Experiment #10, Final Project: The RISC-Y Processor

During the semester, you have modeled and simulated the following modules:

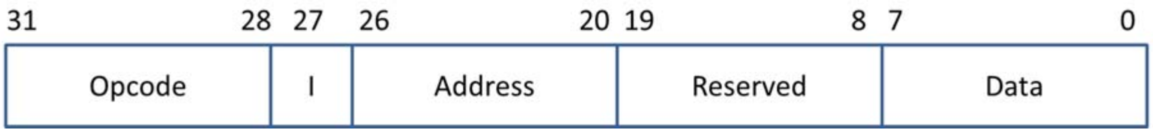
1. 8-bit Register (Labs #1-3),
2. Counter and AASD (Lab #4),
3. Scaleable MUX (Lab #5),
4. Scalable Register Files (Lab #7),
5. 8-bit ALU (Lab #8),
6. Sequence Controller and Phase Generator (Lab #9)

These modules can be put together to form the RISC-Y system as described here. However, instead of instantiating the register from Lab 3, you will model several registers of appropriate sizes behaviorally. They should be done by creating a scalable behavioral register model rather than unique designs for each register size. Because some registers need to be reset and others do not, it is preferable to create two scalable register models, one with a reset input and one without.

The RISC-Y processor is typical of modern Reduced Instruction Set Computer architectures such as the ARM family of processors used in virtually all smart phones in that it has a small, regular instruction set and a load-store architecture. Though it is far smaller and simpler than any current-production machine, it is a complete microcontroller and could be programmed to accomplish general purpose computing task.

To be physically produced and turned into something useful, the ROM would need to be replaced with a FLASH memory and a suitable program download interface added. Other than that, the RISC-Y processor represents a complete design.

Each RISC-Y instruction will use the following format:



Lab 10 Figure 1: Instruction Format

Bits [7:0] are immediate data. When the opcode is Load and the I bit (Bit 27) is set, these data are loaded into the target register indicated by the Address field (Bits [26:20]). If the I bit is not set, then the Data field contains the address of the data to be loaded into the target register. Since this implementation of the RISC-Y processor only has 64 memory words (32 RAM and 32 ROM), only the least significant bits of the Data field are used for the address pointer. These bits are loaded into the Memory Instruction Register and used for accessing the RAM for Load and Store operations other than Load Immediate.

Note that the Address field needs seven bits because the processor has memory-mapped registers and peripherals, but six bits are all that are needed for memory addressing. Each memory only needs five address bits. The other bits are used for selecting which device to enable.

The RISC-Y processor has two addressing modes: Immediate and Direct. Immediate data are held in the Data field of the instruction. Direct data has the *address* of the data in the Data field. They are distinguished by the I bit. When the I bit is set, the addressing mode is Immediate. Otherwise, it is Direct.

There are only two possible objects that may be stored: the ALU output and the port. When the opcode is STORE, the Address field is checked to see if it contains 67, which is the address of the port. If there is a match, port data are stored in the address indicated by the least significant bits of the Data field. For any other address, the ALU contents are stored.

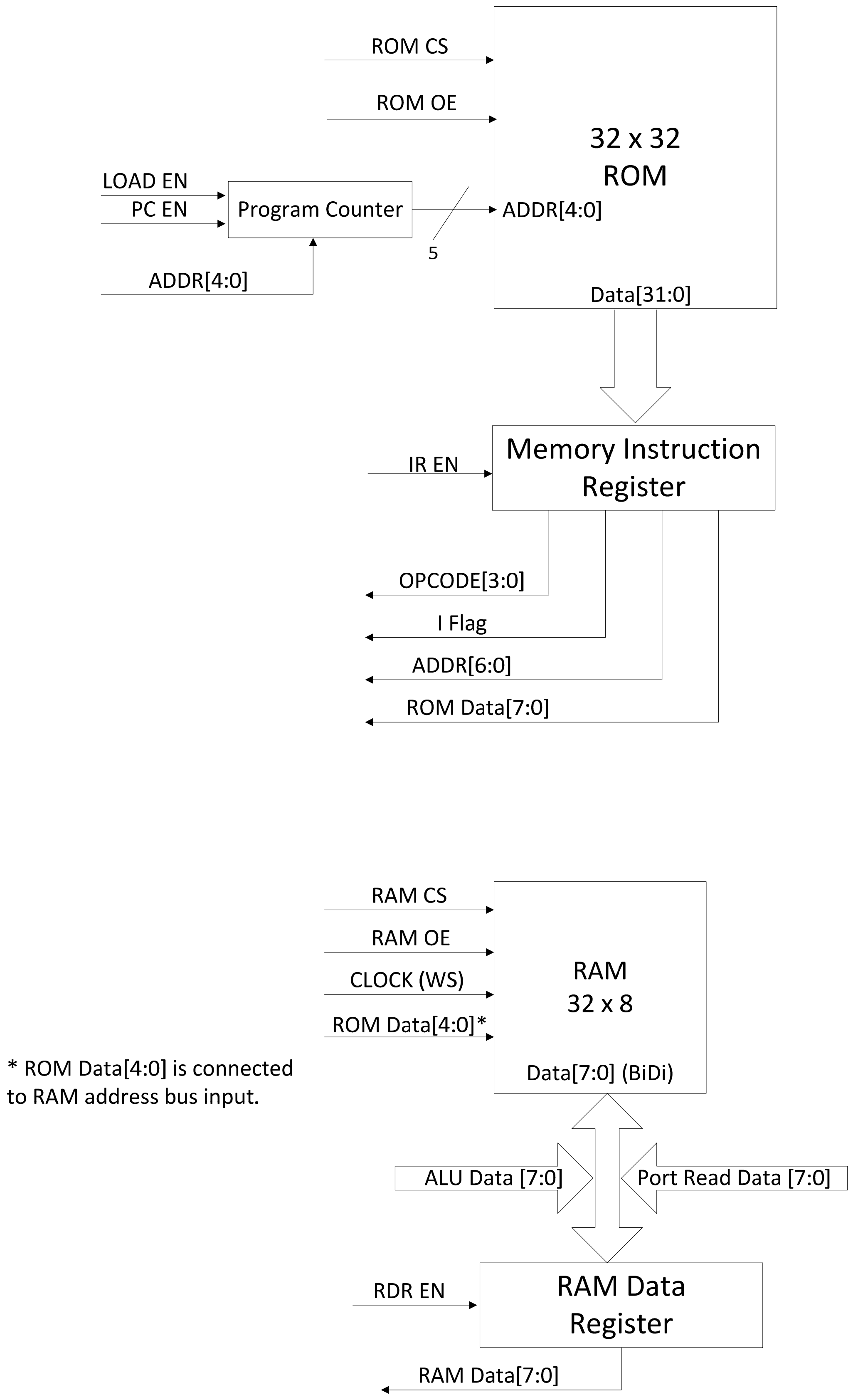
ALU opcodes (ADD, SUB, AND, OR, XOR and NOT) do not need to reference any fields other than the opcode. They always use the A and B registers for operands and the result always goes to the ALU results register.

The remaining opcodes are all branches. For these operations, the branch address will be contained in the lowest five address bits. These bits are loaded into the Program Counter. It is only possible to branch to ROM addresses.

To avoid conflict on the bidirectional memory bus, the ALU may be put into high impedance mode. When ALU data are being written to the memory, both the port and the memory outputs shall be put into high impedance state. When reading from the port or memory, the ALU output must be high impedance. When none of the three needs to be driving the bus, it is still necessary to keep one of them enabled to prevent the bus from floating. Which one is the default enabled signal does not matter.

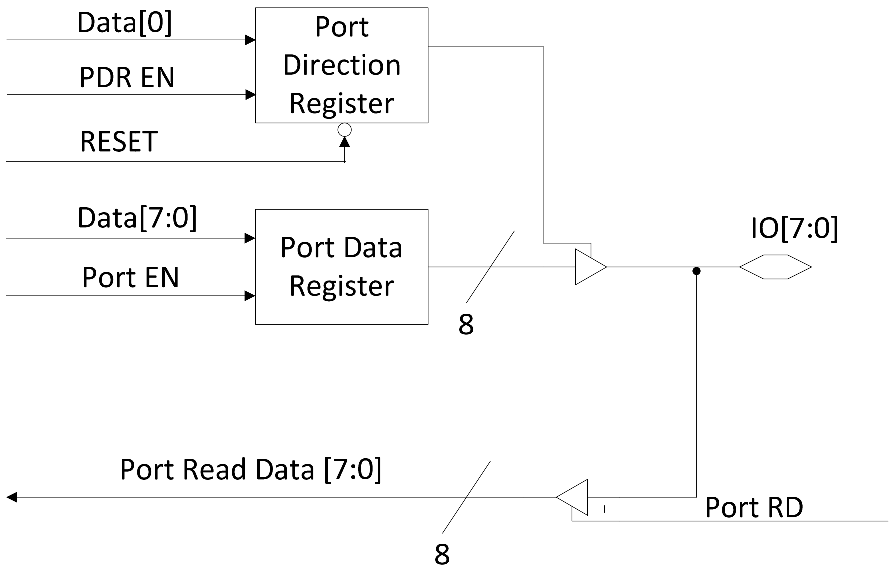
There is no provision in operation of the processor to disable the phase generator from Lab 9, so the enable signal on it may be tied to a constant logic level.

The major component groups of the RISC-Y processor are shown in Figures 2 through 5 below. Clock and reset signals are generally not shown. All registers operate on the rising edge of the master clock. Most data registers do not need to be reset, but the port direction register does. It must be initialized to input mode.

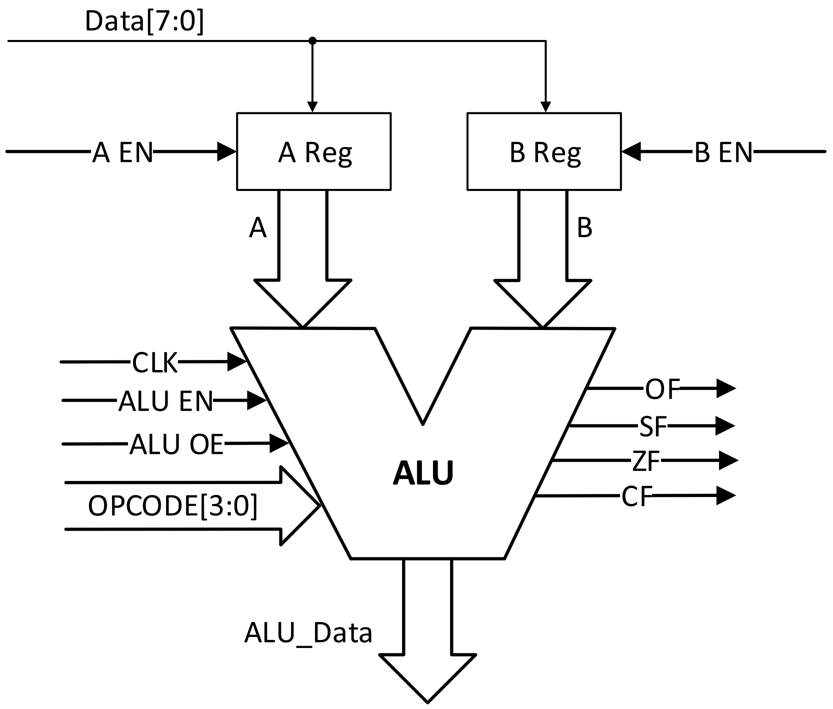


Lab 10 Figure 2: RISC-Y Memory Subsystem

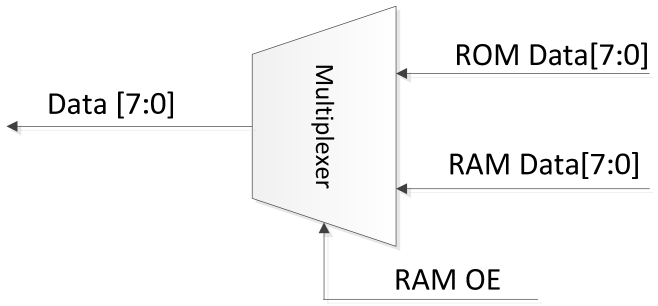
In addition to the blocks shown in these diagrams, the sequencer and its associated phase generator from Experiment 9 need to be instantiated and connected. Use the enumerated type created in Experiment 9. Do not convert the phase to plain binary.



Lab 10 Figure 3: IO Subsystem



Lab 10 Figure 4: ALU Subsystem



Lab 10 Figure 5: Memory to Data Bus Subsystem

A. Creating a Verilog module for the RISC-Y Processor:

1. Create a Verilog module for the RISC-Y processor. Use the following module header:

```
`timescale 1ns/1ns
module RISCY (CLK, RST, IO);
  input CLK, RST;
  inout [7:0] IO;
     o  o  o
endmodule
```

2. The RISC-Y processor has the following specifications:

- A Sequence Controller steps through a fetch/decode/execute/update sequence. This sequence requires four master clock cycles.
- The ROM will be 32 bits wide and 32 words deep, or 32 x 32.
- The RAM will be eight bits wide and 32 deep, or 8 x 32.
- The design is to be totally synchronous. The only clock is the primary input master clock. No derived signals may ever be used as a clock. The external, asynchronous reset must be synchronized with an AASD module.
- While it would be logically correct to use the primitive operator register design from lab 3, that function should be re-designed using behavioral code. The behavioral code should not have any delays specified. It should be scalable so that it can be instantiated in multiple locations. Most of the registers do not need a reset signal, so the scalable register can be written without one.
- The Port Direction Register does need a reset, so it can be written as a separate module that is only used one time.

- An eight-bit tristate buffer cell should also be created an instantiated for use on the IO port. This cell only requires a single line of behavioral code.

- The system shall use an asynchronous assert, synchronous de-assert reset. A module implementing this function must be incorporated into the design.

- A Program Counter (PC) points to the next location in memory from which the processor will fetch instructions or data. The PC is the counter from Lab 4, but the RISC-Y processor only needs five bits, not six. Modify your counter design to be scalable and overload it to be five bits wide when instantiated in the top level design. The PC may be loaded with branch addresses or incremented to go on to the next address in sequence.

- The ALU only receives operands from the A and B registers, which are eight bits apiece.

- There is one eight-bit input-output port. When it is an output, data can be written to it by executing a LOAD operation with the ports address (67) in the address field. It is read by performing a STORE operation with the store address in the data field and the port address of 67 in the address field.

- The data path is eight bits wide. Data may be stored in the eight LSBs of an instruction. Data also may be stored in the RAM. Once the ROM is initialized, it cannot be changed during operation.

- The port direction is set by writing to the Port Direction Register, address 66. This is a one-bit register. To simplify the design, all port bits operate together as either input or output. Direction may be changed under program control.

- Data for the one-bit Data Direction Register will be taken from the least significant bit of the referenced memory location. It can only be changed via a LOAD operation. There is no provision for reading this register.

- The tristate data bus must never be allowed to float. Exactly one driver must always be active. This means that some modification to the logic controlling memory read and write will be necessary to prevent a floating bus.

- The processor can execute the following instructions:

Lab 10 Table 1: RISC-Y Instruction Set

MNEMONIC	INSTRUCTION	OPCODE
LOAD	Load a register	4'b0000
STORE	Store data in memory*	4’b0001
ADD	Add A and B	4’b0010
SUB	Subtract B from A	4’b0011
AND	Bitwise AND of A and B	4’b0100
OR	Bitwise OR of A and B	4’b0101
XOR	Bitwise XOR of A and B	4’b0110
NOT	Bitwise inversion of A	4’b0111
B	Unconditional Branch	4’b1000
BZ	Branch if Z flag is set	4’b1001
BN	Branch if N flag is set	4’b1010
BV	Branch if OVF flag is set	4’b1011
BC	Branch if C flag is set	4’b1100

*Only the ALU output or data read from the port may be directly stored. Store always refers to the ALU output unless the address of the port appears in the address field of the STORE instruction.

The top-level RISC-Y module should contain only instances of the lower-level modules. There should be no behavioral statements in the top-level file.

Operational Notes

The RISC-Y processor uses a load/store architecture typical of RISC machines. The central feature of LS architectures is the simple nature of what individual instructions can do. Accordingly, there is no instruction that can directly move data from one memory location to another. Data in memory can only be loaded into a register. Only the accumulator or port data may be stored in memory. An implication of this is that a load operation never writes to memory and a store never reads memory beyond the initial opcode fetch.

Because the ROM is not directly connected to a multiply-driven bus, it can always be selected and its outputs always enabled. Its chip select and output enable signals can be tied to constant logic values (logic 0 and logic 1, respectively). Leaving it constantly enabled would waste power, but that is not a concern in the RISC-Y design.

Changing the port direction requires a load operation prior to the port load or store. First the LSB of the data field is stored in the Port Data Direction register. In a subsequent operation, data may be read from or written to the port. The port will retain its direction until changed. It must default at reset to input mode.

The RISC-Y processor does not use any pipelining, much less branch prediction. Each instruction takes four clock cycles, no matter how many are run in sequence. Adding a pipeline would substantially increase throughput, though at the cost of adding substantial complexity, including a mechanism for flushing the pipeline on branches.

B. Testing with a Stimulus File:

Create a testbench to verify the RISC-Y processor module. To accomplish this, write a program and load it into the memory using **$readmemh**. After the system is reset, the processor should start executing your instructions. If all the components of the system are working properly, the processor will:

- Fetch an instruction from the ROM memory.
- Decode the instruction.
- Fetch a data operand either memory if required by the instruction.
- Perform any ALU operations required by the instruction.
- Store the result when so instructed.
- Repeat the cycle by fetching the next instruction.

There is no provision to write to the ROM under program control. You may initialize it using $readmemb or $readmemh and a suitable text file. RAM will start out unknown. You can write to it under program control.

Programming the RISC-Y processor requires reverting to the techniques of the earliest days of programming. There is neither an assembler nor compiler for it. Programs are written in hex or binary.

All steps should be well documented and the results well presented! All instructions and conditions should be tested.

Multiple test program files are likely to prove advantageous rather than trying to demonstrate correct functioning of all instructions in one file.

Remember to zoom in so that submitted waveforms are readable. You may, if you wish, also submit one or more “zoomed out” overview plots.