intel

iAPX 86,88,186 MICROPROCESSORS PART II

WORKSHOP NOTEBOOK

VERSION 2.0 JULY 1984

Order No. 210954-002

intel

iAPX 86,88,186 MICROPROCESSORS PART II

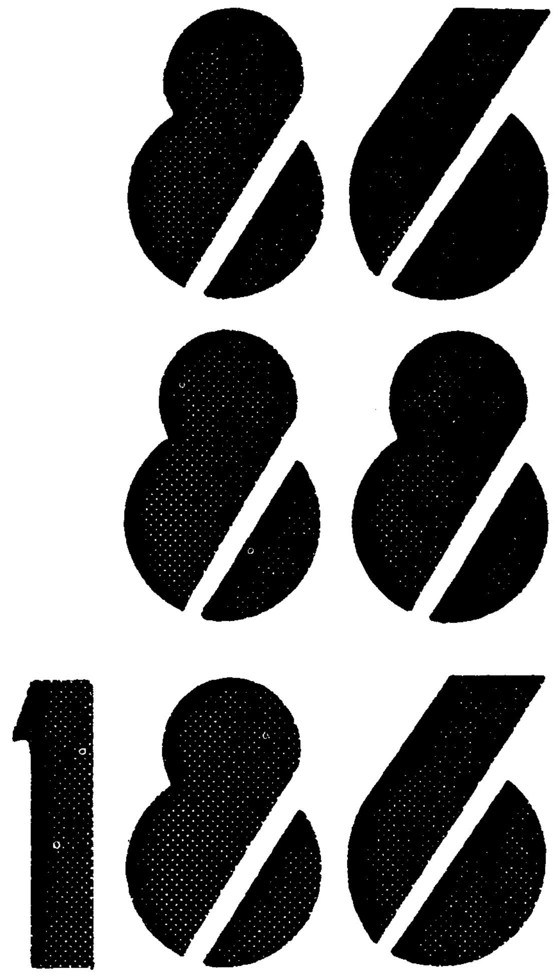


TABLE OF CONTENTS

iAPX 86,88,186 MICROPROCESSORS PART II

WORKSHOP SCHEDULE

DAY 1 OBJECTIVES

BY THE TIME YOU FINISH TODAY YOU WILL:

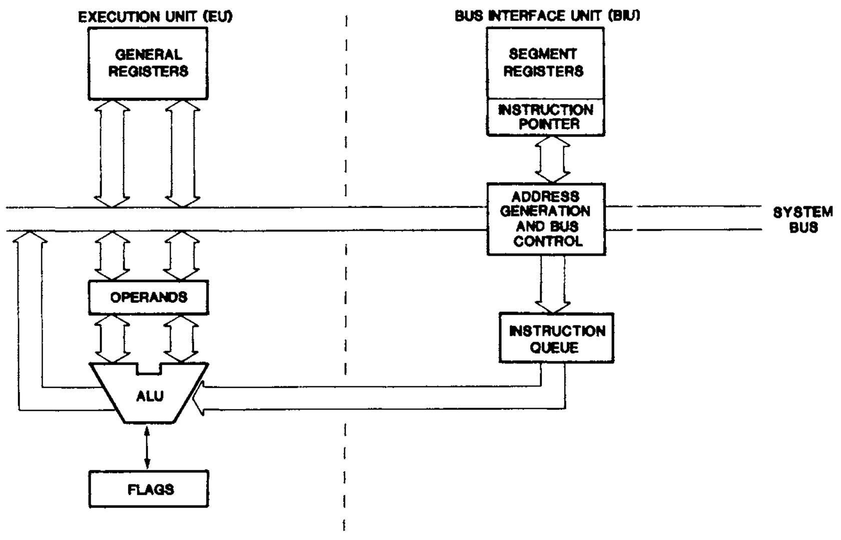
- REVIEW BASIC 8086 ARCHITECTURE AND SEGMENTATION CONCEPTS
- REVIEW BASIC ASM86 CONCEPTS
- SEE THE ENTIRE INSTRUCTION SET OF THE 8086/88
- USE ADVANCED SEGMENT ATTRIBUTES (ALIGN-TYPE, COMBINE-TYPE, CLASSNAMES)
- USE MODULAR PROGRAMMING TECHNIQUES

CHAPTER 1

ARCHITECTURAL REVIEW

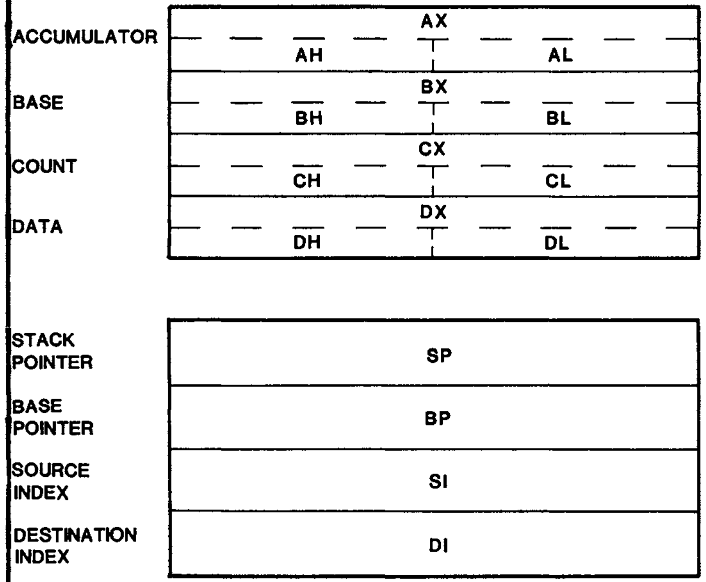
- DESCRIPTION OF THE iAPX 86,88
- REVIEW OF THE iAPX 86,88 ARCHITECTURE

INTERNAL ARCHITECTURE



- BIU PERFORMS ALL BUS TRANSFERS
- EU EXECUTES ALL INSTRUCTIONS
- INSTRUCTION FETCHES OVERLAPPED WITH INSTRUCTION EXECUTION

GENERAL REGISTERS



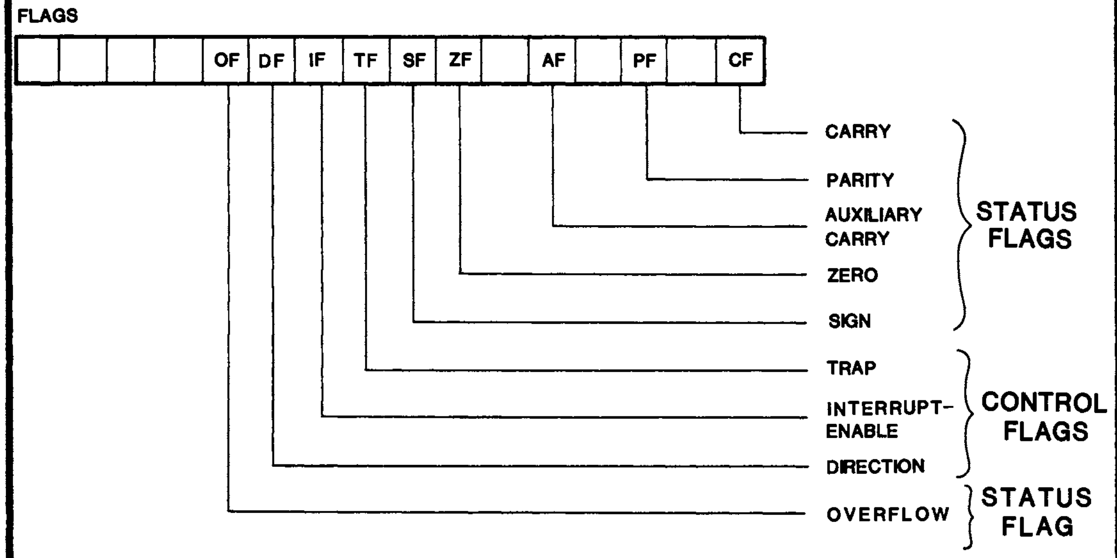
- DATA GROUP

  AX (AL/AH)
  BX (BL/BH)
  CX (CL/CH)
  DX (DL/DH)

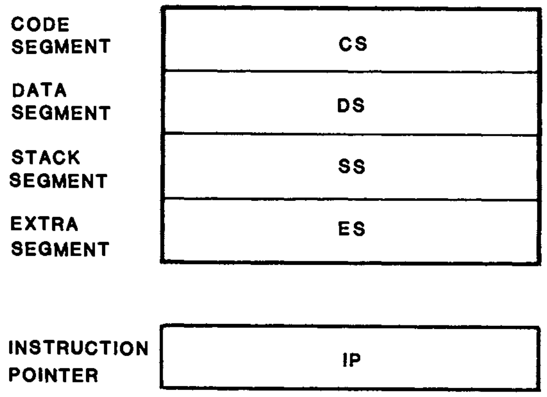
- POINTER AND INDEX GROUP

SP	BX
BP	DX (I/O)
SI	
DI	

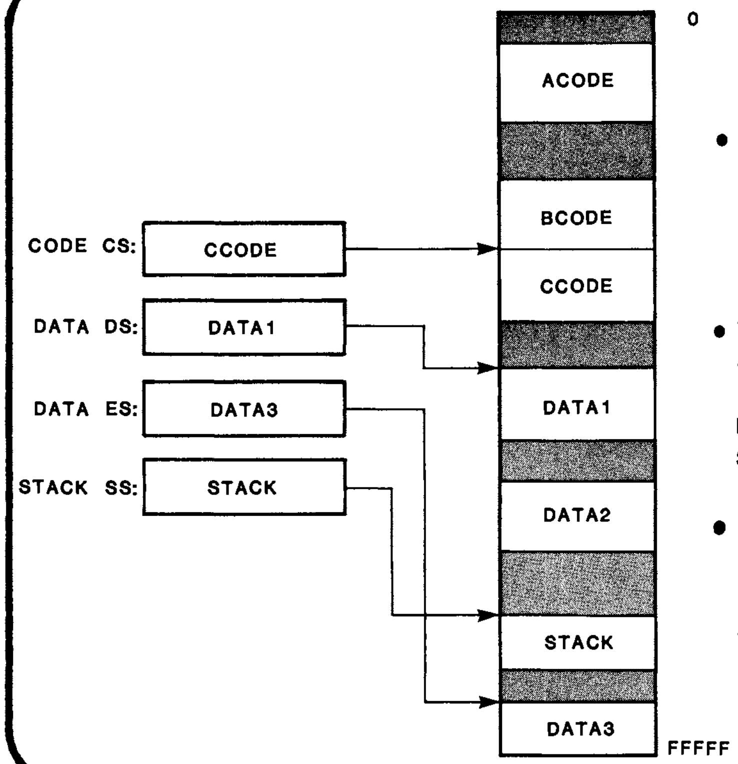
FLAG WORD



SEGMENT REGISTERS AND INSTRUCTION POINTER

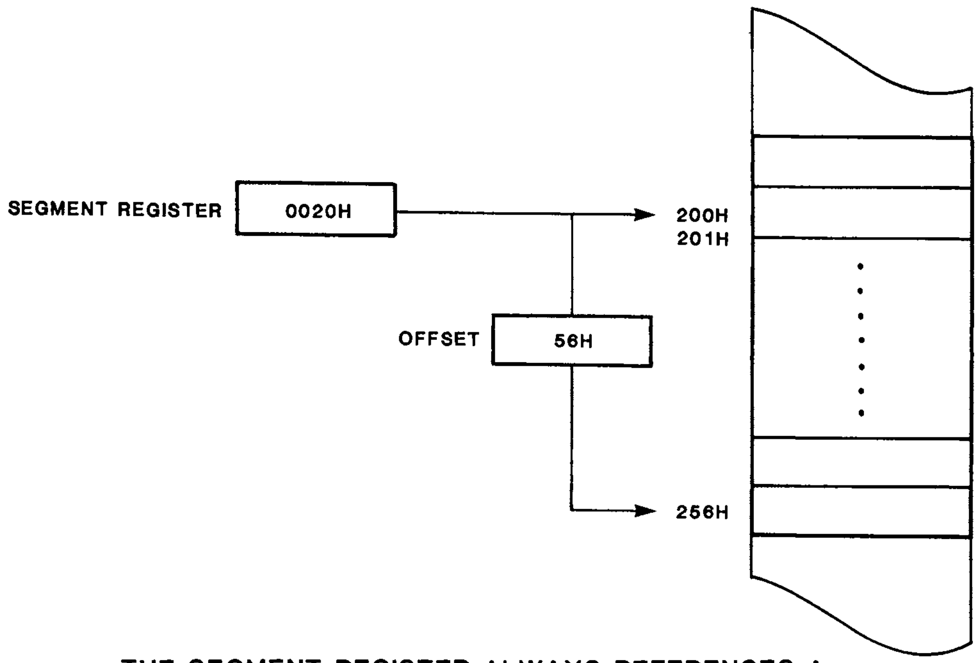


- MEMORY IS MAPPED INTO LOGICAL SEGMENTS OF UP TO 64K BYTES EACH
- THE SEGMENT REGISTERS POINT TO THE FOUR CURRENTLY ADDRESSABLE SEGMENTS
- THE IP AND CS REGISTER WORK TOGETHER FOR ADDRESSING INSTRUCTION MEMORY



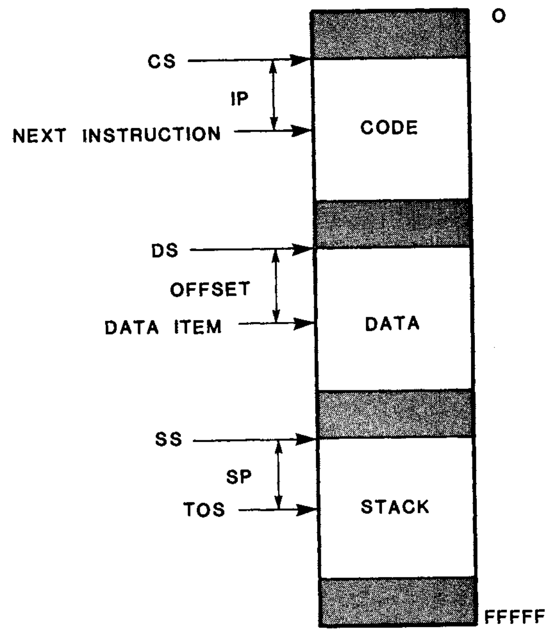
- THE 8086 CAN ACCESS ANY ITEM THAT RESIDES IN A SEGMENT CURRENTLY POINTED TO BY ONE OF THE SEGMENT REGISTERS.
- TO ACCESS ITEMS IN OTHER SEGMENTS, THE SEGMENT REGISTER IS CHANGED TO POINT TO THE OTHER SEGMENT
- WHAT IS THE MAXIMUM AMOUNT OF MEMORY THAT THE 8086 CAN ACCESS AT ANY GIVEN INSTANT?

SEGMENT REGISTER CONTENTS



- THE SEGMENT REGISTER ALWAYS REFERENCES A HEXADECIMAL ADDRESS BOUNDARY IN MEMORY

APPLICATION OF THE SEGMENT REGISTERS



WHERE TO FIND MORE INFORMATION

iAPX 86.88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)

CHAPTER 3 - ARCHITECTURE AND INSTRUCTIONS

CHAPTER 2
INSTRUCTION SET REVIEW : CONSTRUCTS

- INSTRUCTION FORMAT
- DATA DEFINITION
- ASSUME STATEMENT

SEGMENT DEFINITION

```
            NAME       EXAMPLE
STACK       SEGMENT
            ; STACK DEFINITIONS
STACK       ENDS
 DATA       SEGMENT
            ; DATA DEFINITIONS
 DATA       ENDS
 CODE       SEGMENT
            ASSUME CS: CODE, DS: DATA
            ASSUME SS: STACK
            ; EXECUTABLE CODE
 CODE       ENDS
            END
```

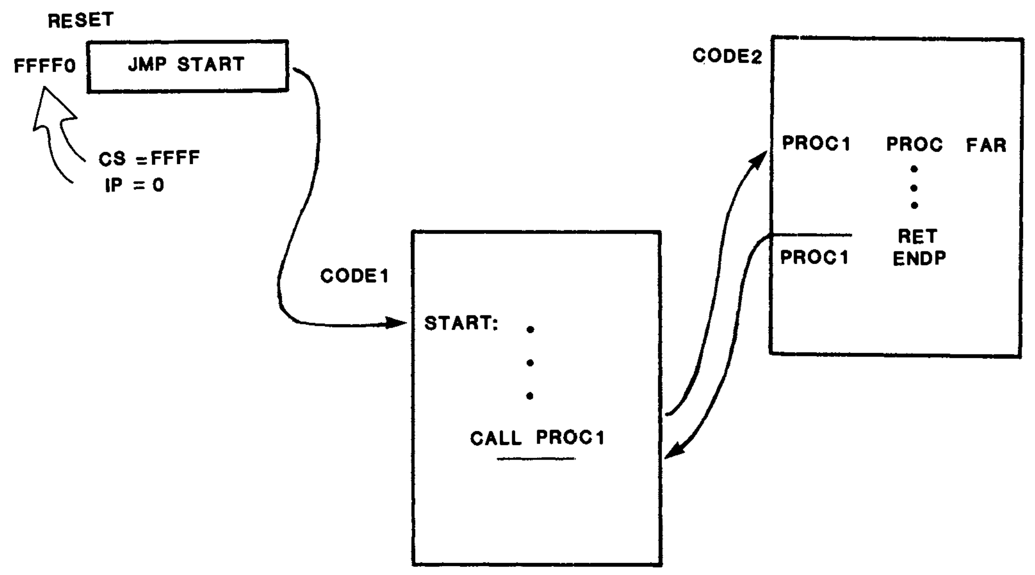
SEGMENT REGISTER INITIALIZATION

```
CODE    SEGMENT

        ASSUME CS:CODE, DS:DATA
        ASSUME SS:STACK

        MOV AX,DATA
        MOV DS,AX
        MOV AX,STACK
        MOV SS,AX
          •
          •
          •
CODE    ENDS
```

- THE ASSUME DIRECTIVE IS A "PROMISE" TO THE ASSEMBLER THAT INSTRUCTIONS AND DATA ARE ADDRESSABLE THROUGH CERTAIN SEGMENT REGISTERS.

- THE ASSUME DIRECTIVE DOES NOT INITIALIZE THE SEGMENT REGISTERS.

INITIALIZATION AND MODIFICATION OF THE CS REGISTER



DATA DEFINITIONS

TYPES

DB	-	DEFINE BYTE
DW	-	DEFINE WORD
DD	-	DEFINE DOUBLE WORD (8087 SHORT REAL, SHORT INTEGER)
DQ	-	DEFINE QUAD WORD (8087 LONG REAL, LONG INTEGER)
DT	-	DEFINE TEN BYTE (8087 PACKED DECIMAL, TEMPORARY REAL)

MORE ON 8087 DATA TYPES IN CHAPTER 10 !

EXAMPLES

```
XYZ      DB   ?            : UNINITIALIZED BYTE
ARRAY    DB   100 DUP (?)  : UNINITIALIZED ARRAY
ABC      DB   3            : INITIALIZED BYTE
MSG1     DB   'WORKSHOPS'  : INITIALIZED ARRAY
PI       DQ   3.142        : INITIALIZED LONG REAL
ANDY     DT   5            : INITIALIZED PACKED DECIMAL
```

ATTRIBUTES OF DATA ITEMS

- FOR EVERY DATA DEFINITION, THE ASSEMBLER KEEPS TRACK OF THREE ATTRIBUTES.
  - SEGMENT
  - OFFSET
  - TYPE
- THE ASSEMBLER USES THESE ATTRIBUTES TO GENERATE THE CORRECT INSTRUCTION FORM.

EXAMPLE:

```
DATA_1  SEGMENT
XYZ     DB        ?
YYY     DW        ?
DATA_1  ENDS
CODE_1  SEGMENT
          .
          .
          .
        MOV     XYZ, 10H   ;BYTE OPERATION
                           ;MOVE 10H INTO MEMORY LOCATION XYZ
        MOV     YYY, 20H   ; WORD OPERATION
          .                ; MOVE 0020H INTO MEMORY LOCATION YYY
          .
          .
```

ASSEMBLY LANGUAGE INSTRUCTIONS

- BYTE OR WORD OPERATIONS USE THE SAME MNEMONIC
- IN GENERAL, BOTH OPERANDS MUST BE THE SAME TYPE, BYTE OR WORD
- MOST OPERATIONS APPLY TO ANY OF THE GENERAL REGISTERS AND/OR MEMORY
- IMMEDIATE DATA CAN ALSO BE SPECIFIED IN AN INSTRUCTION
- EXAMPLES

```
MOV   AL,BL      ; BYTE OPERATION
MOV   AX,BX      ; WORD OPERATION
MOV   BX,AL      ; ILLEGAL

MOV   AL,20      ; BYTE OPERATION
MOV   BX,20      ; WORD OPERATION
MOV   FRED,10    ; WORD OPERATION (TYPE OF FRED IS WORD)
```

THE MEMORY OPERAND

- MANY INSRUCTIONS CAN REFERENCE AN OPERAND IN MEMORY

  EG ADD FRED,1

- OFFSET OF OPERAND MAY BE SPECIFIED BY ......

$$\text{OFFSET} = \begin{bmatrix}\text{VARIABLE}\\ \text{NAME}\end{bmatrix} + \begin{bmatrix}[\text{BX}]\\ [\text{BP}]\end{bmatrix} + \begin{bmatrix}[\text{SI}]\\ [\text{DI}]\end{bmatrix} + \begin{bmatrix}\text{DISPLACEMENT}\end{bmatrix}$$

EG NOT TABLE [BX] - 6

MORE ON THE USE OF ADDRESSING MODES IN CHAPTER 6!

EXERCISE 2.1

IF AN INSTRUCTION HAS THE OPTION OF AN OPERAND IN MEMORY, ANY OF THE AVAILABLE ADDRESSING MODES MAY BE USED.

1. CAN THE XOR INSTRUCTION HAVE A MEMORY OPERAND?
2. IF SO, DOES THE ADDRESSING MODE AFFECT THE TIMING OF THE INSTRUCTION (CLUE - 'EA')
3. WHY SHOULD THE ADDRESSING MODE AFFECT THE TIMING?
4. WHAT IS THE MINIMUM NUMBER OF CLOCKS THE FOLLOWING INSTRUCTION WOULD TAKE? ....

   XOR ARRAY BX ,AX

   LOOK IN THE ASM86 LANGUAGE REFERENCE MANUAL.

   EA = EFFECTIVE ADDRESS CALCULATION TIME.

ASSUME AND SEGMENT OVERRIDE PREFIXES

```
                NAME         EXAMPLE

DATA_1          SEGMENT
XYZ             DW           ?
BUFFER          DB           100 DUP(?)
DATA_1          ENDS

DATA_2          SEGMENT
ABC             DB           ?
DATA_2          ENDS
```

ASSUME AND SEGMENT OVERRIDE PREFIXES (cont)

```
CODE        SEGMENT
            ASSUME      CS:CODE ,DS:DATA-1    ;NO ASSUME FOR ES.
FIVE        DB          5
            MOV         AX, DATA_1
            MOV         DS,AX
            MOV         AX,DATA_2
            MOV         ES,AX
             •
             •
             •
            MOV         DS:XYZ,0            ;DS USED. NO OVERRIDE NECESSARY.
            MOV         AL,DS:BUFFER[5]     ;DS USED. NO OVERRIDE NECESSARY.
            ADD         AL,ES:ABC           ;ES USED. OVERRIDE INSERTED.
            SUB         AL,FIVE             ;CS USED. OVERRIDE INSERTED BY ASM86
            XOR         AX, [BX]            ;DS USED
             •
             •
             •
CODE        ENDS
            END
```

SEGMENT OVERRIDE

- SEGMENT OVERRIDE PREFIX

001	REG	110

ONE BYTE PREFIX TO A MEMORY REFERENCE INSTRUCTION. THE "REG" FIELD IDENTIFIES THE SEGMENT REGISTER TO BE USED IN CALCULATING THE PHYSICAL ADDRESS.

- USE OF SEGMENT OVERRIDE

OFFSET REGISTER	DEFAULT	WITH OVERRIDE PREFIX
IP (CODE ADDRESS)	CS	NEVER
SP (STACK ADDRESS)	SS	NEVER
BP (STACK ADDRESS OR STACK MARKER)	SS	DS, ES, OR CS
BX	DS	ES, SS, OR CS
SI OR DI (NOT INCL. STRINGS)	DS	ES, SS, OR CS
SI (IMPLICIT SOURCE ADDR FOR STRINGS)	DS	ES, SS, OR CS
DI (IMPLICIT DEST ADDR FOR STRINGS)	ES	NEVER

NOTE: IF BP USED IN ADDRESSING MODE
(eg MOV AX, [BP] [SI]),SS IS USED

WHERE TO FIND MORE INFORMATION

ASM86 LANGUAGE REFERENCE MANUAL

CHAPTER 2 -SEGMENTATION
CHAPTER 3- DEFINING AND INITIALIZING DATA

AN INTRODUCTION TO ASM86

CHAPTER 3

INSTRUCTION SET REVIEW

* INSTRUCTION SET BY CLASS

DATA TRANSFER INSTRUCTIONS

GENERAL PURPOSE

MOV	MOV BYTE OR WORD
PUSH	PUSH WORD ONTO STACK
POP	POP WORD OFF STACK
XCHG	EXCHANGE BYTE OR WORD
XLAT	TRANSLATE BYTE

INPUT / OUTPUT

IN	INPUT BYTE OR WORD
OUT	OUTPUT BYTE OR WORD

TRANSLATE INSTRUCTION

- USEFUL FOR TABLE LOOKUP

$$AL \longleftarrow [BX+AL]$$

```
TABLE   DB      10,20,14,17,23,41,60,72
         •
         •
         •
        IN      AL,0
        LEA     BX,TABLE
        XLATB
        OUT     0,AL
         •
         •
         •
```

ADDRESS OBJECT

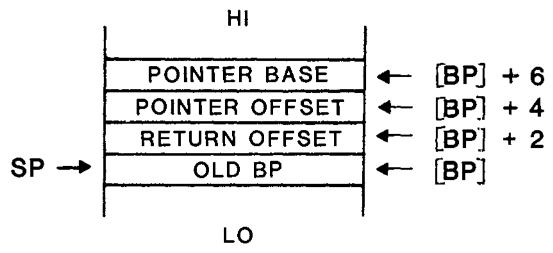
LEA LOAD EFFECTIVE ADDRESS

- * EXAMPLE - LEA BX,TABLE [SI]

LES/LDS LOAD POINTER USING ES/DS

- * USEFUL WITH STRING INSTRUCTIONS
- * USEFUL FOR ACCESSING POINTER PARAMETERS PASSED ON STACK
- * EXAMPLE - LOAD A POINTER PARAMETER FROM A STACK FRAME INTO ES:DI

LES DI , [BP] + 4



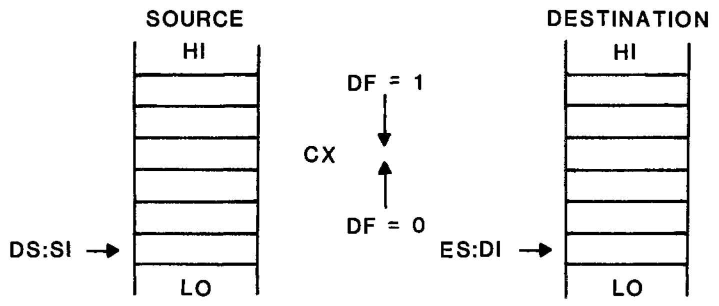
STRING INSTRUCTIONS

- ONE BYTE INSTRUCTIONS WITH AUTO INCREMENT/DECREMENT OF INDEX REGISTERS
- OPERATE ON BYTES OR WORDS
- CAN USE OPERANDS FOR TYPING (BYTE/WORD) OR MNEMOMIC (EG MOVSB, MOVSW)

PRIMITIVES (OPERATE ON SINGLE BYTES/WORDS ONLY)

MOVS	MOVE BYTE OR WORD FROM SOURCE TO DESTINATION STRING
CMPS	COMPARE SOURCE TO DESTINATION STRING
SCAS	SCAN DESTINATION STRING FOR MATCH/NO MATCH WITH AL/AX
STOS	STORE AL/AX TO DESTINATION STRING
LODS	LOAD AL/AX FROM SOURCE STRING

STRING INSTRUCTIONS : REGISTER AND FLAG USE



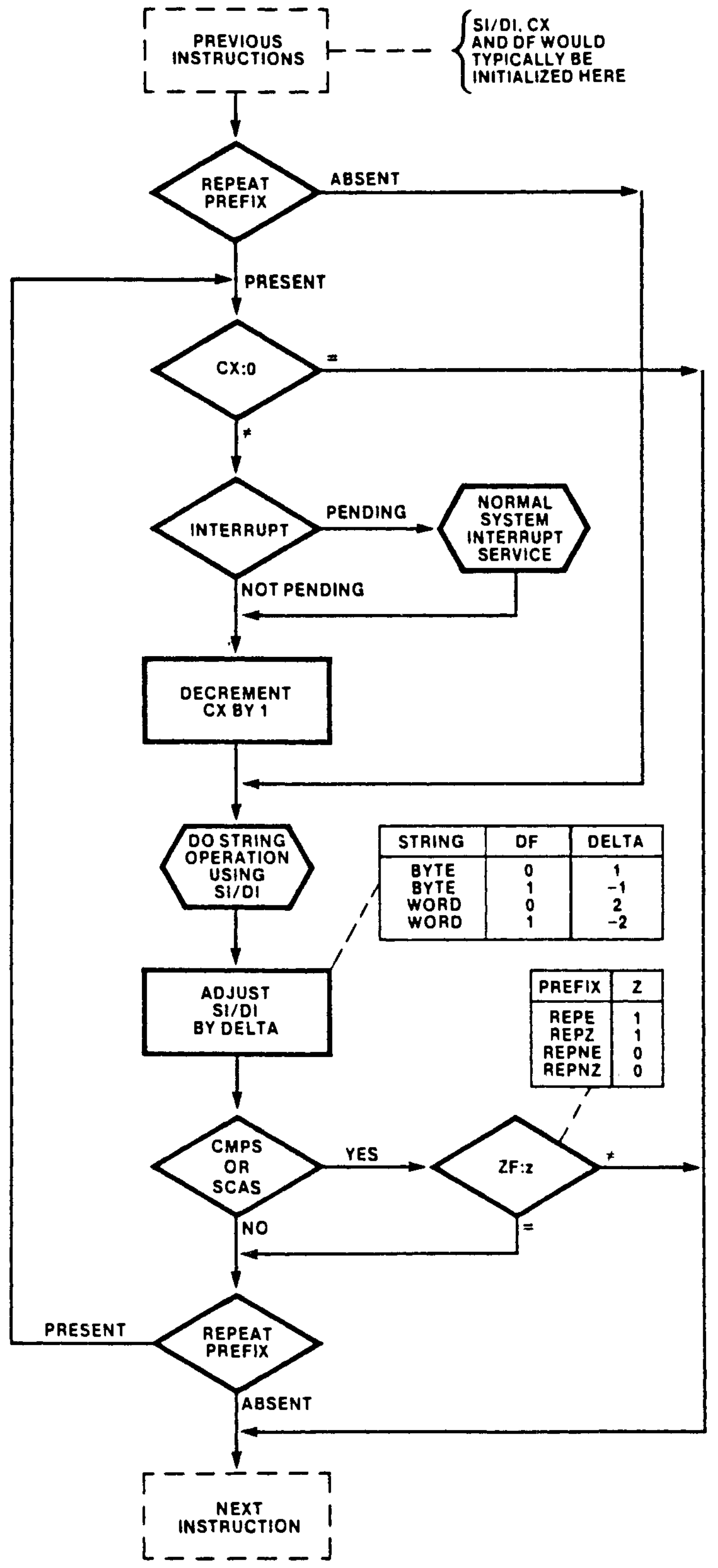
- DIRECTION FLAG SPECIFIES AUTO INCREMENT/DECREMENT OF SI/DI
- CAN OVERRIDE USE OF DS TO ADDRESS SOURCE SEGMENT

REPEAT PREFIXES (REPEAT PRIMITIVE CX TIMES)

REP	REPEAT
REPE/REPZ	REPEAT WHILE EQUAL/ZERO (FLAG SET BY CMPS OR SCAS)
REPNE/REPNZ	REPEAT WHILE NOT EQUAL/NOT ZERO

EXAMPLE : STRING INSTRUCTIONS AND REGISTER USAGE

```
                NAME  CANTEEN_USAGE
DATA_1          SEGMENT
STUDENT         DB   14 DUP (?)
DATA_1          ENDS
DATA_2          SEGMENT
CANTEEN_SEATS DB 50 DUP (?)
DATA_2          ENDS
CODE_1          SEGMENT
                ASSUME   CS:CODE_1, DS:DATA_1, ES: DATA_2
STUD_PTR        DD  STUDENT
CANT_PTR        DD  CANTEEN_SEATS
MOV_IT          PROC
                LDS  SI,STUD_PTR              ; LOAD DS:SI
                LES  DI,CANT_PTR              ; LOAD ES:DI
                MOV  CX,LENGTH STUDENT        ; LOAD REPEAT COUNT
            REP MOVS CANTEEN_SEATS,STUDENT    ; MOVE ALL STUDENTS INTO
                RET                             CANTEEN
MOV_IT          ENDP
CODE_1          ENDS
                END
```



STRING	DF	DELTA
BYTE	0	1
BYTE	1	−1
WORD	0	2
WORD	1	−2

PREFIX	Z
REPE	1
REPZ	1
REPNE	0
REPNZ	0

EXAMPLE : STRING COMPARE

```
         NAME     CHECK_PASSWORD

         PANIC  EQU  ØØH
         EMPLOYEE EQU ØFFH

DATA_1   SEGMENT

REPLY    DB       80 DUP (?)

DATA_1   ENDS

CODE_1   SEGMENT
         ASSUME   CS:CODE_1, DS:DATA_1, ES:CODE_1

CORRECT REPLY  DB  'OPEN SESAME'
REPLY_PTR      DD  REPLY

CHECK    PROC

         LES      DI,REPLY_PTR           ; LOAD ES:DI
         LEA      SI,CORRECT_REPLY       ; CS:SI ADDRESS CORRECT PASSWORD
         MOV      CX,LENGTH CORRECT_REPLY ; LOAD REPEAT COUNT
AGAIN: REPE CMPS  REPLY,CORRECT_REPLY    ; CS OVERRIDE ON SOURCE
         JNE      SPY                    ; STRINGS DID NOT COMPARE
         JCXZ     OK                     ; STRINGS COMPARE
         JMP      AGAIN                  ; REPEAT UNFINISHED
SPY:     MOV      AX,PANIC
         RET
OK:      MOV      AX,EMPLOYEE
         RET

CHECK    ENDP

CODE_1   ENDS

         END
```

ITERATION CONTROLS

- UNCONDITIONAL LOOPS :

LOOP	LOOP CX TIMES

- LOOPS WITH CONDITIONAL TERMINATION :

LOOPE/LOOPZ	LOOP CX TIMES WHILE ZERO FLAG IS SET
LOOPNE/LOOPNZ	LOOP CX TIMES WHILE ZERO FLAG IS RESET

- SAFETY FEATURE FOR USE WITH LOOPS :

JCXZ	SPECIAL JUMP TO TEST COUNT IN CX. A ZERO COUNT WOULD CAUSE A 64K LOOP COUNT.

UNCONDITIONAL TRANSFERS

CALL	CALL PROCEDURE
RET	RETURN FROM PROCEDURE
JMP	JUMP

INTERRUPTS

INT N	SOFTWARE INTERRUPT TYPE N (N = 0 TO 255)
INTO	INTERRUPT IF OVERFLOW FLAG SET
IRET	RETURN FROM INTERRUPT

FLAG INSTRUCTIONS

FLAG TRANSFER

LAHF	LOAD AH FROM FLAGS (LS BYTE OF FLAGS REGISTER)
SAHF	STORE AH INTO FLAGS
PUSHF	PUSH FLAGS ONTO STACK
POPF	POP STACK INTO FLAGS

FLAG OPERATIONS

STC	SET CARRY FLAG
CLC	CLEAR CARRY FLAG
CMC	COMPLEMENT CARRY FLAG
STD	SET DIRECTION FLAG
CLD	CLEAR DIRECTION FLAG
STI	SET INTERRUPT ENABLE FLAG
CLI	CLEAR INTERRUPT ENABLE FLAG

HOW DO I KNOW IF AN INSTRUCTION WILL AFFECT THE FLAGS ?

• SEE ASM86 LANGUAGE REFERENCE MANUAL . . . .

EFFECT CODE	EFFECT
X	MODIFIED BY THE INSTRUCTION ; RESULT DEPENDS ON OPERANDS
–	NOT MODIFIED
U	UNDEFINED AFTER THE INSTRUCTION
1	SET TO 1 BY THE INSTRUCTION
0	SET TO 0 BY THE INSTRUCTION

CONDITIONAL TRANSFERS

- THREE TYPES OF CONDITIONAL JUMP ...
  - FOR UNSIGNED NUMBERS (USE 'ABOVE' AND 'BELOW')
  - FOR SIGNED NUMBERS (USE 'GREATER' AND 'LESS')
  - FOR EITHER (THEY EXAMINE INDIVIDUAL FLAGS)
- OPTIONAL MNEMONICS FOR SOME CONDITIONAL JUMPS
- ALL CONDITIONAL JUMPS ARE SHORT JUMPS (THEY JUMP +127/-128 BYTES)

CONDITIONAL TRANSFERS

UNSIGNED :	JA / JNBE	JUMP IF ABOVE / NOT BELOW OR EQUAL
	JAE / JNB	JUMP IF ABOVE OR EQUAL / NOT BELOW
	JBE / JNAE	JUMP IF BELOW OR EQUAL / NOT ABOVE NOR EQUAL
	JB / JNA	JUMP IF BELOW / NOT ABOVE
SIGNED :	JG / JNLE	JUMP IF GREATER / NOT LESS NOR EQUAL
	JGE / JNL	JUMP IF GREATER OR EQUAL / NOT LESS
	JL / JNGE	JUMP IF LESS / NOT GREATER OR EQUAL
FLAGS:	J(N)C	JUMP IF CARRY FLAG (NOT) SET
	J(N)Z/J(N)Z	JUMP IF ZERO FLAG (NOT) SET
	J(N)O	JUMP IF OVERFLOW FLAG (NOT) SET
	J(N)S	JUMP IF SIGN FLAG (NOT) SET
	JPE/JPO	JUMP IF PARITY EVEN/ODD

BIT MANIPULATION INSTRUCTIONS

LOGICALS

NOT	COMPLEMENT ALL BITS
AND, OR, XOR	LOGICAL AND, OR, EXCLUSIVE OR
TEST	NON-DESTRUCTIVE AND FOR TESTING BITS

SHIFTS (ONE PLACE OR CL TIMES)

SHL/SAL	SHIFT LEFT/ARITHMETIC LEFT
SHR	SHIFT RIGHT
SAR	SHIFT ARITHMETIC RIGHT

ROTATES (ONE PLACE OR CL TIMES)

ROL	ROTATE LEFT
ROR	ROTATE RIGHT
RCL	ROTATE LEFT THROUGH CARRY
RCR	ROTATE RIGHT THROUGH CARRY

3-15

ARITHMETIC INSTRUCTIONS

ADDITION

ADD	ADD
ADC	ADD WITH CARRY
INC	INCREMENT

SUBTRACTION

SUB	SUBTRACT
SBB	SUBTRACT WITH BORROW
DEC	DECREMENT
CMP	COMPARE (NON-DESTRUCTIVE SUBTRACT)
NEG	NEGATE

ARITHMETIC INSTRUCTION (CONT.)

MULTIPLICATION (8*8= 16 BITS OR 16*16 = 32 BITS)

MUL	UNSIGNED MULTIPLY
IMUL	INTEGER MULTIPLY

DIVISION (16 / 8 = 8 BITS OR 32 / 16 = 16 BITS)

DIV	UNSIGNED DIVIDE
IDIV	INTEGER DIVIDE

QUESTION: WHAT HAPPENS IF THE RESULT OF A DIVISION WILL NOT FIT INTO THE DESTINATION REGISTER (PROBABLY BECAUSE OF A DIVIDE BY ZERO)?

THE ADJUST INSTRUCTIONS)

DECIMAL ADJUSTMENTS:

DAA: DECIMAL ADJUST FOR ADD - ADD TWO BCD NUMBERS, ADJUST RESULT

EXAMPLE:

```
MOV    AL,26H    ; BCD 26
ADD    AL,27H    ; ADD BCD 27, RESULT IS 4DH
DAA              ; RESULT IS ADJUSTED TO BCD 53
```

* CARRY FLAG WILL INDICATE 100 (MAXIMUM SUM WOULD 99 + 99 = 198)
* ONLY WORKS FOLLOWING ADDITION OF TWO PACKED BCD DIGITS

THE ADJUST INSTRUCTIONS (CONT.)

DAS : DECIMAL ADJUST FOR SUBTRACT

∗ CARRY FLAG INDICATES 100 'BORROWED'

∗ ONLY WORKS FOLLOWING SUBTRACTION OF TWO PACKED BCD DIGITS

EXAMPLE :

```
    MOV     AL,6        ; AL = BCD 6
    SUB     AL,27H      ; SUBTRACT BCD 27, RESULT IS DFH
    DAS                 ; RESULT ADJUSTED TO 79, CARRY FLAG SET
```

* RESULT WAS 6 - 27 = -21. 100 WAS BORROWED FROM NEXT MOST SIGNIFICANT BYTE OF THE OPERAND (WHEN SUBTRACTING STRINGS OF BCD NUMBERS). CARRY INDICATES 100 WAS BORROWED, -21 + 100 = 79.

** CAN ADD/SUBTRACT BCD STRINGS USING SEVERAL ADD/SUBTRACT AND DAA/DAS INSTRUCTIONS CONNECTED BY CARRY FLAG.

THE ADJUST INSTRUCTIONS (CONT.)

ASCII ADJUSTMENTS :

AAA : ASCII ADJUST FOR ADD - ADJUST RESULT OF ADDING TWO UNPACKED DIGITS

AAS : ASCII ADJUST FOR SUBTRACT - AAA FOR SUBTRACTION

AAM : ASCII ADJUST FOR MULTIPLY - HAVING MULTIPLIED TWO DIGITS (RESULT IN HEX), SPLITS PRODUCT INTOTWO DECIMAL DIGITS IN AH, AL

AAD : ASCII ADJUST FOR DIVIDE - CONVERTS TWO UNPACKED BCD DIGITS INTO THEIR 8-BIT BINARY EQUIVALENT READY FOR A DIVIDE OPERATION

∗ ASCII OFFSET OF 30H IS LOST. YOU OR IT BACK IN TO AN ADJUSTED RESULT

∗ SEE ASM86 LANGUAGE REFERENCE MANUAL FOR DETAILS OF THESE INSTRUCTIONS

EXTERNAL SYNCHRONIZATION

HLT	HALT UNTIL INTERRUPT OR RESET
WAIT	WAIT FOR TEST PIN ACTIVE
ESC	ESCAPE TO EXTERNAL PROCESSOR (EG 8087)
LOCK	LOCK BUS FOR DURATION OF NEXT INSTRUCTION

NO OPERATION

NOP	NO OPERATION (XCHG AX,AX)

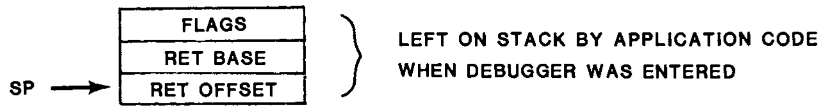
EXERCISE 3.1

1. WHAT IS THE PURPOSE OF THE ARGUMENT TO XLAT IN THE FOLLOWING CODE:

```
CODE    SEGMENT
        ASSUME   CS:CODE
TABLE   DB       1,3,5,7 ...
        .....
        LEA      BX,TABLE
        XLAT     TABLE
```

2. LIST THE TYPES OF UNCONDITIONAL CALLS (eg indirect FAR) AND HOW ASM86 RECOGNIZES WHICH TYPE IT IS TO ENCODE.

3. YOU ARE WRITING A DEBUGGER ROUTINE WHICH IS TO SINGLE STEP THROUGH YOUR APPLICATION CODE. ASSUME THAT YOU ENTERED THE DEBUGGER WITH A STACK FRAME AS SHOWN. USE POP, PUSH, IRET TO SINGLE STEP YOUR CODE FROM THE ADDRESS CURRENTLY IN ES:DI. DO THIS BY SETTING THE TRAP FLAG ON 'RETURN' TO THE APPLICATION CODE



WHERE TO FIND MORE INFORMATION

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)
CHAPTER 3 - ARCHITECTURE AND INSTRUCTIONS

AN INTRODUCTION TO ASM86

ASM86 LANGUAGE REFERENCE MANUAL

RELATED TOPICS ...

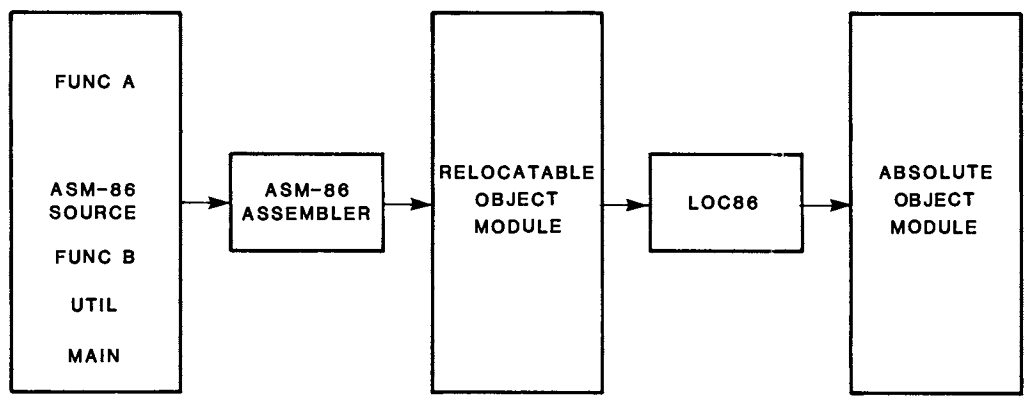
BIT CODINGS FOR INSTRUCTIONS ARE NOT COVERED IN THIS COURSE. FOR INFORMATION, SEE iAPX 86/88, 186/188 USER'S GUIDE. YOU WILL FIND THE ASM86 MACRO ASSEMBLER POCKET REFERENCE USEFUL (SEE FRONT OF IT FOR BIT ENCODING INFORMATION).

CHAPTER 4

MODULAR PROGRAM DEVELOPMENT

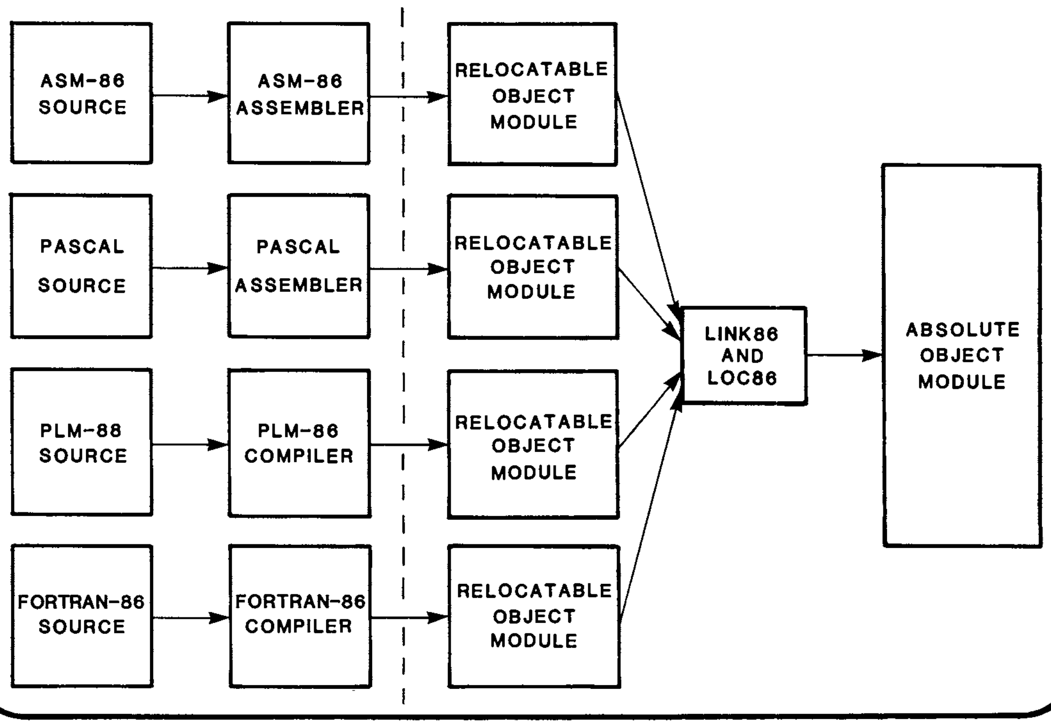
- INTRODUCTION TO LINKAGE AND LOCATION
- PROCEDURES
- LINKAGE DIRECTIVES
- REFERENCING EXTERNAL PROGRAM LABELS AND DATA ITEMS
- SEGMENT COMBINATION
- SEGMENT OPTIONS

SINGLE MODULE PROGRAM DEVELOPMENT



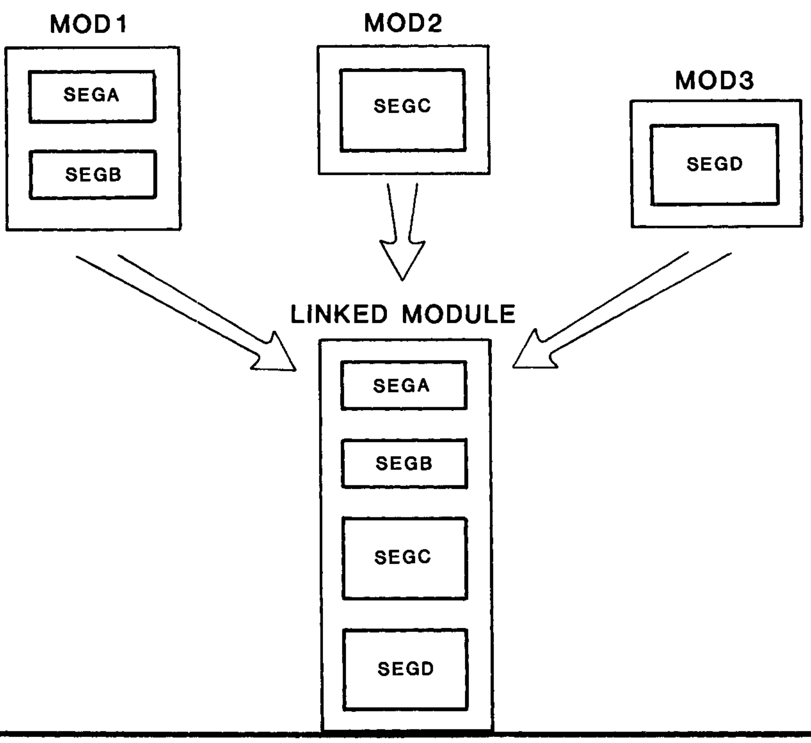
4-1

MODULAR PROGRAM DEVELOPMENT

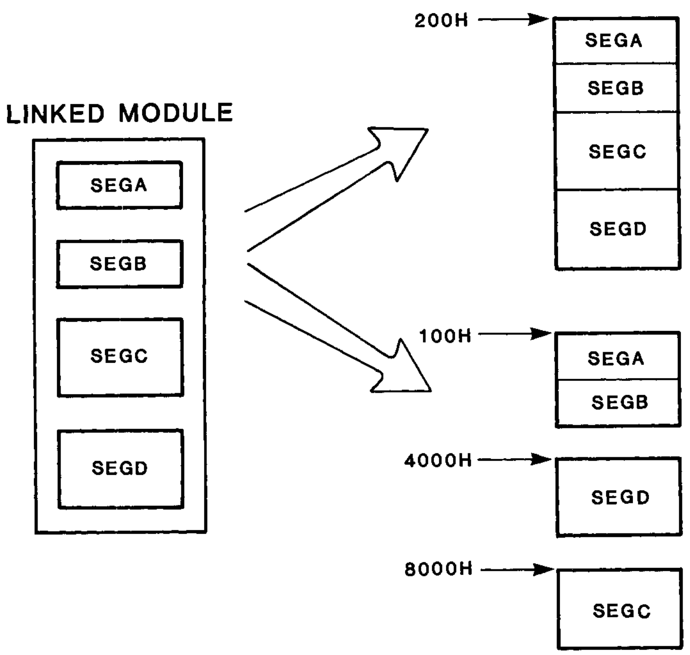


4-2

LINKAGE



LOCATION



- LOCATION IS PERFORMED ON A SEGMENT BASIS.

PROCEDURES

- MODULES OF A PROGRAM USUALLY LINKED BY PROCEDURE CALLS
- MULTI MODULE PROGRAM WILL USUALLY HAVE A 'MAIN MODULE'
- MAIN MODULE CONTAINS PROGRAM START ADDRESS
- OTHER MODULES CONTAIN PROCEDURES AND DATA DEFINITIONS
- IN ASM86 'END START' DEFINES MAIN MODULE (ALL OTHERS JUST HAVE END)
- LINKER WILL TRAP 'MORE THAN 1 MAIN MODULE'

PROCEDURE DEFINITION

```
CODE_1      SEGMENT
            ASSUME    CS:CODE_1

WALLY       PROC      FAR         ; default is NEAR

                                  ; insert useful
                                  ; code here

            RET

WALLY       ENDP

CODE_1      ENDS
```

EXERCISE 4.1 NEAR AND FAR PROCEDURES

TRUE OR FALSE?

* GIVING A PROCEDURE THE FAR ATTRIBUTE DOES THE FOLLOWING THINGS...

1. ENCODES A FAR RET INSTRUCTION
2. TAGS THE PROCEDURE AS FAR
3. BECAUSE OF 2, ALL CALLS TO THIS PROCEDURE WILL TAKE 3 BYTES

* CALLING A FAR PROCEDURE FROM THE SEGMENT IN WHICH IT WAS DEFINED PRODUCES A NEAR CALL
* IF IN IGNORANCE I NEAR CALL A PROCEDURE WHICH IS DEFINED IN ANOTHER MODULE AS FAR THE RET INSTRUCTION PRINTS AN ERROR MESSAGE

  'HELP – I CAN'T FIND A SEGMENT TO RETURN TO !'

. . . AND DON'T FORGET THE STACK!

```
STACK     SEGMENT
DW        100 DUP (?)
T_O_S     LABEL      WORD
STACK     ENDS
MAIN      SEGMENT
          ASSUME     CS:MAIN, SS:STACK
START:    MOV        AX,STACK
          MOV        SS,AX
          LEA        SP,T_O_S
          CALL       WALLY
          - - - - -
MAIN      ENDS
          END        START
```

INTER-MODULE REFERENCES

BY USING PUBLIC AND EXTRN DECLARATIVES WITH THE TWO MODULES LINK86 CAN RESOLVE EXTERNAL REFERENCES

```
        NAME      MODA           |            NAME       MODB
                                 |
        EXTRN     PROCA:FAR      |            PUBLIC     PROCA
SEGA    SEGMENT                  |   SEGB     SEGMENT
        ASSUME    CS:SEGA        |            ASSUME     CS:SEGB
          •                      |              •
          •                      |              •
        CALL      PROCA          |   PROCA    PROC       FAR
          •                      |              •
          •                      |              •
SEGA    ENDS                     |   PROCA    ENDP
        END                      |   SEGB     ENDS
                                 |            END
```

PUBLIC AND EXTERNAL DECLARATIVES

- PUBLIC MAKES A NAME AVAILABLE TO THE LINKER.
- EXTRN TELLS ASM86 TO LET THE LINKER RESOLVE THE SYMBOL.

EXAMPLES:

```
PUBLIC      XYZ, WP, ERS     ; VARIABLES AND PROCEDURES DEFINED
                               ELSEWHERE IN THIS MODULE

EXTRN       FOO: BYTE        ; VARIABLES AND PROCS NOT
```

ATTRIBUTES OF AN EXTERNAL REFERENCE:

NEAR, FAR	(EXTERNAL PROCEDURE)
BYTE, WORD, DWORD	(EXTERNAL VARIABLE)
ABS	(EXTERNAL CONSTANT)

REFERENCING EXTERNAL PROCEDURES AND PROGRAMS LABELS

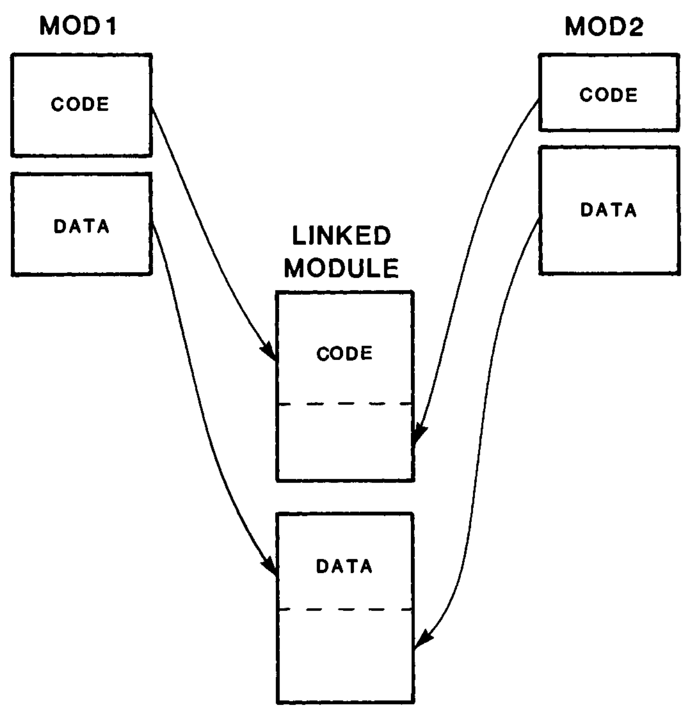
```
                NAME      MOD1
                PUBLIC    XYZ,ABC
                EXTRN     PROCA:FAR
CODE1           SEGMENT
                ASSUME    CS:CODE1

                CALL   PROCA

XYZ:            ____

ABC:            ____

CODE1           ENDS
                END
```

```
                NAME      MOD2
                PUBLIC    PROCA
                EXTRN     XYZ:FAR,ABC:FAR
CODE2           SEGMENT
                ASSUME    CS:CODE2
PROCA           PROC      FAR

PROCA           ENDP

                JMP       XYZ

                JMP       ABC

CODE2           ENDS
                END
```

WHY DID PROCA HAVE TO BE A FAR PROCEDURE?

4-11

COMBINING SEGMENTS



SEGMENT COMBINATION USING PUBLIC SEGMENTS

```
          NAME      MOD1                 |                  NAME      MOD2
          PUBLIC    XYZ                  |                  PUBLIC    PROCA
          EXTRN     PROCA:NEAR           |                  EXTRN     XYZ:NEAR
CODE      SEGMENT   PUBLIC               |        CODE      SEGMENT   PUBLIC
          ASSUME    CS:CODE              |                  ASSUME    CS:CODE
            ϟ                            |        PROCA     PROC      NEAR
          CALL      PROCA                |                    ϟ
            ϟ                            |                  RET
XYZ:      ______                         |        PROCA     ENDP
            ϟ                            |                    ϟ
CODE      ENDS                           |                  JMP       XYZ
          END                            |                    ϟ
                                         |        CODE      ENDS
                                         |                  END
```

REFERENCING EXTERNAL DATA ITEMS

```
                            PUBLIC    BUFFER
               DATA         SEGMENT
MODULE A:
               BUFFER       DB        100 DUP (?)
               DATA         ENDS
                            END
           ________________________________________
                            EXTRN     BUFFER:BYTE
               CODE         SEGMENT
                            ASSUME    CS:CODE,DS:______
MODULE B:                   MOV       AX, ______
                            MOV       DS, AX

                            MOV       BUFFER, AL
               CODE         ENDS
                            END
```

- HOW WOULD WE REFERENCE MULTIPLE EXTERNAL DATA ITEMS

REFERENCING MULTIPLE EXTERNAL DATA ITEMS

```
                NAME       MOD1

                PUBLIC     XYZ,ABC
DATA            SEGMENT    PUBLIC
XYZ             DB           ?
ABC             DW           ?
DATA            ENDS
                END
```

```
                NAME       MOD2

DATA            SEGMENT    PUBLIC              ;DUMMY SEGMENT
                EXTRN      XYZ:BYTE, ABC:WORD
DATA            ENDS

CODE            SEGMENT
                ASSUME     CS:CODE, DS:______

                MOV        AX,______
                MOV        DS, AX
                  ⋮
                MOV        AL, XYZ
                MOV        AH, 0
                MOV        ABC, AX
                  ⋮
CODE            ENDS
                END
```

ALIGNMENT TYPES

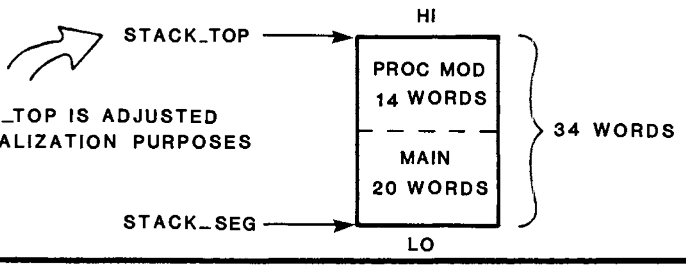
```
            NAME       ALIGNMENT_EXAMPLE

DATA        SEGMENT    WORD         PUBLIC
XYZ         DB         ?
            EVEN
ARRAY       DW         100 DUP(?)
DATA        ENDS

CODE        SEGMENT    BYTE         PUBLIC
            ASSUME     CS:CODE, DS:DATA

TABLE       DW         50, 30, 25, 62, 75
START:      MOV        AX, DATA
            MOV        DS, AX
              •
              •
              •
CODE        ENDS
            END
```

- EVEN DIRECTIVE ENSURES EVEN BOUNDARY ALIGNMENT WITHIN A SEGMENT.
- EVEN DIRECTIVE CAUSES ERROR IN BYTE ALIGNED SEGMENTS.

COMBINE TYPE : STACK

```
           NAME       MAIN
STACK_SEG  SEGMENT    STACK
           DW         20 DUP (?)
STACK_TOP  LABEL      WORD
STACK_SEG  ENDS

CODE       SEGMENT
           ASSUME     CS:CODE,SS:STACK
           MOV        AX,STACK
           MOV        DS,AX
           LEA        SP,STACK_TOP

CODE       ENDS
           END
```

```
           NAME       PROC_MOD
STACK_SEG  SEGMENT    STACK
           DW         14 DUP (?)
STACK_SEG  ENDS

           END
```



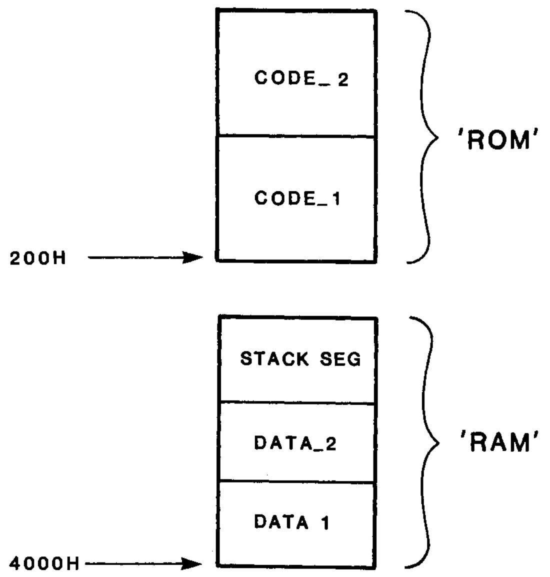
CLASS NAMES

- PERMITS CLASSIFICATION OF SEGMENTS UNDER A COMMON NAME THAT CAN BE USED AT LOCATE TIME.
- EXAMPLE

```
           NAME       MODA
DATA_1     SEGMENT    'RAM'

DATA_1     ENDS

STACK_SEG  SEGMENT    STACK 'RAM'

STACK_SEG  ENDS

CODE_1     SEGMENT    'ROM'
           ASSUME     CS:CODE_1,DS:DATA_1

CODE_1     ENDS
           END
```

```
           NAME       MODB
DATA_2     SEGMENT    'RAM'

DATA_2     ENDS

STACK_SEG  SEGMENT    STACK 'RAM'

STACK_SEG  ENDS

CODE_2     SEGMENT    'ROM'
           ASSUME     CS:CODE_2,DS:DATA_2

CODE_2     ENDS
           END
```

LOCATION BY CLASS NAME



WHERE TO FIND MORE INFORMATION

ASM86 LANGUAGE REFERENCE MANUAL

CHAPTER 5 – PROGRAM LINKAGE DIRECTIVES

AN INTRODUCTION TO ASM86

CHAPTER 4 – MODULAR PROGRAMMING

DAY 2 OBJECTIVES

BY THE TIME YOU FINISH TODAY YOU WILL:

- USE ASSEMBLER BUILT-IN FUNCTIONS
- USE ASSEMBLER DIRECTIVES
- WRITE ASM86 TEXT MACROS
- DEFINE COMPLEX DATA STRUCTURES IN ASM86
- USE APPROPRIATE ADDRESSING MODES FOR ACCESSING COMPLEX DATA STRUCTURES
- DEFINE AND USE SEGMENT GROUPS
- LEARN HOW TO USE LINK86 AND LOC86
- LINK ASSEMBLER PROGRAMS TO PL/M-86 PROGRAMS
- SEE HOW LINKING TO OTHER HIGH LEVEL LANGUAGES COMPARES TO LINKING WITH PL/M-86

CHAPTER 5
ASSEMBLER FEATURES

- ASSEMBLER DIRECTIVES
- ASSEMBLER BUILT-INS
- TEXT MACROS

THE EVEN DIRECTIVE

```
DATA      SEGMENT  WORD
BITE      DB  ?
          EVEN
BERNTH    DW  1234      ; GUARANTEED WORD ALIGNED
DATA      ENDS
```

- ENSURES WORD ALIGNMENT
- MIGHT WASTE A BYTE OF STORAGE
- DON'T MAKE DATA SEGMENTS BYTE-ALIGNED !!!!

THE LABEL DIRECTIVE

- ASSOCIATES A USER DEFINED SYMBOL NAME WITH THE CURRENT ASSEMBLER LOCATION COUNTER
- USEFUL FOR CODE AND DATA LABELS
- EXAMPLE

```
DATA         SEGMENT
LO_BYTE      LABEL       BYTE
WORD_VAR     DW          ?
DATA         ENDS
STACK        SEGMENT
             DW          20 DUP(?)
STACK_TOP    LABEL       WORD
STACK        ENDS
CODE         SEGMENT
             ASSUME      CS:CODE, SS:STACK, DS:DATA
             MOV         AX,STACK  ;INITIALIZE STACK
             MOV         SS,AX
             LEA         SP,STACK_TOP
             MOV         AX,DATA
             MOV         DS,AX
             MOV         AL, LO_BYTE
              .
              .
              .
CODE         ENDS
```

ASSEMBLER BUILT-INS

- ASSEMBLER HAS BUILT-IN OPERATORS TO AID IN PROGRAMMING

TYPE -	RETURNS TYPE OF DATA DEFINITION		
	DB	1	BYTE
	DW	2	BYTES
	DD	4	BYTES
LENGTH -	RETURNS NUMBER OF UNITS		
SIZE -	RETURNS NUMBER OF BYTES		

- EXAMPLE

```
             ARRAY    DW    100    DUP(?)
             MOV      BX,   0
             MOV      CX, LENGTH ARRAY
      NEXT:  ADD      ARRAY[BX],50
             ADD      BX, TYPE ARRAY
             LOOP     NEXT
             MOV      AX, SIZE ARRAY
             CALL     SAVE_ON_DISK
```

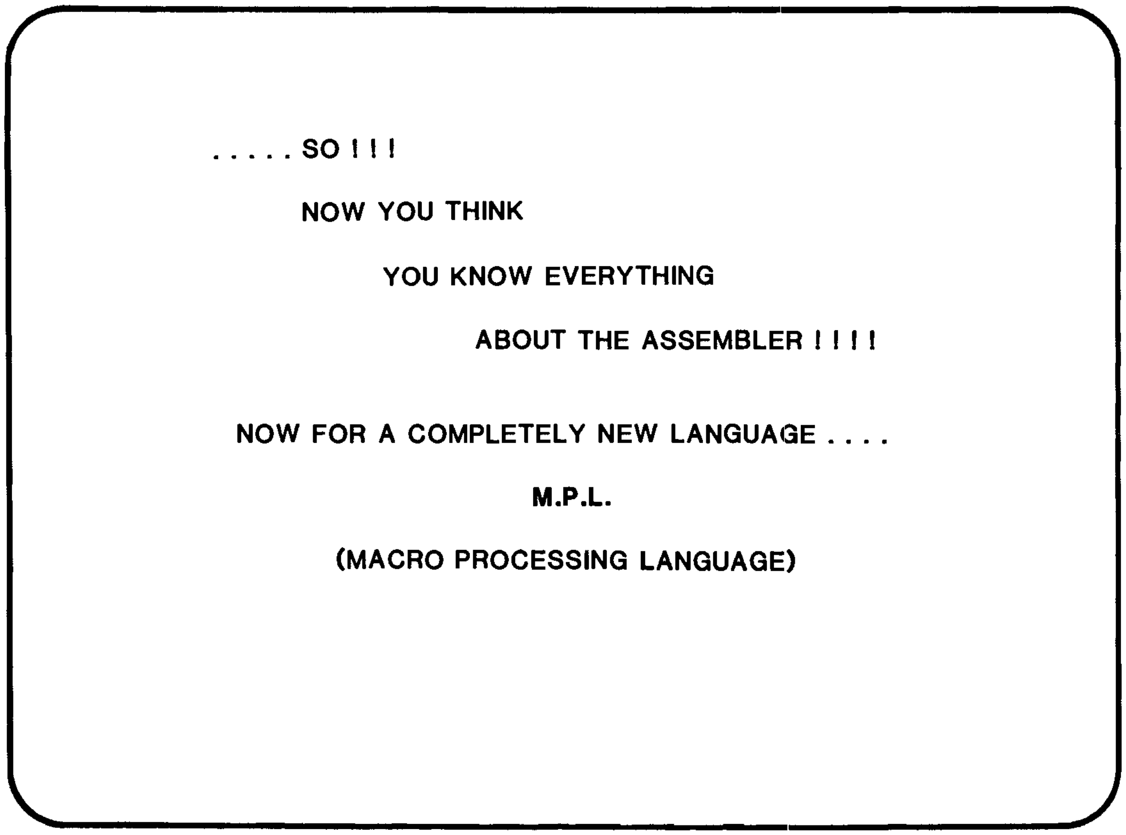
EQU STATEMENT

- THE EQU STATEMENT PROVIDES MORE MEANINGFUL NAMES FOR EXPRESSIONS

NUMBER	THREE	EQU	3
ADDRESS EXPRESSION	XYZ	EQU	ALPHA [SI]
REGISTER	COUNT	EQU	CX

- EXAMPLE

```
MOV     AL, THREE               ;SAME AS  AL,3
MOV     AX, XYZ                 ;SAME AS  AX, ALPHA [SI]
MOV     COUNT,LENGTH ARRAY      ;SAME AS  CX,LENGTH ARRAY
```



. . . . . SO ! ! !

NOW YOU THINK

YOU KNOW EVERYTHING

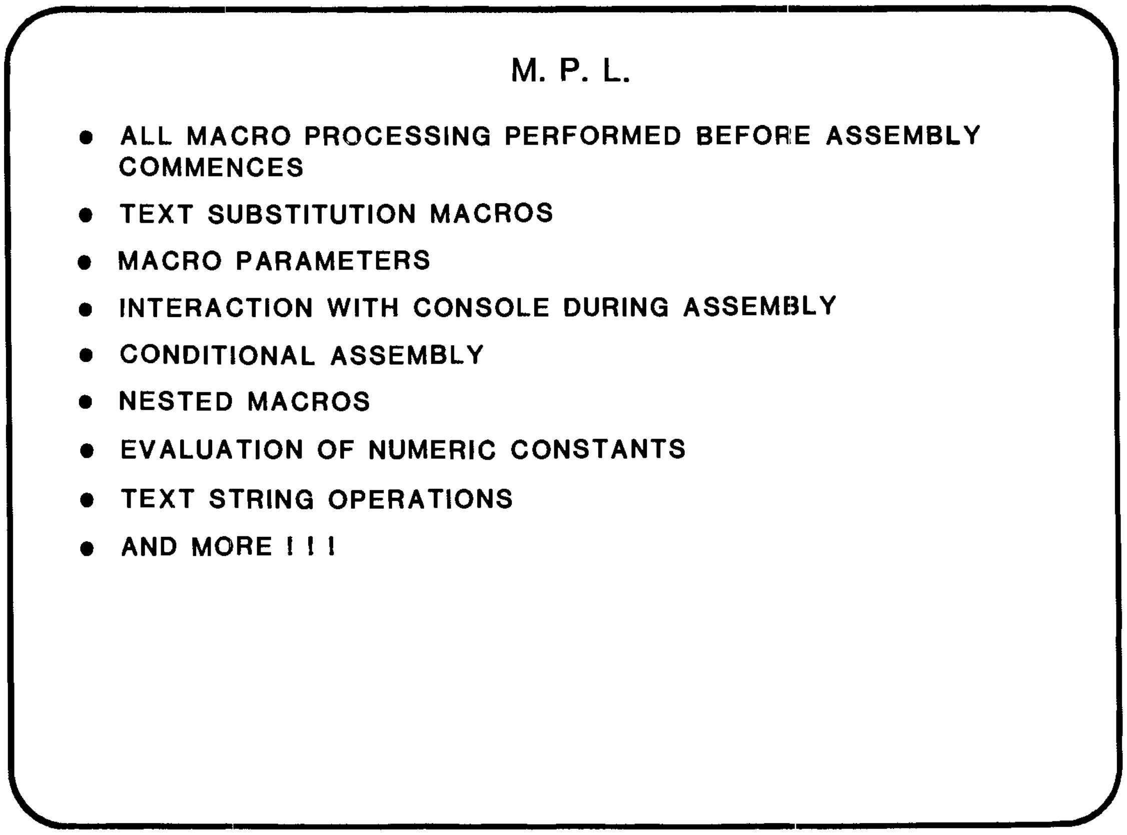
ABOUT THE ASSEMBLER ! ! ! !

NOW FOR A COMPLETELY NEW LANGUAGE . . . .

M.P.L.

(MACRO PROCESSING LANGUAGE)

5-5



M. P. L.

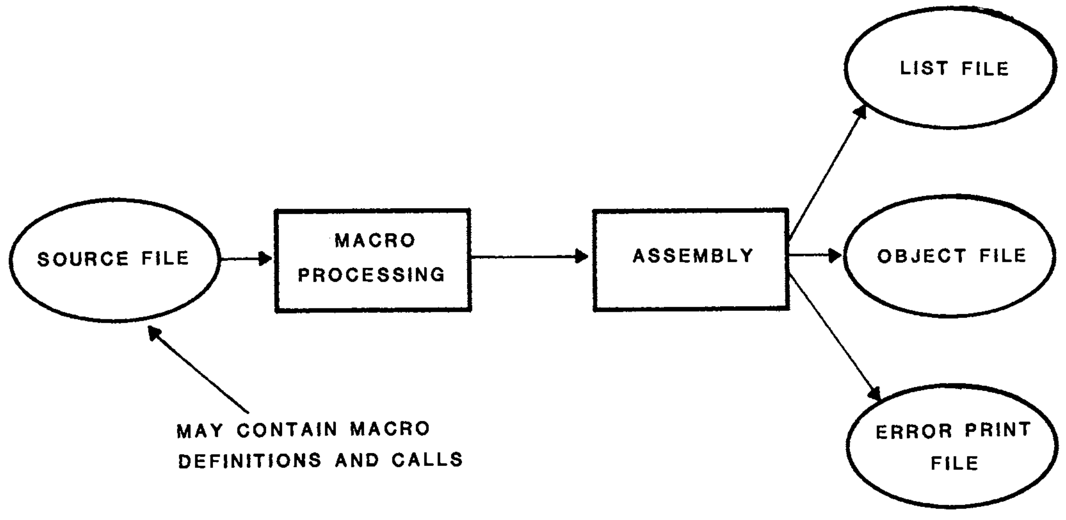
- ALL MACRO PROCESSING PERFORMED BEFORE ASSEMBLY COMMENCES
- TEXT SUBSTITUTION MACROS
- MACRO PARAMETERS
- INTERACTION WITH CONSOLE DURING ASSEMBLY
- CONDITIONAL ASSEMBLY
- NESTED MACROS
- EVALUATION OF NUMERIC CONSTANTS
- TEXT STRING OPERATIONS
- AND MORE ! ! !

ALL MACRO PROCESSING PERFORMED BEFORE ASSEMBLY COMMENCES

GENERAL FORMAT OF MACRO DEFINITION IS . . .

```
% DEFINE ( MACRO-NAME [(PARAMETER-LIST)] ) (MACRO-BODY)
```

MACRO PROCESSING AND ASSEMBLY



TEXT SUBSTITUTION MACROS

THIS MACRO WILL PASS A POINTER TO A PRINT PROCEDURE. THE POINTER IS COMPOSED OF AN OFFSET (PRE-LOADED INTO AX) AND THE CURRENT VALUE OF THE CODE SEGMENT REGISTER (THE MESSAGES ARE CONTAINED IN THE CODE SEGMENT). IT WOULD EASE THE INTERFACE BETWEEN ASM86 AND PL/M-86.

```
%*DEFINE(WRITE)(
PUSH        CS
PUSH        AX
CALL        PRINT
)
INVOCATION :
LEA       AX,MESSAGE_1  ; AX = OFFSET OF MESSAGE
%WRITE
```

MACRO PARAMETERS

THIS MACRO WILL PASS A POINTER TO A PRINT PROCEDURE. THE POINTER IS COMPOSED OF AN OFFSET (NOW PASSED AS A PARAMETER) AND THE CURRENT VALUE OF THE CODE SEGMENT REGISTER (THE MESSAGES ARE CONTAINED IN THE CODE SEGMENT).

```
%*DEFINE(WRITE(STRING_ADDRESS))(
LEA       AX,%STRING_ADDRESS
PUSH      CS
PUSH      AX
CALL      PRINT
)
INVOCATION :
          %WRITE(MESSAGE_1)   ; PRINT MESSAGE 1
```

* CAN HAVE MANY PARAMETERS IF REQUIRED

INTERACTION WITH CONSOLE DURING ASSEMBLY

THIS MACRO WILL ASK THE USER WHAT BAUD RATE IS REQUIRED FOR THIS SYSTEM

```
%*DEFINE       (WHAT_BAUD) (
               %OUT (ENTER REQUIRED BAUD RATE . . .)
BAUD-RATE DW      %IN
)

INVOCATION :
DATA_1         SEGMENT
%WHAT_BAUD
DATA_1         ENDS

-RUN ASM86 :F1:TEST.ASM
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0
ENTER REQUIRED BAUD RATE .... 9600<CR>
ASSEMBLY COMPLETE, NO ERRORS
-

YOUR LISTING WILL NOW SHOW    BAUD-RATE DW 9600
```

EVALUATION OF NUMERIC CONSTANTS

NOW USE THE EVALUATE COMMAND TO MAKE THIS A LITTLE MORE CLEVER. THE MACRO WILL NOW CALCULATE THE CORRECT NUMBER FOR YOUR COUNTER WHICH WILL PRODUCE THE REQUIRED BAUD RATE

```
%*DEFINE         (WHAT_BAUD) (
                 %OUT (ENTER REQUIRED BAUD RATE ...)
%SET(BAUD_RATE,  % IN)
COUNT_VAL EQU    %EVAL( (%BAUD_RATE / 27 ) - 13)
)
```

CONDITIONALS IN MACROS

AND NOW LET'S SEE IF WE CAN MAKE THIS THING FOOLPROOF !

```
%*DEFINE      (WHAT_BAUD) (
              %OUT (ENTER REQUIRED BAUD RATE ...)
              %SET (BAUD_RATE, % IN)
              %IF(%BAUD_RATE LT 100) THEN
              (%OUT (HOW SLOW DID YOU SAY ????))
              FI
              %IF( %BAUD_RATE GT 9600) THEN
              (%OUT (YOU MUST BE JOKING !!!))
              FI
COUNT_VAL EQU       %EVAL( (%BAUD_RATE /27 ) - 13)
                      )
```

5-13

INSTEAD OF COMMENTING ON A BAUD RATE, LET'S DISALLOW IT,TOO

```
%*DEFINE  (WHAT_BAUD) (
          %SET(REPLY,1)
          %WHILE(%REPLY) (
               %OUT (ENTER REQUIRED BAUD RATE ...)
               %SET (BAUD_RATE, %IN)
               %IF((%BAUD_RATE GE 110) AND
                    (%BAUD_RATE LE 9600))  THEN
                    (%SET(REPLY,0))
                    ELSE
                      (%OUT (VALID RANGE IS 110 TO 9600...))
                    FI

                        )
          COUNT_VAL EQU   %EVAL( (%BAUD_RATE / 27 ) - 13)
                    )
```

5-14

WHAT OTHER GOODIES DOES M.P.L. HAVE?

- DOCUMENTATION - CHAPTER 7 OF ASM86 LANGUAGE REFERENCE MANUAL ! ! !

CLASS EXERCISE 5.1

WRITE A MACRO THAT WILL MOVE A BYTE STRING FROM ONE LOCATION TO ANOTHER IN MEMORY. THE MACRO SHOULD ACCEPT AND USE THREE PARAMETERS. THEY ARE:

1. SOURCE
2. DESTINATION
3. COUNT

ASSUME THAT SOURCE AND DESTINATION ARE BOTH IN A SEGMENT CURRENTLY ADDRESSED BY DS. YOU MAY DESTROY ES AND CX.

WHERE TO FIND MORE INFORMATION

ASM86 LANGUAGE REFERENCE MANUAL
CHAPTER 4 - ACCESSING DATA--OPERANDS AND EXPRESSIONS
CHAPTER 7 - THE MACRO PROCESSING LANGUAGE

ASM86 MACRO ASSEMBLER OPERATING INSTRUCTIONS
CHAPTER 3 - ASSEMBLER CONTROLS

RELATED TOPICS:

THERE IS ANOTHER TYPE OF MACRO CALLED A CODEMACRO. THIS IS ALMOST ANOTHER LANGUAGE IN ITSELF. IT COULD BE USED TO RE-WRITE INTEL'S INSTRUCTION SET MNEMONICS OR ADD CUSTOM INSTRUCTIONS TO THE INSTRUCTION SET TO HANDLE (FOR INSTANCE) YOUR CUSTOM COPROCESSOR. DETAILS MAY BE FOUND IN THE ASM86 LANGUAGE REFERENCE MANUAL, APPENDIX A.

CHAPTER 6

COMPLEX DATA STRUCTURES

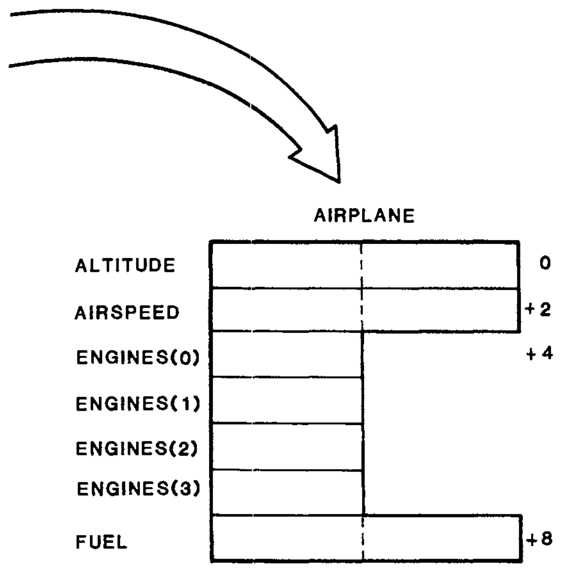
- STRUCTURES
- RECORDS

STRUCTURES

- A STRUCTURE IS A CONTIGUOUS COLLECTION OF DISSIMILAR BUT RELATED DATA ELEMENTS.
- THE STRUC DIRECTIVE ALLOWS YOU TO DEFINE A TEMPLATE THAT CAN BE USED TO FORMAT STORAGE ALLOCATION.

STRUCTURE EXAMPLE

```
AIRPLANE    STRUC
    ALTITUDE    DW    ?
    AIRSPEED    DW    ?
    ENGINES     DB    4    DUP(?)
    FUEL        DW    ?
AIRPLANE    ENDS
```



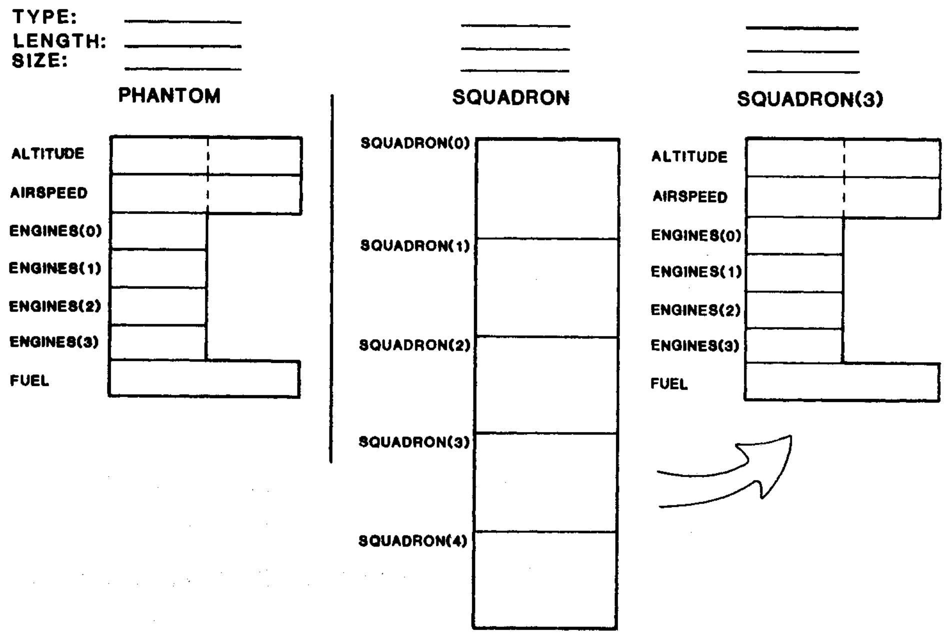
- THE STRUCTURE DIRECTIVE ONLY SETS UP A TEMPLATE. IT DOES NOT ALLOCATE ANY STORAGE.

DEFINING STORAGE

- USE THE STRUCTURE DEFINITION AS A NEW DATA TYPE
- EXAMPLE

```
DATA        SEGMENT

AIRPLANE    STRUC
   ALTITUDE       DW     ?
   AIRSPEED       DW     ?
   ENGINES        DB     4 DUP(?)
   FUEL           DW     ?
AIRPLANE    ENDS

PHANTOM     AIRPLANE     <>          ;ALLOCATES STORAGE FOR
                                     ;ONE STRUCTURE
SQUADRON    AIRPLANE     5 DUP(< >)  ;ALLOCATES STORAGE FOR
                                     ;AN ARRAY OF FIVE
                                     ;STRUCTURES.

DATA        ENDS
```

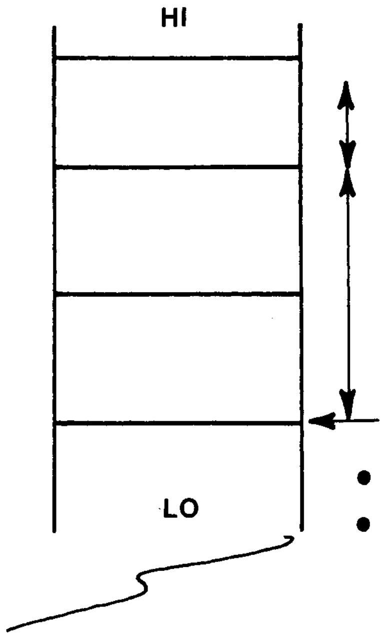
STORAGE ALLOCATION



REFERENCING STRUCTURE ELEMENTS

```
                 NAME       EXAMPLE
DATA          SEGMENT
AIRPLANE      STRUC
   ALTITUDE          DW       ?
   AIRSPEED          DW       ?
   ENGINES           DB       4 DUP(?)
   FUEL              DW       ?
AIRPLANE      ENDS

PHANTOM       AIRPLANE        < >
SQUADRON      AIRPLANE        5 DUP (< >)
DATA          ENDS

CODE          SEGMENT
              ASSUME          CS:CODE, DS:DATA
                •
                •
                •
              ;ACCESS PHANTOM'S ALTITUDE
              MOV             AX,PHANTOM.ALTITUDE     ; ACCESS PHANTOM'S ALTITUDE
              ; ACCESS THE AIRSPEED OF THE THIRD AIRPLANE  IN SQUADRON
              MOV             SQUADRON.AIRSPEED[2*TYPE SQUADRON], BX
              ; ACCESS THE THIRD ENGINE OF THE FIFTH AIRPLANE IN SQUADRON
              MOV             DH,SQUADRON.ENGINES[2*TYPE ENGINES][4*TYPE SQUADRON]
                •
                •
                •
CODE          ENDS
              END
```

REFERENCING STRUCTURE ELEMENTS (cont)



AIRSPEED – DISPLACEMENT INTO STRUCTURE

2 * TYPE SQUADRON = INDEX INTO ARRAY

SQUADRON – OFFSET OF ARRAY BASE

- ALL INDEXING ON BYTE LEVEL
- FINAL OFFSET IS SUM OF COMPONENTS

INITIALIZING A STRUCTURE

• TWO APPROACHES

```
AIRPLANE    STRUC
   ALTITUDE     DW       5000
   AIRSPEED     DW       600
   ENGINES      DB       0,0,0,0
   FUEL         DW       500
AIRPLANE    ENDS

JET         AIRPLANE     < >

;CAN ALSO CHANGE ANY OF
;THE INITIALIZED ELEMENTS

PROP        AIRPLANE     <250,,,,,200>
```

```
TEST         STRUC
   SAMPLES       DW      ?
   HIGH SCORE    DW      ?
   LOW SCORE     DW      ?
   MEAN          DW      ?
TEST         ENDS

MID_TERM     TEST        <50,100,43,72>
FINAL        TEST        <47,98,51,83>
```

CLASS EXERCISE 6.1

DEFINE AND ALLOCATE STORAGE FOR AN ARRAY OF 100 STRUCTURES. EACH OF THE STRUCTURES SHOULD CONTAIN THE FOLLOWING DATA:

LAST_NAME	– 10 BYTES
FIRST_NAME	– 10 BYTES
MI	– 1 BYTE
DIVISION	– 1 WORD
DEPT	– 1 WORD

WRITE A PROGRAM LOOP TO MAKE EACH EMPLOYEE'S DIVISION NUMBER EQUAL TO 12.

ADDRESSING VARIABLES

- ADDRESSING MODE MODEL

$$\text{OFFSET} = \begin{bmatrix}\text{VARIABLE}\\ \text{NAME}\end{bmatrix} + \begin{bmatrix}[\text{BX}]\\ [\text{BP}]\end{bmatrix} + \begin{bmatrix}[\text{SI}]\\ [\text{DI}]\end{bmatrix} + \begin{bmatrix}\text{DISPLACEMENT}\end{bmatrix}$$

- TAILORED FOR BASED ADDRESSING

$$\text{OFFSET} = \underbrace{\begin{bmatrix}[\text{BX}]\\ [\text{BP}]\end{bmatrix}}_{\text{BASE REG}} + \underbrace{\begin{bmatrix}[\text{SI}]\\ [\text{DI}]\end{bmatrix}}_{\text{INDEX REG}} + \begin{bmatrix}\text{DISPLACEMENT}\end{bmatrix}$$

BASED ADDRESSING MODES

ADDRESSING MODE	APPLICATION
[BASE REG]	BASED SCALARS
[BASE REG] + DISP	BASED STRUCTURES
[BASE REG] [INDEX REG]	BASED ARRAYS
[BASE REG] [INDEX REG] + DISP	BASED ARRAYS OF STRUCTURES

RECORDS

- BIT PATTERN USED TO FORMAT BYTES AND WORDS.
- CAN BE USED STRICTLY FOR FIELD REFERENCING (I.E. MASKING).
- CAN ALSO BE USED TO ALLOCATE STORAGE OF FORMATTED DATA.

USING RECORDS

- EXAMPLE

```
COORDINATE_FRAME    RECORD        X:8,Y:8
```

X	Y
00000000	00000000

- ALLOCATE STORAGE USING THE RECORD DEFINITION

```
COORD_0             COORDINATE_FRAME    < >
```

00000000	00000000

```
COORD_ABC           COORDINATE_FRAME    <4,3>
```

00000100	00000011

```
COORD_ARRAY         COORDINATE_FRAME    50 DUP(< >)    ;50 CONTIGUOUS COPIES OF
                                                         COORDINATE_FRAME
```

- LOAD A RECORD COPY INTO A CPU REGISTER

```
MOV                 CX,COORD_ABC

MOV                 BX,COORD_ARRAY [4*TYPE COORD_ARRAY]
```

USING RECORDS FOR FIELD REFERENCING ONLY

• EXAMPLE

```
STATUS_51 RECORD
&DSR:1,SYNDET:1,FE:1,OE:1,
&PE:1,TXE:1,RXRDY:1,TXRDY:1
```

DSR	SYNDET	FE	OE	PE	TXE	RXRDY	TXRDY
?	?	?	?	?	?	?	?

• MASK OUT IRRELEVENT BITS USING THE MASK OPERATOR

```
            MOV     DX,0F8H
WAIT_LOOP:  IN      AL,DX
            TEST    AL,MASK RXRDY
            JZ      WAIT_LOOP
```

MASK VALUE

$\boxed{00000010}$

USING RECORDS
(cont)

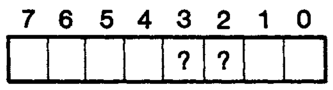
- USE RECORD TO SET UP SHIFT COUNT
- PL/M USES LS BIT TO TEST TRUE/FALSE

EXAMPLE:

```
8251_READY PROC
      IN      AL,OF8H     ; READ STATUS REGISTER
      MOV     CL,RXRDY
      SHR     AL,CL       ; PUT RXRDY IN LS BIT
      RET
8251_READY ENDP
```

CLASS EXERCISE 6.2

DEFINE A RECORD THAT WILL ALLOW YOU TO ISOLATE BITS 2 AND 3 OF A BYTE VALUE AS A SINGLE TWO BIT FIELD



2.) WRITE AN ASSEMBLY LANGUAGE INSTRUCTION USING THIS RECORD TO ISOLATE BITS 2 AND 3 OF THE AL REGISTER.

3.) WHAT IS THE TYPE OF THIS RECORD?

WHERE TO FIND MORE INFORMATION...

ASM86 LANGUAGE REFERENCE MANUAL

CHAPTER 3 - DEFINING AND INITIALIZING DATA

CHAPTER 7
GROUPS

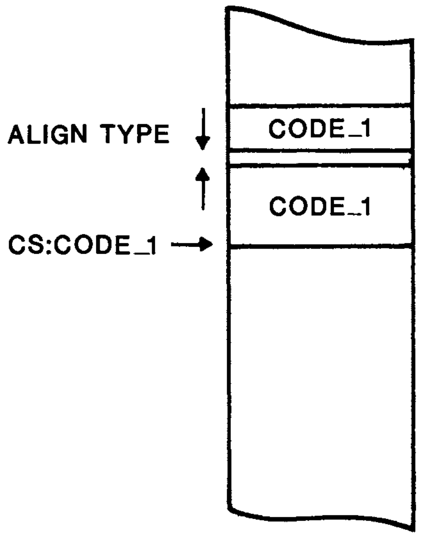
- WHAT THEY ARE ?
- HOW TO USE THEM ?

DEFINING A SEGMENT

```
NAME    SEGMENT   [ALIGN TYPE][COMBINE TYPE]['CLASSNAME']
           .
           .
           .
NAME    ENDS
```

7-1

COMBINING SEGMENTS : PUBLIC SEGMENTS



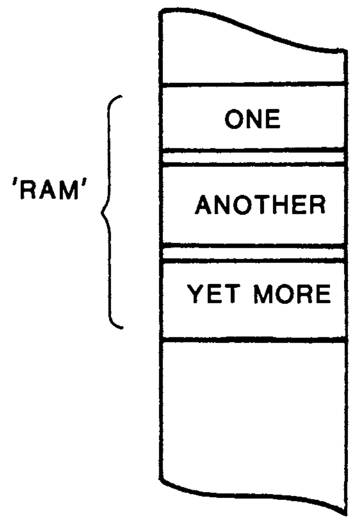
```
CODE_1      SEGMENT WORD PUBLIC
              .
              .
              .
CODE_1      ENDS
```

FEATURES:

- ONE SEGMENT NAME
- COMMON SEGMENT BASE ALLOWS NEAR CALLS/JUMPS, EASY DATA ACCESS
- SEGMENTS JOINED AT NEXT ADDRESS BOUNDARY SATISFYING ALIGN TYPE
- MAXIMUM TOTAL SIZE 64K

7-2

COMBINING SEGMENTS : CLASSNAMES

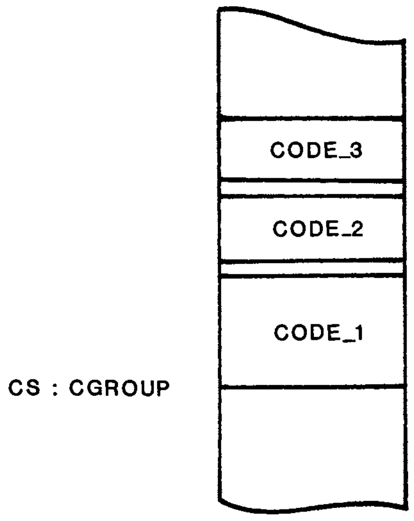


ANOTHER SEGMENT 'RAM'
ANOTHER ENDS

FEATURES:

- ALLOWS GATHERING OF SEGMENTS DESTINED FOR THE SAME AREA IN MEMORY
- EACH SEGMENT HAS SEPARATE NAME AND BASE
- MAXIMUM CLASS SIZE I M

COMBINING SEGMENTS : GROUPS



```
        NAME  MOD1
CGROUP  CODE_1
```

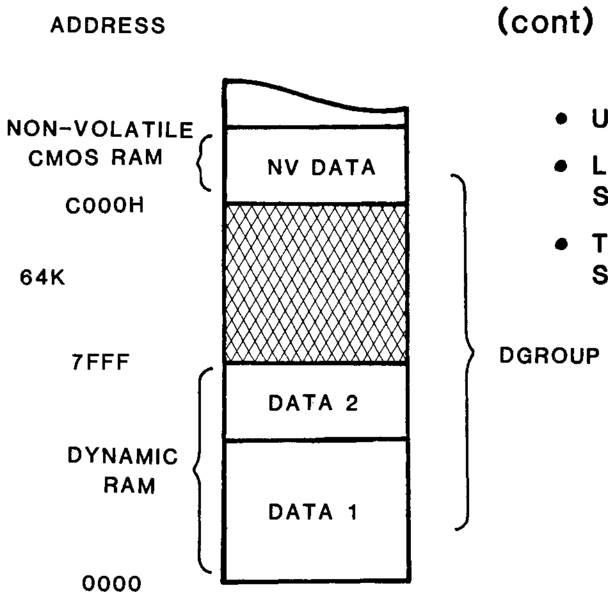
```
  NAME  MOD2
CGROUP CODE_2, CODE_3
```

FEATURES:

- COMMON GROUP BASE
- SEGMENTS HAVE DIFFERENT NAMES
- MAXIMUM TOTAL SIZE 64K

. . . SO WHY NOT USE PUBLIC SEGMENTS?

COMBINING SEGMENTS : GROUPS (cont)



- UNIQUE SEGMENT NAMES
- LOCATER CAN SEPARATE SEGMENTS TO SUIT MEMORY MAP
- TYPICALLY USED IN SMALL SYSTEMS

USING THE GROUP DIRECTIVE

```
            NAME        GROUP_EXAMPLE

CGROUP      GROUP       CODE_1,CODE_2,CODE_3

CODE_1      SEGMENT
            ASSUME      CS:CGROUP
            •
            •
CODE_1      ENDS

CODE_2      SEGMENT
            ASSUME      CS:CGROUP
            •
            •
CODE_2      ENDS
CODE_3      SEGMENT
            ASSUME      CS:CGROUP
            •
            •
CODE_3      ENDS
            END
```

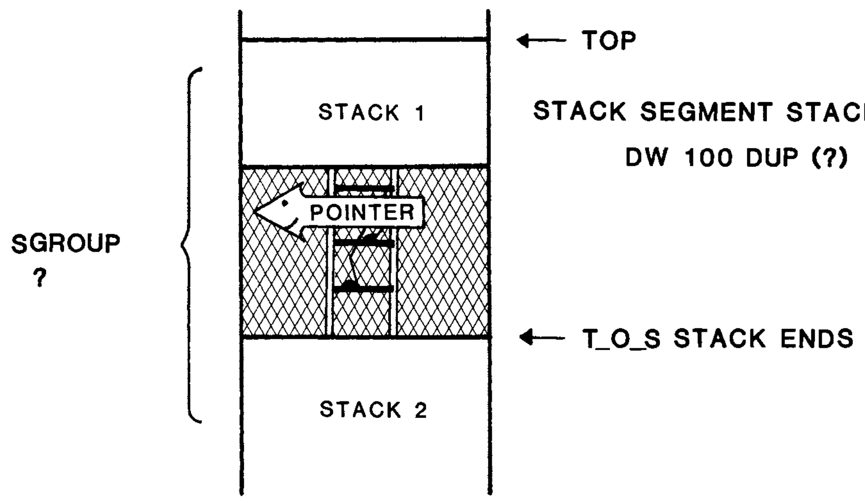
NOW THAT WE HAVE A GROUP:

```
CGROUP      GROUP      CODE_1, CODE_2, CODE_3
DGROUP      GROUP      DATA_1, DATA_2, NV_DATA

CODE_3      SEGMENT BYTE
            ASSUME  CS:CGROUP, DS:DGROUP, SS:STACK
CONSTANT    DW  1Ø
            MOV     AX, DGROUP
            MOV     DS, AX
            LEA     BX,CONSTANT           ; OK !
            MOV     BX,OFFSET CONSTANT    ; OFFSET IS FROM CODE_3
            MOV     BX,OFFSET CGROUP:CONSTANT
            MOV     AL, [BX]
```

. . . WHICH OF THOSE LAST TWO OFFSETS IS LIKELY TO GENERATE THE CORRECT ADDRESS?

<u>BEWARE ! !</u>

WOULD YOU USE A GROUP FOR BUILDING ONE STACK?



STACK SEGMENT STACK 'STACK'
DW 100 DUP (?)

THE STACK IS ALWAYS A SINGLE CONTIGUOUS SEGMENT WITH COMBINE TYPE 'STACK'

WHERE TO FIND MORE INFORMATION...

ASM86 LANGUAGE REFERENCE MANUAL

CHAPTER 2 - SEGMENTATION

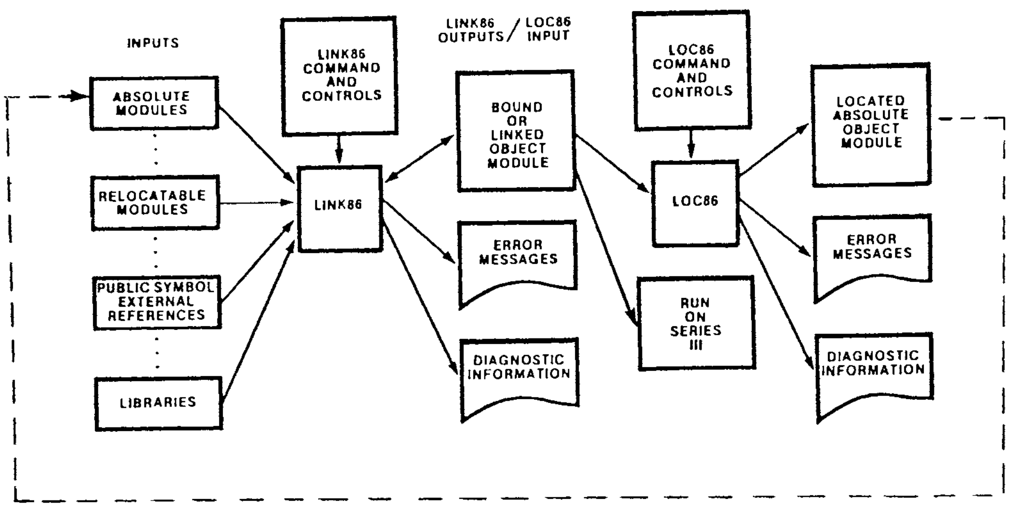
AN INTRODUCTION TO ASM86

CHAPTER 4 - MODULAR PROGRAMMING

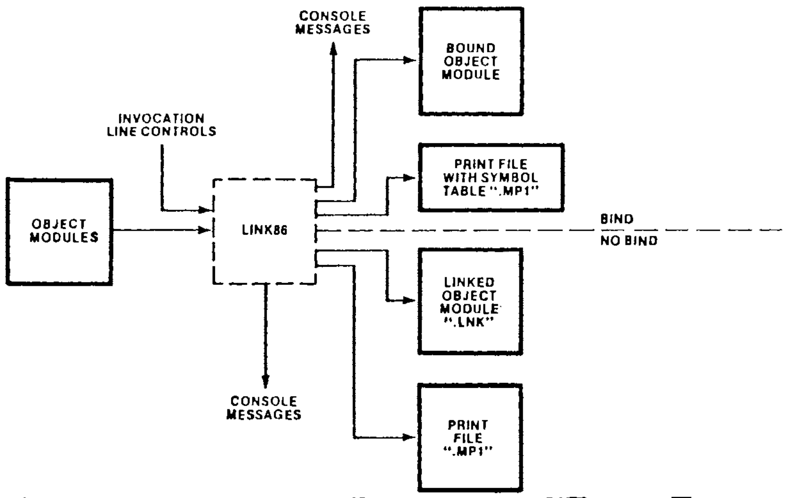
CHAPTER 8

LINK86 AND LOC86

DEVELOPMENT CYCLE WITH LINK86 AND LOC86


INPUTS
ABSOLUTE MODULES
RELOCATABLE MODULES
PUBLIC SYMBOL EXTERNAL REFERENCES
LIBRARIES
LINK86 COMMAND AND CONTROLS
LINK86
LINK86 OUTPUTS / LOC86 INPUT
BOUND OR LINKED OBJECT MODULE
ERROR MESSAGES
DIAGNOSTIC INFORMATION
LOC86 COMMAND AND CONTROLS
LOC86
RUN ON SERIES III
LOCATED ABSOLUTE OBJECT MODULE
ERROR MESSAGES
DIAGNOSTIC INFORMATION

LINK86 COMMAND



[:Fn:] LINK86 input_list [TO output_file][controls]

- OUTPUT FILE DEFAULTS TO 1ST INPUT FILE WITH .LNK EXTENSION.
- MAP FILE DEFAULTS TO OUTPUT FILE WITH .MP1 EXTENSION.

LINK86 CONTROLS

LINK86 input list [NAME(mod-name)]

[MAP*/NOMAP]

[SYMBOLS*/NOSYMBOLS]

[LINES*/NOLINES]

[PRINT (file-name)*/NOPRINT]

[SYMBOLCOLUMNS (1/2*/3/4)]

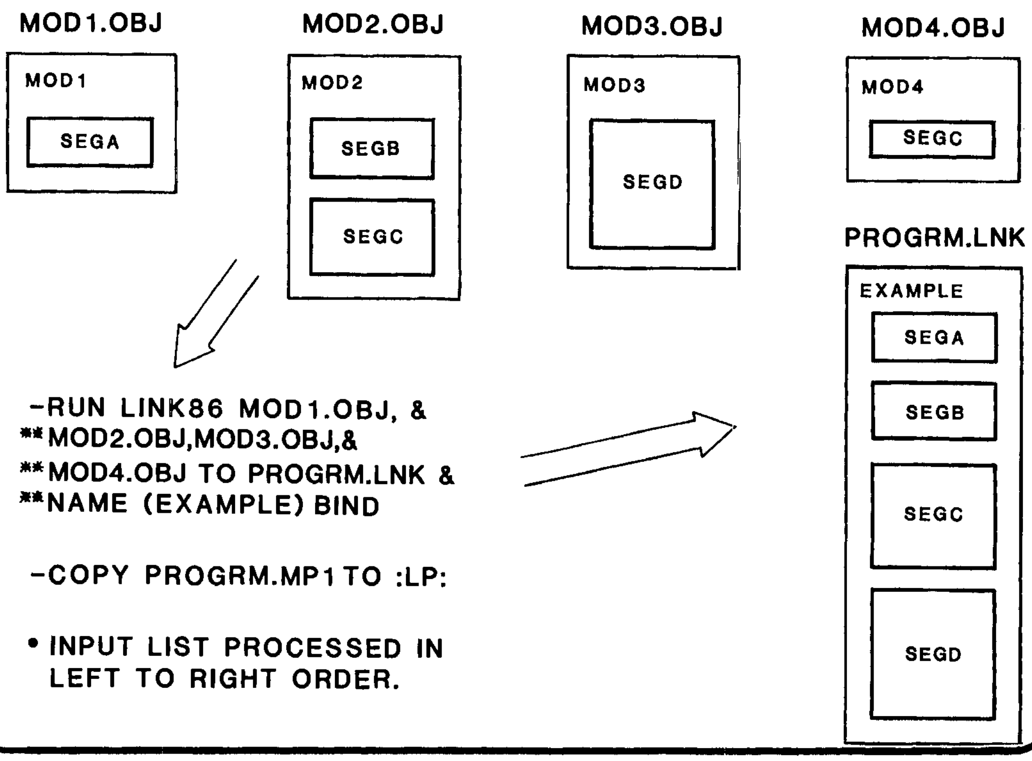
[TYPE*/NOTYPE]

[PURGE/NOPURGE*]

-DEFAULT [BIND/NOBIND]

NOTES:
1) OTHER CONTROLS ARE AVAILABLE.
2) THE CONTROLS CAN BE ABBREVIATED.
3) SEE THE iAPX 86,88 FAMILY UTILITIES USER'S GUIDE CHAPTER 2 FOR DETAILS

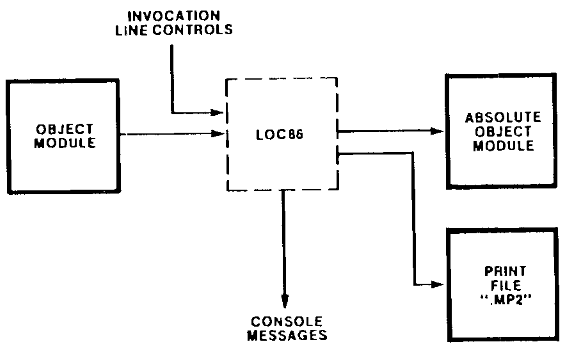
HOW MODULES ARE LINKED



```
-RUN LINK86 MOD1.OBJ, &
**MOD2.OBJ,MOD3.OBJ,&
**MOD4.OBJ TO PROGRM.LNK &
**NAME (EXAMPLE) BIND

-COPY PROGRM.MP1 TO :LP:
```

• INPUT LIST PROCESSED IN LEFT TO RIGHT ORDER.

LOC86 COMMAND



[:Fn:] LOC86 input_file [TO output_file] [controls]

- OUTPUT FILE DEFAULTS TO INPUT FILE MINUS THE EXTENSION.
- MAP FILE DEFAULTS TO OUTPUT FILE WITH .MP2 EXTENSION.

LOC86 CONTROLS

LOC86 input file [NAME(mod-name)]
[MAP*/NOMAP]
[SYMBOLS*/NOSYMBOLS]
[LINES*/NOLINES]
[PUBLICS*/NOPUBLICS]
[PRINT (file-name)*/NOPRINT]
[SYMBOLCOLUMNS (1/2*/3/4)]
[PURGE/NOPURGE*]

* -DEFAULT

NOTES: 1) OTHER CONTROLS ARE AVAILABLE.
2) THE CONTROLS CAN BE ABBREVIATED.
3) SEE THE iAPX 86,88 FAMILY UTILITIES USER'S GUIDE CHAPTER 3 FOR DETAILS

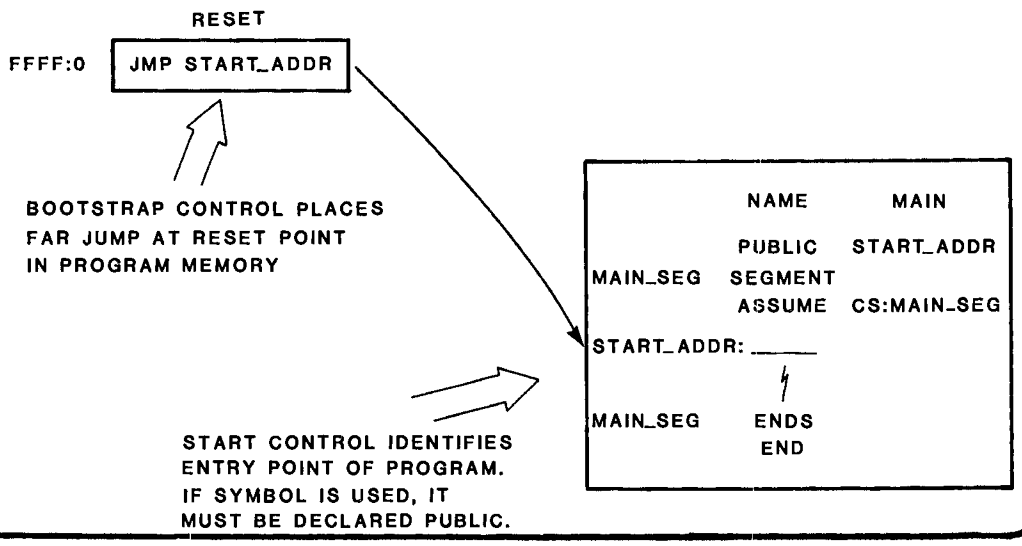
BOOTSTRAP CONTROL

[B]OOTSTRA[P]

[S]TART (symbol/segment,offset)]

EXAMPLE

```
-RUN LOC86 MAIN.LNK BOOTSTRAP START (START_ADDR)
```



```
                 RESET
FFFF:0     JMP START_ADDR

                                             NAME      MAIN

                                             PUBLIC    START_ADDR
                                 MAIN_SEG    SEGMENT
                                             ASSUME    CS:MAIN_SEG
                                 START_ADDR: ______

                                 MAIN_SEG    ENDS
                                             END
```

BOOTSTRAP CONTROL PLACES
FAR JUMP AT RESET POINT
IN PROGRAM MEMORY

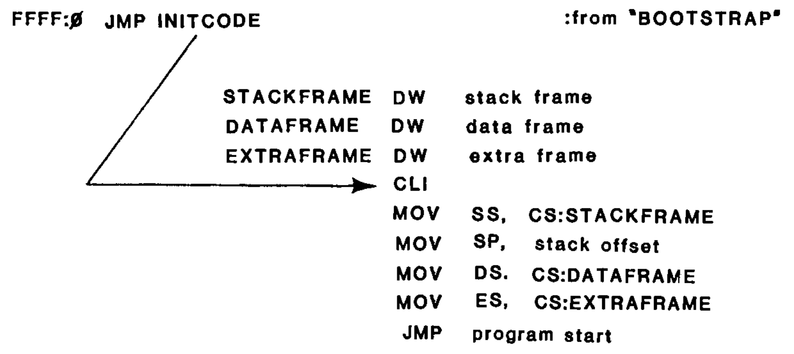
START CONTROL IDENTIFIES
ENTRY POINT OF PROGRAM.
IF SYMBOL IS USED, IT
MUST BE DECLARED PUBLIC.

INITCODE CONTROL

INITCODE (ADDRESS)

EXAMPLE

```
-RUN LOC86 MAIN.LNK BOOTSTRAP INITCODE (F0000H)
```



```
FFFF:0  JMP INITCODE                          :from 'BOOTSTRAP'

               STACKFRAME  DW    stack frame
               DATAFRAME   DW    data frame
               EXTRAFRAME  DW    extra frame
                           CLI
                           MOV   SS, CS:STACKFRAME
                           MOV   SP, stack offset
                           MOV   DS. CS:DATAFRAME
                           MOV   ES, CS:EXTRAFRAME
                           JMP   program start
```

SEGMENT INITIALIZATION RECORD:

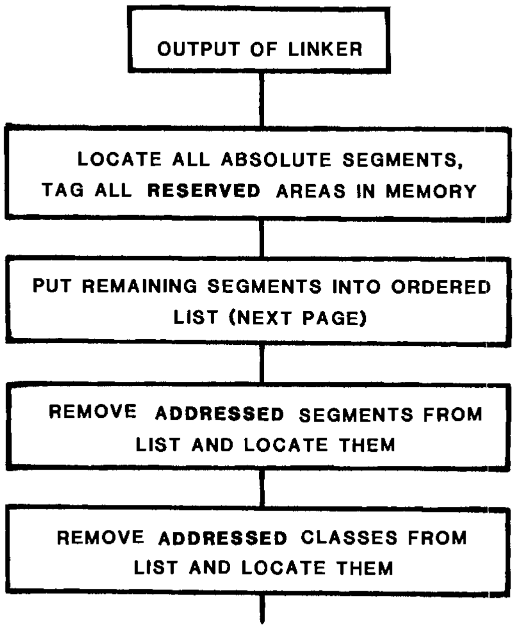
1. FROM END START,SS:STACKFRAME,DS:DATAFRAME,ES:EXTRAFRAME,SP:T_O_S
2. AUTOMATICALLY PRODUCED BY COMPILERS FOR MAIN MODULES

LOCATE CONTROLS

```
ADDRESSES      ( | SEGMENTS ( segname ( addr ) ,... )  | )
               ( | CLASSES ( classname ( addr ) ,... ) | )
               ( |*GROUPS ( groupname ( addr ) ,... )  | )

ORDER          ( | SEGMENTS ( segname ,... )   | )
               ( | CLASSES ( classname ,... )  | )
SEGSIZE        ( SEGNAME ( |±| VALUE ) ,... )
RESERVE        ( addr TO addr ,... )
```

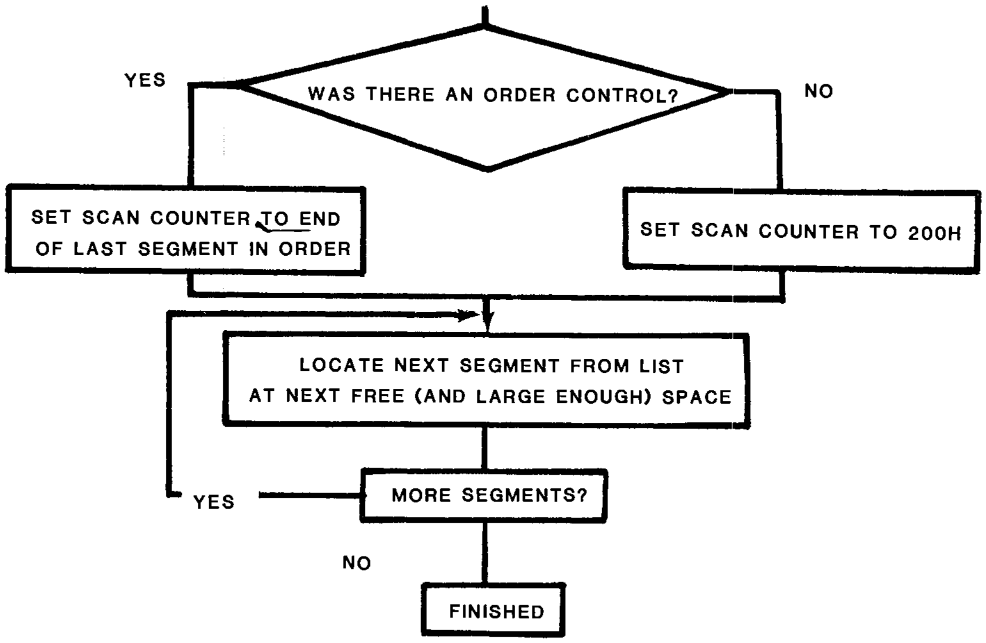
* "GROUPS" WILL NOT LOCATE A GROUP. IT WILL ASSIGN A GROUP BASE ADDRESS. ALL SEGMENTS IN GROUP MUST ALREADY BE WITHIN 64K (TYPICALLY FROM LOCATION BY CLASS).

LOCATE SEQUENCE


OUTPUT OF LINKER
LOCATE ALL ABSOLUTE SEGMENTS, TAG ALL **RESERVED** AREAS IN MEMORY
PUT REMAINING SEGMENTS INTO ORDERED LIST (NEXT PAGE)
REMOVE **ADDRESSED** SEGMENTS FROM LIST AND LOCATE THEM
REMOVE **ADDRESSED** CLASSES FROM LIST AND LOCATE THEM

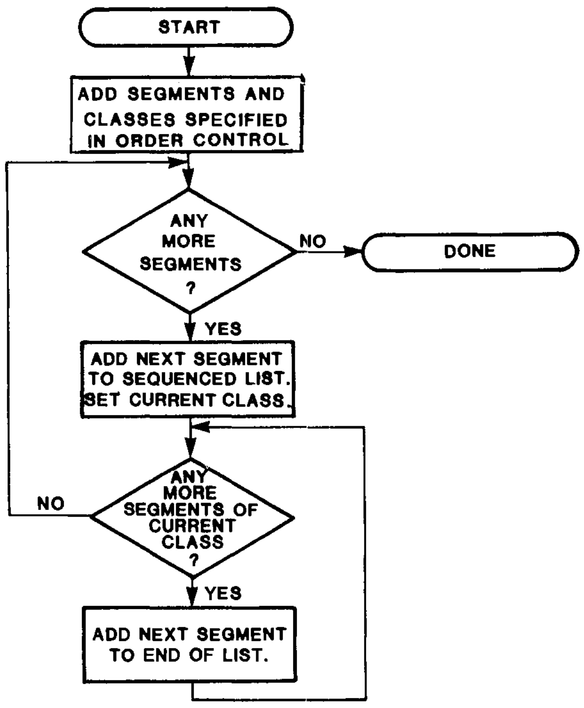
...CONTINUED

LOCATE SEQUENCE (cont)


WAS THERE AN ORDER CONTROL?
YES
NO
SET SCAN COUNTER TO END OF LAST SEGMENT IN ORDER
SET SCAN COUNTER TO 200H
LOCATE NEXT SEGMENT FROM LIST AT NEXT FREE (AND LARGE ENOUGH) SPACE
MORE SEGMENTS?
YES
NO
FINISHED

THE ORDERED LIST

- USED TO LOCATE SEGMENTS NOT ASSIGNED TO ADDRESSES OR ORDER CONTROLS.


START
ADD SEGMENTS AND CLASSES SPECIFIED IN ORDER CONTROL
ANY MORE SEGMENTS ?
NO
DONE
YES
ADD NEXT SEGMENT TO SEQUENCED LIST. SET CURRENT CLASS.
ANY MORE SEGMENTS OF CURRENT CLASS ?
NO
YES
ADD NEXT SEGMENT TO END OF LIST.

EXAMPLE: AN ORDERED LIST

```
RUN LOC86 EXAMPL.LNK ORDER(SEGMENTS(D),CLASSES(X2))
```

SEGMENT NAME	CLASS NAME
A	X1
B	X2
C	X2
D	X1
E	X3
F	X1
G	X2
H	X2
I	X1

$\longrightarrow$

SEGMENT	CLASS

... CONTINUED

EXAMPLE (CON'T.)

```
... ADDRESSES (SEGMENTS (D(100H),H(8000H)), CLASSES(X2(15ØØH)))
```

ORDERED LIST:

SEGMENT NAME	CLASS NAME	
D	X1	← 1ØØH
B	X2	
C	X2	← 15ØØH
G	X2	
H	X2	← 8ØØØH
A	X1	
F	X1	
I	X1	
E	X3	

$\Rightarrow$

	SEGMENT	CLASS
1ØØH →		
15ØØH →		
8ØØØH →		

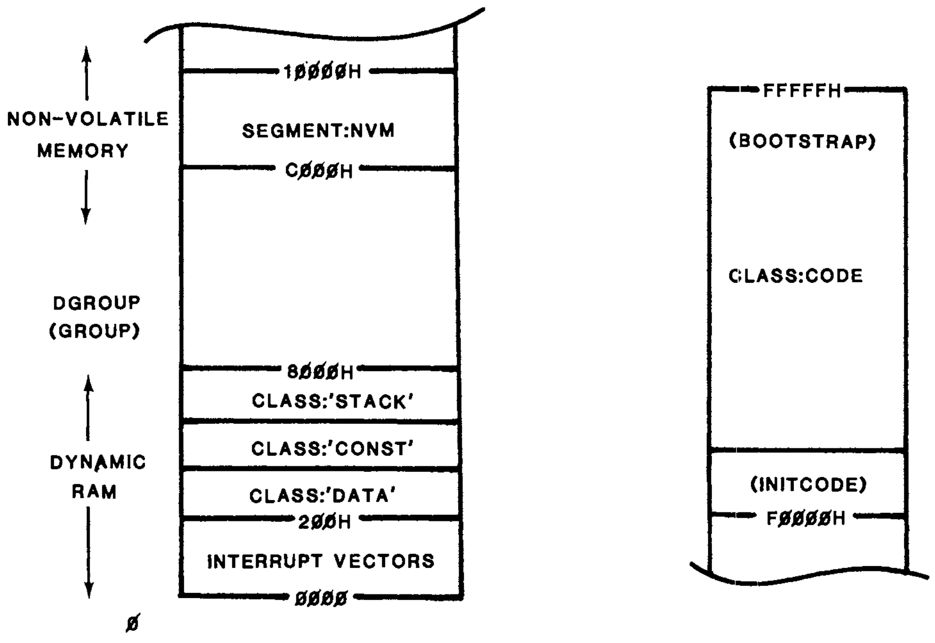
CLASS EXERCISE: USE OF LINK AND LOCATE

YOU ARE REQUIRED TO WRITE THE CORRECT LINK AND LOCATE CONTROLS TO LOCATE YOUR FINISHED PROGRAM AS DEMANDED BY THE ADDRESS MAP OF YOUR HARDWARE. THE REQUIREMENTS ARE ILLUSTRATED ON THE NEXT PAGE.
THE THREE INPUT MODULES ARE..

1.	PROG.OBJ	WRITTEN IN PL/M. IT DEFINES THE USE OF DGROUP
2.	PROCS.OBJ	WRITTEN IN ASM86. IT DEFINES THE SEGMENT NVM
3.	SMALL.LIB	A SUPPORT LIBRARY FOR THE SMALL MODEL OF PL/M

THE CLASSES IN DGROUP SHOULD APPEAR IN THE ORDER SHOWN, WITH THE FIRST CLASS STARTING AT ADDRESS 200H. NOTE THAT IF THE LOCATER TRIES TO LOCATE A CLASS WHERE A SEGMENT IS ALREADY LOCATED, IT WILL LOCATE THE CLASS AT THE NEXT AVAILABLE LOCATION (THIS SHOULD HELP WITH INITCODE AND THE CLASS 'CODE').

CLASS EXERCISE: ADDRESS MAP



WHERE TO FIND MORE INFORMATION...

iAPX 86/88 FAMILY UTILITIES USER'S GUIDE

CHAPTER 9

LINKING ASM86 WITH PL/M 86

- PL/M PROCEDURE DECLARATIONS
- PARAMETER PASSING
- COMPATIBLE DATA TYPES
- COMPILATIONS MODELS
- CONVENTIONS FOR MEMORY ALLOCATION
- CONVENTIONS FOR PROCEDURE AND LABEL DEFINITIONS
- CONVENTIONS FOR DATA DEFINITIONS

LINKING ASSEMBLY LANGUAGE MODULES WITH PL/M MODULES

- A HIGH LEVEL LANGUAGE (HLL) COMPILER USES A STANDARD SET OF RULES AND CONVENTIONS IN DEFINING CODE AND DATA.
- AN ASSEMBLY LANGUAGE MODULE TO BE LINKED TO A HLL MODULE MUST BE DESIGNED SUCH THAT IT SUPPORTS THESE RULES AND CONVENTIONS.
- GENERALLY, THE LINKAGE OF ASSEMBLY LANGUAGE AND A HIGH LEVEL LANGUAGE IS IMPLEMENTED ON A PROCEDURE BASIS.

PL/M PROCEDURE DECLARATION

```
proc_name:   PROCEDURE   [(parm1,parm2,...)]   [type];
      [DECLARE parm1...]
                                                  BYTE
                                                  WORD
      (PROCEDURE BODY)                            INTEGER
                                                  POINTER
        [RETURN value]                            REAL

END proc_name;
```

- A PL/M PROCEDURE CAN ACCEPT AS MANY INPUT PARAMETERS AS REQUIRED.
- A PL/M PROCEDURE CAN ALSO RETURN A SINGLE ITEM OF THE TYPE DEFINED IN THE PROCEDURE DECLARATION.

UNTYPED PL/M PROCEDURES

- DEFINITION

```
CLEAR_PORT: PROCEDURE (PORT);
        DECLARE PORT WORD;
        OUTPUT (PORT)=0;
END CLEAR_PORT;
```

- INVOCATION

```
CALL CLEAR_PORT (20);
```

TYPED PL/M PROCEDURES

- DEFINITION

```
ADD: PROCEDURE (PARM1,PARM2) WORD;
      DECLARE  PARM1  BYTE,
               PARM2  BYTE;
      RETURN   PARM1+PARM2;
END ADD;
```

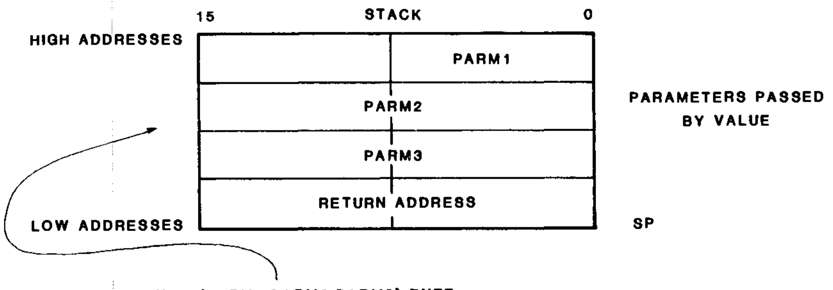
- INVOCATION

```
SUM=ADD (XYZ,ABC);
```

COMPATIBLE DATA TYPES

PL/M DATA TYPE	ASSEMBLY LANGUAGE DATA TYPE
BYTE	DB
WORD	DW
INTEGER	DW
POINTER	DW or DD
REAL	DD
STRUCTURE	STRUC

PL/M PARAMETER PASSING



```
PROC1: PROCEDURE (PARM1,PARM2,PARM3) BYTE;
          DECLARE  PARM1   BYTE,
                   PARM2   WORD,
                   PARM3   WORD,
                   RESULT  BYTE;

          RETURN   RESULT;
END PROC1;
```

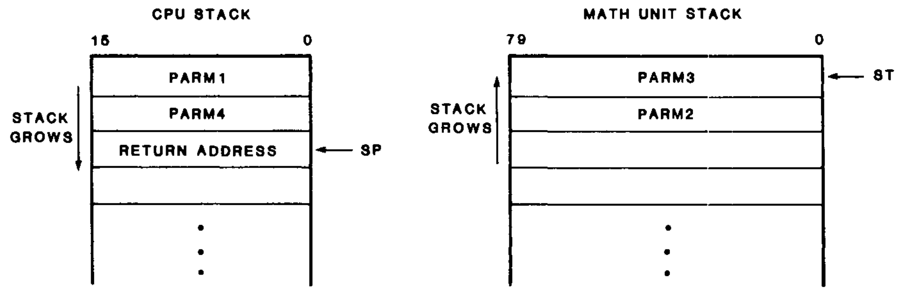
RETURNED VALUE

BYTE - AL
WORD - AX
INTEGER - AX
POINTER (SMALL) - BX OR ES:BX
POINTER (COMPACT, MEDIUM,LARGE) - ES:BX
REAL - TOP OF REAL MATH UNIT STACK

PASSING REAL PARAMETERS

- THE FIRST SEVEN REAL PARAMETERS ARE PASSED ON THE MATH UNIT STACK. ANY REMAINING ONES ARE PASSED ON THE CPU STACK.

```
PROC2:  PROCEDURE (PARM1,PARM2,PARM3, PARM4);
        DECLARE  PARM1    BYTE,
                 PARM2    REAL,
                 PARM3    REAL,
                 PARM4    INTEGER;


END PROC2;
```



WHICH REGISTERS CAN A PROCEDURE MODIFY?

REGISTER	MUST PRESERVE	USAGE
AX	NO	Return BYTE (AL), WORD and INTEGER values
BX	NO	Return POINTER values
CX	NO	—
DX	NO	—
SP	YES *	Stack pointer
BP	YES	Stack marker
SI	NO	—
DI	NO	—
FLAGS	NO	—
CS	YES	Caller's code segment
DS	YES	Caller's data segment
SS	YES	Caller's stack segment
ES	NO	Return POINTER values

* SP MUST BE ADJUSTED SO THAT ALL PARAMETERS ARE REMOVED FROM THE STACK UPON RETURN.

ASSEMBLY LANGUAGE INTERFACE RULES

- AN ASSEMBLY LANGUAGE PROCEDURE WHICH IS CALLED BY A HLL PROGRAM MUST REMOVE ALL PARAMETERS FROM THE STACK.
- AN ASSEMBLY LANGUAGE PROGRAM CAN EXPECT THE STACK, UPON RETURN FROM HLL PROCEDURE, TO NO LONGER CONTAIN THE PARAMETERS IT PUSHED.
- AN ASSEMBLY LANGUAGE PROCEDURE WHICH IS CALLED BY A HLL PROGRAM MUST SAVE DS, SS, SP, AND BP, IF THEY ARE TO BE MODIFIED.
- AN ASSEMBLY LANGUAGE PROGRAM CALLING A HLL PROCEDURE CANNOT EXPECT ANY REGISTERS EXCEPT DS, SS, SP, AND BP TO BE PRESERVED.

EXAMPLE

- A PL/M COMPATIBLE PROCEDURE IS REQUIRED TO FIND THE MEAN OF TWO VALUES. ASSUME THAT THE PROCEDURE MUST BE OF TYPE FAR.

GIVEN:

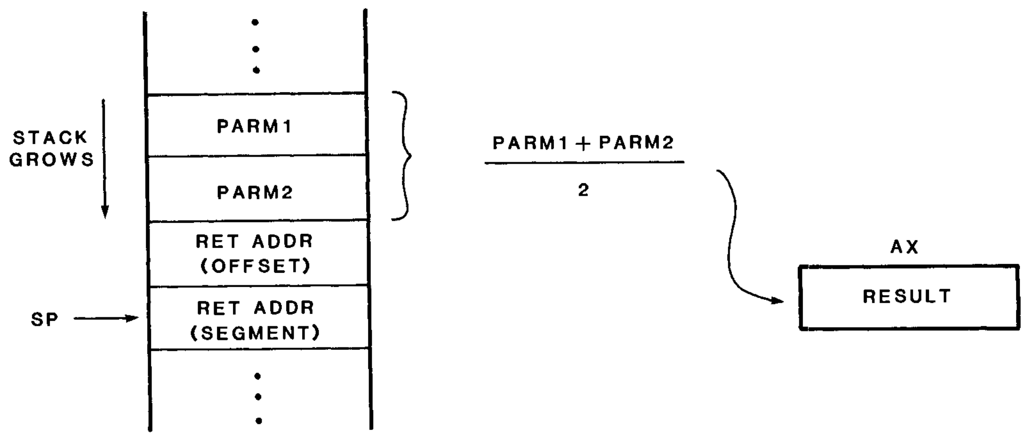
```
MEAN: PROCEDURE (PARM1, PARM2) INTEGER EXTERNAL;
   DECLARE PARM1 INTEGER,
           PARM2 INTEGER;

END MEAN;
```

WHERE DO WE FIND THE INPUT PARAMETERS?

WHERE DO WE LEAVE THE RESULT?

EXAMPLE (CONT.)

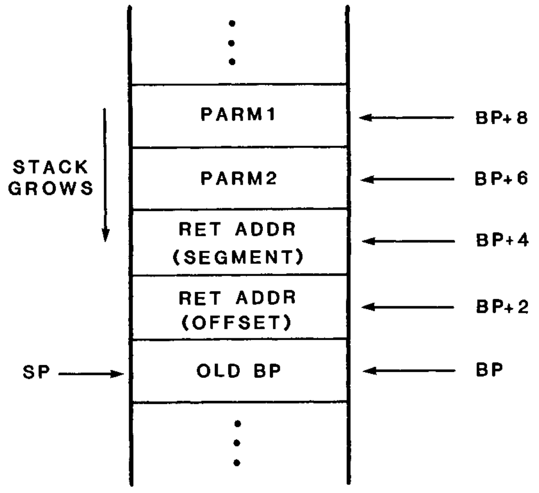
PARAMETER PASSING



HOW DO WE REFERENCE PARAMETERS ON THE STACK?

EXAMPLE (CONT.)

STACK FRAME AFTER SAVING THE BP REGISTER



EXAMPLE (CONT.)

ASSEMBLY LANGUAGE MODULE

```
                 NAME        MEAN_VALUE

                 PUBLIC      MEAN

MEAN_SEG         SEGMENT    'CODE'
                 ASSUME      CS:MEAN_SEG

MEAN             PROC        FAR
                 PUSH        BP                ;SAVE CALLER'S BP.
                 MOV         BP, SP            ;SET UP NEW BP.
                 MOV         AX, [BP+8]        ;GET PARM1.
                 ADD         AX, [BP+6]        ;ADD IT TO PARM2.
                 SAR         AX, 1             ;DIVIDE RESULT BY 2,
                 POP         BP                ;RESTORE BP.
                 RET         4                 ;RETURN AND CLEAN UP STACK.
                                               ;RESULT LEFT IN AX.

MEAN             ENDP
MEAN_SEG         ENDS
                 END
```

USING A STRUCTURE AS A STACK TEMPLATE

```
              NAME        MEAN_VALUE_1
              PUBLIC      MEAN
STACK_FRAME   STRUC
    OLD_BP      DW        ?
    RET_ADDR    DD        ?
    PARM2       DW        ?
    PARM1       DW        ?
STACK_FRAME   ENDS

MEAN_SEG      SEGMENT
              ASSUME      CS:MEAN_SEG
MEAN          PROC        FAR
              PUSH        BP                  ;SAVE CALLER'S BP.
              MOV         BP,SP               ;SET UP NEW BP.
              MOV         AX, [BP].PARM1      ;GET PARM1.
              ADD         AX, [BP].PARM2      ;ADD IT TO PARM2.
              SAR         AX,1                ;DIVIDE RESULT BY 2.
              POP         BP                  ;RESTORE BP.
              RET         4                   ;RETURN AND CLEAN UP STACK.
                                              ;RESULT LEFT IN AX.
MEAN          ENDP
MEAN_SEG      ENDS
              END
```

CLASS EXERCISE 9.1

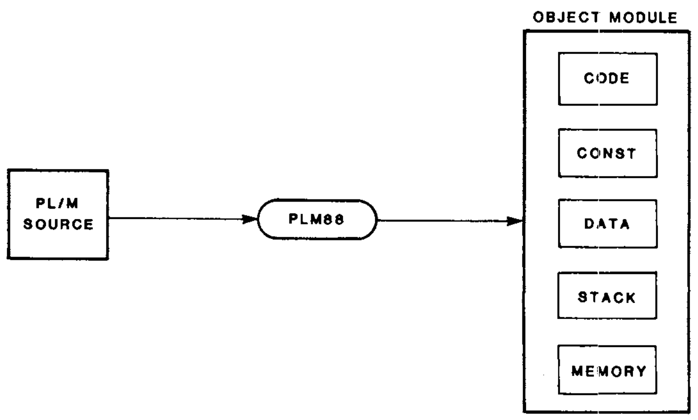
WRITE A PL/M COMPATIBLE PROCEDURE THAT WILL COMPARE TWO BYTE ARRAYS FOR "COUNT" NUMBER OF BYTES. IF THE STRINGS COMPARE, RETURN A VALUE OF TRUE (0FFH). OTHERWISE, RETURN A VALUE OF FALSE (0).

REFER TO THE FOLLOWING PL/M PROCEDURE DECLARATION WHEN WRITING YOUR CODE:

```
   CMP_STRING: PROCEDURE  (STR1_PTR,STR2_PTR,COUNT) BYTE EXTERNAL;
        DECLARE (STR1_PTR,STR2_PTR) POINTER,
                  COUNT WORD;
   END CMP_STRING;
```

PLACE YOUR CODE IN A GROUP NAMED CGROUP. PLACE ANY DATA YOU DEFINE IN A GROUP NAMED DGROUP. ASSUME THAT THE DS REGISTER IS ALREADY POINTING TO DGROUP. ALSO, ASSUME THAT ALL DATA POINTERS ARE 16 BITS AND THAT ALL PROCEDURES ARE OF TYPE NEAR.

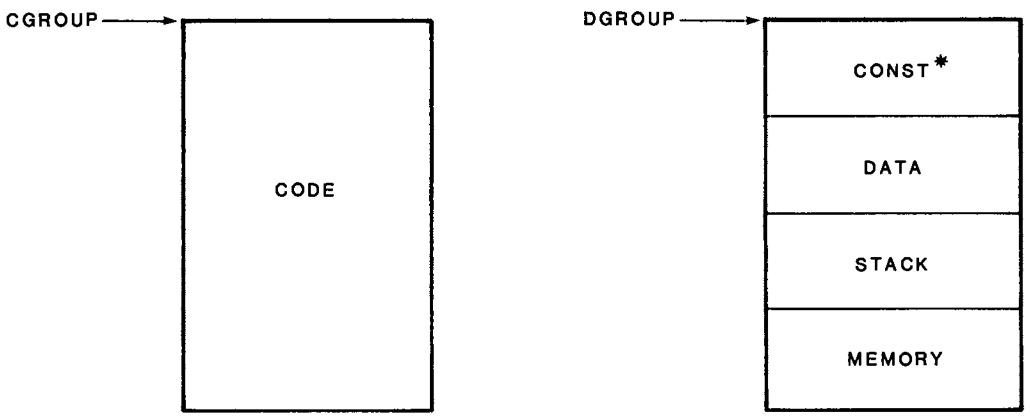
PL/M MEMORY ALLOCATION



- THE OBJECT MODULE PRODUCED BY THE COMPILER CONTAINS FIVE SECTIONS OR SEGMENTS.
- THE MAXIMUM SIZE ALLOWABLE FOR EACH OF THESE SEGMENTS IS DETERMINED BY THE SELECTED COMPILER SIZE CONTROL.

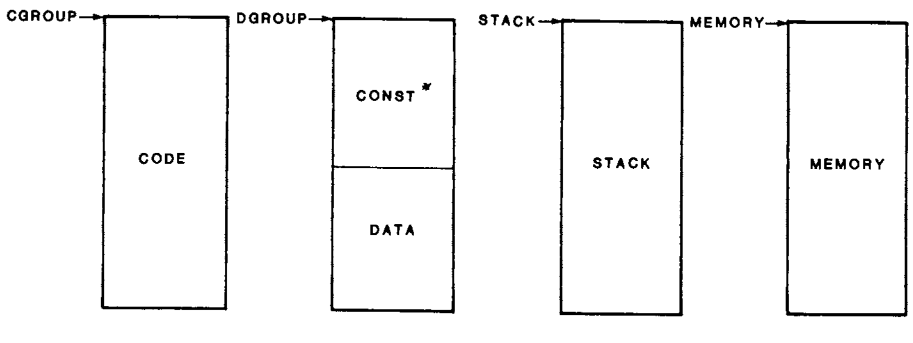
-SMALL	-MEDIUM
-COMPACT	-LARGE

SMALL MODEL



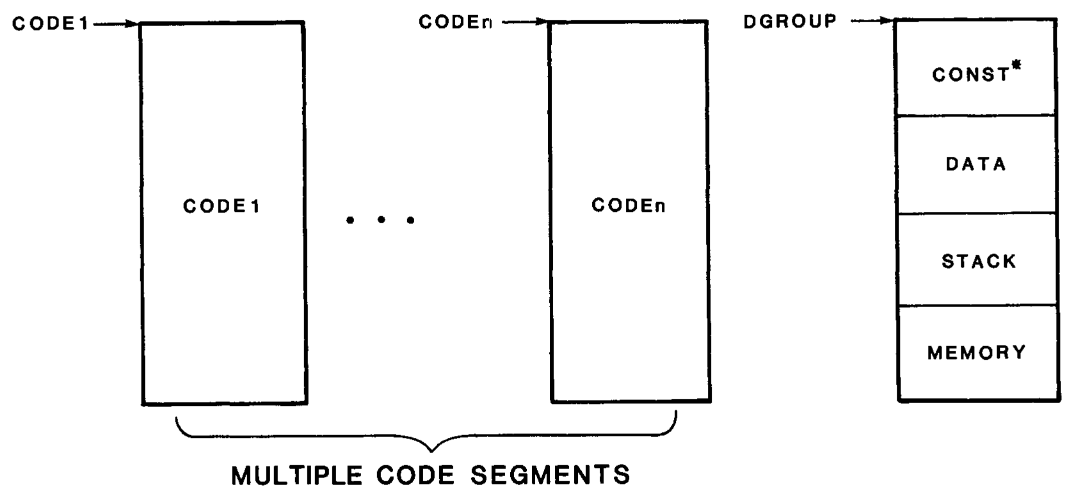
* – THE CONST SEGMENT CAN BE PLACED IN CGROUP IF THE "ROM" CONTROL IS SPECIFIED AT COMPILE TIME.

COMPACT MODEL



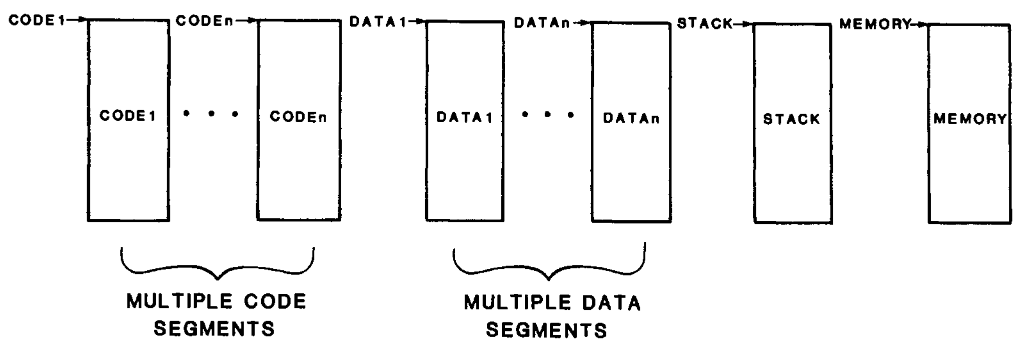
* – THE CONST SEGMENT CAN BE PLACED IN CGROUP IF THE "ROM" CONTROL IS SPECIFIED AT COMPILE TIME.

MEDIUM MODEL



* – THE CONST SEGMENT WITHIN EACH MODULE CAN BE MERGED WITH ITS CORRESPONDING CODE SEGMENT IF THE "ROM" CONTROL IS USED AT COMPILE TIME.

LARGE MODEL



NOTE: THE CONST SEGMENT WITHIN EACH MODULE IS MERGED WITH ITS CORRESPONDING CODE SEGMENT.

PLM CLASS NAMES

TYPE OF SEGMENT	CLASS NAME
CODE	CODE
CONSTANT	CONST *
DATA	DATA
STACK	STACK
MEMORY	MEMORY

* CONSTANTS ARE MERGED WITH THE CODE SEGMENT WHEN USING LARGE MODEL.

CONVENTIONS FOR PROCEDURES AND PROGRAM LABEL DEFINITIONS
SMALL AND COMPACT MODELS

```
                NAME     CODE_EXAMPLE_1

CGROUP          GROUP    CODE1

                PUBLIC   START,PROC1
                EXTRN    PROC2:NEAR,PROC3:NEAR

CODE1           SEGMENT  'CODE'
                ASSUME   CS:CGROUP

PROC1           PROC     NEAR
                 •
                 •
                 •
                RET
PROC1           ENDP

START:          CALL     PROC1
                CALL     PROC2
                CALL     PROC3
                JMP      START

CODE1           ENDS
                END
```

- ALL LOGICAL CODE SEGMENTS ARE CONTAINED IN ONE PHYSICAL GROUP NAMED CGROUP.
- ALL PROCEDURES AND PROGRAM LABELS COMMON TO BOTH PL/M AND ASSEMBLY LANGUAGE MODULES MUST BE DEFINED AS NEAR.

CONVENTIONS FOR PROCEDURE AND PROGRAM LABEL DEFINITIONS MEDIUM AND LARGE MODELS

```
                NAME     CODE_EXAMPLE_2

                PUBLIC   START,PROC1
                EXTRN    PROC2:FAR,PROC3:FAR

CODE1           SEGMENT  'CODE'
                ASSUME   CS:CODE1

PROC1           PROC     FAR
                 •
                 •
                 •
                RET
PROC1           ENDP

START:          CALL     PROC1
                CALL     PROC2
                CALL     PROC3
                JMP      START

CODE1           ENDS
                END
```

- **CODE IS CONTAINED IN A NUMBER OF PHYSICAL CODE SEGMENTS.**
- **ALL PROCEDURES AND PROGRAM LABELS COMMON TO BOTH PL/M AND ASSEMBLY LANGUAGE MODULES MUST BE DEFINED AS FAR.**

CONVENTIONS FOR DATA DEFINITIONS SMALL MODEL

```
               NAME     DATA_EXAMPLE_1

DGROUP         GROUP    CONST1,DATA1,STACK,MEMORY

CONST1         SEGMENT  PUBLIC 'DATA'
;              Constant data definitions go here.
;              Don't forget that constants could be
;              merged with the code segments.
CONST1         ENDS

DATA1          SEGMENT PUBLIC 'DATA'
;              Variable data definitions go here.
DATA1          ENDS

STACK          SEGMENT STACK    'STACK'
;              Stack definitions go here.
;              Make sure that the segment definition
;              is identical to the one used by PL/M.
STACK          ENDS

MEMORY         SEGMENT MEMORY  'MEMORY'
;              Data to be placed in the memory segment
;              is defined here.
;              Make sure that the segment definition
;              is identical to the one used by PL/M.
MEMORY         ENDS

CGROUP         GROUP   CODE1

CODE1          SEGMENT PUBLIC 'CODE'
               ASSUME  CS:GROUP
               ASSUME  DS:DGROUP,SS:DGROUP
                 .
                 .
                 .
CODE1          ENDS
               END
```

- **ALL PROGRAM DATA IS CONTAINED IN A GROUP NAMED DGROUP.**
- **DATA POINTERS IN SMALL MODEL WITH CONSTANTS IN DGROUP ARE 16 BITS (OFFSET ONLY). WITH CONSTANTS IN CGROUP, THE DATA POINTERS ARE 32 BITS. (SEGMENT : OFFSET)**

CONVENTIONS FOR DATA DEFINITIONS MEDIUM MODEL

```
                NAME     DATA_EXAMPLE_2

DGROUP          GROUP    CONST1,DATA1,STACK,MEMORY

CONST1          SEGMENT PUBLIC 'DATA"
;               Constant data definitions go here.
;               Don't forget that constants could be
;               merged with the code segments.
CONST1          ENDS

DATA1           SEGMENT PUBLIC 'DATA'
;               Variable data definitions go here.
DATA1           ENDS

STACK           SEGMENT STACK 'STACK'
;               Stack definitions go here.
;               Make sure that the segment definition
;               is identical to the one used by PL/M.
STACK           ENDS

MEMORY          SEGMENT MEMORY 'MEMORY'
;               Data to be placed in the memory segment
;               is defined here.
;               Make sure that the segment definition
;               is identical to the one used by PL/M.
MEMORY          ENDS

CODE1           SEGMENT 'CODE'
                ASSUME  CS:CODE1
                ASSUME  DS:DGROUP,SS:DGROUP
                   .
                   .
                   .
CODE1           ENDS
                END
```

- ALL PROGRAM DATA IS CONTAINED IN A GROUP NAMED DGROUP.
- DATA POINTERS IN MEDIUM MODEL ARE 32 BITS (SEGMENT : OFFSET).

CONVENTIONS FOR DATA DEFINITIONS COMPACT MODEL

```
                NAME     DATA_EXAMPLE_3

DGROUP          GROUP    CONST1,DATA1

CONST1          SEGMENT PUBLIC 'DATA'
;               Constant data definitions go here.
;               Don't forget that constants could be
;               merged with the code segments.
CONST1          ENDS

DATA1           SEGMENT PUBLIC 'DATA'
;               Variable data definitions go here.
DATA1           ENDS

STACK           SEGMENT STACK 'STACK'
;               Stack definitions go here.
;               Make sure that the segment definition
;               is identical to the one used by PL/M.
STACK           ENDS

MEMORY          SEGMENT MEMORY  'MEMORY'
;               Data to be placed in the memory segment
;               is defined here.
;               Make sure that the segment definition
;               is identical to the one used by PL/M.
MEMORY          ENDS

CGROUP          GROUP    CODE1

CODE1           SEGMENT PUBLIC 'CODE'
                ASSUME  CS:GROUP
                ASSUME  DS:GROUP,DS:STACK
                   .
                   .
                   .
CODE1           ENDS
                END
```

- VARIABLE AND CONSTANT DATA IS CONTAINED IN A GROUP NAMED DGROUP.
- STACK AND MEMORY ARE EACH ALLOCATED ONE PHYSICAL SEGMENT.
- DATA POINTERS ARE 32 BITS (SEGMENT : OFFSET).

CONVENTIONS FOR DATA DEFINITIONS LARGE MODEL

```
                NAME    DATA_EXAMPLE_4

DATA1           SEGMENT 'DATA'
;               Variable data definitions go here.
DATA1           ENDS

DATA2           SEGMENT 'DATA'
;               Variable data definitions go here.
DATA2           ENDS

STACK           SEGMENT STACK 'STACK'
;               Stack definitions go here.
;               Make sure that the segment definition
;               is identical to the one used by PL/M.
STACK           ENDS

MEMORY          SEGMENT MEMORY 'MEMORY'
;               Data to be placed in the memory segment
;               is defined here.
;               Make sure that the segment definition
;               is identical to the one used by PL/M.
MEMORY          ENDS

CODE1           SEGMENT 'CODE'
                ASSUME  CS:CODE1,DS:DATA1,SS:STACK

;               Constants are defined within the code
;               segment.
                  .
                  .
                  .
CODE1           ENDS
                END
```

- **VARIABLE DATA IS CONTAINED IN MULTIPLE DATA SEGMENTS.**
- **CONSTANT DATA IS MERGED WITH A MODULE'S CODE SEGMENT.**
- **STACK AND MEMORY ARE EACH ALLOCATED ONE PHYSICAL SEGMENT.**
- **DATA POINTERS ARE ALL 32 BITS (SEGMENT : OFFSET).**

EXAMPLE

- A PL/M COMPATIBLE PROCEDURE IS REQUIRED TO SUM THE ELEMENTS OF A BYTE ARRAY. ASSUME THAT THE PL/M MODULE HAS BEEN COMPILED USING THE SMALL MODEL OF SEGMENTATION.

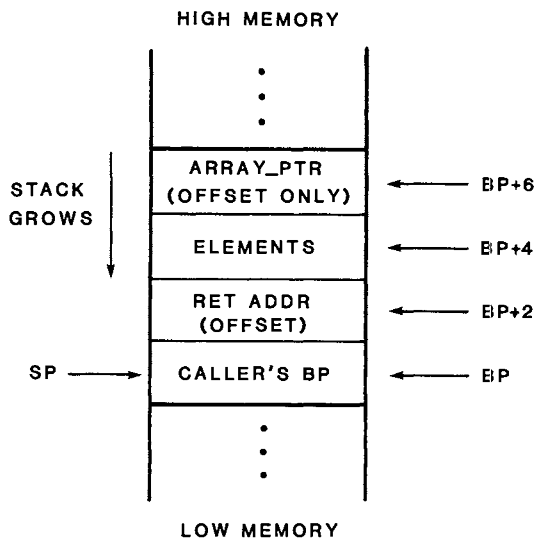
GIVEN:

```
ARRAY_SUM: PROCEDURE(ARRAY_PTR,ELEMENTS) INTEGER EXTERNAL;
    DECLARE ARRAY_PTR POINTER,
            ELEMENTS WORD;
    END  ARRAY_SUM;
```

WHAT MAKES UP A POINTER IN SMALL MODEL?

EXAMPLE (CONT.)

STACK FRAME



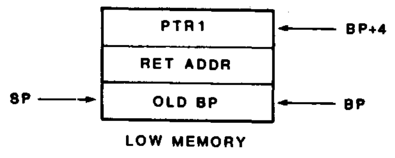
EXAMPLE (CONT.)

ASSEMBLY LANGUAGE MODULE

```
                NAME       ARRAY_SUM_MOD

                PUBLIC     ARRAY_SUM

CGROUP          GROUP      ARRAY_SUM_SEG

ARRAY_SUM_SEG   SEGMENT    'CODE'
                ASSUME     CS:CGROUP

ARRAY_SUM       PROC       NEAR
                PUSH       BP              ;SAVE CALLER'S BP.
                MOV        BP,SP           ;SET UP NEW BP.
                MOV        BX, [BP+6]      ;SET UP ARRAY POINTER.
                MOV        CX, [BP+4]      ;SET UP ITEM COUNT.
                MOV        AX,0            ;CLEAR SUM.
AGAIN:          ADD        AX,DS: [BX]     ;ADD ARRAY ELEMENT TO SUM.
                INC        BX              ;UPDATE ARRAY POINTER.
                LOOP       AGAIN           ;IF CX≠0, DO IT AGAIN.
                POP        BP              ;RESTORE BP.
                RET        4               ;RETURN AND CLEAN UP STACK.
                                           ;RESULT LEFT IN AX.

ARRAY_SUM       ENDP
ARRAY_SUM_SEG   ENDS
                END
```

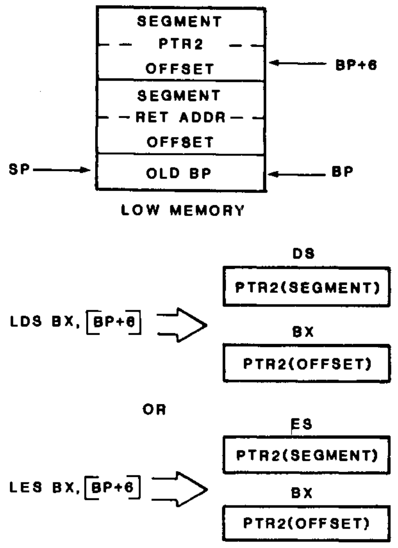
LOADING POINTERS

16 BIT POINTERS



MOV BX,[BP+4]

32 BIT POINTERS



CLASS EXERCISE 9.2

REWRITE THE ARRAY SUM PROCEDURE. THIS TIME ASSUME THAT IT MUST INTERFACE WITH A PL/M MODULE COMPILED LARGE

WHERE TO FIND MORE INFORMATION...

AN INTRODUCTION TO ASM86
    CHAPTER 5 - COMBINING ASM86 AND PL/M-86 MODULES

PL/M-86 USER'S GUIDE
    APPENDIX F - LINKING TO MODULES WRITTEN IN OTHER LANGUAGES

CHAPTER 10

LINKAGE WITH OTHER HIGH LEVEL LANGUAGES

- LINKING WITH 'C'
- LINKAGE WITH PASCAL
- LINKAGE WITH FORTRAN

THINGS TO CONSIDER WHEN LINKING TO HLL'S

- COMPATIBLE DATA TYPES
- COMPILATION MODELS (SMALL, LARGE ETC.)
- PASSING PARAMETERS TO PROCEDURES

●● MANY PRINCIPLES OF LINKING TO PL/M ARE APPLICABLE

COMPATIBLE DATA TYPES

ASM86	PL/M86	PASCAL	FORTRAN	'C'	COMMENTS
UNSIGNED DATA TYPES					
DB	BYTE	CHAR	CHARACTER	CHAR	
DW	WORD	WORD	------	------	
DD	DWORD	------	------	------	
INTEGERS					
DB	------	------	INTEGER*1	------	
DW	INTEGER	INTEGER	INTEGER*2	INT,SHORT	
DD	------	LONGINT	INTEGER*4	LONG	8087 SHORT INTEGER
BOOLEAN VALUES (IE TRUE/FALSE)					
DB	BYTE	BOOLEAN	LOGICAL*1	------	
DW	------	------	LOGICAL*2	------	
DD	------	------	LOGICAL*4	------	
REAL NUMBERS					
DD	REAL	REAL	REAL*4	FLOAT	8087 SHORT REAL
DQ	------	------	REAL*8, DOUBLE PRECISION	DOUBLE	8087 LONG REAL
DT	LONGREAL	TEMPREAL	TEMPREAL	------	8087 TEMPORARY REAL

* OTHER DATA TYPES : THESE LANGUAGES SUPPORT ARRAYS AND STRUCTURES IN VARYING DEGREES. THEY ALSO USE POINTERS (16 BITS IN SMALL , 32 BITS OTHERWISE). SEE APPROPRIATE LANGUAGE REFERENCE MANUAL FOR DETAILS.

COMPILATION MODELS

- LANGUAGES SAME CONVENTIONS (CLASS NAMES, GROUPS ETC.) AS PL/M.

	SMALL	COMPACT	MEDIUM	LARGE
PL/M	X	X	X	X
PASCAL	X	X		X
FORTRAN				X
C	X			X

PARAMETER PASSING

- ALL THREE LANGUAGES PASS PARAMETERS INTO PROCEDURES ON STACK.
- RETURNED VALUES (FUNCTIONS, TYPED PROCEDURES) PASSED IN REGISTERS, REALS ON TOP OF 8087 STACK.
- PARAMETERS PASSED IN ONE OF TWO WAYS:

  1) BY VALUE - PARAMETER READ FROM MEMORY AND PUSHED ONTO STACK.

  2) BY REFERENCE - ADDRESS OF PARAMETER IS PASSED TO PROCEDURE. SAME AS PASSING POINTERS IN PL/M.

PARAMETER PASSING : PASCAL

- PARAMETERS USUALLY PASSED BY VALUE
- 'VAR' PARAMETERS PASSED BY REFERENCE
- PARAMETERS PUSHED LEFT-TO-RIGHT
- 8087 STACK USED FOR FIRST SEVEN REALS
- PROCEDURE CLEANS PARAMETERS FROM STACK

EXAMPLE

```
PROCEDURE  PROC (PARM1, PARM2: INTEGER; PARM3:real;
                 VAR PARM4:INTEGER; PARM5:REAL);

PROC (A,B,C,D,E);
```

PARAMETER PASSING : FORTRAN

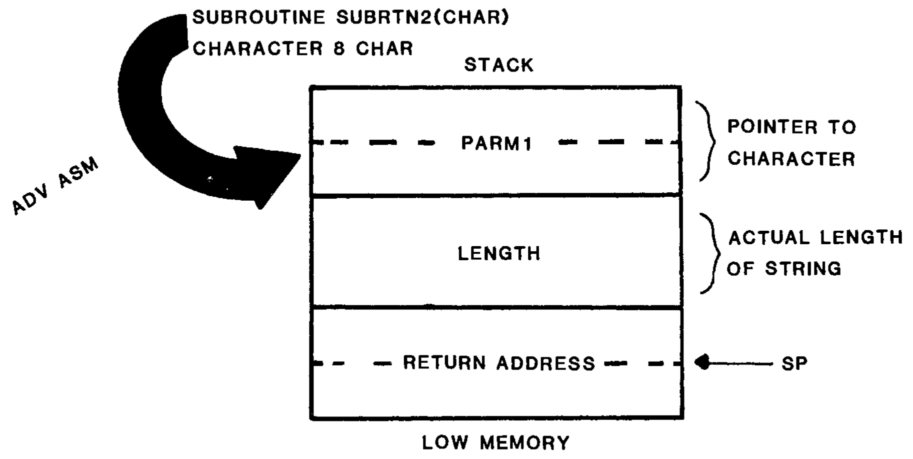
- ALL PARAMETERS PASSED BY REFERENCE (ALL POINTERS 32 BITS)
- PARAMETERS PASSED LEFT-TO-RIGHT
- REALS ALSO PASSED BY REFERENCE
- PROCEDURE CLEANS PARAMETERS FROM STACK

EXAMPLE

```
SUBROUTINE SBRTNI (PARM1, PARM2, PARM3, PARM4)

CALL SBRTNI (A,B,C,D)
```

PASSING PARAMETERS OF THE CHARACTER DATA TYPE

- TO PASS A CHARACTER TYPE DATA ARGUMENT, A POINTER TO THE CHARACTER STRING AND THE ACTUAL LENGTH OF THE STRING (IN BYTES) IS PUSHED ON THE STACK.



PARAMETER PASSING : 'C'

- ALL PARAMETERS PASSED BY VALUE
- PARAMETERS PUSHED RIGHT-TO-LEFT
- VARIABLE NUMBER OF PARAMETERS ALLOWED
- 8087 STACK USED FOR FIRST SEVEN REALS
- CALLING PROGRAM REMOVES PARAMETERS FROM STACK

```
INT X,*P; /*X IS INTEGER, P POINTER TO INTEGER*/

INT F (); /*F IS FUNCTION, NO PARAMETER COUNT*/

   F(X,P);  /*X PASSED BY VALUE, P IS A POINTER*/
```

HIGH LEVEL LANGUAGE INTERFACING : CHECK LIST

1. PUBLIC AND EXTERNAL DATA DEFINITIONS MUST MATCH HLL DATA TYPE
2. FOLLOW COMPILATION MODEL (SMALL, COMPACT ...) RULES
   - USE CORRECT CLASSNAMES/GROUPS
   - IF USING GROUPS:
     - CS, DS, ES ADDRESS GROUP BASE (NOT SEGMENT BASE)
     - USE OF MOV BX, OFFSET DGROUP: VARIABLE (OR USE LEA INSTRUCTION)
   - ARE POINTERS (AND RETURN ADDRESSES) 16 BITS OR 32 BITS?
3. PASSING PARAMETERS
   - IS THE STACK FRAME RIGHT?
   - REMOVE CORRECT BYTE COUNT ON RETURN FROM PROCEDURE
   - LEAVE RETURN VALUES IN CORRECT REGISTERS
4. REGISTERS WHICH ONES WILL/MAY BE DESTROYED? BP IS SACRED!

WHERE TO FIND MORE INFORMATION...

PASCAL-86 USER'S GUIDE

APPENDIX J - LINKING TO MODULES WRITTEN IN OTHER LANGUAGES

FORTRAN-86 USER'S GUIDE

APPENDIX H - LINKING TO SUBPROGRAMS WRITTEN IN OTHER LANGUAGES

C-86 COMPILER USER'S GUIDE

DAY 3 OBJECTIVES

BY THE TIME YOU FINISH TODAY YOU WILL:

- SEE THE ARCHITECTURE OF THE 8087
- DEFINE THE 8086-8087 INTERFACE (HARDWARE AND SOFTWARE)
- DEFINE THE DATA FORMATS USED FOR REAL, INTEGER AND BCD NUMBERS
- USE THE 8087 INSTRUCTION SET
- INITIALIZE THE 8087
- DISCUSS EXCEPTION HANDLING FOR ARITHMETIC ERRORS
- DEFINE THE USE OF THE 8087 SUPPORT LIBRARIES

CHAPTER 11

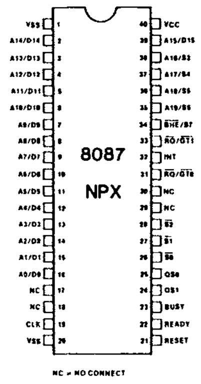
INTRODUCTION TO THE 8087 NUMERIC PROCESSOR EXTENSION

- MOTIVATION FOR USING THE 8087
- ARCHITECTURAL DESCRIPTION
- HARDWARE INTERFACE
- SOFTWARE INTERFACE

8087 80-BIT HMOS NUMERIC PROCESSOR EXTENSION

- FULL INTERNAL 80-BIT ARCHITECTURE FOR HIGH PERFORMANCE
- IMPLEMENTS PROPOSED IEEE FLOATING POINT STANDARD
- EXPANDS HOST CPU DATATYPES TO INCLUDE 32-, 64-BIT INTEGERS, 32-, 64-, 80-BIT FLOATING POINT, AND 18-DIGIT BCD OPERANDS
- ALL HOST CPU ADDRESSING MODES AVAILABLE
- DIRECTLY EXTENDS HOST CPU'S INSTRUCTION SET TO TRIGONOMETRIC, LOGARITHMIC, EXPONENTIAL AND ARITHMETIC INSTRUCTIONS FOR ALL DATATYPES
- 8 x 80-BIT, INDIVIDUALLY ADDRESSABLE, NUMERIC REGISTER STACK
- BUILT-IN EXCEPTION HANDLING FUNCTIONS

NOTE: THE 8087 IS AN EXTENSION
OF THE HOST CPU
(iAPX 86,88 OR iAPX 186,188)



WHY USE AN 8087?

- TO MAKE IT EASIER TO PROGRAM <u>ACCURATE</u> ARITHMETIC SOFTWARE
- TO BRING ABOUT <u>STANDARDIZATION</u> OF NUMERIC PROGRAMS AND DATA
- TO MEET THE <u>HIGH PERFORMANCE</u> MATH REQUIREMENTS OF VARIOUS APPLICATION PROGRAMS

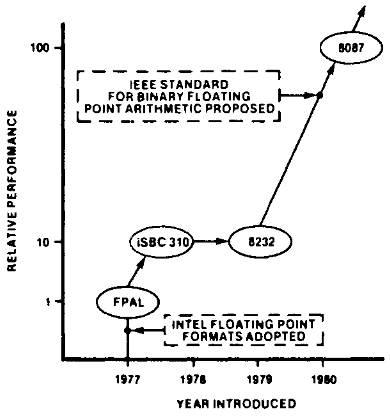
RELIABILITY - WHAT CAN AN 8087 DO FOR YOU?

- THE 8087 IS DESIGNED TO DELIVER STABLE, ACCURATE RESULTS
- IT CAN PROCESS DECIMAL NUMBERS UP TO 18 DIGITS OF SIGNIFICANCE - WITHOUT ROUND-OFF ERRORS
- IT CAN PERFORM EXACT ARITHMETIC ON INTEGERS AS LARGE AS 2^{64} (APPROXIMATELY EQUAL TO 1.845×10^{19}).

STANDARDIZATION

- THE 8087 IS THE FIRST FULL IMPLEMENTATION OF THE PROPOSED IEEE FLOATING POINT STANDARD
- DATA FORMATS AND BASIC ARITHMETIC FUNCTIONS ARE CONSISTENT WITH WITH OTHER INTEL PRODUCTS
  - iSBC-310
  - 8232
  - FPAL
  - ASM-86
  - PL/M-86
  - FORTRAN-86
  - PASCAL-86

HIGH PERFORMANCE

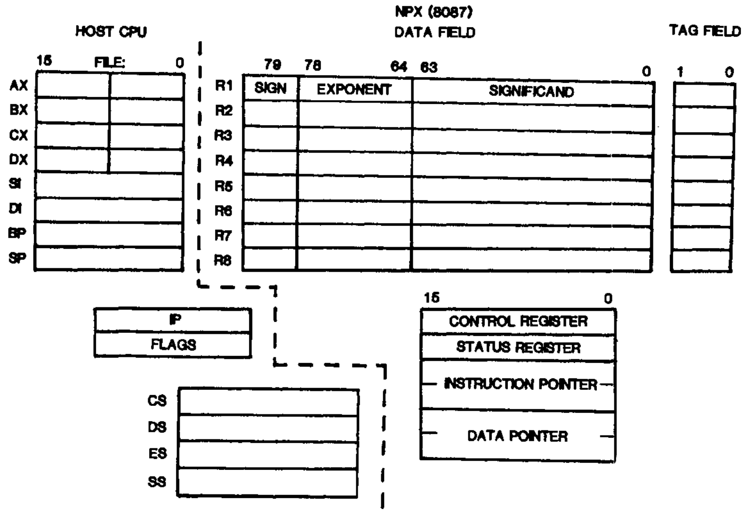


8087 EVOLUTION AND RELATIVE PERFORMANCE

Instruction	Approximate Execution Time (μs) (5 MHz Clock)	
	8087	8086 Emulation
Multiply (single precision)	19	1,600
Multiply (double precision)	27	2,100
Add	17	1,600
Divide (single precision)	39	3,200
Compare	9	1,300
Load (single precision)	9	1,700
Store (single precision)	18	1,200
Square root	36	19,600
Tangent	90	13,000
Exponentiation	100	17,100

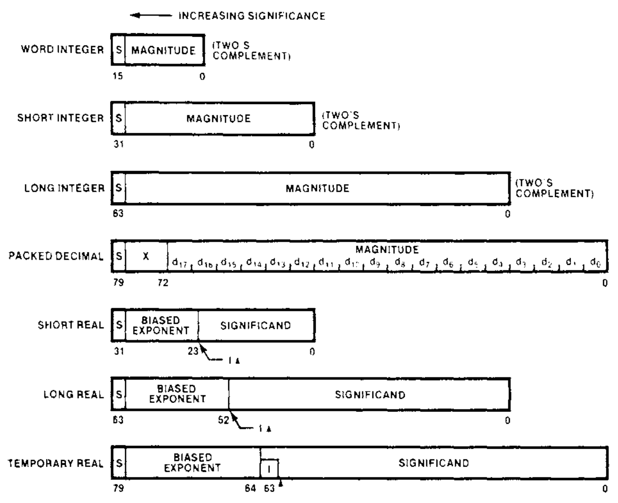
8087 vs SOFTWARE COMPARISON

iAPX 86/20, 88/20, 186/20, 188/20 ARCHITECTURE



- THE 8087 IS AN ARCHITECTURAL EXTENSION OF THE HOST CPU.
- TO USE THE 8087, ADDITIONAL OPCODES AND OPERANDS ARE INCLUDED IN THE HOST CPU's INSTRUCTION SET.

8087 DATA TYPES AND FORMATS



NOTES:
S = Sign bit (0 = positive, 1 = negative)
d_n = Decimal digit (two per byte)
X = Bits have no significance; 8087 ignores when loading, zeros when storing.
▲ = Position of implicit binary point
I = Integer bit of significand; stored in temporary real, implicit in short and long real
Exponent Bias (normalized values):
Short Real: 127 (7FH)
Long Real: 1023 (3FFH)
Temporary Real: 16383 (3FFFH)

ALL INTERNAL 8087 DATA IS IN THIS FORM. THE SIZE OF THE TEMPORARY REAL FORMAT CONTRIBUTES TO THE OVERALL ACCURACY AND STABILITY OF THE 8087

REAL FORMATS

SIGN – Ø = POSITIVE NUMBER
1 = NEGATIVE NUMBER

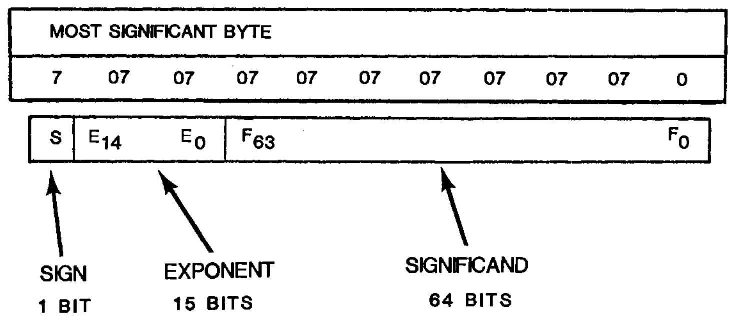
EXPONENT – EXPONENT IS BIASED TO ELIMINATE NEED FOR HANDLING NEGATIVE EXPONENTS. SHORT REAL HAS 8 BIT EXPONENT:

TRUE EXPONENT: $-127 \rightarrow +127$
BIASED EXPONENT: $0 \rightarrow +254$ BIAS = +127

SIGNIFICAND – CONTAINS SIGNIFICANT BITS (MANTISSA) OF NUMBER. IT IS USUALLY NORMALIZED, MEANING THAT IT CONTAINS BOTH A FRACTION AND WHOLE NUMBER. THIS ENSURES THE GREATEST PRECISION FOR A GIVEN REAL FORMAT

ASSUME WE HAVE A 5 DIGIT SIGNIFICAND AND WE WANT TO REPRESENT THIS NUMBER:
3,174,231
NORMALIZED NUMBER: 3.1742×10^6
UNNORMALIZED NUMBER: 0.0031×10^9

TEMPORARY REAL FORMAT



TEMPORARY REAL (CONT.)

EXPONENT -

SMALLEST -VE NUMBER = $2^{-16382} \simeq 3.36 \times 10^{-4932}$
LARGEST +VE NUMBER = $2^{16384} \simeq 1.19 \times 10^{4932}$
RADIUS OF UNIVERSE $\simeq$ 13 BILLION LIGHT YEARS
$\therefore$ VOLUME $\simeq 7.77 \times 10^{84}$ CM^3
IT WOULD TAKE $\simeq 10^{122}$ ELECTRONS TO FILL THIS VOLUME

SIGNIFICAND

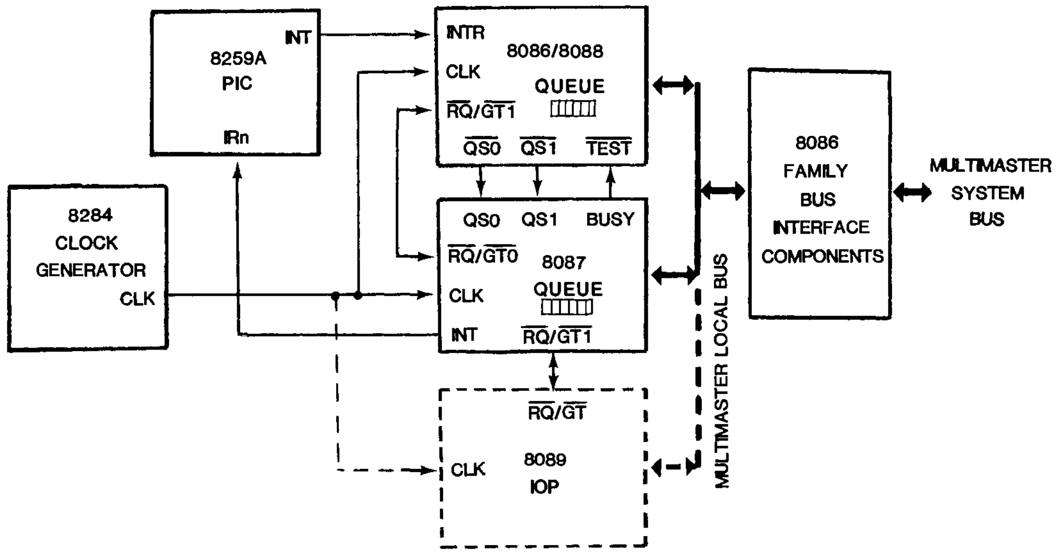
ACCURACY OF ONE PART IN 2^{64}

COMPARES WITH THE RADIUS OF A HYDROGEN ATOM NEXT TO RADIUS OF MOONS ORBIT ABOUT THE EARTH

8086↔8087 INTERFACE

- PROCESSORS CAN OPERATE IN PARALLEL
- SHARE SINGLE INSTRUCTION STREAM
- 8087 TRACKS QUEUE OF 8086 USING QUEUE STATUS LINES
- ALL 8086 ADDRESSING MODES AVAILABLE
- 8086 SUPPLIES OPERAND ADDRESSES TO 8087 BY ISSUING DUMMY READ

HARDWARE INTERFACE


8259A
PIC
INT
IRn
8284
CLOCK
GENERATOR
CLK
INTR
8086/8088
CLK
QUEUE
$\overline{RQ}/\overline{GT1}$
$\overline{QS0}$ $\overline{QS1}$ $\overline{TEST}$
QS0 QS1 BUSY
$\overline{RQ}/\overline{GT0}$
8087
CLK
QUEUE
INT
$\overline{RQ}/\overline{GT1}$
$\overline{RQ}/\overline{GT}$
CLK
8089
IOP
MULTIMASTER LOCAL BUS
8086
FAMILY
BUS
INTERFACE
COMPONENTS
MULTIMASTER
SYSTEM
BUS

- THE 8087 HAS A DIRECT HARDWARE INTERFACE WITH THE HOST CPU

PRINCIPAL INSTRUCTIONS OF THE 8087

CLASS	INSTRUCTION TYPES
DATA TRANSFER	LOAD AND STORE (FOR ALL DATA TYPES), EXCHANGE, FREE
ARITHMETIC	ADD, SUBTRACT, MULTIPLY, DIVIDE, SUBTRACT REVERSED, DIVIDE REVERSED, CALCULATE SQUARE ROOT, SCALE, INCREMENT, DECREMENT, USE REMAINDER, ROUND TO INTEGER, CHANGE SIGN, ABSOLUTE VALUE, EXTRACT MANTISSA OR EXPONENT
LOGICAL/RELATIONAL	COMPARE, EXAMINE, TEST
TRANSCENDENTAL *	CALCULATE TANGENT, ARCTANGENT, $2^X - 1$, $Y \cdot \log_2 X$, $Y \cdot (\log_2 X + 1)$
CONSTANTS *	$0, 1, \pi, \log_{10} 2, \log_e 2, \log_2 10, \log_2 e$
PROCESSOR CONTROL	LOAD CONTROL WORD, STORE CONTROL WORD, STORE STATUS WORD, LOAD ENVIRONMENT, STORE ENVIRONMENT, SAVE, RESTORE, SET INTERRUPT-ENABLE ENABLE, CLEAR INTERRUPT-ENABLE, CLEAR ERRORS, INITIALIZE

* COMBINING THESE INSTRUCTIONS IN VERY SIMPLE ROUTINES PROVIDES ALL THE COMMON TRIGONOMETRIC, INVERSE HYPERBOLIC, INVERSE HYPERBOLIC, LOGARITHMIC, AND POWER FUNCTIONS.

8087 SOFTWARE SUPPORT

- 8087 SOFTWARE EMULATORS
  - FULL 16K EMULATOR (E8087)
  - PARTIAL 8K EMULATOR (PE8087) FOR PL/M-86
- LANGUAGE SUPPORT
  - ASM-86
  - PL/M-86
  - FORTRAN-86
  - PASCAL-86
- SUPPORT LIBRARIES

- CEL87	COMMON ELEMENTARY FUNCTIONS
- DCON87	DECIMAL CONVERSION
- EH87	ERROR HANDLER

WHERE TO FIND MORE INFORMATION...

APPLICATION NOTE AP-113 - GETTING STARTED WITH THE NUMERIC DATA PROCESSOR

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)
CHAPTER 6 - THE 8087 NUMERIC PROCESSOR EXTENSION

CHAPTER 12

PROGRAMMING THE 8087

- INSTRUCTION FORMAT
- DATA FORMATS
- DATA TRANSFER INSTRUCTIONS
- ARITHMETIC INSTRUCTIONS
- TRANSCENDENTAL INSTRUCTIONS
- CONSTANT INSTRUCTIONS

INSTRUCTION FORMAT

OPCODE [OP1 [,OP2]]

- DEPENDING ON INSTRUCTION TYPE, ONE OR TWO OPERANDS MAY BE EXPLICITLY SPECIFIED

- WITH SOME INSTRUCTIONS, THE OPERAND(S) MAY BE IMPLICITLY SPECIFIED

OPERANDS

- 3 TYPES
  - IMPLICIT REFERENCE TO THE STACK TOP (ST) AND POSSIBLY THE NEXT STACK ELEMENT (ST(1))

```
EX.     FLDZ                 ; PUSH Ø.Ø ONTO THE STACK
```

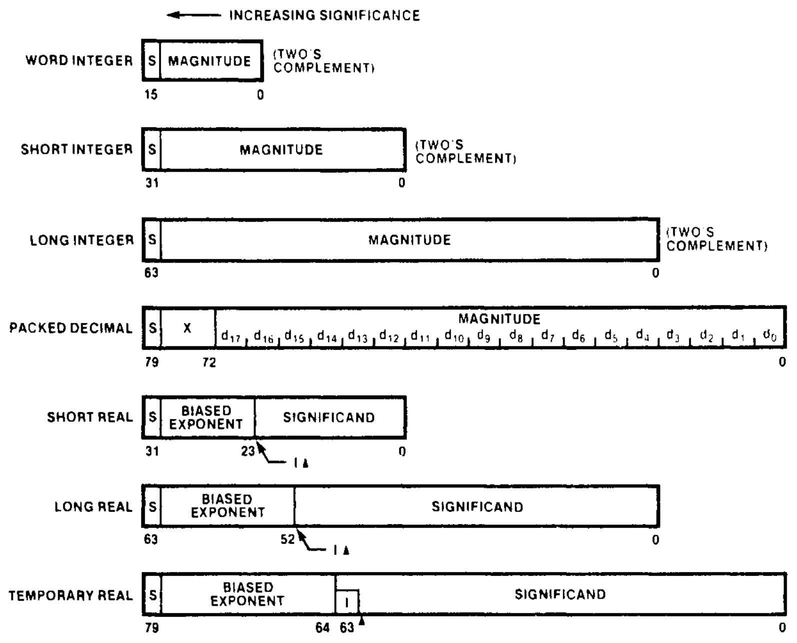
  - EXPLICIT REFERENCE TO STACK ELEMENT(S)

```
EX.     FADD    ST(1),ST     ;ST(1) = ST(1) + ST
```

  - EXPLICIT REFERENCE TO A MEMORY ITEM

```
EX.     FMUL    VAR1         ;ST = ST*VAR1
```

DATA FORMATS FOR MEMORY OPERANDS



STORAGE ALLOCATION DIRECTIVE

DIRECTIVE	MEANING	USE
DW	DEFINE WORD	WORD INTEGER
DD	DEFINE DOUBLEWORD	SHORT INTEGER, SHORT REAL
DQ	DEFINE QUADWORD	LONG INTEGER, LONG REAL
DT	DEFINE TENBYTE	PACKED DECIMAL, TEMPORARY REAL

PTR DIRECTIVES

WORD PTR	DWORD PTR
QWORD PTR	TBYTE PTR

ADDRESSING MODE EXAMPLES

CODING		INTERPRETATION
FIADD	ALPHA	ALPHA IS A SIMPLE SCALAR (MODE IS DIRECT)
FDIVR	ALPHA.BETA	BETA IS A FIELD IN A STRUCTURE THAT IS "OVERLAID" ON ALPHA (MODE IS DIRECT)
FMUL	QWORD PTR [BX]	BX CONTAINS THE ADDRESS OFA LONG REAL VARIABLE (MODE IS REGISTER INDIRECT)
FSUB	ALPHA [SI]	ALPHA IS AN ARRAY AND SI CONTAINS THE OFFSET OF AN ARRAY ELEMENT FROM THE START OF THE ARRAY (MODE IS INDEXED)
FILD	[BP].BETA	BP CONTAINS THE ADDRESS OF A STRUCTURE ON THE CPU STACK AND BETA IS A FIELD IN THE STRUCTURE (MODE IS BASED)
FBLD	TBYTE PTR [BX][DI]	BX CONTAINS THE ADDRESS OF A PACKED DECIMAL ARRAY AND DI CONTAINS THE OFFSET OF AN ARRAY ELEMENT (MODE IS BASED INDEXED)

12-5

INSTRUCTION SET

- DATA TRANSFER
- ARITHMETIC
- LOGICAL/RELATIONAL
- TRANSCENDENTAL
- CONSTANTS
- PROCESSOR CONTROL

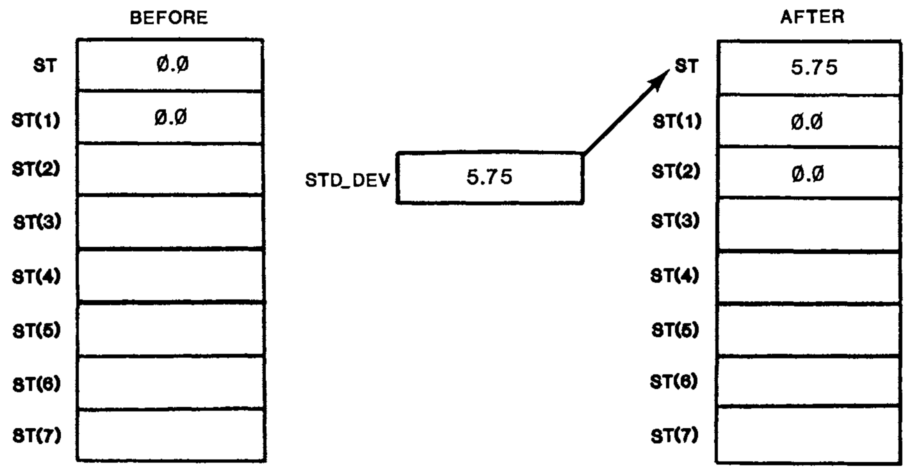
DATA TRANSFER INSTRUCTION

REAL TRANSFERS	
FLD	LOAD REAL
FST	STORE REAL
FSTP	STORE REAL AND POP
FXCH	EXCHANGE REGISTERS
INTEGER TRANSFERS	
FILD	INTEGER LOAD
FIST	INTEGER STORE
FISTP	INTEGER STORE AND POP
PACKED DECIMAL TRANSFERS	
FBLD	PACKED DECIMAL (BCD) LOAD
FBSTP	PACKED DECIMAL (BCD) STORE AND POP

- **THESE INSTRUCTIONS MOVE OPERANDS AMONG ELEMENTS OF THE REGISTER STACK, AND BETWEEN THE STACK TOP AND MEMORY**

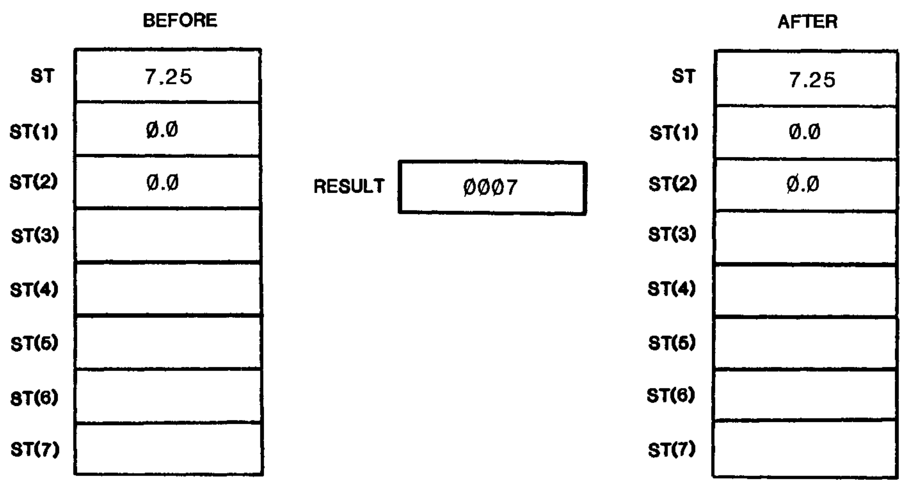
REGISTER LOAD

EX: FLD STD_DEV



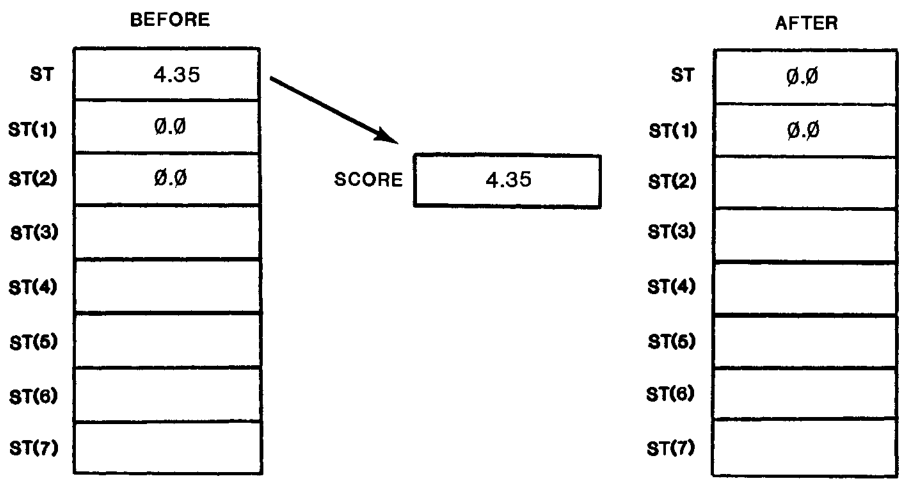
REGISTER STORE

EX: FIST RESULT



REGISTER STORE WITH POP

EX: FSTP SCORE



ARITHMETIC INSTRUCTIONS

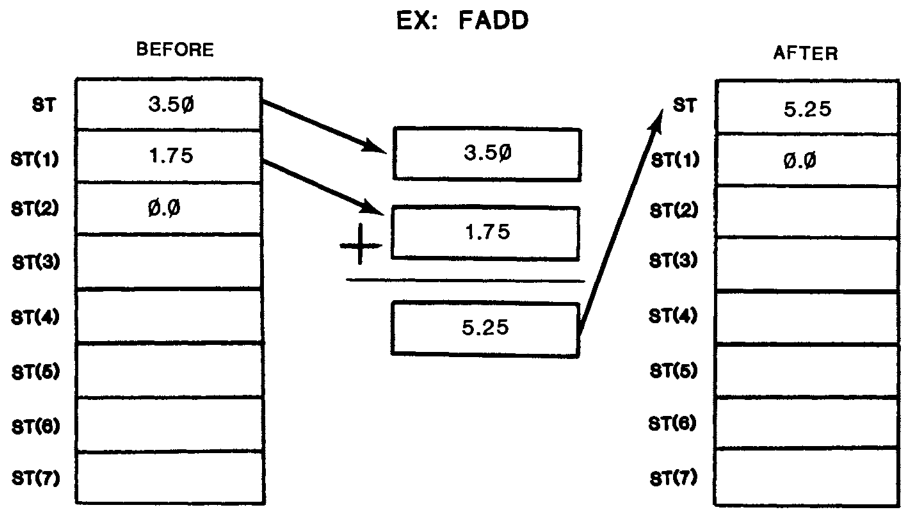
Addition	
FADD	Add real
FADDP	Add real and pop
FIADD	Integer add
Subtraction	
FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Integer subtract
FSUBR	Subtract real reversed
FSUBRP	Subtract real reversed and pop
FISUBR	Integer subtract reversed
Multiplication	
FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Integer multiply
Division	
FDIV	Divide real
FDIVP	Divide real and pop
FIDIV	Integer divide
FDIVR	Divide real reversed
FDIVRP	Divide real reversed and pop
FIDIVR	Integer divide reversed

Instruction Form	Mnemonic Form	Operand Forms destination, source	ASM-86 Example
Classical stack	*Fop*	{ST(1),ST}	FADD
Register	*Fop*	ST(i),ST or ST,ST(i)	FSUB ST,ST(3)
Register pop	*FopP*	ST(i),ST	FMULP ST(2),ST
Real memory	*Fop*	{ST,} short-real/long-real	FDIV AZIMUTH
Integer memory	*FIop*	{ST,} word-integer/short-integer	FIDIV N_ PULSES

NOTES: Braces { } surround *implicit* operands; these are not coded, and are shown here for information only.

op =	ADD	destination ← destination + source
	SUB	destination ← destination − source
	SUBR	destination ← source − destination
	MUL	destination ← destination • source
	DIV	destination ← destination ÷ source
	DIVR	destination ← source ÷ destination

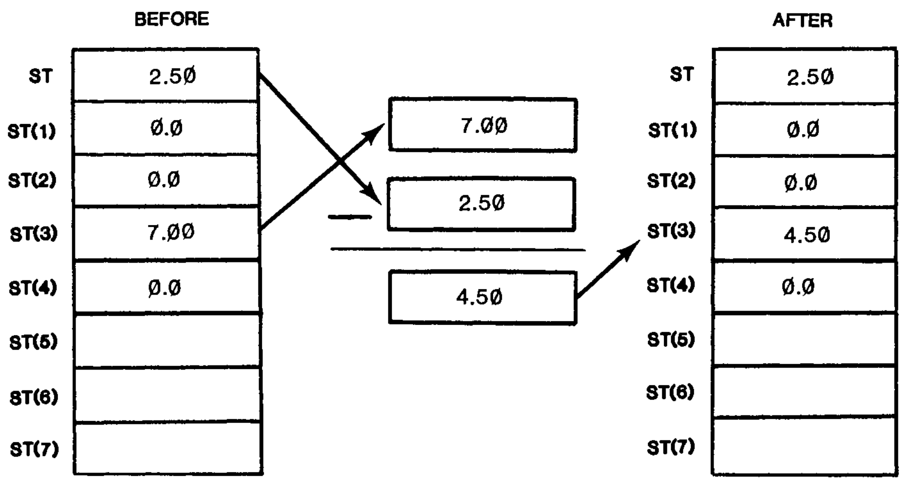
CLASSICAL STACK OPERATION



• NOTE: CLASSICAL STACK MODE INCLUDES A POP

REGISTER OPERATION

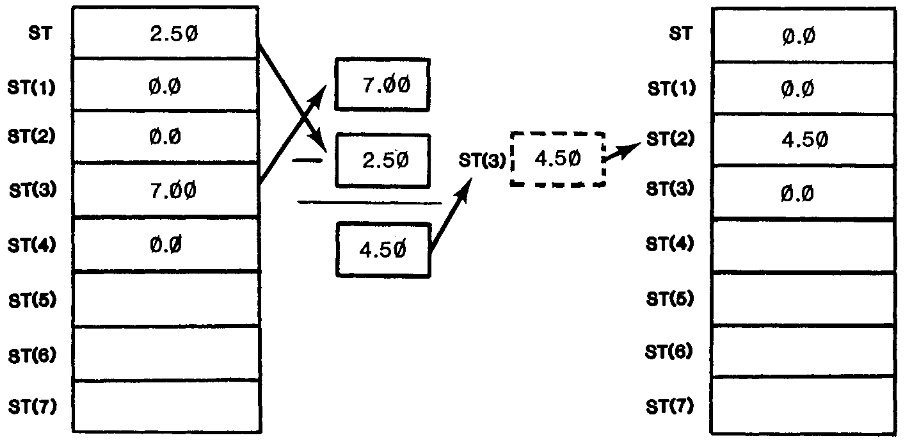
EX: FSUB ST(3),ST



NOTE: ONE OF THE OPERANDS MUST BE ST.
ST(i),ST OR ST,ST(i)

REGISTER OPERATION WITH POP

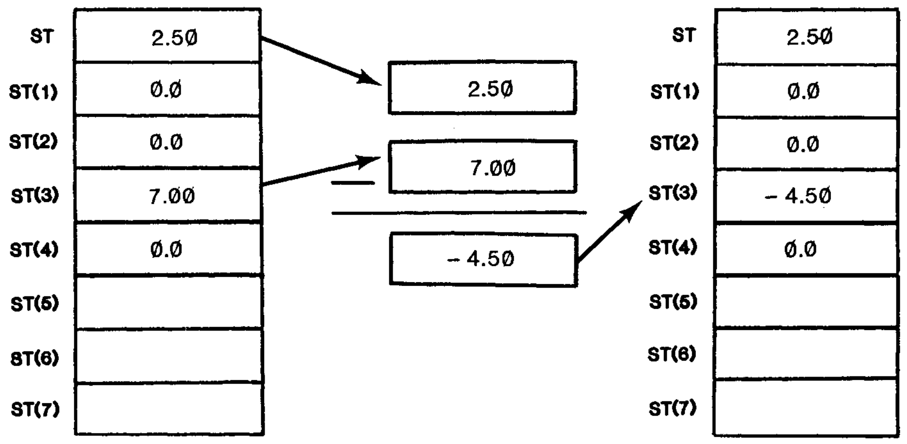
EX: FSUBP ST(3),ST



NOTE: OPERANDS MUST BE IN THIS FORM
ST(i),ST

REVERSED OPERATION

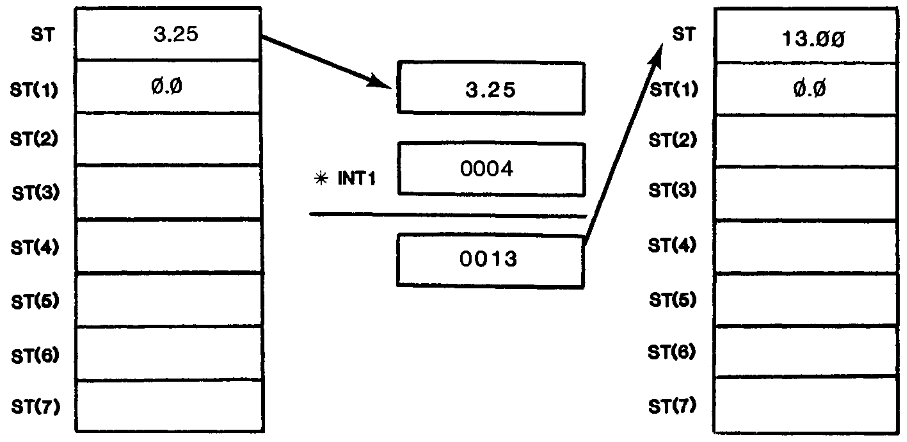
EX: FSUBR ST(3),ST



NOTE: THERE IS ALSO A REVERSED FORM OF THE INSTRUCTION FOR DIVISION

MEMORY OPERATION

EX: FIMUL INT1



OTHER ARITHMETIC INSTRUCTIONS

FSQRT	-	SQUARE ROOT
FSCALE	-	SCALE BY INTEGRAL POWERS OF TWO
FPREM	-	PARTIAL REMAINDER (MODULO REDUCTION)
FRNDINT	-	ROUND TO INTEGER
FXTRACT	-	EXTRACT EXPONENT AND SIGNIFICAND
FABS	-	ABSOLUTE VALUE
FCHS	-	CHANGE SIGN

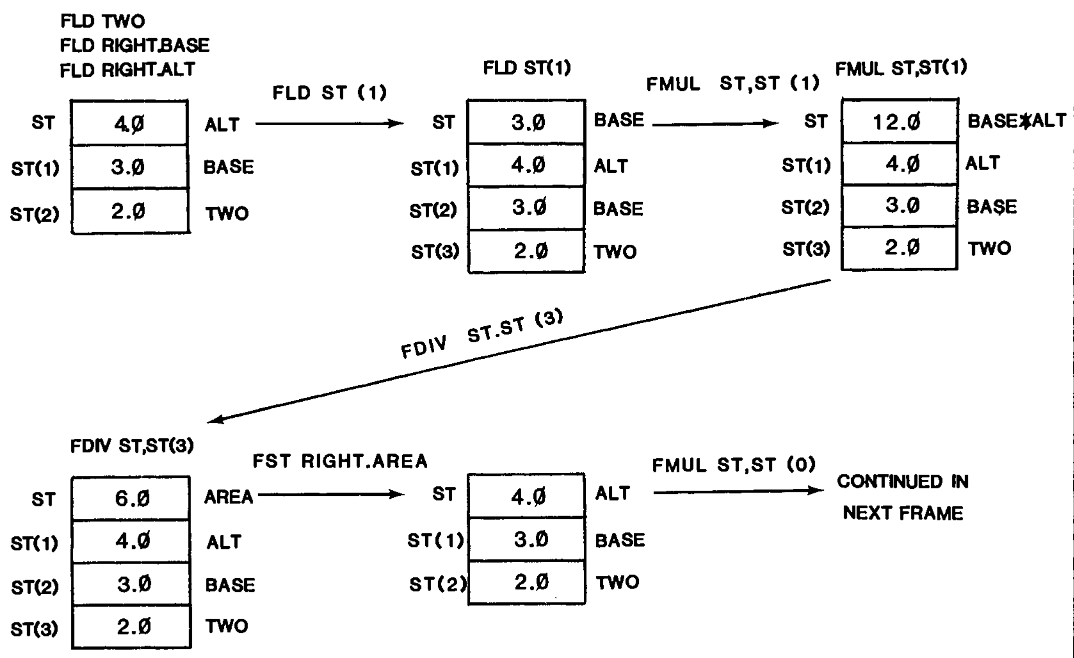
EXAMPLE

```
        NAME    PYTHAGORUS

        EXTRN   INIT87:FAR
;
;  Define a structure used to represent a right triangle.
;
TRIANGLE        STRUC
        BASE    DD      3.Ø     ; The DD memory allocation allows
        ALT     DD      4.Ø     ; enough  space for the variables
        HYP     DD      ?       ; to be defined in the SHORT REAL
        AREA    DD      ?       ; format.
TRIANGLE        ENDS


DATA    SEGMENT PUBLIC  'DATA'

RIGHT   TRIANGLE        <>      ; Allocate storage for one triangle.
TWO     DD              2.Ø     ; Define a real constant equal to 2.

DATA    ENDS


STACK   SEGMENT STACK  'STACK'

        DW      1ØØ DUP(?)
TOS     LABEL   WORD

STACK ENDS
```

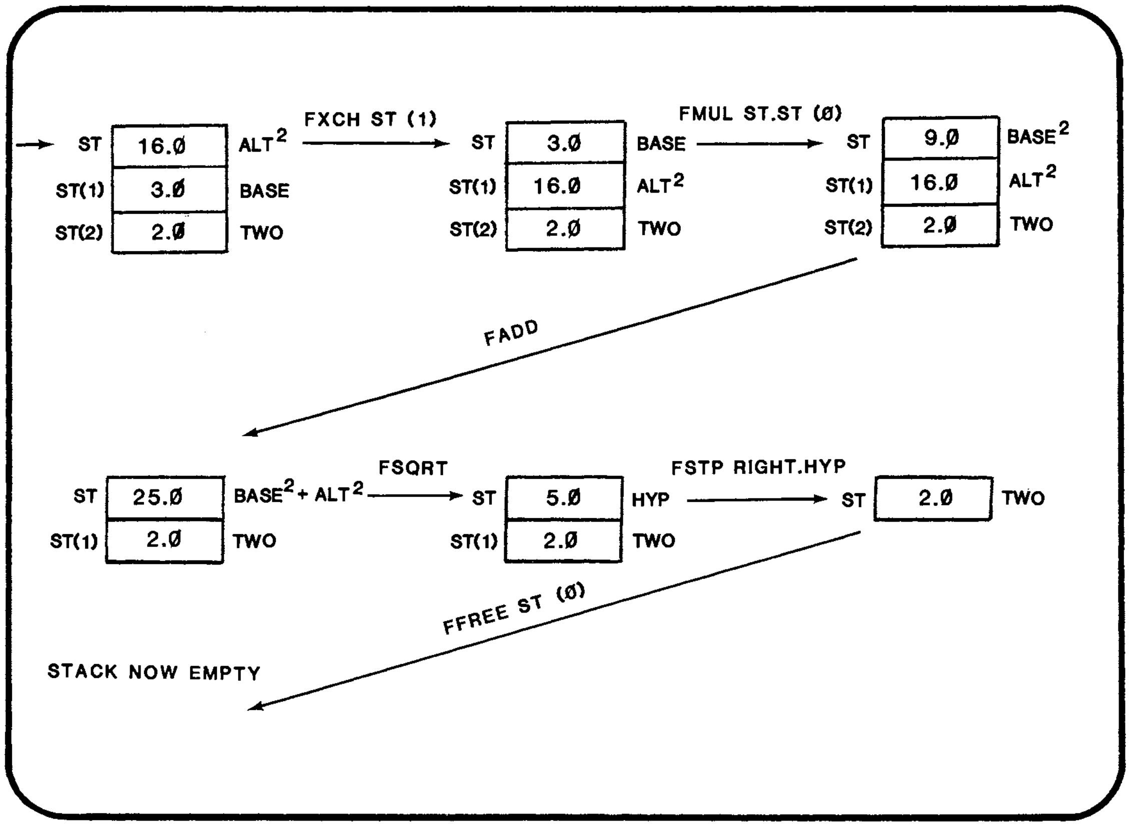
```
CODE    SEGMENT PUBLIC 'CODE'
        ASSUME  CS:CODE,DS:DATA,SS:STACK
;
; INITIALIZE 8Ø87
;
INIT:   CALL    INIT87          ; This routine is in a library.
                                ; It sets up the default environment
                                ; for the 8Ø87.
;
; PLACE INPUT OPERANDS ON 8Ø87 STACK
;
SETUP:  FLD     TWO             ; Put 2.Ø in STACK TOP (ST)
        FLD     RIGHT.BASE      ; ST <--BASE
        FLD     RIGHT.ALT       ; ST <--ALT
;
; CALCULATE AREA = (BASE*ALT)/2 AND STORE IN MEMORY
;
CALC:   FLD     ST(1)           ; Duplicate BASE in ST
        FMUL    ST,ST(1)        ; ST <--BASE * ALT
        FDIV    ST,ST(3)        ; ST <--ST/2
        FSTP    RIGHT.AREA      ; Store ST in AREA then discard
;
; CALCULATE HYPOTENUSE = ((BASE**2)+(ALT**2))**Ø.5
;
        FMUL    ST,ST(Ø)        ; Square ALT
        FXCH    ST(1)           ; Exchange ALT**2 and BASE
        FMUL    ST,ST(Ø)        ; Square BASE
        FADD                    ; ST <--BASE**2 + ALT**2
        FSQRT                   ; ST <--ST**Ø.5
FSTP    RIGHT.HYP               ; Store ST in HYP then discard
        FFREE   ST(Ø)           ; Clear out ST
                                ; Register STACK now empty
DONE:   HLT

CODE    ENDS
        END     INIT,DS:DATA,SS:STACK:TOS
```

12-19

REGISTER STACK USAGE



12-20



CLASS EXERCISE 12.1

WRITE A MATH PROGRAM THAT WILL PERFORM THE FOLLOWING OPERATION:

RESULT = ((A + B)/C)*D

DEFINE A,B,C AND D AS CONSTANTS USING THE SHORT REAL DATA TYPE. DEFINE RESULT AS A LONG REAL.

USE VALUES OF YOUR OWN CHOICE WHEN SETTING UP THE CONSTANTS.

TRANSCENDENTAL AND CONSTANT INSTRUCTIONS

FPTAN	PARTIAL TANGENT
FPATAN	PARTIAL ARCTANGENT
F2XM1	$2^X - 1$
FYL2X	$Y \cdot \log_2 X$
FYL2XP1	$Y \cdot \log_2 (X + 1)$

FLDZ	LOAD +0.0
FLD1	LOAD +1.0
FLDPI	LOAD π
FLDL2T	LOAD $\log_2 10$
FLDL2E	LOAD $\log_2 e$
FLDLG2	LOAD $\log_{10} 2$
FLDLN2	LOAD $\log_e 2$

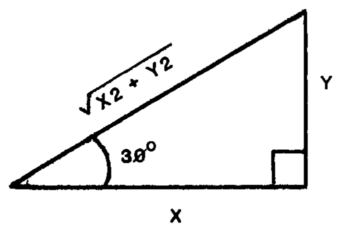
- THE TRANSCENDENTAL INSTRUCTIONS PERFORM THE TIME-CONSUMING CORE CALCULATIONS OF THE FOLLOWING FUNCTIONS:
  - TRIGONOMETRIC
  - INVERSE TRIGONOMETRIC
  - HYPERBOLIC
  - INVERSE HYPERBOLIC
  - LOGARITHMIC
  - EXPONENTIAL
- IN CONJUNCTION WITH THE CONSTANT AND ARITHMETIC INSTRUCTIONS, THE TRANSCENDENTAL INSTRUCTIONS CAN BE USED TO DERIVE ALL OF THE ABOVE LISTED FUNCTIONS

EXAMPLE

FPTAN

BEFORE

ST	Ø.5236	← $\pi/6$ RADIANS (3Ø°)
ST(1)	Ø.Ø	
ST(2)	Ø.Ø	
ST(3)		
ST(4)		
ST(5)		
ST(6)		
ST(7)		



AFTER

ST	1.732Ø	← X ($\sqrt{3}$)
ST(1)	1.Ø	← Y
ST(2)	Ø.Ø	
ST(3)		
ST(4)		
ST(5)		
ST(6)		
ST(7)		

$$\tan(\theta) = Y/X$$
$$\sin(\theta) = Y/\sqrt{X^2 + Y^2}$$
$$\cos(\theta) = X/\sqrt{X^2 + Y^2}$$

CLASS EXERCISE 12.2

WRITE A PROGRAM TO CALCULATE THE TANGENT, SINE AND COSINE OF A 60° ANGLE ($\pi/3$ RADIANS).

USE THE CONSTANT INSTRUCTIONS TO DERIVE $\pi/3$. STORE THE DESIRED RESULTS IN MEMORY USING A LONG REAL STORAGE FORMAT.

INSTRUCTION SYNCHRONIZATION

- NORMALLY, THE HOST CPU AND THE 8087 OPERATE ASYNCHRONOUSLY WITH RESPECT TO ONE ANOTHER. HOWEVER, THERE ARE TWO CASES WHEN IT IS NECESSARY TO SYNCHRONIZE THE PROCESSORS.
  1) AN 8087 INSTRUCTION MUST NOT BE STARTED IF THE 8087 IS BUSY EXECUTING A PREVIOUS INSTRUCTION
  2) THE HOST CPU MUST NOT ACCESS A MEMORY OPERAND BEING REFERENCED BY THE 8087 UNTIL THE 8087 HAS COMPLETED ITS CURRENT OPERATION
- THE FWAIT INSTRUCTION ALLOWS SOFTWARE TO SYNCHRONIZE THE TWO PROCESSORS, SUCH THAT THE HOST CPU WILL NOT EXECUTE ANY MORE INSTRUCTIONS UNTIL THE 8087 IS FINISHED WITH ITS CURRENT INSTRUCTION

- THE ASSEMBLER AUTOMATICALLY TAKES CARE OF THE FIRST CASE

EXAMPLE: FOR THE FOLLOWING TWO SOURCE STATEMENTS,

```
FMUL    ; MULTIPLY
FDIV    ; DIVIDE
```

THE ASSEMBLER PRODUCES FOUR MACHINE INSTRUCTIONS,

```
"FWAIT"
FMUL
"FWAIT"
FDIV
```

- THE FWAIT INSTRUCTIONS INSURE THAT ANY PREVIOUS 8087 INSTRUCTION RUNS TO COMPLETION, BEFORE A NEW 8087 INSTRUCTION IS STARTED

- TO SATISFY THE SECOND CASE, THE PROGRAMMER SHOULD EXPLICITLY CODE THE FWAIT INSTRUCTION IMMEDIATELY BEFORE A CPU INSTRUCTION THAT ACCESSES A MEMORY OPERAND READ OR WRITTEN BY A PREVIOUS 8087 INSTRUCTION

```
EXAMPLE:   FIST     VAR_1          ; STORE INTEGER
           FWAIT                   ; WAIT FOR 8087
           MOV      AX,VAR_1
```

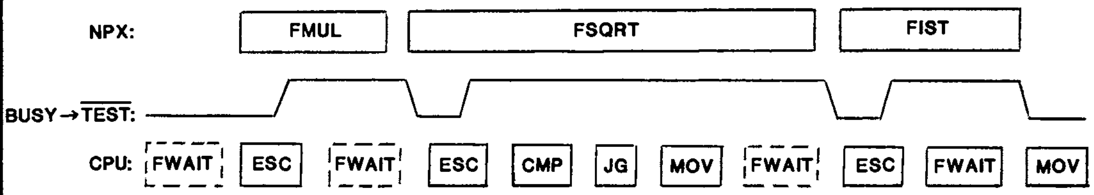
- THE FWAIT OPCODE CAUSES THE ASSSEMBLER TO CREATE A CPU WAIT INSTRUCTION THAT CAN BE ELIMINATED AT LINK TIME IF THE PROGRAM IS TO RUN ON AN 8087 EMULATOR. THE WAIT OPCODE DOES NOT PROVIDE THIS FLEXIBILITY.

FWAIT – CAN BE ELIMINATED IF EMULATOR USED

WAIT – FIXED WITHIN PROGRAM. TEST PIN MUST BE IMPLEMENTED.

PROCESSOR SYNCHRONIZATION

```
           ;ASSUME 8087 REGISTER STACK IS LOADED WITH OPERANDS,
           ;       NEU IS NOT BUSY,
           ;       AND THAT 'ALPHA' AND 'BETA' ARE WORD
           ;       INTEGERS.
           ;
           FMUL                     ;MULTIPLY TOP STACK
                                    ;ELEMENTS
           FSQRT                    ;SQUARE ROOT OF PRODUCT
           CMP     ALPHA,100        ;ALPHA   100?
           JG      CONTINUE         ;YES, LEAVE UNALTERED
           MOV     ALPHA,100        ;NO, SET TO 100
CONTINUE:  FIST    BETA             ;STORE ROOT AS INTEGER WORD
           FWAIT                    ;WAIT FOR 8087
                                    ;STORE OF BETA
           MOV     AX,BETA          ;PROCEED TO PROCESS BETA
```



NOTES:

- FWAIT = ASSEMBLER-GENERATED INSTRUCTION

WHERE TO FIND MORE INFORMATION

APPLICATION NOTE AP-113 – GETTING STARTED WITH THE NUMERIC DATA PROCESSOR

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)
CHAPTER 6 – THE 8087 NUMERIC PROCESSOR EXTENSION

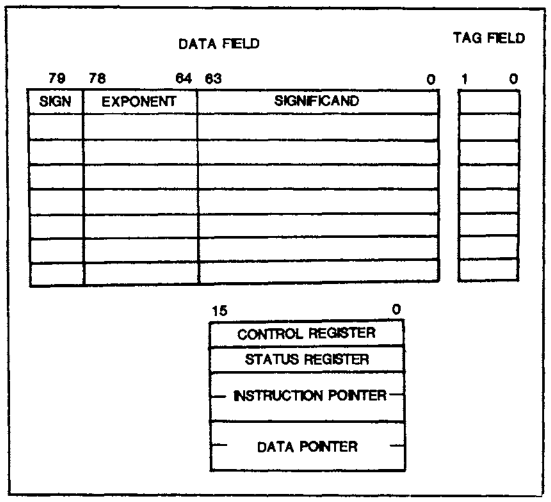
ASM86 LANGUAGE REFERENCE MANUAL
CHAPTER 6 – THE 8086/8087/8088 INSTRUCTION N SET

CHAPTER 13

MORE ON THE 8087

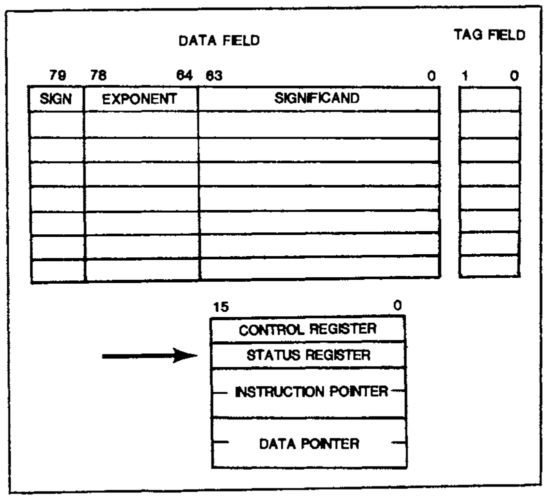
- STATUS WORD
- LOGICAL INSTRUCTIONS
- CONTROL WORD
- INITIALIZING THE 8087
- PROCESSOR CONTROL INSTRUCTIONS

THE REGISTER STACK



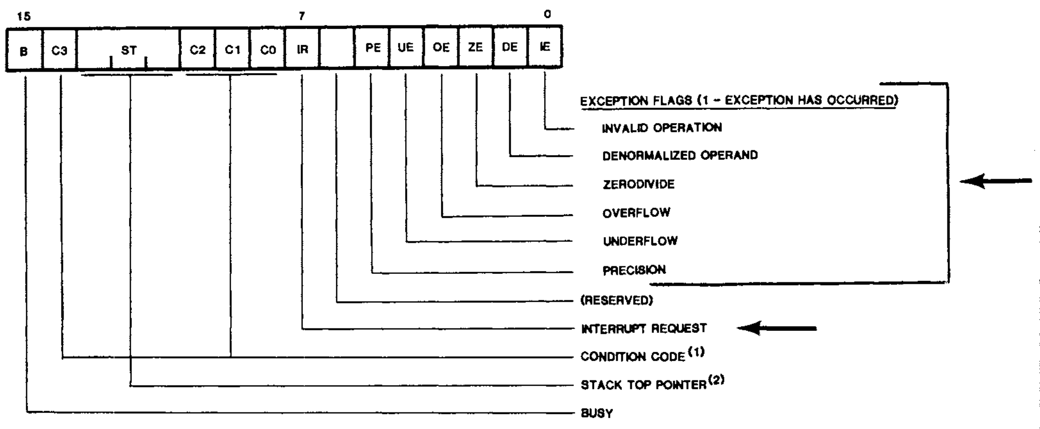
- THE REGISTERS ARE ORGANIZED AS AN 8 ELEMENT STACK
- THE STACK TOP POINTER WITHIN THE STATUS WORD IDENTIFIES THE CURRENT TOP OF STACK
- THE TAG WORD IDENTIFIES THE CONTENTS OF EACH REGISTER AS BEING VALID OR INVALID

STATUS WORD



- THE STATUS WORD REFLECTS THE OVERALL CONDITION OF THE 8087
- THE STATUS WORD MAY BE EXAMINED BY STORING IT INTO MEMORY WITH AN NDP INSTRUCTION AND THEN INSPECTING IT WITH CPU CODE

STATUS WORD (CONT)



INTERRUPT REQUEST	- USED TO RECORD A PENDING INTERRUPT
EXCEPTION FLAGS	- USED TO IDENTIFY THE TYPE OF EXCEPTION(S) THAT HAVE OCCURRED SINCE THE FLAGS WERE LAST INITIALIZED

EXCEPTIONS

INVALID OPERATION
- ATTEMPT TO LOAD A REGISTER THAT IS NOT EMPTY
- ATTEMPT TO POP AN OPERAND FROM A REGISTER THAT IS EMPTY
- OPERAND IS A NAN (NOT A NUMBER)
- OPERANDS CAUSE OPERATION TO BE INDETERMINATE (0/0, $\sqrt{-\text{NUMBER}}$)

DENORMALIZED OPERAND
- ATTEMPT TO USE AN OPERAND THAT IS NOT NORMALIZED

ZERODIVIDE
- ATTEMPT TO DIVIDE BY ZERO

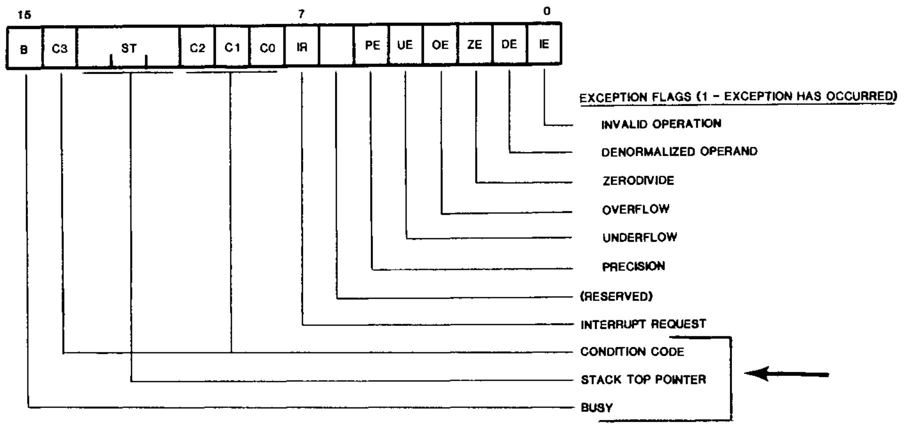
OVERFLOW
- RESULT TOO LARGE FOR DESTINATION FORMAT

UNDERFLOW
- RESULT TOO SMALL FOR DESTINATION FORMAT

PRECISION
- RESULT NOT EXACTLY REPRESENTABLE IN DESTINATION FORMAT
- 8087 ROUNDS RESULT

NOTE: EXCEPTION BITS ARE "STICKY" AND CAN BE CLEARED ONLY BY THE FCLEX (CLEAR EXCEPTIONS) INSTRUCTION

STATUS WORD (CONT)



BUSY	– USED TO IDENTIFY IF THE 8087 IS EXECUTING AN INSTRUCTION
STACK TOP POINTER	– USED TO IDENTIFY THE REGISTER THAT IS THE CURRENT STACK TOP
CONDITION CODE	– USED TO POST RESULTS OF COMPARE/EXAMINE TYPE INSTRUCTIONS AND ALSO THE FPREM (PARTIAL REMAINDER) INSTRUCTION

COMPARISON INSTRUCTIONS

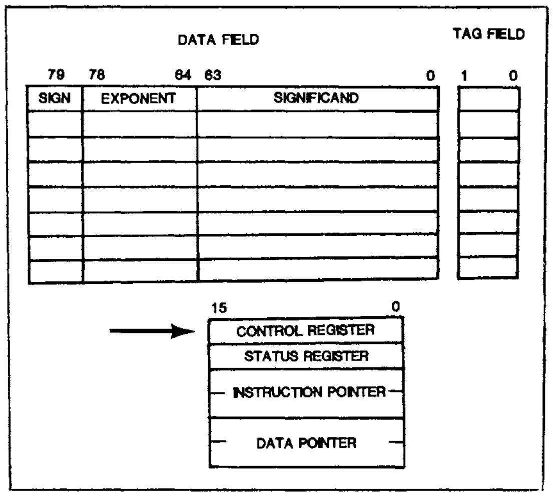
FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Integer compare
FICOMP	Integer compare and pop
FTST	Test
FXAM	Examine

CONDITION CODE INTERPRETATION

Instruction	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	X	X	0	A > B
	0	X	X	1	A < B
See Note	1	X	X	0	A = B
	1	X	X	1	A ? B (not comparable)
Examine	0	0	0	0	Valid, positive, unnormalized
	0	0	0	1	Invalid, positive, exponent ≠ 0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent ≠ 0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

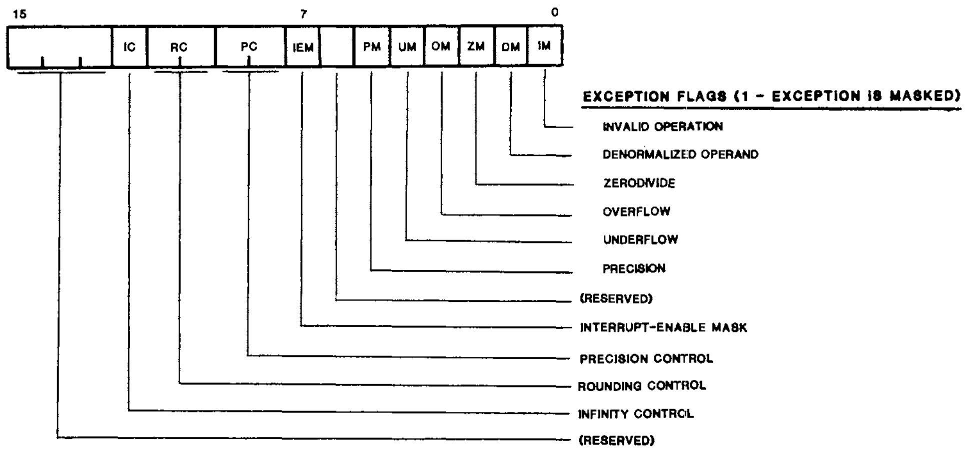
NOTE: COMPARE INSTRUCTIONS – A = ST, B = SOURCE
TEST INSTRUCTION – A = ST, B = Ø

CONTROL WORD



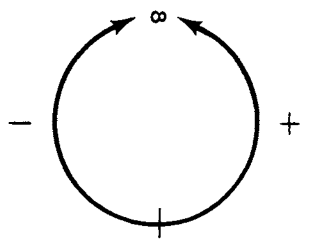
- THE CONTROL WORD IS USED TO CONFIGURE THE OPERATING MODE OF THE 8087
- THE CONTROL WORD IS LOADED FROM MEMORY BY THE FLDCW (LOAD CONTROL WORD) INSTRUCTION

CONTROL WORD (CONT)

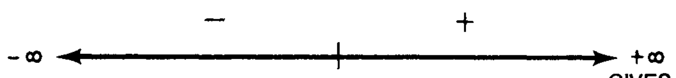


Interrupt-Enable Mask:
  0 = Interrupts Enabled
  1 = Interrupts Disabled (Masked)
Precision Control:
  00 = 24 bits
  01 = (reserved)
  10 = 53 bits
  11 = 64 bits
Rounding Control:
  00 = Round to Nearest or Even
  01 = Round Down (toward $-\infty$)
  10 = Round Up (toward $+\infty$)
  11 = Chop (Truncate Toward Zero)
Infinity Control:
  0 = Projective
  1 = Affine

INFINITY CONTROL



PROJECTIVE CLOSURE ← DEFAULT – RECOMMENDED FOR MOST COMPUTATIONS

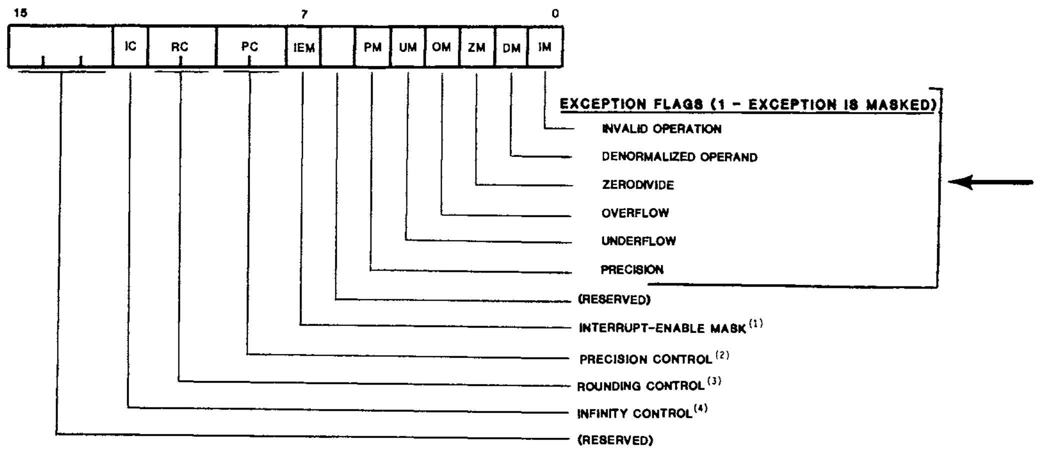


AFFINE CLOSURE ← GIVES MORE INFORMATION, BUT THERE ARE OCCASIONS WHEN THE SIGN REPRESENTS MISINFORMATION

$+\emptyset = -\emptyset$

$1/+\emptyset = 1/-\emptyset$

CONTROL WORD (CONT)

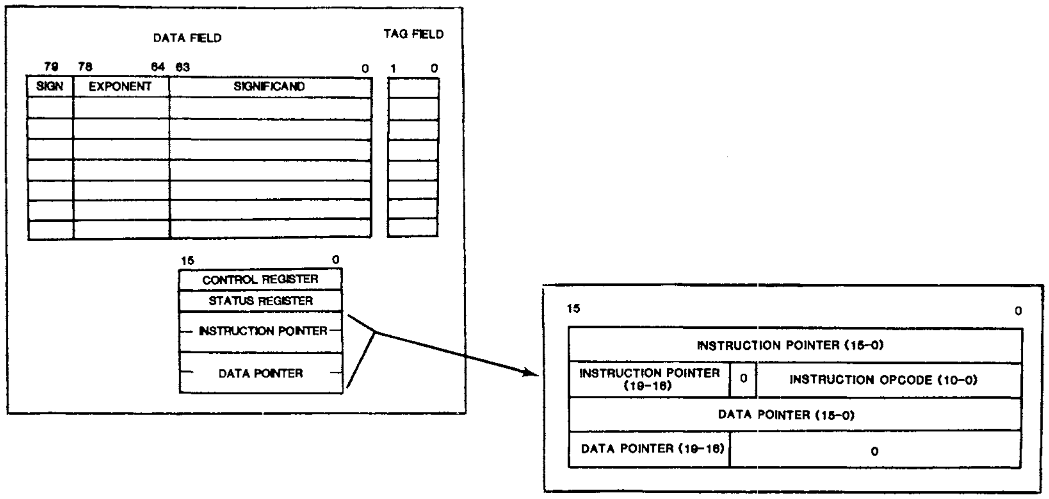


EXCEPTION MASKS – USED TO DETERMINE WHETHER THE 8087 SHOULD FIELD AN EXCEPTION ON ITS OWN OR SEND AN INTERRUPT REQUEST TO THE HOST CPU

NOTE: A MASKED RESPONSE PRODUCES A RESULT AND THEN PROCEEDS WITH THE CURRENT INSTRUCTION

AN UNMASKED RESPONSE TRAPS TO USER SOFTWARE BY INTERRUPTING THE HOST CPU

EXCEPTION POINTERS



- STORE IN MEMORY WITH FSTENV (STORE ENVIRONMENT) INSTRUCTION
- PROVIDED FOR USER WRITTEN EXCEPTION HANDLERS
- WHENEVER THE 8087 EXECUTES AN INSTRUCTION, IT SAVES THE INSTRUCTION ADDRESS, THE OPERAND ADDRESS (IF PRESENT) AND THE INSTRUCTION OPCODE

13-11

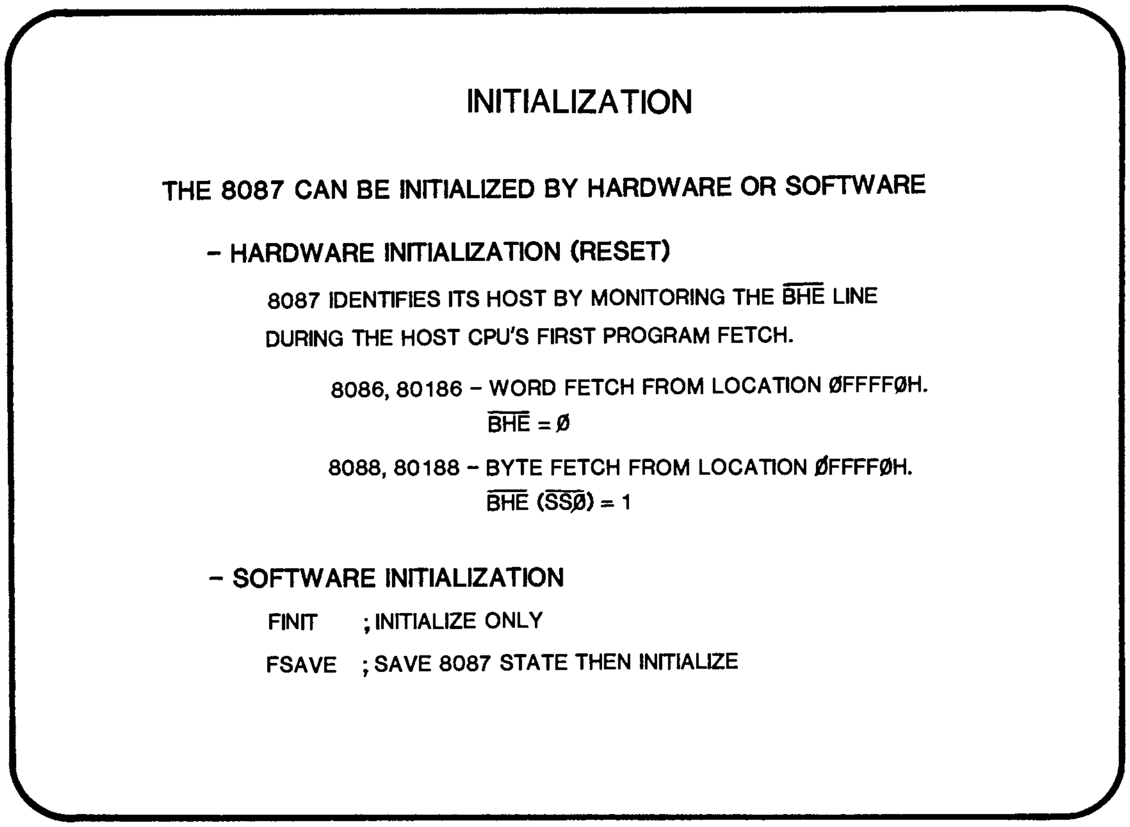
HOW WOULD THE AVERAGE USER CONFIGURE THE 8087?

1. USE THE DEFAULT CONFIGURATION WITH ALL EXCEPTIONS MASKED. THE 8087 WILL GENERATE A DEFAULT RESULT IF AN ERROR OCCURS.

2. UNMASK THE INVALILD OPERATION EXCEPTION, AND KILL THE COMPUTATIONAL ALGORITHM IF AN INTERRUPT OCCURS.

3. UNMASK ALL THE EXCEPTIONS, AND KILL THE COMPUTATIONAL ALGORITHM IF AN INTERRUPT OCCURS.

NOTE: THE 8087 IS A VERY FLEXIBLE MATH PROCESSOR. HOWEVER, MOST OF THIS FLEXIBILITY WOULD BE USED ONLY IF VERY SERIOUS NUMERIC ANALYSIS IS REQUIRED



INITIALIZATION

THE 8087 CAN BE INITIALIZED BY HARDWARE OR SOFTWARE

- HARDWARE INITIALIZATION (RESET)

  8087 IDENTIFIES ITS HOST BY MONITORING THE $\overline{BHE}$ LINE DURING THE HOST CPU'S FIRST PROGRAM FETCH.

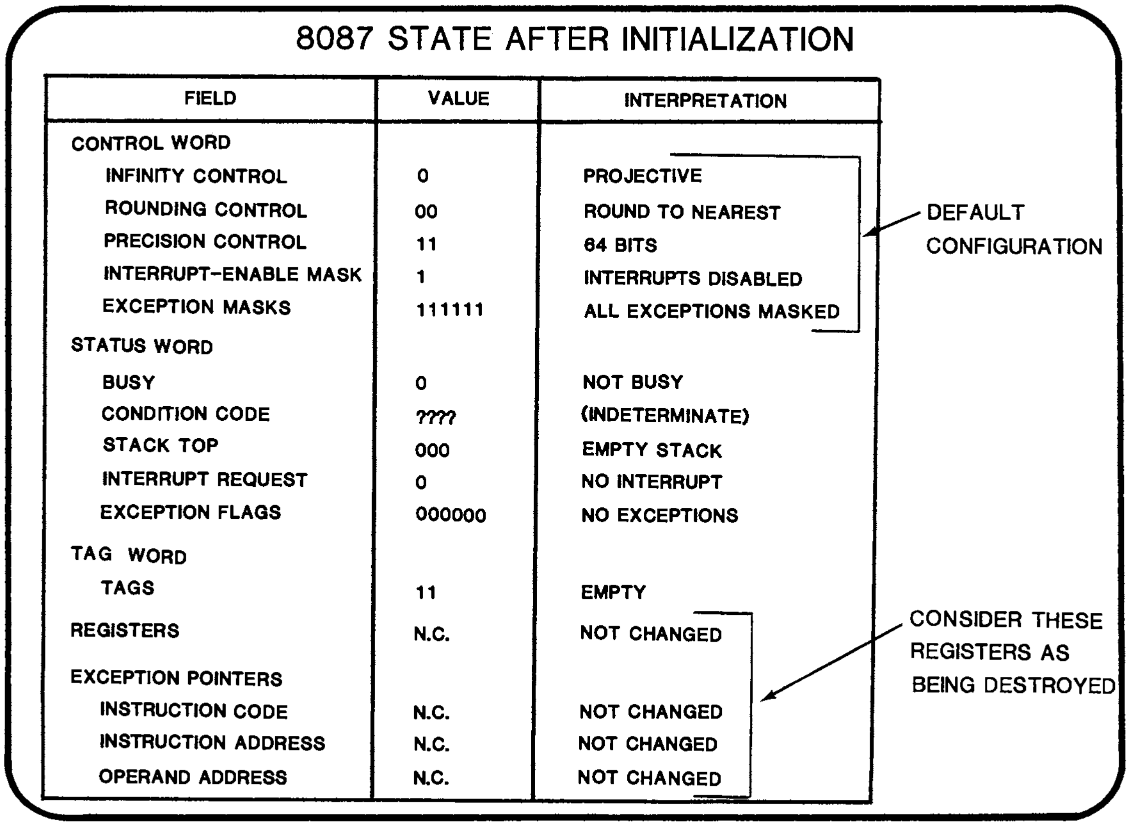
  8086, 80186 – WORD FETCH FROM LOCATION ØFFFFØH.
  $\overline{BHE}$ = Ø

  8088, 80188 – BYTE FETCH FROM LOCATION ØFFFFØH.
  $\overline{BHE}$ ($\overline{SSØ}$) = 1

- SOFTWARE INITIALIZATION

  FINIT ; INITIALIZE ONLY

  FSAVE ; SAVE 8087 STATE THEN INITIALIZE



8087 STATE AFTER INITIALIZATION

FIELD	VALUE	INTERPRETATION
CONTROL WORD		
INFINITY CONTROL	0	PROJECTIVE
ROUNDING CONTROL	00	ROUND TO NEAREST
PRECISION CONTROL	11	64 BITS
INTERRUPT-ENABLE MASK	1	INTERRUPTS DISABLED
EXCEPTION MASKS	111111	ALL EXCEPTIONS MASKED
STATUS WORD		
BUSY	0	NOT BUSY
CONDITION CODE	????	(INDETERMINATE)
STACK TOP	000	EMPTY STACK
INTERRUPT REQUEST	0	NO INTERRUPT
EXCEPTION FLAGS	000000	NO EXCEPTIONS
TAG WORD		
TAGS	11	EMPTY
REGISTERS	N.C.	NOT CHANGED
EXCEPTION POINTERS		
INSTRUCTION CODE	N.C.	NOT CHANGED
INSTRUCTION ADDRESS	N.C.	NOT CHANGED
OPERAND ADDRESS	N.C.	NOT CHANGED

DEFAULT CONFIGURATION

CONSIDER THESE REGISTERS AS BEING DESTROYED

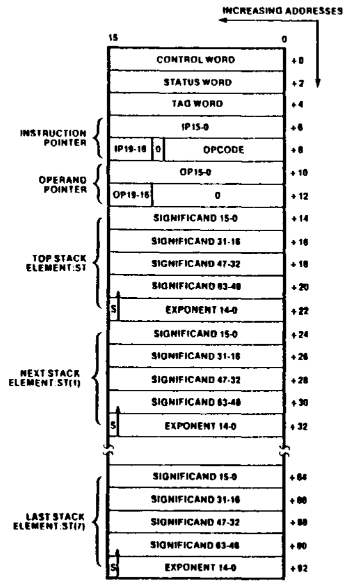
PROCESSOR CONTROL INSTRUCTIONS

FINIT/FNINIT	INITIALIZE PROCESSOR
FDISI/FNDISI	DISABLE INTERRUPTS
FENI/FNENI	ENABLE INTERRUPTS
FLDCW	LOAD CONTROL WORD
FSTCW/FNSTCW	STORE CONTROL WORD
FSTSW / FNSTSW	STORE STATUS WORD
FCLEX/FNCLEX	CLEAR EXCEPTIONS
FSTENV/FNSTENV	STORE ENVIRONMENT
FLDENV	LOAD ENVIRONMENT
FSAVE/FNSAVE	SAVE STATE
FRSTOR	RESTORE STATE
FINCSTP	INCREMENT STACK POINTER
FDECSTP	DECREMENT STACK POINTER
FFREE	FREE REGISTER
FNOP	NO OPERATION
FWAIT	CPU WAIT

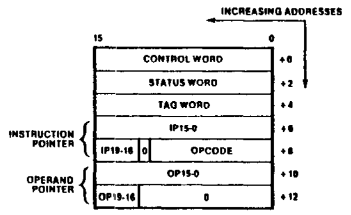
- THE OPCODES, DISTINGUISHED BY A SECOND CHARACTER OF "N", INSTRUCT THE ASSEMBLER NOT TO PREFIX THE INSTRUCTION WITH A CPU WAIT INSTRUCTION. INSTEAD, A CPU NOP IS USED

MEMORY REQUIREMENTS FOR STORING THE 8087's STATE AND ENVIRONMENT

STATE



ENVIRONMENT



WHERE TO FIND MORE INFORMATION

APPLICATION NOTE AP-113 - GETTING STARTED WITH THE NUMERIC DATA PROCESSOR

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)
CHAPTER 6 - THE 8087 NUMERIC PROCESSOR EXTENSION

ASM86 LANGUAGE REFERENCE MANUAL
CHAPTER 6 - THE 8086/8087/8088 INSTRUCTION N SET

CHAPTER 14

OVERVIEW OF THE 8087 SUPPORT LIBRARIES

- INTERFACE LIBRARIES
- DECIMAL CONVERSION LIBRARY
- COMMON ELEMENTARY FUNCTION LIBRARY
- ERROR HANDLER LIBRARY

8087 INTERFACE LIBRARIES

FULL EMULATOR

E8087	-	EMULATOR
E8087.LIB	-	INTERFACE LIBRARY

PARTIAL EMULATOR (PL/M-86 ONLY)

PE8087	-	PARTIAL EMULATOR
E8087.LIB	-	INTERFACE LIBRARY

8087 CHIP

8087.LIB	-	INTERFACE LIBRARY

LINKING TO LIBRARIES

(PARTIAL) EMULATOR

- LINKER REMOVES ALL FWAIT INSTRUCTIONS, INSERTS "CALLS" TO EMULATOR ROUTINES VIA SOFTWARE INTERRUPTS.
- INIT87 SETS UP INTERRUPT VECTORS TO INTERFACE TO EMULATOR

8087.LIB

- 8087 INSTRUCTIONS LEFT INTACT
- INIT87 INITIALIZES 8087, MASKS ALL EXCEPTIONS

DECIMAL CONVERSION LIBRARY

- DCON87.LIB
  - CONVERT BETWEEN DIFFERENT REAL FORMATS
  - CONVERT FROM DECIMAL STRING TO BINARY FORMAT
  - CONVERT FROM BINARY FORMAT TO DECIMAL STRING

NOTES: 1) SUPPORTS PL/M-86 MEDIUM AND LARGE MODELS.
2) MUST USE FULL EMULATOR OR ACTUAL CHIP (8087).

COMMON ELEMENTARY FUNCTION LIBRARY

- CEL87.LIB
  - ROUNDING AND TRUNCATION FUNCTIONS
  - LOGARITHMIC AND EXPONENTIAL FUNCTIONS
  - TRIGONOMETRIC AND HYPERBOLIC FUNCTIONS

NOTES: 1) SUPPORTS PL/M-86 MEDIUM AND LARGE MODELS
2) MUST USE FULL EMULATOR OR ACTUAL CHIP (8087)

ERROR HANDLER LIBRARY

- EH87.LIB
  - CONTAINS FIVE UTILITY PROCEDURES FOR WRITING YOUR OWN EXCEPTION HANDLERS

NOTES: 1) SUPPORTS PL/M-86 MEDIUM AND LARGE MODELS
2) MUST USE FULL EMULATOR OR ACTUAL CHIP (8087)

LINKAGE EXAMPLES

- FULL EMULATOR
  - RUN LINK86 :F1:MYPROG.OBJ, E8087.LIB, E8087
- DCON87, CEL87 AND CHIP
  - RUN LINK86 :F1:MYPROG.OBJ, DCON87.LIB, CEL87.LIB, 8087.LIB
- CEL87, EH87 AND EMULATOR
  - RUN LINK86 :F1:MYPROG.OBJ, CEL87, EH87, E8087.LIB, E8087

WHERE TO FIND MORE INFORMATION

8087 SUPPORT LIBRARY REFERENCE MANUAL.

DAY 4 OBJECTIVES

BY THE TIME YOU FINISH TODAY YOU WILL:

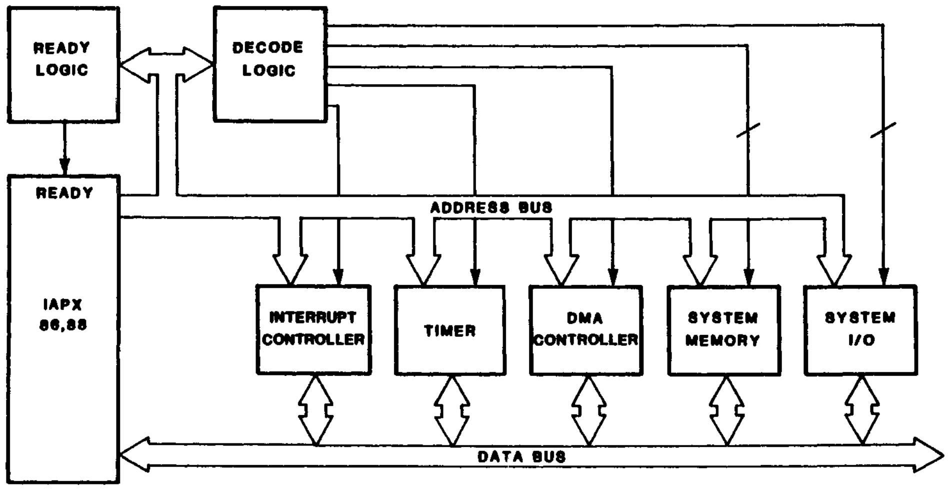
- DEINE THE ADVANTAGES OF THE 80186/188
- USE THE ENHANCED INSTRUCTION SET OF THE 80186/188
- DEFINE THE FORMAT OF THE CHIP SELECT LINES AND THE USE OF WAIT STATES IN AN 80186. PROGRAM THE CHIP SELECT LINES TO MEET A REQUIREMENT
- DEFINE THE MODES OF OPERATION OF THE THREE TIMERS ON THE 80186 AND PROGRAM THEM TO OPERATE IN A REQUIRED MODE
- DEFINE THE OPERATIONAL MODES OF THE TWO DMA CHANNELS ON THE 80186 AND PROGRAM THEM TO OPERATE IN A REQUIRED MODE

CHAPTER 15

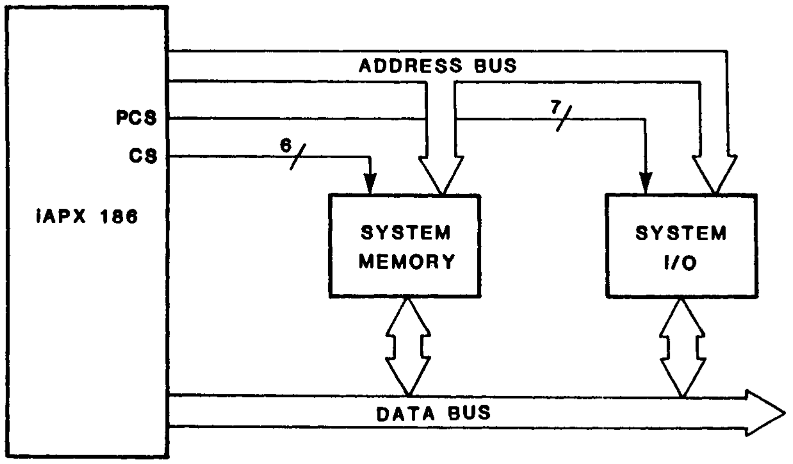
INTRODUCTION TO THE 186

- DESCRIPTION
- ENHANCEMENTS
- NEW INSTRUCTIONS

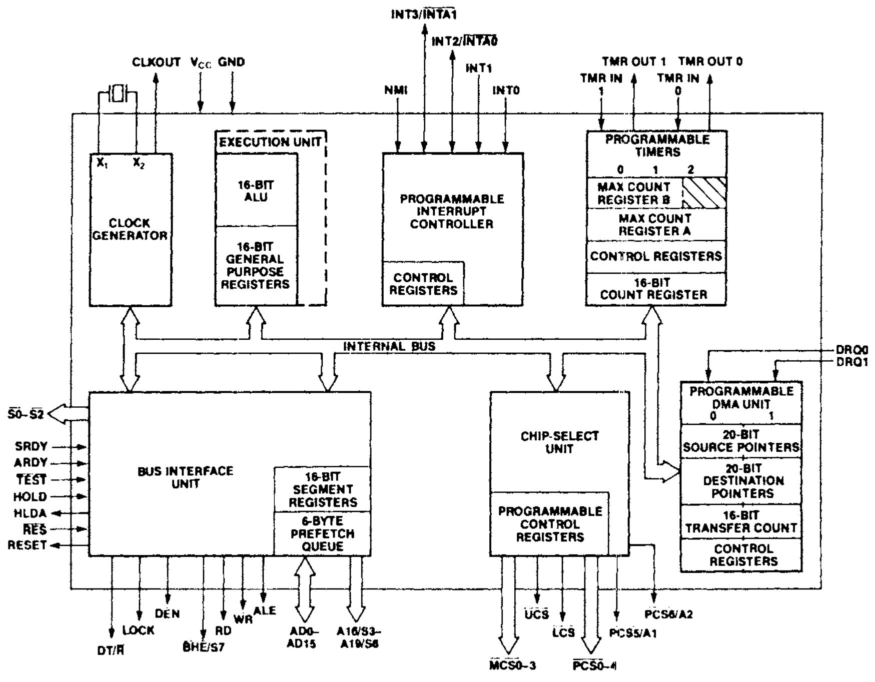
TYPICAL iAPX 86,88 SYSTEM



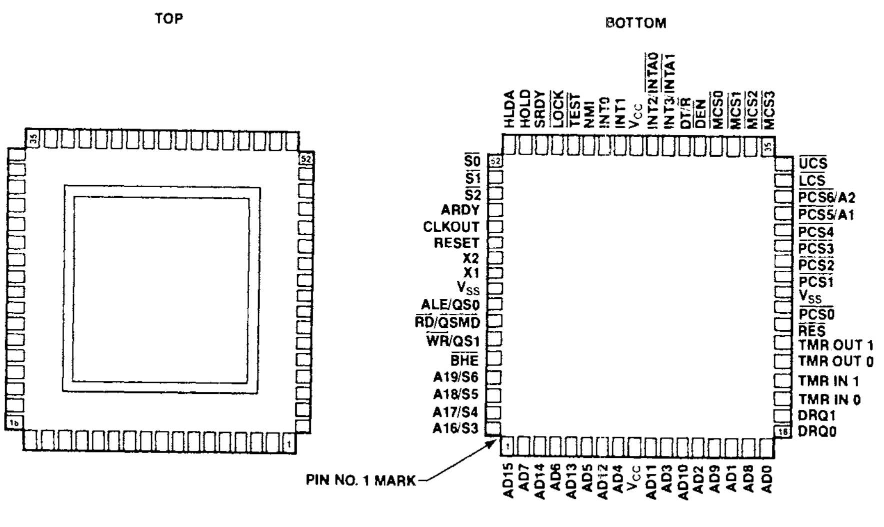
SAME SYSTEM USING THE iAPX 186, 188



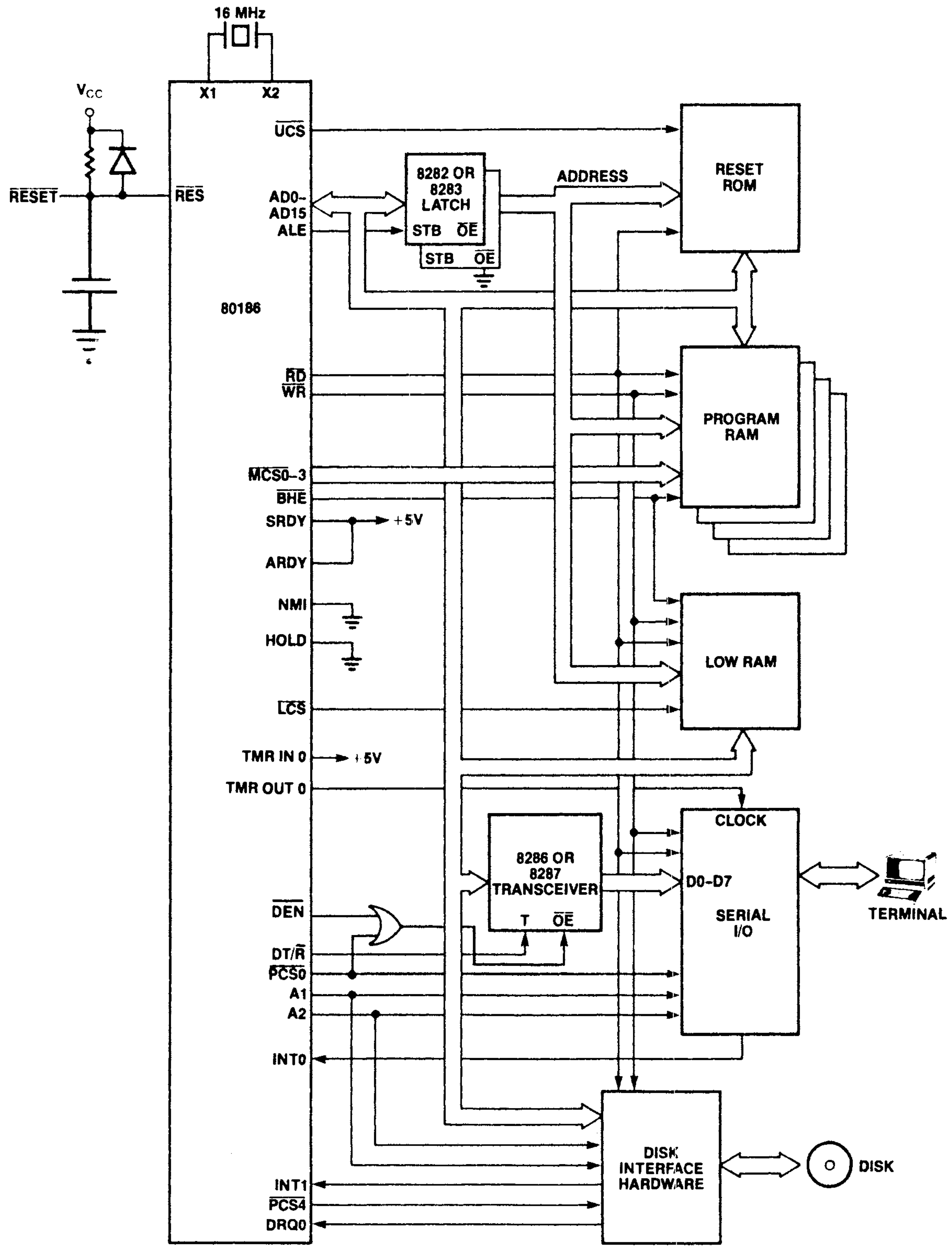
iAPX 186, 188 BLOCK DIAGRAM



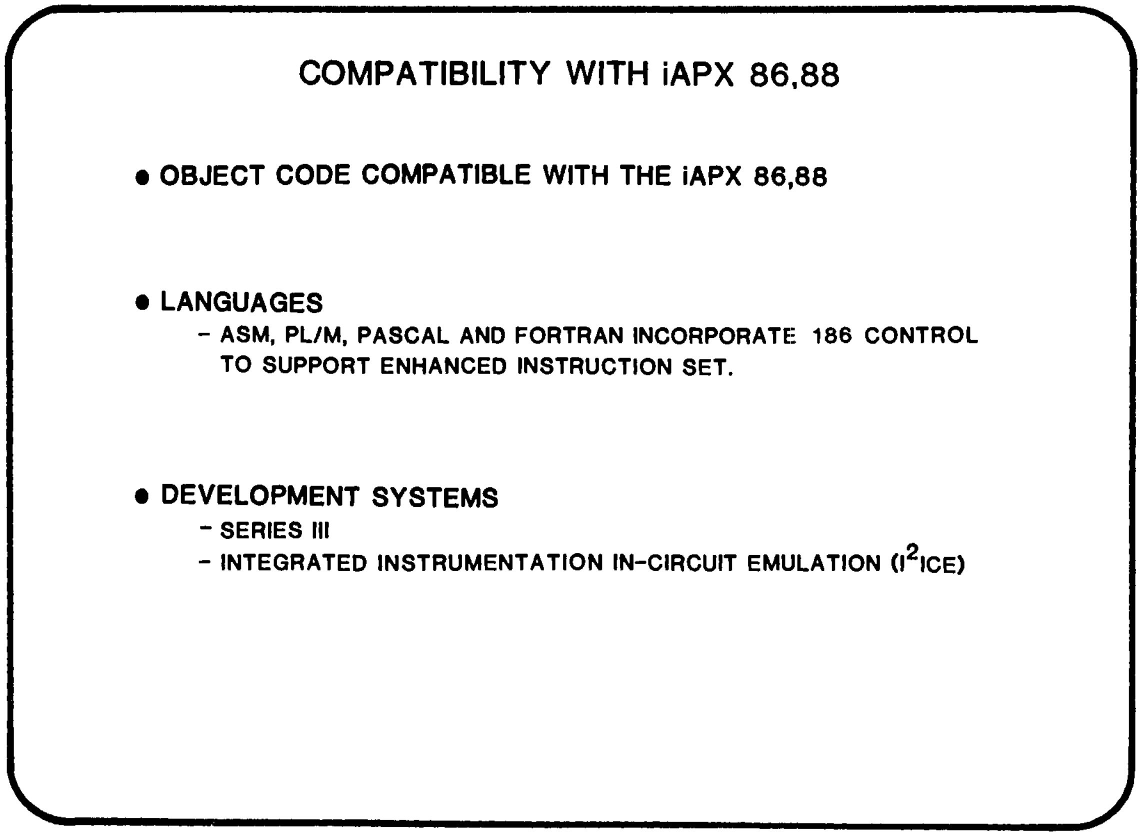
iAPX 186, 188 PINOUT



TYPICAL iAPX 186, 188 COMPUTER SYSTEM



● $\overline{\text{BHE}}$ NOT IMPLEMENTED ON iAPX 188



COMPATIBILITY WITH iAPX 86,88

- **OBJECT CODE COMPATIBLE WITH THE iAPX 86,88**

- **LANGUAGES**
  - ASM, PL/M, PASCAL AND FORTRAN INCORPORATE 186 CONTROL TO SUPPORT ENHANCED INSTRUCTION SET.

- **DEVELOPMENT SYSTEMS**
  - SERIES III
  - INTEGRATED INSTRUMENTATION IN-CIRCUIT EMULATION (I^2ICE)

iAPX 186, 188 RELATIVE PERFORMANCE (8 MHz STANDARD CLOCK RATE)

Instruction	8086 (5MHz)	8086-2 (8MHz)
MOV REG TO MEM	2.0–2.9X	1.2–1.8X
ADD MEM TO REG MUL REG 16 DIV REG 16	2.0–2.9X >5.4X >6.1X	1.2–1.8X >3.4X >3.8X
MULTIPLE (4-BITS) SHIFT/ROTATE MEMORY	3.1–3.7X	1.95–2.3X
CONDITIONAL JUMP	1.9X	1.2X
BLOCK MOVE (100 BYTES)	3.4X	2.1X

OVERALL: 2x PERFORMANCE OF 5 MHz iAPX 86
1.3x PERFORMANCE OF 8 MHz iAPX 86

NOTE: SAME COMPARISONS APPLY TO iAPX 188 and iAPX 88

iAPX 186, 188 CPU ENHANCEMENTS

- EFFECTIVE ADDRESS CALCULATIONS(EA)
  - CALCULATION OF BASE + DISPLACEMENT + INDEX
  - 3 – 6X FASTER IN THE iAPX 186,188
- 16-BIT INTEGER MULTIPLY AND DIVIDE HARDWARE
  - 3X THE 8MHz iAPX 86, 88
- STRING MOVE
  - 2X THE 8MHz iAPX 86 ,88
- TRAP ON UNUSED OPCODES
  - PRE-DEFINED INTERRUPT VECTOR
- MULTIPLE-BIT SHIFT/ROTATE SPEED-UP
  - 1.5 – 2.5X THE 8MHz iAPX 86,88
- NEW INSTRUCTIONS

NEW iAPX 186, 188 INSTRUCTIONS

- SHIFT/ROTATE IMMEDIATE
  - SHIFT OR ROTATE BY AN 8-BIT UNSIGNED IMMEDIATE OPERAND

```
SHL     AX, 12
ROR     BL, 4
SAR     DX, 3
RCR     XYZ, 2
```

- MULTIPLY IMMEDIATE (IMUL)
  - IMMEDIATE SIGNED 16-BIT MULTIPLICATION WITH 16-BIT RESULT
  - IMMEDIATE OPERAND CAN BE A 16-BIT INTEGER OR A SIGNED EXTENDED 8-BIT INTEGER
  - USEFUL WHEN PROCESSING AN ARRAY INDEX

REG16 ⟵ REG/MEM 16 * IMMED 8/16

```
IMUL    BX, SI, 5       ;BX= SI * 5
IMUL    SI, -200        ;SI = SI *  -200
IMUL    DI, XYZ, 20     ;DI = XYZ * 20
```

- **PUSH IMMEDIATE (PUSH)**
  - PUSHES AN IMMEDIATE 16-BIT VALUE OR A SIGNED EXTENDED 8-BIT VALUE ONTO THE STACK

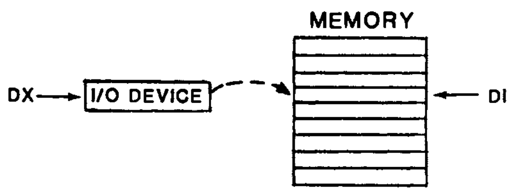
```
          PUSH  50          ;PLACE 50 ON THE TOP
                            :OF THE STACK
```

- **PUSH ALL/POP ALL (PUSHA/POPA)**
  - PUSHES/POPS ALL 8 GENERAL PURPOSE REGISTERS ONTO/OFF THE STACK

```
INT_SRV:  PUSHA             ;SAVE REGISTERS
            •
            •
            •
          POPA              ;RESTORE REGISTERS
          IRET
```

- **BLOCK I/O (INS,OUTS)**
  - MOVES A STRING OF BYTES OR WORDS BETWEEN MEMORY AND AN I/O PORT
  - SYNCHRONIZING POSSIBLE VIA READY LINE

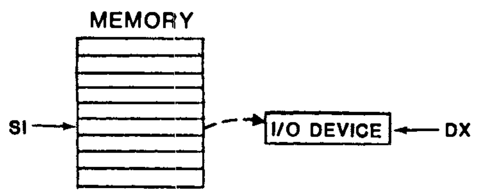
INS



INSB (BYTE TRANSFER)
INSW (WORD TRANSFER)
} [DI] ← I/O[DX]
DI ← DI +/- INCR*

*+/- INCR: + WHEN DF = 0 (CLD)
- WHEN DF = 1 (STD)

OUTS



OUTSB (BYTE TRANSFER)
OUTSW (WORD TRANSFER)
} I/O[DX] ← [SI]
SI ← SI +/- INCR*

INCR: 1 FOR BYTE TRANSFERS
2 FOR WORD TRANSFERS

HIGH LEVEL LANGUAGE SUPPORT

- CHECK ARRAY BOUNDS (BOUND)
  - CHECKS AN ARRAY INDEX REGISTER AGAINST THE ARRAY BOUNDS WHICH ARE STORED IN A 2 WORD MEMORY BLOCK
- ENTER PROCEDURE (ENTER)
  - SAVES STACK FRAME POINTERS FROM CALLING PROCEDURE AND SETS UP NEW STACK FRAME FOR CURRENT PROCEDURE
- LEAVE PROCEDURE (LEAVE)
  - RESTORES CALLER'S STACK FRAME UPON PROCEDURE EXIT

FORMAT OF "BOUND" INSTRUCTION

BOUND 16 BIT REGISTER, ARRAY LIMITS

ie

```
DATA SEGMENT
        ⌇
     ARRAY_1 DB 100 DUP (?)
     ARRAY_1_ LIMITS DW OFFSET ARRAY_1
        ⌇                   DW OFFSET ARRAY_1 +(SIZE ARRAY_1-1)
DATA ENDS
CODE SEGMENT
     ASSUME  CS:CODE, DS:DATA
        ⌇
     BOUND BX, ARRAY_1_LIMITS
     MOV AL,  BX
        ⌇
```

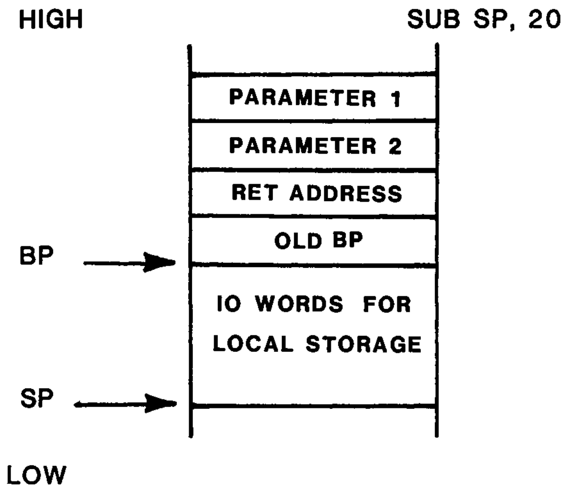
IF BX IS OUTSIDE THE LIMITS THEN AN INTERNAL INTERRUPT OF TYPE 5 IS GENERATED.

FORMAT OF "ENTER" INSTRUCTION

1. ENTER LOCAL STORAGE, NESTING LEVEL

ie

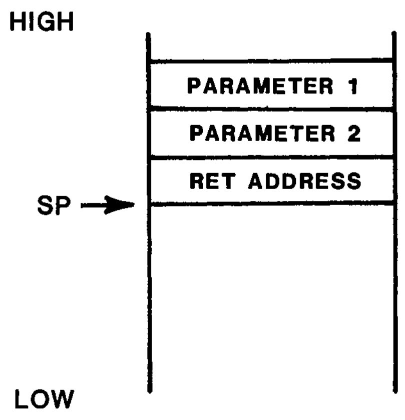
```
ENTER 20, 0     ≡     PUSH BP
                      MOV BP, SP
                      SUB SP, 20
```



FORMAT OF "LEAVE" INSTRUCTION

2. LEAVE

```
ie  LEAVE     ≡     MOV SP, BP
                    POP BP
```



CLASS EXERCISE 15.1

USE A MULTIPLY IMMEDIATE INSTRUCTION TO MULTIPLY THE CONTENTS OF BYTE PORT ØD8H TIMES A VALUE OF -5. OUTPUT THE RESULT TO WORD PORT ØFFFAH .

WHERE TO FIND MORE INFORMATION

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)

CHAPTER 5 - iAPX 186,188 HARDWARE DESIGN OVERVIEW

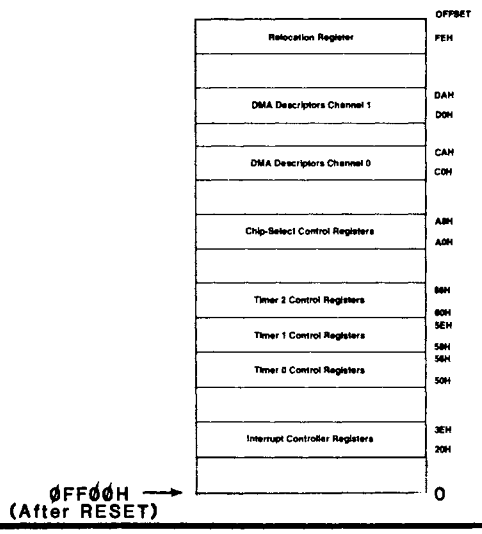
CHAPTER 16

iAPX 186, 188 CONTROL BLOCK CHIP SELECT AND WAIT STATE LOGIC

- CONTROL REGISTER BLOCK
- MEMORY CHIP SELECTS
- PERIPHERAL CHIP SELECTS
- WAIT STATE LOGIC

PERIPHERAL CONTROL

- ON-CHIP PERIPHERALS PROGRAMMED VIA AN INTERNAL REGISTER BLOCK.
- REGISTER BLOCK INITIALLY PLACED IN I/O ADDRESS SPACE AT A BASE ADDRESS OF ØFFØØH.
- BASE ADDRESS CAN BE CHANGED USING RELOCATION REGISTER.


OFFSET
Relocation Register
FEH
DMA Descriptors Channel 1
DAH
D0H
DMA Descriptors Channel 0
CAH
C0H
Chip-Select Control Registers
A8H
A0H
Timer 2 Control Registers
66H
60H
Timer 1 Control Registers
5EH
58H
Timer 0 Control Registers
56H
50H
Interrupt Controller Registers
3EH
20H
0
ØFFØØH
(After RESET)

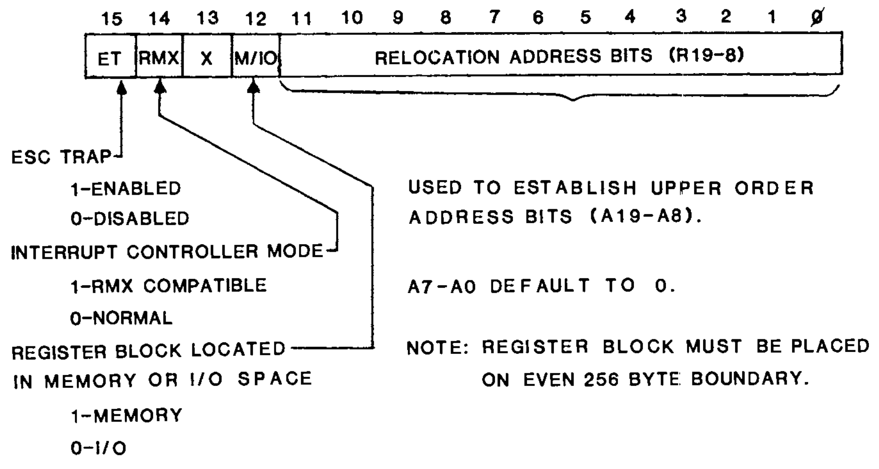
ACCESSING INTERNAL REGISTERS

- REGISTERS ARE REFERENCED AS NORMAL I/O PORTS OR MEMORY LOCATIONS.
- UPON DETECTING ANY ADDRESS WITHIN THE REGISTER BLOCK, CPU DIRECTS ACCESS TO APPROPRIATE INTERNAL REGISTER.
- EXAMPLE - READ INTERRUPT MASK REGISTER (OFFSET = 28H).

```
I/O MAPPED:
                    MOV   DX, ØFF28H
                    IN    AX, DX
MEMORY MAPPED:
                    LEA   BX, REGISTER_BLOCK
                    MOV   AX, [BX + 28H]
```

CHANGING REGISTER BLOCK BASE ADDRESS

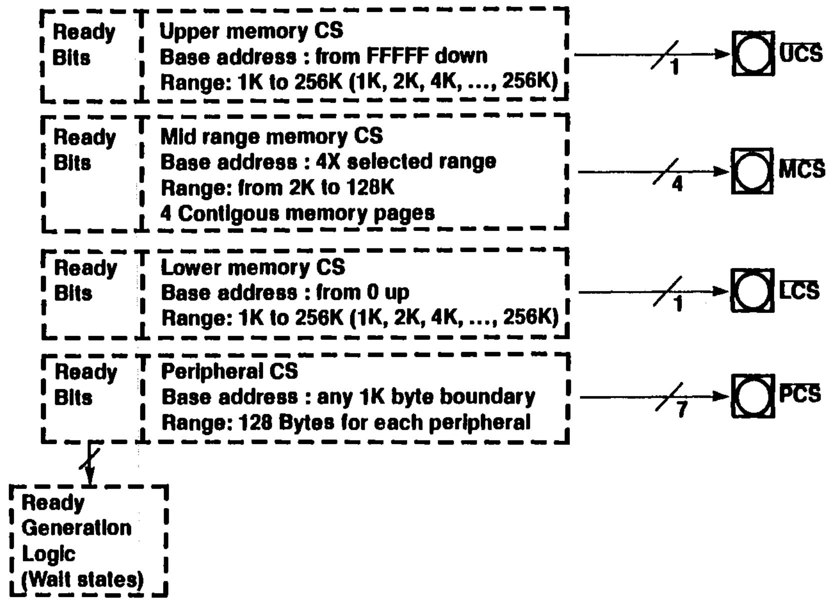
- BASE ADDRESS CAN BE MODIFIED USING RELOCATION REGISTER. THIS REGISTER IS FOUND IN REGISTER BLOCK AT AN OFFSET OF ØFEH.
- AFTER RESET, RELOCATION REGISTER CONTAINS 2ØFFH. THIS VALUE ESTABLISHES BASE ADDRESS OF ØFFØØH.


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 Ø
ET
RMX
X
M/IO
RELOCATION ADDRESS BITS (R19-8)
ESC TRAP
1-ENABLED
0-DISABLED
INTERRUPT CONTROLLER MODE
1-RMX COMPATIBLE
0-NORMAL
REGISTER BLOCK LOCATED IN MEMORY OR I/O SPACE
1-MEMORY
0-I/O
USED TO ESTABLISH UPPER ORDER ADDRESS BITS (A19-A8).
A7-A0 DEFAULT TO 0.
NOTE: REGISTER BLOCK MUST BE PLACED ON EVEN 256 BYTE BOUNDARY.

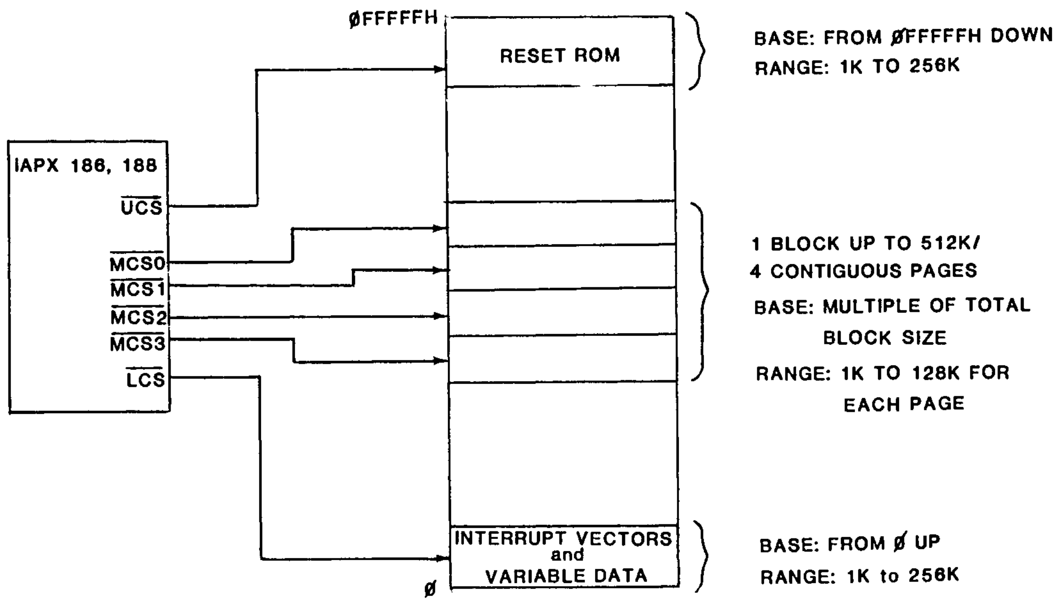
iAPX 186,188 CHIP SELECT/READY GENERATION LOGIC

- PROVIDES CHIP SELECT AND WAIT STATES FOR UP TO 6 MEMORY BANKS
- PROVIDES CHIP SELECT AND WAIT STATES FOR UP TO 7 PERIPHERAL DEVICES
- 0-3 WAIT STATES CAN BE PROGRAMMED FOR EACH RANGE

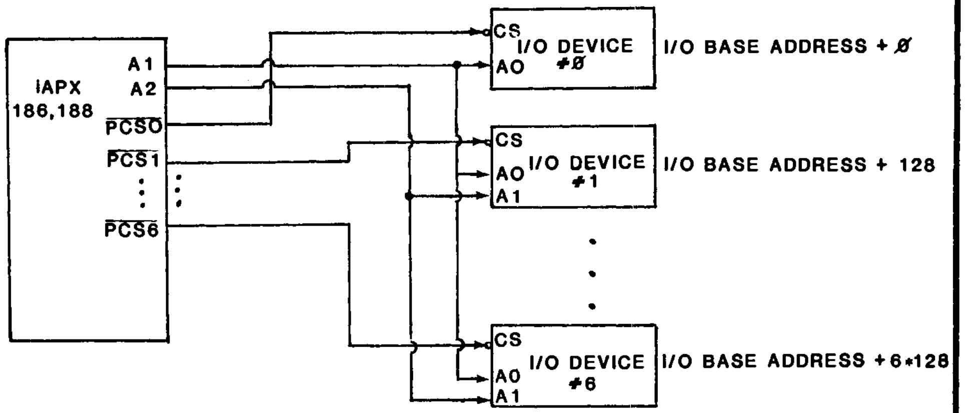
CHIP SELECT/READY GENERATION BLOCK DIAGRAM


Ready Bits
Upper memory CS
Base address : from FFFFF down
Range: 1K to 256K (1K, 2K, 4K, ..., 256K)
1
UCS
Ready Bits
Mid range memory CS
Base address : 4X selected range
Range: from 2K to 128K
4 Contigous memory pages
4
MCS
Ready Bits
Lower memory CS
Base address : from 0 up
Range: 1K to 256K (1K, 2K, 4K, ..., 256K)
1
LCS
Ready Bits
Peripheral CS
Base address : any 1K byte boundary
Range: 128 Bytes for each peripheral
7
PCS
Ready Generation Logic (Wait states)

MEMORY CHIP SELECTS

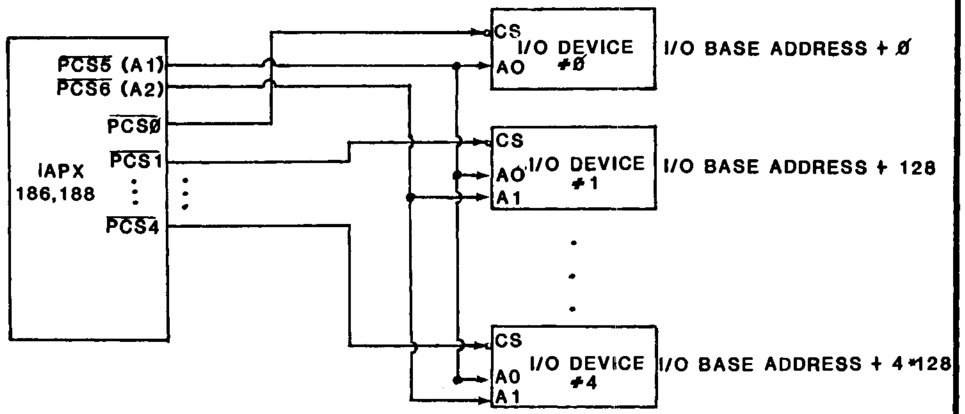


PERIPHERAL CHIP SELECTS



- MUST KEEP I/O DEVICES ON EVEN BOUNDARIES. NOT REQUIRED WITH iAPX 188.

ALTERNATIVE APPROACH



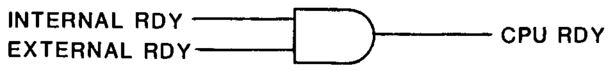
- $\overline{\text{PCS5}}$ AND $\overline{\text{PCS6}}$ PROVIDE LATCHED ADDRESS BITS A1 AND A2

READY/WAIT STATE PROGRAMMING

READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

- IMPLEMENTATION OF EXTERNAL RDY



UPPER MEMORY CHIP SELECT PROGRAMMING

UMCS REGISTER

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	∅
OFFSET:A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R∅

$\underbrace{13 \ldots 6}$ A19-A10 OF BASE ADDRESS
UPPER LIMIT = ∅FFFFFH

$\underbrace{2 \ldots 0}$ READY MODE SELECTION

UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

NOTE: AFTER RESET, THE UMCS REGISTER IS INITIALIZED TO ∅FFFBH
BASE ADDRESS = 0FFC00H
BLOCK SIZE = 1K
READY MODE = 3 WAIT STATES, EXTERNAL RDY USED

LOW MEMORY CHIP SELECT PROGRAMMING

LMCS REGISTER

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	∅
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0

A19-A10 OF UPPER ADDRESS
BASE ADDRESS = ∅

READY MODE
SELECTION

LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

NOTE: AFTER RESET, THE LMCS REGISTER IS UNDEFINED.
THE $\overline{\text{LCS}}$ CHIP SELECT LINE REMAINS INACTIVE UNTIL
THE LMCS REGISTER IS PROGRAMMED.

MID-RANGE MEMORY CHIP SELECT PROGRAMMING

MPCS REGISTER

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	∅
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

RELATE TO
PERIPHERAL
CHIP SELECTS

PERIPHERAL
READY MODE
SELECTION

MPCS Programming Values

Total Block Size	Individual Select Size	M6-M0
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

CAUTION: ONLY ONE BIT SHOULD BE SET. OTHERWISE, UNPREDICTABLE OPERATION OF MCS LINES WILL OCCUR.

MMCS REGISTER

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	∅
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0

A19-A13 OF BASE ADDRESS
MUST BE MULTIPLE OF TOTAL BLOCK SIZE.

MID-RANGE MEMORY
READY MODE
SELECTION

NOTE: AFTER RESET, THE MPCS and MMCS REGISTERS ARE UNDEFINED. THE MCS LINES REMAIN INACTIVE UNTIL BOTH THE MPCS AND MMCS REGISTERS ARE PROGRAMMED.

PERIPHERAL CHIP SELECT PROGRAMMING

PACS REGISTER

OFFSET: A4H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	∅
U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0

A19 - A10 OF BASE ADDRESS
MUST BE MULTIPLE OF 1K

READY MODE SELECTION
FOR $\overline{PCS0}$ - $\overline{PCS3}$

PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

MPCS REGISTER

OFFSET: A8H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	∅
1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

MID - RANGE MEMORY CONTROL

READY MODE SELECTION
FOR $\overline{PCS4}$ - $\overline{PCS6}$

MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided. 1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

WHERE TO FIND MORE INFORMATION

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)

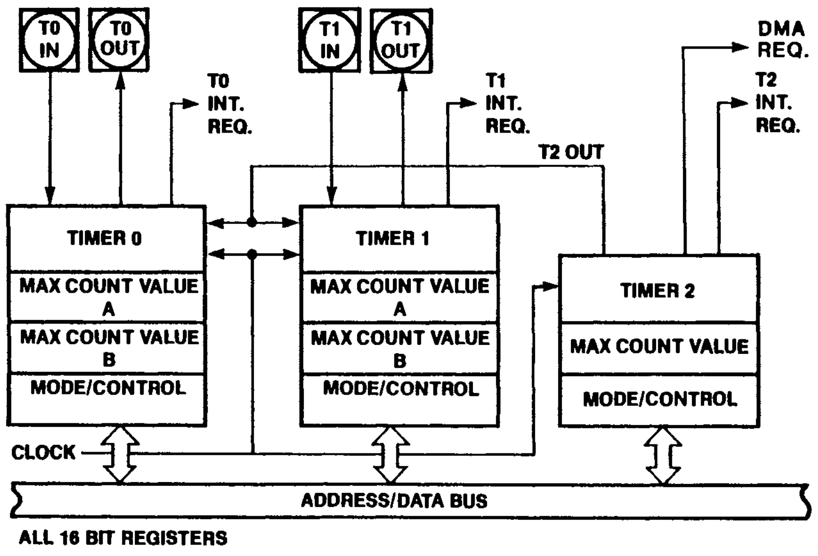
CHAPTER 5 - iAPX 186,188 HARDWARE DESIGN OVERVIEW

CHAPTER 17

iAPX 186,188 TIMER

- DESCRIPTION
- FEATURES
- PROGRAMMING

iAPX 186,188 TIMER/COUNTER BLOCK DIAGRAM

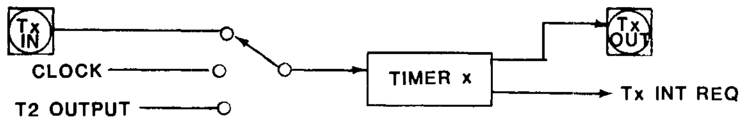


iAPX 186,188 TIMER FEATURES

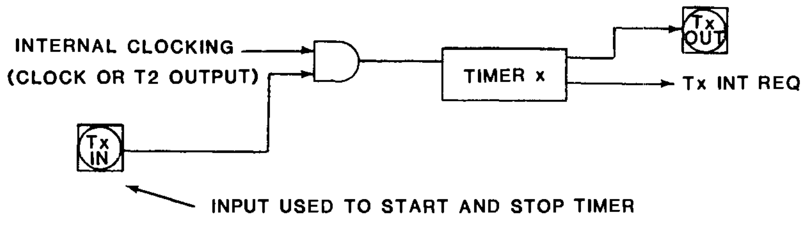
- 3 INDEPENDENT 16-BIT PROGRAMMABLE TIMER/COUNTERS (64K MAX COUNT)
- TIMERS COUNT UP
- TIMER REGISTERS MAY BE READ OR WRITTEN AT ANY TIME
- TIMERS CAN INTERRUPT ON TERMINAL COUNT VIA INTERNAL INTERRUPT CONTROLLER
- TIMERS CAN HALT OR CONTINUE ON TERMINAL COUNT

TIMER Ø AND TIMER 1 OPTIONS

- COUNT INTERNAL OR EXTERNAL PULSES



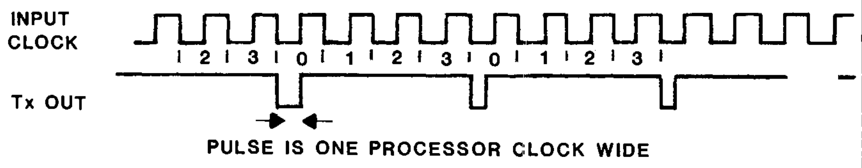
- GATE OR RETRIGGER THE TIMER



TIMER Ø AND TIMER 1 OPTIONS (CONT.)

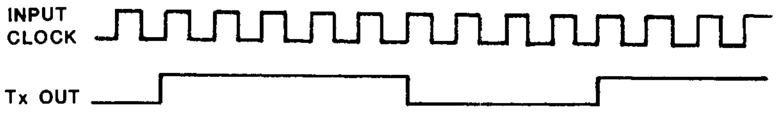
- GENERATE PULSE OUTPUT USING SINGLE MAX COUNT REGISTER.

MAX COUNT REGISTER A = 4



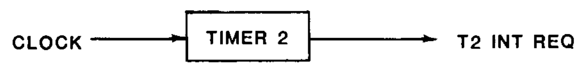
- GENERATE PULSE OUTPUTS OF ANY DUTY CYCLE USING BOTH MAX COUNT REGISTERS.

MAX COUNT REGISTER A = 5
MAX COUNT REGISTER B = 4

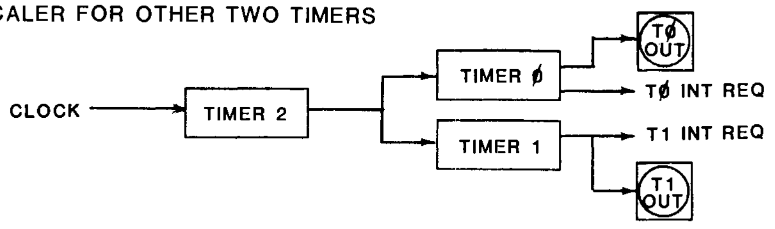


TIMER 2 OPTIONS

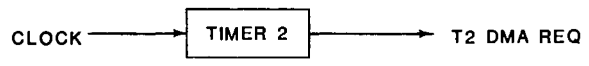
- CLOCK COUNTER - REAL TIME CLOCK, TIME DELAY



- PRESCALER FOR OTHER TWO TIMERS



- DMA REQUEST SOURCE - TIMED DMA TRANSFERS



TIMER Ø AND TIMER 1 MODE/CONTROL REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
EN	$\overline{\text{INH}}$	INT	RIU	Ø	Ø	Ø	Ø	Ø	Ø	MC	RTG	P	EXT	ALT	CONT

RIU — REGISTER IN USE
Ø = MAX COUNT REG A
1 = MAX COUNT REG B

INT — INTERRUPT ENABLE
Ø = DISABLED
1 = ENABLED

$\overline{\text{INH}}$ — INHIBIT UPDATE OF EN BIT DURING WRITE TO MODE/ CONTROL REGISTER
Ø = EN BIT UNAFFECTED
1 = EN BIT MODIFIED

EN — TIMER ENABLE
Ø = DISABLED
1 = ENABLED

CONT — CONTINUOUS COUNT
Ø = ONE SHOT
1 = CONTINUOUS COUNT

ALT — ALTERNATE BETWEEN MAX COUNT REGISTERS
Ø = REG A ALWAYS USED
1 = ALTERNATE

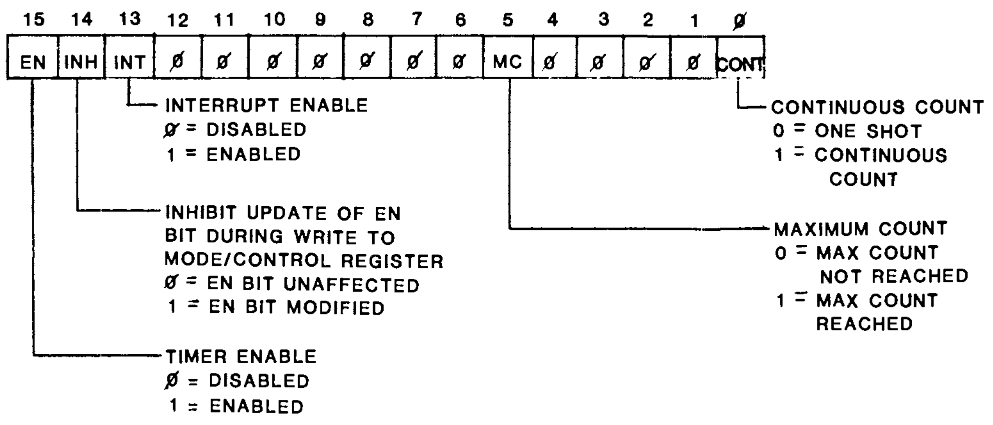
EXT — EXTERNAL INPUT
0=INTERNAL CLOCK
1=EXTERNAL CLOCK

P — PRESCALER MODE ENABLE
Ø = DISABLED
1 = ENABLED

RTG — RETRIGGER MODE ENABLE
0=GATED
1=RETRIGGERED

MC — MAXIMUM COUNT
0=MAX COUNT NOT REACHED
1=MAX COUNT REACHED

TIMER 2 MODE/CONTROL REGISTER



TIMER CONTROL BLOCK FORMAT

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

- THE COUNT REGISTERS CAN BE READ OR WRITTEN AT ANY TIME.
- AFTER RESET, THE FOLLOWING CONDITIONS EXIST:
  1) ALL EN BITS ARE RESET PREVENTING TIMER COUNTING
  2) ALL TIMER OUT PINS ARE HIGH

WHERE TO FIND MORE INFORMATION

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)

CHAPTER 5 - iAPX 186,188 HARDWARE DESIGN OVERVIEW

CHAPTER 18

iAPX 186,188 DMA CONTROLLER

- MOTIVATION FOR DIRECT MEMORY ACCESS
- DESCRIPTION OF CONTROLLER
- FEATURES
- PROGRAMMING

WHY DIRECT MEMORY ACCESS?

- TO BRING ABOUT HIGH SPEED DATA TRANSFERS WITHIN THE SYSTEM'S MEMORY AND/OR I/O ADDRESS SPACES.

- LET'S ASSUME THAT A DISK CONTROLLER UTILIZES A 500 KHz CLOCK. THIS MEANS THAT EACH BIT CELL ON THE DISK OCCUPIES A WINDOW 2µsec IN WIDTH. THEREFORE, ONE BYTE OF DATA IS TRANSFERRED EVERY 16 µsec.

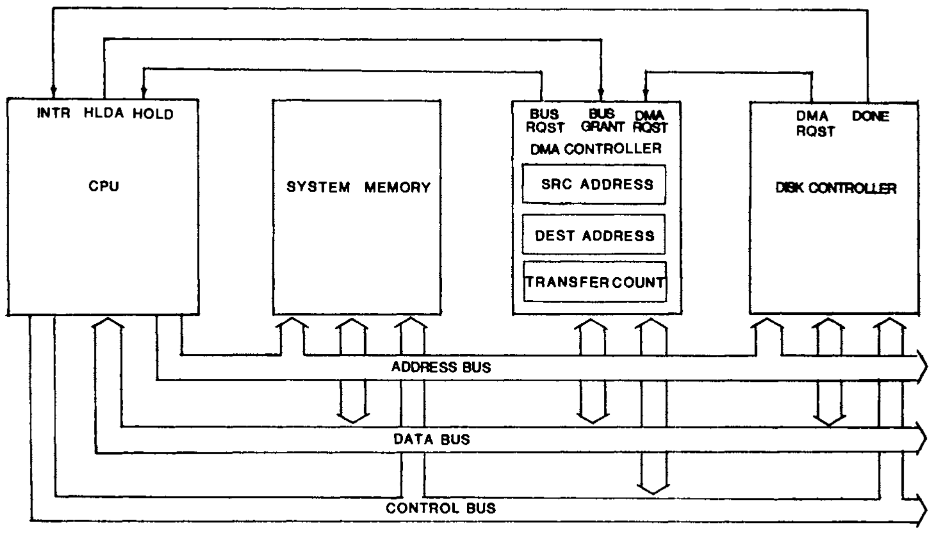
- USING INTERRUPT DRIVEN I/O, THE INTERRUPT RESPONSE AND EXECUTION TIME MUST BE LESS THAN 16 µsec IN ORDER TO TRANSFER A BYTE TO OR FROM THE CONTROLLER.

  FACTORS AFFECTING INTERRUPT RESPONSE AND EXECUTION TIME:

  1) WORST CASE INSTRUCTION LENGTH (EXECUTION TIME)
  2) PROCESSOR RESPONSE TO INTERRUPT
  3) REGISTER
  4) I/O SERVICING
  5) REGISTER RESTORE
  6) INTERRUPT RETURN

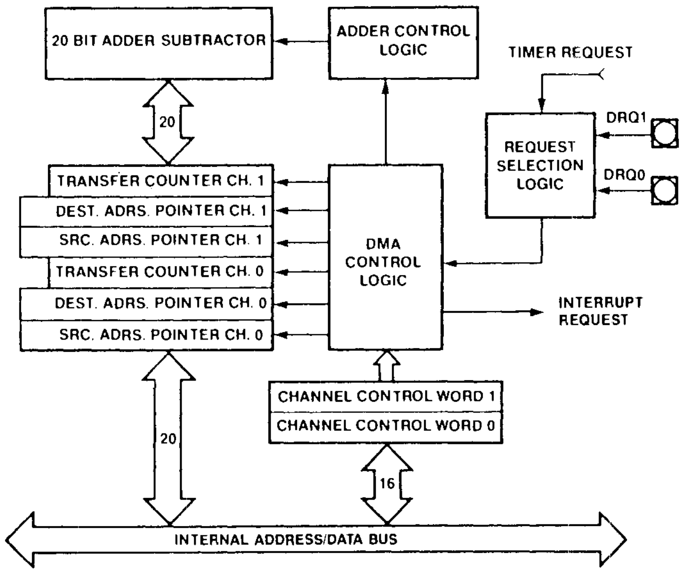
WILL WE MAKE IT?

DMA EXAMPLE



- DMA CONTROLLER ELIMINATES PROCESSOR "MIDDLEMAN" WHEN PERFORMING TRANSFERS WITHIN THE MEMORY AND/OR I/O ADDRESS SPACES. BY DOING THIS, SYSTEM THROUGHPUT IS GREATLY ENHANCED. (0.5 TO 2.0 µsec/BYTE TRANSFERRED)

iAPX 186, 188 DMA CONTROLLER BLOCK DIAGRAM



iAPX 186, 188 DMA CONTROLLER FEATURES

- TWO INDEPENDENT HIGH-SPEED CHANNELS
- SUPPORTS ALL COMBINATIONS OF TRANSFER MODES
  - MEMORY-TO-MEMORY
  - MEMORY TO-I/O
  - I/O-TO-MEMORY
  - I/O-TO-I/O

  } TWO BUS CYCLE TRANSFER
- BYTE OR WORD TRANSFERS
  - WORDS CAN BE TRANSFERRED TO/FROM ODD OR EVEN ADDRESSES
- 20-BIT SOURCE AND DESTINATION POINTER FOR EACH CHANNEL
  - CAN BE INCREMENTED/DECREMENTED INDEPENDENTLY DURING TRANSFER
- 16-BIT TRANSFER COUNTER
  - PROGRAMMABLE TERMINATE AND/OR INTERRUPT REQUEST WHEN COUNTER REACHES 0
- DMA REQUESTS CAN BE GENERATED BY TIMER 2
- 2MBYTE/SECOND MAXIMUM TRANSFER RATE

POINTER AND TRANSFER COUNT REGISTERS

- POINTER REGISTERS

UPPER 4 BITS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	A19	A18	A17	A16

LOWER 16 BITS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	AØ

- TRANSFER COUNT REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TCØ

CHANNEL CONTROL REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
DESTINATION			SOURCE										CHG	ST/	
M/$\overline{\text{IO}}$	DEC	INC	M/$\overline{\text{IO}}$	DEC	INC	TC	INT	SYN		P	TDRQ	X	NOCHG	$\overline{\text{STOP}}$	B/W

M/IO - POINTER IS IN MEMORY/IO SPACE (1/0)

INC - INCREMENT POINTER AFTER TRANSFER
Ø - NO INCREMENT
1 - INCREMENT

DEC - DECREMENT POINTER AFTER TRANSFER
Ø - NO DECREMENT
1 - DECREMENT

DMA TERMINATE WHEN TRANSFER COUNT REACHES 0
Ø - DISABLED
1 - ENABLED

INTERRUPT ON BYTE COUNT TERMINATION
Ø - DISABLED
1 - ENABLED

BYTE/WORD (Ø/1) TRANSFER

START/STOP (1/0) CHANNEL

CHANGE/DO NOT CHANGE ST/STOP BIT WHEN WRITING TO CHANNEL CONTROL REGISTER

TIMER 2 DMA REQUEST
Ø - DISABLED
1 - ENABLED

CHANNEL PRIORITY - RELATIVE TO OTHER CHANNEL
Ø - LOW PRIORITY
1 - HIGH PRIORITY
CHANNELS ALTERNATE CYCLES IF SET TO SAME PRIORITY.

CHANNEL SYNCHRONIZATION
ØØ - NO SYNC
Ø1 - SOURCE SYNC
1Ø - DESTINATION SYNC
11 - UNUSED

DMA CONTROL BLOCK FORMAT

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

- AFTER RESET, BOTH CHANNELS ARE DISABLED BY RESETTING THEIR ST/STOP BITS.

WHERE TO FIND MORE INFORMATION

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)

CHAPTER 5 – iAPX 186,188 HARDWARE DESIGN OVERVIEW

DAY 5 OBJECTIVES

BY THE TIME YOU FINISH TODAY YOU WILL:

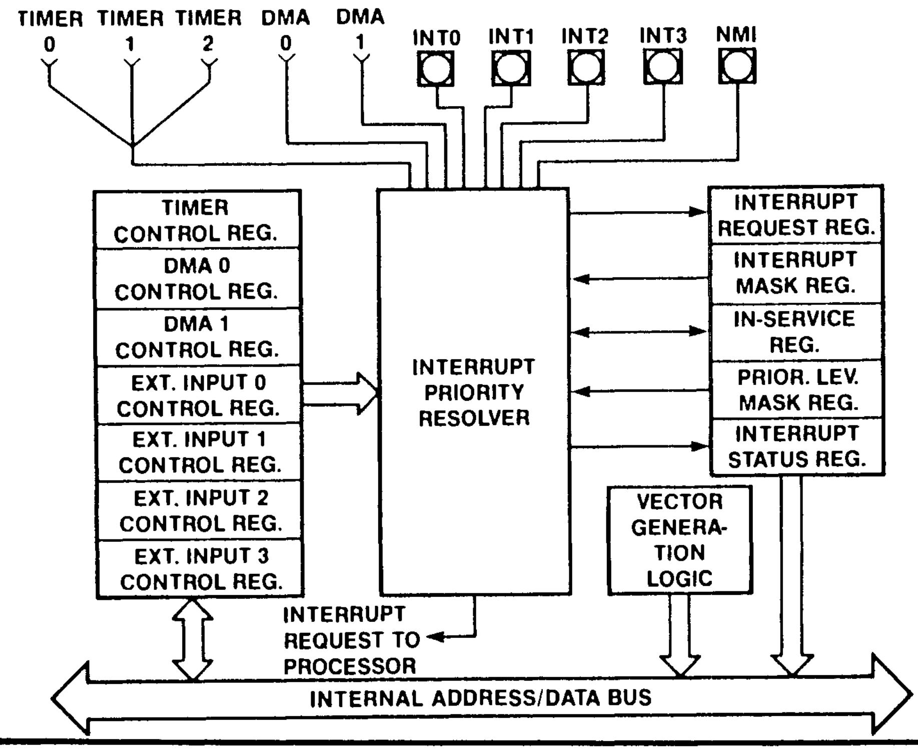
- DEFINE THE OPERATIONAL MODES OF THE 80186 INTERRUPT CONTROLLER AND PROGRAM IT TO OPERATE IN A REQUIRED MODE
- SEE HOW TO USE THE LIBRARIAN (LIB86) AND THE MODULE CROSS-REFERENCER (CREF86)
- DEFINE THE ROLE OF THE 8089 I/O PROCESSOR
- DEFINE THE SOFTWARE INTERFACE BETWEEN THE 8086 AND THE 8089

CHAPTER 19

iAPX 186,188 INTERRUPT CONTROL UNIT

- DESCRIPTION
- FEATURES
- PROGRAMMING

iAPX 186, 188 INTERRUPT CONTROL UNIT BLOCK DIAGRAM



iAPX 186,188 INTERRUPT CONTROL UNIT

- **ACCEPTS INTERRUPTS FROM INTERNAL SOURCES (DMA, TIMERS) AND FROM 5 EXTERNAL PINS (NMI + 4 INTERRUPT PINS)**
- **PROVIDES FULLY NESTED, SPECIAL FULLY NESTED FEATURES OF THE 8259A**
- **EXPANDABLE TO 128 EXTERNAL INTERRUPTS BY CASCADING MULTIPLE 8259A'S**
  - iAPX 186 CAN BE CONFIGURED TO SUPPORT TWO MASTER 8259A'S
- **EIGHT DISTINCT PRIORITY LEVELS**
- **PROGRAMMABLE PRIORITY LEVEL FOR EACH INTERRUPT SOURCE**
- **LEVEL OR EDGE TRIGGERED PROGRAMMABLE MODES FOR EACH EXTERNAL INTERRUPT SOURCE.**

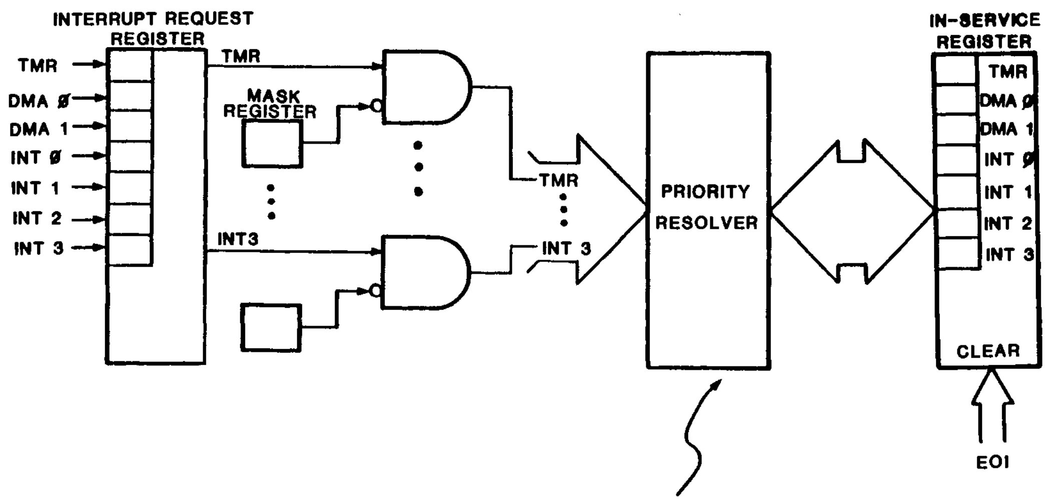
iAPX 186,188 PRE-ASSIGNED INTERRUPT TYPES

Interrupt Name	Vector Type	Comments
Type 0	0	Divide error trap
Type 1	1	Single step trap
NMI	2	Non-maskable interrupt
Type 3	3	Breakpoint trap
INT0	4	Trap on overflow
Array bounds trap	5	BOUND instruction trap
Unused op trap	6	Invalid op-code trap
ESCAPE op trap	7	Supports 8087 emulation
Timer 0	8	Internal h/w interrupt
Timer 1	18	Internal h/w interrupt
Timer 2	19	Internal h/w interrupt
DMA 0	10	Internal h/w interrupt
DMA 1	11	Internal h/w interrupt
Reserved	9	*Reserved*
INT0	12	External interrupt 0
INT1	13	External interrupt 1
INT2/INTA0	14	External interrupt 2
INT3/INTA1	15	External interrupt 3

INTERRUPT VECTORING HIGHLIGHTS

- FASTER INTERRUPT RESPONSE TIME FOR INTERNALLY GENERATED INTERRUPTS (42 CLOCKS) VS. iAPX 86 (61 CLOCKS)
  - 1.5X THE 8086
- SHORTER INTERRUPT (FUNCTION OF THE LONGEST INSTRUCTION MUL AND DIV TIMES ARE 1/3 THE 8MHz iAPX 86)

INTERRUPT CONTROL UNIT OPERATION



LOOKS AT CURRENT REQUESTS AND ALSO ANY INTERRUPTS IN-SERVICE. IF REQUESTING LEVEL HAS HIGHEST PRIORITY, IT IS PUT IN-SERVICE.

- INTERRUPT REQUEST, IN-SERVICE AND MASK REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I3	I2	I1	IØ	D1	DØ	Ø	TMR

INTERRUPT REQUEST REGISTER - READ ONLY
IN-SERVICE AND MASK REGISTERS - READ AND WRITE

- EOI REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
SPEC/NSPEC	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	S4	S3	S2	S1	SØ

SPECIFIC/NONSPECIFIC (Ø/1) END OF INTERRUPT

INTERRUPT VECTOR TYPE

EOI REGISTER - WRITE ONLY

• TIMER AND DMA CONTROL REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	MSK	PR2	PR1	PRØ

MASK BIT
Ø - NO MASK
1 - MASK

PRIORITY

• INTØ AND INT1 CONTROL REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	SFNM	C	LTM	MSK	PR2	PR1	PRØ

SPECIAL FULLY NESTED MODE
Ø - NORMAL FULLY NESTED MODE
1 - SPECIAL FULLY NESTED MODE

LEVEL TRIGGER MODE
Ø - EDGE TRIGGER
1 - LEVEL TRIGGER

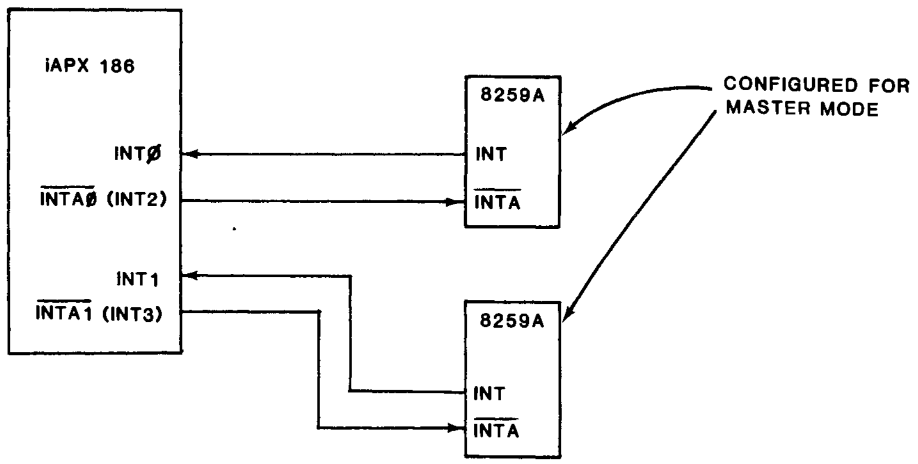
CASCADE MODE
Ø - DIRECT INPUT
1 - CASCADED INPUT

• INT2 AND INT3 CONTROL REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	LTM	MSK	PR2	PR1	PRØ

NOTE: EACH INTERRUPT SOURCE IS PROGRAMMED INDEPENDENTLY OF THE OTHERS.

CASCADE MODE


iAPX 186
INTØ
$\overline{\text{INTAØ}}$ (INT2)
INT1
$\overline{\text{INTA1}}$ (INT3)
8259A
INT
$\overline{\text{INTA}}$
8259A
INT
$\overline{\text{INTA}}$
CONFIGURED FOR MASTER MODE

• COULD HAVE UP TO 128 PRIORITIZED INTERRUPT LEVELS BY CASCADING SLAVE UNITS INTO THE MASTER UNITS SHOWN.

• USING EXTERNAL 8259's, INTO AND/OR INT1 WOULD BE CONFIGURED IN SPECIAL FULLY NESTED MODE.

OPERATIONAL CONTROL

- PRIORITY MASK REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	PRM2	PRM1	PRMØ

$\underbrace{\text{PRM2 PRM1 PRMØ}}_{\text{PRIORITY MASK}}$

USED TO DISABLE INTERRUPTS BELOW A SPECIFIED LEVEL.

- INTERRUPT STATUS REGISTER

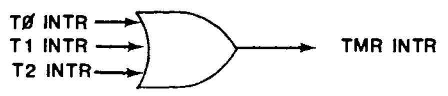
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
DHLT	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	IRT2	IRT1	IRTØ

DMA HALT
- -SETTING BIT HALTS ALL DMA TRANSFERS
- -RESET BY IRET INSTRUCTION
- -PERMITS PROMPT SERVICE OF INTERRUPT REQUEST
- -AUTOMATICALLY SET BY NMI

IRTØ — INTERRUPT REQUEST -TIMER Ø

IRT1 — INTERRUPT REQUEST -TIMER1

IRT2 — INTERRUPT REQUEST -TIMER 2



OPERATION IN A POLLED ENVIRONMENT

- CPU MUST PERIODICALLY INTERROGATE INTERRUPT CONTROL UNIT TO DETERMINE IF THERE IS A PENDING INTERRUPT REQUEST.

- POLL AND POLL STATUS REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
INT REQ	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	S4	S3	S2	S1	SØ

INT REQ — INTERRUPT REQUEST BIT
Ø-NO REQUEST
1-REQUEST PENDING

$\underbrace{\text{S4 S3 S2 S1 SØ}}$ VECTOR TYPE OF HIGHEST PRIORITY INTERRUPTING SOURCE

VALID ONLY WHEN INTREQ=1

POLL REGISTER-READING THIS REGISTER WILL AUTOMATICALLY SET IN-SERVICE BIT OF HIGHEST PRIORITY PENDING INTERRUPT.

POLL STATUS REGISTER-HAS NO EFFECT ON IN-SERVICE REGISTER.

INTERRUPT CONTROL BLOCK FORMAT

- AFTER RESET, ALL INTERRUPTS ARE DISABLED.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT CONTROLLER STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

WHERE TO FIND MORE INFORMATION

iAPX 86/88, 186/188 USER'S MANUAL (PROGRAMMER'S REFERENCE)

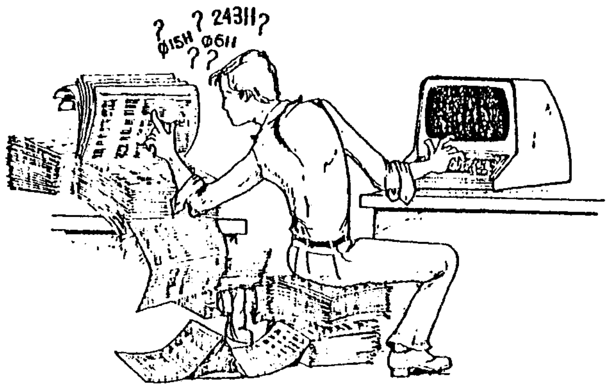
CHAPTER 5 – iAPX 186,188 HARDWARE DESIGN OVERVIEW

CHAPTER 20

LIBRARIES & MODULE CROSS-REFERENCES

- LIBRARY CHARACTERISTICS
- LIBRARY COMMANDS
- USING LIBRARIES
- INTER-MODULE CROSS REFERENCING (CREF86)

SOFTWARE DEVELOPMENT ORGANIZATION



ISIS-II LIBRARIAN

A COLLECTION OF OBJECT MODULES SUPPLIED BY SYSTEM USER

OR BY INTEL

A SPECIAL FILE CONTAINING A DIRECTORY OF PUBLICS

ALLOWS SELECTION OF JUST THOSE MODULES NEEDED BY
THE PROGRAM BY LINKING TO LIBRARY

LINKING AS PROGRAM WITHOUT A LIBRARY

OVCONT.OBJ

OVENCONTROLBLOCK:	
WRITE	EXTERNAL
READTEMP	EXTERNAL

UTIL.OBJ

UTILBLOCK :	
DELAY	EXTERNAL
READTEMP	PUBLIC

DELAY.OBJ

DELAYBLOCK:	
DELAY	PUBLIC

CONSOL.LNK

CONSOLIOBLOCK:	
READ	PUBLIC
WRITE	PUBLIC
CI	PUBLIC
CO	PUBLIC

```
- RUN LINK86 OVCONT.OBJ &
             UTIL.OBJ &
             DELAY.OBJ. &
             CONSOL.LNK TO PROCES.LNK
```

- ENTIRE MODULE CONSOLE.LNK IS INCLUDED EVEN THOUGH PUBLIC PROCEDURES 'READ' AND 'CI' ARE NEVER USED
- INSTEAD OF LINKING OBJECT MODULES INTO CONSOLE.LNK, PUT THEM IN A LIBRARY ...

LINKING A PROGRAM WITH A LIBRARY

OVCONT.OBJ

OVENCONTROLBLOCK:	
WRITE	EXTERNAL
READTEMP	EXTERNAL

UTIL.OBJ

UTILBLOCK:	
DELAY	EXTERNAL
READTEMP	PUBLIC

DELAY.OBJ

DELAYBLOCK:	
DELAY	PUBLIC

CONSOL.LIB

READ_MODULE		
	READ	PUBLIC
	CI	EXTERNAL
	CO	EXTERNAL
WRITE_MODULE		
	WRITE	PUBLIC
	CO	EXTERNAL
CI-MODULE		
	CI	PUBLIC
CO-MODULE		
	CO	PUBLIC

```
- RUN LINK86 OVCONT.OBJ. &
             UTIL.OBJ &
             DELAY.OBJ, &
             CONSOL.LIB TO PROCES.LNK
```

- ONLY INCLUDES LIBRARY MODULES REQUIRED TO SATISFY EXTERNAL REFERENCES

ISIS-II LIB86 COMMAND

```
- RUN LIB86
*
```

NO PARAMETERS ARE ALLOWED IN THE INVOCATION. LIB86 RESPONDS WITH AN ASTERISK AND WAITS FOR COMMANDS:

CREATE	- CREATE A NEW LIBRARY
ADD	- ADD OBJECT MODULES TO A LIBRARY
DELETE	- DELETE OBJECT MODULES FROM A LIBRARY
LIST	- LIST THE CONTENTS OF A LIBRARY
EXIT	- EXIT LIBRARIAN

USING LIB86 COMMANDS

```
- RUN LIB86
* CREATE CONSOL.LIB
* ADD READ.OBJ, WRITE.OBJ, OTHER.LIB (CI, CO) TO CONSOL.LIB
* DELETE OTHER.LIB (CI, CO)
* LIST CONSOL.LIB PUBLICS
CONSOL.LIB
   READ_MODULE
        READ
   WRITE_MODULE
        WRITE
   CI_MODULE
        CI
   CO_MODULE
        CO
* LIST CONSOL.LIB TO  :LP: PUBLICS
```

CLASS EXERCISE

1. THE LIBRARY CAN CONTAIN OBJECT MODULES, LINKED MODULES AND LOCATED MODULES. THE LIBRARY LISTING SHOWS THESE ENTRIES BY MODULE NAME. WHERE DOES THIS MODULE COME FROM?

   a) FOR AN ASSEMBLED OBJECT MODULE

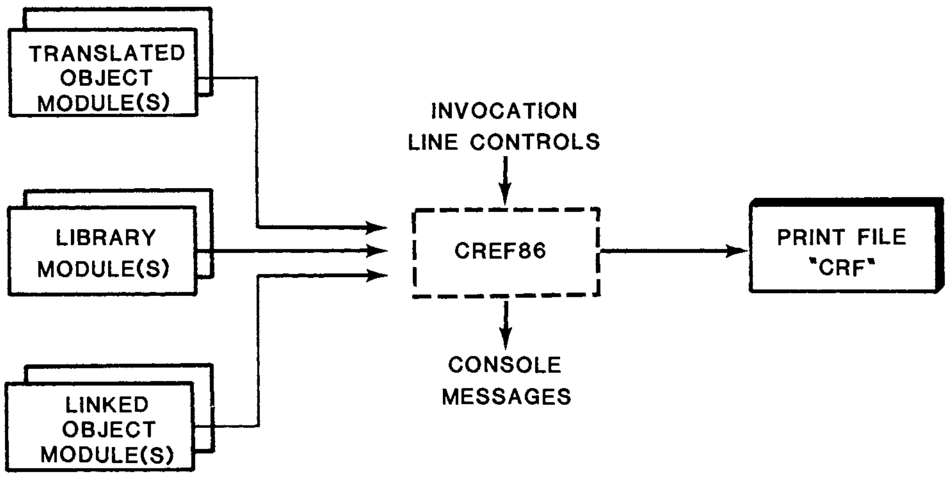
   b) FOR A LINKED MODULE

   c) FOR A LOCATED MODULE

2. WHAT ADVANTABE MIGHT BE HAD BY HAVING A LARGE NUMBER OF LIBRARY MODULES EACH WITH ONE PUBLIC SYMBOL ONLY, RATHER THAN A FEW LIBRARY MODULES EACH WITH SEVERAL PUBLIC SYMBOLS.

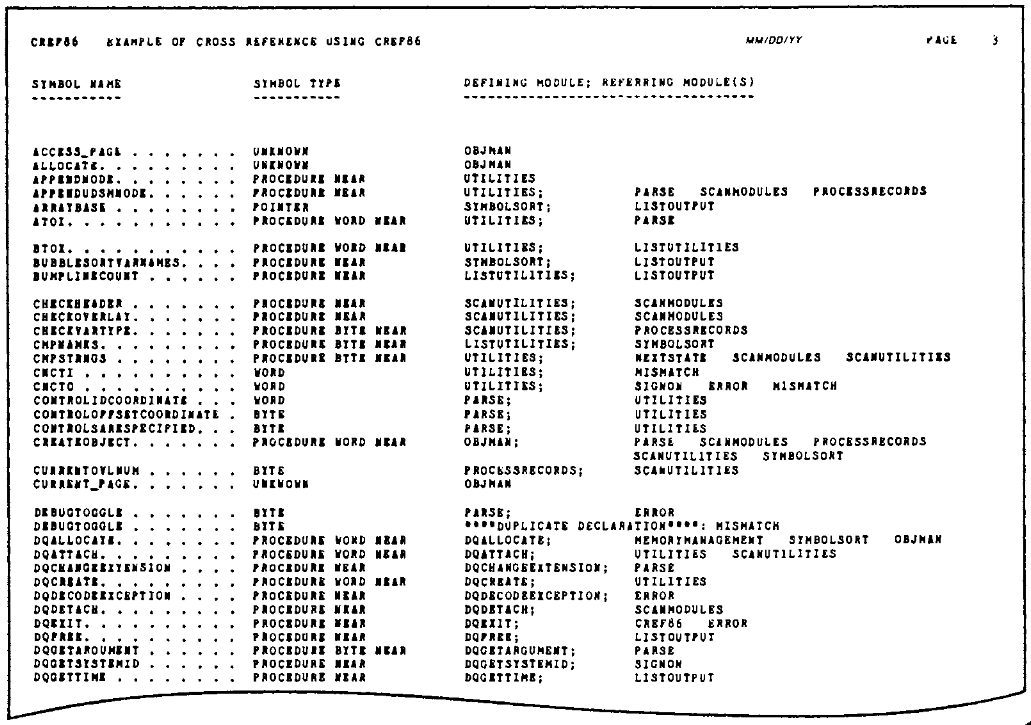
CREF86

- PROVIDES A CROSS REFERENCE LIST OF PUBLICS AND EXTERNALS USED BY MODULES OF A PROGRAM
- TYPE CHECKING OF PUBLICS/EXTERNALS
- TYPICALLY USES SAME INPUT LIST AS YOUR FINAL LINK



- RUN CREF86 INPUT-LIST [CONTROLS]

EXAMPLE: CROSS-REFERENCE LISTING



```
CREF86   EXAMPLE OF CROSS REFERENCE USING CREF86                          MM/DD/YY          PAGE    3

SYMBOL NAME                SYMBOL TYPE           DEFINING MODULE; REFERRING MODULE(S)
-----------                -----------           ------------------------------------


ACCESS_PAGE . . . . . . .  UNKNOWN               OBJMAN
ALLOCATE. . . . . . . . .  UNKNOWN               OBJMAN
APPENDNODE. . . . . . . .  PROCEDURE NEAR        UTILITIES
APPENDUDSMNODE. . . . . .  PROCEDURE NEAR        UTILITIES;          PARSE   SCANMODULES   PROCESSRECORDS
ARRAYBASE . . . . . . . .  POINTER               SYMBOLSORT;         LISTOUTPUT
ATOI. . . . . . . . . . .  PROCEDURE WORD NEAR   UTILITIES;          PARSE

BTOX. . . . . . . . . . .  PROCEDURE WORD NEAR   UTILITIES;          LISTUTILITIES
BUBBLESORTVARNAMES. . . .  PROCEDURE NEAR        SYMBOLSORT;         LISTOUTPUT
BUMPLINECOUNT . . . . . .  PROCEDURE NEAR        LISTUTILITIES;      LISTOUTPUT

CHECKHEADER . . . . . . .  PROCEDURE NEAR        SCANUTILITIES;      SCANMODULES
CHECKOVERLAY. . . . . . .  PROCEDURE NEAR        SCANUTILITIES;      SCANMODULES
CHECKVARTYPE. . . . . . .  PROCEDURE BYTE NEAR   SCANUTILITIES;      PROCESSRECORDS
CMPNAMES. . . . . . . . .  PROCEDURE BYTE NEAR   LISTUTILITIES;      SYMBOLSORT
CMPSTRNGS . . . . . . . .  PROCEDURE BYTE NEAR   UTILITIES;          NEXTSTATE   SCANMODULES   SCANUTILITIES
CNCTI . . . . . . . . . .  WORD                  UTILITIES;          MISMATCH
CNCTO . . . . . . . . . .  WORD                  UTILITIES;          SIGNON   ERROR   MISMATCH
CONTROLIDCOORDINATE . . .  WORD                  PARSE;              UTILITIES
CONTROLOFFSETCOORDINATE .  BYTE                  PARSE;              UTILITIES
CONTROLSARESPECIFIED. . .  BYTE                  PARSE;              UTILITIES
CREATEOBJECT. . . . . . .  PROCEDURE WORD NEAR   OBJMAN;             PARSE   SCANMODULES   PROCESSRECORDS
                                                                     SCANUTILITIES   SYMBOLSORT
CURRENTOVLNUM . . . . . .  BYTE                  PROCESSRECORDS;     SCANUTILITIES
CURRENT_PAGE. . . . . . .  UNKNOWN               OBJMAN

DEBUGTOGGLE . . . . . . .  BYTE                  PARSE;              ERROR
DEBUGTOGGLE . . . . . . .  BYTE                  ****DUPLICATE DECLARATION****: MISMATCH
DQALLOCATE. . . . . . . .  PROCEDURE WORD NEAR   DQALLOCATE;         MEMORYMANAGEMENT   SYMBOLSORT   OBJMAN
DQATTACH. . . . . . . . .  PROCEDURE WORD NEAR   DQATTACH;           UTILITIES   SCANUTILITIES
DQCHANGEEXTENSION . . . .  PROCEDURE NEAR        DQCHANGEEXTENSION;  PARSE
DQCREATE. . . . . . . . .  PROCEDURE WORD NEAR   DQCREATE;           UTILITIES
DQDECODEEXCEPTION . . . .  PROCEDURE NEAR        DQDECODEEXCEPTION;  ERROR
DQDETACH. . . . . . . . .  PROCEDURE NEAR        DQDETACH;           SCANMODULES
DQEXIT. . . . . . . . . .  PROCEDURE NEAR        DQEXIT;             CREF86   ERROR
DQFREE. . . . . . . . . .  PROCEDURE NEAR        DQFREE;             LISTOUTPUT
DQGETARGUMENT . . . . . .  PROCEDURE BYTE NEAR   DQGETARGUMENT;      PARSE
DQGETSYSTEMID . . . . . .  PROCEDURE NEAR        DQGETSYSTEMID;      SIGNON
DQGETTIME . . . . . . . .  PROCEDURE NEAR        DQGETTIME;          LISTOUTPUT
```

WHERE TO FIND MORE INFORMATION

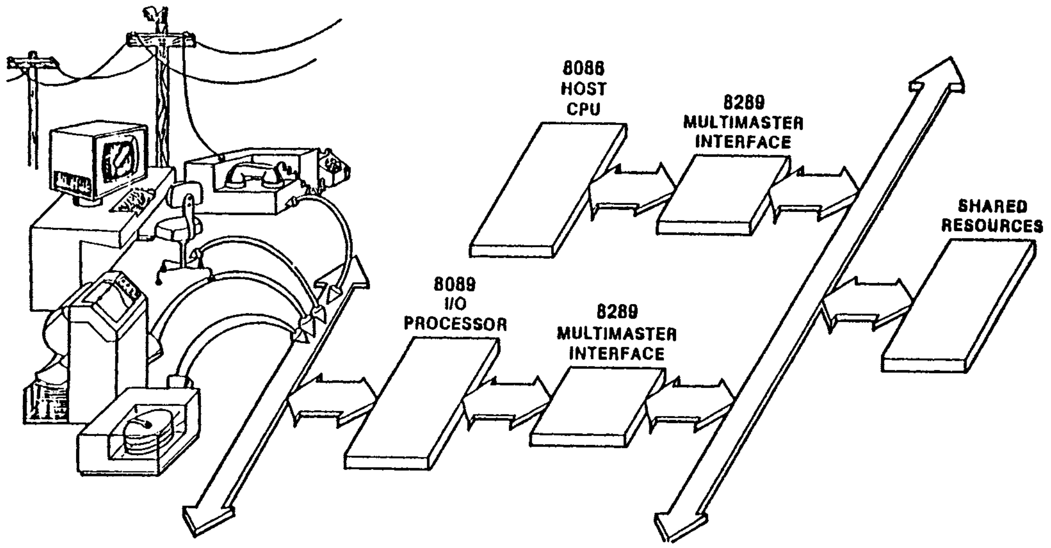
iAPX 86,88 FAMILY UTILITIES USER'S GUIDE

CHAPTER 21

OVERVIEW OF THE 8089 I/O PROCESSOR

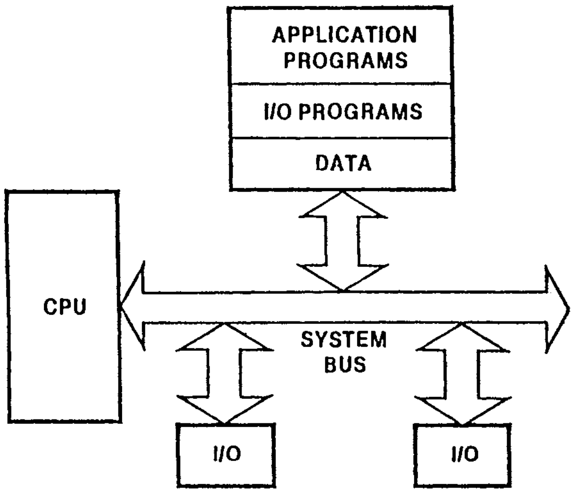
- MOTIVATION FOR USING THE 8089
- PRODUCT DESCRIPTION
- INTERFACING WITH THE 8089
- PRODUCT FEATURES
- DEVELOPMENT SUPPORT

8089 I/O PROCESSOR (IOP)



21-1

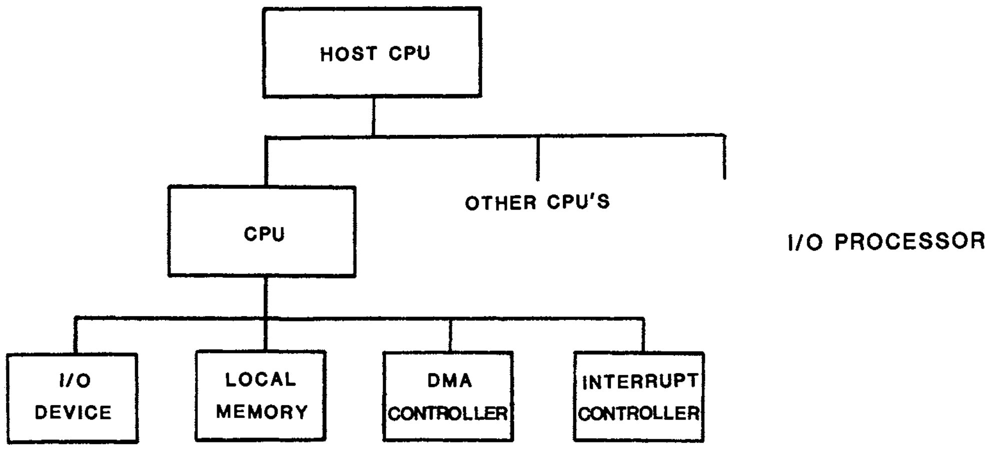
TYPICAL SYSTEM WITH I/O



- CPU EXECUTES APPLICATION AND I/O PROGRAMS
- I/O RESIDES ON SYSTEM BUS
- HIGH SPEED I/O IMPEDES APPLICATION PROGRAM EXECUTION

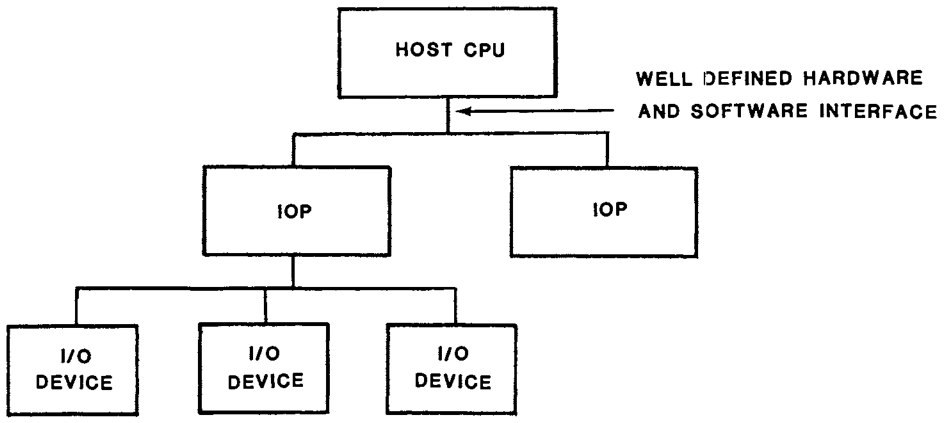
21-2

ANOTHER SYSTEM WITH I/O



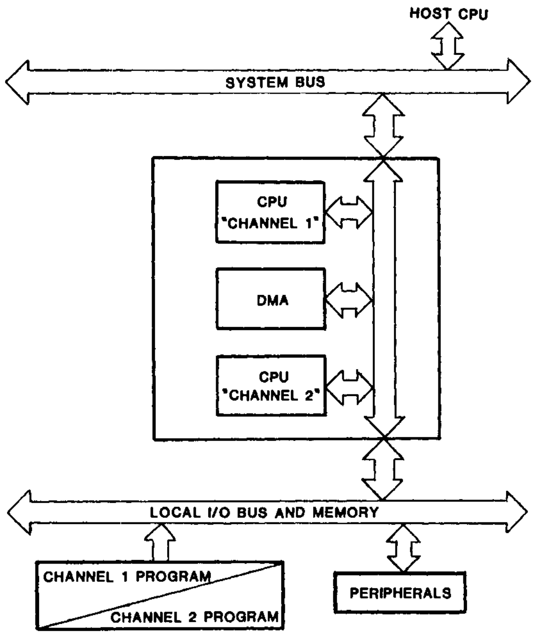
- MULTIPLE CHIP SOLUTION
- USUALLY A NONSTANDARD COMMUNICATION INTERFACE WITH HOST CPU
- DMA FACILITIES ARE NOT FLEXIBLE

SYSTEM WITH I/O PROCESSOR



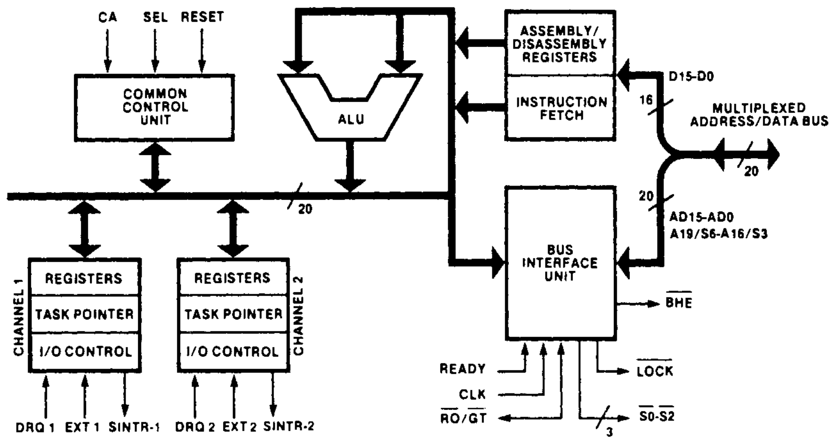
- I/O CONTROL FUNCTIONS INTEGRATED WITH DMA FACILITIES
- FASTER RESPONSE
- FLEXIBLE DMA FACILITIES

8089 CONTAINS 2 INDEPENDENT I/O CHANNELS

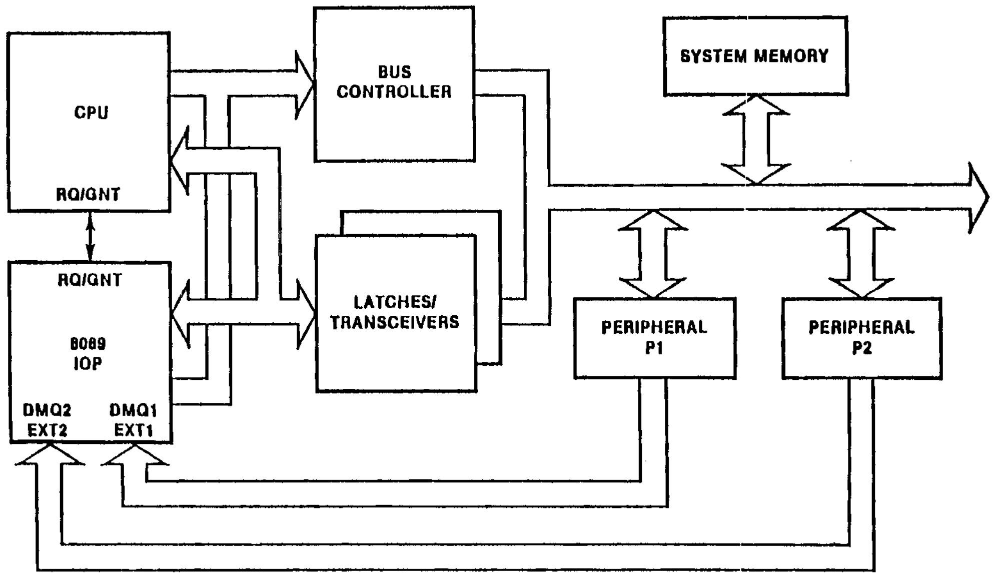


- 2 INDEPENDENT I/O CHANNELS
- 2 REGISTER SETS, 2 INSTRUCTION POINTERS
- 2 LOGICAL BUSES
- 2 I/O PROGRAMS CAN EXECUTE CONCURRENTLY
- I/O PROGRAMS CAN BE LOCATED IN I/O OR SYSTEM SPACE

8089 BLOCK DIAGRAM

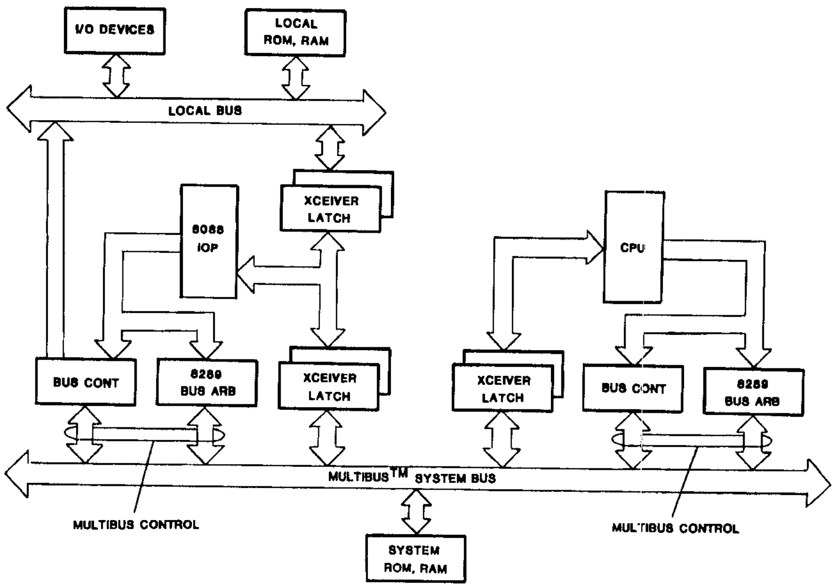


LOCAL CONFIGURATION



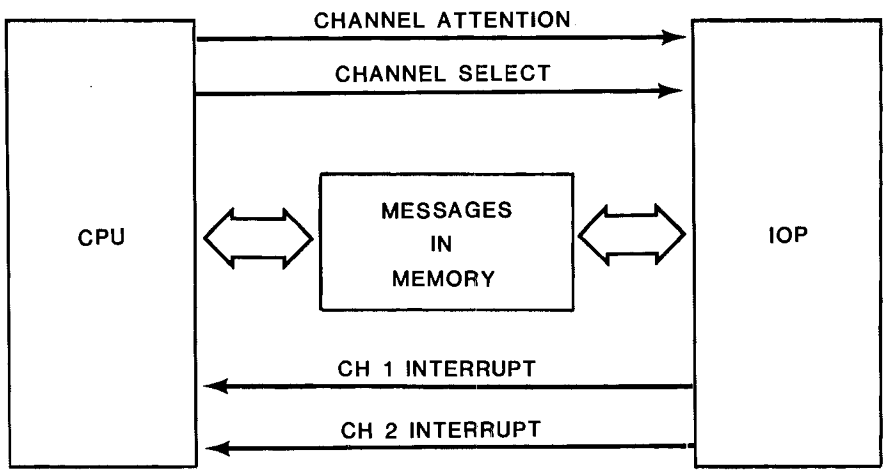
- IOP SHARES THE SYSTEM BUS INTERFACE LOGIC WITH THE HOST CPU

REMOTE CONFIGURATION



- REMOTE CONFIGURATION ALLOWS PARALLEL PROCESSING

CPU/IOP COMMUNICATION



- CPU CAN WRITE TO A PORT MAPPED TO MULTIBUS. PORT IS ON 8089 BOARD WITH TWO PINS CONNECTED TO ATTENTION/SELECT

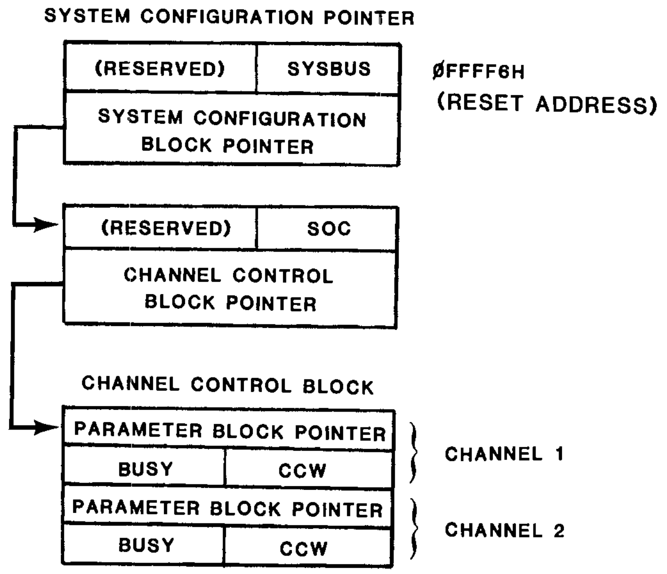
INITIALIZATION OF CHANNEL CONTROL BLOCK

SYSBUS:
8/16 BIT (8088/8086)

SYSTEM OPERATION COMMAND:

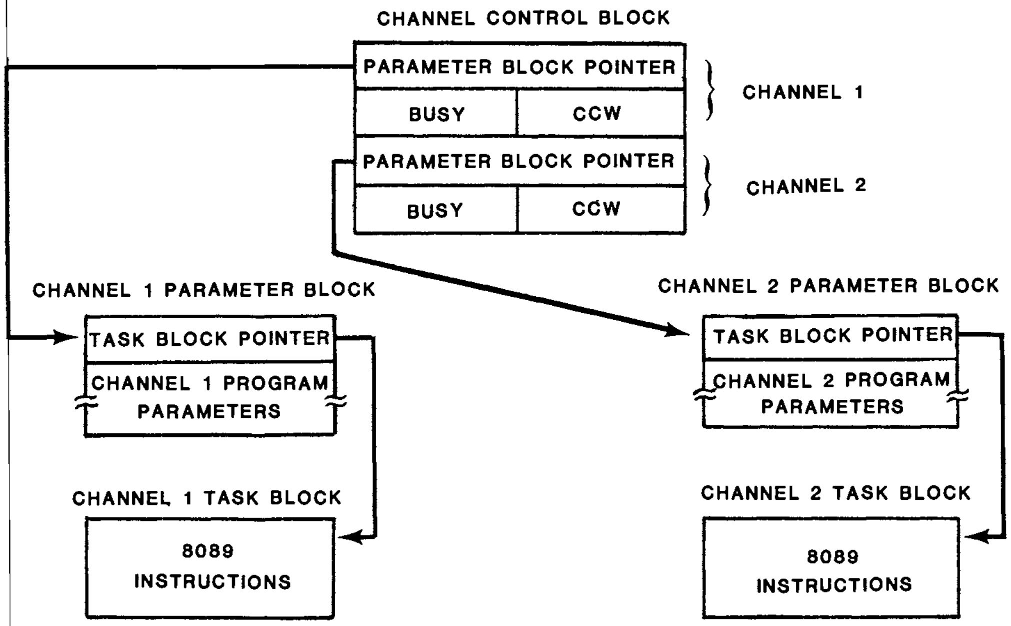
a) I/O BUS WIDTH

b) REQUEST/GRANT MODE



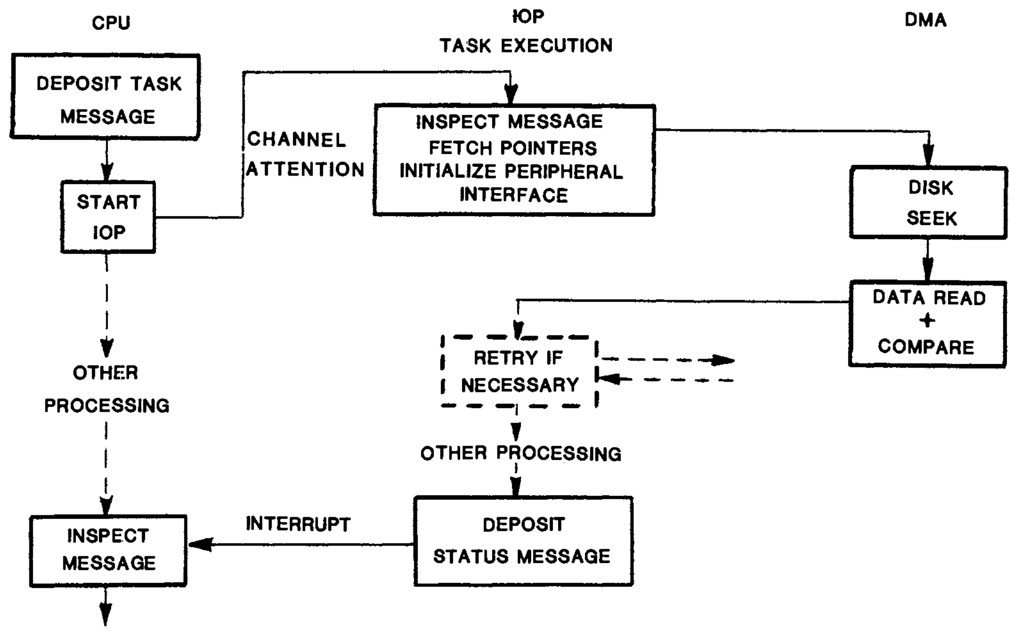
- UPON SEEING FIRST CHANNEL ATTENTION AFTER RESET, 8089 INITIALIZES ITSELF

MESSAGE STRUCTURE



- CHANNEL COMMAND WORD (CCW) SAYS WHAT TO DO (eg START, SUSPEND, ENABLE INTERRUPTS)

TYPICAL TASK FLOW
(DISK EXAMPLE)



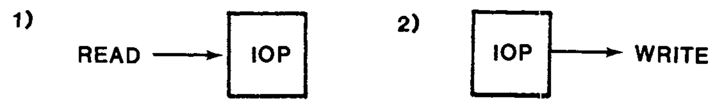
IOP PROCESSES IN PARALLEL, ENTERS DMA MODE, THEN RETURNS TO PROCESSING INSTRUCTIONS

INSTRUCTION SET OPTIMIZED FOR I/O PROCESSING AND CONTROL

- TAILORED SPECIFICALLY FOR I/O OPERATIONS
  - LOGIC INSTRUCTIONS (MASKING)
  - BIT MANIPULATIONS, BRANCHING
  - ELEMENTARY ARITHMETICS
  - GENERALIZED MOVE
- CONTROL TRANSFERS
  - BRANCH RELATIVE
  - PROCEDURAL CALL/RETURN
- VERSATILE ADDRESSING MODES OPERATE ON 8 OR 16 BIT DATA
  - BASED
  - BASED RELATIVE
  - BASED INDEXED
  - BASED INDEXED WITH AUTO INCREMENT
- DMA CONTROL
  - SYSTEM AND I/O BUS WIDTH SPECIFICATION
  - DMA ACTIVATION

DMA FACILITIES

- TWO CYCLE TRANSFER



- FLEXIBLE BUS MAPPING
  - 8 BIT TO 8 BIT
  - 16 BIT TO 16 BIT
  - 8 BIT TO 16 BIT
  - 16 BIT TO 8 BIT
- FLEXIBLE I/O DEVICE SYNCHRONIZATION
  - SOURCE
  - DESTINATION

DMA FACILITIES (CONT.)

- **FLEXIBLE TRANSFER CAPABILITIES**
  - MEMORY TO I/O
  - I/O TO MEMORY
  - MEMORY TO MEMORY
  - I/O TO I/O
- **PERFORM MASKED COMPARES FOR DATA PATTERN AS TRANSFER OCCURS**
  - 8 BIT MASK, 8 BIT COMPARE
- **TRANSLATE DURING TRANSFER**
  - BYTES TRANSLATED THROUGH 256-BYTE LOOKUP TABLE
- **FLEXIBLE TERMINATION CONDITIONS**
  - BYTE COUNT EXPIRED
  - MASKED COMPARE PASSES OR FAILS
  - SINGLE BYTE
  - EXTERNAL SOURCE

8089 PERFORMANCE

DMA TRANSFER	5 MHz	8 MHz
(16-BIT TRANSFERS)	1.25 Mbyte	2.0 Mbyte
DMA BYTE SEARCH 8 BIT/16 BIT SOURCE	0.6125/0.833 Mbyte	1.0/1.33 Mbyte
DMA BYTE TRANSLATE	0.333 Mbyte	0.533 Mbyte
DMA BYTE SEARCH AND TRANSLATE	0.333 Mbyte	0.533 Mbyte
DMA RESPONSE (LATENCY) SINGLE CHANNEL/DUAL CHANNEL	1.0/2.2μs	0.6251/1.375μs

DEVELOPMENT SUPPORT

- ASM89 - 8089 MACRO ASSEMBLER
- LINK86
- LOC86
- LIB86
- RBF89 - REAL TIME BREAKPOINT FACILITY
  SOFTWARE DRIVER THAT USES
  EXISTING ICE 86,88 HARDWARE

WHERE TO FIND MORE INFORMATION

iAPX 86,88 FAMILY UTILITIES USER'S GUIDE
CHAPTER 7 - THE 8089 INPUT/OUTPUT PROCESSOR

APPENDIX A

LAB PROJECTS

I N T R O D U C T I O N   T O   L A B   E X E R C I S E S

The lab exercises you will be doing this week build on each other. You will be using your solution to today's lab to implement tomorrow's lab exercise. These labs are self paced. You are required to complete a minimum part of today's exercise to enable you to continue tomorrow. The minumum requirement is marked in each of the exercises.

If you finish the required part of the lab and have time to spare you can then continue with the optional parts of the lab. In cases where there are several options you should choose the option which is of most interest to you rather than follow the options in the order in which they have been written. These options offer greater detail of subject matter and give examples of the less-often used aspects of the assembly language.

Each day's exercise starts with a description of what you will achieve, then suggests a number of steps to follow in order to complete the exercise. You will have to refer to the ASM86 language reference manual from time to time to find details of how to use instructions. If you have any trouble using this manual, ask your instructor for guidance. Use :F1: for all of your programs. If you are using floppy disks, don't IDISK the user disk since it contains useful code ! If you are working on a network (NDSII), your insructor will tell you how to drive it. Finally, you can use DEBUG (the series III debugger) to debug your code. If you do not know how to use this, ask your instructor for guidance. Good luck !!!

LAB: INTEL TEXT/MATH PROCESSOR

The lab exercise, which you will build on day by day implements a text and math processor. On day one you will write the code to select one of a number of processes (listed below). As the week progresses and you learn more about the capabilities of ASM86, the 8087 and the 80186 you can implement the options offered. You will also be linking your code to a high level language (PL/M).

The processing options are ...

OPTION	NAME	FUNCTION
1	ASCII MATH	Implements math functions on ASCII strings
2	REAL MATH	Implements real number mathematics
3	SALARY BOOSTER	186 exercise to award salary increases
4	unused	

Day 1 Function Selector

INTRODUCTION

In todays lab you will write a program to select one of the processes offered by the text/math processor. You will prompt for a number which will be read from the keyboard and be used to select the appropriate procedure using a branch table. Today all these procedures will do is print a message to indicate that they were properly selected. You will write the options in full in later lab exercises.

PART 1

Your first task is to write a procedure which will print text to the screen (call it PRINT). You should pass the procedure the offset of the message. Use the external (far) procedure CHARACTER_OUT which is provided on your system directory in the file CICOL.OBJ. This procedure expects you to give it a single character passed as a word on the stack. CHARACTER_OUT outputs the character to the screen and will remove it from the stack on return. Terminate your text string with OFFH. Presume that CHARACTER_OUT will destroy all registers except BP and DS. Use LODS to read the character string from memory.

The format of the message string will be, for example ...

```
CR    EQU   ODH
LF    EQU   OAH

MESSAGE    DB    'WELCOME TO THE REAL NUMBERS PROCESSOR',CR,LF
           DB    'This procedure not yet written !',CR,LF,OFFH
```

When you have written the procedure PRINT, write a short program to call it and print a message. Don't forget to set up a stack ! Assemble your program and link it like this .....

```
RUN LINK86 :F1:<YOUR_PROGRAM>.OBJ,CICOL.OBJ,LARGE.LIB BIND
```

PART 2

In your main program, use your PRINT procedure to prompt for an input number. Read the number in using the external (far) procedure CHARACTER_IN which will return a character entered at the keyboard in AL. Use this index number to implement a branch table (ie use an indirect call of the form CALL TABLE[SI] to call the selected option, where TABLE is a table of procedure addresses). Each procedure in the branch table should print a sign-on message (using your PRINT procedure) to indicate that it has been successfully selected. Check that the number entered was not overrange (don't forget that the number will be in ASCII). The last procedure will print 'Oh dear, I selected an invalid process', or similar rebuke of your choice when an overrange selection is entered.

** THIS IS AS FAR AS YOU NEED TO GO TO BE ABLE TO CONTINUE TOMORROW **

PART 3 : use of Ascii Adjust instructions (OPTIONAL)

One of the procedures is an ascii maths processor. To start with, have it prompt for (ie print 'enter a number > ') and input two ascii digit strings using CHARACTER_IN. Add these ascii strings (use AAA) and print the result. If you have time, have the procedure prompt for a function (+, -, *, /) and implement that function using the appropriate AA- instructions. Do not start the multiply and divide options unless you have a lot of time remaining.

Day 2 Linkage with PL/M

PART 1

Yesterday you wrote a program which would write a text string. Today you have been supplied with an executive PL/M program which will link into your program and use it. Small model of PL/M has been used to compile it. The PL/M executive module will first print a message on your screen by calling your PRINT routine, and will then run your program. The PL/M call to your procedure looks like this ....

CALL PRINT (@MESSAGE)

Edit your PRINT procedure to make it PL/M compatible. In doing so, you should remember the following points ...

1/ All of your data segments and your stack segment should be added to DGROUP.
2/ All of your code segments should be added to CGROUP.

3/ Your segment registers should be assumed to be pointing to group bases rather than segment bases. Since the PL/M executive module will load the segment registers, you should not be loading them in your code.

4/ Since in SMALL model of PL/M all data is in a single group (enabling all pointers to be just a 16 bit offset), your message strings must also be in the data group. If your messages are defined in your code segment, use a block move in AEDIT to move them into your data segment which you will add to the data group.

5/ Since PL/M now provides the main module, you will not need to specify the program start address in you END statement

6/ Use the correct type of external procedures (near/far) for SMALL model and make sure that your PUBLIC procedure PRINT is of the correct type (near/far).

The points listed above if not noted will stop your program from working. In addition to this you should use the most appropriate combine type, align type and class names for each of your segments. PL/M dictates these.

Define a structure to describe the stack frame used by your PRINT procedure and use this stucture to access parameters on the stack. When PL/M has welcomed you it will call your program which should now be made into a procedure called ENTIRE_PROGRAM.

In order to link your program ..

```
RUN LINK86 EXECS.OBJ,<YOUR_PROGRAM>.OBJ,CICOS.OBJ,SMALL.LIB BIND &
  TO :F1:MAINS BIND
```

Take a look at the link map.

PART 2

Now edit your program so that you can link it to the large model of PL/M. This time use LDS to read @MESSAGE (a far pointer in large model PL/M) from the stack frame. Has the stack frame changed much ? Edit your stack frame structure to reflect the changes. You will have to change your program quite a lot, so think carefully about near/far procedures, requirements for groups and classes. Remember that you must preserve DS, which you will destroy when you access a pointer from the stack using LDS. Also, since each module in LARGE has it's own data segment, you will assume that DS addresses yours. Unlike DGROUP of SMALL model it is now your job to load DS to match the assumption if you need to use DS to access your segment. Test your editted program by linking to large model programs thus ...

```
RUN LINK86 EXECL.OBJ, YOUR_PROGRAM .OBJ,CICOL.OBJ, LARGE.LIB BIND &
TO :F1:MAINL BIND
```

Take a look at the link map.

PART 3 : modular programming

You have already used external procedures CHARACTER_IN and CHARACTER_OUT. You are now required to write your program in modules. The rest of the programs you will be writing this week will be assembled in seperate modules and linked into the main program you have written thus far. LARGE model of PL/M will be used for the rest of the week.

Use AEDIT to separate out your process procedures (they currently just indicate that the process procedure has been activated) and put them in a separate module. Call your main program :F1:LAB2L1.ASM and the file containing the procedures :F1:LAB2L2.ASM. AEDIT is great for this. If you don't know how best to use it for this purpose, ask your instructor for guidance. The entries in your branch table will now be references to external routines. Will the table implement near or far calls (remember we are compatible with LARGE model PL/M) ? Should the code leading up the indirect jump change ? Assemble and link your component programs and check that everything still works OK.

** this is as far as you need to get to be able to continue with the lab exercise tomorrow **

PART 4 : using LINK86 and LOC86 (OPTIONAL)

Use LINK86 without bind to link together the program modules you used in part 2, then locate your program according to the requirements layed out below ....

INITCODE		at address F000H
CLASS	'CODE'	following on after INITCODE
CLASS	'DATA'	at address 200H (leaving space for interrupt vectors)
SEGMENT	STACK	following class 'DATA'

Take a look at the locate map to see that all is well

PART 5 : Text macros (OPTIONAL)

Large model of PL/M says that each module has it's own data segment. Write a macro which you can use as a header to all of you ASM86 procedures. This macro should push BP, copy SP into BP, push DS then load DS with the data segment you are using in your module. Call it %BEGIN. Write a matching end-of-procedure macro which accepts a parameter to say how many bytes should be removed from the stack by the RET instruction. Try these out with your STRING_READ procedure.

PART 6 : Records (OPTIONAL)

Use a record to represent the MODRM field of an instruction. (see ASM86 language reference manual for the format of instructions. Ask your instructor for guidance if necessary). Using this record, construct simple instructions to replace instructions in your code previously assembled by the assembler. Start with a simple 'MOV reg,immed' and work up to a complex addressing mode. Use DEBUG to dissassemble your code to check it. If you need help, ask your instructor

Day 3 exercises with real numbers

The PL/M executive module also had a real number math function which we could not use until now. You are going to use the 8087 emulator to provide this function. Write this exercise in a seperate module which you can then link with all of the other modules you have used/written so far. You are provided with the means to input real numbers and also print them. The procedures that allow you to do this are contained in the file REAL.OBJ which you will find on your system disk. These programs are written in LARGE model PL/M.

The procedures you can use are as follows ...

```
READ_REAL       ; Prompts for input and returns the number you key in
                ; on the 8087 stack top ST. (no parameters required)

PRINT_REAL      ; Prints a real number on the screen. Pass the number to
                ; the procedure in ST

PRINT_REAL_B     ; As PRINT_REAL, but displays number in binary (short real
                ; format)
```

PART 1 : using the real number procedures

First make sure that you can drive these procedures. Write a procedure with a program loop to read in a number and then display it. Display it in it's binary format too (a simple number like 2.0 is easiest to understand). Don't spend too much time relating the binary format to the decimal number as you are unlikely ever to have to do this in practise. Call your program :F1:LAB31.ASM. Make your procedure a public one and give it the same name as the dummy option you had in LAB212.OBJ. It is the second process of the text/math processor and should be linked into the rest of the processor using the command shown below.

** DON'T FORGET TO CALL INIT87 BEFORE USING THE 8087 EMULATOR ! **

```
SUBMIT :F1:LAB3(1)
```

PART 2 : some real number calculations

Now that you can input and output real numbers you are ready to do some real number calculations. Call your program :F1:LAB32.ASM.

Have your program prompt for a number (ie print 'enter a number ...' on the screen). This number will be used as the length of a simple pendulum (expressed in metres). Calculate and display the period of the pendulum using the formula period = (2*pi*sqrt(l/g)). Period is in seconds. Use FLDPI to read load the value of PI. The value of g (the acceleration due to gravity) is 9.80665. Use a long real format for this number in memory. Store the result in a short real number format. Print your result on the screen and if you have a calculator to hand, see if the result is correct.

To link your object code ... SUBMIT :F1:LAB3(2)

PART 3 : using DCON87 (OPTIONAL)

DCON87 is a useful library for helping you debug programs. It is difficult to decipher those nasty real number bit patterns and DCON87 was used to enable PRINT_REAL to print a readable decimal number on the screen. Rewrite this procedure under a different name and use your procedure to print out your real results. You will need to read the 8087 SUPPORT LIBRARY REFERENCE MANUAL to find out how to do this. The routine which you will use is mqcBIN_DECLOW. This will provide you with a string of ASCII characters which you can then print out in a format of your choice using CHARACTER_OUT. You can use the submit file used above since it's link command includes DCON87.LIB.

Day 4   using the enhanced instructions set of the 80186

This lab will implement the SALARY BOOSTER option of the text/math processor. Do as much as you can in the time available. There is no requirement for you to finish the lab up to a particular point. We do not have a 186 in the development system, so the 186 instructions are going to be emulated using CODEMACROS. Don't be alarmed at the amount of code produced by your 186 instructions. When you come to debug your program, you will see that the code is for a sequence of 8086 instructions that do the same job (a lot less efficiently). To gain access to these CODEMACROS ...

```
$INCLUDE (E186.INC)
```

In this lab you will be calling a PL/M (LARGE model) program which will read a data file from disk containing employee payscale information for a new startup called 'YURE COMPANY'. It only has 7 employees right now. Before commencing you should run a program to initialise the data file on disk ...

```
(RUN) SCALE.
```

This will write the file :F1:SCALE.PAY

PART 1 : awarding an increase (use of IMUL immed, PUSH immed)

You are going to write a 'friendly' program which writes a lot of messages, so before anything else write a text macro which will print a message on the screen when you invoke it . . .

```
%MESSAGE(NAME_OF_MESSAGE)
```

This macro will be very similar to one you saw in class on Tuesday. Use the assembler control $NOGEN to avoid expansion of the macro in your listing. This will make your listing very readable and avoid several lines of code each time you want to type out a message on the screen.

You will be reading a data file from disk. Define a structure to match the format of this data file. It has information as follows ..

employee's first name	10 characters
employee's last name	12 characters
employee' salary	maximum 65 535 pounds

Now set aside storage space for an array of seven such structures. Call a LARGE model PL/M program to fill this array. The program looks like this ...

```
READ_FILE:    PROCEDURE (ARRAY_POINTER,ARRAY_LENGTH) PUBLIC;
    DECLARE   ARRAY_POINTER  POINTER,
              ARRAY_LENGTH   BYTE;
END;
```

The length of the array is the number of employees, not the number of bytes in the array. Use PUSH immediate to pass this value to the procedure.

Print a message to ask which employee (0-6) is to get an increase, then use CHARACTER_IN to read in the reply (remember it will be returned in ASCII). In a similar way, ask for the percentage increase (0-9) to be awarded. Use the employee number to index into the array of structures. To locate this employees salary index into the array by (employee number * type structure). Use IMUL immediate to calculate this index. Having located his salary in this way you can print it out by calling another PL/M procedure which will convert the number to decimal and print it on the screen ...

```
BINOUT:   PROCEDURE (NUMBER) PUBLIC;
     DECLARE   NUMBER WORD;
END;
```

An appropriate message prior to printing the number would be nice, All employees started on 10 000 pounds. Now add the required increase to the salary. Display the new salary together with an appropriate message. Also, write the salary back into the array of structures. In order to update the data file on disk, yet another (LARGE) PL/M procedure has been provided ...

```
WRITE_FILE:     PROCEDURE (ARRAY_POINTER,ARRAY_LENGTH) PUBLIC;
     DECLARE    ARRAY_POINTER  POINTER,
                ARRAY_LENGTH   BYTE;
END;
```

To link your program to everything else you have done so far ...

```
SUBMIT :F1:LAB4
```

This link is getting large and will take a while to do, so check your program carefully before going ahead.

PART 2 : BOUND check

Use the BOUND instruction to check that you have not exceeded the bounds of the array of structures. To do this, precede the array with a bound check of the following form ...

```
BOUND_CHECK     DW   WORKFORCE,(WORKFORCE + SIZE WORKFORCE)-1
WORKFORCE       DW   EMPLOYEE_STRUCTURE 7 DUP (<>)
```

Since you have used BOUND, you should get an interrupt of type 5 if you specify an increase for employee 7 or above. Do you ?

PART 3 : PUSHA, POPA

One of the principle uses of PUSHA will be in an interrupt sevice procedure. Write an interrupt service procedure for the BOUND interrupt (ask your instructor for help if you are not sure how to do this) to print out an error message. Since printing a message will destroy registers, use PUSHA and POPA to safequard registers.

PART 4 : SHIFT/ROTATE immed

You have finished the salary booster now. Return to the prodedure selection routine you wrote on day 1. You had to multiply your option selection by 4 to index into a table of double words. Now use a single multiple shift instruction to do this.

PART 4 : ENTER, LEAVE

These instructions are quite complex. They will typically be used by compilers rather than assembly language programmers, though you might want to use them when interfacing to a high level language. Do not attempt this part of the exercise unless you are clear about everything else so far and have a fair amount of time left to spend on the exercise.

Study the ASM86 LANGUAGE REFERENCE MANUAL until you think you understand the instructions. Enter is quite clear and is ideal for languages such as PASCAL, but is overkill for PL/M which does not copy all the old stack frame pointers down from the previous stack frame. Use the ENTER instruction with a nesting level of 0 to provide the front end of the PRINT procedure which you wrote on day 1. Use LEAVE to exit from the procedure.

APPENDIX B

LAB SOLUTIONS

LAB 1 SOLUTION

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB1
OBJECT MODULE PLACED IN :F1:LAB1.OBJ
NO INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                             1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                             2 +1  $DEBUG
                             3             NAME    LAB1
                             4
                             5             EXTRN   CHARACTER_OUT:FAR,CHARACTER_IN:FAR
                             6
----                         7     STACK   SEGMENT STACK
0000 (100                    8             DW      100 DUP (?)
     ????
     )
00C8                         9     TOS     LABEL   WORD
----                        10     STACK   ENDS
                            11
----                        12     DATA    SEGMENT
                            13     ;
                            14     ;       MESSAGES ....
                            15     ;
  000D                      16     CR      EQU     0DH
  000A                      17     LF      EQU     0AH
  00FF                      18     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                            19
0000 0D                     20     GREETING        DB      CR,LF,'WELCOME TO THE REAL NUMBERS PROCESSOR',CR,LF
0001 0A
0002 57454C434F4D45
     20544F20544845
     205245414C204E
     554D4245525320
     50524F43455353
     4F52
0027 0D
0028 0A
0029 54686973207072     21                     DB      'This procedure not yet written !',CR,LF,LAST
     6F636564757265
     206E6F74207965
     74207772697474
     656E2021
0049 0D
004A 0A
004B FF
                            22
004C 0D                     23     PROMPT          DB      CR,LF,LF,'ENTER PROCESSING OPTION ....',CR,LF,LAST
004D 0A
004E 0A
004F 454E5445522050
     524F4345535349
     4E47204F505449
     4F4E202E2E2E2E
006B 0D
006C 0A
006D FF
```

```
LOC  OBJ                  LINE     SOURCE

                            24
006E 4F5054494F4E20         25     OPTION1M        DB      'OPTION 1 HERE !',CR,LF,LAST
     31204845524520
     21
007D 0D
007E 0A
007F FF
0080 4F5054494F4E20         26     OPTION2M        DB      'OPTION 2 HERE !',CR,LF,LAST
     32204845524520
     21
008F 0D
0090 0A
0091 FF
0092 4F5054494F4E20         27     OPTION3M        DB      'OPTION 3 HERE !',CR,LF,LAST
     33204845524520
     21
00A1 0D
00A2 0A
00A3 FF
00A4 4F5054494F4E20         28     OPTION4M        DB      'OPTION 4 HERE !',CR,LF,LAST
     34204845524520
     21
00B3 0D
00B4 0A
00B5 FF
00B6 594F5520524541         29     ERRORM          DB      'YOU REALLY         THAT ONE UP !!!',CR,LF,LAST
     4C4C5920534352
     45574544205448
     4154204F4E4520
     55502021212l
00D8 0D
00D9 0A
00DA FF
                            30
----                        31     DATA    ENDS
                            32
----                        33     CODE1   SEGMENT
                            34             ASSUME  CS:CODE1,DS:DATA,SS:STACK
                            35
0000                        36     PRINT_STRING    PROC
                            37     ;       Procedure to print a text string. The text string will be
                            38     ; terminated with 0FFh and a near pointer to it will be passed on
                            39     ; the stack
                            40     ;
0000 55                     41             PUSH    BP              ; SAVE OLD STACK MARKER
0001 8BEC                   42             MOV     BP,SP           ; LOAD NEW STACK BASE POINTER
0003 8B7604                 43             MOV     SI,[BP]+4       ; READ OFFSET OF STRING FROM STACK
0006 AC                     44     NEXT:   LODSB                   ; FETCH NEXT CHARACTER
0007 3CFF                   45             CMP     AL,LAST         ; CHECK FOR LAST CHARACTER
0009 740A                   46             JE      EXIT            ;  AND EXIT IF SO
000B 56                     47             PUSH    SI              ; IN CASE CHARACTER_OUT DESTROYS IT
000C 50                     48             PUSH    AX              ; PASS CHARACTER TO CHARACTER_OUT
000D 9A0000----    E        49             CALL    CHARACTER_OUT   ;  AND PRINT THE CHARACTER
0012 5E                     50             POP     SI              ; RESTORE POINTER TO CHARACTER STRING
0013 EBF1                   51             JMP     NEXT            ; REPEAT FOR NEXT CHARACTER
```

```
LOC  OBJ                  LINE     SOURCE

0015 5D                     52     EXIT:   POP     BP
0016 C20200                 53             RET     2               ; RETURN AND REMOVE NEAR POINTER FROM STACK
                            54
                            55     PRINT_STRING    ENDP
                            56
0019                        57     OPTION1P        PROC
0019 8D066E00               58             LEA     AX,OPTION1M     ;
001D 50                     59             PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
001E E8DFFF                 60             CALL    PRINT_STRING
0021 C3                     61             RET
                            62     OPTION1P        ENDP
                            63
0022                        64     OPTION2P        PROC
0022 8D068000               65             LEA     AX,OPTION2M     ;
0026 50                     66             PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0027 E8D6FF                 67             CALL    PRINT_STRING
002A C3                     68             RET
                            69     OPTION2P        ENDP
                            70
002B                        71     OPTION3P        PROC
002B 8D069200               72             LEA     AX,OPTION3M     ;
002F 50                     73             PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0030 E8CDFF                 74             CALL    PRINT_STRING
0033 C3                     75             RET
                            76     OPTION3P        ENDP
                            77
0034                        78     OPTION4P        PROC
0034 8D06A400               79             LEA     AX,OPTION4M     ;
0038 50                     80             PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0039 E8C4FF                 81             CALL    PRINT_STRING
003C C3                     82             RET
                            83     OPTION4P        ENDP
                            84
003D                        85     ERROR           PROC
003D 8D068600               86             LEA     AX,ERRORM
0041 50                     87             PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0042 E8BBFF                 88             CALL    PRINT_STRING
0045 C3                     89             RET
                            90     ERROR           ENDP
                            91
0046 3D00                   92     BTABLE  DW      ERROR,OPTION1P,OPTION2P,OPTION3P,OPTION4P
0048 1900
004A 2200
004C 2B00
004E 3400
                            93
0050 B8----          R      94     START:  MOV     AX,DATA         ; LOAD DS
0053 8ED8                   95             MOV     DS,AX           ;  AS ASSUMED
0055 B8----          R      96             MOV     AX,STACK        ; LOAD SS
0058 8ED0                   97             MOV     SS,AX           ;  AS ASSUMED
005A 8D26C800        R      98             LEA     SP,TOS          ; INITIALISE STACK POINTER
                            99
005E 8D060000              100             LEA     AX,GREETING     ; PASS POINTER TO
0062 50                    101             PUSH    AX              ;   MESSAGE
0063 E89AFF                102             CALL    PRINT_STRING    ;     AND PRINT IT
```

```
LOC  OBJ                  LINE     SOURCE

                           103
0066 8D064C0C              104     AGAIN:  LEA     AX,PROMPT         ; PRINT
006A 50                    105             PUSH    AX                ;   MESSAGE
006B E892FF                106             CALL    PRINT_STRING      ;    TO INPUT SELECTION
                           107
006E 9A0000----      E     108             CALL    CHARACTER_IN      ; READ PROCESSING OPTION FROM KEYBOARD
0073 2C30                  109             SUB     AL,'0'            ; REMOVE ASCII OFFSET FROM CHARACTER
                           110
0075 3C04                  111             CMP     AL,(LENGTH BTABLE)-1    ; TEST FOR OVERRANGE
0077 7602                  112             JBE     INRANGE
0079 32C0                  113             XOR     AL,AL             ; ERROR ROUTINE IS OPTION 0
                           114
007B 32E4                  115     INRANGE: XOR    AH,AH             ; EXTEND SELECTION NUMBER TO 16 BITS
007D D1E0                  116             SHL     AX,1              ; DOUBLE, SINCE TABLE CONTAINS WORDS
007F 8BD8                  117             MOV     BX,AX             ; SINCE AX IS NOT AN INDEX REGISTER
0081 2EFF5746              118             CALL    BTABLE[BX]        ; CALL TO SELECTED ROUTINE
0085 EBDF                  119             JMP     AGAIN
                           120
----                       121     CODE1   ENDS
                           122
                           123             END     START

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

LAB 2 PART 1 SOLUTION

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB2_SMALL
OBJECT MODULE PLACED IN :F1:LAB2S.OBJ
 INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                             1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                             2 +1  $DEBUG
                             3             NAME    LAB2_SMALL
                             4
                             5             EXTRN   CHARACTER_OUT:NEAR,CHARACTER_IN:NEAR
                             6             PUBLIC  PRINT_STRING,ENTIRE_PROGRAM
                             7
                             8     CGROUP  GROUP   CODE1
                             9     DGROUP  GROUP   DATA,STACK
                            10
                            11
----                        12     STACK   SEGMENT STACK 'STACK'
0000 (100                   13             DW      100 DUP (?)
     ????
     )
----                        14     STACK   ENDS
                            15
----                        16     DATA    SEGMENT 'DATA'
                            17     ;
                            18     ;       MESSAGES ....
                            19     ;
  000D                      20     CR      EQU     0DH
  000A                      21     LF      EQU     0AH
  00FF                      22     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                            23
0000 0D                     24     GREETING        DB      CR,LF,'WELCOME TO THE REAL NUMBERS PROCESSOR',CR,LF
0001 0A
0002 57454C434F4D45
     20544F20544845
     205245414C204E
     554D4245525320
     50524F43455353
     4F52
0027 0D
0028 0A
0029 54686973207072         25                     DB      'This procedure not yet written !',CR,LF,LAST
     6F636564757265
     206E6F74207965
     74207772697474
     656E2021
0049 0D
004A 0A
004B FF
                            26
004C 0D                     27     PROMPT          DB      CR,LF,LF,'ENTER PROCESSING OPTION ....',CR,LF,LAST
004D 0A
 04E 0A
 04F 454E5445522050
     524F4345535349
     4E47204F505449
```

```
LOC  OBJ                  LINE     SOURCE

     4F4E202E2E2E2E
006B 0D
006C 0A
006D FF
                            28
006E 4F5054494F4E20         29     OPTION1M       DB      'OPTION 1 HERE !',CR,LF,LAST
     31204845524520
     21
007D 0D
007E 0A
007F FF
0080 4F5054494F4E20         30     OPTION2M       DB      'OPTION 2 HERE !',CR,LF,LAST
     32204845524520
     21
008F 0D
0090 0A
0091 FF
0092 4F5054494F4E20         31     OPTION3M       DB      'OPTION 3 HERE !',CR,LF,LAST
     33204845524520
     21
00A1 0D
00A2 0A
00A3 FF
00A4 4F5054494F4E20         32     OPTION4M       DB      'OPTION 4 HERE !',CR,LF,LAST
     34204845524520
     21
00B3 0D
00B4 0A
00B5 FF
00B6 594F5520524541         33     ERRORM         DB      'YOU REALLY         THAT ONE UP !!!',CR,LF,LAST
     4C4C5920534352
     45574544205448
     4154204F4E4520
     55502021212121
00D8 0D
00D9 0A
00DA FF
                            34
----                        35     DATA    ENDS
                            36
----                        37     CODE1   SEGMENT BYTE    'CODE'
                            38             ASSUME  CS:CGROUP,DS:DGROUP,SS:DGROUP
                            39
                            40
0000                        41     PRINT_STRING    PROC
                            42     ;       Procedure to print a text string. The text string will be
                            43     ; terminated with 0FFh and a near pointer to it will be passed on
                            44     ; the stack
                            45     ;
----                        46     FRAME   STRUC
0000                        47     OLD_BP  DW      ?
0002                        48     RET_OFF DW      ?
0004                        49     STRING  DW      ?
----                        50     FRAME   ENDS
                            51
```

```
LOC  OBJ                  LINE     SOURCE

0000 55                     52              PUSH    BP              ; SAVE OLD STACK MARKER
0001 8BEC                   53              MOV     BP,SP           ; LOAD NEW STACK BASE POINTER
0003 1E                     54              PUSH    DS              ; I NEED IT FOR LODS
0004 8B7604                 55              MOV     SI,[BP].STRING  ; READ OFFSET OF STRING FROM STACK
0007 AC                     56     NEXT:    LODSB                   ; FETCH NEXT CHARACTER
0008 3CFF                   57              CMP     AL,LAST         ; CHECK FOR LAST CHARACTER
000A 7408                   58              JE      EXIT            ;  AND EXIT IF SO
000C 56                     59              PUSH    SI              ; IN CASE CHARACTER_OUT DESTROYS IT
000D 50                     60              PUSH    AX              ; PASS CHARACTER TO CHARACTER_OUT
000E E80000          E      61              CALL    CHARACTER_OUT   ;  AND PRINT THE CHARACTER
0011 5E                     62              POP     SI              ; RESTORE POINTER TO CHARACTER STRING
0012 EBF3                   63              JMP     NEXT            ; REPEAT FOR NEXT CHARACTER
0014 1F                     64     EXIT:    POP     DS
0015 5D                     65              POP     BP
0016 C20200                 66              RET     2               ; RETURN AND REMOVE NEAR POINTER FROM STACK
                            67
                            68     PRINT_STRING     ENDP
                            69
0019                        70     OPTION1P         PROC
0019 8D066E00        R      71              LEA     AX,OPTION1M     ;
001D 50                     72              PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
001E E8DFFF                 73              CALL    PRINT_STRING
0021 C3                     74              RET
                            75     OPTION1P         ENDP
                            76
0022                        77     OPTION2P         PROC
0022 8D068000        R      78              LEA     AX,OPTION2M     ;
0026 50                     79              PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0027 E8D6FF                 80              CALL    PRINT_STRING
002A C3                     81              RET
                            82     OPTION2P         ENDP
                            83
002B                        84     OPTION3P         PROC
002B 8D069200        R      85              LEA     AX,OPTION3M     ;
002F 50                     86              PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0030 E8CDFF                 87              CALL    PRINT_STRING
0033 C3                     88              RET
                            89     OPTION3P         ENDP
                            90
0034                        91     OPTION4P         PROC
0034 8D06A400        R      92              LEA     AX,OPTION4M     ;
0038 50                     93              PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0039 E8C4FF                 94              CALL    PRINT_STRING
003C C3                     95              RET
                            96     OPTION4P         ENDP
                            97
003D                        98     ERROR            PROC
003D 8D06B600        R      99              LEA     AX,ERRORM
0041 50                    100              PUSH    AX              ; PASS OFFSET OF POINTER TO MESSAGE
0042 E8BBFF                101              CALL    PRINT_STRING
0045 C3                    102              RET
                           103     ERROR            ENDP
                           104
0046 3D00            R     105     BTABLE  DW       ERROR,OPTION1P,OPTION2P,OPTION3P,OPTION4P
0048 1900
```

```
LOC  OBJ               LINE     SOURCE

004A 2200
004C 2B00
004E 3400
                        106
0050                    107     ENTIRE_PROGRAM  PROC
                        108
0050 8D060000      R    109             LEA     AX,GREETING     ; PASS POINTER TO
0054 50                 110             PUSH    AX              ;   MESSAGE
0055 E8A8FF             111             CALL    PRINT_STRING    ;     AND PRINT IT
                        112
0058 8D064C00      R    113     AGAIN:  LEA     AX,PROMPT       ; PRINT
005C 50                 114             PUSH    AX              ;   MESSAGE
005D E8A0FF             115             CALL    PRINT_STRING    ;    TO INPUT SELECTION
                        116
0060 E80000        E    117             CALL    CHARACTER_IN    ; READ PROCESSING OPTION FROM KEYBOARD
0063 2C30               118             SUB     AL,'0'          ; REMOVE ASCII OFFSET FROM CHARACTER
                        119
0065 3C04               120             CMP     AL,(LENGTH BTABLE)-1    ; TEST FOR OVERRANGE
0067 7602               121             JBE     INRANGE
0069 32C0               122             XOR     AL,AL           ; ERROR ROUTINE IS OPTION 0
                        123
006B 32E4               124     INRANGE: XOR    AH,AH           ; EXTEND SELECTION NUMBER TO 16 BITS
006D D1E0               125             SHL     AX,1            ; DOUBLE, SINCE TABLE CONTAINS WORDS
006F 8BD8               126             MOV     BX,AX           ; SINCE AX IS NOT AN INDEX REGISTER
0071 2EFF974600    R    127             CALL    BTABLE[BX]      ; CALL TO SELECTED ROUTINE
0076 C3                 128             RET
                        129
                        130     ENTIRE_PROGRAM  ENDP
                        131
----                    132     CODE1   ENDS
                        133
                        134             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

LAB 2 PART 3 SOLUTION: 1 of 2

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB2_L_MAIN
OBJECT MODULE PLACED IN :F1:LAB2L1.OBJ
NO INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                             1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                             2 +1  $DEBUG
                             3             NAME    LAB2_L_MAIN
                             4
                             5             EXTRN   CHARACTER_OUT:FAR,CHARACTER_IN:FAR
                             6             EXTRN   ERROR:FAR,OPTION1P:FAR,OPTION2P:FAR,OPTION3P:FAR,OPTION4P:FAR
                             7             PUBLIC  PRINT_STRING,ENTIRE_PROGRAM
                             8
----                         9     STACK   SEGMENT STACK 'STACK'
0000 (100                   10             DW      100 DUP (?)
     ????
     )
----                        11     STACK   ENDS
                            12
----                        13     DATA1   SEGMENT 'DATA'
                            14
----                        15     DATA1   ENDS
                            16
----                        17     CODE1   SEGMENT BYTE    'CODE'
                            18             ASSUME  CS:CODE1,DS:DATA1,SS:STACK
                            19
                            20     ;
                            21     ;       MESSAGES ....
                            22     ;
  000D                      23     CR      EQU     0DH
  000A                      24     LF      EQU     0AH
  00FF                      25     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                            26
0000 0D                     27     GREETING        DB      CR,LF,'WELCOME TO THE OPTION SELECTOR !',CR,LF,LAST
0001 0A
0002 57454C434F4D45
     20544F20544845
     204F5054494F4E
     2053454C454354
     4F522021
0022 0D
0023 0A
0024 FF
                            28
0025 0D                     29     PROMPT          DB      CR,LF,LF,'ENTER PROCESSING OPTION ....',CR,LF,LAST
0026 0A
0027 0A
0028 454E5445522050
     524F4345535349
     4E47204F505449
     4F4E202E2E2E2E
0044 0D
0045 0A
0046 FF
                            30
```

```
LOC  OBJ                LINE     SOURCE

                          31
0047                      32     PRINT_STRING    PROC    FAR
                          33     ;       Procedure to print a text string. The text string will be
                          34     ; terminated with 0FFh and a near pointer to it will be passed on
                          35     ; the stack
                          36     ;
----                      37     FRAME   STRUC
0000                      38     OLD_BP  DW      ?
0002                      39     RET_OFF DW      ?
0004                      40     RET_BASE DW     ?
0006                      41     STRING  DD      ?
----                      42     FRAME   ENDS
                          43
0047 55                   44             PUSH    BP              ; SAVE OLD STACK MARKER
0048 8BEC                 45             MOV     BP,SP           ; LOAD NEW STACK BASE POINTER
004A 1E                   46             PUSH    DS              ; I NEED IT FOR LODS
004B C57606               47             LDS     SI,[BP].STRING  ; READ BASE:OFFSET OF STRING FROM STACK
004E AC                   48     NEXT:   LODSB                   ; FETCH NEXT CHARACTER
004F 3CFF                 49             CMP     AL,LAST         ; CHECK FOR LAST CHARACTER
0051 740A                 50             JE      EXIT            ;  AND EXIT IF SO
0053 56                   51             PUSH    SI              ; IN CASE CHARACTER_OUT DESTROYS IT
0054 50                   52             PUSH    AX              ; PASS CHARACTER TO CHARACTER_OUT
0055 9A0000----     E     53             CALL    CHARACTER_OUT   ;  AND PRINT THE CHARACTER
005A 5E                   54             POP     SI              ; RESTORE POINTER TO CHARACTER STRING
005B EBF1                 55             JMP     NEXT            ; REPEAT FOR NEXT CHARACTER
005D 1F                   56     EXIT:   POP     DS
005E 5D                   57             POP     BP
005F CA0400               58             RET     4               ; RETURN AND REMOVE NEAR POINTER FROM STACK
                          59
                          60     PRINT_STRING    ENDP
                          61
0062 0000----       E     62     BTABLE  DD      ERROR,OPTION1P,OPTION2P,OPTION3P,OPTION4P
0066 0000----
006A 0000----
006E 0000----
0072 0000----
                          63
0076                      64     ENTIRE_PROGRAM  PROC    FAR
                          65
0076 8D060000       R     66             LEA     AX,GREETING     ; PASS POINTER TO
007A 0E                   67             PUSH    CS              ;  GREETING
007B 50                   68             PUSH    AX              ;   MESSAGE
007C 9A4700----     R     69             CALL    PRINT_STRING    ;     AND PRINT IT
                          70
0081 8D062500       R     71     AGAIN:  LEA     AX,PROMPT       ; PRINT
0085 0E                   72             PUSH    CS              ;  PROMPT
0086 50                   73             PUSH    AX              ;   MESSAGE
0087 9A4700----     R     74             CALL    PRINT_STRING    ;    TO INPUT SELECTION
                          75
008C 9A0000----     E     76             CALL    CHARACTER_IN    ; READ PROCESSING OPTION FROM KEYBOARD
0091 2C30                 77             SUB     AL,'0'          ; REMOVE ASCII OFFSET FROM CHARACTER
                          78
0093 3C04                 79             CMP     AL,(LENGTH BTABLE)-1    ; TEST FOR OVERRANGE
0095 7602                 80             JBE     INRANGE
0097 32C0                 81             XOR     AL,AL           ; ERROR ROUTINE IS OPTION 0
```

```
LOC  OBJ                  LINE     SOURCE

                            82
0099 32E4                   83     INRANGE: XOR    AH,AH           ; EXTEND SELECTION NUMBER TO 16 BITS
009B D1E0                   84             SHL     AX,1            ; DOUBLE, SINCE TABLE CONTAINS WORDS
009D D1E0                   85             SHL     AX,1            ;  THEN TWICE FOR DOUBLE WORDS
009F 8BD8                   86             MOV     BX,AX           ; SINCE AX IS NOT AN INDEX REGISTER
00A1 2EFF9F6200      R      87             CALL    BTABLE[BX]      ; CALL TO SELECTED ROUTINE
00A6 CB                     88             RET
                            89
                            90     ENTIRE_PROGRAM  ENDP
                            91
----                        92     CODE1   ENDS
                            93
                            94             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

LAB 2 PART 3 SOLUTION: 2 of 2

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB2_L_PROCS
OBJECT MODULE PLACED IN :F1:LAB2L2.OBJ
NO INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                            1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                            2 +1  $DEBUG
                            3             NAME    LAB2_L_PROCS
                            4
                            5             EXTRN   CHARACTER_OUT:FAR,CHARACTER_IN:FAR,PRINT_STRING:FAR
                            6             PUBLIC  ERROR,OPTION1P,OPTION2P,OPTION3P,OPTION4P
                            7
----                        8     CODE2   SEGMENT 'CODE'
                            9             ASSUME  CS:CODE2
                           10
  000D                     11     CR      EQU     0DH
  000A                     12     LF      EQU     0AH
  00FF                     13     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                           14
0000 4F5054494F4E20        15     OPTION1M        DB      'OPTION 1 HERE !',CR,LF,LAST
     31204845524520
     21
000F 0D
0010 0A
0011 FF
0012 4F5054494F4E20        16     OPTION2M        DB      'OPTION 2 HERE !',CR,LF,LAST
     32204845524520
     21
0021 0D
0022 0A
0023 FF
0024 4F5054494F4E20        17     OPTION3M        DB      'OPTION 3 HERE !',CR,LF,LAST
     33204845524520
     21
0033 0D
0034 0A
0035 FF
0036 4F5054494F4E20        18     OPTION4M        DB      'OPTION 4 HERE !',CR,LF,LAST
     34204845524520
     21
0045 0D
0046 0A
0047 FF
0048 594F5520524541        19     ERRORM          DB      'YOU REALLY SCREWED THAT ONE UP !!!',CR,LF,LAST
     4C4C5920534352
     45574544205448
     4154204F4E4520
     55502021212I
006A 0D
006B 0A
006C FF
                           20
006D                       21     OPTION1P        PROC    FAR
006D 0E                    22             PUSH    CS              ; PASS BASE OF MESSAGE STRING
006E 8D060000              23             LEA     AX,OPTION1M     ;
```

```
LOC OBJ              LINE   SOURCE

0072 50                24            PUSH   AX               ; PASS OFFSET OF POINTER TO MESSAGE
0073 9A0000----    E   25            CALL   PRINT_STRING
0078 CB                26            RET
                       27   OPTION1P        ENDP
                       28
0079                   29   OPTION2P        PROC    FAR
0079 0E                30            PUSH   CS               ; PASS BASE OF MESSAGE STRING
007A 8D061200          31            LEA    AX,OPTION2M      ;
007E 50                32            PUSH   AX               ; PASS OFFSET OF POINTER TO MESSAGE
007F 9A0000----    E   33            CALL   PRINT_STRING
0084 CB                34            RET
                       35   OPTION2P        ENDP
                       36
0085                   37   OPTION3P        PROC    FAR
0085 0E                38            PUSH   CS               ; PASS BASE OF MESSAGE STRING
0086 8D062400          39            LEA    AX,OPTION3M      ;
008A 50                40            PUSH   AX               ; PASS OFFSET OF POINTER TO MESSAGE
008B 9A0000----    E   41            CALL   PRINT_STRING
0090 CB                42            RET
                       43   OPTION3P        ENDP
                       44
0091                   45   OPTION4P        PROC    FAR
0091 0E                46            PUSH   CS               ; PASS BASE OF MESSAGE STRING
0092 8D063600          47            LEA    AX,OPTION4M      ;
0096 50                48            PUSH   AX               ; PASS OFFSET OF POINTER TO MESSAGE
0097 9A0000----    E   49            CALL   PRINT_STRING
009C CB                50            RET
                       51   OPTION4P        ENDP
                       52
009D                   53   ERROR           PROC    FAR
009D 0E                54            PUSH   CS               ; PASS BASE OF MESSAGE STRING
009E 8D064800          55            LEA    AX,ERRORM
00A2 50                56            PUSH   AX               ; PASS OFFSET OF POINTER TO MESSAGE
00A3 9A0000----    E   57            CALL   PRINT_STRING
00A8 CB                58            RET
                       59   ERROR           ENDP
                       60
----                   61   CODE2    ENDS
                       62
                       63            END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

LAB 3 PART 1 SOLUTION

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB3_PART_1
OBJECT MODULE PLACED IN :F1:LAB31.OBJ
NO INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                            1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                            2 +1  $DEBUG
                            3             NAME    LAB3_PART_1
                            4
                            5             EXTRN   CHARACTER_OUT:FAR,CHARACTER_IN:FAR,PRINT_STRING:FAR
                            6             EXTRN   READ_REAL:FAR,PRINT_REAL:FAR,PRINT_REAL_B:FAR,INIT87:FAR
                            7             PUBLIC  OPTION2P
                            8
----                        9     STACK   SEGMENT STACK   'STACK'
0000 (50                   10             DW      50 DUP (?)
     ????
     )
----                       11     STACK   ENDS
                           12
----                       13     DATA3   SEGMENT 'DATA'
----                       14     DATA3   ENDS
                           15
----                       16     CODE3   SEGMENT 'CODE'
                           17             ASSUME  CS:CODE3,DS:DATA3,SS:STACK
                           18     ;
                           19     ;       MESSAGES ....
                           20     ;
  000D                     21     CR      EQU     0DH
  000A                     22     LF      EQU     0AH
  00FF                     23     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                           24
0000 54484953204953        25     GREET   DB      'THIS IS LAB3, THE REAL NUMBERS LAB .',CR,LF,LAST
     204C4142332C20
     54484520524541
     4C204E554D4245
     5253204C414220
     2E
0024 0D
0025 0A
0026 FF
                           26
0027                       27     OPTION2P        PROC    FAR
                           28
0027 8D060000              29             LEA     AX,GREET        ; PRINT
002B 0E                    30             PUSH    CS              ;  GREETING
002C 50                    31             PUSH    AX              ;   MESSAGE
002D 9A0000----      E     32             CALL    PRINT_STRING
0032 9A0000----      E     33             CALL    INIT87
0037 9A0000----      E     34             CALL    READ_REAL       ; READ REAL NUMBER ONTO TOP OF 8087 STACK
003C 9A0000----      E     35             CALL    PRINT_REAL      ; PRINT NUMBER CURRENTLY ON ST
0041 9A0000----      E     36             CALL    PRINT_REAL_B    ; PRINT IT IN BINARY
0046 CB                    37             RET
                           38
                           39     OPTION2P        ENDP
                           40
```

```
LOC  OBJ                  LINE     SOURCE

----                        41     CODE3   ENDS
                            42
                            43             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

LAB 3 PART 2 SOLUTION

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB3_PART_2
OBJECT MODULE PLACED IN :F1:LAB32.OBJ
NO INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                             1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                             2 +1  $DEBUG
                             3             NAME    LAB3_PART_2
                             4
                             5             EXTRN   CHARACTER_OUT:FAR,CHARACTER_IN:FAR,PRINT_STRING:FAR
                             6             EXTRN   READ_REAL:FAR,PRINT_REAL:FAR,PRINT_REAL_B:FAR,INIT87:FAR
                             7             PUBLIC  OPTION2P
                             8
----                         9     STACK   SEGMENT STACK   'STACK'
0000 (50                    10             DW      50 DUP (?)
     ????
     )
----                        11     STACK   ENDS
                            12
----                        13     DATA3   SEGMENT 'DATA'
----                        14     DATA3   ENDS
                            15
----                        16     CODE3   SEGMENT 'CODE'
                            17             ASSUME  CS:CODE3,SS:STACK
                            18     ;
                            19     ;       MESSAGES ....
                            20     ;
  000D                      21     CR      EQU     0DH
  000A                      22     LF      EQU     0AH
  00FF                      23     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                            24
0000 54484953204953         25     GREET   DB      'THIS IS LAB3, THE REAL NUMBERS LAB .',CR,LF,LAST
     204C4142332C20
     54484520524541
     4C204E554D4245
     5253204C414220
     2E
0024 0D
0025 0A
0026 FF
0027 454E544552204C         26     PROMPT  DB      'ENTER LENGTH OF PENDULUM IN METRES ...',CR,LF,LAST
     454E475448204F
     462050454E4455
     4C554D20494E20
     4D455452455320
     2E2E2E
004D 0D
004E 0A
004F FF
0050 0D                     27     RESULT  DB      CR,LF,'PERIOD OF PENDULUM IS ',LAST
0051 0A
0052 504552494F4420
     4F462050454E44
     554C554D204953
     20
```

```
8086/8087/8088 MACRO ASSEMBLER   SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES        09:07:46  06/01/84  PAGE    2


LOC  OBJ                  LINE     SOURCE

0068 FF
0069 205345434F4E44         28     UNIT     DB       ' SECONDS',CR,LF,LAST
     53
0071 0D
0072 0A
0073 FF
                            29
0074 05A3923A019D23         30     G        DQ       9.80665          ; ACCELERATION DUE TO GRAVITY
     40
007C 0000000000000          31     TWO      DQ       2.0
     40
                            32
0084                        33     OPTION2P          PROC    FAR
                            34
0084 8D060000               35              LEA      AX,GREET         ; PRINT
0088 0E                     36              PUSH     CS               ;  GREETING
0089 50                     37              PUSH     AX               ;   MESSAGE
008A 9A0000----       E     38              CALL     PRINT_STRING
008F 9A0000----       E     39              CALL     INIT87
                            40
0094 8D062700               41              LEA      AX,PROMPT        ; ASK
0098 0E                     42              PUSH     CS               ;  FOR
0099 50                     43              PUSH     AX               ;   LENGTH
009A 9A0000----       E     44              CALL     PRINT_STRING     ;    OF PENDULUM
                            45
009F 9A0000----       E     46              CALL     READ_REAL        ; READ REAL NUMBER ONTO TOP OF 8087 STACK
                            47
00A4 9B2EDC367400           48              FDIV     G                ; ST = L/G
00AA 9BD9FA                 49              FSQRT                     ; ST = SQRT(L/G)
00AD 9BD9EB                 50              FLDPI                     ; ST = PI, ST(1) = SQRT(L/G)
00B0 9BDEC9                 51              FMUL                      ; ST = PI * SQRT(L/G)
00B3 9B2EDC0E7C00           52              FMUL     TWO              ; ST = 2 * PI * SQRT(L/G)
                            53
00B9 8D065000               54              LEA      AX,RESULT        ; PRINT
00BD 0E                     55              PUSH     CS               ;  START
00BE 50                     56              PUSH     AX               ;   OF RESULT
00BF 9A0000----       E     57              CALL     PRINT_STRING     ;    MESSAGE
00C4 9A0000----       E     58              CALL     PRINT_REAL       ; PRINT NUMBER CURRENTLY ON ST
00C9 8D066900               59              LEA      AX,UNIT          ; PRINT
00CD 0E                     60              PUSH     CS               ;  END
00CE 50                     61              PUSH     AX               ;   OF RESULT
00CF 9A0000----       E     62              CALL     PRINT_STRING     ;    MESSAGE
                            63
00D4 CB                     64              RET
                            65
                            66     OPTION2P          ENDP
                            67
----                        68     CODE3    ENDS
                            69
                            70              END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

LAB 4 SOLUTION

```
SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.0 ASSEMBLY OF MODULE LAB4
OBJECT MODULE PLACED IN :F1:LAB4.OBJ
NO INVOCATION LINE CONTROLS


LOC  OBJ                  LINE     SOURCE

                             1 +1  $TITLE ('SOLUTIONS TO IAPX86/88/186 PART II LAB EXERCISES ')
                             2 +1  $DEBUG
                             3 +1  $INCLUDE (E186.INC)
                       =1    4 +1  $SAVE
                       =1    5 +1  $NOLIST
                           348 +1  $NOGEN
                           349
                           350             NAME    LAB4
                           351
                           352             EXTRN   CHARACTER_OUT:FAR,CHARACTER_IN:FAR,PRINT_STRING:FAR
                           353             EXTRN   READ_REAL:FAR,PRINT_REAL:FAR,PRINT_REAL_B:FAR,INIT87:FAR
                           354             EXTRN   READ_FILE:FAR,WRITE_FILE:FAR,BINOUT:FAR
                           355             PUBLIC  OPTION3P
                           356
                           357     %*DEFINE (MESSAGE(NAME))(
                                           LEA     AX,%NAME
                                           PUSH    CS
                                           PUSH    AX
                                           CALL    PRINT_STRING
                                           )
                           358
                           359
----                       360     STACK   SEGMENT STACK   'STACK'
0000 (50                   361             DW      50 DUP (?)
     ????
     )
----                       362     STACK   ENDS
                           363
----                       364     EMPLOYEE        STRUC
0000                       365     FIRST_NAME      DB 10 DUP (?)
000A                       366     LAST_NAME       DB 12 DUP (?)
0016                       367     PAY             DW      ?
----                       368     EMPLOYEE        ENDS
                           369
----                       370     DATA4   SEGMENT 'DATA'
                           371
0000 0400                  372     BOUND_CHECK     DW      WORKFORCE,(WORKFORCE + SIZE WORKFORCE)-1
0002 AB00
0004 (7                    373     WORKFORCE       EMPLOYEE 7 DUP (<>)
      (10
     ??
      )
      (12
     ??
      )
     ????
     )
                           374
-----                      375     DATA4   ENDS
                           376     $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           377
----                       378     CODE4   SEGMENT 'CODE'
                           379             ASSUME  CS:CODE4,DS:DATA4,SS:STACK
                           380     ;
                           381     ;       MESSAGES ....
                           382     ;
  000D                     383     CR      EQU     0DH
  000A                     384     LF      EQU     0AH
  00FF                     385     LAST    EQU     0FFH            ; LAST CHARACTER MARKER
                           386
0000 0D                    387     INCREASE?       DB      CR,LF,' HOW MANY PERCENT INCREASE ? ',LAST
0001 0A
0002 20484F57204D41
     4E592050455243
     454E5420494E43
     5245415345203F
     20
001F FF
0020 0D                    388     WHO?            DB      CR,LF,LF,'AND WHICH EMPLOYEE IS THE LUCKY RECIPIENT ?'
0021 0A
0022 0A
0023 414E4420574849
     434820454D504C
     4F594545204953
     20544845204C55
     434B5920524543
     495049454E5420
     3F
004E FF                    389                     DB      LAST
004F 0D                    390     CURRENT?        DB      CR,LF,'..... CURRENT SALARY IS ',LAST
0050 0A
0051 2E2E2E2E2E2043
     555252454E5420
     53414C41525920
     495320
0069 FF
006A 20504F554E4453        391     CURRENCY        DB      ' POUNDS',CR,LF,LAST
0071 0D
0072 0A
0073 FF
0074 2E2E2E2E2E2E2E        392     NEW             DB      '....... NEW SALARY IS ',LAST
     204E4557205341
     4C415259204953
     20
008A FF
008B FF                    393                     DB      LAST
                           394
                           395     $EJECT
```

```
LOC  OBJ                 LINE     SOURCE

                          396
008C                      397     OPTION3P        PROC FAR
                          398
008C 1E                   399             PUSH    DS              ; MY PROGRAM HAS IT'S OWN DATA SEGMENT
008D B8----         R     400             MOV     AX,DATA4        ; LOAD DS
0090 8ED8                 401             MOV     DS,AX           ;  TO MATCH ASSUME
                          402
0092 8D060400             403             LEA     AX,WORKFORCE    ; PASS POINTER
0096 1E                   404             PUSH    DS              ;  TO PAYSCALES
0097 50                   405             PUSH    AX              ;   ARRAY
0098 555589E5C74602       406             PUSH    LENGTH WORKFORCE ; PASS LENGTH OF ARRAY
     07005D
00A2 9A0000----     E     407             CALL    READ_FILE       ; CALL PL/M PROGRAM TO READ FILE FROM DISK
                          408
                          409             %MESSAGE(INCREASE?)
                    E     415
00B2 9A0000----     E     416             CALL    CHARACTER_IN    ; READ INCREASE FROM KEYBOARD
00B7 2C30                 417             SUB     AL,'0'          ;  REMOVE ASCII OFFSET
00B9 50                   418             PUSH    AX              ;   AND SAVE IT ON THE STACK
                          419
                          420             %MESSAGE(WHO?)
                    E     426
00C5 9A0000----     E     427             CALL    CHARACTER_IN    ; READ EMPLOYEE NUMBER FROM KEYBOARD
00CA 2C30                 428             SUB     AL,'0'          ;  AND REMOVE ASCII OFFSET
                          429
00CC FFF05589E55052       430             IMUL    AX,TYPE WORKFORCE       ; AX WILL BE INDEX INTO ARRAY OF STRUCTURES
     B81800F76E0287
     46025A585D58
00E0 051A00               431             ADD     AX,OFFSET WORKFORCE.PAY ; AX IS NOW OFFSET TO NTH PAY
00E3 8BD8                 432             MOV     BX,AX           ; VALID INDEX REGISTER
                          433
00E5 505389D88B1E00       434             BOUND   BX,BOUND_CHECK  ; CHECK ARRAY BOUNDS
     0039077602CD05
     81C30200390773
     02CD055B58
                          435
00FF 53                   436             PUSH    BX
                          437             %MESSAGE(CURRENT?)
010B 5B             E     443             POP     BX
                          444
010C 53                   445             PUSH    BX
                          446
010D 8B07                 447             MOV     AX,[BX]         ; FETCH CURRENT SALARY
010F 50                   448             PUSH    AX              ; DISPLAY CURRENT
0110 9A0000----     E     449             CALL    BINOUT          ;  SALARY
                          450             %MESSAGE(CURRENCY)
                    E     456
0120 5B                   457             POP     BX
                          458
0121 59                   459             POP     CX              ; POP INCREASE INTO CL
0122 32ED                 460             XOR     CH,CH           ;  AND ADD LEADING ZEROS
0124 8B07                 461             MOV     AX,[BX]         ; FETCH CURRENT SALARY
0126 F7E1                 462             MUL     CX              ; MULTIPLY ORIGINAL SALARY BY INCREASE
0128 B96400               463             MOV     CX,100          ; DIVIDE BY
012B F7F1                 464             DIV     CX              ;  100 FOR PERCENT
```

```
LOC  OBJ                  LINE     SOURCE

012D 0107                  465             ADD     [BX],AX          ; ADD INCREASE TO SALARY
                           466
012F FF37                  467             PUSH    WORD PTR [BX]    ; NEW SALARY ON STACK FOR BINOUT
                           468             %MESSAGE(NEW)
013C 9A0000----       E    474             CALL    BINOUT           ;  SALARY
                           475             %MESSAGE(CURRENCY)
                      E    481
014C 8D060400              482             LEA     AX,WORKFORCE     ; PASS POINTER
0150 1E                    483             PUSH    DS               ;  TO PAYSCALES
0151 50                    484             PUSH    AX               ;   ARRAY
0152 555589E5C74602        485             PUSH    LENGTH WORKFORCE ; PASS LENGTH OF ARRAY
     07005D
015C 9A0000----       E    486             CALL    WRITE_FILE       ; WRITE ARRAY BACK ONTO DISK
                           487
0161 1F                    488             POP     DS
0162 CB                    489             RET
                           490
                           491     OPTION3P        ENDP
                           492
----                       493     CODE4   ENDS
                           494
                           495             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX C

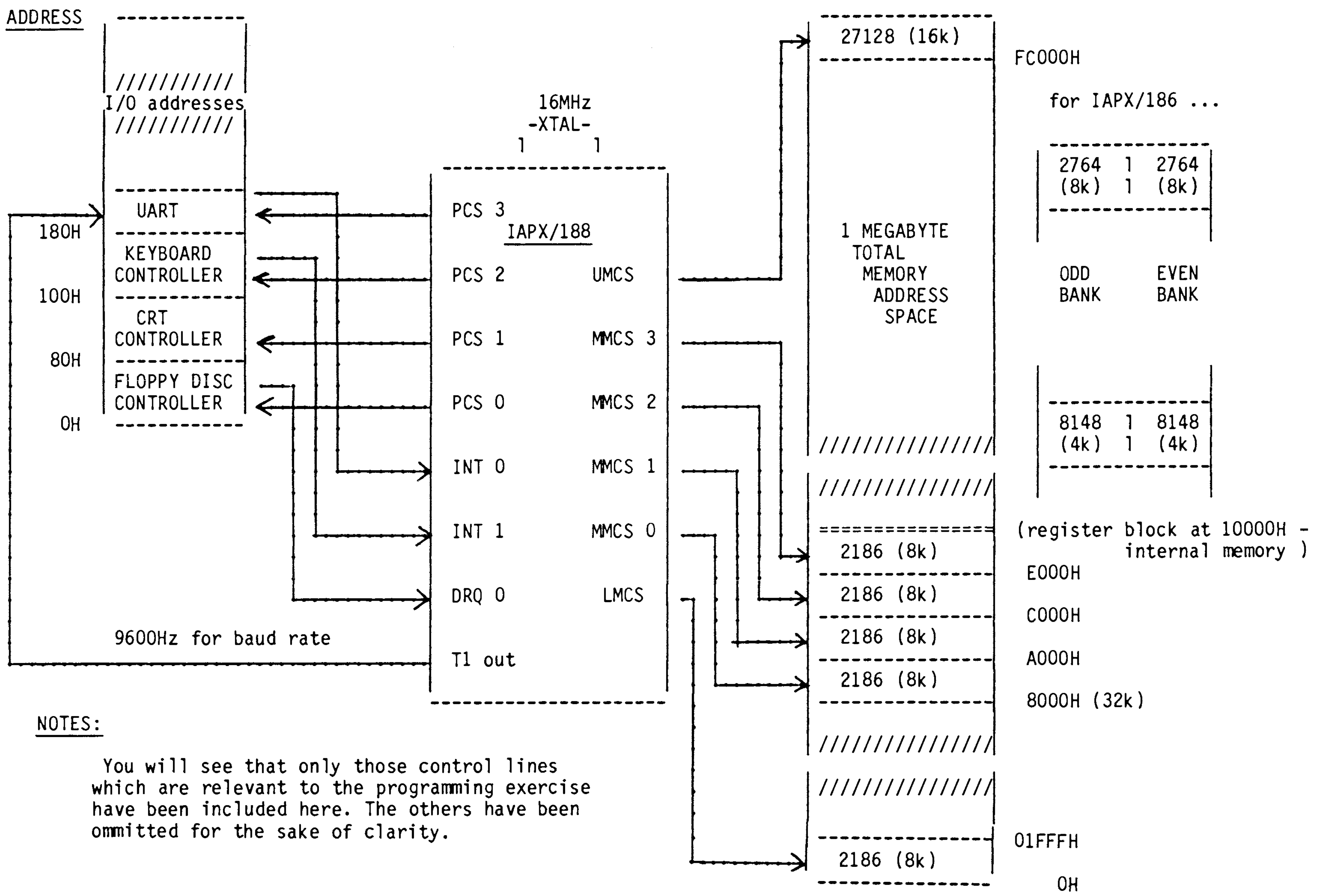
80/88 DESIGN EXAMPLE

iAPX/186 APPLICATION EXAMPLE

In this appendix you have a diagram showing how the 186 (or 188, they look the same from a software standpoint) could be used as the basis for a small business computer. It shows how the various peripherals are connected up, both in terms of addressing and in how they utilize the 186 via interrupts and DMA. The memory address mapping is also shown. In this appendix, which you will fill in as you learn the various functions of the 186, you are going to set up the 186 to handle this computer.

Because we made the 186 versatile it has many options on how to use the interrupts, timers, DMA controllers, etc. As you work through this appendix you will appreciate that there is a lot of work to do in getting all the right bits into the right control registers. Fortunately, this is something which to a large extent you program once (for a given hardware configuration) and that's the 186 set up for your system. There are also status registers which allow you to monitor the state of the various internal peripherals and command registers for run-time control of these peripherals.

IAPX/188 APPLICATION - SMALL BUSSINESS COMPUTER



NOTES:

You will see that only those control lines which are relevant to the programming exercise have been included here. The others have been ommitted for the sake of clarity.

SETTING UP REGISTER BLOCK AND CHIP SELECT LOGIC

1) Locate the register block at location 10000H in memory space, enable trapping of escape codes and set the interrupt controller into normal mode.

```
REG_BLOCK   SEGMENT AT 10000H
TABLE       LABEL   WORD
; set segment aside for control register block
REG_BLOCK   ENDS

CODE_1      SEGMENT
            ASSUME CS:CODE_1,DS:REG_BLOCK

            DEFAULT EQU     OFF00H
START:      MOV     AX,REG_BLOCK
            MOV     DS,AX
            MOV     DX,DEFAULT+OFEH
            MOV     AX,1001000100000000B
            OUT     DX,AX
```

2) Program your upper memory chip select. Your memory needs 1 wait state and no external ready synchronisation is required.

```
    MOV     TABLE+OAOH,_______________________ ;PROGRAM UMCS
```

3) Program your lower memory chip select. No wait states are required and no external ready synch is needed.

```
    _________________________________________ ;PROGRAM LMCS
```

4) Program the mid range chip selects. No wait states are required, nor is external ready synch needed. Leave the bits for the peripheral chips blank. We'll return to them later.

```
    _________________________________________ ;PROGRAM MMCS
    _________________________________________ ;PROGRAM MPCS
```

(... continued)

5) Now program the peripheral chip selects. Your peripherals should be I/O mapped and each requires two wait states and no external ready synch. You need address lines A1 and A2 from PCS5,6. Go back and fill in the rest of the MPCS bits from part 4.

```
____________________________________ ;PROGRAM PACS
```

SETTING UP THE TIMERS

TIMER 0

This is being used as a straight 16 bit divider to reduce the crystal frequency to a baud rate of 9600. A square wave is required, so use both count A and count B registers. No interrupt is required on terminal count.

$$\begin{tikzcd}\text{XTAL(16MHZ)} \arrow[r] & \boxed{\div 2} \arrow[r] & \boxed{\div 4} \arrow[r] & \boxed{\text{TIMER 0 (x1/???)}} \arrow[r] & \text{9600 Hz}\end{tikzcd}$$

TIMER 2

This timer is being used as a prescaler to divide by ???? . A single count register will be used (we have no option about this), and no interrupt on terminal count is required.

$$\begin{tikzcd}\text{XTAL(16MHZ)} \arrow[r] & \boxed{\div 2} \arrow[r] & \boxed{\div 4} \arrow[r] & \boxed{\text{TIMER 2 (x1/????)}} \arrow[r] & \text{1KHz}\end{tikzcd}$$

TIMER 1

A real time clock interrupt is to be generated from this timer. It is to be fed from the timer 2 prescaler and is to produce an interrupt every 1 second

$$\begin{tikzcd}\text{TIMER 2} \arrow[r] & \text{1KHz} \arrow[r] & \boxed{\text{TIMER 1 (x1/????)}} \arrow[r] & \text{1Hz} \arrow[r] & \text{interrupt}\end{tikzcd}$$

continued ....

SETTING UP THE TIMERS (continued)

On the previous sheet I explained the operating modes required of the three timers. The timer control registers appear in contiguous locations inside the control register block. It might make sense to take advantage of this fact. To this end, the basis of a block move solution for loading the registers is suggested here. It is neater than loading all of the registers one at a time using in-line code. You still have to do the nasty bit-picking for some of the registers, but at least you only have to do it once (provided that you got it right !).

```
CODE_1    SEGMENT
          ASSUME  CS:CODE_1,ES:REG_BLOCK

          MOV     AX,REG_BLOCK     : ADDRESS REGISTER
          MOV     ES,AX            :  BLOCK WITH
          LEA     DI,TABLE+50H     ;   ES:DI (includes offset to
                                   ;    first timer control register)

          LEA     SI,PROG_TIME     ; ADDRESS TABLE OF BIT PATTERNS (below)
          MOV     CX,12            ; COUNT OF REGISTERS TO LOAD
      REP MOVS    TABLE,PROG_TIME  ; ASSEMBLE WILL GIVE CS: OVERRIDE
                                   :  TO ACCESS PROG_TIME
          etc....

PROG_TIME: DW     ______________   ; TMR 0 COUNT REGISTER
          DW      ______________   ;       MAX COUNT A
          DW      ______________   ;       MAX COUNT B
          DW      ______________   ;       MODE/CONTROL WORD
          DW      ______________   ; TMR 1 COUNT REGISTER
          DW      ______________   ;       MAX COUNT A
          DW      ______________   ;       MAX COUNT B
          DW      ______________   ;       MODE/CONTROL WORD
          DW      ______________   ; TMR 2 COUNT REGISTER
          DW      ______________   ;       MAX COUNT A
          DW      ______________   ;       THERE IS NO MAX COUNT B
          DW      ______________   ;       MODE/CONTROL WORD
```

SETTING UP THE DMA CONTROL BLOCK

You have just located a 128 byte sector on a floppy disc and wish to DMA the data into memory at the address C000H onward. The floppy disk will synchronise the transfer. Use DMA channel 0 and provide an interrupt when the transfer is complete. Give this DMA channel high priority. You will be transferring bytes and the transfer is to start immediately. Use a block move method for loading the registers like the example given for loading the timer control registers.

```
CODE_1    SEGMENT
          ASSUME  ______________________

          _________________________;_________________________________
          _________________________;_________________________________
          _________________________;_________________________________
          _________________________;_________________________________
          _________________________;_________________________________

     REP  MOVS    TABLE,PROG_DMA   ; ASSEMBLER WILL GIVE CS: OVERIDE PREFIX
                                   ;  TO ACCESS PROG_DMA

          etc....

PROG_DMA: DW      ______________   ; SOURCE POINTER LS 16 BITS
          DW      ______________   ; SOURCE POINTER MS 4 BITS
          DW      ______________   ; DESTINATION POINTER LS 16 BITS
          DW      ______________   ; DESTINATION POINTER MS 4 BITS
          DW      ______________   ; TRANSFER COUNT
          DW      ______________   ; CONTROL WORD
```

THE INTERRUPT CONTROL BLOCK

The requirements for the interrupts are as follows ...

TIMER 0	no interrupt (mask it out)
TIMER 1	interrupt, priority level 3 (real time clock)
TIMER 2	no interrupt
DMA 0	interrupt, priority level 4 (floppy disk data)
DMA 1	not used (mask it out)
INT 0	interrupt, priority level 2, level triggered (UART ready),
INT 1	interrupt, priority level 5, level triggered (keyboard interrupt)
INT 2	unused (mask it out)
INT 3	unused

...(low number = high priority)

Note that when it comes to the timer interrupts, the individual timers are programmed to produce an interrupt or not. In the interrupt control block you will see that all three timers would produce the same interrupt. In this case the interrupt has to be from TIMER 1, but generally you will have to read the interrupt status register to see which one interrupted. Since we are going to mask out interrupts from TIMER 0, TIMER 2, and INT2/3 we don't need to set up their control registers.Use the priority mask register to block out interrupt levels 5,6 and 7. You might find page 30 of the data booklet helpful here.

```
MOV     TABLE+28H,____________________ ; SET MASK REGISTER
______________________________________ ; PRIORITY MASK REGISTER
______________________________________ ; TIMER CONTROL REGISTER
______________________________________ ; DMA 0 CONTROL REGISTER
______________________________________ ; INT 0 CONTROL REGISTER
______________________________________ : INT 1 CONTROL REGISTER
```

IN THE EVENT OF AN INTERRUPT ....

There you are, at peace with the world when suddenly you get an interrupt to say that the floppy disk controller (via DMA channel 0) has just finished passing it's 128 byte block of data to you. You have written the service routine to handle this event, but before you return from the routine you must remember to tell the interrupt controller that you have finished. It needs to know this in case a lower priority interrupt is pending, waiting for you to finish. Write the code to tell the interrupt controller, via the EOI register that you have finished servicing this interrupt.

```
_______________________________ ; SEND END OF INTERRUPT
```

APPENDIX D

DAILY QUIZZES

Quiz #1

1. The following is a list of implicit uses of the iAPX 86,88 general register set. Supply the register name for each:

   Word multiply, word divide, word I/O

   BYTE multiply, BYTE divide, BYTE I/O

   Translate

   Word multiply, word divide, indirect I/O

   Loops

   Variable shift and rotate

2. Which four general purpose registers can be used in an address expression?

3. What does the assembler use to associate a particular segment register wi a particular segment?

4. What determines the type (near or far) of a RET instruction contained within a procedure?

5. For every variable definition, the assembler tracks what three attributes?

6. Fill in the blank fields in the following chart:

TYPE OF MEMORY REFERENCE	DEFAULT SEG REG	ALT SEGREG	OFFSET SUPPLIED BY
OP CODE FETCH	CS		IP
STACK OPERATION	SS	NONE	
STRING SOURCE		CS,ES,SS	SI
STRING DEST	ES		DI
GENERAL DATA ACCESS	DS		EFFECTIVE ADDRESS
BP USED AS BASE		CS,ES,DS	EFFECTIVE ADDRESS

1. An assembly language procedure is required which will be linked to a PL/M program. The declaration of the procedure in PL/M and a calling sequence are as follows ...

```
ASSEMBLER_CODE: PROCEDURE(ARRAY_PTR,COUNT) EXTERNAL;
                    DECLARE ARRAY_PTR POINTER,
                            COUNT BYTE;
END;

CALL     ASSEMBLER_CODE(@TABLE,1);
```

Define a structure in assembly language which will describe the stack frame which your assembly language program will use. The large model of compilation has been used for the PL/M program.

2. List the abbreviation for each of the following LINK86 controls:

MAP

SYMBOLS

BIND

PRINT

NO LINES

3. Circle the general purpose registers which you must preserve when linking an assembly language procedure to a PL/M program.

AX BX CX DX SI DI BP SP

4. Given the following data segment:

```
DATA SEGMENT

   DIRECTORY STRUC

        LAST NAME    DB    10 DUP (?)
        FIRST NAME   DB       ?
        DEPT         DW       ?
        XTENSION     DB    4 DUP (?)

   DIRECTORY ENDS

   PHONE   DIRECTORY     1000 DUP (< >)

DATA ENDS
```

Evaluate the following expressions*

a. TYPE DEPT

b. TYPE PHONE

c. SIZE LAST NAME

d. TYPE DIRECTORY

e. LENGTH PHONE

f. .DEPT

g. SIZE PHONE

Quiz #3

1. Supply the ASM86 variable definition required for each of the following 8087 data types:

   LONG REAL

   PACKED DECIMAL

   WORD INTEGER

   SHORT INTEGER

   TEMPORARY REAL

TRUE OR FALSE:

2. The 8087 stores all variables internally in the temporary real format. T

3. The 8087 always fetches and stores its operands as bytes so that it will be compatible with the 8088. T F

4. How does the execution of these two instructions differ?

```
    FADD                          FADDP   ST(1),ST
```

Quiz #4

1. At what address is the 80186 peripheral control block following a reset ?

2. Are the following instructions valid on a 186 ...

```
MUL     AX,6
ROR     FRED,13
PUSHI   11
POP     6
IMUL    AX,BX,5
```

3. On reset, which memory bank will be selected by the 186 chip select lines How large is the memory partition assumed to be ?

4. Once the DMA channels have been programmed to start, the first DMA cycle will start (choose one) ...

    1) immediately
    2) next time a DMA request occurs
    3) one instruction after the start command was sent to the DMA chann

5. What are the principle uses of timer 2 ?

    1)
    2)
    3)

APPENDIX E

CLASS EXERCISE SOLUTIONS

CLASS EXERCISE SOLUTIONS

EXERCISE 2.1

1. YES (Memory with Reg, Immed to Memory)
2. YES, by +EA (see page 2 ASM86 Macro Assembler Pocket Ref.)
3. Because register contents and numbers may have to be added together at run time.
4. 16 + EA = 25 clocks

EXERCISE 3.1

1. To tell the assembler that a CS:override prefix is required
2. DIRECT NEAR - destination is a near label
   DIRECT FAR - destination is a far label
   INDIRECT NEAR - destination is a word register or a word variable
   INDIRECT FAR - destination is a double word variable
3.

```
ADD   SP,4  ;waste return base and offset
POP   AX    ;flags into AX
OR    AX,1ØH;set trap flag bit
PUSH  AX    ;print flags image on stack
PUSH  ES    ;print return base on stack
PUSH  DI    ;print return offset on stack
IRET        ;return to new address, setting trap flag
```

EXERCISE 4.1 NEAR AND FAR PROCEDURES

TRUE OR FALSE?

Solutions	
	* Giving a procedure the FAR attribute does the following things...
TRUE	1. encodes a far RET instruction
TRUE	2. tags the procedure as far
FALSE (5 bytes)	3. because of 2, all calls to this procedure will take 3 bytes
FALSE	* Calling a FAR procedure from the segment in which it was defined produces a near call
FALSE	* If in ignorance I near call a procedure which is defined in another module as far the RET instruction prints an error message ... 'HELP - I can't find a segment to return to !'

EXERCISE 5.1

```
%*DEFINE (STRING_MOVE (SOURCE, DEST, COUNT)) (
      MOV   CX, % COUNT
      LEA   SI, % SOURCE
      LEA   DI, % DEST
      PUSH  DS
      POP   ES
  REP MOVSB)
```

EXERCISE 6.1

```
          EMPLOYEE        STRUC
          LAST_NAME       DB  IØ  DUP (?)
          FIRST_NAME      DB  IØ  DUP (?)
          MI              DB  ?
          DIVISION        DW  ?
          DEPT            DW  ?
          EMPLOYEE        ENDS

          WORKFORCE EMPLOYEE IØØ DUP  (< >)

          MOV      CX,LENGTH  WORKFORCE
          LEA      BX,WORKFORCE.DIVISION
    NEXT: MOV      WORD PTR  [BX] , 12
          ADD      BX,TYPE WORKFORCE
          LOOP     NEXT
```

EXERCISE 6.2

1. BITE RECORD B23:2, RUBBIS:2
2. AND AL, MASK B23
3. TYPE BITE IS 1 (it takes 1 byte to store it)

EXERCISE 8.1

```
RUN LINK86  PROG.OBJ, PROCS.OBJ, SMALL.LIB

RUN LOC86   PROG.LNK &
            ORDER (CLASSES(DATA, CONST,STACK)) &
            ADDRESSES (CLASSES(DATA(2ØØH),CODE(FØØØØH))
                       SEGMENTS (NVM(CØØØH))) &
                       INITCODE (FØØØØH) &
                       BOOTSTRAP
```

EXERCISE 9.1

```
CGROUP   GROUP   CODE1
CODE1   SEGMENT
        ASSUME  CS:CODE1
CMP_STRING   PROC
             PUSH    BP
             MOV     BP,SP
             MOV     CX, [BP]  + 4; STRING COUNT
             MOV     DI, [BP]  + 6; STRING 2 POINTER
             MOV     SI  [BP]  + 8; STRING 1 POINTER
             PUSH    DS
             POP     ES            ; BASE FOR STRING 2
       REPE  CMPSB
             MOV     AL,Ø          ; ASSUME MISMATCH
             JNE     EXIT
             MOV     AL,ØFFH       ; STRINGS MATCH
      EXIT:  POP     BP
             RET     6
CMP_STRING   ENDP
CODE1        ENDS
```

EXERCISE 9.2

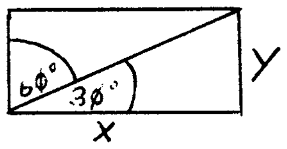
```
ARRAY_SUM_SEG SEGMENT 'CODE'
ARRAY_SUM  PROC    FAR
           PUSH    BP
           MOV     BP,SP
           LES DI, [BP]  + 8      ; 32 BIT POINTER TO ARRAY
           MOV CX, [BP]  + 6      ; LENGTH OF ARRAY
           MOV AX,Ø               ; CLEAR SUM
   AGAIN:  ADD AX,ES: [DI]        ; ADD ARRAY ELEMENT TO SUM
           INC SI                 ; UPDATE ARRAY POINTER
           LOOP AGAIN             ; REPEAT CX TIMES
           POP  BP
           RET  6
ARRAY_SUM  ENDP
ARRAY_SUM_SEG   ENDS
```

EXERCISE 12.1

```
DATA SEGMENT
A  DD   1.234
B  DD   234
C  DD   1ØØØ.
D  DD   9.82
RESULT DQ  ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
          :
        FLD     A
        FADD    B
        FDIV    C
        FMUL    D
        FSTP    RESULT
```

EXERCISE 12.2

```
        EXTRN   INIT87:FAR

DATA_1  SEGMENT

COS_THETA       DQ      ?

SIN_THETA       DQ      ?

TAN_THETA       DQ      ?


DATA_1  ENDS

CODE_1  SEGMENT
        ASSUME  CS:CODE_1,DS:DATA_1

SIX     DQ              6.0 ; MUST BE SHORT OR LONG REAL (NOT TEMP)
```



```
;
;
;
;
; REMEMBER THAT OPPOSITE AND ADJACENT SIDES OF THE RIGHT TRIANGLE INCLUDING
; THE 30 DEGREE ANGLE ARE SWOPPED FROM THE MIRROR IMAGE TRIANGLE CONTAINING
; THE ORIGINAL 60 DEGREE ANGLE. HENCE TAN = X/Y, SIN = X/HYPOT, COS = Y/HYPOT
;
TRIG    PROC    FAR

        CALL    INIT87
                                ; ST(0) ST(1)   ST(2)   ST(3)   ST(4)   ST(5)
        FLDPI                   ; PI    -       -       -       -       -
        FDIV    SIX             ; PI/6  -       -       -       -       -
        FPTAN                   ; X     Y       -       -       -       -
        FLD     ST(1)           ; Y     X       Y       -       -       -
        FLD     ST(1)           ; X     Y       X       Y       -       -
        FDIV    ST,ST(1)        ; X/Y   Y       X       Y       -       -
        FSTP    TAN_THETA       ; Y     X       Y       -       -       -
        FLD     ST(1)           ; X     Y       X       Y       -       -
        FMUL    ST,ST(0)        ; X^2   Y       X       Y       -       -
        FXCH                    ; Y     X^2     X       Y       -       -
        FMUL    ST,ST(0)        ; Y^2   X^2     X       Y       -       -
        FADD                    ; X^2+Y^2 X     Y       -       -       -
; ** FADD IN CLASSICAL STACK MODE DOES INCLUDE A POP !!! **
        FSQRT                   ; HYPOT X       Y       -       -       -
        FDIV    ST(1),ST        ; HYPOT SIN     Y       -       -       -
        FDIVP   ST(2),ST        ; SIN   COS     -       -       -       -
        FSTP    SIN_THETA       ; COS   -       -       -       -       -
        FSTP    COS_THETA       ; -     -       -       -       -       -
        RET
TRIG    ENDP

CODE_1  ENDS
```

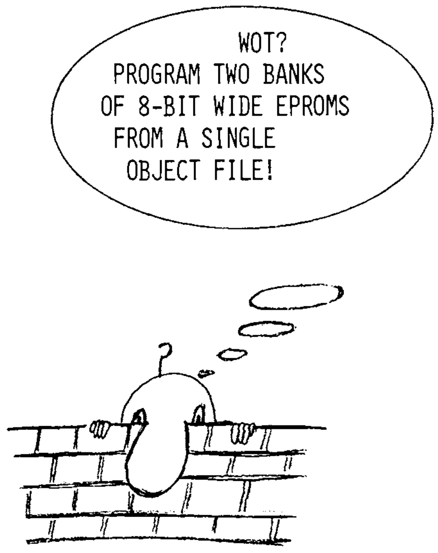
EXFRCISE 15.1

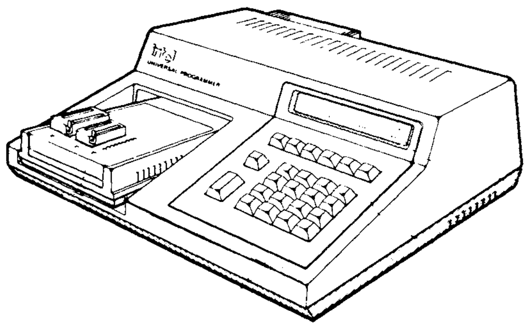
```
        IN      AL, ØD8H
        CBW
        IMUL    AX, -5
        MOV     DX, ØFFFAH
        OUT     DX, AX
```

APPENDIX F

INTRODUCTION TO PROM PROGRAMMING

PROM PROGRAMMING

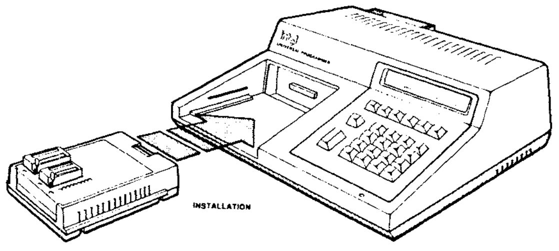




SHOWN: ıUP-2Ø1 PROM PROGRAMMER

ıUP-200/201 INTEL UNIVERSAL PROGRAMMER

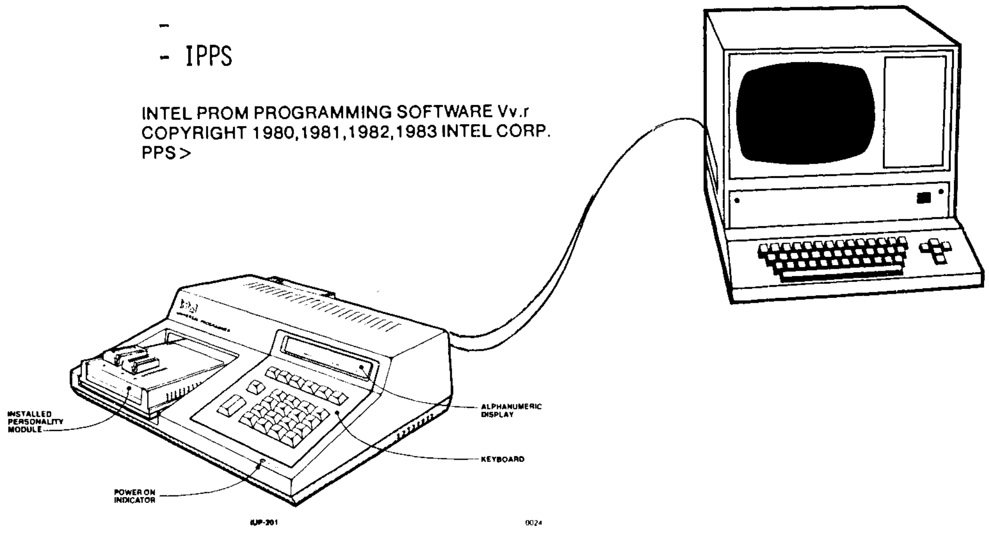
- PROGRAMS A VARIETY OF PROMS/EPROMS USING VARIOUS PERSONALITY MODULES
- READS ROMS/PROMS/EPROMS
- READS/WRITES DISK FILES ON HOST MDS
- ALLOWS EDITING OF OBJECT CODE
- FORMATS OBJECT CODE TO SUIT PROM CONFIGURATION (EG. 2 BANKS 8 BIT WIDE = 16 BIT WIDE MEMORY)



DRIVING PROGRAMMER FROM MDS

- IPPS

INTEL PROM PROGRAMMING SOFTWARE Vv.r
COPYRIGHT 1980,1981,1982,1983 INTEL CORP.
PPS>



PROGRAMMER OPERATION

- SIMPLE COMMAND LANGUAGE EG,

  PPS >COPY :F1:LOWER.BYT TO PROM

- DO IT ALL FROM A SUBMIT FILE
- IF YOU CAN'T REMEMBER ... PPS >HELP
- IF TOTALLY LOST ... PPS >HELP HELP

FOR DETAILS . . .

iUP - 200/201 UNIVERSAL PROGRAMMER USER'S GUIDE

EXAMPLE : FORMATTING A FILE

- TAKE INPUT FILE NIBBLE.OLD AND USE FIRST 4K BYTES TO PRODUCE TWO OUTPUT FILES FOR BLOWING ODD AND EVEN BANK PROMS

  BOLD TYPE IS OPERATOR ENTRY

```
PPS>FORMAT NIBBLE.OLD (0,FFFH)
LOGICAL UNIT (BIT=1,NIBBLE=2,BYTE=3,N-BYTE=4)
LU = 3
INPUT BLOCK SIZE (N BYTES)
N = 2
OUTPUT BLOCK SIZE (N BYTES)
N = 1
INPUT BLOCK STRUCTURE:
NUMBER OF INPUT LOGICAL UNITS = 002

LSB
 ----------
 |00|01|
 ----------

NUMBER OF OUTPUT LOGICAL UNITS = 001
OUTPUT SPECIFICATION (CR TO EXIT):
*0 TO :F1:LOWER.BYT
OUTPUT STORED
*1 TO :F1:UPPER.BYT
OUTPUT STORED
*<CR>
PPS>
```

INTEL WORKSHOPS

Microcomputer Workshops—Architecture & Assembly Language

- Introduction to Microprocessors
- MCS®-48/49 Microcontrollers
- MCS®-51 Microcontrollers
- MCS®-96 16-Bit Microcontrollers
- MCS®-80/85 Microprocessors
- iAPX 86, 88, 186 Microprocessors, Part I
- iAPX 86, 88, 186 Microprocessors, Part II
- iAPX 286 Microprocessors
- Data Communications including Ethernet
- Speech Communication with Computers
- iCEL™ VLSI Design

Programming and Operating Systems Workshops

- Beginning Programming Using Pascal
- PL/M Programming
- PL/M-iRMX™ 51 Operating System
- iRMX™ 86 Operating System
- XENIX*/C Programming
- System 86/300 Applications Programming
- iDIS™ Database Information System
- iTPS Transaction Processing System
- Development System Seminars

System 2000® Database Management Workshops

- System 2000® For Non-Programmers
- System 2000® Technical Fundamentals
- System 2000® Applications Programming
- System 2000® Report Writing
- System 2000® Database Design and Implementation

Self-Study Introduction to Microprocessors
System 2000® Multimedia Course

BOSTON AREA
27 Industrial Avenue, Chelmsford, MA 01824 (617) 256-1374

CHICAGO AREA
Gould Center, East Tower
2550 Golf Road, Suite 815, Rolling Meadows, IL 60008 (312) 981-7250

DALLAS AREA
12300 Ford Road, Suite 380, Dallas, TX 75234 (214) 484-8051

SAN FRANCISCO AREA
1350 Shorebird Way, Mt. View, CA 94043 (415) 940-7800

WASHINGTON D.C. AREA
7833 Walker Drive, 5th Fl., Greenbelt, MD 20770 (301) 474-2878

LOS ANGELES AREA
Kilroy Airport Center, 2250 Imperial Highway, El Segundo, CA 90245 (415) 940-7800

CANADA
190 Attwell Drive, Toronto, Ontario M9W 6H8 (416) 675-2105

intel®
customer
training