Spring 2009

Name: Mark F.

Computer Organization Test 2

Question 1. (10 points) Select the best answer to the following true-or-false questions:	**Circle the correct answer**
a. A jump instruction changes the flow of execution by changing the AC.	True [✓ False]
b. Registers are storage locations within the CPU itself.	[✓ True] False
c. A two-pass assembler generally creates a symbol table during the first pass and finishes the complete translation from assembly language to machine language on the second pass.	[✓ True] False
d. The AC, MAR, MBR, PC, and IR registers in MARIE can be used to hold arbitrary data.	True [✓ False]
e. One assembly language instruction generally translates to one machine language instruction.	[✓ True] False
f. One high-level language (e.g., Ada, C++, Java, etc.) instruction generally translates to one machine language instruction.	True [✓ False]

Question 2. (20 points) Translate the following high-level language code segment to MARIE assembly language. Use the variable labels indicated in the code.

```
INPUT X
WHILE X < 0 DO
   SUM = SUM + X
   INPUT X
END WHILE
```

```
           INPUT
           STORE X
WHILE,     SKIPCOND 000        / If AC<0, skip
           JUMP DO
           JUMP END_WHILE
DO,        ADD SUM
           STORE SUM
           INPUT
           JUMP WHILE
END_WHILE, HALT
X,         DEC 0
SUM,       DEC 0
```

Question 3.

a) (5 points) For the below MARIE program, what would the symbol table be?

LABEL	ADDR.	LABEL	ADDR
ELSE	105_{16}	Y	10A
END_IF	108		
IF	101		
ONE	10B		
X	109		

b) (10 points) Translate the given MARIE assembly language into machine language.

Address	**Label**	**Assembly Language**	**Machine Language** (in hex)
100_{16}		LOAD X	1109
101_{16}	IF,	SKIPCOND 800	8800
102_{16}		JUMP ELSE	9105
103_{16}		STORE Y	210A
104_{16}		JUMP END_IF	9108
105_{16}	ELSE,	ADD Y	310A
106_{16}		SUBT ONE	410B
107_{16}		STORE Y	210A
108_{16}	END_IF,	HALT	7000
109_{16}	X,	DEC 10	000A
$10A_{16}$	Y,	DEC 0	0000
$10B_{16}$	ONE,	DEC 1	0001

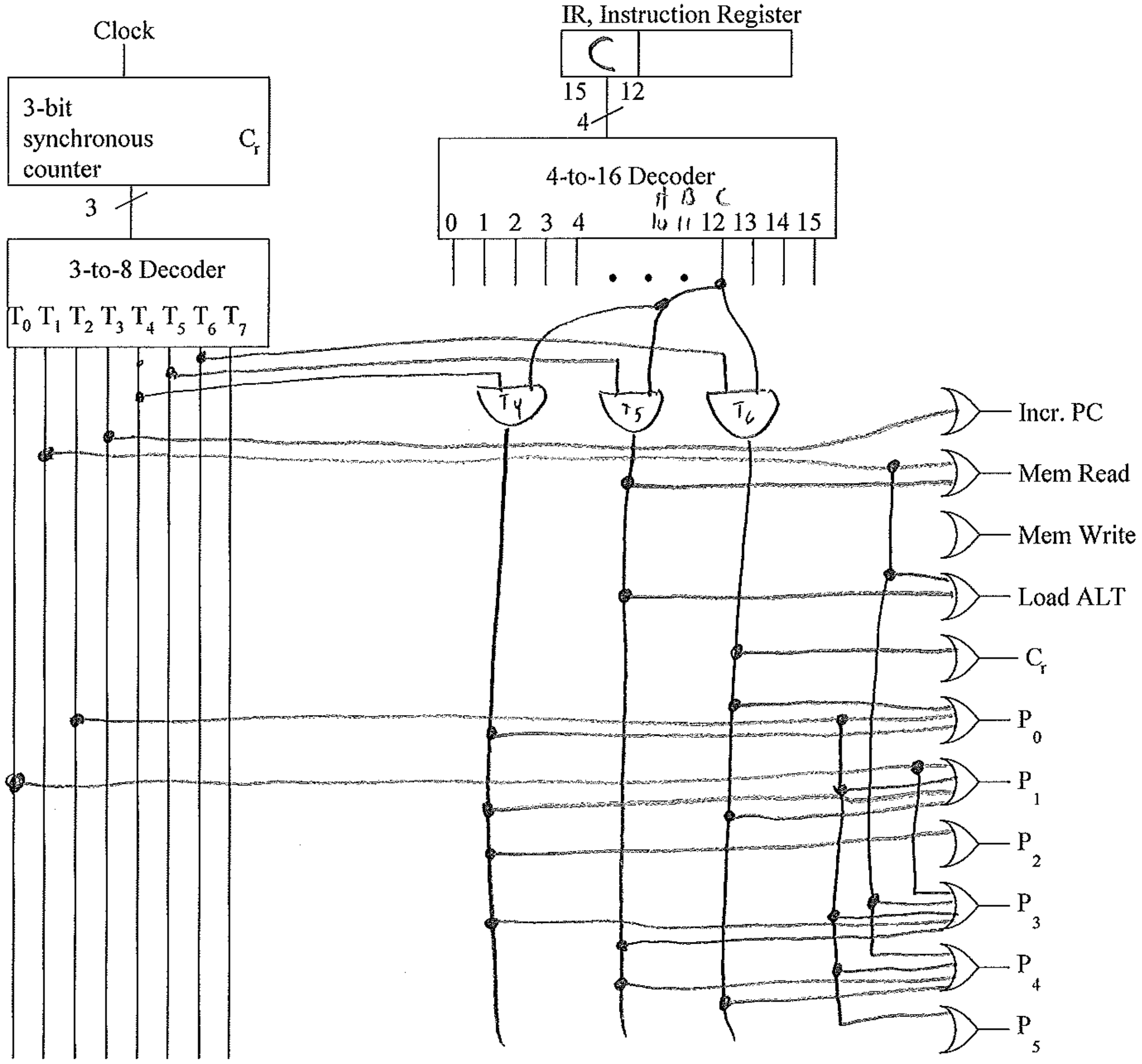
c) (10 points) Translate the above MARIE assembly language into high-level language "pseudo" code.

```
If   X > 0 then
        Y = X
else
        Y = X + Y - 1
end_if
```

Question 4. (10 points) Which control signals should contain a "1" for each steps in the JUMPI instruction?

Step	RTN	Step #	P_5	P_4	P_3	P_2	P_1	P_0	C_r	Incr PC	Mem Read	Mem Write	Load ALT
Fetch	$\overset{1}{MAR} \leftarrow \overset{2}{PC}$	T_0	0	0	1	0	1	0	0	0	0	0	0
	$MBR \leftarrow M[MAR]$	T_1	0	1	1	–	–	–	0	0	1	0	1
	$IR^7 \leftarrow MBR^3$	T_2	1	1	1	0	1	1	0	0	0	0	0
Decode IR[15-12]	$PC \leftarrow PC + 1$	T_3	0	0	0	–	–	–	0	1	0	0	0
Get operand	$MAR^1 \leftarrow IR[11\text{-}0]^7$	T_4	0	0	1	1	1	1	0	0	0	0	0
Execute	$MBR \leftarrow M[MAR]$	T_5	0	1	1	–	–	–	0	0	1	0	1
	$PC^2 \leftarrow MBR^3$	T_6	0	1	0	0	1	1	1	0	0	0	0
		T_7											

Question 5. (10 points) Draw the partial combinational logic of the hardwired control unit to handle the JUMPI (opcode C_{16}) instruction.



Question 6. Recall that the microprogrammed version of MARIE executes a fixed micropogam to perform the fetch-decode-execute cycle. The instruction format for the microinstructions could look like:

MicroOp1	MicroOp2	Jump	Dest
17 13	12 8	7	6 0

`MicroOp1` encodes the type of register transfer notation (RTN) to perform (see Table 4.8 below)

`MicroOp2` contains the binary codes for each instruction to allow comparison to the IR opcode (IR[15-12]).

`Jump` is a single bit indicating that the value in the **Dest** field is a valid micro-address and should be placed in the microsequencer; if `Jump` is "FALSE" (0), then increment to the next microinstruction.

Table 4.8. Microoperation Codes and Corresponding MARIE RTN (p. 221)

NOTE TO CURRENT STUDENT'S: This table is used in edition 2 of the textbook. We are using edition 3 so our table is different.

MicroOp Code	Microoperation	MicroOp Code	Microoperation
00000	NOP	01100	MBR ← M[MAR]
00001	AC ← 0	01101	OutREG ← AC
00010	AC ← AC − MBR	01110	PC ← IR[11-0]
00011	AC ← AC + MBR	01111	PC ← MBR
00100	AC ← InREG	10000	PC ← PC + 1
00101	IR ← M[MAR]	10001	If AC = 00
00110	M[MAR] ← MBR	10010	If AC > 0
00111	MAR ← IR[11-0]	10011	If AC < 0
01000	MAR ← MBR	10100	If IR[11-10] = 00
01001	MAR ← PC	10101	If IR[11-10] = 01
01010	MAR ← X	10110	If IR[11-10] = 10
01011	MBR ← AC	10111	If IR[15-12] = MicroOp2[4-1]

a) (8 points) Explain why a microprogrammed control unit is slower than a hardwired control unit?

Hardwired control unit uses a circuit which is fast.
The microprogrammed control unit adds another level of interpretation since microinstructions are used to Fetch, Decode, and Execute each step (RTN). The decoding is especially slow due to the "Jump Table".

b) (7 points) The PC ← PC + 1 microinstruction at μAddresses 3 of the microprogram on the next page is the last line of the "Fetch", so it gets performed for every machine-language instruction. However, the JUMP and JUMPI instructions wipe out the PC value later when "Executed". Describe how we could modify the microprogram to eliminate this inefficiency. Two ways are possible;

① Move PC⇐PC+1 to start of execution for sequential instructions

② Move JUMP and JUMPI decode uinstr. before PC⇐PC+1

uaddr
0 MAR⇐PC
1 MBR⇐M[MAR]
2 IR⇐MBR
3 If JUMP
4 If JUMPI
5 PC⇐PC+1

c) (15 points) Extend the partial microprogram below to include microoperations to decode and implement the execution of the instructions: ADDI and JUMPI. **(Fill in only the bolded boxes)**

Revised Figure 4.21 Partial Microprogram

Part of Cycle	RTN (of MicroOp1)	μAddr	MicroOp1	MicroOp2	Jump	Dest
Fetch	MAR ← PC	0	01001	0000	0	0
	MBR ← M[MAR]	1	01100	0000	0	0
	IR ← MBR	2	00101	0000	0	0
	PC ← PC + 1	3	10000	0000	0	0
Decode	If ADD, Jump	4	10111	00110	1	
	If LOAD, Jump	5	10111	00010	1	
(“Jump	If STORE, Jump	6	10111	00100	1	
Table”)	If SKIPCOND, Jump	7	10111	10000	1	
	If SUBT, Jump	8	10111	01000	1	
	If JUMP, Jump	9	10111	10010	1	
	If ADDI, Jump	10	10111	10110	1	17_{10}
	If CLEAR, Jump	11	10111	10100	1	
	If JNS, Jump	12	10111	00000	1	
	If JUMPI, Jump	13	10111	11000	1	22_{10}
	If INPUT, Jump	14	10111	01010	1	
	If OUTPUT, Jump	15	10111	01100	1	
	If HALT, Jump	16	10111	01110	1	
Execute ADDI	MAR ← IR[11-0]	17	00111			
	MBR ← M[MAR]	18	01100			
	MAR ← MBR	19	01000			
	MBR ← M[MAR]	20	01100			
	AC ← AC + MBR	21	00011			
Execute JUMPI	MAR ← IR[11-0]	22	00111			
	MBR ← M[MAR]	23	01100			
	PC ← MBR	24	01111			