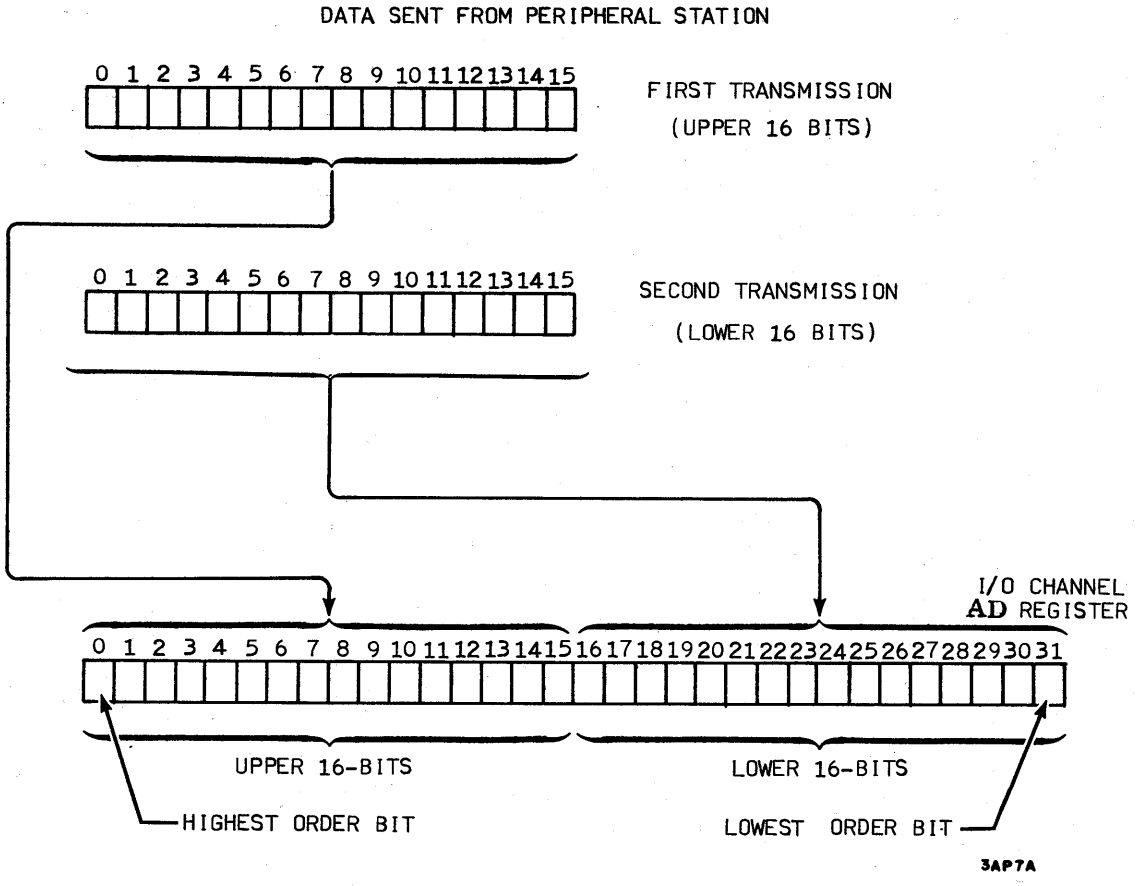


Figure 3-8. I/O Data Formats

I/O ADDRESSING

Figure 3-9 shows that the starting address for an I/O read or write operation is sent from the peripheral station as two 16-bit transmissions. The first 16 bits contain the upper or lower 1000K memory selection bit and the high-order 5 bits of the sword address. The second 16 bits contain the low-order 7 sword bits, the 5-bank selection bits†, the quarter-sword address, and the half-word address. The 12 sword address and 6 bank address bits are transmitted to the channel address register where they are incremented as sword boundaries and are crossed during central storage references. The quarter-sword address bits are sent to I/O control where they determine the

† The bank selection bits and the 1000K MCS selection bit are combined to form the 6-bit bank address as shown in figure 3-8.

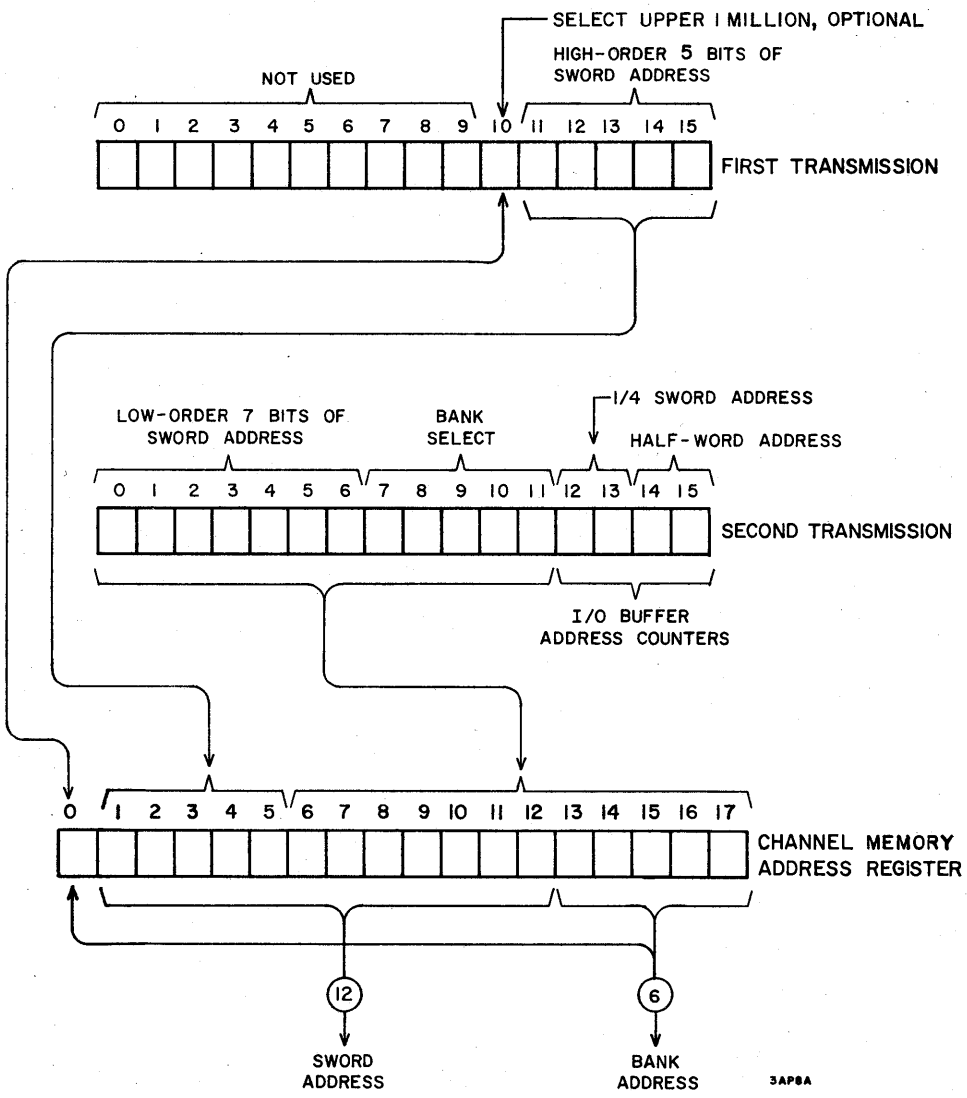


Figure 3-9. I/O Address Formats

quarter-sword that is loaded into or transmitted from the I/O buffer. The half-word address bits determine the 32-bit half-word that is loaded into or transmitted from the I/O buffer.

I/O CHANNEL PRIORITY

The I/O channels have priority over scalar operations. There are two modes of operation to determine priority, random mode, and stream or slot mode.

1. Random Mode
   When a channel needs a memory access, the request goes through a channel priority. Channel 1 has the highest priority and channel 12 the lowest priority. After channel priority is granted, the request goes through a system priority. Before the access is allowed, no read next instruction (RNI) request can be present. At this point, the memory busy is checked and if not busy, the access is granted. If the memory is busy, the requesting channel is limited to making requests on alternate I/O timing signals, thus allowing a lower priority channel to make a request while the higher priority channel waits for the memory to go not busy. The requesting channel requests memory on alternate access cycles (I/O timing signals) until the access is granted or until a higher priority channel makes an access request.

2. Stream or Slot Mode
   When a channel needs a memory access, the first check is to ensure that the bank requested is in the slot. When the selected memory bank is in the slot, the channel makes a request. This request goes through the channel priority where channel 1 has the highest priority and channel 12 has the lowest priority. After channel priority grants the request, the request goes through the system priority. Before the access is allowed, no RNI request can be present. At this point, the memory busy† is checked and if not busy, the memory request is granted. If the memory bank is busy, the channel waits until the next time the banks go not busy.

†Memory busy is unlikely in slot mode.

SYSTEM COMMUNICATIONS

The CPU (A) and first level stations (B) communicate by exchanging control and interrupt information. Signals sent from the CPU are called control from A (CFA) and signals sent to the CPU are B to A interrupts.

The control from A function codes are defined as follows:

Channel Flag	A channel flag is transmitted by the execution of an 08 instruction. Twelve channel flags are available in the computer, one for each I/O channel. The 08 instruction designates the I/O channel. Table 3-5 shows the assignment of the channel flags. A typical use of a channel flag is to indicate the CPU has a message concerning normal communication from system software placed in a prearranged area of storage.
External Flag	An external flag directs B to master clear and enter an autoload sequence. The external flag is initiated through the maintenance control unit.
Suspend	A suspend code directs B to cease transmission on the channel and go into a stand-by mode. Any master clear involving the scalar processor causes a suspend code. The suspend code is transmitted to all stations simultaneously.

TABLE 3-5. CHANNEL FLAG ASSIGNMENTS

Channel Flag†	Assignment
0	Not available
1	I/O channel 1
2	I/O channel 2
3	I/O channel 3
4	I/O channel 4
5	I/O channel 5
6	I/O channel 6
7	I/O channel 7
8	I/O channel 8
9	I/O channel 9
A	I/O channel 10
B	I/O channel 11
C	I/O channel 12
D	Not used
E	Not used
F	Not used

†Refer to the 08 instruction in section 6.

MAINTENANCE CONTROL UNIT 4

DESCRIPTION

The maintenance control unit (MCU) provides system autoload and system performance monitoring capabilities. The MCU also provides the capability of loading, controlling, and monitoring the central processor unit (CPU) diagnostics. The MCU consists of a control unit, line printer, disk drive, and 3000-channel interface. Connections from the MCU to the central computer are made through central computer I/O channel 12 and special internally connected interfaces (figure 4-1). The interfaces allow the MCU to monitor CPU status and gather performance statistics.

The primary purpose of the MCU is to support the reliability, availability, and maintainability of the central computer. Customer Engineering has priority use of the MCU for these purposes. The MCU provides operators with the means of autoloading the operating system, checking the CPU status, and gathering event counter data.

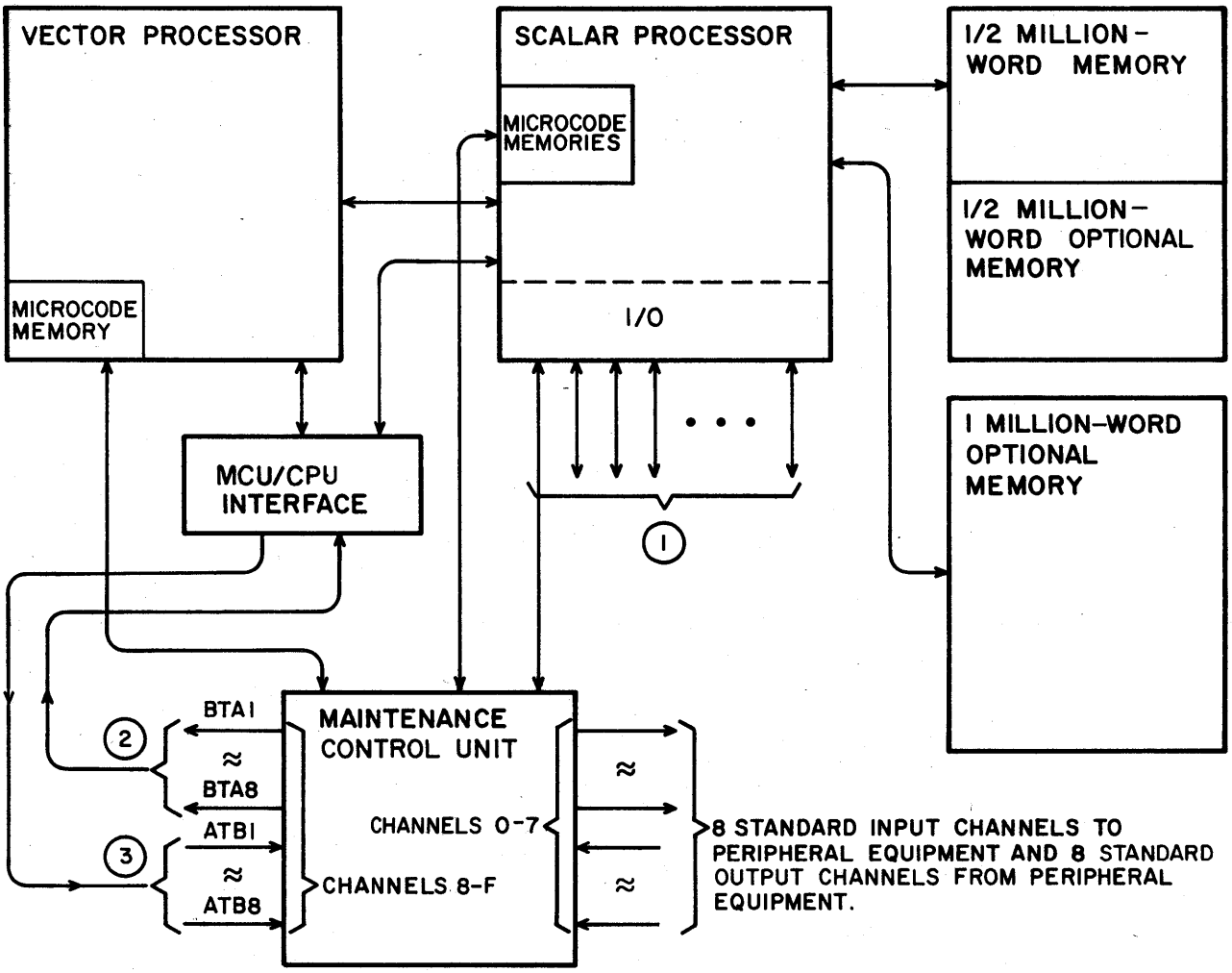
The MCU operates in off-line and on-line software modes.

- In an off-line mode, the MCU loads CPU diagnostic routines from the disk drive. The MCU then controls and monitors the diagnostic operations and furnishes the results of the operations to a display unit or a line printer.
- In an on-line mode, the MCU performs real-time monitoring of the CPU and displays its status.

MCU/CPU INTERFACE

The MCU connects to the CPU via eight, pulsed, normal channels in each direction (BTA and ATB). The channels that carry information from the CPU to the MCU (referred to as ATB) are numbered ATB1 through ATB8 and connect to MCU input channels 8 through F. The channels that carry information from the maintenance station to the CPU (referred to as BTA) are numbered BTA1 through BTA8 and connect to MCU output channels 8 through F.

Tables 4-1 through 4-8 list the ATB channel bits and their functions; tables 4-9 through 4-16 list the BTA channel bits and their functions. The connector for each channel is contained in the table title.



NOTES:

(1) CHANNEL 12 IS CONNECTED TO THE MCU

(2) 8 PULSED NORMAL OUTPUT CHANNELS NUMBERED BTA1-BTA8 CONNECT TO MCU OUTPUT CHANNELS 8-F

(3) 8 PULSED NORMAL INPUT CHANNELS NUMBERED ATB1-ATB8 CONNECT TO MCU INPUT CHANNELS 8-F

Figure 4-1. Maintenance Control Unit Interface

TABLE 4-1. CHANNEL ATB1 (CONNECTOR ATB12)

Bit No.	Function	
0	Bit 0	Current instruction address register
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
A	10	
B	11	
C	12	
D	13	
E	14	
F	15	

TABLE 4-2. CHANNEL ATB2 (CONNECTOR ATB12)

Bit No.	Function	
0	Bit 16	Current instruction address register
1	17	
2	18	
3	19	
4	20	
5	21	
6	22	
7	23	
8	24	
9	25	
A	26	
B	27	
C	28	
D	29	
E	30	
F	31	

TABLE 4-3. CHANNEL ATB3 (CONNECTOR ATB34)

Bit No.	Function	
0	Bit 32	Current instruction address register
1	33	
2	34	
3	35	
4	36	
5	37	
6	38	
7	39	
8	40	
9	41	
A	42	
B	43	
C	44	
D	45	
E	46	
F	47	

TABLE 4-4. CHANNEL ATB4 (CONNECTOR ATB34)

Bit No.	Function	
0	Bit 0	Display register; displays the register selected by bits C through F of channel BTA1 in the MCU.
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
A	10	
B	11	
C	12	
D	13	
E	14	
F	15	

TABLE 4-5. CHANNEL ATB5 (CONNECTOR ATB56)

Bit No.	Function	
0	Bit 16	Display register; displays the register selected by bits C through F of channel BTA1 in the MCU.
1	17	
2	18	
3	19	
4	20	
5	21	
6	22	
7	23	
8	24	
9	25	
A	26	
B	27	
C	28	
D	29	
E	30	
F	31	

TABLE 4-6. CHANNEL ATB6 (CONNECTOR ATB56)

Bit No.	Function	
0	Bit 32	Display register
1	33	
2	34	
3	35	
4	36	
5	37	
6	38	
7	39	
8	40	
9	41	
A	42	
B	43	
C	44	
D	45	
E	46	
F	47	

TABLE 4-7. CHANNEL ATB7 (CONNECTOR ATB78)

Bit No.	Function	
0	Bit 48	Display register
1	49	
2	50	
3	51	
4	52	
5	53	
6	54	
7	55	
8	56	
9	57	
A	58	
B	59	
C	60	
D	61	
E	62	
F	63	

TABLE 4-8. CHANNEL ATB8 (CONNECTOR ATB78)

Bit No.	Function
0	These lines indicate why the CPU has stopped. — Memory SECDED fault or instruction stack parity
1	MIC memory parity fault
2	Multiple match
3	Absolute sword bounds hit
4	Event stop
5	Single SECDED error
6	CPU clock; used for gating data back to the CPU. The MCU buffer controller cannot read this line.
7	Monitor mode
8	Temperature /dewpoint alarm
9	Not used
A	Section power fail
B	60 Hz input power fail, mainframe-memory MG
C	60 Hz input power fail, optional memory MG
D	Not used
E	CPU idle
F	CPU stopped

TABLE 4-9. CHANNEL BTA1 (CONNECTOR BTA12)

Bit No.	Function
0	MAC master clear; master clear to memory access control and central memory only. This includes the I/O channels. This signal must be set a minimum of 3 microseconds.
1	Stop; CPU stops before next instruction issues.
2†	Step; execute one instruction. Store the register file and the invisible package (job mode only); then stop. Faults must be cleared before the computer can be stepped.
3†	Run; start CPU from manual stop or fault stop. Faults must be cleared before computer can be started.
4†	Store associative registers and register file; associative registers are stored starting at absolute address 4000_{16}. The register file is stored starting at absolute address 0000_{16} in monitor mode and virtual address 0000_{16} in job mode. This operation destroys the contents of the associative registers. Therefore, after this operation, they must be reloaded by executing a load associative register command (BTA1 bit 5).
5†	Load associative registers and register file; associative registers are loaded starting from absolute address 4000_{16}. The register file is loaded starting at absolute address 0000_{16} in monitor mode and virtual address 0000_{16} in job mode.
6	CPU master clear; master clear to scalar unit, stream, associating, and floating-point only. MAC, I/O channels, and central memory are not included. This signal must be set a minimum of 3 microseconds.

†Computer must be stopped before executing these commands.

TABLE 4-9. CHANNEL BTA1 (CONNECTOR BTA12) (Contd)

Bit No.	Function
7	Clear fault conditions; this signal clears the following conditions and allows the computer to be restarted with a run signal (bit 3): • SECDED double error condition • MIC memory parity fault • Multiple match • Absolute sword bounds hit • Bounds hit address is released • Reference to illegal address in microcode • Instructional stack parity error
8	Clear, SECDED single error, SECDED fault address and syndrome bits.
9	MCU sync; this signal is used in the CPU to gate the CPU data back to the MCU. When reading the display registers, the MCU sync signal must be set after the read signal is set.
A	Not used.
B	Read; transfer selected register and current instruction address register into the display register.
C D E F	Display register selection; see display registers in this section.

TABLE 4-10. CHANNEL BTA2 (CONNECTOR BTA12)

Bit No.	Function	
0†	Static	Latch memory size code
1†	Static	Interrupt gate; when this signal is a 1, time interrupts and external interrupts will only be processed between instructions.
2†	Static	Memory time degrade code
3†		000 512K memory
4†		001 512K memory; force section 1 to section 0
		010 512K memory; force section 2 to section 0
		011 512K memory; force section 3 to section 0
		100 1 meg memory
		101 1 meg memory; force upper meg to lower meg
		110 2 meg memory
5†	Static	Select mainframe clock frequency††
6†		000 Nominal
7†		001 Increase clock frequency (1)
		010 Decrease clock frequency (1)
		011 Select variable frequency (adjustment on oscillator pak)
		100 Increase clock frequency (2)
		101 Increase clock frequency (3)
		110 Decrease clock frequency (2)
		111 Decrease clock frequency (3)
8†	Static	Delay trailing edge; delay the trailing edge of all of the clocks on the panel which are specified by bits B through F of channel BTA2. If bits 8 and 9 are set, only the odd or even clocks on a panel are moved depending on bit A.
9†	Static	Delay leading edge; delay the leading edge of all the clocks on the panel which are specified by bits B through F of channel BTA2. If bits 8 and 9 are set, only the odd or even clocks on a panel are moved, depending on bit A.

TABLE 4-10. CHANNEL BTA2 (CONNECTOR BTA12) (Contd)

Bit No.	Function	
A†	Static	0; move even clocks (see description for bit 8 or 9). 1; move odd clocks.
B (2^4) C (2^3) D (2^2) E (2^1) F (2^0)		Panel designator for clock margins; bit B is the leftmost bit of the designator. The designators are defined as follows:

Designator$_{16}$	Panel(s)	
00	All panels	
01	All floating point panels	
02	All MAC panels	
03	All stream and string panels	
04	Not used	
05	Not used	
06	Panel AA	Floating point
07	Panel AB	
08	Panel BA	
09	Panel BB	
0A	Panel CA	
0B	Panel CB	
0C	Panel DA	
0D	Panel DB	
0E	Panel EA	
0F	Panel EB	
10	Panel KA	Memory Access Control
11	Panel KB	
12	Not used	
13	Not used	
14	Not used	
15	Not used	
16	Panel PA	Stream, string
17	Panel PB	
18	Panel FA	
19	Panel FB	
1A	Panel GA	
1B	Panel GB	
1C	Panel HA	
1D	Panel HB	
1E	Panel JA	
1F	Panel JB	

†Computer must be stopped before executing these commands.

††If clock frequency codes 4 through 7 are used, code 3 is not available. Either code 0 through 3 or 0 through 2 and 4 through 7 are available.

TABLE 4-11. CHANNEL REGISTER FROM CHANNEL BTA3 (CONNECTOR BTA34)

Bit No.	Function
0	Not used
1	Send external flag on the channel specified by the channel select code in bits 4 through 8.†, ††
2	Set channel disable on the channel specified by the channel select code in bits 4 through 8.†, †††
3	Clear channel disable on the channel specified by the channel select code in bits 4 through 8.†, †††
4 5 6 7	Channel select code. A code of 1_{16} through C_{16} selects a channel (1_{10} through 12_{10}) for the operation specified in bits 1, 2, and 3.† Bit 7 of BTA3 is bit 3 of the select code.
8	Select all channels for the operation specified in bits 1, 2, and 3.†
9	Stop on SECDED single error detection.
A	Disable stop on SECDED double error detection.
B	Block external interrupt.
C	Disable error correction on all read buses.
D	Swap register file read on exchange.
E	Not used
F	Not used

† The channel select code in bits 4 through 8 must be set before any commands are sent on bits 1, 2, and 3, and it must remain set until after the command has dropped.

†† The external flag is transmitted to the device on the I/O channel corresponding to the code in bits 4 through 8. External flag instructs the device to autoload. Refer to Systems Communications, section 3, for a description of external flag.

††† If the disable line for a channel is set, no central memory references are allowed from that channel. Data transfers proceed in and out of the channel buffer in an end around type operation.

TABLE 4-12. CHANNEL REGISTER FROM CHANNEL BTA4 (CONNECTOR BTA34)

Bit No.	Function
0	Checkword bit 0 — Used for toggling I/O checkword bits 0 through 6.
1	1
2	2
3	3
4	4
5	5
6	6
7	Block write enable on SECDED error
8	Complement memory add bit 39
9	Select SECDED error log mode 2
A	Force register file store at bit address 20,000 on initial exchange
B	Force instruction stack parity
C	Enable I/O simulator
D	Initiate I/O simulator on channel flag
E	Not used
F	Not used

TABLE 4-13. CHANNEL BTA5 (CONNECTOR BTA56)

Bit No.	Function
0 1 2 3 4	Not used
5 6 7	Bounds limit load code 0 Null 1 Load bits (35-42) upper bounds 2 Load bits (51-58) upper bounds 3 Load bits (43-50) upper bounds 4 Null 5 Load bits (51-58) lower bounds 6 Load bits (35-42) lower bounds 7 Load bits (43-50) lower bounds
8 9 A B C D E F	Due to the operational characteristics of the maintenance interface, only one bit of the code can be changed at one time. Address bits must be loaded leaving the load code bits undisturbed. Address bits are transferred on the leading edge of a code change, the address bits must be set up before a code change occurs. Address bits are loaded as follows, starting and ending with a null code: Code 0 Null 1 Set up bits (35-42) upper bounds 3 Set up bits (43-50) upper bounds 2 Set up bits (51-48) upper bounds 6 Set up bits (35-42) lower bounds 7 Set up bits (43-50) lower bounds 5 Set up bits (51-58) lower bounds 4 Null Bound limits are absolute, physical halfword addresses. Bits (35-36) and (55-58) must be zero.

TABLE 4-14. CHANNEL BTA6 (CONNECTOR BTA56)

Bit No.	Function	
0	Check bounds on memory reads	If bits 0 and 1 or bits 2 and 3 are zero, no bounds hits can occur.
1	Check bounds on memory writes	
2	Check bounds on CPU references	
3	Check bounds on channel references	
4	Stop CPU on bounds hit	
5	Not used.	
6	Count A; monitoring counter A is enabled while this line is a 1 and held clear when this line is a 0. The proper counter specification and bits 8 through E of channel BTA6 must not be changed while this line is enabled.	
7	Count B; monitoring counter B is enabled while this line is a 1 and held clear when this line is a 0. The proper counter specification and bits 8 through E of channel BTA6 must not be changed while this line is enabled.	
8	Clear counter overflow bits only [see monitoring with counters (code 6) in this section]	
9	Stop CPU on Counter A increment	See monitoring with counters in this section.
A	Stop CPU on Counter B increment	
B	Enable carry into A1	
C	Enable carry into A2	
D	Enable carry into B1	
E	Enable carry into B2	
F	0; bits 0 through F of channel BTA7 are the count specification for counter A. 1; bits 0 through F of channel BTA7 are the count specification for counter B. This bit should be set to the proper counter before the count specification is set into channel BTA7.	

TABLE 4-15. CHANNEL BTA7 (CONNECTOR BTA78)

Bit No.	Function	
0	Event select for counters A1 and B1; see monitoring with counters in this section for codes.	
1		
2		
3		
4		
5	Event select for counters A2 and B2; see monitoring with counters in this section for codes.	
6		
7		
8		
9		
A	Not used	Event counter gates; see monitoring with counters in this section
B	Selected job gate	
C	Monitor mode gate	
D	Job mode gate	
E	Data flag 56 gate	
F	Data flag 57 gate	

TABLE 4-16. CHANNEL BTA8 (CONNECTOR BTA78)

Bit No.	Function
0 1 2 3 4 5 6 7	8-bit function select code. Bit 0 is the leftmost bit of the code. See event number 12_{16} in monitoring with counters in this section.
8 9 A B C D E F	8-bit mask. Bit 8 is the leftmost bit of the mask. See event number 12_{16} in monitoring with counters in this section.

MCU/MICROCODE MEMORY INTERFACE

A 16-bit channel, similar to an I/O channel, connects the MCU to the microcode memory, providing the MCU with the ability to load and store the microcode and load microcode diagnostic routines. The channel also provides control for running the routines under MCU control.

The MCU interface to this channel is connected to the MCU as a second coupler. The programming is the same as the other block transfer channel except different normal MCU channels are used for control.

The microcode memory interface to this channel is similar to an A-coupler with channel control and fan-ins and fan outs between the 16-bit data channel and the 224-bit microcode words.

The A-coupler does not use the following lines, normally found on a standard central computer channel.

- Parity error from A
  The A-coupler does not check parity on any function from the B-coupler and does not send a parity error on a microcode parity error. The MCU can check for a microcode parity error on the normal channel maintenance lines between the MCU and the CPU just as for any other CPU parity error.
- Illegal from A
- Interrupt from B
- System control

MICROCODE MEMORY CHANNEL PROGRAMMING

The B-coupler uses the MCU block transfer channel for the transfer of data and MCU normal channel 6 for function codes and coupler control. The following is a description of the setup and use of the channel.

B-COUPLER SETUP

The following steps are taken to set up the B-coupler:

1. Select the B-coupler by clearing bit 9 and setting bit A of channel 2.
2. Connect channel 6 to the B-coupler by setting bit 8 and clearing bit 9 of channel 5.

CHANNEL 6 FUNCTIONS

The following control bits are sent to the B-coupler via channel 6:

Bit	Definition
0	Initiate functions
1	Function bit 0 (2^2)
2	Function bit 1 (2^1)
3	Function bit 2 (2^0)
4	Interrupt (not used)
5	Clear fault
6-F	Not used

B-COUPLER TO A-COUPLER FUNCTION CODES

The functions normally found on a central computer channel are redefined for this interface. For all channel functions, the null function and the address that accompanies the channel function are ignored. Table 4-17 shows three bit function codes and their functions for control of microcode memory.

TABLE 4-17. B- AND A-COUPLER FUNCTION CODES

Bit 0	Bit 1	Bit 2	Function
0	0	0	Null - Automatically sent by the B interface as the second half of any other function.
0	0	1	Read Memory - Read a block of microcode memory from the current microcode P address.
0	1	0	Write Memory - Write a block of microcode memory from the current microcode P address.
0	1	1	Not normally used but will perform the same as a EOP in the P section.
1	0	0	Data - Automatically sent with the data during a write microcode memory operation.
1	0	1	Read Status - Read the current microcode status.
1	1	0	Write Switches - The switches provide control of microcode execution.
1	1	1	EOP - End of Operation clears the interface of all previous functions and also clears the counter that controls the data fan-in and fan-out to/from the channel.

MICROCODE SWITCHES

Microcode switches are 1-bit terms controlling the microcode memory. Each switch is one bit of the write switch control word. The 110_2 function code (write switch) causes the microcode memory to store the write switch control word in a register.

The B-coupler receives this data from the block transfer channel and sends it to the microcode control. Table 4-18 describes the microcode switch functions.

TABLE 4-18. MICROCODE SWITCH FUNCTIONS

Bit	Function
0	Go microcode; strobing this bit causes microcode to start execution at the current microcode P address.
1	Kill; setting this bit stops any microcode instructions executing at the time the bit is set. The instruction comes to a normal halt with P pointing to the next word to be executed. Execution can be resumed by setting bit 0.
2	Sense switch; any microcode program can sense the condition of this switch for program control (used mainly by diagnostics).
3	P to 0; strobing this bit forces the P register to 0. Kill should be set previously or in the same word to proceed to a normal halt.
4	Clear checkpoint; strobing this bit clears the checkpoint flip-flop.
5	Drop control-setting; this bit disables control of the CPU and the ICs from microcode preventing undefined CPU operation due to a microcode memory test.
6	Change status word 2 definition; bits 8 through F of status word 2 become bits 0 through 7 of an IC register (refer to table 4-19).
7	Enable control of the register logical pipe from microcode.
8	Function for scalar microcode not yet defined.
9	Sweep scalar microcode.
A	Write scalar microcode; must be set to write. Scalar microcode disables P-section write enables.
B	1; enables scalar microcode to sweep PM00. 0; enables scalar microcode to sweep PM01.
C, D, E, F	Functions for scalar microcode not yet defined.

The switch functions have the following uses.

1. Switch functions 0, 3, and 4 are one-shot functions having the required bit set in the even 16-bit word of a transfer and clear in the odd 16-bit word. For example, if the bit is set into both halves of a 32-bit transfer, the function is performed in that transfer but is ignored if sent in the next transfer.
2. Switch functions 0 and 3 are delayed by one cycle so other functions sent in the same data word have time to propagate. For example, kill and P to 0 together are legal as are sense switch and go microcode. Other combinations are also legal.
3. Switch functions 1, 2, 5, 6, 7, 9, A, and B are latching functions that are caught and held until another function is sent. However, a single function consists of two or more data transfers, each clearing and loading over previous data transfers so a switch that is meant to be valid during and after the function must be sent in both halves of a 32-bit data transfer. Any latching function that is supposed to remain valid through another send switch function must be sent again with that function present in both halves of the 32-bit data word.

STREAM MICROCODE STATUS

The input of status to the MCU can be a number of words; but all words after the first are word 2 of status. The input of status does not have an effect on microcode or microcode controls. Table 4-19 shows the status words loaded in the channel when the A interface receives a channel function code of 101_2.

MCU MONITORING

The MCU monitors the output of two display registers as its main monitoring of system activity. One display register contains the output of the current instruction address register (CIAR). The other display register contains the output of the register selected by the MCU. A 4-bit code sent from the MCU (channel BTA1, bits C through F) selects the appropriate display register. In addition to monitoring the display registers, the MCU can also monitor the microcode memory status and other CPU status.

TABLE 4-19. MICROCODE STATUS FUNCTIONS

Bits	Function
	WORD 1
0	Checkpoint; software uses this bit to indicate to the MCU that the microcode has reached a predefined status, found an error, or reached a predefined address for debugging.
1 2 3 4	Flags; the current state of flags 0, 1, 2, and 3.
5 6 7 8 9 A B C D E F	P; the current state of the P (microcode address) register.†
	WORD 2
0	Run; this bit is used to indicate the microcode is executing.
1 2 3 4	J1; the current state of the least significant 4 bits of the J1 register.
5 6 7 8 9 A B C D E F	J2; the current state of the J2 register (see bit 6 of the switch function control word).

†The contents of P do not indicate the address at which microcode has stopped until the second minor cycle after the run bit has gone to 0. Thus, it is necessary to read the status word twice: once to determine that microcode is not running, and once to read P.

DISPLAY REGISTER

The MCU sends a read signal to enable the CIAR and the selected register into the two 64-bit display registers. The read signal is defined as bit B on channel BTA1, and its leading edge simultaneously transfers both registers into the display registers. The MCU determines the register select code (table 4-20) before transmitting the read signal to the CPU. All unaccounted for bits coming into and going out of the display registers are undefined.

The MCU receives the CIAR on channels ATB1 through ATB3, and receives the selected register on channels ATB4 through ATB7.

The CIAR and the event counters may be read anytime. Other displays are examined only when the CPU is not running.

TABLE 4-20. DISPLAY REGISTER SELECT CODES

$Code_{16}$	Register(s)	Bits
0	Current instruction register	0-63
1	Data flag register	3-15, 19-31, 35-47, 51-58
2	Invisible package address (absolute sword address)	0-22
	Page zero address (absolute small page address)	38-48
3	External interrupt register	17-31
	Channel 1	17
	2	18
	3	19
	4	20
	5	21
	6	22
	7	23
	8	24
	9	25
	10	26
	11	27
	12	28
	Not used	29
	Not used	30
	Monitor interval timer	31

TABLE 4-20. DISPLAY REGISTER SELECT CODES (Contd)

$Code_{16}$	Register(s)	Bits
	Channel read active - write active	32-55
	Channel 1	32-33
	2	34-35
	3	36-37
	4	38-39
	5	40-41
	6	42-43
	7	44-45
	8	46-47
	9	48-49
	10	50-51
	11	52-53
	12	54-55
4	SECDED Fault Read Bus Code	0-2
	I/O Bus = Code 0	
	R1 Bus = Code 1	
	R2 Bus = Code 2	
	R3 Bus = Code 3	
	Scalar Bus = Code 4	
	RNS Bus = Code 5	
	Space Table Search in Process	3
	This line is to be used in conjunction with SECDED Error to determine if the error occurred while doing a Space Table Search.	
	Instruction Stack Parity Fault	4
	MIC Memory 0 Parity Fault	5
	MIC Memory 1 Parity Fault	6
	Scalar MIC Parity Fault	7
	Double SECDED Error. Syndrome Bits must be checked to determine if address and bus code are valid.	8
	Syndrome Bits	9-15
	Parity Fault on Auxiliary Board 0	16
	Parity Fault on Auxiliary Board 1	17
	Parity Fault on Auxiliary Board 2	18
	Parity Fault on Auxiliary Board 3	19
	Parity Fault on Auxiliary Board 4	20

TABLE 4-20. DISPLAY REGISTER SELECT CODES (Contd)

$Code_{16}$	Register(s)	Bits
	Parity Fault on Auxiliary Board 5	21
	Parity Fault on Auxiliary Board 6	22
	Parity Fault on Auxiliary Board 7	23
	Parity Fault on Auxiliary Board 8	24
	PM01 Enabled for Parity Checking	25
	Scalar Microcode Address - Bit 0	26
	Scalar Microcode Address - Bit 1	27
	Scalar Microcode Address - Bit 2	28
	Scalar Microcode Address - Bit 3	29
	Scalar Microcode Address - Bit 4	30
	Scalar Microcode Address - Bit 5	31
	Scalar Microcode Address - Bit 6	32
	Scalar Microcode Address - Bit 7	33
	NOTE: All Fault/Error conditions are cleared by the Clear Fault signal from the MCU except the SECDED Error and the Syndrome bits. These are cleared/released by the Clear Single Error signal from the MCU.	
	SECDED Fault Address (Absolute physical bit address, significant to the half-word level)	34-63
	The address of the first SECDED error is retained in this register.	
	The SECDED Fault Address is released by the Clear Single Error Condition Signal from the MCU.	
5	Bounds Hit Address (Absolute physical bit address, right justified)	0-31
	The address of the first bounds hit is retained in this register. The bounds hit address is released by the Clear Fault Condition signal from the MCU. The bounds checking is performed on half-word boundaries only.	
6	Counter A1	0-15
	Counter A2	16-31
	Counter B1	32-47
	Counter B2	48-63

TABLE 4-20. DISPLAY REGISTER SELECT CODES (Contd)

$Code_{16}$	Register(s)	Bits
	If bit 8 of channel BTA6 in the MCU is a 0, both counters will be cleared after the read signal is received and after both counters are transferred into the display register. If bit 8 is a 1, the counters will not be cleared. To ensure proper initialization of the counters, the count lines must be made zero prior to the new count selection.	

MONITORING COUNTERS

For monitoring purposes, the CPU has four 16-bit counters (figure 4-2). Each of these counters can be connected to an event line selected by a command from the MCU. Table 4-21 contains a list of events which can be counted and their corresponding select codes. There are two pairs of 16-bit counters: A1, A2 and B1, B2. The A and B counters are completely independent and cannot be tied together. However, they do share the same input event lines (figure 4-3). The counters are selected for display via the MCU display register. They can also be combined in various ways to form one or two 32-bit counters. This configuration is accomplished via the carry lines from the MCU. The counters are enabled by hardware and software gates selected with a mask from the MCU. The MCU has the option of stopping the CPU on a count condition by enabling the stop lines.

COUNT GAGES AND CPU LINES

The event counters are incremented when the selected event occurs, the count line is up, and one or more of the following gate-line conditions are satisfied.

1. The event counter enable bit is set in the invisible package of the job currently being executed and the selected job gate from the MCU is set. This allows counts to be made during selected jobs only.
2. The CPU is in job mode and the job mode gate from the MCU is set.
3. The CPU is in monitor mode and the monitor mode gate from the MCU is set.

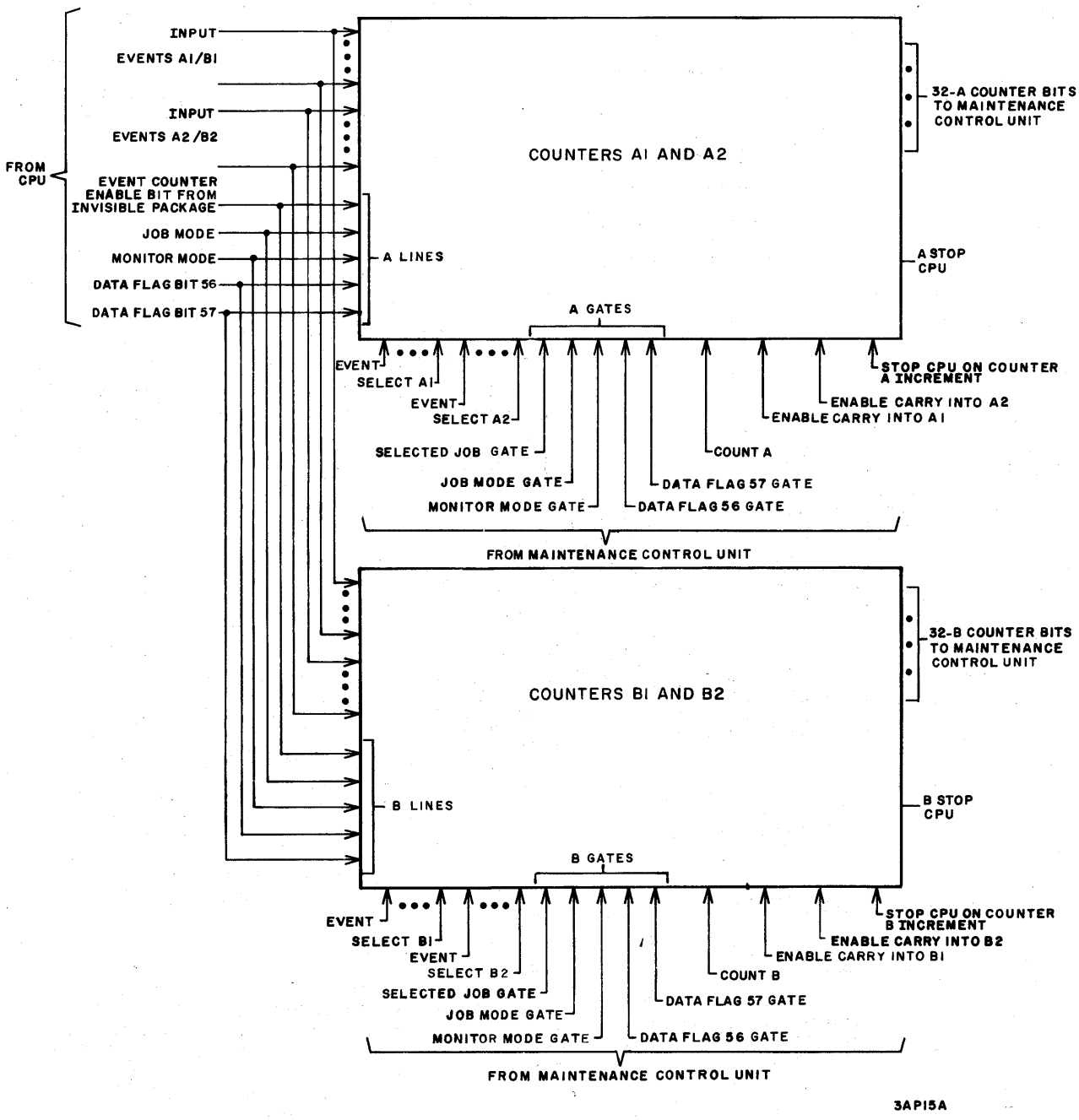


Figure 4-2. Block Diagram of Counter Logic Lines

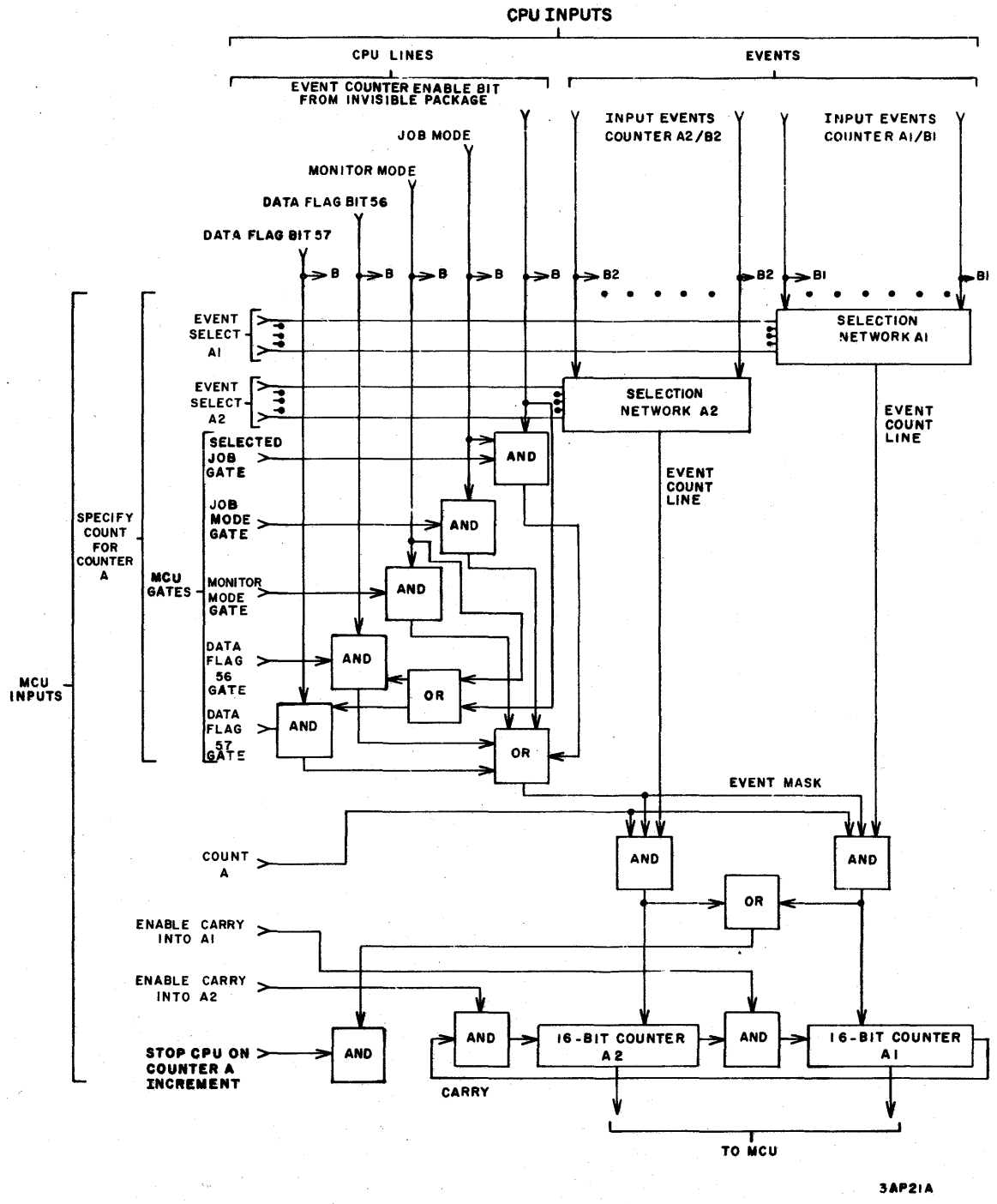


Figure 4-3. Block Diagram of Counter A

TABLE 4-21. COUNTER EVENTS

Codes$_{16}$		Events
Counter A1/B1†	Counter A2/B2	
04		Number of times microcode MON = 1 is selected
05		Number of space table searches
11		Number of minor cycles from selected instruction issue to next nonselected issue. The counter will begin counting when an instruction whose function code meets the conditions described in code 12 is loaded into IRO. It will stop counting when the next instruction not meeting the conditions is loaded into IRO.
12		Number of times a particular function code or particular category of function codes is executed. The count condition is determined by an 8-bit select code and an 8-bit mask sent to the CPU on channel BTA8. If the select code bits and the corresponding instruction function code bits are equal wherever there is a 1 in the mask, the counter is incremented. If the mask contains all zeros, all instructions are counted.
	12	Time - 1 MHz
13		Time between selecting microcode monitor field, MON=2 and selecting MON=3.
	13	Number of cycles where data is not available at the output of a functional unit (string or floating-point) once data has been requested for all input streams. This time does not include the time required for initial setup (preceding requests for memory) or shutdown (following the input of the last operands to a functional unit) of vector or string instructions. This count thus permits the programmer to analyze the amount of time required for startup memory accesses, pipeline/functional unit length, space table searches, and memory conflicts for a specific instruction.

† Due to differences in clock rates for the scalar unit and maintenance interface unit, counter-event codes 11 and 12 (counter A1/B1) may not be accurate or available.

4. Data flag bit 56 or 57 is set in the data flag register of the CPU, the data flag 56 or 57 gate from the MCU is set, and the CPU is in monitor mode.

5. Data flag bit 56 or 57 is set in the data flag register of the CPU, the data flag 56 or 57 gate from the MCU is set, and the event counter enable bit is set in the invisible package of the job currently being executed.

   There is one set of gate-line enable logic for counters A1 and A2 and one set for counters B1 and B2; therefore, the A counters may be enabled by different gates than the B counters.

The CPU lines are:

- Data flag bit 56
- Data flag bit 57
- Monitor mode
- Job mode
- Job enable of monitoring counters from invisible package.

The MCU gates are:

- Data flag 56
- Data flag 57
- Monitor mode
- All jobs mode
- Selected jobs mode

CARRY LINES

There is one enable carry line associated with each 16-bit counter. Enable carry line A1 enables the carry into counter A1 from counter A2. Enable carry line A2 enables the carry into counter A2 from counter A1. There are equivalent lines for the B counter. A zero on carry lines A1 and A2 allows the counters to operate as two 16-bit counters. Only half of the total number of events are available at the selection network for one counter A1 or A2; therefore, if a 32-bit count is desired, either counter may contain the lower bits. For example, if an event is enabled to counter A1 and a 32-bit count is desired, then enable carry line A1 must equal 0 and enable carry line A2 must be a 1. In this example, counter A1 has the least significant bits and counter A2 has the most significant.

STOP LINES

There is one stop line associated with each counter pair: one for the A counters and one for B counters. When the stop line is a 1, an event incrementing either 16-bit counter stops the computer. Mode line event stop is returned to the MCU (bit 4, channel ATB8) to show why the CPU has stopped. The MCU, after sending a clear fault signal, may restart the CPU.

COUNTER SETUP

Typically, the four counters would be set up by the MCU as follows:

1. Set the following bits as required.
    - Stop CPU on A increment (bit 9, channel BTA6)
    - Stop CPU on B increment (bit A, channel BTA6)
    - Enable carry into A1 (bit B, channel BTA6)
    - Enable carry into A2 (bit C, channel BTA6)
    - Enable carry into B1 (bit D, channel BTA6)
    - Enable carry into B2 (bit E, channel BTA6)
2. With bit F (channel BTA6) set to 0, set event and mask selection for counter A into channel BTA7.

3. Set bit F, channel BTA6 to a 1.

4. Set event and mask selection for counter B into channel BTA7.

5. If A1/B1 event code 12 for function counting has been selected, set channel BTA8 to the desired function and mask.

6. Set count line A or B (bit 6 or 7, channel BTA6) as desired.

The counters are now counting events and will continue to count until their respective count lines are dropped.

LOGIC FAULT MONITORING

There are three types of logic faults detected in the computer.

- Memory SECDED
- MIC memory parity
- Multiple match

When a logic fault is detected, the computer stops between instructions. The type of fault may be sensed on channel ATB8.

After sensing the logic fault, the MCU clears the fault via bit 7 of channel BTA1. The MCU determines the appropriate response to the fault and has the option of restarting the CPU by setting bit 3 of channel BTA1.

Information on memory SECDED faults may be found in section 3 of this manual.

Information on MIC memory parity faults may be found in the microcode description in section 3 of this manual.

Information on multiple match faults may be found in section 3 of this manual.

TEMPERATURE AND DEW POINT MONITORING

The system contains a monitoring unit which monitors critical chassis temperatures and the room dew point. If the temperature or dew point exceeds the safe limits set for the system, the monitor circuit rings an audible alarm and sends a signal to the MCU (bit 8, channel ATB8).

For memory and scalar processor sections, the temperature is monitored within the chassis. If the temperature exceeds the safe limits set for the chassis, a 15-second delayed shutdown of the chassis occurs. Simultaneously, a temperature fault signal is sent to the system power control which initiates a 20-second delayed system shutdown. If the chassis completes its shutdown in 15 seconds, a normal signal is sent to the system power control which returns the system to normal operation.

In all sections if no action is taken to correct the fault within 20 seconds, the monitoring circuit disconnects system power and locates the source of the fault.

In addition to the monitoring unit, the vector section contains a thermostat (thermisters in the scalar and memory sections). If the temperature in a particular machine section exceeds the safe upper limit, the corresponding thermostat (or thermisters) immediately disconnects power in that section.

POWER FAIL MONITORING

If the input power to the motor-generator drops for more than 100 milliseconds, the 60-Hz power fail signal is transmitted to the MCU (bit 9, channel ATB8). The system power remains up for approximately 500 milliseconds after the 60-Hz input power drops.

If 400-Hz power drops in any section of the central computer, the section power fail signal is sent to the MCU (bit A, channel ATB8). For the vector processor section, a short circuit in any section trips the corresponding circuit breaker and lights an indicator, locating where the short exists in the section. This set of indicators is contained on the annunciator panel in each section. A test switch on each panel tests the indicators.

COMPRESSOR MONITORING

High head pressure, low oil pressure, or a compressor motor fault on either condensing unit lights an indicator on the temperature monitor box, initiates an alarm, and initiates a power-down sequence. Each fault also causes an audible alarm on the condensing units.

A refrigerant liquid line temperature fault or a condenser cooling water fault lights an indicator on the monitor box. This is a warning device and is not connected into the alarm and power-down circuits.

PROGRAMMING CONSIDERATIONS 5

GENERAL

This section describes the various registers and operations of the central computer that are of particular interest to the programmer. Included are descriptions of job and monitor modes, interrupts, the invisible package, addressing modes, real-time counters, the register files, the data flag branch register, addressing modes, and general definitions and programming guides.

MONITOR AND JOB MODES

The central processor unit (CPU) operates in one of two programming modes:

- Monitor mode
- Job mode

The CPU automatically exchanges the job mode for the monitor mode when it receives an interrupt or when a job program executes an exit force (09) instruction. The monitor mode disables all interrupts and virtual addressing† and permits absolute addressing† to central storage. Any interrupts that occur during the monitor mode temporarily store until the monitor program executes an idle (00) or an exit force (09) instruction. The idle instruction causes the CPU to wait until an interrupt occurs. The exit force (09) instruction switches the CPU to the job mode and starts executing the selected job program. Switching to the job mode enables the interrupts and virtual addressing.

The purpose of the exchange is to change the prime role of the CPU. In job mode, job tasks are performed. In monitor mode, the system decisions are made and the page table is altered.

Some instructions in progress may be interrupted prior to their completion. The flags stored in the invisible package are used to restart the interrupted instruction exactly where it left off.

†Absolute and virtual addressing are described later in this section.

EXCHANGE FROM MONITOR MODE TO JOB MODE

This is always accomplished with an exit force (09) instruction. The monitor program must set up the invisible package† for the job prior to changing modes for that job via the exit force (09) instruction. The exit force operation is as follows:

1. The register file for monitor is stored into absolute memory locations 0 through $3FC0_{16}$. The register file for the job is loaded from the job's virtual memory locations 0 through $3FC0_{16}$. Any job mode reference to this area of a job's virtual memory causes the executing instruction to be treated as an illegal instruction. The absolute bit address of the job's virtual page zero is in the monitor's register S specified by the exit force instruction.

2. The CPU's major control registers and flags are loaded from the invisible package which is located starting at the absolute bit address in the monitor's register T specified by the exit force instruction. This starting address is saved in a register to provide for storing the current invisible package when returning to the monitor program.

3. The CPU's mode is changed from monitor mode to job mode. This enables the virtual address mechanism and the interrupts.

4. The contents of P (program address register) is then read up using virtual addressing, and either the initial start or the restart sequence is executed. An initial start is executed if the program is at the beginning of an instruction; a restart is executed if the program is in the middle of an instruction, that is, continuing an interrupted vector or string instruction.

ILLEGAL INSTRUCTION IN MONITOR MODE

If an attempt is made by the monitor program to perform an illegal instruction code, an automatic branch is made to the absolute address contained in the monitor's register 4. This hardware trap is to aid in the debugging of the monitor software and to trap some hardware failures. This trap is not to be utilized by the monitor software as a normal branch.

†The invisible package is described in detail later in this section.

EXCHANGE FROM JOB MODE TO MONITOR MODE

The exit force (09) instruction, channel interrupt, and access interrupt are the three normal ways of getting from job mode to the monitor program in monitor mode. Attempting to execute either a monitor-type instruction in job mode or an illegal instruction is the fourth way into the monitor. Except for the starting point in the monitor program, the operations performed in getting to the monitor are identical for the four.

The operation is as follows:

1. The current invisible registers and flags are stored into the invisible package starting at the same address used to load the invisible package when the job was entered.
2. The register file for the job is stored in virtual memory locations 0 through $3FC0_{16}$. Absolute memory locations 0 through $3FC0_{16}$ are read into the register file.
3. The CPU is changed from job to monitor mode and the virtual addressing mechanism is disabled. Any external interrupts that occur after this point are honored only if the CPU executes an idle instruction. Otherwise, the interrupts are saved until the CPU reverts to job mode, or until the monitor program clears the interrupts with a translate external interrupt (0E) instruction.
4. The monitor program executes starting at the absolute address contained in the rightmost 48 bits of registers 3, 5, 6, or 7 in the monitor's register file. The method used to enter monitor mode determines the register selection. The address in the selected register transfers to the program address register (P register).

Method of Getting to the Monitor	Register in Monitor's Register File used for Starting Address (P Address)
1. Illegal instruction, monitor-type instruction in job mode, or a reference to the register file as memory (bit address 0000 - $3FFF_{16}$).	Register 3

Method of Getting to the Monitor	Register in Monitor's Register File used for Starting Address (P Address)
2. Illegal instruction in monitor or reference to the register file as memory (bit address 0000 - $3FFF_{16}$).	Register 4
3. Exit force	Register 5
4. External interrupt	Register 6
5. Storage access interrupt	Register 7

INTERRUPTS

Interrupts consist of two main types:

- Storage access
- External

The occurrence of either type of interrupt during the job mode causes the CPU to switch to monitor mode. The monitor program then processes the interrupt.

During the monitor mode, the interrupt system is disabled except during the idle (00) instruction. Any external interrupts that occur are stored until the CPU switches back to the job mode or until the monitor program clears the interrupts with the translate external interrupt (0E) instruction.

STORAGE ACCESS INTERRUPTS

A storage access interrupt occurs when a job program attempts to reference a central storage page that does not contain the corresponding word in the page table. A storage access interrupt also occurs when a job program attempts a storage reference that violates the corresponding lockout code.

Any CPU storage reference can cause an access interrupt even if it occurs in the middle of a vector or string instruction. The virtual address of the reference causing the interrupt and bits indicating the reason for the access interrupt (cause bits) are stored in word address $xx...xxE_{16}$ of the invisible package for the corresponding job (figure 5-1). Refer to the invisible package explanation in this section.

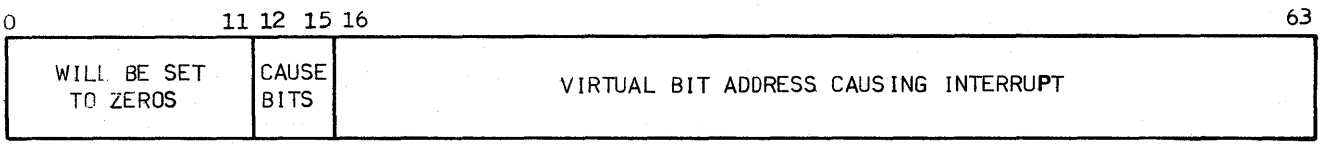


Figure 5-1. Invisible Package Word $xx...xxE_{16}$ Format for Access Interrupt

The condition of the cause bits indicate the type of storage reference that initiated the access interrupt as shown below:

Cause Bits				Type of Access Attempted
12	13	14	15	
0	1	0	0	Write operand violation
1	0	0	0	Associative word not in the page table
1	1	0	0†	Associative word not in the page table and reference attempted was a write operation
0	0	1	0	Read operand violation
0	0	0	1	Read instruction violation

Following the access interrupt, the CPU switches to the monitor mode. The program then branches to the absolute address contained in the rightmost 48 bits of register 7 in the register file for the monitor program. The monitor program proceeds to allocate space for the requested page and/or procures the requested page directly. The monitor program can restart the job where it was interrupted by using the exit force (09) instruction. If the job is to be restarted, however, the monitor program must alter the page table and central storage to include the new page.

† This is the only case where more than one cause bit is set at one time.

EXTERNAL INTERRUPTS

Each input/output (I/O) channel and the monitor interval timer can interrupt the CPU by transmitting an interrupt signal on the assigned interrupt line. The interrupt signal sets the corresponding flag bit in the external interrupt register. The external line assignments are listed in table 5-1.

I/O CHANNEL INTERRUPT LINES

As shown in table 5-1, each I/O channel has an external interrupt line assignment. The transmission of the interrupt from B (IFB) signal on the corresponding external interrupt line sets the corresponding external interrupt register flag bit. The setting of this bit indicates to the CPU that the I/O device (peripheral station) has stored a message in a predetermined location in central storage.

TABLE 5-1. EXTERNAL INTERRUPT LINES

External Interrupt Line	Assignment
0	Not available
1	I/O channel 1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	I/O channel 12
13	Not used
14	Not used
15	Monitor interval timer

MONITOR INTERVAL TIMER INTERRUPT

When the monitor interval timer (described in this section) decrements to a zero count, an external interrupt signal is transmitted on line 15. The resultant setting of external register flag bit 15 indicates to the CPU that the specified period initially set in the monitor interval timer has elapsed, requiring processing by the monitor program.

INVISIBLE PACKAGE

The invisible package contains the address and control information necessary to begin a new job or to continue a job interrupted during execution. Each invisible package is associated with a job. The invisible package for a particular job is stored at 16 consecutive word addresses in central storage beginning at the initial address assigned by the monitor program. The invisible package is always stored starting at an even numbered sword address. Therefore, the rightmost 10 bits of the starting address of the invisible package must be zeros. Refer to the exit force (09) instruction in the instructions section of this manual.

The monitor must set up an invisible package for each job. There is no invisible package for the monitor program itself.

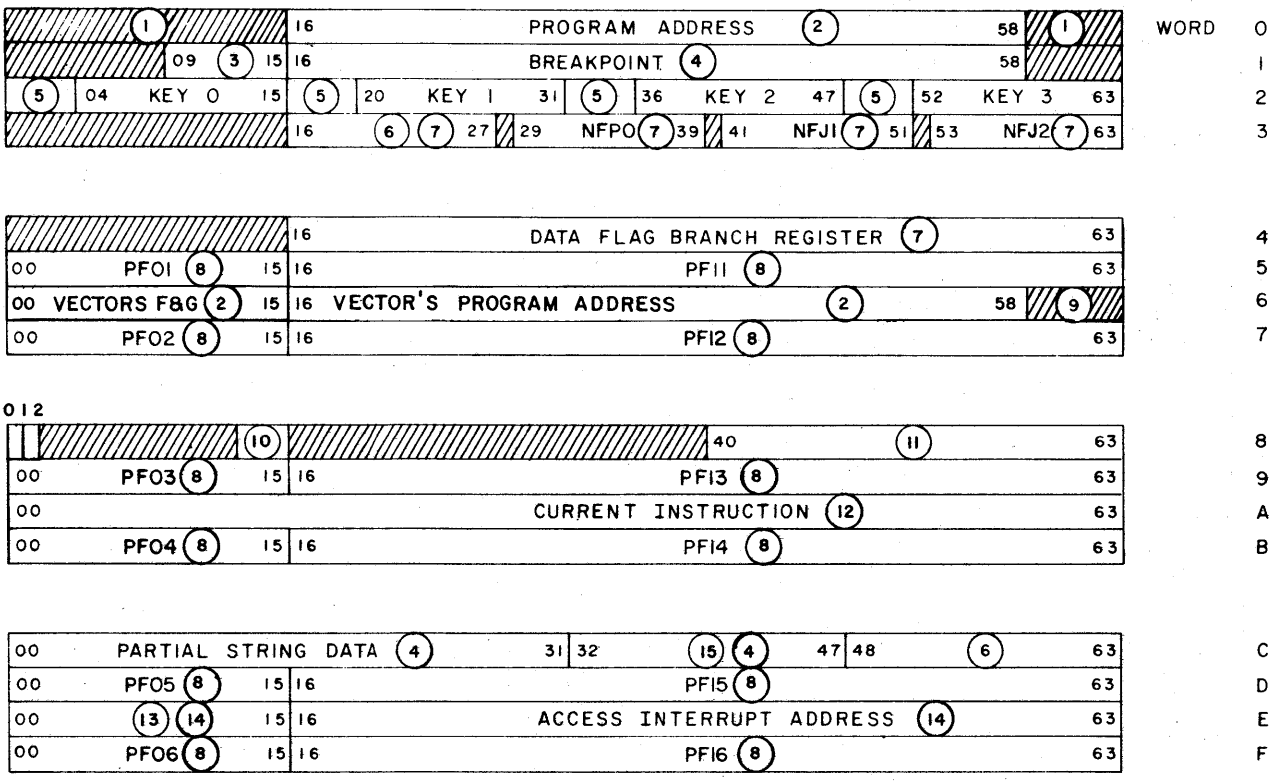
When the CPU switches from monitor to job mode, the invisible package for the corresponding job is automatically loaded from central storage beginning at the address assigned to that job. The invisible package data is loaded into the appropriate registers in the CPU.

When the CPU switches from job to monitor mode, as in an interrupt, the contents of the corresponding registers are automatically stored in central storage as the invisible package for that job.

If a job is to be reentered, the monitor should not alter the job's invisible package except for possibly the keys.

Both the vector and scalar processors load and store the invisible package. The scalar processor stores in memory first. The write enables must be controlled by the vector processor to ensure that current information from the scalar processor is not written over by the vector processor.

Figure 5-2 shows the invisible package format.



NOTES:

1. Bits 0 through 15 and 59 through 63 are not used and must be set to zeros.
2. Quantity is loaded or read/stored or written by the scalar processor only.
3. Usage bits for breakpoint register.
4. Quantity is loaded/stored by vector processor only.
5. Usage lockout bits for each key.

   Bit 0 of the first two key words (bit 0 and bit 16 of word 2) specify the small page size.

   Bit 1, if set, locks out CPU write operation.

   Bit 2, if set, locks out CPU read operation.

   Bit 3, if set, locks out CPU instruction references.

   Quantity is loaded by the scalar processor and not stored by either processor.

Figure 5-2. Invisible Package Format

(6) Bit 16 Flag 0
Bit 17 Flag 1
Bit 18 Flag 2
Bit 19 Flag 3
Bit 20 Interrupt flag
Bit 21 Not used
Bit 22 Load/store 1
Bit 23 Load/store 2
Bit 24 Subfunction bit 0
Bit 25 Subfunction bit 1
Bit 26 Subfunction bit 2
Bit 27 Subfunction bit 3

(7) Quantity is loaded/stored by the vector processor only.

(8) Words 5, 7, 9, B, D, and F are loaded by both the scalar and vector processors. These words are stored by the vector processor if the vector restart bit (word 8 bit 0) equal to 1 and by the scalar processor if the bit equal to 0.

(9) Bits 59 through 63 are not used and must be set to zeros.

(10) Bit 0 vector restart bit. The vector processor's instruction register receives bits 0 through 15, word 6 and bits 16 through 63, word A. A vector restarts without reloading the vector instruction from memory only if bits 16 through 63, word A are not needed to restart (bit 0, word 8 equal to 1).

Bit 1 Register file's scalar enable (bits 0 and 1 are loaded by the scalar processor and stored by the vector processor).
Bits 2-11 are not used; bits 8-11 are reserved for possible use as a small page size mask.
Bit 12 Stall bit. This is 1 if no data is processed.
Bit 13 Not used.
Bit 14 Monitoring counters enable.
Bit 15 ASCII=0, EBCDIC=1 (bits 12 through 15 are loaded/stored by the vector processor only.)

(11) Job interval timer. Quantity is loaded/stored by the vector processor only.

(12) Quantity is stored by the scalar processor and loaded by neither.

(13) Access interrupt cause bits (address X0 and XE) 0 through 11 are not used and are set to zeros.
Bit 12 associative word not in page table.
Bit 13 write operand violation attempted.
Bit 14 read operand violation attempted.
Bit 15 read instruction violation attempted.

(14) Quantity is stored by the scalar processor and loaded by neither.

(15) String internal data and control. The data control saved in bits 32 through 63 of invisible package word C is dependent on the instruction being interrupted.

Figure 5-2. Invisible Package Format (Contd)

ADDRESSING MODES

The computer system uses two modes of addressing central storage.

- Virtual addressing
- Absolute addressing

VIRTUAL ADDRESSING

Virtual addressing provides an efficient, dynamic method of allotting portions of central storage to each job program by the monitor program. Virtual addressing is used exclusively when the CPU is in job mode. The switching of the CPU to monitor mode automatically disables virtual addressing. However, central storage recognizes all addresses as being absolute. Thus, the virtual addressing control circuits convert virtual addresses to the corresponding absolute addresses.

PAGES

Portions of central memory are logically partitioned into pages; the central computer has small and large page sizes. A small page contains either 512, 2048, or 8192 64-bit words selected by bits 0 and 16 in the third word (keys) of the invisible package. The bits are interpreted as follows:

Bits		Description
0	16	
0	0	Small pages are 512 words
1	0	Small pages are 2048 words
1	1	Small pages are 8142 words
0	1	Undefined

Only one small page size may reside in the associative page table. The default size is 512 words. A large page contains 65,536 64-bit words.

The monitor program allots a page or pages to each job program. All of the words in a page are identified by a common page identifier. The common page identifier is an absolute address which locates the page in central memory.

VIRTUAL ADDRESS FORMAT

Figure 5-3 shows the virtual address formats for the 512-, 2K-, 8K-, and 65K-word pages, respectively. Note that the size of the virtual page identifier varies depending on the word page size. Table 5-2 shows the page size and the virtual page and word identifiers' bit sizes for each word page. This difference results from the number of bits needed to locate the word in the page.

The bit, byte, halfword, and word identifier portions of the virtual address are absolute. Thus, when the virtual page identifier is converted into an absolute page identifier, these portions of the virtual address are substituted directly into the absolute address.

TABLE 5-2. PAGE SIZE SPECIFICATION

Page Size	Virtual Page Identifier (bits)	Word Identifier (bits)
512	33	9
2048	31	11
8192	29	13
65,536	26	16

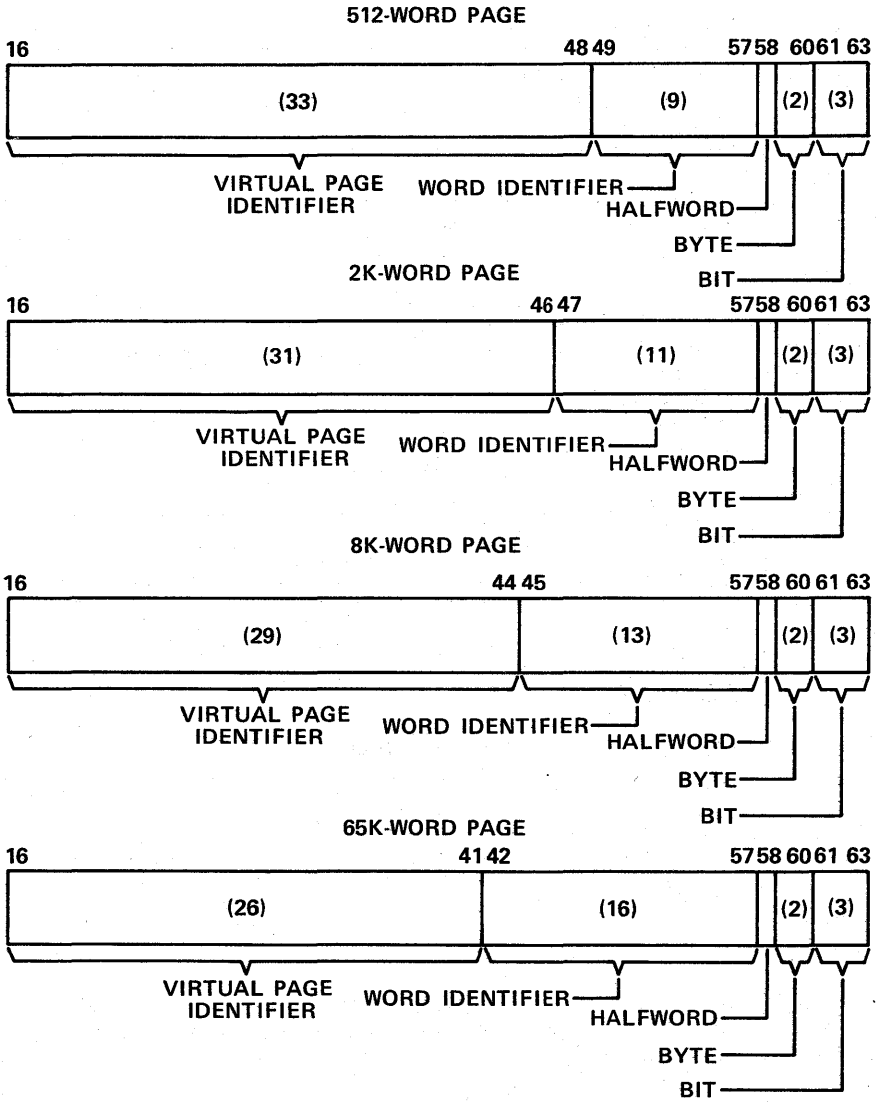
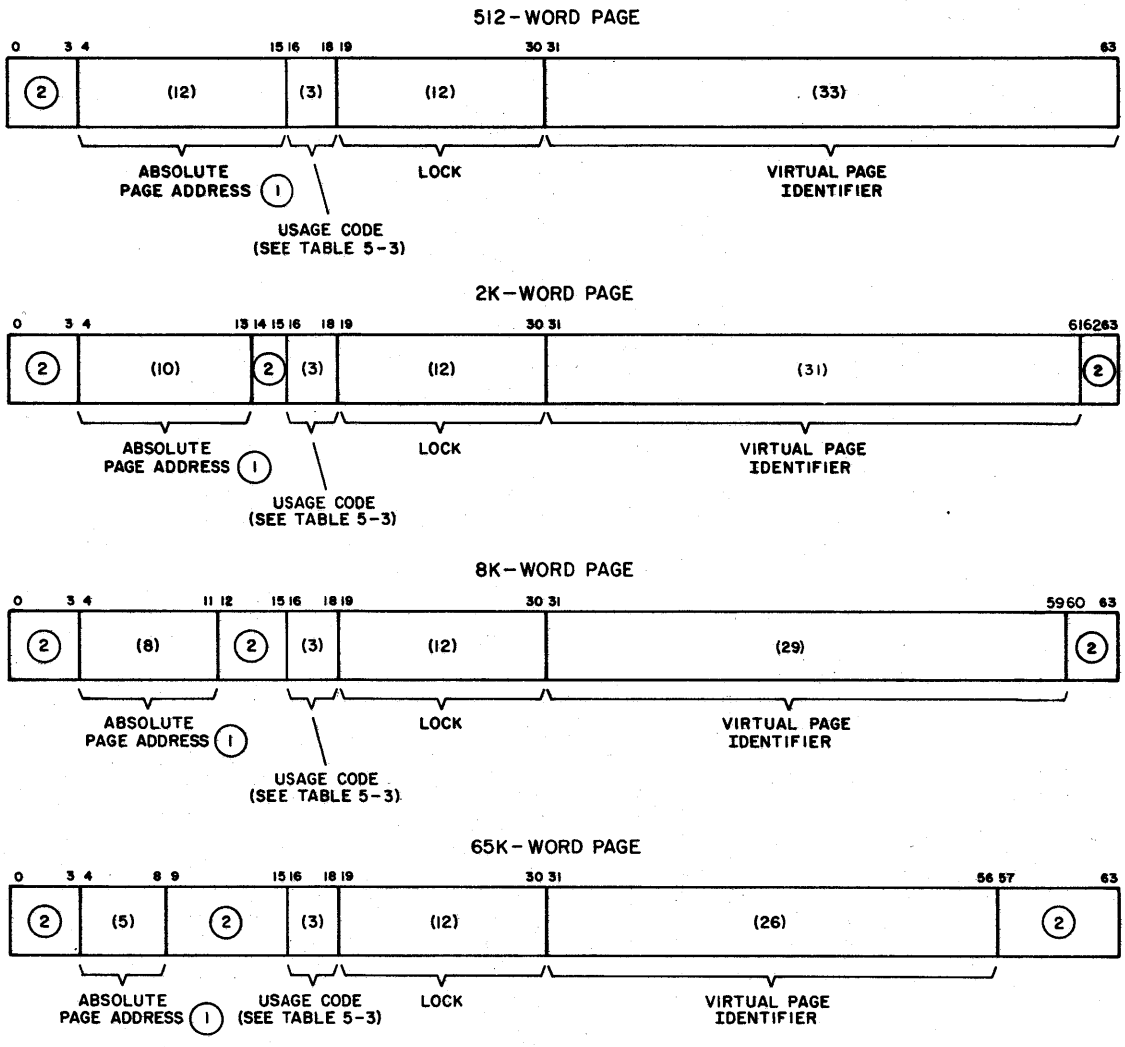


Figure 5-3. Virtual Address Formats

ASSOCIATIVE WORDS

The associative words contain the information necessary to convert a virtual address into an absolute address. The monitor program must assemble the associative words into a page table as necessary for a given run. Figure 5-4 shows the formats of the associative words for the 512-word page and 65K-word page, respectively.

If a page has been referenced with code bits in table 5-3, a job program has made at least one storage reference to the page defined by the associative word. If a page is



(1) IF 500K WORD TOTAL CENTRAL STORAGE IS USED, BITS 4 AND 5 MUST BE A 0.
IF 1000K WORD TOTAL CENTRAL STORAGE IS USED, BIT 4 MUST BE A 0.

(2) BITS MUST BE SET TO ZEROS.

Figure 5-4. Associative Word Formats

TABLE 5-3. ASSOCIATIVE WORD USAGE CODES

Code Bits (16 17 18)	Definition
000	End of page table
001	Null associative word
010	Small page has not been referenced by the CPU
011	Large page has not been referenced by the CPU
100	Small page has been referenced by the CPU
101	Large page has been referenced by the CPU
110	Small page has been altered by the CPU
111	Large page has been altered by the CPU

altered, a job program has performed a write operation on at least one bit in the page defined by the associative word. In the monitor mode, the CPU does not use the associative words in addressing. Thus, alteration or referencing storage by the monitor program is not recorded in the associative words.

LOCK

A lock is a 12-bit quantity contained in each associative word (figure 5-4). The lock associates a page of central storage with a job program or several job programs.

KEYS

The monitor assigns four 12-bit keys to each job. The keys for a particular job are read from central storage as part of the invisible package for that job. The monitor program transfers the keys to the virtual address key register (figure 5-5). After the virtual page address portion of an associative word matches with the corresponding portion of a virtual address, one of the four keys for the job must match the lock in the associative word before the storage reference can take place.

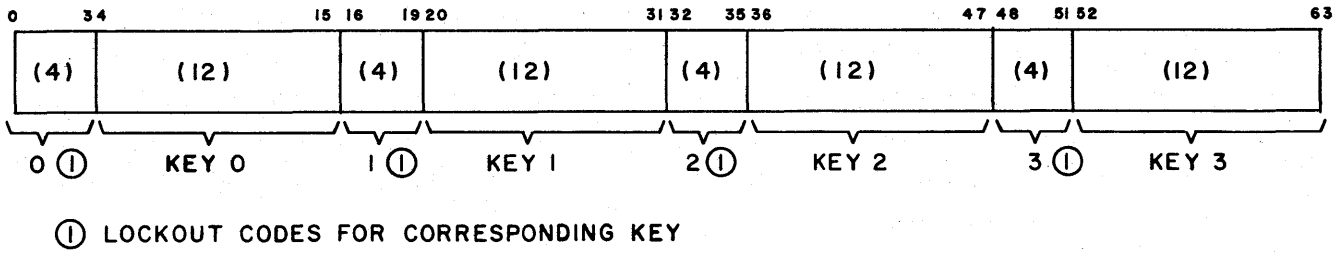


Figure 5-5. Virtual Address Key Register Format

Figure 5-5 shows that each key is associated with a four-bit lockout code. The setting of a particular bit in this code locks out the corresponding type of storage reference. Table 5-4 lists each bit of the lockout code and the type of storage reference locked out if the bit is set.

If a key matches the lock of an associative word for a particular storage reference, but the operation is disabled by the lockout code for that type of reference, a storage access interrupt takes place. A storage access interrupt causes an exchange to the monitor mode.

TABLE 5-4. LOCKOUT CODES

Bit Position	
0 1 2 3	Type of Storage Reference Locked Out
1 X X X	†
X 1 X X	CPU write operations
X X 1 X	CPU read operations
X X X 1	CPU instruction references
NOTES 1. The actual bit number depends on the key field to which it corresponds (figure 5-5). 2. X denotes that the bit can be 0 or 1.	
†Bits 0 and 16 define the small page size; bits 32 and 48 must be set to zero.	

ASSOCIATIVE REGISTERS

The scalar processor contains 16 64-bit associative registers (ARs). Each AR contains one associative word. The ARs contain the first 16 associative words in the page table. For example, if the computer system consists of one million words of central storage and if only 65K-word pages are selected, the associative words for all 16 pages would be contained in the ARs. In the monitor mode, the contents of the ARs can be stored into or loaded from central storage with the store associative registers (0C) or load associative registers (0D) instructions, respectively.

The contents of the ARs cannot be referenced directly for read or write operations except through the 0C and 0D instructions.

SPACE TABLE

The space table (figure 5-6) consists of the locations in central storage that contain the list of associative words. The space table starts at absolute bit address 4400_{16} (word address 0110_{16}) and may continue to $1FFEOO_{16}$. The space table extends into central storage until an end of page table code is found in the usage bits (table 5-3) of the corresponding associative word. If no end of page table entry is found before location $3FFCO_{16}$, the search hardware will loop between addresses $20,000_{16}$ and $3FFCO_{16}$, resulting in a CPU hang. Thus, the space table serves as an extension of the ARs to make up a complete page table.

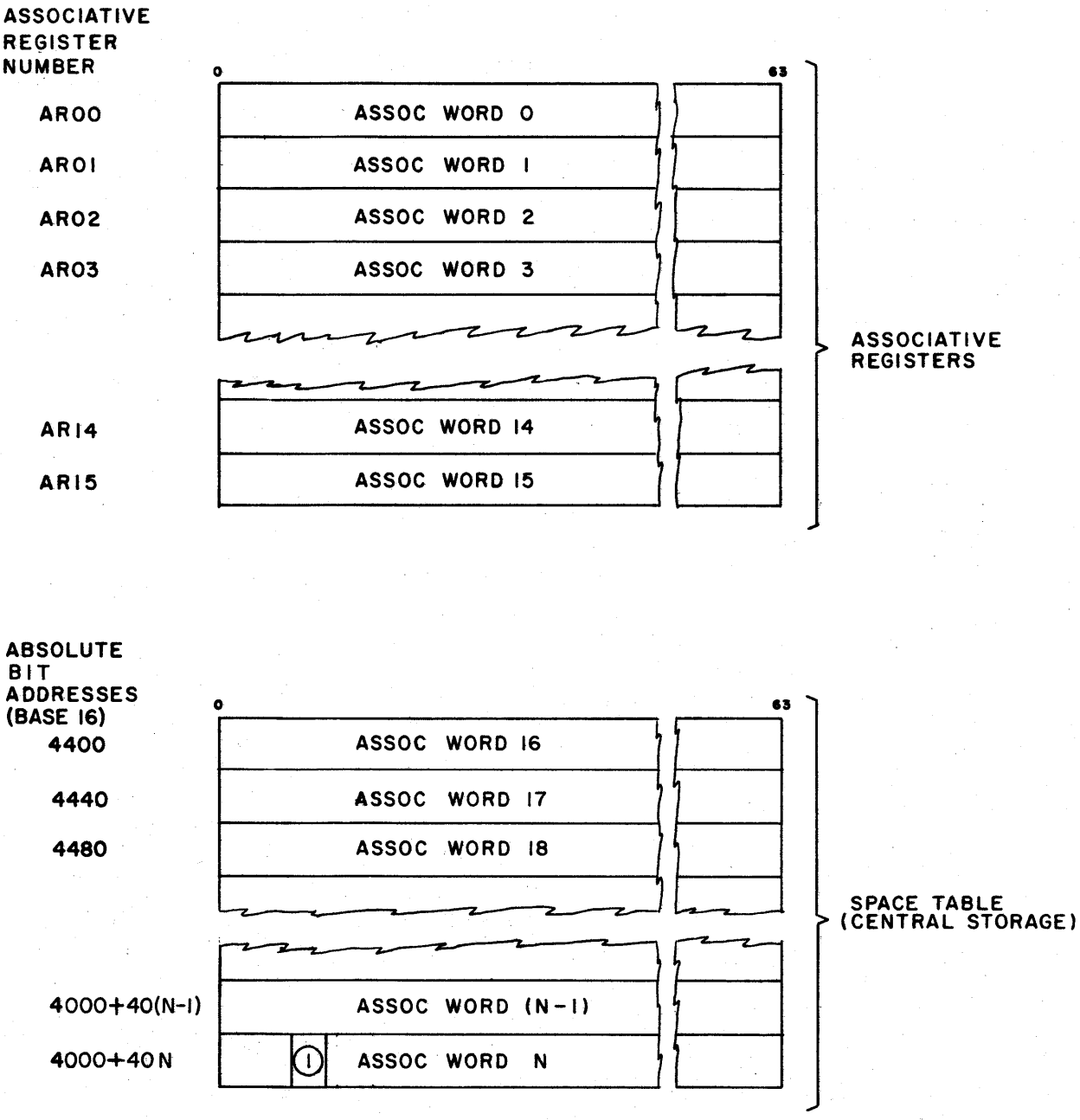
PAGE TABLE

The page table contains the complete list of associative words and includes both the associative registers and space table. The associative words contained in the page table define the pages currently allotted space in central storage. Figure 5-6 shows the format of the page table. Note that if the associative words in the associative registers are stored in central storage with the store associative registers (0C) instruction, they are stored in 16 consecutive 64-bit storage locations of absolute bit addresses 4000_{16} through $43C0_{16}$.

Table 5-5 lists page table restrictions and requirements.

TABLE 5-5. PAGE TABLE RESTRICTIONS AND REQUIREMENTS

Number	Restrictions and Requirements
1	There must be at least one END in the page table.
2	A page must appear only once in the page table. If a page appears more than once, the results are undefined, and the multiple match fault may be set.
3	Before looking at the page table, the ARs must be stored in memory. The page table, in memory, starts at address 4000_{16}.
4	Data words, after the end of table word in the same sword and in the sword following, may be altered during space table searches.



(1) END OF PAGE TABLE USAGE CODE

Figure 5-6. Page Table Format

OPERATION OF VIRTUAL ADDRESSING

In the processing of a job program, each virtual address is transmitted from the stream unit to the scalar processor. The scalar processor compares the virtual page identifier in the virtual address (figure 5-3) with the corresponding portion of each associative word (figure 5-4) in the page table. If the virtual page identifiers match and the lock matches one of the four keys, a match condition occurs. If a match results, the absolute page address associated with the match-producing entry in the page table is combined with the applicable portion of the word identifier sent from stream. The upper 17 bits of this combined address references one sword (eight 64-bit words) from central storage. The remaining word, half-word, byte, and bit identifiers remain in stream and select the word, half-word, byte, and/or bit from the words transmitted from the scalar processor. If the end of the page table is detected with no preceding match condition, or if a match results but the operation is disabled by the lockout code, a storage access interrupt results.

For a description of a page table search, refer to the scalar processor area of the central processor section of this manual (section 3).

ABSOLUTE ADDRESS

The absolute address formed by page table translation receives the page address portion from bits 4 through 15 of the associative word (figure 5-7). For 512 word pages, 12 bits (4 through 15) are placed in bit locations 37 through 48 of the absolute address allowing use of 4096 possible pages in job mode with two million word memory size configuration. Bits 49 through 54 of the absolute address receive the corresponding bits from the virtual address. For 2K word pages, only 10 bits (4 through 13) are placed in bit locations 37 through 46 of the absolute address. Bits 47 through 54 of the absolute address receive the corresponding bits from the virtual address. For 8K word pages, only 8 bits (4 through 11) are placed in bit locations 37 through 44 of the absolute address. Bits 45 through 54 of the absolute address receive the corresponding bits from the virtual address. For 65K word pages, only five bits (4 through 8) are placed in bit locations 37 through 41 of the absolute address. Bits 42 through 54 of the absolute address receive the corresponding bits from the virtual address; this allows 32 large pages usable with a two million word memory.

In a two million memory configuration, bit 37 of the absolute address indicates which upper or lower million word portion of memory is referenced. In a one million memory configuration, bit 38 of the absolute address indicates which upper or lower half-million portion of memory is referenced. If bit 4 of the absolute page address in the associative word is set for either page size, the absolute address formed attempts to reference nonexistent upper words of memory. This type of memory reference is undefined, and an illegal reference to memory occurs. In a 512K word memory configuration, if bit 4 or bit 5 of the absolute page address in the associative word is set for either page size, the absolute address formed attempts to reference nonexistent upper words of memory. This type of memory reference is undefined, and an illegal reference to memory occurs.

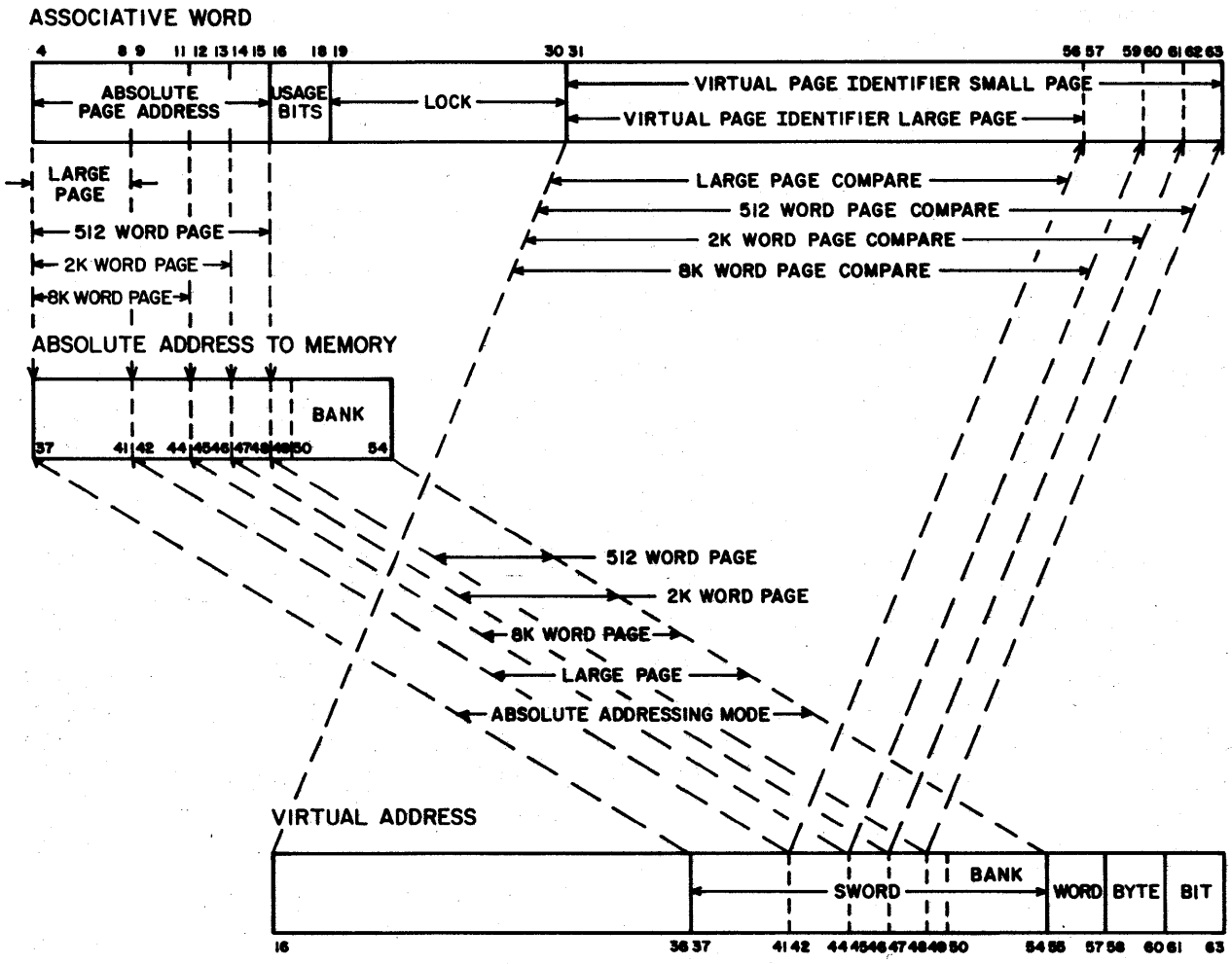


Figure 5-7. Virtual Address to Absolute Address

REAL-TIME COUNTERS

The CPU contains three counters that can be used for real-time programming applications:

- Free running clock
- Monitor interval timer
- Job interval timer

Each of these counters is described in the following paragraphs.

FREE RUNNING CLOCK COUNTER

This counter consists of a free running 47-bit counter that is incremented at a 1-MHz rate, and a positive sign bit for a total of 48 bits. The free running clock counter is never cleared. The contents of this counter can be stored in a designated register T with the transmit real-time clock to T (39) instruction.

MONITOR INTERVAL TIMER

This 24-bit counter is decremented at a 1-MHz rate. The transmit (R) to monitor interval timer (0A) instruction loads the contents of the designated register R into the monitor interval timer counter when the computer is in the monitor mode. The timer can be activated by loading it with any quantity other than all zeros. Once it is activated, the timer decrements at a 1-MHz rate until it reaches an all zero count. When the counter reaches a zero count, it causes an external interrupt on channel 15 which is processed like any other external interrupt. At this point the timer is deactivated until it is loaded with some value other than zero.

The monitor interval timer is deactivated by any one of the following three methods.

1. Master clear.
2. Loading it with all zeros.
3. Decrementing it to a zero count.

JOB INTERVAL TIMER

This 24-bit counter is decremented at a 1-MHz rate and can be loaded (job mode only) from a designated register R using the transmit R to job interval timer (3A) instruction. Once loaded, the job interval timer continues to decrement until either an exchange to monitor mode occurs, the timer decrements to zero, or the timer is loaded with zeros. If an exchange to monitor mode occurs, the job interval timer stops decrementing and the operation stores the current contents of the timer in the invisible package for that job. When the execution of that job resumes, the operation loads the job interval timer from the invisible package and resumes decrementing it. When the timer is decremented to zero, the CPU sets bit 36 in the DFB register. Refer to the data flag branch register description in this section.

Loading zeros deactivates the timer. This action does not set bit 36 of the data flag branch register. Master clear also deactivates the timer.

The job interval timer is deactivated by any one of the following three methods.

- Master Clear.
- Loading it with all zeros.
- Decrementing it to a zero count.

REGISTER FILE

For register operations, the 8-bit instruction designators directly address the 256_{10} registers of the register file. During program execution (monitor or job), these registers reside in the CPU's register file. When an exchange operation occurs, the registers are stored into the first 256_{10} memory locations of the particular job or monitor mode program beginning at bit address zero (absolute address if in monitor mode and virtual if in job mode). The registers may not be referenced as memory by their associated monitor or job program. The only exceptions to this rule are the B7 and BA instructions with G-bit 7 set. (Refer to B7 and BA instructions in section 6 of this manual.)

Figure 5-8 shows a map of the register file and the relationship between the register, its storage address, and its 8-bit designator. The number on the right is the bit address and the number on the left is the value of the 8-bit designator for the 64-bit operand. The number inside the register represents the value of the 8-bit designator for the 32-bit operand.

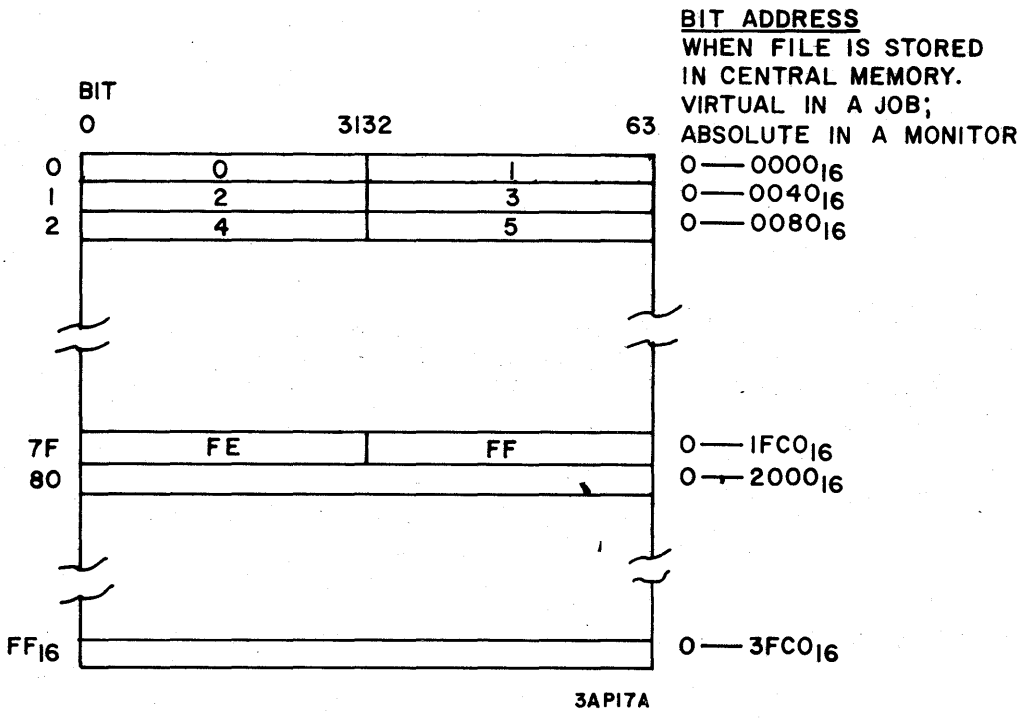


Figure 5-8. Register File

REGISTER FILE RESTRICTIONS

Certain registers within the register file have programming restrictions. The restrictions are grouped according to the instruction designator number of the register.

REGISTER 0 (TRACE REGISTER) RESTRICTIONS

Register file address zero (figure 5-9) is used as the trace register in the 64-bit mode only. The trace register contains the address from which the most recent branch was taken. Register zero can be referenced by executing a 7D instruction. Refer to the instruction section for the mode of the 7D instruction which moves register zero to central memory. The maintenance station reads register zero by storing the register file and reading virtual/absolute zero in central memory. After a job to monitor exchange, the job's virtual address zero in memory contains the address of the last branch taken prior to the exchange operation. After a monitor to job exchange, monitor's address zero (absolute zero) contains the address of the last branch taken prior to the exchange operation. The B9 and BA instructions can also read register zero for data.

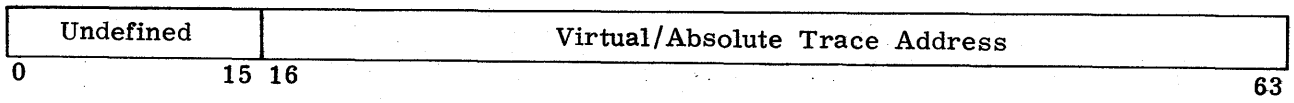


Figure 5-9. Virtual/Absolute Address Zero

REGISTER 0 CONTENTS RESULTING FROM AN EXCHANGE OPERATION

During a monitor to job exchange, the content of the trace register and the appropriate memory location for register zero exchange as follows:

	Content Before Exchange	Content After Exchange
Absolute address zero	A	C
Virtual address zero	B	B
Trace register	C	B

During a job to monitor exchange, the content of the trace register and the appropriate memory location for register zero exchange (swap) as follows:

	Content Before Exchange	Content After Exchange
Absolute address zero	A	A
Virtual address zero	B	C
Trace register	C	A

If monitor and job mode share a common register file (refer to common register files for job and monitor modes in this section), the following will occur upon a monitor to job or job to monitor exchange.

	Content Before Exchange	Content After Exchange
Absolute address zero	A	B
Virtual address zero	A	B
Trace register	B	B

REGISTER 0 CONTENT RESULTING FROM A SWAP (7D) INSTRUCTION

During a swap (7D) instruction involving register zero as part of the register field, note a required peculiarity. Although the current content of the trace register is sent to the appropriate memory location for register zero, the current content of the trace register is not altered.

	Content Before 7D	Content After 7D
Memory location for register zero	A	B
Trace register	B	B

REGISTER 0 WHEN REFERENCED BY AN INSTRUCTION DESIGNATOR

When referenced by an instruction designator, register zero provides machine zero as an operand except when used as a source register for a base address or other description for a vector or string instruction. In this case, register zero appears to contain 64 zero bits. The use of a zero address may cause the instruction to be treated as an illegal instruction. The use of a zero field length may cause the instruction to become undefined as when used in the A0 to AF instruction. If register zero is specified as the destination register, the instruction typically performs normally with data flags being set, if warranted, but no data is stored. Some instructions become undefined if register zero is specified as a destination register.

Table 5-6 shows which operand is obtained when register zero is specified for a source operand. To simplify the table, the specifying of register zero as a destination register is ignored since it causes the result to be lost. A blank in the table indicates that register zero cannot be specified or that register zero may only be specified as a destination register. The instruction designators R, S, T, G, X, A, Y, B, Z, and C are used for convenience, although they do not apply to all instructions. The following list contains definitions of symbols in the table.

Symbol	Result When Register Zero is Used as an Operand
A	All zeros are provided.
C	No control vector is used.
M	Machine zero is provided. $8000\ 0000\ 0000\ 0000_{16}$ 64-bit mode $8000\ 0000_{16}$ 32-bit mode
N	Instruction performs as a no-op.
O	A mask of all ones is provided.
Z	All zeros in the used portion. In this instance, the leftmost bit is not used; thus, machine zero and all zeros are undistinguishable.

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO

Op Code	Instruction Designators R	S	T
00			
04	Z		
06			
08			
09		Z	Z
0A	Z		
OC			
OD			
OE	Z	Z	
OF	Z	A	
10	M		
11	Z		
12	Z	Z	
13	Z	Z	Z
14	A	A	A
15	A	A	A
16	A	A	A
17	A	A	A
18	Z	Z	A
19		Z	A
1A		Z	A
1B	Z	Z	A
1C	A	A	A
1D	A	A	A
1E	A	Z	
1F	A	Z	

Op Code	Instruction Designators R	S	T
20	M	M	Z
21	M	M	Z
22	M	M	Z
23	M	M	Z
24	M	M	Z
25	M	M	Z
26	M	M	Z
27	M	M	Z
28		Z	A
29		Z	A
2A			
2B	M	Z	
2C	M	M	
2D	M	M	
2E	M	M	
2F		Z	Z
30	M		
31	Z	Z	Z
32		Z	Z
33			Z
34	M	Z	
35	Z	Z	Z
36		Z	Z
37			
38	M		
39			
3A	Z		
3B	Z		
3C	Z	Z	
3D	Z	Z	
3E			
3F	Z		

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators R	S	T
40	M	M	
41	M	M	
42	M	M	
44	M	M	
45	M	M	
46	M	M	
48	M	M	
49	M	M	
4B	M	M	
4C	M	M	
4D			
4E	Z		
4F	M	M	
50	M		
51	M		
52	M		
53	M		
54	M	Z	
55	M	M	
58	M		
59	M		
5A	M		
5B	Z	Z	
5C	M		
5D	M		
5E	Z	Z	
5F	Z	Z	M

Op Code	Instruction Designators R	S	T
60	M	M	
61	M	M	
62	M	M	
63	M	Z	
64	M	M	
65	M	M	
66	M	M	
67	M	Z	
68	M	M	
69	M	M	
6B	M	M	
6C	M	M	
6D	M	Z	
6E	M	Z	
6F	M	M	
70	M		
71	M		
72	M		
73	M		
74	M	Z	
75	M	Z	
76	M		
77	M		
78	M		
79	M		
7A	M		
7B	Z	Z	
7C	M		
7D	A	†	A
7E	Z	Z	
7F	Z	Z	M

† Refer to the swap 7D instruction in section 6 of this manual.

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators G	X	A	Y	B	Z	C
80		Z	A†	Z	A†	C	A
81		Z	A†	Z	A†	C	A
82		Z	A†	Z	A†	C	A
83		Z	A†	Z	A†	C	A
84		Z	A†	Z	A†	C	A
85		Z	A†	Z	A†	C	A
86		Z	A†	Z	A†	C	A
87		Z	A†	Z	A†	C	A
88		Z	A†	Z	A†	C	A
89		Z	A†	Z	A†	C	A
8B		Z	A†	Z	A†	C	A
8C		Z	A†	Z	A†	C	A
8F		Z	A†	Z	A†	C	A
90		Z	A†			C	A
91		Z	A†			C	A
92		Z	A†			C	A
93		Z	A†			C	A
94		Z	A†	Z	A†	C	A
95		Z	A†	Z	A†	C	A
96		Z	A†			C	A
97		Z	A†			C	A
98		Z	A†			C	A
99		Z	A†			C	A
9A		Z	A†			C	A
9B		Z	A†	Z	A†	C	A
9C		Z	A†			C	A

Op Code	Instruction Designators G	X	A	Y	B	Z	C
A0		A	Z†	A	Z†	A	Z
A1		A	Z†	A	Z†	A	Z
A2		A	Z†	A	Z†	A	Z
A4		A	Z†	A	Z†	A	Z
A5		A	Z†	A	Z†	A	Z
A6		A	Z†	A	Z†	A	Z
A8		A	Z†	A	Z†	A	Z
A9		A	Z†	A	Z†	A	Z
AB		A	Z†	A	Z†	A	Z
AC		A	Z†	A	Z†	A	Z
AF		A	Z†	A	Z†	A	Z
B0		Z	M	Z	Z	Z	
B1		Z	M	Z	Z	Z	
B2		Z	M	Z	Z	Z	
B3		Z	M	Z	Z	Z	
B4		Z	M	Z	Z	Z	
B5		Z	M	Z	Z	Z	
B6	Z						
B7		Z	A	Z	A†	Z	A
B8		Z	A			C	A
B9††		Z	Z			Z	Z
BA††		Z	Z	Z	Z	Z	A
BB			A†		A†	A	Z
BC			Z			A	Z
BD			Z		Z	A†	A
BE							
BF							

†If register zero is selected to broadcast a constant, machine zero is that constant.
††The B9 and BA instructions can read register zero for data.

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators G	X	A	Y	B	Z	C
C0		Z	A†	Z	A†	C	
C1		Z	A†	Z	A†	C	
C2		Z	A†	Z	A†	C	
C3		Z	A†	Z	A†	C	
C4		Z	A†	Z	A†	A	
C5		Z	A†	Z	A†	A	
C6		Z	A†	Z	A†	A	
C7		Z	A†	Z	A†	A	
C8			A		A	C	Z
C9			A		A	C	Z
CA			A		A	C	Z
CB			A		A	C	Z
CD							
CE	Z						
CF		Z	A	Z	A†	Z	Z
D0		Z	A†	Z	A†	C	A
D1		Z	A			C	A
D4		Z	A†	Z	A†	C	A
D5		Z	A			C	A
D6	Z	Z	A	Z	A	A	O
D7		Z	A	Z	Z	Z	Z
D8		Z	A			C	
D9		Z	A			C	
DA		Z	A			C	
DB		Z	A			C	
DC		Z	A†	Z	A†	C	
DD		A	A†	A	A†		
DE		Z	A†	Z	A	C	A
DF			M		M	C	A

Op Code	Instruction Designators G	X	A	Y	B	Z	C
E0		Z	A	Z	A	Z	A
E1		Z	A	Z	A	Z	A
E2		Z	A	Z	A	Z	A
E3		Z	A	Z	A	Z	A
E4		Z	A	Z	A	Z	N
E5		Z	A	Z	A	Z	N
E6		Z	A	Z	A	Z	N
E7		Z	A	Z	A	Z	N
E8		Z	A	Z	A		
E9		Z	A	Z	A		
EA		Z	A	Z	A	Z	A
EB		Z	A	Z	A	Z	A
EC		Z	A	Z	A	Z	A
ED		Z	A	Z	A	Z	A
EE		Z	A	Z	Z	Z	A
EF		Z	A	Z	Z	Z	A
F0		Z	A	Z	A	Z	A
F1		Z	A	Z	A	Z	A
F2		Z	A	Z	A	Z	A
F3		Z	A	Z	A	Z	A
F4		Z	A	Z	A	Z	A
F5		Z	A	Z	A	Z	A
F6		Z	A	Z	A	Z	A
F7		Z	A	Z	A	Z	A
F8		Z	A			Z	A
F9		Z	A			Z	A
FA		Z	A		Z	Z	A
FB		Z	A			Z	A
FC		Z	A			Z	A
FD		Z	A	Z	A	Z	O
FE		Z	A	Z	A	Z	O
FF		Z	A	Z	A	Z	O

†If register zero is used to broadcast a constant, machine zero is that constant.

REGISTERS 1 AND 2 (64-BIT), 2 THROUGH 5 (32-BIT) RESTRICTIONS

If data flag branches are used, 64-bit registers 1 and 2 must be reserved exclusively for that function. Register 1 stores the data flag branch exit address and register 2 the data flag branch entry address. Refer to the data flag branch register description in this section.

REGISTERS 0 THROUGH 7 (64-BIT), 0 THROUGH F (32-BIT) MONITOR MODE RESTRICTIONS

In 64-bit mode, registers 0, 1, and 2 (or in 32-bit mode registers 0 through 5) have the restrictions during monitor mode as previously described. In 64-bit mode, registers 3 through 7 (or in 32-bit mode registers 6 through F) are used for the undefined instructions, exit force, external interrupt, and storage access interrupt entry points. Refer to the exchange from job mode to monitor mode description in this section.

REGISTER 1 (32-BIT) RIGHTMOST HALF OF 64-BIT REGISTER 0

Any reference to 32-bit register one is undefined.

COMMON REGISTER FILE FOR MONITOR AND JOB MODES

Monitor and job modes have perfectly overlapping register files if monitor executes an exit force instruction (09) with either designator S or the contents of register S equal to zero. In an exchange from monitor to job mode, the monitor's register file is stored starting at absolute bit address zero. The job's register file is then loaded from the first 256 locations of its virtual page zero. Since register S contains the absolute address of the job's virtual page zero (refer to exit force instruction) and in this case S is equal to zero, the register file for the job is loaded from the same memory locations as the monitor's register file was stored. Also, since the rightmost 15 bits of register S must contain zeros (refer to exit force instruction), only a perfect overlap occurs. Thus, following the exchange, the job's register file is identical to the monitor's register file.

When exchanging from job mode back to monitor mode, the job's register file is stored where it came from; in this case, starting at absolute bit address zero. The monitor's register file is then loaded from the same locations causing it to be identical to the job's register file.

DATA FLAG BRANCH REGISTER

The data flag branch (DFB) register is a 64-bit register (figure 5-10) that provides the programmer with an automatic branching feature to a special subroutine for certain operands, results, conditions, etc. The DFB register eliminates the time penalty of explicitly checking for special programming conditions. If a condition previously selected to cause an automatic branch occurs during the execution of an instruction, the computer completes the instruction, stores the address of the next instruction that would have been executed in the address portion of register 01, and branches to the address contained in register 02.

Many register instructions are executed in parallel, and there may be some uncertainty as to which instruction caused the data flag condition. The data flag set condition may have occurred during an instruction which was issued a number of instructions before the one just completed. A flag on a scalar register instruction (divide, square root and convert BCD to binary) could have occurred 0 to 35 instructions earlier. A flag on the other register instructions could have occurred 0 to 5 instructions earlier.

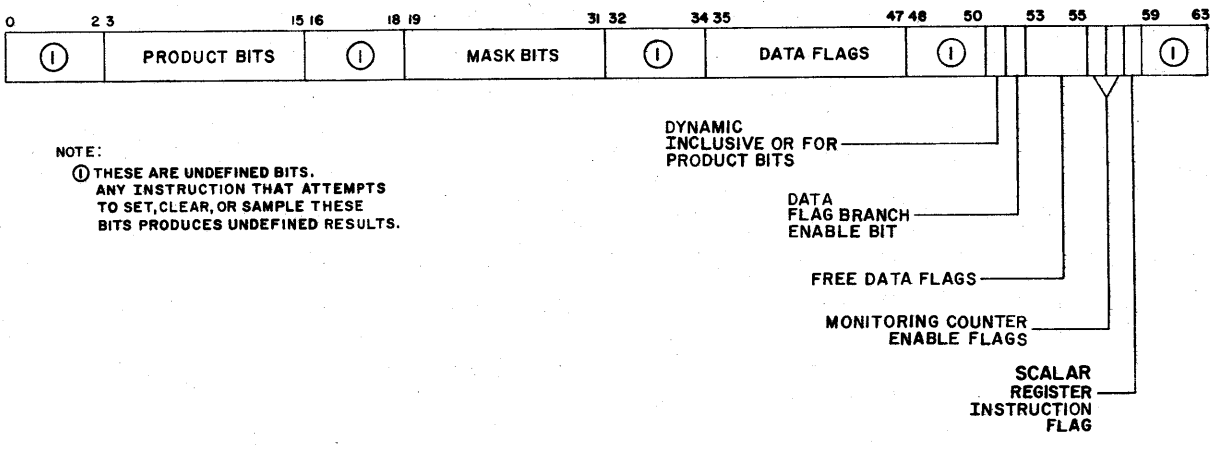


Figure 5-10. DFB Register Format

DATA FLAGS

Data flag bits are bits 35 through 47 of the DFB register. These bits indicate conditions that have occurred. For example, the machine sets bit 37 at the end of a search for masked key word (FF), byte (FE), or bit (D6) instruction if the operation detects no match. If a subsequent search for masked key instruction detects a match, the machine does not clear DFB bit 37. Bits 35 through 47 of the DFB register are cleared only by the data flag register bit branch and alter (32) and the data flag register load/store (3B) instructions.

Refer to appendix D for a complete list of data flag applications to instructions. Data flag bit 36 is applicable only to the job interval timer rather than a specific instruction and therefore not listed.

If a control vector (refer to Control Vector under Vector Instruction in section 6) is being used, the current control vector bit must be permissive for the operation to set any of the data flags. For example, if a divide fault occurs but the control vector bit for that result element is not permissive, that result element would not set the divide fault data flag bit.

Table 5-7 lists the data flag register bit assignments and associated mask and product bits described in the following paragraphs.

MASK BITS

The mask bits are bits 19 through 31 of the DFB register. They select the conditions that cause the automatic data flag branch to occur when the selected condition takes place.

The 33 or 3B instruction sets and clears the mask bits. A mask bit need not be set for its corresponding data flag bit to be set when the condition occurs. The mask bits enable the setting of a corresponding bit in the product field when the associated masked data flag bit is set. A product bit is set regardless of the order the mask bit and its associated data flag bit are set.

PRODUCT BITS

Products bits are bits 3 through 15 of the DFB register. Each is the dynamic logical product of a data flag bit and associated mask bit being set. The computer executes a data flag branch when there is at least one bit set in the product field and the data flag branch enable bit is set.

DYNAMIC INCLUSIVE OR FOR PRODUCT BITS

The dynamic inclusive OR for product bits is bit 51 of the DFB register. This bit is set by setting any one of the product bits. It cannot be cleared directly.

DATA FLAG BRANCH ENABLE BIT

The data flag branch enable bit is bit 52 of the DFB register. This bit must be set for an automatic data flag branch to take place. When bits 51 and 52 are both set, (setting may occur in either order) the data flag branch takes place at the end of the current instruction. The computer automatically clears bit 52 when a data flag branch takes place. The data flag register bit branch and alter or a data flag register load/store instruction resets the data flag branch enable bit which reenables the data flag branch operation.

FREE DATA FLAGS

Free data flag bits are bits 53 through 55 of the DFB register. Table 5-8 lists each of the free data flag bits and the corresponding assignments. There are no product or mask bits associated with these bits. Each of the bits are cleared automatically, unless the instruction is a no-operation (no-op), during the initial phase of the instruction which may set them. If applicable, these bits must be sampled before executing another instruction which would clear them. The setting of the bits does not cause a data flag branch operation.

MONITORING COUNTER ENABLE FLAGS

Monitoring counter enable flags are bits 56 and 57 of the DFB register. These flags enable the monitoring counters under certain conditions. (Refer to Count Gates in section 4.)

SCALAR REGISTER INSTRUCTION FLAG

The scalar register instruction flag is bit 58 of the DFB register. This flag indicates that one of the other data flags has been set by a scalar instruction. The flag is cleared by the 33 or 3B instructions.

TABLE 5-7. DATA FLAG REGISTER BIT ASSIGNMENTS

Product Bit	Mask Bit	Data Flag Bit	Assignment/Description
3	19	35	Soft interrupt: Monitor software can set bit 35 of a job's DFB register while the register is stored in the job's invisible package. If, after exchanging back to job mode, bit 35 and its corresponding mask bit are set, a normal data flag branch occurs following completion of the current instruction.
4	20	36	Job interval timer.
5	21	37	Selected condition not met. Search for masked key - no match. Count of nonzero translated bytes > $2^{16}-1$.
6	22	38†	Decimal data fault: A sign is found in a digit position or a digit is found in a sign position. If data flag bit 38 is set, DFB 39 is undefined.
7	23	39†	Leading nonzero digits have been truncated. Leading nonzero bits have been truncated. Divide by zero; E3 and E7. The binary result exceeds the range of $\pm 2^{47} - 1$.
8	24	40	Bit 40 is the inclusive OR of bits 37, 38, and 39. Bit 24 masks bit 40. Bit 8 is the logical product of bits 24 and 40.

† For those instructions which may set with data flag bit 38 and 39 (E4, E5, E6 and E7), the following is true. If both a data fault and nonfloating-point arithmetic overflow exist in the data, either one or the other or both of these flags are set, depending on the algorithm used in the particular machine.

TABLE 5-7. DATA FLAG REGISTER BIT ASSIGNMENTS (Contd)

Product Bit	Mask Bit	Data Flag Bit	Assignment / Description
9	25	41	Floating point divide fault: The divisor has an all zero coefficient or the divisor, as read from the register file or from central storage, is machine zero. If the divisor and/or dividend is indefinite, no divide fault exists. If a divisor causes a divide fault, the quotient is set to indefinite. The exponent overflow and result machine zero data flags are not set by a divide operation whose divisor caused a divide fault.
10	26	42	Exponent overflow: The exponent of the result is larger than 6FFF (6F for 32-bit arithmetic). Results are not checked for exponent overflow until after the exponent adjustment for normalization or significance has taken place. In the adjust exponent instructions, if a left shift exceeds the number of places required for normalization, this data flag bit is set. Exponent overflow causes the result to be set to indefinite; thus, the indefinite flag is always set on an exponent overflow. The exponent overflow data flag bit is not set if either source operand from central storage or the register file is indefinite or by a divide instruction whose division causes a divide fault.
11	27	43	Result machine zero: The exponent of the result returned to central storage or to the register file is less than 9000 (90 for 32-bit arithmetic). Exponent underflow causes the result to be set to machine zero. Results are not checked for exponent underflow until after the exponent adjustment for normalization is completed. This data flag bit is not set by a divide whose divisor causes a divide fault.
12	28	44	Bit 44 is the inclusive OR of bits 41, 42, and 43. Bit 28 masks bit 44. Bit 12 is the logical product of bits 28 and 44.

TABLE 5-7. DATA FLAG REGISTER BIT ASSIGNMENTS (Contd)

Product Bit	Mask Bit	Data Flag Bit	Assignment/Description
13	29	45	**Square** root result imaginary: A negative source operand was detected in a square root instruction. The square root of the absolute value of the operand is formed and the two's complement of this square root is stored as the result.
14	30	46	Indefinite result: An indefinite result was placed in central storage or into the register file. Bit 46 is also set if either or both operands of a floating point compare were indefinite. An indefinite result may be caused by one or both operands of a floating point arithmetic operation being indefinite or by the occurrence of either a divide fault or an exponent overflow.
15	31	47	Breakpoint: DFB bit 47 is set on the breakpoint instruction if breakpoint address and usage conditions are met. Applicable instruction: 04

TABLE 5-8. FREE DATA FLAG BIT ASSIGNMENTS

Free Data Flag Bit	Assignment	Applicable Instructions
53	Result field all zeros.	Logical string (F0 through F7)
54	Result field mixed.	
55	Result field all ones.	
53	Equal operands	String compares (E8, E9, and FD)
54	First operand high	
55	First operand low	
53	Last edited field is zero	Edit and mark (EB)
54	Last edited field nonzero with negative sign or unsigned (T flip-flop set)	
55	Last edited field nonzero with positive sign (T flip-flop clear)	
53	Termination due to length or delimiter rather than nonzero translated byte	Translate and test (EF)
54	Termination due to nonzero translated byte which is not the last data byte in the A field	
55	Termination due to nonzero translated byte which is the last data byte in the A field	

TABLE 5-8. FREE DATA FLAG BIT ASSIGNMENTS (Contd)

Free Data Flag Bit	Assignment	Applicable Instructions
53	Ones were counted	Count leading equals (1E)
54	Undefined	
55	Undefined	
53	Undefined	Maximum (D8)
54	Multiple hits	Minimum (D9)
55	Undefined	
53	Whole field scan, no hit	Scan right (19)
54	Undefined	Scan equal (28)
55	Undefined	Scan unequal (29)
53	All translated bytes are equal	Translate and mark (D7)
54	Undefined	
55	Undefined	
53	A byte plus B byte < G for all bytes	Modulo add (EC)
54	A byte plus B byte $\geq$ G for one or more but not all bytes	
55	A byte plus B byte $\geq$ G for all bytes	
53	A byte < B byte for all bytes	Modulo subtract (ED)
54	A byte $\geq$ B byte for one or more but not all bytes	
55	A byte $\geq$ B byte for all bytes	
53	No equal/unequal found	Scan equal (28)
54	Undefined	Scan unequal (29)
55	Undefined	

DATA FLAG BRANCH OPERATION

If a mask field bit and the associated data flag bit are set, the corresponding product field bit is set. Free data flag field bit 51 is also set since this bit is the dynamic inclusive OR of all bits in the product field. Under these conditions, the setting of bit 52 (data flag branch enable bit) initiates an automatic data flag branch operation.

The data flag branch operation begins at the termination of the instruction that caused the data flag branch condition. The execution of the data flag branch transfers the bit address of the next instruction into the rightmost 48 bits of register 01 of the register file. A branch takes place to the bit address in the rightmost 48 bits of register 02. The data flag branch operation automatically clears bit 52 at this time. The data flag branch also clears the leftmost 16 bits of register 01.

NOTE

The clearing of bit 52 disables the data flag branch operation. Caution must be used to ensure that all data branch conditions are eliminated before resetting bit 52 or the program may enter a tight loop operation. The sampling of bit 51 for a zero before setting bit 52 prevents this situation in all cases except those involving the job interval timer.

When using the job interval timer, the setting of DFB bit 36 occurs asynchronously with respect to instruction execution once the job interval timer is loaded. Thus, the timer may set bit 36 after the check of bit 51 and before the branch to the content of register 01.

This situation can be programmed by examining the content of register 01 upon entering the routine for processing data flag branches. If register 01 indicates that the branch occurred outside the DFB routine, the content of register 01 could be transferred to a temporary storage location.

If register 01 indicates that the branch occurred within the DFB routine, the content of register 01 would not be transferred to a temporary storage location. At the end of the DFB routine, the program would branch to the content of the temporary storage location.

A simpler method of programming the above condition is to combine the setting of bit 52 and the branch to the content of register 01 into a single 33 instruction (33603401).

DATA FLAG BRANCH TIMING CONSIDERATIONS

The automatic data flag branch (ADFB) can occur up to 35 instructions after the instruction which caused it. The point at which the branch occurs can vary between executions of the same program as a result of the asynchronous I/O activity affecting the load/store operations.

The following points pertain to the central computer use of the data flag register (DFR).

- The content of the DFR, as stored into the register file by a 3B instruction, reflects all previous activity on it. Also, activity prior to the 3B instruction does not affect the new contents of the DFR.
- ADFBs caused by a 3B instruction or any instruction previous to it may occur after the next one or two instructions, but no later.
- Sampling or altering a data flag bit with a 33 instruction may occur out of sequence with a previous pipeline instruction, up to 35 instructions earlier.
- If a 33 instruction alters a bit which causes an ADFB, the branch may occur up to two instructions later, even though all previous pipeline instructions may have finished. If the ADFB is contingent on the completion of a pipeline instruction, the ADFB may occur up to 35 instructions after the instruction which caused it.

When registers 1, 2, or 4 in the central computer register file are altered by an instruction, and this instruction is followed by an automatic data flag branch or illegal monitor mode instruction branch, the store operation may occur out of sequence with the branch operation. For example, if a 7E instruction loads register 4, and this instruction is followed by an illegal monitor mode instruction, the automatic branch is to the address specified by either the old or new contents of register 4, depending on the timing of the 7E and the instruction stream.

GENERAL DEFINITIONS AND PROGRAMMING GUIDES

The following paragraphs provide general definitions and guides to aid in the programming of the computer system.

OVERLAP OF OPERAND AND RESULT FIELDS

If (in instructions such as vector, string, etc.) the result field overlays a source field such that elements of the result are stored in the source field before elements in this portion of the source field are read, undefined results may occur. The source elements may be the original elements or they may be the newly-stored elements. In the latter case, the instruction results become undefined. Some instructions prohibit any overlap of source and destination fields. This restriction is included in the instruction descriptions.

ILLEGAL INSTRUCTIONS

Illegal instructions are those with function codes that are not part of the computer instruction set listed in the instruction list table in section 6. An illegal instruction, when used in job mode, causes an exchange to the monitor mode. Instruction execution then begins at the address specified by the content of the register file absolute register 3. An illegal instruction, when used in monitor mode, causes a branch to the register file absolute register 4. Instruction execution then begins at the address specified by the content of the register file absolute register 4.

INSTRUCTIONS WHICH CAUSE UNDEFINED RESULTS OR OPERATIONS

Instructions which contain unused bits must have those bits set to zero or instructions cause undefined results or operations. The unused bit areas of the instructions are shown with cross-hatched lines in the instruction word formats in section 6.

The job mode of operation protects memory from any undefined results or operations with the key-lock virtual addressing mechanism. This mechanism permits memory storage only to pages assigned to the current job for which the write lockout bits are not set.

The monitor mode of operation does not have the protection against undefined results or operations because it makes all memory references with absolute addresses.

String instructions E0 through E7 and EB use data flag bits 38 and/or 39 to indicate data fault and overflow conditions, respectively. During instruction exection, the contents of result field C is undefined for these instructions when data flag bit 38 or 39 is set.

ITEM COUNT

Item count is a term used in the instruction descriptions (section 6) to highlight the fact that certain instructions perform operations on a number of items. The term is general and refers to items which may be in bits, bytes, half-words, or words. Descriptions which use the term are those which specify instruction field lengths, offsets, indexes, and/or shift counts.

The size of the items in an item count is specified for applicable instructions in the instruction list tables (located near the front of section 6). The item size is listed under the table heading, number of bits in the operand. In an example from the tables (shown below), the operand size is 8 which indicates that the field lengths and indexes for the E1 instruction are expressed in bytes.

E1 3 8 ST Binary Sub; A-B-C

In another example (shown below), the operand is E. This indicates that the instuction uses 32-bit or 64-bit items, depending on the status of instruction bit 8 (G bit 0). An item count for a field length of this instruction means that the field contains 100 32-bit items or 100 64-bit items, depending on instruction bit 8.

80 1 E VT ADD U; A+B → C

When an item count (other than a field length) is contained in a 16-bit field, at least one sign bit must be present. Item counts in 16-bit fields are therefore limited to the range of $2^{15}-1$ to -2^{15}. (Refer to the following description of field length.) When an item count other than an index consists of 48 bits, the leftmost 33 bits of the item count must be identical sign bits. Sign bits must always be extended to the left to fill the 16-bit or 48-bit field that contains it.

FIELD LENGTH AND OFFSET

Vector, vector macro, sparse vector, logical string, and some nontypical instructions use a field length. An offset is used in vector, vector macro, and some nontypical instructions.

The field length as read from the register file before possible offset modification, is always interpreted as a positive number in the range of 0 to $2^{16}-1$ (65,535).

If a vector or other data field has no offset, the field is considered terminated before the reading of the first operand if the specified field length is zero.

Instructions having offsets must have 32 identical sign bits. The offsets are in the range -2^{16} to $2^{16}-1$. If the offset is not in this range, the operation of the instruction is undefined. The resulting field length after subtracting the offset from the field length (read from register A, B, or C) must be positive and less than $2^{16}-1$ or the field length is treated as zero.

INDEX

String, some branch, and some nontypical instructions use an index. The sign of an index may be either positive or negative. The maximum magnitude of an index depends on its use as defined in the instruction descriptions. The machine left shifts the indexes end-off zero, three, five, or six positions before the index is added to the base address. The number of positions shifted depends on whether the unit for the index is bits, bytes, half-words, or words, respectively.

DATA FAULT

A data fault occurs when a sign code is detected in an unexpected position of a packed binary coded decimal (BCD) number. A sign code in the leftmost four bits of any byte always produces a data fault. When only one BCD number is expected in a field, a sign code in any position other than the rightmost bits of the rightmost byte is a data fault. If a data fault is detected, the instruction operation is undefined.

OPERAND SIZE DEFINITIONS

Following is a listing of operand sizes which apply throughout this manual unless otherwise stated.

Word	A 64-bit quantity having the address of the leftmost bit always being a multiple of 64_{10}.
Half-word	A 32-bit quantity having the address of the leftmost bit always being a multiple of 32_{10}.
Byte	An 8-bit quantity having the address of the leftmost bit always being a multiple of 8_{10}.
Character	An 8-bit quantity, generally having some particular significance associated with the particular bit pattern or code.
Digit	A 4-bit binary coded decimal number or sign. In zoned format there is one digit per byte and in packed BCD format there are two digits per byte (refer to the string instructions description for more detail).
Sword	512 bits (or eight 64-bit words).

RESTRICTION ON SELF-MODIFYING PROGRAMS

It is difficult to use self-modifying programs properly in machines utilizing high-speed parallel architecture. Therefore, it is necessary to serialize the operation of the machine. This usually results in reduced performance.

Sophisticated methods requiring intimate familiarity with the machine can be utilized to execute self-modifying routines with less negative impact on performance. Guidelines are presented here to provide a basic method for satisfying most system requirements. The following operations must be performed in the order indicated.

1. Program modification must be performed only with the 13, 32, 5F, or 7F instructions.
2. An instruction must be executed which will guarantee that the former 13, 32, 5F, or 7F instruction is completed before the latter 03 instruction starts. One such instruction is the 3284XX01, where XX is any register containing a valid memory address.
3. An 03 instruction must follow the instruction given in step 2, and precede the modified code. This voids the instruction stack and initiates an out-of-stack branch.

RESULT VECTOR 64-SWORD LOOKAHEAD

The length of the result vector for the following instructions is input data dependent:

- Sparse vector (A0 through AF) and the compress (CF) instruction; the length of the result vector (C) depends on the number of 1 bits in the output order vector (Z).
- Compress (BC) instruction; the length of the result vector (C) depends on the number of 1 bits in the order vector (Z).
- Translate (D7, EE, F8, and F9) instructions where termination is on the input and the input is delimiter limited; the length of the result vector (C) depends on the position of the delimiter in the input field.

As the computer proceeds through the execution of the above instructions, it checks that an extra 64-sword page (small page) of result field is available if needed (64-sword lookahead). Therefore, it is necessary to provide one more small page for the result vector beyond the expected length.

For the sparse vector (A0 through AF) instructions, it is not necessary to provide an extra small page beyond the maximum possible result field length. The maximum possible length of result vector C is equal to the field length of output order vector Z.

INSTRUCTIONS 6

GENERAL

This section describes instruction word formats, instruction types, and instruction descriptions. The instruction word format description explains the content of 32-bit and 64-bit instruction formats used in the computer. The instruction type description explains the instruction groups according to the operations they perform. The instruction description gives detailed explanations and examples of individual instructions.

As an aid in finding instruction designator information and individual instruction descriptions, refer to:

- Table 6-1 for instruction designators.
- Table 6-2 or inside front cover for locating instructions by function code.
- Table 6-3 for locating instructions by instruction type.

INSTRUCTION WORD FORMATS

The 32-bit and 64-bit instruction words have 12 types of formats (figure 6-1). The formats have hexadecimal numbers, 1 through C, which are used as references in tables 6-2 and 6-3. The bits in the instruction word formats number from left to right, 0 through 31 or 0 through 63.

INSTRUCTION DESIGNATORS

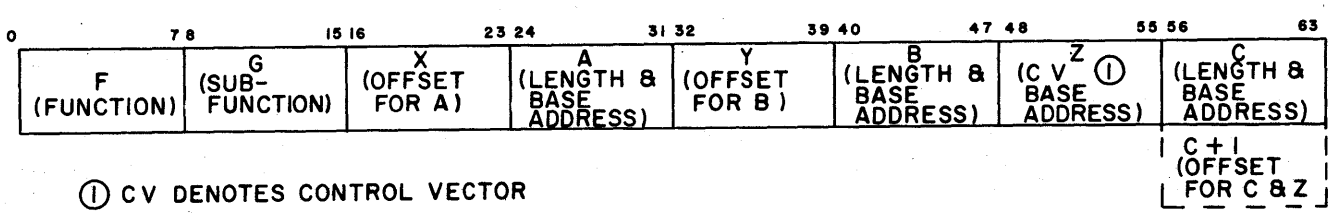
Each instruction word format is divided into bit groups that have assigned instruction designators shown in figure 6-1. The designator letters (such as F, R, S, and T in format 4) and their definitions are listed in table 6-1. The definitions are general and may vary between instructions. The instruction descriptions give more specific designator information as it applies to individual instructions.

When the C + 1 designator is used, the C designator must specify an even-numbered register. If the C designator specifies an odd-numbered register, the results of the instruction become undefined.

Bits 0 through 7 are commonly used by each instruction word as the function code designator (F). The computer uses function codes in the range of 00 through FF. The function codes in the range of 00 through 7F use 32-bit instruction word formats. The function codes in the range of 80 through FF use 64-bit instruction word formats.

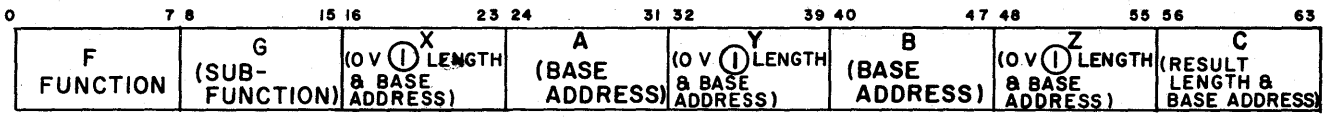
UNUSED BIT AREAS

Cross-hatched lines like those shown in formats A, B, and C of figure 6-1 indicate unused bit areas. These areas must be cleared to all zeros or the instructions will cause undefined results or operations.



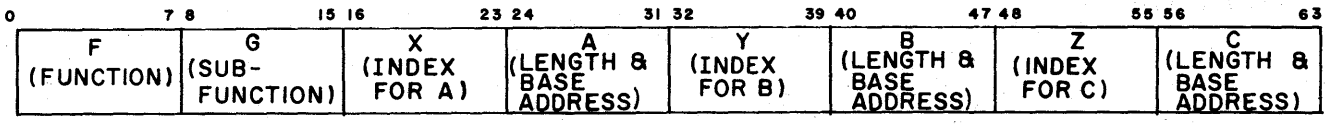
ⓘ C V DENOTES CONTROL VECTOR

FORMAT 1 - USED FOR VECTOR, VECTOR MACRO, AND SOME NONTYPICAL INSTRUCTIONS

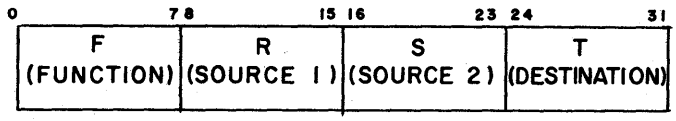


ⓘ O V DENOTES ORDER VECTOR

FORMAT 2 - USED FOR SPARSE VECTOR AND SOME NONTYPICAL INSTRUCTIONS

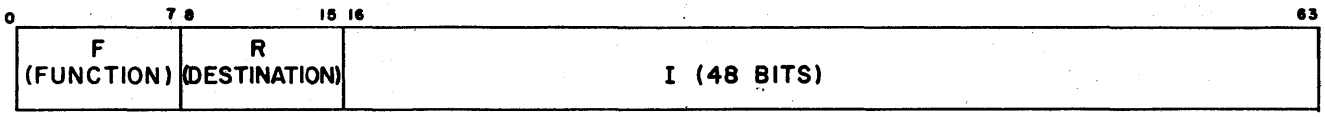


FORMAT 3 USED FOR LOGICAL STRING AND STRING INSTRUCTIONS

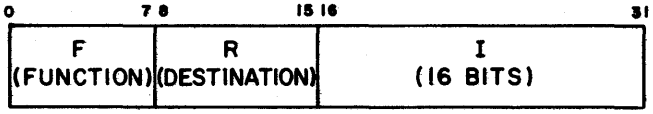


FORMAT 4 USED FOR SOME REGISTER, ALL MONITOR, THE 3D AND 04 NONTYPICAL INSTRUCTIONS

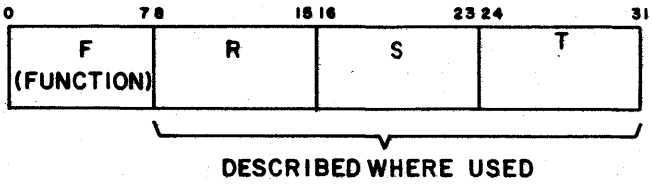
Figure 6-1. Instruction Formats



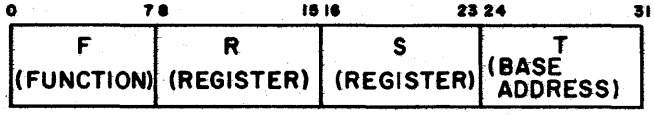
FORMAT 5 USED FOR THE BE, BF, CD, AND CE INDEX INSTRUCTIONS AND FOR THE B6 BRANCH INSTRUCTION



FORMAT 6 USED FOR THE 3E, 3F, 4D, AND 4E INDEX INSTRUCTIONS AND THE 2A REGISTER INSTRUCTION

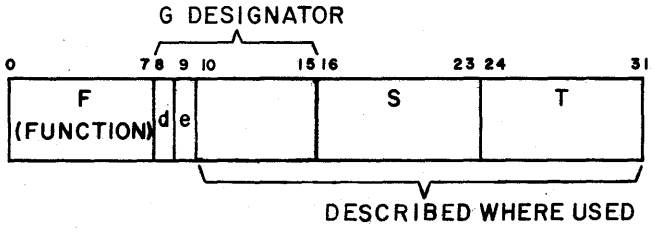


FORMAT 7 USED FOR SOME BRANCH AND NONTYPICAL INSTRUCTIONS

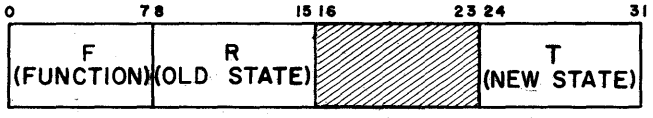


FORMAT 8 USED FOR SOME BRANCH INSTRUCTIONS

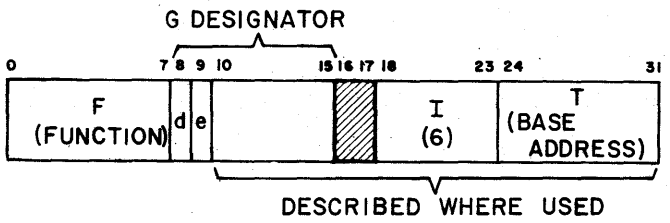
Figure 6-1. Instruction Formats (Contd)



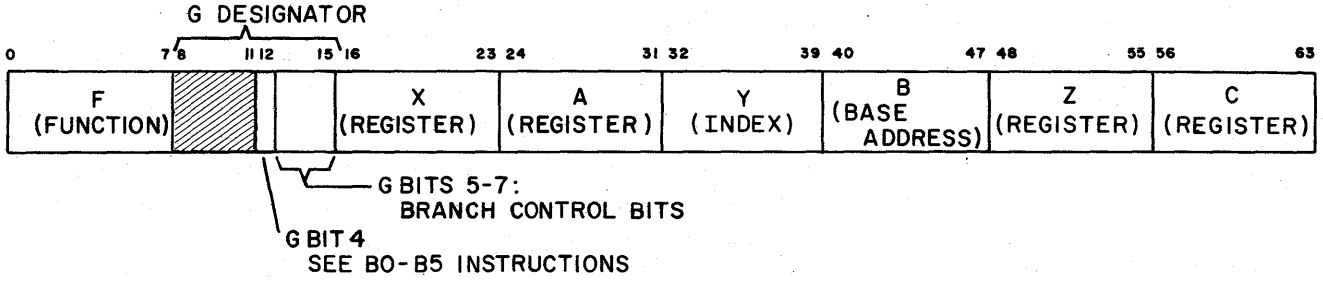
FORMAT 9 USED FOR THE 32 BRANCH INSTRUCTION



FORMAT A USED FOR SOME INDEX, BRANCH, AND REGISTER INSTRUCTIONS



FORMAT B USED FOR THE 33 BRANCH INSTRUCTION



FORMAT C. USED FOR THE BO – B5 BRANCH INSTRUCTIONS

Figure 6-1. Instruction Formats (Contd)

TABLE 6-1. INSTRUCTION DESIGNATORS

Designator	Format Type	Definition
A	1 & 3	This 8-bit designator specifies a register that contains a field length and base address for the corresponding source vector or string field.
	2	This 8-bit designator specifies a register that contains the base address for a source sparse vector field.
	C	Specifies a register that contains a two's complement integer in the rightmost 48 bits.
B	1 & 3	This 8-bit designator specifies a register that contains a field length and base address for the corresponding source vector or string field.
	2	This 8-bit designator specifies a register that contains the base address for a source sparse vector field.
	C	This 8-bit designator specifies a register that contains the branch base address in the rightmost 48 bits.
C	1, 2, &3	This 8-bit designator specifies a register that contains the field length and base address for storing the result vector, sparse vector, or string field.
	C	Specifies the register that contains the two's complement sum of (A) + (X) in the rightmost 48 bits. The leftmost 16 bits are cleared.
C+1	1	This 8-bit designator specifies a register that contains the offset for the C and Z vector fields.
d	9 & B	This 2-bit designator is contained within the G designator and specifies the branch conditions for the corresponding branch instructions.
e	9 & B	This 2-bit designator is contained within the G designator and specifies the object bit altering conditions for the corresponding branch instructions.

TABLE 6-1. INSTRUCTION DESIGNATORS (Contd)

Designator	Format Type	Definition
F	1 - C	This 8-bit designator is used in all instruction format types to specify the instruction function code. This designator is always contained in the leftmost eight bits of the instruction and is expressed in hexadecimal for all instruction descriptions. Thus, the function code range is $00\text{-}FF_{16}$. However, not all of the possible function codes are used.
G	1, 2, 3, 9, B, &C	This 8-bit designator specifies certain subfunction conditions for the corresponding instruction. The subfunctions include the length of the operands (32- or 64-bit), normal or broadcast source vectors, etc. The number of bits that are used in the G designator vary with individual instructions. (Appendix C lists the G bit usage codes according to function code.) The G designator bits have bit positions 8 through 15 in the word format. The manual references these bits as G bits 0 through 7. G bit 0 corresponds to bit position 8 in the word format. Other G bits follow, in order, from left to right.
I	5	This 48-bit designator functions as an index used to form the branch address in a B6 branch instruction. In the CD and CE index instructions, operand I is contained in the rightmost 24 bits. In the BE and BF index instructions, I is a 48-bit operand.
	6	In the 3E, 3F, 4D, and 4E index instructions, I functions as a 16-bit operand.
	B	In the 33 branch instruction, the 6-bit I designator specifies the number of the data flag branch register bit used in the branching operation.

TABLE 6-1. INSTRUCTION DESIGNATORS (Contd)

Designator	Format Type	Definition
R	4	This 8-bit designator specifies a register that contains an operand to be used in an arithmetic operation in the register and 3D instructions.
	5 & 6	In the BE, BF, CD, CE, 3E, 3F, 4D, and 4E index instructions, R functions as a destination register for the transfer of an operand or operand sum. In the B6 branch instruction, R specifies a register that contains an item count which is used to form the branch address.
	7, 8, & A	In these format types, R specifies registers and branching conditions that are described in the individual instruction descriptions.
S	4	This 8-bit designator specifies a register that contains an operand to be used in an arithmetic operation in the register and 3D instructions.
	7, 8, & 9	In these format types, S specifies registers and branching conditions that are described in the individual instruction descriptions.
T	4	This 8-bit designator specifies a destination register for the transfer of the arithmetic results.
	7, 8, 9, & B	In these formats, T specifies a register that contains the base address, and in some cases, the field length of the corresponding result field or branch address.
	A	In this format, T specifies a register that contains the old state of a register, data flag branch register, etc.; in an index, branch or interregister transfer operation.

TABLE 6-1. INSTRUCTION DESIGNATORS (Contd)

Designator	Format Type	Definition
X	1 & 3	This 8-bit designator specifies a register that contains the offset or index for vector or string source field A.
	2	In this case, X specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field A.
	C	In the B0 through B5 branch instructions, X specifies a register that contains a signed, two's complement integer in the rightmost 48 bits which is used as an operand in the branching operation.
Y	1 & 3	This 8-bit designator specifies a register that contains the offset or index for vector or string field B.
	2	In this format, Y specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field B.
	C	In the B0 through B5 branch instructions, Y specifies a register that contains an index that is used to form the branch address.
Z	1	This 8-bit designator specifies a register that contains the base address for the control vector used to control the result vector in field C.
	2	In this case, Z specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field C.
	3	In this format, Z specifies a register that contains the index for result field C.
	C	In the B0 through B5 branch instructions, Z specifies a register that contains a signed two's complement integer in the rightmost 48 bits. This integer is used as the comparison operand in determining whether the branch condition is met.

INSTRUCTION TYPES

The following 10 types of instructions are grouped according to the operations they perform.

- Index (IN)
- Register (RG)
- Branch (BR)
- Vector (VT)
- Sparse vector (SV)
- Vector macro (VM)
- String (ST)
- Logical string (LS)
- Nontypical (NT)
- Monitor (MN)

Table 6-2 lists each instruction code in the central computer instruction repertoire; the list is in the numerical order (hexadecimal) of the function code. Table 6-3 lists the instruction codes according to general type; the general types are in the same order as previously listed. The unused and illegal function codes are omitted from tables 6-2 and 6-3.

A page number is given for each instruction code in tables 6-2 and 6-3. These page numbers refer to the description of the corresponding instruction. Figure 6-2 provides additional explanations for using the tables.

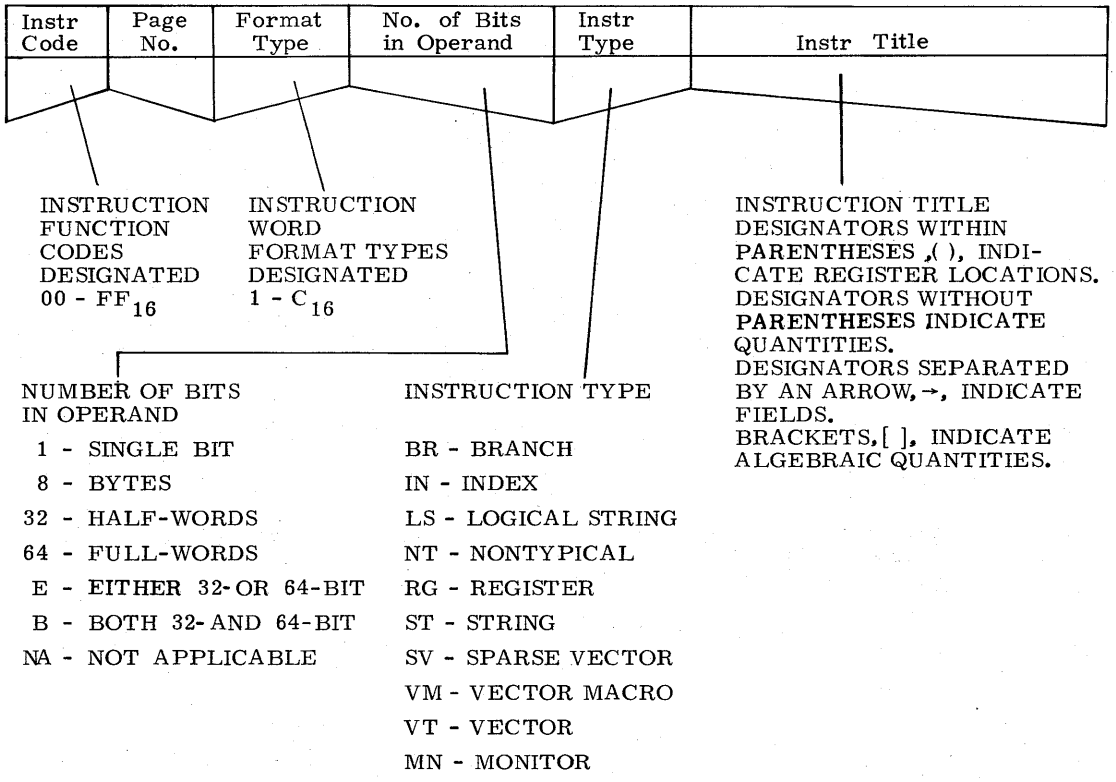


Figure 6-2. Instruction Listing Format

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
00	6-244	4	NA	MN	IDLE
03	6-241	4	64	NT	KEYPOINT - MAINTENANCE
04	6-241	4	64	NT	BREAKPOINT - MAINTENANCE
05	6-242.1	4	64	NT	VOID STACK AND BRANCH
06	6-243	7	NA	MN	FAULT TEST - MAINTENANCE
08	6-244	4	64	MN	INPUT/OUTPUT PER R
09	6-58	4	64	BR	EXIT FORCE
0A	6-247	4	64	MN	TRANSMIT (R) TO MONITOR INTERVAL TIMER
0C	6-245	4	64	MN	STORE ASSOCIATIVE REGISTERS
0D	6-245	4	64	MN	LOAD ASSOCIATIVE REGISTERS
0E	6-245	4	64	MN	TRANSLATE EXTERNAL INTERRUPT
0F	6-246	4	64	MN	LOAD KEYS FROM (R), TRANSLATE ADDRESS (S) TO (T)
10	6-42	A	64	RG	CONVERT BCD TO BINARY, FIXED LENGTH
11	6-42	A	64	RG	CONVERT BINARY TO BCD, FIXED LENGTH
12	6-196	7	64	NT	LOAD BYTE (T) PER (S), (R)
13	6-196	7	64	NT	STORE BYTE (T) PER (S), (R)
14	6-205	7	1	NT	BIT COMPRESS
15	6-207	7	1	NT	BIT MERGE
16	6-207	7	1	NT	BIT MASK
17	6-211	7	8	NT	CHARACTER STRING MERGE
18	6-231	7	8	NT	MOVE BYTES RIGHT
19	6-234	7	8	NT	SCAN RIGHT
1A	6-238	7	8	NT	FILL FIELD T WITH BYTE R
1B	6-238	7	8	NT	FILL FIELD T WITH BYTE (R)
1C	6-238	7	1	NT	FORM REPEATED BIT MASK WITH LEADING ZEROS
1D	6-238	7	1	NT	FORM REPEATED BIT MASK WITH LEADING ONES
1E	6-239	7	1	NT	COUNT LEADING EQUALS
1F	6-241	7	1	NT	COUNT ONES IN FIELD R, COUNT TO (T)
20	6-50	8	32	BR	BRANCH IF (R)=(S)(32 BIT FP)
21	6-50	8	32	BR	BRANCH IF (R)≠(S)(32 BIT FP)
22	6-50	8	32	BR	BRANCH IF (R)≥(S)(32 BIT FP)

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
23	6-50	8	32	BR	BRANCH IF (R)<(S)(32 BIT FP)
24	6-50	8	64	BR	BRANCH IF (R)=(S)(64 BIT FP)
25	6-50	8	64	BR	BRANCH IF (R)≠(S)(64 BIT FP)
26	6-50	8	64	BR	BRANCH IF (R)≥(S)(64 BIT FP)
27	6-50	8	64	BR	BRANCH IF (R)<(S)(64 BIT FP)
28	6-234	7	8	NT	SCAN EQUAL
29	6-234	7	8	NT	SCAN UNEQUAL
2A	6-48	6	64	RG	ENTER LENGTH OF (R) WITH I (16 BITS)
2B	6-48	4	64	RG	ADD TO LENGTH FIELD
2C	6-33	4	64	RG	LOGICAL EXCLUSIVE OR (R), (S), TO (T)
2D	6-33	4	64	RG	LOGICAL AND (R), (S) TO (T)
2E	6-33	4	64	RG	LOGICAL INCLUSIVE OR (R), (S), TO (T)
2F	6-51	9	1	BR	REGISTER BIT BRANCH AND ALTER
30	6-33	7	64	RG	SHIFT (R) PER S TO (T)
31	6-57	7	64	BR	INCREASE (R) AND BRANCH IF (R) ≠ 0
32	6-54	9	1	BR	BIT BRANCH AND ALTER
33	6-52	B	1	BR	DATA FLAG REGISTER BIT BRANCH AND ALTER
34	6-34	4	64	RG	SHIFT (R) PER (S) TO (T)
35	6-57	7	64	BR	DECREASE (R) AND BRANCH IF (R) ≠ 0
36	6-57	7	64	BR	BRANCH AND SET (R) TO NEXT INSTRUCTION
37	6-196	A	64	NT	TRANSMIT JOB INTERVAL TIMER TO (T)
38	6-32	A	64	IN	TRANSMIT (R BITS 00-15) TO (T BITS 00-15)
39	6-198	A	64	NT	TRANSMIT REAL-TIME CLOCK TO (T)
3A	6-198	A	64	NT	TRANSMIT (R) TO JOB INTERVAL TIMER
3B	6-54	A	64	BR	DATA FLAG REGISTER LOAD/ STORE
3C	6-195	4	32	NT	HALF WORD INDEX MULTIPLY (R)·(S) TO (T)
3D	6-195	4	64	NT	INDEX MULTIPLE (R)·(S) TO (T)
3E	6-30	6	64	IN	ENTER (R) WITH I (16 BITS)

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
3F	6-30	6	64	IN	INCREASE (R) BY I (16 BITS)
40	6-37	4	32	RG	ADD U; (R) + (S) TO (T)
41	6-37	4	32	RG	ADD L; (R) + (S) TO (T)
42	6-37	4	32	RG	ADD N; (R) + (S) TO (T)
44	6-37	4	32	RG	SUB U; (R) - (S) TO (T)
45	6-37	4	32	RG	SUB L; (R) - (S) TO (T)
46	6-37	4	32	RG	SUB N; (R) - (S) TO (T)
48	6-37	4	32	RG	MPY U; (R)·(S) TO (T)
49	6-37	4	32	RG	MPY L; (R)·(S) TO (T)
4B	6-37	4	32	RG	MPY S; (R)·(S) TO (T)
4C	6-37	4	32	RG	DIV U; (R)/(S) TO (T)
4D	6-30	6	32	IN	HALF WORD ENTER (R) WITH I (16 BITS)
4E	6-30	6	32	IN	HALF WORD INCREASE (R) BY I (16 BITS)
4F	6-37	4	32	RG	DIV S; (R) / (S) TO (T)
50	6-38	A	32	RG	TRUNCATE (R) TO (T)
51	6-38	A	32	RG	FLOOR (R) TO (T)
52	6-38	A	32	RG	CEILING (R) TO (T)
53	6-42	A	32	RG	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
54	6-47	4	32	RG	ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)
55	6-47	4	32	RG	ADJUST EXPONENT OF (R) PER (S) TO (T)
58	6-38	A	32	RG	TRANSMIT (R) TO (T)
59	6-38	A	32	RG	ABSOLUTE (R) TO (T)
5A	6-38	A	32	RG	EXPONENT OF (R) TO (T)
5B	6-41	4	32	RG	PACK (R), (S) TO (T)
5C	6-42	A	B	RG	EXTEND 32 BIT (R) TO 64 BIT (T)
5D	6-42	A	B	RG	INDEX EXTEND 32 BIT (R) TO 64 BIT (T)
5E	6-196	7	32	NT	LOAD (T) PER (S), (R)
5F	6-196	7	32	NT	STORE (T) PER (S), (R)
60	6-37	4	64	RG	ADD U; (R) + (S) TO (T)
61	6-37	4	64	RG	ADD L; (R) + (S) TO (T)

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
62	6-37	4	64	RG	ADD N; (R) + (S) TO (T)
63	6-38	4	64	RG	ADD ADDRESS (R) + (S) TO (T)
64	6-37	4	63	RG	SUB U; (R) - (S) TO (T)
65	6-37	4	64	RG	SUB L; (R) - (S) TO (T)
66	6-37	4	64	RG	SUB N; (R) - (S) TO (T)
67	6-38	4	64	RG	SUB ADDRESS (R) - (S) TO (T)
68	6-37	4	64	RG	MPY U; (R)·(S) TO (T)
69	6-37	4	64	RG	MPY L; (R)·(S) TO (T)
6B	6-37	4	64	RG	MPY S; (R)·(S) TO (T)
6C	6-37	4	64	RG	DIV U; (R) / (S) TO (T)
6D	6-35	4	64	RG	INSERT BITS FROM (R) TO (T) PER (S)
6E	6-36	4	64	RG	EXTRACT BITS FROM (R) TO (T) PER (S)
6F	6-37	4	64	RG	DIV S; (R) / (S) TO (T)
70	6-38	A	64	RG	TRUNCATE (R) TO (T)
71	6-38	A	64	RG	FLOOR (R) TO (T)
72	6-38	A	64	RG	CEILING (R) TO (T)
73	6-42	A	64	RG	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
74	6-47	4	64	RG	ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)
75	6-47	4	64	RG	ADJUST EXPONENT OF (R) PER (S) TO (T)
76	6-42	A	B	RG	CONTRACT 64 BIT (R) TO 32 BIT (T)
77	6-42	A	B	RG	ROUNDED CONTRACT 64 BIT (R) TO 32 BIT (T)
78	6-38	A	64	RG	TRANSMIT (R) TO (T)
79	6-38	A	64	RG	ABSOLUTE (R) TO (T)
7A	6-38	A	64	RG	EXPONENT OF (R) TO (T)
7B	6-41	4	64	RG	PACK (R),(S) TO (T)
7C	6-42	A	64	RG	LENGTH OF (R) TO (T)
7D	6-197	7	64	NT	SWAP S → T AND R → S
7E	6-196	7	64	NT	LOAD (T) PER (S), (R)
7F	6-196	7	64	NT	STORE (T) PER (S), (R)
80†	6-73	1	E	VT	ADD U; A + B → C
81†	6-73	1	E	VT	ADD L; A + B → C

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
82 †	6-73	1	E	VT	ADD N; A + B → C
83	6-74	1	64	VT	ADD A; A + B → C
84 †	6-73	1	E	VT	SUB U; A - B → C
85 †	6-73	1	E	VT	SUB L; A - B → C
86 †	6-73	1	E	VT	SUB N; A - B → C
87	6-74	1	64	VT	SUB A; A - B → C
88 †	6-73	1	E	VT	MPY U; A·B → C
89 †	6-73	1	E	VT	MPY L; A·B → C
8B †	6-73	1	E	VT	MPY S; A·B → C
8C †	6-73	1	E	VT	DIV U; A/B → C
8F †	6-73	1	E	VT	DIV S; A/B → C
90	6-75	1	E	VT	TRUNCATE A → C
91	6-75	1	E	VT	FLOOR A → C
92	6-75	1	E	VT	CEILING A → C
93 †	6-82	1	E	VT	SIGNIFICANT SQUARE ROOT OF A → C
94	6-86	1	E	VT	ADJUST SIGNIFICANCE OF A PER B → C
95	6-86	1	E	VT	ADJUST EXPONENT OF A PER B → C
96	6-82	1	B	VT	CONTRACT 64 BIT A → 32 BIT C
97	6-82	1	B	VT	ROUNDED CONTRACT 64 BIT A → 32 BIT C
98	6-75	1	E	VT	TRANSMIT A → C
99	6-75	1	E	VT	ABSOLUTE A → C
9A	6-75	1	E	VT	EXPONENT OF A → C
9B	6-80	1	E	VT	PACK A, B → C
9C	6-82	1	B	VT	EXTEND 32 BIT A → 64 BIT C
A0 †	6-94	2	E	SV	ADD U; A + B → C
A1 †	6-94	2	E	SV	ADD L; A + B → C
A2 †	6-94	2	E	SV	ADD N; A + B → C
A4 †	6-94	2	E	SV	SUB U; A - B → C
A5 †	6-94	2	E	SV	SUB L; A - B → C
A6 †	6-94	2	E	SV	SUB N; A - B → C
A8 †	6-98	2	E	SV	MPY U; A · B → C
A9 †	6-98	2	E	SV	MPY L; A · B → C
AB	6-98	2	E	SV	MPY S; A · B → C

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
AC†	6-98	2	E	SV	DIV U; A/B → C
AF†	6-98	2	E	SV	DIV S; A/B → C
B0	6-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) = (Z)
B1	6-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) ≠ (Z)
B2	6-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) ≥ (Z)
B3	6-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) < (Z)
B4	6-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) ≤ (Z)
B5	6-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) > (Z)
B0	6-62	C	64	BR	COMPARE FP, BRANCH IF (A) = (X)
B1	6-62	C	64	BR	COMPARE FP, BRANCH IF (A) ≠ (X)
B2	6-62	C	64	BR	COMPARE FP, BRANCH IF (A) ≥ (X)
B3	6-62	C	64	BR	COMPARE FP, BRANCH IF (A) < (X)
B4	6-62	C	64	BR	COMPARE FP, BRANCH IF (A) ≤ (X)
B5	6-62	C	64	BR	COMPARE FP, BRANCH IF (A) > (X)
B0	6-216	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) = (Z)
B1	6-216	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) ≠ (Z)
B2	6-216	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) ≥ (Z)
B3	6-216	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) < (Z)
B4	6-216	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) ≤ (Z)
B5	6-216	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) > (Z)

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
B0	6-217	C	64	NT	COMPARE FP, SET CONDITION (A) = (X)
B1	6-217	C	64	NT	COMPARE FP, SET CONDITION (A) ≠ (X)
B2	6-217	C	64	NT	COMPARE FP, SET CONDITION (A) ≥ (X)
B3	6-217	C	64	NT	COMPARE FP, SET CONDITION (A) < (X)
B4	6-217	C	64	NT	COMPARE FP, SET CONDITION (A) ≤ (X)
B5	6-217	C	64	NT	COMPARE FP, SET CONDITION (A) > (X)
B6	6-214	5	NA	BR	BRANCH TO IMMEDIATE ADDRESS (R) + I (48 BITS)
B7	6-122	1	E	VM	TRANSMIT LIST → INDEXED C
B8	6-111	1	E	VM	TRANSMIT REVERSE A → C
B9	6-226	1	E	NT	TRANSPOSE/MOVE
BA	6-119	1	E	VM	TRANSMIT INDEXED LIST → C
BB	6-198	2	E	NT	MASK A, B → C PER Z
BC	6-198	2	E	NT	COMPRESS A → C PER Z
BD	6-203	2	E	NT	MERGE A, B → C PER Z
BE	6-31	5	64	IN	ENTER (R) WITH I (48 BITS)
BF	6-31	5	64	IN	INCREASE (R) BY I (48 BITS)
C0	6-102	1	E	VM	SELECT EQ; A = B, ITEM COUNT TO (C)
C1	6-102	1	E	VM	SELECT NE; A ≠ B, ITEM COUNT TO (C)
C2	6-102	1	E	VM	SELECT GE; A ≥ B, ITEM COUNT TO (C)
C3	6-102	1	E	VM	SELECT LT; A < B, ITEM COUNT TO (C)
C4	6-218	1	E	NT	COMPARE EQ; A = B, ORDER VECTOR → Z
C5	6-218	1	E	NT	COMPARE NE; A ≠ B, ORDER VECTOR → Z
C6	6-218	1	E	NT	COMPARE GE; A ≥ B, ORDER VECTOR → Z
C7	6-218	1	E	NT	COMPARE LT; A < B, ORDER VECTOR → Z

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
C8	6-221	1	E	NT	SEARCH EQ; A = B, INDEX LIST → C
C9	6-221	1	E	NT	SEARCH NE; A ≠ B, INDEX LIST → C
CA	6-221	1	E	NT	SEARCH GE; A ≥ B, INDEX LIST → C
CB	6-221	1	E	NT	SEARCH LT; A < B, INDEX LIST → C
CD	6-31	5	32	IN	HALF WORD ENTER (R) WITH I (24 BITS)
CE	6-31	5	32	IN	HALF WORD INCREASE (R) BY I (24 BITS)
CF†	6-200	1	E	NT	ARITH. COMPRESS A → C PER B
D0	6-110	1	E	VM	AVERAGE $(A_n+B_n)/2 \rightarrow C_n$
D1	6-108	1	E	VM	ADJ. MEAN $(A_{n+1}A_n)/2 \rightarrow C_n$
D4	6-110	1	E	VM	AVE. DIFF. $(A_n-B_n)/2 \rightarrow C_n$
D5	6-108	1	E	VM	DELTA $(A_{n+1}-A_n) \rightarrow C_n$
D6††	6-165	3	1	ST	SEARCH FOR MASKED KEY; BIT, A, B PER C, G
D7†††	6-174	3	8	ST	TRANSLATE AND MARK A PER B → C
D8†	6-224	1	E	NT	MAX. OF A TO (C), ITEM COUNT TO (B)
D9†	6-224	1	E	NT	MIN. OF A TO (C), ITEM COUNT TO (B)
DA	6-105	1	E	VM	SUM $(A_0+A_1+A_2+\ldots A_n)$ TO (C) AND (C + 1)
DB	6-106	1	E	VM	PRODUCT $(A_0, A_1, A_2, \ldots A_n)$ TO (C)
DC	6-124	1	E	VM	VECTOR DOT PRODUCT TO (C) AND (C + 1)
DD	6-213	2	E	NT	SPARSE DOT PRODUCT TO (C) AND (C + 1)
DE	6-113	1	E	VM	POLYNOMINAL EVALUATION
DF	6-116	1	E	VM	INTERVAL A PER B → C

† These instructions have sign control capability.
†† Automatic index incrementing takes place on these instructions. (See the individual instruction descriptions.)
††† Delimiters may be used on these instructions, automatic index incrementing also takes place. (Refer to the individual instruction descriptions.)

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
E0	6-135	3	8	ST	BINARY ADD; A + B → C
E1	6-135	3	8	ST	BINARY SUB; A - B → C
E2	6-135	3	8	ST	BINARY MPY; A · B → C
E3	6-135	3	8	ST	BINARY DVD; A/B → C
E4	6-151	3	8	ST	DECIMAL ADD; A + B → C
E5	6-151	3	8	ST	DECIMAL SUB; A - B → C
E6	6-151	3	8	ST	DECIMAL MPY; A · B → C
E7	6-151	3	8	ST	DECIMAL DIV; A/B → C
E8	6-190	3	8	ST	COMPARE BINARY A, B
E9	6-190	3	8	ST	COMPARE DECIMAL A, B
EA	6-161	3	8	ST	MERGE PER BYTE MASK A, B PER G → C
EB	6-176	3	8	ST	EDIT AND MARK A PER B → C
EC	6-138	3	8	ST	MODULO ADD A + B → C
ED	6-138	3	8	ST	MODULO SUB A - B → C
EE †	6-170	3	8	ST	TRANSLATE A PER B → C
EF †	6-173	3	8	ST	TRANSLATE AND TEST A PER B TO C
F0	6-192	3	1	LS	LOGICAL EXCLUSIVE OR A, B → C
F1	6-192	3	1	LS	LOGICAL AND A, B → C
F2	6-192	3	1	LS	LOGICAL INCLUSIVE OR A, B → C
F3	6-192	3	1	LS	LOGICAL STROKE A, B → C
F4	6-192	3	1	LS	LOGICAL PIERCE A, B → C
F5	6-192	3	1	LS	LOGICAL IMPLICATION A, B → C
F6	6-192	3	1	LS	LOGICAL INHIBIT A, B → C
F7	6-192	3	1	LS	LOGICAL EQUIVALENCE A, B, → C
F8 †	6-158	3	8	ST	MOVE BYTES LEFT A → C
F9 †	6-158	3	8	ST	MOVE BYTES LEFT ONES COMP. A → C
FA	6-154	3	8	ST	MOVE AND SCALE; A → C
FB	6-140	3	8	ST	PACK ZONED TO BCD, A → C
FC	6-140	3	8	ST	UNPACK BCD TO ZONED; A → C
FD †	6-163	3	8	ST	COMPARE BYTES A, B PER MASK FIELD C

† Automatic index incrementing takes place on these instructions. (See the individual instruction descriptions.)

TABLE 6-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
FE †	6-165	3	8	ST	SEARCH FOR MASKED KEY BYTE; A, B PER C, G
FF †	6-165	3	64	ST	SEARCH FOR MASKED KEY WORD; A, B PER C, G

†Automatic index incrementing takes place on these instructions. (See the individual instruction descriptions.)

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
			INDEX INSTRUCTIONS (IN)	
3E	6-30	6	64	ENTER (R) WITH I (16 BITS)
3F	6-30	6	64	INCREASE (R) BY I (16 BITS)
4D	6-30	6	32	HALF WORD ENTER (R) WITH I (16 BITS)
4E	6-30	6	32	HALF WORD INCREASE (R) BY I (16 BITS)
CD	6-31	5	32	HALF WORD ENTER (R) WITH I (24 BITS)
CE	6-31	5	32	HALF WORD INCREASE (R) BY I (24 BITS)
BE	6-31	5	64	ENTER (R) WITH I (48 BITS)
BF	6-31	5	64	INCREASE (R) BY I (48 BITS)
38	6-32	A	64	TRANSMIT (R BITS 00-15) TO (T BITS 00-15)
			REGISTER INSTRUCTIONS (RG)	
2C	6-33	4	64	LOGICAL EXCLUSIVE OR(R),(S),TO(T)
2D	6-33	4	64	LOGICAL AND(R),(S),TO(T)
2E	6-33	4	64	LOGICAL INCLUSIVE OR(R),(S),TO(T)
30	6-33	7	64	SHIFT(R) PER S TO(T)
34	6-34	4	64	SHIFT(R) PER (S) TO (T)
6D	6-35	4	64	INSERT BITS FROM(R) TO(T) PER(S)
6E	6-36	4	64	EXTRACT BITS FROM(R) TO(T) PER(S)
40/60	6-37	4	32/64	ADD U; (R) + (S) TO (T)
41/61	6-37	4	32/64	ADD L; (R) + (S) TO (T)
42/62	6-37	4	32/64	ADD N; (R) + (S) TO (T)
44/64	6-37	4	32/64	SUB U; (R) - (S) TO (T)
45/65	6-37	4	32/64	SUB L; (R) - (S) TO (T)
46/66	6-37	4	32/64	SUB N; (R) - (S) TO (T)
48/68	6-37	4	32/64	MPY U; (R) · (S) TO (T)
49/69	6-37	4	32/64	MPY L; (R) · (S) TO (T)
4B/6B	6-37	4	32/64	MPY S; (R) · (S) TO (T)
4C/6C	6-37	4	32/64	DIV U; (R) / (S) TO (T)
4F/6F	6-37	4	32/64	DIV S; (R) / (S) TO (T)
63	6-38	4	64	ADD ADDRESS (R) + (S) TO (T)
67	6-38	4	64	SUB ADDRESS (R) - (S) TO (T)
58/78	6-38	A	32/64	TRANSMIT (R) TO (T)
59/79	6-38	A	32/64	ABSOLUTE (R) TO (T)
51/71	6-38	A	32/64	FLOOR (R) TO (T)

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
52/72	6-38	A	32/64	CEILING (R) TO (T)
5A/7A	6-38	A	32/64	EXPONENT OF (R) TO (T)
50/70	6-38	A	32/64	TRUNCATE (R) TO (T)
5B/7B	6-41	4	32/64	PACK (R), (S) TO (T)
5C	6-42	A	B	EXTEND 32 BIT (R) TO 64 BIT (T)
5D	6-42	A	B	INDEX EXTEND 32 BIT (R) TO 64 BIT (T)
76	6-42	A	B	CONTRACT 64 BIT (R) TO 32 BIT (T)
77	6-42	A	B	ROUNDED CONTRACT 64 BIT (R) TO 32 BIT (T)
7C	6-42	A	64	LENGTH OF (R) TO (T)
53/73	6-42	A	32/64	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
10	6-42	A	64	CONVERT BCD TO BINARY, FIXED LENGTH
11	6-42	A	64	CONVERT BINARY TO BCD, FIXED LENGTH
54/74	6-47	4	32/64	ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)
55/75	6-47	4	32/64	ADJUST EXPONENT OF (R) PER (S) TO (T)
2A	6-48	6	64	ENTER LENGTH OF (R) WITH I (16 BITS)
2B	6-48	4	64	ADD TO LENGTH FIELD
			BRANCH INSTRUCTIONS (BR)	
20/24	6-50	8	32/64	BRANCH IF(R)=(S)(32/64 BIT FP)
21/25	6-50	8	32/64	BRANCH IF(R)≠(S)(32/64 BIT FP)
22/26	6-50	8	32/64	BRANCH IF(R)≥(S)(32/64 BIT FP)
23/27	6-50	8	32/64	BRANCH IF(R)<(S)(32/64 BIT FP)
2F	6-51	9	1	REGISTER BIT BRANCH AND ALTER
33	6-52	B	1	DATA FLAG REGISTER BIT BRANCH AND ALTER
3B	6-54	A	64	DATA FLAG REGISTER LOAD/STORE
32	6-54	9	1	BIT BRANCH AND ALTER
36	6-57	7	64	BRANCH AND SET (R) TO NEXT INSTRUCTION
31	6-57	7	64	INCREASE (R) AND BRANCH IF (R) ≠ 0
35	6-57	7	64	DECREASE (R) AND BRANCH IF (R) ≠ 0
09	6-58	4	64	EXIT FORCE
B0	6-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) = (Z)

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
B1	6-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) $\neq$ (Z)
B2	6-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (Z) $\geq$ (Z)
B3	6-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) $<$ (Z)
B4	6-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) $\leq$ (Z)
B5	6-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) $>$ (Z)
B0	6-62	C	64	COMPARE FP, BRANCH IF (A) = (X)
B1	6-62	C	64	COMPARE FP, BRANCH IF (A) $\neq$ (X)
B2	6-62	C	64	COMPARE FP, BRANCH IF (A) $>$ (X)
B3	6-62	C	64	COMPARE FP, BRANCH IF (A) $\not<$ (X)
B4	6-62	C	64	COMPARE FP, BRANCH IF (A) $<$ (X)
B5	6-62	C	64	COMPARE FP, BRANCH IF (A) $\not>$ (X)
B6	6-64	5	NA	BRANCH TO IMMEDIATE ADDRESS (R) + I (48 BITS)
				VECTOR INSTRUCTIONS (VT)
80†	6-73	1	E	ADD U; A + B → C
81†	6-73	1	E	ADD L; A + B → C
82†	6-73	1	E	ADD N; A + B → C
84†	6-73	1	E	SUB U; A - B → C
85†	6-73	1	E	SUB L; A - B → C
86†	6-73	1	E	SUB N; A - B → C
88†	6-73	1	E	MPY U; A · B → C
89†	6-73	1	E	MPY L; A · B → C
8B†	6-73	1	E	MPY S; A · B → C
8C†	6-73	1	E	DIV U; A/B → C
8F†	6-73	1	E	DIV S; A/B → C
83	6-74	1	64	ADD A; A + B → C
87	6-74	1	64	SUB A; A - B → C
98	6-75	1	E	TRANSMIT A → C
99	6-75	1	E	ABSOLUTE A → C
91	6-75	1	E	FLOOR A → C
92	6-75	1	E	CEILING A → C
9A	6-75	1	E	EXPONENT OF A → C
90	6-75	1	E	TRUNCATE A → C
9B	6-80	1	E	PACK A, B → C
9C	6-82	1	B	EXTEND 32 BIT A → 64 BIT C
96	6-82	1	B	CONTRACT 64 BIT A → 32 BIT C
97	6-82	1	B	ROUNDED CONTRACT 64 BIT A → 32 BIT C
93†	6-82	1	E	SIGNIFICANT SQUARE ROOT OF A → C
94	6-86	1	E	ADJUST SIGNIFICANT OF A PER B → C
95	6-86	1	E	ADJUST EXPONENT OF A PER B → C

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
			SPARSE VECTOR INSTRUCTIONS (SV)	
A0†	6-94	2	E	ADD U; A + B → C
A1†	6-94	2	E	ADD L; A + B → C
A2†	6-94	2	E	ADD N; A + B → C
A4†	6-94	2	E	SUB U; A - B → C
A5†	6-94	2	E	SUB L; A - B → C
A6†	6-94	2	E	SUB N; A - B → C
A8†	6-98	2	E	MPY U; A · B → C
A9†	6-98	2	E	MPY L; A · B → C
AB†	6-98	2	E	MPY S; A · B → C
AC†	6-98	2	E	DIV U; A / B → C
AF†	6-98	2	E	DIV S; A / B → C
			VECTOR MACRO INSTRUCTIONS (VM)	
C0	6-102	1	E	SELECT EQ; A = B, ITEM COUNT TO(C)
C1	6-102	1	E	SELECT NE; A ≠ B, ITEM COUNT TO(C)
C2	6-102	1	E	SELECT GE; A ≥ B, ITEM COUNT TO(C)
C3	6-102	1	E	SELECT LT; A < B, ITEM COUNT TO(C)
DA	6-105	1	E	SUM $A_0 + A_1 + A_2 + \ldots A_n$) TO (C) AND(C + 1)
DB	6-106	1	E	PRODUCT (A_0, A_1, A_2,... A_n) TO (C)
D5	6-108	1	E	DELTA $\{ A_{n+1} - A_n \} \rightarrow C_n$
D1	6-108	1	E	ADJ. MEAN $\{ A_{n+1} + A_n \} / 2 \rightarrow C_n$
D0	6-110	1	E	AVERAGE $\{ A_n + B_n \} / 2 \rightarrow C_n$
D4	6-110	1	E	AVE. DIFF. $\{ A_n - B_n \} / 2 \rightarrow C_n$
B8	6-111	1	E	TRANSMIT REVERSE A → C
DE	6-113	1	E	POLYNOMIAL EVALUATION
DF	6-116	1	E	INTERVAL A PER B → C
BA	6-119	1	E	TRANSMIT INDEXED LIST → C
B7	6-122	1	E	TRANSMIT LIST → INDEXED C
DC	6-124	1	E	VECTOR DOT PRODUCT TO(C) AND(C + 1)

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
			STRING INSTRUCTIONS (ST)	
E0	6-135	3	8	BINARY ADD; A + B → C
E1	6-135	3	8	BINARY SUB; A - B → C
E2	6-135	3	8	BINARY MPY; A · B → C
E3	6-135	3	8	BINARY DVD; A / B → C
EC	6-138	3	8	MODULO ADD A + B → C
ED	6-138	3	8	MODULO SUB A - B → C
FB	6-140	3	8	PACK ZONED TO BCD; A → C
FC	6-140	3	8	UNPACK BCD TO ZONED; A → C
E4	6-151	3	8	DECIMAL ADD; A + B → C
E5	6-151	3	8	DECIMAL SUB; A - B → C
E6	6-151	3	8	DECIMAL MPY; A · B → C
E7	6-151	3	8	DECIMAL DVD; A / B → C
FA	6-154	3	8	MOVE AND SCALE; A → C
F8††	6-158	3	8	MOVE BYTES LEFT; A → C
F9††	6-158	3	8	MOVE BYTES LEFT, ONES COMP. A → C
EA	6-161	3	8	MERGE PER BYTE MASK A, B PER G → C
FD††	6-163	3	8	COMPARE BYTES A, B PER MASK FIELD C
FE†††	6-165	3	8	SEARCH FOR MASKED KEY BYTE; A, B PER C, G
FF†††	6-165	3	64	SEARCH FOR MASKED KEY WORD; A, B PER C, G
D6†††	6-165	3	1	SEARCH FOR MASKED KEY BIT; A, B PER C, G
EE††	6-170	3	8	TRANSLATE A PER B → C
EF††	6-173	3	8	TRANSLATE AND TEST PER B → C
D7††	6-174	3	8	TRANSLATE AND MARK A PER B → C
EB	6-176	3	8	EDIT AND MARK A PER B → C
E8	6-190	3	8	COMPARE BINARY A, B
E9	6-190	3	8	COMPARE DECIMAL A, B
			LOGICAL STRING INSTRUCTIONS (LS)	
F0	6-192	3	1	LOGICAL EXCLUSIVE OR A, B → C
F1	6-192	3	1	LOGICAL AND A, B → C
F2	6-192	3	1	LOGICAL INCLUSIVE OR A, B → C

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
F3	6-192	3	1	LOGICAL STROKE A, B → C
F4	6-192	3	1	LOGICAL PIERCE A, B → C
F5	6-192	3	1	LOGICAL IMPLICATION A, B → C
F6	6-192	3	1	LOGICAL INHIBIT A, B → C
F7	6-192	3	1	LOGICAL EQUIVALENCE A, B → C
			NONTYPICAL INSTRUCTIONS (NT)	
3D	6-195	4	64	INDEX MULTIPLY (R) · (S) TO (T)
3C	6-195	4	32	HALF WORD INDEX MULTIPLY (R) · (S) TO (T)
5E/7E	6-196	7	32	LOAD (T) PER (S), (R)
5F/7F	6-196	7	32	STORE (T) PER (S), (R)
12/13	6-196	7	64	LOAD/STORE BYTE (T) PER (S), (R)
37	6-196	A	64	TRANSMIT JOB INTERVAL TIMER TO (T)
7D	6-197	7	64	SWAP S→T and R→S
39	6-198	A	64	TRANSMIT REAL-TIME CLOCK TO (T)
3A	6-198	A	64	TRANSMIT (R) TO JOB INTERVAL TIMER
BB	6-198	2	E	MASK A, B → C PER Z
BC	6-199	2	E	COMPRESS A → C PER Z
CF†	6-200	1	E	ARITH. COMPRESS A → C PER B
BD	6-203	2	E	MERGE A, B → C PER Z
14	6-205	7	1	BIT COMPRESS
15	6-207	7	1	BIT MERGE
16	6-207	7	1	BIT MASK
17	6-211	7	8	CHARACTER STRING MERGE
DD	6-213	2	E	SPARSE DOT PRODUCT TO (C) AND (C + 1)
B0	6-216	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) = (Z)
B1	6-216	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) ≠ (Z)
B2	6-216	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) ≥ (Z)
B3	6-216	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) < (Z)
B4	6-216	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) ≤ (Z)

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
B5	6-216	C	64	COMPARE INTEGER, SET CONDITION (A) > (X)
B0	6-217	C	64	COMPARE FP, SET CONDITION (A) = (X)
B1	6-217	C	64	COMPARE FP, SET CONDITION (A) ≠ (X)
B2	6-217	C	64	COMPARE FP, SET CONDITION (A) ≥ (X)
B3	6-217	C	64	COMPARE FP, SET CONDITION (A) < (X)
B4	6-217	C	64	COMPARE FP, SET CONDITION (A) ≤ (X)
B5	6-217	C	64	COMPARE FP, SET CONDITION (A) > (X)
C4	6-218	1	E	COMPARE EQ; A = B, ORDER VECTOR → Z
C5	6-218	1	E	COMPARE NE; A ≠ B, ORDER VECTOR → Z
C6	6-218	1	E	COMPARE GE; A ≥ B, ORDER VECTOR → Z
C7	6-218	1	E	COMPARE LT; A < B, ORDER VECTOR → Z
C8	6-221	1	E	SEARCH EQ; A = B, INDEX LIST → C
C9	6-221	1	E	SEARCH NE; A ≠ B, INDEX LIST → C
CA	6-221	1	E	SEARCH GE; A ≥ B, INDEX LIST → C
CB	6-221	1	E	SEARCH LT; A < B, INDEX LIST → C
D8†	6-224	1	E	MAX. OF A TO (C) ITEM COUNT TO (B)
D9†	6-224	1	E	MIN. OF A TO (C) ITEM COUNT TO (B)
B9	6-226	1	E	TRANSPOSE/MOVE
18	6-231	7	8	MOVE BYTES RIGHT
19	6-234	7	8	SCAN RIGHT
28	6-234	7	8	SCAN EQUAL
29	6-234	7	8	SCAN UNEQUAL
1A	6-238	7	8	FILL FIELD T WITH BYTE R
1B	6-238	7	8	FILL FIELD T WITH BYTE (R)
1C	6-238	7	1	FORM REPEATED BIT MASK WITH LEADING ZEROS

TABLE 6-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
1D	6-238	7	1	FORM REPEATED BIT MASK WITH LEADING ONES
1E	6-239	7	1	COUNT LEADING EQUALS
1F	6-241	7	1	COUNT ONES IN FIELD R. COUNT TO (T)
03	6-241	4	64	KEYPOINT - MAINTENANCE
04	6-241	4	64	BREAKPOINT - MAINTENANCE
05	6-242.1	4	64	VOID STACK AND BRANCH
06	6-243	7	NA	FAULT TEST - MAINTENANCE
			MONITOR INSTRUCTIONS (MN)	
00	6-244	4	NA	IDLE
08	6-244	4	64	INPUT/OUTPUT PER R
0C	6-245	4	64	STORE ASSOCIATIVE REGISTERS
0D	6-245	4	64	LOAD ASSOCIATIVE REGISTERS
0E	6-245	4	64	TRANSLATE EXTERNAL INTERRUPT
0F	6-246	4	64	LOAD KEYS FROM (R), TRANSLATE ADDRESS (S) TO (T)
0A	6-247	4	64	TRANSMIT (R) TO MONITOR INTERVAL TIMER

† These instructions have sign control capability.
†† Delimiters may be used on these instructions, and automatic index incrementing also takes place. (Refer to the individual instruction descriptions.)
††† Automatic index incrementing takes place on these instructions. (Refer to the individual instruction descriptions.)

INSTRUCTION DESCRIPTIONS

The instruction descriptions are grouped in the following order.

- Index Instructions
- Register Instructions
- Branch Instructions
- Vector Instructions
- Sparse Vector Instructions
- Vector Macro Instructions
- String Instructions
- Logical String Instructions
- Nontypical Instructions
- Monitor Instructions

The description of each of the general types of instructions contains the instruction formats, operating parameters, and instruction termination conditions that are applicable to the instruction. The individual instructions within a general type are grouped according to the specific functions they perform within that group. Instructions that differ slightly in the functions they perform have a common description. For example, the index branch instructions (B0 through B5) differ only by the sign or magnitude of the branch quantity. Thus, these instructions have a common description.

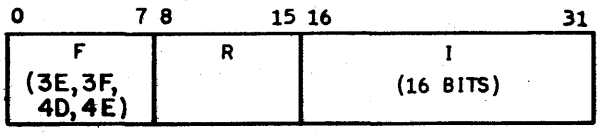
Each description begins with a listing of the function code (hexadecimal) and title of the instruction. This listing is followed by the instruction format. The formats specifically apply to the listed instructions and show the variations from the general format types shown in the beginning of this section.

Where applicable, the instruction descriptions include examples. These examples show a simplified illustration of the instruction operation using arbitrarily assumed operands, register contents, indexes, etc. The assumed operands and operating parameters are selected mainly to illustrate the instruction operation and are not necessarily typical operating values. The numbers used in the examples are in hexadecimal notation unless otherwise noted.

INDEX INSTRUCTIONS

The index instructions manipulate sixteen 24- or 48-bit operands in the designated operational registers. These instructions are used primarily in performing numerical calculations on field lengths and addresses.

3E ENTER (R) WITH I (16 BITS)
3F INCREASE (R) WITH I (16 BITS)
4D HALF WORD ENTER (R) WITH I (16 BITS)
4E HALF WORD INCREASE (R) BY I (16 BITS)



3E ENTER (R) WITH I (16 BITS)

This instruction enters the 16-bit operand I into the rightmost 48 bits of the 64-bit register designated by R. The sign bit of the immediate 16-bit operand is extended through bit 16 of the destination register R. Register R is cleared before the transfer of I.

3F INCREASE (R) WITH I (16 BITS)

This instruction replaces the rightmost 48 bits of the 64-bit register designated by R with the sum of these bits and the 16-bit operand I. The leftmost 16 bits of register R are unaltered. The sign bit of the immediate 16-bit operand is extended through bit 16 in the addition. Arithmetic overflow is ignored if it occurs.

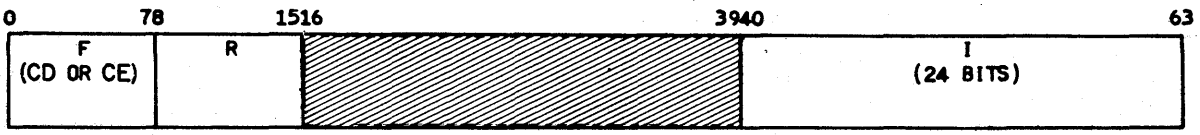
4D HALF WORD ENTER (R) WITH I (16 BITS)

This instruction enters the 16-bit operand I into the rightmost 24 bits of the 32-bit register designated by R. The sign of the immediate 16-bit operand is extended through bit 8 of the destination register R. Register R is cleared before the transfer of I.

4E HALF WORD INCREASE (R) BY I (16 BITS)

This instruction replaces the rightmost 24 bits of the 32-bit register designated by R with the sum of these bits and the 16-bit operand I. The leftmost 8 bits of register R are unaltered. The sign of the operand is extended through bit 8 for the addition. Arithmetic overflow is ignored if it occurs.

CD HALF WORD ENTER (R) WITH I (24 BITS)
CE HALF WORD INCREASE (R) WITH I (24 BITS)



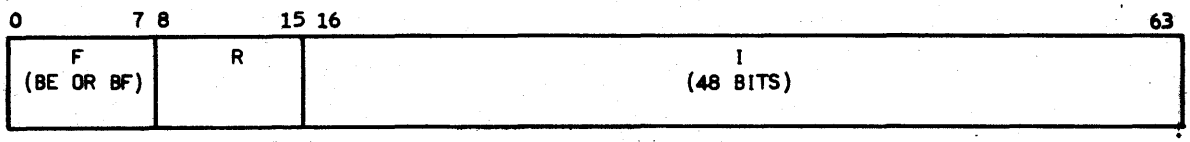
CD HALF WORD ENTER (R) WITH I (24 BITS)

This instruction clears the 32-bit register designated by R and enters the operand I, contained in the rightmost 24 bits of this instruction, into the rightmost 24 bits of register R.

CE HALF WORD INCREASE (R) WITH I (24 BITS)

This instruction replaces the rightmost 24 bits of the 32-bit register designated by R with the sum of these bits and operand I, contained in the rightmost 24 bits of this instruction. The leftmost 8 bits of register R are unaltered. Arithmetic overflow is ignored if it occurs.

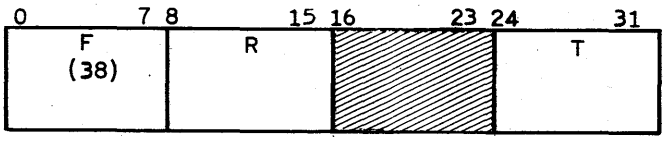
BE ENTER (R) WITH I (48 BITS)
BF INCREASE (R) WITH I (48 BITS)



The BE instruction enters the 48-bit operand I into the rightmost 48 bits of the R register. Register R is cleared before the transfer of I.

The BF instruction replaces the rightmost 48 bits of the R register with the sum of these bits and the 48-bit operand I. The leftmost 16 bits of R are unaltered. Arithmetic overflow is ignored.

38 TRANSMIT (R BITS 00-15) TO (T BITS 00-15)



This instruction replaces the leftmost 16 bits of register T with the leftmost 16 bits of register R. The remaining bits of register T are unaltered.

REGISTER INSTRUCTIONS

The source and result operands of register instructions are contained in specified registers in the register file. The 8-bit R, S, and T designators, contained in the instructions, denote the numbers of the registers to be used in the operation. For example, if a 64-bit, floating point, add upper instruction is executed (instruction code 60) with R = 02, S = 03, and T = 7F, the content of register 02 is added to the contents of register 03 (floating point format), and the upper result is stored in destination 7F.

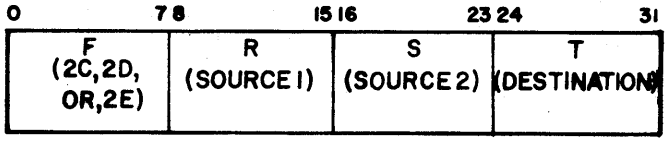
A register may contain one or both source operands as well as the result. Register 00 provides a special case. If this register is designated as containing the source operand, the instruction uses machine zero as the source operand (8X 000000 for 32-bit operands and 8XXX 000000 000000 for 64-bit operands where X represents any hexadecimal digit). If the instruction specifies 00 as the destination register, no result is stored. However, the instruction sets the corresponding data flags if applicable.

Unless the individual instruction description states differently, register-to-register operations do not change the content of the source registers. These operations clear the destination register before the result is transferred into it.

2C LOGICAL EXCLUSIVE OR (R),(S),TO (T)

2D LOGICAL AND (R),(S), TO (T)

2F LOGICAL INCLUSIVE OR (R),(S),TO (T)

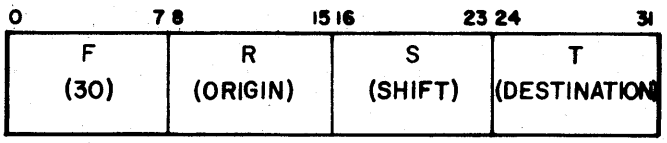


These instructions perform the following logical functions. The function occurs bit by bit on the 64-bit operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

R	S	Exclusive OR R-S	AND R•S	Inclusive OR R+S
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	1

If the R or S designators equal zero, register zero contains machine zero.

30 SHIFT (R) PER S TO (T)



This instruction shifts the 64-bit operand from the register designated by R and stores the result into the register designated by T. The S designator specifies the type and amount of the shift.

If the S designator is in the range from 0 through $3F_{16}$ (0 through 63_{10}), the operand from register R shifts left end-around the number of specified places and then stores in register T.

If the S designator is in the range from FF_{16} through $C1_{16}$ (-1 through -63_{10}), the operand from register R shifts right with sign extension and then stores into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the S designator.

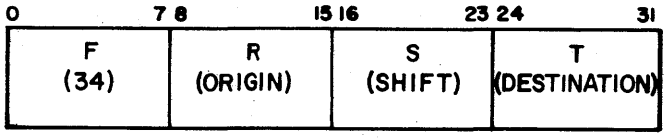
If, for example, S is equal to FE_{16}, the operand from register R shifts right two places.

If the S designator is greater than $3F_{16}$ or less than $C1_{16}$, the results of this instruction are undefined.

If the R designator is equal to zero, register zero provides machine zero.

This instruction does not test for machine zero, indefinite or does not set any data flags.

34 SHIFT (R) PER (S) TO (T)



This instruction shifts the 64-bit operand from the register designated by R and stores the result into the register designated by T. The register designated by S specifies the type and amount of the shift.

If the rightmost byte of register S is in the range from 0 through $3F_{16}$ (0 through 63_{10}), the operand from register R shifts left end-around the number of specified places and then stores into register T.

If the rightmost byte of register S is in the range from FF_{16} through $C1_{16}$ (-1 through -63_{10}), the operand from register R shifts right with sign extension and then stores into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the rightmost byte of register S.

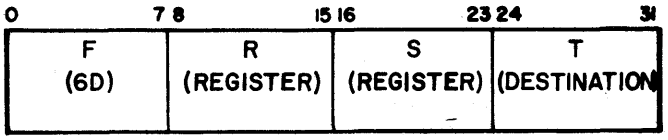
If the rightmost byte of register S is greater than $3F_{16}$ or less than $C1_{16}$, the results of this instruction are undefined.

The leftmost seven bytes of register S are ignored.

If the R designator is equal to zero, register zero provides machine zero.

This instruction does not cause a test for machine zero, indefinite or does not set any data flags.

6D INSERT BITS FROM (R) TO (T) PER (S)



This instruction inserts a number of rightmost bits (m) from the register designated R to the register designated T (figure 6-3). In the register designated S, bits 10 through 15 specify the number of bits (m) to be inserted, and bits 58 through 63 specify the location (n) in register T for the leftmost bit of the inserted bits. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zeros.

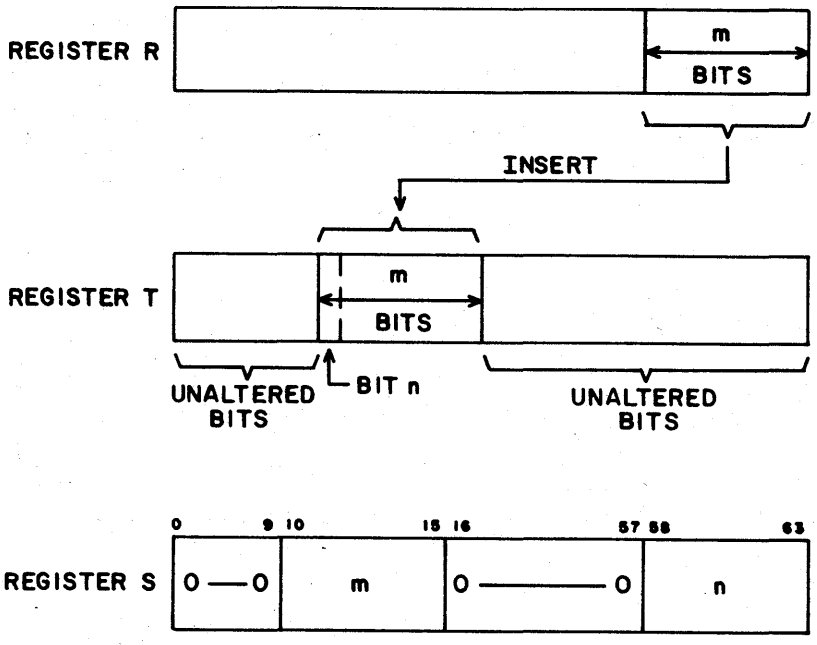
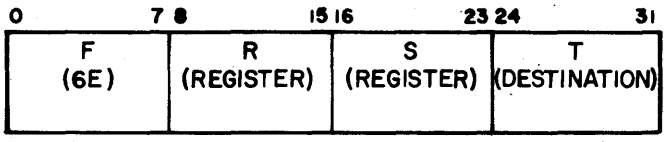


Figure 6-3. Example of Register Content for an Insert, Bits from (R) to (T) Per (S) Instruction

If the R designator is equal to zero, register zero provides machine zero. If m plus n is greater than 64_{10}, of if m is equal to zero, the results of this instruction are undefined. The maximum number of bits specified by m is 63_{10}.

6E EXTRACT BITS FROM (R) TO (T) PER (S)



This instruction extracts a number of bits (m) from the register designated R and stores them in the rightmost part of the register designated T (figure 6-4). Register T is cleared before receiving the extracted bits. In the register designated S, bits 10 through 15 contain the number of bits (m) to be extracted and bits 58 through 63 specify the leftmost bit number of the extracted bits in register R. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zeros.

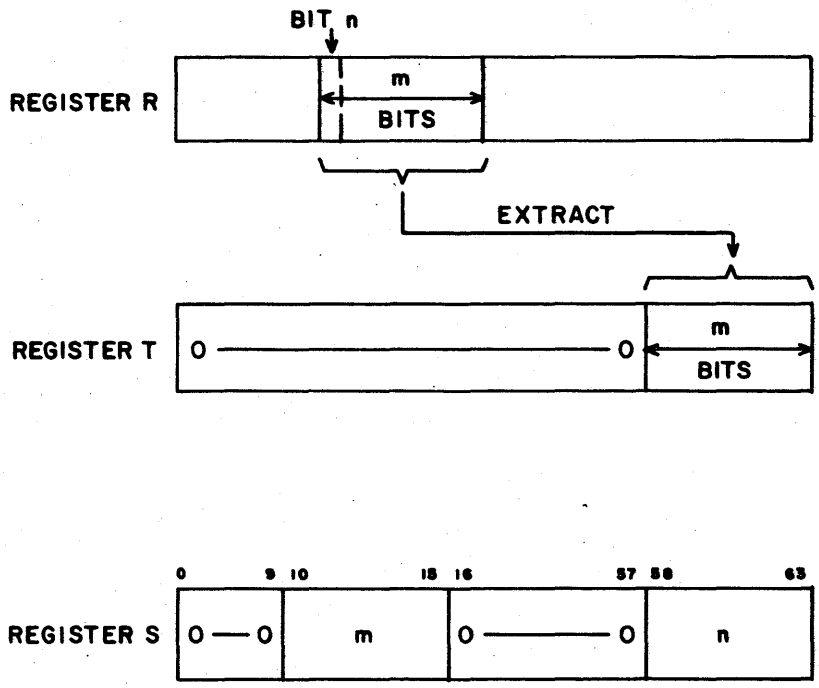
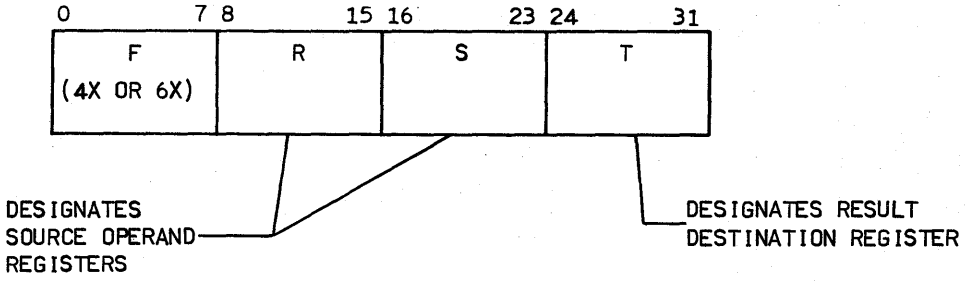


Figure 6-4. Example of Register Contents for an Extract, Bits from (R) to (T) Per (S) Instruction

If the R designator is equal to zero, register zero provides machine zero. If m plus n is greater than 64_{10}, of if m is equal to zero, the results of this instruction are undefined. The maximum number of bits specified by m is 63_{10}.

4C	ADD U; (R) + (S) TO (T)
41/61	ADD L; (R) + (S) TO (T)
42/62	ADD N; (R) + (S) TO (T)
44/64	SUB U; (R) – (S) TO (T)
45/65	SUB L; (R) – (S) TO (T)
46/66	SUB N; (R) – (S) TO (T)
48/68	MPY U; (R) • (S) TO (T)
49/69	MPY L; (R) • (S) TO (T)
4B/6B	MPY S; (R) • (S) TO (T)
4C/6C	DIV U; (R)/(S) TO (T)
4F/6F	DIV S; (R)/(S) TO (T)



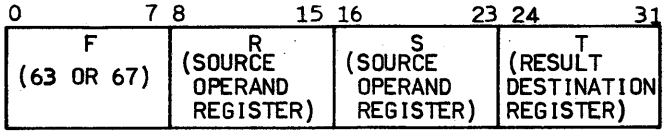
These instructions perform the indicated floating-point arithmetic operation on the 32-bit (4X function codes) or 64-bit (6X function codes) operands contained in the registers designated by R and S. Appendix B describes the floating-point operations and operand formats. This appendix also describes how certain instructions are order-dependent and will result in unexpected answers unless the execution order is known. An example is shown in the appendix under Order-Dependent Result Considerations. The arithmetic operation is the same for the 32-bit or 64-bit operands with adjustment for bit length of the result. The instruction, in each case, stores the arithmetic result in destination register T.

Designator U signifies that the upper result is stored, L signifies that the lower result is stored, N signifies that the normalized upper result is stored, and S signifies the significant result is stored. Appendix B of this manual defines the U, L, N, and S results.

Data flag bits 41 (floating-point divide fault), 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result) are set by the applicable instructions if the necessary operating and result conditions are present.

63 ADD ADDRESS (R) + (S) TO (T)

67 SUB ADDRESS (R) – (S) TO (T)



These instructions add/subtract bits 16 through 63 in register S to/from bits 16 through 63 in register R. The instructions then store the result in corresponding bits of register T. The instructions operate on bits 16 through 63 as 48-bit, positive, unsigned integers. Arithmetic overflow is ignored if it occurs. The instructions transmit bits 0 through 15 of register R to corresponding bit positions of register T without modification.

58/78 TRANSMIT (R) TO (T)

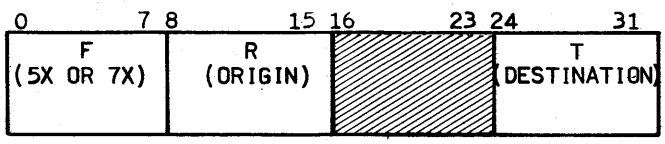
59/79 ABSOLUTE (R) TO (T)

51/71 FLOOR (R) TO (T)

52/72 CEILING (R) TO (T)

5A/7A EXPONENT OF (R) TO (T)

50/70 TRUNCATE (R) TO (T)



58/78 TRANSMIT (R) TO (T)

This instruction transmits the 32-bit (58) or 64-bit (78) operand in the register designated by R to the register designated by T.

59/79 ABSOLUTE (R) TO (T)

This instruction transmits the absolute value of the 32-bit (59) or 64-bit (79) floating-point operand in register R to register T. If the coefficient of the initial operand is negative, the operand is complemented and is transmitted to register T. If the initial coefficient is positive, it is sent to register T as it is. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

51/71 FLOOR (R) TO (T)

This instruction transmits the closest integer less than or equal to the 32-bit (51) or 64-bit (71) floating point operand in register R to register T. This integer (T) is expressed by an unnormalized 32-bit or 64-bit floating-point number with a positive exponent.

If the exponent of the source operand is positive (greater than or equal to zero), the operand is transmitted directly to register T. If the exponent of the source operand is negative, the machine right-shifts the coefficient end-off and increases the exponent by one for each shift. Sign bits are extended on the left during the shift. When the exponent becomes zero, the shifting stops and the machine transmits the shifted coefficient and zero exponent to register T. If machine zero is used as the source operand, 32/64 zeros are transmitted to register T.

The applicable data flag bit is 46 (indefinite result).

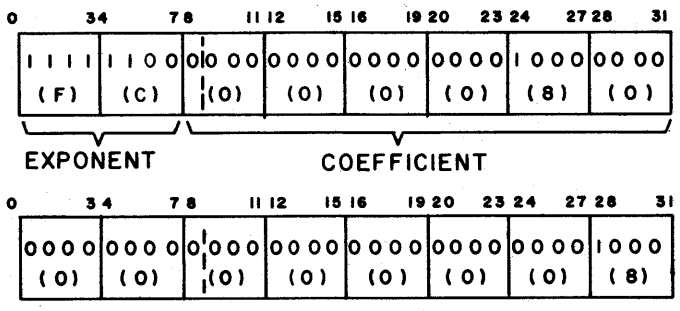
52/72 CEILING (R) TO (T)

This instruction transmits the closest integer greater than or equal to the 32-bit (64-bit for 72 function code) operand in origin register R to destination register T. This integer is represented as an unnormalized 32-bit (64-bit) floating point number with a positive exponent.

If the source operand exponent is positive (greater than or equal to zero), the instruction transmits the source operand directly to register T.

If the source operand exponent is negative, the machine right-shifts the two's complement of the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. The shift operation extends the sign. The instruction then recomplements the shifted coefficient and transmits it with zero exponent to register T. Figure 6-5 shows the results of a ceiling (R) to (T), 52/72, instruction with a source operand having a negative exponent. In this example, a shift of four was necessary to reduce the exponent to zero. The example shows the complement of the shifted coefficient with zero exponent in register T.

If machine zero is used as the source operand, the machine transmits 32/64 zeros as a result. The applicable data flag bit is 46 (indefinite result).



NUMBERS IN PARENTHESES REPRESENT HEXADECIMAL DIGITS FOR EACH BINARY GROUP.

Figure 6-5. Example of Register Content for a Ceiling (R) to (T) Instruction

5A/7A EXPONENT OF (R) TO (T)

This instruction transmits the exponent in the leftmost 8 bits (16 bits for 64-bit operands) of register R to the rightmost 8 bits (16 bits for 64-bit operands) of register T. The instruction extends the sign of the exponent through bit 8 of register T. The exponent portion (leftmost 8 or 16 bits) of register T is cleared.

50/70 TRUNCATE (R) TO (T)

This instruction transmits the closest integer the magnitude of which is less than or equal to the 32-bit (64-bit for 70 function code) operand in origin register specified by R to destination register T. This integer is represented by an unnormalized 32-bit (64-bit) floating point number with a positive exponent.

If the origin operand exponent is positive (greater than or equal to zero), the instruction transmits the origin operand directly to register T.

If the origin operand exponent is negative, the machine right-shifts the magnitude of the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. The operation extends zeros on the left during the shift. If the coefficient of the origin operand was positive, the shifted coefficient with zero exponent is transmitted to the destination register. If the coefficient of the origin operand was negative, the two's complement of the shifted coefficient and zero exponent is transmitted to the destination register. If machine zero is used as the origin operand, 32/64 zeros are transmitted as a result.

Figure 6-6 shows the results of a truncate (R) to (T), 50/70, instruction with an origin operand having a negative exponent and positive coefficient. A right shift of eight is required to reduce the negative exponent to zero.

The applicable data flag bit is 46 (indefinite result).

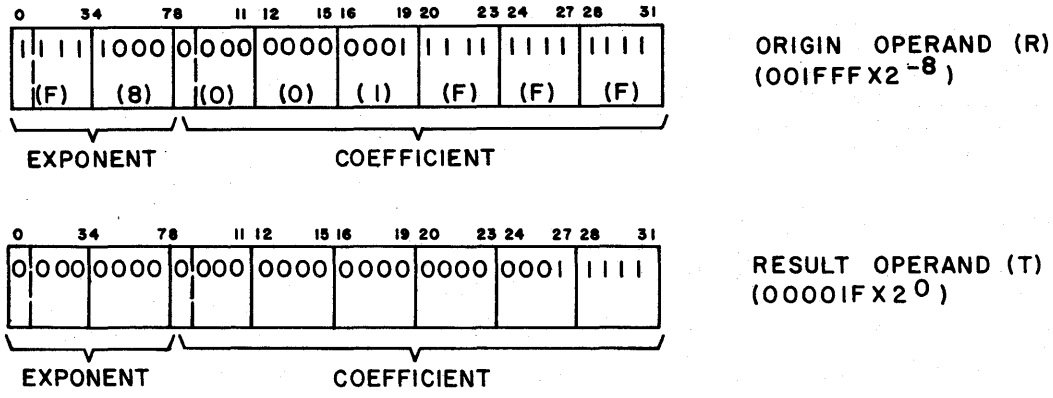
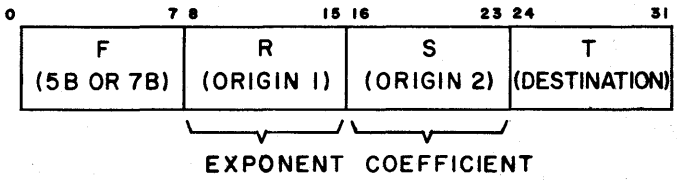


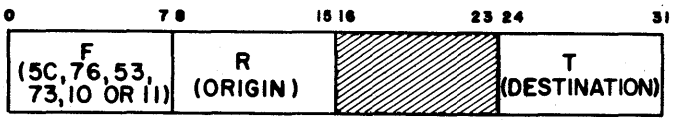
Figure 6-6. Example of Register Content for a Truncate (R) to (T) Instruction

5B/7B PACK (R), (S) TO (T)



This instruction transmits a 32-bit (64-bit for the 7B function code) floating-point number to the destination register T. The instruction transmits the exponent of the number from the rightmost 8 bits (16 bits for 7B) of register R and the coefficient from the rightmost 24 bits (48 bits for 7B) of register S.

5C	EXTEND 32 BIT (R) TO 64 BIT (T)
5D	INDEX EXTEND 32 BIT (R)TO 64 BIT (T)
76	CONTRACT 64 BIT (R) TO 32 BIT (T)
77	ROUNDED CONTRACT 64 BIT (R) TO 32 BIT (T)
7C	LENGTH OF (R) TO (T)
53/73	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
10	CONVERT BCD TO BINARY, FIXED LENGTH
11	CONVERT BINARY TO BCD, FIXED LENGTH



5C EXTEND 32 BIT (R) TO 64 BIT (T)

This instruction extends the 32-bit floating point number from register R into a 64-bit floating point number and transmits the result to 64-bit register T (figure 6-7). The value of the resulting exponent is 24_{10} less than the exponent of the origin operand. The result coefficient results from the transmission of the origin coefficient to bits 16 through 39 of register T. The instruction clears the rightmost 24 bits of the destination register.

If the contents of register R is indefinite, the result in register T is also indefinite and data flag bit 46 (indefinite result) is set. If the contents of register R is machine zero, register T contains machine zero, and data flag bit 43 (result machine zero) is set.

5D INDEX EXTEND 32 BIT (R) TO 64 BIT (T)

This instruction extends the 32-bit floating point number from register R into a 64-bit floating point number and transmits the result to 64-bit register T. The value of the resulting 16-bit exponent is the same as the origin operand's exponent with the sign bit extended through bit 0 of the result exponent.

The result coefficient results from the transmission of the rightmost 24 bits of the origin register into bits 40 through 63 of the destination register. Bits 16 through 39 of the destination register are set to the sign of the origin coefficient.

If the contents of register R is indefinite, the result in register T is also indefinite and data flag bit 46 (indefinite result) is set. If the contents of register R is machine zero, register T contains machine zero and data flag bit 43 (result machine zero) is set.

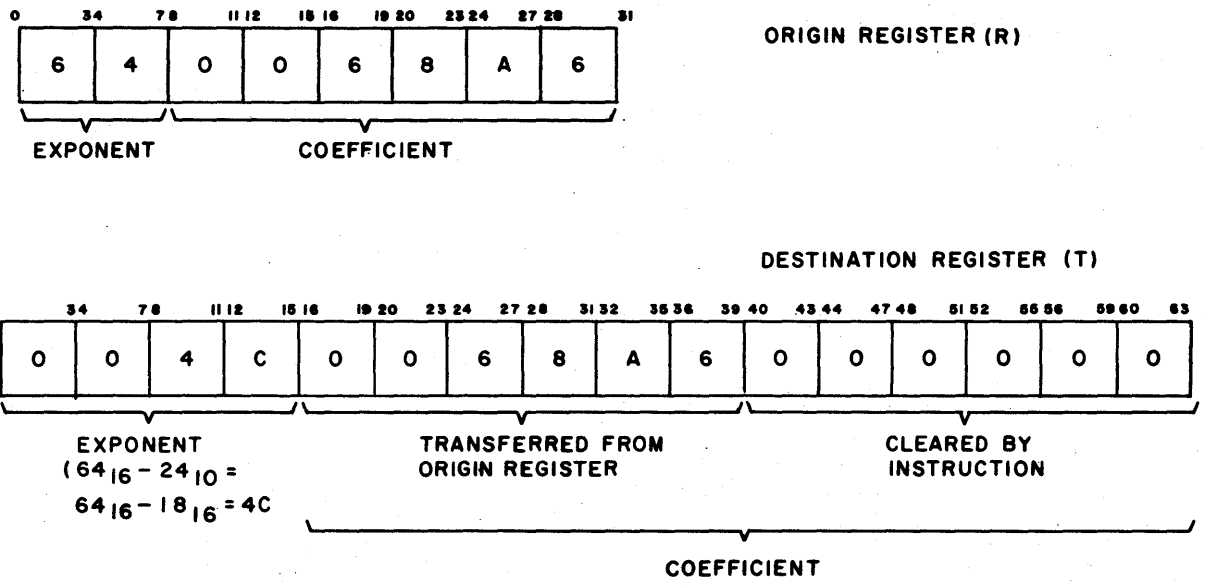


Figure 6-7. Example of Register Content for an Extend 32-Bit (R) to 64-Bit (T) Instruction

76 CONTRACT 64-BIT (R) TO-32 BIT (T)

This instruction (figure 6-8) contracts the 64-bit floating point number from register R into a 32-bit floating-point number. The instruction then transmits the result to a 32-bit register designated by T. The resulting 8-bit exponent represents the sum of the least-significant eight bits of the origin exponent and 24_{10}. If the result exponent cannot be contained in eight bits, exponent overflow or underflow is detected.

The following input exponent conditions are listed with the corresponding results of the 76 instruction execution.

Input Exponent	Result
7FFF	Result indefinite
.	
.	
7000	Indefinite data flag bit 46 (indefinite result) is set.
6FFF	Result indefinite
.	
.	
0058	Data flag bits 42 (exponent overflow) and 46 (indefinite result) are set.
0057	Result exponent is 24_{10} larger than the input exponent. The leftmost 24 bits of the input coefficient are transferred.
.	
.	
FF78	
FF77	Result is machine zero. Data flag bit 43 (result machine zero) is set.
.	
.	
8000	

Bits 16 through 39 of the origin are transmitted directly to the rightmost 24 bits of register T as the result coefficient. This operation contracts all source operands having a negative coefficient with an absolute value of less than 2^{24} to -1 (figure 6-8) and positive coefficients with an absolute value of less than 2^{24} to zero.

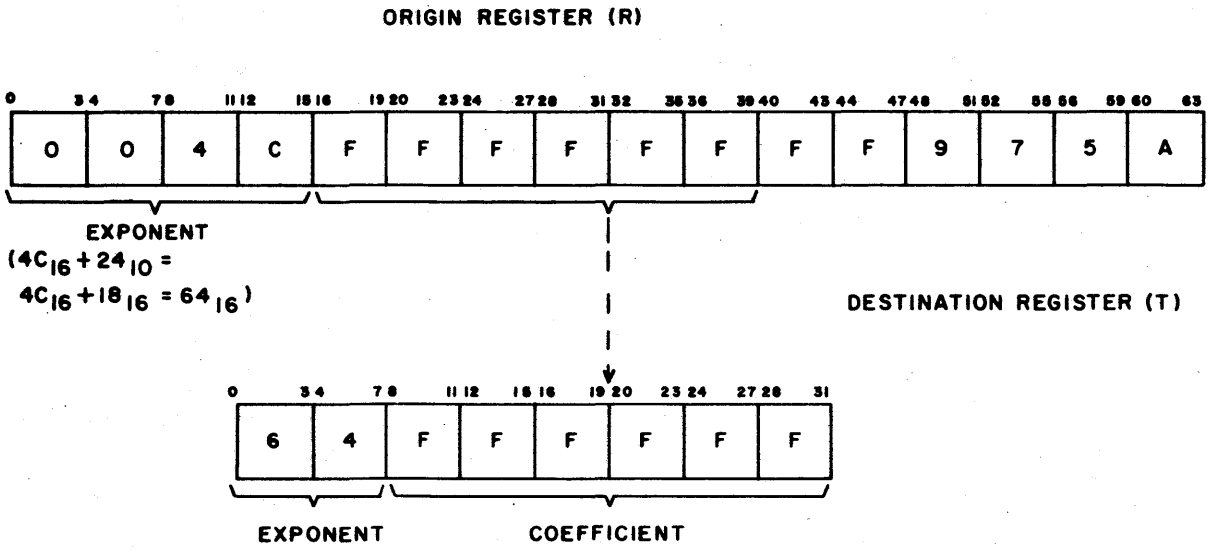


Figure 6-8. Example of Register Content for a Contract 64-Bit (R) to 32-Bit (T) Instruction

77 ROUNDED CONTRACT 64 BIT (R) TO 32 BIT (T)

This instruction performs a rounded contract operation on the 64-bit, floating-point operand in origin register R and transmits the 32-bit floating point result to destination register T (figure 6-9). The resulting 8-bit exponent represents the sum of the least-significant eight bits of the origin exponent and 24_{10}. If the result exponent cannot be contained in eight bits, exponent overflow or underflow is detected. The instruction then adds a +1 to bit position 40 of the origin operand and coefficient. If overflow occurs, the instruction increases the exponent by one and right-shifts the coefficient one place. The leftmost 24 bits of the shifted result coefficient are transmitted to the corresponding bits of the destination register. The 8-bit exponent of each nonend case result element is 24_{10} (25_{10} if overflow occurred) greater than the exponent of the corresponding source element.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

7C LENGTH OF (R) TO (T)

This instruction transmits the leftmost 16 bits of origin register R to the rightmost 16-bit positions of destination register T. The leftmost 48 bits of register T are cleared.

53/73 SIGNIFICANT SQUARE ROOT OF (R) TO (T)

This instruction transmits the square root† of a 32-bit (53 function code) or 64-bit (73 function code) operand in register R to register T. The result contains the same number of significant bits as the source operand. Applicable data flag bits are 45 (square root result imaginary), 46 (indefinite result), and 43(result machine zero).

10 CONVERT BCD TO BINARY, FIXED LENGTH

This instruction converts the packed BCD number in register R to a signed (two's complement) binary number and transfers the result to the rightmost 48 bits of register T. Figure 6-10 shows an example of the register contents following a convert BCD to binary, fixed length instruction. The leftmost 16 bits of register T are cleared by this instruction. The conversion is undefined for binary results greater than $+(2^{47}-1)$ or less than $-(2^{47}-1)$. Thus, the largest decimal number that this instruction can convert is ± 140, 737, 488, 355, 327. The instruction sets data flag bit 39 (refer to data flag register bit assignments in section 5) for numbers outside this range.

If the input number is not a valid BCD number, the results are undefined.

11 CONVERT BINARY TO BCD, FIXED LENGTH

This instruction converts the rightmost 48 bits (two's complement, binary number) of register R to a packed BCD number and transfers the result to register T. The result is a number containing 15 packed BCD digits (four bits per digit and the sign in bits 60 through 63). Figure 6-10 shows the packed BCD format; the binary range is $\pm (2^{47}-1)$.

† Appendix B describes the floating-point square root operation.

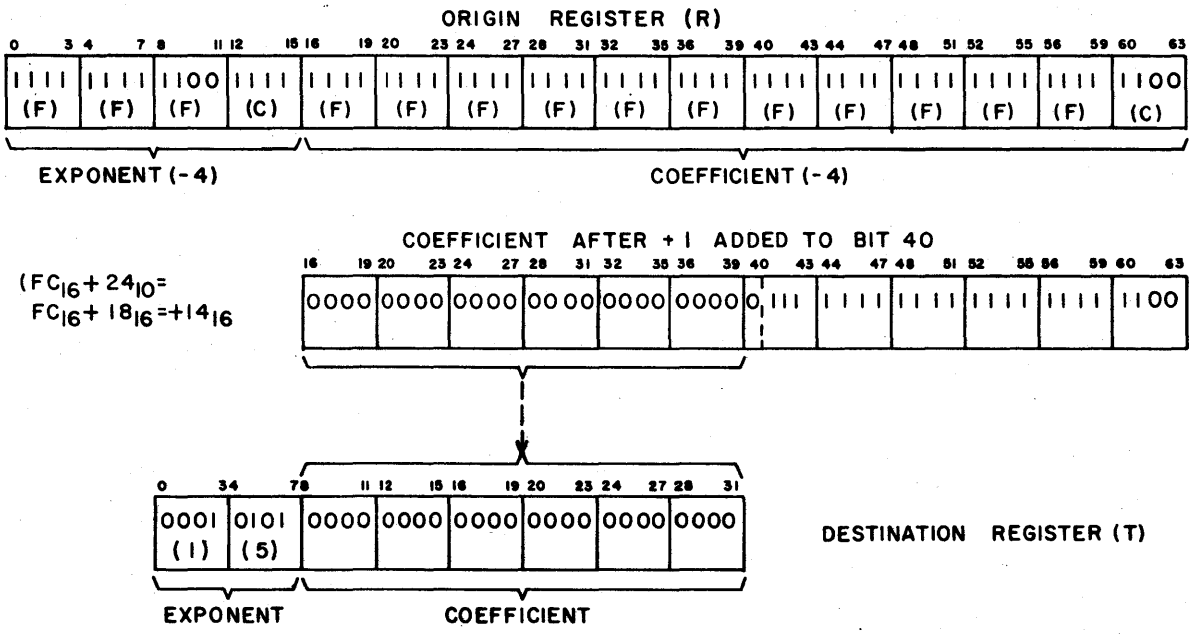


Figure 6-9. Example of Register Content for a Rounded Contract 64-Bit (R) to 32-Bit (T) Instruction

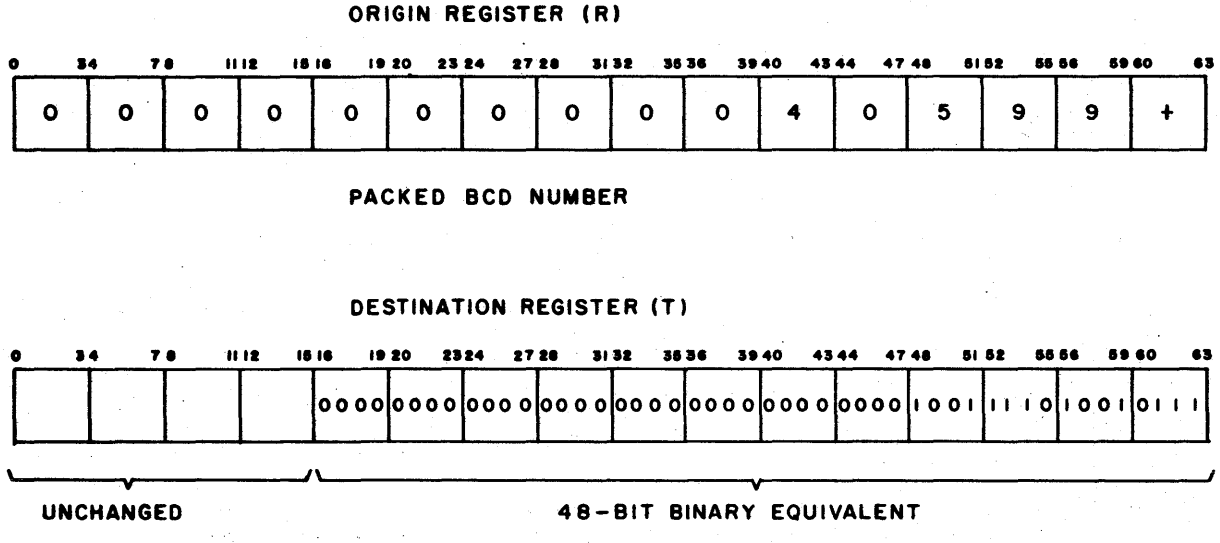
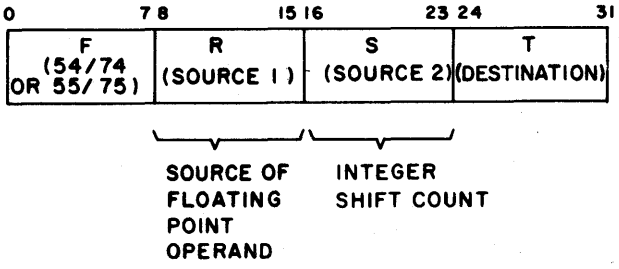


Figure 6-10. Example of Register Content for a Convert BCD to Binary, Fixed-Length Instruction

54/74 ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)

55/75 ADJUST EXPONENT OF (R) PER (S) TO (T)



54/74 ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)

This instruction adjusts the significance† of the floating-point operand in register R and transmits the adjusted result to register T. The rightmost 24 bits (48 bits for 74 function code) of register S contains a signed, two's complement integer. The absolute value of this integer is a shift count.

If the shift count is positive, the machine shifts the coefficient of the operand left the number of positions specified by the shift count or the number of positions needed to normalize† the coefficient, whichever is the smaller number.

In either case, the instruction reduces the exponent of the operand by one count for each position shifted. The instruction left-shifts an all zero coefficient the number of positions specified.

If the shift count is negative, the instruction shifts the coefficient of the operand right the number of positions specified by the shift count and increases the exponent of the operand by one count for each position shifted. If (R) is indefinite, the machine sets the (T) to indefinite and sets data flag bit 46 (indefinite result). If (R) equals machine zero, the machine sets (T) to machine zero but does not set data flag bit 42 (exponent overflow).

Ths instruction is undefined if the absolute value of the shift count is greater than 23_{10} for the 54 or 47_{10} for the 74 instruction. The addition of the shift count can cause either exponent overflow or exponent underflow.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

† Appendix B describes the process of adjusting a floating point operand for significance and of normalizing a floating-point number.

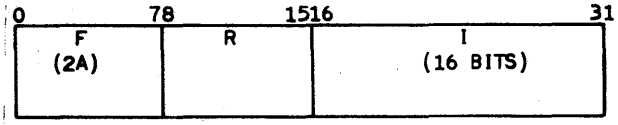
55/75 ADJUST EXPONENT OF (R) PER (S) TO (T)

This instruction transmits the adjusted operand from register R to result register T. The instruction sets the result exponent equal to the exponent of the operand in register S. The machine forms the coefficient of the result by shifting the coefficient of the operand from register R.

The shift count is the difference between the exponents in registers R and S. If the exponent in register R is greater than the exponent in register S, the machine shifts the coefficient left. The shift is to the right if the exponent in register R is less than the exponent in register S. If register R contains a zero coefficient, the exponent in register S is transferred to register T with an all zero coefficient. Figure 6-11 shows that the exponent in register S exceeds the exponent in register R by 4 (62 - 5E = 4); thus, the machine right-shifts the coefficient in register R four positions.

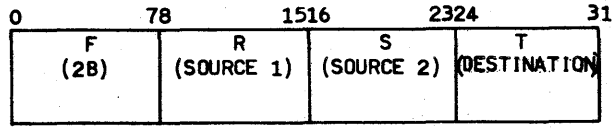
If a left shift exceeds the number of positions required for normalization, the machine sets the result to indefinite and sets data flag bit 42 (exponent overflow). If either or both operands are indefinite or machine zero, the machine also sets the result to indefinite. However, in this case, data flag bit 46 (indefinite result) is set and data flag bit 42 (exponent overflow) is not set.

2A ENTER LENGTH OF (R) WITH I (16 BITS)



This instruction transfers operand I contained in the rightmost 16 bits of the instruction word to the leftmost 16 bits of the 64-bit register specified by R. The rightmost 48 bits of register R are left unchanged.

2B ADD TO LENGTH FIELD



This instruction adds bits 0 through 15 of the 64-bit register specified by R to bits 48 through 63 of 64-bit register S and stores the result in bits 0 through 15 of register T. Overflow is ignored if it occurs. Bits 16 through 63 of register R are transferred to bits 16 through 63 of register T.

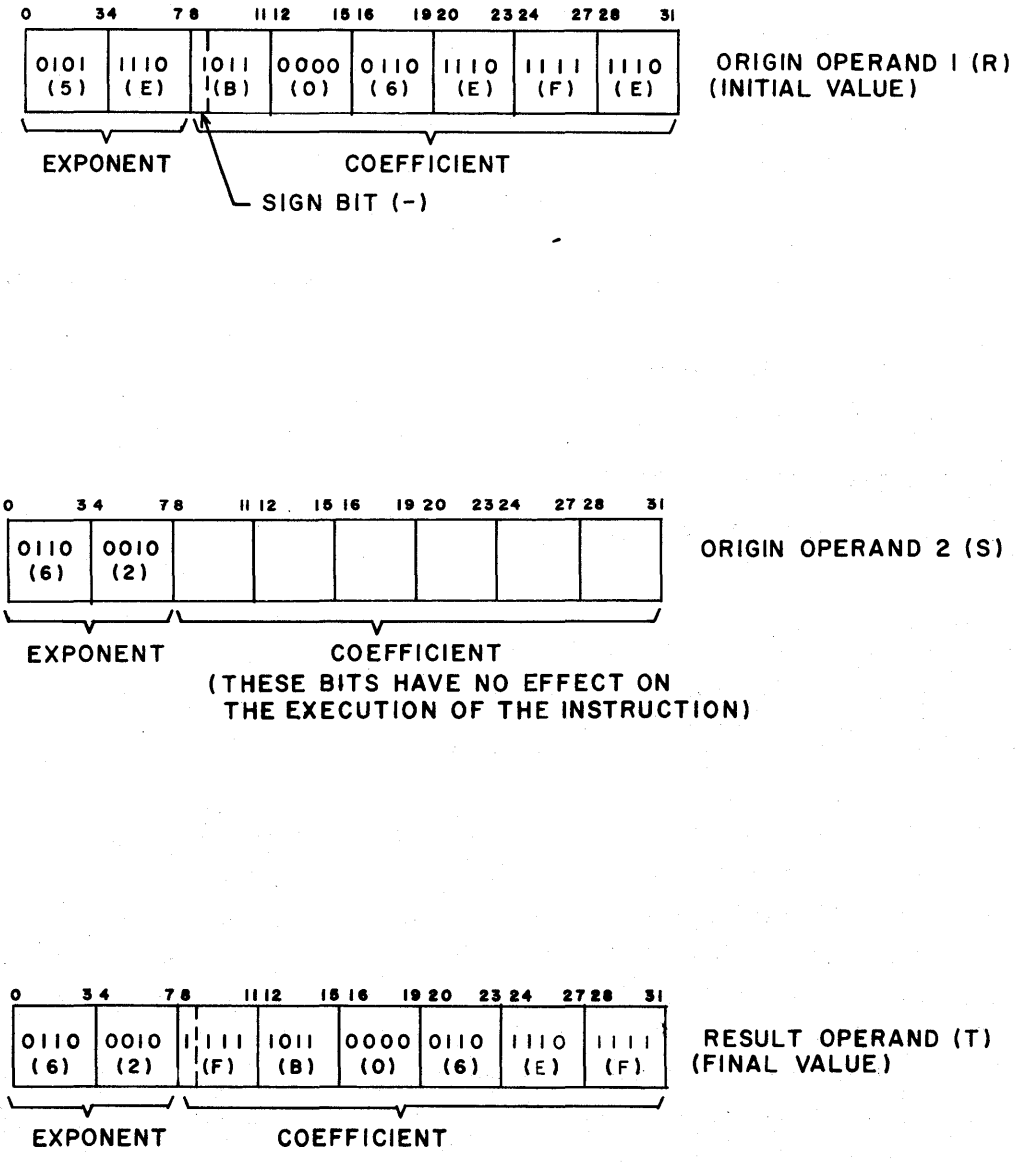


Figure 6-11. Example of Register Content for an Adjust Exponent of (R) Per (S) to (T)

BRANCH INSTRUCTIONS

The branch instructions compare or examine single bits, a 48-bit index, 32-bit floating-point operands, or 64-bit operands. The results of the comparison or examination determine whether the program continues with the next sequential instruction (branch condition not met) or branches to a different instruction sequence (branch condition met). The different instruction sequence may consist of a single instruction or a series of instructions beginning at the branch address specified in the branch instruction format.

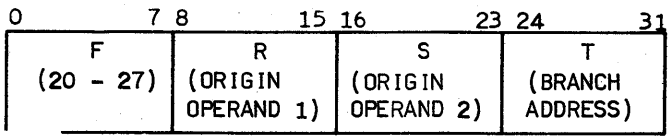
A special branch instruction provides for entering or leaving the monitor program.

20/24 BRANCH IF (R) = (S) (32/64 BIT FP)

21/25 BRANCH IF (R) ≠ (S) (32/64 BIT FP)

22/26 BRANCH IF (R) ≥ (S) (32/64 BIT FP)

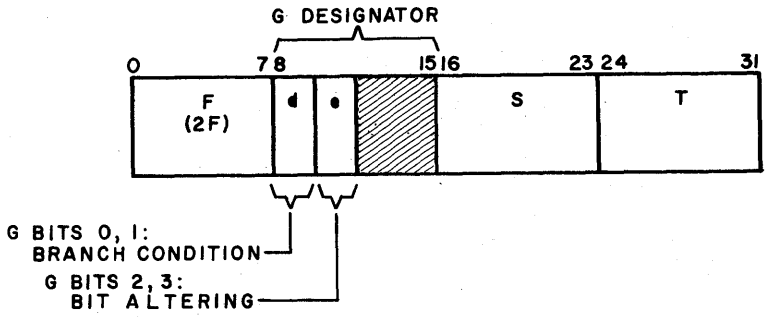
23/27 BRANCH IF (R) < (S) (32/64 BIT FP)



These instructions perform the indicated comparison of the 32-bit (64-bit for the 24 through 27 function codes) floating-point (FF) operands in the registers designated by R and S. If the specified comparison condition is met, the next instruction is read from the branch address, contained in the rightmost 48 bits of 64-bit register T. Register T is a 64-bit register for the 20 through 27 instruction codes. The byte and bit portions of the address (bits 59 through 63) are ignored in the reading of an instruction. If the specified comparison condition is not met, the next instruction is read from the next sequential program address. The comparison of (R) and (S) is based on the floating point compare rules in appendix B. An example of a 22 instruction is also in appendix B.

If either or both of the compared operands are indefinite, data flag bit 46 is set.

2F REGISTER BIT BRANCH AND ALTER



This instruction examines bit 63 of register T as specified by the G designator. A branch is made to the address contained in the rightmost 48 bits of register S. The branch occurs according to G bits 0 and 1 (table 6-4)..

TABLE 6-4. BIT BRANCHING CONDITIONS

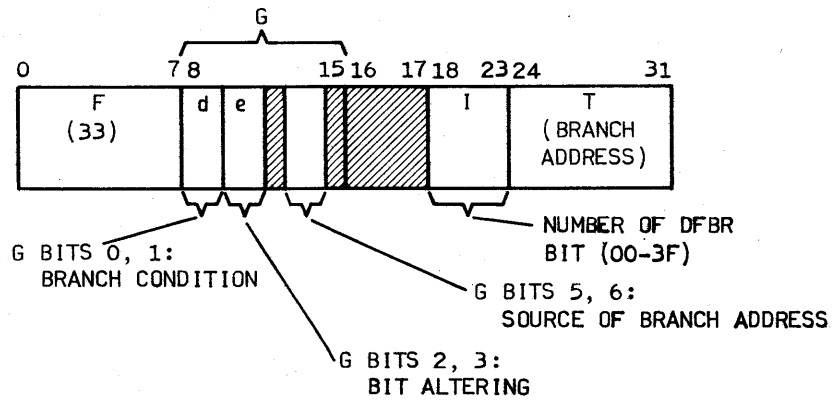
G Designator		Branch Conditions
Bit 0	Bit 1	
0	0	No branch
0	1	Unconditional branch
1	0	Branch if object bit = 1
1	1	Branch if object bit = 0

After the branch decision has been made and regardless of the decision, the object bit is altered according to G bits 2 and 3 (table 6-5)..

TABLE 6-5. BIT ALTERING CONDITIONS

G Designator		Altering Conditions
Bit 2	Bit 3	
0	0	No altering
0	1	Toggle the bit
1	0	Set the bit 1
1	1	Clear the bit 0

33 DATA FLAG REGISTER BIT BRANCH AND ALTER



This instruction examines the state of a specified bit in the data flag branch register (DFBR). If the designated branch condition is met, the next instruction is read from the half-word address as specified by G designator bits 5 and 6. If the designated branch condition is not met, the next instruction is read from the next sequential program address. In either case, the state of the DFBR bit is altered as specified by G bits 2 and 3.

The 6-bit designator I specifies the number of the DFBR bit. The bit numbers range from 00 through 3F (00 through 63_{10}). The 2-bit designator denotes the branch condition (table 6-6).

TABLE 6-6. DFBR BIT BRANCH CONDITIONS

G Designator		Branch Condition
Bit 0	Bit 1	
0	0	No branch
0	1	Unconditional branch
1	0	Branch if selected DFBR bit = 1
1	1	Branch if selected DFBR bit = 0

After the branch decision is made, the instruction alters the DFBR bit according to G designator bits 2 and 3 (table 6-7).. The bit altering occurs regardless of the branch decision.

TABLE 6-7. DFBR BIT ALTERING CONDITIONS

G Designator		Altering Conditions
Bit 2	Bit 3	
0	0	No altering
0	1	Toggle the bit
1	0	Set the bit 1
1	1	Clear the bit 0

NOTE

Do not attempt to alter bits in the DFBR product field since the altering of these bits is only a function of the corresponding data flag and flag mask bits.

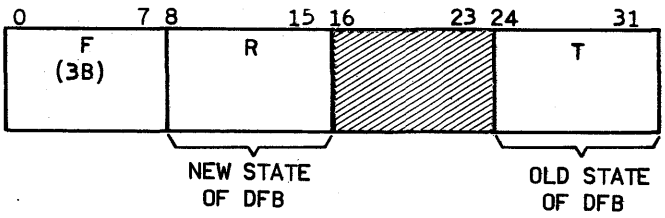
Since the 33 instruction begins execution without waiting until the machine has completed all operations, the data flag bits may set on any minor cycle during execution of this instruction. Therefore, the object bit is sampled two minor cycles after the 33 instruction is loaded into instruction register 0 (IR0). This sampled object bit is used to control the decision to branch and the altering of the actual object bit in the data flag register. Consequently, any data flag bits setting after the object bit is sampled will not affect the decision to branch. Also, if the sampled object bit is a zero, any data flag bits setting afterwards will not be cleared or toggled to a zero.

The source of the branch address is determined by the state of G designator bits 5 and 6 (table 6-8).

TABLE 6-8. DFBR BRANCH ADDRESS SOURCE CONDITIONS

G Designator		Branch Address Source Conditions
Bit 5	Bit 6	
0	0 or 1	Register T contains the branch address.
1	0	Branch address is formed by addition of the T designator, used as a halfword item count, to the content of the program address register.
1	1	Branch address is formed by the subtraction of the T designator, used as a halfword item count, from the contents of the program address register.

3B DATA FLAG REGISTER LOAD/STORE

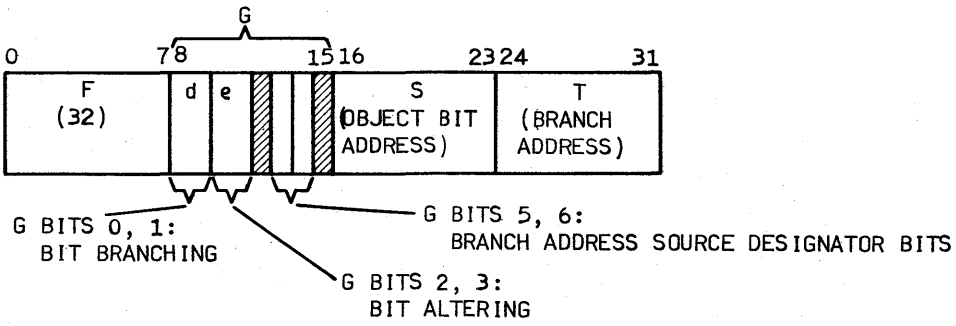


This instruction transfers the content of register R to the DFB register. The 3B instruction also transmits the previous content of the DFB to the T register. Since the DFB is a 64-bit register, both R and T must be 64-bit registers. The R and T designators may be equal which exchanges data flag values.

NOTE

An immediate data flag branch results at the termination of this instruction if the new content of the DFB register meets the appropriate branch conditions.

32 BIT BRANCH AND ALTER



This instruction reads the word from the address contained in the register designated by S and examines the specified object bit. The remaining bits are not used in the instruction. If the object bit meets the branch condition specified by G designator bits 0 and 1, the next instruction is read from the branch address contained in the T register. If the branch condition is not met, the next instruction is read from the next sequential program address. In either case, G designator bits 2 and 3 determine the final state of the object bit. Tables 6-9 and 6-10 list the bit branching and altering conditions, respectively. Table 6-11 lists branch address source conditions.

TABLE 6-9. BIT BRANCHING CONDITIONS

G Designator		Branch Conditions
Bit 0	Bit 1	
0	0	No branch
0	1	Unconditional branch
1	0	Branch if object bit = 1
1	1	Branch if object bit = 0

TABLE 6-10. BIT ALTERING CONDITIONS

G Designator		Altering Conditions
Bit 2	Bit 3	
0	0	No altering
0	1	Toggle the bit
1	0	Set the bit 1
1	1	Clear the bit 0

NOTE

If G bits 0, 2, and 3 = 0, the word containing the object bit is not read and the object bit is not altered.

If G bit 0 = 1 and G bits 2 and 3 = 0, the word is read but the object bit is not written.

TABLE 6-11. BRANCH ADDRESS SOURCE CONDITIONS

G Designator		Branch Address Source Conditions
Bit 5	Bit 6	
0	0 or 1	Register T contains the branch address.
1	0	Branch address is formed by addition of the T designator, used as a halfword item count, to the contents of the program address register.
1	1	Branch address is formed by the subtraction of the T designator, used as a halfword item count, from the contents of the program address register.

Figure 6-12 shows an example of the bit branch and alter instruction with assumed register content and branch conditions. The object bit is located in bit 7 of byte 3 of word 100000. Since G bit 0 equals 1 and G bit 1 equals 0 and the object bit is a 1, a branch takes place to the assumed branch address which is contained in the T register as specified by G designator bits 5 and 6.

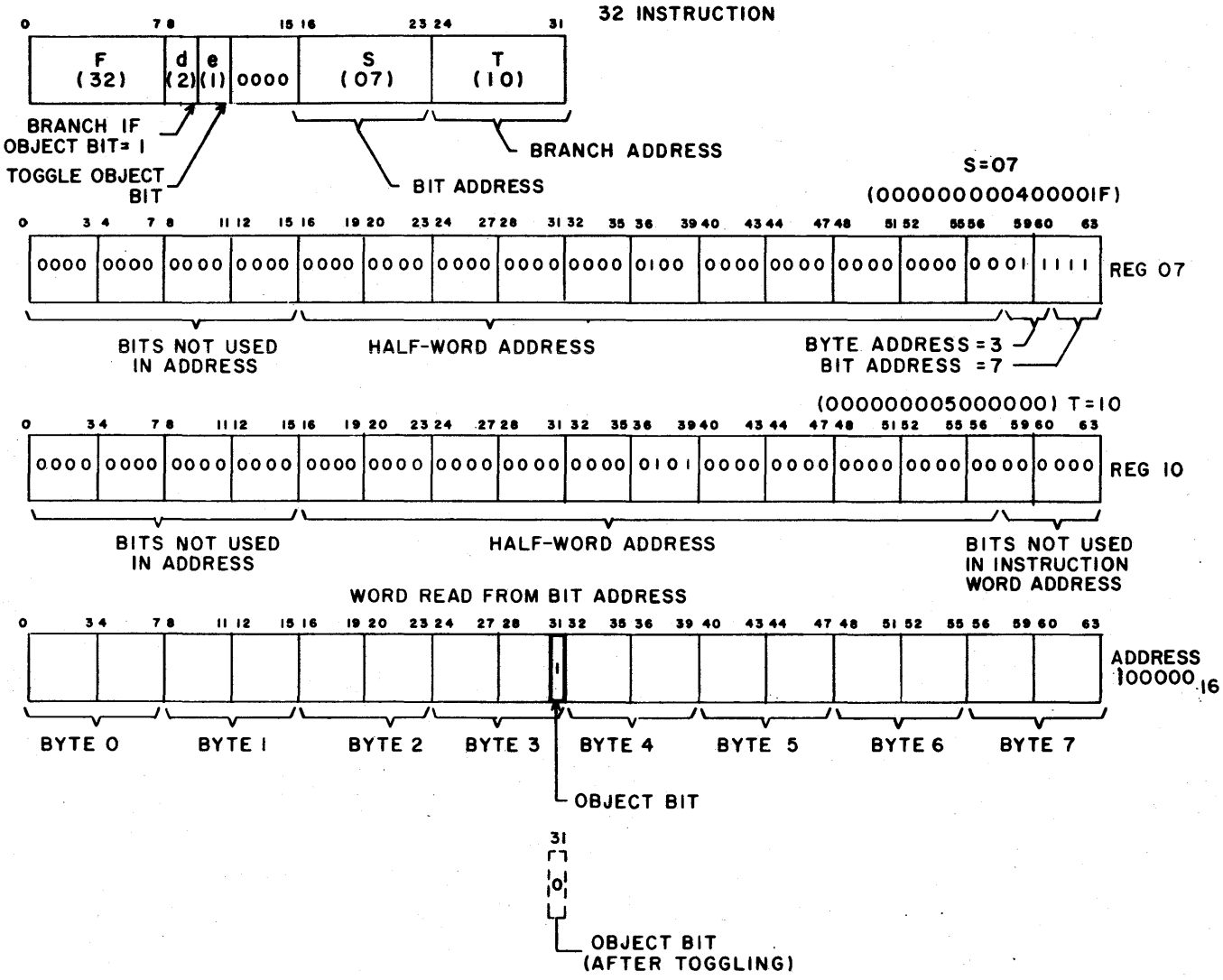
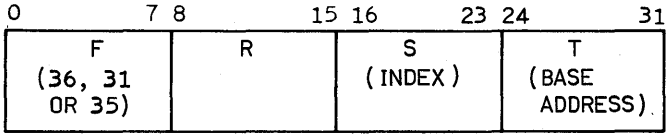


Figure 6-12. Example of Bit Branch and Alter Instruction

36 BRANCH AND SET (R) TO NEXT INSTRUCTION

31 INCREASE (R) AND BRANCH IF (R) ≠ 0

35 DECREASE (R) AND BRANCH IF (R) ≠ 0



36 BRANCH AND SET (R) TO NEXT INSTRUCTION

This instruction first stores the address of the next sequential instruction into register R. The program then branches to (S) + (T), where (S) represents an item count (index) of half-words and (T) specifies the base address. The machine forces bits 0 through 15 of register R to zeros. Bits 59 through 63 are undefined. If the instruction designator R is equal to the designator S, the results of this instruction are undefined.

If S = 0 and R = T, this instruction sets register R to the half-word address of the next instruction. The program then continues at the next instruction. This method provides a means of sampling the Program address register.

31 INCREASE (R) AND BRANCH IF (R) ≠ 0

35 DECREASE (R) AND BRANCH IF (R) ≠ 0

This instruction first increments (31 function code) or decrements (35 function code) the rightmost 48 bits of register R by one. The leftmost 16 bits of register R are not altered and arithmetic overflow (if it occurs) is ignored.

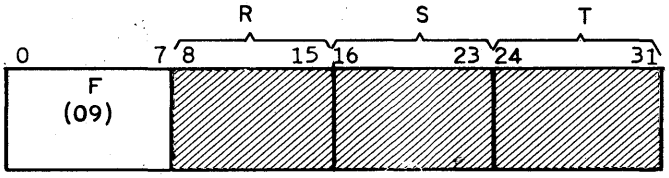
If the increment/decrement operation produces zeros in the rightmost 48 bits of R, the program reads the next sequential instruction. If the rightmost 48 bits of R are not all zeros, the program branches to (S) + (T), where (S) represents an item count in half-words and (T) specifies the base address.

09 EXIT FORCE

This instruction provides a means of exchanging program control between a job and monitor program. For example, if the machine is operating in the job mode, the exit force instruction causes a branch to the beginning address of a portion of the monitor program. Similarly, in a monitor program, the exit force performs a branch to a job program. The starting address of the invisible package and register file for the job is defined by the content of the register designated by T and S, respectively. For either type of exchange (job to monitor or monitor to job), the invisible package and register file for the current job are transferred to/from central storage. (Refer to section 5 for a more comprehensive description of monitor and job operations.)

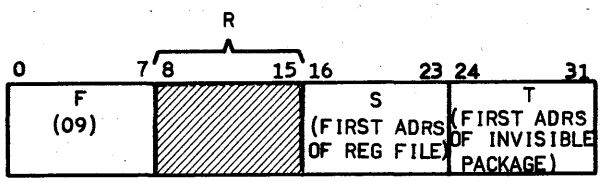
JOB TO MONITOR

The following exit force instruction format is an exchange from a job to a monitor program. The R, S, and T designators are unused and must be zeros. In this case, the instruction switches the machine to the monitor mode and unconditionally branches to the address specified by the rightmost 48 bits of register 05 in the register file. Register 05 address is an absolute bit address since the machine was switched to the monitor mode. The monitor program then proceeds from this beginning address.



MONITOR TO JOB

The following instruction format is an exchange from the monitor to a job program. The R designator is unused and must be zeros.



When exchanging from the monitor mode to a job, this instruction loads the registers from the register file stored in central storage, beginning at the address contained in the register specified by S. The instruction also loads the invisible package for the applicable job from central storage, beginning at the address in the register specified by T. The S and T addresses are absolute bit addresses. Figure 6-13 shows formats of the addresses in the S and T registers.

In the S register, bits 38 through 63 define the starting address in central storage for loading the 256_{10} words in the register file. The starting address is the same as the first address of the page and must be on a small page boundary. In a small page starting address, bits 49 through 63 are always zeros. This means that the absolute bit range of the register file starting address is 0000000_{16} through $3FF8000_{16}$. Since the register file is loaded from central storage in sequential 64-bit words, the bit, byte, and half-word bits of the address are not advanced. Thus, from an assumed starting address of $XX00000_{16}$, the sequence of loading the register file advances the address of a value of $XX3FC0_{16}$. If either the S designator or the content of register S is equal to zero, the job's register file and the monitor's register file are identical.

In the T register, bits 38 through 63 define the starting address in central storage for loading the invisible package into 16 sequential word locations.

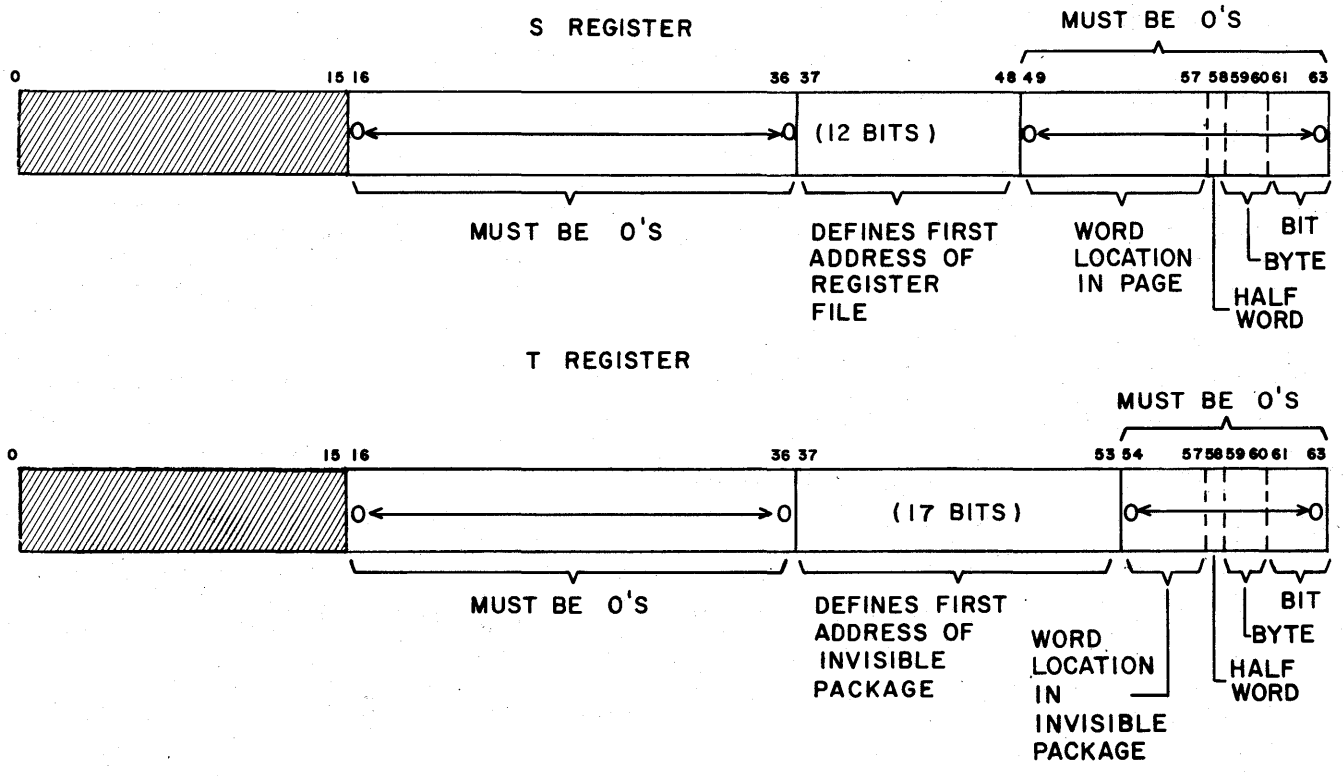
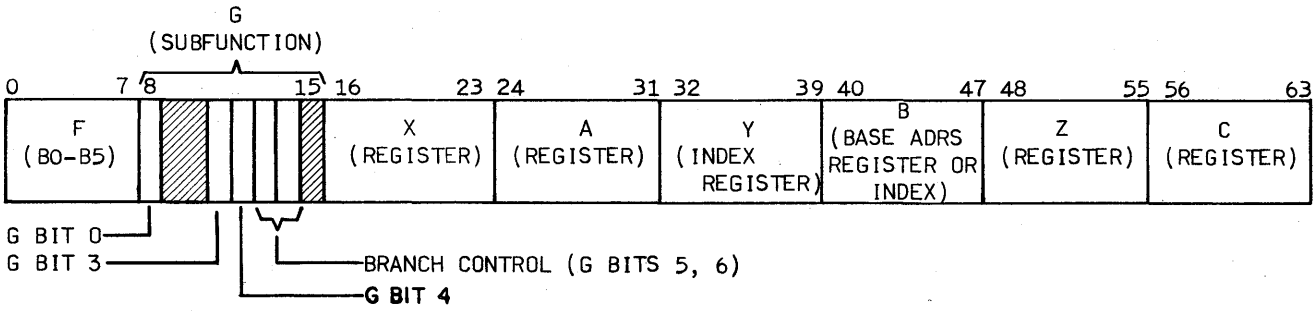


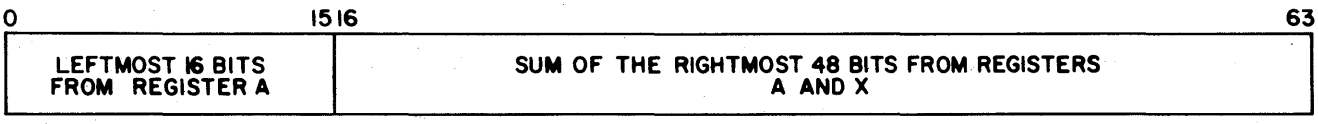
Figure 6-13. Address Formats for Exit Force Instruction (Monitor to Job)

B0 COMPARE INTEGER, BRANCH IF (A) + (X) = (Z)
B1 COMPARE INTEGER, BRANCH IF (A) + (X) ≠ (Z)
B2 COMPARE INTEGER, BRANCH IF (A) + (X) ≥ (Z)
B3 COMPARE INTEGER, BRANCH IF (A) + (X) < (Z)
B4 COMPARE INTEGER, BRANCH IF (A) + (X) ≤ (Z)
B5 COMPARE INTEGER, BRANCH IF (A) + (X) > (Z)



For these instructions, G bit 1 and 2 are 0. If G bit 0 is cleared (0), registers A, X, C, and Z are 64 bits. If G bit 0 is set (1), registers A, X, C, and Z are 32 bits. Registers B and Y are 64 bits.

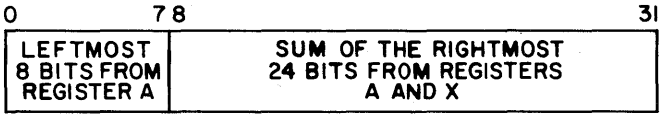
If G bit 0 is 0, the sum of the rightmost 48-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 48 bits of register Z, according to the specified branch condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 48 bits of register C. The leftmost 16 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



Then the sum of the rightmost 48 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values:

G bit 3 = 0	The integers compared are the 48-bit result of registers A and X and the rightmost 48 bits read from register Z.
G bit 3 = 1	The integers compared are the 64 bits stored in register C and the 64 bits read from register Z. This compare is defined for the B0 and B1 instructions only.
G bit 4 = 0	The integers compared are interpreted as signed two's complement numbers.
G bit 4 = 1	The integers compared are interpreted as unsigned numbers.

If G bit 0 is 1, the sum of the rightmost 24-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 24 bits of register Z, according to the specified branch condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 24 bits of register C. The leftmost 8 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



Then the sum of the rightmost 24 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values:

G bit 3 = 0	The integers compared are the 24-bit result of registers A and X and the rightmost 24 bits read from register Z.
G bit 3 = 1	Undefined.
G bit 4 = 0	The integers compared are interpreted as unsigned two's complement numbers.
G bit 4 = 1	The integers compared are interpreted as signed numbers.

If the specified branch condition is met, the program address branches to the address specified by the branch control bits in the G designator (table 6-12). In all cases, the index is an item count in halfwords that is left-shifted five places before the addition or subtraction.

TABLE 6-12. INDEX BRANCH INSTRUCTION DESIGNATORS

G Designator Bit State	Branch Address
Bit 5 = 0	Branch to address formed by adding the item count in register Y to the base address in register B. The item count is left-shifted five places before the addition. Overflow, if any, is ignored. If the Y or B designator is equal to the C designator, the instruction is undefined.
Bit 5 = 1	Branch according to the state of G designator bit 6 as follows:
Bit 6 = 0	Branch to address formed by adding 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before addition.
Bit 6 = 1	Branch to address formed by subtracting 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before subtraction.

If the branch condition is not met, the program reads the next sequential instruction.

If one of the following conditions occur, the operation becomes undefined.

- G bit 0 is 1 and G bit 3 is 1
- G bit 3 is 1 for instructions B2, B3, B4, and B5
- G bit 5 is 0 and G bit 6 is 1

Table 6-13 relates integer ranges to the state of G bit 4.

TABLE 6-13. INTEGER RANGES

	48-bit hexadecimal quantities in descending order from the largest to the smallest, from top to bottom.	
	G bit 4 = 0	G bit 4 = 1
Largest ↓ Smallest	7F ------ FF 7F ------ FE 00 ------ 01 00 ------ 00 FF ------ FF 80 ------ 01 80 ------ 00	FF ------ FF FF ------ FE 80 ------ 01 80 ------ 00 7F ------ FF 00 ------ 01 00 ------ 00

B0 COMPARE FP, BRANCH IF (A) = (X)

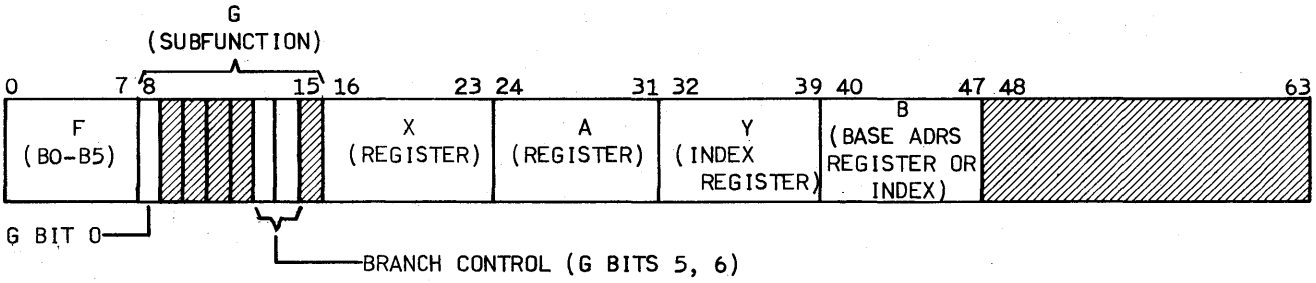
B1 COMPARE FP, BRANCH IF (A) ≠ (X)

B2 COMPARE FP, BRANCH IF (A) ≥ (X)

B3 COMPARE FP, BRANCH IF (A) < (X)

B4 COMPARE FP, BRANCH IF (A) ≤ (X)

B5 COMPARE FP, BRANCH IF (A) > (X)



If G bit 1 is 1 and G bit 2 is 0, these instructions compare the two floating-point operands from registers A and X according to the floating-point compare rule in appendix B. If G bit 0 is clear (0), the registers contain 64 bits. If G bit 0 is set (1), the registers contain 32 bits. Registers B and Y are always 64 bits.

If the specified branch condition is met, the program address branches to the address specified by the branch control bits in the G designator (table 6-14). In all cases, the index is an item count in halfwords that is left-shifted five places before the addition or subtraction.

TABLE 6-14. INDEX BRANCH INSTRUCTION DESIGNATORS

G Designator Bit State	Branch Address
Bit 5 = 0	Branch to address formed by adding the halfwords item count in register Y to the base address in register B. The item count is left-shifted five places before the addition. Overflow, if any, is ignored. If the B or Y designator is equal to the C designator, the instruction is undefined.
Bit 5 = 1	Branch according to the state of G designator bit 6 as follows:
Bit 6 = 0	Branch to address formed by adding 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before addition.
Bit 6 = 1	Branch to address formed by subtracting 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before subtraction.

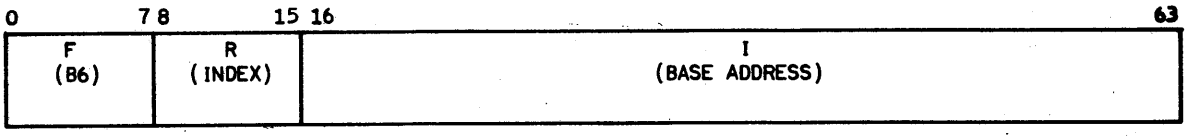
If the branch condition is not met, the program reads the next sequential instruction.

If one of the following conditions occur, the operation becomes undefined.

- G bit 3=1, G bit 4=1, or G bit 7=1
- Designator C and/or Z not equal to 0.
- G bit 5=0 and G bit 6=1

The applicable data flag bit is 46 (indefinite result).

B6 BRANCH TO IMMEDIATE ADDRESS (R) + I (48 BITS)



This instruction branches unconditionally to the address formed by the sum of the rightmost 48 bits of register R as the index and I as the base address. The index represents an item count of half-words which is shifted left five positions before being added to the base address. Overflow, if any, is ignored.

The instruction makes a direct branch to the base address if the R designator is zero or if the rightmost 43 bits of register R are zeros.

VECTOR INSTRUCTIONS

The vector instructions perform operations on ordered scalars. Generally, the vector instructions read the scalars, which are in the form of 32-bit or 64-bit floating point operands, from consecutive storage locations over a specified address range (field). These instructions perform the designated operation on each set of operands and store the results in consecutive addresses of a result field, beginning at a specified starting address. Thus, a single vector instruction can perform operations on two source fields of vector operands and automatically store the results in a result field of storage.

INSTRUCTION FORMATS

All vector instructions use the same general instruction format (figure 6-14). Table 6-15 lists each of the 8-bit designators in the vector instructions and gives a brief description of the function.

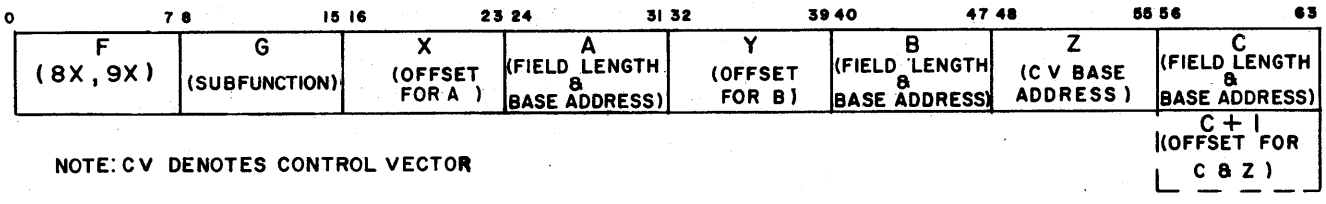


Figure 6-14. General Vector Instruction Format

TABLE 6-15. VECTOR INSTRUCTION DESIGNATORS

Designator	Function
F	Function code
G	Subfunction code
X, Y	Specify registers that hold address offsets for corresponding source operand fields
A, B	Specify registers that hold base addresses and field lengths for source operand fields
Z	Specifies register that contains the base address of the control vector (CV)
C	Specifies register that contains the base address and field length of the result field If C+1 is used by the instruction, C must be an even number since the machine forms C+1 by forcing the rightmost bit of C to a 1. If the C designator specifies an odd-numbered register, the results of the instruction become undefined.
C+1	Specifies register that holds offset for the control vector and the result field; C+1 always references an odd register

SUBFUNCTION BITS

Table 6-16 lists the subfunction bits and their general usage. Table 6-17 gives the sign control subfunction bits.

If the Z designator is zero, no control vector is used; thus, G-bit 1 becomes undefined. If G bit 3 and/or G bit 4 = 1, the A and/or B designator denotes a constant which is used as each element of the respective vector field. The instruction ignores the associated offsets in this case. The registers specified by A and B, respectively, contain these constants. Registers A and B are always 64-bit registers except when G bits 3 and 4 indicate a broadcast. When broadcasting, the size of registers A and B track the size specified by G bit 0 (refer to table 6-16).

Appendix C gives a composite listing of the G designator bits usage according to function code.

If bit 3 of G, 4 of G, or both are ones, then the A, B, or both source fields are constants used as each element of the respective vector stream and the associated offsets are ignored. These constants are found in the registers specified by A and B, respectively. If bit 3, 4, or both are ones and bit 0 of G is a one, register A, B, or both are 32-bit registers. For all other cases, registers A and B are 64-bit registers.

TABLE 6-16. SUBFUNCTION BITS

Bit No.	State	Subfunction
0	0	64-bit operands (words)
	1	32-bit operands (half-words)
1	0	Control vector operates on ones
	1	Control vector operates on zeros
2	0	No offset for result field and control vector
	1	Offset for result field and control vector
3	0	Normal source vectors A
	1	Broadcast repeated (A)
4	0	Normal source vectors B
	1	Broadcast repeated (B)
5	X	Sign control (refer to table 6-17)
6	X	
7	X	

TABLE 6-17. SIGN CONTROL SUBFUNCTION BITS

Bit 5	Bit 6	Bit 7	Control Operation
0	0	X	The operands from the A stream are used in the normal manner.
0	1	X	The coefficients of the operands from the A stream are complemented before they are used.
1	0	X	The magnitude of the operands from the A stream is used.
1	1	X	The coefficients of all positive operands from the A stream are made negative before they are used. Negative operands are not altered.
X	X	0	The operands from the B stream are used in the normal manner.
X	X	1	The magnitude of the coefficients of the operands from the B stream is used.

NOTE

1. X denotes that the bit can be either a 0 or a 1.
2. Any required complementing is two's complement. Complementing is performed before the operand is used in the specified arithmetic operation. If the complement of the coefficient 8000 0000 0000 is required, the operand is used as 4000 0000 0000 with 1 added to the exponent.
3. Any necessary significance calculation is performed before the previous complementing is performed.

FIELD LENGTHS, BASE ADDRESS, AND OFFSETS

Figures 6-15 and 6-16 show the formats of the register contents for the field lengths, base addresses, and offsets. The computer allows 16-bit field lengths to be specified and assumes them to be positive. The field lengths are in the range of 0 through $2^{16}-1$ before any offset adjustments. The offsets are taken from a 48-bit register and must have at least 32 identical sign bits. The offsets are in the range of -2^{16} to $2^{16}-1$.

The operation of subtracting the offset from the field length must result in a field length which is positive and less than 2^{16}. If the resulting vector length is not positive and less than 2^{16}, it is treated as a zero vector length. The instruction obtains the beginning address by adding the offset (including sign extension) to the base address (figures 6-16 and 6-19). In the (offset + base address) addition, the offset is first shifted left five (half-words) or six (words) places since the bit and byte bits are not used in the vector operand field address.

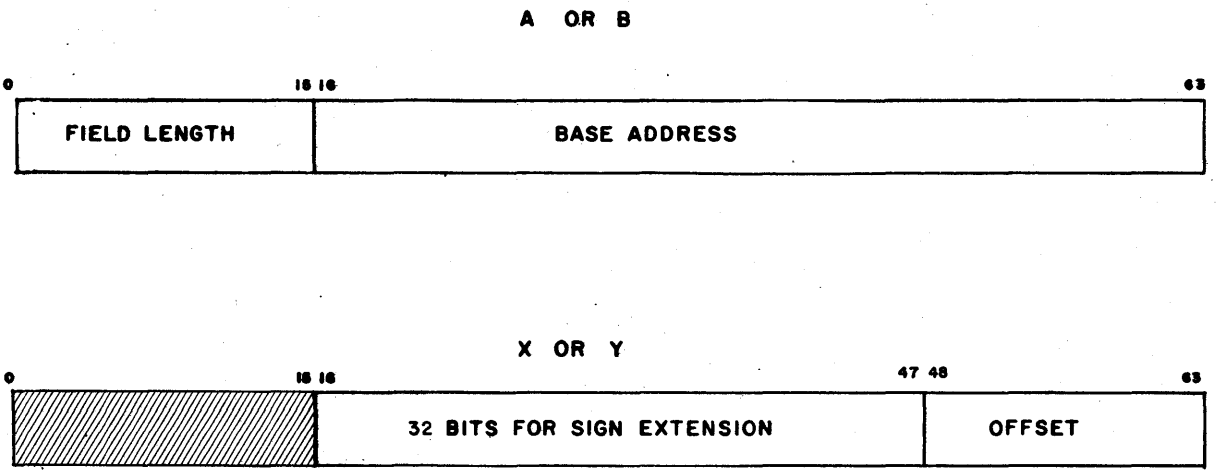


Figure 6-15. Operand Field Length, Base Address, and Offset Formats

The C and C+1 registers are identical in format to the A or B and X or Y content, respectively. If bit 2 specifies that vector field C is to be offset, register C+1 contains the offset.

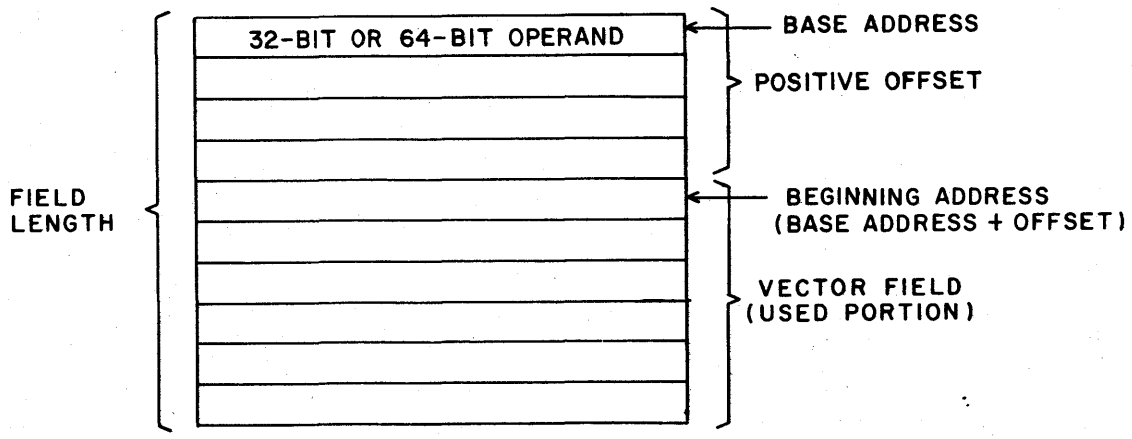


Figure 6-16. Vector Field Address Format

CONTROL VECTOR

When the instruction specifies a control vector (Z designator $\neq 0$), a single bit from the control vector controls the storing of each element in the result field. When a bit from the control vector prohibits the storing of a result element, the instruction does not alter the previous content of the corresponding storage address. Thus, the nth bit read from the control vector prohibits or allows the storing of the nth result in the result vector field.

Bit 1 of the G designator selects whether a 0 or a 1 control vector bit allows the storing of the result (table 6-11). If bit 1 of the G designator is a 0 or a 1, the instruction stores the nth result if the nth bit of the control vector is a 1 or a 0, respectively.

The rightmost 48 bits of the register designated by Z contains the base address of the control vector (figure 6-17). The control vector uses the same field length as result vector C.

The addition of the offset and base address provides the starting bit address of the control vector. Since offsets are item counts, the result vector and control vector use the same offset; however, the control vector offset represents a bit offset.

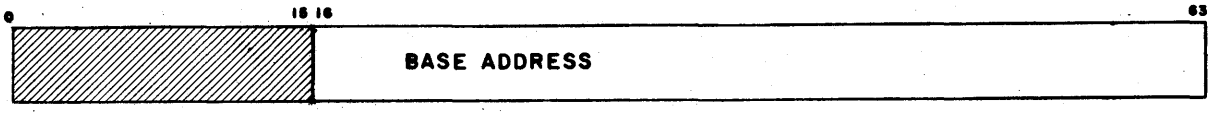


Figure 6-17. Control Vector Base Address Format (Z)

VECTOR INSTRUCTION TERMINATION

Vector instructions terminate when the result vector field is exhausted.

1. Exhausting a vector which has an offset.

   A vector is deemed exhausted prior to the first operand fetch if the result of subtracting the offset from the field length is zero or negative.

   For cases of zero field length, the resulting vector length used is the rightmost 16 bits of the two's complement of the offset. If this 16-bit quantity is zero or negative, the vector is deemed exhausted prior to the first operand fetch.

   A vector is exhausted when the result of subtracting both the offset and the number of operands encountered thus far, from the field length, is zero.

2. Exhausting a vector which has no offset and exhausting other data fields or data strings.

   The string, field, or vector is deemed exhausted prior to the first operand fetch if its length is zero. These strings, fields, and vectors are exhausted when the result of subtracting the number of elements encountered thus far from the field length is zero.

†Appendix C provides a complete listing of the various vector instruction field conditions and the resulting termination conditions.

EXAMPLE OF VECTOR INSTRUCTION OPERATION

Figure 6-18 shows the register content and figure 6-19 shows the resulting vector address fields of an assumed add U, A+B→C (80) vector instruction. Although an 80 instruction is used, the general sequence of operations is the same for all vector instructions.

The G designator bits used in the example specify the following conditions for the operation of the instruction.

G-Designator Bit	Condition
0 = 1	32-bit, floating point operands
1 = 0	Control vector operates on ones (ones in control vector enable storage of corresponding control vector)
2 = 1	Result vector and control vector fields are offset (C+1 designator is used)
3 = 0	Normal vector source stream A
4 = 0	Normal vector source stream B
5 = 0 6 = 0	Use the operands from the A stream in the normal manner
7 = 0	Use the operands from the B stream in the normal manner

The X, A, Y, B, Z, and C register designator numbers are shown in parentheses. Thus, register 10 contains the offset for vector field A, register 11 contains the base address for vector field A, etc.

Since the bit and byte address bits are not used in the vector field addresses, successive half-word addresses are shown. Thus, incrementing address 10000_{16} by a half-word count gives 10020_{16} as the next successive address.

With the A vector offset equal to +4 and the B vector offset equal to -4 (figures 6-18 and 6-19), the first vector add U, A+B→C operation adds the A and B operands from the respective addresses 10080_{16} and $1FF80_{16}$. The result of the first add operation does not store, because bit 7 of the addressed control vector field is a zero. Successive add operations add successive A and B operands, storing the results only when a corresponding one appears in the control vector.

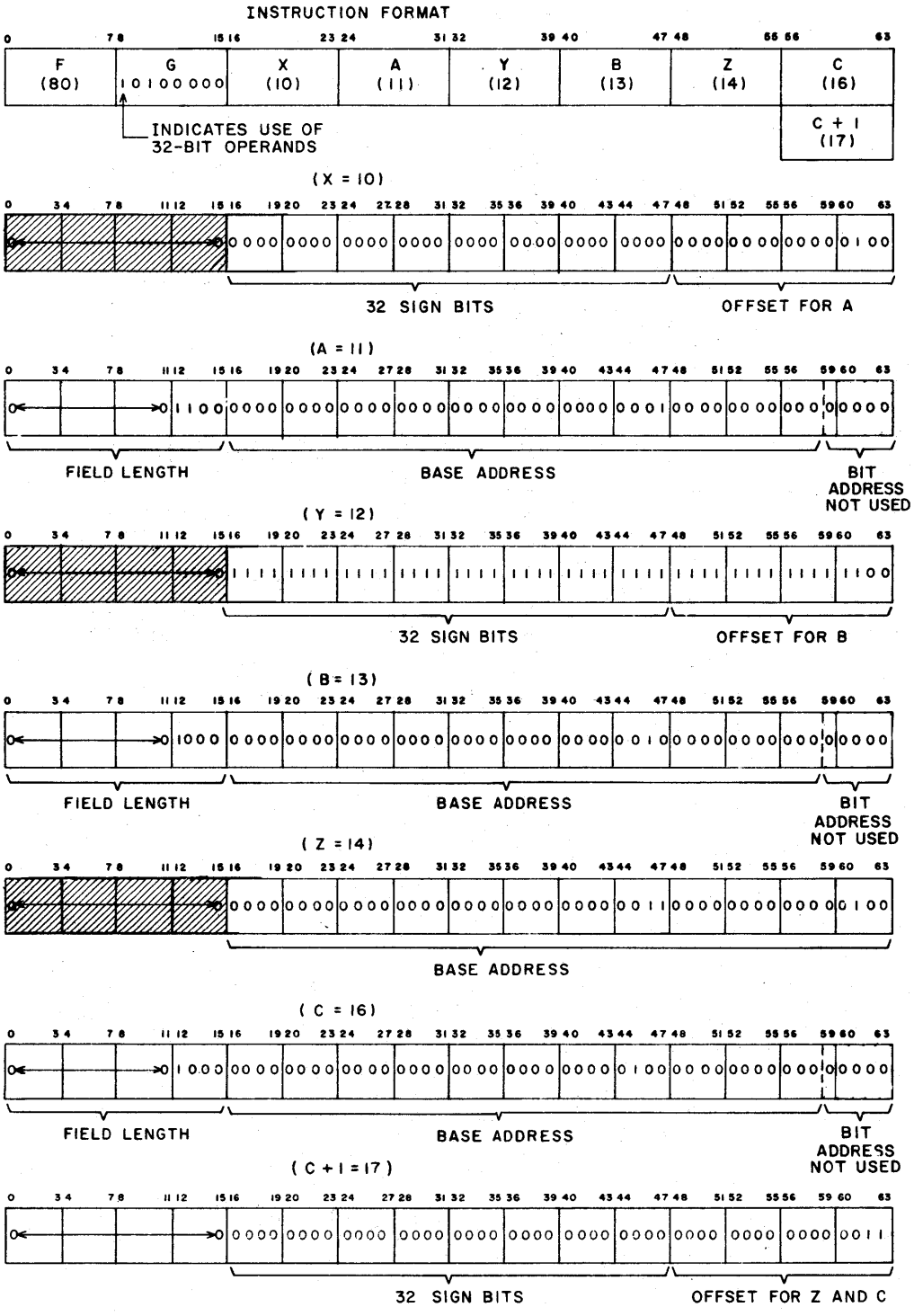


Figure 6-18. Vector Instruction Example of Register Content and Instruction Format

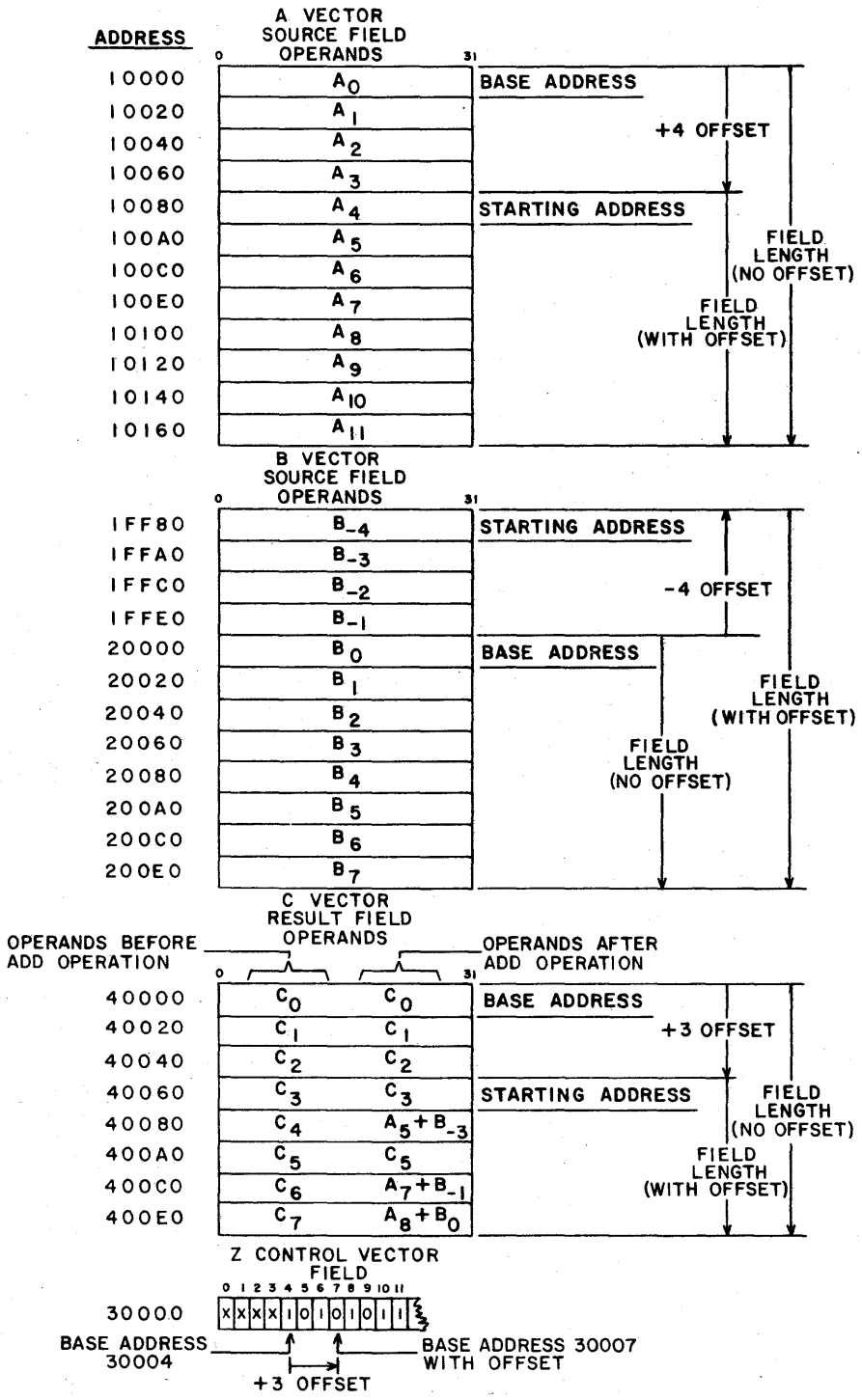
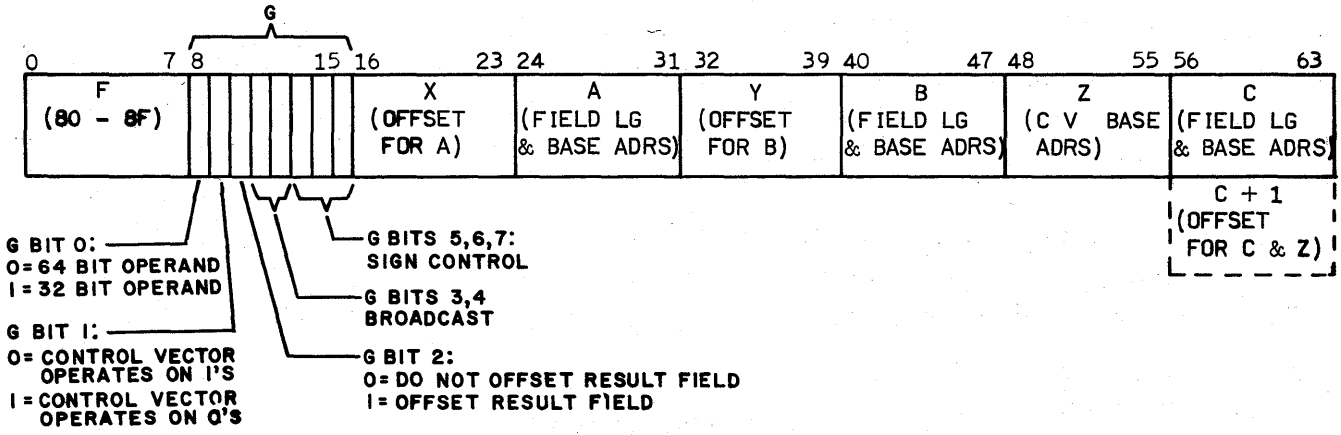


Figure 6-19. Vector Address Fields for Vector Instruction Example

80 ADD U; A + B→C
81 ADD L; A + B→C
82 ADD N; A + B→C
84 SUB U; A − B→C
85 SUB L; A − B→C
86 SUB N; A − B→C
88 MPY U; A • B→C
89 MPY L; A • B→C
8B MPY S; A • B→C
8C DIV U; A/B→C
8F DIV S; A/B→C



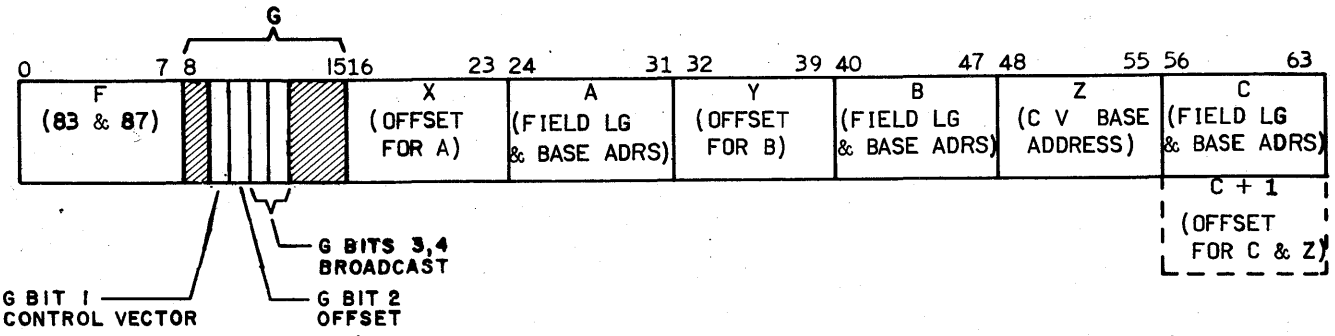
These instructions perform the indicated floating point† arithmetic operations on the elements of vector fields A and B. The instructions store the result elements in vector field C. All of the vector elements are in the form of 32-bit or 64-bit floating-point operands. The U, L, N, and S designators specify the upper, lower, normalized upper, or significant results, respectively.

Applicable data flag bits are 41 (floating point divide fault), 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

†Appendix B describes the floating point arithmetic operations.

83 ADD A; A + B → C

87 SUB A; A − B → C



These instructions add/subtract bits 16 through 63 of the B vector elements to/from bits 16 through 63 of the A vector elements (figure 6-20). The instructions store the results in bits 16 through 63 of the C vector elements. Bits 16 through 63 of the source vector elements are treated as 48-bit, positive integers. Arithmetic overflow is ignored if it occurs.

The instructions transmit bits 0 through 15 of the A vector elements to corresponding portions of the C vector elements. As shown in the previous instruction format, bit 0 of the G designator must be zero since only 64-bit operands are used.

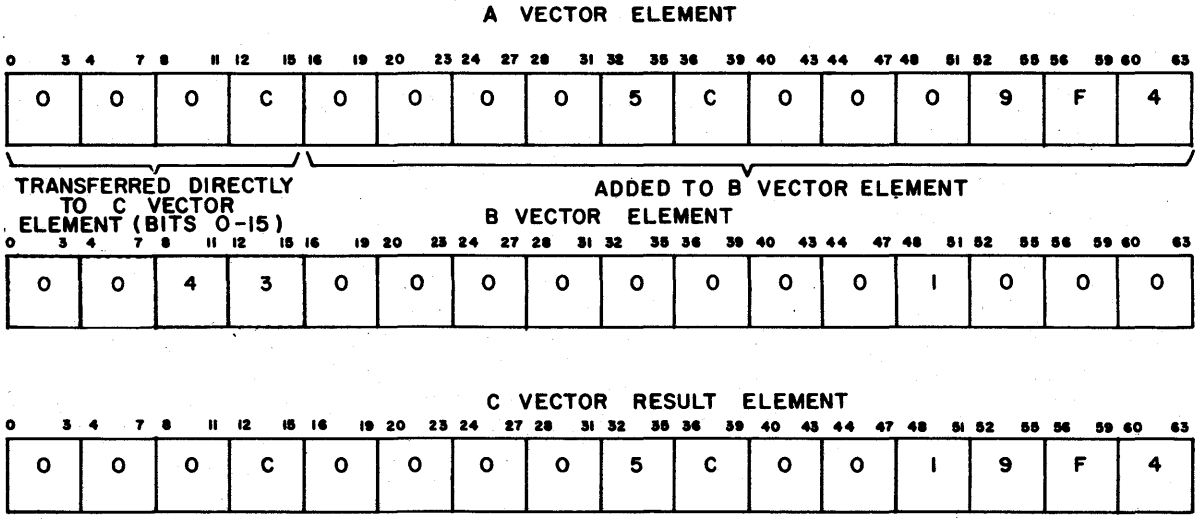


Figure 6-20. Example of an Add A; A + B → C Instruction

98 TRANSMIT A→C

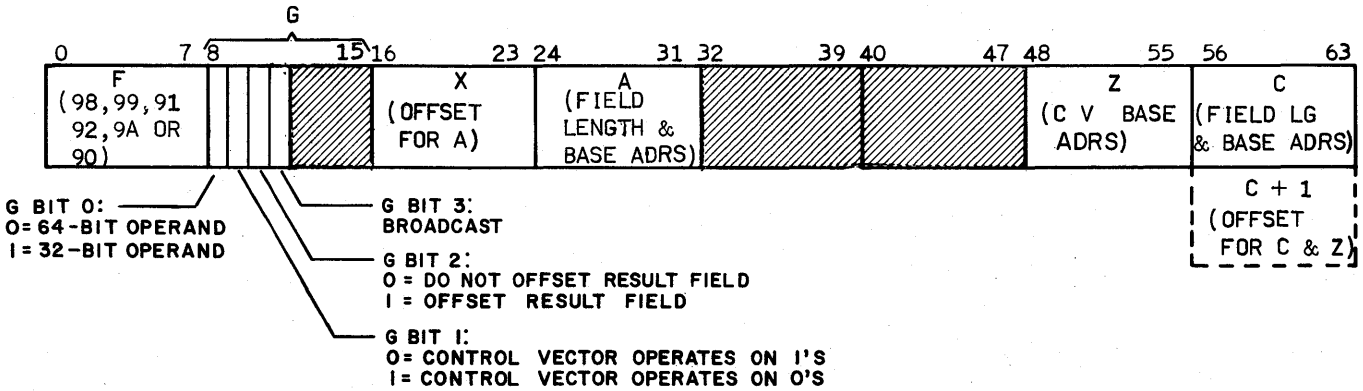
99 ABSOLUTE A→C

91 FLOOR A→C

92 CEILING A→C

9A EXPONENT OF A→C

90 TRUNCATE A→C



98 TRANSMIT A → C

This instruction transmits each element of the source field A to successive elements of result field C throughout the modified field length.

99 ABSOLUTE A → C

This instruction transmits the absolute value of each element of the source field A to successive elements of result field C throughout the modified field length. All vector elements are 32- or 64-bit, floating-point operands. If the coefficient of the source operand is positive, the element is transmitted directly to the result vector field; if the coefficient is negative, the coefficient is complemented before transmission.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

91 FLOOR A → C

This instruction converts each floating point element of source field A to the nearest integer less than or equal to it. The resulting integers are transmitted to corresponding elements of result field C throughout the modified field length. The resulting integer is always an unnormalized, floating point number with a positive exponent.

If the exponent of the source element is positive (greater than or equal to zero), the instruction transmits the element directly to the result field. If the exponent of the source element is negative, the instruction right-shifts the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The instruction then transmits the shifted coefficient with zero exponent to the corresponding element of result field C.

The Y and B designators and G bits 4 through 7 are unused and must be zeros. If zero is used as a source element, the instruction transmits all zeros as the corresponding result element.

Figure 6-21 shows an example of a floor A → C (91) operation with one assumed source vector element. Since the exponent of the source element is negative, the instruction right-shifts the coefficient three places and increments the exponent plus three. The sign bits are extended on the left. The result element becomes a minus one. Thus, the floor A → C (91) instruction provides a means of converting positive fractions to zero and negative fractions to a minus one.

The applicable data flag bit is 46 (indefinite result).

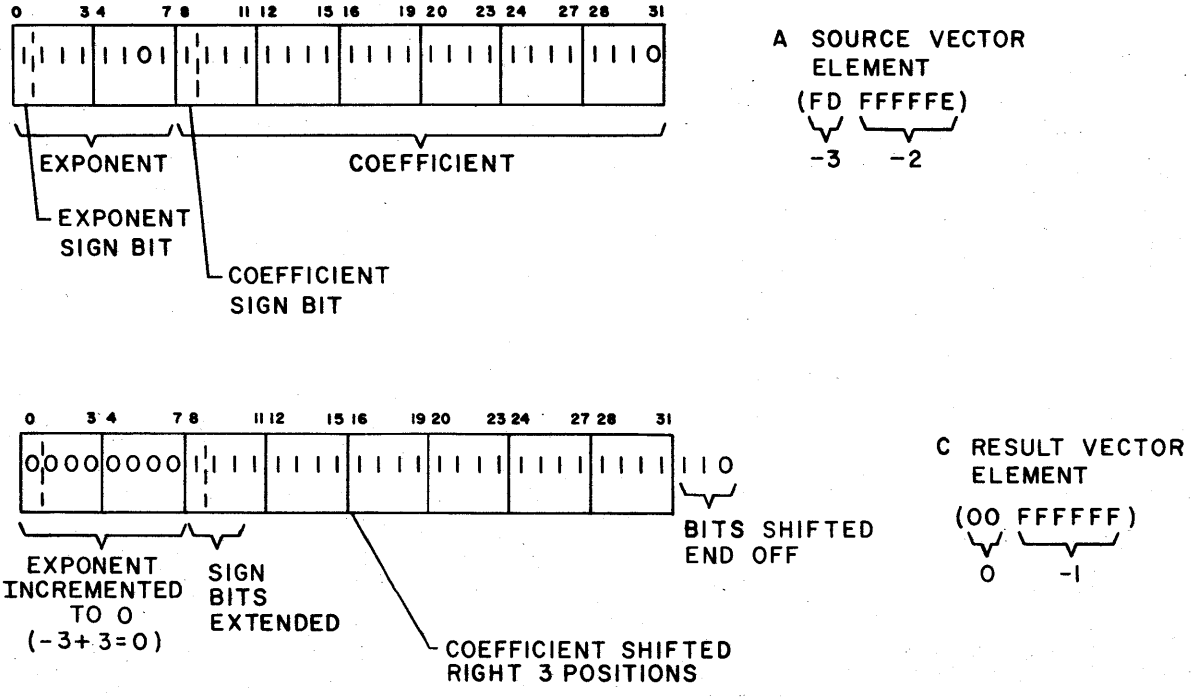


Figure 6-21. Example of Floor A → C Instruction with Negative Exponent

92 CEILING A → C

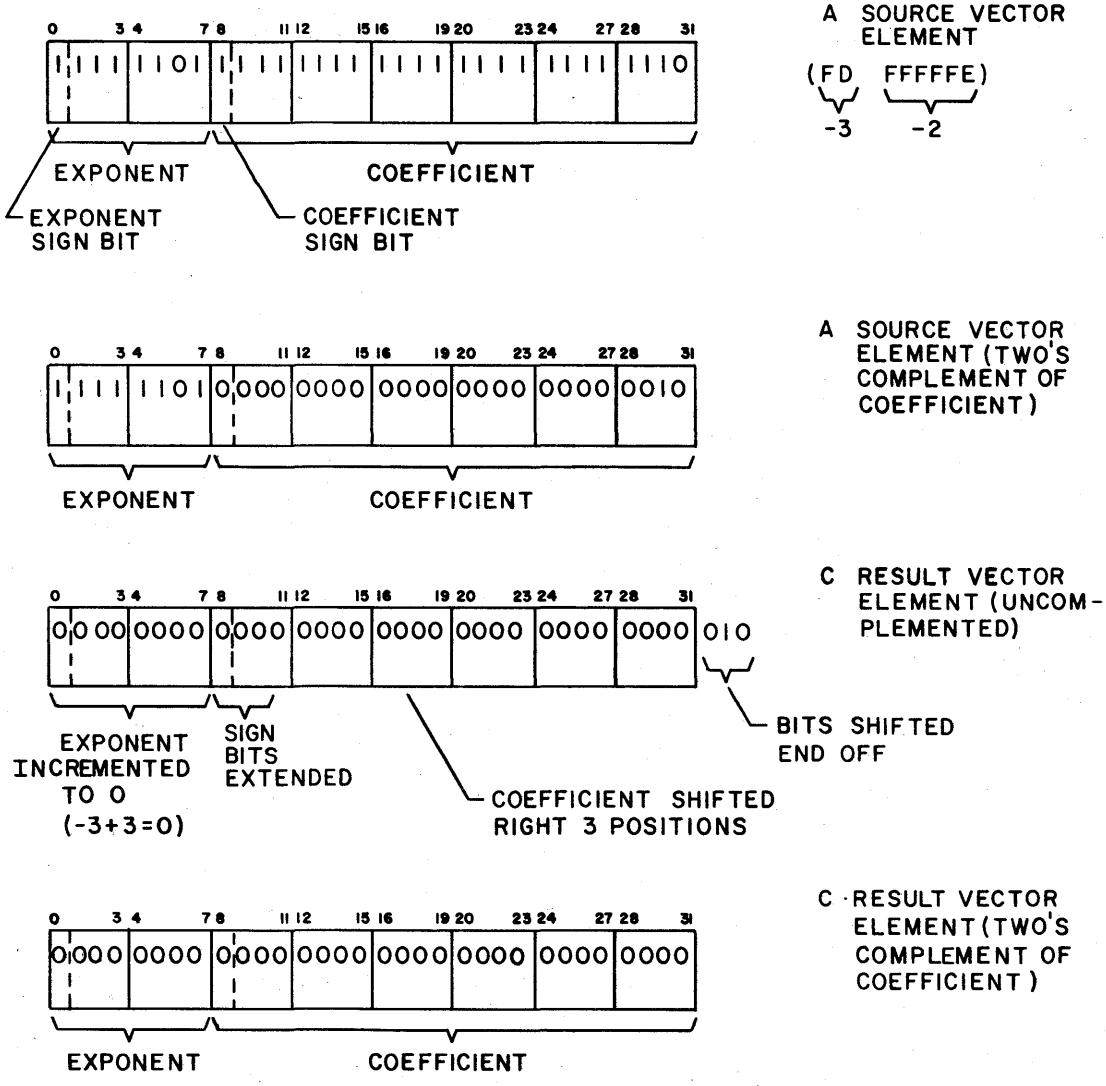
This instruction converts each floating point element of source field A to the nearest integer greater than or equal to it. The resulting integers are transmitted to corresponding elements of result field C throughout the modified field length. The resulting integer is always an unnormalized floating-point number with a positive exponent.

If the exponent of the source element is positive, the instruction transmits the element directly to the result field. If the exponent of the source element is negative, the instruction right-shifts the two's complement of the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The instruction then recomplements the shifted coefficient and transmits it with zero exponent to the corresponding element of the result field.

The Y and B designators and G bits 4 through 7 are undefined and must be zeros. If machine zero is used as a source element, the instruction transmits all zeros as the corresponding result element.

Figure 6-22 shows an example of a ceiling A → C (92) operation with one assumed source vector element. Since the exponent of the source element is negative, the instruction right-shifts the two's complement of the coefficient three places and increments the exponent by plus three. The zero sign bits are extended on the left The result element becomes all zeros. Thus, zero is the closest integer greater than the A source vector element. The ceiling A → C (92) instruction provides a means of converting negative fractions to zero and positive fractions to plus one.

The applicable data flag bit is 46 (indefinite result).



NOTE: 32-BIT OPERANDS ARE ASSUMED.

Figure 6-22. Example of Ceiling A → C Instruction with Negative Exponent

9A EXPONENT OF A → C

The elements of result vector C are formed by storing the exponents from input vector A into the rightmost position of the coefficients of vector C. The sign of the exponent is extended left to the coefficient sign bit position. The exponent portion of each element of vector C is cleared to zero.

The Y and B designators and bits 4 through 7 of the G designator are unused and must be set to zeros.

90 TRUNCATE A → C

This instruction transmits to elements of vector C the nearest integer the magnitude of which is less than or equal to the corresponding elements of source vector A. These integers are represented by unnormalized floating point numbers having positive exponents.

If the origin-operand exponent is positive (greater than or equal to zero), the instruction transmits the source element directly to the corresponding result elements.

If the source-element exponents are negative, the machine right-shifts the magnitude of the corresponding coefficients end-off and increases the exponent by one for each position shifted until the exponent becomes zero.

The operation extends zeros on the left during the shift after complementing if the coefficient is negative. If the coefficient of a source element is positive, the shifted coefficient with zero exponent is transmitted to the corresponding result element. If the coefficient of a source element is negative, the two's complement of the shifted coefficient and zero exponent are transmitted to the corresponding result element. If zeros are transmitted as a source element, zero is also transmitted as the corresponding result element.

Figure 6-23 shows a typical source element and the corresponding result element for a truncate A → C (90) instruction. A 32-bit source element with a positive coefficient and negative exponent is assumed. A right shift of eight is required to reduce the negative exponent to zero.

The applicable data flag bit is 46 (indefinite result).

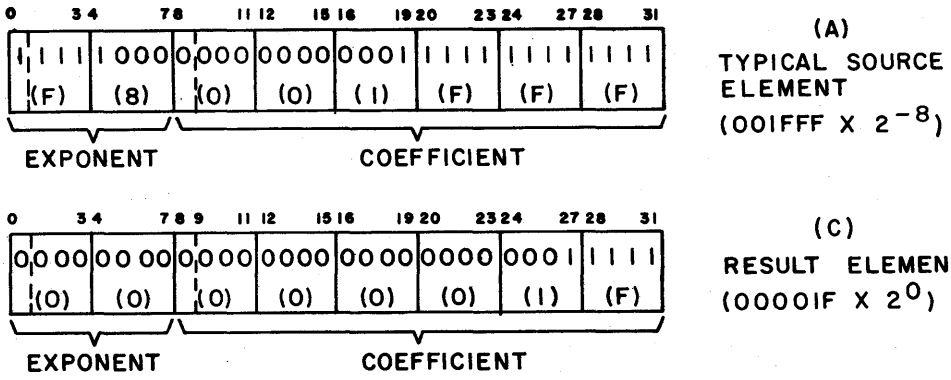
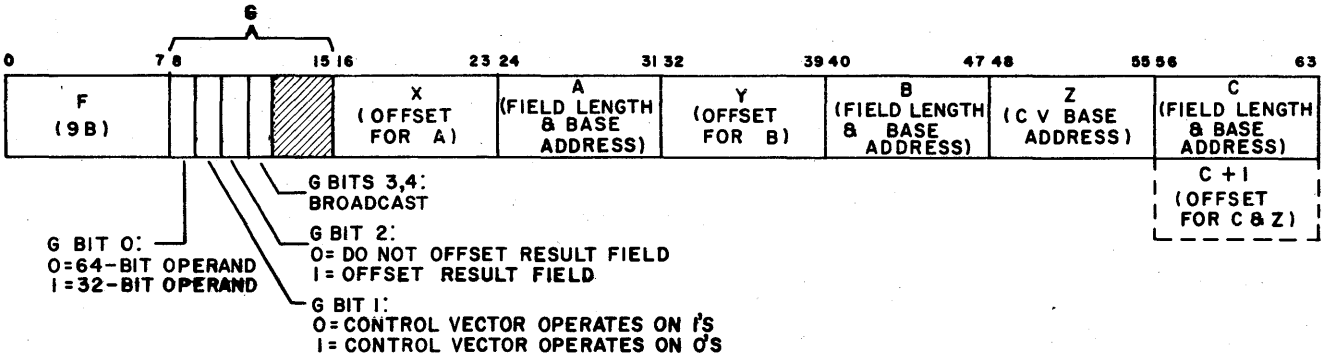


Figure 6-23. Example of Source and Result Elements for a Truncate A → C Instruction

9B PACK A,B→C



This instruction moves an exponent from an element of source vector A and a coefficient from an element of source vector B into the corresponding exponent and coefficient positions of result vector C.

This instruction forms the elements of a floating point result vector C. The elements of result vector C consist of exponents from the rightmost 16 bits (64-bit operands) or 8 bits (32-bit operands) of source vector A elements and coefficients from the rightmost 48 bits/24 bits of the corresponding elements of source vector B.

Figure 6-24 shows an example of an assumed A source and B source vector element used in forming a C result vector element in a pack A, B → C instruction.

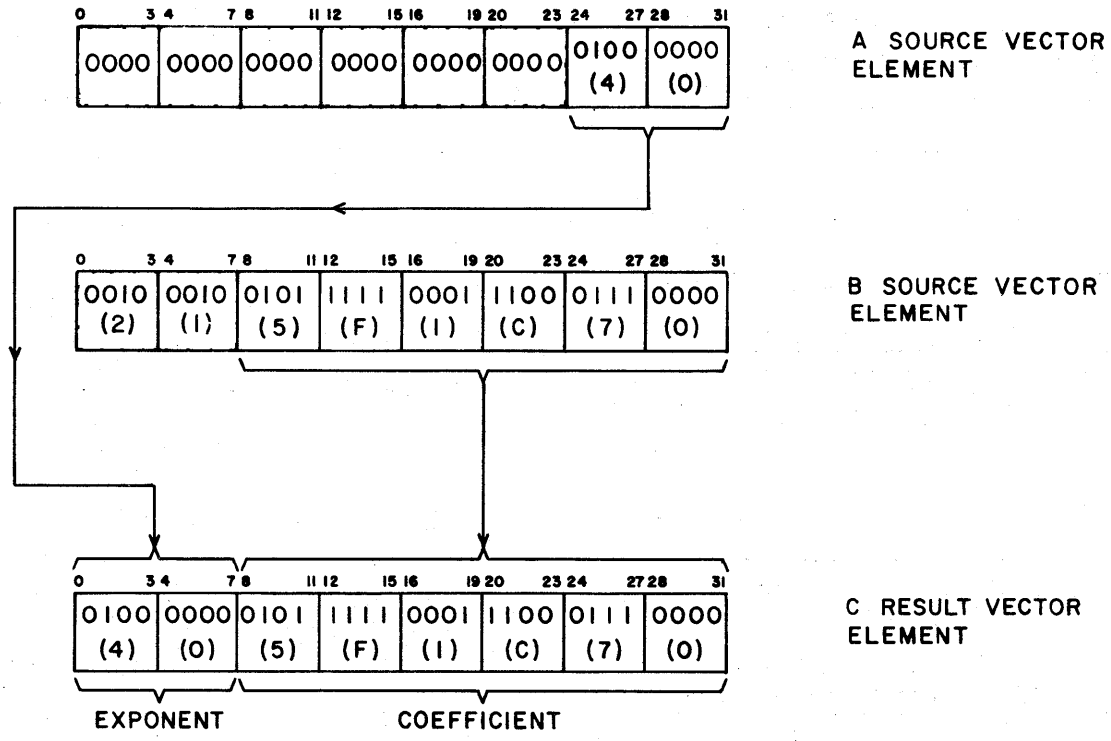


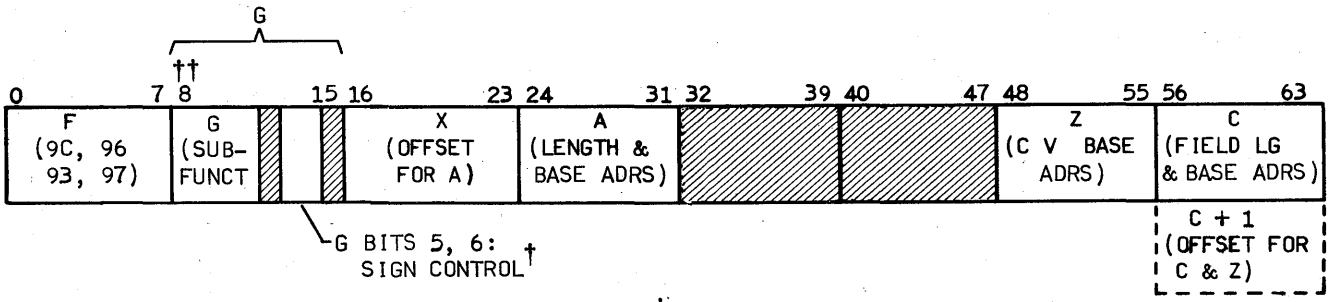
Figure 6-24. Example of Pack A, B → C Instruction

9C EXTEND 32 BIT A→64 BIT C

96 CONTRACT 64 BIT A→32 BIT C

97 ROUNDED CONTRACT 64 BIT A→32 BIT C

93 SIGNIFICANT SQUARE ROOT OF A→C



† IN THIS GROUP OF INSTRUCTIONS, THE SIGN CONTROL BITS ARE USED IN INSTRUCTION 93 ONLY. IN ALL OTHER CASES, THESE BITS MUST BE ZERO.

†† G BIT 0 MUST BE A ZERO FOR THE 9C, 96, AND 97 INSTRUCTION BUT MAY BE A ZERO OR A ONE FOR THE 93 INSTRUCTION.

9C EXTEND 32 BIT A → 64 BIT C

This instruction forms the elements of result vector C by extending the 32-bit, floating point operands of vector field A into 64-bit, floating point operands. The instruction reduces the exponent of the result elements by 24_{10}. The 9C instruction transmits the rightmost 24 bits of the corresponding source elements to bits 16 through 39 of the result elements. The rightmost 24 bits of each result element are cleared.

If an element of vector A is indefinite, the instruction sets the corresponding element of vector C to indefinite and sets data flag bit 46. If an element of vector A is machine zero, the instruction stores machine zero as the corresponding element of vector C and sets data flag bit 43 (result machine zero).

If bit 3 of the G designator is set, indicating broadcast of the A register, the 8-bit A designator is a 32-bit register designator.

Since the instruction uses only one source field, the Y and B designators and bits 0, and 4 through 7 of the G designator are not used. These bits must be zeros.

Figure 6-25 shows an example of the extension of one assumed source element into the corresponding result element. The instruction reduces the exponent of the assumed source element ($4F_{16}$) by 24_{10} to 37_{16}. The sign of the result exponent is extended in bits 0 through 7. The 9C instruction always clears bits 40 through 63 of the result-element coefficients.

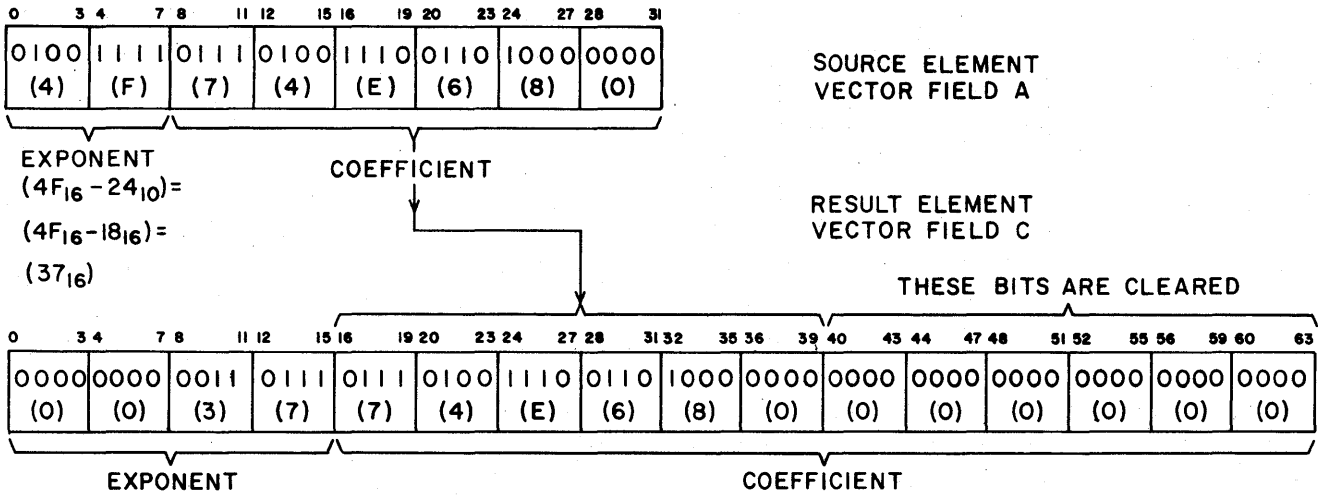


Figure 6-25. Example of Extend 32 Bit A → 64 Bit C Instruction

96 CONTRACT 64 BIT A → 32 BIT C

This instruction contracts each 64-bit, floating-point element of vector field A into its corresponding 32-bit floating point result. The result element becomes the corresponding element of result vector field C. The instruction increases each non-end case source-element exponent by 24_{10} in forming the 8-bit exponent for the result element.

The following is a list of input exponents and the corresponding result of the 96 instruction execution.

Input Exponent	Result
7FFF . .	Result indefinite
7000 6FFF . .	Data flag bit 46 (indefinite result) is set.
0058 0057 . .	Data flag bits 42 (exponent overflow) and 46 (indefinite result) are set. Result exponent is 24_{10} larger than the input exponent. The leftmost 24 bits of the input coefficient are transferred.
FF78 FF77 . .	Result is machine zero. Data flag bit 43 (result machine zero) is set.
8000	

The coefficient of the result element becomes the leftmost 24 bits of the source element coefficient. This operation contracts the coefficients of all elements with an absolute value of less than 2^{24} (neglecting the exponent) to minus one for negative coefficients and zero for positive coefficients.

The Y and B designators and bits 0 and 4 through 7 of the G designator are not used and must be zeros. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

97 ROUNDED CONTRACT 64 BIT A → 32 BIT C

This instruction performs a rounded contract operation on the 64-bit, floating point elements of vector field A and transmits the 32-bit, floating point results to elements of vector field C (figure 6-26). Each resulting 8-bit exponent represents the sum of the least significant eight bits of the source element and 24_{10}. If the result exponent cannot be contained in eight bits, exponent overflow or underflow is detected.

The instruction then adds a plus one to bit positions 40 of the source-element coefficients. If overflow occurs (figure 6-26), the instruction increases the exponent by one and right-shifts the coefficient one place. (Since the result coefficient in figure 6-26 contains all zeros, the example does not show the right-shift of one place.) The leftmost 24 bits of the shifted result coefficient are transmitted to the corresponding bits of result element C. The exponent of each non-end case result element is 24_{10} (25_{10} if overflow occurred) greater than the exponent of the corresponding source element.

The Y and B designators and bits 0 and 4 through 7 of the G designator are not used and must be zeros. Data flag bits 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result) conditions are probed by the execution of this instruction.

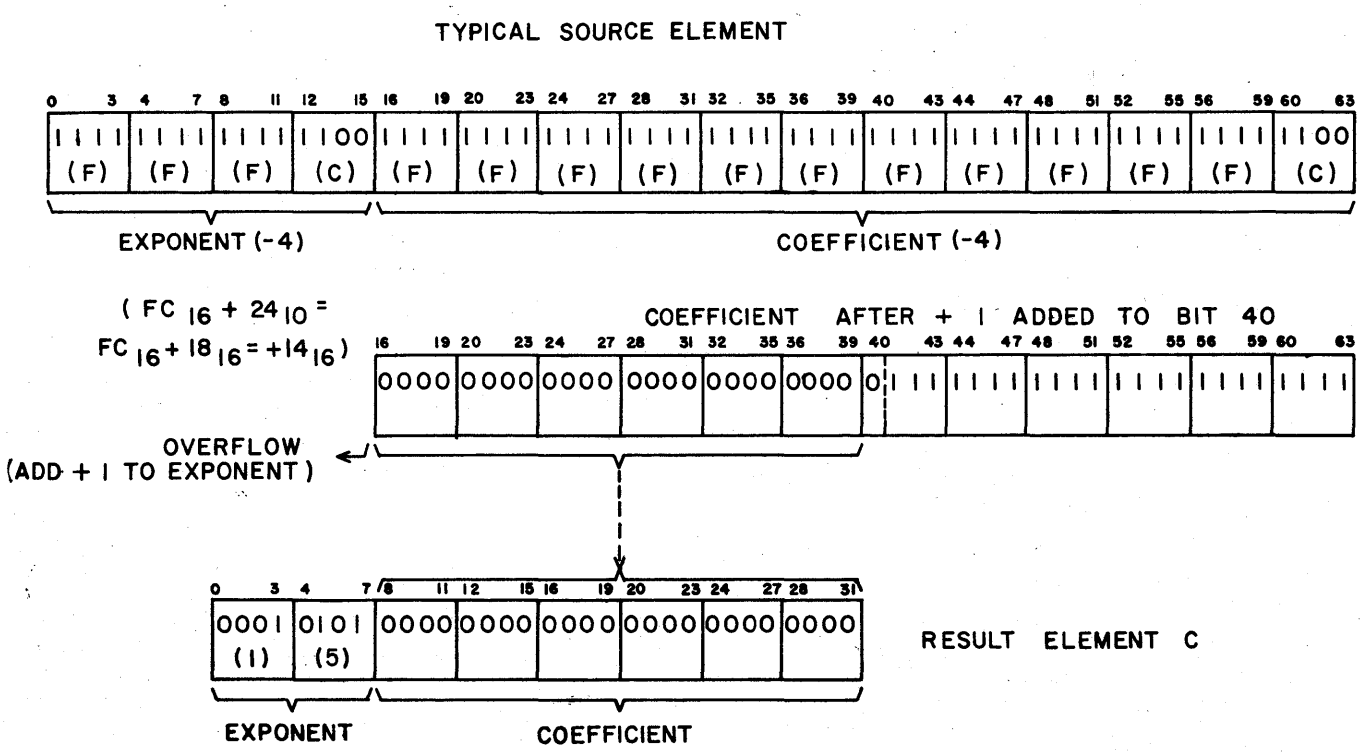


Figure 6-26. Example of Vector Elements for a Rounded Contract 64-Bit AC→32-Bit C Instruction

93 SIGNIFICANT SQUARE ROOT OF A →C

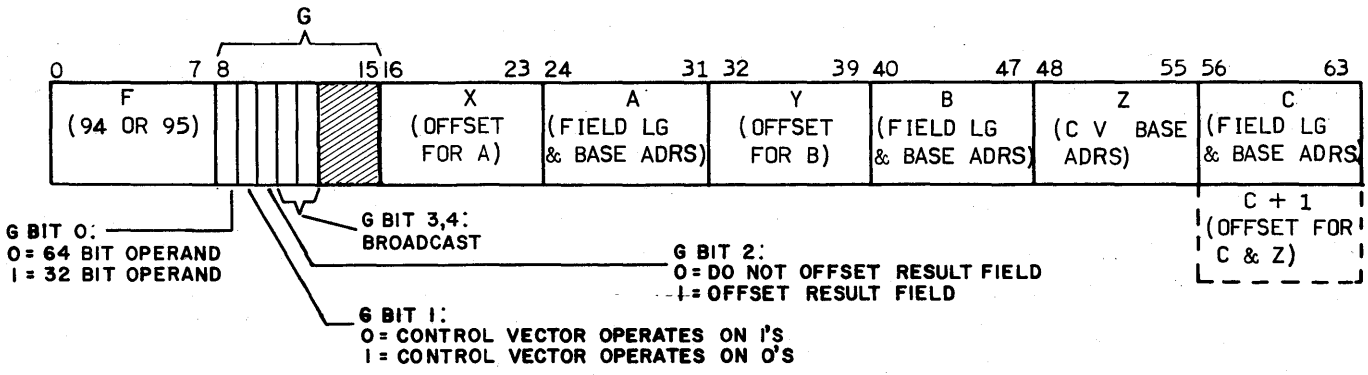
This instruction forms the square root† of each element of vector field A and places the result in each corresponding element of vector field C. Each result element contains the same number of significant bits as the corresponding source element.

Since the instruction uses only one source field, the Y and B designators and bits 4 and 7 of the G designator are not used and must be zeros. Bits 5 and 6 of the G designator perform sign control functions as given in table 6-17. Applicable data flag bits are 43 (result machine zero), 45 (square root result imaginary), and 46 (indefinite result).

† Appendix B describes the floating-point square root operation.

94 ADJUST SIGNIFICANCE OF A PER B ➔ C

95 ADJUST EXPONENT OF A PER B ➔ C



94 ADJUST SIGNIFICANCE OF A PER B → C

This instruction adjusts the significance† of floating point elements from vector field A and transmits the adjusted elements to corresponding elements of vector field C. The rightmost 48 (64-bit operands)/24 (32-bit operands) bits of the elements in vector field B contain signed, two's complement integers. The absolute values of these integers are shift counts.

If a shift count is positive, the instruction left-shifts the coefficient of the element from vector field A the number of positions specified by the shift count or by the number of positions necessary to normalize the coefficient, whichever is smaller. In either case, the instruction reduces the exponent of the source element by one for each position shifted. The instruction left-shifts an all zero coefficient by the specified number of positions.

If a shift count is negative, the instruction right-shifts the coefficient of the source element by the shift count. The instruction increases the exponent by one for each position shifted. If the absolute value of the shift count is greater than 47_{10}, the shift operation is undefined. The addition of the shift count can cause either exponent overflow or underflow.

If the source element is indefinite, the instruction sets the corresponding result element to indefinite and sets data flag bit 46 (indefinite result). If the source element is machine zero, the instruction sets the corresponding result element to machine zero (result machine zero) and sets data flag bit 43. Data flag bit 42 (exponent overflow) is also applicable.

†Appendix B describes the operation of adjusting floating-point operands.

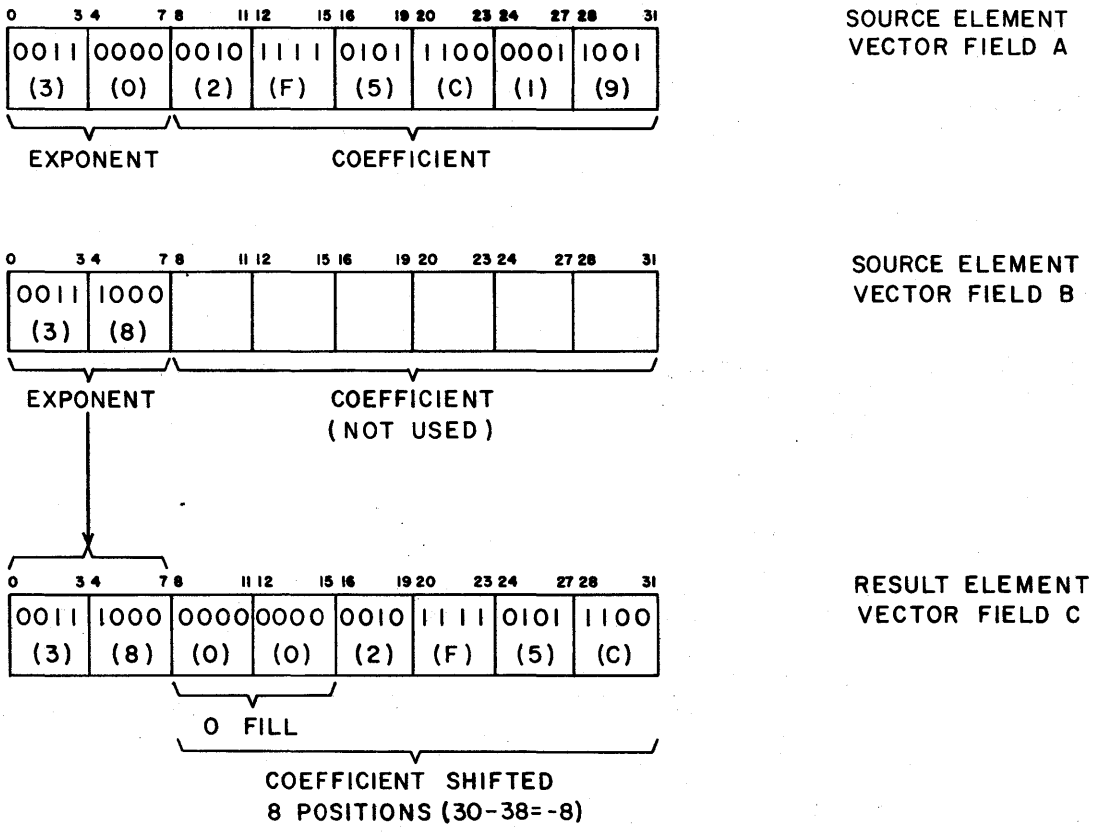
95 ADJUST EXPONENT OF A PER B→C

This instruction transmits adjusted source elements from vector field A to corresponding result elements in vector field C. The instruction sets the exponent of a result element equal to the exponent of the associated source element in vector field B. The coefficients of the result elements are formed by shifting the coefficients of the source elements from vector field A.

The difference between the exponents of associated elements from vector fields A and B forms the shift count. If the exponent from A is greater/less than the exponent of the element from B, the shift is to the left/right, respectively. If A contains a zero coefficient, the exponent of the corresponding element of B is transferred to the corresponding element of C with an all zero coefficient. If a left shift exceeds the number of positions required for normalization, the corresponding result element is set to indefinite, and data flag bit 42 (exponent overflow) is set.

If either or both source elements are indefinite or machine zero, the instruction sets the result element to indefinite. In this case, data flag bit 46 (indefinite result) is set and data flag bit 42 (exponent overflow) is not set.

Figure 6-27 shows one adjust exponent of A per B→ C operation with assumed 32-bit source elements for vector fields A and B. The exponent of the source element in vector field B is greater than the source element from field A by eight. As a result, the instruction right-shifts the coefficient eight positions end-off. The vacated positions on the left are filled with zeros.



NOTE: 32-BIT OPERANDS ARE ASSUMED.

Figure 6-27. Example of Adjust Exponent of A Per B→C Operation

SPARSE VECTOR INSTRUCTIONS

Arithmetic operations may reduce many elements of a vector field to a zero or near-zero value. Except for positional significance, the near zero values need not occupy storage locations as floating point operands in the vector field. In order to conserve storage space and calculating time, the sparse vector instructions make possible the expansion and compression of vectors of this type into sparse vectors.

A sparse vector consists of a vector pair [one of which is a bit string, identified as an order vector, and the other is a floating point array (32- or 64-bit) identified as the data vector]. Sparse order vectors determine the positional significance of the segments of the corresponding sparse data vector.

Typically, a sparse vector is formed by the following procedure.

1. The compare instructions generate an order vector.
2. The compress A → C per Z (BC) instruction reduces the corresponding vector to a sparse vector.
3. The BC instruction uses the generated order vector as a means of discarding all near-zero elements and still maintain their positional significance through the order vector.

Figure 6-28 shows an example of compressing an initial vector into a sparse vector. Initial vector elements A_0 through A_8 are contained in consecutive, half-word addresses, beginning at arbitrary address m. A compare instruction first generates an order vector from the initial vector. The compare instruction sets the bits in the order vector corresponding to vector elements that are to be retained in the data vector. Conversely, zeros in the order vector designate the near zero elements that are to be discarded in the sparse vector field.

The compress A → C per Z instruction stores the vector elements in consecutive addresses of the data vector corresponding to ones in the order vector. Thus, the initial vector is now represented or the sparse vector consisting of the order vector and data vector.

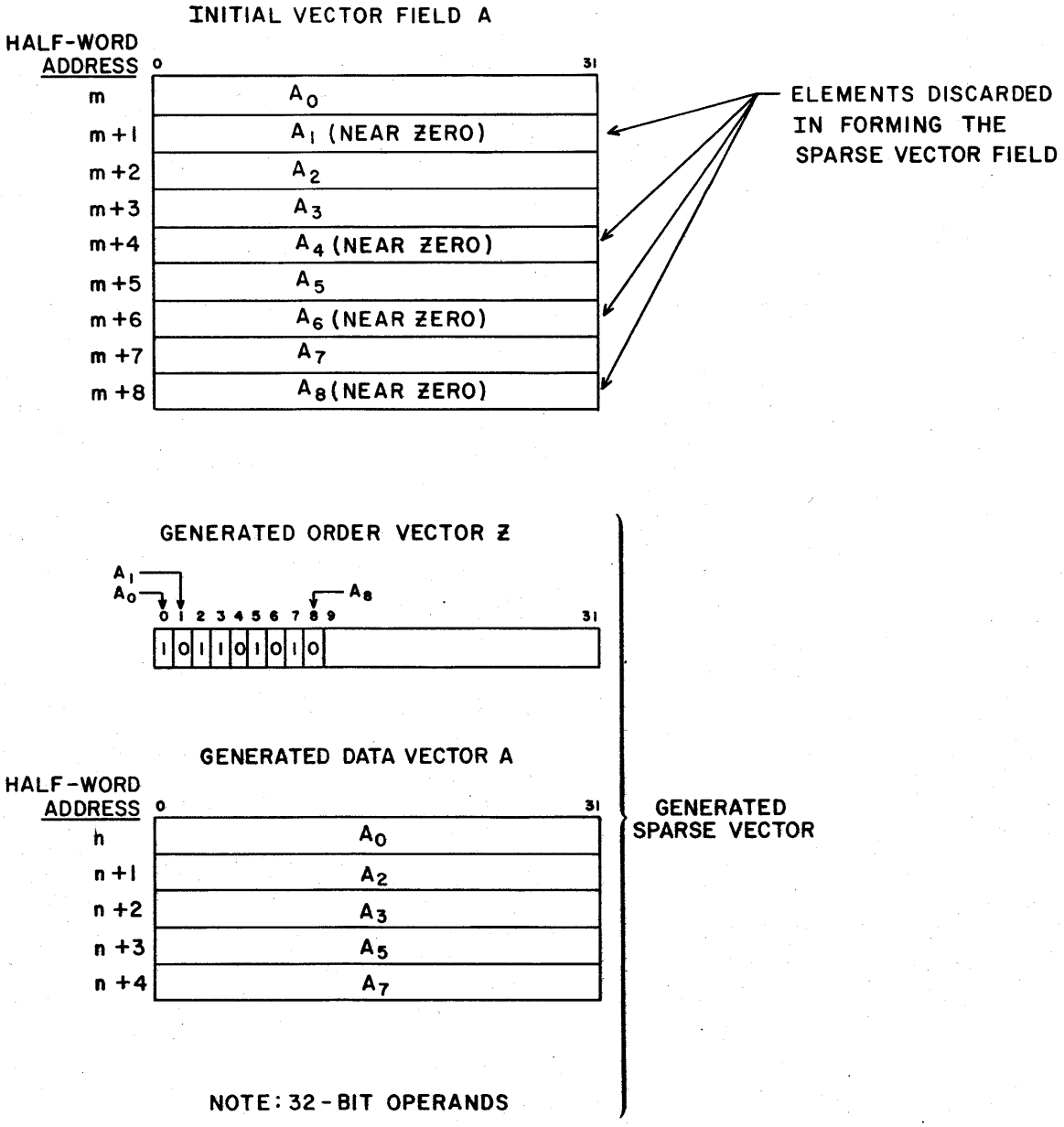


Figure 6-28. Example of Compressing Initial Vector Field into Sparse Vector Field

SPARSE VECTOR INSTRUCTION FORMAT

All sparse vector instructions use the same general format as shown in figure 6-29. Table 6-18 lists each of the 8-bit designator portions of the sparse vector instruction format and the corresponding definition.

BASE ADDRESSES AND FIELD LENGTHS

Figure 6-30 shows that the base addresses and field lengths for the sparse data vectors are the same format as the corresponding field lengths and base addresses of the normal vectors. However, the field lengths associated with source sparse data vectors are not used; thus, figure 6-30 shows bits 0 through 15 of the registers designated by A, B, and C as not used. The field lengths for these vectors are determined by the number of ones in the corresponding order vectors. The field lengths of the source order vectors (X and Y) and the result order vector (Z) are item counts in bits. The addresses to these order vectors are bit addresses.

SPARSE VECTOR INSTRUCTION TERMINATION

Sparse vector instructions terminate when the result order vector, as defined by corresponding field length, is filled. If the Z designator is zero or if the Z field length is zero, the instructions set no data flag bits and become no-operation (no-op) instructions. The sparse vector instructions terminate differently from the vector or vector macro instructions.

Source order vectors with a zero or short field length are extended with zeros as required. If vector Z contains a nonzero field length and the C designator is zero, the results of the instruction are undefined.

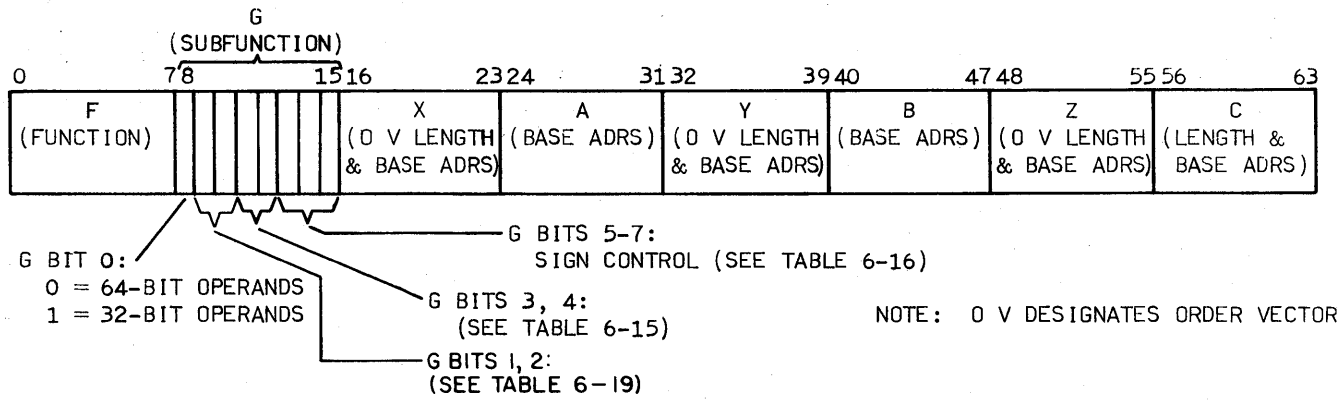


Figure 6-29. General Sparse Vector Instruction Format

TABLE 6-18. SPARSE VECTOR INSTRUCTION DESIGNATORS

8-Bit Designator	Definition
F	Instruction code
G	Suboperation code; the state of G bit 0 denotes the following: State — Designation 0 — 64-bit operands 1 — 32-bit operands G bits 1 and 2 are as defined by table 6-19. When bit 3 is set, the function is broadcast A. When bit 4 is set, the function is broadcast B. G bits 5 through 7 function as sign control bits (refer to table 6-17). †
X, Y	Specify the register that contains the base address and field length of the source order vector associated with source sparse data vectors A and B, respectively
A, B	Specify the register that contains the base address of the corresponding source sparse data vector
C	Specifies the register that contains the base address of the result sparse data vector
Z	Specifies the register that contains the base address and the field length of the result sparse order vector associated with result sparse data vector C

†Appendix C provides a composite listing of the G designator bits usage according to function code.

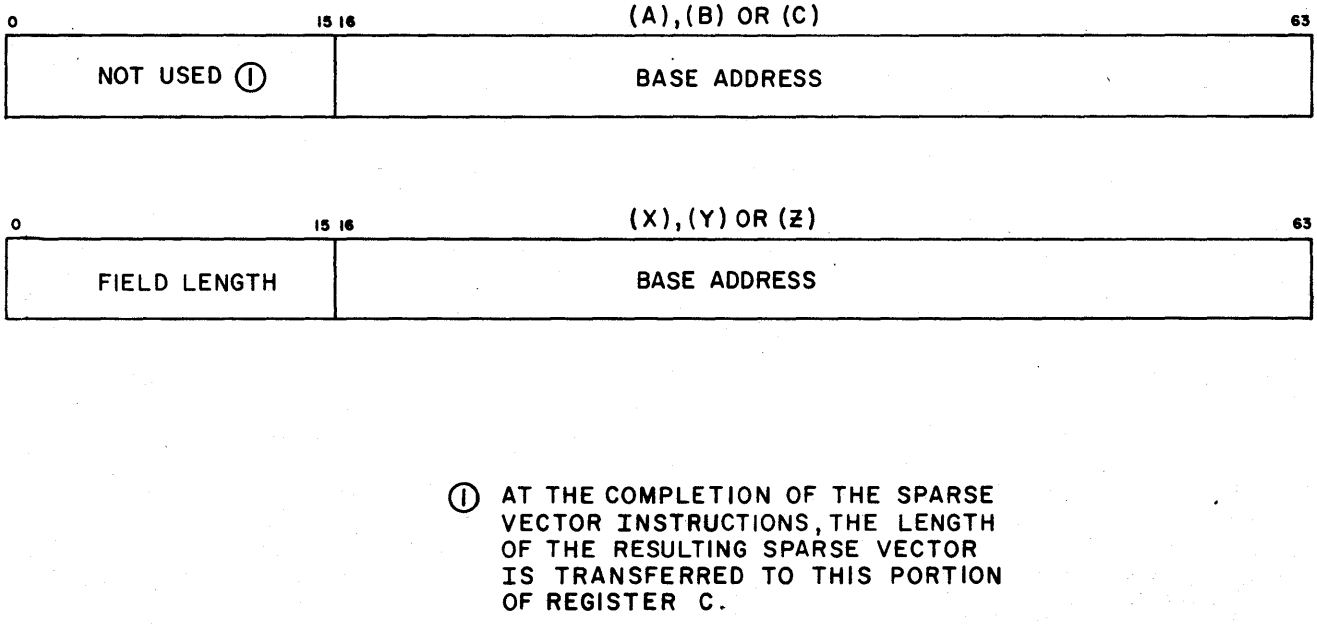


Figure 6-30. Sparse Vector Field Length and Base Address Formats

INSTRUCTIONS A0 THROUGH AF

These instructions have different forms depending on G bits 1 and 2. Table 6-19 shows the operations associated with the values assigned to G bits 1 and 2.

TABLE 6-19. G BIT 1 AND 2 OPERATIONS

G Bit 1	G Bit 2	Operation
0	0	Normal order vector generation (logical OR for ADD/SUB, logical AND for MULT/DIV)
0	1	Reverse logical operation (AND instead of OR for ADD/SUB, OR instead of AND for MULT/DIV)
1	0	Exclusive OR
1	1	Implication

A0 ADD U; A + B→C

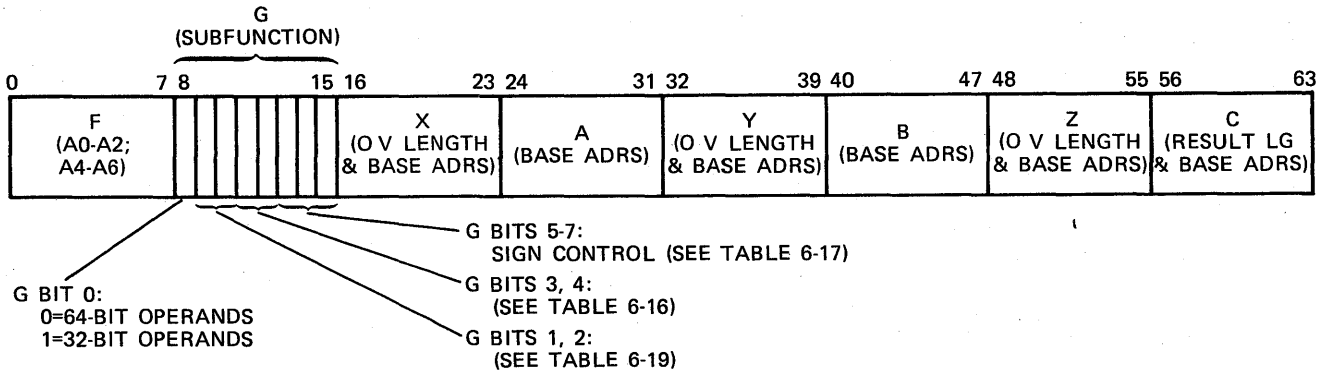
A1 ADD L; A + B→C

A2 ADD N; A + B→C

A4 SUB U; A − B→C

A5 SUB L; A − B→C

A6 SUB N; A −B→C



These instructions perform the indicated floating-point operations on elements of sparse data vectors A and B. The instructions return the results to elements of sparse data vector C. The instructions read an element from sparse data vector A and/or B when the corresponding sparse order vector X and/or Y contains a one in the associated bit position. A zero in a source order vector causes machine zero to be used as the associated A and/or B element. The instructions generate an element in the C field when a one is in the corresponding bit position of order vector X and/or Y. Each bit position of order vector Z is the bit-by-bit inclusive OR of order vectors X and Y. The instruction transfers the resulting field length of sparse vector C to bits 0 through 15 of register C.

In the previous sparse vector instructions, U, L, and N denote that upper, lower, and normalized floating-point† results are generated, respectively. Applicable data flag bits for the sparse vector instructions are 42 (exponent overflow), 43 (exponent underflow), and 46 (indefinite operand). However, the instructions set the data flag bits only when an element is actually stored in the result vector.

Figures 6-31 and 6-32 show examples of an add U; A + B → C sparse vector instruction operation with assumed register contents and vector address fields for specific values of G bit 1 and 2. Although an A0 instruction is used in the examples, the general execution sequence is the same for all the previous instructions. The dashed lines in figures 6-31 and 6-32 connect the elements of the sparse data vector with the corresponding order vector bits. The results of the logical operations for instructions A0 through A6 are shown in table 6-19.1.

TABLE 6-19.1. RESULTS OF THE LOGICAL OPERATIONS (A0 THROUGH A6)

Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR	G Bit 1 = 0 G Bit 2 = 1 AND	G Bit 1 = 1 G Bit 2 = 0 Exclusive OR	G Bit 1 = 1 G Bit 2 = 1 Implication
X	Y	A	B				
0	0	MZ	MZ	N	N	N	MZ
0	1	MZ	B	±B	N	±B	N
1	0	A	MZ	A	N	A	A
1	1	A	B	A±B	A±B	N	A±B

NOTES:

A A stream operand
B B stream operand
N No result produced
MZ Machine zero

†Appendix B describes the normalized floating-point operations.

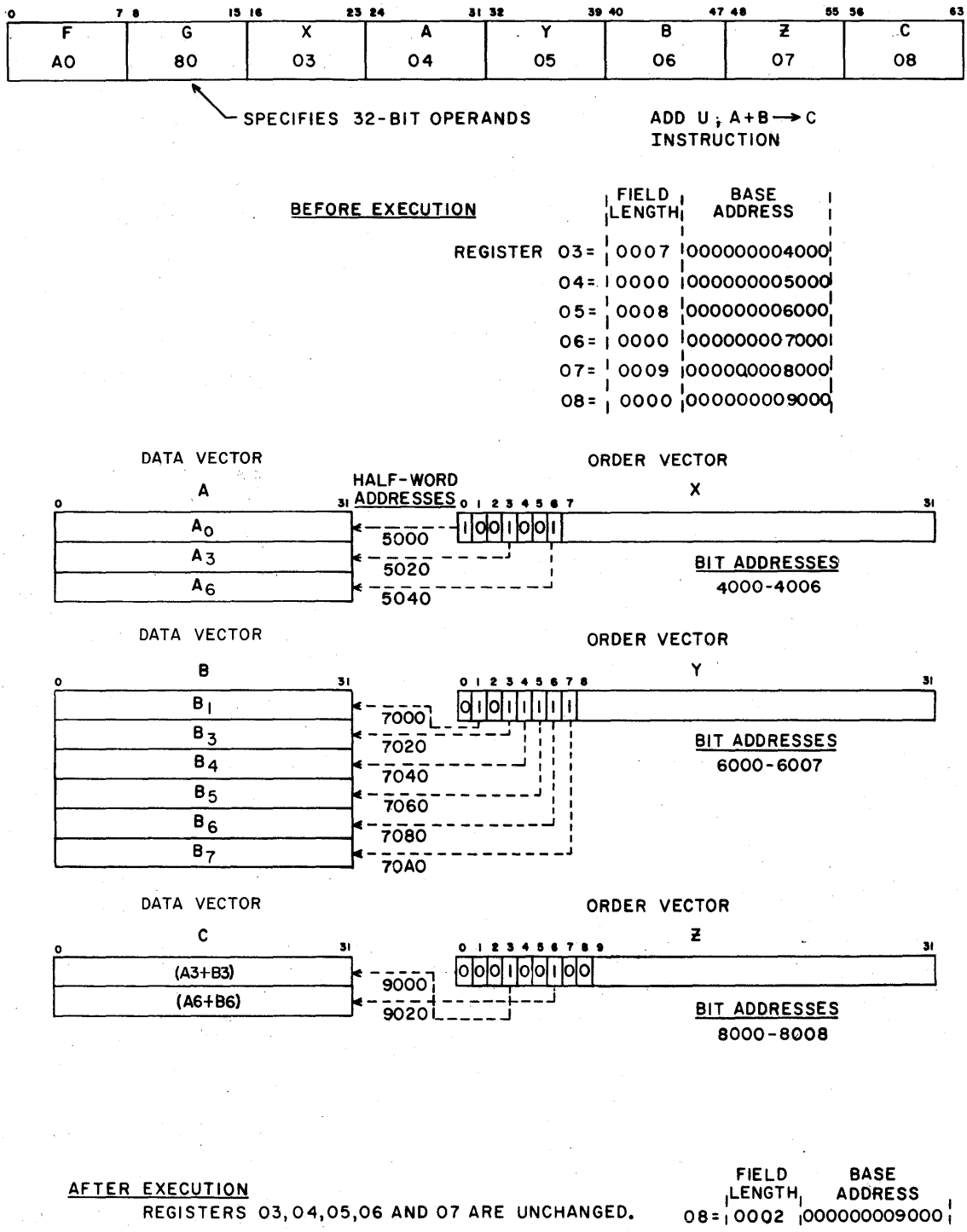


Figure 6-31. Example of an Add U; A + B →C Sparse Vector Instruction when G Bit 1 = 0 and G Bit 2 = 1

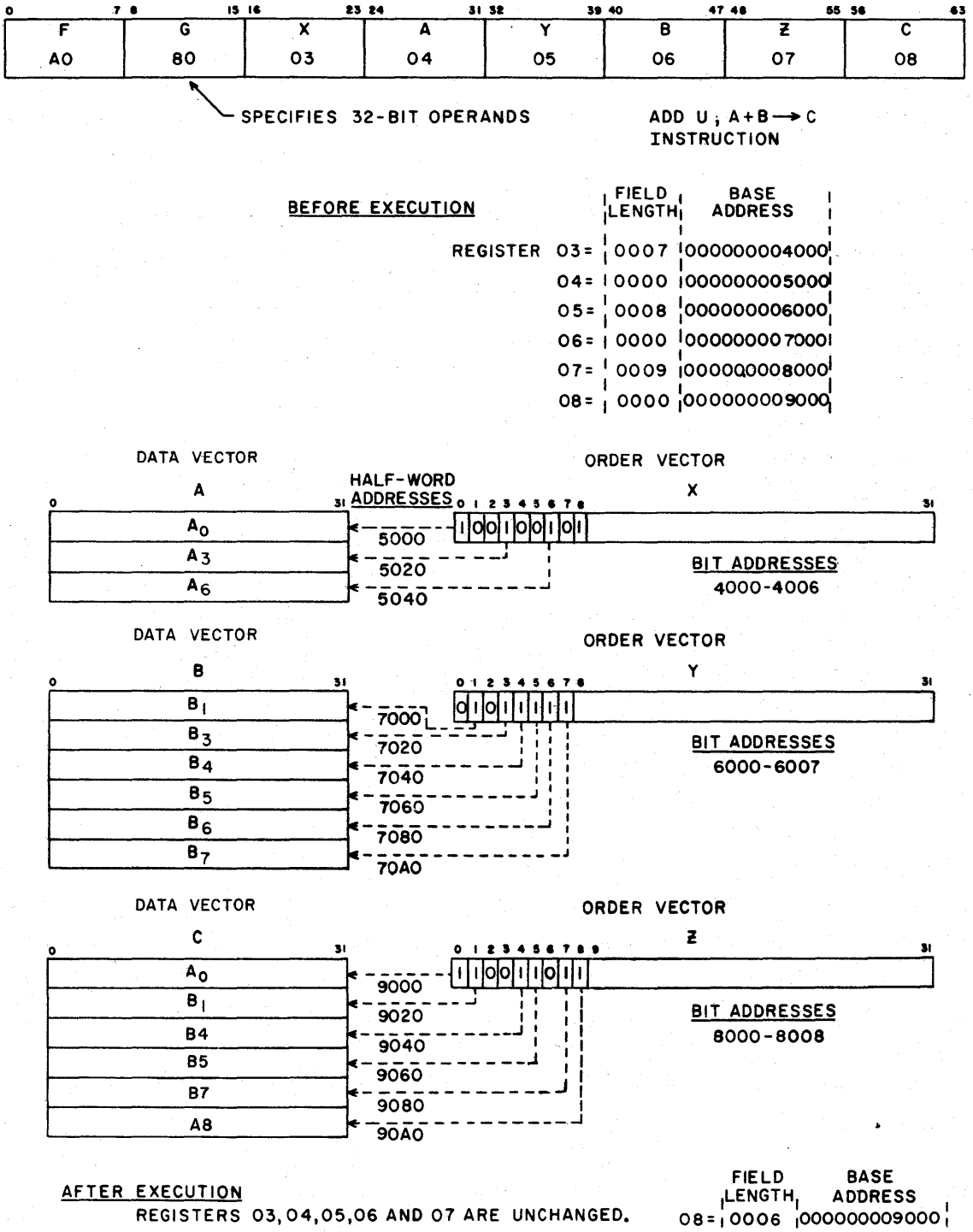
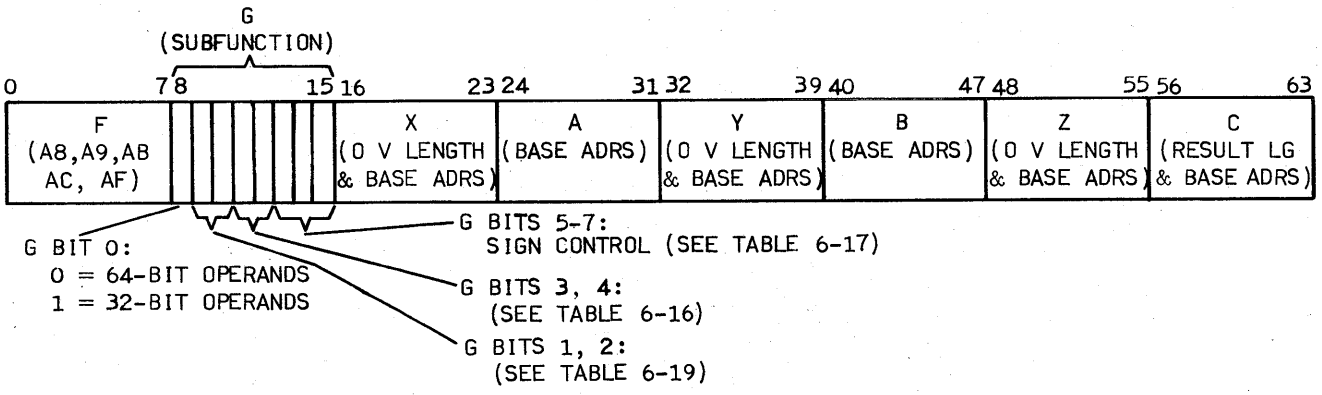


Figure 6-32. Example of an Add U; A + B →C Sparse Vector Instruction when G Bit 1 = 1 and G Bit 2 = 0

In an A0 instruction operation, an actual addition of an element from data vector A to an element from data vector B takes place only when the corresponding source order vector bits are both ones. For example, the $A_3 + B_3$ addition takes place because bit 3 of X and Y order vectors is a one. In cases where a source order vector bit is a one and the corresponding bit for the other source order vector bit is a zero, machine zero is essentially added to the sparse vector element.

At the end of the sparse vector operation, the resulting output data vector length is inserted in the corresponding portion of the register designated by C. In the example, the instruction transfers a 0007_{16} to the leftmost 16 bits of register 08. The 0007 denotes the number of elements in the result data vector C.

A8 MPY U; A • B➔C
A9 MPY L; A • B➔C
AB MPY S; A • B➔C
AC DIV U; A/B➔C
AF DIV S; A/B➔C



These instructions perform the indicated floating-point†, multiply, and divide operations on elements of sparse data vectors A and B. The instructions store the results in elements of sparse data vector C. The instructions read an element from vector A and/or B if the bit position of the corresponding order vector X and/or Y is a one. An element is generated for sparse data vector C when both the X and Y order vectors contain a one in the corresponding bit position. Result order vector is the bit-by-bit, logical AND of order vectors X and Y.

†Appendix B describes the floating point arithmetic operations.

In the sparse vector instructions previously listed, U, L, and S denote that upper, lower, and significant upper floating-point results are generated, respectively. Applicable data flag bits for the multiply and divide sparse vector instructions are 41 (floating-point divide fault), 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). However, the instructions set the data flag bits only when an element is actually stored in the result vector.

Figures 6-33 and 6-34 show examples of multiply U; A • B → C sparse vector instruction operation with assumed register contents and vector address fields with specific values for G bits 1 and 2. Although an A8 instruction is used, the general execution sequence is the same for all instructions of this type. Dashed lines connect the elements of the sparse data vector with the corresponding order vector bits.

In an A8 operation, an actual product is generated as an element of data vector C only when the corresponding order vector bits for the A and B data elements are both ones. In cases where one or both of the source order vector bits are zero, no multiplication takes place, and the corresponding result order vector bit is cleared. In Figures 6-33 and 6-34, only three products are generated by the instruction (A_3 • B_3), (A6 • B_6), and (A_7 • B_7).

At the end of the sparse vector operations, the resulting output data vector length is inserted in the corresponding portion on the register designated by C. In the example, the instruction transfers a 0003 to the leftmost 16 bits of register 09. The 0003 denotes the number of elements in result data vector C. The results of the logical operations for instructions A8 through AF are shown in tables 6-19.2 and 6-19.3.

TABLE 6-19.2. RESULTS OF THE LOGICAL OPERATIONS (A8 THROUGH AB)

Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 1 OR	G Bit 1 = 0 G Bit 2 = 0 AND	G Bit 1 = 1 G Bit 2 = 0 Exclusive OR	G Bit 1 = 1 G Bit 2 = 1 Implication
X	Y	A	B				
0	0	MZ	MZ	N	N	N	MZ
0	1	MZ	B	MZ	N	MZ	N
1	0	A	MZ	MZ	N	MZ	MZ
1	1	A	B	A*B	A*B	N	A*B

NOTES:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

TABLE 6-19.3. RESULTS OF THE LOGICAL OPERATIONS (AC, AF)

Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 1 OR	G Bit 1 = 0 G Bit 2 = 0 AND	G Bit 1 = 1 G Bit 2 = 0 Exclusive OR	G Bit 1 = 1 G Bit 2 = 1 Implication
X	Y	A	B				
0	0	MZ	MZ	N	N	N	IND
0	1	MZ	B	MZ	N	MZ	N
1	0	A	MZ	IND	N	IND	IND
1	1	A	B	A/B	A/B	N	A/B

NOTES:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero
IND	Indefinite

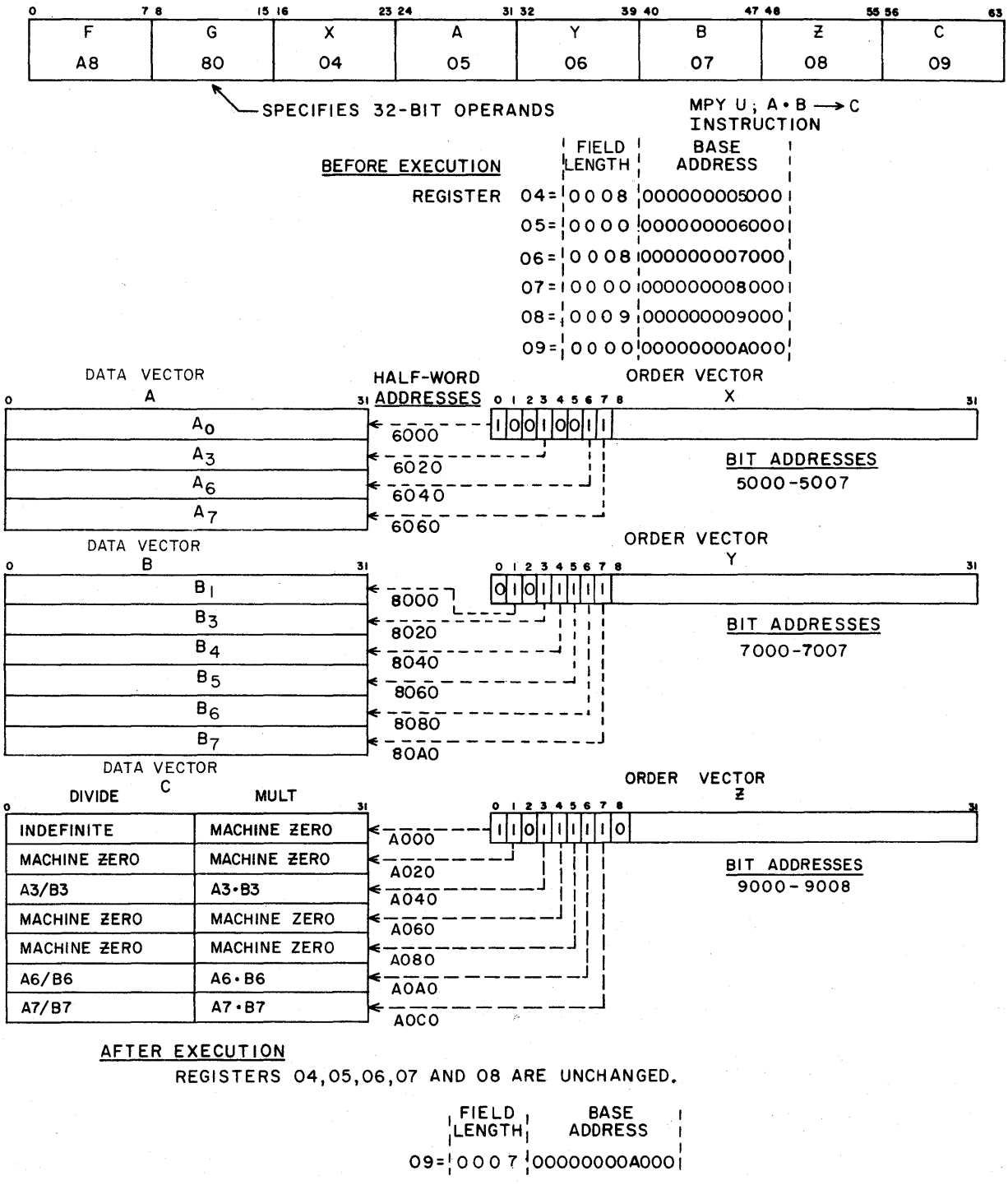
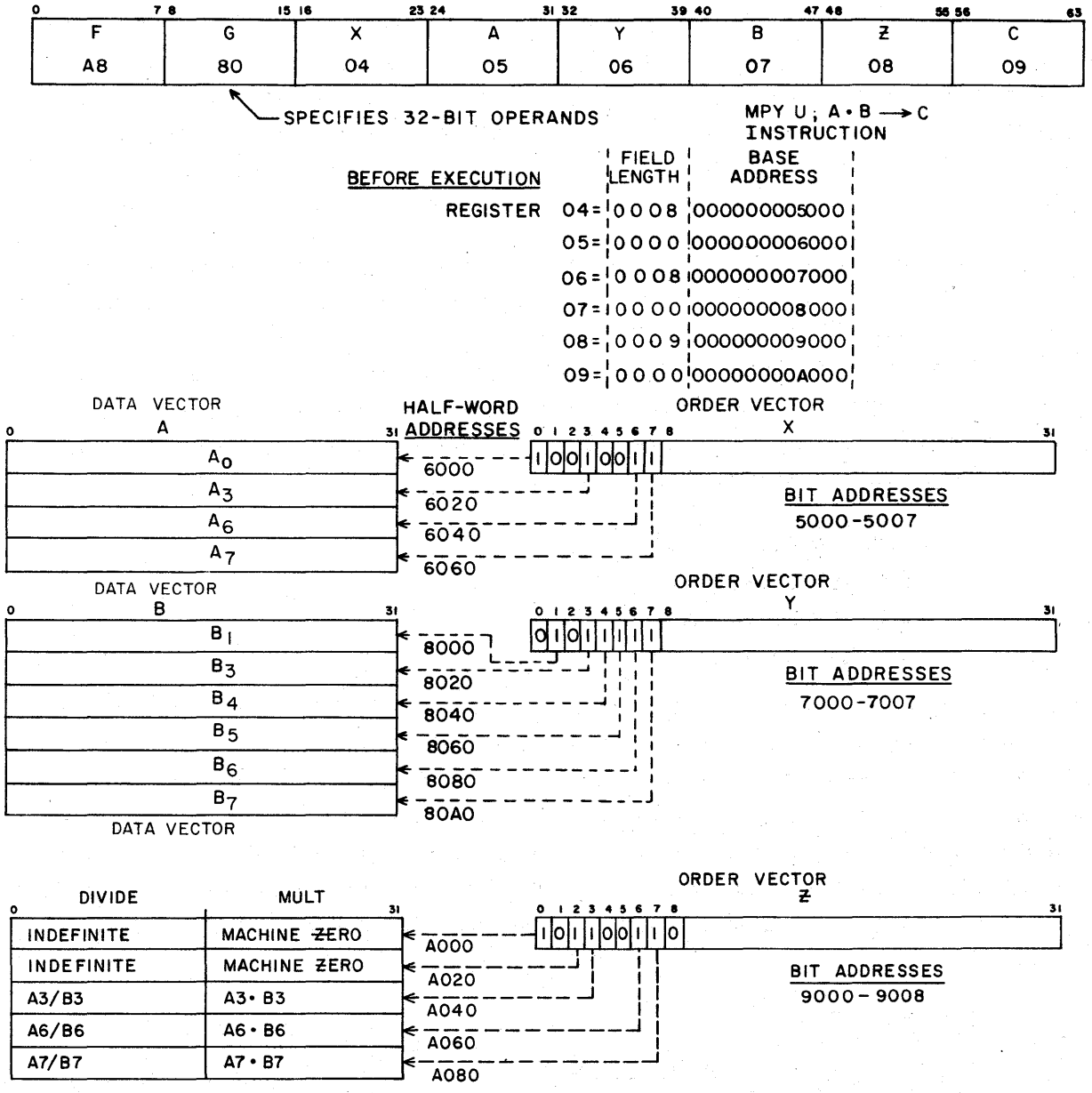


Figure 6-33. Example of a Div or Mpy U Sparse Vector Instruction when G Bit 1 = 0 and G Bit 2 = 1



AFTER EXECUTION

REGISTERS 04,05,06,07 AND 08 ARE UNCHANGED.

	FIELD LENGTH	BASE ADDRESS
09=	0005	00000000A000

Figure 6-34. Example of a Div or Mpy U Sparse Vector Instruction when G Bit 1 = 1 and G Bit 2 = 1

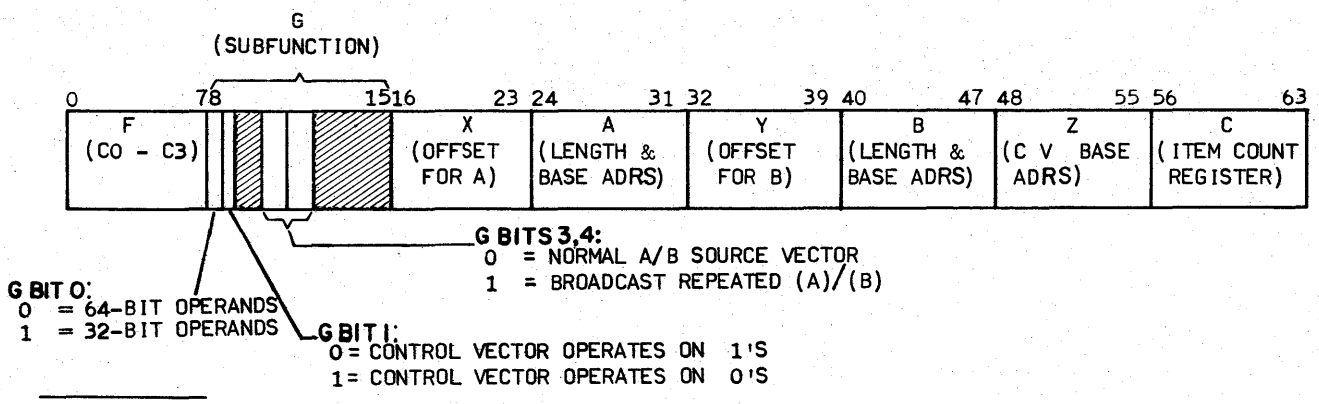
VECTOR MACRO INSTRUCTIONS

Vector macro instructions perform operations similar to vector instructions. However, some vector macro instructions do not form result vector fields, but store the results in one or two registers which are specified by the instruction. In these instructions, the control vector contains neither length nor offset, but controls the use of elements of the source vectors. Bit 2 of the G designator is undefined and must be a zero. Designators C and C + 1 denote 32-bit registers when bit 0 of the G designator† specifies 32-bit operands. In the vector macro instructions that produce result vector fields, the control vector performs the same function as in the vector instructions.

Vector macro instructions with result fields (as opposed to result registers) extend short source fields with machine zeros or normalized ones and terminate in an identical fashion to the vector instructions. The other vector macro instructions do not extend short source vectors but terminate when either source vector is exhausted. For instructions of this type, broadcasting both source fields causes an undefined condition to exist. Appendix C gives a complete listing of the various field conditions and the resulting termination condition.

C0 SELECT EQ; A = B, ITEM COUNT TO(C)
C1 SELECT NE; A≠B, ITEM COUNT TO(C)
C2 SELECT GE; A≥B, ITEM COUNT TO(C)
C3 SELECT LT; A<B, ITEM COUNT TO(C)

These instructions compare each element of vector field A with its corresponding element of vector field B by subtracting vector B from vector A. The conditions for comparing floating point operands are described in the Floating-Point Compare Rules, appendix B. The comparing operation proceeds until the compare condition is met (for a pair of elements not inhibited by the corresponding bit of the control vector) or the shorter of the two vector fields is exhausted. If broadcast is selected for field A or B (but not both), the instruction will terminate when the nonbroadcast field terminates.



†Appendix C provides a comprehensive listing of the G designator bits usage according to function code.

If the compare condition is met, the item count equals the number of pairs of elements encountered up to (but not including) the pair meeting the specified condition, including the pairs inhibited by the control vector. If the compare condition is not met, the item count equals the length of the shorter vector after the offset adjustment. The instruction stores the item count into the rightmost 48 bits of a cleared register C. †

The control vector, if used, determines which pairs of elements are compared. For example, if G designator bit equals zero, a one bit in the control vector enables the comparison of the corresponding pair of source elements. A zero bit in a control vector disables the comparison of the corresponding pair of source elements. The item count, as previously described, includes all pairs of elements encountered, including the pairs for which the comparison was inhibited. If a control vector is used and either source vector A or B is exhausted before a permissive control vector bit is encountered, the instruction makes no comparisons. In this case, the item count represents the length of the shorter vector field minus the offset. Applicable data flag bits are 37 (select condition not met) and 46 (indefinite result).

Figure 6-35 shows an example of a select EQ; A=B; item count → C(C0) instruction with assumed instruction codes, register contents, and vector fields. The G designator specifies 32-bit operands and broadcast source vector A_0. Since the B offset equals 3, the first comparison takes place between source element B_3 and broadcast vector A_0; this comparison is not met. Element B_5 satisfies the comparison condition, but the zero in bit 5 of the control vector disables the comparison. Element B_6 satisfies the comparison condition, and the control vector enables the comparison. Thus, the item count of three is transmitted to the rightmost 48 bits of register 0A. The item count includes the B_5 comparison although the control vector disabled this comparison.

†If the C designator is zero, this instruction produces undefined results.

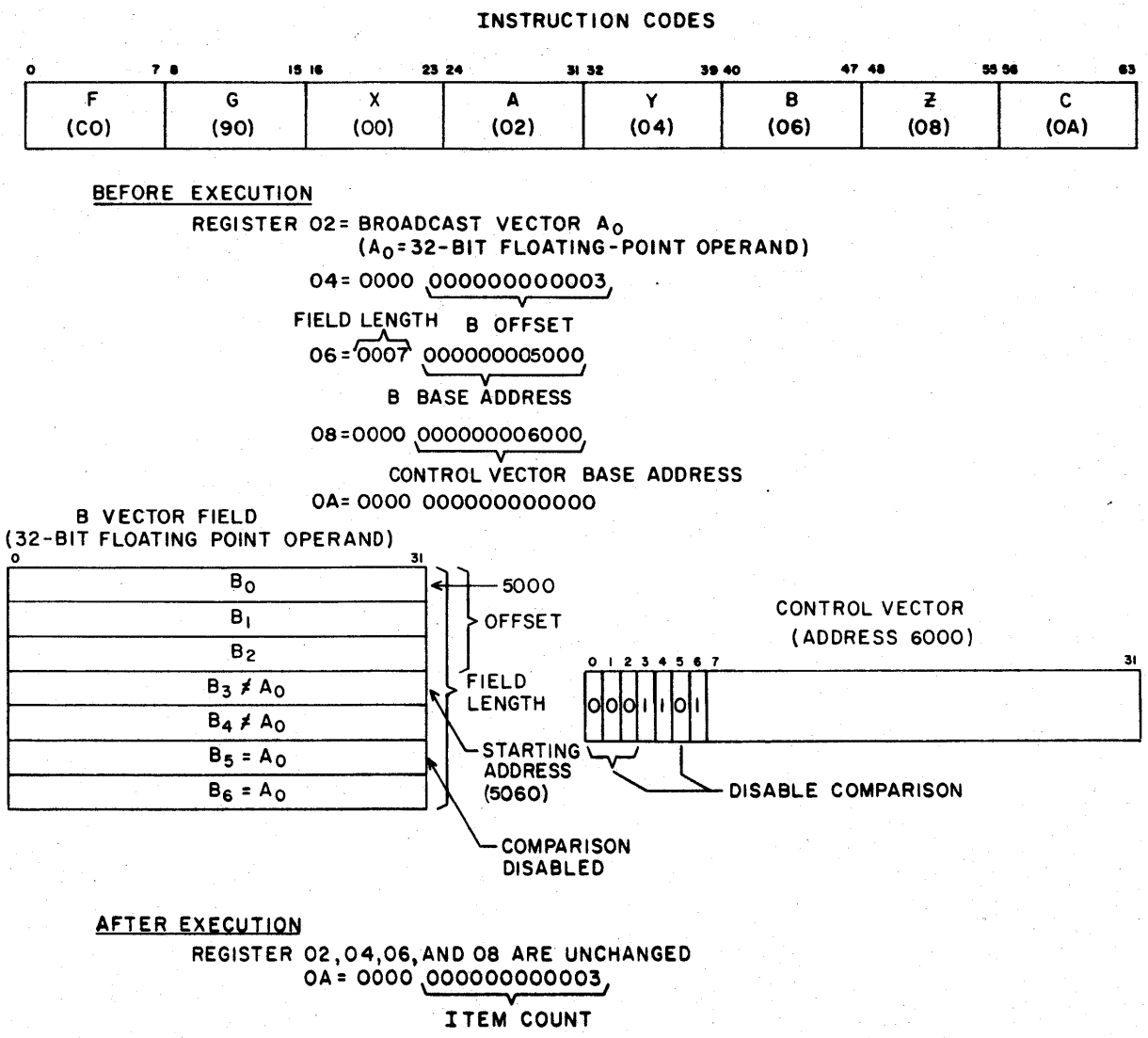
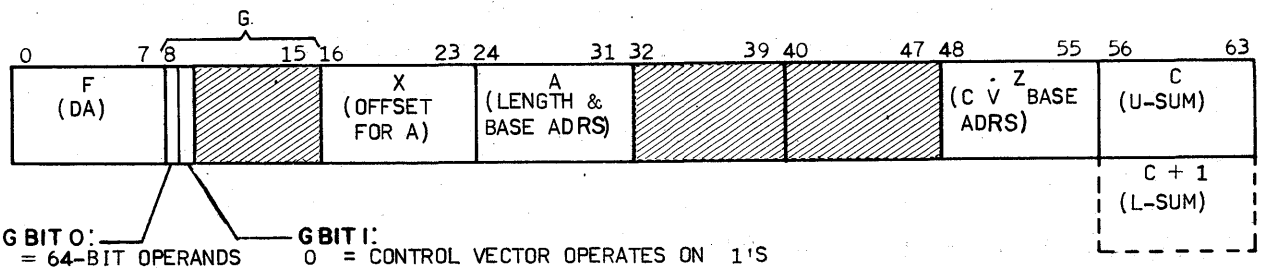


Figure 6-35 Example of Select EQ; A=B, Item Count to C

DA SUM ($A_0 + A_1 + A_2 + \bullet\bullet\bullet\bullet A_n$) TO (C) AND (C+1)



This instruction forms the double-precision, unnormalized, floating-point sum † of all the elements of vector field A. The instruction is executed in the following manner.

$A_0 + A_2 + A_4 + A_6 + \ldots$ = sum X

$A_1 + A_3 + A_5 + A_7 + \ldots$ = sum Y

Where A_0, A_1, A_2, . . . are elements of vector A.

If necessary, the instruction right normalizes the partial sums after each addition. Sums X and Y (both double-precision quantities) are then added to form the final sum. The instruction transmits the upper result portion of the sum to the register specified by C and the lower result to the register designated by C+1.

Registers C and C + 1 are either 32- or 64-bit registers, depending on the state of G bit 0 in the instruction. Register C must be even; if register C is odd or zero, the instruction results are undefined.

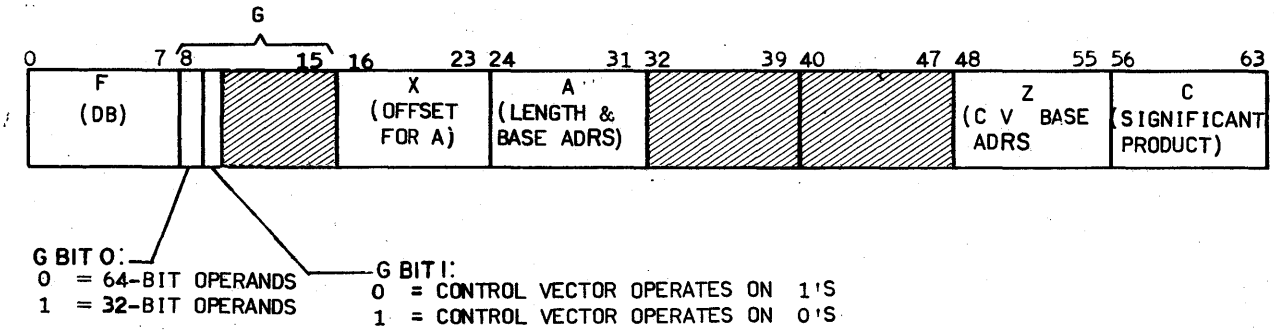
The Y and B designators (bits 32 through 47) and bits 2 through 7 of the G designator are not used and must be zeros. There is no length specification for control vector Z. The instruction terminates when the source vector field A is exhausted. If the control vector allows no elements to be summed, the instruction sets the result to machine zero.

If a control vector (CV) is specified and contains no permissive elements, the result is machine zero. The instruction does not specify control vector length or offset.

† Appendix B describes the double-precision addition of floating-point operands and order-dependent result considerations.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). Data flag bits 43 and 46 are determined only by the final upper and lower results; if the upper result is indefinite, the lower result is undefined. Data flag bit 43 is set if the exponent of the lower result is less than 9000_{16} for 64-bit mode and 90_{16} for 32-bit mode. In this case, the exponent of the upper result may be greater than 9000_{16} and will be stored as is and will not be forced to machine zero. The instruction sets flag bit 42 if any of the add operations overflow.

DB PRODUCT (A_0, A_1, A_2, • • • • An) TO C



This instruction forms the significant product† of successive, floating-point elements in source field A. The instruction is executed in the following manner.

$$A_0 \bullet A_2 = X_1 \qquad A_1 \bullet A_3 = Y_1$$
$$X_1 \bullet A_4 = X_2 \qquad Y_1 \bullet A_5 = Y_2$$
$$X_2 \bullet A_6 = X_3 \qquad Y_2 \bullet A_7 = Y_3$$
$$\vdots \qquad\qquad \vdots$$
$$(X_{n-1}) \bullet A_n = X_n \qquad (Y_{n-1}) \bullet A_n = Y_n$$

Where A_0, A_1, A_2, . . . are elements of source field A, and X and Y are partial products.

Sums X and Y are then multiplied to form the final product. The instruction then stores the final significant product in the register specified by C. Register C is either a 32- or 64-bit register, depending on whether 32- or 64-bit operands are used, respectively.

†Appendix B describes the floating point multiplication operation and order-dependent result considerations.

In the execution of the DB instruction, the following result differences may occur. The central computer may multiply the partial products (X and Y) by a normalized one (EA40 0000 in 32-bit mode or FFD2 4000 0000 0000 in 64-bit mode) an indeterminate number of times, depending on discontinuities in the input data stream. If the coefficients of the partial products are nonzero, the partial products are unchanged by the additional multiply. However, if the coefficient is all zeros, EA or FFD2 is added to the exponent. This is normal under the definition of significant multiply. If the interruptions last long enough, the exponent may decrease to machine zero, setting data flag 43.

Input Stream			Partial Products		
00FF	FFFF		1800	0000	1st
0080	0000				
Interruption occurs here →		(First normalized one)	0200	0000	2nd
			EC00	0000	3rd
			D600	0000	4th
			C000	0000	5th
			AA00	0000	6th
			9400	0000	7th
			8E00	0000	8th

All of the above products are equal under the floating-point compare rules. The last product, however, sets data flag 43 and 46. Data flag 42 sets if any multiply operation overflows.

These discontinuities may be caused by hardware-generated gaps in the input data or by machine interrupts.

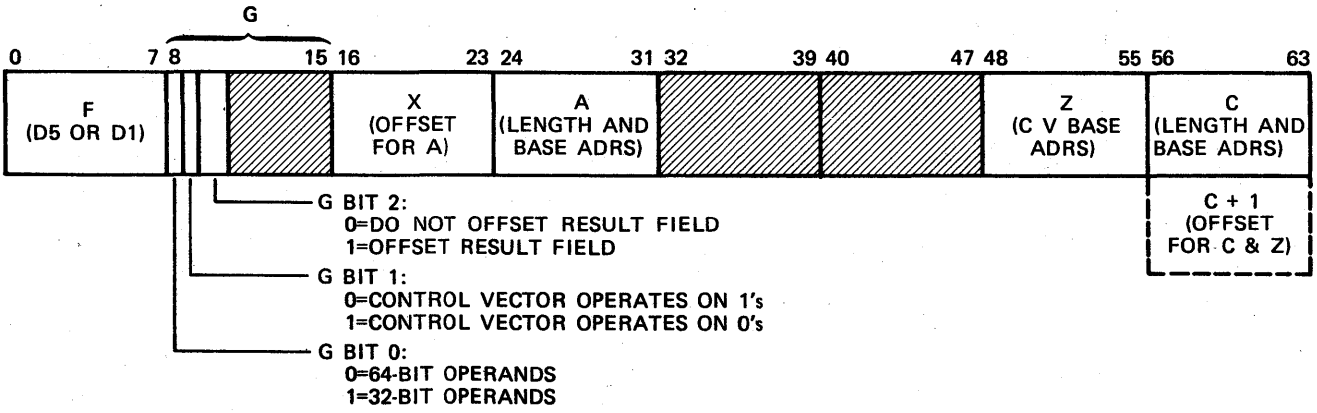
The Y and B designators (bits 32 through 47) and bits 2 through 7 of the G designator are not used and must be zeros. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

If bit 1 of the G designator is a zero, for example, a zero bit in the control vector disables the multiplication of the corresponding source element and the partial product. Thus, the multiplication of a source element and the partial product takes place only when the corresponding control vector bit is enabled. This instruction contains no length specification for the control vector. The instruction terminates when the A source field is exhausted. If the control vector contains no enabling elements, the result is a normalized one.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

D5 DELTA (An + 1 - An)→Cn

D1 ADJ. MEAN (An+1+An) /2→Cn



D5 DELTA $A_{n+1} - A_n \rightarrow C_n$

This instruction forms the nth element of result vector field C by subtracting the nth element of source field A from the n+1th element of A. Normalized, floating-point arithmetic is used in the subtraction. Figure 6-36 shows an example of a delta instruction with assumed instruction codes, operands, and register contents.

The example shows that since there is no offset of the A vector, the first subtraction consists of $A_1 - A_0$ which produces result element C_0. The subtraction of the A vector elements continues in this manner until element C_4 is reached. The corresponding Z control vector bit is a zero which prohibits the storing of the result element C_4 and leaves the C_4 result field location unchanged.

Since the source field is one element shorter than the result field, C_5 becomes minus A_5 and C_6 becomes zero. The delta (D5) instruction terminates when the result field is exhausted.

The Y and B designators and bits 3 through 7 of the G designator are unused and must be zeros.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

INSTRUCTION CODE

0 7 8 15 16 23 24 31 32 39 40 47 48 55 56 63							
F (D5)	G (80)	X (00)	A (02)	Y (00)	B (00)	Z (03)	C (04)

A VECTOR SOURCE FIELD

0 31	ADDRESS
A_0	6000
A_1	6020
A_2	6040
A_3	6060
A_4	6080
A_5	60A0

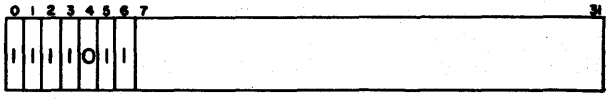
REGISTER CONTENTS

02=0006 000000006000
03=0000 000000007000
04=0007 000000008000

C VECTOR RESULT FIELD

0 31	ADDRESS
$C_0(A_1-A_0)$	8000
$C_1(A_2-A_1)$	8020
$C_2(A_3-A_2)$	8040
$C_3(A_4-A_3)$	8060
C_4 NO CHANGE	8080
$C_5(0-A_5)$	80A0
$C_6(0)$	80C0

Z CONTROL VECTOR



ADDRESS
7000

NOTE: VALUES IN PARENTHESES INDICATE
A VECTOR ELEMENTS SUBTRACTED
FOR CORRESPONDING C VECTOR ELEMENT.

Figure 6-36. Example of Delta Instruction

D1 ADJ. MEAN $\{A_{n+1} + A_n\} /2 \rightarrow C_n$

This instruction forms the nth element of result vector field C by the normalized addition of the nth and n+1th elements of source field A. The instruction then divides the result element by two, producing the mean of the two source elements. The mean result is stored as the corresponding result element in vector C. All operands and arithmetic operations are expressed in floating point.

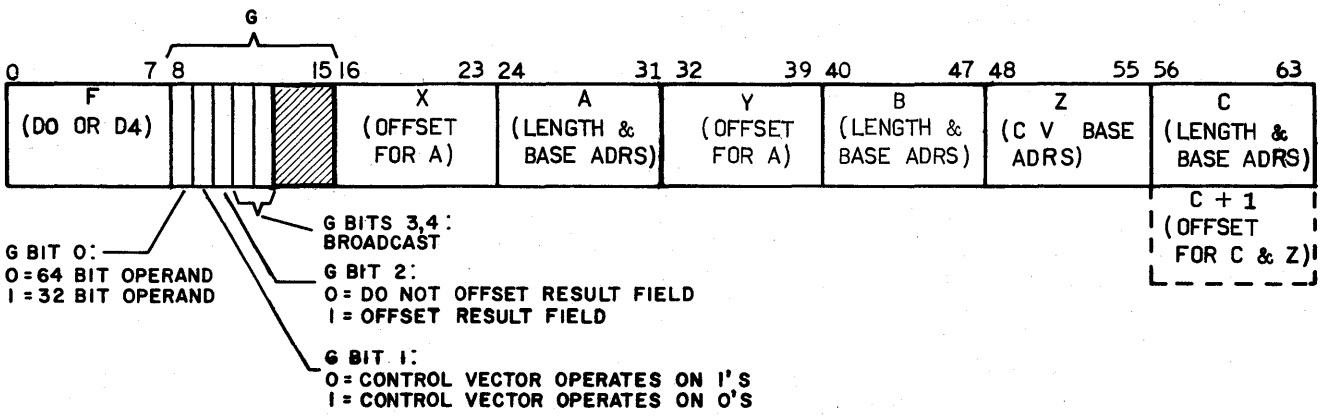
The division by two is accomplished by subtracting one from the exponent of the result element.

The Y and B designators and bits 3 through 7 of the G designator are not used and must be zeros.

Applicable data flag bits are 43 (result machine zero) and 46 (indefinite result).

D0 AVERAGE $\{An+ Bn\} / 2 \rightarrow Cn$

D4 AVE. DIFF. $\{An - Bn\} / 2 \rightarrow Cn$

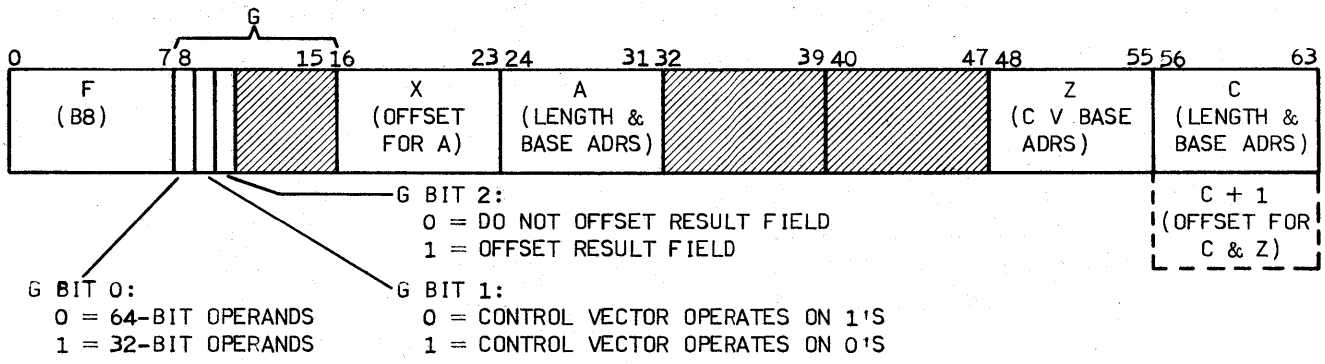


These two instructions form the normalized average and normalized average difference, respectively, of elements A_N and B_N in the A and B vector fields. The sum (D0) or difference (D4) of elements A(N) and B(N) is divided by two. The result elements become corresponding elements of result vector field C. The division by two is accomplished by subtracting one from the exponent.

In all other respects, these instructions function the same as the normal vector instructions described under Vector Instructions in this section. Thus, short source fields are extended with machine zeros. These instructions terminate when the result field is exhausted.

Applicable data flag bits are 43 (result machine zero) and 46 (indefinite result).

B8 TRANSMIT REVERSE A→C



This instruction transmits the elements of vector source field A to vector result field C. The elements are transmitted in reverse order from A to C. Thus, the last element of vector A becomes the first element of vector C, the next to the last element of vector A becomes the second element of vector C, etc.

This instruction terminates when the result field is exhausted. Short source fields are extended with machine zero elements. If the source and result fields overlap in storage, the results of the instruction are undefined.

The Y and B designators and bits 3 through 7 of the G designator are not used and must be zeros. This instruction sets no data flag bits.

Figure 6-37 shows an example of the operation of a transmit reverse A → C instruction with assumed instruction codes, addresses, field lengths, and vector fields.

Since the offsets for the A and C vector fields are equal (+3), the first operation transmits element A_7 to C_3. The operations continue in this manner until bit 5 of the control vector is reached. Since bit 5 = zero, the transmission of A_5 to C_5 is disabled, and C_5 remains unaltered.

The last three elements of vector field C (C_8, C_9, and C_A) are set to machine zero since the result field length is three elements longer than the source length. The dashed lines show the order of transfer of elements from the A vector source field to the C vector result field.

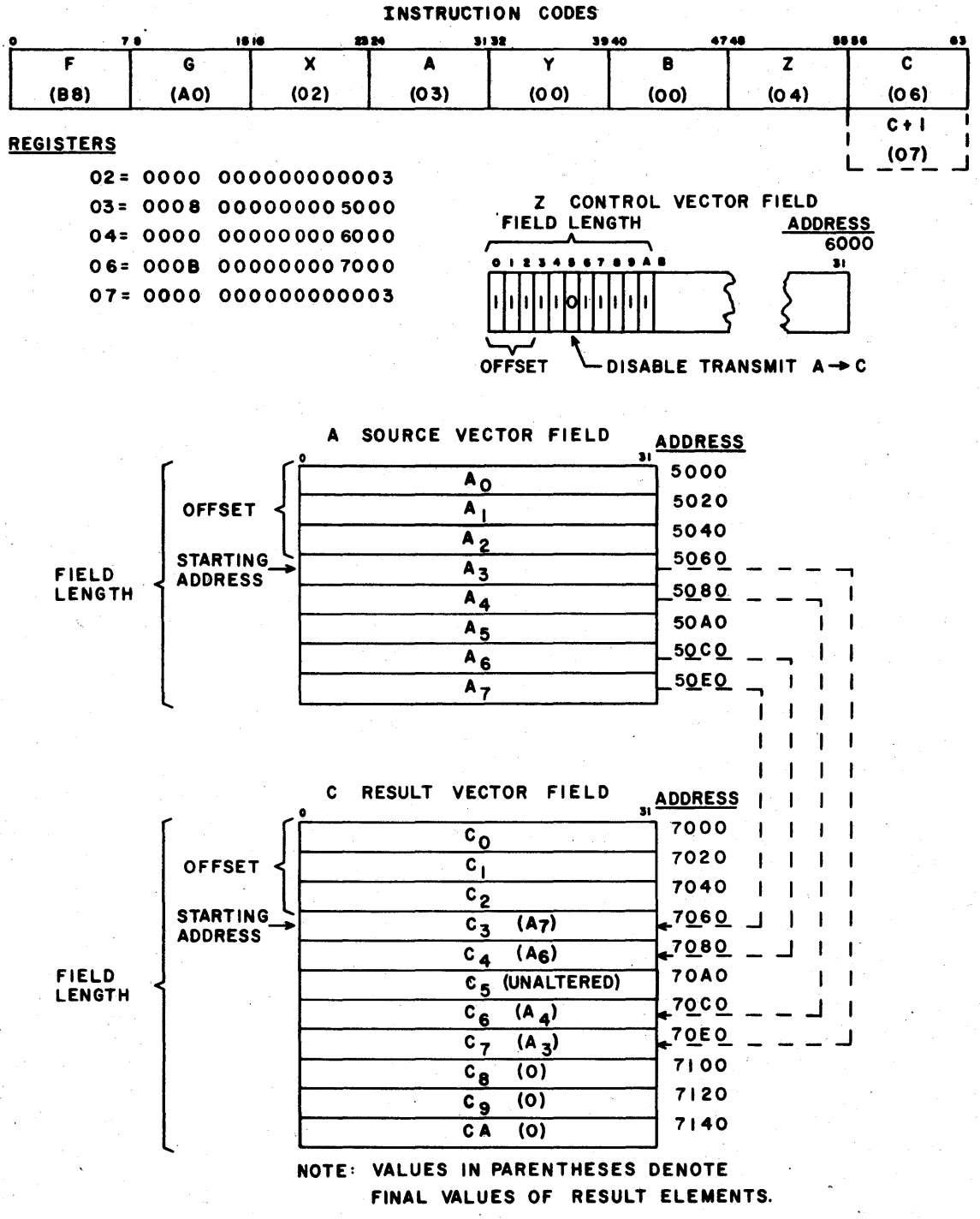
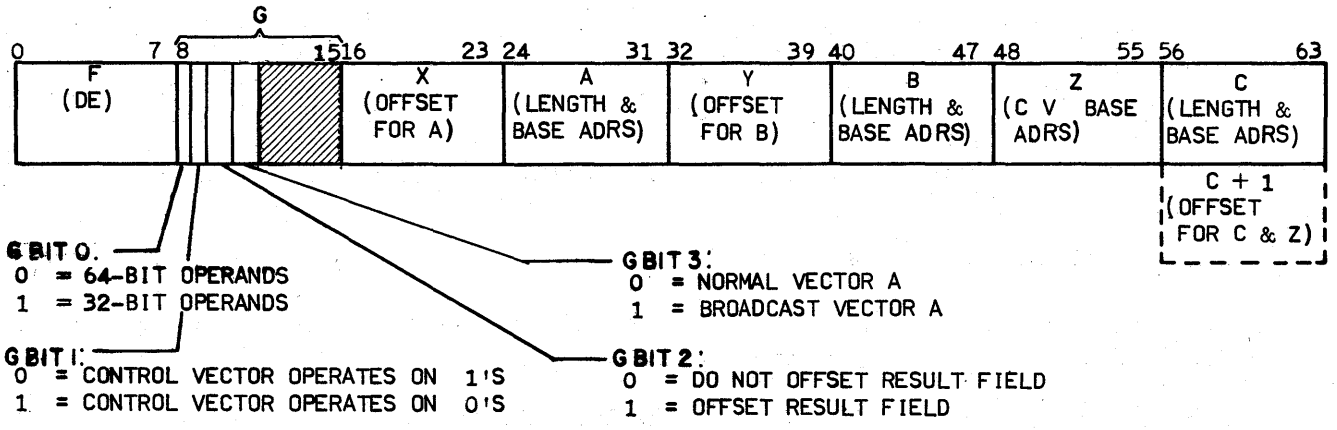


Figure 6-37. Example of Transmit Reverse A →C Instruction

DE POLYNOMIAL EVALUATION



This instruction forms result elements, each of which represents a polynomial evaluation of the repeated product of an element from vector field A and the constants from vector field B. All operands are in floating point format. The elements of source vector A contain the arguments while the elements of source vector B contain the constants necessary for the polynomial evaluation. The instruction forms each result element by evaluating the polynomial at each argument of source vector A. The instruction uses significant upper multiplication and unnormalized addition † (add upper) in performing the evaluation. All arithmetic operations are performed in floating point format. Bits 4 through 7 of the G designator are not used and must be zeros.

This instruction evaluates polynomials of the following general form.

$$Y = K_nX^0 + K_{n-1}X^1 + K_{n-2}X^2 + \ldots + K_2X^{n-2} + K_1X^{n-1} + K_0X^n$$

The grouping of terms produces the expression of the same polynomial of the following form.

$$Y = K_n + X\left\{ K_{n-1} + X\left[K_{n-2} + \ldots X(K_2 + X\left[K_1 + K_0X\right])\right]\right\}$$

The DE instruction evaluates the polynomials expressed in the previous general form.

† Appendix B describes the significant upper and unnormalized floating-point operations.

The substitution of vector-element terms in the preceding polynomial expression yields the following.

$$C_0 = B_n + A_0 \left\{ B_{n-1} + A_0 \left[B_{n-2} + \cdots A_0 \left(B_2 + A_0 \left[B_1 + B_0 A_0 \right] \right) \right] \right\}$$

$$\vdots$$

$$C_m = B_n + A_m \left\{ B_{n-1} + A_m \left[B_{n-2} + \cdots A_m \left(B_2 + A_m \left[B_1 + B_0 A_m \right] \right) \right] \right\}$$

In the previous polynomial expressions:

B_0 represents the first element of vector field B (the highest order constant in the polynomial B_x) and B_n denotes the lowest order element.

A_0 represents the first element and Am the last element of vector field A.

C_0 denotes the first element and Cm the last element of result vector field C.

The DE instruction forms each element of result vector field C (polynomial evaluation) by performing the series of floating point multiplications and additions indicated in the preceding polynomial expressions. Figure 6-38 illustrates the basic sequence of arithmetic operations in the execution of the polynomial evaluation DE, instruction.

Figure 6-38 shows that the first pass multiplies each element of field A by the first element of field B. The computer stores the result from the first pass and all successive passes in field C. The second pass adds each element of field C to the second element of field B and stores the result in field C. The third pass multiplies each element of field C by its respective element of field A and stores the results in field C. The rest of the passes are like the second and third: add, multiply, add, multiply, etc. Each add pass decrements field B to the next lower order field B operand. The instruction terminates when field B is exhausted.

Short A vector source fields are extended with normalized ones. If in Figure 6-38, for example, the A vector source field was only two elements in length, C_2 would equal the sum of all the B vector elements ($B_0 + B_1 + B_2 + B_3 + B_4$) since all of the A vector source elements in the evaluation would equal one. As indicated by the instruction format, the A vector can be a single broadcast element.

The B vector cannot be broadcast. In regard to control vectors and offsets, the DE instruction functions are the same as normal vector instructions.

If the B vector length (length minus offset) equals zero before the reading of the first operand, the instruction operates as a no-operation (no-op). If the B field length equals one, two, or three, the results are as follows:

B field Length	Result
One	Undefined
Two	$B_1 + AB_0$
Three	$B_2 + A\ (B_1 + AB_0)$

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). The setting of data flag bits 43 and 46 is determined only by each result stored into field C and not by any partial result in forming that element. Data flag bit 42 (exponent overflow) is set if overflow occurs in any add or multiply operation.

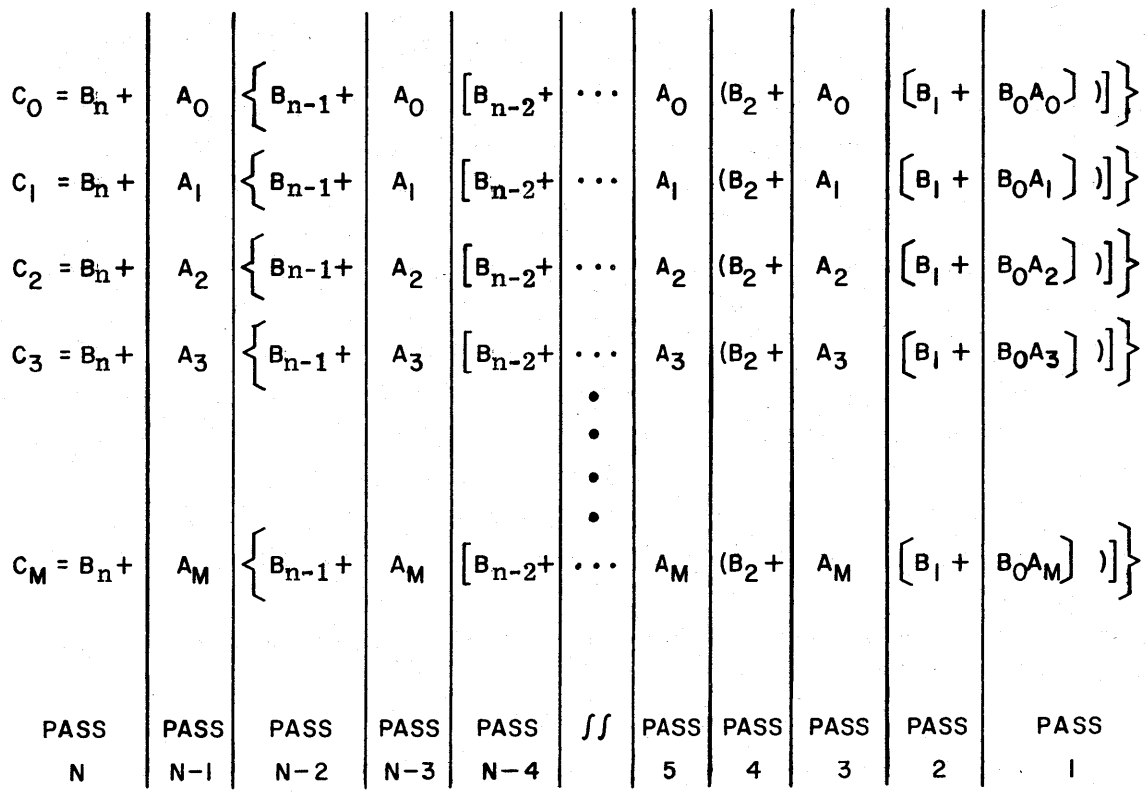
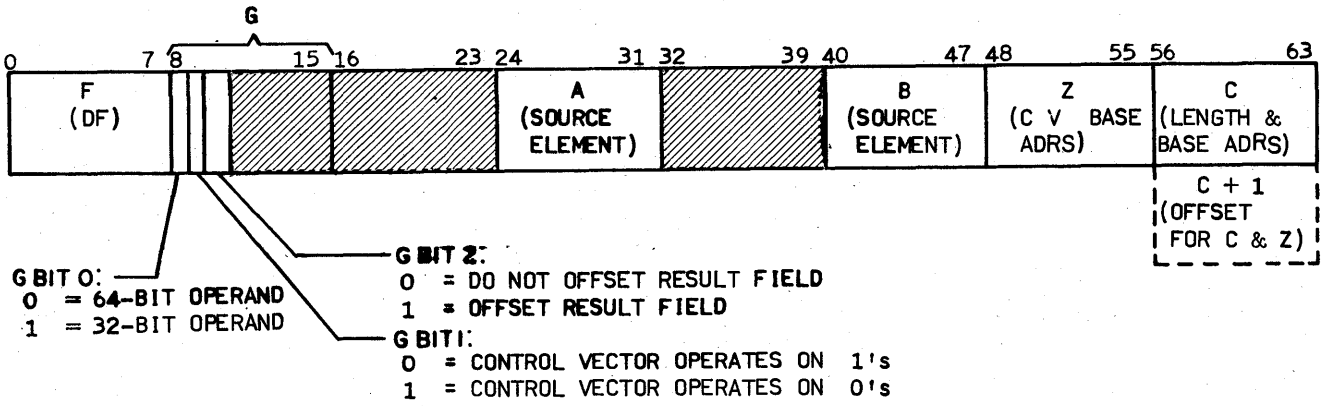


Figure 6-38. Basic Arithmetic Sequence for Polynomial Evaluation Instruction

DF INTERVAL A PER B➔C



This instruction forms a result vector field D. The initial element of vector field D is the constant from the register designated by B. The instruction forms each succeeding result element by adding the constant in register B to the preceding element. Thus, the second element of vector D equals the first element plus the content of register B. The third element of vector D equals the second element plus the content of register B, etc. The instruction uses unnormalized, floating point addition.† Thus, the first element of D_0 = B and the succeeding elements are $D_n = D_{n-1} + B$.

If the instruction uses a control vector, an element is generated for each control bit of the field length, although it may not be stored in the result field. If the instruction detects a nonpermissive bit in the control vector, the addition operation is performed, but the result element is not stored in the result field. If the control vector disables the storing of a result element and this element is indefinite, data flag bit 46 (indefinite result) is not set until a permissive bit is detected in the control vector. Similarly, data flag bit 42 (exponent overflow) or 43 (result machine zero) is set on the next permitted store although the iterative step which overflowed was not stored.

The X and Y designators and bits 3 through 7 of the G designator are not used and must be zeros.

†Appendix B describes floating point arithmetic and order-dependent result considerations.

The central computer executes the DF instruction (table 6-20) with the pipelines performing an add operation.

TABLE 6-20. DF INTERVAL A PER B → C INSTRUCTION

NOTE

A is A operand, B is B operand, MZ is machine zero, SSA is short stop A, and SSB is short stop B.

	Pipeline 1			Pipeline 2		
Cycle	A Input	B Input	Output	B Input	A Input	Output
1	B	B	X	X	X	X
2	B	B	X	X	X	X
3	B	B	X	B	B	X
4	B	B	X	MZ	MZ	X
5	2B ←SSA—	2B ←SSB—	2B	B	MZ	X
6	2B	2B	2B	B	B	X
7	2B	2B	2B	B	2B ←SSA—	2B
8	2B	2B	2B	A	MZ	MZ
9	MZ	4B ←SSB—	4B	A	B	B
10	MZ	4B	4B	A	2B	2B
11	MZ	4B	4B	A	3B	3B
12	MZ	4B	4B→	4B	A	A
13	MZ	4B	4B	4B	A+B	A+B
14	MZ	4B	4B	4B	A+2B	A+2B
15	MZ	4B	4B	4B	A+3B	A+3B
16	MZ	4B	4B	4B	A+4B	A+4B
17	MZ	4B	4B	4B	A+5B	(A+B)+4B
18	MZ	4B	4B	4B	A+6B	(A+2B)+4B
↓	↓	↓	↓	↓	↓	Results to Stream

The results to stream may be modified slightly if an interrupt occurs. For example, if an interrupt occurs at cycle 12, the instruction progresses as shown in table 6-21.

TABLE 6-21. DF INTERVAL INSTRUCTION WITH INTERRUPT

NOTE

A is A operand, B is B operand, MZ is machine zero, SSA is short stop A, and SSB is short stop B.

	Pipeline 1			Pipeline 2		
Cycle	A Input	B Input	Output	B Input	A Input	Output
11	MZ	4B	4B	4B	3B	3B
12	MZ	4B	4B	4B	A	A
						A+B
13	Interrupt			Held in Stream during Interrupt		
14	B	B	X	X	X	X
15	B	B	X	X	X	X
16	B	B	X	B	B	X
17	B	B	X	MZ	MZ	X
18	2B ← SSA	2B ← SSB	2B	B	MZ	X
19	2B	2B	2B	B	B	X
20	2B	2B	2B	B	2B ← SSA	2B
21	2B	2B	2B	A+B	MZ	MZ
22	MZ	4B ← SSB	4B	A+B	B	B
23	MZ	4B	4B	A+B	2B	2B
24	MZ	4B	4B	A+B	3B	3B
25	MZ	4B	4B →	4B	A+B	A+B
26	MZ	4B	4B	4B	A+B+B	(A+B)+B
27	MZ	4B	4B	4B	A+B+2B	(A+B)+2B
	↓	↓	↓	↓	↓	Results to Stream

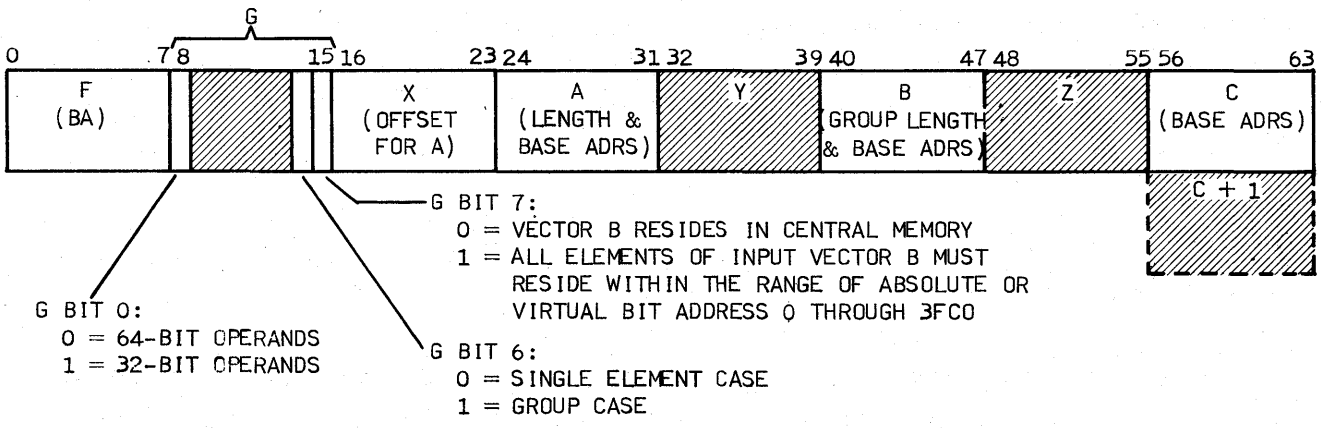
After an interrupt, the instruction is restarted in a manner similar to its initial startup but with the next result after the interrupt used in place of A. In table 6-21, A+B is the first valid result after the interrupt followed by A+B+B etc. (A+B)+B would correspond to the result A+2B in the case where no interrupt occurred. Since add is order dependent, these two quantities may not be equal.

Example:

A = 01	000001
B = 00	000001
A+B = 01	000001
+B = 00	000001
(A+B)+B = 01	000001
B = 00	000001
+B = 00	000001
2B = 00	000002
A = 01	000001
+2B = 00	000002
A+2B = 01	000002

Not Equal

BA TRANSMIT INDEXED LIST➔C



This instruction forms an indexed list of result elements in vector field C by transferring elements from addresses in vector field B as indexed by the item counts in the A-vector field. The rightmost 48 bits (no half-word option) of each element of vector A contains an item count. The instruction adds the first item count in vector A to the base address of the first element of vector B. The element at the new address is

transferred to result vector C. Before the addition of the item count (index) to the base address, the index is left-shifted five places (32-bit operands) or six places (64-bit operands) to form the half-word or full-word address, respectively.

The instruction then adds the next element of vector A to the base address of the first element of vector B. The resulting address indexes the second element of vector B. This process continues until vector A is exhausted.

The elements of vector A are always 64-bit operands, while G bit 0 specifies the B and C vector element size.

Bits 1 through 5 of G designator are not used and must be set to zero.

If G bit 6 is a zero, the instruction transmits single elements as previously described. If G bit 6 is set, a group of elements is transmitted from vector B to vector C for each element of vector A. The group length is specified in the upper 16 bits of register B. All groups are of equal length.

If G bit 7 is set, all elements of input vector B must reside in the register file within the range of absolute or virtual bit addresses 0 through 3FCO. Reference to the register file as central memory is normally not allowed. This instruction and the B7 instruction are the only instructions which permit this type of reference to occur. Refer to section 5 for other register file restrictions. If all the addresses for vector B are not contained in the register file, this instruction is undefined. This instruction is also undefined if G bits 6 and 7 are both set.

The search: index list→C (C8 through CB) instructions may be used to produce the index list for the BA instruction.

Figure 6-39 shows an example of a transmit indexed list→C instruction with assumed instruction codes, register content, and vector fields. The first item count is read from address 4000. This value indexes the B vector base address by five half-words after the left shift of five. Thus, the instruction transfers the first B vector element from address 70A0 to the C vector element address 9000. Six B vector elements are transferred to the C vector.

No data flag bits are set by the BA instruction.

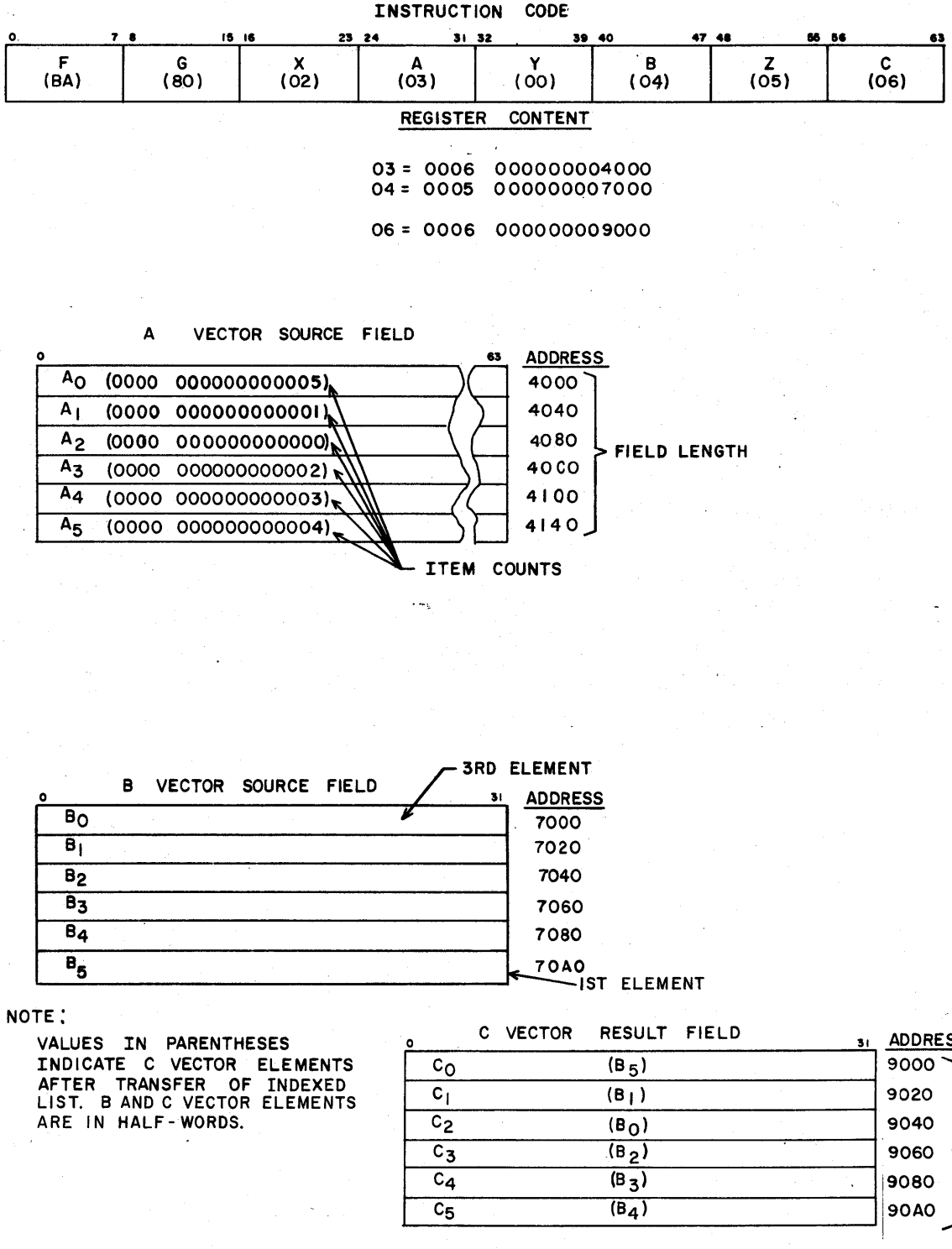
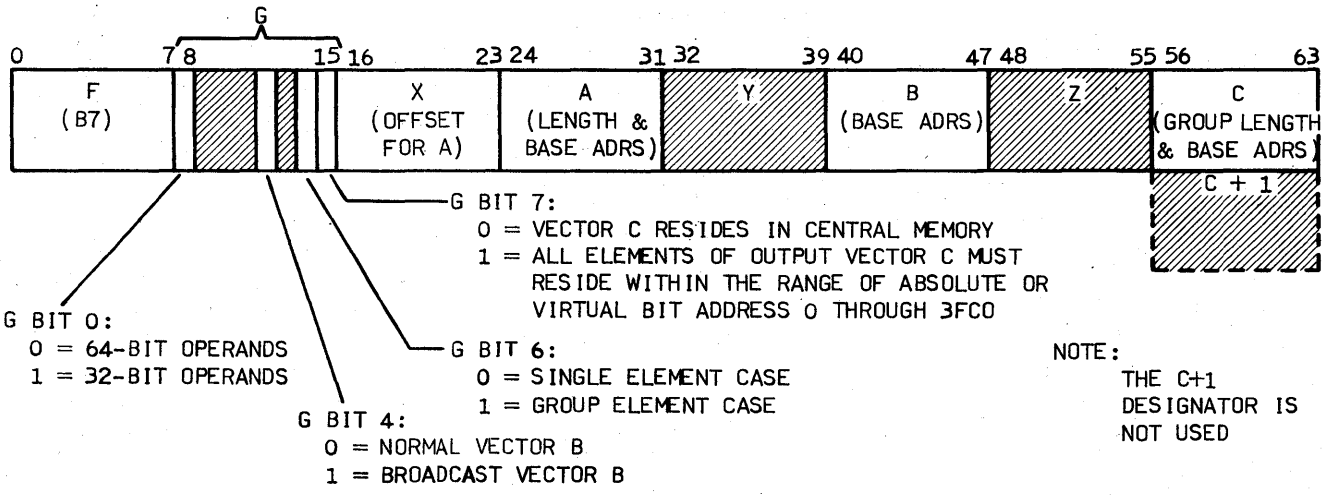


Figure 6-39. Example of Transmit Indexed List →C Instruction

B7 TRANSMIT LIST ➔ INDEXED C



This instruction adds the rightmost 48 bits of the first element of vector field A to the base address in register C to form the address of the first element of result vector field C. The instruction then transmits the first element of vector field B to the computed address in C. The rightmost 48 bits of each element of vector field A is an item count. Before the addition of the item count (index) to the base address, the index is left-shifted five places (32-bit operands) or six places (64-bit operands) to form the half-word or full-word address, respectively.

Similarly, the instruction forms the address of the second element of vector field C by adding the second element of vector field A to the base address in register C. The second element of vector field B is then transmitted to the computed address in the result vector field C. The instruction continues in this manner until the A vector field length is exhausted.

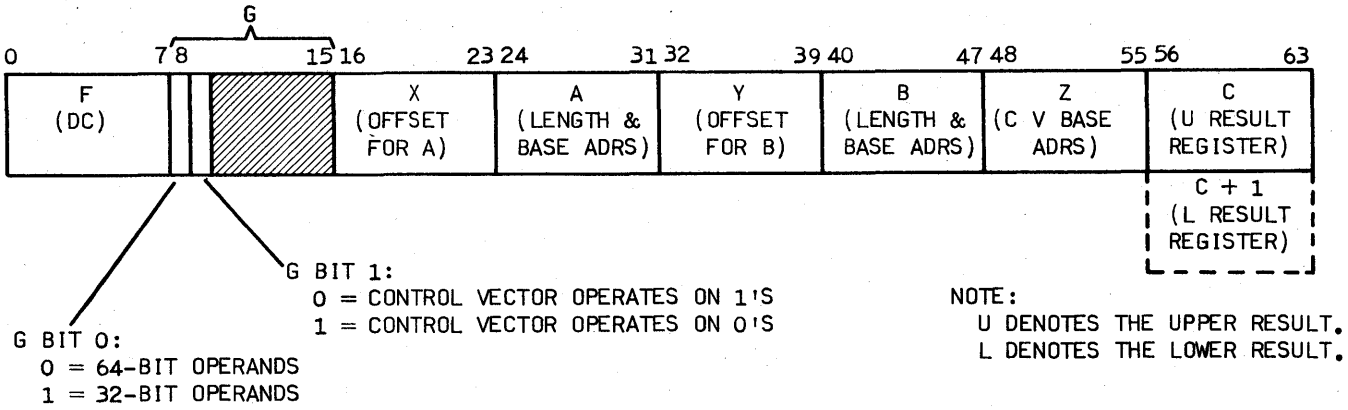
The Y, Z, and C+1 designators are not used and must be zeros. The elements of vector field A are 64 bits while the elements of vectors B and C are 64 bits or 32 bits as specified by G designator bit 0.

Bits 1, 2, 3, and 5 of the G designator are not used and must be set to zero. Vector B is broadcast when bit 4 of the G designator is set and bit 6 is a zero.

If G bit 6 is a zero, the instruction transmits single elements as previously described. If G bit 6 is set, a group of elements is transmitted from vector B to vector C for each element of vector A. The group length is specified in the upper 16 bits of register C. All groups are of equal length.

If G bit 7 is set, all elements of output vector C must reside in the register file, within the range of absolute or virtual bit addresses 0 through 3FCO. Reference to the register file as central memory is normally not allowed. This instruction and the BA instruction are the only instructions which permit this type of reference. Refer to section 5 for other register file restrictions. If all the addresses for vector C are not contained in the register file, the instruction is undefined. This instruction is also undefined if either G bits 4 and 6 or G bits 6 and 7 are set.

DC VECTOR DOT PRODUCT TO (C) AND (C+1)



This instruction multiplies corresponding elements of vector fields A and B and forms the sum of the products. This instruction uses double-precision, unnormalized, floating-point† arithmetic in the operation. The sum of the double-precision products is of the following form.

$$(A_0 \bullet B_0) + (A_2 \bullet B_2) + \ldots + (An \bullet Bn) = X$$
$$(A_1 \bullet B_1) + (A_3 \bullet B_3) + \ldots + (An \bullet Bn) = Y$$

where: A_n are elements of vector A, B_n are elements of vector B, and X and Y are partial sums of the product.

Sum X and sum Y (both double precision quantities) are then added to form the final sum. The instruction transmits the upper result portion of the sum to the register specified by C and the lower result to the register designated by C+1. The DC instruction terminates when the shorter of the two source fields is exhausted. If the control vector contains no enabling elements, the result is set to machine zero.

Bits 2 through 7 of the G designator are not used and must be zeros. The instruction contains no length designator for the control vector Z.

C must specify an even-numbered register. If C specifies an odd-numbered register, the instruction results are undefined.

† Appendix B describes floating-point arithmetic and order-dependent result considerations.

The DC instruction probes the setting of data flag bits 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). Data flag bits 43 and 46 are determined only by the final upper and lower results; if the upper result is indefinite, the lower result is undefined. Data flag bit 43 is set if the exponent of the lower result is less than 9000_{16}. In this case, the exponent of the upper result may be greater than 9000_{16} and will be stored as is and will not be forced to machine zero. The instruction sets data flag bit 42 if any of the multiply operations overflow.

STRING INSTRUCTIONS

The string instructions typically perform arithmetic and logical operations on strings of data in the form of 8-bit bytes. The 8-bit byte size allows handling large alphabets; this size is also compatible with ASCII and EBCDIC codes. The data strings are in the general format shown in figure 6-40.

The field length of the data string can extend beyond one 64-bit word. The field length of the data string can also be less than one data word.

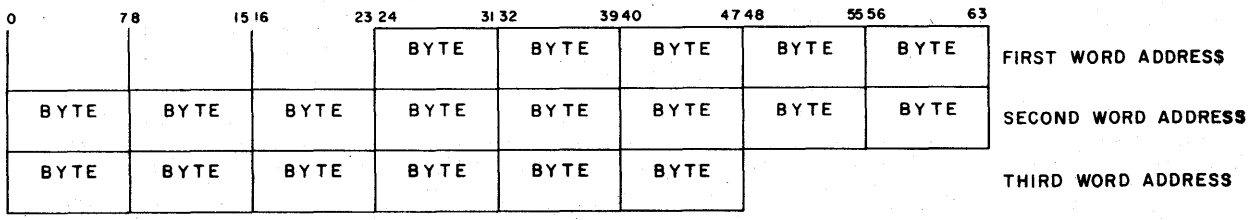


Figure 6-40. Example of General Format of a Data String Field

The order of processing the bytes in the string instructions may be from right to left or left to right as described in the instruction descriptions in this section. For string instruction terminations, refer to the descriptions of the individual instructions.

STRING INSTRUCTION DATA CODE AND FORMATS

String instructions perform operations on data strings using decimal data codes in packed binary coded decimal (BCD), zoned BCD, and binary formats. The following paragraphs describe these codes and formats.

DECIMAL DATA CODES

The string instructions represent decimal numbers as signed magnitudes. Four bits represent the sign. Each group of four succeeding bits represents a decimal digit. Table 6-22 lists each decimal digit and sign representation and the corresponding binary code.

TABLE 6-22. DECIMAL DATA CODES

Decimal Digit	Binary Code	Sign	Binary Code
0	0000	+	1010
1	0001	-	1011
2	0010	+	1100
3	0011	-	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

Although several binary codes represent the decimal sign in string instruction operations, the four plus sign codes are equal. Similarly, the two minus signs equal each other.

During the job mode, the sign and zone bits (table 6-23)) are generated by the decimal string instructions and are conditioned by the ASCII/EBCDIC bit in the job invisible package. During monitor mode, only ASCII codes are generated. The move and scale A → C (FA) instruction, which transmits the sign bits directly, represents the only exception to this principle.

TABLE 6-23. RESULT SIGNS

Sign	ASCII Selected	EBCDIC Selected
+	1010	1100
-	1011	1101
Zone	0011	1111

PACKED BCD

The string instructions perform decimal arithmetic on data in the packed format in figure 6-41..

Figure 6-41 shows that the rightmost four bits of the rightmost byte in the field contain the sign. The leftmost four bits of the rightmost byte contain the least significant digit of the number. All other bytes in the field contain two 4-bit digits.

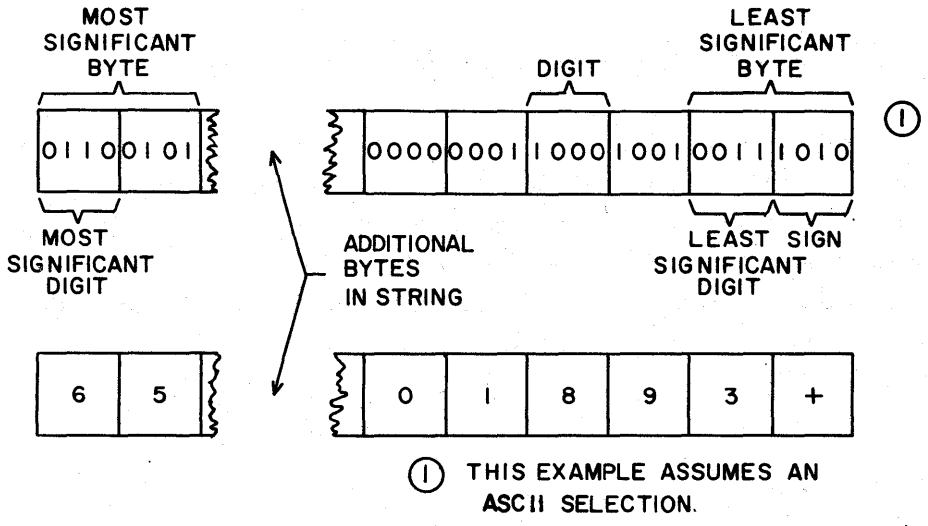


Figure 6-41. Example of the Packed Decimal Format

ZONED BCD

The zoned BCD is used mainly in input/output operations. In the zoned BCD format, each byte contains one BCD digit (figure 6-42).

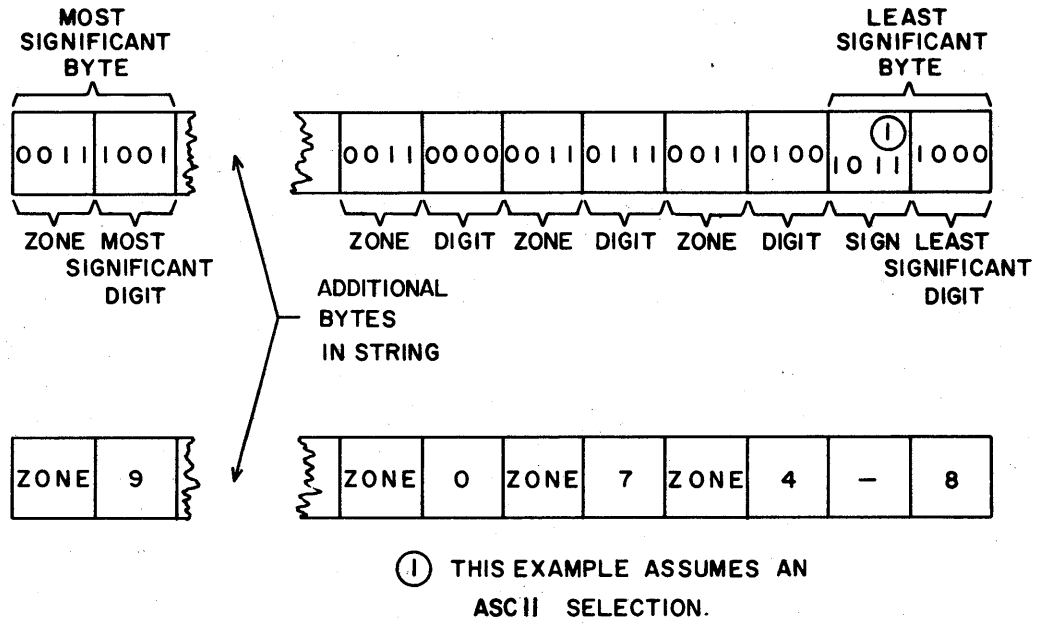


Figure 6-42. Example of the Zoned BCD Format

Figure 6-42 shows that the left four bits of the rightmost byte in the field contain the sign. The leftmost four bits of all other bytes contain the zone designator for the corresponding digit. Since an ASCII selection is assumed in the example, a zone code of 0011 corresponds to a decimal digit. Refer to the ASCII conversion table in appendix A. Some string instructions pack zoned numbers into the packed decimal format and unpack packed decimal numbers into the zoned format.

BINARY FORMAT

String instructions represent binary numbers as strings of 8-bit bytes. The least significant bit is the rightmost bit of the rightmost byte. The leftmost bit of the leftmost byte contains the sign bit. Positive numbers have a zero sign bit. Negative numbers are expressed in two's complement form and have a one for the sign bit. All binary numbers in string instructions must have the sign extended through the sign bit. The length of the binary numbers is dependent upon the specified field length as described in the following paragraphs.

STRING INSTRUCTION FORMAT

The string instructions use the general format shown in Figure 6-43.

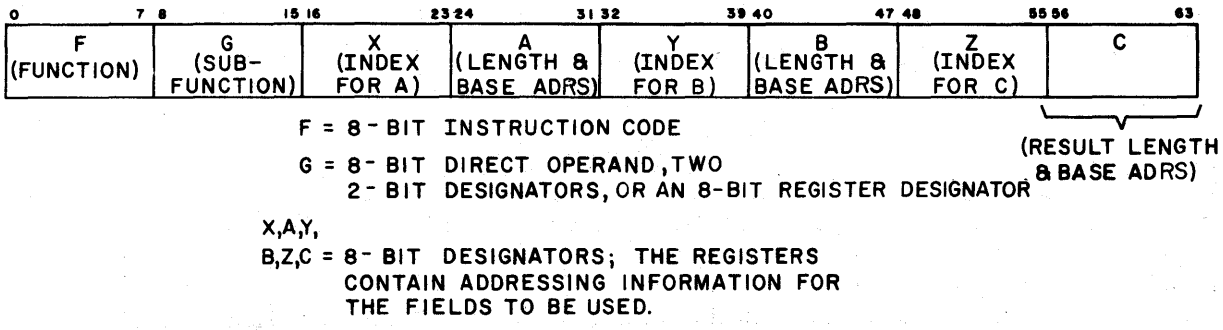


Figure 6-43. General String Instruction Format

FIELD LENGTHS, BASE ADDRESSES, AND INDEXES

Figure 6-44 shows the format of the registers containing the field length, base address, and index for a given data string.

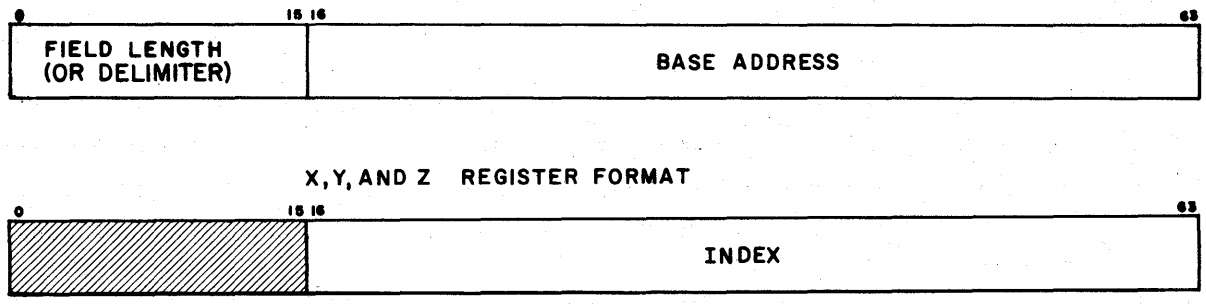


Figure 6-44. String Instruction Register Formats

If any of the 8-bit designators X, Y, or Z are 00_{16}, the instruction does not use indexing for that string but obtains the address of the initial byte from the base address. Figure 6-45 shows an example of the addition of the index to an initial address to obtain the initial byte and field length for the data string.

Figure 6-45 shows that the effective length of the data field is the same as the field length contained in the specified register. Indexing does not affect the effective field length as does offsetting in the vector instructions.

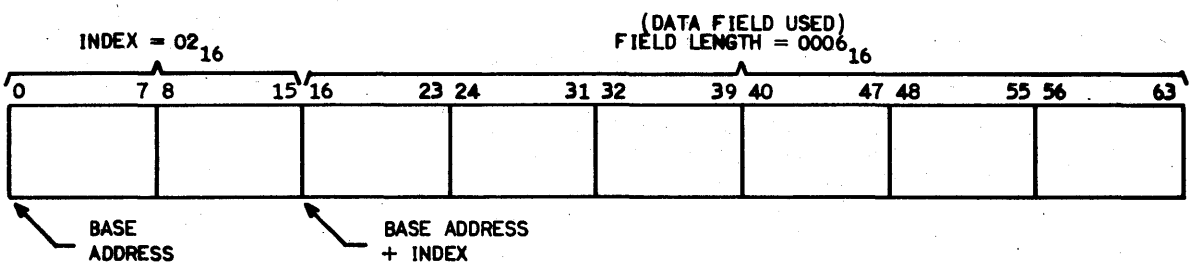


Figure 6-45. Example of Index and Field Length Applied to a Data Field

If the specified length of a string source field is zero, that field is identical to a normal field containing positive zero. If the specified length of the result field is zero, the instruction functions as a no-op.

STRING INDEXES

In all string instructions, indexes are item counts in bytes except for the search for masked key bit (D6) and masked key word (FF) instructions. String indexes differ from vector offsets in that the range of vector offsets is limited to $\pm 2^{16}-1$ while string indexes have any value up to $\pm 2^{45}-1$ for byte item counts. Since byte indexes are left-shifted three places before they are added to the base address, the leftmost three bits of a string index are not used. The sign bit of negative indexes must be extended through bit 16 (figure 6-44). Overflows are ignored when indexes are added to base addresses.

DELIMITERS

The following six instructions can use delimiter termination.

1. Move bytes left; A → C (F8)
2. Move byte left, one's complement (F9).
3. Compare bytes A, B per mask field C (FD).
4. Translate A per B → C (EE).
5. Translate and test A per B→C (EF).
6. Translate and mark (D7).

All other string instructions contain fields that are limited by the specified field length.

Delimiters are contained in the field length specification (bits 0 through 15) of the designated register as shown in figure 6-44. When a delimiter character is used, the field terminates when a character matching the delimiter is reached in the data field. Figure 6-46 shows an example of a delimiter used in a data field. The subfunction (G designator bits) controls the selection of field length or delimiter character as follows:

d (G bits 0 and 1) = designator for fields A and B

e (G bits 2 and 3) = designator for field C
(G bits 4 and 6) = undefined, must be zeros except for instructions D7 and FD
(G bits 5 and 7) = when used, these bits control the incrementing of the A and C field indexes, respectively

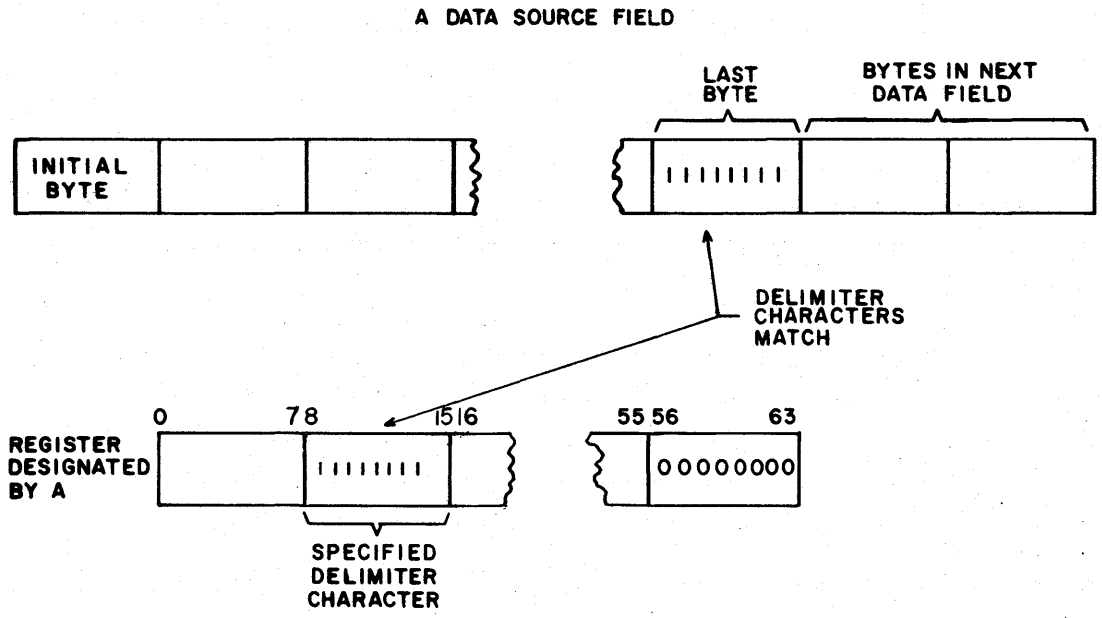


Figure 6-46. Example of Delimiter Termination of a Data Field

Table 6-24 lists the bit values for the G bit d and/or e designators and the corresponding functions.

TABLE 6-24. G DESIGNATORS FOR STRING INSTRUCTIONS †

Designator	d/e Bit Value	Function
d and/or e	00	The 16-bit length specification in A, B, and/or C represents an item count of the number of items in the field (field length). This item count has the range of $+2^{16}$ -1.
d and/or e	10	The rightmost eight bits of the length specification in A, B, and/or C are used as a delimiter character.
d and/or e	11	The entire 16 bits of the length specification in A, B, and/or C are used as a delimiter character.
d	01	The rightmost eight bits of the length specification function as a delimiter character. The leftmost eight bits serve as a mask on the comparison. Bits in the delimiter character and the operand byte are compared only where ones exist in the mask. This specification applies only to source fields. Any instruction becomes undefined if this specification is used for a result field.

If a delimiter is specified for a source field, the instruction does not use the delimiter character as an operand. In the case of a 16-bit delimiter, the field terminates when the leftmost eight bits and the rightmost eight bits of the 16-bit delimiter character match two consecutive source bytes.

If an 8-bit or 16-bit delimiter is specified for the result field, the instruction stores the delimiter character at the end of the result field. The delimiter does not specify a field length in this case since the instruction does not search the result field for the delimiter. If a 16-bit delimiter is used, the instruction stores the leftmost eight bits and rightmost eight bits in consecutive order at the end of the result field.

†Appendix C provides a comprehensive listing of the G designator bits usage according to function code.

In the translate A per B → C (EE) instruction, the use of a delimiter character for the result field causes the instruction to terminate when the A field is exhausted.

INDEX INCREMENTS

The following instructions contain index incrementing capabilities.

1. Move bytes left; A → C (F8).
2. Move bytes left, one's complement (F9).
3. Compare bytes A, B per mask field C (FD).
4. Search for masked key, byte A, B per C (FE).
5. Search for masked key, word A, B per C (FF).
6. Search for masked key bit A, B per C (D6).
7. Translate A per B → C (EE).
8. Translate and test A per B → C (EF).
9. Translate and mark A per B → C (D7).

At the termination of these instructions, the index registers associated with the fields will be in no increment, partial increment, or full increment, as described in the following paragraphs.

NO INCREMENT

In this state, the index register remains at the initial value. Index registers associated with a translate table provide an example of this state. In this case, the instruction adds the characters to be translated to the indexed address of the table to obtain the translated character. The index associated with the table does not change during the instruction execution.

PARTIAL INCREMENT

In this case, the index register is incremented to specify a particular character or word in its associated field. The compare bytes A, B per mask field C (FD) instruction, which searches two byte strings for inequality, provides an example of this type of indexing. When the instruction finds an inequality, the search terminates and the number of no-hit byte compares is added to each index; the fields may not have reached the end of their specified lengths. However, the storage location of the characters that were unequal can be found by manipulating the incremented index register and the base address.

FULL INCREMENT

In this case, the index register is incremented by one for each byte from the corresponding field. When the translate A per B to C instruction terminates, for example, the index associated with source field A is incremented throughout the length of field A. Thus, this index indicates the starting point of the next consecutive field. If a delimiter character specifies a field length, the instruction searches the field for the delimiter character. The instruction then increments the index of the associated field so that the starting point of the next field is one byte beyond the delimiter character.

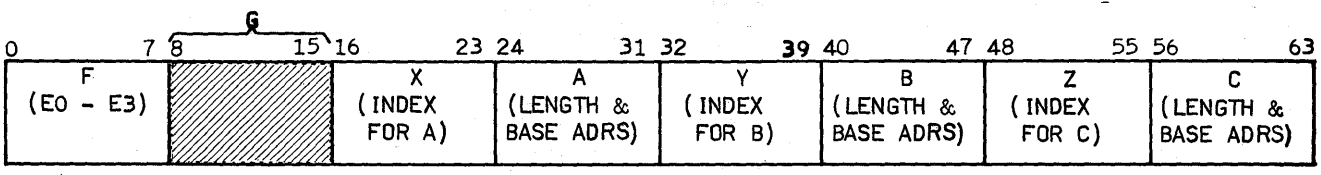
Where appropriate, each instruction description contains a table providing information concerning indexing. Each of these tables specifies the state of the index for each field following the termination of the instruction.

E0 BINARY ADD; A + B ➔C

E1 BINARY SUB; A — B ➔C

E2 BINARY MPY; A • B➔C

E3 BINARY DIV; A / B ➔C



These instructions use the instruction format shown; the G designator is not used and must be all zeros. All indexing is in bytes.

If the length of the destination field C is too short to correctly contain the result of the operation, overflow occurs. This causes data flag 39 (string arithmetic overflow) to be set and the contents of output field C is undefined.

E0 BINARY ADD; A + B → C and E1 BINARY SUB; A - B → C

These instructions add/subtract binary field B to/from binary field A. The instructions use two's complement arithmetic in the operation. If the source field lengths are unequal, the instruction automatically extends the sign bit of the shorter field.

These instructions produce a result binary field C with the sign bit extended, if necessary, to fill out the specified field length.

Figure 6-47 shows an example of a binary add; A + B → C (E0) operation with assumed instruction codes, register contents, and source fields. The sign bit of the A source field is extended in the addition operation. The addition operation is a conventional, two's complement add.

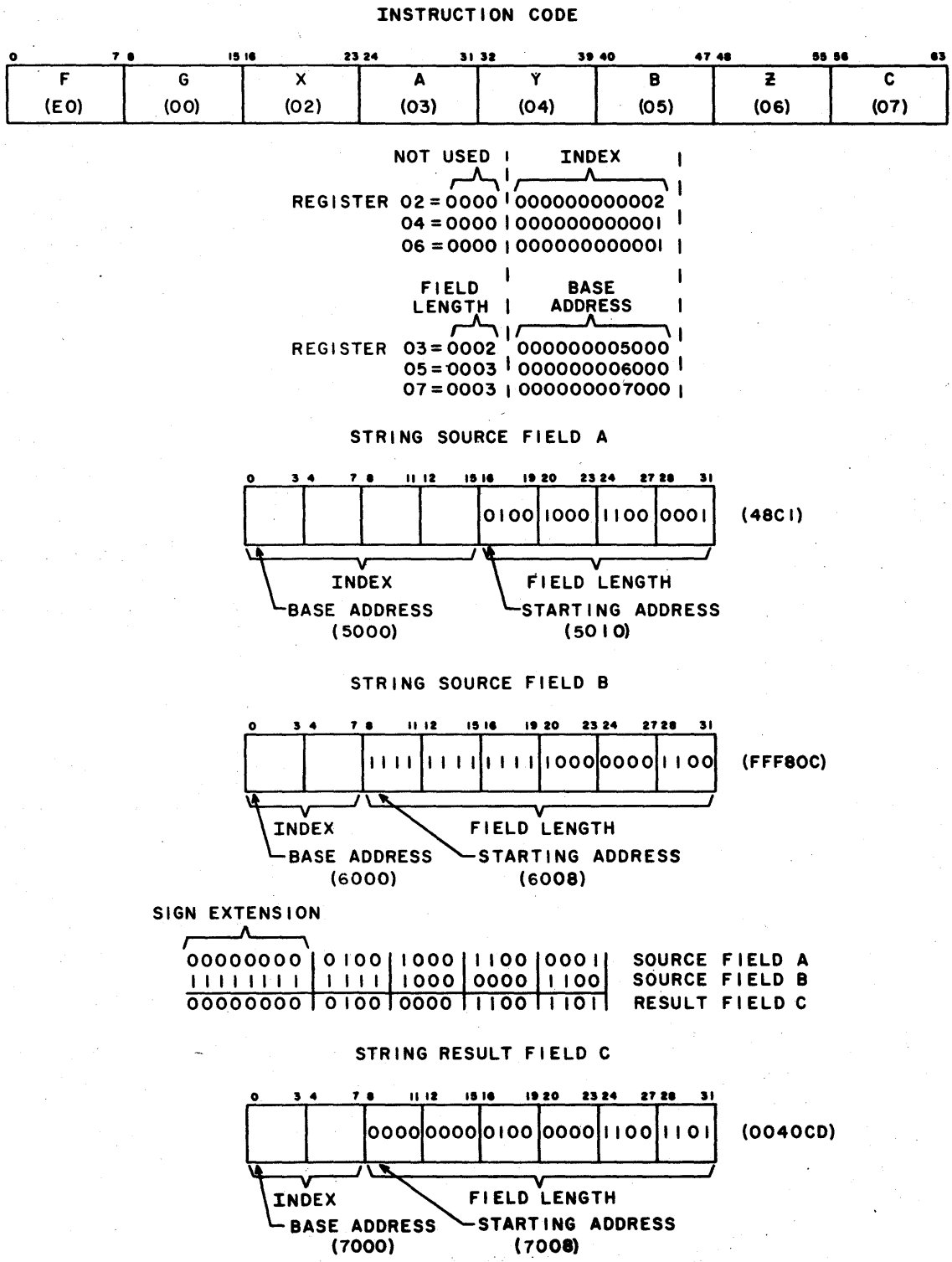


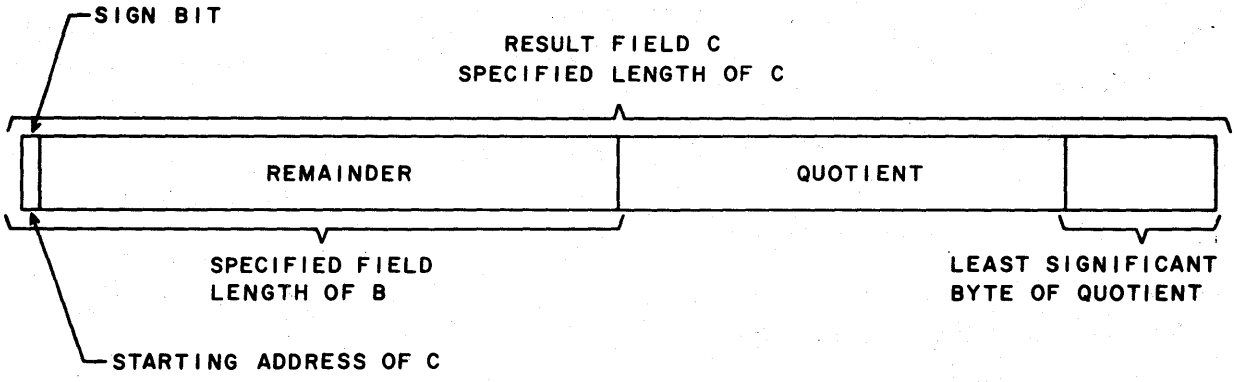
Figure 6-47. Example of Binary Add; A + B → C Instruction

E2 BINARY MPY; A • B → C

This instruction multiplies binary field A by binary field B, using two's complement arithmetic. The instruction produces a binary product which is stored as result field C with the sign bit extended, if necessary, to fill out the specified field length. If the C field overlaps the A or B field, the instruction results are undefined.

E3 BINARY DVD; A/B → C

This instruction divides binary field A by binary field B, using two's complement arithmetic. The result is a remainder, having a field length equal to the field length of B and a quotient with a field length equal to the specified length of C minus the specified length of B. Figure 6-48 shows that the remainder is stored at the B length portion of the C field, beginning at the starting address. The quotient is stored in the remaining portion of the C field length. The sign of the quotient is extended, if necessary, to fill the specified field length of C. If the C field overlaps the A or B field, the results of the instruction become undefined.



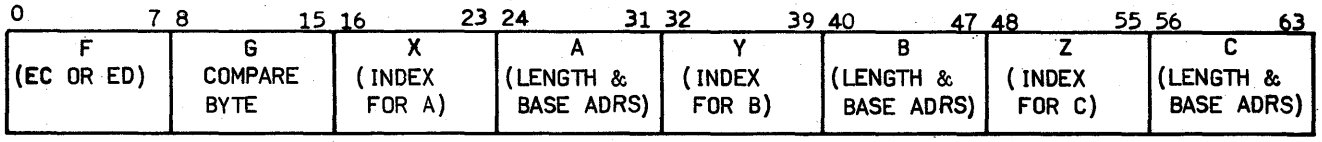
NOTE

The sign of the remainder conforms to (quotient x divisor) + remainder = dividend; that is, the sign of the remainder is the same as the sign of the dividend unless the remainder is 0 and the dividend is negative.

Figure 6-48. Format of Binary Divide Result Field

EC MODULO ADD A + B→C

ED MODULO SUB A – B→C



EC MODULO ADD A + B → C

This instruction performs a modulo add on the bytes in two binary strings, A and B. The source strings are considered positive. The instruction performs the add on a byte-by-byte basis from left to right and does not permit carries to propagate across byte boundaries. Each byte sum is compared to the byte in the G portion of the instruction code on the following basis.

Compare Condition	Result
(A byte + B byte) < G byte	(A byte + B byte) → C byte
(A byte + B byte) ≥ G byte	(A byte + B byte - G byte) → C byte

The G field may be assigned any value in the range of 0 through FF_{16} with 0 acting as though it were 100_{16}. Therefore, if the A byte plus the B byte is greater than or equal to 100_{16}, A byte plus B byte minus G is stored into the C byte.

If the A or B source string is shorter than the C string, the length of the A or B source string is extended with zero bytes until the length of the corresponding source string equals the length of the C string.

The compare byte in G may have any value in the range of 0 through 255_{10} = 0 through FF_{16}. A zero G value functions as a 256_{10} value.

At the termination of this instruction, data flag bits 53, 54, and 55 are set according to the results of the byte compare operation (table 6-25).

TABLE 6-25. DFB CONDITIONS FOR THE EC INSTRUCTION

DFB Bit	Conditions
53	(A byte + B byte) < G byte for all bytes
54	(A byte + B byte) ≥ G byte for one or more bytes but not for all bytes
55	(A byte + B byte) ≥ G byte for all bytes

ED MODULO SUB A-B → C

This instruction performs a modulo subtract on the bytes in two binary source strings, A and B. The binary source strings are considered positive. The instruction performs the subtracts on a byte-by-byte basis from left to right and does not permit borrows to propagate across byte boundaries. As part of each subtract operation, the A byte is compared to the B byte on the following basis.

Compare Conditions	Results
A byte $\geq$ B byte	(A byte - B byte) → C byte
A byte < B byte	(A byte - B byte + G byte) → C byte

If the A and/or B source string is shorter than the C string, the A and/or B source string is extended with zero bytes until the length of the corresponding source string equals the length of the C string.

Table 6-26 gives the conditions for setting data flags 53, 54, and 55.

TABLE 6-26. DFB CONDITIONS FOR THE ED INSTRUCTION

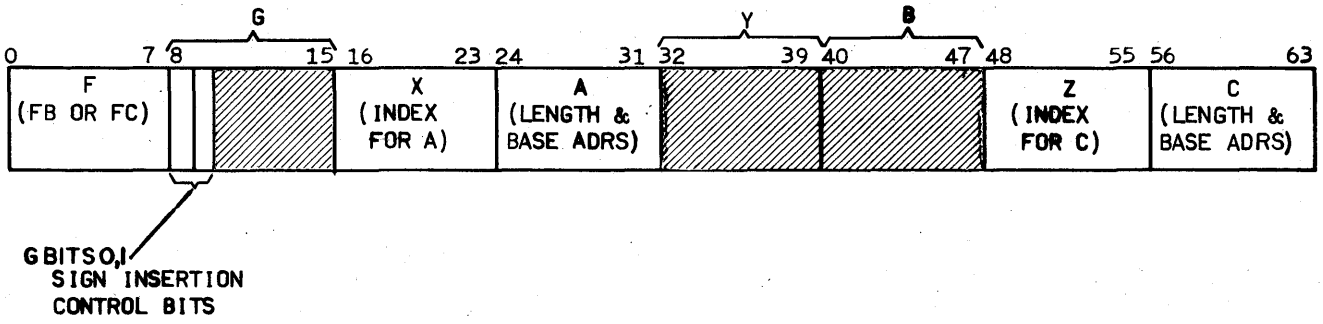
DFB Bit	Conditions
53	A byte < B byte for all bytes
54	A byte $\geq$ B byte for one or more bytes but not for all bytes
55	A byte $\geq$ B byte for all bytes

The byte in G may have any value in the range of 0 through 255_{10} (0 through FF_{16}). A zero G value functions as a 256_{10} value.

At the termination of this instruction, data flag bits 53, 54, and 55 are set according to the results of the byte compare operation (refer to table 6-23).

FB PACK ZONED TO BCD; A→C

FC UNPACK BCD TO ZONED; A→C



FB PACK ZONED TO BCD; A → C

This instruction converts a string data field in the zoned format into a result field C that is packed in the BCD format. All zone bits in the source field are discarded except the bits in the least significant byte which constitute the sign. Both the source and result fields must be specified by a field length. The operation proceeds from right to left. The Y and B designators and bits 2 through 7 of the G designator are not used and must be zeros. Bits 0 and 1 of the G designator control the translation and insertion of the sign bits.

If the source field contains fewer digits than the result field, the instruction inserts zeros in the high order digit positions of the result field (figure 6-49). The lengths of the source and result fields are item counts in bytes.

If the source field contains more digits than the result field can contain, the instruction truncates the result field by discarding the necessary number of high order digits in the source field.

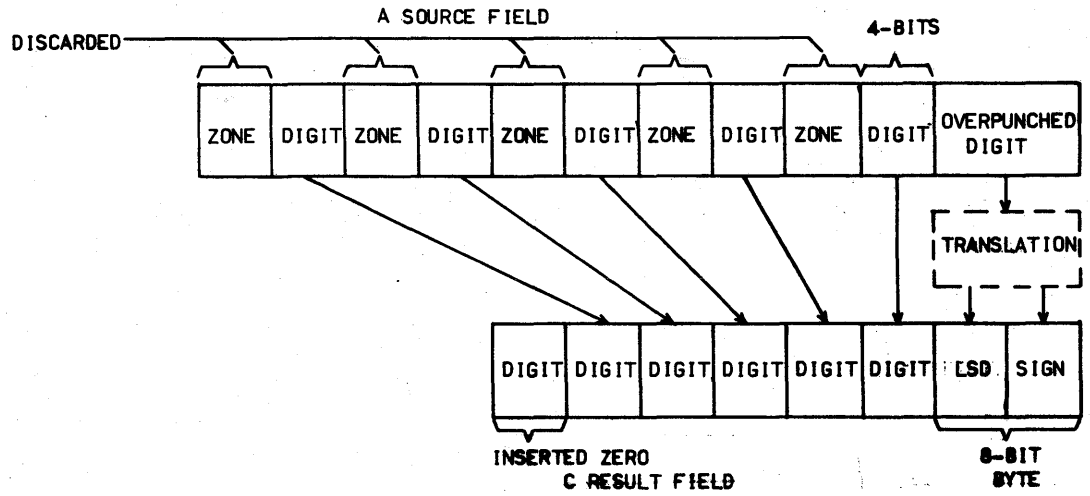


Figure 6-49. Example of Zoned to BCD Format Conversion
(G Bit 0 = 0, G Bit 1 = 0 and ASCII Selected)

Table 6-27 lists the digit and sign codes that are used in the pack operation. Six sign codes are recognized as valid codes.

TABLE 6-27. PACK ZONED TO BCD DIGIT AND SIGN CODES

Digit	Code	Sign	Code
0	0000	+	1010
1	0001	-	1011
2	0010	+	1100
3	0011	-	1101
4	0100	+	1110
5	0101	+	1111
6	0110		
7	0111		
8	1000		
9	1001		

G DESINATOR BIT 0 = 0, BIT 1 = 0 (ASCII MODE)

The rightmost byte of the A field is assumed to contain an overpunched digit which is translated into a sign and least significant digit (LSD) according to a translate table (table 6-28). For the remaining bytes, the FB instruction discards the zone bits and copies the data bits without checking the validity of the codes (figure 6-49).

TABLE 6-28. PACK ZONED TO BCD SIGN AND LSD TRANSLATION TABLE (ASCII MODE)

Character	Code	LSD	Sign†
0	30	0	A (+)
1	31	1	A
2	32	2	A
3	33	3	A
4	34	4	A
5	35	5	A
6	36	6	A
7	37	7	A
8	38	8	A
9	39	9	A
{	7B	0	A
A	41	1	A
B	42	2	A
C	43	3	A
D	44	4	A
E	45	5	A
F	46	6	A
G	47	7	A
H	48	8	A
I	49	9	A
}	7D	0	B (-)
J	4A	1	B
K	4B	2	B
L	4C	3	B
M	4D	4	B
N	4E	5	B
O	4F	6	B
P	50	7	B
Q	51	8	B
R	52	9	B

†The preferred signs are shown in hexadecimal notation (for example, A = 1010 and B = 1011).

G DESIGNATOR BIT 0 = 0, BIT 1 = 0 (EBCDIC MODE)

As in the previous operation, the operation discards the zone bits and copies the data bits without checking the validity of the codes (figure 6-50). The operation then samples the sign (assumed to be the leftmost four bits of the least significant byte of the A field) and inserts the appropriate preferred sign code in the C field according to table 6-29. If the sign position of the A field does not contain one of the recognized six sign codes (table 6-23), the rightmost four bits of the C field become undefined.

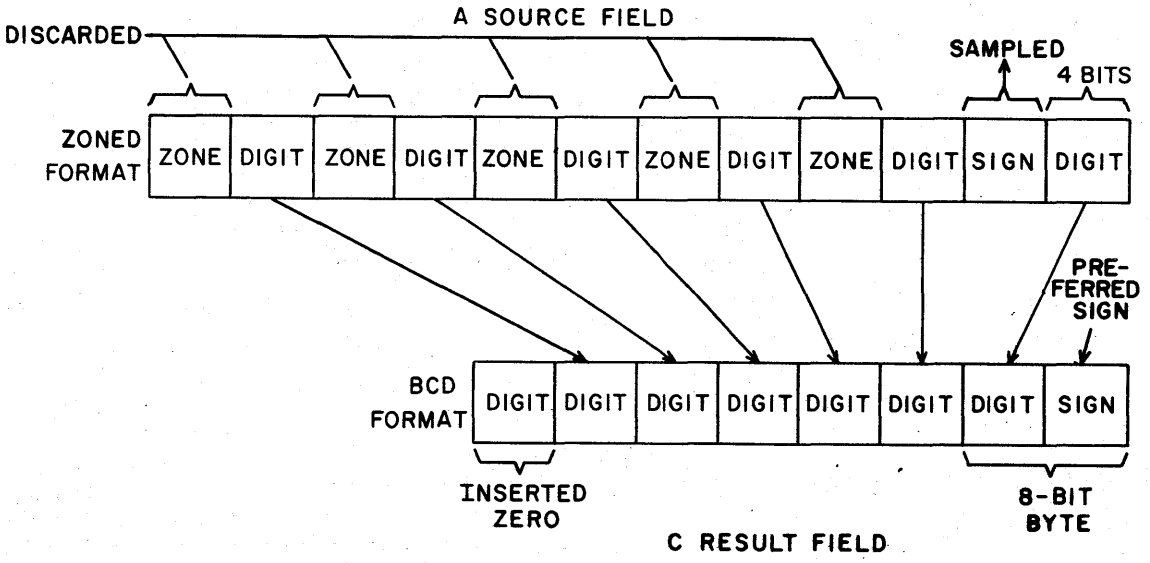


Figure 6-50. Example of Zoned to BCD Format Conversion (G Bit 0 = 0 and EBCDIC Selected)

TABLE 6-29. PREFERRED SIGN CODES

Sign	ASCII Mode	EBCDIC Mode
Positive	1010	1100
Negative	1011	1101

G DESIGNATOR BIT 0 = 0 AND G DESIGNATOR BIT 1 = 1

The instruction becomes undefined.

G DESIGNATOR BIT 0 = 1 AND BIT 1 = 0

The operation assumes that the rightmost byte of the A field contains a sign character according to the ASCII or EBCDIC selection (table 6-30). If the byte does not contain a sign character or a zoned digit, the content of the C field becomes undefined. The instruction discards the zone bits in the remaining bytes of the A field and copies the digits in the C field without checking for validity.

TABLE 6-30. ZONE BITS AND SIGN CODES

Character Types	ASCII Code	EBCDIC Code
Zone bits	0011XXXX†	1111 XXXX†
Sign positive	0010 1011	0100 1110
Sign negative	0010 1101	0110 0000
† X's denote a digit code.		

If the rightmost byte of the A field contains the proper representation for a sign character, the instruction inserts the preferred 4-bit positive/negative sign code in the rightmost four bits of the C field when it detects a positive/negative sign character in the rightmost byte of the A field (figure 6-51).

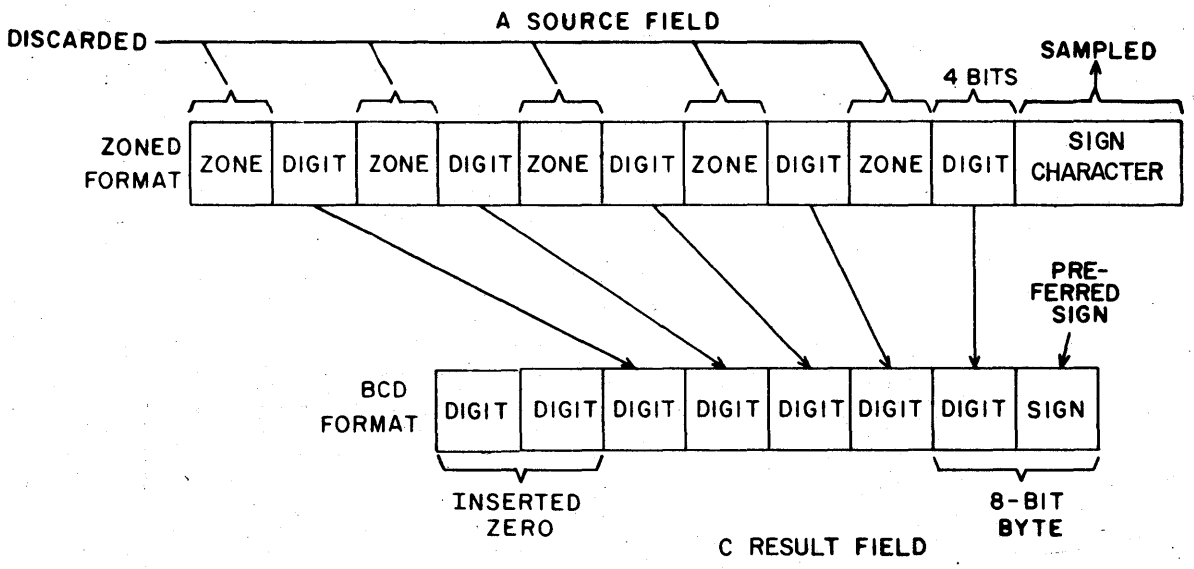


Figure 6-51. Example of Zoned to BCD Format Conversion (G Bit 0 = 1 and G Bit 1 = 0)

G DESIGNATOR BIT 0 = 1 AND BIT 1 = 1

The pack operation inserts the preferred positive sign in the least significant four bits of the rightmost byte of the C field (figure 6-52). The instruction then discards the zone digits and copies the digit bits in the C field as previously described.

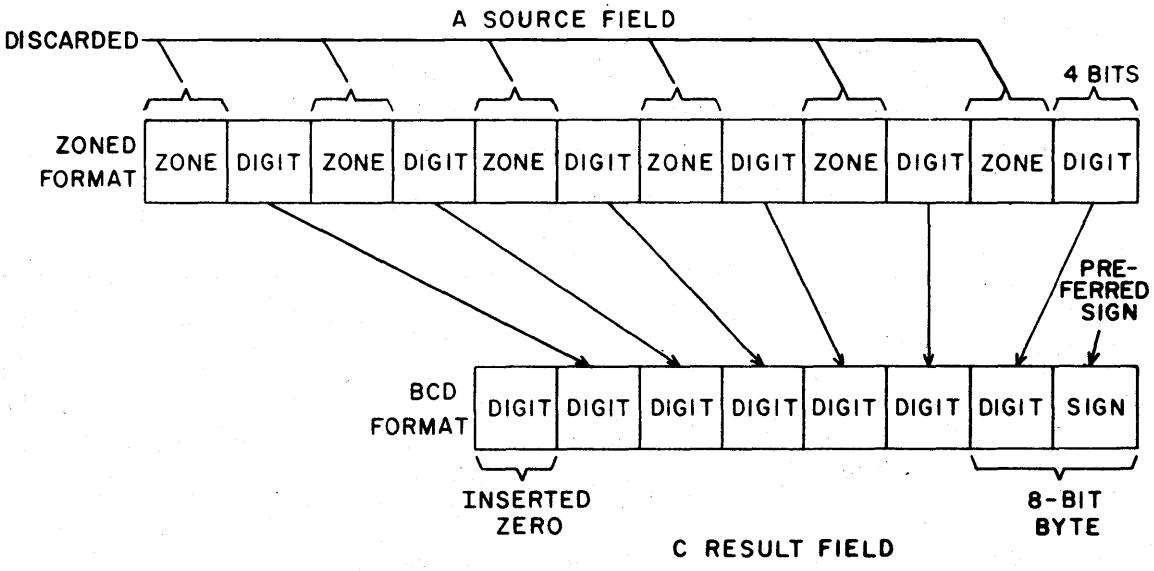


Figure 6-52. Example of Zoned to BCD Format Conversion (G Bit 0 = 1 and G Bit 1 = 1)

FC UNPACK BCD TO ZONED; A → C

This instruction converts a string source field A in packed BCD format to result field C that is in the zoned format. The sign of the C field is determined by sampling the sign portion of the packed BCD number. The instruction inserts the preferred sign character in the corresponding portion of the C field under the control of G designator bits 0 and 1. The operation proceeds from right to left.

If the source field contains fewer digits than the result field can store, the instruction fills out the result field with characters consisting of the zone code with a zero digit. If the source field contains more digits than the result field can store, the necessary number of digits are discarded from the source field, truncating the result field.

The instruction must contain length specifications for both the source and result fields. The Y and B designators and bits 2 through 7 of the G designator are undefined and must be zeros.

The instruction generates the zone bits, sign characters, and preferred sign bits according to the ASCII or EBCDIC selection (tables 6-29 and 6-30).

The following paragraphs describe the translation and insertion of the sign bits for each condition of G designator bits 0 and 1. In each case, the digits are copied and the zone bits generated in the result field as previously described. These operations are not described individually for each case.

G DESIGNATOR BITS 0 = 0 AND BIT 1 = 0 (ASCII MODE)

The operation translates the sign and LSD and places the appropriate overpunched digit in the rightmost byte of the C field (figure 6-53) according to the translations listed in table 6-31. If the rightmost four bits of the A field do not contain one of the six sign codes, the rightmost byte of the C field becomes undefined.

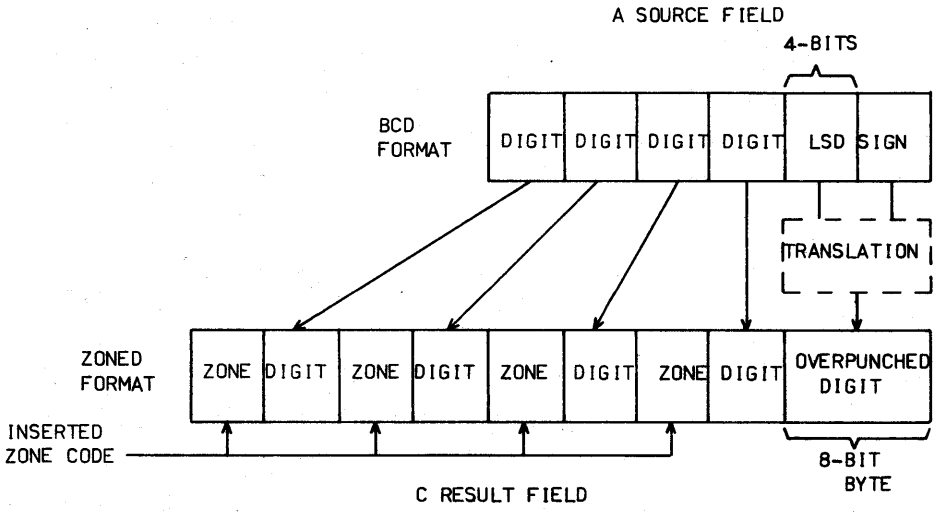


Figure 6-53. Example of BCD to Zoned Format Conversion (G Bit 0 = 0 and G Bit 1 = 0 ASCII Mode)

TABLE 6-31. UNPACK BCD TO ZONED SIGN AND LSD TRANSLATION TABLE (ASCII MODE)

Sign	LSD	Character	Code	Sign	LSD	Character	Code
+	0	{	7B	-	0	}	7D
+	1	A	41	-	1	J	4A
+	2	B	42	-	2	K	4B
+	3	C	43	-	3	L	4C
+	4	D	44	-	4	M	4D
+	5	E	45	-	5	N	4E
+	6	F	46	-	6	O	4F
+	7	G	47	-	7	P	50
+	8	H	48	-	8	Q	51
+	9	I	49	-	9	R	52

G DESIGNATOR BIT 0 = 0 AND BIT 1 = 0 (EBCDIC MODE)

The sign in the rightmost four bits of the A field is sampled and the appropriate preferred sign code is inserted in the C field (figure 6-54). If the rightmost four bits of the A field do not contain one of the six recognized sign codes (table 6-27), the four bits in the sign position are undefined.

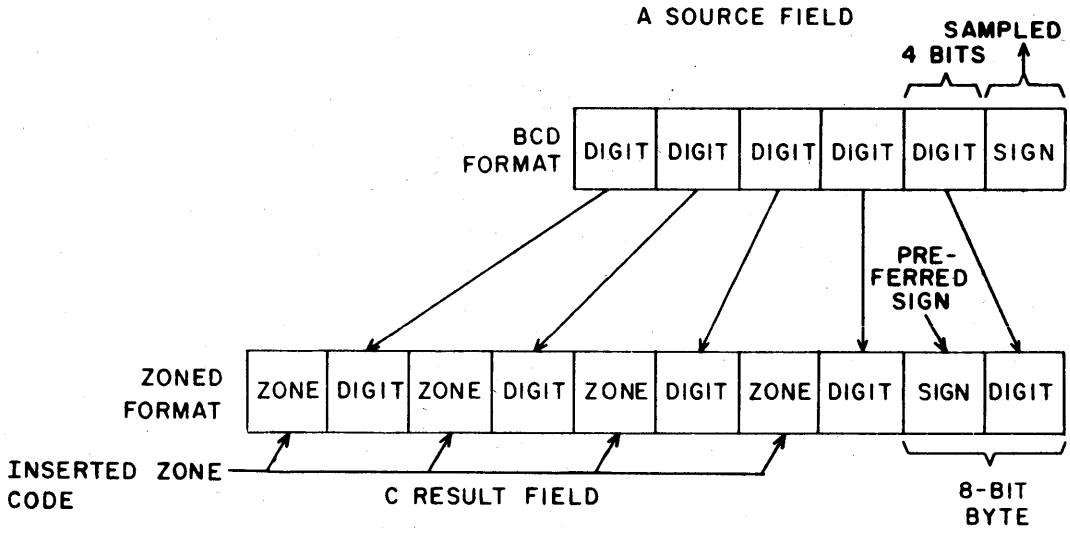


Figure 6-54. Example of BCD to Zoned Format Conversion
(G Bit 0 = 0 and G Bit 1 = 0 EBCDIC Mode)

G DESIGNATOR BIT 0 = 0 AND BIT 1 = 1

The instruction becomes undefined.

G DESIGNATOR BIT 0 = 1 AND BIT 1 = 0

The instruction assumes that the rightmost four bits of the A field (figure 6-55) contain one of the six valid sign codes. The operation inserts the appropriate 8-bit sign character for the positive or negative sign code according to the ASCII or EBCDIC selection (table 6-30) in the rightmost byte of the C field. If the sign position of the A field does not contain one of the six recognized sign codes, the rightmost byte of the C field becomes undefined.

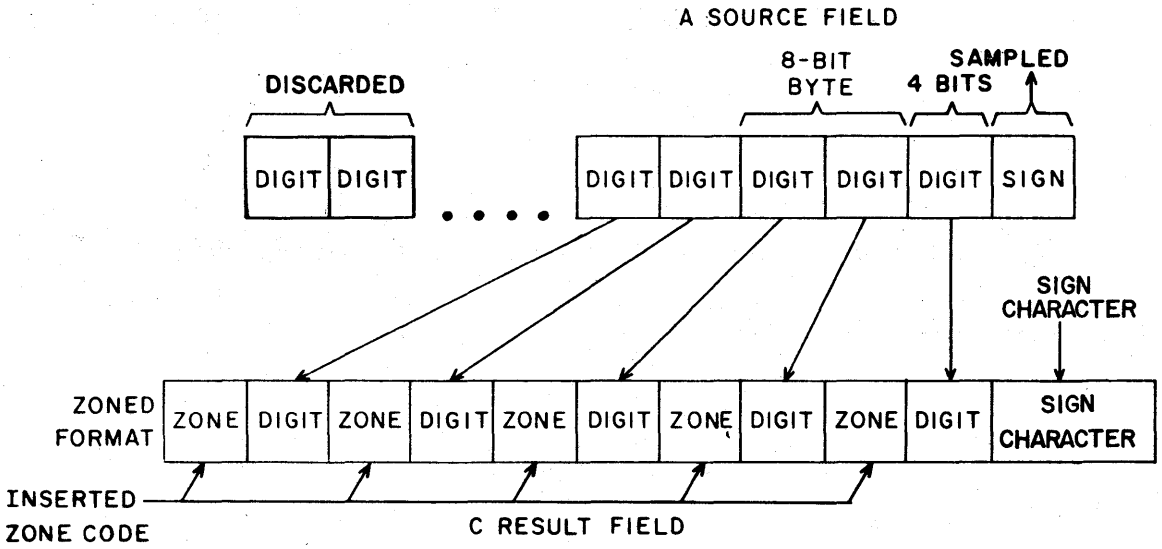


Figure 6-55. Example of BCD to Zoned Format Conversion (G Bit 0 = 1 and G Bit 1 = 0)

G DESIGNATOR BIT 0 = 1 AND BIT 1 = 1

The instruction (figure 6-56) samples the rightmost four bits of the A field, inserts the appropriate 8-bit sign character for the positive or negative sign code, and sets data flag bit 38 (decimal data fault) if the sign code is negative. If the sign position of the A field contains no recognized sign code, the state of data flag bit 38 and the rightmost byte of the C field become undefined. The digits in the A field are copied and the zone codes are generated in the C field as previously described.

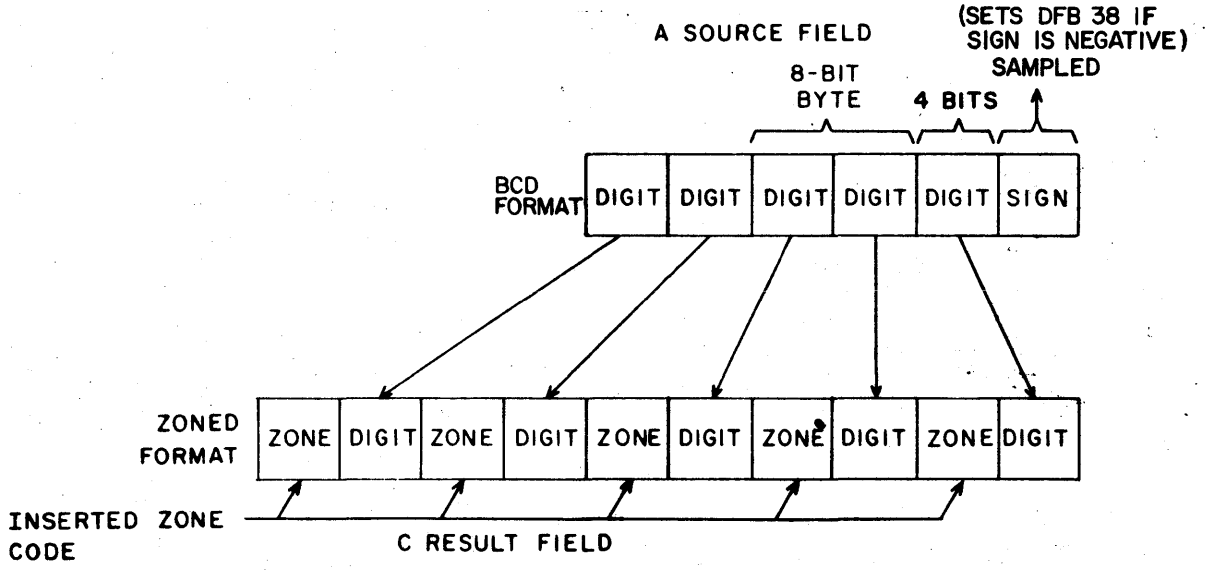
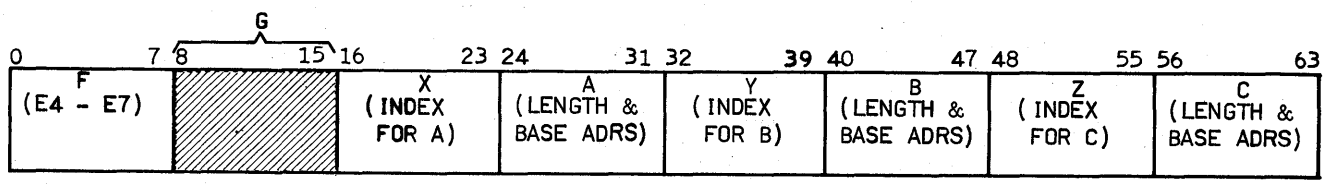


Figure 6-56. Example of BCD to Zoned Format Conversion (G Bit 0 = 1 and G Bit 1 = 1)

E4 DECIMAL ADD; A + B→C
E5 DECIMAL SUB; A — B→C
E6 DECIMAL MPY; A • B→C
E7 DECIMAL DIV ; A / B→C



The decimal add, subtract, multiply, and divide instructions perform the indicated arithmetic operations on the A and B source fields which are in the BCD packed format. The result field is also in the packed BCD format. All of the indexes and field lengths are item counts in bytes. These instructions extend the sum, difference, product, and quotient to the left with zero digits, if necessary, to fill the specified result field length.

If the C designator or the C field length is zero, the instruction sets no data flag bits and becomes a no-op. If the A and/or B designator is a zero or if the field length of A and/or B is a zero, the instruction uses a positive zero for the corresponding source field.

If the instruction detects a sign in a digit position or a digit in a sign position, data flag 38 (decimal data fault) is set. When this condition occurs, the state of data flag bit 39 (string arithmetic overflow) becomes undefined. Data flag bit 39 sets if the instruction truncates nonzero result digits (too small a result field) or attempts a divide with a zero divisor. The contents of output field C is undefined whenever data flag bit 38 or 39 sets.

The G designator is not used and must be all zeros.

E4 DECIMAL ADD; A + B → C and E5 DECIMAL SUB; A - B → C

These instructions add/subtract source field B to/from source field A. The sum/difference is stored in result field C. All data fields are in the packed BCD format. The arithmetic operations proceed from right to left. These instructions force a zero result positive. The field lengths are specified in bytes. Figure 6-57 shows an example of a decimal add; A + B→C (E4) operation with assumed instruction codes, register contents, and string data fields. The index values are shifted three positions before they are added to the base addresses, and the result field is extended with one zero digit to fill out the specified field length.

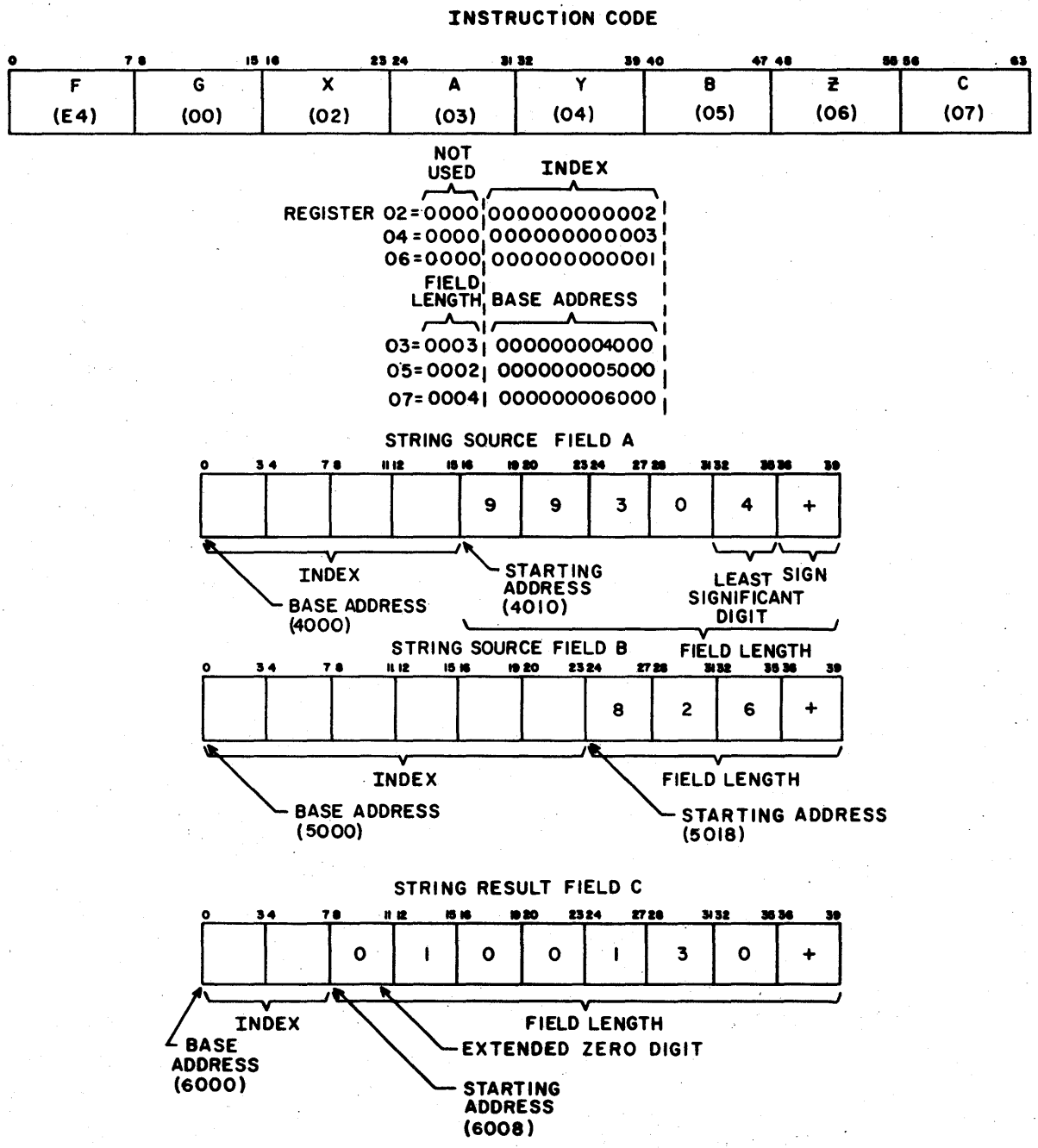


Figure 6-57. Example of Decimal Add; A + B →C Instruction

E6 DECIMAL MPY; A • B → C

This instruction multiplies source field A by source field B and stores the product in result field C. All data fields are in the packed BCD format. The sign of the product follows the rules of algebra. If the field lengths of either or both source fields are initially equal to zero, the result is forced to a positive zero. If the result field overlaps either source field, the instruction produces undefined results. The field lengths are expressed in bytes.

E7 DECIMAL DVD; A/B→C

This instruction divides the dividend in source field A by the divisor in source field B and stores the quotient and remainder in result field C (figure 6-58). All data fields are in the packed BCD format. The sign of the quotient follows the rules of algebra. The sign of the remainder equals the sign of the dividend. If the result field overlaps either source field, the instruction produces undefined results. The field lengths are expressed in bytes.

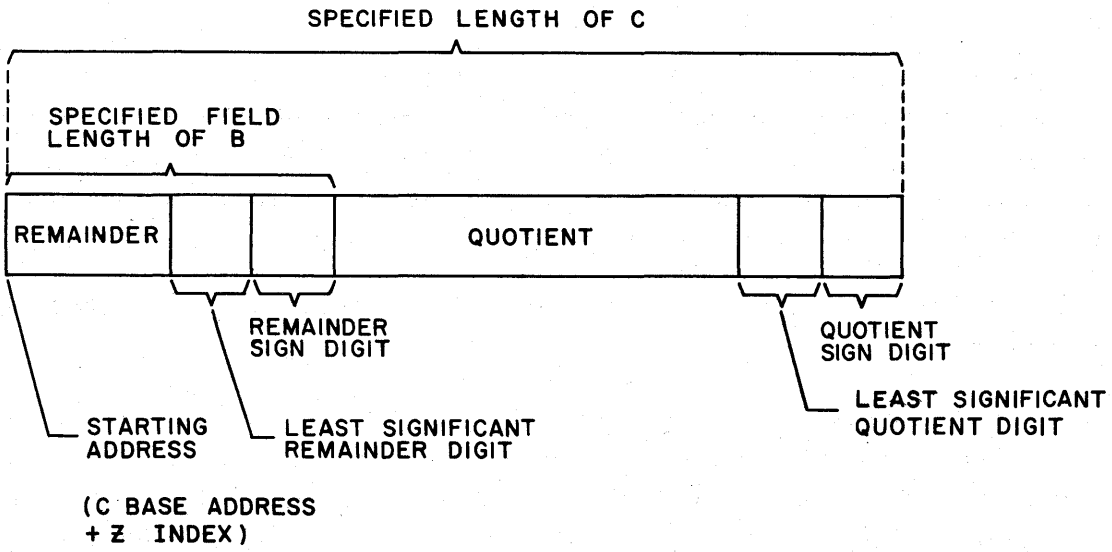
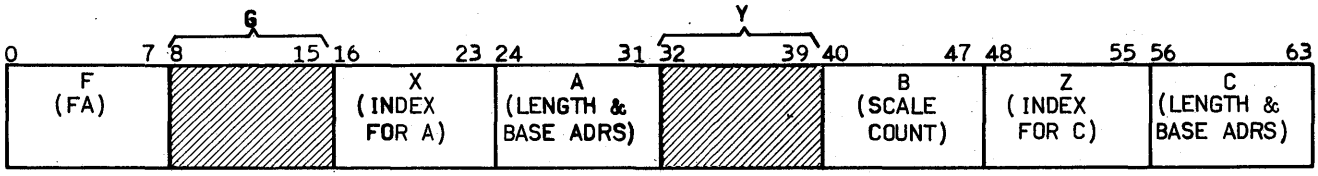


Figure 6-58. Format of Decimal Divide Result Field

FA MOVE AND SCALE; A→C



This instruction moves source field A to result field C and scales the source field within the result field right or left by as many decimal positions as specified by the scale count contained in register B. The scale count represents an item count of the number of 4-bit, decimal digits to be shifted. The scale count is expressed as a two's complement, signed integer, contained in the rightmost 48 bits of register designated by B. The shift is relative to the right end of the result field. The G and Y designators are not used and must all be zeros. The source and result fields are in the packed decimal format.

If the scale count is positive, the instruction shifts the source field left within the result field. The instruction inserts zeros in the rightmost, decimal digit positions of the result field that are vacated by the left shift. However, the sign digit remains in the rightmost four bits of the result field. With a positive scale count, the operation is equivalent to multiplying an integer by the positive power of 10_{10}.

The scaling operation proceeds from right to left. The overlapping of fields produces undefined results.

If the scale count is negative, the instruction shifts the source field right within the result field. This shift is end-off; thus, digits that are shifted into the sign position are discarded. The original sign of the source field is always retained in the sign position of the result field. The instruction inserts BCD zeros in the leftmost digit positions of the result field vacated by the right shift.

If the source-field length is shorter than the length of the result field, the instruction extends the result field with zero digits. If the relative magnitudes of the source and result field lengths and a positive scale count (left shift) prohibit the storage of the BCD number in the result field, the necessary number of high order digits of the result field are truncated. If any nonzero digit is truncated, the instruction sets data flag bit 39 (string arithmetic overflow).

The indexes and field lengths are expressed in bytes. The instruction terminates when the result field is filled and after checking the remaining source-field characters on a right shift for a nonzero character.

Figure 6-59 shows an example of a move and scale instruction with assumed instruction codes, register content, and source field. The negative scale count in register B denotes a right shift of two. As a result, the instruction shifts the low order, two BCD digits off the right end. The instruction extends the field length with BCD zeros and retains the original sign in the sign position of the result field.

Figure 6-60 shows an example of a move and scale instruction; however, a positive scale count is used. Thus, the instruction left-shifts the field two BCD positions and inserts zero BCD digits in the low order, two BCD positions. Since the length of the result field is set at four bytes, the high order, two BCD digits are truncated. The instruction sets data flag bit 39 (string arithmetic overflow), indicating that nonzero digits of the result field were truncated.

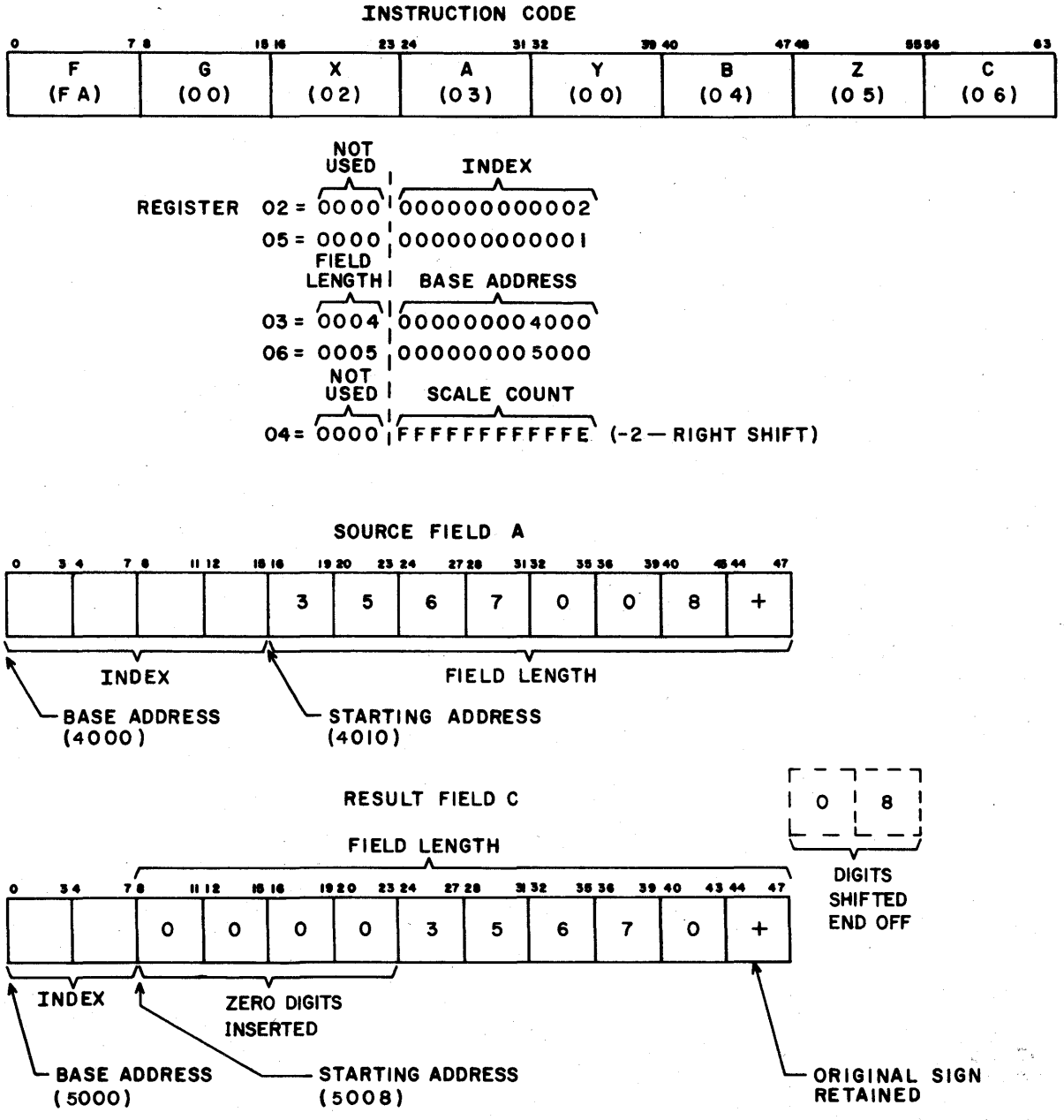


Figure 6-59. Example of Move and Scale; A →C Instruction with Negative Scale Count

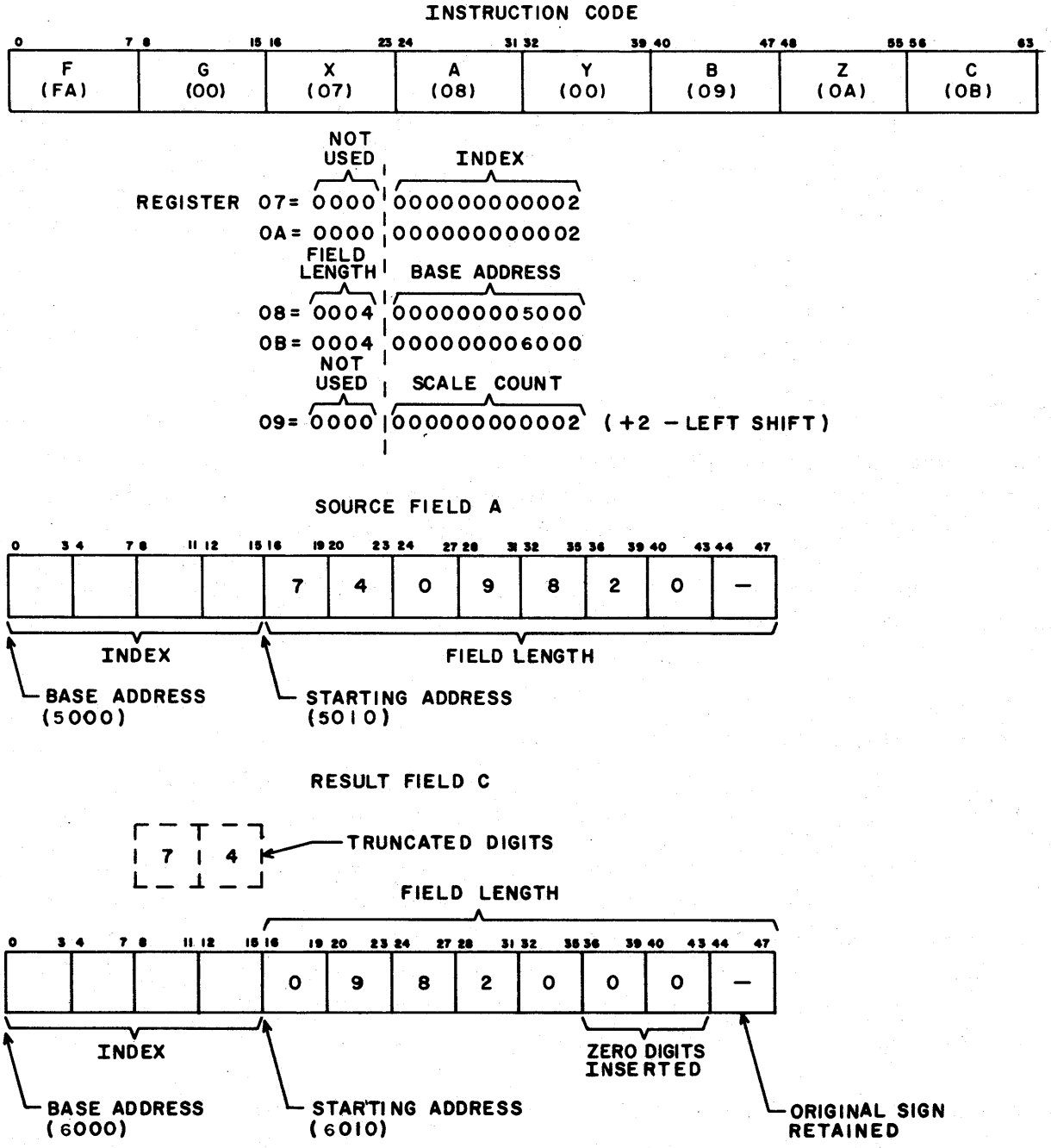
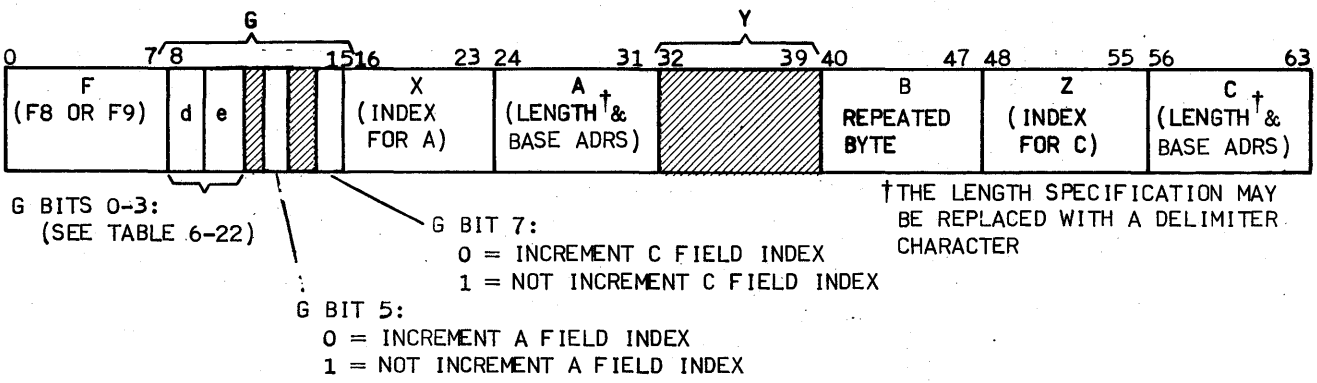


Figure 6-60. Example of Move and Scale; A →C Instruction with Positive Scale Count

F8 MOVE BYTES LEFT; A➔C

F9 MOVE BYTES LEFT, ONES COMP . A➔C



F8 MOVE BYTES LEFT; A→C

This instruction moves source field A to result field C. The bytes in the field are considered from left to right. Thus, the most significant byte of the source field is moved to the most significant byte position of the result field.

The d and e designators in the instruction indicate whether field lengths or delimiting is specified for the A and C fields.

When the destination field is delimited by a length rather than a delimiter character, the following rules apply.

1. If the origin field is shorter than the destination field, the destination field is filled in with the repeated byte found in the B designator of the instruction.
2. If the origin field is longer than the destination field, the operation is truncated when the destination field is exhausted. For this case, if the origin field was character delimited, the origin field is searched for the delimiter character so that its associated index may be properly incremented. If the origin field was length delimited, its associated index is incremented by the length rather than the actual number of bytes transferred.

When the destination field is delimited by a character rather than a length, the move continues until the origin field reaches its length specification. The operation is then terminated, and the delimiter character specified for the destination is stored as the last byte of the destination field. The delimiter character is stored even if the A field length is initially zero.

The index increments allowed for the A and C fields are specified by G designator bits 5 and 7 in table 6-32.

TABLE 6-32. INDEX INCREMENTS FOR A AND C FIELDS FOR F8 AND F9 INSTRUCTIONS

Field	G Bit 5	G Bit 7	Index Increment†
A	0	-	Full increment
A	1	-	No increment
C	-	0	Full increment
C	-	1	No increment

† For a complete definition of index incrementing, refer to Index Increments at the beginning of the string instructions.

The Y designator and G designator bits 4 and 6 are not used and must be zeros.

Figure 6-61 shows an example of a move bytes left instruction with assumed instruction codes, register content, and source field. The G designator gives d and e values of 10_2. Thus, the rightmost eight bits of the length specification for A and C denote the delimiter character for the respective field. In the example, G designator bits 5 and 7 are both zeros. Thus, the A and C fields are incremented.

The instruction moves the bytes in field A to the corresponding positions of field C, beginning at the starting address of both fields. When the delimiter character (FF) is detected in field A, the operation terminates with the insertion of the delimiter character (EE) in the result field. Before termination, the instruction increments the indexes for A and C by their respective field lengths.

F9 MOVE BYTES LEFT, ONES COMP. A→C

This instruction operates identically to the move bytes left; A→C instruction except that the one's complement of field A is moved to field C. If a delimiter field is specified for the source field, the instruction searches the uncomplemented field for the delimiter character. The instruction complements only the data in the source field. Neither the repeated byte (when used) nor the delimiter character specified for the result field is complemented.

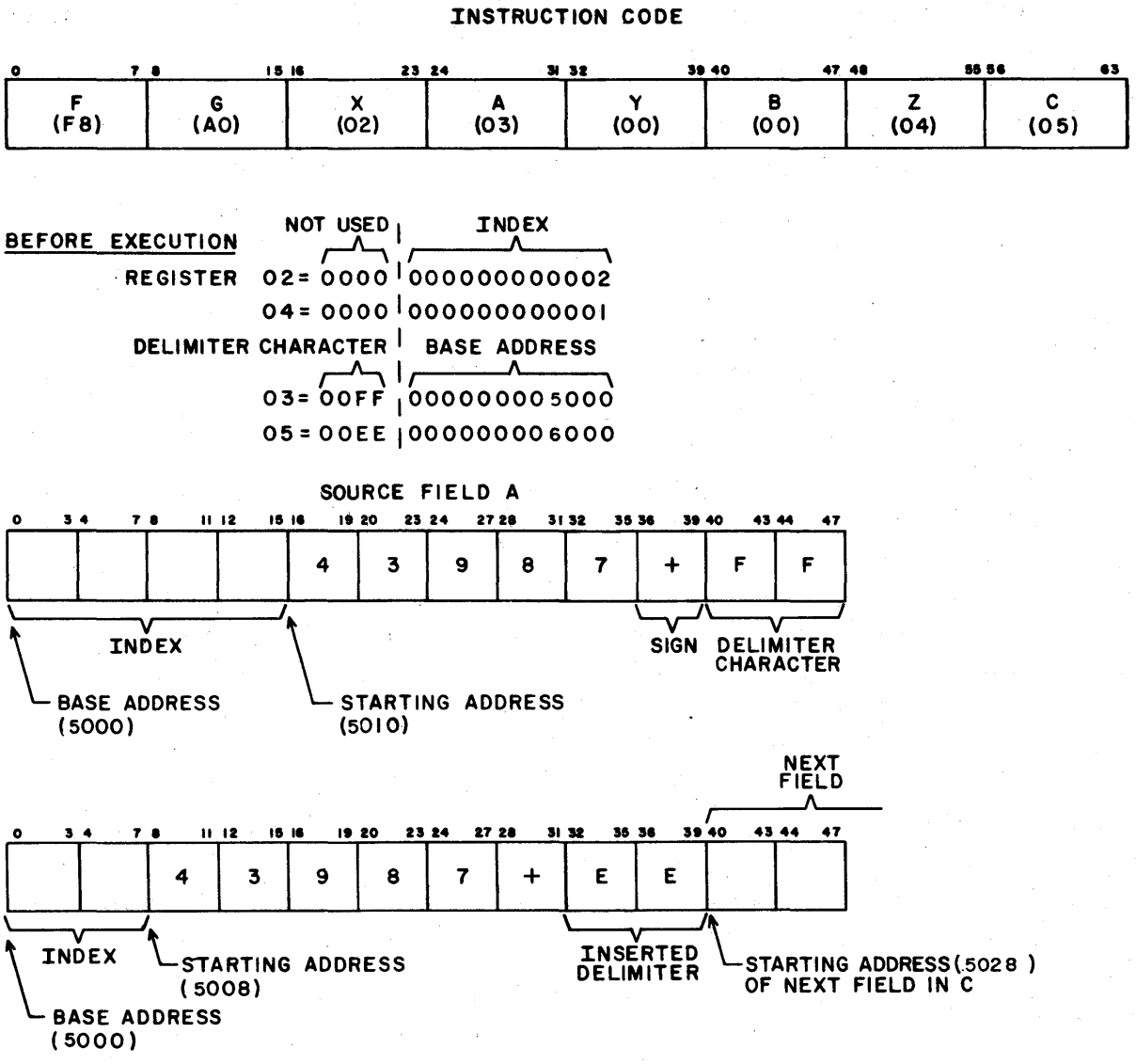
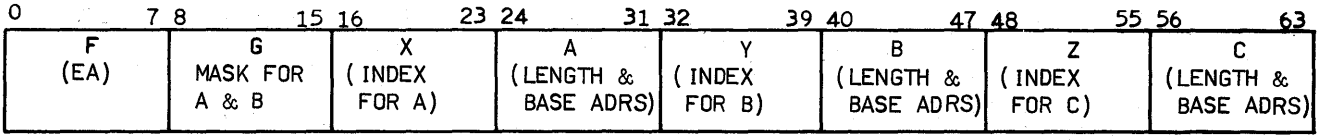


Figure 6-61. Example of Move Bytes Left; A →C Instruction

EA MERGE PER BYTE MASK A, B PER G→C



This instruction merges the bits from the bytes in source field A with the bits from the bytes in source field B according to the 8-bit mask in the G designator portion of the instruction word. The result is stored in corresponding bytes of result field C. The instruction uses bits of A corresponding to one bits in the mask byte and bits of B corresponding to zero bits in the mask byte. The operation proceeds from left to right; thus, the leftmost byte of field A is merged with the leftmost byte of field B and is stored in the leftmost byte of field C.

If one of the two source fields is shorter than the other, the instruction extends the shorter source field with null bytes (00_{16}). If the result field is shorter than the longer source field, the operation terminates when the result field is filled. If the result field is longer than either source field, the instruction fills out the result field with null bytes.

Figure 6-62 is an example of a merge byte mask instruction used to convert zoned ASCII to zoned EBCDIC formats. The example uses assumed instruction codes, register content, and source fields. The mask (G designator) is expanded below the instruction code. Positions 8 through 11 of the mask contain zero bits while positions 12 through 15 contain one bits.

Thus, the instruction substitutes the zone bits of source field B for the zone bits in source field A in corresponding positions of result field C. Similarly, the one bits in the mask enable the transfer of the digit bits in source field A to corresponding positions of result field C. As a result, the zone bits from field B are merged with the digit bits from field A and are stored in corresponding bytes of field C.

Since the assumed length of result field C is one byte longer than either source field, the instruction inserts a null byte to fill the field. No index incrementing takes place for this instruction.

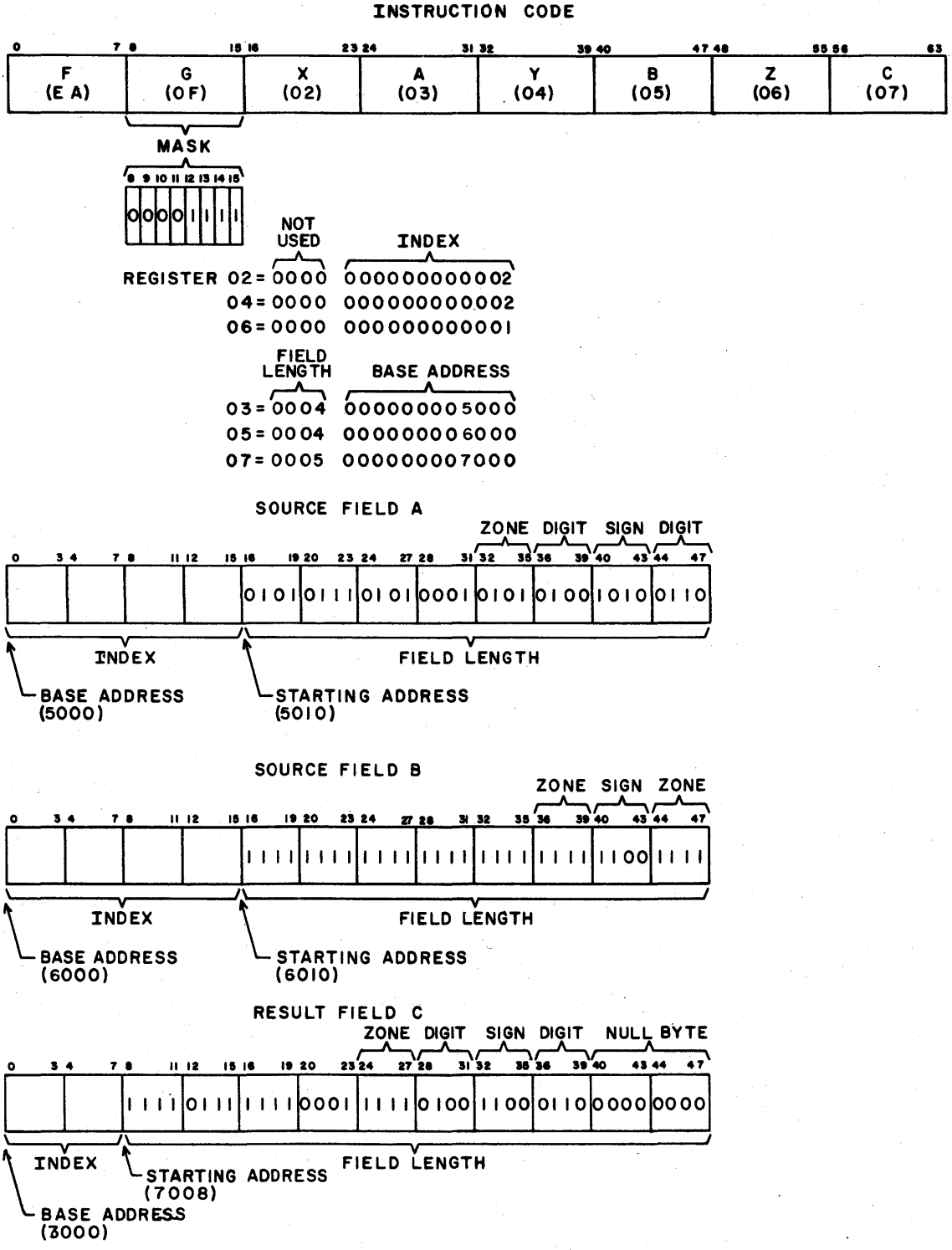
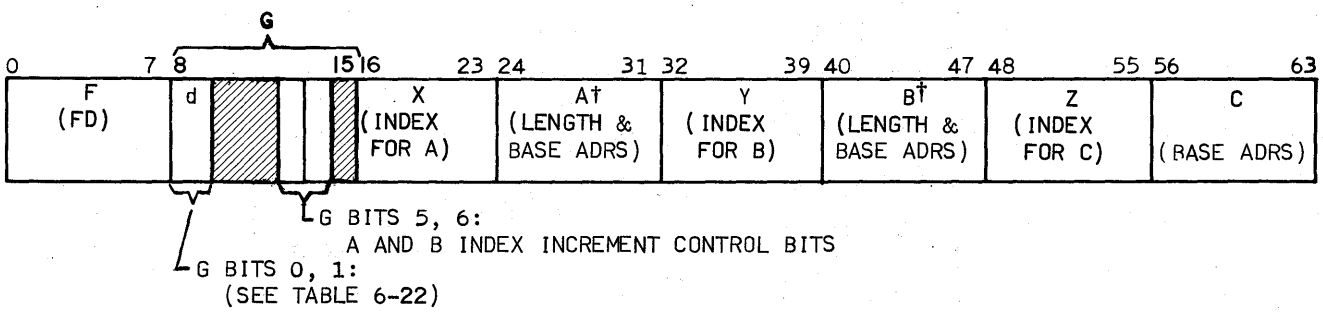


Figure 6-62. Example of Merge Per Byte Mask A, B Per G→C Instruction

FD COMPARE BYTES A, B PER MASK FIELD C



This instruction compares the bytes in field A with the bytes in field B for masked inequality. The instruction compares the bits in the pair of bytes only where corresponding bits in mask field C are ones. The comparison continues byte-by-byte from left to right until the instruction detects inequality of a byte pair or one of the following occurs.

1. Both of the source fields terminate
2. A and B field delimiter comparison††

The shorter source field is extended with blanks. Figure 6-61 shows the basic format of the data source and mask fields for the compare bytes A, B per mask field C instruction.

If the C designator portion of the instruction is zero, the instruction uses a mask containing all ones. The length of this mask is extended until one of the termination conditions is detected. If a mask field is used, the length specification is undefined. As shown in figure 6-63, the mask field must be at least as long as the longer of the two source fields.

NOTE

If the mask field is shorter than the longer source field, the instruction will continue to read consecutive bytes of field C until a normal terminating condition is detected. Thus, the results of such an operation would be undefined.

† The length specification may be replaced with a delimiter character.

†† Termination of the instruction does not occur with one field delimiter hit. Instead, beginning with this field delimiter byte, the input field is extended with blank bytes.

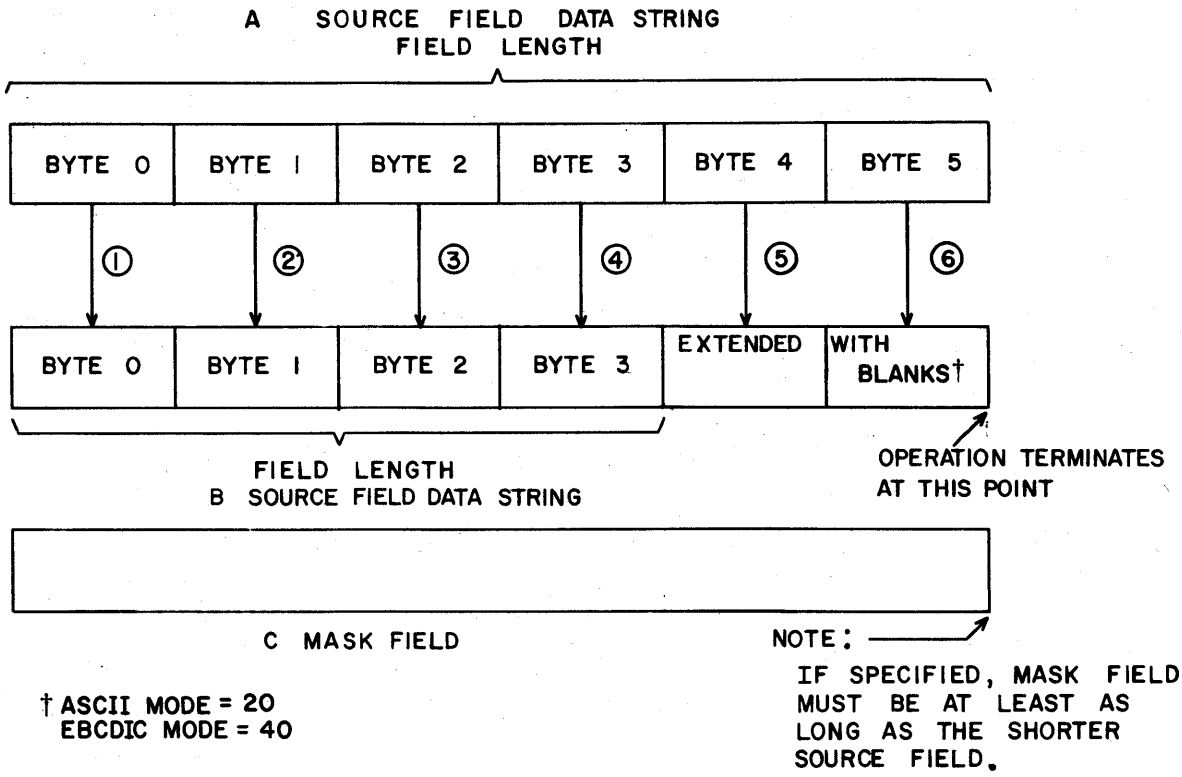


Figure 6-63. Basic Field Formats for Compare Bytes A, B Per Mask Field C Instruction

TERMINATION DUE TO MASKED INEQUALITY

If the instruction terminates because it detects masked inequality of a byte pair, the indexes of the two source fields are incremented by the same value if enabled by the corresponding A, B index control bit (table 6-32). This value equals the number of masked byte compares made before (but not including) the compare that caused termination.

TERMINATION DUE TO EXHAUSTING A SOURCE FIELD

If the instruction terminates because the source fields are exhausted, the instruction increments each source field index by the corresponding field length (table 6-33). In this case, the masked operands (source fields) are equal. If delimiter characters are used, the instruction searches each source field for the corresponding delimiter character. The index associated with each source field is incremented so that the corresponding base address plus the index locates the first byte of the next field. The types of length specification (length, single character delimiter, or 16-bit delimiter) for fields A and B are equal since the d designator in the instruction word governs the termination of both fields (table 6-24).

TABLE 6-33. INDEX INCREMENTS FOR COMPARE BYTES A, B PER MASK FIELD C INSTRUCTION

Field	G Bit 5	G Bit 6	Index Increment †
A	0	-	Full increment if equal Partial increment if not equal
	1	-	No increment
B	-	0	Full increment if equal Partial increment if not equal
	-	1	No increment
C			No increment

†For a complete definition of index increment, refer to Index Increments at the beginning of the string instructions.

DATA FLAG BITS

Before the instruction exits, data flag bit 53, 54, or 55 is set according to the result of the byte compare operations. Table 6-34 lists the three data flag bits and the condition for setting the corresponding bit.

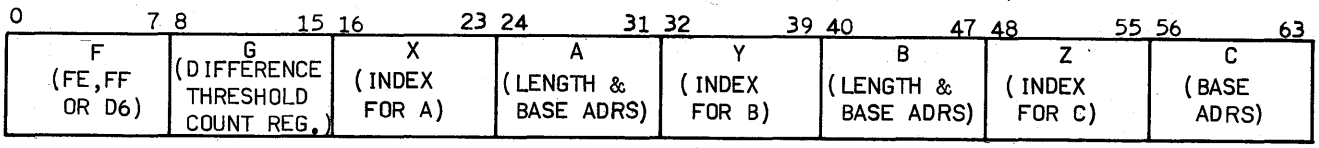
TABLE 6-34. DFB CONDITIONS FOR THE FD INSTRUCTION

DFB Bit	Condition
53	Masked operands are equal
54	First masked operand is greater (A > B)
55	First masked operand is less (A < B)

FE SEARCH FOR MASKED KEY BYTE; A, B PER C, G

FF SEARCH FOR MASKED KEY WORD; A, B PER C, G

D6 SEARCH FOR MASKED KEY BIT; A, B PER C,G



FE SEARCH FOR MASKED KEY BYTE; A, B PER C, G

This instruction searches source field A (reference field) for a match with source field B (key field). The first search compares the first byte of field A with the first byte of field B. If there is no difference in the comparison, the instruction compares the second byte of field A with the second byte of field B. This process continues until the key field is exhausted or the instruction detects a difference in the comparison of a pair of bytes. If the entire key field is compared with a portion of the reference field with no differences in the byte compares, a match results and the instruction terminates. If a compare difference is found, the instruction terminates that search and begins a new search by comparing the first byte of field B with the second byte of field A, the second byte of field B with the third byte of field A, and so on. This process continues until the key field B is exhausted or a compare difference is detected. The instruction continues this process of repeated searches until it detects a match or searches the reference field for all possible matches. If no match is made, the maximum number of searches is equal to the length of A minus the length of B plus one (A-B+1). If no match is detected, data flag bit 37 (select condition not met) is set.

The A index is incremented by one after each search not resulting in a match. However, if no match is found, the A index is increased by the length of the A field. When a match is found, the A index provides a means of locating the portion of the reference field matching the key field. Table 6-35 lists index increments for the instruction.

TABLE 6-35. INDEX INCREMENTS FOR SEARCH FOR MASKED KEY BYTE; A, B PER C, G INSTRUCTION

Field	Index Increment†
A	Full increment (no match)
A	Partial increment (match)
B	No increment

† For a complete definition of index increment, refer to Index Increments at the beginning of the string instructions.

Field C serves as a mask such that the instruction makes a byte-by-byte comparison only when there are ones in the corresponding bit positions of the mask. Bits of the reference field and the key field are considered to match wherever there is a zero bit in the mask field. The mask field C is assumed to be as long as the key field B. There is no length specification for field C; the instruction represents field C as being at least as long as key field B. The mask field is associated with the key field such that on the second search, the instruction compares the first byte of B with the second byte of A, using the first byte of C as a mask. If the C designator is 00_{16}, the instruction generates a mask of all ones.

Figure 6-64 is an example of search for masked key, byte; A, B per C, G instruction with assumed instruction codes, register content, and data fields. Although the C designator specifies a particular register, the mask field is set to all ones. Thus, all bits are compared in the byte compare operations.

In figure 6-64, the solid arrows indicate the first complete search and the dashed arrows indicate the second complete search although a complete search does not actually take place in these cases. The third and subsequent searches follow the same pattern. The bytes in fields A and B are assumed to contain representations of alphabetical characters.

If no match is detected, the maximum number of complete searches equals the length of field A minus the length of field B plus one (A - B + 1), which in the example would be 6 - 3 + 1 = 4. In the example, a match is detected on the fourth and final search.

If any of the following conditions are present, the results of the instruction become undefined.

1. Any or all of the A, B, or X designators are 00_{16}.
2. The length of the A and/or B field is 00_{16}
3. The B field is longer than the A field.

For certain applications, it is desirable to allow a match in two strings of bytes in which there are no more than a specified number of compare differences. For example, if one difference is allowed, the key field (MINNEAPOLIS) would match the portion of the reference field (MINNZAPOLIS). The character Z represents the one allowed difference in the reference field for a match. The maximum number of allowed compare differences is termed the difference threshold count. This count is contained in the rightmost 16 bits of the register designated by G (figure 6-64). Only a positive, two's complement number is meaningful as a difference threshold count.

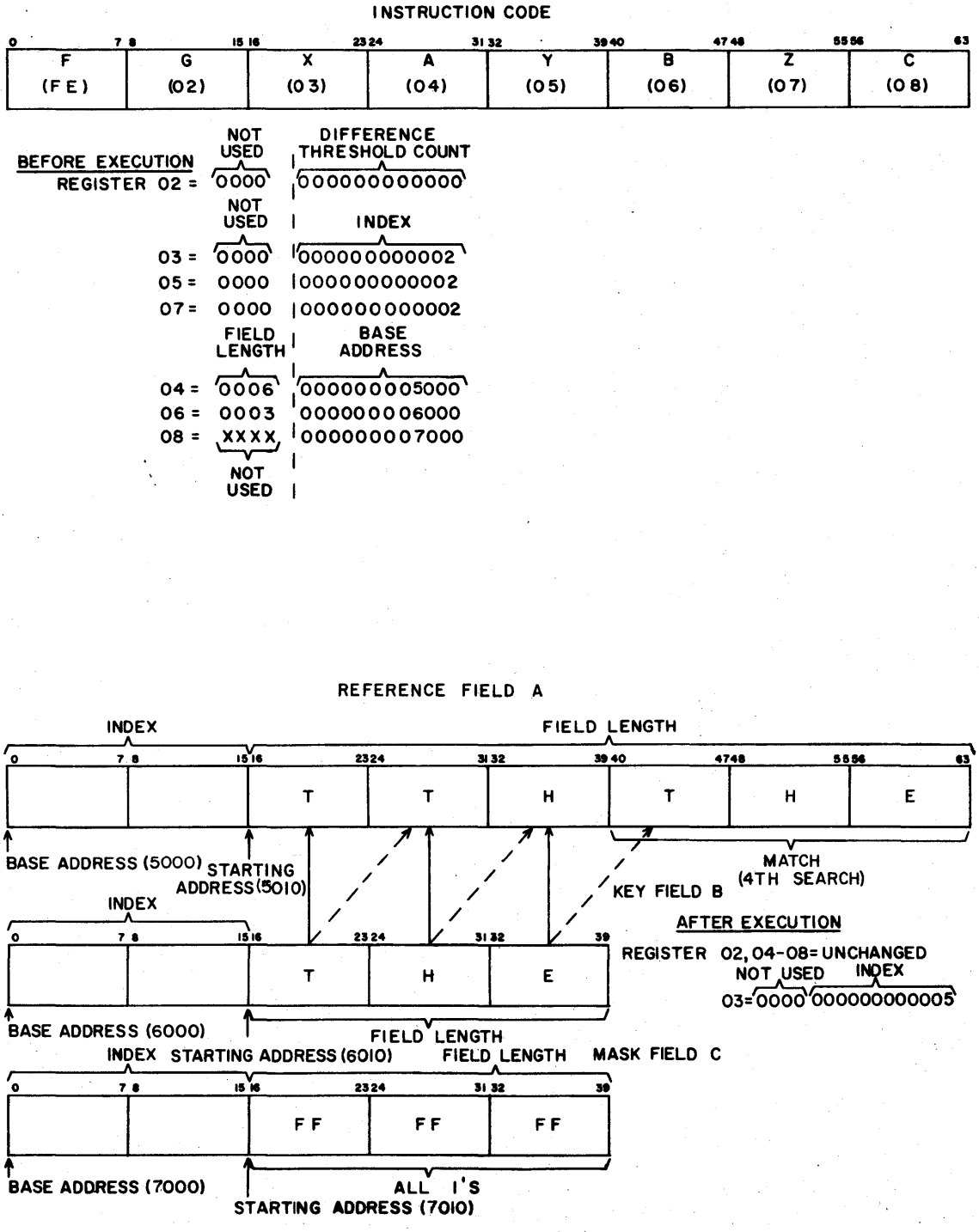


Figure 6-64. Example of Search for Masked Key Byte; A, B Per C, G Instruction

If the C designator is zero, the operation is identical to that with a mask of all ones. If the A and/or B designator is zero, if the length of field A and/or B is zero, or if the B field is longer than the A field, the results of this instruction are undefined.

The difference threshold count indicates the number of allowed differences on any one search. In figure 6-64, the instruction compares the character T in the key field with the same character in the reference field in the first search. Since there is no difference in this comparison, the instruction compares the character H in the key field with the character T in the reference field, and a difference occurs. Thus, the first search terminates. The instruction would then initiate the second search which would not detect a difference until the third byte comparison (E in the key field is compared with T in the reference field). The instruction initiates successive searches until it detects a match which, in the example, occurs on the fourth search.

If the difference threshold count is set to one, the instruction allows one difference on any one search, if the difference threshold is set to two, the instruction allows two differences, etc. In the example, a difference threshold of one gives a match on the second search, and a threshold of two gives a match on the first search.

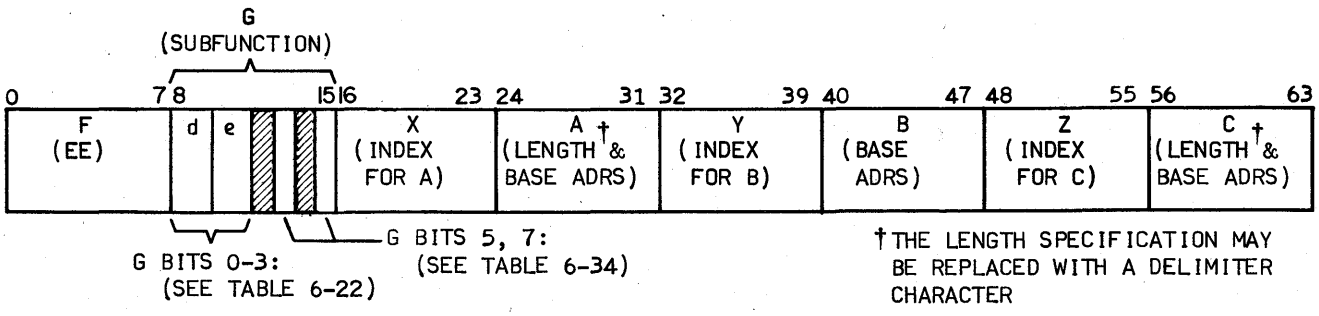
FF SEARCH FOR MASKED KEY, WORD; A, B PER C, G

This instruction is identical in operation to the search for masked key, bytes (FE) instruction, except masked words are compared rather than bytes. The length specifications and indexes are expressed in words instead of bytes. The instruction compares masked full words. The only possible matches take place at word boundaries and the instruction initiates new searches at word boundaries. As in the FE instruction, the FF instruction sets data flag bit 37 if no match is found.

D6 SEARCH FOR MASKED KEY, BIT; A, B PER C, G

This instruction is identical in operation to the search for masked key, bytes (FE) instruction, except masked bits are compared rather than bytes. The length specifications and indexes are expressed in bits rather than bytes; the instruction compares masked bits. The only possible matches take place at bit boundaries and the instruction initiates new searches at bit boundaries. As in the FE instruction, the D6 instruction sets data flag bit 37 if no match is found.

EE TRANSLATE A PER B→C



This instruction translates the bytes (from left to right) in field A. A translate table, which is stored in field B, controls the translation. The instruction stores the translated bytes in result field C.

The bytes read from field A serve as item counts. The instruction first shifts each item count left three places and then adds it, after indexing, to the starting address of the B field to form a new address. The byte at this new B field address is transmitted to a position in the C field that corresponds to the item count that produced the shift. Thus, the C field contains the translated bytes. Field B is not incremented although the Y designator specifies an index in bytes for the B field.

No field length or delimiter may be specified for the B field (translate table). However, the effective length of the table cannot exceed 256_{10} bytes, because a byte (8 bits) is used to index the translate table. The computer loads the entire translate table into a buffer memory at the beginning of the instruction execution. If this table crosses a page boundary (but the portion actually used by the programmer is contained in the first page), it is possible for the computer to generate an unnecessary access interrupt while loading what will become the unused portion of the table.

When field C is length-limited and field A is exhausted before field C is exhausted, field C is filled out with null (00_{16}) bytes. If fields A and C are length-limited and field C is filled before A is exhausted, the index associated with the A field will be incremented by the A length rather than the actual munber of bytes translated.

If field C is length-limited, field A is delimiter-limited, and field C is exhausted before field A, then field A is searched for its delimiter character so its index may be properly incremented. When field C is delimiter-limited, the instruction proceeds until field A is exhausted. The delimiter for field C is then stored immediately following the last translated byte which was stored. The delimiter is stored even if the A field length is initially zero.

Index incrementation takes place for the A and C fields as specified by bits 5 and 7 of the G designator (table 6-36); the B field index is not incremented.

TABLE 6-36. INDEX INCREMENTS FOR TRANSLATE A PER B → C INSTRUCTION

Field	G Bit 5	G Bit 7	Index Increment †
A	0	-	Full increment
A	1	-	No increment
B	-	-	No increment
C	-	0	Full increment
C	-	1	No increment

† For a complete definition of index increment, refer to Index Increment at the beginning of the string instructions.

Figure 6-65 is an example of a translate A per B→ C (EE) instruction with assumed instruction codes, register contents, and A and B fields. The example uses a delimiter character for the A field and a length specification for the C field. G designator bits 5 and 7 are zeros. Thus, both the A and C index are incremented.

In the example, each byte in the A field represents a digit of a decimal number. The consecutive bytes of the translate table in the B field contain the translation code for the corresponding digits. The example translates the digits in the A field into translated characters and transmits them to consecutive bytes of the C field. For example, the digit 3 is shifted left three places and is added to the starting address of the B field:

$$\begin{array}{llllr} 0110 & 0000 & 0000 & 1000 & (6008_{16}) \\ 0000 & 0000 & 0001 & 1000 & (0018_{16}) \\ \hline 0110 & 0000 & 0010 & 0000 & (2020_{16}) \end{array}$$

Thus, 2020_{16} becomes the address in the B field of the translation for character 3 in the A field. This translation is then transmitted to the leftmost byte of the C field. This process continues until the C field is filled. The A index (register 02) is incremented by seven. The C index is incremented by six.

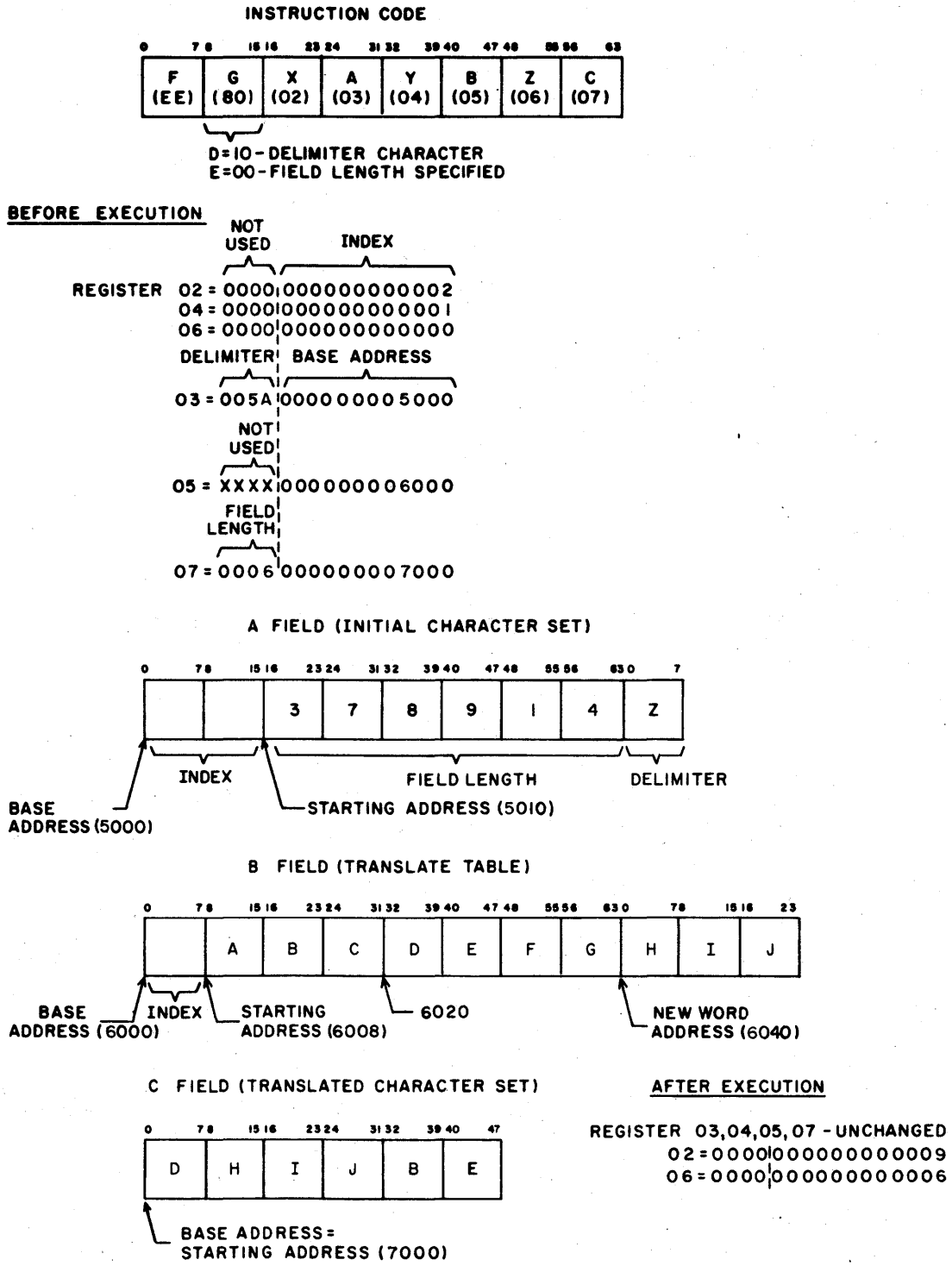
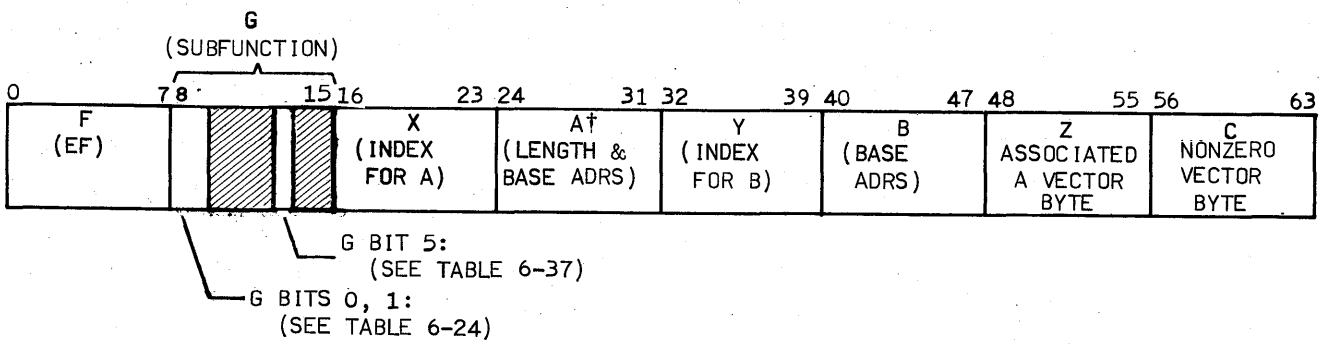


Figure 6-65. Example of Translate A Per B →C Instruction

EF TRANSLATE AND TEST A PER B→C



This instruction translates the bytes (from left to right) in field A. A translate table, which is stored in field B, controls the translation.

The bytes read from field A serve as item counts. The instruction first shifts each item count left three places and then adds it, after indexing, to the starting address of the B field to form a new address. The new address references a byte in the translate table (B field). If the byte in the translate table is zero, the next byte to the right of the one referenced in the A field is referenced and translated. This process continues until the instruction reads a nonzero byte from the translate table or exhausts the A field.

No field length or delimiter may be specified for the B field (translate table). However, the effective length of the table cannot exceed 256_{10} bytes, because a byte (8 bits) is used to index into the translate table. The computer loads the entire translate table into a buffer memory at the beginning of the instruction execution. If this table crosses a page boundary (but the portion actually used by the programmer is contained in the first page), it is possible for the computer to generate an unnecessary access interrupt while loading what will become the unused portion of the table.

If the A field is delimiter limited, the delimiter character is not translated. When a nonzero translated byte is found, it is stored in register C and the associated byte from field A is stored in register Z. The bytes are stored in the rightmost 8 bits, and the leftmost 56 bits in these two registers are cleared. If no nonzero translated byte is found, registers C and Z are not altered. The X (when G bit 5 = 0), C, and Z register results are undefined if the C and Z designators are equal in this instruction.

The instruction terminates if a nonzero byte is referenced from the translate table or if field A is exhausted, whichever occurs first. The instruction increments the A index according to whether a nonzero byte is referenced as specified by G designator bit 5 (table 6-37). Field B is not incremented, although the Y designator specifies an index in bytes for the B field.

† The length specification may be replaced with a delimiter character.

TABLE 6-37. INDEX INCREMENTS FOR TRANSLATE AND TEST A PER B→C INSTRUCTION

Field	G Bit 5	Index Increment †
A	0	Partial increment (nonzero byte)
A	0	Full increment (all bytes zero)
A	1	No increment
B	-	No increment

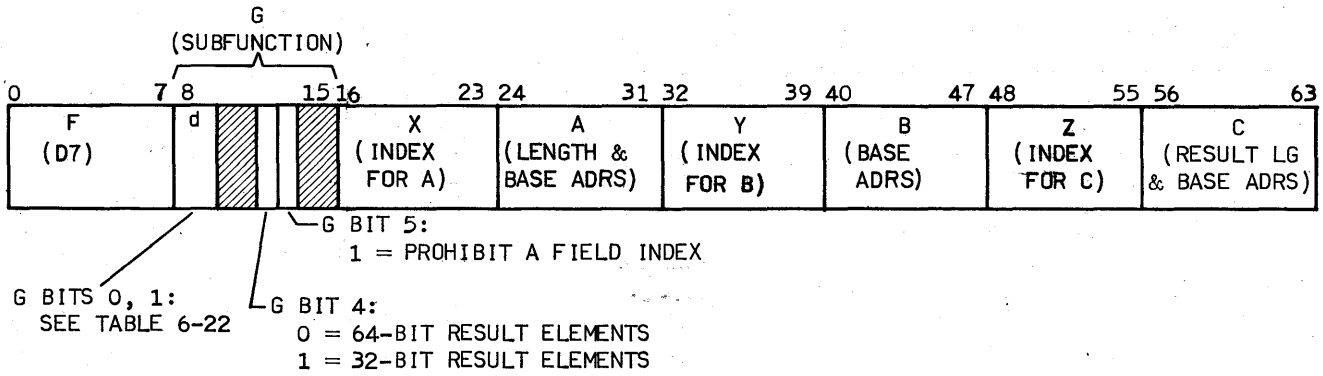
† For a complete definition of index increment, refer to Index Increments at the beginning of the string instructions.

The instruction sets the data flag bits according to the results of the instruction (table 6-38).

TABLE 6-38. DFB CONDITIONS FOR THE EF INSTRUCTION

DFB Bit	Condition
53	Termination due to length or delimiter rather than nonzero translated byte
54	Termination due to nonzero translated byte which is not the last data byte in the A field
55	Termination due to nonzero translated byte which is the last data byte in the A field

D7 TRANSLATE AND MARK A PER B→C



This instruction translates the bytes (from left to right) in field A. A previously stored translate table in the B field controls the translation.

The bytes read from field A are item counts. The instruction first shifts each item count three places and then adds it to the starting address of the B field to form a new address. The new address references a byte in the translate table (B field). If the byte in the translate table is zero, the next byte to the right of the one referenced in the A field is referenced and translated. This process continues until the instruction reads a nonzero byte from the translate table or exhausts the A field. Field B is not incremented although the Y designator specifies an index in bytes for the B field.

No field length or delimiter may be specified for the B field (translate table). However, the effective length of the table cannot exceed 256_{10} bytes, because a byte (8 bits) is used to index into the translate table. The computer loads the entire translate table into a buffer memory at the beginning of the instruction execution. If this table crosses a page boundary (but the portion actually used by the programmer is contained in the first page), it is possible for the computer to generate an unnecessary access interrupt while loading what will become the unused portion of the table.

If the A field is delimiter limited, the delimiter character will not be translated. When a nonzero translated byte is found, it is stored (right justified) in the cleared exponent portion of result vector C. The partially incremented A field index is stored (right justified) in the cleared coefficient portion of result vector C. The translate then continues with every nonzero translated byte and its associated index being stored in vector C until the A field is exhausted.

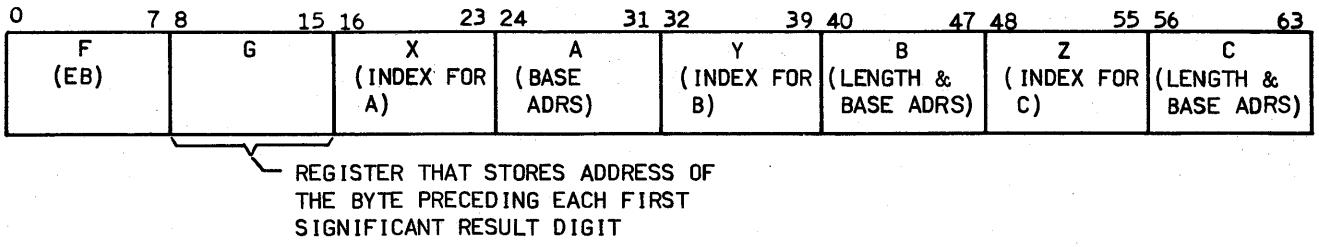
When the A field is exhausted, the operation enters the number of nonzero translated bytes into the field length portion of the register designated by C and terminates the instruction. When the number of nonzero translated bytes exceeds $2^{16}-1$, the instruction sets data flag bit 37 (select condition not met). The instruction makes no further indication if the count exceeds $2^{16}-1$ more than once. If all the translated bytes are zero, data flag bit 53 is set.

If G bit 4 is cleared, register Z specifies a word index for the result vector C which consists of 64-bit elements. If G bit 4 is set, register Z specifies a half-word index for the result vector C which consists of 32-bit elements. In forming the 32-bit element, the rightmost 24 bits of the partially incremented A field index are stored in bits 8 through 31 of each element. The leftmost 24 bits of that A field index are ignored for this case.

If G bit 5 is not used, the instruction full indexes the A field index. If G bit 5 is a one, the A field is prohibited from being indexed. G bit 5 controls only the updating of the A field index at the termination of the instruction. Thus, if G bit 5 is not set, the A field index retains appropriate updated index of the translated bytes.

The B and C field indexes are not incremented; the C field is in half words.

EB EDIT AND MARK A PER B➔C



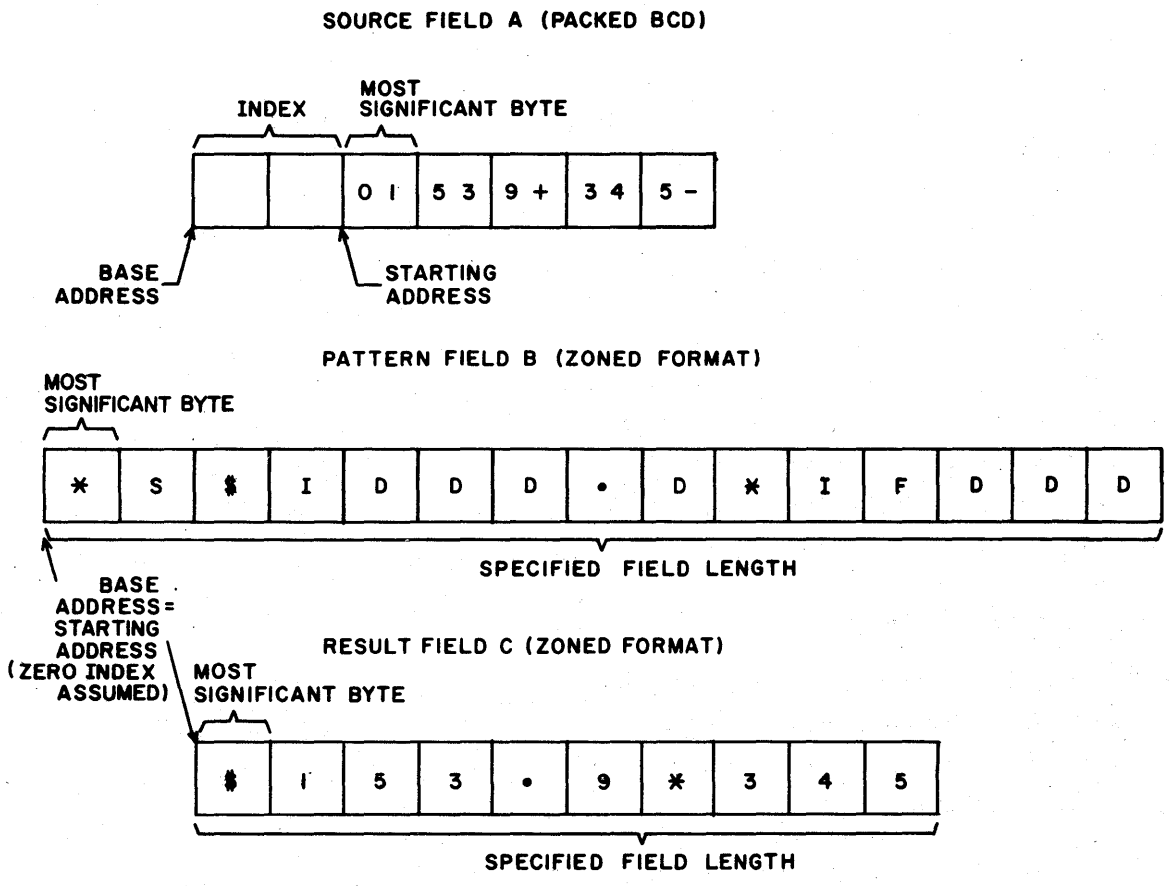
This instruction edits field A under the control of pattern field B and stores the result in field C. The editing operation proceeds from left to right. Figure 6-66 shows the general format of the fields for the EB instruction. Source field A is in packed BCD format while pattern field B and result field C are in the zoned format.

Each of the characters in the pattern field and result field are contained in one 8-bit byte. The bytes are processed from left to right. The instruction examines the pattern characters in conjunction with the corresponding source digits and determines the result characters. The definitions and zoned codes for the pattern characters are listed in figure 6-66. Subsequent paragraphs describe the pattern characters in more detail.

The field length specifications for fields B and C are item counts in bytes. The field length specification for field A is not used. Delimiter characters are not allowed in this instruction. This instruction permits the editing of multiple source fields with the use of a field separation character in the pattern field. As shown in the instruction format, the instruction stores the address of the byte preceding each first significant result digit of field C in the register designated by G.

The instruction determines the character to be stored in the result field by an examination of the pattern character and then, if necessary, the state of that T flip-flop and/or the digit read from the source field. The instruction stores characters in the result field according to one of the following conditions.

1. The source digit (A field) is expanded to zoned format and transmitted to the result field.
2. The pattern character is transmitted to the result field.
3. The fill character is transmitted to the result field.



NOTE:
IN THIS EXAMPLE AN ASCII SELECTION IS ASSUMED.

PATTERN FIELD CHARACTER DEFINITIONS

S = SIGNIFICANT START CHARACTER (1110 0001)

D = DIGIT SELECT CHARACTER (1110 0000)

*• MESSAGE INSERTION CHARACTER

I = IGNORE-DIGIT CHARACTER (1110 0011)

F = FIELD SEPARATION CHARACTER (1110 0010)

Figure 6-66. Example of Field Formats for Edit and Mark A Per B →C Instruction

T FLIP-FLOP

The T flip-flop controls the placement of source digits, fill characters, and pattern characters in the result field. Initially, the instruction clears the T flip-flop. Subsequently, pattern characters and source digits direct the setting and clearing of the T flip-flop.

The detection of a plus sign in the proper position of the source field clears the T flip-flop although it was previously set by a nonzero digit in the same source byte. This operation is described further in Source Digits.

The address of the byte that precedes each first significant result digit stored in output field C is recorded in the rightmost 48-bit register designated by G. The leftmost 16 bits are cleared to zero. The first significant result digit is defined as the first digit stored following a significance start character before a field separator. The first significant result digit may also result from a digit being stored as the result of a digit-select character when the T flip-flop is cleared and the digit is nonzero. This condition may occur several times during the execution of a single EB instruction. If no first significant result digit is stored, the contents of the G register is not altered.

PATTERN CHARACTERS

Any 8-bit byte may appear in the pattern field. The instruction interprets all bytes as message insertion characters except for the four special pattern characters. The four pattern characters with special significance are the digit-select, significance-start, field-separation, and ignore-digit characters.

Table 6-39 lists each of the pattern characters and the conditions in which they function.

DIGIT-SELECT CHARACTER

This character causes either a source digit or the fill character to be transmitted to the result field.

SIGNIFICANCE-START CHARACTER

This character sets the T flip-flop which permits only source digits to be transmitted for digit select characters until the occurrence of a field separator. Nothing is transmitted to the result field.

FIELD-SEPARATION CHARACTER

This character identifies individual fields in a multiple source-field operation. When the instruction detects a field-separation character, it clears the T flip-flop and nothing is transmitted to the result field.

TABLE 6-39. PATTERN SELECT CHARACTERS

Character	Code	Examine Digit	Initial State of T Flip-Flop	Source Digit Status	Result Character	Resulting State of T Flip-Flop
Digit-select	XX10 0000	Yes	t = 1		Digit	t = 1
			t = 0	d ≠ 0	Digit	t = 1
			t = 0	d = 0	Fill	t = 0
Significance-start	XX10 0001	No	t = 1		None	t = 1
			t = 0		None	t = 1
Field-separation	XX10 0010	No	t = 0 or 1	-	None	t = 0
Ignore-digit	XX10 0011	Yes	t = 0	-	None	t = 0
			t = 1	-	None	t = 1
Message-insertion	Any other character	No	t = 1	-	Pattern	t = 1
			t = 0	-	Fill	t = 0

Symbols:

d - Represents a source digit.
t - T flip-flop (cleared by plus signs or field separation characters).
digit - The source digit is expanded to eight bits (zoned and is stored in the result field).
fill - The fill character is stored in the result field.
pattern - The pattern character is stored in the result field.
XX - 11 in ASCII mode
00 in EBCDIC mode

IGNORE-DIGIT CHARACTER

This character causes the next source digit to be skipped. The digit is not sampled for a zero/nonzero status and nothing is transmitted to the output field. Since the normal samples for sign codes are made, the ignore-digit character could result in a data fault or the clearing of the T flip-flops.

MESSAGE-INSERTION CHARACTERS

The instruction does not examine a source digit when it reads a message-insertion character from the pattern field. If the T flip-flop is a 1 at this time, the instruction transmits the message-insertion character to the result field. If the T flip-flop is a 0, the fill character is transmitted. The exception is if a message-insertion character appears as the first character of the pattern field (T=0), the message-insertion character defines the fill character for the instruction. No character is transmitted to the result field for this first pattern field character.

SOURCE DIGIT

When the instruction stores the source digit in the result field, it expands the source digit code from the packed BCD format to the zoned format by attaching a zone code as the leftmost four bits of the byte. The zone code is conditioned by the ASCII/EBCDIC bit in the job's invisible package. If ASCII is selected, the zone code is 0011. If EBCDIC is selected, the zone code is 1111.

Each byte in a source field contains two digits or a digit and a sign. When a byte contains a sign, the sign is in the rightmost four bits. The sign is processed in conjunction with the digit in the leftmost four bits of the same byte. A positive sign clears the T flip-flop. If a sign should be in the leftmost four bits of a source byte, data flag 38 (decimal data fault) sets.

The source digits are examined once during an editing operation. The instruction examines the leftmost four bits of the byte first. The rightmost four bits are then checked for sign. If these bits are not a sign, they are available for the next pattern character that calls for a digit examination.

If the instruction detects a data fault in the source field, the contents of the result field and data flag bits 53, 54, and 55 are undefined. The instruction also sets data flag bit 38 for this condition.

Any of the plus sign codes (1010, 1100, 1110, or 1111) clear the T flip-flop after the preceding digit is examined. The minus sign codes (1011 and 1101) do not affect the state of the T flip-flop. When one of the sign codes is encountered in the rightmost bits, the bits are no longer treated as a digit. In this case, the next digit to be examined is in the leftmost bits of the next character. Digits in the source field are only examined for digit-select and ignore-digit pattern characters.

FILL CHARACTER

The fill character is the first character of the pattern field unless that character is one of the four special pattern characters. When the instruction reads the fill character from the pattern field, it retains this character for later use. It is not transmitted to the first character position of the result field. The instruction continues with an examination of the second character in the pattern field to determine the first character in the result field.

If one of the four special pattern characters is the first character of the pattern field, the instruction uses the blank character (20_{16} for ASCII or 40_{16} for EBCDIC) as the fill character. The instruction continues with a reexamination of the first character of the pattern field to determine the first character of the result field.

When the instruction detects a field-separation character in the pattern field, the fill character is neither changed nor is a source digit examined. In this case, the instruction clears the T flip-flop and continues with an examination of the next pattern character.

DATA FLAG BITS

The EB instruction uses data flag bits to indicate the sign and zero status of the last source field edited. The state of the data flag bits pertains to fields specified by the field-separation characters, regardless of the number of signs contained within the field. For multiple field editing operations, the data flag bits indicate only the field following the last field-separation character. Thus, when the last character of the pattern field is a field-separation character, the data flag bits indicate an all zero field. Figure 6-66 shows that the last source field contains a negative sign code; thus, data flag bit 54 is set.

The instruction examines all source digits in a field for the zero code (0000_2) because of a digit-select. At the termination of the instruction, the data flag bits indicate whether the field edited after the last field-separation character contained all zero digits. When the last edited field contains all zero digits, the instruction sets data flag bit 53. If the T flip-flop is cleared and the last edited field contains at least one nonzero digit, the instruction sets data flag bit 55. Table 6-40 lists the data flag bits affected by the EB instruction and the corresponding conditions under which they are set.

TABLE 6-40. DFB CONDITIONS FOR THE EB INSTRUCTION

DFB Bit	Condition
38	Decimal data fault
53	Last edited field is zero
54	Last edited field nonzero with negative sign or unsigned (T flip-flop set)
55	Last edited field nonzero with positive sign (T flip-flop clear)

DATA FAULT

The instruction sets data fault flag bit 38 whenever the operation encounters a sign code in the leftmost four bits of a byte in the A field. The flag is also set whenever more than one numeric field is encountered by a single pattern field (that is, between the start of the pattern field and the first field-separator or between any two field-separator characters in the pattern). This condition occurs when a digit-select or ignore-digit character is detected in the pattern field after a sign code was examined and before a field-separator. If a data fault occurs, the contents of the output field C is undefined, data flags 53, 54, and 55 are undefined, and data flag 38 is set.

TERMINATION

The instruction terminates by filling result field C or by attempting to read beyond pattern field B. At termination, the instruction sets the data flag bits as listed in table 6-40.

EXAMPLES

In the following examples, the character codes are used as defined. In each case, the starting address for the result field C is 40000_{16}, and the initial content of register G is 10000_{16}. For purposes of clarity, all field indexes are assumed to be zero. All field lengths are assumed to equal the lengths actually shown in the examples. In the source fields, the BCD digits are shown in their normal decimal notation in the corresponding byte positions. The pattern and result fields are shown marked off in bytes with digit or symbolic representation of the character in each byte position. No bit, word, or byte addresses are shown for the fields and all fields are to be processed left to right. In the examples, solid lines (with arrows) show the actual transfer of a character or digit to the result field, while a dashed line indicates the pattern character that controlled the transfer of a digit from the source field or the fill character.

The following symbols are defined for use in the examples.

Symbol	Definition
B	Blank character
D	Digit-select character
S	Significance-start character
F	Field-separation character
I	Ignore-digit character

EXAMPLE 1

Figure 6-67 shows an example of an edit/mark A per B → C instruction with a single source field containing a positive sign. Table 6-41 lists the step-by-step operation of the instruction for example 1.

Figure 6-67 shows the retaining of the fill character (*) and its transfer to the corresponding byte positions in the result field. The final content of G represents the address of the byte preceding the first significant result digit (3) stored in output field C. This address occupies the rightmost 48 bits of G; the leftmost 16 bits of G are cleared to zero.

TABLE 6-41. OPERATION OF EDIT AND MARK A PER B → C INSTRUCTION

Pattern Character	Source Digit	T Flip-Flop State	Conditions for Result Field
*		0	This character is retained as the fill character.
D	0	0	Fill character (*)
D	0	0	Fill character (*)
,		0	Fill character (*)
D	3	1	Digit (3) - First nonzero digit sets the T flip-flop.
D	6	1	Digit (6)
D	3	1	Digit (3)
S		1	Significance-start character would have set T flip-flop if not already set.
.		1	Pattern (.)
D	2	1	Digit (2)
D	9	1	Digit (9)
	+	0	No output to result field. Plus sign clears T flip-flop.
B		0	Fill character (*)
C		0	Fill character (*)
R		0	Fill character (*)

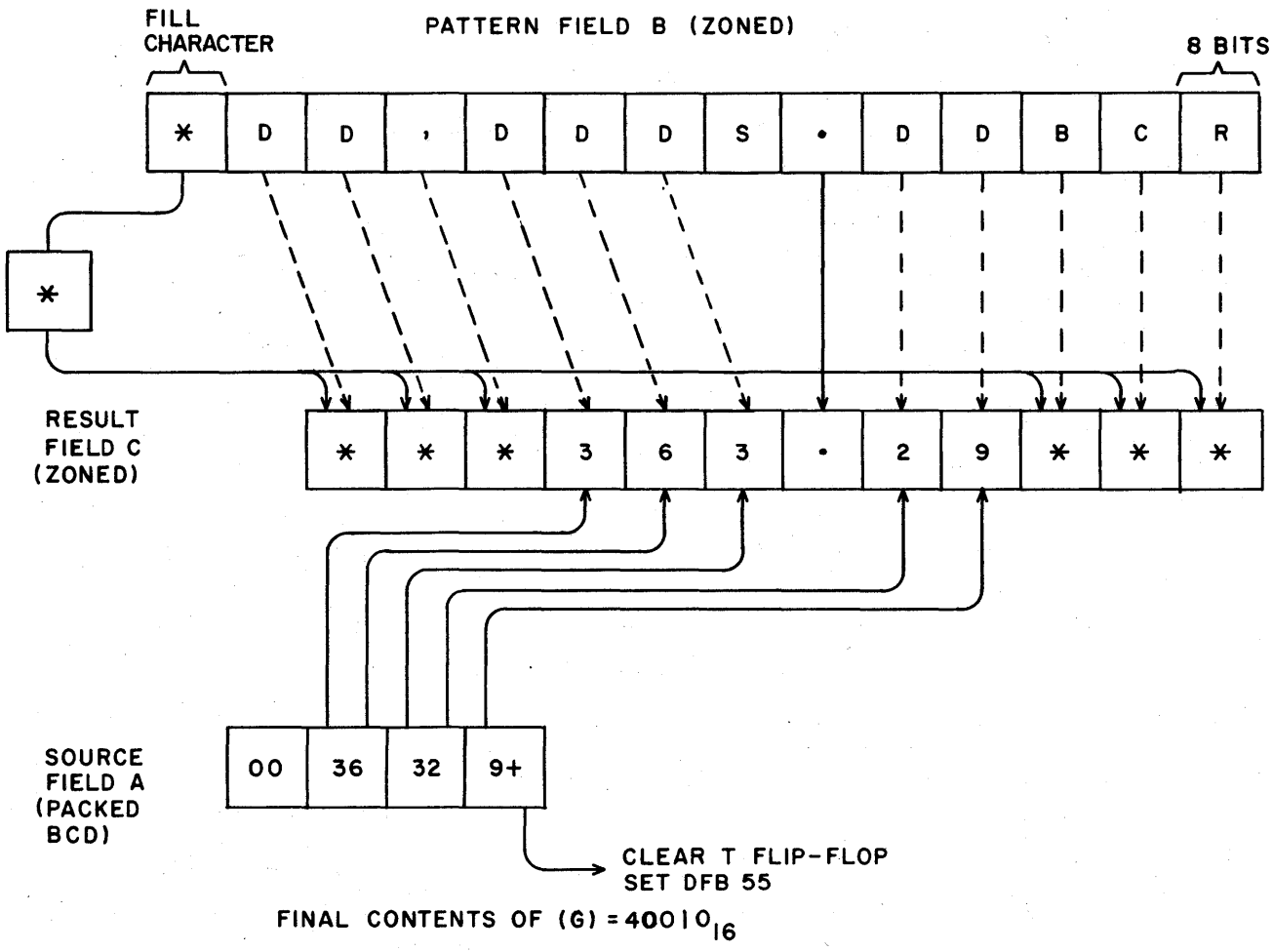


Figure 6-67. Example 1 of Edit and Mark A Per B → C Instruction (Single Source Field, Sign +)

EXAMPLE 2

Example 2 (figure 6-68) shows a pattern field identical to the pattern field shown in Figure 6-67. However, in example 2, only one fill character is used in the result field since the source field contains only one leading zero digit. As a result, the T flip-flop is set by the first significant digit (1).

Example 2 shows that the T flip-flop remains set due to the negative sign in the source field. Thus, the instruction transfers pattern characters blank (B), C, and R to the result field instead of the fill characters that are transferred in example 1.

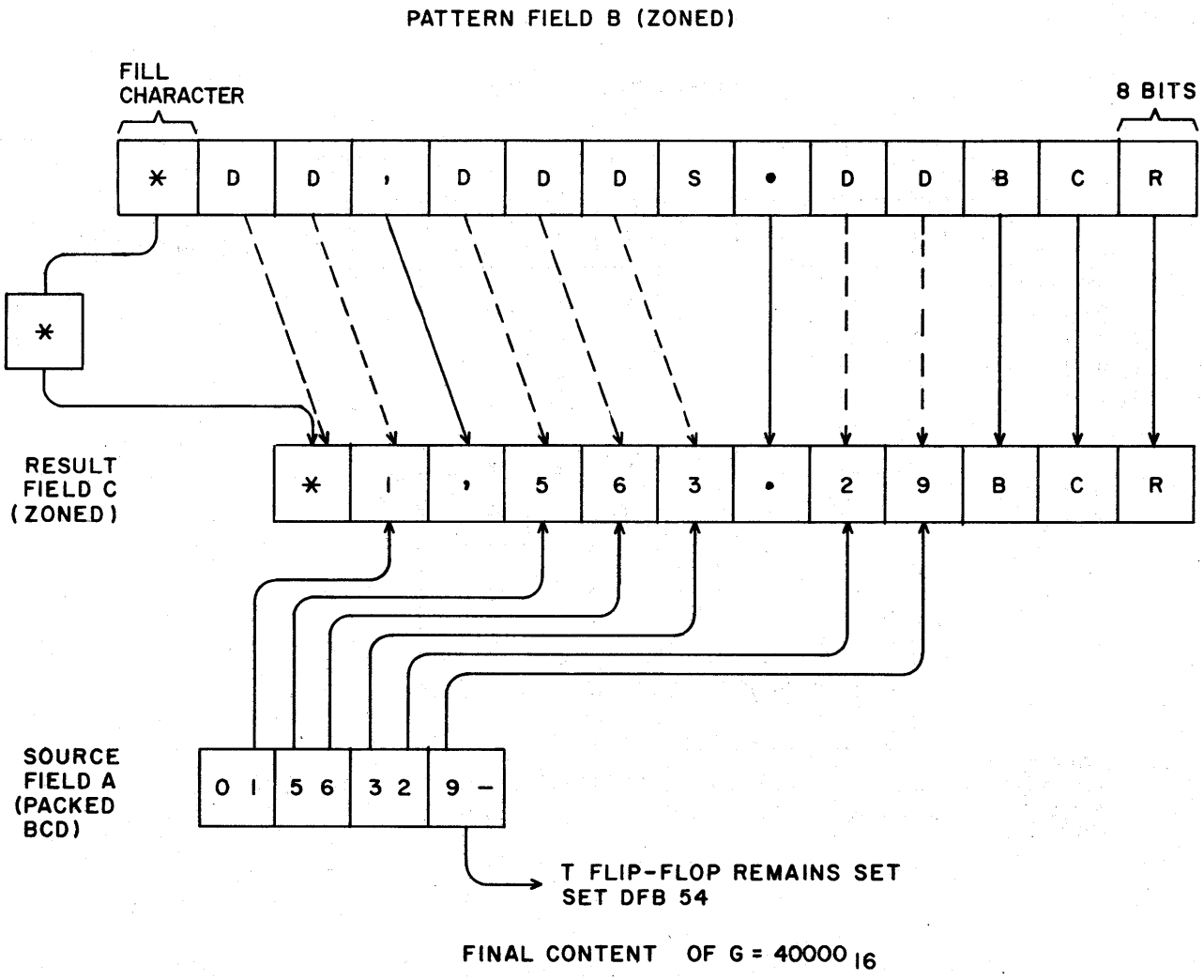


Figure 6-68. Example 2 of Edit and Mark A Per B → C Instruction (Single Source Field, Sign -)

EXAMPLE 3

The first character of the pattern field in example 3 (figure 6-69) is significance-start character (S).

As a result, the blank character (B) is retained as the fill character. The pattern field also contains a field-separation character in the last byte of the field. Since there is no second BCD field in field A (following the sign), the T flip-flop is cleared and a blank character is stored as the last character in the result field. Thus, the instruction sets data flag bit 53, indicating that the last field edited contained all zero digits.

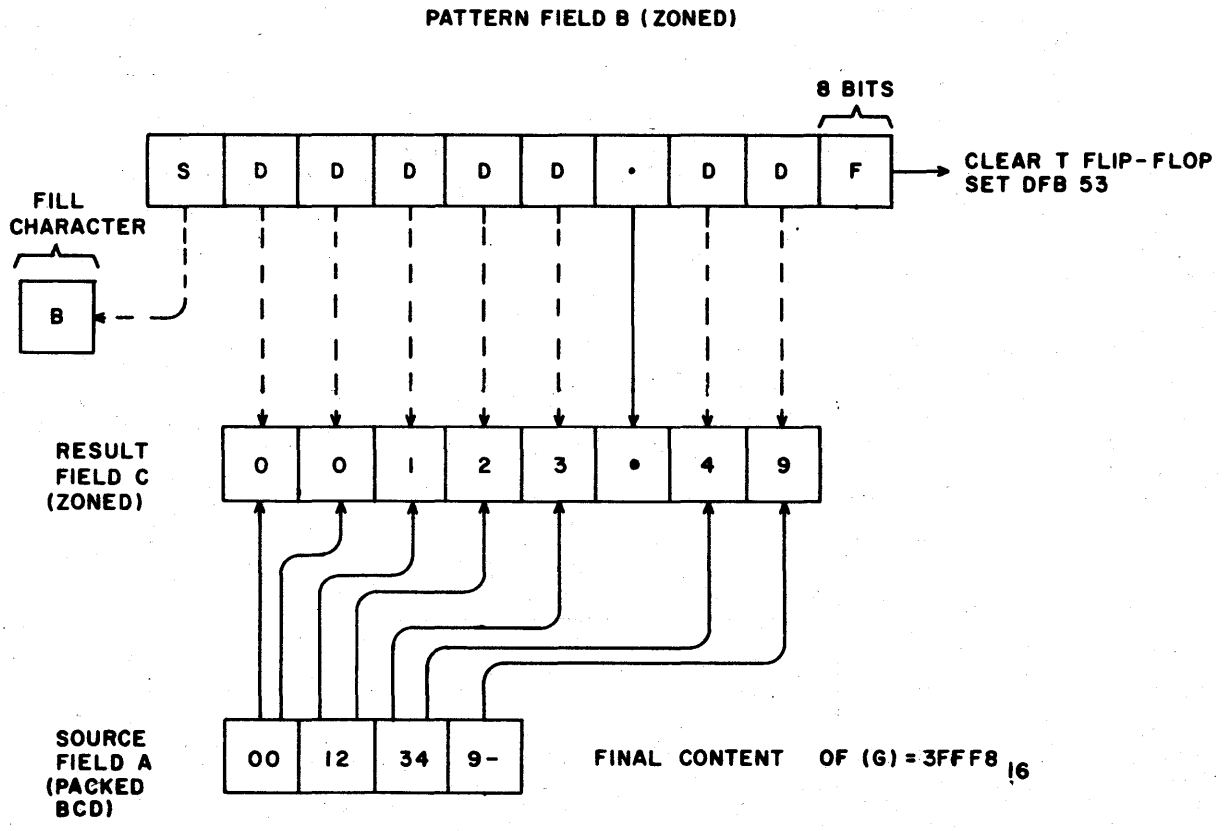


Figure 6-69. Example 3 of Edit and Mark A Per B → C Instruction (Field Separator Specified, No Second Field)

EXAMPLE 4

Example 4 (figure 6-70) shows a multiple source field editing operation. The first field is edited in the usual manner with the fill character (*) being retained. When the plus sign of the first field is detected, the instruction clears the T flip-flop. Thus, the fill character is inserted in the bytes of the result field corresponding to the two blank characters (B) and the two digit-select (D) characters which correspond to the two leading zero digits in the second source field. The detection of the first nonzero digit in the second source field sets the T flip-flop. The T flip-flop remains set since the sign of the second source field is negative. As a result, the instruction sets data flag bit 54 and transmits the byte address (40048_{16}) to register G.

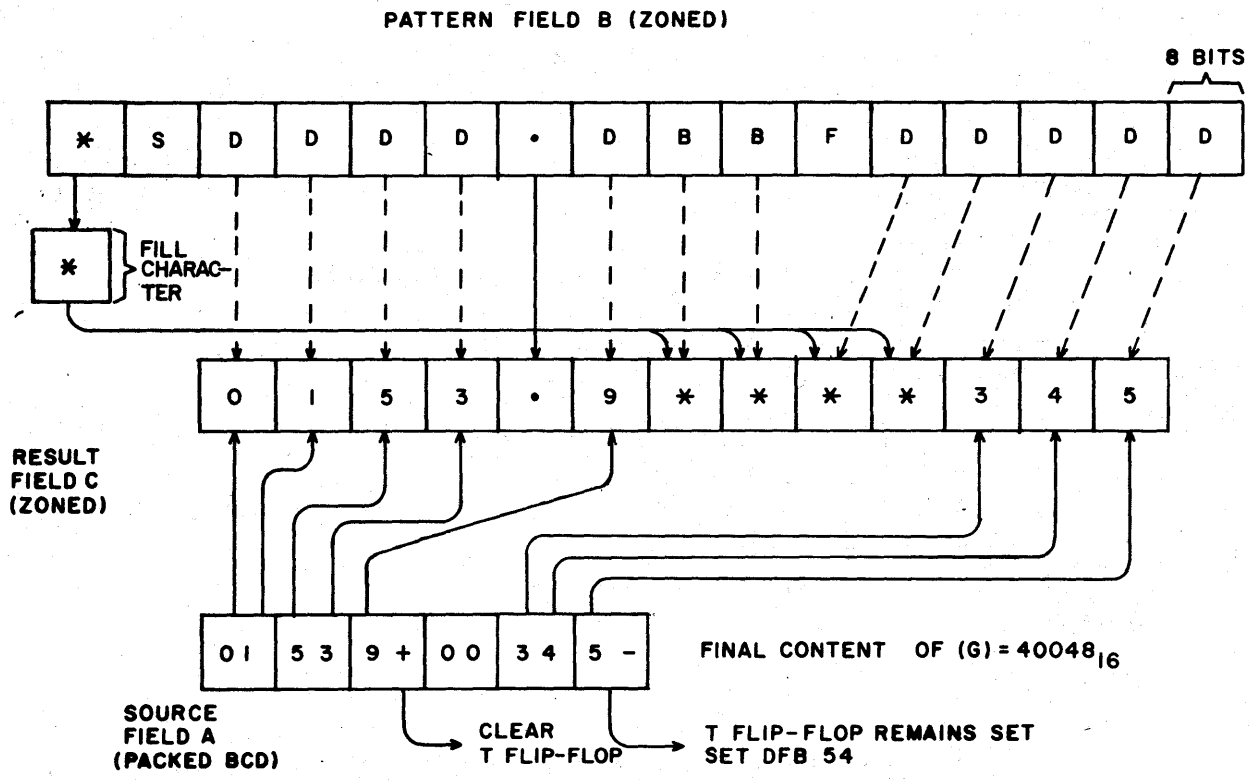


Figure 6-70. Example 4 of Edit and Mark A Per B → C Instruction (Multiple Field Editing)

EXAMPLE 5

Example 5 (figure 6-71) shows the results of a result field termination before the pattern field. The pattern and source fields in example 5 are identical to the corresponding fields in example 4. However, in example 5 the result field is three bytes shorter than the pattern field. As a result, the last three characters of the pattern and source fields are not examined. Since no significant characters of the second source field are examined in this case, the T flip-flop remains cleared, and the instruction sets data flag bit 53, indicating that the second source field contains all zero digits.

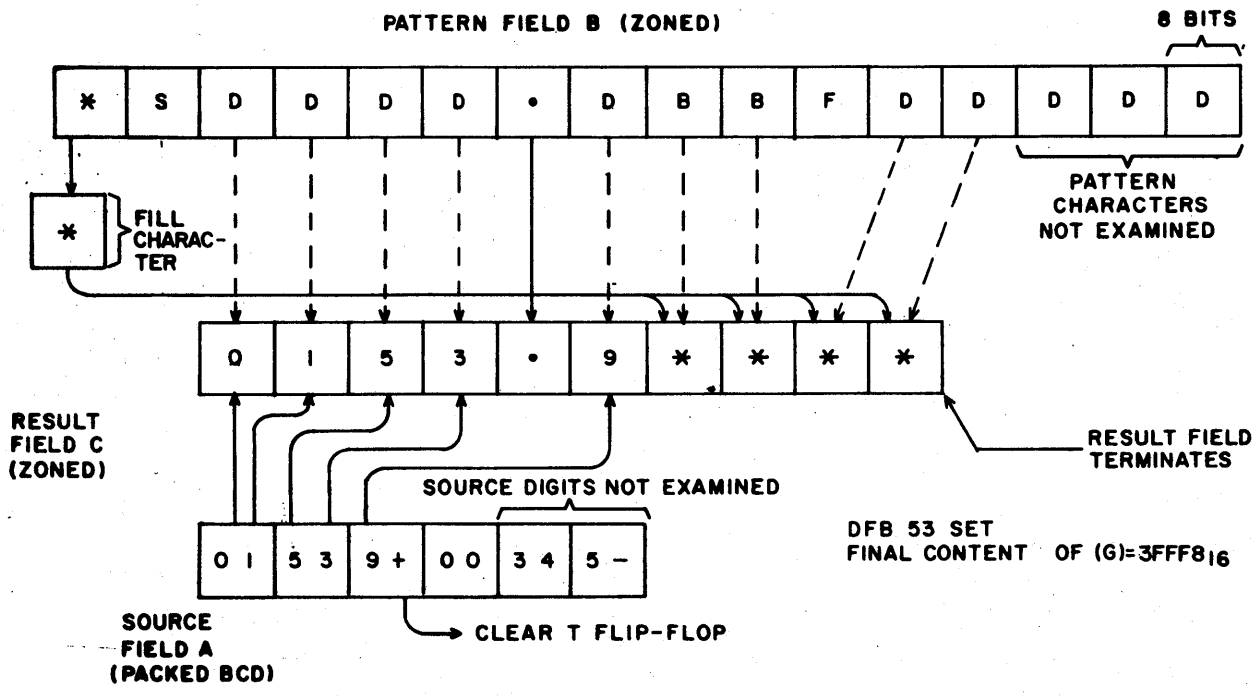


Figure 6-71. Example 5 of Edit and Mark A Per B → C Instruction (Result Field Shorter than Pattern Field)

EXAMPLE 6

Example 6 (figure 6-72) shows a source field with the sign character in the wrong position of the last byte. Thus, the contents of the result field and the contents of register G become undefined. In addition, the instruction sets data flag bit 38 (decimal data fault).

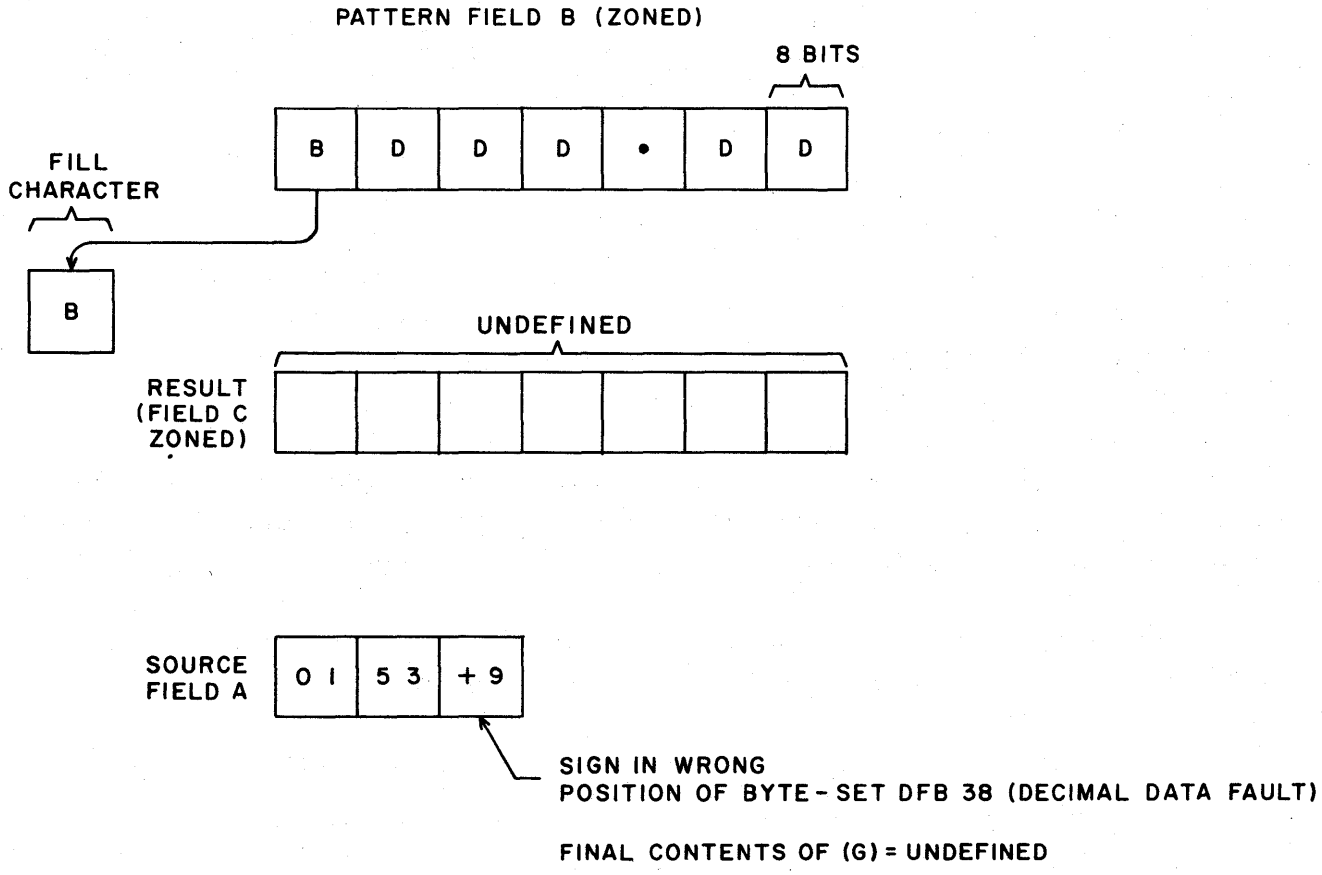
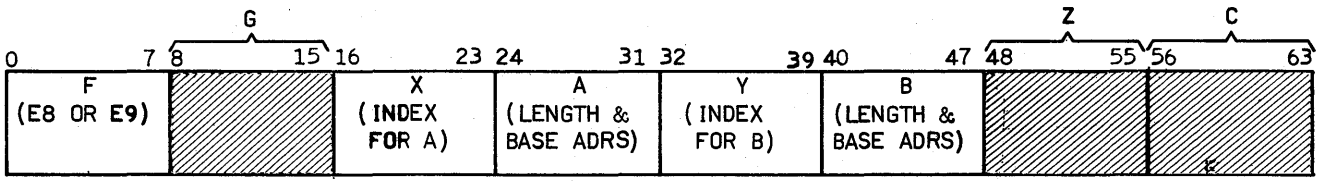


Figure 6-72. Example 6 of Edit and Mark A Per B → C Instruction (Decimal Data Fault, Undefined Results)

E8 COMPARE BINARY A, B

E9 COMPARE DECIMAL A, B



These two instructions compare source fields A and B for inequality. The comparison is from right to left. If the A and/or B designator is zero or if the length of one or both of the source fields is zero, the instruction generates a corresponding source field containing positive zero.

In the E8 instruction, source fields A and B contain two's complement, signed numbers (figure 6-73). If the source fields are unequal in length, the shorter of the two fields is extended with sign bits to equal the length of the other field.

At the termination of the E8 instruction, data flag bits 53, 54, and 55 are set according to the results of the compare operation as listed in table 6-42.

TABLE 6-42. DFB CONDITIONS FOR E8 AND E9 INSTRUCTIONS

DFB Bit	Condition
53	Equal operands (A = B)
54	Operand A is high
55	Operand A is low

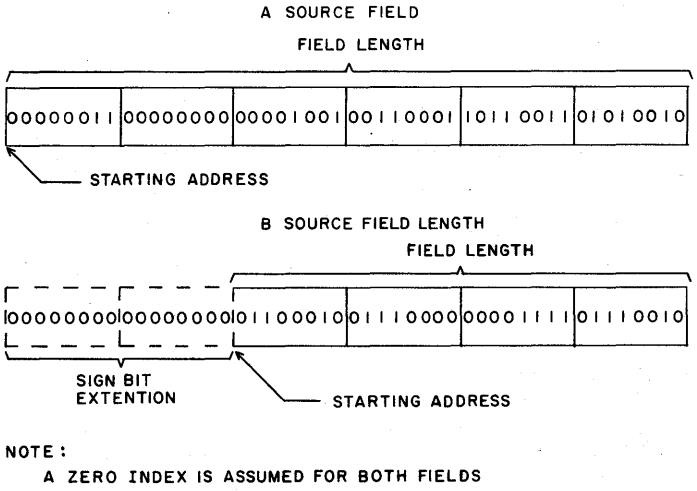


Figure 6-73. Example of Field Formats for the Compare Binary A, B Instruction

In the E9 instruction, source fields A and B contain packed BCD numbers (figure 6-74). If the two source fields are unequal in length, the shorter field is extended with zero digits to equal the other field. The E9 instruction compares the numbers from right to left and makes the comparison on the signed magnitudes of the two fields. Applicable data flag bits are 38 (decimal data fault), 53, 54, and 55 (table 6-41).

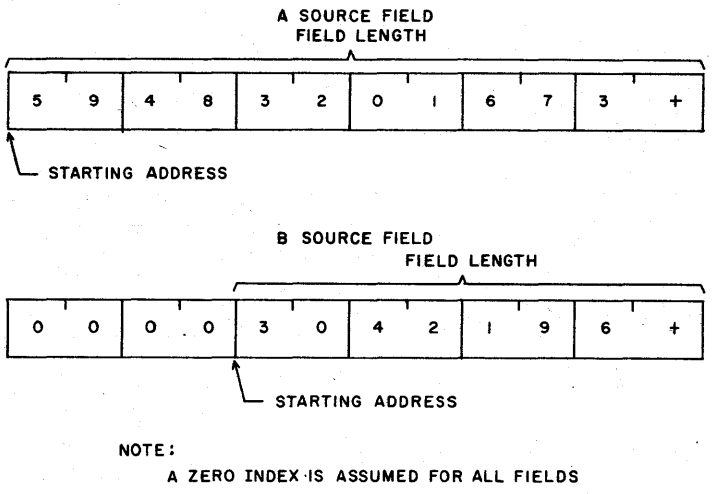
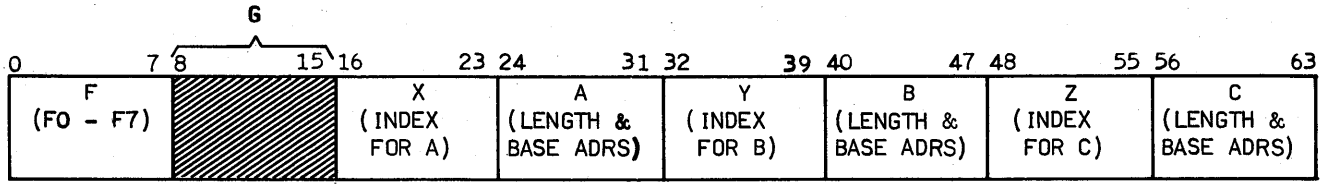


Figure 6-74. Example of Field Formats for the Compare Decimal A, B Instruction

LOGICAL STRING INSTRUCTIONS

The logical string instructions function in the same general manner as corresponding string instructions. Logical string instructions operate with indexes and data fields identical to those of the string instructions except that the item counts and indexes are expressed in bits instead of bytes. Thus, the logical string instructions perform bit operations on bit boundaries while string instructions perform byte operations on byte boundaries.

F0 LOGICAL EXCLUSIVE OR A, B ➔C
F1 LOGICAL AND A, B➔C
F2 LOGICAL INCLUSIVE OR A, B➔C
F3 LOGICAL STROKE A, B➔C
F4 LOGICAL PIERCE A, B ➔C
F5 LOGICAL IMPLICATION A, B ➔C
F6 LOGICAL INHIBIT A, B➔C
F7 LOGICAL EQUIVALENCE A, B➔C



These instructions perform bit-by-bit logical functions on binary source fields A and B and store the results in binary field C. Table 6-43 lists the variations of source bits A and B with the corresponding result bit for each of the logical string instructions.

TABLE 6-43. TRUTH TABLE FOR LOGICAL STRING INSTRUCTIONS

Source Bits		OR	AND	Exclusive OR	Stroke	Pierce	Implication	Inhibit	Equivalence
A	B	(A+B)	(A•B)	(A-B)	$\overline{(A \cdot B)}$	$\overline{(A+B)}$	$(A+\overline{B})$	$(A \cdot \overline{B})$	$\overline{(A-B)}$
0	0	0	0	0	1	1	1	0	1
0	1	1	0	1	1	0	0	0	0
1	0	1	0	1	1	0	1	1	0
1	1	1	1	0	0	0	1	0	1

Fields A, B, and C are strings of bits. The instruction proceeds from left to right and terminates when the result field C is filled. The instruction extends source fields A and/or B with zeros if they are shorter than field C. The G designator is not used and must be all zeros.

Data flag bit 53, 54, or 55 is set according to the condition of the result field as shown in table 6-44.

TABLE 6-44. DFB CONDITIONS FOR F0 THROUGH F7 INSTRUCTIONS

DFB Bit	Condition
53	Result field all zeros
54	Result field mixed
55	Result field all ones

Figure 6-75 shows an example of a logical string instruction operation. A logical exclusive OR (F0) instruction is used for the example. In the example, source field B contains a mask of all ones which is used to complement the binary number in source field A through the exclusive OR function. All indexes and field lengths are item counts, expressed in bits (for example, the source and result field lengths equal 28_{16} bits). The operation proceeds from the starting addresses of A, B, and C to the end of the result field (to address 7030_{16}). Each operation forms the exclusive OR or the corresponding bits in source fields A and B and stores the result in the corresponding position of field C.

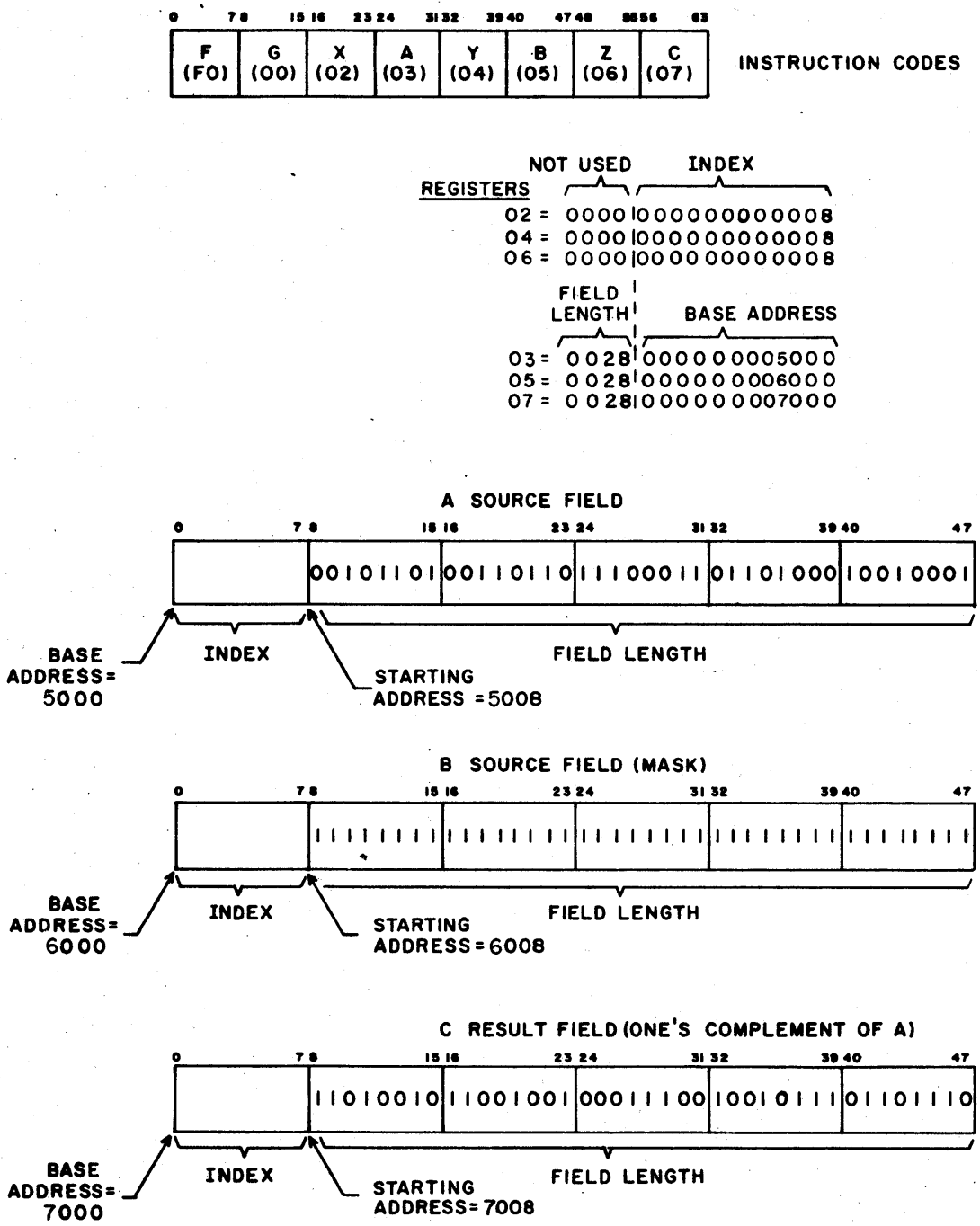
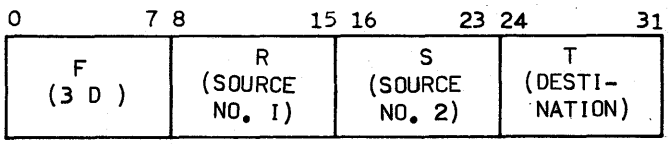


Figure 6-75. Example of Logical String Instruction (Logical Exclusive OR)

NONTYPICAL INSTRUCTIONS

These instructions perform operations such as register to storage transfers, formation of repeated bit masks, and maximum/minimum determinations that do not fall into any of the preceding categories of instructions. The separate instruction descriptions define the format and operation for these instructions. Appendix C provides a complete listing of the various nontypical instruction fields and the resulting termination conditions.

3D INDEX MULTIPLY (R) • (S) TO (T)
3C HALF WORD INDEX MULTIPLY (R) • (S) TO (T)



3D INDEX MULTIPLY (R) • (S) TO (T)

This instruction forms the product of the two's complement integers contained in the rightmost 48 bits of the registers specified by the R and S designators, respectively. The instruction stores the product in the rightmost 48 bits of register T and clears the leftmost 16 bits.

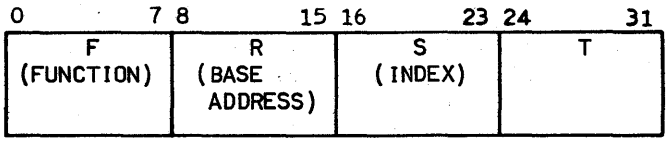
If the product or either operand exceeds $\pm 2^{47}-1$, the result is undefined.

3C HALF WORD INDEX MULTIPLY (R) • (S) TO (T)

This instruction forms the product of the two's complement integers contained in the rightmost 24 bits of the registers specified by the R and S designators, respectively. The instruction stores the product in the rightmost 24 bits of register T and clears the leftmost eight bits.

If the product or either operand exceeds $\pm 2^{23}-1$, the result is undefined.

5E/7E LOAD (T) PER (S),(R)
5F/7F STORE (T) PER (S),(R)
12/13 LOAD/STORE BYTE (T) PER (S),(R)



5E/7E Load (T) Per (S), (R)

These instructions load the 32/64-bit register T with the content of the address specified by (S) + (R), where (R) is the base address. For the 5E instruction, (S) is an item count in half-words, and for the 7E instruction, (S) is an item count in words. The index in S is shifted five/six places to the left before it is added to the base address. S and R are 64-bit registers. Overflow resulting from this addition has no effect if it occurs.

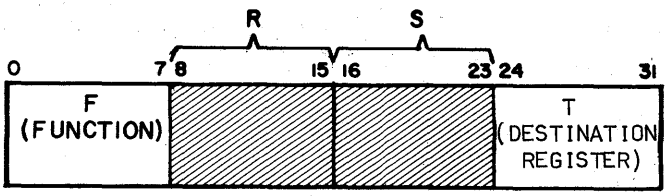
5F/7F Store (T) Per (S), (R)

These instructions store the content of the 32/64-bit register T in the address specified by (S) + (R), where (R) is the base address. For the 5F instruction, (S) is an item count in half-words, and for the 7F instruction, (S) is an item count in words. The index in S is shifted five/six places to the left before it is added to the base address. S and R are 64-bit registers. These instructions do not detect overflow if it occurs.

12/13 Load/Store Byte (T) Per (S), (R)

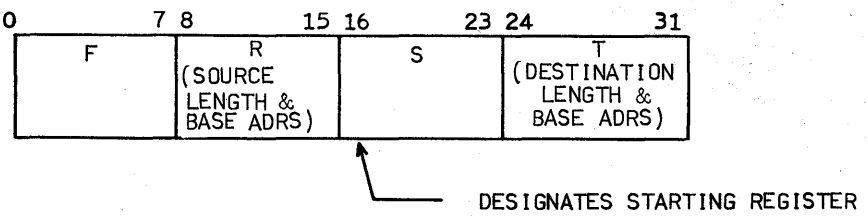
These instructions load/store a byte from/into the address specified by (R) + (S), where (R) is the base address and (S) is an item count in bytes. The index in S is shifted three places to the left before it is added to the base address. The byte is transmitted into/from bits 56 through 63 of register T. The remaining bits in T are cleared on a load and ignored on a store.

37 TRANSMIT JOB INTERVAL TIMER TO (T)



This instruction transmits the contents of the job interval timer into bits 40 through 63 of register T and clears bits 0 through 39 to zero. The designators R and S are undefined and must be set to zero. When executed in monitor mode, the operation of this instruction is undefined. This instruction does not deactivate the timer.

7D SWAP S→T AND R→S



This instruction moves to destination field T, a portion of the register file beginning at the 64-bit register specified by the rightmost eight bits of register S. The instruction also transmits source field R to the register file beginning at the 64-bit register specified by the rightmost eight bits of register S.

The leftmost 16 bits of registers R and T specify the field length in words for the source and destination fields, respectively. The field lengths of the source and destination fields may be different, but each must be even. A zero field length indicates no transfer for that field. Any transfer of words into or out of the register file that becomes exhausted of registers (beyond the bounds of the register file) causes the instruction to become undefined.

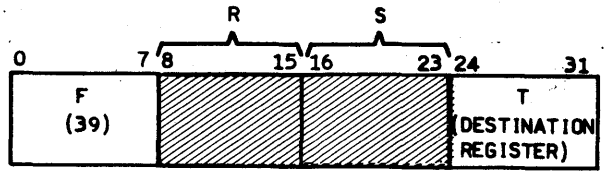
The rightmost 48 bits of registers R and T specify the base address of the source and destination fields, respectively. These addresses must specify an even 64-bit word in central storage. Bits 57 through 63 of registers R and T are undefined and must be set to zero. Overlap of the source and destination fields is allowed only if the base addresses for both fields are equal.

There are no restrictions relating to registers R, S, or T being in the range of the registers being swapped.

The starting register in the file specified by the rightmost eight bits of the register specified by S must be an even register.

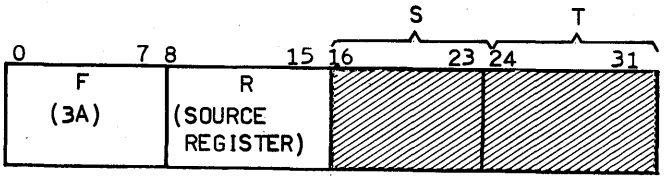
If the source field from the register file includes register zero, the computer transmits the trace register. However, new data from memory is never written into register zero by the swap (7D) instruction.

39 TRANSMIT REALTIME CLOCK TO (T)



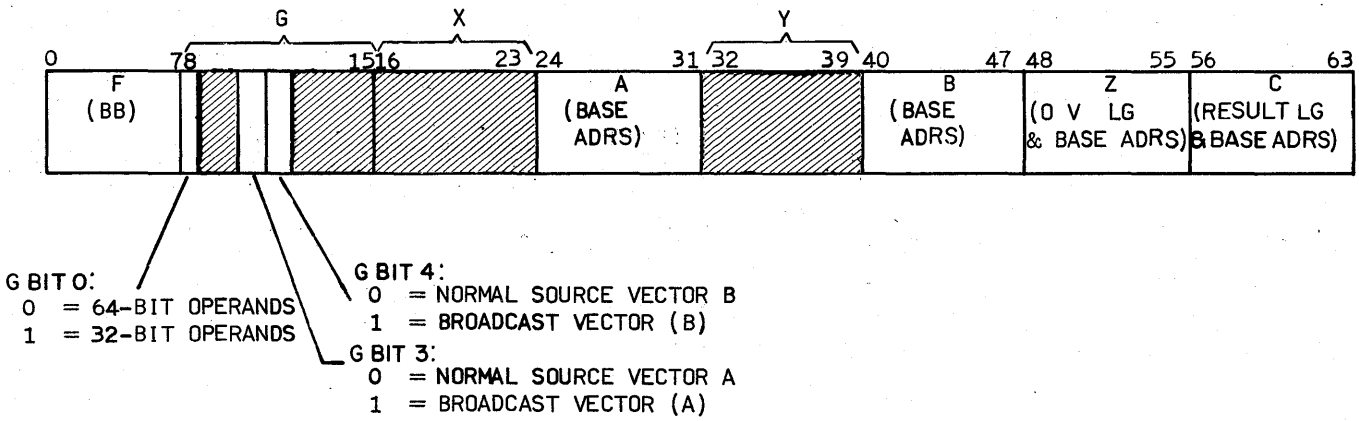
This instruction transmits the contents of the real-time clock to bits 16 through 63 of the register designated by T. Bits 0 through 15 of register T are cleared.

3A TRANSMIT (R) TO JOB INTERVAL TIMER



This instruction transmits bits 40 through 63 of the register designated by R to the job interval timer. When executed in the monitor mode, this instruction functions as a no-op.

BB MASK A, B→C PER Z



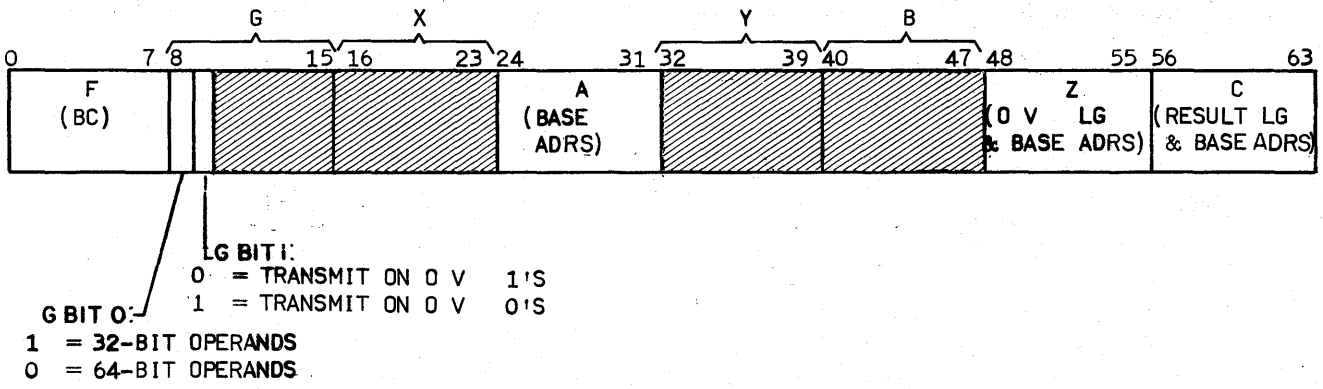
This instruction combines elements of vectors A and B to form result vector C as controlled by order vector Z. The general operation of this instruction follows the process described for sparse vector instructions in this section. When a one is detected in order vector Z, the next element of vector A is inserted into result vector C and the corresponding element

of vector B is skipped. When the instruction detects a zero in order vector Z, the instruction inserts the next element of vector B and skips the corresponding element of vector A. When all elements of A and B have been merged, the instruction transmits the resulting length of vector C to the length specification portion of register C as shown in figure 6-32.

The instruction format shows that bit 0 of the G designator determines whether 64- or 32-bit operands are used for the A and B vectors. The X and Y designators and bits 1, 2, and 5 through 7 of the G designator are not used and must be zeros. G bits 3 and 4 determine whether normal vector elements or broadcast elements are used for vectors A and B, respectively. The use of normal or broadcast source vectors are described in Vector Instructions in this section.

This instruction terminates when all bits of the order vector have been examined. The instruction recognizes no lengths for vectors A and B.

BC COMPRESS A→C PER Z



This instruction forms a sparse data vector field C by compressing vector field A. Sparse data vector field C consists of elements of vector field A corresponding to ones in sparse order vector Z. Thus, the elements of vector field A that correspond to the positions of ones in sparse order vector Z transfer in order to corresponding elements of sparse data vector field C if G designator bit 1 equals zero. If this bit is one, the elements of vector field A that correspond to zeros in sparse order vector Z are transferred to corresponding elements of sparse data vector field C.

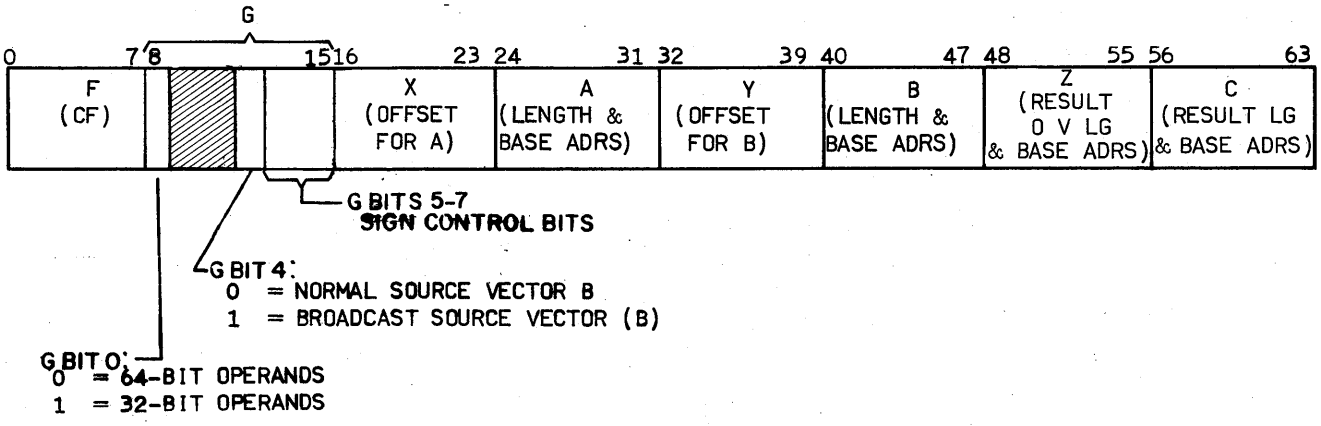
In a typical operation, one of the compare instructions first generates sparse order vector Z. The BC instruction uses the generated order vector as a means of discarding all near-zero elements of vector field A and still maintaining their positional significance through the order vector.

The instruction transfers the resulting length of sparse data vector C to the length specification portion of the register designated by C in the instruction word. If bit 0 of the G designator is zero/one, the operand size (elements of vectors A and C) is 64/32 bits, respectively. As shown in the instruction format, the X, Y, and B designators and bits 2 through 7 of the G designator are not used and must be zeros.

The instruction terminates when all bits of sparse order vector Z are used. The length specification portion of registers A and C (initial) is not used.

Figure 6-30 shows a simplified example of compressing a vector field into a sparse vector field.

CF ARITH. COMPRESS A→C PER B



This instruction forms sparse data vector† C and the associated sparse order vector Z by performing a floating point compare operation between elements of vector A and the elements of vector B. Each element of vector B is subtracted from the corresponding element of vector A. The conditions for comparing floating point operands are described in the Floating Point Compare Rules, appendix B. If an element of vector A is greater than or equal to the corresponding element of vector B (An $\geq$ Bn), the instruction stores the element of A as the corresponding element of sparse data vector C and sets the associated order vector bit. If the element of vector A is less than the corresponding element of vector B (An < Bn), the element of A is not stored in sparse data vector C and the associated order vector bit is cleared. The element of C is not skipped if An < Bn. Thus, in the case of broadcast vector (B), this instruction provides a means of generating a sparse vector field by comparing the elements of a source vector field with a fixed threshold element.

The registers designated by X and Y contain the offsets for vectors A and B, respectively.

† The sparse vector part of this section describes the general format of sparse vectors.

The elements of vectors A and B are in floating point format.† The sign control bits of the G field may specify operations on the elements of vector A and/or B before the floating point compare is made. However, the element of A, if stored in C, will be the original element as read from A. The compare operation follows the floating point compare conditions as described in the branch instruction section. In the comparison, only (R) $\geq$ (S) condition is detected where, in this case, An and Bn are substituted for (R) and S), respectively. If the instruction detects an indefinite operand for vector A and/or B, the indefinite operand is stored as the corresponding element of vector C and the associated bit of the order vector is set.

The instruction format shows that if bit 0 of the G designator is a zero/one, the vector elements are 64-bit/32-bit operands, respectively. If bit 4 of the G designator is a one, a constant element is broadcast for vector B as described in Vector Instructions in this section. In this case, the Y designator is not used. G bits 1 through 3 are not used and must be zeros. G bits 5 through 7 function as sign control bits as described in Vector Instructions.

This instruction terminates when all the elements of vector A have been compared. At termination, the instruction stores the length (in bits) of the generated order vector into the length portion (bits 0 through 15) of the register specified by Z. The number of elements stored in vector C is stored in the length portion of register C, thus providing the field length of the generated sparse vector. If the length of vector B is shorter than the length of vector A, the instruction extends the B field with machine zero elements to equal the A field length. The applicable data flag bit is 46 (indefinite result).

Figure 6-76 is an example of an arithmetic compress instruction with assumed instruction code, register contents, and source vector field A. In this example, a broadcast floating point constant B is compared with source vector elements A_1 through A_6. Element A0 is not compared because of the offset. The A vector elements are indicated as being An $\geq$ B or An < B. Thus, the instruction in this example generates a 4-element result vector C and a 6-bit order vector Z. The 6 and 4 values are stored in the field length portions of registers 08 and 09, respectively.

† Appendix B describes the floating point format.

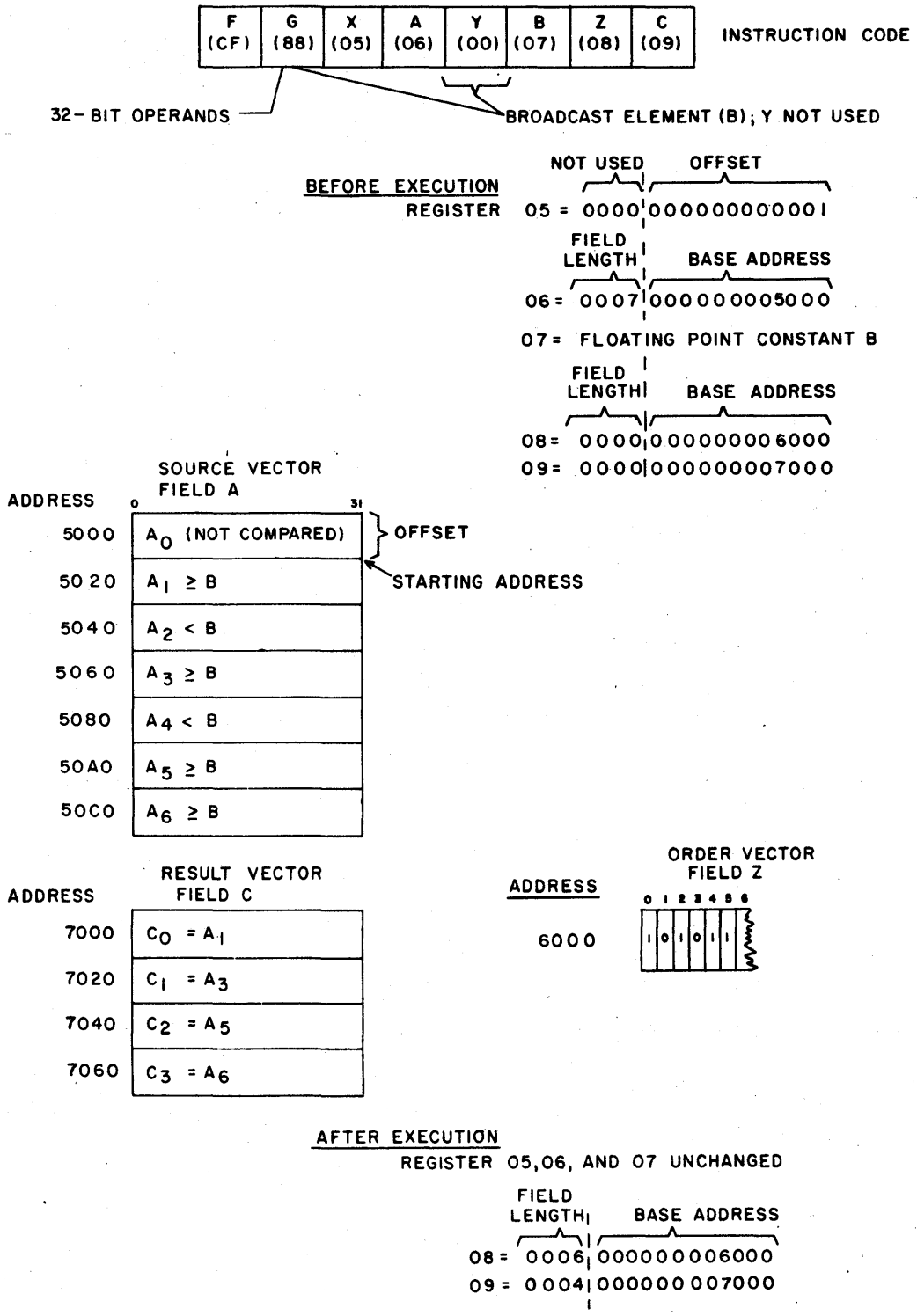
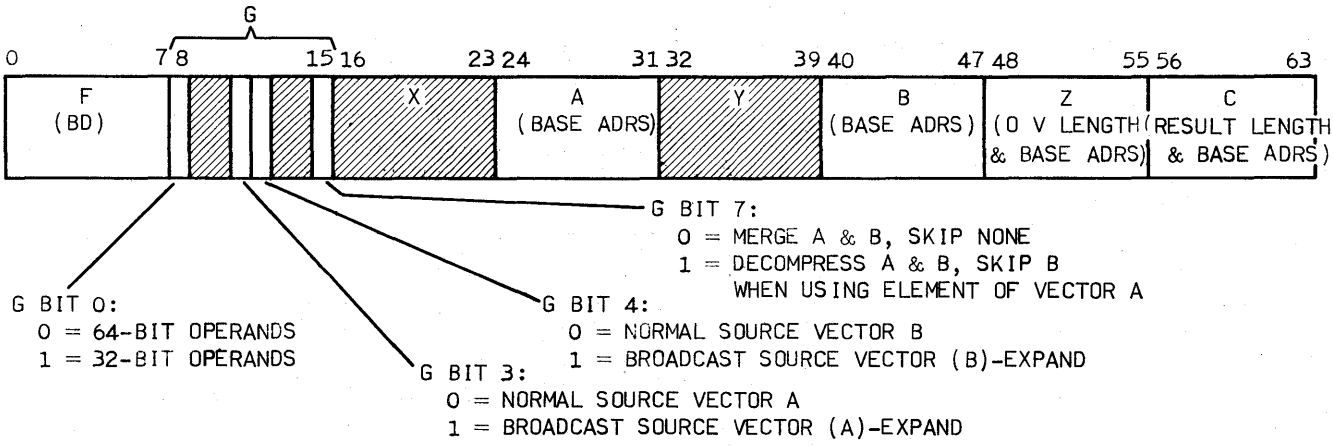


Figure 6-76. Example of Arithmetic Compress A→C Per B Instruction

BD MERGE A, B→C PER Z



This instruction merges the elements of vector field A with the elements of vector field B to form result vector field C as controlled by order vector Z. Thus, this instruction could be used to reform a vector field from a sparse vector with a broadcast near-zero element. When the order vector Z contains a one in a given position, the instruction inserts the next element from vector field A into vector field C. If the order vector contains a zero, the instruction inserts the next element from vector field B in the result field (figure 6-77). The instruction transmits the resulting length of vector C to the length specification portion (bits 0 through 15) of register C.

Field B vector elements are controlled by G bit 7. When G bit 7 is a zero, the operation (called merge) combines vectors A and B. When G bit 7 is set (decompress), an element of vector B is skipped for each element of vector A used. No elements of vector A are skipped when elements of vector B are used.

The instruction format diagram shows that if bit 0 of the G designator is a zero/one, the operand size (vector A and B elements) is 64/32 bits, respectively. The X and Y designators and G bits 1, 2, and 5 through 7 are not used and must be zeros. Bits 3 and 4 of the G designator determine whether a constant element is broadcast from the registers designated by A and B, respectively. If G bit 3 or 4 is a one, the operation is called expand.

The BD instruction terminates when all of the bits of the order vector have been processed. The field length specifications for vectors A and B are not used.

THE Z-BIT STRING IS USED FOR ALL THREE EXAMPLES. G BITS NOT INDICATED ARE ZEROS.

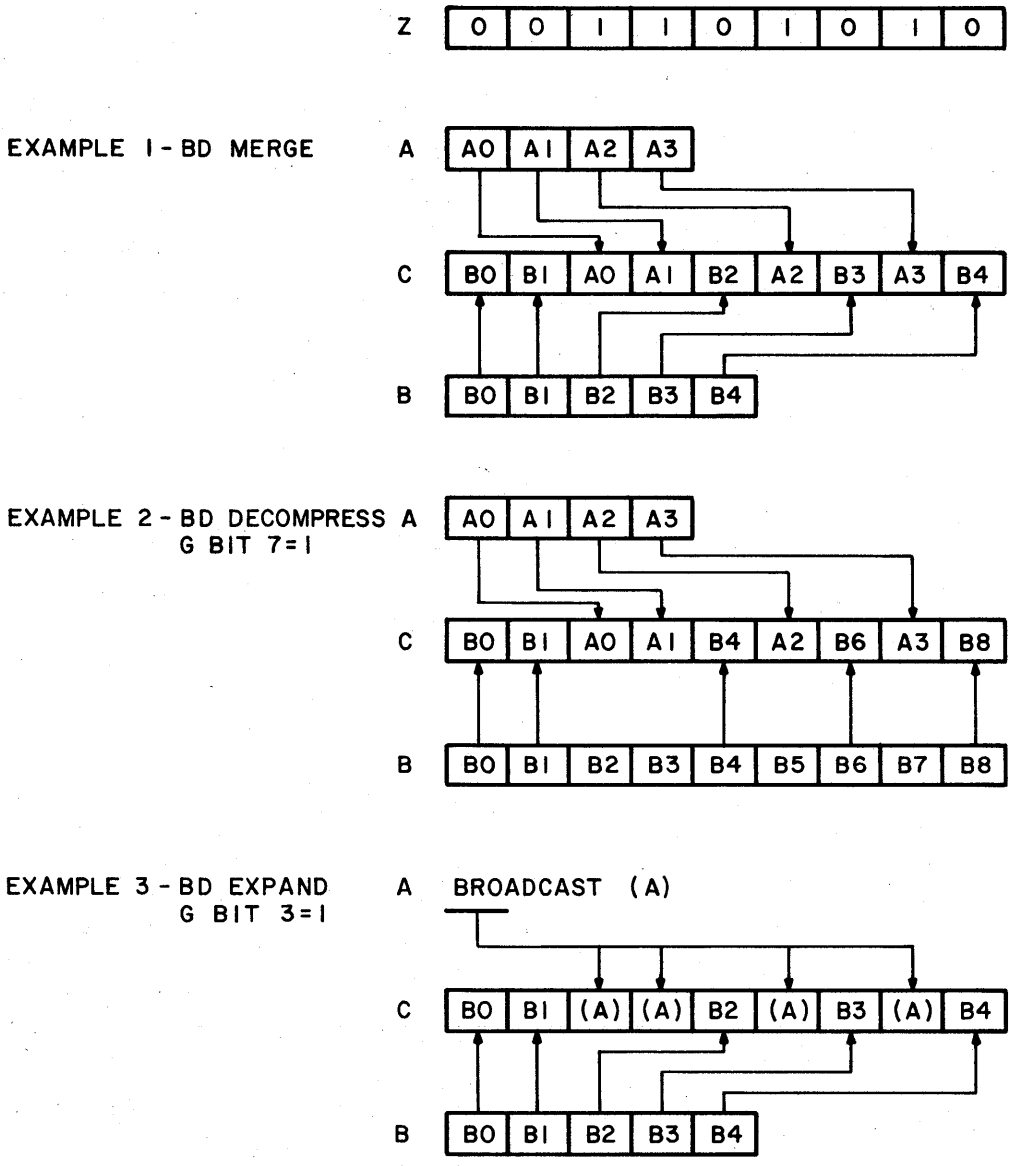
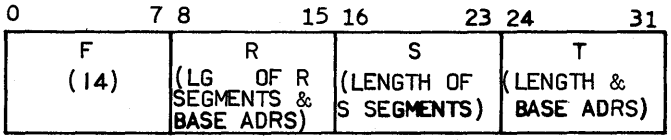


Figure 6-77. Examples of BD Merge Instruction

14 BIT COMPRESS



This instruction compresses specified segment lengths (in bits) of source field R into result field T. The R designator code in the instruction specifies a 64-bit register which contains the length of the R segments in the leftmost 16 bits and the base address of the source field in the rightmost 48 bits (figure 6-78). The register denoted by S contains the length of the segments in the source field to be skipped in the compress operation. The rightmost 48 bits of register S are not used.

Register T contains the destination field length in the leftmost 16 bits and the base address of the destination field in the rightmost 48 bits.

The bit compress operation successively transmits the segment lengths of the source field, as specified by R, to corresponding lengths of the destination field. The instruction moves from left to right in the source and destination fields. The instruction skips the segment lengths of the source field as specified by S.

Figure 6-78 shows that the instruction transfers segments R_1, R_2, and R_3 in the source field to corresponding segment lengths of the result field. Source field segments S_1 and S_2 are skipped. The operation continues until the T field length is filled. If the field length specified by R or T is zero, the instruction functions as a no-op.

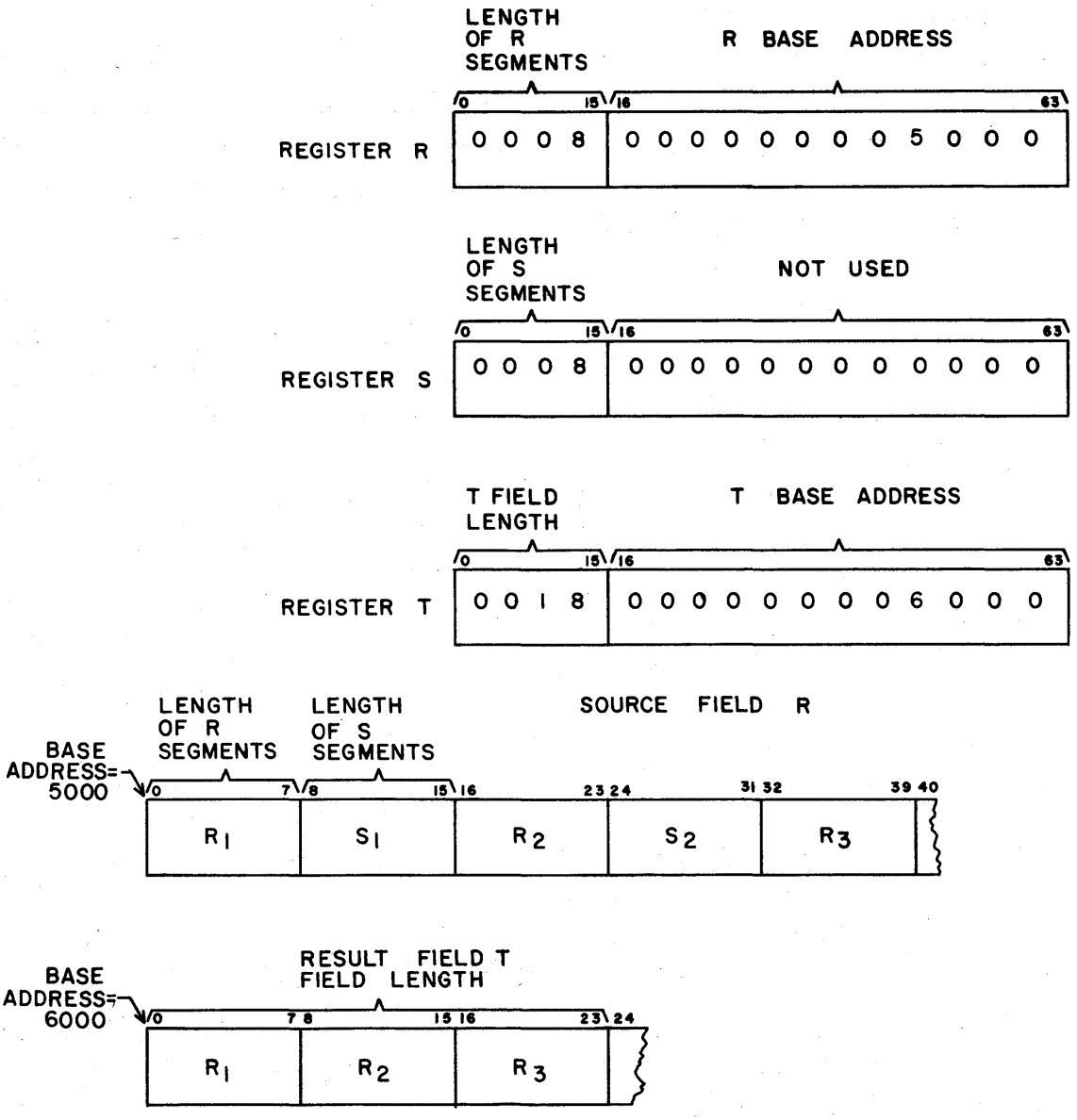
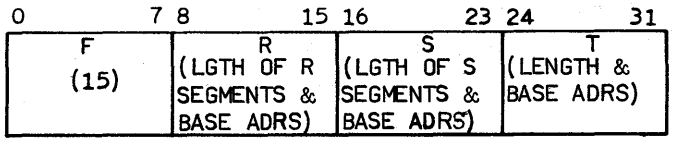


Figure 6-78. Example of Bit Compress Instruction

15 BIT MERGE

16 BIT MASK



15 BIT MERGE

The bit merge instruction merges specified segment lengths (in bits) of source fields R and S into result field T. The 64-bit register specified by R contains the length of the R segments in the leftmost 16 bits and the base address of the R source field in the rightmost 48 bits (figure 6-79). The register denoted by S contains the length of the S segments in the leftmost 16 bits and the base address of the S source field in the rightmost 48 bits. Register T contains the destination field length in the leftmost 16 bits and the base address of the destination field in the rightmost 48 bits.

The bit merge operation successively merges the segment lengths of the R source field with segment lengths of the S source field into corresponding lengths of the destination field. The instruction moves from left to right in the source and destination fields.

Figure 6-79 shows that the 15 instruction merges segments R_1, R_2, and R_3 in source field R with segments S_1 and S_2 into corresponding segment lengths of the destination field. The operation continues until the T field length is filled.

If bits 16 through 63 of the S register are cleared, the instruction transmits zeros to the corresponding segment lengths in the destination field. If the field length specified by the R, S, or T registers is zero, the instruction functions as a no-op.

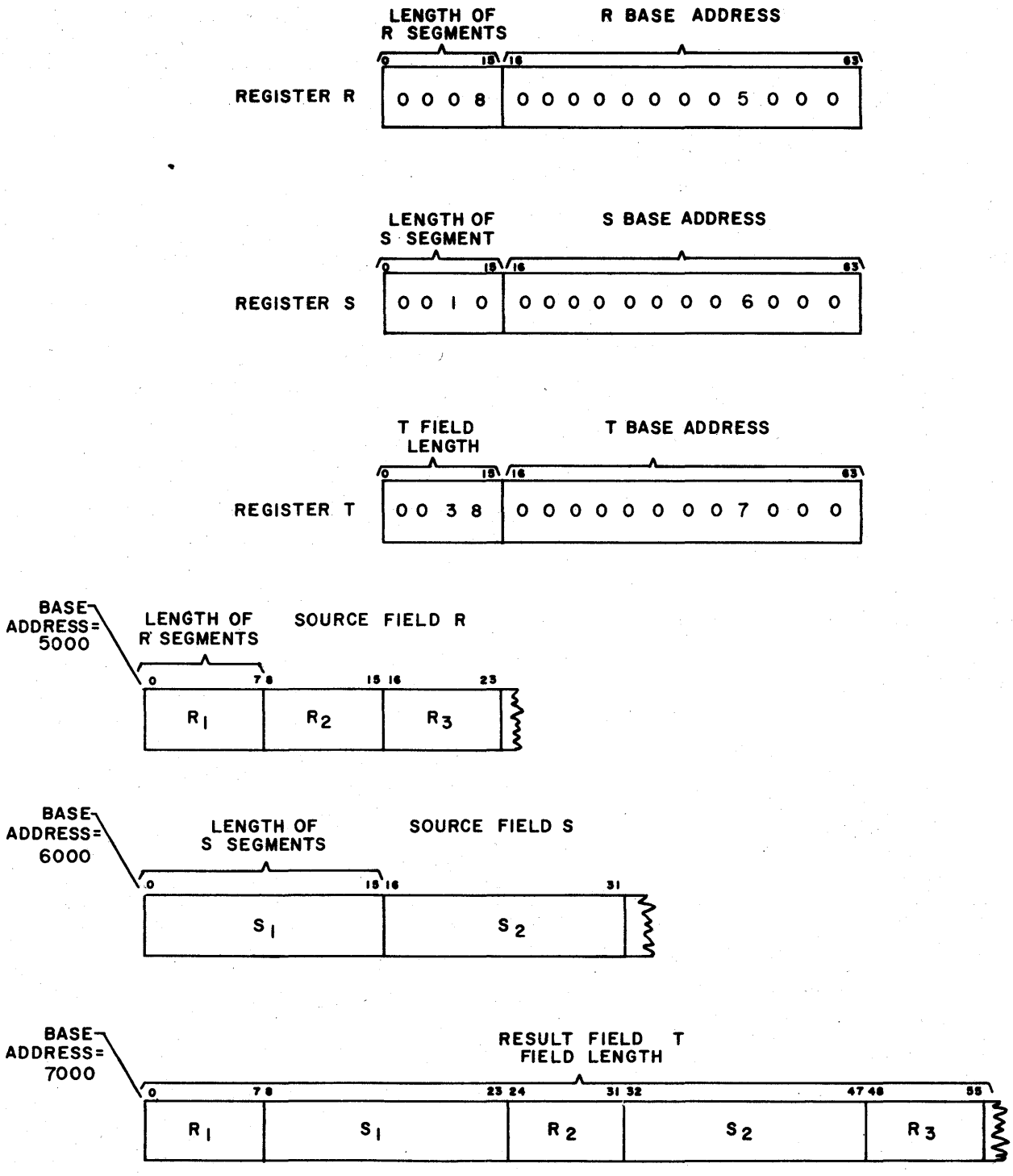


Figure 6-79. Example of Bit Merge Instruction

16 BIT MASK

The bit mask instruction is similar to the bit merge instruction. The specified R, S, and T registers contain segment lengths, base address, and field length in the same manner. However, the bit mask instruction (figure 6-80), moving from left to right, transmits a segment equal to the length specified by R to the corresponding segment length in the destination field. The 16 instruction then transmits a segment of field S equal to the segment length specified by the S register starting at an address equal to the base address plus the R segment length. The next segment of the R source field to be transmitted to the destination field starts at an address equal to the R base address plus the R segment length plus the S segment length. As in the bit merge instruction, if bits 16 through 63 of the S register are cleared, the instruction transmits zeros to the corresponding segment lengths in the destination field. In the same manner, if the field lengths specified by the R, S, or T register is zero, the instruction becomes a no-op. The bit mask operation continues in this manner until the destination field is filled.

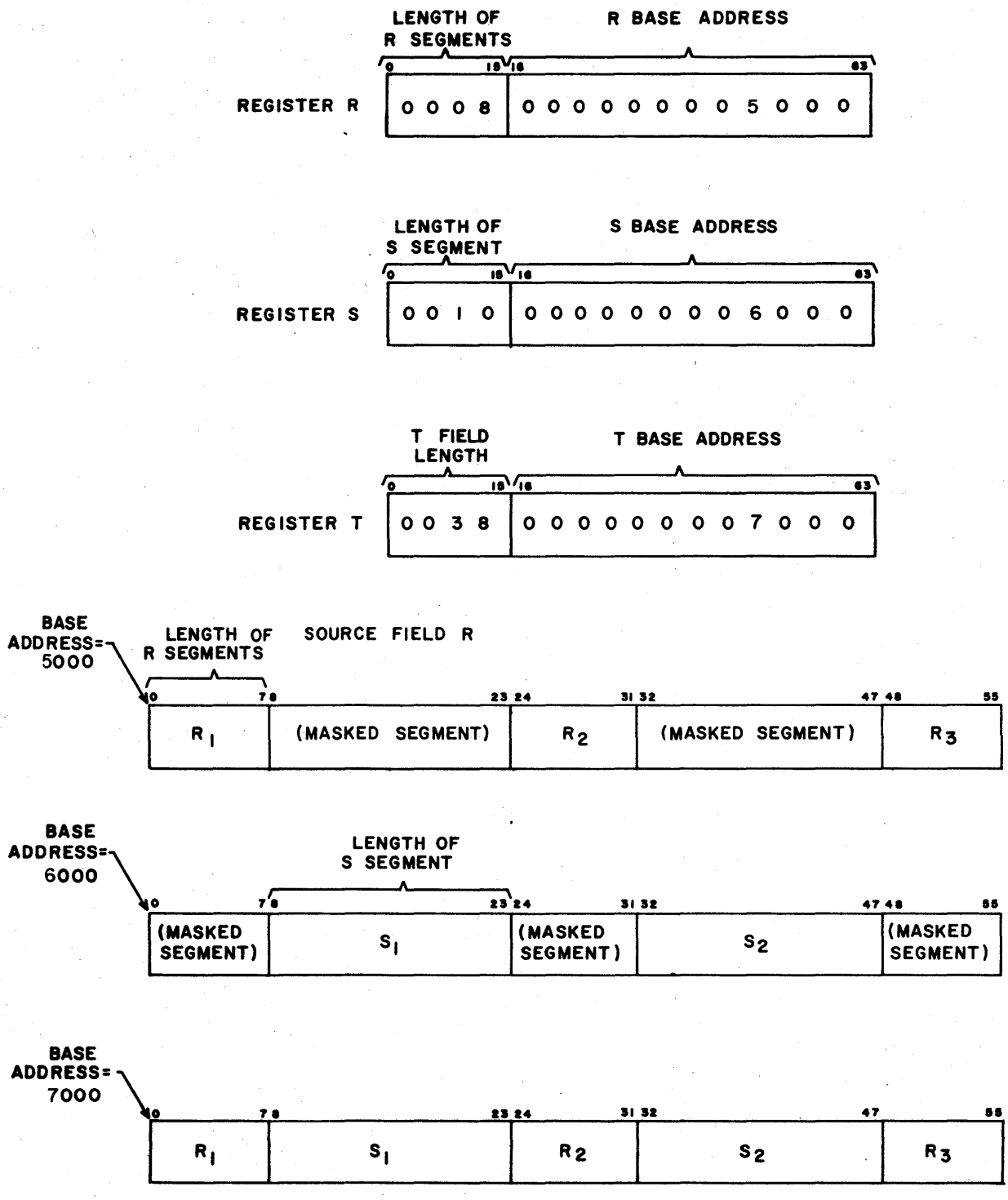
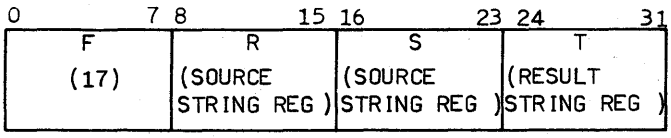


Figure 6-80. Example of Bit Mask Instruction

17 CHARACTER STRING MERGE



This instruction merges records† in the string specified by R with the records in the string specified by S in ascending order. The resulting merged records are stored in the string specified by T. Bits 0 through 15 of register T contain an item count of the number of bytes in a record. Bits 16 through 63 contain the starting address of the result string.

The registers specified by R and S specify the two source strings. Bits 0 through 15 of these registers contain the number of records in the corresponding string. Bits 16 through 63 specify the starting address of the string.

The instruction merges the R and S strings by comparing the leading records of each string and by transferring the numerically†† smaller of the two records to the result field, starting at the base address. The next record in the string from which the record was transferred becomes the new leading record. The comparisons continue in this manner until one of the source strings is exhausted. The instruction then moves the remainder of the unexhausted string to the end of the result string.

If the record length specified by T is zero, or the number of records specified by both R and S are zero, this instruction becomes a no-op.

Figure 6-81 shows an example of the character string merge instruction. Note that in the example, alphabetical characters are used to denote the relative size of the records. Each character represents one 8-bit byte of data. For example in the first comparison, leading record AA (R string) is smaller than leading record AB (S string). As a result, record AA transfers to the result string and the next record in the R string (BA) becomes the leading record which is compared with AB in the second comparison.

The comparisons continue as shown in figure 6-81 until the R string is exhausted in the eleventh comparison. Following this comparison, the remainder of the S string (record MN) transfers as the last record of the result string.

†In this case, a record is defined as a number of 8-bit bytes.

††The records are all assumed to be positive. The R record is transferred when identical records are compared.

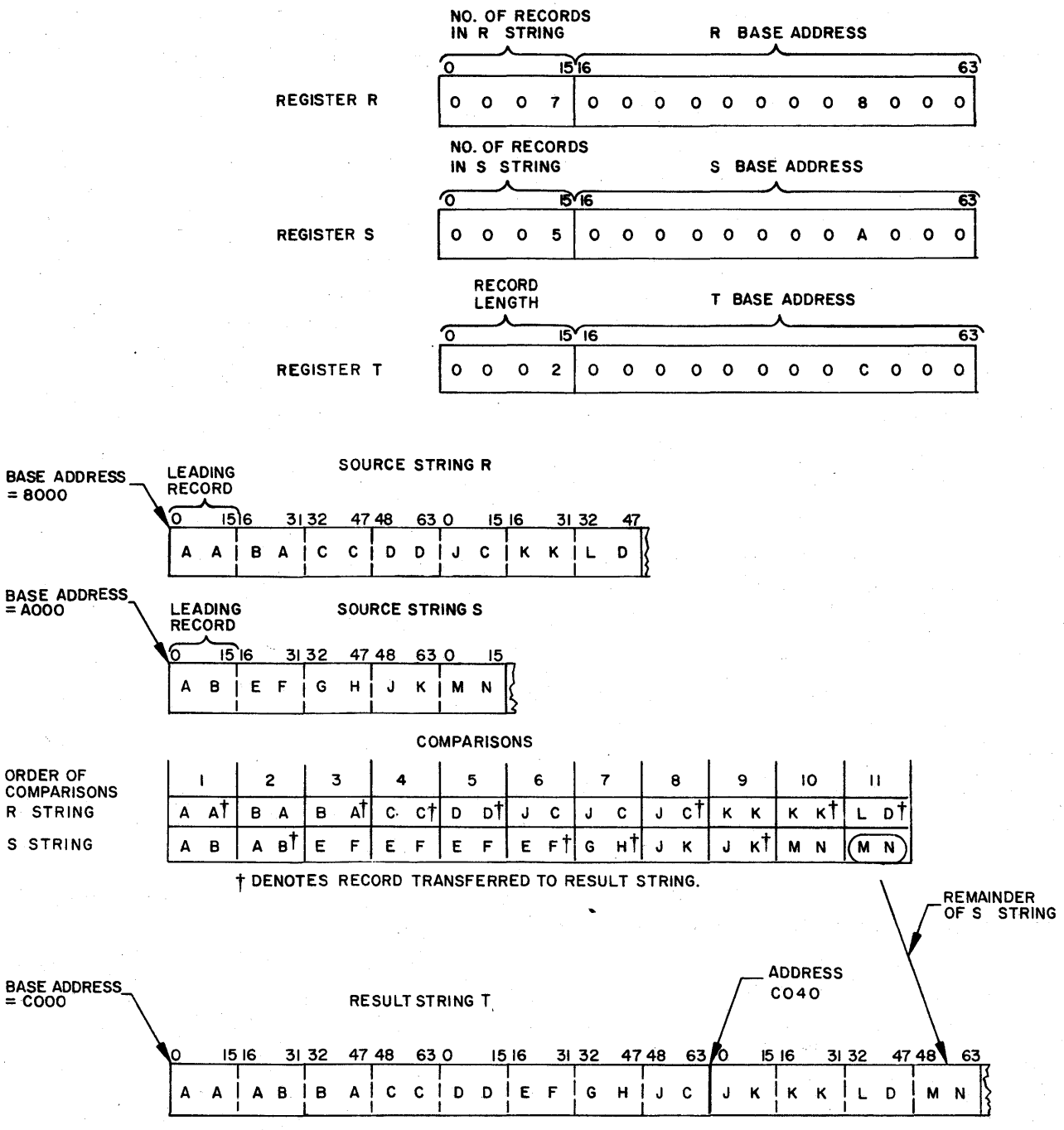
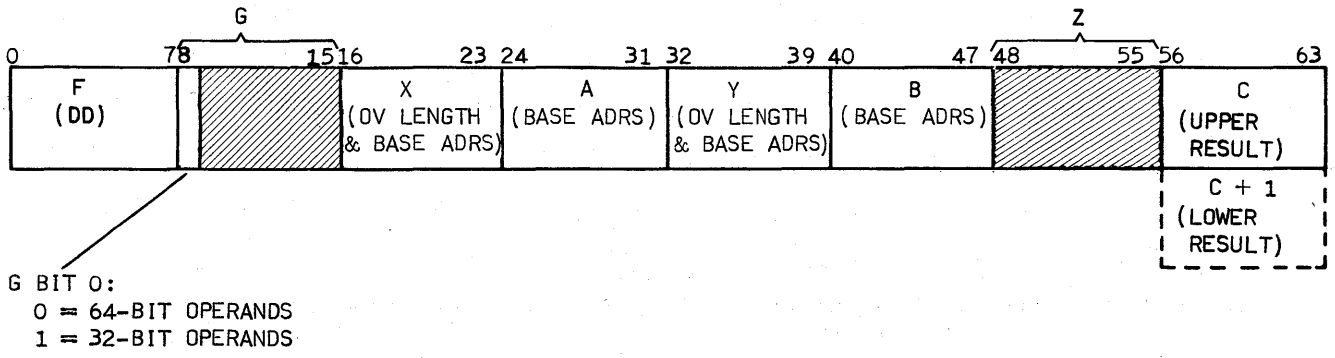


Figure 6-81. Example of the Character String Merge Instruction

DD SPARSE DOT PRODUCT TO (C) AND (C+1)



This instruction multiplies the elements of sparse vector A by the elements of sparse vector B and forms the sum of the products. This instruction functions much like a sparse vector multiply instruction, except rather than producing a sparse vector as a result, the DD instruction forms the sum of all the individual products as a result. The operation uses double precision, unnormalized, floating-point arithmetic† for both the multiply and subsequent addition. Vector A and B each are associated with an order vector as in the sparse vector instructions. The product of a given pair of vector A and B elements is added to the accumulating sum only when the corresponding, bit-by-bit, logical AND of the two source order vectors is a one. The instruction stores the upper and lower result in the registers denoted by C and C + 1, respectively.

The instruction code shows that if G bit 0 is a zero/one, the operands (vector elements) are 64/32 bits, respectively. The Z designator and G bits 1 through 7 are not used and must be zeros. Registers X and Y contain the addresses and lengths of the A and B source order vectors in the rightmost 48 and leftmost 16 bits, respectively.

This instruction terminates when all of the bits of the shorter of the two source order vectors have been examined. The C designator must be an even number. If this number is odd or zero, the results of the instruction are undefined. If the order vectors disable any multiply operations, the corresponding result is machine zero.

† Appendix B describes floating-point arithmetic and order-dependent result considerations.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). Data flag bits 43 and 46 are determined only by the final upper and lower result. If the upper result is indefinite, the lower result is less than 9000_{16}. In this case, the exponent of the upper result may be greater than 9000_{16} and will be stored as is and will not be forced to machine zero. The instruction sets data flag bit 42 if any of the multiply operations overflow.

The computer forms two partial products, X and Y. The sum of the products are accumulated in the following manner, dependent upon the logical AND of the two source order vectors. The order vectors used in this example correspond to figure 6-83.

Match	Match	Order Vector

$(A_0 \bullet B_0) + (A_2 \bullet B_2) + \ldots + (A_n \bullet B_n) = X$

No Match	B Exhausted	Order Vector

$(A_1 \bullet B_1) + (A_3 \bullet B_3) + \ldots + (A_n \bullet B_n) = Y$

where: A_n are elements of vector A
B_n are elements of vector B, and
X and Y are partial sums of the product

Sum X and sum Y (both double precision quantities) are then added to form the final sum.

Figure 6-82 is an example of a sparse dot product instruction with assumed instruction code, register content, and vector source fields. In this example, the B source field and order vector is one element shorter than the A source field and A order vector. Thus, the instruction terminates after the $A_2 \bullet B_2$ product is added to the sum.

Since bit 1 of the B order vector is a zero, the bit-by-bit AND does not enable the additon of the $A_1 \bullet B_1$ product to the accumulating sum. The final sum of the products (S_f) is stored in registers C(0A) and C + 1(0B), respectively.

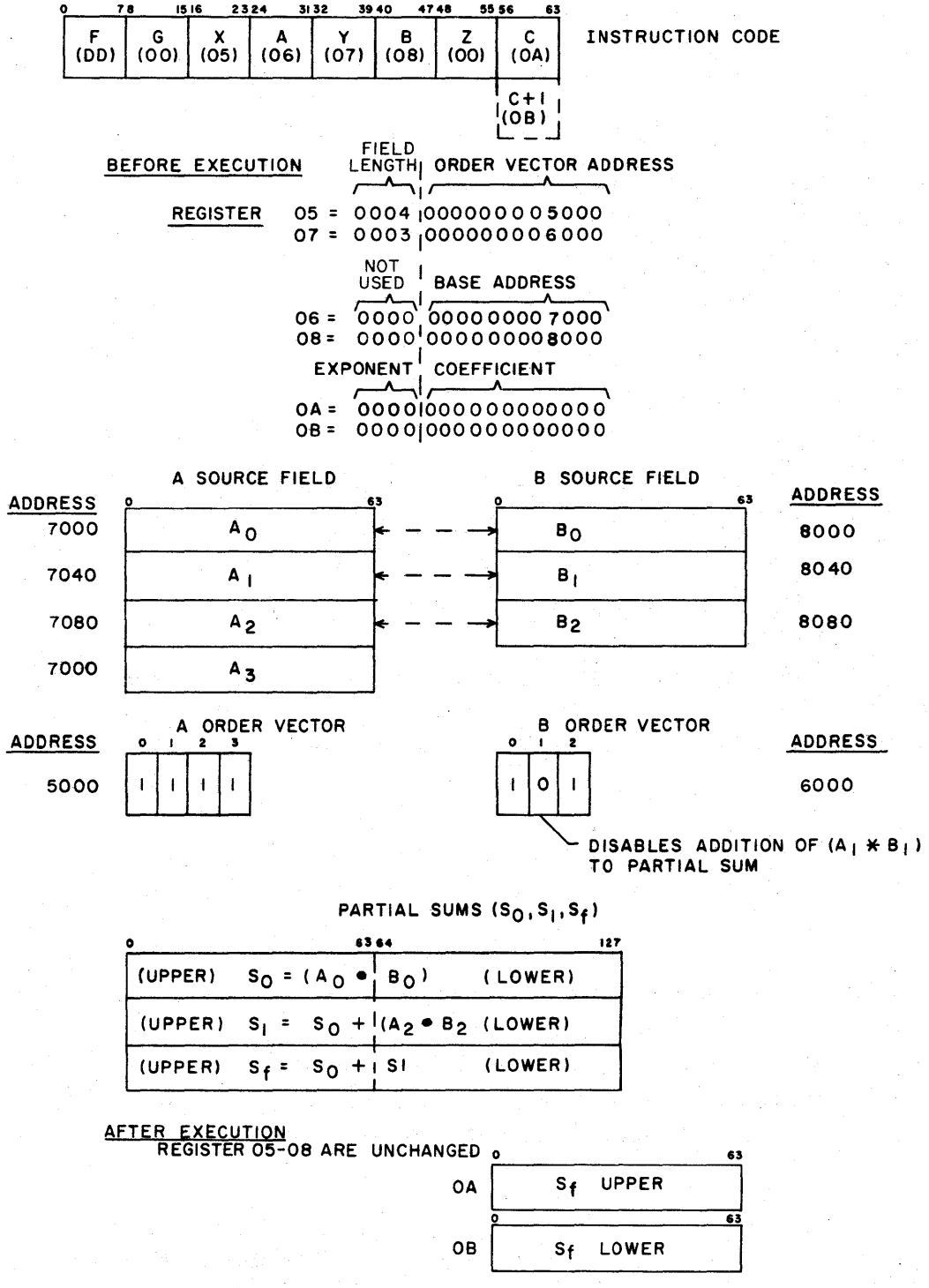
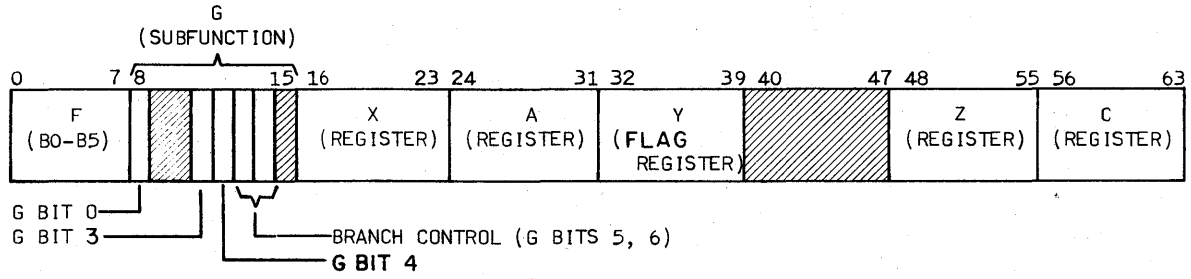


Figure 6-82. Example of Sparse Dot Product to (C) and (C+1) Instruction

COMPARE INSTRUCTIONS (B0 THROUGH B5)

The central computer has expanded capabilities for the B0 through B5 instructions (refer to Branch Instructions in this section for other use of these instructions). Which set of B0 through B5 instructions to use depends on the values given to G bits 0 through 3.

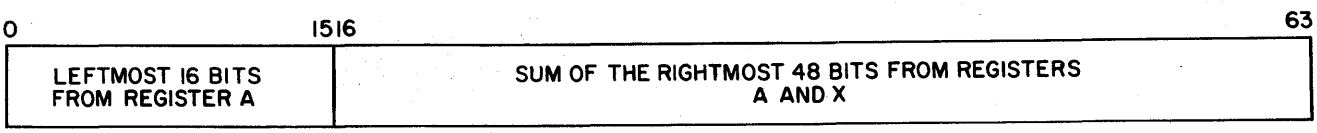
B0 COMPARE INTEGER, SET CONDITION IF (A) + (X) = (Z)
B1 COMPARE INTEGER, SET CONDITION IF (A) + (X) ≠ (Z)
B2 COMPARE INTEGER, SET CONDITION IF (A) + (X) ≥ (Z)
B3 COMPARE INTEGER, SET CONDITION IF (A) + (X) < (Z)
B4 COMPARE INTEGER, SET CONDITION IF (A) + (X) ≤ (Z)
B5 COMPARE INTEGER, SET CONDITION IF (A) + (X) > (Z)



If G bit 1 is 0 and G bit 2 is 1, these instructions compare two integer operands from register A and X. If G bit 0 is clear (0), registers A, X, Y, C, and Z are 64 bits. If G bit 0 is set (1), these registers are 32 bits. Register B is not used and must be set to zero.

For these instructions, G bit 1 and 2 are 0. If G bit 0 is cleared (0), registers A, X, C, and Z are 64 bits. If G bit 0 is set (1), registers A, X, C, and Z are 32 bits. Registers B and Y are 64 bits.

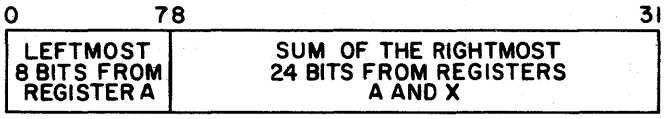
If G bit 0 is 0, the sum of the rightmost 48-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 48 bits of register Z according to the specified condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 48 bits of register C. The leftmost 16 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



The sum of the rightmost 48 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values:

G bit 3 = 0	The integers compared are the 48-bit result of registers A and X and the rightmost 48 bits read from register Z.
G bit 3 = 1	The integers compared are the 64 bits stored in register C and the 64 bits read from register Z. This compare is defined for the B0 and B1 instructions only.
G bit 4 = 0	The integers compared are interpreted as signed two's complement numbers.
G bit 4 = 1	The integers compared are interpreted as unsigned numbers.

If G bit 0 is 1, the sum of the rightmost 24-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 24 bits of register Z, according to the specified condition. The original content of register *Z* is read before the sum of registers A and X is stored in the rightmost 24 bits of register C. The leftmost 8 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



Then the sum of the rightmost 24 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values:

G bit 3 = 0	The integers compared are the 24-bit result of registers A and X and the rightmost 24 bits read from register Z.
Git bit 3 = 1	Undefined.
G bit 4 = 0	The integers compared are interpreted as unsigned two's complement numbers.
G bit 4 = 1	The integers compared are interpreted as signed numbers.

Refer to table 6-13 for integer ranges.

If the specified compare condition is met, a 64- or 32-bit quantity (depending on G bit 0) 00....0001 is stored in register Y and the program reads the next sequential instruction.

If the specified compare condition is not met, a 64- or 32-bit quantity (depending on G bit 0) 00....000 is stored in register Y and the program reads the next sequential instruction.

If one of the following conditions occurs, the operation becomes undefined.

- G bit 0 is 1 and G bit 3 is 1
- G bit 3 is 1 for B2, B3, B4, and B5
- G bit 5 is 1, G bit 6 is 1, or G bit 7 is 1
- The C designator is equal to the Z designator

B0 COMPARE FP, SET CONDITION IF (A) = (X)

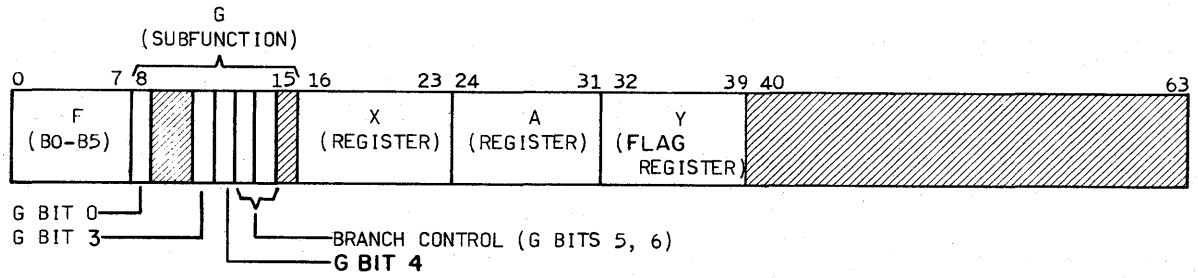
B1 COMPARE FP, SET CONDITION IF (A) ≠ (X)

B2 COMPARE FP, SET CONDITION IF (A) ≥ (X)

B3 COMPARE FP, SET CONDITION IF (A) < (X)

B4 COMPARE FP, SET CONDITION IF (A) ≤ (X)

B5 COMPARE FP, SET CONDITION IF (A) > (X)



If G bit 1 is 1 and G bit 2 is 1, these instructions compare two floating-point operands from register A and X according to the floating-point compare rule in appendix B. If G bit 0 is clear (0), registers A, X, and Y are 64 bits. If G bit 0 is set (1), these registers are 32 bits. Registers B, C, and Z are not used and must be set to zero.

If the specified compare condition is met, a 64- or 32-bit quantity (depending on G bit 0) 00....0000 is stored in register Y and the program reads the next sequential instruction.

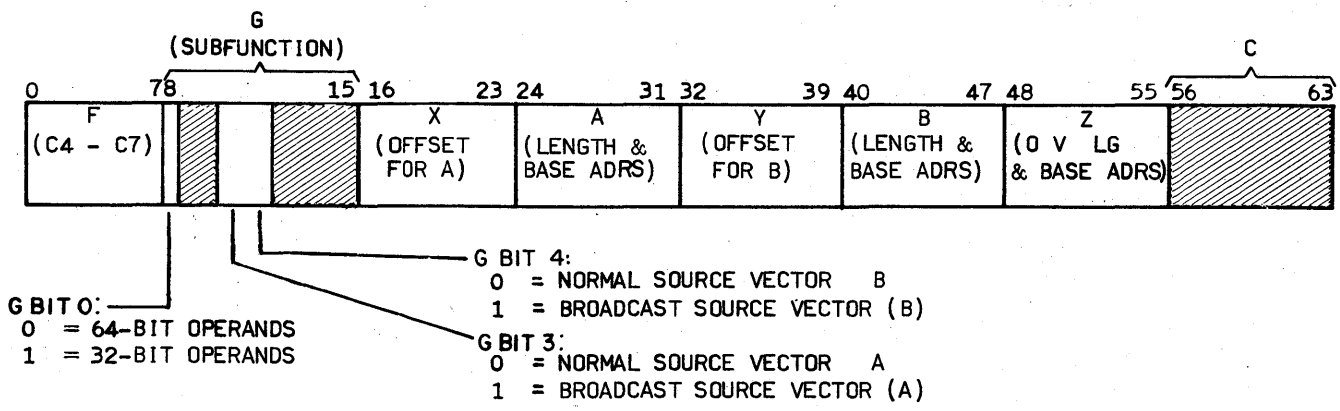
If the specified compare condition is not met, a 64- or 32-bit quantity (depending on G bit 0) 00....001 is stored in register Y and the program reads the next sequential instruction.

If one of the following conditions occurs, the operation becomes undefined.

- Any one of G bits 3 through 7 is set (1).
- Designators B, Z, and/or C are not equal to zero.

Applicable data flag bit is 46 (indefinite result).

C4 COMPARE EQ; A = B, ORDER VECTOR→Z
C5 COMPARE NE; A ≠ B, ORDER VECTOR →Z
C6 COMPARE GE; A ≥ B, ORDER VECTOR →Z
C7 COMPARE LT; A < B, ORDER VECTOR →Z



NOTE: THE C + 1 DESIGNATOR IS NOT USED BY THIS INSTRUCTION.

These instructions compare successive elements of vector A with corresponding elements of vector B by subtracting vector B from vector A. The elements of the vectors are in floating-point format.† The conditions for comparing floating-point operands are described in the Floating-Point Compare Rules, appendix B. If the specified compare condition is met (A =, ≠, ≥, or < B), the instruction sets the corresponding bit of order vector Z. If the condition is not met, the instruction clears the corresponding bit of Z. The instruction terminates when the order vector Z field is filled. Thus, the compare instructions provide a means of generating an order vector for reducing a vector field to a sparse vector field.

The instruction format shows that G bits 1, 2, 5 through 7, and the C designator are not used and must be zeros. The C + 1 designator is not used. Thus, no offset can be assigned to order vector Z. The floating-point compare conditions as described in branch instructions are used in the comparisons of the vector elements.

†Appendix B describes the floating-point formats.

The registers specified by X and Y contain the offsets for vectors A and B, respectively. When a constant is broadcast for either source vector, no field length is specified for that vector, and the offset is not used.

The field lengths and base addresses for vectors A, B, and Z are contained in the registers specified by A, B, and Z, respectively. The lengths of vectors A and B are in words (64-bit operands) or half-words (32-bit operands), and the length of order vector Z is in bits.

The applicable data flag bit is 46 (indefinite result).

Figure 6-83 is a simplified example of a compare instruction (C6) with assumed instruction codes, register contents, and source vector field A. In the example, a broadcast constant of +1 is used for vector field B. The elements of vector field A at addresses 5040, 5060, 50E0, and 5100 set the corresponding bits of order vector Z, while the elements at addresses 5080, 50A0, and 50C0 clear the corresponding bits. Although the coefficients of the elements at addresses 5080 and 50A0 are larger than the coefficient of constant B, the negative exponents cause the results of the floating point subtract operation (normalized upper) to be negative (A < B).

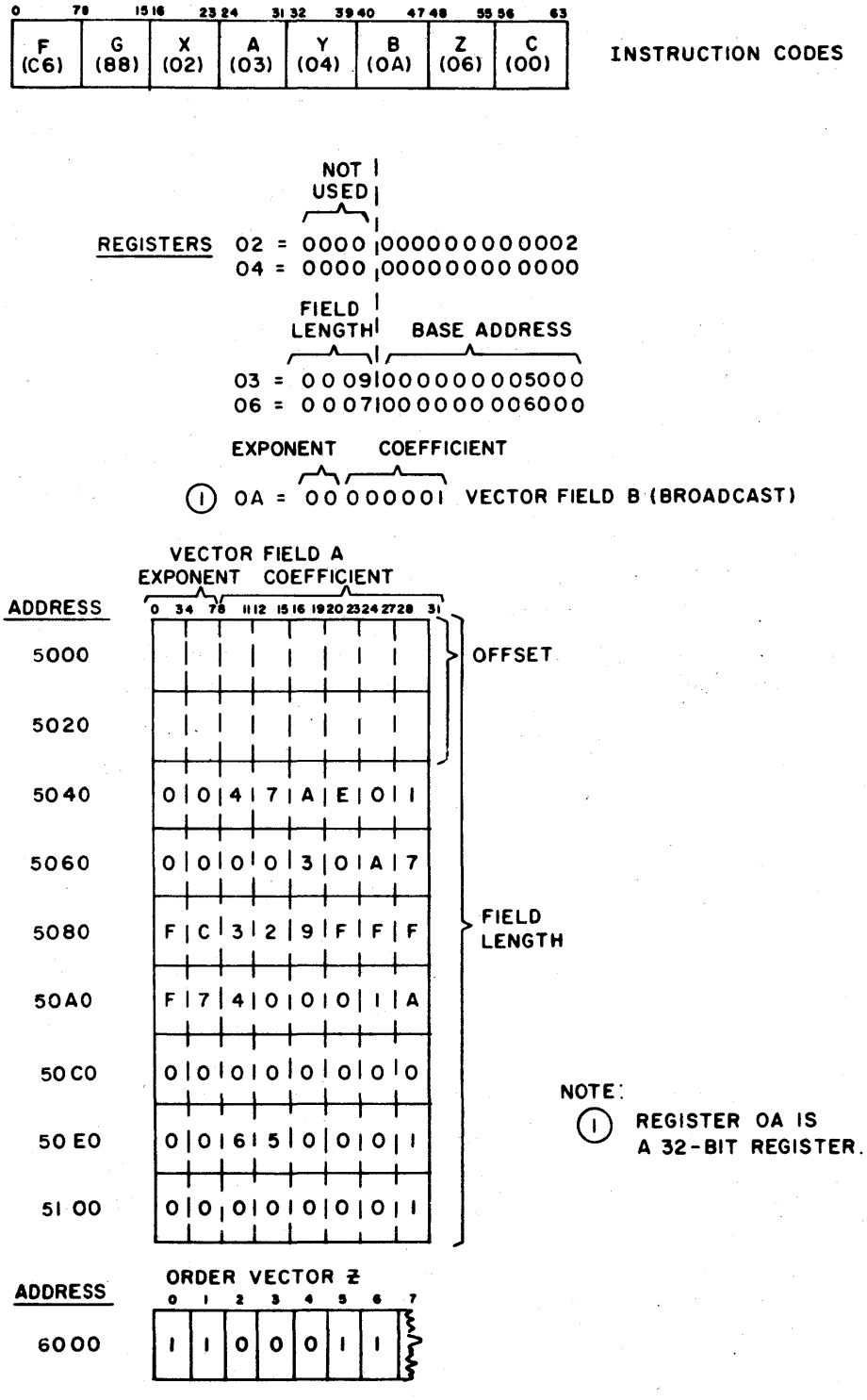


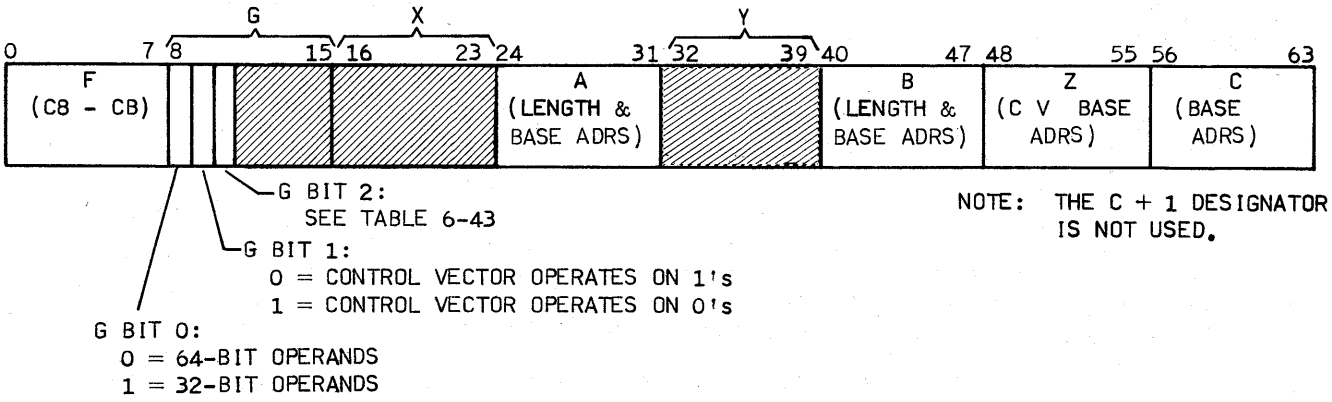
Figure 6-83. Example of Compare GE; A $\geq$ B; Order Vector → Z Instruction

C8 SEARCH EQ; A = B, INDEX LIST ➔ C

C9 SEARCH NE; A ≠ B, INDEX LIST ➔ C

CA SEARCH GE; A ≥ B, INDEX LIST ➔ C

CB SEARCH LT; A < B, INDEX LIST ➔ C



These instructions search and compare each element of vector field A with the successive elements of vector field B by subtracting vector B from vector A. The conditions for comparing floating-point operands are described in the Floating-Point Compare Rules, appendix B. The comparison and search of a given element of A with the elements of B, as specified by G designator bit 2, is defined as one search iteration. Each search iteration terminates when the condition specified by the instruction is found (A =, ≠, ≥, or < B) or when each element of B has been searched.

After each interation, the instruction clears the corresponding element of result vector C, if the control vector bit is permissive, and transfers to this element an item count of the number of elements of B that were searched without the specified condition being found (no hit). The item count does not include the hit condition if one is found. Regardless of the operand size (32- or 64-bit elements), the resulting item count is contained in the rightmost 48 bits of a 64-bit word. The leftmost 16 bits of each C vector element are cleared. If no element in the B vector causes a hit condition, the item count equals the field length of the B vector. The control vector controls the storing of the elements of vector C as specified by bit 1 of the G designator. The function of the control vector is described in Vector Instructions in this section.

These instructions use the floating point compare conditions as described in Branch Instructions in this section. The conditions specified by bits 0 and 1 of the G designator are shown in the previous instruction format. The conditions specified by bit 2 of the G designator are listed in table 6-45. The instruction format also shows that the X and Y designators and G bits 11 through 15 are not used and must be zeros. These instructions use no field lengths or offsets for vectors C and Z. Thus, the C + 1 designator is not used.

TABLE 6-45. SEARCH ITERATION STARTING DESIGNATOR CONDITIONS

G Bit 2	Conditions
0	Start at the beginning of vector B for each each element of vector A
1	Start at the location of the last hit in vector B for each element of vector A

These instructions terminate when each element of vector A has been compared with each element of vector B. The applicable data flag bit is 46 (indefinite result).

Figure 6-84 is an example of a search equal (C8) instruction with assumed instruction codes, register content, and vector fields. In the example, two search iterations compare the two elements of the A vector with the four elements of the B vector. The comparisons in the first iteration are represented by solid lines while those in the second iteration are indicated by dashed lines. Since bit 2 of the G designator is a zero for this case, each search iteration starts at the beginning of vector B. If the B vector becomes exhausted and G bit 2=1, all search iterations start and end with the end of the B vector. If the length of vector B is initially zero, all indexes stored are zero.

In the first iteration, three comparisons take place before the hit condition (A = B) is detected. As a result, an item count of three is entered into the first result element. No hit is detected in the second iteration; thus, the second result element equals the field length of the B vector (4). Since the two corresponding bits of the control vector are set, both result elements are stored.

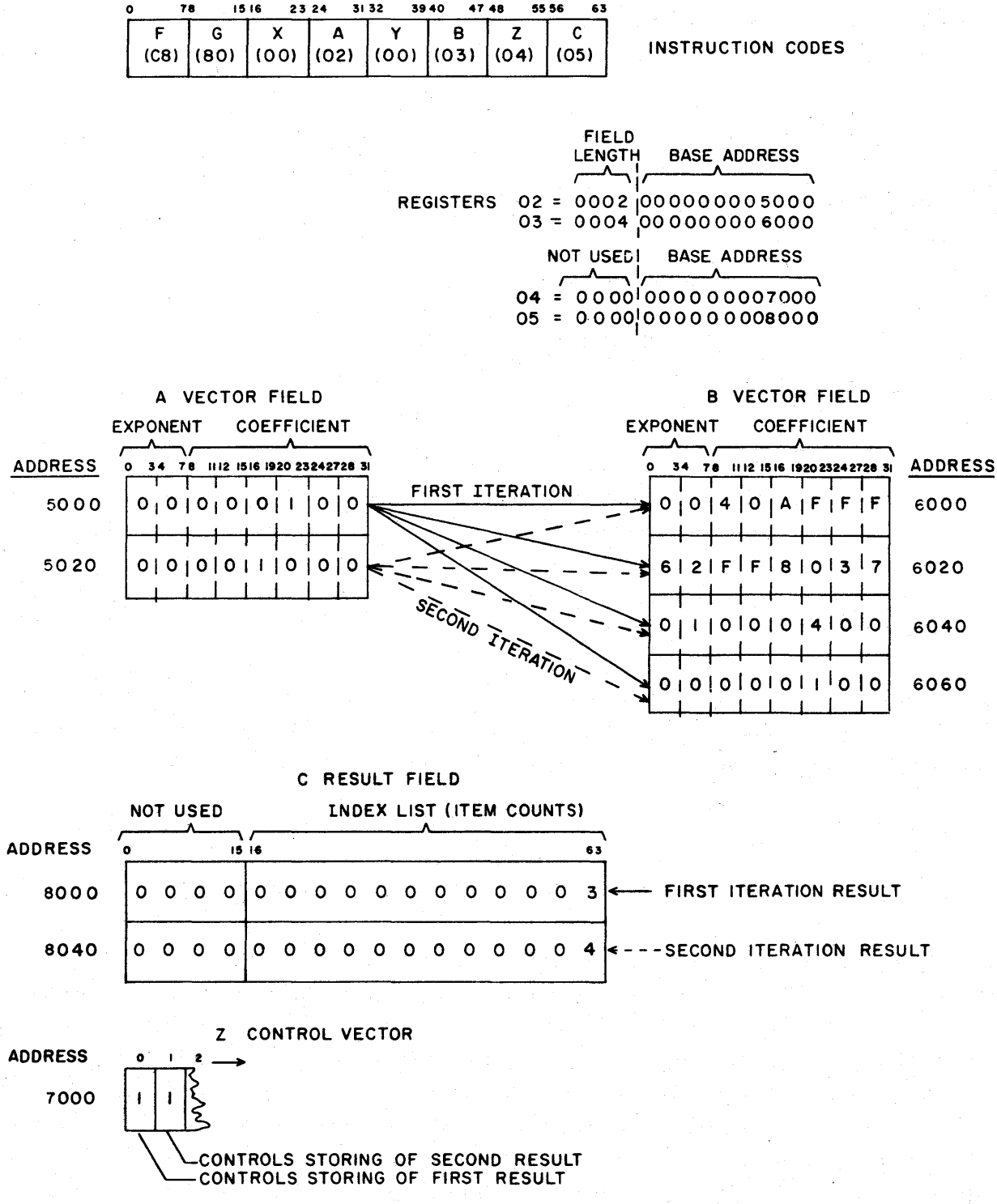
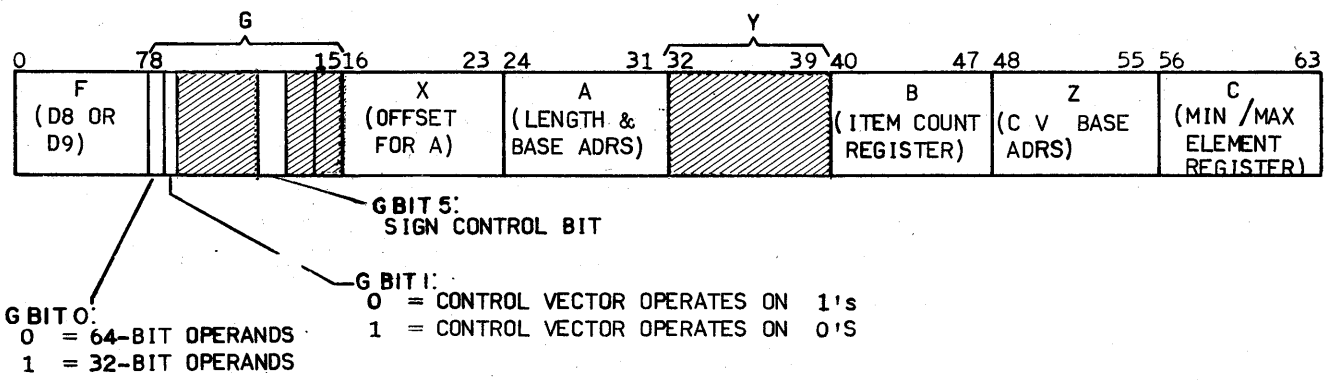


Figure 6-84. Example of Search EQ; A = B, Index List →C

D8 MAX. OF A TO (C) ITEM COUNT TO (B)
D9 MIN. OF A TO (C) ITEM COUNT TO (B)



These instructions search and compare the successive elements of vector A for the maximum/minimum element, using floating-point rules. The instructions then transmit the element to the register designated by C. The number of elements in vector A before, but not including, the maximum/minimum element is the item count which is stored in the rightmost 48 bits of a cleared register designated by B. The instructions terminate when vector A is exhausted.

If multiple maximum/minimum elements occur, the instruction sets data flag bit 54 and the first multiple maximum/minimum element examined is the one recorded. When this happens, the elements, although equal, are not necessarily identical.

If an indefinite element is encountered and examined, the register designated by C sets to indefinite and data flag bit 46 (indefinite result) sets. When this happens the content of the register designated by B and data flag bit 54 is undefined.

The Z designator of the instructions provides the base address for a control vector. If used, the control vector determines which of the vector A elements the instruction compares. This is possible by the association of individual control vector bits with single elements of vector A. Only permissive control vector bits permit compares for their associated vector A elements. If a control vector is used without any permissive elements, none of the content of the register designated by C is undefined. In this case the item count stored in the register designated by B is the length of the vector A minus the A offset. The instruction does not use an offset for the control vector.

Bit 0 of the G bits determines the size of the A operands and register C. Bit 5 of the G bits provides sign control. When bit 5 is set, the magnitude of the elements of A vector are compared. The unaltered element as read from A vector stores in the register designated by C.

Applicable data flag bits are 46 (indefinite result) and free data flag bit 54.

The instruction format shows that the Y designator and G bits 2 through 4, 6, and 7 of the G designator are not used and must be zeros. Bit 5 provides sign control for vector A as described in Vector Instructions. There is no B vector sign control for this instruction; thus, bit 7 of the G designator is undefined and must be a zero.

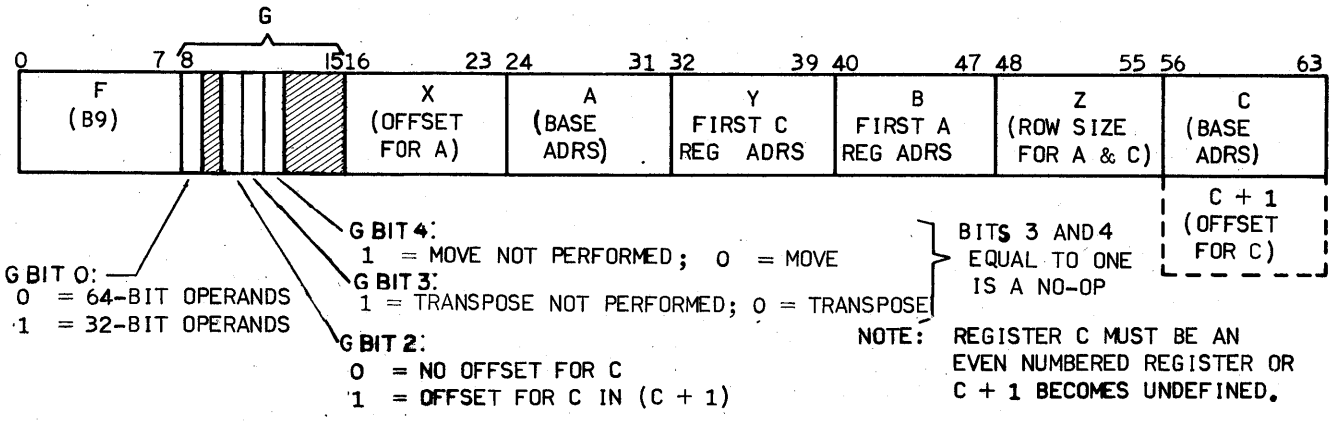
If the instruction specifies a control vector and the control vector contains no enabling bits, the instruction examines no elements of vector A, and the contents of register C becomes undefined. In this case, the item count in register B equals the field length of vector A minus the A offset.

If the instruction examines (enabling bit in control vector) an indefinite element, the instruction sets register C to indefinite and sets data flag bit 46 (indefinite result). In this case, data flag bit 54 is undefined. The instruction also probes the setting of data flag bit 43 (result machine zero).

The operands are compared by subtracting the current element of vector A from the next element of vector A and checking the result coefficient. If the result is not equal to zero, the maximum or minimum operand (depending upon the instruction) is used for the next compare with a new element of vector A. If the result is equal to zero, the most recent element of A is used for the next compare. The relative positions of the elements within the vector dictate the order of the subtract. Since this type of compare operation is order dependent,† the final maximum or minimum can be affected by the order of the elements within the vector.

†Appendix B describes floating point compare rules.

B9 TRANSPOSE/MOVE



This instruction transposes an 8 row by 8 column segment of matrix A and enters the transposed matrix into 64 consecutively numbered registers beginning at the register designated by B. The instruction then moves the matrix segment from 64 consecutively numbered registers beginning at the register specified by Y to matrix C. The register specified by Z contains the row size of matrices A and C. This row size is an item count contained in the rightmost 48 bits of register Z. The leftmost 16 bits are cleared. The instruction completes the transpose operation before the move operation begins. Thus, it is possible to return a transposed 8 by 8 matrix segment to its original location with a single instruction.

Matrix A must be located at consecutive storage locations. The base address in register A locates the first word of the first row. If an A offset is used, the instruction adds the rightmost 48 bits to the base address in register A to locate the first element of the first row. Successive elements of the first row are stored at consecutive storage locations.

The address of the first element of each of the following rows is the address of the first element of the previous row plus the row size (register Z). For example, the address of the first element in row 2 is the base address (register A) plus the row size (register Z). The address for the first element of row 3 is the address of the first element of row 2 plus the row size. Since the instruction transposes matrix segments of eight rows by eight columns, a row size of less than eight gives unpredictable results. If used, the register designated by C + 1 contains the C matrix offset in the rightmost 48 bits. The C designator must be an even number, or C + 1 becomes undefined.

The instruction uses no length specification or control vector. The instruction terminates when the last transposed segment is stored in result vector C.

Any transfer of words into or out of the register file that becomes exhausted of registers (that is, beyond the bounds of the register file) causes the instruction to become undefined.

Table 6-46 lists each of the instruction designators, the corresponding register length, and function of the contents.

TABLE 6-46. TRANSPOSE/MOVE INSTRUCTION DESIGNATORS

Register Designator	Register Length (Bits)	Function
A	64	Base address of matrix A
X	64	Item count (rightmost 48 bits) of the offset which locates the first element in the first row of the matrix segment read from matrix A
B	Either (operand size)	First of 64 consecutive registers used to hold the transposed segment of matrix A
C	64	Contains the base address of matrix C
C + 1	64	Item count (rightmost 48 bits) of the offset which locates the first element in the first row of the matrix segment to be stored into matrix C
Y	Either (operand size)	First of 64 consecutive registers used to hold the transposed segment of matrix C
Z	64	Contains (rightmost 48 bits) the item count of the row size for matrix A and matrix C

EXAMPLES OF TRANSPOSE/MOVE INSTRUCTION

Figure 6-85 is an example of an assumed 10 row by 10 column matrix. The matrix segment to be transposed is outlined by heavy lines.

For clarity of illustration, consecutive decimal numbers represent the elements of the matrix which, in the case of this example, would be 32-bit operands. In this example, the outlined 8 by 8 segment is transposed (row exchanged for column and column exchanged for row) and is restored in the same matrix shown in figure 6-86. Rows 8 and 9 and columns 8 and 9 are not affected by the transpose operation since they are outside the outlined segment.

ROWS \ COLUMNS	0	1	2	3	4	5	6	7	8	9
0	01	11	21	31	41	51	61	71	09	10
1	02	12	22	32	42	52	62	72	19	20
2	03	13	23	33	43	53	63	73	29	30
3	04	14	24	34	44	54	64	74	39	40
4	05	15	25	35	45	55	65	75	49	50
5	06	16	26	36	46	56	66	76	59	60
6	07	17	27	37	47	57	67	77	69	70
7	08	18	28	38	48	58	68	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

NOTE

8 X 8 SEGMENT OUTLINED WITH HEAVY LINES.

Figure 6-85. Example of Initial 10 x 10 Matrix

ROWS \ COLUMNS	0	1	2	3	4	5	6	7	8	9
0	01	02	03	04	05	06	07	08	09	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

Figure 6-86. Example of Transposed 8 x 8 Segment in 10 x 10 Matrix

Figure 6-87 is an example of the transpose/move instruction codes used to perform the transposition of the 8 by 8 segment of the matrix shown in Figures 6-83 and 6-84. No offsets are specified for the matrices.

Since the segment is transposed and returned to the original matrix, the A and C designators are equal. Thus, the base address for both the A and C matrices is 5000_{16}. In a similar manner, the Y and B designators are equal. Thus, the first register address in each case is 06_{16}.

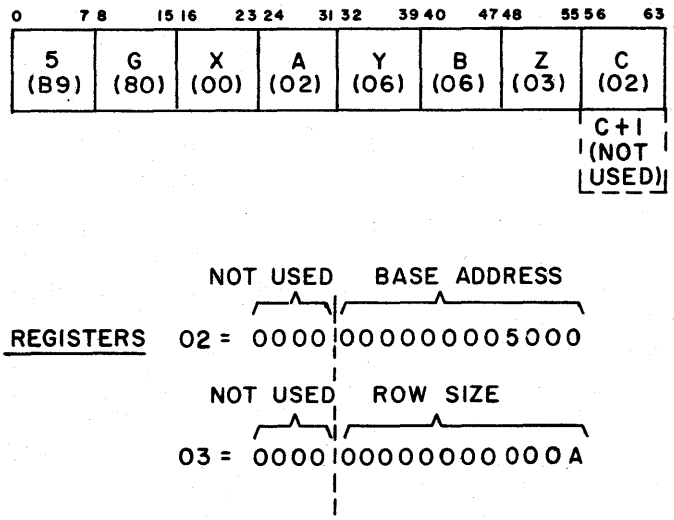


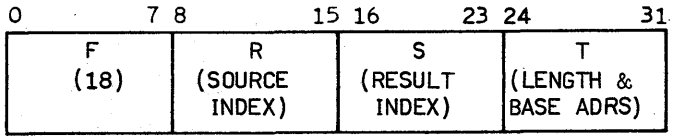
Figure 6-87. Example of Transpose/Move Instruction Codes

Table 6-47 lists the storage and register address mapping for the example.

TABLE 6-47. EXAMPLE OF STORAGE AND REGISTER MAPPING FOR TRANSPOSE/MOVE INSTRUCTION

	Matrix A (Initial)		Matrix A (Transposed Segment)			Matrix C (Result)		
Row No.	Storage Address	Content	Row No.	Register Address	Contents	Row No.	Storage Address	Content
0	5000_{16}	01	0	06_{16}	01	0	5000_{16}	01
	5020	02		07	11		5020	11
	5040	03		08	21		5040	21
	5060	04		09	31		5060	31
	5080	05		0A	41		5080	41
	50A0	06		0B	51		50A0	51
	50C0	07		0C	61		50C0	61
	50E0	08		0D	71		50E0	71
	5100	09	1	0E	02		5100	09
	5120	10		0F	12		5120	10
1	5140	11		10	22	1	5140	02
	5160	12		11	32		5160	12
	5180	13		12	42		5180	22
	51A0	14		13	52		51A0	32
	51C0	15		14	62		51C0	42
	51E0	16		15	72		51E0	52
	5200	17	2	16	03		5200	62
	5220	18		17	13		5220	72
	5240	19		18	23		5240	19
	5260	20		19	33		5260	20
9	5C20	98	7	43	76	9	5C20	98
	5C40	99		44	77		5C40	99
	5C60	100		45	78		5C60	100

18 MOVE BYTES RIGHT



This instruction moves source field T starting with the rightmost byte and terminating with the leftmost byte. The register designated by T contains the field length and base address of field T in the leftmost 16 bits and rightmost 48 bits, respectively. The rightmost 48 bits of registers R and S contain signed, two's complement indexes. The R and S indexes are item counts in bytes and offset the source and result fields, respectively. The instruction left-shifts these indexes three positions before adding them to the base address.

The instruction determines the address of the first byte of the source field by adding the T length and R index to the T base address and subtracting one (byte) from the sum. The address of the first byte of the result field is found by adding the T length, the R index, and the S index and then subtracting one byte. The instruction then moves the first source byte to the first result field address. The instruction continues to move successive source bytes to consecutive result field byte addresses until the result field is filled. The length of the result field equals the length of the source field (T length). If the T length is zero, the instruction functions as a no-op.

EXAMPLES OF MOVE BYTES RIGHT OPERATIONS

Figure 6-88 shows an example of a move bytes right operation with a positive S index. In this example, the T base address is 5000_{16} and the R index is set to two. The bytes are numbered in the order in which they are moved. The address of the first source byte becomes 1048_{16} ($T_{BA} + T_L + R_I - 1 = 5000 + 8 + 2 - 1 = 5048_{16}$).

where:
- T_{BA} = T - base address
- T_L = T length
- R_I = R index
- S_I = S index

T_L, R_I, and 1 (byte) are shifted left three positions before the addition. The instruction determines the address of the first result byte in a similar manner except that the S index is added to the preceding sum. A positive S index may be less than the T length since the rightmost byte of the source field is moved first.

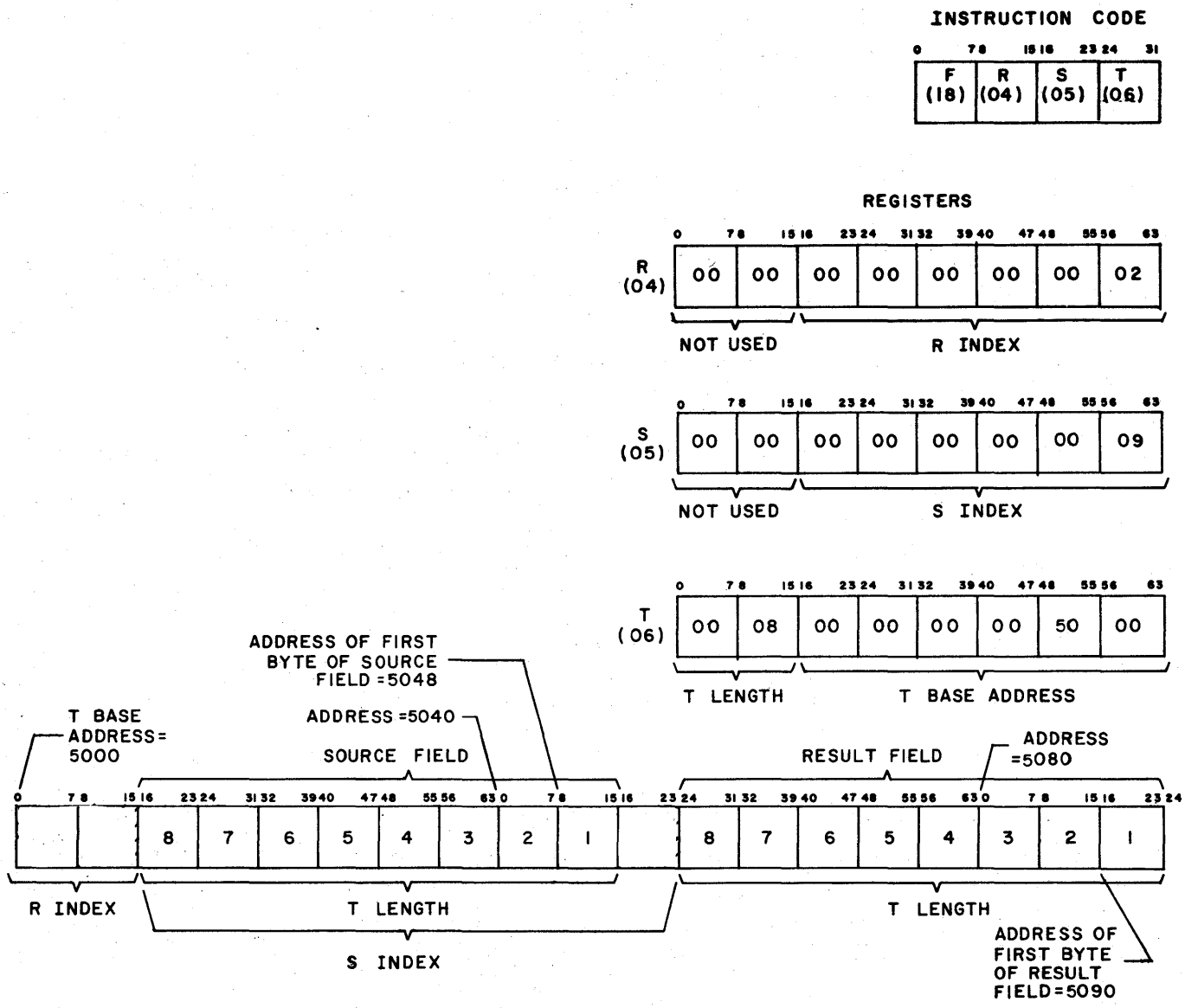


Figure 6-88. Example of a Move Bytes Right Instruction with a Positive S Index

Figure 6-89 is an example of a move bytes right instruction with a negative S index. The negative S index causes the instruction to move the source field left. In this example, the T length remains at eight, but the T base address is now 5048_{16}. Thus, the address of the first byte in the source field becomes $T_{BA} + T_L + R_I - 1 = 5090_{16}$.

Since S index is B_{16}, the address of the first byte in the result field becomes $T_{BA} + T_L + R_I + S_I - 1 = 5048 + 8 + 2 - B - 1 = 5038_{16}$. With a negative index, an overlap of result and source field causes the instruction results to become undefined. For example, if the S index is set to -7 in Figure 6-87, the address of the first result byte would be $5048 + 8 + 2 - 7 - 1 = 5058_{16}$. Thus, the first source byte would be stored in the eighth source byte position, producing undefined results.

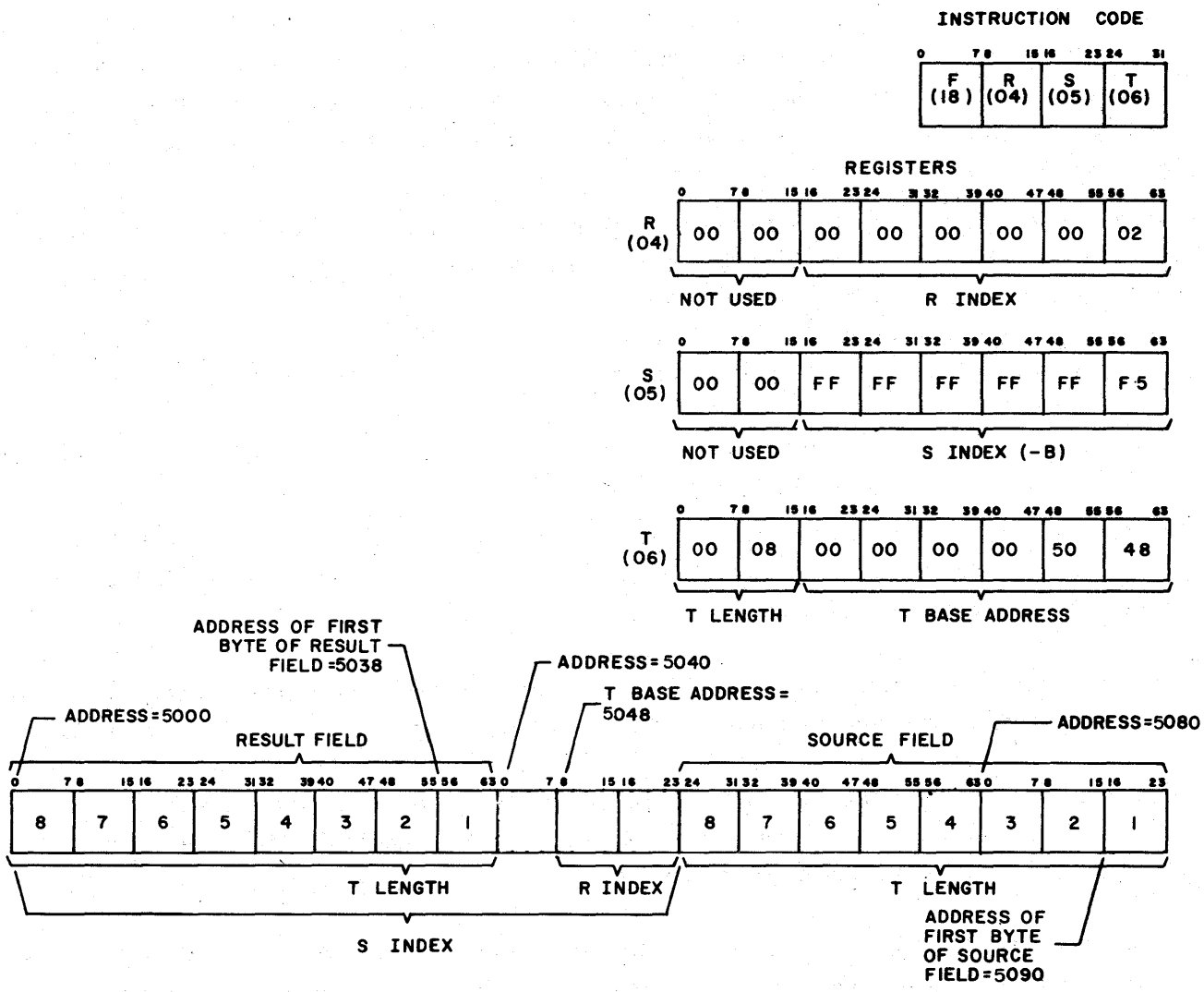
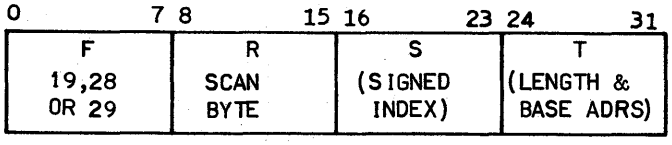


Figure 6-89. Example of a Move Bytes Right Instruction with a Negative S Index

19 SCAN RIGHT

28/29 SCAN EQUAL/UNEQUAL



19 SCAN RIGHT

This instruction (figure 6-90) scans the bytes in source field T, from right to left, until the scan operation locates the first byte not equal to byte R, contained in the instruction word. The scan operation is indexed by the signed scan index, contained in the rightmost 48 bits of the register denoted by S. When the operation locates the first unequal byte, the instruction stops the scanning and decrements the scan index by the number of bytes scanned before the unequal byte was found.

The register specified by T contains the field length and base address of the source field in the leftmost 16 bits and rightmost 48 bits, respectively. The address of the first byte read from the source field is determined as follows:

$T_{BA} + T_L + S_I - 1$ (byte) = SA

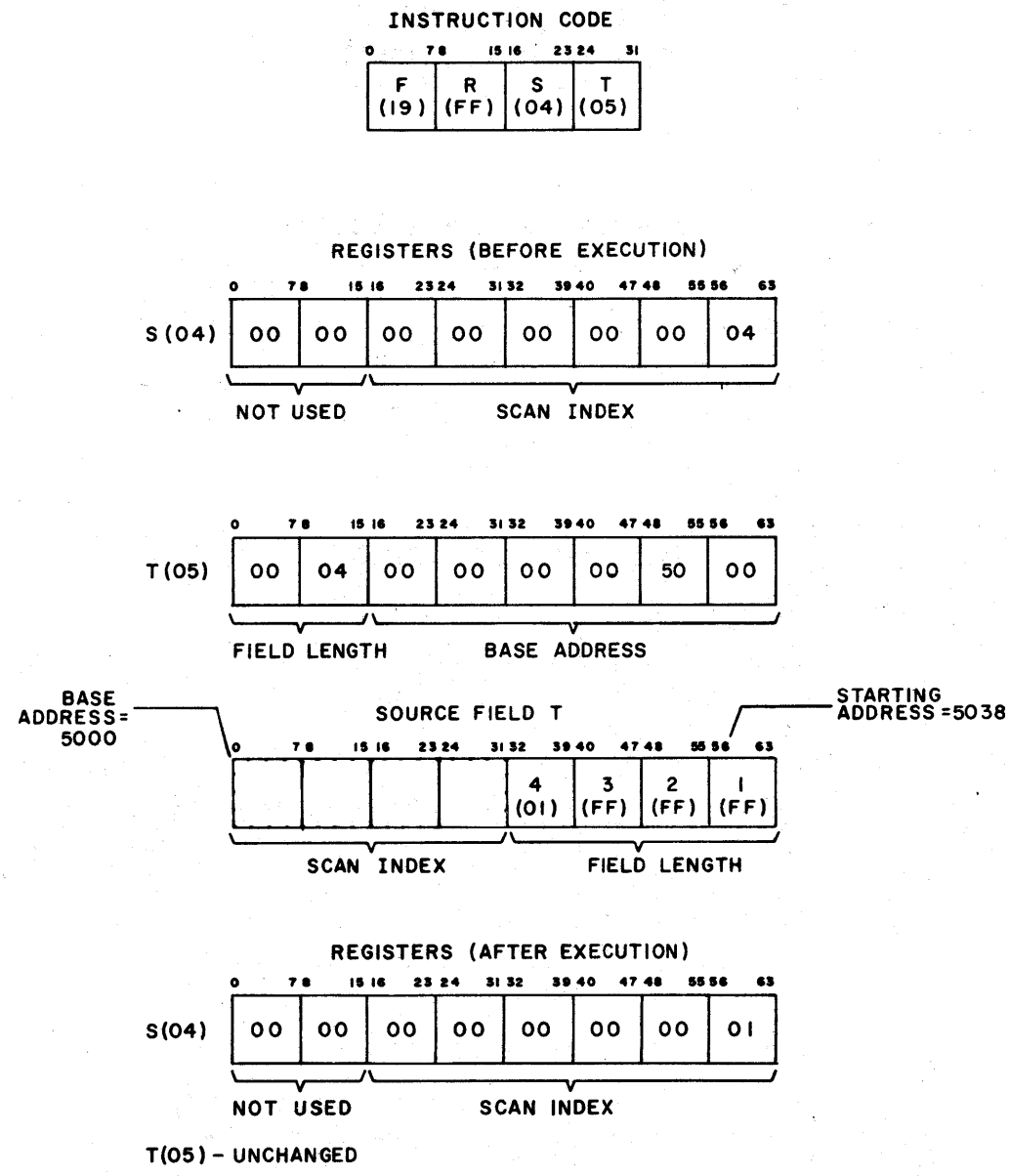
where:
- T_{BA} = T base address
- T_L = T length
- S_I = scan index
- SA = starting address

In figure 6-90, the starting address becomes $SA = T_{BA} + T_L + S_I - 1 = 5000 + 4 + 4 - 1 = 5038_{16}$. Since T_L and S_I are item counts in bytes, these values are left-shifted three places before the addition.

The instruction sets data flag bit 53 if no unequal byte is found in the source field. In this case, the instruction terminates when the entire source field length is scanned.

Figure 6-90 is an example of a scan right instruction with a postive scan index. In this case, three equal bytes are scanned before the first unequal byte is detected. Thus, the scan index is decremented by three, giving a final value of +1.

Figure 6-91 is an example of a scan right instruction with a negative scan index. The same instruction codes and T register values are used as in Figure 6-90, however, in this case, the scan index is set to a -7. Thus, the starting address becomes $SA = T_{BA} + T_L + S_I - 1 = 5000 + 4 - 7 - 1 = 4FE0_{16}$. Since three equal bytes are again scanned before the unequal byte is detected, the final scan index is $(-7-3) = -A_{16}$.



NOTES: IN SOURCE FIELD T

① NUMBERS NOT IN PARENTHESES DENOTE ORDER OF BYTES SCANNED.

② NUMBERS IN PARENTHESES DENOTE BYTE VALUES.

Figure 6-90. Example of Scan Right Instruction with a Positive Scan Index

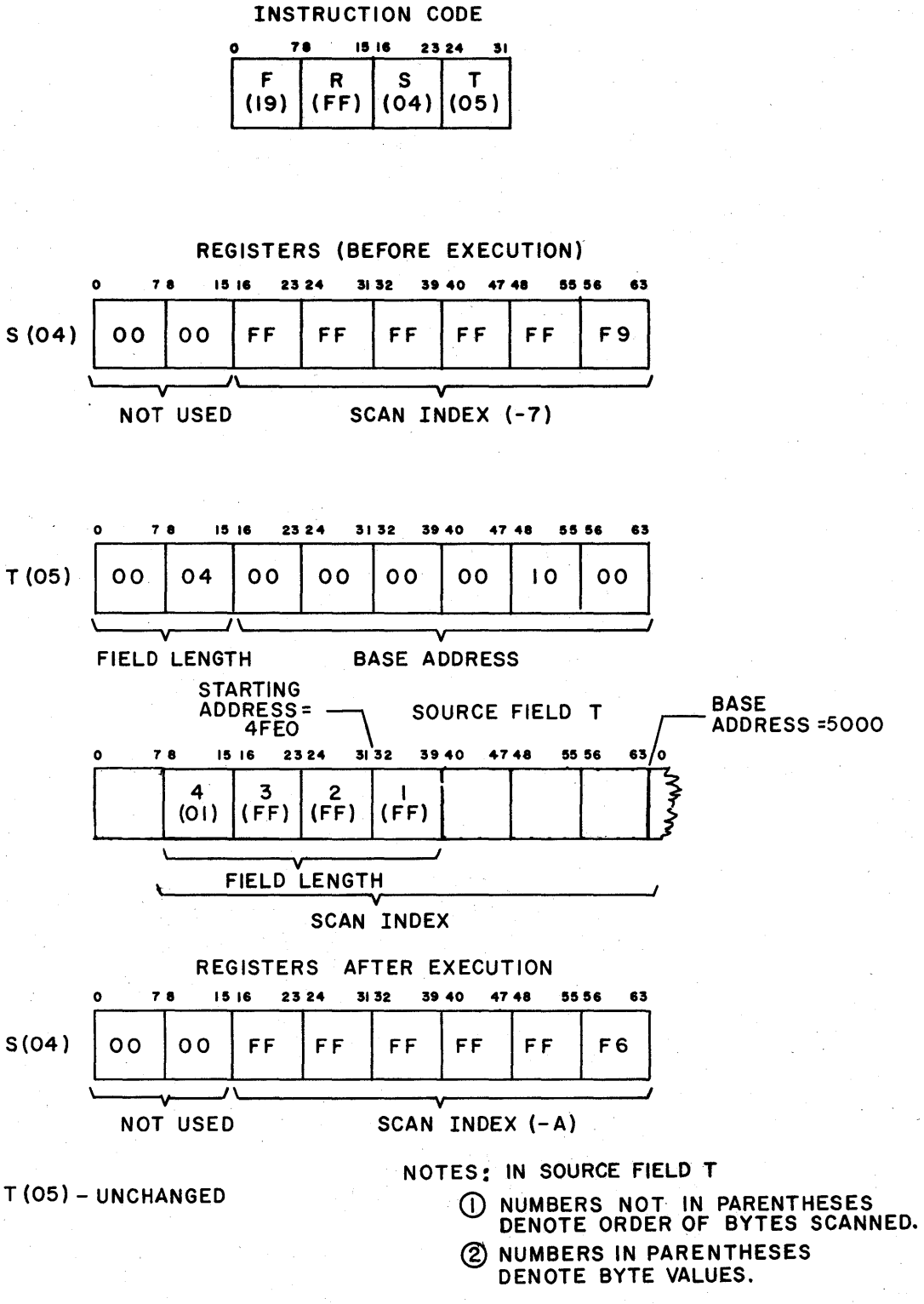


Figure 6-91. Example of Scan Right Instruction with a Negative Scan Index

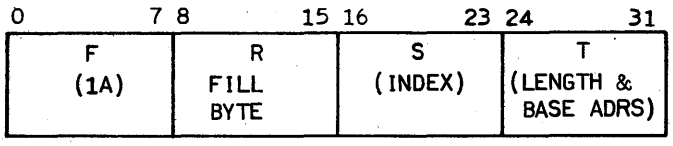
28/29 SCAN EQUAL/UNEQUAL

These instructions scan the bytes in field T, from left to right, until the scan operation locates the first byte equal/unequal to byte R, contained in the instruction word. The scan operation is indexed by the signed scan index, located in the rightmost 48 bits of the register denoted by S. When the operation locates the first equal/unequal byte, the instruction stops scanning and increments the scan index (S) by the number of bytes scanned before the equal/unequal byte was found.

The register specified by T contains the field length and base address of the source field in the leftmost 16 bits and rightmost 48 bits, respectively. Since the T field length and S index are item counts in bytes, they are left-shifted three places before they are added to the base address.

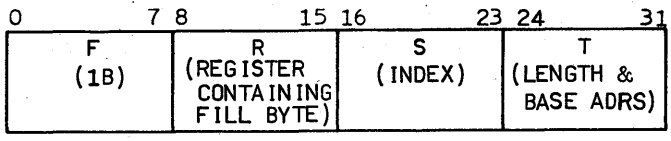
The instruction sets data flag bit 53 if no equal/unequal byte is found, and the S index is incremented by the number of bytes in the T field. In this case, the instruction terminates when the entire source field is scanned.

1A FILL FIELD T WITH BYTE R



This instruction fills field T, from left to right, with bytes identical to the R portion of the instruction word. The register designated by T contains field length (number of bytes) and base address in the leftmost 16 and rightmost 48 bits, respectively. Register S contains an index. The instruction adds the index to the base address (after left-shifting three positions). The resulting sum is the starting address of the T field. The instruction terminates when the T field is filled.

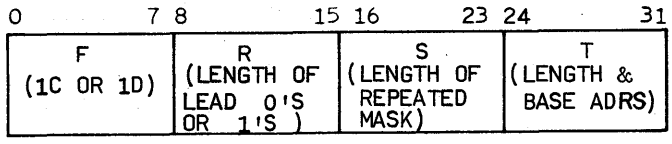
1B FILL FIELD T WITH BYTE (R)



This instruction fills field T, from left to right, with bytes identical to the byte contained in the rightmost eight bits of the register designated by R. Bits 0 through 55 of register R are not used. The register designated by T contains the field length (number of bytes) and base address in the leftmost 16 and rightmost 48 bits, respectively. Register S contains an index in bytes which is added to the base address (after left-shifting three places). The resulting sum is the starting address of the T field. The instruction terminates when the T field is filled.

1C FORM REPEATED BIT MASK WITH LEADING ZEROS

1D FORM REPEATED BIT MASK WITH LEADING ONES



These left to right instructions form a repeated mask in field T. The mask consists of a string of zeros/ones followed by a string of ones/zeros. The repeated mask consists of one combined string of zeros and ones or ones and zeros as shown in figure 6-92. All length specifications shown in figure 6-92 are in bits.

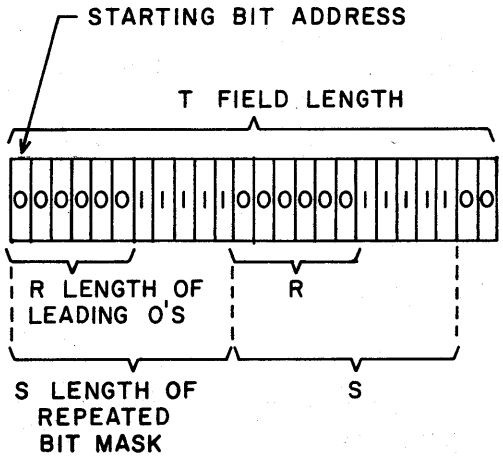
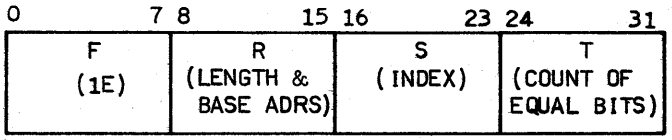


Figure 6-92. Example of Repeated Bit Mask Data Format (Leading Zeros)

The register specified by R (instruction format) contains the length of the string of zeros/ones in the leftmost 16 bits. The length of the repeated mask is contained in the leftmost 16 bits of register S. The rightmost 48 bits of registers R and S are undefined and require clearing before execution of the instruction. If the field length specified by the S register is zero, the instruction becomes a no-op. The register specified by T contains the length and starting bit address of the T field in the leftmost 16 bits and rightmost 48 bits, respectively. The instruction terminates when the T field is filled. If length R is equal to length S, a string of zeros (1C) or ones (1D) is formed. If length R is zero, a string of ones/zeros is formed.

1E COUNT LEADING EQUALS



This instruction scans the bits in field R, from left to right, until a bit unequal to the leftmost bit in the field is detected. The scanning operation starts with the bit immediately to the right of the leftmost bit in the field (figure 6-93). The instruction stores the count of the number of bits equal to the leftmost bit of the binary field in the rightmost bits of the register designated by T. The entire T register is cleared before the count is stored into it.

The register designated by R contains the length (in bits) and the base address in the leftmost 16 bits and rightmost 48 bits, respectively. Register S contains an index (in bits) which is added to the base address to form the starting address of the field. The instruction terminates when it either detects a bit unequal to the leftmost bit in the field or scans the entire

field. In the latter case, the instruction stores a count equal to the field length minus one. In figure 6-93, a count of B_{16} is stored in register T.

The instruction sets data flag bit 53 if the leftmost bit of the binary field is a one.

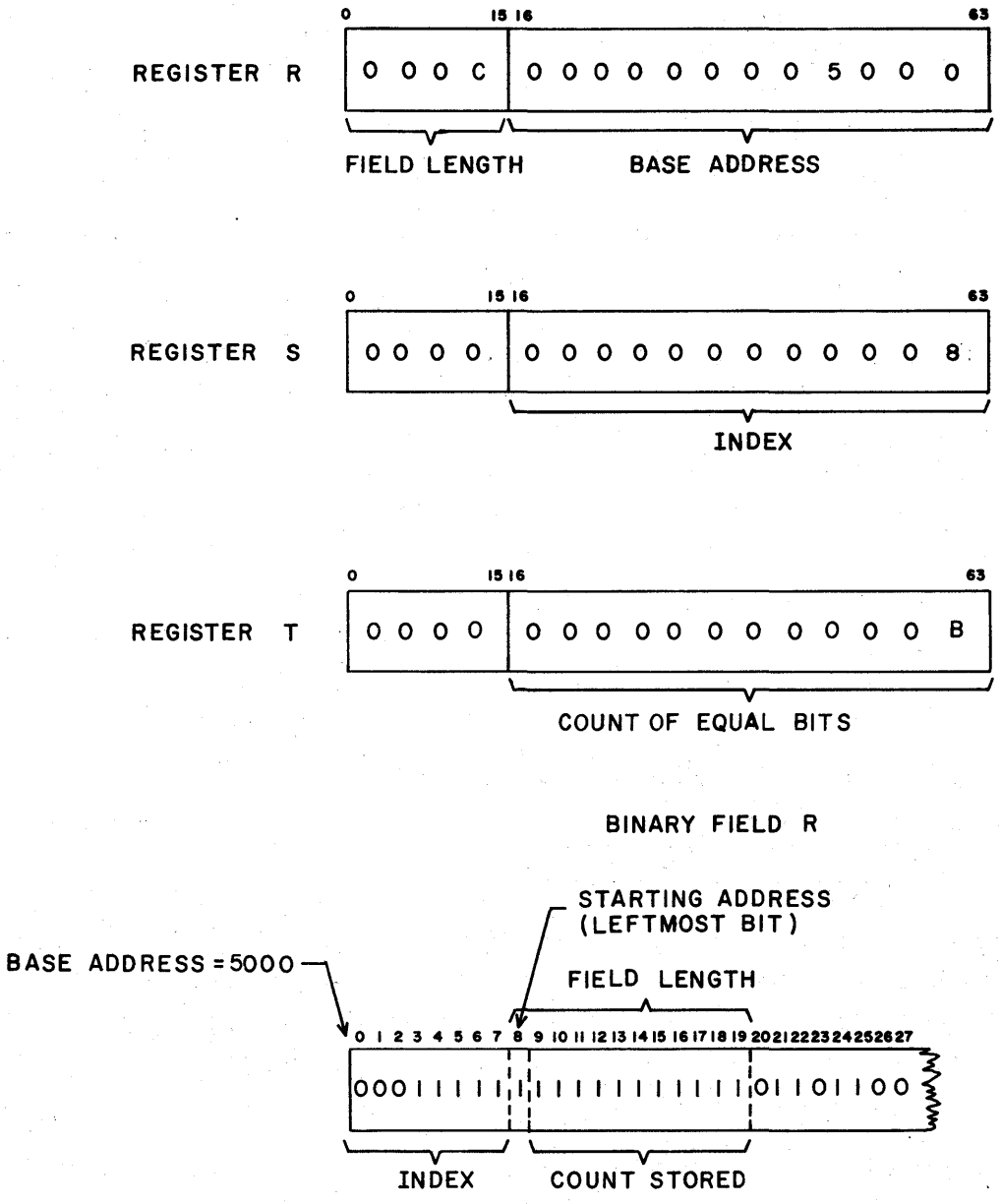
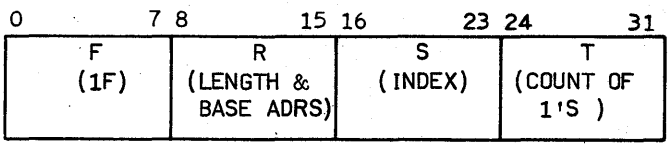


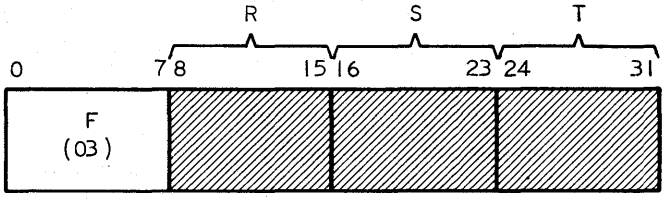
Figure 6-93. Example of Count Leading Equals Data and Register Format

IF COUNT ONES IN FIELD R, COUNT TO T



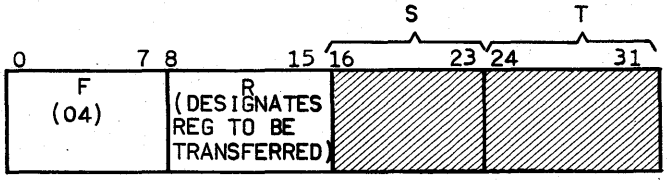
This instruction scans left to right, counts the number of binary ones in field R, and transmits this count to the rightmost bits of the register specified by T. The entire T register is cleared before receiving the count of ones. The register specified by R contains the length and base address of the R field in the leftmost 16 and rightmost 48 bits, respectively. The rightmost 48 bits of register S contain an index (in bits), which the instruction adds to the base address to form the starting address of the R field. The instruction terminates when all bits in the R field have been scanned.

03 KEYPOINT — MAINTENANCE



This instruction is a one-cycle no-operation instruction. The designators R, S, and T are undefined and must be set to zero.

04 BREAKPOINT — MAINTENANCE



The breakpoint instruction is a special instruction reserved as a maintenance and program debugging aid. This instruction transfers the content of the 64-bit register designated by R into the breakpoint register. The format of the breakpoint register is shown in figure 6-94. The breakpoint register is initially loaded from the invisible package of a job and is not sensed on any scalar memory references.

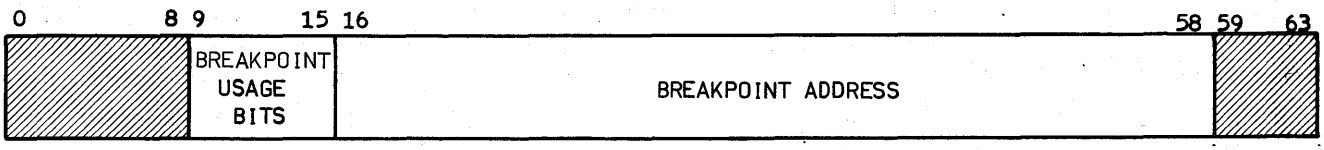


Figure 6-94. Breakpoint Register Format

The breakpoint address is compared with the addresses listed in table 6-48. If the breakpoint address matches one of these addresses and the proper usage bit is set, bit 47 of the data flag branch register is set, indicating a breakpoint condition. Any combination of usage bits is permissible. Therefore, the breakpoint address can be checked against any or all of the addresses listed in the table.

Space table search or I/O channel references cannot cause a breakpoint match condition. Clearing the R designator bits to zeros and executing the instruction causes the instruction to stop.

In job mode, virtual addresses are compared with breakpoint, and in monitor mode, absolute addresses are compared with breakpoint. Since the monitor program does not have an invisible package, the breakpoint register must be loaded each time the monitor program is entered. During the exchange to monitor mode, the breakpoint register is automatically cleared. Program address compares are made on half-word boundaries, and all other compares are made on sword boundaries.

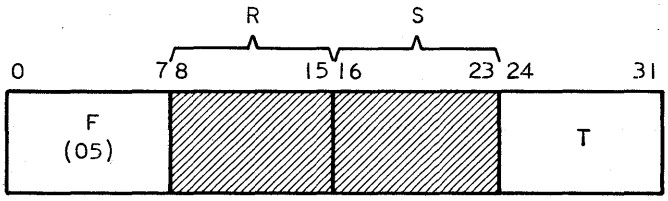
TABLE 6-48. BREAKPOINT CONDITIONS

Usage Bit No.	Breakpoint Condition
9	Breakpoint on half-word content of the program address register (P) immediately after the execution of the instruction at that location
10	Breakpoint on the A operand address for a vector or the read operand on a random addressing instruction
11	Breakpoint on the B operand address for a vector instruction
12	Breakpoint on the C operand address for a vector or string instruction, or the write operand on a random addressing instruction
13	Breakpoint on the Z control vector or operand address for a vector or string instruction
14	Breakpoint on the X order vector address or operand address for a string instruction
15	Breakpoint on the Y order vector address or operand address for a string instruction

NOTE

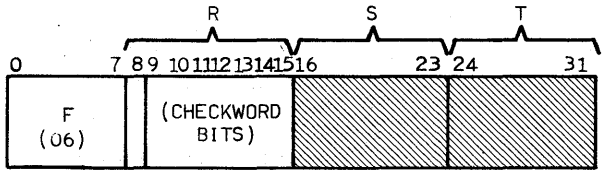
The breakpoints occur just after execution of the instruction at the breakpoint address.

05 VOID STACK AND BRANCH



This instruction voids the instruction stack and branches out-of-stack to the contents of the register designated by T. The designators R and S are undefined and must be set to zero.

06 FAULT TEST — MAINTENANCE



This instruction is used to complement checkword bits on the scalar write bus so the read SECDED circuitry may be checked. It can also be used to disable the error correction circuitry on all read buses allowing data to be passed through the SECDED hardware without any correction taking place.

This instruction is enabled during monitor mode only; in job mode it is a no-op.

The modes are set by executing the instruction with 1 in the appropriate R designator bit and cleared by a 0 in the same bit location. Table 6-49 shows the R designator bit definitions.

TABLE 6-49. R DESIGNATOR BIT DEFINITIONS

R Designator Bit	Definition
8	Disable error correction on all read buses.
9-15	Checkword bit to be complemented.

The R designator bits must be set to zero before any monitor to job exchange operation. If these bits are not set to zero via this instruction, the connection network could produce invalid data on the read and invalid data could be written into memory.

SECDED FAULT TESTING

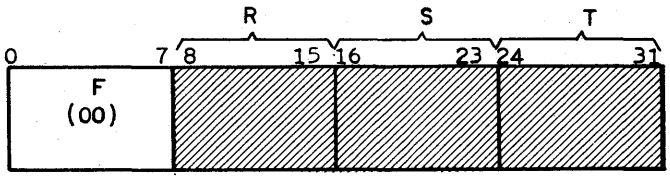
In this test, R designator bits 9 through 15 are selected to complement the respective checkword bits of halfwords 0, 1, 2, and 3 on the write scalar bus to central memory. By selection of data bits and complementing checkword bits, SECDED fault generation on all read buses is possible allowing complete checking of the read SECDED hardware and the fault recording hardware for type and address of fault.

The forced complementing of the checkword bits is discontinued by executing the instruction with bit 9 through 15 of the R designator set to 0.

MONITOR INSTRUCTIONS

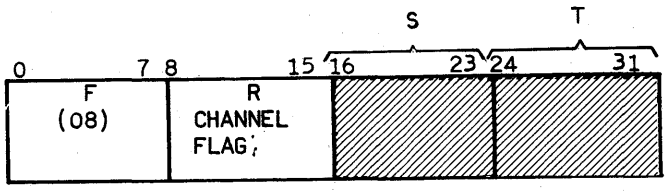
The monitor instructions function only during the monitor mode of operation. When the machine is in the job mode, the attempt to execute a monitor instruction is detected in the same way as an attempt to execute an undefined instruction code. The result of such an attempt is that the function code (F) and virtual program address (P) of the current instruction are stored in the appropriate positions of the invisible package. The machine then exchanges to the monitor program starting at the address contained in register 03. Refer to section 5 for a more detailed description of job to monitor exchange operations.

00 IDLE



If in the monitor mode, this instruction enables the external interrupt and halts program operation until an external interrupt occurs. The R, S, and T designators are not used and must be zeros.

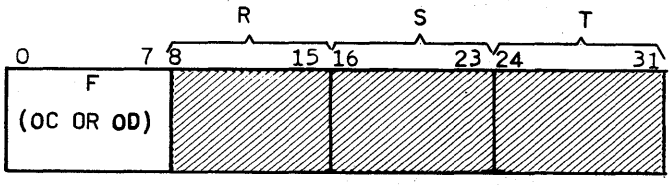
08 INPUT/OUTPUT PER R



In the monitor mode, this instruction sets the channel flag bit in the I/O channel designated by the hexadecimal code in the designator R. The setting of this bit indicates that the CPU has stored data at a predetermined location in central storage for the designated channel. The corresponding I/O channel then processes the stored data. If the R designator specifies a non-existent channel other than I/O 1 through 12, the instruction becomes undefined. The S and T designators are not used and must be zeros.

0C STORE ASSOCIATIVE REGISTERS

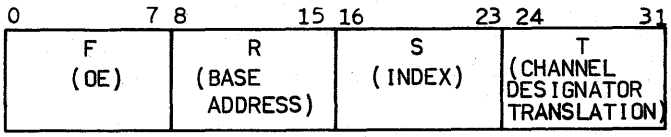
0D LOAD ASSOCIATIVE REGISTERS



These instructions store (0C)/load (0D) the contents of the 16 associative registers into/from consecutive absolute addresses of central storage beginning at 4000_{16}. The transfer is an ordered operation; thus, associative register 0 transfers to/from address 4000_{16}. The contents of associative register 1 transfers to/from address 4040_{16}, etc. The content of the associative registers are undefined following the execution of the 0C instruction.

The R, S, and T designators are not used and must be zeros.

0E TRANSLATE EXTERNAL INTERRUPT



This instruction translates the lowest numbered bit set in the external interrupt register (EIR) into its associated, 4-bit code and transmits the code to the rightmost four bits of the register designated by T. The leftmost 60 bits of register T are cleared to zeros. If only one bit in EIR is set, the program branches to the address formed by the sum of the content of the registers designated by S and R. The rightmost 48 bits of register S contain an index in half-words and the rightmost 48 bits of register R contain the base address. If more than one bit in EIR is set, the program executes the next instruction.

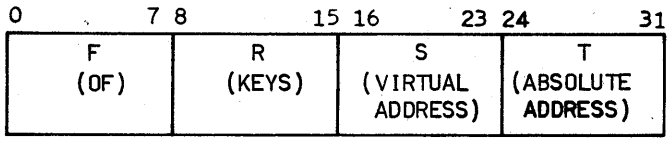
Whether the branch condition is met or not, the instruction clears the EIR bit corresponding to the channel designator that was transmitted to register T. If the T and S designators are equal, the interrupting channel designator is the branch index.

If no bit in EIR is set, the instruction clears register T and performs no branch operation. Bit zero of EIR is never set since this bit is reserved for maintenance purposes.

Each bit in the EIR is associated with one of the I/O channels or the monitor interval timer. The EIR bit assignments are as follows:

Bits	Assignments
0	Not available
1-12	I/O channels 1 through 12
13	Not assigned
14	Not assigned
15	Monitor interval timer

0F LOAD KEYS FROM (R), TRANSLATE ADDRESS (S) TO (T)



This instruction loads the four keys found in the register designated by R into the virtual address key registers. The instruction then translates the virtual address† located in the rightmost 48 bits of register S into an absolute bit address, using the four keys loaded from R and the associative words from the page table. The resulting absolute bit address is transmitted to the rightmost 48 bits of the register designated by T. If no translation is possible before the end of the page table is reached, the instruction clears the rightmost 48 bits of register T. The leftmost 16 bits of register S are transmitted to the corresponding portion of register T. The associative word used to make the translation is placed at the top of the page table (associative register 0). The instruction moves the position of the associative words down in the page table, if necessary, when searching for the associative word used to make the translation. The 3-bit usage code in the associative word is not altered by this instruction. Figure 6-95 shows the formats for the R, S, and T registers as they are used for this instruction.

† Virtual addressing operation is described in section 4.

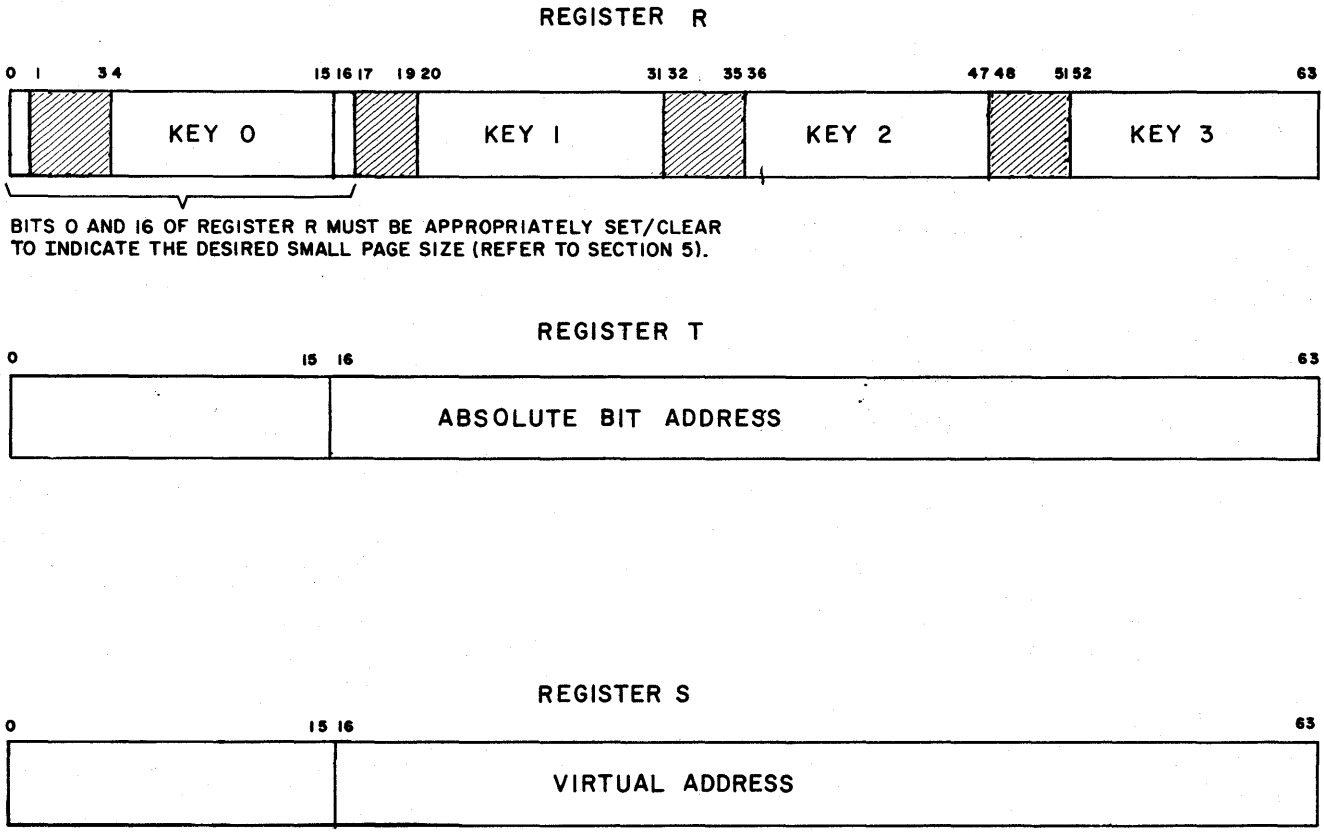
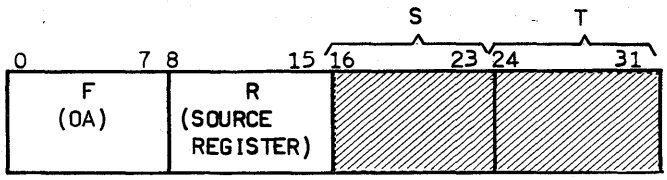


Figure 6-95. Register Formats for the 0F Instruction

0A TRANSMIT (R) TO MONITOR INTERVAL TIMER



In the monitor mode, this instruction transmits bits 40 through 63 of the 64-bit register specified by the R designator to the monitor interval timer. The function of the monitor interval timer is described in section 5. The leftmost 40 bits of register R are ignored.

NUMBER SYSTEMS AND TABLES A

GENERAL

Any number system may be defined by the radix or base. The radix or base is the number of unique symbols used in the system. The decimal system has ten symbols, 0 through 9. Modulus is the number of unique quantities or magnitudes a given device can distinguish. For example, an adding machine with 10 digits, or counting wheels, has a modulus of $10^{10}-1$. The adding machine has a modulus because the highest number which this machine can express is 9,999,999,999.

Most number systems are positional; that is, the relative position of a symbol determines its magnitude. In the decimal system, a 5 in the units column represents a different quantity than a 5 in the 10's column. Quantities equal to or greater than 1 may be represented by using the 10 symbols as coefficients of ascending powers of the base 10. The number 984_{10} becomes:

$$\begin{aligned} 9 \times 10^2 &= 9 \times 100 = 900 \\ +8 \times 10^1 &= 8 \times 10 = 80 \\ +4 \times 10^0 &= 4 \times 1 = 4 \\ &\qquad\qquad 984_{10} \end{aligned}$$

Quantities less than 1 may be represented by using the 10 symbols as coefficients of ascending negative powers of the base 10. The number 0.593_{10} may be represented as:

$$\begin{aligned} 5 \times 10^{-1} &= 5 \times .1 = .5 \\ 9 \times 10^{-2} &= 9 \times .01 = .09 \\ 3 \times 10^{-3} &= 3 \times .001 = .003 \\ &\qquad\qquad .593_{10} \end{aligned}$$

BINARY NUMBER SYSTEM

Internal operations in the computer use the binary number system. This system uses two symbols, 0 and 1; the base is 2. Because of the two-state characteristics, the binary system lends itself well to representation by the electronic switching circuits in the computer. The following numbers show the positional value of the binary number system:

$$\ldots\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

32 16 8 4 2 1 Binary point

The binary number 011010 represents:

$$
\begin{aligned}
0 \times 2^5 &= 0 \times 32 = 0 \\
+1 \times 2^4 &= 1 \times 16 = 16 \\
+1 \times 2^3 &= 1 \times 8 = 8 \\
+0 \times 2^2 &= 0 \times 4 = 0 \\
+1 \times 2^1 &= 1 \times 2 = 2 \\
+0 \times 2^0 &= 0 \times 1 = 0 \\
&\qquad\qquad\qquad\quad 26_{10}
\end{aligned}
$$

Fractional binary numbers may be represented by using the symbols as coefficients of ascending negative powers of the base.

	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	---
Binary point	1/2	1/4	1/8	1/16	1/32	

The binary number 0.10110 may be represented as:

$$
\begin{aligned}
1 \times 2^{-1} &= 1 \times 1/2 = 1/2 = 8/16 \\
0 \times 2^{-2} &= 0 \times 1/4 = 0 = 0 \\
1 \times 2^{-3} &= 1 \times 1/8 = 1/8 = 2/16 \\
1 \times 2^{-4} &= 1 \times 1/16 = 1/16 = 1/16 \\
0 \times 2^{-5} &= 0 \times 1/32 = 0 = 0 \\
&\qquad\qquad\qquad\quad 11/16_{10}
\end{aligned}
$$

HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system uses 16 discrete symbols (base 16). Table A-1 shows the 16 hexadecimal symbols with the decimal and binary equivalents. Note that the first 10 hexadecimal symbols are identical to the corresponding decimal symbols. The remaining six symbols are represented by alphabetical characters A-F.

NOTE

To avoid confusion between hexadecimal and decimal numbers in the instruction manuals, all numbers shown without the base number affixed are hexadecimal numbers. Decimal numbers are shown with the base designator 10 affixed in the conventional manner. For example, the number 79847 represents a hexadecimal number. Conversely, 79847_{10} represents a decimal number.

With base 16, the positional value of hexadecimal numbers is:

16^5	16^4	16^3	16^2	16^1	16^0
$1,048,576_{10}$	$65,536_{10}$	$4,096_{10}$	256_{10}	16_{10}	1

The hexadecimal number 859F is:

$$8 \times 16^3 = 8 \times 4,096_{10} = 32,768_{10}$$
$$5 \times 16^2 = 5 \times 256_{10} = 1,280_{10}$$
$$9 \times 16^1 = 9 \times 16_{10} = 144_{10}$$
$$F \times 16^0 = F\dagger \times 1 = 15_{10}$$
$$34,207_{10}$$

Fractional hexadecimal numbers may be represented by using the sumbols as coefficients of ascending negative powers of the base.

16^{-1}	16^{-2}	16^{-3}	16^{-4}	...
$1/16_{10}$	$1/256_{10}$	$1/4096_{10}$	$1/65536_{10}$	

TABLE A-1. HEXADECIMAL EQUIVALENTS

Binary	Decimal	Hexadecimal
00000	00	00
00001	01	01
00010	02	02
00011	03	03
00100	04	04
00101	05	05
00110	06	06
00111	07	07
01000	08	08
01001	09	09
01010	10	0A
01011	11	0B
01100	12	0C
01101	13	0D
01110	14	0E
01111	15	0F
10000	16	10

†To perform this multiplication, the hexadecimal symbol F is first converted to its decimal equivalent 15 (table A-1).

The hexadecimal number .48C0 represents:

$$4 \times 16^{-1} = 4 \times 1/16 = \frac{1024}{4096_{10}}$$

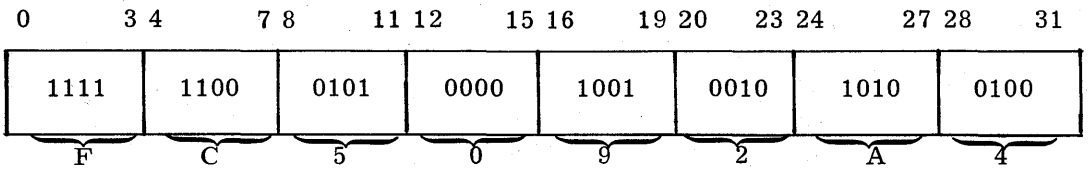
$$8 \times 16^{-2} = 8 \times 1/256 = \frac{128}{4096_{10}}$$

$$C \times 16^{-3} = C \times 1/4096 = \frac{12}{4096_{10}}$$

$$\frac{1164}{4096_{10}} = \frac{291}{1024_{10}} = .284$$

Since a group of four bits can represent any one of the 16 hexadecimal symbols, this notation is used throughout the instruction manuals for instruction codes, operands, addressing, etc. Table A-1 shows the hexadecimal equivalents for each unique group of four bits.

The hexadecimal number system enables direct substitution of a hexadecimal symbol for a group of four bits. Figure A-1 illustrates the substitution of a hexadecimal number for a 32-bit operand. Thus, the equivalent hexadecimal symbol is substituted for each successive group of four bits, producing the complete hexadecimal equivalent.



Equivalent Hexadecimal Number = FC5092A4

Figure A-1. Example of Hexadecimal Substitution for a Binary Number

Figure A-2 provides an easy way to add or multiply hexadecimal numbers.

ADDITION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION

1	1														
2	2	4													
3	3	6	9												
4	4	8	C	10											
5	5	A	F	14	19										
6	6	C	12	18	1E	24									
7	7	E	15	1C	23	2A	31								
8	8	10	18	20	28	30	38	40							
9	9	12	1B	24	2D	36	3F	48	51						
A	A	14	1E	28	32	3C	46	50	5A	64					
B	B	16	21	2C	37	42	4D	58	63	6E	79				
C	C	18	24	30	3C	48	54	60	6C	78	84	90			
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9		
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Figure A-2. Hexadecimal Matrices

BINARY ARITHMETIC

The following subparagraphs present a brief description of binary arithmetic, including the one's and two's complement systems.

ADDITION AND SUBTRACTION

Binary numbers are added according to the following rules:

1 + 1 = 0 with a carry of 1
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1

The addition of binary numbers proceeds as follows (the hexadecimal and decimal equivalents verify the result):

Augend	1001	(9)
Addend	0101	(5)
Partial Sum	1100	
Carrys	0010	
Sum	1110 = E_{16} = 14_{10}	

Binary numbers are subtracted according to rules shown as follows:

0 - 0 = 0
0 - 1 = 1 with a borrow of 1
1 - 0 = 1
1 - 1 = 0

An example of binary subtraction is shown as follows:

Minuend	1001	(9)
Subtrahend	0101	(5)
Partial Difference	1100	
Borrows	1000	
Difference	0100	(4)

Numbers can also be subtracted by adding the complement of the addend as shown below:

Augend	1010	(A) (10_{10})
Addend	1100	(-3) One's complement of +3.
Partial Sum	0110	
Carrys	0001	(End around carry)
Sum	0111	(+7)

The example above shows that the carry generated by the most significant stage of the add is added to the least significant stage (end around carry). The procedure for obtaining the one's complement of a binary number is described in the following subparagraphs.

ONE's COMPLEMENT

In this system, positive numbers are represented by the binary equivalent. The negative numbers are represented in one's complement notation of the corresponding positive number.

The one's complement of a number is found by subtracting each bit of the number from 1. For example:

```
 1111
-0101 (5)
 1010 (One's complement of 5)
```

The substitution of ones for zeros and zeros for ones also produces the one's complement representation of a negative number.

In general, a negative number in the one's complement system contains a 1 in the most significant bit (sign bit). Conversely, a positive number contains a 0 in the most significant bit. This feature divides the range (modulus) of numbers that a given machine can represent into two halves. One half of the range represents positive numbers while the other half represents negative numbers. A machine with modulus of 8 has the following range of numbers:

SIGN BIT

($-7F_{16}$) (-127_{10}) 10000000_2 (Maximum negative number)

($+7F_{16}$) ($+127_{10}$) 01111111_2 (Maximum positive number)

Figure A-2. Example of a Modulus 8 System

Thus, this machine has a modulus of $\pm (2^7-1)$.

If a 1 is added to the maximum positive number shown in the example, the result equals the maximum negative number as shown in figure A-3.

Such a result is termed an overflow because the result exceeds the modulus of the machine.

```
                  01111111
                        +1
Partial Sum       01111110
Carrys            11111110
Sum               10000000
                  ↑—OVERFLOW
```

Figure A-3. Example of Overflow

In a similar manner, figure A-4 shows that the subtraction of a one from the maximum negative number produces a result that exceeds the modulus of the machine in a negative direction. This result is termed an underflow.

```
                  10000000
                        -1
Partial Difference 10000001
Borrows           11111110
                  01111111
                  ↑—UNDERFLOW
```

Figure A-4. Example of Underflow

In the central computer, underflows and overflows are detected. In most cases, the detection of an overflow or underflow causes forced results. The type of forced results caused by the detection is included with the applicable instruction description.

TWO's COMPLEMENT

The two's complement system is used exclusively in central computer arithmetic operations. The system is similar to the one's complement system. Positive numbers are represented identically in the two systems. Negative numbers differ by one count. Table A-2 shows a comparison of one's and two's complement representations of counts 0-9. In the one's complement system, there are two representations for zero: a +0 and -0. Table A-2 shows the -0 as all ones in parentheses. This feature causes negative numbers in the one's and two's complement systems to vary by one count.

TABLE A-2. COMPARISON OF ONE'S AND TWO'S COMPLEMENT REPRESENTATIONS

Count	Two's Complement Representation	One's Complement Representation
+9	01001	01001
+8	01000	01000
+7	00111	00111
+6	00110	00110
+5	00101	00101
+4	00100	00100
+3	00011	00011
+2	00010	00010
+1	00001	00001
0	00000	00000 (11111)
-1	11111	11110
-2	11110	11101
-3	11101	11100
-4	11100	11011
-5	11011	11010
-6	11010	11001
-7	11001	11000
-8	11000	10111
-9	10111	10110

Positive numbers in the two's complement system can be converted to the equivalent negative numbers by first taking the one's complement of the positive number and then adding +1 to the result. Figure A-5 shows an example of the procedure.

```
00111   (+7)
11000   (One's complement = -7)
  +1    (Add one)
-----
11001   (Two's Complement = -7)
```

Figure A-5. Example of Converting a Positive Number to a Negative Number in Two's Complement

Addition and subtraction in the two's complement system are performed in the same way as in the one's complement system. However, the end-around carry and borrow features, used in the one's complement system, do not apply to the two's complement system. Figure A-6 shows a comparison of adding a -1 to a +8 in the one's and two's complement systems, respectively.

One's Complement		Two's Complement	
01000	(+8)	01000	(+8)
11110	(-1)	11111	(-1)
10110	(Partial Sum)	10111	(Partial Sum
10001	(Carrys)	10000	(Carrys)
	End-Around Carry		No End-Around Carry
00111	(Sum = +7)	00111	(Sum = +7)

Figure A-6. Comparison of Addition in the One's and Two's Complement Systems

MULTIPLICATION

Binary multiplication proceeds according to the following rules.

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Multiplication is always performed on a bit-by-bit basis.

Decimal example:

Multiplicand	14	
Multiplier	12	
Partial Products	28	
	14	(shifted left one place)
Product	168_{10}	

Binary example:

Multiplicand	(14_{10})	1110	
Multiplier	(12_{10})	1100	
Partial Products		0000	
		0000	shift to place
		1110	digits in proper
		1110	columns
Product	(168_{10})	10101000_2	

The following example is one method of computer multiplication. The central computer uses variations of this method. However, the following example is valid for explanation.

The computer determines the running subtotal of the partial products. Rather than shifting the partial product to the left to position it correctly, the computer right-shifts the summation of the partial products one place before the next addition is made. When the multiplier bit is a 1 , the multiplicand is added to the running total and the result is shifted to the right one place. When the multiplier is a 0 , the running total is shifted to the right, effectively multiplying the quantity by 10_2. Figure A-7 shows an example of the multiplication procedure used in the computer.

Multiplicand	1110	(14_{10})	
Multiplier	1100	(12_{10})	
(Multiplier Bit = "0")		0000	First Running Total (Shifted Right One)
(Multiplier Bit = "0")		00000	Second Running Total
		1110	(Shifted Right One)
(Multiplier Bit = "1")		111000	Third Running Total
		1110	(Shifted Right One)
		10101000	Product (168_{10})

Figure A-7. Example of Computer Multiplication Procedure

DIVISION

The following examples show the familiar method of decimal division.

```
                 14      Quotient
Divisor  13 ⌈185         Dividend
            13
             55          Partial Dividend
             52
              3          Remainder
```

The computer performs division in a similar manner (using binary equivalents):

```
                  1110     Quotient (14)
Divisor  1101 ⌈10111001    Dividend
              1101
              10100
               1101        Partial Dividends
               01110
                1101
                   11      Remainder (3)
```

However, instead of shifting the divisor right to position it for subtraction from the partial dividend (shown above), the computer shifts the partial dividend left, accomplishing the same result. Following each left shift, the divisor is subtracted from the dividend. If the result is positive, the corresponding bit of the quotient is set (1) and the resulting partial dividend is shifted left one position. If the result is negative, indicating that the divisor cannot be contained in the partial dividend, the corresponding bit of the quotient is cleared (0) and the previous partial dividend is shifted left one place. The process continues until the proper number (determined by the number of bits in the dividend) of subtraction and left-shift operations take place.

Figure A-8 shows an example of the division procedure used in the computer. Note that the first subtraction in the example would produce a negative result. Thus, the most significant bit of the quotient is cleared and the previous partial dividend (in this case, the initial dividend) is shifted left one position.

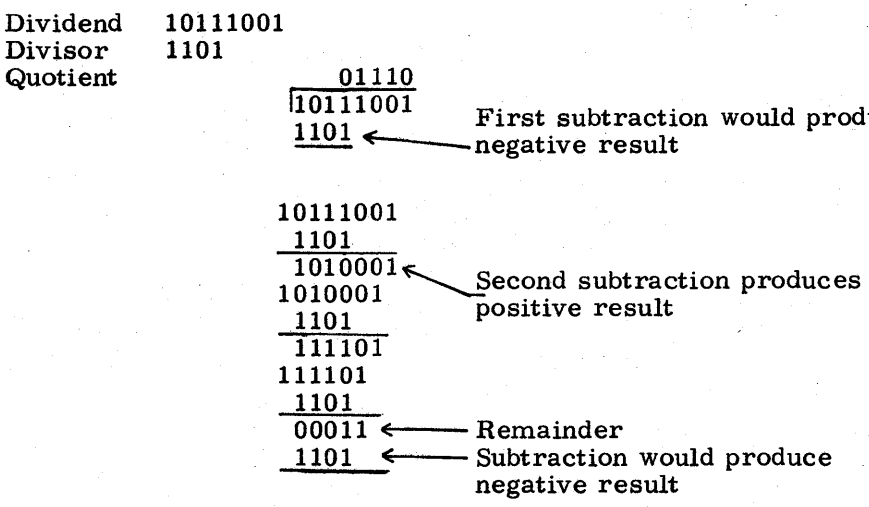


Figure A-8. Example of Computer Division Procedure

The second subtraction produces a positive result. Thus, the next most significant bit of the quotient is set and the result of the subtraction (partial dividend) is left shifted one place.

Note that the result of the third subtraction is retained as the remainder since the fourth (final) subtraction would produce a negative result.

NUMBER CONVERSIONS

The procedures that may be used when converting a number from one number system to another are power addition, radix arithmetic, and substitution. Table A-3 lists the recommended conversion procedures.

TABLE A-3. RECOMMENDED CONVERSION PROCEDURES (INTEGER AND FRACTIONAL)

Conversion	Recommended Method
Binary to Decimal	Power Addition
Decimal to Hexadecimal†	Power Addition
Decimal to Binary	Radix Arithmetic
Hexadecimal to Decimal†	Radix Arithmetic
Binary to Hexadecimal	Substitution
Hexadecimal to Binary	Substitution

General Rules

$r_i > r_f$: Use Radix Arithmetic, Substitution

$r_i < r_f$: Use Power Addition, Substitution

r_i = Radix of initial system

r_f = Radix of final system

†The Programming Reference Aids Manual (Control Data Pub. No. 60158600) lists the decimal to hexadecimal conversions for decimal numbers 0-40959.

POWER ADDITION

To convert a number from r_i to r_f ($r_i < r_f$), write the number in its expanded r_i polynomial form and simplify using r_f arithmetic.

Example 1: Binary to Decimal (Integer)

$$\begin{aligned} 010111_2 &= 1(2^4) + 0(2^3) + 1(2^2) + 1(2^1) + 1(2^0) \\ &= 1(16) + 0(8) + 1(4) + 1(2) + 1(1) \\ &= 16 + 0 + 4 + 2 + 1 \\ &= 23_{10} \end{aligned}$$

Example 2: Binary to Decimal (Fractional)

$$\begin{aligned} .0101_2 &= 0(2^{-1}) + 1(2^{-2}) + 0(2^{-3}) + 1(2^{-4}) \\ &= 0 + 1/4 + 0 + 1/16 \\ &= 5/16_{10} \end{aligned}$$

Example 3: Decimal to Hexadecimal (Integer)

$$
\begin{aligned}
875_{10} &= 8(10^2) + 7(10^1) + 5(10^0) \\
&= 8(A^2_{16}) + 7(A^1_{16}) + 5(A^0_{16}) \\
&= 8(64_{16}) + 7(A_{16}) + 5(1) \\
&= 320_{16} + 46_{16} + 5 \\
&= 36B_{16}
\end{aligned}
$$

NOTE

The base 10 is changed to the hexadecimal equivalent (A). The subsequent arithmetic is then performed in the hexadecimal system.

Example 4: Decimal to Hexadecimal (Fractional)

$$
\begin{aligned}
.25_{10} &= 2(10^{-1}) + 5(10^{-2}) \\
&= 2(A^{-1}_{16}) + 5(A^{-2}_{16}) \\
&= 2/A_{16} + 5/64_{16} \\
&= 19_{16}/64_{16} \\
&\approx .4/16
\end{aligned}
$$

RADIX ARITHMETIC

To convert a whole number from r_i to r_f ($r_i > r_f$):

1. Divide the number to be converted by r_f, as expressed in r_i notation, using r_i arithmetic.
2. The remainder is the lowest-order digit in the new expression.
3. Divide the integral part from the previous step by r_f, as expressed in r_i notation.
4. The remainder is the next higher-order digit in the new expression.
5. The process continues until the division produces only a remainder which will be the highest-order bit in the r_f expression.

To convert a fractional number from r_i to r_f:

1. Multiply the number to be converted by r_f, as expressed in r_i notation, using r_i arithmetic.
2. The integral part is the highest-order bit in the new expression.
3. Multiply the fractional part from the previous operation by r_f, as expressed in r_i notation.
4. The integral part is the next lower-order bit in the new expression.
5. The process continues until sufficient precision is achieved or the process terminates.

EXAMPLE 1 Decimal to Binary (Integer)

45 ÷ 2 = 22 remainder 1; record 1

22 ÷ 2 = 11 remainder 0; record 0

11 ÷ 2 = 5 remainder 1; record 1

5 ÷ 2 = 2 remainder 1; record 1

2 ÷ 2 = 1 remainder 0; record 0

1 ÷ 2 = 0 remainder 1; record 1

1 0 1 1 0 1

Thus $45_{10} = 101101_2$

EXAMPLE 2 Decimal to Binary (Fractional)

.25 x 2 = 0.5; record 0

.5 x 2 = 1.0; record 1

.0 x 2 = 0.0; record 0

.010

Thus $.25_{10} = .010_2$

EXAMPLE 3 Hexadecimal to Decimal (Integer)

9FC ÷ 10_{10} (A_{16}) = 0FF remainder 6; record 6

0FF ÷ A_{16} = 19 remainder 5; record 5

019 ÷ A_{16} = 2 remainder 5; record 5

2 ÷ A_{16} = 0 remainder 2; record 2

2556

Thus $9FC_{16} = 2556_{10}$

EXAMPLE 4 Hexadecimal to Decimal (Fractional)

.2AC x 10_{10} (A_{16}) = 1.AB8; record 1

.AB8 x A_{16} = 6.B30; record 6

.B30 x A_{16} = 6.FE0; record 6

.FE0 x A_{16} = 9.EC0; record 9

-- --

-- --

.1669--

Thus $.2AC_{16} \cong .1669_{10}$

SUBSTITUTION

This method permits easy conversion between hexadecimal and binary numbers. If a binary number is partitioned into groups of four bits to the left and right of the binary point, each group of four bits converts into a hexadecimal digit. Similarly, each hexadecimal digit converts directly into a group of four bits. Table A-1 lists the hexadecimal digits and the corresponding binary equivalents.

Example 1: Binary to Hexadecimal

Binary =	1110	0000	0101.	1011	0010	1001
Hexadecimal =	E	0	5 .	B	2	9

Example 2: Hexadecimal to Binary

Hexadecimal =	5	F	8 .	7	C	A
	0101	1111	1000.	0111	1100	1010

Tables A-4 and A-5 are translation tables for extended binary coded decimal interchange code (EBCDIC) and American National Standard Code for Information Interchange (ASCII). The double row of squares around the top and left edge of each table show the binary and hexadecimal codes for the characters in the table. The following list gives a description of the control characters in the tables.

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4 (Stop)
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell (audible or attention signal)	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (punched card skip) (FE)	EM	End of Medium
		SUB	Substitute
LF	Line Feed (FE)	ESC	Escape
VT	Vertical Tabulation (FE)	FS	File Separator (IS)
FF	Form Feed (FE)	GS	Group Separator (IS)
CR	Carriage Return (FE)	RS	Record Separator (IS)
SO	Shift Out	US	Unit Separator (IS)
SI	Shift In	DEL	Delete†

NOTE

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

Bits in the tables are identified by b_8, b_7, b_6, ... b_1 where b_8 is the highest order or most significant bit. Their numerical significance in binary is as follows:

Bit Identification $b_8\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1$

Significance $2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

†In the strict sense, DEL is not a control character.

TABLE A-4. EBCDIC TRANSLATION TABLE

b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1	ROW \ COL. →	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL	DLE			SP	&	–						{	}	\	0
0 0 0 1	1	SOH	DC1					/		a	j	~		A	J		1
0 0 1 0	2	STX	DC2		SYN					b	k	s		B	K	S	2
0 0 1 1	3	ETX	DC3							c	l	t		C	L	T	3
0 1 0 0	4									d	m	u		D	M	U	4
0 1 0 1	5	HT		LF						e	n	v		E	N	V	5
0 1 1 0	6		BS	ETB						f	o	w		F	O	W	6
0 1 1 1	7	DEL		ESC	EOT					g	p	x		G	P	X	7
1 0 0 0	8		CAN							h	q	y		H	Q	Y	8
1 0 0 1	9		EM						`	i	r	z		I	R	Z	9
1 0 1 0	10 (A)					[	]	¦	:								
1 0 1 1	11 (B)	VT				.	$	,	#								
1 1 0 0	12 (C)	FF	FS		DC4	<	*	%	@								
1 1 0 1	13 (D)	CR	GS	ENQ	NAK	(	)	—	'								
1 1 1 0	14 (E)	SO	RS	ACK		+	;	>	=								
1 1 1 1	15 (F)	SI	US	BEL	SUB	!	^	?	"								EO

TABLE A-5. ASCII TRANSLATION TABLE

b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1	ROW \ COL. →	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
		0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p								
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q								
0 0 1 0	2	STX	DC2	"	2	B	R	b	r								
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s								
0 1 0 0	4	EOT	DC4	$	4	D	T	d	T								
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u								
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v								
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w								
1 0 0 0	8	BS	CAN	(	8	H	X	h	x								
1 0 0 1	9	HT	EM	)	9	I	Y	i	y								
1 0 1 0	10 (A)	LF	SUB	*	:	J	Z	j	z								
1 0 1 1	11 (B)	VT	ESC	+	;	K	[	k	{								
1 1 0 0	12 (C)	FF	FS	,	<	L	\	l	¦								
1 1 0 1	13 (D)	CR	GS	−	=	M	]	m	}								
1 1 1 0	14 (E)	SO	RS	.	>	N	^	n	~								
1 1 1 1	15 (F)	SI	US	/	?	O	_	o	DEL								EO

FLOATING-POINT ARITHMETIC B

GENERAL

Most programmed arithmetic in the computer system takes place using two's complement, floating point procedures. The following paragraphs describe the formats and procedures used in performing floating-point operations. Unless otherwise specified, numbers are expressed in hexadecimal notation (base 16).

FLOATING POINT TECHNIQUE

The floating point technique allows the computer to represent numbers with variable radix points and to perform computations on these numbers. Using floating-point procedures, the computer automatically places the radix point of the result at the proper position following a computation. Thus, by shifting the radix point and increasing or decreasing the exponent, computations on widely varying quantities which do not exceed the capacity of the machine can be performed.

Floating-point numbers within the computer are represented in a form similar to scientific notation, that is, a coefficient multiplied by a number raised to a power. Since the computer operates only on binary numbers, the numbers are multiplied by powers of two.

$C \cdot 2^E$ Where: C = coefficient
E = exponent

In floating-point, different coefficients need not relate to the same power of the base as do fixed point numbers. Therefore, the format of a floating point number includes both the coefficient and exponent. All coefficients and exponents represented in the equipment are signed integers.

OPERAND FORMATS

Floating-point operations are performed on 32-bit and 64-bit operands. The function codes of the corresponding instructions specify whether 32-bit or 64-bit operands are to be used. The following subparagraphs describe the 32-bit and 64-bit formats.

32-BIT FORMAT

Figure B-1 shows the format of the 32-bit floating point operands. Note that the bit positions of all operands are numbered left to right with the least significant bits in the rightmost bit positions of the word.

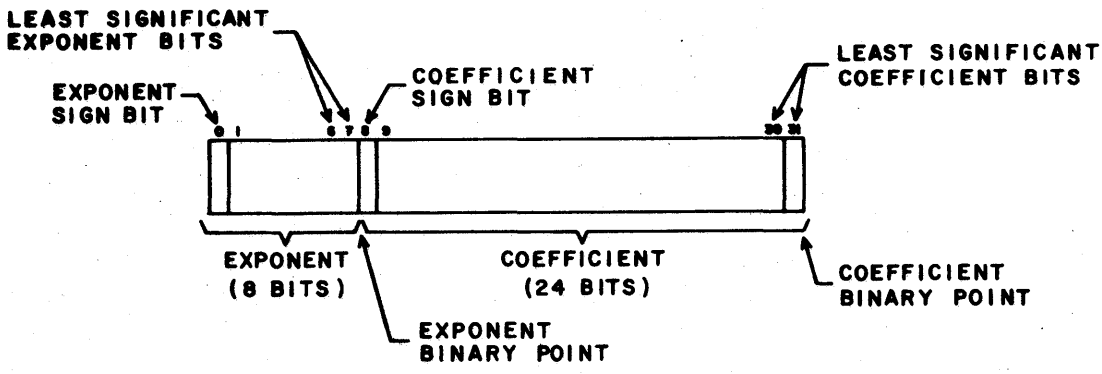


Figure B-1. 32-Bit Floating-Point Operand Format

The range of useful coefficients in the 32-bit format is from 800000 to 7FFFFF which provides a range of $-(2^{23})._{10}$ through $+2^{23}-1)_{10}$.

Useful exponents range from 90 to 6F which gives a range of -112_{10} to $+111_{10}$. Numbers 70 through 8F fall into a special end-case range as listed in table B-1.

TABLE B-1. SPECIAL END CASE RANGE FOR THE 32-BIT FORMAT

Number	Definition
8XXXXXXX	Machine Zero
7XXXXXXX	Indefinite
Note: X = Any Hexadecimal Digit	

Table B-2 lists some floating point numbers in the 32-bit format. Unless otherwise indicated, all numbers are in two's complement, hexadecimal notation.

TABLE B-2. FLOATING-POINT NUMBERS IN 32-BIT FORMAT

Number (Base 10)	Floating Point Format	
	Exponent	Coefficient
+1	00	000001
+1 Normalized†	EA	400000
-1	00	FFFFFF
-1 Normalized†	E9	800000
+26790.0	00	0068A6
$+1/4 = +.25 = +.40_{16}$ Normalized†	E8	400000
256	00	000100

†In these examples, the coefficients are left shifted (normalized) until the sign bit is unequal to the bit immediately to its right. The exponent is reduced by one for each left shift.

Note that in two's complement notation, a negative number is one more than the corresponding one's complement notation for the same number. For example, in two's complement, -1 = FFFFFF (all ones) while in one's complement -1 = FFFFFE. Positive numbers in two's complement are identical to the corresponding one's complement notation for the same number.

64-BIT FORMAT

Figure B-2 shows the format of the 64-bit floating point operands.

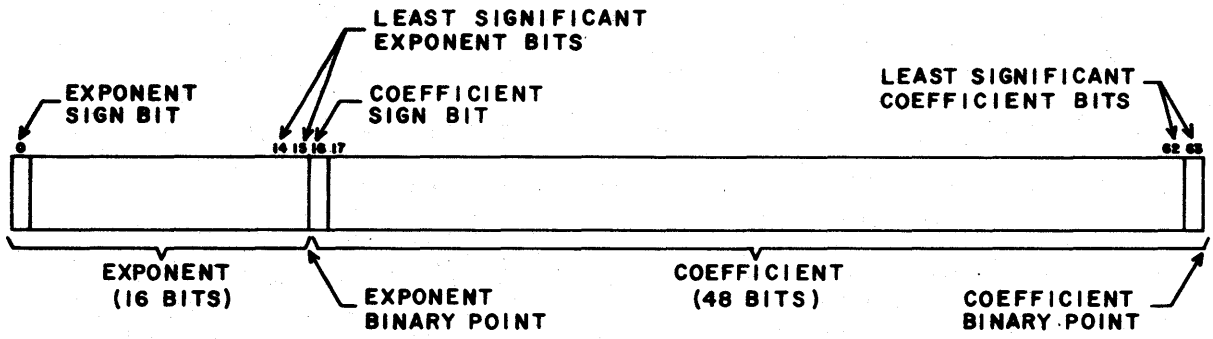


Figure B-2. 64-Bit Floating-Point Operand Format

The range of useful coefficients in the 64-bit format is from 8000 0000 0000 to 7FFF FFFF FFFF which provides a range of $-(2^{47})_{10}$ through $+(2^{47}-1)_{10}$.

Useful exponents range from 9000 to 6FFF which gives a range of $-28,672_{10}$ to $+28,671_{10}$. Numbers 7000 through 8FFF fall into a special end case range as listed in table B-3.

TABLE B-3. SPECIAL END CASE RANGE FOR THE 64-BIT FORMAT

Number	Definition
8XXX XXXX XXXX XXXX	Machine Zero
7XXX XXXX XXXX XXXX	Indefinite
Note: X = Any Hexadecimal Digit.	

The use of an undefined exponent in an arithmetic operation produces undefined results.

Table B-4 lists some floating point numbers in the 64-bit format.

TABLE B-4. FLOATING-POINT NUMBERS IN 64-BIT FORMAT

Number Base 10	Floating Point Format	
	Exponent	Coefficient
+1	0000	0000 0000 0001
+1 Normalized †	FFD2	4000 0000 0000
-1	0000	FFFF FFFF FFFF
-1 Normalized †	FFD1	8000 0000 0000
+26790.0	0000	0000 0000 68A6
$+1/4 = +.25 = +.40_{16}$	FFD0	4000 0000 0000*
$+256_{10}$	0000	0000 0000 0100

†In these examples, the coefficients are left shifted (normalized) until the sign bit is unequal to the bit immediately to its right. The exponent is reduced by one for each shift.

FLOATING POINT OPERATIONS

In the following descriptions of floating point operations, the 32-bit format is used for all examples. All descriptions and definitions of the operations apply to 64-bit operands with the adjustment for bit length. The following bit length substitutions are made for operations using 64-bit operands.

Bit Lengths For 32-Bit Operands	Bit Lengths For 64-Bit Operands
22	46
23	47
46	94
47	95
11	23

DOUBLE PRECISION RESULTS

Several instructions produce double precision results. The double precision add operation is a floating point add producing both an upper and a lower result simultaneously and retaining both of these results for the next floating point add operation. Thus the partial result in 64-bit arithmetic consists of 94 coefficient bits plus sign information. The partial result in 32-bit arithmetic consists of 46 bits plus sign information.

Dot Product instructions add both the upper and lower results of the multiply operations to the partial results of the add operations as described above.

UPPER AND LOWER RESULTS

Floating point add, subtract, and multiply instructions generate result coefficients twice the length of the source-operand coefficients. The left and right halves of the result operands are called the upper (U) result and lower (L) result, respectively. Figure B-3 shows the format of the result operands.

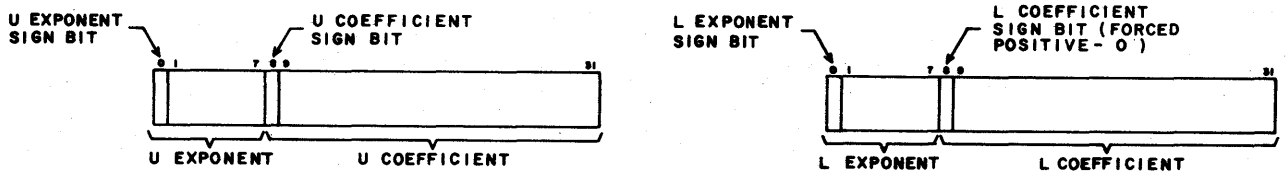


Figure B-3. Add, Subtract and Multiply Result Format

The sign bit of the lower result coefficient is forced positive. The remaining bits of the lower coefficient are the normal results of the computations. Since the sign bit of the lower result coefficient is forced positive, the lower result is not meaningful alone, but must be used in conjunction with the upper result.

END CASES

If an indefinite operand is used in a floating point operation, the upper and lower results are indefinite. Table B-5 lists each of the end case conditions and the result of each condition. In table B-5, 0 represents machine zero and N represents an operand that is not machine zero or indefinite. The coefficient of N is not all zeros.

TABLE B-5. END CASE CONDITIONS AND RESULTS

Condition	Result	Condition	Result
0 ± 0	0	N · 0	0
0 ± N	±N	0 ÷ 0	Indefinite
N ± 0	N	0 ÷ N	0
0 · 0	0	N ÷ 0	Indefinite
0 · N	0		

FLOATING-POINT COMPARE RULES

The rules governing the comparison of floating point operands are described on the following pages.

Neither Operand Indefinite or Machine Zero

If the signs of the coefficients of the two operands are unlike, the operands are unequal. The operand with the positive exponent is the larger of the two. If the signs of the coefficient are alike, the machine performs a floating point subtract upper. This operation subtracts operand (S) from operand (R). Each of the arithmetic results are listed below with the corresponding compare results.

Arithmetic Result	Compare Result
Coefficient upper 24 bits all zeros (48 bits for 24 through 27 instructions)	(R) = (S)
Coefficient upper 24 bits not all zeros (48 bits for 24 through 27 instructions)	(R) ≠ (S)
Coefficient positive	(R) ≥ (S)
Coefficient negative	(R) < (S)

The compare results (R) = (S) and (R) ≠ (S) do not guarantee that (S) = (R) when (R) = (S).

The order of events of the floating point subtract upper is first to complement the subtrahend, then align the coefficient associated with the smaller exponent, and finally to perform a floating point add operation. The following is an example of (R) = (S) but (S) ≠ (R) for 64-bit compares.

Operand (R) =	0104	0000	0000	0001	
(S) =	0100	0000	0000	0001	
Complement (S)	0100	FFFF	FFFF	FFFF	
Align (S)	0104	FFFF	FFFF	FFFF	F
Add (R) and	0104	0000	0000	0001	
complemented,	0104	FFFF	FFFF	FFFF	F
aligned (S)	0104	0000	0000	0000	F

Since the upper 48 bits of the result coefficient are all zeros, the pair of operands are considered equal. However, if the operands are interchanged, the following happens:

Operand (R) =	0100	0000	0000	0001	
(S) =	0104	0000	0000	0001	
Complement (S)	0104	FFFF	FFFF	FFFF	
Align R	0104	0000	0000	0000	1
Add aligned (R) and complemented (S)	0104	FFFF	FFFF	FFFF	1

Since the upper 48 bits of the result coefficient are not all zeros, the pair of operands are considered unequal.

Figure B-4 shows an example of the results of a branch if (R) $\geq$ (S) (32/64 bit FP), 22 instruction with the assumed instruction codes and register content. Note that in the initial comparison of the coefficient signs of (R) and (S) that they are alike. Thus a floating point subtract operation contains a positive sign which indicates that (R) > (S). Since this result satisfies the assumed branch condition, the program branches to the indicated branch address.

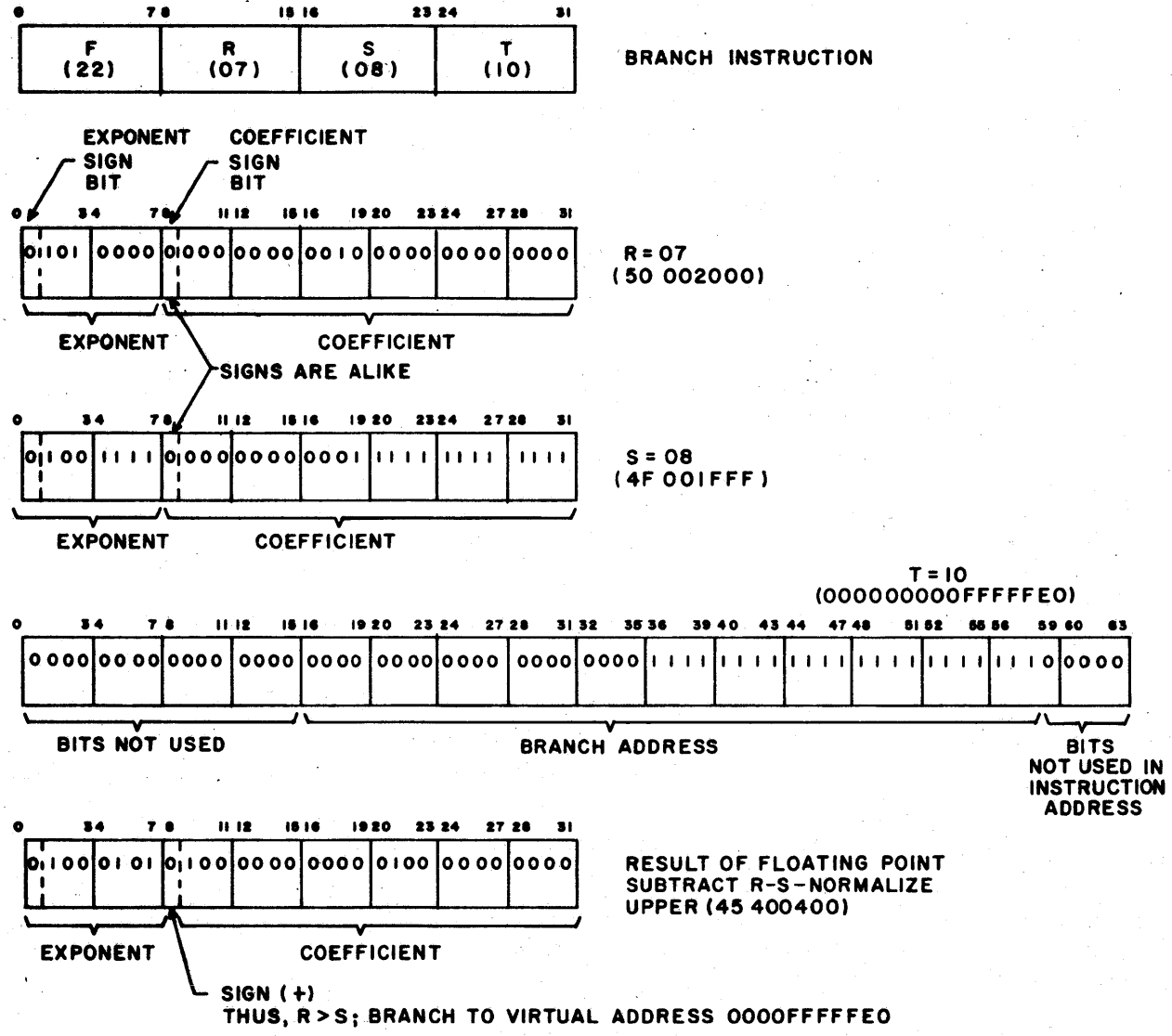


Figure B-4. Example of Branch if (R) $\geq$ (S) (32/64 Bit FP) Instruction

One or Both Operands Indefinite

If one operand is indefinite, the compare condition is not met since indefinite is not greater than, less than, equal to, or not equal to any other operand. If both operands are indefinite, the (R) = (S) and (R) $\geq$ (S) conditions can be met since indefinite equals indefinite.

Neither Operand Indefinite But One or Both Operands Are Machine Zero

Under this condition, the following definitions apply to the comparison.

1. Any nonindefinite, nonmachine zero operand with a positive, nonzero coefficient is greater than machine zero.
2. Any nonindefinite, nonmachine zero operand with a negative coefficient is less than machine zero.
3. Machine zero is considered equal only to itself and to any number having a finite exponent and an all zero coefficient.

RIGHT NORMALIZATION

When the upper result coefficient overflows, the machine shifts the entire 47-bit result (with sign extension) one place to the right. The upper exponent is increased by one. The machine performs this operation, termed right normalization, when necessary, although normalization may not have been specified by the instruction.

Figure B-5 shows an example of right normalization. In this example, assume that the following floating point numbers are added, causing the upper result coefficient to overflow.

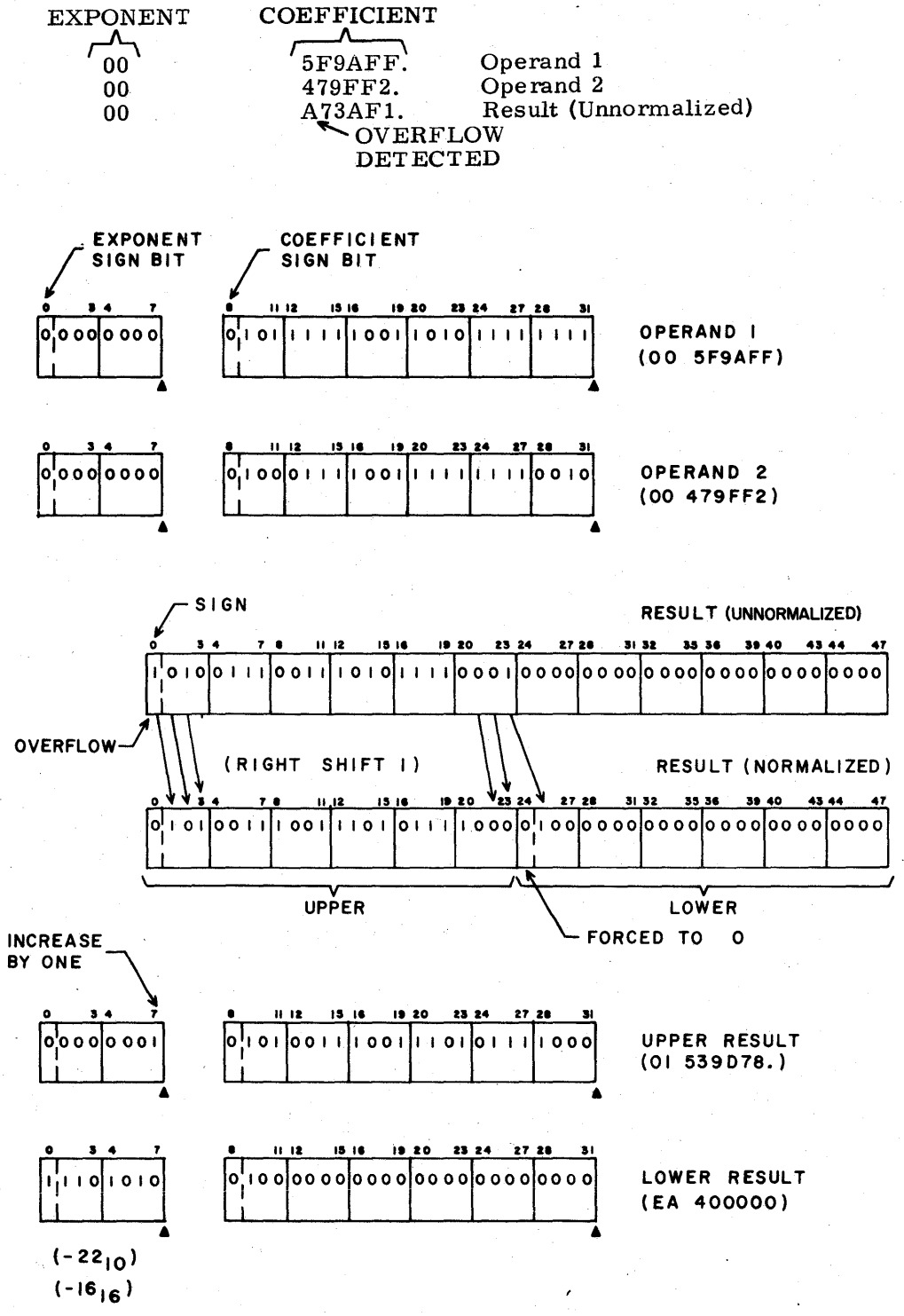


Figure B-5. Example of Right-Normalization

Note in the example that the sign bit of the lower result is forced positive (0) and bit 23 is shifted around it.

ADD AND SUBTRACT OPERATIONS

Before the computer adds or subtracts floating-point numbers, the exponents are made equal by a procedure called alignment. The alignment procedure successively right shifts the coefficient of the operand with the smallest exponent one bit and increases the exponent by one until the two exponents are equal. The sign of the shifted coefficients is extended from the left to the right during the shift. Negative coefficients approach a minus one and positive coefficients approach zero as they are shifted.

Figure B-6 shows an example of floating-point addition with both operands positive. In figure B-6, the exponent of operand 2 is one less than the exponent of operand 1. The alignment procedure right shifts the coefficient of operand 2 one place to the right and increases its exponent by one, making it equal to the exponent of operand 1. Note that the least significant bit of operand 2 is shifted into bit 25 of the lower result (around the sign bit).

The addition of the coefficients takes place, using conventional binary addition procedures. After right normalization, if required, the result is 46 bits (not including the sign bits). The leftmost 23 bits contain the coefficient for the upper result and the rightmost 23 bits contain the coefficient for the lower result.

The exponent for the upper result equals the larger of the two source operand exponents. Note that right normalization (not necessary in the example) increases this exponent by one. The exponent for the lower result equals the upper result exponent -23_{10} (17_{16}) in all but the following three conditions.

1. Right normalization causes the upper result exponent to overflow. In this case, the computer sets the upper result to indefinite. The lower exponent will equal 59_{16} ($6FD1_{16}$ for 64-bit operands).
2. If the subtraction of 23_{10} from the upper result exponent causes the lower result exponent to underflow, the computer sets the lower result to machine zero.
3. If one or both operands were indefinite, the computer sets the upper and lower results to indefinite.

Figure B-7 shows an example of floating-point addition with one operand negative and the other positive.

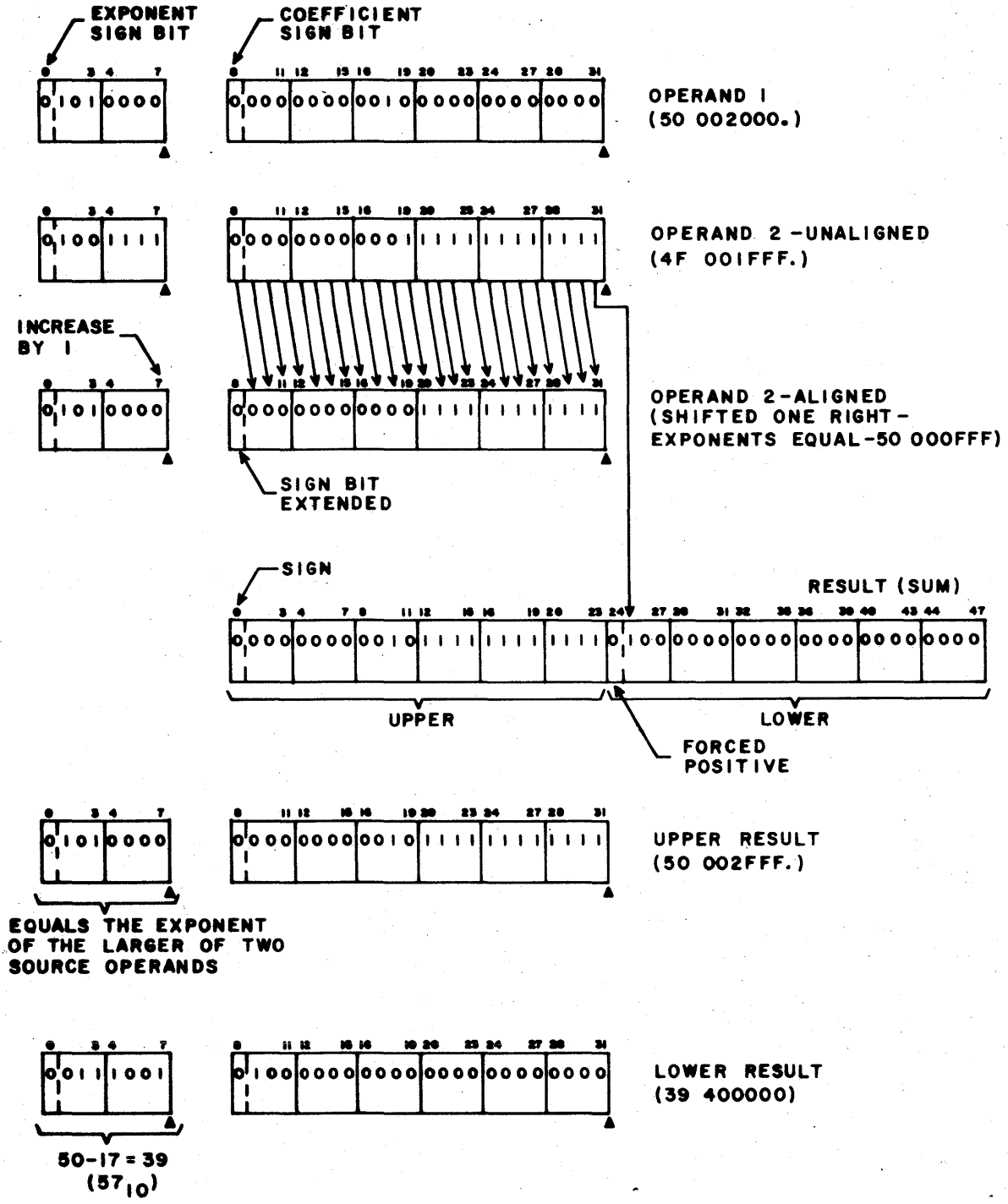


Figure B-6. Example of Floating-Point Addition (Both Operands Positive)

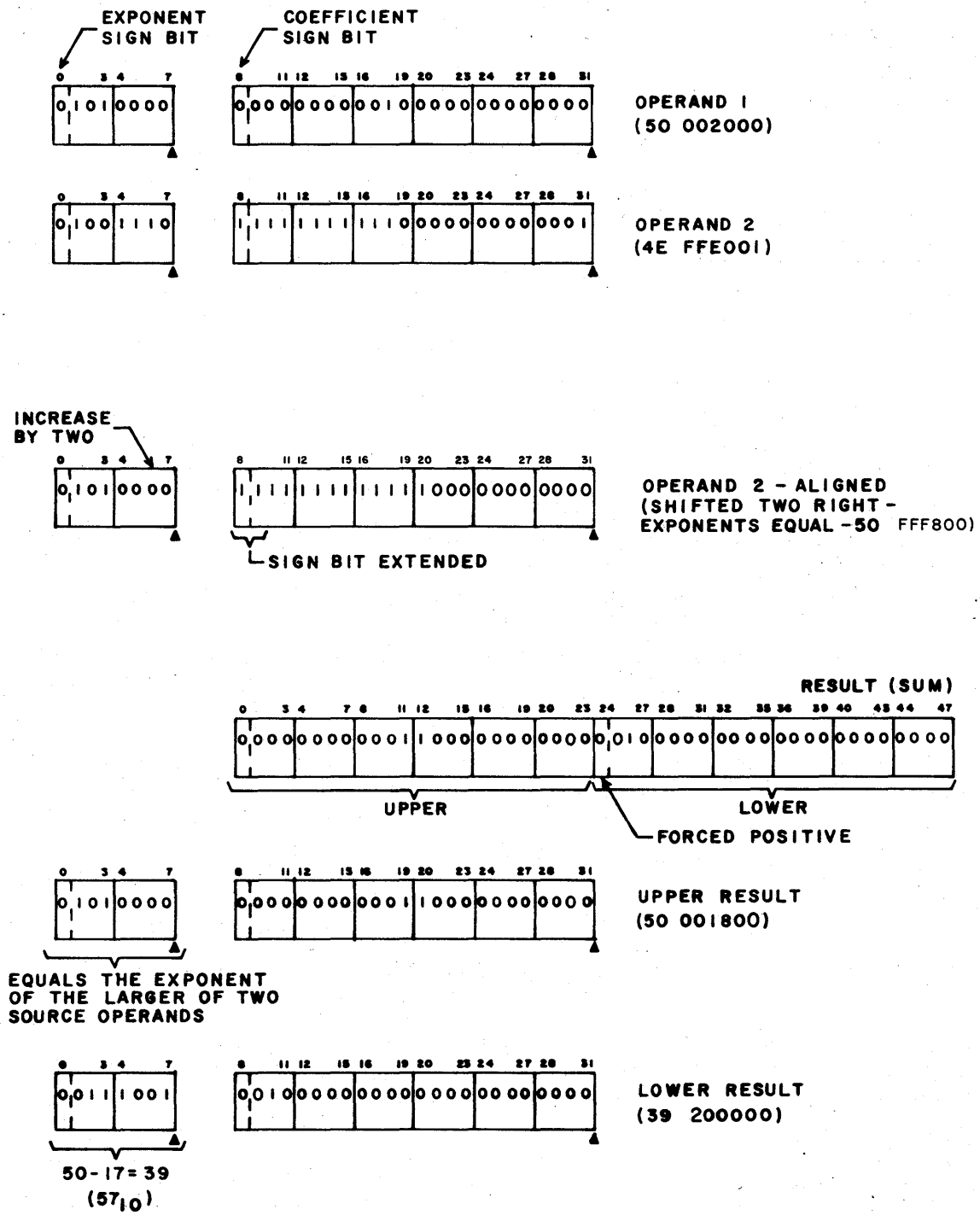


Figure B-7. Example of Floating-Point Addition (One Operand Negative and One Operand Positive)

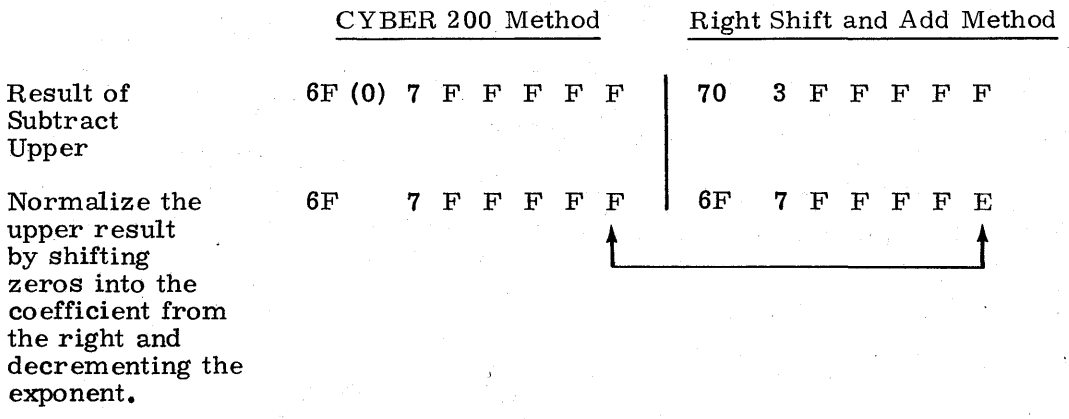
A floating-point subtraction consists of complementing the coefficient of the subtrahend and performing a floating-point addition. In 32-bit format, a 24-bit two's complement operation is performed before the operands are shifted. The complement of an 800000 coefficient is 400000 with one added to the value of the exponent associated with the coefficient.

The central computer hardware used for floating-point add or subtract operations has an extra (or extended) coefficient sign bit. This means that 8000 is complemented without the specified right shift of one and increase of the exponent by one. This causes a result which, although not mathematically incorrect, may differ from the specified result when all of the following conditions are met for any given pair of operands.

- The operand having the larger exponent must have a coefficient of 8000. If the exponents of the two operands are equal, one of the two must have a coefficient of 8000.
- The operand described in condition 1, having a coefficient of 8000, must be complemented. This may be due to the operand being the subtrahend in a subtract operation or because of sign control in either a subtract or add operation.
- The other operand must have a negative coefficient.

Figure B-8 shows two examples of floating-point subtraction using an extra coefficient sign bit.

If this operation is a subtract upper, the specified result is indefinite (with the appropriate data flags). The central computer result did not overflow. If this operation is a subtract normalized, the following results occur.



EXAMPLE 1 A - B

A 60 F F F 0 0 0

B 64 8 0 0 0 0 0

		CYBER 200 METHOD (EXTRA SIGN BIT)		RIGHT SHIFT AND ADD 1 TO EXPONENT METHOD	
COMPLEMENT B	B	64	(1) 8 0 0 0 0 0	64	8 0 0 0 0 0
	$\overline{B}$	64	(0) 8 0 0 0 0 0	65	4 0 0 0 0 0
ALIGN OPERAND WITH SMALLER EXPONENT		60	(1) F F F 0 0 0	60	F F F 0 0 0
		64	(1) F F F F 0 0	65	F F F F 8 0
ADD A PLUS COMPLEMENT OF B	A	64	(1) F F F F 0 0	65	F F F F 8 0
	$+\overline{B}$	64	(0) 8 0 0 0 0 0	65	4 0 0 0 0 0
		64	(0) 7 F F F 0 0	65	3 F F F 8 0
		64	7 F F F 0 0	65	3 F F F 8 0

EXAMPLE 2 A - B

A 50 F F F 0 0 0

B 6F 8 0 0 0 0 0

		CYBER 200 METHOD (EXTRA SIGN BIT)		RIGHT SHIFT AND ADD 1 TO EXPONENT METHOD	
COMPLEMENT B	B	6F	(1) 8 0 0 0 0 0	6F	8 0 0 0 0 0
	$\overline{B}$	6F	(0) 8 0 0 0 0 0	70	4 0 0 0 0 0
ALIGN OPERAND WITH SMALLER EXPONENT		50	(1) F F F 0 0 0	50	F F F 0 0 0
		6F	(1) F F F F F F	70	F F F F F F
ADD A PLUS COMPLEMENT OF B	A	6F	(1) F F F F F F	70	F F F F F F
	$+\overline{B}$	6F	(0) 8 0 0 0 0 0	70	4 0 0 0 0 0
		6F	(0) 7 F F F F F	70	3 F F F F F

Figure B-8. Examples of Floating-Point Subtraction Using an Extra Coefficient Sign Bit

The normalized add and subtract instructions generate an intermediate result identical to the final result of the add U and the subtract U instructions. Normalizing of the intermediate, 24-bit result then takes place. In this operation (figure B-9), the computer left shifts the 24 upper result bits until the sign bit and the bit immediately to the right of the sign bit are different.

The machine attaches zeros to the right of the result as it is shifted. The result exponent is reduced by the number of places shifted. If reducing the exponent by one causes exponent underflow, the result is set to machine zero. If the original coefficient consists of 24 zero bits, the result of the normalization becomes machine zero. If normalization is not specified in an add or subtract instruction, a zero coefficient and any exponent may result, and if reducing the exponent during shifting causes an exponent underflow, the machine sets the result to machine zero.

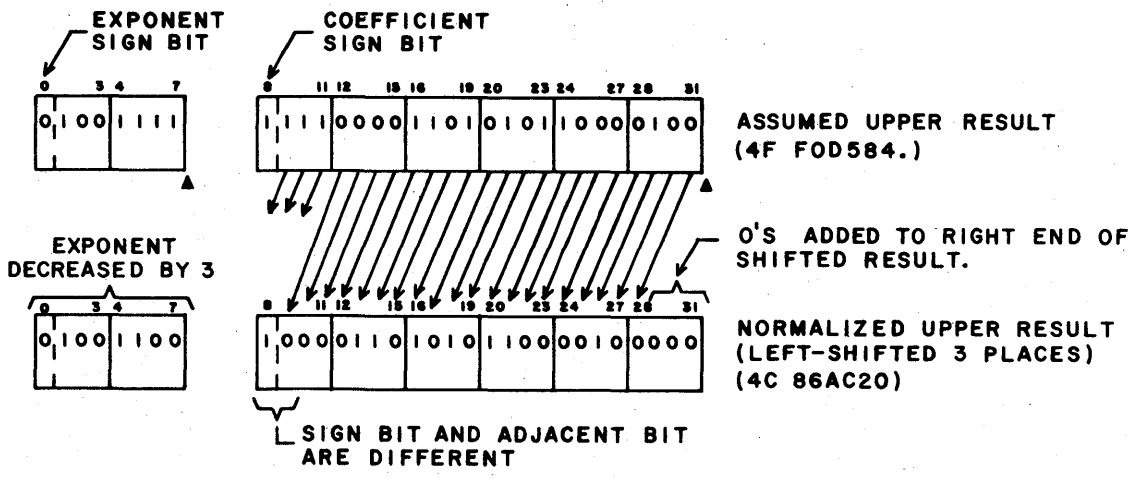


Figure B-9. Example of Normalized Upper Result

ORDER DEPENDENT RESULT CONSIDERATIONS

The result of any sequence of floating point operations may be operand-order dependent [for instance, $(A + B) + C \neq A + (B + C)$] .

The following example using 32-bit operands demonstrates this effect.

A =	00	000001
B =	00	000003
C =	01	000001
A	00	000001
+B	00	000003
A+B	00	000004
+C	01	000001
(A+B)+C	01	000003
B	00	000003
+C	01	000001
B+C	01	000002
+A	00	000001
A+(B+C)	01	000002

Coefficients not equal

It is important that this characteristic of floating point arithmetic be considered when predicting the results of the DA, DB, DC, DD, and DF instructions.

MULTIPLY OPERATIONS

The multiplication of two floating-point operands produces a result coefficient with the least-significant 23 product bits in the lower result and the higher order 23 product bits in the upper result (figure B-10). Note that as in addition and subtraction, the sign bit of the lower result is cleared, forcing the lower result positive. The sign bit of the upper result is determined using the usual procedures of algebraic multiplication. Thus, in the example shown in figure B-10, the sign bit of the upper result is a zero (+) since both source operands are positive.

In the multiply operation, the positive forms of the input operands are used. The signs of the input operands are recorded to determine the sign of the upper result and whether the resultant coefficient should be complemented. If either of the input operands contains a coefficient of 800000, the operation changes the operand to a positive form by right shifting its coefficient by one (with sign extension) and adding one to its exponent. This gives a coefficient of C00000 which will then be complemented to 400000.

The lower result exponent is the sum of the exponents for the two source operands and the upper result exponent equals the lower result exponent plus 17_{16} or 23_{10} with the following exceptions.

1. The sum of the source operands' exponents (plus 23_{10}, if upper result) exceeds $6F_{16}$, in which case the result exponent is set to indefinite.
2. The sum of the source operands' exponents (plus 23_{10}, if upper result) is less than 90_{16}, in which case the result exponent is set to machine zero.
3. Either or both operands are indefinite, in which case the result exponent is set to indefinite.
4. Neither operand is indefinite but either or both operands are machine zero, in which case the result exponent is set to machine zero.

DIVIDE OPERATIONS

In divide operations, a floating point dividend is divided by a prenormalized divisor, producing a 23-bit coefficient (not including sign bit) of the quotient which appears as the upper result. If one or both source operands are negative, they are complemented and the absolute values are used in the divide operation. The signs of the original source operands determine the sign of the final coefficient according to the normal procedures of algebraic divisons.

Figure B-11 shows an example of floating point division with both dividend and divisor positive. Note that prenormalization left shifts the divisor until the most significant one bit is adjacent to the sign bit. The normalize count (NC) is stored and will partially determine the exponent of the quotient.

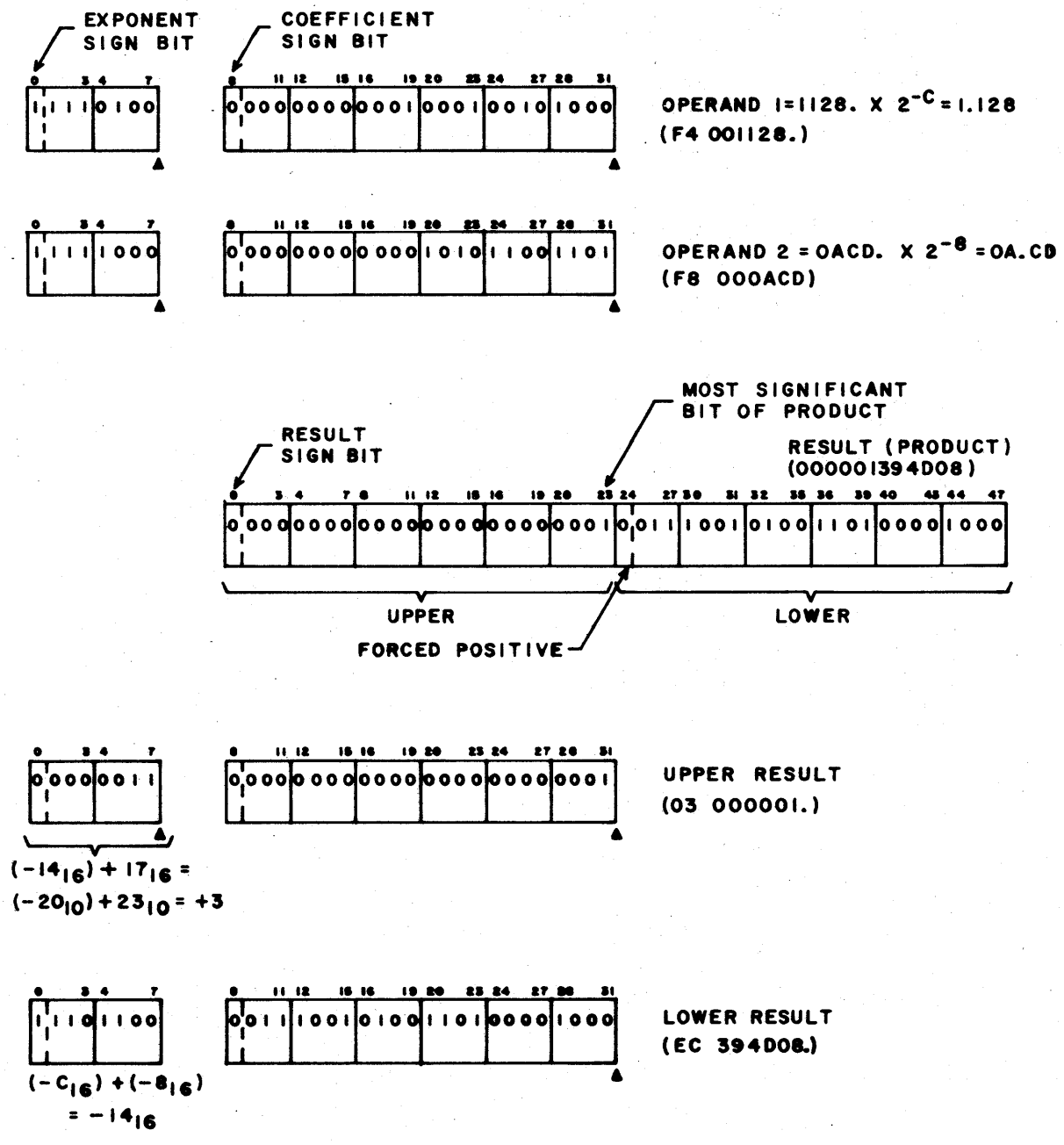


Figure B-10. Example of Floating Point Multiply

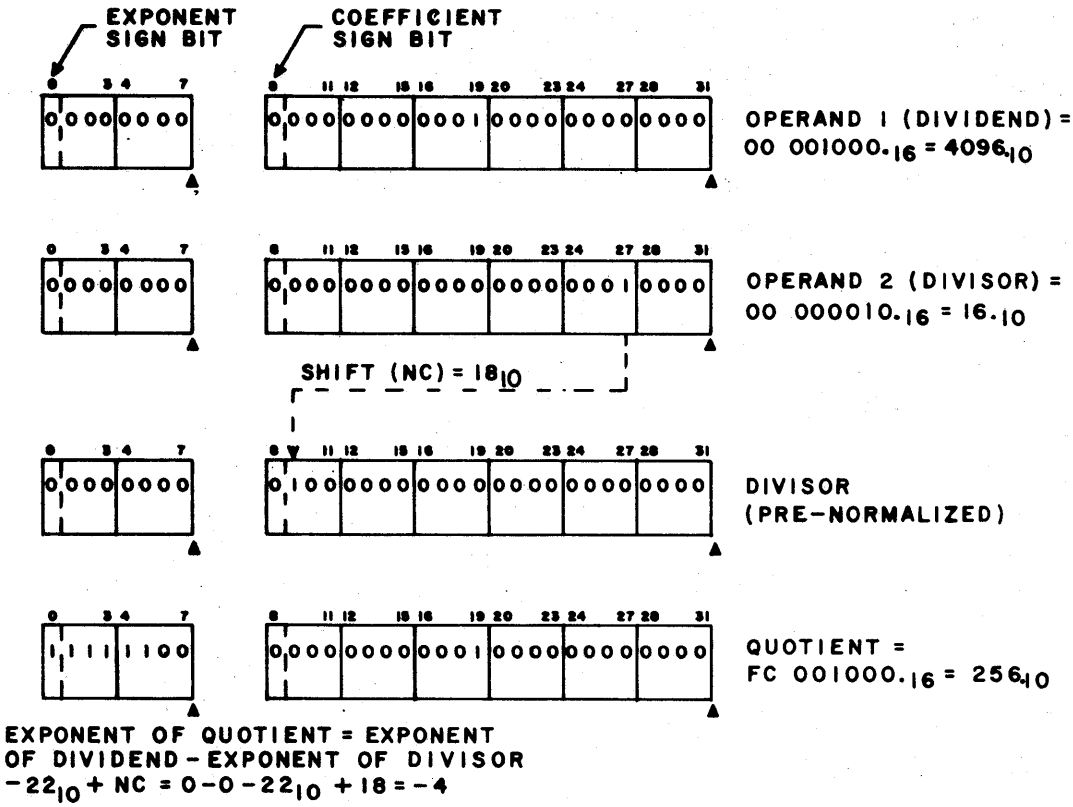


Figure B-11. Example of Floating Point Divide
(Dividend and Divisor Both Positive)

The prenormalized divisor is then subtracted from the dividend and the corresponding bit of the quotient is determined. After each subtraction, the partial dividend is left shifted one position and the subtraction is repeated as in a conventional binary division operation.

After 23 subtract and 22 shift operations have been completed, the absolute value of the quotient coefficient appears as the upper result. If either the original dividend or divisor (but not both) were negative, the coefficient of the quotient is complemented. The rightmost bit of the quotient is neither rounded nor adjusted. The remainder is not retained.

The exponent of the quotient is determined by the equation shown in figure B-11.

Figure B-12 shows another example of floating-point division. However, in this case, the dividend is positive and the divisor is negative. As a result, the original divisor is complemented before the prenormalization takes place. Note that the quotient is complemented to form the negative final quotient.

SIGNIFICANT RESULTS

Certain multiply and divide instructions specify that the significant results of the product or quotient be obtained. The significant bit count for a floating point number equals the number of bit positions in the number (excluding sign bit) minus the left shift count necessary to normalize the number. Refer to example in figure B-13.

A coefficient containing all zeros or all ones has a significant bit count of zero. Note that in a nonzero coefficient that is an exact power of two, the positive form of the coefficient results in a significant bit count that is one greater than the significant bit count of the negative form of the same coefficient. The operation determines the significance of an input operand as originally read from a register or from MCS before any operations such as sign control or the left shift for odd exponents in square root are performed.

Significant arithmetic determines which of the source operands contains the smaller significant bit count and records that count. After the following arithmetic operation, the sequence determines the significant bit count of the result after any necessary sign correction. The input significant bit count and the result significant bit count are then compared. If the significant bit count of the result is less than the significant bit count of the input, the sequence left-shifts (with zeros shifted in) the result coefficient according to the difference in significant bit counts and reduces the exponent accordingly. If the result and input significant bit counts are equal, the sequence does not shift the coefficient and does not adjust the exponent. If the result significant bit count is greater than the input significant bit count, the operation right-shifts (end off with sign extension) and increases the exponent accordingly. Note that for multiply, the entire 95-bit result (47 bits for 32-bit multiply) is shifted as required.

Exponent overflow, exponent underflow, and divide fault cause forced results as previously described. Adjusting for significance can cause exponent overflow or underflow or it can take a result out of the exponent overflow or underflow condition.

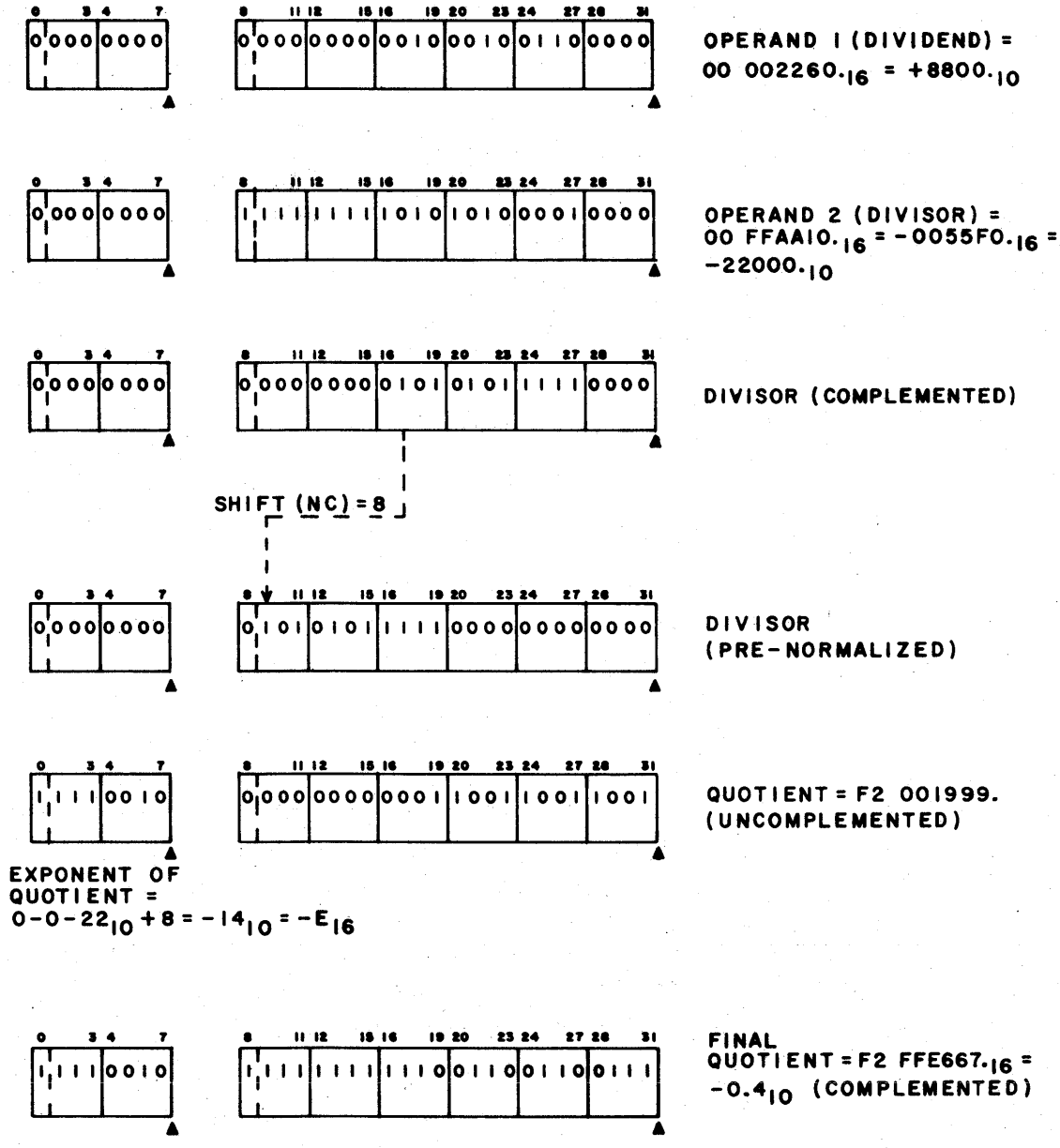


Figure B-12. Example of Floating Point Divide (Dividend Positive, Divisor Negative)

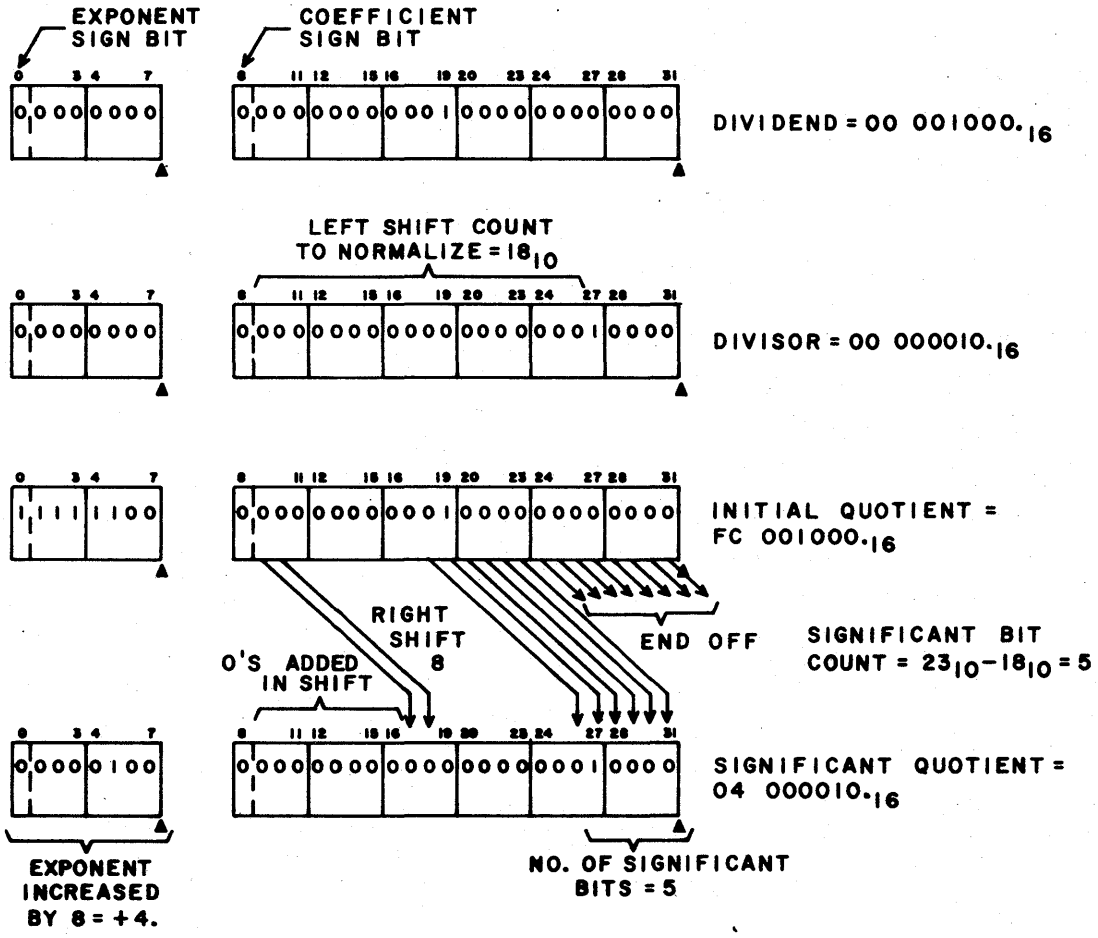


Figure B-13. Example of Significant Results of Floating Point Divide

SQUARE ROOT OPERATIONS

In floating point, square root operations, the following steps are performed.

1. The significance of the coefficient of the input operand is determined and recorded.
2. If negative, the input operand is complemented to its positive form.
3. If the exponent of the input operand is odd, it is reduced by one and the coefficient obtained in step 2 is multiplied by two. If the exponent is even, no modification is performed.

4. The machine now obtains the square root of the coefficient from step 3. Note that enough zeros are attached to the right end of the coefficient to produce 23 result bits (47 for 64-bit operands).

5. If the original input operand was negative, the result coefficient is complemented. If the input operand was positive, no modification takes place.

6. The result exponent is formed by dividing the exponent by two and subtracting 11_{10} from the exponent obtained in step 3. (Subtract 23_{10} for 64-bit square root.)

7. The result coefficient is adjusted to produce a coefficient with the same significance as the input operand. The significance count obtained in step 1 is used in the operation. The exponent of the result is also adjusted to compensate for the change in magnitude of the result coefficient.

8. A source operand having an all zero coefficient will produce a result with an all zero coefficient. The operand exponent effectively divides by two by right shifting one place with sign extension. If the source operand is negative, data flag bit 45 is set. If the source operand is indefinite or machine zero, the result is indefinite or machine zero, respectively. In these two cases, data flag bit 45 is not set.

Figure B-14 shows an example of a floating point, square root operation. In this example a positive input source operand is used. Thus, no complementing is necessary.

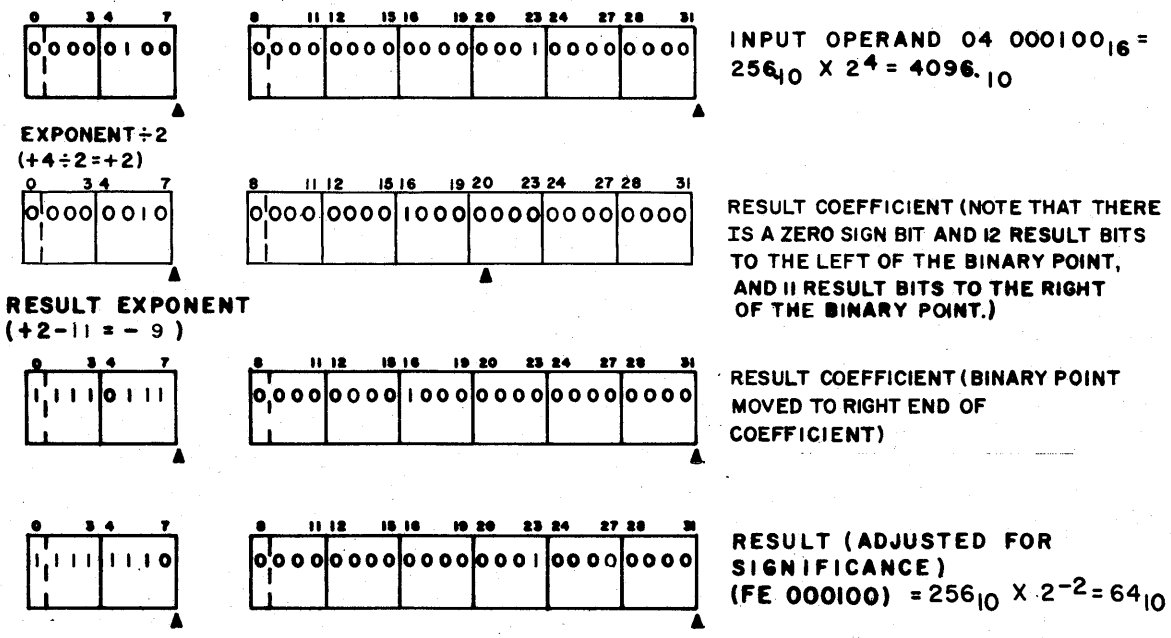


Figure B-14. Example of Floating Point Square Root

G BITS AND TERMINATING CONDITIONS C

G BIT USAGES

The following tables provide the instruction G bit usages in a condensed form. Thus, these tables provide quick look-up charts for determining the G bit control configuration for a particular instruction to which they apply. Note that the G bit usages tables are arranged according to instruction type (vector[VT], sparse vector [SV], etc.) and according to function code within that type of instructions.

The key to the abbreviations used to designate the G bit usage conditions is given below:

G Bit	Abbreviation	Meaning
0	E	Either 32- or 64-bit operands
1	C	Control vector
2	O	Offset
3, 4	B	Broadcast
5, 6, 7	S	Sign control†
5, 6, 7	I	Optional index increment
0, 1, 2, 3	D	Delimiter control
Any	X	Defined in individual instruction description

†The operand flow chart (figure C-1) illustrates the order of operations when sign control is selected.

TABLE C-1. G BIT USAGES FOR VECTOR (VT) INSTRUCTIONS

NOTE

A blank space in the tables indicates that the corresponding G bit does not apply for that instruction and must be a zero.

Function Code	G Bit/Usage								Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
80	E	C	O	B	B	S	S	S	90	E	C	O	B				
81	E	C	O	B	B	S	S	S	91	E	C	O	B				
82	E	C	O	B	B	S	S	S	92	E	C	O	B				
83		C	O	B	B				93	E	C	O	B		S	S	
84	E	C	O	B	B	S	S	S	94	E	C	O	B	B			
85	E	C	O	B	B	S	S	S	95	E	C	O	B	B			
86	E	C	O	B	B	S	S	S	96		C	O	B				
87		C	O	B	B				97		C	O	B				
88	E	C	O	B	B	S	S	S	98	E	C	O	B				
89	E	C	O	B	B	S	S	S	99	E	C	O	B				
8B	E	C	O	B	B	S	S	S	9A	E	C	O	B				
8C	E	C	O	B	B	S	S	S	9B	E	C	O	B	B			
8F	E	C	O	B	B	S	S	S	9C		C	O	B				

TABLE C-2. G BIT USAGES FOR SPARSE VECTOR (SV) INSTRUCTIONS

Function Code	G Bit/Usage								Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
A0	E	X	X	B	B	S	S	S	A8	E	X	X	B	B	S	S	S
A1	E	X	X	B	B	S	S	S	A9	E	X	X	B	B	S	S	S
A2	E	X	X	B	B	S	S	S	AB	E	X	X	B	B	S	S	S
A4	E	X	X	B	B	S	S	S	AC	E	X	X	B	B	S	S	S
A5	E	X	X	B	B	S	S	S	AF	E	X	X	B	B	S	S	S
A6	E	X	X	B	B	S	S	S									

TABLE C-3. G BIT USAGES FOR BRANCH (BR) INSTRUCTIONS

Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7
B0	X	X	X	X	X	X	X	X
B1	X	X	X	X	X	X	X	X
B2	X	X	X		X	X	X	X
B3	X	X	X		X	X	X	X
B4	X	X	X		X	X	X	X
B5	X	X	X		X	X	X	X

NOTE

Instructions 2F, 32, and 33 are not listed in this table because their G bits are used for control purposes and do not follow the bit definitions at the beginning of this section.

TABLE C-4. G BIT USAGES FOR VECTOR MACRO (VM) INSTRUCTIONS

Function Code	G Bit/Usage								Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
B7†	E				B		X	X	D1	E	C	O					
B8	E	C	O						D4	E	C	O	B	B			
BA††	E						X	X	D5	E	C	O					
C0	E	C		B	B				DA	E	C						
C1	E	C		B	B				DB	E	C						
C2	E	C		B	B				DC	E	C						
C3	E	C		B	B				DE	E	C	O	B				
D0	E	C	O	B	B				DF	E	C	O					

†This instruction is undefined if G bits 4 and 6 are both set, or if G bits 6 and 7 are both set.

††This instruction is undefined if G bits 6 and 7 are both set.

TABLE C-5. G BIT USAGES FOR NONTYPICAL (NT) INSTRUCTIONS

Function Code	G Bit/Usage								Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
B9	E		X	X	X				C8	E	C	X					
BB	E			B	B				C9	E	C	X					
BC	E	X							CA	E	C	X					
BD	E			B	B			X	CB	E	C	X					
C4	E			B	B				CF	E				B	S	S	S
C5	E			B	B				D8	E	C				S		
C6	E			B	B				D9	E	C				S		
C7	E			B	B				DD	E							

TABLE C-6. G BIT USAGES FOR STRING (ST) INSTRUCTIONS

Function Code	G Bit/Usage								Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
D6	← DESIGNATOR →								F8	D	D	D	D		I		I
D7	D	D			X	I			F9	D	D	D	D		I		I
EA	← MASK →								FB	X	X						
EB	← DESIGNATOR →								FC	X	X						
EC	← MODULUS →								FD	D	D				I	I	
ED	← MODULUS →								FE	← DESIGNATOR →							
EE	D	D	D	D		I		I	FF	← DESIGNATOR →							
EF	D	D				I											

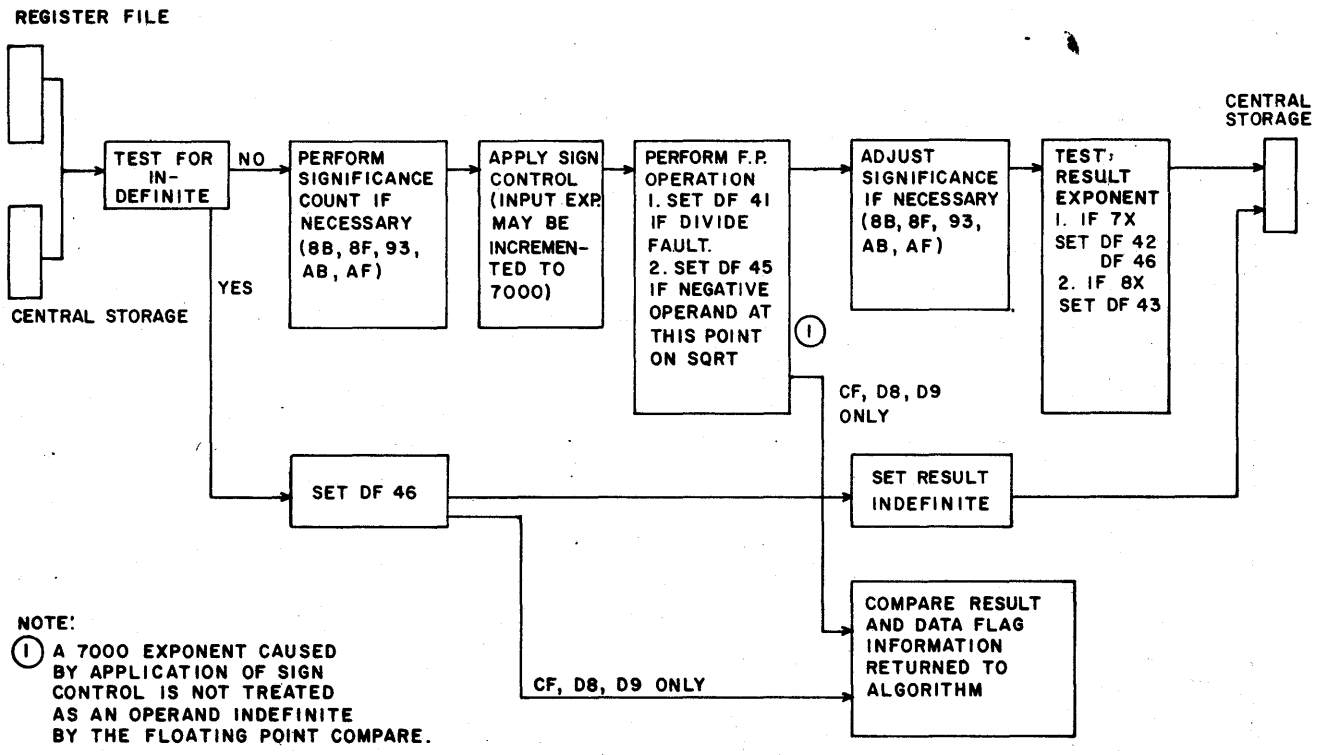


Figure C-1. Operand Flow For Instructions Having Sign Control

INSTRUCTION TERMINATING CONDITIONS

For instructions which terminate upon exhausting the length of a data field, data string or a vector: if that item is exhausted prior to the first operand fetch, the instruction becomes a no-op; no data is fetched and no data flags are altered.

The following paragraphs and tables address the termination of multiple operand instructions. Sparse vector instructions terminate as follows:
Sparse vector instructions terminate when vector Z (the result order vector) is exhausted. If the Z designator is zero or if the Z length is zero, no data flags are set and the instruction is a no-op. Zero length or short source order vectors are extended, as required, with zero bits. If vector Z has a nonzero length and the C designator is zero, the results of the instruction are undefined.

For string instruction terminating conditions see the individual instruction descriptions in section 6 of this manual.

The tables are arranged according to the general instruction types and that the instruction codes within that type are grouped, as much as possible, according to common data field terminating conditions.

Note that in the tables, M-zero and N-one designate machine zero and normalized one, respectively. In addition, the availability of a control vector for the result field is (C or Z) designated by a yes or no and in the case of the vector macro or nontypical instructions, the yes condition is followed by an I or O designator if the control vector applies to an input or output, respectively.

TABLE C-7. VECTOR INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field			B Field			C Field		
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if C field is exhausted	C field length initially zero	Control Vector
80, 81, 82 83, 84, 85 86 & 87	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
88, 89, 8B 8C & 8F	Extend	N-One	Extend	Extend	N-One	Extend	Terminate	No-Op	Yes
90, 91, 92 & 93	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes
94 & 95	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
96, 97, 98 99 & 9A	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes
9B	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
9C	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes

TABLE C-8. VECTOR MACRO INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field			B Field			C Field		
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if C field is exhausted	C field length initially zero	Control Vector
B7	Terminate	NA	No-Op	NA	NA	NA	NA	NA	No
B8	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes (O)
BA	Terminate	NA	No-Op	NA	NA	NA	NA	NA	No
C0, C1, C2 & C3	Terminate	NA	No-Op	Terminate	NA	No-Op	NA	NA	Yes (I)
D0 & D4	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes (O)
D1 & D5	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes (O)
DA & DB	Terminate	NA	No-Op	NA	NA	NA	NA	NA	Yes (I)
DC	Terminate	NA	No-Op	Terminate	NA	No-Op	NA	NA	Yes (I)
DE	Extend	N-One	Extend	NA	NA	No-Op	Terminate	No-Op	Yes (O)
DF	NA	NA	NA	NA	NA	NA	Terminate	No-Op	Yes (O)

O = Output vector

I = Input vector

TABLE C-9. TERMINATING CONDITIONS FOR NONTYPICAL (32-BIT FORMAT) INSTRUCTIONS HAVING MULTIPLE OPERANDS

Instruction Code	R Field		S Field		T Field	
	Result if R Field is exhausted	R Field length initially zero	Result if S Field is exhausted	S Field length initially zero	Result if T Field is exhausted	T Field length initially zero
14	NA	No-Op	NA	Zero R bits Skipped	Terminate	No-Op
15 & 16	NA	No-Op	NA	No-Op	Terminate	No-Op
17	NA	No-Op	NA	No-Op	NA	No-Op
18, 1A & 1B	NA	NA	NA	NA	Terminate	No-Op
19	NA	NA	NA	NA	Terminate†	No-Op
1C & 1D	NA	String of all 1's	NA	No-Op	Terminate	No-Op
1E	Terminate†	No-Op	NA	NA	NA	NA
1F	Terminate	No-Op	NA	NA	NA	NA
28 & 29	NA	NA	NA	NA	Terminate†	No-Op
7D	Terminate data transfer to Reg file.	No data transfer to Reg file.	NA	NA	Terminate data transfer from Reg file.	No data transfer from Reg file.

†These instructions may terminate for reasons other than the exhausting of the field length.

TABLE C-10. NONTYPICAL (64-BIT FORMAT) INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field				B Field		Z Field		
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if Z field is exhausted	Z field length initially zero	Control Vector
B9	NA	NA	NA	NA	NA	NA	NA	NA	No
BB, BC & BD	NA	NA	NA	NA	NA	NA	Terminate	No-Op	No
C4, C5, C6 & C7	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	No
C8, C9, CA & CB	Terminate	NA	No-Op	Exit search iteration	NA	Exit search iteration	NA	NA	Yes (O)
CF	Terminate	NA	No-Op	Extend	M-Zero	Extend	NA	NA	No
D8 & D9	Terminate	NA	No-Op	NA	NA	NA	NA	NA	Yes (I)

I = Input vector
O = Output vector

INSTR CODE ↓	37	38	39	41	42	43	45	46	47	53 54 55
00 01 02 03										
04 05 06 07									X	
08 09 0A 0B										
0C 0D 0E 0F										
10 11 12 13			X							
14 15 16 17										
13 19 1A 1B										X
1C 1D 1E 1F										X

INSTR CODE ↓	37	38	39	41	42	43	45	46	47	53 54 55
20 21 22 23								X		
24 25 26 27								X		
28 29 2A 2B										X X
2C 2D 2E 2F										
30 31 32 33										
34 35 36 37										
38 39 3A 3B										
3C 3D 3E 3F										

INSTR CODE	DATA FLAG BITS									53 54
	37	38	39	41	42	43	45	46	47	55
40					X	X		X		
41					X	X		X		
42					X	X		X		
43										
44					X	X		X		
45					X	X		X		
46					X	X		X		
47										
48					X	X		X		
49					X	X		X		
4A										
4B					X	X		X		
4C				X	X	X		X		
4D										
4E										
4F				X	X	X		X		
50								X		
51								X		
52								X		
53						X	X	X		
54					X	X		X		
55					X			X		
56										
57										
58										
59					X	X		X		
5A										
5B										
5C						X		X		
5D						X		X		
5E										
5F										

INSTR CODE	DATA FLAG BITS									53 54
	37	38	39	41	42	43	45	46	47	55
60					X	X		X		
61					X	X		X		
62				X	X		X			
63										
64					X	X		X		
65					X	X		X		
66					X	X		X		
67										
68					X	X		X		
69					X	X		X		
6A										
6B					X	X		X		
6C				X	X	X		X		
6D										
6E										
6F				X	X	X		X		
70								X		
71								X		
72								X		
73						X	X	X		
74					X	X		X		
75					X			X		
76					X	X		X		
77					X	X		X		
78										
79					X	X		X		
7A										
7B										
7C										
7D										
7E										
7F										

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
80					X	X		X		
81					X	X		X		
82					X	X		X		
83										
84					X	X		X		
85					X	X		X		
86					X	X		X		
87										
88					X	X		X		
89					X	X		X		
8A										
8B					X	X		X		
8C				X	X	X		X		
8D										
8E										
8F				X	X	X		X		
90								X		
91								X		
92								X		
93						X	X	X		
94					X	X		X		
95					X			X		
96					X	X		X		
97					X	X		X		
98										
99					X	X		X		
9A										
9B										
9C						X		X		
9D										
9E										
9F										

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
A0					X	X		X		
A1					X	X		X		
A2					X	X		X		
A3										
A4					X	X		X		
A5					X	X		X		
A6					X	X		X		
A7										
A8					X	X		X		
A9					X	X		X		
AA										
AB					X	X		X		
AC				X	X	X		X		
AD										
AE										
AF				X	X	X		X		
B0								X	X†	
B1								X	X†	
B2								X	X†	
B3								X	X†	
B4								X	X†	
B5								X	X†	
B6										
B7										
B8										
B9										
BA										
BB										
BC										
BD										
BE										
BF										

† G bit 1 = 1.

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
C0	X							X		
C1	X							X		
C2	X							X		
C3	X							X		
C4								X		
C5								X		
C6								X		
C7								X		
C8								X		
C9								X		
CA								X		
CB								X		
CC										
CD										
CE										
CF								X		
D0						X		X		
D1						X		X		
D2										
D3										
D4						X		X		
D5					X	X		X		
D6	X									
D7	X									X
D8								X		X
D9								X		X
DA					X	X		X		
DB					X	X		X		
DC					X	X		X		
DD					X	X		X		
DE					X	X		X		
DF					X	X		X		

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
E0			X							
E1			X							
E2			X							
E3			X							
E4		X	X							
E5		X	X							
E6		X	X							
E7		X	X							
E8										X
E9		X								X
EA										
EB		X								X
EC										X
ED										X
EE										
EF										X
F0										X
F1										X
F2										X
F3										X
F4										X
F5										X
F6										X
F7										X
F8										
F9										
FA			X							
FB										
FC		X								
FD										X
FE	X									
FF	X									

COMMENT SHEET

MANUAL TITLE CDC CYBER 200 Model 203 Computer System
Hardware Reference Manual

PUBLICATION NO. 60256010 REVISION 02

FROM: NAME: _____

BUSINESS ADDRESS: _____

COMMENTS:

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

CUT ALONG LINE

AA3419 REV. 1/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

OLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

OLD

FOLD

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION