Floating Point Systems, Inc.,
Corporate Training Department Materials

FPS-164 System Overview and FORTRAN User's Materials

FPS-164 Software Programming Class
Revision 2.0 July 1984

FPS-164 FORTRAN USER'S COURSE OUTLINE

Day 1

- Introduction and course overview
- FPS' Product introduction (AP family desc.)
- FPS164 Hardware and Software Overview
- The Program Development sequence
- APFTN64 - Use and Features
- Lab 1: APFTN64 and executing a program on the FPS-164
- Introduction to the Systems Job Executive (SJE)
- Lab 2: Using Basic SJE commands

Day 2

- Quiz and Review
- Host Data Conversion Utilities
- Lab 3: Using SJE with the data conversion utilities
- Details of the permanent file system (FMS)
- Lab 4: Using SJE with data conversion and the permanent file system
- SJE system backup and restore
- Program Debugging with APDEBUG64
- Lab 5: Using APDEBUG64
- Program Conversion
- Lab 6: Writing an APEX64 subroutine in ADC mode

Day 3

- Quiz and Review
- Program Timing Utilities
- Lab 7: Subroutine timing
- Details of APEX64
- Lab 8: Accessing an APEX64 subroutine in UDC mode
- Problem Reporting (the T.A.R.)
- FPService and technical newsletters
- Course Evaluation
- (Optional Topics)

July 23, 1984

38-BIT PRODUCTS

1970 - FPS INCORPORATED.
MADE FLOATING POINT H/W

1974 - AP-120B, FOR MINI'S
- 38-BIT ACCURACY (8 D.D.)
- 6 MHz CLOCK ↔ 167 nsec.
- 10 OPS./CYCLE ⇒ 60 MIPS
  ⇒ 12 MFLOPS

1976 - AP-190L, FOR MAIN'S

1980 - FPS-100, FOR MINI'S
- 38-BIT ACCURACY
- 4 MHz CLOCK ↔ 250 nsec

1981 - AP-180V, FOR VAX + DR780

38-BIT PRODUCTS

1983- FPS-5000 FAMILY

- CONTROL PROCESSOR BASED UPON EARLIER PRODUCTS
- OPTIONAL ARITHMETIC COPROCESSORS:
  - 32-BIT ACCURACY
  - 18 MFLOPS EACH

64-BIT PRODUCTS

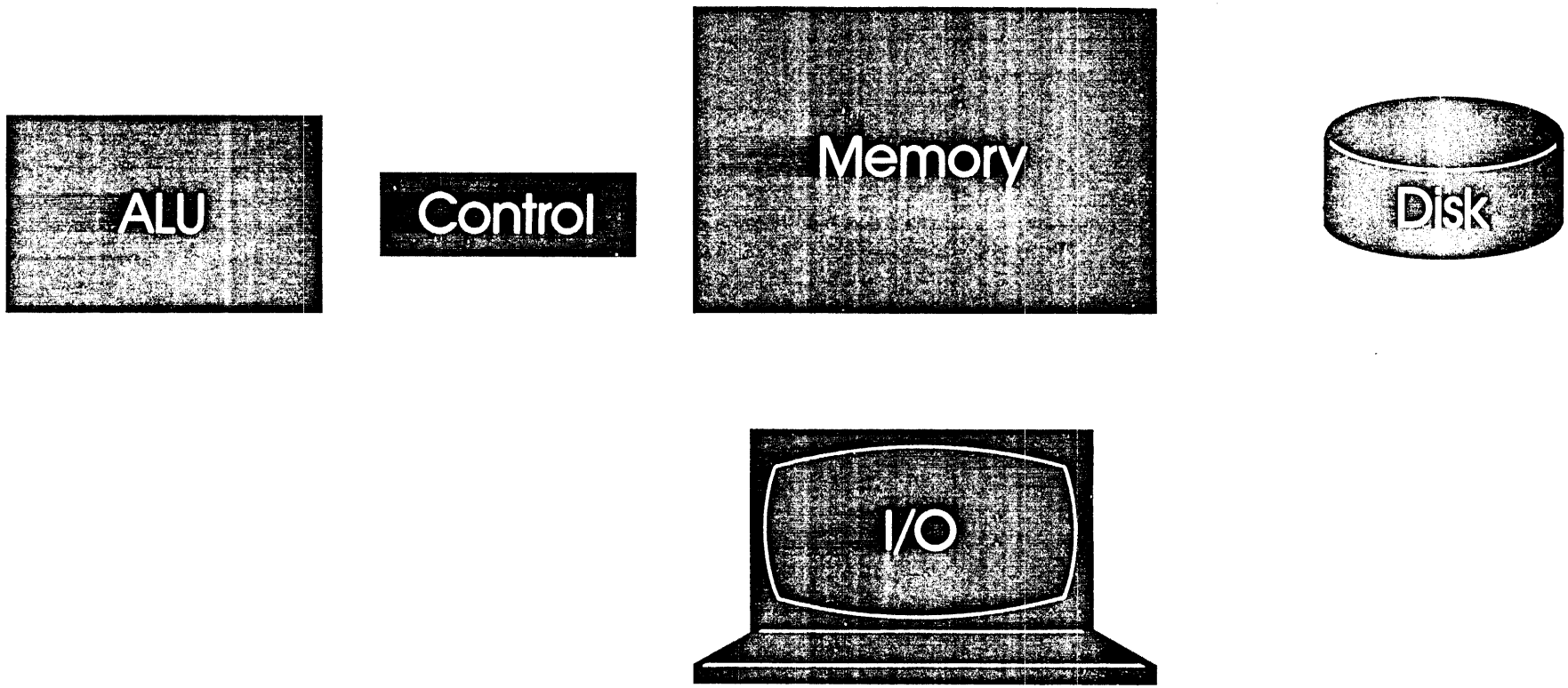
1981 - FPS-164

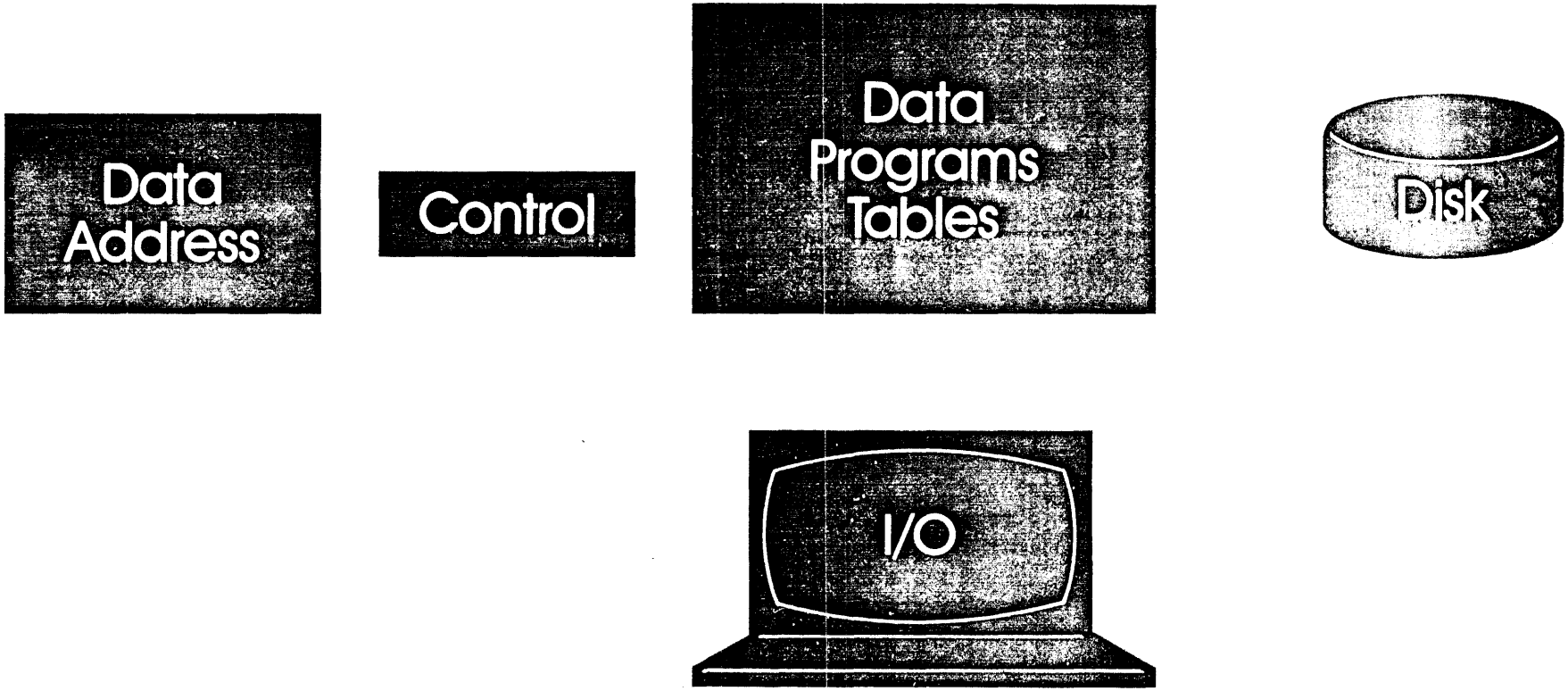
- "ATTACHED SCIENTIFIC COMPUTER" WITH ON-BOARD DIAGNOSTICS
- 64-BIT ACCURACY ⇒ 15.3 DECIMAL DIGITS
- 182 nSEC/CYCLE
- 55 MIPS OR 11 MFLOPS
- OPTIONAL H/W ALLOWS UP TO 341 MFLOPS (FPS-164/MAX)

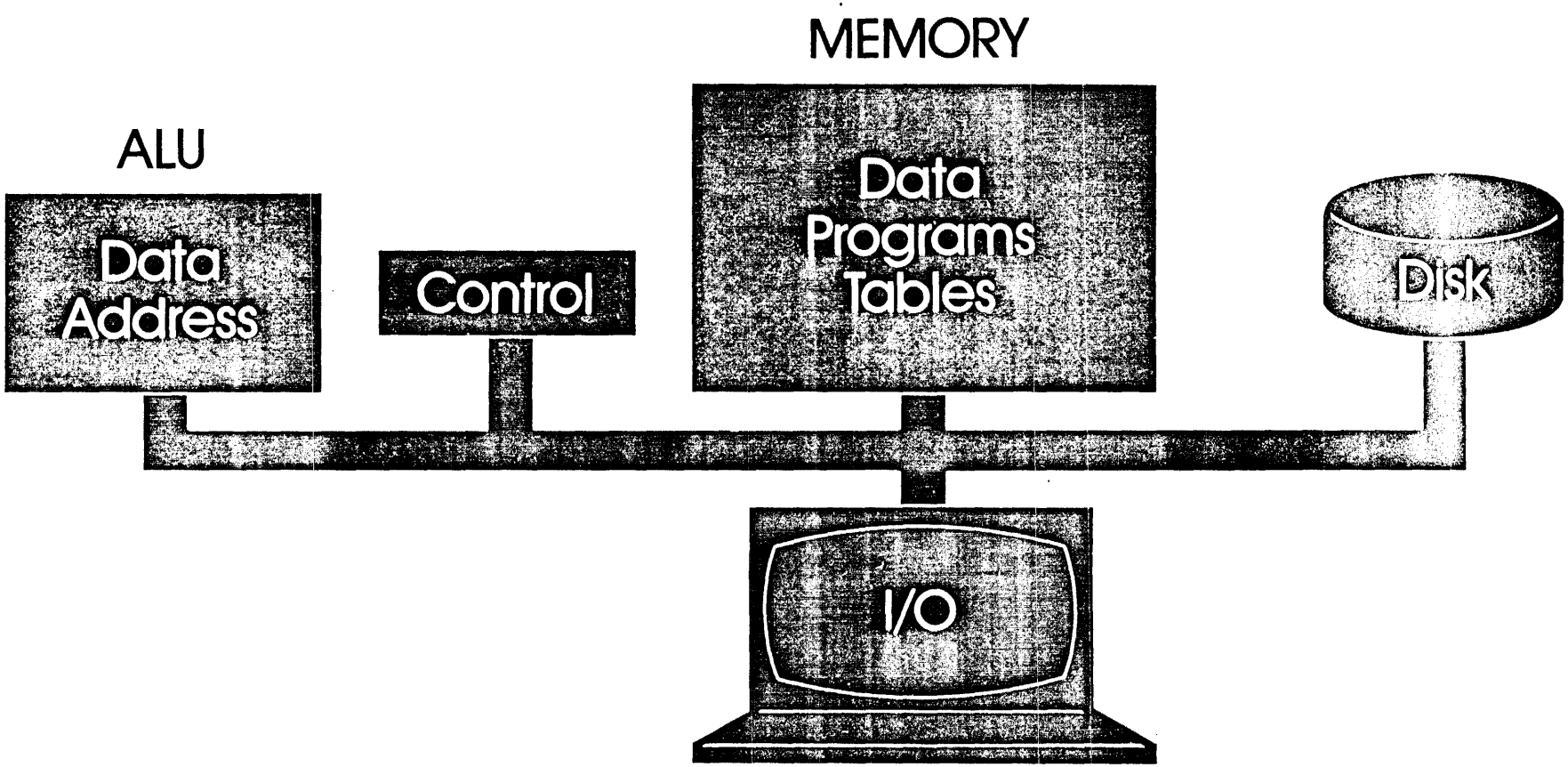
PRESENTATION SUMMARY

Part I	Traditional Hardware	Floating Point Systems Architecture and Characteristics
Part II	Traditional Software	Floating Point Systems AP Programming Techniques and Software Support

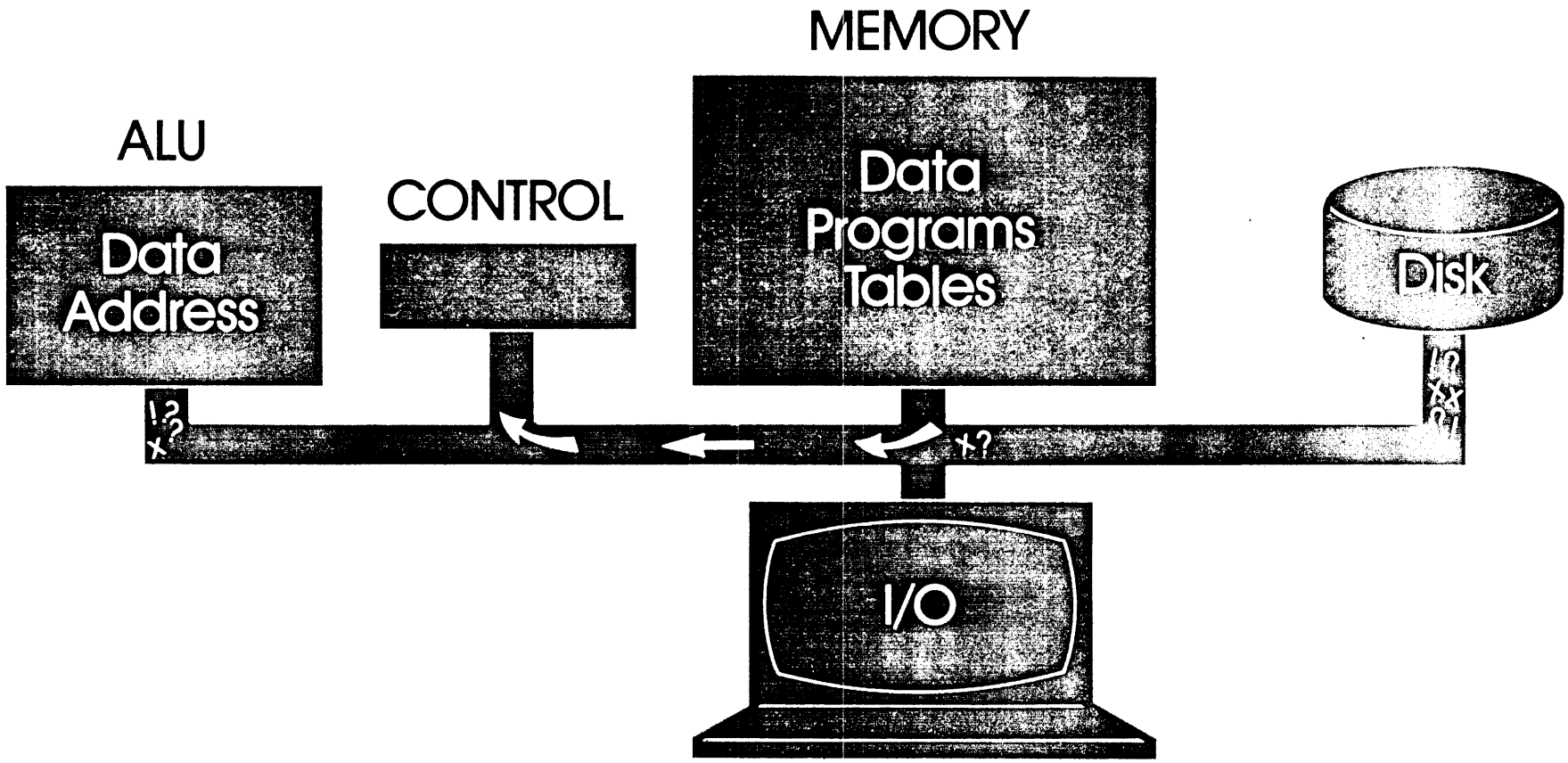
TYPICAL COMPUTER ELEMENTS




Data
Address
Control
Data
Programs
Tables
Disk
I/O



Single bus provides a pathway for access to all parts.

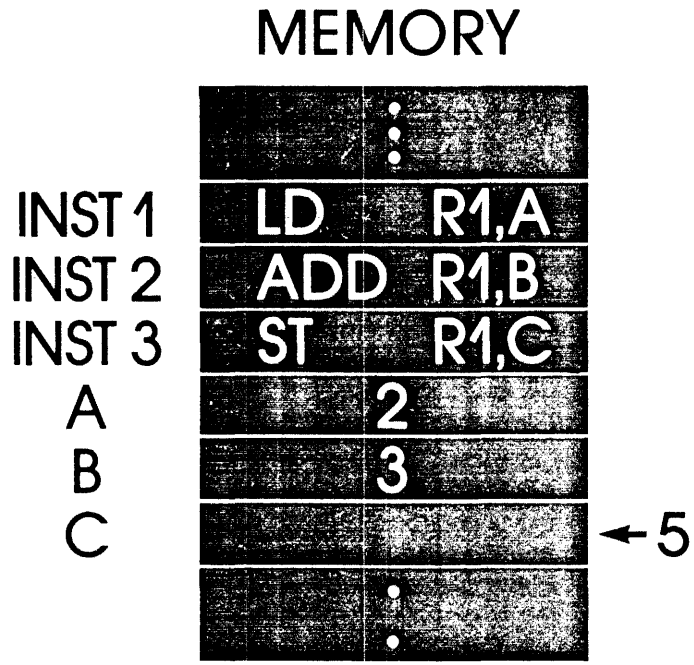


1. Each device must perform multiple functions.
2. Single bus limits access to all devices.

ADD TWO NUMBERS (Assembly Level)

C = A + B

```
LD    R1,A      "Load A into R1
ADD   R1,B      "Add A and B
ST    R1,C      "Store answer in C
```



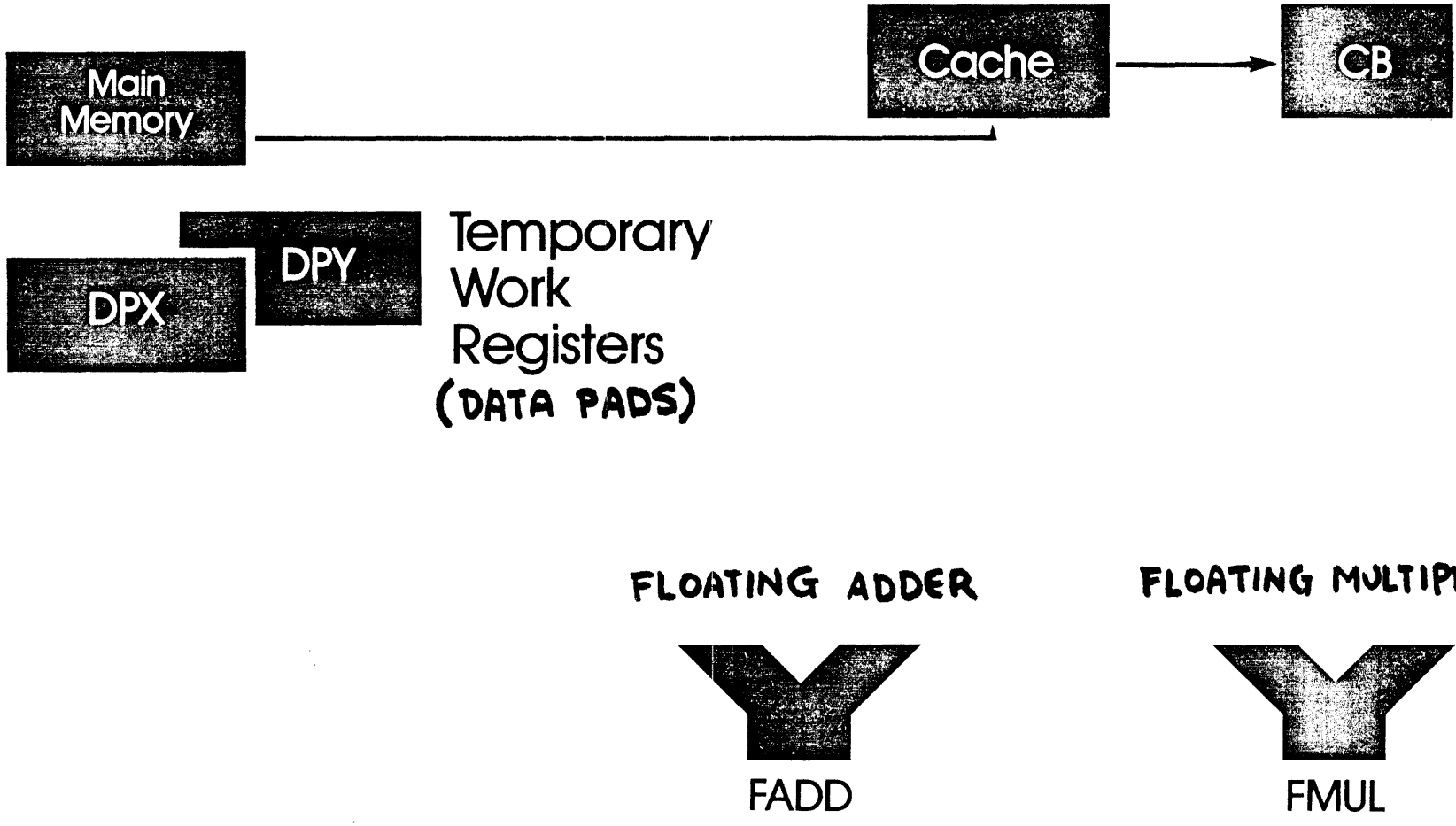
Fetch INST 1
Access Data A LD R1,A

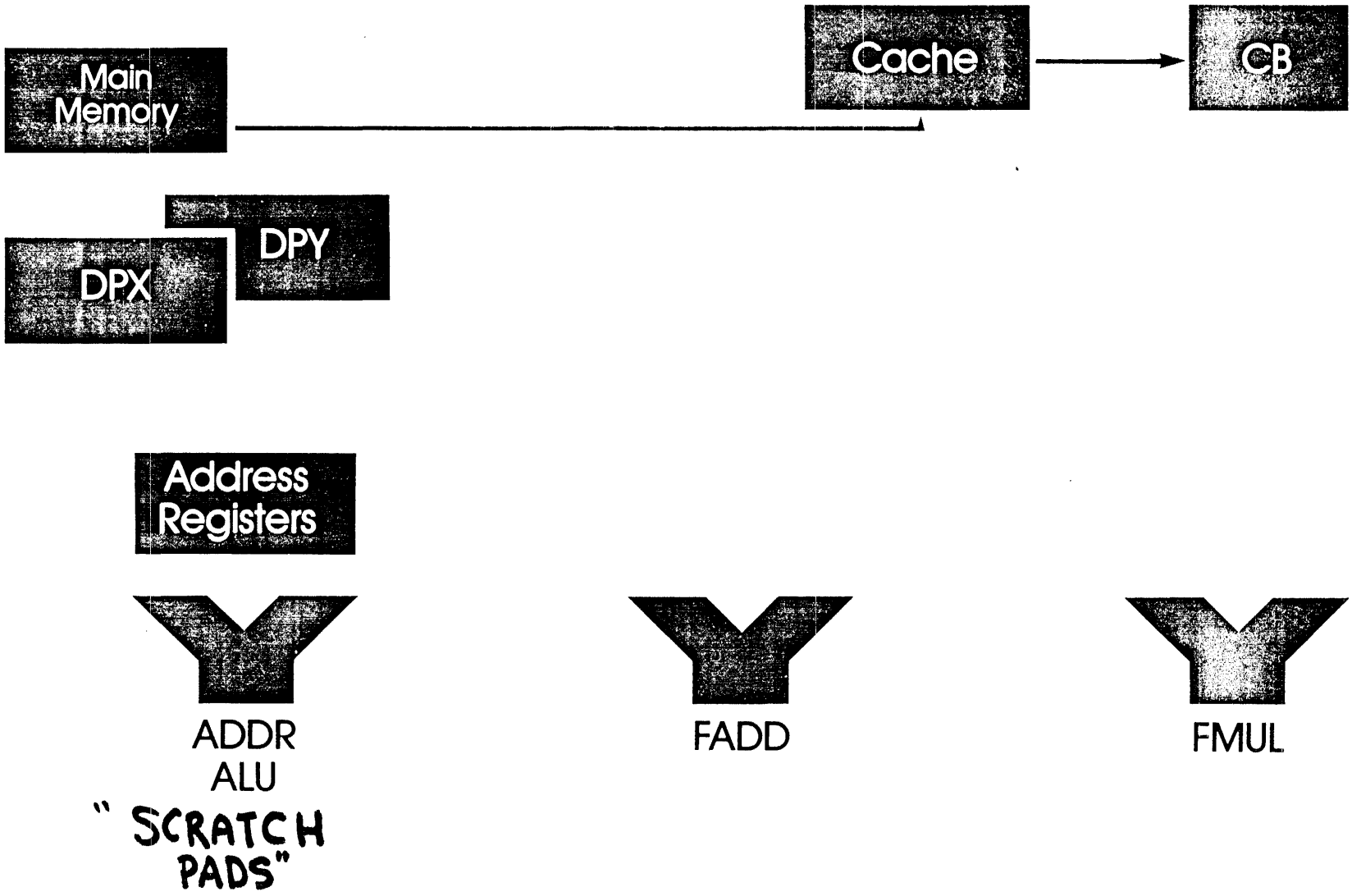
Fetch INST 2
Acess Data B ADD R1,B

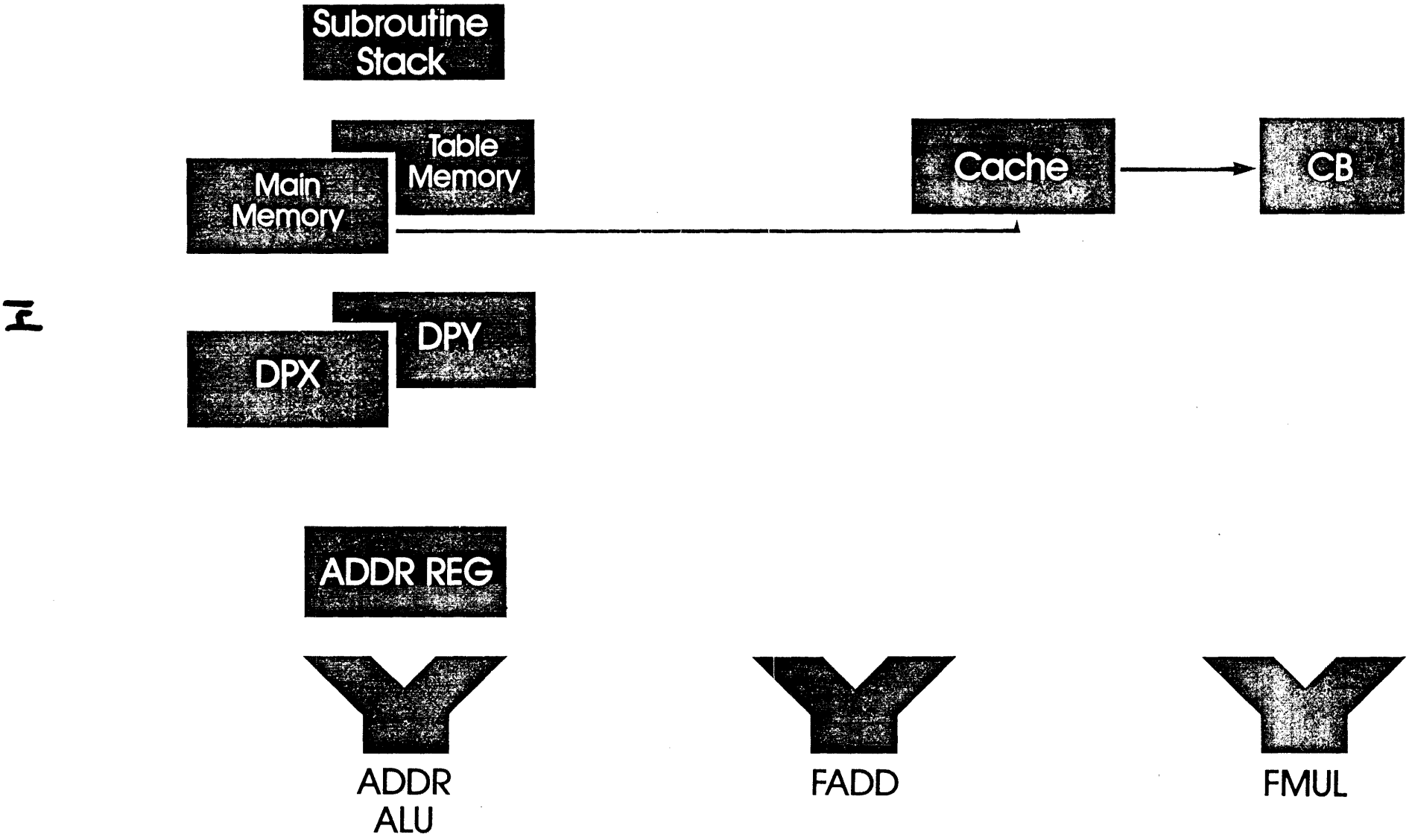
Fetch INST 3
Access Data C ST R1,C

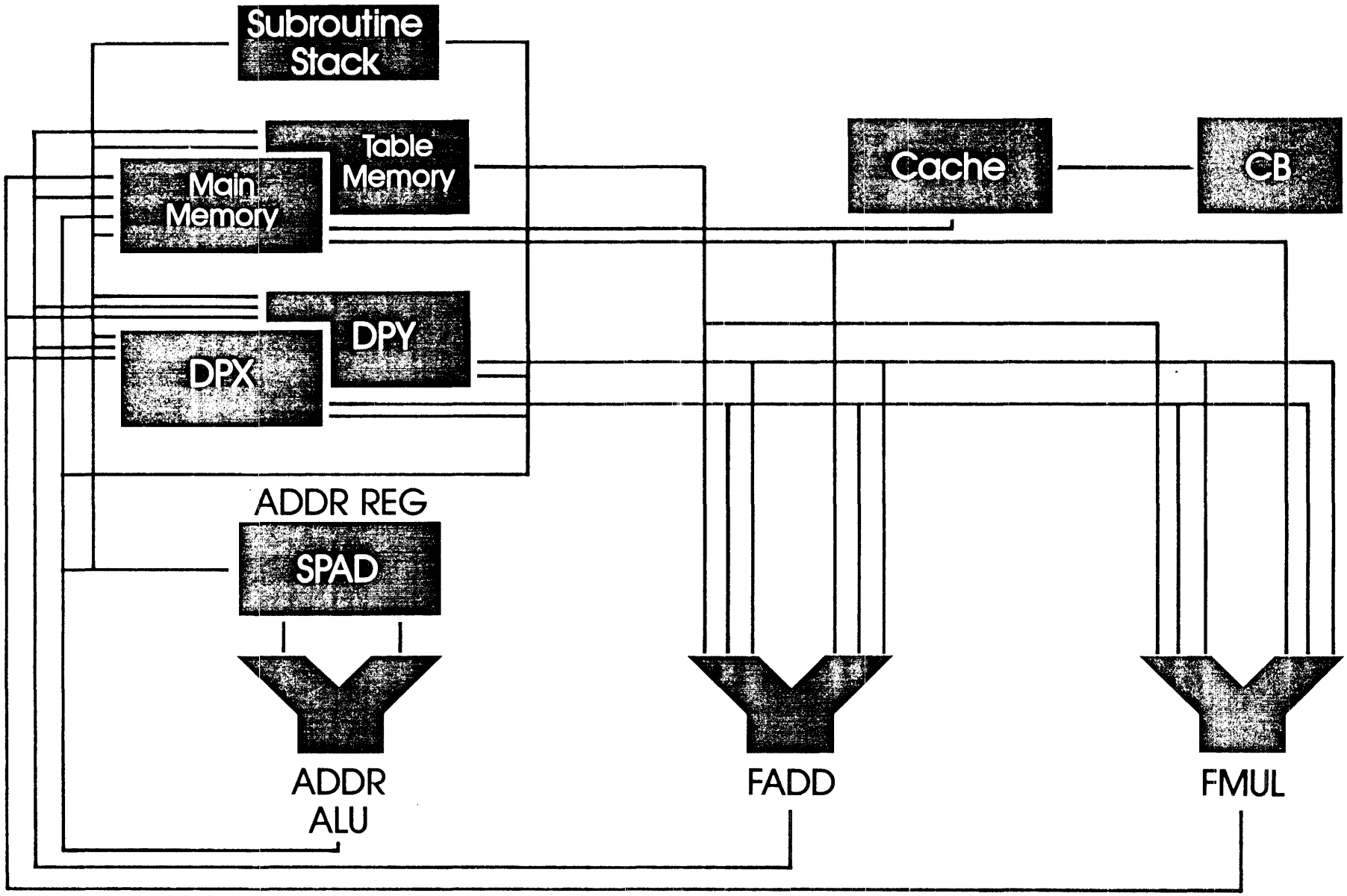
3 Data references + 3 instruction loads = 6 total

BUILD AN FPS-164 ATTACHED PROCESSOR

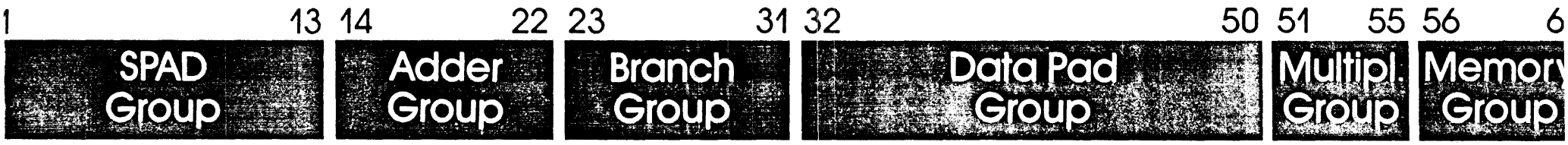

Main Memory
Cache
CB
DPX
DPY
Temporary Work Registers
(DATA PADS)
FLOATING ADDER
FADD
FLOATING MULTIPLIER
FMUL


Main Memory
Cache
CB
DPY
DPX
Address Registers
ADDR ALU
FADD
FMUL
" SCRATCH PADS"


Subroutine Stack
Table Memory
Main Memory
Cache
CB
DPY
DPX
ADDR REG
ADDR ALU
FADD
FMUL


Subroutine Stack
Table Memory
Main Memory
Cache
CB
DPY
DPX
ADDR REG
SPAD
ADDR ALU
FADD
FMUL

Primary Instruction Word Groups



GROUP	FUNCTIONAL UNIT CONTROLLED
SPAD	Address unit (SPAD)
Adder	Adder unit (FA)
Branch	Control unit (short branches)
Data Pad	Data registers (X/Y), Data Pad Bus (DPBS)
Multiplier	Multiplier (FM)
Memory	Main memory address, and store data source Table memory address Data Pad base address

INSTRUCTION SET OVERVIEW

Aggregate capability functional units per 182 nsec CPU cycle

- Two Data Computations
- Two Memory Accesses
- An Address Computation
- Four Data Register Accesses
- A Conditional Branch

FPS-164
D64 DISK SUBSYSTEM

- Consists of adapter, controller, plus drives
- 135MB Winchester drives
- Up to four drives per subsystem
- Up to six subsystems per 164 (3 Gigabytes)
- Required for SJE

FPS-164
D64 DISK SUBSYSTEM

• Rotational speed	3600 RPM
• Average latency	8.33 MS
• Average seek time	30 MS
• Tracks per cylinder	10
• Cylinders per drive	823
• Density	6220 BPI
• Sector size	512 B

DIAGNOSTIC PROCESSOR

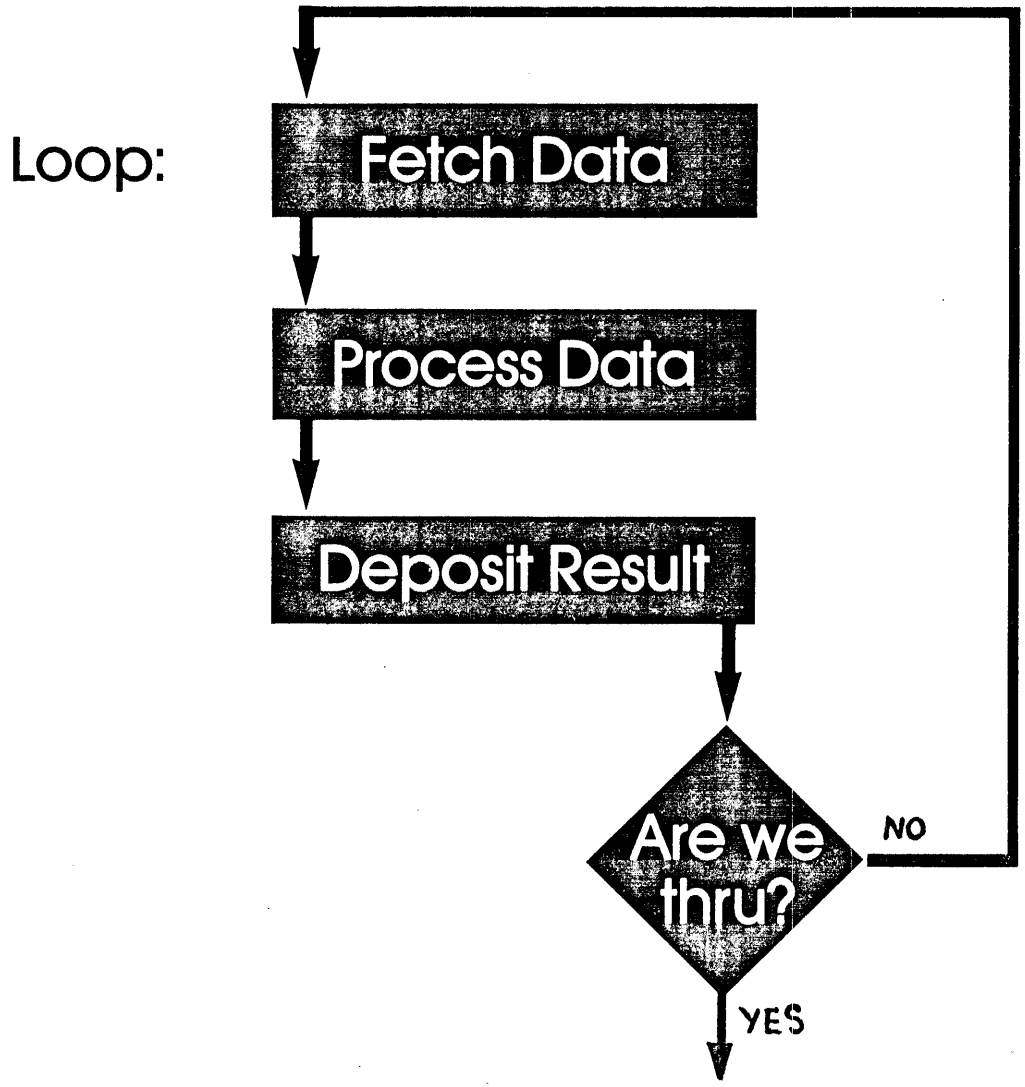
- Micro-processor and floppy disk
- Multiple-level diagnostic routines
- Independent Diagnostic Bus
- On-line logging of errors
- Board-level replacement strategy
- Remote diagnostic capability

REMOTE DIAGNOSTIC CAPABILITY

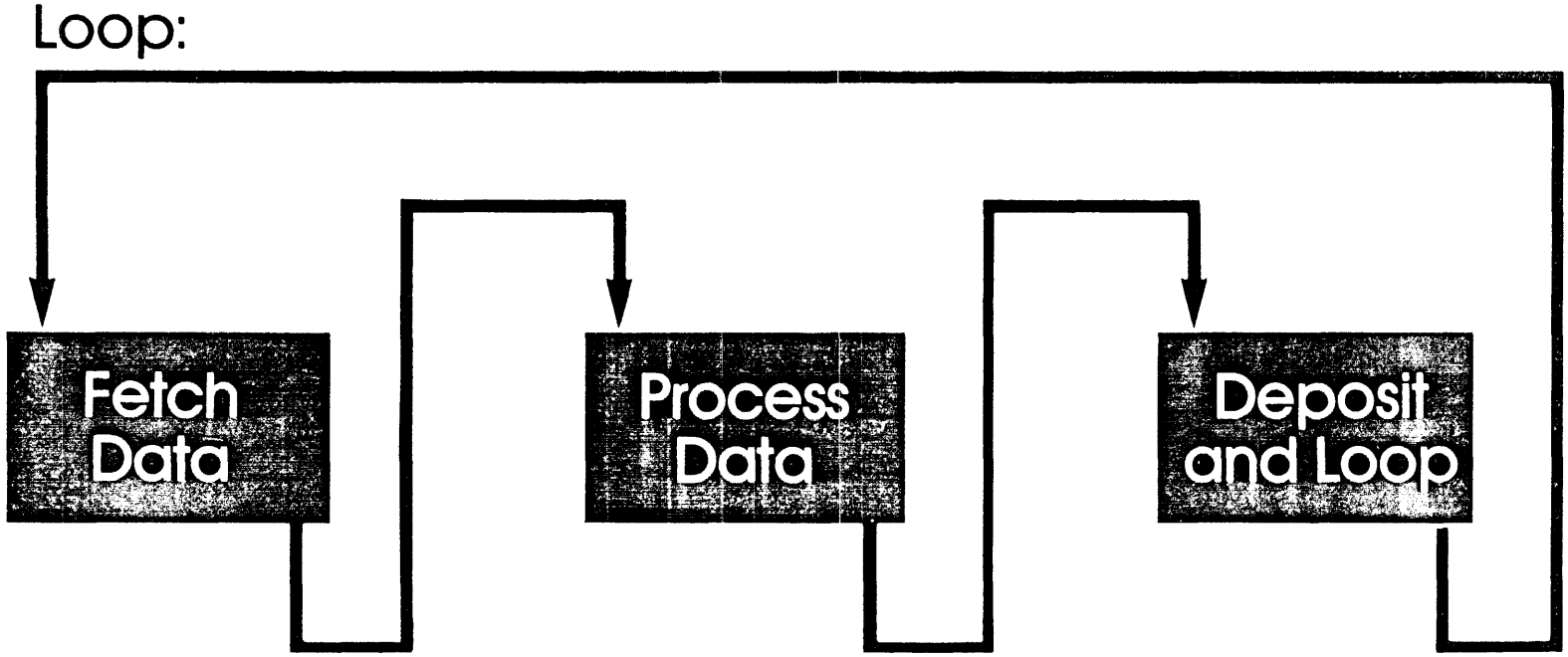
- FPS supplies VT101 Terminal
- FPS supplies modem
- Customer supplies phone
- Customer supplies RS232 cable
- For IBM . . . ASCII port must be defined in the operating system

SOFTWARE CONCEPTS

LOOP IN LINEAR FORM



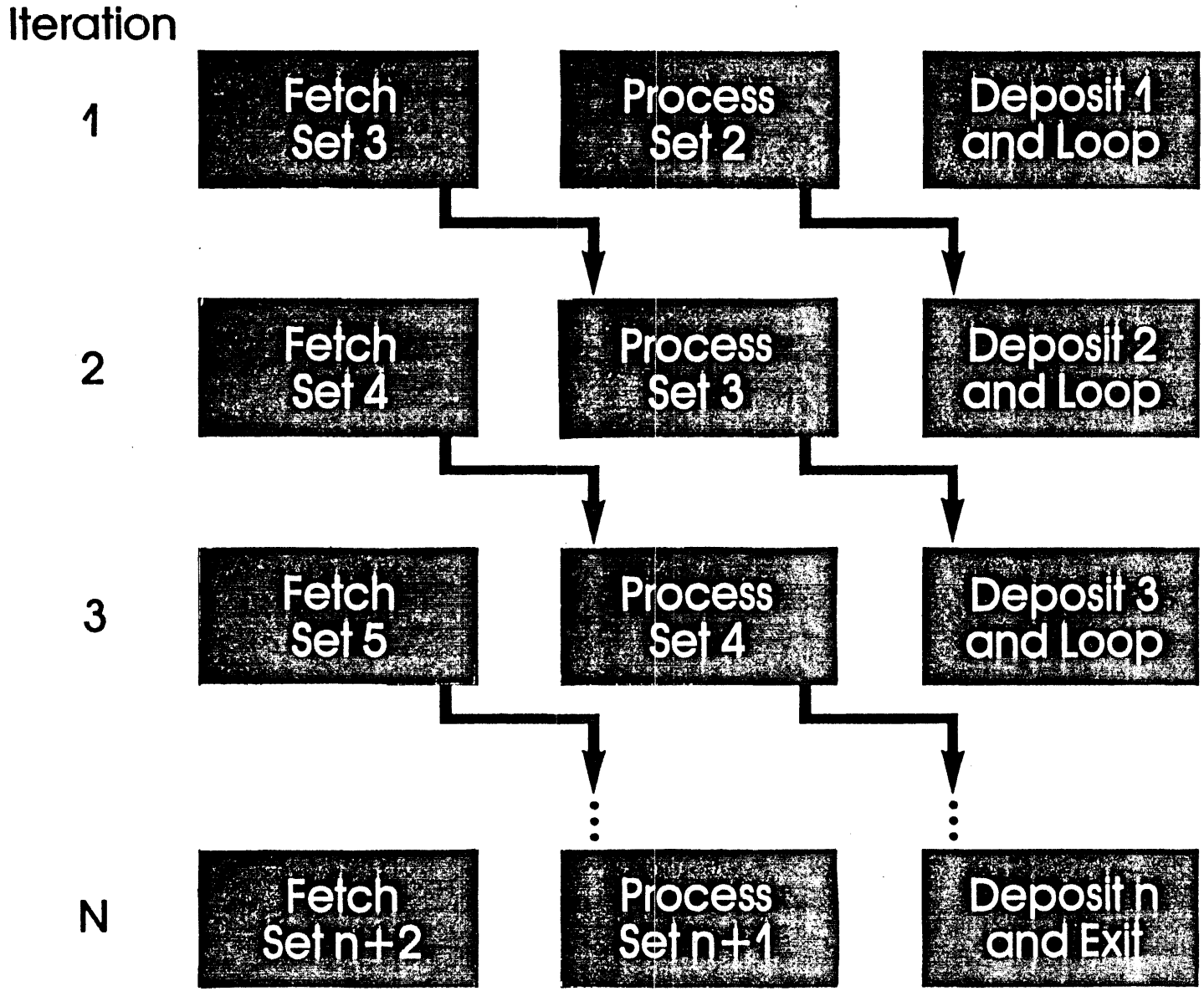
LOOP IN PARALLEL FORM



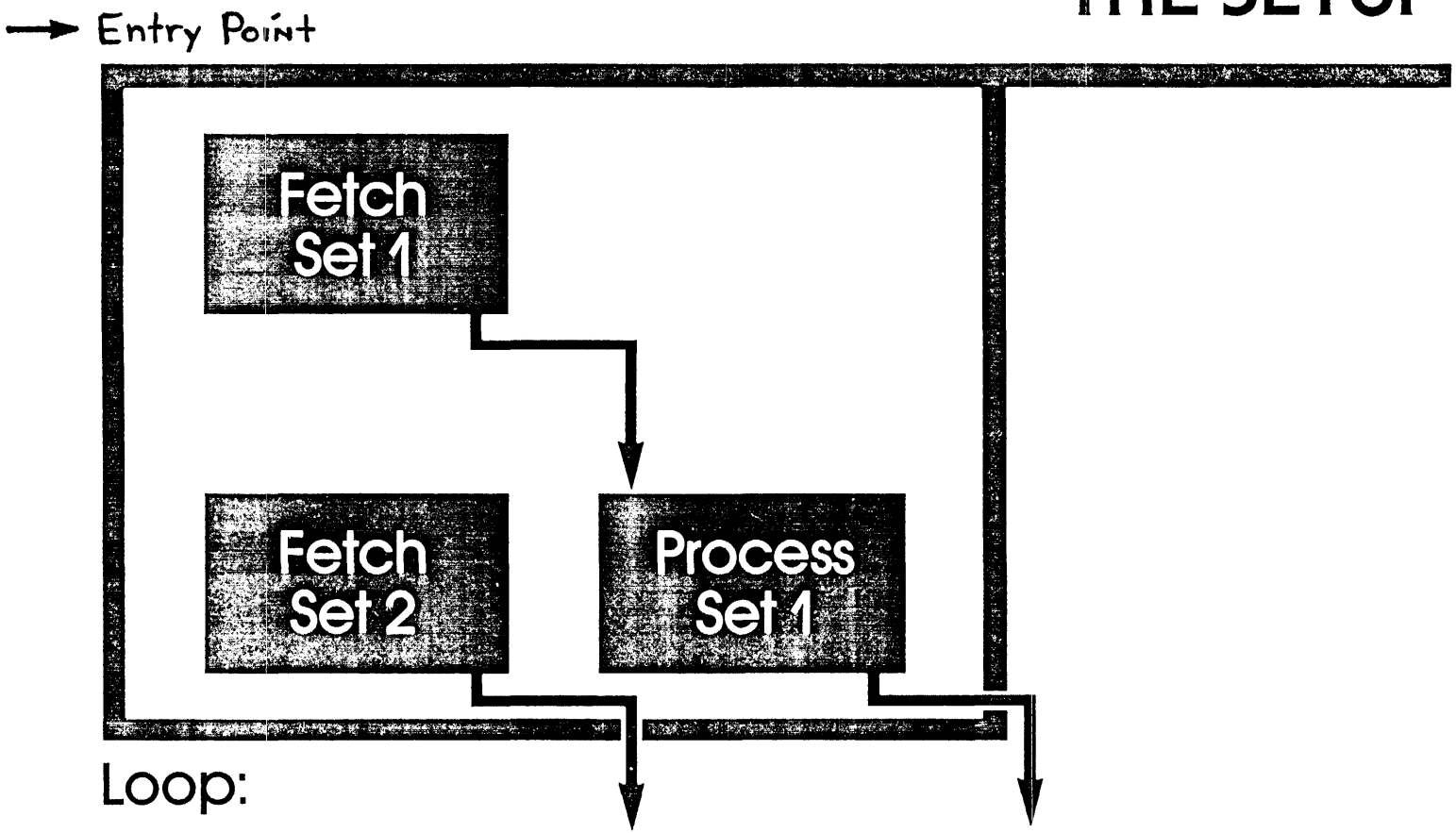
DEFINITION: SOFTWARE PIPELINE

A Software Pipeline is a software construct whereby multiple elements of an array are concurrently being processed, and each element is at a different stage of processing.

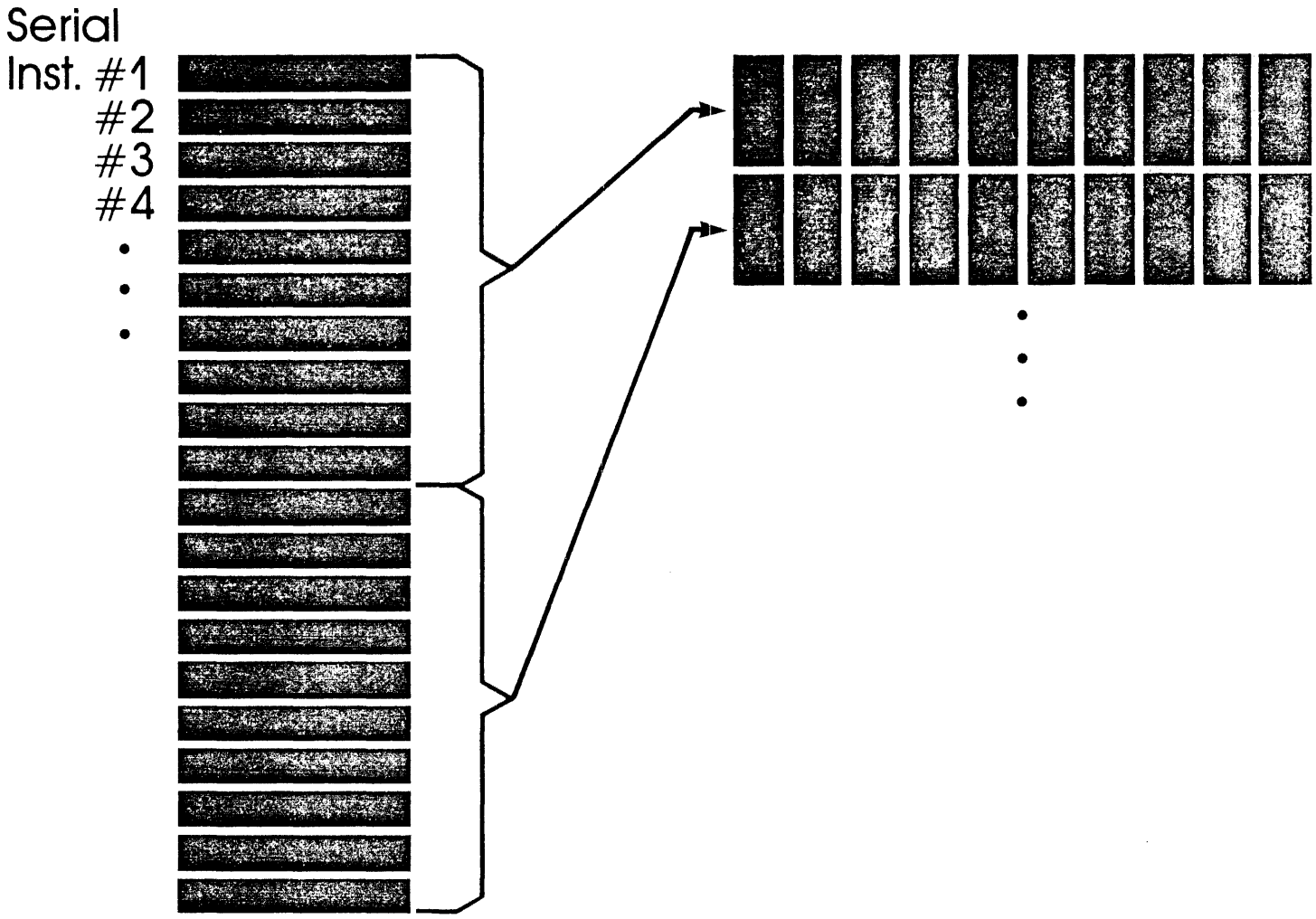
"PIPELINING"


Iteration
1
2
3
N
Fetch Set 3
Process Set 2
Deposit 1 and Loop
Fetch Set 4
Process Set 3
Deposit 2 and Loop
Fetch Set 5
Process Set 4
Deposit 3 and Loop
Fetch Set n+2
Process Set n+1
Deposit n and Exit

THE SETUP



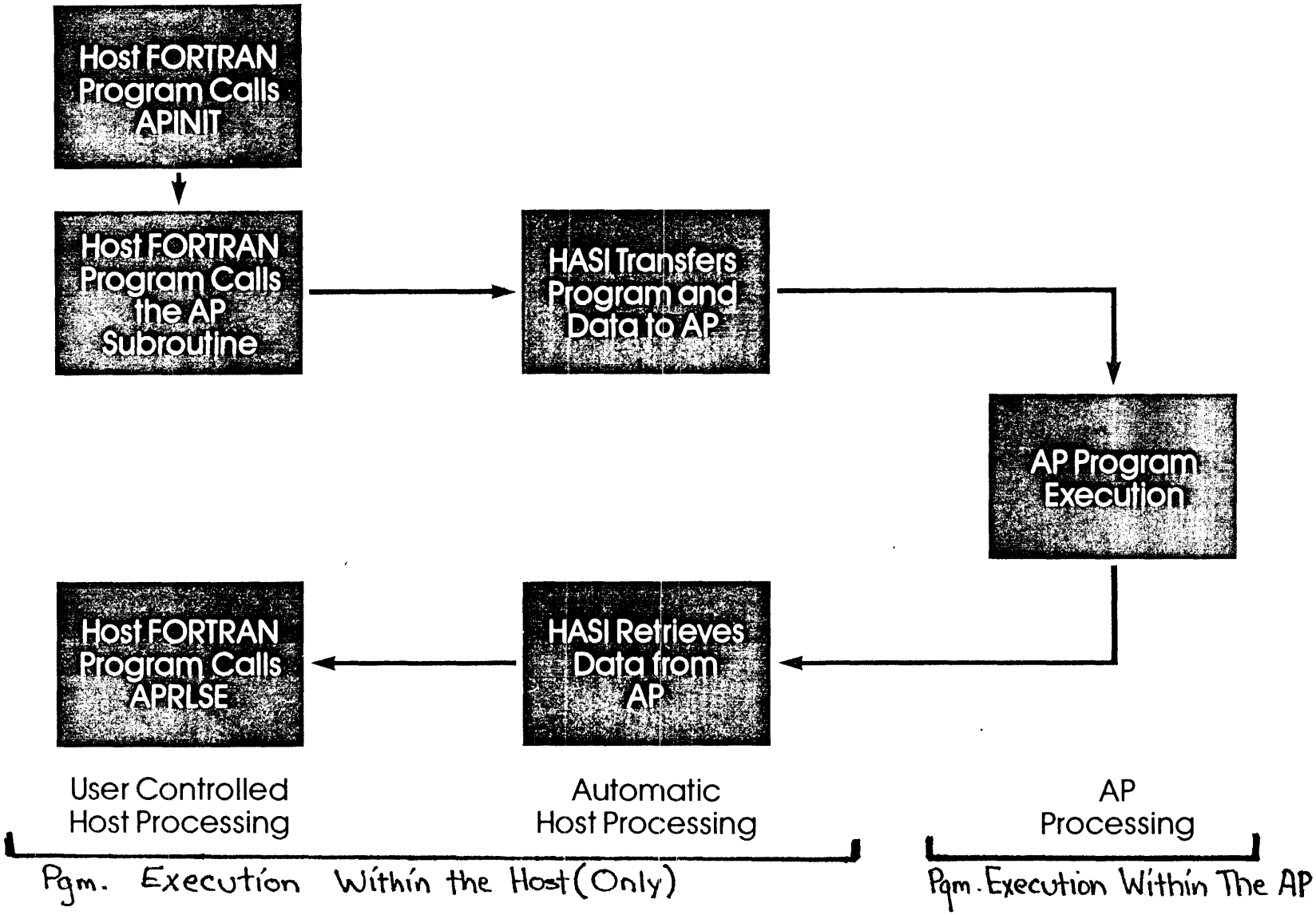
Serial Execution Vs. Parallel Execution



APPLICATIONS DESIGN

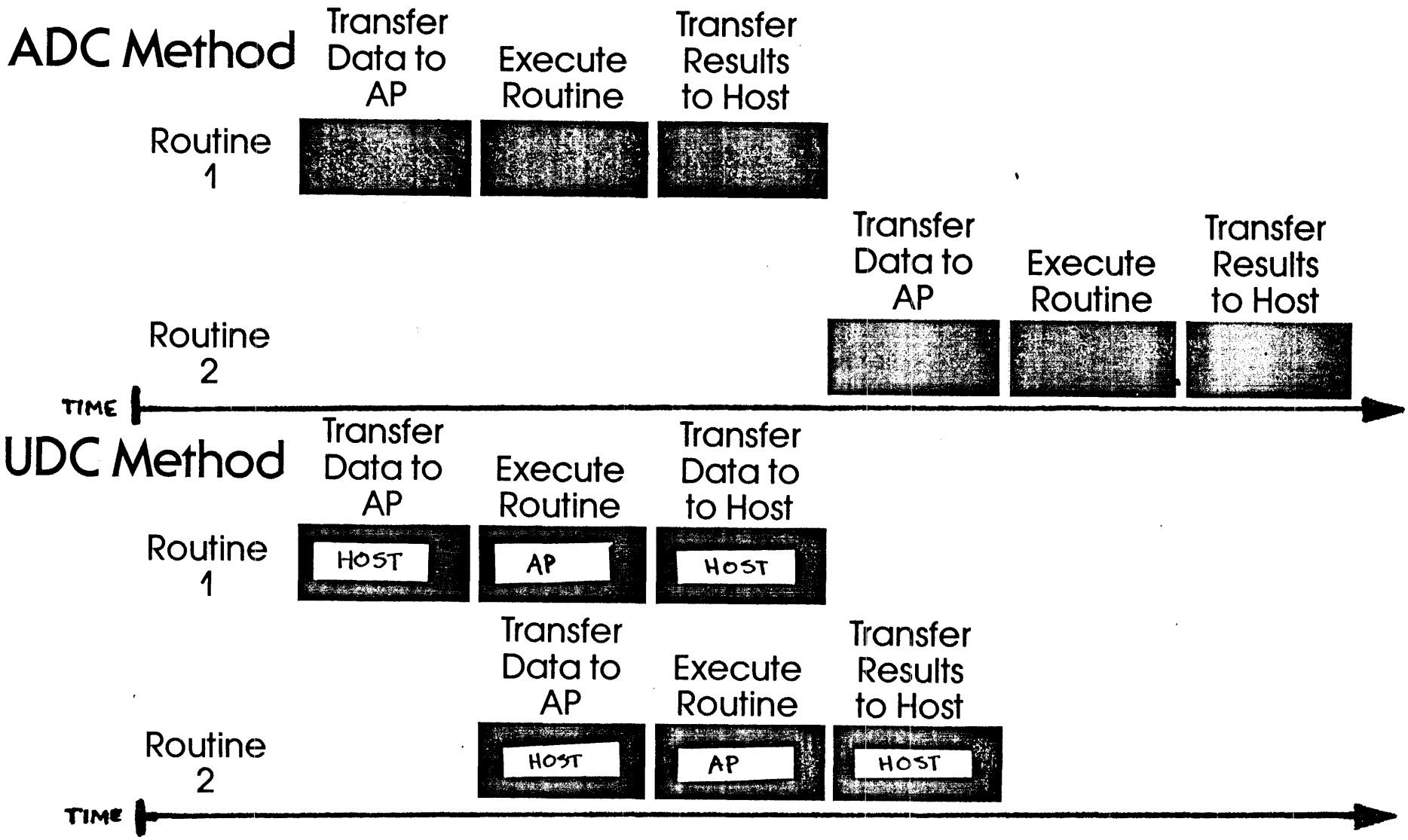
- APPROACHES:
  - ENTIRE PGM. ON 164
  - HOST RESIDENT MAINLINE CALLS 164 SUBROUTINES
- CHOICES:
  - SJE
  - APEX64

ADC MODE OF OPERATION (APEX64)

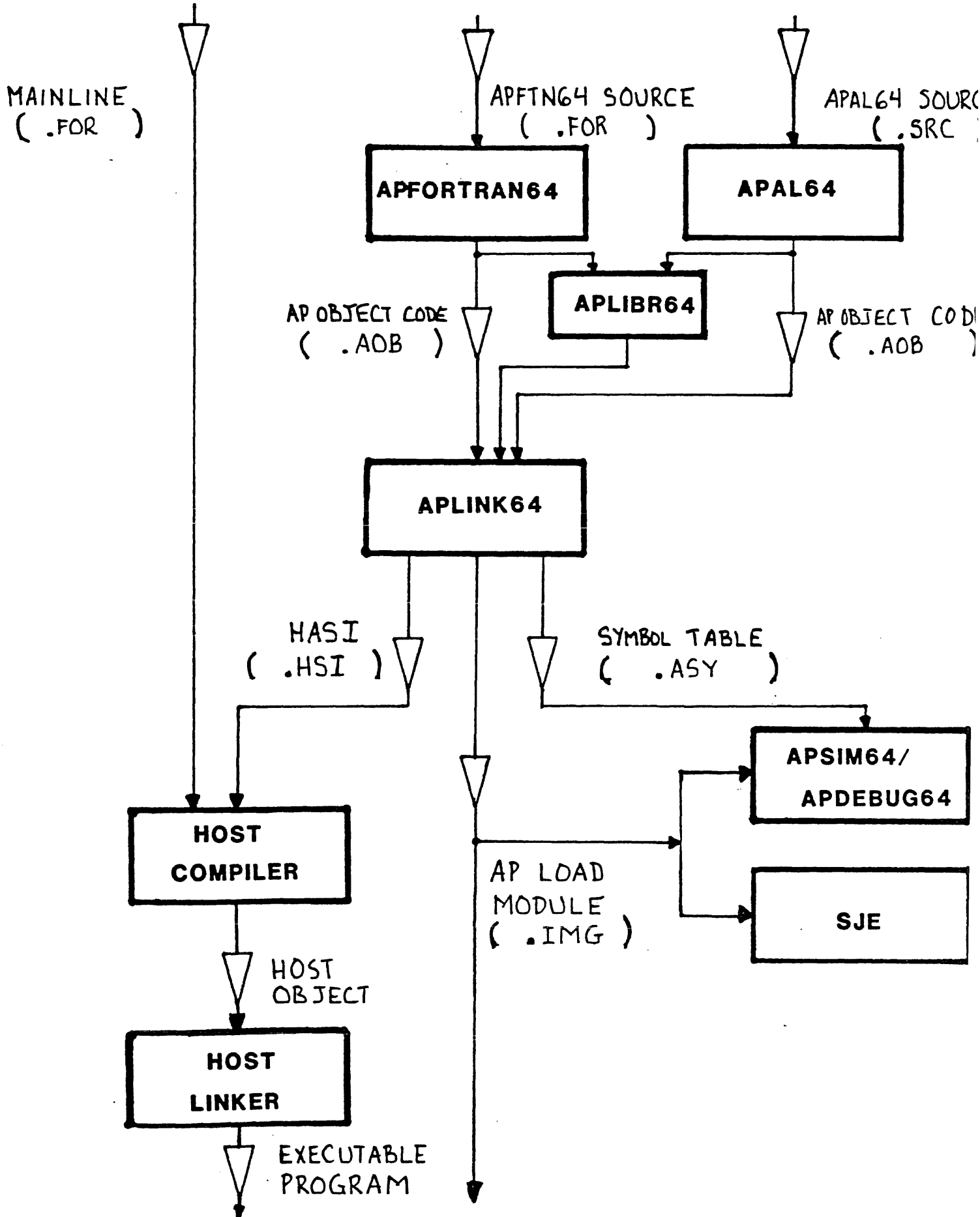


(APEX64 ONLY)

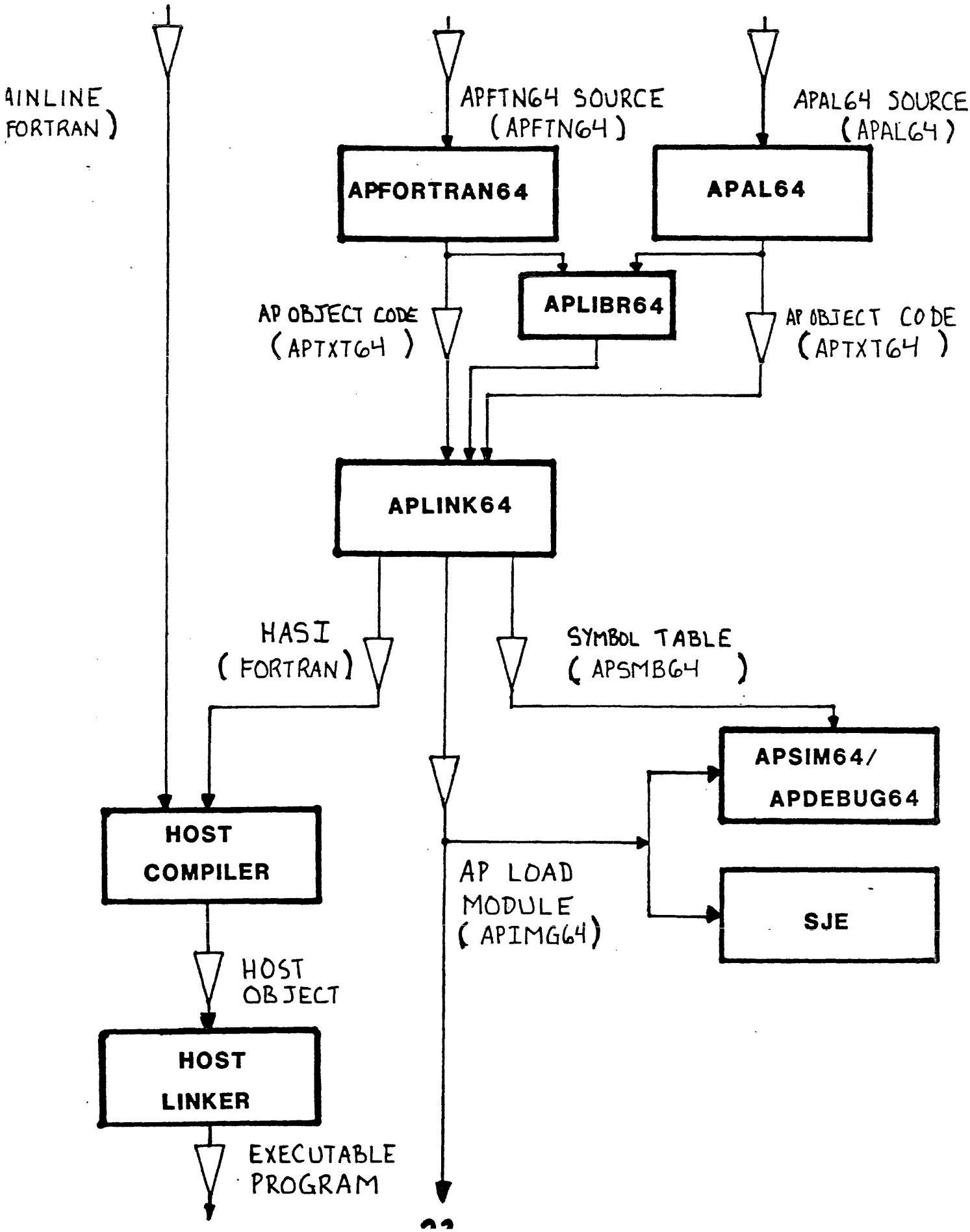
OVERLAPPING DATA TRANSFER AND AP EXECUTION



VAX/VMS PDS FLOWCHART



IBM/CMS PDS FLOWCHART



1. APFTN64

- EXTENSION OF FTN-77
- CROSS-COMPILER
- 2 PGM. DESIGNS:
  - A. AP-RESIDENT SUBROUTINES CONTROLLED BY HOST-RES. MAINLINE [APEX64]
  - B. AP-RESIDENT MAINLINES + SUBROUTINES [SJE]
- CHOICE OF METHOD:
  - SEE p. 3-2; USER'S HNDBK. AND MASTER INDEX (E.REL)

• LANGUAGE REQUIREMENTS

- 1ST STMT. MUST BE:
  PROGRAM, SUBROUTINE, FUNCTION,
  BLOCK DATA, APROUTINE,
  APFUNCTION

- LAST STMT. MUST BE **END**

• RESTRICTIONS

- NO 128-BIT DBL. PRECISION

- RESERVED WORDS IF ASSY. SRC.
  OUTPUT OPTION USED [P. 2-5]

• FEATURES

- SYMBOL NAMES : 31 CHAR. MAX.
  ALPHANUMERIC PLUS $ AND –

- RADIX : DEFAULT IS **DECIMAL**
  HEX : **Z**'ØA2'
  OCTAL : **O**'773'
  BINARY : **B**'11Ø'

- ARGUMENT CONTROL STMTS.
  PURPOSE : REDUCE I/O

APROUTINE
APFUNCTION

APIN [HOST → AP]
APOUT [AP → HOST]
APIO [HOST ↔ AP]

- EXAMPLE:

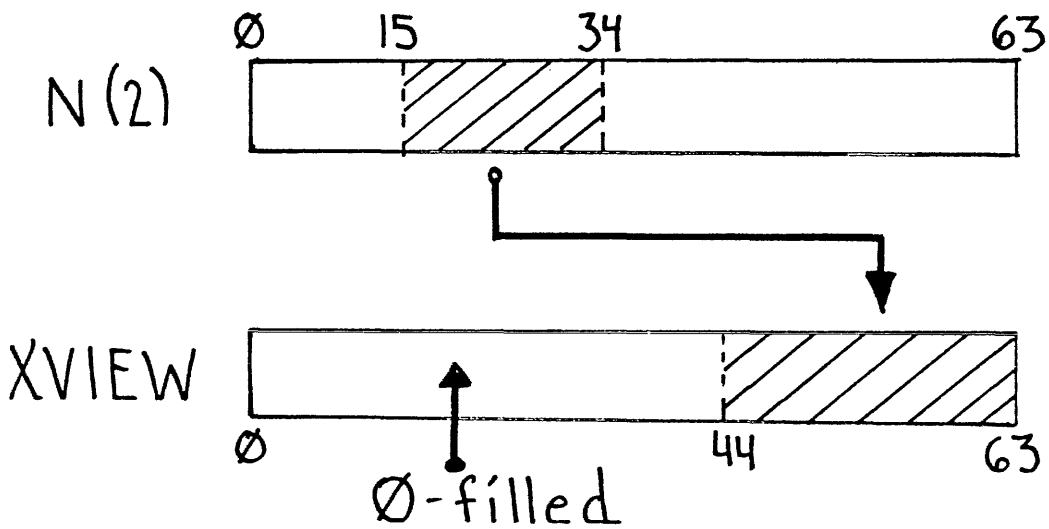
```
APROUTINE GEN (X,Y,Z)
DIMENSION X(10),Y(10),Z(10)
COMMON /IN/ C(10)
COMMON /OUT/ D(10)
APIN /IN/, X(10), Y(10), Z(10)
APOUT /OUT/
      :
```

- EXAMPLE [DYNAMIC DIMENSIONS]:

```
APROUTINE GEN (X,Y,Z,N)
DIMENSION X(N),Y(N),Z(N)
      :
APIN X,Y,N
APOUT Z
      :
```

- IF STARTS WITH SUBROUTINE OR FUNCTION:
  - ALL ARGS. + COMMON ARE TRANSFERRED BOTH WAYS
- IF STARTS WITH APROUTINE:
  - IF IT ISN'T DECLARED WITH APIN, APOUT, OR APIO, NO TRANSFER
- IF STARTS WITH APFUNCTION:
  - UNDECLARED COMMON NOT TRANSFERED
  - UNDECLARED ARGS. HOST → AP
  - FCN. RESULT XFERED BACK TO THE HOST

- ADDITIONAL INTRINSIC FCNS.
  - PROVIDE STRING MANIPULATION + LOGICAL OPERATIONS
  - WORK ON 64-BIT DATA
  - LOGICAL: AND, OR, SHIFT, EQV, NEQV, COMPL
  - STRING: LOC, EXTRACT, INSERT
- EXAMPLE:
  - XVIEW = EXTRACT(N(2), 15, 20)



- Additional Compiler Directives

- Control aspects of compilation
- Placed in col. 1 of APFTN64 src.:

```
$INSERT 'filename'
$LIST
$NOLIST
$FOOTER
$EJECT
```

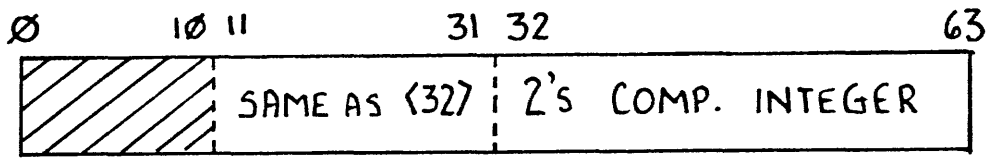
- Compilation options take precedence

• DATA TYPES:

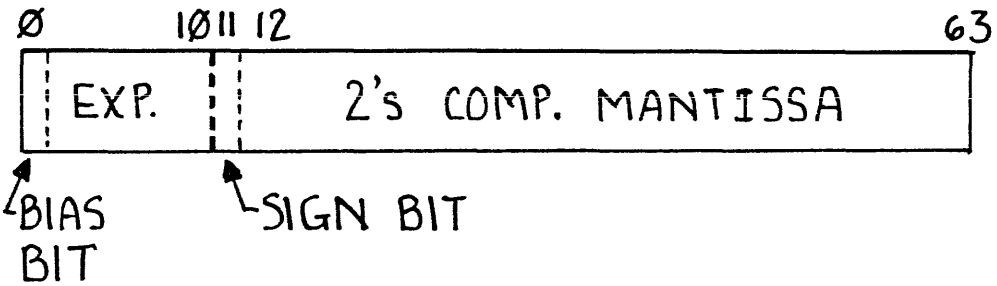
- 8 TYPES:

INTEGER	REAL
COMPLEX	DOUBLE PRECISION
LOGICAL	HOLLERITH
WORD	CHARACTER

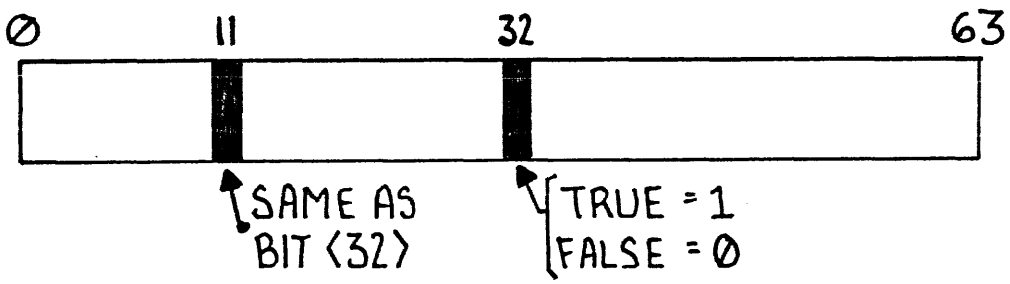
- INTEGER [*2, *4, *8]



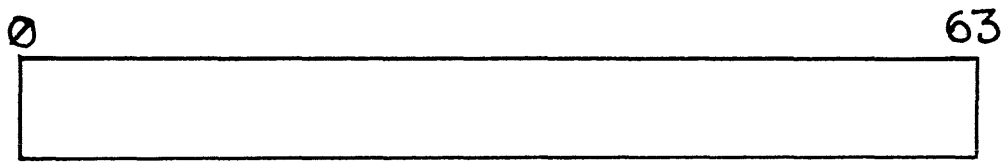
- REAL [*4]



- DOUBLE PRECISION [REAL*8 OR*16]
  - SAME FORMAT AS REAL
- COMPLEX [*8, *16]
  - 2 CONSECUTIVE REAL NUMBERS.
  - 1ST WORD = REAL PART
  - 2ND WORD = IMAGINARY PART
- LOGICAL [*1, *2, *4, *8]

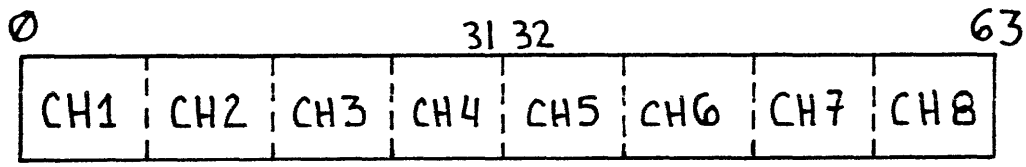


- WORD



  - RIGHT JUSTIFED
  - 0 FILLED

- HOLLERITH +CHARACTER



- ASCII FORMAT
- CHARACTER IS Ø FILLED
- HOLLERITH IS ASCII SPACE (O'40') FILLED
- 2 COMPILER OPTIONS:

  H4 : 4 CHAR./WORD

  H8 : 8 CHAR./WORD [DEFAULT]

• COMPILER DIAGS.

- PROVIDE SRC. LINE NUMBER, SEVERITY LEVEL, MESSAGE, AND OPTIONAL DETAILS

- 4 ERROR SEVERITY LEVELS:

  WARNING (W)
  ERROR (E)
  SEVERE (S)
  TERMINAL (T) 

• OPTIMIZATION

- AP INDEPENDENT OPTS:

  1. COMMON SUBEXPRESSION ELIM.
  2. CONSTANT FOLDING
  3. DEAD CODE ELIMINATION
  4. INVARIANT CODE MOTION
  5. INDUCTION VARIABLE ELIM.

- AP DEPENDENT OPTS:

  a. FUNCTIONAL UNIT SCHEDULING
  b. STRENGTH REDUCTION
  c. REGISTER ALLOCATION
  d. SOFTWARE PIPELINING

- OPTIMIZATION LEVELS

Ø LOOKS AT EACH SRC. LINE, 1 AT A TIME. FASTEST. [1, 2, SIMPLE B]

1 BASIC BLOCK ≡ CODE W/SINGLE ENTRY + EXIT. LEVEL Ø ON BASIC BLOCK + [a]

2 LEVEL 1 ON ENTIRE PGM. ALSO [c, 3, + SOME CODE rearrangement (5)]

3 "PIPELINER". PERFORMS [d] ON "WELL BEHAVED" LOOPS

4 LEVEL 3 PLUS [4]

•COMPILER OPTIONS

- DEFAULTS ARE: [VAX/VMS SYNTAX]

  /OPT=1

  /OBJECT

  /DIAGS=WARNING

  /FAILURE=ERROR

  /NOONETRIP

  /XOFF=ALL

- USEFUL OPTIONS

  /LIST

  /CODE

  /APAL

  /MAP

  /DEBUG

• COMPILER OPTIONS

- DEFAULTS ARE: [IBM/CMS SYNTAX]

  OPT(1)
  OBJECT
  DIAG(WARNING)
  FAILURE(ERROR)
  H8

- USEFUL OPTIONS:

  LIST
  CODE
  APAL
  MAP (LEVEL)
  DEBUG

MATH LIBRARY (APMATH64)

ALSO 3 BOEING LIBRARY & Fast Matrix Solution LIBRARY

- PURPOSE:

  COLLECTION OF ≈500 OPTIMIZED ASSY. LANGUAGE ROUTINES

- USE:

  CALLABLE FROM APFTN64 OR APAL64

- INDEX:
  - CH. 8; APFTN64 USER'S GUIDE
  - CH. 2; APMATH64 PART 1
  - APP. F; APMATH64 PART 3

MATH LIBRARY (APMATH64)

- TYPES OF ROUTINES:
  - SCALAR
  - VECTOR
    - BASIC MATH
    - VECTOR → SCALAR
    - VECTOR COMPARISONS
    - COMPLEX ARITHMETIC
    - MATRIX OPERATIONS
    - FFT'S
    - SIGNAL PROCESSING
    - SPARSE MATRIX OPERATION
    - ADVANCED MATH FUNCTION

EFFICIENT APFTN64 PROGRAMS

Avoid using loops with small iteration counts when compiling at OPT=3 or OPT=4.

Use DO-loops, instead of IF and GOTO statements, in loops that can be pipelined.

Use ONETRIP compiler option.

Use APMATH64 routines if

- One routine can replace an entire loop.
- The iteration count of the loop is 60 or more.
- One math library intrinsic function can be used.

2. APLINK64 - INTRO.

- **PURPOSE**

  LINK AND RELOCATE AP OBJECT CODE.

- **OUTPUT**

  HASI [HOST-TO-AP S/W INTERFACE]
  LOAD MODULE

- **INVOKATION**

  [VAX/VMS] $APLINK64 AP OBJ. FILE-SPEC

  [IBM/CMS] APLINK64 AP OBJ. NAME

Exercise 1

This exercise is designed to give you a chance to use the APFTN64 language and compiler, to see differences and features of this product.

1. Write an APFTN64 program called "WXYZ" which will solve the folowing equation:

$$Zi = \frac{((Xi * Yi) + Wi)^2}{pi * sqrt(2) * i}$$

   - W, X, Y, and Z each have 20 elements, and i varies from 1 to 20
   - This program should create data for W, X, and Y, using the following values:

     W: 1 to 20, by 1
     X: 2 to 40, by 2
     Y: 6 to 25, by 1

   - Have your program display its results

2. Compile the program 3 seperate times to get a chance to see what the various APFTN64 options do and become familiar with the syntax. Some possibilities are:

     Pass 1: LIST
     Pass 2: LIST, CODE, and opt. level 3
     Pass 3: DEBUG, APAL, LIST

3. Use APLINK64 to create the load module.

4. Use the following commands to execute your program. Substitute the name of the load module for XXXX in the command list below:

```
SJE
ATTACH/WAIT
COPYIN/B     'XXXX',PROG
PROG
DETACH
QUIT
```

Floating Point Systems, Inc.,
Corporate Training Department Materials

System Job Executive (SJE)

FPS-164 Software Programming Class

SJE FEATURES

* Processing of a complete user job on the AP.
* FORTRAN 77 I/O and TERMINAL I/O.
* I/O directly to host disk from an AP program.
* A Job Definition Language supporting user job control.
* File transfer between the host and AP file systems of both text and binary files.
* Permanent disk file system on the D64. The File Management System (FMS) provides:
    * Separate user directories.
    * Access keys.
* Roll-in/Roll-out to provide sharing of the AP by several jobs.
* Accounting to record CPU time, total elapsed time, and number of disk I/O's for each job in the file.

- SUM = 64 KWORDS
- FMS STRUCTURE
  - 2 LEVEL DIRECTORY STRUCTURE, WITH "SYSTEM" DIR. AT TOP, FOLLOWED BY USER DIRECTORIES
  - 3 CLASSES OF FILES: [P. 2-2]
    - UNNAMED TEMPORARY
    - SEMI-PERMANENT
    - PERMANENT
  - FILE + DIRECTORY PROTECTION PROVIDED BY 2 OPTIONAL PASSWORDS CALLED "KEYS" [P. 2-3]
    - READER KEY: Allows read access
    - OWNER KEY: READ, WRITE, DELETE, CREATE, RENAME

- TEMPORARY DIRECTORY CALLED ":SCRATCH:" CREATED WHEN ATTACHED
  - CONTAINS SEMI-PERMANENT FILES
- FILES + DIRECTORIES CAN SPAN ACROSS D64'S + BAD LOCATIONS

• ROLL-IN / ROLL-OUT [P. 3-25; VOL. 1]

- ALLOWS ROUND-ROBIN SHARING OF AP BY SJE + APEX64 JOBS
- REQUIRES D-64
- MAX. OF 31 JOBS IN THE QUEUE. SIZE OF QUEUE + TIME SLICE ARE SITE-CHANGEABLE IN SITE PARAMETER FILE.
- APEX64 + SJE JOBS TREATED THE SAME

•ACCOUNTING [P. 3-25; VOL. 1]

- A D64 FILE CONTAINS INFORMATION ABOUT WHO USED THE AP (IN EITHER SJE OR APEX64 MODE) AND:
  - Attach Time (Wall Clock)
  - Execution Time
  - No. of I/O Operations Performed
- The File, called ":USAGE64", IS STORED AS BINARY DATA, AND A TRANSLATION PGM. MUST BE USED TO READ IT.

AP FILE NAMES

AP file names can be 1 to 128 characters in length (including directory names and keys). AP file names can contain any of the following ASCII characters:

* letters (A...Z)

* digits (Ø...9)

* dollar sign ($)

* period (.)

* underscore (_)

Colons are used to separate directories from file names, and keys are enclosed in parentheses. Lowercase letters are treated the same as uppercase.

- **HOST I/O FILENAMING** [P. 2-5; OP. SYS. VOL. 3]
  - RESERVED NAMES- ":INPUT" AND ":OUTPUT"
  - DEFAULTS: :INPUT = 5 , :OUTPUT = 6
  - PREFIXES: ":HOST:" AND ":HOSTCHAR:"
  - USED TO REFER TO BINARY + CHARACTER HOST RESIDENT FILES

ATTach [/Wait] [/TMram] [n]

* Attempt to become the current AP user.

* /Wait option - waits for an available AP.

* /TMram option - specifies that Table Memory RAM is to be used in this job.

* 'n' option - specifies a particular AP to assign.

EXAMPLES:

ATTACH	Attach to any AP, but don't wait if none are available
ATT/W 3	Attach only to AP number 3, and wait if it is not available.

COPYIn[/Binary] [/DRives=XY,...] <source_file>[,<dest_file>]

* Copy a file from the Host's file system to the SJE file system.

* /Binary option - transfers a binary file. Text file transfers are the default.

* If the <dest_file> is not specified, then the <source_file> is used.

* SJE supports the transfer of sequential access files only.

* /DRives specifies up to 10 drives on the FPS-D64 Disk Subsystem to place the file on. "X", which has a value from 2 to 7, refers to the subsystem desired and "Y" refers to the drive on the desired subsystem with a value from 0 to 3.

EXAMPLES:

COPYIN HOSTTEXT.TXT	Copy the Host text file, HOSTTEXT.TXT, to a new SJE file of the same name.
COPYI/B MYPROG.IMG,GO	Copy the host binary file, MYPROG.IMG to an SJE file named GO.

PROGRAM EXECUTION

<Program> <Parameter List>

- <Program> is the file name of the program to be executed
- The parameter list may be empty.
- Examples:

```
MYPROG A,B          ! Execute the user program in the image file
                    ! "MYPROG" which has been copied into the SJE.
                    ! Pass the Parameter List "A,B" as a string to
                    !MYPROG

COPY FILE1,FILE2    ! Execute a user-written utility
                    ! to copy FILE1 to FILE2 on the AP.
```

COPYOUT [/Binary] <source_filename> [,<dest_filename>]

* Copy a file from the SJE file system to the HOST file system.

* /Binary option - transfers a binary file. Text file transfers are the default.

* If the <dest_filename> is not specified, then the <source_filename> is used.

* SJE supports the transfer of sequential access files only.

EXAMPLES:

COPYOUT LISFILE.LIS	Copy the SJE text file, LISFILE.LIS, to a new HOST file of the same name.
COPYO/B MY,MYDAT.DAT	Copy the SJE binary file MY to a host file, MYDAT.DAT.

DETach

* Release the AP for use by other users.

Any files remaining in the :SCRATCH: directory at the time of a DETACH are lost.

EXAMPLE:

```
DET
```

QUIT

* Quit interacting with SJE and return to the host command level.

* May be used whenever the user is not attached to the AP.

EXAMPLES:

QUIT	Quit interacting with SJE.

JDL EXAMPLE

```
$ SJE

SJE Version 1.0

SJE> ATTACH/WAIT
SJE-I-ATTACH, Assigned processor number:  1.

SJE> COPYIN/BINARY MYPROG.IMG,MYPROG
SJE-I-COPYIN, File copied in.

SJE> COPYIN/BINARY DATAFILE1.DAT,DATAFILE
SJE-I-COPYIN, File copied in.

SJE> MYPROG

[user interacts with the executing program "MYPROG"]

SJE> COPYOUT/BINARY DATAFILE2,DATAFILE2.DAT
SJE-I-COPYOUT, File copied out.

SJE> COPYOUT LISTFILE,LISTFILE.LIS
SJE-I-COPYOUT, File copied out.

SJE> DETACH

SJE> QUIT

$
```

SJE: Lab Exercise 1

Purpose:

This lab familiarizes you with the basic JDL command set in SJE, and in control and execution of a simple program under SJE.

1. Write a small host FORTRAN program which creates 3 FORMATTED data files. These files should each contain 25 real numbers which will be used by the "WXYZ" APFTN64 program written earlier. Suggested names for the files might be WDATA, XDATA, and YDATA, with the file type appropriate for your host (VAX = .DAT; CMS = DATA).

2. Modify the host-resident mainline that you wrote earlier to expect the input data from the 3 files created in step 1, but the file names used in your OPEN statements should be WDATA, XDATA, and YDATA without any file types. Your mainline should also write the results of the program into a new FORMATTED file called ZDATA (no file type).

3. Use APFTN64 and APLINK64 to compile and link the modified WXYZ program.

4. Use SJE to assign an AP, bring your load module and data files into the FMS, execute your program, and bring the results back to your host.

5. Visually verify that the file brought back to your host contains correct data.

User Attention

* User Attention is a host-dependent terminal operation which gains the attention of the SJE system for the purpose of typing a command.

* On the VAX, User Attention is a control-c.
* IBM/CMS: EITHER HIT "ENTER" OR "PA1" TWICE

Only the ABORT, CONTINUE, and DEBUG/NOW commands may be used following the User Attention.

After performing this operation, a 15-second delay can occur before SJE's attention is gained.

ABORT

* Terminate the executing AP program.

* Used following a User Attention.

EXAMPLES:

ABORT	Abort the program.

CONtinue

* Resume execution of an interrupted AP program.

The CONTINUE statement is used following a control-c or a FORTRAN PAUSE.

EXAMPLE:

```
   CON              Resume execution of the AP program.
```

DEBug [/Now]

* Invokes the FPS interactive symbolic debugger, APDEBUG64.

* /NOW option - causes immediate execution of the debugger.

* /Defer option causes the debugger to gain control when the next program is loaded. /Defer is the default.

EXAMPLES:

DEBUG	SJE will run debugger when next program is loaded.

HOST CONVERSION UTILITIES

OVERVIEW

- Perform data conversion between Host and FPS-164 data formats.
- Perform file conversion between Host and SJE file formats.
- Conversion Utility use:

  1) Transfer binary (machine format) data from a file which was written using FORTRAN Unformatted WRITE statements;

  2) Read the file on the destination system using the matching FORTRAN Unformatted READ statements.

- The user can combine program calls to these utilities with Host I/O services to create Host files that can be transferred as binary files.

DATA AND FILE CONVERSION ROUTINES

- Data conversion routines convert to and from the following types of data:
  - Host integers
  - Host real (floating-point) numbers
  - Host double-precision numbers
  - SJE integers
  - SJE real (floating-point) numbers
  - HOST LOGICALS
  - HOST CHARACTERS
- File conversion routines convert FORTRAN Unformatted File records between Host and AP formats.

- CONVERSION ROUTINES IN UTILITY LIBRARY:

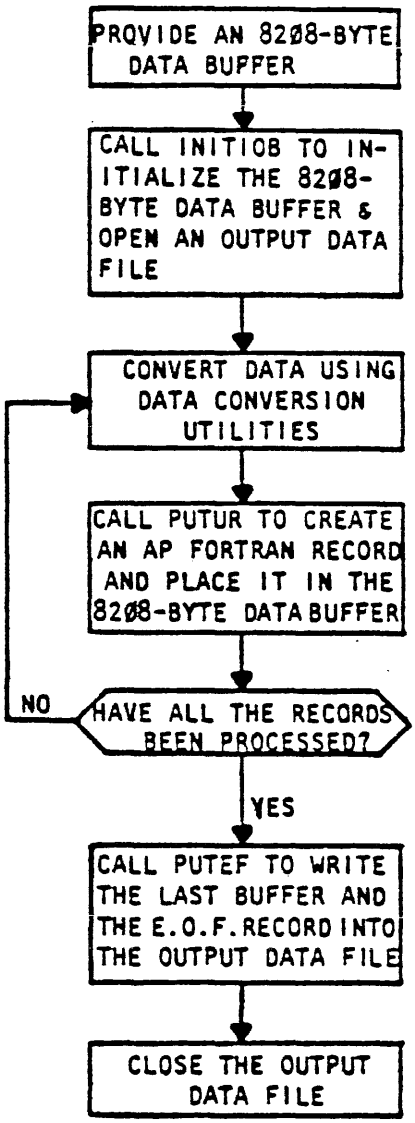
VAX/VMS: [FPS]UTIL64.OLB

IBM/CMS: UTIL64 TXTLIB

IBM/MVS: FPS.UTIL64

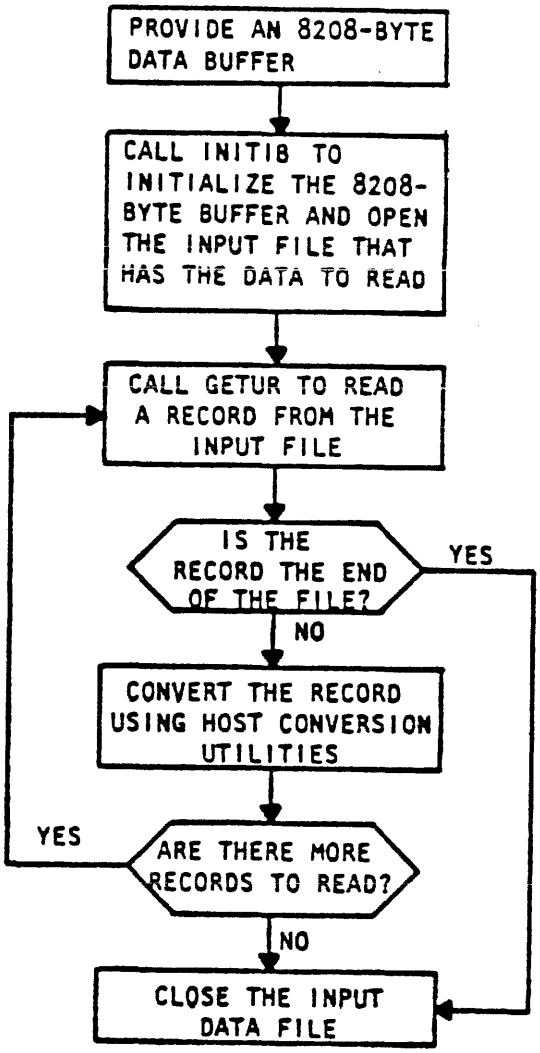
CONVERSION PROCESS

HOST → AP



CONVERSION PROCESS

AP → HOST



Conversion Process

- Control Routines

  INITOB

  INITIB

  PUTUR

  GETUR

  PUTEF

- All Are INTEGER Functions

Conversion Process

- Host ➡ AP Routines

  FPHI2I

  FPHR2R

  FPHD2R

  FPHL2L

  FPHC2C

Conversion Process

- AP ➡ Host Routines

  FPI2HI

  FPR2HR

  FPR2HD

  FPL2HL

  FPC2HC

SJE: Lab Exercise 2

Purpose:

This lab familiarizes you with the host conversion utilities.

1. Write a host program which creates an UNFORMATTED data file containing 10 integers, with values frrom 1 to 10.

2. Write another host program which converts the data in the file created in step 1 to proper AP integer format. You will have to store the converted data in a new file.

3. Write a short APFTN64 program which reads an AP file, squares each value, and write the results as real numbers to a new UNFORMATTED file which is created by this program on your host.

4. Use SJE to copy the converted data and the AP load module to the AP and execute the program.

5. Write another host program to convert the unformatted data file written from the AP. Verify that your programs work correctly.

ACCESS <directory_name>

* Set the current directory allowing access to it and the files in the directory.

* The access given to the directory depends on the password specified within the <directory_name>.

EXAMPLES:

ACCESS :USER(SECRET)	Change the current directory to :USER, specifying the password "SECRET".
AC :SCRATCH:	Change the current directory to the user's personal scratch directory. This directory is assigned to the user when connected to the AP.
ACC (PWORD):	Set the current directory to the system directory. "PWORD" is its password.

CHange [/options] <file_name>

* Change the attributes of a file.

* The attributes that can be changed are:

RECTYP	record type
FILORG	file organization
FMTTYP	format type
EXTSIZE	extent size
MAXSIZE	maximum size
LENGTH	record length

EXAMPLE:

CHANGE/FILORG=SEQUENTIAL A.FTN Change the file organization of file "A.FTN" to sequential.

COpy [/options] <source_name> <dest_name>

* Copy files and/or directories from one place to another in the AP file system.

* The file or files in the directory named by the <source_name> are copied to the destination file or directory named by the <dest_name>.

* No data conversion is performed on the copied file. The destination file is created with the same attributes as the source file except for the keys. These attributes of the destination file are assigned blank owner and read keys unless the /KEYS, /OKEY, or /RKEY options are specified.

* The destination directory must already exist.

The following are the options for the COPY command:

/REplace — Specifies that if a destination file with the same name as a source file exists, the old file is replaced with the new file and its Existing files are not replaced unless the option is used.

/ALL — Specifies that all the files in the source directory are to be copied to the destination directory. Files of similar names are not replaced unless the /RE option is specified.

/KEys — Specifies that the source file's keys are to be used as the keys for the destination file.

/OKey=ownerkey — Specifies that the owner key of the destination file is assigned the simple string in "ownerkey".

/RKey=readkey — Specifies that the read key of the destination file is assigned the simple string in "readkey".

COpy [/options] <source_name> <dest_name>

* Copy files and/or directories from one place to another in the AP file system.

EXAMPLES:

CO AFILE :BACKUP:AFILE Copy the file "AFILE" in the current directory to the file "AFILE" in the directory :BACKUP.

COPY/REP :TIM:ONE :JOE:LEVEL1
Copy the file "ONE" in the directory :TIM to file "LEVEL1" in the directory :JOE, replacing it if it already exists.

COPY/ALL/REPLACE :GEORGE: :SUE:
After the copy statement is complete, the directory :SUE contains the files it already had and all of the files from the directory :GEORGE. If :SUE had any file with the same name as any in :GEORGE, the files from :GEORGE overwrote the files in :SUE.

CReate[/options] <file_name>

* Create files or directories in the AP file system.

* Options include:

  /DIrectory - Creates a directory with the name specified in <file_name>.

  /RKey=readkey - Assigns password readkey to allow read access to the created file or directory.

  /OKey=ownerkey - Assigns password ownerkey to allow owner access (read,write,delete,and change the attributes) of the created file or directory.

  /DRives=(XY,...) - Specifies D64 disk drives to use for file storage. The default is any subsystem and drive. X specifies the subsystem and can be an integer from 2 to 7. Y specifies the drive on the subsystem and can be and integer from 0 to 3.

* Options valid only for files (not directories) include:

  /SIze=size - Specifies the initial size for the file as an integer from 1K to 2(2**21)K. The default size is 18K words.

  /REctyp=type - Type can be Fixed, Variable, or None. The default is None.

  /FIlorg=otype - Otype can be Sequential or Direct. The default is Sequential.

CReate[/options] <file_name>

* Create files or directories in the AP file system.

* Options valid only for files (not directories) include

  /LEngth=reclen - Specifies the size of each record for files using a record structure. The default is is 8192 bytes per record.

  /MAxsize=msize - Specifies the maximum size, in K words of the file. An msize of -1, the default value, indicates the maximum possible (2**21 K words) size.

  /EXtsize=esize - Specifies the file extent size. Esiz specifies the number of K words for each extension to the file. Default esize is one-fourth of the SIze attribute.

EXAMPLES:

CREATE FILE1	Create FILE1 in the currently accessed directory.
CR/DIR (SYSPASSWD):D1	Create a directory called :D1, specifying the System Directory password "SYSPASSWD".
CR/RE=F/FI=DIR & FILE1	Create FILE1 with a fixed record structure, and a direct file organization. "&" continues a line.

DElete [/ALl] [/DIrectory] <file_1>,...<file_n>

* Remove the specified files or directories from the AP file system.

* /ALl option deletes all of the files in the specified directory.

* /DIrectory option indicates that a directory is to be deleted. The directory must be empty to delete.

EXAMPLE:

DE/ALL/DIR :USER:	Delete all the files in the directory :USER and delete the directory itself.
DE DATA, :USER:JUNK	Delete Data in the current directory and delete JUNK in the directory USER.

DIrectory [/options] [<file>,...]

Lists information concerning the specified files. If no file names are specified, all the files in the current directory are listed. Not for host files.

The options for the DIRECTORY command are:

* /ATtributes - List all attributes of the specified files (subject to access rights). By default, only the file name is listed.

* /OUtput=<output_name> Output of the directory listing goes to <output_name>, instead of to the terminal. <output_name> will be over-written, if it exists.

* /ONLY List only the directory and not the files in it.

EXAMPLES:

DI List the current directory.

DI :USER: List the names of all the files in directory :USER.

DI : List all of the files and directories in the System Directory.

DI/OUT=DIR.LIS/ATT & DATA.DAT, CLASS.CL List all of the file attributes for DATA.DAT and CLASS.CL. The listing is written to DIR.LIS "&" Continues the line.

RENname [/RKey=key] <old_name> [<new_name>]

* Change the name and/or read and owner keys of the specified file or directory.

* /RKey - Changes the old read key for <old_name> to key. Owner access to the directory is required to change keys.

* /OKey - Changes the old owner key for <old_name> to key. Owner access to the directory is required to change keys.

* <new_name> - Changes the name of a file. Owner access to the file or directory is required.

EXAMPLE:

REN MINE URS Rename the file MINE, URS.

SEt [/ACCOunt=acctname] [/LIMit=timelimit]

* Set certain characteristics of the user job.

* /Account option - records the simple or quoted string in acctname in the job accounting file.

* /Limit option - sets the value in timelimit to be the amount of AP CPU time, in seconds, allowed each program within a job before it is timed-out and aborted. The default is no time limit.

EXAMPLES:

SE/LIM=2Ø After 2Ø seconds, the job will be aborte

SHow [/option]

* Report information to the user.

* /ACcess option - Show the currently accessed directory.

* /DIsks option - Gives the status of the disks in system

* /LIMit option - Shows the AP CPU time limit.

* /OUTput=<file_name> option - causes the information to be sent to <file_name> instead of the users terminal

EXAMPLE:

SH/AC List the current directory name.

Type <file_name>

* Print a text file to the terminal.

* <file_name> is typed to the terminal.

EXAMPLE:

TYPE :USER:MYFILE	List the MYFILE in directory :USER: to the terminal.

SJE: Lab Exercise 3

Purpose:

a. Use the conversion utilities
b. Access files from various directories within FMS from SJE
c. Experiment with more advanced JDL commands

1. Supplies:
   a. In your directory, a file called BDATA, which is UNFORMATTED and contains the values 5 to 50, by 5.
   b. On the AP, in the directory :JRAB:, a FORMATTED file called ADATA which contains 10 reals (F5.2) ranging from 1 to 10, by 1.

2. Write an APFTN64 program called VECTADD which will add the 2 files mentioned in step 1 together and create a third file called CDATA:

   a. ADATA resided in the AP directory :JRAB:
   b. BDATA will be converted into AP format and brought over to the AP with the name NEWDATA and placed in your scratch directory
   c. VECTADD creates an UNFORMATTED file called CDATA containing the sums of each data pair in ADATA and NEWDATA

3. Write a host program which converts the 10 reals in BDATA to proper AP format in a file called NEWDATA.

4. Start up SJE and perform the following:

   a. Determine your current directory
   b. Create a new directory named the same as your login name
   c. Change your default to the new directory, and verify
   d. Copy NEWDATA from the host to your scratch directory without changing your default directory
   e. Reset to your scratch directory
   f. Determine the file attributes of NEWDATA
   g. Without bringing in your VECTADD load module, start it executing (think about the file naming syntax)
   h. Bring CDATA out from the AP back to the host
   i. Delete the directory you created and any files that it might contain
   j. Detach and quit

5. Write a host program to convert the 10 reals in CDATA to host format. Display the data to verify proper operation of all steps.

PREserve [/parameters] [<input_specifier>]

* Save one or more AP files in one host file on the host

* These files can later be restored to the AP file system using the RESTORE statement.

* <input_specifier> lists one or more names of existing FMS files or directories to be preserved.

* If a directory name is specified, all files and directories in that directory are preserved.

* A :SCRATCH: directory cannot be preserved.

PREserve [/parameters] [<input_specifier>]

The options for the PRESERVE command are:

* /FILe=:HOST:<host_file> - File will be preserved on host Disk, under the name <host_file>.

* /ID=<name> - <name> is the name stored with the preserve file for identification purposes.

* /TREE - Save the currently accessed directory, including sub-directories.

* /LIst[=list_file] - Write the name and length for each file saved to the file <list_file>. If <list_file> is omitted, the information is written to the user's terminal.

PREserve [/parameters] [<input_specifier>]

* The following are closely related options:

* /TAPe=:HOST:<host_device> - File will be preserved on host tape drive, under the name <host_device>.

* /DENsity=dnum - Use in conjunction with the /TAPe option, where dnum is an integer that specifies the density of information in bits per inch of tape. The default is 1600 bits per inch.

* /SEGnum=snum - Use in conjunction with the /TAPe option, where snum is an integer that identifies the file's relative position on the tape. The preserve file is written as the first file on the tape, by default. More than one preserve file can be written on a single tape by specifying a different snum for each file. A single preserve file can span more than one tape.

PREserve [/parameters] [<input_specifier>]

EXAMPLE:

The following example illustrates the use of ACCESS, PRESERVE and RESTORE. In this example, several FMS files are saved to a host disk file.

```
PRESERVE/FILE=:HOST:DRAØ:[DIRECT1]PRESERVE1.DAT &
/ID=MYFILES/LIST=:HOSTCHAR:PRESERVE1.LIS &
FILES,FILEB,FILEC,FILED

RESTORE/FILE=:HOST:DRAØ:[DIRECT1]PRESERVE1.DAT &
/ID=MYFILES/REPLACE FILED
```

DRAØ:[DIRECT1]PRESERVE1.DAT is a preserve file having MYFILES for an ID and containing the four FMS files FILEA, FILEB, FILEC, and FILED. The output from PRESERVE is written to the file PRESERVE1.LIS in the user's default directory on the host.

This RESTORE replaces the FMS file FILED with the FILED found in DRAØ:[DIRECT1]PRESERVE1.DAT.

REStore [/parameters] [<output_specifier>]

* Restore FMS files and directories from a preserve file on the host created by the PRESERVE command.

* /FILe=:HOST:<host_file> - <host_file> is the name of the disk file in which the files or directories specified by <output_specifier> were preserved. The /FILE parameter must be used when files are to be restored from host disk.

* /ID=<name> - <name> is the name stored with the preserve file for identification purposes. The /ID parameter is required, and must be the same name as was used when preserved.

* /REPlace - Replace existing FMS files by versions found in the preserve file.

* /LIst[=list_file] - Write the name and length for each file restored to the file <list_file>. If <list_file> is omitted, the information is written to the user's terminal.

REStore [/parameters] [<output_specifier>]

* The following are closely related options:

* /TAPe=:HOST:<host_device> - File will be preserved on host tape drive, under the name <host_device>.

* /DENsity=dnum - Use in conjunction with the /TAPe option, where dnum is an integer that specifies the density of information in bits per inch of tape. The default is 1600 bits per inch.

* /SEGnum=snum - Use in conjunction with the /TAPe option, where snum is an integer that identifies the file's relative position on the tape. The preserve file is written as the first file on the tape, by default. More than one preserve file can be written on a single tape by specifying a different snum for each file. A single preserve file can span more than one tape.

REStore [/parameters] [<output_specifier>]

EXAMPLE:

The following example illustrates the use of ACCESS, PRESERVE and RESTORE. In this example, the entire FMS is saved onto magnetic tape using PRESERVE and then restored using RESTORE. ACCESS establishes a right to the system directory.

```
ACCESS (system_password):
PRESERVE /TAPE=:HOST:MTA0/ID=JAN.01.1983/TREE

ACCESS (system_password):
RESTORE /TAPE=:HOST:MTA0/ID=JAN.01.1983
```

TREE specifies the currently accessed directory to be saved, including all sub-directories. PRESERVE dynamically ALLOCATEs tape drive MTA0 and MOUNTs a tape on it, if necessary. PRESERVE aborts if MTA0 cannot be allocated. The user's console and the operators console both receive MOUNT requests at the time the tape should be physically placed on MTA0.

REStore [/parameters] [<output_specifier>]

EXAMPLE:

The following example illustrates the use of ACCESS, PRESERVE and RESTORE. In this example, the entire FMS is saved onto magnetic tape using PRESERVE and then restored using RESTORE. ACCESS establishes a right to the system directory.

```
ACCESS (system_password):
PRESERVE /TAPE=:HOST:MTA0/ID=JAN.01.1983/TREE

ACCESS (system_password):
RESTORE /TAPE=:HOST:MTA0/ID=JAN.01.1983
```

PRESERVE defaults the tape density to 1600 bpi and writes the preserve file as the first and only file on the tape. The ID, JAN.01.1983, is kept in the preserve file header for identification in a RESTORE operation.

RESTore [/parameters] [<output_specifier>]

EXAMPLE:

The following example illustrates the use of ACCESS, PRESERVE and RESTORE. In this example, the entire FMS is saved onto magnetic tape using PRESERVE and then restored using RESTORE. ACCESS establishes a right to the system directory.

```
ACCESS (system_password):
PRESERVE /TAPE=:HOST:MTA0/ID=JAN.01.1983/TREE

ACCESS (system_password):
RESTORE /TAPE=:HOST:MTA0/ID=JAN.01.1983
```

RESTORE interfaces with tape drive MTA0. The first file on the tape must have an ID of "JAN.01.1983" or RESTORE aborts. All FMS files and directories contained in the preserve file will be created and restored to FMS if they no longer exist. The REPLACE parameter can be used to replace existing FMS files with those found in the preserve file.

PROGRAM DEBUGGING

APDEBUG64

- INTERACTIVE SYMBOLIC DEBUGGER
- ACCESS TO ALL USER-VISIBLE REGISTERS AND MEMORIES
- SET, CLEAR, AND EXAMINE BREAKPOINTS
- EXECUTE PROGRAMS IN SINGLE-STEP OR FREE-RUNNING MODE
- DISPLAY OF ELAPSED EXECUTION TIME
- ACCESS TO GLOBAL AND LOCAL SYMBOLS
- STAND-ALONE OR FORTRAN-CALLABLE

INVOKING APDEBUG64

- THE AP COMPILATION SHOULD BE PERFORMED WITH THE DEBUG OPTION
- THE APLINK64 SHOULD BE PERFORMED WITH THE SYM OPTION

APSIM64

SIMULATION LIBRARY

ALLOWS AP SOFTWARE TO EXECUTE ON HOST

FUNCTIONAL SIMULATION OF THE APEX INTERFACE

BITWISE SIMULATION OF THE AP

CAN BE LINKED WITH APDEBUG64 TO PROVIDE AN INTERACTIVE SIMULATOR WITHOUT USING THE AP

INVOKING APSIM64

- THE SIMULATOR IS SLOW SO IT IS INADVISABLE TO EXECUTE A LARGE AMOUNT OF FORTRAN CODE ON THE SIMULATOR

- THE ONLY CHANGE IN THE PROCESS IS TO DEFINE THE APSIM64 LIBRARY IN PLACE OF THE APEX64 LIBRARY

VAX environment

```
$ FORTRAN/DEBUG/NOOPT VMUL.HSI
$ FORTRAN/DEBUG/NOOPT/LIST MAIN
$ LINK/DEBUG MAIN,VMUL,-
FPS:APDEBUG64.OLB/LIB,-
FPS:APSIM64.OLB/LIB,-
FPS:UTIL64.OLB/LIB
$ EXIT
```

IBM (MVS/TSO) environment

```
LINK MAIN.OBJ,SUB.OBJ LOAD(MAIN.LOAD(MAIN)) FORTLIB +
LIB('X64.S.IAPEX.LIB','X64.S.APDBUG.LIB',-
'X64.S.APSIM.LIB','X64.S.UTIL.LIB')
```

APSIM64/APDEBUG64

OVERVIEW FOR APFTN64 USERS

- **PURPOSE**

  PROVIDE INTERACTIVE DEBUGGING + SOFTWARE SIMULATION ENVIRONMENT

- **APDEBUG64**

  1. USES THE HARDWARE
  2. ACCESSED IN 3 WAYS:
     - STAND ALONE PGM (INDIVIDUAL RTNS)
     - IN APEX64 MODE, WHEN AP SUBROUTINE IS CALLED
     - FROM SJE

- **APSIM64**

  1. HOST RESIDENT BIT-WISE SIMULATION
  2. DIFFERENCES
     - NO I/O SUPPORT
     - UNAVAILABLE FROM SJE
     - ONLY 64K WORDS main memory
     - Not good for UDC mode

• ACCESS : APEX64 Mode

1. AUTOMATICALLY STARTED WHEN AP SUBROUTINE CALLED. SUBROUTINE IS NOT EXECUTED. DATA, PGM. → AP
2. UPON EXIT, MAINLINE CONTINUES
3. PROGRAM DEVELOPMENT STEPS:
   - USE "DEBUG" OPTION WITH APFTN64
   - USE "SYM" OPTION WITH APLINK64
   - HOST LINK/LOAD WITH APDEBUG64 AND/OR APSIM64 LIBRARIES

• ACCESS: STANDALONE

1. ONLY NEED LOAD MODULE CREATED BY APLINK64 + SYMBOL TABLE
2. YOU WILL NEED TO SET UP ALL AP REGISTERS, MEMORIES, ETC., MANUALLY

• ACCESS: SJE

1. SAME AS WITH APEX64, EXCEPT NO HOST LINKAGE IS PERFORMED
2. CANNOT USE APSIM64

COMMANDS

ACTIVATE Symbols: SYmbol

1. FOR SYMBOLIC DEBUGGING, MUST ACTIVATE SYMBOL TABLE EACH TIME APSIM64/APDEBUG64 IS ENTERED.
2. TO ACTIVATE SYMBOL TABLE: SYM 'filename'
3. ACTIVATE LOCAL SYMBOLS: SYM entrypoint
4. DISPLAY SYMBOL INFO: SYM

EXAMINE A VARIABLE: Examine

1. USED TO LOOK AT A VARIABLE, REGISTER, OR MEMORY
2. CAN LOOK AT:
   - VARIABLES (INDEX, ADATA(2), ···)
   - STMT. LABELS (.12Ø)
   - SRC. LINE # ($2Ø)
   - ENTRYPOINT
   - REGISTERS + MEMORIES
3. SYNTAX:
   - E INDEX
   - E ADATA(1):ADATA(1Ø)
   - E T(1):T(1ØØ):1Ø

• CHANGE A VARIABLE: Deposit

1. USED TO CHANGE THE VALUE OF A VARIABLE, REGISTER, OR MEMORY
2. SYNTAX:
```
D INDEX = 22
D ADATA(1):ADATA(5) = 2.5
```

• EXECUTE A PGM: Run

1. STARTS PROGRAM EXECUTION
2. SYNTAX: R [ENTRYPOINT]
3. WHEN PGM. COMPLETES OR STOPS, ELAPSED TIME IS SHOWN, BASED ON 167 nsec. CLOCK

• EXIT DEBUGGER: Quit

1. IN APEX64 MODE, WILL RETURN TO MAINLINE PGM. AND CONTINUE
2. IN SJE, WILL RETURN TO JDL COMMAND LEVEL

• SET BREAKPOINTS: Break

1. USED TO STOP PGM. EXECUTION AT A PARTICULAR POINT
2. MAX. OF 2 BREAKPOINTS AT A TIME
3. SYNTAX:

B $71	(SRC. LINE 71)
B .20	(STMT. LBL. 20)
B	(DISPLAY BREAKPOINTS)

4. OPTIONAL QUALIFIERS:

/EVERY:n	every n^{th} time
/AFTER:n	after n times
/UNTIL:n	clear after n breaks
/IF(RELATIONAL EXPRESSION)	break if TRUE

5. EXAMPLES:

B/EVERY:4 $13

B/IF (YMIN .EQ. 0.0) .39

• SET WATCHPOINTS: Watch

1. USED TO STOP PGM. EXECUTION WHEN A VARIABLE IS ACCESSED
2. 1 WATCHPOINT ALLOWED AT A TIME
3. SYNTAX:

   W IDATE

4. QUALIFIERS:

   /RD — WHENEVER VARIABLE READ

   /WR — WHENEVER VAR. IS WRITTEN

   /RW — WHENEVER VAR. IS WRITTEN OR READ

   DEFAULT = /WR

   ... AND ALL OPTIONS WITH **BREAK**

5. EXAMPLES:

   W/IF (APTR(1) .LE. AMAX)/RD Z(28)

   W/RW OHOH(7)

- **CLEAR WATCHPOINTS AND BREAKPOINTS : Clear**

1. USED TO CLEAR WATCHPOINTS AND BREAKPOINTS

2. SYNTAX :

   C IDATE

3. OPTIONS :

   $ALL - all watchpoints and breakpoints

LAB: USING APDEBUG64

1. USE "WXYZ" PROGRAM WHICH CREATES DATA INTERNALLY.
2. RECOMPILE WITH "DEBUG" OPTION + BUILD SYMBOL TABLE.
3. RUN UNDER SJE AND:
   a. OPEN YOUR SYMBOL TABLE
   b. LIST ALL 20 "W" VALUES W/1 CMD.
   c. CHANGE Y(1) TO 7.5
   D. CAUSE PGM. HALT WHEN Y(1) IS READ
   E. EXECUTE THE PROGRAM
   F. LIST ALL Z'S WITH 1 COMMAND
   G. EXIT APDEBUG64 + SJE

Floating Point Systems, Inc.,
Corporate Training Department Materials

Program Conversion To Use The FPS-164

FPS-164 Software Programming Class

ELEMENTARY CONVERSION PROCESS

- DETERMINE WHERE THE PROGRAM IS TO BE DIVIDED, AND WHICH SUBROUTINE(S) ARE TO RUN ON THE FPS-164
- USE APFTN64 TO COMPILE THE ROUTINES FOR THE FPS-164
- USE APLINK64 TO PROVIDE LINKAGE AND DECLARE THE SUBROUTINE ENTRY POINTS FROM THE HOST PROGRAM
- INSERT TWO APEX SUBROUTINE CALLS IN THE HOST PROGRAM

  CALL APINIT (INUM, 0,0,0, IASG, ISTAT) TO INITIALIZE FPS-164

  AND

  CALL APRLSE TO RELEASE FPS-164 AFTER USE

- COMPILE WITH THE HOST FORTRAN COMPILER THE HOST PORTION OF THE PROGRAM AND THE HASI OUTPUT FROM APLINK64
- PERFORM HOST LINK OF ALL ROUTINES
- RUN THE PROGRAM USING THE FPS-164 AND VERIFY THE RESULT

3. APEX64 - INTRO.

- **PURPOSE**

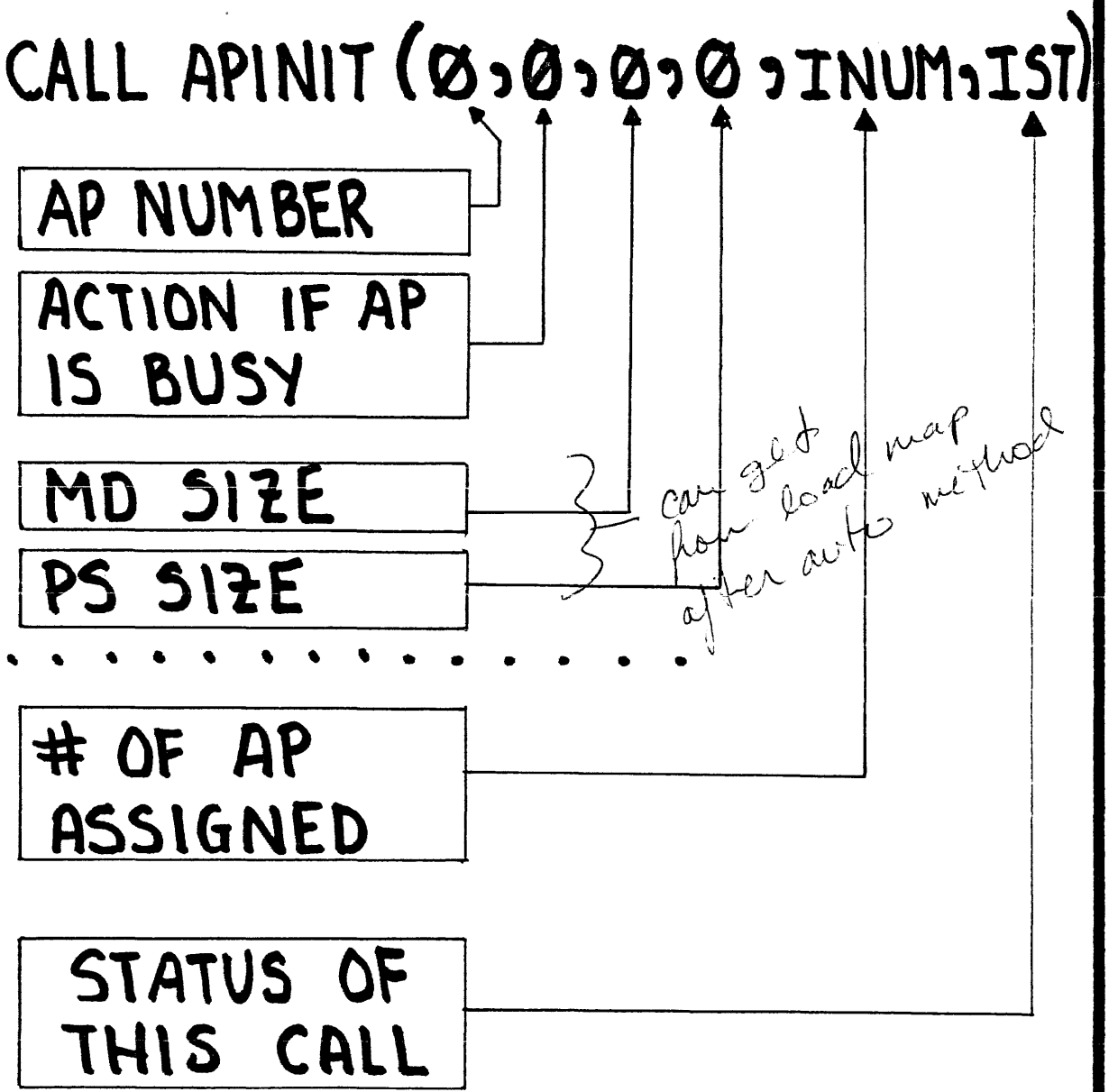
  ACCESS TO AND CONTROL OF THE AP

- **SIMPLIST ACCESS METHOD**

```
      ⋮
CALL APINIT(Ø,Ø,Ø,Ø,IS,IN)
      ⋮
CALL APPGM(...)
CALL APRLSE
      ⋮
```

"MAINLINE" PGM.

BASIC APEX64 CALLS



SAMPLE PROGRAM PRIOR TO MODIFICATION

- CONTAINS TWO SUBROUTINE CALLS
- I/O STATEMENTS
- COMMON BLOCK
- PARAMETERS

```
      PROGRAM MAIN
      COMMON /ARRY/ A(10),B(10),C(10),D(10)
      DO 100 I=1,10
         A(I) = FLOAT (I)
         B(I) = FLOAT (I)
         C(I) = FLOAT (I)
100   CONTINUE
      WRITE (6,1000) I
      N = I-1
      CALL VMUL (N,APCPU1)
      WRITE (6,1010) D(N)
      CALL VSQ (N,APCPU2)
      APCPU = APCPU1 + APCPU2
      WRITE (6,1020) APCPU
      WRITE (6,1030) (D(J), J=1,10)
      STOP
1000  FORMAT (' LOOP COUNT =',I3)
1010  FORMAT (' LAST TERM OF COMPUTED ARRAY =',F10.4)
1020  FORMAT (' TOTAL FPS-164 TIME =',F8.4)
1030  FORMAT (5(2X,F10.4))
      END
```

```
      SUBROUTINE VMUL (N,APCPU1)
      COMMON /ARRY/ A(10),B(10),C(10),D(10)
      DO 100 I=1,N
         D(I) = A(I)*B(I) + C(I)
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE VSQ (N,APCPU2)
      COMMON /ARRY/ A(10),B(10),C(10),D(10)
      DO 100 I=1,N
         D(I) = D(I)*D(I)
100   CONTINUE
      RETURN
      END
```

SAMPLE PROGRAM AFTER MODIFICATION

- INSERT APEX CALLS (APINIT AND APRLSE)
- INSERT I/O TO VERIFY AP CONNECTION
- REQUIRES TWO CALLS TO THE FPS-164, HENCE INEFFICIENT

```
      PROGRAM MAIN
      COMMON /ARRY/ A(10),B(10),C(10),D(10)
      DO 100 I=1,10
         A(I) = FLOAT (I)
         B(I) = FLOAT (I)
         C(I) = FLOAT (I)
 100  CONTINUE
      WRITE (6,1000) I
      N = I-1
|     WRITE (6,9997)
|     READ (5,9998) INUM
|9997 FORMAT (' APNUM =')
|9998 FORMAT (I1)
|     CALL APINIT (INUM, 0, 0, 0, IASG, ISTAT)
|     IF (IASG .EQ. 0) STOP
|     WRITE (6,9999) IASG,ISTAT
|9999 FORMAT (' IASG =',I3,' ISTAT =',I6)
      CALL VMUL (N,APCPU1)
      WRITE (6,1010) D(N)
      CALL VSQ (N,APCPU2)
      APCPU = APCPU1 + APCPU2
|     CALL APRLSE
      WRITE (6,1020) APCPU
      WRITE (6,1030) (D(J), J=1,10)
      STOP
 1000 FORMAT (' LOOP COUNT =',I3)
 1010 FORMAT (' LAST TERM OF COMPUTED ARRAY =',F10.4)
 1020 FORMAT (' TOTAL FPS-164 TIME =',F8.4)
 1030 FORMAT (5(2X,F10.4))
      END
```

SAMPLE PROGRAM AFTER REDUCING I/O

- A NEW SUBROUTINE IS CREATED THAT CALLS THE TWO SUBROUTINES AND PASSES THE DATA FOR I/O AS A PARAMETER [REDUCES I/O AND APEX64 OVERHEAD]

- THE HOST PROGRAM IS MODIFIED TO WRITE THE I/O DATA FROM THE FPS-164

```
      PROGRAM MAIN
      COMMON /ARRY/ A(10),B(10),C(10),D(10)
      DO 100 I=1,10
         A(I) = FLOAT (I)
         B(I) = FLOAT (I)
         C(I) = FLOAT (I)
 100  CONTINUE
      WRITE (6,1000) I
      N = I-1
      WRITE (6,9997)
      READ (5,9998) INUM
 9997 FORMAT (' APNUM =')
 9998 FORMAT (I1)
      CALL APINIT (INUM,0,0,0,IASG,ISTAT)
      IF (IASG .EQ. 0) STOP
      WRITE (6,9999) IASG,ISTAT
 9999 FORMAT (' IASG =',I3,' ISTAT =,I6)
|     CALL SUB (N,APCPU,PARM)
|     WRITE (6,1010) PARM
      WRITE (6,1020) APCPU
      WRITE (6,1030) (D(J), J=1,10)
      STOP
 1000 FORMAT (' LOOP COUNT =',I3)
 1010 FORMAT (' LAST TERM OF COMPUTED ARRAY =',F10.4)
 1020 FORMAT (' TOTAL FPS-164 TIME =',F8.4)
 1030 FORMAT (5(2X,F10.4))
      END
```

```
|     SUBROUTINE SUB (N,APCPU,PARM)
|     COMMON /ARRY/ A(10),B(10),C(10),D(10)
|     CALL VMUL (N,APCPU1)
|     PARM = D(N)
|     CALL VSQ (N,APCPU2)
|     APCPU = APCPU1 + APCPU2
|     RETURN
|     END
```

SAMPLE FPS-164 SUBROUTINE MINIMIZING I/O

- USES APROUTINE [OR APFUNCTION]
- VARIABLES ARE DEFINED AS EITHER IN OR OUT

```
      APROUTINE SUB (N,APCPU,PARM)
      COMMON /ARRY/ A(10),B(10),C(10),D(10)
|     APIN A,B,C
|     APIN N
|     APOUT D
|     APOUT APCPU,PARM
      CALL VMUL (N,APCPU1)
      PARM = D(N)
      CALL VSQ (N,APCPU2)
      APCPU = APCPU1 + APCPU2
      RETURN
      END
```

HOST COMPILE AND LINK

- COMPILE MAINLINE + HASI
- LINK OBJECTS/TEXT WITH FPS-SUPPLIED LIBRARIES:
  - HARDWARE RUN: APEX64, UTIL64
  - HARDWARE DEBUG: APDEBUG64, APEX64, UTIL64
  - SIMULATOR RUN: APSIM64, UTIL64
  - SIMULATOR DEBUG: APDEBUG64, APSIM64, UTIL64

order significant & controlled by global statement

APEX64 Exercise

This exercise is designed to let you convert an existing program to run in APEX64 mode.

1. Modify the APFTN64 program "WXYZ" (written earlier) to be called as a subroutine from a FORTRAN program executing on your host machine.

   - W, X, Y, and Z each have 20 elements, and i varies from 1 to 20
   - Use APROUTINE to declare the entrypoint
   - W, X, and Y are passed from the host in a COMMON block called DATAIN
   - Z and a number representing the length of each vector are passed as arguments to WXYZ

2. Use APFTN64 to compile the WXYZ subroutine.

3. Use APLINK64 to create the load module and the HASI.

4. Write a mainline program which will execute on your host and call the subroutine WXYZ. In this mainline, create the input values for the vectors W, X, and Y. Also include the necessary APEX64 calls to assign and release the FPS-164.

5. Compile the mainline and the HASI using your host's FORTRAN compiler.

6. Link the compiled mainline and HASI together and search the FPS supplied libraries for the APEX64 subroutines:

   - VAX/VMS: $ LINK main,hasi,FPS164:APEX64/L,UTIL64/L
   - IBM/CMS: GLOBAL TXTLIB APEX64 UTIL64 ...

7. Execute your complete module.

ADDITIONAL TECHNIQUES

- COMMON BLOCKS

  ALL COMMON BLOCKS USED IN BOTH AP ROUTINES AND THE THE HOST PROGRAM MUST BE DEFINED IN THE SUBROUTINE CALLED DIRECTLY FROM THE MAINLINE PROGRAM.

- EXAMPLE:

  - HOST PROGRAM

```
COMMON  /A/ . .
COMMON  /B/ . . .
COMMON  /C/ . . .
   .
   .
   .
CALL  APINIT(...)
CALL  FPS164
CALL  APRLSE
   .
   .
   .
STOP
```

  - AP ROUTINES FPS164

```
SUBROUTINE FPS164          SUBROUTINE SUB
COMMON  /A/...             COMMON /B/...
   .                          .
   .                          .
   .                          .
CALL SUB                      .
   .                       RETURN
   .
   .
RETURN
```

ADDITIONAL TECHNIQUES

- EQUIVALENCE STATEMENTS

  TWO OR MORE DATA TYPES CANNOT BE MIXED IN AN ARRAY THAT IS TRANSFERRED TO/FROM THE AP

  THE HASI CAUSES THE TRANSFER OF THE WHOLE ARRAY AS ONE DATA TYPE.

- UNINITIALIZED DATA

  THE APFTN64 COMPILER DOES NOT INITIALIZE THE DATA IN ARRAYS.

- LITERALS

  PARAMETERS IN SUBROUTINE CALLS FROM THE HOST TO THE FPS164 SHOULD NOT BE PASSED AS LITERALS

  I.E. CALL SUB (1,A) SHOULD BE REPLACED WITH

```
IA = 1
CALL SUB  (IA, A)
```

INTERPRETATION OF ERROR MESSAGES

- DOUBLE ERROR DETECT

  DOUBLE-BIT PARITY ERROR DETECTED

- LOAD MODULE ID's DO NOT MATCH

  THE IMAGE FILE (LOAD MODULE) ID AND HASI ID ARE NOT THE SAME - DUE TO NEW APLINK64 AND FAILURE TO RECOMPILE HASI

- INVALID I/O CHANNEL

  THE FPS-164 WAS NOT ASSIGNED TO THE USER
  - NO APINIT IN HOST PROGRAM

- INSUFFICIENT MAP REGISTERS

  TOO MANY PARAMETERS/ARRAYS FOR MAP REGISTERS IN THE HOST(VAX)
  - CANNOT BE FORESEEN BY USER
  - USE APEX ROUTINE
    CALL APSETS (-)

- HISP DETECTED ERRORS

  GENERALLY A HARDWARE PROBLEM EXISTS
  - POSSIBLE SOLUTION IS TO FORCE STEP MODE BY AN APEX SUBROUTINE
  - CALL APMODE (2)

INTERPRETATION OF ERROR MESSAGES

- FORMAT OVERFLOW/UNDERFLOW

  THE DATA TRANSFERRED BACK TO THE HOST EXCEEDS THE DYNAMIC RANGE OF THE HOST COMPUTER FORMAT
  THE MESSAGE CAN BE MASKED BY AN APEX SUBROUTINE
  CALL APWUCM (- - -)

- DEVICE NOT IN CONFIGURATION

  HOST OPERATING SYSTEM IS NOT AWARE OF THE FPS-164
  ---X64 MANAGER IS NOT ACTIVE ON THE HOST (VAX)

- INSUFFICIENT PS FOR LOAD MODULE . . .

  WARNING MESSAGE ONLY
  INFORMS USER THAT APEX IS REALLOCATING THE AP MEMORY TO SUIT THE USER PROGRAM.

- ARRAY OUT OF BOUNDS

  SUBSCRIPT CHECKING WAS PERFORMED AND FOUND AN ARRAY ADDRESSING ERROR.

- PROM HAS DETECTED ERRONEOUS . . . . (HISP)

  THE RESULT OF A MISMATCH OF THE DATA TYPES BETWEEN THE HOST COMPUTER AND THE AP
  ---INCONSISTENT USAGE OF DOUBLE PRECISION

TIMING PROGRAMS ON THE FPS-164

- APEX ROUTINE CAN ACCESS REGISTERS IN THE FPS-164 FROM THE HOST COMPUTER.

- EXAMINE HARDWARE CYCLE COUNTER

```
DIMENSION   IA (2)
CALL APEXAM  (IA, 38, 232)
```

  IA IS THE RETURNED VALUES
  38,232 ARE PARAMETERS SPECIFYING THE HARDWARE CYCLE COUNTER

- EXAMINE THE USER ELAPSED TIME COUNTER

```
DIMENSION  ISA(2)
CALL APEXAM  (ISA, 12, 0)
```

ISA IS THE RETURNED VALUES
12, 0 ARE PARAMETERS SPECIFYING THE USER CYCLE COUNTER

TIMING PROGRAMS ON THE FPS-164

- SYS$CLTIME AND SYS$RDTIME

  SYS$CLTIME CLEARS USER SOFTWARE TIMING REGISTERS

  SYS$RDTIME READS USERS ACCUMULATED TIME FROM SOFTWARE REGISTERS

- EXAMPLE:

```
      SUBROUTINE VMUL (N,APCPU)

      COMMON /ARRY/ A(10),B(10),C(10),D(10)
      CALL SYS$CLTIME
      DO 100 I=1,N
         D(I) = A(I)*B(I) +C(I)
 100  CONTINUE
      CALL SYS$RDTIME(CYCLES)
      APCPU =        0.181818E-06*CYCLES
      RETURN
      END
```

LAB: USING TIMING UTILITIES

1. MODIFY "WXYZ" TO INCLUDE AP-RESIDENT TIMING ROUTINES. DISPLAY EXECUTION TIME.

2. RECOMPILE AT DIFF. OPT. LEVELS TO SEE IF EXECUTION TIMES CHANGE.

Floating Point Systems, Inc.,
Corporate Training Department Materials

AP EXECUTIVE (APEX64)

FPS-164 Software Programming Class
Revision

Primary Functions of Apex

- Initialization of AP
- Release of AP
- Execution of HASI
- Primitive access to AP
- Debugging Support

APEX/SUM INTERACTION

- APEX64 initializes task contro tables in SUM memory.

- APEX64 sends commands to SUM t

  *Allocate memory

  *Start program running.

SUM/APEX INTERACTION

- SUM commands HISP to interrupt host when task completes.
- SUM receives control on FTN PAUSE and STOP.
- Under SUM, AP never physically halts.

HOST INTERFACE SUPPORT PROCESSOR

micro in FPS-164.

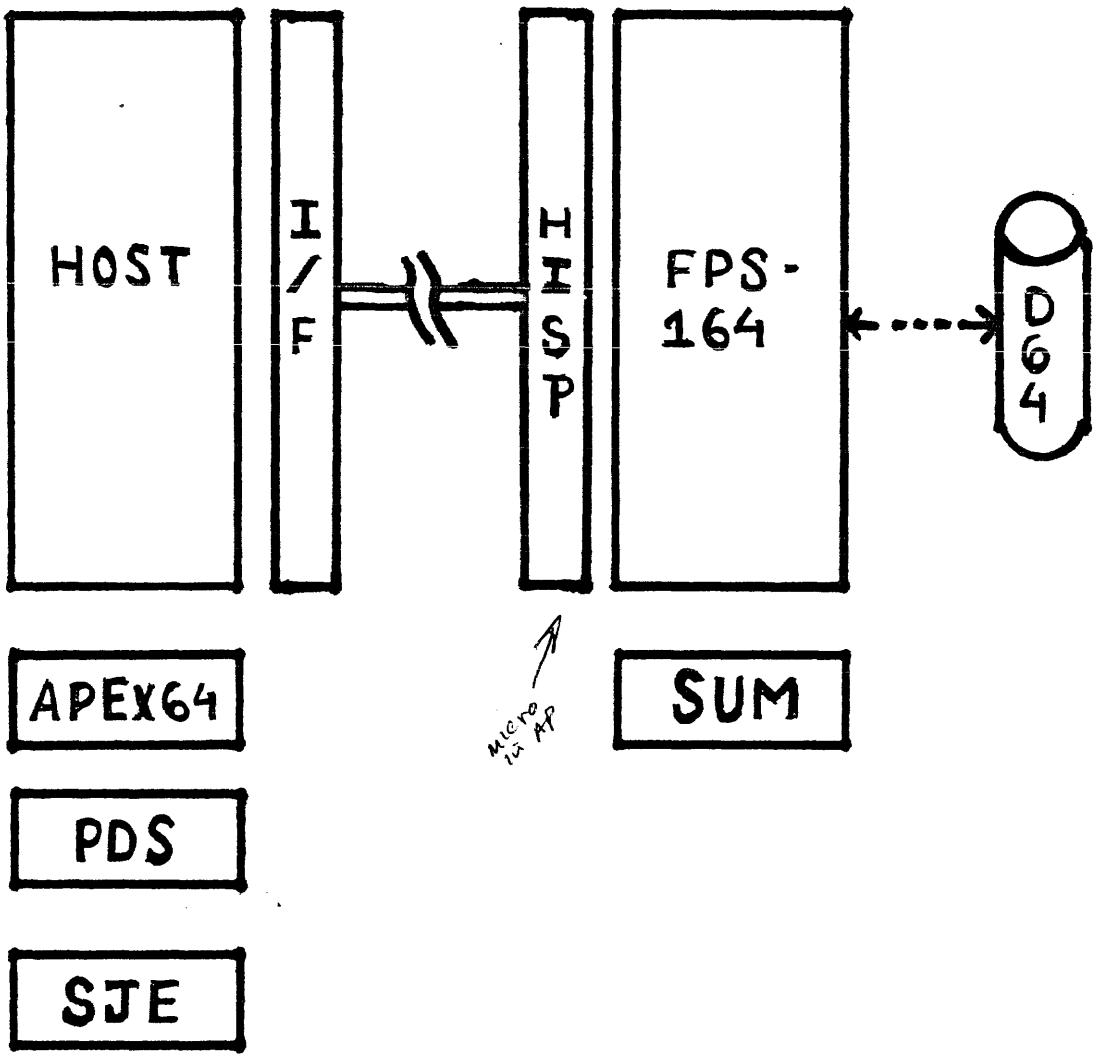
- PURPOSE:

  REDUCE HOST OVERHEAD

- ACTIVITIES:
  - ADDRESS TRANSLATION
  - SETUP DMA CHANNELS

    Direct memory access.
  - FORMAT DATA DURING DMA

SYSTEM ELEMENTS



DATA DESCRIPTOR BLOCK

- DEFINITION:

  A HISP-RESIDENT CHANNEL PROGRAM USED TO CONTROL DATA TRANSFERS

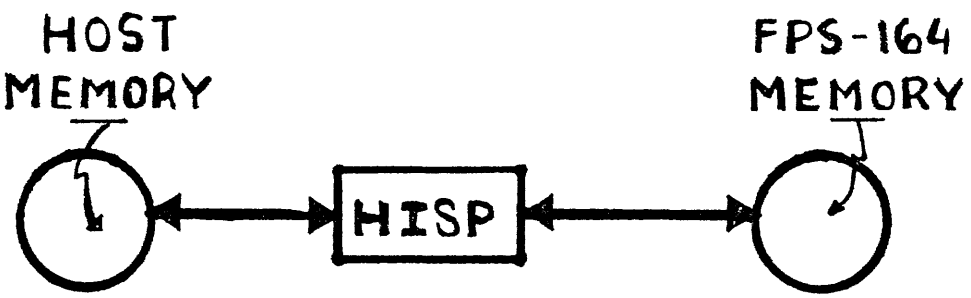
CHANNEL PROGRAMMING

- DEFINITION:

  "BUNDLING TOGETHER" A SERIES OF I/O CALLS

- PURPOSE:

  REDUCE HOST OVERHEAD



DEFINITIONS:

APEX64 EXECUTION MODES

- Step mode
- Chain mode
- Automatic mode

  Substep threshold

  Chaining threshold

SUBSTEP MODE

APEX64 EXECUTION MODES

- Mode control (APMODE)

- Mode reporting (APGMOD)

APLINK64

- PURPOSE:

  PROCESS APFTN64 + APAL64
  OUTPUT + COMBINE WITH
  PRIVATE + FPS-SUPPLIED
  AP OBJECT LIBRARIES

- FUNCTIONS:
  - MEMORY SPACE ALLOCATION
  - RESOLVE EXTERNALS
  - CODE RELOCATION

APLINK64

- OUTPUT:
  - FOR APEX64-mode SUB-ROUTINES CREATES HASI
  - CREATES LOAD MODULE FOR BOTH APEX64-mode + SIE SUBROUTINES
  - OPTIONAL LOAD MAPS AND SYMBOL TABLE

HOST-TO-AP SOFTWARE INTERFACE (HASI)

- PURPOSE:

  PROVIDES HOST/AP SOFTWARE COMMUNICATION AND CONTROL

- 2 TYPES:

  ADC + UDC

ADC VS. UDC HASI

- Data transfer between the host and the AP:

  building DDBs

  executing DDBs.

- Load module loading.

- Load module execution.

AUTO-DIRECTED CALLS (ADC)

- ALLOWS AP SUBROUTINES TO BE TREATED LIKE NORMAL FORTRAN SUBROUTINES
- ALL DATA TRANSFER AUTOMATIC
- HOST/AP EXECUTION IS SERIAL AND SYNCHRONIZED
- USED BY SJE AND APEX64

```
      SUBROUTINE VADDAPF (A0, A1, A2, A3)
C
      REAL A0(100), A1(100), A2(100)
      INTEGER A3
C
      INTEGER A4(6), A5(66)
C
      CALL VADDAP
      CALL APIDB (A5, 66)
C
      A4(1) = 4
      A4(2) = 16
      A4(3) = A4(2) + 100
      A4(4) = A4(3) + 100
      A4(5) = A4(4) + 100
      A4(6) = A4(5) + 1
      CALL APBDD (A5, A4, 5, 0, 11, 2, 1, 1)
C
      CALL APBDD (A5, A0, 100, 0, A4(2), 2, 5, 1)
      CALL APBDD (A5, A1, 100, 0, A4(3), 2, 5, 1)
      CALL APBDD (A5, A2, 100, 0, A4(4), 2, 5, 0)
      CALL APBDD (A5, A3, 1, 0, A4(5), 2, 0, 1)
C
      CALL APPREP (A5, 8, 11, A4(6), 0)
      CALL APXDDB (A5, 1, 1)
      CALL APWR
      CALL APXDDB (A5, 0, 0)
      CALL APXTSK
      RETURN
      END
C
      SUBROUTINE VADDAP
C
      INTEGER A0(2), A1(9), A2(134)
C
      DATA A0(       1)/ :11143564551 /, A0(       2)/ :37737777306 /
C
      DATA A1(       1)/ :00000000111 /, A1(       2)/ :00000000137 /
      DATA A1(       3)/ :00000000126 /, A1(       4)/ :00000000101 /
      DATA A1(       5)/ :00000000104 /, A1(       6)/ :00000000104 /
      DATA A1(       7)/ :00000000101 /, A1(       8)/ :00000000120 /
      DATA A1(       9)/ :00000000106 /
C
      CALL APLMLD (5, A1, 9, A0, A2)
      RETURN
      END
```

HOST CALLABLE ENTRYPT.

initialize data descriptor block (DDB) template

MAIN Memory Address Setup

BUILD DATA DESCRIPTOR

put it here.

transfer type

create script

TRANSFER ARGS., COMMON, AND DATA, AND START SUBRTN.

RETRIEVE RESULTS

start script execution

does it in either chain or step mode; decide on most efficient.

TRANSFER LOAD MODULE

User Directed Calls (UDC)

- Greater Flexibility Of Control
- Mainline More Complex
- Host/AP Execution Is Asynchronous And Parallel
- Advantage:

  May Reduce Overall Pgm. Execution Time

```
      SUBROUTINE VADDUDC (A0, A1, A2, A3, A4, A5, A6)
C
      INTEGER A0
      INTEGER A1, A2, A3, A4, A5, A6
C
      INTEGER A7(8), A8(50)
C
      CALL VADDUD
      CALL APIDB (A8, 50)
C
      A7(1) = 7
      A7(2) = A0
      A7(3) = A1
      A7(4) = A2
      A7(5) = A3
      A7(6) = A4
      A7(7) = A5
      A7(8) = A6
      CALL APBDD (A8, A7, 8, 0, 0, 2, 1, 1)
C
      CALL APPREP (A8, 2, 0, 8, 0)
      CALL APXDDB (A8, 1, 1)
      RETURN
      END
C
      SUBROUTINE VADDUD
C
      INTEGER A0(2), A1(11), A2(38)
C
      DATA A0(       1)/'10574766357'O/, A0(       2)/'12620242117'O/
C
      DATA A1(       1)/'00000000126'O/, A1(       2)/'00000000101'O/
      DATA A1(       3)/'00000000104'O/, A1(       4)/'00000000104'O/
      DATA A1(       5)/'00000000125'O/, A1(       6)/'00000000104'O/
      DATA A1(       7)/'00000000103'O/, A1(       8)/'00000000056'O/
      DATA A1(       9)/'00000000111'O/, A1(      10)/'00000000115'O/
      DATA A1(      11)/'00000000107'O/
C
      CALL APLMLD (5, A1, 11, A0, A2)
      RETURN
      END
```

ENTRY POINT

TRANSFER ARGS. AND START SUB.

TRANSFER LOAD MODULE

THE UDC MAINLINE

1. DIMENSION 2 INTEGER ARRAYS FOR THE DDB'S
2. SET UP MD ADDRESSES
3. INITIALIZE HOST → AP DDB WITH APIDB
4. INITIALIZE AP WITH APINIT
5. FORCE STEP MODE WITH APMOI
6. CREATE DDB'S TO TRANSFER DATA TO THE AP WITH APPU

THE UDC MAINLINE

7. EXECUTE THE DDB WITH APXDDB
3. SYNCHRONIZE WITH APWD
9. CALL THE AP SUBROUTINE
0. PERFORM HOST ACTIVITY. USE APWR + APWD TO SYNCHRONIZE
1. SETUP AP→HOST DDB WITH APIDB
2. CREATE DDB'S TO TRANSFER DATA BACK TO THE HOST WITH APGET

THE UDC MAINLINE

13. EXECUTE THE DDB WITH APXDI
14. SYNCHRONIZE WITH APWD
15. RELEASE THE AP WITH APRLSE

Mainline Calling a UDC Mode Routine

```
C
C       FORTRAN mainline program to call the UDC mode APAL
C       routine VADDUDC.
C
      DIMENSION A(100),B(100),C(100)
      INTEGER AINC,BINC,CINC,CTR
C
C       Now set up matricies to contain the data descriptors
C       for data transfer.  The matricies must be of type
C       INTEGER and the size to specify is determined as:
C            ( 4 * no. of APPUT's or APGET's) + 6
C
      INTEGER INDDB(30),OUTDDB(10)
C
C       Create dummy values for the A and B matricies
C
      CTR = 100
      DO 10 I=1,CTR
      A(I) = I
      B(I) = (CTR + 1) - I
10    CONTINUE
C
C       Setup element increments
C
      AINC = 1
      BINC = 1
      CINC = 1
C
C       Now setup the main memory addresses for each parameter
C       passed over to the AP164.  Where we chose to start the
C       addresses is purely arbitrary.
C
      IAPTR = 10
      IBPTR = IAPTR + CTR
      ICPTR = IBPTR + CTR
      ICTR  = ICPTR + CTR
      IAINC = ICTR  + 1
      IBINC = IAINC + 1
      ICINC = IBINC + 1
C
C       Now initialize the DDB for host to AP transfer
C
      CALL APIDB(INDDB,30)
C
C       Assign the AP164
C
      CALL APINIT(0,0,4096,4096,INUM,ISTAT)
C
C       Force STEP mode for UDC mode execution
C
      CALL APMODE(2)
```

STEP 1

STEP 2

STEP 3

STEP 4

STEP 5

Mainline Calling a UDC Mode Routine

```
C
C       Setup the calls to transfer the data over to the AP
C
        CALL APPUT(INDDB,A,IAPTR,CTR,5)
        CALL APPUT(INDDB,B,IBPTR,CTR,5)
        CALL APPUT(INDDB,CTR,ICTR,1,0)
        CALL APPUT(INDDB,AINC,IAINC,1,0)
        CALL APPUT(INDDB,BINC,IBINC,1,0)
        CALL APPUT(INDDB,CINC,ICINC,1,0)
        CALL APXDDB(INDDB,0,1)
C
C       Wait until the DMA is complete
C
        CALL APWD
C
C       Call the APAL subroutine
C
        CALL VADDUDC(IAPTR,IBPTR,ICPTR,ICTR,IAINC,IBINC,ICINC)
C
C       Take advantage of UDC mode by having the host perform some
C       I/O to the terminal while the AP subroutine is busy.
C
        WRITE(6,101) (I,A(I),B(I), I= 1,CTR)
101     FORMAT(100(2X,'Elements ',I3,2(3x,F5.1),/))
C
C       Wait until the AP program has completed.
C
        CALL APWR
C
C       Setup the DDB for transferring the results back from the AP
C
        CALL APIDB(OUTDDB,10)
        CALL APGET(OUTDDB,C,ICPTR,CTR,5)
        CALL APXDDB(OUTDDB,0,0)
C
C       Wait until the DMA is complete
C
        CALL APWD
C
C       Release the AP164
C
        CALL APRLSE
C
C       Now display the results
C
        WRITE(6,100) (A(I),B(I),C(I),I=1,CTR)
100     FORMAT(100(3(F5.1,3X),/))
        CALL EXIT
        END
```

address in AP

STEP 6

STEP 7

STEP 8

STEP 9

STEP 10

STEP 11

STEP 12

STEP 13

STEP 14

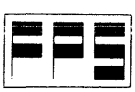
STEP 15

COMPARISON: ADC -vs- UDC

- BOTH USE THE HASI
- BOTH ALLOW DISK OR MEMORY RESIDENT LOAD MODULES
- ADC EASIER TO DEVELOP + USE
- UDC MAY EXECUTE FASTER, IF PGM. PROPERLY DESIGNED

LAB: ACCESS A UDC SUB.

1. USE "WXYZ" SUBROUTINE FROM PREVIOUS LABS
2. REBUILD + CREATE A UDC HASI. INCLUDE TIMING SUBROUTINES AND GENERATE LISTINGS FROM BOTH APFTN64 + APLINK64
3. REWRITE MAINLINE FOR UDC MODE. USE 10,000 AS MD BASE ADDR.
4. BUILD ALL PIECES + EXECUTE. IS AP SUBROUTINE EXECUTION TIME DIFFERENT THAN WITH THE ADC SUBROUTINE ?



FLOATING POINT
SYSTEMS, INC.

...the world leader in array processors

CALL TOLL FREE (800) 547-1445
Ex. 4999. P.O. Box 23489 (S 500),
Portland, OR 97223 (503) 641-3151,
TLX: 360470 FLOATPOIN BEAV