

## Datapath Design

### Computer Science 240

#### Laboratory 7

The goal of today's lab is to begin implementation of a datapath for a simple, single-cycle MIPS processor. The lab machine will be very similar to the one you have begun to study in lecture. The only difference between the two will be in the size of the address and data buses (to make the circuit you build in lab somewhat more manageable to work with).

You have already studied the circuits that are major components of the datapath. In Labs 4 - 6, you saw how the ALU, registers, a register file, and instruction memory operate. You also designed some necessary datapath subcircuits, including a sign extend, shift left, 2x16 multiplexer, and a 16-bit adder.

In fact, the CPU consists of these circuits connected together with the proper logic to execute one of the 9 instructions from the simple MIPS instruction set you have been introduced to. So, if you understand these circuits, you can understand how a computer works!

#### Storing and Accessing Instructions

*Exercise 1:* Recall from last lab that the **Instruction Memory** contains instructions to be executed by the processor, and is loaded with these instructions by the programmer prior to execution of the program.

To execute the program, the processor must access memory to fetch an instruction. The program counter, or **PC**, is a register that contains the address in memory of the current instruction.

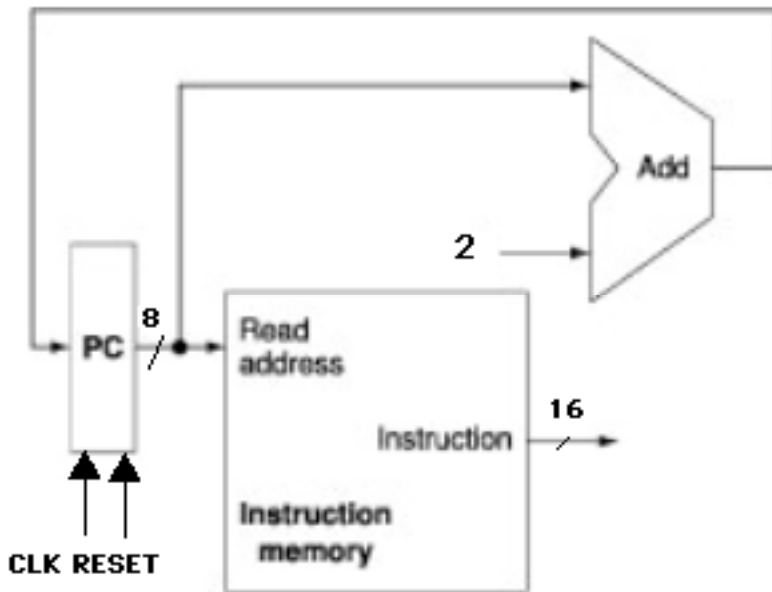
The PC is connected to the address inputs of the memory device, and the instruction stored at that address is output on the data output lines of the memory device (and is then fed into the processor for execution).

The PC must be updated after an instruction is fetched, with the address of the next instruction stored in memory. Often, this will simply be  $PC + 2$ , since the default is that instructions execute in sequential order.

To create this circuit, you will need 16-bit register to hold the current value of the PC, an adder to add 2 to the current value, and an instruction memory such as the

one you used in lab 6. You can use the instruction memory and adder from last week's lab, along with a 16-bit register that is available in the Lab Materials folder.

Use these components (available in the Lab Materials folder) to create the following circuit:



Implement the circuit in LogicWorks. Display the address and data outputs (instruction) from the Instruction memory using hex displays.

To simulate operation of the circuit, connect binary switches to the CLK and RESET lines of the PC (RESET connects to the CLR input of the register). Begin by activating the RESET line, which will initialize the PC to address 0.

Step through the first 6 addresses, and record the instruction stored at each address by activating the CLK input to update the PC:

<u>Address</u>	<u>Instruction</u>
0000	

Do these values look familiar to you? Demonstrate to the instructor. Save your circuit.

## Branch Address

*Exercise 2:* The address of the next instruction to fetch is not always  $PC + 2$ . If the Branch-if-equal (BEQ) instruction is executed, a new address specified by the instruction must be fed into the PC. Therefore, it is necessary to calculate this new address when a BEQ is executed.

The format for the BEQ instruction is:

BEQ  $R_s$   $R_t$  offset      if  $R_s = R_t$ , then  $PC = PC + 2 + 2 * \text{offset}$

Since offset specifies the number of words away from the next value of the PC that you wish to branch to, the offset must be multiplied by 2.

Remember that offset is a 4-bit value (bits 3..0 of the instruction). To perform the calculation of the branch address, you must sign extend the offset to 16 bits. Then, to multiply by 2, shift left by 1 position.

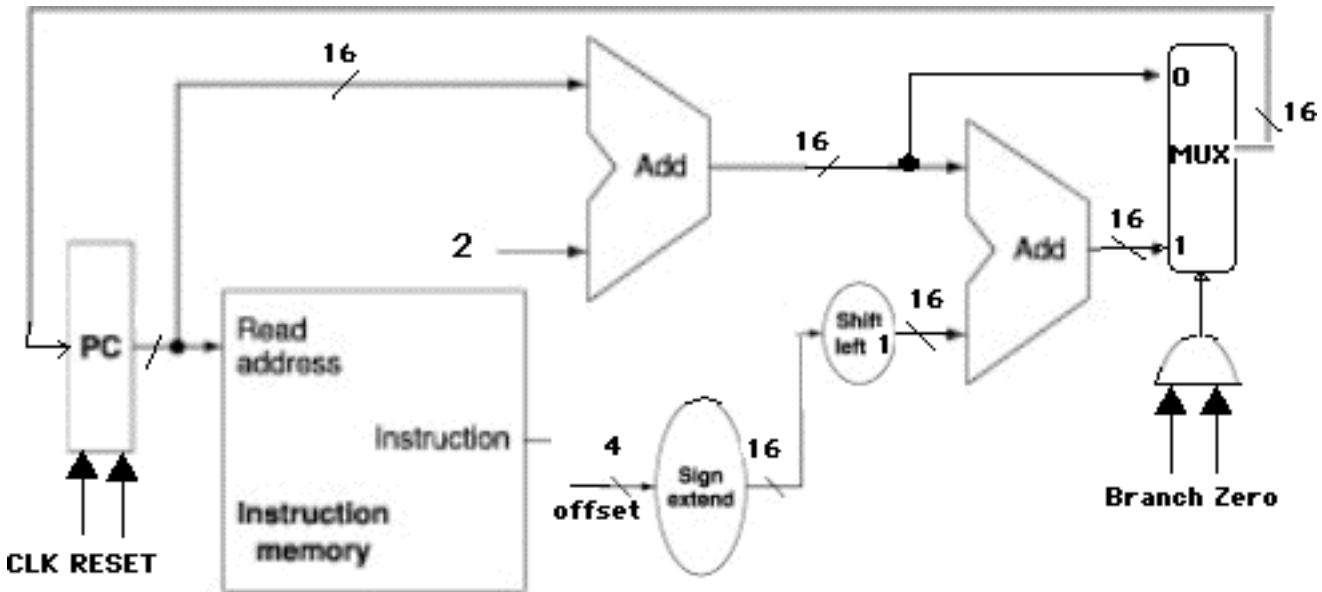
A 16-bit adder can be used to calculate  $PC + 2 + 2 * \text{offset}$ .

A multiplexer can be used to select whether  $PC + 2$  or  $PC + 2 + 2 * \text{offset}$  will be the next value of the PC.

Assume that a control line labeled **Branch** will be set to 1 if a BEQ instruction is being executed. It is also necessary to check whether  $R_s = R_t$ ; the **Zero** bit from the ALU (which performs  $R_s - R_t$  for this instruction) will be set to 1 if they are equal.

Therefore, if both  $\text{Branch} = 1$  and  $\text{Zero} = 1$ , then the calculated branch address will be the next value of the PC. Otherwise, it will simply be  $PC + 2$ .

Add a sign-extend, shift-left, 16-bit adder, and a 2x16 multiplexer (all designed in last week's lab, and available in the Lab Materials folder) to your circuit from exercise 1 to calculate the branch address:



In addition to the CLK and RESET inputs, you now also need to simulate a 4-bit offset with a hex keyboard, and Branch and Zero bits with binary switches.

Begin by activating the RESET line, which will initialize the PC to address 0.

Set the offset = 3, Branch = 1, Zero = 1.

What do you expect the next value of the PC to be? Show your calculation below:

Activate the CLK input, and verify that the PC is what you expect.

Set Zero = 0. What are you simulating when you do this?

Activate the CLK input. What is the next value of the PC? Why?

Set Branch = 0, Zero = 1, and activate the CLK input. What is the next value of the PC? Why?

Set Branch = 1 and Zero = 1. Set the offset to the following values, and record the PC after you activate the CLK for each offset:

offset      PC

C  
5  
2  
9

Do the results make sense? Demonstrate to the instructor. Save your circuit.

### Register File and ALU Operation

*Exercise 3:* You have learned in lecture that there are several instructions called R-type instructions (ADD,SUB,AND,OR,SLT). The general format of these instructions is:

Inst    Rs    Rt    Rd

For these instructions, the processor will:

- read Rs and Rt from the register file
- perform an ALU operation on the contents of the registers
- write the result to register Rd in the register file

There are also memory access instructions , load word and store word (LW and SW), which use the register file and ALU. The general form of these 2 instructions is:

Inst.    Rs    Rt    offset

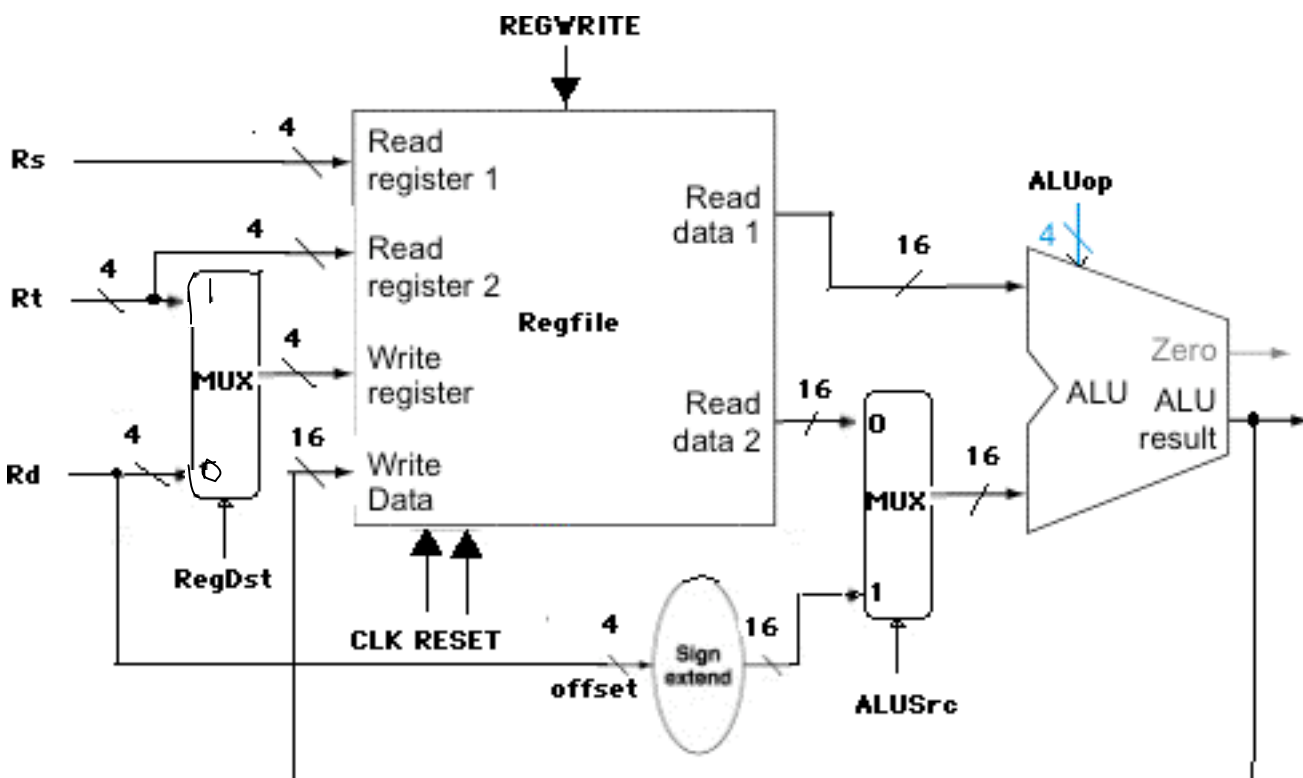
- Compute a memory address by adding Rs to the sign extended 4-bit offset
- For a store, the value to be stored is read from Rt.
- For a load, the value read from memory is written into Rt.

When an R-type instruction is executed, the destination register is specified by Rd. However, when a memory access instruction is executed, the destination register is specified by Rt. Therefore, a multiplexer is needed to choose the source for destination register, which is then fed into the Write input of the Regfile.

When an R-type instruction is executed, the Read data 1 and Read data 2 outputs of the register file select the contents of Rs and Rt, and feed the values into the A and B sides of the ALU to perform the desired operation. However, when a memory access instruction is executed, the ALU needs to add Rs to the sign extended offset to calculate a memory address.

So, the ALU must be able to calculate either  $R_s + R_t$ , or  $R_s + \text{sign-extended offset}$ .  $R_s$  always feeds in to side A of the ALU, but a multiplexer is needed to select whether the B side comes from  $R_t$  or from the offset.

The diagram below shows the required components to accomplish these operations. Find the register file and the ALU in the Lab Materials folder on FirstClass. Use the 2x16 multiplexer and the Sign Extend circuit from exercise 1. A 2x4 multiplexer is also required, which you can find in the LogicWorks library.



Display Read data 1 and Read data 2 from the Regfile, and the ALU result using hex displays.

To simulate operation of the circuit, connect binary switches to the CLK, RESET, and REGWRITE lines of the register file, and to the RegDst and ALUSrc select lines of the two multiplexers. Use hex keyboards for Rs, Rt, Rd, and ALUop.

Begin by activating the RESET line, which will initialize the registers to 0 (except register 1, which is initialized to a value of 1).

The following table describes which ALU function will be produced by a given value of ALUop:

<u>ALUop</u>	<u>ALU function</u>
0	a AND b
1	a OR b
2	a + b (add)
6	a - b
7	set on less than

Set the following values on the input devices. For each test, activate the CLK input to perform the operation, and record the ALU result:

**Test 1:**

<u>ALUop</u>	<u>Rs</u>	<u>Rt</u>	<u>Rd</u>	<u>RegWrite</u>	<u>RegDst</u>	<u>ALUSrc</u>	<u>ALU result</u>
2	1	1	6	1	0	0	

Explain the operation and result:

**Test 2:**

<u>ALUop</u>	<u>Rs</u>	<u>Rt</u>	<u>Rd</u>	<u>Regwrite</u>	<u>RegDst</u>	<u>ALUSrc</u>	<u>ALU result</u>
2	6	6	3	1	0	0	

Explain the operation and result:

**Test 3:**

<u>ALUop</u>	<u>Rs</u>	<u>Rt</u>	<u>Rd</u>	<u>RegWrite</u>	<u>RegDst</u>	<u>ALUSrc</u>	<u>ALU result</u>
2	6	1	7	0	1	1	

Explain the operation and result:

Assume this test simulates a Store Word (SW) instruction. What value would be stored to what address?

**Test 4:**

<u>ALUop</u>	<u>Rs</u>	<u>Rt</u>	<u>Rd</u>	<u>RegWrite</u>	<u>RegDst</u>	<u>ALUSrc</u>	<u>ALU result</u>
2	3	6	5	1	1	1	

Explain the operation and result:

Assume this test simulates a Load Word (LW) instruction. What address is being accessed?

What register will receive the value stored at that address?

Demonstrate to the instructor. Save your file.