LAB 5: Support Material: CPU

Part I. Load and Store Word

Description

In the previous lab, we have the ALU instructions to do the simple calculation. In this lab, we will add the **load and store instructions** (I-type) to achieve loading data from the data memory and storing data to the data memory.

I-type format

Opcode	Rs	Rt	Signed Immediate Value
6-bit	5-bit	5-bit	16-bit

Instruction	Opcode	Operation	PC
ADDI	001000	Rt = Rs + immd	PC = PC + 4
SET	000001	Rt = immd	PC = PC + 4
LW	100011	Rt = mem[immd]	PC = PC + 4
SW	101011	mem[immd] = Rt	PC= PC + 4

Instruction & Date memory

The reasons why we need more space is registers are fast and convenient, but we have only 32 of them, it’s not enough to hold data structures like large arrays. Therefore, we need to add some main memory such as RAM to the system. We adapt the Harvard architecture in this simple 3-stage pipelined CPU; it means that programs and data will be stored in separate memories.

1. Instruction memory
   - Contains instructions to execute
   - It is read-only
2. Data memory
   - Contains the data of the program
   - Can be read and written

Loading and storing words

In this lab, we have to add load and store instructions for accessing memory. There are some examples as follows

1. Load word instruction (**LW**) transfers one word of data from the data memory to a register.

```
LW R3, immd
```

```
R3 = mem[immd]
memory address = immd value
```

2. Store word instruction (**SW**) transfers one word of data from a register into the data memory.

```
SW R3, immd
```

```
mem[immd] = R3
memory address = immd value
```

Module description

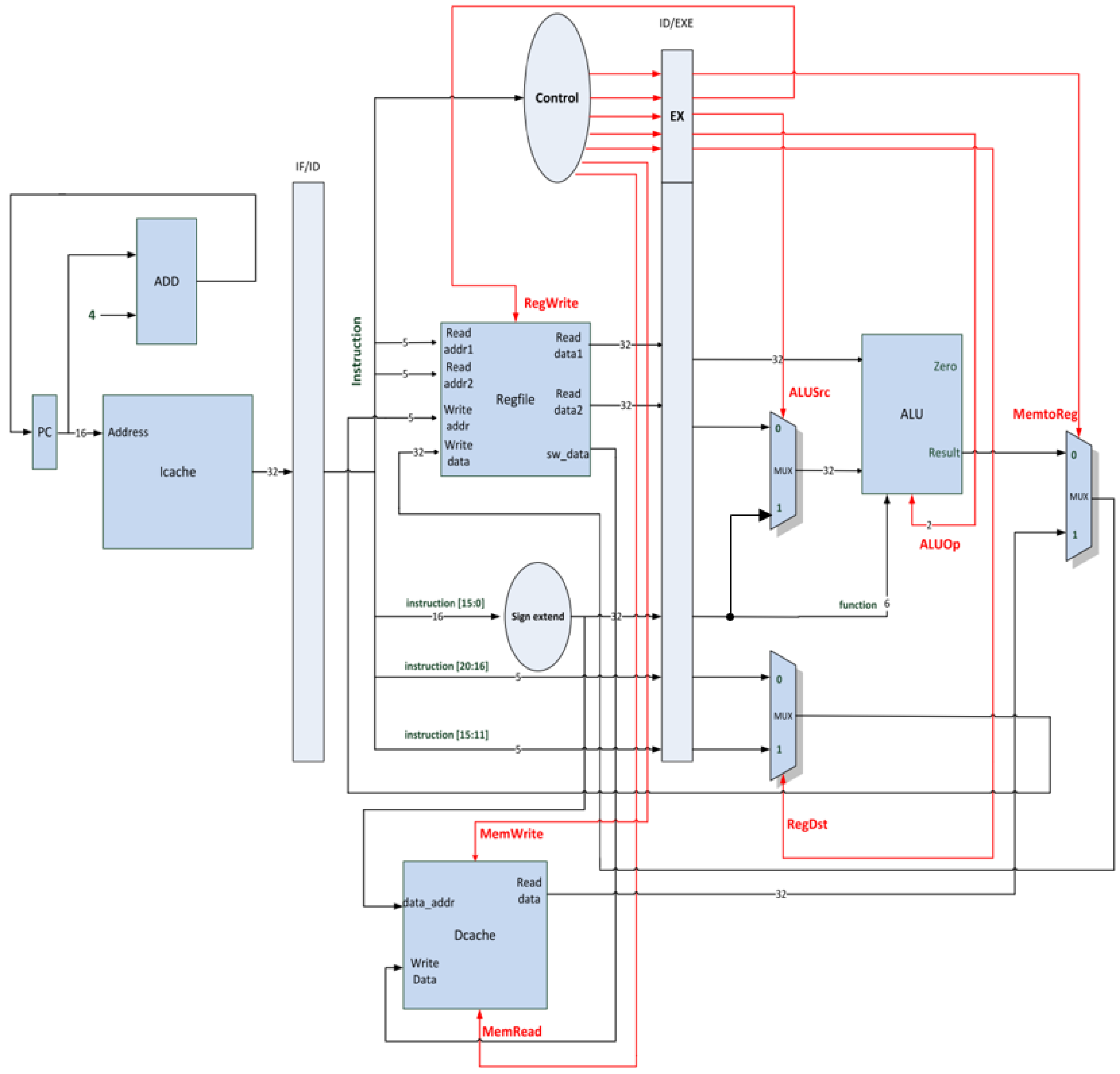
1. Program counter Send the program counter (PC) to instruction memory, update the PC to the next sequential PC by adding 4 (since each instruction is 4 bytes) to the PC.

2. Controller

Control Signals

Instruction	Opcode	RegWrite	RegDst	ALUSrc	ALUOp	MemRead	MemWrite	MemtoReg
R-type	000000	1	1	0	10	0	0	0
ADDI	001000	1	0	1	00	0	0	0
SET	000001	1	0	1	00	0	0	0
LW	100011	1	0	1	10	1	0	1
SW	101011	0	0	1	00	0	1	0

Figure 1 Block Diagram of the System Datapath



Part II. Branch

Description

Control hazards can cause a greater performance loss for our CPU than do data hazards. When a **branch** is executed, it may change the PC to something other than its current value plus 4. Recall that if a branch changes the PC to its target address, it is a **taken** branch; if it falls through, it is a **not taken**. Figure.2 shows that the simplest method of dealing with branches is to redo the fetch of the instruction following a branch, once we detect the branch during ID (when instructions are decoded). The first IF cycle is essentially a stall because it never performs useful work.

Figure 2 A Branch Causes a One-Cycle Stall in the Three-Stage Pipeline

	Clock t	**Clock t + 1**	**Clock t + 2**	**Clock t + 3**	**Clock t + 4**	**Clock t + 5**
Branch instruction	IF	ID	EXE			
Branch successor + 1		IF	IF	ID	EXE	
Branch successor + 2				IF	ID	EXE

Because the **BEQ** instruction will cause the **control hazard**, please write a **FSM** (Fig. 4) for the controller's control signal to prevent this problem. The BEQ instruction description and control signals are shown in Table 1 and Table 2.

BEQ Instruction

Instruction	Opcode	Operation
BEQ	000100	If Rs = Rt, then PC = PC + 4 + 4*offset else PC = PC + 4

Control Signals

Instruction	Opcode	RegWrite	RegDst	ALUSrc	ALUOp	MemRead	MemWrite	MemtoReg	Branch
R-type	000000	1	1	0	10	0	0	0	0
ADDI	001000	1	0	1	00	0	0	0	0
SET	000001	1	0	1	00	0	0	0	0
LW	100011	1	0	1	10	1	0	1	0
SW	101011	0	0	1	00	0	1	0	0
BEQ	000100	0	0	0	01	0	0	0	1

Figure.3 Block Diagram of Pipelined Three-Staged CPU

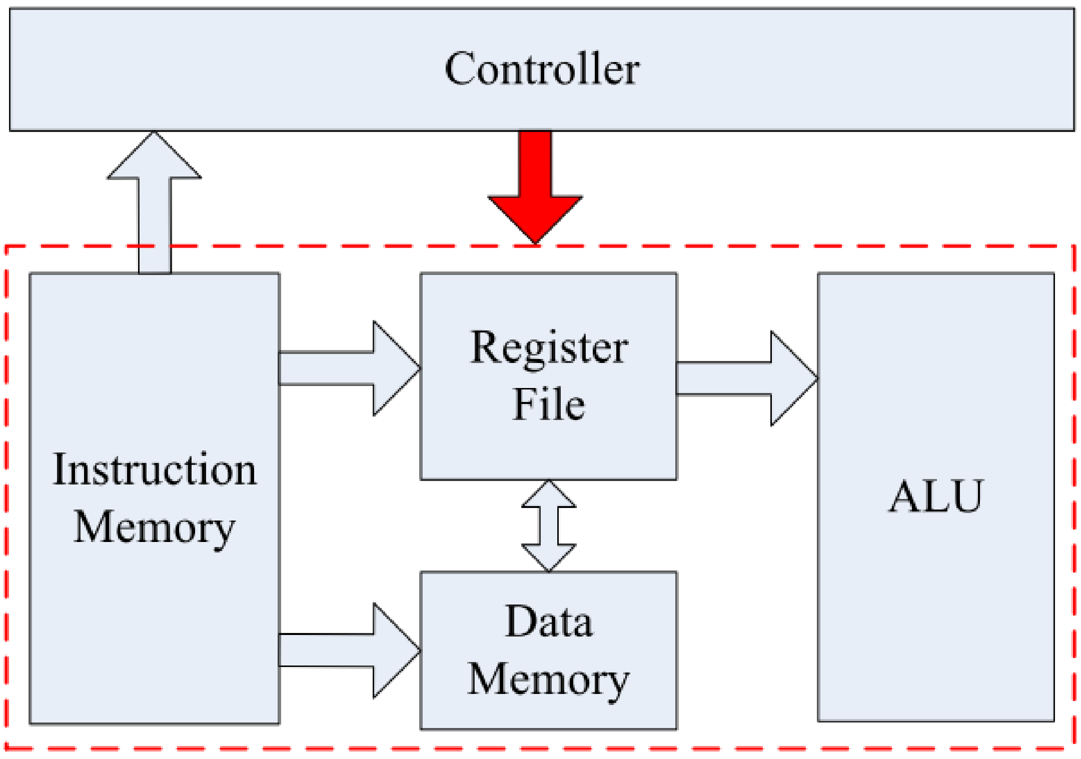
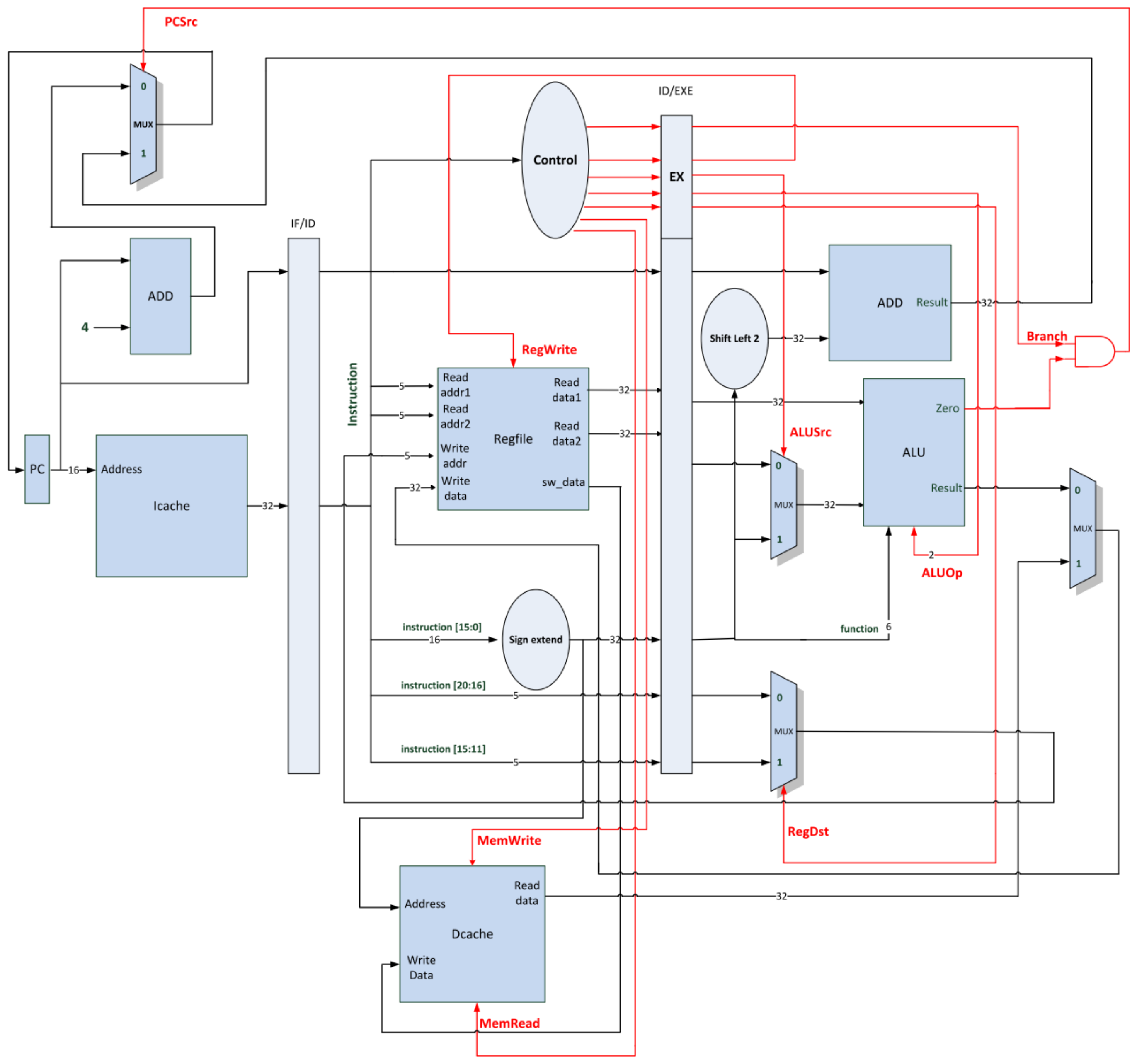


Figure.4 Block Diagram of the System Datapath and Control Signals


PCSrc
MUX
0
1
ADD
4
PC
16
Address
Icache
32
IF/ID
Instruction
Control
ID/EXE
EX
RegWrite
Read addr1
Read addr2
Write addr
Write data
Regfile
Read data1
Read data2
sw_data
5
instruction [15:0]
Sign extend
instruction [20:16]
instruction [15:11]
Shift Left 2
ADD
Result
Branch
ALUSrc
ALU
Zero
Result
ALUOp
function
6
2
RegDst
MemWrite
Address
Read data
Dcache
Write Data
MemRead