CS161 - Homework 2
DUE Monday, May 6, 2019 IN CLASS!

1. The basic single-cycle MIPS implementation, shown in Figure 1 on page 4, can only implement some instructions.
New instructions can be added to an existing Instruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed addition introduces into the processor datapath and control. The first three problems in this exercise refer to the new instruction:

Instruction: `LWI Rt, Rd(Rs)`

Interpretation: `Reg[Rt] = Mem[Reg[Rd] + Reg[Rs]]`

Assume that this instruction is represented as an R-type instruction (unlike LW).

a. Which existing blocks (if any) can be used for this instruction? Blocks refer to any data path component, such as registers, ALUs, muxes, etc.

b. Which new functional blocks (if any) do we need for this instruction? If you need new functional blocks, please add them to the exiting data path in Figure 1.

c. What new signals do we need (if any) from the control unit to support this instruction? If you need new signals, please extend the control table in Figure 1.

2. Problems in this exercise assume that logic blocks needed to implement a processor's datapath have the following latencies:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

a. If the only thing we need to do in a processor is fetch consecutive instructions, what would the cycle time be? *Hint: Identify the critical path.*

b. Now let's design a datapath similar to Figure 1, but for a processor that only has one type of instruction: unconditional PC-relative branch. (Essentially, a branch instruction that is always true. Unlike Jump which uses absolute addressing, branches use PC-relative addressing.) Assume your new datapath is streamlined and only include the logic blocks required to support this instruction only. (Blocks not used can be eliminated.) What would the cycle time be if you designed this datapath?

c. Repeat part b, but this time we need to support only conditional PC-relative branches (i.e. your normal branch instruction).

3. For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

a. In what fraction of all cycles is the data memory used?

b. In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?

4. 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
```

```
which is SW r2, 20(r3)
```

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

a. In Figure 2 on page 5, what are the outputs of the sign-extend and the jump "Shift left 2" unit (the top one) for this instruction word?

b. What are the values of the ALU control unit's inputs for this instruction? (What is ALUOp[1-0] and Instruction[5-0]?)

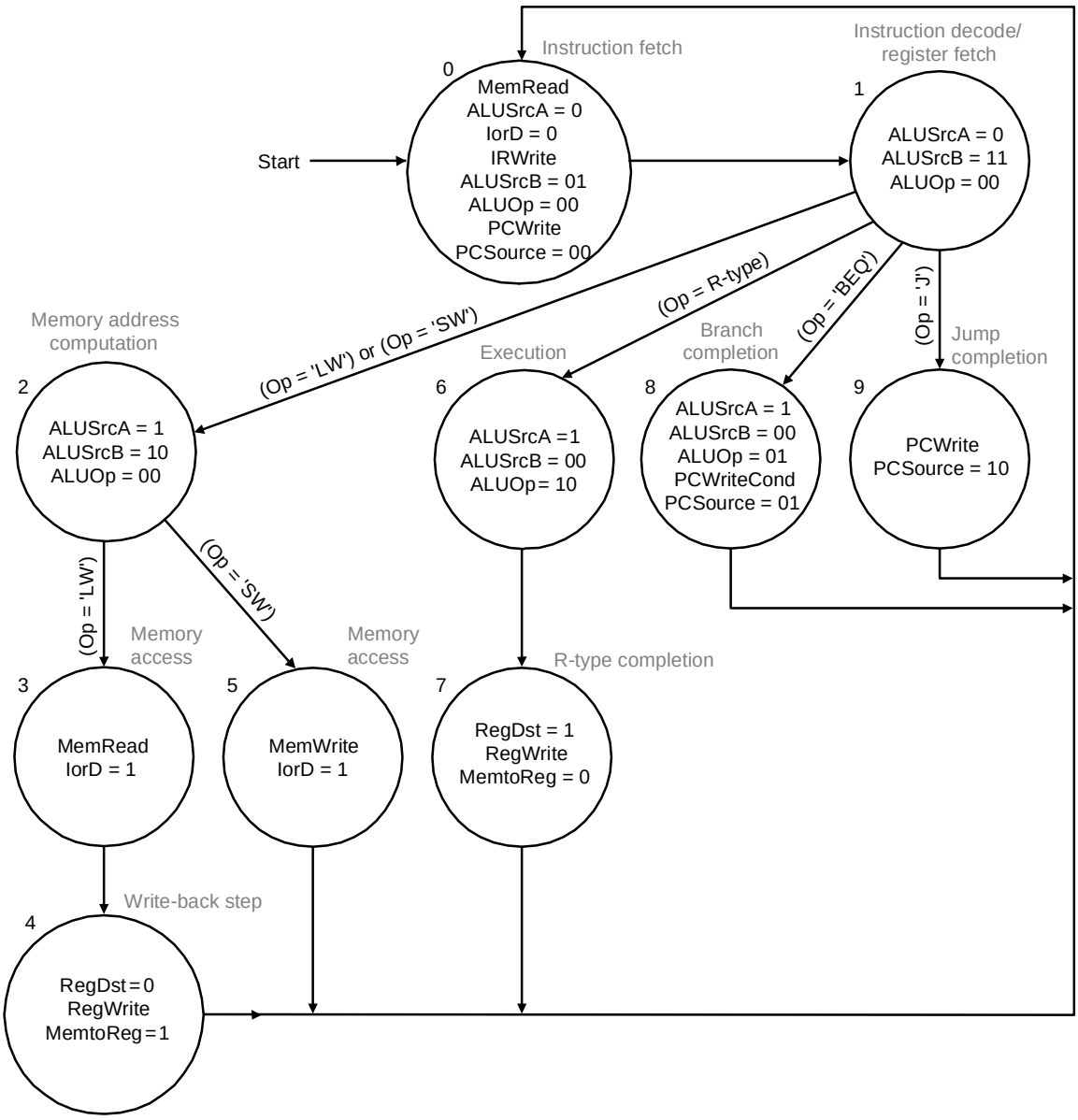
c. What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

d. For each Mux, show the values of its data output during the execution of this instruction and these register values.

e. For the ALU and the two add units, what are their data input values?

f. What are the values of all inputs for the "Registers" unit?

5. Add an addi instruction the multi-cycle: Now we wish to add the instruction addi (add immediate). Add any necessary changes to the datapath and to the control signals. Show the modification required to the state machine below to support addi.


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

R-Type	LW	SW	BEQ	J	Jump	Branch	Reg Dst	ALU Src	Memto-Reg	Reg Write	Mem Read	Mem Write	ALU Op[1]	ALU Op[0]
1	0	0	0	0	0	0	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	1	1	1	1	0	0	0
0	0	1	0	0	0	0	X	1	X	0	0	1	0	0
0	0	0	1	0	0	1	X	0	X	0	0	0	0	1
0	0	0	0	1	1	0	X	X	X	0	0	0	X	X

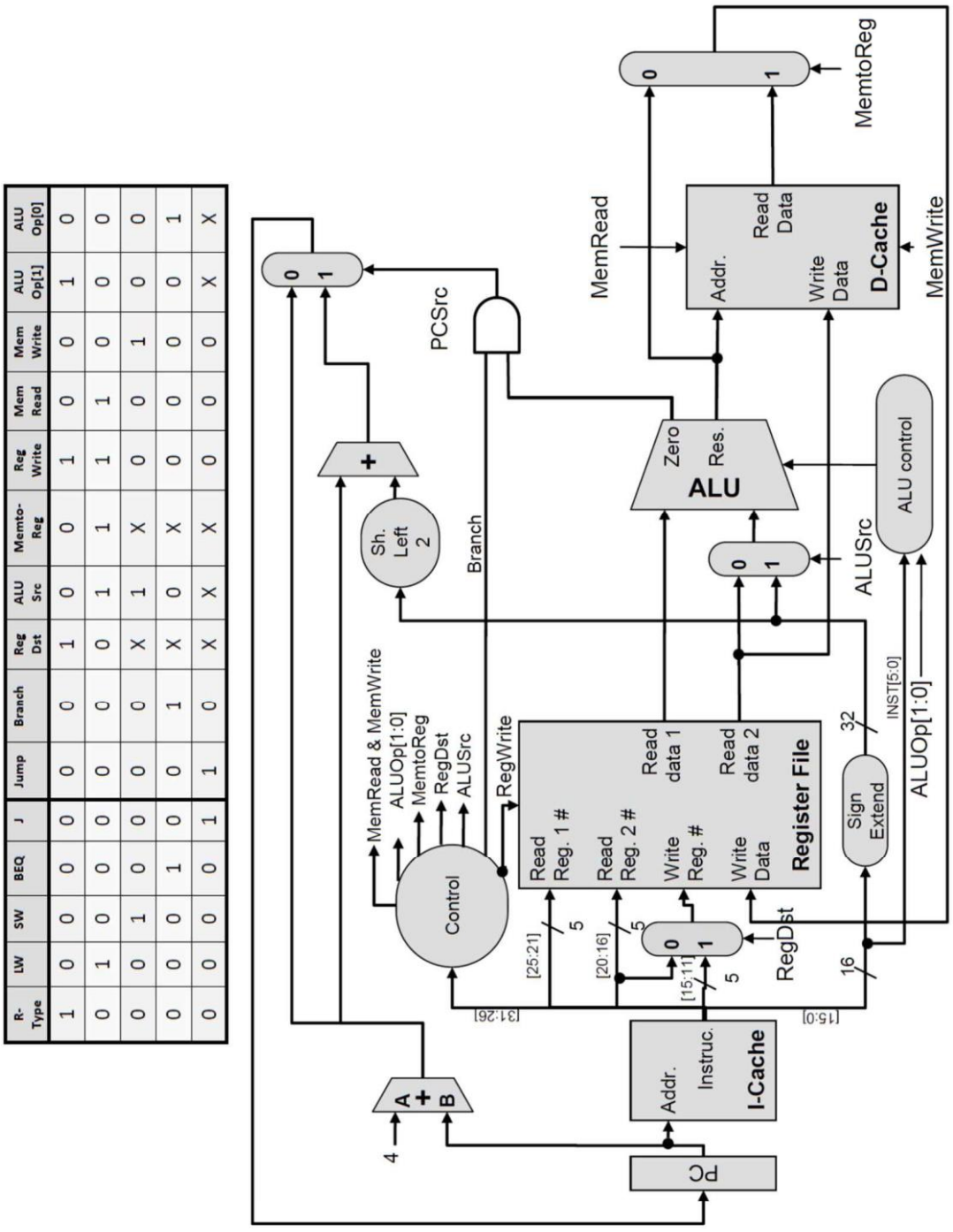


Figure 1: Single Cycle CPU

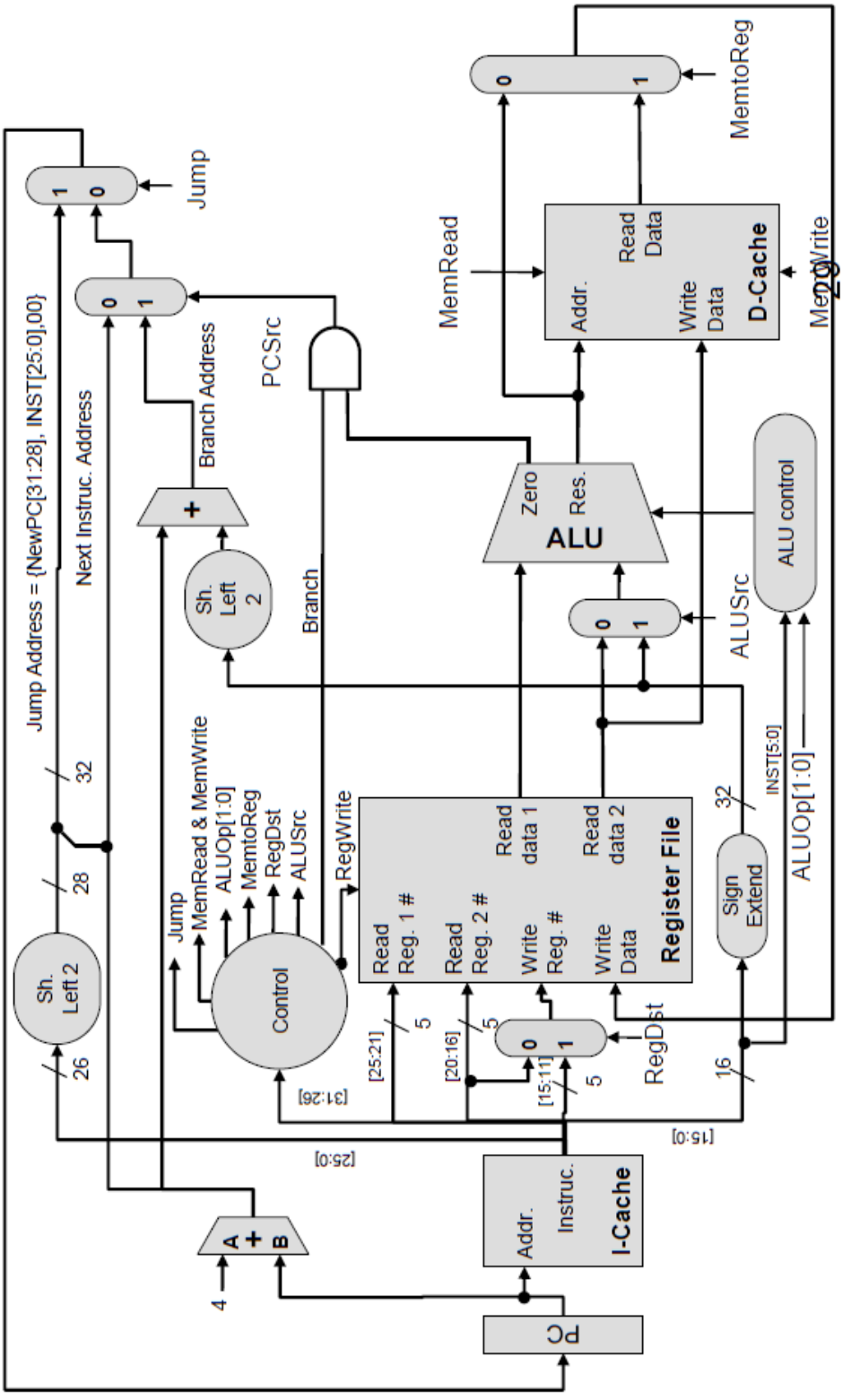


Figure 2: Single Cycle with Jump Support