EECC 550 - Winter 2012
Microprogramming Project
Due 12 Noon Monday February 18

You are to write and submit a microprogram to interpret the following 8-bit accumulator-based target machine instruction set (ISA) for a multicycle CPU design with a given datapath and microinstruction format using the MicroTiger microprogramming tool.

MicroTiger is a graphical microcode simulator with a reconfigurable datapath that runs on any Windows PC. This tool was developed by Brian VanBuren for his CE MS Thesis at RIT entitled "Graphical Microcode Simulator with a Reconfigurable Datapath."

From myCourses (or course web page) download the project package "project.zip". The package contains an executable that runs under Windows, along with support files, start microprogram and ISA test programs. project.zip contains the following files:

- MicroTiger executable: microtiger-student.exe
- Required program support DLLs: mingwm10.dll wxbase26_gcc_custom.dll wxmsw26_core_gcc_custom.dll wxmsw26_html_gcc_custom.dll
- Target datapath file: datapath.dp . Start microprogram file: start-microprogram.mc
- Five ISA test programs (will be used to evaluate the correctness of your microprogram):
  memory_inherent.isa memory_otherbranches.isa memory_otherslogical.isa
  memory_othersmath.isa memory_storebranch.isa

You do not have to install anything to run MicroTiger, just unzip all files to a single directory.

- How To Use MicroTiger: Once running MicroTiger, a full user guide is available through the Help->Contents menu.
- Modify/complete the given start microprogram to implement the project ISA. The given start microprogram initializes the datapath, fetches the instruction opcode byte and starts the decode process (each instruction, however, is just treated as a No-Op).
- For this project, submit your completed and fully-commented microprogram file (.mc) to the myCourses dropbox for the project.
- In addition to submitting your microprogram to the dropbox, you should submit **a written report** with the following items (electronic version to dropbox and printed copy):

(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 microprogram as submitted in dropbox.
(6) A description of testing procedures used and observations.
(7) Any additional remarks or conclusions you deem noteworthy of mention.

Target Machine & Instruction Set Architecture

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 in this ISA:

(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 or flags 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**
00000001 - **NOP** - No operation		- - - -
00010001 - **HALT** - Halt the machine		- - - -
00100001 - **CLA** - Clear the accumulator		0100
00110001 - **CMA** - 1's complement the accumulator		xx00
01000001 - **INCA** - Increment the accumulator by one		xx0x
01010001 - **DECA** - Decrement the accumulator by one		xx0x
01100001 - **RET** - Return from subroutine (post incrementing SP)		- - - -
01110001 - **ROLCA** - Circular shift Carry bit & Acc left 1 bit		xx0x
10000001 - **CLC** - Clear Carry Flag bit		- - - 0
10010001 - **STC** - Set Carry Flag bit		- - - 1

Store and Branch Instructions: Two additional address bytes

Opcode	Condition Code:	**NZVC**
00001z10 - **STA** - Store accumulator (use STAin for indirect)		- - - -
00011z10 - **JMP** - Unconditional branch		- - - -
00101z10 - **JEQ** - Branch if equal to zero (Z=1)		- - - -
00111z10 - **JCS** - Branch if carry (C=1)		- - - -
01001z10 - **JLT** - branch if negative (N=1)		- - - -
01011z10 - **JVS** - branch if overflow (V=1)		- - - -
01101z10 - **JSR** - jump to subroutine (push PC on stack - predecrement SP)		- - - -

Other Instructions: One or two additional bytes

Opcode	Condition Codes: **NZVC**
0000yz11 - LDA - Load Acc (use LDAim for immediate)	xx00
0001yz11 - ORA - Inclusive OR to Accumulator	xx00
0010yz11 - EOR - Exclusive OR to Accumulator	xx00
0011yz11 - AND - Logical AND to Accumulator	xx00
0100yz11 - ADD - Add to Accumulator	xxxx
0101yz11 - SUBA - Subtract from Accumulator	xxxx
0110yz11 - 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.

Instruction Opcodes

Inherent Instructions Opcodes

	Binary	Hex
NOP	00000001	01
HALT	00010001	11
CLA	00100001	21
CMA	00110001	31
INCA	01000001	41
DECA	01010001	51
RET	01100001	61
ROLCA	01110001	71
CLC	10000001	81
STC	10010001	91

Store and Branch Instructions Opcodes

	Binary	Hex	
		(Direct z=0)	(Indirect z=1)
STA	00001z10	0a	0e
JMP	00011z10	1a	1e
JEQ	00101z10	2a	2e
JCS	00111z10	3a	3e
JLT	01001z10	4a	4e
JVS	01011z10	5a	5e
JSR	01101z10	6a	6e

Other Instructions Opcodes

	Binary	Hex		
		Direct y = z =0	Indirect y=0 z =1	Immediate y=1 z=0
LDA	0000yz11	03	07	0b
ORA	0001yz11	13	17	1b
EOR	0010yz11	23	27	2b
AND	0011yz11	33	37	3b
ADD	0100yz11	43	47	4b
SUBA	0101yz11	53	57	5b
LDS	0110yz11	63	67	6b

Test Programs: (in project zip file)

memory_inherent.isa

```
;  test file for Microprogramming Project
;
01        ;NOP
41        ;INCA
41        ;INCA
41        ;INCA
51        ;DECA
21        ;CLA
51        ;DECA from 00
41        ;INCA
31        ;CMA
21        ;CLA
41        ;INCA
91        ;STC
71        ;ROLCA
91        ;STC
81        ;CLC
11        ;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 Programs: (in project zip file)

memory_otherbranches.isa

```
;  test file for Microprogramming Project
; test branch and store instructions
;   -specifically test JEQ,JEQind,JCS,JCSind,JLT,JLTind
41              ;01: INCA                                       (00)
21              ;02: CLA                                        (01)
2a 00 06        ;03: JEQ 0006                                   (02-04)
41              ;04: INCA - filler (should not execute)         (05)
51              ;05: DECA - should jump here from line 03       (06)
2a 00 0c        ;06: JEQ 000c                                   (07-09)
41              ;07: INCA - should be executed, no jump         (0a)
2e 00 0f        ;08: JEQind 000f                                (0b-0d)
41              ;09: INCA - filler (should not be executed)     (0e)
00 12           ;10: address to jump to from line 08            (0f-10)
41              ;11: INCA - filler (should not be executed)     (11)
51              ;12: DECA - should jump here from line 08       (12)
2e 00 0f        ;13: JEQIND 000f                                (13-15)
41              ;14: INCA - should be executed, no jump         (16)
91              ;15: STC                                        (17)
3a 00 1c        ;16: JCS 001c                                   (18-1a)
41              ;17: INCA - filler (should not be executed)     (1b)
81              ;18: CLC - should jump here from line 16        (1c)
3a 00 1c        ;19: JCS 001c                                   (1d-1f)
91              ;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)
41              ;23: INCA - should not be executed              (26)
81              ;24: CLC                                        (27)
3e 00 24        ;25: JCSind 0024                                (28-2a)
51              ;26: DECA - should be executed, no jump         (2b)
4a 00 30        ;27: JLT 0030                                   (2c-2e)
51              ;28: DECA - filler                              (2f)
41              ;29: INCA - should jump here from line 27       (30)
4a 00 30        ;30: JLT 0030                                   (31-33)
51              ;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)
51              ;34: DECA - filler                              (3a)
41              ;35: INCA - should jump here from line 32       (3b)
4e 00 38        ;36: JLTind 0038                                (3c-3e)
11              ;37: HALT, no jump                              (3f)

;  program should execute as follows:
;  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 Programs: (in project zip file)

memory_otherslogical.isa

```
;  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

03 00 06         ;01: LDA 0006                                  (00-02)
1a 00 07        ;02: JMP 0007                                   (03-05)
69              ;03: value to load into accumulator             (06)
07 00 0d        ;04: LDAind 000d                                (07-09)
1a 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)
0b bc           ;08: LDAimm bc                                  (10-11)
13 00 18        ;09: ORA 0018                                   (12-14)
1a 00 19        ;10: JMP 0019                                   (15-17)
ab              ;11: value to OR with accumulator               (18)
17 00 1f        ;12: ORAind 001f                                (19-1b)
1a 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)
1b 3b           ;16: ORAimm 3b                                  (22-23)
23 00 2a        ;17: EOR 002a                                   (24-26)
1a 00 2b        ;18: JMP 002b                                   (27-29)
24              ;19: value to EOR with accumulator              (2a)
27 00 31        ;20: EORind 0031                                (2b-2d)
1a 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)
2b fe           ;24: EORimm fe                                  (34-35)
33 00 3c        ;25: AND 003c                                   (36-38)
1a 00 3d        ;26: JMP 003d                                   (39-3b)
aa              ;27: value to AND with accumulator              (3c)
37 00 43        ;28: ANDind 0043                                (3d-3f)
1a 00 46        ;29: JMP 0046                                   (40-42)
00 45           ;30: address of value to AND with accumulator   (43-44)
37              ;31: value to AND with accumulator              (45)
3b 94           ;32: ANDimm 94                                  (46-47)
11              ;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 Programs: (in project zip file)

memory_othersmath.isa

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

43 00 06               ;01: ADD 0006                                  (00-02)
1a 00 07               ;02: JMP 0007                                  (03-05)
5a                     ;03: value to add to accumulator               (06)
47 00 0d               ;04: ADDind 000d                               (07-09)
1a 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)
4b 33                  ;08: ADDimm 33                                 (10-11)
53 00 18               ;09: SUBA 0018                                 (12-14)
1a 00 19               ;10: JMP 0019                                  (15-17)
13                     ;11: value to subtract from accumulator        (18)
57 00 1f               ;12: SUBAind 001f                              (19-1b)
1a 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)
5b 25                  ;16: SUBAimm 25                                (22-23)
63 00 2a               ;17: LDS 002a                                  (24-26)
1a 00 2c               ;18: JMP 002c                                  (27-29)
aa 69                  ;19: value to load into stack pointer          (2a-2b)
67 00 32               ;20: LDSind 0032                               (2c-2e)
1a 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)
6b ab cd               ;24: LDSimm abcd                               (36-38)
11                     ;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 Programs: (in project zip file)

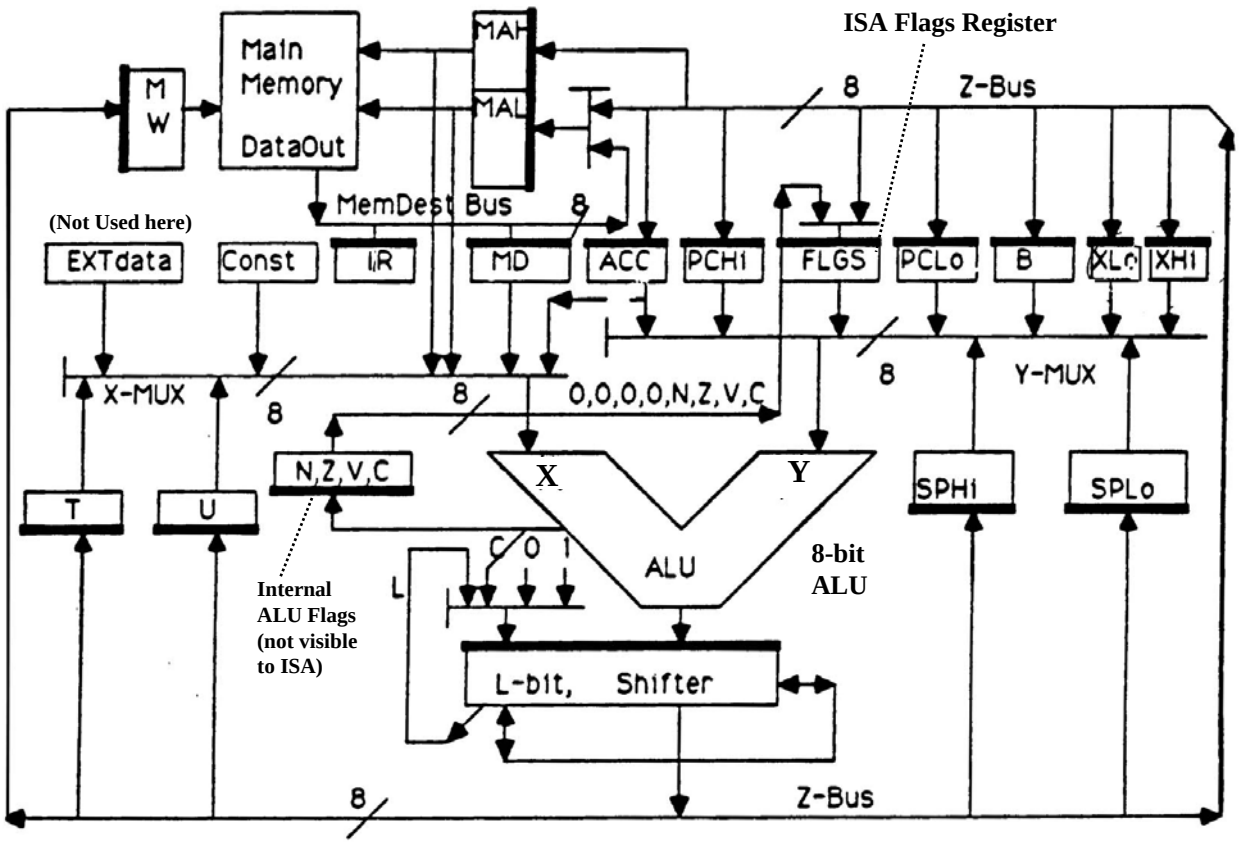
memory_storebranch.isa

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

41           ;01: INCA (00)
41           ;02: INCA (01)
0a 0f 01     ;03: STA 0f01 (02-04)
41           ;04: INCA (05)
0e 00 0d     ;05: STAind 000d (06-08)
1a 00 0f     ;06: JMP 000f (09-0b)
41           ;07: INCA - filler (should not execute) (0c)
0f 05        ;08: address to store at from line 5 (0d-0e)
51           ;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)
41           ;12: INCA - filler (should not execute) (15)
51           ;13: DECA - should jump here from line 10 (16)
6a 00 1e     ;14: JSR 001e (17-19)
6e 00 20     ;15: JSRind 0020 - should return here from RET on line 18 (1a-1c)
11           ;16: HALT - end of program - returns from line 22 (1d)
31           ;17: CMA - should jump here from line 14 (1e)
61           ;18: RET - should return to line 15 (1f)
00 23        ;19: address to jump to from line 15 (20-21)
41           ;20: INCA - FILLER (should not execute) (22)
21           ;21: CLA - should jump here from line 15 (23)
61           ;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, low four bits are the flags: NZVC**
  - **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 <u>two clock cycles</u> from the start of a memory read or write.**
- **Temporary Datapath/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 (field H).**

The Control Microinstruction Format

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

Bit	Field	Name	Operations
0 1	A	MEMDEST	00 (0) - NOP (See Note 2) 01 (1) - MD 10 (2) - MD and MALow 11 (3) - MD and MALow and IR
2 3	B	LCNTRL	00 (0) - Leave L alone 01 (1) - Clear L 10 (2) - Set L 11 (3) - L = Carry Out of ALU
4 5	C	SHFTNTRL	00 (0) - No Shift 01 (1) - Shift Right 10 (2) - Shift Left 11 (3) - Not Used
6 7 8 9	D	ALUCTRL	0000 (0) - X 0001 (1) - Y 0010 (2) - X plus Y 0011 (3) - X plus Y plus 1 0100 (4) - X and Y 0101 (5) - X or Y 0110 (6) - X xor Y 0111 (7) - not Y 1000 (8) - X plus 1 1001 (9) - Y plus 1 1010 (10) - X and 1 1011 (11) - Y and 1 1100 (12) - Y plus not X plus 1 1101 (13) - not X 1110 (14)- minus 1 (i.e. the hex value FF) 1111 (15) - 0
10 11 12 13	E	YSOURCE	0000 (0) - none 0001 (1) - ACC 0010 (2) - PCLo 0011 (3) - SPLo 0100 (4) - B 0101 (5) - FLAGS 0110 (6) - XHi 0111 (7) - XLo 1000 (8) - PCHi 1001 (9) - SPHi 1010 (10) - unused 1011 (11) - unused 1100 (12) - unused 1101 (13) - unused 1110 (14) - unused 1111 (15) - unused
14 15 16	F	XSOURCE	000 (0) - ACC 001 (1) - MD 010 (2) - CONST (Constant Field from Microinstruction) 011 (3) - External Data (not used here) 100 (4) - T 101 (5) - MALo 110 (6) - MAHi 111 (7) - U
17 18 19 20	G	ALUDEST	0000 (0) - none 0001 (1) - ACC 0010 (2) - PCLo 0011 (3) - SPLo 0100 (4) - B 0101 (5) - FLAGS 0110 (6) - XHi 0111 (7) - XLo 1000 (8) - PCHi 1001 (9) - SPHi 1010 (10) - MAHi 1011 (11) - MALo, Read (Starts Memory Read) 1100 (12) - T 1101 (13) - MALo, Write (Starts Memory Write) 1110 (14) - U 1111 (15) - 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 Bits 0-1 (2 bits)	LCNTRL Bits 2-3 (2 bits)	SHFTCTRL Bits 4-5 (2 bits)	ALUCTRL Bits 6-9 (4 bits)	YSOURCE Bits 10-13 (4 bits)	XSOURCE Bits 14-16 (3 bits)	ALUDEST Bits 17-20 (4 bits)	CONST Bits 21-25 (5 bits)	LOADFLGS Bit 26 (1 bit)	TEST Bits 27-28 (2 bits)	INTERNAB Bit 29 (1 bit)	ADDRF Bits 30-38 (9 bits)	COND Bits 39-40 (2 bits)	OPCODE Bits 41-42 (2 bits)

Bit	Field	Name	Operations
27 28	J	TEST	00 (0) - Branch on NBIT (See Note 1) 01 (1) - Branch on ZBIT 10 (2) - Branch on VBIT 11 (3) - 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

From project zip file: start-microprogram.mc

```
# Sample microprogram start segment
#
#  A - MemDest
# | B - LCtrl
# | | C - ShftCtrl
# | | | D - AluCtrl
# | | | | E - YSrc
# | | | | | F - XSrc
# | | | | | | G - AluDest
# | | | | | | | H - Const
# | | | | | | | | I - LdFlg
# | | | | | | | | | J - Test
# | | | | | | | | | | K - Intrne
# | | | | | | | | | | | L - Addr
# | | | | | | | | | | | | M - Cond
# | | | | | | | | | | | | | N - OpCode
# | | | | | | | | | | | | | |; addr: comment

# Initialize
# 000: PcHi ← 0
memdest=0,lctrl=0,shftctrl=0,aluctrl=15,ysource=0,xsource=0,aludest=8,const=0,loadflgs=0,test=0,intrenab=0,addrf=1,cond=0,opcode=0;
# 001: PcLo ← 0
memdest=0,lctrl=0,shftctrl=0,aluctrl=15,ysource=0,xsource=0,aludest=2,const=0,loadflgs=0,test=0,intrenab=0,addrf=2,cond=0,opcode=0;
# 002: SpHi ← 255
memdest=0,lctrl=0,shftctrl=0,aluctrl=14,ysource=0,xsource=0,aludest=9,const=0,loadflgs=0,test=0,intrenab=0,addrf=3,cond=0,opcode=0;
# 003: SpLo ← 255
memdest=0,lctrl=0,shftctrl=0,aluctrl=14,ysource=0,xsource=0,aludest=3,const=0,loadflgs=0,test=0,intrenab=0,addrf=4,cond=0,opcode=0;
# 004: ACC ← 0
memdest=0,lctrl=0,shftctrl=0,aluctrl=15,ysource=0,xsource=0,aludest=1,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;


# Fetch
# 005: MaHi ← PcHi  - isa  label used to calculate CPI
isa:
memdest=0,lctrl=0,shftctrl=0,aluctrl=1,ysource=8,xsource=0,aludest=10,const=0,loadflgs=0,test=0,intrenab=0,addrf=6,cond=0,opcode=0;
# 006: MaLo ← PcLo, MemRead (start fetch)
memdest=0,lctrl=0,shftctrl=0,aluctrl=1,ysource=2,xsource=0,aludest=11,const=0,loadflgs=0,test=0,intrenab=0,addrf=7,cond=0,opcode=0;
# 007: PcLo += 1  (also wait cycle)
memdest=0,lctrl=0,shftctrl=0,aluctrl=9,ysource=2,xsource=0,aludest=2,const=0,loadflgs=0,test=3,intrenab=0,addrf=8,cond=1,opcode=0;
# 008: C is 0,  IR ← Mem(PC)
memdest=3,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=10,cond=0,opcode=0;
# 009: C is 1, IR ← Mem(PC),  PcHi += 1
memdest=3,lctrl=0,shftctrl=0,aluctrl=9,ysource=8,xsource=0,aludest=8,const=0,loadflgs=0,test=0,intrenab=0,addrf=10,cond=0,opcode=0;

#  END OPCODE FETCH, START DECODE
```

Sample Microprogram Start Segment (continued)

```
# 010:  Start decode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=3,opcode=0;
# 011: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 012: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 013: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 014: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;
# 015: No-OP Filler
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=16,cond=0,opcode=0;

#;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


# Begin first stage decode - 16-way branch on lower half of opcode byte
# 016: 0 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 017: 1 Inherent
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 018: 2 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 019: 3 Other: Direct Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 020: 4 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 021: 5 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 029: 6 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 023: 7 Other: Indirect Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 024: 8 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 025: 9 NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 026: A Store and Branch: Direct Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 027: B Other: Immediate Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 028: C NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 029: D NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=2,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 030: E Store and Branch: Indirect Addressing
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;
# 031: F NoSuchOpCode
memdest=0,lctrl=0,shftctrl=0,aluctrl=0,ysource=0,xsource=0,aludest=0,const=0,loadflgs=0,test=0,intrenab=0,addrf=5,cond=0,opcode=0;

#;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```