

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	<u>False</u>
b. Registers are storage locations within the CPU itself.	<u>True</u>	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.	<u>True</u>	False
d. The AC, MAR, MBR, PC, and IR registers in MARIE can be used to hold arbitrary data.	True	<u>False</u>
e. One assembly language instruction generally translates to one machine language instruction.	<u>True</u>	False
f. One high-level language (e.g., Ada, C++, Java, etc.) instruction generally translates to one machine language instruction.	True	<u>False</u>

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
 INPUT X
 END WHILE
 SUM = SUM + X

+3 (INPUT
 STORE X
 WHILE (SKIPCOND 000
 JUMP END_WHILE
 +4 + cond. branch
 +3 body (INPUT
 STORE X
 JUMP WHILE
 +3 Jump to make wsp
 END_WHILE (ADD SUM
 STORE SUM
 +4 post processing
 16 over

12

Question 3.

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

Symbol	Addr.	Symbol	Addr
IF	101	Y	10A
ELSE	105	ONE	10B
END_IF	108		
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

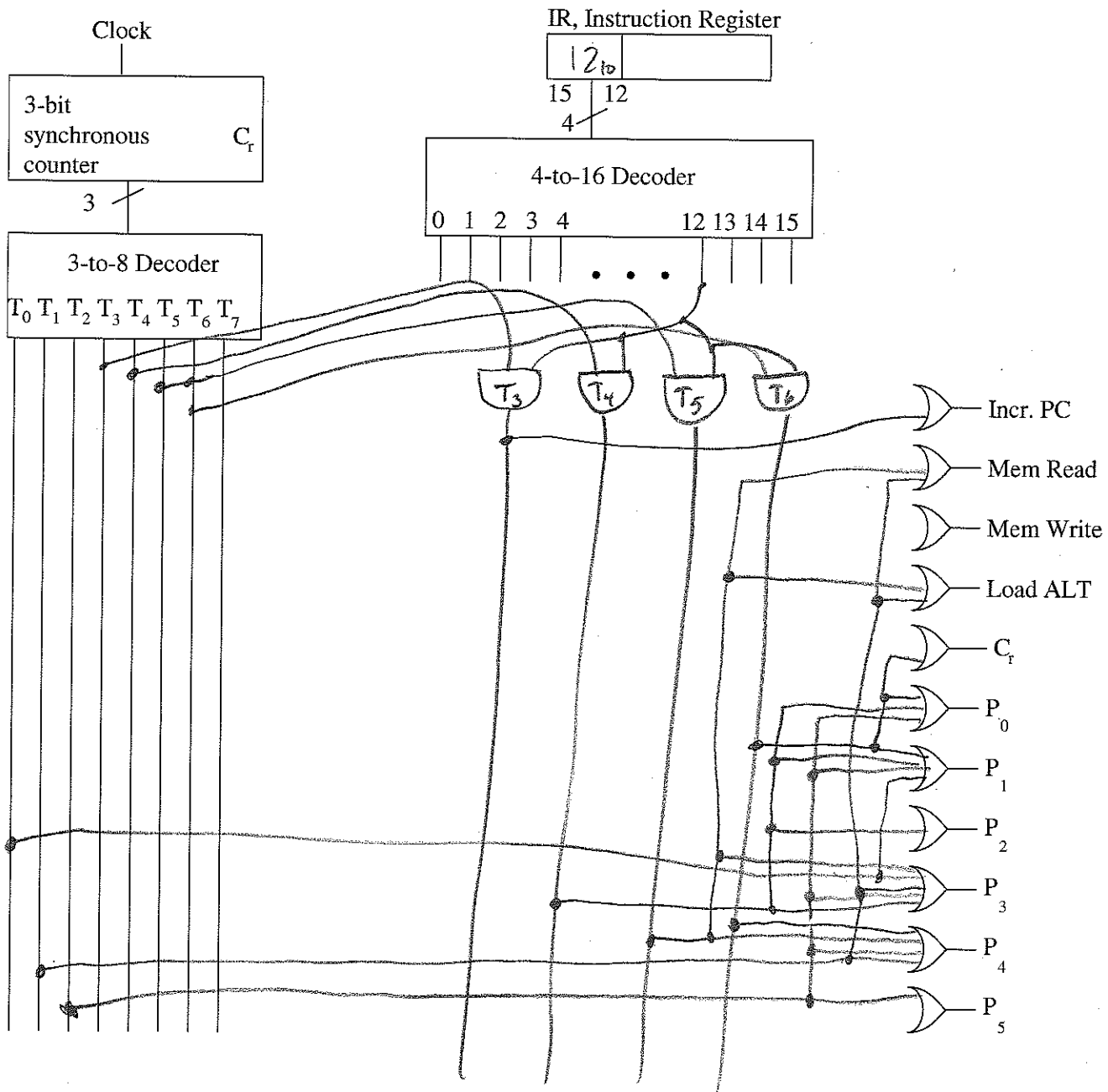
Y = X + Y - 1

ENDIF

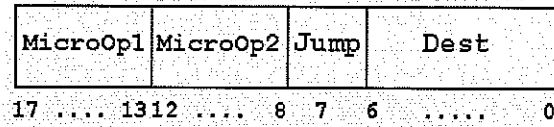
Question 4. (10 points) Which control signals should contain a "1" for each step in the JUMPI instruction?

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	MAR ← PC	T ₀	0	0	1	0	1	0	0	0	0	0	0
	MBR ← M[MAR]	T ₁	0	1	1	-	-	-	0	0	1	0	1
	IR ← MBR	T ₂	1	1	1	0	1	1	0	0	0	0	0
Decode IR[15-12]	PC ← PC + 1	T ₃	0	0	0	-	-	-	0	1	0	0	0
Get operand	MAR ← IR[11-0]	T ₄	0	0	1	1	1	1	0	0	0	0	0
Execute	MBR ← M[MAR]	T ₅	0	1	1	-	-	-	0	0	1	0	1
	PC ← MBR	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 C₁₆) instruction.



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



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)

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?

Decoding in hardwired takes only a short time since it runs through a decoder. In microprogrammed control, multiple clocks needed to decoder in the "Jump Table" part of the microprogram.

b) (7 points) The PC ← PC + 1 microinstruction at μ Addresses 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:

- 0 MAREPC
- 1 MBR ←
- 2 IRE MBR
- 3 IF JUMP, Jump
- 4 IF JUMPI, jump
- 5 PC ← PC + 1
- 6 IF JUMP, ...

Decode

Put PC ← PC + 1 at start of execution of instructions that need it.

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)	μ Addr	MicroOp1	MicroOp2	Jump	Dest
Fetch	MAR \leftarrow PC	0	01001	0000	0	0
	MBR \leftarrow M[MAR]	1	01100	0000	0	0
	IR \leftarrow MBR	2	00101	0000	0	0
	PC \leftarrow PC + 1	3	10000	0000	0	0
Decode	If ADD, Jump	4	10111	00110	1	
	If LOAD, Jump	5	10111	00010	1	
("Jump Table")	If STORE, Jump	6	10111	00100	1	
	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
	If CLEAR, Jump	11	10111	10100	1	
	If JNS, Jump	12	10111	00000	1	
	If JUMPI, Jump	13	10111	11000	1	22
	If INPUT, Jump	14	10111	01010	1	
	If OUTPUT, Jump	15	10111	01100	1	
Execute ADDI	If HALT, Jump	16	10111	01110	1	
	MAR \leftarrow IR[11-0]	17	00111	-	0	-
	MBR \leftarrow M[MAR]	18	01100	-	0	-
	MAR \leftarrow MBR	19	00000	-	0	-
	MBR \leftarrow M[MAR]	20	01100	-	0	-
	AC \leftarrow AC + MBR	21	00011	-	1	0
Execute JUMPI	MAR \leftarrow IR[11-0]	22	00111	-	0	-
	MBR \leftarrow M[MAR]	23	01100	-	0	-
	PC \leftarrow MBR	24	01111	-	1	0

15 8 5 2 5