ADVANCED COMPUTER SYSTEMS

AUERBACH INSTITUTE

ADVANCED COMPUTER SYSTEMS APPLICATIONS, TECHNIQUES AND CONCEPTS

June, 1968

Training Course
Prepared and Presented
by
AUERBACH Institute

AUERBACH
®

TABLE OF CONTENTS

NOTES

- First and only time course will be offered
- Course was prepared under contract for NASA
- Instructors are not Auerbach employees

- Instructors
  - Jim Anderson
  - Beryl Blickstein

AUERBACH

OBJECTIVE OF FIRST HALF OF COURSE

- DESCRIBE THE ORIGINS AND DEVELOPMENTS OF SYSTEMS ARCHITECTURAL FEATURES PRESENT IN 3RD GENERATION COMPUTERS.
  - HARDWARE

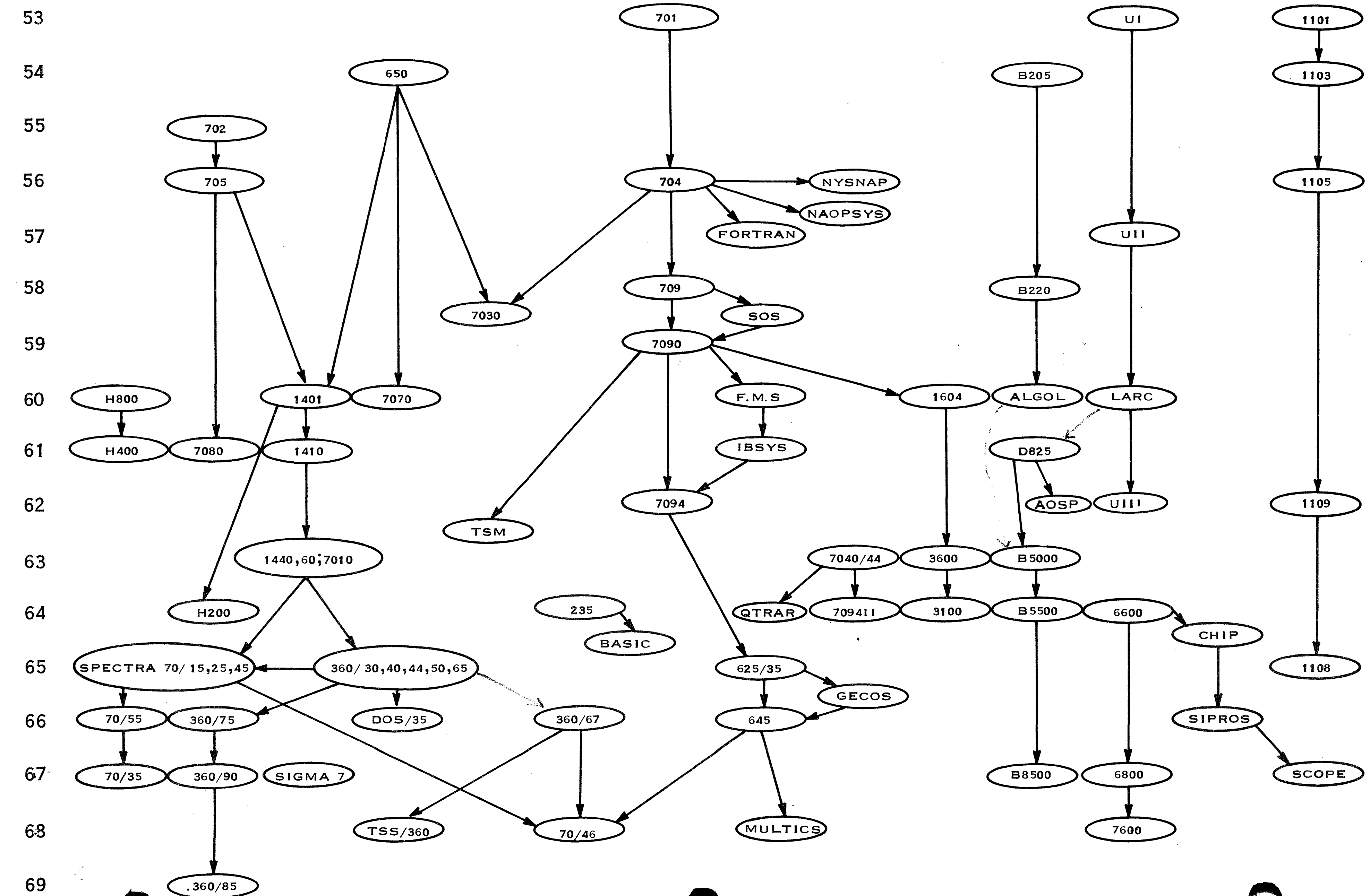
    MULTIPROCESSORS
    MICROPROGRAMMING
    INTERRUPT SYSTEMS
    SCRATCHPAD
    HIGH PERFORMANCE MACHINES
    STACK MACHINES
    TIME SHARING SYSTEMS

  - OPERATING SYSTEMS

    MULTIPROGRAMMING
    RESOURCE ALLOCATION
    TIME-SHARING
    REAL-TIME

PURPOSE: TO IDENTIFY IMPORTANT ORGANIZATIONAL CONCEPTS FOR COMPUTING SYSTEMS, AND THEIR AREA OF GREATEST APPLICABILITY.

CHRONOLOGY AND GENEALOGY OF COMPUTER SYSTEMS



MAJOR INFLUENCES leading to O.S.

- OPERATING EASE
- THRUPUT
- PROGRAMMER SERVICE
- LANGUAGE SUPPORT

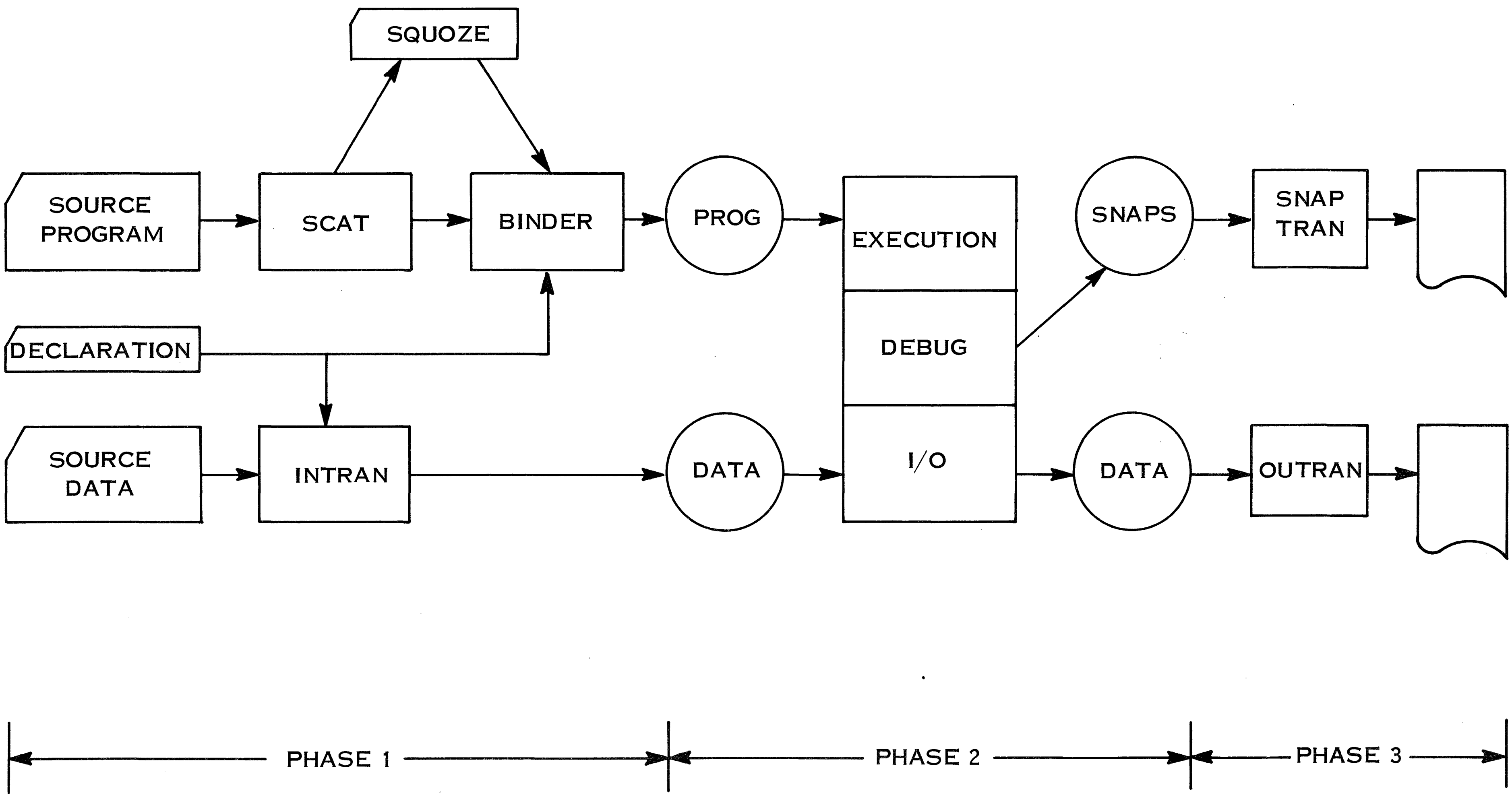
TRADEOFFS of O.S.

- MORE SERVICE
- LESS SPACE
- MORE EXECUTION TIME
- LESS PROGRAMMING TIME
- THE EGO QUESTION

THE SYSTEM AS AN ENVIRONMENT

- USER HAS NO CHOICE
- FACILITIES PROVIDED:
  - PROGRAM INSERTION
  - DEBUGGING
  - LANGUAGE TRANSLATION
  - CORRECTION
- MORE SOPHISTICATION
  - I/O
  - LINKAGE
  - MULTIPLE LANGUAGE SUPPORT

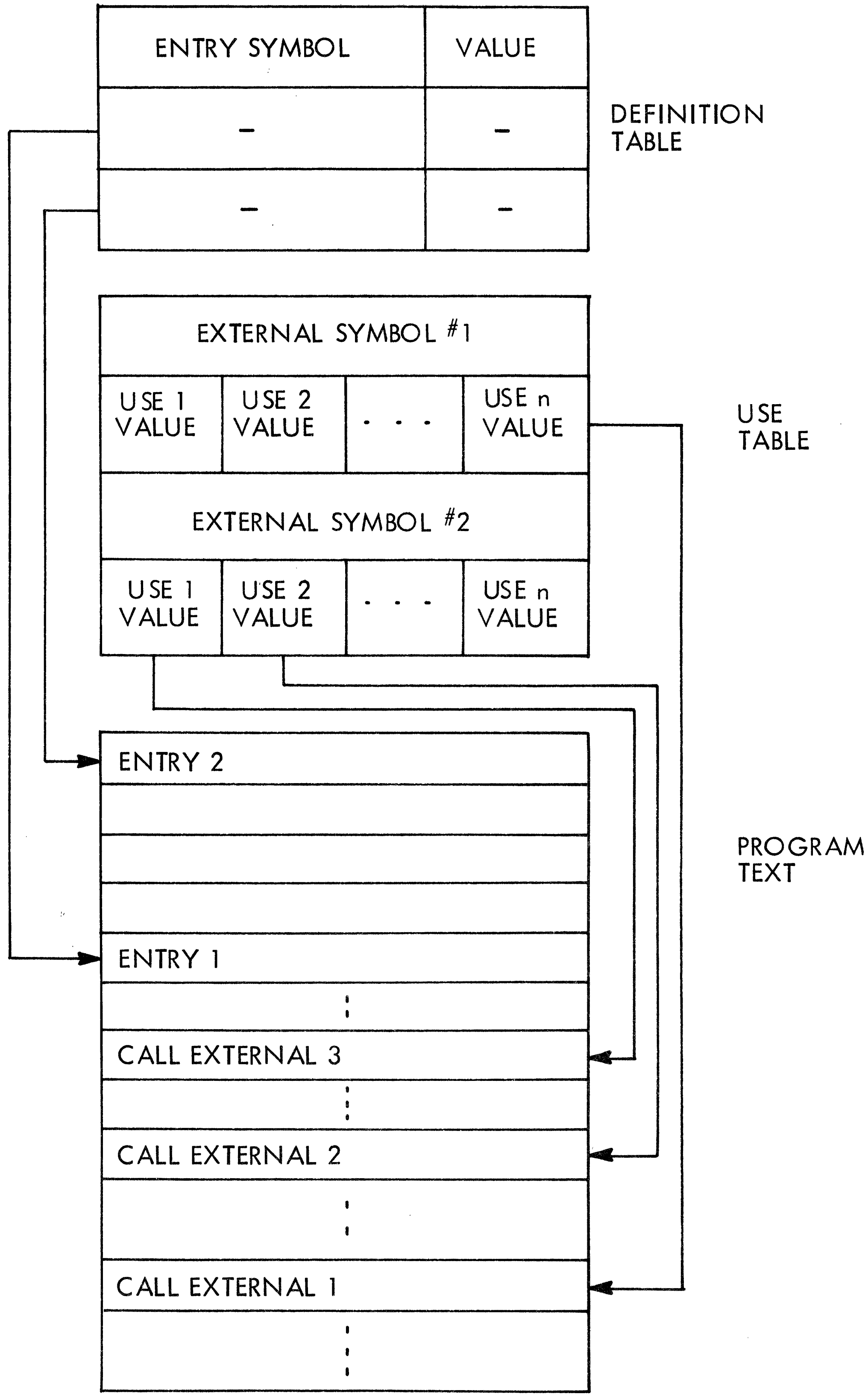
A 3–PHASE OPERATING SYSTEM



BINDING CONCEPT

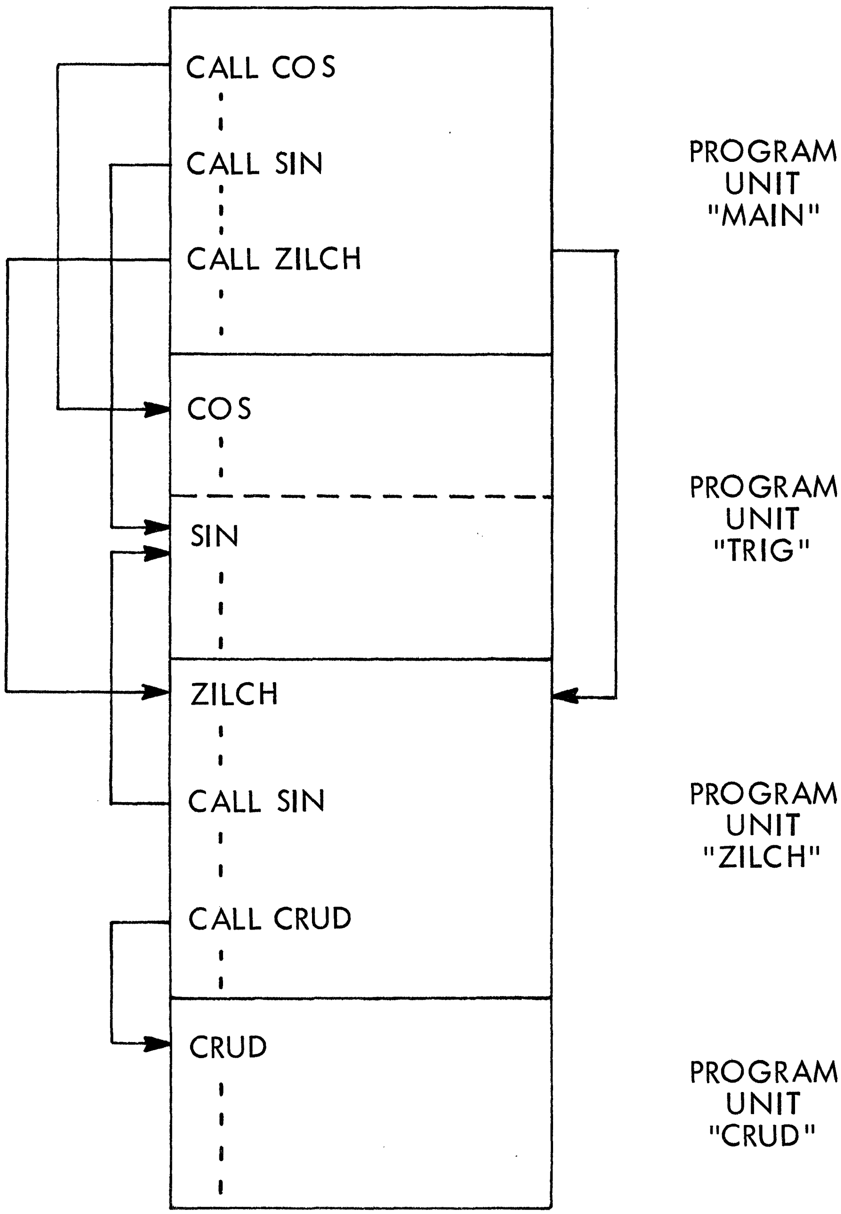
- TIME WHEN PROGRAM IS ASSIGNED ACTUAL LOCATIONS IN MEMORY
- EARLY SYSTEMS – DURING CODING
- BY ASSEMBLER
- BY RELOCATION AND LINKAGE PROCESS
- BY SYSTEM, VIA COMPACTING
- DYNAMICALLY, BY PROCESS CALL

USE AND DEFINITION TABLES FOR A PROGRAM UNIT

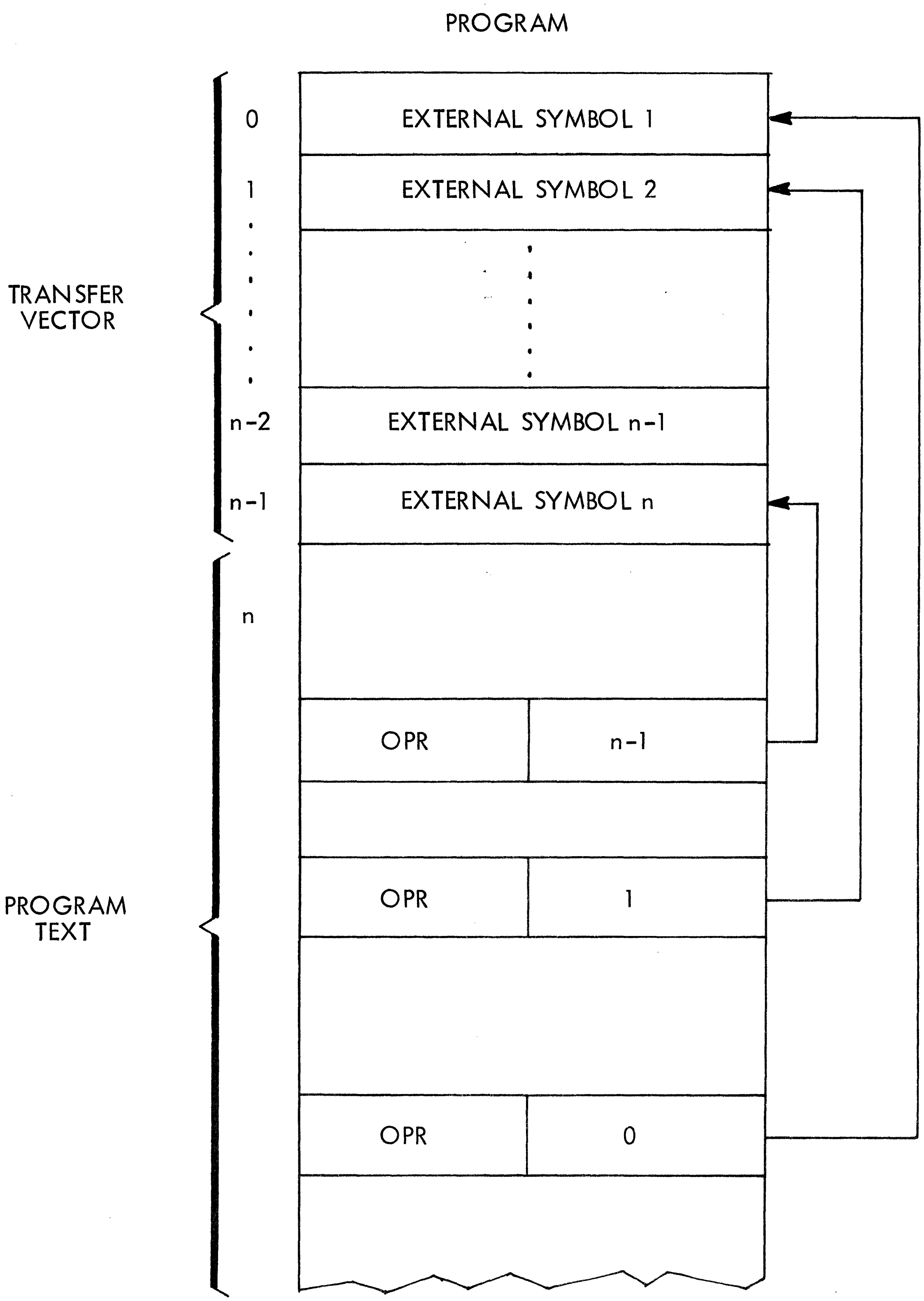


MAIN/SUB-PROGRAM ORGANIZATION

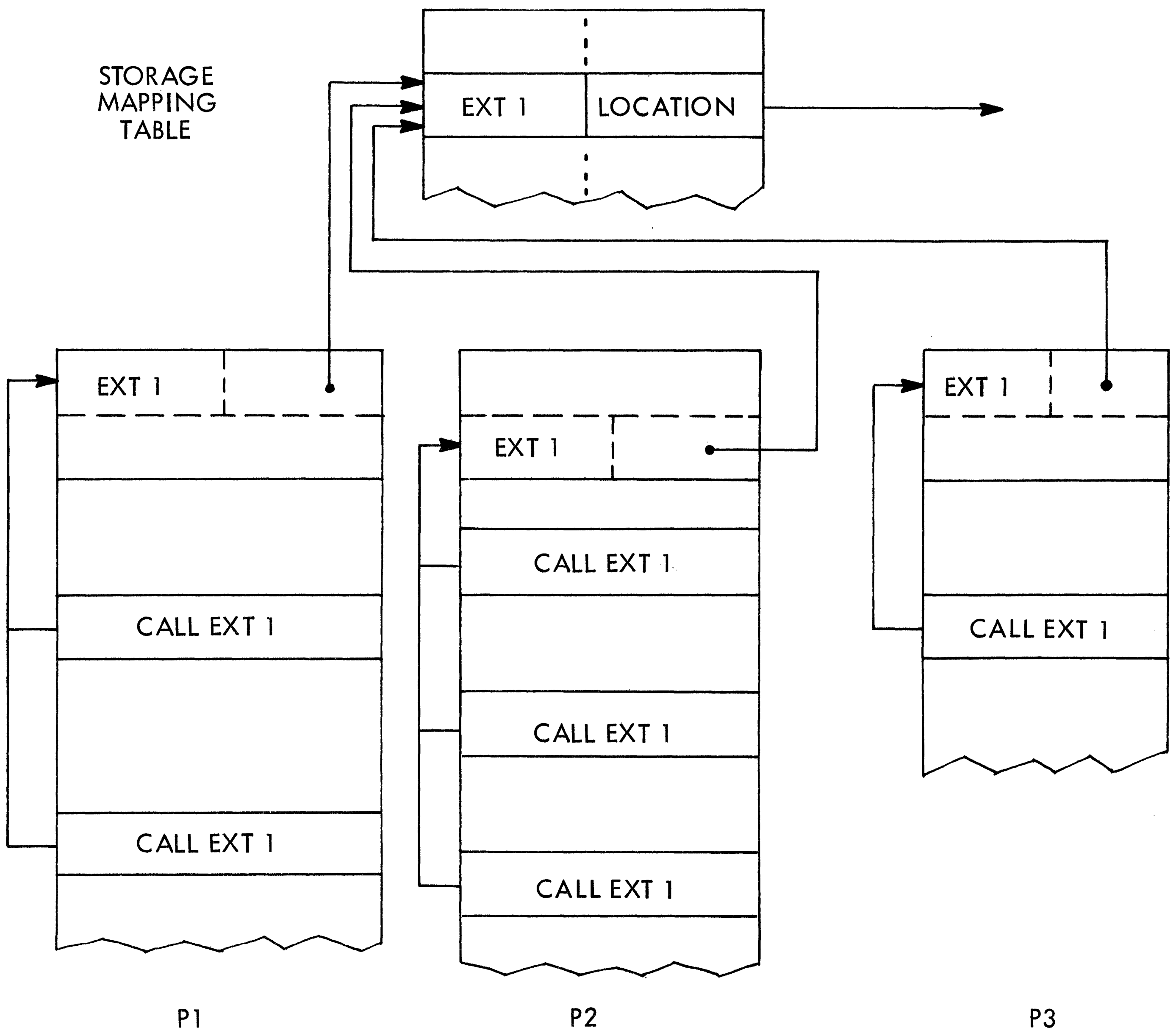
LINKAGE METHOD 1: DIRECT



LINKAGE METHOD 2: TRANSFER VECTOR



LINKAGE METHOD 3: EXECUTION MAPPING



FIELDS OF TYPICAL ASSEMBLER STATEMENT

- SYMBOLIC LOCATION NAME OR LABEL
- OPERATION
- OPERAND
- COMMENTS
- SERIAL IDENTIFICATION

RELOCATABLE SYMBOL RULES

REL: LABEL IN MACHINE ORDER OR LOCATION-DEFINING OPERATION
NONREL: LABEL IN NON-LOCATION-DEFINING OPERATION

REL + 5 ⟶ REL

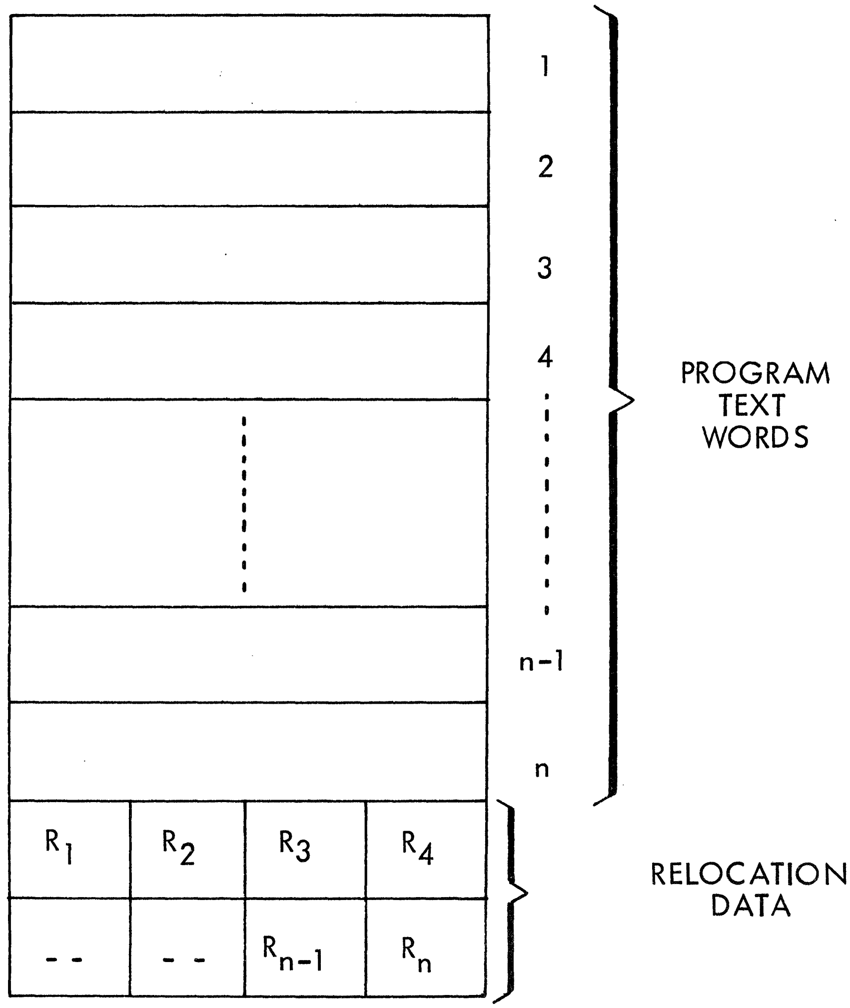
REL + NONREL ⟶ REL

REL + REL ⟶ NONREL

REL * REL ⟶ NONREL

ALPHA	CLA	B	(REL)
Z	EQU	7	(NONREL)
Y	EQU	ALPHA	(REL)
X	EQU	ALPHA+7	(REL)
W	EQU	ALPHA*Z	(NONREL)

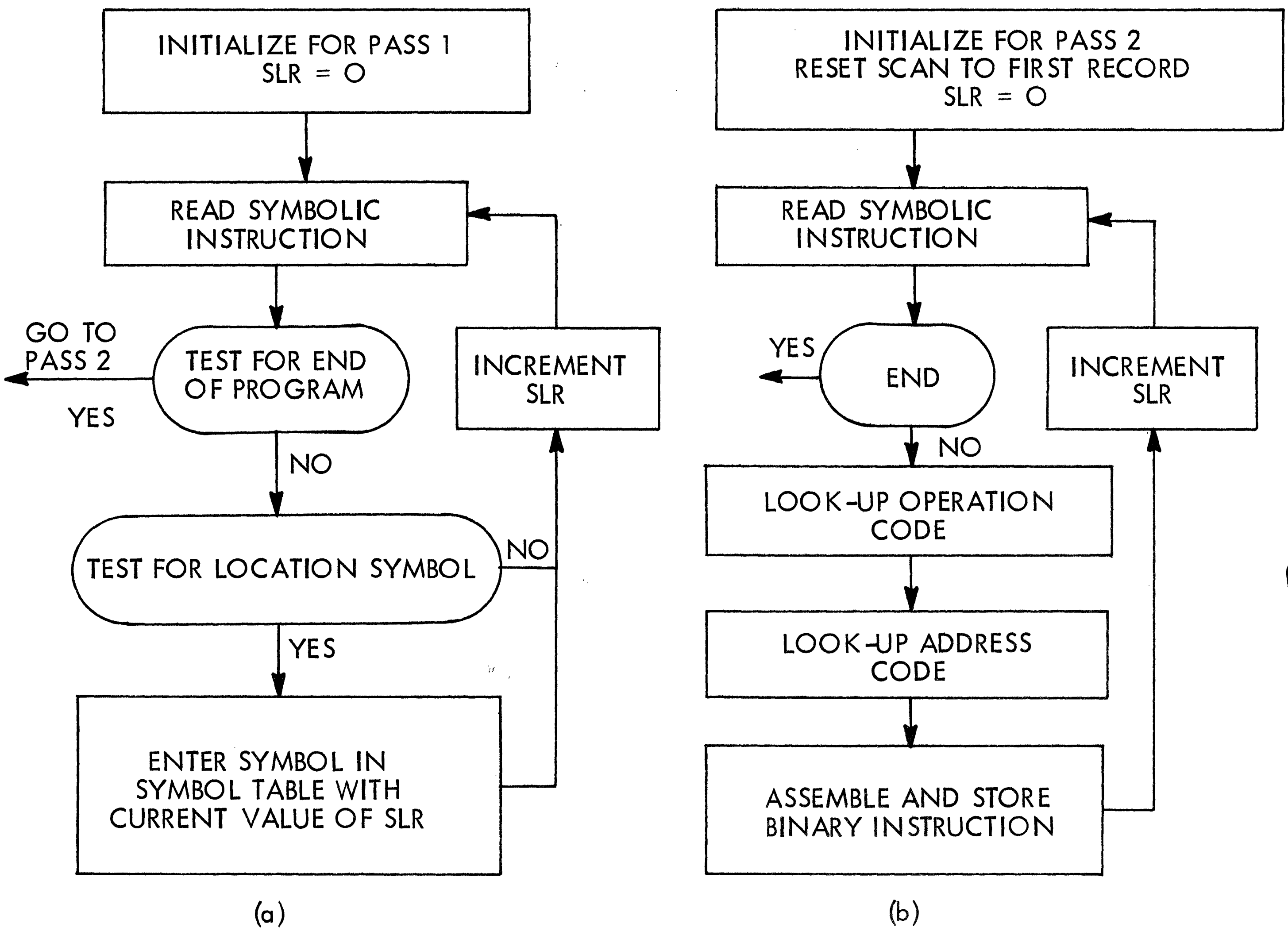
TEXT AND RELOCATION DATA



TYPES OF STATEMENTS IN A TYPICAL ASSEMBLER

- MACHINE INSTRUCTIONS
- DATA DEFINING PSEUDO-OPERATIONS
- "BUILT-IN" SYSTEM MACROS
- ASSEMBLER CONTROL PSEUDO-OPERATIONS
- CONDITIONAL AND ASSIGNMENT OPERATIONS
- MACRO DECLARATIONS AND CALLS

OPERATION OF A SIMPLE TWO-PASS ASSEMBLER (a) PASS 1; (b) PASS 2



DATA DEFINING PSEUDO-OPERATIONS:

- OCT
- DEC
- CHAR
- PREFIX CODES

BUILT-IN SYSTEM MACROS:

- CALL
- SAVE
- RETURN
- VARIOUS I/O OPERATIONS
- SUPERVISOR SERVICE REQUESTS

ASSEMBLER CONTROL PSEUDO-OPERATIONS:

- START
- END
- PRINT
- PUNCH
- ORG
- BSS
- USE
- ENTRY
- EXTERNAL

CONDITIONAL AND ASSIGNMENT STATEMENTS

S	SET	E	V(E) ⟶ V(S)
S	SET	1	1 ⟶ V(S)
S	SET	S + 1	V(S)+ 1 ⟶ V(S)

IF A,B,L

(IF V(A) = (V(B) THEN SKIP ASSEMBLY TO LOCATION L)

IFF A,B

(IF V(A) = V(B), THEN SKIP COUNTER BY 2)

CALCULATION OF N FACTORIAL BY ASSEMBLER

```
N      EQU        .
.       .         .
.       .         .
.       .         .
.       .
S      SET        1
K      SET        1
M      IF         S,N,L
S      SET        S + 1
K      SET        K * S
       GO TO      M
L      CONTINUE   :
.       :         .
.
```

S	K
1	1
2	2
3	6
4	24
5	120

MACRO DEFINITION AND CALL

```
SUM      MACRO    A,B,C
         LDA      A
         ADD      B
         STO      C
         ENDM

ALPHA    SUM      ADDEND, AUGEND, TOTAL

ALPHA    LDA      ADDEND
         ADD      AUGEND
         STO      TOTAL
```

ITERATIVE REPEAT FUNCTION

```
SUM      MACRO    A,B,C
         LDA      A
         ADD      B
         IRP      C
         STO      C
         IRP
         ENDM

ALPHA    SUM      X,Y, (Z1,Z2,Z3)

ALPHA    LDA      X
         ADD      Y
         STO      Z1
         STO      Z2
         STO      Z3
```

DEFINITION OF MULTIPROGRAMMING

THE TIME SHARING OF A CPU BY THE SEQUENTIAL OPERATION OF MULTIPLE PROGRAMS.

ORIGINS OF MULTIPROGRAMMING

- CPU TIME < < I/O TIME
- VISIBLY SLOW EARLY MACHINES
- INTRODUCTION OF LARGER (E.G. 32K) MEMORIES

FUNCTION OF SIMPLE MULTIPROGRAMMING SUPERVISOR

- DECIDE THE ORDER OF EXECUTION AMONG RESIDENT JOBS BASED ON:

  AVAILABILITY OF DATA AND FACILITIES

  THE PRIORITY OF THE JOB

  RELATIVE PRIORITIES OF OTHER JOBS

TYPES OF MULTIPROGRAMMING

JOB MIX / MEMORY ALLOCATION	FIXED CONTENT	FIXED NUMBER	VARIABLE NUMBER AND CONTENT
FIXED PARTITIONS	X	X	
VARIABLE—STATIC		X	X
VARIABLE—DYNAMIC			X

FIXED PARTITION – FIXED CONTENT

- EARLIEST MULTIPROGRAMMING
- IN EFFECT COMBINED TWO PROGRAMS INTO ONE; PROGRAMS SHIFTED CONTROL BACK AND FORTH.
- CHOICE OF PROGRAMS CRITICAL – ONE 'COMPUTATIONAL,' ONE I/O BOUND
- NO INTERNAL SCHEDULING – COOPERATIVE CONTROL

FIXED PARTITION FIXED NUMBER

- MODEL FOR PRESENT 360 DOS
- EARLY EMPHASIS ON MIX (E.G. COMPUTATIONAL AND I/O)
- IN PRINCIPLE ANY PROGRAM CAN BE RUN AS LONG AS IT FITS PARTITION
- USES EXECUTIVE TO SCHEDULE CPU TIME ON (POTENTIALLY)

  POSITION IN MEMORY
  I/O ACTIVITY
  PRIORITY
- MINIMUM USEFUL LEVEL OF MULTIPROGRAMMING

VARIABLE–STATIC FIXED NUMBER

- ALMOST COMPLETE – FIXED NUMBER OF PROGRAM ESTABLISHED TO FIX SIZE OF OP. SYSTEM TABLES

- SEQUENCING THROUGH RESIDENT PROGRAMS

  ROUND–ROBIN
  FIFO
  PRIORITY
  JOB LIST POSITION–DEPENDENT
  TIMER LIMITATIONS

- INTRODUCES MEMORY MANAGEMENT PROGRAMS

  COMPACTING FOR FREE SPACE
  ALLOCATION MADE AT LOAD TIME
  PERMITS QUEUEING JOBS ON SECONDARY STORAGE

VARIABLE—STATIC VARIABLE NUMBER AND CONTENT

- SIMILAR CAPABILITIES AS WITH FIXED NUMBER — MAY BE ABLE TO GET SOME FEW MORE PROGRAMS IN.

- REQUIRES MEMORY ALLOCATION FOR OPERATING SYSTEM AS WELL.

VARIABLE—DYNAMIC VARIABLE NUMBER AND CONTENT

- MODEL FOR MOST 'LARGE SCALE' MULTIPROGRAMMING SYSTEMS
- PERMITS RUN—TIME ALLOCATION OF MEMORY FOR HANDLING COMPLEX PROGRAM STRUCTURES
- PERMITS RUN—TIME COLLECTION AND BINDING OF PROGRAMS
  - FORK
  - JOIN

OTHER MULTIPROGRAMMING OPERATING SYSTEM ISSUES

- CONTROL INTERPRETERS

- RESOURCE ALLOCATION FOR QUEUED JOBS
    - PERIPHERAL DEVICES
    - MEMORY
    - RESERVATION TECHNIQUES

CONTRIBUTIONS TO MACHINE ORGANIZATION

- HONEYWELL 800

- BASE REGISTER CONCEPT

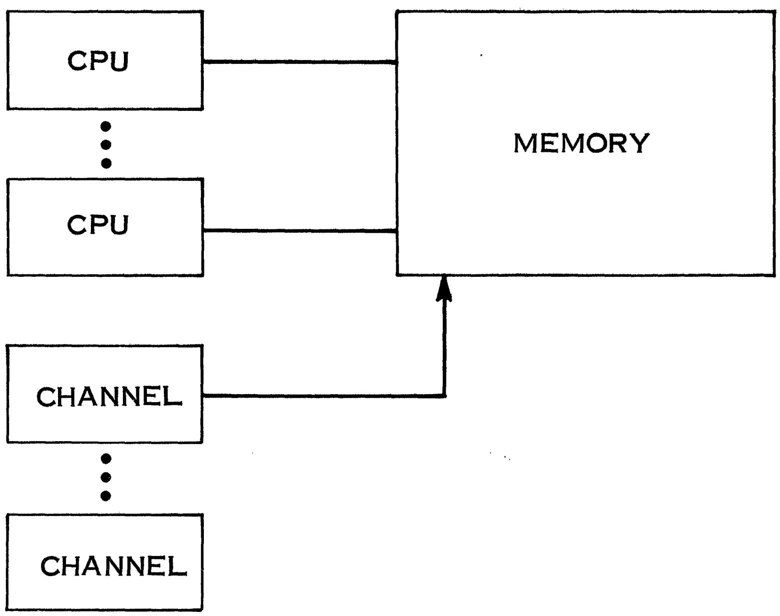
THREE STAGES OF MULTIPROCESSOR DEVELOPMENT

1. HIGHER PERFORMANCE SYSTEMS THROUGH CONCURRENT PROCESSING

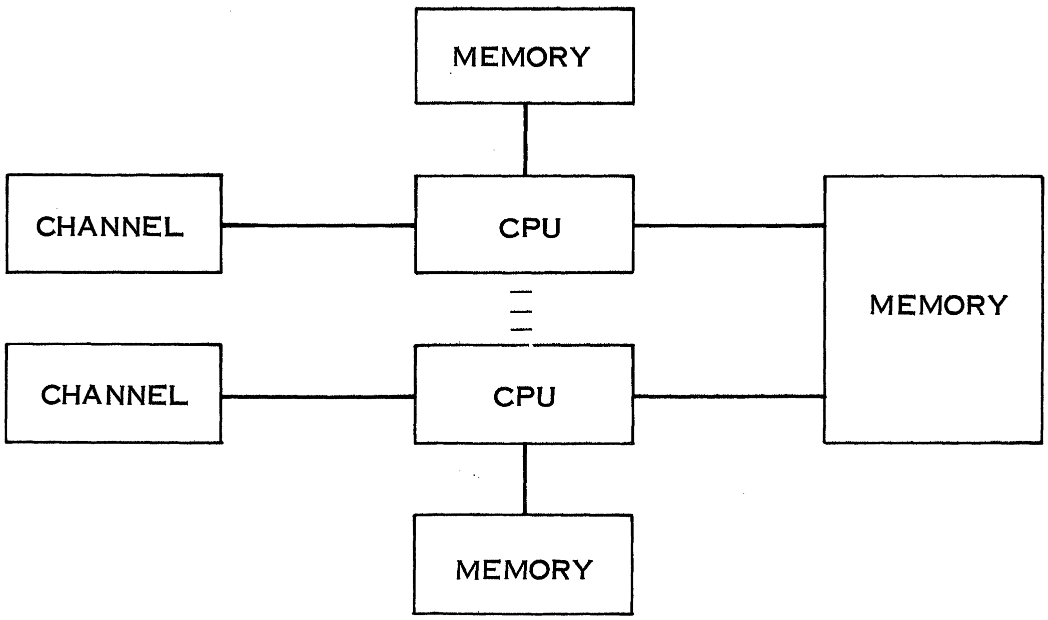
2. HIGH RELIABILITY SYSTEMS

3. IMPROVED PERFORMANCE AND SYSTEMS BALANCE

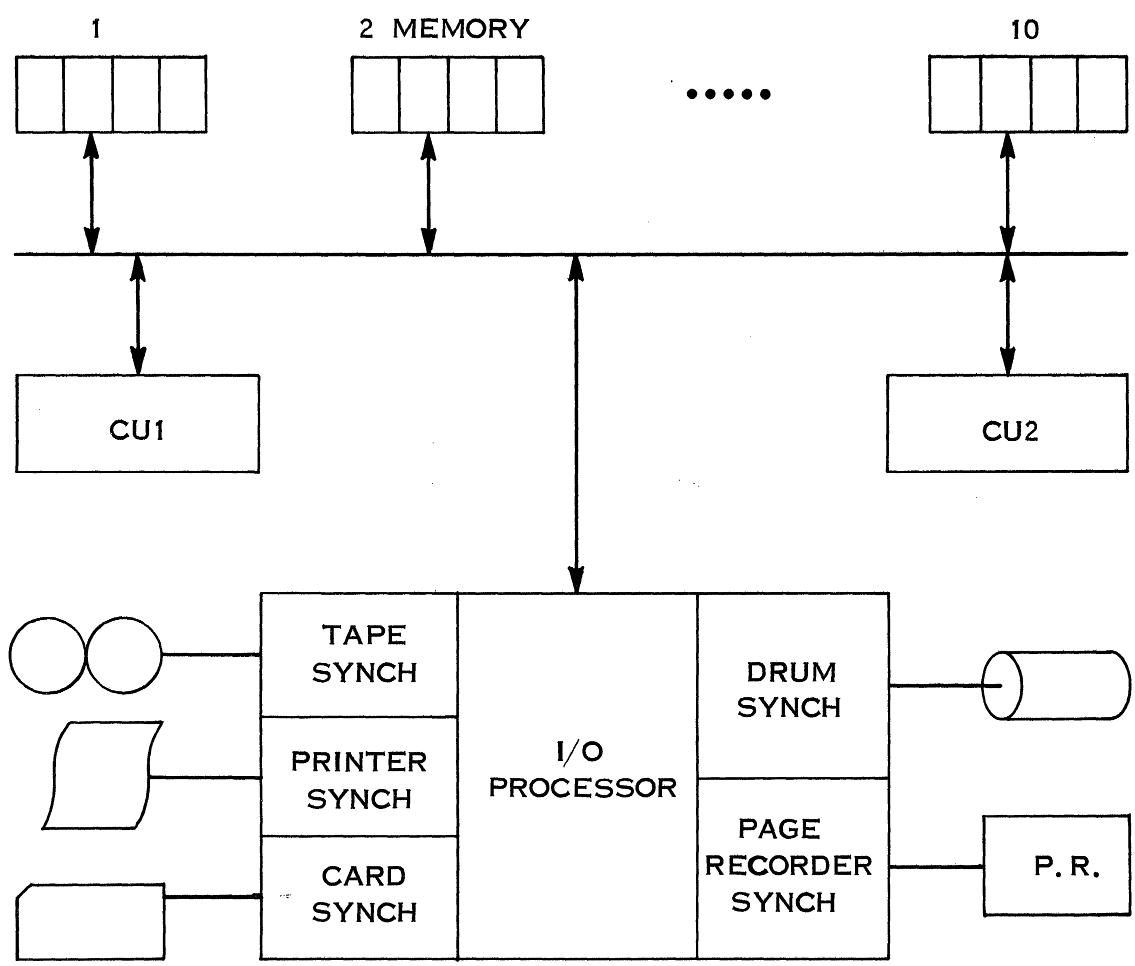
MULTIPROCESSOR



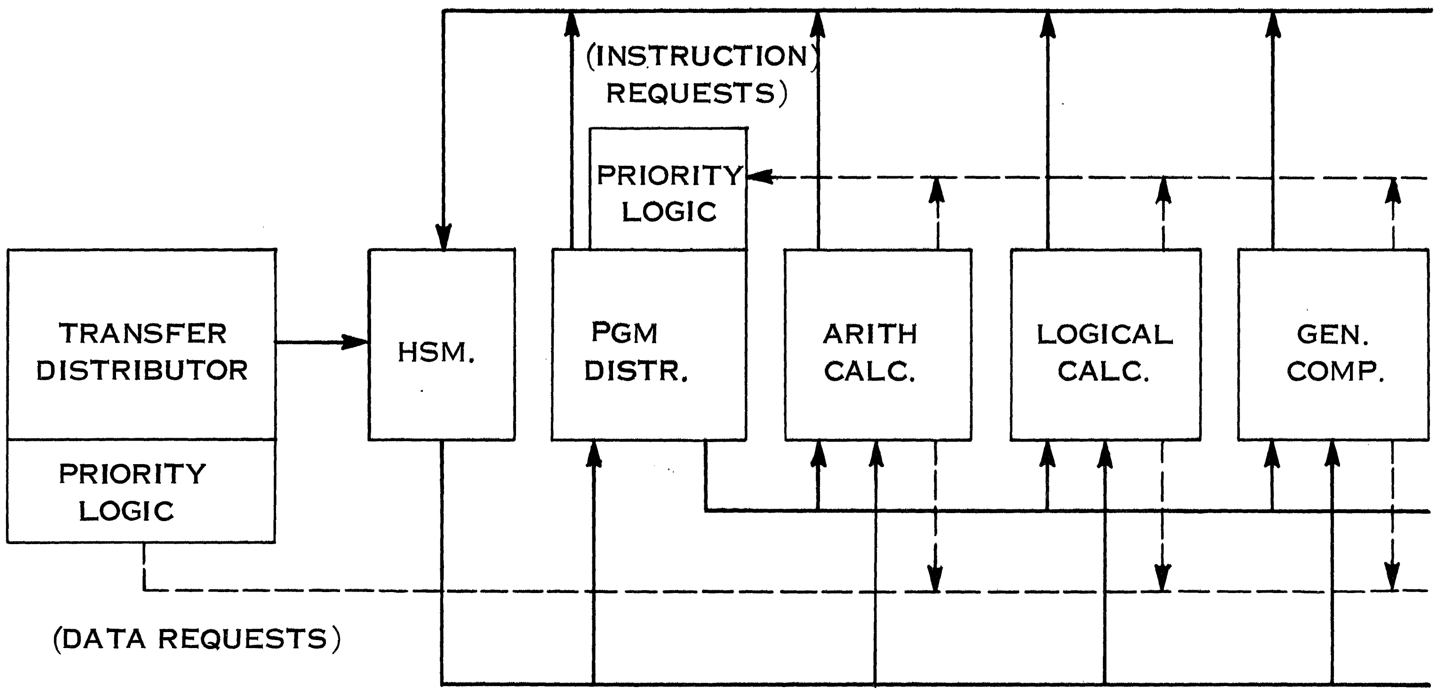
MULTICOMPUTER



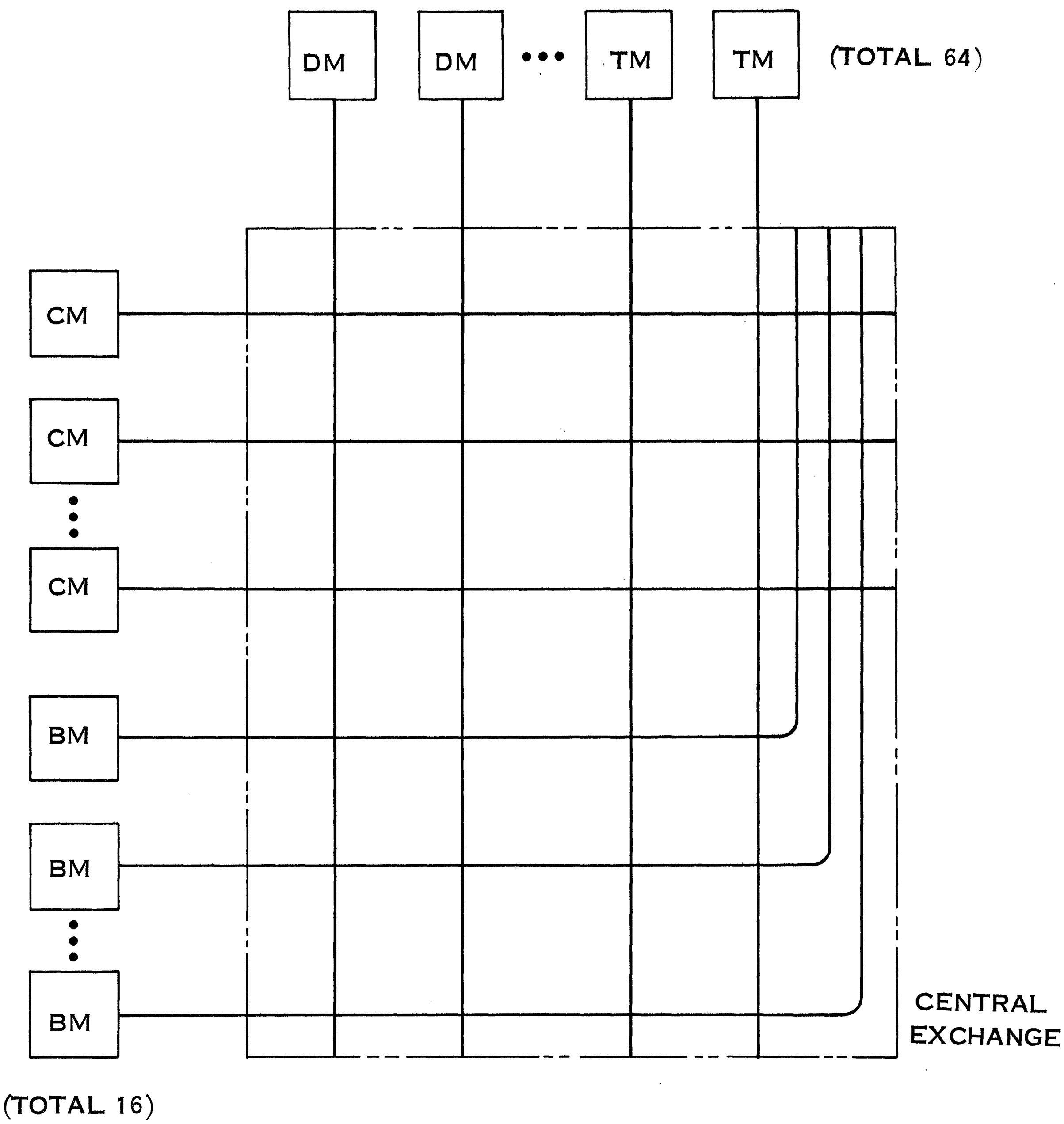
UNIVAC LARC


1
2 MEMORY
10
CU1
CU2
TAPE SYNCH
PRINTER SYNCH
CARD SYNCH
I/O PROCESSOR
DRUM SYNCH
PAGE RECORDER SYNCH
P. R.

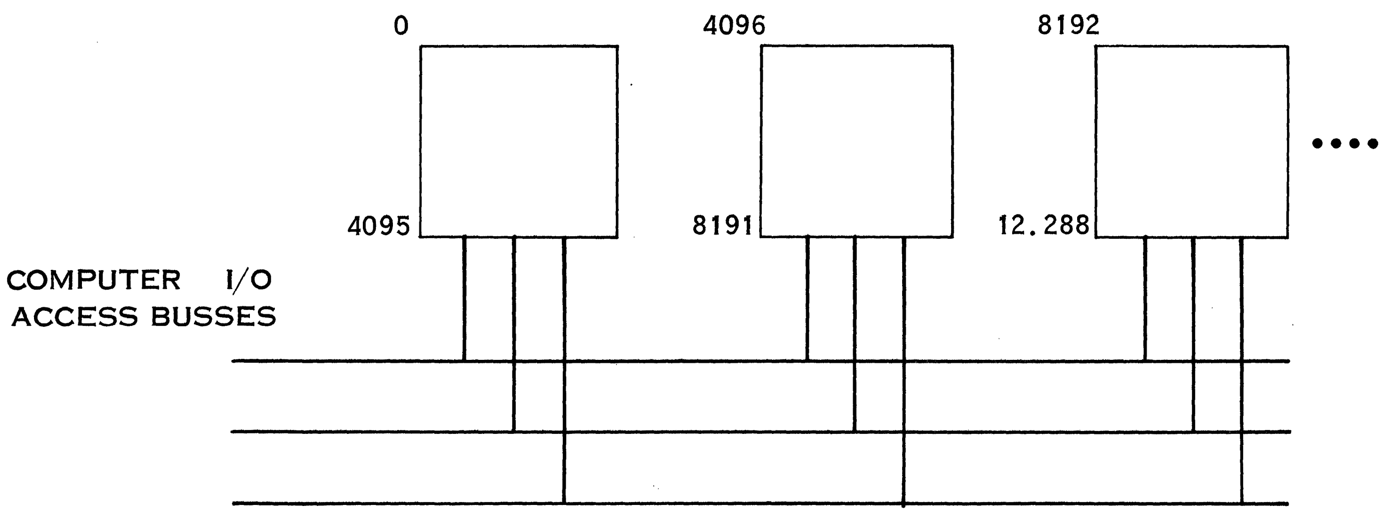
GAMMA 60


(INSTRUCTION)
REQUESTS)
PRIORITY
LOGIC
TRANSFER
DISTRIBUTOR
PRIORITY
LOGIC
HSM.
PGM
DISTR.
ARITH
CALC.
LOGICAL
CALC.
GEN.
COMP.
(DATA REQUESTS)

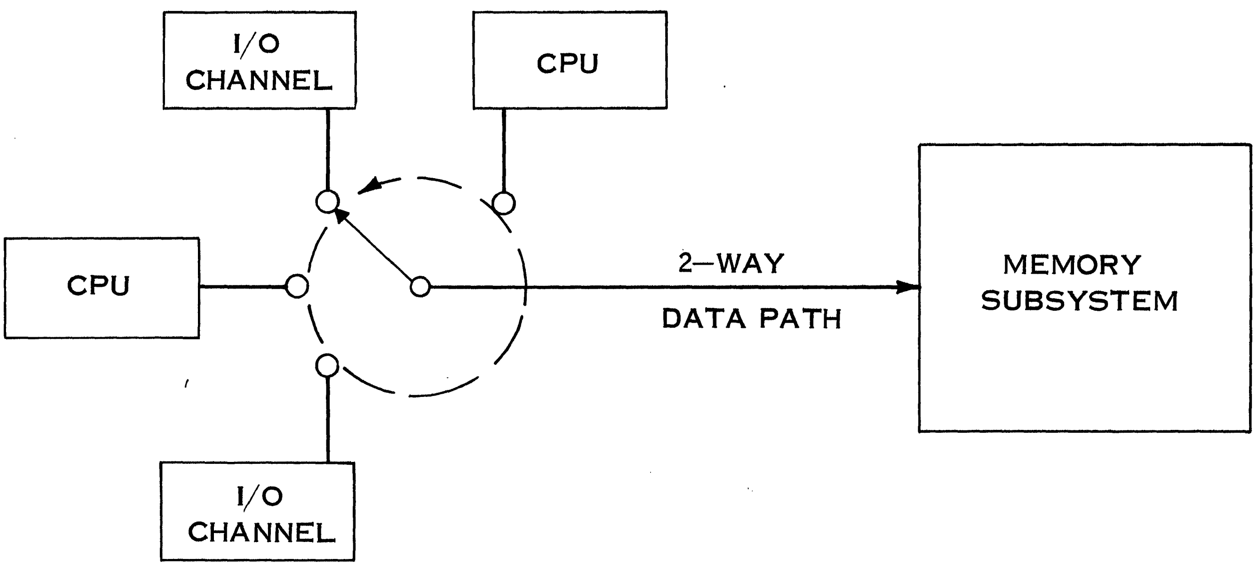
TRW—400



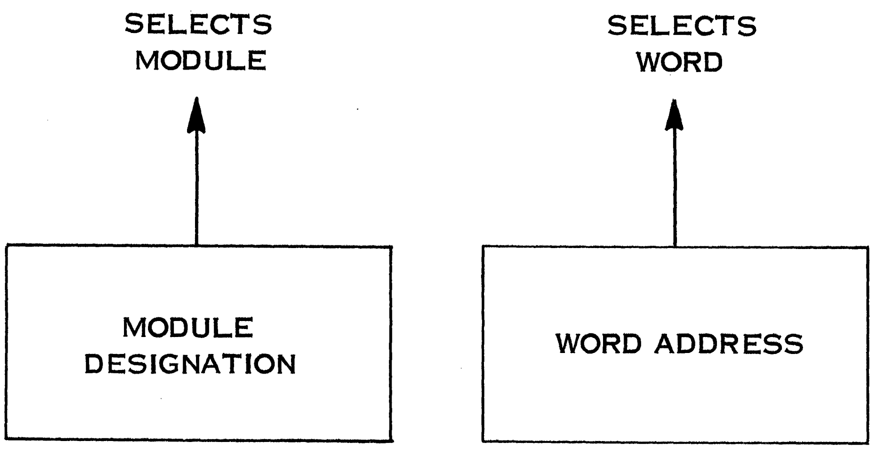
MULTIPROCESSOR MODULAR MEMORY



TIME-SLOTTED BUS



BANK SWITCHING (3600)

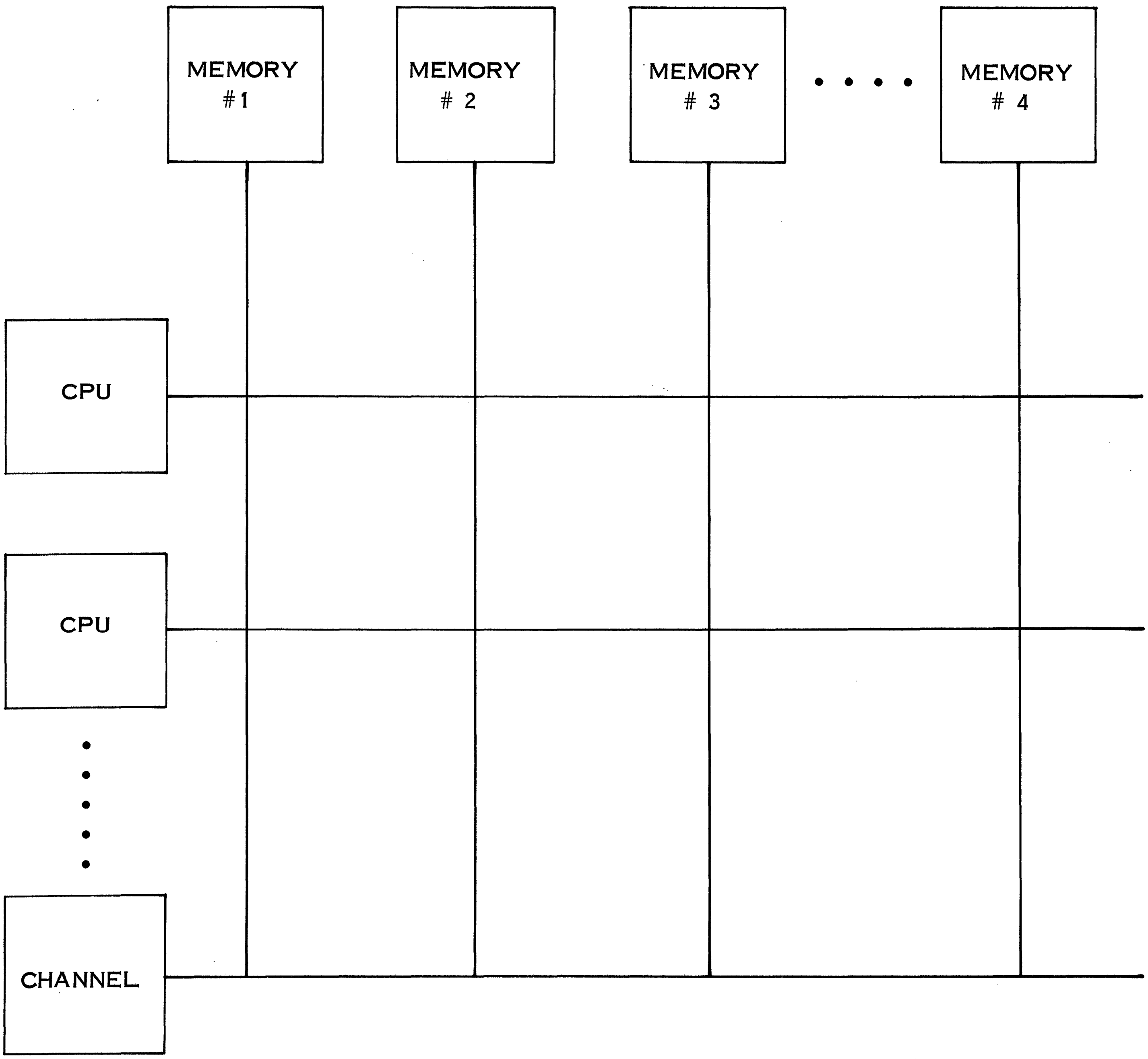


ACCESS DISTRIBUTION ON LARC BUS

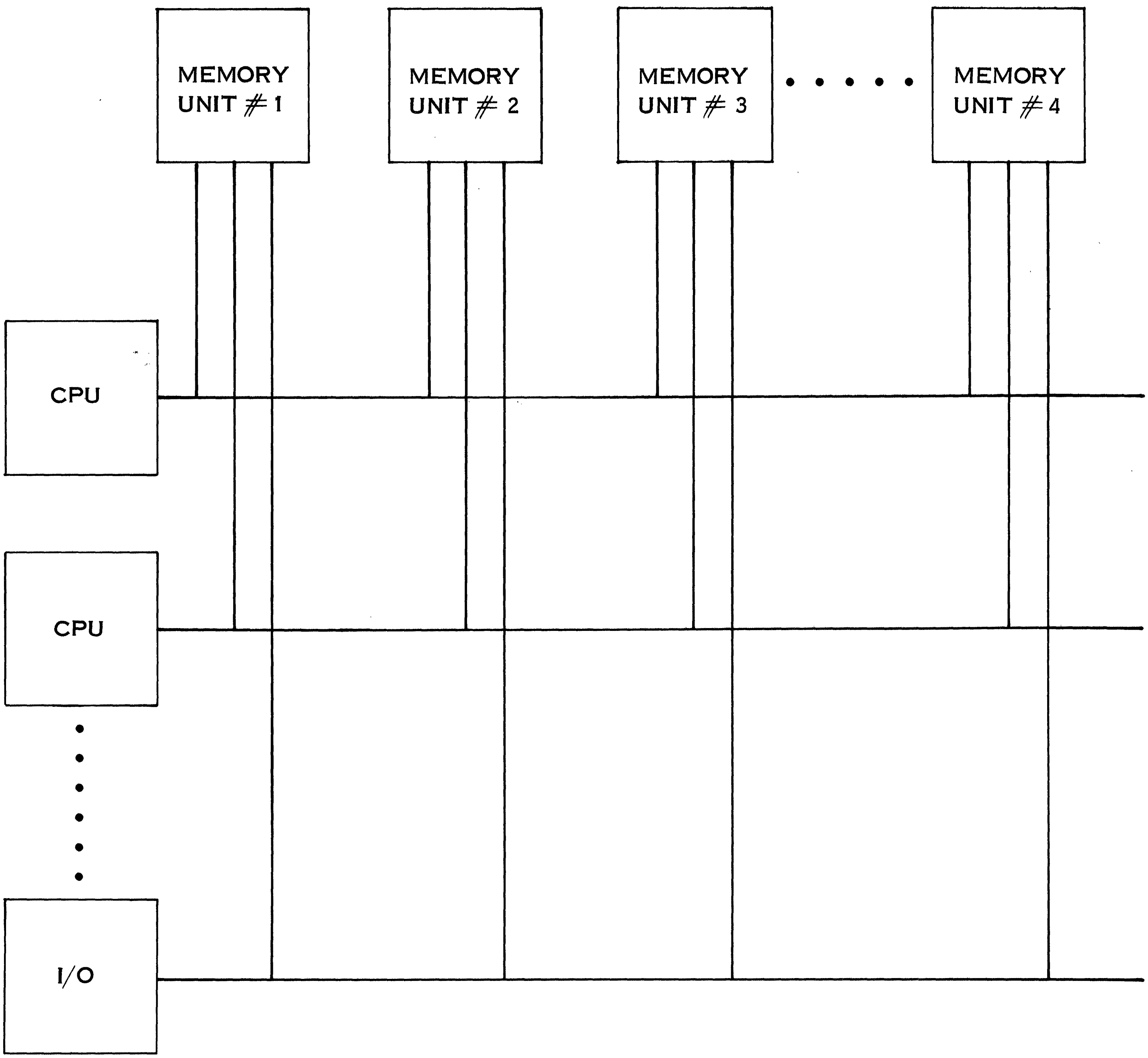
4 μS

I/OP INST. OR OPERAND ACCESS	COMP 1 INST. ACCESS	COMP 2 OPERAND ACCESS	I/O P DISPACT. ACCESS	NOT USED	COMP 2 INST. ACCESS	COMP 1 OPERAND ACCESS	I/O DISP. ACCESS
← .5 μS →							

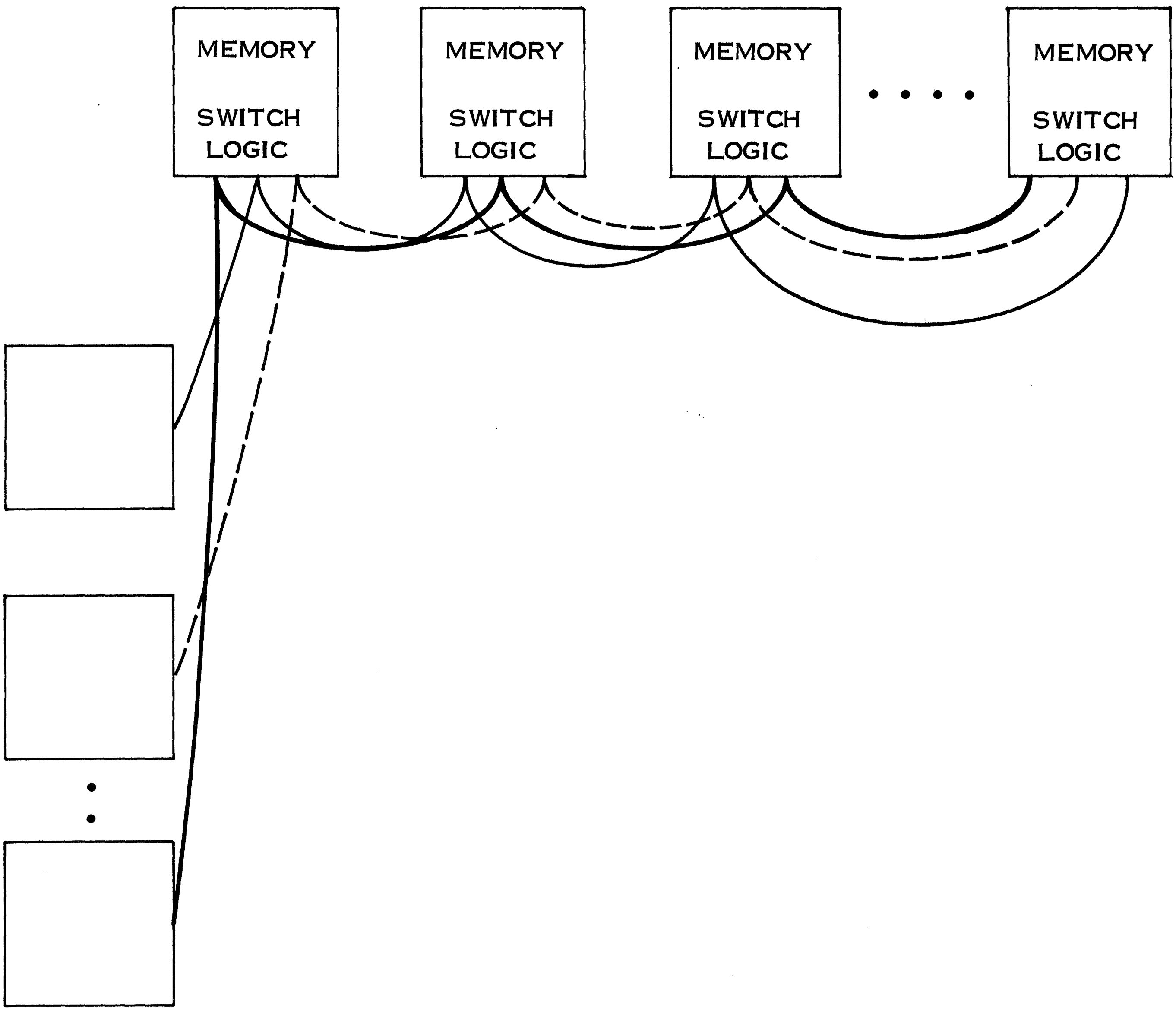
CROSS BAR SWITCHED MEMORY



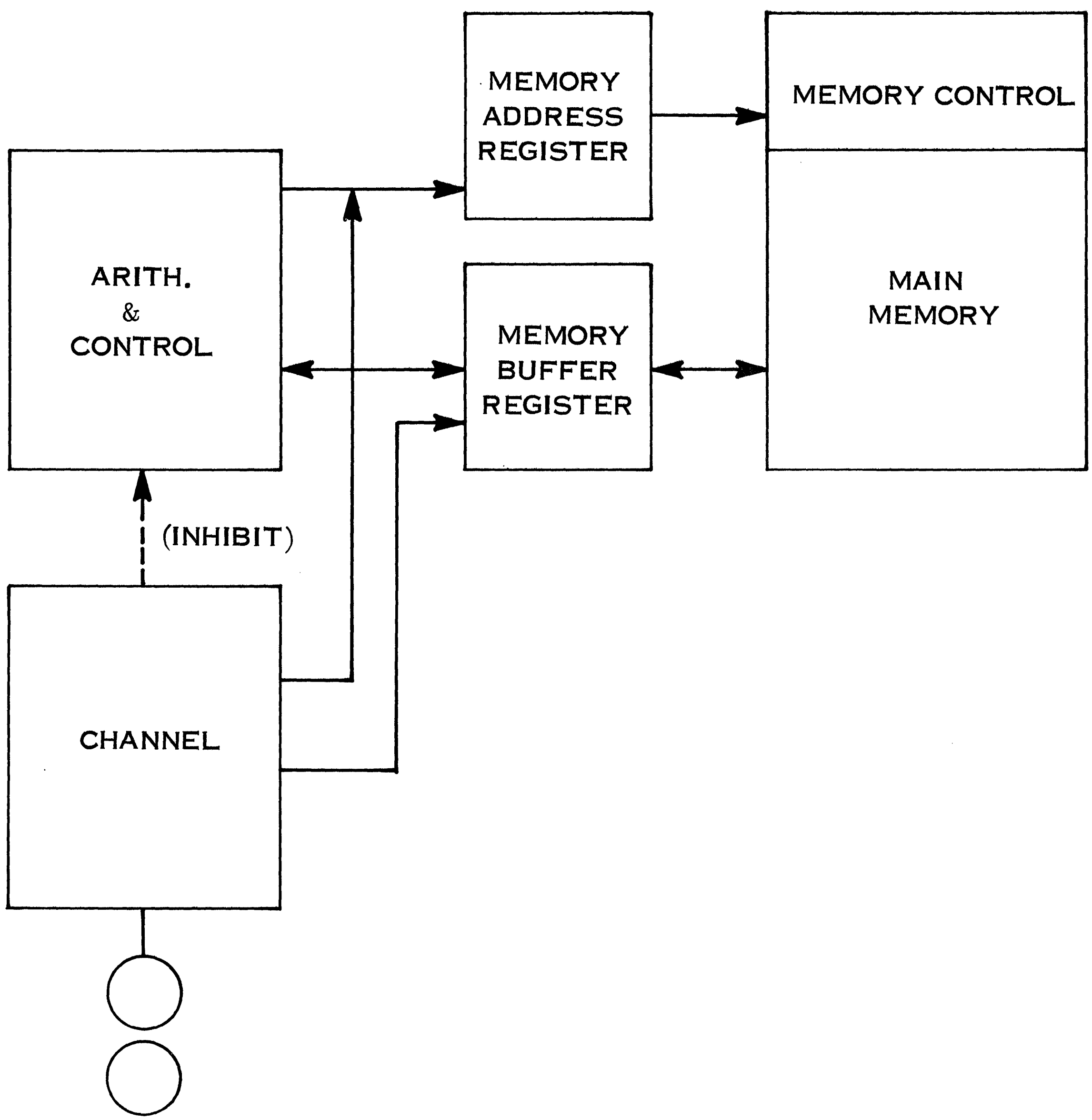
CROSS BAR SWITCH MEMORY (SHOWING UNIQUE CONNECTION TO EACH PROCESSOR BUS)



"DISTRIBUTED" CROSS BAR SWITCH



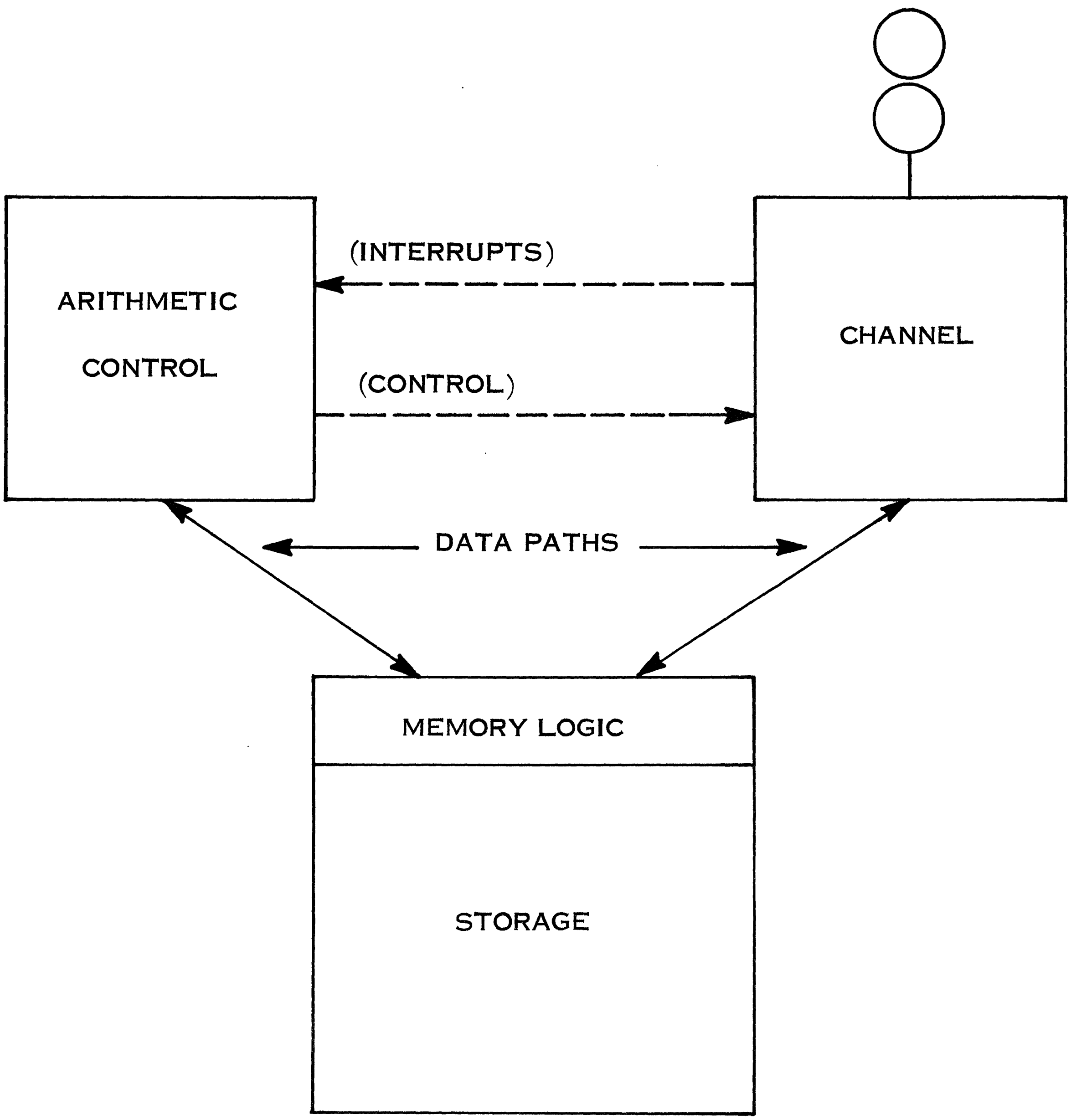
CHANNEL SHARING MEMORY CIRCUITS OF CPU



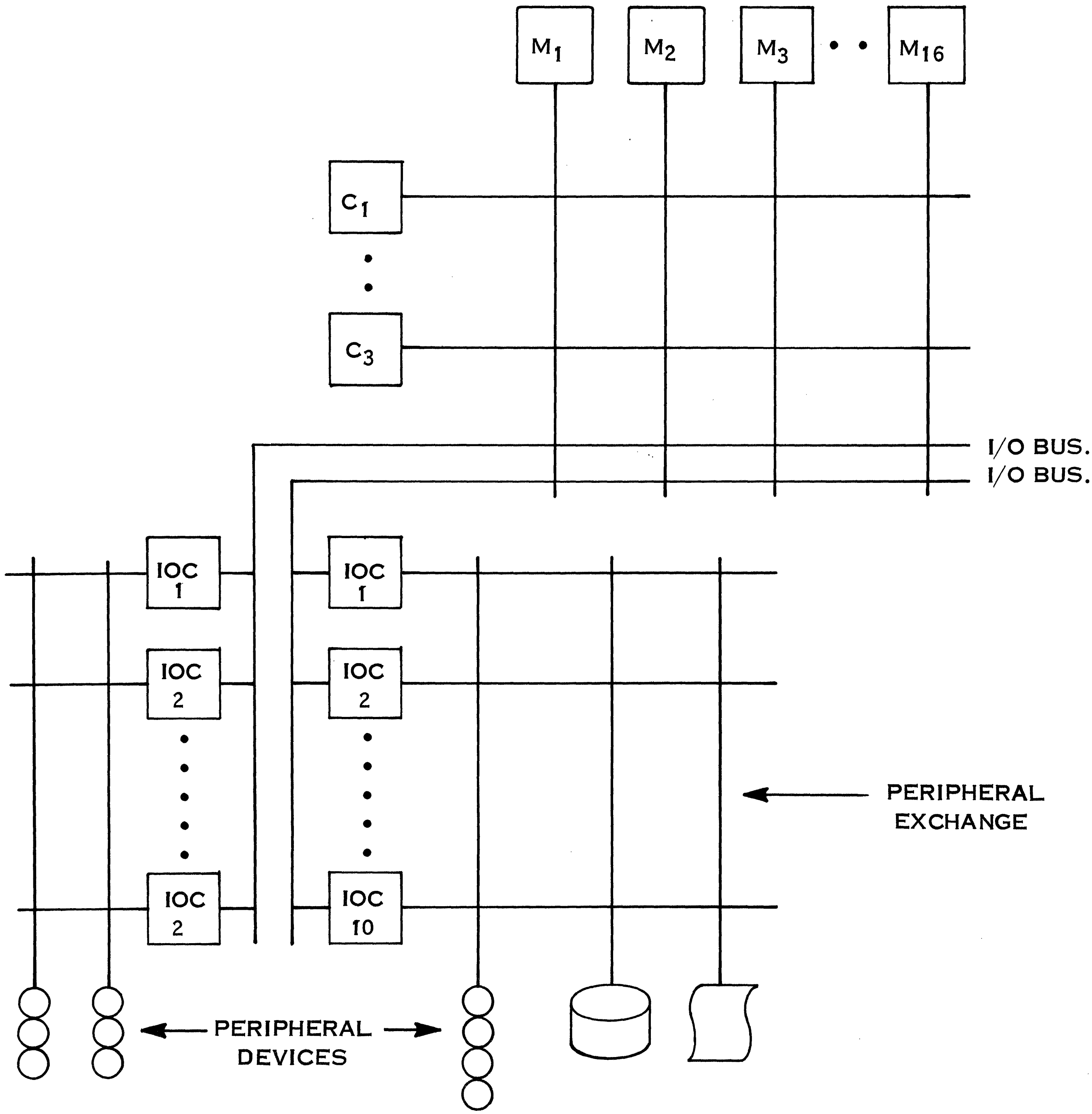
REPRESENTATIVE MEMORY MODULE SIZES

MACHINE	SIZE	MAXIMUM PERMITTED IN SYSTEM
1108	32K WORDS	8
360/65, 67	256K BYTES. (32K WORDS)	8
625/35/45	32K OR 64K	8 (4)

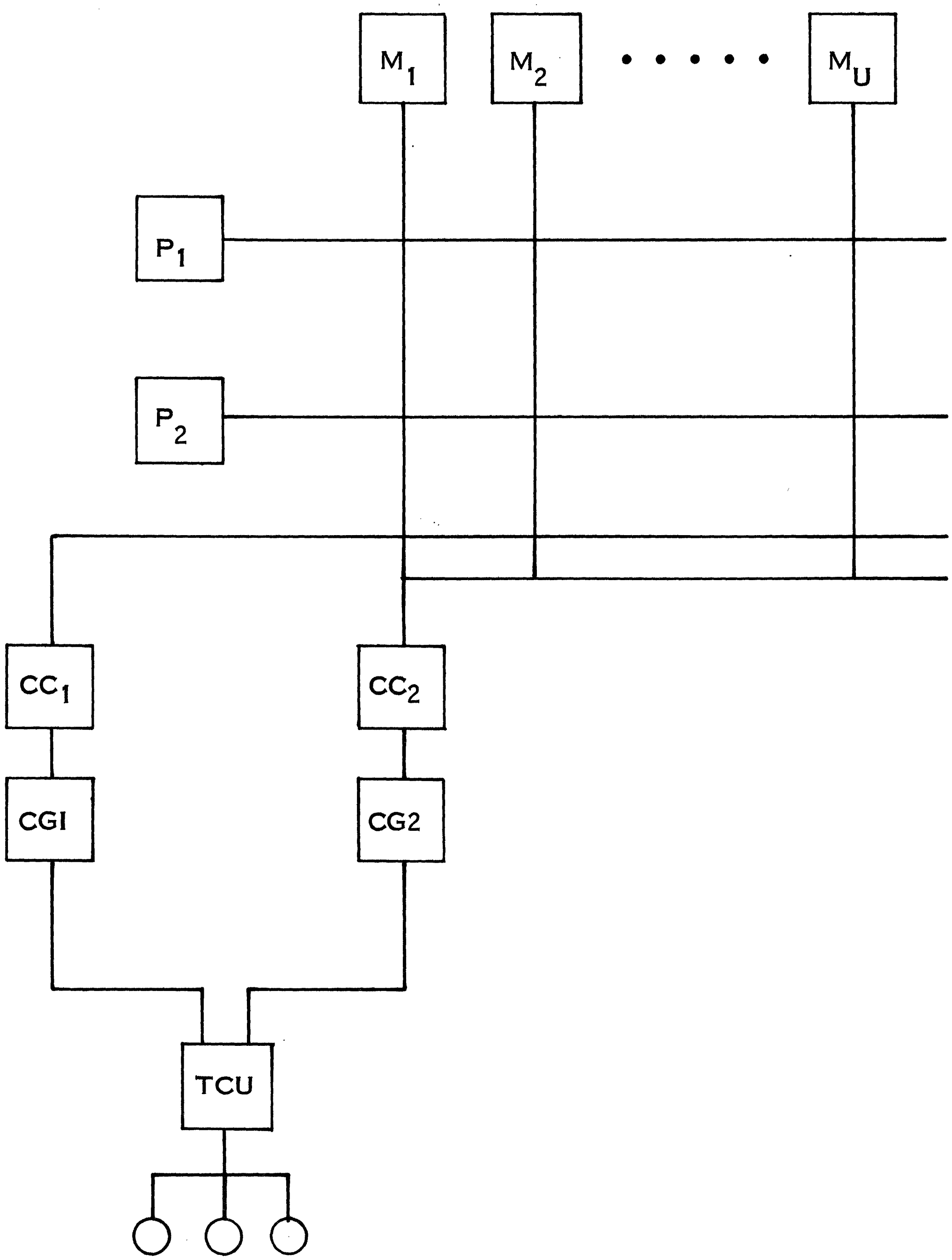
INDEPENDENT I/O CHANNEL



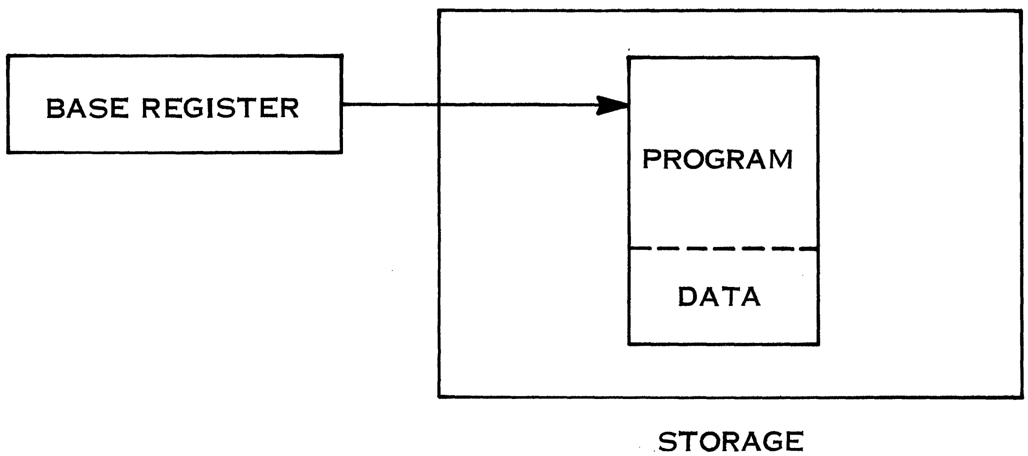
D825 CHANNEL ARRANGEMENT



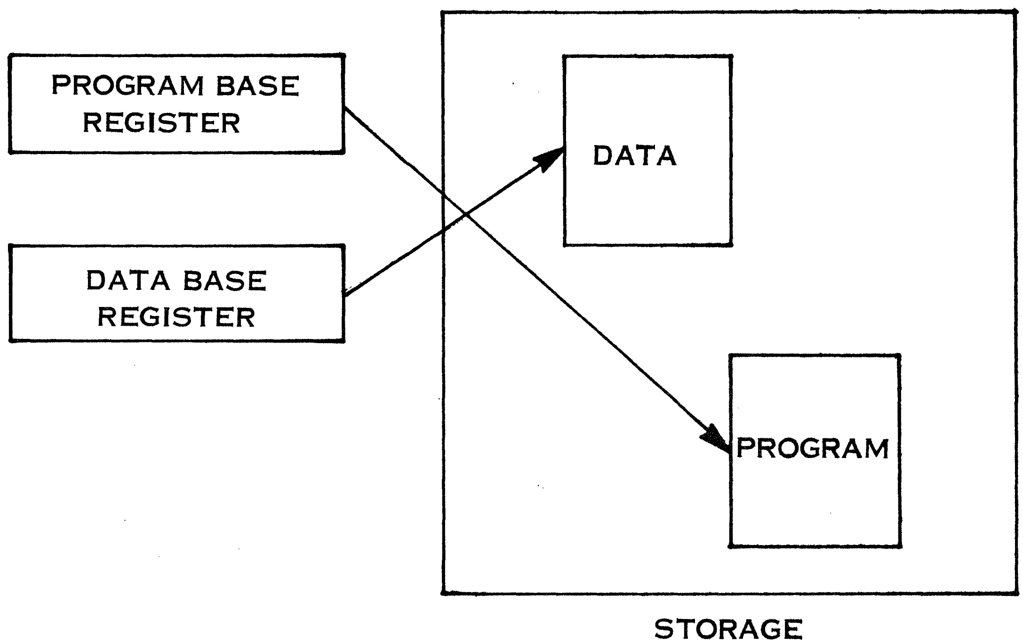
360/67 CHANNEL ARRANGEMENT
(SIMPLIFIED)



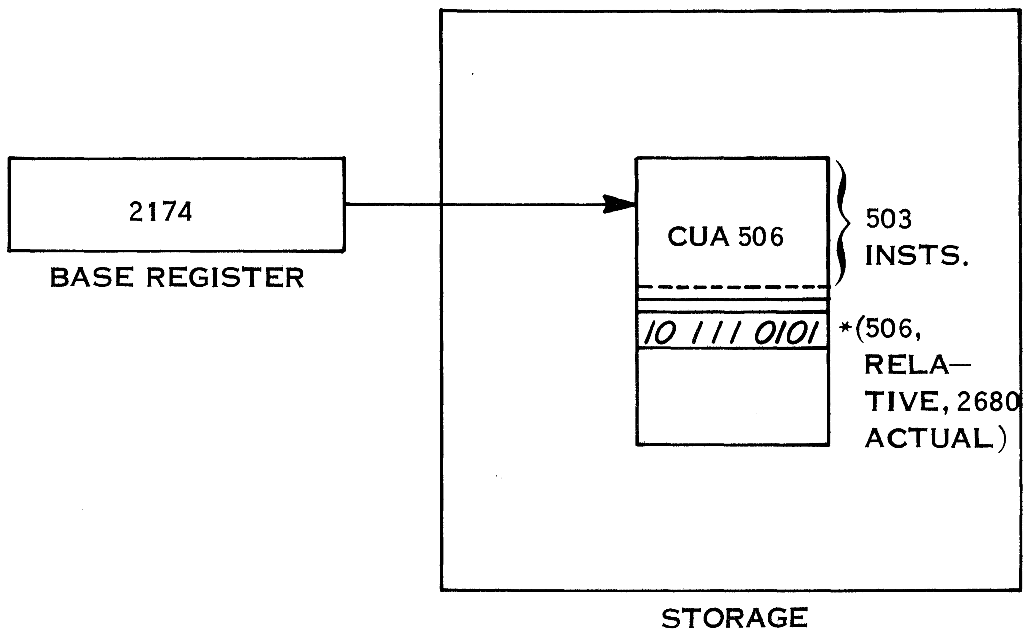
SINGLE BASE REGISTER AND MEMORY ALLOCATION



SEPARATE PROGRAM AND DATA BASE REGISTERS



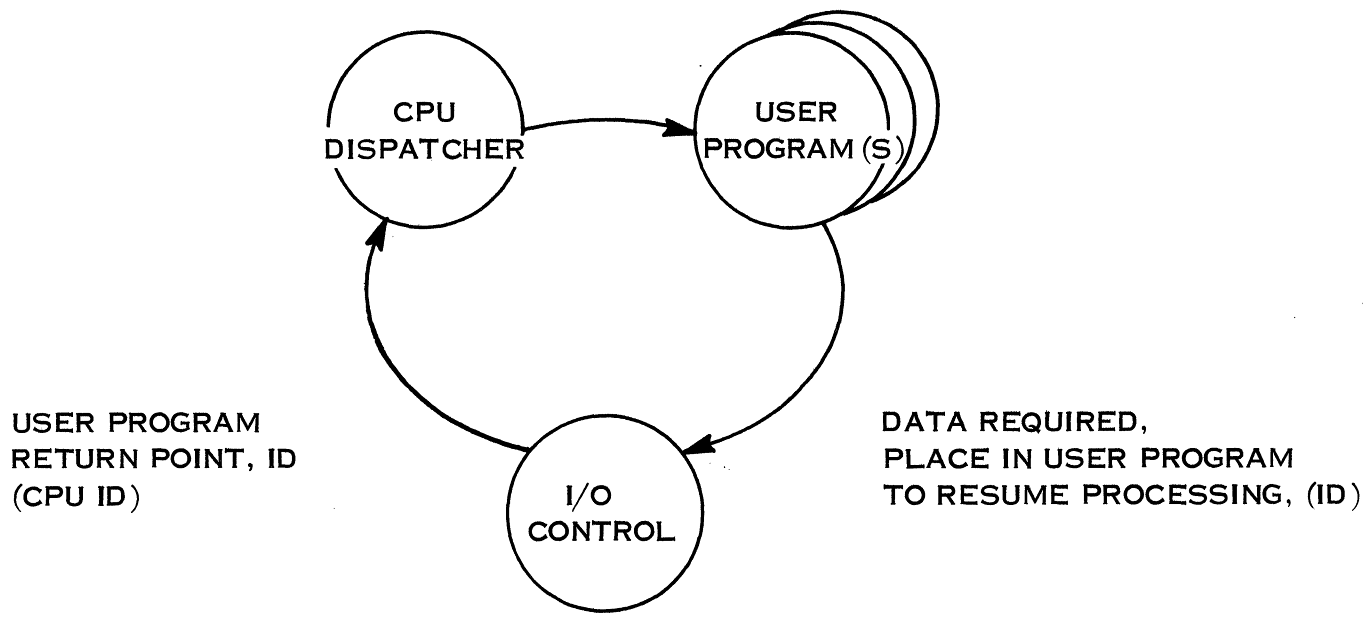
BASE REGISTER ADDRESSING



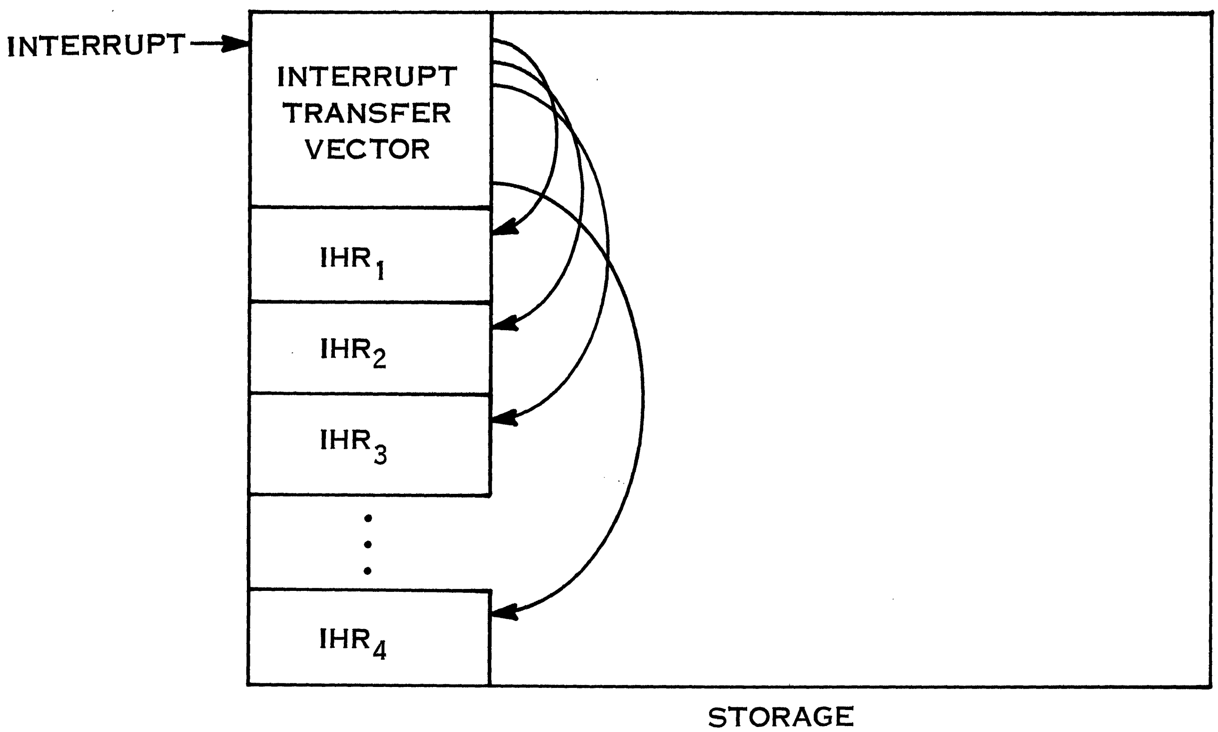
360.RX INSTRUCTION

OP CODE	R_1	X_2	B_2	D_2

EVENT PROPAGATION IN AN OPERATING SYSTEM



TYPICAL UNIPROCESSOR INTERRUPT IMPLEMENTATION



INTERRUPT CLASSES ON 360

MACHINE CHECK	(FAULT)
EXTERNAL	
SUPERVISOR CALL	
PROGRAM	(FAULT)
I/O	

360 CHANNEL STATUS WORD

KEY	0000	COMMAND ADDRESS

0 3 7 31

STATUS	COUNT

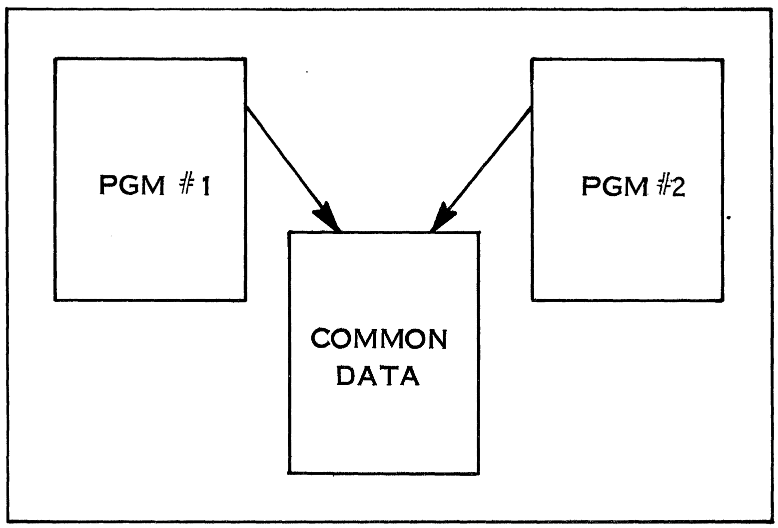
32 47

32	ATTENTION	40	PROGRAM–CONTROLLED INTERRUPT
33	STATUS MODIFIER	41	INCORRECT LENGTH
34	CONTROL UNIT END	42	PROGRAM CHECK
35	BUSY	43	PROTECTION CHECK
36	CHANNEL END	44	CHANNEL DATA CHECK
37	DEVICE END	45	CHANNEL CONTROL CHECK
38	UNIT CHECK	46	INTERFACE CONTROL CHECK
39	UNIT EXCEPTION	47	CHAINING CHECK

METHODS OF PROGRAM & DATA PROTECTION

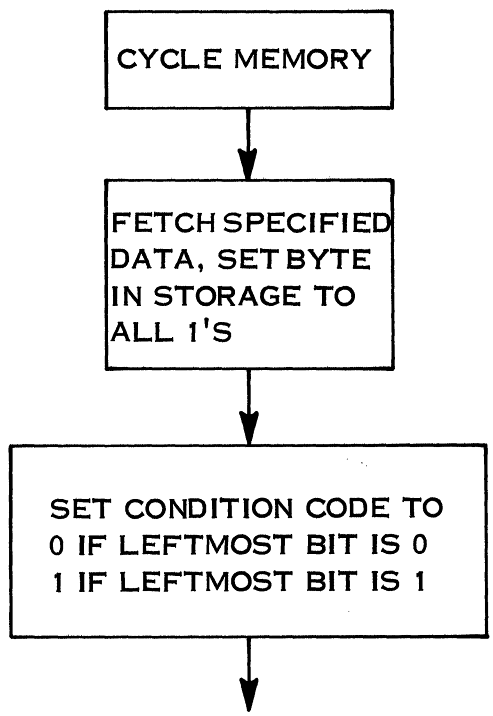
- BOUNDS REGISTERS
- STORAGE LOCKS

COMMON DATA IN A REAL–TIME APPLICATION

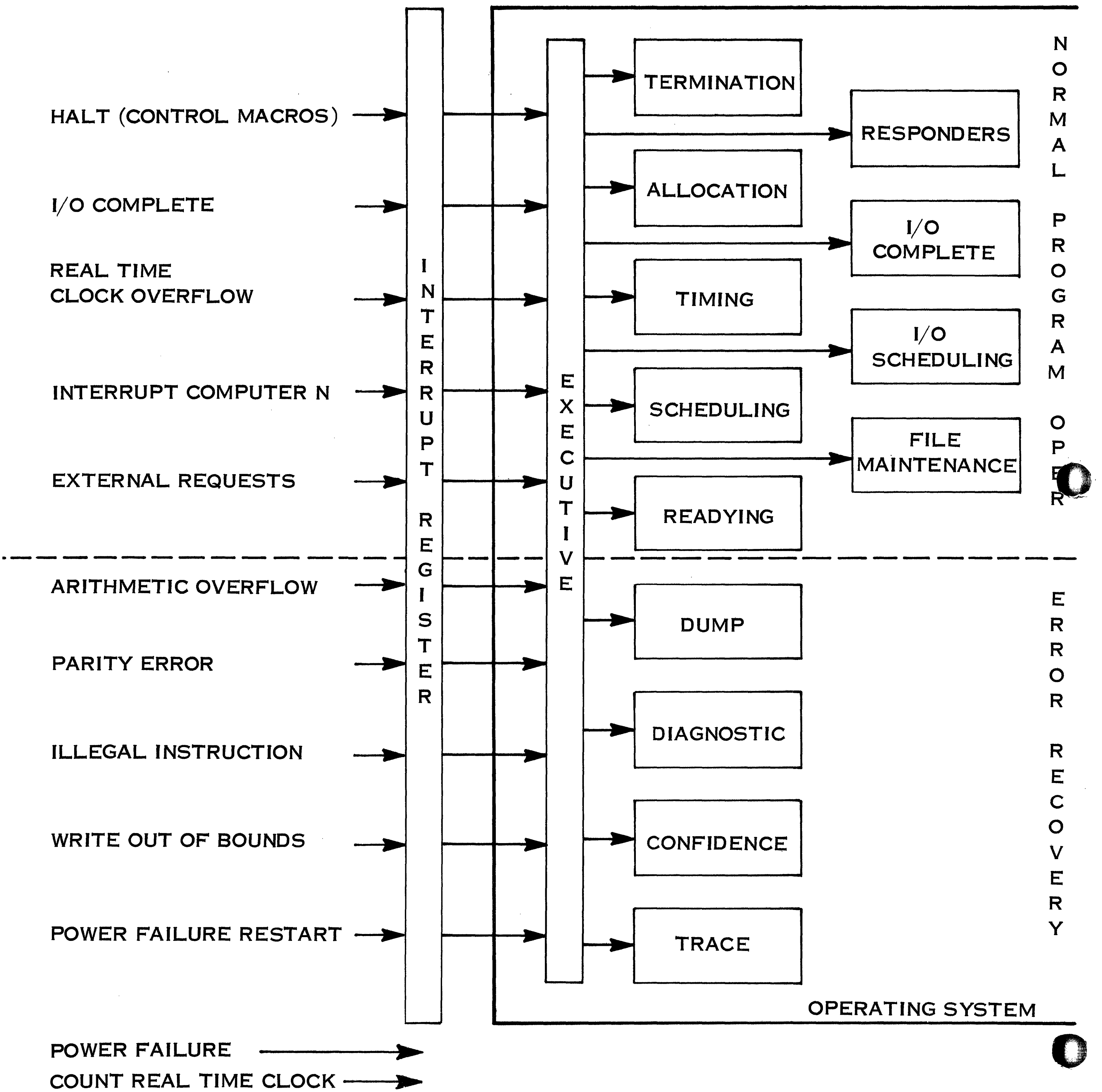


STORAGE

TEST AND SET


CYCLE MEMORY
FETCH SPECIFIED DATA, SET BYTE IN STORAGE TO ALL 1'S
SET CONDITION CODE TO
0 IF LEFTMOST BIT IS 0
1 IF LEFTMOST BIT IS 1

STRUCTURE OF INTERRUPT DRIVEN OPERATING SYSTEM



FACTORS ENTERING INTO SYSTEM/360 DESIGN

- LARGE NUMBER OF EXISTING IBM INSTALLATIONS.
- COSTS OF MAINTAINING SEPARATE SOFTWARE SUPPORT FOR DIVERSE MACHINES.
- IMPACT OF HONEYWELL AND CDC.
- DESIRE TO CONSOLIDATE ALL LINES.
- BREAKDOWN OF SCIENTIFIC/BUSINESS DISTINCTION.
- GROWTH OF REAL—TIME APPLICATIONS.

SYSTEMS FAMILIES

704
|
709
|
7090
|
7094
|
7094II

1401
|
1410
|
1440/7010

1604
|
3600

- LARGELY COMPATIBLE AT MACHINE LANGUAGE LEVEL

METHODS FOR CONVERTING BETWEEN MACHINES

- SIMULATION
- RE–COMPILATION
- LANGUAGE TRANSLATORS
- SUB–MACHINES

MACHINE SIMULATION AS CONVERSION AID

- ATTEMPTS TO COPE WITH CONVERSION AT MACHINE—LANGUAGE LEVEL
- COMPLEX PROGRAM
- USUALLY CANNOT HANDLE I/O DIRECTLY
- RUNS 1/100 — 1/1000 SPEED OF MACHINE BEING SIMULATED
- PRACTICAL ONLY IF A VERY SMALL NUMBER OF INFREQUENTLY RUN PROGRAMS WILL BE RUN ON SIMULATOR
- MOST FREQUENTLY USED AS A DESIGN TOOL FOR NEW MACHINES
- NEW MACHINE(S) SIMULATED ON AN OLDER MACHINE.
- OTHER DIFFICULTIES
  - WORD SIZE COMPATIBILITY
  - SPECIAL INSTRUCTIONS (E.G., WORD MARK HANDLING ON 1401)
  - EASY TO OVERLOOK SUBTLE MACHINE FEATURES
  - LIMITS SIZE OF PROGRAM THAT CAN RUN.

RECOMPILATION AS CONVERSION AID

- ASSUMES ALL OF PROGRAMS WRITTEN IN POL
- ORIGINAL COMPILER CAN'T HAVE 'EXTENSIONS' NOT PRESENT IN SECOND COMPILER
- PROGRAM DOES NOT TAKE ADVANTAGE OF STRUCTURE OF ORIGINAL MACHINE
- IN GENERAL FEASIBLE ONLY IF LOWEST COMMON DENOMINATOR BETWEEN TWO COMPILERS WAS USED
- POL'S STILL NOT UNIVERSALLY IN USE
- SLOW DEVELOPMENT AND ACCEPTANCE OF LANGUAGE STANDARDS
- OBJECT PROGRAMS RUN AT TARGET MACHINE SPEED.

LANGUAGE TRANSLATOR AS CONVERSION AID

- WITH RECOMPILATION, MOST SUCCESSFUL.
- CAN OPERATE AT MACHINE–LANGUAGE OR POL LEVEL
  - — HONEYWELL LIBERATOR
  - — BURROUGHS FORTRAN–TO–ALGOL TRANSLATOR.
- TO OPERATE AT MACHINE–LANGUAGE LEVEL, TARGET MACHINE MUST BE CLOSE REPLICA OF SOURCE MACHINE
  - — HONEYWELL 200 LIKE IBM 1410.
- REQUIRES MANUAL FIXUP FOR I/O.
- WITH POL'S, CAN TRANSLATE TO EQUIVALENT LANGUAGE, ALTHOUGH FIXUP FOR MISSING FEATURES REQUIRED
  - — BURROUGHS FORTRAN-TO-ALGOL SIMULATES SENSE SWITCH (LITE) OPERATORS IN FORTRAN
  - — SIMSCRIPT TRANSLATES TO FORTRAN.
- OBJECT PROGRAMS RUN AT TARGET MACHINE SPEED.

SUBMACHINES AS CONVERSION AID

- WITHIN—FAMILIES, CAN BE USED.
- UNIVAC II OPERATED IN UNIVAC I MODE
- COMPATIBILITY SWITCH ON 709 TO RUN 704 PROGRAMS.
- DOESN'T HELP ACROSS MACHINE (MFGR.) LINES.
- MINIMUM COMPATIBILITY OF WORD SIZE, TAPE FORMATS.
- NOVEL, BUT NOT DONE EXCEPT WITH OLDER FAMILIES.

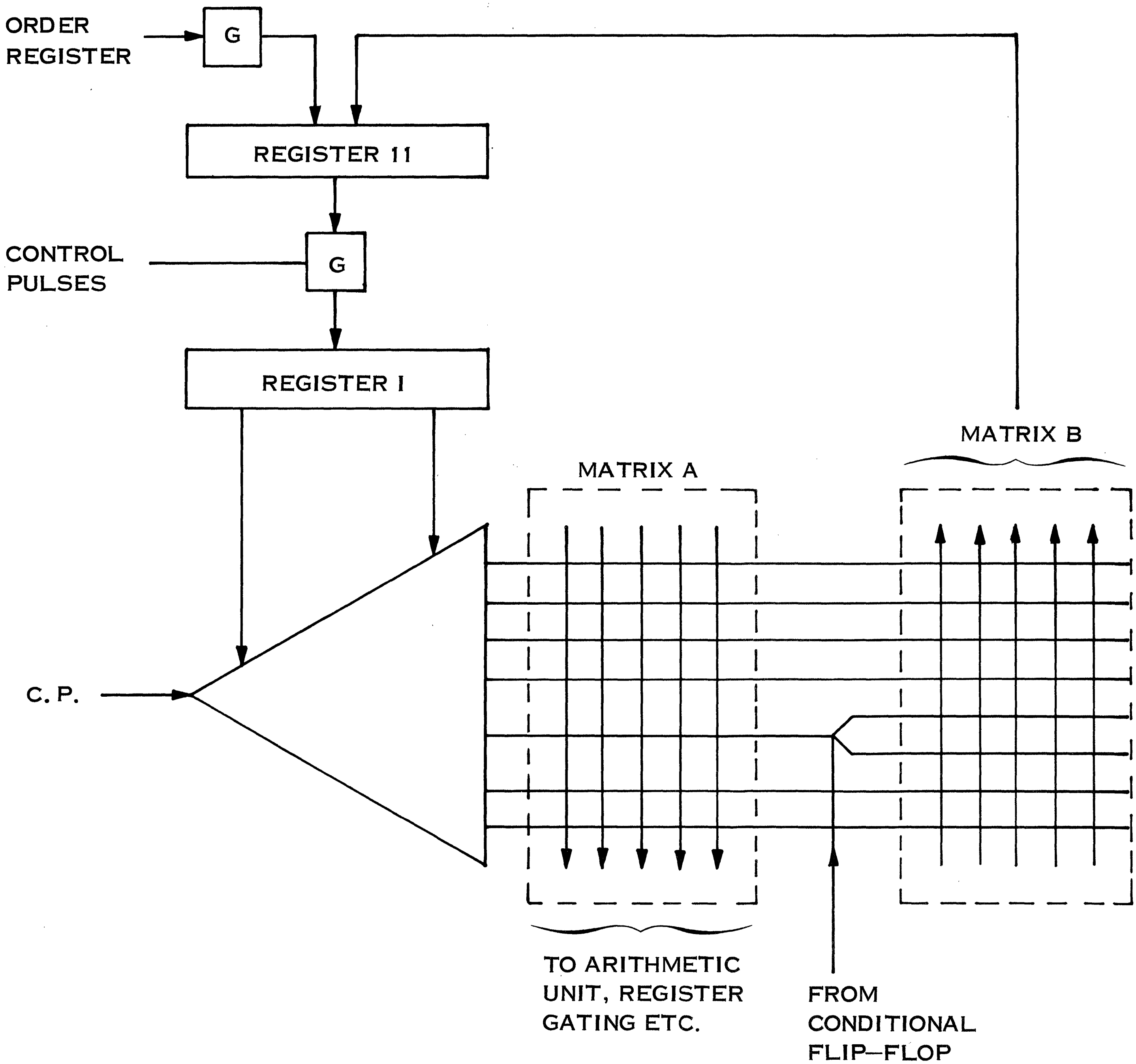
EMULATION – A SOLUTION TO CONVERSION PROBLEMS

- COMBINATION SIMULATION AND MICROPROGRAMMING.
- OBJECTIVES – TO EASE CONVERSION BY PROVIDING SIMULATION AT CLOSE TO ORIGINAL SPEEDS.
- PERMITS ORDERLY CHANGE OF MACHINES.
- WAS ALMOST MANDATORY WITH 360.
- MICROPROGRAMMING VALUABLE IN ITS OWN RIGHT.

MICROPROGRAMMING

- PROGRAMMING WITH ELEMENTARY MACHINE OPERATIONS.
- ELEMENTARY OPERATIONS
  - — REGISTER TRANSFERS
  - — ONE BIT SHIFT
  - — MICROCODE BRANCHING.
- WILKES MACHINE.
- OBJECTIVES OF MICROPROGRAMMING PER SE
  - — CUSTOM—TAILORED INSTRUCTION SETS
  - — COST REDUCTION
  - — CONTROL SYSTEM SIMPLIFICATION.

WILKES MICROPROGRAM CONTROL



IMPORTANT FEATURES OF WILKES DESIGN

- CONDITIONAL BRANCH.
- USE OF OP CODE AS ADDRESS OF FIRST MICROORDER.

OTHER IMPORTANT MICROPROGRAMMING DEVELOPMENTS

- MICRO SUBROUTINES.
- MICRO CONSTANTS.
- GROUPING FIELDS AND DECODING TO CONTROL PARTICULAR DATA PATHS.
- WRITABLE CONTROL STORAGE.
- TAILORED MACHINE LANGUAGE INSTRUCTION SETS.

USE OF MICROPROGRAMMING IN 360

- REDUCE CONTROL COSTS IN SMALLER MODELS.
- GIVE COMPREHENSIVE INSTRUCTION SETS ACROSS ALL MACHINE MODELS.
- PERMITS TAILORING FOR SPECIAL APPLICATIONS OR FOR VARIANTS ON BASIC LINE.
- READ–ONLY MEMORY (ROM) STORES MICRO–ORDERS.

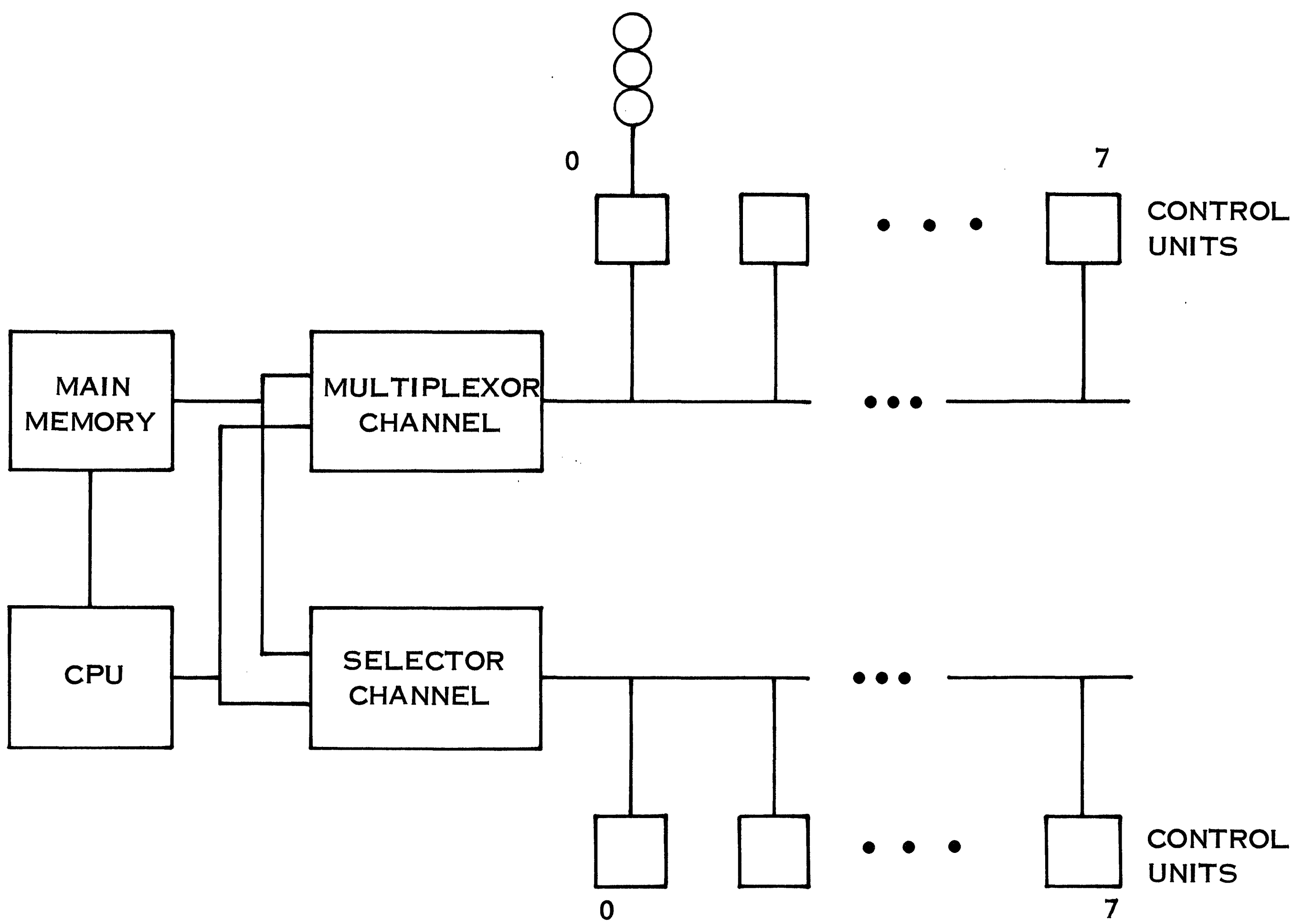
EMULATOR DESIGN COMPONENTS

- DEDICATED ROM FOR EMULATOR (MAY BE SEPARATE ROM FOR MACHINE INSTRUCTIONS).
- SELECTION OF SPECIAL INSTRUCTIONS TO ADD TO BASIC (EMULATING MACHINE)
  - — DIL
  - — BRANCH IF.
- DETERMINE WHETHER FULL COMPATIBILITY OR SOME PROGRAMMED OPERATIONS
  - — COST
  - — COMPLEXITY
  - — FREQUENCY OF OCCURRENCE.

LIMITATIONS OF MICROPROGRAMMING/EMULATOR APPROACH

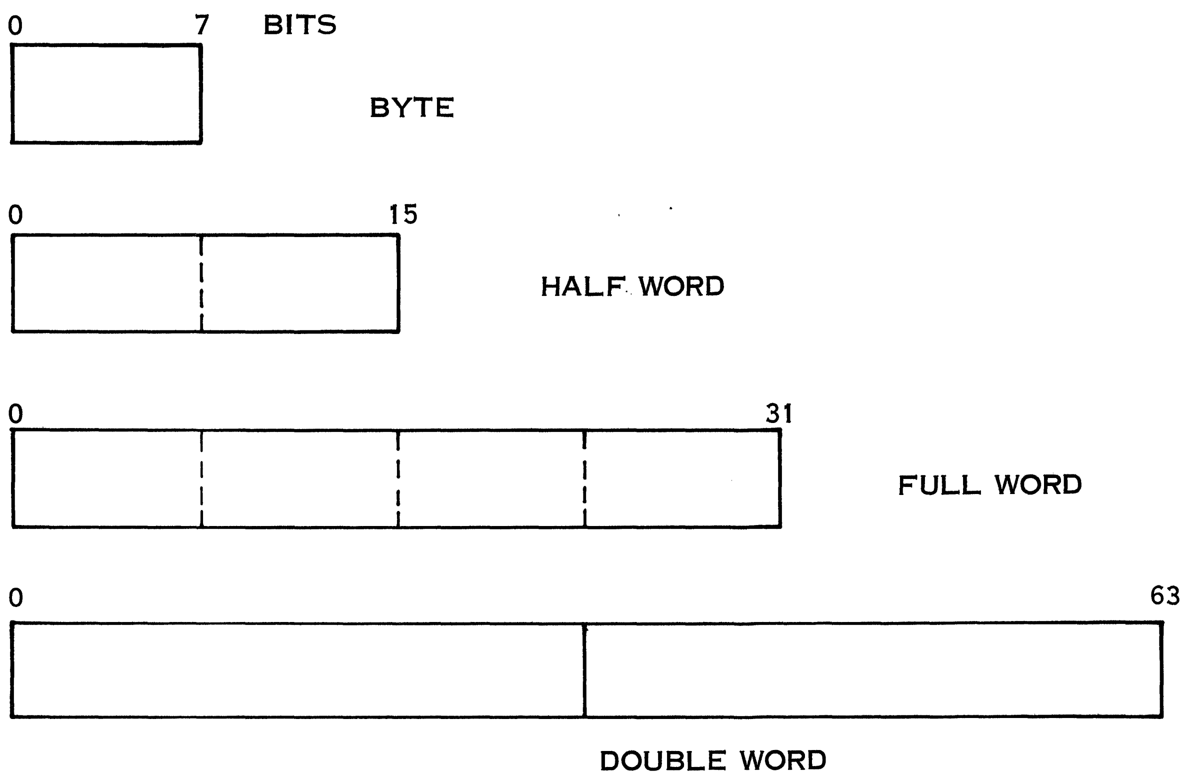
- MICROPROGRAMMING ATTRACTIVE FOR COMPLEX AND/OR LARGE INSTRUCTION SETS.
- ROM TECHNIQUE FAST, INFLEXIBLE (TO USER).
- MAIN MEMORY (WRITEABLE CONTROL STORE) PERMITS GREATEST FLEXIBILITY TO USER.
- EMULATOR (MICROPROGRAM + PROGRAMS) FOR LARGER MACHINES.
- COMPATIBLE (THRU MICROPROGRAM) FOR SMALLER MACHINES.
- EMULATING MACHINE REGISTER STRUCTURE AND DATA PATHS MUST BE COMPATIBLE WITH TARGET MACHINE. GREATER DEVIATION, MORE COMPLEX AND DIFFICULT.
- NO RPQ OR OTHER NON—STANDARD FEATURES ON SOURCE MACHINE.

PRINCIPAL COMPONENTS OF 360 SYSTEM

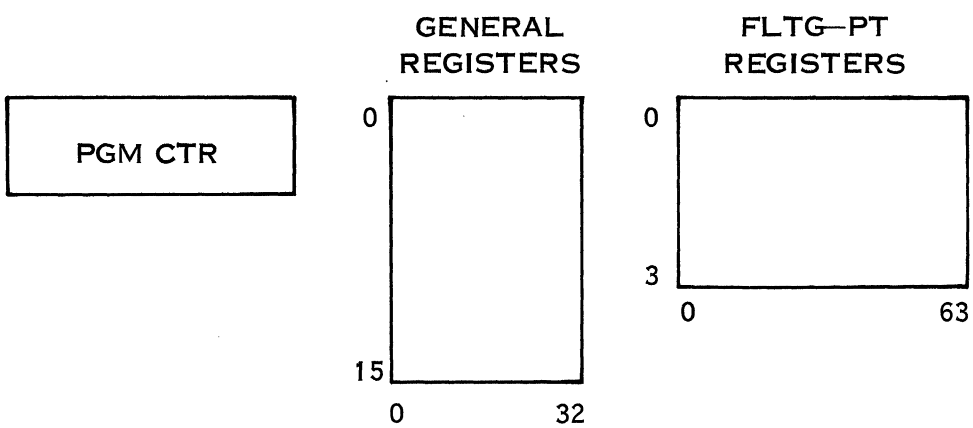


INFORMATION STRUCTURE IN 360

- BYTE — (FAT CHARACTER)
- HALF WORD (2 BYTES)
- FULL WORD (4 BYTES)



360 CPU STRUCTURE



360 INSTRUCTION FORMATS

OP	R_1	R_2

REG. TO REGISTER

$(R_1) < OP > (R_2) \longrightarrow R_1$

OP	R_1	X_2	B_2	D_2

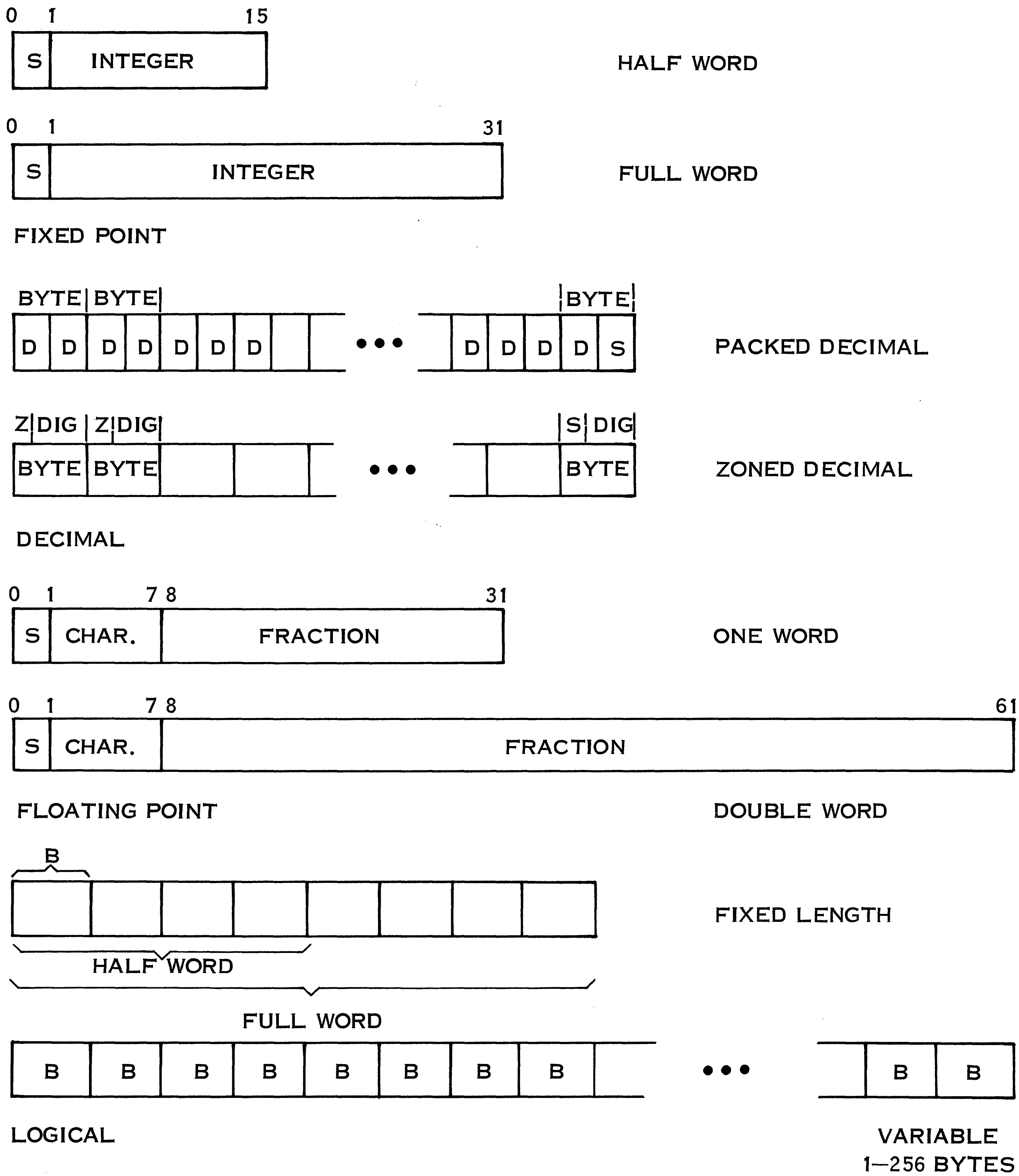
$(R_1) < OP > (D_2 ** X_2 B_2) \longrightarrow R_1$

OP	R_1	R_3	B_2	D_2

OP	I_2	B_1	D_1

OP	L_1	L_2	B_1	D_1	B_2	D_2

360 TYPES OF OPERATIONS AND DATA FORMATS



360 STATEWORD — PSW

SYSTEM MASK	KEY	AMWP	INTERRUPT CODE

ILC	CC	PROGRAM MASK	INSTRUCTION ADDRESS

A — ASCII—8 MODE

M — MACHINE CHECK MASK

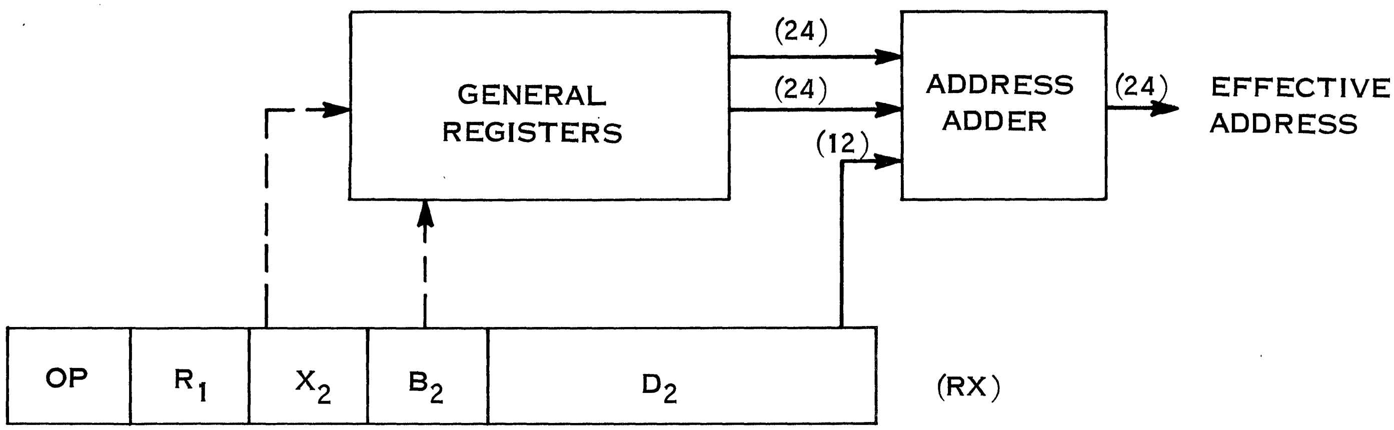
W — WAIT STATE

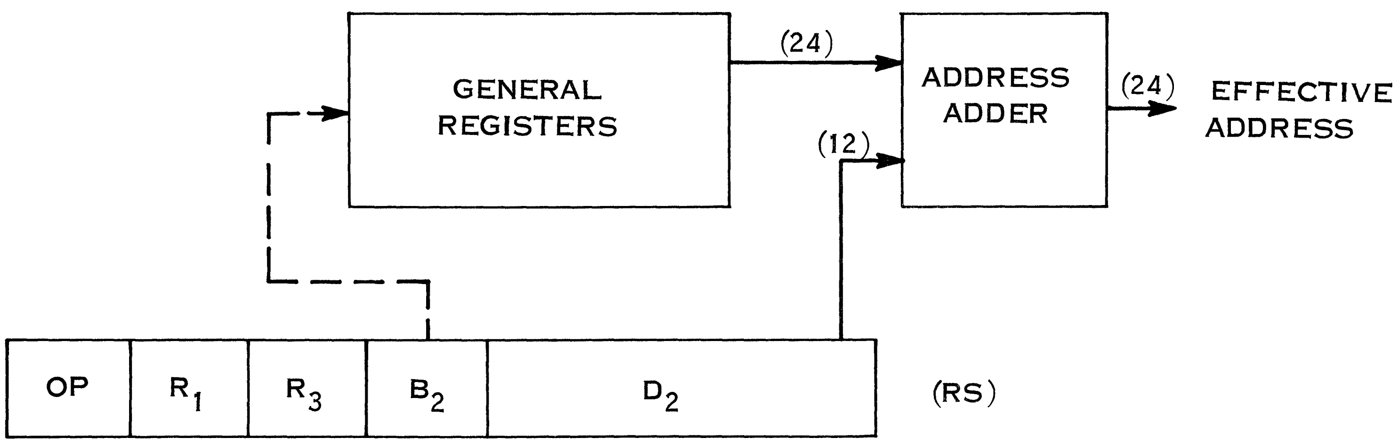
P — PROBLEM STATE

ILC — INST. LENGTH CODE

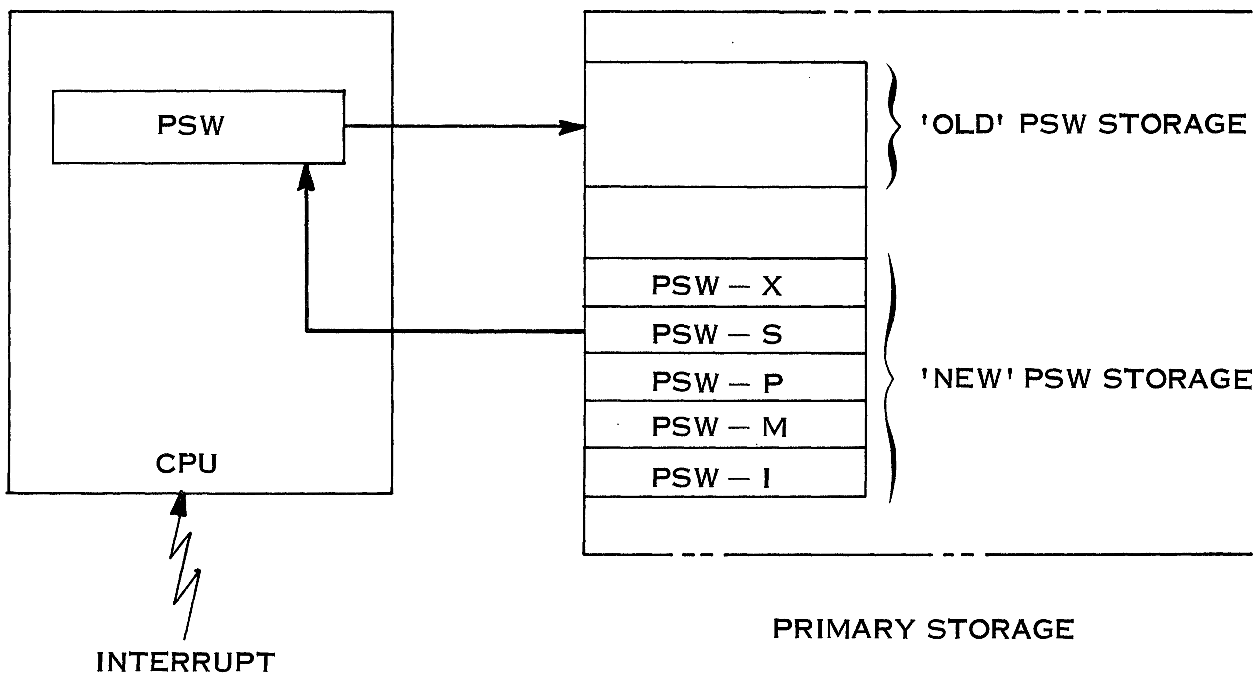
CC — CONDITION CODE

ADDRESS FORMATION IN 360





360 INTERRUPT SYSTEM



360 CPU FEATURES FOR MULTIPROGRAMMING AND MULTIPROCESSING

- PROVIDES MULTIPLE BASE ADDRESSING (NOT IN ALL INSTRUCTIONS)
- COMPREHENSIVE INTERRUPT SYSTEM
- PROBLEM STATE/SUPERVISOR STATE
- NO INDIRECT ADDRESSING
- FIXED INTERRUPT RESPONSE LOCATIONS

MEMORY SUBSYSTEM — 360

MODEL	MINIMUM PRIMARY STORAGE	MAXIMUM PRIMARY STORAGE
30	8,192	65,536
40	16,384	262,144
44	32,768	262,144
50	65,536	524,288
65	131,072	1,048,576
67	262,144	1,048,576
75	262,144	1,048,576
85	524,288	4,194,304
91	1,048,576	6,291,456

STORAGE PROTECTION – 360

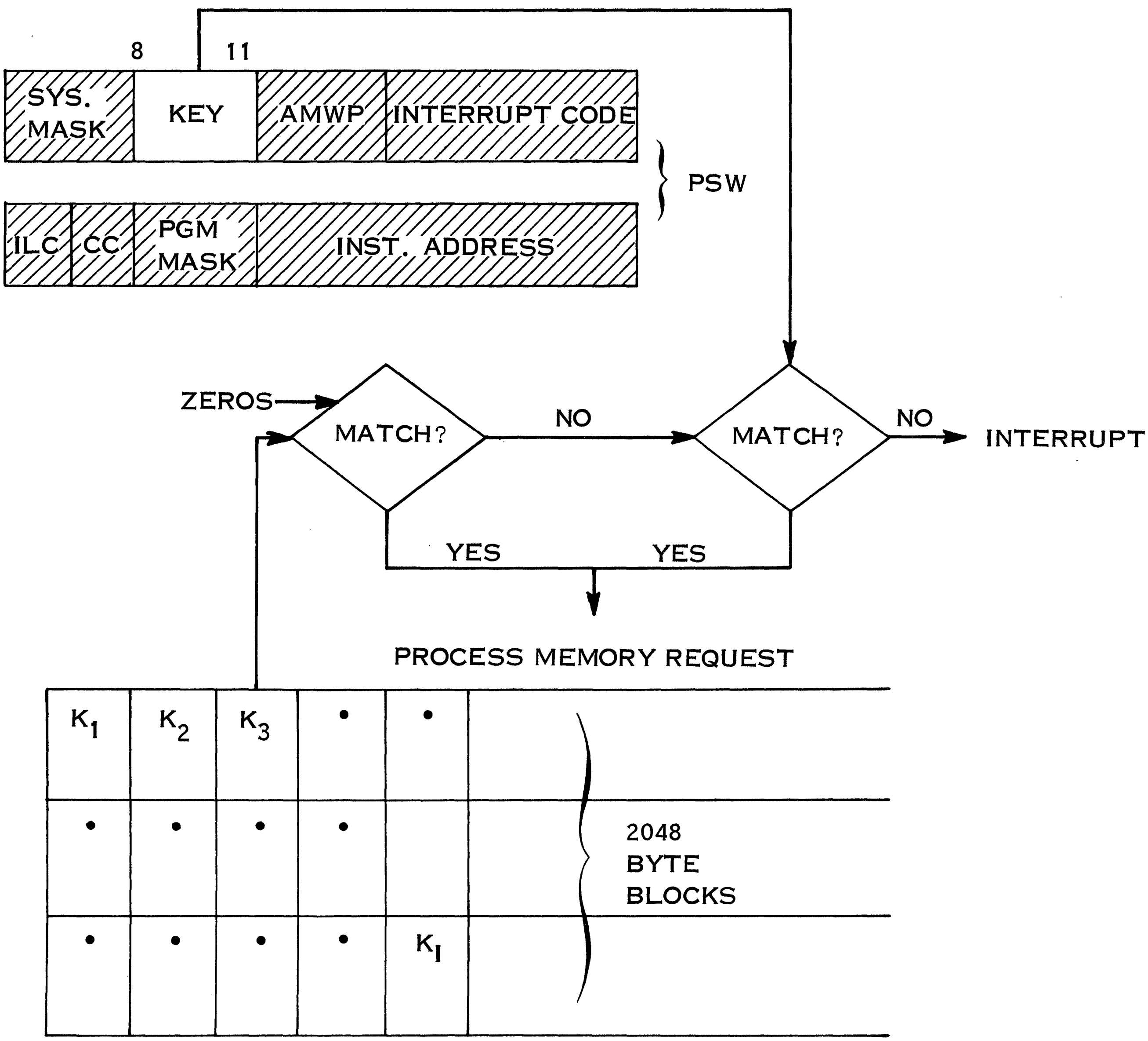
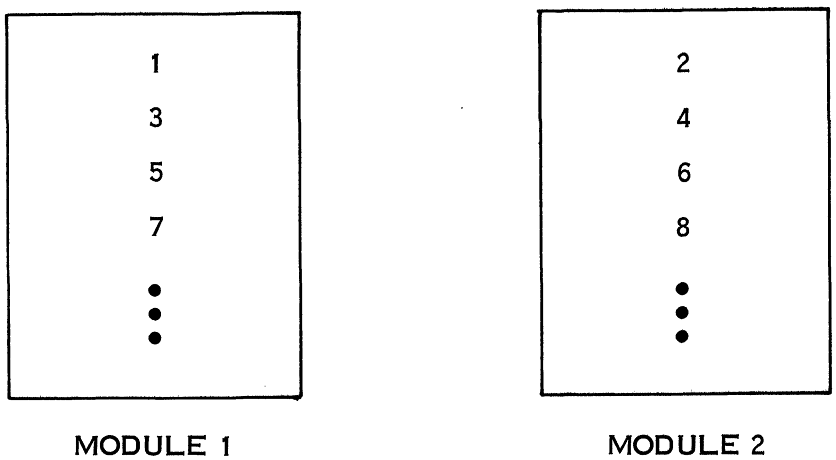


ILLUSTRATION OF MEMORY INTERLACE



2—WAY INTERLACE

- FASTER OPERATION (ON AVERAGE) BY NOT HAVING TO WAIT FOR WRITE HALF—CYCLE
- FAILURE IN ONE MODULE EXCLUDES USE OF OTHER

LCS – SYSTEM IMPLICATIONS

- SIZE – 1M, 2M (UP TO 8M)
- SPEED – 8 μS
- AVAILABLE FOR MOD 50, 65, 75 ⟶
- WHAT TO DO WITH IT
  - SYSTEM PROGRAMS RESIDENCE
  - FILE DIRECTORIES
  - OPERATING SYSTEM RESIDENCE
  - SWAPPING STORE

360 CHANNELS

- SELECTOR
  - 'BURST MODE' OPERATIONS
  - HIGH SPEED DEVICES
- MULTIPLEXOR
  - SLOWER DEVICES
  - SHARE MULTIPLEXOR LOGIC
  - USING SUBCHANNELS

CHANNEL COMMAND WORD — AN I/O PROGRAM

COMMAND CODE	DATA ADDRESS

FLAGS	000		COUNT.

FLAGS: CHAIN DATA

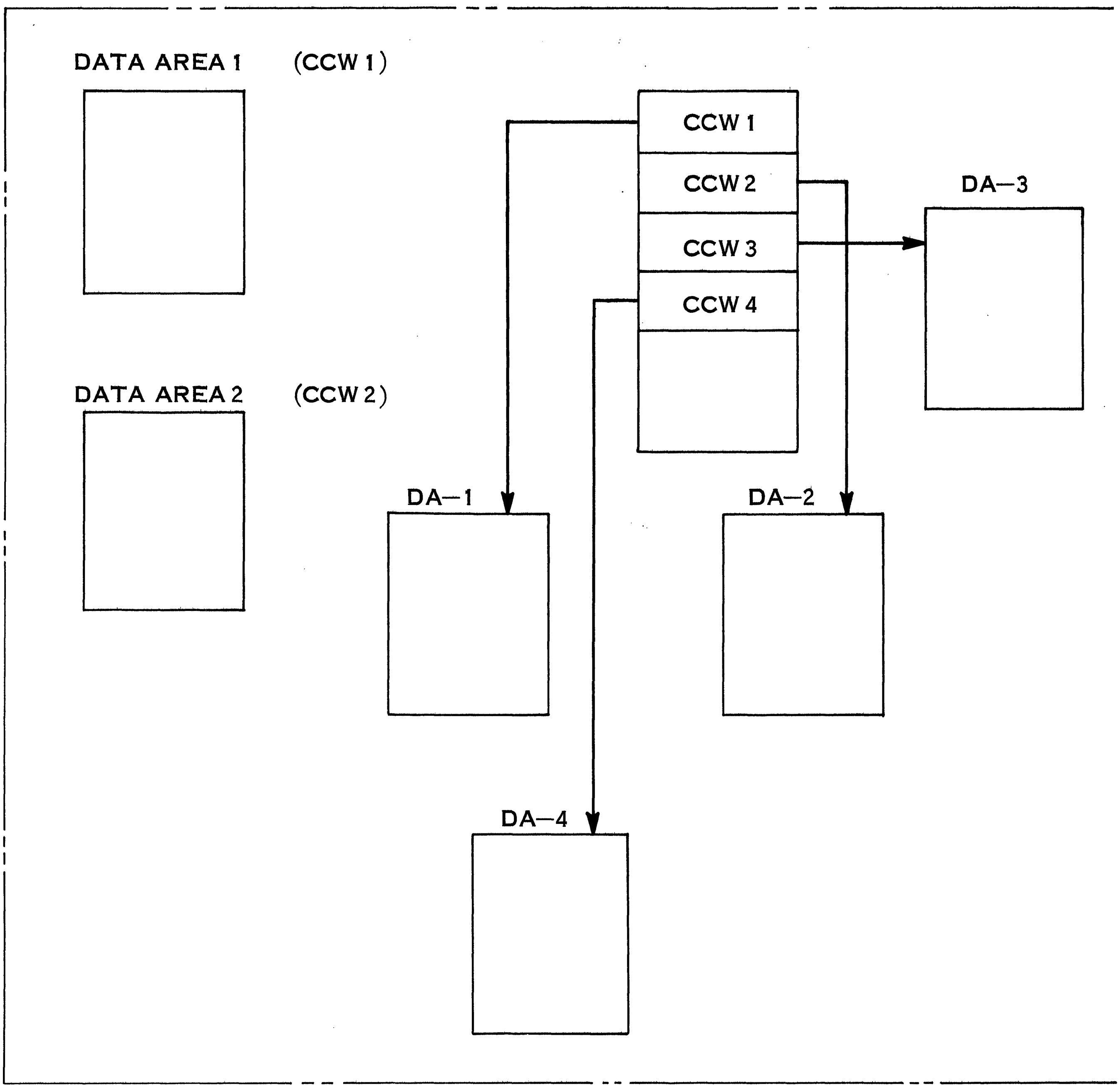
CHAIN COMMAND

SUPPRESS LENGTH INDICATION

SKIP

PROGRAM CONTROLLED INTERRUPT

CHAINING



PRIMARY STORAGE

360 PROVISIONS FOR MULTISYSTEM OPERATION

- CPU COMMUNICATION
  - SHARED I/O – DISK
  - CHANNEL TO CHANNEL
  - SHARED STORAGE
  - CPU START SIGNAL (FROM ANOTHER CPU)

- INSTRUCTION AIDS
  - READ (WRITE) DIRECT
  - EXTERNAL INTERRUPT LINES
  - PERMANENT STORAGE RELOCATION AND ALTERNATE LOC. (PREFIX)
  - TEST AND SET

DIAGNOSTIC FACILITIES FOR 360

- 5 CLASSES OF INTERRUPTS
  - I/O
  - MACHINE CHECK
  - PROGRAM CHECK
  - SUPERVISOR CALL
  - EXTERNAL.

- PROGRAM CHECK ON
  - OPERATION EXCEPTION
  - PRIVILEGED—OPERATION EXCEPTION
  - EXECUTE EXCEPTION
  - PROTECTION EXCEPTION
  - ADDRESSING EXCEPTION
  - SPECIFICATION EXCEPTION
  - DATA EXCEPTION
  - FIXED POINT OVERFLOW
  - FIXED POINT DIVIDE
  - DECIMAL OVERFLOW
  - DECIMAL DIVIDE
  - EXPONENT OVERFLOW
  - EXPONENT UNDERFLOW
  - SIGNIFICANCE
  - FLOATING POINT DIVIDE

FEATURES OF RCA SPECTRA/70

- COPY OF 360 (PROBLEM MODE)
- HAS 4 PROCESSOR STATES

    1. PROBLEM (USER) STATE
    2. INTERRUPT RESPONSE STATE
    3. INTERRUPT CONTROL STATE
    4. MACHINE CONDITION STATE

SPECTRA 70 PROCESSOR STATE REGISTERS

REGISTER	STATE			
	P_1	P_2	P_3	P_4
PROGRAM COUNTER	1	1	1	1
GENERAL REGISTERS	16	16	6	5
FLOATING POINT REGISTERS	4	—	—	—
INTERRUPT STATUS REGISTERS	1	1	1	1
INTERRUPT MASK REGISTERS	1	1	1	1

SUMMARY OF IMPORTANT CHARACTERISTICS OF S/70

- PROVIDES MULTICOMPUTER ARRANGEMENTS THROUGH DIRECT CONTROL TRUNK.

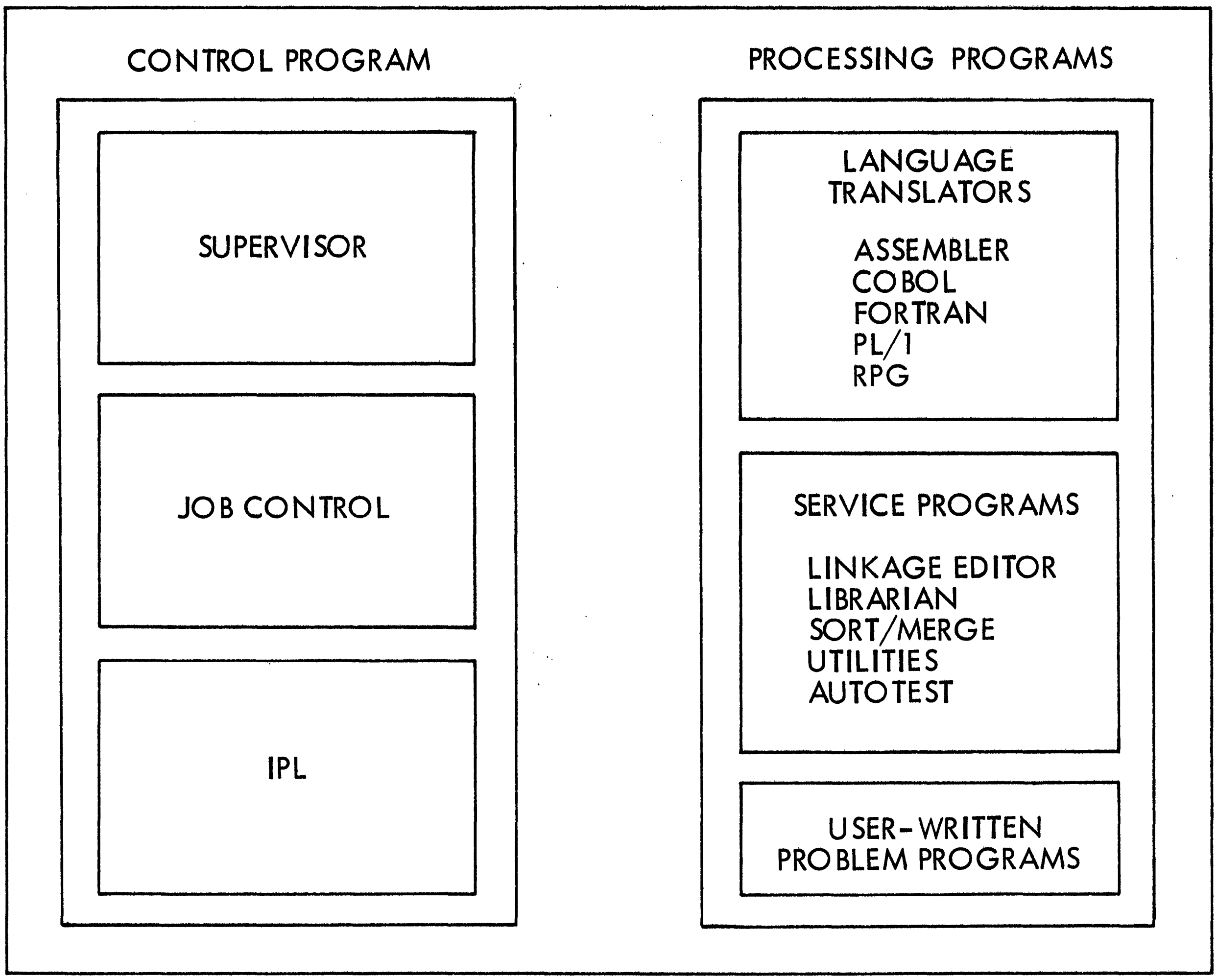
- EMULATORS FOR 301, 501 (RCA) 1401 1410

- INTERNAL OPERATION LIKE 360

OPERATING SYSTEMS FOR 360

- BPS (BASIC PROGRAMMING SUPPORT)
- DOS (DISK)
- TOS (TAPE)
- OS (FULL)
- MFT (FULL WITH MULTIPROGRAMMING)
- MVT (FULL WITH VARIABLE TASKING)

THE DOS ENVIRONMENT



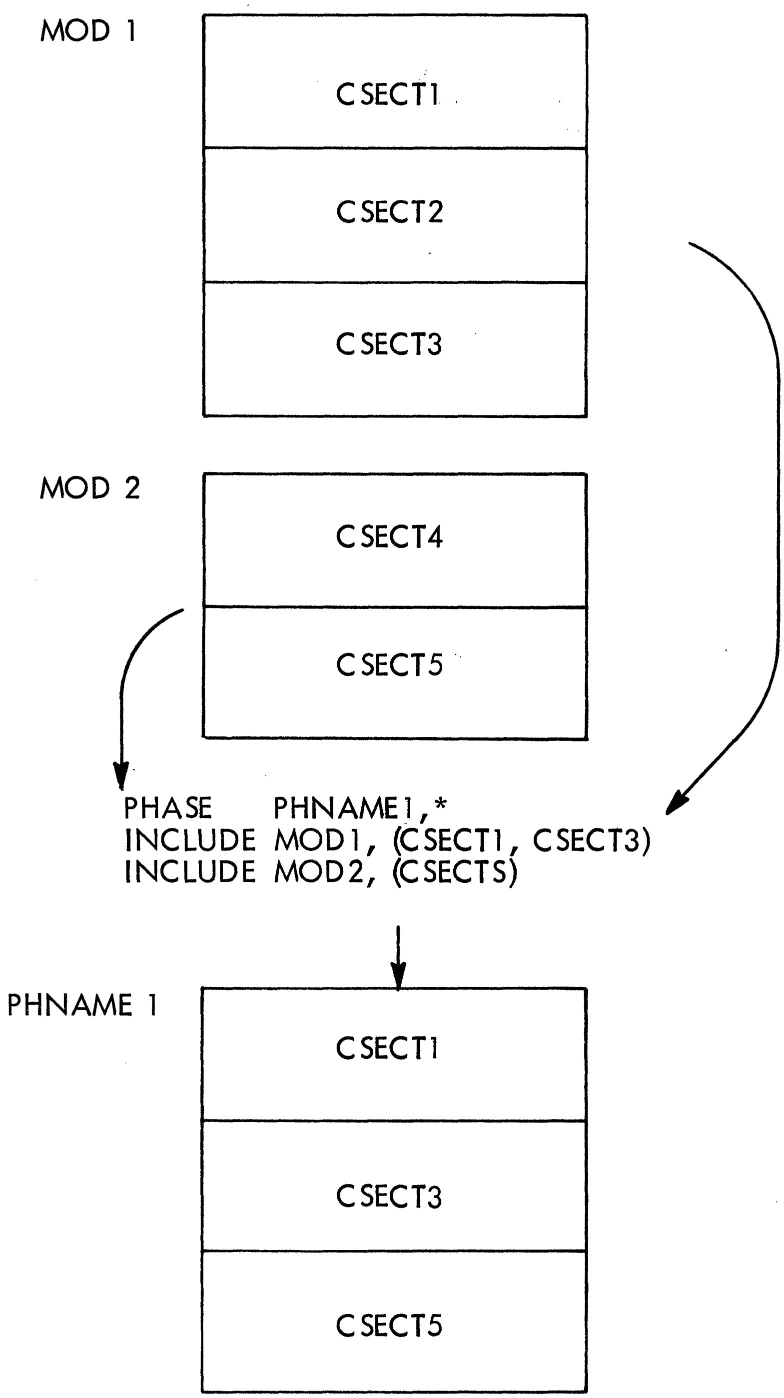
▲ BACKGROUND

- CONTROL STREAM
- SEQUENTIAL
- USES JOB CONTROL
- NO OPERATOR INTERVENTION
- LOWEST PRIORITY

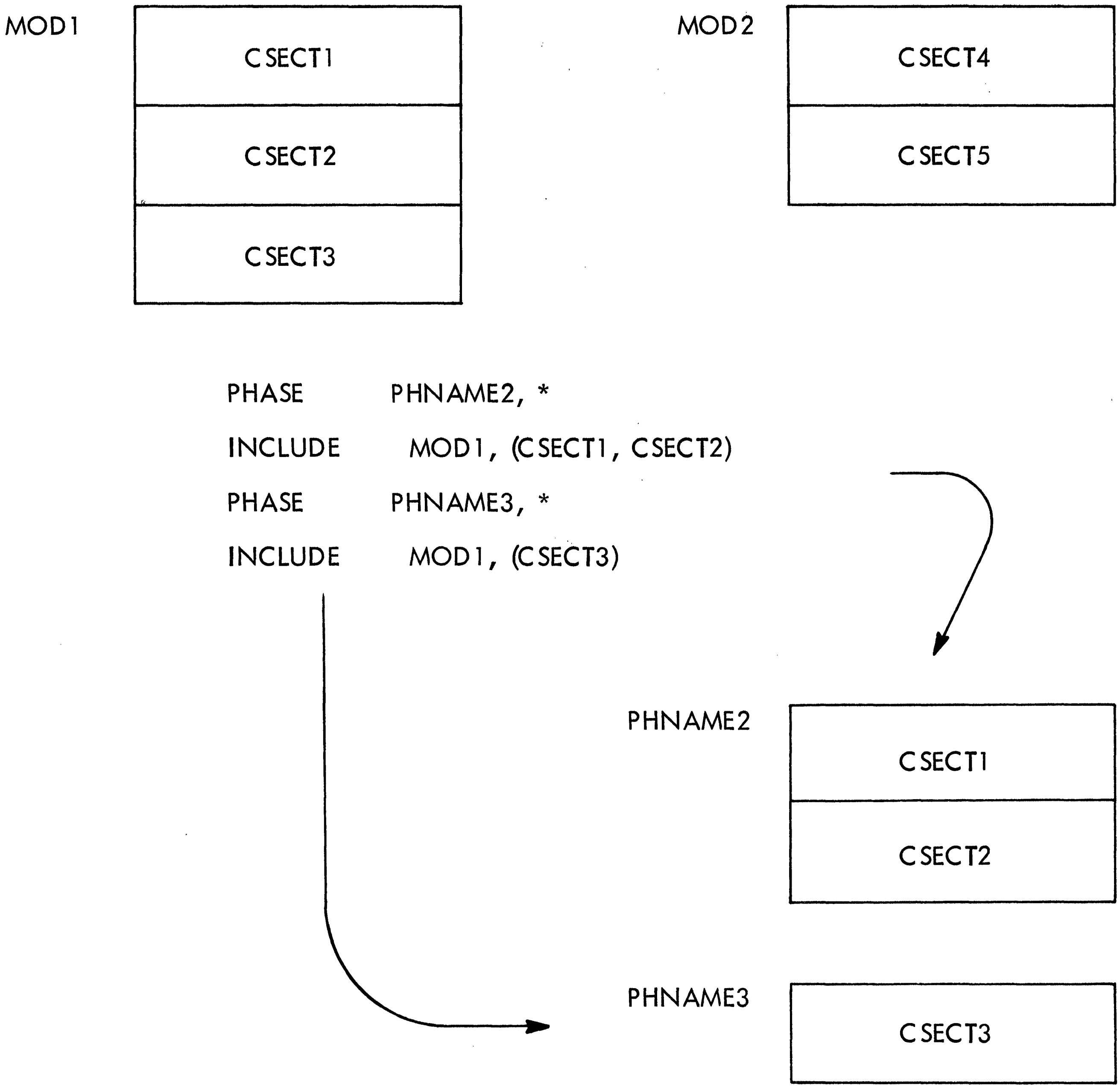
▲ FOREGROUND

- NO CONTROL STREAM
- OPERATOR CONTROLLED
- USES INITIATORS
- HIGHEST PRIORITY

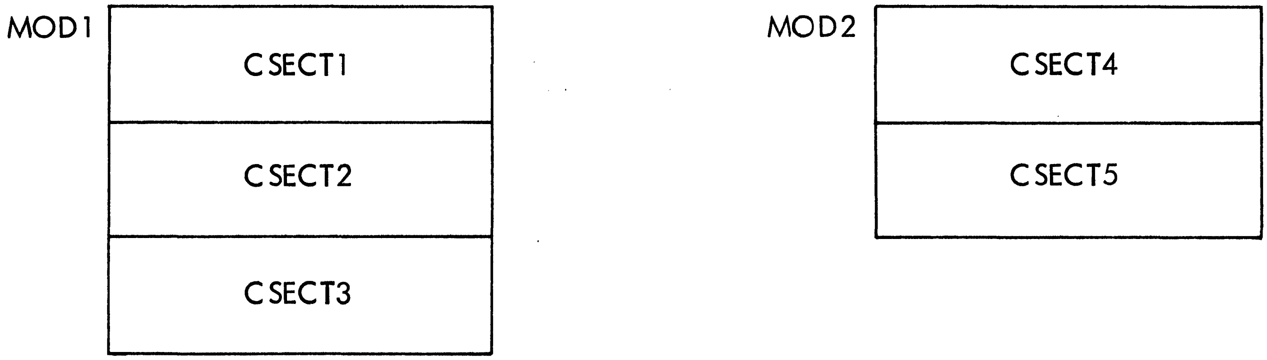
CREATION OF OVERLAY PHASE



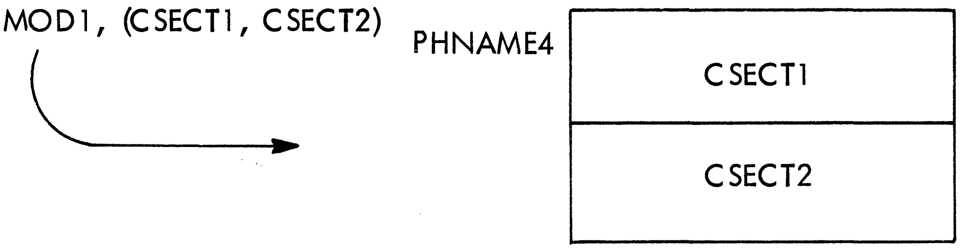
TWO PHASES FROM ONE OBJECT MODULE



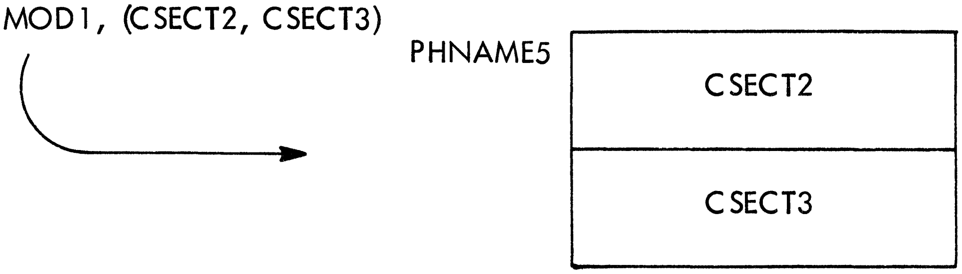
USING SAME OBJECT MODULE TWICE

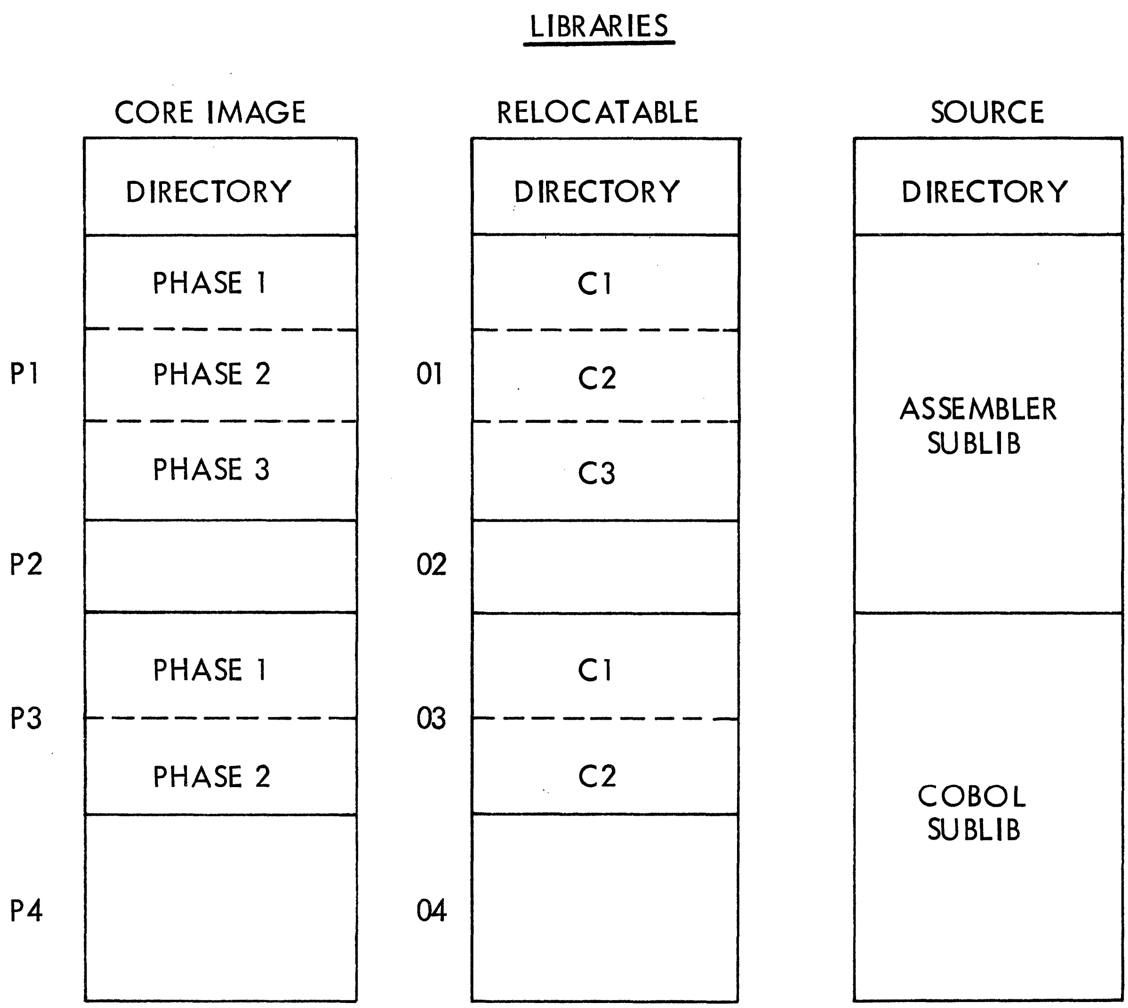



```
PHASE     PHNAME4,*
INCLUDE   MOD1, (CSECT1, CSECT2)
```

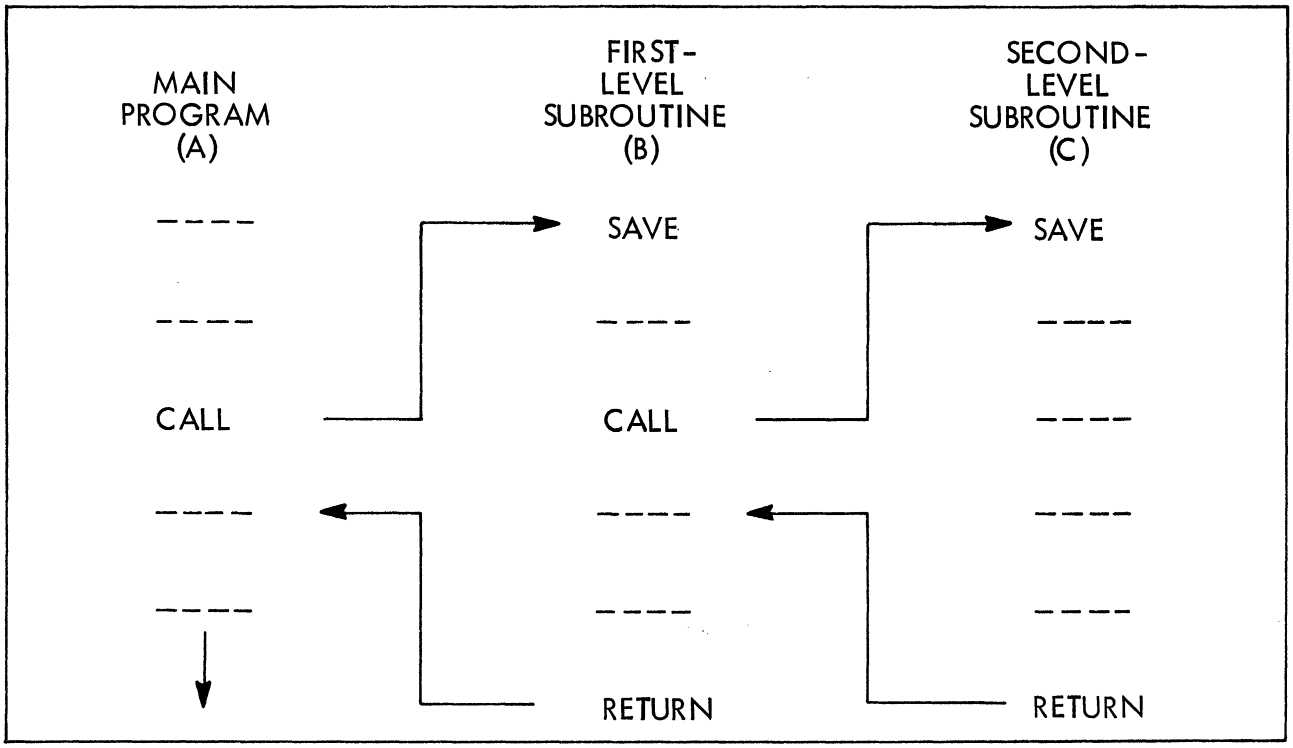



```
PHASE     PHNAME5,*
INCLUDE   MOD1, (CSECT2, CSECT3)
```

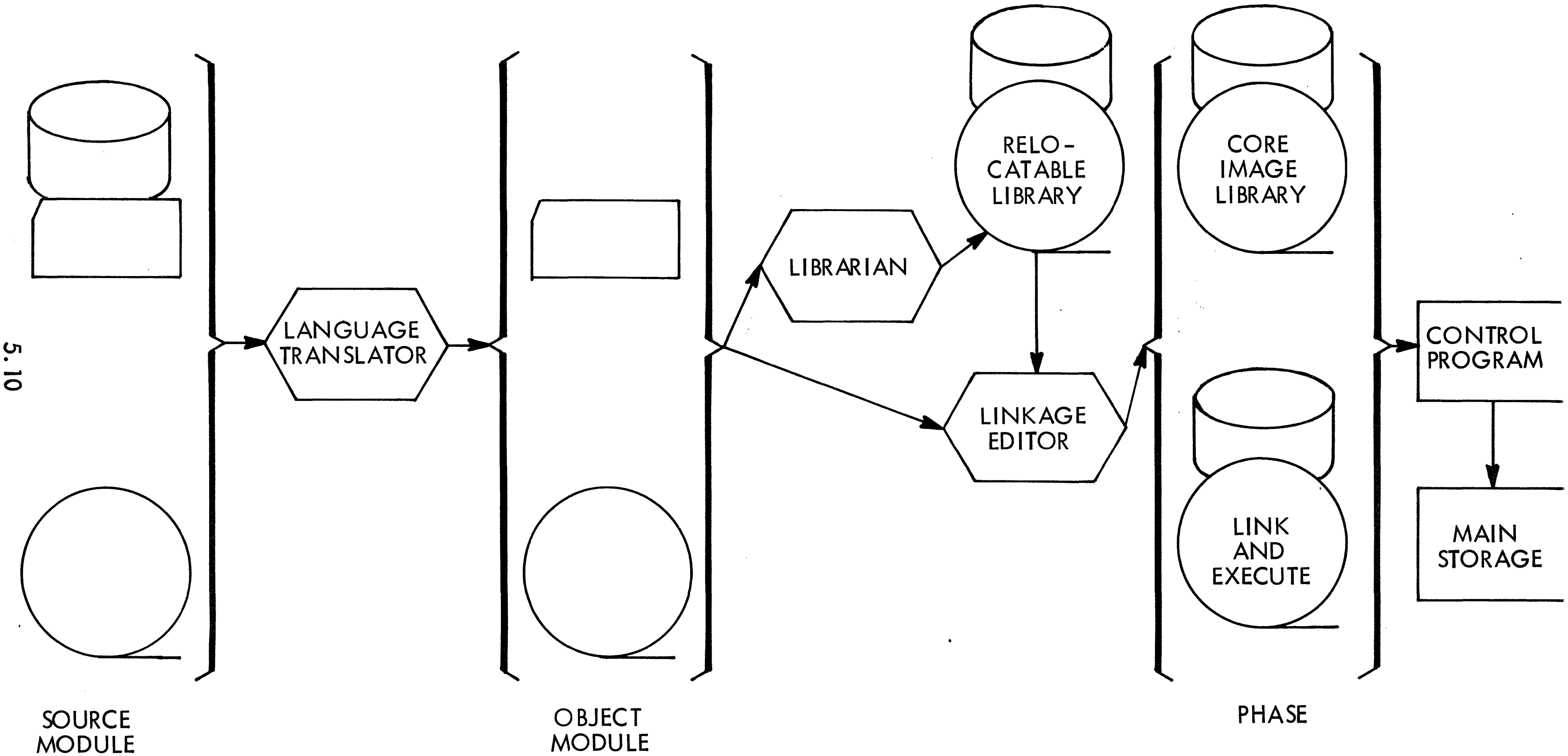



LIBRARIES
CORE IMAGE
RELOCATABLE
SOURCE
DIRECTORY
DIRECTORY
DIRECTORY
PHASE 1
PHASE 2
PHASE 3
P1
P2
P3
P4
PHASE 1
PHASE 2
C1
C2
C3
01
02
03
04
C1
C2
ASSEMBLER
SUBLIB
COBOL
SUBLIB

DIRECT LINKAGES

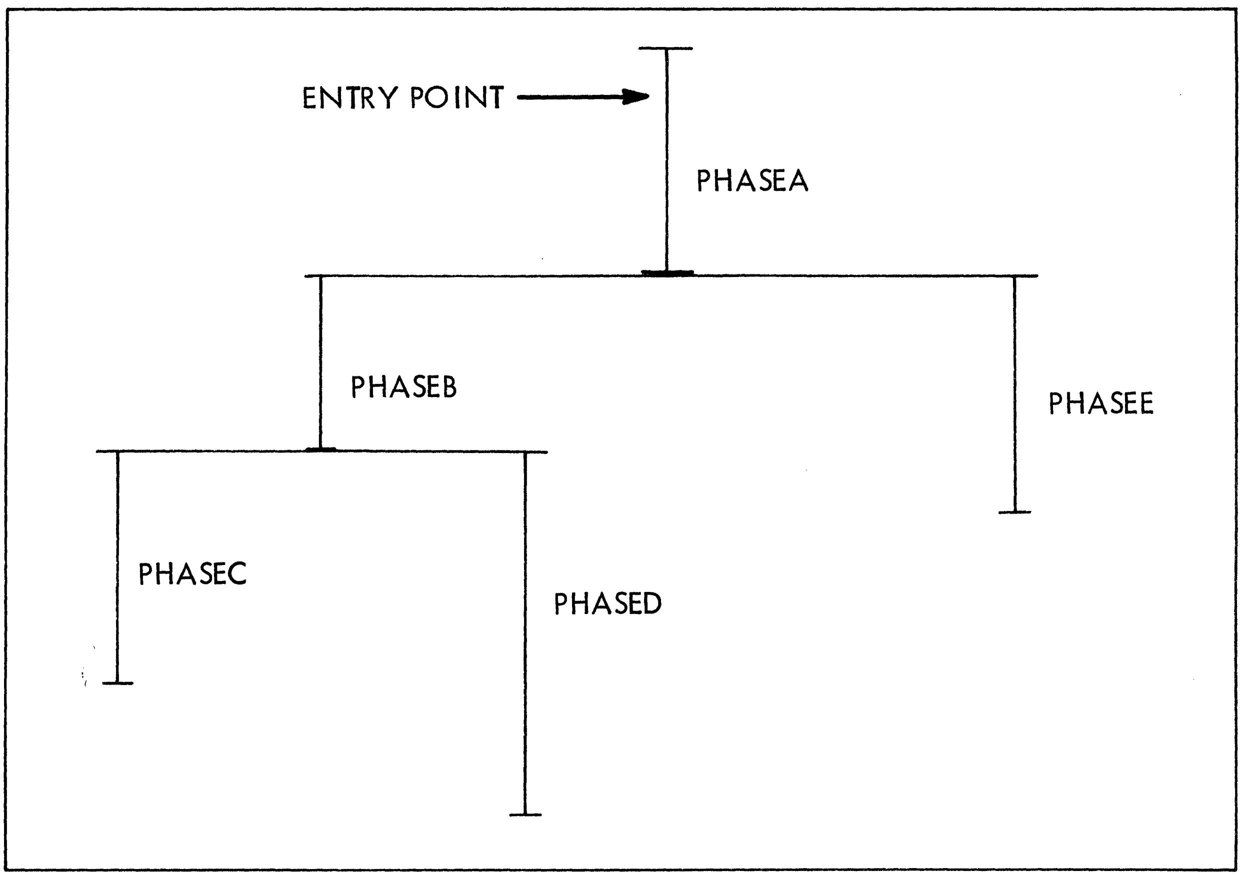


PROGRAM STAGES



GENERATION OF AN OVERLAY TREE STRUCTURE

```
PHASE   PHASEA,ROOT
PHASE   PHASEB,*
PHASE   PHASEC,*
PHASE   PHASED,PHASEC
PHASE   PHASEE,PHASEB
```



JOB CONTROL EXAMPLE

```
//  JOB EXAMPLE1                                          }
    .                                                     }
    .                                                     }
    .                                                     }
//  OPTION LINK,LIST               }                      }
//  EXEC COBOL                     }  Step 1              }
    (COBOL Source Deck)            }                      }  Job 1
/*                                 }                      }
//  EXEC LNKEDT                    }  Step 2              }
//  EXEC                           }                      }
    (Data for Object Program)      }  Step 3              }
/*                                 }                      }
/&                                                        }
//  JOB EXAMPLE2                                          }
    .                                                     }
    .                                                     }
    .                                                     }
//  VOL SYS004,MASTER                                     }  Job 2
//  TPLAB 'label-information'                             }
//  EXEC PAYROLL                                          }
    (Data for Payroll Program)                            }
/*                                                        }
/&                                                        }
```

OPERATING SYSTEM ELEMENTS

Control Program Elements

- Job Management
- Task Management
- Data Management

Processing Program Elements

Languages	Service Programs	Application Programs
ALGOL Assembler COBOL FORTRAN PL/I RPG	Data Set Utilities Independent Utilities Linkage Editor Sort/Merge System Utilities TESTRAN	User Written

PRODUCING A LOAD MODULE

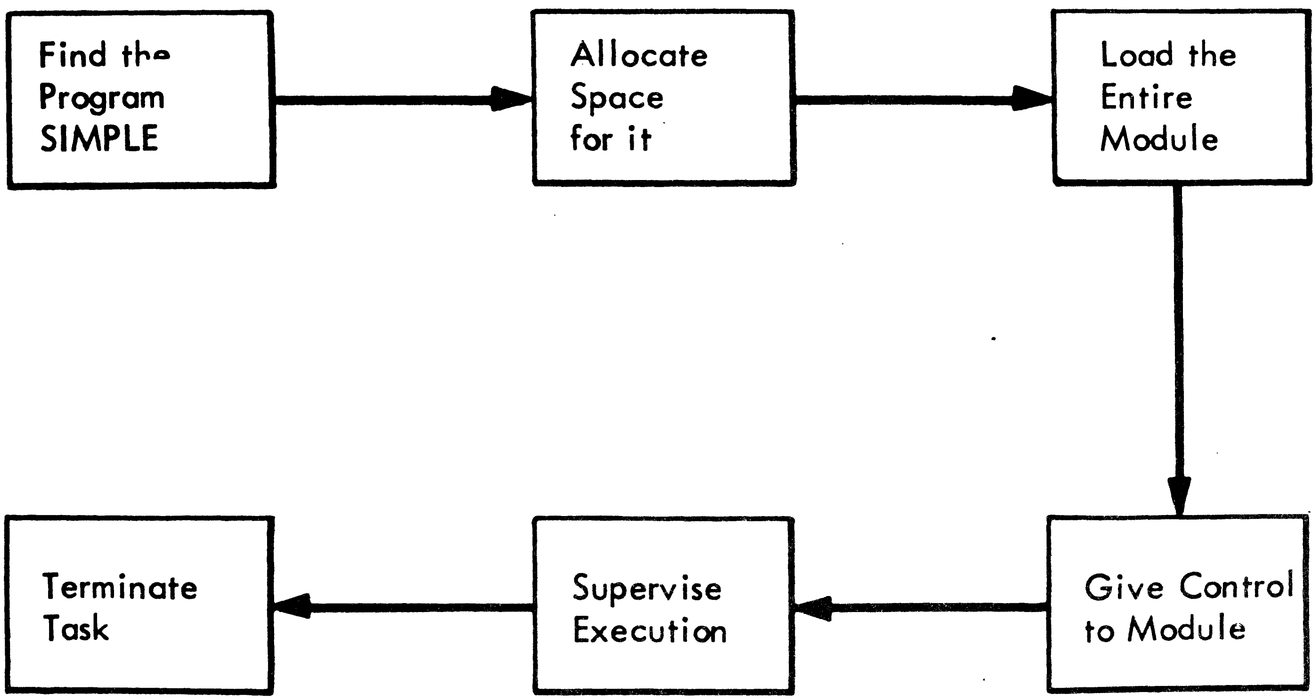
User Input	Source Program + Translator Control Statements	Transcribed to Cards or Tape
Operating System Component		Processed by a Language Translator
Output of Language Translator and Input to Linkage Editor	Linkage Editor Control Statements	Which Yields An Object Module
Operating System Component		Processed by the Linkage Editor
Output of Linkage Editor		Which Yields A Load Module

LOAD MODULE ATTRIBUTES

Structure Type	Loaded All At One Time	Passes Control to Other Load Modules
Simple	Yes	No
Planned Overlay	No	No or Yes[1]
Dynamic	Yes or No[1]	Yes

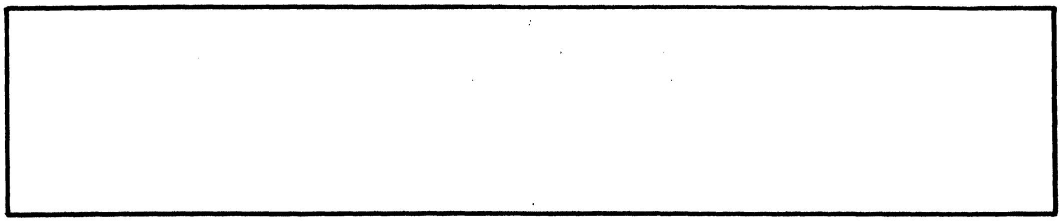
[1]A segment of a load module can dynamically call another load module.

SYSTEM LOGIC FLOW FOR A SIMPLE STRUCTURE

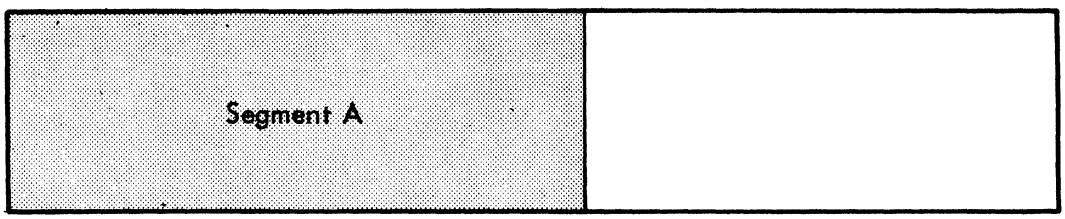


STORAGE ALLOCATION FOR A PLANNED OVERLAY STRUCTURE

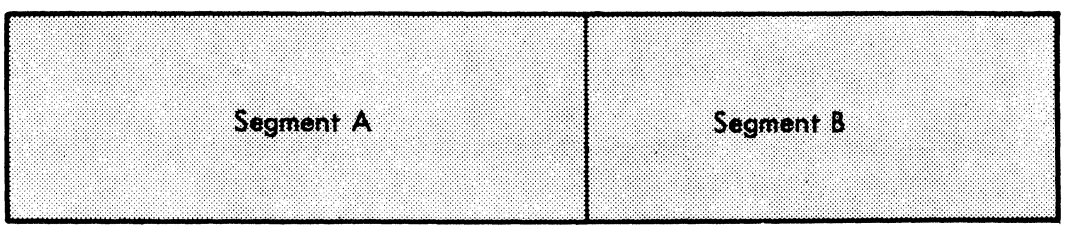
Storage Available to OVERLAY



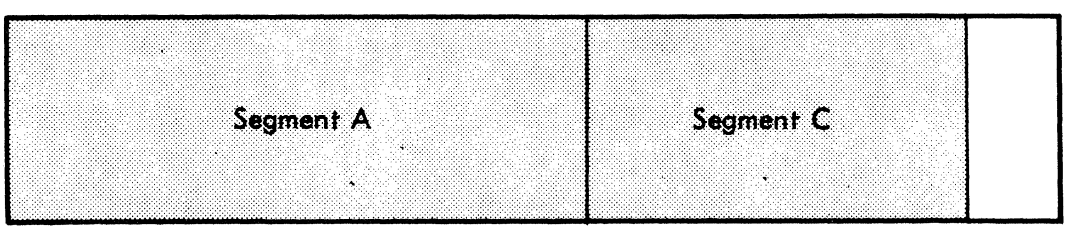
Storage Occupied by Segment A (the Root Segment)



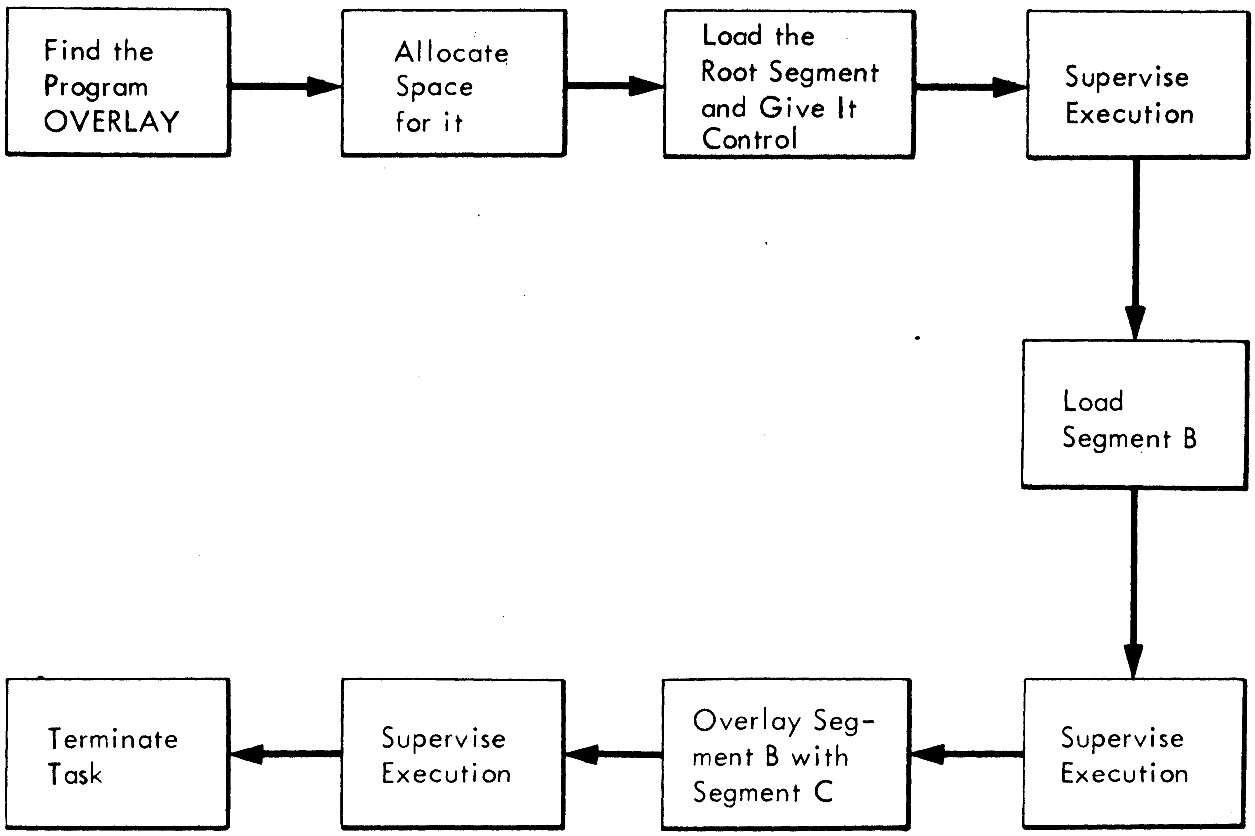
Storage When Segments A and B are Resident



Storage After Segment C Overlays Segment B

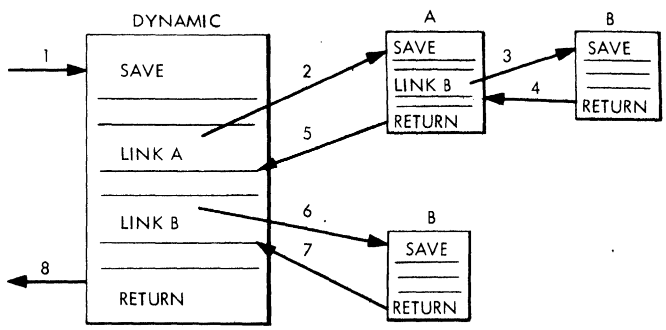


SYSTEM RESPONSE FOR A PLANNED OVERLAY STRUCTURE

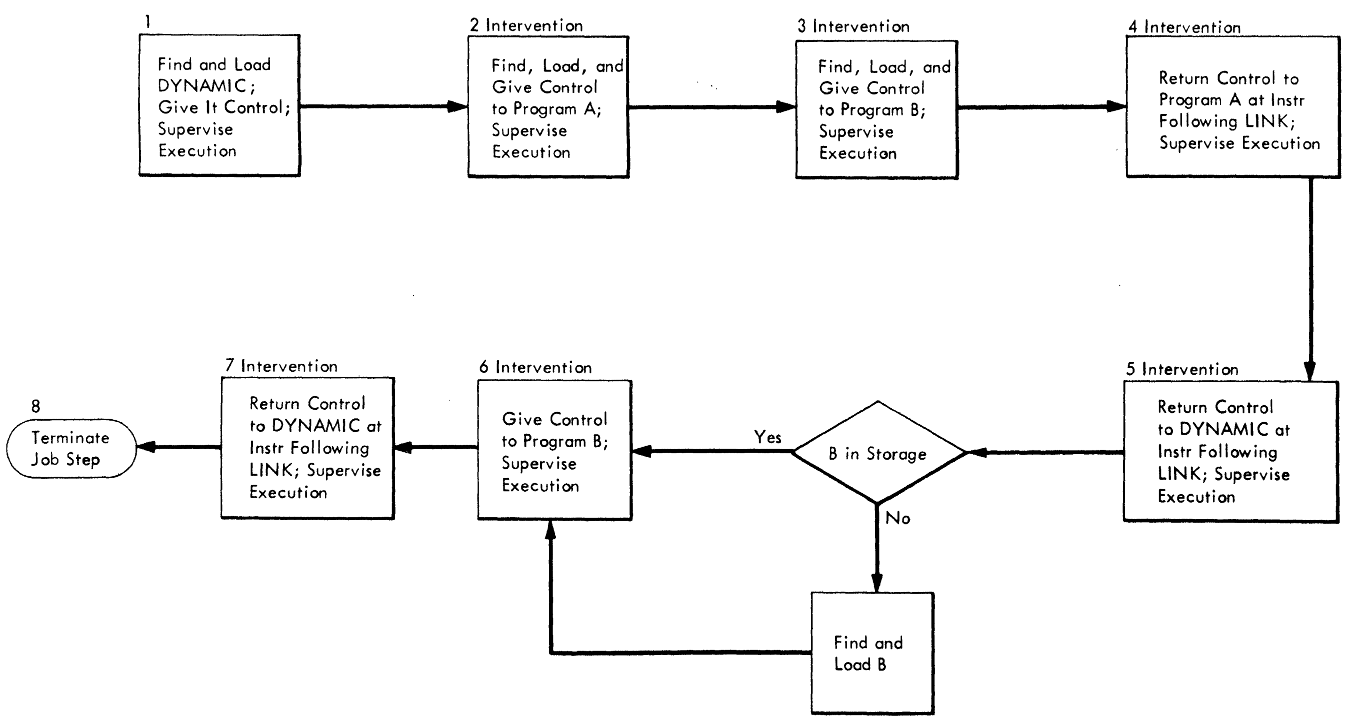


DYNAMIC EXECUTION, ONE TASK PER JOB STEP

USER'S REQUEST

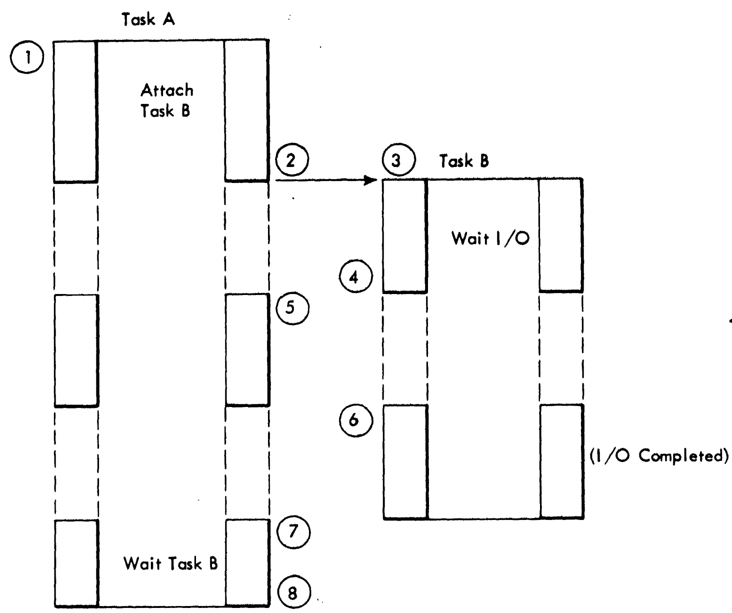


SYSTEM RESPONSE

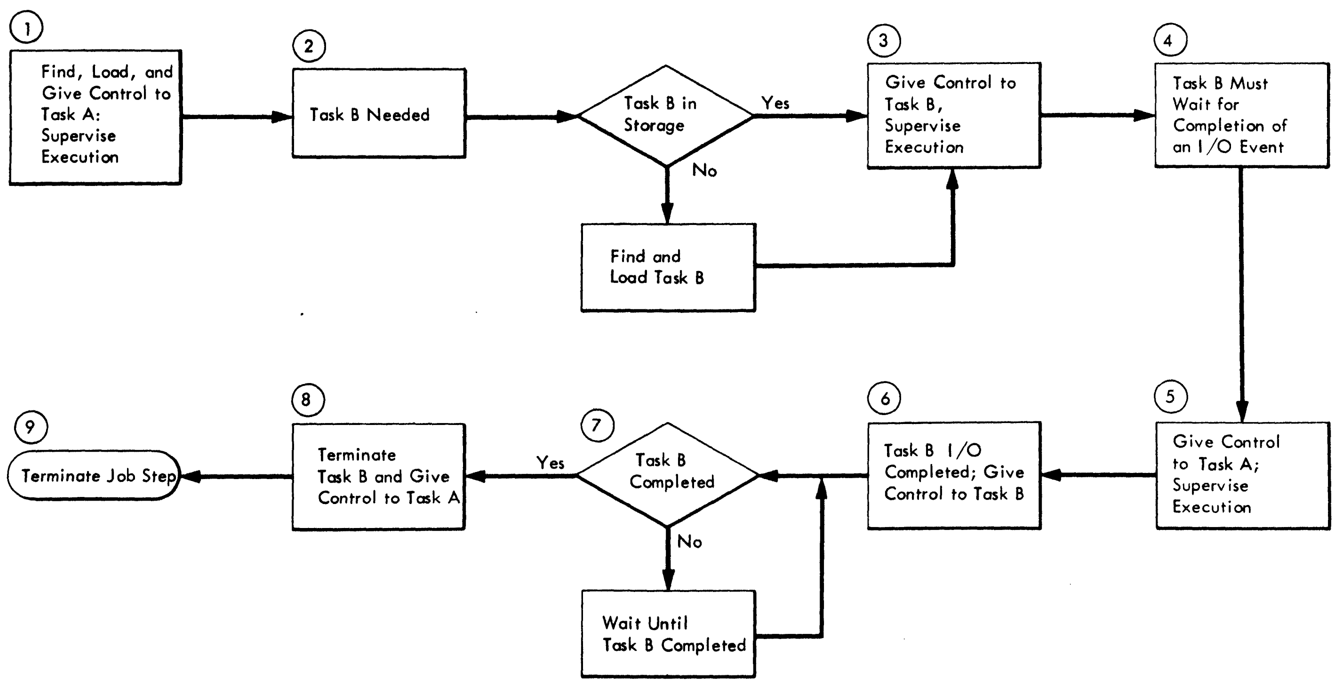


DYNAMIC EXECUTION, MORE THAN ONE TASK PER JOB STEP

User Requests



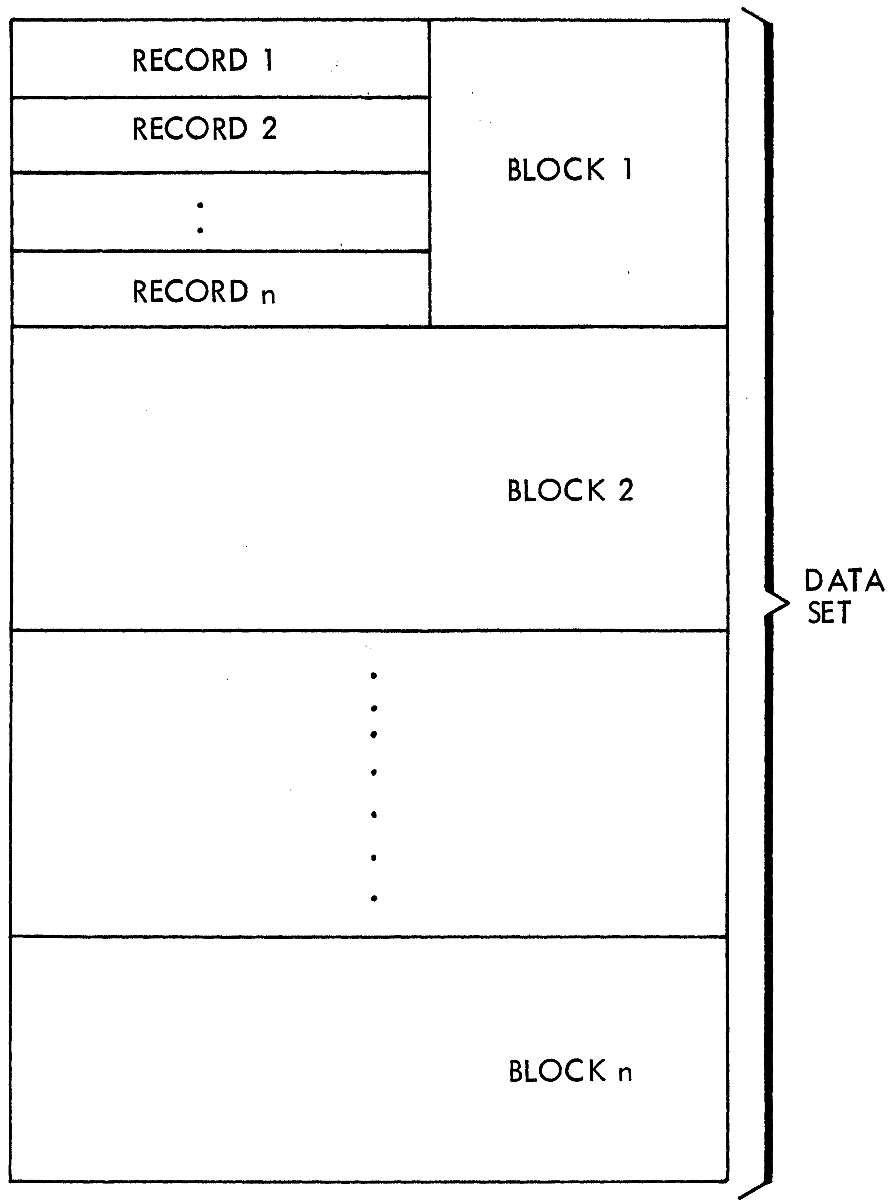
System Response



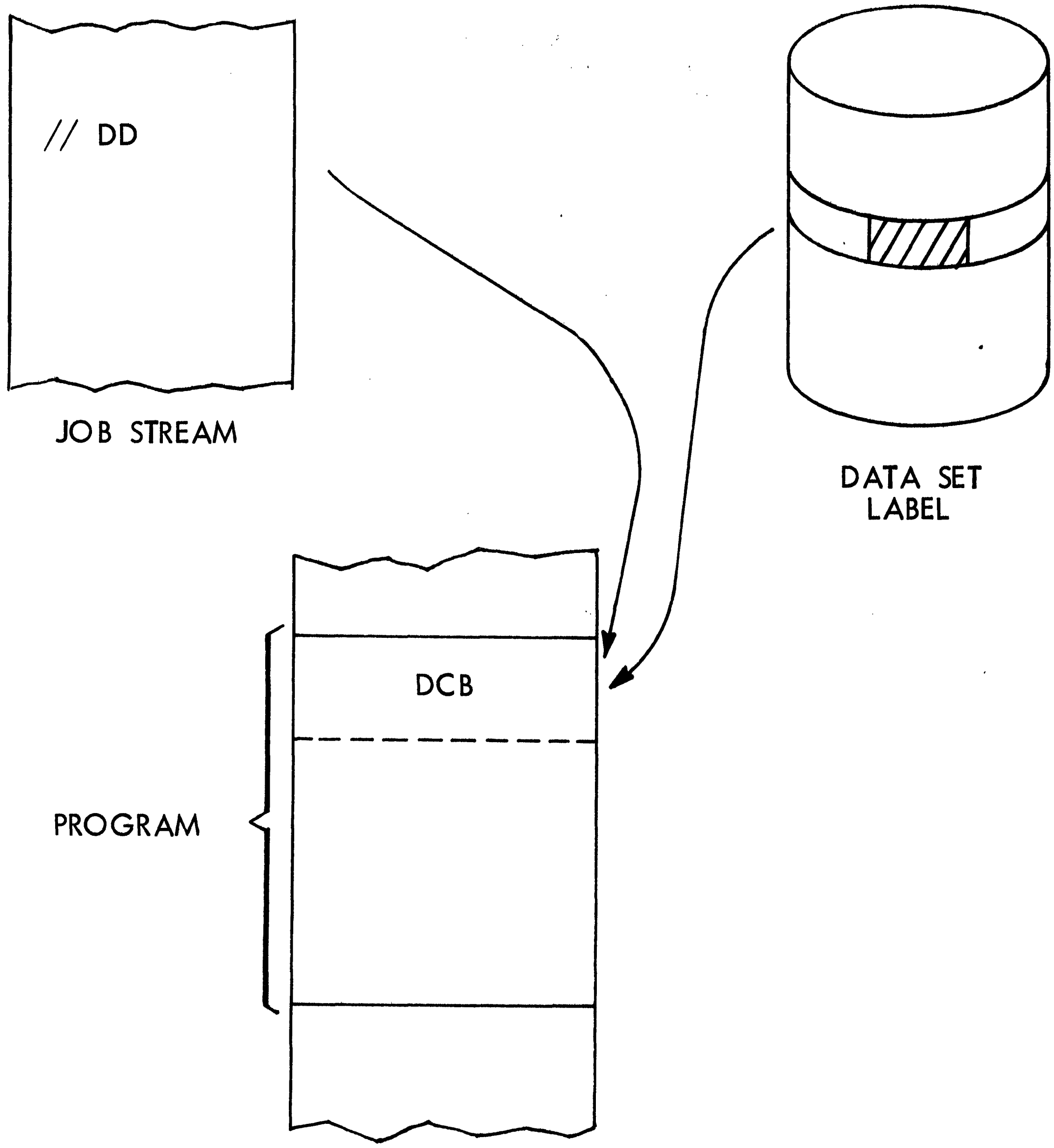
REUSABILITY

- NON-REUSABLE
- SERIALLY REUSABLE
- REENTERABLE

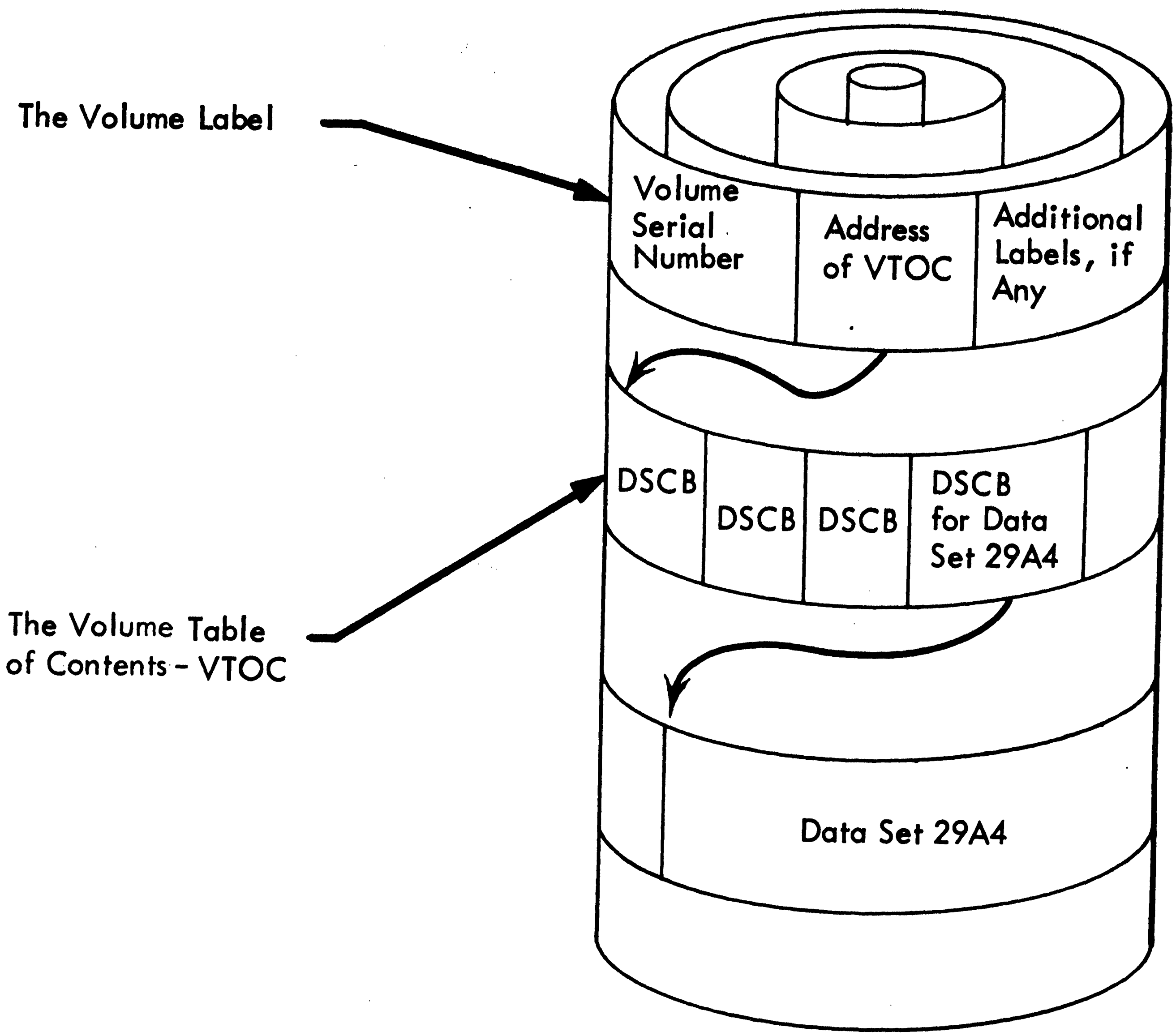
DATA SETS, BLOCKS, AND RECORDS



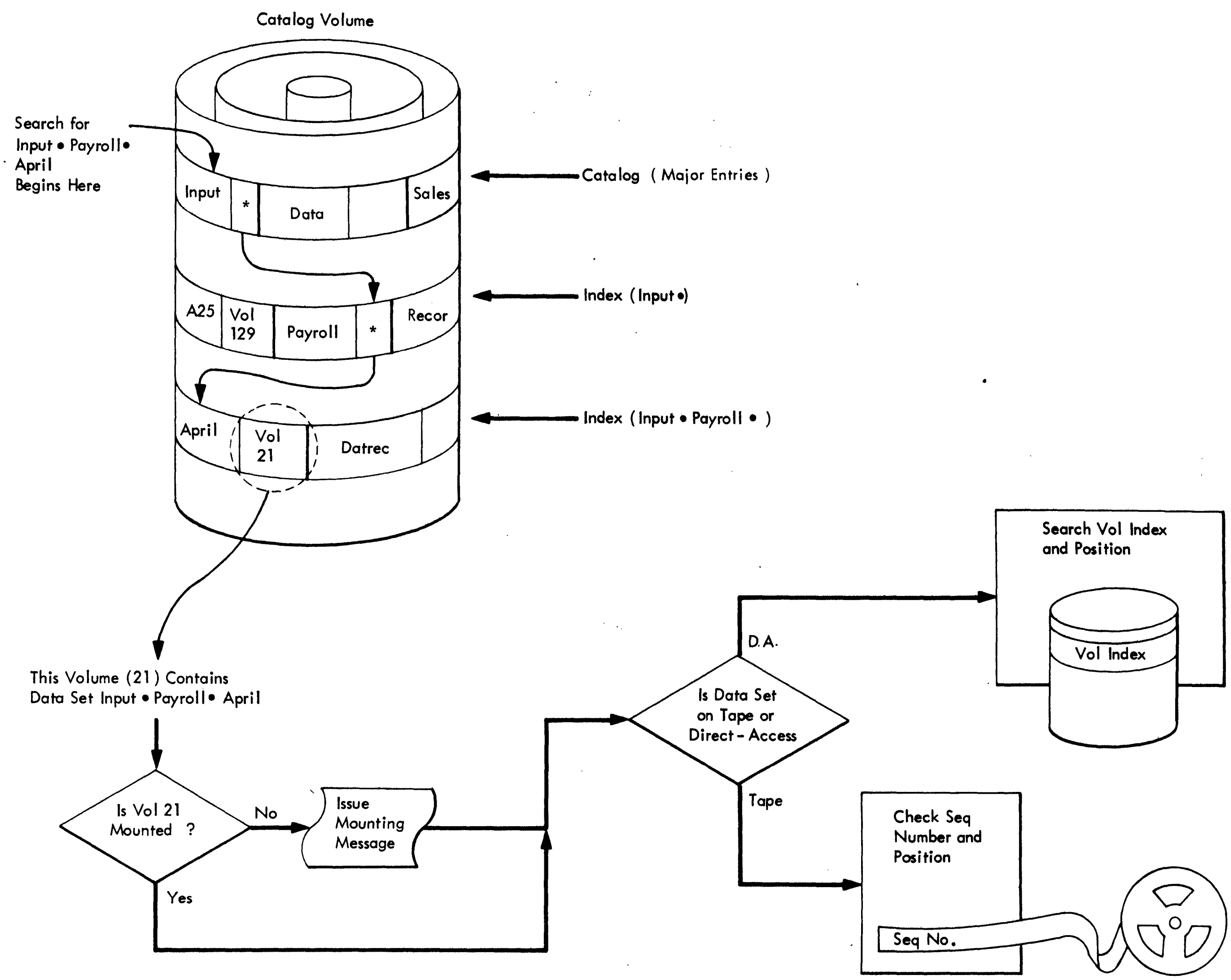
DESCRIBING A DATA SET



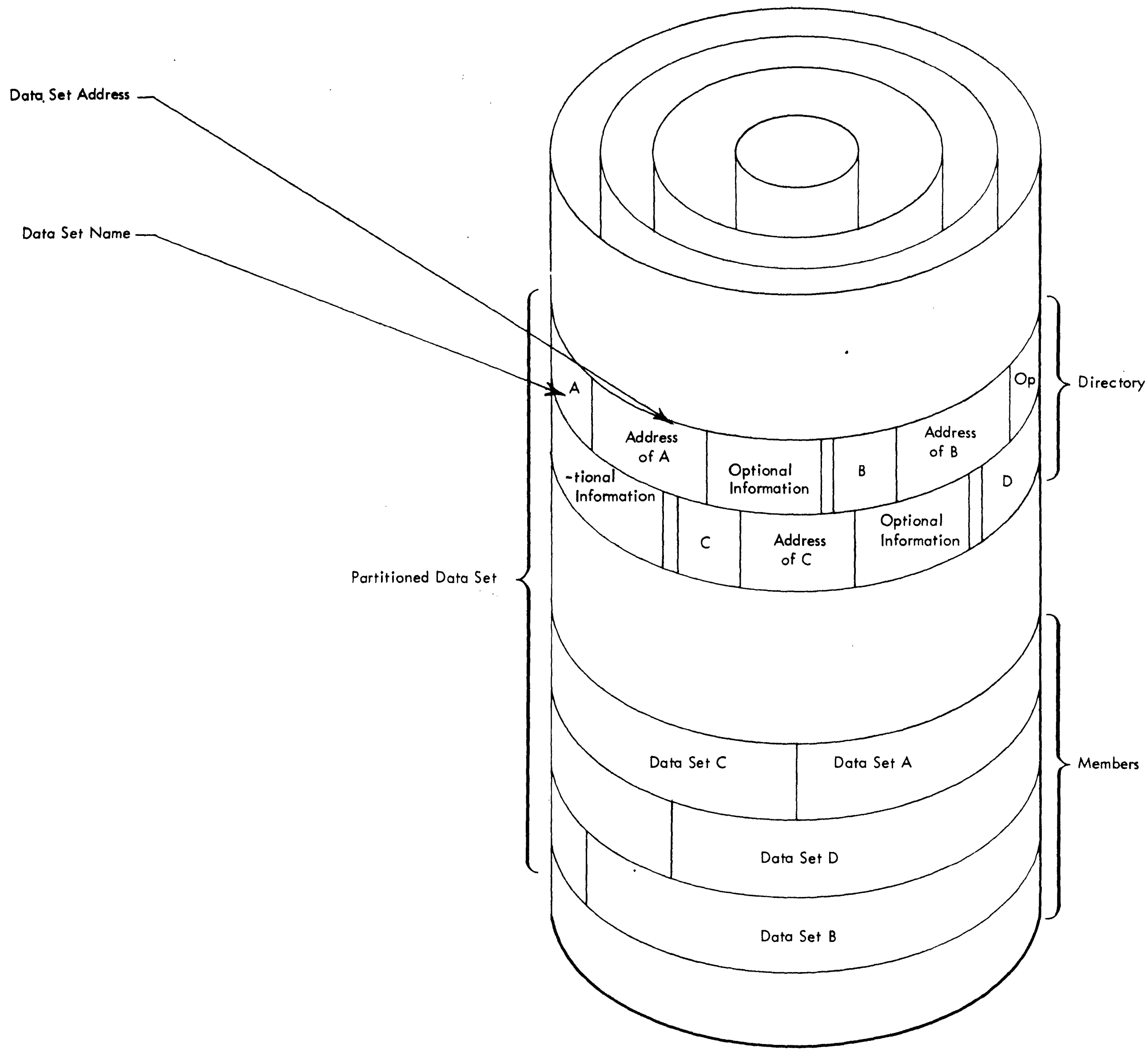
DIRECT-ACCESS LABEL



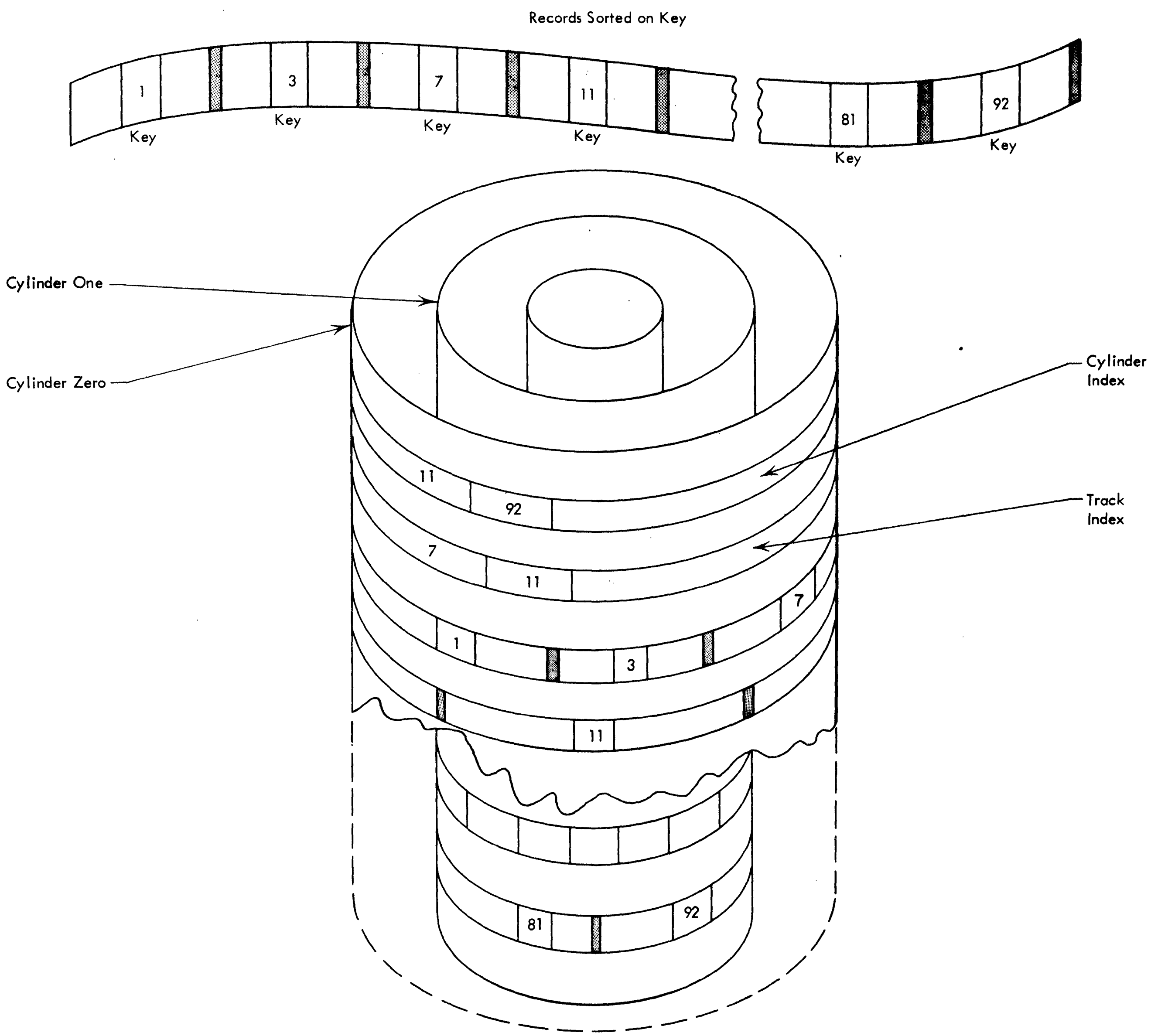
DATA SET RETRIEVAL THROUGH THE CATALOG



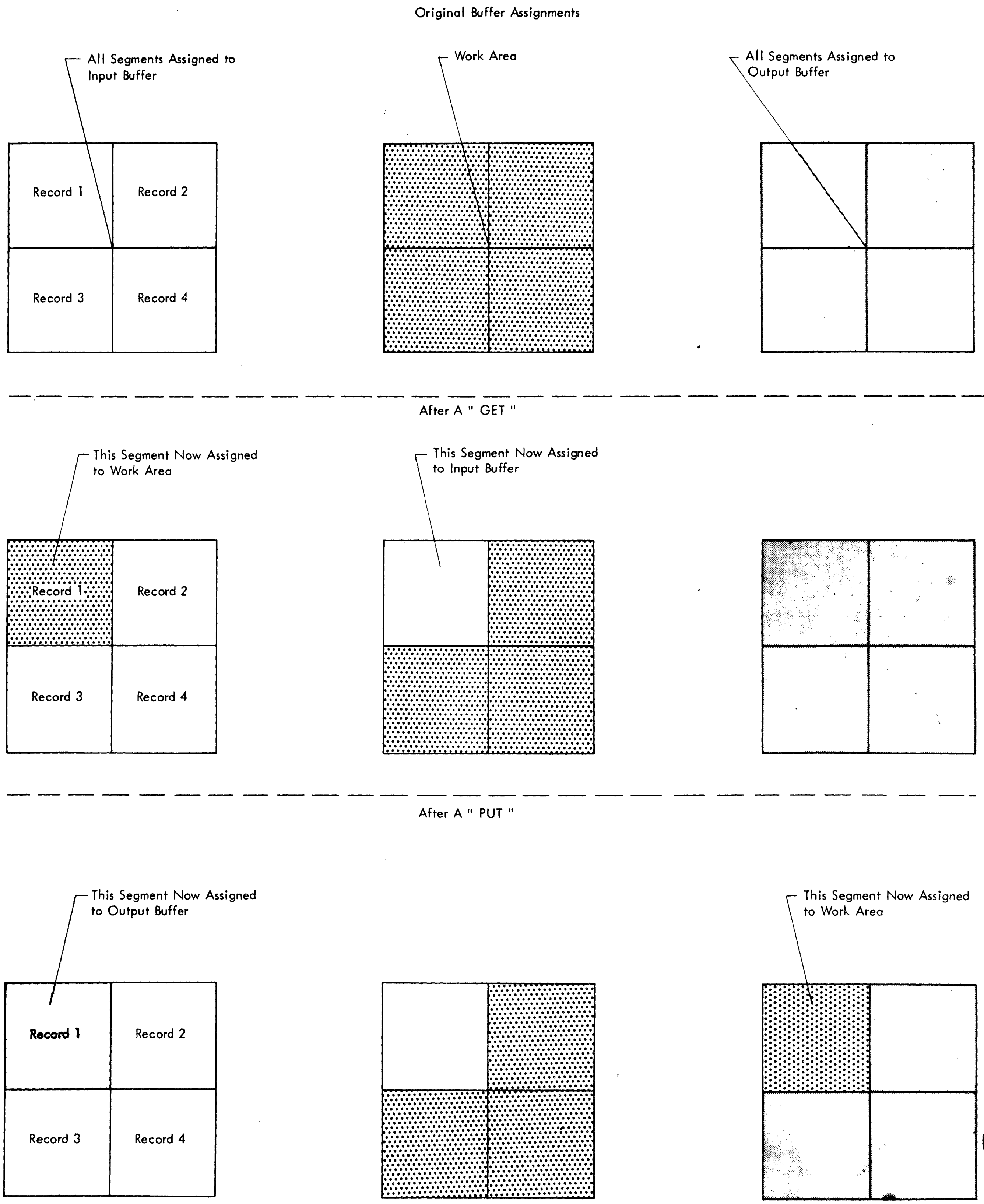
PARTITIONED DATA SET



INDEXED SEQUENTIAL DATA SET



EXCHANGE BUFFERING -- SUBSTITUTE MODE



ACCESS METHOD SUMMARY

Organization	Sequential		Partitioned	Indexed Sequential			Direct
				QISAM			
Access Method	QSAM	BSAM	BPAM	LOAD	SCAN	BISAM	BDAM
Primary macro instructions*	GET, PUT, PUTX	READ WRITE	READ,WRITE FIND,STOW	PUT	SETL,GET, PUTX	READ WRITE	READ WRITE
Synchronization of program with I/O	Automatic	CHECK	CHECK	Automatic	Automatic	WAIT	WAIT
Record format transmitted	Logical F,V Block U	Block F,V,U	Block (Part of member) F,V,U	Logical F,V	Logical F,V	Logical F,V	Block F,V,U
Buffer creation and construction	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic
Buffer technique	Automatic Simple Exchange	GETBUF FREEBUF	GETBUF FREEBUF	Automatic, Simple	Automatic Simple	GETBUF, FREEBUF Dynamic FREEDBUF	GETBUF, FREEBUF Dynamic FREEDBUF
Transmittal modes (work area/buffer)	Move, locate, substitute			Move, Locate	Move, Locate		

*All macro instructions introduced in this table are defined in the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647.

▲ JOB MANAGEMENT FUNCTIONS

- ANALYSIS OF INPUT STREAM (JCL)
- ALLOCATION OF I/O DEVICES
- OVERALL JOB SCHEDULING
- TRANSCRIPTION OF INPUT/OUTPUT DATA
- OPERATOR COMMUNICATIONS

▲ FEATURES OF JOB CONTROL LANGUAGE

- REFERENCING EXISTING STATEMENTS
- DATA SET NAME RETRIEVING
- OPTIMIZATION OF I/O
- PASSING DATA SETS AMONG JOB STEPS
- SHARING DATA SETS AMONG JOBS

TYPICAL JOB STATEMENTS

```
//    DEMO1  JOB  62-7
//    DEMO2  JOB  (131-22,AZ6), TOM, MSGLVL = 1
//    DEMO3  JOB  62-7, AL, PRTY = 13, REGION = 32K
//    DEMO4  JOB  135, JOE, COND= (12,GT)
```

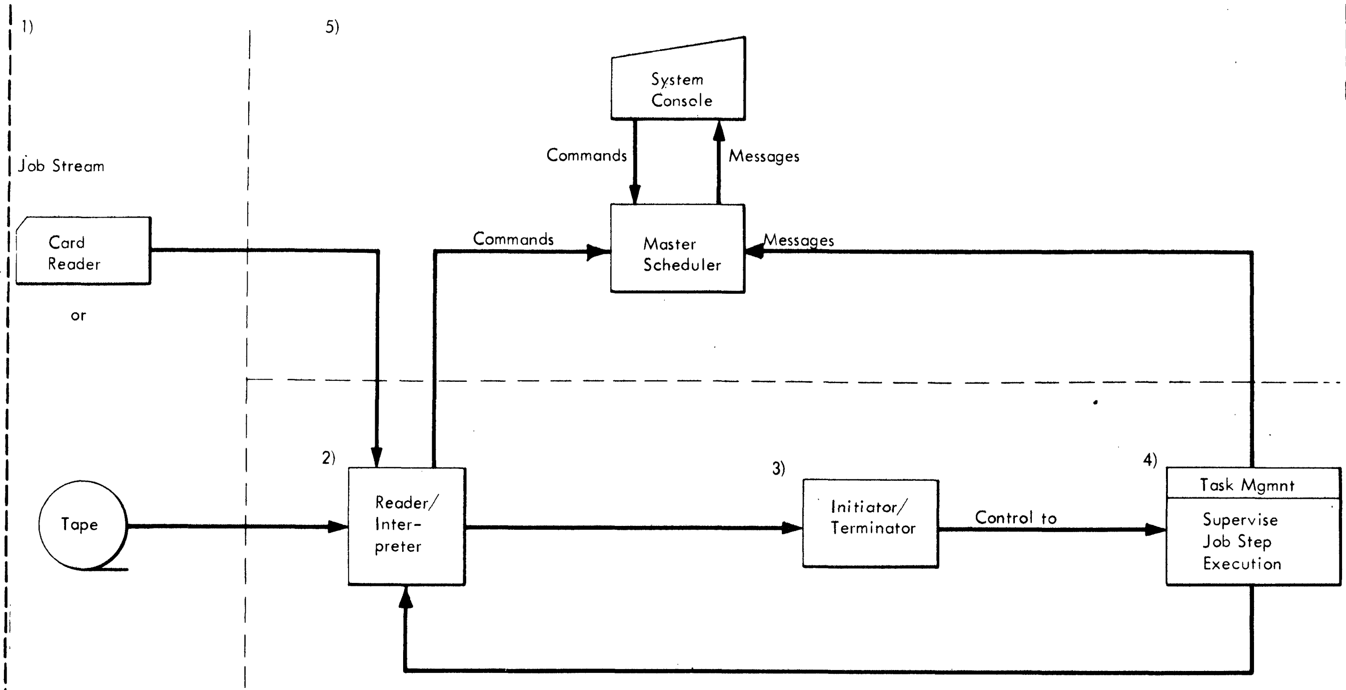
TYPICAL EXEC STATEMENTS

```
//   STEP 1   EXEC   PGM = MYCODE

//   STEP 2   EXEC   PGM = *.STEP6.MYDATA
//   STEP 2   EXEC   PGM = *.STEP7.PRSTEP2.YOURDATA
//   STEP 2   EXEC   PROC = CATPROC

//   STEP 3   EXEC   PGM = YOURCODE,COND = (17,EQ, STEP9)

//   STEP 4   EXEC   PGM = INTERP, TIME = (2,10),REGION = 64K
```

SOME TYPICAL DD STATEMENTS

```
//   MYDATA     DD   SYSOUT = Z
//   YOURDATA   DD   SYSOUT = 9,SPACE = (CYL,(7,1),RELSE,ROUND)
//   HISDATA    DD   UNIT = 180,DSNAME = HISSET, DISP = (CATLG, KEEP)
//   HERDATA    DD   UNIT = 2311, DSNAME = HERSET,DISP = (CATLG)
                     ,SPACE = (CYL,3,,,,ROUND)
//   OURDATA    DD   DSNAME = OURSET, DISP = MOD, UNIT = TAPE, DEFER
//   OLDDATA    DD   DSNAME = OLDSET,DISP =OLD, VOLUME = PRIVATE, RETAIN
//   PASSDATA   DD   DSNAME =*.STEP3.HISDATA,DISP = (OLD,PASS)
```

A SEQUENTIAL SCHEDULING SYSTEM



1. Your programs, in the form of jobs or job steps defined through the job control language, may enter the system in the input stream from a card or tape device. Input data may be entered into the system with the control statements.
2. The reader/interpreter reads in the control statements for <u>one</u> job step.
3. The initiator/terminator allocates the required I/O devices, notifies the operator of volumes to be mounted (if any), and requests the task management programs to supervise execution of the named job step.
4. The task management programs turn control over to the first load module and supervise its execution.
5. The master scheduler accepts and takes action on commands.

A PRIORITY SCHEDULING SYSTEM

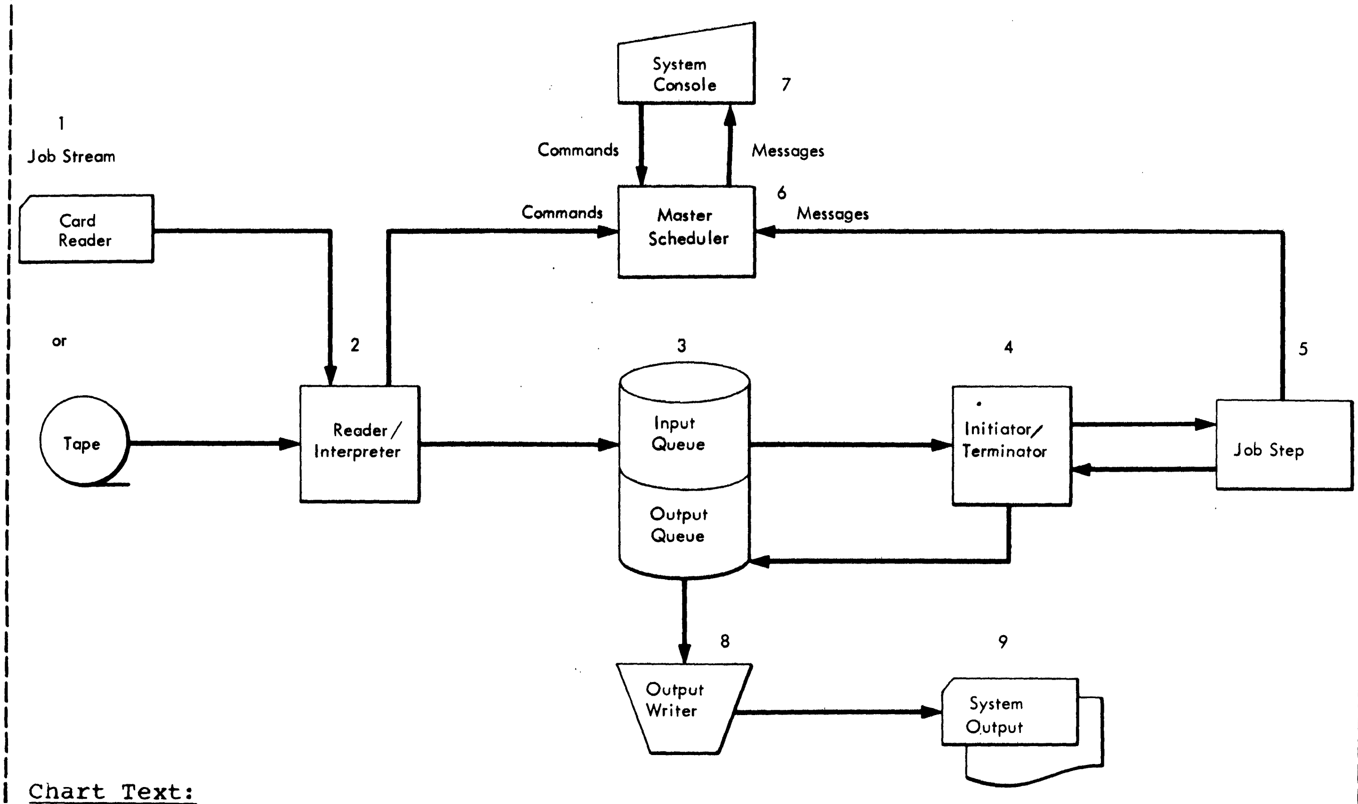
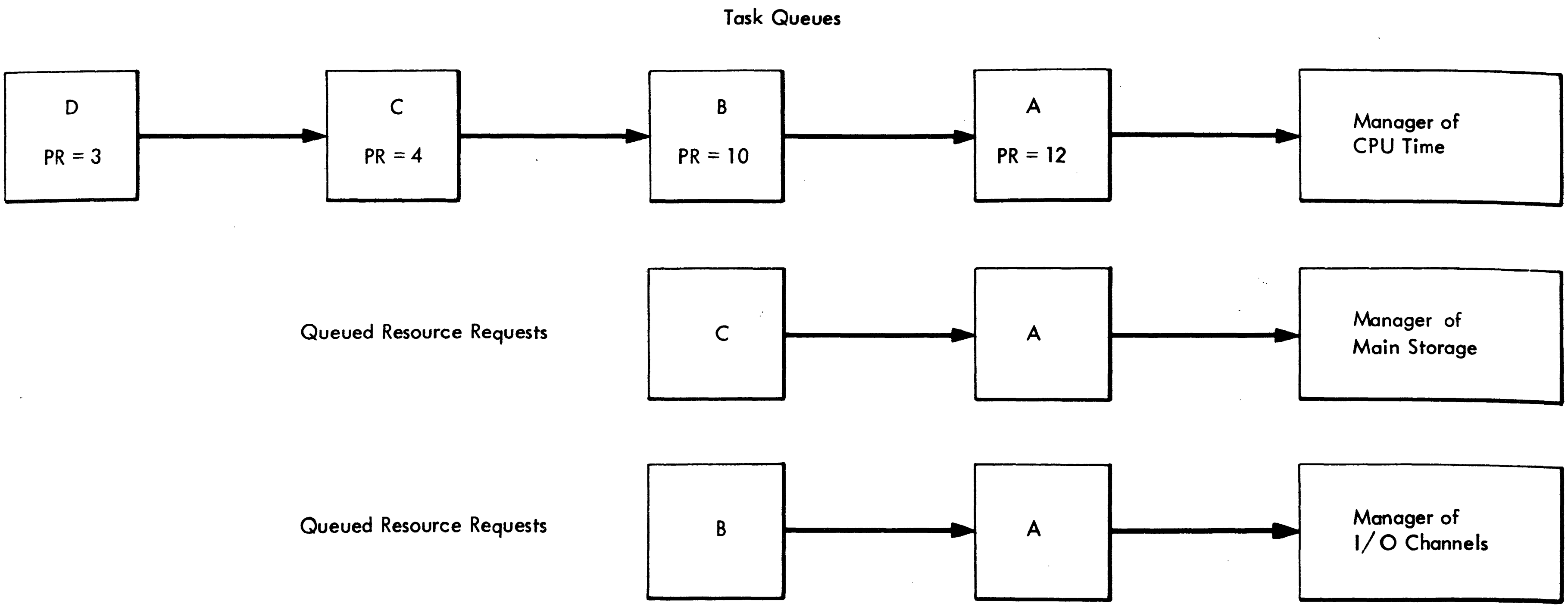


Chart Text:

1. Your programs, defined as jobs or job steps by the job control language, enter the system through the input stream from a card or tape device.
2. The reader/interpreter reads in control statements for one or <u>more</u> jobs and places them, by priority, on the input work queue.
3. The job with the highest priority is selected for execution by the initiator/terminator.
4. The initiator/terminator turns your job step over to the task management programs, which supervise its execution.
5. The master scheduler accepts and takes action on commands.
6. Output is written (by job step priority) when the job has terminated and while other jobs are being processed.

RESOURCE QUEUES



▶ TOPICS FOR THIS SESSION

SCRATCH PAD MEMORY

COMPUTER NETWORKS

UNIVAC 1108

SYSTEM APPROACHES TO HIGH PERFORMANCE MACHINES.

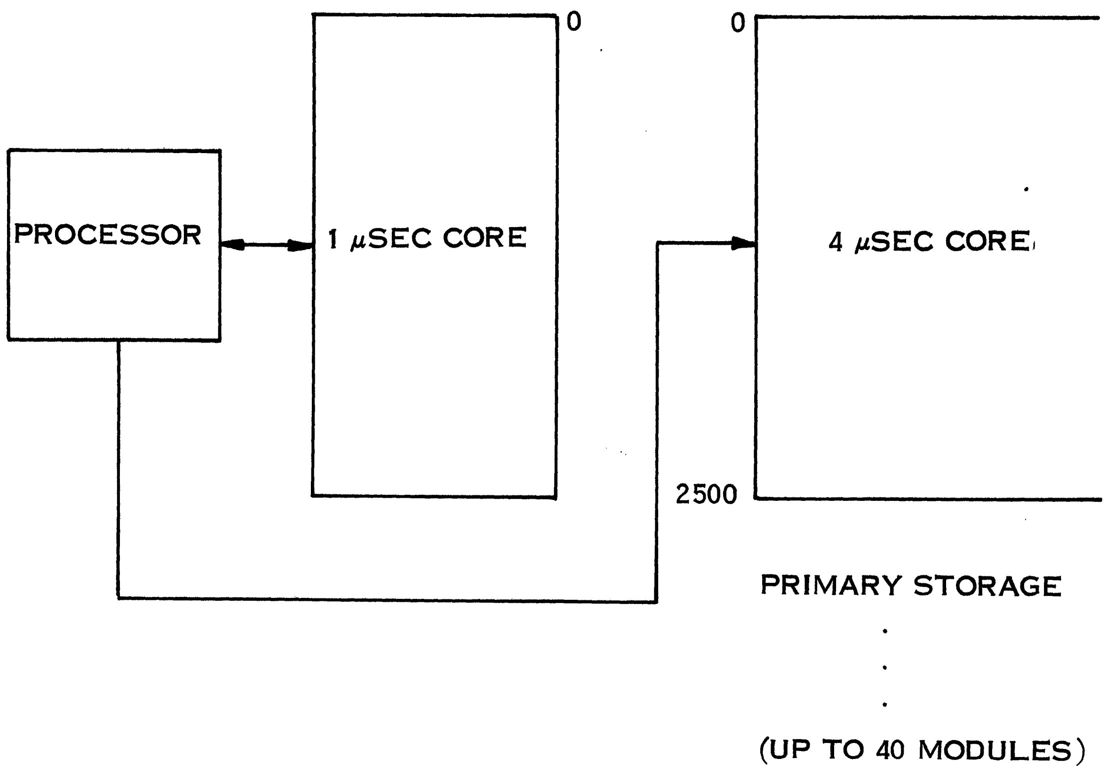
DEFINITION OF SCRATCH PAD MEMORIES

SCRATCH—PAD MEMORIES:

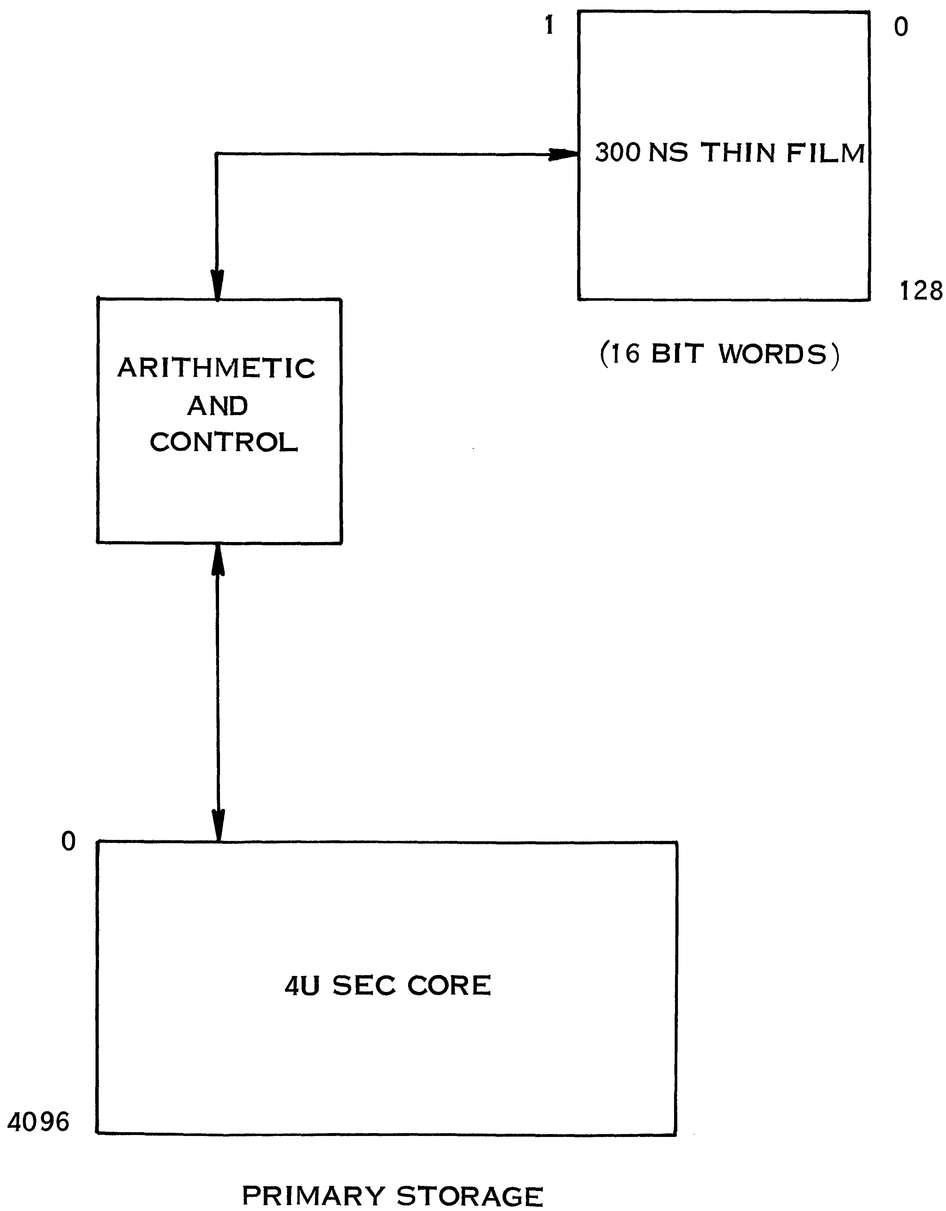
SMALL, LOGIC—SPEED MATCHED MEMORIES USED FOR REGISTERS AND/OR VERY HIGH SPEED WORKING STORAGE.

UNIVAC LARC

ACCUMULATOR OR INDEX REGISTER



D825 WITH THIN FILM REGISTER MEMORY

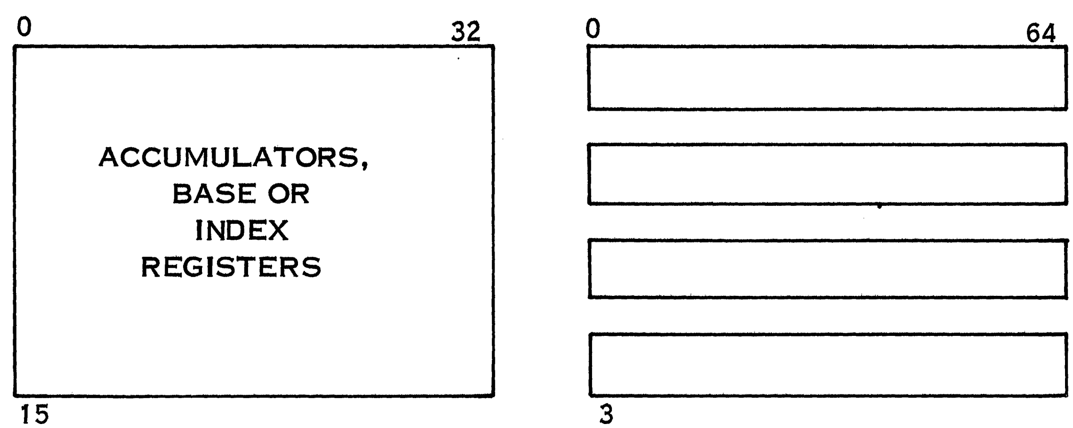


D825 THIN FILM REGISTERS

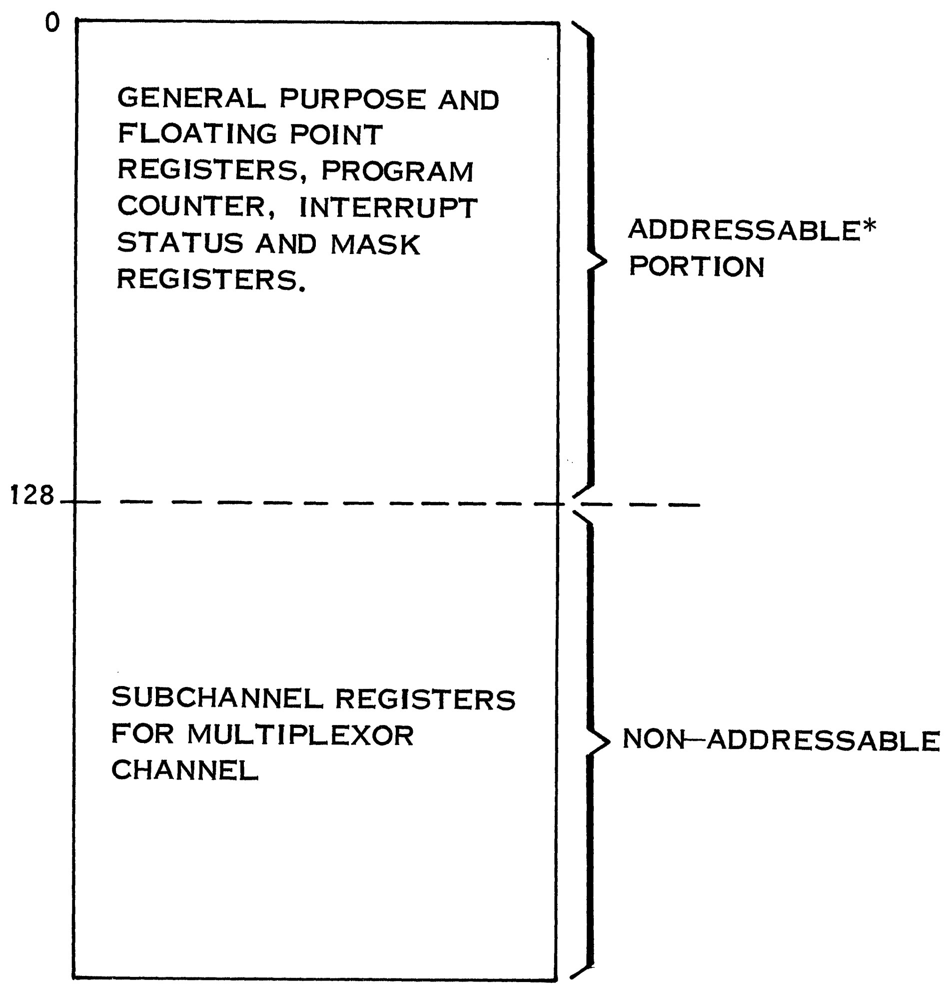
PROGRAM STORAGE REGISTER 1	(48)
PROGRAM STORAGE REGISTER 2	(48)
INTERRUPT PROGRAM REGISTER	(48)
REAL–TIME CLOCK	(24)
REPEAT COUNT REGISTER	(12)
INDEX INCREMENT REGISTER	(12)
CHARACTER COUNT REGISTER	(12)
3 REPEAT INCREMENT REGISTERS	(12 EA)
T–F C REGISTER	(48)
STACK 1	(48)
STACK 2	(48)
STACK 3	(48)
STACK 4	(48)

INTERRUPT STORAGE REGISTER	(48)
SUBROUTING STORAGE REGISTER	(48)
REPEAT PROGRAM REGISTER	(64)
INTERRUPT DUMP REGISTER	(16)
POWER FAILURE DUMP REGISTER	(32)
PROGRAM COUNT REGISTER	(16)
BASE PROGRAM REGISTER	(16)
BASE ADDRESS REGISTER	(16)
SUBROUTINE BASE ADDRESS REGISTER	(16)
INTERRUPT BASE ADDRESS REGISTER	(16)
15 INDEX REGISTERS	(16 EA)
15 LIMIT REGISTERS	(16 EA)

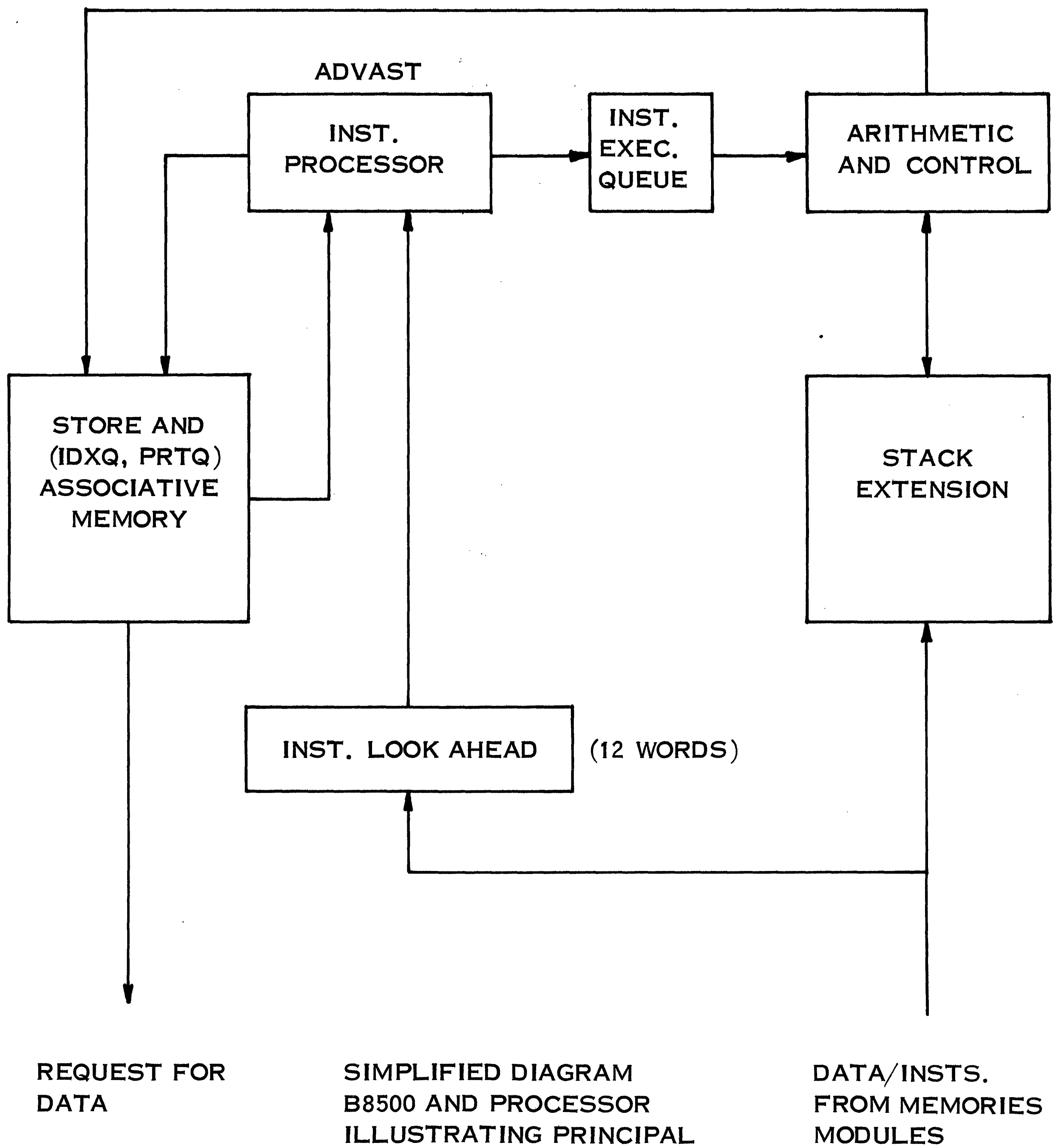
360 GENERAL REGISTERS



SPECTRA 70/35 SCRATCH—PAD MEMORY

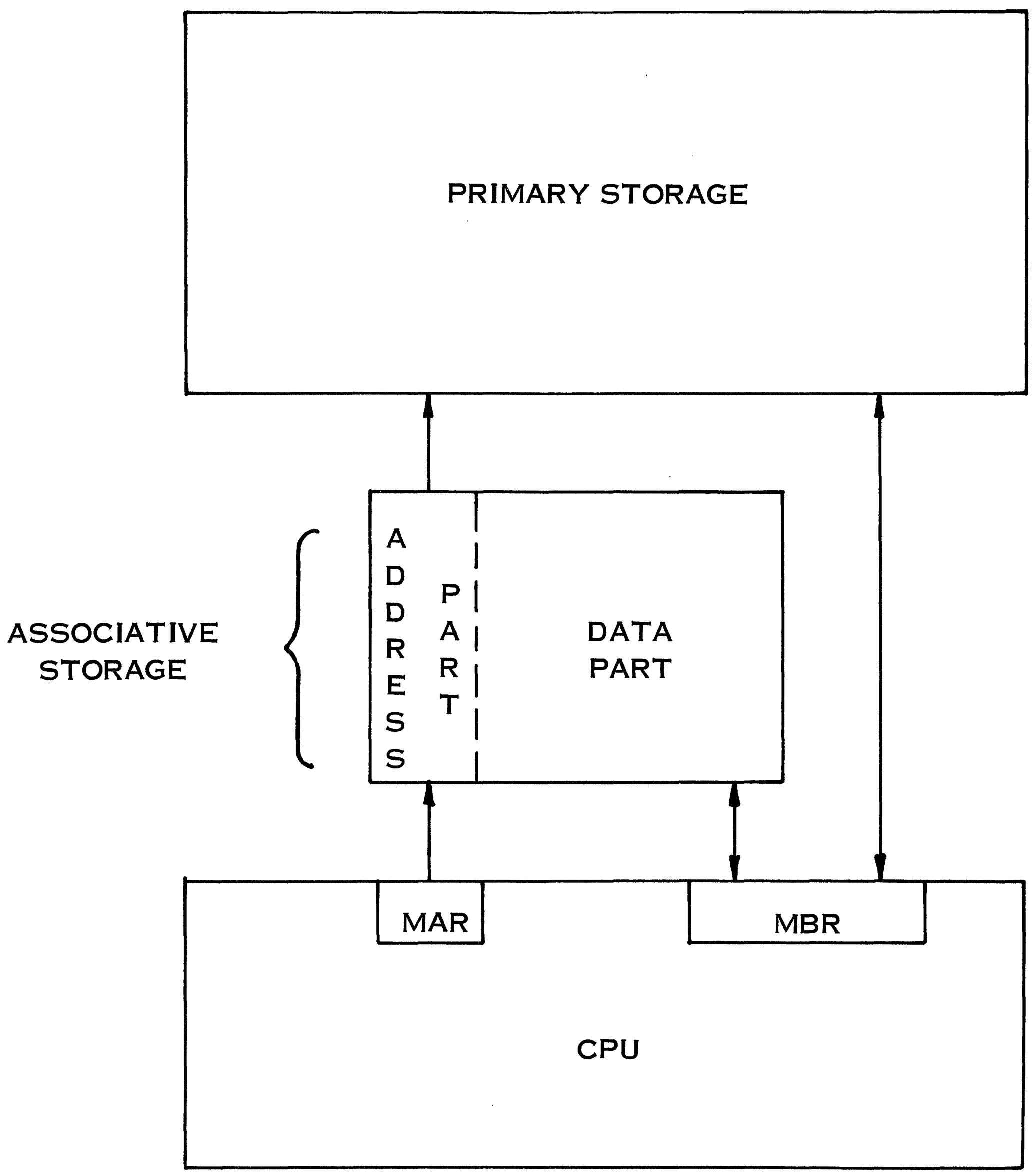


*IMPLEMENTED AS A SEPARATE MEMORY ON 70/45, 70/55.



SIMPLIFIED DIAGRAM
B8500 AND PROCESSOR
ILLUSTRATING PRINCIPAL
SCRATCH—PAD MEMORIES

LOOK–ASIDE MEMORY



SUMMARY OF SCRATCH–PAD CHARACTERISTICS

- PRIMARY FUNCTIONS

  CLOSE–IN STORAGE MATCHED TO LOGIC SPEEDS, INEXPENSIVE IMPLEMENTATION OF CONTROL REGISTERS, MASK REAL SPEED OF PRIMARY STORAGE

- POTENTIAL PROBLEMS

  CONTENTS OF SCRATCH–PAD BECOMES PART OF THE STATE OF AN ACTIVE PROCESS

- SOLUTIONS

  ASSOCIATIVE STORE

  MULTIPLE SCRATCH–PAD

TYPES OF COMPUTER NETWORKS

- DEDICATED

  COMMUNICATIONS SWITCH

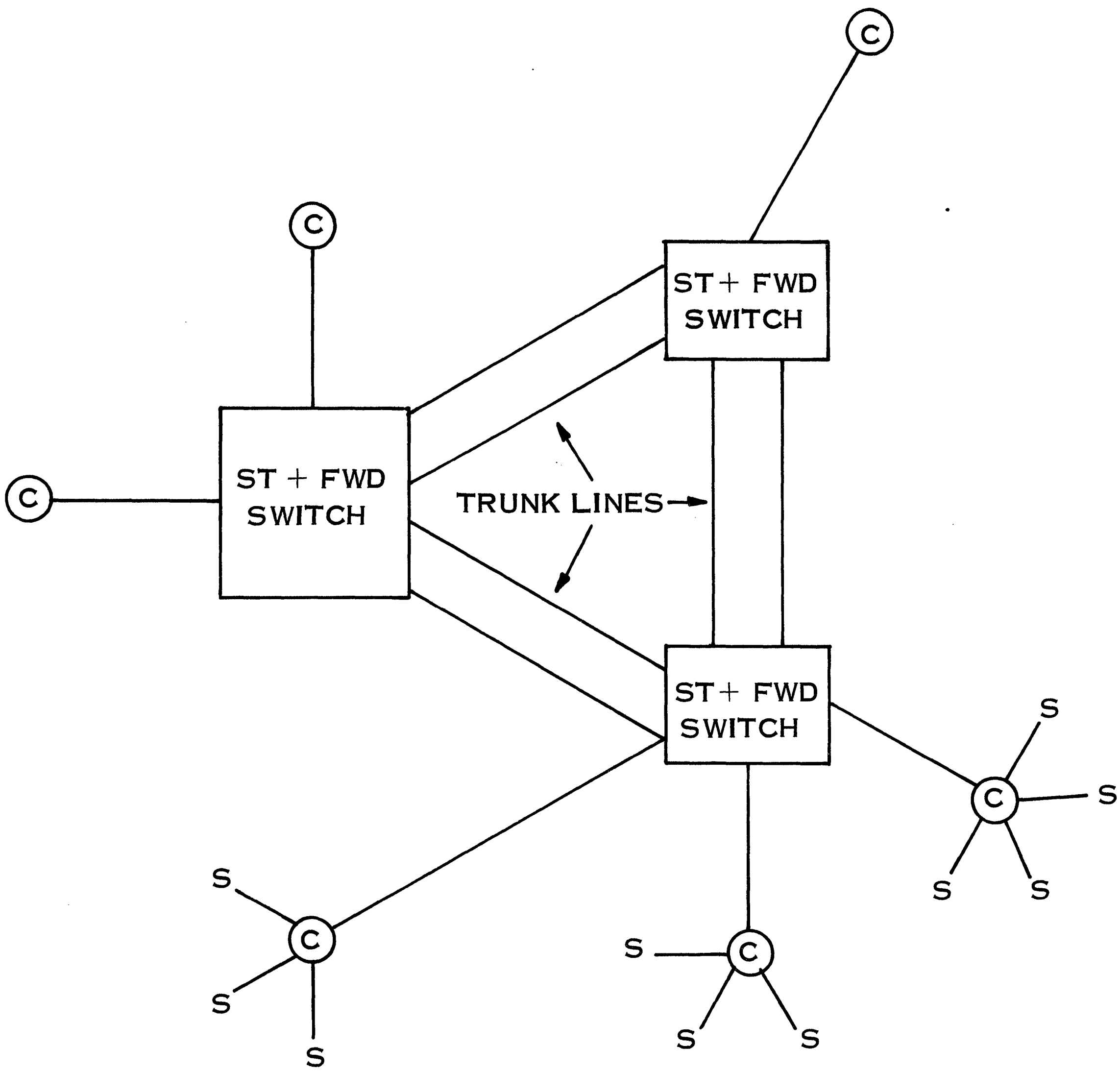
  RESERVATION SYSTEMS

  AIR DEFENSE SYSTEMS

- LOAD—SHARING

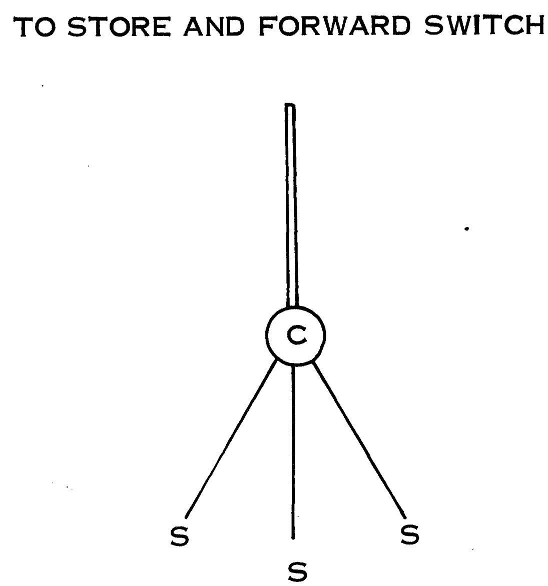
  REMOTE COMPUTING

COMMUNICATIONS SWITCHING SYSTEM

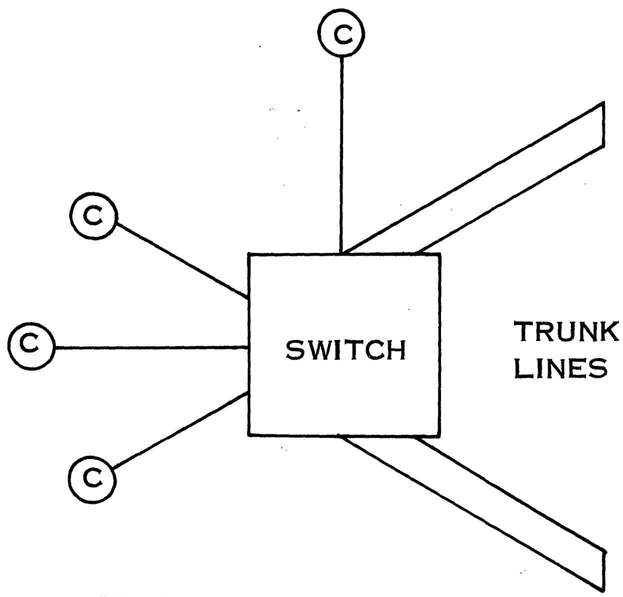


FUNCTIONS OF CONCENTRATOR

- IT'S A COMPUTER
- SPEED MATCHING
- BUFFER FOR ECONOMICAL TRANSMISSION TO SWITCH
- LOCAL DISTRIBUTION



FUNCTIONS OF SWITCH



MULTIPLE–ADDRESS ROUTING

STORE AND FORWARD NETWORK PROBLEMS

- RELIABILITY

  MULTIPLE COMMUNICATIONS PATHS

  MULTIPROCESSOR OR MULTICOMPUTER ELEMENTS

  TRANSMISSION CONTROL

  CHECKING

  DISTRIBUTED CONTROL

- LONG TERM STORAGE

  MULTIPLE ADDRESS MESSAGES

  STATION LOGS

- EFFICIENT PROCESSING

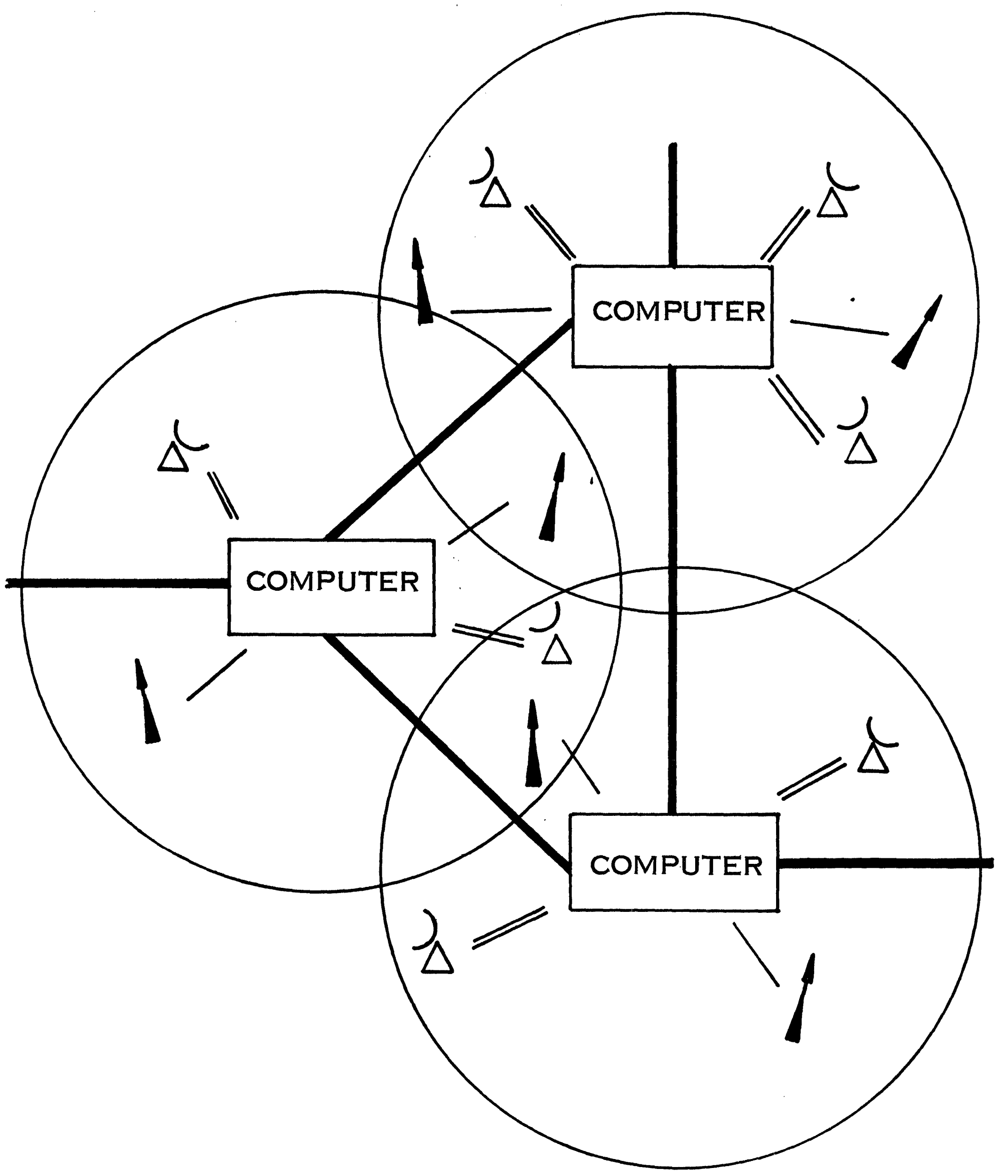
  INDEPENDENT I/O (COMMUNICATIONS) CHANNELS

- PEAK LOADS

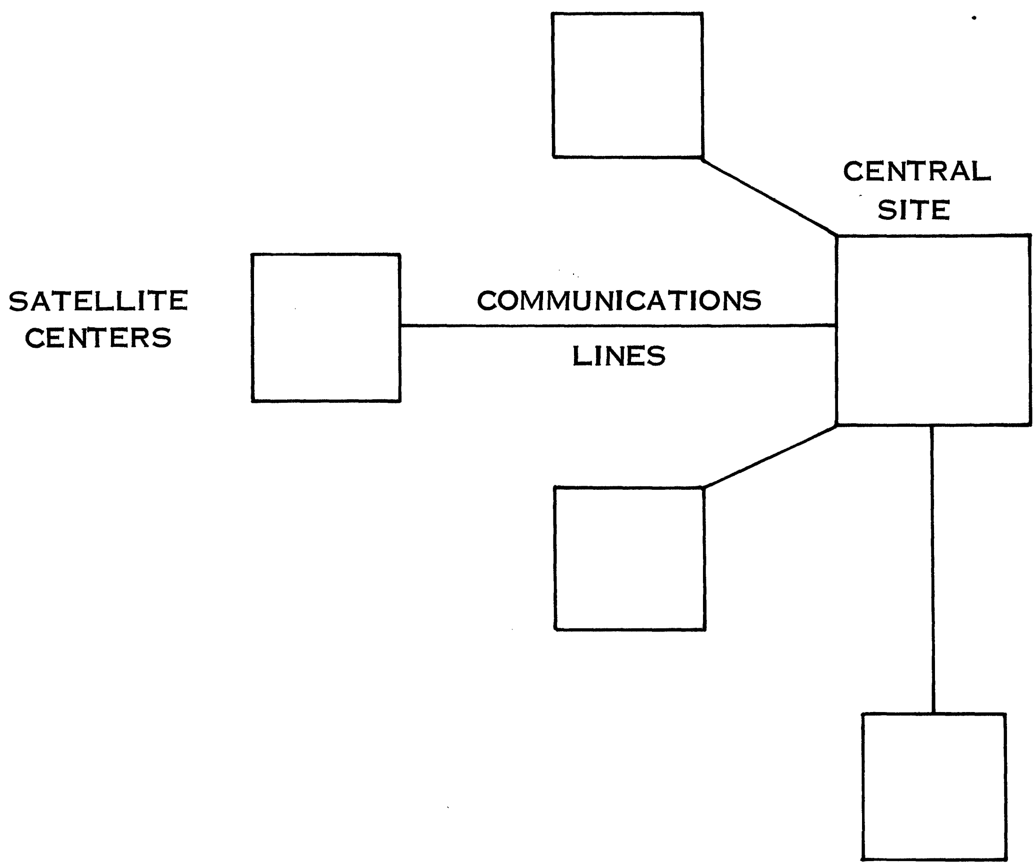
  SUFFICIENT SECONDARY STORAGE FOR BUFFERING
  DISC/DRUM

  TAPE

AIR–DEFENSE NETWORK



LOAD SHARING NETWORK



TYPES OF LOAD–SHARING

- REMOTE JOB PROCESSING
  - SATELLITES ACCEPT DATA AND CONTROL INFORMATION
  - CENTER QUEUES JOB FOR EXECUTION
  - CENTER RETURNS RESULTS TO SATELLITE
  - SATELLITE PRINTS RESULT
  - GENERALIZATION OF DCS CONCEPT
- ACTIVE SATELLITES
  - SMALL JOBS PERFORMED IN SATELLITE
  - LARGE (FOR SATELLITE) JOBS PERFORMED REMOTELY
- FULL SHARING
  - 2 OR MORE CENTERS
  - ALL JOBS DONE AT CENTER
  - OVERLOAD AT ONE CENTER TRANSMITTED TO ANOTHER

SOME LOAD SHARING PROBLEMS

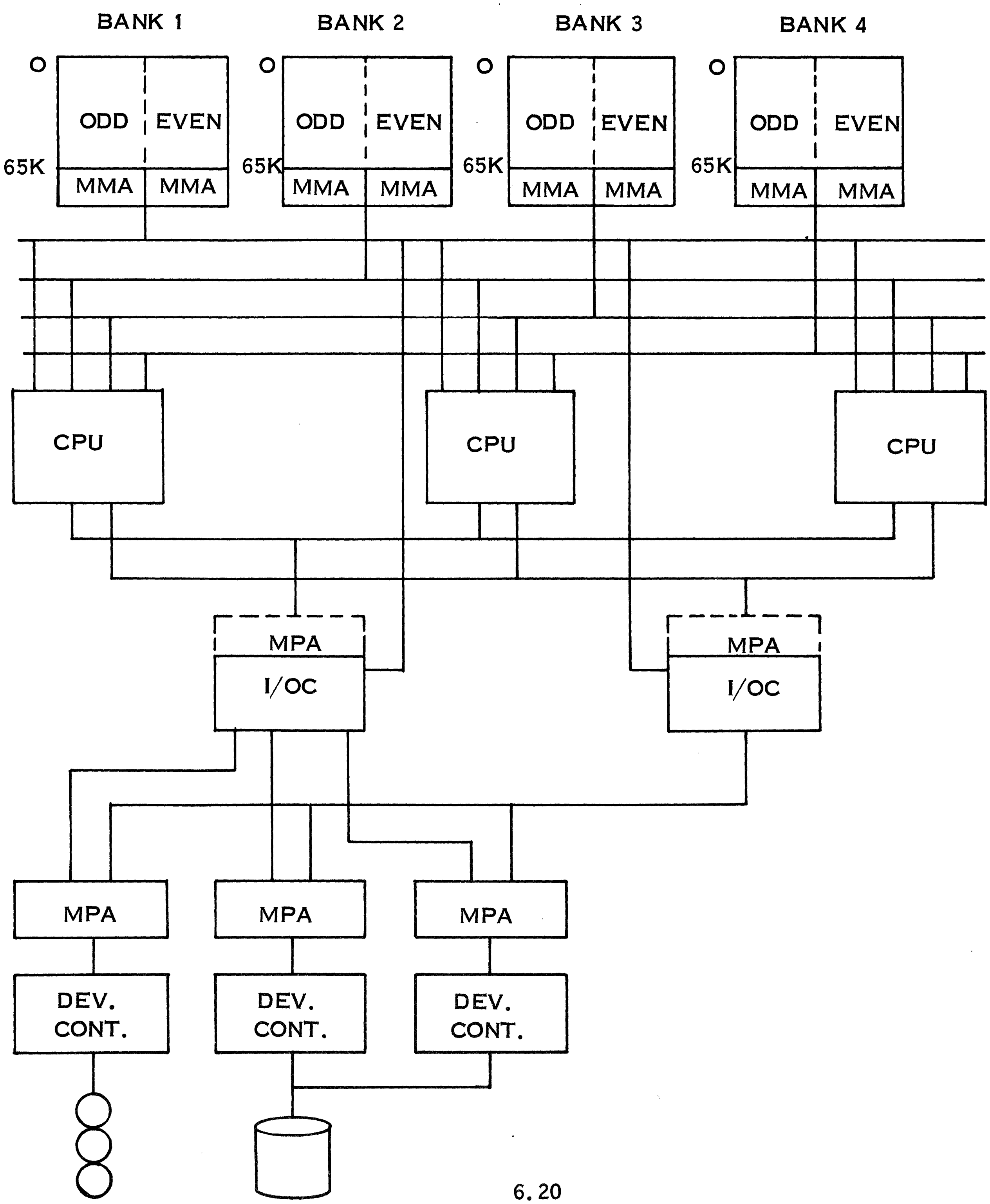
EQUIPMENT AND CONFIGURATION COMPATIBILITY

PROGRAM AND DATA LOCATION

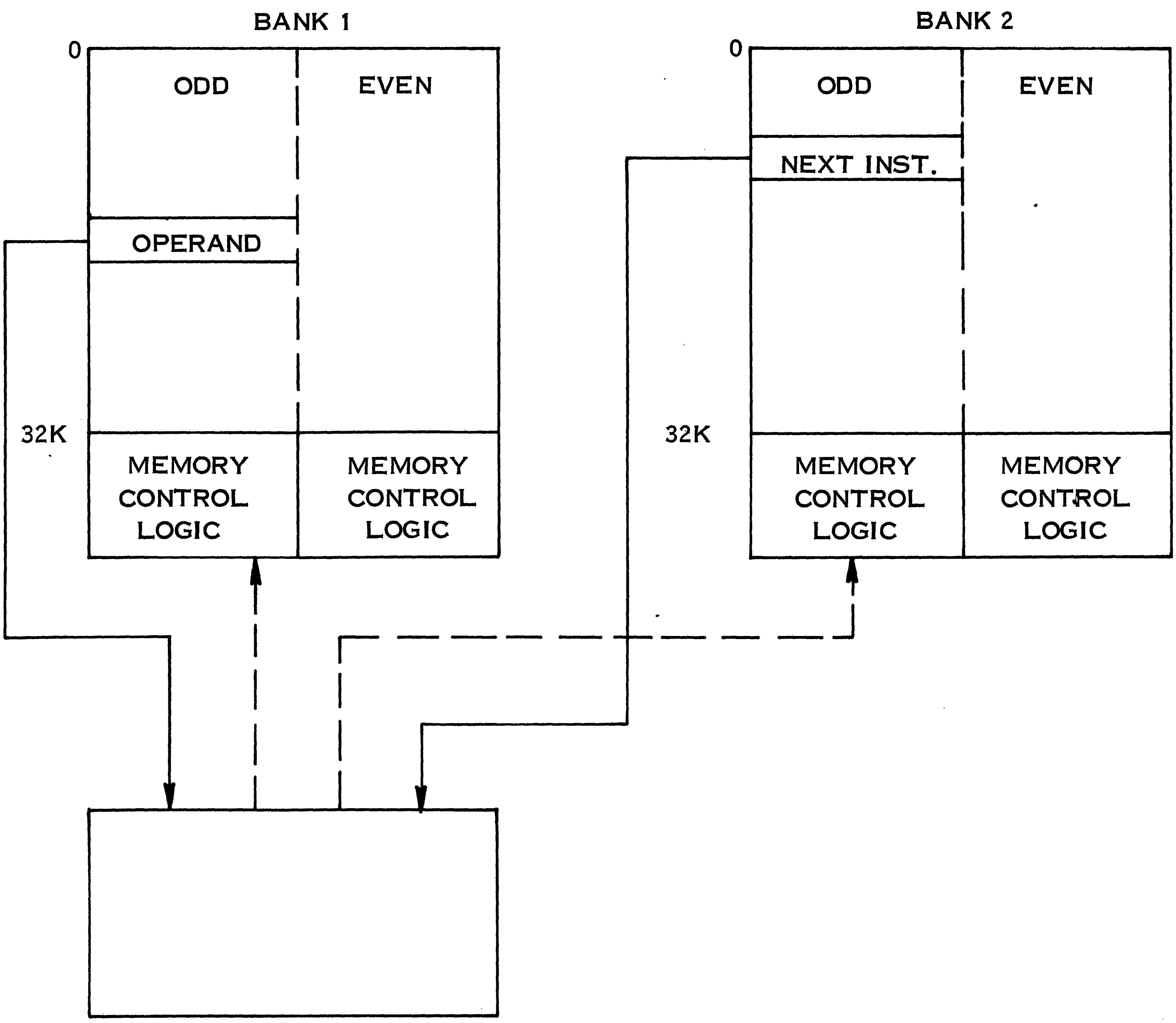
AUTONOMOUS CENTERS

ALL THE COMMUNICATIONS PROBLEMS

UNIVAC 1108 SIMPLIFIED MULTIPROCESSOR CONFIGURATION

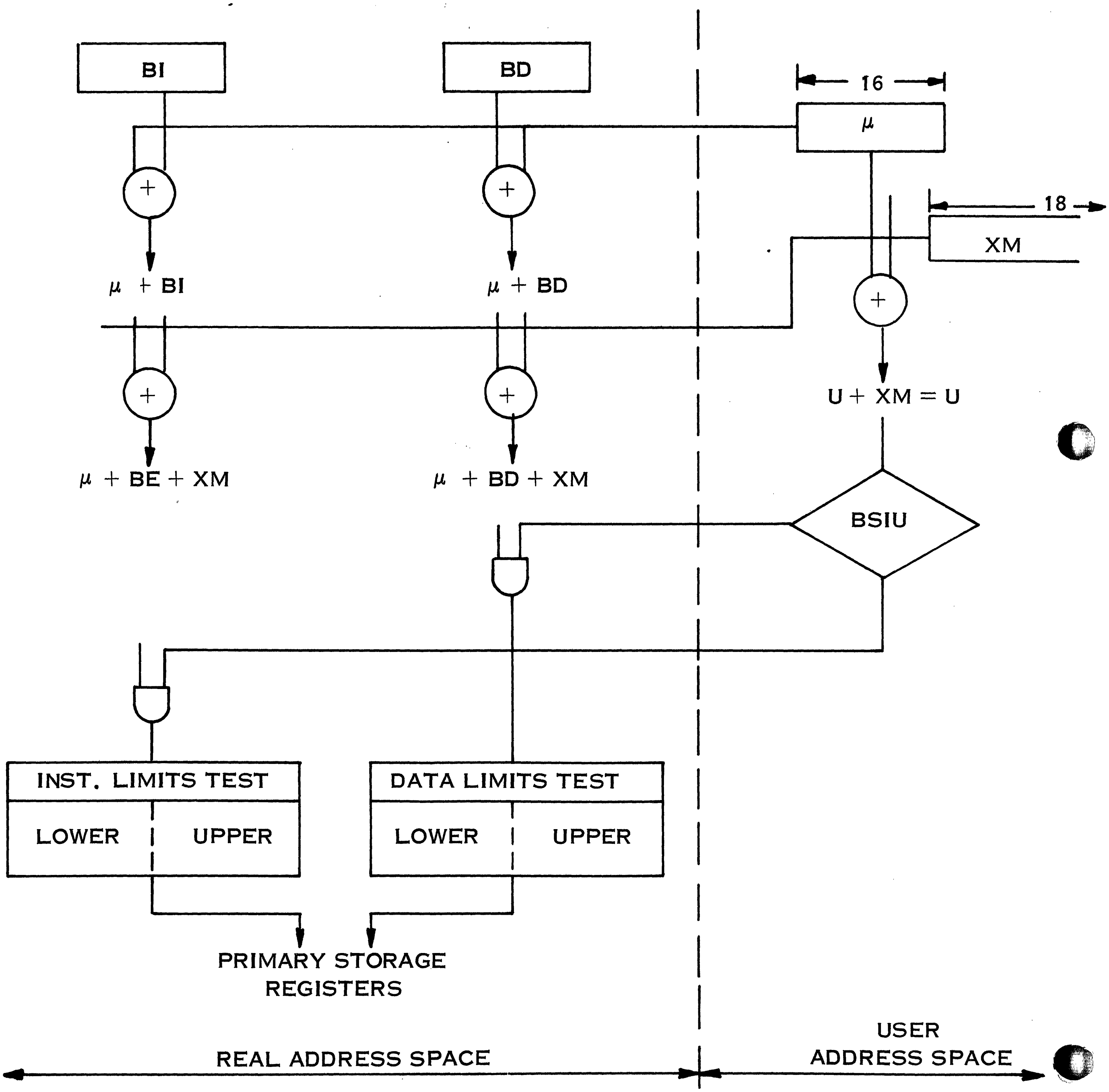


OVERLAPPED FETCH IN UNIVAC 1108

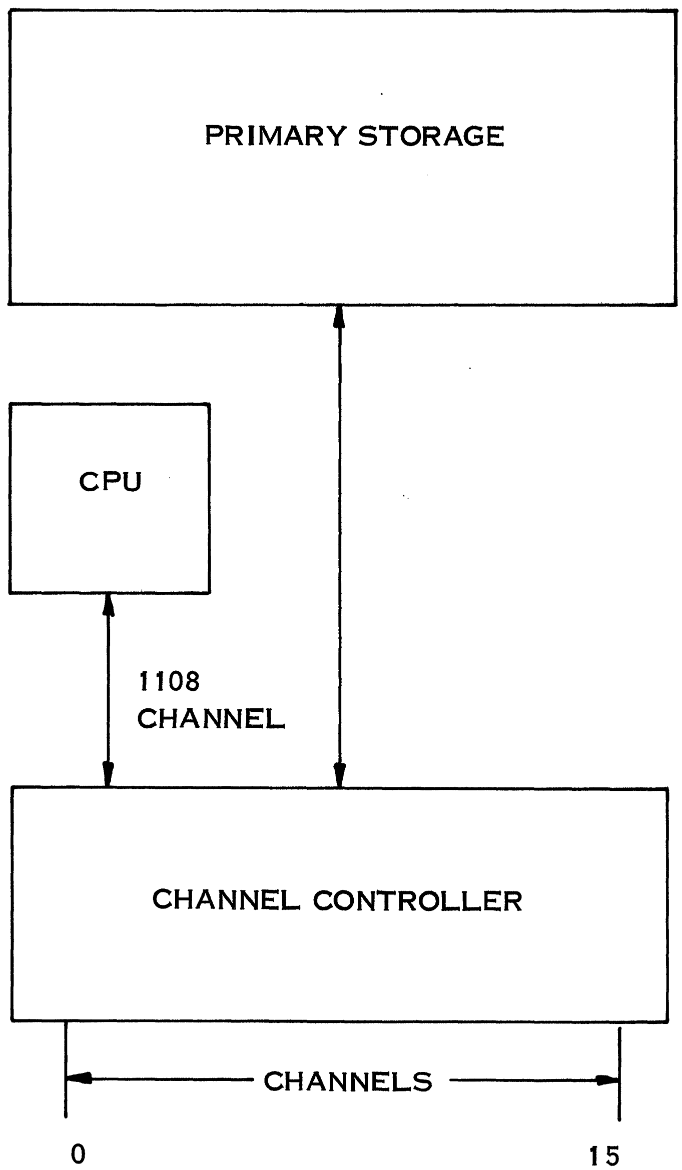


PROCESSOR

ADDRESSING AND STORAGE PROTECTION — UNIVAC 1108 — SIMPLIFIED



1108 I/O


PRIMARY STORAGE
CPU
1108
CHANNEL
CHANNEL CONTROLLER
CHANNELS
0
15

1108 AS A MULTIPROCESSOR

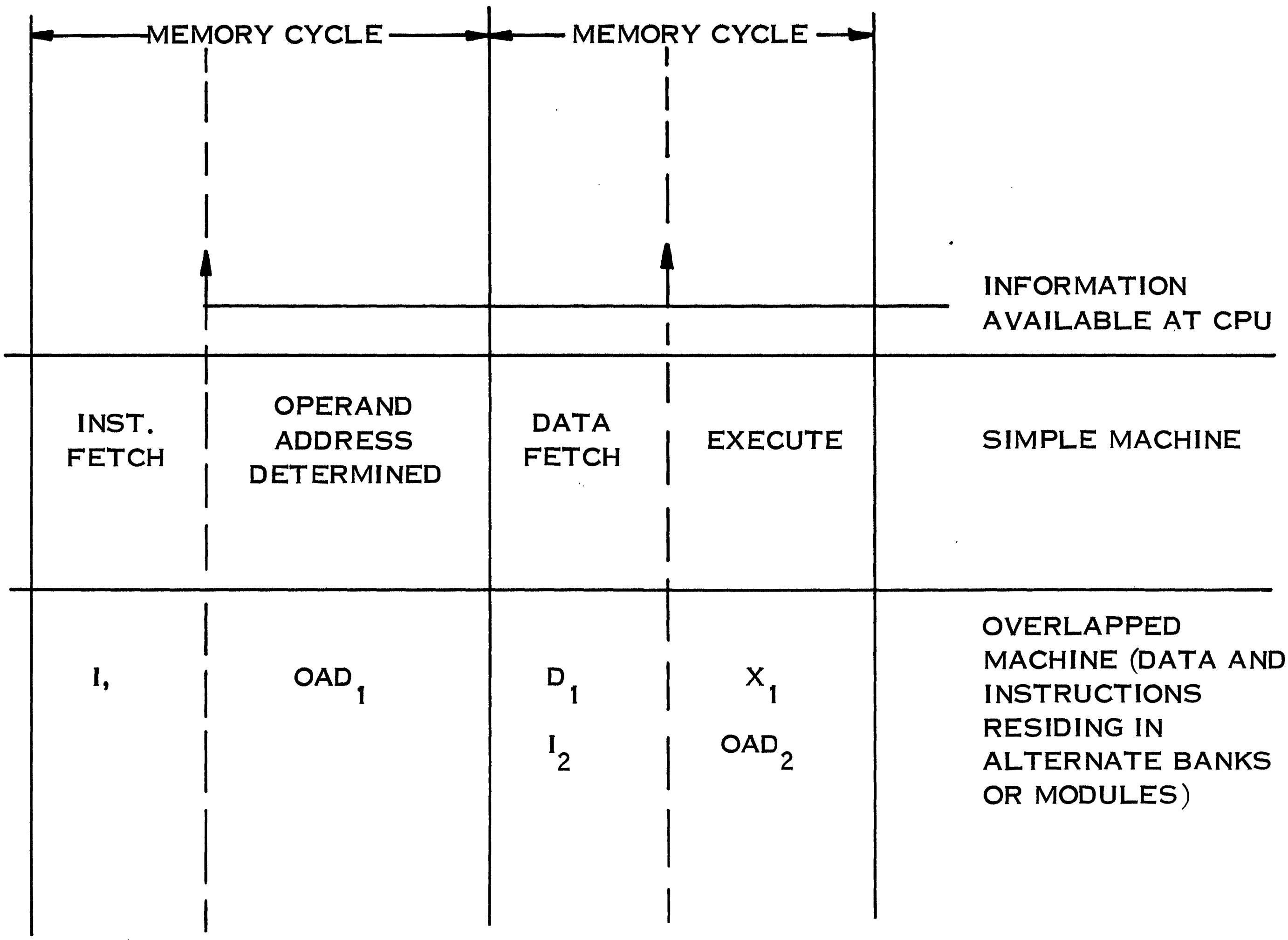
- DESIGNED AS A UNIPROCESSOR
- MULTIPROCESSING CONNECTIONS THROUGH ADAPTORS

  MMA
  MPA
- FULL 1107 COMPATIBILITY
- GUARD MODE ≡ USER STATE (MODE)
- SEPARATE PROGRAM AND DATA AREA BOUNDS REGISTERS
- I/O OPERATES WITHOUT STORAGE PROTECT FEATURE
- ADDITIONAL MODULE FOR MULTIPROCESSOR SYSTEMS – AVAILABILITY CONTROL UNIT

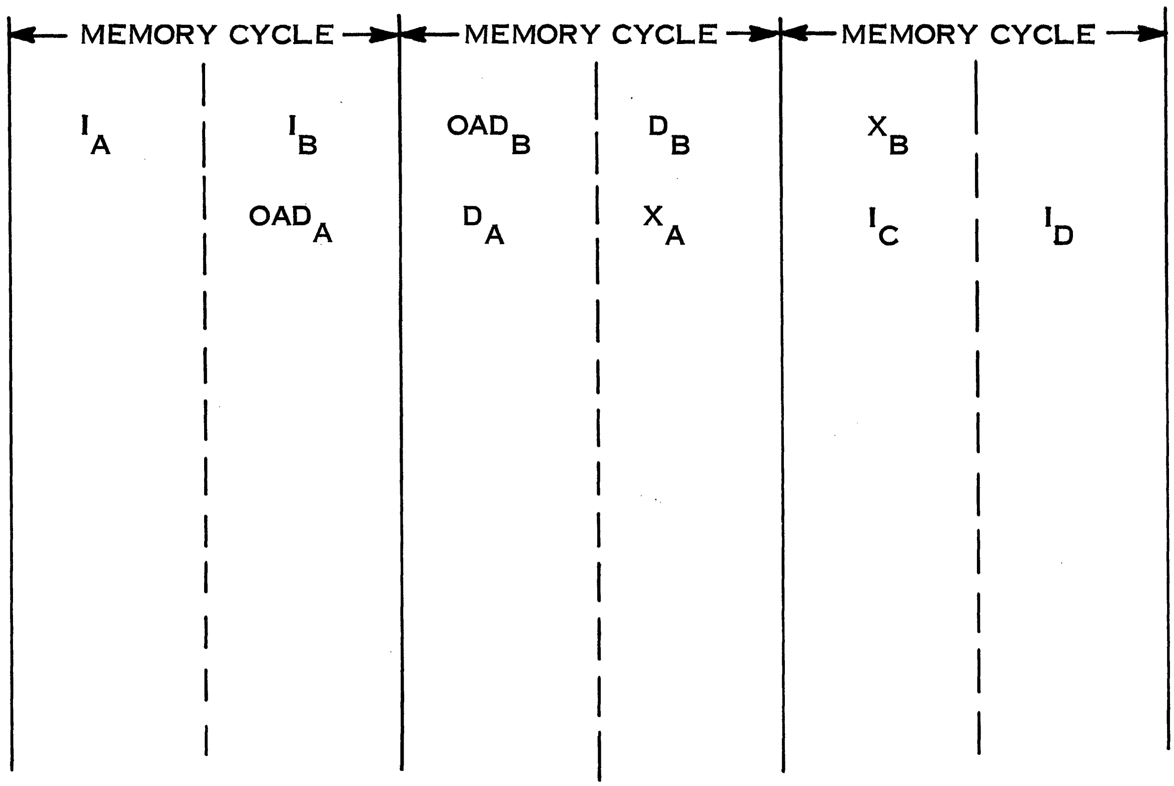
PROBLEMS IN ATTAINING HIGH PERFORMANCE SYSTEMS

- EXTREME MISMATCH BETWEEN SPEED OF LOGIC AND PRIMARY STORAGE
- MISMATCH BETWEEN PRIMARY AND SECONDARY STORAGE
- SERIAL REPRESENTATION OF PROGRAMS

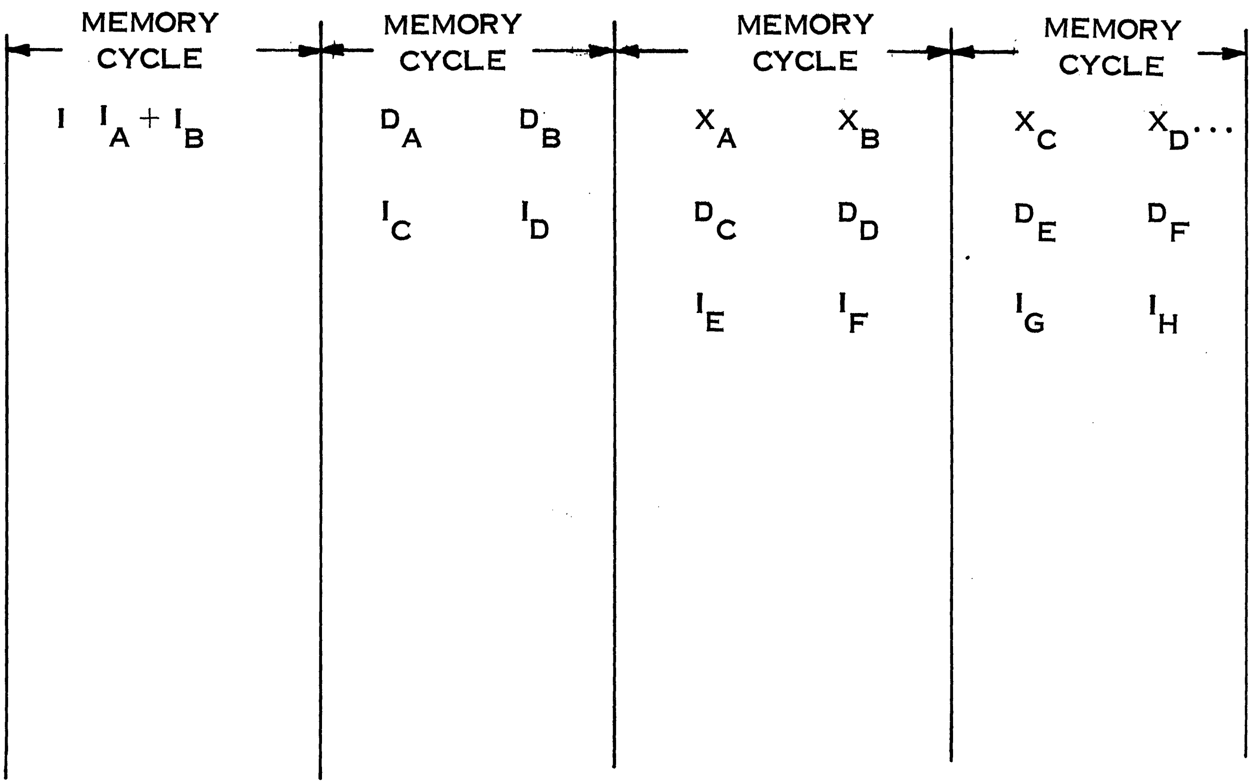
OVERLAP


MEMORY CYCLE
MEMORY CYCLE
INFORMATION AVAILABLE AT CPU
INST. FETCH
OPERAND ADDRESS DETERMINED
DATA FETCH
EXECUTE
SIMPLE MACHINE
I,
OAD1
D1
X1
I2
OAD2
OVERLAPPED MACHINE (DATA AND INSTRUCTIONS RESIDING IN ALTERNATE BANKS OR MODULES)

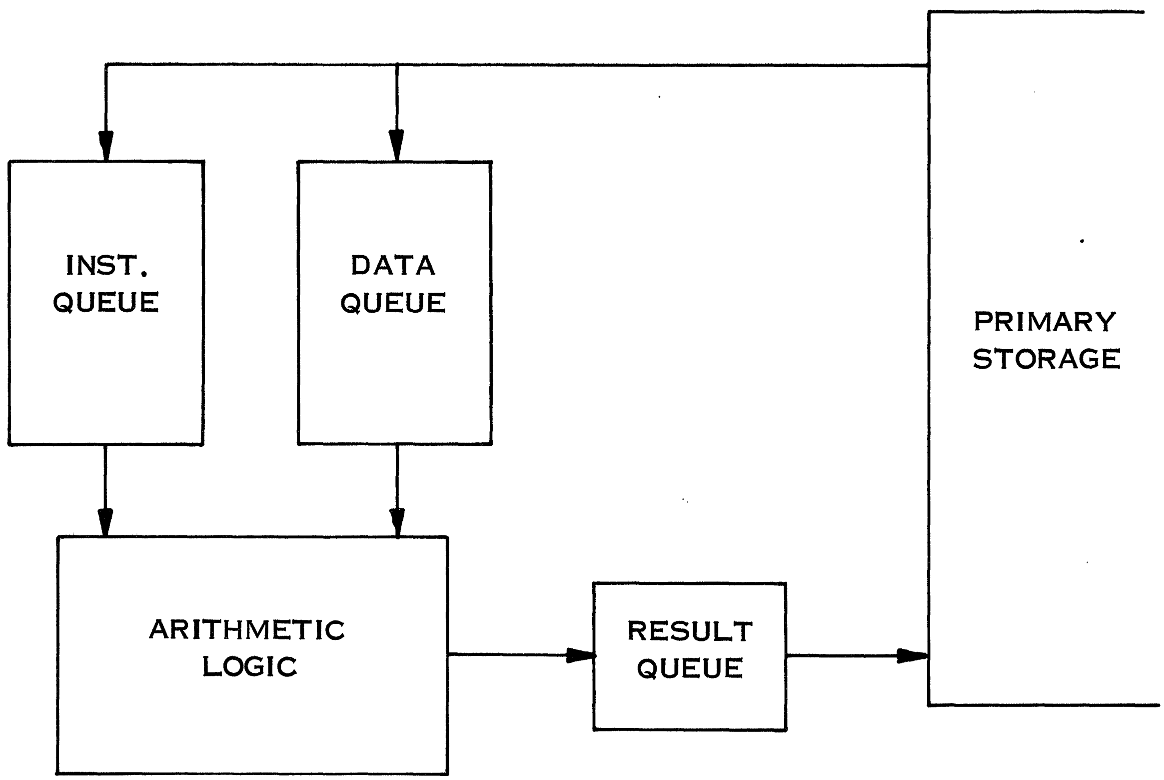
LOOK–AHEAD


MEMORY CYCLE
MEMORY CYCLE
MEMORY CYCLE
I_A
I_B
OAD_B
D_B
X_B
OAD_A
D_A
X_A
I_C
I_D

PIPELINE


MEMORY CYCLE
MEMORY CYCLE
MEMORY CYCLE
MEMORY CYCLE
I $I_A + I_B$
D_A D_B
I_C I_D
X_A X_B
D_C D_D
I_E I_F
X_C X_D . . .
D_E D_F
I_G I_H

FUNCTIONAL OUTLINE PIPELINE MACHINE



OTHER TECHNIQUES TO REDUCE LOGIC—MEMORY SPEED MISMATCH

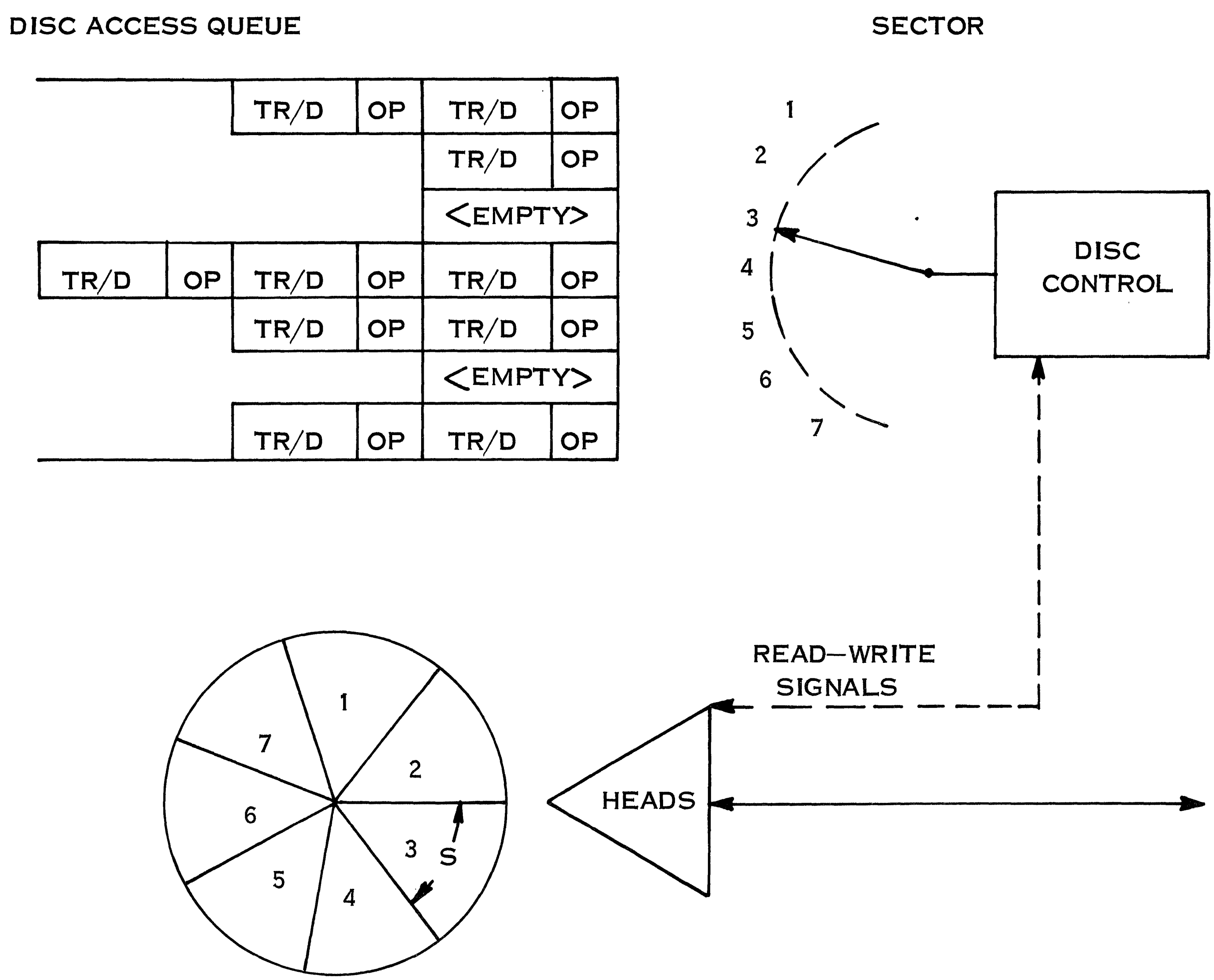
- LOOKASIDE

- SCRATCHPADS

TECHNIQUES FOR REDUCING PRIMARY—SECONDARY STORAGE SPEED MISMATCH

- MULTIPLE CHANNELS
- HEAD PER TRACK DISC UNIT
- SECTOR QUEUES

DISC—SECTOR QUEUEING FUNCTIONAL DIAGRAM

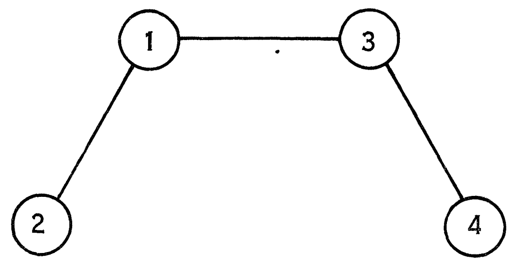


SOURCE OF PARALLELISM IN PROGRAMS

- INDEPENDENT OPERATIONS
  - STATEMENT LEVEL
  - ARITHMETIC EXPRESSION LEVEL
- PARALLEL LOOPS
- OVERLAPPED LOOPS

(1) A = B

(2) C = A + 1



(3) D = B + 2

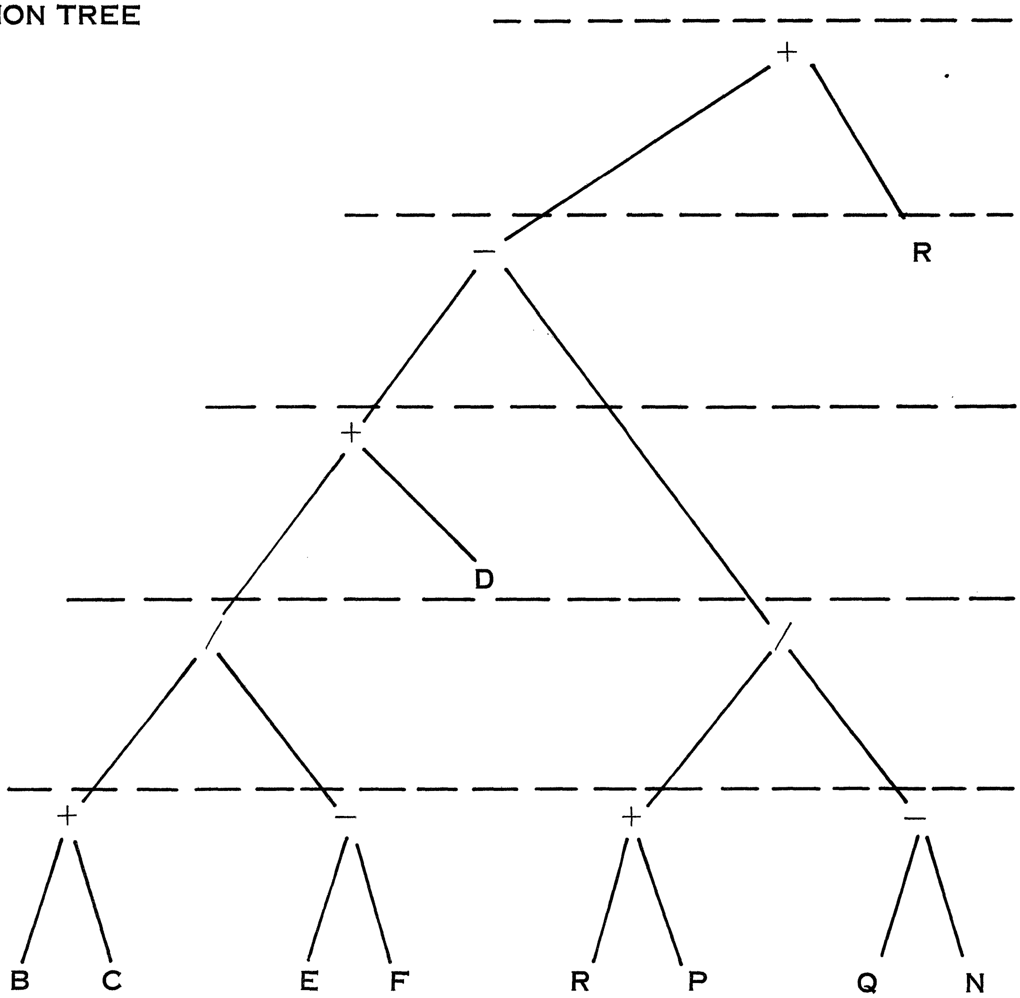
(4) B = B + 1

INDEPENDENT STATEMENTS, 1, 3
2, 4

EXPRESSION PARALLELISM

EXPRESSION: (B + C)/(E − F) + D − (R + P)/Q − M) + R

EXPRESSION TREE



ALL OPERATIONS AT SAME LEVEL ARE INDEPENDENT AND CAN BE EXECUTED IN PARALLEL

PARALLEL LOOP

```
      R = 5
   DO 1Ø I = 1, 1Ø
      M = I + R
    A(I) = B(I) + M
   1Ø CONTINUE
```

ITERATION 1	ITERATION 2	ITERATION 3	. . . .
M = 1 + R	M = 2 + R	M = 3 + R	
A(1) = B(1) + M	A(2) = B(2) + M	A(3) = B(3) + M	

PARALLEL LOOP CHARACTERISTICS

- SAME OPERATION(S) APPLIED TO DIFFERENT DATA
- INDEX SET DETERMINES DATA IN A REGULAR MANNER
- PERMITS BULK EXECUTION OF PROGRAMS

- CAPABILITIES
  - BATCH PROCESSING
  - DEMAND REMOTE
  - REAL–TIME COMMUNICATIONS
- FEATURES
  - PROGRAM PROTECTION
    - MEMORY
    - RESERVED OPERATIONS
  - MASS STORAGE UTILIZATION
  - ELABORATE PROGRAM FILE SYSTEM
  - CONTROL STATEMENTS MAY BE CATALOGUED
  - MULTIPLE VERSIONS
- LANGUAGES
  - FORTRAN
  - COBOL
  - ASSEMBLY
  - ALGOL
  - CONVERSATIONAL FORTRAN

▶ BASIC CONCEPTS AND DEFINITIONS

- ACTIVITY
- BATCH
- COLLECTION
- FILES
  - GRANULES
  - PACKETS
- RUN
- TASK
- SWAPPING
- PRIVILEGED INSTRUCTIONS

▶ SYSTEM COMPONENTS

- SUPERVISOR
- EXECUTIVE REQUESTS
- SYMBIONTS
- I/O HANDLERS
- OPERATOR COMMUNICATIONS
- FILE CONTROL
- DATA HANDLING
- FILE UTILITIES
- AUXILIARY PROCESSORS
  - — COLLECTOR
  - — PROCEDURE DEFINITION
  - — LANGUAGE PROCESSORS
    - PROCESSOR INTERFACE ROUTINES
- DIAGNOSTIC SYSTEM
  - — SNAPSHOTS
  - — POST—MORTEM
- SYSTEM GENERATION
- UTILITY ROUTINES

▶ STATEMENT FORMAT

@ [<LABEL>] : <COMMAND> [<, OPTIONS>] <SPEC. LIST> <COMMENTS>

▶ STATEMENT TYPES

- ORGANIZATIONAL
- I/O SPECS
- PROCESSOR CALLS
- PROGRAM EXECUTION
- CONDITIONAL

ORGANIZATIONAL STATEMENTS

@ RUN	APPEARS AT THE BEGINNING OF EACH RUN. PROVIDES ACCOUNTING AND IDENTIFICATION INFORMATION.
@ FIN	APPEARS AT THE END OF EACH RUN.
@ LOG	PLACES USER SPECIFIED INFORMATION IN THE SYSTEM LOG.
@ MSG	PLACES A MESSAGE ON THE CENTRAL–SITE CONSOLE TYPEWRITER.
@ HDG	USED TO PLACE A HEADING LINE ON PRINT OUTPUT.
@ ADD	USED TO DYNAMICALLY EXPAND THE RUN STREAM.
@ START	USED TO SCHEDULE THE EXECUTION OF AN INDEPENDENT RUN.
@ SYM	USED TO SCHEDULE NON–STANDARD SYMBIONT ACTION.
@ COL	USED TO SPECIFY VARIOUS FORMS OF INPUT.
@ CKPT	USED TO ESTABLISH A CHECKPOINT DUMP THAT MAY BE USED FOR RESTART AT SOME FUTURE TIME.
@ RSTRT	USED TO RESTART A RUN AT SOME PREVIOUSLY TAKEN CHECKPOINT.

INPUT/OUTPUT SPECIFICATION STATEMENTS

@ ASG	USED TO ASSIGN A PARTICULAR INPUT/OUTPUT DEVICE OR MASS STORAGE FILE TO A RUN. THERE ARE FOUR TYPES OF @ ASG STATEMENTS: FASTRAND TAPE DRUM ARBITRARY DEVICE ALSO USED TO CATALOGUE FILES.
@ MODE	USED TO CHANGE THE MODE SETTINGS (DENSITY, PARITY, ETC.) OF A TAPE FILE.
@ CAT	CATALOGUES FASTRAND FORMATTED OR EXISTING TAPE FILES.
@ FREE	USED TO DEASSIGN A FILE AND ITS INPUT/OUTPUT DEVICE OR MASS STORAGE AREA.
@ USE	USED TO SET UP A CORRESPONDENCE BETWEEN INTERNAL AND EXTERNAL FILE NAMES.
@ ELT	INSERTS OR UPDATES A PROGRAM–FILE ELEMENT FROM THE CONTROL STREAM.
@ DATA	USED TO INTRODUCE OR UPDATE A DATA FILE FROM THE CONTROL STREAM.
@ END	USED TO TERMINATE A DATA FILE.
@ FILE	USED TO CAUSE THE DIRECT CREATION OF A FILE CONTAINING DATA TAKEN FROM THE CONTROL STREAM.
@ ENDF	USED TO TERMINATE THE DATA THAT FOLLOWS THE @FILE STATEMENT.
@ QUAL	USED TO DEFINE A STANDARD FILE NAME QUALIFIER.

PROGRAM EXECUTION STATEMENTS

@ MAP	USED TO CALL THE COLLECTOR AND PREPARE AN ABSOLUTE ELEMENT.
@ XQT	USED TO INITIATE THE EXECUTION OF A PROGRAM.
@ EOF	USED TO SEPARATE DATA WITHIN THE CONTROL STREAM.
@ PMD	USED TO TAKE EDITED POST—MORTEM DUMPS OF THE PROGRAM JUST EXECUTED.

PROCESSOR CALL STATEMENTS

@ PROCESSOR	USED TO EXECUTE A PROCESSOR (@COB FOR COBOL COMPILER, @FOR FOR FORTRAN, @ASM FOR ASSEMBLER, ETC.)

CONDITIONAL STATEMENTS

@ LABEL:	USED TO ATTACH A LABEL TO AN EXISTING CONTROL STATEMENT.
@ SETC	PLACES A VALUE IN THE 'CONDITION' WORD.
@ JUMP	USED TO BRANCH CONTROL WITHIN THE CONTROL STREAM.
@ TEST	USED TO TEST THE 'CONDITION' WORD IN THE COURSE OF DECIDING THE EFFECTIVE CONTROL STREAM.

BATCH PROCESSING

- SIMPLE FORTRAN LOAD–AND–GO EXAMPLE:

```
@  RUN      AK4,888,OPTICS,5,75
@  ASG,T   ATMOS,T,A341
@  FOR
   .....
   .....
@  FORTRAN SOURCE
   .....
     XQT
   .....
   .....
   DATA
   .....
   .....
@    PMD
@    FIN
```

- A MORE COMPLEX EXAMPLE:

```
@    RUN     AL5,888,OPTICS,10
@    ASG,T   ATMOS,T,A341
@    ASG     SPEC,F                               SPECIAL FILE
@    FOR     PROGS.MURK(15) , PROGS.MURK/ABER
   ...............
   CORRECTIONS TO CREATE MURK/ABER FROM MURK (15)
   ...............
@    MAP
   IN         PROGS.MURK/ABER
@    XQT
@    SYM      PRNT,SPEC
@    FIN
```

DEMAND PROCESSING EXAMPLE

▶ USER SIGN—ON:

U1108 T/S 1	(TERMINAL IDENTIFIED WITH WRU.)
READY	(THE SYSTEM IS READY FOR FIRST INPUT.)
# RUN XYZ,311202,DEMO	(THE RUN BEGINS WITH RUNID, ACCOUNT, AND PROJECT NUMBER TO IDENTIFY THE USER.)
# ASG,C PF,F/5	(A 5 TRACK FILE 'DEMO PF' IS ASSIGNED, TO BE CATALOGUED AT THE END OF RUN.)
# ASM,I PF.ODDEVEN	(START ASSEMBLY OF ELEMENT CALLED 'ODDEVEN'.)
ASM 1/1/67	(THE ASSEMBLER IS READY TO ACCEPT INPUT.)

DEMAND PROCESSING EXAMPLE

▶ ASSEMBLY LANGUAGE PROGRAM:

```
        REGNAM                            (A PROC TO DEFINE REGISTER NAMES
                                          IS CALLED FROM THE SYSTEM LIBRARY.)

P       FORM      12,6,18                 (AS THE USER TYPES, THE ASSEMBLY
ST *    P$RINT    (P 5,4,STMSG)           IS TAKING PLACE.  THE SYMBIONTS WILL
        R$EAD     (+ EXIT$,INPUT)         QUEUE A LINE IF NECESSARY WHEN THE
                                          USER GETS AHEAD OF THE ASSEMBLER.)

        L         A1, INPUT ?             (FORGOT ',S1'; DELETE IMAGE AND TRY
                                          AGAIN.)

        L,S1      A1, INPUT
        L         A0, (P 1'4'ODD)
        JB        A1,ST+1
        L         A0, (P 1'4E',EVEN)      (WENT BACK TO FIX A MISSING COMMA,
                                          (DOUBLE QUOTE–TTY.))

        J         ST+1
INPUT RES         14
STMSG 'TYPE A SINGLE NUMBER.'
ODD     'IT'S ODD; TRY ANOTHER.'
EVEN   'IT'S EVEN; TRY ANOTHER.'
        END       ST
```

DEMAND PROCESSING EXAMPLE

▶ EXECUTION OF PROGRAM AND SIGN–OFF

ASM COMPLETE $0 000043	(THE ASSEMBLY IS FINISHED. PRO–GRAM IS 043 WORDS LONG.) (REQUEST EXECUTION.)
# XQT,N	
TYPE A SINGLE NUMBER. 1 IT'S ODD; TRY ANOTHER. 4 IT'S EVEN; TRY ANOTHER. A	(NOW THE PROGRAM AND THE USER CONVERSE.)
IT'S EVEN; TRY ANOTHER.	(SMART PROGRAM— .)
# FIN	(THAT'S ENOUGH.)
27/ 3/67 0945	
RUNID: XYZ ACCOUNT: 311202 PROJECT: DEMO TIME: 0000.02 IN: 00023 OUT: 00000 PAGES: 0001	
(EOT)	(END OF TRANSMISSION REQUEST TO QUIT THE LINE.)
LINE RELEASED	(LAST WORDS FROM SYSTEM.)

SUPERVISOR COMPONENTS

▶ RESIDENT ROUTINES

- INTERRUPT SUPERVISOR.
- CPU DISPATCHER.
- INPUT/OUTPUT CONTROL.
- DEVICE HANDLERS FOR TAPE, FASTRAND, COMMUNICATIONS SUB–SYSTEMS, ETC. (RECOVERY SEQUENCES ARE TRANSIENT).
- DRUM HANDLER, INCLUDING RECOVERY SEQUENCES.
- DYNAMIC ALLOCATOR.
- CORE CONTENTS CONTROL.
- EXECUTIVE REQUEST SUPERVISOR.
- REAL–TIME CLOCK AND DAY CLOCK ROUTINES.
- BLOCK BUFFERING PACKAGE.
- TASK AND SEGMENT LOADER.
- CONSOLE CONTROL.
- BASIC QUEUEING PACKAGE AND QUEUE AREA.
- READS AND PRINTS.
- LOGGING CONTROL.
- ERROR INTERRUPT SUPERVISOR.
- CORE PARITY RECOVERY ROUTINE.
- POWER–LOSS CONTROL ROUTINE.

SUPERVISOR COMPONENTS

▶ TRANSIENT ROUTINES

- CONTROL STATEMENT INTERPRETER.
- COARSE SCHEDULER.
- DEMAND CONTROL
- FACILITIES INVENTORY.
- SECONDARY FASTRAND SPACE ASSIGNMENT.
- COMMUNICATIONS INTERFACE ROUTINES.
- CLT DIAL–UP AND AUTOMATIC–ANSWER
- SYMBIONT PROBE ROUTINES.
- MISCELLANEOUS DEVICE HANDLERS (PAPER TAPE, ETC.) .
- SYMBIONTS.
- CONSOLE HANDLER.
- LOGGING AND ACCOUNTING.
- I/O ERROR RECOVERY SEQUENCES FOR TAPE, FASTRAND, ETC.
- TAPE LABEL CHECKING.
- ABSOLUTE DUMP ROUTINE.

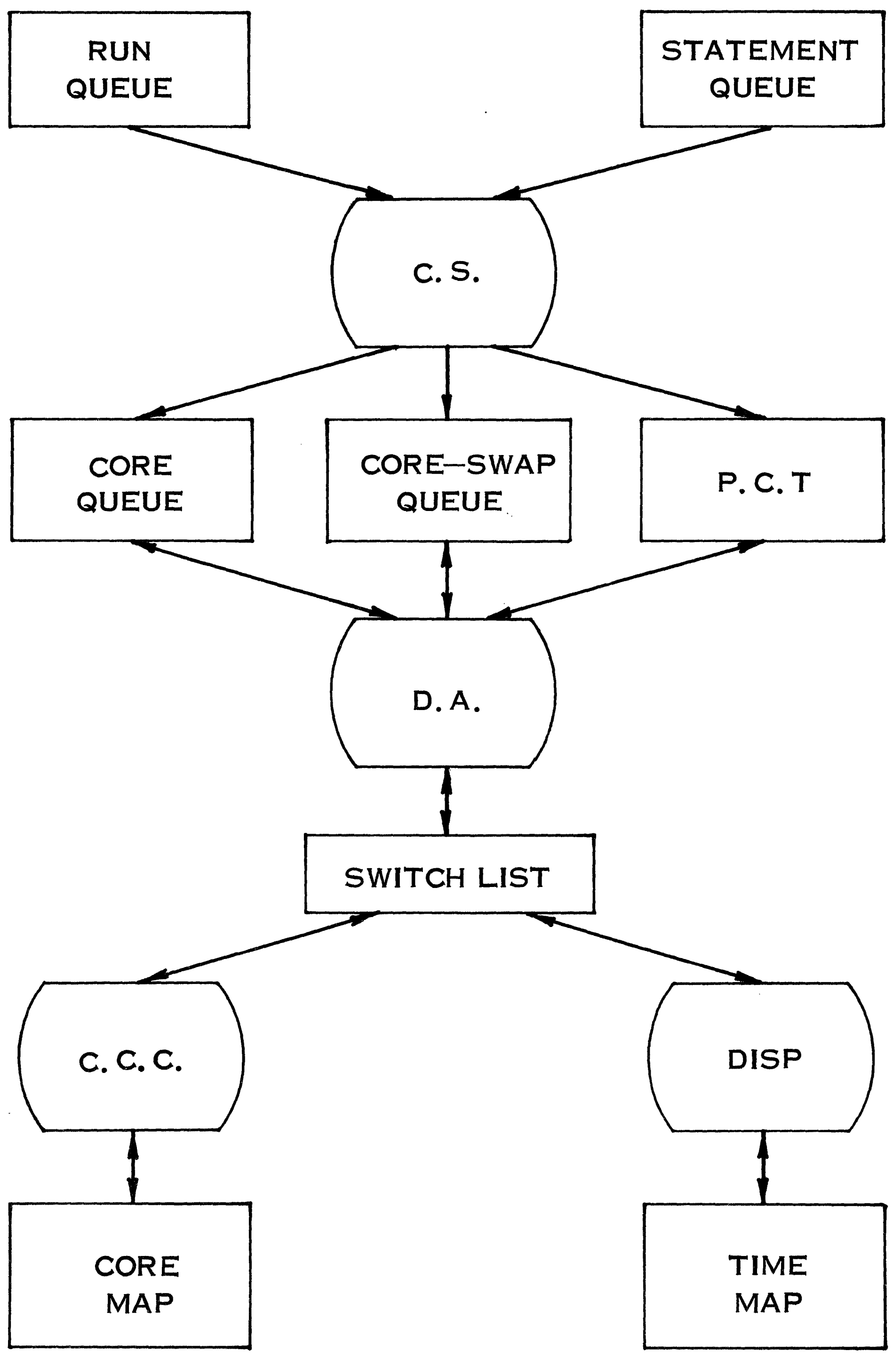
COARSE SCHEDULER

- BATCH PROCESSING
  - RUN QUEUE
  - STATEMENT QUEUE
    - WAIT FOR FACILITIES
    - BEING PROCESSED BY C. S.
    - IN CORE QUEUE
    - WAITING FOR OPERATOR
  - CORE QUEUE
    - ACTIVE
    - SUSPENDED
    - READY
- DEMAND PROCESSING
  - RUN
  - STATEMENT
  - CORE–SWAP QUEUE
    - ACTIVE
    - SWAPPED–OUT
    - READY
    - INPUT–WAIT

DYNAMIC ALLOCATOR

- CORE ALLOCATION
  - USES CORE CONTENT CONTROL (C. C. C)
- TIME ALLOCATION
  - DISPATCHER
  - PRIORITIES
    - REAL–TIME
    - CRITICAL DEADLINE
    - DEMAND
    - BATCH
- PROGRAM STATES
  - TERMINATED
  - SUSPENDED FOR HIGHER PRIORITY
  - WAITING FOR COMPLETION OF EXTERNAL EVENT
  - INPUT–WAIT
  - ACTIVE

DATA FLOW IN THE SUPERVISOR



THE SWITCH LIST

- N–LEVEL, MULTIPLE ENTRY (L = 0, 1, 2,. . . , N)
- INITIAL LEVEL = 0
- LEVEL L HAS PRIORITY OVER LEVEL L+1
- WITHIN LEVEL, CDU TIME PRIORITIES ARE EQUAL
- PROGRAM LOSES CONTROL BY VOLUNTARY OR INVOLUNTARY ACTION
  - THE TIME–LIMIT QUANTUM Q:
    - $T_L = 2^L$
    - A = ALLOCATION FACTOR BY D. A.
    - F = PRIORITY FACTOR
    - $Q = A * (1 + P/F) * T_L$
  - IF Q IS EXCEEDED, $L + 1 \rightarrow L$ FOR THAT TASK
- SWITCH LIST FUNCTIONS FOR DA:
  - ENTER (INITIAL L FOR A TASK)
  - SET (ALTERS VALUE OF A FOR A TASK)
  - MOVE (ALTERS VALUE OF L FOR A TASK)
  - MOVE 1 (ALTERS VALUE OF L FOR ALL TASKS OF GIVEN TYPE)
  - MOVE 2 (INCREMENTS OR DECREMENTS L FOR ALL TASKS OF GIVEN TYPE)

DISPATCHER

- CPU GIVEN TO HIGHEST PRIORITY
- FULL LEVEL–CYCLE MUST BE COMPLETED
- DISPATCHER USES SWITCH LIST FOR:
  - ENTRY POINT
  - RUN ID.
  - STATEWORD
  - ACTIVITY MARK
  - MEMORY LOCKOUTS
  - RUNNING TIME
  - P. C. T. ADDRESS POINTER

FILE CONTROL SYSTEM

▶ FUNCTIONS

- DIRECTORY MAINTENANCE
- MASS STORAGE ALLOCATION
- INTERFACE WORKER PROGRAMS AND DEVICE HANDLERS
- PROTECTION

COLLECTOR EXAMPLE

FILEA ELEMENTS NAME/VERSION	REFERENCES OUTSIDE OF FILEA REQUIRED FILE, NAME/VERSION
MAIN	FILEA, A1, B1, F1
A1/A	
A2/A	LIB1, SIN/X
A3/A	LIB2, COS/X
B1/B	LIB1, SQRT/X
B2/B	
B3/B	
C1/C	LIB1, SQRT/X
C2/C	
D1/D	LIB2, CAT/Y
D2/D	
E1/E	LIB2, CAT/Y
E2/E	
F1	
F2	
G1/G	LIB1, SIN/X
G2/G	LIB2, COS/X
G3/G	

A PARTICULAR COLLECTION SETUP FOR SEGMENTING A PROGRAM FROM THIS FILE MIGHT BE AS FOLLOWS:

```
MAP, L      ,X
SEG         MAIN
IN          FILEA, MAIN
SEG         A*, (MAIN)
IN          FILEA, A1/A, A2/A, A3/A
SEG         B*, (A)
IN          FILEA, B1/B, B2/B, B3/B
SEG         C*, B
IN          FILEA, C1/C, C2/C
SEG         D*, (B, C)
IN          FILEA, D1/D, D2/D
SEG         E*, D
IN          FILEA, E1/E, E2/E
DSEG        F*, (D, G)
IN          FILEA, F1, F2
SEG         G*, (MAIN)
IN          FILEA, G1/G, G2/G
LIB         LIB1, LIB2
@ XQT
```

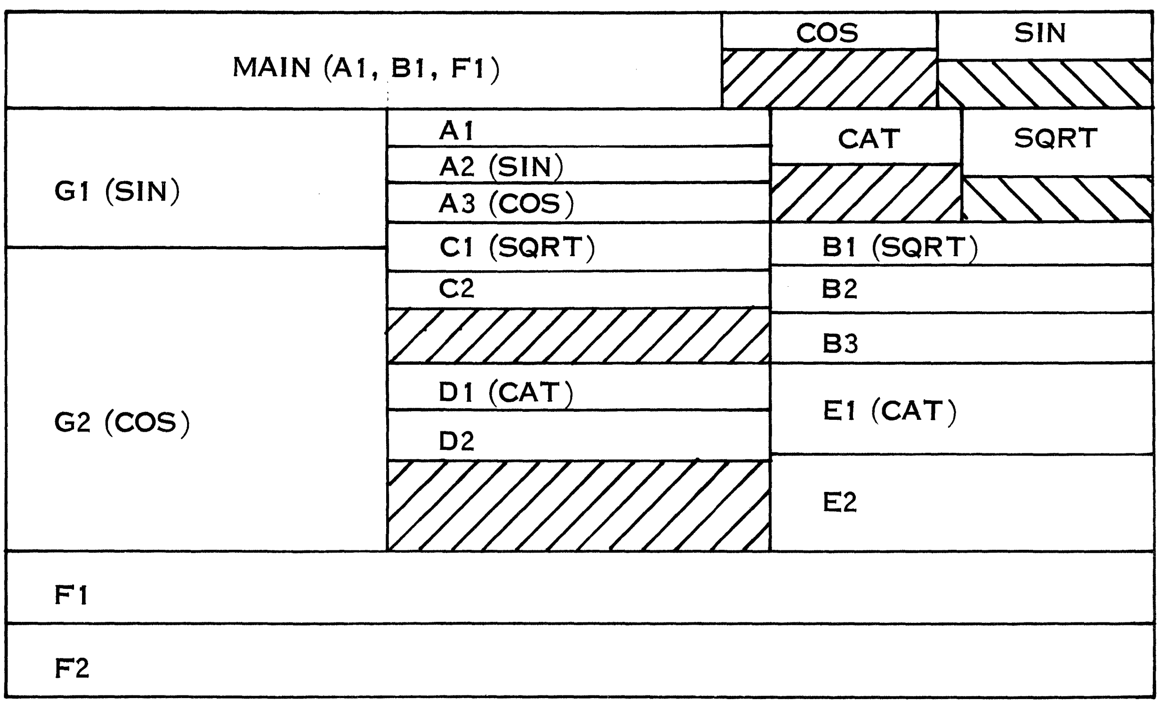
STORAGE MAP

INSTRUCTION AREA MEMORY MAP

01000			K	M	
	CAT	-B1-B2-B3---			
	SQRT		-D1-D2------		
COS	-A1- A2-A3				
SIN			-E1-E2------		
-MAIN------		C1 C2		-F1-- F2------	
	- G1--- G2-	----------------	--------------	--------------	

DATA AREA MEMORY MAP

N				O P
	CAT	- B1- B2- B3------		
ILDS$	SQRT		-D1-D2---	
COS	- A1- A2 -A3 --		-E1-E2---	
SIN		--- C1 -C2--------		
LT- BC-- MAIN-				- F1- F2--------------
	-G1-- G2 ----			



CONVERSATIONAL FORTRAN

- SERVICE LANGUAGE
  - PROGRAM ENVIRONMENT STATEMENTS
  - EXECUTION CONTROL
  - STATEMENT MODIFICATION
  - DISPLAY
  - TEST FUNCTIONS

— TRACE	(REPORT VALUE CHANGES)
— TRAP	(REPORT ALL TRANSFERS)
— TRAIL	(REPORT ALL EXTERNAL PROCEDURE CALLS)
— DUMP	
— LIMIT	(REPORT VALUE OUTSIDE LIMITS)
— KEYIN	(ALLOW CONSOLE CONTROL)
— EX	(IMMEDIATE, BUT NOT PERMANENT)
— EXR	(IMMEDIATE AND PERMANENT)
— OFF	

CONVERSATIONAL FORTRAN

```
@CFOR
+NOTE       CONVERSATIONAL        FORTRAN IN EFFECT
 101.       READY               @EX
            READY                         Z = SQRT (CONSTANT)
                                          Z = VALUE
            READY                         Y = SIN (CONSTANT)
                                          Y = VALUE
            READY                         R = SIN (CONSTANT)
                                          R = VALUE
            READY               @OFF  (EX)
 101.       READY
            .
            .
            .
@CFOR
+NOTE       CONVERSATIONAL        FORTRAN IN EFFECT
 101.       READY               @ACTIVITY TEST
 101.       READY                           READ (2, 20) , A, B, C
 102.       READY                       10  A = B + C
 103.       READY               @UPDATE
*           READY               -101, 101
* 101.      READY                           READ (2, 20) , B, C
* 101. 1    READY               @OFF (UPDATE)
 103.       READY               @TRACE A
 103.       READY                           R = B/A + C
 104.       READY
            .
            .
            .
@CFOR
+NOTE       CONVERSATIONAL        FORTRAN IN EFFECT
 101.       READY               @ACTIVITY EXAMPLE
 101.       READY               @TRACE A, B, C
 101.       READY                           READ (2, 20) , A, B, C
 102.       READY                           D = A - B + C
 103.       READY                           A = D+C/A
 104.       READY                             B = A-D
 105.       READY                       20    FORMAT (F8.3)
 106.       READY               @BEGIN
-101.       READY (INPUT VALUES ENTERED FOR A, B AND C)
+TRC                              103.        A = VALUE
+TRC                              104.        B = VALUE
 106.       READY
```

CONVERSATIONAL FORTRAN

```
@ CFOR
+NOTE      CONVERSATIONAL        FORTRAN IN EFFECT
 101.      READY                             DIMENSION A(100)
 102.      READY                @ EXR
 102.      READY                     20      FORMAT (F8.3)
 103.      READY                             READ (2, 20) , B, C
—103.      READY (INPUT VALUES ENTERED FOR B AND C)
 104.      READY                             D = 20
 105.      READY                             E = 20—B—C
                                             E = VALUE
 106.      READY                             E = 20—C—B
                                             E = VALUE
 107.      READY                 EX          E = B/C+19.9
                                             E = VALUE
 107.      READY                @ UPDATE
*          READY                —102
* 102.1    READY                             READ (2, 50) , (A (I;
                                                 ) , I = 1, 100)
* 102.2    READY                @ OFF  (UPDATE)
 107.      READY                              A(2) =B—C
 108.      READY                              A(3) =—A (2)
 109.      READY
```

CONVERSATIONAL FORTRAN

```
@ CFOR
+NOTE        CONVERSATIONAL         FORTRAN IN EFFECT
 101.        READY               @ EXR
 101.        READY                      10    FORMAT (13)
 102.        READY                       5    READ (2, 10) , J, K, L
-102.        READY  (INPUT VALUES ENTERED FOR J, K AND L)
 103.        READY                       4    IF (J-K) 5, 7, 9
 104.        READY                            IF (L) 4, 5, 9
+ERR                STATEMENT AT 104. REQUIRES A LABEL
 104.        READY                       7    IF (L) 4, 5, 9
 105.        READY                       9    L = L - 1
                                              L = VALUE
 106.        READY               @ OFF (EXR)
 106.        READY               @ UPDATE
*            READY                -103, 103
*103.        READY                       4    IF (K-J) 5, 7, 9
*103.1       READY               @ LIST
 101.                                   10    FORMAT (13)
 102.                                    5    READ (2, 10) , J, K, L
 103.                                    4    IF (K-J) 5, 7, 9
 104.                                    7    IF (L) 4, 5, 9
 105.                                    9    L = L- 1
```

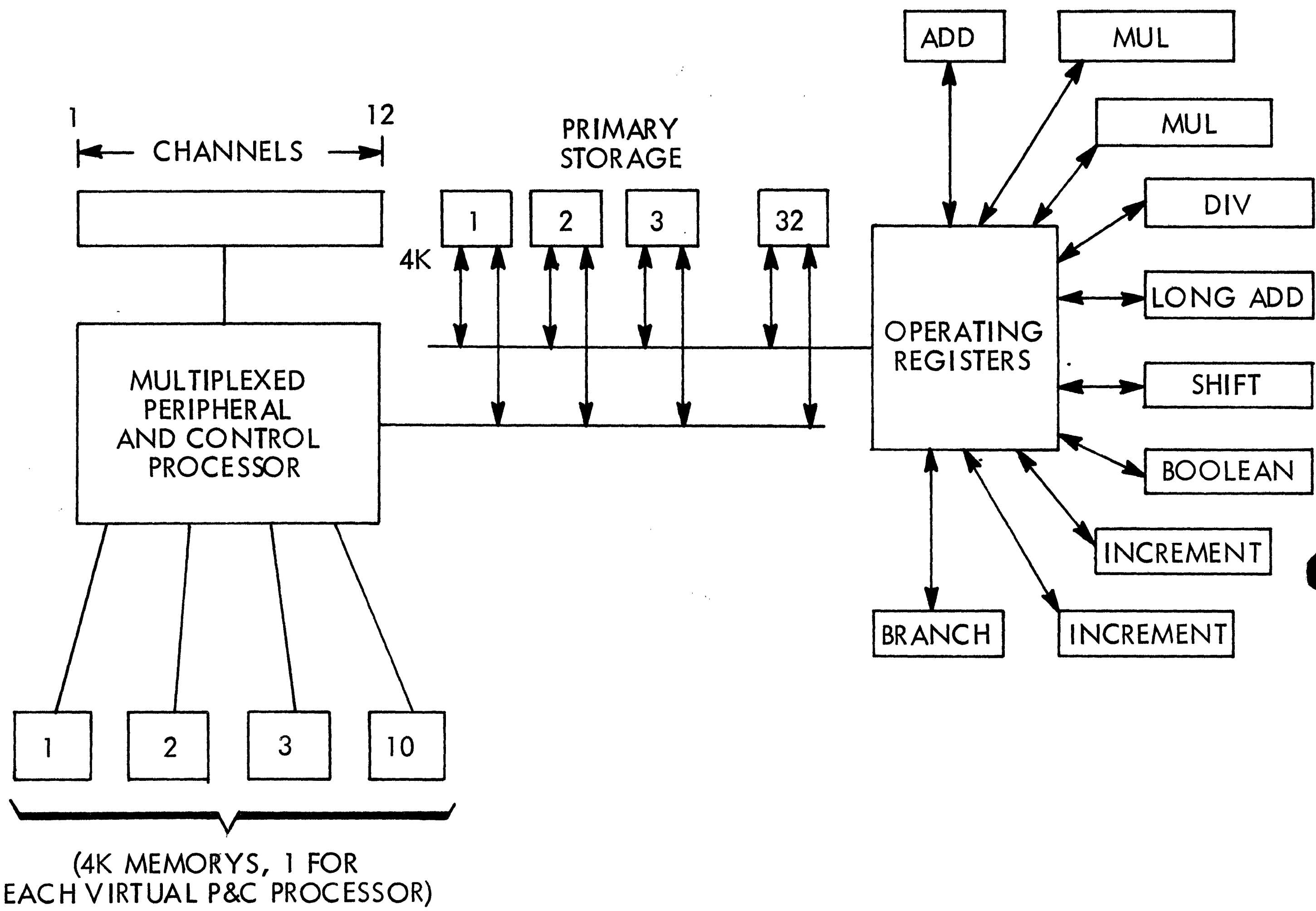
CONVERSATIONAL FORTRAN

```
@ CFOR
+ NOTE     CONVERSATIONAL        FORTRAN IN EFFECT
  101.     READY         @ LIMIT A.GT.20
  101.     READY         @ EXR
  102.     READY                   20 FORMAT (F8.3)
  103.     READY                           READ (2,20), A,I
 -103.     READY  (INPUT VALUES ENTERED FOR A AND I)
  104.     READY         @ OFF (EXR)
  104.     READY                           READ (2,20). (B(J). ; J=1,I)
  105.     READY                   3 IF  (B(I)) 5,15,4
  106.     READY                   4 A = B(I) + SIN(B (I))
  107.     READY                   5 A = B(I) * A
  108.     READY                     I   I - 1
  109.     READY                     GO TO 3
  110.     READY                  15 A = COS (B(I))
  111.     READY         @ BEGIN 104
  101.     READY                   DIMENSION B (100)
 -104.     READY (INPUT VALUES ENTERED FOR B-ARRAY)
 +LMT            A.GT.20   110.      A = 21.75
           READY
           •
           •
```

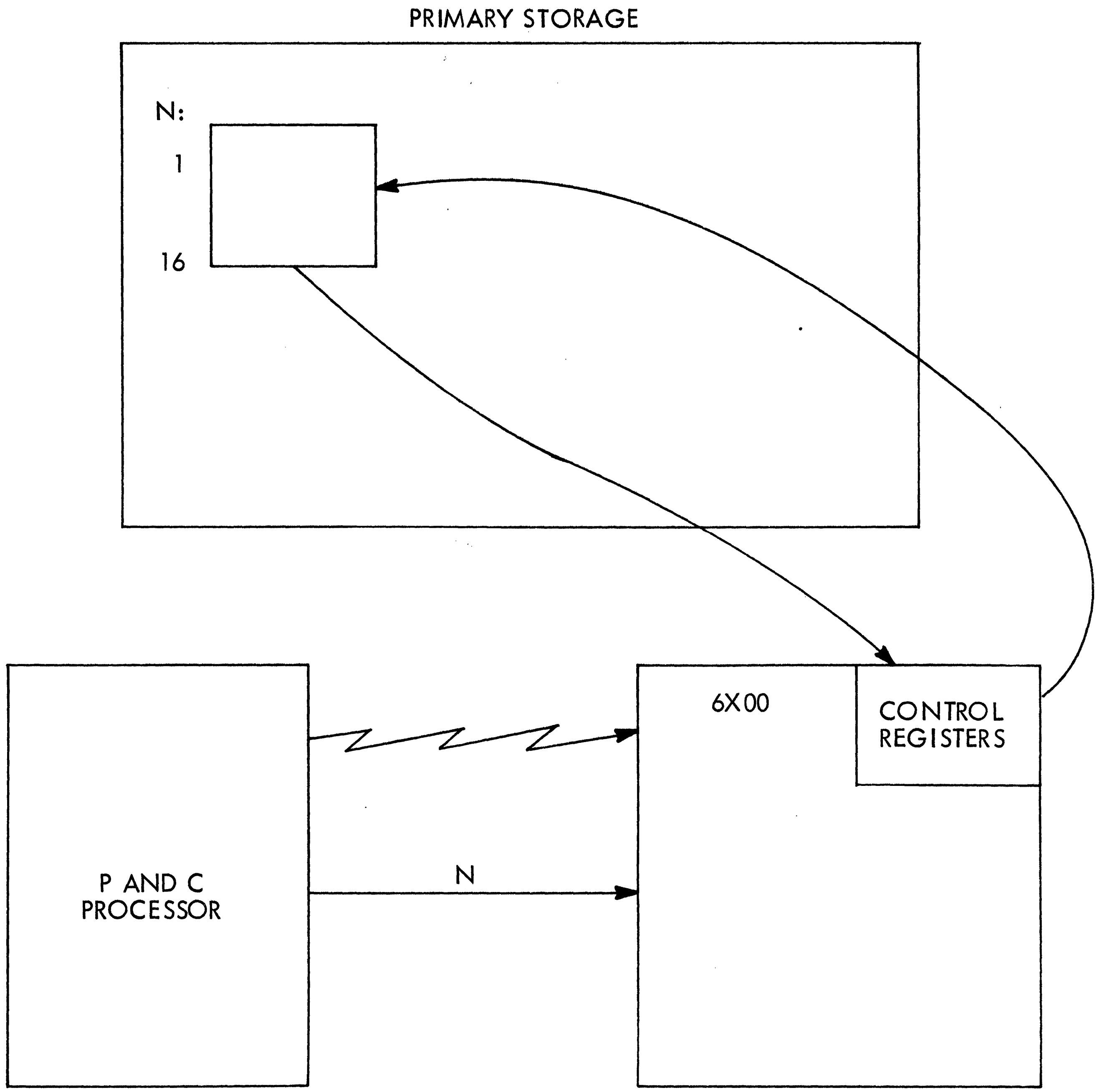
TOPICS COVERED THIS SESSION

- HIGH PERFORMANCE MACHINES
  - 6600
  - 360/9X, 360/85
  - B8500
  - ILLIAC IV

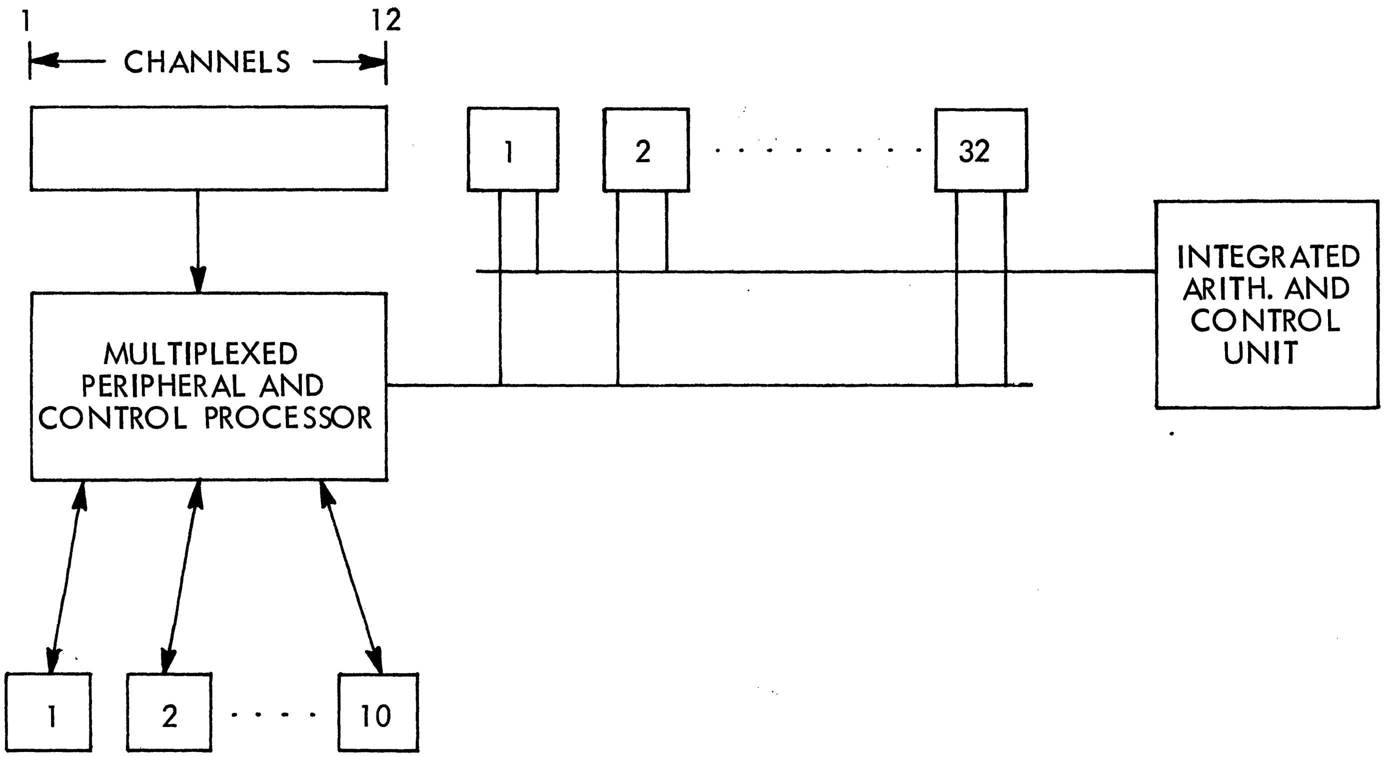
FUNCTIONAL ORGANIZATION-CDC 6600



PROGRAM INITIATION - 6600

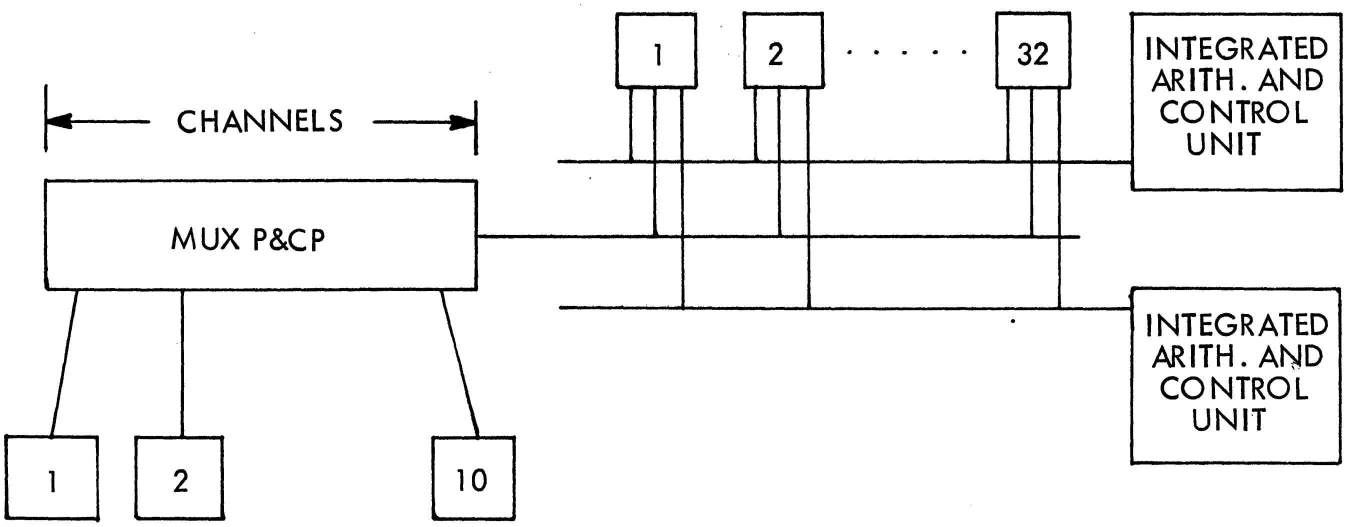


CDC-6400

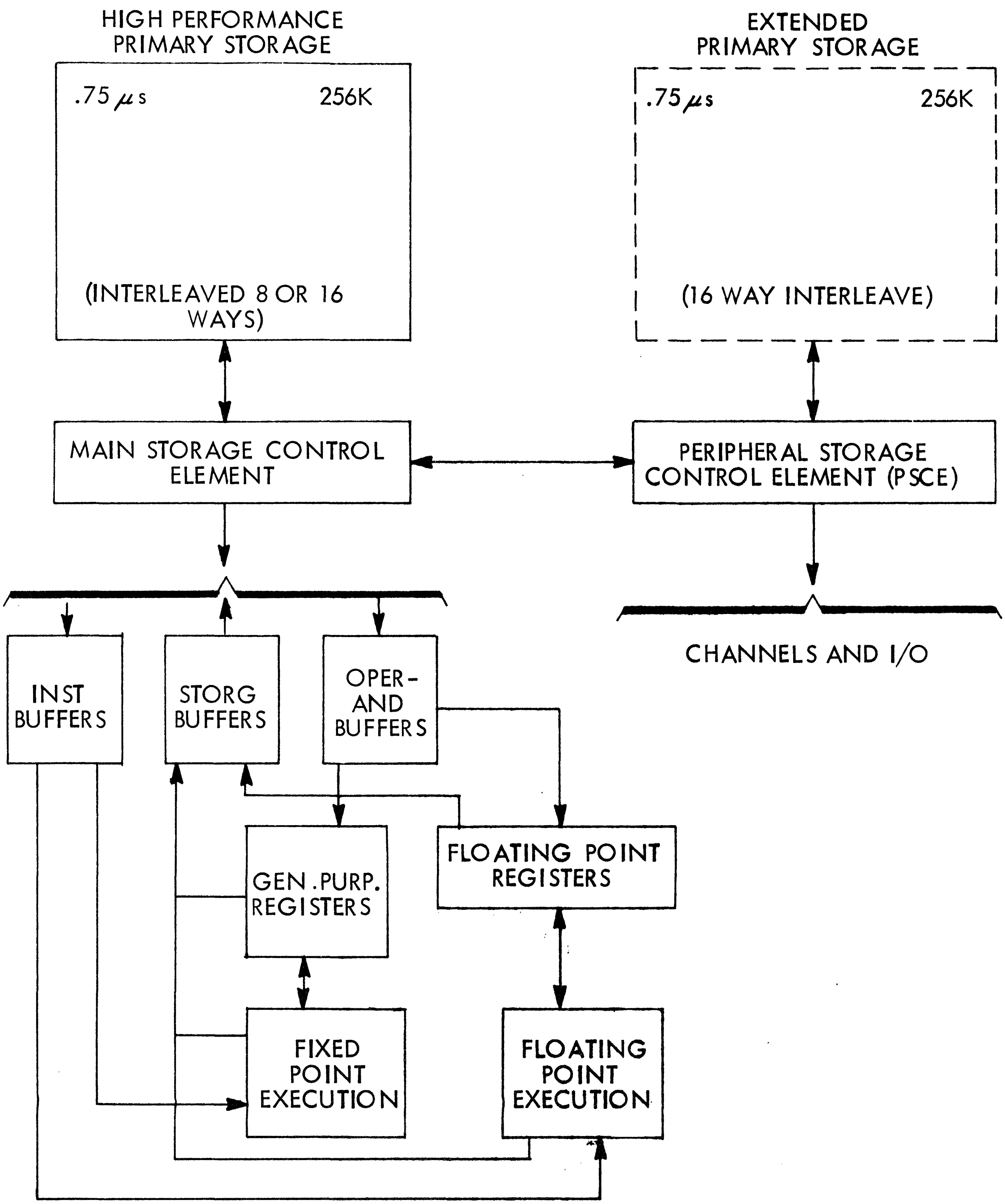


4K MEMORY, 1 FOR EACH VIRTUAL P&C PROCESSOR

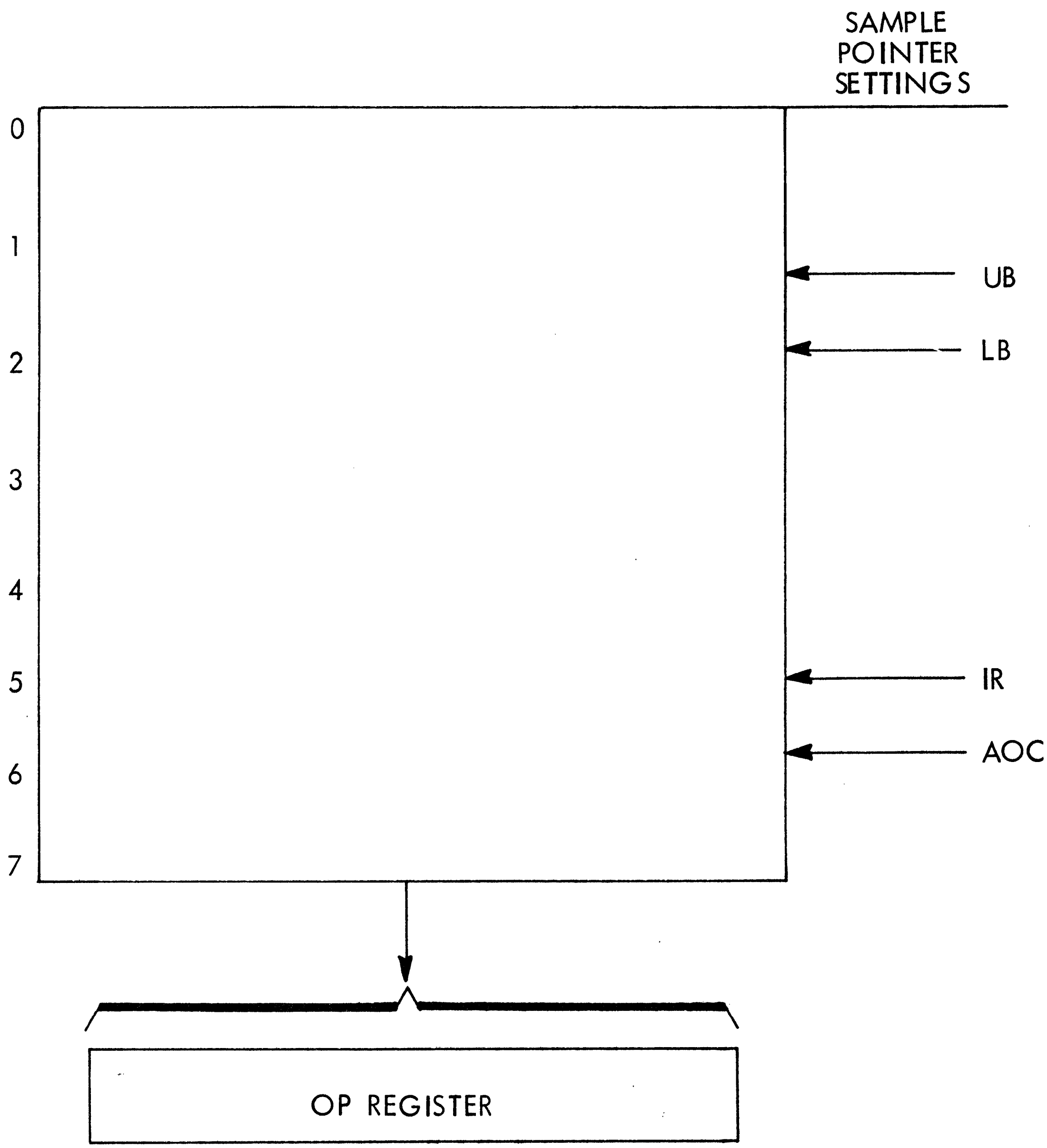
CDC-6500


1
2
32
INTEGRATED ARITH. AND CONTROL UNIT
CHANNELS
MUX P&CP
INTEGRATED ARITH. AND CONTROL UNIT
1
2
10

360/91 FUNCTIONAL DIAGRAM



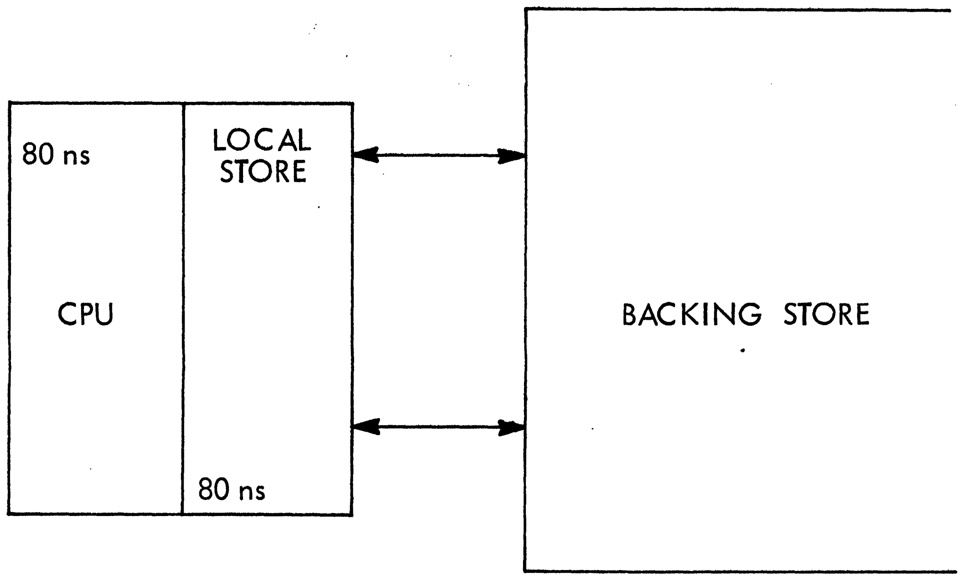
MODULO 8 INSTRUCTION STACK - 360/91



ELEMENTS OF 360/91 CONTRIBUTING TO SPEED

- MULTIPLE INTERLEAVED HIGH SPEED STORAGE
- STORAGE ACCESS BUFFERING
- INSTRUCTION BUFFERING
  - INST FETCH LOOKAHEAD
  - SHORT LOOP EXECUTION
- OPERAND FETCH AND STORE BUFFERING
- MULTIPLE ARITHMETIC EXECUTION ELEMENTS

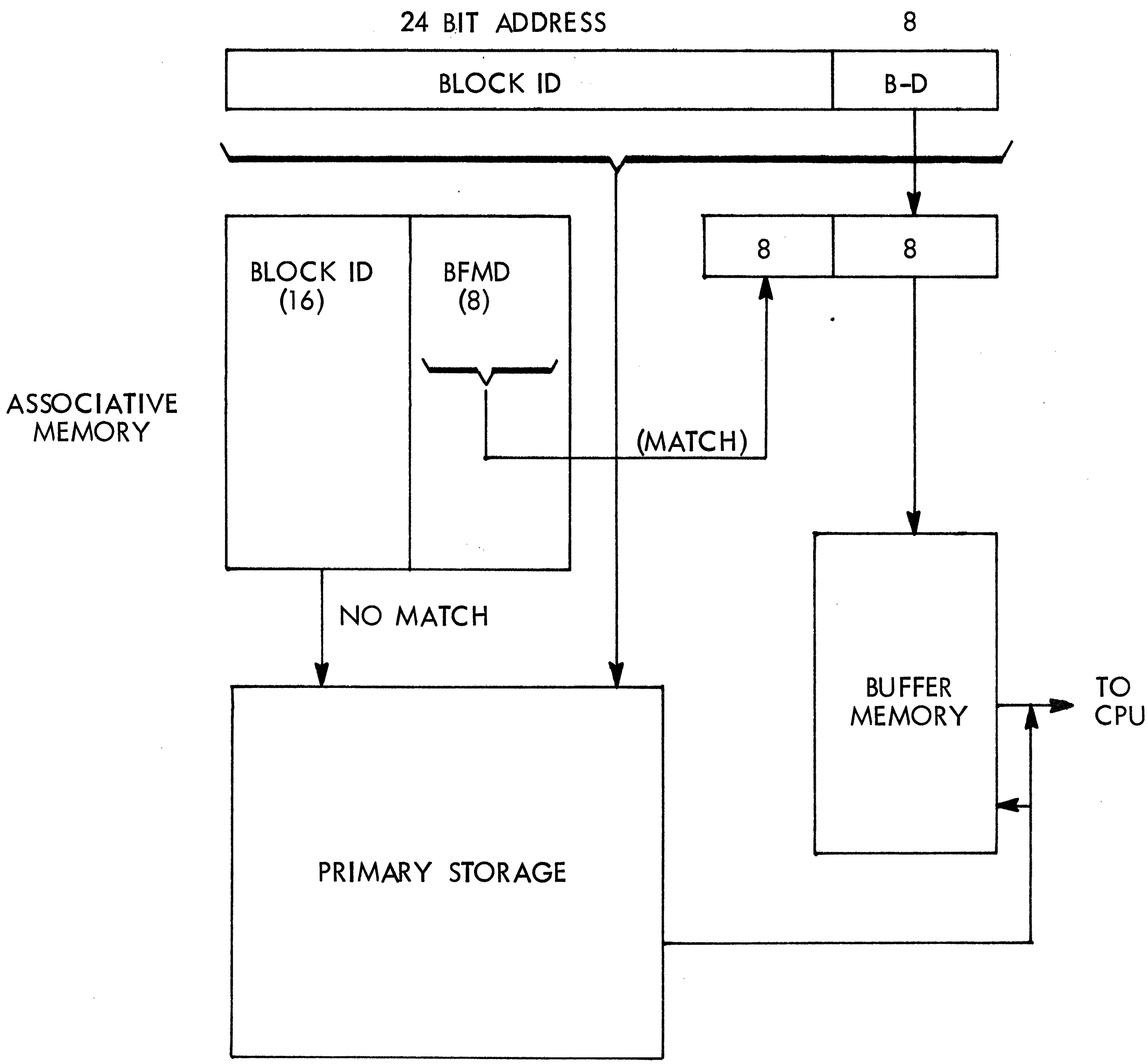
360/85 TWO LEVEL STORAGE SYSTEM



OBJECTIVES OF 360/85 2-LEVEL STORE SYSTEM

- GENERALIZATION OF LOOK-ASIDE MEMORY
- 'PAGE' CONCEPT APPLIED FOR INCREASED PERFORMANCE
- AMORTIZE ACTUAL ACCESS TIME OVER SEVERAL WORDS

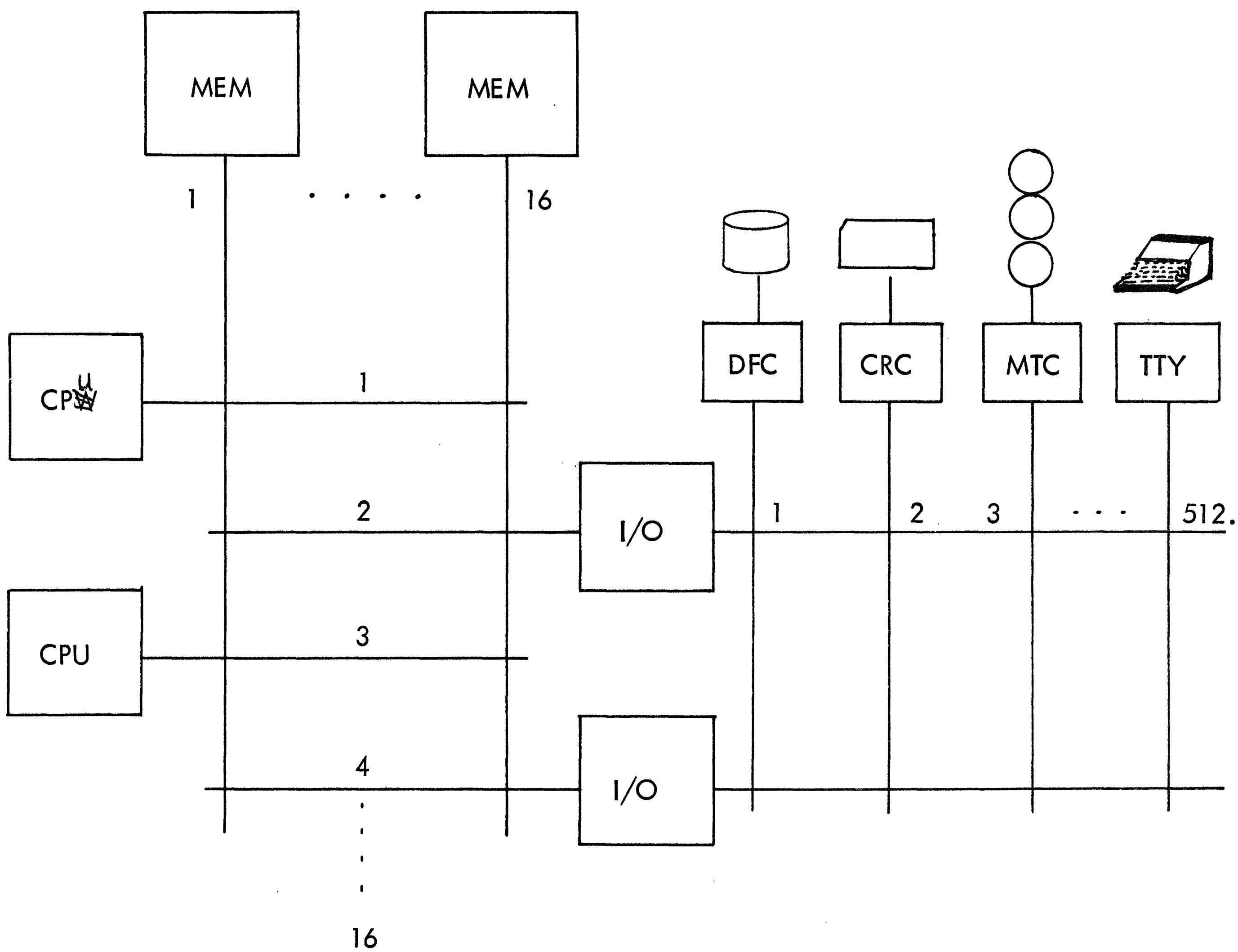
360/85 BUFFER MEMORY LOGIC



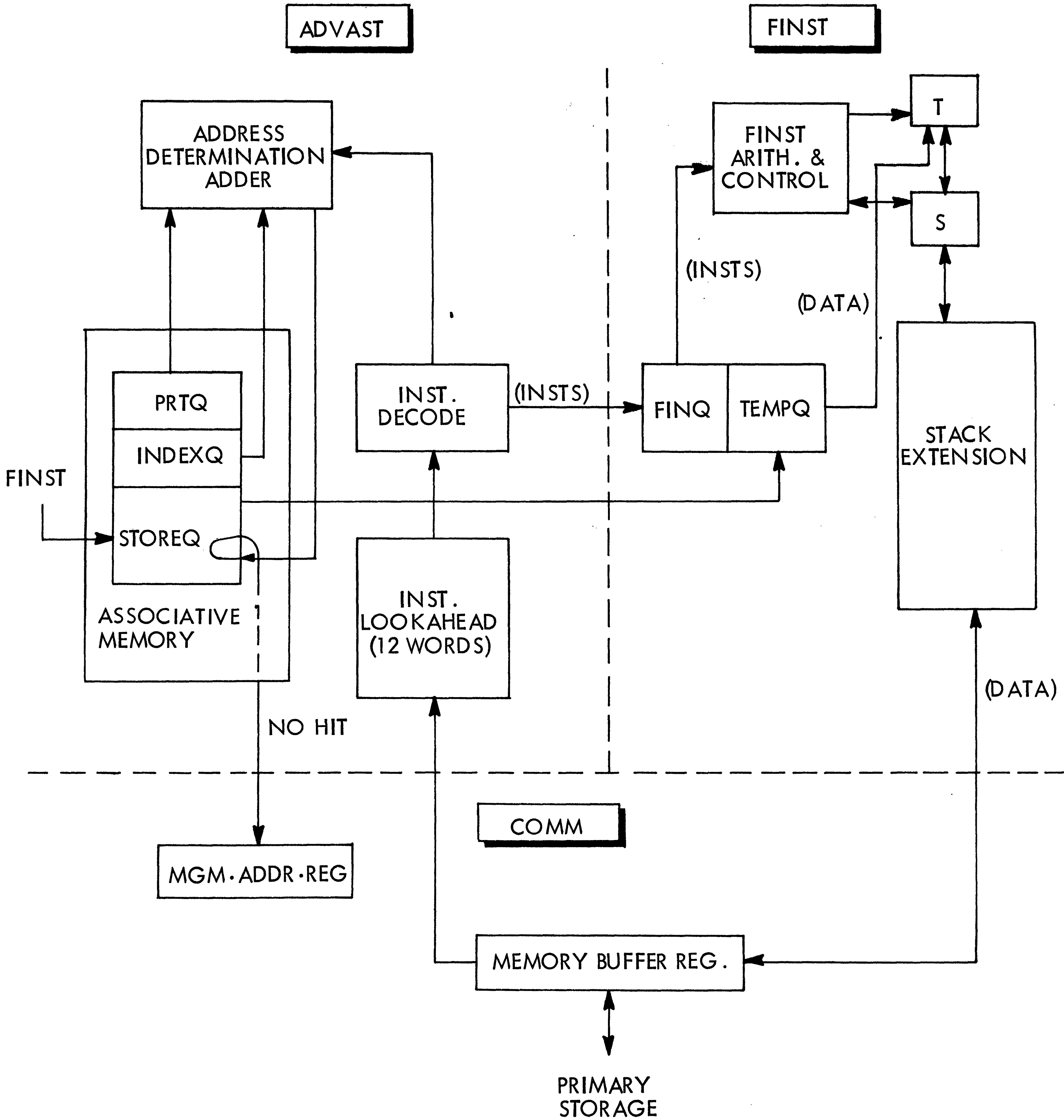
SUMMARY OF MODEL 85 CHARACTERISTICS

- MODEL 85 EMBODIES 'LOOK ASIDE' CONCEPT
- IMPLEMENTATION SIMILAR TO PAGING IN 360/67 (TO BE DISCUSSED)
- WITH THE PARAMETERS CHOSEN, DATA OR INSTRUCTIONS FOUND IN BUFFER MEMORY BETTER THAN 95% OF THE TIME
- SIMULATION STUDIES SHOWED THAT STORAGE FOR ~128 BLOCKS WAS SUFFICIENT TO LOWER REFERENCES OUTSIDE OF BUFFER STORE TO LESS THAN 5% REGARDLESS OF THE PROGRAM SIZE
- THE ADDRESSING PATTERN OF THE PROGRAM IS THE ONLY SIGNIFICANT CHARACTERISTIC AFFECTING THE EFFICIENCY OF THE BUFFER

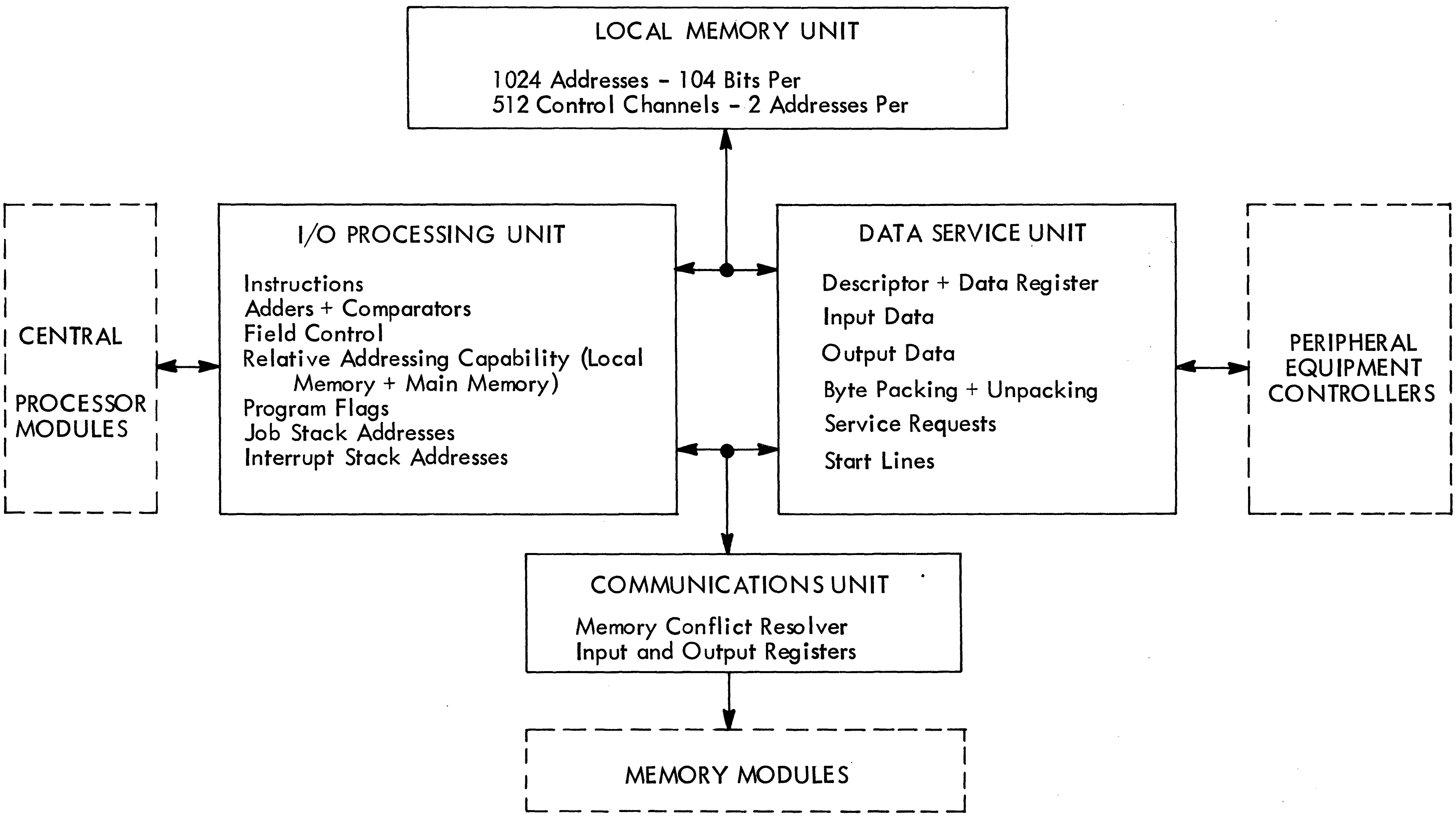
SYSTEM ORGANIZATION - B8500



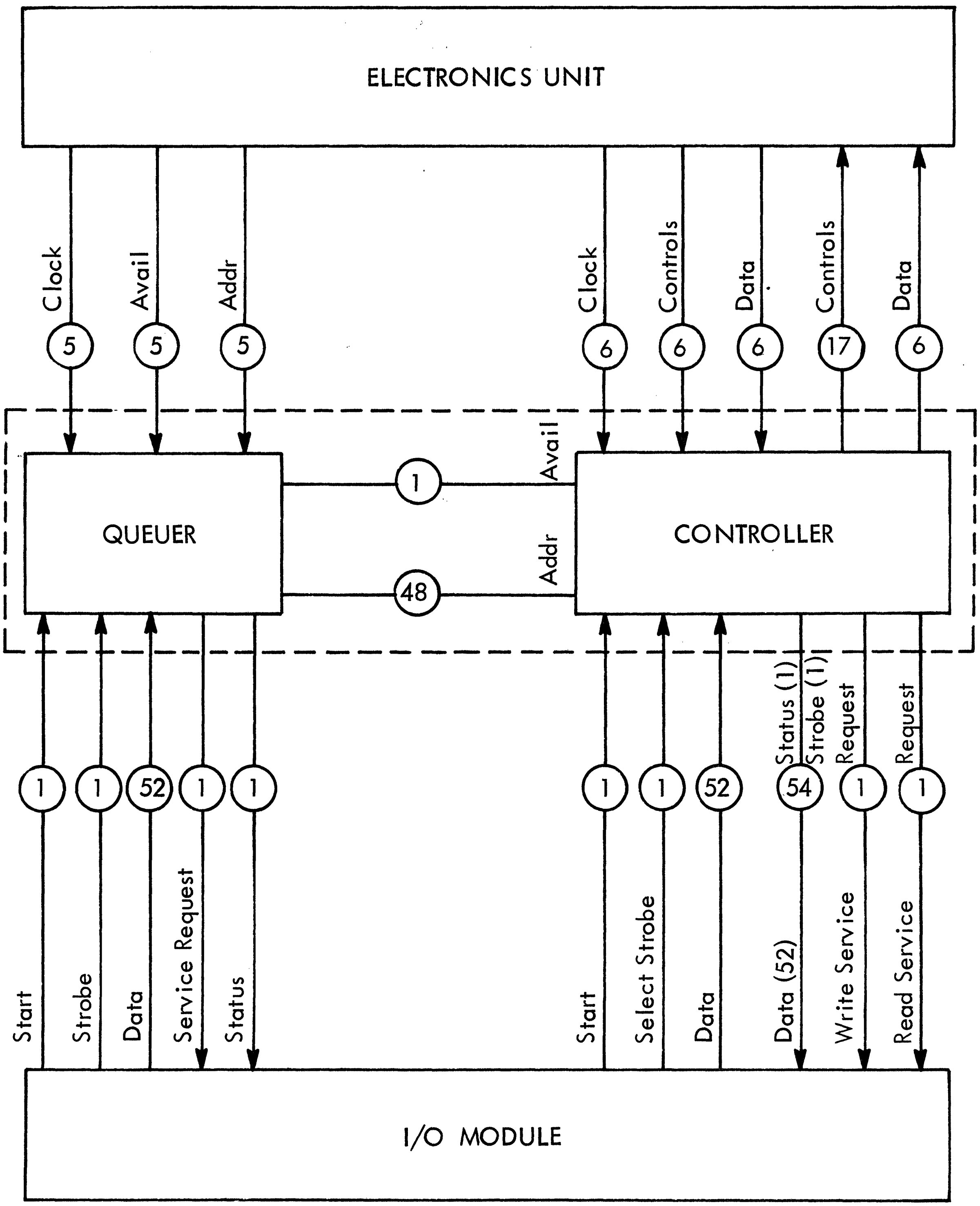
B8500 CPU - SIMPLIFIED FUNCTIONAL DESCRIPTION



B8500 INPUT/OUTPUT MODULE BLOCK DIAGRAM

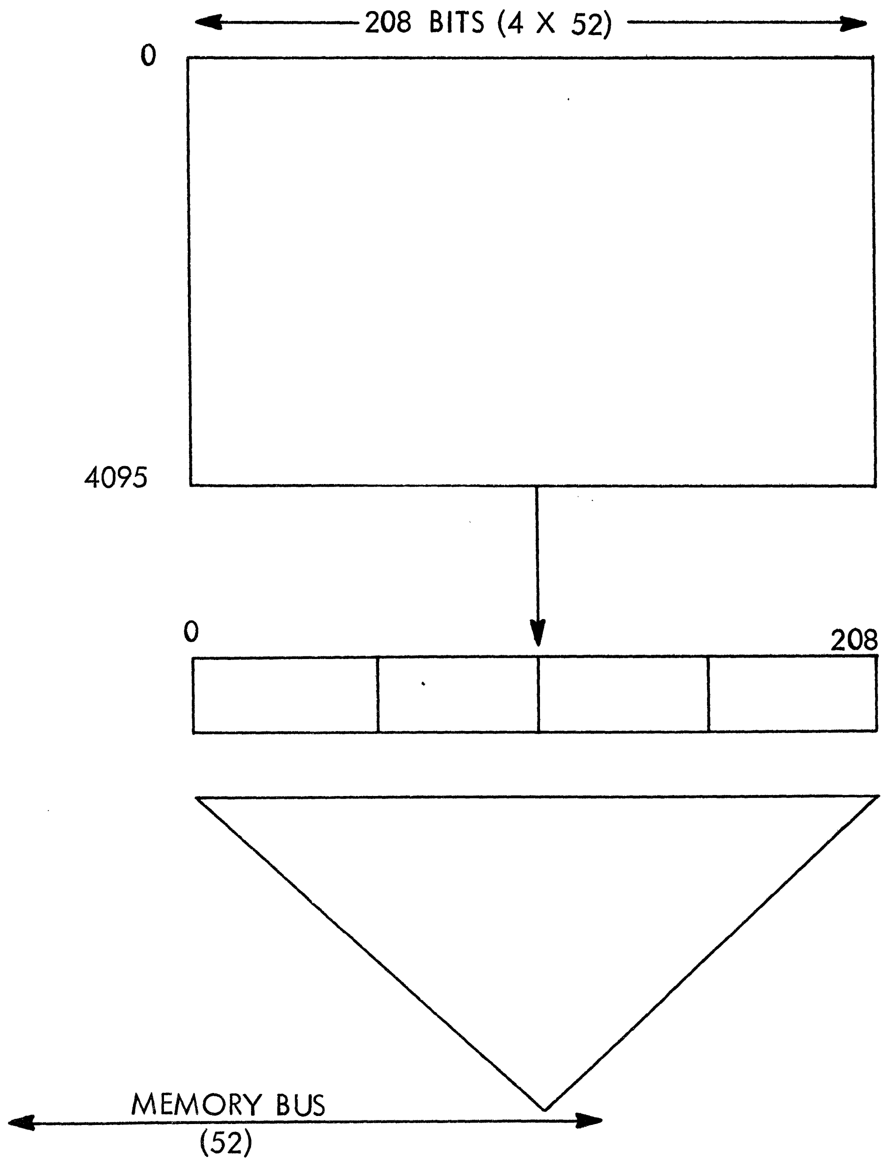


B8500 - DISC FILE CONTROLLER DETAILED INTERFACE



FUNCTIONAL CHARACTERISTICS - B8500 MEMORY MODULE

- 500 NS CYCLE, 200 NS READ ACCESS, 300 NS REGENERATE
- FETCH/STORE 1 OR 4 WORDS



ELEMENTS OF B8500 CONTRIBUTING TO PERFORMANCE

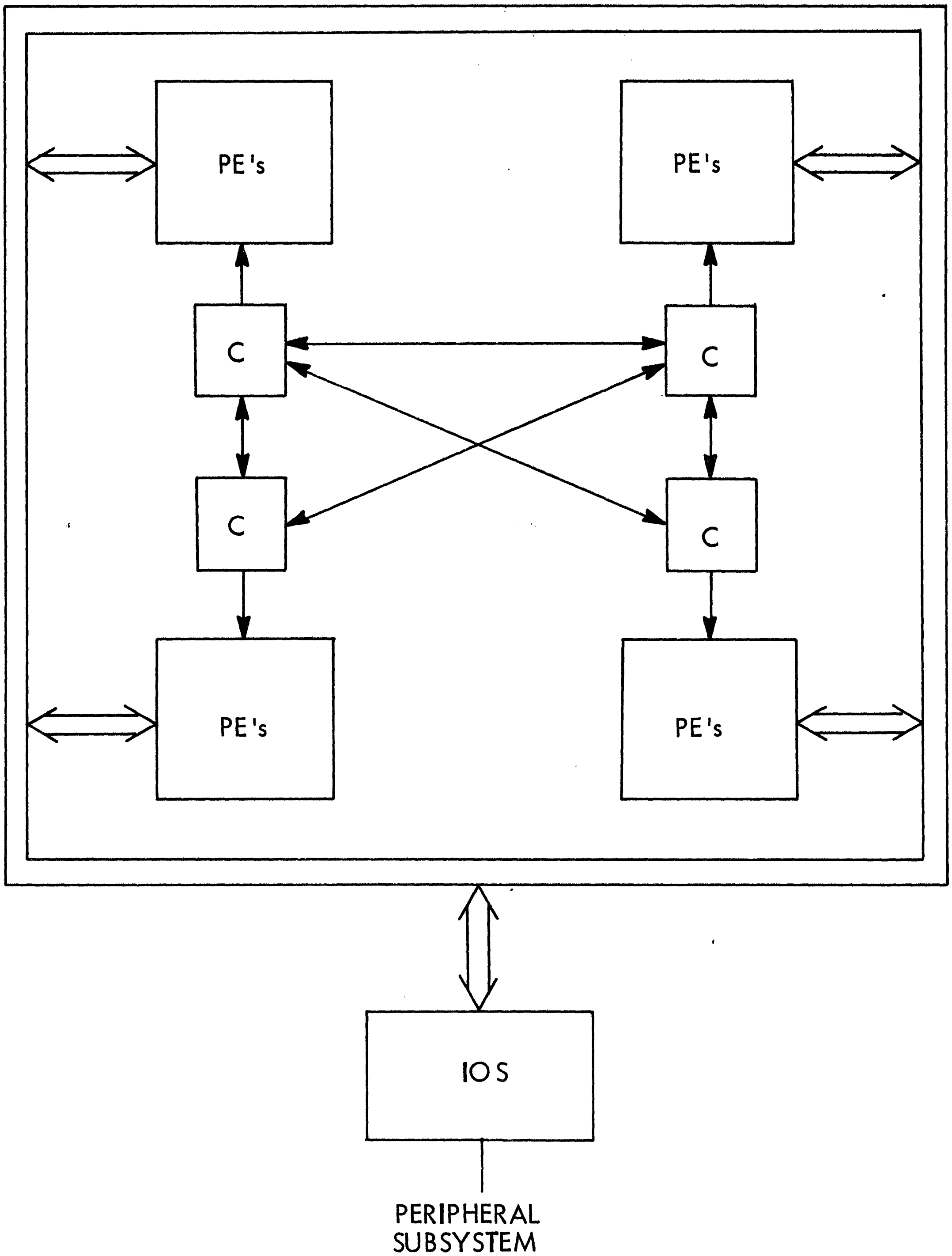
- MULTIPLE INDEPENDENT MEMORY MODULES
- MULTIPLE PROCESSORS/ CHANNELS
- QUEUED ACCESS DISC CONTROLLER
- FUNCTIONAL SEPARATION OF INSTRUCTION PREPARATION, EXECUTION LOGIC, AND COMMUNICATION WITH MEMORY
- INCORPORATION OF ASSOCIATIVE MEMORY FOR

    INDEX VALUES

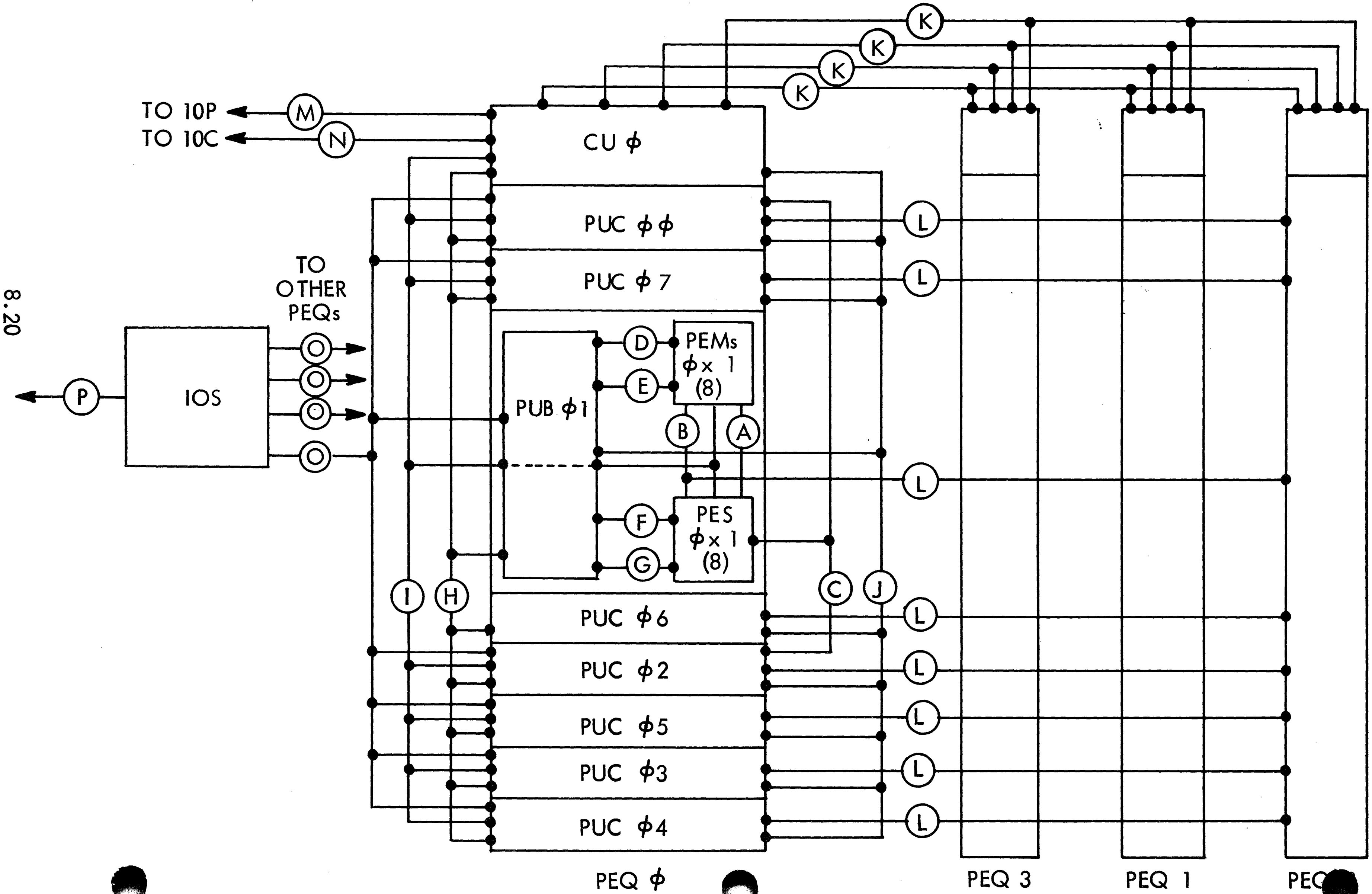
    DESCRIPTORS

    STORE BUFFER

ILLIAC IV ORGANIZATION



SYSTEM DATA INTERCONNECTIONS



SYSTEM DATA INTERCONNECTIONS - II

(A) A FULL WORD (64 BITS) BIDIRECTIONAL PATH BETWEEN THE PROCESSING ELEMENT AND ITS OWN MEMORY MODULE FOR DATA FETCHING AND STORING.

(B) A PARTIAL WORD (16 BITS), UNIDIRECTIONAL PATH BETWEEN THE PROCESSING ELEMENT AND ITS OWN MEMORY MODULE FOR ALL ARRAY MEMORY ADDRESSING.

(C) A FULL WORD (64 BITS) BIDIRECTIONAL PATH BETWEEN THE PROCESSING ELEMENT AND EACH OF ITS FOUR DESIGNATED ORTHOGONAL NEIGHBORS FOR INTERNETWORK DATA TRANSFERS.

(D) A B-WORD (256 BITS) UNIDIRECTIONAL PATH BETWEEN EACH MEMORY MODULE AND THE PROCESSING UNIT BUFFER (PUB) FOR TRANSFERS TO IOS AND THE CU.

(E) A 2-WORD (128 BITS) UNIDIRECTIONAL PATH BETWEEN THE PROCESSING UNIT BUFFER OF THE PROCESSING UNIT CABINET AND THE PROCESSING ELEMENT MEMORIES FOR I/O STORES.

(F) A 2-WORD (128 BITS) BIDIRECTIONAL PATH BETWEEN TWO PROCESSING UNITS AND THE PROCESSING UNIT BUFFER FOR INTERQUADRANT ROUTING.

(G) A 1-WORD (64 BITS) UNIDIRECTIONAL PATH BETWEEN THE PROCESSING UNIT BUFFER AND ALL EIGHT PROCESSING UNITS IN THE CABINET (CDB).

(H) A FULL WORD (64 BITS) UNIDIRECTIONAL PATH FROM THE CONTROL UNIT TO EACH OF ITS EIGHT PROCESSING UNIT CABINETS FOR OPERAND BROADCASTING, MEMORY ADDRESSING AND SHIFT COUNT TRANSFERS.

(I) A 200-BIT (APPROXIMATELY) UNIDIRECTIONAL PATH FOR CONTROL UNIT SEQUENCING OF THE PROCESSING ELEMENT QUADRANT.

(J) AN 8-WORD (512 BITS) UNIBIDIRECTIONAL PATH (ONE WORD FROM EACH PUB) FOR DATA TRANSFERS TO THE CONTROL UNIT.

(K) A FULL WORD (72 BITS) BIDIRECTIONAL PATH BETWEEN EACH OF THE FOUR CONTROL UNITS IN THE SYSTEM FOR SYNCHRONIZING AND FOR THE DISTRIBUTION OF COMMON OPERANDS IN THE UNITED ARRAY MODE.

(L) A FULL WORD (64 BITS) BIDIRECTIONAL PATH BETWEEN ADJACENT PROCESSING ELEMENT CABINETS IN ALL FOUR QUADRANTS FOR INTERQUADRANT ROUTING.

SYSTEM DATA INTERCONNECTIONS - II (Cont)

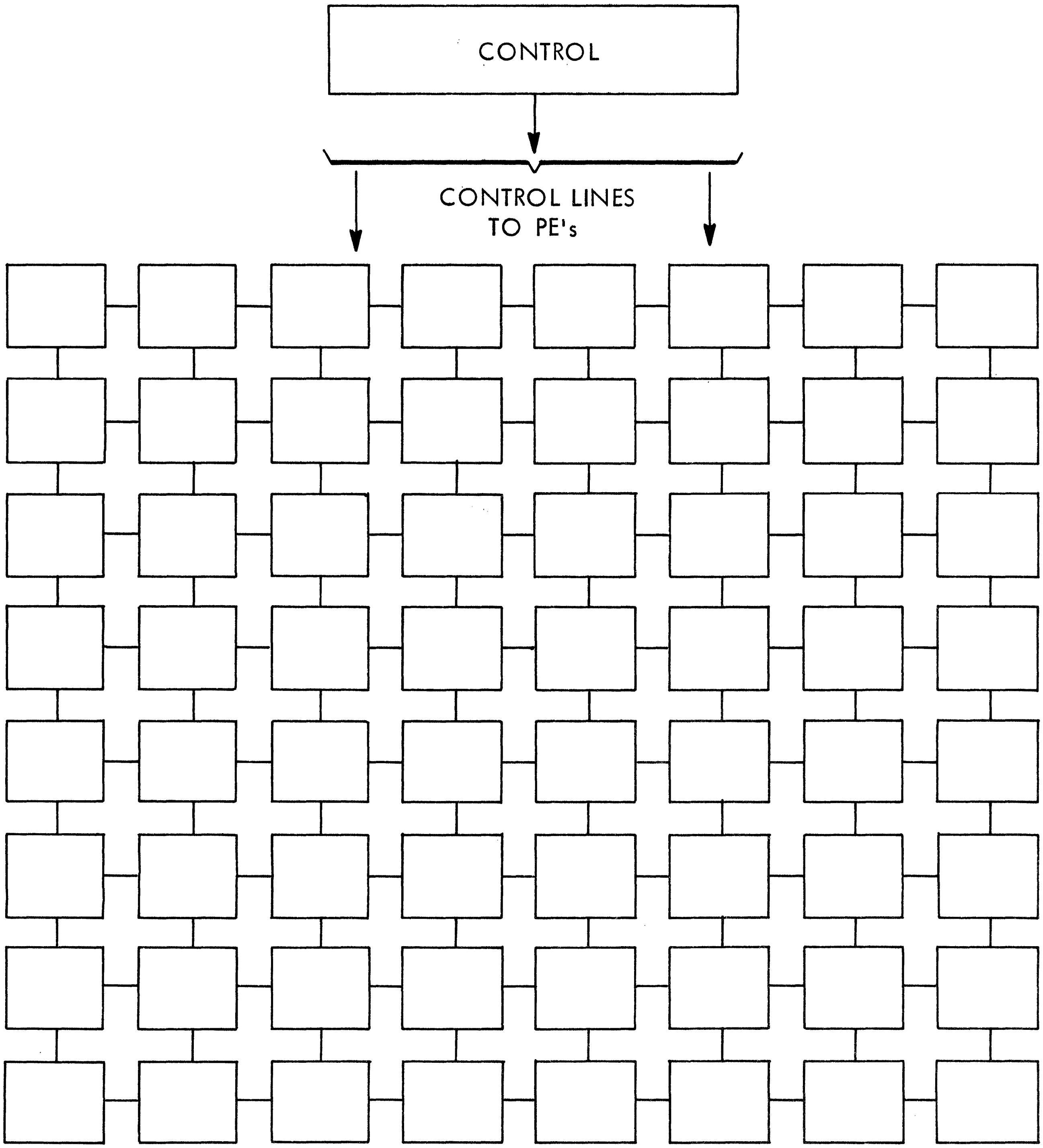
(M) A FULL WORD (64 BITS) BIDIRECTIONAL PATH BETWEEN THE FOUR CONTROL UNITS AND THE I/O SUBSYSTEM.

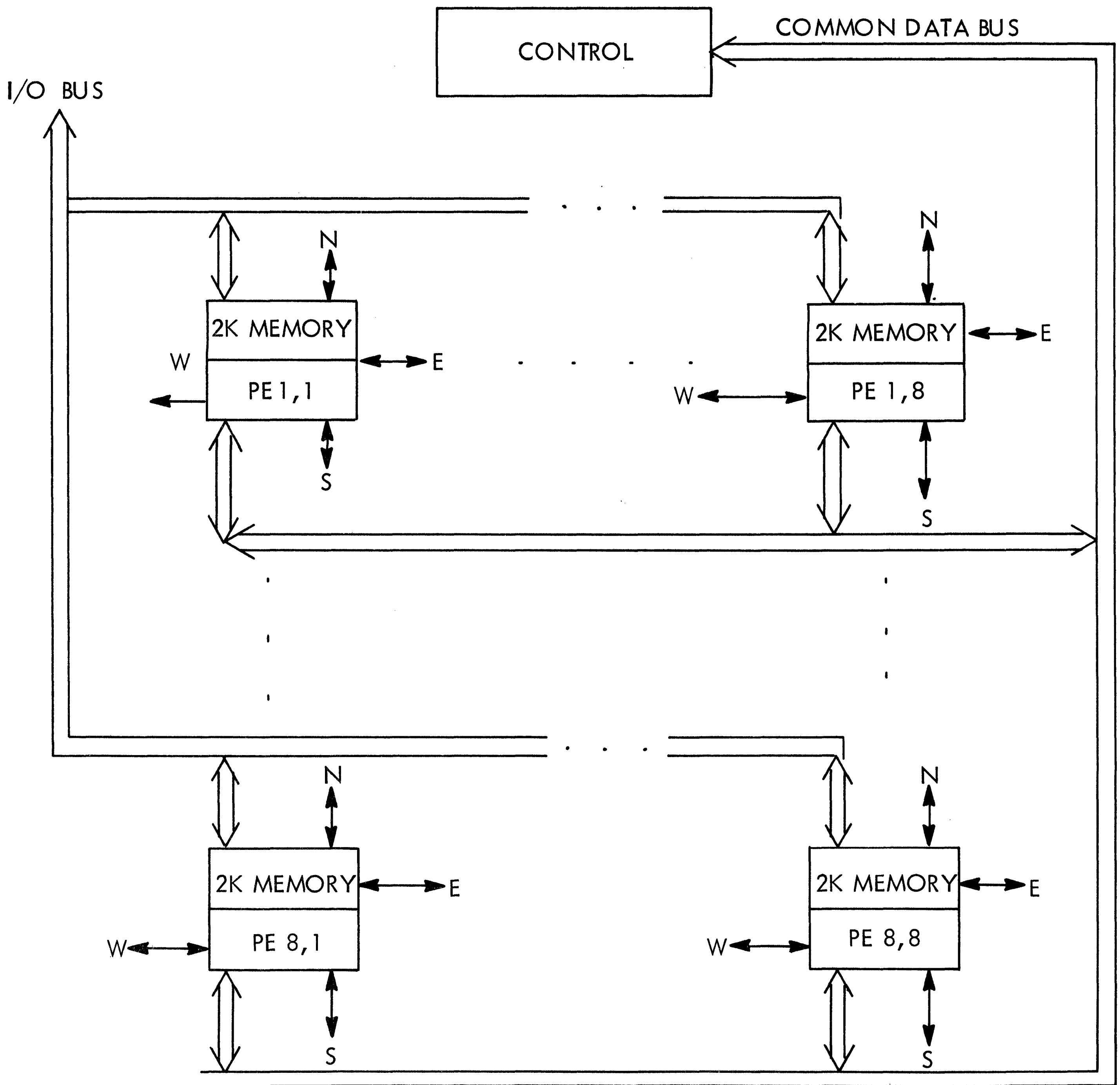
(N) A PART WORD (32 BITS) UNIDIRECTIONAL PATH BETWEEN THE FOUR CONTROL UNITS AND THE I/O CONTROLLER FOR MEMORY ADDRESSING.

(O) A 16-WORD (1024 BITS) BIDIRECTIONAL PATH BETWEEN THE INPUT/OUTPUT SWITCH AND EACH PROCESSING ELEMENT QUADRANT.

(P) A 16-WORD (1024 BITS) BIDIRECTIONAL PATH BETWEEN THE INPUT/OUTPUT SWITCH AND THE I/O SUBSYSTEM.

ILLIAC IV SUBARRAY




CONTROL
COMMON DATA BUS
I/O BUS
N
2K MEMORY
PE 1,1
W
E
S
N
2K MEMORY
PE 1,8
W
E
S
N
2K MEMORY
PE 8,1
W
E
S
N
2K MEMORY
PE 8,8
W
E
S

DISCUSSION OF ARRAY PROCESSORS

- WHERE DEALING WITH ARRAYS, VERY HIGH PERFORMANCE IS POSSIBLE (UP TO 256 TIMES A VERY HIGH PERFORMANCE SERIAL SYSTEM)
- DATA PLACEMENT CRITICAL IN ILLIAC IV BECAUSE OF LIMITATIONS OF SYSTEM CONNECTIVITY
- INTRODUCES CONCEPT OF PROCESSOR-RELATIVE ADDRESSING.
- CONTROL PROBLEMS COMPOUNDED WHEN INDEXING EXCEEDS DIMENSIONS OF ARRAYS
- ULTIMATE LIMITATION IS HIGHLY PARALLEL ACCESS MEMORY,

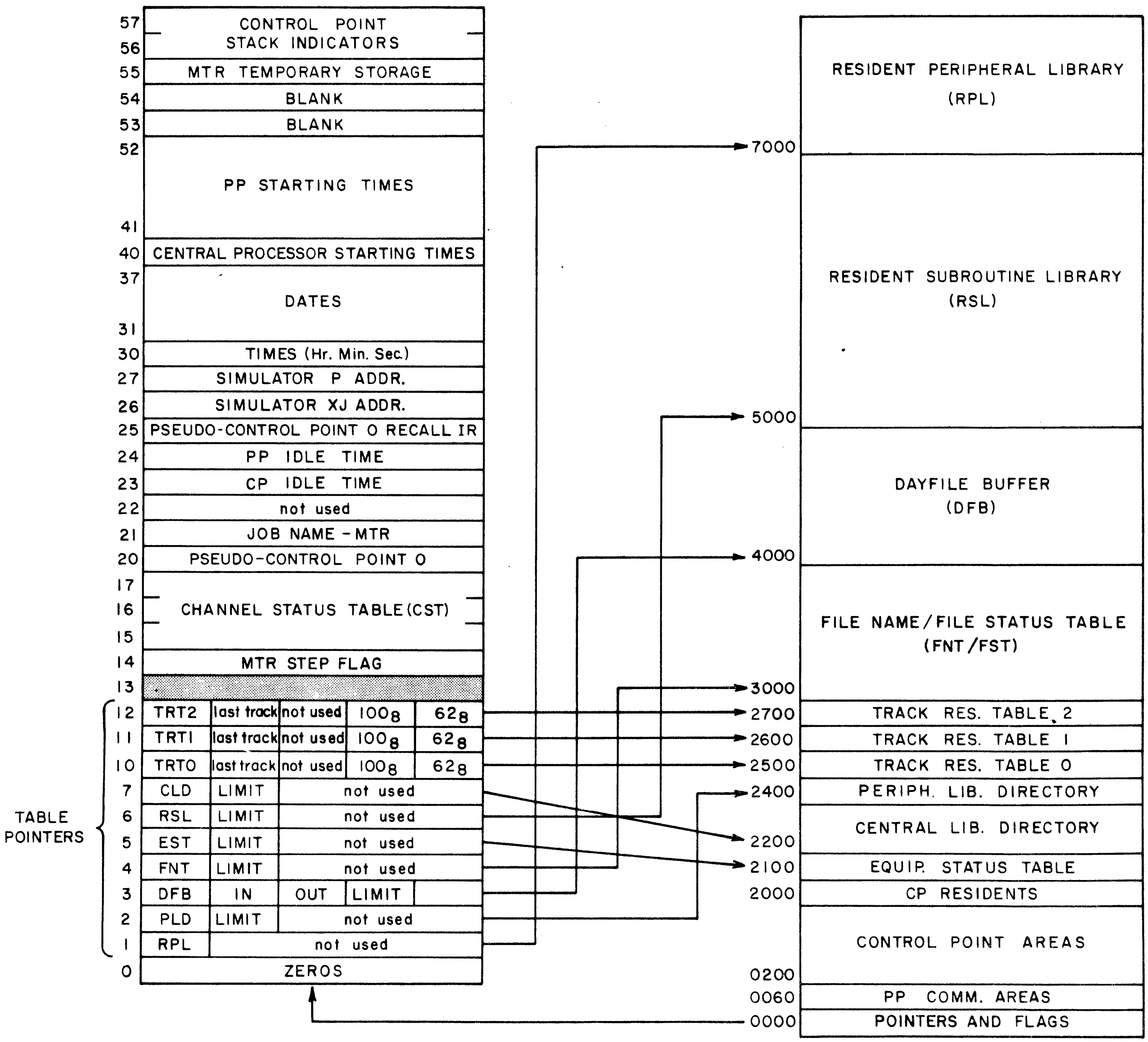
  WITH ILLIAC IV CONNECTIVITY, ONLY 4 PORTS NEEDED FOR EACH MEMORY MODULE

  WITH SAME NUMBER OF PE'S AS A VECTOR CONNECTIVITY EACH MEMORY WOULD REQUIRE 64 PORTS
- EFFICIENCY DEPENDENT ON SOLUTION METHOD ISOMORPHISM WITH STRUCTURE

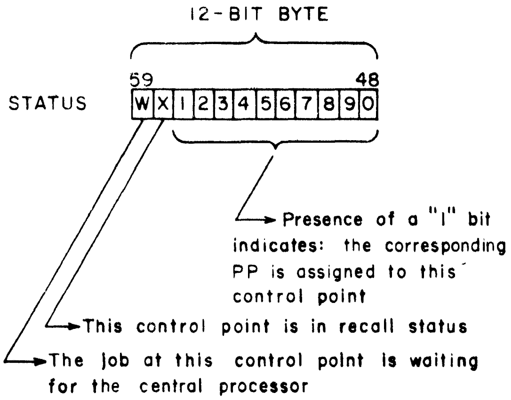
PERIPHERAL PROCESSOR MEMORY ALLOCATION

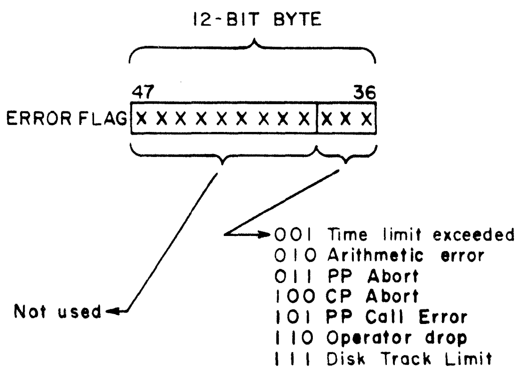
Address	PP1-8	PP9	PP0
0000	Temporary Storage	Temporary Storage	Temporary Storage
0075	Communication Area Addresses	Communication Area Addresses	Communication Area Addresses
0100	Peripheral Resident Program	Peripheral Resident Program	Peripheral Resident Program
0773	Basic Transient Programs	System Display	System Monitor
1773	Equipment Driver Overlays	System Display	System Monitor
7777			

RESIDENT CENTRAL STORAGE (TYPICAL)



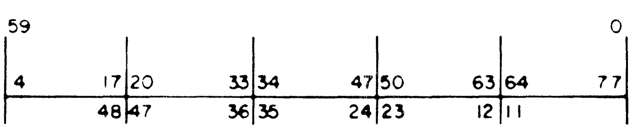
CONTROL POINT AREAS AND EXCHANGE JUMP AREA



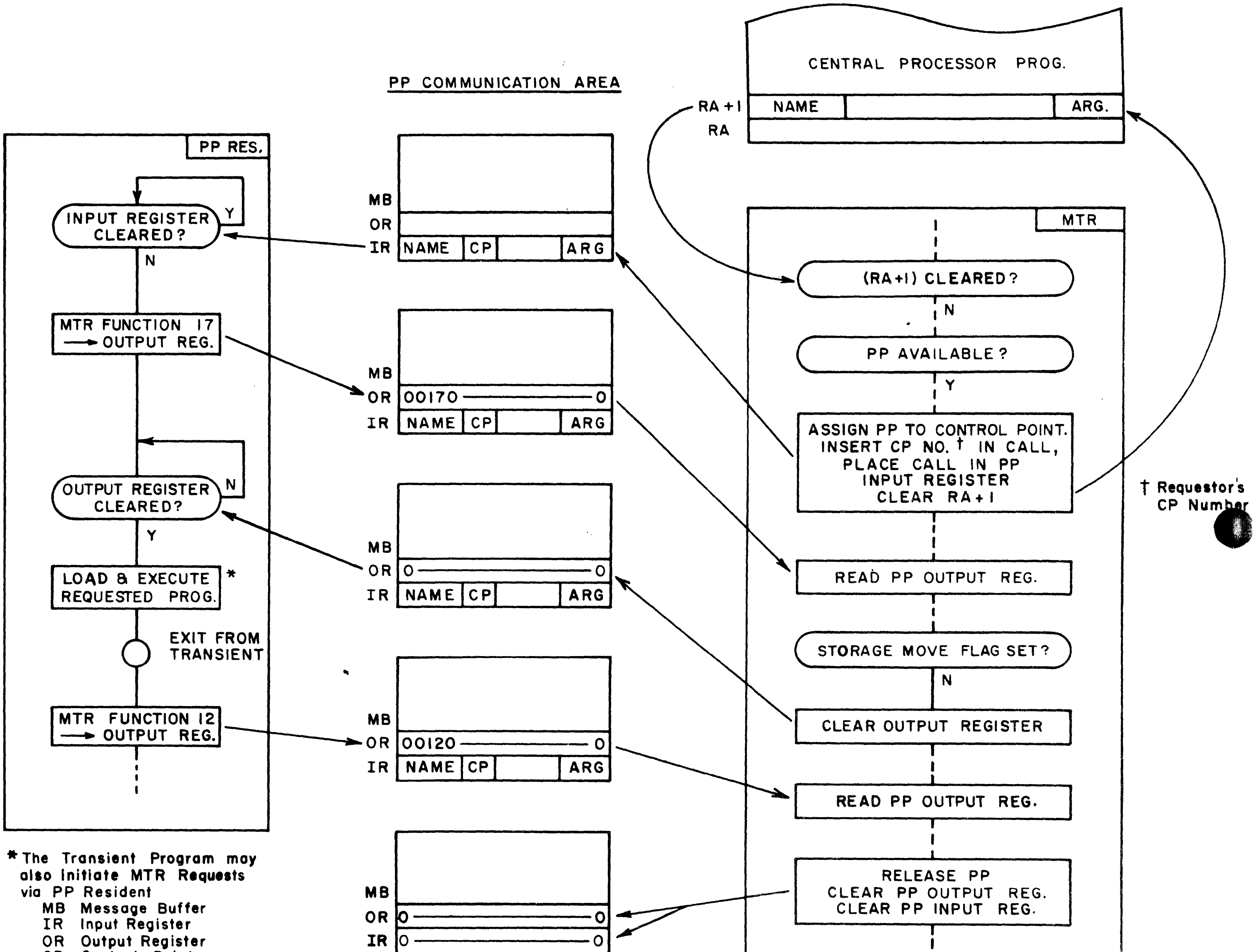


					Words	
	Program Address (P)	A0 (Address Registers)			0	EXCHANGE PACKAGE (words 0–17)
	Reference Address (RA)	A1	B1 (Increment Register)		1	
	Field Length (FL)	A2	B2		2	
	Exit Mode (EM)	A3	B3		3	
		A4	B4		4	
		A5	B5		5	
		A6	B6		6	
		A7	B7		7	
X0 (Operand Registers)					10	
X1					11	
X2					12	
X3					13	
X4					14	
X5					15	
X6					16	
X7					17	
STATUS	ERROR FLAG	stor. move flag	RA (Hundreds)	FL (Hundreds)	20	
JOB NAME (DISPLAY CODE)				next cont. stat.	21	POINTER TO NEXT STATEMENT IN BUFFER
PRIORITY	MSG. COUNT	TRACK COUNT	TIME LIMIT	op. assign. equip	22	
		CP TIME (SECS)		(MSECS)	23	CURRENT RUNNING TIMES
		PP TIME (SECS)		(MSECS)	24	
PP RECALL REG.					25	HOLDS PP INPUT REGISTER DURING PP RECALL
SENSE SWITCHES, LIGHTS					26	
EQUIPMENT ASSIGNED					27	
LAST DAYFILE MESSAGE (OR CONSOLE MESSAGE)					30–37	
CONTROL STATEMENT BUFFER (PACKED DISPLAY CODE)					40–177	

EQUIPMENT ASSIGNMENT



MONITOR/PP COMMUNICATION AREA



* The Transient Program may also initiate MTR Requests via PP Resident

- MB Message Buffer
- IR Input Register
- OR Output Register
- CP Control Point
- ARG Argument

DAYFILE DISPLAY

Hours — Min — ACTUAL TIME — Sec.

Name of JOB to which message belongs

SYSTEM TAPE LABEL

JOB NAME

This column represents the time each control statement was requested for execution. (A total of 32 lines may be contained on the dayfile)

```
 00.12.55.  PROGRAMMING CHECKOUT
00.00.16.  MERGE  .  MERGE,7,1000,1000.
00.00.17.  MERGE  .  ASSIGN 50,A.
00.00.17.  MERGE  .    (50 ASSIGNED)
00.00.17.  MERGE  .  ASSIGN 51,B.
00.00.17.  MERGE  .    (51 ASSIGNED)
00.00.18.  MERGE  .  REWIND (F)
00.00.25.  MERGE  .  REWIND (F)
00.00.25.  MERGE  .  COPYBF (F.D)
00.00.27.  BETA   .  READ.
00.00.30.  MERGE  .  REWIND (F)
00.00.30.  MERGE  .  REWIND (F)
00.00.30.  MERGE  .  CP 006.(D) SEC.
00.00.30.  MERGE  .  PP 019.421 SEC.
00.00.30.  MERGE  .  PRINT.
00.00.30.  MERGE  .  PP 000 SEC.
00.00.31.  BETA   .  PP 015 SEC.
00.00.31.  BETA   .  BETA,77,70000,50000.
00.00.31.  BETA   .  DIS.
00.01.04.  BETA   .  INPUT.
00.01.05.  BETA   .  LOC.
00.01.11.  BETA   .  BUFFER ARG ERROR.
00.01.12.  BETA   .  CP 002.575 SEC.
00.01.12.  BETA   .  PP 020.265 SEC.
00.01.12.  BETA   .  PRINT.
00.01.24.  BETA   .  PP 011 SEC.
00.04.05   BETA   .  READ
00.04.10   BETA   .  PP 015 SEC.
00.04.10   BETA   .  BETA,77,70000,50000.
00.04.10   BETA   .  DIS.
00.04.40   BETA   .  INPUT.
00.04.40   BETA   .  LOC.
```

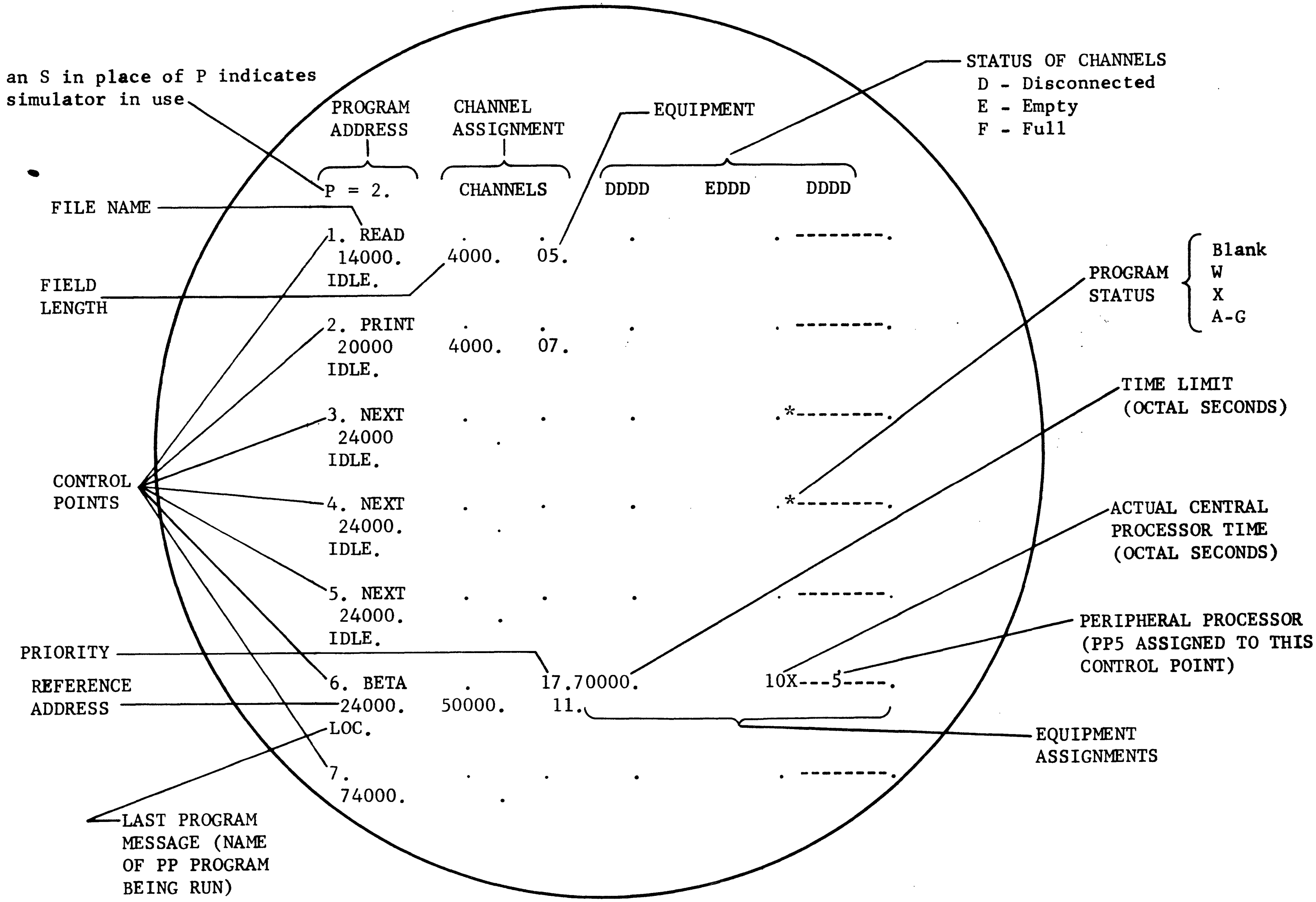
This column represents the control statements introduced via card input and contains the system's history.

A summary of the day's total run may be printed out upon request.

New dayfile information appears at the bottom of the screen automatically; old dayfile information is deleted at the top of the column as new times are entered into the dayfile.

NOTE: Dayfile display data will appear on the printout at the end of each job automatically

JOB STATUS DISPLAY



STORAGE DISPLAYS

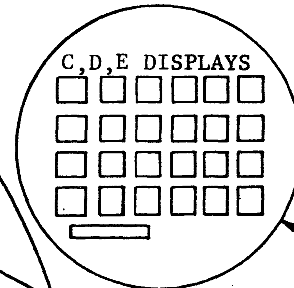
F,G DISPLAYS

00.17.21. PROGRAMMING CHECKOUT

ADDRESS — CONTENTS OF ADDRESS

010000	32670	62330	60133	70520
010001	30645	26010	14316	53261
010002	07125	06232	00101	43165
010003	32610	70401	00235	00000
010004	20002	06602	00053	11413
010005	02000	76101	00010	00000
010006	00000	00000	00000	00000
010007	35022	40000	00000	00126
010010	01000	00002	00224	13024
010011	12020	51430	03100	70705
010012	02002	56130	17000	00200
010013	24010	32100	00020	02561
010014	02002	56102	00210	13004
010015	10660	56730	00346	23464
010016	30615	46334	05140	06010
010017	30625	41330	05341	43053
010020	10063	15510	06315	41602
010021	62103	06434	13306	53414
010022	30531	00631	55100	65154
010023	16036	21001	00200	00000
010024	00000	00000	00000	00100
010025	00003	03102	00074	13032
010026	76000	20023	01140	03403
010027	30725	40430	32162	07600
010030	74002	00050	00710	02700
010031	04113	40137	04150	55501
010032	04020	67303	03000	07500
010033	02002	30112	02040	50200
010034	23011	22004	12305	10200
010035	07513	62210	03352	10100
010036	21000	00036	03110	50407
010037	30321	63076	00010	02115

DISPLAY AREA FOR OPERATOR TYPEIN (Always on left screen)



FOUR GROUPS OF FIVE-OCTAL DIGITS--(OR FIVE GROUPS OF FOUR-OCTAL DIGITS)

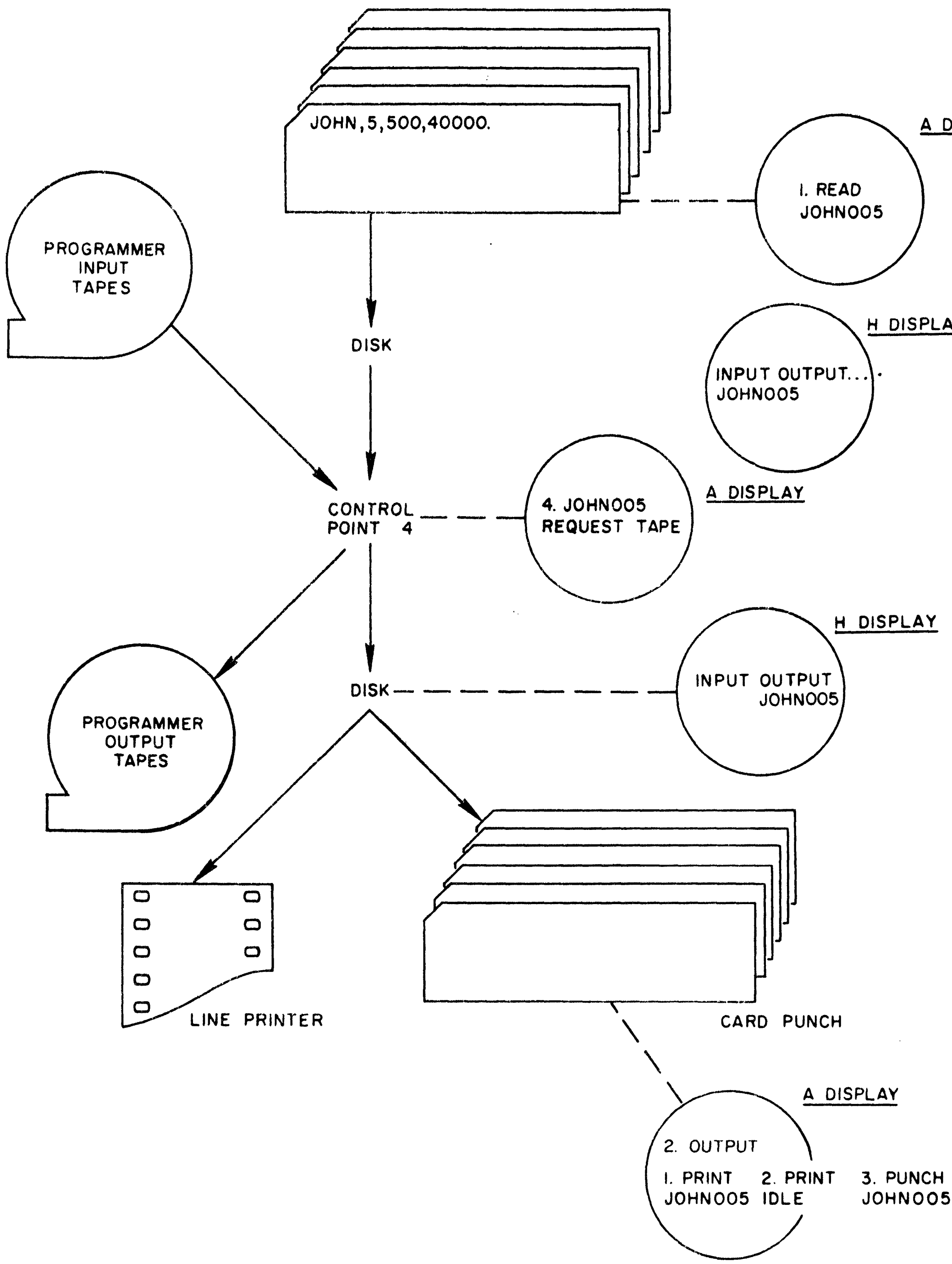
H DISPLAY

INPUT	OUTPUT
Job 1	Job N
Job 2	Job X
	Job Y

Job Names (Input stack)

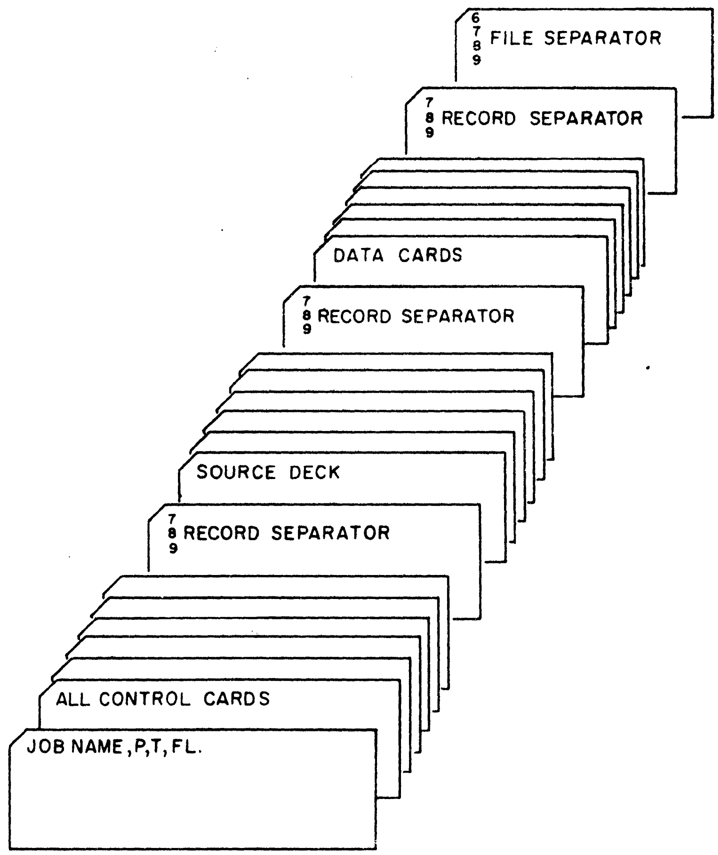
Jobs stacked for output (If no jobs are waiting for output none will appear)

JOB FLOW


JOHN,5,500,40000.
A DISPLAY
I. READ
JOHN005
PROGRAMMER
INPUT
TAPES
DISK
H DISPLAY
INPUT OUTPUT...
JOHN005
A DISPLAY
CONTROL
POINT 4
4. JOHN005
REQUEST TAPE
H DISPLAY
PROGRAMMER
OUTPUT
TAPES
DISK
INPUT OUTPUT
JOHN005
LINE PRINTER
CARD PUNCH
A DISPLAY
2. OUTPUT
I. PRINT
JOHN005
2. PRINT
IDLE
3. PUNCH
JOHN005

TYPICAL DECK SEQUENCE

Card arrangement to begin a job, separate records, and terminate.



FORTRAN LOAD AND RUN

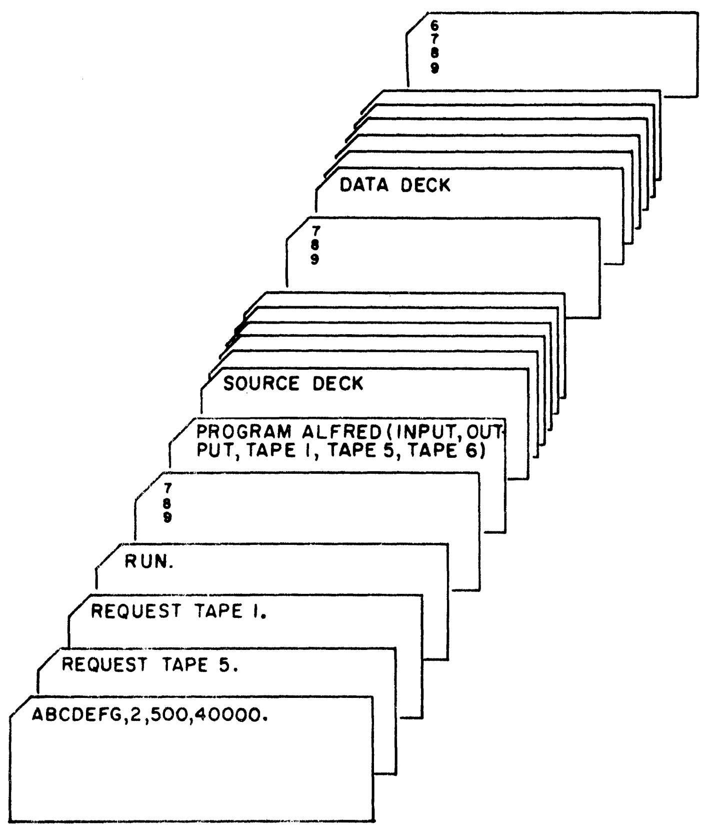
Card arrangement for a FORTRAN Load and Run job:

Tape references:

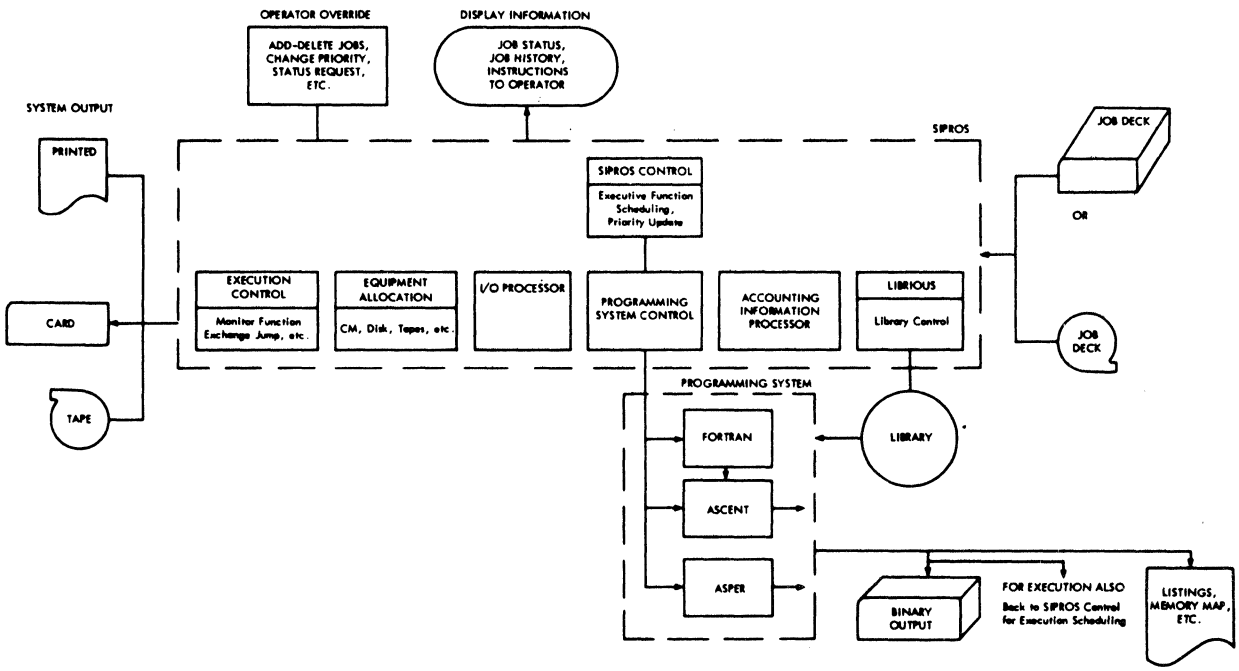
TAPE1 - assumed input tape which operator loads on a particular unit

TAPE5 - output tape drawn from tape pool

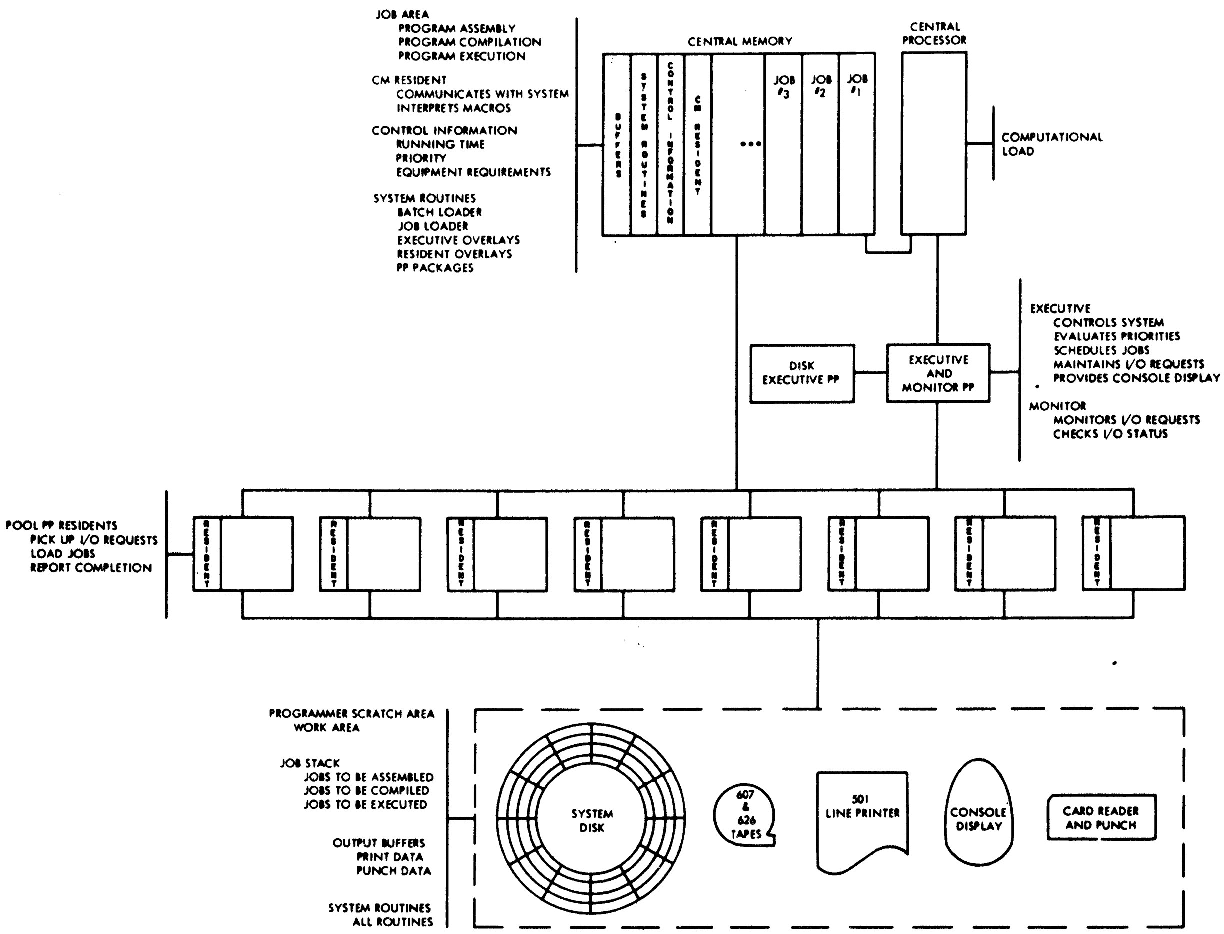
TAPE6 - scratch file on disk



SIPROS ENVIRONMENT



SYSTEMS COMPONENTS



JOB CONTROL

CONTROL CARDS

(*REQUIRED CONTROL CARDS)

JOB IDENTIFICATION
- * JOB NAME AND ACCOUNT NUMBER
- PRIORITY
- CENTRAL PROCESSOR RUNNING TIME LIMIT

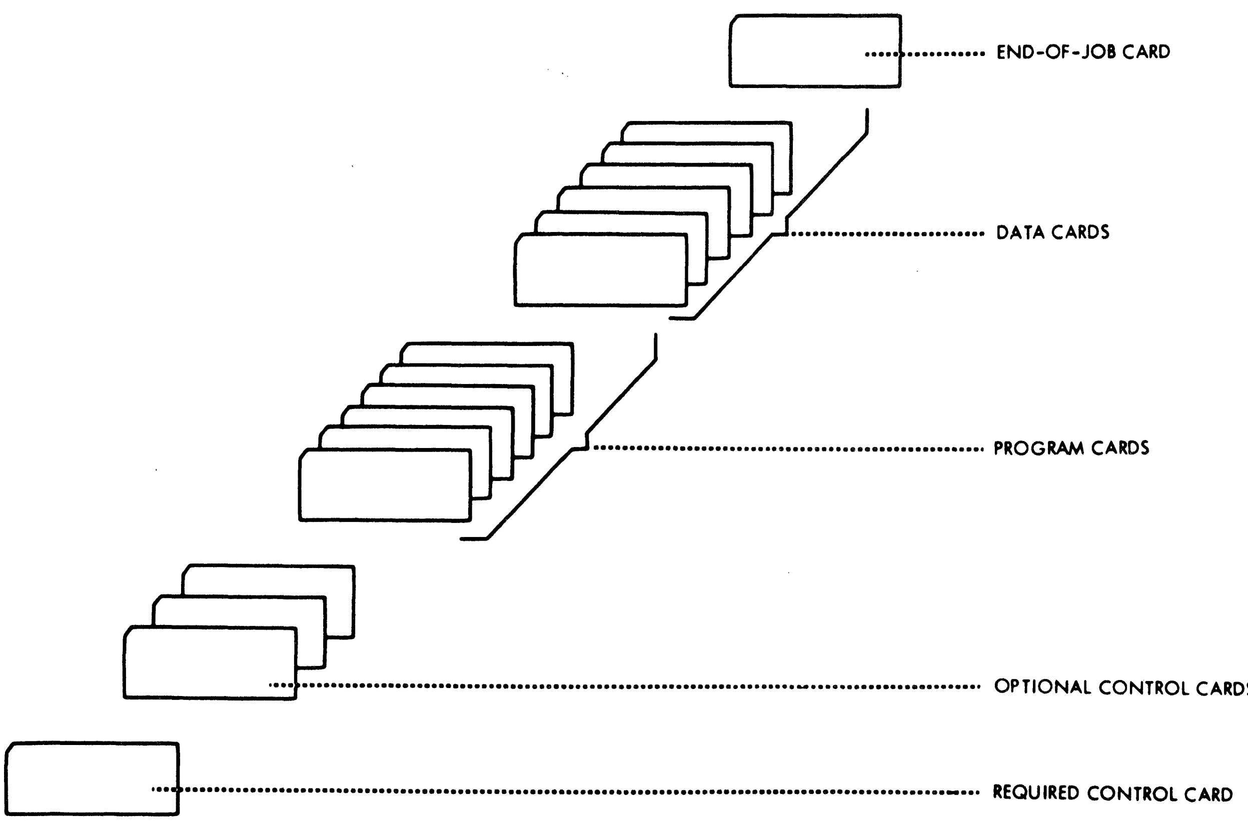
EQUIPMENT
- SCRATCH TAPE
- INPUT TAPE
- OUTPUT TAPE
- PRINTER
- DISK
- CARD READER
- CARD PUNCH
- PERIPHERAL PROCESSOR
- VARIATIONS
  - VARIABLE vs FIXED REQUIREMENTS
  - EQUIPMENT EXCHANGE
  - SPECIFIC ASSIGNMENT

MEMORY ESTIMATE
- CENTRAL MEMORY
  - FIXED
  - VARIABLE
- DISK MEMORY
  - FIXED
  - VARIABLE

DEBUGGING
- MEMORY DUMP
- MEMORY MAP
- CONSOLE DEBUGGING
- ERROR HALT CONDITIONS

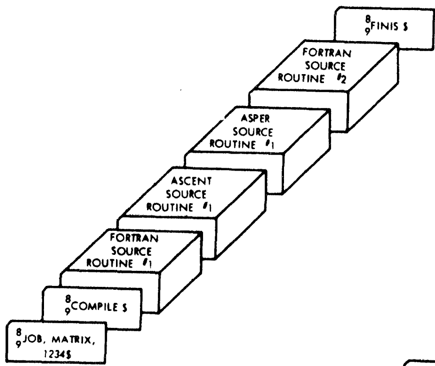
OTHER
- IGNORE EXPONENT OVERFLOW
- IGNORE INDEFINITE RESULT
- IGNORE EXPONENT OVERFLOW AND INDEFINITE RESULT
- COMPILE PROGRAM
- * FINIS

CARD DECK LAYOUT

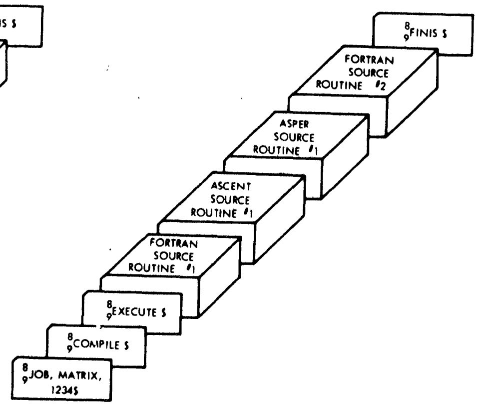


JOB DECK EXAMPLES

CASE A: JOB COMPILATION

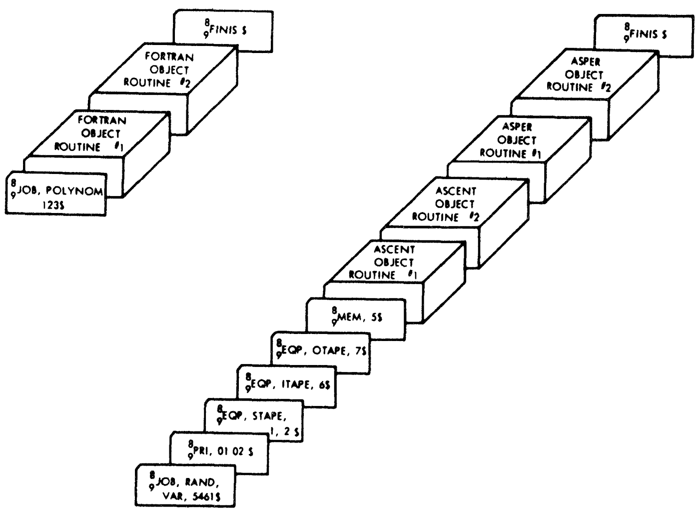


CASE B: JOB COMPILATION AND EXECUTION

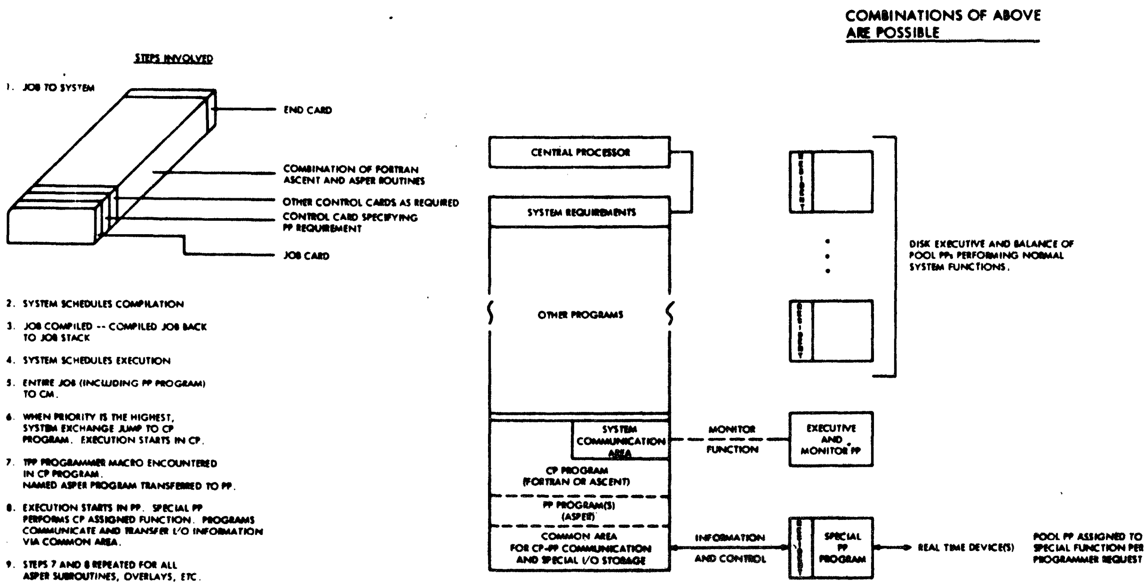


CASE C: JOB EXECUTION (NO OPTIONAL CARDS)

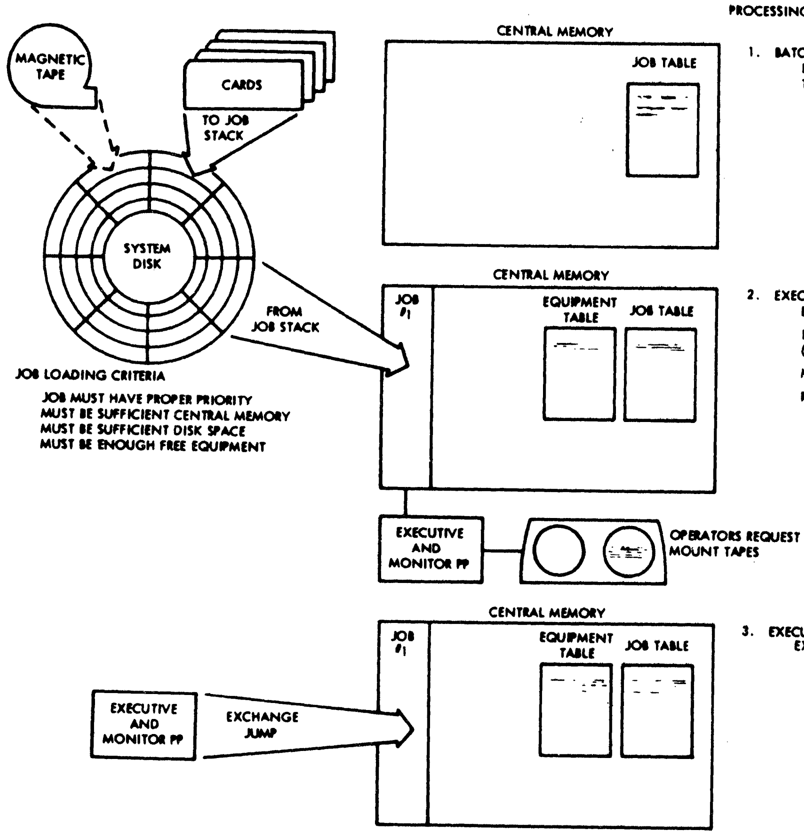
CASE D: JOB EXECUTION (OPTIONAL CONTROL CARDS)



USE OF MEMORY



PROCESSING STEPS



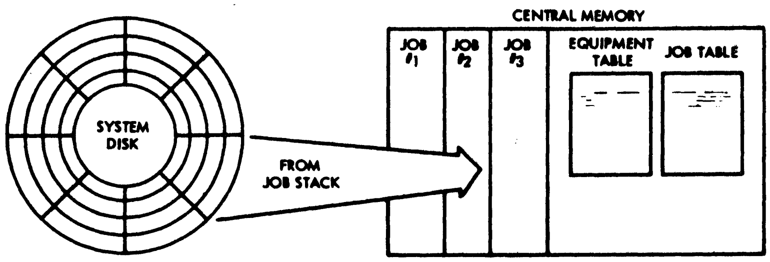
PROCESSING STEPS

1. BATCH LOADER
   LOADS JOB INTO JOB STACK ON DISK FROM CARDS OR TAPE MAKES ENTRY IN JOB TABLE FOR EACH JOB LOADED

2. EXECUTIVE
   EXAMINES JOB TABLE FOR JOBS TO BE LOADED INTO CM
   INSTRUCTS JOB LOADER TO LOAD JOB WITH HIGHEST PRIORITY (IF IT MEETS LOADING REQUIREMENTS)
   MAKES EQUIPMENT ASSIGNMENTS IN EQUIPMENT TABLE
   REQUESTS OPERATOR TO PREPARE EQUIPMENT

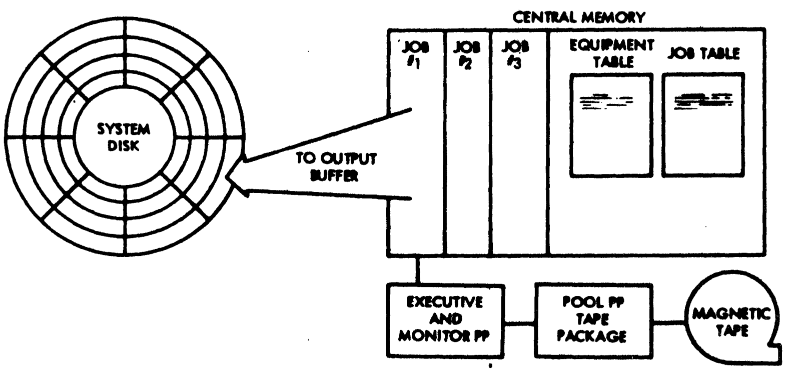
3. EXECUTIVE
   EXCHANGE JUMPS TO JOB TO BE EXECUTED

PROCESSING STEPS (CONT.)



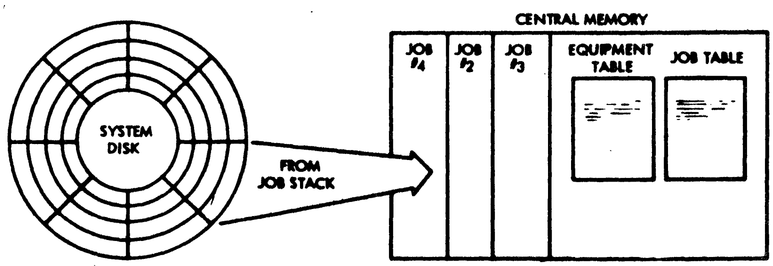
4. EXECUTIVE
   INSTRUCTS JOB LOADER TO LOAD OTHER JOBS INTO CENTRAL MEMORY UNTIL IT IS FULL

   MULTIPROCESSES JOBS IN CENTRAL MEMORY



5. EXECUTIVE
   DIRECTS OUTPUT DATA FOR PRINTER AND PUNCH TO OUTPUT BUFFER ON DISK

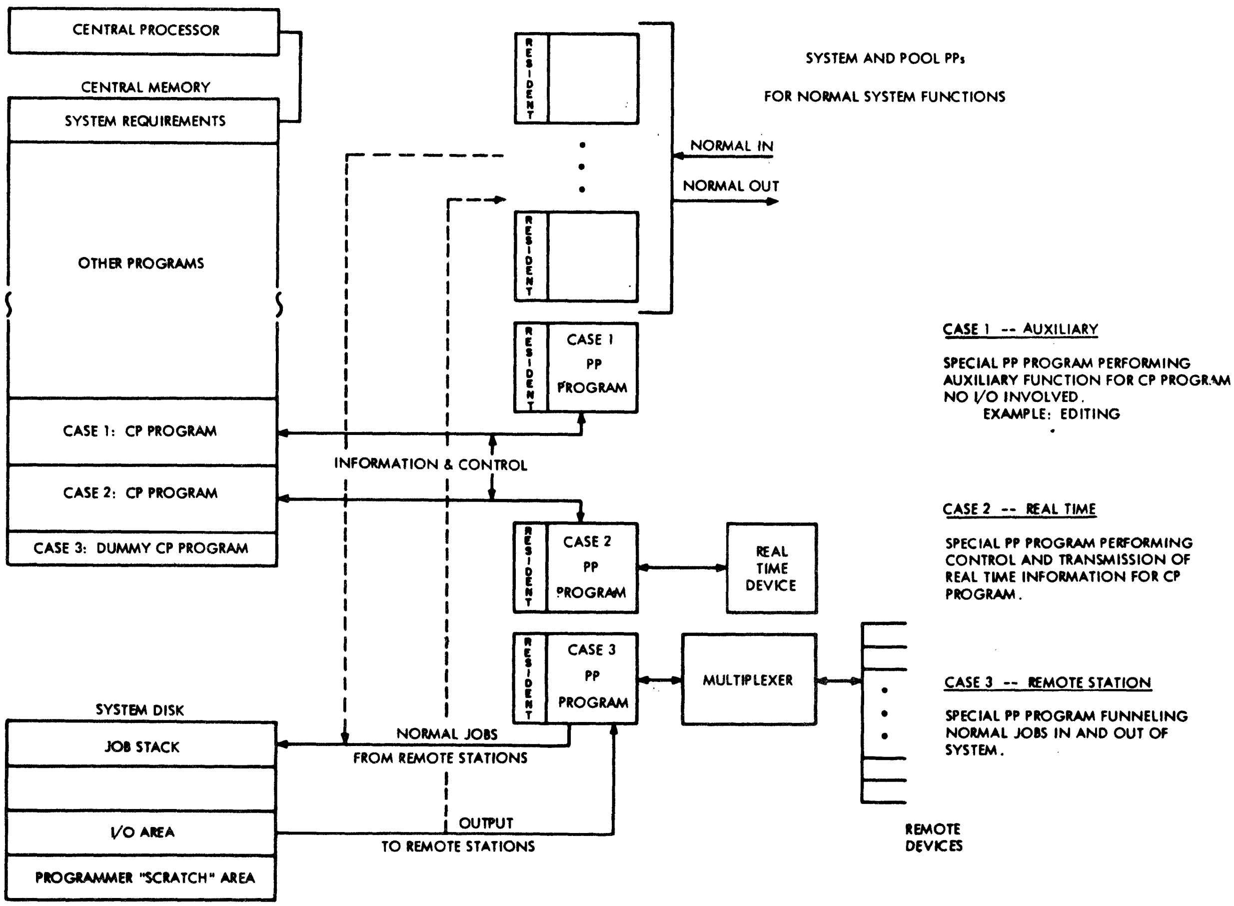
   DIRECTS OUTPUT DATA FOR TAPE TO POOL PP WHICH WRITES TAPE



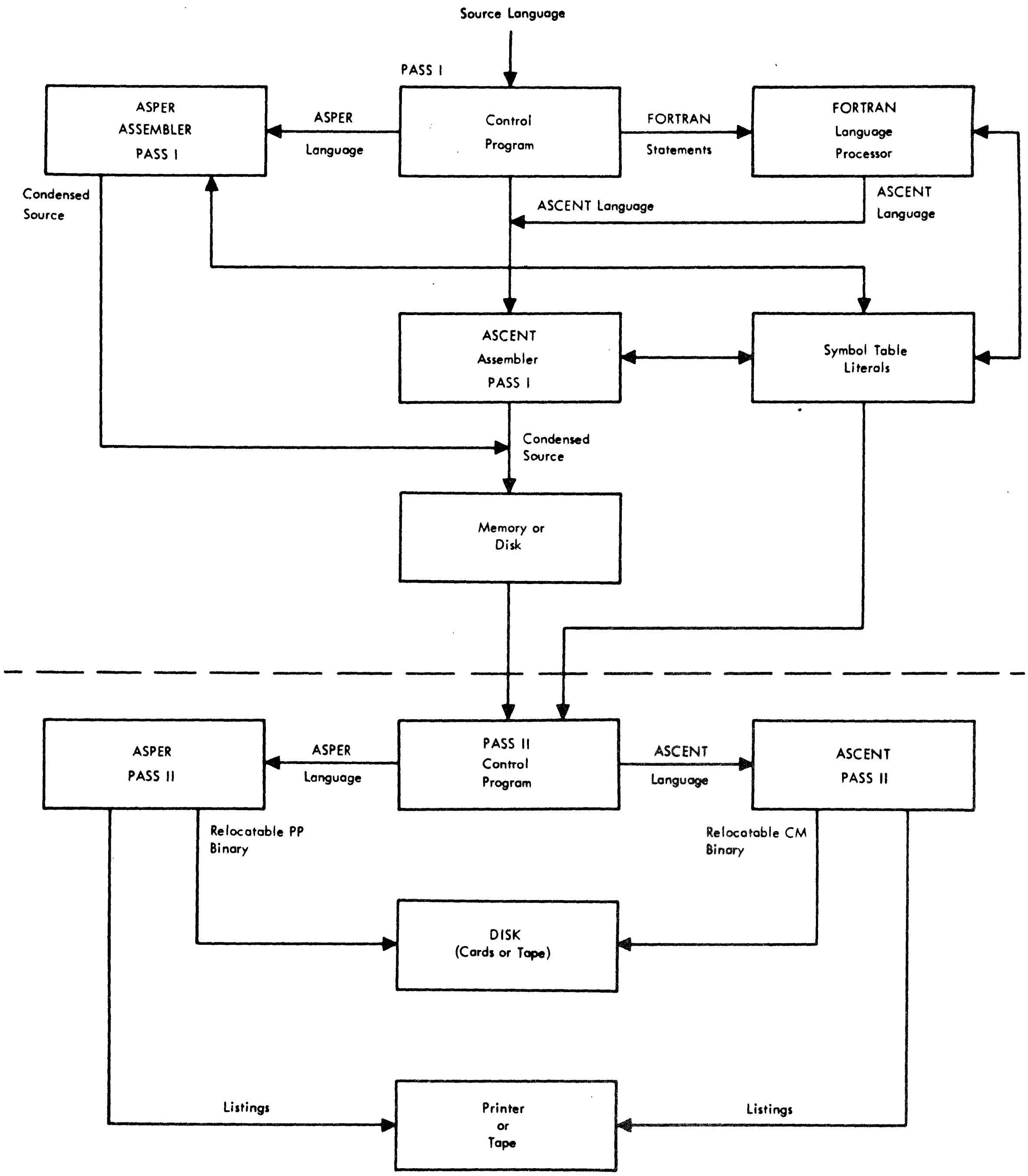
6. EXECUTIVE
   SCHEDULES NEW JOB FOR CM WHEN JOB TERMINATES

   INSTRUCTS JOB LOADER TO LOAD NEW JOB FROM JOB STACK ON DISK INTO CM

REMOTE HOOKUPS



LANGUAGE PROCESSING



SCOPE FEATURES

- CHIPPEWA SUPERVISOR
- DATA MANAGEMENT
- ADVANCED LOADER
- REMOTE PACKAGE

STORAGE ASSIGNMENT DURING SEGMENTATION

Loading Order	Segment Level	Contents of User's Job Area in Memory after Loading of Segment			
1	0	SEG 0		Unused Storage Area	
2	3	SEG 0	SEG 3		
3	4	SEG 0	SEG 3	SEG 4	
4	9	SEG 0	SEG 3	SEG 4	SEG 9
5	2	SEG 0	SEG 2		
6	1	SEG 0	SEG 1		
7	5	SEG 0	SEG 1	SEG 5	
8	8	SEG 0	SEG 1	SEG 5	SEG 8
9	7	SEG 0	SEG 1	SEG 5	SEG 7

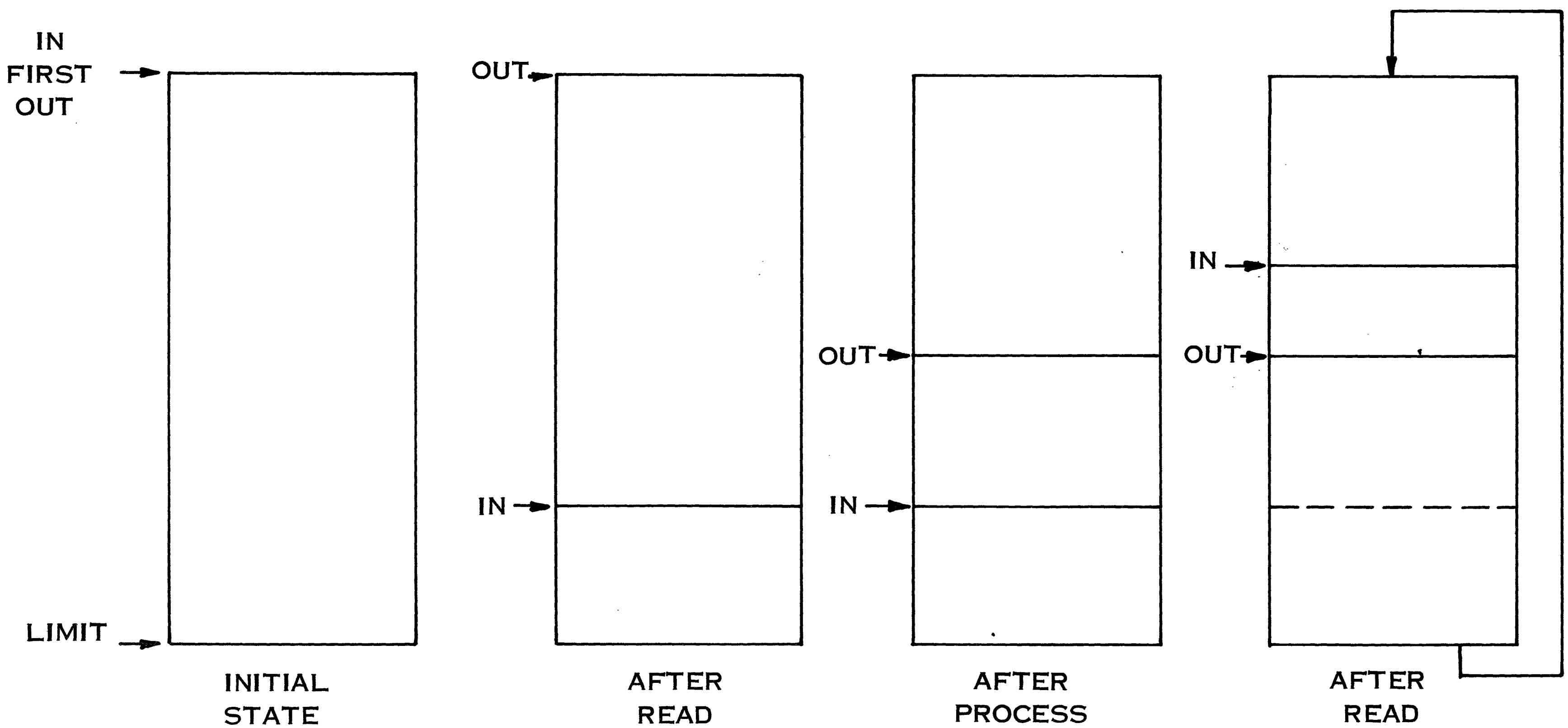
STORAGE ASSIGNMENTS FOR OVERLAYS

Loading Order	Primary Level Number	Secondary Level Number	Contents of User's Job Area in Memory after Loading of Overlay		
1	0	0	(0,0)		Unused Storage Area
2	1	0	(0,0)	(1,1)	
3	1	1	(0,0)	(1,0)	(1,1)
4	1	2	(0,0)	(1,0)	(1,2)
5	1	1	(0,0)	(1,0)	(1,1)
6	1	3	(0,0)	(1,0)	(1,3)
7	1	2	(0,0)	(1,0)	(1,2)
8	2	0	(0,0)	(2,0)	
9	2	1	(0,0)	(2,0)	(2,1)
10	2	2	(0,0)	(2,0)	(2,2)
11	3	0	(0,0)	(3,0)	
12	4	0	(0,0)	(4,0)	

FILE ENVIRONMENT TABLE (FET)

<table>
<tr><th>Bits 59</th><th>47</th><th></th><th></th><th>44</th><th></th><th>35</th><th>32</th><th>29</th><th>23</th><th>17   0</th><th>Words</th></tr>
<tr><td colspan="10">logical file name (lfn)</td><td>code and status</td><td>1</td></tr>
<tr><td>device type</td><td>r</td><td>n</td><td>u p</td><td>e p</td><td></td><td colspan="3">disposition code</td><td>ℓ</td><td>FIRST</td><td>2</td></tr>
<tr><td colspan="10">0</td><td>IN</td><td>3</td></tr>
<tr><td colspan="10">0</td><td>OUT</td><td>4</td></tr>
<tr><td>FNT pointer</td><td colspan="6">record block size</td><td colspan="3">physical record unit size</td><td>LIMIT</td><td>5</td></tr>
<tr><td></td><td colspan="7">working storage fwa</td><td colspan="2"></td><td>working storage lwa+1</td><td>6</td></tr>
<tr><td colspan="8"></td><td colspan="3">record request/return information</td><td>7</td></tr>
<tr><td>record number</td><td colspan="7"></td><td colspan="2">index length</td><td>index address</td><td>8</td></tr>
<tr><td></td><td colspan="7">EOI address</td><td colspan="2"></td><td>error address</td><td>9</td></tr>
<tr><td colspan="11">Label file name (first 10 chars)</td><td>10</td></tr>
<tr><td colspan="11">Label file name (last 10 chars)</td><td>11</td></tr>
<tr><td>edition number</td><td colspan="7">retention cycle</td><td colspan="3">creation date</td><td>12</td></tr>
<tr><td>position number</td><td colspan="7">multi-file name</td><td colspan="3">reel number</td><td>13</td></tr>
</table>

BUFFERING DURING A READ



RESPOND COMMANDS TO SCOPE

- COMPILE
- ASSEMBLE
- EXECUTE
- COPY
- SUBMIT

SYSTEM ACTION REQUESTS

- MEMORY
- CKPT
- RECALL
- MESSAGE
- ENDRUN
- ABORT
- LOADER
- TIME/DATE

FILE ACTION REQUESTS

- REQUEST
- OPEN
- CLOSE
- EVICT
- READ
- WRITE
- SKIP
- BKSP
- REWIND
- UNLOAD

A RESPOND DIALOGUE

```
LOGIN JRV, 2359 Δ
          CONTINUE

FORMAT FTN 80 TAB 2,7Δ
          CONTINUE

INPUT FTN

0010 ↑↑ PROGRAM EOQ (INPUT=
          TAPE1,OUTPUT=TAPE2)Δ
0020 ↑↑ LU1=TAPE1 Δ
0030 ↑↑ LU2=TAPE2 Δ
0040 ↑5↑ READ(LU1,10)USE,POC,UC Δ
0050 ↑10↑ FORMAT (3F8.2)Δ
0060 ↑↑ IF (USE.EQ.7777)40,30Δ
0070 ↑30↑ CONTINUE Δ
0080 . . . .
     .
     .
     .
iii0 ↑ 40 ↑CALL REPORT(QTY,POC,UCOST,
          TCOST) Δ
     .
     .
     .
iij0 ↑↑ RETURN Δ
iik0 ↑↑ END      Δ
iim0 ↓ EOF       Δ
          CONTINUE

FILE EOQ, 10 TO iim0
          CONTINUE

COMPILE EOQ

          Job name from SCOPE
          Notification of job completion

LIST FILESΔ
          PRIVATE FILES
          TAPE1   150 DIS  80  1  1/1/67
          EOQ     130 DIS  80  1  3/1/67
          EOQ L   230 DIS  VL  1  2/12/67
          EOQ B    52 BIN  20  1  2/12/67
```

A RESPOND DIALOGUE

```
EXECUTE EOQ B, INPUT=TAPE1,
        OUTPUT=TAPE2 Δ
        Job name from SCOPE
        Notification of job completion

OPEN TAPE1 Δ
        CONTINUE

DISPLAY RECORD 1 TO 5  Δ

        ORDER PO       UNIT    TOTAL
          QTY  COST    COST     COST
          942  100.00  120.00  113140.00
          330    8.00   33.50  110663.00
          481    1.20    9.80    4715.00

DISPLAY RECORD 5 TO 10  Δ
          481    1.20    9.80    4715.00
          366    1.80    5.50    2014.80

TOTAL 358320.15

DISPLAY RECORD 1, 2, 5 TO 8 Δ
        ORDER  PO    UNIT  TOTAL
          QTY  COST  COST  COST
          481  1.20  9.80   4715.00
          366  1.80  5.50   2014.80

TOTAL 358320.15

OPEN FILE EOQ L  Δ
        CONTINUE

DELETE EOQ L  Δ
        CONTINUE

COPY TAPE2 TO PRINTER  Δ
        Job name from SCOPE
        Notification of job completion

LIST FILES  Δ
        PRIVATE FILES
        TAPE1  150 DIS  80  2  1/1/67
        EOQ    130 DIS  80  1  3/1/67
        EOQ B   52 BIN  20  1  2/12/67
        TAPE2    8 DIS  80  3  2/12/67

LOGOUT  Δ
        TIME 00.35.05
```

A RESPOND DIALOGUE

```
LOGIN GFC, 2106 Δ
         . CONTINUE

INPUT Δ
         0010                 ASPER MUX
         0020     TERM        EQU 12 Δ
         0030     CHAN        EQU 13B Δ
         0040     CONN        EQU 5001B Δ
         0050
         iii0     IOP         FNC CHAN, CONN Δ

         iij0                 END   Δ
         iik0 ↓  EOF Δ
         CONTINUE

FILE MUXIO Δ
         CONTINUE

ASSEMBLE MUXIO Δ
         Job name from SCOPE
         Notification of job completion

LIST FILES Δ
         PRIVATE FILES
         MUXIO      132 DIS  80  1  2/10/67
         MUXIO L    352 DIS  VL  1  2/10/67
         MUXIO B     37 BIN  20  1  2/10/67

LOGOUT Δ
         TIME 00.13.20
```

▶ TOPICS COVERED THIS SESSION

- THE INFLUENCE OF PROGRAMMING LANGUAGE ON MACHINE DESIGN - PARTICULARLY THE EFFECT OF ALGOL 60

- SEVERAL MACHINE DESIGNS REFLECTING THIS INFLUENCE

  B 5000

  KDF 9

  B 55/65/7500

▶ NEW NOTIONS IN ALGOL 60

- ORIGINS IN ALGOL '58

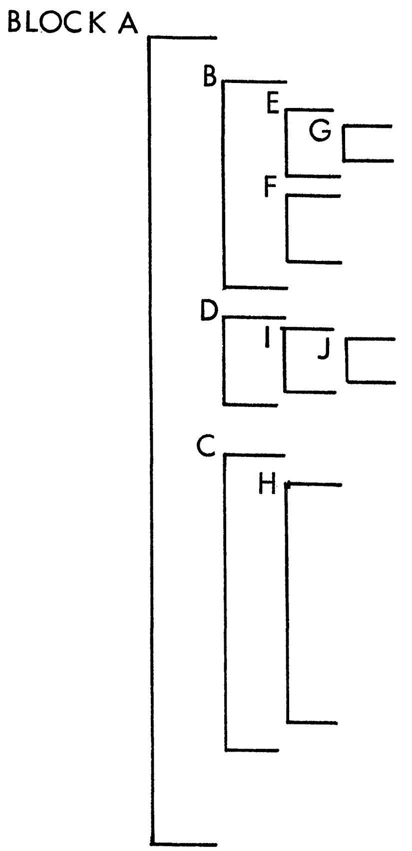
  PRODUCED BALGOL
  MAD
  NELLIAC
  JOVIAL

- BLOCK STRUCTURE
  (STATIC LEVELS)

- RECURSION IN PROCEDURES
  (DYNAMIC LEVELS)

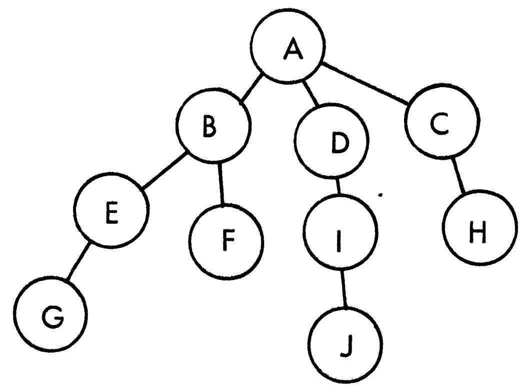
- MIXED MODE ARITHMETIC

BLOCK STRUCTURE AND STORAGE ALLOCATION



a.

BLOCK STRUCTURE
AS WRITTEN



b.

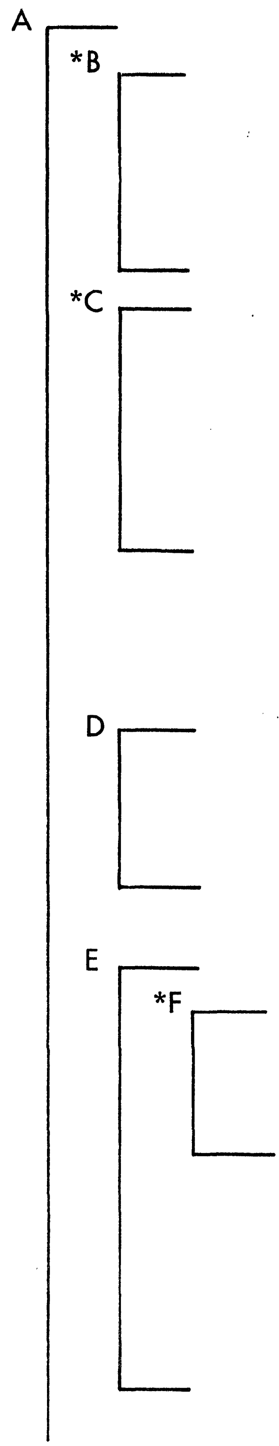
TREE FORM FOR CODE

PROGRAM STORAGE = MAX (ACH, ADIJ, ABF, ABEG)
TO BE RESERVED

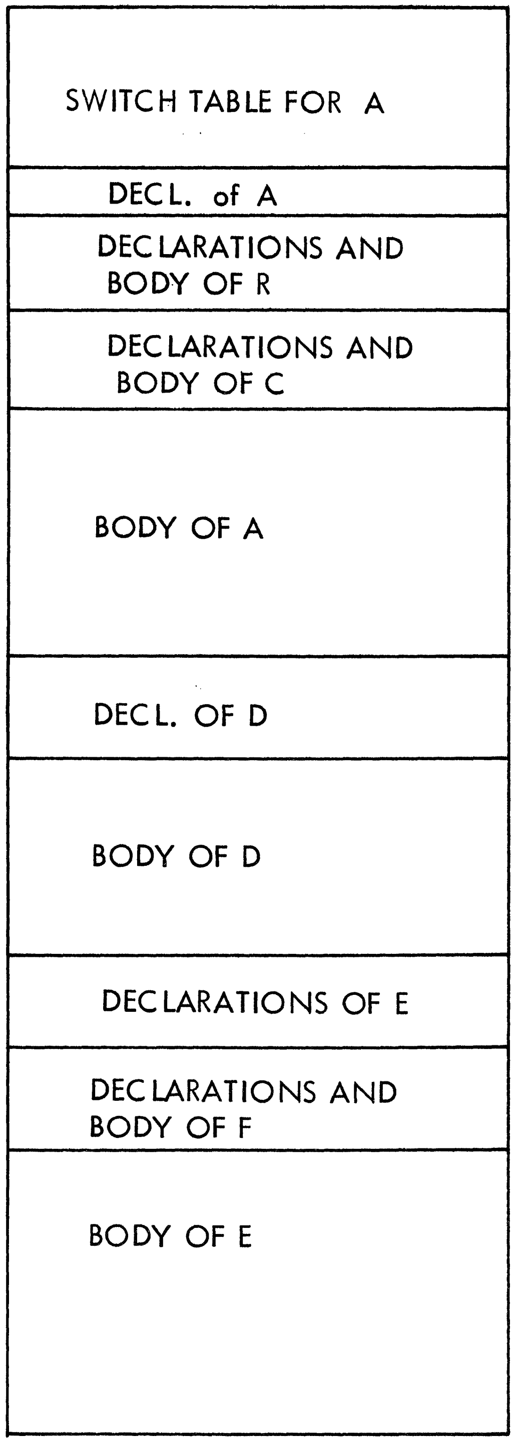
SAMPLE ALGOL PROGRAM WITH BLOCK STRUCTURE AND SUBROUTINES

```
A:  BEGIN REAL SCR, THETA; REAL ARRAY VAL, (1:29);
    INTEGER ARRAY M (1:50, 1:15), PLT (1:50, 1:15), V (1:29);
    INTEGER i, j, k, n, p, q, score, length, wd, rnk;

    PROCEDURE B (m, n, ℓ, PLT);
            VALUE m, n, e;
        BEGIN INTEGER i, s, n;
            FOR i: = 1 STEP 1 UNTIL N DO
                BEGIN FOR j : = 1 STEP 1 UNTIL 2 DO
                    BEGIN k := K   1;
                    PLT [k]: = PLT [i, j]. 10000000 END;
                    k : [k]  1 END END END B:

    PROCEDURE C (length, score, q, plt);
            VALUE length, q;
        BEGIN INTEGER t, u;
            t : = length . q;
            B(t, u, length, PLT);
            score : = PLT/u end C;

    IF (PLT [i] ≠ 0) ∧ (PLT [i] ≠ wd) then
        go to D else if PLT [i] >SCR then
        go to E else

            t : = rnk [k];
            B (t, ℓ, q, m);
            C (i, k, p, m);

D:  BEGIN REAL k;
          ℓ[i]  : = k;
        val[j]  : = k;
            i   : = j + 1;
            k   : = 1       end D;

E:  BEGIN REAL k;

    PROCEDURE F (j, k);
            value j;
            k : = j   5      end F;
            q : = n.p;
            F(q, wd);
            t : = q/lgth.     end E;

    END A AND PROGRAM;
```

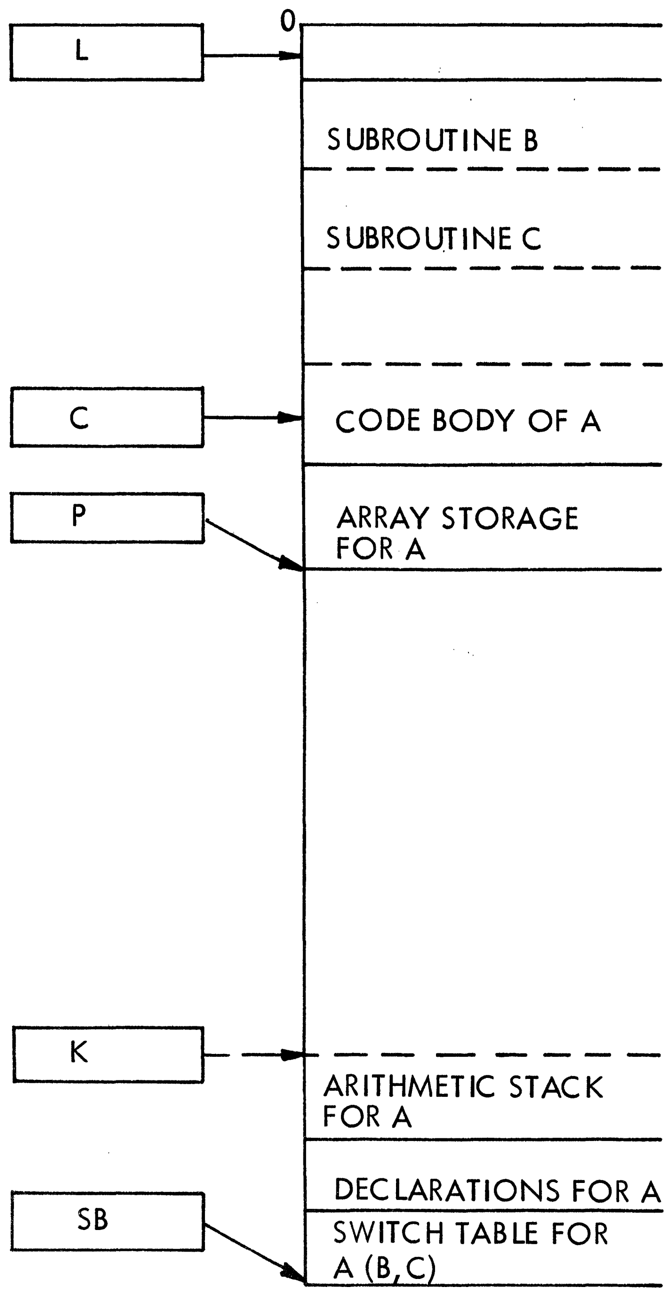
BLOCK STRUCTURE OF SAMPLE PROGRAM



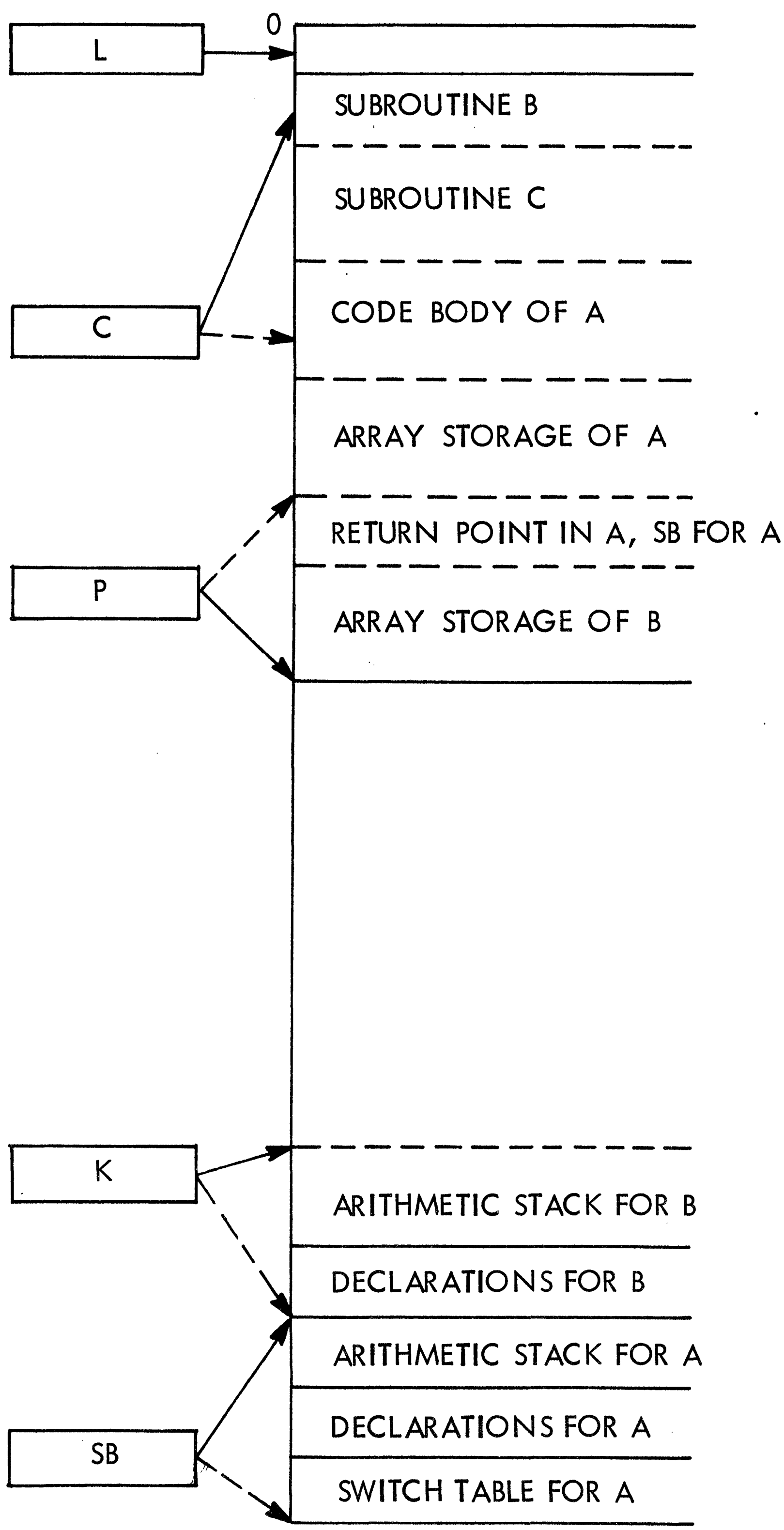
STRUCTURE OF PROGRAM A ON TAPE, REARRANGEMENT SUPPLIED BY COMPILER



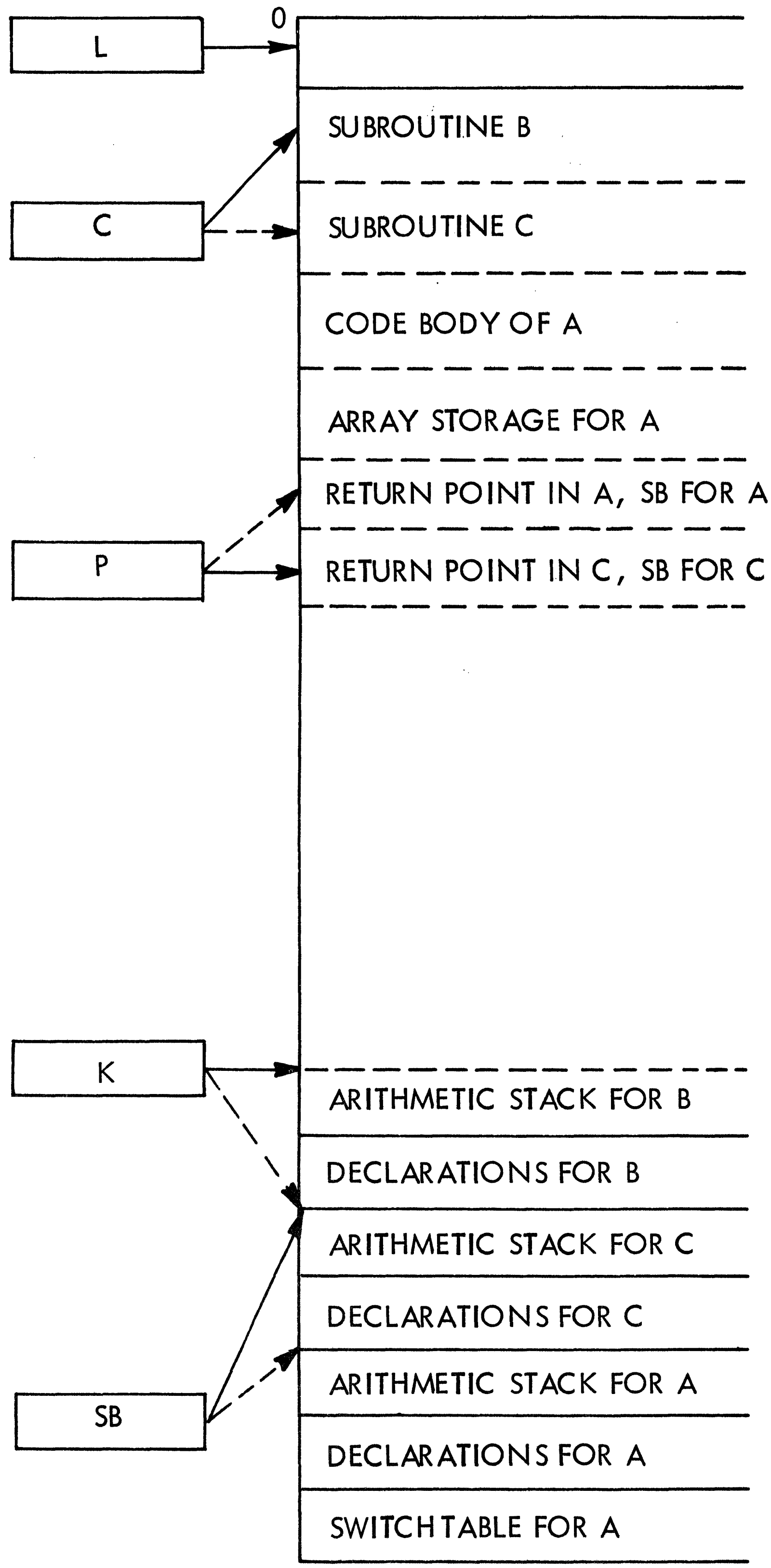
MEMORY ALLOCATION AFTER INITIAL LOADING



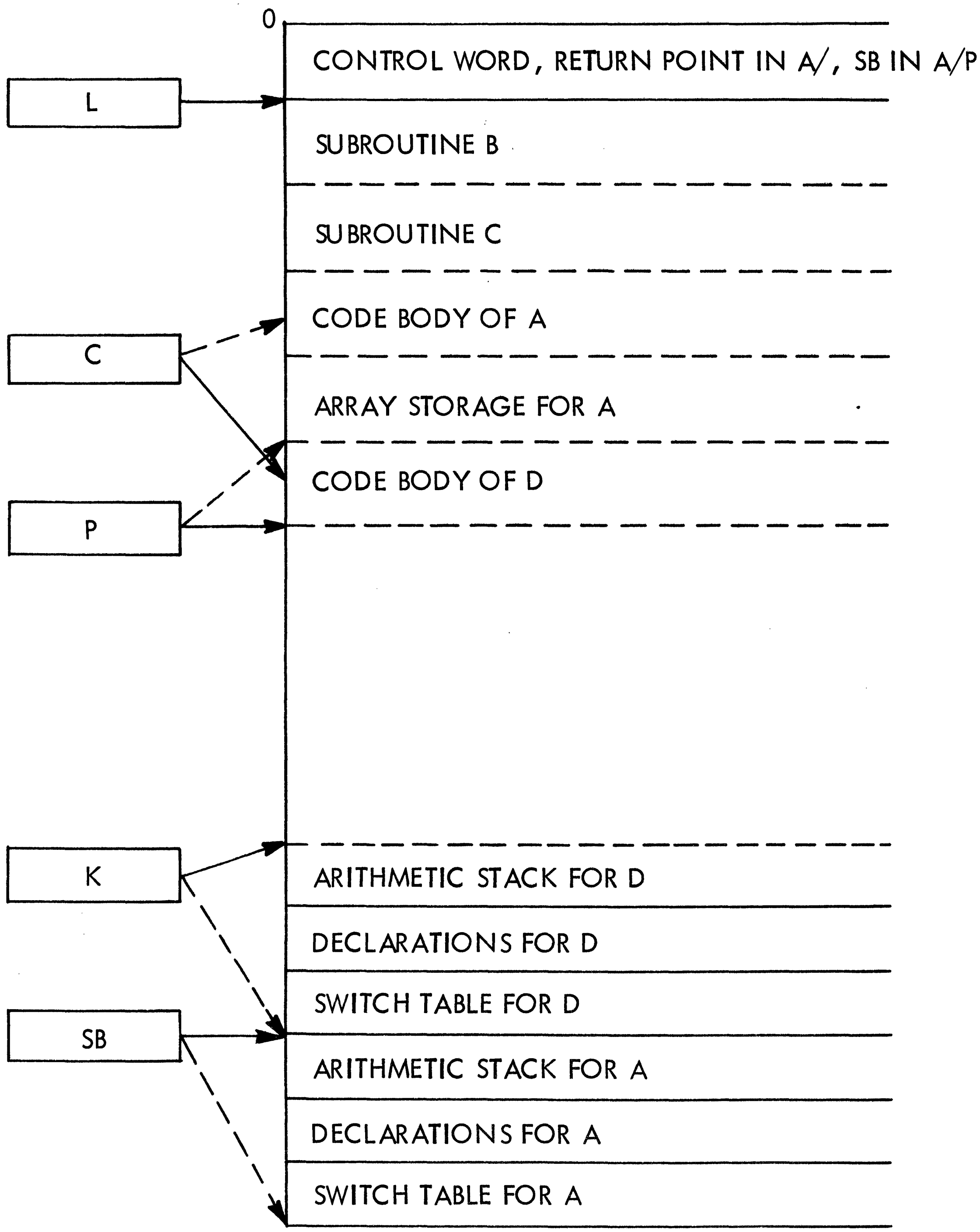
MEMORY AFTER CALL ON B



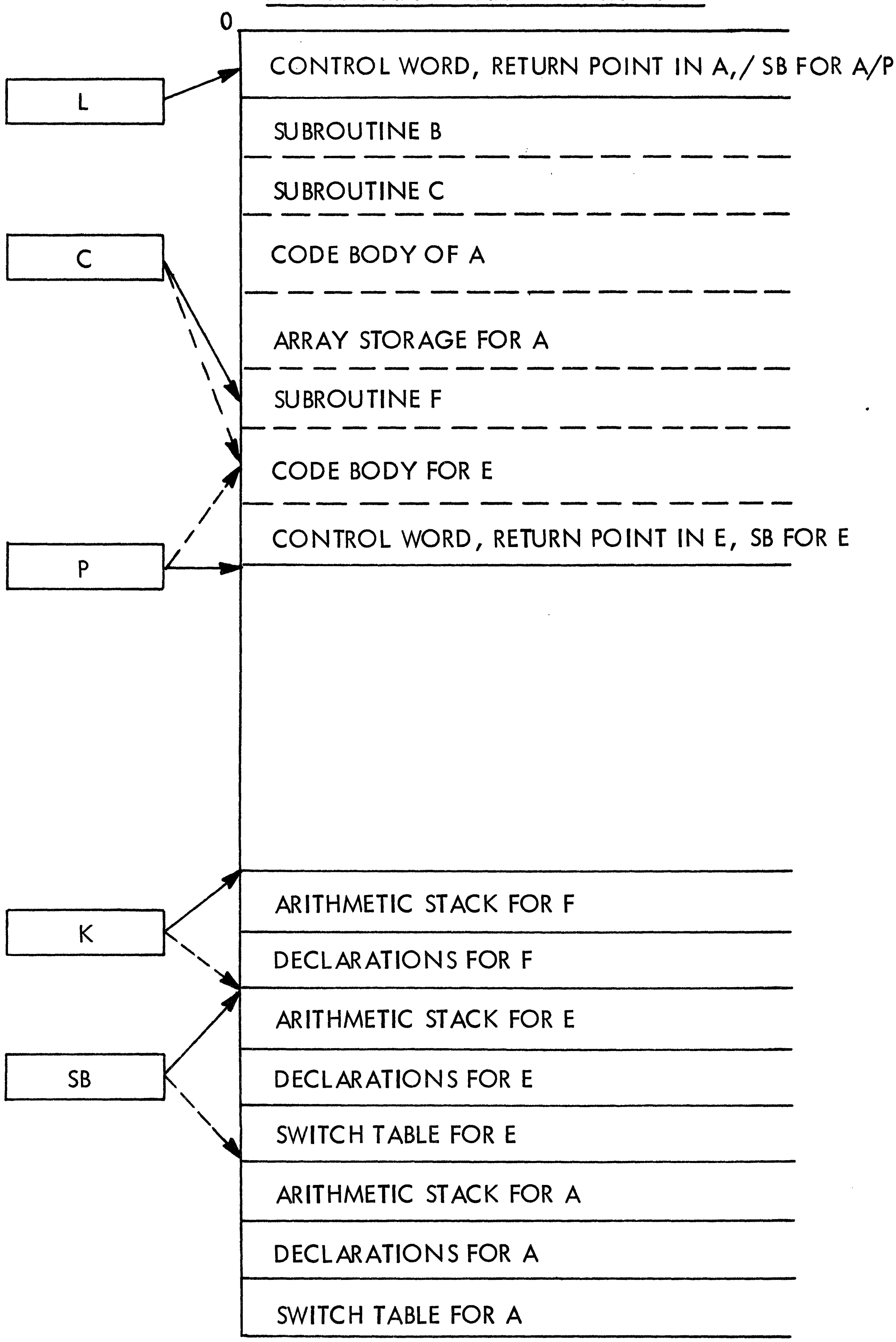
MEMORY AFTER CALL ON C



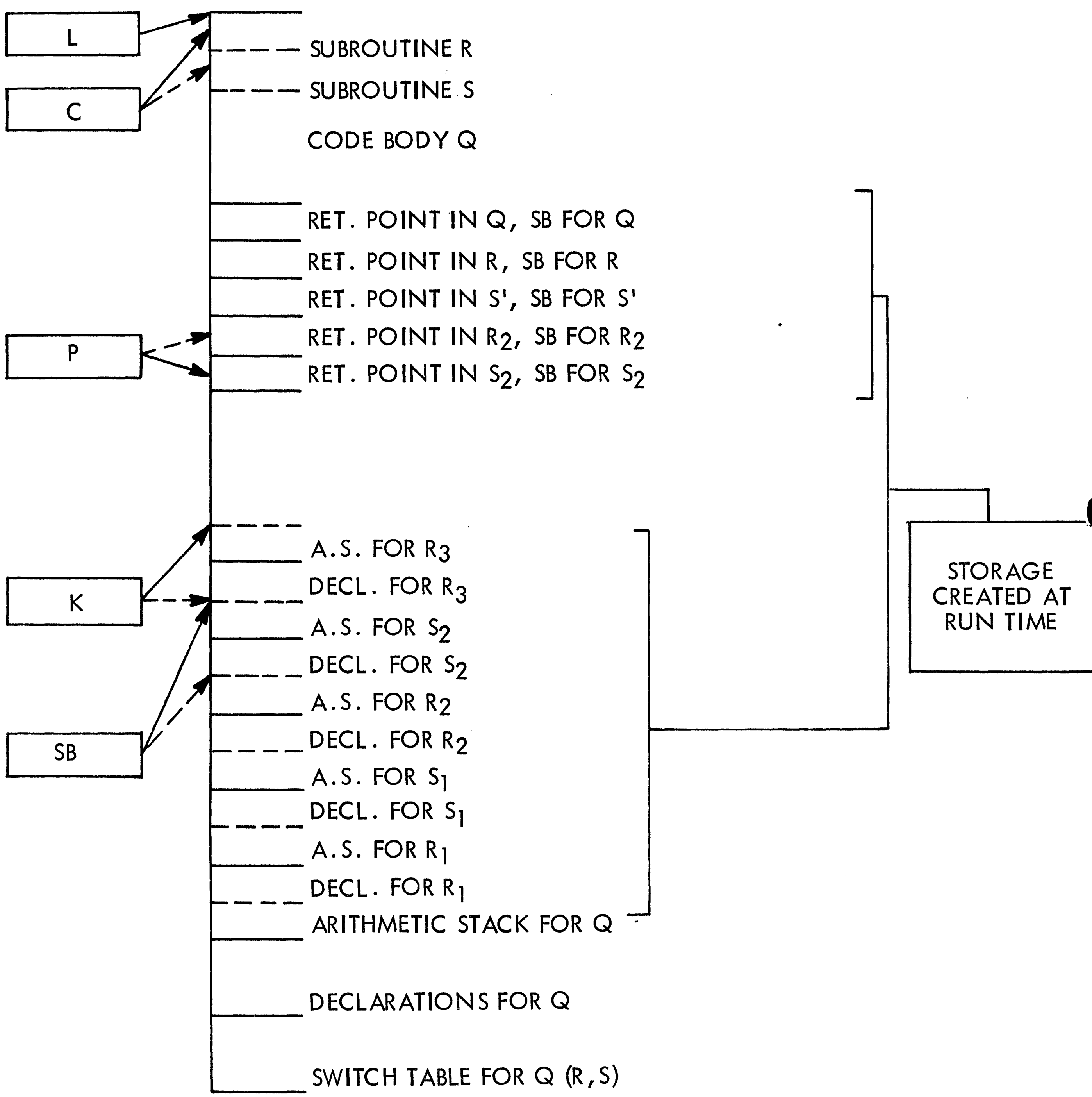
ENTRY INTO NEW BLOCK D



MEMORY AFTER CALL ON F WITHIN E



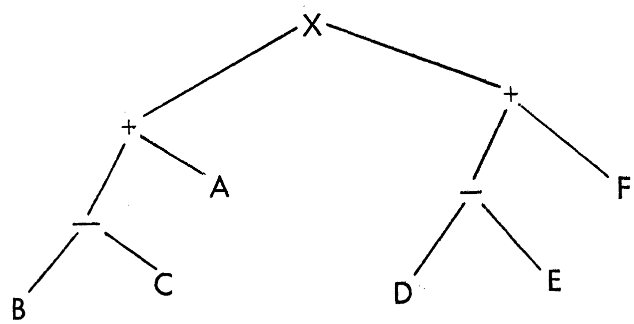
RECURSION IN AN UNRELATED PROGRAM



POLISH NOTATION AND ARITHMETIC EXPRESSIONS

$$\underbrace{\underbrace{(A + \underbrace{(B - C)}_{1})}_{2} \times \underbrace{(\underbrace{D/E}_{3} + F)}_{4}}_{5}$$

EVALUATION ORDER



TREE REPRESENTATION OF EXPRESSION

A, B, C, -, +, F, D, E, /, +, X

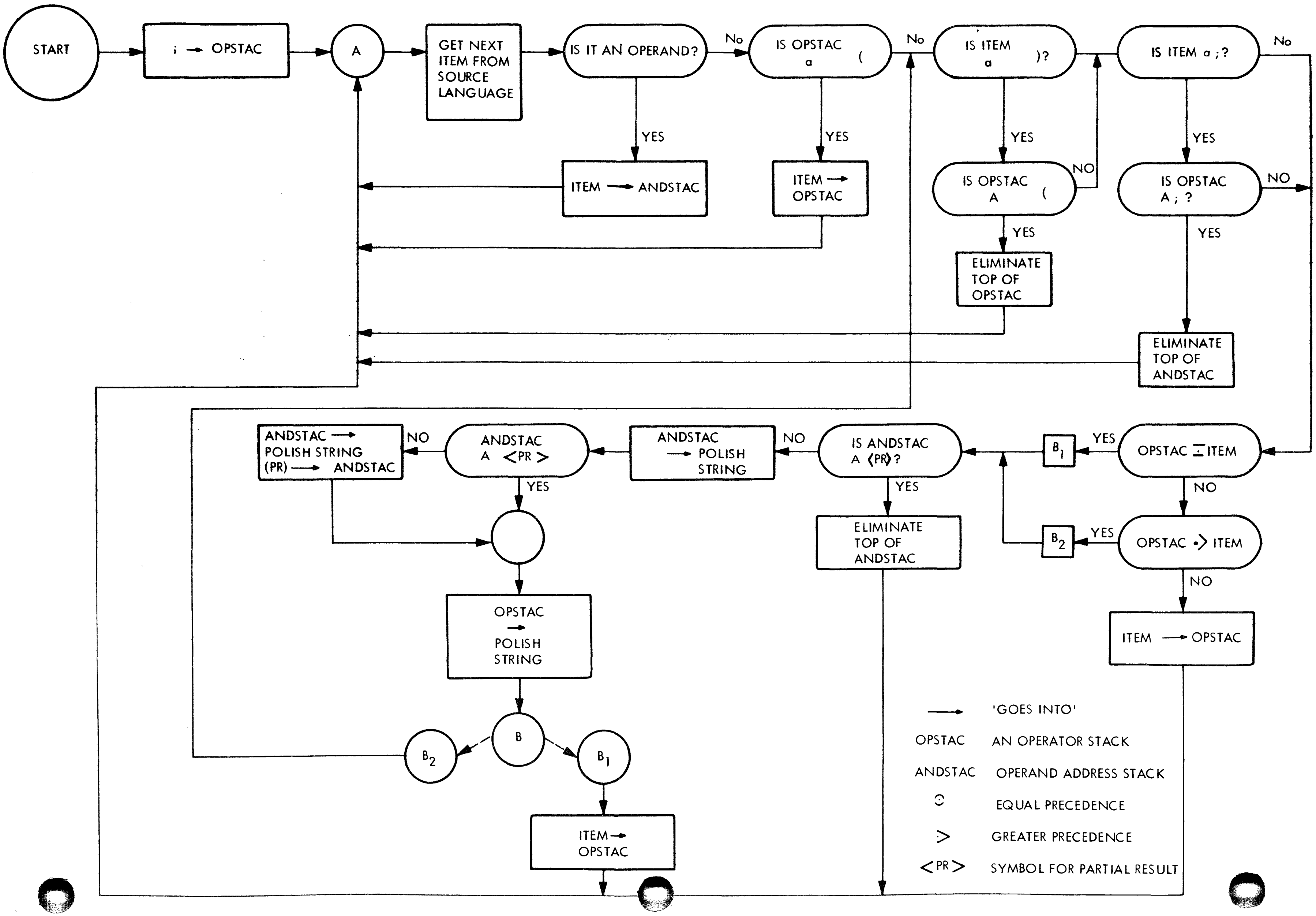
B, C, -, A, +, D, E, /, F, +, X

EQUIVALENT SUFFIX POLISH FORMS FOR EXPRESSION

X, +, -, B, C, A, +, /, D, E, F

POLISH PREFIX FORM

FLOWCHART FOR CONVERTING EXPRESSIONS TO SUFFIX POLISH FORM BASED ON OPERATOR HIERARCHY



DATA FETCH AS A SEPARATE OPERATOR

```
CLA     B
SUB     C
ADD     A
STO     TEMP
CLA     D
DIV     E
ADD     F
MUL     TEMP
```

SINGLE ADDRESS INST
PROGRAM FOR
EXPRESSION

```
FETCH B
FETCH C
SUB
FETCH A
ADD
FETCH D
FETCH E
DIV
FETCH F
ADD
MUL
```

'STACK' PROGRAM
FOR
EXPRESSION

FUNCTIONS IN STACK MACHINE

SIN (X)

MAX (a, b, c, d, e . . .)

FUNCTIONS

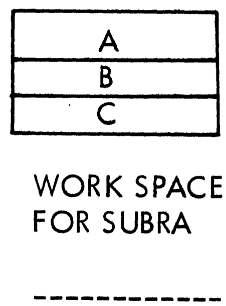
X, SIN, <SRE>

a, b, c, d, e . . . . , MAX, <SRE>

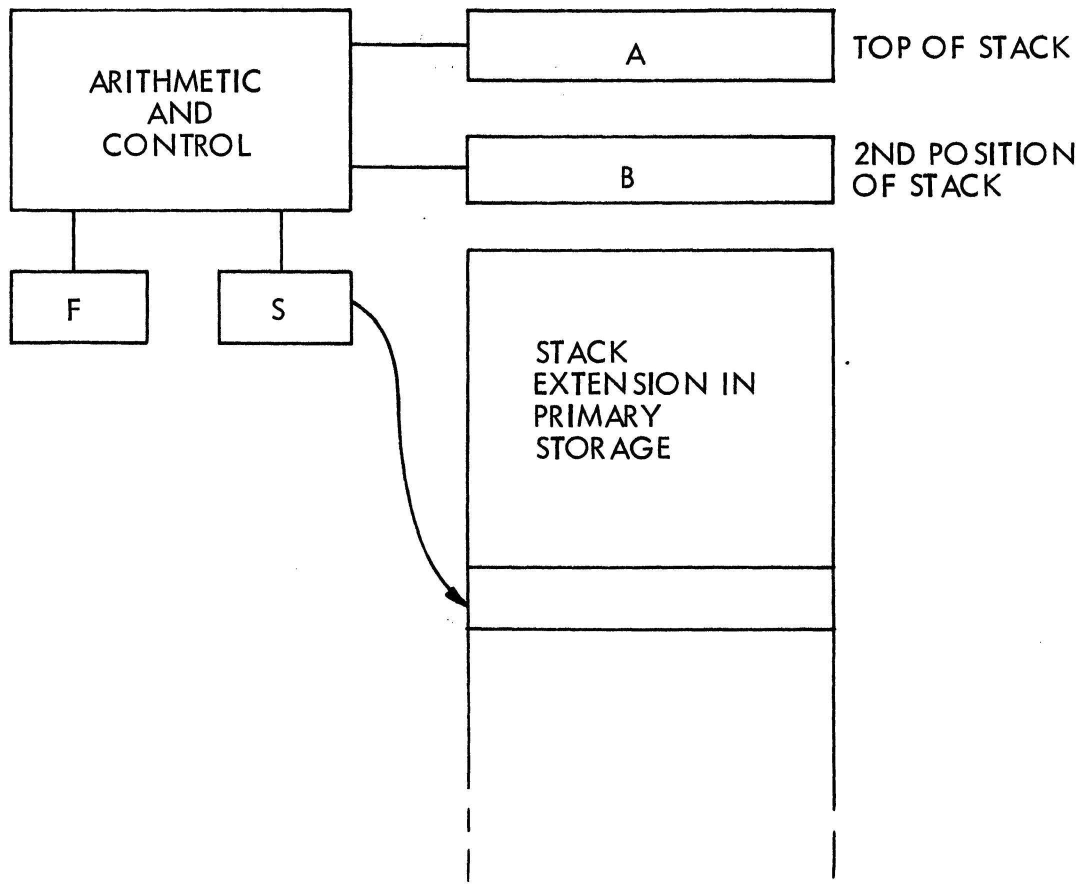
POLISH FORM

STACK AS A COMMUNICATIONS MEDIUM

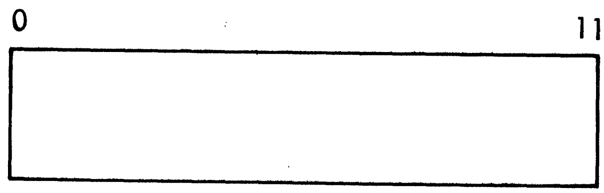
SUBRA (A, B, C)



SIMPLIFIED FUNCTIONAL DIAGRAM OF B5000 ORGANIZATION



INSTRUCTIONS IN B5000



BIT 10	BIT 11	SYLLABLE TYPE
0	0	LITERAL CALL
0	1	OPERATOR
1	0	OPERAND CALL
1	1	DESCRIPTOR CALL

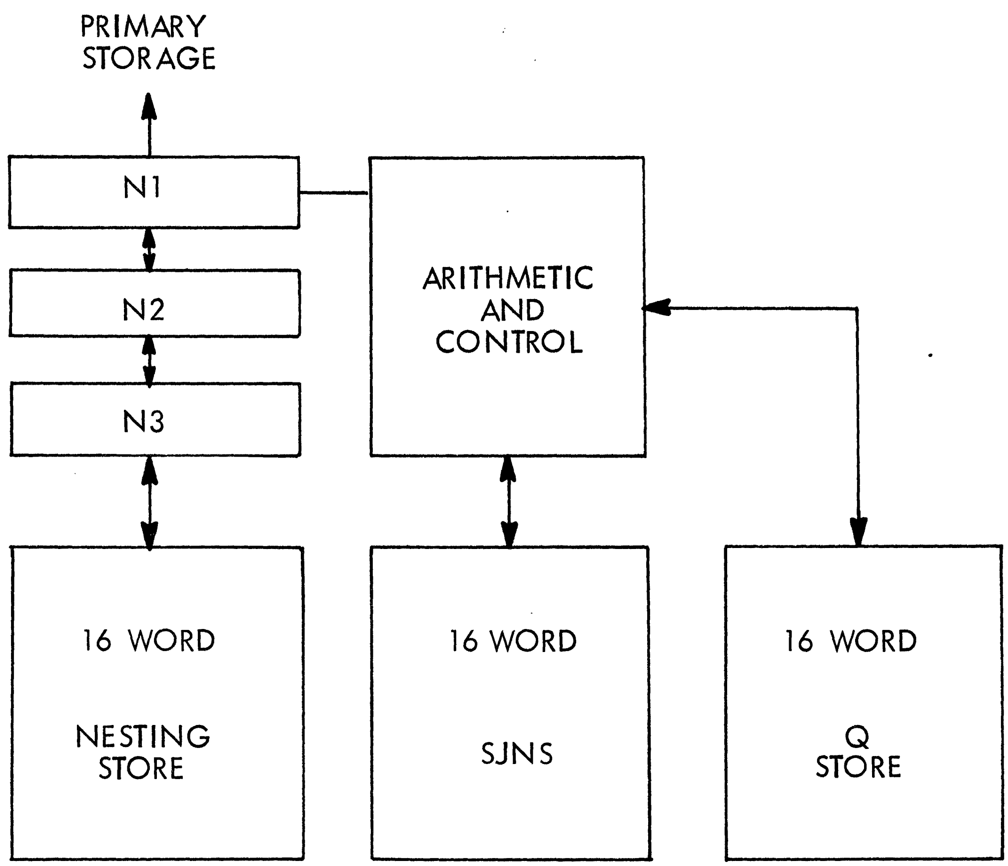
EFFECT OF OPERAND CALL SYLLABLE IN B5000

TYPE OF WORD ACCESSED	ACTION
OPERAND	PLACE IN TOP OF STACK
CONTROL WORD	PLACE IN TOP OF STACK, TREAT AS AN OPERAND
DATA DESCRIPTOR	WORD ADDRESSED BY DESCRIPTOR PLACED IN TOS, TREATED AS AN OPERAND
PROGRAM DESCRIPTOR	PLACE A RETURN CONTROL WORD IN TOS, BRANCH TO SUBROUTINE

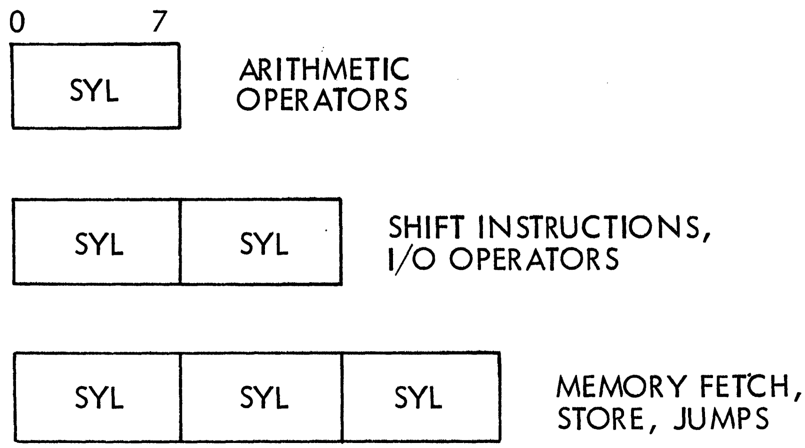
EFFECT OF DESCRIPTOR CALL SYLLABLE IN B5000

TYPE OF WORD ACCESSED	ACTION
OPERAND	GENERATE A DATA DESCRIPTOR WITH ABSOLUTE ADDRESS OF OPERAND AND PLACE IN TOS
DATA DESCRIPTOR	PLACE DATA DESCRIPTOR IN TOS
PROGRAM DESCRIPTOR	PLACE A RETURN CONTROL WORD IN TOS, BRANCH TO SUBROUTINE

THE KDF-9 COMPUTER SYSTEM



VARIABLE LENGTH INSTRUCTIONS IN KDF-9



SUMMARY OF STACK MACHINE DESIGN PRINCIPLES

- STACK CONCEPT PROVIDES 'AUTOMATIC AND ANONYMOUS' TEMPORARY STORAGE
- STACK PROVIDES DYNAMIC STORAGE ALLOCATION FOR NESTED AND RECURSIVE SUBROUTINES
- POLISH NOTATION SUGGESTS SYLLABIC INSTRUCTION FORMATS
- SEPARATE FETCH AND STORE OPERATORS PERMITS HARDWARE DETECTION AND INTERPRETATION OF CONTROL WORDS AND DESCRIPTORS
- STACK MACHINES SIMPLIFY COMPILING BECAUSE INTERNAL STRUCTURE MATCHES A 'NATURAL' INTERMEDIATE LANGUAGE, AND ELIMINATES NEED TO KEEP TRACK OF TEMPORARY STORAGE

▶ THIS SESSION WILL COVER

- OPERATING SYSTEMS OVERHEAD
- OPERATING SYSTEMS DESIGN FOR 'STACK' MACHINES
- PRECISION IN COMPUTERS

DISTRIBUTION OF FUNCTIONS IN OPERATING SYSTEMS

- FUNCTIONS REQUIRED BY USER IN EXECUTION OF THIS PROGRAM(S)
  - I/O
  - SUPERVISORY SERVICES (OBTAINING OVERLAYS, EXECUTION OF COMMON SUBROUTINES)

- FUNCTIONS TRANSPARENT TO USER
  - MEMORY ALLOCATION/DEALLOCATION
  - SCHEDULING/DISPATCHING
  - INTERRUPT SERVICING
  - SWAPPING (IF PRESENT)

▶ SOURCES OF OVERHEAD

- SPACE REQUIRED BY OPERATING SYSTEM (RESIDENT AND NON-RESIDENT)
- TIME REQUIRED TO RE-DIRECT CPU FOR INTERRUPT PROCESSING
- SWAPPING FOR CONVENIENCE OF OPERATING SYSTEM

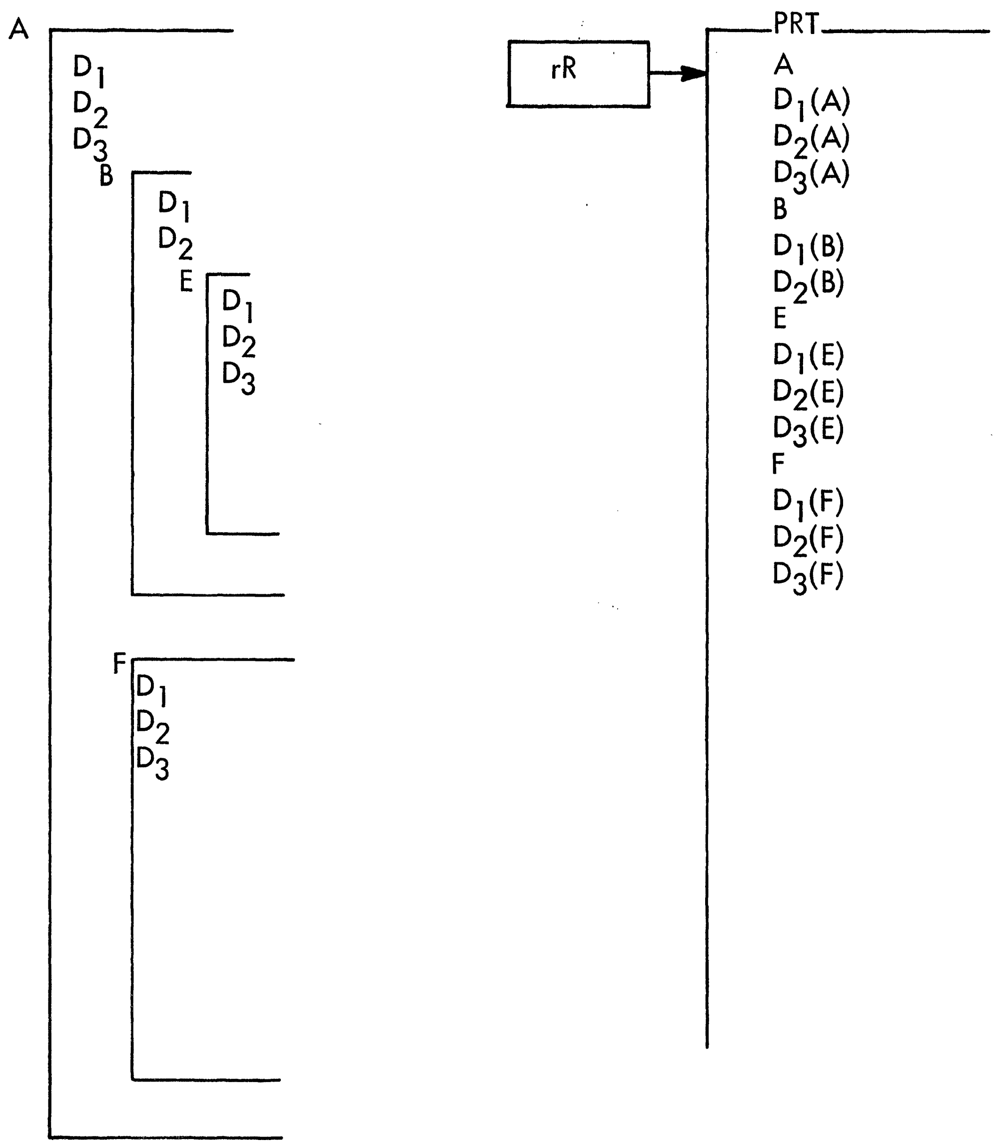
▶ METHODS FOR REDUCING OVERHEAD

- 'WIRED-IN' OPERATING SYSTEMS MICROPROGRAMMING
  DEDICATED STORAGE FOR SYSTEM TABLES
- MULTIPLE CONTROL STATES WITH SEPARATE STATE WORDS
  - RCA SPECTRA 70 SERIES
  - SDS SIGMA 7
- ASSOCIATIVE STORAGE FOR SCRATCHPAD REGISTERS
- INDEPENDENT CHANNELS
- HIGH SPEED BULK STORAGE
  - LCS
  - QUEUE DRIVEN ROTATING STORAGE

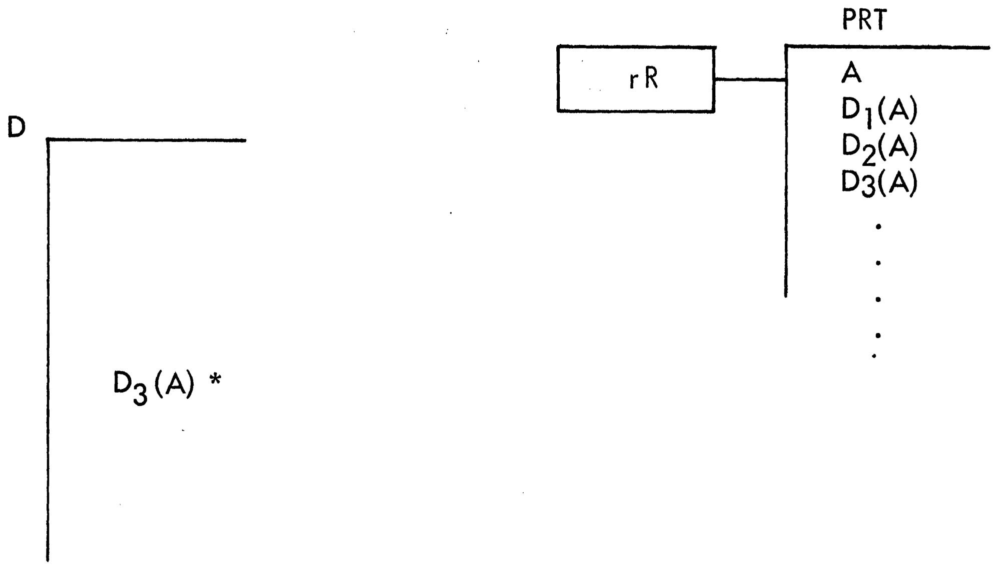
▶ OPERATING SYSTEM CONCERNS FOR STACK MACHINES

- DYNAMIC STORAGE ALLOCATION FOR BLOCK STRUCTURES
- DYNAMIC STORAGE ALLOCATION FOR STACK EXTENSION INTO PRIMARY STORAGE
    - RECURSIVE SUBROUTINES
    - DYNAMIC ARRAYS
    - ARITHMETIC STACK

B5500 PROGRAM STRUCTURE AND PRT

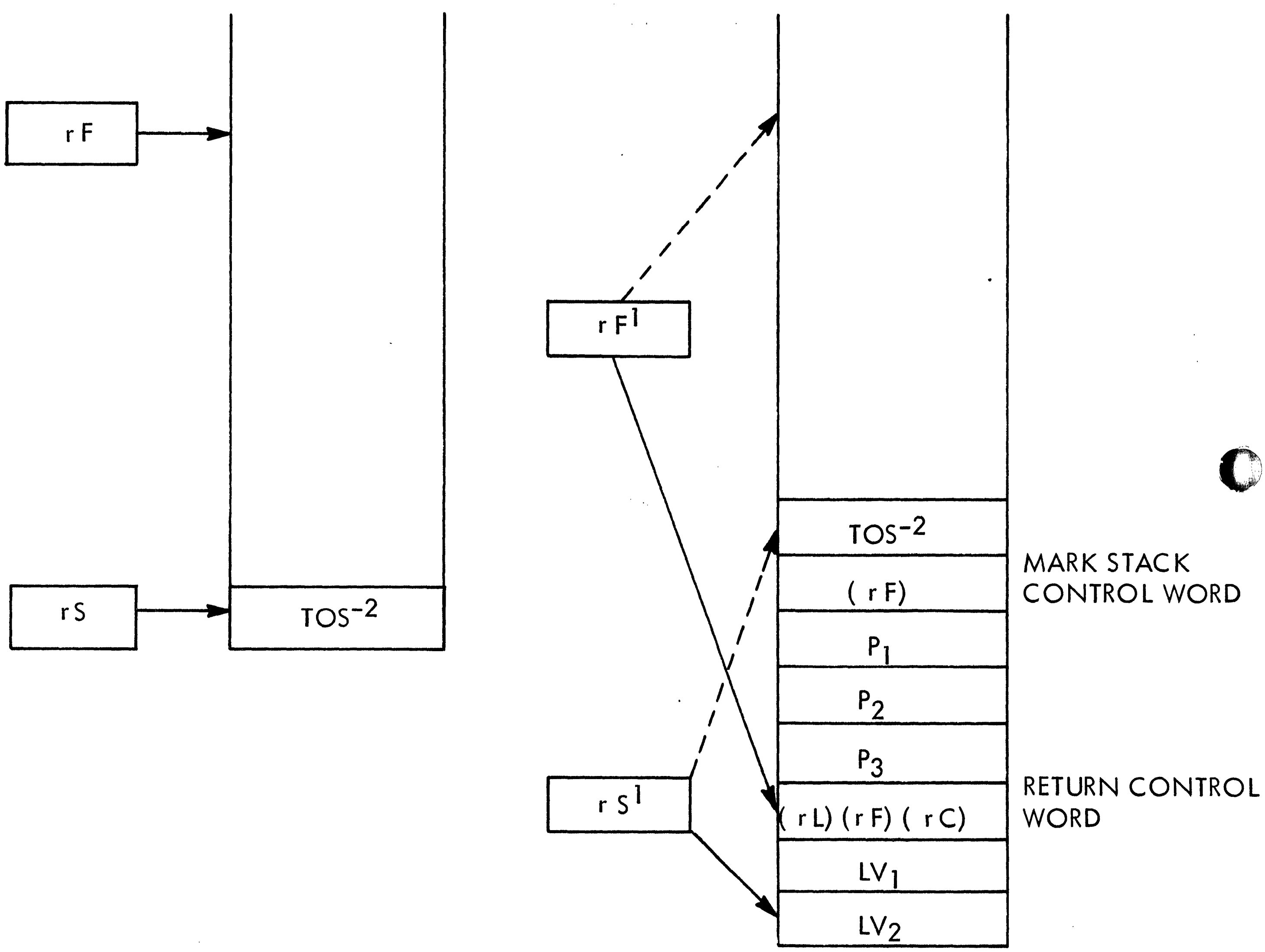


ADDRESSING HIGHER LEVEL BLOCK DATA



$(rR) + PRT \cdot$ REL ADDRESS OF A + INDEX OF D_3.

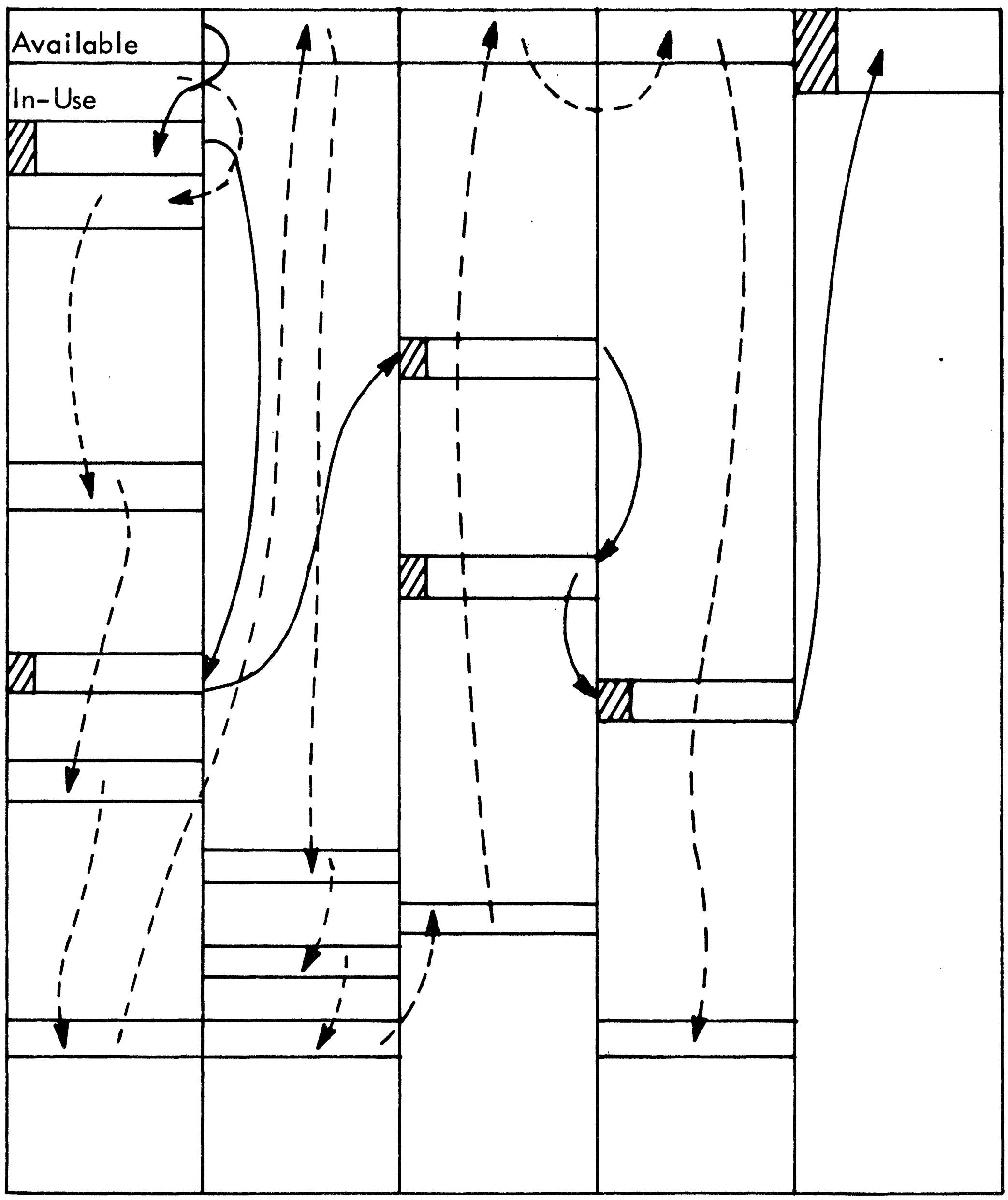
B5500 STACK STRUCTURE FOR SUBROUTINES



▶ DF MCP CLASSIFICATION OF PRIMARY STORAGE

- NON-OVERLAYABLE STORAGE
  - RESIDENT MCP
  - SYSTEM TABLES
  - PROGRAM PRT AND STACK AREAS
- OVERLAYABLE STORAGE
  - PROGRAM SEGMENTS
  - DATA AREAS. (ARRAYS)
- AVAILABLE STORAGE

DFMCP ORGANIZATION OF PRIMARY STORAGE



PRIMARY STORAGE

DF MCP PROCEDURES

- STATUS
- CONTROL CARD
- SELECTION
- RUN
- INITIATE
- PRESENCE BIT

DF MCP CONTROL PROCEDURES

- SLEEP
- NOTHINGTODO
- GETSPACE
- OLAY
- FORGETSPACE
- ESPBIT

B5500 PARALLEL PROCESSING AND CHECKPOINT FACILITIES

- PARALLEL PROCESSING AND PRIORITY INTERRUPTS
- BREAKOUT, RESTART, EMERGENCY INTERRUPT

CHARACTERISTICS OF B5500 OP. SYSTEM.

- MULTIPROGRAMMING DESIGNED IN AT THE START
- PROVIDES MULTIPROCESSING CONTROL
- PROVIDES DYNAMIC STORAGE ALLOCATION FOR
  - PROGRAM SEGMENTS
  - DATA (ARRAYS)
- STACK MECHANISM HANDLES
  - RECURSIVE SUBROUTINES
  - ARITHMETIC STACK
- SUPPORTS ON-LINE USE
  - THE INTERP SYSTEM
  - DATACOMM SYSTEM

PRECISION COMPARISONS

SYSTEM	INTEGERS	SINGLE FLOATING		DOUBLE FLOATING			
		CHAR.	MANT.	CHAR.	MANT.	S/H	
B5500	48	6+S	39+S				
B8500	48	10+S	35+S				
CDC 36/3800	48	11	36	11	84	H	
CDC 6600	60	11+S	48	11+S	96	S	(SOFTWARE)
GE 625/35/45	36,72	7+S	27 S	7+S	63+S	H	
IBM 360	16,32	7	24	7	56+S	H	
IBM 360/44	16,32	7	24	7	24,32,40,48,56	H	
RCA SPECTRA 70	16,32	7	24	7	56+S	H	
SDS SIGMA 7	32			7	56+S	H	
UNIVAC 494	15,30			11	48+S	H	
UNIVAC 1108	36	8	27 S	11	60+S	H	(+, - ONLY)

EARLY DEVELOPMENTS

- MILITARY INFLUENCE
  - SAGE
  - L-SYSTEMS
- SHARED-DEVICE SYSTEMS
  - ASP/HASP
  - ON-LINE 1401
- MIT/CTSS
- DARTMOUTH BASIC
- IBM QUIKTRAN

BASIC ELEMENTS OF TIME-SHARING

- ON-LINE UTILIZATION
- TERMINAL INTERFACE
- ILLUSORY USE OF VIRTUAL MACHINE
- HUMAN VS. MACHINE RESPONSE TIME

TYPES OF MULTIPROGRAMMING SYSTEMS

- SPECIAL PURPOSE
  - DEDICATED MACHINE
  - FIXED PROGRAM STRUCTURE
  - HIGHLY VARIABLE DATA LOADS
  - EXAMPLES:

    AIRLINE RESERVATIONS

    THEATER TICKET

    BROKERAGE
- LIMITED PURPOSE
  - DESIGNED FOR ONE LANGUAGE
  - BASIC
  - QUIKTRAN
- GENERAL PURPOSE
  - PURE MULTIPLE BATCH
  - PURE ON-LINE
  - MIXED BACKGROUND/FOREGROUND

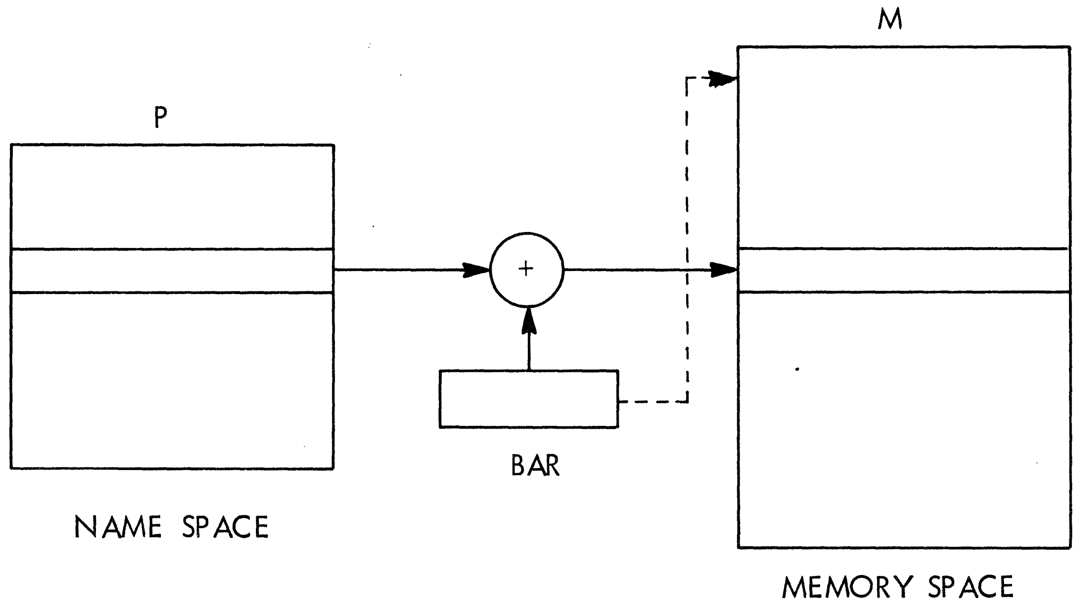
OPERATING SYSTEM PRINCIPLES

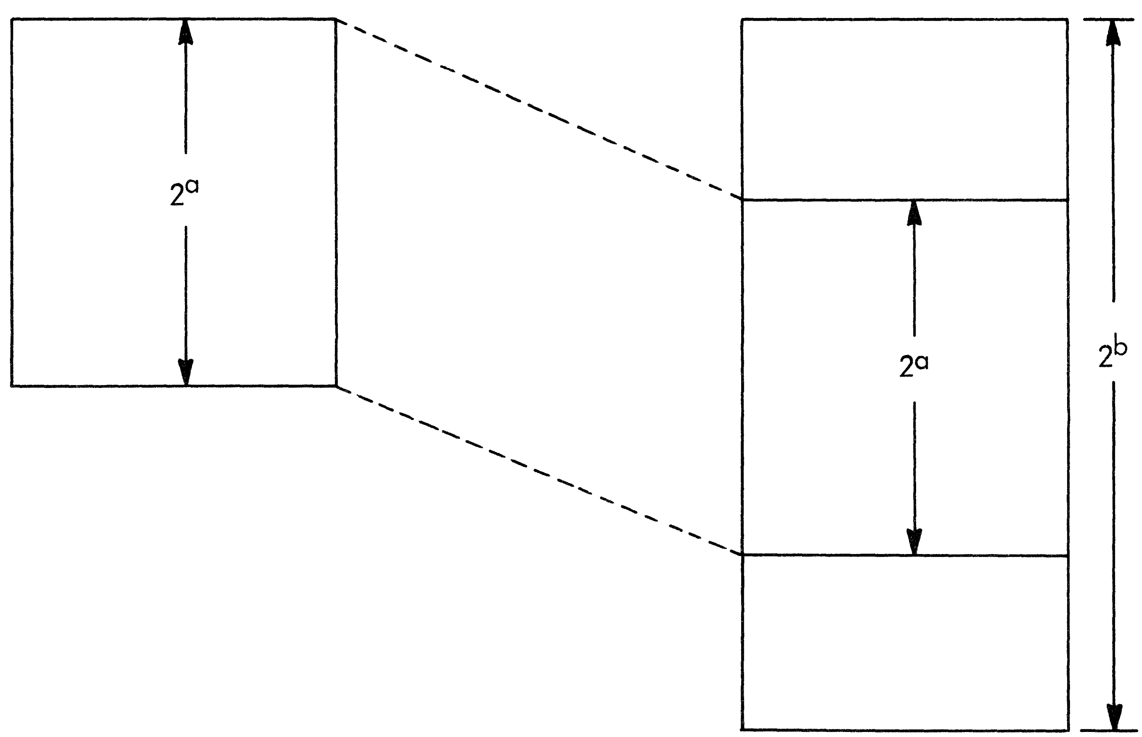
- MUST ACCOMMODATE MULTIPROCESSING
- HANDLES MANY USERS
- HANDLES VARIED USER NEEDS
- COMPUTER UTILITY
- ALLOCATION OF ALL FACILITIES
- DEVICE-INDEPENDENCE
- SCHEDULING
- SWAPPING
- RESPONSIVENESS AND RELIABILITY

OTHER CONSIDERATIONS

- NEED FOR ON-LINE LANGUAGES
- CONVERSATIONAL/NONCONVERSATIONAL
- MIXED-MODE OPERATIONS

MAPPING OF 2^a ONTO 2^b, $(b > a)$.

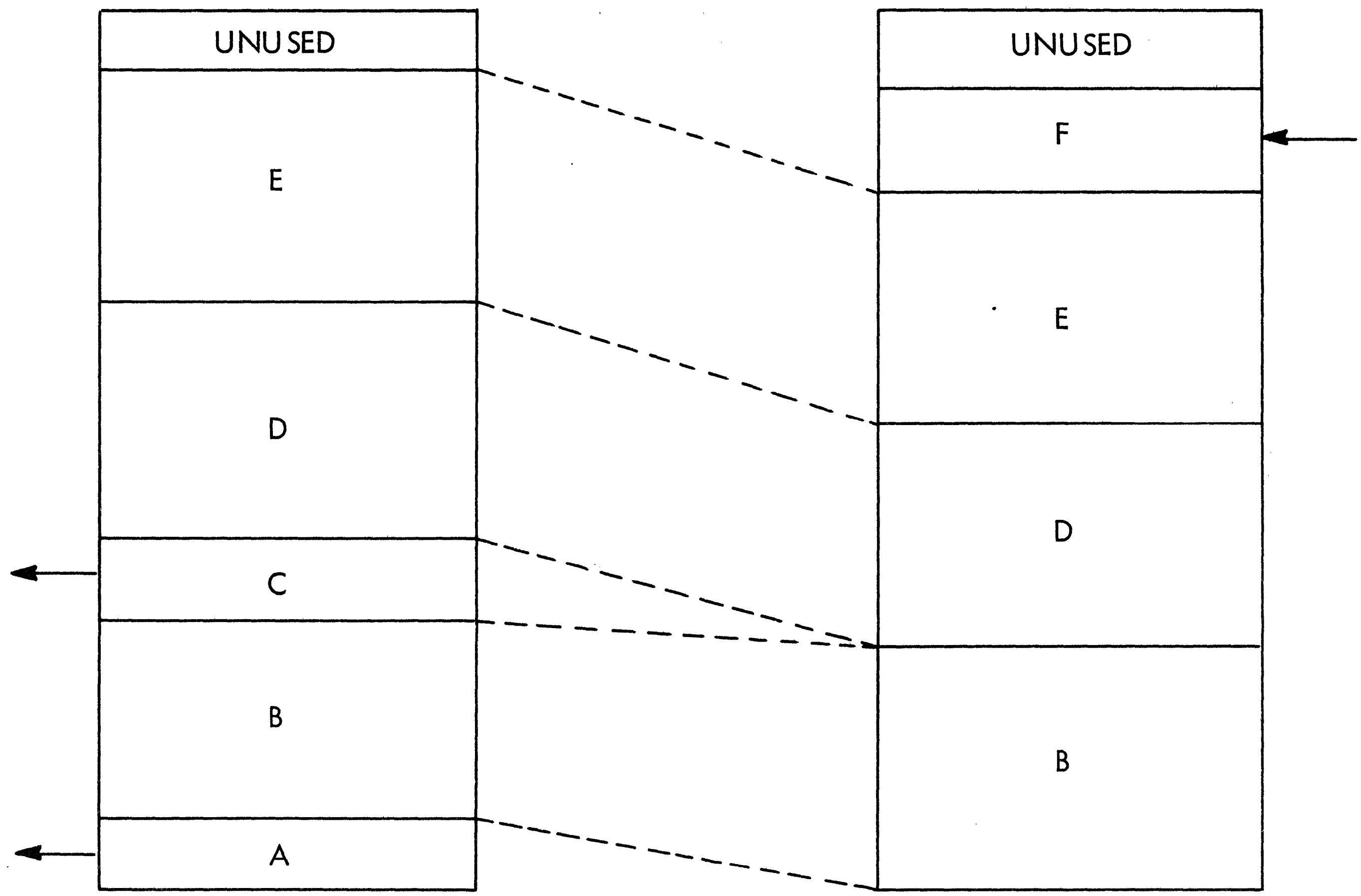




REASONS FOR COMPACTING

- FREE LARGEST BLOCKS OF CONTIGUOUS CORE
- ALLOWS FLEXIBILITY IN CHOOSING NEXT USER
- PROVIDES CORE REQUEST/RELEASE
- TO PROVIDE A MEMORY SPACE 2^b WHICH ACCOMMODATES THE NAME SPACE 2^a $(b > a)$.
- CONTROL SWITCHED BY RE-SETTING BAR.

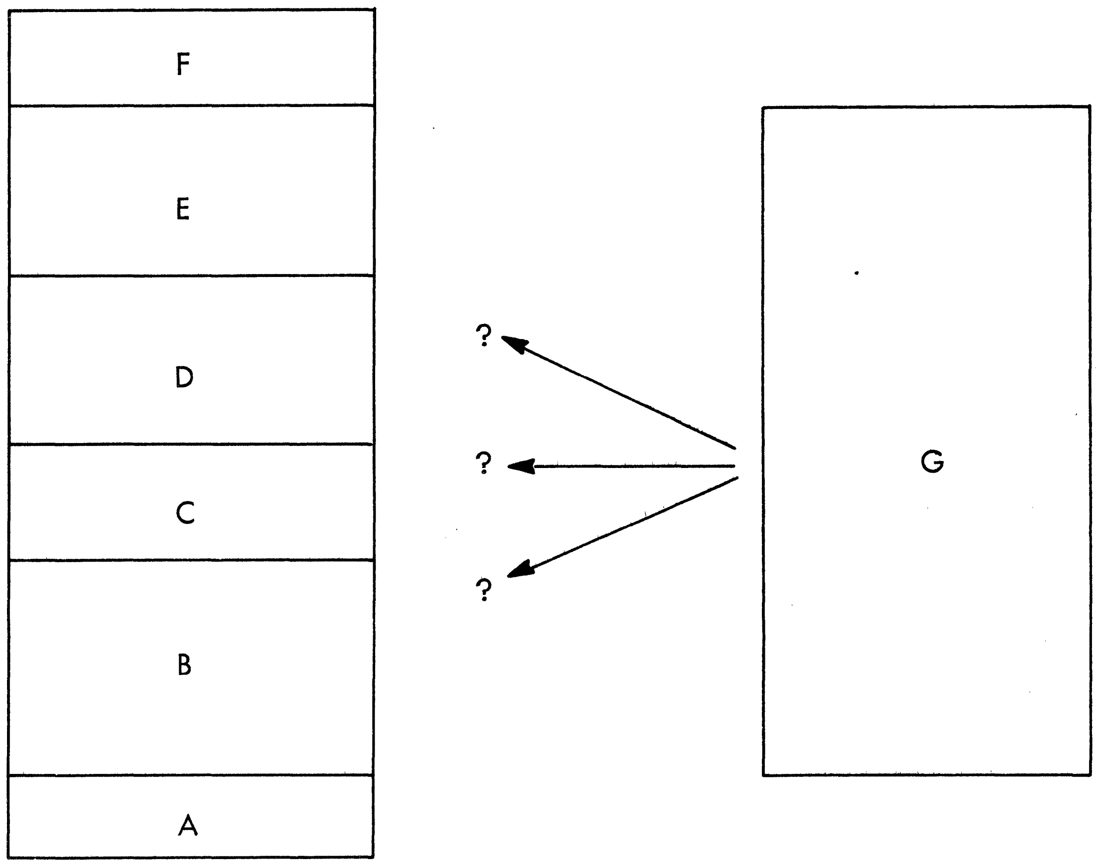
COMPACTING FOR MEMORY RE-ALLOCATION



REQUIREMENTS FOR COMPACTING

- ALL PROGRAMS PRE–BOUND
- ALL PROGRAMS SELF–RELATIVE
- BASE ADDRESS REGISTER USED
- NO MOVING DURING I/O OPERATIONS
- ALL QUEUES MUST BE DRAINED
- ALL BUFFERING MUST RE–START
- NO SHARED REFERENCES

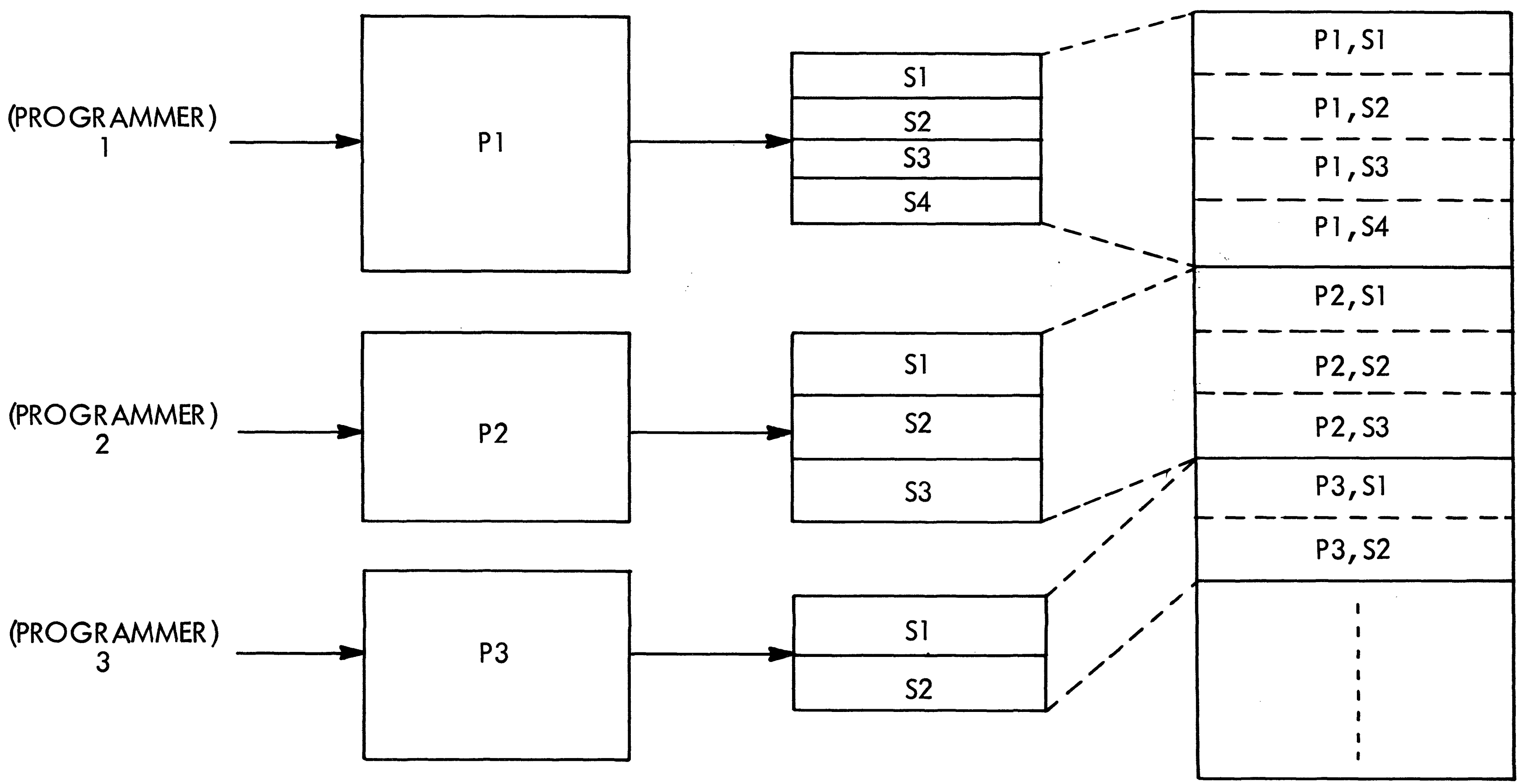
ALLOCATION PROBLEM: NAME SPACE OF G > MEMORY SPACE



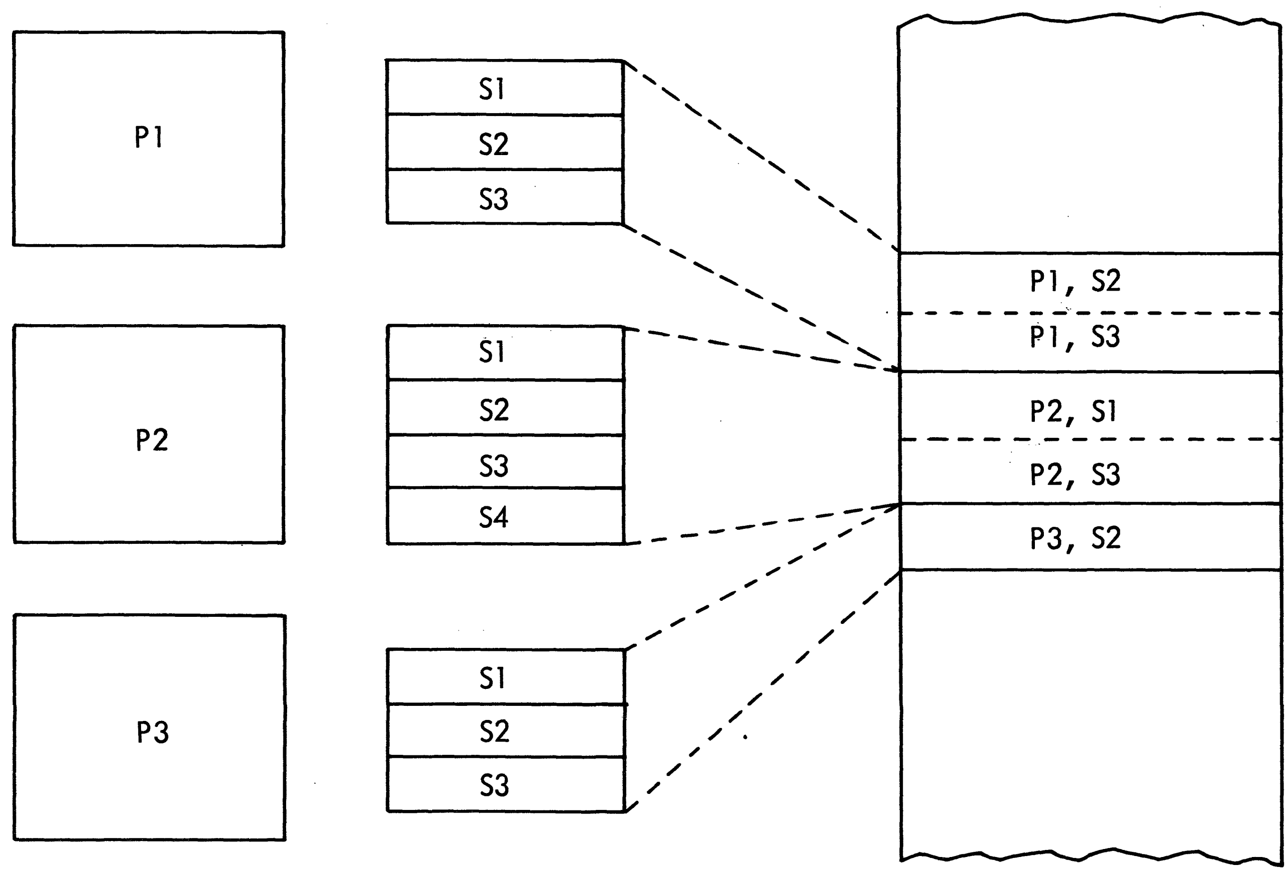
RESOURCE-INDEPENDENCE

- PROGRAMMER CONTROLS TIME SEQUENCE
- SYSTEM CONTROLS RESOURCE ALLOCATION
- PROGRAMMER REFERENCES NAMES
- SYSTEM TRANSFORMS NAMES TO DEVICES
- PROGRAMMER USES VIRTUAL LANGUAGE
- SYSTEM INTERPRETS VIRTUAL LANGUAGE
- PROGRAMMER SEES VIRTUAL PROCESSOR, VIRTUAL MEMORY, VIRTUAL REGISTERS
- SYSTEM ALLOCATES PHYSICAL RESOURCES TO MATCH VIRTUAL RESOURCES

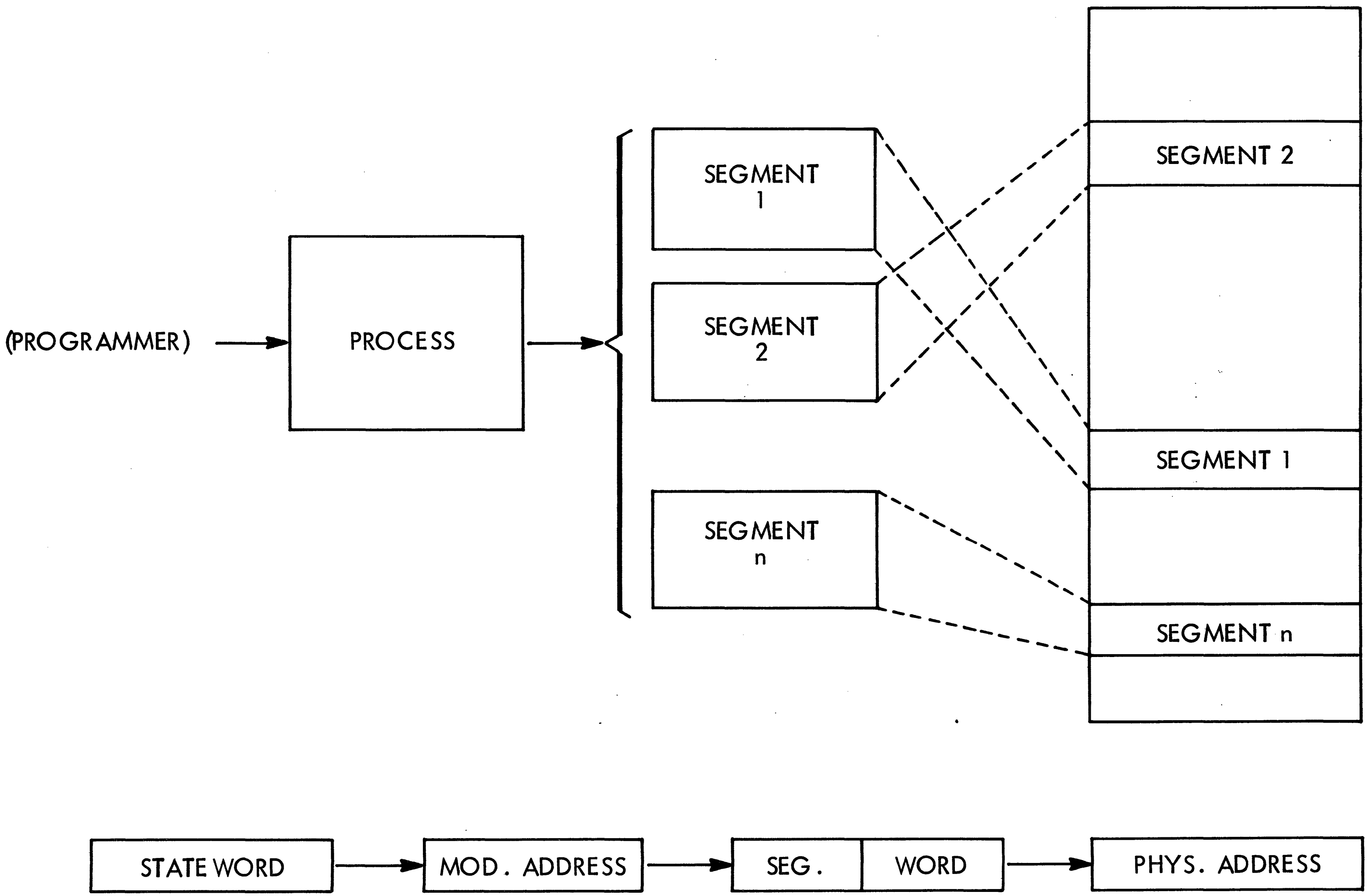
SEGMENTATION WITH FIXED BLOCKS



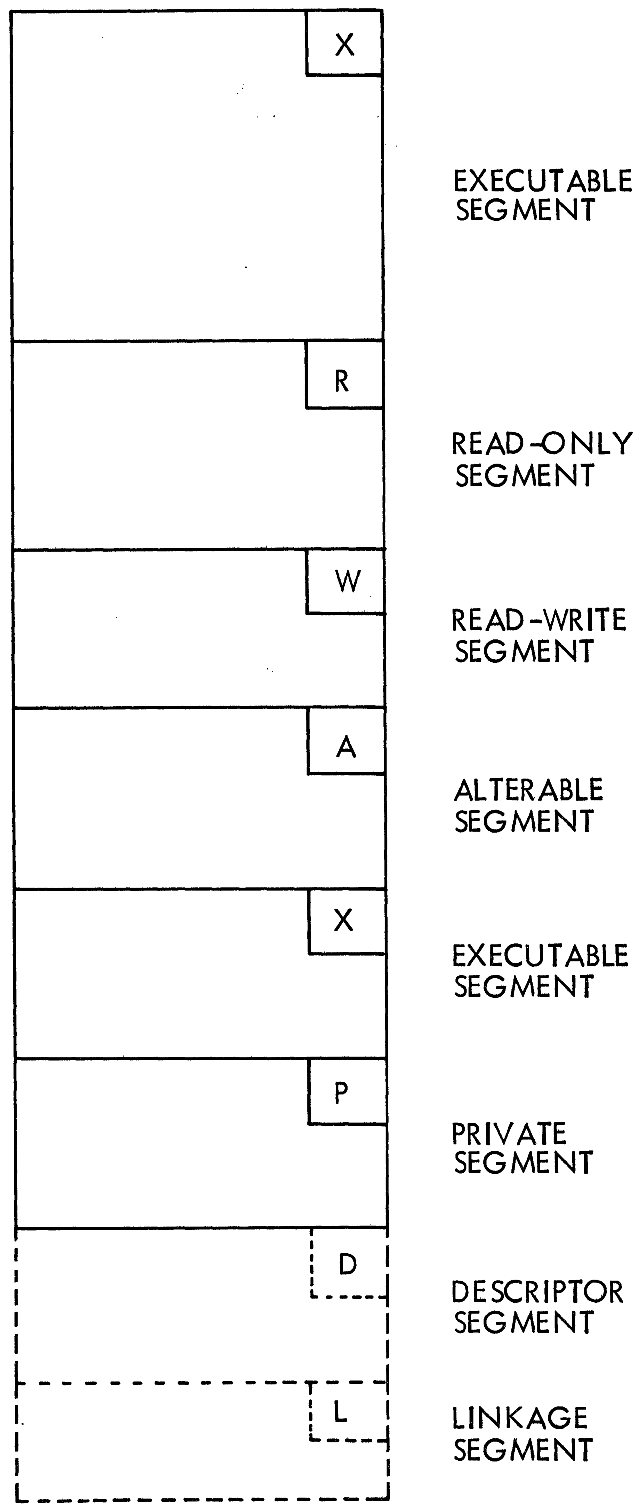
ALLOCATION OF ACTIVE SEGMENTS



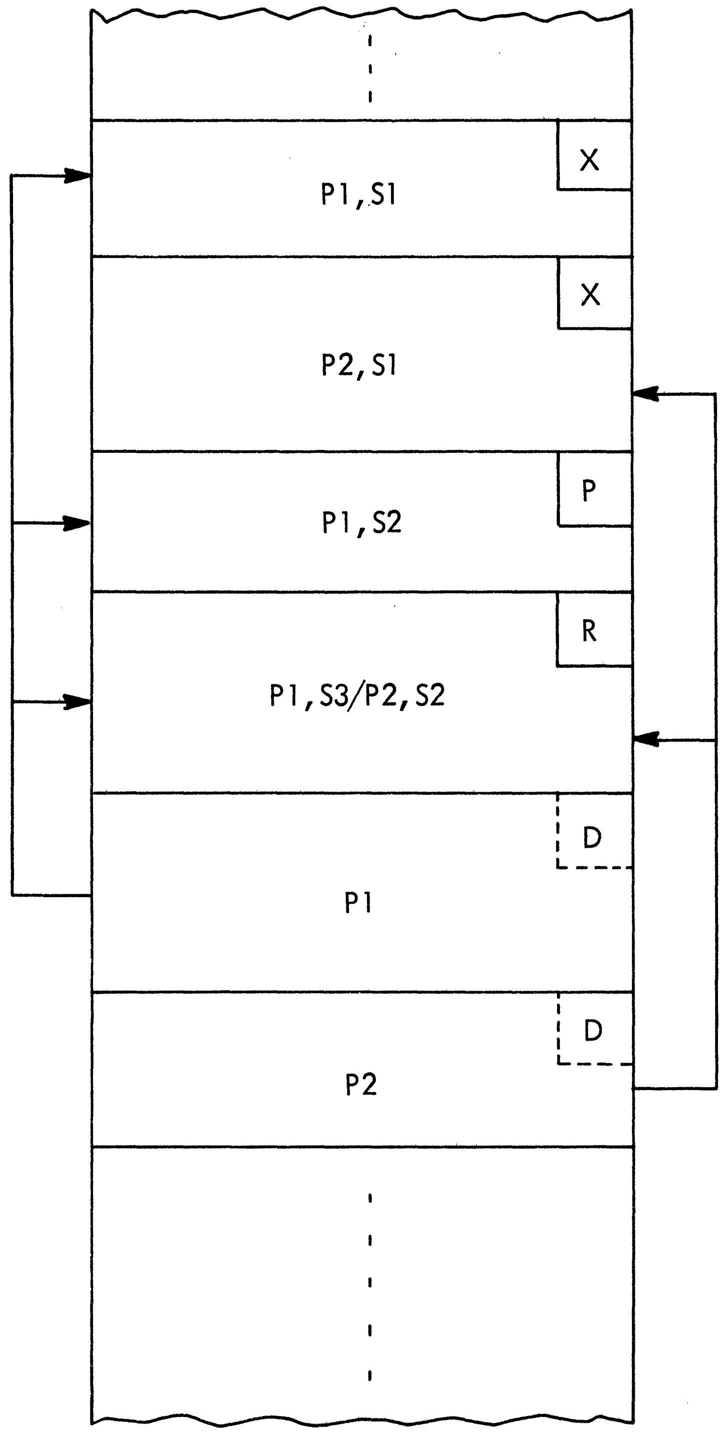
SEGMENTATION INTO NON-CONTIGUOUS MEMORY



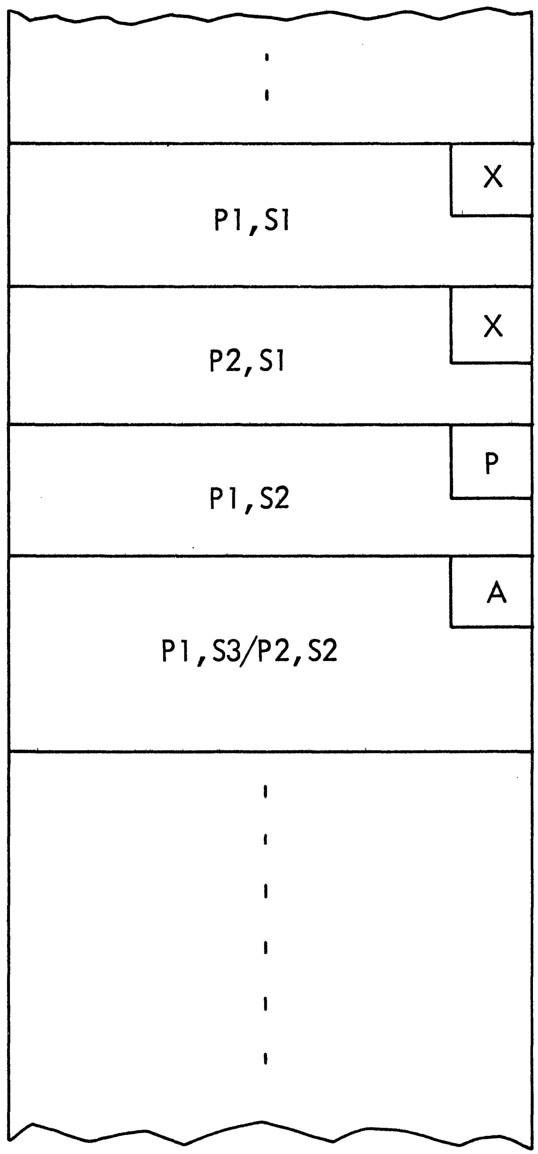
THE SEGMENTS OF A PROCESS



TWO PROCESSES WITH A SHARED SEGMENT



USE OF SHARED SEGMENT FOR SYSOUT



PAGING

- PROVIDES ADDITIONAL LEVEL OF CORE USAGE
- IMPLEMENTED BY HARDWARE
- REQUIRES SUBSTANTIAL SOFTWARE INTEGRATION
- IMPORTANT FOR ADVANCED SYSTEMS

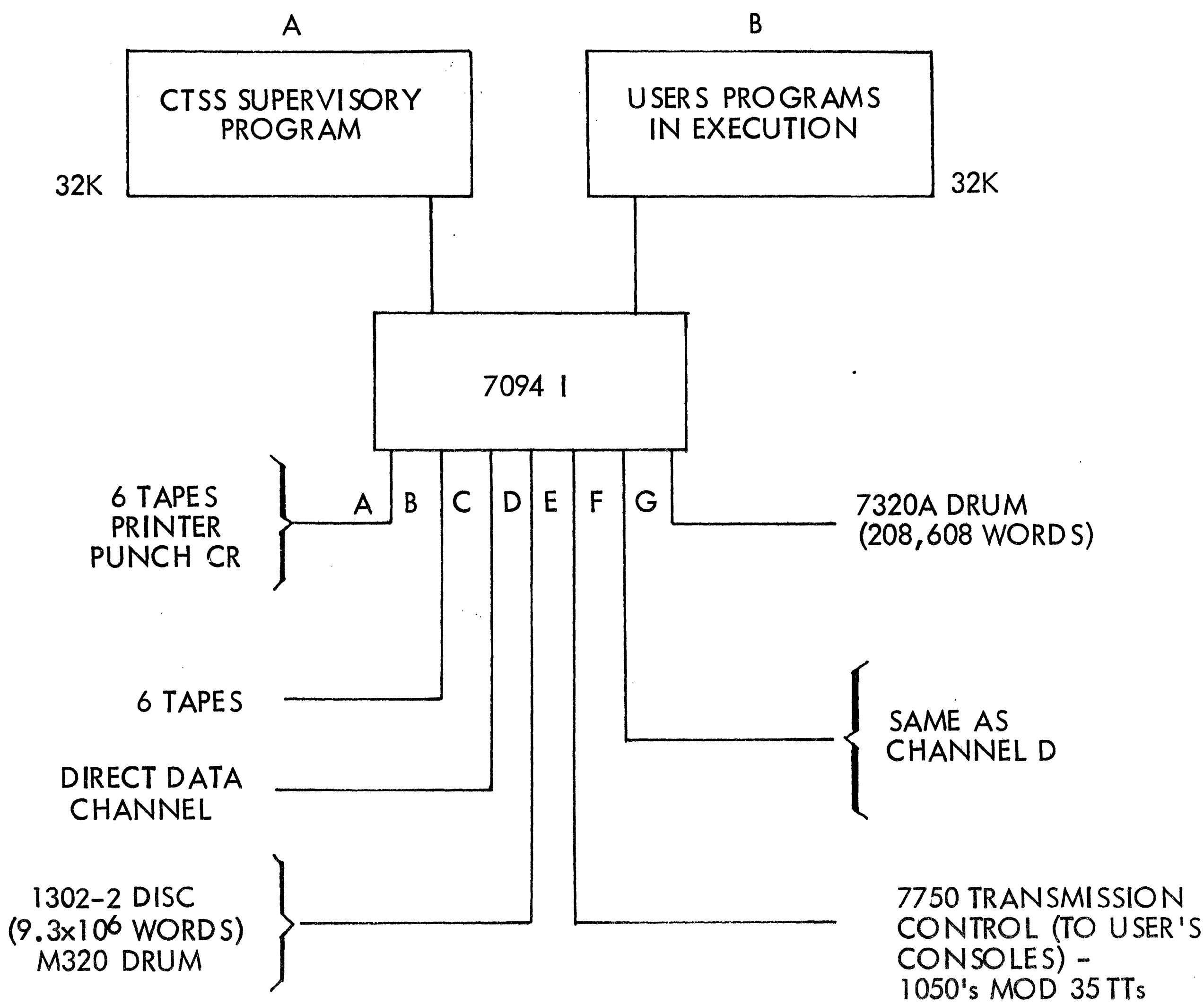
▶ TOPICS TO BE COVERED THIS SESSION

- REVIEW OF MULTIPROGRAMMED/MULTIPROCESSOR CONTROL PHILOSOPHY
- ORIGINS OF 'TIME-SHARING'
- HARDWARE/SYSTEMS DEVELOPMENTS FOR TIME-SHARING
- GE 645
- 360/67
- OTHER 'TIME-SHARING' SYSTEMS

▶ TIME -SHARING CHARACTERISTICS

- TIME-SHARING IS AN OUTGROWTH OF MULTIPROGRAMMING
- TERM ASSOCIATED WITH 'INTERACTIVE' OR 'ON-LINE' COMPUTING WHERE USERS PRESENCE (OR INTERVENTION) IS REQUIRED FOR SUCCESSFUL OPERATION OF A PROGRAM
- LACK OF ON-LINE COMPONENT YIELDS SIMPLE MULTI-PROGRAMMING
- ON-LINE COMPONENT PERMITS SYSTEM TO SERVE MANY MORE ON-LINE USERS BECAUSE OF USER INTRODUCED DELAY (SO-CALLED 'THINK' TIME)
- NEEDS MECHANISM FOR MAKING PHYSICAL SPACE AVAILABLE TO USERS -- SWAPPING

CTSS SYSTEM - FUNCTIONAL DESCRIPTION



▶ PERTAINENT EXPERIENCE WITH CTSS

- HIGH OVERHEAD FOR SWAPPING
- 'GROWTH' OF DATA AREAS-LIST PROCESSING, ON-LINE COMPILING/ASSEMBLY
- PRACTICAL LIMIT OF 25-30 ON-LINE USERS
- GENERAL COMPUTING REQUIREMENTS
- NOTION OF COMPUTER UTILITY

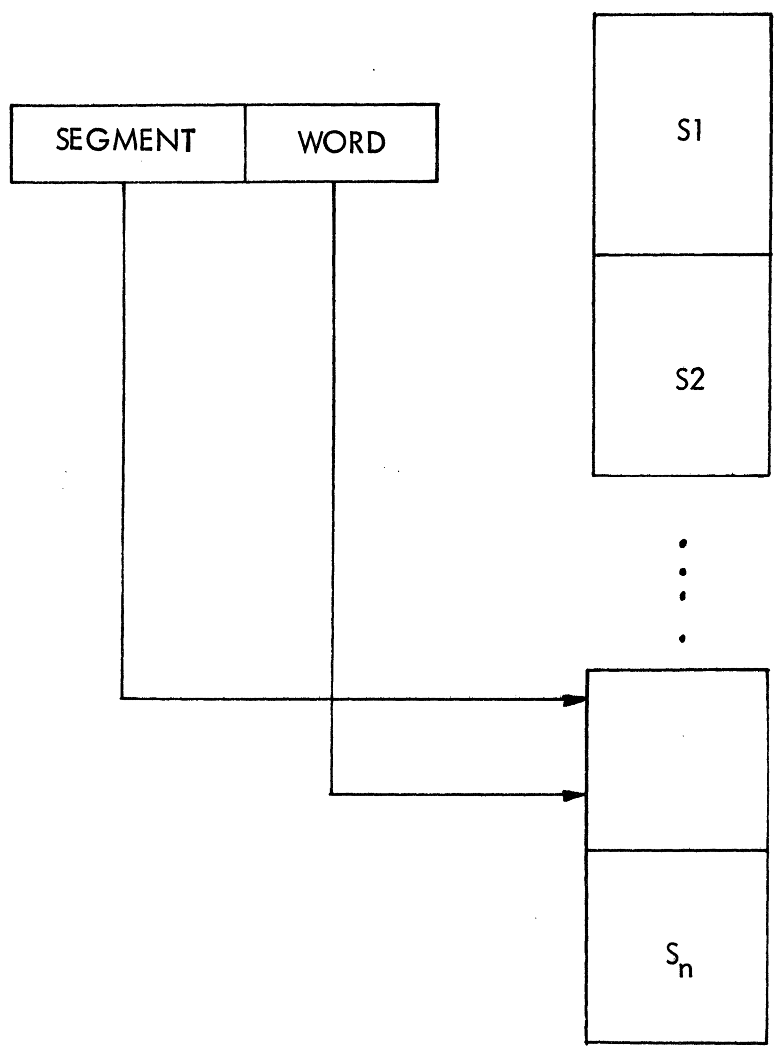
▶ APPROACHES TO PROVIDING USER ADDRESS SPACE

- EARLY ASSEMBLERS

  REGIONAL ADDRESSING

- ALGOL BLOCK STRUCTURE
- SEGMENT RELATIVE ADDRESSING

TWO COMPONENT ADDRESSING



SEGMENT SELECTS INFORMATION STRUCTURE SEGMENT

WORD SELECTS WORD WITHIN SELECTED SEGMENT

▶ DEFINITIONS

- SEGMENT: AN OBJECT (CODE, DATA, etc.) IN USER ADDRESS SPACE

GENERALIZED ADDRESS: CONTAINS

SEGMENT #

WORD #

SEGMENT	WORD

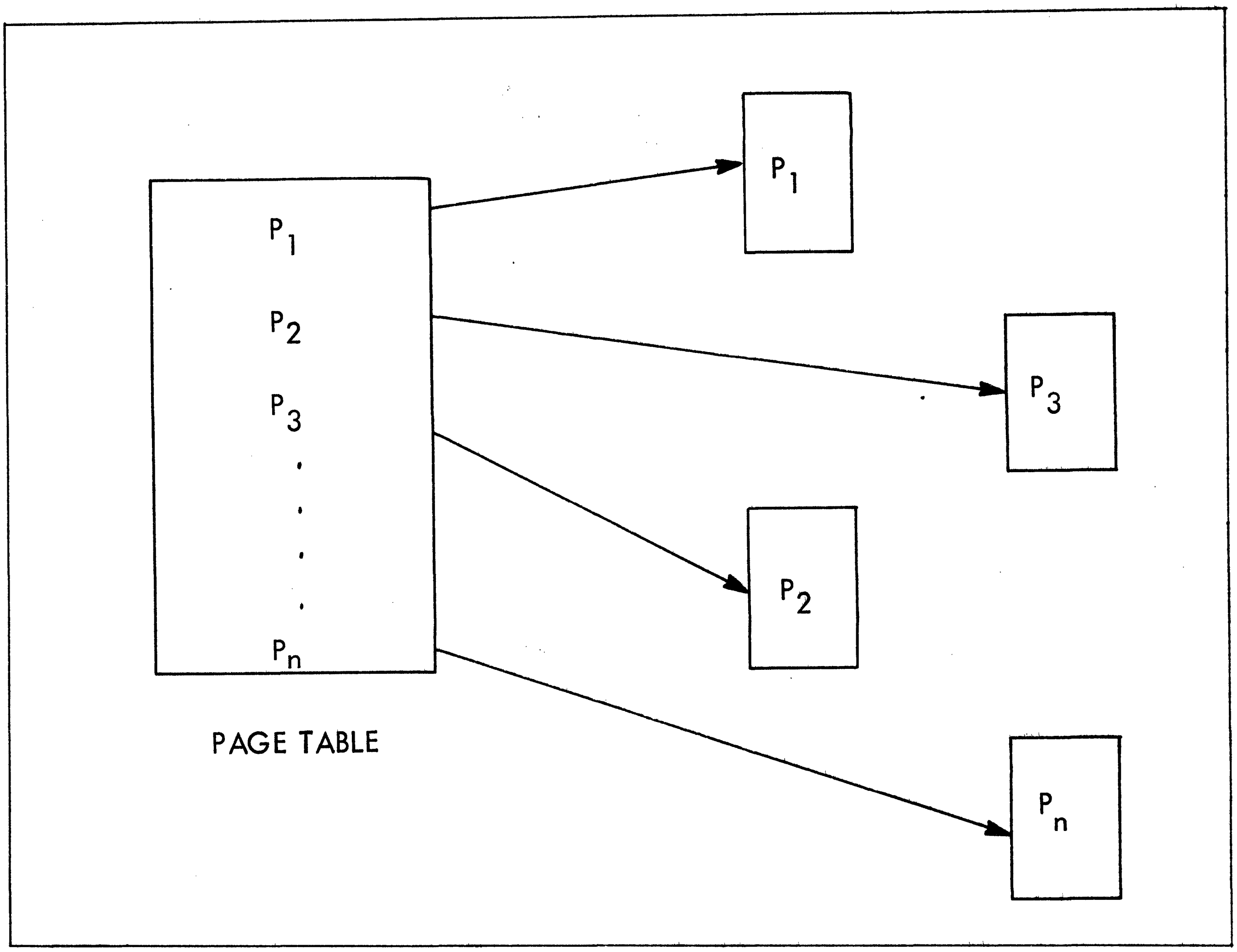
DESCRIPTOR: DEFINES AND LOCATES INFORMATION IN PHYSICAL MEMORY, - A BASE ADDRESS

▶ PAGE CONCEPT

- ORIGINS IN ATLAS SYSTEM
- FITS SWAPPING REQUIREMENT
- DEFINITION:

  UNIT OF RELOCATABLE STORAGE

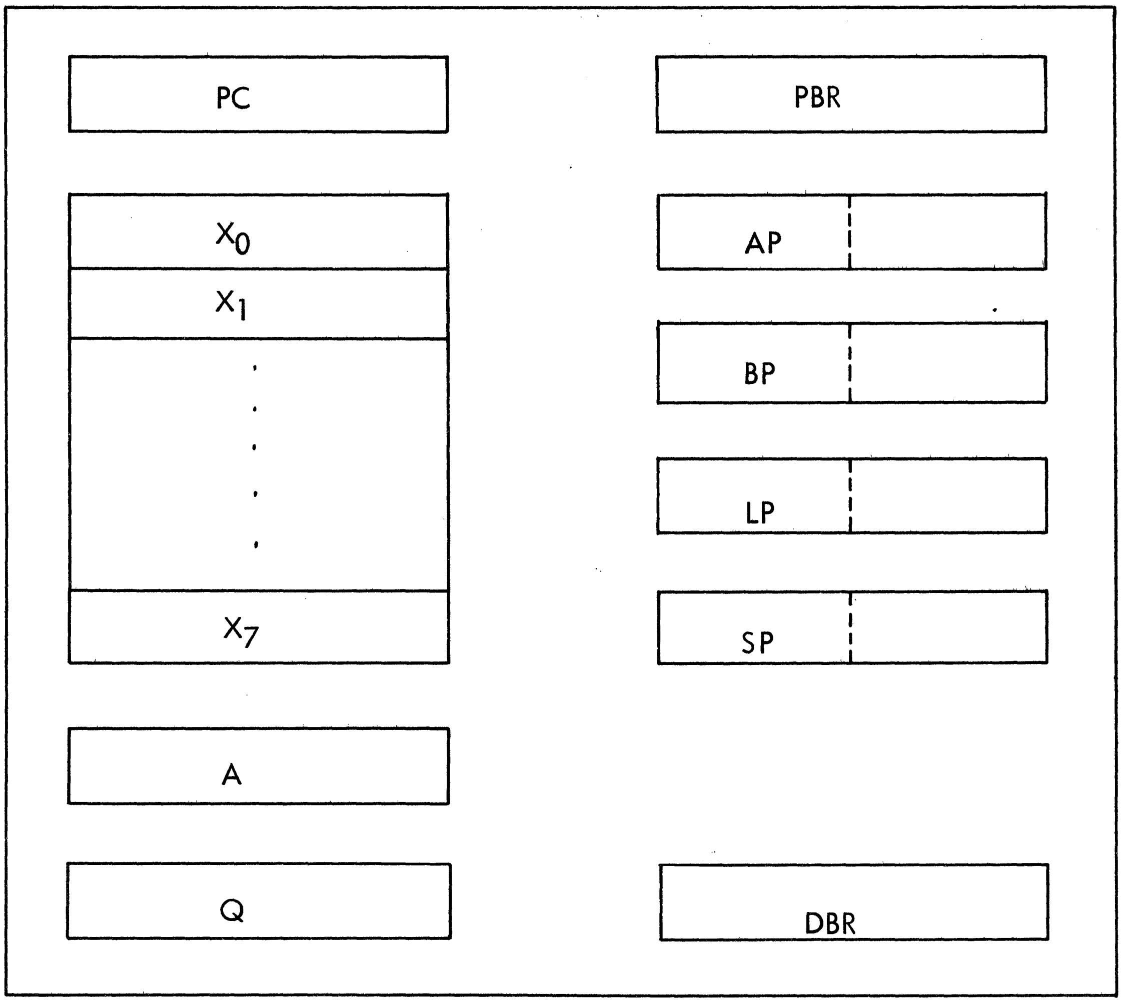
PRIMARY STORAGE



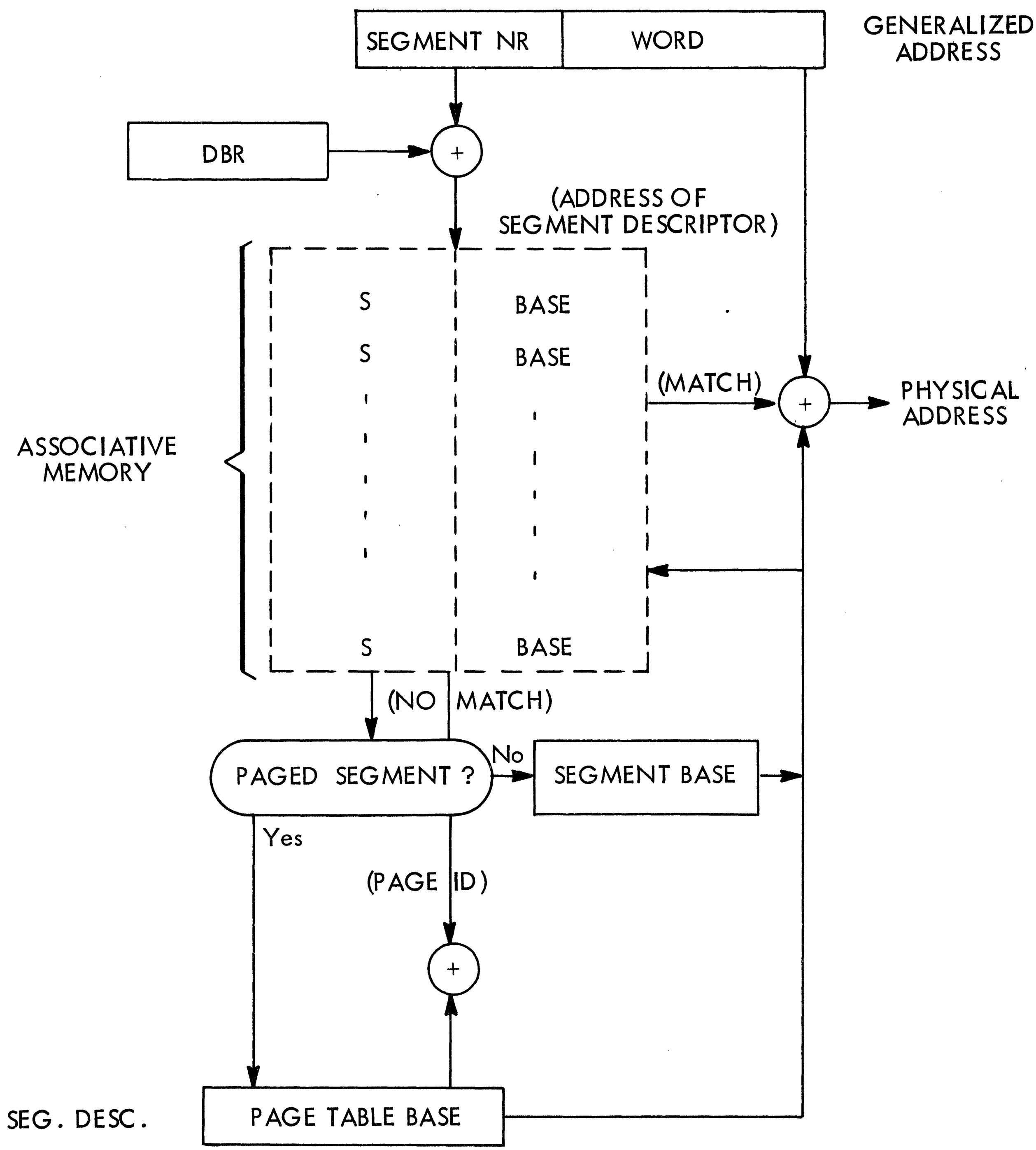
ILLUSTRATE USE OF DESCRIPTORS IN PAGE TABLE

EACH DESCRIPTOR POINTS TO A BLOCK (PAGE)

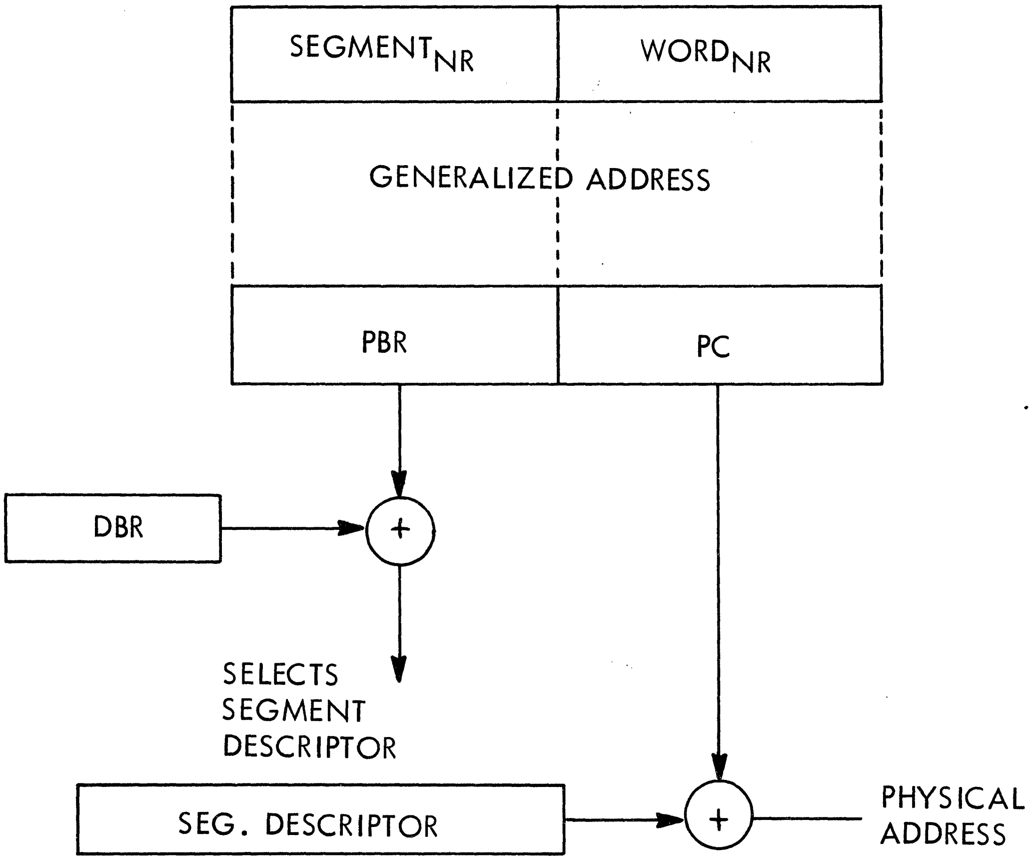
645 REGISTERS



GE645 ADDRESSING



645 INSTRUCTION ADDRESSING



▶ 645 ADDRESSING CHARACTERISTICS

- INFORMATION STRUCTURE MAY BE

  2^{18} SEGMENTS

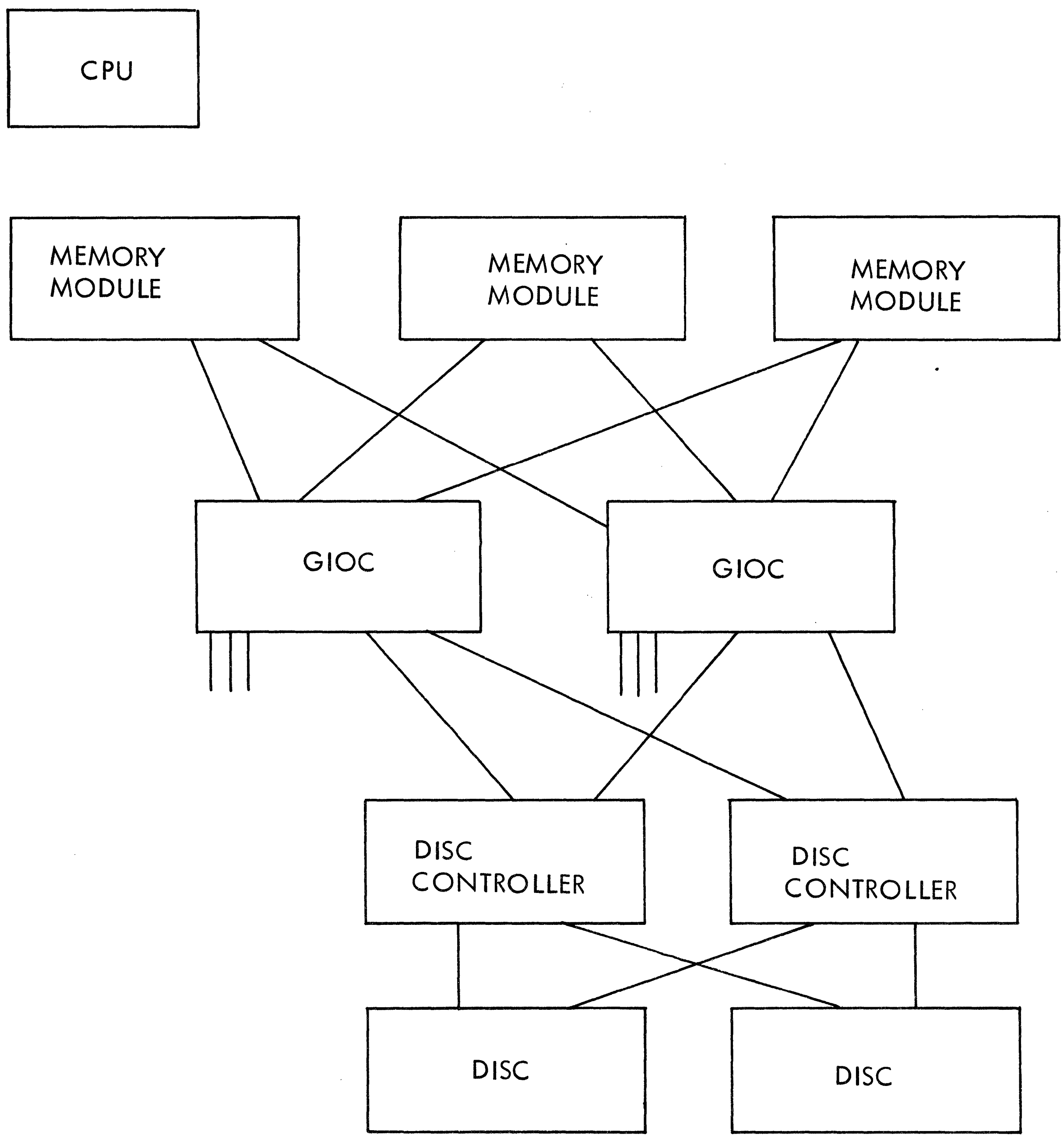
  EACH SEGMENT MAY BE

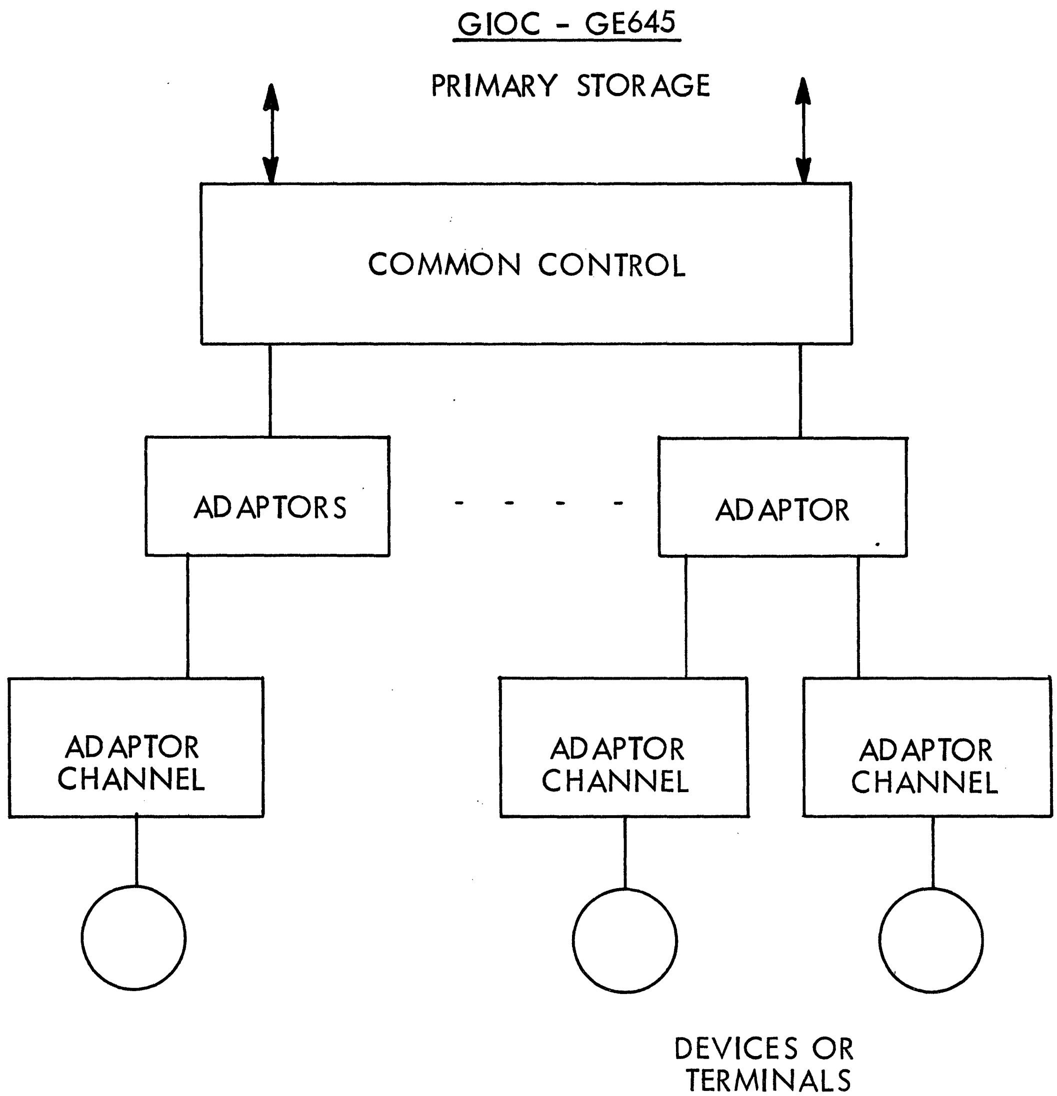
  2^{18} WORDS.

- PAGES

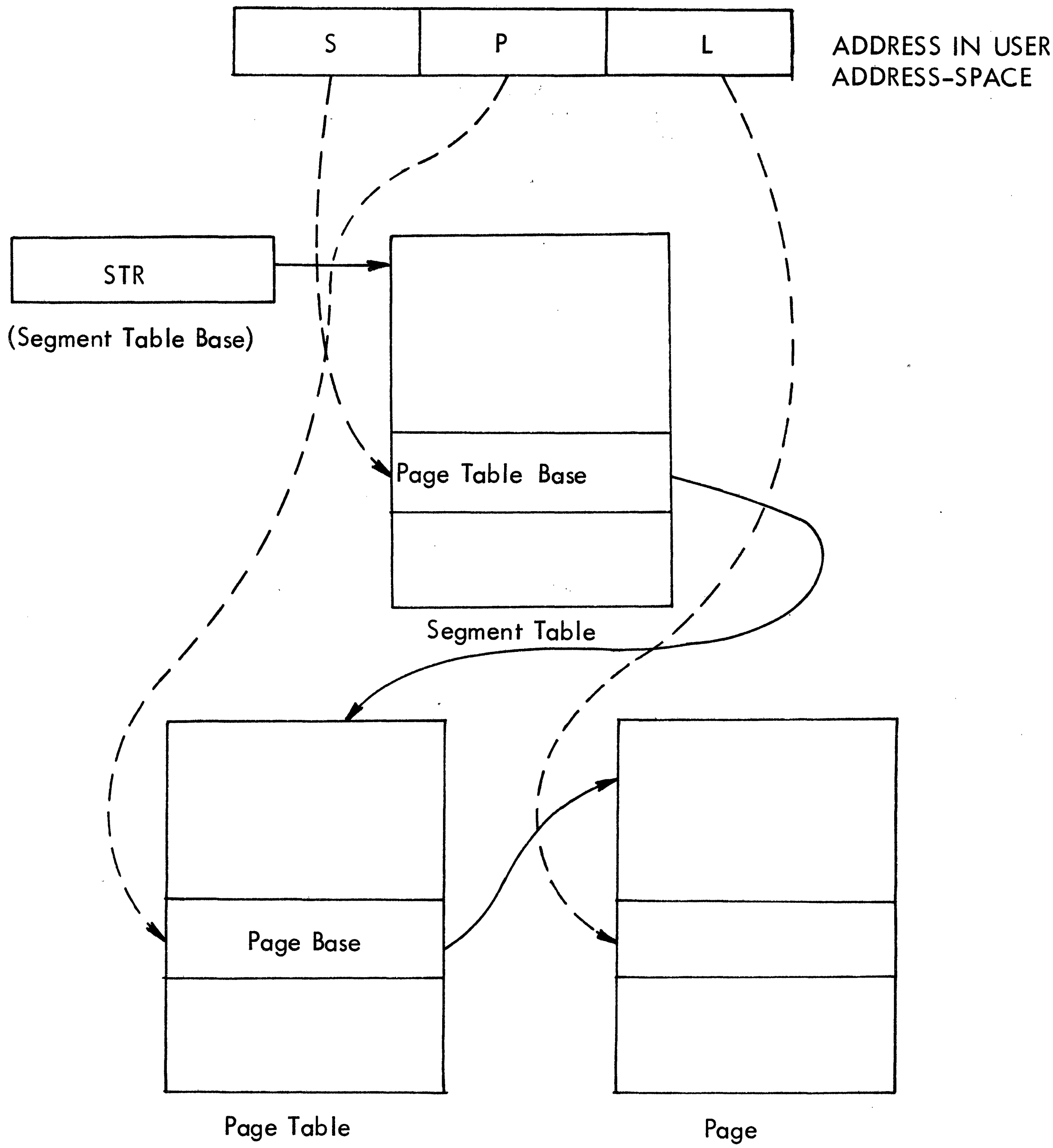
  64 OR 1024 <u>WORDS</u>

I/O CONTROL - GE 645

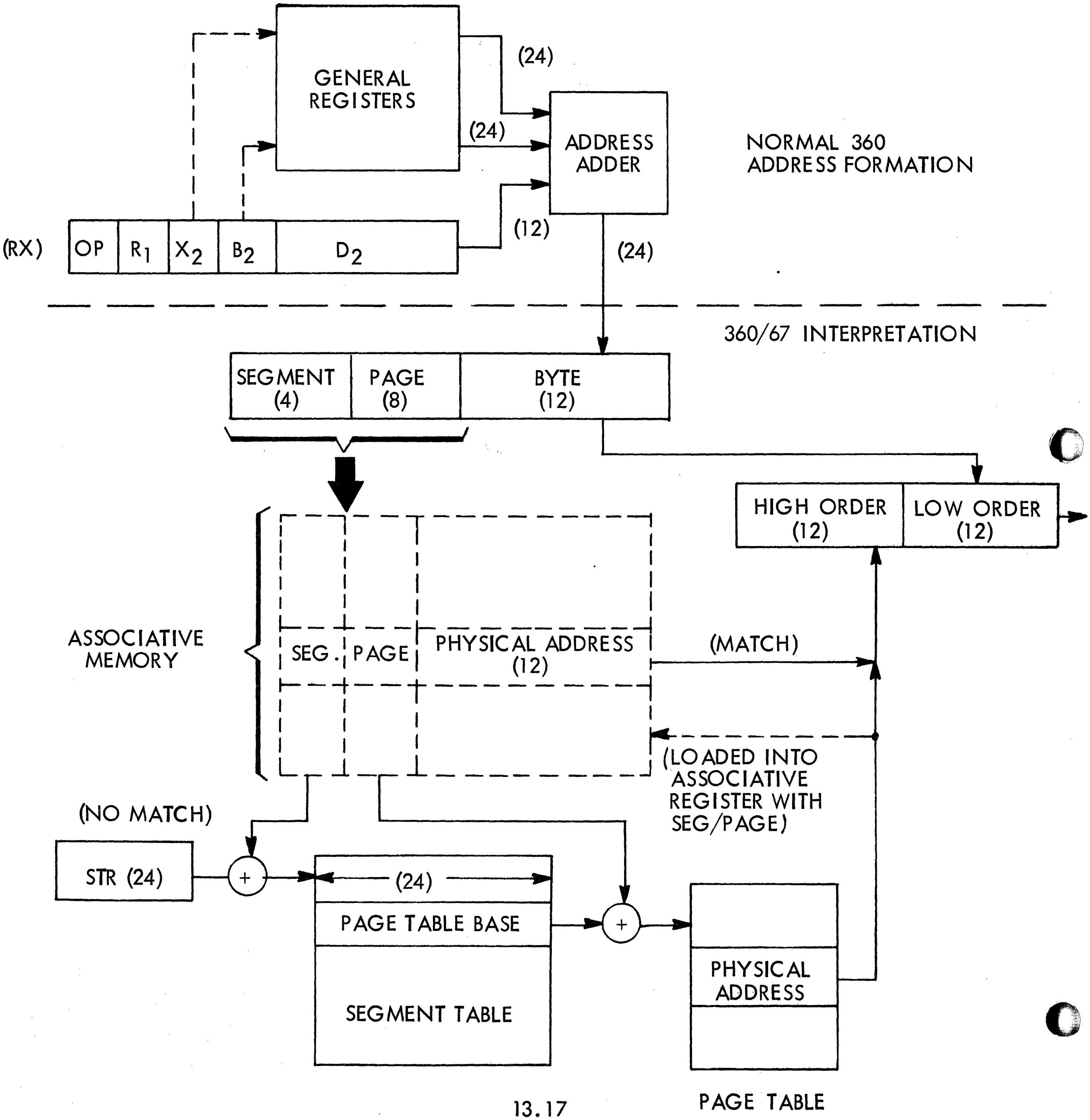



GIOC - GE645
PRIMARY STORAGE
COMMON CONTROL
ADAPTORS
ADAPTOR
ADAPTOR CHANNEL
ADAPTOR CHANNEL
ADAPTOR CHANNEL
DEVICES OR TERMINALS

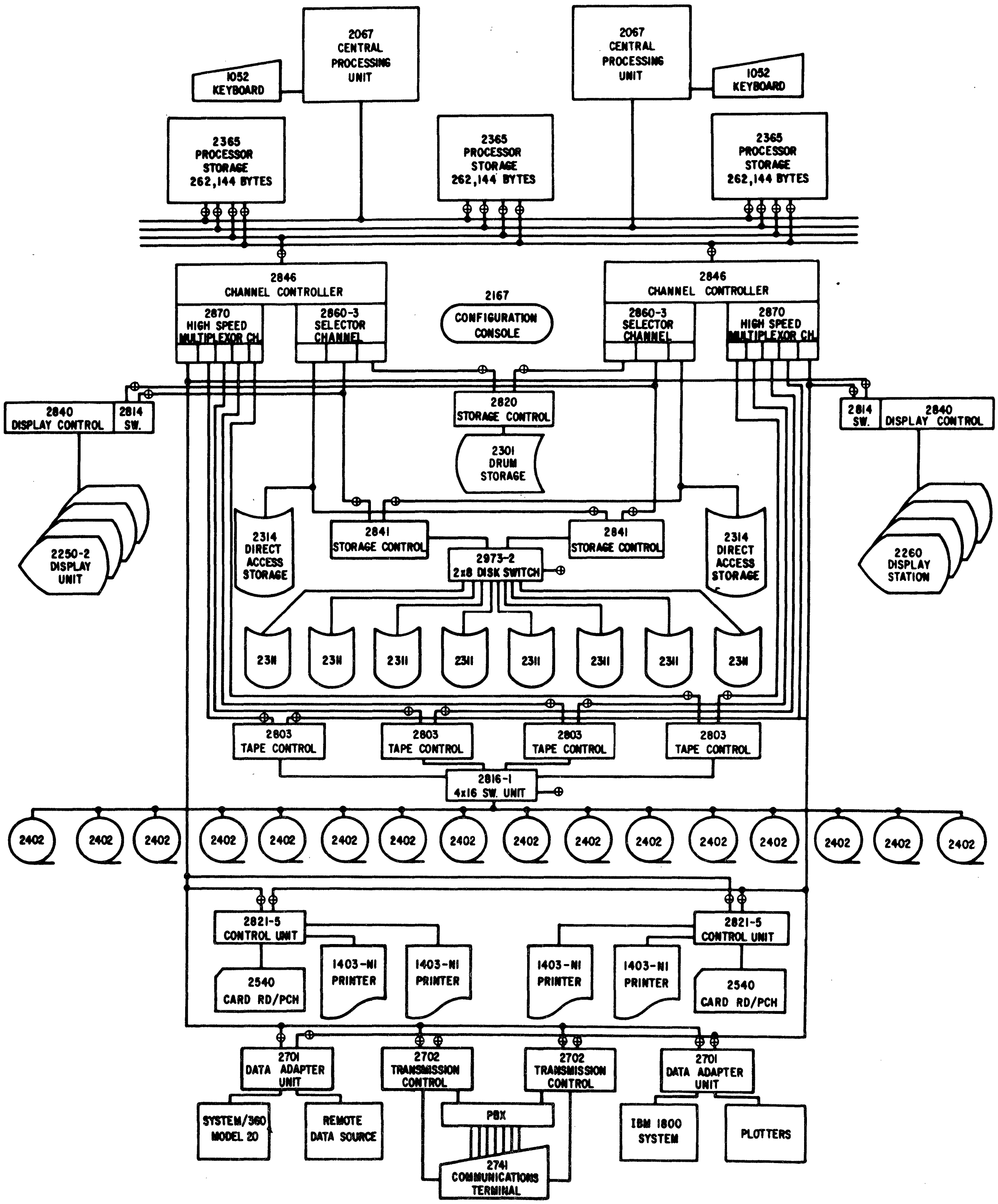
SEGMENT-PAGE ADDRESSING



360/67 ADDRESSING



360/67 WITH PARTITIONING SWITCHES



PAGING ADVANTAGES AND DISADVANTAGES

- PERMITS ARBITRARY ALLOCATION OF STORAGE IN SMALL BLOCKS
- DEFERS BINDING UNTIL EXECUTION TIME PERMITS ALLOCATION AND EXECUTION OF FRAGMENTS OF PROGRAMS
- COUPLED WITH OPERATING SYSTEM, PERMITS EACH USER TO HAVE EXTREMELY LARGE ADDRESS SPACE
- NOT ALL PROGRAMS REQUIRE TREATMENT AS ABOVE
- EXPENSIVE IN TIME AND MONEY FOR MANY APPLICATIONS
- THERE ARE OTHER WAYS TO ACHIEVE SAME ENDS

OTHER MACHINES ORIENTED TO TIME-SHARING

- SDS 940
- SDS SIGMA 7
- CDC 3500
- PDP 10

▶ SUMMARY OF PERTAINENT ADDRESSING CONCEPTS

SEGMENTS - COMPONENT OF USER ADDRESS SPACE

PAGE COMPONENT OF PHYSICAL ADDRESS SPACE

PAGING: MAPS SEGMENTS (USER ADDRESS SPACE) INTO PAGES (PHYSICAL ADDRESS SPACE)

MOTIVATION

- MULTIPLE INFORMATION AND COMPUTING SERVICE
- COMPUTER UTILITY

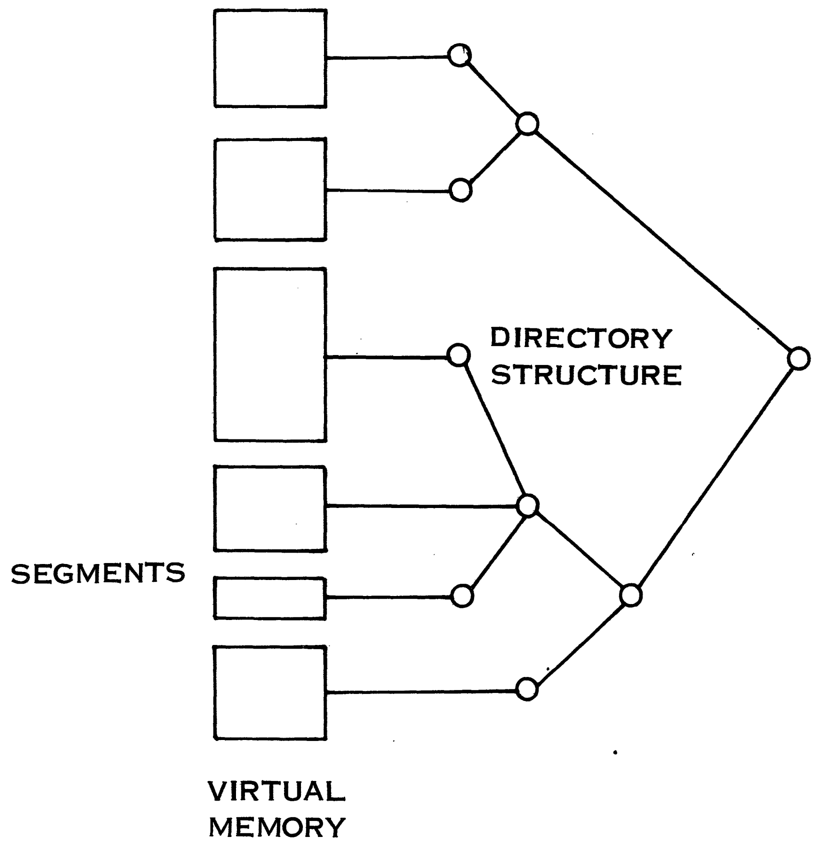
HARDWARE

- TWO—LEVEL ADDRESSING
- ONE—LEVEL STORE
- SEGMENTATION BY USER
- PAGING BY SYSTEM

SOFTWARE

- SYMBOLIC SEGMENT REFERENCES
- RECURSIVE PROCEDURES
- LOCATION—INDEPENDENCE
- PRIVATE STACK FOR TEMPORARY STORAGE
- FILE SYSTEM
    - SYMBOLIC
    - ACCESS—CONTROLLED

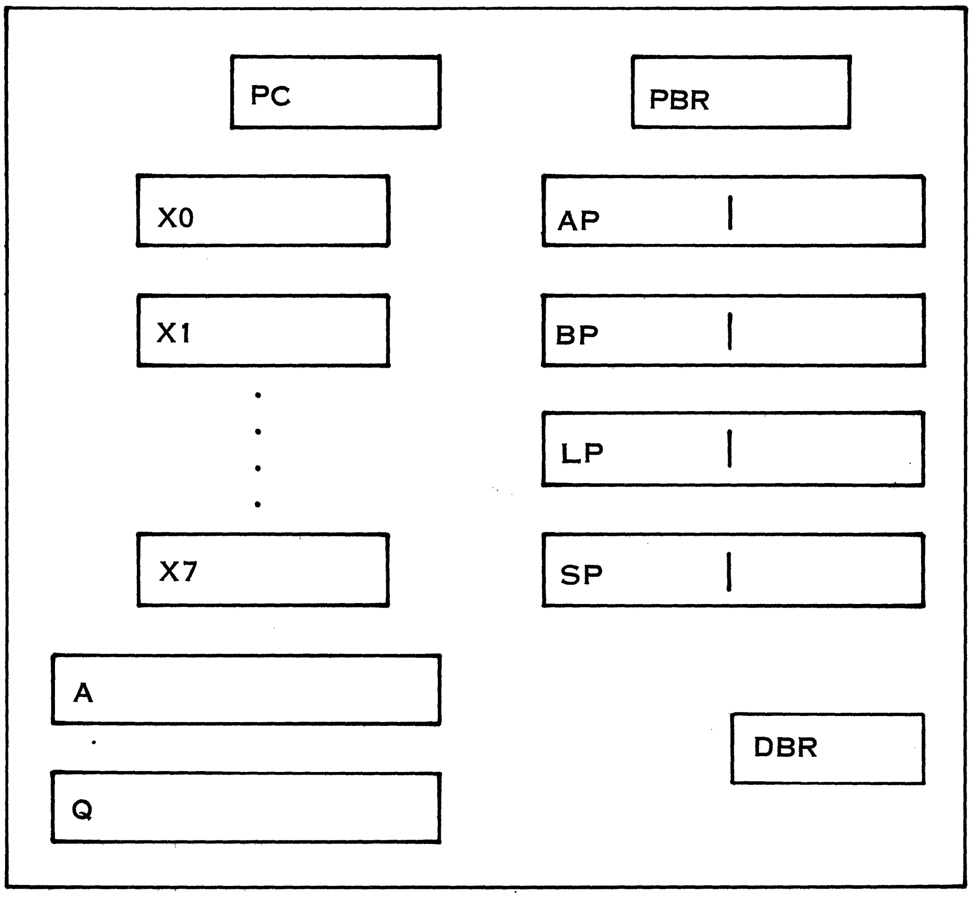
VIRTUAL MEMORY OF A MULTICS PROCESS



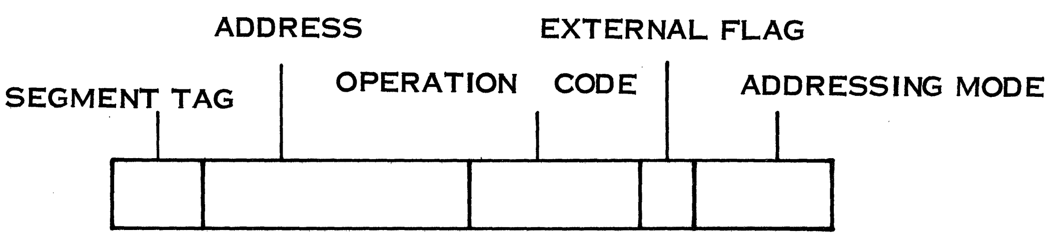
THE GENERALIZED ADDRESS

SEGMENT NUMBER	WORD NUMBER

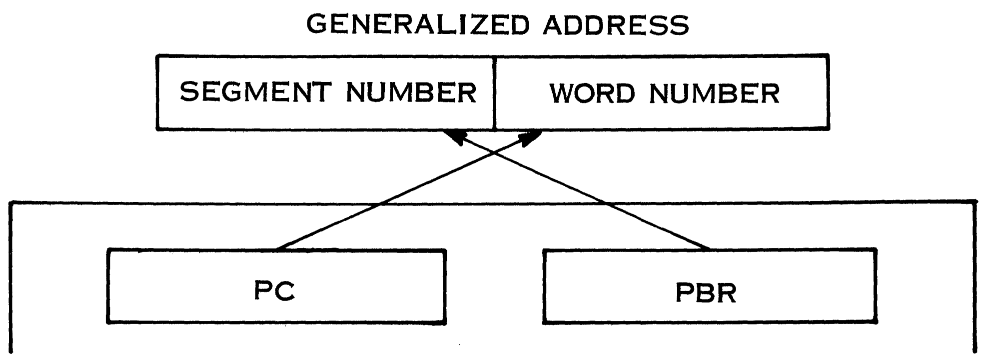
PROCESSOR REGISTERS FOR ADDRESS FORMATION



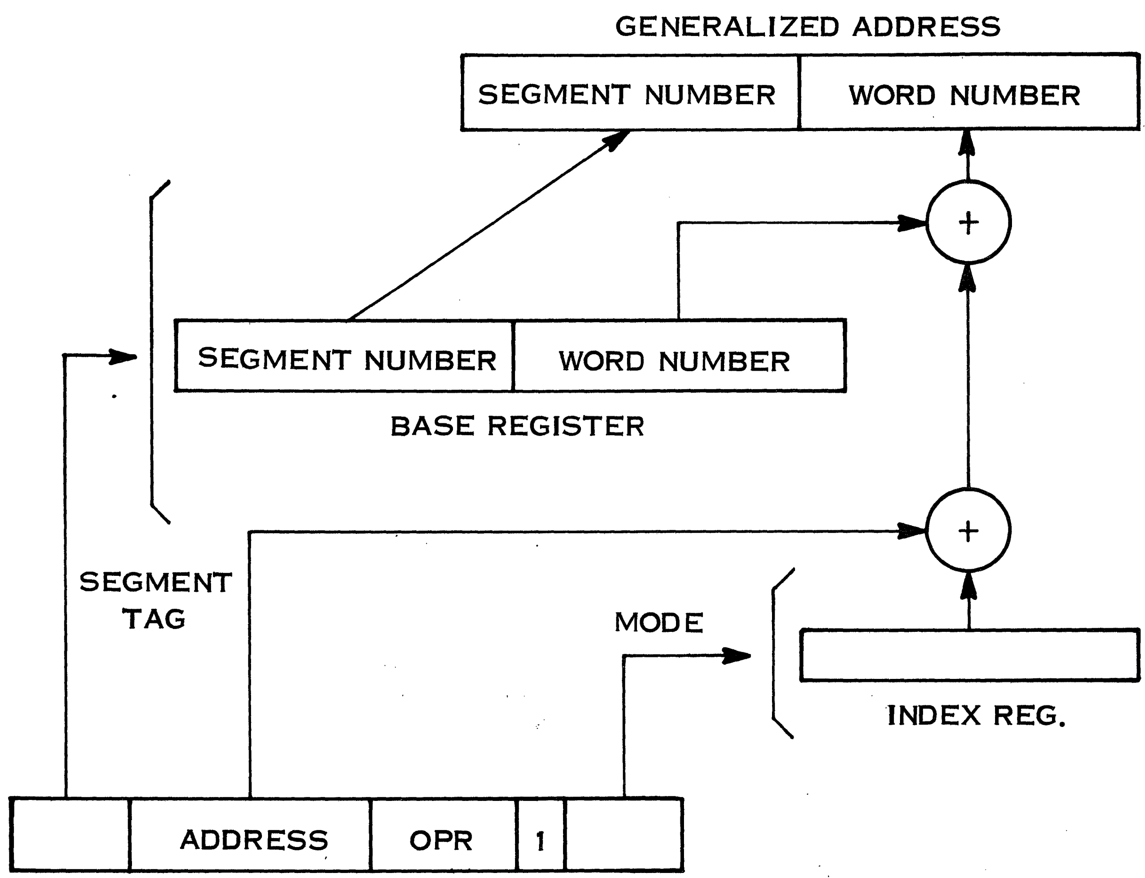
INSTRUCTION FORMAT



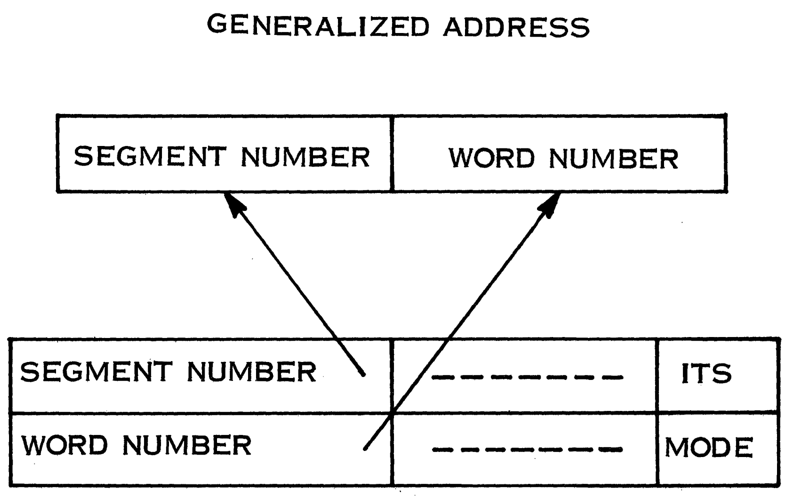
ADDRESS FORMATION FOR INSTRUCTION FETCH



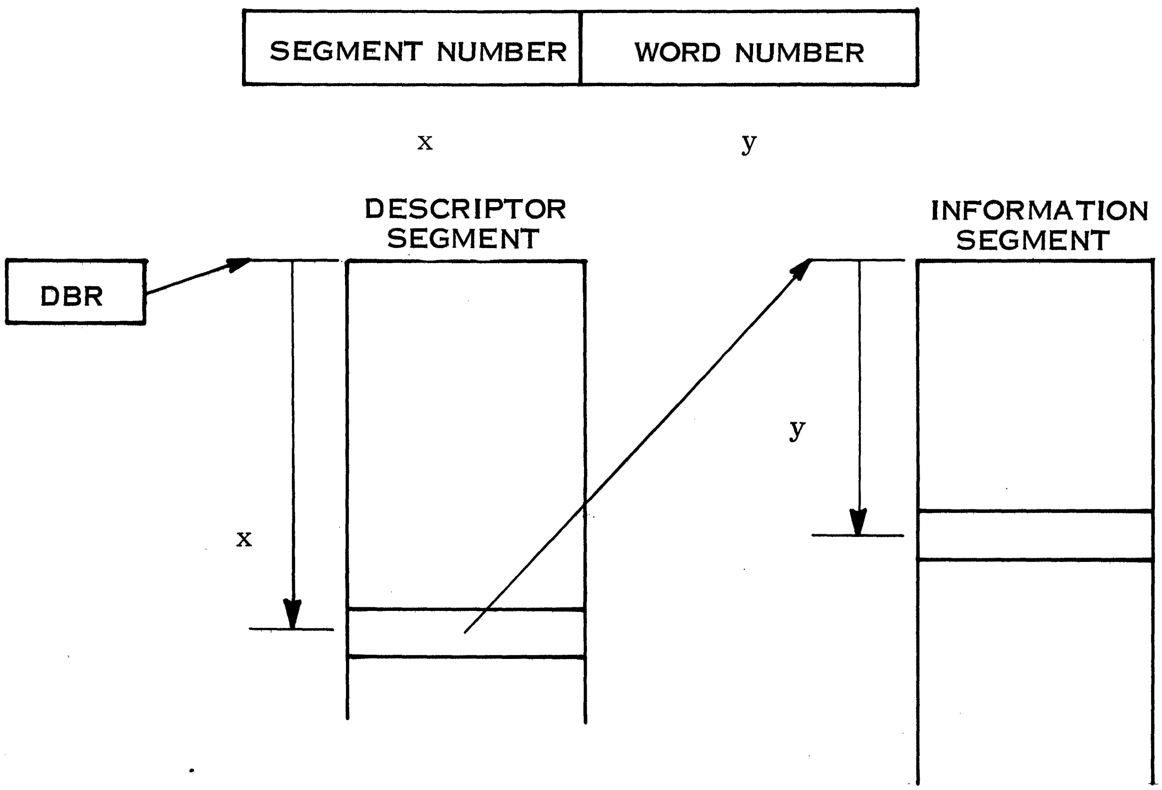
ADDRESS FORMATION FOR DATA ACCESS



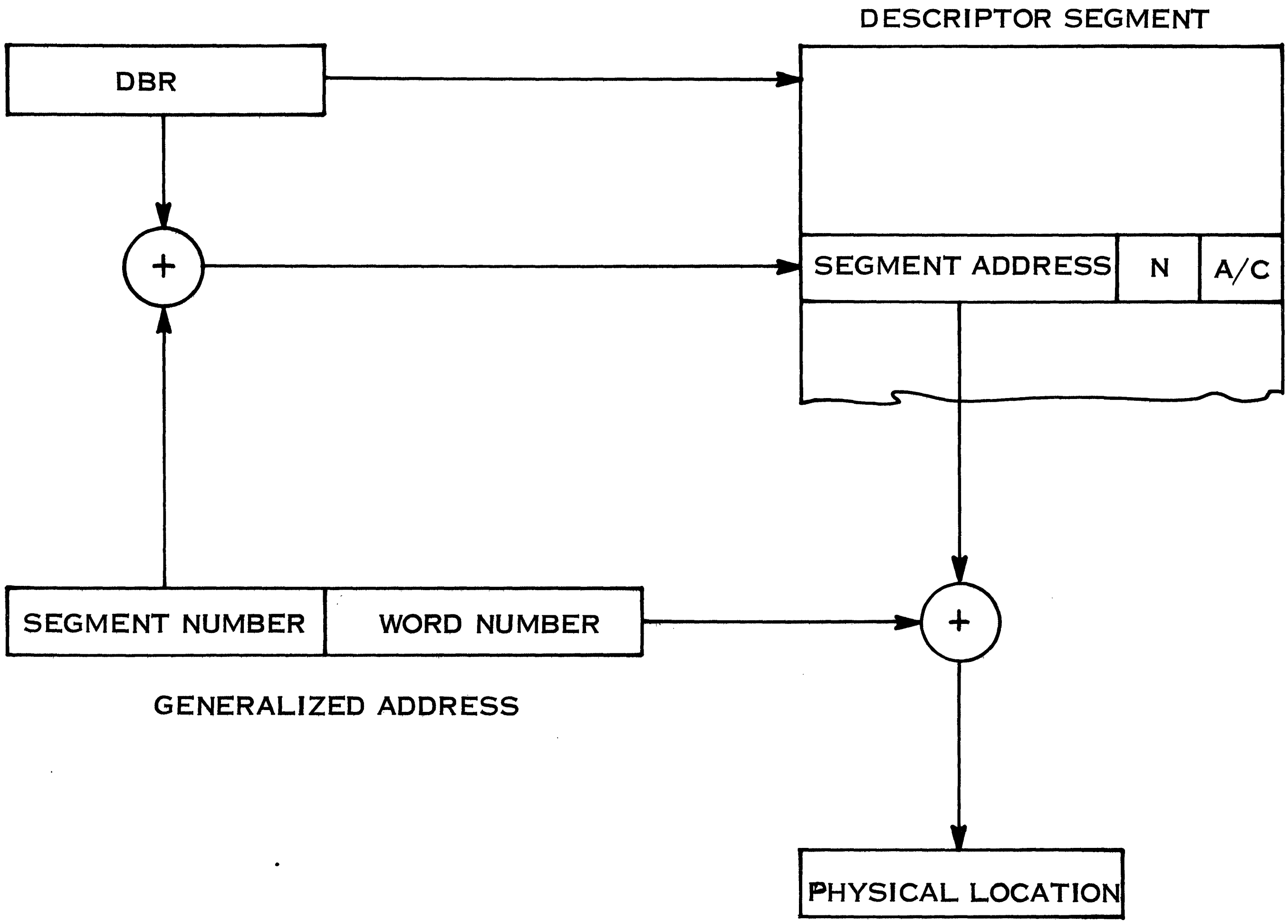
INTERPRETATION OF WORD PAIR AS INDIRECT ADDRESS



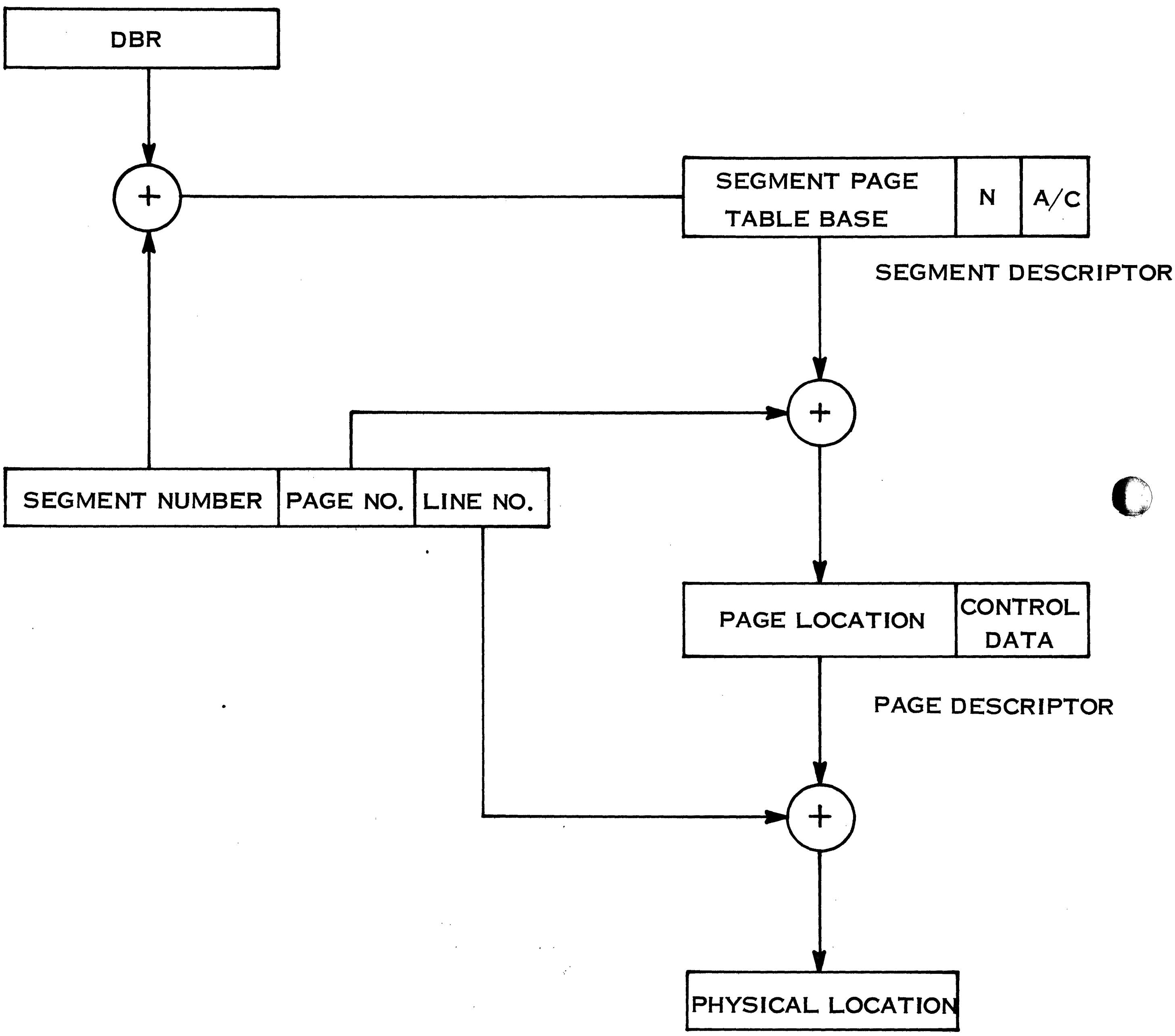
ADDRESSING BY GENERALIZED ADDRESS



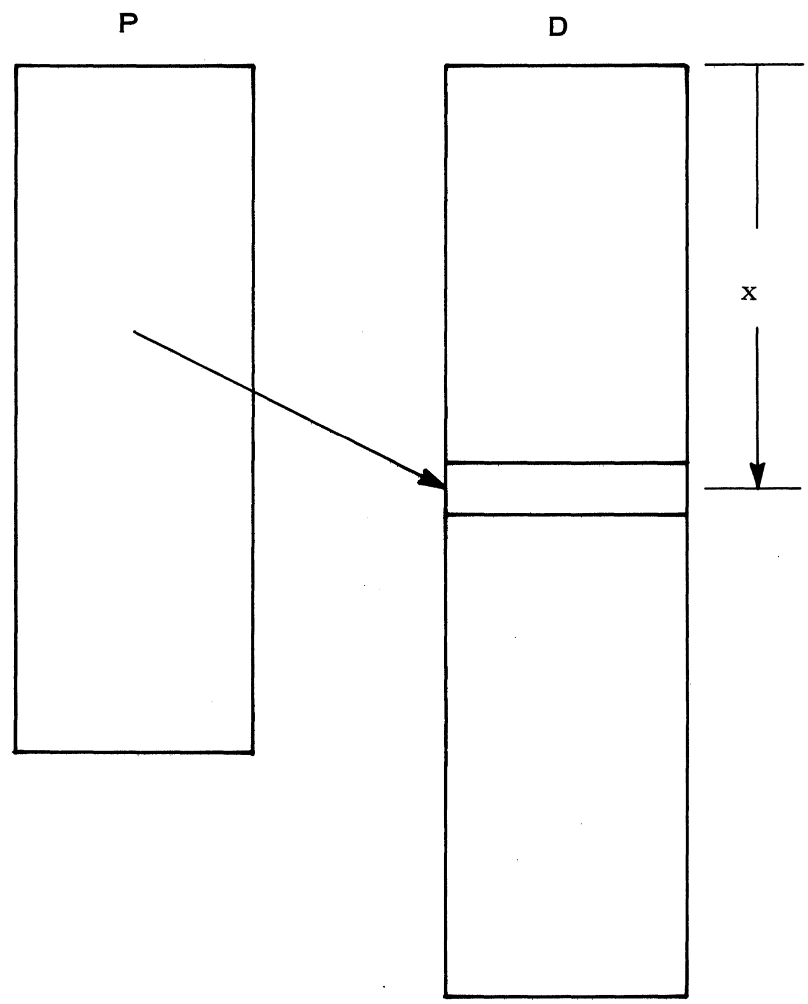
ADDRESS FORMATION FOR AN UN—PAGED SEGMENT



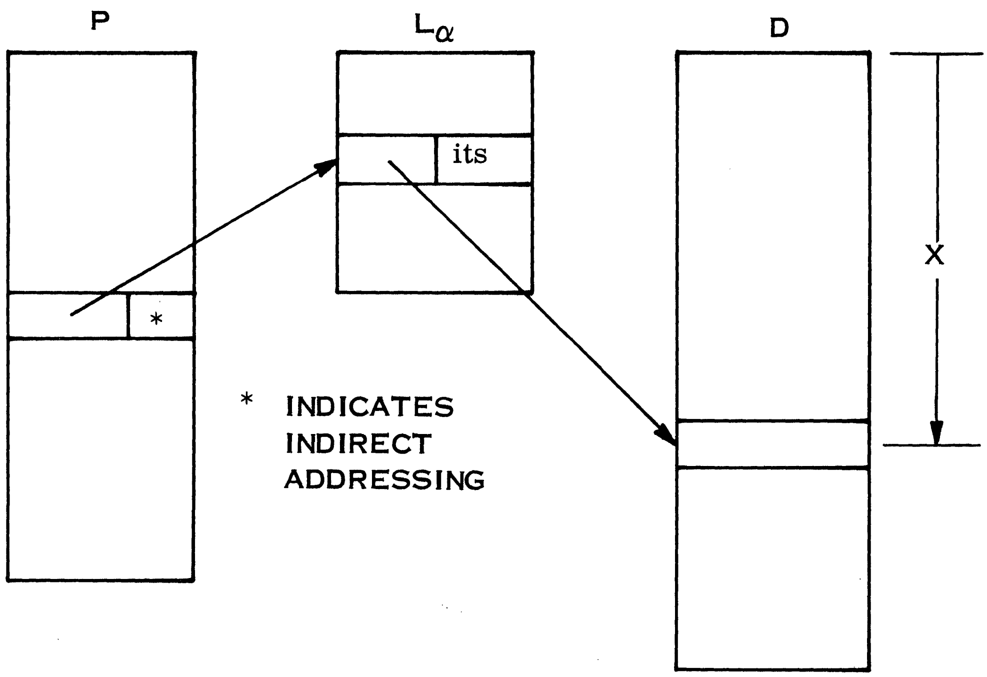
ADDRESS FORMATION FOR A PAGED SEGMENT

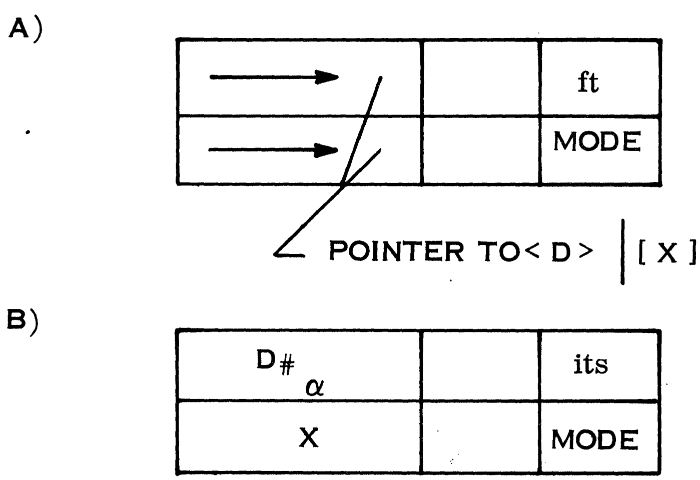


AN INTERSEGMENT REFERENCE BY PROCEDURE P



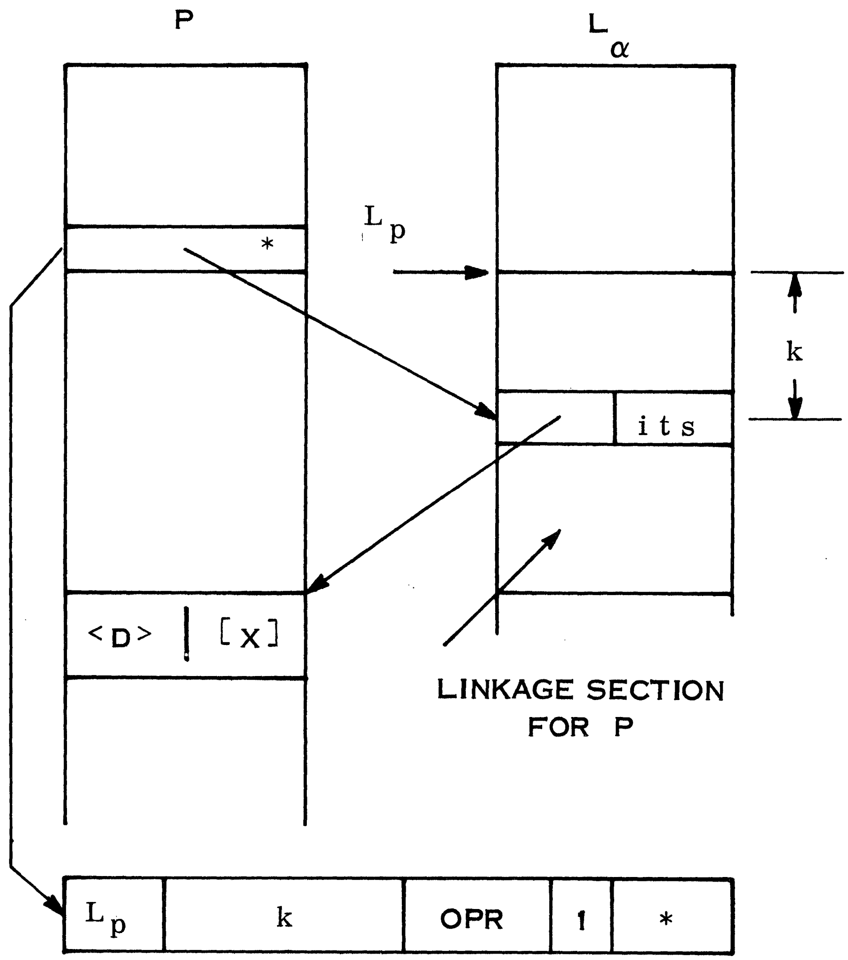
LINKAGE OF P TO D | X FOR PROCESS α



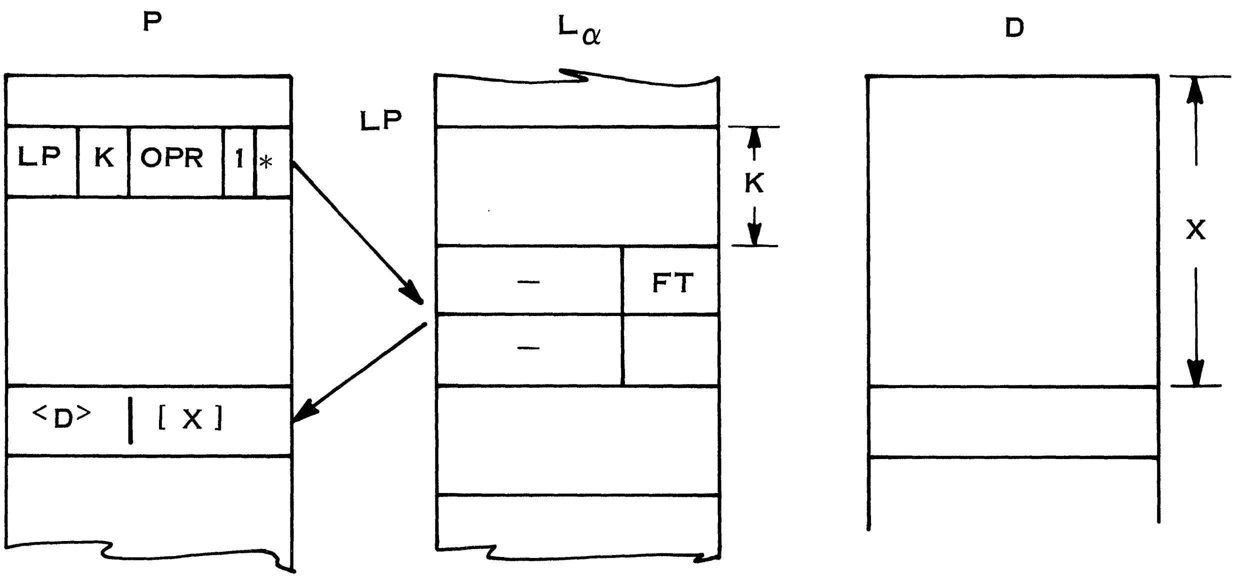


STATES OF THE LINK DATA

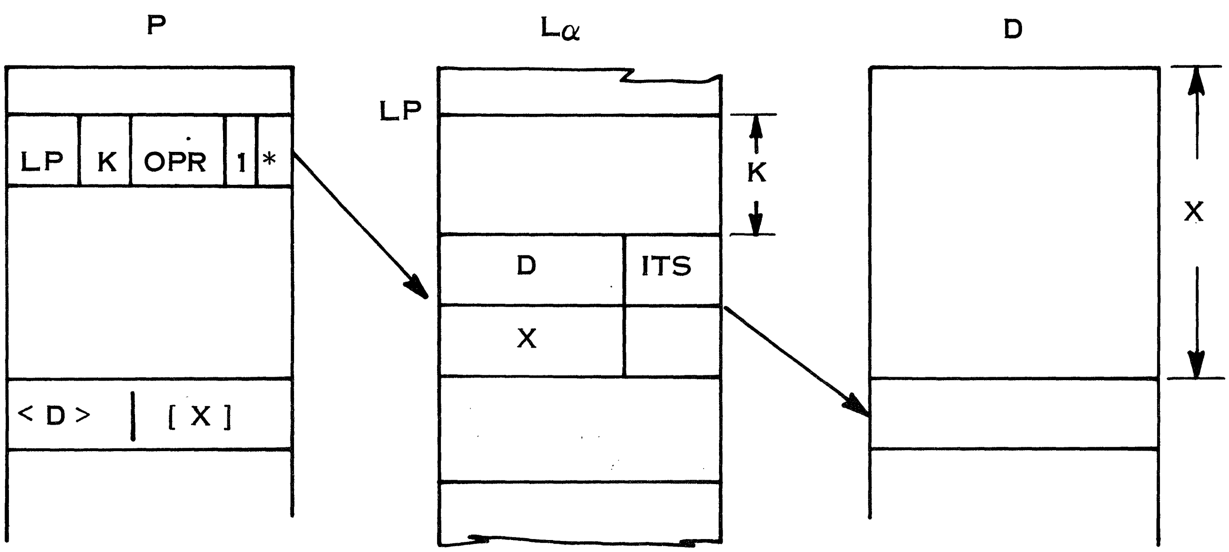
ADDRESSING THE LINK DATA



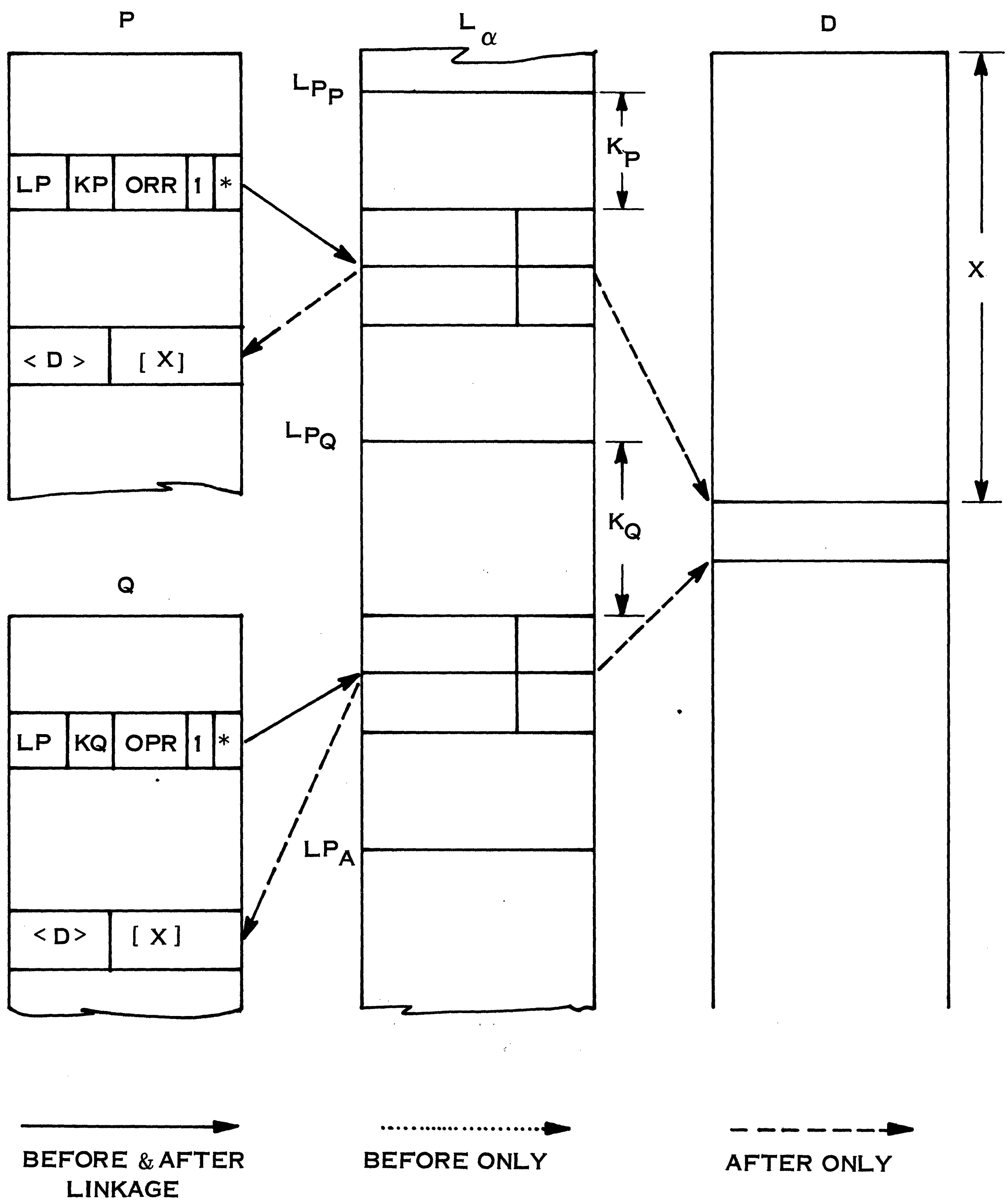
PROCEDURE P IN PROCESS α BEFORE LINKAGE



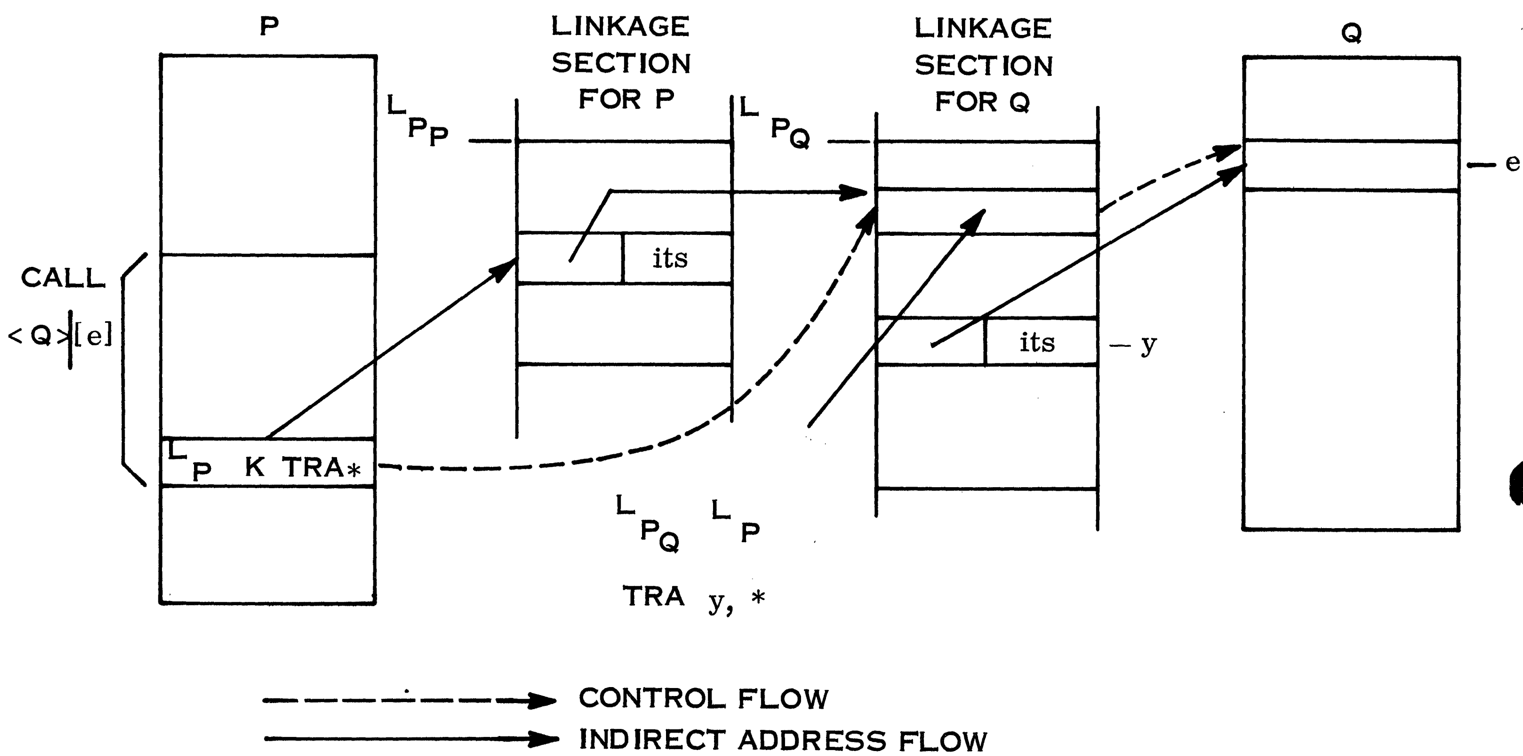
PROCEDURE P IN PROCESS α AFTER FIRST EXECUTION



REFERENCE TO COMMON DATA SEGMENT BY TWO PROCEDURES



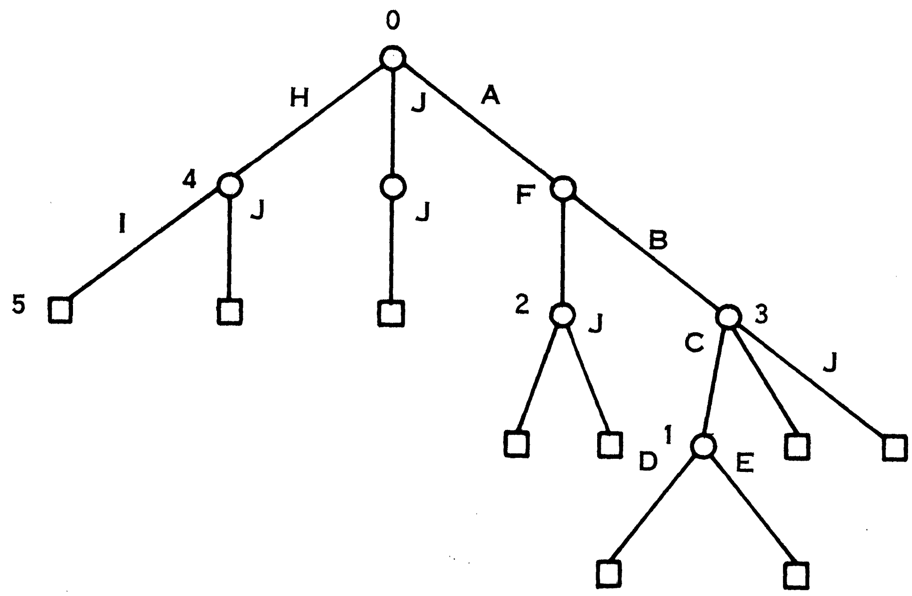
LINKAGE MECHANISM FOR PROCEDURE ENTRY



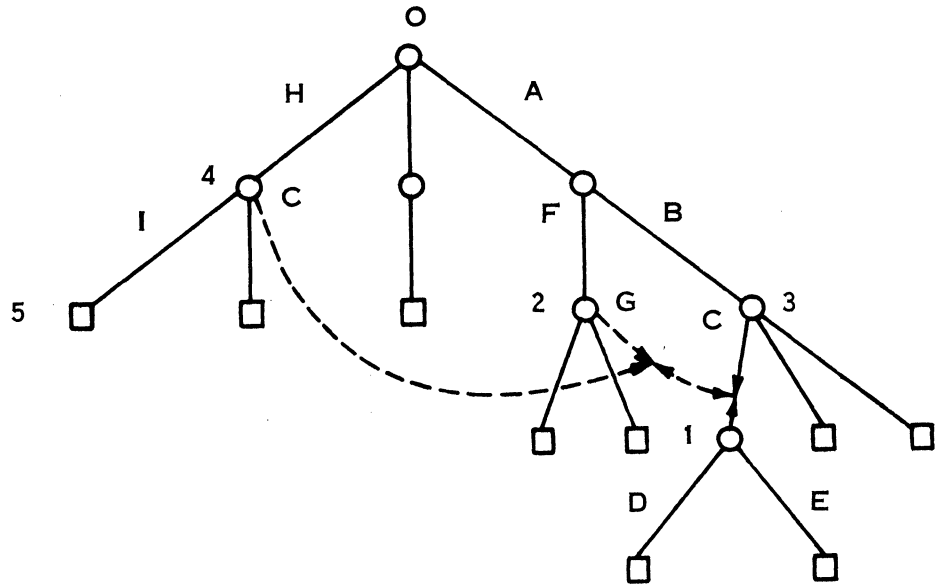
STATUS OF A PROCESS

- RUNNING
- READY
- BLOCKED

AN EXAMPLE OF A HIERARCHY



THE SAME HIERARCHY WITH LINKS ADDED



DIRECTORY MANIPULATION

1. SUPPOSE CURRENT WORKING DIRECTORY IS 4 (PATHNAME H)

2. THE COMMAND CHANGE DIRECTORY : C WILL ALTER THE WORKING DIRECTORY TO 1 (PATHNAME H: C)

3. A SUBSEQUENT REFERENCE TO :* : I WILL THEN INDICATE BRANCH 5.

ACCESS CONTROL

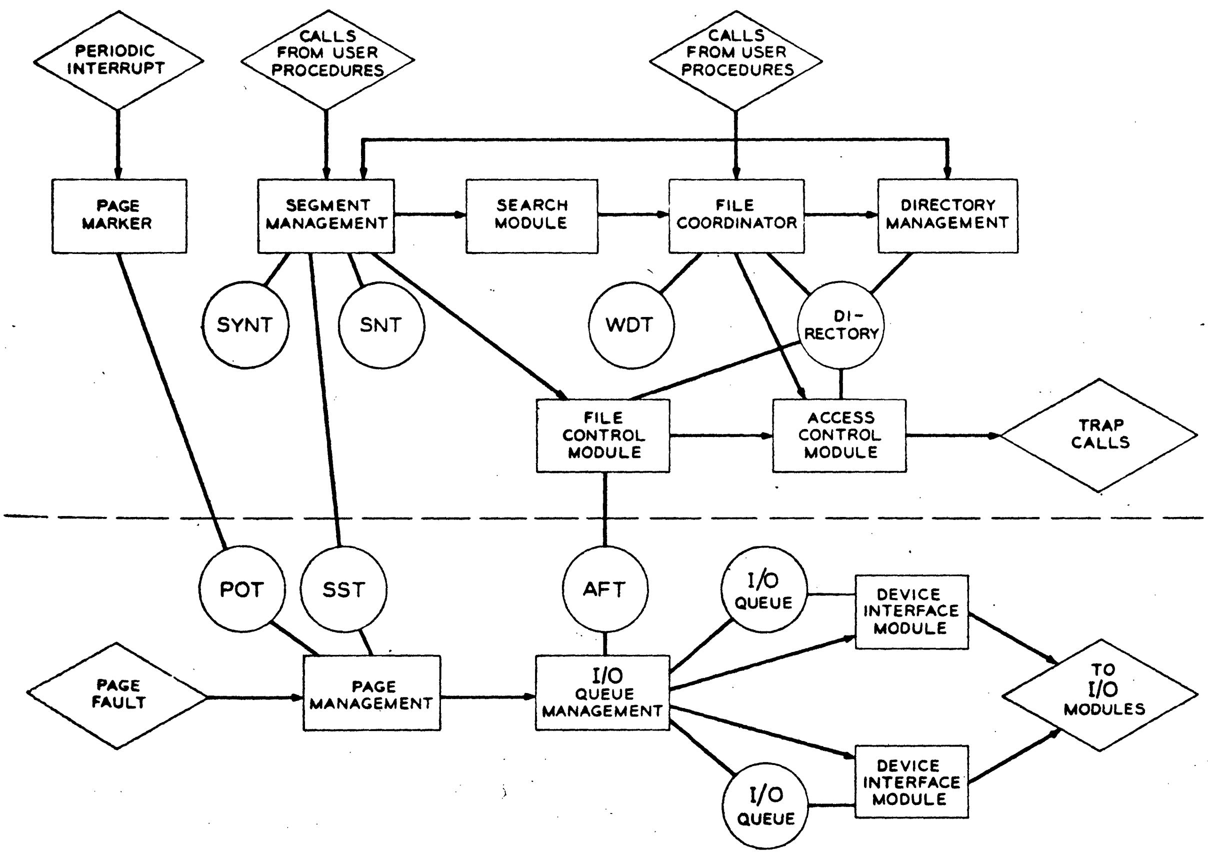
- USER ACCESS CONTROL LIST

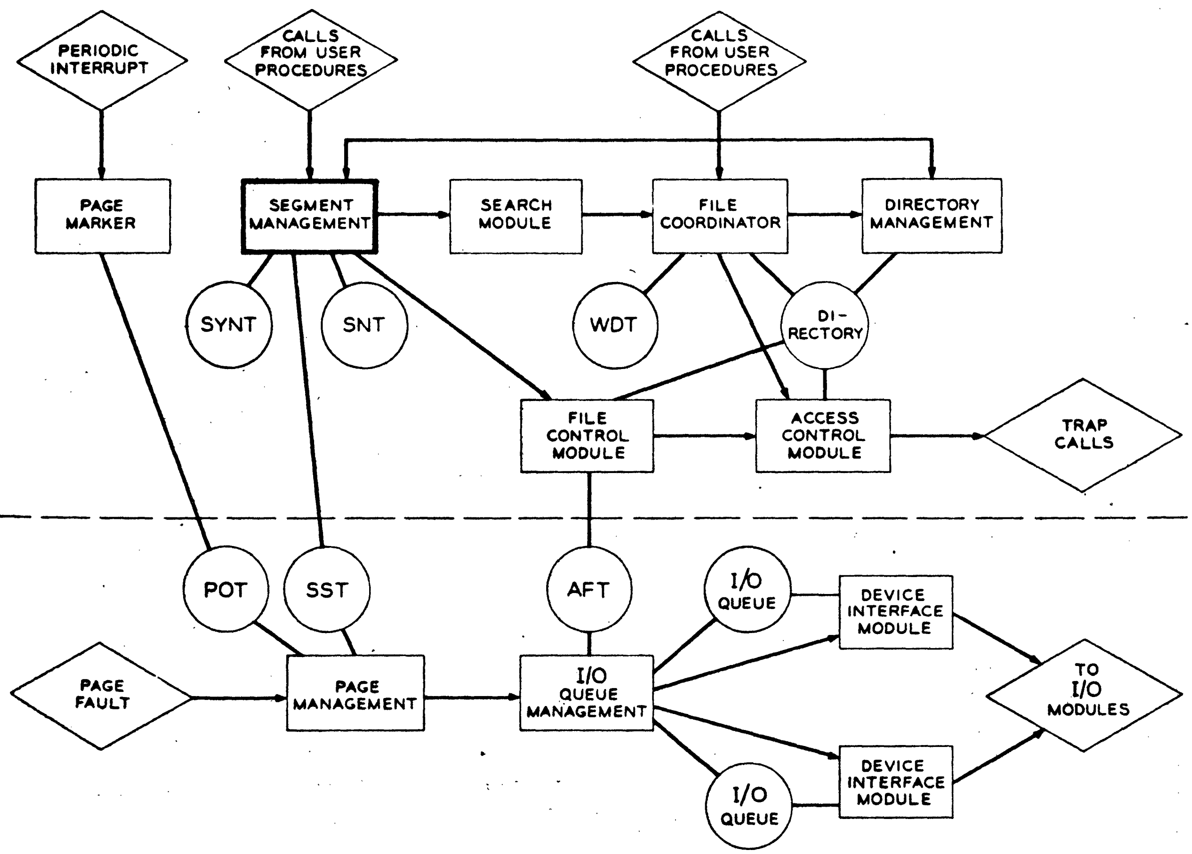
- MODE ATTRIBUTES:

MODE	DIRECTORY BRANCH	NON–DIRECTORY BRANCH
READ:	READ AVAIL. CONTENTS	READ FILE
WRITE:	ALTER EXISTING ENTRIES	WRITE FILE
EXECUTE:	SEARCH THE DIRECTORY	EXECUTE PROCEDURE
APPEND:	ADD NEW ENTRIES	WRITE AT E.O.F.

- THE TRAP ATTRIBUTE
    - MONITORS FILE USAGE
    - RESTRICTS ACCESS
    - DYNAMIC REFERENCE CONTROL

THE BASIC FILE SYSTEM





SEGMENT MANAGEMENT

- MAINTAINS RECORD OF ALL KNOWN SEGMENTS (S. N. T.)

  ACTIVE: IF PAGE TABLE IN CORE (S. S. T.)

  INACTIVE: IF PAGE TABLE NOT IN CORE

- CALLS LINKER FOR FIRST–TIME REFERENCE
- IF NOT IN SNT,

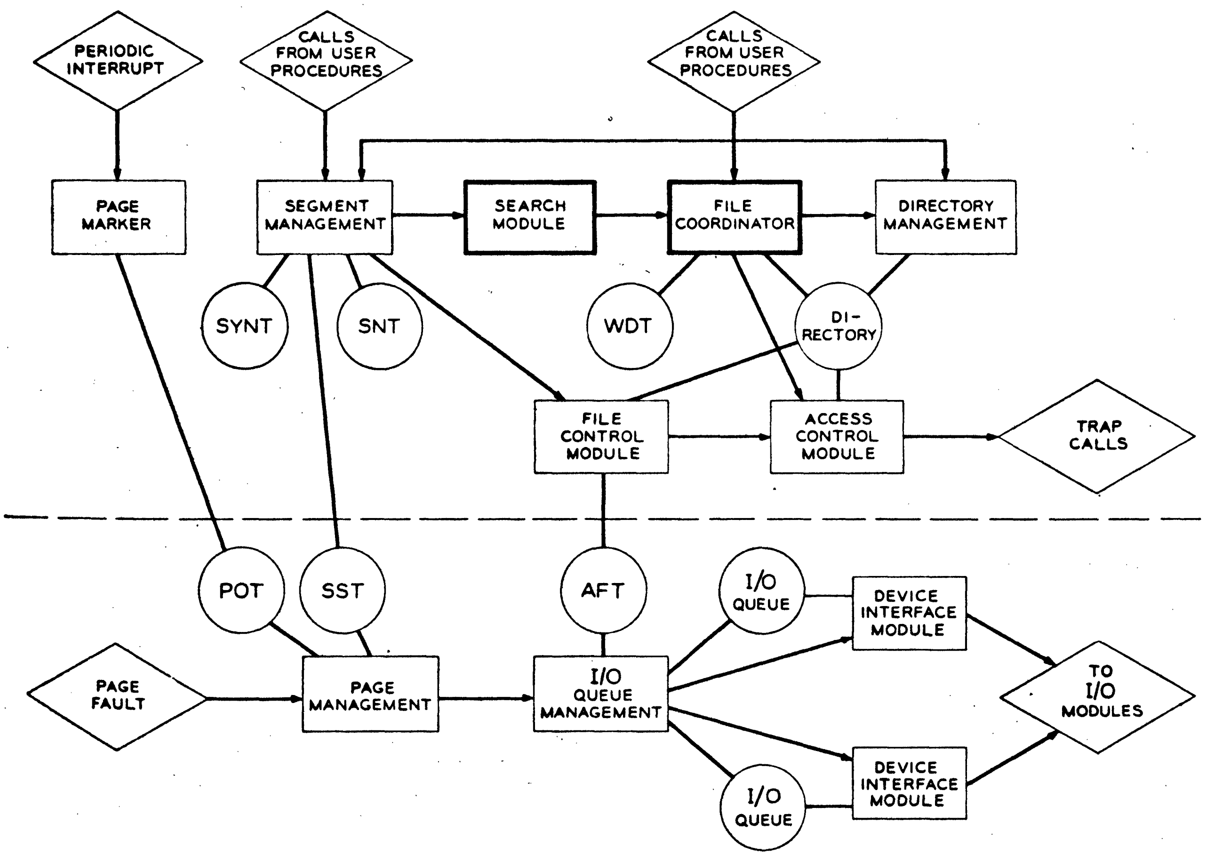
  LOCATE SEGMENT, ASSIGN SEGMENT NUMBER, UPDATE SNT,

  OPEN FILE, CREATE SST ENTRY, SET UP PAGE TABLE AND

  SEGMENT DESCRIPTOR; THEN

- RETURN SEGMENT NUMBER TO CALLING PROCEDURE
- IF IN SNT BUT INACTIVE, ACTIVATE
- OTHER FUNCTIONS:

  RELEASE, REASSIGN, VERIFY, CREATE, TERMINATE

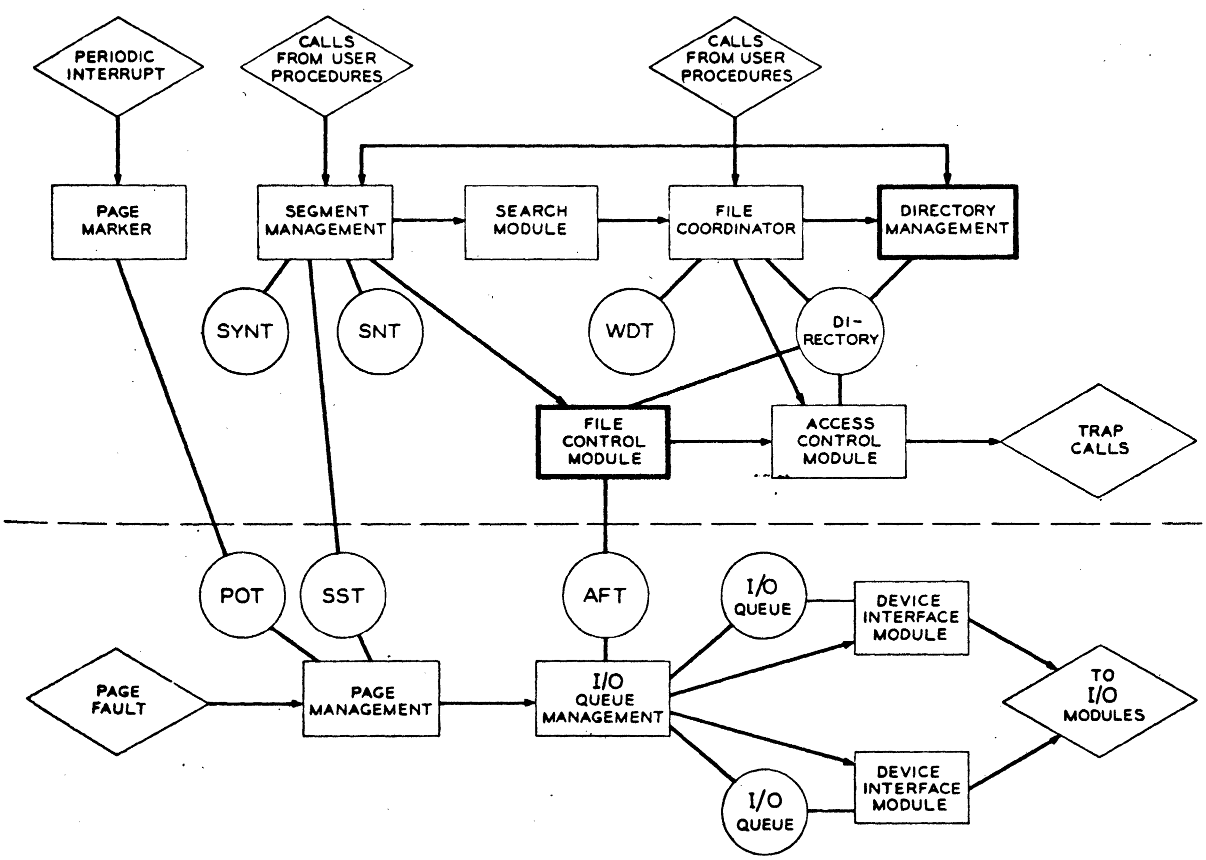


SEARCH MODULE

- CALLED SEGMENT MANAGEMENT
- USES FILE COORDINATOR
- LOCATES SPECIFIC BRANCH IN USER'S HIERARCHY

FILE COORDINATOR

- BASIC WORKING DIRECTORY ENTRY MANIPULATION
- INTERFACES WITH ACCESS CONTROL FOR PERMISSION
- KEEP TREE NAME OF WORKING DIRECTORY IN WDT
- FUNCTIONS:
  - CREATE, DELETE, RENAME AN ENTRY
  - STATUS OF AN ENTRY
  - CHANGE ACCESS CONTROL FOR A BRANCH
  - CHANGE WORKING DIRECTORY

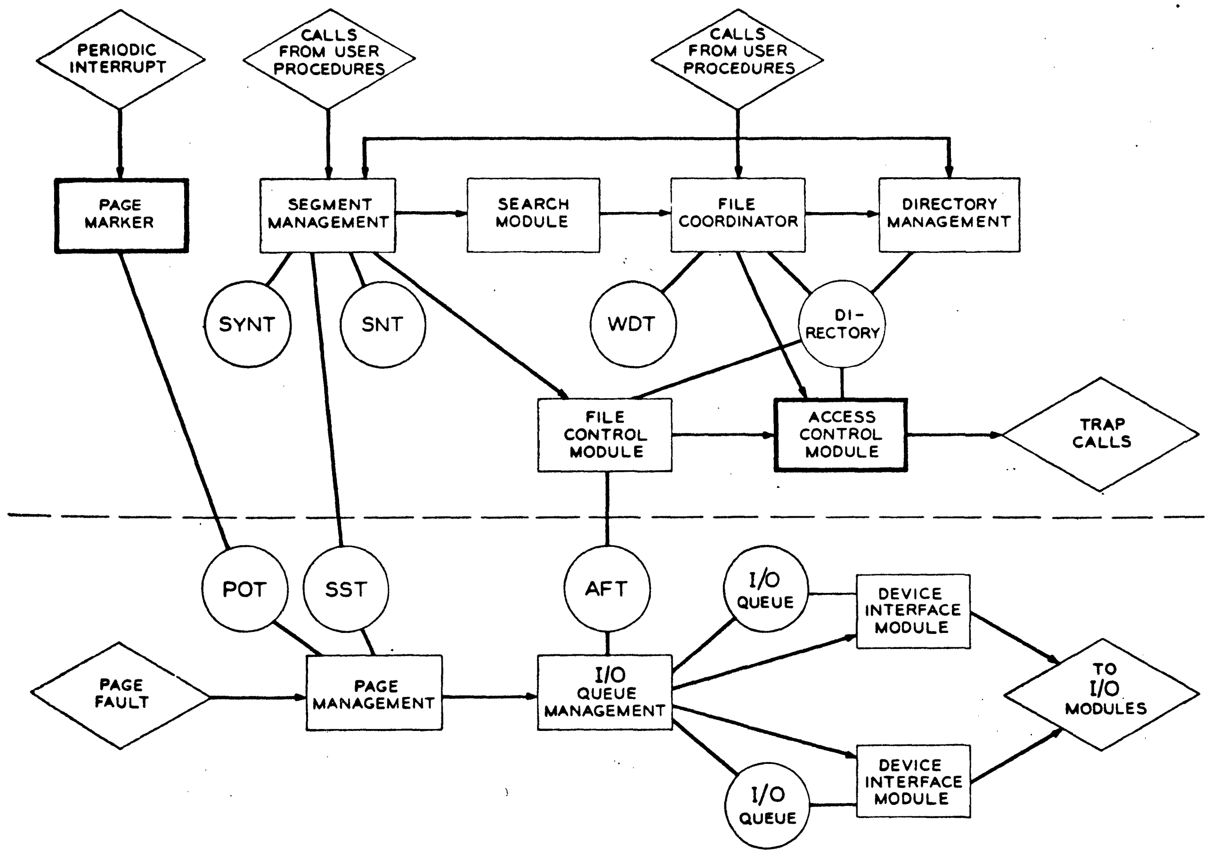


DIRECTORY MANAGEMENT

- SEARCHES FOR A SINGLE DIRECTORY BY TREE NAME
- MAY CALL SEGMENT MANAGEMENT TO GET SEGMENT NUMBER
- MAY BE RE—CALLED BY SEGMENT MANAGEMENT
- RECURSION MAY REACH TO ROOT OF TREE

FILE CONTROL MODULE

- OPENS FILES FOR SEGMENT MANAGEMENT
- MAKES ENTRY IN ACTIVE FILE TABLE (AFT)
- RETURN AFT POINTER
- GETS PERMISSION FROM ACCESS CONTROL MODULE
- MAY BLOCK PROCESS ON INCOMPATIBLE REQUEST

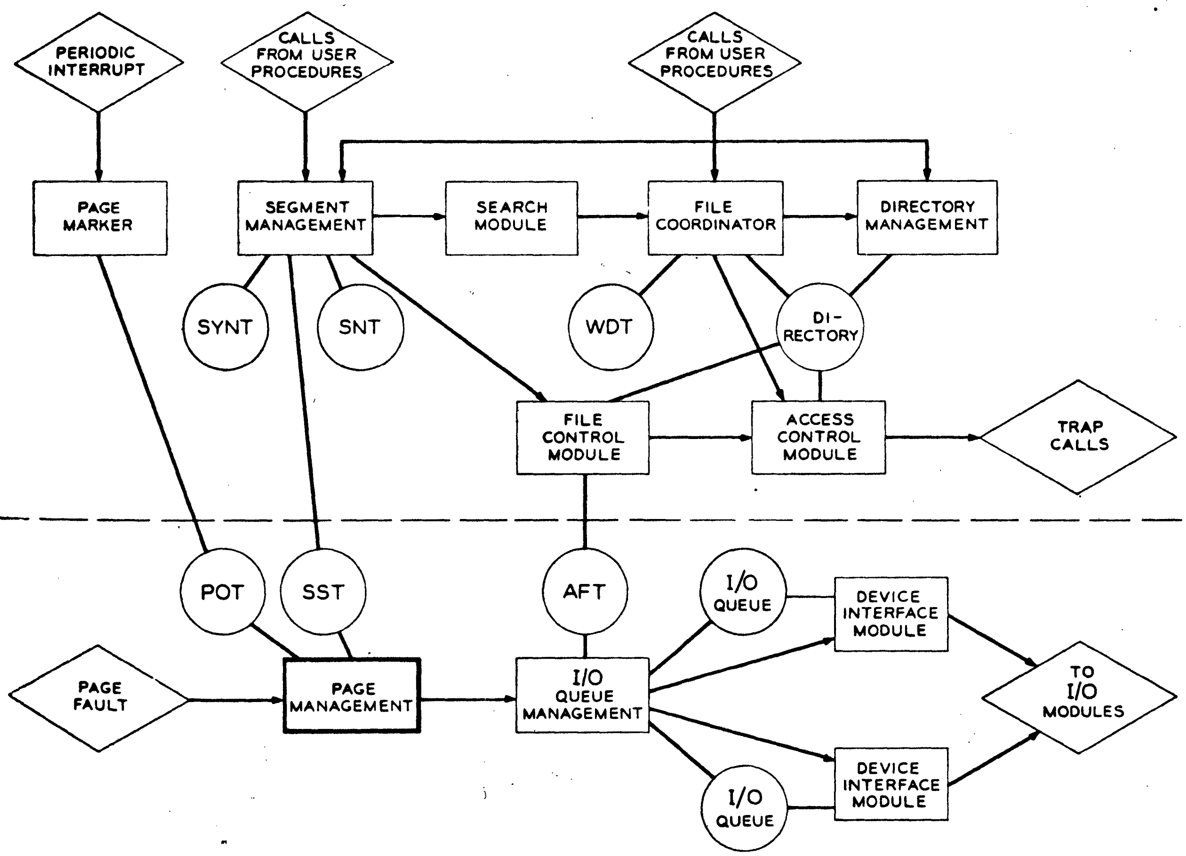


ACCESS CONTROL MODULE

- CHECKS DIRECTORY, RETURN EFFECTIVE MODE
- FOR TRAP MODE, PASSES CONTROL TO INDICATED PROCEDURE FOR EFFECTIVE MODE DETERMINATION

PAGE MARKER

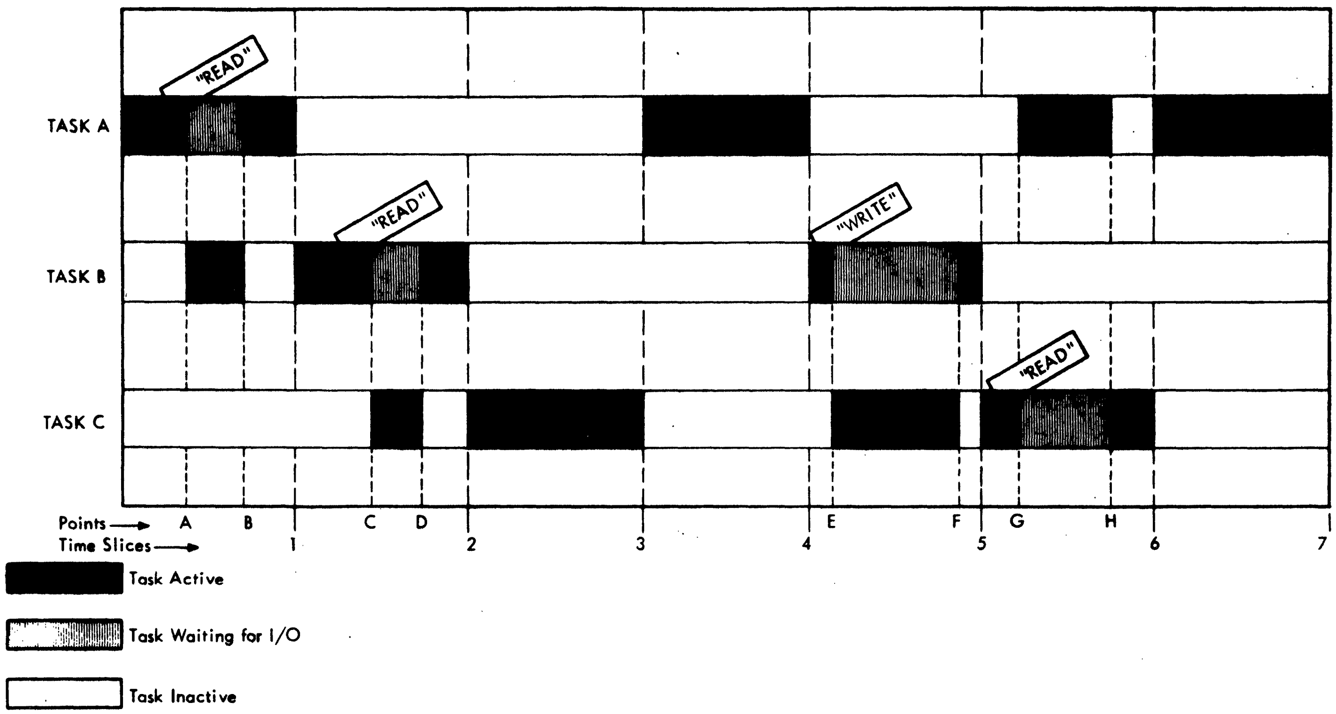
- PERIODICALLY INTERRUPTS
- RESETS PAGE USE BITS
- PUTS SELDOM—USED PAGE DATA IN PAGE OUT TABLE (POT)



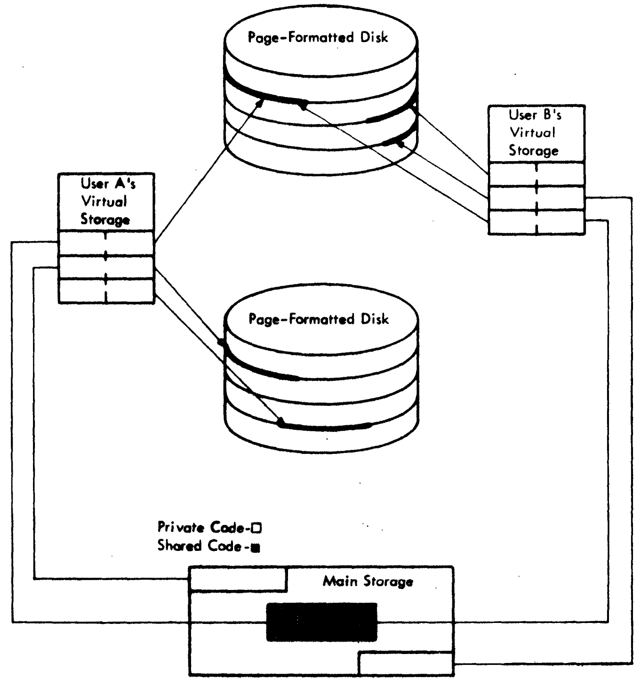
PAGE MANAGEMENT MODULE

- ENTERED BY MISSING PAGE FAULT
- ASSIGNS FREE PAGE FROM AVAILABLE SPACE OR POT
- FOR NEW PAGE, POINTER FROM PAGE TABLE TO SEGMENT STATUS TABLE USED TO GET POINTER TO ACTIVE FILE
- POINTER PASSED TO I/O QUEUE MANAGEMENT TO READ PAGE

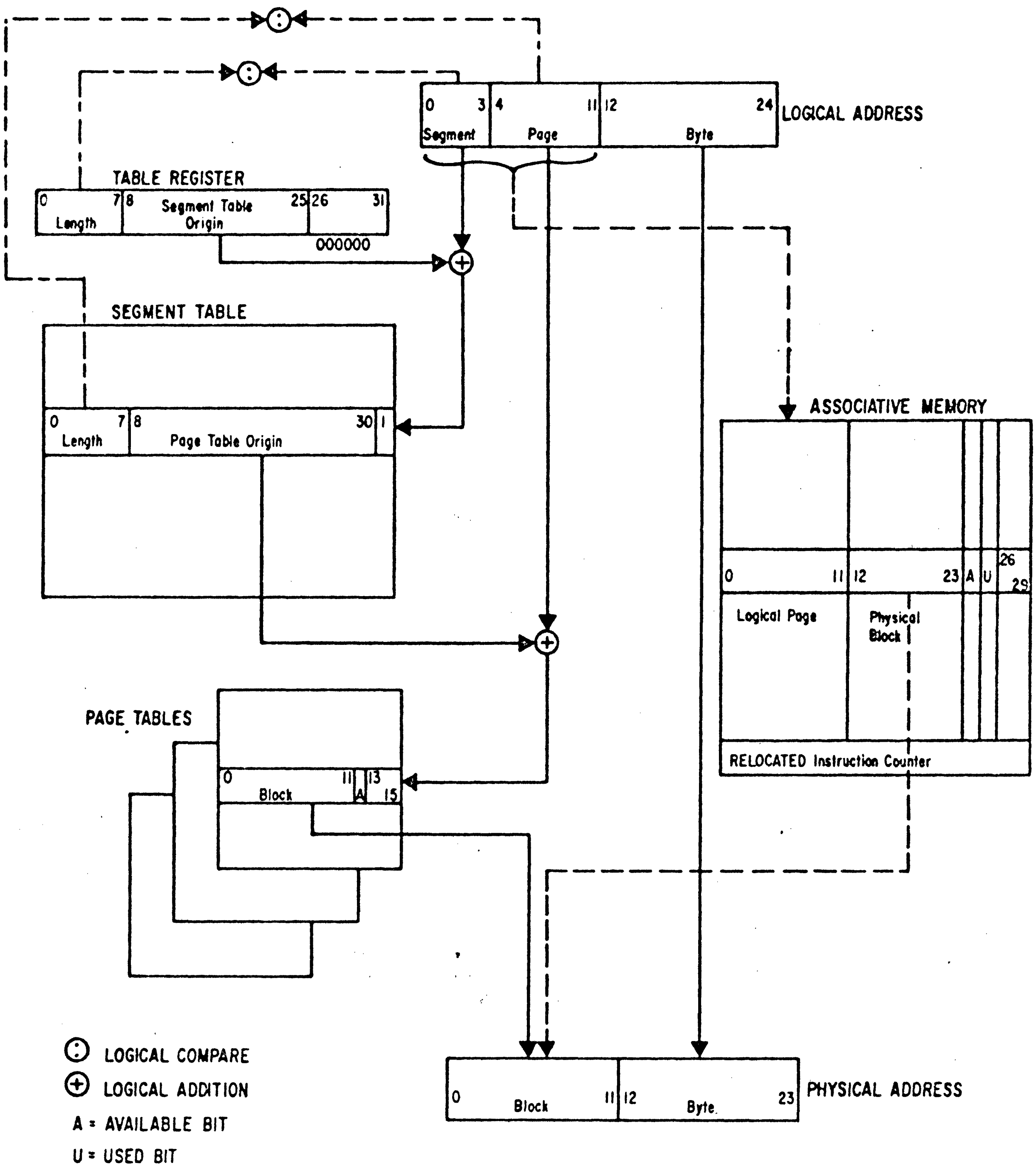
TIME SLICING AMONG THREE TASKS IN TSS/360



PRIVATE CODE AND SHARED CODE



DYNAMIC ADDRESS TRANSLATION



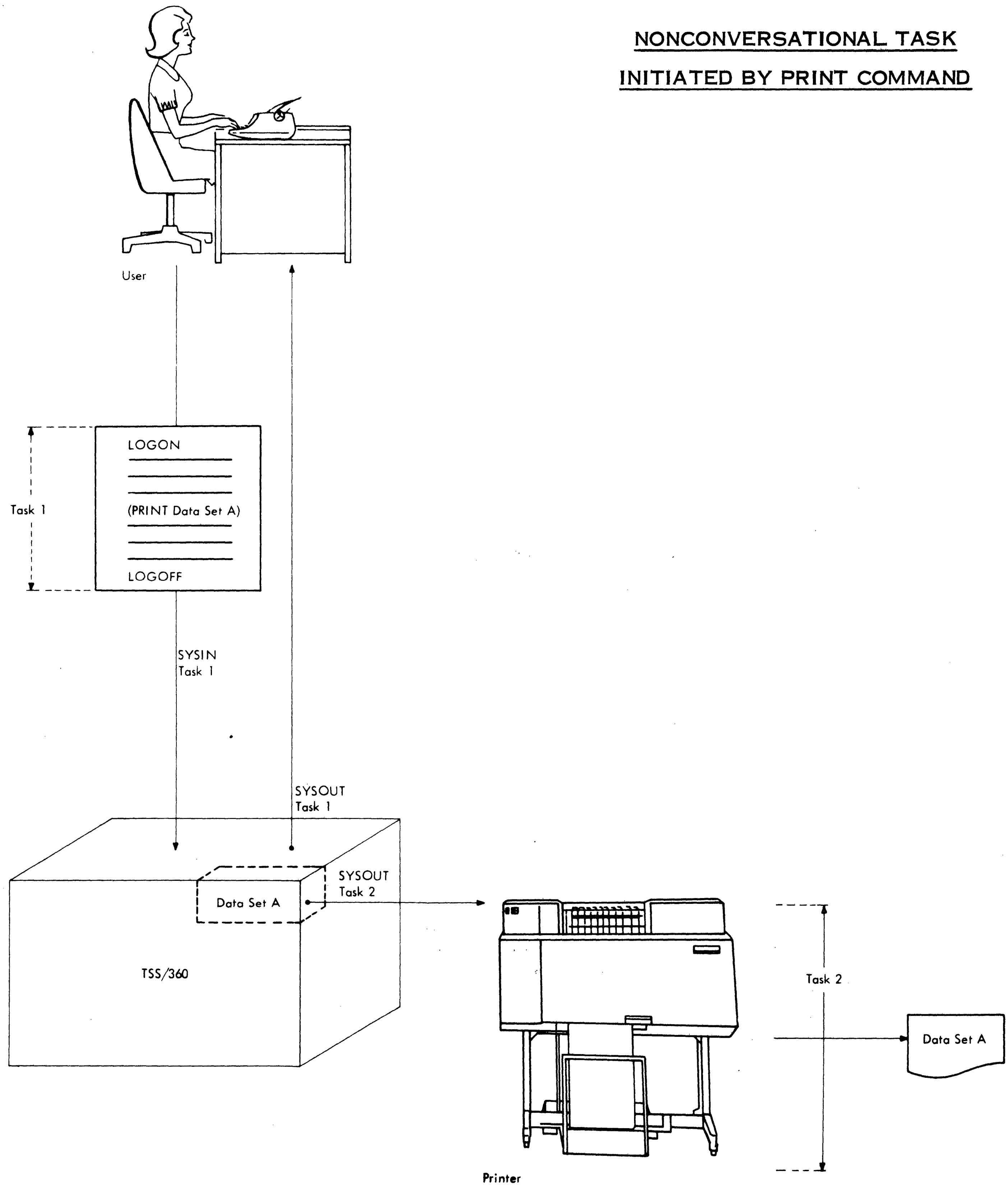
PROTECTION LEVELS

	RE-ENTERABLE?	STATE	USER'S VIRTUAL MEMORY?	RE-LOCATED?	PAGED?	PROTECTION BY
PROBLEM PROGRAMS FORTRAN, ASSEMBLER	?	PROBLEM	YES	YES	YES	NONE
SERVICE PROGRAMS ACCESS METHODS, COMMAND LANGUAGE	YES	SUPERVISOR	YES	YES	YES	PROTECT KEY
RESIDENT SUPERVISOR	NO	SUPERVISOR	NO	NO	NO	NOT IN VIRTUAL MEMORY
HARDWARE MICRO-PROGRAM	-	-	-	-	-	READ-ONLY

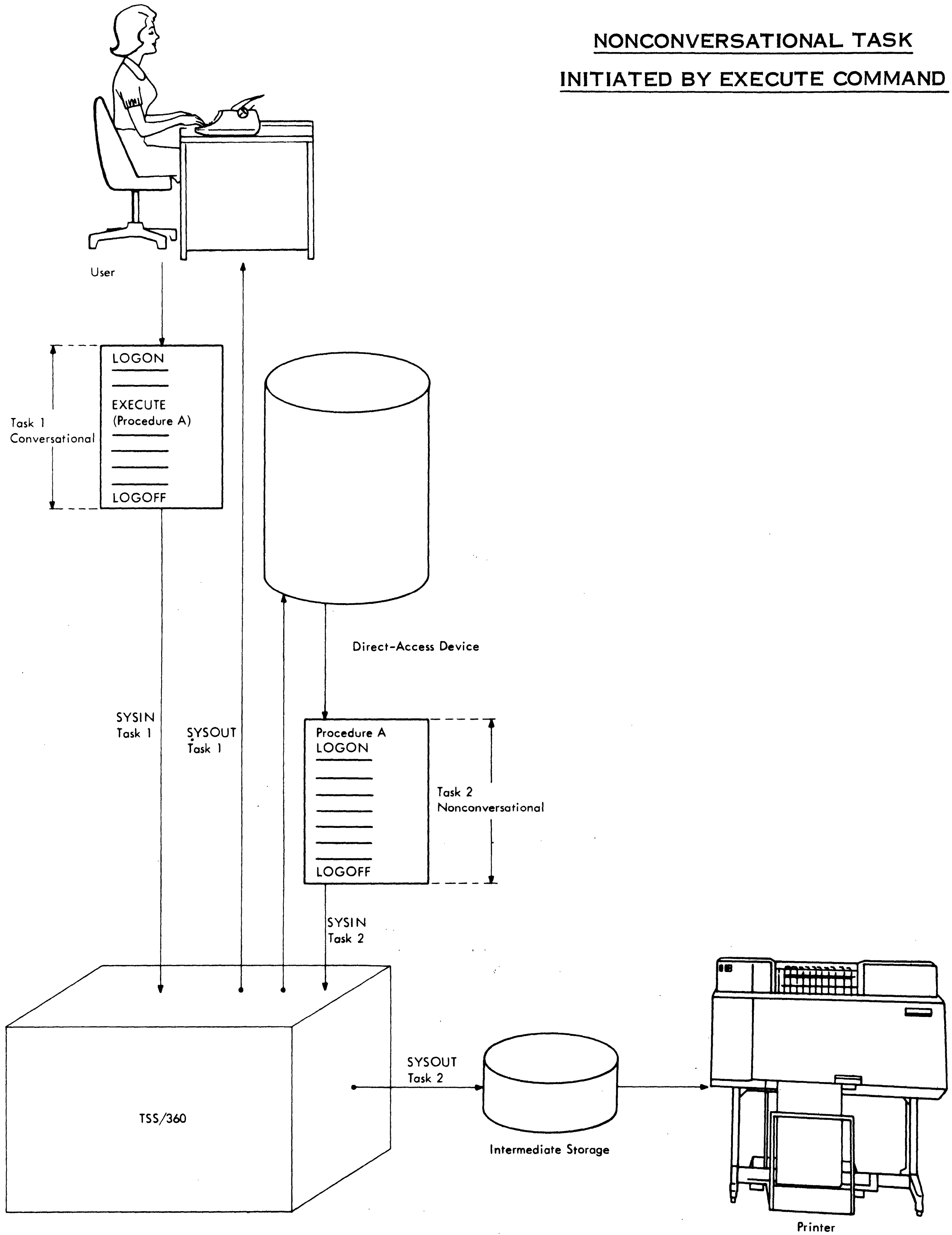
HOW CALLS WORK IN A BEGIN I/O OPERATION EXAMPLE

- SVC SUPERVISOR CALL INSTRUCTION (PROBLEM PROGRAMS → SERVICE PROGRAMS)
- IOCAL MACRO (SERVICE PROGRAMS → RESIDENT SUPERVISOR)
- SIO PRIVILEGED INSTRUCTION (RESIDENT SUPERVISOR → HARDWARE MICRO-PROGRAM)

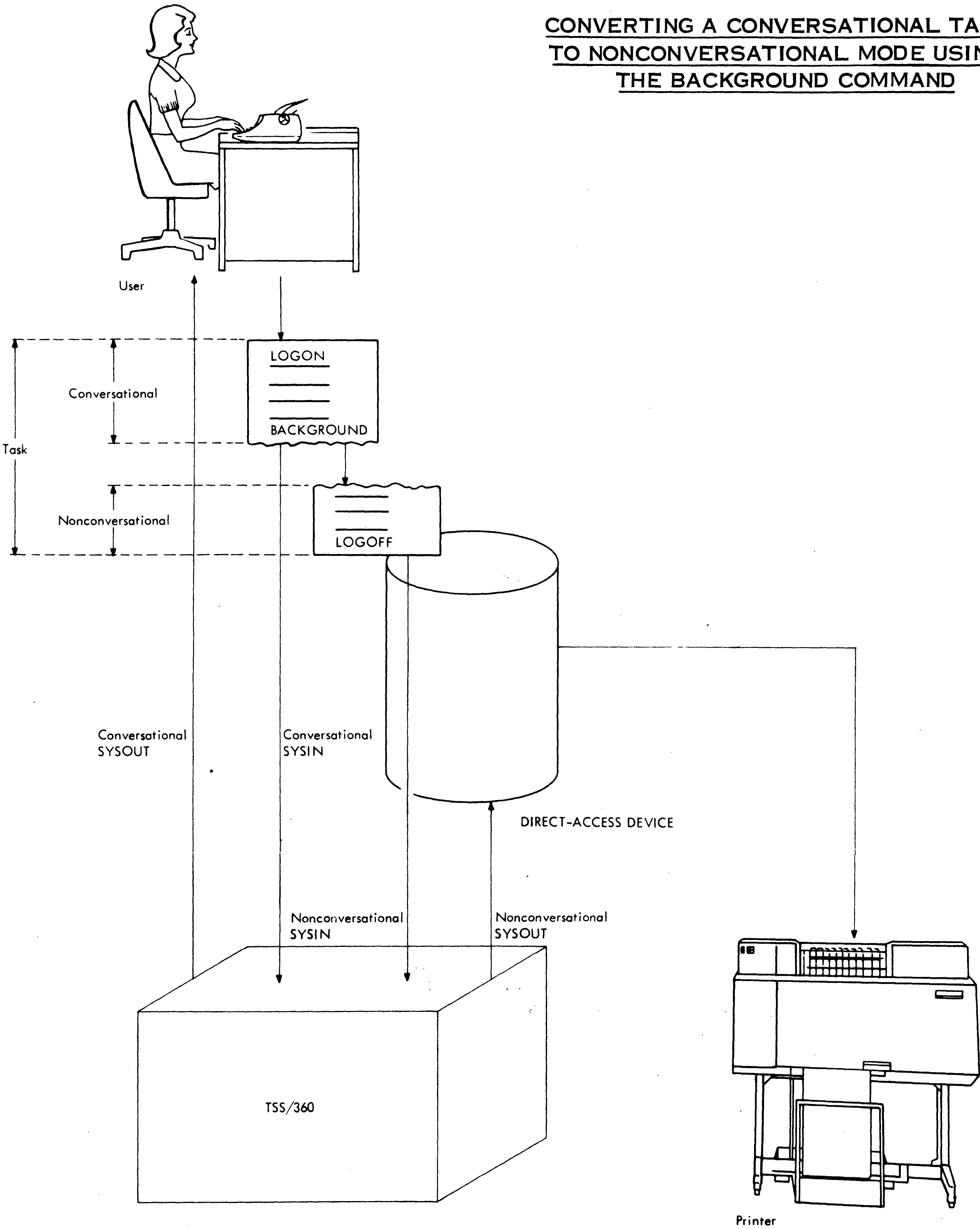
NONCONVERSATIONAL TASK INITIATED BY PRINT COMMAND



NONCONVERSATIONAL TASK INITIATED BY EXECUTE COMMAND



CONVERTING A CONVERSATIONAL TASK TO NONCONVERSATIONAL MODE USING THE BACKGROUND COMMAND



TIME SHARING SYSTEM/360 DATA MANAGEMENT FACILITIES

DATA MANAGEMENT FACILITIES

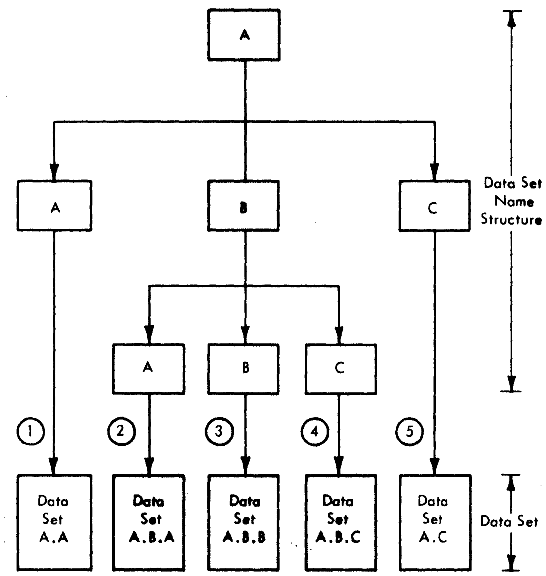
DATA SET MANAGEMENT

- Cataloging
  - System catalog
  - Cataloging facilities
    - (including CATALOG and DELETE commands)
- Sharing
  - PERMIT command
  - SHARE command
- Manipulation
  - MODIFY command
  - COPY DATA SET command
  - ERASE command
- Definition
  - DEFINE DATA command
  - CALL DATA DEFINITION command
  - RELEASE command
  - SECURE command

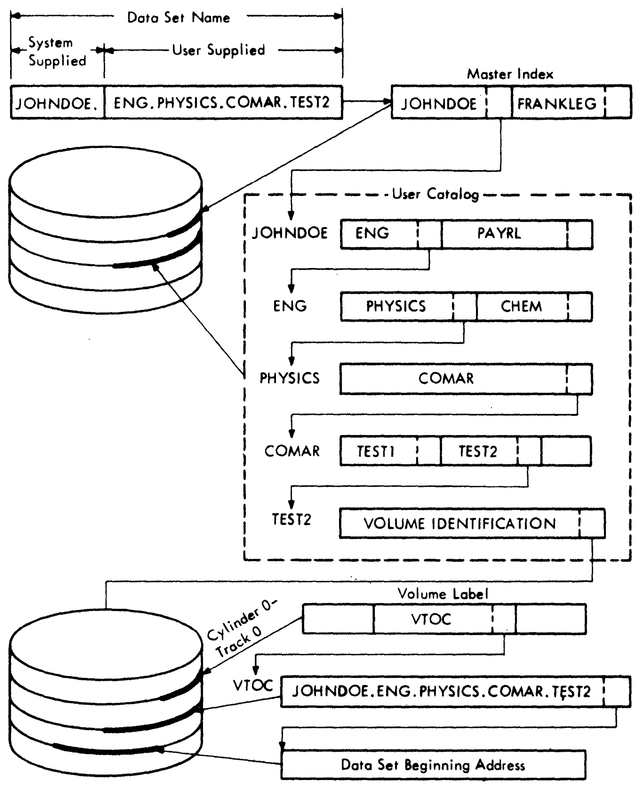
PROBLEM PROGRAM I/O

- Prestore input data in system
  - DATA command -- by user
  - READ CARDS command -- by operator
  - READ TAPE command -- by operator
- Obtain input data and generate output data during program execution
  - Conventional I/O facilities, using I/O
    - statements in source program
  - Dynamic I/O facilities, using program
    - checkout commands and statements, and special source language statements
- Transfer data from system storage to standard I/O devices
  - PRINT command
  - PUNCH command
  - WRITE TAPE command

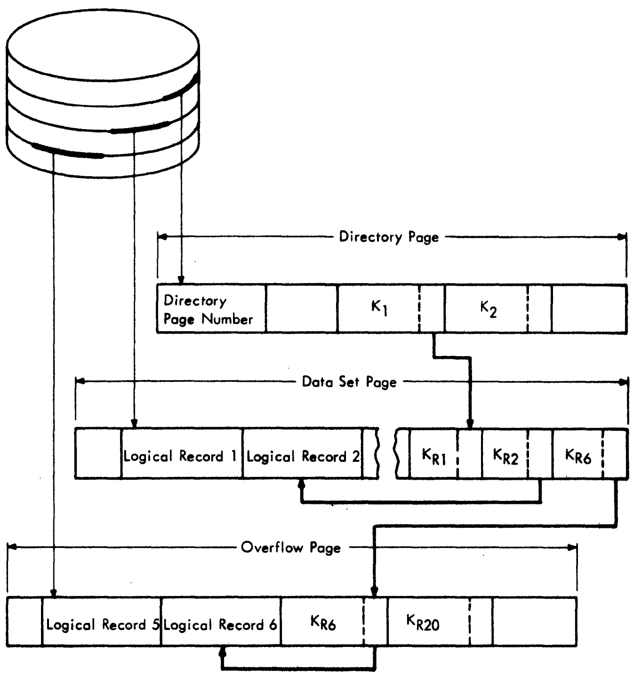
FULLY AND PARTIALLY QUALIFIED NAMES



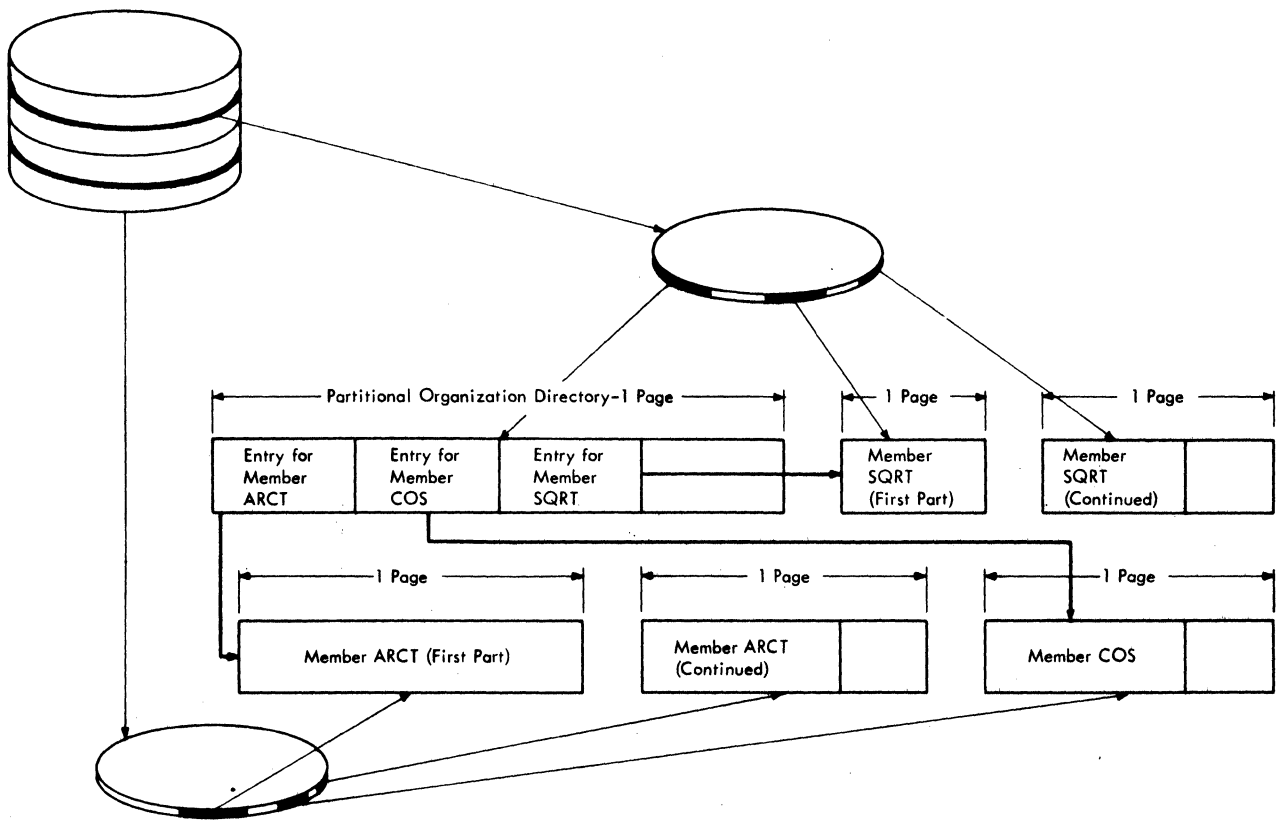
SYSTEM CATALOG CONCEPT



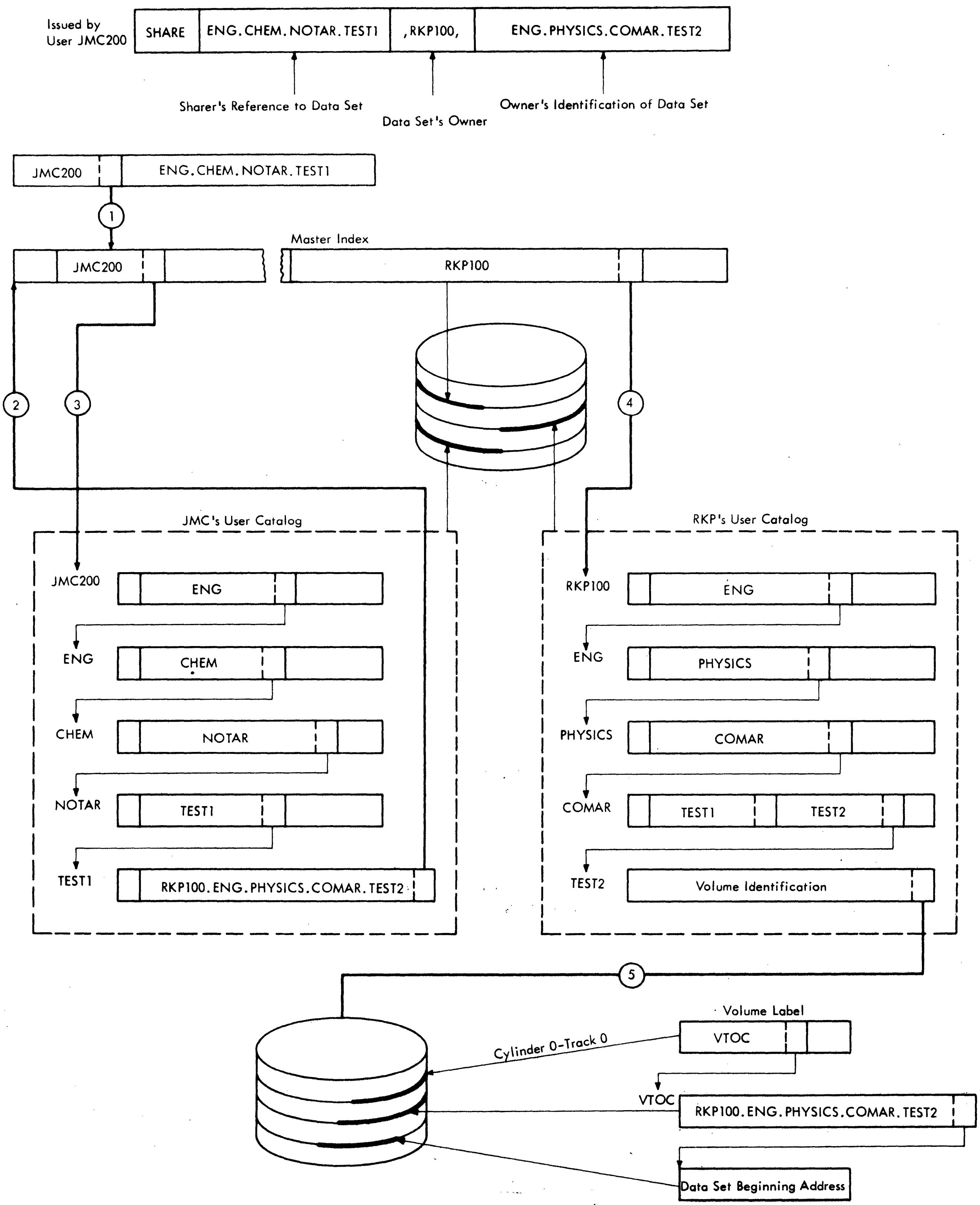
TYPICAL VIRTUAL INDEX SEQUENTIAL DATA SET



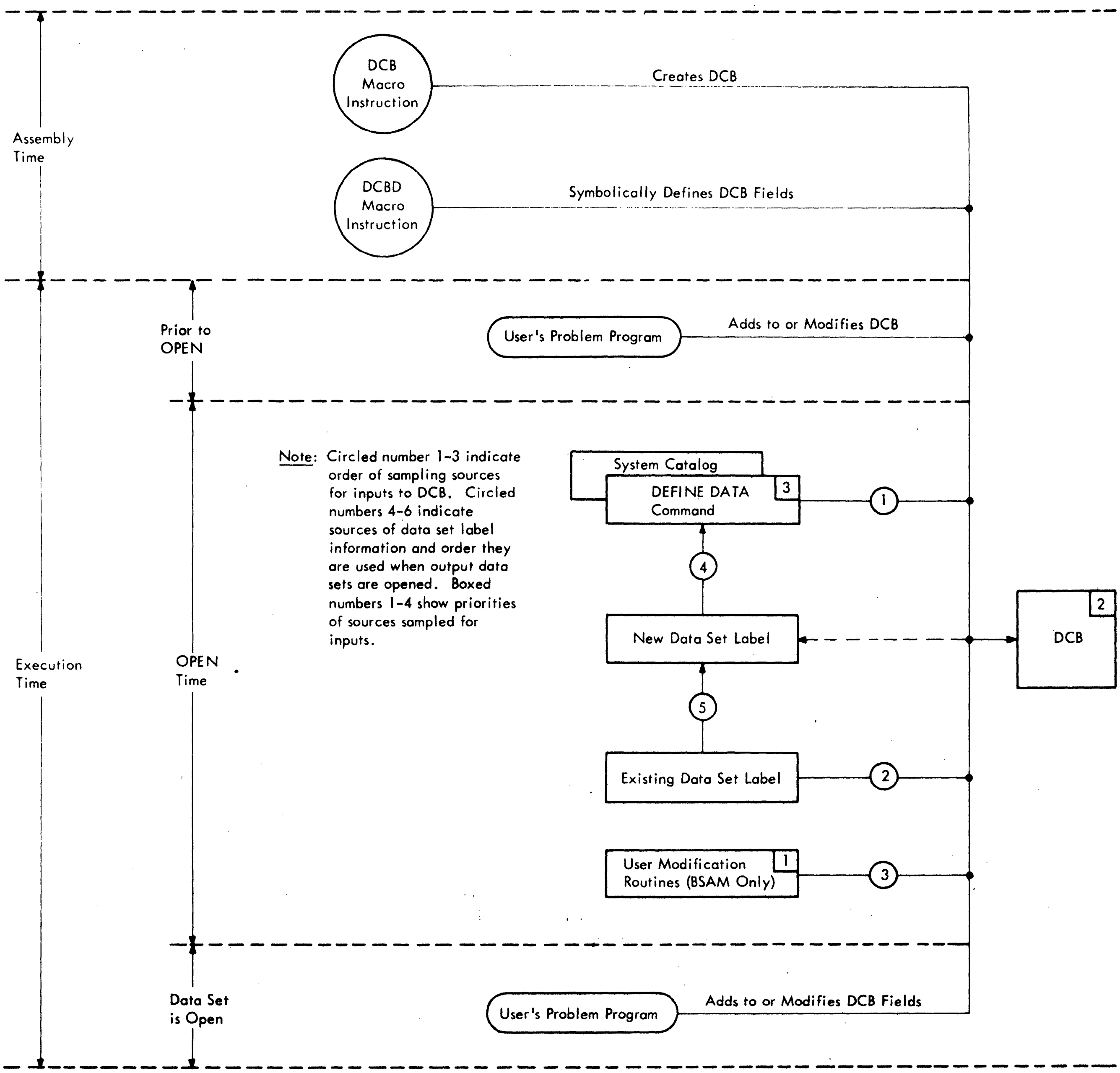
VIRTUAL PARTITIONED DATA SET



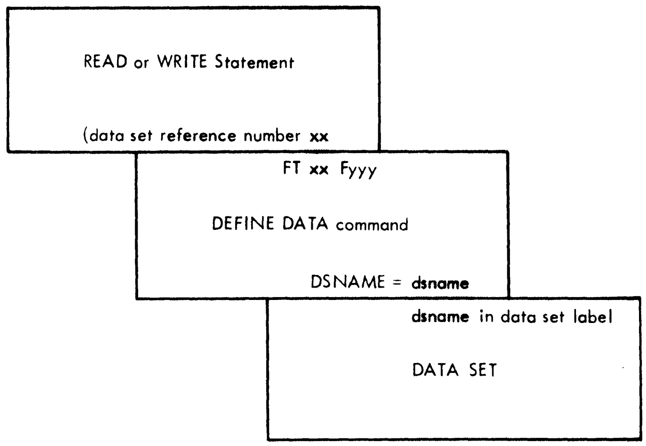
SHARING OF CATALOGED DATA SETS



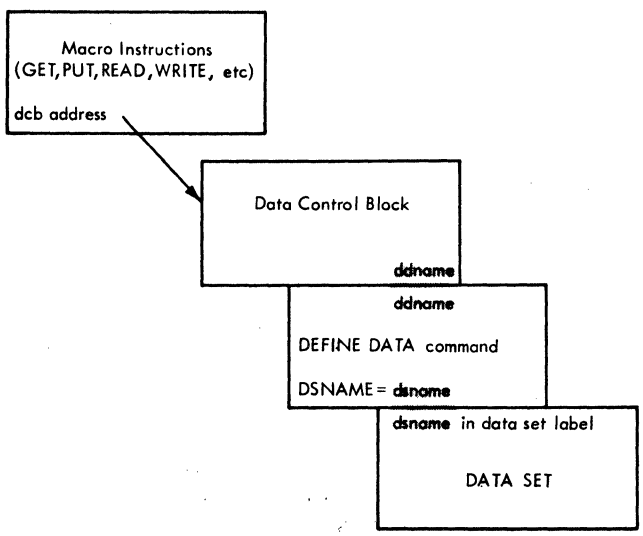
FLOW OF INFORMATION TO AND FROM A DATA CONTROL BLOCK



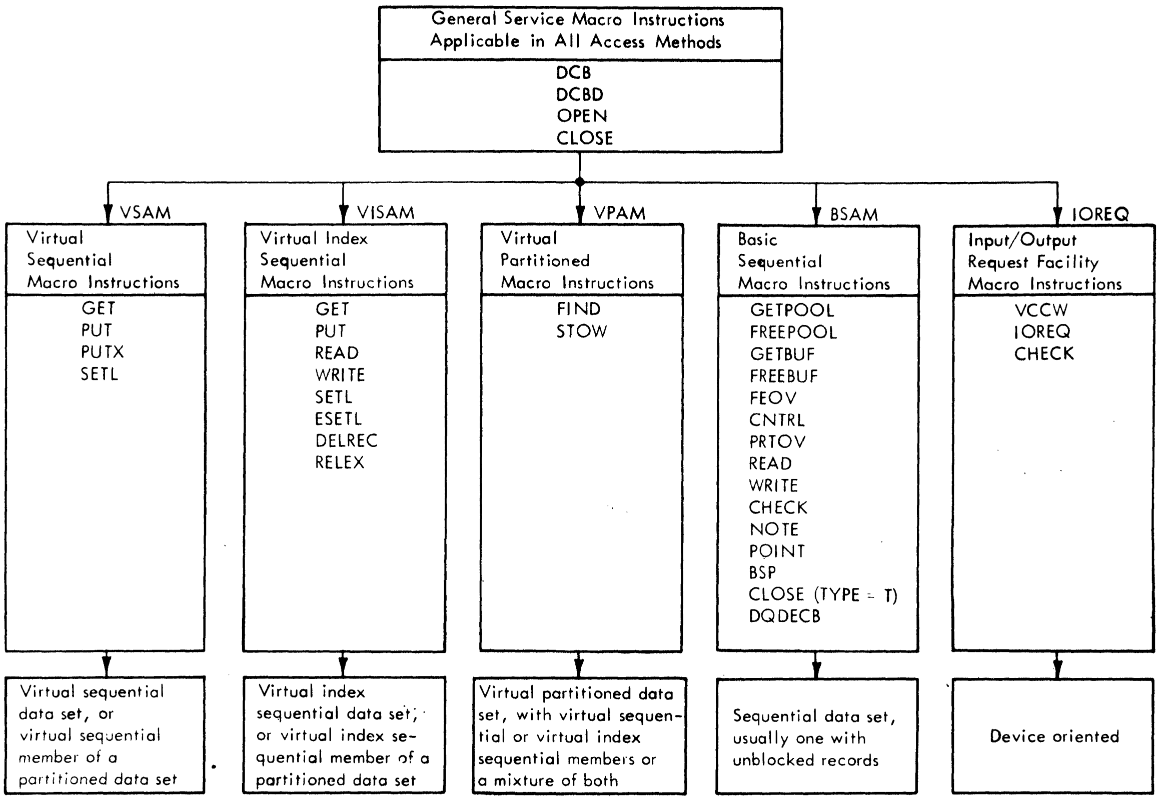
DATA SET IDENTIFICATION, FORTRAN-WRITTEN PROGRAMS



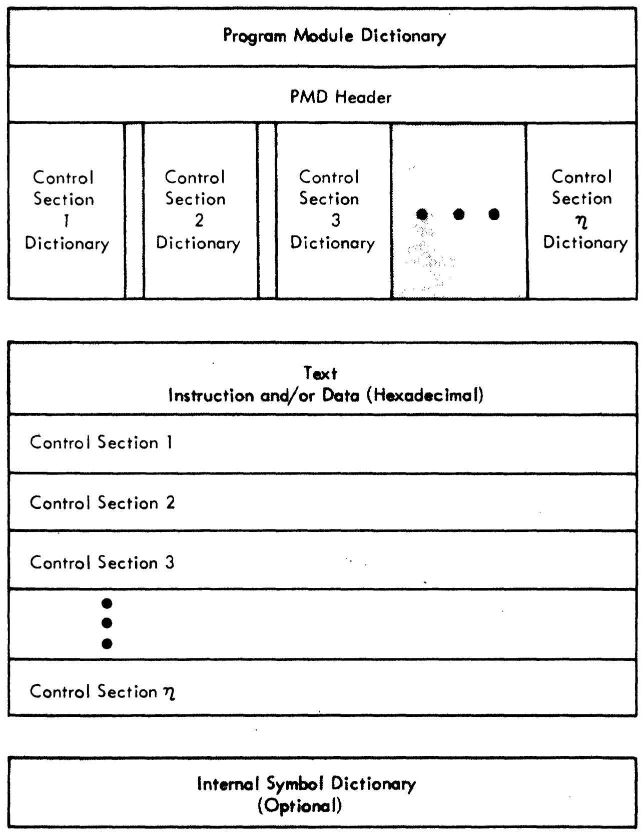
DATA SET IDENTIFICATION, ASSEMBLER LANGUAGE PROGRAM



SUMMARY OF DATA MANAGEMENT SYSTEM MACRO INSTRUCTIONS AND DATA SET ORGANIZATIONS



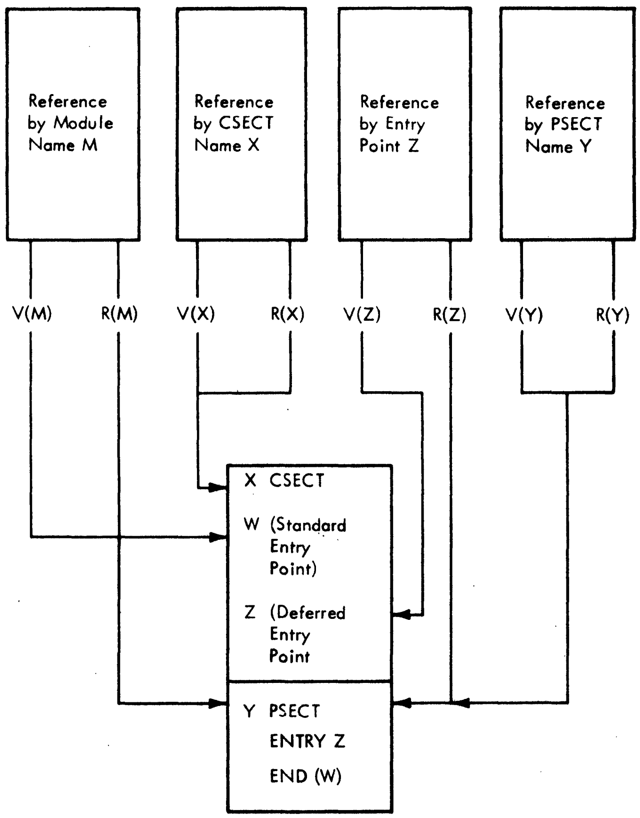
FORMAT OF AN OBJECT PROGRAM MODULE



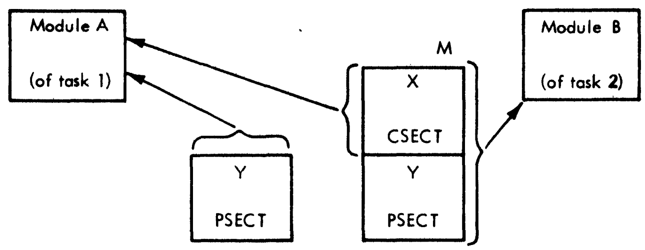
ATTRIBUTES OF CONTROL SECTIONS

- READONLY
- PUBLIC
- PSECT
- COM
- PRVLGD
- VARIABLE

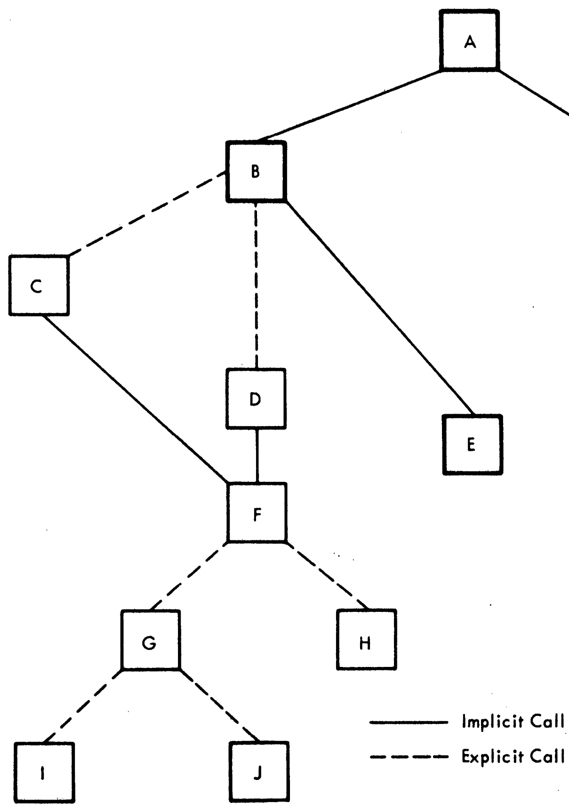
V– AND R–VALUES OF EXTERNAL SYMBOLS



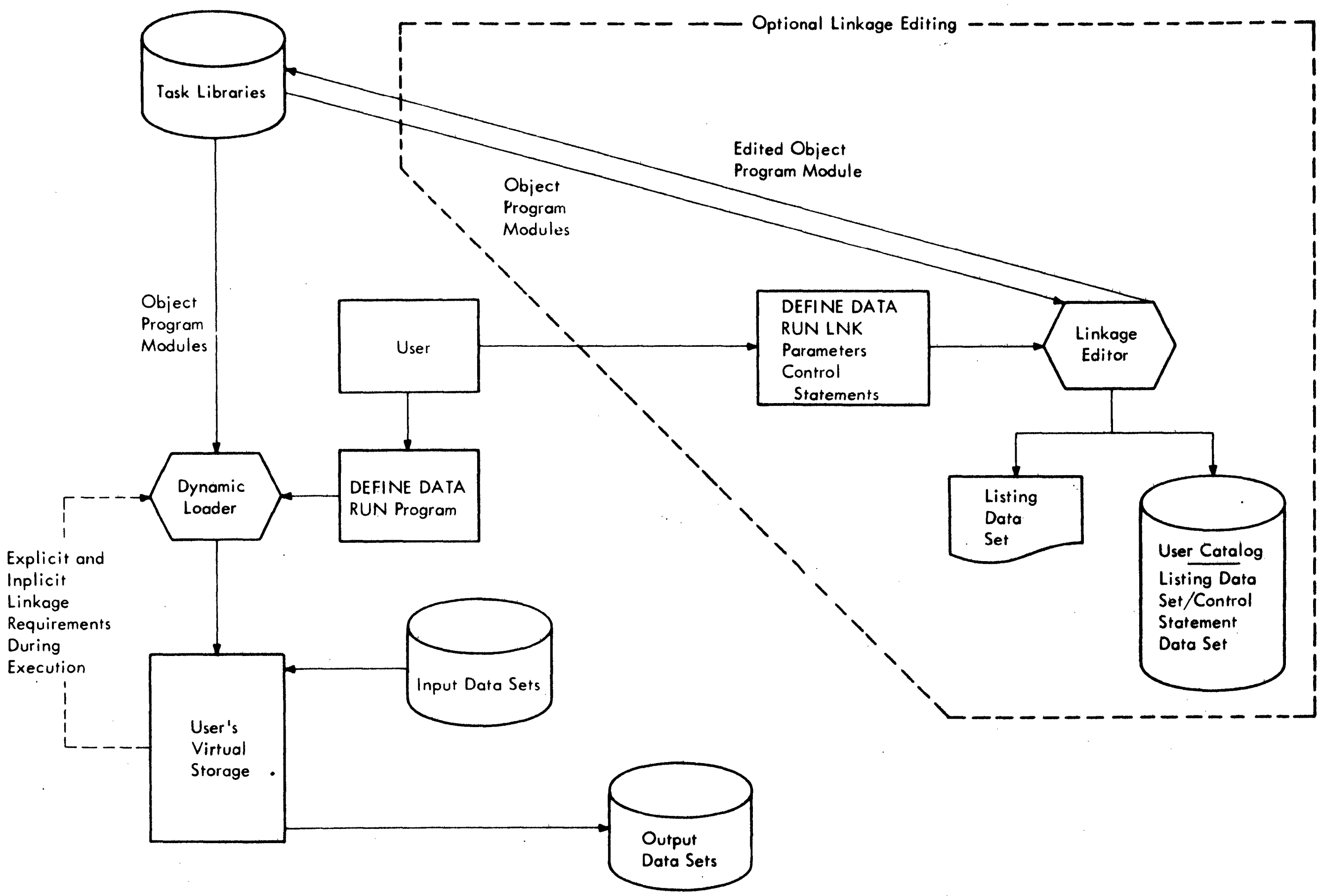
SHARING A MODULE



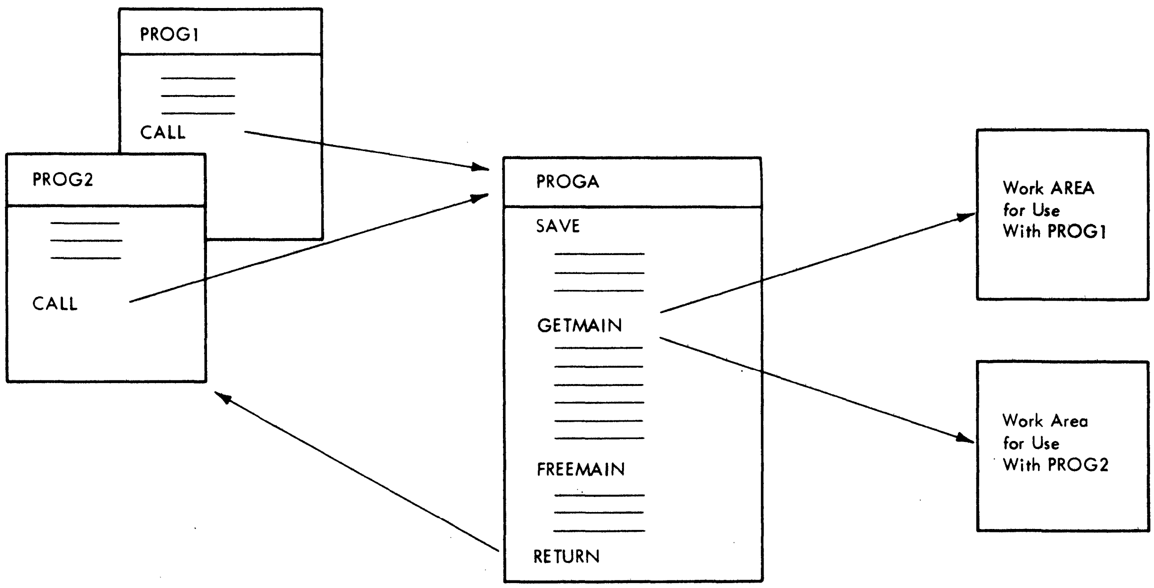
PROGRAM WITH IMPLICIT AND EXPLICIT LINKAGES



OBJECT PROGRAM MODULE COMBINATION



A REENTERABLE ROUTINE THAT REQUESTS ITS OWN TEMPORARY STORAGE



▶ INSIGHTS INTO

- MACHINE ORGANIZATION
- PROGRAMMING LANGUAGES
- PROGRAMMING SYSTEMS

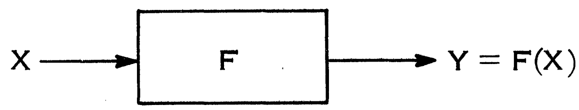
▶ BY MEANS OF

- CONCEPTUAL FRAMEWORK
- CASE STUDIES

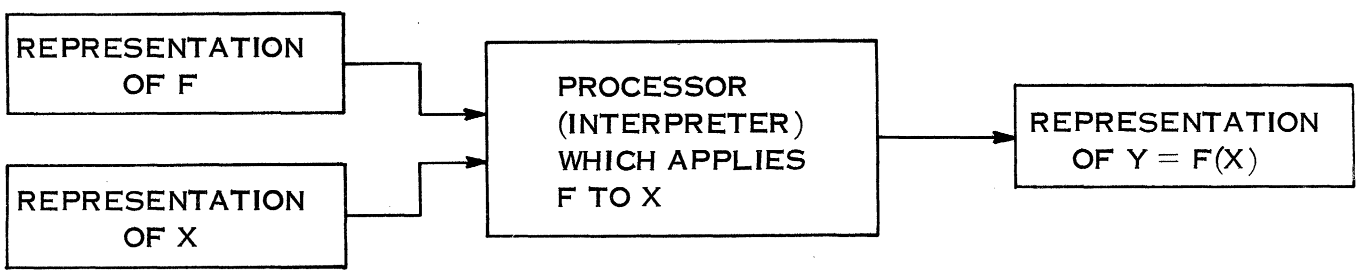
BASIC DEFINITIONS

ALGORITHM	— A RULE FOR COMPUTING THE SOLUTION TO A PROBLEM OR CLASS OF PROBLEMS IN A FINITE NUMBER OF STEPS.
PROGRAM	— REPRESENTATION OF AN ALGORITHM IN SOME PROGRAMMING LANGUAGE.
COMPUTER	— MECHANICAL DEVICE FOR PROGRAM EXECUTION.
COMPILER (TRANSLATOR)	— PROGRAM FOR TRANSLATING FROM ONE PROGRAMMING LANGUAGE TO ANOTHER.
SOURCE LANGUAGE	— PROGRAMMING LANGUAGE IN WHICH PROGRAMS ARE SPECIFIED BY THE PROGRAMMER <u>OR</u> PROGRAMMING LANGUAGE WHICH SERVES AS INPUT TO A COMPILER.
TARGET LANGUAGE	— PROGRAMMING LANGUAGE WHICH SERVES AS OUTPUT FROM A COMPILER.
ASSEMBLER	— SPECIAL CASE OF A COMPILER WHEN TRANSLATION FROM THE SOURCE LANGUAGE TO THE TARGET LANGUAGE INVOLVED MAINLY TRANSLITERATION.
PROGRAMMING SYSTEM	— A SET OF PROGRAMS FOR A COMPUTER WHICH ALLOWS SEQUENCES OF USER PROGRAMS TO BE EXECUTED WITHOUT MANUAL INTERVENTION. THE TERM PROGRAMMING SYSTEM SOMETIMES DENOTES THE HARDWARE OF THE COMPUTER SYSTEM TOGETHER WITH THE SET OF PROGRAMS THAT CONSTITUTE THE INTERFACE BETWEEN THE HARDWARE AND THE USER.

CONCEPTS OF A FUNCTION



MATHEMATICAL CONCEPT OF A FUNCTION

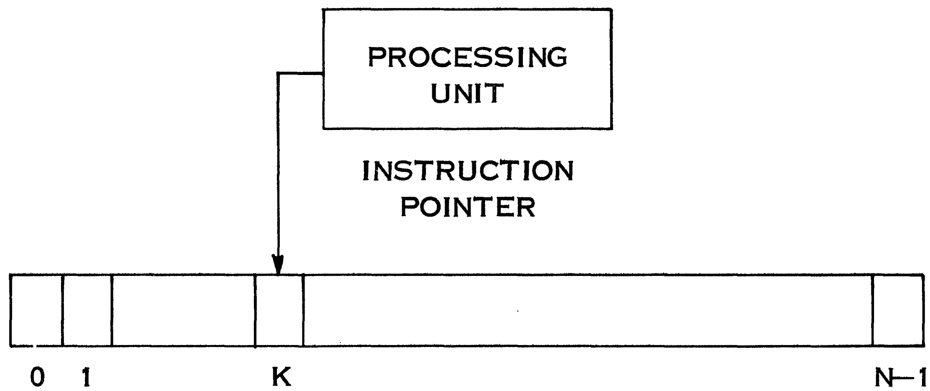


COMPUTATIONAL CONCEPT OF A FUNCTION

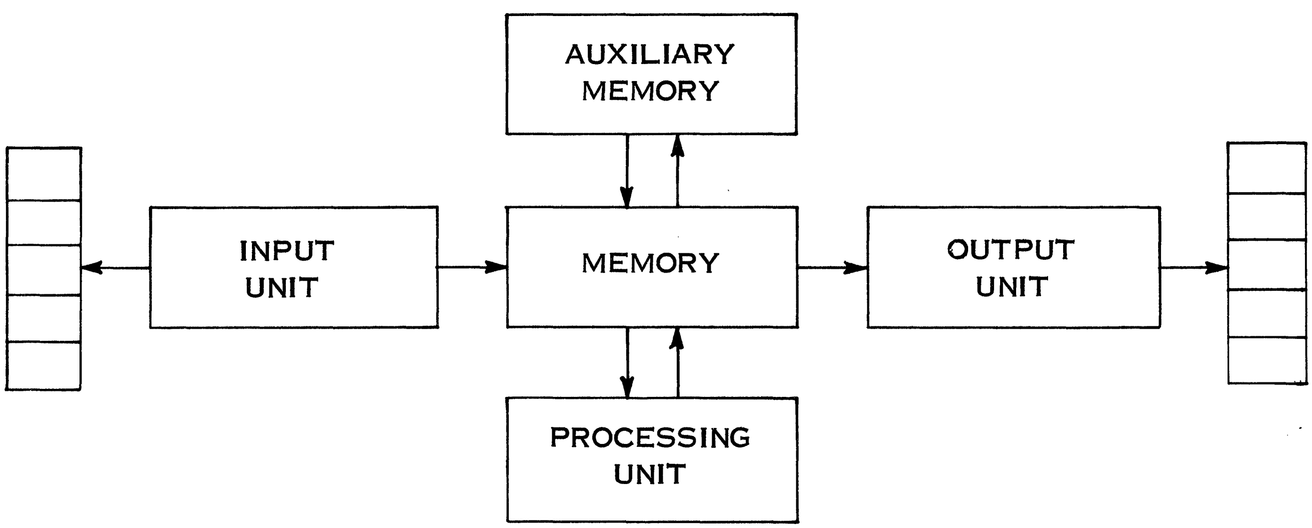
REPRESENTATIONS OF A FUNCTION

A REPRESENTATION OF A FUNCTION F TOGETHER WITH ITS DATA X CONSTITUTES AN INFORMATION STRUCTURE. A FINITE COMPUTATION CAN BE CHARACTERIZED BY AN INITIAL INFORMATION STRUCTURE I_O, AND BY THE SEQUENCE OF INFORMATION STRUCTURES I_1; $I_2 \ldots I_N$ GENERATED FROM I_O BY THE EXECUTION OF INSTRUCTIONS. I_O IS SAID TO BE THE INITIAL REPRESENTATION AND I_N IS SAID TO BE THE FINAL REPRESENTATION. AN INFORMATION STRUCTURE I_J WHICH CAPTURES THE COMPLETE STATE OF THE COMPUTATION AT A GIVEN POINT IN ITS LIFETIME IS SAID TO BE AN INSTANTANEOUS DESCRIPTION.

FUNCTIONAL COMPONENTS OF A COMPUTER



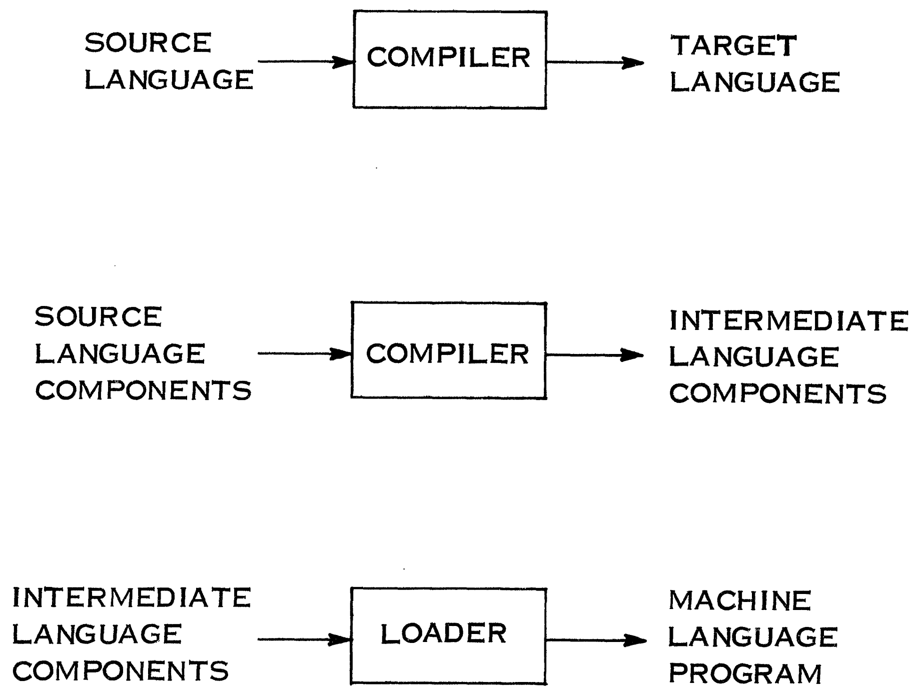
A SIMPLE COMPUTER



FUNCTIONAL COMPONENTS OF A COMPUTER

5638

TRANSLATION, COMPILATION AND LOADING



5637

REQUIRED PROPERTIES OF INTERMEDIATE LANGUAGE (COMPILER)

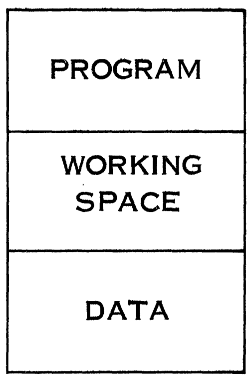
- PROGRAM REPRESENTATION INDEPENDENT OF MACHINE STORAGE LOCATIONS.

- PROVISION FOR CROSS–REFERENCING BETWEEN PROGRAM COMPONENTS.

- TRANSLATION TO PURE MACHINE LANGUAGE AS EFFICIENT AS POSSIBLE.

PROGRAM STRUCTURE FOR FORTRAN

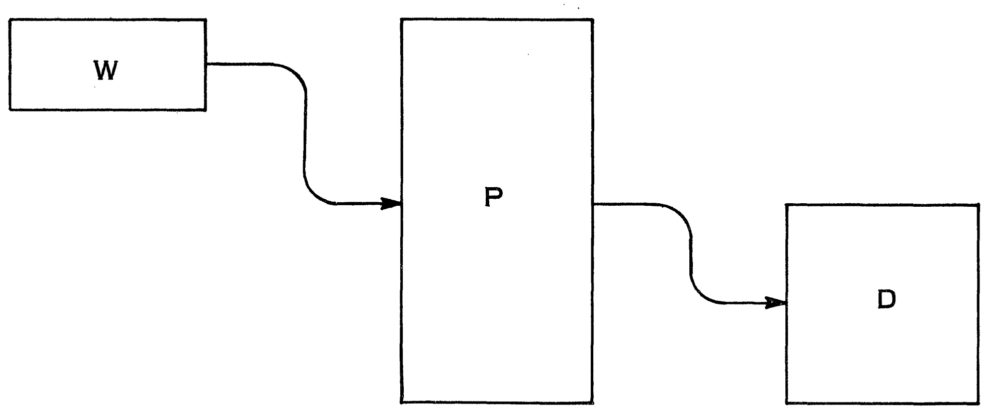
- MAIN PROGRAM
- SUBROUTINES
- COMMON DATA BLOCKS



PRINCIPAL COMPONENTS OF A FORTRAN PROGRAM UNIT

FUNCTIONAL COMPONENTS OF A PROGRAM

- A PROGRAM PART P WHICH SPECIFIES THE PROGRAM TO BE EXECUTED.
- A DATA PART D WHICH SPECIFIES THE DATA FOR THE PROGRAM.
- A STATEWORD W WHICH CONTAINS INFORMATION IN THE PROCESSING UNIT OF AN ACTUAL COMPUTER, INCLUDING AN INSTRUCTION POINTER WHICH POINTS TO THE NEXT STATEMENT OR SUBEXPRESSION TO BE EXECUTED.

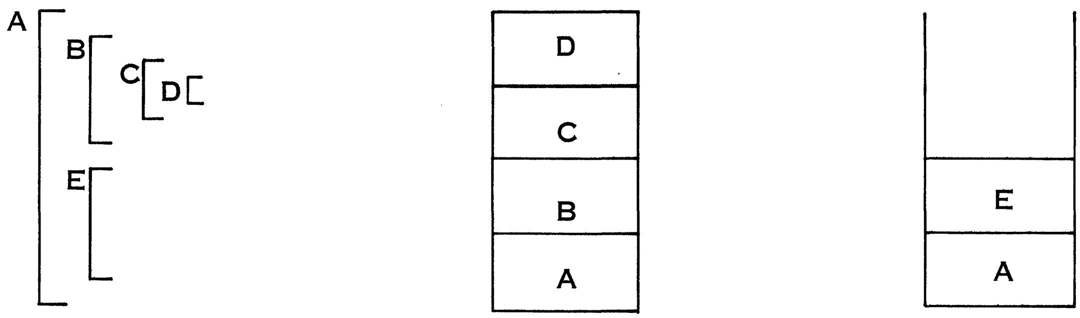


LOGICAL PROGRAM STRUCTURE

DEFINITIONS OF FUNCTIONS

- ACTIVATION RECORD
- REENTRANT FUNCTIONS
- RECURSIVE FUNCTIONS

SEQUENCE OF FUNCTIONAL COMPONENTS



PROGRAM STRUCTURE

STACK WHEN EXECUTION IS IN D

STACK WHEN EXECUTION IS IN E

PROGRAM STRUCTURE AND ACTIVATION RECORD STACK

PROGRAM EXECUTION

- LOGICAL STRUCTURE
- PHYSICAL STRUCTURE
- MACHINE ORGANIZATION

5629

COMMUNICATION BETWEEN FUNCTION MODULES

- SYMBOLIC CROSS REFERENCES
- TRANSFER VECTORS
- LOAD TIME LINKAGE
- ONE AND TWO-STAGE INDIRECT ADDRESSING
- INCREMENTAL LINKAGE

ONE AND TWO-STAGE INDIRECT ADDRESSING

USE AND DEFINITION TABLES FOR PROGRAMS IN THE INTERMEDIATE LANAGUAGE.

EXTERNALLY DEFINED SYMBOL		
USE 1	USE 2	

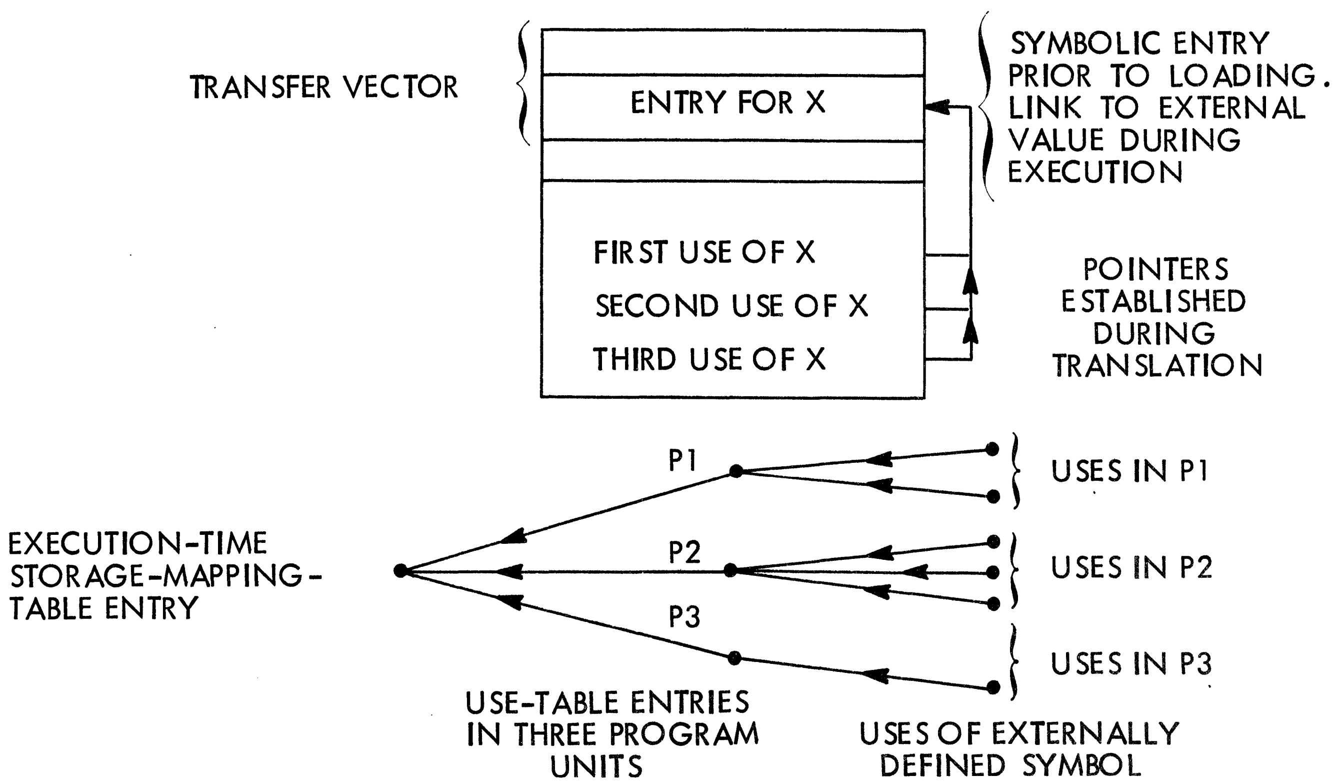
USE-TABLE ENTRY

SYMBOL BEING DEFINED
SYMBOL DEFINITION

DEFINITION-TABLE ENTRY

USE TABLE
DEFINITION TABLE
BODY OF PROGRAM UNIT

INDIRECT ADDRESSING OF STORAGE-MAPPING TABLE



STARTING POINT FOR THE STUDY OF PROGRAMMING

ALGORITHMS

COMPUTERS

INFORMATION STRUCTURES

COMPUTER SCIENCE CAN BE DEFINED AS THE STUDY OF REPRESENTATION AND TRANSFORMATION OF INFORMATION STRUCTURES.

INFORMATION STRUCTURES

ALPHABET T

INFORMATION STRUCTURE OVER I IS A SYMBOL STRING OVER T

SUBSTRUCTURE IMPOSED ON STRINGS BY A GRAMMAR

BEGIN REAL X; X: = 3 + 4 x 5 END

DECLARATION

EXPRESSION

STATEMENT

BLOCK

PROGRAMMING LANGUAGE - SET OF INFORMATION STRUCTURES

SYNTAX - SPECIFIES REPRESENTATION

SEMANTICS - SPECIFIES TRANSFORMATION

INFORMATION STRUCTURE MODELS

(I,F) I is set of information structures

F is set of transformations

I - syntactic component - specified by syntax

F - semantic component - specified by semantics

computation $I_o \xrightarrow{f} I_1 \xrightarrow{f} I_2 \; \ldots \xrightarrow{f} I_n$

$I_o \in I$ initial representation

I_i intermediate representations - instantaneous descriptions

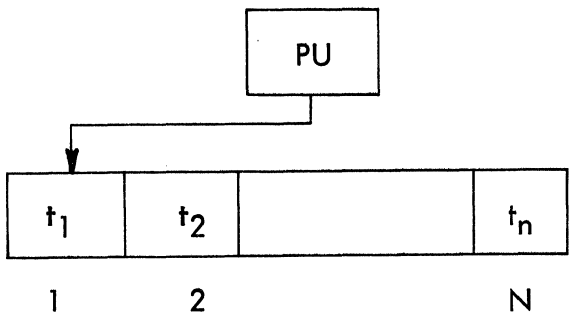
I_n final representation - no elements of f are applicable

Closure of I - set of all information structures which can be generated from I by finite sequences of f.

INFORMATION STRUCTURE MODEL FOR COMPUTERS

STORAGE STRUCTURES

PRIMITIVE INSTRUCTIONS



Principal information components

Processing unit component PU

Memory component M

Instruction pointer component PTR

Syntax: I ⟶ PU M PTR

PU ⟶ AC MQ BITS

etc

Semantics: Specify instructions in terms of which information fields they transform.

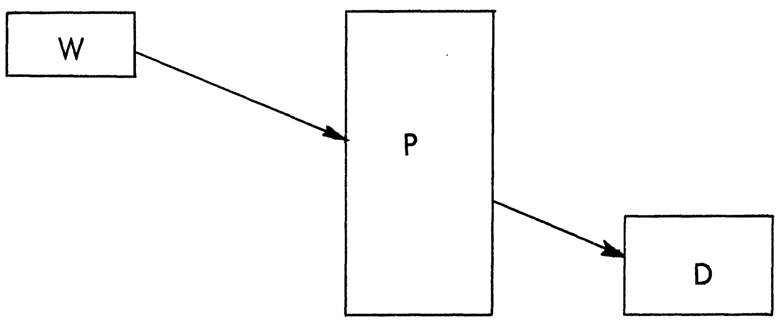
Recognition Phase

Transformation Phase

Interpretation step: <u>if</u> p_1 <u>then</u> A_1 <u>else</u> <u>if</u> p_2 <u>then</u> A_2 . . . <u>else</u> <u>if</u> p_n <u>then</u> A_n.

INFORMATION STRUCTURE MODEL FOR PROGRAMMING LANGUAGES

Stateword Component	W
Program Component	P
Data Component	D



W component is usually of fixed size

P consists of interacting function modules

reentrant function modules

Programming languages may be characterized by the structure of their D component.

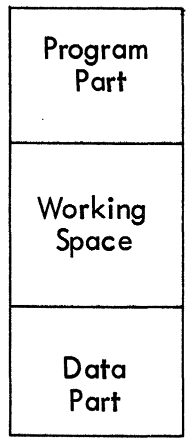
FORTRAN - All information fields of the D component are determined prior to execution.

ALGOL - The D component is a stack with respect to creation and deletion of information structures.

List Processing Languages - More flexible creation and deletion.

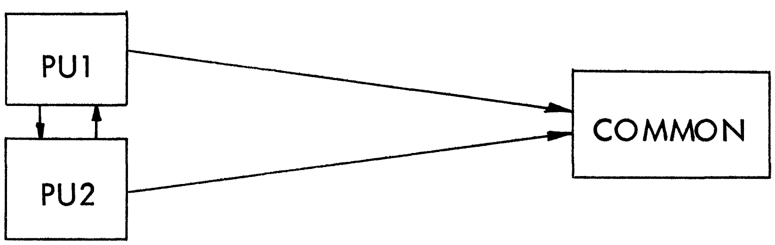
FORTRAN

Function module - subroutine or main program



One-to-one correspondence between program and data components of function module.

Complete program - set of interacting function modules and COMMON data blocks.



Program with two function modules and a COMMON data block.

COMMUNICATION BETWEEN FUNCTION MODULES

SIZE OF FUNCTION MODULES KNOWN AT TRANSLATION TIME

RELATIVE ADDRESSING WITHIN FUNCTION MODULE

RELATIVE ADDRESS FOR COMMON DATA BLOCKS

SYMBOLIC SUBROUTINE REFERENCES

PARAMETERS – RELATIVE ADDRESSING WITH RESPECT TO POINT OF CALL

```
TSR   S, 4
 A1
 A2
 A3
```

A1, A2, A3 ARE ADDRESSES OF PARAMETER VALUES

ACTUAL PARAMETER EXPRESSION IS EVALUATED PRIOR TO SUBROUTINE ENTRY

CALL BY REFERENCE

ALGOL

A PROGRAM CONSISTS OF A SINGLE FUNCTION MODULE CALLED A BLOCK WHICH MAY HAVE NESTED FUNCTION MODULES.

BEGIN

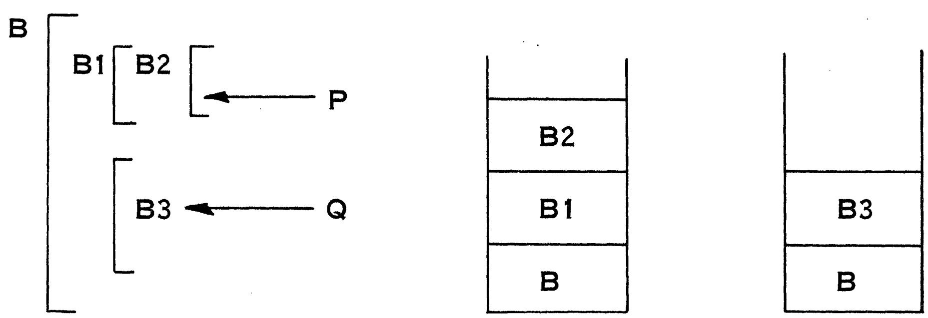
DECLARATIONS [REAL X;] [PROCEDURE P(X) BODY]

STATEMENTS [X:=X+1;] [NESTED BLOCK]

END

DECLARED INFORMATION STRUCTURES ARE CREATED ON ENTRY TO BLOCK AND DELETED ON EXIT FROM BLOCK+

NESTED FUNCTION MODULES — ACTIVATION RECORD STACK



FIXED PROGRAM PART EXECUTION AT P EXECUTION AT Q

STATIC AND DYNAMIC NESTING OF FUNCTION MODULES

PROCEDURE CALLS ARE IMPLICITLY NESTED

OWN VARIABLES — ENDURE BETWEEN ACTIVATIONS

INFORMATION STRUCTURE MODEL FOR ALGOL

FIXED PROGRAM COMPONENT	P
STATEWORD COMPONENT	W
STACK COMPONENT	S
INPUT COMPONENT	IN
OUTPUT COMPONENT	OUT
OWN VARIABLE COMPONENT	X

$I = (P, W, S, IN, OUT, X)$

SPECIFY TRANSFORMATION F IN TERMS OF HOW THEY AFFECT INFORMATION COMPONENTS

EMPHASIZE CREATION AND DELETION OF INFORMATION FIELDS

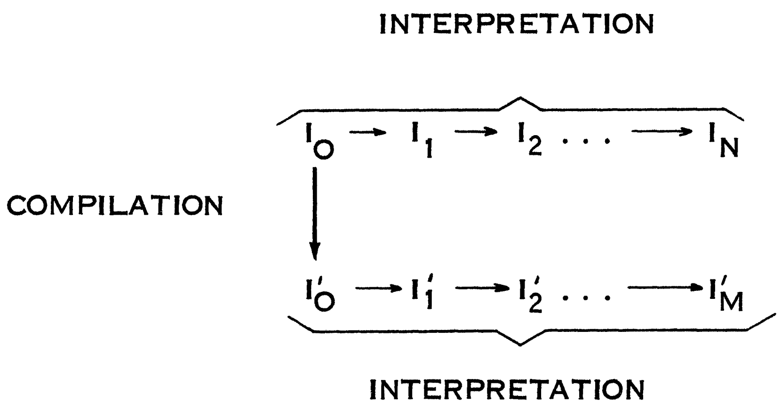
CREATION OF ACTIVATION RECORDS ON ENTRY TO FUNCTION MODULES – DELETION ON EXIT FROM FUNCTION MODULES.

CREATION OF TEMPORILY INFORMATION FIELDS DURING EXPRESSION EVALUATION.

ASSIGNMENT STATEMENT MAY MODIFY AN INFORMATION FIELD IN THE INTERIOR OF THE STACK.

INTERPRETATION VERSUS COMPILATION

COMPILATION IS A TRANSFORMATION FROM ONE INITIAL REPRESENTATION TO ANOTHER



INTERPRETATION PROCESS IS INSENSITIVE TO COMPILATIONS WHICH PRESERVE THE IDENTITY OF OPERATORS AND OPERANDS AND THE ORDER IN WHICH OPERATORS ARE APPLIED TO OPERANDS.

INTERPRETATION IS MORE RELEVANT TO MACHINE ORGANIZATION THAN COMPILATION.

COMPILERS CONSTITUTE AN INTERESTING CLASS OF COMPUTATIONS TO STUDY BUT TELL US LITTLE ABOUT THE SEMANTICS OF PROGRAMMING LANGUAGES BEING COMPILED.

MODELLING LANGUAGES

A LANGUAGE FOR SPECIFYING INFORMATION STRUCTURE MODELS IS CALLED A MODELLING LANGUAGE.

A MODELLING LANGUAGE MUST CONTAIN SYNTACTIC SPECIFICATION FACILITIES FOR SPECIFYING THE I COMPONENT OF INFORMATION STRUCTURE MODELS, AND FLEXIBLE FACILITIES FOR SPECIFYING CREATIONS, DELETION AND MODIFICATION OF INFORMATION STRUCTURES.

THERE ARE SIMILARITIES BETWEEN MODELLING LANGUAGES AND COMPILER–COMPILER LANGUAGES, BUT MODELLING LANGUAGES ARE CONCERNED WITH INTERPRETATION RATHER THAN WITH COMPILATION.

A SPECIFICATION OF AN INFORMATION STRUCTURE MODEL IN A MODELLING LANGUAGE WILL BE CALLED A SYNTAX DIRECTED INTERPRETER.

AN IMPLEMENTATION OF A MODELLING LANGUAGE WILL BE CALLED AN INTERPRETER–INTERPRETER SINCE IT IS AN INTERPRETER WHICH EXECUTES INTERPRETERS.

BINDING TIME

DECLARATIVE ACTION — REAL N;

IMPERATIVE ACTION — X: = 5;

DECLARATIVE ATTRIBUTES REMAIN INVARIANT DURING LIFETIME OF STRUCTURE.

IMPERATIVE ATTRIBUTES MAY BE MODIFIED DURING EXECUTION.

BINDING TIME OF AN ATTRIBUTE

TYPE IS BOUND AT DECLARATION TIME

VALUE IS BOUND AT ASSIGNMENT TIME

FORTRAN — ALL DATA STRUCTURES ARE CREATED (BOUND) PRIOR TO EXECUTION.

ALGOL — DATA STRUCTURES MAY BE NESTED ON BLOCK ENTRY.

PL/I — TEMPLATES FOR NEW DATA STRUCTURES MAY BE DECLARED.

EXAMPLES OF BINDING

Compilation - early binding of target language

Interpretation - late binding of target language

Macros - binding of users body by substitution

Procedures - no binding by physical substitution

Parameter call by value - bind parameter at time of entry to procedure

Parameter call by name - bind parameter value when it is used in the body of the procedure.

Parameter call by reference - bind parameter address at the time of entry to the procedure

Early binding - greater efficiency

Late binding - greater flexibility

SIDE EFFECTS

When does difference in binding strategy yield different results

Strategy A - bind value V at time T_1

Strategy B - bind value V at time T_2

Different result if value of V changes between T_1 and T_2

Example - call by value - T_1 is procedure entry time - call by name - T_2 is parameter use time

Difference in result if parameter value can be changed between procedure entry and parameter use

Procedures which may change values of external parameters during execution are said to have side effects.

OBJECTIVES

- OBJECTIVES — TO DEVELOP INSIGHT AND UNDERSTANDING OF THE STRUCTURE OF THE PROGRAMMING LANGUAGES.
- START WITH A DISCUSSION OF ALGOL 60 — COMMUNICATIONS OF THE ACM JANUARY 1963.
- DEVELOPED AS AN INTERNATIONAL ALGEBRAIC LANGUAGE.
- USED AS A LANGUAGE FOR THE COMMUNICATION OF ALGORITHMS — ALGORITHMS SECTION OF THE COMMUNICATIONS OF THE ACM.
- NOT AS WIDELY USED FOR PRACTICAL PROGRAMMING AS FORTRAN.
- BUT HAS A MORE INTERESTING STRUCTURE THAN FORTRAN.
- PRIME PURPOSE IS NOT TO TEACH ALGOL PROGRAMMING BUT TO DEVELOP A MODEL FOR THE STUDY OF PROGRAMMING LANGUAGES.
- THE CONCEPTS DEVELOPED FOR ALGOL WILL SERVE AS A STARTING POINT FOR THE DISCUSSION OF OTHER PROGRAMMING LANGUAGES.
- DISCUSSION OF ALGOL IMPLEMENTATION WILL SERVE AS A STARTING POINT FOR A DISCUSSION OF MACHINE ORGANIZATION AND FOR THE BUILDING OF MODELS OF IMPLEMENTATION.

BASIC CONSTITUENTS OF A PROGRAMMING LANGUAGE

- CONSTANTS OF A NUMBER OF DIFFERENT TYPES SUCH AS INTEGERS, FLOATING POINT NUMBERS, LOGICAL CONSTANTS.
- VARIABLES (IDENTIFIERS) WHOSE VALUES MAY BE ELEMENTS OF A GIVEN CLASS OF CONSTANTS.
- OPERATORS — EACH OPERATOR HAS A DEGREE WHICH SPECIFIES THE NUMBER OF ARGUMENTS — THE TYPE PERMITTED FOR EACH ARGUMENT AND THE TYPE PERMITTED FOR THE RESULT MUST BE SPECIFIED.
- EXPRESSIONS — WHICH SPECIFY OPERATORS WITH THEIR ARGUMENTS AND YIELD A VALUE ON EVALUATION. AN EXPRESSION MAY HAVE SUBEXPRESSIONS WHOSE VALUES ARE ARGUMENTS OF HIGHER LEVEL EXPRESSIONS.
- ASSIGNMENT STATEMENTS WHOSE PRINCIPAL EFFECT IS TO CHANGE THE VALUE OF A VARIABLE.
- BRANCHING STATEMENTS, CONDITIONAL STATEMENTS AND ITERATION STATEMENTS WHICH DETERMINE THE FLOW OF CONTROL IN A PROGRAM.
- DECLARATIONS WHICH SPECIFY THE TYPE AND ATTRIBUTES OF VARIABLES.

CONSTITUENTS OF ALGOL

COMPLETE ALGOL PROGRAM — CONSISTS OF AN ALGOL BLOCK

```
BEGIN
      DECLARATIONS
      STATEMENTS
END
```

DATA DECLARATIONS
INTEGER X; X IS AN INTEGER
REAL Y, Z; Y AND Z ARE FLOATING POINT NUMBERS
BOOLEAN X; X IS A FLOATING POINT VARIABLE

ARRAYS OF DATA ELEMENTS
REAL ARRAY A[1:N] ; A IS AN N–ELEMENT VECTOR OF FLOATING POINT NUMBERS

PROCEDURE DECLARATION
INTEGER PROCEDURE P(X,Y) SPECIFICATIONS BODY — DECLARATION OF A TWO–PARAMETER PROCEDURE P WHICH PRODUCES A VALUE OF THE TYPE INTEGER. THE SPECIFICATIONS SPECIFY PARAMETER TYPES. THE BODY IS A PROGRAM WHICH SPECIFIES THE ACTION TO BE PERFORMED WHEN THE PROCEDURE IS CALLED.

LABEL AND SWITCH DECLARATIONS
LABEL L; (IMPLICIT DECLARATION)
SWITCH S: = L1;L2; L3; L4; S IS INITIALIZED TO A 4–ELEMENT ARRAY OF LABELS

STATEMENTS INCLUDE ASSIGNMENT STATEMENTS (X : = X + 1;), BRANCHING STATEMENTS, CONDITIONAL STATEMENTS AND ITERATION STATEMENTS.

A BLOCK IS CONSIDERED TO BE A STATEMENT SO THAT STATEMENTS MAY HAVE BLOCKS NESTED INSIDE THEM.

CONSTANTS, VARIABLES AND EXPRESSIONS

CONSTANTS
CONSTANTS OF THE TYPE INTEGER 3; 4, 536
CONSTANTS OF THE TYPE REAL 3.5, 4.372
CONSTANTS OF THE TYPE BOOLEAN TRUE, FALSE

OPERATORS WITH OPERANDS	
INTEGER ADDITION	3 + 4
FLOATING POINT ADDITION	3.5 + 5.3
COMPOSITION OF OPERATIONS	3 + 4 X 5
PRECEDENCE OF X OVER +	(3 + 4) X 5
VARIABLES	X + Y X Z
STATEMENTS	Z: = X + Y;

TYPE SPECIFICATION
REAL X, Y, Z; INTEGER I, J;
Z: = X + Y;

MIXED EXPRESSIONS
Z: = X + I;
IMPLICIT CONVERSION FUNCTION
$X +_F$ CONVERT (I, REAL) FIRST CONVERT I TO REAL THEN USE FLOATING POINT ADDITION

RELATIONAL OPERATORS $< \leqq = \neq \geqq >$
RELATION EXPRESSION, X > Y; NUMERICAL ARGUMENTS, BOOLEAN RESULT

BOOLEAN OPERATORS $\neg \wedge \vee \supset \equiv$
BOOLEAN EXPRESSIONS; A $\wedge$ B, BOOLEAN ARGUMENTS, BOOLEAN RESULTS

STATEMENTS

V: = E;

LABELLED STATEMENT

L: x : = 1;

L: M: x : = 1;

MULTIPLE ASSIGNMENT

x: = y: = 1;

VALUE OF ASSIGNMENT STATEMENT IS VALUE OF ASSIGNED EXPRESSION

GO TO STATEMENT

GO TO L;

CONDITIONAL STATEMENTS AND CONDITIONAL EXPRESSIONS

STATEMENT

IF B THEN S_1 ELSE S_2

IF x = 0 THEN y: = y + 1; ELSE y: = y - 1;

IF B THEN S

EQUIVALENT TO IF B THEN S ELSE (NOTHING)

EXPRESSION

IF B THEN E1 ELSE E2

y: = IF x = 0 THEN y + 1 ELSE y - 1;

y: = y + (IF x = 0 THEN 1 ELSE -1);

DESIGNATIONAL EXPRESSION

GO TO IF x = 0 THEN L1 ELSE L2;

BLOCKS

COMPOUND STATEMENTS

```
BEGIN
    x: = 5;
    y: = 4
END
```

BLOCKS

```
BEGIN REAL K;
    K: = X;
    X: = Y;
    Y: = K
END
```

K IS A LOCAL VARIABLE
IT IS NESTED ON ENTRY TO THE BLOCK, AND DESTROYED ON EXIT FROM THE BLOCK

SCOPE RULES

Example: Nomenclature rules for nested blocks are as follows:

```
B:  begin real x,y;
        x: = 3;
        y: = 4;
    B1: begin real x,z;
            x: = 5;
            y: = 6;
            z: = 7;
        end;
     print (x,y,z)
    end
```

This sequence of ALGOL statements consists of a block B1 nested in a block B. The identifier y of the outer block can be used throughout the block B. However, the identifier x declared in the outer block cannot be used in the inner block because an identifier of the same name is declared in the inner block. The identifier x is bound in the inner block in the sense that if the two occurrences of the name x in the inner block were changed to another name, say u, then the computation defined by this program would be unaltered. The identifiers x, z of the inner block have meaning only in the inner block. In the print statement "print (x,y,z);" the identifiers x and y are associated with the declarations of x, y in the outer block and have the values x = 3, y = 6. The identifier z is undefined, so that this print statement would result in a diagnostic unless this program fragment were embedded in a block containing a declaration for the identifier z in its blockhead.

ITERATION STATEMENTS

Iteration statements have the following form:

<u>for</u> V: = for list <u>do</u> S   Execute the statement S for values of the variable V specified in the for list. It will be seen below that statements S may consist of arbitrarily complex nests of other statements, so that the restriction that the range of iteration be restricted to a single statement is not so severe as it appears.

The for-list elements may have one or more of the following three forms:

1. Individual expressions E.

2. Expressions of the form "E_1 <u>step</u> E_2 <u>until</u> E_3" indicating execution of S for values of V starting with E_1 and moving by increments of E_2 until E_3 is exceeded. Modification of E_2 and E_3 during execution of the statement S is permitted but not advised, since it may lead to trouble.

3. Expressions of the form "E <u>while</u> B", which specify execution of S with V = E as long as the value of B is <u>true</u>. In this case the statement S must be such that it can change the value of B to accomplish loop termination. S will normally also modify E when necessary.

The following example illustrates the use of a <u>for</u> statement to scan an N-element vector:

```
SUM: = 0;
for I: = 1 step 1 until N do
SUM: = SUM + A[I];
```

FUNCTION AND STATEMENT TYPE PROCEDURES

```
procedure ADD(A,N,SUM);
    real array A; integer N; real SUM;
    begin integer I;
        SUM: = 0;
        for I: = 1 step 1 until N do
        SUM: = SUM + A[I];
    end
```

This declaration is a statement-type procedure. The first line specifies the name and formal parameters of the procedure. The second line specifies the types of formal parameters. The first two lines together are said to constitute the procedure heading. The remaining lines of the procedure constitute the procedure body, which in this case consists of a single block. The effect of the procedure is to SUM N elements of the array which constitutes the first parameter and store the result as the value of the third parameter.

Procedure Statement: ADD(X,IS,S)

```
real procedure SUM(A,N);
    real array A; integer N;
    begin integer I; real X;
        X: = 0;
        for I: = 1 step 1 until N do
        X: = X + A[I];
        SUM: = X;
    end
```

This function-type procedure has one parameter less than the corresponding statement-type procedure, since the value is identified with the name and does not have to be explicitly specified by a parameter. The quantity X is used in the procedure body for accumulating the sum since an occurrence of SUM on the right-hand side of an assignment statement would be interpreted as a reentrant call of the procedure.

Call of Function Type Procedure

```
X: = SUM (A,15)↑ 2 x SUM(B,20);
```

PARAMETER CALLING

CALL BY VALUE - EVALUATE ON ENTRY TO PROCEDURE

CALL BY NAME - EVALUATE WHEN USED DURING PROCEDURE EXECUTION

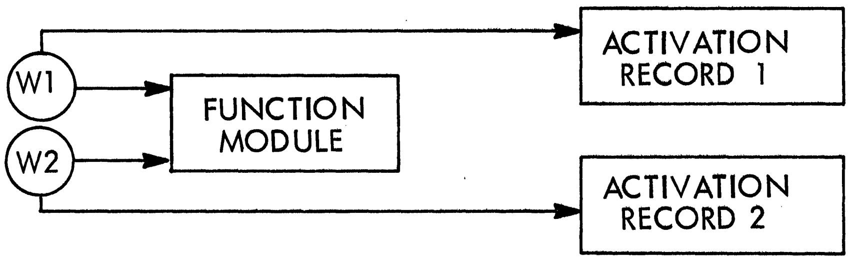
```
REAL PROCEDURE P(A);
    REAL A;
    BEGIN
        K: = 5;
        P: = A
    END
```

IF A IS CALLED BY NAME, P(K) IS ALWAYS 5

IF A IS CALLED BY VALUE, P(K) IS GIVEN BY THE VALUE OF K ON ENTRY TO THE PROCEDURE.

ACTIVATION RECORDS

REPRESENTATION OF FUNCTION MODULES



THE STRUCTURE OF A COMPLETE PROGRAM CAN BE DESCRIBED IN TERMS OF THE STRUCTURE OF ITS FUNCTION MODULES.

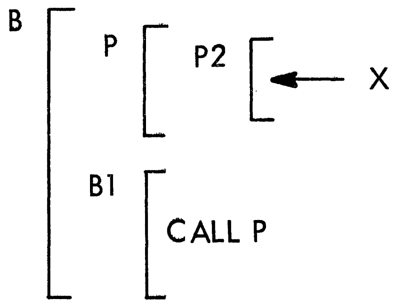
ENTRY TO AND EXIT FROM FUNCTION MODULES IS IN LAST-IN-FIRST-OUT ORDER.

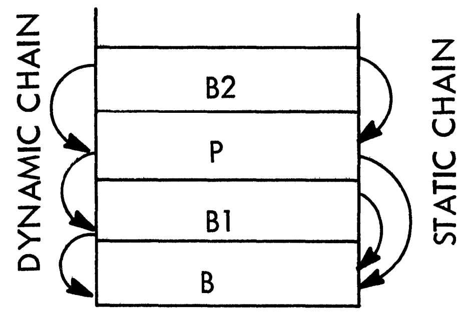
FUNCTION MODULES CAN BE STORED IN A STACK.

STATIC AND DYNAMIC NESTING

STATIC NESTING

DYNAMIC NESTING





EXECUTION AT X

AT X, STATIC NESTING LEVEL IS 3, DYNAMIC NESTING LEVEL IS 4.

STATIC NESTING LEVEL IS A PROGRAM INVARIANT.

DYNAMIC NESTING LEVEL MAY BE ARBITRARILY DEEP WHEN CELLS ARE RECURSIVE.

REPRESENTATION OF IDENTIFIERS BY INTEGER PAIRS

(L,J) REPRESENTATION OF IDENTIFIERS

L - LEVEL OF STATIC NESTING

J - RELATIVE ADDRESS WITHIN ACTIVATION RECORD

(L,J) ADDRESS CAN BE USED FOR ACCESSING

CURRENT ENVIRONMENT VECTOR MODEL

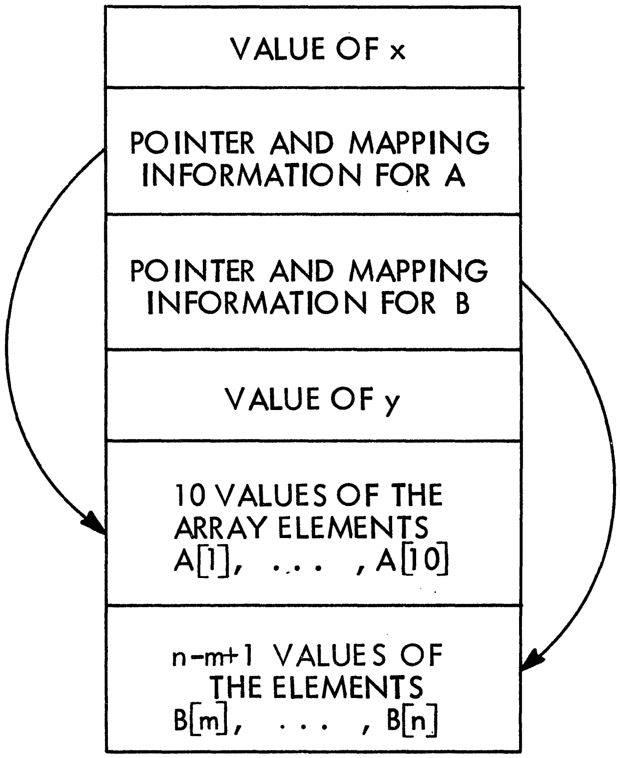
STATIC CHAIN MODEL

WITH STATIC CHAIN MODEL USE ADDRESS (R,J) WHERE R IS THE DIFFERENCE IN THE STATIC LEVEL OF NESTING BETWEEN THE POINT OF REFERENCE AND POINT OF USE OF THE IDENTIFIER.

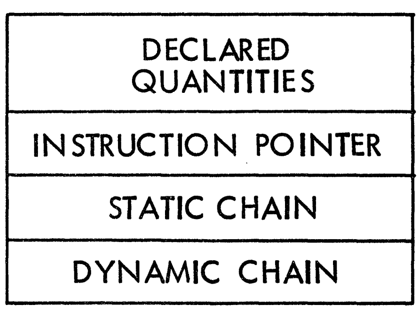
R IS THE NUMBER OF STATIC CHAIN LINKS WHICH MUST BE FOLLOWED TO REACH THE ACTIVATION RECORD WHICH CONTAINS THE VALUE OF THE IDENTIFIER.

RELATIVE ADDRESSING WITHIN PROCEDURE.
STORAGE FOR DECLARED QUANTITIES

BEGIN REAL x; REAL ARRAY A[1:10], B[m,:n]; REAL y; . . . END



ACTIVATION-RECORD DATA STRUCTURE CORRESPONDING TO THE BLOCKHEAD BEGIN REAL x; REAL ARRAY A[1:10], B[m:n]; REAL y;.



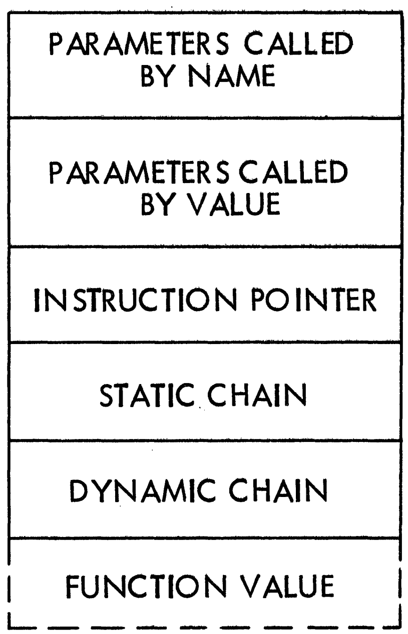
STORAGE FOR ORGANIZATIONAL QUANTITIES

PROCEDURE ACTIVATION RECORDS

PARAMETERS CALLED BY VALUE - STORE VALUES

PARAMETERS CALLED BY NAME - STORE PROCEDURE CALLS

STORE VALUE OF FUNCTION TYPE PROCEDURES ON COMPLETION.



ENVIRONMENT MODIFICATION

ON ENTRY TO AND EXIT FROM A BLOCK

ON ENTRY TO AND EXIT FROM A PROCEDURE

ON EVALUATION OF A PARAMETER CALLED BY NAME WITHIN A PROCEDURE

ON JUMP TO A LABEL

MODE OF ACCESS TO INFORMATION

SYSTEM SYMBOLS - DENOTE FIXED INFORMATION STRUCTURES DEFINED BY THE SYSTEMS

BEGIN, FOR, +, 11.63

LOCAL IDENTIFIERS - LOCAL TO THE BLOCK CURRENTLY BEING EXECUTED.

NON LOCAL IDENTIFIERS - IN ENCLOSING BLOCKS

PROCEDURE PARAMETERS - ACCESS INFORMATION THROUGH POINT OF CALL.

- BY VALUE
- BY NAME

COMPILATION OF ALGOL PROGRAMS

EDIT FOR MORE CONVENIENT EXECUTION

EXPLICIT LABEL DECLARATIONS IN BLOCKHEADS

REPRESENT INTEGERS BY IDENTIFIER PAIRS

FUNCTION HEADING REPLACED BY STORAGE ALLOCATION INSTRUCTIONS

EXECUTABLE STRINGS ARE CONVERTED EITHER TO POSTFED NOTATION OR TO MACHINE LANGUAGE.

INFORMATION STRUCTURE MODEL FOR ALGOL

FIXED PROGRAM COMPONENT	P
STATEWORD COMPONENT	W
STACK COMPONENT	S
INPUT COMPONENT	IN
OUTPUT COMPONENT	OUT
OWN VARIABLE COMPONENT	X

I = (P,W,S,IN,OUT,X)

SPECIFY TRANSFORMATION F IN TERMS OF HOW THEY AFFECT INFORMATION COMPONENTS.

EMPHASIZE CREATION AND DELETION OF INFORMATION FIELDS.

CREATION OF ACTIVATION RECORDS ON ENTRY TO FUNCTION MODULES - DELETION ON EXIT FROM FUNCTION MODULES.

CREATION OF TEMPORARY INFORMATION FIELDS DURING EXPRESSION EVALUATION.

ASSIGNMENT STATEMENT MAY MODIFY AN INFORMATION FIELD IN THE INTERIOR OF THE STACK.

OUTLINE

1. THE STRUCTURE OF THE DATA MANAGEMENT ENVIRONMENT
2. THE JOB MANAGEMENT FUNCTION
3. THE EXTERNAL FILE SYSTEM
4. THE INTERNAL FILE SYSTEM
5. REVIEW OF DATA MANAGEMENT TECHNOLOGY

OBJECTIVES OF THE SESSIONS ON DATA MANAGEMENT

- TO PRESENT DATA MANAGEMENT CONCEPTS
- TO CONSTRUCT A FRAMEWORK FOR THE STUDY OF DATA MANAGEMENT PROBLEMS
- TO PROJECT AN APPROACH TO A MULTI-USER COMMON DATA BASE SYSTEM
- TO EXAMINE SOME CURRENT AND PROPOSED DESIGNS FOR DATA MANAGEMENT SYSTEMS

THE DATA BASE

- THE ON-GOING DATA BASE
- THE PROBLEM OF SCALE
- SYSTEM RESPONSIBILITIES
  - MULTI-LEVEL STORAGE MANAGEMENT
  - ARCHIVING AND RECOVERY
  - DATA INTEGRITY

PROGRAM STRUCTURES AND THE DATA BASE

- THE PROGRAM DATA DECLARATION AS A TEMPLATE
- THE COMMON DATA BASE
- PROGRAM/DATA INDEPENDENCE

THE DATA MANAGEMENT SYSTEM

– A DEFINITION

THE STORAGE, ASSOCIATION, AND RETRIEVAL OF DIVERSE DATA ELEMENTS IN RESPONSE TO A VARIETY OF PROCESSING DEMANDS

SOFTWARE TO	DEFINE DATA
	USE IT
	MAINTAIN IT
	LINK IT TO PROGRAMS
	LINK IT TO PEOPLE

THE DATA MANAGEMENT SYSTEM

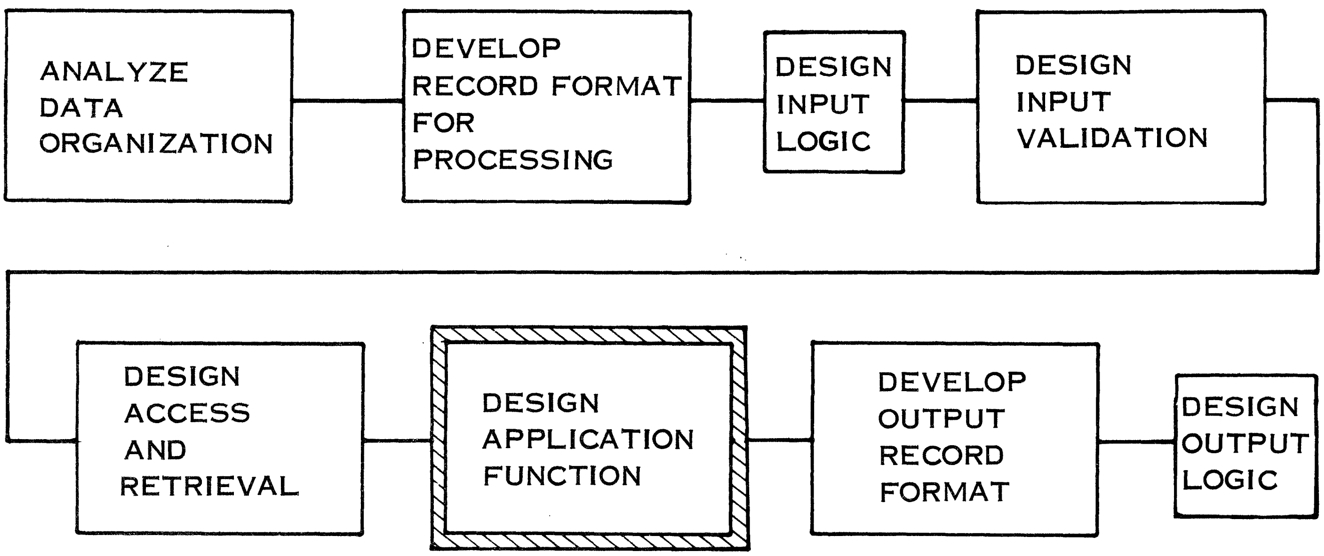
— OBJECTIVES

- CENTRAL RESPONSIBILITY FOR STORAGE; RETRIEVAL; AND REPORTING SERVICES TO THE USER.
- CENTRAL RESPONSIBILITY FOR DATA INTEGRITY
- SERVICES TO THE APPLICATION PROGRAMMER
- REDUCTION OF PROGRAM DEVELOPMENT COSTS
- INCREASE IN PROGRAM LIFE
- ADAPTABILITY OF DATA STRUCTURES
- OPTIMIZATION OF DATA UTILIZATION

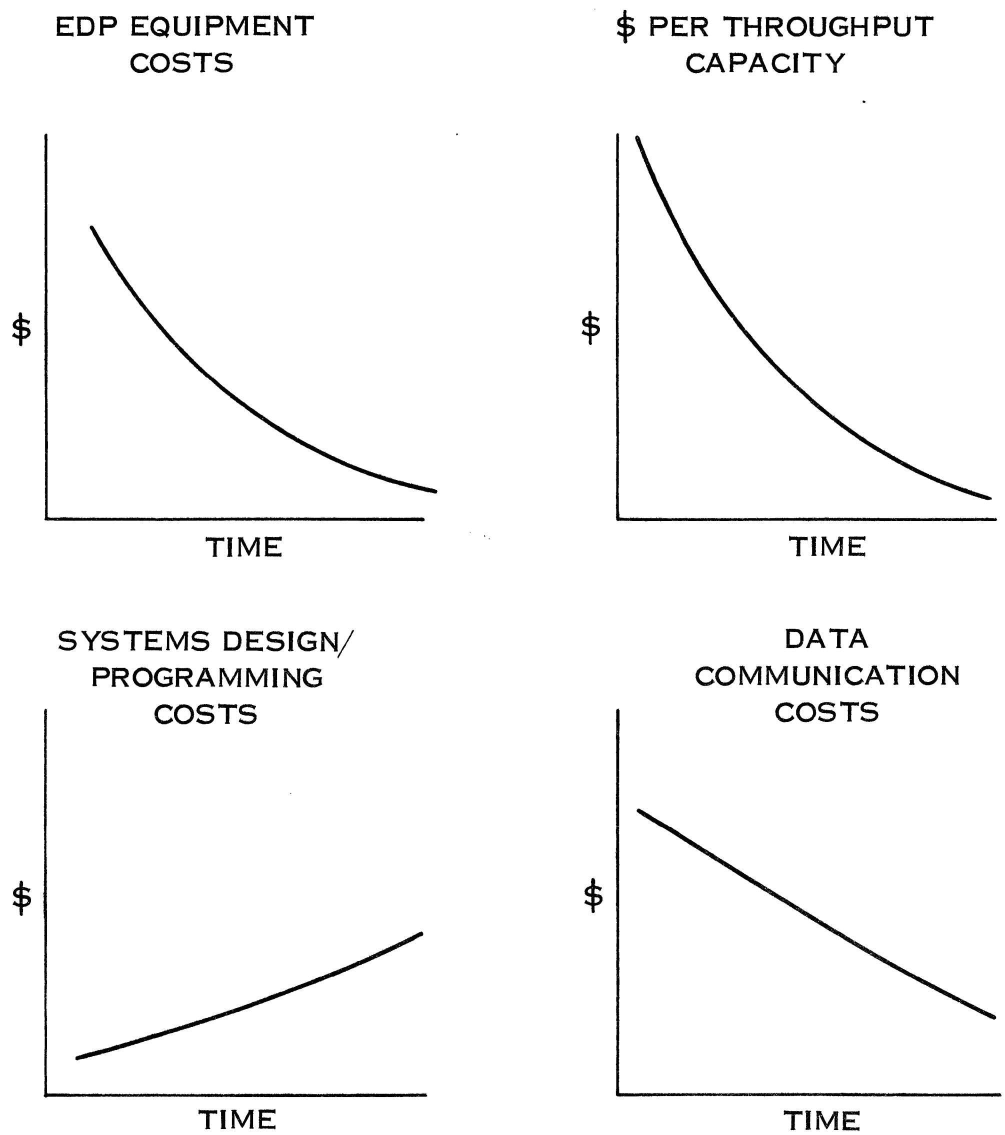
— PRICE

- "OVERHEAD"
- SURRENDER OF TACTICAL DECISIONS
- REDUCTION OF PROGRAMMER OPTIONS

PROGRAMMING COSTS



ECONOMIC TRENDS



APPLICATIONS

- BUSINESS DATA PROCESSING
- MANAGEMENT INFORMATION SYSTEMS
- COMMAND AND CONTROL
- INTERACTIVE SYSTEMS
- INFORMATION RETRIEVAL SYSTEMS
- MULTI–USER SYSTEMS

ENVIRONMENT

- DATA CENTERS
- CENTRALIZED COMPUTATION SERVICES
- THE COMPUTING UTILITY
- THE OPERATIONS CONTROL CENTER
- THE CORPORATE DATA PROCESSING CENTER

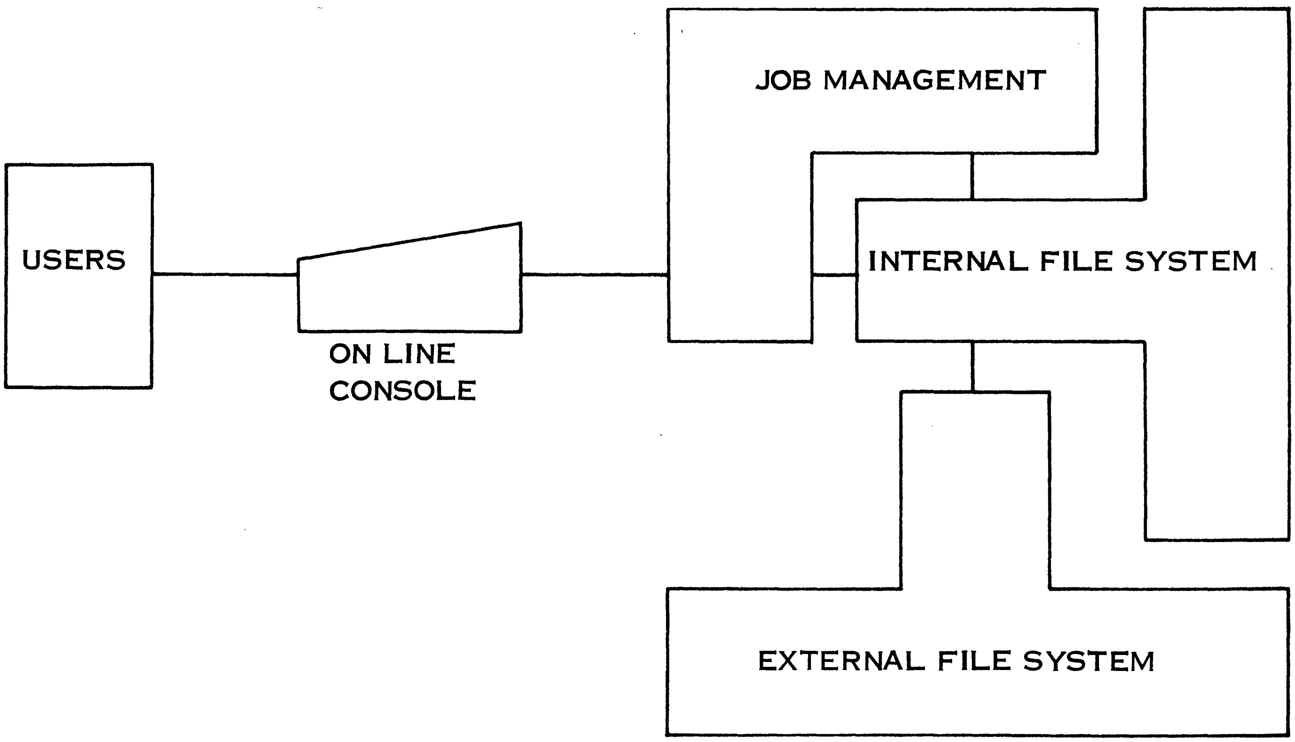
TYPICAL HARDWARE

- LARGE SCALE COMPUTER
- MASS RANDOM ACCESS STORES
- REMOTE ACCESS TERMINALS

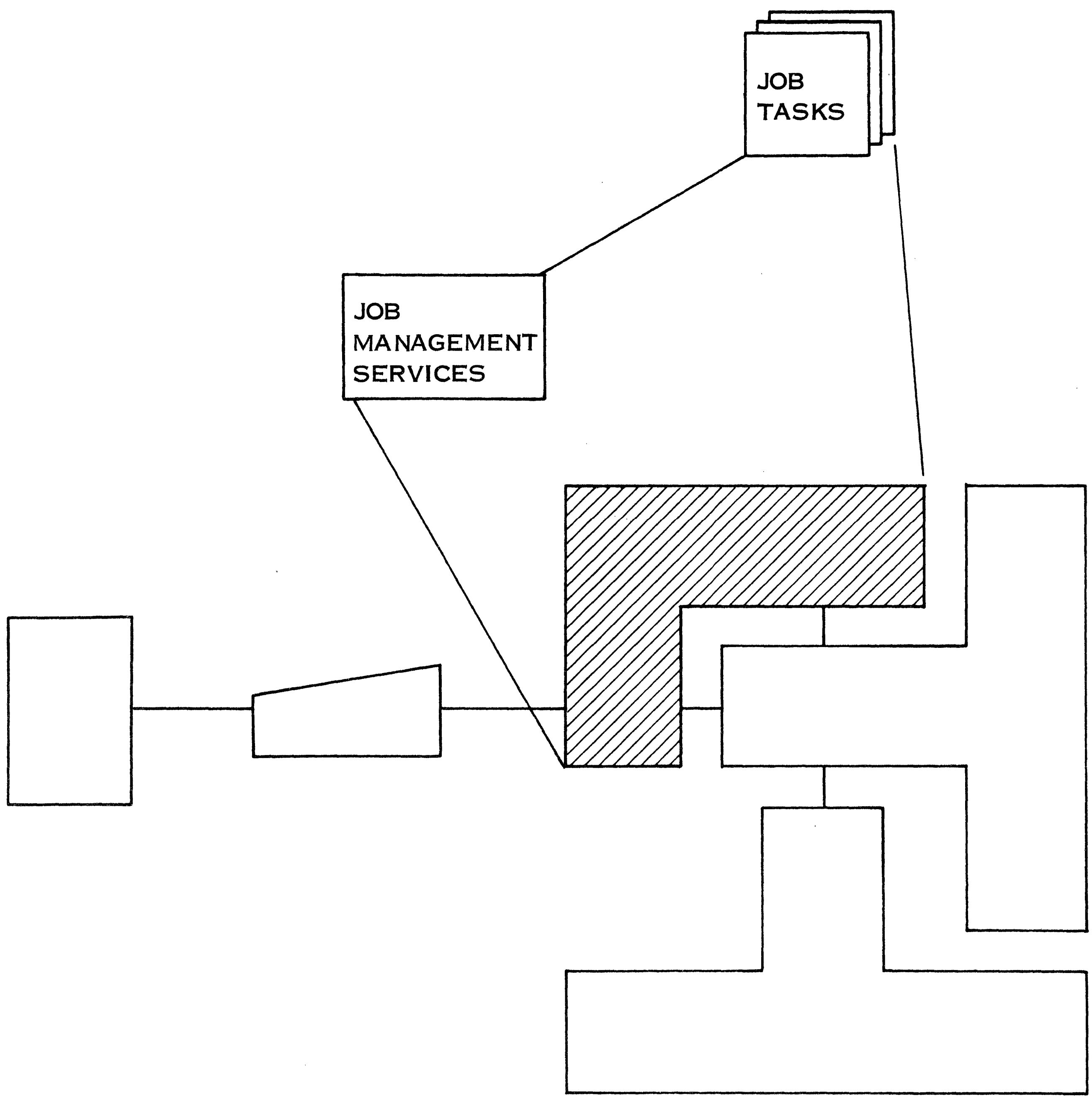
COMPONENTS OF A FULL–SERVICE GENERALIZED DATA MANAGEMENT SYSTEM

- INTERNAL FILE SYSTEM
- EXTERNAL FILE SYSTEM
- JOB MANAGEMENT SYSTEM
- THE USERS
- SYSTEM SUPPORT FUNCTIONS

A FULL–SERVICE GENERALIZED DATA MANAGEMENT SYSTEM



THE JOB MANAGEMENT SYSTEM



USER COMMANDS

- CONTROL
- FILE MANIPULATION
- RELATION MANIPULATION
- FIELD MANIPULATION
- BLOCK TRANSFER

EXAMPLE OF RESPONSE TO *DEFINE FILE.

*DEFINE FILE

FILE DEFINITION. DO YOU WANT INSTRUCTIONS: YES
PROVIDE THE FOLLOWING (12 CHARACTER MAXIMUM FOR EACH):
NAME OF FILE
TYPE OF FILE (NAMED OR NUMBERED)
THE NAME OF EACH DATA FIELD FOLLOWED BY ITS CODING
ACCEPTABLE CODINGS ARE THE FOLLOWING:
BCD, INTEGER, FLT. POINT, BCD LIST, INTEGER LIST, FLT. PT. LIST

A CARRIAGE RETURN MUST FOLLOW EACH INPUT TERM.
THE WORD* DONE TERMINATES INPUT.

COMPUTER
NAMED
RENTAL
INTEGER
ADD TIME
FLT. POINT
CYCLE TIME
FLT. POINT
CORE STORAGE
INTEGER
DRUM STORE
INTEGER
WORD SIZE
BCD
SPEC FEATURE
BCD LIST
*DONE

EXAMPLE (CONTD)

THE INPUT TABLE FOLLOWS:

COMPUTER	NAMED
RENTAL	INTEGER
ADD TIME	FLT. POINT
CYCLE TIME	FLT. POINT
CORE STORAGE	INTEGER
DRUM STORAGE	INTEGER
WORD SIZE	BCD
SPEC FEATURE	BCD LIST

IS THIS WHAT YOU WANT. IF NOT, TYPE "NO" AND START AGAIN.
YES
FILE SET—UP COMPLETED.

COMMAND EXECUTED.

GIVE COMMAND OR TYPE *CHOICES.

EXAMPLE OF RESPONSE TO *INPUT ENTRIES.

*INPUT ENTRIES

TYPE:
FILE NAME
*INSTRUCTIONS OR *NO

COMPUTER
* INSTRUCTIONS

FOR EACH ENTRY TO BE ADDED:

1. WAIT UNTIL "READY" IS TYPED
2. LIST CONTENTS OF THE DATA FIELDS
   A. IF SOME FIELD IS ITSELF A LIST,
      A BLANK LINE SIGNIFIES THE END OF THE LIST
   B. FORMATS ARE:
      FOR BCD : FIELD LENGTH=6, LEFT JUSTIFY DATA
      FOR INTEGERS : FIELD LENGTH=12, RIGHT JUSTIFY DATA
      FOR FLT. PT. : FIELD LENGTH=16; PROVIDE DECIMAL PT.
3. TYPE THE PARENT OF THIS ENTRY FOR EACH RELATION LISTED
4. TO TERMINATE INPUT OF ENTRIES, PRESS CR AFTER "READY" IS TYPED

EXAMPLE (CONTD.)

DATA FIELDS

NAME	CODING
NAME	BCD
ADD TIME	FLOATING POINT
CORE STORAGE	INTEGER
CYCLE TIME	FLOATING POINT
DRUM STORE	INTEGER
RENTAL	INTEGER
SPEC FEATURE	BCD LIST
WORD SIZE	BCD

RELATIONS
THERE ARE NO RELATIONS

READY

IBM 7094 11
1.4
32
1.4
186
160
IN 'RUP
16XR'S
FLT. PT
IN ADD
648

EXAMPLE OF RESPONSE TO *SEARCH FILE.

```
*SEARCH FILE
THE ACTIVE FILES ARE :

COMPUTER
HOME ADDRESS
STREET

PROVIDE FILE NAME:  COMPUTER

              (FILE DESCRIPTION)

COMPUTER     IS A FILE WITH NAMED ENTRIES.
NO. OF DATA FIELDS PER ENTRY = 7

SAMPLE ENTRY FOLLOWS:

ENTRY: CDC 3600
 ADD TIME       :      2.00
 CORE STORAGE   :      262
 CYCLE TIME     :      1.50
 DRUM STORE     :      0
 RENTAL         :      55
 SPEC FEATURE   :  IN'RUP
                   6XR'S
                   FLT. PT
                   IN'ADD
 WORD SIZE      :  488
```

EXAMPLE (CONTD)

(START OF SEARCH)

PROVIDE FIELD NAME: CYCLE TIME
PROVIDE CONDITION (EQ,LT,GT,LTOREQ,GTOREQ): LT
PROVIDE TEST VALUE (FLTG. POINT NUMBER): 4.0
DO YOU WNT FULL ENTRIES PRINTED: YES

(START OF SUBFILE)

ENTRY: CDC 3600

ADD TIME	:	2.00
CORE STORAGE	:	262
CYCLE TIME	:	1.50
DRUM STORE	:	0
RENTAL	:	55
SPEC FEATURE	:	IN'RUP
		6XR'S
		FLT. PT
		IN' ADD
WORD SIZE	:	488

FILE MANIPULATION COMMANDS

- DEFINE FILE
- INPUT ENTRIES
- SEARCH FILE
- LIST FILES
- PRINT FILE
- FIND VALUE
- DELETE FILE

EXAMPLE (CONTD)

MANUAL MODE

LIST THE NAMES OF THE PARENT ENTRIES FOLLOWED BY THE NAMES OF THEIR RELATED SUBFILE ENTRIES. TO TERMINATE THE LIST OF SUBFILE ENTRIES LEAVE A LINE BLANK. TO TERMINATE INPUT LEAVE ANOTHER LINE BLANK. WAIT FOR THE WORK "READY" BEFORE TYPING IN EACH GROUP OF PARENT AND LINKEES.

WHICH MODE DO YOU WANT* MANUAL

READY

WOBURN
ALLEN MARGAR
ATHANS MICHAEL
CORR DAVID F

READY

CAMBRIDGE
ANDERSON ALL
COHEN MITCHE
CURTISS ARTHUR
FALB PETER L

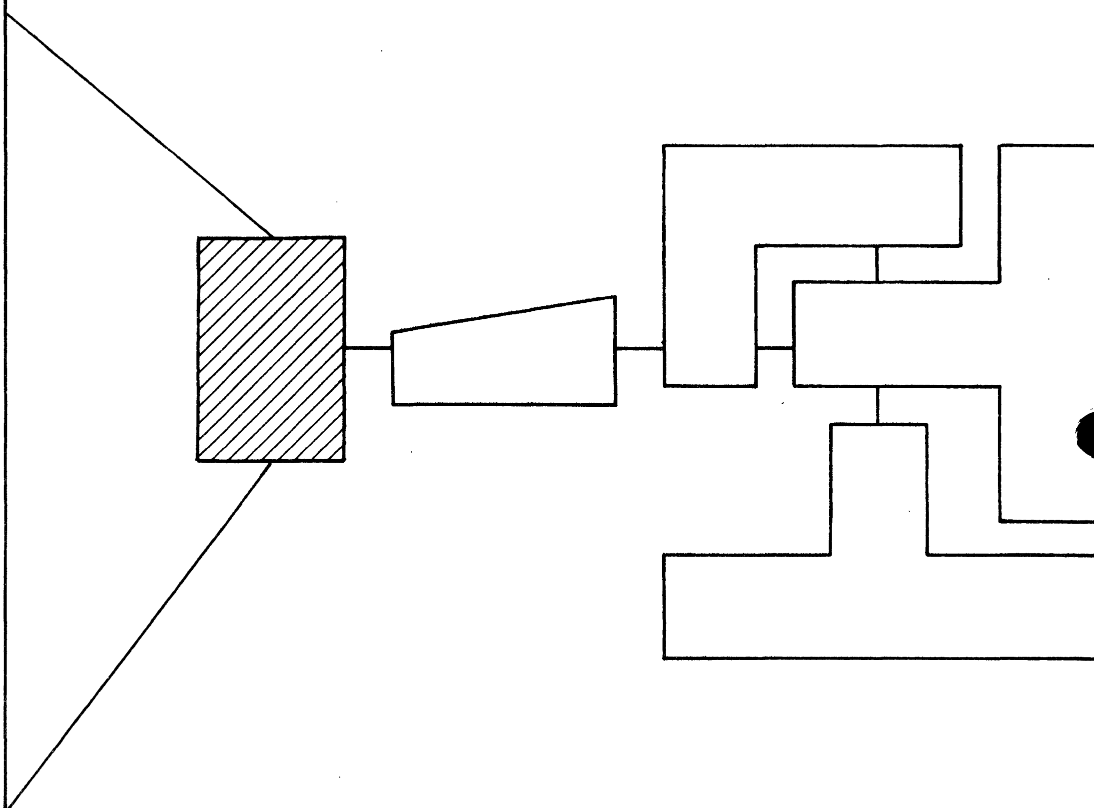
THE USERS OF THE DATA MANAGEMENT SYSTEM

MANAGERS
- AD HOC REPORTS
- QUERIES
- ADMINISTRATION

PROGRAMMERS
- LOGICAL DATA SERVICES
- DATA–INDEPENDENT PROGRAMS

ANALYSTS
- BUILD SYSTEM JOBS
- GENERAL–PURPOSE MODULARITY

DATA ADMINISTRATOR
- DATA BASE CONTROL
- MONITOR USE
- CONTROL ACCESS



RELATION MANIPULATION COMMANDS

- DEFINE RELATION
- SEARCH RELATION
- LIST RELATIONS
- DESCRIBE RELATIONS
- FIND PARENT
- FIND LINKEE
- RELATE ENTRY
- DELETE RELATION

FIELD MANIPULATION COMMANDS

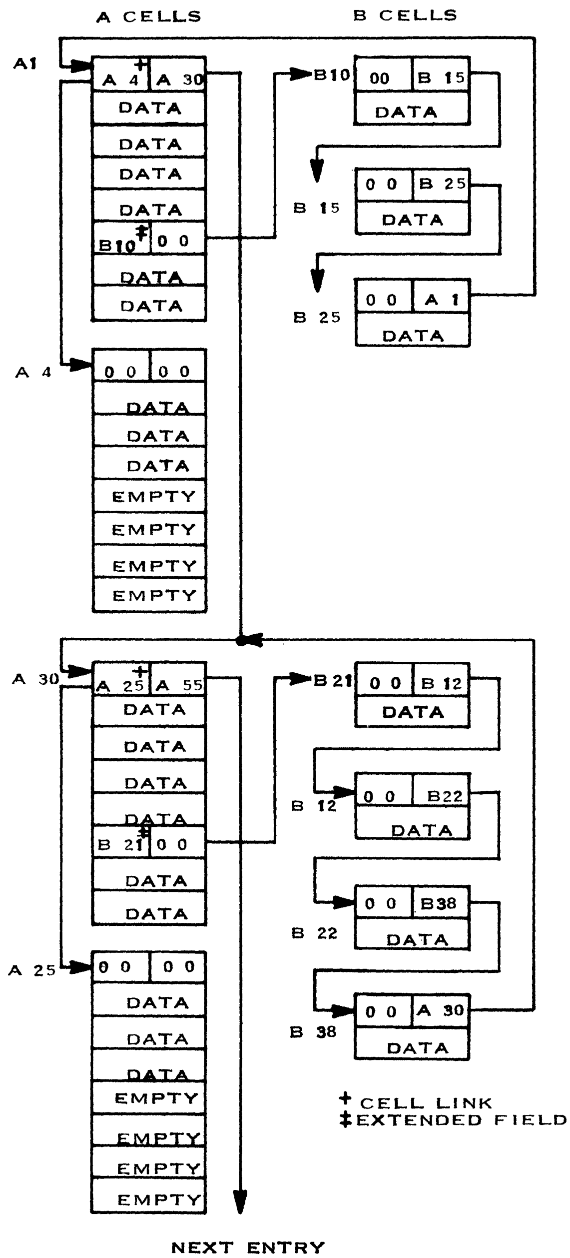
- DEFINE DATA FIELD
- DELETE DATA FIELD
- DEFINE FIELD VALUE

BLOCK DATA TRANSFER COMMANDS

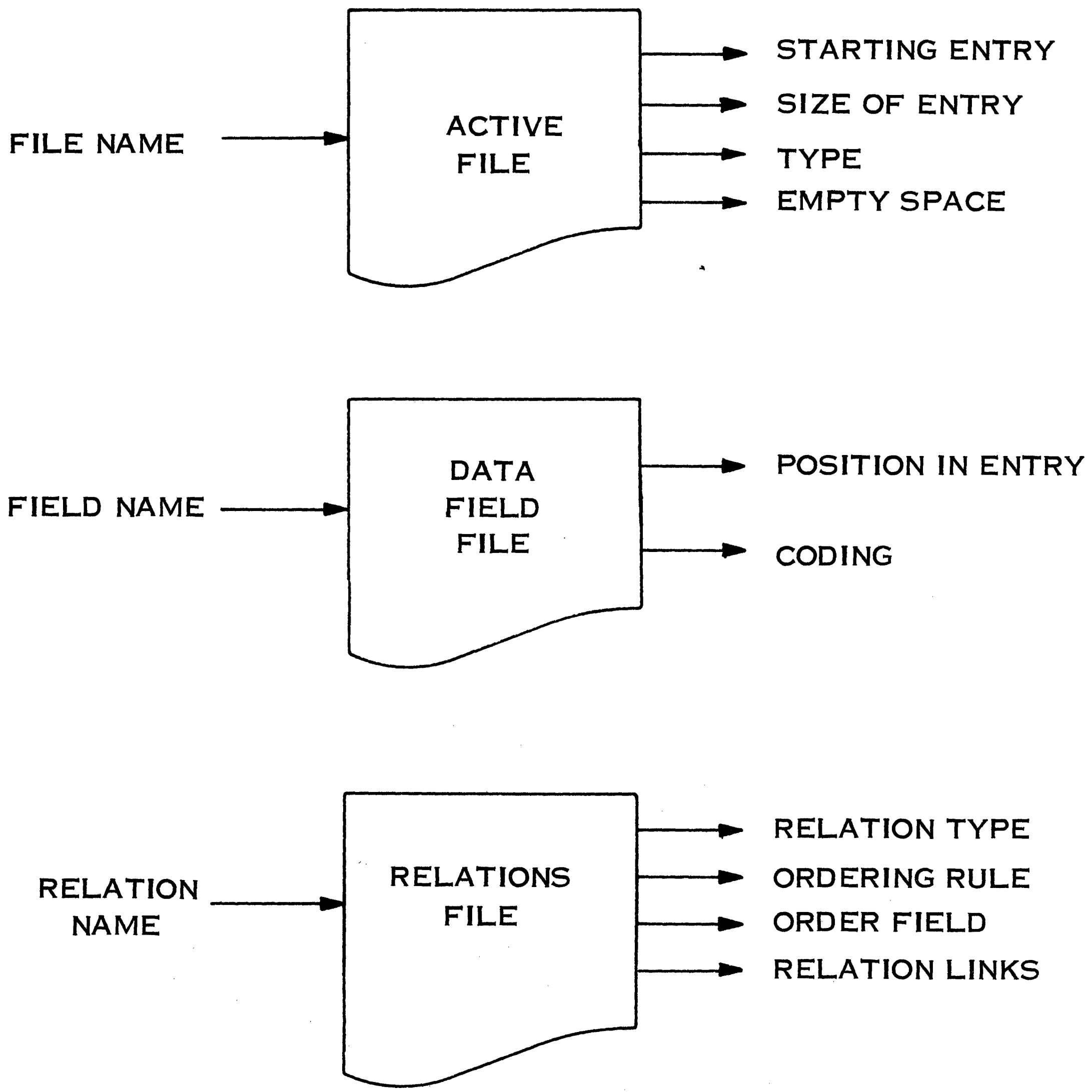
— READ CARDS

— WRITE TAPE

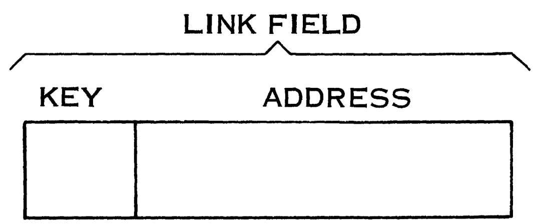
CELL STRUCTURE OF SAMPLE FILE



BASIC FILES

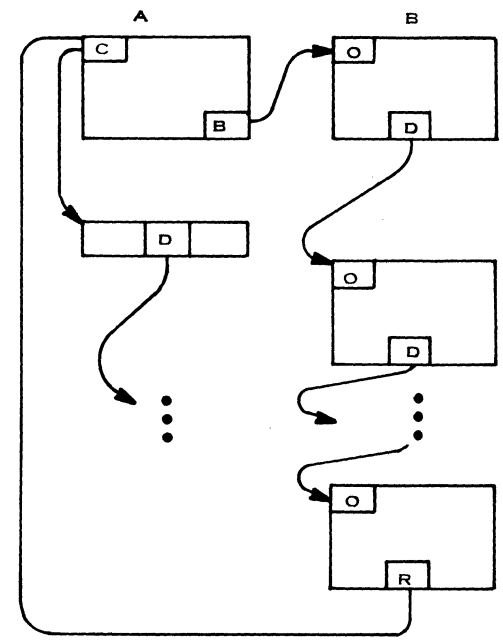

FILE NAME
ACTIVE FILE
STARTING ENTRY
SIZE OF ENTRY
TYPE
EMPTY SPACE
FIELD NAME
DATA FIELD FILE
POSITION IN ENTRY
CODING
RELATION NAME
RELATIONS FILE
RELATION TYPE
ORDERING RULE
ORDER FIELD
RELATION LINKS

LINK TYPES

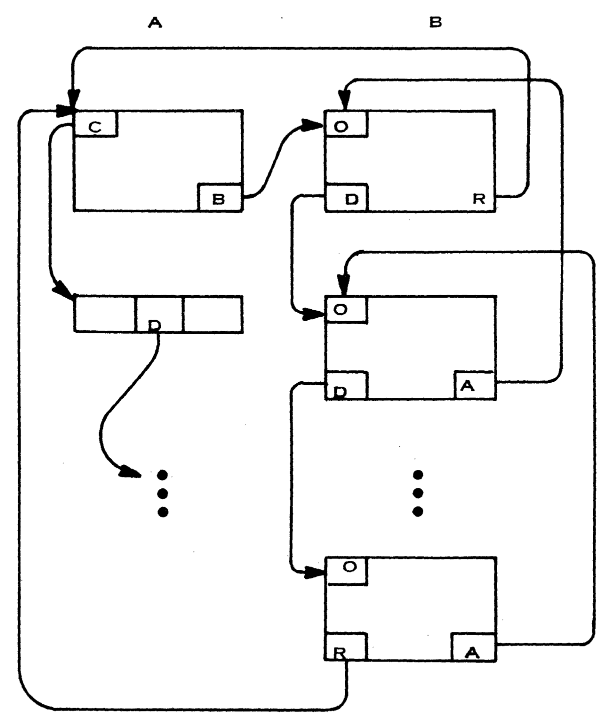


KEY	MEANING OF ADDRESS	
P	POINTER FO FILE ENTRY	ASSOCIATIVE LINKS
B	BRANCH TO SUBFILE	
D	DESCEND TO NEXT FILE ENTRY	
A	ASCEND TO PRECEDING ENTRY	
R	RETURN FROM SUBFILE TO PARENT FILE	
U	UNUSED LINK FIELD	
C	CELL LINK	
E	EMPTY SUBLIST INDICATOR	

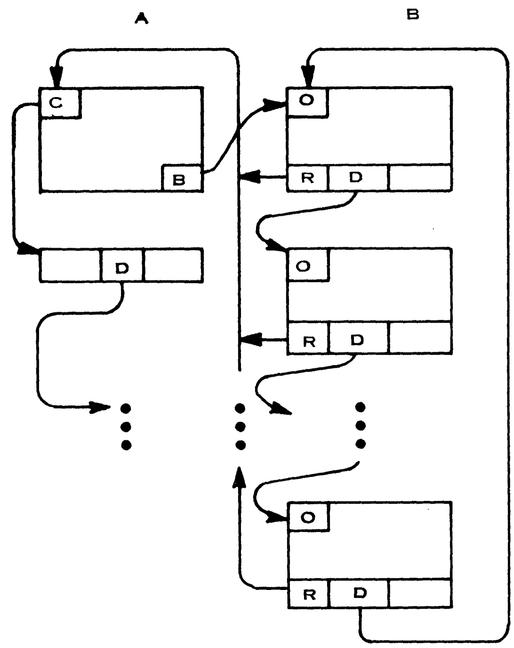
ONE—WAY LIST



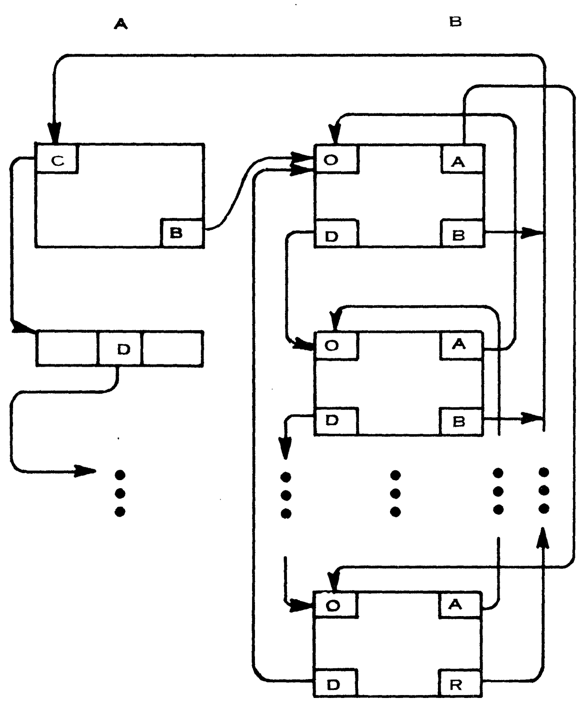
TWO—WAY LIST



ONE—WAY RING



TWO—WAY RING



STATEMENT GROUPING

```
DO;                    DO STATEMENT
X = 5;
Y = 3;
END;                   END STATEMENT


DO I = 1 BY 1 TO N;
SUM = SUM   A[I];
END;


DO  I = E1  BY  E2 TO  E3;   E1, E2, E3  ARE INITIALIZED
BY VALUE
FOR  I  =  E1  STEP  E2  UNTIL  E3  DO  S;
E1, E2, E3 ARE INITIALIZED BY NAME
```

BLOCKS

BEGIN

STATEMENTS
AND DECLARATIONS

END

DECLARATIONS NEED NOT OCCUR AT THE BEGINNING OF THE BLOCK BUT ARE ASSUMED EXECUTED AS THOUGH THEY WERE AT THE BEGINNING OF THE BLOCK.

SIMULTANEOUS DECLARATIONS

LAYERS OF DECLARATION AS IN CPL

DYNAMIC DECLARATIONS - NEW DECLARATION EVERY TIME IT IS ENCOUNTERED DURING EXECUTION - LIKE A PROCEDURE CALL WHOSE EFFECT IS TO DECLARE RATHER THAN TO EXECUTE.

PROCEDURES

NAME: PROCEDURE(P) SPECIFICATIONS

DECLARATIONS AND STATEMENTS

END;

NAME IS LIKE A LABEL

PARAMETERS ARE CALLED BY REFERENCE

RETURN STATEMENT

RETURN (EXP) VALUE OF EXP IS RETAINED TO POINT OF CALL

DECLARATIONS

DECLARE NAME ATTRIBUTES

DECLARE (N1,N2) A

DECLARE (N1 A1, N2 A2) A3

CLASSIFICATION OF ATTRIBUTES

TYPE ATTRIBUTES - LIKE DATA TYPES OF ALGOL - SPECIFY THE RANGE OF VALUES AND SET OF OPERATIONS APPLICABLE TO THE IDENTIFIER.

STRUCTURE ATTRIBUTES - SPECIFY SUBSTRUCTURE OF THE INFORMATION STRUCTURE DENOTED BY THE IDENTIFIER.

SCOPE ATTRIBUTES - SPECIFY THE RANGE OF STATEMENTS OF THE STATIC SOURCE PROGRAM OVER WHICH THE IDENTIFIER HAS MEANING.

STORAGE ATTRIBUTES - SPECIFY THE LIFETIME OF THE INFORMATION STRUCTURE.

DATA ATTRIBUTES

BASE ATTRIBUTES - DECIMAL, BINARY

SCALE ATTRIBUTES - FIXED, FLOAT

MODE ATTRIBUTES - REAL, COMPLEX

PRECISION ATTRIBUTES - (N, M)

DECLARE A DECIMAL FIXED REAL (3,2);

DEFAULT ATTRIBUTES

BINARY FIXED REAL

DEFAULT PRECISION IS IMPLEMENTATION-DEFINED

CHARACTERS, LOGICALS AND POINTERS

NON-ARITHMETIC DATA TYPES

CHARACTERS AND CHARACTER STRINGS

DECLARE A

BITS AND BIT - STRINGS

STRING CONSTANTS 'ABC', '0100'B

STRING VARIABLES X 'ABC'; Y '0100'B

POINTERS AND POINTER VALUED VARIABLES

POINTER P, Q;

FUNCTION ADDR(X) - RETURNS POINTER TO X

P = ADDR(A)
P → A = 5

ARRAYS AND STRUCTURES

VARIABLE DIMENSIONS - LOWER AND UPPER BOUNDS
DECLARE A(I, 5:10);

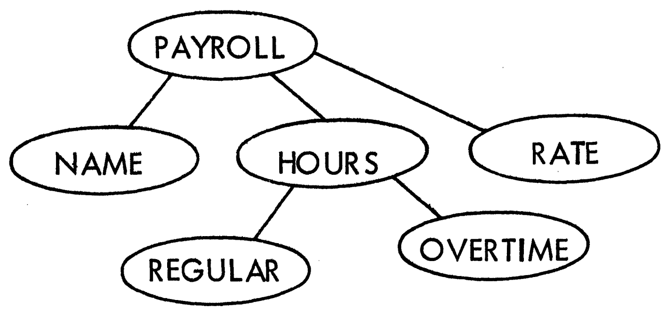
ARRAYS ARE RESTRICTED TO BE RECTANGULAR, AND TO HAVE ALL ELEMENTS BE OF THE SAME TYPE

STRUCTURES - DATA ELEMENTS MAY BE OF DIFFERENT TYPES - NOT RESTRICTED TO BE RECTANGULAR

DATA ELEMENTS OF A STRUCTURE ARE TERMINAL VERTICES OF A TREE

LEVEL NUMBERS

```
DECLARE   1 PAYROLL
          2 NAME
          2 HOURS
            3 REGULAR
            3 OVERTIME
          2 RATE
```



ATTRIBUTES AND NAMES OF STRUCTURE COMPONENTS

ASSOCIATE ATTRIBUTES WITH DATA ITEMS

```
DECLARE   1 PAYROLL,
          2 NAME CHARACTER (50) VARYING,
          2 HOURS,
            3 REGULAR FIXED,
            3 OVERTIME FIXED,
          2 RATE FLOAT;
```

TREE NAMES

PAYROLL. HOURS. REGULAR

DEFAULT NAMES IF UNAMBIGUOUS

PAYROLL. REGULAR
HOURS. REGULAR
REGULAR

SCOPE ATTRIBUTES

SCOPE - INTERNAL OR EXTERNAL

INTERNAL - KNOWN ONLY WITHIN THE BLOCK IT IS DECLARED

EXTERNAL - GLOBALLY KNOWN

EXTERNAL PROCEDURE NAME - LIKE FORTRAN SUBROUTINE NAME

A PL/I PROGRAM CONSISTS OF A GROUP OF EXTERNAL PROCEDURES

EXTERNAL DATA NAME - LIKE COMMON IN FORTRAN

STORAGE ALLOCATION ATTRIBUTES

PL/I HAS FORTRAN, ALGOL AND LIST PROCESSING MODES OF STORAGE ALLOCATION.

FORTRAN MODE - STATIC

LIFETIME OF STATIC STRUCTURES IS THE WHOLE COMPUTATION

ALGOL MODE - AUTOMATIC

LIFETIME OF AUTOMATIC STRUCTURES IS THE BLOCK ON WHICH THEY ARE DECLARED.

LIST PROCESSING MODE - CONTROLLED

A CONTROLLED STORAGE ALLOCATION DECLARATION CREATES A TEMPLATE FOR THE DECLARED STRUCTURE.

INSTANCES OF A STRUCTURE CREATED BY A CONTROLLED STORAGE ALLOCATION DECLARATION ARE CREATED BY ON ALLOCATE COMMAND AND DELETED BY A FREE COMMAND.

CONTROLLED STORAGE ALLOCATION

```
DECLARE 1  A CONTROLLED
           2 X FIXED
           2 Y POINTER
```

TEMPLATE:	X:	FIXED
	Y:	POINTER

```
ALLOCATE  A
A·X  = 5
  X  = X + 1
A·Y  = ADDR(Z)
ALLOCATE  A
FREE      A
FREE      A
```

MULTIPLE ALLOCATION CAUSES AUTOMATIC STACKING OF INSTANCES.

ACCESS TO INSTANCES THROUGH POINTERS

```
ALLOCATE A SET P
ALLOCATE A SET Q
P → A·X = 5
Q → X = 6
```

BASED STORAGE ALLOCATION

DECLARE 1 A BASED (P)

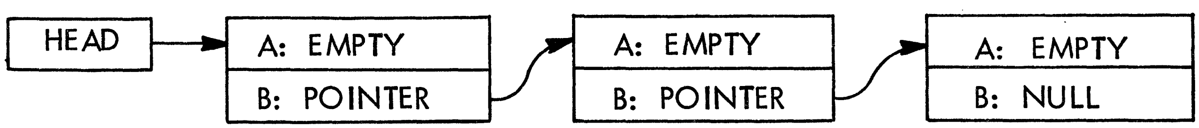
2 X FIXED

2 Y POINTER

ALLOCATE A

ALLOCATE A

WHEN SECOND COPY IS NESTED, FIRST COPY IS DESTROYED.
CREATE THE FOLLOWING THREE-ELEMENT LIST.




```
DECLARE (Q, HEAD) POINTER;
DECLARE 1 ELEMENT BASED (P),
          2 A FIXED,
          2 B POINTER;
ALLOCATE ELEMENT;
HEAD = P;
Q = P;
ALLOCATE ELEMENT;
Q → B = P;
Q = P;
ALLOCATE ELEMENT;
Q → B = P;
B = NULL;
```

This program declares Q and HEAD to be of type POINTER and ELEMENT to be a structure based on P. The instruction "ALLOCATE ELEMENT" automatically sets P to the most recent instance of ELEMENT. The assignment statements "HEAD = P; Q = P;" assign the value of the pointer P to the pointers Q, HEAD. When the second instance of ELEMENT has been created, Q points to the first instance, and "Q → B = P" sets the pointer B of the first instance to point to the second instance. Similarly after creation of the third instance of ELEMENT, "Q → B = P" sets the pointer B in the second instance to point to the third instance. Finally "B = NULL", which is equivalent to "P → B = NULL" sets the current instance of B to the special pointer value NULL, which indicates the end of the list.

LISP LIST PROCESSING OPERATIONS IN PL/I

STRUCTURE DECLARATION FOR LIST ELEMENT

```
DECLARE 1   LISPCELL BASED(P),
            2  CAR POINTER,
            2  CDR POINTER,
            2  MODE BIT(6);
```

This declaration specifies the basic format of a list cell in LISP to consist of two pointer fields named CAR and CDR and a 6-bit mode field.

HEAD AND TAIL OPERATIONS

```
HEAD:   PROCEDURE(P) POINTER;
        DECLARE 1   ELEMENT
                      BASED(P),
                    2 CAR
                      POINTER,
                    2 CDR
                      POINTER;
        RETURN(CAR);
TAIL:   ENTRY(P);
        RETURN(CDR);
        END HEAD;
```

This pointer-valued procedure has two entry points, HEAD and TAIL. The declaration of ELEMENT specifies the structure pointed to by the procedure parameter P. The structure itself is assumed to have been created outside the procedure and to be an element of a list of structures of the kind arising in LISP. The call HEAD(P) returns with a value given by the pointer in the first field of the structure pointed to by P while the call TAIL(P) returns with a value given by the second field of the structure pointed to be P.

CONS OPERATOR

```
CONS:   PROCEDURE(P,Q) POINTER;
        DECLARE 1   ELEMENT
                      BASED(X),
                    2 LEFT
                      POINTER,
                    2 RIGHT
                      POINTER;
        ALLOCATE ELEMENT;
        LEFT  =  P;
        RIGHT  =  Q;
        RETURN(X);
        END CONS;
```

This pointer-valued procedure has two pointer-valued parameters, P and Q. It allocates an instance of the structure ELEMENT, stores the pointers P and Q in the first and second registers of the newly created structure, and delivers a pointer to the newly created structure as its value.

FEATURES WHICH FACILITATE LIST PROCESSING

VARIABLES OF TYPE POINTER WHICH ALLOW LINKS BETWEEN INFORMATION STRUCTURES TO BE EXPLICITLY SPECIFIED AND MANIPULATED

STRUCTURE DECLARATIONS WHICH ALLOW LIST ELEMENTS CONTAINING SEVERAL POINTER AND VALUE FIELDS OF DIFFERENT TYPES TO BE EXPLICITLY DECLARED

CONTROLLED STORAGE ALLOCATION, WHICH ALLOWS STRUCTURES TO BE DYNAMICALLY CREATED AND DELETED AS THEY ARE REQUIRED.

IN A GIVEN LIST PROCESSING LANGUAGES ALL LIST STRUCTURES ARE FORMED OUT OF LIST ELEMENTS OF A LIMITED NUMBER OF PRIMITIVE TYPES

IN PL/I NEW PRIMITIVE TYPES OF LIST ELEMENTS MAY BE DEFINED BY STRUCTURE DECLARATIONS

IMPLEMENTATION OF CONTROLLED STORAGE ALLOCATION

CREATION AND DELETION IN UNPREDICTABLE ORDER

INSTANCES CANNOT BE STORED IN A STACK

FREE STORAGE AREA IS REQUIRED

ALLOCATE AND RETURN BLOCKS AS REQUIRED

FRAGMENTATION OF MEMORY

GARBAGE COLLECTION IS SOMETIMES NECESSARY

STATEMENT GROUPING IN FORTRAN, ALGOL AND PL/I

PURPOSES OF STATEMENT GROUPING

1. TO DELIMIT A PROCEDURE WHICH MAY BE CALLED IN SEVERAL PLACES
2. TO DELIMIT THE SCOPE OF NAMES
3. TO GROUP STATEMENTS FOR CONTROL PURPOSES
4. TO SPECIFY THE LIFETIME OF INFORMATION ITEMS

Purpose	FORTRAN	ALGOL	PL/I
1. Delimit procedures	Program unit	begin-end (procedure heading)	PROCEDURE-END
2. Scope of nomenclature	Program unit	begin-end	PROCEDURE-END BEGIN-END (INTERNAL EXTERNAL)
3. Unit for control purposes	DO-loop	begin-end (for clause)	BEGIN-END DO-END (DO-statement)
4. Lifetime of information	not needed	begin-end (own)	BEGIN-END for AUTOMATIC (STATIC AUTOMATIC CONTROLLED)

INTERRUPT FUNCTION MODULES

CONDITION PREFIXES

(ZERODIVIDE): L:X = A/B;

ON STATEMENT

ON CONDITION ACTION

LIKE A PROCEDURE DECLARATION

ENTRY WHEN (INTERRUPT) CONDITION OCCURS RATHER THAN BY EXPLICIT CALL - INTERRUPT FUNCTION MODULE

BEGIN BLOCKS - ENTRY AND EXIT IN LINE

PROCEDURE BLOCKS - CALL AND RETURN

ON MODULES - INTERRUPT AND RETURN

ENTRY AND EXIT FOR ALL THREE TYPES IS MUTUALLY IN A LAST IN FIRST OUT ORDER

ACTIVATION RECORDS MAY BE STORED IN A STACK

CHARACTERISTICS OF THE SYSTEM

- MULTI-USE
- ON-GOING DATA BASE
- COMMON DATA BASE
- JOB LIBRARY
- PREREQUISITE SCHEDULING
- REAL-TIME SCHEDULING

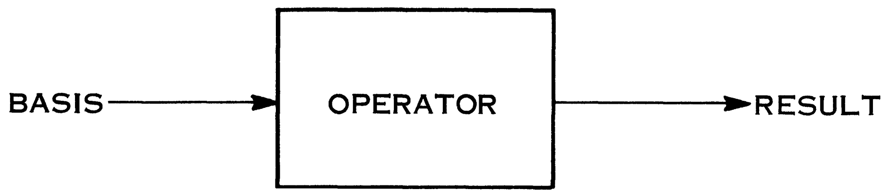
THE COHERENT SYSTEM CONCEPT

- COHERENCE OF PROGRAMS
- COHERENCE OF DATA
- COHERENCE OF CONTROL
- RESPONSIBILITY FOR COHERENCE:

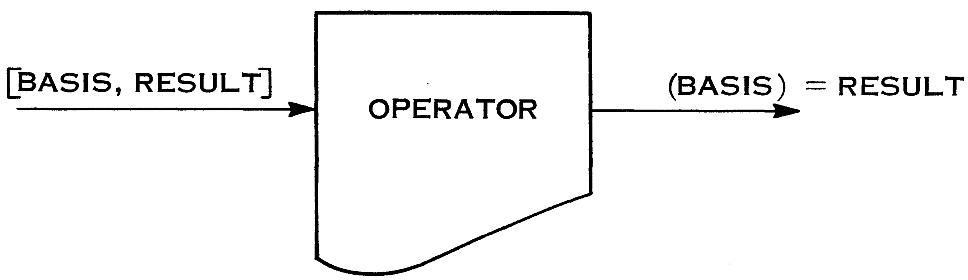
  PROGRAMMER
  PROGRAM TRANSLATORS
  SYSTEM

DATA TRANSFORMATION FUNCTIONS

BY PROGRAM



BY TABLE



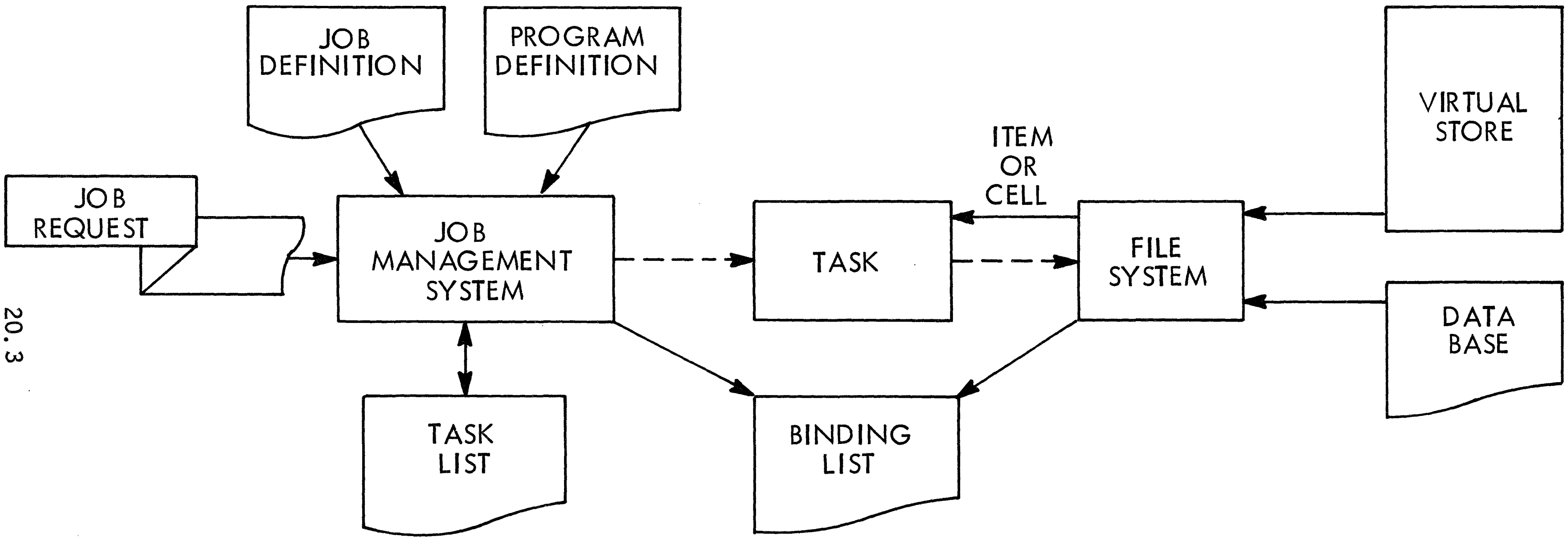
FUNCTIONAL NOTATION

OPERATOR (BASIS) = RESULT

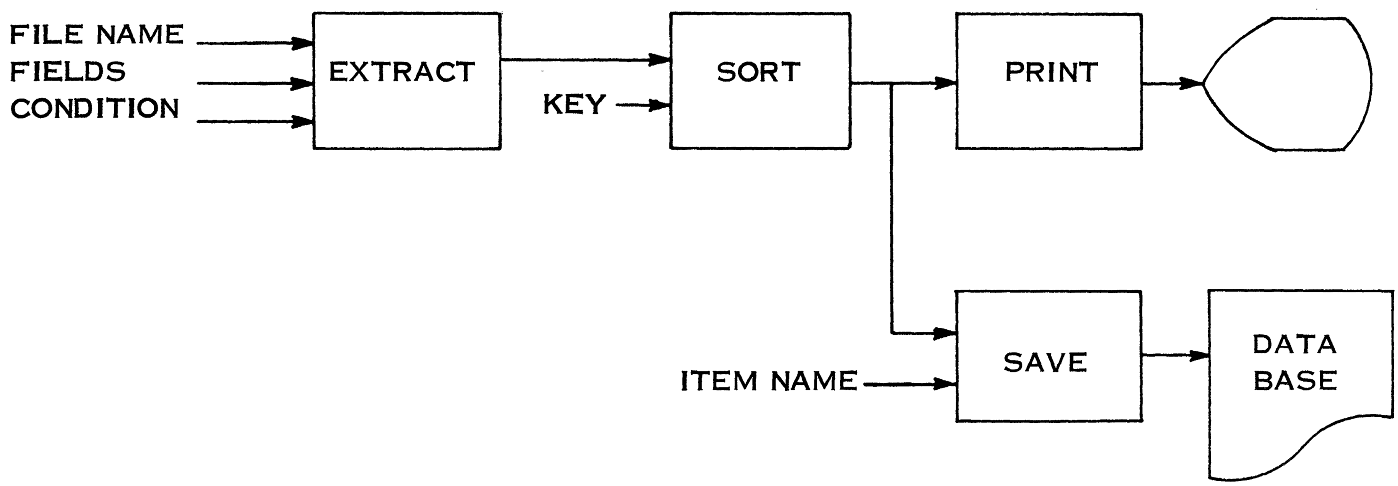
F (X) = Y

SQRT (4) = 2

INTERFACE BETWEEN TASK, JOB MANAGEMENT SYSTEM, AND FILE SYSTEM



JOB DEFINITION EXAMPLE



DEFINE JOB : PERSONNEL LIST (CONDITION, ITEM NAME)

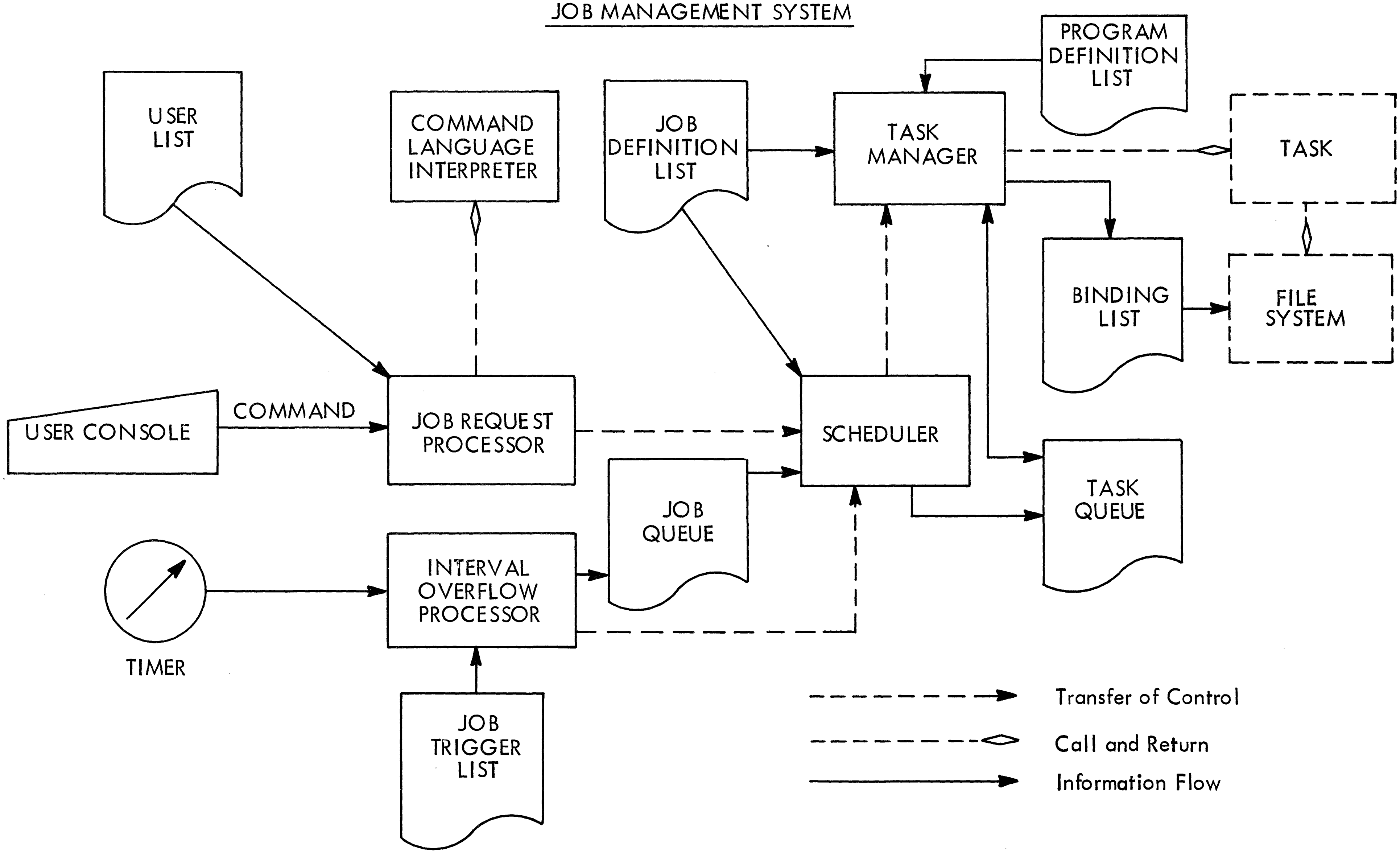
EXTRACT (PERSONNEL FILE, (NAME, EMPL. NO, POS), CONDITION) = *1

SORT (*1, NAME) = * 2

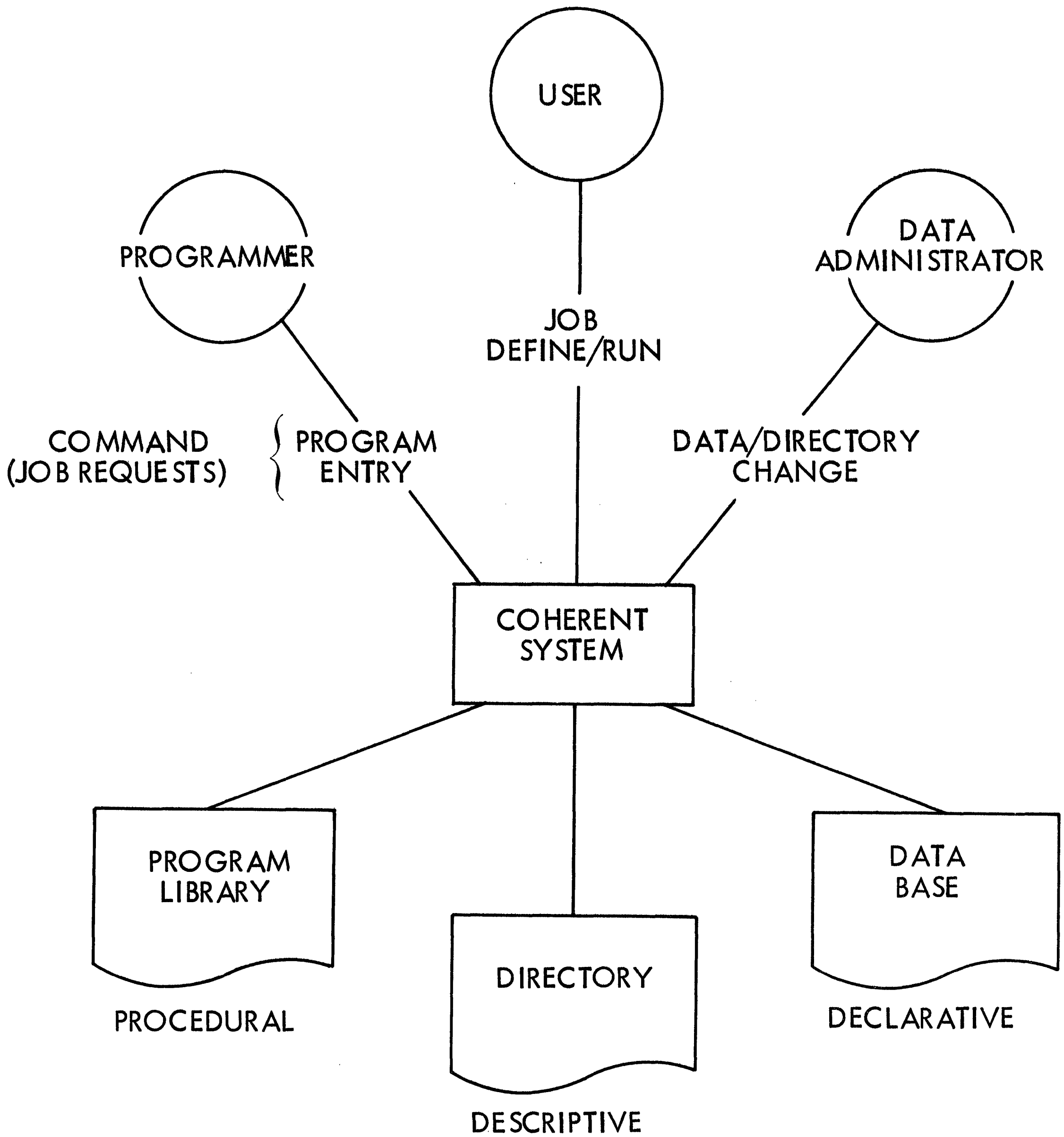
SAVE (*2, ITEM NAME)

PRINT (*2).

JOB MANAGEMENT SYSTEM



USER INTERFACE AND SYSTEM LANGUAGES



USER LANGUAGES

- JOB REQUEST (COMMAND) LANGUAGE
- DATA ITEM DEFINITION LANGUAGE
- DATA ITEM INPUT LANGUAGE
- JOB DESCRIPTION LANGUAGE
- DATA SERVICE REQUEST LANGUAGE
- ON-LINE (INTERPRETIVE) COMPUTATIONAL LANGUAGES
- COMPILER LANGUAGE
- MACRO ASSEMBLER LANGUAGE

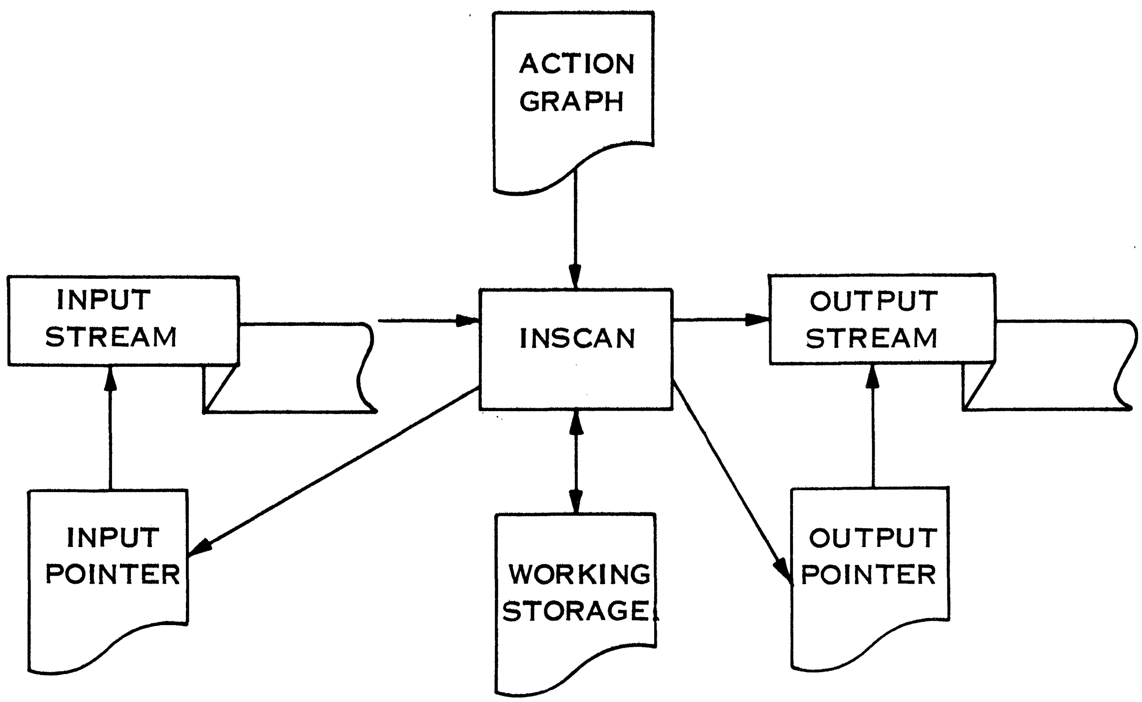
THE NEED FOR LANGUAGE ADAPTABILITY

- CHANGES IN CAPABILITY
- CHANGES IN USERS
- HUMAN FACTORS EXPERIMENTS

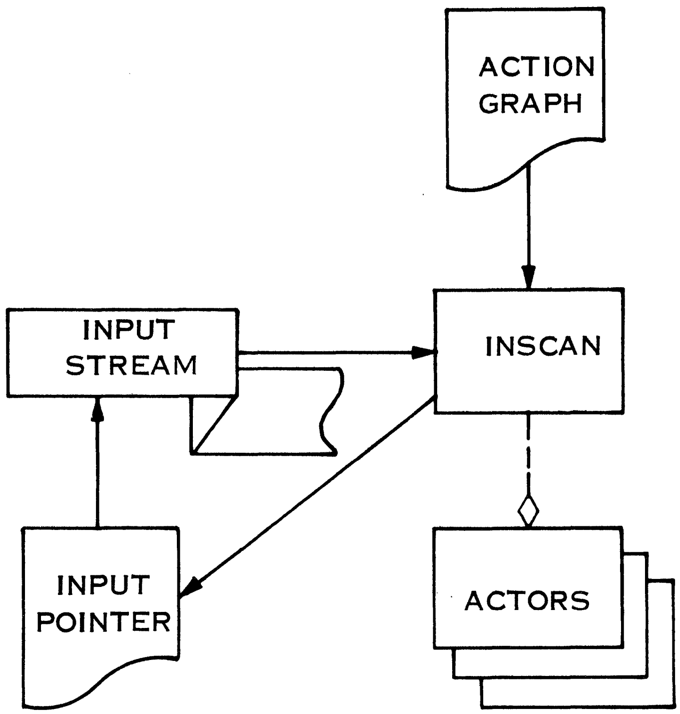
SYNTAX DIRECTED PROCESSING

- SYNTAX
- SEMANTICS
- SCANNER/ANALYZER

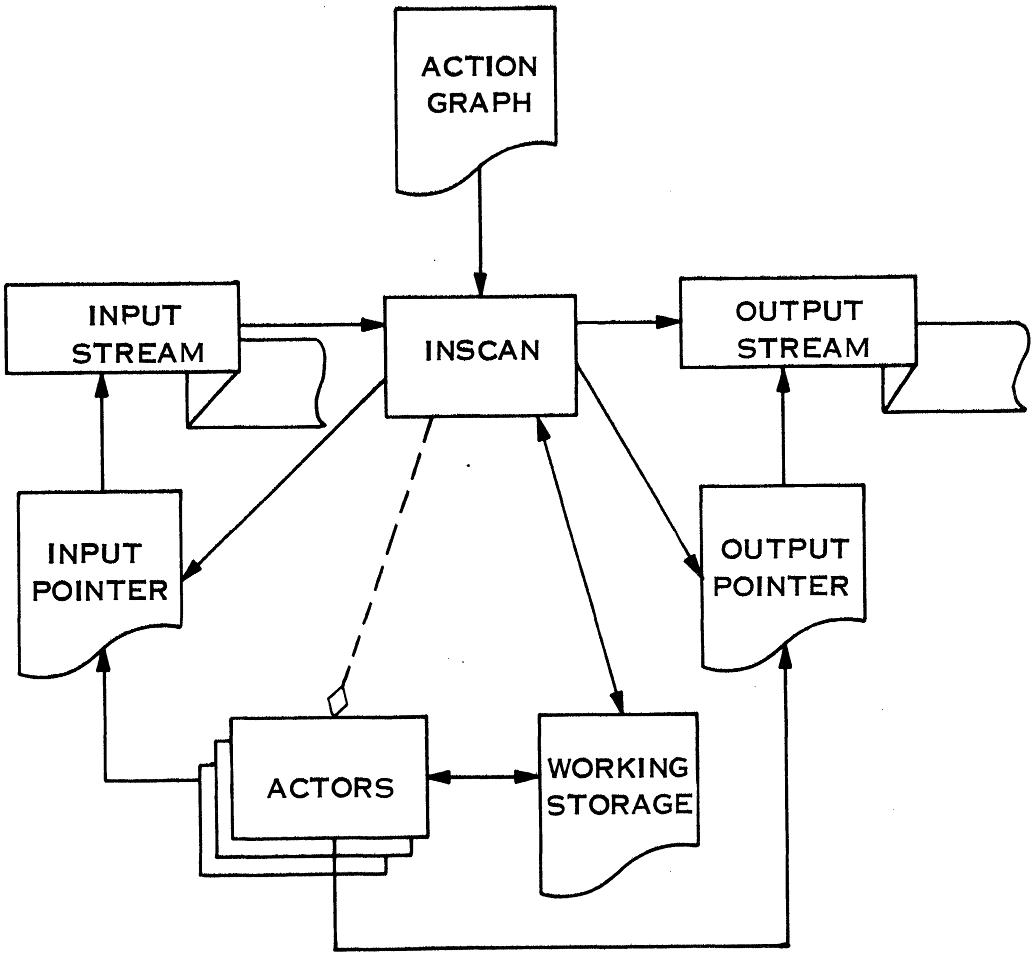
INSCAN: TRANSLATION MODE



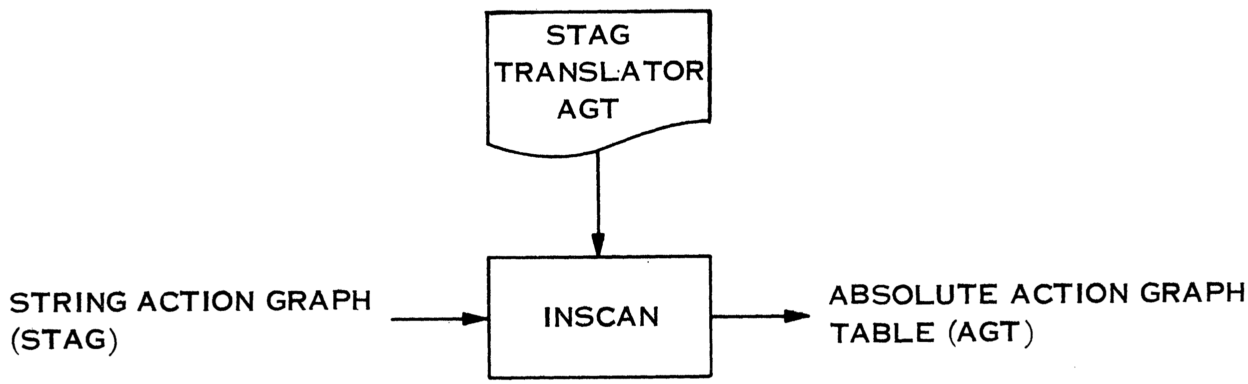
INSCAN: INTERPRETIVE MODE



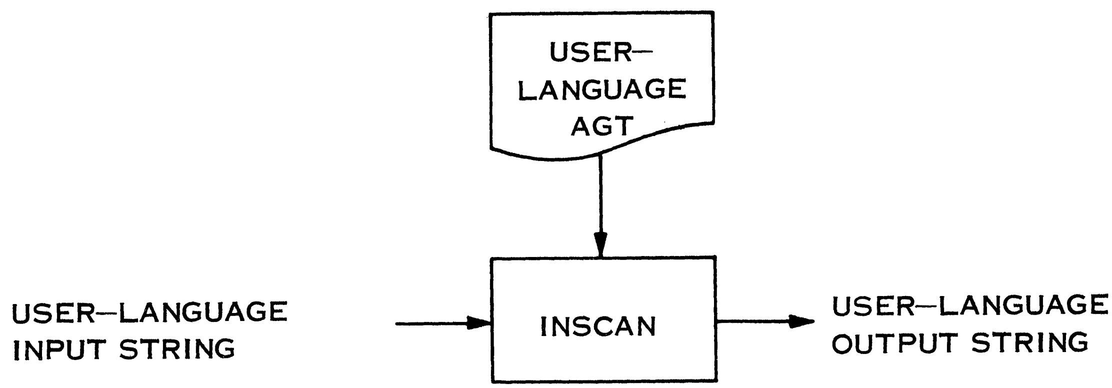
GENERAL INSCAN CONFIGURATION



PHASES OF USER LANGUAGE PROCESSING



(A) "ASSEMBLY" TIME

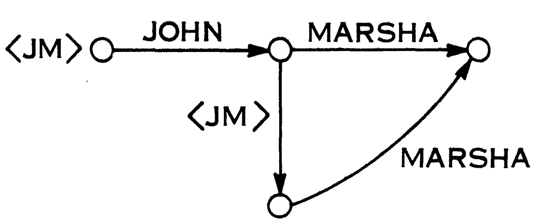


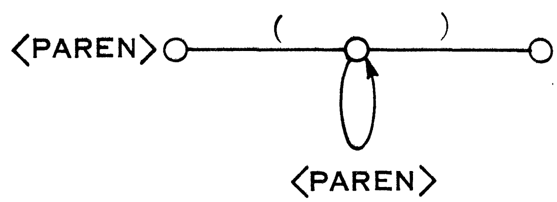
(B) "EXECUTION" TIME

EXAMPLES OF SYNTAX SPECIFICATIONS

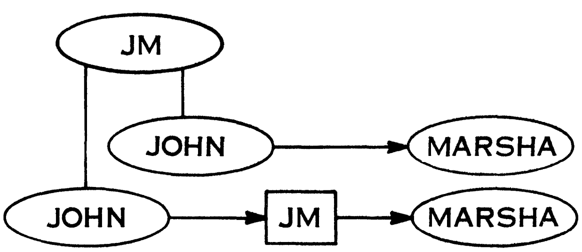
JM	PAREN
⟨JM⟩:: = JOHN MARSHA | JOHN ⟨JM⟩ MARSHA	⟨PAREN⟩ :: = ( ) | (⟨PAREN STRING⟩) ⟨PAREN STRING⟩::=⟨PAREN⟩ | ⟨PAREN⟩⟨PAREN STRING⟩

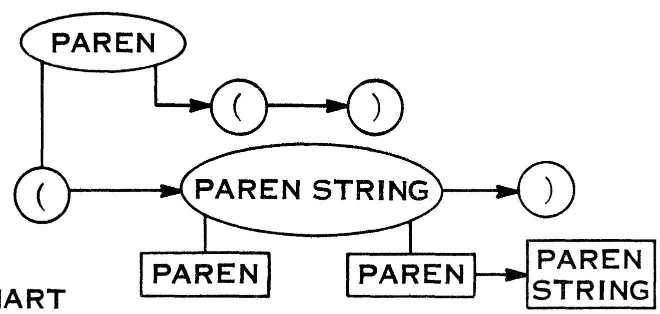
(A) BNF



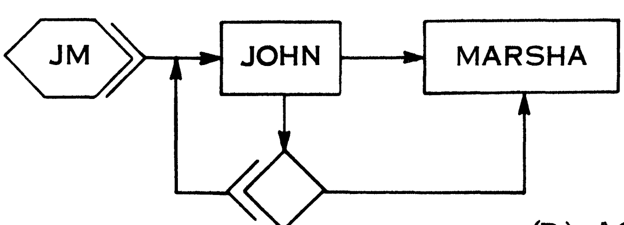


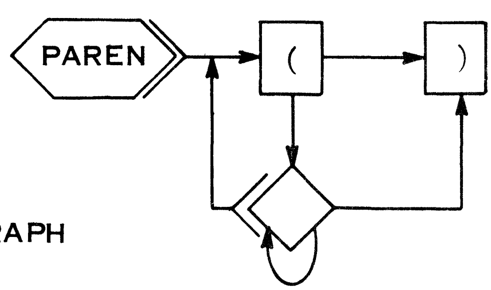
(B) TRANSITION DIAGRAM





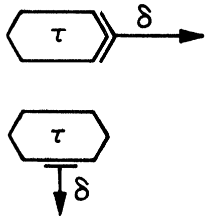
(C) SYNTAX CHART



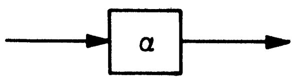


(D) ACTION GRAPH

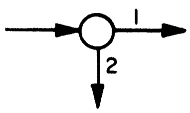
ACTION GRAPH SYMBOLS



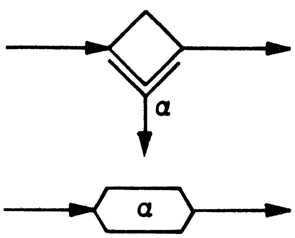
GRAPH NAME: τ IS DEFINED BY PATH δ



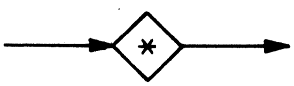
SCAN: READ INPUT SYMBOL AND MATCH α



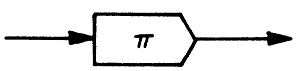
CHOICE: TRY ALTERNATIVES 1 AND 2



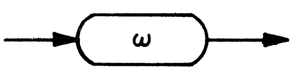
SUBGRAPH: EXECUTE GRAPH α AND RETURN



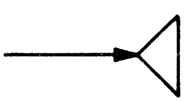
RECURSE: EXECUTE THIS GRAPH RECURSIVELY AND RETURN



EXTERNAL ACTION: DO SUBROUTINE π AND RETURN

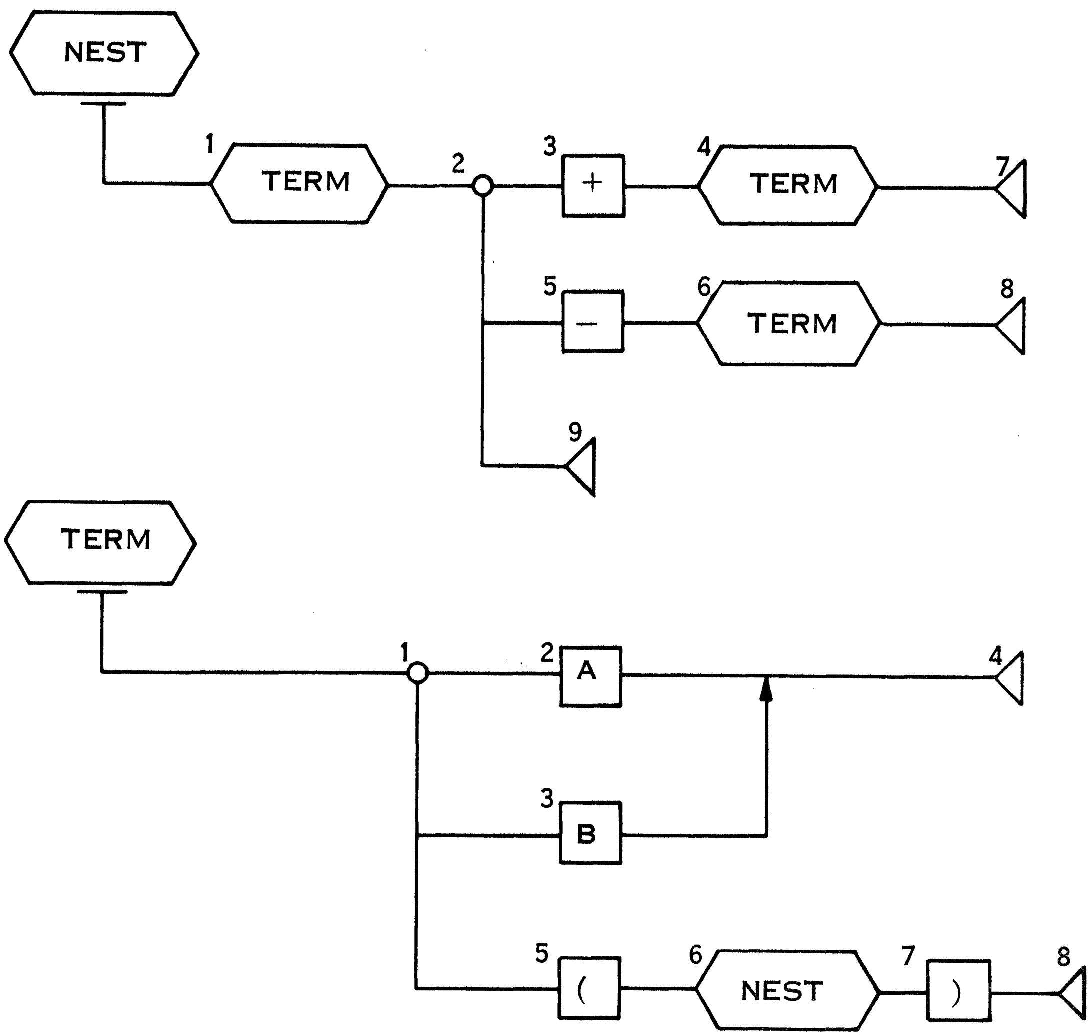


INTERNAL ACTION: DO OPERATIONS ω



END: RETURN TO PARENT GRAPH

INFIX RECOGNIZER



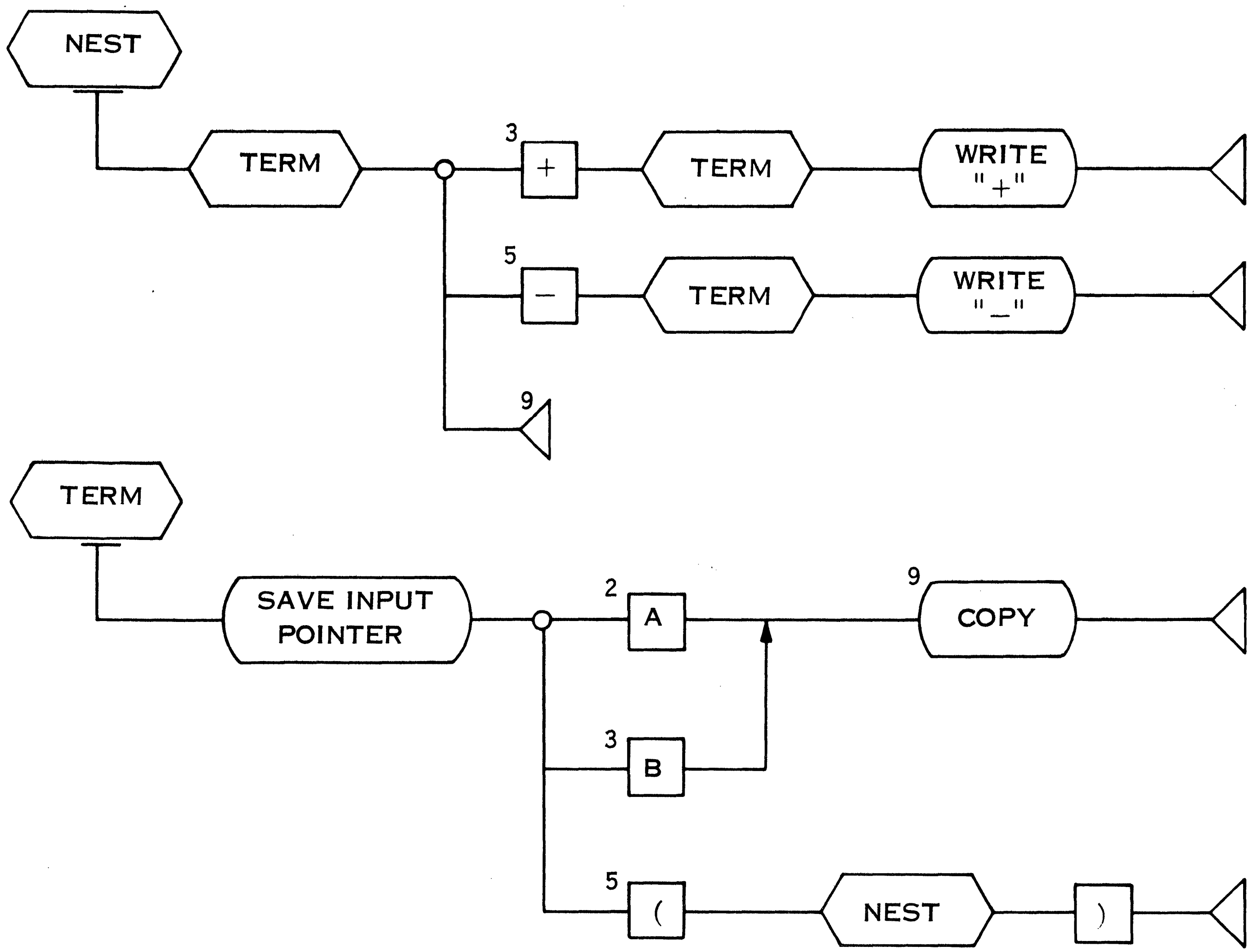
SAMPLE INPUT STRING FOR THE INFIX RECOGNIZER

(	A	+	B	)	—	(	A	—	B	)
1	2	3	4	5	6	7	8	9	10	11

OPERATION OF THE INFIX RECOGNIZER

PORTION OF STRING RECOGNIZED	RECOGNIZED BY
A	TERM
B	TERM
A + B	NEST
(A + B)	TERM
A	TERM
B	TERM
A − B	NEST
(A − B)	TERM
(A + B) − (A − B)	NEST

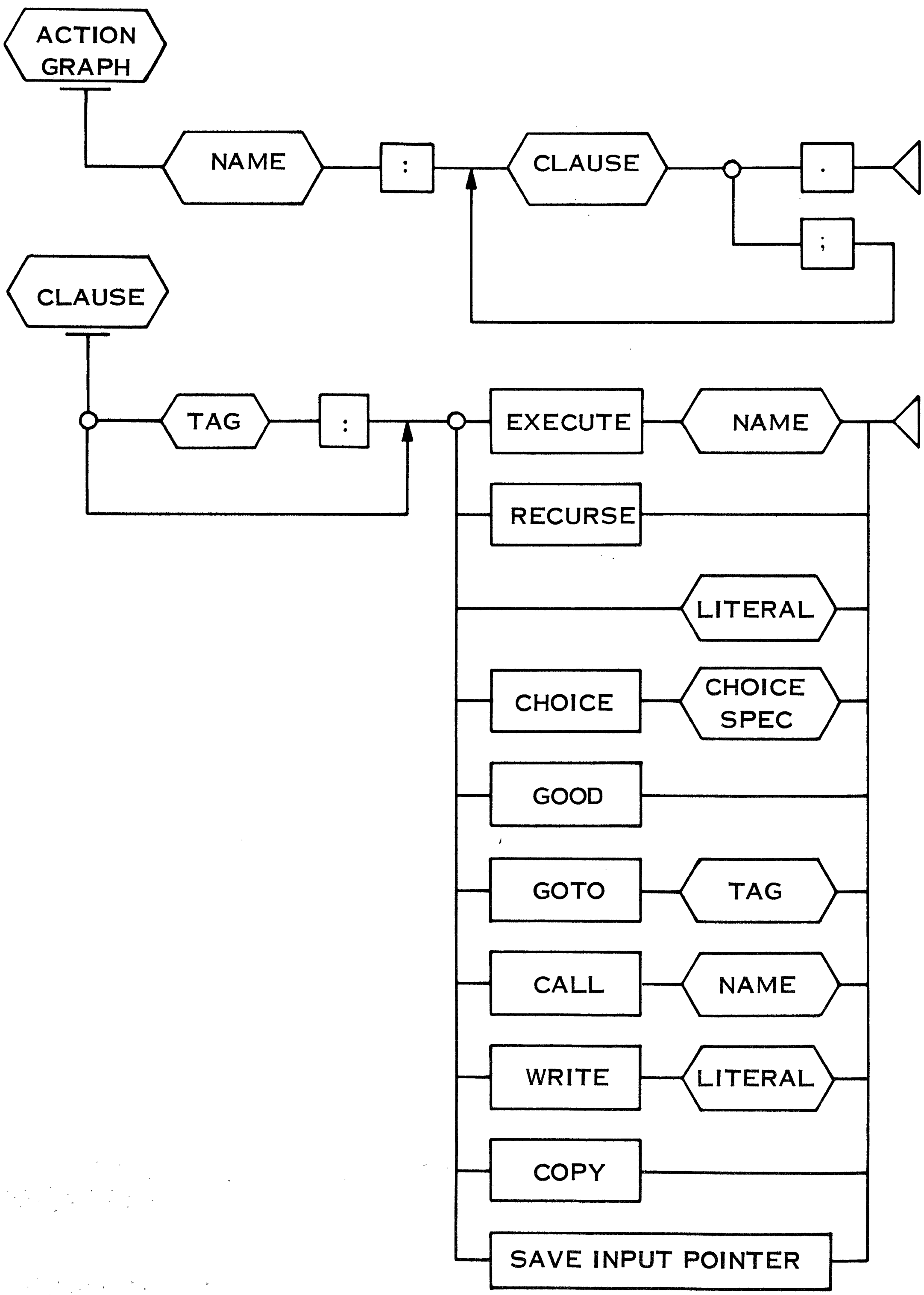
INFIX-TO-SUFFIX TRANSLATOR



OPERATION OF THE INFIX-TO-SUFFIX TRANSLATOR

PORTION OF STRING RECOGNIZED	RECOGNIZED BY	SYMBOL ADDED TO OUTPUT STRING
A	TERM	A
B	TERM	B
A + B	NEST	+
(A+ B)	TERM	
A	TERM	A
B	TERM	B
A – B	NEST	–
(A – B)	TERM	
(A + B) – (A – B)	NEST	–

STAG SYNTAX


ACTION GRAPH
NAME
:
CLAUSE
.
;
CLAUSE
TAG
:
EXECUTE
NAME
RECURSE
LITERAL
CHOICE
CHOICE SPEC
GOOD
GOTO
TAG
CALL
NAME
WRITE
LITERAL
COPY
SAVE INPUT POINTER

STAG - LANGUAGE ACTION GRAPHS FOR THE INFIX-TO-SUFFIX TRANSLATOR

NEST: EXECUTE TERM; CHOICE (3, 5, 9) ;

3: "+"; EXECUTE TERM; WRITE "+"; GOOD;

5: "–"; EXECUTE TERM; WRITE "–";

9: GOOD.

TERM: SAVE INPUT POINTER; CHOICE (2, 3, 5) ;

2: "A" ; 9: COPY; GOOD;

3: "B"; GOTO 9;

5: "("; EXECUTE NEST ;")" ; GOOD.