Chapter 4 Problems (part 2)

Question 1 (past exam question)

Consider our initial pipelined MIPS machine which has 5 pipe stages: (IF ID EX MEM WB) Assume that for a conditional branch instruction, the target address is computed in the second stage (ID stage) and the branch outcome (i.e., branch decision) is determined in the fourth stage (MEM stage). Assume that 30% of all instructions are conditional branches and that 75% of these are taken. Assume an ideal CPI of 1. We want to study the effect of various techniques used for reducing the pipeline branch penalties. Ignore all other types of hazards.

a. Compute the actual CPI if no technique is used; i.e., pipeline is stalled until the branch is complete.

Branch completed

```
Branch: IF ID EX MEM | WB
                     | IF  ──▶  Next instr. is fetched only after this decision
```

CPI = 1 + 0.30 x 3 = 1.9

Question 1.b

b. Compute the actual CPI if the branch is always predicted to be *taken*.

- If predicted taken but branch is not taken

IF	ID	EX	MEM	WB
	IFold	IFnew	IDnew	IFold

3 stalls

- If predicted taken and branch is taken

IF	ID	EX	MEM	WB
	IF	IFnew	IDnew	EXnew

1 stall

CPI = 1 + 0.3 [0.25 * 3 + 0.75 *1] = 1.45

Question 1.c

c. Compute the actual CPI if the branch is always predicted to be *not taken*.

- If predicted not taken but branch is not taken

IF	ID	EX	MEM	WB	
	IFnew	IDnew	EXnew	MEMnew	WBnew

No stalls!

- If predicted not taken and branch is taken

IF	ID	EX	MEM	WB
	IF	ID	EX	IFnew

3 stalls

CPI = 1 + 0.3 [0.25 * 0 + 0.75 *3] = 1.675

Question 2

Assume that we have the following MIPS code.

```
lw    $s5, 200 ($s4)
add   $s6,  $s5, $s8
add   $s7,  $s5, $s9
sub   $s3,  $s7, $s10
```

What is the total number of cycles required to execute the above code on a pipelined MIPS. Draw the pipeline chart and show all necessary stalls and/or forwarding.

Question 2.a

- NO data forwarding; and it is NOT allowed to write and read a register at the same cycle

```
lw   $s5, 200 ($s4)
add $s6, $s5, $s8
add $s7, $s5, $s9
sub $s3, $s7, $s10
```

IF	ID	EX	MEM	WB									
	IF	S	S	S	ID	EX	MEM	WB					
					IF	ID	EX	MEM	WB				
						IF	S	S	S	ID	EX	MEM	WB

# of cycles = 14

(3 + 3 = 6 stalls)

Question 2.b

- NO data forwarding; and it is allowed to write and read a register at the same cycle

```
lw   $s5, 200 ($s4)
add $s6, $s5, $s8
add $s7, $s5, $s9
sub $s3, $s7, $s10
```

IF	ID	EX	MEM	WB							
	IF	S	S	ID	EX	MEM	WB				
				IF	ID	EX	MEM	WB			
					IF	S	S	ID	EX	MEM	WB

# of cycles = 12

(2 + 2 = 4 stalls)

Question 2.c

4

- data forwarding; and it is allowed to write and read a register at the same cycle

```
lw   $s5, 200 ($s4)
add $s6, $s5, $s8
add $s7, $s5, $s9
sub $s3, $s7, $s10
```

IF	ID	EX	MEM	WB				
	IF	ID	S	EX	MEM	WB		
		IF	S	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

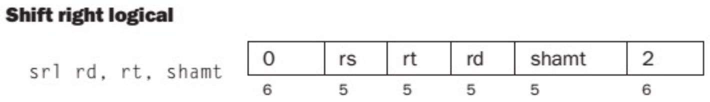
# of cycles = 9

(1 stall, 2 forwarding)

Question 3 (past exam question)

Assume that we extend our single-cycle MIPS implementation so that it handles the “srl”(shift right logical) instruction.

srl $t1,$t2, 4



Shift right logical

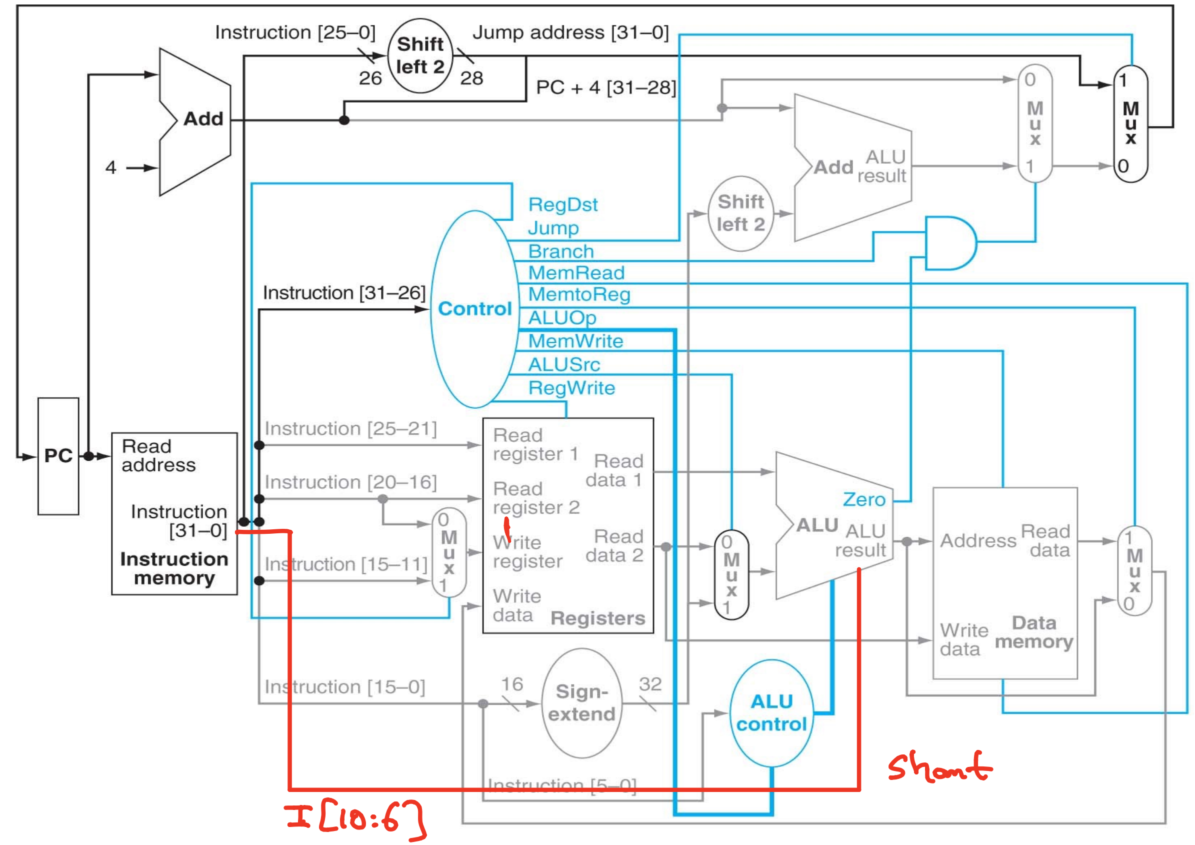
`srl rd, rt, shamt`

0	rs	rt	rd	shamt	2
6	5	5	5	5	6

Note: $rs = \emptyset$ (R-type)

a. **Explain** the required changes in the complete single-cycle datapath clearly (including changes in any component of the datapath including ALU, additional wires, muxes and control/selector signals). In case of no change in the datapath, then express it accordingly.

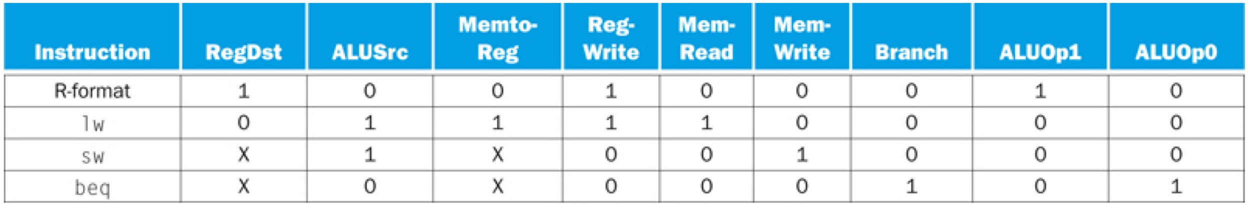
Single Cycle Datapath



Question 3.b

b. For the srl instruction, write the values of control lines given in Figure 4.18 at page 323 in new edition appropriately (i.e., 0, 1 or x for don't care). Include any new control signal(s) if needed.

RegDst	ALUsrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop
1	0	0	1	0	0	0	10



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Question 4 (From Textbook)

4.7 In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:

10101100011000100000000000010100.

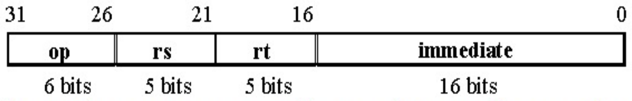
Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	–1	2	–3	–4	10	6	8	2	–16

Question 4.7 (Textbook)

4.7 In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:

10101100011000100000000000010100.



Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	–1	2	–3	–4	10	6	8	2	–16

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

①

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	`add`	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	`addi`	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	`addiu`	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	`addu`	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	`and`	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	`andi`	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	`beq`	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	`bne`	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	`j`	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	`jal`	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	`jr`	R	PC=R[rs]		$0 / 08_{hex}$
Load Byte Unsigned	`lbu`	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	`lhu`	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	`ll`	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	`lui`	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	`lw`	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	`nor`	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$
Or	`or`	R	R[rd] = R[rs] | R[rt]		$0 / 25_{hex}$
Or Immediate	`ori`	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	`slt`	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		$0 / 2a_{hex}$
Set Less Than Imm.	`slti`	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	`sltiu`	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	`sltu`	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0 / 2b_{hex}$
Shift Left Logical	`sll`	R	R[rd] = R[rt] << shamt		$0 / 00_{hex}$
Shift Right Logical	`srl`	R	R[rd] = R[rt] >>> shamt		$0 / 02_{hex}$
Store Byte	`sb`	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	`sc`	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	`sh`	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	`sw`	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	`sub`	R	R[rd] = R[rs] - R[rt]	(1)	$0 / 22_{hex}$
Subtract Unsigned	`subu`	R	R[rd] = R[rs] - R[rt]		$0 / 23_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	`bc1t`	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	`bc1f`	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	`div`	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	`divu`	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	`add.s`	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	`add.d`	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	`c.`*x*`.s`*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	`c.`*x*`.d`*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is `eq`, `lt`, or `le`) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	`div.s`	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	`div.d`	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	`mul.s`	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	`mul.d`	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	`sub.s`	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	`sub.d`	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	`lwc1`	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	`ldc1`	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	`mfhi`	R	R[rd] = Hi		0 /--/--/10
Move From Lo	`mflo`	R	R[rd] = Lo		0 /--/--/12
Move From Control	`mfc0`	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	`mult`	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	`multu`	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	`sra`	R	R[rd] = R[rt] >> shamt		0/--/--/3
Store FP Single	`swc1`	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	`sdc1`	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		

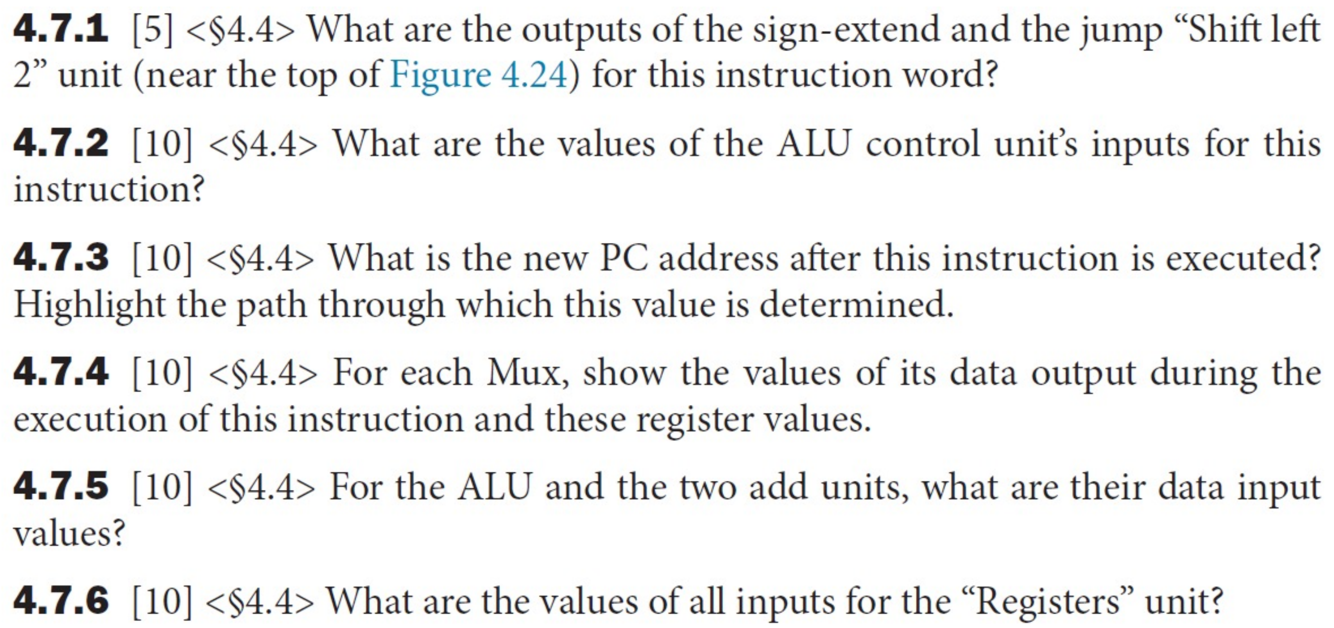
PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	`blt`	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	`bgt`	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	`ble`	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	`bge`	if(R[rs]>=R[rt]) PC = Label
Load Immediate	`li`	R[rd] = immediate
Move	`move`	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Question 4.7 (Textbook)



4.7.1 [5] <§4.4> What are the outputs of the sign-extend and the jump "Shift left 2" unit (near the top of Figure 4.24) for this instruction word?

4.7.2 [10] <§4.4> What are the values of the ALU control unit's inputs for this instruction?

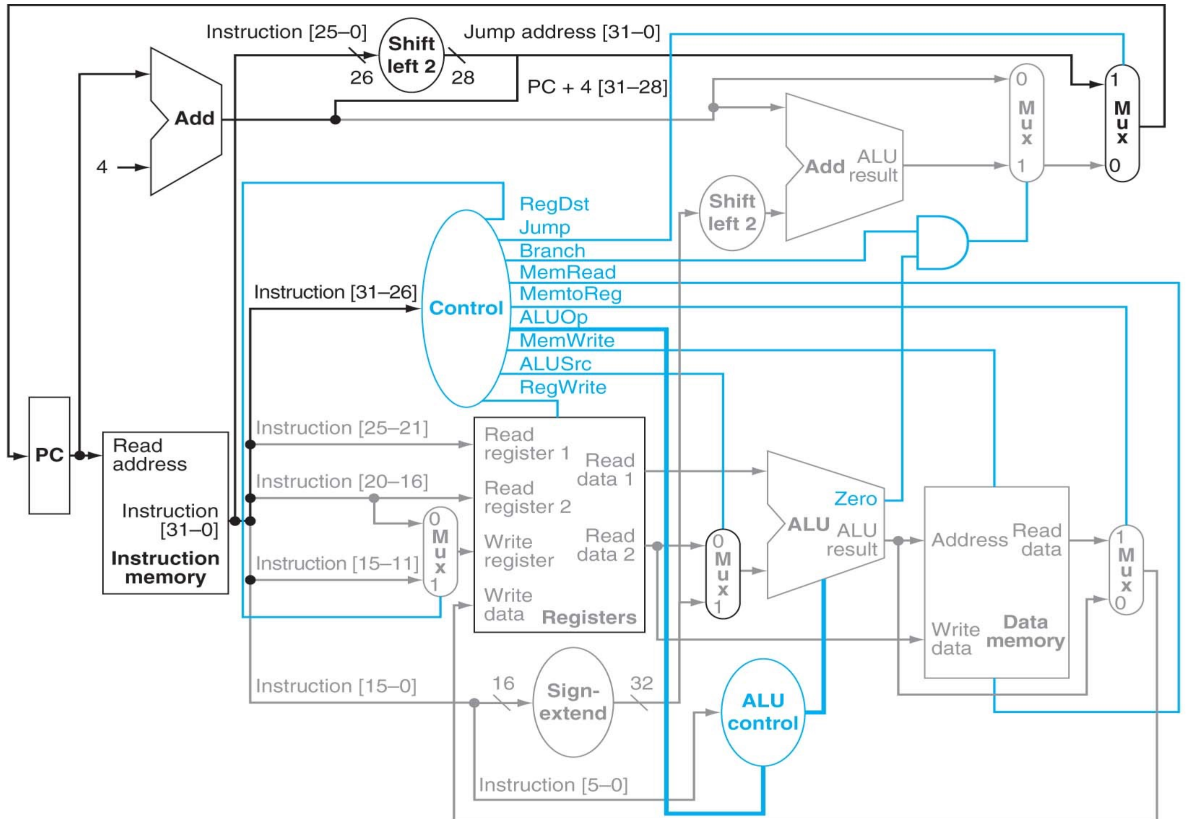
4.7.3 [10] <§4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

4.7.4 [10] <§4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.

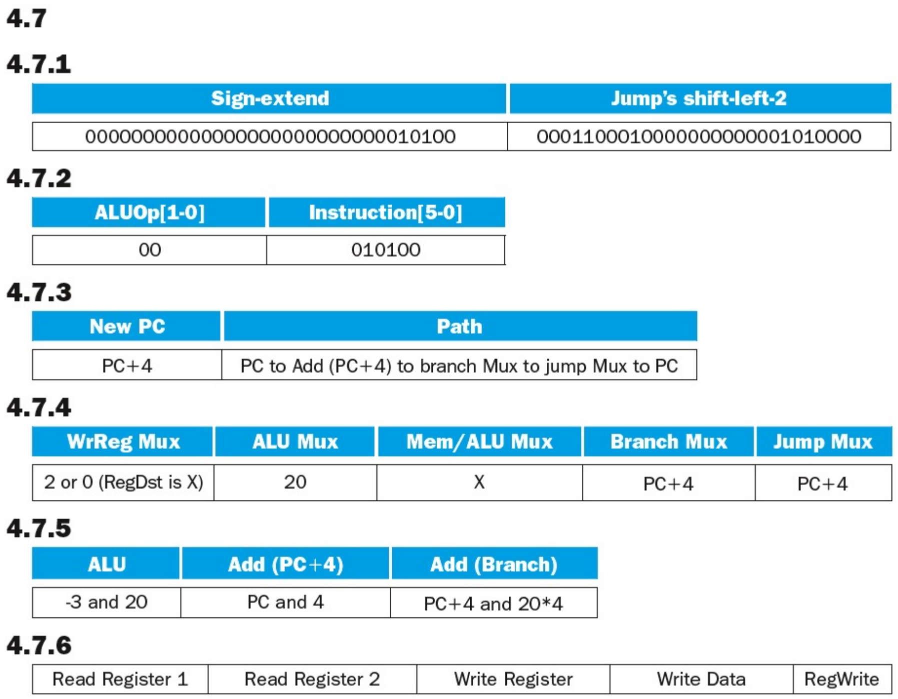
4.7.5 [10] <§4.4> For the ALU and the two add units, what are their data input values?

4.7.6 [10] <§4.4> What are the values of all inputs for the "Registers" unit?

Single Cycle Datapath



Question 4.7 (Book - Solution)



4.7

4.7.1

Sign-extend	Jump's shift-left-2
00000000000000000000000000010100	0001100010000000000001010000

4.7.2

ALUOp[1-0]	Instruction[5-0]
00	010100

4.7.3

New PC	Path
PC+4	PC to Add (PC+4) to branch Mux to jump Mux to PC

4.7.4

WrReg Mux	ALU Mux	Mem/ALU Mux	Branch Mux	Jump Mux
2 or 0 (RegDst is X)	20	X	PC+4	PC+4

4.7.5

ALU	Add (PC+4)	Add (Branch)
-3 and 20	PC and 4	PC+4 and 20*4

4.7.6

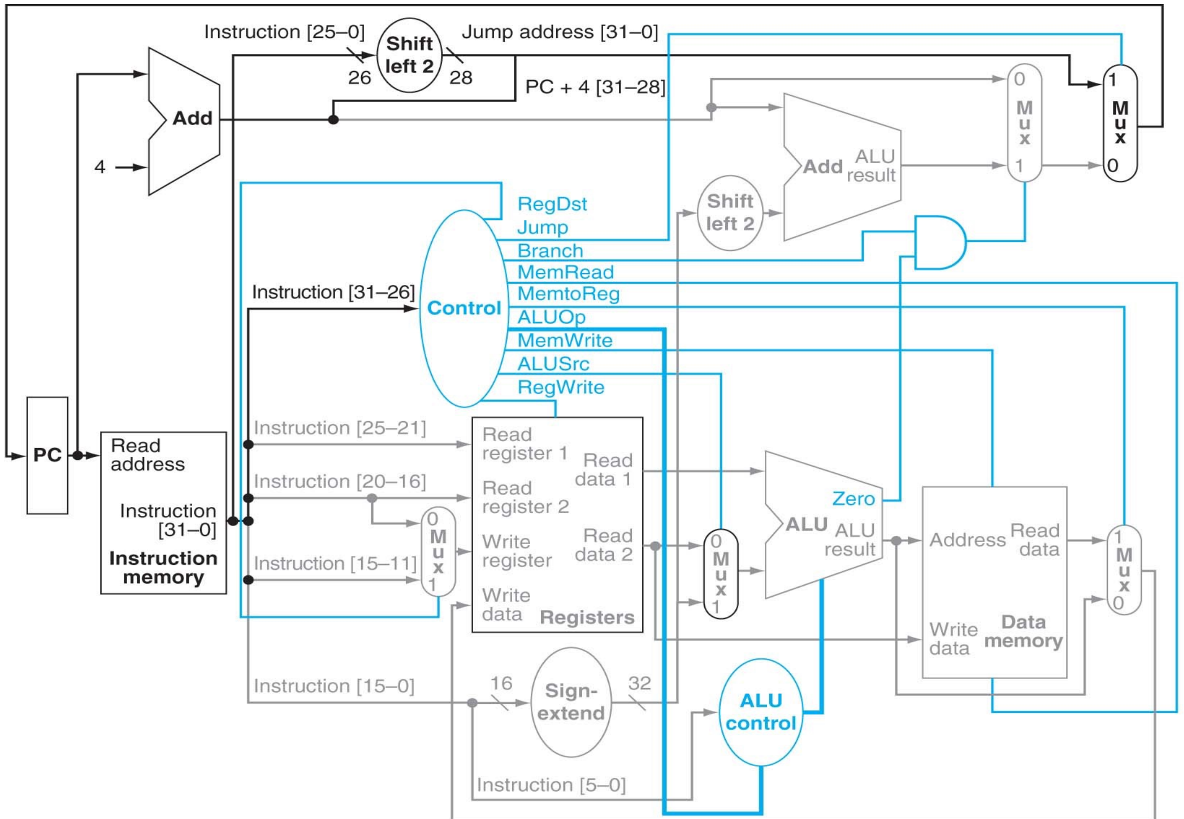
Read Register 1	Read Register 2	Write Register	Write Data	RegWrite

Question 5

We wish to add the instruction **lui (load upper immediate)** to the single cycle datapath described in the chapter. Add any necessary datapaths and control signals to the single-cycle datapath of the Figure 4.24 which represents the datapath and show the necessary additions (if any) to the Figure 4.18 which represents the setting of the control lines.

lui $s1, 20

Single Cycle Datapath



Question 5

Answer: (There might be different solutions)

- Add Shift unit which takes I[15:0] as an input and perform 16-bit shift left operation on it.
- The 32-bit output of shift unit can be given as a third input to MemToReg mux.
- The selector bits of MemToReg mux should be increased to two bits.

o `00 for ALU result, 01 for DM result, 10 for shift result.`