EECC 550 - Spring 2003
Microprogramming Project
Due May 14

You are to write and submit a microprogram to interpret the following target machine instruction set on the data-flow path given in the figure using the microprogramming tool "AMISS".

AMISS is run at the command line on Grace while in the directory where your microprogram memory control file "cmemory", and test program "memory" reside as follows:

~meseec/AMISS/CPU -d

You are required to to e-mail your fully-commented "cmemory", "memory" files containing the machine instruction programs that you have used to test your control microprogram to:

"meseec@osfmail.isc.rit.edu".

Target Machine & Instruction Set

The target Accumulator-based machine has a datapath width of eight bits. However, the memory address bus width is sixteen bits (can access 65536 memory bytes).

There are three types of instructions:

(1) **Inherent instructions:** One-byte instructions, just the Opcode byte.

(2) **Immediate instructions**: Two-byte instructions. The instruction Opcode byte is followed by one additional byte of immediate data.

(3) **Memory reference instructions:** three-byte instructions. The instruction Opcode byte is followed by two bytes of address information. The high-order byte of the address appears in the byte memory location immediately following the Opcode byte.

- The condition code bits are "N" for negative, "Z" for zero, "V" for overflow, and "C" for carry.
- Condition code settings are "0" for cleared, "1" for set, "-" for no change, and "x" for possible change.

Inherent Instructions: One Byte Only

Opcode	Condition Codes:	**NZVC**
00001010 - **NOP** - No operation		- - - -
00011010 - **HALT** - Halt the machine		- - - -
00101010 - **CLA** - Clear the accumulator		0100
00111010 - **CMA** - 1's complement the accumulator		xx00
01001010 - **INCA** - Increment the accumulator by one		xx0x
01011010 - **DECA** - Decrement the accumulator by one		xx0x
01101010 - **RET** - Return from subroutine (post incrementing SP)		- - - -
01111010 - **ROLCA** - Circular shift Carrybit & Acc left 1 bit		xx0x
10001010 - **CLC** - Clear Carry Flag bit		- - - 0
10011010 - **STC** - Set Carry Flag bit		- - - 1

Store and Branch Instructions: Two additional address bytes

Opcode	Condition Code:	**NZVC**
000011z0 - **STA** - Store accumulator (use STAin for indirect)		- - - -
000111z0 - **JMP** - Unconditional branch		- - - -
001011z0 - **JEQ** - Branch if equal to zero (Z=1)		- - - -
001111z0 - **JCS** - Branch if carry (C=1)		- - - -
010011z0 - **JLT** - branch if negative (N=1)		- - - -
010111z0 - **JVS** - branch if overflow (V=1)		- - - -
011011z0 - **JSR** - jump to subroutine (push PC on stack - predecrement SP)		- - - -

Other Instructions: One or two additional bytes

Opcode	Condition Codes: **NZVC**
00000yz1 - LDA - Load Acc (use LDAim for immediate)	xx00
00010yz1 - ORA - Inclusive OR to Accumulator	xx00
00100yz1 - EOR - Exclusive OR to Accumulator	xx00
00110yz1 - AND - Logical AND to Accumulator	xx00
01000yz1 - ADD - Add to Accumulator	xxxx
01010yz1 - SUBA - Subtract from Accumulator	xxxx
01100yz1 - LDS - Load stack pointer SP with 16-bit value	- - - -

- Where the "y" bit is set to indicate that an immediate operand follows the opcode (add "im" to instruction name).
- The "z" bit is set to indicate that indirect addressing is to be performed through the location specified by the two bytes following the opcode (add "indir" to instruction name).
- Note that it is not allowed to have both the "y" bit and the "z" bit to be set simultaneously for these instructions.
- In addition to submitting your "cmemory" and " memory files by e-mail, you should submit a written report with the following items:

(1) A brief description of your approach to the assignment, and the features of your solution.

(2) Dependent RTN statements for all the instructions implemented.

(3) The resulting CPI for the different instruction types.

(4) A statement of any relevant problems or difficulties encountered during the assignment.

(5) A listing of your fully-commented "cmemory" and " memory" files as e-mailed.

(6) A description of testing procedures used and observations.

(7) Any additional remarks or conclusions you deem noteworthy of mention.

Instruction Opcodes

Inherent Instructions Opcodes

	Binary	Hex
NOP	00001010	0a
HALT	00011010	1a
CLA	00101010	2a
CMA	00111010	3a
INCA	01001010	4a
DECA	01011010	5a
RET	01101010	6a
ROLCA	01111010	7a
CLC	10001010	8a
STC	10011010	9a

Store and Branch Instructions Opcodes

	Binary	Hex	
		(Direct z=0)	(Indirect z=1)
STA	000011z0	0c	0e
JMP	000111z0	1c	1e
JEQ	001011z0	2c	2e
JCS	001111z0	3c	3e
JLT	010011z0	4c	4e
JVS	010111z0	5c	5e
JSR	011011z0	6c	6e

Other Instructions Opcodes

	Binary	Hex		
		Direct y = z =0	Indirect y=0 z =1	Immediate y=1 z=0
LDA	00000yz1	01	03	05
ORA	00010yz1	11	13	15
EOR	00100yz1	21	23	25
AND	00110yz1	31	33	35
ADD	01000yz1	41	43	45
SUBA	01010yz1	51	53	55
LDS	01100yz1	61	63	65

Test Memory Files (also posted online as separate text files)
memory.inherent

```
;  test file for Microprogramming Project
;
; test NOP
0a        ;NOP
4a        ;INCA
4a        ;INCA
4a        ;INCA
5a        ;DECA
2a        ;CLA
5a        ;DECA from 00
4a        ;INCA
3a        ;CMA
2a        ;CLA
4a        ;INCA
9a        ;STC
7a        ;ROLCA
9a        ;STC
8a        ;CLC
1a        ;HALT

;should do nothing for first instruction,
;  then increment accumulator by 3,
;  then decrement it by 1,
;  then clear accumulator,
;  then decrement by 1 resulting in FF in ACC and N=1,V=0,
;  then increment by 1 to 00 with Z,V=0
;  then ones-complement accumulator to get FF and C=0,V=0
;  then clear accumulator,
;  then increment to 1,
;  then set carry bit to 1
;  then roll carry and accumulator from 1 00000001 to 0 00000011
;  then set carry bit,
;  then clear carry bit,
;  then HALT execution
```

Test Memory Files (also posted online as separate text files)

memory.otherbranches

```
;  test file for Microprogramming Project
; test branch and store instructions
;   -specifically test JEQ,JEQind,JCS,JCSind,JLT,JLTind

4a              ;01: INCA                                                   (00)
2a              ;02: CLA                                                    (01)
2c 00 06        ;03: JEQ 0006                                               (02-04)
4a              ;04: INCA - filler (should not execute)          (05)
5a              ;05: DECA - should jump here from line 03        (06)
2c 00 0c        ;06: JEQ 000c                                               (07-09)
4a              ;07: INCA - should be executed, no jump                  (0a)
2e 00 0f        ;08: JEQind 000f                                            (0b-0d)
4a              ;09: INCA - filler (should not be executed)      (0e)
00 12           ;10: address to jump to from line 08                     (0f-10)
4a              ;11: INCA - filler (should not be executed)      (11)
5a              ;12: DECA - should jump here from line 08        (12)
2e 00 0f        ;13: JEQIND 000f                                            (13-15)
4a              ;14: INCA - should be executed, no jump                  (16)
9a              ;15: STC                                                    (17)
3c 00 1c        ;16: JCS 001c                                               (18-1a)
4a              ;17: INCA - filler (should not be executed)      (1b)
8a              ;18: CLC - should jump here from line 16         (1c)
3c 00 1c        ;19: JCS 001c                                               (1d-1f)
9a              ;20: STC - should be executed, no jump                   (20)
3e 00 24        ;21: JCSind 0024                                         (21-23)
00 27           ;22: address to jump to from line 21                     (24-25)
4a              ;23: INCA - should not be executed                       (26)
8a              ;24: CLC                                                    (27)
3e 00 24        ;25: JCSind 0024                                         (28-2a)
5a              ;26: DECA - should be executed, no jump                  (2b)
4c 00 30        ;27: JLT 0030                                               (2c-2e)
5a              ;28: DECA - filler                                          (2f)
4a              ;29: INCA - should jump here from line 27        (30)
4c 00 30        ;30: JLT 0030                                               (31-33)
5a              ;31: DECA - should be executed, no jump                  (34)
4e 00 38        ;32: JLTind 0038                                         (35-37)
00 3b           ;33: address to jump to from line 32                     (38-39)
5a              ;34: DECA - filler                                          (3a)
4a              ;35: INCA - should jump here from line 32        (3b)
4e 00 38        ;36: JLTind 0038                                         (3c-3e)
1a              ;37: HALT, no jump                                          (3f)
;execution should be followed
;  increment accumulator by one to 1
;  clear the accumulator
;  jump to address 6, line 5
;  decrement accumulator to FF
;  jump NOT taken, accumulator incremented to 0
;  jump to address 12, line 12
;  decrement accumulator to FF
;  jump NOT taken, accumulator incremented to 0
;  set the carry bit
;  jump to address 1C, line 18
;  clear the carry bit
;  jump NOT taken, set the carry bit
;  jump to address 27, line 24
;  clear the carry bit
;  jump NOT taken, decrement accumulator to FF
;  jump to address 30, line 29
;  increment accumulator to 0
;  jump NOT taken, decrement accumulator to FF
;  jump to address 3b, line 35
;  increment accumulator to 0
;  jump NOT taken, program HALTed
```

Test Memory Files (also posted online as separate text files)

memory.otherslogical

```
;  test file for Microprogramming Project
;
; test LDA,LDAimm,LDAind,ORA,ORAimm,ORAind,EOR,EORimm,EORind,AND,ANDimm,ANDind,
;      ADD,ADDimm,ADDind,SUBA,SUBAimm,SUBAind,LDS,LDSimm,LDSind,JVS,JVSind

01 00 06         ;01: LDA 0006                                                    (00-02)
1c 00 07        ;02: JMP 0007                                 (03-05)
69              ;03: value to load into accumulator           (06)
03 00 0d        ;04: LDAind 000d                              (07-09)
1c 00 10        ;05: JMP 0010                                 (0a-0c)
00 0f           ;06: address of value to load into accumulator (0d-0e)
aa              ;07: value to load into accumulator           (0f)
05 bc           ;08: LDAimm bc                                (10-11)
11 00 18        ;09: ORA 0018                                 (12-14)
1c 00 19        ;10: JMP 0019                                 (15-17)
ab              ;11: value to OR with accumulator             (18)
13 00 1f        ;12: ORAind 001f                              (19-1b)
1c 00 22        ;13: JMP 0022                                 (1c-1e)
00 21           ;14: address of value to OR with accumulator  (1f-20)
ec              ;15: value to OR with the accumulator         (21)
15 3b           ;16: ORAimm 3b                                (22-23)
21 00 2a        ;17: EOR 002a                                 (24-26)
1c 00 2b        ;18: JMP 002b                                 (27-29)
24              ;19: value to EOR with accumulator            (2a)
23 00 31        ;20: EORind 0031                              (2b-2d)
1c 00 34        ;21: JMP 0034                                 (2e-30)
00 33           ;22: address of value to EOR with accumulator (31-32)
98              ;23: value to EOR with accumulator            (33)
25 fe           ;24: EORimm fe                                (34-35)
31 00 3c        ;25: AND 003c                                 (36-38)
1c 00 3d        ;26: JMP 003d                                 (39-3b)
aa              ;27: value to AND with accumulator            (3c)
33 00 43        ;28: ANDind 0043                              (3d-3f)
1c 00 46        ;29: JMP 0046                                 (40-42)
00 45           ;30: address of value to AND with accumulator (43-44)
73              ;31: value to AND with accumulator            (45)
35 94           ;32: ANDimm 94                                (46-47)
1a              ;33: HALT                                     (48)

;program should execute as follows:
;  load accumulator with 69
;  jump to address 7, line 4
;  load accumulator with aa
;  jump to address 10, line 8
;  load accumulator with bc
;  OR accumulator with ab, ab or bc = bf
;  jump to address 19, line 12
;  OR accumulator with ec, bf or ec = ff
;  jump to address 22, line 16
;  OR accumulator with 3b, ff or 3b = ff
;  EOR accumulator with 24, ff xor 24 = db
;  jump to address 2b, line 20
;  EOR accumulator with 98, db xor 98 = 43
;  jump to address 34, line 24
;  EOR accumulator with fe, 43 xor fe = bd
;  AND accumulator with aa, bd and aa = a8
;  jump to address 3d, line 28
;  AND accumulator with 37, a8 and 37 = 20
;  jump to address 46, line 32
;  AND accumulator with 94, 20 and 94 = 00
;  end program with HALT
```

Test Memory Files (also posted online as separate text files)

memory.othersmath

```
;  test file for Microprogramming Project
;
;  test ADD,ADDimm,ADDind,SUBA,SUBAimm,SUBAind,LDS,LDSimm,LDSind

41 00 06               ;01: ADD 0006                                  (00-02)
1c 00 07               ;02: JMP 0007                                  (03-05)
5a                     ;03: value to add to accumulator               (06)
43 00 0d               ;04: ADDind 000d                               (07-09)
1c 00 10               ;05: JMP 0010                                  (0a-0c)
00 0f                  ;06: address of value to add to accumulator    (0d-0e)
24                     ;07: value to add to accumulator               (0f)
45 33                  ;08: ADDimm 33                                 (10-11)
51 00 18               ;09: SUBA 0018                                 (12-14)
1c 00 19               ;10: JMP 0019                                  (15-17)
13                     ;11: value to subtract from accumulator        (18)
53 00 1f               ;12: SUBAind 001f                              (19-1b)
1c 00 22               ;13: JMP 0022                                  (1c-1e)
00 21                  ;14: address of value to subtract from accum.  (1f-20)
8f                     ;15: value to subtract from accumulator        (21)
55 25                  ;16: SUBAimm 25                                (22-23)
61 00 2a               ;17: LDS 002a                                  (24-26)
1c 00 2c               ;18: JMP 002c                                  (27-29)
aa 69                  ;19: value to load into stack pointer          (2a-2b)
63 00 32               ;20: LDSind 0032                               (2c-2e)
1c 00 36               ;21: JMP 0036                                  (2f-31)
00 34                  ;22: address of value to load into SP          (32-33)
69 aa                  ;23: value to load into stack pointer          (34-35)
65 ab cd               ;24: LDSimm abcd                               (36-38)
1a                     ;25: HALT                                      (39)

;program should execute as follows:
;  add 5a to 0 in accumulator
;  jump to address 7, line 4
;  add 24 to accumulator, 5a + 24 = 7e
;  jump to address 10, line 8
;  add 33 to accumulator, 7e + 33 = b1
;  subtract 13 from accumulator, b1 - 13 = 9e
;  jump to address 19, line 12
;  subtract 8f from accumulator, 9e - 8f = f
;  jump to address 22, line 16
;  subtract 25 from accumulator, f - 25 = -EA
;  load aa into SPHi, 69 into SPLo
;  jump to address 40, line 28
;  load 69 into SPHi, aa into SPLo
;  jump to address 4a, line 32
;  load ab into SPHi, cd into SPLo
;  end program with HALT
```

Test Memory Files (also posted online as separate text files)

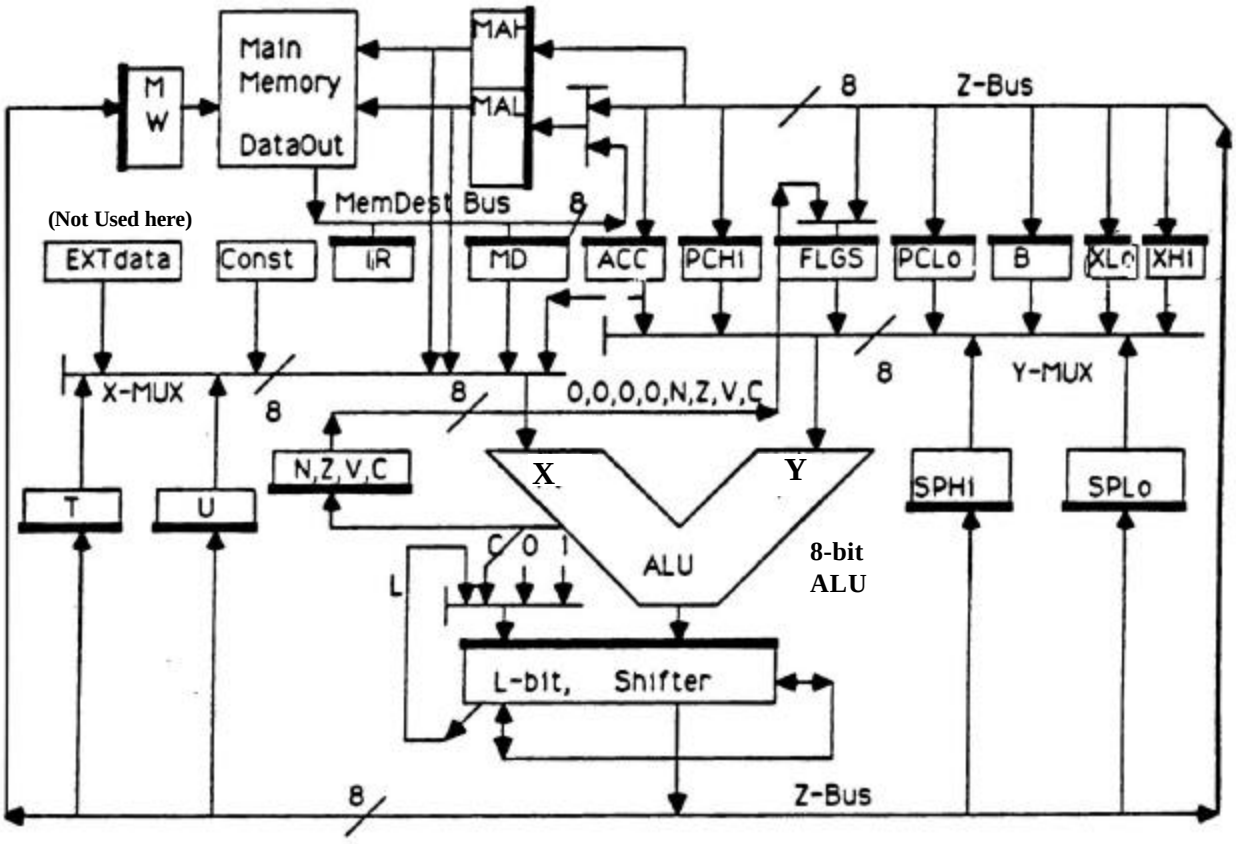
memory.storebranch

```
;  test file for Microprogramming Project
;
; test branch and store instructions
;   -specifically test STA,STAind,JMP,JMPind,JSR,JSRind,and RET

4a            ;01: INCA (00)
4a            ;02: INCA (01)
0c 0f 01      ;03: STA 0f01 (02-04)
4a            ;04: INCA (05)
0e 00 0d      ;05: STAind 000d (06-08)
1c 00 0f      ;06: JMP 000f (09-0b)
4a            ;07: INCA - filler (should not execute) (0c)
0f 05         ;08: address to store at from line 5 (0d-0e)
5a            ;09: DECA - should jump here from line 6 (0f)
1e 00 13      ;10: JMPind 0013 (10-12)
00 16         ;11: address to jump to from line 10 (13-14)
4a            ;12: INCA - filler (should not execute) (15)
5a            ;13: DECA - should jump here from line 10 (16)
6c 00 1e      ;14: JSR 001e (17-19)
6e 00 20      ;15: JSRind 0020 - should return here from RET on line 18 (1a-1c)
1a            ;16: HALT - end of program - returns from line 22 (1d)
3a            ;17: CMA - should jump here from line 14 (1e)
6a            ;18: RET - should return to line 15 (1f)
00 23         ;19: address to jump to from line 15 (20-21)
4a            ;20: INCA - FILLER (should not execute) (22)
2a            ;21: CLA - should jump here from line 15 (23)
6a            ;22: RET - should return to line 16 (24)


;execution should be as follows -
;  increment accumulator twice to 2
;  store the accumulator at the address 0f01
;  increment accumulator to 3
;  store accumulator at address 0f05
;  jump to address 000F, which is line 9 above
;  decrement accumulator to 2
;  jump to address at address 0013 which is 0016, or line 13
;  decrement accumulator to 1
;  jump to subroutine at address 001e, line 17
;  do one's complement of accumulator
;  return from subroutine to address 1a, line 15
;  jump to subroutine at the address that is at address 0020,
;      which is 0023, line 21
;  clear the accumulator
;  return from subroutine to address 1d, line 16
;  end program with HALT
```

The Datapath



- **ISA Registers:**
  - **Program Counter (PC) 16 bits implemented in datapath by two 8-bit registers PCHi, PCLo**
  - **Accumulator (ACC) 8-bit register.**
  - **Flags Register (FLGS) 8-bit register**
  - **Stack Pointer (SP) 16-bit register implemented in datapath by two 8-bit registers SPHi, SPLo**
- **Memory Details**
  - **A single main memory used for instructions and data. 16-bit address.**
  - **Memory Address High Register (MAH) must be loaded first for both reads and writes.**
  - **For memory writes the MW (Memory Write) register must be loaded with the byte to be stored.**
  - **Reading from and writing to memory is triggered by loading MAL (Memory Address Low) register using ALUDEST microinstruction field options.**
  - **Memory read/write operations take two clock cycles from the start of a memory read or write.**
- **Temporary Microprogram Registers:**
  - **The following 8 -bit registers can be used by the control microprogram to store temporary values as needed and are not visible to the ISA or user programs: T, U, B, XLo, Xhi**
- **The ALU Output Shifter:**
  - **Combinational logic shifter that can shift the ALU output one position left or right and also manipulate the most significant bit of the ALU output (L-bit) before shifting in a single cycle.**
- **Constant Value (Const):** **Shown as a possible input to the ALU is 5-bit value that can be specified by a microinstruction field.**

The Control Microinstruction Format

A	B	C	D	E	F	G	H	I	J	K	L	M	N
MEMDEST	LCNTRL	SHFTCTRL	ALUCTRL	YSOURCE	XSOURCE	ALUDEST	CONST	LOADFLGS	TEST	INTERNAB	ADDRF	COND	OPCODE
Bits 0-1 (2 bits)	Bits 2-3 (2 bits)	Bits 4-5 (2 bits)	Bits 6-9 (4 bits)	Bits 10-13 (4 bits)	Bits 14-16 (3 bits)	Bits 17-20 (4 bits)	Bits 21-25 (5 bits)	Bit 26 (1 bit)	Bits 27-28 (2 bits)	Bit 29 (1 bit)	Bits 30-38 (9 bits)	Bits 39-40 (2 bits)	Bits 41-42 (2 bits)

Bit	Field	Name	Operations
0 1	A	MEMDEST	00 - NOP (See Note 2) 01 - MD 10 - MD and MALow 11 - MD and MALow and IR
2 3	B	LCNTRL	00 - Leave L alone 01 - Clear L 10 - Set L 11 - L = Carry Out of ALU
4 5	C	SHFTNTRL	00 - No Shift 01 - Shift Right 10 - Shift Left 11 - Not Used
6 7 8 9	D	ALUCTRL	0000 - X 0001 - Y 0010 - X plus Y 0011 - X plus Y plus 1 0100 - X and Y 0101 - X or Y 0110 - X xor Y 0111 - not Y 1000 - X plus 1 1001 - Y plus 1 1010 - X and 1 1011 - Y and 1 1100 - Y plus not X plus 1 1101 - not X 1110 - minus 1 1111 - 0
10 11 12 13	E	YSOURCE	0000 - none 0001 - ACC 0010 - PCLo 0011 - SPLo 0100 - B 0101 - FLAGS 0110 - XHi 0111 - XLo 1000 - PCHi 1001 - SPHi 1010 - unused 1011 - unused 1100 - unused 1101 - unused 1110 - unused 1111 - unused
14 15 16	F	XSOURCE	000 - ACC 001 - MD 010 - CONST (Constant Field from Microinstruction) 011 - External Data (not used here) 100 - T 101 - MALo 110 - MAHi 111 - U
17 18 19 20	G	ALUDEST	0000 - none 0001 - ACC 0010 - PCLo 0011 - SPLo 0100 - B 0101 - FLAGS 0110 - XHi 0111 - XLo 1000 - PCHi 1001 - SPHi 1010 - MAHi 1011 - MALo, Read (Starts Memory Read) 1100 - T 1101 - MALo, Write (Starts Memory Write) 1110 - U 1111 - MW
21 22 23 24 25	H	CONST	Unsigned 5-bit constant for XSOURCE
26	I	LOADFLGS	When 1 loads FLAGS from internal ALU flags NBIT, ZBIT, VBIT, CBIT

The Control Microinstruction Format (Continued)

A	B	C	D	E	F	G	H	I	J	K	L	M	N
MEMDEST	LCNTRL	SHFTCTRL	ALUCTRL	YSOURCE	XSOURCE	ALUDEST	CONST	LOADFLGS	TEST	INTERNAB	ADDRF	COND	OPCODE
Bits 0-1 (2 bits)	Bits 2-3 (2 bits)	Bits 4-5 (2 bits)	Bits 6-9 (4 bits)	Bits 10-13 (4 bits)	Bits 14-16 (3 bits)	Bits 17-20 (4 bits)	Bits 21-25 (5 bits)	Bit 26 (1 bit)	Bits 27-28 (2 bits)	Bit 29 (1 bit)	Bits 30-38 (9 bits)	Bits 39-40 (2 bits)	Bits 41-42 (2 bits)

Bit	Field	Name	Operations
27 28	J	TEST	00 - Branch on NBIT (See Note 1) 01 - Branch on ZBIT 10 - Branch on VBIT 11 - Branch on CBIT
29	K	INTRENAB	Not Used, always put 0 in this field
30 31 32 33 34 35 36 36 38	L	ADDRF	9-bit address field of Next microinstruction bit 30: Most Significant Bit of Address (See Note 1)
39 40	M	COND	Determines Type of Next Microinstruction Address (See note 1)
41 42	N	OPCODE	Opcode: Format of Microinstruction Only one format used here, always put 0 in this field.

Micrinstruction Fields Notes:

Note 1:

If COND = 00 then MPC (next microinstruction address) = ADDRF (I.e bits 30-38) as given

If COND = 01 Then MPC (next microinstruction address) is determined by bits 30-37 of ADDRF along with the particular test bit specified by TEST field from the ALU replacing the least significant bit of 38 of ADDRF (i.e two way branch on the condition bit tested).

If COND = 10 Then MPC (next microinstruction address) by bits 30-34 (five most significant bits of ADDRF) along with the 4 most significant bits of IR (instruction register) replacing the low 4 bits 35-38 of ADDRF (I.e 16-way branch on the 4 most significant bits of IR).

If COND = 11 Then MPC (next microinstruction address) by bits 30-34 (five most significant bits of ADDRF) along with the 4 least significant bits of IR (instruction register) replacing the low 4 bits 35-38 of ADDRF (I.e 16-way branch on the 4 least significant bits of IR).

Note 2: The Memory destination from the memory bus MBUS (memory data bus) is as follows:

When the MEMDEST field is not 00, MD is loaded from MBUS.

When the field is 10 (2 decimal) MD and MALo registers are loaded from MBUS

When the field is 11 (3 decimal) MD, MALo and IR registers are loaded from MBUS

Note 3: For every microinstruction field, use the decimal value of the binary field values specified above, as shown in the sample microprogram start segment on the next page.

Sample Microprogram Start Segment

```
;A - MEMDEST
;|   B - LCNTRL
;|   |   C - SHFTCNTRL
;|   |   |   D - ALUCNTRL
;|   |   |   |   E - YSRCE
;|   |   |   |   |   F - XSRCE
;|   |   |   |   |   |   G - ALUDEST
;|   |   |   |   |   |   |   H - CONST
;|   |   |   |   |   |   |   |   I - LDFLG
;|   |   |   |   |   |   |   |   |   J - TEST
;|   |   |   |   |   |   |   |   |   |   K - INTRNE
;|   |   |   |   |   |   |   |   |   |   |   L - ADDR
;|   |   |   |   |   |   |   |   |   |   |   |   M - COND
;|   |   |   |   |   |   |   |   |   |   |   |   |   N - OPCODE
;|   |   |   |   |   |   |   |   |   |   |   |   |   |  ADDRESS   COMMENT
 0   0   0   15  0   7   8   0   0   0   0   1   0   0; 000 :PCHI <- 0
 0   0   0   15  0   7   2   0   0   0   0   2   0   0; 001 :PCLO <- 0
 0   0   0   13  0   2   9   0   0   0   0   3   0   0; 002 :SPHI <- FF
 0   0   0   13  0   2   3   0   0   0   0   4   0   0; 003 :SPLO <- FF
 0   0   0   15  0   7   1   0   4   0   0   5   0   0; 004 :ACC <- 0
 0   0   0   1   8   7   10  0   0   0   0   6   0   0; 005 :MAHI <- PCHI
 0   0   0   1   2   7   11  0   0   0   0   7   0   0; 006 :MALO <- PCLO
 0   0   0   9   2   7   2   0   0   3   0   8   1   0; 007 :PCLO += 1
 3   0   0   0   0   7   0   0   0   0   0   10  0   0; 008 :IR <- M
 3   0   0   9   8   7   8   0   0   0   0   10  0   0; 009 :IR <- M, PCHI+=1
;END FETCH, START DECODE
 0   0   0   0   0   7   0   0   0   0   0   16  3   0; 010 :START DECODE
 0   0   0   0   0   7   0   0   0   0   0   16  0   0; 011 :NO-OP
 0   0   0   0   0   7   0   0   0   0   0   16  0   0; 012 :NO-OP
 0   0   0   0   0   7   0   0   0   0   0   16  0   0; 013 :NO-OP
 0   0   0   0   0   7   0   0   0   0   0   16  0   0; 014 :NO-OP
 0   0   0   0   0   7   0   0   0   0   0   16  0   0; 015 :NO-OP
;BEGIN FIRST STAGE DECODE
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 016 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   32  0   0; 017 :NO-OP DIRECT S/B
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 018 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   64  0   0; 019 :NO-OP DIRECT OTHER
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 020 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   96  0   0; 021 :NO-OP INDIRECT S/B
 0   0   0   0   0   7   0   0   0   0   0   319 0   0; 022 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   144 0   0; 023 :NO-OP INDIRECT OTHER
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 024 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 025 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   192 2   0; 026 :NO-OP INHERENT
 0   0   0   0   0   7   0   0   0   0   0   208 0   0; 027 :NO-OP IMMEDIATE OTHER
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 028 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 029 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 030 :NO-OP ERROR
 0   0   0   0   0   7   0   0   0   0   0   349 0   0; 031 :NO-OP ERROR
;END FIRST STAGE DECODE, BEGIN SECOND STAGE
```