Homework #5 : 5.15, 5.20, 5.23, 5.25 Matt Krueger

5.15 [E] We have seen how all RISC-style instructions can be executed using the steps in Figure 5.4 on the multi-stage hardware of Figure 5.8. Autoincrement and Autodecrement addressing modes are not included in RISC-style instruction sets. Explain why the instruction

Load R3, (R5)+ // retrieves data from EA: [R5] + 4 & places into R3

cannot be executed on the hardware in Figure 5.8.

Characteristics:

RISC:
- Single word Instructions
- work done on registers
- simple addressing modes
- ★ load Store

CISC:
- instructions ≥1 word(s)
- work done in memory
- complex addressing modes
- ★ not constrained to load/store

Autoincrement & Autodecrement:
- useful in stacks
- apply offset to pointer (pointing to registers) either before (pop) or after (push)

Why can't RISC Accommodate these?

CISC can operate directly in memory while RISC cannot (due to its adherence to load/store). This means CISC is able to perform operations such as increments/decrements on values registers point to in memory.

Equivalent RISC instructions:

LOAD R3, (R5) // load pointer to memory location into R3

ADD R3, R3, #4 // apply increment on contents in R3 assuming word length of 32 bits (4 bytes)

ANSWER

5.20 [M] Consider the actions needed to execute the instructions given in Section 5.4.1. Derive the logic expressions to generate the signals C_select, MA_select, and Y_select in Figures 5.18 and 5.19 for these instructions.

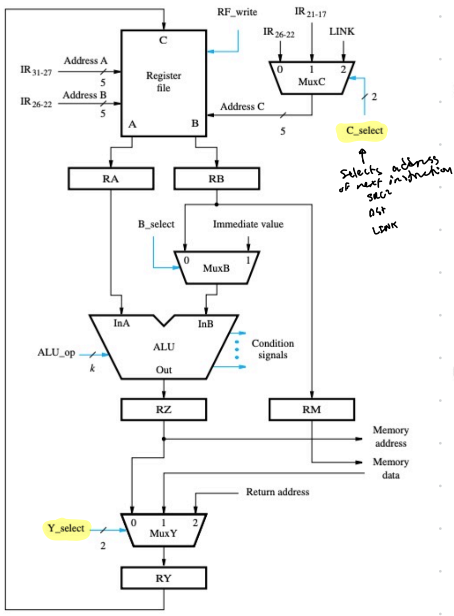


Figure 5.18 Control signals for the datapath.

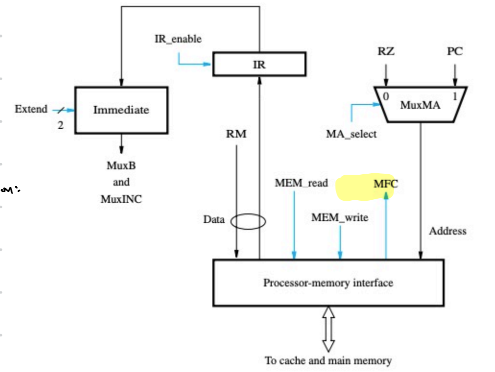
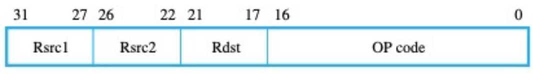


Figure 5.19 Processor-memory interface and IR control signals.



(a) Register-operand format

✱ There are multiple Instructions Given; I am not sure of which to use

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction
3	PC ← [PC] + Branch offset
4	No action
5	No action

1. unconditional branch

No Logic; branches to "Loop" regardless

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R5], RB ← [R6]
3	Compare [RA] to [RB], If [RA] = [RB], then PC ← [PC] + Branch offset
4	No action
5	No action

2. Branch_if_[R5] = [R6]

if contents in R5 = R6 → branch

C_select:

$([R5]=[R6]) \cdot (PC+1) + \overline{([R5]=[R6])} \cdot Loop$

MA_Select: 0

Y_Select: 0

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction, RA ← [R9]
3	PC-Temp ← [PC], PC ← [RA]
4	RY ← [PC-Temp]
5	Register LINK ← [RY]

3. Call_Register R9

Simply call register contents in R9

C_select: R9

MA_Select: 0

Y_Select: 0

5.23 [M] Derive the logic expressions to generate the signals PC_select and INC_select shown in Figure 5.20, taking into account the actions needed when executing the following instructions:

B : Branch: All branch instructions, with a 16-bit branch offset given in the instruction

C : Call_register: A subroutine-call instruction with the subroutine address given in a general-purpose register

O : Other: All other instructions that do not involve branching

Not muxes; but instruction is 1 of 3 vals, which determine select signals for MuxPC & MuxINC

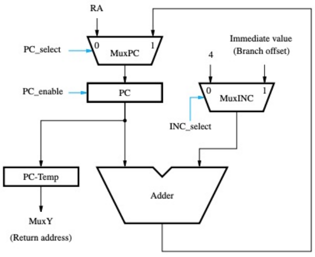


Figure 5.20 Control signals for the instruction address generator.

P1	P0	b	y	
0	0	0		m0
0	0	1		m1
0	1	0		m2
0	1	1		m3
1	0	0		m4
1	0	1		m5
1	1	0		m6
1	1	1		m7

P1 \ P0b	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

	PC_select	INC_select
If branch →	1	0
If call →	1	0
If other →	0	1

B	C	O	PC_Select	
0	0	0	x	impossible
0	0	1	1	OTHER
0	1	0	0	CALL
0	1	1	x	impossible
1	0	0	0	BRANCH
1	0	0	x	impossible
1	1	1	x	impossible
1	1	0	x	impossible

B	C	O	INC_Select	
0	0	0	x	impossible
0	0	1	0	OTHER
0	1	0	1	CALL
0	1	1	x	impossible
1	0	0	1	BRANCH
1	0	0	x	impossible
1	1	1	x	impossible
1	1	0	x	impossible

C′ C

B \ CO	00	01	11	10
0	X_1	0_0	X_0	1
1	1	X_0	X_0	X_1

O′ O O′

PC_Select: O′

O′

C′ C

B \ CO	00	01	11	10
0	X	1	X	0
1	0	X	X	X

O′ O O′

INC_Select: O

O

5.25 [M] Give the sequence of steps needed to fetch and execute the instruction

Load R3, (R5)+

on the processor of Figure 5.24. Assume 32-bit operands.

LOAD R3, (R5)+

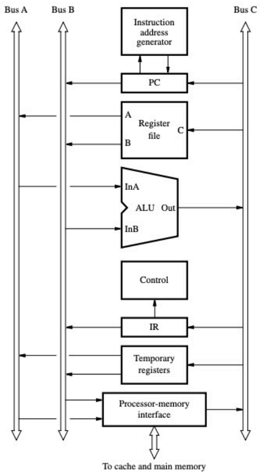


Figure 5.24 Three-bus CISC-style processor organization.

1: //Fetch Instruction
memory Address ← [PC],
read Memory,
wait for Memory function complete signal (MFC),
IR ← memory data,
PC ← [PC]+4

2: //Decode Instruction
Decode Instruction

3: //Execute Instruction ; temp1 holds memory address, PC incremented again
memory Address ← [PC],
read Memory,
wait for Memory function complete signal (MFC),
Temp1 ← memory data,
PC ← [PC]+4

4: // Temp 2 holds address in R3 (Temp1)
Temp2 ← [Temp1] = [R3]

5: // Temp 1 applies Register offset
Temp1 ← [Temp1] + [R5]

6: // write back
memory Address ← [Temp2],
Memory Data ← [Temp1],
Write memory,
wait for Memory function complete signal (MFC)