AMD CUSTOMER EDUCATION CEC $15.00

# EDSYS29

# INTRODUCTION TO THE AMC SYS/29 DEVELOPMENT SYSTEM

## LABS & EXERCISES

Publication number: CEC.PUB-29-10
2nd Edition
February 10, 1981

901 Thompson Place, P.O. Box 453, Sunnyvale, California 94086
(408)732-2400 TWX: 910-339-9280 TELEX: 34-6306

# EDSYS29

# STUDY GUIDE AND LAB BOOK

## INCLUDING EXAMPLE PROGRAM LISTINGS

for the Advanced Micro Computers

AmSYS29

by

Donnamaie E. White

Advanced Micro Devices, Inc.

Customer Education Center

2nd Edition

February, 1981

## PREFACE

This lab book was designed to accompany the Advanced Micro Devices, Inc. seminar/workshop on the Advanced Micro Computers AmSYS29 Development System. The seminar concentrates on the AmSYS29 as a development tool for 2900 Family designs and as such emphasizes the microprogram development tools. It does not cover the use of the system for 8080, 8085, Z80, or Z8000 software/hardware development.

The seminar was designed to follow the ED2900A "Introduction to Design with the 2900 Family" seminar, which introduces the 2900 family and also introduces the concept of structured microprogramming, including the use of mnemonics. This includes the concepts behind the .DEF and .SRC AMDASM files.

For those who have not taken this prerequisite, the initial labs may be more difficult. For this reason, the problems from ED2900A are included in the lab book for study and review. The complete solution to the coffee machine problem is included. The instructor facing a class, where most or all of the students have not taken ED2900A (or the new ED2910), should very carefully go over the problem, the manual solution,and then refer to the .DEF and .SRC files as examples during the lecture.

The instructor and student should be sure to have the correct data disk to go with the labs, since the labs will refer to specific files. If such a disk is not available or cannot be obtained in time, the disk can be created from the sample programs that are included throughout the lab book.

This seminar takes the student up to interfacing to a prototype, but does not include actual driving of a prototype from the writeable control store. The ED2900B "Microprogrammable Computer Architecture" or "Advanced Design with the 2900 Family" seminar does include labs wherein the AmSYS29 is interconnected to a HEX-29 minicomputer. EDSYS29 is a prerequisite for ED2900B. ED2900C is the non-lab (non-HEX-29) version of ED2900B.

None of the labs include interconnection to a PROM-burner, but the Data-I/O application note on such interconnection is included.

Two reference .DEF files are included, which should be mentioned. AM2900.LIB is an older file which was created to provide users with a file from which they could patch-up a .DEF file with a minimum of effort. It contains both older and newer devices, and is not as easy to use as it might be. Included also is the AM2903.DEF file, which includes a new .DEF file for the Am2903, the Am2910, and a partial for the Am2904. (The Am2904 uses bit-steering and it would be prohibitive to list all possibilities.) This file is referenced in ED2900A and ED2900B/C. It is considerably easier to use, especially for those who are new to the 2900 family or the AmSYS29 system. By altering the definition statements and the word width, and adding those equates and other statements as needed, this file can be used by those who are developing code for an Am2903-based design.

Donnamaie E. White, Ph.D.

## INDEX

### DAY 1

************************************************************

## STUDY GUIDE INDEX

### DAY 2

# LAB ONE:

# GETTING ACQUAINTED WITH

# THE AMC SYS/29

## – FILE HANDLING

## LAB ONE - INTRODUCTION TO FILE HANDLING

### RULES OF THE GAME

1. BREAK UP INTO TEAMS - USE YOUR HOTELS, HOME ADDRESS, OR OTHER DATA TO CHOOSE PARTNERS YOU WILL MAINTAIN THROUGHOUT THE CLASS

2. THOSE OF YOU WITH 2900 FAMILY DESIGN EXPERIENCE SHOULD BE SPREAD OUT AMONG THE OTHERS

3. TRY TO DISTRIBUTE HARDWARE AND SOFTWARE EXPERTISE AMONG THE TEAMS

4. HOMEWORK AND LAB SESSIONS ARE TO BE A GROUP EFFORT - YOU WILL LEARN MORE THAT WAY!

PURPOSE OF THE LAB: FAMILIARIZATION WITH BASIC FILE HANDLING FUNCTIONS OF AMC SYSTEM 29.

## LAB STATIONS

EACH LAB STATION CONSISTS OF:

- ONE AmSYS 29 CPU BOX - FRONT PANEL SWITCHES
  NOTE: THERE IS ANOTHER MODEL WHICH
  LOOKS SLIGHTLY DIFFERENT - WORKS THE SAME
- ONE LINE PRINTER
- ONE CRT - EITHER ADDS OR THE OLDER TELERAY
- ONE DUAL 8" FLOPPY DISK DRIVE SYSTEM

CHECK TO SEE THAT YOUR TEAM HAS THE FOLLOWING:

- A SYSTEM DISK LABELED VER 1.4
- A DATA DISK (WHICH IS NOT BLANK)

## DO NOT MISHANDLE THE DISKETTES!

- DO NOT FOLD, SPINDLE OR MUTILATE
- NO COFFEE, SUGAR OR CREAM
- NO SODA, MAYONNAISE, ETC.
- DO NOT WRITE WITH A PENCIL OR A BALL POINT PEN

CHECK TO SEE THAT THERE IS PAPER FEEDING INTO THE PRINTER

- FEEDING STRAIGHT IN
- ON THE TRACTOR PINS
- PRINT IS CLEAR (MAY NOT BE SHARP BLACK BUT SHOULD BE READABLE)
- DO NOT STEP ON, PLACE CHAIRS ON OR OTHERWISE BLOCK PAPER FEED

## NORMAL SYSTEM POWER-UP

IF YOU OWN AN AmSYS29 THERE IS A CONVENTIONAL METHOD OF POWER-ON AND SIGN-ON

WE WILL BYPASS THIS

## LAB POWER-ON PROCEDURE

- MAKE SURE THAT ALL UNITS ARE PROPERLY PLUGGED IN
- MAKE CERTAIN THAT THE CRT IS CONNECTED TO THE BACK OF THE SYSTEM 29 CPU
- MAKE CERTAIN THAT THE PRINTER IS CONNECTED
- MAKE CERTAIN THAT THE FLOPPY DISK SYSTEM IS CONNECTED
- TURN ON ALL SWITCHES

    2 SWITCHES ON FRONT OF SYSTEM 29 CPU
    CHECK ALL OTHER SWITCHES ON THE CPU

    1 DIAL SWITCH ON THE CRT (TELERAY)
    OR SWITCH ON RIGHT BOTTOM OF ADDS TERMINAL

    1 SWITCH ON THE FRONT OF THE DISK DRIVE

    1 ON THE LOWER LEFT REAR OF THE PRINTER

## DISKETTE INSERTION

- PLACE THE SYSTEM VER 1.4 DISKETTE IN THE RIGHT SIDE OF THE DISK UNIT, THIS IS DRIVE "A" (NEWER SYSTEMS STACK THE DRIVES, THE TOP UNIT IS DRIVE A)

  BE SURE THAT THE DISKETTE IS HELD BY THE LABEL EDGE

  BE SURE THE LABEL EDGE EXTENDS TO THE FRONT

  BE SURE THAT THE LABEL IS FACING UP WHEN PLACING THE DISKETTE INTO THE DRIVE

  INSERT FIRMLY BUT DO NOT BEND

  CLOSE DOOR - BUT NOT ON DISKETTE!

- PRESS "RESET" SWITCH ON THE FRONT OF THE AmSYS/29 CPU

YOU SHOULD HEAR SOME ACTIVITY FROM THE DISK UNIT AND SEE SOME MESSAGE ON THE SCREEN

  IF YOU DO, YOU'RE OK

  IF YOU DO NOT, YOU'RE NOT BOOTING UP

## SIGN ON

* ONCE BOOTED, TYPE THE "CTRL" KEY AND THE Z KEY AT THE SAME TIME (Zc) THEN PRESS THE "RETURN" KEY (RET) WHICH IS LABELED AS "NEW LINE" ON THE ADDS TERMINAL

* THE SYSTEM WILL RESPOND WITH

```
A>_
```

- A> REFERS TO THE ACTIVE OR SIGNED-ON DRIVE
- THE _ MARKS THE CURSOR POSITION
- THE > IS CONSIDERED TO BE THE SYSTEM PROMPT - DIFFERENT SYSTEMS USE DIFFERENT SYMBOLS AS PROMPTS (YOU WILL BE SEEING . > *)

* INSERT YOUR DATA DISKETTE IN DRIVE "B" (LEFT HAND SIDE)

ANY TIME THAT A DISKETTE IS CHANGED YOU MUST LET THE SYSTEM KNOW BY TYPING Cc AND "RETURN" (Ret) (THIS IS A "WARM START")

TYPE: Cc AND Ret AND THEN LISTEN (YOU SHOULD HEAR SOME ACTIVITY FROM THE DRIVE)

## COMMAND SUMMARY

| | |
|---|---|
| DIR | DIRECTORY, LISTING OF FILES |
| Pc | ENABLE/DISABLE PRINTER |
| STAT | DISKETTE STATUS |
| PIP | PERIPHERAL INTERCHANGE PROGRAM |
| | (SAY IT THREE TIMES QUICKLY!) |
| ERA | ERASE FILES |
| TYPE | PRINT A FILE TO SCREEN |
| | (EVELYN WOOD SPEED READING!) |
| | USE WHEN PRINTER IS ENABLED TO OBTAIN A LISTING |
| Cc | WARM START |
| DISPL | DISPLAY A FILE PAGE AT A TIME |
| | USE WHEN PRINTER DISABLED, FOR HUMAN-CRT DISPLAY |

## LAB EXERCISES

- TYPE THE FOLLOWING:

```
DIR(Ret)         (I WILL USE (Ret) TO MEAN RETURN KEY)
```

- THIS IS A REQUEST TO THE SYSTEM TO LIST THE DIRECTORY OF THE DISKETTE CURRENTLY ON THE SIGN-ON DRIVE (DRIVE "A")

```
DIRPc
```

- THE Pc TURNS ON THE PRINTER AND THE DIRECTORY WILL NOW APPEAR BOTH ON THE CRT SCREEN AND ON THE PRINTER

```
Pc
```

- THE PRINTER SHOULD BECOME QUIET

```
B:(Ret)
```

- THIS LOGS ON DRIVE "B" AND THE SCREEN SHOULD BE DISPLAYING

```
        B>_
```

```
DIR(Ret)
```

- THIS TIME NOTE THE B> WHICH PRECEEDS THE FILES LISTED

  DO YOU SEE COFFEE.DEF, COFFEE.SRC ?
  YOU SHOULD ALSO SEE FILES WITH THE
  PRIMARY NAMES OF:
  - MONITOR
  - SIMPLE
  - AM2903

```
DIR A:(Ret)
```

- WHILE DEFAULT IS TO THE LOGGED ON DRIVE, IT CAN BE OVERRIDDEN BY SPECIFICATION OF THE DESIRED DRIVE

```
A:(Ret)
```

- TYPE THE FOLLOWING IN ORDER (ASSUME (Ret) AT THE END OF EACH)

```
STAT
```

STATUS REQUEST

```
STAT *.*
```

```
STAT B:*.*
```

- CONTINUE TYPING

```
TYPE B:SIMPLE.DEF         <----   TYPE THE WORD "TYPE"
        USED WITH Pc WILL ALLOW PRINT OUT OF A FILE
```

```
DISPL B:SIMPLE.DEF
          THIS IS A PAGED FILE DISPLAY
          HIT (Ret) UNTIL "*" APPEARS
          TWICE AT THE BOTTOM OF THE SCREEN
```

```
E(Ret)
          E STANDS FOR EXIT
```

```
DIR B:SIMPLE.*
        THESE CHECK TO FIND A FILE IN A DIRECTORY
```

- <u>TURN PRINTER ON</u>: Pc

```
TYPE B:COFFEE.DEF

TYPE B:COFFEE.SRC

Pc
```

CONTINUE:

```
AMDASM P1 B:COFFEE P2 B:COFFEE
```

DO NOT PANIC! THE DISKS ARE NOISY!

```
Pc
TYPE B:COFFEE.P2L
Pc
```

```
PIP B:SIMPLE.TST=B:SIMPLE.DEF[V]
```

PERIPHERAL INTERCHANGE PROGRAM "COPY"
PIP IS ALWAYS "TO FILE" = "FROM FILE"
NOTE: YOU SHOULD CLOSE THE BRACKET
(IT WILL ACCEPT (Ret) AS A TERMINATOR)

```
DIR B:SIMPLE.*
```

```
PIP B:TESTONE=A:SIMPLE.DEF[V]
```

```
DIR B:TESTONE
```

```
TYPE B:TESTONE
```

TYPE Sc IMMEDIATELY TO HALT SCREEN
TYPE (Ret) OR Sc TO CONTINUE

```
ERA B:TESTONE
```

```
PIP B:=A:AM2903.DEF[V]
```

- TYPE THE FOLLOWING AND FOLLOW BY (Ret)

```
PIP
LST:=AM2903.DEF    }  or  {  PIP LST:=AM2903.DEF
(Ret) or Cc           or  {  PIP PRN:=AM2903.DEF
```

```
PIP B:LIGHT.DEF=A:AM2903.DEF[VSJMAPZcQTWBZcT8]
```

- MOVE IS INCLUSIVE

```
TYPE B:LIGHT.DEFPc

Pc
```

END OF EXERCISE

* REMOVE DISKETTES

* POWER DOWN

# EXERCISES:

## FILENAMES

## AMDASM

## EXERCISES

ARE THESE PROPER FILE NAMES OR FILE NAME REFERENCES?

_____DOOR.DEF

_____DOOR.*

_____B:*.DEF

_____D?OR.D?F

_____123.456

_____TEMP

_____DOOR.SRC

_____B:DOOR.SRC

_____*.*

_____B:X?X.*

_____1-2-3

_____B:TEMP

WHAT EXTENSION NAME IS REQUIRED FOR THE DEFINITION FILE BEFORE IT CAN BE ASSEMBLED VIA AMDASM?

WHAT EXTENSION NAME IS REQUIRED FOR THE SOURCE FILE BEFORE IT CAN BE ASSEMBLED BY AMDASM?

IS THE EXTENSION REQUIRED WHEN CALLING FOR AN ASSEMBLY?

## EXERCISES

WHAT SYMBOL STARTS A COMMENT?

WHAT SYMBOL STARTS A LINE THAT IS A CONTINUATION OF THE PRECEEDING LINE?

WHAT IS THE DESIGNATOR FOR A HEX CONSTANT?

HOW MANY CHARACTERS CAN BE IN A VARIABLE NAME?

WHAT CHARACTERS MAY BE THE FIRST CHARACTER IN A VARIABLE NAME?

WHAT DETERMINES IF % IS A MODIFIER OR AN ATTRIBUTE?

WHAT IS THE ATTRIBUTE $ EQUIVALENT TO?

IF NO BASE IS GIVEN IN AN EQU STATEMENT, WHAT IS THE DEFAULT?

IF NO BASE IS GIVEN IN A DEF STATEMENT, WHAT IS THE DEFAULT?

IF NO BASE IS GIVEN IN AN ASSEMBLY STATEMENT VARIABLE FIELD SUBSTITUTION, WHAT IS THE DEFAULT?

CAN EQU STATEMENTS APPEAR IN THE DEF AND SRC FILES?

CAN DEF STATEMENTS APPEAR IN THE DEF AND SRC FILES?

WHAT IS THE STATEMENT WORD USED FOR?

HOW WIDE CAN A VARIABLE FIELD BE (NUMBER OF BITS)?

HOW WIDE CAN A DON'T CARE FIELD BE?

WHAT IS THE MAXIMUM NUMBER OF FIELDS ALLOWED IN A DEF STATEMENT?

# THE COFFEE MACHINE REVISITED

## DEF & SRC FILES

2900 FAMILY STUDY GUIDE
SIMPLE PROBLEMS FOR BEGINNERS
THE FAMOUS COFFEE MACHINE
EDSYS29 EXAMPLE

You are to design a coffee machine controller that will handle a simple, non-fault diagnostic coffee dispenser. It will work as follows:

1. Do nothing until a coin is detected.
2. On coin detection, turn on the busy light and drop a cup.
3. The cup has 1.5 seconds to get into place.
4. There is no way to know if the cup is correctly positioned or if it even is there.
5. Water is turned on for 1.0 second prior to the release of powders (so it isn't unsightly in the bottom of the cup).
6. Water will remain on continuously for a total of 10 seconds.
7. The busy light will remain on until the sequence is completed.
8. Depending upon the selection, either coffee, soup, or chocolate will be dispensed.

| | |
|---|---|
| coffee | 2.5 seconds |
| soup | 2.0 seconds |
| chocolate | 3.5 seconds |

9. If coffee was selected, when the coffee is finished, sugar or cream is dispensed.

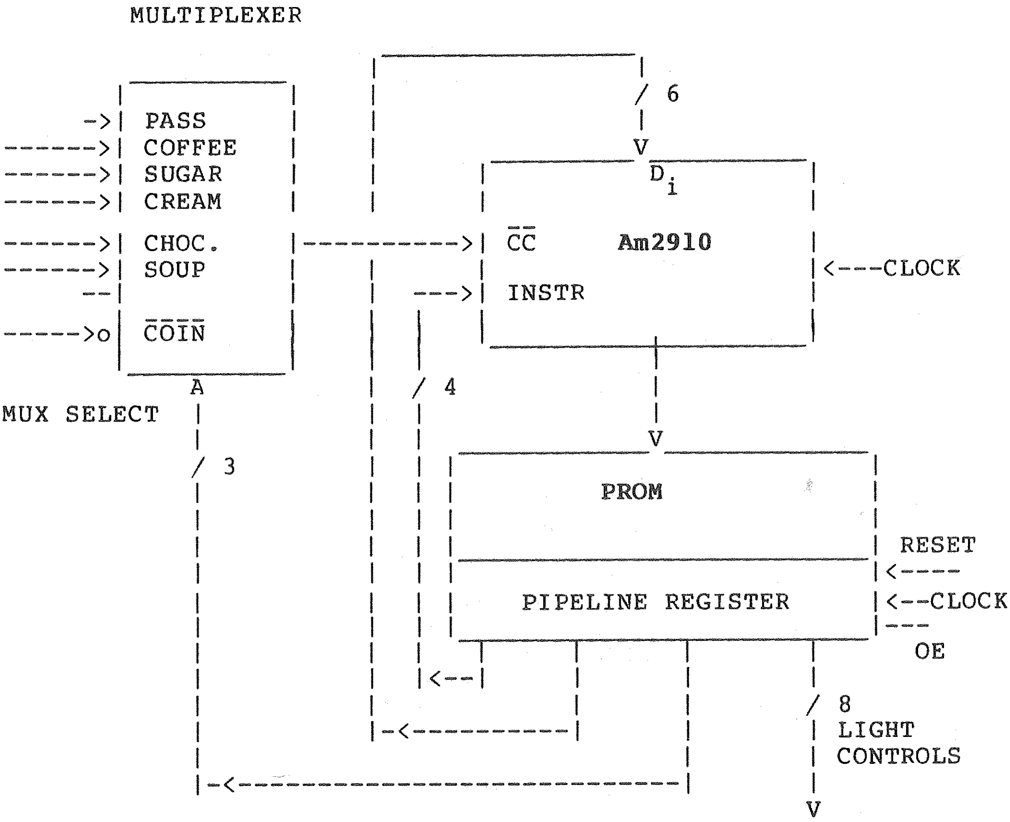
| | |
|---|---|
| sugar | 1.5 seconds |
| cream | 2.0 seconds |

10. If sugar and cream was the selection, when the sugar is finished, start cream.
11. After the water has completed filling the cup, allow 3.5 seconds for cup removal before testing for the presence of the next coin.
12. You have a 0.5 second clock pulse.

There are six possible sequences:

COFFEE, BLACK
COFFEE, CREAM
COFFEE, SUGAR
COFFEE, CREAM AND SUGAR
CHOCOLATE
SOUP

## Am2910 HARDWARE SOLUTION



A = COFFEE     B = CREAM     C = SUGAR

MUX INPUT:
COFFEE-ON <=> A + AB + AC + ABC
CREAM-ON <=> AB + ABC
SUGAR-ON <=> AC + ABC

## THE MICROPROGRAM

| LABEL | ADDR | Am2910 INSTR. | COND. MUX | BRANCH COUNTER | MACHINE CONTROL |
|---|---|---|---|---|---|
| ZERO | 0 | CJP | $\overline{\text{COIN}}$ | ZERO | OFF |
| | 1 | CONT | # | # | CUP-BUSY |
| | 2 | CONT | # | # | BUSY |
| | 3 | CONT | # | # | BUSY |
| | 4 | CJP | CHOC | CHOC | WATER-BUSY |
| | 5 | CJP | SOUP | SOUP | WATER-BUSY |
| COFFEE | 6 | LDCT | # | 12 | COFFEE-WTR-BUSY |
| | 7 | CONT | # | # | COFFEE-WTR-BUSY |
| | 8 | CONT | # | # | COFFEE-WTR-BUSY |
| | 9 | CJP | SUGAR | SUGAR | COFFEE-WTR-BUSY |
| | 10 | CJP | CREAM | CRE-2 | COFFEE-WTR-BUSY |
| LOOP | 11 | RPCT | # | LOOP | WATER-BUSY |
| | 12 | LDCT | # | 4 | BUSY |
| BUSY | 13 | RPCT | # | BUSY | BUSY |
| | 14 | JZ | # | ZERO | BUSY |
| SUGAR | 15 | CONT | # | # | COFFEE-WTR-BUSY |
| | 16 | LDCT | # | 9 | SUGAR-WTR-BUSY |
| | 17 | CJP | CREAM | CREAM | SUGAR-WTR-BUSY |
| | 18 | CJP | PASS | LOOP | SUGAR-WTR-BUSY |
| CREAM | 19 | LDCT | # | 5 | SUGAR-WTR-BUSY |
| | 20 | CONT | # | # | CREAM-WTR-BUSY |
| | 21 | CONT | # | # | CREAM-WTR-BUSY |
| ENTRY | 22 | CONT | # | # | CREAM-WTR-BUSY |
| | 23 | CJP | PASS | LOOP | CREAM-WTR-BUSY |
| CRE-2 | 24 | LDCT | # | 8 | CREAM-WTR-BUSY |
| | 25 | CJP | PASS | ENTRY | CREAM-WTR-BUSY |

```
CHOC    26    CONT    #      #       WATER-BUSY
        27    LDCT    #      5       CHOC-WTR-BUSY
CHOCLP  28    RPCT    #      CHOCLP  CHOC-WTR-BUSY
        29    LDCT    #      8       WATER-BUSY
        30    CJP     PASS   LOOP    WATER-BUSY

SOUP    31    CONT    #      #       SOUP-WTR-BUSY
        32    CONT    #      #       SOUP-WTR-BUSY
        33    LDCT    #      13      SOUP-WTR-BUSY
        34    CJP     PASS   LOOP    SOUP-WTR-BUSY

        :                            .

        63    JZ      #      #       OFF
```

# COFFEE MACHINE .DEF and .SRC

THE FAMOUS COFFEE MACHINE

- COFFEE.DEF listing
- COFFEE.SRC listing
- COFFEE.P2L AMDASM ASSEMBLY listing
    - .SRC SEQUENCED
    - CONTROL MEMORY PRINTOUT (X for Don't Cares)
    - SYMBOLS list

## .DEF

```
TITLE COFFEE MACHINE
;
WORD 21
;
;AM2910 INSTRUCTIONS
;
JZ:      EQU      H#0      ;JUMP TO ZERO
CJP:     EQU      H#3      ;;CONDITIONAL JUMP
RPCT:    EQU      H#9      ;DO-LOOP
LDCT:    EQU      H#C      ;LOAD COUNTER AND CONTINUE
CONT:    EQU      H#E      ;CONTINUE
;
;CONDITIONAL MUX SELECT FIELD
;
NOCOIN:  EQU      Q#0      ;TEST FOR COIN
NULL:    EQU      Q#1
SOUPTST:EQU       Q#2
CHOCTST:EQU       Q#3
CREMTST:EQU       Q#4
SUGRTST:EQU       Q#5
CAFETST:EQU       Q#6
PASS:    EQU      Q#7      ;ACTIVE LOW PASS
;
;MACHINE CONTROLS
;
; THE BIT LAYOUT PATTERN IS
; BUSY(LIGHT)-CUP(DROP)-WATER-COFFEE-SUGAR-CREAM-CHOCOLATE-SOUP
;
OFF:     EQU      H#00
BUSYON:  EQU      H#80
CUPDROP:EQU       H#C0
WATERON:EQU       H#A0
COFFEON:EQU       H#B0
SUGARON:EQU       H#A8
CREAMON:EQU       H#A4
CHOCON:  EQU      H#A2
SOUPON:  EQU      H#A1
;
;FORMAT DEFINITION
;
MACHINE: DEF      13X,     8VH#00               ;MACHINE CONTROLS
SEQ:     DEF      4VH#E,   3VQ#1,  6V$X,8X      ;NEXT ADDRESS CTRL
;
END
```

```
TITLE COFFEE MACHINE SRC FILE
;
;LABEL
;-----------------------------------------------------------------------------
ZERO:   SEQ CJP, NOCOIN, ZERO    & MACHINE OFF   ; WAIT FOR COIN
        SEQ                      & MACHINE CUPDROP
        SEQ                      & MACHINE BUSYON
        SEQ                      & MACHINE BUSYON
;
; TURN ON WATER AND TEST TO FIND ROUTINE
;
        SEQ CJP, CHOCTST, CHOC   & MACHINE WATERON
        SEQ CJP, SOUPTST, SOUP   & MACHINE WATERON
;----------------------------------------------------
;
; COFFEE BUTTON HAS BEEN PUSHED - DISPENSE COFFEE
;
COFFEE: SEQ LDCT,       , H#C    & MACHINE COFFEON
        SEQ                      & MACHINE COFFEON
        SEQ                      & MACHINE COFFEON
        SEQ CJP,SUGRTST, SUGAR   & MACHINE COFFEON
        SEQ CJP,CREMTST, CRM2    & MACHINE COFFEON
;----------------------------------------------------
;
; FINISH FILLING THE CUP WITH WATER
;
LOOP:   SEQ RPCT,      , LOOP    & MACHINE WATERON
        SEQ LDCT,      , H#4     & MACHINE BUSYON
;----------------------------------------------------
;
; ALLOW TIME TO REMOVE CUP BEFORE STARTING OVER
;
BUSY:   SEQ RPCT,      , BUSY    & MACHINE BUSYON
        SEQ JZ                   & MACHINE BUSYON
;----------------------------------------------------
;
; DISPENSE SUGAR
;
SUGAR:  SEQ                        & MACHINE COFFEON
        SEQ LDCT,       , H#9      & MACHINE SUGARON
        SEQ CJP,CREMTST, CREAM     & MACHINE SUGARON
        SEQ CJP, PASS   , LOOP     & MACHINE SUGARON
;----------------------------------------------------
;
; DISPENSE CREAM (COFFEE, SUGAR AND CREAM ENTRY)
;
CREAM:  SEQ LDCT,       , H#5      & MACHINE SUGARON
        SEQ                        & MACHINE CREAMON
        SEQ                        & MACHINE CREAMON
ENTRY:  SEQ                        & MACHINE CREAMON
        SEQ CJP, PASS   , LOOP     & MACHINE CREAMON
;----------------------------------------------------
;
; DISPENSE CREAM (COFFEE AND CREAM ENTRY)
;
```

```
CRM2:   SEQ LDCT,        , H#8        & MACHINE CREAMON
        SEQ CJP, PASS    , ENTRY      & MACHINE CREAMON
;------------------------------------------------------
;
; DISPENSE CHOCOLATE
;
CHOC:   SEQ                           & MACHINE WATERON
        SEQ LDCT,        , H#5        & MACHINE CHOCON
CHOCLP: SEQ RPCT,        , CHOCLP     & MACHINE CHOCON
        SEQ LDCT,        , H#8        & MACHINE WATERON
        SEQ CJP, PASS    , LOOP       & MACHINE WATERON
;------------------------------------------------------
;
; DISPENSE SOUP
;
SOUP:   SEQ                           & MACHINE SOUPON
        SEQ                           & MACHINE SOUPON
        SEQ LDCT,        , H#D        & MACHINE SOUPON
        SEQ CJP, PASS    , LOOP       & MACHINE SOUPON
;------------------------------------------------------
;
; SKIP OVER MEMORY WHICH IS UNUSED
;
ORG 63
        SEQ JZ                        & MACHINE
;
END
```

NOTICE THAT

IF THE DEFINITION OF THE FORMAT HAD BEEN:

```
SEQ: DEF 4VH#E, 3VQ#1, 6V$X, 8VH#00
```

THEN THE SRC FILE WOULD HAVE BEEN ALTERED TO BE:

```
COFFEE: SEQ LDCT,      , H#C, COFFEON

        SEQ     ,      ,    , COFFEON


BUSY:   SEQ RPCT,      , BUSY, BUSYON


        etc.
```

WHICH METHOD IS CORRECT?

BOTH ARE

WHICH SHOULD YOU USE?

EITHER

```
      TITLE COFFEE MACHINE SRC FILE
      ;
      ;LABEL    COFFEE  MACHINE      PHASE 2
      ;---------------------------------------------------
 0000 ZERO:    SEQ CJP, NOCOIN, ZERO     & MACHINE OFF   ; WAIT FOR COIN
 0001          SEQ                       & MACHINE CUPDROP
 0002          SEQ                       & MACHINE BUSYON
 0003          SEQ                       & MACHINE BUSYON
      ;
      ; TURN ON WATER AND TEST TO FIND ROUTINE
      ;
 0004          SEQ CJP, CHOCTST, CHOC    & MACHINE WATERON
 0005          SEQ CJP, SOUPTST, SOUP    & MACHINE WATERON
      ;---------------------------------------------------
      ;
      ; COFFEE BUTTON HAS BEEN PUSHED - DISPENSE COFFEE
      ;
 0006 COFFEE:  SEQ LDCT,        , H#C    & MACHINE COFFEON
 0007          SEQ                       & MACHINE COFFEON
 0008          SEQ                       & MACHINE COFFEON
 0009          SEQ CJP,SUGRTST, SUGAR    & MACHINE COFFEON
 000A          SEQ CJP,CREMTST, CRM2     & MACHINE COFFEON
      ;----------------------------------------------------
      ;
      ; FINISH FILLING THE CUP WITH WATER
      ;
 000B LOOP:    SEQ RPCT,       , LOOP    & MACHINE WATERON
 000C          SEQ LDCT,       , H#4     & MACHINE BUSYON
      ;----------------------------------------------------
      ;
      ; ALLOW TIME TO REMOVE CUP BEFORE STARTING OVER
      ;
 000D BUSY:    SEQ RPCT,       , BUSY    & MACHINE BUSYON
 000E          SEQ JZ                    & MACHINE BUSYON
      ;----------------------------------------------------
      ;
      ; DISPENSE SUGAR
      ;
 000F SUGAR:   SEQ                        & MACHINE COFFEON
 0010          SEQ LDCT,       , H#9      & MACHINE SUGARON
 0011          SEQ CJP,CREMTST, CREAM     & MACHINE SUGARON
 0012          SEQ CJP, PASS   , LOOP     & MACHINE SUGARON
      ;----------------------------------------------------
      ;
      ; DISPENSE CREAM (COFFEE, SUGAR AND CREAM ENTRY)
      ;
 0013 CREAM:   SEQ LDCT,       , H#5      & MACHINE SUGARON
 0014          SEQ                        & MACHINE CREAMON
 0015          SEQ                        & MACHINE CREAMON
 0016 ENTRY:   SEQ                        & MACHINE CREAMON
 0017          SEQ CJP, PASS   , LOOP     & MACHINE CREAMON
      ;----------------------------------------------------
```

```
     ; DISPENSE CREAM (COFFEE AND CREAM ENTRY)
     ;
0018 CRM2:   SEQ LDCT,      , H#8      & MACHINE CREAMON
0019         SEQ CJP, PASS  , ENTRY    & MACHINE CREAMON
     ;---------------------------------------------------
     ;
     ; DISPENSE CHOCOLATE
     ;
001A CHOC:   SEQ                       & MACHINE WATERON
001B         SEQ LDCT,      , H#5      & MACHINE CHOCON
001C CHOCLP: SEQ RPCT,      , CHOCLP   & MACHINE CHOCON
001D         SEQ LDCT,      , H#8      & MACHINE WATERON
001E         SEQ CJP, PASS  , LOOP     & MACHINE WATERON
     ;---------------------------------------------------
     ;
     ; DISPENSE SOUP
     ;
001F SOUP:   SEQ                       & MACHINE SOUPON
0020         SEQ                       & MACHINE SOUPON
0021         SEQ LDCT,      , H#D      & MACHINE SOUPON
0022         SEQ CJP, PASS  , LOOP     & MACHINE SOUPON
     ;--------------------------------------------------
     ;
     ; SKIP OVER MEMORY WHICH IS UNUSED
     ;
003F ORG 63
003F         SEQ JZ                    & MACHINE
     ;
     END
```

```
MDOS/29 AMDASM MICRO ASSEMBLER, V1.4
COFFEE MACHINE SRC FILE

 0000 0011000000000000 00000
 0001 1110001XXXXXX110 00000
 0002 1110001XXXXXX100 00000
 0003 1110001XXXXXX100 00000
 0004 0011011011010101 00000
 0005 0011010011111101 00000
 0006 1100001001100101 10000
 0007 1110001XXXXXX101 10000
 0008 1110001XXXXXX101 10000
 0009 0011101001111101 10000
 000A 0011100011000101 10000
 000B 1001001001011101 00000
 000C 1100001000100100 00000
 000D 1001001001101100 00000
 000E 0000001XXXXXX100 00000
 000F 1110001XXXXXX101 10000
 0010 1100001001001101 01000
 0011 0011100010011101 01000
 0012 0011111001011101 01000
 0013 1100001000101101 01000
 0014 1110001XXXXXX101 00100
 0015 1110001XXXXXX101 00100
 0016 1110001XXXXXX101 00100
 0017 0011111001011101 00100
 0018 1100001001000101 00100
 0019 0011111010110101 00100
 001A 1110001XXXXXX101 00000
 001B 1100001000101101 00010
 001C 1001001011100101 00010
 001D 1100001001000101 00000
 001E 0011111001011101 00000
 001F 1110001XXXXXX101 00001
 0020 1110001XXXXXX101 00001
 0021 1100001001101101 00001
 0022 0011111001011101 00001
 003F 0000001XXXXXX000 00000
```

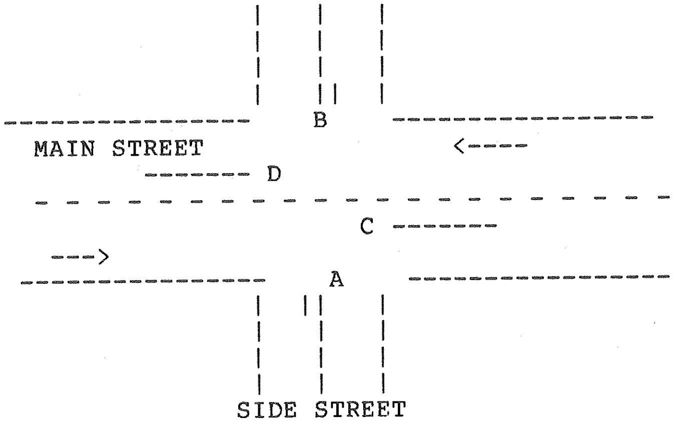
```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.0
COFFEE MACHINE SRC FILE

 SYMBOLS

  BUSY      000D
  BUSYON    0080
  CAFETST   0006
  CHOC      001A
  CHOCLP    001C
  CHOCON    00A2
  CHOCTST   0003
  CJP       0003
  COFFEE    0006
  COFFEON   00B0
  CONT      000E
  CREAM     0013
  CREAMON   00A4
  CREMTST   0004
  CRM2      0018
  CUPDROP   00C0
  ENTRY     0016
  JZ        0000
  LDCT      000C
  LOOP      000B
  NOCOIN    0000
  NULL      0001
  OFF       0000
  PASS      0007
  RPCT      0009
  SOUP      001F
  SOUPON    00A1
  SOUPTST   0002
  SUGAR     000F
  SUGARON   00A8
  SUGRTST   0005
  WATERON   00A0
  ZERO      0000

 TOTAL PHASE 2 ERRORS =    0
```

# DESIGN PROBLEM:

# TRAFFIC LIGHTS

## ADVANCED TRAFFIC LIGHT

You are to design and microcode a controller (using the Am2910!) to handle the following intersection:



There are four lights for straight-through traffic:

| RED | YELLOW | GREEN | |
|---|---|---|---|
| NA | 15s | 80s | MAIN STREET |
| NA | 15s | 40s | SIDE STREET |

There are four lights for protected left turn traffic:

| RED | YELLOW | GREEN ARROW | |
|---|---|---|---|
| NA | 10s | 40s | BOTH STREETS |

The four sensors, A, B, C, D, produce the SLT and MLT signals (side left turn, main left turn).

In case you are wondering, Sunnyvale really has a light that works like this! Precocious students have been known to find it.

To make the problem interesting, there are a few added considerations. If there is an accident, there is a manual override which allows all of the lights to be set to RED FLASH. And, because day traffic is heavier than night traffic, the controller can sense a control signal (timer generated if you wish) that tells it if it is day (normal operation), or night, when the straight-through lights on main street are set to YELLOW FLASH and all others are set to RED FLASH. Remember, you must be able to go back and forth:

DAY <----> NIGHT
DAY <----> MANUAL EMERGENCY
NIGHT <----> MANUAL EMERGENCY

AND

DAY ----> MANUAL EMERGENCY ----> NIGHT
NIGHT ----> MANUAL EMERGENCY ----> DAY

If you have added up all the different lights, there are 5 different states: RED, YELLOW, GREEN, RED FLASH, YELLOW FLASH. We will assume that GREEN and GREEN ARROW are the same.

Five states means three lines of encoded controls into the individual traffic lights:

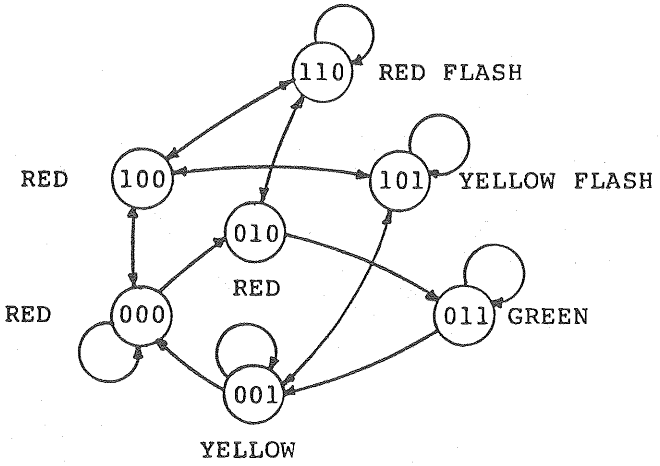
$$I_2 I_1 I_0$$

To make the problem even more interesting, there is a constraint on the way that the lights may be sequenced (remember that you are learning how to implement under constraints - this is a given constraint). Basically, the sequence must be a grey code, ie., only one signal line may change per clock step per light.

YOU HAVE A 5 SECOND CLOCK

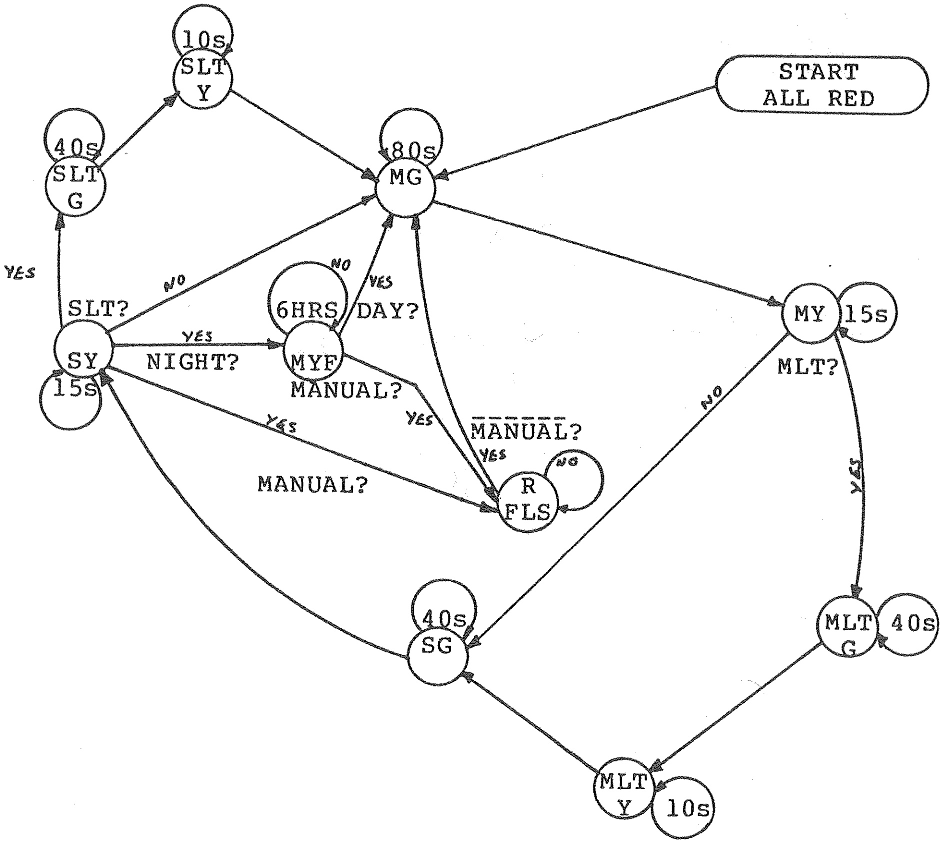
SAMPLE SEQUENCE

| $I_2 I_1 I_0$ | |
|---|---|
| 0 0 0 | RED |
| 0 1 0 | RED |
| 1 0 0 | RED |
| 0 0 1 | YELLOW |
| 0 1 1 | GREEN |
| 1 0 1 | YELLOW FLASH |
| 1 1 0 | RED FLASH |



This is an example and happens to be what the code that is shown in the next pages was written to support. You can modify this slightly and reduce your code (the record so far is 27 microinstructions).

This is the stable-state diagram (does not show transitions). There is a subtle error or two here - you find them!



## BASIC CONTROLLER HARDWARE

```
                                -------------
                                |          |
           MUX         Am2910   |          |
           ----            -----V----<----------CLOCK
MLT ------>|3 |            |    D   |      |
SLT ----->o|2 |            |__   i  |      |
           |  |----------->|CC      |      |
NIGHT ---->|1 |            |__      |      |
MANUAL---->|0 |      ----->|OE      |      |
           ----      |  -->|I       |      |
            ↑SEL.    |  |  --i-------      |
START ----------------  |       |          |
            |      ------       / 6        / 6
            |      |            |          |
            / 2    |   ---------V--------  |
            |      / 4 |                |  |
            |      |   |     PROM       |  |
            |      |   |                |  |
            |      |   ------------------  |
            |      |   | PIPELINE REG.  |<----- CLOCK
            |      |   ------------------  |
            |      |    |   |   |   |      |
            |      -<----   |   |   ------>-
            -<---------------   |
                                / 12
                                |
                                V  LIGHT CONTROL
                                   SIGNALS
```

The output enable of the pipeline register is grounded. The output enables generated by the Am2910 are unused.

## THE MICROPROGRAM (Draft version)

| | | SEQUENCE CONTROL | | | LIGHT CONTROL | | | | |
|---|---|---|---|---|---|---|---|---|---|
| DEC ADR | LABEL | 2910 INSTR | MUX TEST | COUNTER or BRANCH | S | MLT | SLT | M | COMMENT |
| 0 | MAIN | LDCT | | 15 | RO | RO | RO | R2 | LOAD CNTR |
| 1 | MLP | RPCT | | MLP | RO | RO | RO | G | LOOP PL-MAIN |
| 2 | | CONT | | | RO | RO | RO | Y | |
| 3 | | CJP | MLT ? | MLT | RO | RO | RO | Y | MAIN LT TEST |
| 4 | | LDCT | | 7 | R2 | RO | RO | Y | |
| 5 | SIDE | RPCT | | SIDE | G | RO | RO | RO | LOOP PL- SIDE |
| 6 | | CJP | MANUAL? | MANUAL | Y | RO | RO | RO | MANUAL OVERRIDE |
| 7 | | CJP | NIGHT? | NIGHT | Y | RO | RO | RO | NIGHT TEST? |
| 8 | | CJP | $\overline{\text{SLT}}$? | MAIN | Y | RO | RO | RO | SIDE LT TEST |
| 9 | SLT | LDCT | | 7 | RO | RO | R2 | RO | SIDE LT LOOP |
| 10 | SLTLP | RPCT | | SLTLP | RO | RO | G | RO | LOOP PL |
| 11 | | CONT | | | RO | RO | Y | RO | |
| 12 | | JMAP | | MAIN | RO | RO | Y | RO | MAP TIED TO PL |
| 13 | MLT | LDCT | | 7 | RO | R2 | RO | Y | MAIN LT LOOP |
| 14 | MLTLP | RPCT | | MLTLP | RO | G | RO | RO | LOOP PL |
| 15 | | LDCT | | 7 | RO | Y | RO | RO | |
| 16 | | JMAP | | SIDE | R2 | Y | RO | RO | |
| 17 | NIGHT | CONT | | | Y | RO | RO | RO | NIGHT LOOP |
| 18 | | CONT | | | RO | RO | RO | R4 | |
| 19 | | LDCT | | 16 | R4 | R4 | R4 | YF | |
| 20 | NITLP | RPCT | | NITLP | RF | RF | RF | YF | LOOP PL |
| 21 | | LDCT | | 16 | RF | RF | RF | YF | |
| 22 | | CJP | MANUAL? | FIX | RF | RF | RF | YF | MANUAL OVERRIDE? |
| 23 | | CJP | NIGHT? | NITLP | RF | RF | RF | YF | CONTINUE NIGHT? |
| 24 | | CONT | | | R4 | R4 | R4 | R4 | |
| 25 | | JMAP | | MAIN | RO | RO | RO | RO | |
| 26 | MANUAL | CONT | | | Y | RO | RO | RO | MANUAL |
| 27 | | CONT | | | Y | RO | RO | RO | |
| 28 | | CONT | | | RO | RO | RO | RO | |
| 29 | | LDCT | | 16 | R4 | R4 | R4 | R4 | |
| 30 | ENTRY | RPCT | | ENTRY | RF | RF | RF | RF | LOOP PL |
| 31 | | LDCT | | 16 | RF | RF | RF | RF | |
| 32 | | CJP | MANUAL? | ENTRY | RF | RF | RF | RF | CONTINUE MANUAL? |
| 33 | | CONT | | | R4 | R4 | R4 | R4 | |
| 34 | | JMAP | | MAIN | RO | RO | RO | RO | |
| 35 | FIX | JMAP | | ENTRY | RF | RF | RF | R4 | ADJUST SEQ FOR MANUAL |
| • | | | | | | | • | | |
| • | | | | | | | • | | EMPTY AREA |
| • | | | | | | | • | | |
| 63 | | JZ | | MAIN | RO | RO | RO | RO | JUMP ZERO AND RESET |

Review the preceeding microcode carefully and note the following.

a. The actual numerical address is given, in this case in decimal but HEX would have been even better. Whichever you use, note that you label it.

b. Labels are used and they have some relevancy to what is happening in the program. MLT refers to main left turn; NITLP is the RPCT loop for nighttime operation; SIDE is the normal, straight traffic, side street operation, etc.

c. All sequence control (microprogram sequence) fields are grouped together and characterize the individual microinstructions. The Am2910 instruction is given first, followed by the conditional MUX select field (to select which signal line is to be tested), and then the branch address field.

d. The branch address field is an overlay field whose meaning at any particular time is controlled by the Am2910 instruction. This is called "bit steering". The values for the cases when this field is a branch address field are labels. The values for the cases when this field is a counter field are given a decimal numerics. If this field were also a status-mask field for the Am2914, which is possible, those values might be given as HEX values or even better in mnemonics unique to the usage. The point is, when a field serves more than one purpose, make sure that the code is clear as to the usage in any given microinstruction.

e. The controls for the traffic lights are grouped and it is assumed that each field controls two lights. The mnemonics go with the light sequence diagram (R0 = 000 which is RED, R2 = 010 which is also RED, RF and YF are the flashing lights, etc.).

f. Note the comment field. This is important! Comments help someone else read your program and they help you to remember the program when updates, enhancements are being made.

g. Also make note of the vertical and horizontal lines that have been drawn throughout the microprogram. The horizontal lines group the various operations together and each group has a meaningful label. The various vertical lines delineate each field but also group the fields into the microprogram sequence control and light control functions. The vertical lines are not hard to add into the microprogram as you would write it on a AMC SYS/29 (use TAB to allign the fields). The horizontal lines can be easily achieved by using a comment-only line (a semicolon and *s). Whatever you use, the point is to visually separate the code into the different functions for the benefit of the humans who must read it.

h. Examine the testing that occurs following the side street green cycles. There are three tests being made: 1) is there an emergency condition (MANUAL?); 2) is it time to switch to night operation (NIGHT?); and 3) is a protected left turn requested (SLT?). These three tests are being done in the proper priority order, i.e., most important tested first. This is a good example of "polled interrupt" where the microprogram must test for each condition, one at a time.

As an exercise, correct the "errors" that you detect but also reduce the microprogram to fit into three 32x8 PROMS.

# LAB TWO:

# TRAFFIC LIGHT DEF FILE

## LIBRARY

THE AmSYS/29 SYSTEM DISK CONTAINS A FILE CALLED "AM2900.LIB"

THIS FILE CONTAINS DEFINITIONS FOR MOST OF THE Am2900 PARTS, SOME AS COMMENT STATEMENTS, AND DUPLICATION OF MNEMONICS IS PRESENT

YOU ALSO HAVE AM2903.DEF ON YOUR DATA DISK WHICH IS A SUITABLE MASTER FILE FOR AM2903, AM2910, AND A PARTIAL REFERENCE FOR AM2904

BY EDITING THIS FILE A PARTIAL DEFINITION FILE CAN EXIST WITH VERY LITTLE EFFORT - YOU HAVE ALREADY USED THIS TO CREATE PART OF B:LIGHT.DEF (IN LAB ONE)

- POWER UP THE SYSTEMS
- DISPLAY B:LIGHT.DEF
- PRINT THE FILE FOR REFERENCE

```
A>Pc
```

## ASSIGNMENT

USING THE COFFEE.DEF AND THE AM2903.DEF FILES AS REFERENCES YOU ARE TO CREATE THE .DEF FILE FOR THE ADVANCED TRAFFIC LIGHT DESIGN PROBLEM (DISCUSSED IN ED2900A)

- WORK IN TEAMS

- WRITE OUT A DRAFT FILE BEFORE TRYING TO INPUT

- TARGET IS SOME MINIMUM FILE WHICH WILL ALLOW YOU TO WRITE AT LEAST 5 LINES OF MICROCODE

## LAB TWO

- POWER ON ALL UNITS
- SIGN ON TO THE SYSTEM
- USING EDITOR, CREATE THE DEFINITION FILE FOR THE TRAFFIC LIGHT ON THE B DRIVE
- USE THE FILENAME "B:LIGHT.DEF"
- EXIT THE EDITOR WHEN YOU ARE SATISFIED

- ASSEMBLE THE DEFINITION FILE BY TYPING

```
B:(Ret)

AMDASM P1 LIGHT
```

- IF YOU HAVE ERRORS, TYPE:

```
        TYPE B:LIGHT.P1L

                USE Sc TO HALT SCREEN

                USE Pc TO LIST ON PRINTER
```

OR

```
        DISPL B:LIGHT.P1L

                USE Ret TO PAGE FILE

                USE E Ret TO EXIT DISPL PROGRAM
```

- DEBUG USING THE EDITOR

PURPOSE: FAMILIARIZATION WITH THE USE OF THE EDITOR; EXPERIENCE WITH A DEF FILE.

## CREATING A NEW FILE

TYPE --> ED B:LIGHT.DEF

THE SYSTEM WILL COME BACK WITH "NEW FILE" REPLY

ENTER YOUR TEXT (ALWAYS DEVELOP YOUR PROBLEM ON PAPER USING STRUCTURED TECHNIQUES, FLOWCHART, TABLES, ETC.)

WHEN FINISHED,

TYPE --> Zc
B#T

EXAMINE THE FILE FOR ANY ERRORS

IF ERRORS, CORRECT THEM

WHEN COMPLETED,

TYPE --> E

YOU ARE BACK TO THE SYSTEM

```
A>
```

EDITING "LIGHT.DEF" ON THE B DRIVE WHILE SIGNED ON TO THE A DRIVE (A> IS PROMPT):

TYPE --> `ED B:LIGHT.DEF`

THE SYSTEM WILL RESPOND WITH AN ASTERISK AS THE EDITOR PROMPT:

```
*
```

(ALL OF THE FOLLOWING ASSUME A CARRIAGE RETURN)

TYPE --> `#A`

THIS WILL READ THE FILE INTO THE WORKSPACE (MEMORY BUFFER)

TYPE --> `B#T`

THIS DISPLAYS THE ENTIRE FILE ON THE SCREEN

TO CHANGE OR ALTER A LINE THE CP (CHARACTER POINTER) HAS TO BE MOVED TO THE END OF THE LINE

TYPE --> F<line or string to be changed>Zc0LT

0LT (ZERO L T)

TYPE --> KI

THIS KILLS THE OLD LINE AND REQUESTS AN INSERT

NOW TYPE --> <insert the new data>
Zc

(note: <...> means you fill in - DO NOT TYPE THE "<" or ">"!)

THE SYSTEM WILL COME BACK WITH THE "*" PROMPT

TO CHANGE ANY OTHER ERRORS REPEAT THE INSTRUCTION

- ALWAYS GO TO THE TOP OF THE FILE BY TYPING "B" BEFORE SEARCHING FOR A STRING OR A LINE TO BE CHANGED (PROOFING YOUR FILE NEVER HURTS)

- THE "F" (SEARCH) COMMAND ALWAYS GOES FORWARD FROM THE CP POSITION

- AFTER ALL OF THE ERRORS ARE CORRECTED

TYPE --> E

THIS WILL CAUSE YOU TO EXIT EDITOR AND RETURN TO THE SYSTEM YOU WILL SEE:

```
A>
```

TYPE --> B:(Ret)

TYPE --> AMDASM P1 LIGHT

IF THERE ARE ANY ERRORS PRINT OUT A COPY OF THE LISTING AS FOLLOWS:

TURN ON PRINTER

ENABLE PRINTER BY TYPING Pc [IF NOT ENABLED]

TYPE -->

```
TYPE B:LIGHT.PlL
```

(TYPE IS A VALID COMMAND)

GET BACK TO THE EDITOR AND CORRECT THE ERRORS

REASSEMBLE

# LAB THREE:

# TRAFFIC LIGHT SRC FILE

## LAB THREE - SRC FILE

- USING THE EDITOR AS YOU DID FOR LAB TWO
  - CREATE THE FILE

    B:LIGHT.SRC
  - REFER TO YOUR DEF FILE

    B:LIGHT.DEF
- WHEN YOU HAVE FINISHED CREATION
  - PROOF THE FILE BY PRINTING IT OUT AND READING IT
  - CROSS CHECK YOUR MNEMONICS WITH THE DEF FILE

    COMMON ERROR IS MISSPELLING
- ASSEMBLE THE SRC FILE VIA:

```
B>AMDASM P2 LIGHT
```

- CORRECT ERRORS, IF ANY
  - LIST FILE VIA

    ```
    A>TYPE B:LIGHT.P2L
    ```

    Pc SHOULD BE ENABLED
  - DISPLAY FILE VIA:

    ```
    A>DISPL B:LIGHT.P2L
    ```
- REASSEMBLE UNTIL CORRECT

● ● ● YOU NEED AT LEAST FIVE GOOD MICROINSTRUCTIONS ● ● ●

## .SRC DESIGN PROCEDURE

- WRITE OUT THE MICROINSTRUCTIONS BEFORE SIGNING ONTO THE SYSTEM
- HAVE A COPY OF THE DEF FILE HANDY FOR REFERENCE

  DON'T GUESS AT THE CORRECT MNEMONIC
- USE COMMENTS, LOTS OF THEM!
- DEF STATEMENTS CAN BE ADDED TO DEF FILE IF YOU NEED THEM

  (RATHER THAN FREE FORMAT IN .SRC)
- IN ADDITION TO THE DOCUMENTATION IN THE DEF FILE, ADD FORMAT DOCUMENTATION TO THE SRC FILE

  THIS ALLOWS READER TO CHECK WHAT THE FIELD IS SUPPOSED TO BE DOING, ALLOWS CHECK ON MISALIGNMENT (CHECK OUT THE K-1 KIT FILE IN THE MANUAL CAN YOU MAKE SENSE OUT OF IT?)

## EXAMPLE

```
;
; 2910    COND   BR ADDR  2901    2901   2901    Cin  A     B    ...
; INSTR   MUX    COUNTER  SOURCE  FUNC   DESTIN  SEL  ADDR  ADDR ...
; (4)     (2)    (6)      (3)     (3)    (3)     (2)  (4)   (4)  ...
; CONT    FAIL   X        X       X      NOP     LOW  R0    R0   ...
```

# TRAFFIC LIGHT

**.DEF**

```
TITLE TRAFFIC LIGHT DEMO FILE
;
WORD 24
;
; -----------------------
; AM2910 INSTRUCTION SET
; -----------------------
;
; FORMED BY LAB ONE PIP EXERCISE
;
;
JMAP:           EQU     H#2             ; UNCOND JUMP TO MEMORY MAP (Di)
CJP:            EQU     H#3             ; COND JUMP PIPELINE
PUSH:           EQU     H#4             ; PUSH STACK, LOAD REG MAYBE, CONT
JSRP:           EQU     H#5             ; JUMP SUB FROM REG (F) OR PIPE(T)
CJV:            EQU     H#6             ; COND JUMP TO VECTOR INTER (Di)
JRP:            EQU     H#7             ; JUMP TO REG (F) OR PIPE (T)
RFCT:           EQU     H#8             ; DO LOOP REPEAT UNTIL CTR=0 - STACK
RPCT:           EQU     H#9             ; DO LOOP UNTIL CTR=0 - PIPE
CRTN:           EQU     H#A             ; COND RETURN, POP STACK (T)
CJPP:           EQU     H#B             ; COND JUMP PIPELINE, POP STACK
LDCT:           EQU     H#C             ; LOAD REGISTER, CONTINUE
LOOP:           EQU     H#D             ; DO LOOP UNTIL TEST=T - STACK
CONT:           EQU     H#E             ; CONTINUE
;
; TWB WILL BE INCOMPLETE - FIX IT
;
TWB:            EQU     H#F             ; THREE WAY BRANCH (DEAD MAN TIMER)
;
; -------------------------
; CONDITIONAL MUX SELECT
; -------------------------
;
MLT:            EQU     B#11            ; MAIN LEFT TURN REQUEST
NOSLT:          EQU     B#10            ; NO SIDE LEFT TURN REQUEST
NIGHT:          EQU     B#01            ; NIGHT OPERATION
MANUAL:         EQU     B#00            ; EMERGENCY REQUEST
;
; -------------------------
; LIGHT CONTROL SIGNALS
; (SEQUENCE STATES)
; -------------------------
;
R0:             EQU     Q#0
Y:              EQU     Q#1
R2:             EQU     Q#2
G:              EQU     Q#3
R4:             EQU     Q#4
YF:             EQU     Q#5
RF6:            EQU     Q#6
RF7:            EQU     Q#7
;
;
; ---------------------------------
; MICROWORD DEFINITION
; ---------------------------------
;
LITE:   DEF     4VH#E, 2VB#00, 6V$X, 3VQ#7, 3VQ#7, 3VQ#7, 3VQ#7
; DEFAULTS      CONT   MANUAL  X     RF7    RF7    RF7    RF7
;
END
```

# PARTIAL .SRC

```
TITLE TRAFFIC LIGHT SOURCE FILE
;
; TEMPORARY EQUATES SO PARTIAL CODE WILL RUN - REMOVE WHEN
; STATEMENTS WITH THESE LABELS ARE ADDED!!!!!
;
EMER:           EQU     D#26
EVNG:           EQU     D#17
GOTO:           EQU     H#2                             ; GOTO = JMAP
LMLT:           EQU     D#13
;
;
; MAIN STREET SEQUENCE
;------------------------------------------------------------------------
MAIN:   LITE LDCT,  ,   H#F,   R0, R0, R0, R2           ; LOAD COUNTER
MLP:    LITE RPCT,  ,   MLP,   R0, R0, R0, G            ; MAIN GREEN LOOP
        LITE    ,  ,     ,    R0, R0, R0, Y
        LITE CJP, MLT,  LMLT,  R0, R0, R0, Y            ; NEED LEFT TURN?
        LITE LDCT,  ,   H#7,   R2, R0, R0, Y
;-------------------------------------------------------------------------
; SIDE STREET SEQUENCE
;-------------------------------------------------------------------------
SIDE:   LITE RPCT,  ,   SIDE,  G,  R0, R0, R0           ; SIDE GREEN LOOP
        LITE CJP,MANUAL,EMER,  Y,  R0, R0, R0           ; PRIORITY POLLING
        LITE CJP,NIGHT, EVNG,  Y,  R0, R0, R0           ;  SEQUENCE
        LITE CJP,NOSLT, MAIN,  Y,  R0, R0, R0
;-------------------------------------------------------------------------
; SIDE STREET LEFT TURN SEQUENCE
;-------------------------------------------------------------------------
LSLT:   LITE LDCT,  ,   H#7,   R0, R0, R2, R0           ;
SLTLP:  LITE RPCT,  ,  SLTLP,  R0, R0, G,  R0           ; SIDE LEFT LOOP
        LITE    ,  ,       ,  R0, R0, Y,  R0           ; TIME YELLOW LIGHT
        LITE GOTO,  ,   MAIN,  R0, R0, Y,  R0
;--------------------------------------------------------------------------
; ETC.
;
;
END
```

.P2L

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.4
TRAFFIC LIGHT SOURCE FILE

      ;
      ; TEMPORARY EQUATES SO PARTIAL CODE WILL RUN - REMOVE WHEN
      ; STATEMENTS WITH THESE LABELS ARE ADDED!!!!!
      ;
 001A EMER:           EQU     D#26
 0011 EVNG:           EQU     D#17
 0002 GOTO:           EQU     H#2                                   ; GOTO = JMAP
      LMLT:           EQU     D#13
      ;
      ;
      ; MAIN STREET SEQUENCE
 000D ;-----------------------------------------------------------------------
 0000 MAIN:   LITE LDCT,  ,   H#F,   R0, R0, R0, R2                 ; LOAD COUNTER
 0001 MLP:    LITE RPCT,  ,   MLP,   R0, R0, R0, G                  ; MAIN GREEN LOOP
 0002         LITE    ,  ,      ,    R0, R0, R0, Y
 0003         LITE CJP, MLT,  LMLT,  R0, R0, R0, Y                  ; NEED LEFT TURN?
 0004         LITE LDCT,  ,   H#7,   R2, R0, R0, Y
      ;-------------------------------------------------------------------------
      ; SIDE STREET SEQUENCE
      ;-------------------------------------------------------------------------
 0005 SIDE:   LITE RPCT,  ,   SIDE,  G,  R0, R0, R0                 ; SIDE GREEN LOOP
  006         LITE CJP,MANUAL,EMER,  Y,  R0, R0, R0                 ; PRIORITY POLLING
 0007         LITE CJP,NIGHT, EVNG,  Y,  R0, R0, R0                 ;  SEQUENCE
 0008         LITE CJP,NOSLT, MAIN,  Y,  R0, R0, R0
      ;-------------------------------------------------------------------------
      ; SIDE STREET LEFT TURN SEQUENCE
      ;-------------------------------------------------------------------------
 0009 LSLT:   LITE LDCT,  ,   H#7,   R0, R0, R2, R0                 ;
 000A SLTLP:  LITE RPCT,  ,  SLTLP,  R0, R0, G,  R0                 ; SIDE LEFT LOOP
 000B         LITE    ,  ,        ,  R0, R0, Y,  R0                 ; TIME YELLOW
 000C         LITE GOTO,  ,   MAIN,  R0, R0, Y,  R0
      ;--------------------------------------------------------------------------
      ; ETC.
      ;
      ;
      END
```

# CODE

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.4
TRAFFIC LIGHT SOURCE FILE

 0000 1100000011110000 00000010
 0001 1001000000010000 00000011
 0002 111000XXXXXX0000 00000001
 0003 0011110011010000 00000001
 0004 1100000001110100 00000001
 0005 1001000001010110 00000000
 0006 0011000110100010 00000000
 0007 0011010100010010 00000000
 0008 0011100000000010 00000000
 0009 1100000001110000 00010000
 000A 1001000010100000 00011000
 000B 111000XXXXXX0000 00001000
 000C 0010000000000000 00001000
```

# SYMBOL TABLE

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.4
TRAFFIC LIGHT SOURCE FILE

 SYMBOLS

  CJP      0003
  CJPP     000B
  CJV      0006
  CONT     000E
  CRTN     000A
  EMER     001A
  EVNG     0011
  G        0003
  GOTO     0002
  JMAP     0002
  JRP      0007
  JSRP     0005
  LDCT     000C
  LMLT     000D
  LOOP     000D
  LSLT     0009
  MAIN     0000
  MANUAL   0000
  MLP      0001
  MLT      0003
  NIGHT    0001
  NOSLT    0002
  PUSH     0004
  R0       0000
  R2       0002
  R4       0004
  RF6      0006
  RF7      0007
  RFCT     0008
  RPCT     0009
  SIDE     0005
  SLTLP    000A
  TWB      000F
  Y        0001
  YF       0005

 TOTAL PHASE 2 ERRORS =    0
```

# DESIGN PROBLEM:

# A SIMPLE DATA MONITOR

## PROBLEM:

Design a simple data-gatherer such that the data input is read into a 4-bit data-input register. Assume that the data is always ready to be read into the data-in register; we can add a ready-to-receive bit later for "handshaking". When data is output, the monitor waits for an ACK signal before proceeding with its operation. The monitor is to have a 12-bit RALU. A minimum of five registers are to behave as counters.

## DEVICES:

Use an Am2910 and an ALU made up out of Am2901s.

## DESIGN APPROACH:

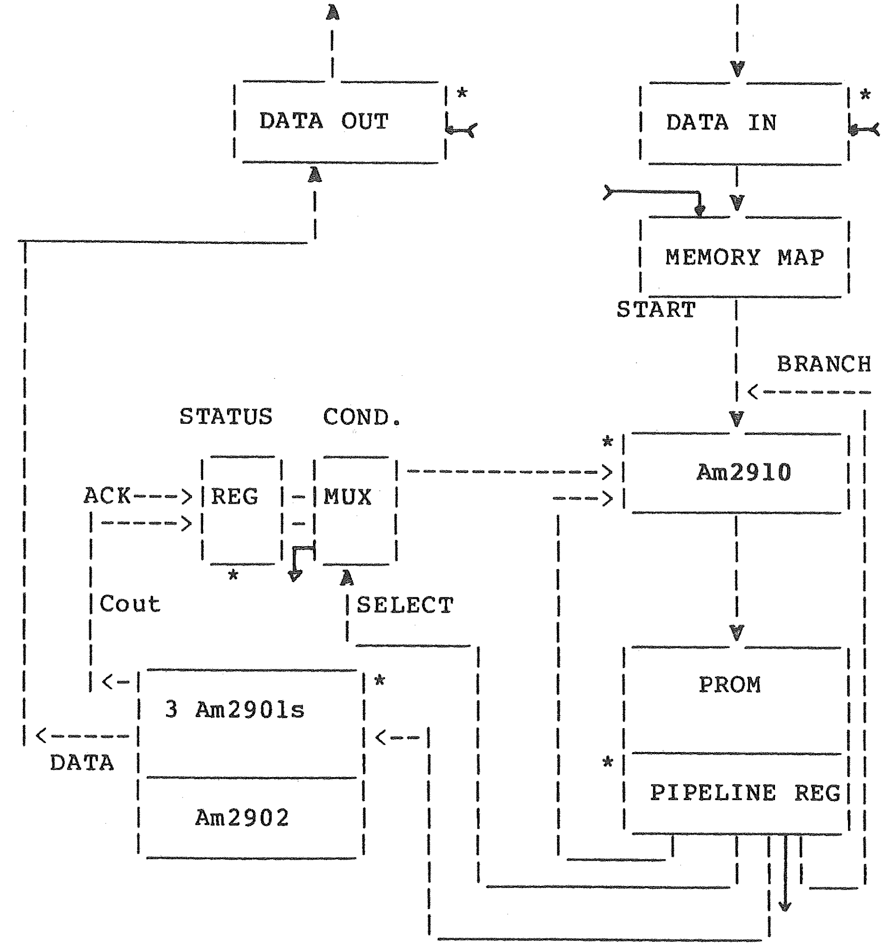
Use a pipelined PROM control memory, a status register (1-bit) and a memory map to decode the data input.

## MICROPROGRAM DESCRIPTION:

```
START
    1. INITIALIZE REGISTERS
         $R_0$ <-- 0    $R_1$ <-- 0    $R_2$ <-- 0    $R_3$ <-- 0    $R_4$ <-- 0
NEXT:2. LOAD DATA-IN REGISTER
          DATA-IN <-- DATA-BUS
    3. IF DATA-IN = 0
            THEN $R_0$ <-- $R_0$ + 1
                 IF $C_{n+4}$ = 1
                     THEN JUMP SUB0
                          $R_1$ --> DATA-OUT
           ELSE $R_1$ <-- $R_1$ + 1          ALL NUMBERS ARE POSITIVE
                IF $C_{n+4}$ = 1
                    THEN JUMP SUB0
                         $R_0$ --> DATA-OUT
    4. CASE BRANCH
       IF $0 \leq$ DATA-IN $\leq 5$
           THEN $R_2$ <-- $R_2$ + 1
       IF $5 <$ DATA-IN $\leq 10$
           THEN $R_3$ <-- $R_3$ + 1
       IF $10 <$ DATA-IN
           THEN $R_4$ <-- $R_4$ + 1
    5. BRANCH TO NEXT
```

```
SUB0:6. $R_2$ --> DATA-OUT
    7. IF $\overline{ACK}$
          THEN WAIT
    8. $R_3$ --> DATA-OUT
    9. IF $\overline{ACK}$
          THEN WAIT
    10. $R_4$ --> DATA-OUT
    11. IF $\overline{ACK}$
           THEN WAIT
    12. RESET REGISTERS
          $R_2$ <-- 0     $R_3$ <-- 0     $R_4$ <-- 0
    13. RETURN
```

## BASIC HARDWARE



* CLOCK INTO:

- Am2910
- Am2901s
- PIPELINE REGISTER
- DATA REGISTERS (12 BITS AND 4 BITS)
- STATUS REGISTER (2 BITS)

## MICROWORD FORMAT [ DRAFT VERSION ]

| LABEL ADDR | 2910 INSTR | COND MUX | BR ADDR COUNTER | SRCE A | FUNC L | DEST U | CARRY IN | A ADDR | B ADDR | DATA IN CTRL | DATA OUT CTRL |
|---|---|---|---|---|---|---|---|---|---|---|---|

The final number of bits that are required for the microword is a function of the final microprogram. An initial guess can be made by examining what exists so far in the microword format.

| | | |
|---|---|---|
| 1. | LABEL/ADDR | This will be the actual in-place address and is filled in last. Labels are filled in as created. |
| 2. | 2910 INSTR | The 4-bit instruction field for the microprogram controller |
| 3. | COND MUX | The conditional MUX select control is 1 bit so far. |
| 4. | BR ADDR/COUNTER | The branch address field 4-6 bits is a good guess so far. |
| 5. | ALU CONTROL | These three fields are each 3 bits. |
| 6. | CARRY IN | For the ALU, 1 bit. |
| 7. | A ADDR<br>B ADDR | Selects the ALU registers, need 5 registers, so two 3 bit fields. |
| 8. | DATA IN CTRL | Load the DATA-IN register, 1 bit. |
| 9. | DATA OUT CTRL | Load the DATA-OUT register, 1 bit. |

Both 8 and 9 will expand to 2 bits if enables are also needed.

Total # bits: 28 bits ±1

| LABEL /ADDR | 2910 INSTR | COND MUX | BR ADDR COUNTER | SRCE A | FUNC L | DEST U | C IN | ADDR A | ADDR B | DATA-IN CONTROL | DATA-OUT CONTROL | MEMORY MAP SEL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * FIRST WRITE CODE TO INITIALIZE REGISTERS | | | | | | | | | | | | |
| START | CONT | # | # | AB | EXOR | RAMF | # | R0 | R0 | NOLOAD | NOLOAD | 0 |
| 1 | CONT | # | # | AB | EXOR | RAMF | # | R1 | R1 | NOLOAD | NOLOAD | 0 |
| 2 | CONT | # | # | AB | EXOR | RAMF | # | R2 | R2 | NOLOAD | NOLOAD | 0 |
| 3 | CONT | # | # | AB | EXOR | RAMF | # | R3 | R3 | NOLOAD | NOLOAD | 0 |
| 4 | CONT | # | # | AB | EXOR | RAMF | # | R4 | R4 | NOLOAD | NOLOAD | 0 |
| NEXT | CONT | # | # | # | # | NOP | # | # | # | LOAD | NOLOAD | 0 |

NOW CONSIDER HOW YOU ARE GOING TO HANDLE THE MULTIPLE BRANCHES? ONE SOLUTION WHICH HAS BEEN STARTED HERE IS TO HAVE A LARGER MEMORY MAP AND AN EXTRA ADDRESS LINE INTO IT SO THAT THE INPUT DATA CAN BE DECODED TWICE USING DIFFERENT PORTIONS OF THE MAP FOR EACH ACCESS.

ANOTHER SOLUTION WOULD BE TO READ THE DATA INTO AN ALU REGISTER AND USE THE ZERO TEST STATUS TO MAKE THE FIRST DECISION AND USE THE MEMORY MAP FOR THE CASE BRANCH DECISION. THE MICROWORD WOULD BE THE SAME WIDTH IN BOTH CASES (THE Z STATUS REQUIRES A WIDER STATUS REGISTER AND CONDITION CODE MUX).

WE WILL EXAMINE BOTH SOLUTIONS.

THE DOUBLE MAP:

| LABEL /ADDR | 2910 INSTR | COND MUX | BR ADDR COUNTER | SRCE A | FUNC L | DEST U | C IN | ADDR A | ADDR B | DATA-IN CONTROL | DATA-OUT CONTROL | MEMORY MAP SEL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | JMAP | # | # | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| DISZ | CONT | # | # | AZ | ADD | RAMF | H | R0 | R0 | NOLOAD | NOLOAD | 0 |
| 8 | JSUB | Cn+4 | SUB0 | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| 9 | CONT | # | # | AB | OR | RAMA | # | R1 | R1 | NOLOAD | LOAD | 0 |
| WAITA | CJP | NOACK | WAITA | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| B | CJP | PASS | CASE | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 1 |
| DGRZ | CONT | # | # | AZ | ADD | RAMF | H | R1 | R1 | NOLOAD | NOLOAD | 0 |
| D | JSUB | Cn+4 | SUB0 | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| E | CONT | # | # | AB | OR | RAMA | # | R0 | R0 | NOLOAD | LOAD | 0 |
| WAITB | CJP | NOACK | WAITB | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 1 |
| CASE | JMAP | # | # | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 1 |
| L5 | CJP | PASS | NEXT | AZ | ADD | RAMF | H | R2 | R2 | NOLOAD | NOLOAD | 0 |
| GR5L10 | CJP | PASS | NEXT | AZ | ADD | RAMF | H | R3 | R3 | NOLOAD | NOLOAD | 0 |
| GR10 | CJP | PASS | NEXT | AZ | ADD | RAMF | H | R4 | R4 | NOLOAD | NOLOAD | 0 |
| *SUBROUTINE | | | | | | | | | | | | |
| SUB0 | CONT | # | # | AB | OR | RAMA | # | R2 | R2 | NOLOAD | LOAD | 0 |
| WAIT1 | CJP | NOACK | WAIT1 | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| 16 | CONT | # | # | AB | OR | RAMA | # | R3 | R3 | NOLOAD | LOAD | 0 |
| WAIT2 | CJP | NOACK | WAIT2 | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| 18 | CONT | # | # | AB | OR | RAMA | # | R4 | R4 | NOLOAD | LOAD | 0 |
| WAIT3 | CJP | NOACK | WAIT3 | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |
| 1A | CONT | # | # | AB | EXOR | RAMF | # | R2 | R2 | NOLOAD | NOLOAD | 0 |
| 1B | CONT | # | # | AB | EXOR | RAMF | # | R3 | R3 | NOLOAD | NOLOAD | 0 |
| 1C | CONT | # | # | AB | EXOR | RAMF | # | R4 | R4 | NOLOAD | NOLOAD | 0 |
| 1D | CRTN | PASS | # | # | # | NOP | # | # | # | NOLOAD | NOLOAD | 0 |

TOTAL NUMBER OF MICROWORDS = 1D (HEX) = 29

TOTAL NUMBER OF BRANCH ADDRESS LINES = 5

# AM2900.LIB FILE

## (1976-7)

```
; TITLE          T H E   A M 2 9 0 0   F A M I L Y   M N E M O N I C S
;           **************************************************
;
;
WORD 1          ; INSERTED AS DUMMY TO PROCESS DEFINITIONS
;13 DECEMBER 1976 JRM
;UPDATED SEPT 28, 1977
;
;AM2901 INSTRUCTION SET
;
;REGISTER DEFINITIONS
;
R0:     EQU     H#0
R1:     EQU     H#1
R2:     EQU     H#2
R3:     EQU     H#3
R4:     EQU     H#4
R5:     EQU     H#5
R6:     EQU     H#6
R7:     EQU     H#7
R8:     EQU     H#8
R9:     EQU     H#9
R10:    EQU     H#A
R11:    EQU     H#B
R12:    EQU     H#C
R13:    EQU     H#D
R14:    EQU     H#E
R15:    EQU     H#F
;
;AM2901 SOURCE OPERANDS (R S)
;
AQ:     EQU     Q#0
AB:     EQU     Q#1
ZQ:     EQU     Q#2
ZB:     EQU     Q#3
ZA:     EQU     Q#4
DA:     EQU     Q#5
DQ:     EQU     Q#6
DZ:     EQU     Q#7
;
;AM2901 ALU FUNCTIONS (R FUNCTION S)
;
ADD:    EQU     Q#0
SUBR:   EQU     Q#1
SUBS:   EQU     Q#2
OR:     EQU     Q#3
AND:    EQU     Q#4
NOTRS:  EQU     Q#5
EXOR:   EQU     Q#6
EXNOR:  EQU     Q#7
;
;AM2901 DESTINATION CONTROL
;
QREG:   EQU     Q#0
NOP:    EQU     Q#1
```

```
RAMA:   EQU     Q#2
RAMF:   EQU     Q#3
RAMQD:  EQU     Q#4
RAMD:   EQU     Q#5
RAMQU:  EQU     Q#6
RAMU:   EQU     Q#7
;
;AM29811 INSTRUCTION SET
;
JZ:     EQU     H#0     ;JUMP TO ADDRESS ZERO
CJS:    EQU     H#1     ;CONDITIONAL JUMP TO SUBROUTINE WITH JUMP
                        ;ADDRESS IN THE PIPELINE REGISTER
JMAP:   EQU     H#2     ;JUMP TO ADDRESS AT MAPPING PROM OUTPUT
CJP:    EQU     H#3     ;CONDITIONAL JUMP TO ADDRESS IN PIPELINE
                        ;REGISTER
PUSH:   EQU     H#4     ;PUSH STACK AND CONDITIONALLY LOAD COUNTER
JSRP:   EQU     H#5     ;JUMP TO SUBROUTINE WITH STARTING ADDRESS
                        ;CONDITIONALLY SELECTED FROM THE AM2911
                        ;R-REGISTER OR PIPELINE ADDRESS
CJV:    EQU     H#6     ;CONDITIONAL JUMP TO VECTOR ADDRESS
JRP:    EQU     H#7     ;JUMP TO ADDRESS CONDITIONALLY SELECTED FROM
                        ;AM2911 R-REGISTER OR PIPELINE REGISTER
RFCT:   EQU     H#8     ;REPEAT LOOP IF COUNTER IS NOT EQUAL TO ZERO
RPCT:   EQU     H#9     ;REPEAT PIPELINE ADDRESS IF COUNTER IS NOT
                        ;EQUAL TO ZERO
CRTN:   EQU     H#A     ;CONDITIONAL RETURN FROM SUBROUTINE
CJPP:   EQU     H#B     ;CONDITIONAL JUMP TO PIPELINE ADDRESS AND POP
                        ;STACK
LDCT:   EQU     H#C     ;LOAD COUNTER AND CONTINUE
LOOP:   EQU     H#D     ;TEST END OF LOOP
CONT:   EQU     H#E     ;CONTINUE TO NEXT ADDRESS
JP:     EQU     H#F     ;JUMP TO PIPELINE REGISTER ADDRESS
;
;AM2910 MICROPROGRAM SEQUENCER
;
;DELETE ";" FROM AM2910 NMEMONICS IF NEEDED
;(DELETE IN EDITOR USING D-RETURN-RETURN)
;
;JZ:    EQU     H#0     ;JUMP ZERO (RESET)
;CJS:   EQU     H#1     ;CONDITIONAL JUMP SUBROUTINE PIPELINE
;JMAP:  EQU     H#2     ;JUMP MAP
;CJP:   EQU     H#3     ;CONDITIONAL JUMP PIPELINE
;PUSH:  EQU     H#4     ;PUSH/CONDITIONAL LOAD COUNTER
;JSRP:  EQU     H#5     ;CONDITIONAL JUMP SUBROUTINE R OR PIPELINE
;CJV:   EQU     H#6     ;CONDITIONAL JUMP VECTOR
;JRP:   EQU     H#7     ;CONDITIONAL JUMP R OR PIPELINE
;RFCT:  EQU     H#8     ;REPEAT LOOP, COUNTER NOT ZERO
;RPCT:  EQU     H#9     ;REPEAT PIPELINE, COUNTER NOT ZERO
;CRTN:  EQU     H#A     ;CONDITIONAL RETURN
;CJPP:  EQU     H#B     ;CONDITIONAL JUMP PIPELINE AND POP
;LDCT:  EQU     H#C     ;LOAD COUNTER AND CONTINUE
;LOOP:  EQU     H#D     ;TEST END LOOP
;CONT:  EQU     H#E     ;CONTINUE
;TWB:   EQU     H#F     ;THREE-WAY BRANCH
                ;THREE-WAY DEFINITION
```

```
                    ;PASS TEST - CONTINUE PC AND POP
                    ;FAIL TEST - REPEAT LOOP IF COUNTER NOT ZERO
                    ;FAIL TEST - JUMP PIPELINE AND POP IF COUNTER ZERO
;
;AM2914 INSTRUCTION SET
;
MCLR:     EQU       H#0       ;MASTER CLEAR
CLRIN:    EQU       H#1       ;CLEAR ALL INTERRUPTS
CLRMB:    EQU       H#2       ;CLEAR INTERRUPTS FROM M-BUS
CLRMR:    EQU       H#3       ;CLEAR INTERRUPTS FROM MASK REGISTER
CLRVC:    EQU       H#4       ;CLEAR INTERRUPT FROM LAST VECTOR READ
RDVC:     EQU       H#5       ;READ VECTOR
RDSTA:    EQU       H#6       ;READ STATUS REGISTER
RDM:      EQU       H#7       ;READ MASK REGISTER
SETM:     EQU       H#8       ;SET MASK REGISTER
LDSTA:    EQU       H#9       ;LOAD STATUS REGISTER
BCLRM:    EQU       H#A       ;BIT CLEAR MASK REGISTER
BSETM:    EQU       H#B       ;BIT SET MASK REGISTER
CLRM:     EQU       H#C       ;CLEAR MASK REGISTER
DISIN:    EQU       H#D       ;DISABLE INTERRUPT REQUEST
LDM:      EQU       H#E       ;LOAD MASK REGISTER
ENIN:     EQU       H#F       ;ENABLE INTERRUPT REQUEST
;
;AM2930 PROGRAM CONTROL UNIT
;
;NON-CONDITIONAL INSTRUCTIONS
;
PRST:     EQU       5H#00:    ;RESET
FPC:      EQU       5H#01:    ;FETCH PC
FR:       EQU       5H#02:    ;FETCH R
FD:       EQU       5H#03:    ;FETCH D
FRD:      EQU       5H#04:    ;FETCH R PLUS D
FPD:      EQU       5H#05:    ;FETCH PC PLUS D
FPR:      EQU       5H#06:    ;FETCH PC PLUS R
FSD:      EQU       5H#07:    ;FETCH S PLUS D
FPLR:     EQU       5H#08:    ;FETBH PC, LOAD R
FRDR:     EQU       5H#09:    ;FETCH R PLUS D, LOAD R
PLDR:     EQU       5H#0A:    ;LOAD R
PSHP:     EQU       5H#0B:    ;PUSH PC
PSHD:     EQU       5H#0C:    ;PUSH D
POPS:     EQU       5H#0D:    ;POP S
POPP:     EQU       5H#0E:    ;POP PC
PHLD:     EQU       5H#0F:    ;HOLD
;
;CONDITIONAL INSTRUCTIONS - FAIL TEST, EXECUTE FPC
;
JMPR:     EQU       5H#10:    ;JUMP R
JMPD:     EQU       5H#11:    ;JUMP D
JMPZ:     EQU       5H#12:    ;JUMP ZERO
JPRD:     EQU       5H#13:    ;JUMP R PLUS D
JPPD:     EQU       5H#14:    ;JUMP PC PLUS D
JPPR:     EQU       5H#15:    ;JUMP PC PLUS R
JSBR:     EQU       5H#16:    ;JUMP SUBROUTINE R
JSBD:     EQU       5H#17:    ;JUMP SUBROUTINE D
JSBZ:     EQU       5H#18:    ;JUMP SUBROUTINE ZERO
```

```
JSRD:   EQU     5H#19:  ;JUMP SUBROUTINE R PLUS D
JSPD:   EQU     5H#1A:  ;JUMP SUBROUTINE PC PLUS D
JSPR:   EQU     5H#1B:  ;JUMP SUBROUTINE PC PLUS R
RTS:    EQU     5H#1C:  ;RETURN S
RTSD:   EQU     5H#1D:  ;RETURN S PLUS D
CHLD:   EQU     5H#1E:  ;HOLD
PSUS:   EQU     5H#1F:  ;SUSPEND
;
;
;AM2932 PROGRAM CONTROL UNIT
;
;DELETE ";" FROM AM2932 NMEMONICS IF NEEDED
;
;PRST:  EQU     H#0     ;RESET
;PSUS:  EQU     H#1     ;SUSPEND
;PSHD:  EQU     H#2     ;PUSH D
;POPS:  EQU     H#3     ;POP STACK
;FPC:   EQU     H#4     ;FETCH PC
;JMPD:  EQU     H#5     ;JUMP D
;PSHP:  EQU     H#6     ;PUSH PC
;RTS:   EQU     H#7     ;RETURN STACK
;FR:    EQU     H#8     ;FETCH R
;FPR:   EQU     H#9     ;FETCH PC PLUS R
;FPLR:  EQU     H#A     ;FETCH PC, LOAD R
;JMPR:  EQU     H#B     ;JUMP R
;JPPR:  EQU     H#C     ;JUMP PC PLUS R
;JSBR:  EQU     H#D     ;JUMP SUBROUTINE R
;JSPR:  EQU     H#E     ;JUMP SUBROUTINE PC PLUS R
;PLDR:  EQU     H#F     ;LOAD R
;
;AM2940 DMA CONTROL UNIT
;
;INSTRUCTIONS
;
WRCR:   EQU     Q#0     ;WRITE CONTROL REGISTER
RDCR:   EQU     Q#1     ;READ CONTROL REGISTER
RDWC:   EQU     Q#2     ;READ WORD COUNTER
RDAC:   EQU     Q#3     ;READ ADDRESS COUNTER
REIN:   EQU     Q#4     ;REINITIALIZE COUNTERS
LDAD:   EQU     Q#5     ;LOAD ADDRESS
LDWC:   EQU     Q#6     ;LOAD WORD COUNT
ENCT:   EQU     Q#7     ;ENABLE COUNTERS
;
;CONTROL MODE BYTE
;NOTE - BITS 3 THROUGH 7 ARE DON'T CARE
;
WC1I:   EQU     8Q#0%   ;WORD COUNT EQUALS ONE, INCREMENT ADDRESS COUNTER
WCCI:   EQU     8Q#1%   ;WORD COUNT COMPARE, INCREMENT ADDRESS COUNTER
ADCI:   EQU     8Q#2%   ;ADDRESS COMPARE, INCREMENT ADDRESS COUNTER
WCOI:   EQU     8Q#3%   ;WORD COUNTER CARRY OUT, INCREMENT ADDRESS COUNTER
WC1D:   EQU     8Q#4%   ;WORD COUNT EQUALS ONE, DECREMENT ADDRESS COUNTER
WCCD:   EQU     8Q#5%   ;WORD COUNT COMPARE, DECREMENT ADDRESS COUNTER
ADCD:   EQU     8Q#6%   ;ADDRESS COMPARE, DECREMENT ADDRESS COUNTER
WCOD:   EQU     8Q#7%   ;WORD COUNTER CARRY OUT, DECREMENT ADDRESS COUNTER
;
```

```
; AM2903 INSTRUCTION SET  (7TH NOVEMBER, 1978    TD)
;
; ALU OPERAND SOURCES
;
; ALU SOURCE OPERAND R  (EA*)
;
RAMA:   EQU     B#0     ; R INPUT FROM RAM PORT A
DA:     EQU     B#1     ; R INPUT FROM DA BUS
;
; ALU SOURCE OPERAND S  (I0,OEB*)
;
RAMB:   EQU     B#00    ; S INPUT FROM RAM PORT B
DB:     EQU     B#01    ; S INPUT FROM DB BUS
Q:      EQU     B#10    ; IF (I4-I1)=0 THEN ALU OUTPUT F IS FORCED HIGH
                        ;    REGARDLESS OF INPUT,
                        ; ELSE S INPUT FROM Q REGISTER
;
; ALU FUNCTIONS  (I4,I3,I2,I1)
;
SPF:    EQU     H#0     ; SPECIAL FUNCTIONS, S OPERAND RAMB OR DB, BUT NOT Q
HIGH:   EQU     H#0     ; ALU OUTPUT F FORCED HIGH, S OPERAND MUST SPECIFY Q
SUBR:   EQU     H#1     ; S MINUS R
SUBS:   EQU     H#2     ; R MINUS S
ADD:    EQU     H#3     ; R ADD S
PASSS:  EQU     H#4     ; PASS S
COMPLS: EQU     H#5     ; 2'S COMPLEMENT S
PASSR:  EQU     H#6     ; PASS R
COMPLR: EQU     H#7     ; 2'S COMPLEMENT R
LOW:    EQU     H#8     ; ALU OUTPUT FORCED LOW
NOTRS:  EQU     H#9     ; COMPLEMENT R, AND WITH S
EXNOR:  EQU     H#A     ; R EXCLUSIVE NOR WITH S
EXOR:   EQU     H#B     ; R EXOR S
AND:    EQU     H#C     ; R AND S
NOR:    EQU     H#D     ; R NOR S
NAND:   EQU     H#E     ; R NAND S
OR:     EQU     H#F     ; R OR S
;
; SPECIAL FUNCTIONS  (I8,I7,I6,I5)
; I8-I5 SPECIFIES SPECIAL FUNCTIONS IF ALU=SPF, AND S OPERAND IS RAMB OR DB
;    OTHERWISE IT SPECIFIES ALU DESTINATION CONTROL
;
USMUL:  EQU     H#0     ; UNSIGNED MULTIPLY
TCMUL:  EQU     H#2     ; 2'S COMPLEMENT MULTIPLY
INC:    EQU     H#4     ; INCREMENT BY ONE OR TWO
SMTC:   EQU     H#5     ; SIGN MAGNITUDE - 2'S COMPLEMENT
TCMLS:  EQU     H#6     ; 2'S COMPLEMENT MULTIPLY LAST STEP
SLN:    EQU     H#8     ; SINGLE LENGTH NORMALIZE
DLN:    EQU     H#A     ; DOUBLE LENGTH NORMALIZE
TCDIV:  EQU     H#C     ; 2'S COMPLEMENT DIVISION
TCDC:   EQU     H#E     ; 2'S COMPLEMENT DIVISION CORRECTION
;
; ALU DESTINATION CONTROL  (I8,I7,I6,I5)
; I8-I5 SPECIFIES ALU DESTINATION CONTROL ONLY IF ALU IS NOT SPF
```

```
;
ADR:      EQU      H#0      ; ARITH SHIFT DOWN, RESULT INTO RAM
LDR:      EQU      H#1      ; LOGICAL SHIFT DOWN, RESULT INTO RAM
ADRQ:     EQU      H#2      ; ARITH SHIFT DOWN, RESULT INTO RAM AND Q
LDRQ:     EQU      H#3      ; LOGICAL SHIFT DOWN, RESULT INTO RAM AND Q
RPT:      EQU      H#4      ; RESULT INTO RAM, GENERATE PARITY
LDQP:     EQU      H#5      ; LOGICAL SHIFT DOWN Q, GENERATE PARITY
QPT:      EQU      H#6      ; RESULT INTO Q, GENERATE PARITY
RQPT:     EQU      H#7      ; RESULT INTO RAM AND Q, GENERATE PARITY
AUR:      EQU      H#8      ; ARITH SHIFT UP, RESULT INTO RAM
LUR:      EQU      H#9      ; LOGICAL SHIFT UP, RESULT INTO RAM
AURQ:     EQU      H#A      ; ARITH SHIFT UP, RESULT INTO RAM AND Q
LURQ:     EQU      H#B      ; LOGICAL SHIFT UP, RESULT INTO RAM AND Q
YBUS:     EQU      H#C      ; RESULT TO Y BUS ONLY
LUQ:      EQU      H#D      ; LOGICAL SHIFT UP Q
SEX:      EQU      H#E      ; SIGN EXTEND
RSEX:     EQU      H#F      ; RESULT TO RAM, SIGN EXTEND
;
          END
```

# MASTER .DEF FILE

## Am2903-2904-2910

## (1980-1)

MASTER .DEF FILE FOR Am2903-2904-2910

- AM2903.DEF listing
- AM2903.SRC listing
  - SAMPLE CODE FOR THE Am2903-2910 from the 2900 Family Study Guide

# AM2903.DEF

```
TITLE EXAMPLE DEFINITION FILE FOR THE AM2903/29203
;
;
WORD    64
;
; AM2903 INSTRUCTION SET
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
; ALU SOURCE OPERANDS (EA, I0, OEB)
; 16 REGISTER - TWO ADDRESS VERSION
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
RAMAB:  EQU     Q#0                 ; RAM A PORT, RAM B PORT
RAMADB: EQU     Q#1                 ; RAM A PORT, DATA BUS B
RAMAQ:  EQU     Q#2                 ; OR Q#3 - RAM A PORT, Q REGISTER
DARAMB: EQU     Q#4                 ; DATA BUS A, RAM B PORT
DADB:   EQU     Q#5                 ; DATA BUS A, DATA BUS B
DAQ:    EQU     Q#6                 ; OR Q#7 - DATA BUS A, Q REGISTER
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; ALU FUNCTIONS - NORMAL MODE (I4, I3, I2, I1, I0 ALL NOT 0)
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
SPECL:  EQU     H#0                 ; I0 MUST BE LOW
HIGH:   EQU     H#0                 ; I0 MUST BE HIGH (Q REGISTER SELECT)
SUBR:   EQU     H#1                 ; F = S - R - 1 + Cin
SUBS:   EQU     H#2                 ; F = R - S - 1 + Cin
ADD:    EQU     H#3                 ; F = R + S + Cin
INCRS:  EQU     H#4                 ; F = S + Cin
INCSNON:EQU     H#5                 ; F = NOT S + Cin
INCRR:  EQU     H#6                 ; F = R + Cin
INCRNON:EQU     H#7                 ; F = NOT R + Cin
LOW:    EQU     H#8                 ; F = LOW
NOTRS:  EQU     H#9                 ; F = NOT R AND S
EXNOR:  EQU     H#A                 ; F = R EXNOR S
EXOR:   EQU     H#B                 ; F = R EXOR S
AND:    EQU     H#C                 ; F = R AND S
NOR:    EQU     H#D                 ; F = R NOR S
NAND:   EQU     H#E                 ; F = R NAND S
OR:     EQU     H#F                 ; F = R OR S
```

```
; * * * * * * * * * * * * * * * * * * * * * * *
;
; ALU DESTINATION CONTROL ( I8 - I7 - I6 - I5)
;     NORMAL FUNCTIONS
; * * * * * * * * * * * * * * * * * * * * * * *
;
;
RAMDA:   EQU      H#0               ; F TO RAM, ARITHMETIC DOWN SHIFT
RAMDL:   EQU      H#1               ; F TO RAM, LOGICAL DOWN SHIFT
RAMQDA:  EQU      H#2               ; DOUBLE PRECISION ARITHMETIC DOWN SHIFT
RAMQDL:  EQU      H#3               ; DOUBLE PRECISION LOGICAL DOWN SHIFT
RAM:     EQU      H#4               ; F TO RAM WITH PARITY
QD:      EQU      H#5               ; F TO Y, DOWN SHIFT Q
LOADQ:   EQU      H#6               ; F TO Q WITH PARITY
RAMQ:    EQU      H#7               ; F TO RAM AND Q WITH PARITY
RAMUPA:  EQU      H#8               ; F TO RAM, ARITHMETIC UP SHIFT
RAMUPL:  EQU      H#9               ; F TO RAM, LOGICAL UP SHIFT
RAMQUPA:EQU       H#A               ; DOUBLE PRECISION ARITHMETIC UP SHIFT
RAMQUPL:EQU       H#B               ; DOUBLE PRECISION LOGICAL UP SHIFT
YBUS:    EQU      H#C               ; F TO Y ONLY
QUP:     EQU      H#D               ; F TO Y, UP SHIFT Q
SIGNEXT:EQU       H#E               ; SIO0 TO Yi
RAMEXT:  EQU      H#F               ; F TO Y, SIGN EXTEND LEAST SIG. BYTE
;
; * * * * * * * * * * * * * * * ** * * * * * * * * * * * *
;
; SPECIAL FUNCTIONS (I8-I7-I6-I5)
;
;
; Am2903 FUNCTIONS ONLY
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
MULT:    EQU      H#0               ; UNSIGNED MULTIPLY
TWOMULT:EQU       H#2               ; TWO'S COMPLEMENT MULTIPLY
TWOLAST:EQU       H#6               ; TWO'S COMPLEMENT MULTIPLY LAST STEP
INCRMNT:EQU       H#4               ; INCREMENT BY 1 + Cin
SGNTWO:  EQU      H#5               ; SIGN MAGNITUDE-TWO'S COMPL CONVERTION
SLN:     EQU      H#8               ; SINGLE LENGTH NORMALIZE
DLN:     EQU      H#A               ; DOUBLE LENGTH NORMALIZE
DIVFRST:EQU       H#A               ; TWO'S COMPLEMENT DIVIDE FIRST STEP
DIVIDE:  EQU      H#C               ; TWO'S COMPLEMENT DIVIDE MIDDLE STEPS
DIVLAST:EQU       H#E               ; TWO'S COMPLEMENT DIVIDE LAST STEP
;
;
;
; Am29203 ADDITIONS FORTHCOMING
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
; DEFINITION FILE FOR FIGURE 29, PAGE 2-57, 1980 DATA BOOK
;
; EXPANDED MEMORY FOR THE Am2903 USING THE Am20705
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; ONLY THE SOURCE FIELDS CHANGE - EXPANDED
; A THIRD ADDRESS FIELD WAS ALSO ADDED (ADDRESS C)
;
;
;
; SOURCE OPERANDS
; ADDED A ADDRESS BITS (A6-A5-A4)
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
AINALU: EQU     Q#0             ; A ADDRESSES 2903 REGISTERS
AIS7051:EQU     Q#1             ; A ADDRESSES FIRST 29705 ADDITION
AIS7052:EQU     Q#2             ; A ADDRESSES SECOND 29705 ADDITION
ACONST: EQU     Q#3             ; A ADDRESSES CONSTANT PROM
ABUS:   EQU     Q#4             ; A FROM BUS
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; ADDED B ADDRESS BITS (B5-B4)
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
BINALU: EQU     B#00            ; B ADDRESSES 2903 REGISTERS
BIS7051:EQU     B#01            ; B ADDRESSES FIRST 29705 ADDTION
BIS7052:EQU     B#10            ; B ADDRESSES SECOND 29705 ADDITION
BBUS:   EQU     B#11            ; B FROM BUS
;
;
; * * * * * * * * * * * * * * * * * * * * * * * *
;
; THREE ADDRESS OPERATION - THIRD ADDRESS FIELD
; ADDED C ADDRESS BITS (C5-C4)
; * * * * * * * * * * * * * * * * * * * * * * * *
;
;
CIN2903:EQU     B#00            ; C ADDRESSES 2903 REGISTERS
CIS7051:EQU     B#01            ; C ADDRESSES FIRST 29705 ADDITION
CIS7052:EQU     B#10            ; C ADDRESSES SECOND 29705 ADDITION
CBUS:   EQU     B#11            ; C TO B BUS OUT
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; IO SOURCE SELECT FIELD REPLACES EA-IO-OEB THREE-BIT FIELD
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
QREGSEL:EQU     B#1                 ; SOURCE IS Q REGISTER
NONQREG:EQU     B#0                 ; SOURCE IS RAMB OR B.BUS
;
;
; * * * * * * * * * * * * * * * * * * * * * * *
;
; MISCELLANEOUS FIELDS
;
; REGISTERS
; * * * * * * * * * * * * * * * * * * * * * * *
;
R0:             EQU     H#0
R1:             EQU     H#1
R2:             EQU     H#2
R3:             EQU     H#3
R4:             EQU     H#4
R5:             EQU     H#5
R6:             EQU     H#6
R7:             EQU     H#7
R8:             EQU     H#8
R9:             EQU     H#9
R10:            EQU     H#A
R11:            EQU     H#B
R12:            EQU     H#C
R13:            EQU     H#D
R14:            EQU     H#E
R15:            EQU     H#F
;
; * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; CARRY BIT (2 BITS FOR NOW)
;
; * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
CARRY:          EQU     B#01
NOCARRY:        EQU     B#00            ;IMAGINATIVE!
IC:             EQU     B#10            ; Cin is Cout
Z:              EQU     B#11            ; Z is Cin
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Am2910 MICROPROGRAM CONTROLLER INSTRUCTION SET
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
JZ:             EQU     H#0             ; RESET STACK, MICROPC, ADDRESS
CJS:            EQU     H#1             ; COND JUMP SUBROUTINE, PUSH STACK
JMAP:           EQU     H#2             ; UNCOND JUMP TO MEMORY MAP (Di)
CJP:            EQU     H#3             ; COND JUMP PIPELINE
PUSH:           EQU     H#4             ; PUSH STACK, LOAD REG MAYBE, CONT
JSRP:           EQU     H#5             ; JUMP SUB FROM REG (F) OR PIPE(T)
CJV:            EQU     H#6             ; COND JUMP TO VECTOR INTER (Di)
JRP:            EQU     H#7             ; JUMP TO REG (F) OR PIPE (T)
RFCT:           EQU     H#8             ; DO LOOP REPEAT UNTIL CTR=0 - STACK
RPCT:           EQU     H#9             ; DO LOOP UNTIL CTR=0 - PIPE
CRTN:           EQU     H#A             ; COND RETURN, POP STACK (T)
CJPP:           EQU     H#B             ; COND JUMP PIPELINE, POP STACK
LDCT:           EQU     H#C             ; LOAD REGISTER, CONTINUE
LOOP:           EQU     H#D             ; DO LOOP UNTIL TEST=T - STACK
CONT:           EQU     H#E             ; CONTINUE
TWB:            EQU     H#F             ; THREE WAY (DEAD MAN TIMER!)
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Am2904 SHIFT INSTRUCTIONS (I9-I8-I7-I6 AND SE)
;    I10 IS TIED TO I8 OF Am2903/29203
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
; DOWN SHIFTING
;
SDZRZQ:          EQU     H#0           ; Z->RN; Z->QN
SDOROQ:          EQU     H#1           ; 1->RN; 1->QN
SLN.RECOVER:     EQU     H#2           ; 0->RN; R0->Mc; MN->QN
DDOR:            EQU     H#3           ; 1->RN; R0->QN
DDMCR:           EQU     H#4           ; Mc->RN; R0->QN
DLN.RECOVER:     EQU     H#5           ; MN->RN; R0->QN
DDZR:            EQU     H#6           ; 0->RN; R0->QN
DDZRQMC:         EQU     H#7           ; 0->RN; R0->QN; Q0->Mc
SDROTMC:         EQU     H#8           ; ROT.R; R0->Mc; ROT.Q
SDROTC:          EQU     H#9           ; ROT.R WITH Mc; ROT.Q
SDROT:           EQU     H#A           ; ROT.R; ROT.Q
SDIC:            EQU     H#B           ; Ic->RN; R0->QN
DDROTC:          EQU     H#C           ; Mc->RN; R0->QN; Q0->Mc
DDROTMC:         EQU     H#D           ; Q0->RN; R0->QN; Q0->Mc
DDINIOVR:        EQU     H#E           ; IN EXOR IOVR -> RN; R0->QN
DDROT:           EQU     H#F           ; DOUBLE PRECISION ROTATE DOWN
;
;
; UP SHIFTING
;
SURZQZ:          EQU     H#2           ; R0<-0; Q0<-0
;
;
; SHIFT ENABLES
;
SE.EN:           EQU     B#0           ; ENABLE SHIFTING
SE.DIS:          EQU     B#1           ; DISABLE SHIFTING
;
;
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Am2904 STATUS REGISTER INSTRUCTION CODES
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; MACHINE STATUS REGISTER INSTRUCTION CODES
;    I5-I4-I3-I2-I1-I0 AND EZ-EC-EN-EOVR-CEM ENABLES
; MICRO STATUS REGISTER INSTRUCTION CODES
;    I5-I4-I3-I2-I1-I0 AND CEu ENABLE
;
; THE FOLLOWING TAKES THESE ALL TOGETHER - YOU MAY WISH TO DO THIS ANOTHER WA
;
; ORDER: 543 210 ZCNOVR CEM CEu
;         Q#  Q#   H#     B#  B#
;
ONELEVEL:       EQU     12Q#0000        ; Y -> MSR; MSR -> USR
SET.MSR:        EQU     12Q#0101        ; SET MACRO STATUS ONLY
SET.USR:        EQU     12Q#0176        ; SET MICRO STATUS ONLY
SWAP.REG:       EQU     12Q#0200        ; MSR <--> USR
;
LOAD.MSR:       EQU     12Q#2001        ; ALU STATUS -> MSR
; THE ABOVE IS ON OF SEVERAL CODES - YOU DON'T NEED THEM ALL!
;
LOAD.USR:       EQU     12Q#2076        ; ALU STATUS -> USR
; DITTO!
;
LOAD.BOTH:      EQU     12Q#2000        ; ALU -> MSR, USR
; AGAIN DITTO!
;
LDINVRTM:       EQU     12Q#3001        ; ALU -> MSR; Ic INVERTED
LDINVRTU:       EQU     12Q#3076        ; ALU -> USR; Ic INVERTED
LOAD.INVERT:    EQU     12Q#3000        ; ALU -> MSR, USR; Ic INVERTED
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Am2904 CONDITION CODE OUTPUT INSTRUCTION CODES
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; caution!  I5-I4-I3-I2-I1-I0 ARE ALSO USED FOR TESTING!!!!
; ENABLE TESTING VIA OEct ENABLE
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
TESTMZ:         EQU     12Q#4477        ; NO STATUS OPERATION
TESTMOVR:       EQU     12Q#4677        ; NO STATUS OPERATION
TESTMC:         EQU     12Q#5277        ; NO STATUS OPERATION
TESTMN:         EQU     12Q#5677        ;
TEST.IOVR:      EQU     12Q#6677        ;
TEST.IC:        EQU     12Q#7277        ;
;
;
; TEST ENABLE
;
OECTEN:         EQU     B#0
OECTDIS:        EQU     B#1
;
;
; OUTPUT ENABLE
;
OEYEN:          EQU     B#0
OEYDIS:         EQU     B#1
;
; INSTRUCTION ENABLE
;
IEN:            EQU     B#0
IENDIS:         EQU     B#1
;
;
; CONDITIONAL CODE MULTIPLEXER (DATA MONITOR)
;
NOACK:          EQU     Q#0
COUT:           EQU     Q#1
PASS:           EQU     Q#7
;
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; TWO ADDRESS OPERATION - NO EXPANDED MEMORY
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;
AM2903:  DEF 19X, 3VQ#0, 4VH#F, 4VH#C, 2VB#00, 4VH#0, 4VH#0, 1VB#0, 1VB#0, 22
;  DEFAULTS        RAMAB  OR     YBUS   NOCin   R0     R0     IEN    OEY.EN
;
;
AM2910:   DEF   4VH#E, 3VX, 12V$X, 45X
; DEFAULTS      CONT    #     #
;
AM2904:   DEF   42X,     12VQ#2001, 1VB#1,   1VB#0, 4VX,  1VB#1, 3X
; DEFAULTS               LOAD.MSR   OECTDIS  OEYEN   X     SE.DIS
;
SHIFT:    DEF   56X,  4VX, 1B#0, 3X
;                     SHIFT SE.EN
TEST:     DEF   42X, 12VQ#7777, 1VB#0, 9X
;                    DISABLED   OECTEN
STATUS:   DEF   42X, 12VQ#2001, B#1, 1VB#0, 4X, B#1, 3X
;                    LOAD.MSR  NO CT OEYEN      SE.DIS


;
; ADDED STATEMENTS FOR DATA MONITOR PROBLEM
;
NOP2903:  DEF   19X, Q#0, H#F, H#C, B#00, H#0, H#0, B#0, B#1, 22X
;
CTRL:     DEF   61X, 1VB#0, 1VB#0, 1VB#0
;                    DATAin DATAout MEMORY MAP SELECT
;                    CTRL   CTRL    FIRST QUADRANT
;
;
;
END
```

```
TITLE TEST FILE FOR AM2903 .DEF FILE

;  -------------------------------------------------------------------------
;                 PARTIAL MICROWORD ONLY !!!!!!!!
;                        DEMO FILE !
;  -------------------------------------------------------------------------
;
;        SAMPLE CODE   Am2903-Am2910
;
; * * * * *
; SAMPLE Am2903 OPERATIONS FROM THE ED2900A CLASS NOTES
;                               THE 2900 FAMILY STUDY GUIDE
;                               THE ED2900B CLASS NOTES
; * * * * *
;------------------------------------------------------------------------------
; 15. DA + DB --> Yi
;------------------------------------------------------------------------------
        AM2910 & AM2903 DADB, ADD, YBUS, NOCARRY
;------------------------------------------------------------------------------
; 16. RA + RB --> RC (ANY THREE REGISTERS)
;------------------------------------------------------------------------------
        AM2910 & AM2903 , ADD, RAM, NOCARRY, R0, R1     ; ADD THIRD REG FIELD
;------------------------------------------------------------------------------
; 18.  INCREMENT R15 AND OUTPUT ITS ORIGINAL VALUE
;------------------------------------------------------------------------------
        AM2910 & AM2903 , INCRR, RAM, CARRY, R15, R15
;------------------------------------------------------------------------------
; 17.  FIRMWARE BYTE SWAP
;------------------------------------------------------------------------------
        AM2910 LDCT,, H#2 & AM2903 , ADD, RAMUPL, IC, R15, R15 & SHIFT SDROT
LA:     AM2910 RPCT,, LA  & AM2903 , ADD, RAMUPL, IC, R15, R15 & SHIFT SDROT
;------------------------------------------------------------------------------
; HARDWARE-ASSISTED BYTE SWAP
;------------------------------------------------------------------------------
        AM2910 & AM2903 DARAMB, INCRR, RAM, NOCARRY, , R15
; OR...
        AM2910 & AM2903 RAMAB,       , RAM,         , , R15, , OEYDIS
;------------------------------------------------------------------------------
; 19. DA --> Q
;------------------------------------------------------------------------------
        AM2910 & AM2903 DARAMB, INCRR, LOADQ, NOCARRY
;------------------------------------------------------------------------------
; 20. OUTPUT R2 AND PERFORM 2*(R2+1) --> R2 IN ONE MICROCYCLE
;------------------------------------------------------------------------------
        AM2910 & AM2903 , INCRR, RAMUPL, CARRY, R2, R2 & SHIFT SURZQZ
; OR...
        AM2910 & AM2903 , INCRR, RAMUPA, CARRY, R2, R2 & SHIFT SURZQZ
;------------------------------------------------------------------------------
; 21. UNSIGNED 16 BIT MULTIPLY (R1*R2)
;------------------------------------------------------------------------------
        AM2910 LDCT ,, H#F & AM2903 , INCRR, LOADQ, NOCARRY, R2
LB:     AM2910 RPCT ,, LB  & AM2903 , SPECL, MULT,  NOCARRY, R1, R0
```

```
;-----------------------------------------------------------------------
; 22. TWO'S COMPLEMENT 16 BIT MULTIPLY (R1*R2)
;-----------------------------------------------------------------------
        AM2910 LDCT ,, H#E & AM2903 , INCRR, LOADQ,   NOCARRY, R2
LC:     AM2910 RPCT ,, LC  & AM2903 , SPECL, TWOMULT, NOCARRY, R1, R0
/&              SHIFT DDOR
        AM2910             & AM2903 , SPECL, TWOLAST, Z,       R1, R0
/&              SHIFT DDOR
;-----------------------------------------------------------------------
; 23. PERFORM A DOUBLE PRECISION DOWN SHIFT USING R2 AND Q
;-----------------------------------------------------------------------
        AM2910 & AM2903 , INCRS, RAMQDL, , , R2 & SHIFT DDZR
;-----------------------------------------------------------------------
; 24. PERFORM 4*R2 --> Q IN ONE MICROCYCLE
;-----------------------------------------------------------------------
        AM2910 & AM2903 , ADD, RAMQUPA, NOCARRY, R2, R2 & SHIFT SURZQZ
        AM2910 & AM2903 , INCRS, LOADQ, NOCARRY,    , R2
; *************** REQUIRES TWO MICROCYCLES! ******************************
;-----------------------------------------------------------------------
; 27. SINGLE LENGTH NORMALIZE OF R6 (16 BIT ALU)
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
; 28. DOUBLE LENGTH NORMALIZE OF R6.R7 (16 BIT ALU)
;-----------------------------------------------------------------------


;
END
```

# LAB FOUR

# DATA MONITOR DEF & SRC FILES

## LAB FOUR - THE DATA MONITOR .DEF AND .SRC FILES

YOUR HOMEWORK LAST NIGHT WAS TO CREATE THE .DEF AND .SRC FILES FOR THE DATA MONITOR 2910-2901 DESIGN PROBLEM (ED2900A)

YOU HAVE UNTIL 10:30 AM TO COMPLETE YOUR FILE CREATION

LABS FIVE AND SIX WILL ASSUME THAT YOU HAVE DONE SO!

# DATA MONITOR

THE DATA MONITOR

- MONITOR2.SRC listing
  ( uses AM2903.DEF )
- MONITOR.P2L AMDASM ASSEMBLY listing
  - .SRC SEQUENCED
  - ENTRY POINTS
  - CONTROL MEMORY PROM CONTENTS
    ( X for Don't Cares )
  - SYMBOLS listing
- MONITOR.OPC listing
- AMMAP ASSEMBLY listing

- .OPC SEQUENCED
- ENTRY POINTS relisted
- MEMORY MAP CONTENTS

## .SRC File

```
TITLE DATA MONITOR - Am2903 VERSION
;
;
;
; ADDED EQUATES FOR MONITOR
; ( USING AM2903.DEF FILE )
;
LOAD:           EQU     B#1
NOLOAD:         EQU     B#0
;
FIVES:          EQU     B#1
ZRANGE:         EQU     B#0
;
; * * * * * * * * * * * * * * *
; CODE TO INITIALIZE REGISTERS
; R0=R1=R2=R3=R4 <-- 0
; * * * * * * * * * * * * * * *
;
START:  AM2910 & AM2903 , EXOR, RAM, , R0, R0
        AM2910 & AM2903 , EXOR, RAM, , R1, R1
        AM2910 & AM2903 , EXOR, RAM, , R2, R2
        AM2910 & AM2903 , EXOR, RAM, , R3, R3
        AM2910 & AM2903 , EXOR, RAM, , R4, R4
;
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; THE ABOVE LINE AND THE BELOW LINE CANNOT BE COMBINED DUE TO THE
; BRANCH ENTRY POINT ( BRANCHING TO NEXT DOES NOT ALWAYS INCLUDE
; RESETTING REGISTER 4)
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;
NEXT:   AM2910 & NOP2903 & CTRL LOAD                   ; LOAD DATAin
;
        AM2910 JMAP & NOP2903                          ; CASE BRANCH
;-----------------------------------
; IF DATAin = 0 THEN INCR R0, etc.
;-----------------------------------
DISZ::  AM2910 & AM2903 DARAMB, INCRR, RAM, CARRY, R0, R0
        AM2910 CJS, COUT,  SUBO  & NOP2903
        AM2910 & AM2903 ,     , RAM, , R1, R1 & CTRL , LOAD
WAITA:  AM2910 CJP, NOACK, WAITA & NOP2903
        AM2910 CJP, PASS,  CASE  & NOP2903    & CTRL ,     , FIVES
;-----------------------------------
; IF DATAin > 0 THEN INCR R1, etc.
;-----------------------------------
DGRZ::  AM2910 & AM2903 DARAMB, INCRR, RAM, CARRY, R1, R1
        AM2910 CJS, COUT,  SUBO  & NOP2903
        AM2910 & AM2903 ,     , RAM, , R0, R0 & CTRL , LOAD
WAITB:  AM2910 CJP, NOACK, WAITB & NOP2903    & CTRL ,     , FIVES
;
;----------------------------------------------------------
; this case statement uses the upper half of the memory map
;----------------------------------------------------------
;
CASE:   AM2910 JMAP & NOP2903                 & CTRL ,     , FIVES
```

```
;
; * * * * * * * * * * * * * * * * * * * * * * * * *
; INCREMENT R2 OR R3 OR R4 BASED ON RANGE OF DATAin
; * * * * * * * * * * * * * * * * * * * * * * * * *
;
L5::    AM2910 CJP, PASS,  NEXT  & AM2903 DARAMB, INCRR, RAM, CARRY, R2, R2
GR5L10::AM2910 CJP, PASS,  NEXT  & AM2903 DARAMB, INCRR, RAM, CARRY, R3, R3
GR10::  AM2910 CJP, PASS,  NEXT  & AM2903 DARAMB, INCRR, RAM, CARRY, R4, R4
;
; ----------------------------------------------------------
; SUBROUTINE SUB0
; ----------------------------------------------------------
;
; output R2, R3, R4 and then reset them all
; wait for an ACK after each send
;
;
;
SUB0:   AM2910 & AM2903 ,     , RAM, , R2, R2 & CTRL , LOAD
WAIT1:  AM2910 CJP, NOACK, WAIT1 & NOP2903
        AM2910 & AM2903 ,     , RAM, , R3, R3 & CTRL , LOAD
WAIT2:  AM2910 CJP, NOACK, WAIT2 & NOP2903
        AM2910 & AM2903 ,     , RAM, , R4, R4 & CTRL , LOAD
WAIT3:  AM2910 CJP, NOACK, WAIT3 & NOP2903
        AM2910            & AM2903 , EXOR, RAM, , R2, R2
        AM2910            & AM2903 , EXOR, RAM, , R3, R3
        AM2910 CRTN, PASS & AM2903 , EXOR, RAM, , R4, R4
;
;
;
;
;
END
```

# PHASE 2 ASSEMBLY

```
DATA MONITOR - AM2903 VERSION

 ENTRY POINTS

  DGRZ      000C
  DISZ      0007
  GR10      0013
  GR5L10    0012
  L5        0011
```

```
     TITLE DATA MONITOR - Am2903 VERSION
     ;
     ;
     ;
     ; ADDED EQUATES FOR MONITOR
     ; ( USING AM2903.DEF FILE )
     ;
0001 LOAD:           EQU     B#1
     NOLOAD:         EQU     B#0
0000 ;
0001 FIVES:          EQU     B#1
     ZRANGE:         EQU     B#0
     ;
     ;
     ; CODE TO INITIALIZE REGISTERS
     ;
0000 ;
0000 START:   AM2910 & AM2903 , EXOR, RAM, , R0, R0
0001          AM2910 & AM2903 , EXOR, RAM, , R1, R1
0002          AM2910 & AM2903 , EXOR, RAM, , R2, R2
0003          AM2910 & AM2903 , EXOR, RAM, , R3, R3
0004          AM2910 & AM2903 , EXOR, RAM, , R4, R4
0005 NEXT:    AM2910 & NOP2903 & CTRL LOAD
0006          AM2910 JMAP & NOP2903
0007 DISZ::   AM2910 & AM2903 DARAMB, INCRR, RAM, CARRY, R0, R0
0008          AM2910 CJS, COUT,  SUB0  & NOP2903
0009          AM2910 & AM2903 ,      , RAM, , R1, R1 & CTRL , LOAD
000A WAITA:   AM2910 CJP, NOACK, WAITA & NOP2903
000B          AM2910 CJP, PASS,  CASE  & NOP2903    & CTRL ,      , FIVES
000C DGRZ::   AM2910 & AM2903 DARAMB, INCRR, RAM, CARRY, R1, R1
000D          AM2910 CJS, COUT,  SUB0  & NOP2903
000E          AM2910 & AM2903 ,      , RAM, , R0, R0 & CTRL , LOAD
000F WAITB:   AM2910 CJP, NOACK, WAITB & NOP2903    & CTRL ,      , FIVES
0010 CASE:    AM2910 JMAP & NOP2903                 & CTRL ,      , FIVES
0011 L5::     AM2910 CJP, PASS,  NEXT  & AM2903 DARAMB, INCRR, RAM, CARRY, R2, R2
0012 GR5L10::AM2910 CJP, PASS,  NEXT  & AM2903 DARAMB, INCRR, RAM, CARRY, R3, R3
0013 GR10::   AM2910 CJP, PASS,  NEXT  & AM2903 DARAMB, INCRR, RAM, CARRY, R4, R4
     ;
     ; SUBROUTINE SUB0
     ;
0014 SUB0:    AM2910 & AM2903 ,      , RAM, , R2, R2 & CTRL , LOAD
0015 WAIT1:   AM2910 CJP, NOACK, WAIT1 & NOP2903
0016          AM2910 & AM2903 ,      , RAM, , R3, R3 & CTRL , LOAD
0017 WAIT2:   AM2910 CJP, NOACK, WAIT2 & NOP2903
0018          AM2910 & AM2903 ,      , RAM, , R4, R4 & CTRL , LOAD
0019 WAIT3:   AM2910 CJP, NOACK, WAIT3 & NOP2903
001A          AM2910 & AM2903 , EXOR, RAM, , R2, R2
001B          AM2910 & AM2903 , EXOR, RAM, , R3, R3
001C          AM2910 & AM2903 , EXOR, RAM, , R4, R4
001D          AM2910 CRTN, PASS        & NOP2903
     ;
     ;
     ;
     ;
     ;
     END
```

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.0
DATA MONITOR - AM2903 VERSION

 0000 1110XXXXXXXXXXXX XXX0001011010000 0000000000XXXXXX XXXXXXXXXXXXXXXX
 0001 1110XXXXXXXXXXXX XXX0001011010000 0001000100XXXXXX XXXXXXXXXXXXXXXX
 0002 1110XXXXXXXXXXXX XXX0001011010000 0010001000XXXXXX XXXXXXXXXXXXXXXX
 0003 1110XXXXXXXXXXXX XXX0001011010000 0011001100XXXXXX XXXXXXXXXXXXXXXX
 0004 1110XXXXXXXXXXXX XXX0001011010000 0100010000XXXXXX XXXXXXXXXXXXXXXX
 0005 1110XXXXXXXXXXXX XXX0001111110000 0000000001XXXXXX XXXXXXXXXXXXX100
 0006 0010XXXXXXXXXXXX XXX0001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 0007 1110XXXXXXXXXXXX XXX1000110010001 0000000000XXXXXX XXXXXXXXXXXXXXXX
 0008 0001001000000010 1000001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 0009 1110XXXXXXXXXXXX XXX0001111010000 0001000100XXXXXX XXXXXXXXXXXXX010
 000A 0011000000000001 0100001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 000B 0011111000000010 0000001111110000 0000000001XXXXXX XXXXXXXXXXXXX001
 000C 1110XXXXXXXXXXXX XXX1000110010001 0001000100XXXXXX XXXXXXXXXXXXXXXX
 000D 0001001000000010 1000001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 000E 1110XXXXXXXXXXXX XXX0001111010000 0000000000XXXXXX XXXXXXXXXXXXX010
 000F 0011000000000001 1110001111110000 0000000001XXXXXX XXXXXXXXXXXXX001
 0010 0010XXXXXXXXXXXX XXX0001111110000 0000000001XXXXXX XXXXXXXXXXXXX001
 0011 0011111000000000 1011000110010001 0010001000XXXXXX XXXXXXXXXXXXXXXX
 0012 0011111000000000 1011000110010001 0011001100XXXXXX XXXXXXXXXXXXXXXX
 0013 0011111000000000 1011000110010001 0100010000XXXXXX XXXXXXXXXXXXXXXX
 0014 1110XXXXXXXXXXXX XXX0001111010000 0010001000XXXXXX XXXXXXXXXXXXX010
 0015 0011000000000010 1010001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 0016 1110XXXXXXXXXXXX XXX0001111010000 0011001100XXXXXX XXXXXXXXXXXXX010
 0017 0011000000000010 1110001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 0018 1110XXXXXXXXXXXX XXX0001111010000 0100010000XXXXXX XXXXXXXXXXXXX010
 0019 0011000000000011 0010001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
 001A 1110XXXXXXXXXXXX XXX0001011010000 0010001000XXXXXX XXXXXXXXXXXXXXXX
 001B 1110XXXXXXXXXXXX XXX0001011010000 0011001100XXXXXX XXXXXXXXXXXXXXXX
 001C 1110XXXXXXXXXXXX XXX0001011010000 0100010000XXXXXX XXXXXXXXXXXXXXXX
 001D 1010111XXXXXXXXX XXX0001111110000 0000000001XXXXXX XXXXXXXXXXXXXXXX
```

```
DATA MONITOR - AM2903 VERSION

 SYMBOLS                  INCRNON   0007          RAM       0004
                          INCRR     0006          RAMAB     0000
  ABUS      0004          INCRS     0004          RAMADB    0001
  ACONST    0003          INCSNON   0005          RAMAQ     0002
  ADD       0003          JMAP      0002          RAMDA     0000
  AINALU    0000          JRP       0007          RAMDL     0001
  AIS7051   0001          JSRP      0005          RAMEXT    000F
  AIS7052   0002          JZ        0000          RAMQ      0007
  AND       000C          L5        0011          RAMQDA    0002
  BBUS      0003          LDCT      000C          RAMQDL    0003
  BINALU    0000          LDINVRTM  0601          RAMQUPA   000A
  BIS7051   0001          LDINVRTU  063E          RAMQUPL   000B
  BIS7052   0002          LOAD      0001          RAMUPA    0008
  CARRY     0001          LOAD.BOT  0400          RAMUPL    0009
  CASE      0010          LOAD.INV  0600          RFCT      0008
  CBUS      0003          LOAD.MSR  0401          RPCT      0009
  CIN2903   0000          LOAD.USR  043E          SDIC      000B
  CIS7051   0001          LOADQ     0006          SDOROQ    0001
  CIS7052   0002          LOOP      000D          SDROT     000A
  CJP       0003          LOW       0008          SDROTC    0009
  CJPP      000B          MULT      0000          SDROTMC   0008
  CJS       0001          NAND      000E          SDZRZQ    0000
  CJV       0006          NEXT      0005          SE.DIS    0001
  CONT      000E          NOACK     0000          SE.EN     0000
  COUT      0001          NOCARRY   0000          SET.MSR   0041
  CRTN      000A          NOLOAD    0000          SET.USR   007E
  DADB      0005          NONQREG   0000          SGNTWO    0005
  DAQ       0006          NOR       000D          SIGNEXT   000E
  DARAMB    0004          NOTRS     0009          SLN       0008
  DDINIOVR  000E          OECTDIS   0001          SLN.RECO  0002
  DDMCR     0004          OECTEN    0000          SPECL     0000
  DDOR      0003          OEYDIS    0001          START     0000
  DDROT     000F          OEYEN     0000          SUB0      0014
  DDROTC    000C          ONELEVEL  0000          SUBR      0001
  DDROTMC   000D          OR        000F          SUBS      0002
  DDZR      0006          PASS      0007          SURZQZ    0002
  DDZRQMC   0007          PUSH      0004          SWAP.REG  0080
  DGRZ      000C          QD        0005          TEST.IC   0EBF
  DISZ      0007          QREGSEL   0001          TEST.IOV  0DBF
  DIVFRST   000A          QUP       000D          TESTMC    0ABF
  DIVIDE    000C          R0        0000          TESTMN    0BBF
  DIVLAST   000E          R1        0001          TESTMOVR  09BF
  DLN       000A          R10       000A          TESTMZ    093F
  DLN.RECO  0005          R11       000B          TWB       000F
  EXNOR     000A          R12       000C          TWOLAST   0006
  EXOR      000B          R13       000D          TWOMULT   0002
  FIVES     0001          R14       000E          WAIT1     0015
  GR10      0013          R15       000F          WAIT2     0017
  GR5L10    0012          R2        0002          WAIT3     0019
  HIGH      0000          R3        0003          WAITA     000A
  IC        0002          R4        0004          WAITB     000F
  IEN       0000          R5        0005          YBUS      000C
  IENDIS    0001          R6        0006          Z         0003
  INCRMNT   0004          R7        0007          ZRANGE    0000
                          R8        0008
                          R9        0009        TOTAL PHASE 2 ERRORS =   0
```

# .OPC FILE

```
TITLE DATA MONITOR MAPPING PROM
;
;
;
WIDTH 8
;
;
           DISZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           DGRZ
           L5
           L5
           L5
           L5
           L5
           L5
           GR5L10
           GR5L10
           GR5L10
           GR5L10
           GR5L10
           GR10
           GR10
           GR10
           GR10
           GR10
;
END
```

# AMMAP ASSEMBLY

```
AMDOS/29 AMMAP ASSEMBLER, V1.1
DATA MONITOR MAPPING PROM

      ;
      ;
      ;
      WIDTH 8
      ;
      ;
0000          DISZ
0001          DGRZ
0002          DGRZ
0003          DGRZ
0004          DGRZ
0005          DGRZ
0006          DGRZ
0007          DGRZ
0008          DGRZ
0009          DGRZ
000A          DGRZ
000B          DGRZ
000C          DGRZ
000D          DGRZ
000E          DGRZ
000F          DGRZ
0010          L5
0011          L5
0012          L5
0013          L5
0014          L5
0015          L5
0016          GR5L10
0017          GR5L10
0018          GR5L10
0019          GR5L10
001A          GR5L10
001B          GR10
001C          GR10
001D          GR10
001E          GR10
001F          GR10
      ;
      END
```

```
AMDOS/29 AMMAP ASSEMBLER, V1.1
DATA MONITOR MAPPING PROM

 ENTRY POINT SYMBOLS

  DGRZ      000C
  DISZ      0007
  GR10      0013
  GR5L10    0012
  L5        0011

 TOTAL ASSEMBLY ERRORS =   0
```

```
AMDOS/29 AMMAP ASSEMBLER, V1.1
DATA MONITOR MAPPING PROM

 0000 00000111 (0007)
 0001 00001100 (000C)
 0002 00001100 (000C)
 0003 00001100 (000C)
 0004 00001100 (000C)
 0005 00001100 (000C)
 0006 00001100 (000C)
 0007 00001100 (000C)
 0008 00001100 (000C)
 0009 00001100 (000C)
 000A 00001100 (000C)
 000B 00001100 (000C)
 000C 00001100 (000C)
 000D 00001100 (000C)
 000E 00001100 (000C)
 000F 00001100 (000C)
 0010 00010001 (0011)
 0011 00010001 (0011)
 0012 00010001 (0011)
 0013 00010001 (0011)
 0014 00010001 (0011)
 0015 00010001 (0011)
 0016 00010010 (0012)
 0017 00010010 (0012)
 0018 00010010 (0012)
 0019 00010010 (0012)
 001A 00010010 (0012)
 001B 00010011 (0013)
 001C 00010011 (0013)
 001D 00010011 (0013)
 001E 00010011 (0013)
 001F 00010011 (0013)
```

# LAB FIVE:

# LBPM

# SBPM

# RBPM

# VBPM

# & DDT29

## LAB FIVE - WCS

- POWER UP THE SYSTEM

- BE CERTAIN THAT THE CLOCK IS OFF

- SET DIP SWITCH ON WRITABLE CONTROL STORE CARD SO THAT THE FIRST THREE FROM THE TOP ARE PUSHED TOWARDS THE REAR OF THE CARD OR CHASSIS

    - DO NOT SHORT OUT THE CARDS
    - USE A FINGER OR OTHER NON-ABRASIVE DEVICE TO MOVE THEM

## START WITH THE DATA MONITOR SRC FILE OUTPUT

- LOAD WCS (LOWER HALF OF MICROWORD)

```
LBPM B:MONITOR WCS CL
```

- SIGN ON TO DDT29 VIA

```
A>DDT29
```

- SYSTEM PROMPT IS NOW A "."

- TYPE

```
D
```

- YOU SHOULD BE LOOKING AT THE FIRST 16 MICROWORDS IN THE WCS

- TYPE

```
D 0 1DPc
```

- YOU SHOULD SEE AND PRINT OUT THE FIRST 6 MICROWORDS
- TURN PRINTER OFF

```
Pc
```

- TYPE

```
S 5 C BBBB

D 5 5
```

- YOU HAVE JUST SUBSTITUTED BBBB INTO BYTES C AND D OF MICROWORD AT ADDRESS 5

- TYPE

```
E
```

- THIS EXITS DDT29

- TYPE

```
SBPM B:FILE1 WC FR 0 TO 1D
```

- THIS WILL SAVE THE ALTERED FILE AS FILE1.SVW

- TYPE

```
RBPM B:FILE1.SVW
```

- RELOADS THE WCS

- TYPE

```
VBPM B:MONITOR WC FR 0 TO 1D CL
```

- COMPARES PATCHED FILES WITH THE WCS

  HELPS YOU REMEMBER PATCHES, ETC

- TYPE

```
LBPM B:MONITOR WCS WA 1E CL DC 1

DDT29

D 0 3BPc
```

## Microprogramming Support Software Programs

| Program | Format |
|---|---|
| DDT29 (Dynamic Debugging Tool-2900) | $\text{Dw} \left\lvert \begin{matrix} X.Y \\ X\ Y \end{matrix} \right\rvert$<br>Display WCS from address X for Y words.<br>Display WCS from address X through address Y. |
| | $\text{DM} \left\lvert \begin{matrix} X.Y \\ X\ Y \end{matrix} \right\rvert$<br>Display mapping memory from address X for Y words.<br>Display mapping memory from address X through address Y. |
| | STatus<br>Display Instrumentation Card status register contents. |
| | $\text{Sw} \left\lvert \text{XYD} \right\rvert$<br>Store Hexadecimal data in WCS beginning at address X, byte Y. |
| | $\text{SM} \left\lvert \text{X D} \right\rvert$<br>Store hexadecimal data in mapping memory beginning at address X. |
| | Halt<br>Stops Microprogrammed System clock. |
| | MS n<br>Micro-step Microprogrammed System clock n (1-65535) steps (1 step = 1 microcycle). (No count = 1.) |
| | SS n<br>Single-step Microprogrammed System clock n (1-65535) phases of a multiphase microcycle. |
| | Run<br>Runs Microprogrammed System clock. |
| | CTL XX<br>Store hexadecimal value 00-FF in Instrumentation Card control register to set mask bits. |
| | Jamccu<br>Jam value in Instrumentation Card address register onto microprogram memory address bus. |
| | IR X<br>Store hexadecimal value X into Instrumentation Card address register. |
| | Z N<br>Sleep N milliseconds. Default value for N is 1. |
| | M N ddt29 subcommand, ddt29 subcommand, . . . . . .ddt29 subcommand<br>Execute N times the user-specified string of DDT 29 subcommands. Default value for N infinity. Execution of the M subcommand can be terminated at any time by pressing the CRT Console DEL key. |
| | DLA<br>Display address of the last microinstruction executed. |
| | DMB<br>Display 20 monitor bits of user selected and wired test point or other data at CRT Console. |
| | Exit<br>Leave DDT29 and enter AMDOS 29. |

### Microprogramming Support Software Programs

<table>
<tr><th>Program</th><th>Format</th></tr>
<tr><td>LBPM (Load Bipolar Memory WCS or CCU Mapping Memory)<br><br>VBPM (Verify Bipolar Memory Contents Previously Loaded Into WCS or CCU Mapping Memory)</td><td>
<table>
<tr><td>LBPM Filename</td><td>WCs<br>MAp</td><td>FRom X</td><td>TO X<br>FOr X</td><td>WA X</td><td>LSb d</td><td>NOClear<br>CLear</td><td>LOwer<br>UPper<br>UL</td><td>VErify<br>NOVerify</td><td>DC b</td><td>SBft</td></tr>
</table>
<table>
<tr><td>VBPM Filename</td><td>WCs<br>MAp</td><td>FRom X</td><td>TO X<br>FOr X</td><td>WA X</td><td>LSb d</td><td>NOClear<br>CLear</td><td>LOwer<br>UPper<br>UL</td><td>DC b</td></tr>
</table>
Where:<br>
Filename is name of diskette file to be loaded into WCS or CCU mapping memory.<br>
FRom designates starting PC within diskette file. (Default is starting file PC.)<br>
TO designates ending PC within diskette file. (Default is entire file.)<br>
FOr designates X number of microwords to load. (Default is entire file.)<br>
WA (Writable Address) designates starting address X within WCS or CCU mapping memory.<br>
LSb designates least significant bit number d (0-127) of microword. (Default is 0, right justified.)<br>
NOClear designates don't clear WCS before loading. (Default is NOClear.)<br>
CLear designates clear each WCS word before loading. (Default is NOClear.)<br>
UL designates all 128 bits of WCS are accessed. (Default is LOwer.)<br>
LOwer designates that only bits 0 to 63 of WCS are accessed. (Default is LOwer.)<br>
UPper designates that only bits 64-127 of WCS are accessed. (Default is LOwer.)<br>
NOVerify designates that verify phase is not run. (Default is VErify.)<br>
VErify designates that verify phase is run automatically.<br>
SBft designates given operands are placed into bipolar format table for subsequent default value use. (See BFt portion of SET command.)<br>
DCb designates value "don't cares" are set to. (Default is 0.)
</td></tr>
<tr><td>SBPM (Save Bipolar Memory)<br><br>RBPM (Restore Bipolar Memory)</td><td>
<table>
<tr><td>SBPM Filename</td><td>WCs<br>MAp</td><td>FRom X</td><td>TO X<br>FOr X</td></tr>
</table>
<table>
<tr><td>RBPM Filename.</td><td>SVW<br>SVM</td></tr>
</table>
Where:<br>
Filename is name of diskette file created to receive saved WCS microcode or CCU mapping memory contents. Complete filename for saved WCS contents is "filename.SVW". Complete filename for CCU mapping memory contents is "filename.SVM" SBPM prompts with a ". . . . erase (Y or N)" if "filename.SVW" or "filename.SVM" already exists. FRom, TO, and FOr are valid for WCs only.<br>
FRom designates starting PC within WCS (Default is 0.)<br>
TO designates ending PC within WCS (Default is FRom and results in one microword saved.)<br>
FOr designates number of microwords to save (Default is 1.)
</td></tr>
</table>

# LAB 6

## AMMAP

## AMSCRM

## AMPROM

## LAB SIX - AMMAP - AMSCRM - AMPROM

### AMMAP

SIMPLE

MONITOR

- POWER UP SYSTEMS
- CHECK DATA DISK FOR SIMPLE.DEF, SIMPLE.SRC

  Am2903.DEF, MONITOR.SRC

  ```
  A>DIR B:SIMPLE.*
  ```

  A>DIR B:MONITOR.*

  A>DIR B:Am2903.*
- ASSEMBLE VIA AMDASM (PHASE 1 AND PHASE 2)

  OR LOCATE B:SIMPLE.MAP

  B:MONITOR.MAP
- CREATE (OR LOCATE) SIMPLE.OPC

  MONITOR.OPC

  (CREATE USING THE EDITOR)
- CALL UP AMMAP

  CREATE MAP FOR THE SIMPLE COMPUTER

  ```
  AMMAP B:SIMPLE MAP = B:SIMPLE
  ```

  AMMAP B:MONITOR MAP = B:MONITOR

.OPC file        .MAP file

- AMMAP CREATES B:SIMPLE.OBM (PROMS, WCS) AND B:SIMPLE.P4L
- PRINT OUT B:SIMPLE.P4L

  B:MONITOR.P4L

## AMSCRM

- ASSEMBLE SIMPLE.DEF, SIMPLE.SRC IF YOU HAVE NOT ALREADY DONE SO (AND NOT ERASED RESULTING FILES)

- USE AMSCRM AND MOVE THE BRANCH ADDRESS FIELD TO THE FAR RIGHT OF THE MICROWORD (SWAP IT WITH F BUS CONTROL)

```
A>AMSCRM OLD=B:SIMPLE.OBJ NEW=B:SIMPLE.XOB
```

## AMPROM

- CALL UP AMPROM FOR MONITOR OR SIMPLE

```
A>AMPROM O B:SIMPLE     <-- B:SIMPLE.OBJ ASSUMED
```

- INPUT THE FOLLOWING
  - DON'T CARES SET TO 0
  - WIDTH OF PROMS USED IS YOUR CHOICE (TRY 8)
  - DEPTH OF PROMS USED IS YOUR CHOICE (TRY 16)
  - PRINT OUT BRANCH ADDRESS PROM(s)
  - PRINT OUT ALL PROMS - FIND THE BRANCH ADDRESS PROMS
- DESIGN YOUR MEMORY SO THAT THE BRANCH ADDRESS FIELD IS IN ITS OWN PROMS
- PRINT OUT B:SIMPLE.P3L

- RENAME B:SIMPLE.XOB TO B:SIMPLE.OBJ (CHANGE THE NAME OF B:SIMPLE.OBJ FIRST OR ERASE IT)
- CALL UP AMPROM AS BEFORE
- USE SAME RESPONSES AS BEFORE TO AMPROM'S QUESTIONS
- PRINT OUT NEW B:SIMPLE.P3L

**WHENEVER YOU USE AS EXISTING FILE RATHER THAN CREATE ONE OF YOUR OWN - CAUTION! SOMEONE ELSE MAY HAVE CHANGED NAMES ON THESE FILES**

# SIMPLE COMPUTER .DEF and .SRC

THE VERY SIMPLE COMPUTER

- SIMPLE.DEF listing
- SIMPLE.SRC listing
- SIMPLE.P2L AMDASM ASSEMBLY listing
  - SRC SEQUENCED
  - CONTROL MEMORY PRINTOUT with ENTRY POINTS
  - SYMBOLS list
- SIMPLE.OPC listing
- AMMAP ASSEMBLY LISTING
  - .OPC SEQUENCED
  - MEMORY MAP CONTENTS
  - ENTRY POINT SYMBOLS (relisted)
- AMPROM ASSEMBLY LISTING
  - PROM MAP (from interactive input)
  - PROM CONTENTS

**.DEF**

```
TITLE  THE VERY SIMPLE COMPUTER (ED2900A)
;
WORD 24
;
; THIS COMPUTER IS NOT BASED ON ANY PARTICULAR PART!
; IT IS SIMPLY AN EXAMPLE OF .DEF AND .SRC FILE CONSTRUCTION
; AND MEMORY MAP USAGE (JMAP Am2910 INSTRUCTION)
;
;
; SEQUENCE CONTROL (VERY LIMITED!)
;
CONT:           EQU     B#00
JMP:            EQU     B#01            ; UNCONDITIONAL (GOTO)
JIFO:           EQU     B#10            ; CONDITIONAL (IF-THEN GOTO)
JOPC:           EQU     B#11            ; JUMP ON OP CODE (JMAP)
;
; MEMORY CONTROL
;
READ:           EQU     B#00
WRITE:          EQU     B#01
DIS:            EQU     B#10            ; NO MEMORY OPERATION (B#11 ALSO VALID
;
; ALU CONTROL
;
ADD:            EQU     Q#0
SUB:            EQU     Q#1
AND:            EQU     Q#2
OR:             EQU     Q#3
EXOR:           EQU     Q#4
ONE:            EQU     Q#5             ; D BUS + 1 TO F
RTF:            EQU     Q#6             ; PASS R BUS
DTF:            EQU     Q#7             ; PASS D BUS
;
; R BUS SOURCE CONTROL
;
DIR:            EQU     B#0             ; DATA-IN TO R BUS
MDOR:           EQU     B#1             ; MEMORY DATA OUT TO R BUS
;
; R BUS DESTINATION CONTROL
;
RIR:            EQU     B#0             ; R BUS TO INSTRUCTION REGISTER
NRIR:           EQU     B#1             ; NOTHING TO INSTR. REG.
;
; D BUS CONTROL
;
ACCD:           EQU     B#0             ; ACCUMULATOR TO D BUS
PCD:            EQU     B#1             ; PC REGISTER TO D BUS
;
; F BUS CONTROL
;
FACC:           EQU     B#00            ; F BUS TO ACC
FPC:            EQU     B#01            ; F BUS TO PC REGISTER
FMAR:           EQU     B#10            ; F BUS TO MAR REGISTER
NFD:            EQU     B#11            ; NO F BUS DESTINATION
;
;
CODE:           DEF     2VX, 12V$X, 2VX, 3VX, 1VX, 1VX, 1VX, 2VX
;      ---> NO DEFAULTS!! <----
;
END
```

```
TITLE SAMPLE .SRC FILE FOR THE SIMPLE COMPUTER (ED2900A)
;
;
;
;
START:  CODE CONT,      ,  DIS,  DTF,      , NRIR,  PCD,  FMAR    ; PC -> MAR
        CODE CONT,      , READ,  RTF, MDOR,  RIR,     ,  FMAR    ; FETCH
        CODE JOPC,      ,  DIS,     ,      , NRIR,     ,  NFD     ; DECODE STEP
LDA::   CODE JMP, FETCH, READ,  RTF, MDOR, NRIR,     ,  FACC    ; LOAD ACC
STO::   CODE JMP, FETCH, WRITE,     ,      , NRIR, ACCD,  NFD     ; STORE ACC
LADD::  CODE JMP, FETCH, READ,  ADD, MDOR, NRIR, ACCD,  FACC ; ACC + MEM
LSUB::  CODE JMP, FETCH, READ,  SUB, MDOR, NRIR, ACCD,  FACC    ; ACC - MEM
LOR::   CODE JMP, FETCH, READ,   OR, MDOR, NRIR, ACCD,  FACC    ; ACC OR MEM
LAND::  CODE JMP, FETCH, READ,  AND, MDOR, NRIR, ACCD,  FACC    ; ACC AND MEM
XOR::   CODE JMP, FETCH, READ, EXOR, MDOR, NRIR, ACCD,  FACC    ; ACC EXOR MEM
INA::   CODE JMP, FETCH,  DIS,  RTF,  DIR, NRIR,     ,  FACC    ; DATA TO ACC
OUT::   CODE JMP, FETCH,  DIS,     ,      , NRIR, ACCD,  NFD     ; ACC TO DATA-OUT
GOTO::  CODE CONT,      ,  DIS,  DTF,      , NRIR,  PCD,  FMAR ; GO TO addr
        CODE JMP, START, READ,  RTF, MDOR, NRIR,     ,  FPC     ; REFETCH INSTR
IF::    CODE JIFO, GOTO,  DIS,     ,      , NRIR,     ,  NFD     ; IF ACC = 0 GO TO addr
FETCH:  CODE JMP, START,  DIS,  ONE,      , NRIR,  PCD,  FPC     ; PC <- PC + 1
;
;
END
```

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.0
SAMPLE .SRC FILE FOR THE SIMPLE COMPUTER (ED2900A)

      TITLE SAMPLE .SRC FILE FOR THE SIMPLE COMPUTER (ED2900A)
      ;
      ;
      ;
        0000 START:  CODE CONT,     ,  DIS,  DTF,     , NRIR,  PCD,  FMAR    ; PC -> MAR
 0001         CODE CONT,     , READ,  RTF, MDOR,  RIR,     ,  FMAR    ; FETCH
 0002         CODE JOPC,     ,  DIS,     ,     , NRIR,     ,  NFD     ; DECODE STEP
 0003 LDA::   CODE JMP, FETCH, READ,  RTF, MDOR, NRIR,     ,  FACC    ; LOAD ACC
 0004 STO::   CODE JMP, FETCH, WRITE,     ,     , NRIR, ACCD,  NFD     ; STORE ACC
 0005 LADD::  CODE JMP, FETCH, READ,  ADD, MDOR, NRIR, ACCD,  FACC  ; ACC + MEM
 0006 LSUB::  CODE JMP, FETCH, READ,  SUB, MDOR, NRIR, ACCD,  FACC    ; ACC - MEM
 0007 LOR::   CODE JMP, FETCH, READ,   OR, MDOR, NRIR, ACCD,  FACC    ; ACC OR MEM
 0008 LAND::  CODE JMP, FETCH, READ,  AND, MDOR, NRIR, ACCD,  FACC    ; ACC AND MEM
 0009 XOR::   CODE JMP, FETCH, READ, EXOR, MDOR, NRIR, ACCD,  FACC    ; ACC EXOR MEM
 000A INA::   CODE JMP, FETCH,  DIS,  RTF,  DIR, NRIR,     ,  FACC    ; DATA TO ACC
 000B OUT::   CODE JMP, FETCH,  DIS,     ,     , NRIR, ACCD,  NFD     ; ACC TO DATA-OUT
 000C GOTO::  CODE CONT,     ,  DIS,  DTF,     , NRIR,  PCD,  FMAR  ; GO TO addr
 000D         CODE JMP, START, READ,  RTF, MDOR, NRIR,     ,  FPC     ; REFETCH INSTR
 000E IF::    CODE JIFO, GOTO,  DIS,     ,     , NRIR,     ,  NFD     ; IF ACC = 0 GO TO addr
 000F FETCH:  CODE JMP, START,  DIS,  ONE,     , NRIR,  PCD,  FPC     ; PC <- PC + 1
      ;
      ;
      END
```

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.0
SAMPLE .SRC FILE FOR THE SIMPLE COMPUTER (ED2900A)

 0000 00XXXXXXXXXXXX10 111X1110
 0001 00XXXXXXXXXXXX00 11010X10
 0002 11XXXXXXXXXXXX10 XXXX1X11
 0003 0100000000111100 11011X00
 0004 0100000000111101 XXXX1011
 0005 0100000000111100 00011000
 0006 0100000000111100 00111000
 0007 0100000000111100 01111000
 0008 0100000000111100 01011000
 0009 0100000000111100 10011000
 000A 0100000000111110 11001X00
 000B 0100000000111110 XXXX1011
 000C 00XXXXXXXXXXXX10 111X1110
 000D 0100000000000000 11011X01
 000E 1000000000110010 XXXX1X11
 000F 0100000000000010 101X1101


SAMPLE .SRC FILE FOR THE SIMPLE COMPUTER (ED2900A)

 ENTRY POINTS

  GOTO      000C
  IF        000E
  INA       000A
  LADD      0005
  LAND      0008
  LDA       0003
  LOR       0007
  LSUB      0006
  OUT       000B
  STO       0004
  XOR       0009
```

## .OPC

```
TITLE MEMORY MAP PROGRAM FOR THE "VERY SIMPLE COMPUTER"
;
;
WIDTH 8
;
; ASSUME A 32 x 8 PROM
;
BASE 16
;
  LDA
  STO
  LADD
  LSUB
  LOR
  LAND
  XOR
  INA
  OUT
  GOTO
;
END
```

## AMMAP

```
AMDOS/29 AMMAP ASSEMBLER, V1.1
MEMORY MAP PROGRAM FOR THE "VERY SIMPLE COMPUTER"

      ;
      ;
      WIDTH 8
      ;
      ; ASSUME A 32 x 8 PROM
      ;
      BASE 16
      ;
 0000   LDA
 0001   STO
 0002   LADD
 0003   LSUB
 0004   LOR
 0005   LAND
 0006   XOR
 0007   INA
 0008   OUT
 0009   GOTO
      ;
      END
```

```
MEMORY MAP PROGRAM FOR THE "VERY SIMPLE COMPUTER"

 0000 00000011 (0003)
 0001 00000100 (0004)
 0002 00000101 (0005)
 0003 00000110 (0006)
 0004 00000111 (0007)
 0005 00001000 (0008)
 0006 00001001 (0009)
 0007 00001010 (000A)
 0008 00001011 (000B)
 0009 00001100 (000C)
```

```
AMDOS/29 AMMAP ASSEMBLER, V1.1
MEMORY MAP PROGRAM FOR THE "VERY SIMPLE COMPUTER"

 ENTRY POINT SYMBOLS

  GOTO      000C
  IF        000E
  INA       000A
  LADD      0005
  LAND      0008
  LDA       0003
  LOR       0007
  LSUB      0006
  OUT       000B
  STO       0004
  XOR       0009

 TOTAL ASSEMBLY ERRORS =    0
```

```
AMD AMPROM UTILITY

SAMPLE .SRC FILE FOR THE SIMPLE COMPUTER (ED2900A)

PROM MAP

     PC     C1    C2    C3

R1  0000     1     2     3
R2  0008     4     5     6


PROM CONTENTS


 PC   ADD P   1       P   2       P   3

0000 000 00000000 00000010 11101110
0001 001 00000000 00000000 11010010
0002 002 11000000 00000010 00001011
0003 003 01000000 00111100 11011000
0004 004 01000000 00111101 00001011
0005 005 01000000 00111100 00011000
0006 006 01000000 00111100 00111000
0007 007 01000000 00111100 01111000


 PC   ADD P   4       P   5       P   6

0008 000 01000000 00111100 01011000
0009 001 01000000 00111100 10011000
000A 002 01000000 00111110 11001000
000B 003 01000000 00111110 00001011
000C 004 00000000 00000010 11101110
000D 005 01000000 00000000 11011001
000E 006 10000000 00110010 00001011
000F 007 01000000 00000010 10101101
```

# Advanced Micro Computers
# ADVANCED MICROPROGRAMMING DEVELOPMENT SYSTEM
# System 29/05 MANUAL

# PART 3
# AMDASM® 29 MANUAL

An affiliate of Siemens
3340 Scott Boulevard
Santa Clara, California 95051

Publication No. 00680102

This manual and the software program it references are the property of Advanced Micro Computers. Use of this manual and the software program is restricted by the Advanced Micro Computers Software License Agreement for System 29, which should be read by each user of this Manual.

No more than five (5) copies of any of the software programs or modification thereof, bearing appropriate copyright notices, may be in existence, without prior written consent of Advanced Micro Computers. The software program, and any copies thereof, may be used only within the Advanced Micro Computers System 29 and only for those purposes indicated in the Software License Agreement.

The information in this publication is believed to be accurate in all respects. However, Advanced Micro Computers disclaims responsibility for any consequences resulting from the use thereof. The information contained herein is subject to change. Revisions to this publication or new editions of it may be issued to incorporate such changes.

|  | REVISION RECORD |
|---|---|
| REVISION | DESCRIPTION |
| A | Original Issue |
| (10/31/78) |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Publication No.
00680102

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning
this manual to:

ADVANCED MICRO COMPUTERS
Publications Department
3340 Scott Boulevard
Santa Clara, CA 95051

# TABLE OF CONTENTS

## TABLE OF CONTENTS (Cont.)

### LIST OF FIGURES

### LIST OF FIGURES (Cont.)

### LIST OF TABLES

# CHAPTER I

## INTRODUCTION AND PURPOSE

An assembler is a program which reads another program written in a symbolic form and produces an output of binary words corresponding to the symbolic input. A microprogram assembler is a special kind of assembler, formally called a meta-assembler. AMDASM 29 is a meta-assembler.

A meta-assembler differs from an ordinary assembler in that most of the symbols are defined by the user prior to the assembly process itself. In an ordinary assembler, the user may define labels for instructions and symbols for particular data words, but the instructions themselves, including their associated word length and format, are generally already defined by the assembler. This makes perfectly good sense in an ordinary assembler, since the assembler is designed to convert an established set of formats into machine language (ones and zeros) for a particular machine such as the AMD Am9080A.

A microprogram assembler, however, must be far more flexible than a traditional assembler, since it must be useful for many hardware configurations. Each different hardware configuration may require a different format and may require word lengths (microinstructions) over 100 bits.

Moreover, in a microassembler, a format rarely establishes the entire contents of a microinstruction, but rather defines only a few bits of the total word.

These requirements imply that a microprogram assembler must consist of two distinct operations. The first operation is establishment of word length and definition of formats and constants (the Definition File). The second operation is the traditional assembly process (Assembly File) performed on a program that uses the formats and constants from the Definition File. The microprogram assembler, therefore, differs from the traditional assembler in that it may be configured, by the user, to accept any word size, formats and constants the user desires.

The assembler written by Advanced Micro Devices is a very powerful meta-assembler, useful not only with the AMD 2900 family, but with any microprogrammed machine. The assembler operates in two phases, the Definition Phase (PHASE1) and the Assembly Phase (PHASE2).

The Assembly Phase is much like any assembler. It reads a symbolic program, handles most common assembler features such as labeling and setting the address counter, and produces a binary output and various listings and cross-reference tables. The Definition Phase is executed first to set up the table which associates the user's format names and constant names with their corresponding bit patterns.

The Definition Phase lets the user define symbols for formats (format names), symbols for constants (constant names), and the microinstruction word length. In the Definition File the length of the microinstruction is defined first. The word may be any length from 1 to 128 bits. This is adequate for all but the most sophisticated processors.

Each of the user defined symbols has a specific bit pattern associated with it. A format name is used to define all, or part, of one microinstruction. The format definition may consist of:

- Numeric fields, which are defined to contain specific bit patterns.
- Variables, which will be filled in when the format is invoked.
- "Don't care" states.

Once the Definition Phase has been executed, its output may be retained and used by future programs.

A useful feature of the AMD assembler is that "don't care" states are retained until defined, which may not happen until after the assembly process, during a third, or post processing, phase. A listing of the microprogram at the conclusion of assembly shows an 'X' for every undefined bit. This is extremely useful during the development process before the microword length has been optimized by sharing fields.

Following assembly of the user's program, a file is retained which contains the assembled microprogram. This file is then available for post processing to create paper tapes for PROM blowers. The output utility can select columns and rows for a given PROM tape, freeing the user from any restrictions regarding the organization of the microprogram memory, and simplifying the generation of a new tape for each of the many PROMs in the system.

The program to be assembled may be written using any of the features specified during the Definition Phase. In the simplest case, the Assembly Phase source program might be written using just strings of ones and zeros, with the Definition Phase consisting only of the microinstruction word length. At the other extreme, the Assembly Phase source program may refer to multiple format names from the Definition Phase for each microinstruction. Any number of formats may be overlayed to define a single microinstruction, as long as the defined or variable fields of each format fall into the "don't care" fields of the other formats invoked. A user might define, for example, a set of formats specifying sequence control operations, another set for data control, and a third set for memory control.

The AMD assembler has been written to maximize its flexibility and ease of use for hardware designers. Every effort has been made to make the program efficient on the machine and efficient at the human interface, with a minimal knowledge of the host machine's operating system required.

NOTE: Throughout this manual examples often refer to the Am2900 Learning and Evaluation Kit shown in Chapter V.

## CHARACTER SET

The following characters are legal in AMDASM source statements:

- The letters of the Alphabet, A through Z. Both upper-and lower-case letters are allowed. Internally, AMDASM treats all letters as though they were upper-case, but the characters are printed exactly as they were input in the source files.
- The digits 0 through 9
- The following special characters:

| **Character** | **Meaning** |
|---|---|
| + | Plus sign |
| – | Minus sign |
| * | Asterisk |
| / | Slash |
| , | Comma |
| ( | Left parenthesis |
| ) | Right parenthesis |
| & | Ampersand |
| : | Colon |
| $ | Dollar sign |
| % | Percent sign |
| Δ | Blank or space |
| ; | Semicolon |
| . | Period |
| cr | Carriage return |
| HT | Horizontal tab |

## DEFINITION OF TERMS

Since there are no standard terms associated with microassemblers, the more common terms used in this manual are listed below:

| **Term** | **Definition** |
|---|---|
| Δ | Indicates a **required** blank character. |
| Name or label | 1-8 characters which are assigned a value by the programmer or the assembly process. Labels are used only in the Assembly File. |
| Constant | A specific pattern of 1-16 bits. |
| Constant name | A name for a constant. |
| Field | A group of adjacent bits in a microinstruction. |
| Format | A model for a microinstruction consisting of fields which contain constants, variables, and "don't cares". |
| Format name | A name for a format. |
| Line | An input line of up to 128 characters on a console, teletype, a paper tape reader, or a diskette file. |
| Modifiers | Symbols (* % : – $) which indicate that the data given for a field is to be modified. |
| Attribute | A modifier which is permanently associated with a field. |
| Designator | A symbol (V, X, B#, Q#, D#, or H#) which indicates the type of field or constant: variable (V), "don't care" (X), binary (B#), octal (Q#), decimal (D#), or hexadecimal (H#). |
| Delimiters | A symbol (: Δ = , /) which indicates the end of a name (: Δ =), the end of a field (,), or the continuation of a statement (/) on another line. |
| Default values | The value which will be substituted if an explicit value is not specified. |
| Options | Choices available which indicate the input and output devices to be used, the type of output listing desired, and processing of one or both phases (Definition and Assembly). |
| { } | Braces indicate that the enclosed parameter is optional. |
| cr | Carriage Return |

## DEFINITION PHASE (PHASE1)

The AMDASM Definition Phase includes the following features:

- A name is a packed group of 1 to 8 characters.
- A name may be assigned to a constant value.
- A name may be used to define a format whose fields are given as variables, "don't cares", explicit bit patterns (values), or specific addresses by using appropriate designators.
- Blanks may be used to improve readability.
- Microword length may be 1 to 128 bits.
- Modifiers include inversion, truncation, negation, and designation of a field as an address field to be right-justified (placing a value in a field at the right with leading bits set to zero).
- The ability to set a "page" size via the attribute $. This permits error detection when the Assembly Phase calls for a jump or branch to an address which is on a different page of the microcode.

Data from the Definition Phase may be retained for use with subsequent Assembly Phase source programs and/or it may be modified as desired.

## ASSEMBLY PHASE (PHASE2)

The Assembly Phase provides for input of the microprogram source statements, conversion of format and constant names to their appropriate bit patterns, substitution of values for variable fields in the format, and generation of listing and binary output. The assembly source program will use references to format names and constant names from the Definition File. It will also contain statements which associate labels with addresses, control assembler operation, and provide program location counter control.

The assembly process provides the user with the following features:

- A microword may be assembled by referring to one or more format names from the Definition File.
- A microword whose format was not specified in the Definition File may be specified by using the built-in free-form format command.
- The programmer may control the program location counter to set the origin and/or to reserve storage.
- The programmer may choose one of four different output listing formats.
- A constant or a variable field may be defined using values and/or expressions.
- Errors are detected and listed. Severe errors cause processing to halt.

Output of the Assembly Phase is an object file which contains the complete microprogram. Post processors can directly convert this object file to any form needed, such as hexadecimal or BNPF punched on paper tape.

## IMPLEMENTATION

AMDASM 29 operates on the Advanced Micro Computers' System 29 under the AMDOS 29 Operating System.

## ASSEMBLER OPERATION

AMDASM is placed into execution by control statements from the console input device.

The Definition File is processed in PHASE1 and if it contains no errors the Assembly Phase begins. PHASE2 Pass 1 assigns values to Assembly File labels and allocates storage. PHASE2 Pass 2 translates the Assembly File source program into object code.

User-selected options determine whether the Definition Phase is to be executed or if a previous execution of that phase has already established the table of formats on a file which will be used by the assembly process.

The AMDOS 29 operating system allocates all necessary input and output resources, such as files, automatically.

# CHAPTER II

## DEFINITION PHASE (PHASE1)

The Definition Phase allows the user to define the microword length, constants, and formats which he will use to write source programs for his target machine.

## DEFINITION FILE

The definitions are input via a sequence of instructions called the Definition File whose content includes the following items:

TITLE (heading to be printed on output listing)
WORD n (defines microinstruction word length)

| |
|---|
| Printing control statements<br>Definition statements<br>Comment statements |

END

The control statement WORD must appear as the first statement in the Definition File after the optional TITLE statement. The END statement must be the last statement in the Definition File.

The other statements (shown boxed) may be interspersed throughout the body of the file.

To facilitate readability, blanks may appear in most parts of these statements, although no blanks are permitted between the letters of the control words TITLE, WORD, END, LIST, NOLIST, DEF, EQU, or SUB. An entire blank line may be inserted by entering a semicolon and a carriage return.

### TITLE

If the user wishes to have a title printed on his Definition File statements, the first statement input should be TITLE. The general form is:

Form:

TITLEΔ title desired by user

TITLE must:

- Begin on a new line
- Be followed by a blank and a maximum of 60 characters.

### WORD

WORD must be the first statement input by the user after the optional TITLE is given. Its general form is:

Form:

WORDΔn

WORDΔ must be followed by a decimal integer value n which indicates the microword size in bits (range 1 – 128).

WORD must:

- Be followed by at least one blank and 1 to 3 decimal digits.
- Be the first input line (second input line if TITLE was used).
- Begin on a separate line.

If WORD is omitted, assembly will halt as the Definition Phase must know the size of the microword in order to proceed.

### END

END indicates the end of the Definition File. If END is omitted an error message will be printed but processing will continue. The general form is:

Form:

END

END must:

- Begin on a new line.
- Be the last statement in the Definition File.
- Be followed by a carriage return.

## PRINTING CONTROL STATEMENTS

Printing control statements are used to control printing.

TITLE was listed separately since it must be the first statement input if it is to be printed at the top of the first page of the output. TITLE may be used elsewhere (i.e., interspersed with the statements shown in the box) in which case it causes this new title to printed at that position in the output file.

A description of the other printing control statements, LIST, NOLIST, EJECT and SPACE.

### LIST

LIST indicates that the following statements are to be printed whenever printing of the Definition File input is requested. This feature will be most useful when correcting or modifying a Definition File. (AMDASM selects LIST as the default option. NOLIST must be specified if the user does not wish to print his Definition File source statements.) The general form is:

Form:

LIST

LIST must:

- Begin on a new line.
- Be followed by a carriage return.
- Precede the Definition File statements which are to be printed.
- Be interspersed between complete definition statements.

### NOLIST

NOLIST turns printing off, and no printing of the Definition File input statements will occur until LIST is encountered. However, *any source statement containing an error will still be listed.*

Form:

NOLIST

NOLIST must:

- Begin on a new line.
- Be followed by a carriage return.
- Precede the Definition File statements which are not to be listed.
- Be interspersed between complete source statements.

## SPACE

SPACE indicates that the assembler is to leave n blank lines before printing the next source statement. The general form is:

---

Form:

---

SPACEΔ n

---

SPACE must:

- Begin on a new line.
- Be followed by Δ and a decimal digit indicating the number of succeeding lines to be left blank.
- Be inserted in the Definition File at the point where the spaces are desired.

## EJECT

When EJECT is encountered, the assembler generates blank lines on a list device so that any previous lines plus the blank lines equals the specified "page" length (default is 66 lines). It then begins a new "page", headed with the title. On a printer a new page is ejected. The general form is:

---

Form:

---

EJECT

---

EJECT must:

- Begin on a new line.
- Be followed by a carriage return.

## DEFINITION STATEMENTS

Definition statements are used to define constants, full microword formats, or partial microword formats. The general form of these statements is:

---

Form:

---

name: definition wordΔ field1, field2, . . ., fieldn
or
constant

---

## DEFINITION WORDS

The definition words and their functions are:

EQU is used to set a name equal to a bit pattern
DEF is used to define a format for a microinstruction
SUB is used to define a format for part of a microinstruction

A complete explanation follows the section defining fields, designators and constants (page 5).

## FIELDS

A field is a contiguous group of bits in a microinstruction (such as branch address, next instruction control, etc.). Each field may be one of three types:

- A constant field whose content is a fixed value or a fixed bit pattern, (for example, the next instruction control).
- A variable field whose content will contain different bit patterns in different situations (for example, an address field).
- A don't care field whose content is not used in this format (for example, the address field for a continue instruction).

The type of data in a particular field is indicated by using "designators".

## DESIGNATORS

Permissable designators and their meanings are.

| **Designator** | **Meaning** | **Example** |
|---|---|---|
| B# | A constant or field whose contents will be represented using binary digits (0 and 1). Each digit has an implicit length of one bit. | B#101 (three bits 101). |
| Q# | A constant or field whose contents will be represented using octal digits (0 through 7). Each digit has an implicit length of three bits. | Q#32 (six bits 011010). |
| D# | A constant or field whose contents will be represented using decimal digits (0 through 9). For a constant name definition using EQU, the implicit length for decimal numbers is the number of bits needed to represent the number in binary. Thus, D#3 has an implicit length of 2, D#4 has an implicit length of 3. For fields in a format (DEF or SUB), the D# must be preceded by decimal digit(s) giving an explicit length (number of bits) for the field. | D#4 (three bits 100)<br>3D#6 (three bits 110) |
| H# | A constant or field whose contents will be represented using hexadecimal digits (0 through 9, A through F). Each digit has an implicit length of four bits. | H#8A (eight bits 10001010) |
| X | A "don't care" field. X must be preceded by decimal digit(s) giving an explicit length for this field (i.e., the bit length). | 4X (4 bit "don't care" field). |
| V | A variable field. V must be preceded by a decimal digit(s) giving an explicit length for this field (i.e., the bit length). | 6V (six bit variable field). |

When a designator B#, Q#, D# or H# is given after a V, it becomes a permanent attribute of that field and the assembler assumes that any value specified for that field will be given in digits appropriate to that designator.

These permanent designators for variable fields may be overridden when using the format during the Assembly Phase. **If a variable field has no designator given, it defaults to binary.** For example, if all variable fields are given as nVQ# in the Definition Phase, all values for this variable field that are octal may be written during the Assembly Phase by writing only the necessary octal digits.

The content of a variable field may be given during the Definition Phase. The V designator may be followed by the B#, Q#, D#, or H# and these may be followed by appropriate digits called the default value for this field.

Thus, 6VQ# indicates a 6-bit variable field whose contents will be given in octal. 6VQ#35 indicates that if no value is substituted in the Assembly Phase, this variable field should assume the default value 011101.

NOTE: The designators B#, Q#, D#, H# must have no blanks between the letter and the #. The desired value for the field is then given in the appropriate digits as shown in the examples.

## FIELD RULES

Each field following a definition word must:

- Contain a maximum of 16 bits unless it is a "don't care" field.
- Be followed by a comma unless it is the last or only field following the definition word.
- Define a constant field using the designators B#, Q#, D#, or H# and the appropriate digits.

or

- Be a variable which gives a bit length and the designator V. If no designator follows the V, the field type defaults to binary.

or

- Be a "don't care" which contains a bit length and the designator X.

or

- Be a constant name or subformat name which has been previously defined.

## NAMES

Names may be user-defined constant names, format names, or subformat names.

Names must:

- Be the first element in a statement.
- Begin with an alphabetic character (A-Z) or a period (.).
- Be terminated by a colon (:).
- Contain a maximum of 8 characters not including the colon.
- Not contain any embedded blanks.
- Be followed by EQU, DEF or SUB.
- Contain only alphabetic characters (A-Z), a period (.) or the digits (0 through 9) in positions 2 through 8.

Names may:

- Contain more than 8 characters but will be truncated after the first 8 characters.
- Be preceded by blanks.
- Be followed by blanks after the : and before the EQU, SUB, or DEF.

Examples of proper names are:

NUMBER:
.SHIFT:
REG.3:

Improper names are:

| | |
|---|---|
| *ADD | (special character used) |
| SHIFT LEFT: | (embedded blank, more than 8 characters) |
| 3MUXCNTL: | (first character not A through Z or period) |

## CONSTANTS

Constants are used to associate a name with a value or to define a specified fixed bit pattern.

Constants may be expressed by using designators and the appropriate digits.

For example
Q#62
defines the bit pattern 110010. This type of constant has an implicit bit length of 6 bits (each octal digit represents 3 bits).

If a decimal digit precedes the designator, as for example in
4H#5
the 4 represents the explicit length of the field, and the bit pattern is 0101.

Explicit and implicit lengths are more fully defined later in this chapter.

Constants must be represented in 16 bits (i.e., $2^{16} - 1$ maximum). The permissible forms for constants are:

| Form | Permissible Digits | Meaning |
|---|---|---|
| n | 0 through 9 | Decimal value (default form) |
| i B#n | 0 or 1 | Binary value |
| i Q#n | 0 through 7 | Octal value |
| i D#n | 0 through 9 | Decimal value |
| i H#n | 0 through 9<br>or A through F | Hexadecimal value |

where i represents optional digits specifying the explicit length.

## EXPRESSIONS

When a field contains an expression, the expression may use designators and/or digits or labels as well as operators.

Operators permitted in expressions are:

| Operator | Description |
|---|---|
| + | Add the value of the left operand to the value of the operand on the right of + |
| − | Subtract the value of the operand to the right of the minus (−) from the value of the operand on the left |
| * | Multiply the left operand by the right operand |
| / | Divide the operand on the left (dividend) by the operand on the right (divisor) |

All expressions:

- *Are evaluated from left to right. There is no hierarchy for the operators and no parenthesis for nesting are permitted.*
- Must result in a value which is a positive constant.
- Are calculated using integers; remainders are discarded.

## DEFINITION WORDS

The definition words EQU, DEF and SUB are described in detail in this section.

### EQU

EQU is used to equate a constant name to a constant value or expression. The general form is:

Form:

name: EQUΔ constant (or expression)

This equates the characters given in the name position to the value of the constant or expression. Only one expression or constant is permitted following the EQU.

The following sets the name R12 equal to the bit pattern 1100:

R12:EQUΔH#C

Future references to the bit pattern 1100 (register 12) may be made by using the name R12.

**The default type is decimal if no designator follows the EQU. (R10:EQUΔ10 assumes the bit pattern 1010, implicit length 4 bits).**

Each EQU must:

- Begin on a new line.
- Begin with a name:
- The name: must be followed by EQUΔ (blanks between : and EQU are optional).
- Contain a constant, expression or a constant name which represents a bit pattern.
- Define a value which can be represented in 16 bits ($2^{16} -1$ maximum).

Each EQU may:

- Be followed by a semicolon and comment after the constant or expression.
- Be continued on additional lines by using / (slash) as the first nonblank character in those lines.
- Be used in the Assembly File as well as in the Definition File.

## DEF

DEF is used to define a complete microword format establishing the contents of unvarying portions of the microword and establishing the position and length of variable and "don't care" fields. In addition, default values for variable portions of the word may be specified. The general form is:

Form:

name: DEFΔ field1, field2, . . ., fieldn

Each DEF must:

- Begin on a new line
- Be preceded by a name:
- Be followed by one or more blanks, then fields separated by commas.
- Have the sum of the lengths of all fields exactly equal the microword length specified by WORD.
- Begin on a new line
- Specify every bit in the microword in terms of constants, "don't cares", or variables.

A DEF may:

- Contain blanks between name: and DEFΔ.
- Be continued on additional lines by using a / (slash) as the first nonblank character in those lines.
- Be followed by a semicolon and a comment after any full field is defined.
- Contain (in any field) a subformat name or constant name which has been PREVIOUSLY defined.
- Contain a variable, "don't care", constant or expression in any field.
- Contain a variable field which specifies a default value for the field. The default value may be a constant or a "don't care".
- Be overlayed on "don't care" fields with another format to obtain a complete microword during the Assembly Phase. Overlaying on other than "don't care" fields will result in errors, so this feature must be used with care.

## SUB

SUB is used to define a subformat which is the format of a portion of the microword. A subformat is the same as a format except that it contains fewer bits than the full microword. The fields may be constants, variables or "don't cares". Its general form is:

Form:

name: SUBΔ field1, field2, . . , fieldn

Each SUB must:

- Be preceded by a name:
- Be followed by one or more blanks, then fields separated by commas.
- Precede the DEF in which it is first referenced.
- Begin on a new line.
- Not be used in the Assembly File.

A SUB may:

- Be less than a microword length in bits.
- Be continued on additional lines by using / (slash) as the first nonblank character in those lines.
- Be followed by a semicolon and a comment after any complete field.
- Contain (for any field) a constant name that was PREVIOUSLY defined, or a constant, expression, variable, or "don't care" specification.

A SUB will be useful when several formats contain identical adjacent fields. In this case, the subformat name may be used in each DEF whenever these fields occur.

## EXAMPLES OF EQU, SUB, DEF

An EQU is used to associate a bit pattern with a symbol (constant name); one example is:

R2: EQUΔ B#010

This defines the name R2 as a 3-bit constant with the bit pattern 010. Whenever the symbol R2 is used, the bit pattern 010 will be substituted.

A SUB might be:

SHFTRT:SUBΔ 3V, B#10110, 5X

This defines SHFTRT as a subformat with a 3-bit variable field (3V), a 5-bit constant field (B#10110), and a 5-bit "don't care" field (5X) for a total of 13 bits.

A DEF is used to associate bit patterns with a symbol (format name). One example is:

ADD: DEFΔ 3V, B#10110, 5X, B#0011, 4X, B#010

This defines ADD as a format with a 3-bit variable field (3V), a 5-bit constant field (B#10110), a 5-bit "don't care" field (5X), a 4-bit constant field (B#0011), a 4-bit "don't care" field (4X), and a 3-bit constant field (B#010). This gives a total microword length of 24 bits.

Alternatively, the same format name could be written using the subformat name (SHFTRT) and the constant name (R2) previously defined by writing:

ADD: DEFΔ SHFTRT, B#0011, 4X, R2

Another example of an EQU is:

TWOK: EQU∆ 2048

This assigns the bit pattern 100000000000 and a length of 12 bits to the name TWOK. The 2048 is assumed to be decimal and the length is taken from the rightmost bit through the leftmost bit in which a 1 appears.

Thus,

EIGHT: EQU∆ 8

yields the bit pattern 1000 with a length of 4

Alternatively, by using different designators, the constant

TWOK: EQU∆ 2048

could be written:

```
TWOK:  EQU∆  B#100000000000
TWOK:  EQU∆  Q#4000
TWOK:  EQU∆  H#800
```

All of these yield the bit pattern 100000000000 and a length of 12.

## FIELD LENGTHS

Each field may be given an explicit or implicit length. An explicit length is indicated for a field by using decimal digit(s) before the designator. The maximum length is 16 bits except for don't care fields whose maximum length is the microword size.

Thus,

3B#101

indicates a field with an explicit length of 3 bits.

Decimal, variable or "don't care" designators **require** an explicit length before the designator D#, V or X.

"Don't care" or variable fields **require** an explicit length since they do not, necessarily, initially contain a definite bit pattern.

Decimal **fields** in a format or subformat require an explicit length since there is no direct correlation between the number of decimal digits given and the number of binary bits desired for this field.

| **Example** | **Description** |
|---|---|
| 4V | Defines a variable field with the explicit length of 4 bits. |
| 5D#16 | Defines a constant field with the explicit length of 5 bits and the bit pattern 10000. |
| R3:EQU∆5 | Defines a constant using the default type decimal, value 5. The implicit bit length is 3. |

## CONSTANT LENGTHS

A constant may have an implicit or an explicit length. An explicit length is given by placing the bit length (in decimal digits) before the designator. Thus,

B:EQU∆4D#8

has an explicit length of 4 and the bit pattern 1000.

If an explicit length is not given, the constant is assigned an implicit length determined by the designator used.

**Table 2-1**
**Implicit Length Attributes of Constants**

| Constant | Implicit Length | Binary Value | Description |
|---|---|---|---|
| AB:EQU∆B#1000 | 4 | 1000 | Each binary digit yields an implicit length of 1 bit per digit. |
| BB:EQU∆Q#10 | 6 | 001 000 | Each octal digit yields an implicit length of 3 bits per digit. |
| CB:EQU∆H#10 | 8 | 0001 0000 | Each hexadecimal digit yields an implicit length of 4 bits per digit |
| DB:EQU∆12 | 4 | 1100 | The 12 is assumed to be decimal, and the implicit length is counted from the rightmost bit through the leftmost 1. |
| EB:EQU∆4 | 3 | 100 | Same as above. Implicit length 3. |

## CONTINUATION

Any statement may be continued on additional lines by placing a / (slash) as the first nonblank character in those lines.

A continuation must:

- Have a slash as the first nonblank character in its line.
- Preferably be indicated after a complete field (including the comma) has been given on the preceding line.
- Never occur between the designators B, D, Q, or H, and the # sign.

Examples are:

```
SHFTRT: SUB∆ 3V, B#10110,
/5X

ADD: DEF∆ 3V, B#10110, 5X,
/B#0011, 4X, B#010
```

## COMMENT STATEMENTS

A comment statement is used to provide information about program variables or program flow. The general form is:

Form:

; comment text

A comment may be a full or a partial line. All data from the semicolon to the end of the input line is ignored by the assembler.

Comments must:

- Begin with a semicolon.
- Be placed after a complete field if used within a DEF or SUB, in which case subsequent fields for that DEF or SUB must begin on a new line with a / (slash) indicating that they are a continuation of this DEF or SUB.

For example:

1. SHFTRT: SUBΔ 3V, ; this is a shift right subformat
2. / B# 10110, 5X; which is continued on a second line
3. ; the ADD given below is a complete microword format
4. ADD: DEFΔ SHFTRT, B#0011, 4X, R2
5. ; total number of bits for SHFTRT is 13
6. ; the bit pattern for SHFTRT will be substituted
7. ; in the ADD given above

Statements 3, 5, 6, and 7 are full comment lines. Statements 1 and 2 are statements to be processed but all characters after the 'semicolon' will be treated as comments. The SUB begun in statement 1 is continued in statement 2 where '/' indicates continuation.

## MODIFIERS AND ATTRIBUTES

Modifiers are placed after a constant or after the designator V. When placed after a constant they alter only the value given. When used after a V, the modifiers are called attributes of that field and are permanently associated with the field. Attributes will modify any default value given with the variable field in the Definition File and they will modify any value substituted for this variable field when the format name is used in the Assembly File.

Permitted modifiers and their actions are:

| **Modifier** | **Action Performed on Constants or Default Values** |
|---|---|
| * | Inversion (one's complement) |
| – | Negate the number (two's complement) |
| : | Truncate on the left to make the value given fit into the number of explicit bits for this field. |
| % | This field is to be considered an address field. Any value given is to be right-justified in the field and any bits remaining on the left are to be filled with zeros. |
| $ | The field is treated as an address within a "paged" memory organization. This attribute permits substitution in this regard and initiates out-of-bounds page checking logic. Used only with variable fields as an attribute (may not follow a default value). |

Examples of correct use of modifiers with constants:

| **Example** | **Description** |
|---|---|
| D#5* | Yields bit pattern 010 (101 (5) is inverted). |
| B#0101– | Yields bit pattern 1011 (0101 is two's complemented). |
| 6Q#357: | Yields bit pattern 101 111 (the left bits 011 (3) are truncated). |
| 12H#A5% | Yields bit pattern 0000 1010 0101 (the A5 is right justified in a 12 bit field). |

Examples of incorrect fields due to ommision of modifiers:

| **Example** | **Description** |
|---|---|
| 4B#101 | Explicit length is 4 bits, only 3 bits follow the B # but no % sign (indicating right justification) is given. |
| 5Q#34 | Explicit length is 5 bits but the 34 generates 6 bits and no : has been given to indicate that the leftmost bit is to be truncated. |

Modifiers must:

- Appear after the value of a constant (i.e., 12H#4C% or 5Q#37:).
- Appear after the V but before the (optional) default value for a variable field (12V%Q#46), if they are to be permanent attributes of the field. The % and the Q# become permanent attributes of this variable field and are also modifiers of the default value. To modify only the default value, modifiers must follow the value (12VQ#46%).
- Not appear with "don't cares" (e.g., 3X% is illegal).

  The modifiers * and – may not both be used for the same field.

A more detailed description and examples are given in Chapter III.

## MODIFIER PRECEDENCE

Modifiers or attributes may appear in any order but will always be processed in the following order:

| **Modifier** | **Description** |
|---|---|
| * or – | Inversion or negation |
| % | Right justification |
| : | Truncation |
| $ | Paged addressing |

## DESIGNATORS AS ATTRIBUTES

Variable fields may use the B#, Q#, D# and H# as attributes. Once given, B#, Q#, D# and H# are permanently associated with that variable field unless overridden. If a variable field has no radix base specified, it will **default to binary.**

If the user always wants to input assembly variables in octal, each variable field in the Definition Phase should be written as nVQ#. Then, in the Assembly Phase the value for this field may be given as, 27, and the program will assume that these are octal digits. If, in the Assembly File, octal is not desired, the field in the Assembly File program could be written as B#010111, or H#27, etc., to override the octal attribute.

If a variable field is defined with a default value (4VH#C), the designator (H#) becomes an attribute of that field.

The attribute H#, if given with a variable field in the Definition File, may need to be repeated in the Assembly File. This is necessary since the program can not distinguish hexadecimal values which begin with A through F from names, which may also begin with the letters A through F.

## $ ATTRIBUTE

The $ attribute may be used only with variable fields to indicate paged addressing.

When the $ is given with a variable field, the % and : attributes are automatically set for that field.

The $ will indicate that this is a field whose remaining upper (leftmost) bits are to be truncated and compared with the corresponding bits of the current Program Counter.

If the truncated bits do not agree with the corresponding bits of the PC, an error occurs.

The desired length of the "page" is determined by the number of bits given as the width of this variable field.

Thus, if a "page" is to be 256 words deep, the variable field would be defined as 8V$. Any value substituted for this field will be truncated on the left and the remaining eight right-hand bits will be substituted into the field. If the truncated left bits do not agree with the corresponding bits of the current program counter value, the substitution would attempt to produce a jump to another page; thus an error message is generated.

## "DON'T CARES"

A "don't care" is used to indicate the bits (a field) whose state (bit pattern) is irrelevant in this microword instruction.

The general form is:

| Form: |
|---|
| nX |

where

n is the number of bits (in decimal), and X indicates "don't care".

"Don't cares":

- Are printed as an X in the Assembly Phase output.
- May be assigned the value 0 or 1 during the post processing phase.
- Are the only fields which may be greater than 16 bits in length.
- Are the only fields in a format which may be overlayed (or'ed) with another format which contains a constant in the same field.

## VARIABLES

Variables are used to define microword fields whose contents need not be assigned until assembly time. A variable field may be assigned a default value in the Definition File. The general forms are:

| Form: |
|---|
| nV |
| nV attributes |
| nV attributes default-value |
| nV attributes default-value modifiers |
| nV default-value modifiers |

A variable field must:

- Be preceded by an explicit length (n) which gives (in decimal) the bit length of the field. (n ≤ 16)
- Contain a V after the length.
- End with a comma (,) if another field follows it.
- Contain a % after the V if an expression or the program counter is to be used as a substitute for this field in the Assembly File.

A variable field may:

- Contain attributes (immediately after the V), such as inversion (*), which will always invert any value given for this field.
- Contain a designator given with or without a default value which will automatically determine the default type for this field.
- Contain a default value given in binary indicated by (B#), octal (Q#), hexadecimal (H#), or decimal (D#) followed by the desired digits.
- Contain modifiers after the default value. These modify **only** the default value and are not permanently associated with this variable field.
- Contain a default value given as X (indicating "don't care") if the user wishes to overlay this field during the Assembly Phase.
- Contain **either** a default value of "don't care" **or** an explicit default value (bit pattern) but not both.

Examples of the correct use of variable fields with a default value of "don't care" are:

3VX<br>
3V*X<br>
3V−%X<br>
3V*:X

## EXAMPLES OF VARIABLE FIELDS

| Field Content | Meaning |
|---|---|
| 3V | A 3-bit field. The content is variable and will be supplied when this format name is used in the Assembly File. The field type defaults to binary. |
| 3VQ# | A 3-bit field whose content is variable. The content will be supplied when the format name is used during the Assembly File. The content may then be given as one octal digit without using the designator Q#. If the content is to be given in binary, decimal, etc., then the designator B# or D# would be placed before the digit(s) given in the Assembly File. |
| 3V*% | A 3-bit field whose content is variable. Any value given for this field within the Assembly File will automatically be inverted and right-justified. Since no designator is given, the field defaults to binary. If the content is to be given in octal, etc. in the Assembly File, the appropriate designator (Q#, H#, D#) must precede the digit(s). |
| 3VQ#5 | A 3-bit field whose content is variable. If no value is specified for this field in the Assembly File, it will assume the default value (specified as Q#5) bit pattern 101. |
| 3VQ#5* | Is the same as above but the 5 is inverted to yield the bit pattern 010. Values *substituted* for this field during the Assembly File are not automatically inverted. |
| 3V*Q#5 | Yields the same pattern as 3VQ#5* but, in addition, any value substituted during the Assembly File for this field will also be automatically inverted since the * follows the V rather than the 5. |
| 3V*Q#5* | Yields a 3-bit variable field with a default value of 5, inverted, then inverted again by the * following the V. The resulting bit pattern is 101. Any value substituted for this field in the Assembly File will be inverted. |

To summarize, attributes placed immediately after the V are permanently attached to this field and will operate on any default value given with the field as well as any value substituted for the field in the Assembly File.

Modifiers placed after a default value apply only to the default value.

Examples of incorrect variable fields are:

| Field Content | Description |
|---|---|
| 3VH#7 | The H#7 yields 4 bits. No : was given to indicate that the left bit should be truncated to fit the 3-bit field. |
| 3:VH#7 | The : is in an incorrect position. It should be 3V:H#7 or 3VH#7: (depending on whether the truncation is a permanent field attribute or a modifier of the default value H#7). |

In short, attributes must be placed immediately after the V. Modifiers must be placed immediately after the digits given for the default value.

## DEFINITION FILE RESERVED WORDS

The following words are used during the assembly phase as assembler control statements and may not be used as format names or constant names in the Definition File :

ALIGN
EJECT
END
EQU
FF
LIST
NOLIST
ORG
RES
SPACE
TITLE

## SAMPLE DEFINITIONS

Some possible ways of defining a few of the fields and formats for the Am2900 Learning and Evaluation Kit (see Figure 5-2) are:

```
 R2:EQUΔH#2  }  Registers
R11:EQUΔH#B  }

CONT:DEFΔ4X, B#0010,24X           }  Next instruction
BREGFEQ0:DEFΔ4VH#,4D#12,24X       }  control
```

Registers 2 and 11 are defined as 4 bits, with the assigned values 2 (0010) and 11 (1011), respectively.

CONT (continue) defines only the four bits (shown as 27-24 in Figure 5-2) with the pattern 0010. All other bits are left as don't cares.

BREGFEQ0 (Branch Register if F = 0) defines the four bits (bit numbers 31-28 in Figure 5-2) as a variable field, to be given a value during the Assembly Phase using hexadecimal digits. The next four bits (bit numbers 27-24 in Figure 5-2) are given the constant pattern 1100 (value 12). All other bits are don't cares.

## NUMBER OF PERMITTED EQUs, DEFs, AND SUBs

There is no fixed maximum number of EQUs, DEFs or SUBs because AMDASM stores all data dynamically. The user of a 32K-byte system has available, in PHASE1, approximately 10K bytes for variable storage; PHASE2 has approximately 8K bytes.

PHASE1 allocates:
- 12 bytes for each EQU
- 12 bytes for each format or subformat name
- 4 bytes for each field in a DEF or SUB

PHASE2 allocates:
- 12 bytes for each format name, constant name and label
- 4 bytes for each format field

## HORIZONTAL TABS

A horizontal tab may be entered for readability as the user inputs his source files. The assembler places the character following the horizontal tab at the next tab position. Tab stops begin with position 1, and occur every eight positions thereafter as follows: position 1, 9, 17, 25, etc. Thus if data is input at character position 5, a tab will place the next character input at position 9. However, if data is input at character position 17, a tab will place the next character at position 25.

**Horizontal tabs may be used in both the Definition and Assembly Files.**

# CHAPTER III

## ASSEMBLY PHASE (PHASE2)

The Assembly Phase reads in the source program statements, assigns values to labels and constants, then translates the source program's **executable** statements into a binary format. The Definition Phase output (a table of format and constant names and their associated bit patterns) is used for this translation.

The user must input his source program statements in the order corresponding to the desired order of his executable statements. The user may allocate blocks of storage, control printing, and set the program counter via nonexecutable assembler control instructions which are interspersed with, and do not affect the order of, his executable statements.

The object code is input via a sequence of instructions called the Assembly File whose content includes the following:

---

TITLE (heading to be printed on the output listing)

Printing control words
Program counter control words
Constant definition word
Executable statements
Comments

END

---

The optional TITLE statement is usually input first so that the desired title appears on the first output page.

The other statements (shown boxed) may be interspersed throughout the body of the file. However, the executable statements must be input in the order that corresponds to the desired sequence of the object (micro) code.

The END statement must be the last statement in the Assembly File.

The permissible Assembly Phase statements are:

| | |
|---|---|
| TITLE<br>LIST<br>NOLIST<br>SPACE<br>EJECT | Printing control words |
| ORG<br>RES<br>ALIGN | Program counter control words |
| EQU | Constant definition word |
| FF | Free form definition word to establish a microword content |

References to format names from the Definition Phase

| | |
|---|---|
| Comments | Used for documentation and program flow. |
| END | End of the Assembly File. |

None of the control words (LIST, ORG, etc.) or format names may contain blanks.

## ASSEMBLY FILE STATEMENTS

Each statement contains an optional label followed by a statement type. Some statement types must be followed by an argument which may be a constant, a constant name, or an expression.

The general form of all Assembly File statements except comments is:

---

Form:

---

$$\left\{\begin{array}{l}\text{label:}\\ \text{name:}\end{array}\right\}\ \begin{array}{l}\text{control word}\\ \text{format name}\\ \text{definition word}\end{array}\ \left\{\ \Delta\text{arguments}\ \right\}$$

---

### CONTINUATION

Any statement may be continued on additional lines by placing a / (slash) as the first nonblank character in those lines.

### LABELS OR NAMES

Labels or names are packed groups of letters and/or symbols which have an associated value.

Labels are permissible with executable statements and names are required with the definition word EQU.

---

Form:

---

name: definition word

or

label: format name

---

A name or label's value is determined by the statement type which follows it. Thus,

name: EQUΔn

equates the symbol "name" with the value given for "n",

while

label: format name Δ VFS, VFS . . .

equates label to the current value of the program counter, so that reference may be made to this location in the microcode by using this label.

A label or name must:

- Begin with an alphabetic character (A through Z) or period (.).
- End with a colon.
- Contain no more than 8 characters, exclusive of the colon. (Excess characters are truncated on the right.)
- Contain no imbedded blanks.
- Each be unique. If duplicates are given, the value given at the first occurrence is used and a warning message is issued for each duplicate.

A label or name may:

- Precede an EQU, RES, ORG, FF, or an executable instruction
- Be used as a variable field substitute (VFS)
- Be used as a field in an FF statement
- Not be a reserved word
- Contain only the letters A-Z, numerals 0-9 or a period (.) in positions 2 through 8.

When a name is defined by an EQU, the definition (source statement) must precede the use of the name as a field or a constant. If the statement

AM2909:DEFΔJSR,28X

is given, it must be physically located in the source program after the statement

JSR:EQUΔH#5

A good general rule is to place all EQUs at the beginning of the Assembly File program.

## ENTRY POINT SYMBOLS

When a label is followed by a double colon (::) it is called an Entry Point. Entry Points are used when generating Mapping PROMs to easily obtain the program (location) counter value associated with certain points in the microcode.

Entry Points are indicated in the assembly source file as

label: : format name Δ VFS, . . .

Except for the double colon, Entry Points are subject to all the rules applicable to labels.

A list of the Entry Points (symbols and values) may be obtained when AMDASM is executed by requesting the MAP option (see Chapter 4, page 20).

## STATEMENT TYPES

The Assembly File uses six general types of statements. These are listed below with their permissible control words:

- Printing control statements (LIST, NOLIST, SPACE, EJECT, TITLE)
- Program counter control statements (RES, ORG, ALIGN)
- Constant definition statement (EQU).
- Executable instruction statements (format names from the Definition Phase, FF).
- Comment Statements (;).
- END Statement

## PRINTING CONTROL STATEMENTS

### TITLE

All data input on the line with TITLE will be printed at the top of each page of output. A maximum of 60 characters may be input for a title. When a new TITLEΔ is encountered the list device ejects blank lines to complete the present page and succeeding "pages" will contain this title. A "page" is not necessarily a physical page since the user may specify the length (number of lines) of a "page". The general form is:

| Form: |
|---|
| TITLE Δ   alphanumeric data to be printed<br>at the top of the page |

### LIST

LIST indicates that the following statements are to be printed whenever printing of the Assembly File input is requested. This feature will be most useful when correcting or modifying an Assembly File. (AMDASM automatically prints the source statements unless NOLIST is specified by the user.) The general form is:

| Form: |
|---|
| LIST |

LIST must:

- Begin on a new line.
- Be followed by a carriage return.
- Precede the Assembly File statements which are to be printed.
- Be interspersed between **complete** assembly statements.

### NOLIST

NOLIST turns off the printing of assembly source statements. Printing of the Assembly File input will be suppressed until LIST is again encountered. Any source statement containing an error will still be printed. The general form is:

| Form: |
|---|
| NOLIST |

NOLIST must:

- Begin on a new line.
- Be followed by a carriage return.
- Precede the Assembly File statements which are not to be listed.
- Be interspersed between complete assembly statements.

### SPACE

SPACE indicates that the assembler is to leave n blank lines before printing the next source statement. The general form is:

| Form: |
|---|
| SPACEΔ n |

SPACE must:

- Begin on a new line.
- Be followed by Δ and a decimal digit indicating the number of succeeding lines to be left blank.
- Be inserted in the Assembly File at the point where the spaces are desired.

### EJECT

When EJECT is encountered, the assembler generates blank lines on a list device so that any previous lines plus the blank lines equals the specified "page" length (default is 66 lines). It then begins a new "page", headed with the title. On a printer a new page is ejected. The general form is:

| Form: |
|---|
| EJECT |

EJECT must:

- Begin on a new line.
- Be followed by a carriage return.

## PROGRAM COUNTER CONTROL STATEMENTS

### ORG

ORG is used to set a new program counter (PC) origin. The next assembled microword will be located at the new origin. The general form is:

Form:

ORGΔ n

ORG must:

- Be followed by at least one blank and n.
- Have n specified using decimal digits unless one of the designators B#, Q# or H# precedes the digits given.
- Be used only for setting the program counter forward.
- Be greater than or equal to the current value of the program counter.

ORG may:

- Contain an expression instead of n.
- Be used an unlimited number of times in the Assembly File.

**If no ORG is specified the assembler uses an initial PC of 0.**

### RES

RES is used to reserve n words of memory. This increments the program counter by n. The reserved words will automatically be filled with "don't cares" by the assembler. The general form is:

Form:

RESΔ n

RES must:

- Be followed by at least one blank and n.
- Have n specified using decimal digits unless one of the designators B#, Q# or H# precedes the digits given.

RES may:

- Contain an expression instead of n.
- Be used an unlimited number of times in the Assembly File.

### ALIGN

ALIGN is used to set the program counter to the next value which is an integral multiple of the value n. It is used to align the program counter to a specific boundary such that the next microinstruction will be assembled at an address which is, for example, the next integral multiple of 2, 4, 8 or 16. The general form is:

Form:

ALIGNΔ n

ALIGN must:

- Be followed by at least one blank and n
- Have n specified using decimal digits unless one of the designators B#, Q#, H# precedes the digits given.

ALIGN may:

- Contain an expression instead of n.
- Be used an unlimited number of times in the Assembly File.

## CONSTANT DEFINITION STATEMENT

### EQU

EQU is used to equate a constant name to a constant value or expression. The general form is:

Form:

name: EQUΔ constant (or expression)

This equates the characters given in the name position to the value of the constant or expression. Only one expression or constant is permitted following the EQU.

Each EQU must:

- Begin on a new line.
- Begin with a name:
- The name: must be followed by EQUΔ (blanks between : and EQU are optional).
- Contain a constant or expression which represents the bit pattern for one field.
- Define a value which can be represented in 16 bits ($2^{16}-1$ maximum).

Each EQU may:

- Be followed by a semicolon and comment after the constant or expression.
- Be continued on additional lines by using / (slash) as the first non-blank character in these lines.
- Be used in the Assembly File even if defined in the Definition File.
- Be equated to the current value of the program counter by using $ as the designator. The $ may be part of an expression.

Examples of EQUs:

ADD:EQUΔQ#0

defines a 3-bit field whose bit pattern is 000.

This could be an ALU function of ADD for the Learning Kit.

PUSH:EQUΔH#9

defines a 4-bit field, bit pattern 1001 which might represent the next microinstruction control field in the Learning Kit.

## EXECUTABLE STATEMENTS

Executable statements form the body of the Assembly Phase Program. When assembled (with appropriate substitution of parameters) they form the binary output code of the Assembly Phase. They must be input in an order which corresponds to the desired order of the object code.

## EXECUTABLE STATEMENTS USING FORMAT NAMES

Most executable instructions will refer to the format names established by the Definition Phase. Their general form is:

---

Form:

---

{label:}format nameΔVFS, VFS

(VFS = Variable Field Substitution)

---

These formats may be referenced singly (with appropriate VFSs) or they may be combined (overlayed) with other formats (and their appropriate VFSs). All cases result in the formation of a single, complete microword.

Executable Instruction Statements must:

- Begin on a new line.
- Contain a format name from the Definition Phase.
- Substitute a constant name, a label, a constant, or an expression for each variable field and these must be separated by commas. If a default value was given in the Definition Phase and is to be used, the VFS may be omitted.

Executable Instruction Statements may:

- Contain a single format name or may contain an unlimited number of format names to be overlayed.
- Contain the current value of the program counter as the value for a field if $ is the VFS used for that field. The $ may be part of an expression ($ + n) given for a VFS.
- Be preceded by a label: or a label::

## FREE FORMAT STATEMENT FF

Executable statements whose instruction formats were not defined in the Definition Phase may be defined in the Assembly Phase by using the built-in free format command FF. The general form is:

---

Form:

---

{ label: }   FFΔ field1, field2, . . ., fieldn

---

An Assembly File may contain an unlimited number of FFs.

Each FF must:

- Begin on a new line.
- Contain a / (slash) as the first nonblank character if continued on another line.
- Have fields separated by commas.
- Have an explicit length "n" given for "don't care" fields (nX) or for fields defined using decimal (nD#m).
- Not contain a variable field.
- Not contain a constant name for a field unless that constant has been **previously** defined in the Assembly or Definition File.
- Not be overlayed with another format name.

Each FF may:

- Be preceded by a label : or label ::
- Contain an expression for any field but the expression must be enclosed in parenthesis and must be preceded by the field length "n", for example:

  FFΔ5X,10($-5),B#101

- Contain a value for an expression which is to be automatically right justified in a field. However, if the number of bits which represent the value is larger than the field length, an error is generated unless the truncation follows the ) for this expression
- Contain a field whose value is the current value of the program counter by using $ for that field (or an expression containing $ may be used).

For example, if the constants

```
WORDΔ  48
AZ:        EQUΔB#01
RB:        EQUΔQ#10
```

were defined in the Definition File, then the Assembly File could contain the following statements:

```
C: EQUΔ H#C
XTRA: FFΔ 12H#3%, AZ, 18X, C, B#10111,
/1X, RB
```

The microinstruction (binary output) for this FF is:

$$\underbrace{000000000011}_{12H\#3\%}\ \underbrace{01}_{AZ}\ \underbrace{XXXXXXXXXXXXXXXXXX}_{18X}$$

$$\underbrace{1100}_{C}\ \ \underbrace{10111}_{B\#10111}\ \ \underbrace{X}_{1X}\ \ \underbrace{001000}_{RB}$$

which will be printed in the following format:

```
00000000001101XX XXXXXXXXXXXXXXXX 110010111X001000
```

## OVERLAYING FORMATS

When formats are overlayed (combined) to form a microword, the general form is:

---

Form:

---

{label:}format nameΔVFS, VFS, &format nameΔVFS, VFS . .

(VFS = Variable Field Substitution) (& = overlay)

---

Formats may be overlayed (combined) with other formats provided that:

- Each bit of format name (#2) that contains a one or zero, must have that bit specified as a "don't care" in the format name (#1) to be overlayed. Subsequent overlays must be on the "don't care" fields remaining after the overlay of all preceding formats.
- Each format is a full microword in length.

Microword instructions defined using the built-in free format (FF) may **not** be overlayed.

For example, if the Definition File contains:

```
ADD: DEFΔ 5X, 8H#A2, 3X
REG1: DEFΔ B#00001, 11X
CARRY: DEFΔ 15X, B#1
```

Then in the Assembly Phase

```
ADRGCY:  ADD & REG1 & CARRY
```

yields

```
00001   10100010   XX1
```

## COMMENT STATEMENTS

Comment statements are nonexecutable statements which are used to provide information about the program variables or the program flow. A comment may be a full line or may follow, for example, a constant definition statement. All characters from the semicolon to the end of the input line are not processed and serve merely as a documentation aid. The general form is:

| Form: |
|---|
| ; comment text desired |

## END

END indicates that the Assembly File is complete and should be processed. The general form is:

| Form: |
|---|
| END |

END must:

- Begin on a new line.
- Be the last statement in the Assembly File.
- Be followed by a carriage return.

## ARGUMENTS

An Argument follows some types of statements as shown in the executable instruction section.

Permissible Arguments are:

Constants
Expressions
Constant names
Labels

The statements

LIST
NOLIST
END
EJECT

require no Arguments.

Executable instructions which contain format names from the Definition File need Arguments only if there were no default values given for variable fields. Arguments which are to be substituted in variable fields are called Variable Field Substitutes (VFS).

All other statements types require Arguments.

## CONSTANTS

Constants are used as Arguments for the commands EQU, ALIGN, RES, SPACE, ORG or as variable field substitutes (VFSs).

Note that in the Assembly File the $ is used to indicate the substitution of the program counter value for the content of a constant or field. The following table lists the designators which may be used to define constants:

| Designator | Meaning |
|---|---|
| B# | A constant or field whose content will be represented using binary digits (0 and 1). |
| Q# | A constant or field whose content will be represented using octal digits (0 through 7). |
| D# | A constant or field whose content will be represented using decimal digits (0 through 9). A D# must be preceded by decimal digit(s) giving an explicit length (number of bits) when representing a field in an FF statement. |
| H# | A constant or field whose content will be represented using hexadecimal digits (0 through 9, A through F). |
| $ | Use the current program counter as the value for this field or constant. |

## CONSTANT LENGTHS

Constant lengths were discussed in detail in Chapter I. However, the length associated with the use of the $ is a special case.

When the $ is detected in the evaluation of a constant field or expression, the current program counter value is substituted in place of the $.

If the PC = 59 at the instruction preceding:

NEXTLOC: EQUΔ$+5

then NEXTLOC is equated to 64.

If the $ is substituted for a field, the length of the PC is calculated by counting the bits from the right to the leftmost significant one bit. The PC length most probably will not agree with the defined (explicit) field length.

Thus, when defining fields in a format in the Definition Phase or in an FF statement, the fields which are to have $ substituted in them should include the % and/or the : attributes. For example, the field definition

4V%:

will permit any PC value to be substituted into it but

4V

will accept only PC values between $0000_2$ and $1111_2$.

## CONSTANT MODIFIERS

Constants may have modifiers following their given value. They must appear after the constant digits where they may be in any order but will be processed in the following order:

| Modifier | Description |
|---|---|
| * or − | Inversion or negation |
| % | Right justification |
| : | Left truncation |
| $ | Paging |

A constant may not be modified by both inversion and negation.

If a constant, including modifiers, is given as a VFS, any attributes (permanent modifiers) given for that field in the Definition File will also modify the value of the constant given.

If, for example the Definition File contains:

A: DEFΔ 5X, $\underbrace{\text{3V*}}_{\text{field\#1}}$, 2X, $\underbrace{\text{5V\%H\#}}_{\text{field\#2}}$, B#10101

and the Assembly File is written:

TEST: AΔ011,9

the binary value 011 is inverted and substituted for field #1, while the 9 (hex) is equated to binary 1001 and right justified for field#2 resulting in the microinstruction

```
XXXXX  100  XX  01001  10101
```

If the Assembly File statement is written

```
TEST2: AΔ001*, 3*
```

the binary value 001 is inverted by the current *, then inverted again by the attribute in the Definition File for field#1. Field#2 hex 3 (binary 0011) is inverted to 1100 and right justified in field#2.

The complete microinstruction is:

```
XXXXX  001  XX  01100  10101
```

## EXPRESSIONS

Expressions may be used when the programmer wishes to have a value calculated as an argument or as a field substitution. An expression assumes the form:

| Form: |
|---|
| Symbol operator symbol operator . . . |

All expressions:

- Are evaluated using integer arithmetic and remainders are discarded
- Must result in a positive value which can be represented in 16 bits ($2^{16}-1$ maximum).
- Use only the operators, + addition, − subtraction, * multiplication, /division, which are described in Chapter II, page 5.
- Are evaluated in strict left to right sequence. There is no hierarchy for the operators and no parenthesis for nesting are permitted.
- May contain the $ as a symbol to indicate that the current value of the program counter is to be substituted.
- Are terminated by a comma or the end of the line except when used as a field in FF where they are enclosed by parenthesis.
- May be continued on the next line by making the first nonblank character a slash (/). A continuation involving a division would thus require a double slash (//).
- May contain constants, constant names or labels.

For example, if SBB is a format name, and the first variable field is to contain the value 3, it might be written as:

```
SBBΔ1 + 2
```

which is the same as SBBΔ3 (1 and 2 are expression symbols, + is an expression operator). The expression

```
JMPΔ$ − 5
```

yields the current value of the program counter minus 5 as the VFS for the first variable field in the format name JMP. ($ and 5 are expression symbols, − is an expression operator). The expression

```
EIGHT: EQUΔ 2*2*2
```

means EIGHT = 8 (2's are the expression symbols, *'s are the operators).

## EXAMPLES OF CORRECT CONSTANT USAGE

```
QREG:EQUΔQ#0
   .
   .
   .
AQ:EQUΔQREG
DQ:EQUΔ4+8/6 (value = 2)
AB:EQUΔQREG+1
AM2901:DEFΔ4V%D#,5X,AQ,3V,17X
```

Definition File

```
EXOR:EQUΔQREG+6
BEGIN:AM2901Δ$+2,EXOR
      AM2901Δ$−1,AB
```

Assembly File

## VARIABLE FIELD SUBSTITUTES (VFS)

When a format is defined in the Definition File some of its fields may be designated as variable fields. If these fields are not given a default value during their definition or if one wishes to override the default value, a substitution must be made for these field(s) in the Assembly File source statements. These substitutes are called Variable Field Substitutes, VFS.

### REQUIRED SUBSTITUTIONS

If the variable field(s) are not given default values in the Definition File, values for these fields **must** be provided in the Assembly File source statements. If omitted, an error message will be provided, and processing of that statement ends.

### SUBSTITUTION SEPARATORS

Each VFS (whether required or optional) represents a single field and must be separated from other VFSs by a comma. Trailing commas may be omitted but the assembler uses the commas to indicate which fields are to be given substitute values (i.e., VFSs are positional and position is determined by the number of commas), so leading or intermediate commas must be given.

For example if the Definition File contains:

$$\text{A:}\quad \text{DEF}\Delta\ \text{5X, }\underbrace{\text{3V*B\#110}}_{\text{field \#1}}\text{, 2X, }\underbrace{\text{5V\%H\#}}_{\text{field\#2}}\text{, B\#10101}$$

Then if the Assembly File is written as

TEST3: AΔ,4

field#1 will assume the default value 001 (from 3V*B#110) while field#2 will be equated to 0100 and right justified in the 5-bit field so that field#2 is 00100.

The complete microinstruction will be

XXXXX 001 XX 00100 10101

If the comma were omitted and

TEST4: AΔ4

were written, the assembler would try to use 4 as the VFS for field#1. Two errors are present. The 4 is not a binary number as required for field #1, and no value is indicated for field #2. Field #2 had no explicit default value, and no VFS is given which is an error. The indicated error would be "illegal character," since the 4 is assumed to go with field #1 which requires binary digits.

If, however, the user wishes to input field#1 as an octal 4 and field#2 as zero, he could write:

TEST5: AΔQ#4,0

which yields the microinstruction

$$\text{XXXXX}\quad \underbrace{011}_{\substack{\text{octal 4}\\ \text{inverted}}}\quad \text{XX}\quad \underbrace{00000}_{\substack{\text{hex 0}\\ \text{right-}\\ \text{justified}}}\quad 10101$$

In short, when forming the microword definition, if a leading or intermediate variable field is to assume a default value but a trailing field requires a VFS, each field to be skipped must be represented by a comma.

This is best explained by an example. Assume a format ADE with three variable fields, each having a default value of zero specified in the Definition File:

ADE: DEFΔ 3VB#000, 3VB#000, 3VB#000

The following example illustrates fields which assume their default values and fields which are given override or substitute values.

| Instruction | Resultant Microword Definition | Meaning |
|---|---|---|
| TEST6: ADEΔ,,010<br>or<br>TEST7: ADEΔ,,Q#2 | 000 000 010<br>000 000 010 | Fields 1 and 2 assume their default values, field 3 contains 010. |
| TEST8:ADEΔQ#4,,B#101 | 100 000 101 | Field 2 assumes its default value, field 1 is 100, field 3 is 101. |
| TEST9: ADEΔ011 | 011 000 000 | Fields 2 and 3 assume their default values, field 1 is 011. |

If the variable field substitutions contain modifiers, using the Definition File statement:

ADE: DEFΔ 3VB#000, 3VB#000, 3VB#000

the Assembly File statements for the previous example could be written:

| Instruction | Resultant Microword Definition | Meaning |
|---|---|---|
| TEST10:ADEΔ,,101* | 000 000 010 | Fields 1 and 2 assume their default values. Field 3 is 101 inverted. |
| TEST11:ADEΔH#4: | 100 000 000 | Field 1 is hex 4 (binary) 0100) truncated to 100. Fields 2 and 3 assume their default values. |

The variable fields may contain attributes in the Definition File such as:

ADE: DEFΔ 3V:H#0 ,3V*B#000, 3V%B#000

The Assembly File Statements written below now generate:

| Instruction | Resultant Microword Definition | Meaning |
|---|---|---|
| TEST12:ADEΔ,,01* | 000 111 010 | Field 1 assumes its default value 000. Field 2 assumes its default value 111. (000 inverted). Field 3 is inverted to 10 then right justified to be 010. |
| TEST13:ADEΔ9, Q#3*,1 | 001 011 001 | Field 1 is hex 9 truncated to 001. Field 2 is octal 3 inverted to 100, then inverted by field#2 attribute (*) to 011. Field 3 is binary 1 right justified to 001. |

## FITTING VARIABLE SUBSTITUTES TO VARIABLE FIELDS

Any value given as a Variable Field Substitute (VFS) must contain exactly the number of bits specified (in the Definition File) for the total length of the variable field unless the modifiers % (right justification), : (truncation), or $ (paged addressing) are given.

These modifiers may be supplied as attributes with the original field definition (Definition File) or they may be supplied with the field substitution value in the Assembly File.

## PAGED AND RELATIVE ADDRESSING

$ is used in two ways in the Assembly File:

a) To indicate that the current value of the program counter is the value to be substituted into this field. This is called relative addressing.
b) As an attribute to indicate that the value substituted for this field must be on the same memory "page" as the microword into which it is substituted. This is called paged addressing.

For relative addressing, the $ alone or as part of an expression is used as a VFS.

For paged addressing, the $ may be given as an attribute of this variable field in the Definition File, or the $ may immediately follow the VFS in the Assembly File source statement.

For example, if the Definition File contains

```
JSR:DEFΔ8X,8V$, H# 27, 12VH#
JSB:DEFΔ8V%D#, 8X, 8Q#013:, 12X
```

the Assembly File could be written

```
Line#

  1    JSR Δ  BEGIN,0BC
  2    JSB Δ  MULT$+5
  3    JSR Δ  MULT, BEGIN$
  4    JSB Δ  H#37
  5    JSB Δ  $+5
        •
        •
        •
       BEGIN:  ADD
        •
        •
        •
       MULT:  MPY
```

Lines 1-3 are examples of $ used for paged addressing. In Line 1, the value of the program counter (where BEGIN: appears) is substituted into the first variable field of the format JSR. This value is truncated on the left, if necessary, to fit into this 8-bit field, and any truncated left bits must be identical to the corresponding bits of the program counter associated with Line 1.

The same type of substitution, truncation, etc. occurs for Lines 2 and 3.

Note that:

- The JSB on line 2 needs a $ after MULT if paged addressing is desired since no $ was given with that variable field in the Definition File.
- For expressions such as line 2, the constant (5) is added to the value of the label (MULT) **before** the check is made to ensure that the value substituted is still on the correct "page".
- The JSR on line 1 needs no $ with the BEGIN since that variable field contained a $ in the Definition File.
- The JSR on line 3 requires a $ after BEGIN since the second variable field did not contain a $ in the Definition File.
- On line 2 a label with a $ may be part of an expression.

Line 5 is an example of relative addressing. The current value of the program counter plus 5 will be substituted for the variable field.

Note that:

- There is no connection between the $ used for paged addressing – as an attribute for a variable field – and the $ used as a variable field substitute to indicate use of the current value of the program counter (relative addressing).

## HEXADECIMAL ATTRIBUTE

The designator H#, if given with a variable field in the Definition File, is a permanent attribute but may need to be repeated in the Assembly File. This is necessary since the program cannot distinguish a hexadecimal value which begins with an A through F from a label or format name.

Thus, if the Definition File contains

```
AM2901:DEFΔ8V%H#,Q#0,21X
```

and the Assembly File statement contains

```
AM2901Δ3A
```

it is clear to the program that the digits 3A are to be substituted into the variable field. (A label or name cannot begin with a numeral).

However, the statement:

```
AM2901ΔAB
```

does not clearly indicate whether the constant name AB is meant, or the value of the hexadecimal digits AB is meant. If the programmer wishes the hex value AB, he must write:

```
AM2901ΔH#AB
```

The statement AM2901ΔAB will substitute the value of the constant named AB in the first variable field. If there is no constant named AB, an error will be generated.

## ASSEMBLER SYMBOL TABLE

The symbol table contains a list of all the symbols (constant names) defined by EQUs and all labels in the Assembly File. The symbol table also includes all the constant names and their associated values defined using EQUs in the Definition File.

For each symbol, the table lists the label and the program counter value of the statement where the label is defined, or if the symbol is a constant name (defined by EQU), it is followed by the value of the constant.

A symbol table is useful when errors occur due to misspelling or the omission of the colon after a label.

A sample symbol table is:

```
SYMBOLS
  A     0001
  S     0023
  X     0000
```

Printing of the Symbol Table is optional and is described in the SYMBOL and NOSYMBOL section of Table 4-1.

## ASSEMBLER ENTRY POINT TABLE

The entry point table contains a list of all the entry point symbols (labels followed by ::) and their associated program counters. These values are useful for mapping PROMs.

Printing of the entry point table is optional and is described in the MAP and NOMAP section of Table 4-1.

## ASSEMBLY FILE – RESERVED WORDS

The following are reserved words used by the assembler program during the Assembly Phase. These words MAY NOT BE USED AS LABELS in the Assembly File statements:

ALIGN
EJECT
END
FF
LIST
NOLIST
ORG
RES
SPACE
TITLE

Format names or constant names from the Definition File.

# CHAPTER IV

# AMDASM 29 OUTPUT, FILENAMES, EXECUTION

## ASSEMBLER OUTPUT

Assembly Phase output includes a choice of one of four types of printed listings.

| Type | Description |
|---|---|
| I | Interleaved format (INTER). One line of source code is printed with the corresponding line of object code printed directly below it. |
| II | Source only format (SRCONLY). Only the Assembly File source statements are printed. |
| III | Object code only format (OBJONLY). Only the Assembly Phase object code is printed. |
| IV | Block format (BLOCK). All lines of source code are printed followed by all lines of the object code. |

Each of these listings contains the location (program) counter associated with each line of source and object code.

A final option is to output the binary object code directly to disc for use as input to the post processing phase. (Disc output is independent of the listing option chosen.) The object code on the disc may then be used, for example, as input to the post processing phase which might punch a paper tape in a format suitable for burning PROMs.

## FILENAMES

Filenames are used to identify unique files on a diskette. They are in two parts, a primary part and a generic part. The general form is:

pppppppp.ggg

where the p's represent from one to eight characters in the primary part and the g's represent from one to three characters in the generic part.

All alphanumerics and special characters except

< > . , ; : = ? * or a blank

may be used for p or g.

In the following section p refers to primary filenames for the Definition File; q refers to primary filenames in the Assembly File. Normally the user will use the same primary name for PHASE1 and PHASE2. Thus, pppppppp will equal qqqqqqqq.

The user may define his own names for p's or q's which are meaningful for this particular application. However, he **must** use the generics listed below in some cases. The MANDATORY generics are underlined. Generics not underlined are defaults and will be assigned or assumed if not specified by the user.

| | | |
|---|---|---|
| pppppppp.DEF | Source input for the Definition File (PHASE1) | |
| pppppppp.TBL | Output from PHASE1 | usually p = q |
| qqqqqqqq.TBL | Input for PHASE2 | |
| qqqqqqqq.SRC | Source input for Assembly File (PHASE2) | |
| pppppppp.P1L | PHASE1 listing output | |
| qqqqqqqq.P2L | PHASE2 listing output | |
| qqqqqqqq.OBJ | PHASE2 output (object code) | |
| qqqqqqqq.MAP | PHASE2 output entry point symbols and their values | |

When creating the input files pppppppp.DEF and qqqqqqqq.SRC the DEF and SRC generics must be typed as a part of the filename when invoking the Editor.

## EXECUTION

NOTE: In examples of execution commands, data to be input by the user is underlined. Other data is output by the system.

After the user has created his Definition File and Assembly File using the AMDOS 29 Editor, he is ready to execute AMDASM 29. After the AMDOS 29 operating system has issued a user prompt (i.e., the characters A>) the microassembler is executed by entering the command:

```
A > AMDASMΔPHASEn=primaryfilename{Δoptions}   cr
```

where

PHASE1=primary filename
or
PHASE1Δprimary filename

specifies execution of the Definition Phase using primary filename for the definition source file.

PHASE2=primary filename
or
PHASE2Δprimary filename

specifies execution of the Assembly Phase using primary filename as the assembly source file.

PHASE1=primaryfilenameΔPHASE2=primaryfilename

specifies execution of both the Definition and Assembly Phases.

Thus,

```
A > AMDASMΔPHASE1ΔB:KIT   cr
```

specifies execution of only the Definition Phase using the file (on drive B) called KIT.DEF.

or

```
A>AMDASMΔPHASE1=B:KITΔPHASE2=B:KIT   cr
```

specifies execution of the Definition and Assembly Phases using the files (on drive B) KIT.DEF as the definition source file and KIT.SRC as the assembly source file.

Either PHASE1 or PHASE2 or both **must** be specified following AMDASMΔ. **P1 and P2 are the alternate abbreviated keywords** used for PHASE1 and PHASE2, respectively.

The generic part of the filename **must** not be typed, and either a Δ or an = may be used before the primary filename as a delimiter. For example, the following are permissible execution commands for PHASE1:

```
AMDASMΔP1=pppppppp          } This assumes pppppppp.DEF
AMDASMΔPHASE1=pppppppp      } was the name assigned
AMDASMΔP1Δpppppppp          } when the Definition
AMDASMΔPHASE1Δpppppppp      } File was created.
```

Following AMDASMΔP1Δprimary filenameΔP2Δprimary filename the user then enters the desired options. Options may be given in any order. They are listed in Table 4-1. The full option may be typed (OBJECT) or the abbreviated option may be typed (O).

If an option is not typed, AMDASM uses the default option given in Table 4-1.

## Table 4-1 AMDASM 29 Options

| OPTION | ABBREVIATED OPTION | DEFAULT | MEANING |
|---|---|---|---|
| DEFTBLΔfilename<br>or<br>DEFTBL=filename | D | pppppppp.TBL<br>or<br>qqqqqqqq.TBL | Specifies the name of the file where output of the Definition Phase is to be stored. When only PHASE2 is executed, this specifies the input file which contains the processed definitions. If no DEFTBLΔfilename is given the default name pppppppp.TBL will be used if PHASE1 is executed; qqqqqqqq.TBL is the default when only PHASE2 is executed. |
| LIST1Δfilename<br>or<br>LIST1=filename | L1 | pppppppp.P1L | Specifies where the Definition output is to go. When LST: is given as the filename, the output will be listed on the line printer. If no list1Δfilename is given, the output goes to the file with the default name pppppppp.P1L. |
| LIST2Δfilename<br>or<br>LIST2=filename | L2 | qqqqqqqq.P2L | Same as LIST1 except this specifies where the PHASE2 (Assembly) output is to go. The default name is the generic P2L appended to the Assembly File source input name (qqqqqqqq.P2L). |
| NOLIST | NL | pppppppp.P1L<br>and/or<br>qqqqqqqq.P2L | Suppresses listing of PHASE1 and/or PHASE2 output. If not specified defaults to LIST1 and LIST2. Output goes to files pppppppp.P1L and qqqqqqqq.P2L. |
| OBJECTΔfilename<br>or<br>OBJECT=filename | O | qqqqqqqq.OBJ | Specifies that the microcode (object code) is to be output on a file with the name (filename). If not given, the microcode is placed on a file with the default name qqqqqqqq.OBJ. |
| NOOBJECT | NO | | Suppresses placement of the microcode onto a file. If block format printing is requested, the object code printing is also suppressed. If not specified defaults to OBJECT and the microcode goes to file qqqqqqqq.OBJ. |
| INTER | IL | BLOCK | Specifies interleaved listing format (a line of source code followed by a line of object code.) |
| BLOCK | BL | | Specifies blocked listing format (all lines of source code, then all lines of object code). |
| SRCONLY | SO | | Specifies source-only listing format (prints only the source code.) |
| OBJONLY | OB | | Specifies object-only listing format (prints only the object code.) |
| WIDTHΔn<br>or<br>WIDTH=n | W | n=80 | Specifies width n, (a decimal number) of characters for listing device. Default is 80. |
| LINESΔn<br>or<br>LINES=n | LN | n=66 | Specifies width n, (a decimal number) of lines per page. If not specified, default is 66 lines (11 inches). |
| MAPΔfilename<br>or<br>MAP=filename | M | qqqqqqqq.MAP | Specifies listing of entry point symbols (i.e., label symbols designated as entry points by double colons "::") and their associated program counter values is to be output on the list device or onto a list file. |
| NOMAP | NM | | Suppresses listing of entry point symbols. If not specified, defaults to MAP and results are stored on a file with the default name qqqqqqqq.MAP. |
| HEX | H | HEX | Specifies listing of location counter in hexadecimal format. |
| OCTAL | Q | | Specifies listing of location counter in octal format. If not specified defaults to HEX. |
| SYMBOL | S | SYMBOL | Specifies listing of constant names and labels and their associated values. |
| NOSYMBOL | NS | | Suppresses listing of Symbol table. If not specified, defaults to SYMBOL. |

## DISK DRIVE DESIGNATORS

Since the AMDASM program is always loaded from the current drive, the user must precede his filenames with a drive designator if his input or output files are not on the current drive.

Thus the general form of all filenames will be

device: primary.generic

where device: is indicated by a A: or B:. A indicates drive A; B indicates drive B.

Examples assume all files are on the current drive. However, when a drive is designated with an input filename, all output default files will be placed on the same drive as the input file for the associated PHASE.

When the user specifies a filename but no drive designator, the file(s) will be placed on the current drive.

## EXAMPLES OF AMDASM EXECUTION

Options need to be separated by at least one blank character from other options in the execution command.

Whenever a user does not specify an option in his execution command AMDASM will use the default values given in the Table 4-1.

The command language for executing AMDASM is best illustrated with examples (current drive is assumed to be drive A) :

```
A > AMDASMΔP1=2900ΔP2=2900   cr
```

specifies execution of both PHASE1 and PHASE2 using 2900.DEF as the input file for PHASE1 and 2900.SRC for PHASE2. Defaults are selected for all other options.

```
A > AMDASMΔP1=2900ΔD=2900R1   cr
```

specifies execution of PHASE1 with 2900.DEF as the input source file and 2900R1.TBL as the definition table output file.

```
A > AMDASMΔP2=SYSTEM1ΔD=2900R1ΔILΔNS   cr
```

specifies execution of PHASE2 with SYSTEM1. SRC as the input source file and 2900R1.TBL as the definition table input file, interleaved listing format, no symbol table listing, and a list of entry point symbols (by default).

The primary default name for the DEFTBL option may assume the PHASE1 (ppppppp) filename or the PHASE2 (qqqqqqq) filename as illustrated in Table 4-1. Thus, if the execution command is:

```
A > AMDASMΔP1ΔAM2900   cr
```

this assumes the input filename is AM2900.DEF and the program will assign the name AM2900.TBL to the definition table output and AM2900.P1L to the output list file.

Now if the user attempts to execute

```
A > AMDASMΔP2ΔSYSTEM1   cr
```

the program will indicate an error since it will be looking for SYSTEM1.TBL as the filename for the DEFTBL input.

The user may, prior to executing the above command, rename his AM2900. TBL file to be SYSTEM1.TBL. Alternatively, he may execute the command

```
A > AMDASMΔP2ΔSYSTEM1ΔDΔAM2900   cr
```

indicating the name AM2900.TBL is the DEFTBL input filename.

In either case, PHASE2 will output files with the default names (including generics):

| | |
|---|---|
| SYSTEM1.OBJ | object code generated |
| SYSTEM1.P2L | PHASE2 listing |
| SYSTEM1.MAP | Mapping PROM file (entry point symbols and their values) |

The user may assign only a primary filename to the DEFTBL option.

All other options may be given a primary or a primary and generic filename if the default option is not used.

## SUBMIT FILES

If the user wishes to have AMDOS 29 automatically execute his AMDASM command, he may create a SUBMIT File as follows:

```
A > EDΔname.SUB   cr
NEW FILE
*I   cr
AMDASMΔP1=$1ΔP2=$2   cr
Control Z
*E   cr
```

SUBMIT files assume the "name.SUB" file is on the current drive, thus it must be created on the diskette which contains AMDASM and this diskette must be mounted on the current drive.

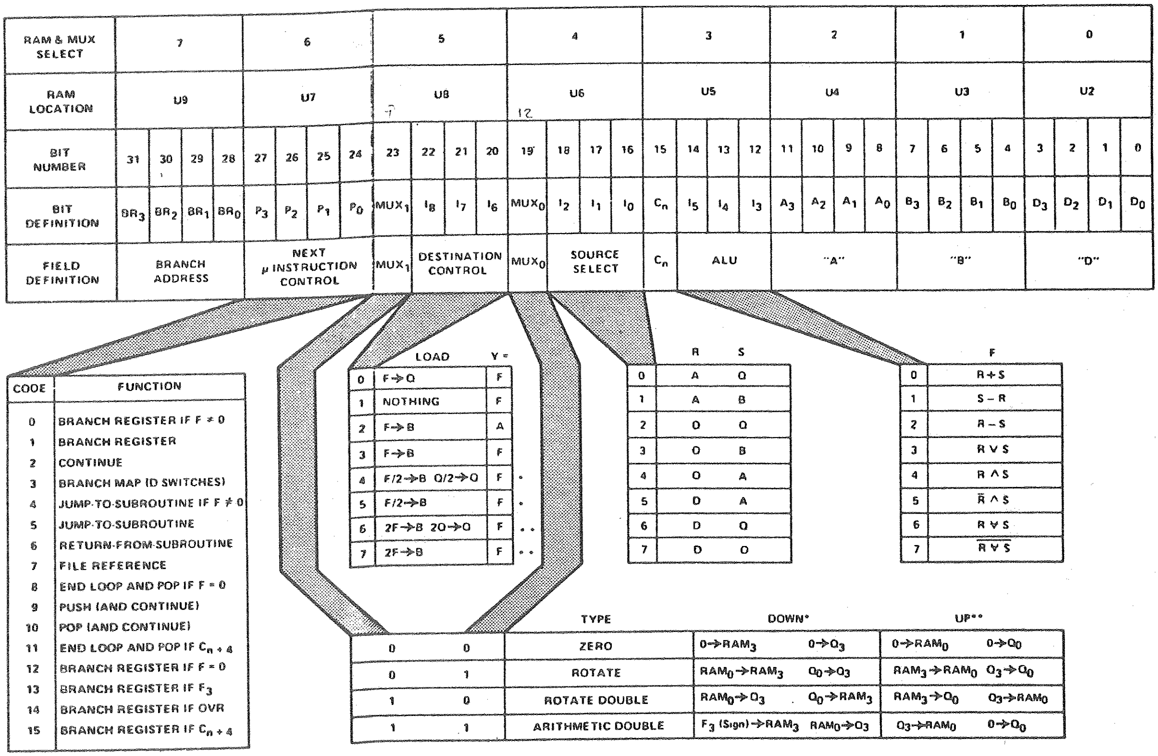
For execution of the above SUBMIT file, the user need merely type:

```
A > SUBMITΔnameΔpppppppΔqqqqqqq
```

AMDOS 29 automatically substitutes ppppppp for $1, qqqqqqq for $2.

SUBMIT files are similar to batch jobs since more than one execution command may be part of the SUBMIT file. Thus, the user may create a SUBMIT file for one or multiple jobs and need not remain at the console.

This is most convenient when the user has a long execution command and/or when he wishes to execute several consecutive assemblies without staying at the console and/or when he wishes to execute the same type of command using many different files. For more detailed information about SUBMIT files, please refer to the System 29 Manuals.



| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RAM & MUX SELECT | 7 | | | | 6 | | | | 5 | | | | 4 | | | | 3 | | | | 2 | | | | 1 | | | | 0 | | | |
| RAM LOCATION | U9 | | | | U7 | | | | U8 | | | | U6 | | | | U5 | | | | U4 | | | | U3 | | | | U2 | | | |
| BIT NUMBER | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BIT DEFINITION | $BR_3$ | $BR_2$ | $BR_1$ | $BR_0$ | $P_3$ | $P_2$ | $P_1$ | $P_0$ | $MUX_1$ | $I_8$ | $I_7$ | $I_6$ | $MUX_0$ | $I_2$ | $I_1$ | $I_0$ | $C_n$ | $I_5$ | $I_4$ | $I_3$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| FIELD DEFINITION | BRANCH ADDRESS | | | | NEXT μ INSTRUCTION CONTROL | | | | $MUX_1$ | DESTINATION CONTROL | | | $MUX_0$ | SOURCE SELECT | | | $C_n$ | ALU | | | "A" | | | | "B" | | | | "D" | | | |

| CODE | FUNCTION |
|---|---|
| 0 | BRANCH REGISTER IF F ≠ 0 |
| 1 | BRANCH REGISTER |
| 2 | CONTINUE |
| 3 | BRANCH MAP (D SWITCHES) |
| 4 | JUMP-TO-SUBROUTINE IF F ≠ 0 |
| 5 | JUMP-TO-SUBROUTINE |
| 6 | RETURN-FROM-SUBROUTINE |
| 7 | FILE REFERENCE |
| 8 | END LOOP AND POP IF F = 0 |
| 9 | PUSH (AND CONTINUE) |
| 10 | POP (AND CONTINUE) |
| 11 | END LOOP AND POP IF $C_{n+4}$ |
| 12 | BRANCH REGISTER IF F = 0 |
| 13 | BRANCH REGISTER IF $F_3$ |
| 14 | BRANCH REGISTER IF OVR |
| 15 | BRANCH REGISTER IF $C_{n+4}$ |

| | LOAD | Y = | |
|---|---|---|---|
| 0 | F→Q | F | |
| 1 | NOTHING | F | |
| 2 | F→B | A | |
| 3 | F→B | F | |
| 4 | F/2→B Q/2→Q | F | * |
| 5 | F/2→B | F | * |
| 6 | 2F→B 2Q→Q | F | ** |
| 7 | 2F→B | F | ** |

| | R | S |
|---|---|---|
| 0 | A | Q |
| 1 | A | B |
| 2 | O | Q |
| 3 | O | B |
| 4 | O | A |
| 5 | D | A |
| 6 | D | Q |
| 7 | D | O |

| | F |
|---|---|
| 0 | R + S |
| 1 | S − R |
| 2 | R − S |
| 3 | $R \vee S$ |
| 4 | $R \wedge S$ |
| 5 | $\bar{R} \wedge S$ |
| 6 | $R \forall S$ |
| 7 | $\overline{R \forall S}$ |

| | | TYPE | DOWN* | | UP** | |
|---|---|---|---|---|---|---|
| 0 | 0 | ZERO | $0 \rightarrow RAM_3$ | $0 \rightarrow Q_3$ | $0 \rightarrow RAM_0$ | $0 \rightarrow Q_0$ |
| 0 | 1 | ROTATE | $RAM_0 \rightarrow RAM_3$ | $Q_0 \rightarrow Q_3$ | $RAM_3 \rightarrow RAM_0$ | $Q_3 \rightarrow Q_0$ |
| 1 | 0 | ROTATE DOUBLE | $RAM_0 \rightarrow Q_3$ | $Q_0 \rightarrow RAM_3$ | $RAM_3 \rightarrow Q_0$ | $Q_3 \rightarrow RAM_0$ |
| 1 | 1 | ARITHMETIC DOUBLE | $F_3$ (Sign) $\rightarrow RAM_3$ | $RAM_0 \rightarrow Q_3$ | $Q_3 \rightarrow RAM_0$ | $0 \rightarrow Q_0$ |

Figure 5-2. Example of Fields and Functions

```
TITLE AM2900 KIT DEFINITIONS

;
WORD 32
;REGISTER DEFINITIONS
;
R0:  EQU H#0
R1:  EQU H#1
R2:  EQU H#2
R3:  EQU H#3
R4:  EQU H#4
R5:  EQU H#5
R6:  EQU H#6
R7:  EQU H#7
R8:  EQU H#8
R9:  EQU H#9
R10: EQU H#A
R11: EQU H#B
R12: EQU H#C
R13: EQU H#D
R14: EQU H#E
R15: EQU H#F
```

"R0" to "R15" are set to hex 0 to F using the "equate" statement. The "H#" designator means the numbers following are in hex, and each digit represents 4 bits.

```
;
;AM2901 SOURCE OPERANDS (R S)
;
AQ: EQU Q#0
AB: EQU Q#1
ZQ: EQU Q#2
ZB: EQU Q#3
ZA: EQU Q#4
DA: EQU Q#5
DQ: EQU Q#6
DZ: EQU Q#7
```

ALU Source operands are assigned octal values using the "equate". The "Q#" designates octal, 3 bits, per digit.

```
;
;AM2901 ALU FUNCTIONS (R FUNCTION S)
;
ADD:    EQU Q#0
SUBR:   EQU Q#1
SUBS:   EQU Q#2
OR:     EQU Q#3
AND:    EQU Q#4
NOTRS:  EQU Q#5
EXOR:   EQU Q#6
EXNOR:  EQU Q#7
```

The 8 ALU functions of the AM2901 are given names.

```
;
;AM2901 DESTINATION CONTROL
;
QREG:   EQU Q#0
NOP     EQU Q#1
RAMA:   EQU Q#2
RAMF:   EQU Q#3
RAMQD:  EQU Q#4
RAMD:   EQU Q#5
RAMQU:  EQU Q#6
RAMU:   EQU Q#7
;
;SHIFT MATRIX CONTROL
;
SHIFT:   DEF 8X,B#0,3X,B#0,19X
ROTATE:  DEF 8X,B#0,3X,B#1,19X
DBLROT:  DEF 8X,B#1,3X,B#0,19X
ARITH:   DEF 8X,B#1,3X,B#1,19X
```

Defines the two separated bits which control the left-right shift multiplexers. The "x's" are "don't-care" bits in between the defined bits.

```
;
NEXT MICROINSTRUCTION ADDRESS SELECT
;
BRFNO:      EQU H#0 ;BRANCH REGISTER IF F NOT EQUAL TO ZERO
BR:         EQU H#1 ;BRANCH REGISTER
CONT:       EQU H#2 ;CONTINUE
BM:         EQU H#3 ;BRANCH MAP
JSRFNO:     EQU H#4 ;JUMP-TO-SUBROUTINE IF F NOT EQUAL TO ZERO
JSR:        EQU H#5 ;JUMP-TO-SUBROUTINE
RTS:        EQU H#6 ;RETURN FROM SUBROUTINE
STKREF:     EQU H#7 ;FILE REFERENCE
LOOPFNO:    EQU H#8 ;END LOOP AND POP IF F=0
PUSH:       EQU H#9 ;PUSH AND CONTINUE
POP:        EQU H#A ;POP AND CONTINUE
LOOPCOUT:   EQU H#B ;END LOOP AND POP IF CN+4
BRFEQ0:     EQU H#C ;BRANCH REGISTER IF F=0
BRF3:       EQU H#D ;BRANCH REGISTER IF F3
BROVR:      EQU H#E ;BRANCH REGISTER IF OVR
BRCOUT:     EQU H#F ;BRANCH REGISTER IF CN+4
```

Definitions for the sequence control instructions used in the second field of the microinstruction.

```
;
;OTHER STUFF
;
CN0:  EQU B#0
CN1:  EQU B#1
LOW:  EQU B#0
HIGH: EQU B#1
ZERO: EQU B#0
ONE:  EQU B#1
;
AM2901: DEF 9X,3VQ#1,1X,3VX,1VX,3VX,4VX,4VX,4X
AM2909: DEF 4VX,4VH#2,24X
DIN:    DEF 28X,4VH#
;
END
```

Format definitions are made for the ALU fields, the sequence control fields, and the data input. Formats contain don't cares (x) and variables (v). Each variable can have a default value. For example, in AM2909, the second four-bit variable defaults to hex 2, and the first four-bit variable defaults to x.

Figure 5-3. Definition File

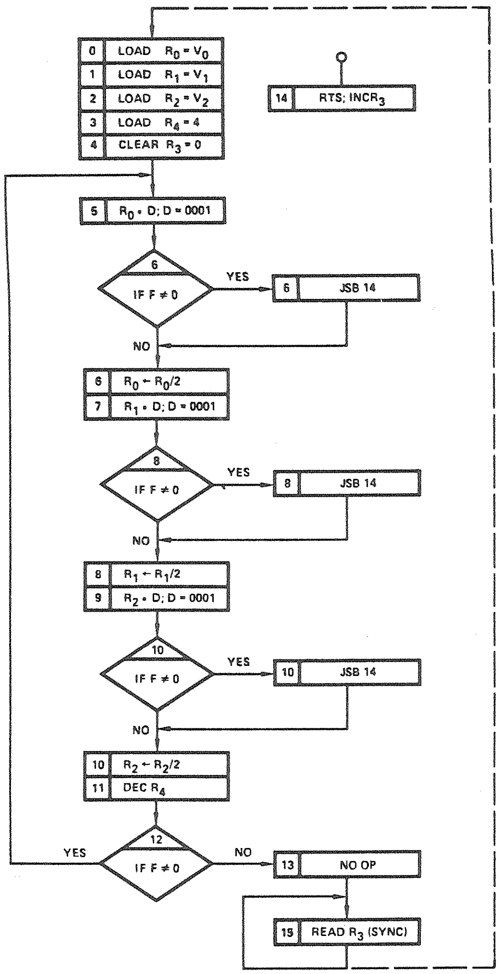


Figure 5-4. Flow Chart of Example

```
0000      AM2909 & AM2901   RAMF, DZ,,OR,,R0 & DIN H#F
0001      AM2909 & AM2901   RAMF, DZ,,OR,,R1 & DIN 9
0002      AM2909 & AM2901   RAMF, DZ,,OR,,R2 & DIN 0
0003      AM2909 & AM2901   RAMF, DZ,,OR,,R4 & DIN 4
0004      AM2909 & AM2901   RAMF, ZB,,AND,,R3
0005 A5:  AM2909 & AM2901    ,DA,,AND,R0,R0 & DIN 1
0006      AM2909 A14,JSRFN0 & AM2901  RAMD, ZB,,OR,,R0
0007      AM2909 & AM2901   ,DA,,AND,R1,R1 & DIN 1
0008      AM2909 A14,JSRFN0 & AM2901  RAMD, ZB,,OR,,R1
0009      AM2909 & AM2901   ,DA,,AND,R2,R2  & DIN 1
000A      AM2909 A14,JSRFN0 & AM2901  RAMD, ZB,,OR,,R2
000B      AM2909 & AM2901  RAMF, ZB, CN0,SUBR,,R4
000C      AM2909 A5,BRFN0 & AM2901
000D      AM2909 A15,BR & AM2901
000E A14: AM2909 ,RTS & AM2901  RAMF, ZB,CN1,ADD,,R3
000F A15: AM2909 A15,BR & AM2901   ,ZB,,OR,,R3
     END

0000 XXXX0010X011X111 X011XXXX00001111
0001 XXXX0010X011X111 X011XXXX00011001
0002 XXXX0010X011X111 X011XXXX00100000
0003 XXXX0010X011X111 X011XXXX01000100
0004 XXXX0010X011X011 X100XXXX0011XXXX
0005 XXXX0010X001X101 X100000000000001
0006 11100100X101X011 X011XXXX0000XXXX
0007 XXXX0010X001X101 X100000100010001
0008 11100100X101X011 X011XXXX0001XXXX
0009 XXXX0010X001X101 X100001000100001
000A 11100100X101X011 X011XXXX0010XXXX
000B XXXX0010X011X011 0001XXXX0100XXXX
000C 01010000X001XXXX XXXXXXXXXXXXXXXX
000D 11110001X001XXXX XXXXXXXXXXXXXXXX
000E XXXX0110X011X011 1000XXXX0011XXXX
000F 11110001X001X011 X011XXXX0011XXXX
```

Figure 5-5. Assembly Output in Block Format

# CHAPTER VI

## AMMAP 29 MAPPING RAM/PROM DATA ASSEMBLER

### AMMAP DESCRIPTION

AMMAP enables System 29 to generate non-microinstruction PROM data. Specifically, AMMAP generates non-microinstruction PROM data for the Mapping RAM in the Computer Control Unit (CCU) card of System 29.

AMDASM 29 outputs a symbol table file of microprogram entry point symbols as an option with the generic file name 'MAP'. The AMMAP assembler uses this file, in conjunction with an assembly source file provided by the user, as a symbol table to generate an object file. The object file, which uses the generic file name OBM, is compatible with the AMDASM 29 object file format. Therefore, it can be loaded/verified by the LBPM/VBPM programs.

AMMAP is a one-pass assembler that allows the user to specify the width of the mapping PROM, the assembler's location counter value, and the microprogram entry point addresses to be assembled into any PROM location.

### MAJOR FUNCTIONS OF AMMAP

The principal function of the AMMAP assembler is to generate Mapping PROM data through a symbolic source program. When AMMAP is called for execution, the user must specify the 'MAP' file to be used for symbol table input. AMMAP builds a symbol table from this file and begins assembly of PROM data.

The individual functions of AMMAP are:

- Entry Point Symbol Table Management – AMMAP will manage and utilize the entry point symbol table built from the user specified 'MAP' file.
- Location Counter Control – AMMAP starts assembly at PROM location 0 unless specified otherwise via user directives that set the location counter value. In addition, it keeps track of locations and assigns locations for each entry point address assembled.
- Data Assembly – Translates symbolic entry point addresses into internal binary equivalents and assembles them into PROM location.
- Assembly Directive Processing – Processes all assembly directives: PROM width specification, number base specification for setting location counter, assembly listing, and object output control, and END directive.
- Assembler Output Generation – Generates an assembly listing, object data output file, and error diagnostics.
- User Command Language Interface – Processes user-specified assembler execution parameters and other user interfaces.

### AMMAP PERFORMANCE CHARACTERISTICS

AMMAP runs under the 32K memory configuration for System 29. It allows at least 8K for entry point symbol table space and can handle more than 600 entry point symbols.

## USER INTERFACE

### PROGRAM AND SOURCE STATEMENT CONCEPTS

The general format of an assembly statement in AMMAP is:

location: entry0, entry1, . . . ., entryn

where:

| | |
|---|---|
| location | is a binary, octal, decimal, or hex constant. The number base is selectable via the BASE directive and default base is hexadecimal. |
| entryn | is an entry point symbol that is defined during AMDASM assembly phase and entered into the symbol table written out as the 'MAP' file. It may also be an absolute address in which case it must be a constant which follows AMDASM syntax rules. |

**NOTE:**

location and colon following it are optional. If not present, AMMAP assigns the next available location. Assembly origin is 0, unless specified otherwise.

### Comment Statements

A comment may be introduced into any source line by preceding the comment with a semi-colon (;). AMMAP will treat all source text on a line after a semi-colon as a comment up to the carriage return.

### ASSEMBLER DIRECTIVES

### PROM Width Directive (WIDTH)

The general format of WIDTH directive is:

WIDTH n

where: n is a decimal constant (which specified the width of Mapping PROM or RAM 1_n_128)

The WIDTH directive must precede any assembly statement because it specifies the width of Mapping PROM or RAM.

### Title Directive (TITLE)

The general format of TITLE directive is:

TITLE text

where: text is a title string of up to 60 characters.

The title will appear in the page header of assembly listings and the title record for object file.

### Location Counter Base Directive (BASE)

The general format of the BASE directive is:

BASE Type

where: type may be one of the following: 2, 8, 10, or 16 to designate that binary octal, decimal, or hex numbers will be used for specifying PROM location.

If a number base is not specified in the program, the default used is 16 (hexadecimal).

## End of Program Directive (END)

The general format of END directive is:

END

The END directive must be used to terminate the AMMAP assembly source input file.

**NOTE:**

Use of TAB characters also allowed as in AMDASM.

## COMMAND LANGUAGE

The AMMAP assembler may be executed with the following AMDOS 29 transient command:

AMMAP filename1 MAP = filename2 options cr

where:

| | |
|---|---|
| filename1 | is the primary filename of the AMMAP source input file which must have the generic file name '.OPC'. |
| filename2 | is the primary file name of the '.MAP' output file from AMDASM to be used as the entry point symbol table. |
| options | are user selectable options described in Table 6−1. |

TABLE 6-1 AMMAP 29 OPTIONS.

| OPTION | ABREVIATED OPTION | DEFAULT | MEANING |
|---|---|---|---|
| LISTΔfilename<br>or<br>LIST=filename | L | | Specifies the listing is to be output to a file with the name (filename). If not given the listing is placed on a file with the default name pppppp.P4L. |
| NOLIST | NL | pppppp.P4L | Suppresses the creation of a listing. If not specified defaults to L=pppppp.P4L. |
| OBJECTΔfilename<br>or<br>OBJECT=filename | O | | Specifies that the microcode (object code) is to be output on a file with the name (filename). If not given, the microcode is placed on a file with the default name qqqqqqqq.OBM. |
| NOOBJECT | NO | qqqqqqqq.OBM | Suppresses placement of the microcode onto a file. If block format printing is requested, the object code printing is also suppressed. If not specified defaults to OBJECT and the microcode goes to file qqqqqqqq.OBJ. |
| WIDTHΔn<br>or<br>WIDTH=n | W | n=80 | Specifies width of n (a decimal number) characters for listing devices. Default is 80. |
| LINESΔn<br>or<br>LINES=n | LN | n=66 | Specifies length of n, (a decimal number) lines per page. If not specified, default is 66 lines (11 inches). |
| HEX | H | | Specifies listing of location counter in hexadecimal format. |
| OCTAL | O | HEX | Specifies listing of location counter in octal format. If not specified defaults to HEX. |
| SYMBOL | S | SYMBOL | Specifies listing of constant names and lables and their associated values. |
| NOSYMBOL | NS | | Suppresses listing of Symbol table. If not specified, defaults to SYMBOL. |

TABLE 6-2 AMMAP ERROR MESSAGES.

| ERROR | MEANING |
|---|---|
| ERROR 1 | Illegai Character |
| ERROR 2 | Undefined Symbol |
| ERROR 3 | Illegal Location Counter Value |
| ERROR 4 | Missing Colon After Location Counter Value |
| ERROR 5 | Missing Delimiter After PROM Data Specification |
| ERROR 6 | Missing End Statement |
| FATAL ERRORS: | |
| ERROR 100 | Command Option Syntax Error |
| ERROR 101 | Illegal Mapping PROM Width Specification |

# CHAPTER VII

# AMSCRM 29 BIT SCRAMBLING POST PROCESSOR

## AMSCRM 29 DESCRIPTION

It is sometimes convenient for the microprogrammer to assign microword fields such that they initially occupy positions that differ from those in the actual hardware implementation. This is often the case when the programmer, for convenience, allocates bits according to the functions to be performed and then needs to translate the object code produced by AMDASM to be consistent with the hardware microprogram memory design.

There is another instance where the ability to shift bit assignments is important to the engineer. As a given product evolves, bits may be added or deleted from the original microword format. At the time that PROMs need to be blown, bits often need to be reassigned to be consistent with the hardware implementation.

At the conclusion of an AMDASM assembly, the user can direct AMSCRM to reassign the bit positions of the microword contents by simply specifying the source and destination bit positions and the length of each field to be moved. In so doing, a reorganized microcode object file is produced.

The leftmost bit in the object code is assumed to be position 0; thus the rightmost bit position will be (microword size-1). **This is the reverse of the numbering used in Figure 5-2.**

AMSCRM is executed after AMDASM but before AMPROM. The object code generated by AMDASM is the input to AMSCRM.

After execution begins, the transformation parameters are entered. These indicate the source bits to be moved, their destinations and the length of the field to be moved.

After execution of AMSCRM the microcode is in its new bit order and is available on a file to be used as input to AMPROM.

## EXECUTION AND FILENAMES FOR AMSCRM 29

After the AMDOS 29 operating system has issued a user prompt (i.e., the characters A >), AMSCRM is executed by entering a command of the form:

```
A > AMSCRMΔOLD=filename1ΔNEW=filename2   cr
```

or

```
A > AMSCRMΔOLDΔfilename1ΔNEWΔfilename2   cr
```

Filename1 is the name given to the file containing the microcode generated by AMDASM. Filename1 will be the assigned name qqqqqqqq.OBJ if AMDASM was executed without specifying OBJECT=filename.

Filename2 is a user-defined name for the file on which the reordered microcode is to be placed. It is recommended that the user make the primary part of Filename2 the same as Filename1, but that he use a different generic. Filename2 must be different from Filename1. There are no required generics for AMSCRM, but if Filename1 does not specify a generic, the generic defaults to .OBJ. Likewise, the default generic for Filename2 is .XOB.

After the execution command and a carriage return is entered, AMSCRM issues a prompt:

```
ENTER TRANSFORMATION PARAMETERS:

S0, D0, W0,  cr

S1, D1, W1,  cr

Sn, Dn, Wn,  cr

.  cr
```

The user enters the underlined data where:

S0 = starting (leftmost) bit position for the first source field to be moved

D0 = destination bit position for the first (leftmost) bit of the first group of bits.

W0 = width of the field to be moved.

S1 = starting (leftmost) bit position for second source field to be moved.

●

●

●

Wn = width of the last field to be moved.

Each group of parameters is ended by a carriage return. **A period and a carriage return are used to terminate input.**

For all microwords the leftmost bit position of the AMDASM printout is considered to be zero; thus the rightmost bit position will be the width of the microword -1.

It is the user's responsibility to see that all bits are properly shifted. Thus, if the user enters:

```
14,28,4  cr
```

(indicating that 4 bits beginning at bit position 14 are to be moved to bit positions 28, 29, 30, 31), he also must enter

```
28,X,4  cr
```

where X indicates the new starting bit position for the bits originally in positions 28-31.

## AMSCRM 29 EXAMPLE

As an illustration, the MUX control bits in the Evaluation Kit are physically separated in the hardware configuration. However, it would be much more convenient to program them as contiguous bits when writing the microcode.

The bit **numbers** shown in Figure 5-2 are numbered right to left; AMDASM and AMSCRM count bit **positions** from left to right.

Thus, if the MUX control bits were assigned to the bit **positions** 8 and 9 (bit **numbers** 23 & 22 in Figure 5-2) during AMDASM, then AMSCRM would require the following command to put them into the positions shown in Figure 5-2. The AMDASM output is assumed to be on the file SYSTEM1.OBJ. SYSTEM1.XOB is the name to be assigned to the AMSCRM output.

```
A > AMSCRMΔOLD=SYSTEM1ΔNEW=SYSTEM1   cr

ENTER TRANSFORMATION PARAMETERS:


9,12,1  cr

10,9,3  cr

.  cr
```

# CHAPTER VIII

## AMPROM 29 PROM PROGRAMMER POST PROCESSOR

### AMPROM DESCRIPTION

When a user has completed an AMDASM assembly and an optional AMSCRM execution, he may wish to output his binary object code in a form which corresponds with his PROM's organization and/or he may wish to punch the object code from his program onto paper tapes to be used as input to a PROM burner.

In order to understand post processing one must know how the PROMs are organized in the computer memory space.

### PROM ORGANIZATION

If, as an example, AMDASM has been executed using the command

```
A > AMDASMΔP1=2900ΔP2=2900    cr
```

AMDASM generates binary object code for the executable statements in the file named 2900.SRC.

This binary object code is output to a file called 2900.OBJ.

For our example we shall assume that the microword is 48 bits wide and the number of executable statements is 1024.

This gives us a matrix 48 wide by 1024 deep as shown in Figure 8-1.

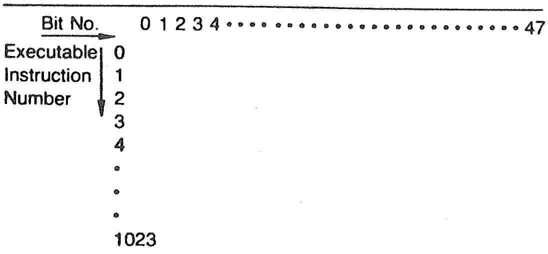


Figure 8-1. Bit Matrix

After PROM width and depth are specified, the Bit Matrix is subdivided to yield a PROM MAP where each PROM is n bits wide by m bits deep. If we assume that the initial program counter is zero for our example, the actual PROM MAP printed might appear as shown in Figure 8-2.

| | PC | C1 | C2 | C3 | C4 | C5 | C6 | C7 | |
|---|---|---|---|---|---|---|---|---|---|
| R1 | 0000 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | PROM No. |
| R2 | 0100 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |
| R3 | 0300 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | |
| R4 | 0380 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | |

where

PC represents the initial program counter value for that PROM row. The PC value is given in hexadecimal.

Figure 8-2. Sample PROM MAP

For the example, PROMs shall be organized as shown in Figure 8-3.

Each executable instruction naturally has a program counter associated with it by virtue of its position in the program and/or the origin(s) that were set during the assembly execution.

This breakup of the matrix is now called a PROM map which has associated with it, not only the PROMs shown, but rows and columns as shown in Figure 8-3. Thus, we may now refer to PROM 19 by using the digits 19, or by referencing R3 for Row 3 or C5 for Column 5.

As shown in Figure 8-4, all PROMs in Row 1 are 256 (instructions) deep. PROMs 1, 3, 5, and 6 are only 4 bits wide, while PROMs 2 and 7 are 8 bits wide and PROM 4 is 16 bits wide.

In Row 2, all PROMs are 512 (instructions) deep and PROMs 8, 10, 12 and 13 are 4 bits wide, PROMs 9 and 14 are 8 bits wide and PROM 11 is 16 bits wide.

Rows 3 and 4 are each 128 (instructions) deep; PROMs 15,22,17,24,19,26,20 and 27 are 4 bits wide; PROMs 16,23,21,28 are 8 bits wide; and PROMs 18 and 25 are 16 bits wide.

If the user requests printing (or punching) of PROM #1 he will obtain data that is 4 by 256.

If the user requests printing of Row 3, he will obtain data (i.e., the contents of PROMs 15 through 21) in the following form:

4 x 128, 8 x 128, 4 x 128, 16 x 128, 4 x 128, 4 x 128, 8 x 128

If the user requests printing of Column 4 he will obtain data (i.e., the contents of PROMs 4, 11, 18, and 25) that is:

16 x 256, 16 x 512, 16 x 128, 16 x 128

| Column # → / Row # ↓ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | PROM# 1 | PROM# 2 | PROM# 3 | PROM# 4 | PROM# 5 | PROM# 6 | PROM# 7 |
| 2 | PROM# 8 | PROM# 9 | PROM# 10 | PROM# 11 | PROM# 12 | PROM# 13 | PROM# 14 |
| 3 | PROM# 15 | PROM# 16 | PROM# 17 | PROM# 18 | PROM# 19 | PROM# 20 | PROM# 21 |
| 4 | PROM# 22 | PROM# 23 | PROM# 24 | PROM# 25 | PROM# 26 | PROM# 27 | PROM# 28 |

Figure 8-3. PROM MAP

| Bit No.[a] → / Executable Instruction Number ↓ | 0–3 | 4–11 | 12–15 | 16–31 | 32–35 | 36–39 | 40–47 |
|---|---|---|---|---|---|---|---|
| 0 to 255 | PROM # 1 | PROM# 2 | PROM# 3 | PROM# 4 | PROM# 5 | PROM# 6 | PROM# 7 |
| 256 to 767 | PROM# 8 | PROM# 9 | PROM# 10 | PROM# 11 | PROM# 12 | PROM# 13 | PROM# 14 |
| 768 to 895 | PROM# 15 | PROM# 16 | PROM# 17 | PROM# 18 | PROM# 19 | PROM# 20 | PROM# 21 |
| 896 to 1023 | PROM# 22 | PROM# 23 | PROM# 24 | PROM# 25 | PROM# 26 | PROM# 27 | PROM# 28 |

Figure 8-4. Organization of PROMs

## POST PROCESSING FEATURES

AMPROM 29 allows the user to specify:

- The depth (number of instructions) and width (bits of the microword) for each PROM.
- Listing or suppression of listing of the PROM MAP.
- Optional punching of PROM contents on paper tape in BNPF or hexadecimal format.
- Listing or suppression of listing of PROM content.
- Listing of the PROM content by PROM rows or PROM columns or by PROM number.
- Optional automatic inversion of all bits except the "don't care" bits.
- Specification of "don't care" bits to be 0 or 1.

## EXECUTION COMMAND FOR AMPROM 29

To execute AMPROM the general form of the command is:

```
A > AMPROMΔO=qqqqqqqq.ggg { Δ options }   cr
```

The primary part of the object code filename **must** be typed. If the generic part is not specified, the default .OBJ is assumed.

Options and their default values are shown in Table 8-1.

**Table 8-1. AMPROM 29 Options**

| OPTION | ABBREVIATED OPTION | DEFAULT | MEANING |
|---|---|---|---|
| OBJECTΔfilename1<br>or<br>OBJECT=filename1 | O | NONE. This is a required input. | Specifies the name of the file on which the AMDASM object code is located. If only the primary part of filename1 is input, the default generic .OBJ is assumed. |
| MAP | M | MAP | Print the PROM map. |
| NOMAP | NM | | Suppress printing the PROM map. If NOMAP is not specified, the program automatically prints the PROM map. |
| HEX | H | HEX | Punch the PROM output in hexadecimal format. |
| BNPF | B | | Punch the PROM output in BNPF format. If BNPF is not specified the output is automatically punched in hexadecimal. |
| INVERT | I | No inversion | If INVERT is specified, all ones are inverted to zeros, and zeros to ones, except for bits specified as "don't cares". If INVERT is not specified there is no modification to the binary object code. |
| PUNCHΔfilename2<br>or<br>PUNCH=filename2 | P | filename1.OUT | Specifies the name of the file or device where punch data is to be output. If not specified the output goes to the file with the default name filename1.OUT. |
| NOPUNCH | NP | | Suppresses punching the PROM contents. If not specified, defaults to PUNCH. |
| LISTΔfilename3<br>or<br>LIST=filename3 | L | filename1.P3L | Specifies the name of the output file device where the AMPROM output listing is to be placed. If not specified, the output automatically goes to the default file named filename1.P3L. |
| NOLIST | NL | | Specifies that the output is not to be listed. This would be used when only punching of the output is desired. If not specified the program defaults to LIST using the default file named filename1.P3L. |

## AMPROM FILENAMES

As part of the options the user may need to specify filename information. Whether filename information is needed will depend on whether or not the user wishes to receive his output at a printer console or punched on paper tape or stored on files with or without default name assignments.

The PUNCHΔfilename and LISTΔfilename must each be preceded by a blank and may be specified in any order. The filename may be any AMDOS 29 device.

If, for example, the user executed AMDASM with the command:

```
A > AMDASMΔP1Δ2900ΔP2Δ2900   cr
```

the binary object code is stored on a file called 2900.OBJ. When executing AMPROM, only 2900 must be given as the input filename.

Thus the command to execute AMPROM is:

```
A > AMPROMΔOΔ2900    cr
```

and since no LIST or PUNCH is specified, all output will be to the default filenames 2900.OUT and 2900.P3L.

## AMPROM EXECUTION EXAMPLES

The command

```
A>AMPROMΔNOLISTΔPUNCHΔPUN:ΔOBJECTΔ2900   cr
```

specifies that listing of the PROM content is to be suppressed, the output is to be punched on paper tape, and the input (binary object code) for execution of AMPROM is to be from a file called 2900.OBJ.

To illustrate execution of AMPROM with list output to the list device, the command:

```
A >AMPROMΔO= qqqqqqqq.gggΔL=LST:    cr
```

specifies the PROM MAP and the PROM content are to be printed on the list device, the content of the PROMs is not to be punched, but will be stored in hexadecimal on the file with the default name qqqqqqqq.OUT.

However,

```
A > AMPROMΔO=qqqqqqqq.gggΔNOLISTΔNOMAPΔPUNCH=PUN:  cr
```

specifies that the content of the PROMs is to be punched on the paper tape punch with no listing of the PROM MAP or PROM content.

NOTE:

- Each option is preceded by a required blank
- Options may be given in any order
- The full option name or the abbreviated option name may be used.
- If filename1 has no generic specified, it defaults to .OBJ.
- If filename2 (PUNCH) is input without a generic, AMPROM assumes no generic, and uses exactly what was input.
- If filename3 (LIST) is input without a generic, AMPROM assumes no generic, and uses exactly what was input.

## INTERACTIVE AMPROM INPUT

Once AMPROM has begun execution the user will be acting interactively with the console. He will receive messages from the console and will be expected to input responses followed by a carriage return. The terminal prints the requested output and messages requesting additional input. When execution is complete, control returns to AMDOS 29.

A sample of the console messages is given below. For this example, underlined numbers are used to illustrate the user's input. Following the example is a table of the acceptable substitutes which may be used for the underlined values.

After the user has input an AMPROM execution command, the terminal responds by printing:

```
DON'T CARES?  1   cr
ENTER PROM WIDTHS  4 • 8, 4   cr

ENTER PROM DEPTHS  128   cr
```

If a MAP listing at the output device is requested the PROM MAP is output here. Then the console prints:

```
WHICH PROMS DO YOU WISH TO PRINT?  5-7   cr
```

If printing of the PROM content was specified, the PROM content is printed here. These same PROMs will be punched unless NOPUNCH was specified. The punch device should be turned on before keying in the PROMs to be printed and punched.

When execution is complete, control is returned to AMDOS 29.

# INPUT SUBSTITUTES

When the terminal requests information the substitutes permitted are shown in Table 8-2.

Table 8-2
AMPROM 29 Input Substitutes

| Console Prompt | Substitutes | Meaning |
|---|---|---|
| DON'T CARES? | 0 or 1 | The value specified here is assigned to all "don't care" bits in the PROM(s). Any value except 0 or 1 is an error and the prompt is repeated. |
| ENTER PROM WIDTHS | n | n is a decimal integer and each PROM is n bits wide. If the microword size is 60 and n is given as 8, 8 PROMs will be generated. The first seven will contain actual microword information but the 8th PROM will contain microword information in its leftmost 4 bits and "don't cares" in the 4 right-hand bits. (i.e., if the microword width is not an even multiple of n, it is padded on the right with "don't cares"). |
|  | I•b | I is a decimal integer indicating a number of PROMs. b is a decimal integer indicating the number of bits wide each of these PROMs should be<br>Thus, 3 • 4 means there are 3 PROMs each 4 bits wide. |
|  | Combinations of n and I•b | For the PROM MAP (Figure 7-4), the user would write 4, 8, 4, 16, 2•4, 8.<br>Any combination of n and I•b is permissible if separated by commas and if the total number of bits is greater than or equal to the microword width. |
| ENTER PROM DEPTHS | r | r is a decimal integer and each PROM is r instructions deep (long). If the binary object code is not an even multiple of r, AMPROM fills the final PROM locations with "don't cares". |
|  | t•d | t is a decimal integer indicating a number of PROMs. d is a decimal integer indicating how many words deep each of these PROMs is to be. Thus 2 • 512 indicates there are 2 PROMs each 512 bits deep. |
|  | Combinations of r and t•d | For the PROM MAP in Figure 7-4, the user would write 256, 512, 2•128.<br>Any combination of r and t•d is permissible if separated by commas. |
| WHICH PROMS DO YOU WISH TO PRINT*** | Y | Y is a decimal integer which is a PROM number. 5 means list the contents of PROM #5. |
|  | $Y_1$-$Y_n$ | $Y_1$ is a decimal integer specifying the number of the first PROM to be listed. $Y_n$ is a decimal integer specifying the last PROM to be listed. Thus, 2-5 specifies listing of PROMs 2, 3, 4 and 5. |
|  | Combinations of Y and $Y_1$-$Y_n$ | 3, 5-7, 9 means print (and punch) PROMs 3, 5, 6, 7 and 9. All combinations of Y and $Y_1$-$Y_n$ are acceptable if separated by commas. |
|  | Cs | C means column and s is a decimal integer which specifies the PROM column desired. C4 means print all PROMs in column 4. |
|  | C$s_1$-$s_n$ | Print columns $s_1$, through $s_n$. C1-6 indicates print PROM columns 1 through 6. |
|  | Combinations of Cs, $s_1$-$s_n$ | C5, 7-9, 11 means print columns 5, 7, 8, 9, 11. C3-6, 10 means print columns 3, 4, 5, 6, 10 (i.e., C is only given once, then the s and/or $s_1$-$s_n$ separated by commas). |
|  | Rs | R means row and s is a decimal integer which specifies the row desired. R1 means print all PROMs in row 1. |
|  | R$s_1$-$s_n$ | List the contents of PROM rows $s_1$, through $s_n$. R2-6 means print all PROMs in rows 2 through row 6. |
|  | Combinations of Rs, $s_1$-$s_n$ | The same as columns. The R is given once, followed by the row numbers separated by commas.<br>R1, 4-6, 11-13 prints rows 1, 4, 5, 6, 11, 12, 13. |
|  | N | The letter N is typed if the user wishes to indicate none of the PROM contents are to be listed |
|  | A | The letter A when typed means all PROMs are to be printed. |

***The same PROMs are printed and/or punched. Thus, all values for printing apply for punching also.

## BNPF PAPER TAPE OPTION

When BNPF is specified as an option, the tape is punched in the BNPF format. B is punched as the first character, then a P (for a one) or an N (for a zero) is punched for each bit in the width of this PROM, then an F is punched as the last character for this row of PROM data. This continues until all rows (the depth) of the PROM are punched.

Before the first BNPF for each PROM is punched, the program punches identification on the tape which consists of:

- 32 Rubouts
- 4 ASCII characters which are the PROM number
- 32 NULs to be used as the leader when loading the PROM burner tape reader

After the PROM data is punched, 40 NULs are punched to facilitate tape handling.

For example, if PROM#5 is 4 bits wide by 128 bits deep, and begins at origin zero, the paper tape will appear as shown in Table 8-3.

Table 8-3.
BNPF Paper Tape Contents

| Tape Contents | Content Explanation |
|---|---|
| $\text{Rubout}_1$<br>•<br>•<br>•<br>$\text{Rubout}_{32}$ | 32 Rubouts |
| Characters 0005 | PROM number |
| $\text{NUL}_1$<br>•<br>•<br>•<br>$\text{NUL}_{32}$ | 32 NULs |
| Character B<br>Character N or P<br>Character N or P<br>Character N or P<br>Character N or P<br>Character F | BPNF format for one row of this 4-bit wide PROM |
| Space | *See Note |
| Character B<br>Character N or P<br>•<br>•<br>• | Repeated 127 times |
| $\text{NUL}_1$<br>•<br>•<br>•<br>$\text{NUL}_{40}$ | 40 trailing NULs |

*Note: Carriage return/line feed for possible listings is inserted after 8 words for PROMs 4 or less bits wide, after 4 words for widths of 16 or less bits, and after one word for widths greater than 16.

## HEXADECIMAL PAPER TAPE OPTION

When punching is desired, and HEX is specified or assumed by default, the PROM contents are punched in the DATA I/O hexadecimal format.

The same initial data (32 Rubouts, PROM number and 32 NULs) is punched as is punched for the BNPF format, followed by the PROM content in hexadecimal.

For PROMs 4 or less bits wide, one hexadecimal character and a space is punched. For PROMs greater than 4 bits wide, two hexadecimal characters and a space are punched. Thus, two characters, space, two characters, space would be punched for either 2 rows of an 8-bit PROM, or for 1 row of a 16-bit wide PROM.

Thus if PROM#7 (16 bits x 128 words) is punched, the output will appear as shown in Table 8-4.

Table 8-4.
Hexadecimal Paper Tape Contents

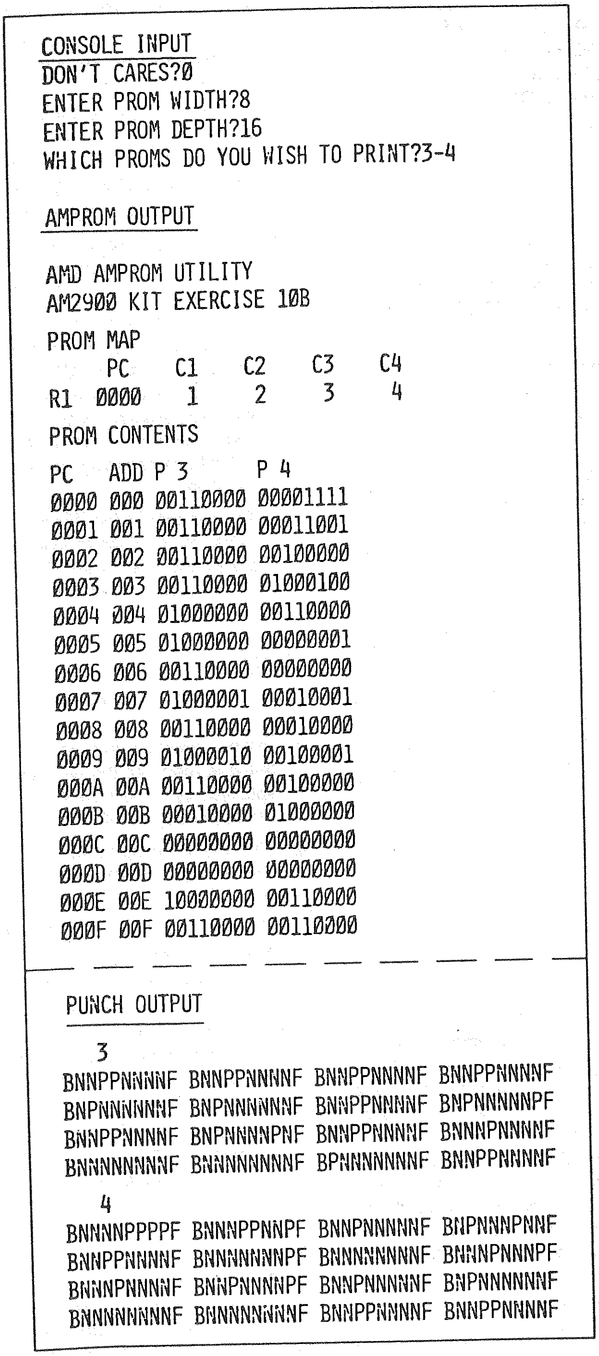
| Tape Contents | Content Explanation |
|---|---|
| $\text{Rubout}_1$<br>•<br>•<br>•<br>$\text{Rubout}_{32}$ | 32 Rubouts |
| Characters 0007 | PROM Number |
| $\text{NUL}_1$<br>•<br>•<br>•<br>$\text{NUL}_{32}$ | 32 NULs |
| SOH | Start of Header |
| Character<br>Character<br>Space<br>Character<br>Character<br>Space | Contents of PROM Row 1 (4 HEX digits) |
| Character<br>Character<br>•<br>•<br>• | Repeated 127 Times<br>*See Note |
| ETX | End of Text |
| $\text{NUL}_1$<br>•<br>•<br>•<br>$\text{NUL}_{40}$ | 40 NULs |

*Note: A carriage return/line feed for possible listings is inserted after 16 groups of hexadecimal characters.

# CHAPTER IX

## EXAMPLE OF AMPROM 29

Figure 9-1 is an example of AMPROM 29 for the Am2900 Learning and Evaluation Kit.



```
CONSOLE INPUT
DON'T CARES?0
ENTER PROM WIDTH?8
ENTER PROM DEPTH?16
WHICH PROMS DO YOU WISH TO PRINT?3-4

AMPROM OUTPUT

AMD AMPROM UTILITY
AM2900 KIT EXERCISE 10B
PROM MAP
      PC    C1    C2    C3    C4
R1  0000     1     2     3     4
PROM CONTENTS
PC   ADD P 3      P 4
0000 000 00110000 00001111
0001 001 00110000 00011001
0002 002 00110000 00100000
0003 003 00110000 01000100
0004 004 01000000 00110000
0005 005 01000000 00000001
0006 006 00110000 00000000
0007 007 01000001 00010001
0008 008 00110000 00010000
0009 009 01000010 00100001
000A 00A 00110000 00100000
000B 00B 00010000 01000000
000C 00C 00000000 00000000
000D 00D 00000000 00000000
000E 00E 10000000 00110000
000F 00F 00110000 00110000

PUNCH OUTPUT

  3
BNNPPNNNNF BNNPPNNNNF BNNPPNNNNF BNNPPNNNNF
BNPNNNNNNF BNPNNNNNNF BNNPPNNNNF BNPNNNNNPF
BNNPPNNNNF BNPNNNNPNF BNNPPNNNNF BNNNPNNNNF
BNNNNNNNNF BNNNNNNNNF BPNNNNNNNF BNNPPNNNNF
  4
BNNNNPPPPF BNNNPPNNPF BNNPNNNNNF BNPNNNPNNF
BNNPPNNNNF BNNNNNNNPF BNNNNNNNNF BNNNPNNNPF
BNNNPNNNNF BNNPNNNNPF BNNPNNNNNF BNPNNNNNNF
BNNNNNNNNF BNNNNNNNNF BNNPPNNNNF BNNPPNNNNF
```

Figure 9-1. AMPROM 29 Output for Am2900 Learning and Evaluation Kit.

# CHAPTER X

# PROM PROGRAMMER SUBSYSTEM

## SUBSYSTEM DESCRIPTION

The PROM Programmer subsystem provides the software routines that reformat the microinstruction fields and output the microcode to the PROM Programmer. Two program files, PFORMAT.COM and PPROG.COM, contain the PROM Programmer subsystem software. PFORMAT.COM converts an AMPROM output file (filename.OUT) to a DATA I/O format file (filename.DIO). PPROG.COM interfaces DATA I/O format files to the PROM Programmer via a set of subsystem commands.

## PFORMAT COMMAND

The PFORMAT command converts an AMPROM output file to a DATA I/O PROM Programmer input file. Each PROM defined on the AMPROM output file is defined by PROM number, on the DATA I/O input file. The format of the PFORMAT command is:

PFORMAT filename1 (.filetype)filename2(.filetype)

filename1 is the name of the AMPROM output file; its filetype is optional and will default to .OUT if omitted.

filename2 identifies the DATA I/O format file; it is optional. When filename2 is not specified, it will default to filename1. The filetype for filename2 is also optional; it will default to .DIO if omitted.

A space is required to delimit PFORMAT from filename1 and delimit filename1 from filename2.

## PPROG COMMAND

The PPROG command selects the PROM Programmer hardware/software interface program. When the PPROG command is entered, the system responds with a P> prompt. Any of the following subcommands can be entered in response to the P> prompt.

| | |
|---|---|
| File | filename.filetype |
| Program | n |
| Verify | n |
| DFile | n |
| DProm | |
| ECho | |
| NOEcho | |
| Exit | |

Any, or all, of the subcommands can be entered on the same line as the PPROG command. Also, the subcommands can be entered on a single line in response to the P> prompt. When PPROG, the subcommands and the appropriate operands are entered on a single command line, they must be separated by one or more delimiters (blank, comma, left parenthesis, right parenthesis, equal sign, or period). Only that portion of the subcommand name which is shown in upper case letters need be entered to activate a subcommand; the lower case letters can be entered if desired. The following description of subcommands and operands also describes the sequence of operations that result when a subcommand is entered.

### File filename.filetype

Opens, for subsequent processing, the DATA I/O format disk file specified by the filename.filetype parameters.

### Program n

Causes the following sequence of events to occur.

1. Prom number n from the file opened by the File subcommand is read into the file input buffer. The decimal number specified by n must be in the range of 1 to 65535. When n is omitted, the first PROM on the file is read.
2. The PROM Programmer is queried for its PROM type; PROM depth, width, and erased state are displayed on the console.
3. The contents of the file input buffer are transferred to the PROM Programmer RAM. A message is displayed on the console stating that the transfer is taking place and the console speaker is beeped at ¾ second intervals to inform the user that the transfer is proceeding normally.
4. An illegal bit test is performed to insure that the user is not trying to unprogram a bit that is already programmed in the PROM.
5. The PROM Programmer RAM is programmed into the PROM. A message is displayed on the console to inform the user that the PROM is being programmed. Also, the console speaker is beeped every 2 seconds to indicate that programming is proceeding normally. A message is displayed on the console to inform the user that the PROM programming operation has been completed successfully.
6. The contents of the PROM are verified against the PROM Programmer RAM to insure that programming has completed and is accurate. A successful verification message is displayed on the console.

### Verify n

Causes the following sequence of events to occur:

1. PROM number n from the file opened by the File subcommand is read into the file input buffer. The decimal number specified by n must be in the 1 to 65535 range. When n is omitted, the first PROM on the file is read.
2. The PROM Programmer is queried for its PROM type. PROM depth, width and erased state are displayed on the console.
3. The contents of the PROM are read into the PROM Programmer RAM.
4. The contents of the PROM Programmer RAM are transferred to the PROM input buffer in System 29 memory. A message is displayed on the console stating that the transfer is taking place and the console speaker is beeped at ¾ second intervals to inform the user that the transfer is proceeding normally.
5. The file input buffer (written in step 1) is compared with the PROM input buffer (written in step 4) and any differences are displayed on the console.

### DFile n

The contents of PROM number n from the open file is displayed on the console. The decimal number specified by n must be in the 1 to 65535 range. When n is omitted, the contents of the first PROM on the file are displayed. A file is displayed as ASCII translated memory images.

### DProm

The contents of the PROM currently in the PROM Programmer socket are displayed on the console in ASCII translated memory images.

### ECho

ECho causes all input/output transactions between the PROM Programmer and System 29 to be displayed on the console.

### NOEcho

NOEcho cancels the operation selected by the ECho subcommand.

### Exit

Exit terminates the PROM Programming subsystem mode and return control to AMDOS 29.

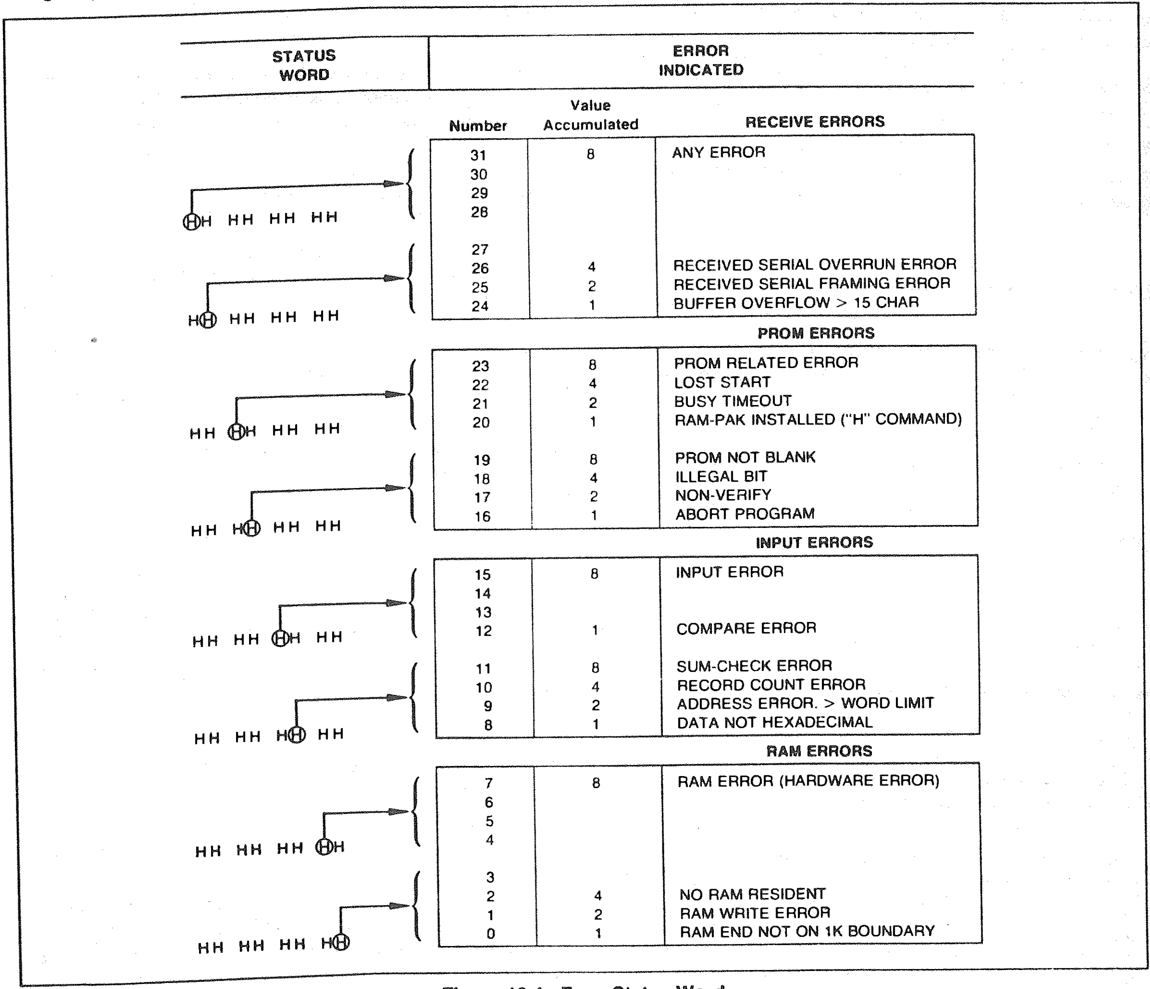
## ERROR STATUS

When any of the steps in Program or Verify fail, an error message describing the failure is displayed on the console. If the failing step involves the PROM Programmer hardware, the Programmer error status word shown in Figure 10.1 is read from the Programmer and displayed on the console. The remaining steps in the sequence are aborted.

Bit 31 is set, a Hexadecimal 8 is displayed, whenever any error information is contained in the error status word. The rest of the error status word indicates, by bits being set, what error conditions have occurred. For example, the error status word 80C80081 is displayed to indicate the following errors:

- 8 – Bit 31 is set to indicate the error status word contains error information.
- 0 – No receive errors
- C – Bits 23 and 22 are set to indicate a PROM related error (bit 23) and a lost start (bit 22).
- 8 – Bit 19 set to indicate PROM is not blank
- 0 – No input errors.
- 0 – No input errors.
- 8 – Bit 7 is set to indicate that there is a RAM error.
- 1 – Bit 0 is set to indicate RAM end not on 1k boundary.

After being displayed, the error status word is reset to zeros.



| STATUS WORD | Number | Value Accumulated | ERROR INDICATED |
|---|---|---|---|
| | | | **RECEIVE ERRORS** |
| ⒽH HH HH HH | 31 | 8 | ANY ERROR |
| | 30 | | |
| | 29 | | |
| | 28 | | |
| HⒽ HH HH HH | 27 | | |
| | 26 | 4 | RECEIVED SERIAL OVERRUN ERROR |
| | 25 | 2 | RECEIVED SERIAL FRAMING ERROR |
| | 24 | 1 | BUFFER OVERFLOW > 15 CHAR |
| | | | **PROM ERRORS** |
| HH ⒽH HH HH | 23 | 8 | PROM RELATED ERROR |
| | 22 | 4 | LOST START |
| | 21 | 2 | BUSY TIMEOUT |
| | 20 | 1 | RAM-PAK INSTALLED ("H" COMMAND) |
| HH HⒽ HH HH | 19 | 8 | PROM NOT BLANK |
| | 18 | 4 | ILLEGAL BIT |
| | 17 | 2 | NON-VERIFY |
| | 16 | 1 | ABORT PROGRAM |
| | | | **INPUT ERRORS** |
| HH HH ⒽH HH | 15 | 8 | INPUT ERROR |
| | 14 | | |
| | 13 | | |
| | 12 | 1 | COMPARE ERROR |
| HH HH HⒽ HH | 11 | 8 | SUM-CHECK ERROR |
| | 10 | 4 | RECORD COUNT ERROR |
| | 9 | 2 | ADDRESS ERROR. > WORD LIMIT |
| | 8 | 1 | DATA NOT HEXADECIMAL |
| | | | **RAM ERRORS** |
| HH HH HH ⒽH | 7 | 8 | RAM ERROR (HARDWARE ERROR) |
| | 6 | | |
| | 5 | | |
| | 4 | | |
| HH HH HH HⒽ | 3 | | |
| | 2 | 4 | NO RAM RESIDENT |
| | 1 | 2 | RAM WRITE ERROR |
| | 0 | 1 | RAM END NOT ON 1K BOUNDARY |

Figure 10-1. Error Status Word

# CHAPTER XI

## ERROR MESSAGES AND INTERPRETATIONS

### AMDASM ERRORS

Each source file input statement is processed until a single error is detected. One missing comma between fields, for example, would result in incorrect processing of the remainder of the statement.

Thus, the assembler stops when an error is encountered, records the error and the statement which caused it, and proceeds to process subsequent source input statements.

Note that console error messages without an error number are AMDOS/29 error messages.

AMDASM and AMPROM error messages will have the form

```
*** ERROR n {y}
```

where n is the error number and y, if present, contains the illegal character or symbol. Fatal error messages appear on the console output device as well as on the assembly list file.

Error messages will sometimes seem inappropriate for the statement being processed. This occurs because the assembler is unable to determine the programmer's intent. This is often the result of a missing comma (,), semicolon (;), blank (Δ) or colon (:).

Errors where n is ≥ 100 halt execution.

It is recommended that the user read the entire error message section.

### ERROR 1 ILLEGAL CHARACTER

The character which cannot be interpreted is printed and the line in which it occurs is also printed. This message may be generated by:

- Striking the wrong console key.
- A missing comma or semicolon (B#101Q#7 is not interpretable).
- A wrong number base used (B#3 or Q#8 cannot be interpreted).

### ERROR 2 UNDEFINED SYMBOL

This message will most often occur when:

- Something is misspelled.

  HERE: EQUΔ100
  GO.TO: DEFΔ HEER (the assembler cannot find HEER)
- The # is missing after a B, Q, D, or H.
- The space is missing after definition words DEF, EQU, SUB, WORD, TITLE, RES, ORG, ALIGN, FF, SPACE
- A symbol is referenced before it is defined by a SUB or an EQU.
- A VFS for a hexadecimal field begins with the letters A through F and the H# designator does not precede the letter.

### ERROR 3 UNDEFINED FORMAT

The format name given is misspelled or was not defined in the Definition Phase or the required blank was not supplied after the format name.

### ERROR 4 DUPLICATE FORMAT

The name given before a format (DEF) has already been used as a name. If names contain more than 8 characters, the first 8 must be unique. Check for misspelled names.

### ERROR 5 DUPLICATE LABEL

This label has been used more than once as a constant name or a label. If the label is more than 8 characters, the first 8 must be unique.

### ERROR 6 DUPLICATE SUBDEFINE

The name given preceding a subformat (SUB) has already been used as a name. If names contain more than 8 characters, the first 8 must be unique. Check for misspelled names.

### ERROR 7 FORMAT FIELD OVERFLOW

The user is permitted a maximum of 128 fields per format name (DEF). This number has been exceeded. The format must be revised and fields must be combined.

### ERROR 8 SUBDEFINE FIELD OVERFLOW

The user is permitted a maximum of 128 fields per subformat name (SUB). This number has been exceeded. Revise the subformat and combine fields or use two subformats for this bit pattern.

### ERROR 9 UNDEFINED DIRECTIVE

No name: was found and the characters given are not TITLE, WORD, LIST, NOLIST, END, ORG, RES, SPACE, or ALIGN.

Check for a missing colon after a name, or misspelling, or blanks in TITLE, WORD, etc.

### ERROR 10 ILLEGAL MICROWORD LENGTH

Each time DEF or FF is encountered, the assembler checks to see if the sum of the bits for all fields for this format name **exactly** equals the microword length.

Thus, the user is assured that each DEF or FF contains an exact number of bits. If the number of bits in this format does not **exactly** equal the number of bits given with WORD, the interpretation of the faulty DEF or FF is bypassed and the assembler attempts interpretation of the next source statement.

### ERROR 11 ILLEGAL FIELD LENGTH

No field, except a "don't care" field, may be more than 16 bits in length. The value calculated for this field cannot be represented in 16 bits.

### ERROR 12 DON'T CARE FIELD TOO LONG.

The explicit length given for a "don't care" field exceeds the microword length specified by WORD. Improper digits may have been assumed for the explicit length due to a missing comma or designator.

### ERROR 13 ARITHMETIC OPERATION ON FIXED FIELD.

If a field is defined as a variable field in the Definition File, an expression cannot be used as a VFS in the Assembly File unless the field contained the % attribute in its definition.

### ERROR 14 ATTRIBUTE ERROR

Both the negative (−) sign and inversion (∗) have been assigned to a single variable or constant. This is not permitted. 4V−∗ or 4B#1011∗− are meaningless.

### ERROR 15 (Not used)

### ERROR 16 MISSING END STATEMENT

The Definition or Assembly File is missing the END statement.

### ERROR 17 ILLEGAL SYMBOL

A character other than A through Z, digits 0 through 9, or period was used in a name, or a comma may be missing between fields.

### ERROR 18 OVERLAY ERROR

This message is given when two formats are overlayed and both of them contain constants for the same bit position. If the assembler is run using each of the formats in the overlay statement as a separate format, and the output is printed in block form, the erroneous bits are easily detected.

For example if the Definition File statements are:

A: DEFΔ4X,B#1011
B: DEFΔB#01111,3X

and the Assembly File statement is

A & B

the overlay error message occurs.

Rerun the Assembly File with source statements given as

A
B

and block output requested which generates

XXXX $\boxed{1}$ 011
0111 $\boxed{1}$ XXX

It can easily be seen that bits $\begin{array}{|c|}\hline 1\\ \hline 1\\ \hline\end{array}$ are causing the overlay error. The improper DEF can then be corrected and the overlay A & B can be used in the Assembly File statement.

### ERROR 19 NO DEFAULT VALUE

A format name was defined with a variable field in the Definition File. Since no default value was given in the definition, a variable field substitute **must** be supplied for this field when the format name is used in the Assembly File. Check for missing commas.

### ERROR 20 FIELD LENGTH CONFLICT

The calculated or implicit field length for the constant or expression given after the designator does not have the same number of bits as the explicit field length. Check for a missing % or :, or a comma missing after the previous field.

This message may be output when commas are left out. For example,

8H#A39Q#274

is missing the comma between 3 and 9. Thus the program assumes A39 is to be substituted into the 8-bit hexadecimal field.

Similarly,

8H#A3, 9Q27, 4

will generate this error message since the comma between the 7 and the 4 is misplaced.

### ERROR 21 $ SPECIFIED FOR NON-ADDRESS FIELD

In order to use the value of the program counter (indicated with a $) as a VFS, that field must contain the % attribute.

### ERROR 22 (Not used)

### ERROR 23 MISSING DESIGNATOR

A field has been encountered which contains only decimal numbers. This is not permitted for a field in a DEF, SUB or FF. Decimal numbers must be input as, n D# digits, where n is the explicit length of the field and digits are the decimal integers which generate the desired bit pattern or field value.

ERROR 24 SPACE DIRECTIVE ERROR

The value input following SPACE is interpreted as less than zero or greater than the number of lines given per page.

ERROR 25 ORG SET TO LESS THAN CURRENT PC

When ORG is encountered, the value given is compared with the current program (location) counter. If ORG is less than the program counter, the value given with ORG is ignored.

ERROR 26 NO FORMAT NAME AFTER &

When a line ends with an & and no continuation (/) is given at the beginning of the next line, this error is generated. A format name is missing after the &, or a / is missing on the continuation line.

ERROR 27 (Not used)

ERROR 28 ADDRESS NOT IN CURRENT PAGE

When the user gives a label or a label$ as a VFS or has defined his variable field with the $ attribute, this message will be generated if the left bits to be truncated do not match the corresponding bits of the current program counter.

ERROR 29 LENGTH REQUIRED FOR $ MODIFIER

Paged addressing (use of the $ as a modifier) requires the field length before the symbol in FF statements. Thus, 6SYMBOL$ is correct but SYMBOL$ is incorrect.

ERROR 30 ILLEGAL FIELD LENGTH IN FF STMT.

A field is greater than 16 bits in a FF statement. Only "don't care" fields may be larger than 16 bits.

ERROR 31 (Not used)

ERROR 32 NO EXPLICIT LENGTH BEFORE (

An expression in a FF statement must be enclosed in (). The explicit field length must precede the (.

## AMDASM ERRORS WHICH HALT EXECUTION

Error messages with n ≥ 100 cause execution to stop. They are listed below:

ERROR 100 COMMAND OPTION SYNTAX ERROR

The input command contains an error. Check for correct spelling of filenames and options, spaces between options, and correct drive specification with filenames.

ERROR 101 DEF TABLE OVERFLOW

ERROR 102 SUB TABLE OVERFLOW

ERROR 103 EQU TABLE OVERFLOW

ERROR 106 FIELD TABLE OVERFLOW

Errors 101, 102, 103 and 106 occur when the amount of memory available has been exceeded.

ERROR 104 INCORRECT OR MISSING WORD SIZE

Either the WORD n command is not given as the first command (or the first command after TITLE) or the value given for n is < 1 or > 128.

ERROR 105 UNEXPECTED END OF FILE

The user has given an incorrect file name or the source file is not correct. AMDASM has encountered an end of file when it was still expecting data.

## AMSCRM ERRORS

The following list illustrates the Error Messages output by AMSCRM:

ERROR 1: COMMAND OPTION ERROR

There is an error in the execution command. Check for delimiters, correct option spelling, etc.

ERROR 2: INPUT/OUTPUT FILE NOT SPECIFIED

The input or output file was not specified in the execution command, or an incorrect filename was given.

ERROR 3: FIELD LENGTH EXCEEDS MAXIMUM

The maximum width of any field to be moved ($W_n$) is 16.

### ERROR 4: FIELD EXCEEDS MICROWORD SIZE

The bit number given or the number of bits to be moved is incorrect. For example, if the microword is 32 bits wide, and the parameters

10,5,28

are given, the program attempts to move 5 bits to positions 28, 29, 30, 31, 32. This is impossible since the bit positions for a 32 bit microword only range from 0-31.

### ERROR 5: TRANSFORMATION PARAMETER ERROR

An incorrect character or value has been given in the user's input $S_n$, $D_n$, $W_n$ or a comma is missing between S, D, or W.

### ERROR 6: TRANSFORMED FIELDS OVERLAP

If the user attempts to move bits into positions where AMSCRM has already moved bits, this error occurs. For example, the parameters

6,9,3

15,11,3

would generate this error since they attempt to move two different bits into bit position 11.

## AMPROM ERRORS

### ERROR 1 DON'T CARE DEFINITION ERROR

A value other than zero or one was input as the value for "don't care" bits. The user has input an incorrect character.

### ERROR 2 WIDTH INPUT SYNTAX ERROR

The PROM width specified using n and/or l*b has been stated incorrectly. Check for missing commas or asterisks.

### ERROR 3 WIDTH EXCEEDS MICROWORD SIZE

The width given for all of the PROMs totals to so many bits that at least one additional PROM width is being specified. For example, if the microword width is 60 and PROM width is specified as 9*8, an error will be generated as there are 12 (72-60) extra bits specified which is greater than the 8-bit width of each PROM. Program execution stops. However, 8*8 will not generate an error since the extra 4 bits (64-60) will fit within one 8-bit wide PROM.

### ERROR 4 TOO MANY PROM COLUMNS

The user is limited to 32 columns in his PROM MAP. When a number of columns greater than 32 is specified this error occurs.

### ERROR 5 DEPTH INPUT SYNTAX ERROR

The data (r and/or t*d) specifying the PROM depths has been input incorrectly. Check for missing commas or asterisks.

### ERROR 6 WARNING DEPTH EXCEEDS MAXIMUM PC

The depth specified by the user will require at least one additional PROM filled with "don't cares".

Thus, if the object code depth is 120 words and the user specifies 3•64 for t•d, the extra 72 words are flagged as an error. However, if the user specified 2•64 (or 128) the extra 8 words would simply be filled with "don't cares". This is issued as a warning message. The additional PROM is filled with "don't cares" and the program continues executing.

### ERROR 7 TOO MANY PROM ROWS

A PROM MAP may contain a maximum of 64 rows. This provides for 64K of storage if the user has chosen 1K PROMs. A PROM MAP with more than 64 rows is not permitted.

### ERROR 8 ILLEGAL VALUE FOR ROWS OR COLUMNS

The user has input something other than a decimal integer Y or Rs or Cs or the letters N or A.

The user may have forgotten the – between $Y_1$ and $Y_n$ or $Cs_1$ and $s_n$, etc.

### ERROR 9 ILLEGAL PROM NO., ROW, OR COLUMN DESIGNATION

The user has requested a PROM number or a PROM row or column using a decimal value greater than any of the PROM numbers, PROM row numbers, or PROM column numbers.

### ERROR 10 UNEXPECTED END OF FILE ON INPUT FILE.

This error only occurs when input to AMPROM is from a file (i.e., the user is not inputting the data interactively). A line giving the "don't care" value, the PROM width or the PROM depth, or the printing information has been omitted.

### ERROR 100 COMMAND OPTION SYNTAX ERROR

This error occurs due to illegal command options or illegal syntax.

Execution halts and the correct command must be entered.

Check for misspelling, missing blanks or =, or incorrect drive specifications.

**NOTE:** Errors 1, 2 and 5 are indicated on the console and the previous data request is repeated. In order to end this loop, the user must input correct data or, if he inputs a Control-C, the loop ends and the system is rebooted.

## AMDOS 29 ERRORS

If a system error occurs which is related to AMDASM 29, AMSCRM 29 or AMPROM 29, AMDOS 29 outputs the following error messages on the console:

---

(name) FILE NOT FOUND

---

The (name) input by the user cannot be located on the designated drive. Check for misspelling of the filename or the wrong drive designator.

---

FILE EXTENSION ERROR

---

This is a system error indicating an attempt to write outside the current file extent.

---

END OF DISK DATA ERROR

---

No more disk space for file data. Delete files from current disk or assign files to another disk.

---

NO DIRECTORY SPACE

---

The diskette directory is full. The user must indicate output is to go to another drive or he must make room on this diskette by deleting some files.

NOTE: If the user has inserted a disk which is write protected, he will receive a variety of error messages including:

VERIFY ERROR
WRITE PROTECTED
FILE ERROR
CLOSE ERROR
etc.

# APPENDIX A
# ERRORS

## AMDASM ERRORS

| | |
|---|---|
| ERROR 1 | ILLEGAL CHARACTER |
| ERROR 2 | UNDEFINED SYMBOL |
| ERROR 3 | UNDEFINED FORMAT |
| ERROR 4 | DUPLICATE FORMAT |
| ERROR 5 | DUPLICATE LABEL |
| ERROR 6 | DUPLICATE SUBDEFINE |
| ERROR 7 | FORMAT FIELD OVERFLOW |
| ERROR 8 | SUBDEFINE FIELD OVERFLOW |
| ERROR 9 | UNDEFINED DIRECTIVE |
| ERROR 10 | ILLEGAL MICROWORD LENGTH |
| ERROR 11 | ILLEGAL FIELD LENGTH |
| ERROR 12 | DON'T CARE FIELD TOO LONG |
| ERROR 13 | ARITHMETIC OPERATION ON FIXED FIELD |
| ERROR 14 | ATTRIBUTE ERROR |
| ERROR 15 | (Not used) |
| ERROR 16 | MISSING END STATEMENT |
| ERROR 17 | ILLEGAL SYMBOL |
| ERROR 18 | OVERLAY ERROR |
| ERROR 19 | NO DEFAULT VALUE |
| ERROR 20 | FIELD LENGTH CONFLICT |
| ERROR 21 | $ SPECIFIED FOR NON-ADDRESS FIELD |
| ERROR 22 | (Not used) |
| ERROR 23 | MISSING DESIGNATORS |
| ERROR 24 | SPACE DIRECTIVE ERROR |
| ERROR 25 | ORG SET TO LESS THAN CURRENT PC |
| ERROR 26 | NO FORMAT NAME AFTER & |
| ERROR 27 | (Not used) |
| ERROR 28 | ADDRESS NOT IN CURRENT PAGE |
| ERROR 29 | LENGTH REQUIRED FOR $ MODIFIER |
| ERROR 30 | ILLEGAL FIELD LENGTH IN FF STMT |
| ERROR 31 | (Not used) |
| ERROR 32 | NO EXPLICIT LENGTH BEFORE ( |

## AMDASM ERRORS WHICH HALT EXECUTION

| | |
|---|---|
| ERROR 100 | COMMAND OPTION SYNTAX ERROR |
| ERROR 101 | DEF TABLE OVERFLOW |
| ERROR 102 | SUB TABLE OVERFLOW |
| ERROR 103 | EQU TABLE OVERFLOW |
| ERROR 104 | INCORRECT OR MISSING WORD SIZE |
| ERROR 105 | UNEXPECTED END OF FILE |
| ERROR 106 | FIELD TABLE OVERFLOW |

## AMSCRM ERRORS

| | |
|---|---|
| ERROR 1 | COMMAND OPTION ERROR |
| ERROR 2 | INPUT OUTPUT FILE NOT SPECIFIED |
| ERROR 3 | FIELD LENGTH EXCEEDS MAXIMUM |
| ERROR 4 | FIELD EXCEEDS MICROWORD SIZE |
| ERROR 5 | TRANSFORMATION PARAMETER ERROR |
| ERROR 6 | TRANSFORMED FIELDS OVERLAP |

## AMPROM ERRORS

| | |
|---|---|
| ERROR 1 | DON'T CARE DEFINITION ERROR |
| ERROR 2 | WIDTH INPUT SYNTAX ERROR |
| ERROR 3 | WIDTH EXCEEDS MICROWORD SIZE |
| ERROR 4 | TOO MANY PROM COLUMNS |
| ERROR 5 | DEPTH INPUT SYNTAX ERROR |
| ERROR 6 | WARNING DEPTH EXCEEDS MAXIMUM PC |
| ERROR 7 | TOO MANY PROM ROWS |
| ERROR 8 | ILLEGAL VALUE FOR ROWS OR COLUMNS |
| ERROR 9 | ILLEGAL PROM NO., ROW, OR COLUMN DESIGNATION |
| ERROR 10 | UNEXPECTED END OF FILE ON INPUT FILE |
| ERROR 100 | COMMAND OPTION SYNTAX ERROR |

## AMDOS 29 ERRORS

(filename) FILE NOT FOUND
FILE EXTENSION ERROR
END OF DISK DATA ERROR
NO DIRECTORY SPACE
VERIFY
WRITE PROTECTED
FILE ERROR
CLOSE ERROR

# APPENDIX B

## AMDASM 29 MICROCODE OBJECT FILE FORMAT

| BYTE NUMBER | DESCRIPTION |
|---|---|
| 0-59 | Title record (60 bytes) |
| 60 | Microword size (i.e., width in bits) |
| 61-62 | Maximum location (program) counter value |
| 63-64 | Number of microinstructions in file |
| 65 | m = Number of 16 bit words required for each microinstruction |
| *66-67 | One microinstruction record |
| **68-(68+2m−1) | |
| ***(68+2m) − (68+4m−1) | |

*Location (program) counter value.

**Mask defining don't care fields bit = 1− means this is a don't care bit; bit 0 − means this is a defined bit.

***Contents of microinstruction. If corresponding bit of mask = 0, this bit is a defined value. Don't care bits = 0.

Subsequent microinstruction records contain * , ** , and ***.

**NOTE:**

1. All values are binary.
2. Bytes 61 and 62 are stored low order byte first, high order byte second, (e.g., if the value is 01FF it would be stored as FF,01). This also applies for bytes 63-64, 66-67, the mask and the microinstruction which are stored and written as 8080 addresses (i.e., 2 bytes with low order first).
3. If the microcode is not continuous (due to the use of ALIGN, ORG or RES), there is no data stored for the "empty" words of microcode.

# APPENDIX C
# AMDASM 29 COMMAND SUMMARY

**Definition Phase**

| | |
|---|---|
| TITLE | Max 60 characters |
| WORD n | $n \leq 128$ |
| EQUΔ | Name:EQUΔconstant/expression |
| SUBΔ | Name:SUBΔfield, . . . 10 fields max |
| DEFΔ | Name:DEFΔfield, . . . 30 fields max |
| NOLIST | Do not print following statements |
| LIST | Print following statements |
| END | End of Definition Source File |

**Assembly Phase**

| | |
|---|---|
| TITLEΔ | Maximum 60 characters |
| EQUΔ | Name:EQUΔconstant/expression |
| NOLIST | Do not print following statements |
| LIST | Print following statements |
| f.n.Δ | Format nameΔVFS, . . . (from DEF) |
| FFΔ | Free format FFΔfield, . . . max 30 |
| SPACEΔn | Spaces n blank lines |
| EJECT | Ejects page |
| ORGΔn | Resets program counter (forward) |
| RESΔn | Reserves n words of code |
| ALIGNΔn | Sets PC to next even multiple of n |
| LABEL: | Precedes f.n. or FF, value = PC |
| LABEL:: | Entry point for mapping PROM |
| ; | Comment statement |

Notes:
Δ = Required space
Names = 8 characters, no blanks
Char 1 = A-Z, or Char 2-8 = A-Z, 1-9.
{ } = Optional

# APPENDIX D
# AMDASM 29 FIELD AND OPERATOR INFORMATION

---

## CONSTANTS, EXPRESSIONS, CONSTANT FIELDS

{n} des digits {mod}

## VARIABLE FIELDS

n V {attr} {des} {digits} {mod} (digits are default value)
n V {attr} X (defaults to X)
max n = 16

## DON'T CARE FIELDS

n V {attr} X max n = word size

## MODIFIERS (mod) and ATTRIBUTES (attr)

Inversion
- Negation
% Right justify or field has expression
: Truncation
$ Paging (relative addressing) ATTRIBUTE only, sets % and :

## EXPRESSION OPERATORS

+ Add
− Subtract
Multiply
/Divide

Evaluated left to right

## DESIGNATORS (des)

B# Binary
D# Decimal
Q# Octal
H # Hexadecimal

## VARIABLE FIELD SUBSTITUTES (VFS)

Label
Label$
Expression
Digits
Des digits {mod}
Constant name

---

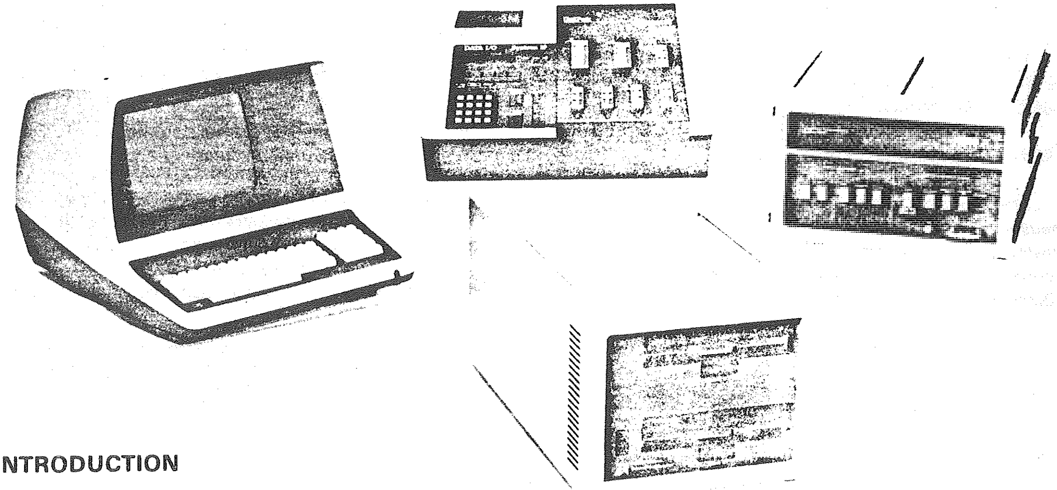
Notes:
{ } = Optional
Des = Designator
Attr = Attribute
Mod = Modifier
Digits = Numbers

APPLICATION NOTE

DATA I/O P.O. Box 308 / 1297 N.W. Mall Issaquah, Washington 98027 (206) 455-3990 Telex 320290

# Bidirectional Communication Between Data I/O Programmers and AMC Development Systems

*Data I/O Models 7 and 9 and Systems 17 and 19 programmers communicate directly with AMC's System 29/05 and AmSys 8/8 Microcomputer Development Systems in the MOS Technology data format.*



## INTRODUCTION

One of the advantages of an intelligent PROM programmer is its ability to communicate with a development system in a compatible format without the use of an intermediate transmission medium such as paper tape. This direct data transfer not only saves time, but it also reduces the possibility of transmission errors. By using a programmer that can accept data in the format specified by the development system manufacturer, the user is spared the task of writing, testing and debugging a format-translation routine.

AMC offers a remote control driver[1] for use with the Data I/O System 19 equipped with remote control (part number 990-1902), but that particular application allows data transfer from the development system to the programmer only.

This note explains the method of *bidirectional* communication between Data I/O programmers and AMC development systems. Data I/O Models 7 and 9 and Systems 17 and 19, with or without remote control, can communicate directly with both the AMC 29/05 and AmSys 8/8 development systems, using the MOS Technology format.

## INTERFACING THE PROGRAMMER AND THE DEVELOPMENT SYSTEM

### Required Equipment

1. One of the following Data I/O programmers:
   - Model 7 or Model 9, with MOS Technology format (055-0081) and serial I/O interface (950-0045)
   - System 17, configuration 990-1712
   - System 19, configuration 990-1901, -1902 or -1903
2. AMC AmSys 8/8 Microcomputer Development System, with serial-printer option

   or

   AMC System 29/05 Microcomputer Development System

**Interconnection**

The **AmSys 8/8** communicates via its P11 serial port. In order to use this port the 8/8 must be equipped with the serial-printer-port hardware option.

The **System 29/05** communicates via the serial port labeled Reader/Punch Port.

Connect the programmer to the microcomputer system as shown in Figure 1.

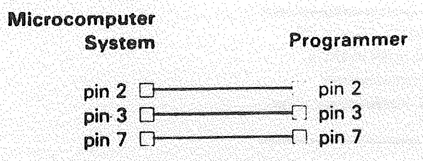


**Figure 1. Interconnection Cable**

This cable may be used with either the AmSys 8/8 or the System 29/05 and may be purchased from AMC² by specifying part number 710111.

## DATA-TRANSMISSION PROCEDURE

The development system requires in its input command sequence an "end-of-file" record. Without this record, the command sequence will not be executed. The end-of-file record is determined by the MOS Technology format. The following paragraphs and Figure 2 explain the format and demonstrate how the end-of-file record is calculated.

### The MOS Technology Format

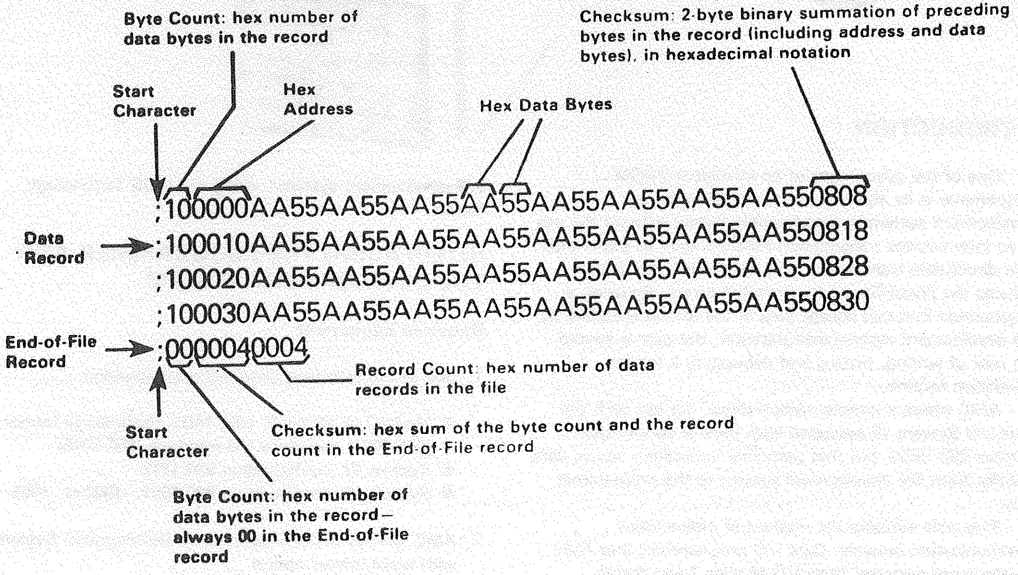
Data is organized into records characterized by expressed addresses and error-check codes. The programmer can accept addresses in nonsequential order.

The data in each record is sandwiched between a seven-character prefix and a four-character suffix. The number of data bytes in each record must be indicated by the byte count in the prefix. The input file can be divided into records of various lengths.

Figure 2 simulates a series of valid data records. Each data record begins with a semicolon (;). The programmer will ignore all characters received prior to the first semicolon. All other characters in the record must be hex digits (0–9, A–F). A two-digit byte count follows the start character; the byte count, expressed in hexadecimal digits, must equal the number of data bytes in the record. The next four digits make up the address of the first data byte in the record. Data bytes follow, each represented by two hexadecimal digits.

The suffix is a four-character checksum, which represents the sixteen-bit binary sum of all bytes in the record, including the address and byte count. Carry from the most significant bit is dropped.

The end-of-file record begins with a byte which is always 00, followed by a checksum and a record count.



```
;100000AA55AA55AA55AA55AA55AA55AA55AA550808
;100010AA55AA55AA55AA55AA55AA55AA55AA550818
;100020AA55AA55AA55AA55AA55AA55AA55AA550828
;100030AA55AA55AA55AA55AA55AA55AA55AA550830
;0000040004
```

**Figure 2. Specifications for Valid MOS Technology Data Files**

*NOTE*

*The end-of-file record may be displayed on the terminal by entering the following command sequence.*

*AmSys 8/8: Key in COPY* sp *CON: = RDR:* Return

*System 29/05: Key in PIP* sp *CON: = RDR:* Return

*Programmer: Initiate an output operation as described in steps 8 and 9, below. If System 19 BLOCK LIMITS*[3] *are to be used in transferring data from the programmer to the development system, the same limits must be set at this time in order to establish the correct end-of-file record.*

## COMMUNICATION WITH THE AmSys 8/8

### Uploading Data from the Programmer to the AmSys 8/8

1. Turn both systems ON.
2. AmSys 8/8: Insert AMDOS8 diskette to drive A (bottom drive).
3. AmSys 8/8: Initialize the system according to the AMDOS operating procedure.
4. Make sure both systems are set to 9600 baud.
5. AmSys 8/8: Check that the copy utility is available on disk drive A.
6. AmSys 8/8: See "A> " on the display. This means the system is ready.
7. AmSys 8/8: Key in the input command:

   COPY *sp* file name = RDR:[Q; end-of-file record *CtlZ* E] *Return*

   *NOTE*

   *When the file name is specified in an upload or download operation, it must include any attribute or extent of that file. For example, if the file name is* PROM1 *and the extent of the file is* .DIO*, the file name to be used in the command sequence would be* PROM1.DIO.

8. Programmer: prepare for communication in the MOS Technology format.

| | |
|---|---|
| **Model 7:** | No selection is necessary. |
| **Model 9:** | The format is selected while initiating the data transfer in step 9. |
| **System 17:** | No selection is necessary. |
| **System 19:** | Press SELECT<br>Key in 81<br>Press START |

9. Programmer: Initiate an output operation.

| | |
|---|---|
| **Model 7:** | Press PROGRAM<br>Press and hold I/O<br>Press EXECUTE |
| **Model 9:** | Press PROGRAM<br>Press and hold I/O<br>Press FWD repeatedly until 81 appears in the display<br>Press EXECUTE |
| **System 17:** | Press MODE SELECT until the REMOTE OUTPUT light comes ON<br>Press START |
| **System 19:** | Press SELECT<br>Key in D5<br>Press START |

### Downloading Data from the AmSys 8/8 to the Programmer

Complete the following sequence to input data to the programmer.

1. Turn both systems ON.
2. AmSys 8/8: Key in the output command:

   COPY *sp* PUN: = file name

   Do not press *Return* at this time.

3. Programmer: Initiate an input operation. Step 4 must then be executed before the programmer's timeout period expires.

| | |
|---|---|
| **Model 7:** | Press LOAD<br>Press and hold I/O<br>Press EXECUTE |
| **Model 9:** | Press LOAD<br>Press and hold I/O<br>Press FWD repeatedly until 81 appears in the display<br>Press EXECUTE |

| | |
|---|---|
| **System 17:** | Press MODE SELECT until<br>REMOTE INPUT light is ON<br>Press START |
| **System 19:** | Press SELECT<br>Key in 81<br>Press START<br>Press SELECT<br>Key in D1<br>Press START |

4. AmSys 8/8: Press *Return*

## COMMUNICATION WITH THE SYSTEM 29/05

### Uploading Data from the Programmer to the System 29/05

1. Turn both systems ON.
2. System 29/05: Insert AMDOS 29 diskette to drive A.
3. System 29/05: Initialize the system according to the AMDOS operating procedure.
4. Make sure the programmer is set to 600 baud.
5. System 29/05: See "A> " on the display. This means the system is ready.
6. System 29/05: Key in the input-command sequence.

   PIP *sp* file name = RDR:[Q;end-of-file record *CtlZ* E] *Return*

   *NOTE*

   *When the file name is specified in an upload or download operation, it must include any attribute or extent of that file. For example, if the file name is* PROM1 *and the extent of the file is* .DIO, *the file name to be used in the command sequence would be* PROM1.DIO.

7. Programmer: Initiate an output operation.

| | |
|---|---|
| **Model 7:** | Press PROGRAM<br>Press and hold I/O<br>Press EXECUTE |
| **Model 9:** | Press PROGRAM<br>Press and hold I/O<br>Press FWD repeatedly until 81<br>appears in the display<br>Press EXECUTE |
| **System 17:** | Press MODE SELECT until<br>REMOTE OUTPUT light is ON<br>Press START |
| **System 19:** | Press SELECT<br>Key in 81<br>Press START<br>Press SELECT<br>Key in D5<br>Press START |

### Downloading Data from the System 29/05 to the Programmer

1. Turn both systems ON.
2. System 29/05: Key in the output command:
   PIP *sp* PUN: = file name

   Do not press *Return* at this time.

3. Programmer: Initiate an input operation. Step 4 must then be executed before the programmer's timeout period expires.

| | |
|---|---|
| **Model 7:** | Press LOAD<br>Press and hold I/O<br>Press EXECUTE |
| **Model 9:** | Press LOAD<br>Press and hold I/O<br>Press FWD repeatedly until 81<br>appears in the display<br>Press EXECUTE |
| **System 17:** | Press MODE SELECT until<br>REMOTE INPUT light is ON<br>Press START |
| **System 19:** | Press SELECT<br>Key in 81<br>Press START<br>Press SELECT<br>Key in D1<br>Press START |

4. System 29/05: Press *Return*

## PROGRAMMING A PROM

A typical application of this interface is the situation in which the PROM-based software for a system needs to be examined in the development system and debugged. The procedure involves 1) removing the PROMs from the system, 2) loading the programmer RAM with the PROM data, 3) uploading the data to the development system, 4) editing the data, 5) downloading the edited data to the programmer and 6) burning a new PROM. The following

# EXERCISE SOLUTIONS

## ANSWERS

ARE THESE PROPER FILE NAMES OR FILE NAME REFERENCES?

Y___DOOR.DEF
Y___DOOR.*
Y___B:*.DEF
Y___D?OR.D?F
N___123.456
Y___TEMP
Y___DOOR.SRC
Y___B:DOOR.SRC
Y___*.*
Y___B:X?X.*
N___1-2-3
Y___B:TEMP

WHAT EXTENSION NAME IS REQUIRED FOR THE DEFINITION FILE BEFORE IT CAN BE ASSEMBLED VIA AMDASM?

.DEF

WHAT EXTENSION NAME IS REQUIRED FOR THE SOURCE FILE BEFORE IT CAN BE ASSEMBLED BY AMDASM?

.SRC

IS THE EXTENSION REQUIRED WHEN CALLING FOR AN ASSEMBLY?

FILE MUST HAVE AN EXTENSION
THE EXTENSION NAME IS NOT REFERENCED

## ANSWERS

WHAT SYMBOL STARTS A COMMENT?

;

WHAT SYMBOL STARTS A LINE THAT IS A CONTINUATION OF THE PRECEEDING LINE?

/

WHAT IS THE DESIGNATOR FOR A HEX CONSTANT?

H#

HOW MANY CHARACTERS CAN BE IN A VARIABLE NAME?

MORE THAN YOU NEED - ONLY THE FIRST 8 ARE REFERENCED

WHAT CHARACTERS MAY BE THE FIRST CHARACTER IN A VARIABLE NAME?

A - Z AND .

WHAT DETERMINES IF % IS A MODIFIER OR AN ATTRIBUTE?

ITS PLACEMENT RELATIVE TO THE BASE DESIGNATOR

WHAT IS THE ATTRIBUTE $ EQUIVALENT TO?

: AND %

IF NO BASE IS GIVEN IN AN EQU STATEMENT, WHAT IS THE DEFAULT?

D#

IF NO BASE IS GIVEN IN A DEF STATEMENT, WHAT IS THE DEFAULT?

B#

IF NO BASE IS GIVEN IN AN ASSEMBLY STATEMENT VARIABLE FIELD SUBSTITUTION, WHAT IS THE DEFAULT?

THE DEFAULT BASE VALUE OR D#

CAN EQU STATEMENTS APPEAR IN THE DEF AND SRC FILES?

YES

CAN DEF STATEMENTS APPEAR IN THE DEF AND SRC FILES?

NO, USE FF IN THE SRC FILE

WHAT IS THE STATEMENT WORD USED FOR?

TO SPECIFY THE MICROWORD WIDTH, AND FOR CHECKING THAT AGAINST THE DEF STATEMENT WIDTHS

HOW WIDE CAN A VARIABLE FIELD BE (NUMBER OF BITS)?

16 BITS

HOW WIDE CAN A DON'T CARE FIELD BE?

UP TO THE MAXIMUM MICROWORD WIDTH (64 OR 128)

WHAT IS THE MAXIMUM NUMBER OF FIELDS ALLOWED IN A DEF STATEMENT?

30