Mini-MIPS DataPath
Computer Science 240

Laboratory 9

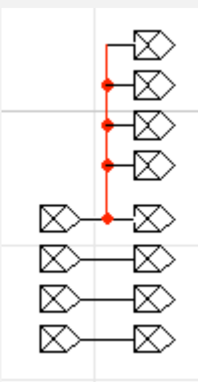
In this lab, you will begin constructing a datapath for the mini-MIPS CPU. The lab machine is very similar to the single-cycle MIPS architecture you are studying in lecture: the main difference between the two will be in the size of the address and data buses (to make the circuit you build in lab somewhat more manageable to work with).

You have already experimented with some of the circuits that are major components of the datapath, including multiplexers, ALU, registers, register file, multiplexers, and an instruction memory. Today, you will design some additional datapath subcircuits, including a sign-extend and shift-left circuit.

You will then begin to construct a datapath for the mini-MIPS CPU, which consists of these circuits connected together with the proper logic to execute the 9 instructions from the simple MIPS instruction set you have been introduced to.

Sign-Extend

Exercise 1: Design a new circuit in LogicWorks which will take a 4-bit value and sign-extend it to 8 bits. Paste the circuit below:



Test the circuit by simulating inputs and outputs.

Abstract your design into what in LogicWorks is called a subcircuit. Here are the steps:

1. For each of your inputs, add a **Port In** connector (from the Connect library of the Parts Palette). For each of your outputs, add a **Port Out** connector. Label the connectors (remember that the label must be attached to the port).

2. Select **New** in the **File** menu, and choose the **Device Symbol** option. Click **OK**.

3. From the **Options** menu, select the **Subcircuit/Part Type** command, and choose the **"Create a subcircuit syjmbol and select an open circuit to attach to it.."** option.

4. Select the sign extend circuit which you have open, and click **OK**. Close the **PartType** configuration dialog by clicking on **DONE.** You will notice that the symbol editor has extracted the names from the port connectors in the subcircuit, and placed them in the Pin List at the left side of the main window.

5. Create a graphic for the subcircuit by selecting the **Autocreate symbol** command from the **Options** menu. To assign every pin listed in the Pin List to a corresponding graphical pin on the device symbol, click on

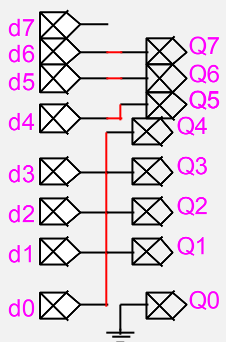
Extract Pin List. Under **Part Name**, type in **sign extend**. Click on **Generate Symbol** to create the new sign extend symbol.

6. From the menu bar, select **File**, then **Save As** from the pulldown menu. When the **Save Part As** window opens, click on the **New Lib** button. When the **Save Library As** window open, type **yourname.clf**, and save it in the **LIBS** folder of **LogicWorks** by clicking on **Save** (if you get a dialog box that asks you if you want to overwrite an already existing library, click **Yes**). Back in the **Save Library As** window, select the library **yourname.clf**, and click on **Save** to store the sign extend subcircuit in the library.

Close the device symbol editor, and open a new design window. Select your sign extend circuit from your library, and place it in your design window. Test your subcircuit by connecting input and output devices, and again verifying the correct operation of the device.

Shift-Left

Exercise 2: Design a new circuit which will take an 8-bit value and shift it left by 1 bit. Paste the circuit below:



Test the circuit by simulating inputs and outputs. Abstract your design into a subcircuit, as you did in exercise 1.

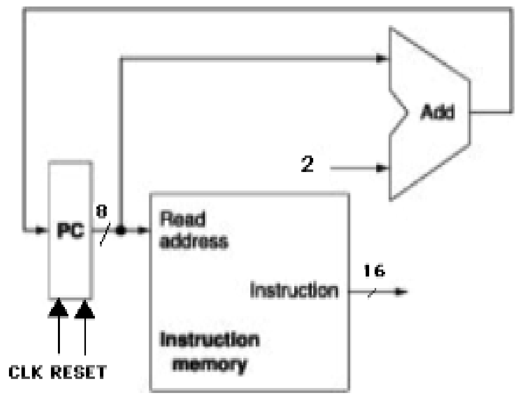
Open a new design window. Select your shift left circuit from your library, and place it in your design window. Test your subcircuit by connecting input and output devices, and again verifying the correct operation of the device.

Instruction Fetch

Exercise 3: To execute a program, the CPU must fetch each instruction from the instruction memory. The program counter, or **PC**, is a register in the CPU that contains the address of the instruction currently being executed.

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 instructions execute in sequential order. All this can be implemented as shown below:



Add an 8-bit register (**Reg-8** in LogicWorks) and an 8-bit adder to your Instruction Memory circuit from last week (you can find this in *parts.cct* from the Google group).

To simulate operation of the circuit, connect binary switches to the **CLK** and **RESET** lines of the **PC** register (**RESET** should connect to the **CLR** input of the register).

Begin by activating the **RESET** (set to 1 and back to 0) line, which will initialize the PC to address 0.

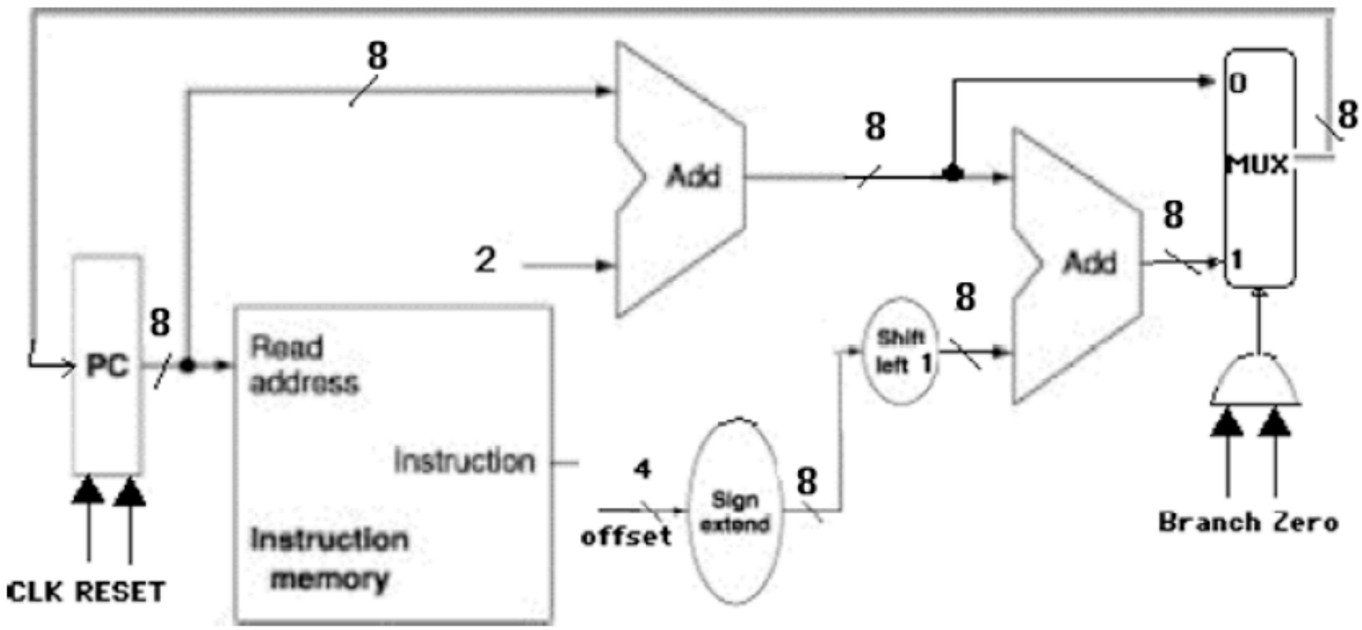
Step through the first 6 addresses in memory, and record the instruction stored at each address by activating the **CLK** input to update the **PC** for each sequential instruction stored in memory:

Address	Instruction	
0010	0x5003	OR R0,R0,R3
0100	0x1220	SW R2,0(R2)
0110	0x0230	LW R3,0(R2)
1000	0x2122	ADD R1,R2,R2
1010	0x8001	JMP 1100

You should recognize the values you just recorded: where have you seen them before?
These are the values we saw in the memory circuit from lab 8, and represent a small mini-MIPS program.

Branch Address

Exercise 4: The address of the next instruction to fetch is not always **PC + 2**, as we designed in the previous exercise. When 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 following circuit will be used for this purpose:



How does this work? The format for the **BEQ** instruction is:

BEQ Rs Rt offset # if Rs = Rt, then PC = PC + 2 + 2*offset

Since **offse**t 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.

The offset is a 4-bit value (bits 3..0 of the instruction), so to perform the calculation of the branch address, you must **sign-extend** the offset to 8 bits. Then, to multiply by 2, **shift left** by 1.

An **8-bit adder** can be used to calculate PC + 2 + 2*offset.

A **8x2 MUX (**from *pars.cct)* is then used to select whether **PC + 2** or **PC + 2 + 2*offset** will be the next value of the **PC**. The MUX selects the branch address is both **Branch** and **Zero** are true: **Branch** = 1 if a **BEQ** instruction is being executed, and **Zero** = 1 if the comparison is met for the **BEQ** instruction.

To construct the circuit:

1. Add the new components to your circuit from the last exercise.

2. Simulate **Branch** and **Zero** with binary switches (since the real signals are produced in the part of the datapath which you have not yet constructed).

3. Also, simulate the 4-bit **offset** with a hex keyboard (the **offset** will come from the instruction being executed when the datapath is complete, but you must simulate it for now).

To test the circuit:

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

2. Set the offset = 3, Branch = 1, Zero = 1.
   What do you expect the next value of the PC to be? Show your calculation below:

PC = PC + 2 + (2 * offset) = 0 + 2 +3*2=2+6=8

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

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

That the registers being compared in the BEQ instruction do not contain equal values.

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

   0A – the BEQ failed so we advance to the next instruction.

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

0C – the BEQ failed so we advance to the next instruction

7. Set Branch = 1 and Zero = 1. Assume **PC** = C to begin, and **do not** clear PC after each test. Set the offset to the following values:
   Record the PC after you activate the CLK for each offset. Show your calculations to verify that **PC** is the correct value.

offset	PC	Calculations
C	06	C+2+2*C=C+2-8=6
5	12	6+2+2*5=18
2	18	0x12+2+2*2=24
9	0C	0x18+2+2*-7=12

Demonstrate to the instructor. Save your circuit.

Register File and ALU Operation

Exercise 5: The R-Type instructions are ADD, SUB, AND, OR, and 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

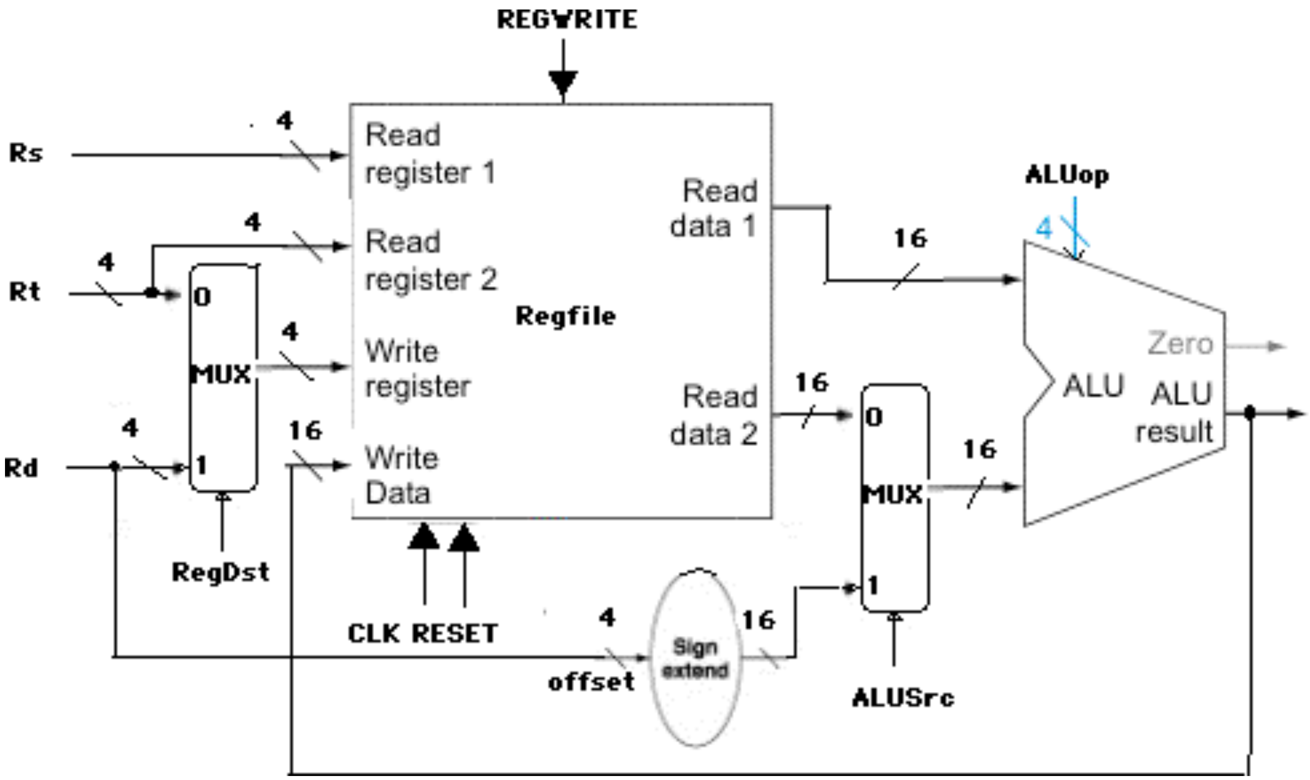
- Compute a memory address by adding **Rs** to the sign extended 4-bit offset
- For a store, the value to be stored is read from **Rt**
- For a load, the value read from memory is written into **Rt**

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 **Rs + Rt**, or **Rs + sign-extended offset**. **Rs** always feeds in to side **A** of the **ALU**, but a multiplexer is needed to select whether the **B** side comes from **Rt** or from the **offset**.

The diagram below shows the required components to accomplish these operations. Find the **register file** and the **ALU** in *parts.cct* circuit - you will also find a **16x2 MUX** and **sign-extend** circuit there. A **2x4 MUX** is also required, but 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). **This RESET is active low**, so set to 0 and back to 1 to perform a reset.

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

ALUop	ALU function
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:

ALUop	Rs	Rt	Rd	RegWrite	RegDst	ALUSrc	ALU result
2	1	1	6	1	0	0	2

Explain the operation and result:
ALU result = R1+R1 saved in R6

Test 2:

ALUop	Rs	Rt	Rd	RegWrite	RegDst	ALUSrc	ALU result
2	6	6	3	1	0	0	4

Explain the operation and result:
R3=R6+R6 (2+2=4)

Test 3:

ALUop	Rs	Rt	Rd	RegWrite	RegDst	ALUSrc	ALU result
2	6	1	7	0	1	1	9

Explain the operation and result:
Calulate an address by adding contents of Rs (base address) + offset
R6 (contains 2) + offset (7) = 9

Assume this test simulates a Store Word (**SW**) instruction. What value would be stored to what address?
The contents of Rt is stored to the specified address

R1 (contains 1) would be stored to address 9

Test 4:

ALUop	Rs	Rt	Rd	RegWrite	RegDst	ALUSrc	ALU result
2	3	6	5	1	1	1	9

Explain the operation and result:
Calulate an address by adding contents of Rs (base address) + offset
R3 (contains 4) + offset (5) = 9

Assume this test simulates a Load Word (**LW**) instruction. What address is being accessed?
Address 9

What register will receive the value stored at that address?
R6 will get a value of 1 from address 9 as a result of this instruction

Demonstrate to the instructor. Save your file.