CSE 315: Computer Organization

Sheet 7

1. In the basic single-cycle implementation, different instructions utilize different hardware blocks. For each of the following instructions, what are the values of control signals generated by the control in Figure 1? Which resources (blocks) perform a useful function for each instruction? Which resources (blocks) produce outputs, but these outputs are not used for the instruction? Which resources produce no outputs for the instruction?

   a. add Rd, Rs, Rt
   b. lw Rt, Offs(Rs)

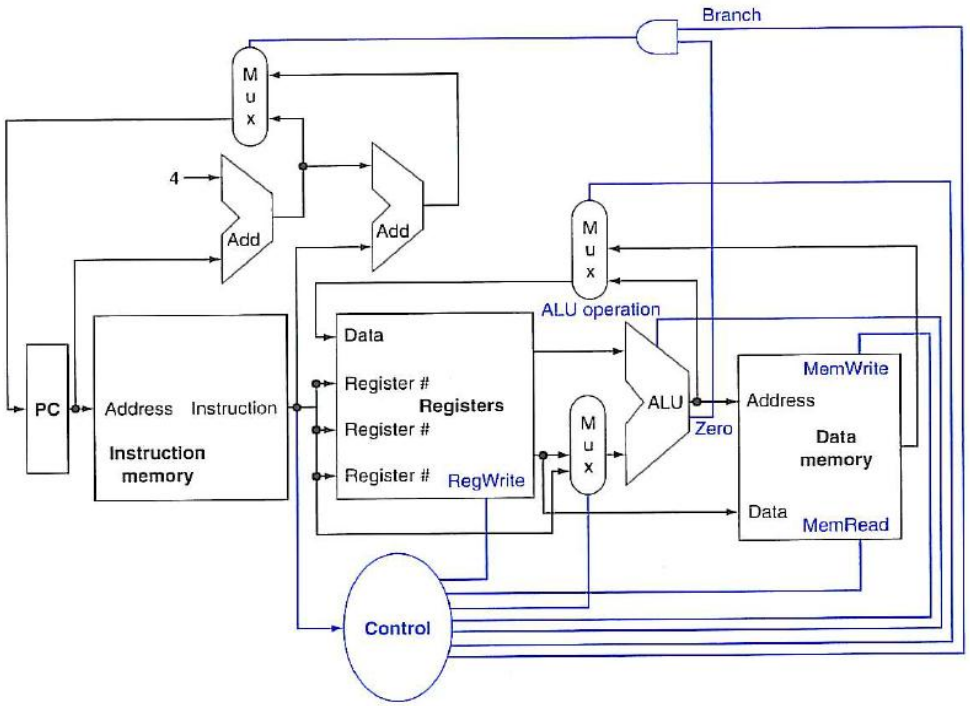


Figure 1

Solution:

The values of the control signals are as follows:

	RegWrite	MemRead	ALUMux	MemWrite	ALUOp	RegMux	Branch
a.	1	0	0(Reg)	0	Add	1(ALU)	0
b.	1	1	1(Imm)	0	Add	1(Mem)	0

ALUMux is the control signal that controls the Mux at the ALU input, 0 (Reg) selects the output of the register file and 1 (Imm) selects the immediate from the instruction word as the second input to the ALU.
RegMux is the control signal that controls the Mux at the Data input to the register file, 0 (ALU) selects the output of the ALU and 1 (Mem) selects the output of memory.
A value of X is a "don't care" (does not matter if the signal is 0 or 1).

Resources performing a useful function for the instruction are:

a. All except Data Memory and branch Add unit
b. All except Add unit and second read port of the Registers

The Resources are as follows:

	Outputs that are not used	**No outputs**
a.	Branch Add	Data Memory
b.	Branch Add, second read port of Registers	None (all units produce outputs)

2. For the following logic blocks' latencies, if the only thing we need to do in a processor is fetch consecutive instructions (Figure 2), what would the cycle time be? Consider a datapath similar to the one in Figure 3, but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath? What if conditional PC-relative branch is the only instruction supported?

	I-Mem	**Add**	**Mux**	**Alu**	**Regs**	**D-Mem**	**Sign-extend**	**Shift-left-2**
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	2ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

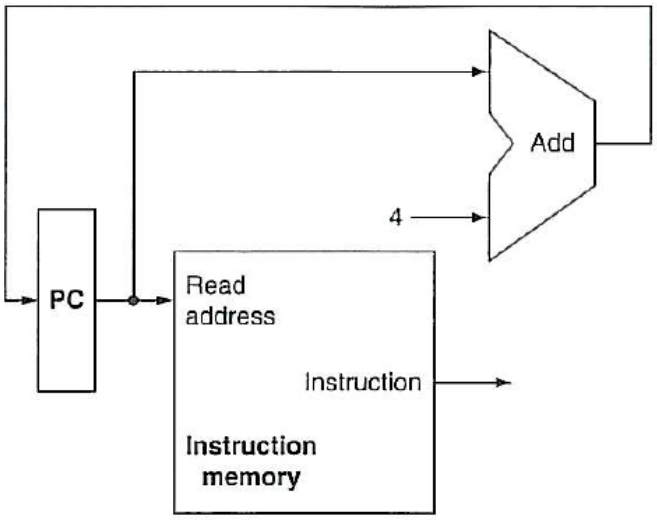


Figure 2

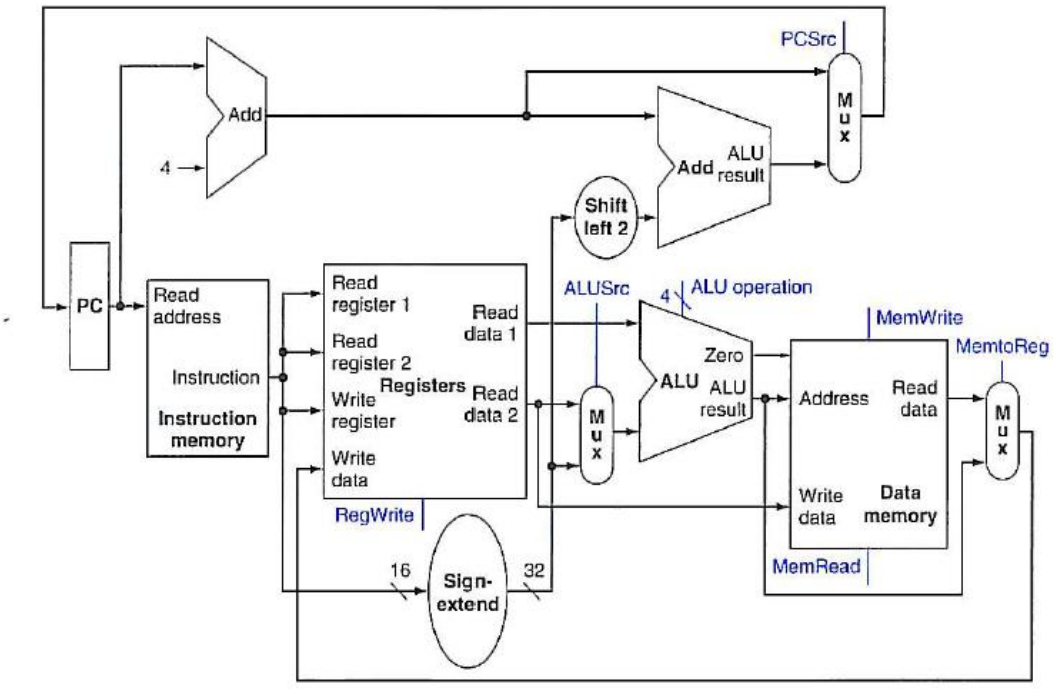


Figure 3

Solution:

I-Mem takes longer than the Add unit, so the clock cycle time is equal to the latency of the I-Mem:

a. 400ps
b. 500ps

The path for the unconditional PC-relative instruction is through the instruction memory, Sign-extend and Shift-left-2 to get the offset, Add unit to compute the new PC, and Mux to select that value instead of PC+4. Thus, the cycle time will be as follows:

a. 400ps+20ps+2ps+100ps+30ps=552ps
b. 500ps+90ps+20ps+150ps+100ps=860ps

Conditional branches have the same long-latency path that computes the branch address as unconditional branches do. Additionally, they have a long-latency path that goes through Registers, Mux, and ALU to compute the PCSrc condition. The critical path is the longer of the two, and the path through PCSrc is longer for these latencies:

a. 400ps+200ps+30ps+120ps+30ps=780ps
b. 500ps+220ps+100ps+180ps+100ps=1100ps

3. Which kinds of instructions require each of the following resources? Assuming that we only support *bne* and *add* instructions, discuss how changes in the latencies of this resource affect the cycle time of the processor. Assume that the latencies of other resources do not change.

    a. Add 4 adder (The adder adding 4 to the PC)
    b. Data Memory

Solution:

The instructions that require the above resources:

a. All instructions except jumps that are not PC-relative (jal, jalr, j, jr).
b. Loads and stores.

Of the two instructions (*bne* and *add*), *bne* has a longer critical path so it determines the clock cycle time. Note that every path for *add* is shorter or equal to the corresponding path for *bne*, so changes in unit latency will not affect this. As a result, we focus on how the unit's latency affects the critical path of *bne*:

a. This unit is not on the critical path, so changes to its latency do not affect the clock cycle time unless the latency of the unit becomes so large to create a new critical path through this unit, the branch add, and the PC Mux.
b. This unit is not used by BNE nor by ADD, so it cannot affect the critical path for either instruction.

4. For the following single-cycle datapath MIPS instructions, using MIPS reference card, using registers numbers as written explicitly in the instruction assembly form, what is the value of the instruction word? What is the register number supplied to the register file's "Read register 1" input? Is this register actually read? How about "Read register 2"? How about "Write register"?

	Instruction
a.	lw $1, 40($6)
b.	Label: bne $1, $2, Label

Solution:

The value of the instruction word:

	Binary	**Hexadecimal**
a.	100011 00110 00001 0000000000101000	8CC10028
b.	000101 00001 00010 1111111111111111	1422FFFF

For "read register 1" and "read register 2", the registers numbers are as follows:

	Read register 1	**Actually read?**	**Read register 2**	**Actually read?**
a.	6 (00110)	Yes	1 (00001)	Yes (but not used)

b.	1 (00001)	Yes	2 (00010)	Yes

For “write register”:

	Write register	Register actually written?
a.	1 (00001)	Yes
b.	Either 2 (00010) or 31 (11111) (RegDst is X)	No

5. What is the value of the following control signals for the jump instruction (Figure 4)?
   a. RegDst
   b. MemRead
   c. RegWrite
   d. Jump
   e. Branch

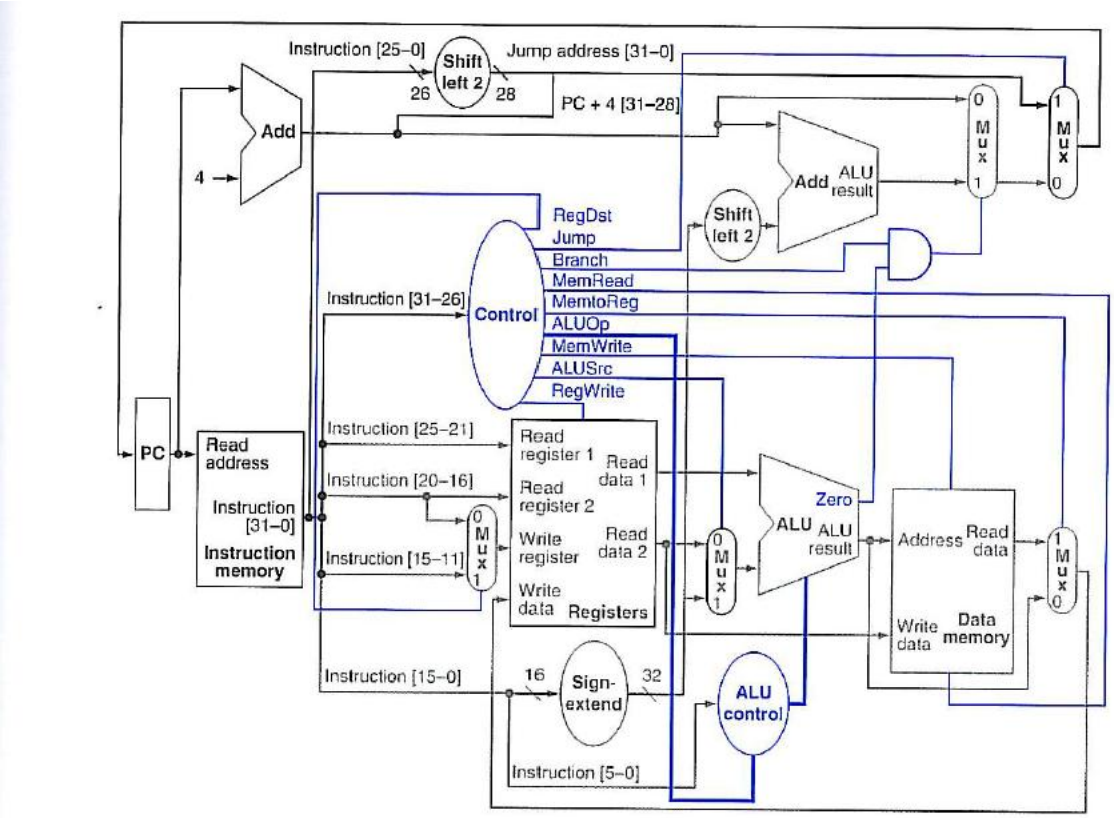


Figure 4

Solution:

The values of the control signals are:

a. RegDst = X

b. MemRead = X
c. RegWrite = 0
d. Jump = 1
e. Branch = X

6. The latencies of individual components of the datapath, shown in Figure 5, affect the clock cycle time of the entire datapath. Thus, for the following latencies, what is the clock cycle time if the only type of instructions we need to support are ALU instructions (add, and, etc…)? What is the clock cycle time if we only had to support lw instructions? What is the clock cycle time if we must support add, beq, lw, and sw instructions? If we can improve the latency of one of the given datapath components by 10%, which component should it be? What is the speed-up from this improvement?

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

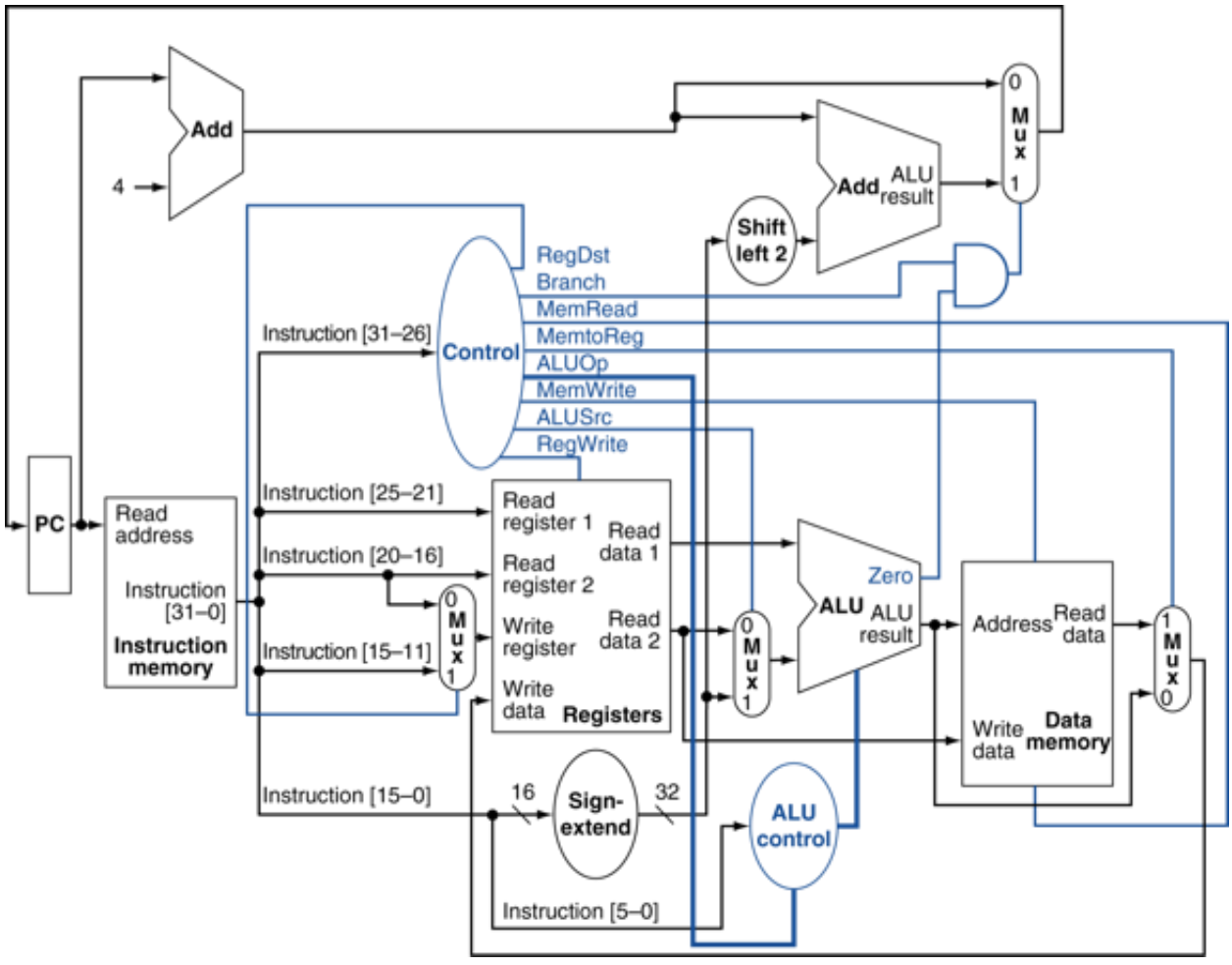


Figure 5

The longest-latency path for ALU operations is through I-Mem, Regs, Mux (to select ALU operand), ALU, and Mux (to select value for register write). Note that the only other path of interest is the PC-increment path through Add (PC+4) and Mux, which is much shorter. So for the I-Mem, Regs, Mux, ALU, Mux path the latencies are:

a. 400ps+200ps+30ps+120ps+30ps=780ps
b. 500ps+220ps+100ps+180ps+100ps=1100ps

The longest-latency path for lw is through I-Mem, Regs, Mux (to select ALU input), ALU, D-Dem, and Mux (to select what is written to register). The only other interesting paths are the PC-increment path (which is much shorter) and the path through Sign-extend unit in address computation instead of through Registers. However, Regs has a longer latency than Sign-extend, so for I-Mem, Regs, Mux, ALU, D-Mem, and Mux path, the latencies are:

a. 400ps+200ps+30ps+120ps+350ps+30ps=1130ps
b. 500ps+220ps+100ps+180ps+1000ps+100ps=2100ps

In case of add, beq, lw, and sw instructions, the answer is the same as the above instructions, because the lw instruction has the longest critical path. The longest path for sw is shorter by one Mux latency (no write to register), and the longest path for add or bne is shorter by one D-Mem latency.

The clock cycle time is determined by the critical path for the instruction that has the longest critical path. This is the lw instruction, and its critical path goes through I-Mem, Regs, Mux, ALU, D-Mem, and Mux.

a. I-Mem has the longest latency, so we reduce its latency from the 400ps to 360ps, making the clock cycle 40ps shorter. The speed-up achieved by reducing the clock cycle time is 1130ps/1090ps=1.037
b. D-Mem has the longest latency, so we reduce its latency from 1000ps to 900ps, making the clock cycle 100ps shorter. The speed-up achieved by reducing the clock cycle time is then 2100ps/2000ps=1.050

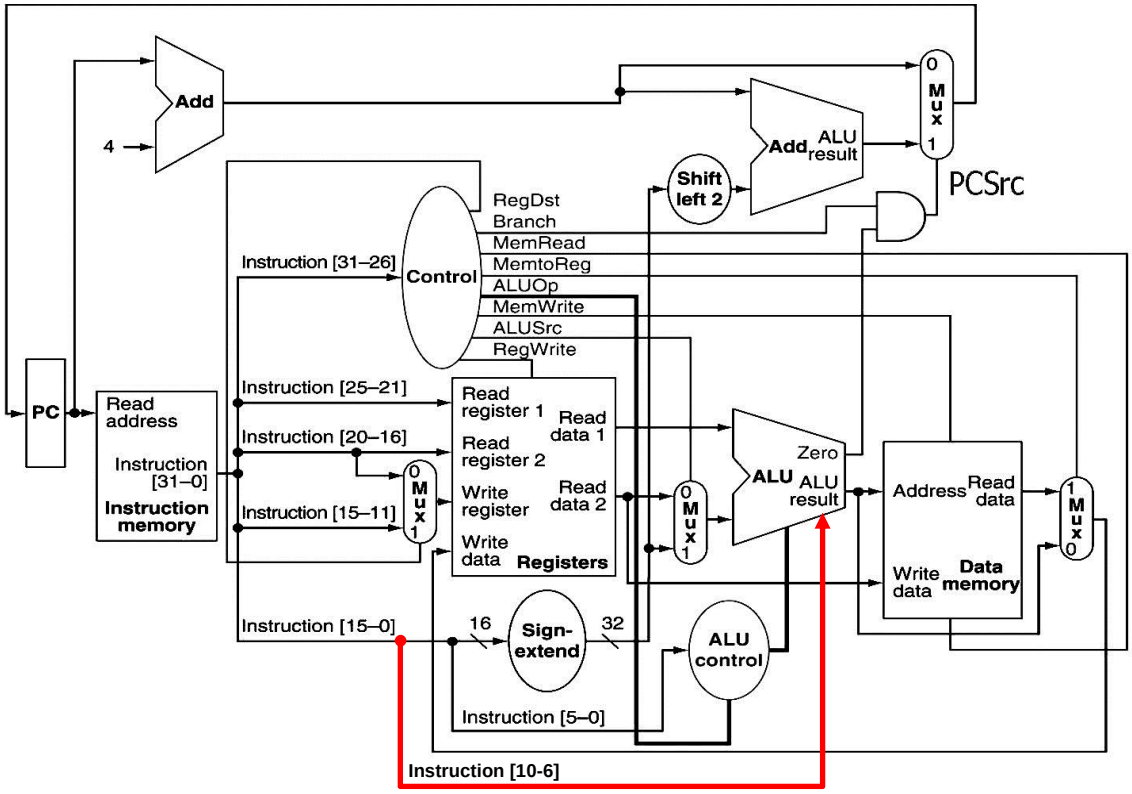
7. Consider the single-cycle datapath in Figure 5, for the instruction listed in table below, show the values of various control signals needed to execute instruction. Note that ALU function is ADD when ALUOp is 00, Subtract when ALUOp is 01 and determined by Funct part of instruction when ALUOp id 10 for R-type instructions.

Solution:

Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp
sw	X	1	X	0	0	1	0	00 (Add)
beq	X	0	X	0	0	0	1	01 (Subtract)

8. We wish to add the instruction sll (shift left logical) to the single-cycle datapath shown in Figure 5. Add any necessary datapaths and control signals. Show any necessary addition for the table of control signals similar to that in Exercise 7, if needed.

Solution:



ALUOp	Funct	ALU Control	Control action
10	00 0000	1110 (shift left logical)	Shift the second ALU operand ($rt) by the amount in the shmat field (Instruction [10-6]), input to the ALU

9. Repeat the above exercise for the instruction lui (load upper immediate).

Solution:

No modification to the datapath is required.

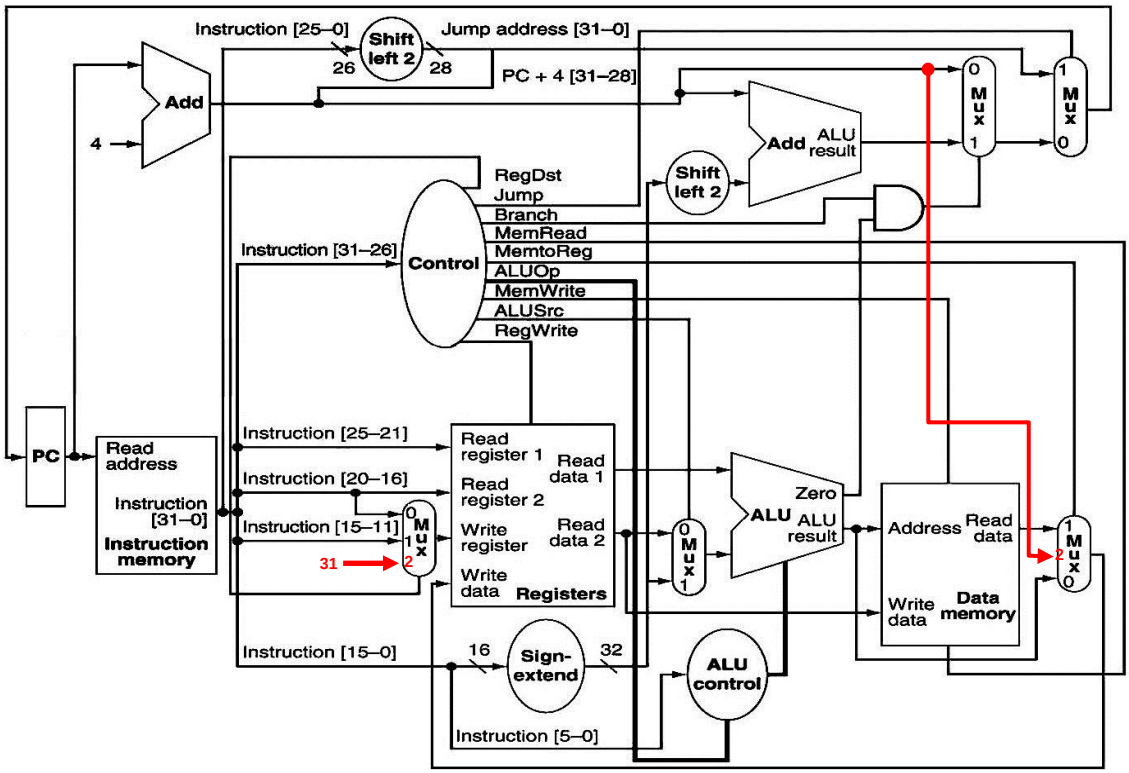
We can add a new operation to the ALU.

ALUOp	Funct	ALU Control	Control action
11	XX XXXX	1101 (shift left logical 16 bits)	Shift the second ALU operand (imm) 16 bits

Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp
lui	0	1	0	1	0	0	0	11

10. Repeat the above exercise for the instruction jal (jump and link). You may find it easier to modify the datapath in Figure 4 above.

Solution:



We already have a way to change the PC based on the specified address (using the datapath for the j instruction), but we will need a way to put PC + 4 into register $ra (31), and this will require changing the datapath.

We can expand the multiplexor controlled by RegDst to include 31 as a new input. We can also expand the multiplexor controlled by MemToReg to have PC + 4 as an input. This requires changing the control lines of these two multiplexors from a single bit to two bits. The Jump control signal needs to be set to 1 to operate as the j instruction. The register file will be enabled to write.

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	ALUOp	Branch	Jump
j	XX	X	XX	0	0	XX	X	1
jal	10	X	10	1	0	XX	X	1

11. Assuming the latencies shown in the table below for single data path in Figure 5, what will be the latency of each of the instructions: Add, Addi, lw, sw and beq.
    What is the maximum clock speed? Can we replace the adders by cheaper (slower) ones without affecting the clock speed of the processor? In case your answer is no, explain why this is the case. In case your answer is yes, compute the maximum latency for the adder that does not affect the clock speed?

I-Mem	Adder	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
600ps	100ps	50ps	150ps	185ps	500ps	10ps	5ps

Solution:

add latency = 600 + 185 + 50 + 150 + 50 = 1035 ps
addi latency = 600 + 185 + 150 + 50 = 985 ps
lw latency = 600 + 185 + 150 + 500 + 50 = 1485 ps
sw latency = 600 + 185 + 150 + 500 = 1435 ps
beq latency = 600 + 185 + 150 = 935 ps
The maximum clock speed is a $1/1485 \times 10^{-12}$ = 673.4 MHz

There are two adders: one to compute the PC+4 and the other one to compute the branch address. Both are not on the critical path of the instructions and can be replaced by slower adders. Note that the second adder exists on 2 non-critical paths. If we call the adder latency X, we have the following constraints
Path1 latency: $2X + 50 \leq 1485$ implying that $X \leq 717.5$ ps
Path2 latency: $600 + 10 + 5 + X + 50 \leq 1485$ implying that $X \leq 820$ ps
Therefore the maximum adder latency of the adders that will not affect the processor speed is 717.5 ps.

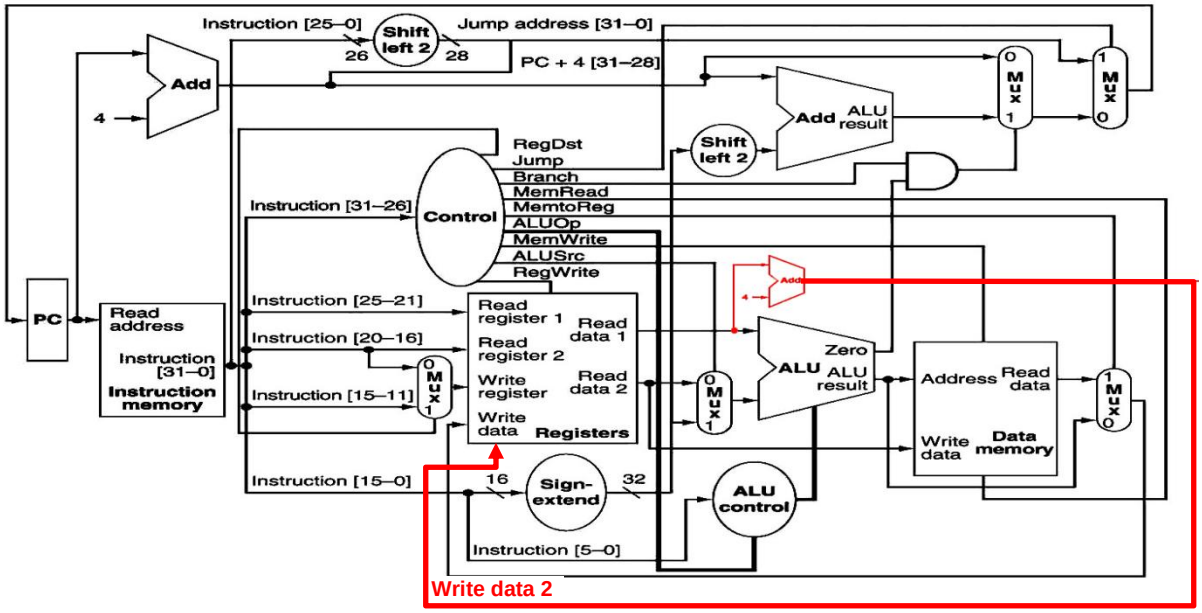
12. We wish to add a variant of the lw (load word) instruction to the single-cycle datapath shown in Figure 5. The instruction increments the index register after loading word from memory. This instruction (l_inc) corresponds to the following two instructions:

```
lw      $rt, L($rs)

addi    $rs, $rs, 4
```

    Show the values of various control signal needed to implement the instruction. Is it possible to modify the single-cycle implementation to implement this instruction without modifying the register file?. Explain why or why not.

Solution:



The following modifications are required for the datapath:

- Perform autoincrement by adding 4 to register $rs through an incrementer (add + 4). This operation is be done all the time.
- We need a second write port to the register file because two register writes are required for this instruction. The RegWrite control signal will now be extended to two bits as shown in the table below. A new “Write data 2” port will be added. We assume that the “Write register 2” identifier is always the same as “Read register 1” ($rs). This way the setting “11” to control signal RegWrite indicates that there is second write to register file to the register identified by "Read register 1" and the data is fed through “Write data 2”.

Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp
l_inc	0	1	1	11	1	0	0	00

This instruction requires two writes to the register file. The only way to implement it is to modify the register file to have two write ports instead of one.