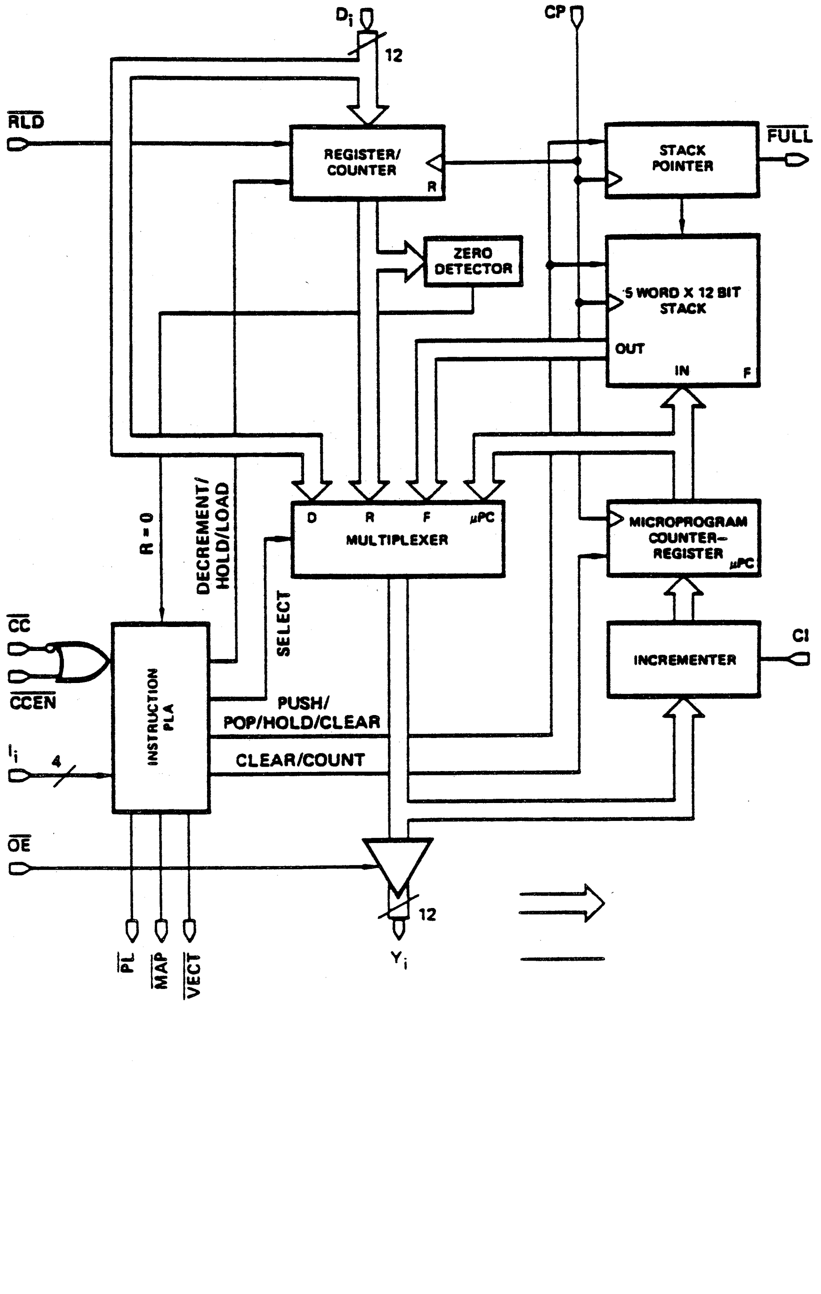
The Am2910 Microprogram Controller

The Am2910 microprogram controller features a basic set of sixteen instructions, and several control lines that effectively expand the functions that may be performed by these instructions.

The device is tristated and its output is controlled by an output enable. Carry-in is normally tied high, but may be used in special interrupt situations to suspend operation. The register-counter may be loaded under intruction control or by pulling $\overline{\text{RLD}}$ low. The test input, $\overline{\text{CC}}$, may be overridden by controlling $\overline{\text{CCEN}}$.

The Am2910 is limited to a 12-bit output, the microprogram address, and is therefore restricted to addressing a 4k control memory. When a larger control memory is needed, a close equivalent to the Am2910 can be achieved by a number of Am2909/11s, one Am29811 and a counter. Fifteen of the sixteen instructions are identical. The differences are the absence of $\overline{\text{CCEN}}$, one of the output enables, $\overline{\text{VECT}}$, and the missing instruction, TWB.



D_i
12
CP
RLD
REGISTER/COUNTER
R
STACK POINTER
FULL
ZERO DETECTOR
5 WORD X 12 BIT STACK
OUT
IN
F
R = 0
DECREMENT/HOLD/LOAD
D
R
F
μPC
MULTIPLEXER
MICROPROGRAM COUNTER-REGISTER
μPC
SELECT
CC
CCEN
INSTRUCTION PLA
INCREMENTER
CI
PUSH/POP/HOLD/CLEAR
CLEAR/COUNT
I_i
4
OE
12
PL
MAP
VECT
Y_i

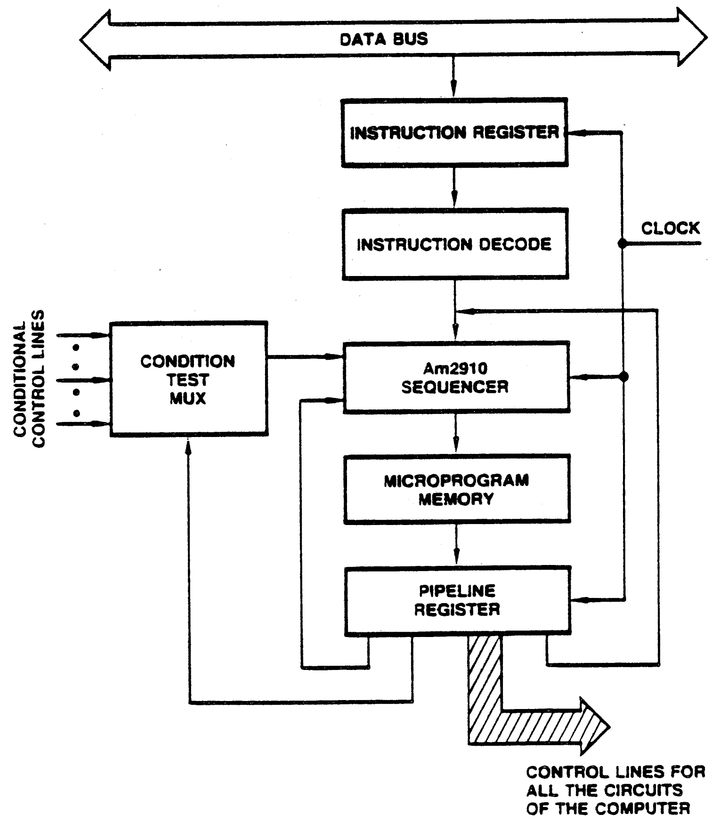
Larger Sequencer

The limit on the addressing capability of the Am2910 is not a problem for the typical CPU and the average controller. The HEX-29 minicomputer, for example, required 1.5k of control memory, 64 bits wide. As a controller becomes more complex, more intelligent and more powerful, larger control memory is required. A larger sequencer, the Am29112, is under development to handle control memories of up to 64k words.

As an example of controller requirements, a disk controller written for an Am29116 application note, required 252 words of 80 bit wide control memory.

CCU Architecture

The conventional architecture of a CCU, as taught in the Advanced Micro Devices Seminars and as appears in AMD application notes, is shown below. It is only one of several options, but is the best compromise between speed of execution and ease of microprogramming. A status register in front of the conditional multiplexer is assumed.



MPR-448

Microprogramming the Am2910

The Am2910 requires one four-bit field in the microword for its instruction lines. The choice for the demonstration program was to define the Am2910 instruction mnemonics as symbolic constants via EQU statements. Another option would be to use the mnemonics as DEF statement labels.

By defining with equates, comma-positional substitution is required. By defining with definition statements, DEF statement overlays are required.

INSTRUCTIONS

			REG/ CNTR CON- TENTS	FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH		PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW			
I_3-I_0	MNEMONIC	NAME		Y	STACK	Y	STACK	REG/ CNTR	ENABLE
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP, CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
10	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
11	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
12	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
13	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
14	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
15	TWB	THREE-WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	POP	PC	POP	HOLD	PL

Note 1: If $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold; else load.   X = Don't Care

```
;
;
;
;************************************************************
; Am2910 MICROPROGRAM CONTROLLER INSTRUCTION SET         [7]
;************************************************************
;
JZ:     EQU     H#0     ; RESET STACK; ADDRESS = 0              (JMP ZERO)
CJS:    EQU     H#1     ; COND JUMP SUB; PUSH STACK             (GO SUB)
JMAP:   EQU     H#2     ; JUMP TO MEMORY MAP ADDRESS            (DECODE)
CJP:    EQU     H#3     ; COND JUMP PIPELINE                    (IF-GOTO)
PUSH:   EQU     H#4     ; PUSH STACK; COND LOAD REG             (SET-UP STACK; CNTR)
JSRP:   EQU     H#5     ; COND JUMP SUB; REG (F) - PIPE (T)     (2-WAY SUB)
CJV:    EQU     H#6     ; COND JUMP VECTOR MAP ADDRESS          (IF-GOTO)
JRP:    EQU     H#7     ; COND JUMP REG (F) OR PIPE (T)         (2-WAY IF-GOTO)
RFCT:   EQU     H#8     ; REPEAT UNTIL CNTR=0; ADDR ON STACK    (DO-LOOP)
RPCT:   EQU     H#9     ; REPEAT UNTIL CNTR=0; PIPELINE ADDR    (DO-LOOP)
CRTN:   EQU     H#A     ; COND RETURN; POP STACK                (IF-RETURN)
CJPP:   EQU     H#B     ; COND JUMP PIPELINE; POP STACK         (EXIT STACK ROUTINE)
LDCT:   EQU     H#C     ; LOAD CNTR; CONTINUE                   (SET-UP COUNTER)
LOOP:   EQU     H#D     ; LOOP UNTIL TEST = TRUE                (DO-UNTIL LOOP)
CONT:   EQU     H#E     ; CONTINUE
TWB:    EQU     H#F     ; 3-WAY BRANCH; PIPE-STACK-CONTINUE     (DEAD MAN TIMER)
;
;
;
```

Control Lines

The Am2910 has additional control lines which may or may not appear in a given microword. $\overline{\text{CCEN}}$ provides the ability to change any of the conditional instructions (CJS, CJP, PUSH, JSRP, CJV, JRP, CRTN, CJPP, LOOP, and TWB) to unconditional instructions. $\overline{\text{RLD}}$ allows the register to be loaded with whatever is on the Di lines. $\overline{\text{OEY}}$en is the Yi output enable control, and is used when the Am2910 shares a bus or when the JZ instruction is created by pull-ups (10K) on the Yi lines during power-on-reset.

```
;
;****************************************************************
; Am2910 ~CCEN - FOR EXPANDED INSTRUCTION SET             [10]
;****************************************************************
;
CCEN:           EQU     B#0     ; TESTING IS NORMAL (~CC ENABLED)
CCDIS:          EQU     B#1     ; ALL TESTS UNCONDITIONALLY TRUE
;
;
;*****************************************************************
; Am2910 RLD - FOR LOADING REGISTER/COUNTER FROM Di       [11]
;*****************************************************************
;
RLD:            EQU     B#0     ; LOAD REGISTER FROM Di
NORLD:          EQU     B#1     ; NO FORCED REGISTER LOAD (INSTRUCTIONS CONTROL IT)
;
```

The Conditional Multiplexer

Some form of testing is normally performed by the sequencer. This can be as simple as a conditional multiplexer, or even a direct input to the $\overline{CC}$ pin. Assuming more than one test is desired, a conditional multiplexer requires a field in the microword for the select lines. The equates shown below assumes an eight-input multiplexer.

PASS

If the multiplexer has sufficient inputs that one can be tied TRUE, then all of the conditional instructions can be made unconditionally true without enlarging the microword. This would have the same effect as providing a field for $\overline{CCEN}$.

Polarity

Also note that CC is active LOW. In some cases, it may be desirable to provide polarity control on the multiplexer output (such as with an Am2922). The select field can either include polarity or a separate field can be added to the microword.

```
;
;*****************************************************************
; CONDITIONAL MUX (EXAMPLES OF TESTS)                         [8]
;*****************************************************************
;
PASS:           EQU     Q#0             ; TRUE
CN:             EQU     Q#1             ; Cn+4 = 1 (MSS)
ZERO:           EQU     Q#2             ; Z PIN = 1
NEG:            EQU     Q#3             ; F3 OF MSS = 1
OVR:            EQU     Q#4             ; OVERFLOW = 1
;
```

Speeding Up the Cycle

The Am2910 is a fast device, however, the basic cycle time can be improved in many cases by not using the three enables produced by the Am2910, $\overline{\text{PL}}$, $\overline{\text{VECT}}$, and $\overline{\text{MAP}}$. Three bits are added instead to the microword to provide control which can reduce the basic cycle time by 20-30ns. The time saved depends upon the design. Also, by providing these separate controls, the basic instruction set of the Am2910 is further expanded.

For example, instead of CJV for an interrupt, where the jump is merely a jump and the routine has no way of returning, CJS may be used with $\overline{\text{VECT}}$ to allow a subroutine to service an interrupt. Note that this must be done with care to avoid overflow of the Am2910 stack (5 deep).

```
;
;
;*************************************************************
; Am2910 Di SOURCE ENABLES - FOR HIGH SPEED CCUs         [9]
;*************************************************************
;
; PIPELINE-MAP-VECTOR ENABLES  (GROUPED CONTROLS)
;
PIPELINE:       EQU     Q#3             ; BRANCH ADDRESS FROM PIPELINE REGISTER
MAP:            EQU     Q#5             ; MEMORY MAP
VECTOR:         EQU     Q#6             ; VECTOR MAP
;
```

Branch Address

The last field that is sequencer related is the branch-address counter-value field. For this program, the maximum field size of 12 was used. The number actually required depends upon the application. This field is used by branching instructions to supply an address to the Di inputs of the Am2910. It is used by other instructions to supply a value to be loaded into the register-counter. In still other instructions, the field is unused.

Sequencer SUB Statements

As for the ALU, SUB statements may be created for the sequencer. The first one shown, "SEQR", includes the "2910 INSTR", "COND MUX", and "BRADDR/CNTR" fields. The second one, "CCU", includes these three fields plus one for the CCEN control line, one for the three-bit output enable, and one for the RLD control line. Again, two different microword formats are involved. SEQR and CCU would not appear in the same source file, although they are perfectly valid appearing together in a definition file.

```
;
;********************************
; EXAMPLE SUB STATEMENTS
;********************************
;
SEQR:    SUB        4VH#E,  3VQ#0,  12V$X
; defaults          CONT    PASS     #
;                    [7]     [8]
;
CCU:     SUB        4VH#E,  1VB#0,  3VQ#0,  12V$X,  3VQ#3,    1VB#1
; defaults          CONT    CCEN    PASS    #       PIPELINE  NORLD
;                   [7]     [10]    [8]             [9]       [11]
;
```

DEF vrs EQU for the Am2910 Instructions

Some programmers do not care for the comma-positional notation of AMDASM. By defining the Am2910 instruction set as DEF statements, rather than equates, the DEF statements may be overlayed with those created for the ALU and other controls. The choice is up to the individual programmer. The criteria should be readability as well as ease of code creation.

The following page indicates some of the ways to program a simple continue-add register instruction.

```
;
;
;
; T O W A R D   M O R E   R E A D A B L E   C O D E
;
;
;*****************************************************************************
; EXAMPLE DEF STATEMENT - TOTALLY DEFINED - NO SUB REFERENCE - COMMA-POSITIONAL
;*****************************************************************************
;
;
;                          F O R M A T
;----------------------------------------------------------------------------------------------------
;          2910   COND   BRANCH SOURCE FUNCT  DEST   CARRY   Ra     Rb     ABMUX         SHIFT
;          INSTR  MUX    ADDR     A      L      U     in     ADDR   ADDR   SEL           SEL
CPU:  DEF  4VH#E, 3VQ#0, 12V$X, 3VQ#0, 3VQ#0, 3VQ#1, 2VB#00, 4VH#0, 4VH#0, 2VB#00, 20X, 4VX
;          CONT   PASS   #      AQ     ADD    NOP    NOC     R0     R0     PIPE
;----------------------------------------------------------------------------------------------------
;
;
;
;*************************************************************************
; EXAMPLE DEF STATEMENTS - USING SUB REFERENCES - COMMA-POSITIONAL
;*************************************************************************
;
;
COMPUTER:   DEF    SEQR,  ALU,   REGS,   24X
;
FASTONE:    DEF    CCU,   ALU,   REGS,   19X
;
POWERFUL:   DEF    CCU,   ALU2,  REGS,   18X
;
;
;
; SAMPLE .SRC STATEMENT ( A COMMENT IN THIS FILE )
;-----------------------------------------------------------
;
; COMPUTER ,,, AB, ADD, RAMF, CIN, R3, R2
;
;
```

```
;
;
;*******************************************************************************
; EXAMPLE DEF STATEMENTS - TOWARD MORE OVERLAY ORIENTATION - USING SUB REFERENCES
;*******************************************************************************
;
;
COMP:   DEF  19X,  ALU,  REGS,  24X
;
CTRL:   DEF  SEQR, 45X
;
;
; SAMPLE .SRC STATEMENT ( A COMMENT IN THIS FILE )
;-------------------------------------------------
;
; CTRL & COMP AB, ADD, RAMF, R3, R2
;
;
;
;*****************************************************************
; EXAMPLE DEF STATEMENTS - HIGHLY SPECIALIZED
;*****************************************************************
;
;
CONTIN:   DEF  CONT, 60X
;
GOSUB:    DEF  CJS, PASS, 57X
;
JMPMAP:   DEF  JMAP, 60X
;
ADD.REG:  DEF  19X, AB, ADD, RAMF, NOC, 4VH#0, 4VH#0, 2VB#00, 24X
;                                       R0     R0     PIPE
;
; SAMPLE .SRC STATEMENT ( A COMMENT IN THIS FILE )
;----------------------------------------------------
;
; CONTIN & ADD.REG R3, R2
;
;
```

Am2901-Am2910 .DEF and .SRC Files

The following pages list the Am2901-Am2910 .DEF and .SRC files which were created for instructional use. The Am2901 and Am2910 equates shown earlier were taken from this file. The various code examples were developed for use in the AMD seminars, and were created to demonstrate the various approaches to writing microcode.

In addition to the simpler code exercises, the "Simple Computer" problem was redone, this time using real parts. The computer was coded once using comma-positional notation, with a minimum of defaults, and a second time, using specifically designed DEF statements. The second approach requires some effort on the part of the programmer and its applicability depends on the repetition present in the code itself.

```
 ;
 ;
 ;
 ;
 WORD 64         ; adjust as needed
 ;
 ;
 ;
 ;
 ; This DEF file was written as part of the application note:
 ; "Toward Structured Programming with AMDASM: The Am2901C and The Am2910A"
 ;  by Donnamaie E. White (Feb 1982)
 ;
 ;
 ;
 ;
```

```
;
;***************************************************************
; Am2901 SOURCE OPERANDS                                    [1]
;***************************************************************
;
AQ:             EQU     Q#0     ; RAMA AND Q REGISTER
AB:             EQU     Q#1     ; RAMA AND RAMB
ZQ:             EQU     Q#2     ; Q ONLY
ZB:             EQU     Q#3     ; RAMB ONLY
ZA:             EQU     Q#4     ; RAMA ONLY
DA:             EQU     Q#5     ; DATA AND RAMA
DQ:             EQU     Q#6     ; DATA AND Q
DZ:             EQU     Q#7     ; DATA ONLY (DA PORT)
;
;***************************************************************
; Am2901 ALU FUNCTIONS                                      [2]
;***************************************************************
;
ADD:            EQU     Q#0     ; R + S
SUBR:           EQU     Q#1     ; S - R
SUBS:           EQU     Q#2     ; R - S
OR:             EQU     Q#3     ; R OR S        (R+S)
AND:            EQU     Q#4     ; R AND S        (RS)
NOTRS:          EQU     Q#5     ; NOT R AND S   (~R)(S)
EXOR:           EQU     Q#6     ; R EXOR S      (RvS)  = RS + (~R)(~S)
EXNOR:          EQU     Q#7     ; R EXNOR S    ~(RvS)  = (~R)S + R(~S)
;
;***************************************************************
; Am2901 DESTINATION CONTROL                                [3]
;***************************************************************
;
QREG:           EQU     Q#0     ; F -> Q only
NOP:            EQU     Q#1     ; F -> Y only
RAMA:           EQU     Q#2     ; F -> B; RAMA -> Y           [ PC OUT; PC + 1 -> PC]
RAMF:           EQU     Q#3     ; F -> B; F -> Y              WRITE TO RAM
RAMQD:          EQU     Q#4     ; F/2 -> B; Q/2 -> Q; F -> Y; DOUBLE DOWN SHIFT
RAMD:           EQU     Q#5     ; F/2 -> B; F -> Y;           SINGLE DOWN SHIFT
RAMQU:          EQU     Q#6     ; 2F -> B; 2Q -> Q; F -> Y;   DOUBLE UP SHIFT
RAMU:           EQU     Q#7     ; 2F -> B; F -> Y;            SINGLE UP SHIFT
;
```

```
 ;
 ;
 ;*************************************************************
 ; CARRY-IN                                                [4]
 ;*************************************************************
 ;
NOC:            EQU     B#00     ; Cin = LOW
CIN:            EQU     B#01     ; Cin = HIGH
IC:             EQU     B#10     ; Cin = Cout of MSS
IZ:             EQU     B#11     ; Cin = Zero detect
 ;
 ;
 ;**********************************************************
 ; OEy                                             [5]
 ;**********************************************************
 ;
OEYEN:          EQU     B#0      ; OUTPUT ENABLE
OEYDIS:         EQU     B#1
 ;
 ;
 ;******************************
 ; EXAMPLE SUB STATEMENTS
 ;******************************
 ;
ALU:     SUB    3VQ#0,  3VQ#0,  3VQ#1,  2VB#00
; defaults       AQ      ADD     NOP     NOC
;                [1]     [2]     [3]     [4]     <-- EQUATE GROUP REFERENCE
;
ALU2:    SUB    3VQ#0,  3VQ#0,  3VQ#1,  1VB#0,  2VB#00
; defaults       AQ      ADD     NOP     OEYEN   NOC
;                [1]     [2]     [3]     [5]     [4]
;
```

```
;
;
;************************************************************
; REGISTERS     (RAM A PORT ADDRESS; RAM B PORT ADDRESS) [R]
;************************************************************
;
R0:             EQU     H#0      ; REGISTER R0
R1:             EQU     H#1      ;          R1
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      ; REGISTER R15
;
;************************************************************
; AB MUX SELECT (REGISTER ADDRESS SOURCE)                [6]
;************************************************************
;
PIPE:           EQU     B#00            ; A, B FROM MICROWORD PIPELINE REGISTER
AIR:            EQU     B#10            ; B FROM PIPELINE; A FROM INSTR REG
BIR:            EQU     B#01            ; A FROM PIPELINE; B FROM INSTR REG
INSTR:          EQU     B#11            ; A, B FROM INSTRUCTION REGISTER
;
; These are just ideas - other choices possible
;
;*********************************
; EXAMPLE SUB STATEMENT
;*********************************
;
REGS:   SUB     4VH#0,  4VH#0,  2VB#00
; defaults       R0      R0      PIPE
;               [R]     [R]     [6]
;
```

```
;
;
;
;***********************************************************
; Am2910 MICROPROGRAM CONTROLLER INSTRUCTION SET        [7]
;***********************************************************
;
JZ:      EQU     H#0     ; RESET STACK; ADDRESS = 0           (JMP ZERO)
CJS:     EQU     H#1     ; COND JUMP SUB; PUSH STACK          (GO SUB)
JMAP:    EQU     H#2     ; JUMP TO MEMORY MAP ADDRESS         (DECODE)
CJP:     EQU     H#3     ; COND JUMP PIPELINE                 (IF-GOTO)
PUSH:    EQU     H#4     ; PUSH STACK; COND LOAD REG          (SET-UP STACK; CNTR)
JSRP:    EQU     H#5     ; COND JUMP SUB; REG (F) - PIPE (T)  (2-WAY SUB)
CJV:     EQU     H#6     ; COND JUMP VECTOR MAP ADDRESS       (IF-GOTO)
JRP:     EQU     H#7     ; COND JUMP REG (F) OR PIPE (T)      (2-WAY IF-GOTO)
RFCT:    EQU     H#8     ; REPEAT UNTIL CNTR=0; ADDR ON STACK (DO-LOOP)
RPCT:    EQU     H#9     ; REPEAT UNTIL CNTR=0; PIPELINE ADDR (DO-LOOP)
CRTN:    EQU     H#A     ; COND RETURN; POP STACK             (IF-RETURN)
CJPP:    EQU     H#B     ; COND JUMP PIPELINE; POP STACK      (EXIT STACK ROUTINE)
LDCT:    EQU     H#C     ; LOAD CNTR; CONTINUE                (SET-UP COUNTER)
LOOP:    EQU     H#D     ; LOOP UNTIL TEST = TRUE             (DO-UNTIL LOOP)
CONT:    EQU     H#E     ; CONTINUE
TWB:     EQU     H#F     ; 3-WAY BRANCH; PIPE-STACK-CONTINUE  (DEAD MAN TIMER)
;
;
;
;************************************************************
; CONDITIONAL MUX (EXAMPLES OF TESTS)                    [8]
;************************************************************
;
PASS:            EQU     Q#0             ; TRUE
CN:              EQU     Q#1             ; Cn+4 = 1 (MSS)
ZERO:            EQU     Q#2             ; Z PIN = 1
NEG:             EQU     Q#3             ; F3 OF MSS = 1
OVR:             EQU     Q#4             ; OVERFLOW = 1
;
```

```
;
;
;
;
;*************************************************************
; Am2910 Di SOURCE ENABLES - FOR HIGH SPEED CCUs          [9]
;*************************************************************
;
; PIPELINE-MAP-VECTOR ENABLES  (GROUPED CONTROLS)
;
PIPELINE:       EQU     Q#3             ; BRANCH ADDRESS FROM PIPELINE REGISTER
MAP:            EQU     Q#5             ; MEMORY MAP
VECTOR:         EQU     Q#6             ; VECTOR MAP
;
;
;**************************************************************
; Am2910 ~CCEN - FOR EXPANDED INSTRUCTION SET           [10]
;**************************************************************
;
CCEN:           EQU     B#0     ; TESTING IS NORMAL (~CC ENABLED)
CCDIS:          EQU     B#1     ; ALL TESTS UNCONDITIONALLY TRUE
;
;
;***************************************************************
; Am2910 RLD - FOR LOADING REGISTER/COUNTER FROM Di     [11]
;***************************************************************
;
RLD:            EQU     B#0     ; LOAD REGISTER FROM Di
NORLD:          EQU     B#1     ; NO FORCED REGISTER LOAD (INSTRUCTIONS CONTROL IT)
;
;
;******************************
; EXAMPLE SUB STATEMENTS
;******************************
;
SEQR:   SUB     4VH#E,  3VQ#0,  12V$X
; defaults      CONT    PASS     #
;                [7]     [8]
;
CCU:    SUB     4VH#E,  1VB#0,  3VQ#0,  12V$X,  3VQ#3,    1VB#1
; defaults      CONT    CCEN    PASS    #       PIPELINE  NORLD
;               [7]     [10]    [8]             [9]       [11]
;
```

```
;
;
;*****************************************************************
; OTHER CONTROLS
;*****************************************************************
;
STATUS:         EQU     B#0             ; LOAD STATUS REGISTER
NO.STAT:        EQU     B#1
;
;
;***********************************************************************
; SHIFT INTERCONNECTION MULTIPLEXER NETWORK (USE Am2904)          [12]
;***********************************************************************
;
ROTUP:          DEF     60X,    H#0     ; R0 <- R3
ROTDN:          DEF     60X,    H#1     ; R3 <- R0
ROTDBLDN:       DEF     60X,    H#2     ; R3 <- Q0, Q3 <- R0
ROTDBLUP:       DEF     60X,    H#3     ; R0 <- Q3, Q0 <- R3
SHIFTUP:        DEF     60X,    H#4     ; R0 <- 0
SHIFTDN:        DEF     60X,    H#5     ; R3 <- 0
;
; overlay these DEF statements with others to complete microword
; (see format description)
;
;
; Note that comma positional notation is a bother to some - change the
; previous statements to DEF statements and overlay to avoid the problem
; or define special combinational DEF statements.  This file has options.
```

```
;
;
;
; T O W A R D   M O R E   R E A D A B L E   C O D E
;
;
;*****************************************************************************
; EXAMPLE DEF STATEMENT - TOTALLY DEFINED - NO SUB REFERENCE - COMMA-POSITIONAL
;*****************************************************************************
;
;
;                         F O R M A T
;----------------------------------------------------------------------------------------------
;          2910   COND   BRANCH SOURCE FUNCT  DEST   CARRY   Ra     Rb     ABMUX         SHIFT
;          INSTR  MUX    ADDR     A      L      U      in    ADDR   ADDR   SEL           SEL
CPU:  DEF  4VH#E, 3VQ#0, 12V$X, 3VQ#0, 3VQ#0, 3VQ#1, 2VB#00, 4VH#0, 4VH#0, 2VB#00, 20X,  4VX
;          CONT   PASS   #      AQ     ADD    NOP    NOC     R0     R0     PIPE
;----------------------------------------------------------------------------------------------
;
;
;
;******************************************************************
; EXAMPLE DEF STATEMENTS - USING SUB REFERENCES - COMMA-POSITIONAL
;******************************************************************
;
;
COMPUTER:   DEF   SEQR,  ALU,  REGS,  24X
;
FASTONE:    DEF   CCU,   ALU,  REGS,  19X
;
POWERFUL:   DEF   CCU,   ALU2, REGS,  18X
;
;
;
; SAMPLE .SRC STATEMENT ( A COMMENT IN THIS FILE )
;--------------------------------------------------------
;
; COMPUTER ,,, AB, ADD, RAMF, CIN, R3, R2
;
;
```

```
;
;
;*****************************************************************************************
; EXAMPLE DEF STATEMENTS - TOWARD MORE OVERLAY ORIENTATION - USING SUB REFERENCES
;*****************************************************************************************
;
;
COMP:   DEF  19X,  ALU,  REGS,  24X
;
CTRL:   DEF  SEQR, 45X
;
;
; SAMPLE .SRC STATEMENT ( A COMMENT IN THIS FILE )
;--------------------------------------------------
;
; CTRL & COMP AB, ADD, RAMF, R3, R2
;
;
;
;**********************************************************************
; EXAMPLE DEF STATEMENTS - HIGHLY SPECIALIZED
;**********************************************************************
;
;
CONTIN:   DEF  CONT, 60X
;
GOSUB:    DEF  CJS, PASS, 57X
;
JMPMAP:   DEF  JMAP, 60X
;
ADD.REG:  DEF  19X, AB, ADD, RAMF, NOC, 4VH#0, 4VH#0, 2VB#00, 24X
;                                        R0     R0      PIPE
;
; SAMPLE .SRC STATEMENT ( A COMMENT IN THIS FILE )
;-----------------------------------------------------
;
; CONTIN & ADD.REG R3, R2
;
;
```

```
;
;***************************************************************************
; ADDITIONAL SPECIALIZED DEF STATEMENTS FOR THE SIMPLE COMPUTER
; - PARTIAL MICROWORD ONLY
;***************************************************************************
;
SEQ:      DEF  4VH#E,  3VQ#0,  12V$X,  45X
NEXT:     DEF  CJP,    PASS,   12V$X,  45X
;
DATAPASS:DEF  19X,     ZA,   OR,   NOP,     NOC,   4X,   4X,  INSTR,   24X
IO:       DEF  19X,    3VX,  OR,  RAMF,     NOC,  4VX,  4VX,  2VB#00,  24X
REGF:     DEF  19X,    AB,  3VX,  RAMF,     NOC,  4VX,  4VX,  INSTR,   24X
REGCF:    DEF  19X,    AB,  3VX,  RAMF,     CIN,  4VX,  4VX,  INSTR,   24X
REGA:     DEF  19X,    ZB,  3VX,  RAMA, 2VB#00,  4VX,  4VX,  2VB#00,  24X
NOOP:     DEF  19X,    ZA,   OR,   NOP,     NOC,  34X
ZEROPC:   DEF  19X,    AB, EXOR,  RAMF,     NOC,  R15,  R15,  PIPE,    24X
;
;************************************************************************
; ANOTHER APPROACH TO CONTROLS - USE DEF STATEMENTS WITH NO EQUs
; - MUST DOCUMENT VERY WELL FOR CLARITY
; - MUST CHOSE MEANINGFUL NAMES
; - BEST FOR CLUSTERS OF CONTROLS WITH OFTEN-REPEATED PATTERNS
;************************************************************************
; MEMORY AND DATA BUS CONTROLS DEF STATEMENTS
;
; FORMAT OF THESE CONTROLS
;------------------------------------------------------------------------------
; OEY(Am2901)  MEMORY CONTROL  MEM DATA    MAR        DATA OUT   DATA IN   INSTR REG
;  EN  DIS     READ WRITE DIS   OUT IN   LD EN DIS    EN DIS     EN DIS    LOAD  DIS
;   0   1       01   10   11     0   1   01 00  11     0  1       0  1      0     1
;
;------------------------------------------------------------------------------
;
LOADMAR:       DEF     40X,    B#011001111,    15X     ; LOAD REG -> MAR
RDMEMALU:      DEF     40X,    B#101000101,    15X     ; READ MEM <MAR> -> ALU REG
WALU2MEM:      DEF     40X,    B#010100111,    15X     ; WRITE ALU REG -> <MAR>
DATA2ALU:      DEF     40X,    B#111011101,    15X     ; LOAD DATA -> REG
ALU2DATA:      DEF     40X,    B#011011011,    15X     ; LOAD REG -> DATA BUS
NOALUOUT:      DEF     40X,    B#111011111,    15X     ; NO ALU OUTPUT
READINSTR:     DEF     40X,    B#101011110,    15X     ; <MAR> -> INSTR REG
;
;
;
END

TOTAL PHASE 1 ERRORS =    0
```

```
       ;
       ;
       ;**************************************************************
       ; E D 2 9 0 0 A  -  E D S Y S 2 9   E X A M P L E S
       ;
       ; VARIOUS MISC. EXAMPLES OF MICROPROGRAMMING THE Am2910-Am2901
       ;**************************************************************
       ;
       ; USING "CPU" DEF STATEMENT - COMMA-POSITIONAL (DETAILED)
       ;-------------------------------------------------------------------------------
       ; 2910  COND  BRANCH  SOURCE  FUNC  DEST  CARRY RA   RB   ABMUX  SHIFT
       ; INSTR MUX   ADDR      A      L     U     IN   ADDR ADDR SEL    SEL
       ;   4    3     12       3      3     3     2    4    4    2      4
       ;
       ;--------------------------------------------------------------------------------
0000 REGADD::CPU , , ,        AB, ADD,  RAMF, NOC, R3, R4                  ; R3 + R4 -> R4
       ;--------------------------------------------------------------------------------
0001 ADD3::  CPU , , ,        AB, ADD,  QREG, NOC, R3, R4                  ; R3 + R4 -> R5
0002         CPU , , ,        ZQ, ADD,  RAMF, NOC,   , R5
       ;--------------------------------------------------------------------------------
0003 LDZERO::CPU , , ,        AB, EXOR, RAMF, NOC, R7, R7                  ; 0 -> R7
       ;-----------------------------------------------------------------------------------
0004 SHFT::  CPU , , ,        ZA, ADD,  RAMU, CIN, R3, R4 & SHIFTUP        ; 2*(R3 + 1) -> R4
       ;-----------------------------------------------------------------------------------
0005 D2Q::   CPU , , ,        DZ, ADD,  QREG, NOC                          ; DATA -> Q
       ;------------------------------------------------------------------------------
0006 D2R::   CPU , , ,        DA, ADD,  RAMF, NOC,   , R5                  ; DATA -> R5
       ;--------------------------------------------------------------------------------
0007 DQ2R::  CPU , , ,        DQ, ADD,  RAMF, NOC,   , R5                  ; DATA + Q -> R5
       ;--------------------------------------------------------------------------------
0008 INOR::  CPU , , ,        AB,  OR,  RAMF, NOC, R2, R3                  ; R2 NOR R3 -> R3
0009         CPU , , ,        AB, NOTRS,QREG, NOC, R3, R1                  ; where R1 = all 1s
000A         CPU , , ,        ZQ,  OR,  RAMF, NOC,   , R3
       ;------------------------------------------------------------------------------------
000B INAND:: CPU , , ,        AB, AND,  RAMF, NOC, R2, R3                  ; R2 NAND R3 -> R3
000C         CPU , , ,        AB, NOTRS,QREG, NOC, R3, R1
000D         CPU , , ,        ZQ,  OR,  RAMF, NOC,   , R3
       ;------------------------------------------------------------------------------------
000E ROT6D:: CPU , , ,        ZA,  OR,  RAMD, NOC, R6, R6, PIPE &ROTDN     ; ROTATE R6 1 DOWN
       ;-----------------------------------------------------------------------------------
000F ROT6U2::CPU , , ,        AB, ADD,  RAMU, CIN, R6, R6, PIPE &ROTUP     ; ROTATE R6 2 UP
       ;-----------------------------------------------------------------------------------
0010 INCR6:: CPU , , ,        ZA, ADD,  RAMF, CIN, R6, R6                  ; R6 + Cin -> R6
       ; OR
0011         CPU , , ,        ZB, ADD,  RAMF, CIN,   , R6
       ;-----------------------------------------------------------------------------------------
0012 SWAP::  CPU LDCT,, H#2, AB, ADD,  RAMU, IC,  R6, R6, PIPE &ROTUP     ; BYTE SWAP (16 BITS)
0013         CPU RPCT,,SWAP, AB, ADD,  RAMU, IC,  R6, R6, PIPE &ROTUP
       ;---------------------------------------------------------------------------------------
0014 PCMAR:: CPU , , ,        ZB, ADD,  RAMA, CIN, R15, R15
       ;     PC -> MAR; PC + 1 -> PC; PC = R15
       ;---------------------------------------------------------------------------------------
```

```
     ;
     ;
     ;*************************************************************************
     ; MICROCODE FOR A SIMPLE COMPUTER - PARTIAL MICROWORD (CCU-ALU ONLY)
     ;*************************************************************************
     ; COMMA-POSITIONAL NOTATION
     ; REGISTER ADDRESSING AND IMPLIED ADDRESSING BOTH
     ;*************************************************************************
     ;
0015 ZEROPC:  CPU    , , ,             AB, EXOR, RAMF,  NOC, R15, R15
0016 LDMAR:   CPU    , , ,             ZB, ADD,  RAMA,  CIN, R15, R15
0017 READMEM: CPU    , , ,               ,    ,  NOP
0018 DECODE:  CPU JMAP, , ,              ,    ,  NOP
     ;
0019 INA::    CPU  CJP, PASS, LDMAR,   DZ,  OR,  RAMF,  NOC, R10, R10, PIPE
001A INR::    CPU  CJP, PASS, LDMAR,   DZ,  OR,  RAMF,  NOC,    ,    , INSTR
001B LDA::    CPU     ,     ,      ,   ZA,  OR,  NOP,   NOC,    ,    , INSTR
001C          CPU  CJP, PASS, LDMAR,   DZ,  OR,  RAMF,  NOC,    , R10, PIPE
001D LDR::    CPU     ,     ,      ,   ZA,  OR,  NOP,   NOC,    ,    , INSTR
001E          CPU  CJP, PASS, LDMAR,   DZ,  OR,  RAMF,  NOC,    ,    , INSTR
001F ADDRR::  CPU  CJP, PASS, LDMAR,   AB, ADD,  RAMF,  NOC,    ,    , INSTR
0020 ADCRR::  CPU  CJP, PASS, LDMAR,   AB, ADD,  RAMF,  CIN,    ,    , INSTR
0021 ORR::    CPU  CJP, PASS, LDMAR,   AB,  OR,  RAMF,     ,    ,    , INSTR
0022 XORR::   CPU  CJP, PASS, LDMAR,   AB, EXOR, RAMF,     ,    ,    , INSTR
0023 IFZ::    CPU  CJP, ZERO, GOTOR,     ,    ,  NOP
0024          CPU  CJP, PASS, READMEM, ZB, ADD,  RAMA,  CIN, R15, R15, PIPE
0025 GOTOR::  CPU  CJP, PASS, LDMAR,   ZA,  OR,  RAMF,  NOC,    , R15, AIR
0026 OUTACC:: CPU  CJP, PASS, LDMAR,   ZB,  OR,  RAMF,  NOC,    , R10, PIPE
0027 OUTR::   CPU  CJP, PASS, LDMAR,   ZB,  OR,  RAMF,  NOC,    ,    , INSTR
     ; etc.
     ;
     ; note: output enable for Am2901 is not shown, nor are bus controls, memory controls
     ;
     ;This method of writing code can become tedious.  The functions are very
     ; clearly defined.  Comma-positioning bothers some programmers.  Note the
     ; obvious repetitive sections of microinstructions  i.e., CJP, PASS, LDMAR
     ; Observation of this type of repetition leads to the specialized DEF
     ; statements and the use of overlay to reduce the programming effort.
     ;
```

```
       ;
       ;
       ;************************************************************************
       ; SIMPLE COMPUTER - PARTIAL MICROWORD (CCU-ALU AND BUS CONTROLS ONLY)
       ;************************************************************************
       ;
       ; SAME CODE AS ABOVE (LABELS CHANGED TO MAKE THEM UNIQUE)
       ; USES SPECIALIZED DEF STATEMENTS TO MAKE CODE MORE READABLE
       ;
       ;************************************************************************
       ;
0028 ZPC:     CONTIN  &  ZEROPC                                         & NOALUOUT
0029 LODMAR:  CONTIN  &  REGA ADD, CIN, R15, R15, PIPE                  & LOADMAR
002A RDMEM:   CONTIN  &  NOOP                                           & READINSTR
002B DECOD:   JMPMAP  &  NOOP                                           & NOALUOUT
       ;
002C DATAINA::        NEXT LODMAR     &  IO    DZ, R10, R10             & DATA2ALU
002D DATAINR::        NEXT LODMAR     &  IO    DZ,    ,    , INSTR      & DATA2ALU
002E LOADA::          CONTIN          &  DATAPASS                       & LOADMAR
002F                  NEXT LODMAR     &  IO    DZ,    , R10             & RDMEMALU
0030 LOADR::          CONTIN          &  DATAPASS                       & LOADMAR
0031                  NEXT LODMAR     &  IO    DZ,    ,    , INSTR      & RDMEMALU
0032 ADDREG::         NEXT LODMAR     &  REGF  ADD                      & NOALUOUT
0033 ADCREG::         NEXT LODMAR     &  REGCF ADD                      & NOALUOUT
0034 ORREG::          NEXT LODMAR     &  REGF  OR                       & NOALUOUT
0035 XORREG::         NEXT LODMAR     &  REGF  EXOR                     & NOALUOUT
0036 IFZERO::         SEQ CJP, ZERO, GOTOREG  & NOOP                     & NOALUOUT
0037                  SEQ CJP, PASS, RDMEM    & NOOP                     & LOADMAR
0038 GOTOREG::        NEXT LODMAR     &  IO    ZA,    , R15, AIR        & NOALUOUT
0039 OUTA::           NEXT LODMAR     &  IO    ZB,    , R10             & ALU2DATA
003A OUTREG::         NEXT LODMAR     &  IO    ZB,    ,    , INSTR      & ALU2DATA
       ; etc.
       ;
       ;
       ;
       END
```

```
0000 1110000XXXXXXXXX XXX0010000110000 11010000XXXXXXXX XXXXXXXXXXXXXXXX
0001 1110000XXXXXXXXX XXX0010000000000 11010000XXXXXXXX XXXXXXXXXXXXXXXX
0002 1110000XXXXXXXXX XXX0100000110000 00010100XXXXXXXX XXXXXXXXXXXXXXXX
0003 1110000XXXXXXXXX XXX0011100110001 11011100XXXXXXXX XXXXXXXXXXXXXXXX
0004 1110000XXXXXXXXX XXX1000001110100 11010000XXXXXXXX XXXXXXXXXXXX0100
0005 1110000XXXXXXXXX XXX1110000000000 00000000XXXXXXXX XXXXXXXXXXXXXXXX
0006 1110000XXXXXXXXX XXX1010000110000 00010100XXXXXXXX XXXXXXXXXXXXXXXX
0007 1110000XXXXXXXXX XXX1100000110000 00010100XXXXXXXX XXXXXXXXXXXXXXXX
0008 1110000XXXXXXXXX XXX0010110110000 10001100XXXXXXXX XXXXXXXXXXXXXXXX
0009 1110000XXXXXXXXX XXX0011010000000 11000100XXXXXXXX XXXXXXXXXXXXXXXX
000A 1110000XXXXXXXXX XXX0100110110000 00001100XXXXXXXX XXXXXXXXXXXXXXXX
000B 1110000XXXXXXXXX XXX0011000110000 10001100XXXXXXXX XXXXXXXXXXXXXXXX
000C 1110000XXXXXXXXX XXX0011010000000 11000100XXXXXXXX XXXXXXXXXXXXXXXX
000D 1110000XXXXXXXXX XXX0100110110000 00001100XXXXXXXX XXXXXXXXXXXXXXXX
000E 1110000XXXXXXXXX XXX1000111010001 10011000XXXXXXXX XXXXXXXXXXXX0001
000F 1110000XXXXXXXXX XXX0010001110101 10011000XXXXXXXX XXXXXXXXXXXX0000
0010 1110000XXXXXXXXX XXX1000000110101 10011000XXXXXXXX XXXXXXXXXXXXXXXX
0011 1110000XXXXXXXXX XXX0110000110100 00011000XXXXXXXX XXXXXXXXXXXXXXXX
0012 1100000000000000 0100010001111001 10011000XXXXXXXX XXXXXXXXXXXX0000
0013 1001000000000010 0100010001111001 10011000XXXXXXXX XXXXXXXXXXXX0000
0014 1110000XXXXXXXXX XXX0110000100111 11111100XXXXXXXX XXXXXXXXXXXXXXXX
0015 1110000XXXXXXXXX XXX0011100110011 11111100XXXXXXXX XXXXXXXXXXXXXXXX
0016 1110000XXXXXXXXX XXX0110000100111 11111100XXXXXXXX XXXXXXXXXXXXXXXX
0017 1110000XXXXXXXXX XXX0000000010000 00000000XXXXXXXX XXXXXXXXXXXXXXXX
0018 0010000XXXXXXXXX XXX0000000010000 00000000XXXXXXXX XXXXXXXXXXXXXXXX
0019 0011000000000010 1101110110110010 10101000XXXXXXXX XXXXXXXXXXXXXXXX
001A 0011000000000010 1101110110110000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
001B 1110000XXXXXXXXX XXX1000110010000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
001C 0011000000000010 1101110110110000 00101000XXXXXXXX XXXXXXXXXXXXXXXX
001D 1110000XXXXXXXXX XXX1000110010000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
001E 0011000000000010 1101110110110000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
001F 0011000000000010 1100010000110000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
0020 0011000000000010 1100010000110100 00000011XXXXXXXX XXXXXXXXXXXXXXXX
0021 0011000000000010 1100010110110000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
0022 0011000000000010 1100011100110000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
0023 0011010000000100 1010000000010000 00000000XXXXXXXX XXXXXXXXXXXXXXXX
0024 0011000000000010 1110110000100111 11111100XXXXXXXX XXXXXXXXXXXXXXXX
0025 0011000000000010 1101000110110000 00111110XXXXXXXX XXXXXXXXXXXXXXXX
0026 0011000000000010 1100110110110000 00101000XXXXXXXX XXXXXXXXXXXXXXXX
0027 0011000000000010 1100110110110000 00000011XXXXXXXX XXXXXXXXXXXXXXXX
0028 1110XXXXXXXXXXXX XXX0011100110011 1111110011101111 1XXXXXXXXXXXXXXX
0029 1110XXXXXXXXXXXX XXX0110000100111 1111110001100111 1XXXXXXXXXXXXXXX
002A 1110XXXXXXXXXXXX XXX10001100100XX XXXXXXXX10101111 0XXXXXXXXXXXXXXX
002B 0010XXXXXXXXXXXX XXX10001100100XX XXXXXXXX11101111 1XXXXXXXXXXXXXXX
002C 0011000000000101 0011110110110010 1010100011101110 1XXXXXXXXXXXXXXX
002D 0011000000000101 00111101101100XX XXXXXX1111101110 1XXXXXXXXXXXXXXX
002E 1110XXXXXXXXXXXX XXX10001100100XX XXXXXX1101100111 1XXXXXXXXXXXXXXX
002F 0011000000000101 00111101101100XX XX10100010100010 1XXXXXXXXXXXXXXX
0030 1110XXXXXXXXXXXX XXX10001100100XX XXXXXX1101100111 1XXXXXXXXXXXXXXX
0031 0011000000000101 00111101101100XX XXXXXX1110100010 1XXXXXXXXXXXXXXX
0032 0011000000000101 00100100001100XX XXXXXX1111101111 1XXXXXXXXXXXXXXX
0033 0011000000000101 00100100001101XX XXXXXX1111101111 1XXXXXXXXXXXXXXX
0034 0011000000000101 00100101101100XX XXXXXX1111101111 1XXXXXXXXXXXXXXX
0035 0011000000000101 00100111001100XX XXXXXX1111101111 1XXXXXXXXXXXXXXX
0036 0011010000000111 00010001100100XX XXXXXXXX11101111 1XXXXXXXXXXXXXXX
0037 0011000000000101 01010001100100XX XXXXXXXX01100111 1XXXXXXXXXXXXXXX
0038 0011000000000101 00110001101100XX XX11111011101111 1XXXXXXXXXXXXXXX
0039 0011000000000101 00101101101100XX XX10100001101101 1XXXXXXXXXXXXXXX
```

```
003A 0011000000000101 00101101101100XX XXXXXX1101101101 1XXXXXXXXXXXXXXX
```

```
 ENTRY POINTS

  ADCREG    0033
  ADCRR     0020
  ADD3      0001
  ADDREG    0032
  ADDRR     001F
  D2Q       0005
  D2R       0006
  DATAINA   002C
  DATAINR   002D
  DQ2R      0007
  GOTOR     0025
  GOTOREG   0038
  IFZ       0023
  IFZERO    0036
  INA       0019
  INAND     000B
  INCR6     0010
  INOR      0008
  INR       001A
  LDA       001B
  LDR       001D
  LDZERO    0003
  LOADA     002E
  LOADR     0030
  ORR       0021
  ORREG     0034
  OUTA      0039
  OUTACC    0026
  OUTR      0027
  OUTREG    003A
  PCMAR     0014
  REGADD    0000
  ROT6D     000E
  ROT6U2    000F
  SHFT      0004
  SWAP      0012
  XORR      0022
  XORREG    0035
```

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.4
SAMPLE AM2910 AND AM2901 MICROCODE

 SYMBOLS

  AB         0001
  ADCREG     0033
  ADCRR      0020
  ADD        0000
  ADD3       0001
  ADDREG     0032
  ADDRR      001F
  AIR        0002
  AND        0004
  AQ         0000
  BIR        0001
  CCDIS      0001
  CCEN       0000
  CIN        0001
  CJP        0003
  CJPP       000B
  CJS        0001
  CJV        0006
  CN         0001
  CONT       000E
  CRTN       000A
  D2Q        0005
  D2R        0006
  DA         0005
  DATAINA    002C
  DATAINR    002D
  DECOD      002B
  DECODE     0018
  DQ         0006
  DQ2R       0007
  DZ         0007
  EXNOR      0007
  EXOR       0006
  GOTOR      0025
  GOTOREG    0038
  IC         0002
  IFZ        0023
  IFZERO     0036
  INA        0019
  INAND      000B
  INCR6      0010
  INOR       0008
  INR        001A
  INSTR      0003
  IZ         0003
  JMAP       0002
  JRP        0007
  JSRP       0005
  JZ         0000
  LDA        001B
  LDCT       000C
  LDMAR      0016
  LDR        001D
  LDZERO     0003
  LOADA      002E
  LOADR      0030
```

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.4
SAMPLE AM2910 AND AM2901 MICROCODE

  LODMAR     0029
  LOOP       000D
  MAP        0005
  NEG        0003
  NO.STAT    0001
  NOC        0000
  NOP        0001
  NORLD      0001
  NOTRS      0005
  OEYDIS     0001
  OEYEN      0000
  OR         0003
  ORR        0021
  ORREG      0034
  OUTA       0039
  OUTACC     0026
  OUTR       0027
  OUTREG     003A
  OVR        0004
  PASS       0000
  PCMAR      0014
  PIPE       0000
  PIPELINE   0003
  PUSH       0004
  QREG       0000
  R0         0000
  R1         0001
  R10        000A
  R11        000B
  R12        000C
  R13        000D
  R14        000E
  R15        000F
  R2         0002
  R3         0003
  R4         0004
  R5         0005
  R6         0006
  R7         0007
  R8         0008
  R9         0009
  RAMA       0002
  RAMD       0005
  RAMF       0003
  RAMQD      0004
  RAMQU      0006
  RAMU       0007
  RDMEM      002A
  READMEM    0017
  REGADD     0000
  RFCT       0008
  RLD        0000
  ROT6D      000E
  ROT6U2     000F
  RPCT       0009
  SHFT       0004
  STATUS     0000
  SUBR       0001
```

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.4
SAMPLE AM2910 AND AM2901 MICROCODE

  SUBS       0002
  SWAP       0012
  TWB        000F
  VECTOR     0006
  XORR       0022
  XORREG     0035
  ZA         0004
  ZB         0003
  ZERO       0002
  ZEROPC     0015
  ZPC        0028
  ZQ         0002

 TOTAL PHASE 2 ERRORS =    0
```