CS251 Winter 2015
Assignment 04
Due Friday March 20 1pm
45 Total Marks

Q1 (10 marks) Consider the assembly language instruction

4000: sw $2, 100($1)

This 32-bit instruction in binary would look like the following:

Opcode	$rs	$rt	Offset
101011	00001	00010	0000000001100100

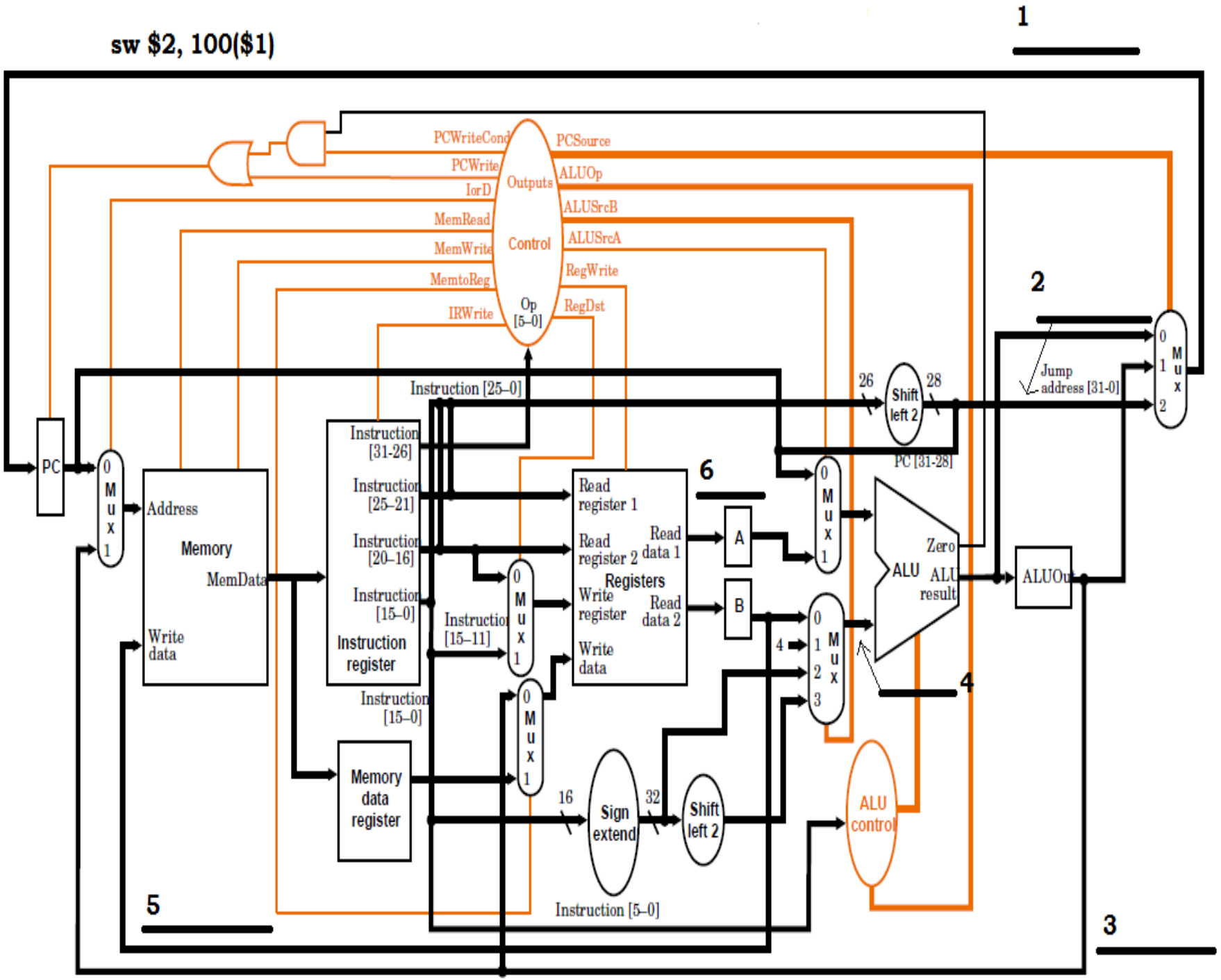
(a) (6 marks) On the next page, there is a figure of the multi-cycle datapath with six dark lines. On each line, write in the value that travels along the corresponding wire(s) when executing this assembly language instruction on the **4th clock cycle**. You may assume during the first clock cycle we execute state 0 of the finite state machine for this datapath. Note: you should write a decimal number on each dark line, and not an expression involving things like 'PC'.

Assume that each register $i (with i > 0) contains the decimal value 1000 + i.

(b) (4 mark) In the table below write the value of each control line used to execute this instruction during the 1st, 2nd, 3rd and 4th clock cycles. You should list the control value of a signal even if it isn't listed in the bubble in the FSM diagram (assume non-listed values would be *Don't Cares* or set to 0 as appropriate)

Control Signal	**Clock Cycle1**	**Clock Cycle2**	**Clock Cycle3**	**Clock Cycle4**
PCWriteCond				
PCWrite				
IorD				
MemRead				
MemWrite				
MemtoReg				
IRWrite				
PCSource				
ALUOp				
ALUSrcA				
ALUSrcB				
RegWrite				
RegDst				

sw $2, 100($1)


1
2
3
4
5
6
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0 M u x 1
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–11]
Instruction [15–0]
Instruction [25–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
0 1 2 3 M u x
Memory data register
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]

Q2: (10 marks) Consider the following MIPS assembly language instruction:

100: maddi $2 100($3) : maddi M[$rt] ← M[$rs] + immediate

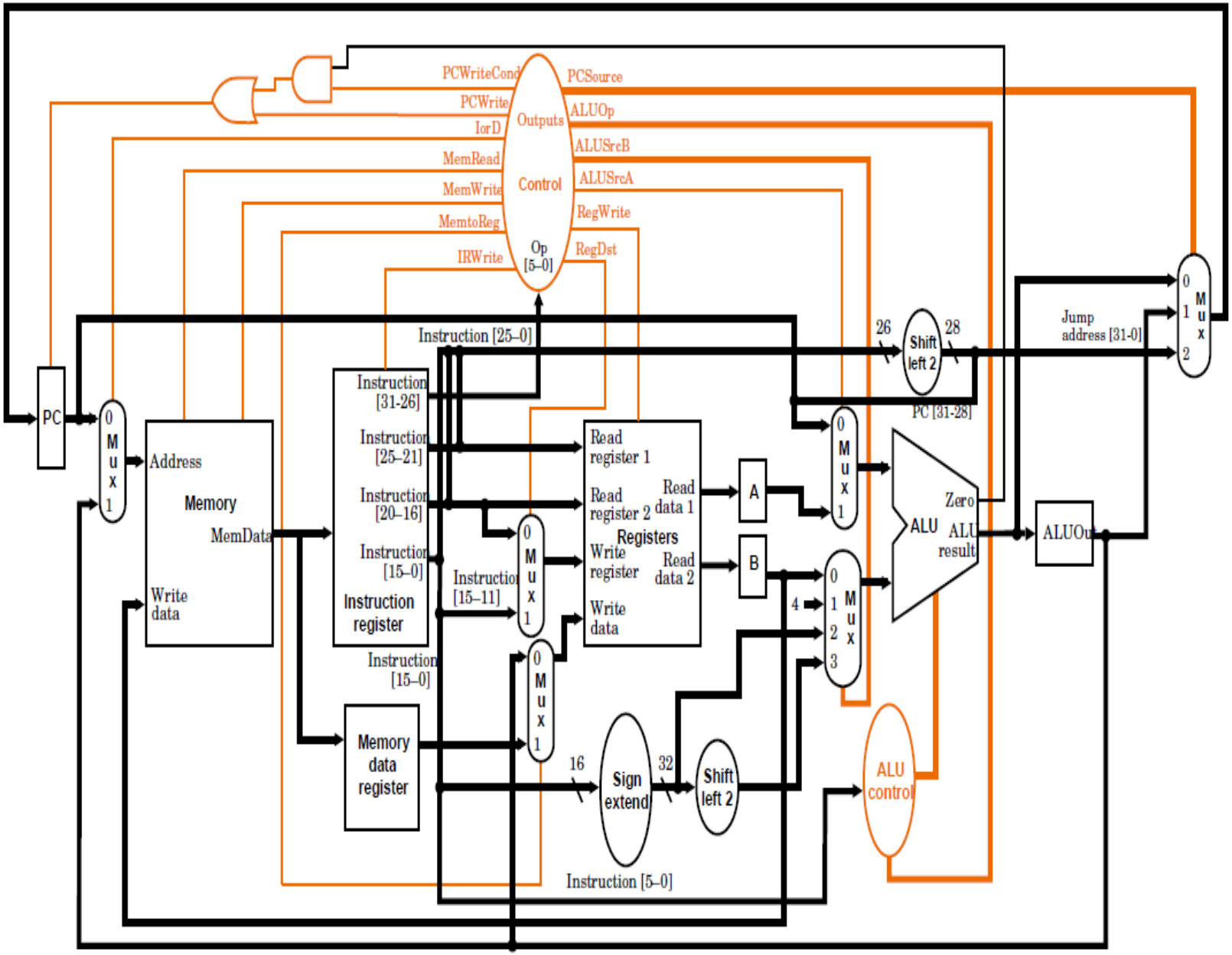
This instruction was discussed in class. The *maddi (memory add immediate)* instruction will compute a 32-bit value by adding the 16 bit immediate value to Memory[$rs]. The computed value will then be stored in Memory[$rt]

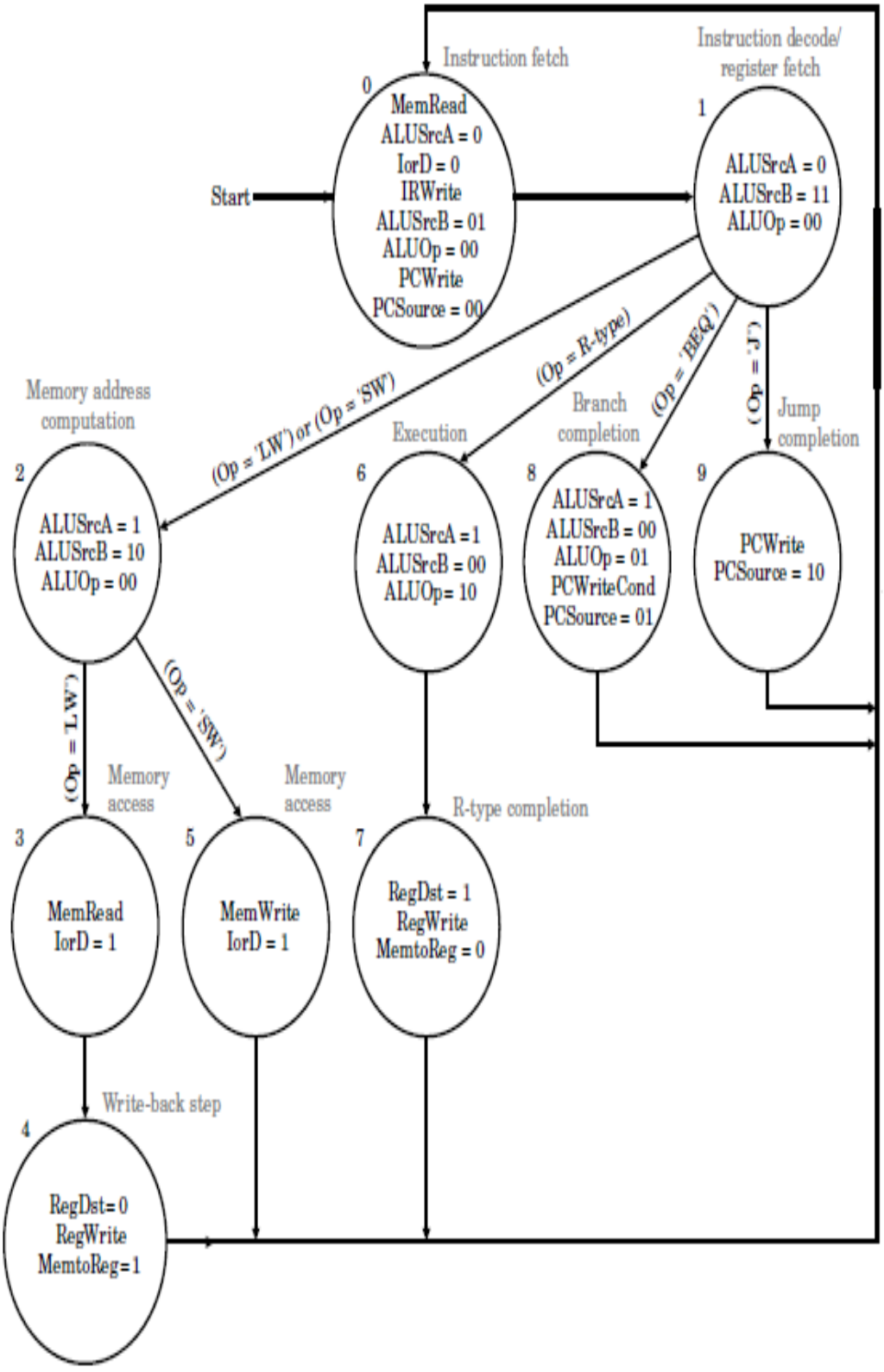
Effect: M[$rt] ← immediate + M[$rs]

a) (5 marks) Modify the Multi-cycle datapath on the next page to incorporate this new instruction exactly as we did in class. You may also add new control lines as needed, and make structural changes to the datapath again, as we did in class. (lecture slides : March 03)
All other instructions covered in class, should still work.

b) (5 marks) Make any necessary changes to the Finite State Machine that implements the Multi-cycle control unit. You must indicate the value of all new and existing control lines necessary to execute the **maddi** instruction on the datapath. Add in new states as appropriate. New states must be labeled.

Complete Multicycle Datapath




Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
Instruction decode/
register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch
completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1

Q3: (7 marks) Consider the following MIPS assembly language instruction:

100: copy $2, $3, $4 : copy $rs, $rt, $rd

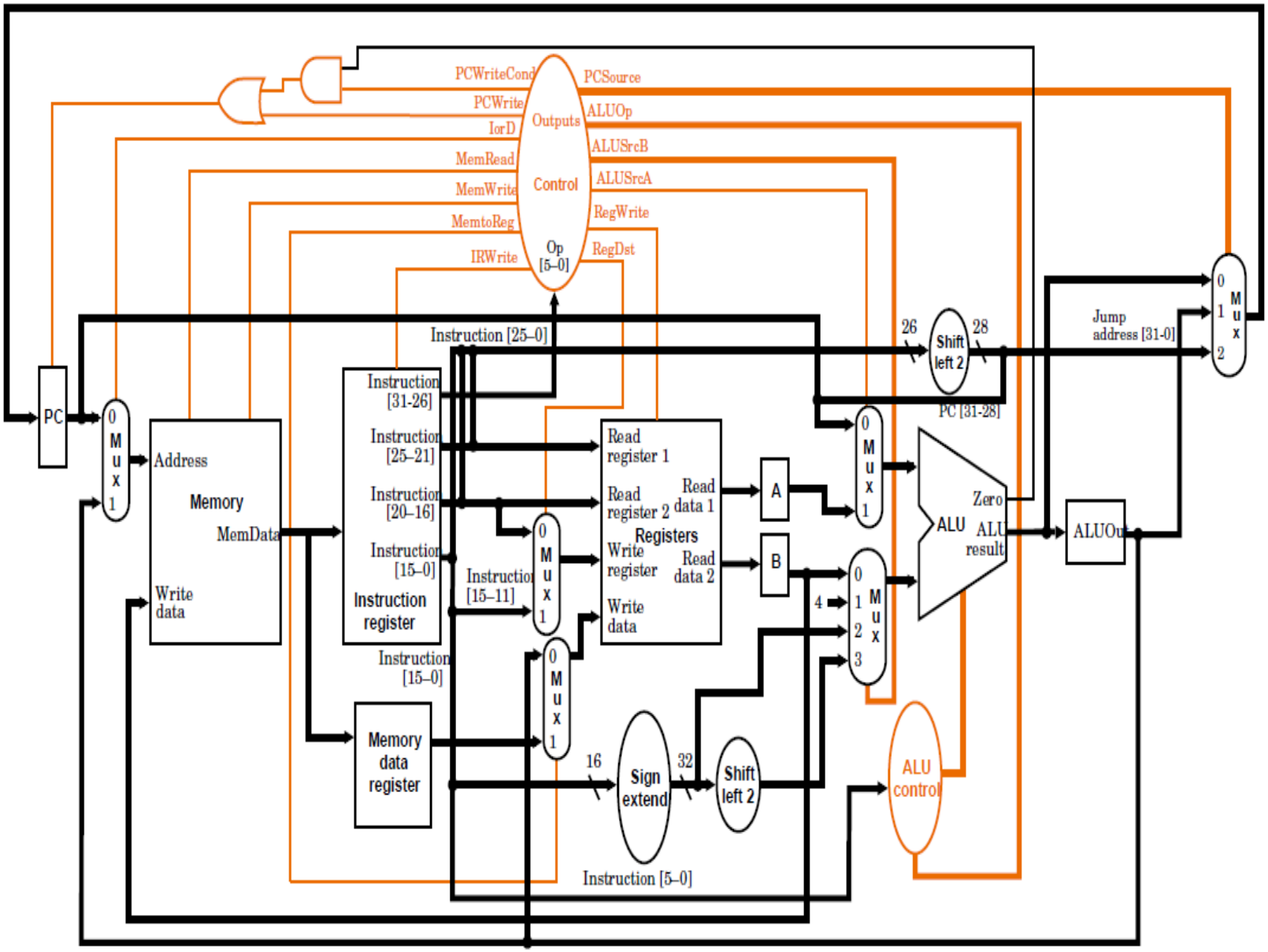
This instruction simply copies the value of $rs into $rt and $rd. This is an R-format instruction, however the shamt and function bits are unused.

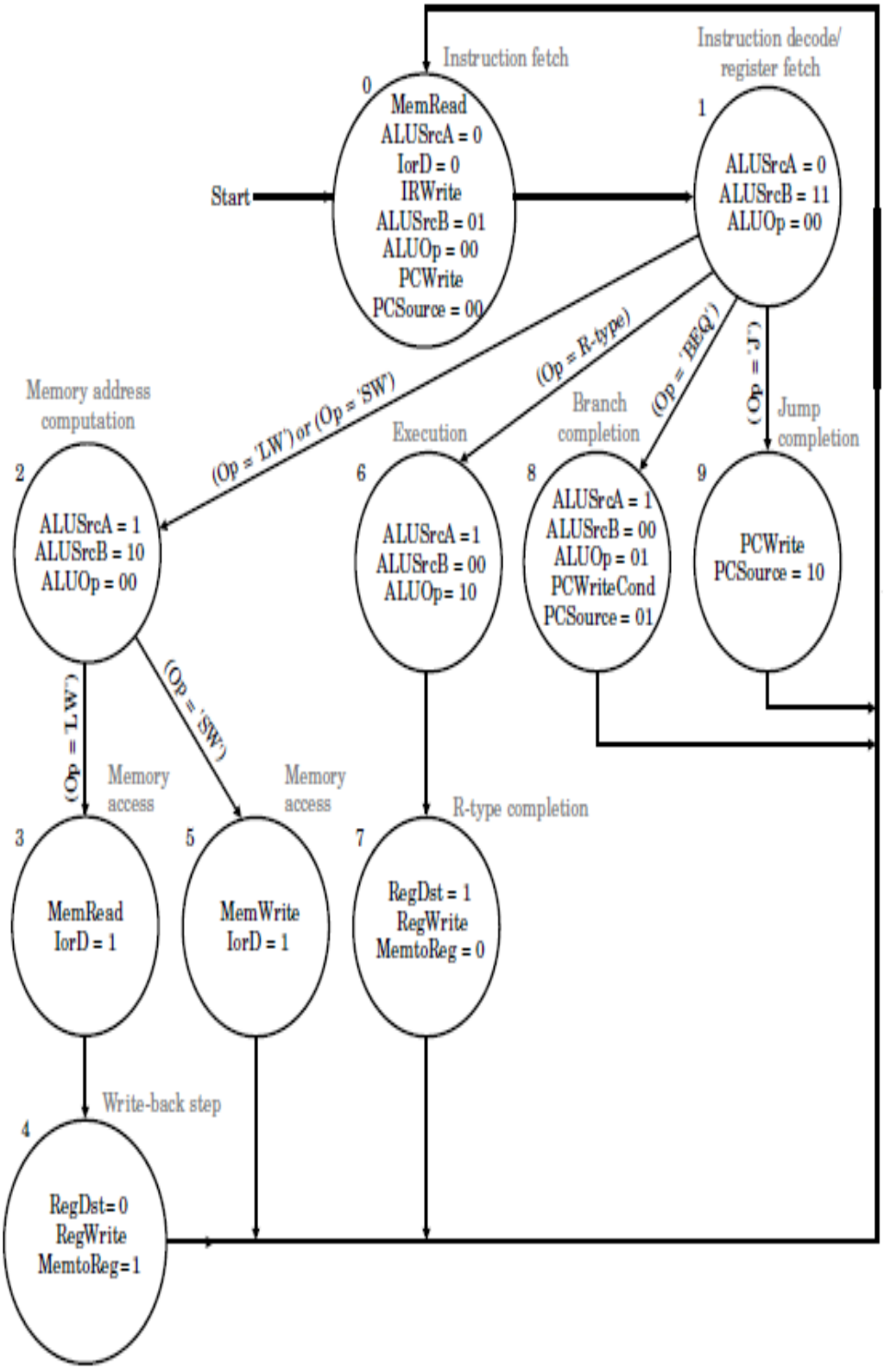
Effect: $rd ← $rs
$rt ← $rs

a) (2 marks) Modify the Multi-cycle datapath on the next page to incorporate this new instruction. You may also add new control lines as needed, and make structural changes to the datapath as needed. However to receive full marks, your solution should re-use as much of the datapath as possible and complete in the fewest number of steps.
All other instructions covered in class, should still work.

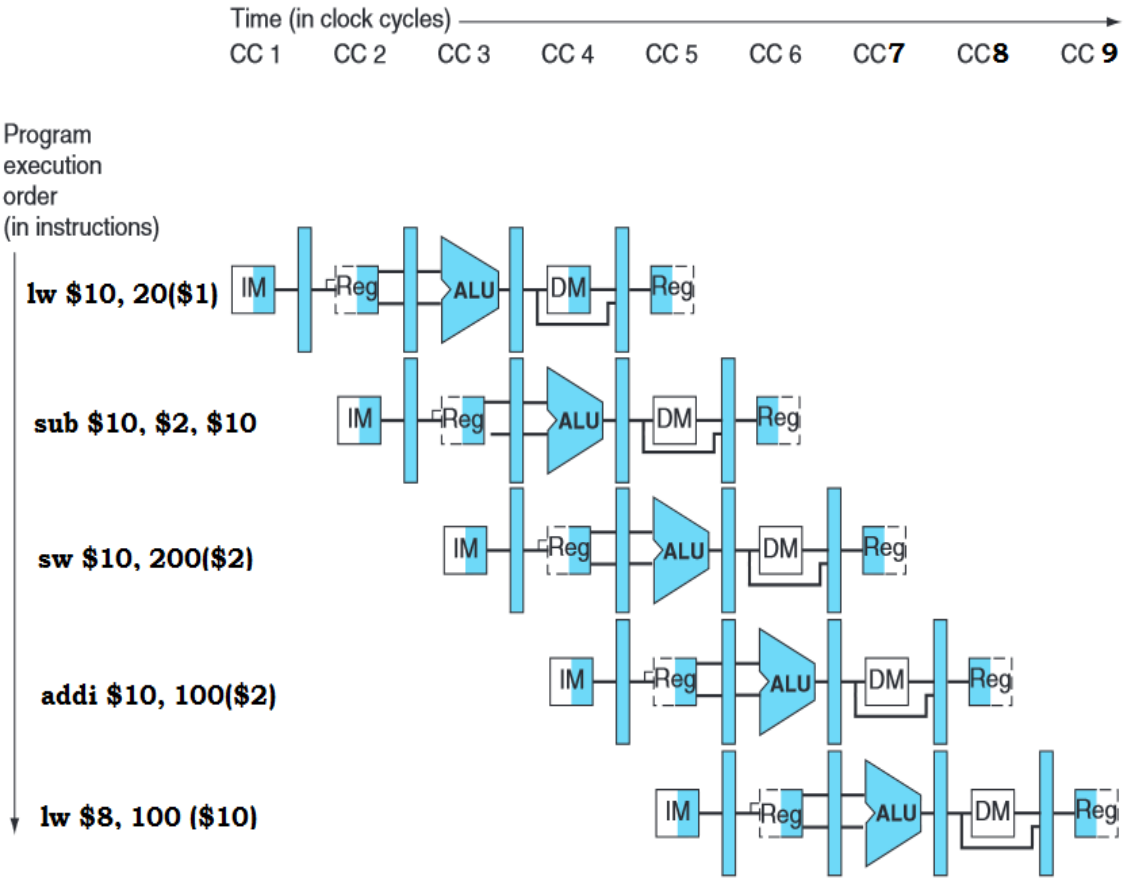
b) (5 marks) Make any necessary changes to the Finite State Machine that implements the Multi-cycle control unit. You must indicate the value of all new and existing control lines necessary to execute the **copy** instruction on the datapath. Add in new states as appropriate and the value of the control lines in these states.

Complete Multicycle Datapath




Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
Instruction decode/
register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch
completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1

Q4. (3 marks) In the diagram below, mark all data dependencies by drawing straight lines (similar to the black lines on figure 6.6 of the course notes). Assume that the code is to run on the pipelined datapath that *does not* implement forwarding).
For each data dependency, label it either as a hazard or as a non-hazard, and assume ***no delay*** from other units.

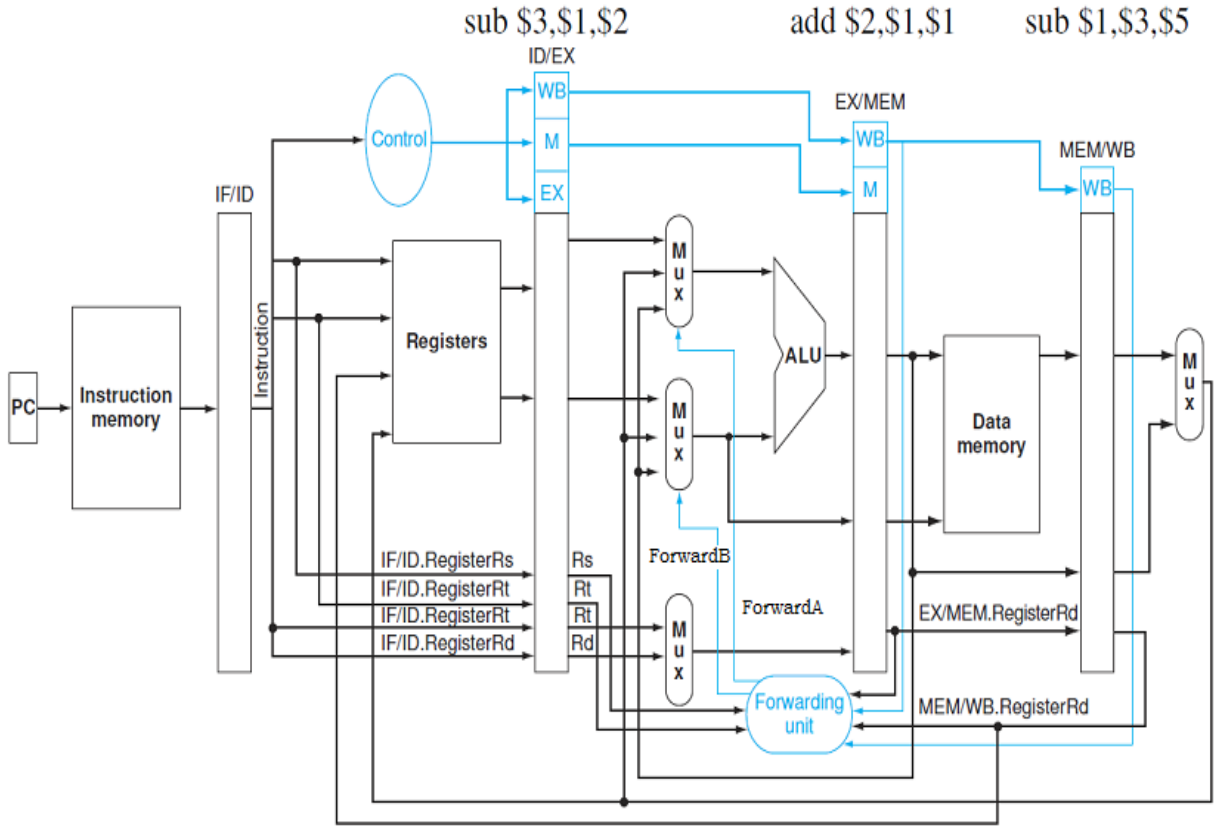


b)(4 marks) Re-write the code segment from part(a) inserting nops where necessary on a datapath that *does not* implement forwarding.

c)(1 mark) Re-write the code segment from part(a) inserting nops where necessary on a datapath that *does* implement forwarding. (assume the forwarding datapath in Q5 below)

Q5. (5 marks) This question refers to the pipelined datapath with forwarding as given below.
Consider the instructions:
100 sub $1, $3, $5
104 add $2, $1, $1
108 sub $3, $1, $2

Consider the situation when the 100 sub instruction is in the WB stage, the 104 add instruction is in the MEM stage, and the 108 sub instruction is in the EX stage. In the figure below,
trace back each of the two inputs to the ALU through the MUXes back to the appropriate set of pipeline registers. State the values of the select lines, ForwardA and ForwardB.



Q6: (5 marks)
Given the following code segments below, re-write the code (code rearrangement to avoid as many stalls as possible). Assume the datapath implements data forwarding and branches are completed in the ID stage, however flushing does not occur in this datapath. You may insert a nop instruction, only if code rearrangement cannot be used.

add $5, $6, $3	
lw $2, 100($3)	
sw $3, 200($2)	
add $5, $4, $3	
subi $2, $2 ,-4	
subi $4, $5, 100	
lw $3, 100($2)	
sw $3, 100($4)	
beq $3, $2, 28	
lw $5, 200($6)	
add $3, $2, $1	