MICROPROGRAMMING THE RCA SPECTRA 70/ MODEL 45

9.1 INTRODUCTION

The RCA Spectra 70/Model 45 is a general purpose, medium-scale processor capable of accepting a wide variety of industrially-accepted codes and programming language. This multilingual system can be used to solve a wide variety of business, scientific, communication, and real-time application problems. Architecturally, it is compatible with the other processors in the Spectra 70 product line, which includes the 70/15, 70/25, 70/35, 70/45, 70/55, and 70/60. However, its internal organization and data flow, its timing, its microprogramming and microprogramming aids are considerably different from all other processors.

The RCA Spectra 70 is architecturally similar to the IBM System/360. Both systems apply a common code structure based on the Extended Binary Code Decimal Interchange Code (EBCDIC) of eight bits plus parity. The standard unit of storage in both systems is called a byte. Both families offer the facilities for using the American Standard Code for Information Interchange (ASCII). Fixed length data of 8, 16, 32, or 64 bits may be processed. Variable-length data of up to 256 characters (bytes) can be processed by both systems.

Both systems have similar and complex instruction formats 2, 4, or 6 bytes

wide, signaling register to register, register to storage, and storage to storage information interchange respectively. Each instruction begins with the operation code byte, followed first by one or two general purpose registers, then by memory addresses as required.

Both systems have a base register which can be indexed by any of the general purpose registers.

The system interrupts, their classification into five classes (program interrupts, external interrupts, supervisor call interrupts, and input/output interrupts), and the priorities of handling them are also similar.

The microprogrammed processors in both product lines apply similar flowchart microprogramming language and a rigidly-formatted instruction box. Both systems apply an encoded control ROS word organization with a built-in sequencing and decision-logic specification.

Other architectural features such as the sixteen 32-bit general purpose registers (GPR's), the four 64-bit floating-point registers (FPR's), the memory-protection features, and other similarities are clearly documented and easily discernible by reading the principles-of-operation manuals for both systems.*

These architectural similarities mask considerable differences in their respective design implementations, circuit technology, and ROS implementation. They benefit the reader since architectural similarities permit the reader to focus immediately on their respective hardware and microprogramming structures.

Some of the more noticeable dissimilarities in the design and implementations of the two systems are:

1. The Spectra 70/45 is a single bus system. Its data flow, its arithmetic and logic box (ALB), and its three memories all communicate with each other over one 18-bit-wide data bus (DB). This must be contrasted with the System/360 Models 40 or 50, with their multidata-path structures which communicate over two or more data buses of different widths.

Another difference in the data flow of the two systems is the abundance of general purpose hardware registers with extensive gating in the System/360 versus the minimum hardware registers in the Spectra 70/45. This is not to suggest that the performance of the 70/45 suffers from the apparent lack of hardware registers and bussing, or that the complexity of data flow of the System/360 by itself makes it in any way superior to the Spectra 70, for the two systems are judged by their performance throughput, turn-around, response time, and other parameters external to the respective systems data-flow structure.

2. The System/360 architecture provides only *one* set of sixteen 32-bit-wide GPR's plus *four* double-word (64-bit) FPR's, all in the high-speed local store. This single set of GPR's services all the demands of the system. When an

*IBM Systems/360 Principles of Operations Manual, Form #A22-6821-4, File #S360-01, and RCA Spectra 70, Systems Information Manual, Doc. 70-00-601, Dec. 1964.

interrupt occurs, the GPR's are stored away and loaded with the new data in the process of shifting from one mode of operation to the next. This implies an expenditure of time necessary to STORE MULTIPLE and LOAD MULTIPLE. This time was saved in the Spectra 70/45 by providing *four* complete sets of GPR's, each set assigned to service one of the four system states.* Addressing of the scratch pad memory (SPM) is in many cases program-state-dependent, allowing the same read-only storage (ROS) algorithm to operate with different physical registers, depending on the operating state. The Spectra 70/45 also stores these four sets of sixteen 32-bit GPR's plus one single set of four double-word FPR's, all in a high-speed destructive read-storage unit—the SPM or fast memory (FM). The SPM is also used as a substitute for the number of hardware registers present in System/360. This use of SPM for intermediate storage and for the supply of data to the ALB is justified on the basis of timing, as we will see later. However, if the two operands are in SPM, two machine cycles are needed to perform any operation through the ALB which involves both operands.

3. The third difference between the two system implementations is not as obvious as the first two. This deals with the microprogramming of the two systems. The System/360 is micro-order-oriented. The micro-orders are primarily organized around the control gate that permits information to flow from one storage and functional station to the next. As such, the System/360 is not highly specified. That is to say, almost every new conceivable micro-instruction can be structured from the available micro-orders, provided that the timing and concurrency of information-gating is permitted. This flexibility by itself does not imply or ensure efficient microprogrammed control. This is to be contrasted with the Spectra 70/45 microinstruction structure [RCA's term for microinstruction is elementary operation (EO)]. The RCA ROS words are highly structured and EO-oriented. Its ROS [or read-only memory (ROM)] word is divided into fields. Each field when decoded specifies an action or specifies a toggle setting but not the state of unique gates. These fields may have a number of different interpretations, depending on the setting of another field. That is to say, the decoding of the Spectra 70/45 fields are, in general, interdependent. Therefore one can conclude that the Spectra 70/45 is highly specified, and its microprogramming, because of its sophisticated ROS word structure, is difficult to master.

The highly specified structure of the Spectra 70 system does not in any way imply an inferior design. It merely points out that the system designers chose to optimize the data processing function by a highly organized, interdependent EO structure. The intention here in contrasting the Spectra 70 with other equipment is once again to underscore the flexibility of microprogramming. The designers of the IBM System/360 chose a general, simple micro-

*The four operating states provided in Model 45 are: the program state, the executive state, the interrupt state, and the machine state. The System/360 has only two states: the program state and the supervisor state.

order structure, and the Spectra 70 designers chose a more sophisticated interdependent EO ROS structure. The appropriateness of each design must be viewed in the context of the total system design and the constraints and trade-offs that were imposed on the respective system designers.

4. There are a number of other dissimilarities between the two systems that will become apparent after one gains more insight into the microprogramming of the two systems. These differences, although they will not be discussed here, will include: the concurrency of operations; the high-speed local storage mapping and addressing; the microinstruction organization and ROM addressing and branching; the extent of hardware facilities, toggles, and triggers to share the control with the microroutines in the spectra 70/45 and others.

9.2 70/45 HARDWARE (FIG. 9.1)

The 70/45 has three memories: the main memory (MM), the SPM or fast memory (FM) (called local storage in the IBM System/360), and the ROM. The first two are used for data and the last for microinstruction storage. The 70/45 has a number of hardware registers, plus an eight-bit-wide arithmetic and logic box (ALB). All these functional units are interconnected by a single data bus two bytes wide.

9.2.1 The Main Memory

The main memory (MM) is expandable to 262,144 bytes. It has a 1.44 microsecond cycle time and a 16-bit word storage structure representing two bytes; a parity bit is stored with each byte in the MM. The DB is 16 bits wide and normally transmits either half of the data register (DR) directly to the utility register (UR), intermediate register (IR), main memory address register (MMAR), or the main memory register (MMR), thus providing a complete interchange flexibility among the ALB and the two data memories. In addition, a second data bus (DBA) is provided for transmittal of the two most significant bits of 18-bit memory address to MMAR. This second bus is driven primarily by the DR and B register (BR). This model 70/45 has a memory-protection feature to prevent destruction of information by programming or input devices. This feature consists of a set of registers that are constantly scanned during instruction-addressing to ascertain that the address falls within the allowable limits. If a violation is detected, an interrupt condition is set, and the appropriate action is taken.

The following describes a typical operation requiring data to be read from MM for transfer to the UR using an address accessed from the SPM. The address accessed from the SPM is transferred to the MMAR via the DR and is incremented during its regeneration into the SPM during the first microinstruc-

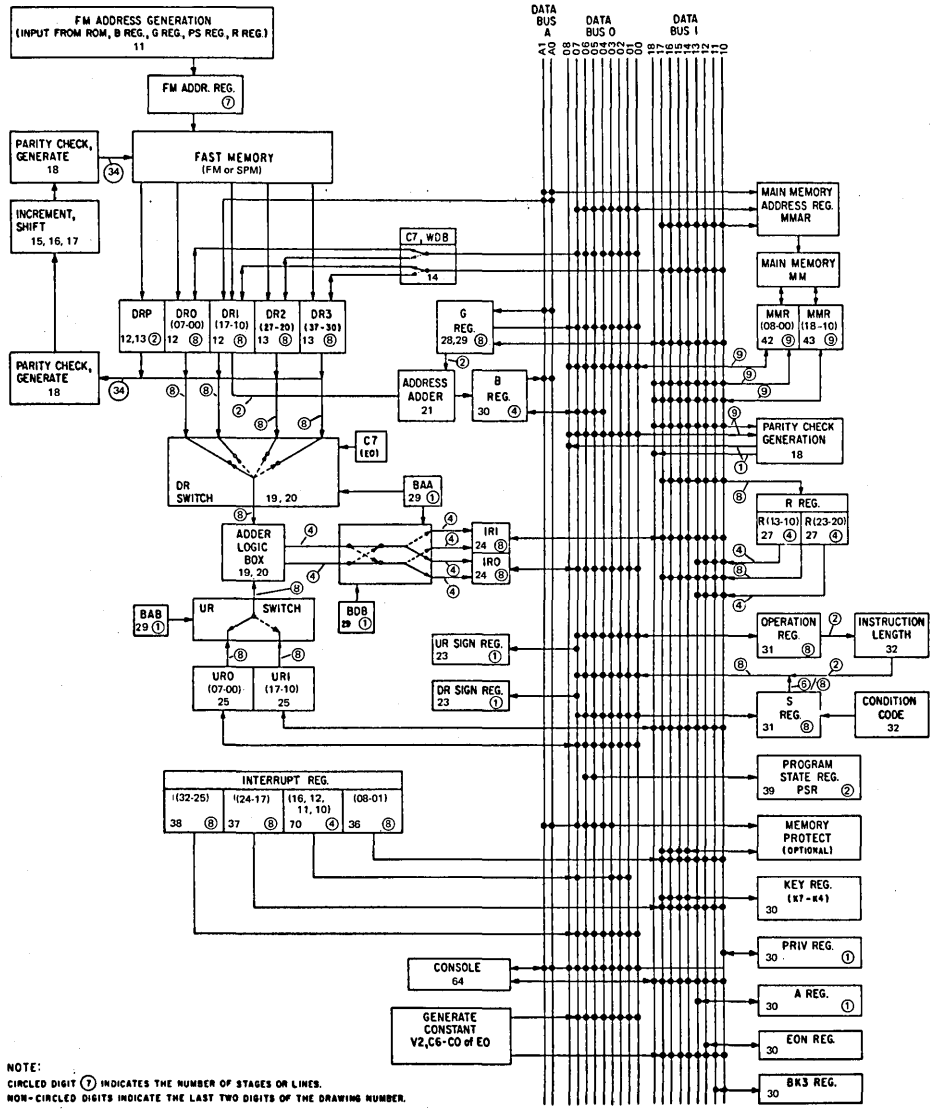


Figure 9.1 Spectra 70/45 block diagram.

tion. The read cycle takes three machine cycles. As far as a main memory read (MMR) is concerned, no useful work is accomplished during the second microinstruction. This second EO cycle can be used to perform other functions not related to the MM data path. If no other function is needed during this second EO cycle, a memory address register W (MARW) micro-order can be specified which causes the machine to wait between the first and third EO time, thus saving one word of ROS. At the beginning of the third microinstruction, the contents of the addressed location in MM are transmitted to the MMR. The

MMR is specified as the source and the UR as the destination during the third microinstruction. This example will be expanded and the regenerate cycle will be discussed in a later section on the 70/45 timing.

9.2.2 The Scratch Pad Memory (SPM)

The 70/45 utilizes a high-speed SPM as an integral part of the data flow. It is used as a substitute for one hundred and twenty-eight 32-bit-wide flip-flop registers. It provides for the storage of four sets of sixteen 4-byte-wide GPR's plus one set of four 8-byte-wide FPR's, input and output control registers, and other utility functions such as storing the source- and destination-memory-address registers. Each of the four system states is also provided with a unique set of registers in SPM as well as their instruction counter, interrupt mask register, and interrupt status register. An SPM map is shown in Table 9.1.

The memory is operated in a linear select fashion, with a fullword access at 120 nanoseconds and an overall cycle time of 480 nanoseconds. This timing permits three SPM cycles to be executed for each MM cycle. This will be discussed in a later section devoted to the systems' timing.

When the system is placed in the emulation mode (EM), the SPM contains several tables (such as a decimal-to-binary-address table), and a mapping of hardware registers of the emulated machines required by this mode.

The scratch pad memory address register (SPMAR) is driven directly from the applicable registers: base address register (BR), general purpose register (GR), program status register (PSR), R register (RR), status register (SR), and the ROM. See Fig. 9.1. This causes a word from the SPM to be read out into the DR and passed through the incrementor and shifter during the same machine cycle. In addition, the DR with its new contents can act as the source register for a transfer to any other register on the data bus, or to the ALB to be operated upon. Note that the DR is 32 bits wide and that two switches (the DR switch and the WDB (word driver bit) toggle switch) are built into the data flow to allow for the selection of one of the four bytes, to be gated into or out of the DR register.

The shift incrementor network in the SPM regeneration path has the ability to shift a 32-bit word one place, either right or left, and to increment or decrement the word by one or two bits. If no shift and increment or decrement microcontrol are specified by the microinstruction, the word read from the SPM will be regenerated unchanged.

9.2.3 The Read-only Memory

The third memory in the 70/45 processor is the read-only memory (ROM). It is part of the control system and stores the sequences of microinstructions used to control the transfer of information, the arithmetic operations, and the machine status switching.

Table 9.1 FAST MEMORY LAYOUT AND REGISTER ASSIGNMENT

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	PROCESSOR UTILITY NO. 1	PROCESSOR UTILITY NO. 2	PROCESSOR UTILITY NO. 3	PROCESSOR UTILITY NO. 4	PROCESSOR UTILITY NO. 5	PROCESSOR UTILITY NO. 6	AAD A ADDRESS REGISTER	BAD B ADDRESS REGISTER	GENERAL PURPOSE REGISTER NO. 8 P4	GENERAL PURPOSE REGISTER NO. 9 P4	GENERAL PURPOSE REGISTER NO. 10 P4	GENERAL PURPOSE REGISTER NO. 11 P4	INTERRUPT MASK REGISTER P4	INTERRUPT STATUS REGISTER P4	PROGRAM COUNTER P4	GENERAL PURPOSE REGISTER NO. 15 (WEIGHT) P4
1	PROCESSOR UTILITY NO. 7	UTILITY NO. 1	I/O CHANNEL REGISTERS – MULTIPLEXOR CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – MULTIPLEXOR CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – MULTIPLEXOR CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – MULTIPLEXOR STATUS REGISTER	I/O CHANNEL REGISTERS – MULTIPLEXOR UTILITY NO. 2	I/O CHANNEL REGISTERS – MULTIPLEXOR UTILITY NO. 3	PROCESSOR UTILITY NO. 8	PROCESSOR UTILITY NO. 9	I/O CHANNEL REGISTERS – SELECTOR NO. 1 CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 1 CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – SELECTOR NO. 1 CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – SELECTOR NO. 1 ASSEMBLY STATUS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 1 UTILITY NO. 2	I/O CHANNEL REGISTERS – SELECTOR NO. 1 UTILITY NO. 3
2	INTERRUPT MASK REGISTER P1	INTERRUPT STATUS REGISTER P1	PROGRAM COUNTER P1	INTERRUPT FLAG REGISTER	INTERRUPT MASK REGISTER P2	INTERRUPT STATUS REGISTER P2	PROGRAM COUNTER P2	GENERAL PURPOSE REGISTER NO. 7 P3	INTERRUPT MASK REGISTER P3	INTERRUPT STATUS REGISTER P3	PROGRAM COUNTER P3	GENERAL PURPOSE REGISTER NO. 11 P3	GENERAL PURPOSE REGISTER NO. 12 P3	GENERAL PURPOSE REGISTER NO. 13 P3	GENERAL PURPOSE REGISTER NO. 14 P3	GENERAL PURPOSE REGISTER NO. 15 (WEIGHT) P3
3	PROCESSOR UTILITY NO. 10	PROCESSOR UTILITY NO. 11	I/O CHANNEL REGISTERS – SELECTOR NO. 2 CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 2 CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – SELECTOR NO. 2 CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – SELECTOR NO. 2 ASSEMBLY STATUS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 2 UTILITY NO. 2	I/O CHANNEL REGISTERS – SELECTOR NO. 2 UTILITY NO. 3	PROCESSOR UTILITY NO. 12	PROCESSOR UTILITY NO. 13	I/O CHANNEL REGISTERS – SELECTOR NO. 3 CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 3 CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – SELECTOR NO. 3 CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – SELECTOR NO. 3 ASSEMBLY STATUS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 3 UTILITY NO. 2	I/O CHANNEL REGISTERS – SELECTOR NO. 3 UTILITY NO. 3
4	GENERAL PURPOSE REGISTER NO. 0 P2	GENERAL PURPOSE REGISTER NO. 1 P2	GENERAL PURPOSE REGISTER NO. 2 P2	GENERAL PURPOSE REGISTER NO. 3 P2	GENERAL PURPOSE REGISTER NO. 4 P2	GENERAL PURPOSE REGISTER NO. 5 P2	GENERAL PURPOSE REGISTER NO. 6 P2	GENERAL PURPOSE REGISTER NO. 7 P2	GENERAL PURPOSE REGISTER NO. 8 P2	GENERAL PURPOSE REGISTER NO. 9 P2	GENERAL PURPOSE REGISTER NO. 10 P2	GENERAL PURPOSE REGISTER NO. 11 P2	GENERAL PURPOSE REGISTER NO. 12 P2	GENERAL PURPOSE REGISTER NO. 13 P2	GENERAL PURPOSE REGISTER NO. 14 P2	GENERAL PURPOSE REGISTER NO. 15 P2
5	PROCESSOR UTILITY NO. 14	PROCESSOR UTILITY NO. 15	I/O CHANNEL REGISTERS – SELECTOR NO. 4 CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 4 CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – SELECTOR NO. 4 CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – SELECTOR NO. 4 ASSEMBLY STATUS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 4 UTILITY NO. 2	I/O CHANNEL REGISTERS – SELECTOR NO. 4 UTILITY NO. 3	PROCESSOR UTILITY NO. 16	PROCESSOR UTILITY NO. 17	I/O CHANNEL REGISTERS – SELECTOR NO. 5 CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 5 CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – SELECTOR NO. 5 CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – SELECTOR NO. 5 ASSEMBLY STATUS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 5 UTILITY NO. 2	I/O CHANNEL REGISTERS – SELECTOR NO. 5 UTILITY NO. 3
6	GENERAL PURPOSE REGISTER NO. 0 P1	GENERAL PURPOSE REGISTER NO. 1 P1	GENERAL PURPOSE REGISTER NO. 2 P1	GENERAL PURPOSE REGISTER NO. 3 P1	GENERAL PURPOSE REGISTER NO. 4 P1	GENERAL PURPOSE REGISTER NO. 5 P1	GENERAL PURPOSE REGISTER NO. 6 P1	GENERAL PURPOSE REGISTER NO. 7 P1	GENERAL PURPOSE REGISTER NO. 8 P1	GENERAL PURPOSE REGISTER NO. 9 P1	GENERAL PURPOSE REGISTER NO. 10 P1	GENERAL PURPOSE REGISTER NO. 11 P1	GENERAL PURPOSE REGISTER NO. 12 P1	GENERAL PURPOSE REGISTER NO. 13 P1	GENERAL PURPOSE REGISTER NO. 14 P1	GENERAL PURPOSE REGISTER NO. 15 P1
7	FLOATING-POINT REGISTER NO. 0		FLOATING-POINT REGISTER NO. 2		FLOATING-POINT REGISTER NO. 4		FLOATING-POINT REGISTER NO. 6		PROCESSOR UTILITY NO. 18	PROCESSOR UTILITY NO. 19	I/O CHANNEL REGISTERS – SELECTOR NO. 6 CHANNEL ADDRESS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 6 CHANNEL COMMAND REGISTER II	I/O CHANNEL REGISTERS – SELECTOR NO. 6 CHANNEL COMMAND REGISTER I	I/O CHANNEL REGISTERS – SELECTOR NO. 6 ASSEMBLY STATUS REGISTER	I/O CHANNEL REGISTERS – SELECTOR NO. 6 UTILITY NO. 2	I/O CHANNEL REGISTERS – SELECTOR NO. 6 UTILITY NO. 3

* Word Address is in Hexadecimal; e.g., 2A = Program Counter P2.

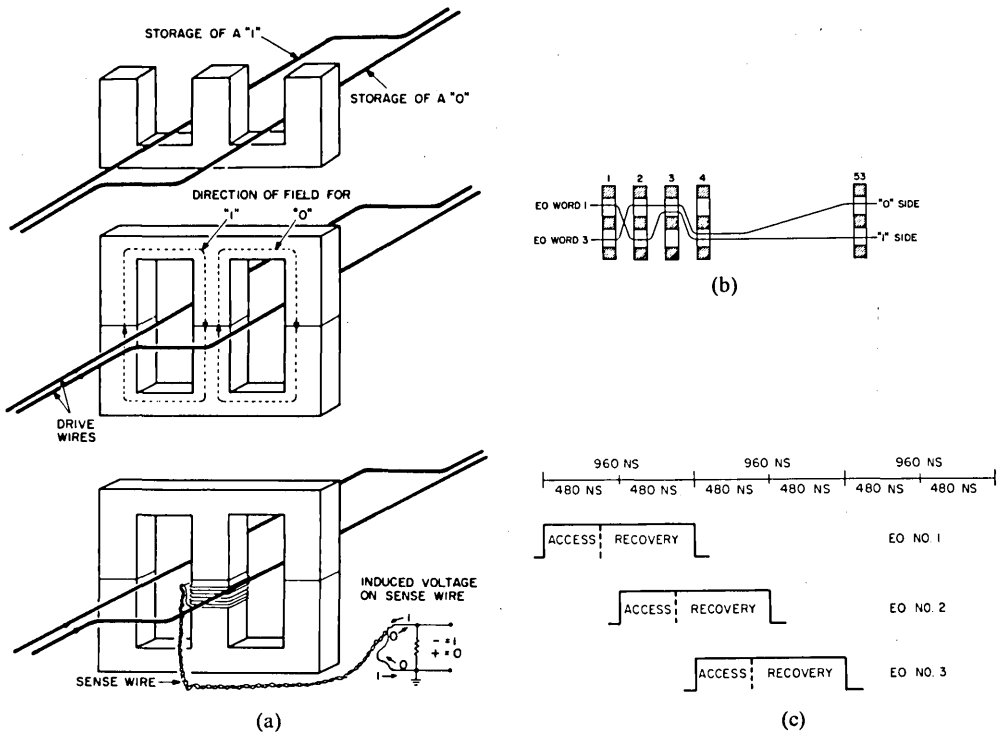


Figure 9.2 E-core ROM technology. (a) E-core wiring; (b) wiring of the elementary operation (EO) words; (c) ROM cycle time.

Figure 9.2 illustrates the basic ROM technology. It uses an E core transformer-type ROM. The bit pattern stored in a ROM word (EO) is determined by the way each word-drive is threaded through each E core that corresponds to one bit in the ROM word. This transformer read only storage (TROS) technology provides bipolar signal outputs. A drive-line threaded through one leg of the E core would produce an output signal of a polarity opposite to that produced if the same word-drive-line is threaded through the other leg of the E core. This difference is sensed by a differential sense amplifier whose output represents the bit stored. The decision as to which polarity of the output signal represents a 1 or a 0 is arbitrary.

The ROM is organized into units of 2048 words of 56 bits each. One 2048-word-bank is used for the 70/45 standard architectural implementation and up to two additional ROM banks may be supplied for the emulator's microcode.

The ROM bank is organized into two separate memory stacks (each memory stack containing 1024 56-bit words) which generally operate on an alternating basis. This organization (Fig. 2.3) effectively halves the access time, by overlapping the access time of one with the recover time of the other half. Because of conditional branches, this microprogramming restriction is

sometimes dispensed with, and sequential addresses are accessed from the same bank. This would imply an automatic 480-macrosecond time-delay penalty.

The 12-bit ROM address register (ROMAR) is used to address ROM. The input to the ROMAR comes from the address generator, the regenerate address (REGAD), or multiplexor regenerator address (MREGAD) registers. (See Fig. 9.3.)

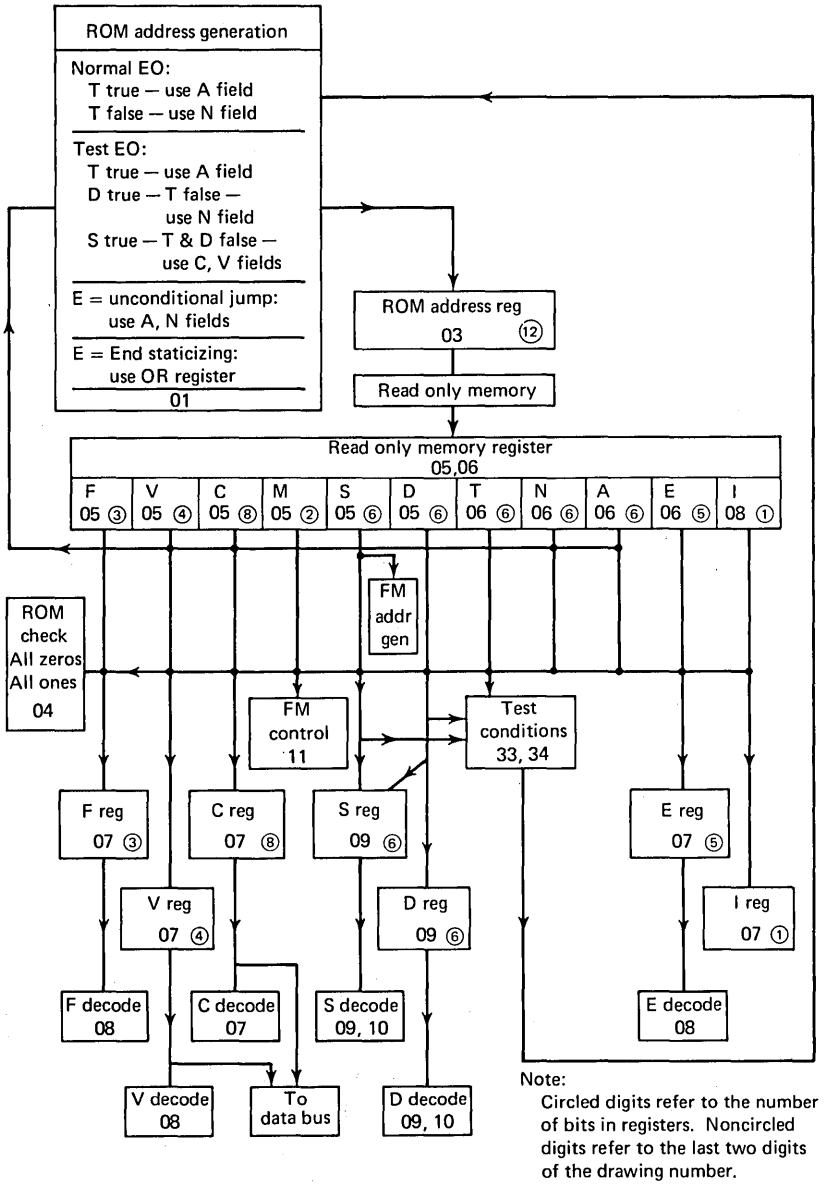


Figure 9.3 Spectra 70/45 UI format and control.

The address generator uses fields of the current microinstruction to generate the address of the next microinstruction. Where only one field is used, it is placed in the least significant section of the 12-bit address (the 6 most significant bits remaining from the current address). The contents of the operation register (OR) may also be used for addressing. The REGAD register normally holds the ROM address that has also been sent to the ROMAR.

The ROM word (56 bits) is organized into eleven fields: F, V, C, M, S, D, T, N, A, E, and I. Some of these fields are logically grouped: F, V, and C belonging to one logical group; M, S, and D belonging to another logical group; and T, N, and A belonging to a third one. The remaining two fields, E and I, are independent functions.

All fields of the ROM word are available in every microinstruction and are read into the ROM DR. In some cases, a field may be multipurpose and have different functions at different times. The alphabetical letter used to indicate the field is an abbreviation, and the full word will give an indication of the function:

F = Function (of microinstruction)
V = Variation (of function)
C = Counter-control (also used as a constant and for addressing)
M = Memory control (controls SPM operation)
S = Source (for DB transfers; also used to specify conditions in special test micro-orders and to address SPM)
D = Destination (for DB transfers; also used to specify conditions in special test microinstructions)
T = Test (specifies test conditions in all microinstructions)
N = Normal (address of next microinstruction, depending on the test results or E field)
A = Alternate (address of next microinstruction, depending on the test results or E field)
E = Exception (specifies checking for validity of addresses or data; also used to specify unconditional jumps)
I = Inhibit (inhibits an I/O servicing break prior to executing the next microinstruction).

These functional field groups are designed to operate as follows:

FVC	MSD	TNA	E	I

1. *The Function Control Fields (F, V, C):* The F, V, and C fields are the most complex to specify because they define a variety of control functions which include the ALB data path, the counters, shift and increment control, and toggle settings. Some of the functions specified by F, V, and C fields can be performed concurrently. In general the F defines the basic function to be performed, the V field further defines the variations and/or further specifications of that function, and the C is the counter control field.

The F field, for example, can specify either shift or increment. If it is a shift function, the V field specifies right or left shift and the disposition of the bits shifted out of or into the SPM location specified. If the F field specifies an increment function, the V will be used to specify whether the counter is incremented or decremented and the value by which the counter is modified. Another example of the FVC control is to specify the ALB operation. The F field indicates a set, perform, or set and perform operation, whereas the V field specifies one of eight different arithmetic, logic, decimal binary, or transfer through ALB operations.

(field bit positions)

$F = (F)_8 = F(2\text{–}0)$—Function

$V = (V)_{16} = V(3\text{–}0)$—Subfunction (Variation)

$C = (C)_{16} \cdot (C)_{16} =$ C(7–4) for the switch- and toggle-setting control the data flow into and out of ALB, also used to introduce a literal onto DB; concatenated with C (3–0) for the counter initialization and decrement control.

(field radix)

(concatenation)

The reader must note the bit significance of the C (7–4) field.

C(7) controls the ingating and outgoing of the leftmost or rightmost two bytes—to and from the DR register. It also sets the WDB toggle which controls the gating of the two bytes of data on the DB (0, 1) into the right or left half of the four-byte register DR, on transfers to the DR during a perform EO.

The toggles BAA, BAB, BDB are specified by the C(6), C(5), and C(4), respectively. They are set to the state of C(0) if specified.

The RR and GR counters are decremented as specified by C(3) and C(2) and are decremented by 0, −1, −2, −3 which is specified by C(1) · C(0).

2. *C(7), MSD Fields (data transfer) F ≠ 0*

The M, S, and D fields control the transfer of information between the functional units. The M designates the operation of the SPM, and the S and D fields designate the source and destination registers for DB transfers. That is, by using the S and D fields, it is possible to transfer from one register to any other register along the DB. Note that any SPM operation must use the DR so that, if a read from the SPM is specified, DR automatically becomes the source register for any transfer on the DB bus. Similarly, in an SPM write operation, any of the hardware registers can be designated as a source register. The contents of the specified hardware register are transmitted to one or the other half of the DR; the remaining half of the DR is accessed from the specified SPM location, and the full word is written into the SPM.

$M = (M)_4 = M(1\text{–}0)$—FM or SPM control

$S = (S)_4 \cdot (S)_{16} = S(5\text{–}4) \cdot S(3\text{–}0)$—FM address or source register

$D = (D)_4 \cdot (D)_{16} = D(5\text{–}4) \cdot D(3\text{–}0)$—specifies source or destination register

The S and D fields can have a number of different interpretations, depending on the M and F fields of the associated microinstruction (EO).

3. *TNA Fields—Branch Control*

Every EO specifies the address for the next EO to be executed.

The T is a 6-bit field which denotes the specific test conditions to be tested. This width of field permits any one of 63 different combinations of hardware states to be tested. If the test is true (T-true), the A field is the low-order 6-bit portion of the address of the next EO within the block of 64. If the test is a false/no test (T-false), the N field is the low-order 6-bit portion of the address of the next EO. Address is determined prior to execution of the EO.

A: If T-true: $A(5\text{–}0) \rightarrow ROMAR(5\text{–}0) \cdot ROMAR(11\text{–}6)$ remains unchanged

N: If $F \neq 5$ and T-false/$F = 5$. T-false · D-true;
$N(5\text{–}0) \rightarrow ROMAR(5\text{–}0) \cdot ROMAR(11\text{–}6)$ remains unchanged

$T = (T)_4 \cdot (T)_{16} = T(5\text{–}4) \cdot T(3\text{–}0)$—test condition code
$N = (N)_8 \cdot (N)_8 = N(5\text{–}3) \cdot N(2\text{–}0)$—next EO address in ROM if test false
$A = (A)_8 \cdot (A)_8 = A(5\text{–}3) \cdot A(2\text{–}0)$—next EO address if test is true

4. *E Field*

$E = (E)_2 \cdot (E)_{16} = E(4) \cdot E(3\text{–}0)$—exception code

5. *I Field*

$I = (I)_2 = I(0)$—inhibit I/O servicing.

A check for correct operations on all the bits of the ROM is made at the appropriate intervals. The check in effect makes use of two special microinstructions, one consisting of a pattern of all *zeroes* and another consisting of all *ones*. These two patterns cause an error to be indicated when not in check mode. In the 56-bit version, there are three parity bits which are checked on each read cycle.

Certain of the fields (F, V, C, S, D, and I) make use of a second level of registers. It is the output of these operation decoders that generally controls the execution of the appropriate operations in the processor.

When an SPM operation ($M(0) = 1$) is specified, the S field is used for SPM addressing. If the operation is a write, a source is required. The source in this instance is indicated by the D field; the D field bits are transferred to the S decoding logic for the operation. If the source is any of the other registers on the DB, 16 bits from the DB are placed in half of the DR as directed by either C7 of the EO or the WDB (the other half of the DR containing the contents of the SPM word addressed), and the entire 32 bits are then placed in the SPM. In the SPM read cycles, the D field is interpreted normally, and the S field is automatically set to indicate the DR as the source. These operations are defined in the S and D field descriptions.

9.2.4 The Arithmetic and Logic Box (ALB)

The arithmetic and logic portion of the processor operates on eight bits in parallel. Any one of the four bytes in the DR register can be selected and transmitted through the 4/1 DR switch to form one byte input to the ALB. The other inputs to the ALB are taken from the UR, which is two bytes wide, and the selection of the byte to be gated to the ALB is specified in the microinstruction field which sets the byte address B (BAB) switch.

The output of the ALB goes to the 16-bit intermediate register (IR). As indicated on the block diagram (Fig. 9.1), the 8-bit output of the ALB is divided into two 4-bit halves which can be switched as dictated by the byte divider bit trigger (BDB). The 8 bits are then further switched to the appropriate half of the IR under the control of the byte address A(BAA) trigger.

One operand of the ALB function is the byte of the UR selected by the BAB toggle; BAB = 0 selects UR0 while BAB = 1 selects UR1. The other operand is the byte of the DR selected by C7 and the BAA toggle.

If C7 = 0 and BAA = 0, select DR 0
If C7 = 0 and BAA = 1, select DR 1
If C7 = 1 and BAA = 0, select DR 2
If C7 = 1 and BAA = 1, select DR 3

The ALB output is set into the byte of the IR selected by the BAA toggle: BAA = 0 selects IR0, BAA = 1 selects IR1. The transfer paths from ALB to IR are further defined by the contents of the V field of the EO and BDB toggle, so that when $V_2V_1V_0 = 000$, the full 8-bit result from the ALB is transferred to IR0 or IR1, ignoring BDB.

The remaining variations of $V_2V_1V_0$ bits select only four bits of the ALB determined by the BDB toggle [BDB = 0 selects ALB (7–4), BDB = 1 selects ALB (3–0)].

Example: If V(2–0) = 100 and BDB = 1, this would transfer ALB (3–0) to IR (3–0), leaving IR(7–4) unchanged.

The 1-bit DR sign register (DRSR) is set under specific conditions during transfers into the IR. Similarly, the 1-bit UR sign register (URSR) is set under specific conditions during transfers into the UR. The DRSR and URSR are set if the sign is negative. The ALB is designed to perform one of eight operations specified by the microinstruction (the V field). These functions are: binary add, binary subtract, decimal add, decimal subtract, transfer utility register UR, transfer the complement of UR, logical OR, and *exclusive* OR. When the ALB is set to any one of these eight functions, it remains in that mode until it is reset by the microprogram. This modification is specified in the F field. The F field would either set, perform, or set and perform. In order to perform an operation on successive data types, the perform microinstruction will continue operations on the data operands using the *carried-over* setting of the control-

carry and result-zero flops. The control flops will be reinitialized only by another set, or set and perform, microinstruction.

9.2.5 Other Hardware Registers

A number of other registers are shown on the simplified 70/45 data flow in Fig. 9.1. These are all connected to the DB bus for various purposes. They include the following:

General Purpose Register (GR) is an 8-bit decrementing counter used as a general purpose counter, and is also used to address SPM.

Status Register (SR) also has properties peculiar to the Spectra 70 (and should not be used in emulators) in that it contains:

1. the condition code representing the results of previous operations (positive, negative, zero, and overflow) the equivalent of the previous result indicators in earlier processors, and
2. four interrupt masks.

The 8-bit SR is fed from DB0. SR provides the lower order bits on the DB(5–0). The two high-order bits of the 8-bit byte may be provided by the instruction length hardware or the S register (Fig. 9.1). The instruction length information is derived from the two high-order bits of the OR. Two bits of S contain the condition code.

Operation Register (OR) is an 8-bit register normally holding the operation code, with the special property that the individual bits of the register can be tested by EO (except for OR7, which cannot be tested by itself). The OR is also used in the staticizing routine as a portion of the 64-way branch facility.

Base Address Register (BR) is a 4-bit BR used for address-indexing and also during shift operations to supply and receive the end data.

R Register (RR) is an 8-bit general purpose, decrementing counter that may also be split into two independent 4-bit counters.

The BR or RR may also be used in addressing the SPM but, when so used, are more restricted than the GR.

Interrupt Register (IR) is a 32-bit register that records the many internal and external conditions capable of causing a program interrupt; when an interrupt occurs, its cause or reason is automatically set into the register.

Other miscellaneous registers in the system include:

Arithmetic Mode Flip-Flop (AR) is a 1-bit register indicating whether the arithmetic operations are being performed in the ASCII or EBCDIC Code.

Program Status (PSR) is a 2-bit register indicating the current program state of the processor; it is also used in SPM addressing, i.e., FPC → MAR.

Key Register (KR) is a 4-bit register holding the memory protection key and is used with the optional memory protect feature.

Privilege Register (PR) is a 1-bit register used to indicate specific operations that cannot be executed when set. The KR, AR, and PR are operated simultaneously as far as transfers to and from the DB are concerned.

Emulator On Register (EON) is a 1-bit register used to indicate that the operation of the 70/45 is in the emulator mode. (This is used in machines provided with the emulator option.)

Bank 3 Register (BK3) is a 1-bit register used to indicate the correct ROM bank to be addressed. (This is used in machines provided with the emulator option.)

9.2.6 Wired-in Control Functions

The following is a set of wired-in control functions designed to assist the microprogram control specified in the EO:

EDR Flip-Flop. Regardless of the F or V fields of an EO, the EDR flip-flop is reset whenever (BR = 0000), (S field = X1000X) and (I/O INST); namely, BR = 0000, and BR is being used to address the SPM. Resetting of EDR causes the DR operand to appear as all zeros to the ALB. The condition continues until EDR is set by a set function or set and perform EO.

RZ Flip-Flop. The RZ flip-flop is set by the set function, or set and perform EO, and V fields $\neq 0$. It is reset by a perform or set and perform EO and a nonzero result at the output of the ALB.

Sign Flip-Flops. When the destination code calls for setting the sign flip-flops (DRS or URS), the sign box is set if the sign is negative. If E = 6/7, the sign flip-flops are set according the 4-bit decimal sign at the input to the ALB (the DR switch or UR switch output); otherwise, the sign flip-flops are set from DB07.

WDB Toggle (Word Driver Bit). The WDB toggle is set according to C7 only in a set function, perform, or set and perform EO.

Transfer Controls From DB to DR. The WDB toggle controls the transfer from the DB to the DR only if FMCY = 1 in a perform EO. The transfer is controlled by C0 and C1 in a merge EO. Otherwise, the transfer is controlled by C7.

Transfer Controls From DR to DB and ALB. The transfer from DR to the DB is always controlled by C7. The transfer from the DR to the ALB is always controlled by EDR, C7, and BAA.

Final EO True Test Condition. The last EO of an instruction should contain E = 00010 (end instruction) and must contain a test with the true branch leading to staticizing (instruction fetch microroutine). The unconditional jump (E = 00001) should not be used to go to staticizing because various starting signals must be generated.

E-Code Requirements. The E codes for end staticizing (E = 00011) and end instruction (E = 00010) require a true test condition T. The A field must be either 000000 or 000001 to specify a standard even or odd entrance address.

I/O Servicing. Not more than four consecutive EO's (1.92 micros) must be permitted without I/O servicing.

Triggering GR or RR. It is not permissible to transfer into or out of the GR or RR and trigger that counter in the same EO.

BAA, BAB, BDB Triggering. It is not permissible to trigger a BAA, BAB, or BDB toggle in one EO and test that toggle with a test and branch function that drops through in the immediately following EO.

Interrupt Register Bit #21. Bit #21 of the interrupt register (supervisor call instruction interrupt in Spectra 70 operations) will be set if the E field equals 00000011 and the OR equals 0000 1010.

Interrupt Register Bit #32. Bit #32 of the interrupt register (debug mode option of the program control instruction in Spectra 70 operations) will be set if the E field equals 0000 0010 and the OR equals 1000 0010 and RR10 equals one.

Testing of a Destination Register. A register that is used as a destination for a DB transfer may not be tested in the same EO.

9.2.7 Data Transfer

All the above logic and storage facilities are interconnected by the two-byte-wide data bus (DB) (Fig. 9.1). A second two-bits-wide bus called DBA is provided to transmit the two most significant bits of the main memory address into the MMAR (the remaining 16 bits of the address are transmitted over the DB bus). The DB bus can be viewed as two separate byte buses called DB0 and DB1. Two bytes can be gated on the DB from a number of different sources and transmitted as a single 16-bit field or as two different bytes to two different destinations. It is imperative that the reader study carefully the extent of the gating of the contents of the different hardware facilities on this bus and the gating of the information on the bus into the different permissible facilities. This gating, although extensive enough for the functions performed by the Spectra 45, does present some limitations of which the microprogrammer must

be aware. Note, for example, that the OR is fed from DB0; hence, UR1 cannot be transmitted to OR. If a byte is to be transmitted from ALB to OR, the BAA bit control switch must be set to transmit the ALB output byte to OR through UR0. The R register, when used as one eight-bit decrementing counter, communicates with other functions via the DB1 bus. The R register can also be split into two independent four-bit counters, each of which is loaded or unloaded by the four least-significant bits of DB1 [DB (10–3)]. Note also that the G register can be loaded from the DB1 and unloaded to the DB0; therefore a transfer of UR1 to GR presents no problem whereas a direct UR0 to GR transfer is not possible. The reader is urged at this point to examine Fig. 9.1 carefully in order to master the capabilities, limitations, and alternate paths through which the data can be transmitted. He must be familiar with the function of every box in Fig. 9.1, its ingating and outgating capabilities, and the switches that affect this gating.

9.3 70/45 OPERATIONAL TIMING

The 70/45 basic EO time interval (central processing unit (CPU) cycle) is 480 nanoseconds. This is split into four equal time periods (TP1, TP2, TP3, and TP4) of 120 nanoseconds each. One exception to this is in the case of a "SP2" XXX (set and perform twice) or set followed by "PR2" XXX (perform twice) where the following timing sequence occurs: TP1, TP2, TP3, TP4, TP3, TP4. The 'SP2' and 'PR2' are used to process two-byte operands in 1.5 EO. The 70/45 has one clock pulse generator to provide and synchronize the timing of the different register and memory transfers. These timing pulses also synchronize the microprogrammed and the hardware control functions in the system.

Figure 9.4(a,b,c,d) shows the timing relationships between the ALB, SPM, MM, ROM, and other toggle set and reset timing controls. These figures are also helpful in tracing the concurrent action of all the functions involved in any given operation.

9.3.1 SPM—Timing

The SPM has a 480-nanosecond memory cycle, and an access time of 120 nanoseconds. The SPM read command would transfer the SPM word (four bytes) into the DR in the first 120 nanoseconds. The regenerate cycle is not started until 180 nanoseconds later, allowing time for incrementing or shifting the data before writing it back into memory. The actual regeneration time is 180 nanoseconds. Hence the SPM is capable of reading and regenerating (or writing) one word each CPU cycle.

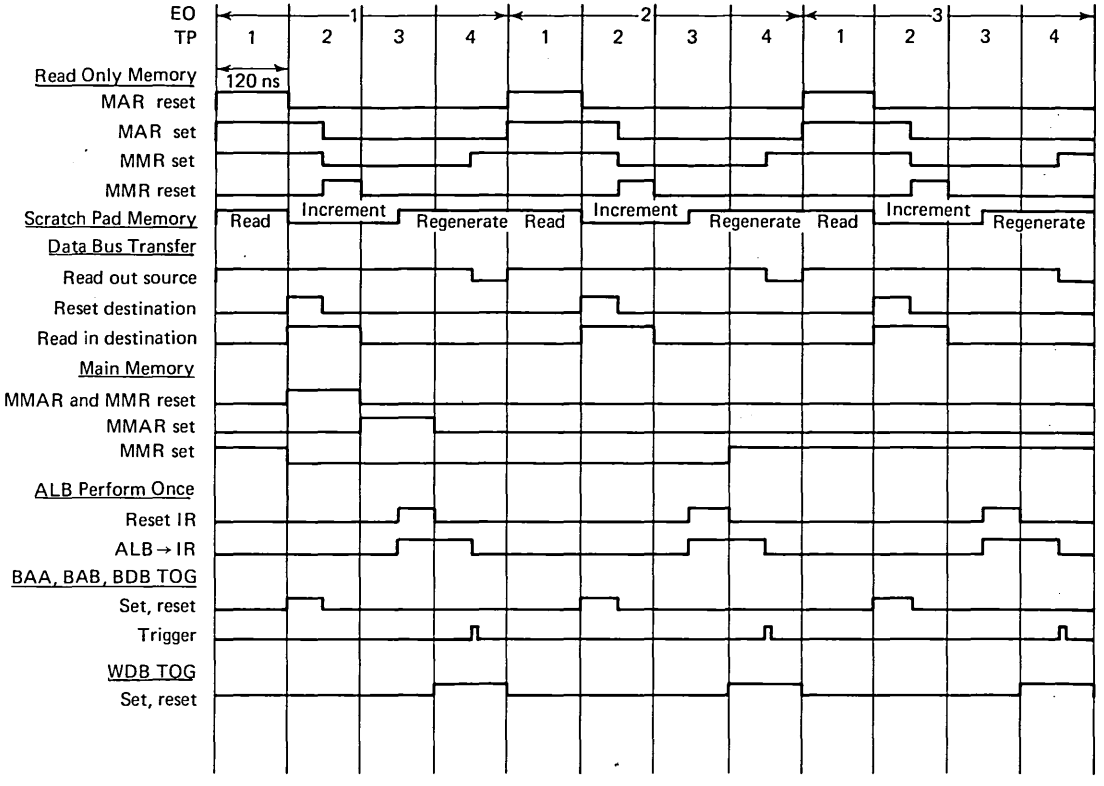


Figure 9.4(a) Spectra 70/45 BPU operational timing diagram.

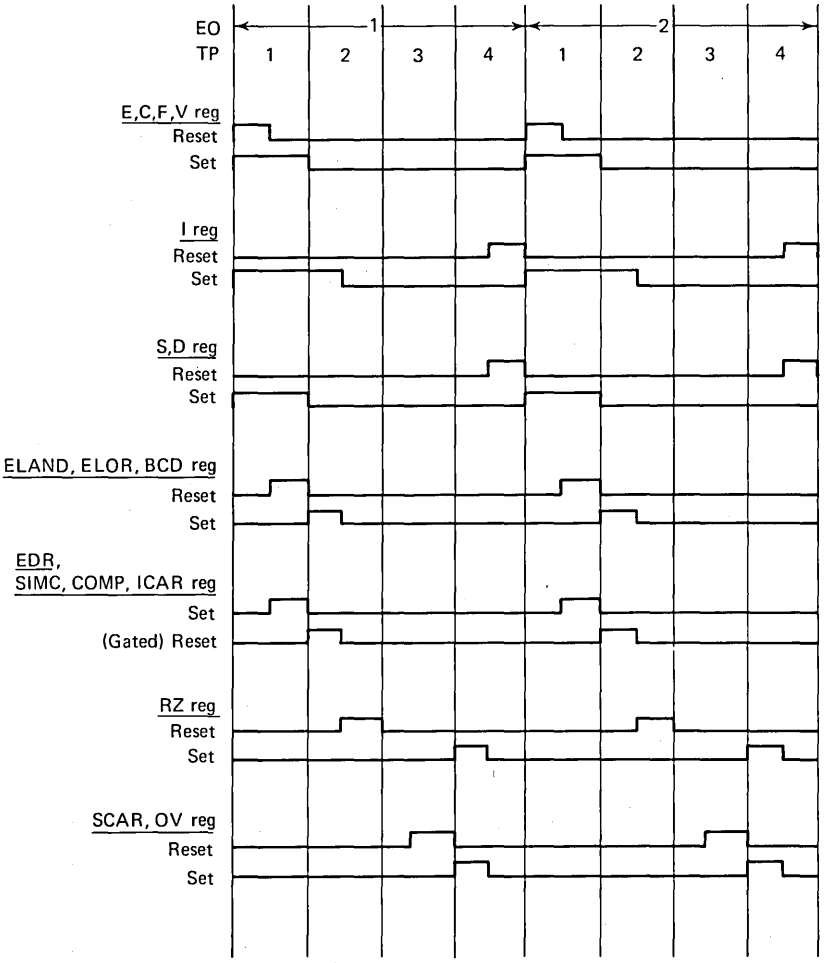


Figure 9.4(b) Spectra 70/45 BPU operational timing diagram.

9.3.2 Main Memory Timing

The main memory (MM) cycle time is 1.44 microseconds. It takes three CPU cycles to read and regenerate the information in MM. In Section 9.2 we considered the data flow and the hardware involved in a typical operation requiring data to be read from MM for transfer to the UR, using an address accessed from SPM. The timing chart in Fig. 9.4(a) shows that the MM address is read out during the first timing point (first 120 nanoseconds) of the first EO cycle. The MMAR and MMR are both reset during the second timing period of the first EO cycle. During the third timing period of the first EO cycle, the MM address read-out from SPM is transferred to MMAR. The

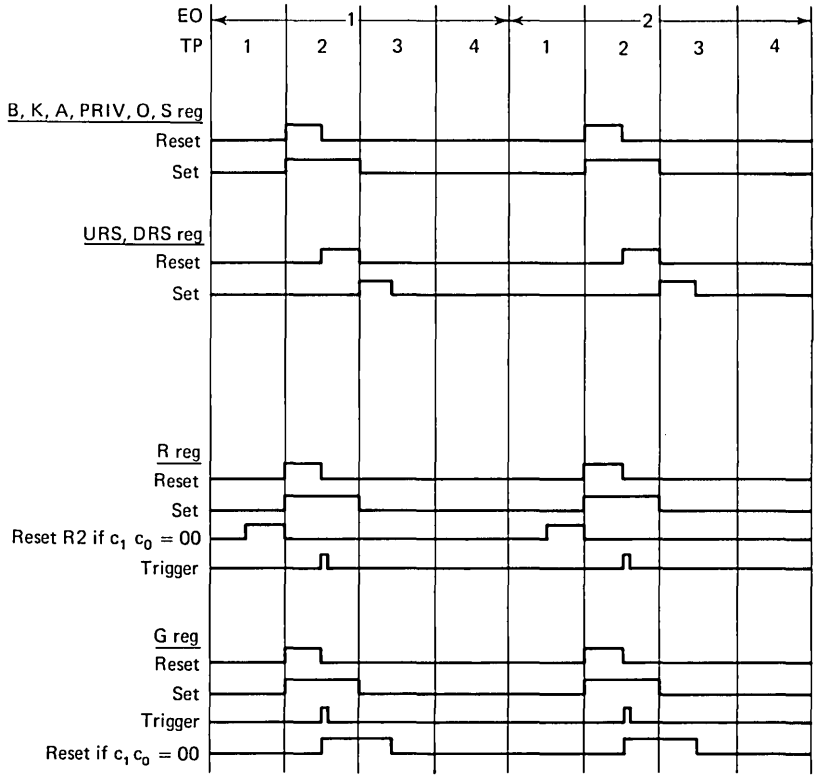


Figure 9.4(c) Spectra 70/45 BPU operational timing diagram.

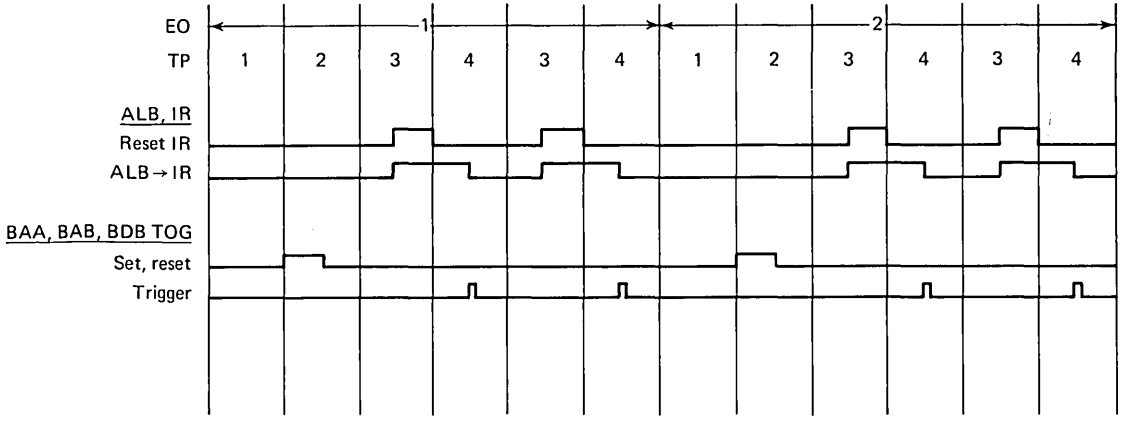


Figure 9.4(d) Spectra 70/45 BPU operational timing diagram.

SPM word is incremented and replaced in the SPM during the first EO cycle. As far as this operation is concerned, no useful work is accomplished during the second EO until its last time period, when the actual MM read-out takes place. If no other useful work is planned for the second EO time, a wait cycle is used. This automatically causes a wait of 480 nanoseconds. The MMR is specified as the source and the UR as the destination during the third EO cycle. As indicated in Fig. 9.4(c), transfers to UR via the DB are accomplished during the second time period of the EO cycle.

Enough time remains in the third EO cycle for an operation through the ALB to the IR with one byte. If a perform-twice EO is specified, time periods three and four are repeated for a total of six time periods rather than the normal four time periods of the EO (i.e., the second execution of the function requires one-half the usual EO time interval since no DB transfer is made). There are two transfers of a byte from ALB to the IR. This byte transfer is controlled by the BAA and BAB toggles and occurs during the fourth timing period of the first EO. The arithmetic function is subsequently performed on the second byte in the DR and UR and the result transferred to the upper half of the IR.

9.3.3 Read-only Memory

The basic ROM cycle time is 960 nanoseconds. The 70/45 uses two banks of ROM that generally operate on an alternating basis to effectively halve the ROM access time to 480 nanoseconds. Therefore, it is possible to access one EO every machine cycle (480 nanoseconds). In case of a jump or branch, it is necessary to access sequential addresses from the same bank, thereby incurring an automatic 480 nanosecond delay as a time penalty. The ROM timing shown in Fig. 9.4(a) is valid only for addressing alternate stacks (even-odd or odd-even). The contents of ROM addressed in the first 120 nanoseconds are available at the end of the EO cycle.

One of the functions that can be specified by the F field is test EO, which permits a test of three conditions for a four-way branch. All other EO's however, permit a test with a two-way branch. In either case, the tests that are specified by the EO are completed before the second basic time period, and at this time period, the ROMAR is set for the next EO. The conditions affecting the test and jump are, therefore, those that have been resolved in previous operations and are not a result of the current EO.

9.3.4 Simultaneity

All operations that are completely independent with regard to data paths and ROM field-bit functions can be carried on simultaneously. Although SPM operations such as increment and shift are normally independent of DB transfers between hardware registers (other than DR), addressing the memory in-

volves the S field and, therefore, concurrent execution of the two functions with one EO is ruled out by the M field, which indicates the SPM operation.

Data bus transfers and ALB operations can be executed in the same EO and can be independent of, or dependent on, each other. The ALB operation takes place after the DB transfer. If the transfer involves either the DR or the UR, the result is involved in an operation through the ALB. If the transfer does not involve either register, the transfer and the ALB operation are independent of each other.

If an ALB operation is specified in one EO together with a bus transfer from the IR, the data transferred from the IR is the data present from the previous operation, and the data in the IR at the end of the operation is the result of the current ALB operation. If an ALB operation is specified on one EO together with a bus transfer *to* the IR, the bus transfer to the IR takes place first and then the IR receives the output of the ALB.

The increment/shift function provides a facility for shifting or incrementing a value as it is written in SPM. The word being shifted or incremented need not have previously been in SPM, but may be operated upon as it is written.

The merge EO function will occur before the incrementing. It provides the only way to modify a single byte of the DR and leave the remaining three undisturbed. Normally, either the upper or the lower half of the DR is loaded from the DB. The merge EO provides for clearing and setting either the DR0 or the DR1 byte from the corresponding bus without altering any other bytes within the DR. Figure 9.5 illustrates the execution time of an EO which

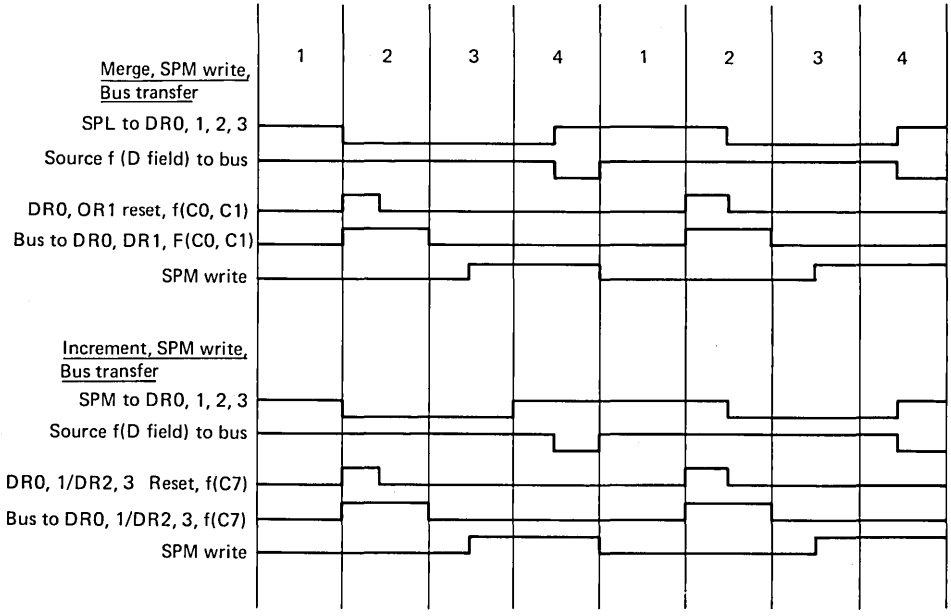


Figure 9.5 Spectra 70/45 EO execution timing.

includes the merge/increment operation with SPM write and DB transfer to the DR. During the first timing period (first 120 nanoseconds), the addressed SPM word is accessed into the DR. Also during the same time, data is put in the DB from the register specified by the destination field. During the second time period in the EO cycle, part of the DR is reset, then is set by the bus. Data in the DR is then transferred to the same location in the SPM from which the original word was read. On the way it is incremented as required by the EO.

9.4 MICROPROGRAMMING THE SPECTRA 70/45

Microprogramming the Spectra 70/45 involves the transfer of information over the DB, the transformation of information through the ALB, and finally the conditional transfer of control. To effect these operations, the EO word is set to specify the combinations of functions that are permitted to happen in one EO time period.

The process of microprogramming this system requires intimate knowledge of the system's data flow, its hardware facilities and functional capability, its data busing system, its ingating and outgating, its timing, and finally the hardware control signals which have higher priority over the microprogrammed control in certain machine states. The microprogrammer must also have a clear understanding of the ROM stack and the ROM word organization. He must master the functions of each micro-operation implied by any unique combination of binary bits in each field in the ROM word. There are complex interrelationships and interdependencies between each group of ROM fields that affect their decoding, even their function. An example here will be the function of the S and D fields. These two fields normally indicate the source and destination registers during an EO where the SPM is not specified or when field M contains 00. However, when M = 01, which implies an SPM read cycle, the contents of the S field are interpreted as the address of an SPM word from which data is to be transmitted, and the D field specifies the destination facility. A third possible interpretation of these two S and D fields is found in the case of the SPM write operation, when M = 11. Here, the DR is automatically selected as the destination register, the S specifies the SPM address (the destination of data transferred in this cycle), and the D field specifies the source.

This interdependency between the functions performed and the decoding of the fields in the ROM word is perhaps the most difficult aspect of mastering the Spectra 70/45 microprogramming. It requires a thorough analysis of the field decoding charts. We will defer this task to the latter part of this section.

Another basic requirement for developing the microprogramming skill for this system is the familiarity with the mechanics of writing a Spectra 70/45 microprogram—the different phases of preparing, assembling, simulating, and

documenting the microroutines. This requires a familiarity with the RCA 70/45 Microprogramming Design Aid System known as the RCA-MIDAS system. Like any other computer-aided design program, the MIDAS requires a fixed input format and specific control card sequence to select the desired facilities, functions, and other options available.

We will begin the analysis of these requirements with the MIDAS system.

9.4.1 MIDAS

MIDAS is a microprogramming design aid system developed by RCA programmers to assist in assembling, debugging, and simulating the microroutines and, finally, to prepare the necessary documentations and data files needed for the manufacture and maintenance of the control section of the 70/45 system. It includes an assembler to permit microprogramming in an easy meaningful symbolic notation, a simulator to test and debug the microroutines and the internal operations of the system, and a number of facilities to update, to delete, or to add new EO's in the microroutines. The MIDAS system is designed to be open-ended with the intent of adding other functions and facilities to it as the need arises.

MIDAS Assembler–Input/Output Forms. The assembler accepts the sequence of microinstructions, punched on standard 80-column punch cards in a fixed format, converts this input into an internal form used as an output to the microprogrammers, as an input to the ROM word and stack manufacturing, as an input to the simulator program, and as an input to the logic flow-chart output routine.

The microprogrammer prepares the input to the MIDAS assembler on a special form (Fig. 9.6) where each line represents an image of the EO record to be punched on one punched card. The form is divided into fields, most of which correspond to the ROM word fields. It also contains a label field in columns C1–6. This field contains a symbolic name for the EO documented on the line. The rules governing this label are:

1. The label can be 1 to 6 alphanumeric characters wide, starting with an alphabetic one and excluding the two characters $ and / (these two characters have been assigned a certain meaning in the MIDAS job control program).
2. The label must be punched left justified, i.e., starting with column 1.
3. It must not contain any blanks.
4. No relative addressing is permitted; that is, if the address of the first EO in a microroutine is assigned a label, for example TALLY, we cannot refer to the next EO as TALLY + 1.

Coding sheets for this source language input will have two sets of headings. The purpose of this is to allow the programmer to write each field in either mnemonic or binary representation.

PROGRAM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

LABEL	I SP	TEST	A	N	M	FROM	TO	F	V	CW MOD	C	EXCEPT	CONST.	ABS. ADDR.	TRACE	NOTATION	K	SEQ	INS
1	7	8	14	20	26	28	34	40	43	46	BDB BAA RAB R G CI CD	55	60	62	66	67	77		

35-00-207

Figure 9.6 Microcoding assembler form.

Notes:

1. It should be noted that when coding mnemonically, FROM and TO columns are unique in that FROM does not necessarily mean the S field and TO does not necessarily mean the D field. The programmer should only consider these fields as FROM and TO, and the assembler will make the determination whether to place the bits in the S or D fields.
2. Descriptions of various fields do not apply when a special form of EO called a TEST EO is used. A complete description of a TEST EO may be found in Appendix K.

The field on the microprogramming form with the heading of "absolute address" permits the microprogrammer to specify an absolute ROM address for any number of EO's in the microroutine. It can be used in conjunction with a label or by itself. Another field entitled "CONST" is used to specify two hexadecimal characters to be put on either the DB0 or DB1 byte bus as specified by the entry in the FV fields. The "notation" or "remark" field is for the microprogrammer and has no effect on the MIDAS assembler. The remaining fields, T, N, A, M, S, D, F, V, I, and E, are used to specify the functions to be performed in that EO.

This input is punched by trained keypunch operators on standard 80-column keypunch cards.

The assembler first checks the syntax of each EO Over twenty-seven validity checks are performed by the MIDAS system. These checks include incompatible combinations of micro-orders within an EO, invalid microcodes, violation of basic timing rules, and checks for certain illegal sequences of EO's.

The normal output of the assembler is illustrated in Fig. 9.7. It contains the assembly listing, the ROM absolute address assigned to each EO, the symbolic micro-orders, and the encoded binary bits in the control word. Another optional output is the microflow-chart printout illustrated in Fig. 9.8. This provides the microprogrammer with a visual aid to check the logical flow of the

```
  DATE 04/20/69     501 READ ONLY MEMORY  (ROM)

LABEL  TEST  A     N        MFROM MOVETO F    V  C FIELD  EXC    ABS T   COMMENT       ROMAR FVC MS D T N A E I


 TITLE 501 READ ONLY MEMORY  (ROM)                                        00000000
AFT     BRTESTTRESETSHR1     FFR1  UR     SP1RADDS SS            4600    00000100      4600  128111A102B0107000
AFLAG I       ASHRD                 FFR2  CONDB0 R             804620    00000200      4620  3D0031C00002146011
AFT1          AFT            FFR1  IR     SLLZB2 S              4775    00000300      4775  428011A0E000046010
AFUT9                AFUT91  FUT9  UR     SP1RADDS SS           4657    00000400      4657  128111810006046010
AFUT91               AFUT9   DR    FUT9   SLLZB2 S             4656    00000500      4656  42803180805746010
AFUT91               AFUT92  IR    FUT9   SRLZB2 S             4660    00000600      4660  4A8031804006146010
AFUT92               AFUT93  FUT9  G      SRLZB2 R             4661    00000700      4661  4A0011818006246010
AFUT93I              AFUT94  GA    MAR    CONDB0         SHADE404662   00000800      4662  394000F1C0063630 41
AFUT94I              AFUT95        FUT9   CONDB0 R             804663    00000900      4663  3D0031800006446011
AFUT951MR17E1AFUT98AFUT96  MMR   GUR    MGEDB1               4664    00001000      4664  300200C133465670 01
AFUT96 OR0EQ0AFUT97AFUT99  DR    IR     SP1BADDS RR          4665    00001100      4665  128000B0E017066000
AFUT97               AF801 BIR   FUT9   SETSAMES             4666    00001200      4666  788031812003547010
AFUT98               P2CALL        F1     CONDB1             444667    00001300      4667  3A4430800000045010
AFUT99               AF801 BIR     FUT9   CONDB1 S           034670    00001400      4670  3A8331812003547010
A1ORT  OR4EQ0A1ORT1AEREX           IRDS   CONDB0             804671    00001500      4671  3D0000000052572000
```

Figure 9.7 Sample MIDAS assembler output listing.

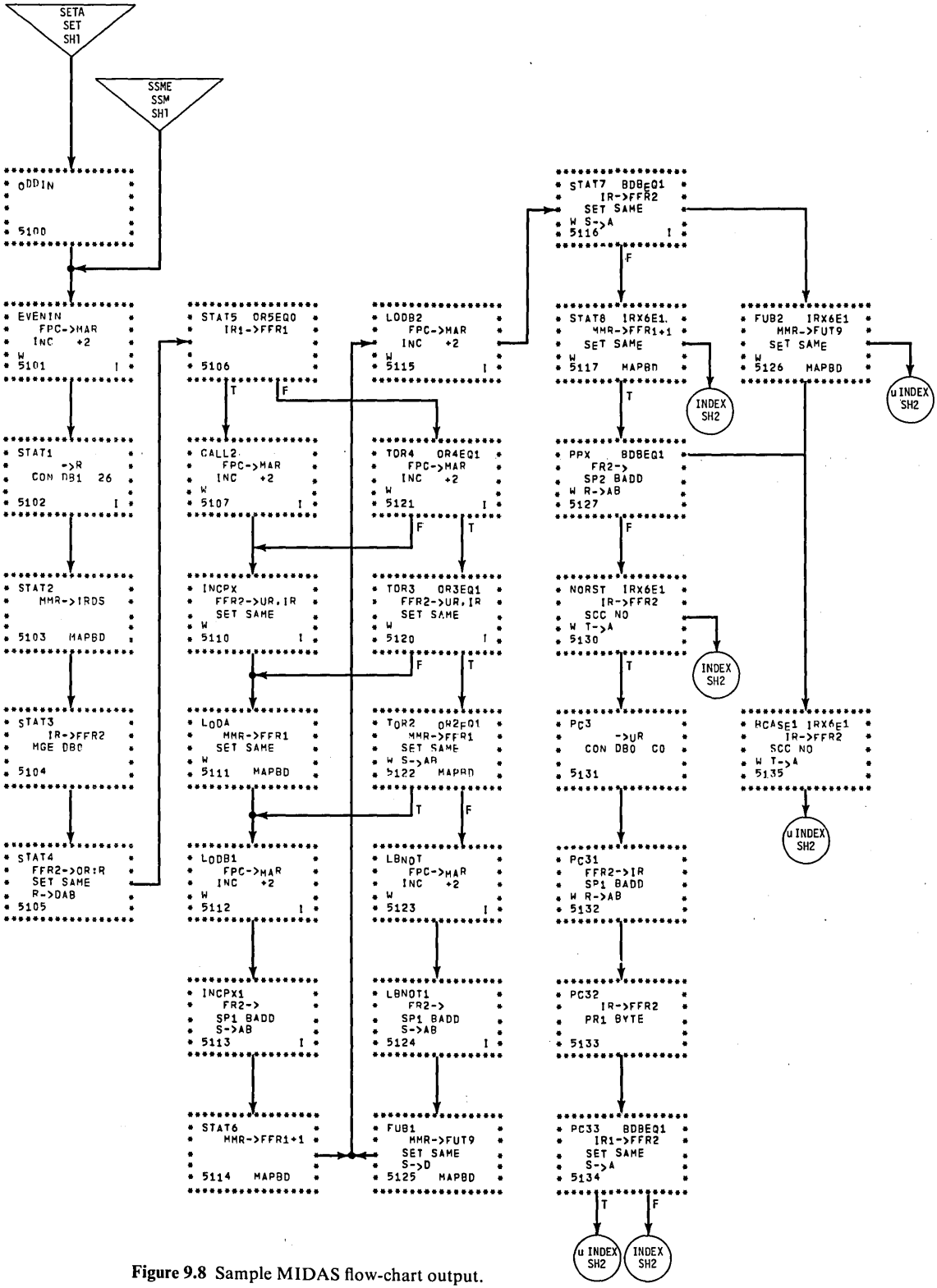


Figure 9.8 Sample MIDAS flow-chart output.

microroutines. The flow-chart program output is similar to the IBM control automation systems output. It consists of EO boxes which have a fixed format with fixed printout positions for each micro-order. For example, the register-to-register transfers are always printed on line 2.

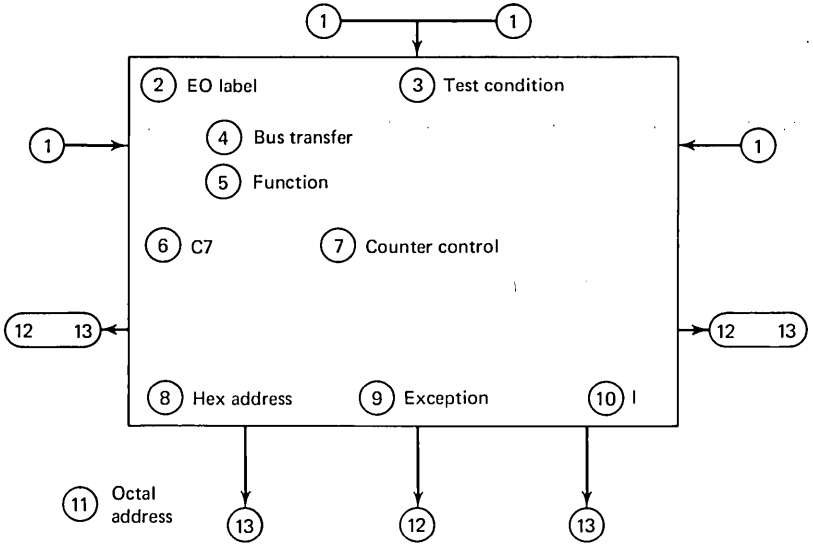


Figure 9.9 EO flow-chart format.

Figure 9.9 shows the format of this microinstruction box. The fields therein have the following meaning:

1. *Entry from Previous Microinstruction (EO).* One or more entries may be specified for an EO. All entries from previous EO's are shown on the EO flow-charts. The entry may be shown as a direct line connecting the EO with the previous EO, or the mnemonic EO label of the previous EO may be shown in a circle or triangle. A circle is used to designate an entry or exit point with the same algorithm. A triangle is used for entry or exit to another algorithm. An EO may be duplicated in different portions of the same algorithm in order to clarify the logical EO flow. An EO may also be shared and will, therefore, appear in different algorithms. In order to avoid confusion in the flow, only the paths that apply to that algorithm will be shown. (Thus, an erroneous exit or entry from or to the algorithm will not be cross-referenced.)

2. *EO Label.* The EO label identifies the EO and the EO flow-charts. It is not represented in the EO word in ROM. The first letter (or group of letters) of the label identifies the algorithm in which the EO is used.

3. *Test Condition.* The test condition mnemonic specifies the test to be performed during execution of the EO. This test condition is represented in the

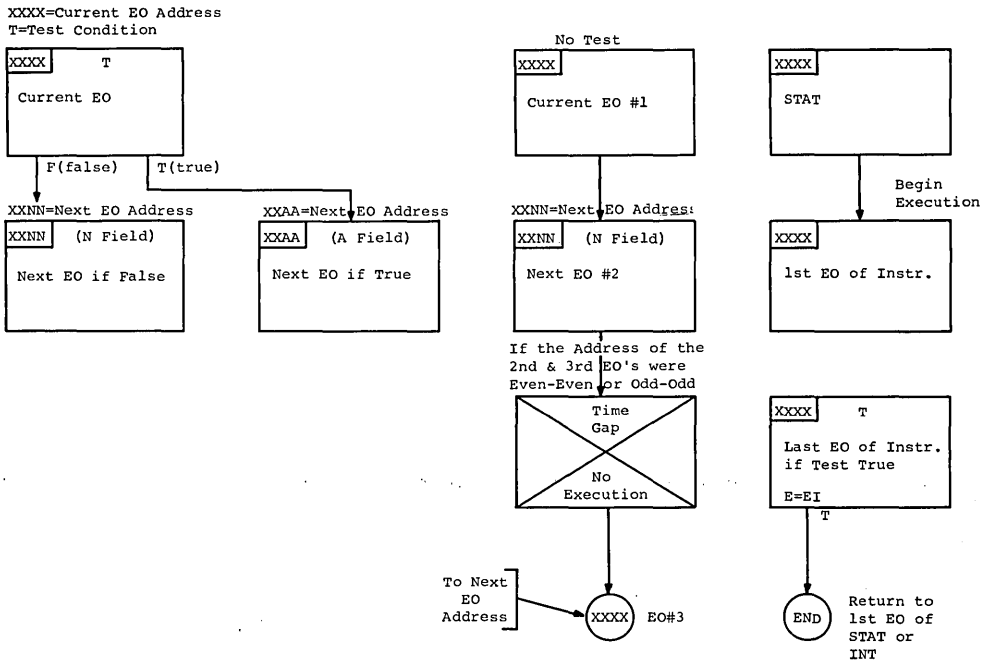


Figure 9.10 TNA fields (branch control).

T field of the EO word in the ROM (Fig. 9.3). Figure 9.10 illustrates the EO flow when a test and branch condition is specified.

4. *Bus Transfer.* The bus transfer mnemonic specifies the data movement to be executed by the EO. The form of the mnemonic is SOURCE DESTINATION. The bus transfer mnemonics are represented in the EO word in the ROM in the M, S, and D fields (Fig. 9.11).

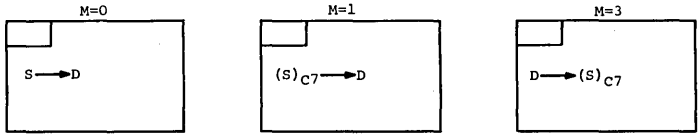


Figure 9.11 C7, MSD fields (data transfer); $F \neq 5$.

If the source is a scratch pad FM location, i.e., the first letter of mnemonic is F, the M field is $(01)_2$.

If the destination is a scratch pad location, the M field is $(11)_2$.

If neither the source nor the destination is a scratch pad location, the M field is $(00)_2$.

If a scratch pad location is specified in a bus transfer, the code for the scratch pad location is in the S field regardless of whether the scratch pad location is a source or a destination.

5. *Function.* The function mnemonic which appears on the EO flow-chart designates the functions to be performed during the EO. These mnemonics are represented in the EO word in the ROM in the F and V fields, and in some cases, in all or a portion of the C field (Fig. 9.12).

6. *C7.* The C7 mnemonic is represented by the C7 bit of the EO word in the ROM. The EO flow-chart mnemonics for C7 are either space or W. If a W is shown in the EO flow-chart, the C7 bit is 1, and DR2 and DR3 are specified. If the EO flow-chart shows a space (blank) in the C7 position, the C7 bit is 0 and DR0 and DR1 are specified.

7. *Counter Control.* The counter control mnemonic specifies the setting, resetting, or triggering of the various toggles and R and G counters. The counter control mnemonics are represented in the C field of the EO word in the ROM.

8. *Hex Address.* This mnemonic in the EO flow-chart is the ROM address of the EO. The form of the address is $X_1X_2X_3X_4$, where

$(X_1)_4$	represents bits	2^{11}-2^{10}	of address
$(X_2)_{16}$	represents bits	2^9 -2^6	of address
$(X_3)_4$	represents bits	2^5 -2^4	of address
$(X_4)_{16}$	represents bits	2^3 -2^0	of address

This form of the address is used in the 70/45 assembler and simulator and is included in the EO flow-charts to facilitate cross-referencing with the assembler listing.

9. *Exceptions.* The exception mnemonic in the EO flow-chart is represented in the ROM in the E field of the EO. If no exception appears in the EO flow-chart, the E field is 00000 if a test condition is specified; if no test condition is specified, the E field is 00001.

10. *I (Inhibit).* The EO flow-chart shows either a space (blank) or 1 in the inhibit position. If the EO flow-chart has a blank in this position, the I field of the EO in the ROM is 0, and I/O servicing is inhibited between this EO and the next EO in the flow-chart.

11. *Octal Address.* This mnemonic in the EO flow-chart is the ROM address of the EO. The form of the address is $Y_1Y_2Y_3Y_4$, where

$(Y_1)_8$	represents bits	2^{11}-2^9	of address
$(Y_2)_8$	represents bits	2^8 -2^6	of address
$(Y_3)_8$	represents bits	2^5 -2^3	of address
$(Y_4)_8$	represents bits	2^2 -2^0	of address

12. *Exit to Next EO if No Test Condition is Specified.* If no test condition is specified, there is only one exit to the next EO. This is shown on the EO flow-chart by a line connecting this EO box directly to the next EO box or to a circle or triangle containing the EO label of the next EO and flow-chart sheet number where the next EO is shown. This line is in one of the positions

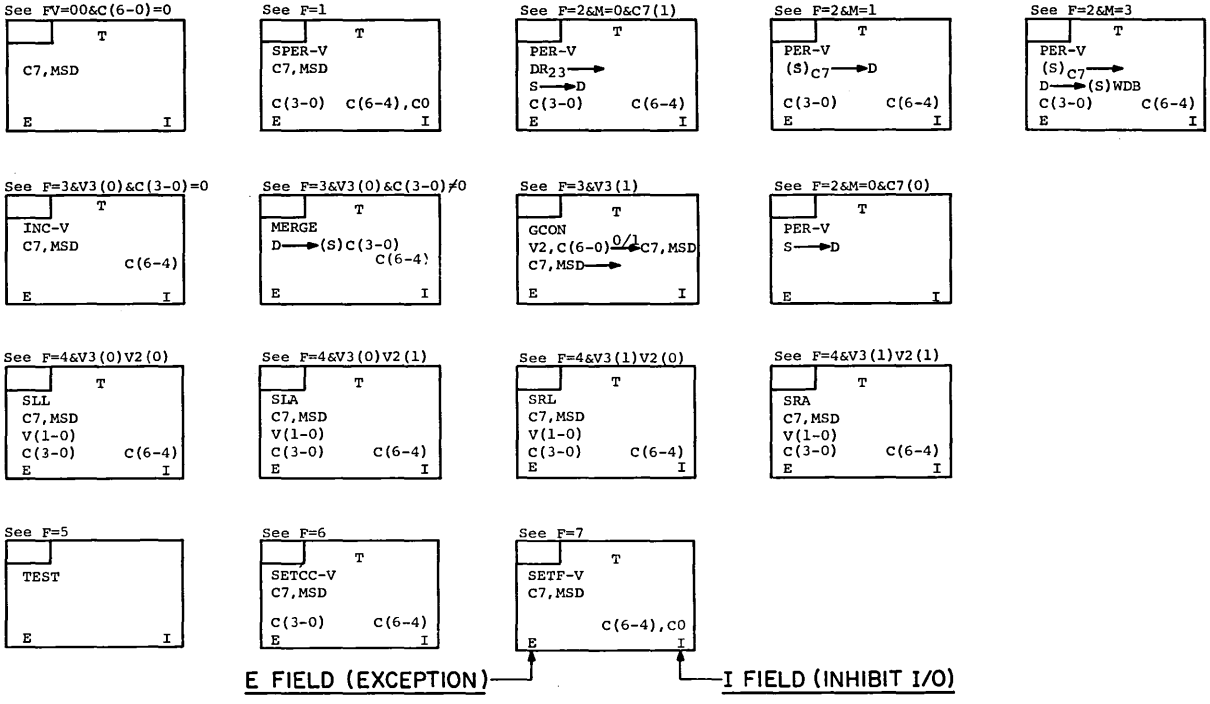


Figure 9.12 FVC field (function).

labeled 12 in Fig. 9.9. The address of the next EO is represented in the EO word in the ROM in the A and N fields as follows:

a. If no test condition is specified and no exception is specified in the EO flow-chart, the A field contains the six most-significant bits of the ROM address of the next EO to be executed. The six least-significant bits of the next EO address are in the N field. (The EO in the ROM has E field coded 00001.)
b. If no test condition is specified and an exception is specified in the EO flow-chart, both the A and the N fields contain the six least-significant bits of the ROM address of the next EO to be executed. (The A field is the same as the N field.) The six most-significant bits of the next EO address are the same as those of the current EO.

13. *Exit to Next EO if a Test Condition is Specified.* If a test condition is specified, there are two possible exits to the next EO. These are shown by 2 separate lines in any of the positions labeled 13 in Fig. 9.9. Each line connects this EO box directly to the next, or to a circle or triangle containing the EO label of the next EO and the flow-chart sheet number where the next EO is shown. A "T" alongside the line indicates the exit if the test is true; an "F" alongside the line indicates the exit if the test is false (Fig. 9.8). The address of the next EO is represented in the EO word in the ROM in the A and N fields as follows:

a. If a test condition is specified in the EO flow-chart and the exception is not ENDIN nor EES, the A field contains the six least-significant bits of the ROM address of the next EO to be executed if the test condition is true, and the N field contains the six least-significant bits of the ROM address of the next EO to be executed if the test condition is false. The six most-significant bits of the ROM address of the next EO to be executed (test condition true or false) are the same as those of the current EO address.
b. If a test condition is specified in the EO flow-charts and the exception is ENDIN and the test condition is true, the A field contains the six least-significant bits of the ROM address of the next EO to be executed. The six most-significant bits of the ROM address of the next EO, if the test is true, are 10100. If the test is false, the N field contains the six least-significant bits of the ROM address of the next EO to be executed; the six most-significant bits of the next EO address are the same as those of the current EO address.
c. If a test condition is specified and the exception is EES and the test is true, the ROM address of the next EO to be executed is obtained as follows:

2^0 of address is 2^0 of A field,
2^5-2^1 of address are a result of "logical OR" of 2^5-2^1 bits of A field with 2^4-2^0 bits of OR, respectively.

2^8-2^6 of address are 2^7-2^5 of OR, respectively,
2^{11}-2^9 of address are 100, respectively.

If the test is false, the N field contains the six least-significant bits of the ROM address of the next EO to be executed; the six most-significant bits of the next EO address are the same as those of the current EO address.

MIDAS Simulator. Another main MIDAS function is provided by the *Simulator* program, which is used as another debugging and performance evaluation aid. It contains the following optional features:

1. *Trace option* is used to trace the sequences of microinstructions as they are executed. The trace may consist of a printout of the contents of specified registers as the microroutines are being executed. Tracing can be performed for the complete microroutine or any part of it between any specified addresses.
2. *Memory Snapshots.* The simulator provides a snapshot of the MM and the SPM points specified by the microprogrammer.
3. *Termination Options.* A simulator problem can be terminated for many reasons such as time and page limits. Termination may also be specified by other exceptional conditions specified for the simulator.
4. *Checking Options.* This option provides a very useful tool in the simulator for checking the contents of the MM and the SPM against predicted results on termination of the problem.

9.5 MICROPROGRAMMING AND EO FIELD DECODING

This section has three objectives:

1. To present the ROM field decoding charts and the conditions which affect and alter the decoding of that field.
2. To relate the fields or any grouping of fields to the hardware facilities shown in Fig. 9.1 that are affected by them.
3. To provide practical illustrations and help the reader develop experience in microprogramming this system.

9.5.1 Data Movement Control Fields (M,S,D)

This group of control fields controls the transfer of data between:

1. register to register
2. register and FM (SPM)
3. register and main storage.

1. The first type is simple if one is familiar with the data-flow diagram (Fig. 9.1), the data transfer capabilities and limitations of the system, and also

the setting of the control switches which affect the byte selecting and gating pattern.

The micro-orders found in these fields are selected to identify which of the three data transfer types is requested in a particular EO. Note the following cases: If the micro-order which is written in the S or the D field starts with an F, it indicates that the FM is involved in a read or write operation. If this micro-order starting with an F is in the S field, it further implies an FM-to-register read operation, and the M field is set to 01. Here the FM is read out to the DR, or two bytes of the addressed FM word are read to the right or left half of the DR as specified by the setting of C7 (bit 7 of the C field), operated upon (if any action is specified in the F and V fields), and then regenerated (with or without shifting or decrementing) into the FM. In this FM cycle the D field specifies the destination register and the S field specifies the address of the source data in the FM. The actual transfer of data here is from the FM to the DR to the destination register specified by the D field. All this is accomplished in one machine cycle.

2. If the micro-order specified in the D field starts with an F, it implies an FM write cycle, and the M field is set to 11. In this case the DR is automatically selected as the destination register in anticipation of a possible DB transfer of information into the DR from a source register specified by the D field. That is, the contents of the register specified by the D field are transmitted to the DR (right or left half, depending on C7), and then the contents of DR are regenerated into the FM location specified by the S field.

3. If neither the S nor the D micro-orders start with an F, this indicates that no FM operation is specified for the associated EO cycle. The M field is therefore set to 00. The S field indicates the source; the D field indicates the destination register in a register-to-register operation. All other EO fields are interpreted normally.

Note that M = 10 is an invalid code, and the results are unpredictable.

We will first illustrate these three cases before introducing the main memory data transfers.

$\alpha 1$: UR(S); IR(D)
$\alpha 2$: IR(S); UR(D)
$\alpha 3$: UR(S); OR : R(D)
$\alpha 4$: FFR1(S); UR, IR(D); s(C7)
$\alpha 5$: BIR(S); FPC(D); s(C7)

The above notation is to be interpreted as follows:

1. αi is the symbolic address of the associated EO.
2. β(S); β(D); s(C)

That is, the micro-order β in the source field S specifies the source data to be transferred to the destination specified by the micro-order β appearing in the D field. The s(C) implies the setting or resetting of the bits in the C field to

effect the setting of data-flow selection switches. In general, one can represent the whole EO format to correspond to the EO input form as follows:

αi; β(I); β(T); β(N); β(A); β(M); β(S); β(D); β(F); β(V); s(C7, B, A, D, C3, C2, C1, C0); Absl; β(E); CONST

Note the Absl field here implies the absolute address specified by the microprogrammer. The CONST is a two hexadecimal digit constant that can be specified by the microprogrammer to be loaded on the DB.

The first EO with symbolic address α1 specifies that the contents of the 16-bit UR are to be transferred to the 16-bit IR register. This is an example of a register-to-register transfer. Other examples of this type of transfer are found in α2 and α3. α3-EO says that the two bytes in the source register UR are to be transmitted to two registers: the UR0 byte goes to the OR and simultaneously the UR1 byte goes to the R register.

EO-α4 represents an FM request. Here the contents of the fast floating-point register 1 (FFR1) in the FM are to be transferred to the DR, then depending on the setting of the C7 switch, either the DR0,1 or DR2,3 contents are transmitted to the destination register(s). In this specific example, DR2,3 is to be transmitted to the two specified destination registers UR, IR. Note that the character (:) in the micro-order OR:R (in α3) specifies a concatenation operation, whereas the comma (,) as in UR, IR (in α4) specifies a parallel transfer.

The fifth example (α5) specifies that two bits from register B, namely, B(1–0), and the full IR register are transmitted over bus A(1–0) and full data bus (forming a total of 18 bits) to the destination register specified by the D field. The fast program counter (FPC) micro-order specifies a program counter register in the FM. This program counter is state-dependent.

An important side issue is the use of the DR to hold the 18-bit memory address field. Figure 9.1 shows that the two-bit A data bus is always gated to DR1 bits (0, 1). This means that the remaining 16 bits transferred on DB0 and DB1 must always be set in the DR2 and DR3 bytes so as to leave DR1 available for the A bus. A general rule which covers this case is to always set bit C7 on, whenever the microprogrammer is arranging for an MM address transfer.

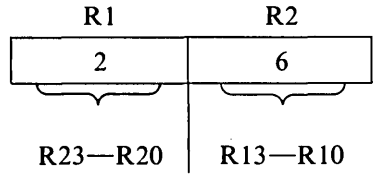
Appendix A gives the FM address codes for the CPU normal EO cycle ($\overline{I/O}$). The leftmost column specifies the encoded S field mnemonic as it appears on the assemble output. Note that the fast memory address register (FMAR) is seven bits wide, FMAR (6–0). It is divided into two address fields, and FMAR (6–4) is used to specify one of four states. For example, there are four independent sets of 16 GPR's; each set will be accessible to the microroutine depending on the state in which the system is operating. The first state's GPR's are stored in the SPM addresses 60 to 6F; the second state's in 40 to 4F; the third in 20 to 2F; and the final state in 00 to 0F. The program status register (PSR) was defined as a 2-bit register indicating the current program state of the processor. These two bits then are used to define the setting of the high order bits of the FMAR.

The floating-point registers are stored in locations 70 to 7F. Only one set of the floating-point registers is used to service all the states.

The remaining four low-order bits of the fast memory address register, FMAR (3-0), are specified by the microprogrammer and are transmitted to FMAR (3-0) from the B register [B (3-0) → FMAR (3-0)], from the left half of the R register, from the right half of the R register, or from the G register. Note that the left half of the R register is designated as R1, and the right half is designated as R2.

Examples

Assume the R register is set to "26":



Then,

1. FFR1 means 72
2. FFR1 + 1 means 73 (72 + 1)
3. FFR2 means 76
4. FFR2 + 1 means 77
5. FR1 means 62, 42, 22, or 02, depending on which state the system is in.

Another type of FM address code is the FB form. The only difference between this and the FR1 or FR2 is that the four low-order FMAR bits come from the B register rather than the R1 or R2. Note that the FB is also program-state-dependent.

The most general format is the FG. It is state-independent and permits the microprogrammer to modify any register in FM. It must be used carefully. The FMAR is totally specified by transmitting the seven leftmost bits of the G register to the FMAR. For example, assume G = "D0" hex $\underbrace{(11000000)}_{60}$. Then the seven leftmost bits when transmitted to FMAR will address location 60. Hence the EO (FG → UR) will transmit the contents of the FM word 60 to UR.

Note that, in the FG and FG + 1 formats, the G(0) = 1 case overrides the C7 setting.

For example, let G = "E2" hex. Then, which FM registers are addressable by the FG format? The correct answer is 71. The following example incorporates the setting of the G register and its use to address the FM hex.

```
α1:  → G(D);  CON(E), DB1(V);  (C7 = 1);  "E2" (CONST)
α2:  FG(S);  UR (D)
```

The first EO ($\alpha 1$) loads the DB1 with the specified constant "E2" hex and gates the contents of DB1 into the G register.

The second EO ($\alpha 2$) transfers G (7–1) to FMAR (6–0) to address location 71. It is important to recognize that the constant E2 could not have gotten to the G register if we loaded it on the DB0 bus since the G register only receives data from the DB1 bus.

Appendix A also specifies the mnemonics for other UR's in the FM. The mnemonic specifies the type of UR, and the specific program-state further defines the exact FM location desired.

Appendices B and C tabulate hardware, source and destination register codes respectively. The structure of these tables is self-explanatory. It consists of the source or destination mnemonic followed by the hexadecimal SD fields code, followed by the source or destination hardware registers. The columns labeled transfer bus show the transfer between the designated register over the data bus A and the data bus bits.

Main Memory Data Transfer. The third and final type of data transfer is the main memory to register transfer. The main memory has a 1.44 microsecond memory cycle and a 16-bit word representing two bytes. The MM cycle is equivalent to three EO cycles. The MM request is made by placing the 18-bit address field in the MMAR. This function is specified by placing the MAR micro-order in the D field. The read cycle takes three machine cycles before the particular MM word is destructively read out and the same or different word is regenerated in that memory location. As far as an MM read cycle is concerned, no useful work is accomplished during the second microinstruction; however, the second EO in this three-cycle MM sequence can be used to specify any action not involving the MMAR or main memory read (MMR). For example,

```
α1:  I;  X(S);  MAR(D)
α2:  I;  anything
α3:  MMR(S);  X(D)
```

The question is, what if no other micro-operation is to be specified in $\alpha 2$? Will we waste an EO to satisfy the MM timing requirement? No; to alleviate this loss of an ROM wire, a read-with-wait micro-order was provided by the designer. This is the memory address read-with-wait (MARW) micro-order. An example of how it is used is the following:

```
α1:  I;  X(S);  MARW(D)
α2:  MMR(S);  X(D)
```

Note that this sequence of read-with-wait takes two ROM words; however, it still requires three EO cycle times to execute. The MARW says transfer

the 18-bit MM address from the source register X into the MMAR; however, wait one EO time before performing the next EO ($\alpha 2$).

The last MM operation is that of write. This write operation is specified by requesting a memory cycle in $\alpha 1$ (by transferring an address into MMAR), and transferring the 16-bit word to be written into selected memory location in the second EO time ($\alpha 2$). The following action takes place. The address in MMAR is decoded and the corresponding X, Y drive lines are activated and destructively read-out the contents of that MM location. Early in the second EO cycle time, the data → MMR is sensed and the MM sense lines are inhibited from setting the MMR; rather the contents of DB0 and DB1 are allowed to set MMR instead. During the third EO time the regeneration cycle from MMR into main storage is completed. For example:

$\alpha 1$: I; X(S); MAR
$\alpha 2$: I; Y(S); MMR
$\alpha 3$: Anything not affecting the memory registers.

9.5.2 I Field

The I field consists of 1 bit. It is staticized in the I bit register and is used to inhibit a break in the EO sequence with a request for servicing. The following EO will be given over to the start of the I/O servicing routine if the new EO does not contain another I inhibit micro-order. This means with the setting of I in each EO, the I/O servicing routine is postponed one more EO cycle time. The maximum time for the execution of EO's without a break for I/O servicing is 1.92 microseconds. This allows for four EO's if they are the normal four-time period type. If any of the EO's are of the perform twice (PR2) or 6 basic machine time periods, then less than four EO's are permitted.

The I/O servicing required the MM and all its interface registers; hence I/O servicing cannot be permitted to occur during a memory cycle. This fact explains why the I field was specified in the $\alpha 1$ and $\alpha 2$ EO's in the last memory example. In the MARW example, the I in the $\alpha 1$—EO controls the wait cycle as well. Such servicing also requires the FM-DR data path since most channel information is stored in FM.

In the normal mode, the next normal EO address is preserved in REGAD and MREGAD—ROMAR (11-0) → REGAD (11-0), MREGAD (11-0). When MULTIPLEX (MUX) servicing breaks in, REGAD is undisturbed. During MUX servicing, the next MUX EO address is preserved in MREGAD —ROMAR (11-0) → MREGAD (11-0). I bit is used during MUX servicing to inhibit SELECTOR (SEL) servicing. When SELECTOR (SEL) servicing breaks in, REGAD and MREGAD are undisturbed.

When returning from SEL service to MUX service, MREGAD (11-0) → ROMAR (11-0), and the BPU continues MUX service. When returning from

SEL/MUX service to normal mode, REGAD (11–0) → ROMAR (11–0), and the BPU continues in the normal mode.

9.5.3 Function Control Fields: F, V, C

The F, V, and C fields are the most complex because they define a variety of functions that can be performed, and because separate functions can be performed in different parts of the processor. In general terms, F defines the basic functions to be performed, V further defines the variations within the broad general function, and C is a counter control field.

One way to present these fields is to look at the eight major classes of functions encoded in the 3-bit F field, and within each class, to examine the variations in action specified by the V and the C field settings.

The F field is encoded to signal the following action:

1. F = 0	No action
2. F = 1	Set and Perform (SPER)
3. F = 2	Perform (PER)
4. F = 3 · V3(0)	Increment/Merge
5. F = 3 · V3(1)	GCON
6. F = 4	SLL/SLA/SRL/SRA
7. F = 5	Test
8. F = 6	SETCC (set condition code)
9. F = 7	SET FUNCTION

In the remainder of this section we will take these major classes of actions and analyze the variations within each one based on the V and C field settings.

1. *F = 000—No Operation.* This EO performs the usual test, data movement, and exceptional checks but does not use or modify the ALB or the shift/increment network. The V and C fields (except C7) are ignored, and the other fields have their usual interpretation.

2. *F = 001—Set and Perform Function.* This EO sets the ALB function control flip-flops to perform functions specified by the V field, sets the toggles, and triggers the counters as specified in the C field. It then executes (performs) the function once or twice and transfers the result into IR, as specified by the V field. After each execution of the ALB function, the toggles are modified as specified in the V field. The other fields have their usual meaning.

The toggles BAA, BAB, and BDB specified by C6, C5, and C4 are set to the state of C0; the toggle WDB is set to the state of C7. The toggles can be set before the ALB function is executed, and complemented after each execution of the function. This complement operation is designed to provide an automatic control when processing two-byte register data through a one-byte ALB (by a set and perform twice microcommand).

The V(3) bit specifies the number of times to execute the ALB function and complement the toggles.

V3 = 0 Execute function once, and complement toggles once.
V3 = 1 Execute function twice, and complement toggles twice.

The V2, V1, V0 bits specify the function to be performed. (See also Set Function F = 111 for definition of these functions.)

V = X000 Function does not change
V = X001 BCD add absolute; set ICAR if IR03(1)
V = X010 Binary address add
V = X011 Binary address subtract
V = X100 Transfer UR into IR
V = X101 Transfer two's complement of UR into IR
V = X110 BCD add absolute
V = X111 BCD subtract absolute

Appendix E (with FV bit patterns between 0010000–0011111) shows the 16 micro-orders variations, eight for SP1 and eight for SP2.

3. *F = 010—Perform Function (PER).* This EO executes the function previously set into the ALB and transfers the result into the IR. The definition of the function set will be reviewed in F = 111 (Set Function). In this operation the V(3) bit specifies the number of times this function is performed. The V (2–0) defines transfer from the ALB into IR. The first execution (perform) of the ALB function is controlled by the initial setting of the three toggles BAA, BAB, BDB, and by the C7 bit of the EO. After each execution, the three toggles and the WDB are modified as specified in the C field; that is, they are not automatically complemented as in set and perform where F = 001. Each execution is controlled by the new states of the three toggle switches.

The timing of the various functions is such that an operand can be read from the SPM or any other source register into the DR or UR, and the 8-bit result of the ALB function using the operand can be set into the IR, all in a single EO time interval. If a second execution of the ALB function is specified by V(3) = 1, no additional DB transfer is made, but the operands already in the DR and UR are used. The second execution of the function therefore requires one-half the usual EO time interval (TP2 and TP3). Hence a 16-bit result of the set-and-perform-twice operation can be set into the IR in a total of 1.5 EO-time or 720 nanoseconds (TP1, TP2, TP3, TP4, TP3, TP4).

One operand of the ALB function is the byte of the UR selected by the BAB toggle:

BAB = 0 selects UR0
BAB = 1 selects UR1

The other operand is the byte of the DR selected by C7 and the BAA toggle:

C7 = 0 BAA = 0 select D0
C7 = 0 BAA = 1 select D1

C7 = 1 BAA = 0 select D2
C7 = 1 BAA = 1 select D3

The result is set into the byte of the IR selected by the BAA toggle:

BAA = 0 select IR0
BAA = 1 select IR1

The transfer path from ALB to IR is further defined by the V field of the EO and the BDB toggle. This transfer is shown in Appendix E.

Note the following rules which govern the toggle setting:

a. In a set, or set and perform, the only allowable mnemonics are S for set, R for reset, or blank. All the toggles must be set, reset, or unchanged. No arbitrary mix is allowed.
b. In the set function any toggle set in that EO is still set at the end of the EO.
c. In a set and perform, any toggle that has an entry (set or reset) will be complemented at the end of the EO during the TP4 timing point.
d. Also, during any perform EO, all toggles that have an entry will be complemented during TP4.

Appendix F gives a list of all C field columns and their valid mnemonics.

Before reviewing the remaining F action implied by F = 3 through F = 6, we will review the F = 111 set function operation.

4. *F = 111—Set Function (SETF).* This EO sets the ALB function control flip-flops to perform the function specified by the V field, and sets the toggles specified by the C field to the state of C0, but does not execute (perform) this function. The other fields have their usual meaning.

The V field designates the setting of the ALB function control flip-flops as follows:

F, V = 70 The function does not change.

F, V = 71 *BCD (decimal) add absolute*, set ICAR if IR03 = 1.
The EDR, BCD, RZ flip-flops are set. The six decimal fill method described in Chapter 8 (used with the IBM System 360 Model 50) is used.

F, V = 72 *Binary address add*
The EDR and RZ flip-flops are set, and all others are reset. This prepares the ALB and the address adder to perform a 10-bit binary addition of the selected portions of the DR and UR.

F, V = 73 *Binary address subtract*
EDR, COMP, ICAR and RZ flip-flops are set, and all others are reset. This operation is done by the two's complement arithmetic.

F, V = 74 *Transfer UR into IR*
The RZ flip-flop is set, and all others are reset. This micro-order prepares the ALB and the address adder to perform a

10-bit transfer of the selected UR register operand. The transfer is performed by binary addition of the UR operand to all zero DR operands.

F, V = 75 *Transfer two's complement of UR into IR*
Similar to F, V = 74, except that the COMP and ICAR triggers are set to permit the two's complement of the UR to be added to all zero DR operands.

F, V = 76 *BCD add absolute*
EDR, BCD, and RZ flip-flops are set, and all others are reset. The ALB is prepared to perform an 8-bit decimal addition of the selected DR and UR operands.

F, V = 77 *BCD subtract absolute*
The EDR, COMP, BCD, ICAR, and RZ flip-flops are set, and all others are reset. This prepares the ALB to perform an 8-bit decimal subtraction of the selected UR operand from the selected DR operands.

F, V = 78 *The function does not change.*

F, V = 79 *Logical OR*
The EDR, ELOR, SIMC, COMP, ICAR, and RZ flip-flops are set, and all others are reset. This prepares the ALB to perform an 8-bit logical OR of the selected UR and DR operands, and prepares the address adder to perform a 2-bit binary subtract of the GR operand from the DB operands.

F,V = 7A *Logical AND*
The EDR, ELAND, SIMC, ICAR, and RZ flip-flops are set, and all others are reset. This function prepares the ALB to perform an 8-bit logical AND of the selected UR and DR operands, and prepares the address adder to perform a 2-bit binary addition of the GR and DR operands with a forced carry from the ALB.

F,V = 7B *Exclusive OR*
The EDR, SIMC, COMP, ICAR, and RZ flip-flops are set, and all others are reset. This prepares the ALB to perform an 8-bit exclusive OR of the selected UR and DR operands, and prepares the address adder to perform a 2-bit binary subtraction of the GR operand from the DR operands.

F,V = 7C *Binary Subtract with ICAR set*
The EDR, COMP, and RZ flip-flops are set, and all others are reset. This prepares the ALB and the address adder to perform a 10-bit binary subtraction of the UR from the DR without the usual initial carry. This is equivalent to adding the one's complement of UR to the DR.

F,V = 7D, 7E, or 7F are not assigned.

The following three examples will serve to review and establish familiarity with the data movements and functional control field specifications. After the problem is stated, the microprogram prepared on the MIDAS input form is given and explained. Note that these examples contain micro-orders in fields such as the TNA and E, which are not yet covered. Their inclusion will serve as an introduction to the remaining ROM fields to be discussed immediately after the examples.

Example 1: Branch on NOT ZERO (Fig. 9.13)

PROGRAM ZERO BRANCH (Branch on 1) PROGRAMMER DATE

If (FFR1) − 1 ≠ 0; FFR2 → FPC
Else FPC = Next Instruction

LABEL	I OP	TEST	A	N	M	FROM S	TO D	F	V	C7 WDDB	C 48 BDDB	BAAB	BAAB	R	G	C5	C6	EXCEPT E 55	CONST. 60
1	7	8	14	20	26	28	34	40	43										
N1Z1B1			N1Z1B1I1				U1R1	C1O1N	D1B1I1										01
N1Z1B1I1			N1Z1B1I2			F1F1R1I1		S1P12	B1S1U1B	S1		S1S1							
N1Z1B1I2		R1Z1=1I1	O1O1O1O1O1O	N1Z1B1I3														E1N1D1I1N	
N1Z1B1I3			N1Z1B1I4			F1F1R1I2													
N1Z1B1I4		T1R1U1E1	O1O1O1O1O1O			D1R1	F1P1C1											E1N1D1I1N	

Figure 9.13 Example 1: ZERO branch microroutine.

If the contents of FFR1 minus 1 are not equal to zero, store the contents of FFR2 in FPC; otherwise FPC = the next instruction.

The five EO's needed to implement this operation are given the labels NZB, NZB1, . . . , NZB4.

NZB loads the specified constant (01 hex) on to the DB1 bus, which is in turn gated into the UR register. (Remember that the UR is first cleared; hence, at the end of this operation UR = 0001 hex.) This EO specifies NZB1 as its successor.

NZB1 The F, V fields specify a set and perform binary subtract twice. The C7, BAB, and BAA are set. The contents of FFR1 (specified by S field) are automatically transmitted into DR2,3, and in this and the following EO cycle, the binary subtraction is performed. The RZ (result = zero) trigger is set to reflect the output. This EO specifies NZB2 as its successor.

NZB2 This EO tests if RZ = 1; if the result is true, we terminate this operation and branch to instruction fetch routine (hardware address generation); otherwise, we branch to NZB3.

NZB3 This EO brings the contents of FFR2 and places it into the DR register. The next address specified is NZB4.

NZB4 Stores the contents of the DR register into the FPC and terminates

the routine by transferring control to the instruction fetch routine (staticizer routine).

Example 2: Byte move (Fig. 9.14)

PROGRAM Byte Move Microroutine PROGRAMMER DATE

LABEL	I SP	TEST	A	N	M	FROM S	TO D	F	V	C 7 WDB	C 48 BDB	C BAB	C R	C G	C S	EXCEPT E	CONST.
MOVE1	I		MOVE1			FFR1	MAR,W	INC	+1	S							
MOVE1			MOVE2			MMR	UR	SP1	TR		S	G-1					
MOVE2	I		MOVE3			FFR2	MAR	INC	+1	S							
MOVE3	I		MOVE4			IR	MMR1	NOP									
MOVE4		G=0	0,0,0,0,0,0	MOVE				NOP								END	

Figure 9.14 Byte move microroutine.

Move number of characters specified in G (assuming G is set to some count) from memory as addressed by FFR1 to memory as addressed by FFR2. This is a memory-to-memory transfer with byte control.

The first EO labeled MOVE transfers the contents of the SPM register FFR1 and sets the leftmost two bytes of FFR1 into DR2,3(C7 = 1), increments this address by +1 and regenerates it into the FFR1 location, transfers the 18-bit address to MMAR, and issues a memory request with a wait cycle. The B in the C field would transfer the zero bit of MAR to BAB trigger to control the UR switch. The I in the I field indicates an I/O inhibit cycle. In this case, since MARW microcommand is used, the I would affect two EO cycles. The function of the third cycle is documented in the EO labeled MOVE1. The MMR contains the two-byte word brought from MM. The MMR contents are transferred to the UR. The ALB function triggers are set and perform one ALB function "Transfer" (UR to IR, 8 bits). The byte transferred is determined by the zero bit of the MAR register (even byte if zero, odd byte if one). Note that, since we incremented the MM address stored in FFR1 by 1, the next time we go through this loop we will select the second UR byte. The G counter is decremented by 1 and an unconditional branch is made to EO labeled MOVE2.

The MOVE2 EO transfers the contents of the SPM word FFR2 (containing the MM move to address) to the DR2,3 and on to the MAR register. Thus, a second MM request has been initiated, and the I inhibit trigger is set. The contents of the FFR2 are incremented by 1 and regenerated into FFR2. An unconditional branch is made to the second EO cycle in the MM write cycle labeled MOVE3.

In MOVE3, the contents of IR1 are stored in MMR and regenerated in

the byte position in the memory specified by the MAR zero bit. Again, an unconditional branch is made to MOVE4.

In MOVE4, a test G = 0 is made. If the test failed, a branch to MOVE is taken; otherwise, a branch to the instruction fetch microroutine is taken.

Exercise: Write a move character-by-character microroutine taking advantage of a two-byte write capability for the two cases.

1. In the properly aligned case
2. In the nonaligned case.

Again, assume in both cases that the G counter is set to the number of bytes to be moved, and that the FFR1 and FFR2 contain the initial addresses of the two main memory blocks.

Example 3: Tally routine (Fig. 9.15)

PROGRAM Tally Microroutine PROGRAMMER DATE

LABEL	I/SP	TEST	A	N	M	FROM S	TO D	F	V	C WDB	C BDB	C BAA	C BAB	C R	C G	C C	EXCEPT E	CONST.
TIAILILIYI	I		TIAILILIYI1			FIFIRI1	MIAIRI	IINIC	+1OI		SI							
TIAILILIYI1	I		TIAILILIYI2				UIRI	CIOIN	DIBI1		SI							01
TIAILILIYI2			TIAILILIYI3			MIMIRI	DIRI	SIPI2	BISIUIB		SI	SISI						
TIAILILIYI3		RIZI≠1	TIAILILIYI6	TIAILILIYI4				NIOIP										
TIAILILIYI4	I			TIAILILIYI5		FIFIRI2		IINIC	+1OI		SI							
TIAILILIYI5				TIAILILIYI6		DIRI	FIPIC1	NIOIP			SI							
TIAILILIYI6	I			TIAILILIYI7		FIFIRIJI	MIAIRI	NIOIP			SI							
TIAILILIYI7	I			TIAILILIYI8		IIRI	MIMIRI	NIOIP										
TIAILILIYI8		TIRIUIE	0I0I0I0I0I0														ENIDIIIN	

Figure 9.15 Tally microroutine.

The following tally routine subtracts a 1 from memory as addressed by R1 and branches to address in R2 when R1 reaches zero. Let us analyze the following microroutine.

The first EO labeled TALLY transfers the contents of FFR1 in main storage into DR2,3, then on to MAR. The address is regenerated into FFR1 unchanged. An MM cycle request is initiated, and the I/O inhibit trigger is set. An unconditional branch to TALLY1 is made by the end of this EO.

TALLY1—EO zeros the UR register, loads DB1 with the specified constant "01" hex, and then gates the data bus into UR. At this point, UR is equal to "0001" hex. Since this is only the second EO cycle of the MMR cycle, the I/O inhibit trigger is set. An unconditional transfer to TALLY2 is made.

TALLY2 specifies that the contents of the MMR is to be transmitted to DR2,3. The BAB and BAA are set to control the ALB data transfer. The ALB function triggers are conditioned to set and perform twice the ALB func-

tion "Binary Subtract." The setting of BAB and BAA and their automatic complement by the end of the first perform cycle insures that the subtraction is carried out systematically (the two low-order bytes in the first perform cycle, then the two high-order bytes). Note that this subtraction is carried out by adding the two's complement of UR to the true DR2,3. At the end of this EO an unconditional branch to TALLY3 is executed.

In TALLY3, if the result of the binary subtraction is not zero—that is, RZ $\neq$ 1, test is false, branch to TALLY6; if RZ = 1 is true, branch to TALLY4.

In TALLY4, the contents of FFR2 in SPM are transferred to DR2,3, then regenerated into FFR2 without any modifications. The I/O inhibit trigger is set. A nonconditional branch to TALLY5 is then executed.

TALLY5 simply stores this address (in DR2,3) into the program counter register (FPC) in SPM, and transfers control to EO TALLY6.

In TALLY6 we find that the contents of FFR1 are again transferred to MAR via the DR2,3; the I/O inhibit trigger is set; DR2,3 is regenerated into FFR1 without any modifications, and finally an unconditional branch to TALLY7 is executed.

TALLY7 forms the second EO cycle of the MM write cycle. The contents of IR, that is, the result of the binary subtract operation, are set into the MMR to be generated (written) into the specified MM location. The I inhibit trigger is set, and an unconditional branch is made to TALLY8.

TALLY8 forces the system to transfer unconditionally to the beginning of the I fetch routine.

We have thus far discussed the function control field action implied in F = 0, 1, 2, and 7 (no action, set and perform, perform, and set function). We will continue this section with the other actions implied in the remaining groups of F classes of functions.

5. *F = 011, V3 = 0—Merge and Increment.* This EO inserts the byte from DB0 into DR0 and/or the byte from DB1 into DR1, or increments or decrements the contents of the DR and stores the result in the SPM. The particular operation is specified by the V field.

The RR and GR counters are not affected by the C field, but they may be specified as destination registers.

The WDB toggle is not changed. Toggles BDB, BAA, and BAB as specified in the C field are set to the value of the DB(10).

For the merge operation (V = 0X000), the DR need not be specified as the destination register. The result of a merge is left in the DR. The timing is such that a word is read out from SPM, or one or two bytes may be merged into that word, and the resulting word is stored in the SPM.

For an increment operation (V = 0X01 or 0X10), the word may be loaded entirely from the SPM, or partly from the MM and partly from any source register as specified by the M, S, and D fields before incrementing, or the contents of the DR may be held over from previous cycles. The contents of the DR are incremented as specified by the V field and stored in the SPM.

The V and C fields have the following interpretations:

V = 0000 merge as specified by C1, C0, increment by 0
V = 0001 merge as specified by C1, C0, increment by 1
V = 0010 merge as specified by C1, C0, increment by 2
V = 0011 merge as specified by C1, C0, increment not defined
V = 0100 merge as specified by C1, C0, increment by 0
V = 0101 merge as specified by C1, C0, increment by 1
V = 0110 merge as specified by C1, C0, increment by 2
V = 0111 merge as specified by C1, C0, increment not defined

C7 = 0 read from DB into DR0 and DR1 if RIDR and C1, C0 = 00
1 read from DB into DR2 and DR3 if RIDR and C1, C0 = 00
(in either case, the state of WDB is not changed)

C6 = 0 do not change DBD
1 set BDB accordingly to DB10

C5 = 0 do not change BAA
1 set BAA according to DB10

C4 = 0 do not change BAB
1 set BAB according to DB10

C3, C2 ignored

C1, C0 = 00 set DR from bus according to C7 if RIDR
01 clear and set DR0 from DB0
10 clear and set DR1 from DB1
11 clear and set DR0 from DB0 and DR1 from DB1

Appendix E gives a listing of the merge/increment micro-orders.

Example 1

Assuming FR1 (in the SPM) contains the value "ABCD/EF12" hex, what will be the contents of DR after the following EO has been executed:

α1: FR1(s); MERGE(F); DB01; C7 = 1

The answer is that after this EO is executed, the FR1 and DR will have EF12EF12.

Example 2

If register UR contains "1234" hex, what will be the value of FR1 after executing the following EO:

α2: UR(s); FR1(D); MERGE(F); DB0(V)

The answer is FR1 = 1200 0000. How do you justify the answer?

6. *F = 011, V3 = 1 Generate Constant.* This EO transmits the byte consisting of V2, C6, C5, C4, C3, C2, C1, C0 onto DB1 and/or DB0 as specified by V1 and V0. Counters and toggles are not changed unless they are specified as destination registers. If the source register specified in the S field and the constant are transmitted on the same bus, the logical "OR" will occur on the same bus.

C7, E, I, and the remaining EO fields have the usual meaning. The V field controls the transmission of the constant on the DB. V2 is the most significant bit of the constant.

V = 0XXX	Do not transmit constant to DB (see M/INC.)
1X000	Do not transmit constant to either DB
1X01	Transmit constant to DB0 only
1X10	Transmit constant DB1 only
1X11	Transmit constant to both DB0 and DB1

Table 9.2 shows the three basic micro-orders which fall in this area of F functions. These mnemonics are also shown in Appendix E.

Table 9.2. GENERATE CONSTANT

			F = 3·V3(1) (G CON)
	Bit Patterns		
	F Field	*V Field*	
Mnemonic	*2 1 0*	*3 2 1 0*	*Function and Remarks*
CON DB0 (YY)	0 1 1	1 X 0 1	Generate constant $(YY)_{16}$ on bus 0 $(YY)_{16} = (V_2C_6C_5C_4C_3C_2C_1C_0)_2$
CON DB1 (YY)	0 1 1	1 X 1 0	Generate constant $(YY)_{16}$ on bus 1 $(YY)_{16} = (V_2C_6C_5C_4C_3C_2C_1C_0)_2$
CON DB01 (YY)	0 1 1	1 X 1 0	Generate constant $(YY)_{16}$ on bus 0 and bus 1 $(YY)_{16} = (V_2C_6C_5C_4C_3C_2C_1C_0)_2$

A number of microprogramming examples have already used these micro-orders; hence we will continue with shift control.

7. *F = 100 Shift.* This field shifts the contents of the DR one place left or right into the SPM. The shift is specified in the V field; all other fields have the same meaning. The DR is loaded as usual from the SPM or any other source register, partially or entirely before shifting it. If no SPM cycle is specified, no shifting occurs.

The V field designates the shift to be performed as follows:

		Lost Bit Stored in	*Vacated Bit Filled from*
V = 0000	Left Logical	—	BR2
0001	Left Logical	BR0	BR2
0010	Left Logical	BR2	0

		Lost Bit Stored in	*Vacated Bit Filled from*
0011	Left Logical	BR2	BR0
0100	Left Algebraic*	—	BR2
0101	Left Algebraic	BR0	BR0
0110	Left Algebraic	BR2	IR07 ≠ URS†
0111	Left Algebraic	BR2	IR07 = URS†
1000	Right Logical	—	BR2
1001	Right Logical	BR0	BR2
1010	Right Logical	BR2	0
1011	Right Logical	BR2	BR0
1100	Right Algebraic	—	BR2
1101	Right Algebraic	BR0	BR2
1110	Right Algebraic	BR2	DR07†
1111	Right Algebraic	BR2	DR07 ≠ OV†

*A left algebraic shift will set the overflow (OV) if the two most-significant bits of the DR are not equal (DR07 ≠ DR06); this EO does not reset OV. Resetting OV can be done with a set function or a set and perform function.

†*Note:* A 1 is supplied if the boolean expression is true; a 0 is supplied if the boolean expression is false.

Appendix E gives the 16 micro-orders designed to perform the shift control.

8. *F = 101 Test and Branch.* This has a very limited usage. It was designed to handle a few special cases. The EO makes three successive tests and selects the next EO according to the first test condition that is true. The three test conditions are specified in the T, D, and S fields, and are executed in that order. If the condition specified by the T field is true, the next EO is selected from the ROM location specified by the A field. If the T condition is false but the D condition is true, the next ROM address is selected from the N field. If both the T and D conditions are false, but the S field test condition is true, the next ROM address comes from the C and V fields. In this case, the C provides the upper 8 bits whereas the V provides the lower 4 bits of the 12-bit ROM address field. If all three conditions are false, the next EO ROM address is obtained by inserting a binary 1 in the least-significant bit position of the current ROM address without changing any other bit.

These conditions and the types of tests performed are summarized in Appendix E.

9. *F = 110 Set Condition Code.* This EO may or may not set the condition code represented by the two bits SREG3 and SREG4. The two bits may be set directly from the EO, or as a complex function of the contents of the OR and several other machine indications. The variations are specified in the V field.

V3 bit specifies whether or not the condition code will be set as follows:

V3 = 0 Set condition code
V3 = 1 Do not set condition code

The V2, V1, and V0 bits define the way in which the condition code is to be set if V3 = 0.

V = 00XX Set condition code as a function of OR, IRO7, UR07, URS, SCAR, OV, RZ, STR
V = 0100 Set condition code to 00
V = 0101 Set condition code to 01
V = 0110 Set condition code to 10
V = 0111 Set condition code to 11
V = 1XXX Condition code is not changed.

Appendix E shows the classes of the machine operations and the conditions in each class under which the condition code is set.

9.5.3 TNA-Branch Control Fields

It was stated earlier that the TNA are test and branch fields, which permit the microprogrammer to specify one of 63 different tests representing hardware or flip-flop conditions that can be tested. The tests are encoded in the 6-bit T field.

The N and the A fields are also 6 bits wide. If the test condition is false or if no test condition is specified, the next EO is taken from the address designated in N. If the test condition is true, the next EO is taken from the specified address in A. In both cases, these fields provide only the low-order 6 bits of the 12-bit ROMAR. This means branching is constrained to a block of 64 ROM words.

Another branch option can be specified in the E field. The specific micro-order therein will cause the A and the N fields to provide a 12-bit jump address. This gives the flexibility to jump anywhere in ROM.

The conditional branch is implemented in the system as follows:

A: If T-true: A (5–0) → ROMAR (5–0). ROMAR (11–6) ROMAR (11–6) remains unchanged.
N: If F ≠ 5 and T-false/F = 5. T-false. D-true N (5–0) → ROMAR (5–0). ROMAR (11–6) ROMAR (11–6) remains unchanged.

Note that when M = 0/1, the T field cannot select a register or an indicator to be tested if the register or indicator is selected by the D field.

Appendix G gives the specific tests and the conditions associated with each. The format is self-explanatory.

In the previous exercises we have seen a number of test and branch conditions and conditional test options. The reader should familiarize himself with the 63 test and branch conditions encoded in the T field and specified in Appendix G.

9.5.4 Exception Check Control—E Field

The E field for the most part is concerned with various exception conditions in the Spectra 70. There are many data and programming exceptions, conditions that may be considered errors or simply indicators to the program and may result in an interrupt. They include overflow, underflow, and data not being in the correct format or not being in the correct memory location. Other conditions tested include the correct decimal code. The E field is also a catchall for all other miscellaneous functions which do not fit in any other EO field, such as the conditional jump.

We have seen how the ENDIN micro-order which is specified in the E field forces an unconditional jump to the beginning of the instruction fetch routine. The ENDIN micro-order is used at the end of the microroutine and will automatically jump back to the appropriate (hardware-generated) address in the ROM to fetch the next instruction. Another unconditional jump ENDST can be used at the end of the instruction fetch to cause a 256-way branch based on the Op code in OR. Appendix H gives a listing of all the exception memories available.

Appendix I summarizes the FM (fast memory) field settings. Appendix J provides a further explanation of, and the mnemonics for, the T, A, N, E field variations. The last, Appendix K, gives a summary of the test and branch EO's discussed earlier in the chapter.

APPENDIX 9A: SCRATCH PAD LOCATION MNEMONICS

Mnemonic	*S Field Bit Pattern 543210*	*Fast Memory Address*						
		6	*5*	*4*	*3*	*2*	*1*	*0*
FB	010000	(PS2)	(PS1)	0	(B3)	(B2)	(B1)	(B0)
FB + 1	010001	(PS2)	(PS1)	0	(B3)	(B2)	(B1)	1
FR1	010010	(PS2)	(PS1)	0	(R13)	(R12)	(R11)	(R10)
FR1 + 1	010011	(PS2)	(PS1)	0	(R13)	(R12)	(R11)	1
FR2	010100	(PS2)	(PS1)	0	(R23)	(R22)	(R21)	(R20)
FR2 + 1	010101	(PS2)	(PS1)	0	(R23)	(R22)	(R21)	1
FG	010110	(G7)	(G6)	(G5)	(G4)	(G3)	(G2)	(G1)
Note: For FG – C7 = 1 if (G0) = 1; C7 = C7 if (G0) = 0								
FG + 1	010111	(G7)	(G6)	(G5)	(G4)	(G3)	(G2)	1
Note: For FG + 1 – C7 = 1 if (G0) = 1; C7 = C7 if (G0) = 0								
FUT9	011000	(PS2)	(PS1)	1	1	0	0	1
FFR1	011010	1	1	1	(R13)	(R12)	(R11)	(R10)
FFR1 + 1	011011	1	1	1	(R13)	(R12)	(R11)	1
FFR2	011100	1	1	1	(R23)	(R22)	(R21)	(R20)
FFR2 + 1	011101	1	1	1	(R23)	(R22)	(R21)	1
FR	011110	0	0	0	0	0	(R23)	(R22)
Note: For FR – C7 = 1 if (R21) = 1; C7 = C7 if (R21) = 0								
FR + 1	011111	0	0	0	0	0	(R23)	1
Note: For FR + 1 – (C7) = 1 if (R21) = 1; (C7) = ROM C7 if (R21) = 0								
F1	001000	0	0	0	0	0	0	0
F2	001001	0	0	0	0	0	0	1
F3	001010	0	0	0	0	0	1	0
F4	001011	0	0	0	0	0	1	1
F5	001100	0	0	0	0	1	0	0
F6	001101	0	0	0	0	1	0	1
F7	001110	0	0	0	0	1	1	0
F8	001111	0	0	0	0	1	1	1
FUT10	000000	(PS2)	$\overline{(PS2)}$	1	0	0	0	0
FUT11	000001	(PS2)	$\overline{(PS2)}$	1	0	0	0	1
FUT8	000010	(PS2)	(PS1)	1	1	0	0	0
FIFR	000011	0	1	0	0	0	1	1
FIM	000100	0	K	0	$\overline{(PS2)}$	$\overline{(PS1)}$	0	0
Note: For FIM – K = 1 if (PS2) or (PS1) = 1								
FIS	000101	0	K	0	$\overline{(PS2)}$	$\overline{(PS1)}$	0	1
Note: For FIS – K = 1 if (PS2) or (PS1) = 1								
FPC	000110	0	K	0	$\overline{(PS2)}$	$\overline{(PS1)}$	1	0
Note: For FPC – K = 1 if (PS2) or (PS1) = 1								
FIW	000111	0	Y	0	1	1	1	1
Note: For FIW – Y = 1 if not PS4 Interrupt								

APPENDIX 9B: SOURCE MNEMONICS

Mnemonic	*S or D Field Bit Pattern 543210*	*Source Register*	*Transfer Bus*
B	010001	B (3-0)	Bus 0 (7-4)
BIR	010010	2 bits from B and Full IR	
		B (1-0)	Bus A (1-0)
		IR0 (7-0)	Bus 0 (7-0)
		IR1 (7-0)	Bus 1 (7-0)
B:KAP	011110	B (3-0)	Bus 0 (3-0)
		K (7-4)	Bus 1 (7-4)
		A	Bus 13
		EON	Bus 12
		BK3	Bus 11
		PRIV	Bus 10
CFF1	011001	Control flip-flops set 1	
		BCD	Bus 07
		ICAR	Bus 06
		RZ	Bus 05
		SCAR	Bus 04
		MARA (1-0)	Bus 1 (1-0)
CFF2	011010	Control flip-flops set 2	
		COMP	Bus 07
		SIMC	Bus 06
		ELOR	Bus 05
		OV	Bus 04
		MREGAD (11-8)	Bus 0 (3-0)
		MREGAD (7-0)	Bus 1 (7-0)
CFF3	011011	Control flip-flops set 3	
		BAA	Bus 07
		BAB	Bus 06
		BDB	Bus 05
		UDB	Bus 04
		US	Bus 03
		DS	Bus 02
		EDR	Bus 01
		ELAND	Bus 00
		FMMAR (6-0)	Bus 1 (6-0)
DR	001011	If C7 = 0, DR01	
		DR0 (7-0)	Bus 0 (7-0)
		DR1 (7-0)	Bus 1 (7-0)
		If C7 = 1, DR23	
		DR2 (7-0)	Bus 0 (7-0)
		DR3 (7-0)	Bus 1 (7-0)
GA	001110	G (7-0)	Bus 0 (7-0)

APPENDIX 9B: SOURCE MNEMONICS (Continued)

Mnemonic	*S or D Field Bit Pattern 543210*	*Source Register*	*Transfer Bus*
GB	001111	G (7-0)	Bus 1 (7-0)
INT1	010100	Interrupt register, lower half	
		INT (32-25)	Bus 0 (7-0)
		INT (24-17)	Bus 1 (7-0)
INT2	010101	Interrupt register, upper half	
		INT (16-09)	Bus 0 (7-0)
		INT (08-01)	Bus 1 (7-0)
IR	001010	Full IR	
		IR0 (7-0)	Bus 0 (7-0)
		IR1 (7-0)	Bus 1 (7-0)
IR1	010110	IR1 (7-0)	Bus 1 (7-0)
MACHE	011111	Machine error and IR	
		IR0 (7-0)	Bus 0 (7-0)
		MRPE	Bus 01
		DRPE	Bus 00
		IR1 (7-0)	Bus 1 (7-0)
MAR	011000	MAR0 (7-0)	Bus 0 (7-0)
		MAR1 (7-0)	Bus 1 (7-0)
MMR	001100	MMR	
		MR0 (8-0)	Bus 0 (8-0)
		MR1 (8-0)	Bus 1 (8-0)
MPMR	100000	Memory protect register	
		MPMR (3-0)	Bus 1 (7-4)
OR:R	011100	OR (7-0)	Bus 0 (7-0)
		R1 (3-0)	Bus 1 (7-4)
		R2 (3-0)	Bus 1 (3-0)
PS	001101	PS (2-1)	Bus 0 (6-5)
R	001001	Full R register	
		R1 (3-0)	Bus 1 (7-4)
		R2 (3-0)	Bus 1 (3-0)
R1	000111	R1 (3-0)	Bus 1 (3-0)
R2	001000	R2 (3-0)	Bus 1 (3-0)
S	010011	S (7-0)	Bus 0 (7-0)
S:G	011101	S and G registers	
		S (7-0)	Bus 0 (7-0)
		G (7-0)	Bus 1 (7-0)
UR	010000	Full UR	
		UR0 (7-0)	Bus 0 (7-0)
		UR1 (7-0)	Bus 1 (7-0)

APPENDIX 9C: DESTINATION MNEMONICS

Mnemonic	*D Field Bit Pattern 543210*	*Destination Register*	*Transfer Bus or Data*
BUR	010010	Full B reg. and lower 12 bits of UR	
		B (3-0)	Bus 0 (7-4)
		UR0 (3-0)	Bus 0 (3-0)
		UR1 (7-0)	Bus 1 (7-0)
		UR0 (7-4)	0000
DR	001000	If C7 = 0, DR01	
		DR0 (7-0)	Bus 0 (7-0)
		DR1 (7-0)	Bus 1 (7-0)
		If C7 = 1, DR23	
		DR2 (7-0)	Bus 0 (7-0)
		DR3 (7-0)	Bus 1 (7-0)
G	011000	G (7-0)	Bus 1 (7-0)
GUR	010011	Full UR, lower 2 bits of G	
		UR0 (7-0)	Bus 0 (7-0)
		UR1 (7-0)	Bus 1 (7-0)
		G (1-0)	Bus A (1-0)
		G (7-2)	000000
IR	001110	Full IR register	
		IR0 (7-0)	Bus 0 (7-0)
		IR1 (7-0)	Bus 1 (7-0)
IRDS	001101	IR register and DR sign reg.	
		IR0 (7-0)	Bus 0 (7-0)
		IR1 (7-0)	Bus 1 (7-0)
		If (E Field ≠ 06/07), DS	Bus 07
		If (E Field = 06/07) · [DRX (3-0) = 3/5/B/D], DS	1
IR, DR	001100	16 bits to both DR and IR	
		If C7 = 0: DR0 (7-0) and	
		IR0 (7-0)	Bus 0 (7-0)
		DR1 (7-0) and IR1 (7-0)	Bus 1 (7-0)
		If C7 = 1: DR2 (7-0) and	
		IR0 (7-0)	Bus 0 (7-0)
		DR3 (7-0) and IR1 (7-0)	Bus 1 (7-0)
KAP	011011	K (7-4)	Bus 1 (7-4)
		A	Bus 13
		EON	Bus 12
		BK3, BK3A	Bus 11
		PRIV	Bus 10

APPENDIX 9C: DESTINATION MNEMONICS (Continued)

Mnemonic	*D Field Bit Pattern 543210*	*Destination Register*	*Transfer Bus or Data*
MAR	011100	Main memory address reg.; perform next EO immediately	
		MARA (1–0)	Bus A (1–0)
		MAR0 (7–0)	Bus 0 (7–0)
		MAR1 (7–0)	Bus 1 (7–0)
MARW	011101	Main memory address reg.; wait one EO time before performing next EO	
		MARA (1–0)	Bus A (1–0)
		MAR0 (7–0)	Bus 0 (7–0)
		MAR1 (7–0)	Bus 1 (7–0)
MMR	011111	Two byte to MMR	
		MR0 (8–0)	Bus 0 (8–0)
		MR1 (8–0)	Bus 1 (8–0)
MMR1	011110	Single byte to MR0 or MR1;	
		If MAR10 (0): MR0 (8–0)	Bus 1 (8–0)
		If MAR10 (1): MR1 (8–0)	Bus 1 (8–0)
MPMAR	100000	MPMAR (6–5)	Bus A (1–0)
		MPMAR (4–0)	Bus 0 (7–3)
MPMR	100001	MPMR (3–0)	Bus 1 (7–4)
OR:R	001011	OR and R registers	
		OR (7–0)	Bus 0 (7–0)
		R1 (3–0)	Bus 1 (7–4)
		R2 (3–0)	Bus 1 (3–0)
PS	001001	Program state registers;	
		If R20 (1): PS (2–1)	Bus 0 (6–5)
		If R20 (0): PS2	(R22)
		PS1	(R21)
R	001010	Full R register	
		R1 (3–0)	Bus 1 (7–4)
		R2 (3–0)	Bus 1 (3–0)
S	011011	S (7–0)	Bus 0 (7–0)
UR	010000	Full UR register	
		UR0 (7–0)	Bus 0 (7–0)
		UR1 (7–0)	Bus 1 (7–0)
UR, IR	001111	Full UR and IR registers	
		UR0 (7–0) and IR0 (7–0)	Bus 0 (7–0)
		UR1 (7–0) and IR1 (7–0)	Bus 1 (7–0)
URUS	010001	UR register and U sign register	
		UR0 (7–0)	Bus 0 (7–0)
		UR1 (7–0)	Bus 1 (7–0)
		If E Field ≠ 06/07: US	Bus 07
		If (E Field = 06/07) · [URX (3–0) = 3/5/B/D]: US	1

APPENDIX 9D: F FIELD MNEMONICS AND REQUIRED BIT SETTING

Mnemonic	*F*	*V*	*Remarks*
NOP	000	0000	No Function
SP1	001	0XXX	Set Function and Perform Once
SP2	001	1XXX	Set Function and Perform Twice
SET	111	XXXX	Set Function
PR1	010	0XXX	Perform Once
PR2	010	1XXX	Perform Twice
INC	011	0XXX	Increment = Set C_1C_0 to 00
CON	011	1XXX	Generate Constant onto Bus (Constant is $V_2C_6C_5C_4C_3C_2C_1C_0$)
MIN	011	0XXX	
MGE	011	0XXX	Merge and Increment
SLL	100	00XX	Shift Left Logical
SLA	100	01XX	Shift Left Algebraic
SRL	100	10XX	Shift Right Logical
SRA	100	11XX	Shift Right Algebraic
TBR	101	XXXX	Test and Branch
SCC	110	XXXX	Set Condition Code
			Note: V field positions shown as X will be set according to the contents of the subfunction field. (Columns 43–46.)

APPENDIX 9E: FV FIELD COMBINATIONS

Mnemonic	*Bit Patterns* F	V	*Function and Remarks*
CON DB0 (YY)	011	1X01	Generate constant $(YY)_{16}$ on bus 0 $(YY)_{16} = (V_2C_6C_5C_4C_3C_2C_1C_0)_2$
DB1 (YY)	011	1X10	Generate constant $(YY)_{16}$ on bus 1 $(YY)_{16} = (V_2C_6C_5C_4C_3C_2C_1C_0)_2$
DB01 (YY)	011	1X11	Generate constant $(YY)_{16}$ on bus 0 and bus 1 $(YY)_{16} = (V_2C_6C_5C_4C_3C_2C_1C_0)_2$
INC + 0	011	0000	Increment scratch pad contents by 0. $C_1C_0 = 00$
+ 1	011	0001	Increment scratch pad contents by 1. $C_1C_0 = 00$
+ 2	011	0010	Increment scratch pad contents by 2. $C_1C_0 = 00$
− 1	011	0101	Decrement scratch pad contents by 1. $C_1C_0 = 00$
− 2	011	0110	Decrement scratch pad contents by 2. $C_1C_0 = 00$
MGE DB0	011	0000	Merge bus 0 to DR0. $C_0 = 1$
DB1	011	0000	Merge bus 1 to DR1. $C_1 = 1$
DB01	011	0000	Merge bus 0 to DR0 and bus 1 to DR1. $C_1C_0 = 11$
MIN H + 0	011	0000	Merge bus 0 to DR0 and increment scratch pad contents by 0. $C_0 = 1$
H + 1	011	0001	Merge bus 0 to DR0 and increment scratch pad contents by 1. $C_0 = 1$
H + 2	011	0010	Merge bus 0 to DR0 and increment scratch pad contents by 2. $C_0 = 1$
H − 1	011	0101	Merge bus 0 to DR0 and decrement scratch pad contents by 1. $C_0 = 1$
H − 2	011	0110	Merge bus 0 to DR0 and decrement scratch pad contents by 2. $C_0 = 1$
L + 0	011	0000	Merge bus 1 to DR1 and increment scratch pad contents by 0. $C_1 = 1$
L + 1	011	0001	Merge bus 1 to DR1 and increment scratch pad contents by 1. $C_1 = 1$
L + 2	011	0010	Merge bus 1 to DR1 and increment scratch pad contents by 2. $C_1 = 1$
L − 1	011	0101	Merge bus 1 to DR1 and decrement scratch pad contents by 1. $C_1 = 1$
L − 2	011	0110	Merge bus 1 to DR1 and decrement scratch pad contents by 2. $C_1 = 1$
NOP	000	0000	No Function
PR1 BYTE	010	0000	Perform previously set ALB function once on 8 bits through ALB to IR.

APPENDIX 9E: FV FIELD COMBINATIONS (Continued)

	Mnemonic	*Bit Patterns* F	V	*Function and Remarks*
	B4	010	0100	Perform previously set ALB function once on 4 bits straight through ALB to IR.
	B4NS	010	0011	Perform previously set ALB function once on 4 bits through ALB to left 4 bits of IR. Right 4 bits of IR from BCD resultant sign. Sign is inverse of that in PR1 B4S.
	B4S	010	0010	Perform previously set ALB function once on 4 bits through ALB to left 4 bits of IR. Right 4 bits of IR from BCD resultant sign.
PR1	B4X	010	0110	Perform previously set ALB function once on 4 bits through ALB crossover to IR.
	B4XZ	010	0111	Perform previously set ALB function once on 4 bits through ALB crossover to IR and 0000 to other 4 bits of IR.
	B4Z	010	0101	Perform previously set ALB function once on 4 bits through ALB straight to IR and 0000 to other 4 bits of IR.
	ZONE	010	0001	Perform previously set ALB function once on right 4 bits through ALB to IR. Left 4 bits of IR from zone generator.
PR2	BYTE	010	1000	Perform previously set ALB function twice on 8 bits through ALB to IR.
	B4	010	1100	Perform previously set ALB function twice on 4 bits straight through ALB to IR.
	B4NS	010	1011	Perform previously set ALB function twice on 4 bits through ALB to left 4 bits of IR. Right 4 bits of IR from BCD resultant sign. Sign is inverse of that in PR2B4S.
	B4S	010	1010	Perform previously set ALB function twice on 4 bits through ALB to left 4 bits of IR. Right 4 bits of IR from BCD resultant sign.
	B4X	010	1110	Perform previously set ALB function twice on 4 bits through ALB crossover to IR.
	B4XZ	010	1111	Perform previously set ALB function twice on 4 bits through ALB crossover to IR and 0000 to other 4 bits of IR.
	B4Z	010	1101	Perform previously set ALB function twice on 4 bits through ALB straight to IR and 0000 to other 4 bits of IR.

Mnemonic	*Bit Patterns* F	V	*Function and Remarks*
ZONE	010	1001	Perform previously set ALB function twice on right 4 bits through ALB to IR. Left 4 bits of IR from zone generator.
SCC NO	110	1000	Condition code does not change.
OR	110	0000	Set condition code as function of OR.
0	110	0100	Set condition code to 0.
1	110	0101	Set condition code to 1.
2	110	0110	Set condition code to 2.
3	110	0111	Set condition code to 3.
SET AND	111	1010	Set ALB function "Logical And"
BADD	111	0010	Set ALB function "Binary Add"
BSUB	111	0011	Set ALB function "Binary Subtract"
DADD	111	0110	Set ALB function "Decimal Add"
DSUB	111	0111	Set ALB function "Decimal Subtract"
EXOR	111	1011	Set ALB function "Exclusive Or"
OR	111	1001	Set ALB function "Logical Or"
SADD	111	0001	Set ALB function "Special Add"
SAME	111	1000	ALB function does not change
SET SSUB	111	1100	Set ALB function "Special Subtract"
TR	111	0100	Set ALB function "Transfer"
TRC	111	0101	Set ALB function "Transfer Complement"
SLA B2B0	100	0101	Shift Left Algebraic. Fill from B2. Shifted bit to B0.
B2L	100	0100	Shift Left Algebraic. Fill from B2. Shifted bit lost.
EQB2	100	0111	Shift Left Algebraic. Fill with 1 if US = IR07. Shifted bit to B2.
NEB2	100	0110	Shift Left Algebraic. Fill with 1 if US $\neq$ IR07. Shifted bit to B2.
SLL B0B2	100	0011	Shift Left Logical. Fill from B0. Shifted bit to B2.
B2B0	100	0001	Shift Left Logical. Fill from B2. Shifted bit to B0.
B2L	100	0000	Shift Left Logical. Fill from B2. Shifted bit lost.
ZB2	100	0010	Shift Left Logical. Fill with 0. Shifted bit to B2.
SP1 BADD	001	0010	Set and perform once ALB function "Binary Add"
BSUB	001	0011	Set and perform once ALB function "Binary Subtract"

	Mnemonic	*Bit Patterns* F	V	*Function and Remarks*
	DADD	001	0110	Set and perform once ALB function "Decimal Add"
	DSUB	001	0111	Set and perform once ALB function "Decimal Subtract"
	SADD	001	0001	Set and perform once ALB function "Special Add"
SP1	SAME	001	0000	Perform previously set ALB function once on 8 bits.
	TR	001	0100	Set and perform once ALB function "Transfer" (UR to IR, 8 bits).
	TRC	001	0101	Set and perform once ALB function "Transfer Complement" ($\overline{\text{UR}}$ to IR, 8 bits).
SP2	BADD	001	1010	Set and perform twice ALB function "Binary Add"
	BSUB	001	1011	Set and perform twice ALB function "Binary Subtract"
	DADD	001	1110	Set and perform twice ALB function "Decimal Add"
	DSUB	001	1111	Set and perform twice ALB function "Decimal Subtract"
	SADD	001	1001	Set and perform twice ALB function "Special Add"
SP2	SAME	001	1000	Perform previously set ALB function twice on 8 bits.
	TR	001	1100	Set and perform twice ALB function "Transfer" (UR to IR, 8 bits).
	TRC	001	1101	Set and perform twice ALB function "Transfer Complement" (UR to IR, 8 bits).
SRA	B2B0	100	1101	Shift Right Algebraic. Fill from B2. Shifted bit to B0.
	B2L	100	1100	Shift Right Algebraic. Fill from B2. Shifted bit lost.
	DRB2	100	1110	Shift Right Algebraic. Fill from DR07. Shifted bit to B2.
SRA	OVB2	100	1111	Shift Right Algebraic. Fill with 1 if DR07 $\neq$ OV; 0 if DR07 = OV. Shifted bit to B2.
SRL	B0B2	100	1011	Shift Right Logical. Fill from B0. Shifted bit to B2.
	B2B0	100	1001	Shift Right Logical. Fill from B2. Shifted bit to B0.
	B2L	100	1000	Shift Right Logical. Fill from B2. Shifted bit lost.
	ZB2	100	1010	Shift Right Logical. Fill with 0. Shifted bit to B2.

APPENDIX 9F: C FIELD (COLUMNS 47–54)

1. A breakdown of C field columns and their valid mnemonics is as follows:

(a) *WDB or C7**
R = set WDB or C7 to 0
S = set WDB or C7 to 1

(b) *BDB, BAA, BAB*
R = set to 0
S = set to 1
T = trigger
B = set to the value of DB10

The setting of BDB, BAA, and BAB is controlled by the same bit in the C field (C0). Therefore, if one of these three columns has a mnemonic, the other two columns must have either the same mnemonic or the blank. For example, if an R appears in the column for BAA, either an R or blank must appear in the columns for BDB and BAB.

In addition, if an S (set to 1) appears in either of these columns (BDB, BAA, BAB) with an F field setting of set and perform (F = 001), this indicates that the specified counters should be set *and* triggered. The mnemonic B may only be used if the function is increment or merge (F = 011).

(c) *R Column* (C3)
S = Set R2 to all ones [f (0R)]
T − 1 = Decrement R register by −1 [f (0R)]
T − 2 = Decrement R register by −2 [f (0R)]
T − 4 = Decrement R register by −4 [f (0R)]

The programmer should be extremely cautious in triggering the R register since the trigger of R is a function of the 0R register.

A trigger can be generated during a Set Condition Code EO (SCC), Shift EO, Set and Perform EO, or a Perform EO. The C3 bit selects R as the register to be triggered and the C1C0 bits select the value to be decremented. The 0R register determines if R is to be treated as an 8-bit counter or two 4-bit counters R1 and R2. Table 9F.1 shows the various combinations of the 0R register and their effects on the R trigger. It should be noted that the mnemonics R and B are invalid for the R column (C3).

*See notes at the end of this appendix.

Table 9F.1. R TRIGGER CONTROL

C3-0 Mnemonic	*0R7-5 = 110*	*0R7-5 = 111*	*0R7-6 = 11*
TΔ-0	R1 (– 1) R2 = F(1111)	R1 (– 1) R2 = F(1111)	R2 = F(1111) R1 – unchanged
TΔ-1	R (– 1)	R1 (– 1) R2 (– 1)	R2 (– 1) R1 – unchanged
TΔ-2	R (– 2)	R1 (– 2) R2 (– 2)	R2 (– 2) R1 – unchanged
TΔ-4	R (– 4)	R1 (– 4) R2 (– 4)	R2 (– 4) R1 – unchanged

(d) *G Column*

Modification of the G register is a function of C2, C1, C0, and the F field.

C2-0 Mnemonic	*Effect on G register*
RΔΔ	Set G register to a value of 0
T-1	Trigger G register by –1
T-2	Trigger G register by –2
T-4	Trigger G register by –4

2. Table 9F.2 shows the allowable C field combinations associated with the particular function shown. The race condition subroutine checks for proper C field combinations.

Table 9F.2. ALLOWABLE CODING FOR C FIELD

Function	*C7*	*C6 BDB*	*C5 BAA*	*C4 BAB*	*C3 R*	*C2 G*	
NOP	S/R	—	—	—	—	—	
SP*	S/R	T/S/R	T/S/R	T/S/R	—	—	
PR	S/R	T	T	T	T/S	T/R	
MIN/MGE/INC	S/R	B	B	B	—	—	
CON	S/R	—	—	—	—	—	
SHIFT	S/R	T	T	T	T/S	T/R	(Same as PR)
TBR	—	—	—	—	—	—	
SCC	S/R	T	T	T	T/S	T/R	(Same as PR)
SET	S/R	S/R	S/R	S/R	—	R	

*Either S or R in a Set and Perform function causes a trigger after the function is executed.

3. The assembler should set the following bits:

WDB or C7	*BDB*	*BAA*	*BAB*	*R*	*G*	*C1*	*C0*	*Assembler Action*
R S								Set C7 = 0 Set C7 = 1
	B R S T							Set C6 = 1 Set C6 = 1, C0 = 0 Set C6 = 1, C0 = 1 Set C6 = 1
		B R S T						Set C5 = 1 Set C5 = 1, C0 = 0 Set C5 = 1, C0 = 1 Set C5 = 1
			B R S T					Set C4 = 1 Set C4 = 1, C0 = 0 Set C4 = 1, C0 = 1 Set C4 = 1
				S T T T		— — —	1 2 4	Set C3 = 1, C1C0 = 00 Set C3 = 1, C1C0 = 01 Set C3 = 1, C1C0 = 10 Set C3 = 1, C1C0 = 11
					R T T T	— — —	1 2 4	Set C2 = 1, C1C0 = 00 Set C2 = 1, C1C0 = 01 Set C2 = 1, C1C0 = 10 Set C2 = 1, C1C0 = 11

4. Since the particular setting of all registers specified in the C field except WDB may be controlled by C1C0, the programmer must use extreme care when specifying the setting of each register. He may only call for a setting in the C field that will cause the required setting of C1C0 to be compatible with all other registers specified in the C field. For example, the mnemonic S when used in BAA column causes C0 to be set to 1. The programmer may not use an R in the G column of the same EO word since this causes C1C0 to be set to 00.

Notes:

1. Programmers must use extreme care in programming for WDB or C7 (Column 47). The following rules apply.

(a) WDB will control the affected half of the DR register *only* when the function (Columns 40–42) is PR1 or PR2 (Perform), and then only when a transfer from the Bus to DR is specified. It should also be noted that this control will be on a *previous* setting of WDB, and *not* the setting

of WDB as specified within the EO word to be executed. In all other cases, the desired half of DR is under control of C7.

(b) WDB may be set only when the functions (Columns 40–42) SP1, SP2 (Set function and perform), PR1, PR2 (Perform), or SET (Set function) are specified.

Note that any transfers to or from DR register specified in an EO word that includes one of the above functions, except as outlined in (a) above, will be under control of C7 and not WDB, but WDB will be set to the value specified in Column 47 of the coding sheet. In all other cases, WDB will not be affected when a value is specified in Column 47.

2. The assembler sets C7 to 0 or 1, depending on whether the mnemonic R or S is used by the programmer. If Column 47 is blank, C7 is reset to zero.

APPENDIX 9G: TEST CONDITION MNEMONICS

Mnemonic	*T Field Bit Pattern*	*Test Condition True If*
B0EQ1	010110	B0 (1)
B0EQ0	010110	Complement test* of B0 (1)
BIR = +1	011001	B3 (1) · B1 (0) · B0 (0) · (IR = 00000000/00000001)
BAAEQ1	100010	BAA (1)
BAAEQ0	100010	Complement test of BAA (1)
BABEQ1	100011	BAB (1)
BABEQ0	100011	Complement test of BAB (1)
BDBEQ1	100100	BDB (1)
BDBEQ0	100100	Complement test of BDB (1)
BRTEST	101011	B2 (1)/(R2 = 0000)
DR1 = FS	010000	DR1 = 00100010
DR0 = FS	010001	DR0 = 00100010
DSEQ0	010010	DS (0)
DSEQ1	010010	Complement test of DS (0)
GORTST	001100	(G = XX1XXXXX)/(OR = XX1XXXXX)/(OR = 01000001)
GTEST	010101	[(OR ≠ 111XXXXX) · (G4 = 0)]/[(OR = 111XXXXX) · (GR6 = 0)]
G = 0	011101	(G = 00000000)/(G = 10000000)
G = ONES	011110	G = 11111111
GGR4	011111	GR ≠ $(00)_{16}$/$(01)_{16}$/$(02)_{16}$/$(03)_{16}$/$(04)_{16}$/$(80)_{16}$/$(81)_{16}$/$(82)_{16}$/$(83)_{16}$/$(84)_{16}$
IR7EQ1	010111	IR07 (1)
IR7EQ0	010111	Complement test IR07 (1)
IREQ0	011010	IR = $(00)_{16}$
IRNE0	011010	Complement test of IR = $(00)_{16}$
IRXR = 0	011011	{BAA (0) · [IR1 (3–0) = 0]}/{BAA (1) · [IR0 (3–0) = 0]}
†IRX6E1	111100	[BAA (0) · IR06 (1)]/[BAA (1) · IR16 (1)]
MR7EQ1	001000	MR07 = (1)
MR7EQ0	001000	Complement test of MR07 = (1)
†MRX6E0	110000	[BAB (0) · MR06 (0)]/[BAB (1) · MR16 (0)]
†MR06E0	110001	MR06 (0)
†MR16E0	110010	MR16 (0)
†MR0616	111011	MR06 (1) · MR16 (1)
†MR17E1	110100	MR17 (1)
†MRX6X7	110101	[BAB (0) · MR06 (1) · MR07 (1)]/[BAB (1) · MR16 (1) · MR17 (1)]
†MRSCWM	111101	MR06 (1) · MR07 (1) · MR16 (1) · MR17 (1)
†MR14E1	110111	MR14 (1)
†MRX5E1	111000	[MAR10 (0) · MR05 (1)]/[MAR10 (1) · MR15 (1)]
†MRX4X5	111001	[MAR10 (0) · MR05 (0) · MR04 (0)]/[MAR10 (1) · MR15 (0) · MR14 (0)]
†MRENC	111010	MR05 (0) · MR04 (0) · MR15 (0) · MR14 (0)
†MRXESP	110011	[MAR10 (0) · MMR0 = XX000001]/[MAR10 (1) · MMR1 = XX000001]
†MRXESI	110110	[MAR10 (0) · MMR0 = XX000001]/[MAR10 (0) · MMR0 = XX111100]/[MAR10 (1) · MMR1 = XX000001]/[MAR10 (1) · MMR1 = XX111100]
†MRXESM	111110	[MAR10 (0) · MMR0 = XX000001]/[MAR10 (0) · MMR0 = XX001100]/[MAR10 (1) · MMR1 = XX000001]/[MAR10 (1) · MMR1 = XX001100]

APPENDIX 9G: TEST CONDITION MNEMONICS (Continued)

Mnemonic	*T Field Bit Pattern*	*Test Condition True If*
OR0EQ0	000001	OR0 (0)
OR0EQ1	000001	Complement test of OR0 (0)
OR1EQ0	000010	OR1 (0)
OR1EQ1	000010	Complement test of OR1 (0)
OR2EQ0	000011	OR2 (0)
OR2EQ1	000011	Complement test of OR2 (0)
OR3EQ0	000100	OR3 (0)
OR3EQ1	000100	Complement test of OR3 (0)
OR4EQ0	000101	OR4 (0)
OR4EQ1	000101	Complement test of OR4 (0)
OR5EQ0	000110	OR5 (0)
OR5EQ1	000110	Complement test of OR5 (0)
OR6EQ0	000111	OR6 (0)
OR6EQ1	000111	Complement test of OR6 (0)
OVEQ1	101001	OV (1)
OVEQ0	101001	Complement test of OV (1)
RZTEST	001001	(RR = 00000000)/[(OR = 111XXXXX)·(R1 = 0000/R2 = 0000)]
RNE0/1	001010	(RR ≠ 00000000)·(RR ≠ 11111111)
REQ1S	001011	RR = XXXXXXXX
R1EQ1S	001101	R1 = 1111
R2EQ0	001110	R2 = 0000
R2EQ1S	001111	R2 = 1111
R21EQ0	010011	R21 (0)
R21EQ1	010011	Complement test of R21 (0)
R1EQ0	010100	R1 = 0000
R1NE0	010100	Complement test of R1 = 0000
R1EQ1	011100	R1 = 0001
R1NE1	011100	Complement test of R1 = 0001
RZ = 1	100000	RZ (1)
RZ = 0	100000	Complement test of RZ (1)
RZSCAR	100001	RZ (0)/(RR = 11111111)/SCAR (0)
SCAR = 0	100101	SCAR (0)
SCAR = 1	100101	Complement test of SCAR (0)
TRUE	111111	No Test. Result always true
USEQ0	100110	US (0)
USEQ1	100110	Complement test of US (0)
USEQ07	101010	[US (0)·IR07 (0)]/[US (1)·IR07 (1)]
URTEST	011000	{BAB (0)·[UR (03–00) > 9]}/{BAB (1)·[UR (13–10) > 9]}

*When any Complement Test is specified, the assembler places the bit pattern of the A field into the N field of the EO word and the bit pattern of the N field is placed in the A field. For all other cases, the A and N fields are placed in their respective positions in the decoded EO word.

†These tests only valid when OR ≠ 100111XX.

APPENDIX 9H: EXCEPTION MNEMONICS

Mnemonic	*E Field Bit Pattern 43210*	*Action and Remarks*
	00000	No exception. If test is true, next EO address is: 2^5–2^0 = 2^5–2^0 of A field 2^{11}–2^6 remain unchanged. If test is false, next EO address is: 2^5–2^0 = 2^5–2^0 of N field 2^{11}–2^6 remain unchanged.
	00001	Unconditional jump. Address of next EO is: 2^5–2^0 = 2^5–2^0 of N field 2^{11}–2^6 = 2^5–2^0 of A field (See Appendix 9K).
EES	10011	Emulator End Staticizing. If test is false, address of next EO is: 2^5–2^0 = 2^5–2^0 of N field 2^{11}–2^6 remain unchanged. If test is true, address of next EO is: 2^0 = 2^0 of A field 2^5–2^1 = result of logical or of 2^5–2^1 of A field with 2^4–2^0 of OR, respectively. 2^8–2^6 = 2^7–2^5 of OR, respectively. 2^{11}–2^9 = 100, respectively.
EXCXX	YYYYY	XX are two hexadecimal digits which, when converted to binary, equal 000YYYYY. For action resulting from these exceptions, refer to 70/45 Processor EO Flow Chart Manual, EDP form 70-45-101.
ENDIN	00010	End of instruction. Address of next EO is: If test is true: 2^5–2^0 = 2^5–2^0 of A field 2^{11}–2^6 = 101001, respectively If test is false: 2^5–2^0 = 2^5–2^0 of N field 2^{11}–2^6 remain unchanged.
ENDST	00011	End of staticizing. If test is false, address of next EO is: 2^5–2^0 = 2^5–2^0 of N field 2^{11}–2^6 remain unchanged. If test is true, address of next EO is: 2^0 = 2^0 of A field 2^5–2^1 = result of logical or of 2^5–2^1 of A field with 2^4–2^0 of OR, respectively.

APPENDIX 9H: EXCEPTION MNEMONICS (Continued)

Mnemonic	*E Field Bit Pattern 43210*	*Action and Remarks*
		2^8–2^6 = 2^7–2^5 of OR, respectively. 2^{11}–2^9 = 000 respectively.
		If OR = $(0A)_{16}$, set INT21
MAPBD	10001	Mapped Memory boundary. If MR1 = $(8F)_{16}$, set INT24.
NOOP	01110	No-op interrupt. Set INT23.
SHADE	00100	Address shaded memory.
TBK3	10010	Trigger BK3 flip-flop.

APPENDIX 9I: M FIELD SETTINGS

The M field setting controls the operation on Fast Memory and the assembler should accomplish the following:

*1. If *neither* FROM or TO is Fast Memory, set M = 00.
2. If FROM *is* Fast Memory and TO is *not* Fast Memory, set M = 01.
3. If FROM is *not* Fast Memory and TO *is* Fast Memory, set M = 11.
*4. If FROM and TO are both the same Fast Memory location, set M = 01. Definition of M field is as follows:

M = 00—Not a Fast Memory operation
M = 01—Fast Memory read
M = 11—Fast Memory write
M = 10—Not used

APPENDIX 9J: T, A, N, AND E FIELD VARIATIONS

Two tables have been included in this appendix to show the assembler action on various combinations of the test field (col. 8–13), the A field (col. 14–19), the N field (col. 20–25) and the Exception field (col. 55–59).

Table 9J.1 contains the possible variations of T, A, and N if the E field was *not coded.* The column headed by "EXCEPT" shows how the assembler automatically codes the EXCEPT field in the decoded EO. The last column explains how the A and N fields are decoded under these conditions.

*The only time the FROM and TO fields may both be Fast Memory is when they are both the same Fast Memory location. M must contain the proper binary code if EO is coded in binary.

Table 9J.1. ASSEMBLER ACTIONS FOR T, A, N, AND E FIELD VARIATIONS WITH UNCODED EXCEPT FIELD

Test	*A*	*N*	*EXCEPT**	*A and N Fields of EO Word Contain*
Coded	Coded	Coded	00	A = 2^5–2^0 of coded A field N = 2^5–2^0 of coded N field
Coded	Coded	Blank	00	A = 2^5–2^0 of coded A field N = EO address +1 (2^5–2^0)
Coded	Blank	Coded	00	A = EO address +1 (2^5–2^0) N = 2^5–2^0 of coded N field
Coded	Blank	Blank	00	A = EO address +1 (2^5–2^0) N = EO address +1 (2^5–2^0)
Blank	Coded	Coded	01	A = 2^{11}–2^6 of coded A field† N = 2^5–2^0 of coded A field
Blank	Coded	Blank	01	A = 2^{11}–2^6 of coded A field N = 2^5–2^0 of coded A field
Blank	Blank	Coded	01	A = 2^{11}–2^6 of coded N-field N = 2^5–2^0 of coded N field
Blank	Blank	Blank	01	A = 2^{11}–2^6 of EO address +1 N = 2^5–2^0 of EO address +1

*EXCEPT column indicates the action taken by the assembler in decoding the EXCEPT field if the programmer has left it blank.

†N field ignored in this case.

Table 9J.2. ASSEMBLER ACTIONS FOR T, A, N, AND E FIELD VARIATIONS WITH CODED EXCEPT FIELD

Test	*A*	*N*	*EXCEPT*	*A and N Fields of EO Word Contain*
Coded	Coded	Coded	01	A = 2^5–2^0 of coded A field* N = 2^5–2^0 of coded N field
Coded	Coded	Blank	01	A = 2^5–2^0 of coded A field* N = 2^5–2^0 of EO address +1
Coded	Blank	Coded	01	A = 2^5–2^0 of EO address +1* N = 2^5–2^0 of coded N field
Coded	Blank	Blank	01	A = 2^5–2^0 of EO address +1* N = 2^5–2^0 of EO address +1
Blank	Coded	Coded	01	A = 2^{11}–2^6 of coded A field† N = 2^5–2^0 of coded A field
Blank	Coded	Blank	01	A = 2^{11}–2^6 of coded A field N = 2^5–2^0 of coded A field
Blank	Blank	Coded	01	A = 2^{11}–2^6 of coded N field N = 2^5–2^0 of coded N field
Blank	Blank	Blank	01	A = 2^{11}–2^6 of EO address +1 N = 2^5–2^0 of EO address +1
Coded	Coded	Coded	≠01	A = 2^5–2^0 of coded A field N = 2^5–2^0 of coded N field
Coded	Coded	Blank	≠01	A = 2^5–2^0 of coded A field N = 2^5–2^0 of EO address +1
Coded	Blank	Coded	≠01	A = 2^5–2^0 of EO address +1 N = 2^5–2^0 of coded N field

Table 9J.2. ASSEMBLER ACTIONS FOR T, A, N, AND E FIELD VARIATIONS WITH CODED EXCEPT FIELD (Continued)

Test	*A*	*N*	*EXCEPT*	*A and N Fields of EO Word Contain*
Coded	Blank	Blank	≠01	A = 2^5–2^0 of EO address +1 N = 2^5–2^0 of EO address +1
Blank	Coded	Coded	≠01	A = 2^5–2^0 of coded A field N = 2^5–2^0 of EO address +1
Blank	Coded	Blank	≠01	A = 2^5–2^0 of coded A field N = 2^5–2^0 of EO address +1
Blank	Blank	Coded	≠01	A = 2^5–2^0 of coded N field N = 2^5–2^0 of EO address +1
Blank	Blank	Blank	≠01	A = 2^5–2^0 of EO address +1 N = 2^5–2^0 of EO address +1

*These cases are probably coding errors since the A and N fields of the EO word are combined to form one 12-bit address when the exception code is 01 and the combination is neither the address of a label nor of N label.

†N field ignored in this case.

Table 9J.2 contains all possible variations of T, A, N, and E if the programmer *codes* the EXCEPT field himself. The table shows the difference in the way the A and N fields are decoded if the EXCEPT field is 01 or if it is not 01 (≠01), since 01 is the only code which changes the addressing scheme.

Notes:

1. The assembler decodes the A and N fields as shown except when a complement test is used. Then the A and N fields are treated as if they were reversed, i.e., A field coding is used as if it were in the N field and vice versa.
2. All 6-bit addresses are checked to see if they are in block.

APPENDIX 9K: TEST AND BRANCH EO

The test and branch EO is a special EO, the fields in the TBR EO having meanings different from the normal EO.

There may be up to three tests and branches within the same EO and if none of the tests is true, Read Only Memory will automatically "drop through" to the next sequential EO. This EO may be coded either symbolically or "binarily" and the special requirements for both cases are described below.

I. *Symbolically Coded Test and Branch EO*

When the test and branch EO is symbolically coded, the mnemonic TBR is used in the F field. Up to three lines of coding may be used for each test EO and TBR *must appear* in the F field (col. 40–42) of all three lines.

In order for the assembler to distinguish between separate EO's, each EO *must* contain a label in the LABEL field on the first line of coding and the LABEL field *must* be blank on the second and third lines of coding if they are used.

Only five fields besides the LABEL field on the coding sheet may be used. These are the TEST field, the A field, the F field, the EXCEPT field, and the ABS ADDR field. These fields are used to describe the particular test flow desired and are set up as follows:

A. *First Card*

1. The *LABEL* field (col. 1–6) *must* be coded and with a valid label. All characters are valid for a label except the use of a numeric for the first character of a label name. The address of this EO is assigned to this label and then placed in the tag table.
2. The *TEST* field (col. 8–13) must be coded with a valid test mnemonic. The bit pattern for this test is placed in T field of the EO word or with the proper binary bit pattern (see Appendix 9B). Complement tests are not legal for a test EO.
3. The *A* field (col. 14–19) must be coded with a valid label of a current EO. The EO referred to must be located within the same ROM block that the test and branch EO is placed. This field will be the jump address of the first test which is contained in the TEST field if the test is true. This is placed in the A field of the assembled word.
4. The *F* field (col. 40–42) *must* contain the mnemonic TBR. The bit pattern is placed in the F field by the assembler.
5. The *EXCEPT* field (col. 55–59) may contain any valid exception mnemonic or may be coded in binary or hexadecimal according to the rules for this field as explained in the specification, or may be left blank. This field is placed in the E field of the EO word.
6. The *ABS ADDR* field (col. 62–65) may contain a valid *octal* number to specify the desired address of the test and branch EO or may be left blank. If blank, the next sequential address is assumed and assigned to this EO.

All other fields *must* be left blank except the I field (col. 7) which may be coded normally to inhibit I/O servicing.

B. *Second Card**

1. The *LABEL* field *must* be blank if this is to be assembled as part of the same EO.
2. The *TEST* field must contain a valid test pattern for the second test. The assembler will place this bit pattern in the D field. Valid tests for the second line of coding are:

*This card not necessary if only one test is desired. If no second test is desired but a third test is used, this card must be present and only the F field is coded with TBR.

Bit Pattern	*Meaning*
000001	SSDEC (R1/R2 = 0)/R = 0
000010	DR1 = significant start symbol
000011	DR0 = significant start symbol

3. The *A* field must contain a valid label of a current EO. The EO referred to must be located within the same ROM block that the test and branch EO is placed. This field will be the jump address of the second test which is contained in the TEST field of the second card if the test is true. The assembler will place this address in the N field of the EO word.
4. The *F* field *must* contain the mnemonic TBR.

All other fields should be left blank.

C. *Third Card**

1. The *LABEL* field *must* be blank.
2. The *TEST* field must contain a valid test pattern for the third test. The assembler will place this bit pattern in the S field. Valid tests for the third line of coding are as follows:

Bit Pattern	*Meaning*
000100	R1 NE condition code
000101	R GR 0
000110	DR1 NE digit select symbol
000111	DR0 NE digit select symbol

3. The *A* field must contain a valid mnemonic label. This field will be the jump address of the third test which is contained in the TEST field of the third card. Because this is a 12-bit address, it is not necessary that this label be in the same block. The assembler will place this address in the VC fields of the EO word in the format $C_7C_6C_5C_4C_3C_2C_1C_0V_3V_2V_1V_0$ with the low-order bits in the V field. It should be noted, however, that the V field is placed directly in front of the C field on the coding form and in the assembled EO word.

II. *Binary Coded Test and Branch EO*

A test and branch EO may also be coded on a single card by using the binary format; requirements for the binary coding are given below in a field-by-field description.

*This card not necessary if only a one or two test EO is desired. The assembler will automatically fill in an insert if this card is missing.

Note: A mnemonic test EO may consist of one, two, or three lines of coding at any time. Error messages are printed after the assembled EO word to aid the microprogrammer in debugging his coding.

1. The LABEL field (col. 1–6) may contain any valid label name.
2. The I field (col. 7) is coded normally to inhibit I/O servicing if necessary. This may be coded either in binary or mnemonically and will be placed in the I field of the EO word.
3. The TEST field (col. 8–13) will contain a valid binary test pattern as found in Appendix 9A or may be coded mnemonically. This is the first test of the EO and will be placed in the T field of the assembled word.
4. The A field (col. 14–19) will contain a 6-bit binary pattern describing the jump address of the first test if it is true. (The most significant 6 bits of the address are omitted because it is required that this jump be within block and the most significant 6 bits describe the block address.) These 6 bits are placed in the A field of the EO word. If desired, this field may also be coded with a valid label of a current EO within block.
5. The N field (col. 20–25) is similar to the A field except this is the jump address of the second test (col. 34–39). This address is decoded into the N field of the EO word and the same coding rules apply for this as for the A field.
6. The M field (col. 26–27) *must* contain the binary 00. Without this, the FROM and TO fields will not be decoded.
7. The FROM field (col. 28–33) must contain a valid bit pattern for the third test as shown above. This bit pattern is placed in the S field of the EO word.
8. The TO field (col. 34–39) must contain a valid bit pattern for the second test as shown above. This bit pattern is placed in the D field of the EO word.
9. The F field (col. 40–42) *must* contain the bit pattern "101" for a binary test and branch EO. Using the mnemonic coding TBR will invalidate the assembly and cause an error flag to be set because it will be assumed to be a mnemonically coded EO.
10. The VC fields (col. 43–54) will contain the 12-bit binary jump address of the third test coded in the FROM field. This address is coded in the format $C_7C_6C_5C_4C_3C_2C_1C_0V_3V_2V_1V_0$ with the least significant four bits coded in the V field (col. 43–46). This address can only be coded by a valid *binary* 12-bit address which represents the four character octal address of the EO that is being referred to. No mnemonic code may be used in this field.
11. The EXCEPT field (col. 55–59) is coded with any valid mnemonic exception as described in Appendix 9H or in straight binary.
12. The CONST field (col. 60–61) is not used in the test and branch EO and is ignored.
13. The ABS ADDR field (col. 62–65) must be blank or contain a valid four character *octal* number.

The remaining fields are not used.

Field	*ROMAR*	*F*	*VC*	*M*	*S*	*D*	*T*	*N*	*A*	*E*	*I*
Description	*EO adder*	*Function*	*3rd test jump adder*	*Not used*	*3rd test*	*2nd test*	*1st test*	*2nd jump adder*	*1st jump adder*	*EXCEPT field*	*I bit*
Prntr. Char.	XXXX†	5*	XXX	XX	XX	XX	XX	XX†	XX†	XX	X
No. of Bits	12	3	12	2	6	6	6	6	6	5	1

Note: X indicates a printer graphic; * indicates function field always containing a "5" for a test EO; † indicates the field in octal.

Assembly Listing Format for Test and Branch EO

The format of the assembled test and branch EO is as follows:

ROMAR	*F*	*V*	*C*	*M*	*S*	*D*	*T*	*N*	*A*	*E*	*I*
XXXX	X	X	XX	X	XX	XX	XX	XX	XX	XX	X

where X indicates the number of printer graphics associated with the appropriate fields. All fields are hexadecimal except ROMAR, N, and A, which are octal. The test and branch EO decoding is shown on page 486.

PRACTICE PROBLEMS

1. Supply the register sizes and bit numbering systems for the following:

   (a) IR (d) DR (g) BR
   (b) UR (e) OR (h) MAR
   (c) MMR (f) R (i) GR

2. How many bits does an MMR read?
3. How many bits does a scratch pad-read read?
4. How many bits does an ROM-read read?
5. What is the cycle of time of MM? ROM? SPM?
6. List the functions specified in the F field.
7. What is the binary configuration in the F field for a set function?
8. What are the differences between set function and the set and perform function?
9. Determine the operations specified by the F and V fields:

	F				V			
a	0	0	1		0	1	0	1
b	1	0	0		0	0	0	1
c	1	1	1		0	1	0	1
d	1	1	0		0	1	1	1
e	0	1	0		1	1	1	0
f	1	0	1		1	0	1	1
g	0	0	1		1	0	0	0
h	0	1	1		0	0	0	0
i	0	1	1		1	1	1	0
j	0	1	0		1	0	0	0
k	1	1	1		1	1	0	0
l	1	0	0		1	1	0	0

10. Match the bit positions of the C field with the appropriate control flip-flops:

BAA	C4
BDB	C7
BAB	C5
WDB	C6

11. Explain the use of WDB.

12. If BAA/BAB/BDB is specified in a set and perform function, will it be triggered and then set to the value of C0?

13. List the bit settings of the M field for the following:
(a) SPM read
(b) SPM write
(c) Register-to-register operation
What is the only bit configuration that is illegal in the M field?

14. Write the binary configuration of the S, D, and M fields to accomplish the following transfers (assume PS1, HSM = 00's):

	S	D	M
(a) IR → UR			
(b) UR → OR:R			
(c) IR → MARW			
(d) MMR → FPC (C7 = 1)			
(e) FPC → MAR (C7 = 1)			
(f) IR → MMR			
(g) → G			
(h) G → IR			

15. Assume in Problem 14 that IR contains (1234_{16}). What will the contents of each of the following be at completion of Problem 14?

IR	_____	FPC	_____
UR	_____	G	_____
OR	_____	DR	_____
R	_____	MAR	_____
MMR	_____		

16. Write an EO program in binary to add UR to DR right (binary add)

$$DR = 11223344_{16}$$
$$UR = 4334_{16}$$

17. What are the settings of the following registers at the completion of Problem 16?

IR	_____
BAA	_____
BAB	_____
UR	_____

18. Write a program in assembler language to multiply the contents of SPM 22 by 4.

19. Assume

$$\begin{aligned} \text{UR} &= \text{FFFF}_{16} \\ \text{DR} &= \text{FFFFFFFF}_{16} \\ \text{IR} &= 432\text{A}_{16} \end{aligned}$$

(a) Write a program that reverses the contents of IR so that IR contains $34\text{A}2_{16}$.
(b) Assuming the same operands, write a program that results in the following conditions:
(1) UR $\text{A}234_{16}$
(2) IR $\text{A}234_{16}$
(3) DR A234 $\text{A}234_{16}$
(4) MM at location 234A_{16} = $\text{A}234_{16}$

20. Assume

$$\begin{aligned} \text{SPM 22} &= 0000124\text{E}_{16} \\ \text{SPM 70} &= 0000126\text{A}_{16} \\ \text{SPM 71} &= 00001280_{16} \end{aligned}$$

Write a program in assembly language that compares the MM locations specified by SPM 72 and 73.
If (72) = (73), Branch to 124E_{16}.
If (72) > (73), Branch to 126A_{16}.
If (72) < (73), Branch to 1280_{16}.

21. Assume

$$\begin{aligned} \text{SPM 70} &= 0000126\text{A}_{16} \\ \text{SPM 71} &= 0000128\text{A}_{16} \\ \text{SPM 22} &= 00003100_{16} \end{aligned}$$

Write a program that subtracts 1 from the field specified by the MM address in SPM 70. If the field at the address specified by SPM 70 = 0, branch to the address specified by SPM 71.

22. Write a program that moves the contents of MM location 1000 through 1002 to MM locations 1003 through 1005. Assume

$$\begin{aligned} \text{SPM 70} &= 1000_{16} \\ \text{SPM 71} &= 1003_{16} \\ \text{MM 1000} &= 010203040506070809_{16} \end{aligned}$$

23. Write a program that transfers N number of bytes from MM 1000_{16} to MM 2000_{16}. Assume

$$\begin{aligned} N &= 63_{10} \\ \text{SPM 70} &= 1000_{16} \\ \text{SPM 71} &= 2000_{16} \\ \text{SPM 74} &= N \end{aligned}$$

24. Code Problem 1 for the assembler and simulator where:

$$\begin{aligned} \text{SPM 74} &= 13_{16} \\ \text{MM 1000} &= 090807060504030201_{16} \end{aligned}$$

$$1009 = 0A0B0C0D0E0F1011_{16}$$
$$1010 = 1213141516171819_{16}$$
$$2000\text{-}2020 = FF_{16's}$$

25. Trace the data processing action in the two microroutines shown in Fig. 9.16.

PROGRAM Microroutine Trace Exercise PROGRAMMER DATE

Exercise-1 ADD: (R1)+R(2) ⇒ R1

LABEL	I SP	TEST	A	N	M	FROM S	TO D	F	V	C	EXCEPT E	CONST.
1	7	8	14	20	26	28	34	40	43	48 55	55	60
RADD1			RADD1			FFR1	UR	SETB	BADD	S S		
RADD2			RADD2			FFR2		PR2	BYTE	S TT		
RADD2		TRUE	00000			IR	FFR1	NOP		S	ENDIN	
EXREG	S	E-2	ASSUM	E SPM	F	2=(00000	01234)16					
				SPM	F	1=(00000	05544)16					
			D	0 W←	U	.V						
ALFA1			ALFA2			F1	UR	SETB	ANDI	S SS		
ALFA2			ALFA3			F2		PR2	BYTE	TT		
ALFA3		TRUE	00000		J	R	F1	SCCN	O	S	ENDIN	

Figure 9.16 Trace microroutine.

REFERENCES

1. "Spectra 70/45 Processor EO Flow Charts," RCA Document #70-45-101, March 1966.
2. Benjamin, R. I., "The Spectra 70/45 Emulator for the RCA 301." Communications of the ACM, December 1965. Vol. 8, No. 12, p. 748–752.
3. Vollbrecht, John R., "MIDAS—Microprogramming Design Aid System for RCA Spectra 70/45." Presented at the ACM Workshop on Microprogramming, Oct. 7–8, 1968; Bedford, Mass. (No proceedings were published.)
4. RCA internal document which "provides a detailed description of the microprogrammed algorithms for the 501 emulator in the Spectra 70/45."
5. "70/45 EO Format Description," RCA Internal Document, December 11, 1967.