EC 413 Computer Organization - Fall 2019
Final Project: Single Cycle RISC-V CPU

Part 1 Due Date: November 18, 2019
Part 2 Due Date: December 4, 2019

1 Introduction and Background

For the final project, you will build on the work you have done in Labs 2 and 3. In previous labs, you created a register file, ALU, simple decode module and memory. Now you will work in a team of two people to add a fetch unit to fetch instructions from a main memory and extend your existing modules to support more instructions.

Remember, you must build test benches for each module in the design. Some Test benches have been provided but there are many cases they do not cover. You must add tests to these test benches to ensure your modules function correctly. Do not modify port list or existing variable names in the template.

Before starting the lab, review page 104 in the RISC-V specification document for instruction encodings. Table 1 lists each instruction you must support and the page number where to find the instruction in the RISC-V specification.

Run the command below to launch ISE on the lab machines in PHO 305. Simulate your test benches in ISE to be sure your modules work.

```
/ad/eng/opt/xilinx/Vivado/2018.2/Vivado/2018.2/bin/vivado &
```

2 Submission Requirements

2.1 Part 1

For Part 1 of the project, you must turn in your register file, the expanded ALU and the Expanded decode module with their test benches.

Submit a zip file named **bu_id_bu_id**_project_part1.zip with your verilog modules and test benches in a directory named **bu_id_bu_id**_project_part1, where bu_id is replaced with both group member's BU ID (including the 'U'). Your Verilog files must keep the original names and port lists of the provided templates. Make sure you have put your names and BU IDs in comments in each of the files. **Submissions that do not follow these requirements will receive a -10% penalty.**

2.2 Part 2

For Part 2 of the project, submit your complete processor, with the new fetch module and top module. Be sure to resubmit the modules used in Part 1. Submit a test bench for each module in the processor.

Submit a zip file named **bu_id_bu_id**_project_part2.zip with your verilog modules and test benches in a directory named **bu_id_bu_id**_project_part1, where bu_id is replaced with both group member's BU ID (including the 'U'). Your Verilog files must keep the original names and port lists of the provided templates. Make sure you have put your names and BU IDs in comments in each of the files. **Submissions that do not follow these requirements will receive a -10% penalty.**

3 Problems

3.1 Register File

You may reuse your register file from Lab 2 (both the module and test benches). If you did not correctly implement the modules in Lab 2, you will have to check out the solution posted on the class website to make corrections, before continuing with the project.

Test your regFile module in the provided test bench template. Add test cases to ensure every operation is functional and that no corner cases exist. Submit your register file module and test bench in the Part 1 submission.

3.2 ALU

In Labs 2 and 3, we did not support every RISC-V RV32I instruction, so our ALU is incomplete. Add the necessary operations to the ALU to support each instruction in Table 1. The table lists each instruction, the ALU control signal and the effective ALU operation.

Also note that to support branch instructions, we have one new input (branch_op) and one new output (branch). These signals are described with the other ports in Table 2.

Test your ALU module in the provided test bench template. Add additional test cases to ensure every operation is functional and that no corner cases exist. Submit your ALU module and test bench in the Part 1 submission.

3.3 Decode

The decode module in Lab 3 only supported a handful of instructions. For the project you will have to support many more instructions including branches and jumps. Supporting these instructions and the new fetch unit requires additional inputs and outputs in the decode module. Table 4 describes each input and output in detail. Table 1 describes each instruction your decode module must support and the ALU_Control signal values that your decode module must generate.

Test your decode module in the provided test bench template. Add additional test cases to ensure every operation is functional and that no corner cases exist. Submit your decode module and test bench in the Part 1 submission.

3.4 RAM

In Lab 3 we built a memory module to support load and store instructions. Now we will have to support loads and store instructions while also fetching instructions. This requires an extra read port in our memory. The new ram module has this extra read port.

The data (load/store) side of the ram module is the same as the memory module in Lab 3 and you may use your Lab 3 memory.v code as a starting point for this module. For the instruction side of the ram module, you must support combinational reads to word aligned addresses. This behavior is similar to the read behavior of the data side.

Table 3 describes each port of the ram module in more detail. Remember that writes should happen on the positive clock edge and reads should happen combinationally. Note that the ram module still uses 16 address bits (as in Lab 3).

Test your ram module in the provided test bench template. Add additional test cases to ensure every operation is functional and that no corner cases exist.

3.5 Fetch

The fetch unit is a simple module that contains the Program Counter (PC) register. The PC register is updated each clock cycle with a branch target address, jump target address or PC+4. The different branch or jump targets are selected in the decode module and only one target is sent to the fetch unit.

Table 5 describes each port in detail. Test your fetch module in the provided test bench template. Add additional test cases to ensure every operation is functional and that no corner cases exist.

3.6 Top Level Module

Instantiate your fetch, ALU, register file, memory and decode modules in the top module template (top.v). Connect the modules together as shown in Figure 1. Your single cycle processor must support all instructions listed in Table 1.

Test the module in the provided test bench. The given test bench runs one of two programs: "fibonnacci.vmh" or "gcd.vmh". The fibonacci program computes the 8th fibonacci number and stores it in register x9. The gcd (Greatest Common Denominator) program computes the greatest common denominator between 64 and 48 and stores the result in register x9.

Your processor must be able to correctly execute both programs and all instructions in Table 1. You can modify the test bench to load a custom memory image (.vmh file) with instructions you have assembled manually.

Instruction	Description	ALU_Control Signal	Effective ALU operation	Page in RISC-V Spec
LUI	Load Upper Immediate	6'b000000	Add	14
AUIPC	Add Upper Immediate to PC	6'b000000	Add	14
JAL	Jump and Link	6'b011111	Pass op_A through	16
JALR	Jump and Ling Register	6'b111111	Pass op_A through	16
BEQ	Branch if Equal	6'b010000	Equals	17
BNE	Branch if Not Equal	6'b010001	Not Equal	17
BLT	Branch if Less Than	6'b010100	Signed Less Than	17
BGE	Branch if Greater Than or Equal	6'b010101	Signed Greater Than or Equal	17
BLTU	Branch if Less Than (unsigned)	6'b010110	Unsigned Less Than	17
BGEU	Branch if Greater Than or Equal (unsigned)	6'b010111	Unsigned Greater Than or Equal	17
LW	Load Word	6'b000000	Add	19
SW	Store Word	6'b000000	Add	19
ADDI	Add immediate	6'b000000	Add	13
SLTI	Set Less Than Immediate	6'b000011	Signed Less Than	13
SLTIU	Set Less Than Immediate (unsigned)	6'b000011	Unsigned Less Than	13
XORI	Bitwise Exclusive OR Immediate	6'b000100	Bitwise XOR	13
ORI	Bitwise OR Immediate	6'b000110	Bitwise OR	13
ANDI	Bitwise AND Immediate	6'b000111	Bitwise AND	13
SLLI	Logical Shift Left Immediate	6'b000001	Logical Shift Left	14
SRLI	Logical Shift Right Immediate	6'b000101	Logical Shift Right	14
SRAI	Arithmetic Shift Right Immediate	6'b001101	Arithmetic Shift Right	14
ADD	Add	6'b000000	Add	15
SUB	Subtract	6'b001000	Subtract	15
SLL	Logical Shift Left	6'b000001	Logical Shift Left	15
SLT	Set Less Than	6'b000010	Signed Less Than	15
SLTU	Set Less Than Unsigned	6'b000010	Unsigned Less Than	15
XOR	Bitwise Exclusive OR	6'b000100	Bitwise XOR	15
SRL	Logical Shift Right	6'b000101	Logical Shift Right	15
SRA	Arithemtic Shift Right	6'b001101	Arithmetic Shift Right	15
OR	Bitwise OR	6'b000110	Bitwise OR	15
AND	Bitwise AND	6'b000111	Bitwise AND	15

Table 1: RISC-V instructions you must support, their ALU control signals and pages in the RISC-V specification.

Port	Direction	Bit-Width	Description
branch_op	input	1	This signal is 1 during a branch instruction. Otherwise, it is zero.
ALU_Control	input	6	This input controls the main MUX in the ALU. This signal is generated in the decode stage of the processor based on the op-code, funct3, funct7. See Table 3 to determine ALU_Control values for each instruction.
operand_A	input	32	This is one operand input to the ALU. If the current instruction reads RS1, it is RS1. Some instruction types (like the U and J-type) do not read RS1. For these instruction the value is 0 (LUI), PC (AUIPC only) or PC+4 (JAL)
operand_B	input	32	This is the second operand input to the ALU. If the current instruction reads RS2, it is RS2. Instructions with an immidiate field place the immediate value in operand_B. Note that not all immediate fileds are the same. Some are 12 bits, some are 20. Check the specification to ensure proper sign extension and padding to 32 bits.
ALU_Result	output	32	This is the main output of the ALU. It holds the result of the operation requested by ALU_Control. Note that this result is not always the data that will be written back to the register file. For example, load instructions will not write the ALU_Result to the register file.
branch	output	1	This signal is 1 when a branch instruction branch is taken. Otherwise, it is 0.

Table 2: ALU Port List

Port	Direction	Bit-Width	Description
clock	input	1	A clock for the module.
wEn	input	1	A write enable signal for the data side. When this is 1, d_write_data should be written to the memory location specified by d_address.
d_address	input	16	The address for load and store operations.
d_write_data	input	32	The store data to be written when wEn is high.
d_read_data	output	32	The load data read from the memory.
i_address	input	16	The address of the current instruction.
i_read_data	output	32	The instruction at the given address.

Table 3: RAM module port list.

Port	Direction	Bit-Width	Description
PC	input	16	The current program counter value. This is used to compute jump and branch target addresses.
instruction	input	32	A 32 bit RV32I instruction.
JALR_target	input	16	The target of a JALR instruction.
branch	input	1	The result of a branch instruction. A 1 means a branch was taken, a 0 means the branch was not taken.
next_PC_select	output	1	A signal to select if the next PC value should be target_PC or PC+4. A value of 0 selects PC+4.
target_PC	output	16	The target of the current JAL, JALR or branch instruction.
read_sel1	output	5	The RS1 register address in the instruction.
read_sel2	output	5	The RS2 register address in the instruction.
write_sel	output	5	The RD register address in the instruction.
wEn	output	1	A write enable signal for the register file.
branch_op	output	1	A signal to tell the ALU that the current instruction is a branch. A 1 means the current instruction is a branch.
imm32	output	32	A sign extended immediate filed for instructions that use it. This should be 0 when an instruction does not have an immediate value.
op_A_sel	output	2	A control signal for a multiplexer to select which signal to feed into the ALU operand_A port. This should select between PC (when 1), PC+4 (when 2), and read_data1 (when 0).
op_B_sel	output	1	A control signal for a multiplexer to select which signal to feed into the ALU operand_B port. This should select between imm32 and read_data2.
ALU_Control	output	6	The 6 bit control signal to select the ALU operation.
mem_wEn	output	1	A write enable signal for the memory module.
wb_sel	output	1	A control signal for a multiplexer to select which data should be written back: ALU_result or the data read from the memory.

Table 4: Decode Port List

Port	Direction	Bit-Width	Description
clock	input	1	A clock for the Memory module.
reset	input	1	An active high reset signal.
next_PC_select	input	1	A signal to select either PC+4 or a jump or branch target value. The selected value is written to PC_reg on the positive clock edge. A value of 0 selects PC+4.
target_PC	input	16	The target address of a branch, JAL or JALR instruction.
PC	output	16	The current value of the program counter register.

Table 5: Fetch module port list.

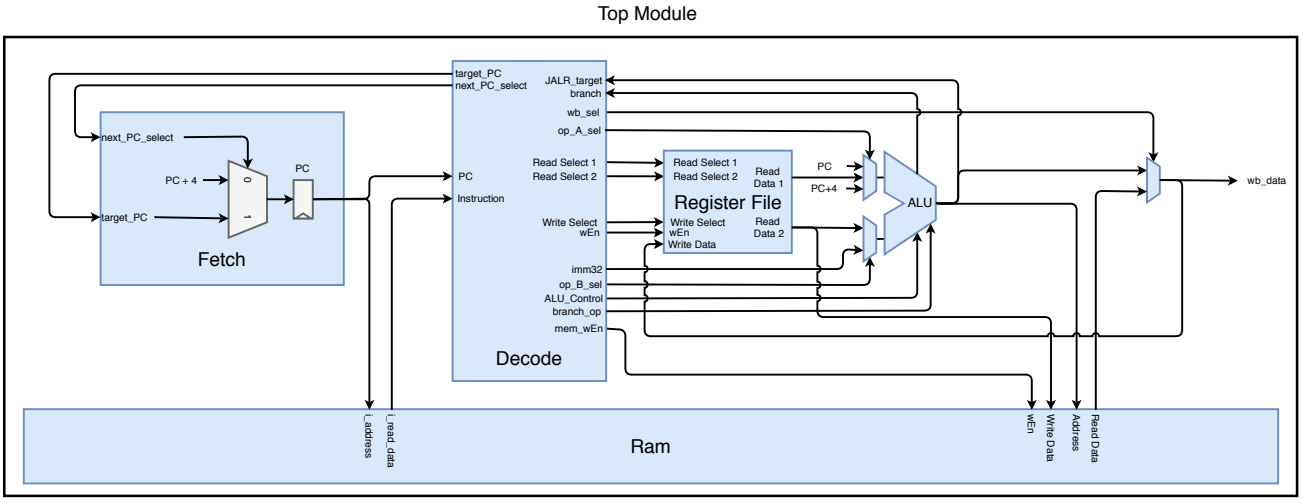

Figure 1: Top level diagram of the complete processor.

Port	Direction	Bit-Width	Description
clock	input	1	A clock for the top level module.
reset	input	1	A reset signal for the top level module.
instruction	input	32	A 32 bit RV32I instruction input to the module.
wb_data	output	32	The data to be written back to the register file.

Table 6: Top Level Port List