SYSTEM /360 O. S. LOGIC
TASK MANAGEMENT
Course Outline

I. Introduction

- A. What is a Task
- B. Where Do They Come From
- C. Task States

II. Task Related Routines

- A. Task Switching Routine
- B. Dispatcher

III. Interrupt Handling

- A. What is an Interrupt
- B. SVC Interrupt Handling
  1. First Level Interrupt Handler
  2. Second Level Interrupt Handler
  3. Transient Area Handler
- C. Exiting Procedures
  1. Type I Exit Routine
  2. Exit Routine
- D. Program Interrupt Handling
  1. First Level Interrupt Handler
  2. Program Interrupt Element (PIE)
  3. Exit Routine
- E. External Interrupts
- F. I/O Interrupts
- G. Machine Check Interrupts

IV. Task Supervisor

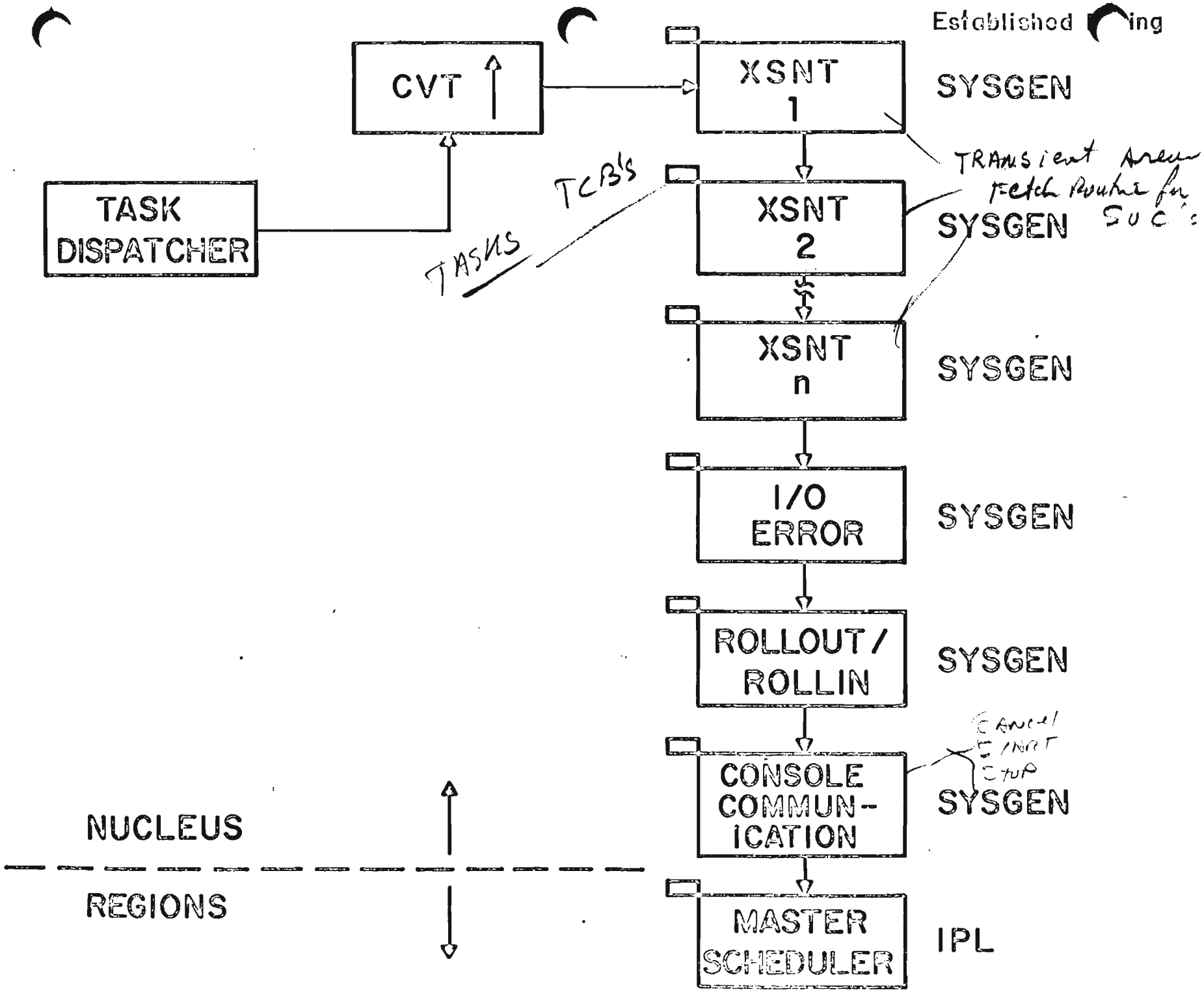
- A. Attaching a Subtask
- B. Detaching a Subtask
- C. Task Termination

V. Contents Supervisor

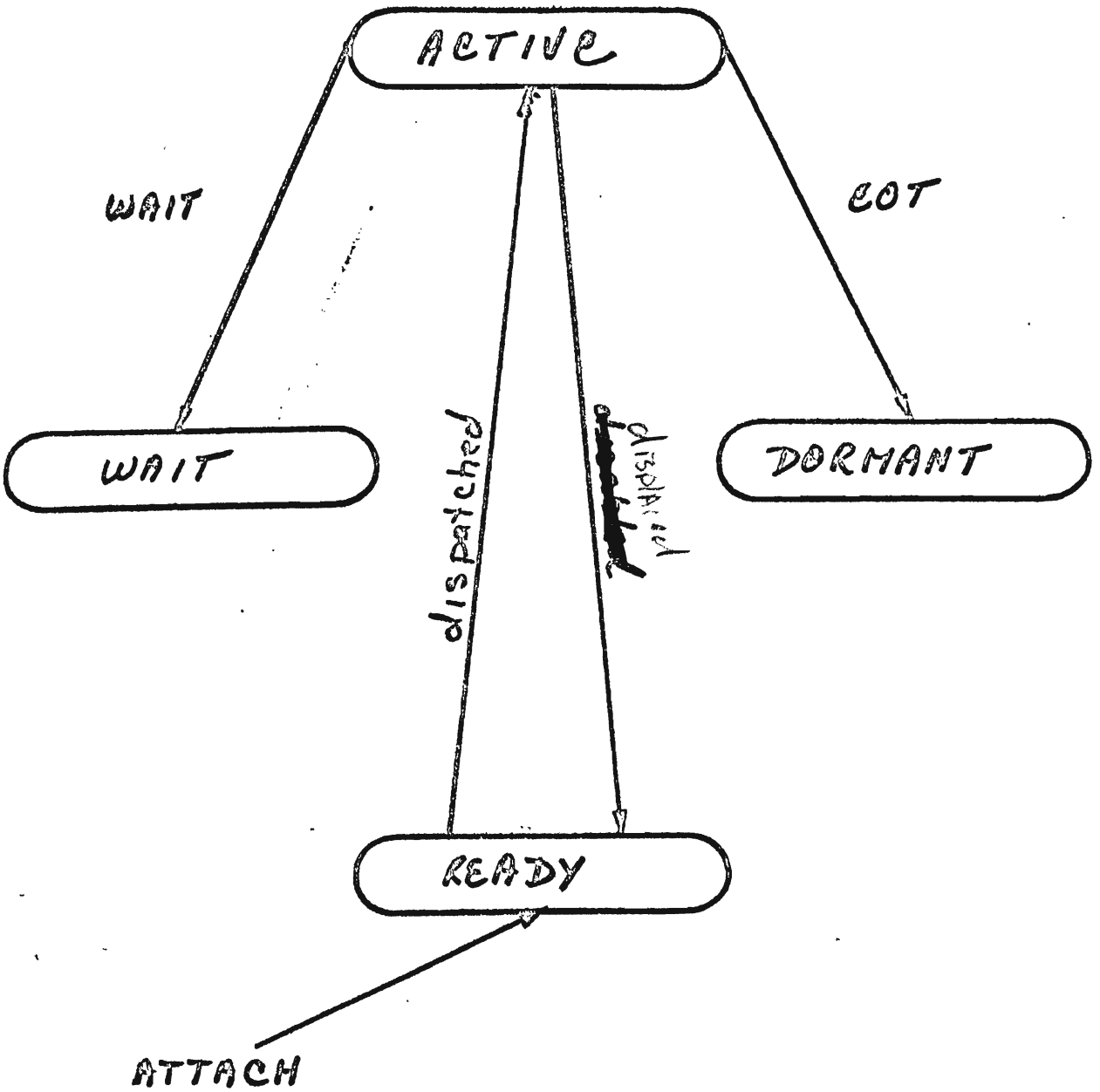
- A. Searching for Modules
- B. Releasing Modules
- C. Special Functions

VI. Main Storage Supervisor

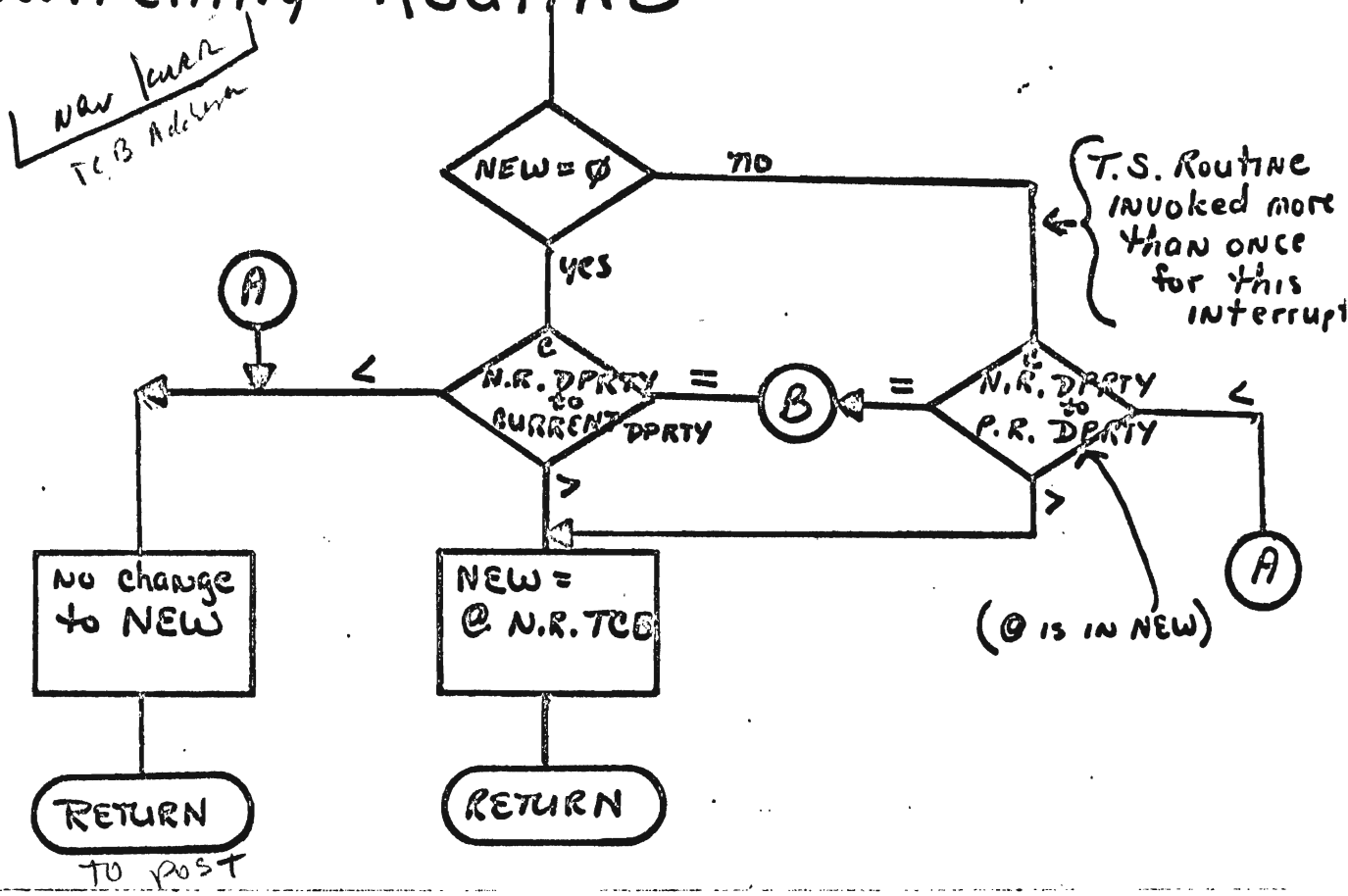
A. Allocating of Main Storage
B. Freeing of Main Storage

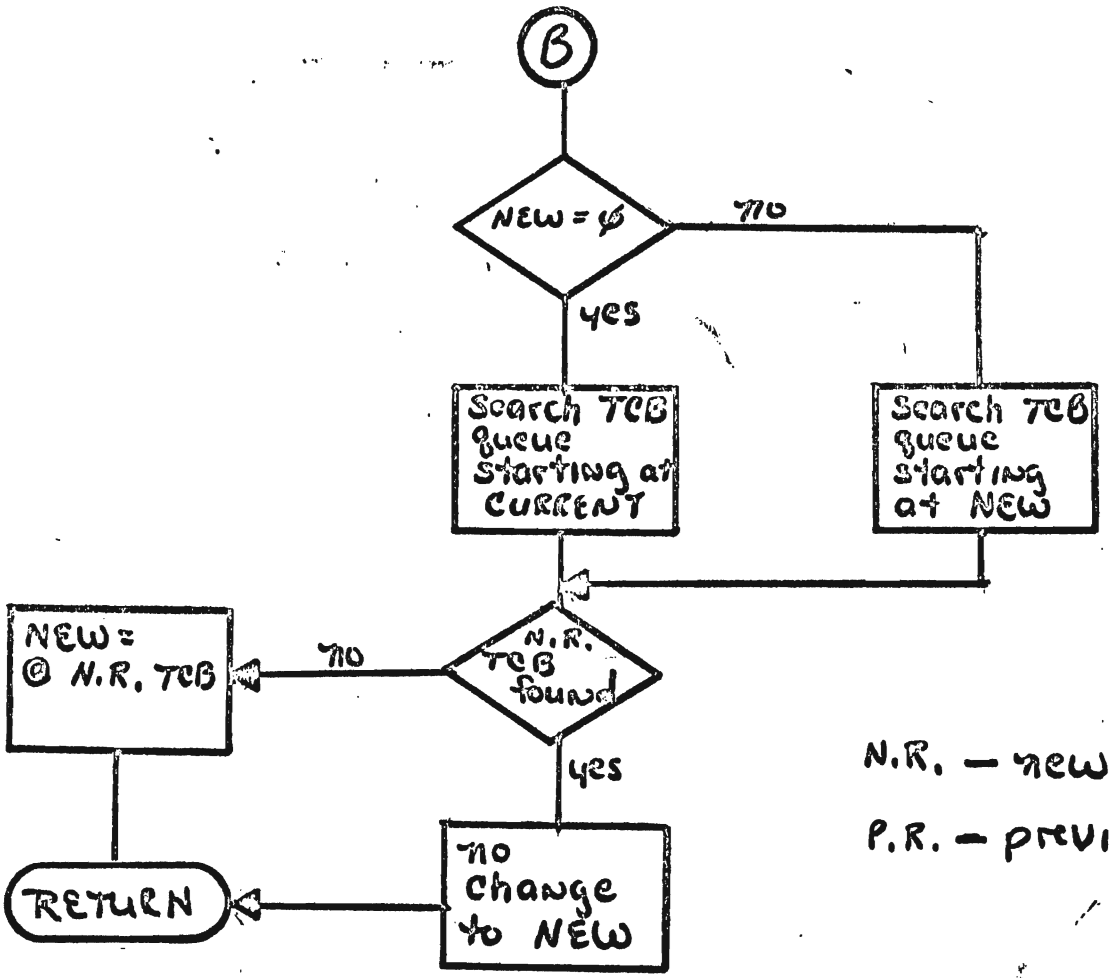

Established
CVT
XSNT 1
SYSGEN
TRANSient Area Fetch Routine for SVC's
TASK DISPATCHER
TCB's
TASKS
XSNT 2
SYSGEN
XSNT n
SYSGEN
I/O ERROR
SYSGEN
ROLLOUT/ ROLLIN
SYSGEN
CANCEL
EINIT
CTOP
CONSOLE COMMUN- ICATION
SYSGEN
NUCLEUS
REGIONS
MASTER SCHEDULER
IPL

TASK STATUS


ACTIVE
WAIT
EOT
WAIT
DORMANT
dispatched
preempted
READY
ATTACH

TASK SWITCHING ROUTINE





N.R. — newly readied

P.R. — previously readied

CONTROL BLOCK DATA

TCB Pointer Block

POINTED TO BY:

CVTTCBP (CVT +0) .

GENERAL FUNCTION:

Used by task dispatcher to determine if a task switch is required and if so, which task should get control

GENERAL CONTENTS:

New and old TCB pointers

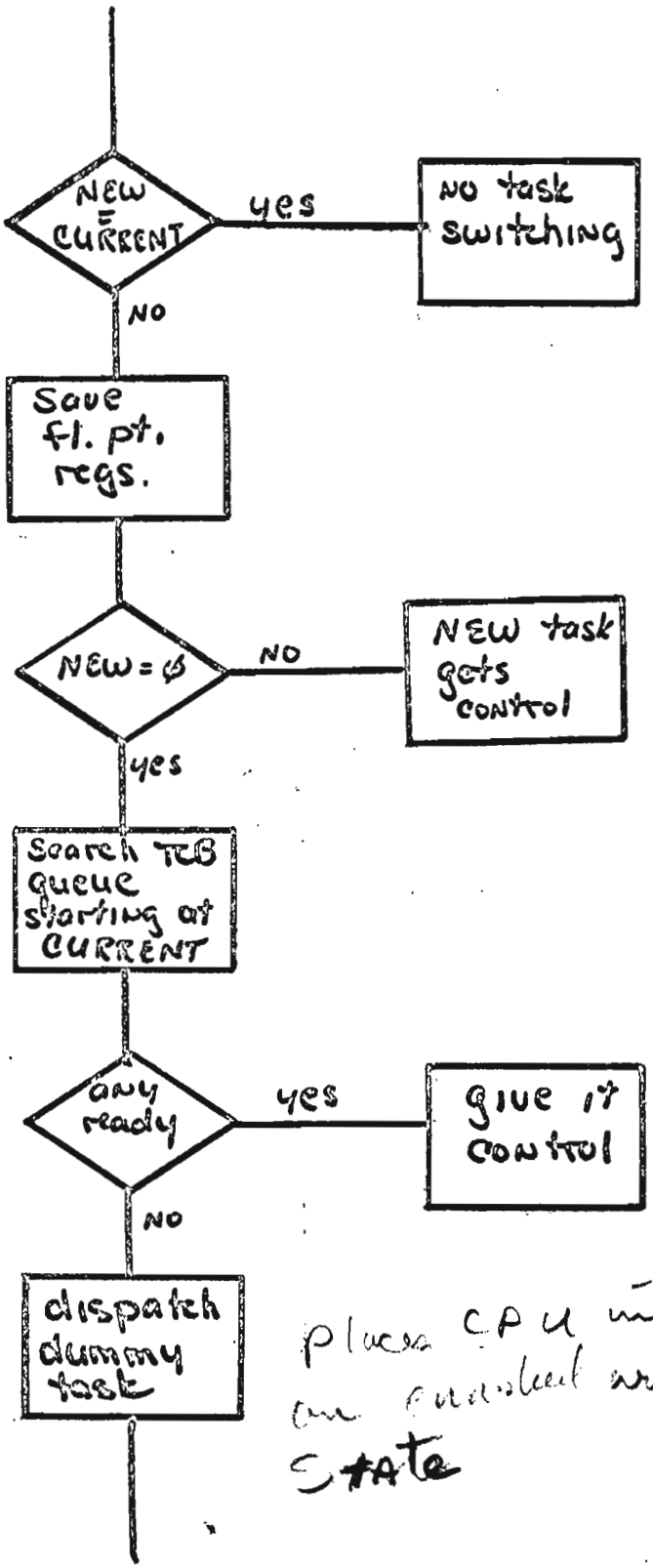
New = old task switch not required

New = old and new = 0 switch to new required

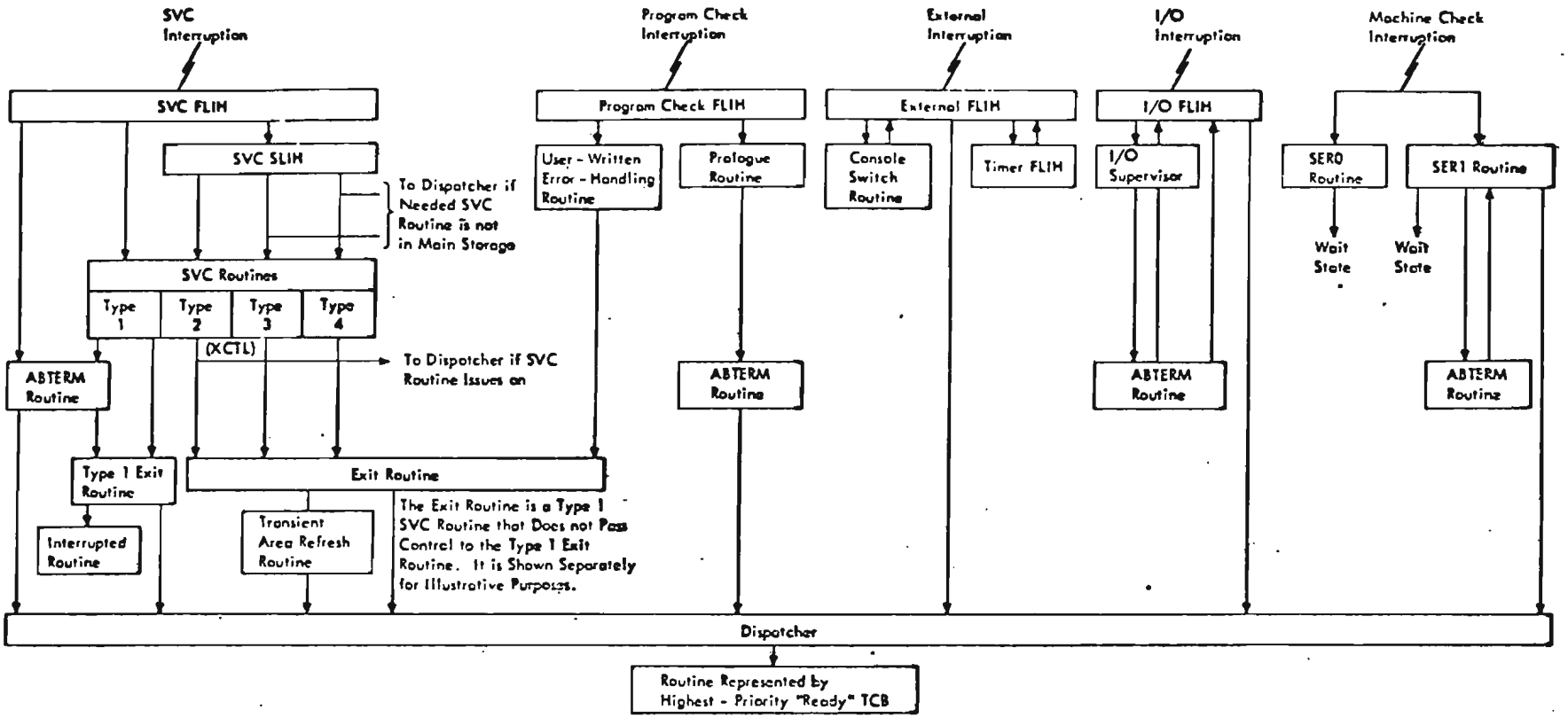
New = old and new = 0 dispatcher determines which task to get control based on priority.

New can be set by any of the supervisory routines associated with task switching (wait, post, ENQ/DEQ, manual purge)

DISPATCHER



place CPU in enabled wait state



Type 1 SVC Routines	Type 2 SVC Routines	Type 3 SVC Routines
CHAP EXIT EXTRACT FREEMAIN GETMAIN POST TIME TTIMER WAIT	Attach Delete DEQ Detach ENQ Exit Effector, Stage 1 Identify Link, Load, XCTL and Synch Overlay Supervisor Spie STIMER	STAE Service WTO, WTOR, WTL
		Type 4 SVC Routines
		ABDUMP ABEND Checkpoint Comm. Task Router Log and Writelog Post Restart

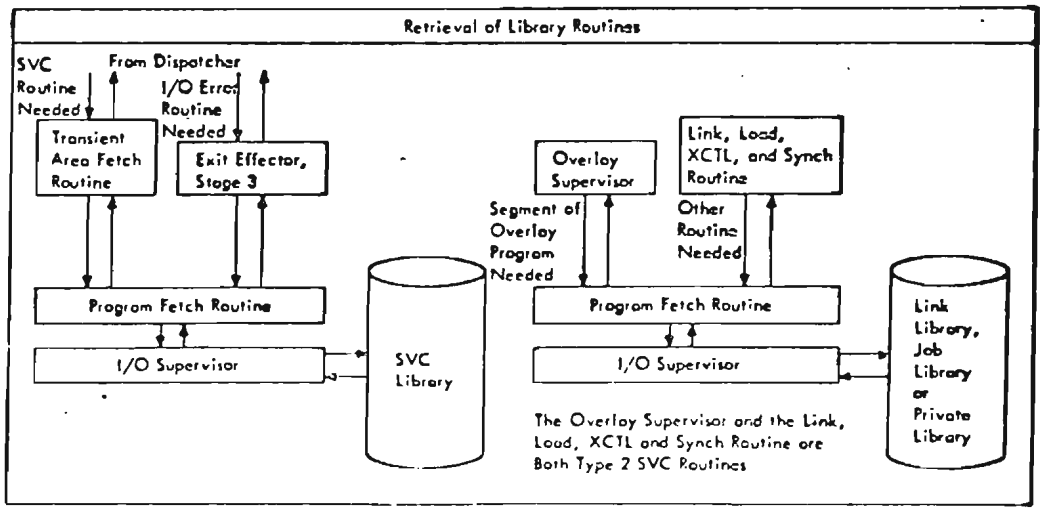


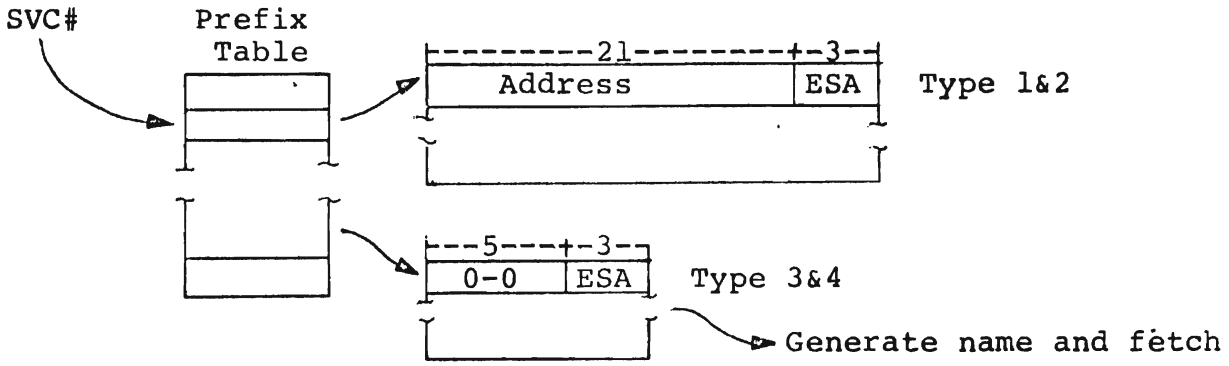
Chart 00. Overall Control Flow of MVT Supervisor

SVC ROUTINE TYPE VERSUS ROUTINE CHARACTERISTICS

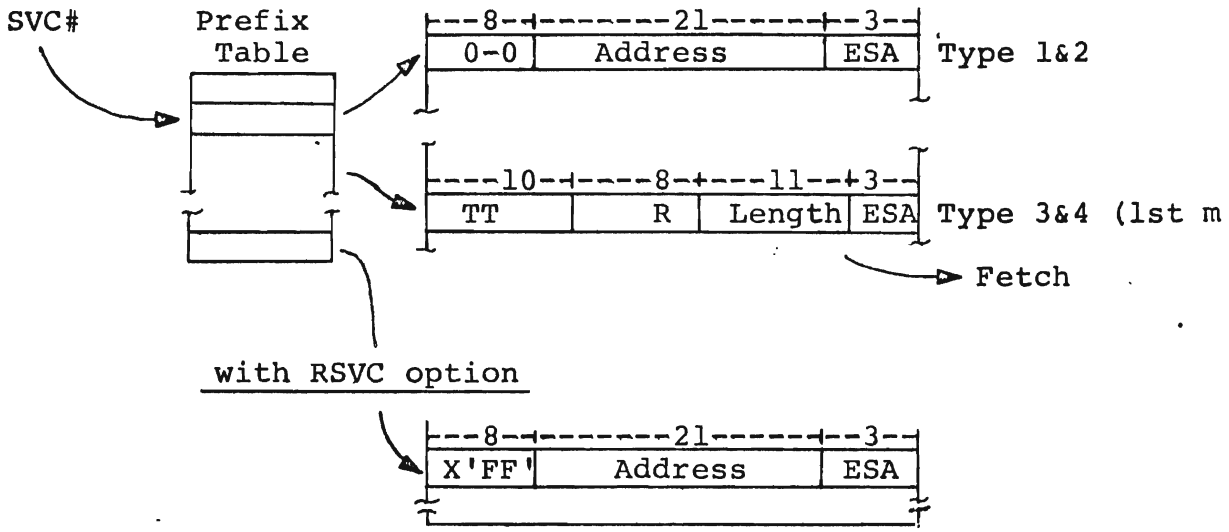
ROUTINE TYPE	CHARACTERISTICS
TYPE 1	SERIALLY REUSABLE OR REENTRANT, NON-INTERRUPTABLE
TYPE 2	REENTRANT, INTERRUPTABLE
TYPE 3	REENTRANT, INTERRUPTABLE, SINGLE ROUTINE LESS THAN OR EQUAL TO 1024 BYTES.
TYPE 4	REENTRANT, INTERRUPTABLE, n ROUTINES LESS THAN OR EQUAL TO 1024 BYTES. ENTERED BY XCTL MACRO.

SVC TABLES

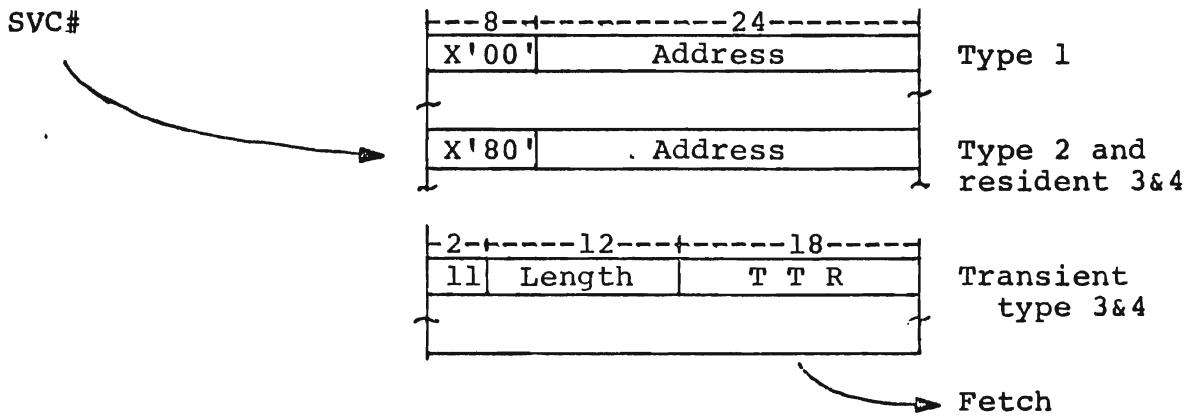
PCP/MFT without extended SVC table option



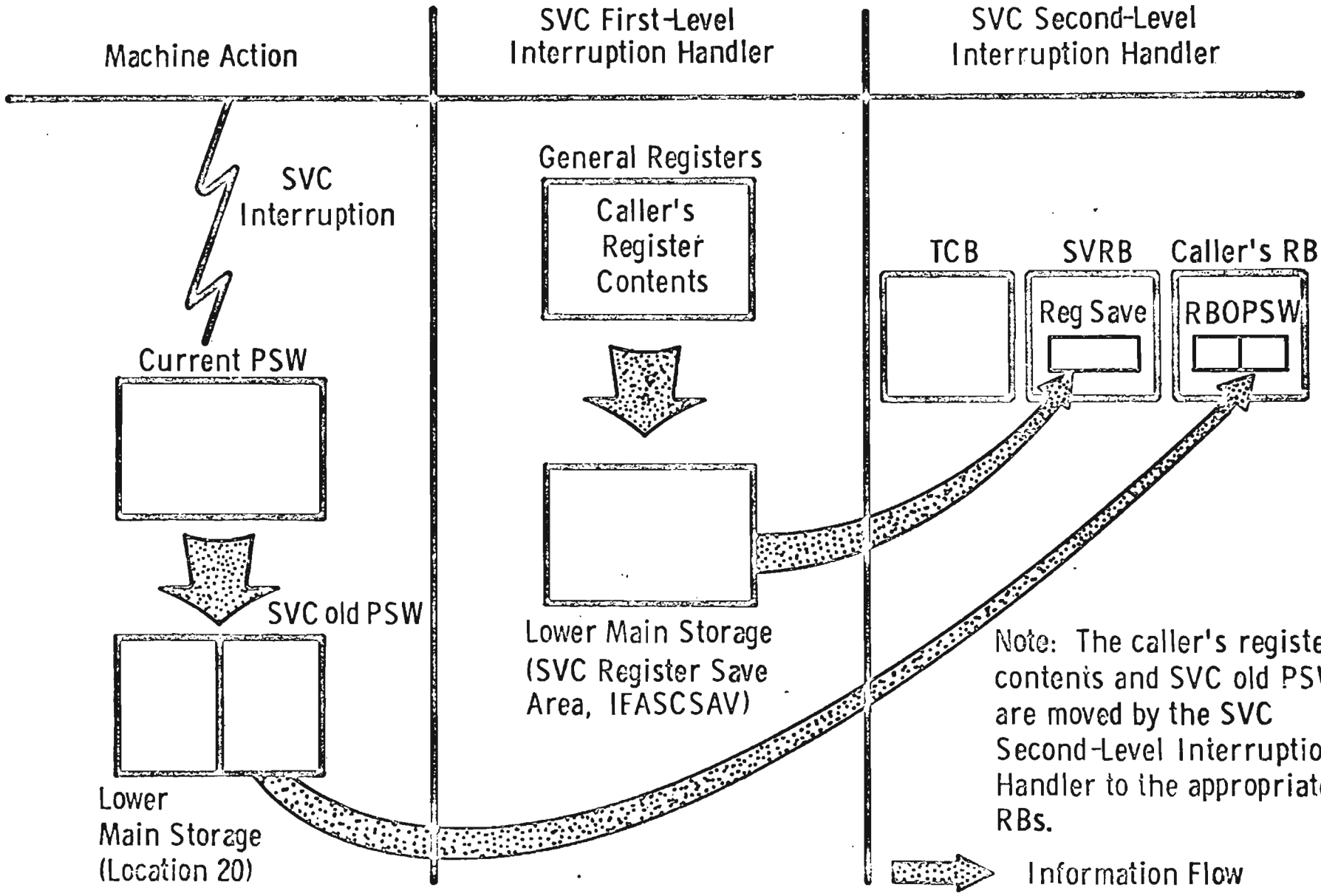
PCP/MFT with extended SVC table option



MVT



STATUS SAVING BY THE SVC INTERRUPTION HANDLERS



SVC SUPERVISOR

Save status of requestor

Determine type of SVC

Create SVRB when necessary

Locate SVC routine

Schedule unsatisfied requests (MVT)

Refresh SVC transient area(s)

CONTROL BLOCK DATA

SVRB - Supervisor Request Block

POINTED TO BY:

TCBRBP (TCB + 0) active or top RB

GENERAL FUNCTION:

Maintains information concerning the use of a Type II, III or IV SVC routine

GENERAL CONTENTS:

See PRB

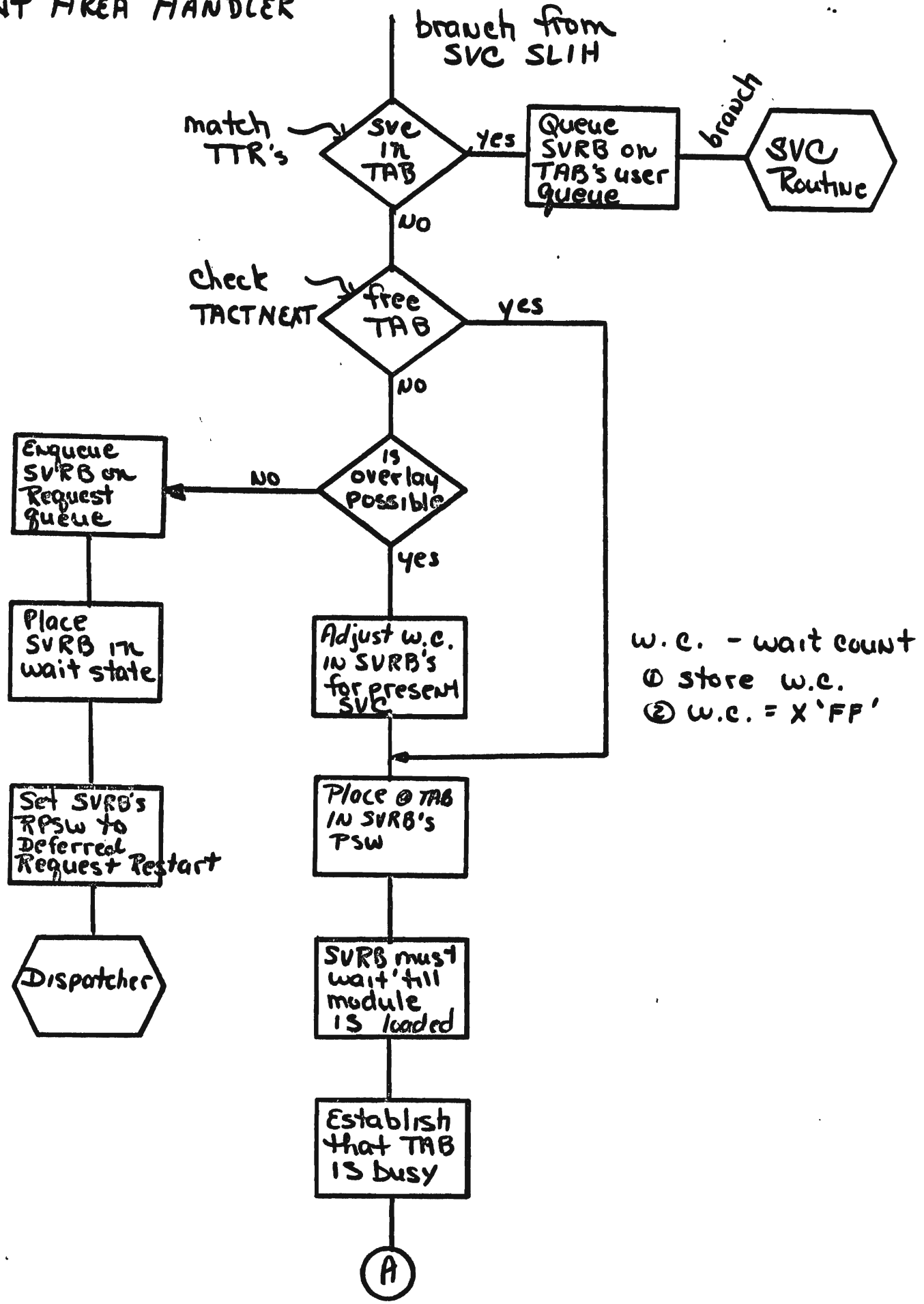
LAYOUT CAN BE FOUND IN:

PCP & MFT - SCB

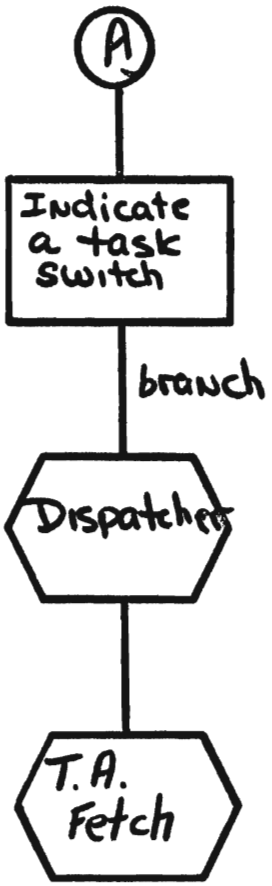
MVT - SCB

MVTS

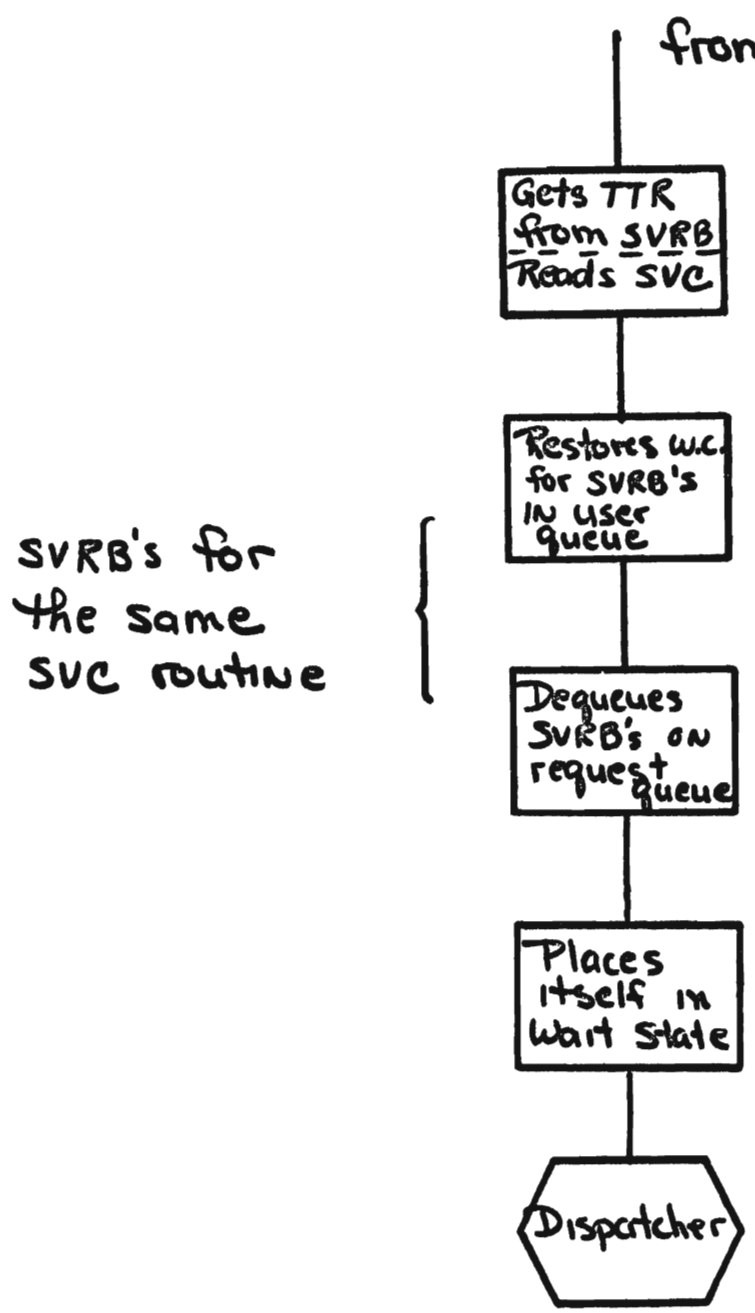
TRANSIENT AREA HANDLER



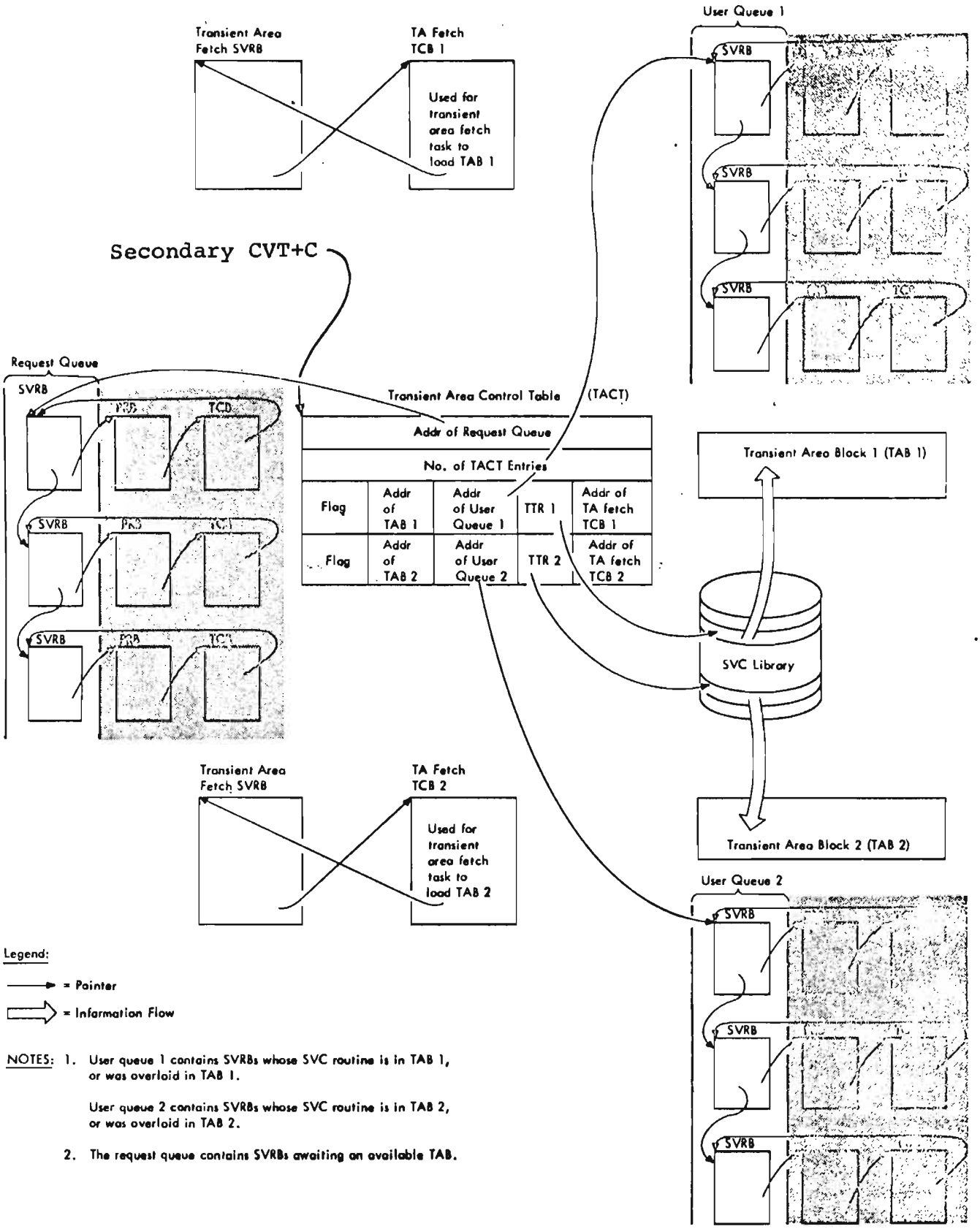
T.A.H.



Transient Area Fetch



w.c. - wait count



Legend:

→ = Pointer

⇨ = Information Flow

NOTES: 1. User queue 1 contains SVRBs whose SVC routine is in TAB 1, or was overlaid in TAB 1.

User queue 2 contains SVRBs whose SVC routine is in TAB 2, or was overlaid in TAB 2.

2. The request queue contains SVRBs awaiting an available TAB.

The Transient Area Queues

REGISTER POINTERS ON ENTERING AN SVC ROUTINE

REGISTER	MEANING
3	Communication Vector Table (CVT) Pointer
4	Task Control Block (TCB) Pointer
5	Supervisor Request Block (SVRB) Pointer for Types 2, 3 and 4. Last Active Request Block for Type 1 Routines
14	Contains return address
0, 1, 15	Used for passing information between routines. Not restored.

ASPECTS FOR CONSIDERATION IN WRITING SVC ROUTINES

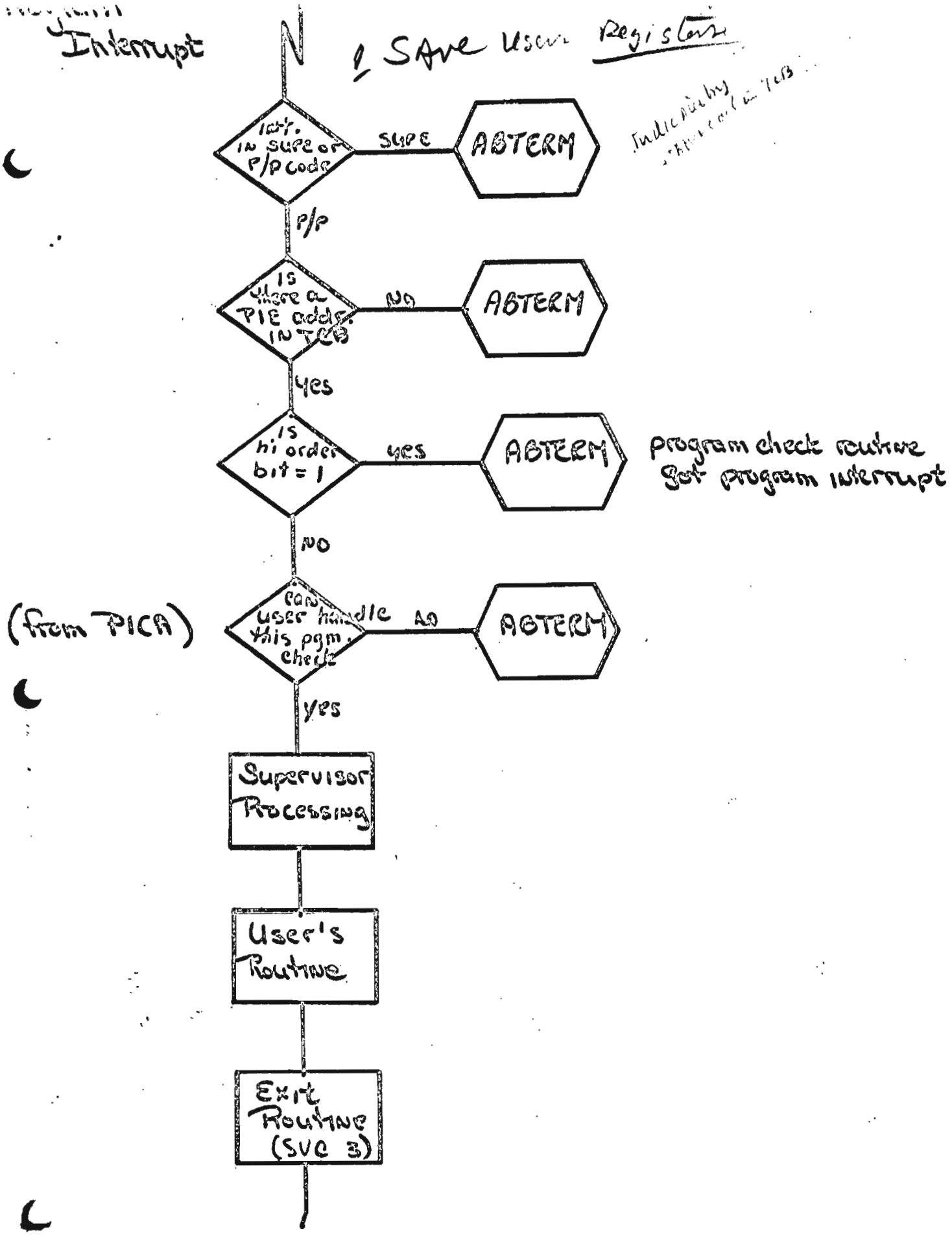
(1) Use proper control section name

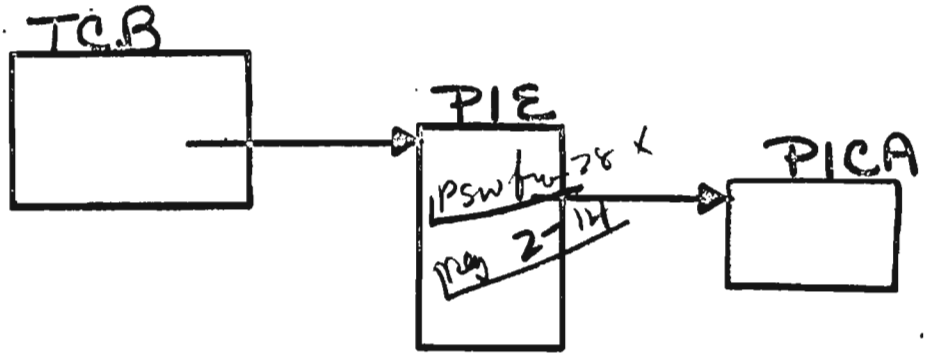
IGCnnn	CSECT	Type 1 and 2
or		
IGC00nnn+	CSECT	Type 3
or		
IGCssnnn+	CSECT	Type 4

(2) Use proper return or exit

BR	14	All types if register 14 has been saved
XCTL	IGCssnnn+	All but last load module of Type 4 routines
SVC	3	Calls EXIT routine. Types 2, 3 and last load module of Type 4.

(3) Do not modify instructions or data in the routine itself. Instead, use register storage and/or GETMAIN macro.


Interrupt
N
SAve User Registers
Intr. in Supe or P/P Code
Supe
ABTERM
P/P
Is there a PIE addr. in TCB
No
ABTERM
Yes
Is hi order bit = 1
yes
ABTERM
Program check routine Set Program Interrupt
No
(from PICA)
Can user handle this pgm. check
No
ABTERM
yes
Supervisor Processing
User's Routine
Exit Routine (SVC 3)



PIE

↑ PICA	
pgm. OPSW	Int. code
R14	↑ SVC 3
R15	↑ Routine
R0	control pgm. info.
R1	↑ PIE
R2	from P/P

} Contents of these regs. when Routine gets control.

PICA

0, pgm mask	↑ Routine	Int. type

CONTROL BLOCK DATA

PIE - Program Interruption Element

POINTED TO BY:

TCBPIE (TCB + 4)

GENERAL FUNCTION:

Provide user control of program interruptions - constructed by use of SPIE MACRO

GENERAL CONTENTS:

PTR to users PICA address

Old PSW save area

Register save area

LAYOUT CAN BE FOUND IN:

PCPS

MVTS

CONTROL BLOCK DATA

PICA - Program Interruption Control Area

POINTED TO BY:

PIEPICA (PIE + 0)

GENERAL FUNCTION:

Provide mask and entry point of user's error routine.

GENERAL CONTENTS:

Program mask to be used in PSW

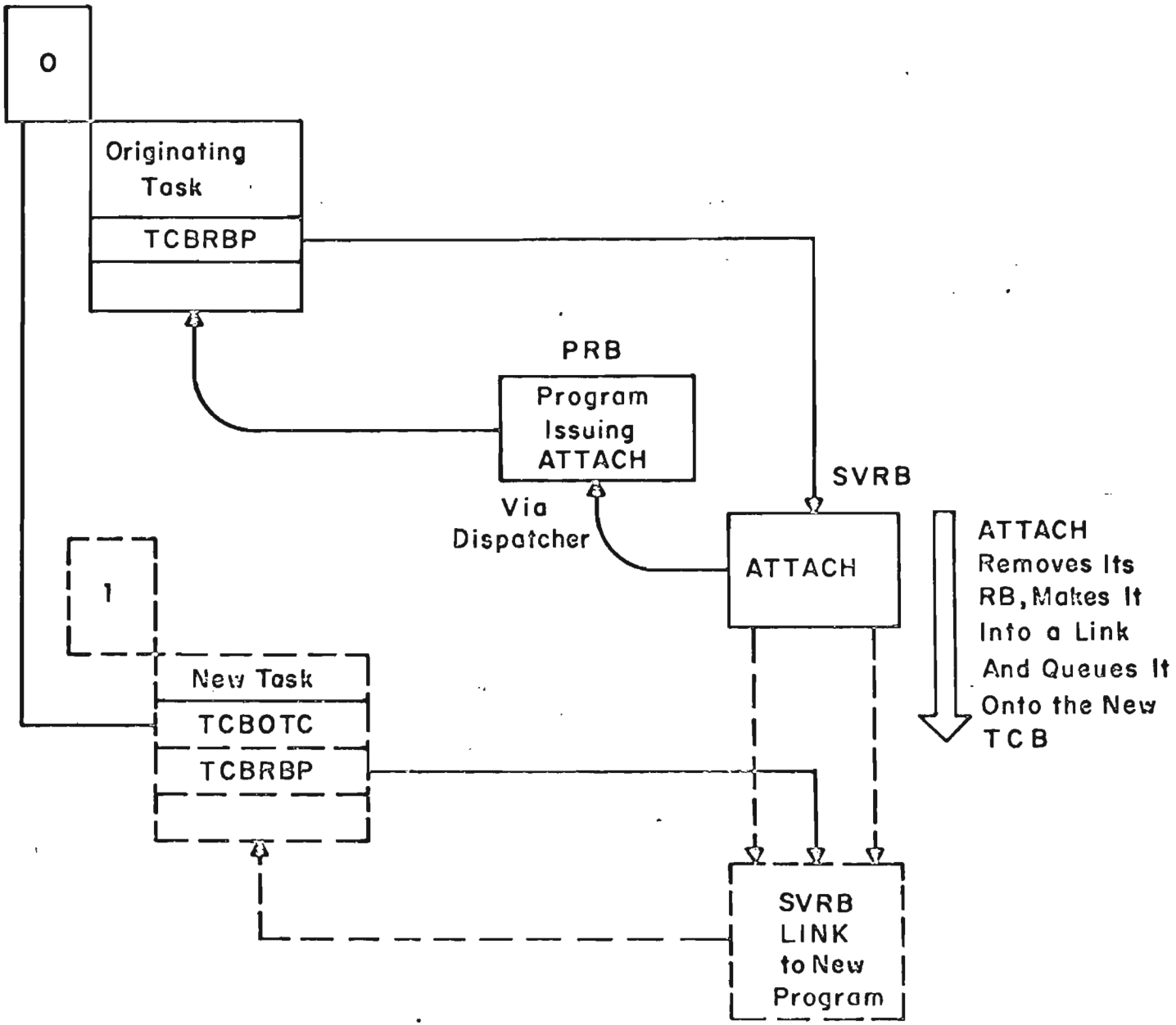
Pointer to user's interruption exit routine

Mask for program interruption types to be handled by user

LAYOUT CAN BE FOUND IN:

MVTS

ATTACH PROCESSING



CONTROL BLOCK DATA

TCB - Task Control Block

POINTED TO BY:

CVTHEAD (CVT + A0) - Highest priority TCB

GENERAL FUNCTION:

Defines a task to the system

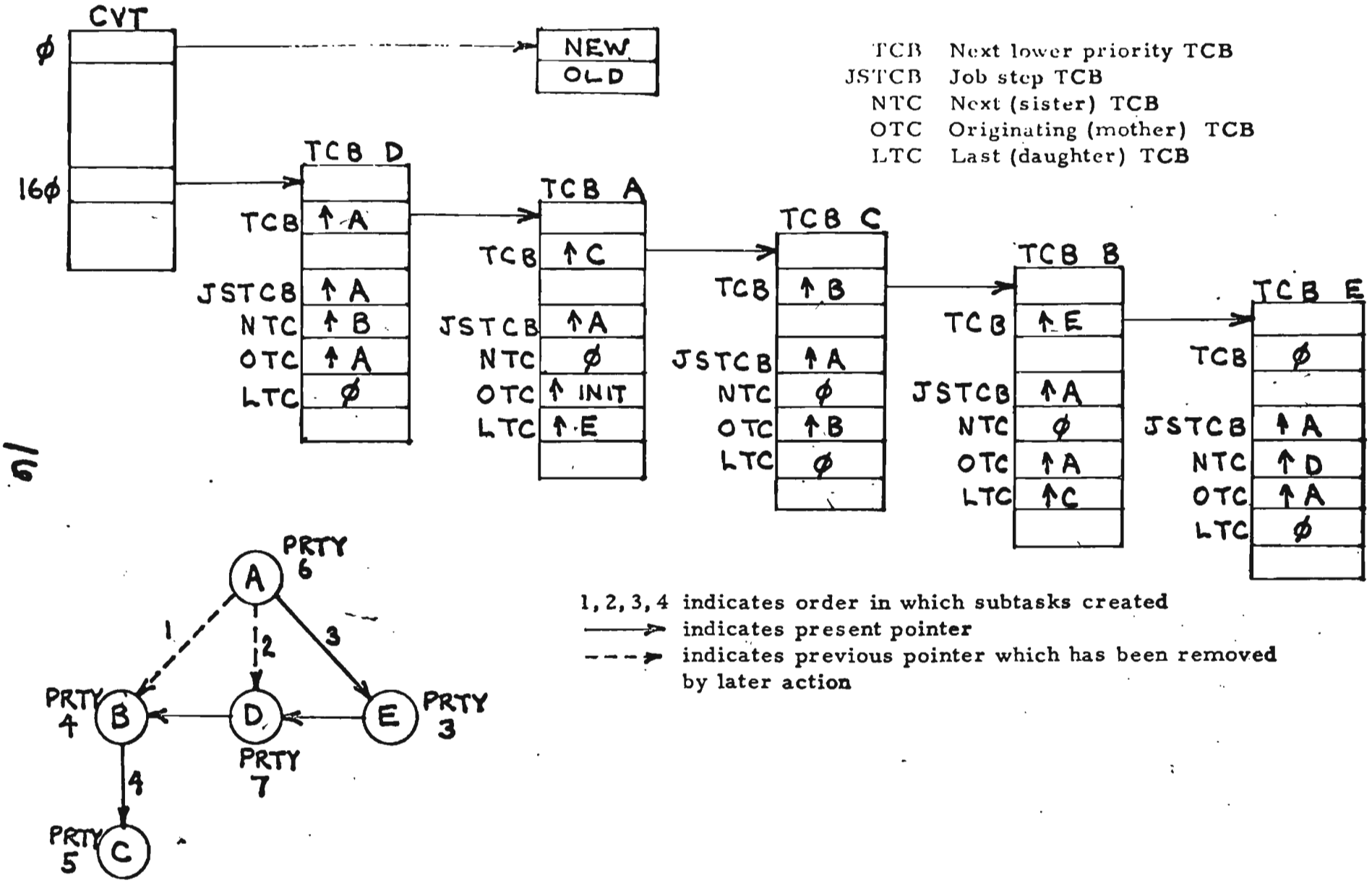
GENERAL CONTENTS:

Information and pointers associated with the task in progress.

LAYOUT CAN BE FOUND IN:

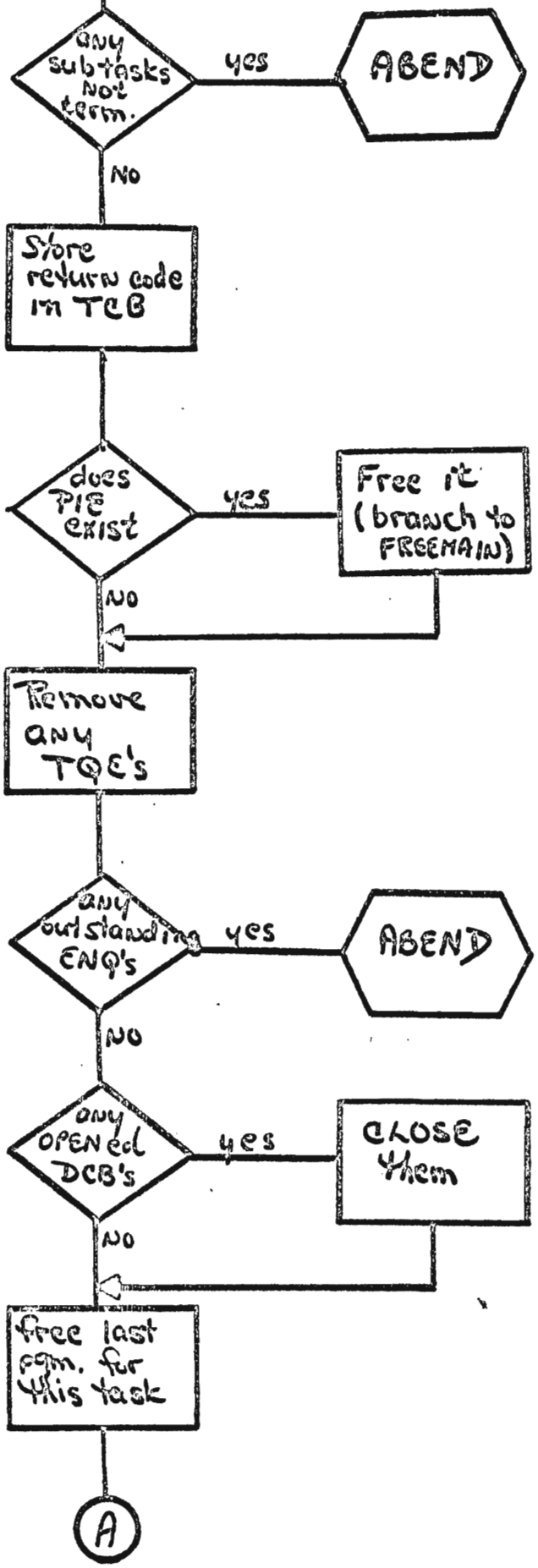
SCB

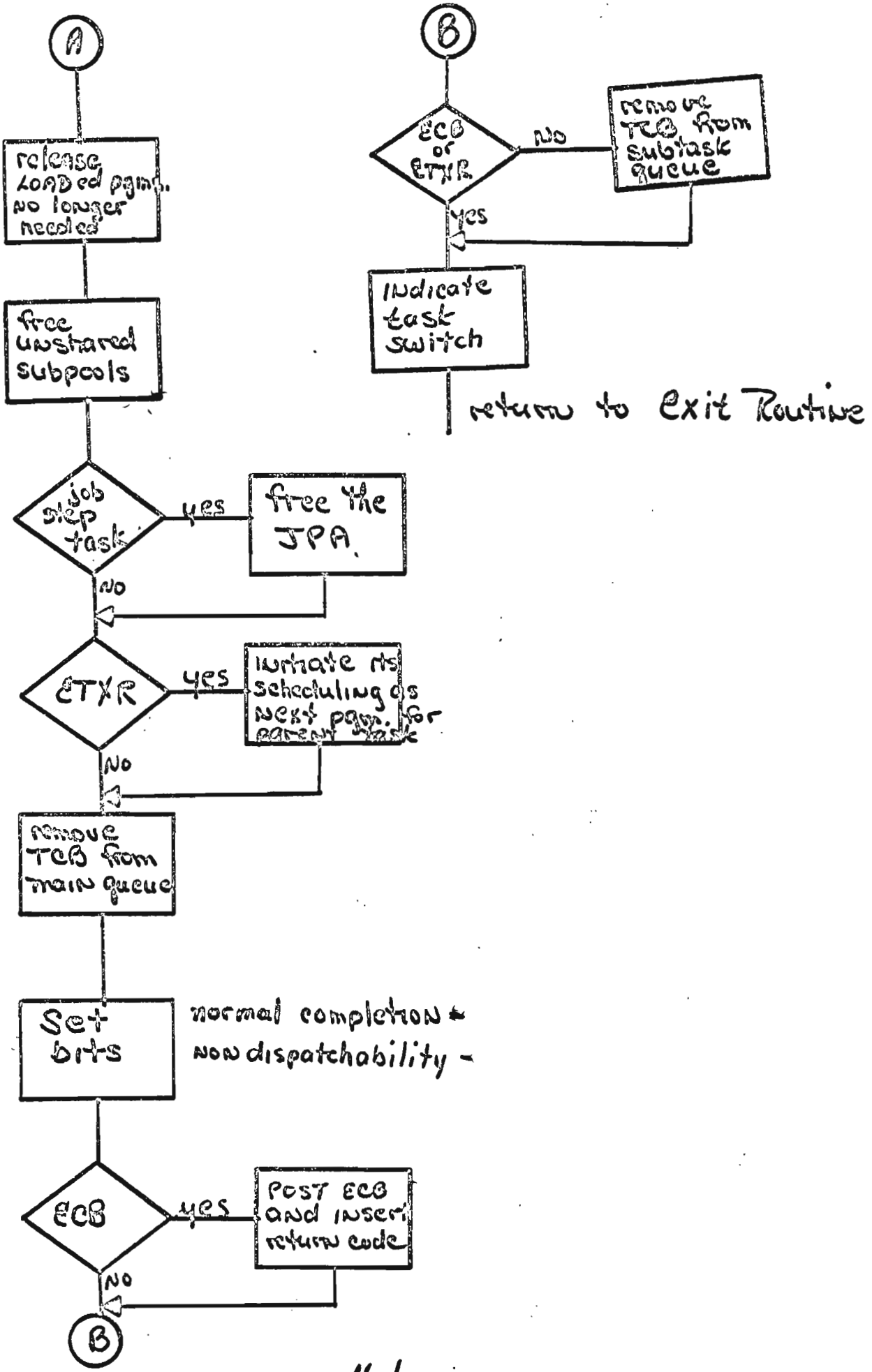
MVTS

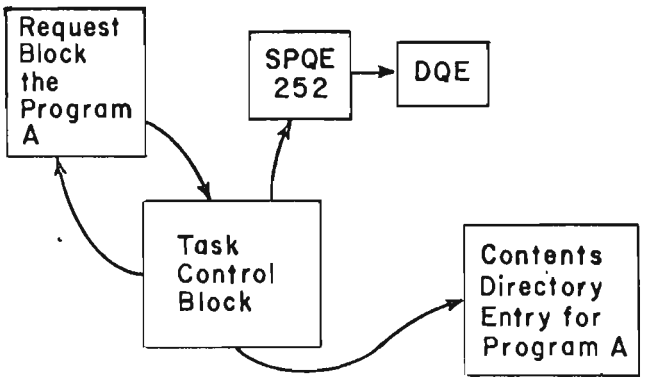


MVT TCB PRIORITY QUEUE

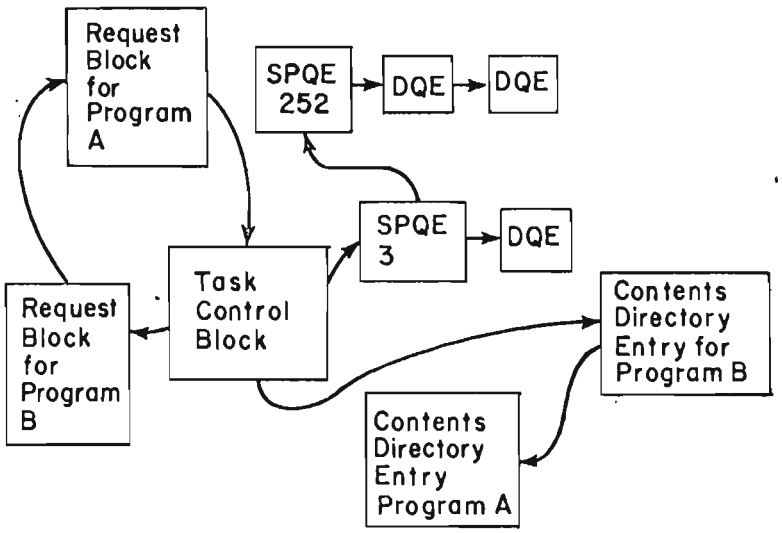
NORMAL TASK TERMINATION



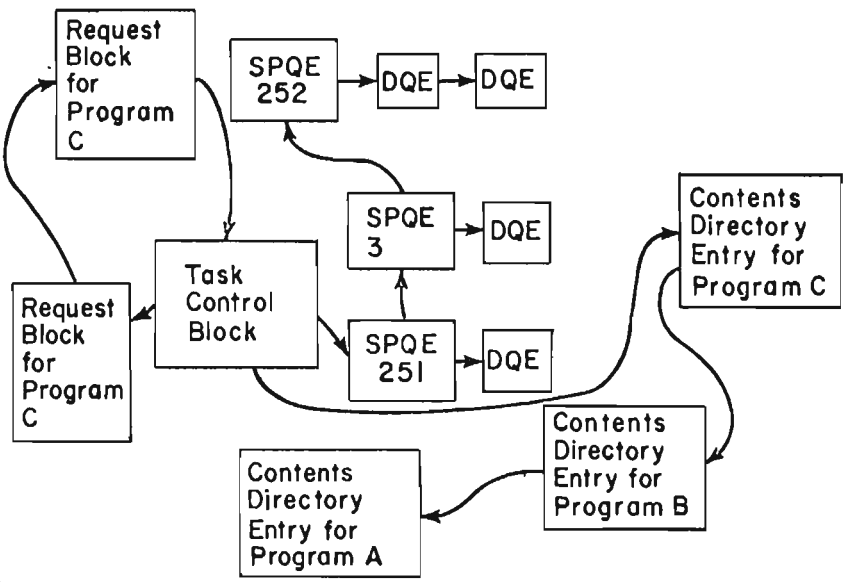

A
release LOADed pgms. no longer needed
free unshared subpools
Is job step task
yes
free the JPA
no
ETXR
yes
Initiate its scheduling as next pgm. for parent task
No
remove TCB from main queue
Set bits
normal completion & non dispatchability -
ECB
yes
POST ECB and insert return code
No
B
B
ECB or ETXR
No
remove TCB from subtask queue
yes
Indicate task switch
return to Exit Routine



(1) Example of the Modification of the Content Directory During a Task



(2) Example of the Modification of the Content Directory During a Task



(3) Example for the Modification of the Content Directory During a Task

CONTROL BLOCK DATA

CDE - Contents Directory Element

POINTED TO BY:

RBCDE (RB + D)

GENERAL FUNCTION:

Defines existance of a non-SVC module in core.

GENERAL CONTENTS

Attributes of the module, name of module, number of current users, entry point, pointer to extent list.

CDE's are chained together through CDCHAIN (DCE +0). Each search may be concerned with 2 CDE chains - link pack area and the job pack area for the region. The link pack area chain is pointed to by CVTQLPAQ (CVT +BC). The job pack area chain is pointed to by the job step TCB at TCBJPQ (TCB +2C).

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

PRB - Program Request Block

POINTED TO BY:

TCBRBP (TCB + 0) active or TOP RB

GENERAL FUNCTION:

Maintains information concerning a non-supervisory routine required for this execution of the module.

GENERAL CONTENTS:

Differs by OS configuration:

PCP & MFT - contains all information for this execution of the module, e.g., module name, attributes, size, resume address if interrupted, etc.

MVT - Contains only the dynamic information for this use of the module

RB's for a task are chained together through the link field (XRBLNK or RBLINK) at RB+1C. The last RB in the chain points back to the controlling TCB.

LAYOUT CAN BE FOUND IN:

PCP & MFT SCB

MVT - MVTS

SCB

CONTROL BLOCK DATA

LLE - Load List Element

POINTED TO BY:

TCBLLS (TCB + 24) beginning of chain.

GENERAL FUNCTION:

Maintain count of number of outstanding load's of a module.

GENERAL CONTENTS:

Pointer to next LLE in chain pointer to CDE for the model.
Use count.

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

XL - Block Extent List

POINTED TO BY:

CDXLMJP (CDE + 14)

GENERAL FUNCTION:

Provide contents supervision with information regarding a particular block of a module plus note lists for overlay structured modules.

GENERAL CONTENTS:

Location of block.

Size of block.

TTR of each overlay segment.

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

GOVRFLB

POINTED TO BY:

Location GOVRFLB in module IEAQGM resident nucleus.

GENERAL FUNCTION:

Origin list for main storage queues.

GENERAL CONTENTS:

PTR to beginning of dynamic area.

PTR to DQE describing SQA.

PTR to PQE describing unassigned main storage.

LAYOUT CAN BE FOUND IN:

MVTS

DPQE - Dummy Partition Queue Element

POINTED TO BY:

GOVRFLB + 8 free core - TCBPQE (TCB + 98) allocated region.

GENERAL FUNCTION:

Pointers to beginning and end of PQE chain.

GENERAL CONTENTS:

Pointer to first PQE in chain pointer to last PQE in chain. In order to locate DPQE, add 8 to the value located in GOVRFLB or TCB.

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

PQE - Partition Queue Element

POINTED TO BY:

DPQE points to beginning and end of chain.

GENERAL FUNCTION:

Used to define a region.

GENERAL CONTENTS:

Backward and forward PQE pointers

Pointers to first and last FBQE's

Pointer to job step TCB owning the region

Size of the region

Core address of the region

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

FBQE - Free Block Queue Element

POINTED TO BY:

PQEFFBQE (PQE + 0) first PQEBFBQE (PQE + 4) last.

GENERAL FUNCTION:

Define free block (multiple of 2K) of core within an allocated region.

GENERAL CONTENTS:

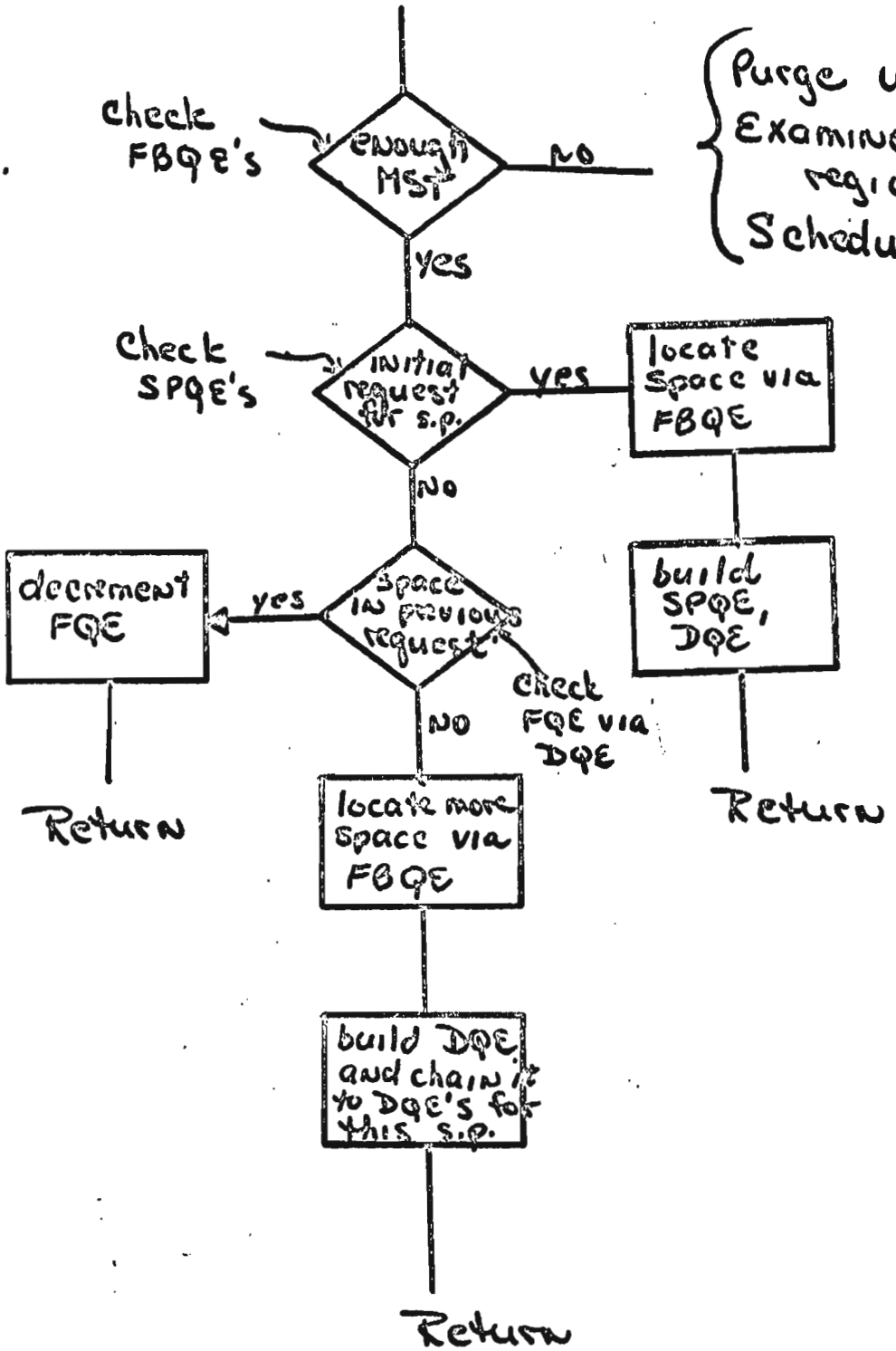
Forward and backward FBQE pointers.

Size of free block.

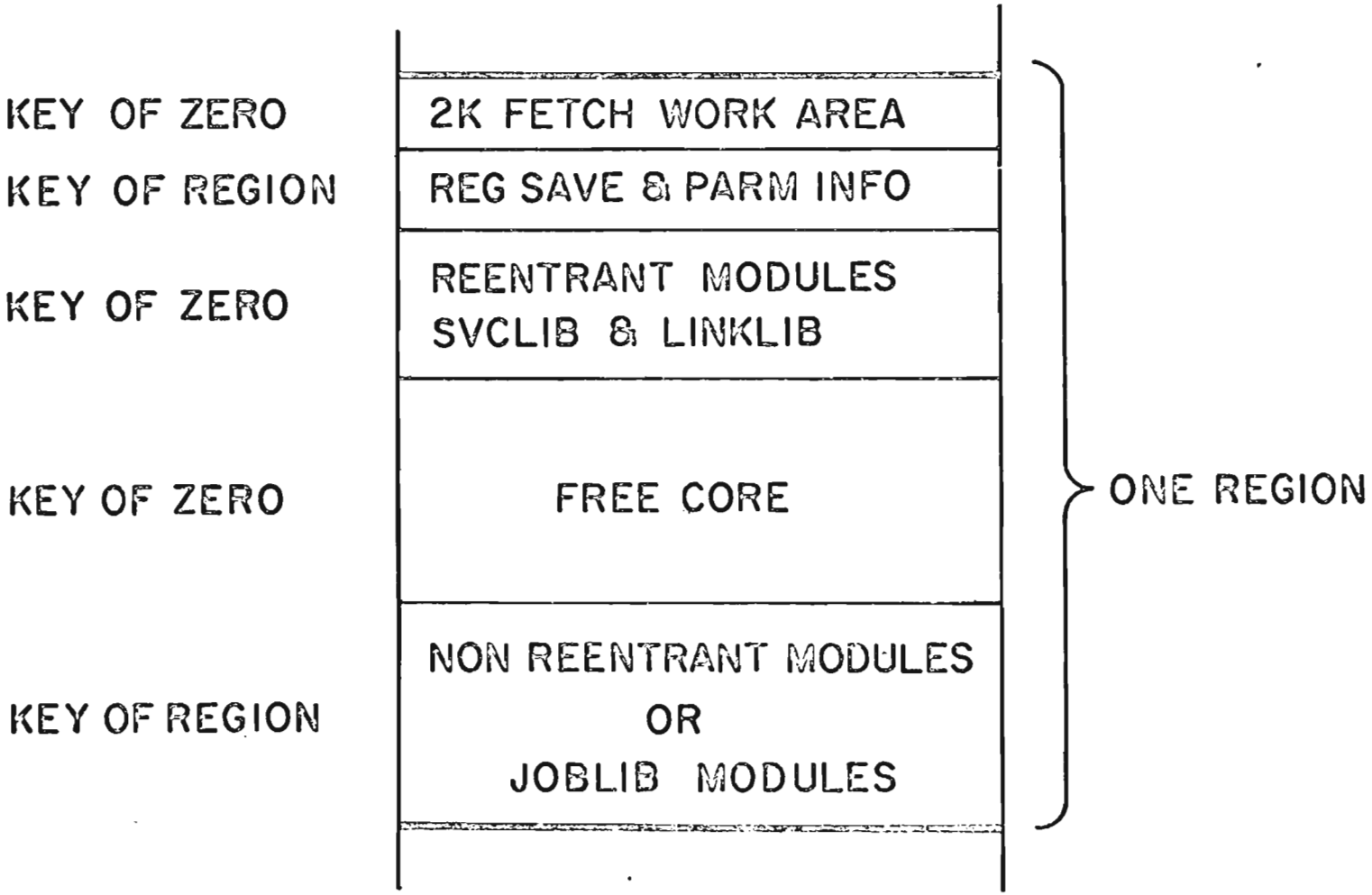
LAYOUT CAN BE FOUND IN:

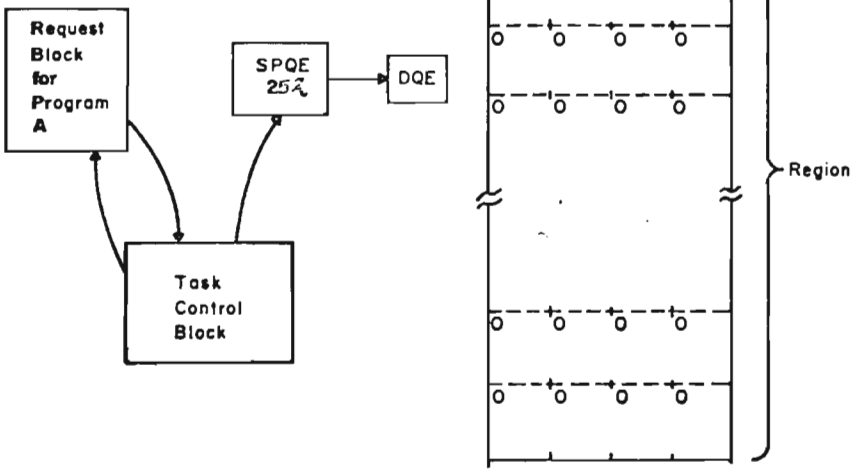
MVTS

GETMAIN

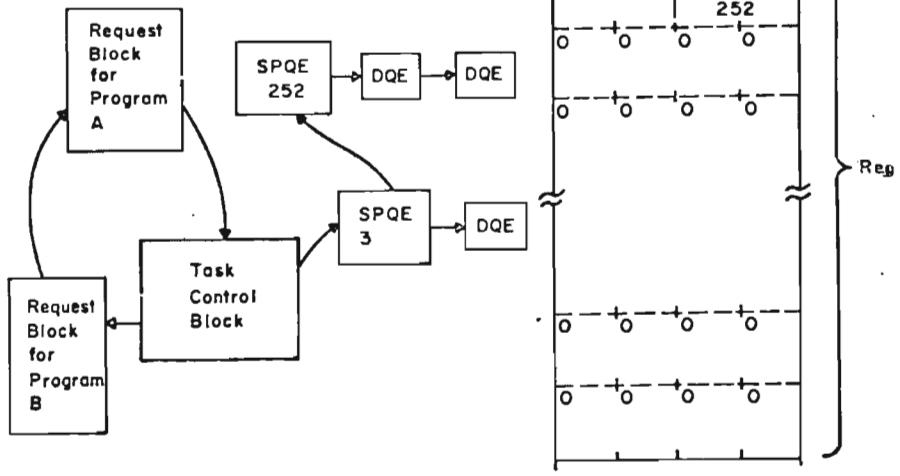

Check FBQE's
enough MSP
NO
Purge unused modules
Examine previously borrowed region (RO/RI)
Schedule RO/RI
yes
Check SPQE's
initial request for S.P.
yes
locate Space via FBQE
NO
decrement FQE
yes
Space in previous request
build SPQE, DQE
Check FQE via DQE
NO
Return
Return
locate more space via FBQE
build DQE and chain it to DQE's for this S.P.
Return

CORE STORAGE ORGANIZATION WITHIN A PROCESSING PROGRAM REGION

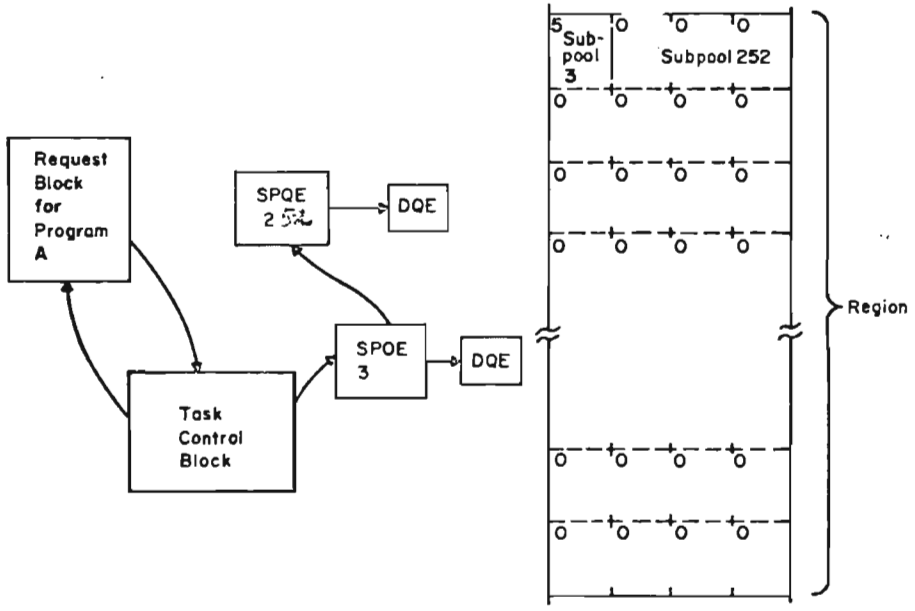




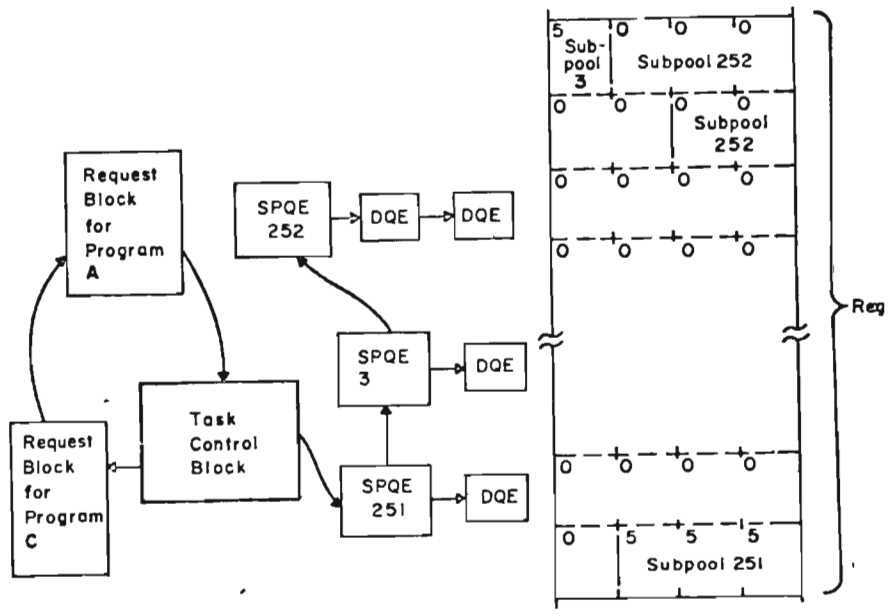
Example of Main Storage Allocation ①



Example of Main Storage Allocation ③



Example of Main Storage Allocation ②



Example of Main Storage Allocation ④

CONTROL BLOCK DATA

SPQE - Subpool Queue Element

POINTED TO BY:

TCBMSS (TCB + 18) beginning of chain

GENERAL FUNCTION:

Define existence and identity of a subpool

GENERAL CONTENTS:

Pointer to next SPQE in chain

Status of subpool, e.g., owned, shared or both identity - subpool number

Pointer to first DQE, or to owning SPQE

One SPQE will exist for each subpool in a region.

May have multiple subpools 1-127 within a region, each one owned by a different task

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

DQE - Descriptor Queue Element

POINTED TO BY:

DQEPTR (SPQE + 5) Beginning of chain.

GENERAL FUNCTION:

Defines a block (multiple of 2K) allocated for a subpool.

GENERAL CONTENTS:

Pointer to first FQE in the block.

Pointer to next DQE for same subpool.

Address of the block.

Size of the block.

LAYOUT CAN BE FOUND IN:

MVTS

CONTROL BLOCK DATA

FQE - Free Queue Element

POINTED TO BY:

FQEPTR (DQE + 0) MVT (BBX + 0) MFT

GENERAL FUNCTION:

Defines contiguous free core.

GENERAL CONTENTS:

Pointer to next FQE.

Number of bytes of free core defined by this FQE.

LAYOUT CAN BE FOUND IN:

MVTS

Logic

Tentative Calendar

Day 1

Introduction to Operating Systems
- Development and Purposes
- OS/360 Concepts and Terminology

Assignment: Read - MVT Control Program Logic Summary (Y28-6658)
MVT Job Management PLM (Y28-6660)
Introduction and Part I

Day 2

Interrupts and Tasks
- Interrupts
- TCB and RB's
- Task Switching Routines
- Dispatcher

System Initialization Overview

IPL Procedure
- LOAD button
- NIP
- Master Scheduler Initialization

Assignment: Read - MVT Job Management (Y28-6660)
Part 2, Part 3, Part 6 - Work Queues and I/O Device Allocation

Day 3

Job Management Overview

The Work Queue
- SYS1.SYSJOBQE
- QCR's
- Logical Tracks
- Enqueueing work
- Dequeueing work

Process Input
- Definition of Input

"Ordinary" Reader Interpreter

Assignment: Read - MVT Job Management (Y28-6660)
Part 3, Part 4, Part 5

Day 4

Process Input (continued)
ASB Reader
RJE Reader

Initiate, Terminate work
Initiator

Assignment: Read - MVT Job Management (Y28-6660)
Part 5, Part 6

Day 5

Initiate, Terminate Work
Terminator

Process Output
Command Processing
where commands may appear
Communication Task
Command Scheduling
Command Types

Assignment: Read - MVT Supervisor (Y28-6659)
Section 1, 2, 9

Day 6

Task Management Overview
Types of Interrupts and Handling of each
SVC Interrupts
Program Check Interrupts
External Interrupts
I/O Interrupts
Machine Checks Interrupts

Assignment: Read - MVT Supervisor (Y28-6659)
Sections 3, 4

Day 7

Task Supervisor

Task Creation (ATTACH)
Task Termination (DETACH)
Serializing use of a resource (ENQ, DEQ)

Contents Supervisor
Macros
Search for Module
Alias Processing
Special Processing
Use/Responsiblility counts

Assignment: Read - MVT Supervisor (Y28-6659)
Sections 5, 10

Day 8

Main Storage Supervisor
Space in a region
Space in SQA and Dynamic Area
Rollout/Rollin

Trace Table
Size and usefulness
Entrys
Trace Table Control

Termination Routines

Assignment: Read - I/O Supervisor (Y28-6616)
Introduction and Part I
I/O Support (Y28-6609), Introduction, opening a data set, closing a data set

Day 9

Data Management Overview

Volumes and Data Sets
DA volumes
Tape volumes

Preparation for I/O
Data set description
Access Methods
Control Blocks
Opening a data set
End appendages
EXCP Supervisor
Validity check

Schedule I/O Request

Assignment: Read - I/O Supervisor (Y28-6616)
. Section II, III

Day 10

I/O Interrupt Handler
Locate requestor
Error Routines
Restart channel

Logic

General Outline

I	Preface	I
	A. Check roster	I A.
	B. Badges	I B.
	C. Description of Course	I C.
	D. Intent (objectives) of Course	I D.
	E. Prerequisites	I E.
II	Introduction to Operating Systems	II
	A. Development and purpose of operating system	II A.
	1. History	
	2. Increase thruput	
	3. Decrease turnaround time	
	B. OS/360 Concepts and Terminology	II B.
	1. Resources	
	2. Effective use of resources, by computer program	
	3. Control Program Concept	
	4. Functions of Control Program	
	5. Processing Programs	
	6. OS/360 resource management routines	
	7. User interface with OS/360	
	8. Job and Step Concepts	

C. External Interrupts XIV C.

1. External FLIH

D. I/O Interrupts XIV D.

1. I/O FLIH

E. Machine Check Interrupts XIV E.

1. Recovery options

2. SER0

3. SER1

4. MCH

F. Asynchronous Exit Routines * XIV F.

1. Definition of

2. How specify such a routine

3. Scheduling of Asynchronous returns and control blocks involved

4. Exit from asynchronous routines

G. Exit procedures (review, for the most part) XIV G.

1. From type 1 SVC routine

2. From user program check handling routine

3. From routine controlled by a SVRB

4. From routine controlled by a PRB

5. From routine controlled by an IRB or SIRB

6. Common processing

H. Task Switching Routines (review) XIV H.

1. When entered and why

2. If NEW=CURRENT

OS/360 Logic
Detail Outline

I Preface I

A. Check Roster I A.

B. Badges I B.

C. Course I C.

1. hours - start, stop
2. length of course
3. no machine problems
4. heavy reading assignments
5. (no grades)

D. Intent of Course I D.

1. Show the parts of OS/360 and their interaction
2. Concepts developed to a detailed level but will not "bit bash"
3. Describe OS/360 but will not justify it nor rationalize it

E. Prerequisites I E.

1. BPT/GPT/SPT or 6 months programming experience
2. OS/360 User
3. Knowledge of BAL

A. Development and purpose of operating systems * II.A.

Ref: Introduction, Part 1 (C28-6534)

1. History

   a. Computers as machines to solve problems

   b. Fast and accurate

   c. Still, not used to capacity nor maximum efficiency

      1) CPU waiting much of the time

      2) Human played too large a roll in computer work

         a) had to program everything he wanted done

         b) operator setup time was too long

         c) human slowness and inaccuracy limited machine's usefulness

   d. much of work done by humans could be done by computer programs

      1) Space allocation on auxiliary storage devices

      2) Main storage location assignments

      3) Translation of instructions from programming to machine language

   e. Such programs could be shared by all users of an installation

      1) Functional modularity developed in programs

      2) Need for and provisions to link from one module to another were developed

f. CPU could "work on" several programs in main storage at once ("simultaneously")

2. Increase Thruput

a. Amount of work performed by a computer in a given time span is increased (thruput)

1) prewritten, commonly used programs increase the speed of execution of a module

2) execution of several programs "simultaneously" increases the amount of work performed in a unit of time

3. Decrease Turnaround time

a. A given program executes faster under newer computers

1) prewritten, commonly used programs increase speed of execution of a module

2) execution of several programs "simultaneously" increases the amount of work performed in a unit of time

B. OS/360 Concepts and Terminology II B.

1. Resources

a. As computers increased in complexity and flexibility, they came to be seen as collections of resources

1) humans

2) programs (in main storage)

3) data

4) CPU time

5) Main storage space

6) I/O channel time

7) Direct access storage space

8) I/O devices

9) the CPU

2. To effectively and completely use the resources, computer programs were written

   a. to allocate a particular resource to a program

   b. to monitor the use and function of the resource

   c. to provide other functions and services needed by most or all users of the computer

3. The programs that allocate and monitor a resource and service requests are known collectively as the control program

   Ref: Handout S1, V25-6156

   a. Since the control program is a program and programs are resources, the control program controls and monitors itself

   b. This concept is essential to an operating system - it interacts with itself; it is a resource to be allocated yet it does the allocating

   c. This "self-monitoring" is accomplished via the hardware/software interrupt scheme.

4. The Control Program has three main functions

   a. to accept and schedule work to be done (Job Mgmt.)

   b. to supervise each unit of work as it is done (Task Mgmt.)

   c. to act as a "cushion" between programs and different types of I/O devices, data formats, storage mediums, etc. (Data Mgmt.)

5. Programs, IBM or user written, that provide functions other than the control program are called processing programs

   a. Language Translators (Assembler, Compilers)

   b. Service programs (Link Editor, Loader)

c. Utilities

d. User programs

6. The resources of OS/360 and their managers are

a. Humans - Master Scheduler

b. Programs (in Main Storage) - Contents Supervisor

c. Data - Data Mgmt. (access methods)

d. CPU Time - Timer Supervisor

e. Main Storage - Main Storage Supervisor

f. I/O Channel Time - I/O Supervisor

g. Direct Access Storage Space - DADSM Returns

h. I/O Devices - Initiator and I/O Supervisor

i. CPU - Dispatcher

7. User Interface with OS/360

a. JCL - language by which user specifies

1) work he wants computer to do

2) sequence in which he wants it done

3) conditions under which he wants it done or skipped

4) data sets and devices his program will need

b. Linkage conventions

1) preserving registers

2) providing another save area

3) chaining the two areas

c. These interfaces are necessarily well defined and, in the case of JCL, rigidly enforced; it is by these conventions that the user

1) informs the system what he wants done and how

2) allows the system to schedule, monitor and execute his program the same way it does all others

d. Without adherence to standard conventions the concept of a flexible, generalized group of programs (the operating system) servicing varied user requests (e.g., doing work) would degenerate into the original state of computers - each use, each job, each application would have to be individually and sequentially set up and customized.

8. Job and Step Concepts

a. A JOB has been variously defined as

1) unit of work to a computing center

2) everything in input stream from one "//JOB" card to the next (or to end of file)

b. A step has been variously defined as

1) unit of work to a computer

2) everything in input stream from one "//EXEC" card to the next (or to next "//JOB" card or to end of file)

c. Both words are "input stream" oriented and are therefore, external and artificial in an attempt to understand OS/360 internals but a job must consist of one or more step

9. Configurations of OS/360

Ref: Storage Estimates for Main Storage Layouts of PCP, MFT, MVT systems (C28-6551)

a. The concept of a control program and its functions require that a portion of that program be in main storage at all times, this is called the nucleus

b. Various configurations of OS/360 differ in the number of destinct programs that can be in execution

at the same time and in the way main storage is assigned and utilized

c. PCP - Primary Control Program

1) Nucleus in low core

2) Optional reentrant, often used routines above it

3) rest of main storage available for the one program that can be executing at any time

d. MFT - Multiprogramming with a Fixed Number of Tasks

1) Nucleus in low core

2) System Queue Area (SQA) - protected area for control blocks - above nucleus

3) Optional, reentrant, often used programs above SQA

4) Rest of main storage divided into sections called Partitions

a) number of partitions and size of each determined when system initialized

b) several programs can be executing in each partition (can have idle partitions though)

e. MVT - Multiprogramming with a Variable Number of Tasks

1) Nucleus in low core

2) System Queue Area above nucleus

3) Link Pack Area (LPA) - often used, reentrant code, not optional - in high core. Modules in LPA can be used by any and all programs in the system

4) Master Scheduler Region just below LPA

5) Rest of main storage available for allocation

as regions - when region is requested for a new program, region is allocated from wherever sufficient space is available (contiguous space)

6) Regions are of varying size and as many can be in existence as dynamic area size will allow

7) One "main program" in each region (Jobstep task)

8) Any number of "sub programs" (subtasks) in the region, only limit is size of region.

10. Tasks, Jobstep and Sub

a. Tasks

Ref: Handout S8

1) a request for the execution of some code

note: Deliberate generality in the word "some", the program specified in the EXEC card or in ATTACH macro is not, by a long shot, all the code that will be executed as that task. Also, "request" means just that, system may not execute the specified code, if various conditions prevent it

2) that to which resources are allocated

3) competitor for system resources (or for CPU)

4) that which ABEND's

b. Every step of a JOB becomes a jobstep task when (and if) that step is executed

c. Tasks in OS/360 configurations

1) PCP - one task in the system at any time

2) MFT - can have more than one task per partition (can have idle partion though),

3) MVT - one jobstep task per region (no such thing as an idle region, unused dynamic space is available or unused), any number of subtasks per region

d. Programs (tasks) can issue ATTACH macro to create other tasks (subtasks)

   1) only tasks with protect key of 0 (generally system tasks) can create jobstep tasks

   2) user ATTACH's create subtasks

e. Every task is represented to the operating system by a Task Control Block (TCB) - a collection pointers and indicators used to keep track of various resources allocated to that task and to control and monitor the task

f. Jobstep and Subtask TCB's are identical in size and format, the fields which distinguish the two types of tasks are

   Ref: MVT Supervisor, Section 12, TCB format (Y28-6659)

   1) JSTCB - pointer to the jobstep task in the region, when a TCB contains its own address in this field, it is the jobstep task in the region

   2) OTCB - originating a mother task - points to the TCB of the task that created this one. Even jobstep tasks were created by another task (an initiator)

   3) NTCB - next or sister task - points to next older task among several tasks with a common mother. Each sister points to the next older sister, the oldest contains zeroes in this field.

   4) LTCB - last or daughter task - points to most recently created (youngest) subtask of this one. Every mother task points only to its youngest daughter, the daughters are chained amongst themselves (NTCB pointer) and all point to the common mother task (OTCB field)

   5) These pointers are used to maintain the subtask queue or "family relationships" between the tasks in a region

11. Tasks States

Ref: Handout S2

a. A task can be in one of several states during its life in the system

1) active - the CPU is executing the code of the task

note: There can be only one active task in a uniprocessing system.

2) ready - capable of using the CPU but doesn't have it

3) wait - not capable of using the CPU even if it had it

4) dormant - thru executing, system is performing housekeeping for the task before destroying the TCB

note: A task "exists" as long as its TCB does

b. A task moves from one state to another by various interrupts occuring

1) active to ready - interrupt occurs, some other task receives control (displaced)

2) ready to active - LPSW executed by dispatcher causes task to become active (dispatched)

3) active to wait - WAIT SVC issued by active task

4) wait to ready - system or another task POST's the event complete that the task was WAITing on

5) active to dormant - task executes final BR 14 thereby signalling its termination (via SVC 3 instruction at address in reg 14)

12. Tasks, Permanent System

Ref: MVT Supervisor, Section 12, after TCB format (Y28-6659)

a. The routines in the nucleus execute as tasks - the control program is a resource and must be monitored and controlled, the means of monitoring and con-

trolling any task is its TCB

b. Certain TCB's are assembled into the Nucleus and thus these tasks are "created" by loading the Nucleus. These tasks are permanent (their TCB's are never freed) and are system tasks (prot. key 0)

   1) one TCB for each SVC transient area (loads the SVC return into the associated area)

   2) System Error TCB (loads and executes I/O error recovery routines)

   3) Rollout/Rollin TCB (optional - performs Rollout/Rollin processing)

   4) Communications TCB (handles I/O from any system consoles)

   5) Master Scheduler TCB (responds to operator commands, creates system tasks)

c. These tasks are usually in the WAIT state, until they are made active by a request for the function they provide

13. Tasks, TCB queue

a. Every TCB in the system is on a master queue of TCB's, regardless of whether the task is for a system program, is a jobstep or subtask

b. The TCB's are enqueued in descending priority; within a group of equal priority tasks, enqueueing is FIFO

c. The origin of the queue is in the CVT

   1) loc. 10 contains the address of the CVT

   2) CVTHEAD contains address of first TCB on the queue

   3) the permanent system tasks are the first TCB's on the queue in the order indicated under topic 12. b.

d. The queuing field by which the TCB's are enqueued is distinct from the "family" queue developed

for the various tasks in a region in MVT

e. If a task changes its priority, it is moved to the corresponding position on the queue

f. When an interrupt is handled and a task is to be dispatched, the dispatcher uses this queue to locate the highest priority ready task and dispatches it

14. CPU - Main Storage - I/O Subsystem

Ref: Principles of Operation, "System Structure" (A22-6821)

a. Basic concept of a computing system has three parts which must be treated separately

b. CPU - Control Processing Unit

1) contains registers

2) fetches each instruction to be executed

3) analyses each instruction

4) executes that instruction

5) fields interrupts

c. Main storage

1) contains instructions to be executed

2) contains data the instructions reference

d. I/O Subsystem - Channels, Control Units, devices

1) Channels are limited CPU's

a) have limited instruction set

b) can access main storage independently of and simultaneously with the CPU

c) once started by CPU, can function independently of the CPU

2) Generate interrupts

3) Transmit data from secondary to main storage and vice versa

e. It is the interaction of these 3 parts which allows multitasking and gives the operating system its speed, flexibility and power

15. Instruction cycling

a. the execution of instructions is accomplished by the CPU fetching instructions, one at a time, from main storage, analyzing them and then performing the operations indicated

b. the execution of instructions is done in a cycle

1) I-Time - interpret time in which the CPU analyzes the instruction for proper format operand addresses, etc.

2) E-Time - execution time in which the operation is performed

3) pause in which an interrupt may occur, during I-Time and E-time, the CPU is not interruptable

c. It must be emphasized that main storage is just the repository for instructions - control program, the nucleus, problem programs, in short the operating system resides in main storage

d. The CPU takes one instruction at a time, sequentially and executes it, thus is the "computing" of a computer accomplished.

A. Interrupts . III A.

1. Relation of Interrupt to active task - interrupt may not be related to the active task, it may be an I/O interrupt that another, waiting, task has initiated

2. Hardware action

   a. In all cases when an interrupt occurs the hardware

      1) stores the current PSW in the fixed location appropriate to the interrupt type

      2) loads the PSW, corresponding to the interrupt type, from the appropriate fixed location

   b. The code pointed to by the new PSW handles the interrupt

3. Software action

   Ref: MVT Supervisor, Section 2 (Y28-6659)

   a. The routine given control as a result of the new PSW being loaded (known as First Level Interrupt Handler - FLIH) preserves the status of the interrupted task by

      1) storing the registers in a save area (a private save area or in the interrupted tasks TCB)

      2) moving the stored "old PSW" to a safe place (request block chained off interrupted tasks TCB)

   b. This routine then analyses the interrupt to try to handle it

c. Processing varies for the various types of interrupts

B. TCB and RB's III B.

1. Use of in interrupt processing

a. In most cases, the registers in use at time of interrupt are eventually stored in the TCB of the interrupted task

b. When this is done varies with the type of interrupt

c. The dispatcher expects the registers to be there the next time that task is dispatched

d. Request blocks chained off TCB represent levels of self-generated, interrupted program control in that task

e. Request blocks used to keep track of module(s) at various stages of execution on behalf of the task

f. Each request block has space for resume PSW for the module the request block represents

note: RB's, not TCB's, are associated with the execution of modules of code. A task is just the request for the execution of some code.

g. When dispatcher next dispatches a task, it expects to get the resume PSW from the RB pointed to from the TCB

h. RB's associated with a task are chained to each other by a link field

i. Due to type of interrupt, and linkage conventions to program to receive control, some RB's have a register save area (SVRB, IRB)

2. Creation and contents of TCB and RB's

Ref: MVT Supervisor, Section 12, TCB and RB's
Section 3 Task Creation (Y28-6659)

a. TCB created as result of ATTACH SVC

b. ATTACH routines

1) determines type of task to be created

a) jobstep - requestors prot-key (in old PSW) is 0

b) subtask - requestors prot key is not 0

2) obtains space in SQA for TCB

3) initializes the TCB

a) PRTY

b) TIOT

c) space mgmt in region

d) JOBLIB/STEPLIB DCB address

e) family TCB pointers

f) registers with parameters needed by first module to be executed on behalf of new task

g) pointer to RB for first module of the task

note: This RB is the SVRB created as result of requestors ATTACH SVC and has been dequeued from requestors TCB and enqueued off new TCB

4) enqueue TCB on main TCB queue according to priority

5) enqueue on subtask queue in proper relationship

6) determine whether mother or daughter has higher priority, set NEW pointer to point to that task

7) exit to dispatcher who dispatches either mother or daughter

note: At this point in course, this discussion of RB and contents supervision is deliberately vague, to explain it in detail at this point would be

confusing

c. The RB thus chained off new TCB contains a PSW pointing to first module to be executed on behalf of new task (it is the Contents Supervision Search routine, which searches for needed module)

d. There are 4 types of RB's in OS/360 MVT

Ref: MVT Supervisor, Section 12, PRB, SVRB, IRB, SIRB and associated sections of text (Y28-6659)

1) PRB - for problem programs, contains

   a) resume PSW

   b) pointer to control block describing the module

   c) wait count field

   d) link field

2) SVRB - for certain SVC routines, contains

   a) resume PSW

   b) indicator of which SVC is being executed

   c) register save area

   d) wait count field

   e) extended save area (misnomer, is used as parameter area between loads of the SVC routine)

   f) link field

3) IRB - Interrupt Request Block for asynchronous routine, contains

   a) address of a save area the asynchronous return can use

   b) wait count field

   c) resume PSW

d) register save area

e) link field

4) SIRB - System Interrupt Request Block for I/O error recovery routines

a) name of error routine

b) wait count field

c) resume PSW

d) register save area

e) link field

C. Task Switching routines III C.

Ref: MVT Supervisor, Section 3 "Services Internal to the Supervisor" (Y28-6659)

1. Function - to indicate, not effect, a task switch the next time the dispatcher is invoked to dispatch a task

2. When entered

a. RB wait count in a tasks top RB is cleared to 0

b. TCB non-dispatchability flag(s) cleared

3. Use of NEW/CURRENT pointers and main TCB queue

a. NEW/CURRENT pointers are a double word of pointers used to determine next task to be dispatched

b. First word of CVT points to NEW/CURRENT pointers

c. Task Switch routines passed address of newly readied task (subject task)

d. If NEW=CURRENT

1) compare subject to NEW

2) if subject priority high, set NEW to subject TCB

3) if subject priority low, no change

4) if subject priority equal to NEW

a) search TCB queue from NEW DOWN

b) if subject TCB not found, set NEW to subject TCB

c) if subject TCB found, no change

e. If NEW≠CURRENT, NEW≠0

(Task switch already indicated but dispatcher not invoked yet)

invoked yet, proceed as in d.)

f. If NEW≠CURRENT, NEW=0

(TCB queue search indicated but dispatcher not
1) compare subject priority to CURRENT

2) if subject priority high, set NEW to subject TCB

3) if subject priority low, no change

4) if subject priority equal to CURRENT

a) search TCB queue from CURRENT down

b) if subject TCB not found, set NEW to subject TCB

c) if subject TCB found, no change

4. WAIT routines are only ones to set NEW field to 0, WAIT is a type I SVC

a. if it exits before awaited event(s) completed,

1) set NEW to 0

2) invoke dispatcher

b. this is one of the few situations in which a type I SVC does not return directly to the requesting program by a LPSW.

D. Disptacher III D.

Ref: MVT Supervisor, Section 9, "Dispatching" (Y28-6659)

1. Function

   a. Selects and makes active the next task to be dispatched

   b. Completes scheduling of user asynchronous exit returns (stage 3 exit effector)

   c. Handles task and jobstep timing

   d. Handles time slicing

2. When entered

   a. Generally, after an interrupt has been processed

   b. Entered by a branch; dispatcher is a resident, non-SVC routine in the nucleus

3. Use of NEW/CURRENT pointers

   a. If NEW=CURRENT

      1) no tasks switch indicated

      2) restores registers from TCB indicated

      3) loads PSW from RB pointed to from that TCB

   b. If NEW≠CURRENT, NEW≠0

      1) task switch indicated, NEW points to next task to be dispatched

      2) makes NEW/CURRENT pointers equal to task being dispatched

      3) restores registers from NEW TCB

      4) loads PSW from RB pointed to from NEW TCB

   c. If NEW≠CURRENT, NEW=0

      1) task queue search indicated, CURRENT task in WAIT state

2) search TCB queue from CURRENT TCB down, for a ready task

   a) wait count field in top RB is 0

   b) no "non-dispatchability" flags in TCB are on

note: When such a task is found, by the organization of the queue, it is highest priority ready task

3) makes NEW/CURRENT pointers equal to TCB being dispatched

4) registers loaded from that TCB

5) PSW loaded from RB pointed to from that TCB

6) if end of TCB Queue reached and no ready task is found

   a) a special pseudo-task is "dispatched"

   b) RB is part of TCB (first word of TCB contains the address of TCB)

   c) the PSW loaded puts system in an enabled WAIT state

IV System Initialization Overview * IV

A. Initial program load IV A.

1. IPL is a hardware/software process by which the operating system is activated and initialized. It consists of

   a. Setting dials on system control panel to address the I/O device containing the IPL text, the SYS1.NUCLEUS data set, etc.

   b. Pushing the LOAD button on the control panel

2. This causes hard wired circuitry to perform the loading of the IPL program which in turn loads the nucleus

B. Nucleus Initialization IV B.

1. The IPL program eventually loads the nucleus initialization program which

   a. builds and initializes various tables and work areas in the nucleus

   b. establishes communication with the operator

   c. performs special processing on the basis of operator instructions

   d. establishes boundaries of

      1) SQA

      2) Master Scheduler Region

      3) LPA

      4) dynamic area

2. Control is then passed to the Master Scheduler Initialization Routine of the Master Scheduler

1. This routine formats various control blocks used by the master scheduler and

   a. displays automatic commands

   b. waits for SET command

   c. initializes work queues if requested to

   d. schedules execution of auto commands

   e. enters wait state

Ref: IPL/NIP PLM (Y28-6061)
IPL Appendix
IPL CSECT (from programming library)
Handout S7

A. LOAD button V A.

1. Hardware Action

   a. Hardware circuitry is set up to

      1) seek to cyl 0, track 0 of device addressed by console dials

      2) read 24 bytes (3 double words) into location 0 of main storage

   b. These 24 bytes consist of:

      1) APSW - unused in IPL of OS/360 but is used in IPL of other systems (DOS, TOS, etc.)

      2) Two CCW's - used to cause reading of IPL bootstrap record

   c. Hardware circuitry causes "execution" of first CCW which brings IPL bootstrap record into main storage at an address greater than size of IPL CSECT

   d. Second CCW is usually a TIC to IPL bootstrap record but may be installation modified to something else

2. Bootstrap Record

   a. Is then "executed" by circuitry

   b. It usually consists of a series of CCW's

   c. When "executed" bootstrap causes IPL CSECT to be brought into location 0 - the location of the

bootstrap record was high enough so as not to be overlayed by IPL CSECT

d. Last instruction of bootstrap is a simulated LPSW which causes first doubleword of IPL CSECT (which is a PSW) to be loaded by the CPU and true instruction cycling begins

3. IPL CSECT

a. Is first true software in IPL Procedure

b. Operator can cause loading of an alternate nucleus or limit storage size by doing an instruction stop at location 80 and inserting appropriate indicators at locations 8 and 9

c. Thus instruction cycling will stop after first instruction of IPL CSECT (which is a BALR 15,0 to establish addressability) allowing insertion of the indicators

note: IPL CSECT uses DC's to construct the PSW at loc. 0, and zero out main storage to location 80 , with the exception of the program check new PSW which is constructed at its required fixed location, first executable instruction is the BALR at loc. 80

4. Software Processing

a. IPL CSECT

1) clears req's 0-14 (15 used as base reg.)

2) inserts address of a program check handling routine in reg. 10

note: IPL CSECT has used DC's to construct the P/C new PSW at appropriate location in low core. It needs that PSW as it clears main storage to 0's by doing a STM until it gets a program check

3) checks loc. 8 for alternate nucleus indicator

a) if 0 - IEANUC01 is fetched

b) if non 0 - append byte 8 to standard name IEANUC0 to form name of nucleus to be loaded

note: Can have 9 different nucleui (IEANUC01-09), all must be members of PDS SYS1.NUCLEUS on pack from which system was IPL'd.

4) checks core size limit indicator

5) clears main storage to 0's

   a) uses STM rather than MVI, MVC as register to storage is faster than storage to storage instruction

   b) starts from end of IPL CSECT

   c) continues until a program check occurs or until limit indicated by loc 9 is reached

6) sets all protect keys to 0

7) searches for nucleus

   a) appends character at loc 8 to IEANUC0 if loc 8 is non zero

   b) if loc 8 is 0, searches for IEANUC01

   c) reads label of system residence device (device you IPL'd from)

   d) locates VTOC of residence device

   e) searches VTOC for SYS1.NUCLEUS data set

   f) reads scatter/translation record for selected nucleus into main storage above IPL CSECT

8) builds tables used in loading and relocating the nucleus

   a) size table - determines size of every CSECT in nucleus, uses CSECT origin (relative to origin of load module) obtained from scatter table (part of scatter/translation record)

note: Nucleus is set up to be scatter loaded - each CSECT can

be loaded into main storage in distinct locations from other CSECT's in the load module, the module does not have to occupy contiguous bytes of storage as a block loaded module does.

b) address table - for each CSECT an entry is constructed indicating where that CSECT will be in main storage. Uses info. in size table and previous entry in address table to arrive at value for each CSECT

note: It is assumed the first two CSECT's of the nucleus will be NIP and I/O Interruption handler. I/O Interruption handler is only CSECT that must reside at a fixed address in main storage - at location 0 - because it contains pre-assembled old and new PSW's that must occupy fixed positions in main storage

c) relocation factor table - for each CSECT, IPL program calculates relation factor to be used in resolving addresses in each CSECT. When Nucleus link edited, relative origins were determined and the address constants in each CSECT were filled in using that relative address. Thus it is necessary to arrive at a true relocation factor based on difference between relative and actual origin.

9) IPL program now relocates its unexecuted code and the tables just built into high core (no higher than 252K though) so nucleus can be loaded into location 0 and zeroes out the area it is vacating

10) Nucleus CSECT's are loaded

a) first CSECT is NIP and is loaded into storage just below relocated IPL code

b) second CSECT is I/O interrupt handler and must be loaded into location 0, tables have been built so this happens

c) other CSECT's loaded as encountered in load module

11) Passes control to NIP program, passing in general

registers

a) size of main storage

b) address of system residence device

c) address of size table and address table, and number of entries per table

d) address of next double word above nucleus

e) IPL branches to loc 16C, which is a LPSW from 170, the PSW at 170 contains address of first instruction of NIP

B. Nucleus Initialization Program V B.

1. Builds and initializes tables

a. CVT

1) CVT is preassembled in I/O Interrupt Handler

2) NIP puts address of CVT in location 10

3) As other control blocks and tables built, their addresses are put in CVT

4) Highest main storage address put in CVT

b. Trace Table initialization - optional

1) retrieves and rounds to 8-word boundaries the entries in the 3-word control area for the trace table

2) address of the 3-word area inserted in loc 84 (54 hex)

c. Determines size of LCS (2361 core storage)

d. Determines console readiness

1) console initialization routine locates console

a) sysgen supplied addresses of primary and alternate consoles (e.g. 009)

b) seaches UCB table for UCB with

that address

2). checks console readiness

a) if not ready, tries alternate console(s), if they are not ready - WAIT state, X'07' error code, and must re-IPL, can't just ready the console

b) if ready, continues processing

note: Bit settings in UCB indicate whether device is ready or not, but all UCB's are assembled at sys gen time as online and ready, therefore a TIO is done to verify the consoles readiness

e. Initializing Ready DA - UCB's (Dumb NIP)

1) Checks, via a TIO instruction, only the DA UCB's

2) All UCB's generated as online and ready

3) If device ready, NIP does the following

a) vol. serial of volume mounted read into UCB

b) TTR of volume VTOC placed in UCB (its in vol. label)

4) Builds table of ready DA devices for later reference

5) Smart NIP continues process

a) checks only non-DA device UCB's

b) if not ready sets bits in UCB to so indicate

f. Initializing the System Residence UCB

1) NIP checks UCB's for one with device address equal to the device you IPL'd from

note: Could have tried to IPL from a device not specified at sysgen

2) If found, UCB marked so volume mounted on it is permanently residence

3) If not found, operator message issued to mount

- SYSRES on a logically (on line) connected device and system goes into wait state and must be re-IPL'd

g. Setting up DEB's

1) Built at hi end of nucleus because multi-extent SVCLIB or multi-extent or multi-volume LINKLIB requires DEB's that vary in size and will alter size of nucleus

2) NIP builds and initializes DEB for SVCLIB

3) DEB for LOGREC is assembled in Nucleus so NIP just initializes it

4) both data sets must be on SYSRES pack

5) DSCB's read into storage, appropriate information moved into DEB's

2. Initializing the Nucleus

a. Timer - Optional

1) sets timer to 6 hours

2) loads a value (varies with model) into reg. 1 and does a one instruction BCT loop

3) checks timer, if decremented, resets to 6 hours and continues

4) if timer hasn't been decremented, issues message timer not working and continues

b. Defines Control Program Areas

1) constructs SQA in temporary location above nucleus (to allow expansion of nucleus by DEB's yet to be constructed)

2) Master Scheduler temporarily defined as all main storage from SQA to NIP

3) builds in SQA

a) dummy PQE for Master Scheduler region

b) PQE for Master Scheduler region

c) dummy PQE to be used for H0 free area

d) PQE for H0 free area, initialized later

e) DQE for SQA

f) FQE for free space in SQA

g) FBQE describing Master Scheduler region (in MS region)

h) a PQE for H1 free space

i) PQE for MS region in H1, will be initialized only if H1 LPA Is specified space is freed if H1 LPA is not built

j) FBQE describing H1 space

k) initial SVRB space is allocated at high end of SQA, address stored in transient area handler routine

note: when SVC interrupt occurs that requires an SVRB, space for one is already available, it is constructed into the SVRB needed for that SVC request and a GETMAIN issued to obtain space for the next one. This is done so the GETMAIN (an SVC) will not have to be issued while processing another SVC

c. Initialize SVC Table

1) SVC table contains entry for every SVC number, is 255 entries long

2) NIP recognizes type III and IV routines and issues BLDL for that routine name (IGC0nnnn) against SYS1.SVCLIB

3) stores TTR of routine in SVC table entry

4) if BLDL can't find an entry, console message is issued to that effect but does not halt processing

note: It is this table that IEHIOSUP utility modifies when it is executed, usually because of moving or modifying the

SYS1.SVCLIB data set.

d. Building LINKLIB DEB

1) LINKLIB need not be on sys residence device, thus NIP handles construction of its control blocks separately from SVCLIB and LOGREC, LINKLIB must be cataloged where ever it is

2) If not on residence volume (determined by check of catalog) NIP determines if volume holding LINKLIB is mounted

a) if it is, initializes the DEB

b) if it isn't, issue mount message, wait for interrupt when device becomes ready

3) if on residence volume, initialize DEB

4) UCB representing device containing LINKLIB is marked permanently resident

note: Since the SVCLIB and LINKLIB data sets may be multi-extent and LINKLIB may be multi-volume and since the size of a DEB depends on the number of extents in the associated data set, space allocations for these blocks cannot be anticipated, thus they must be built at high end of Nucleus to allow them to be whatever size is necessary. To have pre-allocated space, embedded in the nucleus, would not have allowed them to be dynamic in size.

e. Initializing the SYS1.DUMP data set

1) used to contain a dump if system failure occurs

2) NIP checks catalog for data set

a) if cataloged, checks if volume is mounted verifies data set existence and optionally formats the space

b) if not cataloged requests operator to reply with address of device to be used or to suppress the function

c) if tape - NIP verifies a non-labelled tape is mounted and unit is ready

d) NIP initializes the control blocks necessary for use of data set - ECB, DCB, DEB and IOB and places addresses in CVT

f. User options (optional)

1) if requested at sysgen time, NIP indicates model number of CPU then requests operator to "SPECIFY SYSTEM PARAMETERS" such as (selected operands)

a) additional resident modules in LPA (RAM=)

note: If delete RAM, usually have to increase MS region to contain certain access methods (e.g. BSAM) needed by system routines (e.g. log routines)

b) names in BLDL list (BLDL=)

c) additional resident SVC rtns (SVC=)

d) specify larger System Queue Space (SQS=)

e) minimum region for initiation (MIN=)

f) master scheduler region (MPS=)

g. Locating SYS1.PARMLIB

1) Contains lists used in determining

a) modules from LINKLIB to be loaded into LPA

b) modules from SVCLIB to be loaded into LPA

c) modules whose names are to be in BLDL list

2) NIP checks catalog for PARMLIB

a) if cataloged, verifies volume mounted

b) if not cataloged, assumes its on SYSRES and verifies its existence and stores track address of data set for later use

c) if PARMLIB not available, messages indicating resident options cannot be fulfilled are issued

note: Certain modules must be in LPA, this list is not in PARMLIB (just in case PARMLIB not available) but is in Nucleus. Thus those modules can always be loaded into LPA

h. Building list of data sets to be concatenated to LINKLIB

1) list of data sets obtained from PARMLIB

2) NIP tries to LOCATE each data set

3) Checks UCB's to verify necessary volumes are mounted

4) builds a DEB for each data set

5) this construction expands Nucleus, pointer updated to new end of Nucleus each time a new DEB is built

i. Initialization of Recovery Management Routines (SER)

1) checks sysgen specified option

2) loads specified SER module (SER0, SER1)

a) SER0 - not entirely resident NIP loads resident portion, locates & sets up pointer to remaining part

b) SER1 - resident, NIP loads it

j. SQA is relocated to just above nucleus, which has reached its final size

k. Time slicing initialized (optional)

1) time interval converted to times units

2) done for each time slice group

3) if cancelled by operator, skipped

l. Initializing Rollout data set (optional)

1) SYS1.ROLLOUT must be cataloged, NIP checks catalog for it, if not available, operator informed and NIP bypasses further rollout processing

2) If available, NIP checks if it is large enough to hold all of dynamic main storage, if not, scratch and try to reallocate

3) if big enough, NIP formats the data set to allow writing of main storage to predetermined locations on data set. Each location of main storage mapped to a specific location on SYS1.ROLLOUT, algorithm used to format the data set

m. Initialization of MCH and CCH (optional, available only for Mod 65 and 85

1) Pointers used by machine Check Handlers are initialized

2) Writes copies of all refreshable nucleus modules onto SYS1.ASRLIB - if ASRLIB can't be located, MCH cancelled

3) Nucleus Refresh Table (NRT) built at sysgen time, used to locate refreshable nucleus modules

n. Resetting Main Storage Boundaries

1) Expanding SQA if requested

2) NIP then relocates itself to 2K block just above SQA and reestablishes addressability

3) BLDL list built above relocated NIP, optional names being added, then moved to highest core address. BLDL issued and if requested, a list of modules is written on console

3. Link Pack area LOAD'd with

a. LINKLIB and SVCLIB modules required by control program (list internal in NIP)

b. User specified reentrant modules from LINKLIB

c. Resident SVC routine, NIP modifies SVC table entries (for type III and IV modules to indicate they are now resident and inserts main storage address of module in the SVC table.

4. Setting up Final Main Storage Divisions

a. NIP sets Master Scheduler region to 10K unless operator specified another size and sufficient storage is available

1) Master Scheduler is just below LPA

2) Control blocks built describing the region and chained out of MS TCB

b. Dynamic area set up as all storage between SQA and master Scheduler Region

1) Control blocks (D-PQE, PQE, FBQE) built to describe dynamic area

c. NIP then LINK's to Master Scheduler Initialization Rtn

C. Master Scheduler Initialization (IEEVIPL) V C.

Ref: Job Management, Part 1, (Y28-6660)
MVT Supervisor, Section 7, (Y28-6659)
IEEVIPL listing (from sys gen)

1. Console Initialization

a. Performed by LINK'ing to console initialization rtn of communication task, by master scheduler initialization routine

b. Consists of placing address of master control UCB in the Unit Control Module (UCM) - primary control table for console communications, is non-executable, contains ECB's used in console communications

Ref: MVT Supervisor, Section 12, UCM Format (Y28-6659)

c. ECB's constructed in UCM, used in indicating messages to be written, etc.

d. On return, master scheduler IPL rtn (IEEVIPL) writes "READY" on console and displays possible

AUTO commands

c. WAIT's for SET command, specifying an ECB in UCM

2. SET command

a. when issued, master scheduler IPL rtn moves master TIOT, assembled in the routine, to SQA

b. Locates PROCLIB and JOBQE data sets

1) SET command might specify units the data sets are on

2) if not, catalog is checked for them

3) if not cataloged, units specified at sys gen are tried

4) if not there, sys residence volume is checked

5) if not found there, messages to the operator

6) when located, PROCLIB is cataloged on volume it is on

c. Pointers to UCB's for PROCLIB and JOBQE are put in the master TIOT

note: no DCB's, DEB'c, etc., are built for PROCLIB nor JOBQE by these routines. RDR tasks use PROCLIB so they contain necessary control blocks and OPEN the data set JOBQE is handled entirely by Queue Management routines via XDAP and control blocks required for XDAP are in Queue Management Routines.

d. Log Initialization (optional)

1) If included in system, master scheduler IPL routine XCTL's to log initialization routine

2) System log consists of two data sets - SYS1.SYSVLOGX or SYS1.SYSVLOGY - so one can receive messages while other is being dumped to an output device

3) WTL macro and LOG command cause information to be placed in a log data set

4) Initialization Routine

a) locates data sets, they must be cataloged

b) sets up log control areas and buffers

c) creates DCB's for log data sets

d) ATTACH's log writer routine to create and initialize control blocks for the data sets

e) WAIT's on log ECB in Master Scheduler Resident data area

f) ECB posted when log data set full and activates the writer rtn to write the data set on to an output device

e. If requested in SET command, Master Scheduler IPL routine ATTACH's a task to initialize the SYS1.SYSJOBQE data set. This task runs in its own region, Master Scheduler WAIT's on its completion

3. Volume Initialization

a. Master Scheduler IPL routine ATTACH's a task to verify mounting of volumes that must be permanently resident

b. The task accesses SYS1.PARMLIB for member PRESRES and searches UCB's for volumes indicated in PRESRES and marks each UCB that holds such a volume permanently resident

c. Informs operator of status and volume serial numbers of all permanently resident and reserved.

4. Execution of AUTO Commands

a. On completion of volume initialization, AUTO commands selected by operator are displayed on console and

b. Execution of same is scheduled by invoking SVC 34 routines for each command (if SMF is to be initialized, execution of commands pends until after SMF is initialized)

5. SMF Initialization (optional)

a. On return from log initialization, system management facilities are initialized. If included in the system

b. SMF Overview

Ref: Planning for SMF, (C28-6712)

1) Routines of control program to

   a) provide history of each job as it is processed

   b) monitors jobs at various points in processing

2) Control program calls on SMF to collect data

   a) IPL

   b) reader/interpreter

   c) init/term

3) Can link to user exits to do additional monitoring or processing

4) Information is collected and written onto SMF data sets - SYS1.MANX and SYS1.MANY, such information on

   a) machine configuration, I/O devices, storage size at IPL time and when VARY commands bring a device online or offline

   b) Job and Jobstep information - accounting information, start time, CPU time (actual CPU use time) SYSIN, SYSOUT usage, how terminated recorded at Job and Jobstep Termination

   c) Counts of references to user data sets

   d) Counts of 2K blocks assigned to and released by a task

5) User Exits from

   a) Reader/Interpreter - before each JCL statement is interpreted

   b) Init/Term - when job is selected for initiation

   c) Init/Term - when step is selected for initiation

d) Init/Term - when step and/or job is terminated

e) Timer SLIH - if CPU or WAIT time limits are exceeded for a job or step

c. SMF Initialization routines are XCTL'd to by Master Scheduler IPL Rtn and they

1) add assembled DD names SMFMANX and SMFMANY into Master Scheduler TIOT

2) obtain System Management Control Area (SMCA) space in SQA, place pointer to SMCA in CVT

3) OPEN PARMLIB and read member containing SMF parameters check them for validity, requests parameters from operator if any are incorrect, parameters stored in SMCA

4) SMF data sets allocated (e.g., devices allocated) and opened, control blocks (JFCB's) for SYS1.MANX and SYS1.MANY are written into SYS1.SYSJOBQE

5) UCB's containing SMF data sets are marked permanently resident

6) on return from the allocation rtns, Initialization rtn issues ATTACH to create SMF task and pass control to SMF Writer routine who branches to SMF timer routine

7) Which sets 10 minute timer and returns

note: 10 minute timer because at least every 10 minutes SMF rtns accumulate amount of time CPU was in WAIT state

8) On return, SMF writer formats and writes SMF IPL record and XCTL's to Master Scheduler WAIT routine, Master Scheduler Initialization is complete

Ref: Handouts S19, V25-6156

A. The Work Queues VI A.

1. One data set - SYS1.SYSJOBQE, undefined DSORG

2. Entries in this data set represent work to be done by operating system

   a. Jobs to be executed by the system - input

   b. SPOOL'd SYSOUT data sets and system messages to be printed by system writers - output

3. SYS1.SYSJOBQE subdivided into total of 76 subqueues

   a. 15 input queues corresponding to the 15 input classes (CLASS operand on JOB card classes A-O)

   b. 36 output queues corresponding to the 36 output classes (SYSOUT= or MSGCLASS on JOB card, A-Z 0-9)

   c. RJE queue - contains job definitions transmitted to central computer across telecommunications lines

   d. ASB queue - contains condensed JCL images of Job definition, for faster interpretation later

   e. Hold queue - contains job entries (input) for jobs not to be executed until operator releases them (TYPRYN= HOLD or operator command)

   f. 21 unused queues

   g. One master queue indicating unused space in SYS1.SYSJOBQE

4. Physically, the first part of JOBQE contains Queue Control Records (QCR's) one for each queue, used to enqueue an entry on that queue and to dequeue it when system is ready to perform the work the entry represents and describes

5. Rest of space of data set is divided into logical tracks

   a. LTH - Logical Track Leader, serves as address and chain field for the series of records called a logical track

   b. logical track - an installation defined number of 176 byte records, for efficient usage of JOBQE space, should have some relationship to physical track

B. Process Input VI B.

1. One function of Job Management is to process input,

   a. read an "input stream" - collection of JCL, procedure references, data and commands

   b. convert JCL, proc-references to control blocks, place them on a queue

   c. generate messages about the JCL

   d. SPOOL (Simultaneous Peripheral Operation On Line) data onto DA space to be able to process JCL following it

2. There are several ways OS/360 MVT has of "reading an input stream"

   a. "ordinary" Reader/Interpreter - task that reads and interprets (scans for errors, converts to control blocks) the input stream in one task

   b. ASB (Automatic SYSIN Batching) Reader - reads and interprets as distinct tasks

   c. RJE (Remote Job Entry) Reader - handles job stream input submitted from remote work station to central computer across telecommunication lines

   d. Restart Reader - used to prepare jobs for restart that have abnormally terminated but are eligible for automatic step restart

C. Initiate/Terminate Work VI C.

1. The function of removing enqueued input work from the input queue and performing the necessary set-up and scheduling of the work and performing housekeeping when the job is finished is the function of the

Initiator/Terminator

2. Selects a job and a step of that job to be executed

3. Allocates I/O devices to the task

4. Obtains a region for the task

5. Creates the task

6. Establishes time limits for the step if requested to (TIME operands or JOB and EXEC cards)

7. Terminates the task (normally or abnormally) when necessary, frees region and devices

8. Selects next step or job to be processed

D. Process Output VI D.

1. System programs generate messages about a job as they process it, the program may generate data on a SYSOUT data set

2. Both these types of data are temporarily placed on DA space while job is being processed

3. It is a function of Job Mgmt to write these data from where there are (DA space) to where operator/ system programmer/problem programmer want them

4. This function is performed by the system output writers

5. They operate as separate tasks, in their own regions, are an advantage as these are only programs handling above data and thus avoid contention for devices where data is being written - usually unit record devices

6. Also, allow program that creates data or, to which messages apply, to execute and be terminated before SPOOL'd data is handled thus speeding thruput

E. Process Commands VI E.

1. Commands start, alter, stop system tasks, cause services to be performed, display certain actions, make system information known

2. Usually entered thru system console, but can appear in job stream

3. They are a communication from operator/programmer to system, are accepted by the communications task

4. Service, requested by command is analyzed and Master Scheduler sees to it the service is performed

5. Can create a new task to perform service, either in MS region or a new one in its own region

6. Can set bits, POST ECB's so existing tasks can perform requested service

I The Work Queues VII

Ref: Job Management, Part 6, The Work Queues,
Part 1, Initializing the Queue Data Set (Y28-6660)

A. SYS1.SYSJOBQE VII A.

1. Space allocated to the data set at sysgen time

2. Subdivided into total of 76 queues

a. 15 Input queues

b. 36 Output queues

c. Master (free) queue

d. RJE queue

e. ASB queue

f. HOLD queue

g. 21 unused queues

B. Space at physical beginning of the data set contains Queue Control Records (QCR's) one for each of the 76 queues, each QCR contains VII B.

1. A "top pointer", pointing to the next work, in that queue, to be perfcrmed (type of work - job or output - depends on type of Queue the QCR represents)

note: "Top pointer" is used to select next work (job to be executed or output to be written) to be performed. Points to oldest, highest priority work on that queue

2. Fields corresponding to priorities 0-14, pointing to work at each priority in that queue (class); work of equal priority in a class is enqueued in FIFO order

note: These fields are used only for adding new work to a queue, not for dequeueing and performing the work

3. Pointer to an ECB in main storage, ECB built by routines (initiators, writers) that perform the work that the queue entrys represent. When no work is on the queue, the routine WAIT's on the ECB, when work enqueued, the Queue Management rtns POST the ECB, thus informing the routine of work to be done

C. Rest of data set space is formatted into logical Tracks VII C.

1. Logical Track Header (LTH), used to chain several logical tracks together when such space is needed to contain control blocks describing the work

2. logical track - installation defined number of 176-byte records (specified at sysgen time, can be modified at IPL). Should be set up to maximize use of space on a physical track (no track overflow is allowed on JOBQE)

D. Space Management on SYS1.SYSJOBQE VII D.

1. Allocation of space in the data set is done by Queue Management routines which assign and free space and read and write the records on it

2. Space allocated to

   a. Reader (and Interpreter) which create control blocks that describe work and contain messages about a job

   b. Initiator/Terminators which generate messages (allocation, deallocation, etc.)

3. Space is allocated in logical tracks, such space cannot be shared between work entries

4. One logical track allocated at a time, when more space needed, another logical track is allocated from master (free) queue, from wherever there is a free logical track

5. The various logical tracks allocated to a work entry are organized and chained by the LTH's as follows

   a. For control blocks describing input (job's)

      1) all LTH's, other than first, contain pointer to the first

2) Each LTH gets a pointer to the next LTH when and if another logical track is allocated (e.g. first LTH points to 2nd, 2nd points to 3rd etc., etc.)

b. For control blocks describing output (SYSOUT data sets, messages, etc.)

1) all LTH's other than first, contain pointer to the first

2) Each LTH receives a pointer to the next LTH allocated when and if one is allocated

c. A pointer in one of the 176-byte records in the input entry points to the first LTH allocated for output control blocks

6. Until job termination, the various LTH's for input and output control blocks (and a distinct logical track is used for each distinct output class - SYSOUT= or MSGCLASS= used by the job) are treated as a logical group and the pointer in the appropriate input QCR priority field points to the last LTH allocated for input control blocks

7. At job termination time, the logical tracks containing input control blocks are freed and the logical tracks containing control blocks describing that job's output (SYSOUT data sets, messages, etc.) are split up and queued off the appropriate output QCR's according to the job's priority

note1: since the smallest unit of space that can be queued off a QCR is a logical track, the control blocks describing messages and/or data sets in different output classes must be on distinct logical tracks so the logical tracks can be separated and each queued on the appropriate output class QCR

note2: the control blocks representing a job's output to a particular class are enqueued at job termination time (as opposed to each step's output being enqueued at step termination time) because the writers handle the work as a unit and thus print all of job's output together rather than the output of each step interspersed with output of other step of other jobs

E. Enqueueing work VII E.

Ref: Job Management, Part 6, the Work Queues, (Y28-6660)

1. The top pointer in the QCR points to the last LTH which was allocated to the job to contain input control blocks. That (last) LTH points to the first LTH allocated to the job and all other LTH's for input control blocks are hung off that (first) LTH

2. The top pointer points to the highest priority, oldest enqueued job on that queue, that is, the next job that should be initiated

3. Each priority pointer in a QCR points to the last LTH allocated to a job for input records. The job it points to is the most recently enqueued job at that priority

4. If there are several jobs enqueued on a QCR at the same priority, the job's LTH's are chained by putting, in the last LTH for input for each job, a pointer to the last LTH for input for the next newest job at that priority. That is, if there are three priority 4 jobs in a queue, the top pointer points to the oldest job, that (oldest) job points to the next oldest and that job points to the last (newest) job. The Prty 4 pointer in the QCR points to the newest priority 4 job.

5. When a job is added to the queue, the priority pointer, corresponding to the new job's priority, is set to point to the new job's LTH (last LTH for input records). If there were other jobs at that priotiry, the last LTH of the job that was previously the last job enqueued (and is now next-to-the-last), is set to point to the last LTH of the newly enqueued job

F. Dequeueing Work VII F.

1. The initiator follows the top pointer to the job that is next to be enqueued from a queue

2. In that job's last LTH - the LTH the top pointer points to - is a pointer to the next job that should be initiated from the queue

3. The pointer to this job is moved to the top pointer in the QCR

note: It should be stressed that, though the above discussions tend to imply that readers, writers, or initiators enqueue or dequeue work, allocate or free logical tracks, these routines really indicate what they want done to Queue Management Routines and these routines (and only these) actually manipulate the queues.

VIII Process Input — VIII

A. Definition of Input — VIII A.

1. JCL records
2. Procedure library references
3. Operator commands
4. Data

B. "Ordinary" Reader/Interpreter — VIII B.

1. Function

   Ref: Job Management, Part 2, Part 6 - the Interpreter Rtn (Y28-6660)

   a. Read records

   b. Scan for errors

   c. Accumulate complete JCL statement (continuation cards)

   d. Construct control blocks from JCL statements

   e. Cause control blocks to be written on SYS1.SYSJOBQE data set, enqueued off appropriate input class QCR and at appropriate priority in that QCR

   f. SPOOL input stream data to temporary DA space, construct control blocks pointing to such data

   g. Intercept input stream commands and validate and schedule execution of them if they are valid

note: when reading and interpreting functions performed in the same task, the interpreter performs the reading function, not vice versa!

2. Control blocks built

Ref: Job Management, Appendix A (Y28-6660), Handout S3, S4, S13

a. JCT - Job Control Table - built by R/I

1) From information on JOB card

2) Contains, also, pointers to other control blocks built for job

a) first SCT

b) PDQ

c) first SMB

d) job ACT

e) first SCD

f) last DSB

g) DSENQ table (TTR)

3) Is not completed until next JOB card (or EOF) is encountered (needs count of steps in job)

4) Job is not enqueued (& therefore not capable of initiation) until JCT is written out

b. SCT - Step Control Table - built by R/I

1) From information on EXEC card

2) Contains pointers to other control blocks

a) next SCT

b) first SIOT

c) first SMB for next step

d) last SMB for this step

e) ACT for this step

f) VOLT

g) first DSB in message class

h) SIOT of data set referenced by PGM=*. stepname.ddname operand

3) SCT for first step of a job is chained out of JCT, each successive SCT is chained out of preceding SCT for that job

c. SIOT - step I/O Table, built by R/I

1) from device information on each DD card

2) contains

a) DDNAME

b) pointer to next SIOT

c) pointer to JFCB

d) number of units requested

e) volume count

f) SYSOUT class (if its a SYSOUT Data set)

g) pointer to next DSB

3) SIOT written in Input Logical Track space; used for device allocation purposes

4) SIOT chained out of SCT for step the DD card is associated with, other SIOT's for this step chained out of this one and each other

d. JFCB - Job File Control Block - built by R/I

1) from data set information on each DD card

2) Contains

a) data set name

b) label type

c) volume sequence number

d) data set creation, expiration dates

e) DCB information coded on DD card

f) volume serial number(s)

g) space request

3) If it is a multi-volume data set, JFCB can hold first 5 volume serial numbers, if more are specified, a JFCBX is used to contain up to 15 more volume serial numbers, JFCBX chained out of JFCB

4) Is built in input logical track space for named data sets, is built in output logical track space for SYSOUT data sets and is chained off a DSB in such space

5) In the output logical track space, the DSB's and JFCB's for the SYSOUT data sets of the same class are chained together

e. DSB - Data Set Block - space reserved by R/I, filled in by Terminator

1) Built for SYSOUT data sets, at termination time

2) Contains

a) pointer to next DSB

b) DD name

c) pointer to JFCB

d) UCB address of unit allocated to data set

e) name of program to process data set

3) Space reserved for DSB in output logical track space, is filled in by terminator providing address of JFCB for the SYSOUT data set

4) Is chained off SIOT (which is in input logical track space) built for SYSOUT data set

5) Other DSB's for SYSOUT data sets of same class as this are chained off this one in same logical track's allocated for output for this class

f. SMB - System Message Block - built by R/I or I/T as

needed

1) . Built to contain messages generated about the job as it is processed by Reader/Interpreter (JCL images, JCL messages) and Initiator/Terminator (allocation/deallocation messages)

2) JCL images placed two to a SMB

3) allocation/deallocation messages packed in

4) Resides in output logical track space

5) Chained off JCT, other SMB's chained off this and each other

g. ACT - Accounting Control Table - built by R/I

1) Built to contain accounting information on JOB or EXEC card

2) Chained off JCT or SCT depending on source of information

h. PDQ Directory Block/PDQ Block - Passed Data Queue - built by R/I

1) Built when data sets are PASS'd

2) Built in pairs, each pair contains information about three PASS'd data sets

3) Directory Block contains

a) max of 3 data set names

b) pointer to corresponding entry (for each data set) in PDQ Block

4) As many pairs as needed are built and are chained off JCT

5) PDQ Block contains, for each of the 3 data sets

a) address of JFCB for data set

b) address of SIOT for data set

c) address of UCB for data set

i. SCD System Output Class Directory - built by Interpreter

1) Built for SYSOUT classes used

2) Contains an entry for each distinct output class used by job (MSGCLASS, SYSOUT)

3) One or two (max.) SCD's built for a job chained off JCT

4) SCD entries used at step termination to create the DSB's for each SYSOUT data set

j. DSENQ - data set enqueue table - built by R/I

1) Built for each job, chained off JCT

2) Contains all non-temporary data set names used by any step of the job

3) Initiator ENQ's on this list of names before ATTACHing the first step of the JOB

4) ENQ specifies "share" or "exclusive" control depending on DISP parameter on DD card
DISP=SHR - share
DISP=OLD,NEW,PASS - exclusive

k. VOLT - Volume Table - built by R/I

1) built for each step of a job, chained off each SCT

2) Contains volume serial numbers needed for that step

3) Used during device allocation for each step, used to obtain volume information

3. Control of Reader/Interpreter

Ref: Job Management, Part 6, Interpreter Rtn (Y28-6660)
Operator Guide, Chapter 3, Operator Commands (C28-6540)

a. Started by operator command, Reader/Interpreter is a system task executing in its own region - 48K min.

b. Master Scheduler (really, System Task Control Routine) ATTACH's the Reader task, first rtn to receive control is Interpreter Reader Control Rtn which obtains space for and build

1) NEL - Interpreter Entrance list - used in communications between various routines of R/T, contains

   a) pointer to option list

   b) ECB, POST'd when STOP RDR command issued

   c) address of input stream if it is to be processed by special access method

note: Under normal conditions input stream is processed using QSAM but when it can't be so processed (when starting a system task and JCL is constructed in core, or ASB reader reads from JOBQE and can't use QSAM but must interface with Queue Mgmt, etc.) this word points to input stream

   d) optional pointer to QMPA (Queue Manager Parameter Area). If there, control blocks built will be added to queue entry associated with QMPA. If empty, each job is enqueued as a separate entry

   e) console identifier indicating to which console messages should be routed

2) Option list - indicates processing options (SMF function, track stacking, queue full condition etc.) and default values for omitted JCL operands (PRTY, REGION, TIME, jobname to start with, etc.)

3) Exit list - address of exit routines (accounting routines, input access method, queue manager rtn., SMF JCL validation rtn, etc.

c. Interpreter Initialization rtn LINK'd to and

1) Stores initializing options (from start command)

2) GETMAIN's for Interpreter Work Area (IWA) - 2048 bytes, used to build job description tables before they are written to JOBQE

3) GETMAIN's for Local Work Area (LWA) - used by JCL statement processors to do their work

4) Generates unique name base (used to generate

unique name for SYSOUT data sets)

5) OPEN's Input Stream (QSAM)

6) OPEN's PROCLIB (BPAM)

d. Interpreter Control Routine receives control

1) Reads input records

2) Determines type of record

3) Processes commands and data records

4) Passes JCL to JCL scan routine

5) Locates procedures and reads records from them

e. Scan routine

1) converts JCL to internal text

2) accumulates complete JCL statement

3) passes statement to appropriate JCL processors

f. JCL Processors

1) build appropriate tables and control blocks for the job and write them (via Queue Management Rtns) on logical tracks allocated for the job

2) create SMB's and write them, obtain space for DSB's

3) If a JCL error is detected

a) job enqueued on HOLD queue

b) internal CANCEL command executed to flush job thru system

g. Reader/Interpreter Termination

1) End of file on input stream

2) I/O error

3) Operator command

Ref: MVT Job Management, Part 2, Using the ASB Routine as a Reader (Y28-6660)

1. Function

   a. Reads JCL records

   b. Compress JCL statements

   c. Writes compressed statement on JOBQE in 176-byte records allocated to ASB queue

   d. SPOOL's Input stream data and DA space, generates replacement DD card referencing the SPOOL'd data set

   e. locates TTR of cataloged procedures, generates special record identifying and pointing to the procedure

   f. passes unrecognizable JCL statements to command processor

   g. Initiates interpretation of compressed JCL when

      1) number of jobs specified in ASB procedure has been read and placed in ASB queue

      2) input stream exhausted

      3) number of tracks in JOBQE allocated to ASB queue exhausted (number is in ASB procedure)

      4) STOP command issued for ASB Reader

      5) at least one job placed on ASB queue and can't allocate more DA space on which to SPOOL input stream data

2. Control of ASB Reader

   Ref: Job Management, Part 2, Processing Input, (Y28-6660)

   a. Started by operator command (START RDRA) executes in own region - 16K minimum - ATTACH's Interpreter when needed, interpreter executes in its own region (48K) and when finishes, region is freed

b. System Task Control Rtn ATTACH's the ASB Initialization rtn which

1) GETMAIN's space for an ASB Work Area (ASBWA) of 3200 bytes and initializes it, it contains

a) ECB to communicate between ASB Reader and Interpreter

b) ECB to STOP ASB Reader

c) Address input stream

d) Processing options - number of jobs to process before ATTACH'ing Interpreter, number of JOBQE tracks to use, etc.

e) QMPA for accessing JOBQE

f) DCB's for ASB data sets

g) work area

2) ATTACH's Interpreter Region Regulator Rtn (in LPA) which is used to ATTACH Interpreter and communicate between routines of ASB Reader/Interpreter. This time it just performs initialization

3) On return, Initialization rtn XCTL's to input stream processor

c. Input Stream Processor

1) reads input records

2) compresses JCL and writes compressed statement on space allocated to ASB Queue in JOBQE data set

3) SPOOL's input stream data and generates DD card referencing SPOOL'd data

4) for PROCLIB references, locates procedure, generates special statement containing TTR of the procedure

5) passes commands and unrecognizable JCL to command processor (by an XCTL)

6) when necessary, ASB Reader invokes Interpreter Region Regulator Rtn to ATTACH the Interpreter

7) Region Regulator ENQ's an ASB Interpreter (only 1 can execute at a time because it accesses ASB queue - a serially reusable resource, also it economizes on region space being used)

8) Region Regulator obtains region for Interpreter and ATTACH's it

d. Interpreter Control Rtn receives control and

1) constructs NEL, indicating special access method for reading JCL and procedures (special access method rtns are CSECT's in Interpreter Control Rtn)

2) Option list and exit list also constructed

3) ATTACH's Interpreter and WAIT's its completion

e. Interpreter Initialization Rtn receives control to Initialize interpreter as in VIII B. 3. c. above, then LINK's to Interpreter Control Rtn

f. Interpreter Control Rtn

1) Reads records - LINK'ing to special access methods of the ASB Queue Reader - to read and expand the compressed JCL statements in the Special Access Method Work Area (SAMWA)

2) When ASB Queue reader encounters special PROCLIB reference - it causes Interpreter to LINK to ASB FIND rtn to locate the procedure for processing

3) Interpreter interprets JCL as normal, building and enqueueing appropriate control blocks on JOBQE space allocated to the interpreter, chaining each job entry off corresponding input QCR

g. On completion of Interpreting the JCL in the ASB Queue, Interpreter returns to Interpreter Control Rtn of ASB Reader task

1) If no I/O errors nor queue full conditions, ASB Reader is reactivated to process more JCL (Interpreter region is freed)

2) If I/O error - try to recover, if cannot, ATTACH Interpreter to handle what is in ASB queue, then terminate

3) If "queue full"

a) recompress and reenqueue (on ASB queue) JCL for job being processed by interpreter

b) enter timed WAIT to allow jobs to terminate and thus free queue space

c) restart interpreter

h. ASB Reader Termination

1) end of file (input)

2) I/O error

3) STOP command

D. RJE Reader VIII D.

Ref: RJE SRL (C30-2006)
RJE PLM (Y30-2005)

1. RJE Overview

a. Serves as a sophisticated Input/Output program

b. Receives job definition across telecommunication lines from remote work stations

c. Writes job control statements on RJE queue

d. Interprets JCL statements, enqueues on appropriate input queue

e. Allows OS to initiate and execute jobs

f. On termination, sends job output back to remote work station

2. Function of RJE Reader

a. Reads records from RJE queue

b. Separates RJE control statements from OS JCL

c. Lets RJE Interpreter handle RJE control statements

d. ATTACH's OS Interpreter to process OS JCL

1) Interpreter builds usual control blocks

2) Enqueues job entries on appropriate input queues

3) Uses special access methods to "obtain" each JCL statement - they are already in core

3. Control of RJE Reader

a. Operates as a subtask of RJE Collector/Emitter

b. RJE started by operator command

c. Termination is by operator stopping RJE or no remote terminal sending input to central computer

Ref: MVT Job Management, Part 3, (Y28-6660)

A. Initiator IX A.

1. Function

a. Job selection, jobstep selection

1) Uses queue management routines to read a QCR (corresponding to an input class the Initiator is servicing) into storage

a) dequeue highest priority, longest enqueued job

b) update QCR and rewrite it

note: Queue Management rtns ENQ's on the input queues while doing this so no other initiator can (try to) access the queues while this one is checking and modifying them

c) READ JOB's JCT into main storage (Initiator's region)

2) Job Selection rtn checks job failed but in JCT, if on it processes job in "flush mode", message written on console, Queue Management moves output portion of the job queue entry to output work queue frees input queue space

3) If job failed but not on, Queue Management rtns read first SCT (pointed to from JCT) into storage

4) CSCB built when a job is dequeued for initiation, added to chain of CSCB's out of Master Scheduler Resident Data Area contains

Ref: Job Management, Appendix A, CSCB Format (Y28-6660)

a) address of initiator's TCB

b) name of the job

c) pointer to JCT for the job being processed

5) SCD read into main storage and information in it is used to build a QMPA which is used when initiator or terminator generates messages to be added to the job's message class - SMB's containing messages are added to output logical tracks via queue management rtns

6) Job selection rtns read DSENQ Table into storage and construct ENQ parameter list for non-temporary data sets used by the job

a) ENQ not issued yet, parameter list just built

b) each data set is ENQ'd on for exclusive or shared control depending on DISP operand on DD Card

(1) DISP=OLD,NEW,MOD - exclusive

(2) DISP=SHR - shared

c) ENQ issued at beginning of job and lasts duration of job because

(1) increases thruput by avoiding having to issue ENQ at beginning of each step and possible waiting to get access to the required data sets

(2) if one step of a job does some modification of a data set and successive steps perform more changes - the data set must not be used or modified by other jobs in the interval between steps of this job

d) ENQ issued from LPA just before obtaining I/O Device allocation region (first step of the job only), thus any wait for data sets does not tie up a region

b. Region Management

Ref: Job Management, Part 3, Region Management (Y28-6660)

1) Performed by initiator modules

2) Several factors are involved in deciding if a new region is needed and how big it should be

   a) user specified minimum initiator region size, specified at sysgen time, is placed in BAMINPAR field of master scheduler resident area

   b) NIP - calculated minimum jobstep region size - calculated on basis of whether or not IEFSD061 module of Initiator (job selection, termination, region size determination functions) is in LPA

      (1) if IEFSD061 in LPA, minimum jobstep region is difference between BAMINPAR and size of IEFSD061 (ca. 40K), difference stored in BAMIPAR2 field of Master Scheduler Resident Data Area

      (2) if IEFSD061 not in LPA, minimum jobstep region is equal to BAMINPAR and BAMIPAR2 is left as initialized (e.g., zeroes)

   c) User specified region on JOB or EXEC card

3) There are 3 times in processing a job when initiator may (or does) free present region and get another

   a) When initiator is WAIT'ing for work and is POST'd that work is available, a minimum initiator region (BAMINPAR) is obtained, it is in this region that the QCR(s) are read in and a job is selected to be initiated (this region is referred to as job selection region)

   b) When a job has been selected, Initiator frees present region (job selection region) and gets a new region in which to perform I/O Device allocation. The size of this region is the <u>greater</u> of

      (1) SCT specified REGION (from EXEC or JOB card)

      (2) Minimum Initiator Region (BAMINPAR)

note1: I/O Device Allocation Region can<u>not</u> be less than 52K

note2: If is after the job selection region has been freed and before I/O Device Allocation Region has been obtained for first step of a job that the ENQ on the non-temporary data sets is issued

c) When jobstep is to be ATTACH'd a new region is obtained only if the present one (I/O Device Allocation Region) is larger than necessary and if IEFSD061 is in LPA. The region obtained is the larger of:

(1) minimum jobstep region (BAMIPAR2)

(2) SCT specified region (EXEC or JOB card)

4) When initiating a system task (e.g., interpreter, writer, initiator, etc.) if the region specified for that task (on procedure JCL) is smaller than the size needed to terminate a task, the smaller region is allocated and when the task is terminated, a larger region is obtained in which to perform the termination functions

5) Just before getting the I/O Device Allocation Region, Initiator routines in LPA

a) CHAP Initiator's priority to a level similar to that of the job being processed, this is done for all steps of a job

b) issue ENQ on non-temporary data sets, if it is first step of a job

note: This is done so competition for I/O devices and, later, the jobstep region will be done at job's priority instead of the initiator's

2. Allocation of I/O Devices - assigning devices to data sets

Ref: MVT Job Management, Part 6, I/O Device Allocation (Y28-6660)

a. Allocation Interface routine entered after the allocation region is obtained, routines

1) Obtain space for step parameter list (specified in EXEC card)

2) If track stacking specified - initialize the stack, (via Stack Initialization rtns)

3) Build allocation parameter list

4) LINK to I/O device allocation rtn

b. I/O Device Allocation Routines

1) ENQ's on all UCB's

2) Obtain an SMB, for messages to be generated (allocation messages). More SMB's obtained as needed.

3) Write jobname on console (if DISPLAY JOBNAMES command has been issued)

4) Examine COND operand specified on step being processed. If comparison to completion codes in previous SCT's indicates this step should not be run, it is processed in flush mode and return code to allocation interface rtn indicates step is not to be executed

   a) all data sets, except SYSIN and OLD SYSOUT data sets, are treated as DUMMY data sets - no units assigned

   b) TIOT constructed but only entrys are for SYSIN and OLD SYSOUT data sets

note: This is done because SYSIN and OLD SYSOUT data sets have already been allocated devices (SYSIN when data SPOOL'd by R/I; OLD SYSOUT when created by previous steps)

5) If step is to be run, information is gathered from various sources and I/O allocation routine

   a) builds tables, fills them in with information from SIOT, JFCB, SCD, PDQ, etc.

   b) as allocations performed, fills in information in these control blocks (e.g., unit allocated to SYSOUT data sets, newly PASS'd data sets information put in PDQ, etc.)

   c) If DD card specifies a dedicated data set, dedication determination rtns entered

note: Dedicated data sets are allocated in initiator's cataloged procedure when initiator is started; they are available for use by any jobstep the initiator is servicing; request is by DSN = & specifying the name of the dedicated data set

6) Dedication Routines

a) check for correct JCL operands

(1) DSORG not ISAM

(2) DSN matches ddname on one of initiator's dedicated data sets

(3) SPACE parameter specifies average blocks and does not exceed space allocated to dedicated data set when initiator started

b) if reference is valid, routines

(1) force "no checkpoint" indicators in SCT

(2) force DISP=(OLD,KEEP) to avoid deletion of data set

(3) copy unit and volume info about data set into SIOT and JFCB created from the step's DD card referencing the dedicated data set

(4) updates VOLT and PDQ (so dedicated data sets can be passed to successive steps)

c) If reference not valid, control returned to process the DD card (SIOT, JFCB) as a request for a temporary data set (as "&" in front of data set name indicates)

7) If COND operands allow step to be executed, tables are built that will be used in allocation processing

a) AVT - Allocate Volume Table

(1) one entry for each DD card (i.e., data set)

(2) when unit assigned, UCB address placed in AVT entry corresponding to the data set

(3) used to create TIOT, later

b) AWT - Allocate Work Table

(1) one entry for each data set to be allocated a unit

(2) contains bit settings indicating all units available for allocation to that data set

(3) when a unit becomes ineligible for a data set, corresponding bit in that data sets entry is turned off

c) ACB - Allocation Control Block - keep track of all other tables

8) Attempt is made to equalize channel usage

Ref: Job Management, Part 6, I/O DEvice Allocation, Channel Load Assignments (Y28-6650)

a) a channel is a discrete path from a device to the CPU OR main storage; each subchannel in a multi plexor channel is treated as a separate path (channel)

b) Channel Load Table (CLT) built by allocation rtns

(1) each entry in CLT represents a logical channel and shows number of data sets being accessed thru that logical channel

note1: This count is obtained from user count field in the associated UCBs

note2: A "logical channel" for allocation purposes, is the collection of devices accessible thru or by a discrete physical path or channel. This should not be confused with the logical channel concept developed in I/O Supervisor PLM. Though the same phrase is used, the meanings are distinct

c) CLT entrys point to collections of UCB addresses in the Scheduler Lookup Table (SLT). The group of UCBs indicated by a CLT entry defines the devices accessible thru the corresponding logical channel (i.e., the logical channel represented by the CLT entry)

d) CLT entry also contains a count of data sets being accessed thru that logical channel (obtained from user count fields in corresponding UCB's)

e) allocation rtns search SLT for UCB passed to it (by other allocation rtns) and a mask is constructed with bits indicating logical channels the UCB is associated with. This mask is used with channel load information in CLT entry to equalize channel usage

note: A device can "belong to" several logical channels and a

physical channel (or subchannel) can be "in" only one logical channel according to this def'n of "logical channel". Such is not the case in the I/O Supervisor's definition of "logical channel"

9) Demand Allocation routines - allocate to data sets for which no choice of units is possible

   a) volume serial number specified is permanently resident or a reserved volume - such volumes cannot be dismounted and therefore the unit they are mounted on is the only unit that can be allocated to the data set

   b) specific unit is requested (e.g., UNIT = 282 or UNIT=SYSCP where SYSCP list has only one entry)

   c) device range reduction performed - bit settings in AWT entrys set off corresponding to ineligible devices for a data set

      (1) off line units

      (2) primary console

      (3) units holding reserved and private volumes are ineligible unless DD card specifies correct VOL=SER=

      (4) units holding system residence devices unavailable for allocation for private volume requests unless correct volume serial specified

      (5) already allocated units are ineligible except for DA Devices specifying shared requests etc.

10) Automatic Volume Recognition (AVR) Rtns entered if included in system. Allow mounting of volumes before needed or requested by system (when device comes READY, AVR puts vol serial number in UCB)

   a) AVR allocates to data sets specifying volume serial numbers

   b) UCB's scanned for vol serial numbers requested by data sets that are as yet unallocated

c) next AVR allocates on-line devices with no volumes mounted (issues MOUNT messages)

d) finally AVR Allocates units holding unneeded volumes (DISMOUNT & MOUNT messages issued)

e) if not enough online devices available - allocation recovery messages issued, AVR WAIT's on operator response

f) if required units can't be allocated, step terminated (e.g., 4 tapes requested and only 3 are generated in the system)

g) AVR rtns branch to Decision Allocation routines

11) Decision Allocation Routines entered to allocate data sets still not allocated; entered from Demand Allocation if AVR not in the system

a) allocates devices to as yet unallocated data sets requiring private volumes or specifying volume serial numbers

b) allocates devices to PASS'd data sets (volume containing PASS'd data set could have been dismounted)

c) allocates devices to data sets for which eligible units are reduced to point where no choice exists

d) attempts to satisfy channel and unit separation requests

e) units violating separation requests are made ineligible by turning off corresponding bit in AWT mask field corresponding to the data set

f) devices allocated based on "restrictiveness" of request

PASS'd	8
Channel Sep	4
Unit Affinity	2
Chan Affinity	1

thus a data set that was PASS'd and requesting channel separation is a more restrive request (& therefore allocated first) than one that was PASS'd and requesting channel affinity

12) TIOT constructed

a) one entry for each DD card

b) for data sets requiring space on public volume and for which there is a choice of units, TIOT entry contains UCB addresses of the units

c) DADSM rtns invoked to assign space for NEW data sets

d) if space is to be allocated to a data set for which there is a choice of units, DADSM works thru the list of units (if it needs to) to find one with sufficient available space to allocate

e) if space cannot be allocated - step terminated

f) if space allocated - TIOT compressed by replacing list of units with the unit that was allocated

13) Allocation messages constructed in SMB's and SMB's written to JOBQE

14) Allocation routines DEQ on UCB's and return to Allocation Interface

note: Public volume - one the system can allocate to a temporary data set when a nonspecific volume request is made and PRIVATE not coded in VOL operand

c. On return, allocation interface checks return code

1) if non zero - step cannot be run - EXEC COND codes satisfied or devices could not be allocated - alternate step delete routine entered

a) dummy TCB created and passed to regular termination routines

note: This is done to provide interface with termination

routines who "terminate" the step even though it was never run

b) SYSIN data sets deleted

c) execute accounting rtn

d) check JOB card COND codes (if met, rest of job bypassed - JCT job failed but turned on and rest of job processed in "flush mode"

e) initiators priority CHAP'd back up to original value

f) next step selected - this is done in same region that the terminator routines worked in, that is the jobstep region - the step selection routines are in IEFSD061 as are termination rtns

2) If zero, allocation interface prepares to ATTACH jobstep task

d. Flush mode processing

1) performed for

a) jobs with JCL error (JCL job failed bit is on)

b) jobs cancelled by operator (even those cancelled while still enqueued)

c) jobs whose COND operands (in JOB or EXEC cards) are satisfied

d) jobs with unrecoverable errors encountered by interpreter or initiator

2) processing

a) I/O allocation region obtained for <u>each</u> remaining step of the job

b) no device allocation performed though, data sets, except SYSIN, treated as DUMMY

c) TIOT built but only entries are for SYSIN data sets and old SYSOUT data sets

d) Device allocation rtns return non-zero return code to I/O interface rtn

e) I/O interface rtn invokes alternate step termination routine

Ref: Job Management, Part 3, Terminating the Job Step (Y28-6660)

3. Creating Jobstep task

Ref: Job Management, Part 3, Attaching the Jobstep (Y28-6660)

a. Allocation Interface routines

1) Determine priority at which task is to be ATTACH'd

Ref: Operators Guide, Chapter 3, Operator Commands, START (C28-6540)
Job Management, Part 3, Attaching the Jobstep (Y28-6660)
Supervisor and Data Management Services, Section I, Task Creation (C28-6646)

a) limit priority - specified in START INIT command, is maximum priority the initiator can assign to a task it initiates. This is the initiator's own limit priority.

b) force priority - specified in START INIT command, associated with a jobclass. All jobs from that class are ATTACH'd at the force priority, regardless of the JOB/EXEC card prioritys, but not higher than the limit priority

c) DPRTY operand on EXEC card - converted to dispatching priority (internal value) for the step

d) PRTY operand on JOB card - converted to limit priority (internal value) for the step

note: DPRTY operand can have two values: DPRTY=(V1, V2), internal priority calculated as follows: (V1 x 16) + V2; the pair (V1, V2) can range from (0,0) to (15,15); PRTY operand is a single value: PRTY=n, $0 \leq n \leq 13$, internal priority calculated as follows: (n x 16) + 11

e) Task priority determined as follows

(1) force priority used if specified and if

it is less than or equal to limit priority
if force greater than limit, limit value
is used

(2) if no force priority specified, and DPRTY is specified, the smaller of the two values: DPRTY and initiator's limit priority is used

(3) if no force priority nor DPRTY specified, and a PRTY is specified, the smaller of the two values : PRTY and initiator's limit priority is used

(4) if no force, limit, DPRTY nor PRTY values specified, the reader interpreter has inserted a default priority in JCT for job and that value is used

2) TIOT structured into 176 byte records and written onto JOBQE data set; this is done in case system fails and job can be restarted - the TIOT has thus been preserved

3) JCT and SCT rewritten into JOBQE - having been modified by initiation processing

4) if first step of job and JOBLIB or STEPLIB, data sets specified - space for DCB's obtained and opened

5) determines if allowing step to use amount of time specified (on EXEC card) would cause it to exceed JOB time limit (JOB card), sets up timer parameter specifying smaller of: difference between remaining job time and requested step time, and requested step time

6) builds ATTACH parameter list

7) passes control to routine in LPA to

a) purge track stack

b) recreates TIOT in SQA

c) CHAP's initiators priority to value similar to that at which task will be ATTACH'd (jobstep priority could be higher than present priority of initiator (if force

present priority of initiator (if force priority is used, DPRTY on EXEC card, etc.)

d) ATTACH's the jobstep

e) if necessary, re-CHAP's initiators priority to previous level

note: initiator has performed I/O device allocation at a priority similar to the job's JOB card priority which may be lower than the priority assigned to this jobstep task

f) issues STIMER for value arrived at by calculations above

g) issues WAIT on cancel or ATTACH ECB's

B. Terminator IX B.

Ref: Job Management, Part 3, Terminating Jobsteps, (Y28-6660), Handout S19

1. Function

a. Entered whenever a step terminates normally or abnormally

b. Creates interface with termination routines to

1) Dispose of data sets used by task

2) execute accounting routine

3) Check JOB COND operand

4) Release I/O devices allocated to task

5) CHAP's initiator's priority back to original level

c. Step abnormally terminated when

1) Jobstep requests abnormal termination

2) Supervisor abnormally terminates it because of an error

3) Job's CANCEL'ed

4) Steps time interval expires

d. When a job terminates

1) Queue Management rtns read DSENQ into storage, build and issue DEQ move on non-temporary data sets

2) Deletes jobs input queue entry

3) Deletes jobs CSCB from queue and frees its space

4) Enqueues job's logical tracks containing control blocks describing job's SYSOUT data sets and messages on appropriate output QCR's, at job's priority

C. Control of Initiator/Terminator IX C.

Ref: Job Management, Part 3, Initializing the Initiator (Y28-6660)
Operator's Guide, Chap. 3, Operator Commands (C28-6540)

1. Started by operator command, executes in own region of 52K minimum

2. In response to START command, Master Scheduler obtains 52K region and ATTACH's the System Task Control Task in that region

3. System Task Control rtns ATTACH the initiator task, first rtn to receive control is Initiator Initialization rtn which

a. Performs syntax check on START command parameters

1) if no parameters coded, check PARM Field of initiator cataloged procedure

2) if an error in parameters - Initiator is terminated (by System Task Control rtn)
if no errors - continue

b. Space for Linkage Control Table (LCT), two register save areas and QMPA's for message class and input queues is obtained

c. Address of track stack parameter list is put in

QMPA's

d. Buffer number put in track stack parameter area

e. LCT initialized with

1) QMPA pointer

2) UCB list pointer

3) start parameter list (SPL)

4) return address (to System Task Control Rtn)

f. Force and limit priorities are put in LCT

Ref: Job Management, Part 3, Attaching the Jobstep (Y28-6660)
Operator Guide, Chapter 3, Operator Commands, START, (C28-6540)

g. Obtains space for ECB list, one ECB for each job class the initiator is to service, ECB's initialized with complete bit "on"

h. Scans list of Group Control Blocks (GCB) for one corresponding to this initiators procedure name

1) if one exists, increases count in it by 1

2) if none exists, create it and puts it in GCB queue

note: GCB's used by MODIFY command routines

i. Passes control to job selection routine which

1) reads QCR for each input queue that this initiator is to service (QCR read via Queue Management Rtns)

2) examines "top pointer", if it points to work - begin initiating the job

3) if no work, turn off "complete bit" in ECB corresponding to that class and check other QCR's

j. If no queues the initiator is to service have work, frees the region and retreats to LPA and from there issues "WAITING FOR WORK" message and enters WAIT state, waiting on the ECB's for the input queues

note: Last word in each QCR (which the Initiator is to service) points to on ECB associated with the first initiator started to service that class. If there are several initiators servicing the same class, the corresponding ECB's, in the lists built for each initiators, are chained together. When work is enqueued on that QCR, all the ECB's are POST'd and various initiators scramble for the work

4. Stopping on Initiator/Terminator

   a. Operator command

   b. Permanent I/O error

Process Output (Output Writers) X

Ref: Job Management, Part 4, (Y28-6660)

A. Function X A.

1. Access SYS1.SYSJOBQE to find control blocks indicating data to be written

2. Uses DSB's and JFCB's created by Reader/Interpreter and Terminator describing SYSOUT data sets

3. Also uses SMB's containing JCL images and system messages

4. Activated when an ECB is posted by a terminator when it enqueues output work on a class the writer is servicing

5. If separator pages are to be written, rtn to create them is invoked

6. Dequeues on entry, determines if it is a SMB or DSB

   a. If SMB - passes control to SMB processor which extracts and writes message

   b. If DSB - passes control to standard writer rtn or user's program

   Ref: Job Management, Appendix A. CSCB format (Y28-6660)

      1) Obtains space and formats a CSCB used when operator CANCEL's OOE to cause WTR to stop processing which ever data set it is now processing but the WTR task is not terminated

      2) ATTACH's either standard writer routine or user program

      3) WAIT's subtask completion

7. Writer dynamically initializes itself to handle various input data sets

8. When a data set is processed, writer enters rtn to scratch the data set

9. When a job's output has been processed, writer enters rtn to free the logical tracks the queue entry occupied

10. When there is no more work to be done writer enters WAIT state, waiting on ECB list for classes it services

B. Control of Output Writers X B.

Ref: Job Management, Part 4 (Y28-6660)
Operator Guide, Chapter 3, Operator Commands (C28-6540)

1. Started by operator command on up to 8 output classes

2. Executes in own region of each 20K

3. Entered from System Task Control Rtns at Writer Initialization Routine which

   a. Sets up output DCB

   b. Determines if it is to use Standard QSAM, or special access method, uses special access method when

      1) output device is printer or punch

      2) data set uses machine control characters

      3) PCI is <u>not</u> used

      4) 4 or more buffers used for output

   c. Appropriate control blocks are set up if Special Access Method is to be used

   d. Validates START command operands and PARM field on EXEC statement of WTR cataloged procedure

   e. Builds a set of ECB's to be used when waiting for work, one ECB for each class the writer is to service; ECB's initialized with "complete bit" on

f. Sets up special communications ECB, used to communicate between various routines of the writer and between writer and user routines specified to handle a particular data set

g. Passes control to main logic routine to

1) Process a command (e.g., MODIFY)

2) Process an output queue entry

3) Handle a permanenet I/O error (frees workareas, and returns to system task control rtn (i.e., the writer dies))

4. Stopping a writer

a. By operator command (STOP WTR)

b. Permanent I/O error

A. Where commands may appear XI A.

1. Console - operator presses "REQUEST" on console keyboard generating I/O Interrupt

2. Input stream - Reader Interpreter or ASB reader immediately pass control to command processing routines to schedule execution of the command

B. Communication XI B.

Ref: MVT Supervisor, Section 7, (Y28-6659)
Handout S19

1. Activated by I/O Interrupt from operator's console

2. I/O Interrupt Handler, determining the interrupt came from console, passes control to resident attention routine which POST's the communication task

3. When activated, the communication task issues SVC 72 which gives control to a router module which determines the service to be performed and passes control to appropriate process modules

   a. External interruption (from INTERRUPT button on control panel, indicates main console is down and searches for alternate console, if can't find one - OS dies

   b. I/O from 1052 keyboard console

   c. Input from unit record devices

   d. Output to unit record devices

4. Appropriate processor module accepts input from console places it in a buffer and issues SVC 34 to pass control to command scheduling routines

C. Command Scheduling (SVC 34) XI C.

Ref: Job Management, Part 5 (Y28-6660)

1. Determines where command comes from (console or interpreter (JCL))

2. Translates lower case EBCDIC to upper case

3. Passes control to router module to scan buffer for command verb and:

   a. Search verb table

   b. Determine if command source has authority to issue the command

   c. If authorized - passes control to routine addressed by verb table entry

   d. If not authorized - passes control to message routine to say so

4. Command Scheduling handles commands in various ways

   a. Create CSCB - task creating commands, put CSCB on a chain (origin in Master Scheduler Resident Data Area) marks it pending and POST's Master Scheduler who scans chain for pending CSCB's and ATTACH's appropriate task to execute the command

note: Two types of task creating commands - those that execute in MS region and those that execute in their own region

   b. Update existing CSCB and POST's existing task to perform action indicated by settings in CSCB

   c. Build a Command Input Buffer (CIB) chain it off START parameter list (SPL) and notify existing task to check the CIB and take action accordingly SPL chained out of CSCB, SPL points to CIB

   d. Store command in system table and notify existing task which inspects bits in the tables during an operating cycle and begins or ceases to perform the function (e.g. DISPLAY JOBNAMES)

   e. Pass control to a routine of command scheduling routines and specifying action required via parameter in registers

f. When parameters won't fit in registers, build a parameter list and proceed as in e. above

D. Command Types XI D.

1. Task creating commands

a. Generally these are the commands which involve performance of a continuing system function (Initiation) or access a serially reuseable resource (the work queues)

b. A CSCB constructed, master scheduler POST'd to create appropriate task

c. START commands cause Master Scheduler to ATTACH system control routine in its own region (size specified at sysgen time - default of 44K or user value)

Ref: Job Management, Appendix A, CSCB format (Y28-6660)

d. System Task Control Routine (L-Shaped Program)

1) Performs syntax check on command

2) Builds JCL statements from PROCLIB member specified in command (e.g. S RDR, RDR is a cataloged procedure), and operator command operands, and generates a JOB card, these JCL images are stored in main storage; CSCB set up to point to the JOB image

3) XCTL's to Interpreter Control rtn to build NEL for interpreter (Indicating special access method since JCL is in storage)

4) LINK's to Interpreter to interpret the JCL statements and build appropriate control tables but <u>not</u> to write them on SYS1.SYSJOBQE

5) Interpreter Control, on return from LINK, passes control to allocation interface rtn of initiator to allocate devices to data sets needed by the task being started

6) Allocation Interface XCTL's to ATTACHOR rtn of System Task Control Rtn (in LPA) to ATTACH the system task in the same region occupied

by System Task Control Rtns

7) When the system task terminates, it passes control to ATTACHOR which brings in Termination rtns to perform termination functions as for any task

2. Existing task commands

   a. Involve notifying existing system task of a service or function to be performed

   b. Notification by one of methods listed in 4. b., c., d., e., f., above

Ref: Job Management, Part 1, System Restart (Y28-6660)

A. To indicate that IPL is for system restart, omit "F" operand to Q= keyword to SET command XII A.

B. Inspection of SYS1.SYSJOBQE XII B.

1. When Master Scheduler IPL rtn ATTACH's queue initialization routines, these routines do not format queue but inspect it to prepare for restart

2. Possible queue entries

a. Incomplete input entry

1) exist if system halted during input stream processing

2) entries purged, queue space freed

b. Incomplete output entry

1) exist and represent system output for jobs that have not yet been processed

2) if job has not been enqueued, free queue space reserved for output entry - it doesn't yet represent any output

3) if job enqueued, leave queue space as is, job will need it when it (job) is processed

c. Incomplete ASB entry

1) exist if system halted during ASB task

2) queue space freed

d. Incomplete RJE entry

1) exist if system halted while job entry was

being transmitted to control system

2) queue space freed

e. Enqueued input, output, hold, ASB and RJE entries

1) may exist

2) remain in queue, processed normally when system restarted

f. Dequeued input queue entry

1) exist for jobs selected for initiation but not completely processed

2) if job can be restarted, it is; if not input queue space freed, output queue entrys enqueued to be processed

g. Dequeued output queue entry

1) exist for partially completed system output processing

2) queue entry modified so unprocessed data will be processed

h. Dequeued hold queue entry

1) exist if system halted while altering status of a job (cancelling it, releasing it, etc.)

2) entries re-enqueued in hold queue

i. Dequeued ASB queue entries

1) exist for jobs processed by ASB input stream processor and are partially processed by interpreter

2) entries re-enqueued in ASB queue

j. Dequeued RJE queue entries

1) exist for jobs being processed by RJE reader (RJE Collector/Emitter accepts jobs and puts JCL images in RJE queue, RJE reader removes the entry and reads and interprets the JCL)

or processed by RJE reader

2) entries are re-enqueued on RJE queue

C. Processing of entries in queue XII C.

1. Tables constructed

   a. To keep track of each logical track in JOBQE as it is processed (Table A)

   b. To point to first entry in each queue (Table B)

   c. To find control blocks representing system output in queue entries with system output (Table C)

   d. Jobnames Table - names of jobs in incomplete input queue entries and dequeued input queue entries, used to inform operator of jobs that cannot be restarted

   e. Purge Queue - pointing to dequeued input, output, hold, ASB and RJE queue entries; used during later processing of each case

2. All entries in Table A corresponding to unused tracks are set to 0 as are entries corresponding to tracks assigned to enqueued input, output, hold, ASB and RJE entries. (These queue entires remain as is, therefore Table A entries corresponding to their queue space are "processed" by doing nothing)

3. A Purge queue constructed pointing to each

   a. dequeued input entry

   b. dequeued hold queue entry

   c. dequeued ASB and RJE entries

   d. dequeued output entry

4. Jobnames table built, contain jobname from each

   a. incomplete input queue entry

   b. dequeued input queue entry

5. Control blocks used to connect input queue entries

with corresponding system output are read into storage and checked for accuracy, updated if necessary

6. Queue space corresponding to incomplete queue entries is freed

7. Operator informed of jobnames for incomplete input entries (can't be restarted)

8. Processing of remaining queue entries

   a. Dequeued output entry - queue records read, examined, updated to indicate whether or not corresponding SPOOL'd system data set exists and entry is re-enqueued

   b. Dequeued input entry

      1) if processing after step selection but before execution - can't be restarted (TIOT lost) output queue entry enqueued, input queue entry deleted

      2) if processing stopped while a step was in execution, check if RESTART requested if yes initiate restart, if no - enqueue job's output queue entry delete input queue entry

      3) if processing stopped during termination of a step other than last - termination completed, re-enqueue input entry

      4) if processing stopped during termination of last step, complete termination, delete input queue entry

   c. Dequeued RJE queue entry

      1) reenqueued on RJE queue

      2) LTH of RJE queue entry modified if input queue entry had not been enqueued

   d. Dequeued ASB queue entries - re-enqueued on ASB queue (regular input queue entry for this ASB entry is in-complete (else this ASB queue entry wouldn't exist)) and has been freed

   e. Dequeued hold queue - re-enqueued on hold queue

9. List of ready UCB's built and DSCB's searched for data set names corresponding to system generated temporary names

   a. Output data set - increment user count field in UCB by 1

   b. Input data set - uses jobnames table to see if it corresponds to job that cannot be restarted, if it does, data set is scratched

10. This terminates the system restart work, queue initialization region is freed and control returned to master scheduler initialization IPL rtn (to execute AUTO Commands)

Ref: Handout S20, V25-6156

A. Tasks (review) XIII A.

1. Definition
   - a. Request for execution of some code
   - b. Thing to which resources are allocated
   - c. Competitor for system resources
   - d. Thing which ABEND's
   - e. Represented to system by a TCB
2. Types of Tasks
   - a. Permanent system tasks
   - b. Jobstep tasks } not mutually exclusive
   - c. Sub tasks }
   - d. Creation of each
3. Task States
   - a. Active
   - b. Ready
   - c. Wait
   - d. Dormant
4. Task queues
   - a. Main (Dispatcher) queue - origin in CVT
   - b. Subtask (family relationships) queue origin in jobstep task for the region

B. Interrupts (review) XIII B.

Ref: Handout S9

1. OS/360 is interrupt driven - active task remains active until an interrupt occurs that causes it to cease being the active task

2. Interrupt is an electric pulse

3. Originally fielded by hardware

   a. CPU circuitry determines type of interrupt

   b. Stores current PSW (in CPU) in appropriate fixed location in main storage

   c. Makes appropriate new PSW active

   d. This new PSW points to code which analyzes and handles the interrupt

C. Five types of Interrupts, each handled differently XIII C.

1. SVC - 4 types of SVC routines

2. Program Check - depends on state of task (problem program or supervisor) and whether SPIE macro has been issued

3. External - "INTERRUPT" key on control panel or timer interrupt

4. I/O - "first time" switch is checked or set and control passed to I/O Interrupt Handler

5. Machine Check - control passed to SER0, SER1, or MCH routines or system put in WAIT state

D. Task Supervisor XIII D.

1. Creates, monitors and destroys tasks, serializes use of resources when requested to do so

2. Performs ATTACH function

   a. Obtains space for and initializes TCB

   b. Enqueues TCB on appropriate queues

3. Performs DETACH function (when needed)

   a. Frees control blocks chained out of TCB

   b. Dequeues TCB and frees its space

4. Serializes use of a resource - ENQ, DEQ macros

E. Contents Supervisor XIII L.

1. Searches for requested programs

2. Builds or updates control blocks describing such programs

3. Invokes program FETCH to load and relocate the program

4. Constructs RB and inserts PSW pointing to entry point of the program, enqueues RB off requesting task

F. Main Storage Supervisor XIII F.

1. Handles all requests for main storage space, from dynamic area or within a region, or in SQS

2. Searches control blocks representing available space, tries (in most cases) to allocate space from highest possible address

3. Builds and enqueues control blocks describing space so allocated

G. Timer Supervisor * XIII G.

1. Handles requests for establishing a timer interval or expiration of such an interval

2. Builds TQE, inserts it on timer queue

3. Manipulates queue as necessary when task gets or loses control of the CPU

4. Gives control to user timer routines when time interval expires

H. Overlay Supervisor * XIII H.

1. Dynamically loads segments of a planned overlay program when needed

2. Modifies tables to indicate which segments are now in core and where they are

3. Passes control to segment loaded

I. Trace Table * XIII I.

1. Debugging aid

2. Contains entries describing interrupts and initiation of I/O

3. An entry made for all interrupts (except machine check), execution of SIO Instruction and execution of dispatcher

4. Each entry 8 words long, containing

   a. PSW At time entry is made

   b. Reg's, usually 0, 1, and 15, when entry is made

   c. Address of TCB causing entry

   d. Timer at time of entry

5. Size of table determines usefulness of table

6. Table used in cyclic fashion; when full, next entry overlays first entry in the table and space is reused

7. Table controlled by a 3 word control area addressed from location 54 (84 hex), 3 words are

   a. Current entry

   b. Beginning of table

   c. End of table

J. Termination routines XIII J.

1. Entered when SVC 3 instruction executed

2. Not to be confused with terminator routines of Initiator/Terminator

3. Delete control blocks representing programs or execution of programs

4. Decrement use/responsibility counts and when zero, delete the programs (if not in LPA), or mark them for deletion

5. When realize they have been entered from last program of a task, perform task termination functions

   a. Inform appropriate ancestor task of termination of this task and do not destroy terminating task's TCB

   b. or, delete terminating task and do not notify any other task

Ref: MVT Supervisor, Section 2, SVC Interrupt Handling
(Y28-6659)
Handout S9

A. SVC Interrupts XIV A.

1. SVC types

   a. Type 1 - resident in nucleus, runs disabled, no size limit

   b. Type 2 - resident in nucleus, runs enabled (at least part of the time), no size limit

   c. Type 3 - not resident (may be in LPA), runs enabled, less than 1024 bytes, if not in LPA executes in SVC transient area

   d. Type 4 - not resident (may be in LPA), runs enabled, larger than 1024 bytes, structured into 1024 byte XCTL segments, if not in LPA executes in SVC transient area

   e. Types 2, 3 and 4 execute under a SVRB, type one does not

2. SVC FLIH

   a. When an SVC interrupt occurs, new PSW points to SVC first level interrupt handler which

      1) saves registers in IEASCSAV in nucleus (private save area used only by SVC FLIH)

      2) checks SVC table to see if request specifies a valid SVC number (if not - ABEND)

      3) determines SVC type

         a) If type 1 - branch to it, setting reg. 14 to address of Type 1 exit routine

note: No SVRB built, OLD PSW not moved nor are registers

b. If type 2, 3, or 4 pass control to SLIH (Second Level Interrupt Handler) having loaded reg. 14 with appropriate return address - the SVC 3 instruction in the CVT

3. SVC SLIH

a. Initializes SVRB (Supervisor Request Block), contains:

1) size and type of RB

2) queue field

3) resume PSW (constructed, originally, by SLIH)

4) register save area

note: SLIH has a pointer to space already allocated in which to build SVRB, when this SVRB is initialized and enqueued it gets space for next one. This is done to avoid issuing GETMAIN SVC while processing an outstanding SVC and thus cause another SVC interrupt which would destroy first requestors PSW and registers

b. Enqueues SVRB off requestor's TCB - makes it the RB the TCB points to, the "current" RB

c. Moves PSW from low core location to the RB following this (new) SVRB on tasks chain of RB's

d. Moves registers from IEASCSAV to register save area in this SVRB

e. Locates appropriate SVC routine

1) if type 2 - its in nucleus, locate address and branch to it.

2) if type 3 or 4 and in LPA - locate address and branch to it.

3) if type 3 or 4 and not in LPA - invoke SVC transient area handling rtns to bring module into a transient area

4. SVC Transient Areas

a. 1024 byte buffers in nucleus, minimum of one pair, can request more pairs at sysgen time

note: Usefulness of more than 3 pair doubtful, processing to utilize the transient areas is sufficiently involved so system cannot use more than 3 pair efficiently

b. Transient area handling routines invoked to

1) see if required SVC already in an area

2) locate on "available" transient area

3) load required SVC routine into it and set up requesting task to execute it

c. Transient Area Handler

1) determines if required SVC Rtn is already in a transient area

a) searches Transient Area Control Table (TACT) which contains the following for each transient area

(1) address of associated transient area

(2) address of user queue (of SVRB's representing tasks sharing a block not necessarily all executing same SVC rtn)

(3) TTR of SVC rtn (or XCTL segment of a SVC Rtn) presently in the associated block

(4) addr of TCB which loads SVC rtns into associated block

(5) The TACT contains one request queue pointer - not one for each transient area - for requests that must wait for an available area

b) search is done by comparing TTR of requested SVC routine (in SVRB) to TTR of SVC routine in a block (in TACT entry corresponding to that block). If compares are equal, the

required SVC has been located.

2) if Rtn is in a Transient block

a) enqueues new SVRB to user queue for the block

note: The SVRB is now on the queue of RB's chained out of requestor's TCB and on the "user queue" associated with a SVC Transient area - each SVRB has two queueing fields

b) request for execution of SVC Rtn that the SVRB represents is satisfied on a task priority basis

c) Trans Area Handler sets up a branch instruction to point to beginning of the block, loads the registers and branches to the block to begin execution

3) If routine is <u>not</u> in a Transient block

a) Transient Area Handler checks for a block that can be overlaid by requestor (a block can be overlaid if new requestor has priority greater than any tasks now using the block or if all users of the block are not "ready")

note: A user is not ready if one or more non dispatchability bits are on it that users TCB. A block might also be free - the routine in it is not being executed by any task

b) If no area found that can be overlaid, SVRB's wait count made non-zero and it is enqueued on a request queue out of TACT, PSW in SVRB is set to point to an entry point in Trans Area Handler so when SVRB made ready and task dispatched, routines executed are same as for a new SVC request (e.g., search for a block, etc.)

c) If an area is found that can be overlaid, Trans Area Handler

(1) locates all SVRB's on user queue with a TTR equal to TTR in TACT entry for the associated block

note: For given Trans block, the user queue can contain SVRB's for several different SVC rtns; the only SVRB's that

should be put in a WAIT state at this point in time are those SVRB's representing the SVC rtn now in the block

(2) moves present WAIT count in those SVRB's to the wait count save area in the SVRB and replaces it with X'FF'

note: This is another reason for processing only the SVRB's associated with the routine in the block. If other SVRB's have already been processed as in (2), their wait counts are X'FF', if that value was moved to the wait count save area it would overlay the valid wait count in that area. If the block is not in use, there will be no SVRB's on its user queue and (1) and (2) will not be performed

(3) stores in new SVRB a PSW pointing to the transient area block (to be used by dispatcher when requesting task is eventually dispatched)

(4) sets up registers as needed by SVC rtn being located and puts them in requestors TCB

(5) puts this new SVRB in wait state pending loading of the routine

(6) enqueues SVRB on user queue for the block

(7) address of next entry in TACT saved in Trans Area Handler code - used as starting point of search for a block when next SVC request or for an XCTL request issued by a type 4 SVC rtn

note: The different modules of a type 4 SVC can execute in different transient area blocks

(8) makes task that will load the SVC rtn (a permanent system task) ready and indicates a task switch to dispatcher; address of this task is in TACT entry corresponding to the block

(9) Trans Area Handler branches to dispatcher which dispatches the Transient Area Fetch routine under control of the TCB corresponding to the block

note: There is one copy of the fetch routine, each transient area TCB points to a SVRB which contains a PSW pointing to the fetch routine. The routine is in the nucleus and reentrant

d. Transient Area Fetch Routine

1) Uses TTR of SVC Rtn and its length (in SVRB for the SVC being loaded, put there from SVC table when SVRB initialized) to load the routine and relocate it

2) If I/O errors occur, requesting task ABEND'd

3) If no errors, locates all SVRB's (in the transient areas' user queue) requesting module just loaded and resets wait count to the value in the SVRB's wait count save area

4) dequeues and makes ready all SVRB's on request queue

note: There is one request queue, SVRB's put on it (in wait condition) when their priorities do not allow them to get a block when their SVC request was originally issued

5) Transient Area Fetch puts itself in wait condition and branches to dispatcher

6) Eventually the task issuing the SVC request that started all this will be dispatched thus executing the loaded SVC rtn. In the meantime of course another task of higher priority than this requestor could cause overlaying of the block just loaded before the routine just loaded could be executed

e. Transient Area XCTL Routine

1) Entered when XCTL issued by one segment of a type 4 SVC to pass control to next segment

2) dequeues SVRB for that SVC rtn from user queue for the block and proceeds as follows:

a) searches SVRB's chained off permanent TA fetch tasks for name of requested segment. The name will be found in the SVRB if it has been loaded into a transient area and

is still there.

(1) if requested segment found, requestor's SVRB updated with data from TACT (e.g. TTR of segment, length of routine, address of transient area it is in, etc.)

(2) if requested segment not found, proceed as follows

b) search for an "available" transient area as described above in handling a new SVC request

c) if no area is available, defer the request

d) if an area is available, invoke corresponding transient area fetch task to load routine. Set resume PSW in the fetch task's RB to appropriate entry point:

(1) to perform a BLDL on segment name

(2) to bypass the BLDL, if XCTL used DE operand

f. Transient Area Refresh Routine

Ref: MVT Supervisor, Section 9, Trans. Area Refresh Routine (Y28-6659)

1) Entered when an SVC routine terminates with an SVC 3 instruction (types 2, 3, and 4)

2) Determines is an SVC rtn was overlaid in a Trans area and thus needs refreshing

3) Scans user queues for all Trans area blocks for SVRB's with TTR's in SVRB different from TTR of rtn in the block - this means the SVRB's SVC rtn had been overlaid and must be reloaded (the resume PSW in the SVRB points to the next instruction in the SVC rtn to be executed and thus when reloaded, the task can be redispatched and execution of the SVC rtn will resume)

4) The SVC rtn to be reloaded into the block is determined by the highest priority, "ready" SVRB on the user queue for a block. A user SVRB is ready if it is top RB on its TCB queue and TCB has no

non-disptachable bits set

5) Refresh routine puts all user SVRB's in wait condition moves wait count to save area, replace it with 'FF' and makes Fetch TCB associated with the block ready

6) When dispatched the Fetch task loads the required module and proceeds as in 'c.'

7) If all user SVRB's in queue for block just loaded are WAIT'ing, refresh routine removes all SVRB's from request (wait) queue, clears wait count to 0, invokes task switching rtns for each associated task to see if it is of higher priority than current task

8) When finished with this processing, branches to dispatcher

5. SVC Exits

a. Type 1 SVC exit

Ref: MVT Supervisor, Section 9 (Y28-6659)

1) Taken by type 1 SVC's when requesting routine is <u>not</u> in a WAIT state

2) Registers are restored from IEASCSAV

3) PSW loaded from low core location where it was stored when interrupt occurred

4) Dispatcher is <u>not</u> invoked

5) WAIT SVC rtns are type 1, if wait count in current RB is non-zero by the time the WAIT rtn finish processing

a) Requesting rtn cannot be resumed (its still WAIT'ing)

b) PSW Moved into current RB of requesting task

c) Registers moved into requesting TCB

d) NEW TCB Pointer set to zero

e) Dispatcher invoked

b. All other SVC exits

1) Moves registers from exiting SVRB to TCB

2) Stores reg's 0, 1, 15 in TCB as exiting SVC left them (parameters to requesting program, etc.)

3) Dequeues SVRB from task's RB Queue

4) If exiting SVC is type 3 or 4, removes SVRB from user queue associated with a transient area

5) checks for EOT - if so, branches to EOT rtns

6) checks if next RB (now, the top RB) is WAIT'ing if no, insures that a task switch is indicated

7) Frees the SVRB

8) Branches to transient area refresh routine

B. Program Check Interrupts XIV B.

Ref: Supervisor and Data Management Services, Section I, Program Management Services (C28-6646)
Supervisor and Data Management Macros, Section II, SPIE Macro (C28-6647)
MVT Supervisor, Section 2, Program Interruptions, (Y28-6659)
Handout S9

1. SPIE Facility

a. Allows user programs to indicate a routine to receive control in event of certain program checks, also specified by user

b. User codes SPIE macro indicating the routine to be given control and for which program checks

c. SPIE SVC creates a

1) PIE - Program Interrupt Element - constructed only on first issuance of SPIE macro in a task, address put in TCB of task, PIE contains

a) address of current PICA

b) save area for P/C old PSW

c) register save area for reg's 14-2

2) PICA - Program Interrupt Control Area - constructed for each SPIE issued in a task contains

a) mask field used in task's PSW (to allow program checks normally masked off if user rtn is to handle such interrupts)

b) address of user rtn

c) mask field indicating which program checks the user routine is to handle

d. When a program check occurs that the user rtn is to handle

1) High order bit of PIE set to 1

2) P/C old PSW moved into save area in PIE

3) Registers 14-2 moved into reg save area in PIE

4) Problem program registers restored and control passed to user routine

2. When program check occurs, PC FLIH receives control and

a. Saves registers in private save area

b. Checks state of offending task

1) if supervisor state ABEND

2) if problem program state

c. Checks if SPIE has been issued (presence of PIE pointer in TCB)

1) if not - ABEND

2) if so - check

d. If already in SPIE routine (high order bit of PIE set to 1?)

   1) if set to 1 - ABEND

   2) if set to 0 -

e. Checks if user has indicated he can handle the program check that has occurred (mask field in PICA)

   1) if user cannot handle it - ABEND

   2) if user can handle it -

f. PSW moved from low core, where it was saved as result of interrupt, to save area in PIE

g. Registers 14-2 moved from private save area to save area in PIE

h. Registers restored from private save area

i. Reg 14 set to point to an SVC3 instruction located just before entry point of PC FLIH

j. PSW modified to point to user routine and is loaded thus passing control to user rtn

k. User rtn can modify PC old PSW at will but only changes to right half of it will have any effect

l. User rtn should terminate with a BR 14 for two reasons

   1) exit rtns set high order bit of PIE back to 0

   2) cause right half of PC old PSW be made right half of tasks resume PSW

3. User Program Check Routine Exit

Ref: MVT Supervisor, Section 9, Return from User Program Check Routine (Y28-6659)

a. SVC 3 instruction, executed as result of BR 14, causes updated PSW to be stored and SVC new PSW made current

b. First function of SVC 3 rtn (regardless of how entered, they have no way of knowing yet) is

to compare instruction address in stored SVC old PSW to entry point of PC FLIH

1) if equal, SVC 3 rtns were entered from user PC rtn and SVC rtns perform special processing

2) if unequal - SVC 3 rtns were entered from a routine other than a user PC rtn and do not perform special processing

c. Special processing when SVC 3 rtns entered from user PC rtn

1) set high order bit of PIE to 0

2) form composite PSW from

a) left half of PSW stored as result of SVC 3 interrupt

b) right half of PSW in PIE (possibly modified by user rtn)

3) move this composite PSW into the current RB chained off the task that issued SVC3 (SVC3 rtns are type 1 so there is no SVRB for them)

note: This composite PSW is formed so user can modify instruction address portion but cannot change protection key or supervisor/problem state bit of PSW

4) move registers from SVC FLIH private save area to TCB then overlay registers 14-2 in TCB with corresponding data in PIE

4. Abnormal Termination by Prog. Check FLIH

Ref: MVT Supervisor, Section 10, Abnormal Termination (ABTERM Prologue), (Y28-6659)

a. PC FLIH Runs uninterruptable, cannot issue SVC13 (or anyother SVC for that matter)

b. Must cause the offending task to issue its own SVC 13

c. Does this by entering ABTERM Prologue which

1) Refreshes loc 16 hex to point to CVT (just in case it was overlaid by the error)

2) Obtains address of TCB to be terminated from CURRENT TCB Pointer

3) Saves the offending PSW in OPSW field (resume PSW field) of the current RB of the task

4) Moves registers from PC save area in low core to register save area in TCB to be terminated

5) Sets up completion code, it will be stored in the TCB by ABTERM rtns (passed in a register)

6) Sets up return address to be used by ABTERM (return is to the dispatcher)

7) Turn on the dump flag (passed in completion code register) to request a dump

8) Branches to the ABTERM routines

note: ABTERM rtns is a disabled, serially reusable, non-SVC program resident in the nucleus

d. ABTERM rtn

1) Refreshes CVT address (again!)

2) Saves right half of resume PSW (in current RB of task being terminated) in RBABOPSW field of the RB

3) Stores completion code and dump flags in TCB

4) Modifies resume PSW to point to an SVC 13 instruction in the CVT

5) Sets non-dispatchable any incomplete subtasks of the task being terminated

6) Returns to address specified by caller - in this case the dispatcher who dispatches the task to be terminated. The first instruction executed on behalf of this task is the SVC 13

Ref: MVT Supervisor, Section 2, External Interruption, (Y28-6659)
Handout S9

1. External FLIH

a. Receives control when external new PSW made current by interrupt circuitry

b. Saves registers in CURRENT TCB

c. Saves PSW in current RB chained off that TCB

d. Determines cause of interrupt

1) "INTERRUPT" key pressed on control panel

a) means master console is inoperative

b) resident external routine given control

c) it searches for an alternate console(s)

d) if one found messages issued to request designation of a new master console by operator command (such a command can be issued from any alternate console)

e) if one not found, OS dies

2) Timer interval expired

a) Timer SLIH given control

b) checks TQE that has expired, takes action according to macro operands indicated in the TQE

(1) terminates task

(2) clears wait count

(3) reconstructs TQE into IRB, IQE, invokes Stage 2 exit affector to schedule user exit routine

c) Exits to FLIH

e. Branches to dispatcher

D. I/O Interrupts XIV D.

Ref: MVT Supervisor, Section 2, I/O Interruptions (Y28-6659)
Handout S9

1. I/O FLIH

a. Receives control when I/O new PSW made active by circuitry

b. Checks switch to see if another I/O Interrupt was being handled when this one was allowed

1) If bit off (no other I/O Interrupt being handled)

a) saves registers in CURRENT TCB

b) saves PSW in current RB of that TCB

c) passes control to I/O Interrupt handler

2) If bit on (another I/O interrupt was being handled)

a) registers nor PSW saved

b) passes control to I/O Interrupt Handler

note: I/O Interrupt Handler is reusable, when "first" entered, CURRENT task's registers and PSW saved and interrupt handled; while still in control, I/O Interrupt Handler enables I/O Interrupts (to handle stacked interrupts) and if there are stacked interrupts (stacking done in hardware) processes them. When there are no more I/O Interrupts to handle, the "first time switch" is set off and exits to Dispatcher, next I/O Interrupt causes switch to be checked and registers and PSW of them CURRENT task are saved.

On processing stacked interrupts, registers and PSW are not saved as they are not the registers nor PSW of the task that was CURRENT when original I/O Interrupt occurred yet I/O Interrupt Handler is executing under that TCB. Thus if it saved

registers and PSW on recycle for a stacked interrupt it would destroy the registers and PSW of the task that will eventually be redispatched

E. Machine Check Interrupts XIV E.

Ref: MVT Supervisor, Section 2, Machine Interruption, (Y28-6659)
Handout S9

1. Machine Cneck PSW points to recovery routines selected at sys gen time

2. SER0 (System Environment Recording)

   a. Least complex of the options, is not entirely resident, MC new PSW points to a resident routine that loads the rest of the module into storage

   b. Operates without OS/360 facilities and is non reusable

   c. Is given control as result of a machine check or a channel check (in latter case, I/O Interrupt Handler loads the MC new PSW

   d. Functions as follows

      1) Halts I/O on all devices

      2) Reads first 1024 bytes of non-resident module into storage, if can't do this in 10 retrys - ring bell and WAIT

      3) Non-resident module loads rest of itself into storage

      4) Determines whether machine check or channel check has occurred (loc 50 is initially X'FF', a machine check overlays this, channel check does not)

      5) Checks registers for valid parity

      6) Checks busy bit in UCB to see how many UCB's were busy when check occurred

      7) Collects information to be written to

SYS1.LOGREC, and writes it

8) If more checks occur while error data is being accumulated (3 of them) error message is printed and SER0 stops

9) In any case, SER0 tries to write a message to console indicating extent and outcome of its data accumulation

10) Loads a waiting PSW

3. SER1

a. More complex than SER0, collects and writes error data, tries to associate error with a task and terminate that task. Is resident and is serially reusable

b. Uses OS/360 facilities

c. Receives control as result of machine check or channel check causing machine check new PSW to be loaded

d. Collects data and writes it to SYS1.LOGREC using EXCP unless control program was damaged in which case it uses its own I/O routines

e. Tries to associate the error with a task and if it can, invokes ABTERM to terminate that task so system can resume execution

note: no task from master scheduler up (on main TCB queue) will be ABEND'd this way. If error is associated with one of these tasks, system put in WAIT state.

f. If successive machine checks occurred during data collection or error could not be associated with a task or the control program was damaged by an error or the error record could not be written, a WAIT PSW is loaded

4. MCH (valid option only for Model Mod 65 MP or Mod 85)

a. Most sophisticated error recovery routine, is partially resident, other modules on SVCLIB, attempts to recover from a machine check

b. Given control as result of machine check interrupt causing machine check new PSW to be loaded

c. Tries to retry the instruction being executed when failure occurred

d. If instruction retry not possible, tries to repair program damage

1) Defective storage protect feature - repaired by SSK (set storage key) instruction

2) Defective storage location - repaired by reloading (refreshing) module at that location (checks nucleus refresh table - NRT - and tries to refresh module from copy on SYS1.ASRLIB)

e. If program repair is possible, MCH retries interrupted instruction

1) if retry successful, MCH has corrected the error

2) if retry or repair unsuccessful, MCH can continue partial system operation or put CPU in WAIT state

f. Partial system operation attempted if a problem program task was interrupted and damage not too extensive, abnormally terminates jobstep task and tries to continue

g. CPU put in wait state if system task was interrupted or damage extensive

h. In M65MP - Storage Reconfiguration rtn schedules task for selective ABEND and logically removes failing storage from system. System operation resumed

F. Asynchronous Exit Routines * XIV F.

Ref: MVT Supervisor, Section 3, Scheduling a User Exit Routine (Y29-6659)

1. Definition - routines, usually user written, to be given control when certain events occur that cannot be anticipated by the system nor sysnchronized with other system processing

2. How specify

a. ATTACH macro, ETXR operand - routine of parent

task to be given control when the ATTACH'd task terminates

b. STIMER macro, user routine - routine of the requesting task to be given control when specified time interval expires

3. Scheduling of Asynchronous Routines

a. ATTACH, ETXR rtn

1) ATTACH SVC rtns detect ETXR operand

2) Branch to Stage 1 Exit Effector to

a) obtain space for an IQE (Interrupt Queue Element)

b) obtain space for an IRB (Interrupt Request Block) if one does not already exist for the routine

note: Mother task could ATTACH several daughters, specifying same ETXR rtn for each. In each ATTACH, a new IQE would be built but the same IRB would be used for all requests. The IQE is associated with the request, the IRB is associated with the routine

c) Initialize IRB, includes address of exit routine

d) Take type 1 exit to caller (i.e. the ATTACH routines)

3) On return ATTACH rtns initialize IQE and chain it of newly created TCB, IQE points to IRB at this time

4) When this (new) task terminates, termination routines detect the IQE pointer in the TCB being terminated and link to Stage 2 Exit effector to

a) enqueue the IQE on the AEQJ (Asynchronous Exit Queue)

b) set "stage 3" switch and return to caller

5) When dispatcher next entered, it checks "stage 3" switch and finding it set, enters Stage 3 Exit Effector to

a) scan AEQJ for IQE's

b) checks IRB's chained off IQE's

(1) if IRB is already queued off a task - chain IQE off the IRB to indicate reuse of the routine the IRB represents

(2) if IRB not queued off a TCB, enqueue it off the TCB pointed to by the IQE and initialize it

(a) set up PSW to point to entry point of user rtn

(b) move registers now in the TCB to the IRB register save area

(c) set up registers in TCB for entry to user rtn

c) Stage 3 Exit Effector returns to the dispatcher after all IQE's are processed

note: At this time the IRB enqueued off a TCB (the task that issued the ATTACH specifying the ETXR rtn) has at least one IQE (representing at least one request for the associated rtn) chained off it. If several ATTACH's were issued specifying this rtn as an ETXR rtn, and those subtasks have terminated while the rtn was executing for the first such subtask's termination, the IQE's representing the other requests are chained off the first IQE hung out of the IRB

4. Exit

Ref: MVT Supervisor, Section 9, (Y28-6659)

a. When user ETXR rtn terminates with a BR 14 (and therefore an SVC 3 instruction), the termination rtns

1) Dequeue the IQE pointed to from the IRB

2) Check if that IQE points to another IQE (representing a second request for the

same rtn)

a) if no other IQE pointed to - free the one IQE, dequeue and free the IRB (moving registers in IRB to TCB)

b) if it does point to another IQE, free first IQE, re-set IRB to point to next IQE, reset PSW in IRB to point to entry point of user ETXR rtn, re-set registers for user rtn in TCB

b. STIMER, user routine

1) STIMER SVC Rtns build TQE, put timer value in it

2) Enqueue TQE in timer queue at position corresponding to the timer interval

3) When time interval expires, TQE reconstructed into IRB and IQE

4) Stage 2 Exit affector invoked, processing continue as in 4. a. above

G. Exiting Procedures (Review, for the most part) XIV G.

Ref: MVT Supervisor, Section 9 (Y28-6659)
Handout S11

1. Exit from a type 1 SVC

a. Usual exit

1) Restore registers (LM instr) from SVC FLIH private save area

2) Load PSW (LPSW) from SVC old PSW location

3) Requesting task thus "dispatched" by the SVC rtn itself, without using dispatcher rtns

b. Special exit

1) Taken by WAIT - routines (type 1 SVC) when

requesting task's wait count not zero when WAIT SVC rtns have finished processing (would be ridiculous to take usual type 1 exit - that would cause a waiting task to be dispatched)

2) Move PSW to current RB for requesting task

3) Move registers from SVC FLIH save area to TCB

4) Set NEW pointer to 0

5) Branch to dispatcher

2. Exit from User Program Check Routine

a. Exit routines (SVC 3 routines, type 1 SVC) entered by SVC 3 instruction placed just before entry point of Prog. Check FLIH

b. Exit routines compare address in PSW stored as result of SVC 3 interrupt to entry point of PC FLIH - equal compare indicates how they were entered and causes special processing

c. High order bit of PIE set to 0

d. Registers in SVC FLIH save area (put there as result of SVC 3 interrupt) moved to TCB of task that issued SVC 3

e. Composite PSW built, in current RB for that TCB, from

1) Left half of PSW in SVC old PSW - stored as result of SVC 3 interrupt

2) Right half of PSW in PIE - user modified

note: This allows user to modify instruction address in PSW but <u>not</u> the storage protect key nor supervisor mode bit

f. Branch to the dispatcher

3. Exit from a routine controlled by an SVRB

a. Exit routines entered by SVC 3 instruction (SVC 3 in CVT)

b. Exit routines determine that an SVRB is current RB for requesting task

c. Subroutines invoked to

1) Move registers 2-14 from SVRB to TCB

2) Set up registers 0, 1, and 15 in TCB as terminating routine indicates (parameters to caller)

3) If terminating SVC is transient, SVRB dequeued from transient area queue

d. Return to exit common processing (6. below)

4. Exit from routine controlled by a PRB

a. Exit routines entered by SVC 3 instruction in CVT

b. Exit routines determine PRB is current RB on requesting TCB

c. Subroutines invoked to

1) Move registers from SVC FLIH save area to requesting tasks TCB

2) Check if this RB is last on TCB queue

a) if so - branch to end of task routines

b) if not

3) Branch to contents directory rtn to

a) Locate terminating programs' CDE and reduce the use/responsibility count by one

b) If program is serially reusable and there is at least one outstanding request for it (use/resp count not yet 0, also PGMQ field of PRB for terminating program points to next requestor)

(1) Addr. in CDE updated to point to next PRB (from PGMQ field of terminating PRB)

(2) This next PRB 's wait count cleared to 0

(3) Sets PSW in new PRB to point to entry point of requested program

(4) Branch to task switching routines (new RB's task may exceed others in prty)

(5) Task switching rtns return to exit routines which perform common processing (6. below)

c) If there are no outstanding requests for terminating program

(1) Non-functional flag set in CDE for program

(2) Program attributes checked and processed accordingly

(3) If in LPA - no processing, return to exit common processing

(4) If not in LPA and either reusable or re-entrant set purge flag in CDE, which will be tested by GETMAIN when space needed

(5) If neither reusable nor reentrant - storage used by program freed as are its control blocks (CDE, XL)

(6) Return to exit routine common processing

5. Exit from program controlled by IRB or SIRB

a. Exit routines entered from SVC 3 in CVT

b. Exit routines determine IRB or SIRB is current RB on requesting TCB

c. Subroutines invoked to

1) For SIRB - immediate return to caller, no processing

2) For IRB - checks use count in IRB for zero - if not zero parent task has requested multiple use of the exiting routine

3) Dequeues top IQE from IRB and frees it

4) Resets IRB (PSW) for reexecution of user rtn

5) If use count 0 - dequeue IRB, move registers in it to TCB and free IRB

6) Branch to transient area refresh routines

6. Exit common processing

a. Routines of Exit SVC entered after special processing for different RB's has been performed

b. Check if exiting rtn is under control of last RB on its TCB queue - if yes TCB is dequeued and "normal termination" flag is set in it

c. If not last RB, check is made to see if next RB is in Wait state, if it is - test NEW/CURRENT pointers to see if task switch has been indicated, if not, sets NEW to 0; if already has been indicated, no change

d. Dequeues terminating program's RB and frees the RB if it can (cannot free SIRB)

e. Branches to transient area refresh routine - which processes as described in XII A. 4. f.

H. Task Switching Routines (Review) XIV H.

Ref: MVT Supervisor, Section 3, Services Internal to the Supervisor (Y28-6659)

1. When entered and why

2. if NEW=CURRENT

3. if NEW≠CURRENT, NEW≠0

4. if NEW≠CURRENT, NEW=0

I Dispatcher (Review)

Ref: MVT Supervisor, Section 9, Dispatching (Y28-6659)

1. When entered and how
2. Use of NEW/CURRENT pointer
3. Search of TCB queue

Task Supervisor XV

Ref: MVT Supervisor, Section 3 (Y28-6659)

A. Function Overview XV A.

1. Creates tasks as result of ATTACH macro

2. Detaches and deletes tasks on request (DETACH)

3. Serializes use of resources ENQ/DEQ

B. Task Creation (ATTACH) XV B.

Ref: Handout S15

1. ATTACH macro expands into an SVC call

2. ATTACH SVC routines

   a. Obtain space for TCB (also IQE and IRB if necessary)

   b. Initialize TCB with

      1) Priorities (limit and dispatching)

      2) Subpool pointers and subpools

      3) TIOT Pointer (from parent TCB - including initiator)

      4) IQE pointer (if ETXR specified)

      5) ECB address (if ECB specified)

      6) Subtask pointers

      7) Save area address

   c. Determine if new task is jobstep or not by PSW protect key of requestor (requestors PSW is in SVC old PSW)

1) If protect key is 0 - new task is jobstep

2) If protect key is non-0 - new task is subtask

d. New TCB enqueued on TCB queue (dispatcher queue) according to dispatching priority

e. Registers in SVRB for ATTACH routines moved into requestors TCB, reg 1 set to point to new TCB, RB pointer set of next RB on requestors queue

f. SVRB for ATTACH routines is now chained off new TCB, module name specified in ATTACH put in SVRB

g. PSW in SVRB set to point to Contents Supervisor rtns to locate requested module

h. NEW TCB pointer set to higher priority task of the requestor or its new subtask

i. Dispatcher branched to

C. Task Termination (DETACH) XV C.

1. When needed

a. When ATTACH That caused subtask creation specified either or both of ECB, ETXR operands

b. TCB's for subtasks so created will not be deleted unless specifically DETACH'd

2. Freeing TCB

a. All other control blocks associated with the subtask were removed and/or freed when subtask invoked exit routines (SVC 3) and it was determined that a task (rather than a program) was terminating

3. TCB is not freed automatically if ECB and/or ETXR operands coded in ATTACH as these operands indicate parent task wants to be notified of subtasks completion and do some checking on its completion

a. If subtask ATTACH'd with neither ECB nor ETXR operands its termination will cause its TCB to be freed and parent will not be notified of subtask's completion (normal or abnormal)

b. If DETACH issued for a subtask that is incomplete, it is abnormally terminated

c. If a DETACH is issued for a TCB not on requestor's subtask queue, requestor is ABEND'd

D. Serializing Use of a Resource (ENQ/DEQ) XV D.

1. Purpose

a. To allow programs to request use of a serially reusable resource

b. To make their request and eventual use of that resource known to other requestors

c. And thus prevent two requestors from using the same resource simultaneously

d. Requires that a common name be assigned to the resource and known and used by all requestors

note: the specification of the name and consistant use of it is a matter of programmer communication. The ENQ/DEQ routines do not know what resource a name represents nor do they prevent use of that resource if a programmer fails to use an ENQ.

2. How request is issued and handled

a. ENQ macro specifies resource name (Qname and Rname) and exclusive or shared control of resource, also whether it is a system wide or region wide resource

note: Might request exclusive control of a data set if your task is going to modify it. When your program gets the resource, you are the only task using it. Request shared control of the data set if you are not going to modify it but want the fact that you are using it known so no other task (requesting exclusive control) can change it while you are using it

b. ENQ generates SVC, ENQ routines construct control blocks

1) new qname and rname

a) major QCB (Queue Control Blcok) built for qname

b) minor QCB built for rname

c) QEL (Queue Element) built to represent requesting task, chained out of minor QCB

(1) if it's a region wide resource (known only to tasks in requestor's region), region protects key is put in QEL

(2) if its a system wide resource (known to all tasks in the system) a 'FF' is put in the QEL in place of protect key

2) qname already used by this requestor or another requestor, but new rname

a) no major QCB built, one already exists

b) minor QCB built for new rname, queued off other minor QCB already queued off major QCB

c) QEL built representing requestor as above chained off newly built minor QCB

3) Previously used qname and rname

a) no major or minor QCB's built, they already exist

b) QEL built to represent requestor, chained off other QEL's off minor QCB

c. QEL's represent tasks requesting use of the resource named by qname, rname combination

d. Tasks represented by QEL's for a particular resource are allowed to use the resource in a strictly FIFO manner, depending on position of their QEL's in the queue

e. If and when requesting task is top QEL in

queue, that task is allowed to continue or resume execution at instruction after ENQ macro and thus access the requested resource. If task's QEL is not top on queue, it is put in WAIT state

f. Exception to e. above

1) can issue ENQ for test purposes to see if requestor would get immediate use of the resource or to see if the task was already enqueued on the resource (attempting to ENQ on a resource presently ENQ'd on results in ABEND of requestor)

2) If two tasks requesting shared control of some resource are adjacent at top of queue, both tasks are allowed to resume executing together

note: The system does not know and has no way of knowing what resource is represented by a given qname, rname pair and thus the use of a resource is controlled by holding requestor in a WAIT state or allowing it to execute its own code to access the resource.

g. The QEL's chained off a particular minor QCB are manipulated as if they constituted several distinct lists

1) the QEL's with 'FF' representing system wide resources are handled as one list

2) the QEL's with like protect keys are handled as separate lists, a distinct list for each distinct protect key. Since such QEL's represent requests to use a resource known only to tasks in the region with that protect key, the QEL's with like protect keys are handled as separate FIFO lists

I Contents Supervisor XVI

Ref: MVT Supervisor, Section 4, (Y28-6659)
Handout S12

A. Function Overview XVI A.

1. Searches for modules requested to be brought into main storage

2. Loads and relocates the module if necessary

3. Builds control blocks describing the module

4. Passes control to the module or returns address of its entry point to requestor

B. Macros invoking Contents Supervisor via an SVC XVI B.

Ref: Supervisor and Data Management Macros (C28-6647)

1. ATTACH - requests creation of a new task and specifies first program to be located and executed on behalf of the new task

2. LINK - requests that the specified program be located and executed on behalf of requesting task before instruction after the LINK (in requesting program) is executed

3. XCTL - requests the specified program be located and executed and its PRB replace requesting program's PRB in chain hung off requesting task's TCB

4. LOAD - requests the specified program be located or loaded and its entry point address be returned to requestor but the requested program is not to be executed automatically by Contents Supervisor

C. Search for the module XVI C.

1. Requestor's region

a. Modules loaded into a requestors region constitute

the region's job pack area

b. Each module is represented by a CDE (Contents Directory Entry) containing name of module, etc.

c. CDE's queued together, origin of jobpack queue is a pointer in jobstep TCB

d. Contents supervisor searches this queue first for the requested module

1) If it is found in the region

a) Check if module available - it must be

(1) reentrant, or

(2) reusable and not in use, or

(3) nonreusable and unused

b) if unavailable - defer request - chain requestors SVRB (for Contents Supervisor rtn) off RB presently controlling the Module (or last RB of a chain of such deferred requestors)

c) If module available - increment use/ responsiblility count in CDE by 1

d) Build and enqueue a PRB for the module, PRB inserted on requestors TCB queue of RB's after the SVRB for contents supervisor (that is, the PRB is the 2nd RB in that tasks RB queue) PSW in PRB built to point to entry point of module

e) Contents superv rtns exit (SVC 3) thus deleting the SVRB and making new PRB the top RB on the task's queue

2) If module not found in requestor's region

2. Contents Supervisor Checks for private libraries

a. If DCB operand of Macro specifies a DCB address, the data set so specified is searched

1) if module is located

a) space for a CDE is obtained and partially filled in

b) program FETCH invoked to load and relocate the program

(1) issues GETMAIN for area the size of program being loaded

(2) as program loaded, builds extent list (XL) indicating load point and size of each segment of the module

note: For block loaded modules (the usual case) only one entry is made in the XL but for the scatter loaded modules a different entry would be made for each CSECT indicating its load point (e.g., address of first byte of the CSECT) and size

(3) returns to caller the entry point address of the module

c) if LINK, ATTACH or XCTL being processed, since module is now available, a PRB is built and enqueued after SVRB for contents supervisor rtn, PSW built to point to modules entry point, use/responsibility count incremented by 1, contents supervisor rtns exit (SVC3)

d) if LOAD request being processed, LLE built (Load List Element) containing pointer to CDE and ptr to next LLE, entry point of LOAD'd module returned to caller, no PRB built

2) if module not located on this data set, search link pack area

b. If DCB operand to macro is not coded or coded as 0 check for JOBLIB or STEPLIB DCB (address of DCB in TCB if such a data set was specified)

1) if DCB provided - search data set as above

2) if DCB not provided - search link pack area

note: The TCB contains a 1 word pointer to a DCB for the JOBLIB OR STEPLIB data set (initiator creates and OPEN's this DCB) but for any jobstep task (and thus for any of its subtasks) only one such DCB is provided - STEPLIB data set, if provided; JOBLIB data set, if provided and STEPLIB is not.

3. Contents Supervisor searches Link Pack Area

   a. All modules loaded in LPA at NIP time are available to all tasks in the system

   b. Each module represented by a CDE, all chained together origin of queue is in CVT (LPACQ pointer)

   c. LPA queue of CDE's searched for required module

      1) if found, its available (all modules in LPA are reentrant and thus available for use), PRB constructed, and enqueued after Contents Super's SVRB off requesting task, PSW built to point to modules entry point, use/responsibility count incremented by 1 and contents supervisor rtns exit (SVC 3)

   d. If not found in LPA

4. Contents Supervisor Searches LINKLIB processing as for search of private libraries

5. If module not yet located, requestor ABEND'd

D. Alias Processing XVI D.

1. A module can have up to 16 alias, each is a different entry in the directory of the PDS that contains the true name of the module. Alias names are distinguished from true names by a bit setting in the PDS directory entry and the true name is in the user data portion of the alias entry if module is reusable or reentrant

2. If the name specified in a macro is found in a library (as opposed to the JPA or LPA) and it is an Alias, Contents Supervisor rtns get the true name of the module from the ALias directory entry and re-search the jobpack area for the module under the true name

3. Two types of processing can occur depending on whether or not the module is already in main storage

a. If the module is in main storage

1) a major CDE exists for it, containing true name

2) possibly, minor CDE(s) exist for it - for other alias

3) minor CDE built, containing Alias name specified in the request and the address of associated entry point - either prime entry point or alternate entry point - if it has the same name as the Alias

4) the field in the CDE used to point to the XL in a major CDE, is set to point to the major CDE

note: If a nonserially reusable program is requested by an Alias, a major CDE is built, containing the Alias name and no minor CDE is built. This is because when the program terminates, the program and its control blocks will be freed immediately since the program cannot be used again in its present (executed) state

b. If the module is not in main storage

1) two CDE's built, one major and one minor

2) each CDE contains corresponding name of module prime name in major, alias name in minor CDE

3) corresponding entry point(s) inserted in each CDE

4) minor CDE points to major from field usually used (in a major CDE) to point to XL for the module

5) XL constructed as module is loaded (by FETCH) and is chained out of the major CDE

note: A bit is set in first byte of CDE to indicate whether it is a major or minor CDE and therefore, now the "XL/MAJ" field is being used

E. Special Processing for

1. LOAD macro

   a. Load list for the requesting task is searched before the jobpack queue is searched

      1) LOAD list consists of LLE (Load List Elements) constructed as result of previous LOAD's of modules, origin of list is in requesting tasks TCB

      2) LLE contains

         a) pointer to next LLE on queue

         b) pointer to CDE for the LOAD'd module

         c) responsibility count

   b. If module is located - already LOAD'd

      1) responsibility count (in LLE) and use/ responsibility (in CDE) are incremented by 1

      2) address of module returned to requestor

   c. If module is not located on load list or jobpack queue

      1) LLE, CDE and XL created and module loaded

      2) responsibility count (in LLE) and use/ responsibility count (in CDE) set to 1

      3) address of module returned to requestor

   d. If module is located when jobpack queue searched

      1) LLE built and added to load list of requesting task

      2) responsibility count (in LLE) set to 1, use/responsibility count (in CDE) incremented by 1

note: The CDE thus located is already on jobpack queue and remains there, LLE points to the CDE. Thus that CDE is on both the load list of the requesting task and on the jobpack queue for the region

2. XCTL macro

   a. Informs system that issuing program is transferring control to module specified in macro and is not to receive control back from the specified module

   b. Issuer's PRB is deleted

      1) XCTL rtns are SVC and operate under SVRB

      2) They reverse the issuer's PRB and their own SVRB on TCB queue of RB's (now issuer's RB is top on queue)

      3) Issue SVC 3 - which causes issuers PRB to be deleted and appropriate responsibility, use/responsibility counts to be decremented, if zero, space to be freed, etc.

   c. After SVC 3, SVRB for XCTL rtns is top RB on queue, XCTL searches for requested module and either locates it or loads it and builds appropriate control blocks for it

   d. XCTL rtns build PRB for new module and enqueue below SVRB on requesting TCB's RB queue

   e. XCTL rtns issue SVC 3 thereby, deleting their own SVRB

3. IDENTIFY macro

   a. Informs contents super of a dynamically added entry point to a module presently in storage. New entry point and name exist only while that copy of program is in storage

   b. Must specify

      1) Name to be associated with new entry point

      2) Main storage address of the new entry point

   c. IDENTIFY rtns

      1) Build a new minor CDE for new entry entry point

      2) Search existing load list and jobpack queue to determine if specified address falls in

a module loaded for the requesting program or in the requesting program itself

3) If invalid address - return code to requestor indicates new entry point was not established

4) If address valid - build a minor CDE for new entry point and insert the address, insert pointer in minor CDE to major CDE for module in which new entry point is located

5) Exit to dispatcher

F. Use/responsibility and responsibility Counters XVI F.

1. Where maintained

a. In LLE - responsibility count

b. In CDE - use/responsibility count

2. When incremented or decremented

a. LINK,ATTACH,XCTL increment count in CDE by 1

b. LOAD increments counts in LLE and CDE by 1

c. DELETE decrements counts in LLE and CDE by 1

d. SVC 3 routines (exit) decrement count in CDE by 1

e. XCTL decrements count in CDE of the issuer of XCTL by 1

Ref: MVT Supervisor, Section 5, (Y28-6659)
Handout S14

A. Function Overview XVII A.

1. Responds to requests for use of main storage space or to relinquish use of such space

2. Request issued by GETMAIN/FREEMAIN macros

3. Request may be for space in a region or for a region or for space in SQS

4. GETMAIN/FREEMAIN rtns are SVC rtns

B. Space in a region XVII B.

1. The region itself

a. When region obtained for a jobstep task, a PQE (partition queue element) is built for the region

b. A dummy PQE is constructed and 3rd and 4th words used to point to first and last PQE's for a job-step task (can have several "regions" by Rollout facility in which case a new PQE is built as each new region assigned to requestor). If only region for the task, both pointers in D-PQE point to the single PQE describing the region

c. Pointer inserted in jobstep TCB (propogated to all subtask TCB's in that region) to beginning of D-PQE, thus the address is 8 bytes before the pointers to the PQE(s) for the region

d. PQE contains

1) pointer to FBQE at highest address in the region

2) pointer to FBQE at lowest address in the region

3) pointers (2) to next and previous PQE's for the jobstep task (if there are no others, it is 0's)

4) region size

5) region address (first byte of region)

e. Space in a region is either allocated for use (from jobstep tasks point of view, the protect key of allocated space is that of the task) or unallocated - protect key of 0

note: Exception space in SP252 of jobpack area is allocated but has protect key of 0

f. Space allocated for use may, in fact, be unused. Used space is allocated space, the address of which, has been returned to a requestor in response to a GETMAIN. Space is allocated (i.e. protect key set to that of task) in minimum of 2K blocks.

g. Unallocated space in a region is described by FBQE's (Free Block Queue Elements) which reside in the low order three words of the space they describe and contain

1) pointer to next higher FBQE (if highest, address of PQE for region)

2) pointer to next lower FBQE (if lowest, address of PQE)

3) count of number of bytes in the area this FBQE is describing - count includes the FBQE itself

note: PQE points to highest and lowest FBQE's in the region

2. Jobpack area and its subpool numbers

a. Modules loaded into the region constitute the regions' jobpack area

b. Space in which such modules reside is allocated from either end of the region as follows

1) reentrant modules from LINKLIB or SVCLIB -

high end of region, subpool 252, prot key of 0

2) everything else - low end of region, subpool 251, prot key of the task

c. The subpool numbers associated with the space in the jobpack area are simply signals to the GETMAIN routines as to where to supply the space from and whether or not to change its protect key

note: Program FETCH issues GETMAIN for space in jobpack area, into which to load modules are read.

3. Subpools

Ref: Supervisor and Data Management Services, Section I, Main Storage Management, (C28-6646)

a. A subpool is one or more 2K blocks of storage such blocks are not necessarily contiguous

b. GETMAIN requests for space in a region are satisfied from subpool space, such requests must be satisfiable with contiguous bytes of storage or they are not satisfied

c. A number is associated with a subpool of space for convenience in controlling and manipulating it

d. Space is allocated to a subpool (i.e., protect key is changed to that of requesting task except as noted above) and all or part of the allocation is supplied to satisfy a user request Such space so supplied is considered used or in use and cannot be supplied for another request unless it has been FREEMAIN'd (returned to the subpool) in the meantime

e. User can request space from subpools 0-127

f. System Task subpool numbers

Ref: MVT Supervisor, Section 5, (Y28-6659)

1) Subpool 251 - Space in P/P region, protection

key of the task, low end of region, contains modules not in Subpool 252

2) Subpool 252 - Space in P/P region, protection key of 0, high end of the region, contains re-entrant modules from LINKLIB or SVCLIB

3) Subpools 253, 254, 255, Space in SQA

   a) 253 - space will be freed when associated task terminates

   b) 254 - space will be freed when jobstep task associated with the request terminates

   c) 255 - space will not be freed at task termination but must be explicitely FREMAIN'd

   d) Subpool 255 - number used by supervisor programs (i.e. Initiator getting first save area) to request space in P/P region from subpool 0

g. AQE - Allocated Queue Element

1) Built by GETMAIN rtns for each request for space from subpools 253 and 254, describes space allocated in SQA

2) Requests for space from subpools 253 and 254 are incremented by 8 bytes, and AQE built in first 8 bytes of the area so allocated

3) AQE's chained out of TCB on which termination the space (in SQA) will be freed. The TCB an AQE is chained out of may not be the TCB of the requesting task nor the TCB for which the request is made (e.g., CDE's built as result of any task in the region requesting a module, are all chained out of jobstep task)

4) When the task, from which TCB the AQE was chained terminates, the space described by the AQE will be freed

5) AQE contains

   a) pointer to next AQE

b) length (in bytes) of the area the AQE describes - including the AQE itself

4. Subpools are described and managed by 3 control blocks SPQE, DQE and FQE

a. SPQE - Subpool Queue Element - chained out of a TCB, that task "owns" or "shares" the subpool

b. SPQE contains

1) Bit settings indicating owned/shared status of subpool

2) pointer to next SPQE on chain for that task or to SPQE chained off the task that owns the subpool (if this task is sharing it)

3) subpool number

4) pointer to first (or only) DQE for the subpool (0's if no space allocated to the subpool)

c. DQE - Descriptor Queue Element - chained out of SPQE for the subpool whose space it describes, a DQE is a constructed whenever space is allocated to the subpool (in multiples of 2K blocks)

d. DQE contains

1) address of highest addressed FQE in that block of space

2) pointer to next DQE if there is one (0's if not)

3) block address (one or more 2K blocks allocated at same time to the subpool)

4) size of block (in bytes)

e. FQE - Free Queue Element - resides in low order two words of space it describes, used to describe allocated but unused space in a particular block of space allocated to a subpool. Such a block is described by a single DQE

f. FQE contains

1) pointer to next lower FQE in the block, 0's if lowest

2) byte count of the unused space, including the FQE itself

g. Handling of space in a subpool

1) space used from top down in the space allocated to a subpool

2) space supplied to user as result of GETMAIN must be contiguous. If sufficient space not available in present allocation(s) to the subpool more 2K block's are allocated to it (the protect keys set to that of the task) and a DQE is built for such an allocation and space is supplied to requestor from it

3) FQE's describe only the space associated with the DQE out of which the FQE's are chained. Even if two separate allocations to a subpool are coincidentally contiguous, DQE's for the two allocations do not reflect this nor do the FQE's in the two allocations

4) if GETMAIN's are issued for space so that all space in an allocation is supplied for use, the FQE's are destroyed (space they occupied is supplied to requestor) and pointer to high FQE in DQE is zeroed out. This is also true of FBQE's if all the space in a region is allocated to subpools. In this case the pointers in the PQE to the FBQE's are zeroed out.

5) the FQE, by its location in storage with protect key of the task, can be destroyed by the use if he modifies addresses passed to him by GETMAIN routines. This causes the GETMAIN routines much grief and should only be done at programmers own risk

h. Subpools provide a convenient technique for allocating transferring and sharing space between the tasks of a region. They also provide an efficient method of obtaining and freeing space without undue fragmentation (e.g., if a program issues many GETMAIN's for various sized areas of storage, some small, some large, it would

be best to request all small areas from one subpool and the requests for the large areas from another to avoid space fragmentation that would result if all requests were made from a single subpool)

5. Owned/Shared Subpools

   a. When ATTACH a subtask, mother task can specify that certain subpools be given to or shared with the daughter task

   b. Operands are GSPV or GSPL and SHSPV or SHSPL

   c. Giving a subpool to a daughter task

      1) If subpool exists for mother

         a) SPQE dequeued from mother's TCB and queued off daughter TCB

         b) daughter task can use subpool space as it pleases

note: Control blocks describing space allocated to the subpool and its use, are left as they were when subpool given to daughter. They are <u>not</u> reset to indicate all subpool space is available

      2) If subpool does not exist for mother

         a) SPQE created and chained off daughter task

         b) no space (2K blocks) is allocated to the subpool, therefore not DQE is built and DQE Pointer in SPQE is zeroed out

   d. Sharing a subpool with a daughter task

      1) If subpool exists for mother

         a) SPQE duplicated and chained off daughter TCB

         b) bits set in new SPQE indicating it is not owned by daughter but is shared with it by its mother (this prevents daughter from freeing the subpool)

         c) bits set in mother tasks' SPQE indicating

subpool is shared with another task

d) field in daughter task's SPQE normally pointing to DQE is set to point to SPQE chained off owning task's TCB

2) If subpool does not exist for mother

a) two SPQE's built

b) one chained off mother TCB bits set indicating ownership of subpool

c) other chained off daughter TCB, bits set indicating subpool is shared with daughter by mother; DQE field set to point to owning tasks SPQE

d) No DQE built and no space allocated to the subpool

e. A task can share a subpool that it owns with any or all of its daughters

f. A task whose mother is sharing a subpool with it cannot give that subpool to one of its daughters. It can however share the subpool with its daughters

g. If a subpool is shared between two tasks neither may free the entire subpool while the other task is still in existance. The daughter can never free it, the mother can only free it after the daughter terminates

h. If a task gives away a subpool, it can issue a GETMAIN specifying the same subpool number and a distinct subpool (with the same number) will be created. This is true for any two tasks in a region - if they are not sharing a subpool but each issues a GETMAIN specifying the same subpool number, two distinct subpools, with the same number will be created and space in each managed separately

i. Subpool 0 is shared between all tasks in a region. The System Task Control Task of the job's Initiator owns subpool 0 and thus the SPQE's chained off all TCB's in the region have bits set indicating they do not own the subpool and "DQE pointer" field in the SPQE's point instead to the SPQE for subpool and chained off STCT TCB.

C. Space in SQA and Dynamic area XVII C.

1. DQE for SQA

   a. Space constituting SQA is described by a DQE as if it were a single subpool

   b. FQE's in SQA space describe free space in SQA and highest FQE pointed to from DQE

2. PQE for Dynamic Area

   a. Entire dynamic area (regardless of regions in it) described by a PQE

   b. FBQE's built in available dynamic area space, highest and lowest FBQE's pointed to from the PQE

   c. Other PQE fields contain beginning address of dynamic area and its total size

   d. PQE addressed from 3rd and 4th words of a D-PQE

3. GOVRFLB

   a. A control block contained inline in resident GETMAIN code in Nucleus

   b. Governing Free List Block (?) contains

      1) address of first byte above SQA

      2) pointer to DQE for SQA

      3) pointer to D-PQE for PQE for dynamic area

      4) amount of H0 space available after NIP

      5) amount of H1 space available after NIP

      6) address of queue of VQE's (Vary Queue Elements) used when area of main storage varied offline by MP65 MCH routines

   c. GOVRFLB is pointed to from secondary CVT, which is pointed to from primary CVT

D. Rollout/Rollin * XVII D.

Ref: MVT Supervisor, Section 5, Allocating a Borrowed region thru Rollout, (Y28-6659)

1. Overview

   a. Facility that allows a problem program task to dynamically expand the main storage allocated to it as a region by using additional dynamic space or rolling out the contents of another tasks region and using that space

   b. Contents of region being rolled out is written onto SYS1.ROLLOUT data set

      1) must be large enough to contain all of dynamic space after IPL

      2) is formatted at IPL time, region contents are not written onto first available space but to a location on the data set determined by the address of the region

2. When and how invoked

   a. Problem program issues unconditional GETMAIN

   b. There is insufficient space in region to satisfy request

   c. RORI has been included in system

   d. Requestor can (by JCL operand) cause a rollout

   e. RORI routines invoked from GETMAIN module

      1) RORI TCB (permanent system) made ready

      2) Requestors TCB set non dispatchable

      3) Requestors resume PSW decremented by 2 (to point to GETMAIN SVC again)

      4) Task switch indicated so RORI task will be next task dispatched

      5) GETMAIN exits to dispatcher

3. RORI criterion rtn receives control first and

a. Determines if current request is for RO, RI or restart of a deferred request, passes control to appropriate routines in each case

b. Determines if another jobstep has caused a rollout still in effect - if so, enters a (optional) user appendage to determine

1) if concurrent rollouts should be allowed (IBM routines do not allow it)

2) if no user's appendage supplied or appendage decides against concurrent rollouts - defer request

a) IQE representing request put on a queue of deferred requests

b) request is serviced when a region eligible to satisfy the request becomes available

note: If user appendage allows concurrent rollouts (two distinct tasks each with a rollout in effect at some time) if is responsible for preventing interlock that could result (e.g. two tasks in system, each issues request for more space than is available in entire dynamic area - each waits on the other - both requests must be deferred as there is not enough space to satisfy either request)

4. Search for space to allocate

a. RORI routines check dynamic area for an area of contiguous space large enough to satisfy request

1) if found - PQE for space is built and chained off RORI TCB, FBQE built in the space allocated (allocation performed by GETREGION routines so appropriate control blocks are updated)

a) PQE removed from RORI TCB and enqueued off requestors own PQE, sets TCB ptr field in new PQE to 0 to indicate at Rollin time that the space should be freed rather than restored as another tasks region

b) Rollout count (in nucleus?) incremented

by 1

c) sets "borrowed" flag in new PQE

d) Sets bit in requestors TCB indicating Rollout in effect

e) Makes requestors task dispatchable (when next dispatched, GETMAIN SVC will be re-issued but now space is available to satisfy the request)

f) exits

2) if space not found in dynamic area - search for a jobstep to rollout, it must meet these requirements

a) has not itself, caused a rollout still in effect

b) it is eligible to be rolled out (JCL operand)

c) its region is large enough to satisfy requestors GETMAIN

3) Finding a jobstep to check for availability is performed as follows

a) requestor's tCB checked to see if it has a rollout currently in effect if so checks for sufficient space in previously borrowed region(s) to satisfy current request

note: This seems a duplication of effort as the GETMAIN rtns should already have checked the borrowed regions before scheduling the RORI task

b) if space not found, search TCB queue from requestor down, each TCB passed to a routine to test the tasks

(1) if no suitable task of lower priority is found, enters an (optional) user appendage which determines whether or not TCB's of priority higher than requestor should be tested

(2) if higher priority tasks should be tested, TCB queue is reprocessed from top down

(3) if no user appendage is supplied for this condition or no task is suitable to be rolled out, request is deferred see 3.b.2) above

4) A jobstep found by search above, is eligible to be rolled out if:

note: Nested rollouts (a task, with a rollout in effect, cannot be rolled out) are never permitted, no user appendage can contravene this. The reason is because of the way the region is written on the SYS1.ROLLOUT data set - to a location determined by an algorithm that causes each region to be written to a location on the data set corresponding to that region's location in the dynamic area. If a task, with a rollout in effect, were rolled out, both regions belonging to that task would be mapped to their corresponding locations on the data set; the borrowed region overlaying the original contents of the region, written to the data set when earlier rollout occurred

a) selected jobstep has not, itself, caused a rollout still in effect

b) selected jobstep indicated on JCL that it can be rolled out

c) selected jobstep is not ENQ'd on a resource

d) selected jobstep's region is large enough to satisfy GETMAIN request

e) selected jobstep's region has not already been rolled out and loaned to another task

f) jobstep meets additional tests performed by an (optional) user appendage

5) If a jobstep suitable to be rolled out cannot be found on (optional) user appendage is entered it can

a) cause request to be deferred as described in 3.a.2) above

b) abnormally terminate a jobstep task - the requestor or another task in the system

5. Allocation of another region

a. All tasks in region to be rolled out are set non-dispatchable

b. All I/O requests are quiesced

1) queued requests are purged (RQE's freed) but pointers to IOB's are kept to allow requests to be reinstated

2) active (started) I/O requests are allowed to complete

c. Operator replies to tasks in region are deferred (saved in temporary buffers)

d. Contents of region are written out to SYS1.ROLLOUT

1) channel programs set up

2) TTR calculated where writing will begin on data set

3) If an I/O error occurs, partially rolled out region is restarted and search for eligible jobstep to be rolled out is resumed

4) if no error, operator message written stating job causing rollout and job rolled out

e. Region is allocated to requestor, RORI routine

1) sets flags in owning tasks PQE indicating region rolled out and is in use by a borrower

2) builds a new PQE for the rolled out region and chains it off requestors TCB, sets up pointers and bits as follows

a) TCB pointer field in PQE points to owners TCB

b) bits set to indicate region is borrowed

3) Sets all protect keys in borrowed region to 0

4) constructs FBQE in low end of region, describing all of region as free

5) "rollouts invoked" count (in nucleus?) incremented by 1

6) RORI rtns exit, setting requesting task dispatchable

6. Rollin

a. When FREEMAIN rtns invoked to free space and these rtns detect that an entire region has been freed, they invoke the RORI task to perform Rollin

b. FREEMAIN rtns

1) check each PQE chained off TCB of task issuing FREEMAIN to see if any such regions are borrowed (bits in PQE indicate this) and if all space in the region is not allocated to a subpool

2) releases such a region from borrowing task by dequeueing and freeing PQE

3) Invoke RORI to

a) check if region was allocated from free space (if so, TCB pointer in PQE is 0), if so, free the space

b) if region borrowed, its original contents rolled in if owning jobstep has no borrowed regions still rolled out

4) Rollin performed as follows

a) protect keys of region changed to that of owning task

b) contents of region read into region space from SYS1.ROLLOUT data set

c) writes message to greater describing job rolled in

d) If I/O error - ABEND jobstep that was partially rolled in

e) resets rollout flags in owners PQE

f) resets protect key of free blocks in region to 0

g) restores deferred I/O requests

h) restarts deferred operator replys

i) makes dispatchable the tasks in the rolled in region

5) Deferred rollout requests are restarted

a) rollout counts decremented by 1

b) clear rollout flag in borrowers TCB

c) moves IQE for deferred request to RORI TCB, making RORI TCB ready

d) clears non dispatchability in requestors TCB

e) exits to dispatcher

Ref: Job Management, Part 3, Attaching the Jobstep, Terminating the Jobstep (Y28-6660)
MVT Supervisor, Section 6, Section 9, Dispatching, handling job and step timing, Section 3, Services indirectly related to the TCB (WAIT and POST rtns) Section 9, Dispatching, Section 13 Flowcharts of TIME, STIMER, Timer SLIH, etc. (Y28-6659)
Supervisor and Data Management Services, Section I, Program Management Services, Timing Services (C28-6646)

A. Function Overview XVIII A.

1. Allows setting of time intervals (by macros) and cancelling of same

2. Provides date and time of day when requested

3. Allows scheduling of interrupts at specific times of day

4. Is entered from SVC FLIH, SVC SLIH or Timer SLIH

   a. TIME macro - requests date and time of day, is a type I SVC

   b. TTIMER macro - requests time remaining in a previously set interval or to cancel that interval, is a type I SVC

   c. STIMER macro - requests a time interval be established and requesting task be interrupted when interval expires, is type II SVC

   d. Timer SLIH receives control from External FLIH detects a timer interrupt, Timer SLIH in turn passes control to routines of Timer Supervisor to handle the interrupt

B. The Interval Timer XVIII B.

1. Is a hardware feature

2. Values are placed in it (by software rtns) and they are decremented, when the value in the timer becomes negative - the interval has expired and an external interrupt is generated

3. The Timer is initialized and checked if working at IPL time

   a. a 6 hour value placed in the timer and decrementing begins

   b. a 6 hour value placed in a software location called the six hour pseudo clock (SHPC), it is not decremented but each time a new value is placed in the timer, that same value is placed in the SHPC

   c. Another software location, the twenty-four hour pseudo clock (T4PC) is initialized to 0. Every 6 hours after initialization, a 6 hour value is added to the value in the T4PC, except when the value in it is 18 hours, in that case it is re-set to zero and the date is changed

note: Precautions are taken at initialization time that this happens at midnight after IPL

   d. The date, specified in operator's SET command is placed in the CVT, is updated at midnight

   e. The time specified in operator's SET command, is placed in a third software location called the Local Time pseudo clock (LTPC) and is not changed unless the operator executes a SET command with the "CLOCK" operand

C. The Timer queue XVIII C.

1. Is a software feature

2. Consists of Timer Queue Elements (TQE) representing requested intervals of time

3. Is manipulated by STIMER routines and Timer SLIH

4. TQE contains

   a. Bit settings indicating

      1) Type of TQE (supervisor or problem program, whether TQE is on or off queue)

      2) Type of interval request

         a) TASK - interval decremented only while associated task is active

         b) REAL - interval decremented continuously

         c) WAIT - task put in WAIT condition for duration of interval

   b. Address of requesting task

   c. Link fields to preceeding and following TQE's

   d. Time of expiration (TOX) - if TQE is on the queue, or time remaining in interval - if TQE is off the queue

   e. Save area for time remaining when TQE converted from TASK to REAL (jobstep timing)

   f. Address of save area to be used by problem program

   g. Address of user asynchronous timer completion routine (if specified in macro)

   h. Extra space needed when TQE converted to IQE and IRB for scheduling the asynchronous rtn

5. Types of TQE's

   a. Supervisor TQE

      1) Indicated by bits in TQE

      2) Contains an interval of 6 hours (in which case its called a "6 hour TQE") or an interval which will generate an interrupt at midnight after IPL (in which case its

called the "midnight element")

3) These elements are always on the queue, when they are associated with an interrupt the interrupt is handled and the elements are repositioned on the queue

4) Thus, if no other timer interrupts occur (due to tasks issuing STIMER macro) there will be an interrupt every 6 hours and at midnight

5) 6 hour interrupt

   a) Detected as such by combination of supervisor bit in TQE and 6 hour value in TOX field

   b) Processed as follows

      (1) 6 hours subtracted from all other TQE's in queue

      (2) T4PC updated as described above

      (3) 6 hour value placed in TQE and it is repositioned on the queue

6) Midnight interrupt

   a) Detected by combination of supervisor bit in TQE and value of 18 already in T4PC

   b) Processed as follows

      (1) Date in CVT updated

      (2) T4PC set to 0

      (3) 24 hour value placed in TQE and it is repositioned on the queue

b. Task TQE's

1) Indicated by bits in TQE

2) Contains interval specified in task's STIMER macro

3) Contains indicators as to how interval is to be decremented (WAIT, TASK, REAL)

4) Task TQE's placed on the queue and removed from the queue according to type of TQE (TASK/REAL) and whether or not task is active

5) TOX value is recalculated every time a TQE is reenqueued so it reflects true time remaining in the requested interval

6. TQE's on timer queue are arranged in ascending order of Time of Expiration (TOX) or length of time left in the requested interval

7. Whenever a Timer interrupt occurs, the first or top TQE is the one associated with the expired interval

D. STIMER routines XVIII D.

1. Respond to STIMER macro

2. Build and position the TQE on timer queue

a. Indicate type of request - REAL, TASK, WAIT

b. Calculate time of expiration depending on how the interval is specified in STIMER macro

1) interval: TOX =
(SHPC - Timer) + interval requested

SHPC = current value of SHPC
Timer = current value in interval timer

note: The TOX may appear wrong but remember, it is recalculated at various times during the interval it represents

2) Time of day: Interval=
(Time of Expiration) - (Current time of day)
Current time of day =
(LTPC + T4PC + SHPC) - Timer
LTPC = local time pseudo clock, is time specified by operator in set command

T4PC = current value in twenty four hour pseudo clock
SHPC = current value in six hour pseudo clock
Timer = current value in interval timer
The interval so calculated is then used to calculate a TOX as above

3. When enqueued on the timer queue, the enqueue rtns

   a. Compare interval in subject TQE to current value in timer

   b. If TQE interval less than value in timer - update the timer

      1) new interval is developed:

         new interval = SHPC + (TQE interval - Timer)

      2) new interval put in SHPC

      3) old TQE interval put, unchanged, in the timer

      4) new interval put in TQE, replacing previous interval

      5) TQE enqueued at top of timer queue

   c. If TQE interval greater than or equal to value in timer

      1) new interval developed

         new interval = SHPC + (TQE - Timer)

      2) new interval put in TQE, replacing previous interval

      3) TQE enqueued on timer queue according to new interval as TOX

E. Timer Interrupts (Timer SLIH) XVIII E.

1. Occur when value in interval timer becomes negative

2. Timer SLIH rtns receive control from external FLIH

3. Timer SLIH proceeds as follows

   a. Dequeues top TQE from timer queue

   b. Determines type of TQE

      1) WAIT (STIMER issued with WAIT option)

         a) POST the ECB in TQE

         b) Branch to rtn to develop new interval to be put in timer (see 3)b) below)

      2) Supervisor TQE processed as in C. 5. a. 5) and 6) above

      3) Task TQE

         a) If timer completion rtn specified by user (in STIMER macro)

            (1) rebuild TQE into IQE, IRB

            (2) invoke stage 2 exit effector to schedule the asynchronous routine

         b) If no timer completion rtn specified, or on return from stage 2 Exit Effector, proceed as follows

            (1) new interval = (TOX from new top TQE) - (previous SHPC)

            (2) new interval is placed in interval timer

            (3) TOX from new top TQE is placed, unchanged in SHPC

   c. Returns to caller (external FLIH, which branches to Dispatcher)

F. Task Timing

1. A task issues STIMER macro, specifying WAIT, REAL or TASK option

2. STIMER routines

a. Build a TQE, put pointer in it to requesting tasks TCB

b. Put pointer in the TCB to the TQE

c. Enqueue the TQE on the timer queue, interval calculated by Enqueue routine as in D.3. above

3. If WAIT option specified

a. Task will be put in WAIT condition by STIMER rtns

b. TQE put on timer queue and "decrements continuously"

note: "decrements continuously" is a confusing phrase - it seems to imply the TOX in TQE is being decremented while in the TQE. Such is not the case - there is only one interval timer and thus only one interval is decremented. But the decrementing of the timer will have an effect on th other TQE's on the queue, as if those intervals were being decremented, since when an interval expires, the TOX originally calculated for it is subtracted from the TOX in the next TQE on the queue and the result is the new interval placed in the timer. Thus the expiration of one interval causes that interval to be deducted from the next, and that second interval, when it expires will be deducted from the third, etc., etc.

c. When associated TQE is top in queue and a timer interrupt occurs

1) Top TQE dequeued

2) Associated task POST'd out of WAIT state, ECB is in TQE itself

3) Next TQE used to develop a new interval to be put in the Timer, as in E.3.b.3) above

4. If REAL option specified

a. TQE put on timer queue and will "decrement continuously"

b. Task will compete with other tasks for CPU and other resources

c. When timer interrupt occurs and this TQE is top on timer queue

1) TQE dequeued from timer queue

2) If a user timer completion routine was specified, TQE rebuilt into IQE and IRB, stage 2 exit effector invoked to schedule the asynchronous routine

3) On return from stage 2 exit effector, next TQE on queue used to develop a new interval to be placed in interval timer as in E.3.b.3), above

5. If TASK option specified

a. TQE put on queue and will decrement only while associated task is active

b. When task enters SVC WAIT state (e.g., issues SVC 1) the WAIT routines

1) Dequeue the TQE from timer queue (and if it was the top TQE, use next TQE to develop new interval and put it in timer as in E.3.b.3) above

2) Dequeue rtns also calculate "time remaining" in the original interval and replace TOX in TQE with that value

remaining time = old TOX - (SHPC - timer)
old TOX = interval from dequeued TQE

note: There are times when a task is put in WAIT condition by system routines, not by issuing SVC 1 but using a branch entry to the WAIT routines in the nucleus. In these situations the system routines are performing a service for the task (e.g., locating an available transient area block and loading it) and thus the tasks TQE is not dequeued for, although the tasks code is not being executed, system routines are executing on its behalf and thus it is "using" the CPU

c. When task POST'd whether POST clears WAIT count in tasks top RB or not, the POST rtns do nothing to (directly) cause resumption of task timing

d. When dispatcher is to perform a task switch and this task is the "new" task to be dispatched dispatcher routines process as follows

1) Check if "old" (displaced) task was being timed - checks its TCB for a pointer to a TQE

a) if no TQE - continue

b) if there was a TQE

(1) dequeue it, only if it is TASK type!

(2) put in it (in TOX field) absolute time remaining in its interval

(3) if dequeued TQE was top on queue, use next TQE to develop a new interval and put it in interval timer as in E.3.b.3), above

2) Check if "new" task has a TQE chained out of its TCB

a) if no TQE - continue

b) if a TQE is there

(1) check if it is already on timer queue, if it is - no further processing; if not, then

(2) compute new interval as in D.3., above

(3) if new interval is smaller than value currently in timer, replace it with new interval

(4) in any case, enqueue "new" task TQE at appropriate position on timer queue

3) Dispatch the NEW TASK

e. When timer Interrupt occurs and that TQE is top on queue

1) TQE dequeued from queue

2) If a user timer completion rtn was specified, it is scheduled as for a REAL timer expiration

3) In any case, next TQE used to develop a new interval to be placed in interval timer as in E.3.b.3), above

G. Jobstep Timing XVIII G.

1. Allows user to specify TIME values on JOB and/or EXEC cards

2. If TIME operand used on both cards

a. Each jobstep allowed to execute (be active task, use CPU) for time specified on EXEC card as long as total amount of time used by the steps of the job will not exceed JOB card time limit

b. If allowing a jobstep to execute for EXEC TIME value will exceed JOB TIME limit, jobstep is allowed to execute for an interval equal to difference between JOB TIME and EXEC TIME

3. If TIME only on JOB card - first jobstep given JOB TIME limit, it any time remains, 2nd jobstep allowed to execute for balance of the interval, etc., etc.

4. If TIME only on EXEC card - each jobstep allowed to execute for time interval specified

5. If TIME specified on neither card - Reader/Interpreter has inserted installation defined default values

note1: Jobstep timing is a sys gen option, if not selected at that time the routines to affect it are not included in the system and thus it cannot be implemented, TIME operands are ignored

note2: Jobstep time limits apply to the associated jobstep task or any of its subtasks - that is, if any task in the jobstep region is active, the time interval is

being decremented

6. Initiator of a jobstep calculates time interval for that jobstep and, on return from ATTACH that created the jobstep task, issues STIMER macro, TASK option, specifying the interval. Thus the TQE created is chained out of the initiators TCB

7. The TQE is manipulated by various system routines

   a. WAIT routines

      1) When entered via SVC, issued by any task in the jobstep's region, WAIT routines check all tasks in the "family tree"

         a) If all tasks are in SVC WAIT condition, TQE chained off initiator's TCB is located and

            (1) dequeued and remaining time in interval saved, as in F.5.b.2) above

            (2) converted to a REAL TQE

            (3) 30 minute interval put in it

            (4) re-enqueued as a 30 minute "WAIT state time out" TQE

         b) If any task in the jobstep tasks "family tree" is not in SVC WAIT condition, no change to TQE chained out of initiator's TCB

      2) If a timer interrupt occurs and top TQE on queue has following attributes

         a) REAL type TQE

         b) chained out of Initiator's TCB

         c) exit routine specified

         that TQE represents a 30 minute WAIT state time out and the associated jobstep task and all subtasks must be ABEND'd. Processing is as follows:

d) ABTERM routines branched to

(1) schedule task for abnormal termination by pointing PSW to an SVC 13 instruction in CVT

(2) set up ABEND code indicating 30 minute WAIT time limit was exceeded

e) ABEND (SVC 13) rtns will POST the Initiator's ATTACH ECB thus signalling the termination routines that the task has terminated, error code indicates its abnormal

f) On return from ABTERM, timer rtns

(1) dequeue top TQE

(2) convert it back to TASK type

(3) time remaining in jobstep's time interval is calculated and placed in TQE, as in F.5.b.2), above

note: Since task is being terminated because it exceeded 30 minute WAIT limit, there may still be time left in its TIME interval and it may be used by succeeding jobstep of the job

b. POST routines

1) Entered to indicate completion of an event, if wait count in associated RB is not 0 after POST is complete - return to caller. But if associated RB's wait count is 0 - proceed as follows

a) Check if initiator, associated with task being POST'd has a TQE chained out of it, if not - return to caller, if yes

(1) if TQE is TASK type - some task on the "family tree" was not in SVC WAIT and is presently being timed, no further processing is necessary

(2) if TQE is REAL - it is a 30 minute WAIT timer - it is

(a) dequeued

(b) converted to TASK type

(c) "time remaining in interval" (which was saved when it was converted to a REAL WAIT state timer) is put back in the "interval" field of TQE

(d) TQE not reenqueued yet, when it is, new interval will be calculated

c. Dispatcher routines

1) Perform jobstep timing if

a) task switch necessary

b) jobstep timing selected at sys gen time

2) Perform as follows

a) removes from timer queue the TQE (if one exists and is TASK type) associated with the initiator of the "old" task. When dequeued the time remaining in its interval is saved in TQE

note: The "old" task's initiator is located via the "old" tasks "family tree"

b) if dequeued TQE was top on queue, new interval is developed and placed in interval timer as in E.3.b.3), above

c) locates TQE (if one exists) associated with initiator of "new" task and if it is TASK type - enqueues the TQE, enqueue rtns

(1) develop new interval as in D.3., above

(2) if new interval is smaller than that presently in the timer, it updates timer with new value

(3) enqueues the TQE in timer queue

d) If TQE located in c) above is REAL, it indicates all tasks in the "family tree" were WAITing and a 30 minute WAIT timer had been set up. Jobstep timing can now be resumed since the dispatcher is going to dispatch one of the tasks in the tree

(1) TQE dequeued

(2) converted to TASK type

(3) time remaining in the interval (saved when tQE converted to a WAIT timer) is replaced in interval field in TQE, enqueue rtns develop a new interval as in D.3., above and TQE enqueued on timer queue

note: It seems the POST routines should have done this conversion but the dispatcher does check for it and converts the TQE if necessary

e. "new" task is dispatched

d. Timer SLIH

1) If timer interrupt occurs and top TQE is

a) TASK type

b) associated with an initiator

c) user exit specified

the jobstep has exceeded its time limit and must be terminated, processing is as follows

d) TQE dequeued and convert to an IQE and IRB, and initialized

e) Stage 2 exit affector invoked to schedule user (initiator's) timer completion rtn

f) on return from stage 2 exit effector,

next TQE on queue used to develop a new interval and place it in timer as in E.3.b.3), above

g) Initiator's "timer completion rtn" just POST's the CANCEL ECB which invokes terminator to terminate the jobstep task and all subtasks in the region

note: Since the task is being terminated because it ran out of CPU time, there is no "time-remaining" to be calculated and possibly used for the next jobstep, that's why the TQE is not saved as it was in 30 minute WAIT state time out situation

H. TIME macro routines XVIII H.

1. Determine current date and time of day, return them to requestor in registers

2. Date obtained from CVT

3. Time of day is calculated by determining elapsed time

elapsed time = (SHPC + T4PC) - timer
SHPC = 6 hour pseudo clock
T4PC = twenty-four hour pseudo clock
Timer = value currently in interval times

4. Elapsed time is added to LTPC to arrive at time of day
LTPC = Local Time Pseudo Clock - value specified by operator in SET Command

I. TTIMER macro routines XVIII I.

1. Provide time remaining in a previously established interval for requesting task, or

2. Cancel a previously requested interval for the requesting task

3. Remaining time calculated

remaining time = TOX - (SHPC - timer)
TOX = time of expiration in the TQE chained out of requesting tasks TCB
SHPC = current value in SHPC

Timer = current value in interval timer

4. A time interval is cancelled by

   a. dequeueing from the timer queue the TQE chained out of requesting tasks TCB

   b. zeroing out the field in the TCB that pointed to the TQE

   c. freeing the TQE

note: A given task can have only one TQE associated with its TCB. If a second STIMER is issued before a previous interval expires, the new interval replaces the older one and the TQE is repositioned on the queue.

X Topic Deleted

Trace Table XX

Ref: MVT Supervisor, Section 12, Trace Table
Programmers Guide to Debugging, Section 2, (C28-6670)

A. Purpose XX A.

1. Debugging aid, contains entrys describing various interrupts and system information at time of interrupt

B. Size and usefulness XX B.

1. Number of 8-word entrys in the table is specified at sys gen time (table is in nucleus)

2. Table space used in wrap-around manner, when last entry filled, first entry overlayed when it is necessary to make another entry

3. This use of the table makes its usefulness in an MVT environment open to debate - if the table is small a particular entry, containing information about an abnormally terminating task, may be overlayed by newer entries (caused by that task or other tasks in the system) before the trace table can be dumped

C. Entrys XX C.

1. When made

a. When an SVC, I/O, External or Program Interrupt occurs

b. When SIO instruction executed

c. When Dispatcher entered

2. Information in an entry

a. SIO instruction

1) condition code

2) identifier (5th hex dig. in first word of entry is 0)

3) device address

4) CAW

5) CSW

6) reg 1

7) TCB pointer

8) Timer

b. External, SVC, Program Interrupts

1) PSW stored result of interrupt

2) Identifier in 5th hex digit of PSE

a) External - 1

b) SVC - 2

c) Program - 3

3) Reg 15, 0 and 1

4) TCB address

5) Timer

c. I/O Interrupt

1) PSW stored as result of interrupt

2) Identifier - 5th hex digit of PSW is 5

3) CSW

4) Reg 1

5) TCB address (of TCB that requested I/O that has completed)

6) Timer

d. Dispatcher

1) new PSW

2) Identifier, 5th hex digit of new PSW is D

3) reg's 0 , 1, 15

4) new TCB address

5) Timer

D. Trace Table Control XX D.

1. Location 54 (hex) contains address of 3-word trace table control area

2. Control Area Contains

a. address of last used entry

b. address of beginning of table

c. address of end of table

MVT Termination routines - resident, non-SVC type, branch entry XXI

Ref: MVT Supervisor, Section 10, (Y28-6659)

A. Function Overview XXI A.

1. Free resources and control blocks associated with terminating task

2. Optionally, inform an ancester task of the termination of this task

3. Optionally, schedule execution of an ETXR rtn

B. Entered from Exit rtns (SVC 3) when they detect the last RB on a Tasks queue of RB's. SVC 3 rtns have already XXI B.

1. Moved requestors (Terminating tasks) registers from IEASCSAV to requestors TCB

2. Removed TCB FROM "main" or dispatcher TCB queue

3. Set "normal completion" flag in TCB (prevents its being re-dispatched)

C. Normal Termination Processing XXI C.

1. Check for incomplete or un-DETACH'd subtasks of the terminating task - if there are any such subtasks, terminating task is ABEND'd

note: This ABEND will eventually cause the subtask to be ABEND'd but that is handled by the ABEND rtns

2. If no subtasks, store completion code of terminating task in its TCB

3. Check for and free a PIE

4. Purge TQE associated with terminating task from timer queue

5. Check for resources the task is still ENQ'd on - if there are any, ABEND the task

6. Purges any outstanding "WTOR" buffers

7. Closes all OPEN data sets

   a. TCB points to DEB queue there is a DEB for every OPEN data set associated with the task

   b. DEB points to associated DCB, CLOSE macro issued for that DCB

   c. DEB points to next DEB

   d. if errors occur in closing procedure - ABEND task

8. Contents Directory rtns invoked to free or reschedule the program last executed for terminating task

9. Releases programs LOAD'd but not DELETE'd

   a. Check if there are outstanding requests for the program

      1) subtract LLE responsibility count from CDE use/responsibility count, if result greater than 0 - there are outstanding requests and space is not freed

      2) if no outstanding requests - LLE's, are freed

10. Issues FREEMAIN for unshared subpools and SPQE's, DQE's

11. If a jobstep task is terminating - free jobpack area

12. Schedule execution of ETXR if one was specified when terminating task was ATTACH'd

    a. Pass control to Stage 2 Exit Effector to place IQE (chained out of terminating task) on AEQJ

13. Deferred rollout requests for terminating task are purged from rollout queue

14. TCB dequeued from dispatcher queue and "normal completion" and "non-dispatchible" flags set in TCB

note: It appears this is a duplicate effort as the SVC3

rtns have already dequeued the TCB and set "normal completion "flags when they detected on end-of-task condition

15. If an ECB was specified when the terminating task was ATTACH'd, the ECB is now POST'd with tasks completion code, ECB is pointed to by a field in terminating tasks TCB

note: If a jobstep task is terminating, this POST notifies Init/Terminator rtns which perform jobstep task termination functions

16. If neither an ETXR rtn nor an ECB was specified when the terminating task was ATTACH'd, the TCB is dequeued from its subtask queues and its space freed

17. Termination routines indicate the need for a task switch

note: Manual (MVT Supervisor) indicates NEW is set to 0 but this could cause problems. The "POST'ing" of the ECB on the completion of a jobstep tasks would have made READY the initiator and the POST rtns would have set NEW to point to the initiator's TCB, setting NEW to 0 would destroy this and (apparently) the Initiator would not be dispatched except by chance at a later time. The Initiator was not dispatched after the POST rtns completed as they were entered via a branch not via an SVC and they returned to caller rather than take type I SVC exit to the dispatcher

18. Return is to exit rtns (SVC 3) which free last RB and branch to transient area refresh routines

D. Abnormal Termination Processing XXI D.

1. Performs three functions for abnormal termination

   a. ABTERM routine, which schedules a task for abnormal termination, is a resident, non interruptible non SVC rtn

   b. ABEND routine - frees resources from terminating task and its incomplete subtasks and if it is a jobstep task, frees all resources belonging to all tasks in the region

   c. ABDUMP routines - provides a dump of the storage and control blocks associated with the terminating task and control blocks associated with the

terminating tasks direct descendants and ancestors

2. ABTERM Rtn

   a. Refreshes CVT address at loc 10 (hex) in case it has been overlayed by routines in error

   b. Checks address of TCB passed to ABTERM and determines if task should be scheduled for abnormal termination (it could already be scheduled for termination, normal or abnormal) and checks if its subtasks should be set non-dispatchible

      1) if task already terminating normally, returns to caller (of ABTERM)

      2) if task already scheduled for abnormal termination

         a) set subtasks non dispatchible

         b) checks whether it is a jobstep task, and whether or not initiator is the caller (of ABTERM)

   c. If jobstep task and not already in process of abnormal termination

      1) makes task dispatchible, and sets NEW pointer to point to it

      2) set up parameters for use by ABEND

      3) set PSW in top RB of tasks queue to point to SVC 13 instruction in CVT

      4) makes tasks subtasks non disptachible

      5) returns to address set up by caller (usually dispatcher)

   d. If jobstep task is already being abnormally terminated and initiator has not invoked ABTERM

      1) indicates an error in ABEND rtn and is a system error unless re-entry occurs because of error in ABDUMP OPEN or CLOSE rtn

      2) if system error, invoke damage assessment rtn (DAR)

2) if it did not - continue

b. Clears non-dispatchibility flags in tasks (set by ABDUMP)

c. Determines if serious error has occurred (system task is terminating) and passes control to DAR rtns if it has

d. Determines if entry to ABEND is a re-entry

e. Purges deferred rollout requests for abending job

f. Sets all subtasks of ABEND'ing task non dispatchable

g. Checks FQE's in subpools of the task - if they are involved, issuing FREEMAIN's to free the subpools could cause a recursive ABEND and be a system error

h. PIE's, RQE's, WTOR buffers, IQE's for abending task are purged and elements freed

i. Closes all data sets still OPEN whose DEB's are chained out of ABEND'ing TCB

j. Releases partially loaded modules (maybe I/O error in loading caused ABEND)

k. Open SYSABEND or SYSUDUMP data set depending on DD cards for step and dump option flags in RB

l. If I/O error in OPEN'ing dump data set - exit to dispatcher, ABTERM will be re-entered

m. If no error - continue - take the dump and CLOSE data sets

n. Remove SVRB's for transient SVC's from requestor's RB Queue and from transient area user queue

o. Purge all other RB's on tasks queue

p. Purge contents directory and load list

q. FREEMAIN dynamically acquired main storage

r. Release TCB unless it is a subtask of another task that is not being terminated and that

other task specified ETXR or ECB operand when it ATTACH'd the task that is abending

s. Set NEW pointer to 0, exit to dispatcher

4. ABDUMP routines

a. Set all tasks (ancestors and descendants) of abending task non-dispatchible (except requesting task)

b. Formats and dumps storage and control blocks as requested (same routines invoked by SNAP)

Ref: Handout S21, V26-6156

A. Volumes and Data Sets XXII A.

1. Volume - collection of one or more data sets on a recording medium, is usually but not necessarily demountable

2. Data set - collection of related records, or, collection of data

3. System keeps track of and recognizes different volumes and data sets by labels written at specified locations on the volume or with respect to the data set

4. Volume and data set labels for direct access and tape volumes have different sets of restrictions

B. Preparation for I/O XXII B.

1. Before I/O can be performed various control blocks and modules must be created or modified or loaded

2. Access methods - modules to perform device dependent processing, thus relieving the problem programmer from it

3. Control blocks must be built or modified before I/O performed. Some are built by the programmer, other by access method routines still others by OPEN routines

4. OPEN routines complete or build all control blocks and LOAD modules necessary to perform I/O, OPEN'ing a data set logically connects the data set to the system

C. Initiating I/O XXII C.

1. Problem program uses macros to perform linkage to access method routines (BALR) which perform final modification on control blocks and issue SVC to initiate I/O

2. Access methods do not perform standard entry and exit linkages and are thus only loadable (LOAD)

3. Access method routines operate as closed subroutines of the user program

D. EXCP Supervisor XXII D.

1. Access method routines issue EXCP macro which expands into an SVC 0

2. SVC Interrupt handlers pass control to I/O Supervisor EXCP Handler

3. These routines verify the request and schedule the I/O requested

4. After either queueing or starting the I/O, EXCP Handler takes a type 1 exit to calling routine

E. When an I/O Interrupt occurs, I/O new PSW points to I/O FLIH which preserves the status of the interrupted task and passes control to the I/O Interrupt Handler XXII E.

1. I/O interrupt analyzed and catastrophic errors handled by error handling rtns

2. Task requesting I/O is POST'd (via branch entry to POST) and allowed to analyze the completed event when it (the task) is next dispatched

3. Channel, freed by completion of I/O, is restarted by I/O Interrupt Handler

4. All stacked interrupts are handled before exit to dispatcher

Ref: Supervisor and Data Management Services, Section II, Part I; Appendix A (C28-6646)

A. Direct Access Volumes XXIII A.

1. Volume label

   a. Must be on cyl 0, trk 0 of the volume, contains

      1) volume serial number

      2) bits indicating volume type, owner protection, etc.

      3) pointer to VTOC data set

   b. When a device comes READY, AVR rtns move volume serial number and VTOC address into UCB

note: Every DASD volume has a 24-byte IPL record at beginning of track 0, cyl 0 (for non-IPL volumes, this IPL record just puts the system in a WAIT state); volume label follows this 24-byte record. If volume is an IPL pack, IPL bootstrap record and IPL CSECT follow the volume label.

2. VTOC

   a. Data set containing Data Set Control Blocks (DSCB's) which describe

      1) the VTOC data set itself

      2) every data set on the volume, data sets on more than 3 extents require a second DSCB to contain addresses of the extents. Indexed Sequential data sets require two DSCB's to describe the various types of space (Index, Prime, Overflow) allocated to it

3) free space on the volume - when an area, adjacent to an area already free, is freed, the areas are merged and 1 DSCB built to describe the entire area

b. Location and size of VTOC determined by user when volume is initialized

3. DSCB's

Ref: System Control Blocks (Y28-6628)

a. Are labels for the data sets they describe

b. Contain

1) data set name

2) data set organization

3) record format

4) block size

5) logical record size

6) pointer to data set on volume

7) number of extents and address of each

B. Tape Volumes XXIII B.

Ref: Tape Labels (C28-6680)

1. Volume label

a. Optional, not required by OS/360

b. If used, must be first record after load point

c. Contains

1) volume serial number

2) owner, protection bits

2. Data set header labels

   a. Optional under OS/360

   b. If used, must preceed data records

   c. Header labels

      1) standard

         a) two 80-byte records, first 4 characters in each must be HDR1 and HDR2 respectively

         b) contain

            (1) data set name

            (2) data set organization

            (3) record format

            (4) block size

            (5) logical record size

      2) User

         a) up to 8 80-byte records following standard labels (if standard labels are used)

         b) must begin with characters UHL1-UHL8

         c) can contain any information user wishes

         d) are written and read by user supplied routines

3. Trailer Labels

   a. Standard

      1) two 80-byte records following last data record, begin with EOF1 and EOF2 respectively

      2) contain same information as header labels plus a count of number of physical blocks

in the data set (used for error checking when data set reprocessed)

b. User

1) up to 8 80-byte records preceeding standard labels (if standard labels are used)

2) must begin with characters UTL1-UTL8

3) can contain any information the user choses, are written and read by user routines

Ref: System Control Blocks (Y28-6628)
I/O Supervisor, Section V (Y28-6616)

A. Data Set Description XXIV A.

1. DCB - Data Control Block

   a. Created by user, using a macro, is therefore in the user module, will be the major control block when I/O actually performed

   b. Partially filled in by macro expansion from macro operands, necessary operands are

      1) MACRF

      2) DDNAME

      3) DSORG - necessary for proper expansion of macro even though the information is available from the data set label

      4) EODAD - for input files

   c. Completed by OPEN rtns from information in data set label and on DD card

2. JFCB

   Ref: System Control Blocks (Y28-6628)

   a. Created by Reader/Interpreter from data set information on DD card

   b. Contains

      1) data set name

      2) volume serial number

      3) DCB information specified on DD card

4) space allocation information (for NEW data sets)

c. Is read into main storage, from its location on SYS1.SYSJOBQE, when corresponding DCB is OPEN'd

3. Data Set Label (DSCB if on DA)

Ref: System Control Blocks (Y28-6628)
Tape Labels (C28-6680)

a. Skeleton DSCB created by DADSM routines when space allocated to data set at initiation time, filled in at OPEN time; created by OPEN/CLOSE rtns when data set opened on tape

b. Exists in VTOC of volume that contains the data set or is the Header Labels if on tape

c. Contains

1) data set name

2) DCB information

a) DSORG

b) RECFM

c) LRECL

d) BLKSIZE, etc.

3) location of data set, if on DA

B. Access Methods XXIV B.

Ref: Sequential Access Methods (Y28-6604)
Basic Direct Access Methods (Y28-6617)
Indexed Sequential Access Method (Y28-6618)

1. Function and types

a. To perform device dependent processing required for various data set organizations and mediums of storage

b. Relieve programmer of detailed, device dependent

processing

c. Are LOAD'd by OPEN rtns, address of required modules are put in corresponding DCB's

d. Invoked by branch entry (most access method modules are LOAD'able only) and return to caller

e. Construct and modify various control blocks and CCW's that will be used in performing, monitoring and analyzing I/O

f. Issue EXCP macro, which generates an SVC 0 which passes control to I/O EXCP handler to schedule I/O

g. There are 3 types or levels of access methods

1) Queued level (GET/PUT macros)

a) anticipates request for next record

b) de-blocks and blocks records

c) automatically allocates (min of 2) buffers

d) primes buffers at OPEN time

e) constructs and checks all control blocks (except DCB) necessary for I/O

f) issues SVC to schedule I/O

2) Basic level (READ/WRITE/CHECK macros)

a) negation of a) - d) above

b) constructs all control blocks (except DCB and DECB) needed for I/O

c) issues SVC to schedule I/O

3) EXCP level (Execute Channel Program)

a) constructs no control blocks

b) issues SVC to schedule I/O

2. Selection and LOAD'ing

a. User specifies in DCB what macros (MACRF) he intends to use and the organization of the data set (DSORG)

b. OPEN rtns check these two operands and on basis of them and OPEN option (INPUT, OUTPUT, UPDAT, etc.) decides which access method modules to LOAD

c. Modules are LOAD'd, address of module put in appropriate DCB

C. Control Blocks

Ref: System Control Blocks (Y28-6628)
I/O Supervisor, Section V, (Y28-6616),
Handout S22

1. DCB

a. User created and partially filled, completed by OPEN rtns

b. Used by access method rtns to contain device and data set information as I/O is performed

2. IOB - I/O Block

a. Created by access method rtns at OPEN time

b. Contains

1) ECB address

2) Space for CSW stored when an I/O event completes

3) ADdress of channel program

4) DCB address

5) Full, 8-byte seek address

c. One IOB created for each channel program the user requests in DCB operand (NCP).

3. DECB - Data Event Control Block

   a. Created in READ/WRITE macro expansion

   b. Contains pointers and indicators kept elsewhere by GET/PUT rtns

   c. Contains the ECB used to POST completion of I/O event

4. ECB - Event Control Block

   a. Created by GET/PUT rtns or by READ/WRITE macros

   b. Used to WAIT and POST completion of I/O events

5. DEB - Data Extent Block

   a. Created by OPEN rtns, destroyed by CLOSE, exists only when a data set is OPEN

   b. Chained out of TCB as well of associated DCB, all DEB's for a tasks I/O are chained together (so at termination time, system rtns can CLOSE any data sets not yet CLOSE'd)

   c. Contains

      1) TCB address

      2) DCB address

      3) priority of associated task

      4) address of appendage vector table

      5) address of UCb of device allocated to data set

      6) beginning and ending addresses of each extent of the data set

      7) last 2 characters of names of access methods LOAD'd for this data set

note: All access methods are named IGG019__, the last two characters are all that are needed to identify an access method. The identifiers are kept here so CLOSE can issue DELETE's for the routines

6. TIOT - Task I/O Table

   a. Created by Initiator device allocation rtns

   b. Contains an entry for each DD card provided for the jobstep, each entry contains

      1) DDNAME of the card

      2) address of JFCB for the card (TTR address)

      3) address of UCB allocated for the data set described on the card

   c. Chained out of TCB, accessed by OPEN rtns to locate JFCB and thru the UCB, the VTOC and thus the label (DSCB) of the data set being OPEN'd

7. Channel Program

   a. Not really a control block but very important in I/O

   b. Created by access method routines at OPEN time

   c. Modified and scheduled for execution by access method rtns at READ, WRITE, GET or PUT time

D. OPEN'ing the Data Set

Ref: I/O Support, Opening a DCB (Y28-6609)

1. Merge DSCB or label into JFCB

   a. OPEN rtns when invoked by OPEN SVC, locate the DCB being OPEN'd and obtain DDNAME from it

   b. Search TIOT for entry corresponding to the DDNAME and extract JFCB address and UCB address

   c. Using JFCB address, it is read into main storage and data set name is extracted

   d. Using UCB address and its pointer to volume's VTOC, VTOC is searched for data set named in JFCB

note: If it is a tape data set, DD card specifies a data set sequence number and the tape is positioned to that data set and header labels are read

e. DSCB or label is read into main storage

note: If user label handling routines or DCB exits are provided for this, they are given control after OPEN rtns read the labels into storage

f. A zero-merge is performed from the DSCB or label, into the JFCB

note: A zero-merge involves moving only information not specified in the block accepting the merge (in this case, the JFCB) from the other block (in this case, the DSCB or label)

2. Merge JFCB into DCB

a. Next, the "completed" JFCB is zero-merged into the DCB

b. Next, the DCB operand - EXLST is checked for and if coded, the list is scanned for a code 5 entry and if present, the indicated routine is given control

note: OPEN rtns use SYNCH to give control to the routine and thus it executes under its own PRB

3. Access Method routines are LOAD'd

a. On basis of DSORG and MACRF operands in the DCB and OPEN options, access method executors are loaded and given control

b. The executors build and initialize (as much as they can) the IOB, DEB, ECB and channel program

note: If Basic Access Methods are used, channel program and IOB cannot be completed until READ or WRITE instructions executed, also ECB is built by READ/WRITE macro expansion, not by executors

c. The executors LOAD the necessary access method routines - determined by bit settings in DCB, set by user options specified in DCB macro or on DD card (e.g., buffering techniques, modes of data processing (QSAM) etc.)

4. OPEN rtns finally turn on the OPEN bit in the DCB and terminate

note: The CLOSE routines must restore the DCB to the state it was in before OPEN, to do this a mask is built at OPEN time indicating which fields were modified by OPEN, the mask is kept in the DEB. The DDNAME field is overlayed by OPEN and must be restored, an offset into the TIOT (to the entry corresponding to the DDNAME which the DCB had in it) is stored in the DCB and is used by CLOSE to restore the DDNAME field

E. End Appendages XXIV E.

Ref: I/O Supervisor, Section I, Start I/O Subroutine (Y28-6616)

1. Function/Purpose

   a. Given control at various points in I/O operations to allow extra processing or error checking

2. SIO (Start I/O)

   a. Given control just before "SIO" instruction executed

   b. Usually does last minute modification of next channel program to be executed on basis of completion of previous channel program

3. PCI (Program Controlled Interrupt)

Ref: I/O Supervisor, (Y28-6616)

   a. When channel fetches a CCW with PCI bit on, channel generates a PCI Interrupt (an I/O interrupt)

   b. I/O Interrupt handler gives control to PCI appendage addressed from appendage Vector Table in DEB of associated data set

   c. PCI interrupt allows appendage to notify the requesting program (usually by setting bits in a field or POST'ing an ECB) that a particular CCW in the Channel program has been fetched and thus previous CCW's have completed.

4. End of Extent

Ref: I/O Supervisor, (Y28-6616)

a. Entered when seek address in IOB is not within one of the extents of the data set as indicated in the DEB

b. For input operations - ABEND

c. For output - appendage checks if secondary space allocations have been requested and if so, if the data set has used them all. If no more space can be allocated - ABEND

d. If more space is to be allocated, invoke DADSM rtns to do so, modify DEB to reflect new extent of data set, modify CCW to indicate an address in the new extent and reissue the CCW

5. Channel End

Ref: I/O Supervisor, (Y28-6616)

a. Entered when channel end, unit exception with or without channel end, or channel end with wrong length record occurs

b. Performs checking to determine if hardware or software errors have occurred

6. Abnormal End

Ref: I/O Supervisor, (Y28-6616)

a. Entered when abnormal conditions occur - unit check, program check - CCW, protection check chaining checks, etc.

b. Set appropriate indicators, if hardware error, invoke recovery routines, if software errors, indicate it to requesting program

EXCP Supervisor XXV

Ref: I/O Supervisor, Section 2, (Y28-6616)
Handout S16, S17, S18, S21

A. Function Overview XXV A.

1. Receives control as result of EXCP SVC 0

2. Performs validity check on the I/O request

3. Attempts to schedule the I/O request

4. If it cannot be performed immediately, queues the request

5. In either case, takes Type 1 exit to requestor

B. Validity Check XXV B.

1. EXCP handler checks boundary alignment of control blocks passed to it, EXCP passes address of IOB from that can get to DCB and ECB and to TCB

2. If addresses are invalid (wrong alignment) or if DEB not in protected core - ABEND

3. If all okay, try to schedule the I/O request

C. Scheduling the I/O Request XXV C.

1. EXCP handler attempts to get a free RQE (Request Queue Element)

a. All RQE's are in a table - Request Element Table - in the nucleus. Each RQE in an MVT system is 16 bytes long and the number of RQE's is determined at system generation

note: This table is also known as the 16* table. In MFT each RQE is 12 bytes long and the table is known as 12* table

b. RQE's contain a queueing field use to chain free

RQE's or RQE's enqueued for I/O on a particular channel

c. CVT contains address of RQE free list - a one word field pointing to the first free element in the table or, if there are no free elements, X'FFFF'

d. If no RQE is available, requestors resume PSW is backed off by 2, task set non-dispatchible and switch set in code so when an RQE is freed, all tasks waiting for an RQE will be cleared and eventually dispatched - re-issuing the SVC 0

note: If there is just one RQE left, EXCP Supervisor checks if current request is for SVCLIB. If not, then it is processed as if no RQE is available. If current request is for SVCLIB, the last RQE is used. This is done because I/O Error Recovery routines are on SVCLIB and I/O Supervisor doesn't want to be in a situation where it needs to load such a recovery routine and lack the RQE needed to perform the loading.

e. If an RQE is available, it is filled in with information about current I/O request

1) UCB address (from DEB)

2) IOB address (parameter to EXCP)

3) priority of requesting task (from TCB, pointed to from DEB)

4) DEB address (IOB points to DCB, DCB points to DEB)

5) protection key of requesting task (from TCB)

6) TCB address (from DEB)

f. From this point on, all information needed to schedule and perform and POST the I/O event, is in, or can be located from pointers in, the RQE

2. Unit checked for availability
a. If RQE filled in, next check unit for availability

b. Check bits in UCB - unit <u>not</u> available if bits

indicate

1) control unit busy

2) device busy

3) device not ready

4) seek in progress

5) error routine in control

c. If unit not available, queue the RQE for later processing

note: See discussion of LCHWD's, below, for the queue origin.

d. If unit available, locate a path to it

3. A Logical Channel is located

a. A logical channel is the collection of hardware paths to a device from the CPU

note: This logical channel concept is distinctly different from the logical channel concept developed in the Job Management PLM (Y28-6660) when discussing I/O device allocation (by the Initiator) and equalizing channel usage

1) a logical channel may contain (involve) several hardware channels

2) a device is associated with one and only one logical channel

3) at sys gen time the various logical channels are grouped and a routine to search each logical channel for an available hardware path to the associated device is generated and the address of this module (called the Test Channel Module) is inserted in a table

4) The table, containing one entry for each logical channel in the system, is called the Logical Channel Word Table (LCHWD table) and each entry in it is called LCHWD

5) Each entry is two words long and contains

a) queueing field (two halfwords) for RQE's representing I/O requests on the devices associated with the logical channel

note: It is here that the RQE is queued for a request when the device is unavailable

b) the address of the test channel module for the corresponding logical channel

b. The UCB for the device associated with I/O request contains an index factor which is added to the address of the LCHWD table to locate the appropriate LCHWD and thus the appropriate test channel module

c. The test channel module is given control to search for an available hardware path to the required device

1) if no path available - queue RQE off LCHWD and return to requestor

2) if a path is available - locate the SIO module

4. SIO module

a. Each device type on the system requires device dependent commands to be set up for it and executed before the user channel program(s) can be executed

b. At sys gen time a module to do this is created for each type of device and address of module is inserted in the Device Table (DEVTAB)

1) each UCB set up with an index factor into DEVTAB to locate the appropriate entry

2) DEVTAB entrys are 6 bytes long and contain

a) address of enqueue module - to enqueue an RQE to the LCHWD queue according to priority or FIFO options selected for each device at system generation

b) SIO Module Address

c) Address of Trap Code module - to analyze various bit settings in stored CSW's to determine what conditions mean for associated device (e.g., unit check means different things for card reader than for a DA device)

c. SIO Module for the device associated with I/O request is located and given control

d. It constructs device dependent CCW's but does not execute them - such commands as

1) set file mask

2) stand alone seek

3) tape positioning commands

4) write tape mark, etc.

e. SIO Module then branches to SIO Subroutines which

1) enters SIO appendage

2) issues SIO instruction indicating a CAW which indicates appropriate channel program

3) branches to post SIO subroutine which

a) checks condition code set by acceptance by channel of SIO instruction

b) if code is 0 - all okay, address of RQE put in UCB and type 1 exit taken to requestor

c) if code = 3 - device not available - error routine entered to analyze problem

d) if code = 1 CSW Stored and it is examined for error conditions

(1) channel errors - SER or CCH rtns entered

(2) busy - SIO retried or RQE re-enqueued

(3) channel end - immediate command, I/O

event complete

(4) program or protection check - abnormal end appendage entered, if error not reset, I/O event complete with '41' code

(5) unit check - sense command issued and abnormal end appendage entered, then error interface

(6) attention - attention routine for device given control - UCB contains index factor into ATTNTAB to locate appropriate attention rtn for the device

f. Special processing in SIO module for DA devices

1) stand alone seek (not chained to any other CCW's) CCW is constructed

2) SIO Module checks 8-byte seek address, specified in IOB, to the associated data set (these are described in the DEB)

a) if not - end of extent appendage entered

(1) Input - ABEND

(2) Output - check if secondary space allocations were specified when data set created. If so and more allocations are available, DADSM rtns invoked to allocate more space

(a) other routines (EOV) are invoked to modify DEB to describe new extent, then seek CCW modified to point to first track of new extent and the seek is re-exectued.

b) if seek address is within an extent of the data set - continue

3) SIO subroutine entered to execute stand alone seek and set "seek in progress" flag in UCB

4) module then enters a TIO loop until CSW stored

(channel end indicated) and checks bits thus set. (Do not have to enable interrupts to do a TIO and thus "allow" the storing of the CSW)

5) if seek was not completed immediately (arm movement necessary), RQE re-enqueued off LCHWD queue and Type I exit is taken to requestor, request will be handled later by I/O Interrupt Handler

note: RQE address is in UCB and it is also queued off LCHWD in this situation

6) If seek completed immediately, data transfer begun by entering SIO Module again to execute a triple of CCW's

   a) Seek - repeat of stand alone seek, to re-set head register in control unit which might have been modified by other seeks issued on same control unit but different devices while this seek was awaiting completion (only one seek that causes arm movement or head switching can be issued in a channel program)

   b) set file mask - file mask, obtained from DEB, determines what commands can be used in rest of channel program (only one set file mask CCW can be used in any channel program). File mask built by OPEN routines

   c) TIC to user channel program

note: The restrictions noted above about seek and set file mask commands are to protect against accidental (or deliberate) access to data on a pack other than the data set which was OPEN'd.

Ref: I/O Supervisor, Section II, (Y28-6616)
Handout S16, S17, S18, S21

A. Function Overview XXVI A.

1. I/O FLIH receives control as result of an I/O interrupt, causing I/O new PSW to be loaded

2. Switch checked to determine if this interrupt interrupted I/O Interrupt Handler or another program

note: I/O Interrupt Handler is reusable and when once entered, with interrupts masked off, re-enables I/O interrupts at end of processing to handle stacked interrupts. If it was not executing while an interrupt occurred, the status of the interrupted task is preserved but the I/O interrupt handler operates under that task's TCB and on successive re-entry's from itself, will not cause status to be preserved as status (reg's and PSW) would be its own but they be overlaying reg's and PSW of the interrupted task

3. If interrupted another program

   a. Save registers in tasks TCB, PSW in top RB of tasks RB queue

   b. Set switch to prevent such action on re-entry from I/O interrupt handler - continue

4. If interrupted I/O interrupt Handler continue

5. Locate requestor of completed I/O

6. Perform some error checking on completion

7. POST requesting task with completion code

8. Restart hardware channel freed as result of I/O completion

1. Using 3-digit channel, control unit and device address in stored I/O old PSW, Interrupt Handler indexes into UCB look up table to get address of UCB corresponding to the device

   a. UCB look up table built at sys gen time

   b. Contains entry for every UCB in system

2. From the UCB, get address of RQE representing I/O request and thus pointers to TCB, DCB, IOB of requestor

3. Using status bits in CSW stored as result of I/O Interrupt, I/O Interrupt Handler does some error checking

   a. Channel data check, Channel control check, Interface Control Check

      1) put system in WAIT state or

      2) enter SER or CCH rtns

   b. Control unit end - response to interrogation by restart routines, channel restart rtn given control

   c. PCI bit - control passed to PCI appendage

   d. Channel End - completion of an I/O event

      1) CSW moved to IOB

      2) Trap code module entered (address is in DEVTAB entry corresponding to device, index factor in UCB)

      3) Channel End Appendage given control and it indicates further processing

         a) POST request complete and free RQE

         b) free RQE, but do not POST event

         c) reschedule request

d) ignore interruption, pending asynchronous user rtn

e. Device end - either indicates completion of I/O event or device has moved from not ready to ready state. Various cases analyzed and handled

f. Attention - attention rtn interface entered to pass control to attention rtn for device

g. Unit Check - abnormal end appendage taken and indicates further action

1) continue normal processing

2) skip further work on this request

3) reschedule request

4) enter I&M error rtns

h. Unit exception and incorrect length - channel end appendage taken, on return, error rtns entered then abnormal end appendage taken

i. Program check, protection check and chaining check - abnormal end appendage taken, on return, error rtns scheduled

j. Status modifier, busy - control unit busy, UCB located and busy bit set in it

C. Error Routines XXVI C.

1. Interfaces

a. Entered to prepare for execution of various error routines and generally receive control blocks with indicators specifying continued processing

b. Error Routine Interface

1) checks if IOB indicates a permanent error - if no enters POST routine interface

2) determines if I&M error rtns are to be used (indicator in DCB)

3) if IBM rtns are to be used, invokes Stage 2 Exit Effector to chain associated RQE off AEQA and set stage 3 switch and thus schedule error rtn asynchronously

c. SER/CCH Interface

1) Entered when catostrophie errors detected (e.g., channel control check, interface control check)

2) Routes control to SER0 or SER1, CCH or MCH depending on sysgen options

d. ABTERM routine interface

1) Entered to abnormally terminate a task when

a) invalid DEB or UCB's are found

b) UCB address in DEB would give a specification error

c) Protection key specified in DEB does not match that of TCB

d) Protection key in RQE is non-zero and is not same as key of IOB, DCB and IOB

2) In each case, the I/O event is <u>not</u> POST'd complete

e. POST routine interface

1) Not an error rtn it is entered to perform the POST of an ECB

2) Prepares 36 bit completion code and invokes task supervisor POST rtns to perform the POST function

3) If also entered when a task is being abnormally terminated and RQE's are being purged, in which case the SVC Purge rtn (not to be confused with the I/O purge rtn) is WAIT'ing on completion of the purge and the post interface is used to POST that completion

2. Error Routines

a. IBH routines

1) Entered from Error Rtn Interface, passes RQE to Stage 2 exit Effector to schedule asynchronous execution of appropriate rtn by chaining RQE off AEQA and setting Stage 3 switch

2) Exit Effector completes name of required error rtn by appending a 1-byte code (from UCB) onto a standard name in the SIRB chained out of the system error TCB

note: If the free list of RQE's is empty this is not done and offending task is ABEND'd. The loading of the required error rtn requires I/O and thus requires its own RQE

3) The system error rtn loads the rtn into the I/O transient area and executes it

4) If necessary, the I/O Purge Rtn is loaded to purge all RQE's for related requests (related to the one in error)

note: Related requests are distinct I/O requests issued by the same task using the same DCB and DEB such that the first must complete before the next can be started

a) Searches all LCHND queues for RQE's related to one in error and associated with extents defined in the DEB

b) the RQE's are returned to the free list and the corresponding LCB's POST'd with a "permanent error" code

3. SER/CCH Routines

a. SER - see Machine Check Interrupt section

b. CCH _ Channel Check Handler

1) forms a series of bytes describing the error to be used by error rtns in an attempt to retry the operation

2) formats a record describing channel environment when error occurred

1. If no errors found or they are recoverable, I/O Interrupt Handler attempts to restart the hardware channel associated with completed I/O event

2. First, interrupts are enabled and if any occur, I/O FLIH is entered and by switch set on first entry, bypasses storing of registers and PSW and handles the interrupt. After all stacked interrupts are handled, channel restart continues

3. Channel address is multiplied by 4 to index into the channel table to get address of Channel Search Module

4. Channel search module

   a. Searches each LCHWD queue of each logical channel with which the physical channel is associated

   b. Searches first for "seekable" requests - requests that require a stand alone seek

      1) When a seekable request found, unit checked for availability. If not available, next RQE checked. If available, test channel module entered to check if a path to the device is available, if not available next RQE checked. If available, SIO module entered to issue stand alone seek

      2) If RQE represents a request related to another request which has encountered a permanent error (code set in associated DCB), the channel restart module

         a) puts X'48' (purged request) in associated IOB, to be used to POST ECB later

         b) dequeues RQE and returns it to free list

         c) invokes POST rtns to post event complete

   c. When all "seekable" requests that can be, have been started, Channel Search modules search the LCHWD

queues again for a start data transfer request, when found, SIO Module entered to execute the Triple of CCW's leading to the users channel program

d. When data transfer begun, interrupts enabled again

e. If no interrupts occur, control returned to I/O FLIH to reset "first entry" switch (so register and PSW will be saved the next time on I/O Interrupt occurs) and exits to the dispatcher