Toward Structured Microprogramming with AMDASM: The Am2901C and the Am2910A

by
Donnamaie E. White

AMDASM

AMDASM is the macroassembler (intended for microcode applications) which is available on the AmSYS29/10 development system. It is part of a comprehensive package of software provided for bipolar support, including AMMAP, AMPROM, AMSCRM and DDT29.

AMDASM requires that the user define the symbols, or mnemonics, which will appear in the microinstructions of a given source program (the .SRC file). The definitions are primarily grouped in a separate file, called the definition file (the .DEF file). In addition to the mnemonic descriptions, some means of describing the microinstruction format is required.

Mnemonics are described via equate statements (EQUs) and format is descibed via substitution (SUB) and definition (DEF) statements. EQU, SUB and DEF statements appear in the definition file. Equates and "free format" (FF) microinstructions appear in the source file, along with the DEF-statement references.

The step-by-step creation of a definition file is shown on the following pages.

The Am2901

The first Advanced Micro Devices high speed, bipolar, bit-slice RALU (registered ALU) was the Am2901 (see figure 1). The Am2901 is a 4-bit wide slice that includes a set of 16 RAM registers plus a 17th register, called the Q register, for double precision operations. The ALU is capable of binary-two's complement operations.

The Am2901 functions are a subset of those available on the more powerful Am2903 and Am29203 RALUs. Because the Am2901 is a simpler device, it is relatively easy to write microcode with its instruction set. Coupled with the Am2910 microprogram controller, the Am2901 will be used as the demonstration vehicle for a discussion of AMDASM.

Am2901

MICROPROCESSOR SLICE BLOCK DIAGRAM

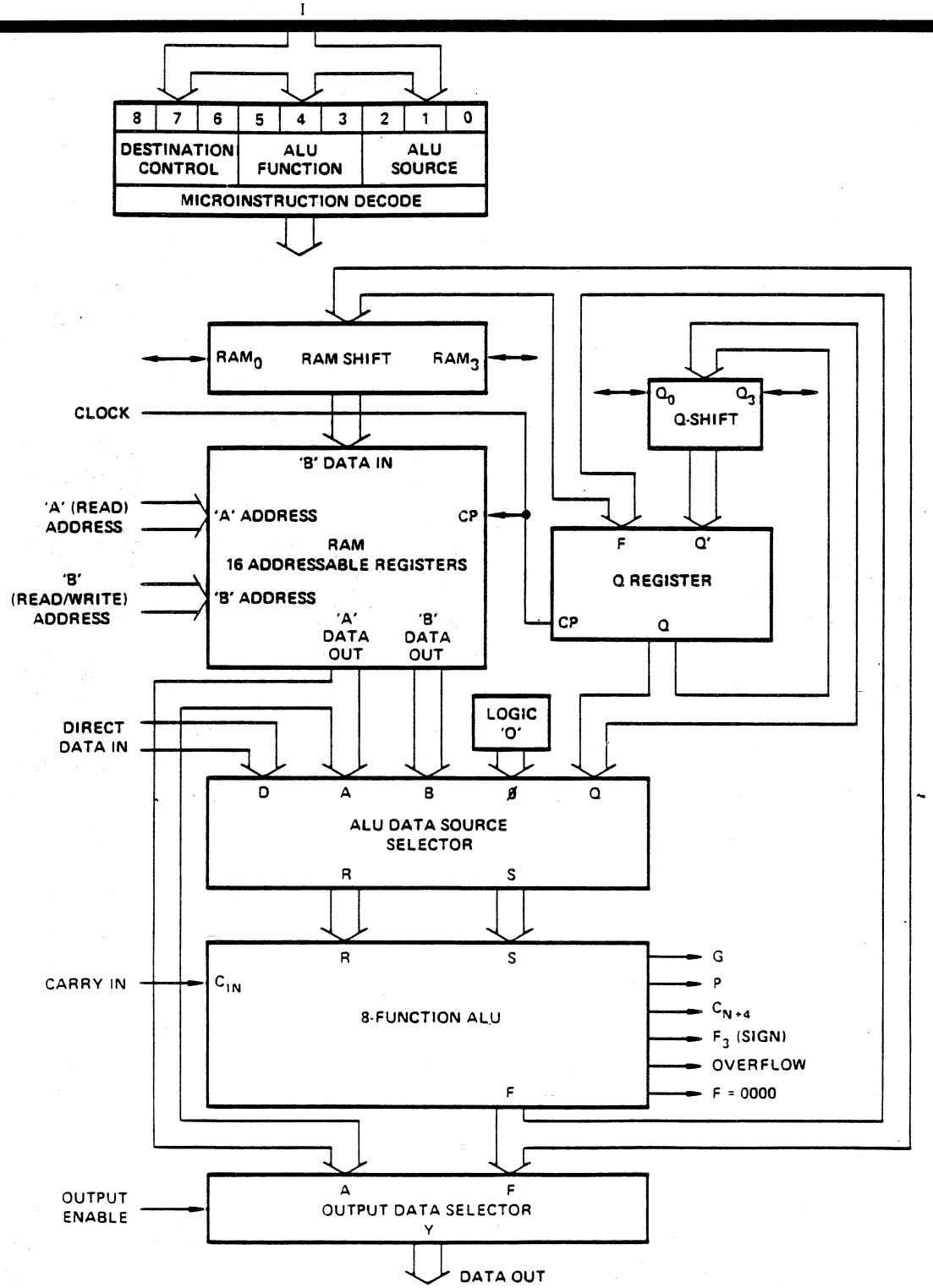
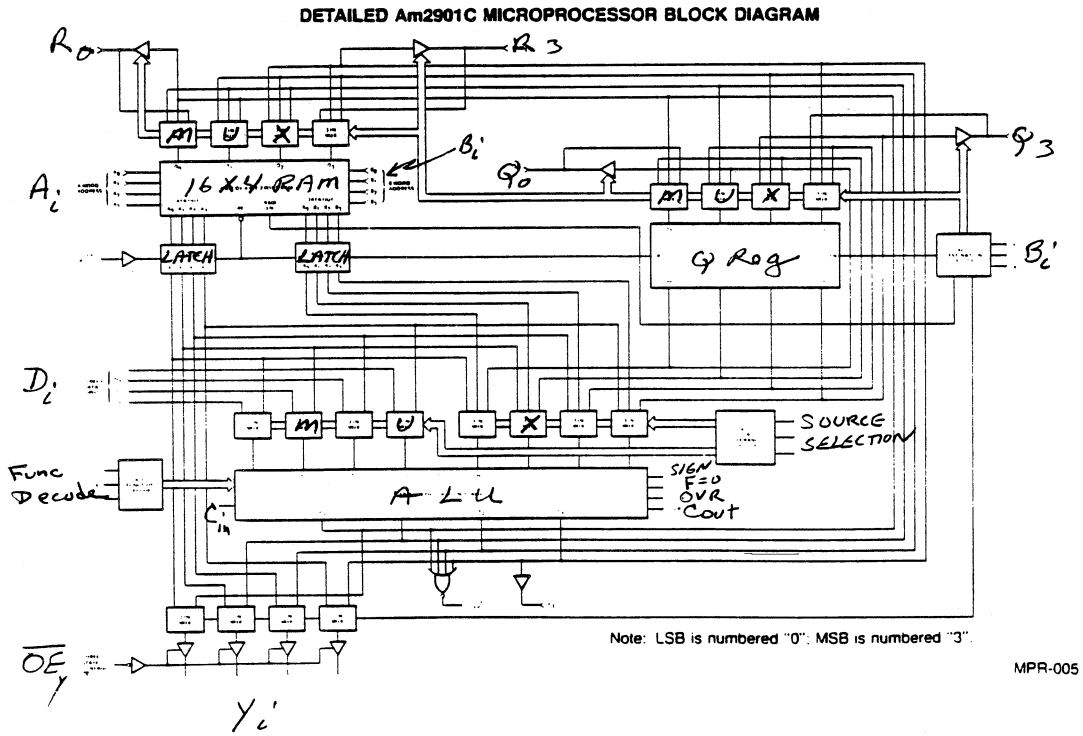


Fig 1

The Am2901 Registers

The Am2901 "scratchpad" registers are a dual-port memory block, i.e., the 16 registers may be addressed two at a time. The outputs, RAMA and RAMB, input directly into input multiplexers (the ALU Data Source Selector of figure 1), which in turn input into the ALU. The registers are addressed via two sets of four address lines, one set per port. The B port address is the source-destination (read-write) address while the A address is a source-only address.

The address lines are labeled Ai and Bi in the figure below. To allow the microinstruction to specify the register addresses, two fields of four bits each are required, labeled RA ADDR and RB ADDR. Each bit represents one of the address lines.



Source Operand Selection

The other inputs to the multiplexers at the ALU input are: outside data (D), the Q register, and logical zero (Z) (see figure 2). The multiplexers are controlled via the Am2901 instruction lines 2,1,0. These lines allow the user to select one of 8 possible source-operand pairs: AQ, AB, ZQ, ZB, ZA, DA, DQ, and DZ. The mnemonics (A, B, Q, D, Z) are specified in the Am2901 data sheet, along with the required microcode pattern.

To program an Am2901-based CPU, the microword must contain a field which supplies values (1 or 0 or even X) to the instruction lines in order to select an appropriate source operand pair. To specify A and B for example:

```
                  ---------------------
        AB:       o o o |0|0|1| o o o
                  ---------------------
                          3 2 1
To select DQ:

                  ---------------------
        DQ:       o o o |1|1|0| o o o
                  ---------------------
                          3 2 1
```

	MICRO CODE				ALU SOURCE OPERANDS	
Mnemonic	I_2	I_1	I_0	Octal Code	R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	O	Q
ZB	L	H	H	3	O	B
ZA	H	L	L	4	O	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	O

ALU Source Operand Control.

Mnemonic	MICRO CODE I_5	I_4	I_3	Octal Code	ALU Function	SYMBOL
ADD	L	L	L	0	R Plus S	R + S
SUBR	L	L	H	1	S Minus R	S − R
SUBS	L	H	L	2	R Minus S	R − S
OR	L	H	H	3	R OR S	$R \vee S$
AND	H	L	L	4	R AND S	$R \wedge S$
NOTRS	H	L	H	5	$\overline{R}$ AND S	$\overline{R} \wedge S$
EXOR	H	H	L	6	R EX-OR S	$R \forall S$
EXNOR	H	H	H	7	R EX-NOR S	$\overline{R \forall S}$

ALU Function Control.

Function Selection

The operations that can be performed on the source operand pair are also specified on the Am2901 data sheet, along with their microcode bit pattern. ADD is specified by "000" and EXOR is specified by "110".

The bit pattern for function selection refers to instruction lines 5, 4, 3. Another field in the microword is required to supply control to these lines.

Destination Selection

The possible destinations of the result are summarized in a third table on the data sheet. The destination is selected via instruction lines 8, 7, 6, and these lines require another field.

Mnemonic	MICRO CODE I_8	I_7	I_6	Octal Code	RAM FUNCTION Shift	Load	Q-REG. FUNCTION Shift	Load	Y OUTPUT	RAM SHIFTER RAM_0	RAM_3	Q SHIFTER Q_0	Q_3
QREG	L	L	L	0	X	NONE	NONE	F → Q	F	X	X	X	X
NOP	L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
RAMA	L	H	L	2	NONE	F → B	X	NONE	A	X	X	X	X
RAMF	L	H	H	3	NONE	F → B	X	NONE	F	X	X	X	X
RAMQD	H	L	L	4	DOWN	F/2 → B	DOWN	Q/2 → Q	F	F_0	IN_3	Q_0	IN_3
RAMD	H	L	H	5	DOWN	F/2 → B	X	NONE	F	F_0	IN_3	Q_0	X
RAMQU	H	H	L	6	UP	2F → B	UP	2Q → Q	F	IN_0	F_3	IN_0	Q_3
RAMU	H	H	H	7	UP	2F → B	X	NONE	F	IN_0	F_3	X	Q_3

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state
B = Register Addressed by B inputs.
UP is toward MSB, DOWN is toward LSB.

ALU Destination Control.

The Microword

Each of these ALU related fields must be present in the same microword and the three fields operate in parallel. The Am2901 portion of the microword consists of:

A source	L function	U destination	ADDRESSES RA ADDR	RB ADDR
$I_3I_2I_1$	$I_5I_4I_3$	$I_8I_7I_6$	$A_3A_2A_1A_0$	$B_3B_2B_1B_0$

To these five fields add a carry-in select field, typically of one or two bits, and an ABMUX selection field, usually two bits to allow flexible source selection. The latter field is required because, in most general-purpose CPUs, the A and B register addresses can be supplied by either the microword (microinstrucion), by the machine-level command (macroinstrucion), or some combination of both. The format now becomes:

field:	source A	funct L	dest U	carry in	RA addr	RB addr	ABMUX SEL	
size:	3	3	3	2	4	4	2	= 21 bits

Depending upon the hardware requirements, an output enable field (OEYen) may be required (between dest and carry).

Describing the Microword to AMDASM

The Am2901 portion of the microword developed so far can be described to AMDASM via detailed field descriptions in either: one or more "SUB" (substitute) statements, one or more "DEF" (definition) statements, or by some combination of these techniques. The choice is a matter of personal preference and code readability, with the emphasis on the latter.

SUB Statement

An AMDASM SUB statement is a substitution statement in that it may appear in more than one DEF statement in place of the more tediously defined individual fields. It is most useful when many different DEF statements are being defined and parts of them are identical. It can also be used to make DEF statements themselves more readable.

SUB statements are constructed of constant and variable field definitions (described later) whose total bit width is less than a complete microword. SUB statements may also reference symbolic constants.

```
;
;*****************************
; EXAMPLE SUB STATEMENTS
;*****************************

;
ALU2:   SUB      3VQ#0,  3VQ#0,  3VQ#1,  1VB#0,  2VB#00
; defaults         AQ      ADD     NOP     OEYEN   NOC

;
REGS:   SUB      4VH#0,  4VH#0,  2VB#00
; defaults         R0      R0      PIPE
```

DEF Statement

An AMDASM DEF statement is constructed of constant and variable field definitions whose total bit width is equal to the width of the microword. One or more DEF statements may appear in a .DEF file. They may define part of or all of a microword. Those that define part of a microword use "don't cares" as filler and are overlayed with other DEF statements in the source file to complete the microword definition. Some DEF statements may be specialized (limited variable substitution) to simplify code creation and to help make the source file more readable. A DEF statement may reference SUB statements and symbolic constants.

```
;
;*************************
; EXAMPLE DEF STATEMENTS
;*************************
;
;
COMP:    DEF  19X,  ALU,  REGS,  24X

;
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

;
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
;
```

Field Definition

The individual fields referenced earlier are defined by specifying whether or not they are constant or variable. Variable fields may have different values in different instructions (supplied in the source file statement). They are usually defined with a default value, the state the controlled lines will be left in should the microprogrammer not specify one. Default values should be selected with care to be obvious, to be the state most commonly used, or to be (at least) not harmful.

As an example, for the ALU SUB statement shown below, 3VQ#0 means a 3-bit variable field (V) with a default value of "000". If that field is the one controlling the source-select field, the default is "AQ". The same notation for the function select field would result in a default of "ADD" while in the destination field it would represent "QREG". In the statement shown below, note the comment lines which document the default values by mnemonics.

```
;
ALU:    SUB     3VQ#0,  3VQ#0,  3VQ#1,  2VB#00
; defaults       AQ      ADD     NOP     NOC
```

Symbol Table Creation

The source program is the microprogram itself. The objectives of the .DEF program is to provide the means to write the microprogram in a straight-forward and readable manner. The microprogram could be written in 1s and 0s, but as the program grows in size, the practicality of that approach would rapidly diminish. It is more desirable to create microcode using meaningful mnemonics in a format that is similiar to ordinary assembly-level programming.

Mnemonic Definition

To program in mnemonics, a symbol table must be created which pairs each mnemonic with a bit pattern. AMDASM handles the symbol table function by providing EQU (equate) statements in both the .DEF and .SRC files. The majority of the EQU statements will appear in the .DEF file.

Labels and Map Entry Points

The remaining symbols, consisting primarily of label identification, appear in the .SRC file and are created by the assembler itself. The labels are paired with the PC value (address) at which they occur. A label is a name followed by ":" or "::". The cases where "::" appears are considered entry points and are also placed in a special file for use in creating the memory map.

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

 SYMBOLS

  AB        0001
  ADCREG    002D
  ADCRR     001D
  ADD       0000
  ADDREG    002C
  ADDRR     001C
  AIR       0002
  AND       0004
  AQ        0000
  BIR       0001
  CCDIS     0001
  CCEN      0000
  CIN       0001

0019 INA::
001A INR::
001B LDA::
001C ADDRR::
001D ADCRR::
001E ORR::
001F XORR::

  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     0027
  READMEM   0017
  REGADD    0000
  RFCT      0008
  RLD       0000
  RPCT      0009
  STATUS    0000
  SUBR      0001
  SUBS      0002

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

        ;
0015 ZEROPC:
0016 LDMAR:
0017 READMEM:
0018 DECODE:
        ;
```

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

 ENTRY POINTS

  ADCREG    002D
  ADCRR     001D
  ADDREG    002C
  ADDRR     001C
  DATAINA   0029
  DATAINR   002A
  GOTOR     0022
  GOTOREG   0032
  IFREGZ    0030
  IFRZ      0020
  INA       0019
  INAND     000B
  INOR      0008
  INR       001A
  LDA       001B
  LDZERO    0003
  LOADA     002B <-------------- 002B LOADA::
  ORR       001E                 002C ADDREG::
  ORREG     002E                 002D ADCREG::
  OUTA      0033                 002E ORREG::
  OUTACC    0023                 002F XORREG::
  OUTR      0024
  OUTREG    0034
  REGADD    0000
  SWAP      0012
  XORR      001F
  XORREG    002F
```

Am2901 Equates

EQU statements are used for the mnemonic-bit pattern definitions for the Am2901. The instruction tables from the data sheet and the corresponding equates are shown on the following page.

Three items need emphasis. First, all equates for a given field should be grouped together. Second, they should appear before any SUB or DEF statement which uses that field. Third, comments should be heavily used, along with meaningful mnemonics, to document the field functions.

Indexing the EQUs

Note the "[1]", "[2]", "[3]", etc. which appears beside each equate group as well as beneath the SUB statement definition as a comment. These indexes are included as a prompt to the human reader to tie a particular set of equates to a particular field. The equated mnemonics are those that may be used in a variable field substitution in the source (.SRC) file.

Mnemonic	MICRO CODE I_2	I_1	I_0	Octal Code	ALU SOURCE OPERANDS R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	O	Q
ZB	L	H	H	3	O	B
ZA	H	L	L	4	O	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	O

Figure 2. ALU Source Operand Control.

Mnemonic	MICRO CODE I_5	I_4	I_3	Octal Code	ALU Function	SYMBOL
ADD	L	L	L	0	R Plus S	R + S
SUBR	L	L	H	1	S Minus R	S − R
SUBS	L	H	L	2	R Minus S	R − S
OR	L	H	H	3	R OR S	R ∨ S
AND	H	L	L	4	R AND S	R ∧ S
NOTRS	H	L	H	5	$\overline{R}$ AND S	$\overline{R}$ ∧ S
EXOR	H	H	L	6	R EX-OR S	R ⊻ S
EXNOR	H	H	H	7	R EX-NOR S	$\overline{R \veebar S}$

Figure 3. ALU Function Control.

Mnemonic	MICRO CODE I_8	I_7	I_6	Octal Code	RAM FUNCTION Shift	Load	Q-REG. FUNCTION Shift	Load	Y OUTPUT	RAM SHIFTER RAM_0	RAM_3	Q SHIFTER Q_0	Q_3
QREG	L	L	L	0	X	NONE	NONE	F → Q	F	X	X	X	X
NOP	L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
RAMA	L	H	L	2	NONE	F → B	X	NONE	A	X	X	X	X
RAMF	L	H	H	3	NONE	F → B	X	NONE	F	X	X	X	X
RAMQD	H	L	L	4	DOWN	F/2 → B	DOWN	Q/2 → Q	F	F_0	IN_3	Q_0	IN_3
RAMD	H	L	H	5	DOWN	F/2 → B	X	NONE	F	F_0	IN_3	Q_0	X
RAMQU	H	H	L	6	UP	2F → B	UP	2Q → Q	F	IN_0	F_3	IN_0	Q_3
RAMU	H	H	H	7	UP	2F → B	X	NONE	F	IN_0	F_3	X	Q_3

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state
B = Register Addressed by B inputs.
UP is toward MSB, DOWN is toward LSB.

Figure 4. ALU Destination Control.

```
;
;*************************************************************
; 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

Another field that appears with most ALUs is the carry-in field. This field will be 1, 2 or more bits wide, as required. The following set of equates assumes a more typical 2-bit field.

```
;
;
;************************************************************
; 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
;
;
```

Output Enable Control

The last field that might be required for the Am2901 is the output enable. It may or may not be necessary in a particular CPU. Typical equates are shown below.

```
;************************************************************
; OEy                                                     [5]
;************************************************************
;
OEYEN:          EQU     B#0     ; OUTPUT ENABLE
OEYDIS:         EQU     B#1
;
:
```

RALU SUB Statements

The SUB statements can follow immediately after the EQU statements or may be grouped at the end of the .DEF file. The following SUB statements are for the Am2901 RALU.

The first one, "ALU", shows the three instruction fields: source, function, and destination. It also shows a typical carry field. The second one, "ALU2", shows the same fields as above but adds a field for the output enable.

These two SUB statements would probably never be used in the same source file since they represent different hardware control patterns. The concept of one master definition file serving more than one source file is realistic, however, and often results in less overall documentation volume.

```
;*****************************
; 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

Addressing the registers of the RALU, discussed earlier, also requires EQU statements, one equate per register name. Since the A and B address fields address the same set of registers, only one set of register EQUs is necessary. Mnemonics may be used to supply values to more than one field.

ABMUX

Since most ALUs allow the macroinstruction to specify the registers in addition to allowing the microinstruction to address the registers, multiplexers are required for each address field. A separate field is required for the select lines to these multiplexers, named ABMUX SEL.

Register SUB Statements

SUB statements can also be defined for the register portion of the microword. The statement shown, "REGS", provides two address fields and the ABMUX field. Optionally, the register fields could have been included in the ALU or CCU SUB statements.

```
;
;
;*************************************************************
; 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]
;
```