ECE/CS 552: INTRODUCTION TO COMPUTER ARCHITECTURE

Project Description – Phase 2

Due on Wednesday, November 7, 2018, 11:59pm

In Phase 2 of this project, the task is to design a 5-stage pipelined processor to implement the WISC-F18 ISA.

Functional blocks of the single-cycle realization in Phase 1 of this project should be reused when possible.

The main task in Phase 2 is the implementation of pipeline control. You will implement (a) control blocks for hazard (both control and data) detection and mitigation (stalls, flushes and forwarding), and (b) control and data path modifications for data forwarding, including register file bypassing. More details are described below.

Either Modelsim or Icrarus should be used as the simulator to verify the design. **You are required to follow the Verilog rules as specified by the rules document uploaded on canvas. NOTE: the only exception to this rule is the required use of *inout* (tri-state logic) in the register file. Do not use *inout* anywhere else in your design.**

1. WISC-F18 ISA Summary

WISC-F18 contains a set of 16 instructions specified for a 16-bit data-path with load/store architecture.

The WISC-F18 memory is byte addressable, even though all accesses (instruction fetches, loads, stores) are restricted to half-word (2-byte), naturally-aligned accesses.

WISC-F18 has a register file, and a 3-bit FLAG register. The register file comprises sixteen 16-bit registers and has 2 read ports and 1 write port. Register $0 is hardwired to 0x0000. The FLAG register contains three bits: Zero (Z), Overflow (V), and Sign (N).

The list of instructions and their opcodes are summarized in Table 1 below. Please refer to the Phase 1 handout for more details.

Table 1: Table of opcodes

Instruction	Opcode
ADD	0000
SUB	0001
XOR	0010
RED	0011
SLL	0100
SRA	0101
ROR	0110
PADDSB	0111
LW	1000
SW	1001
LLB	1010
LHB	1011
B	1100
BR	1101
PCS	1110
HLT	1111

2. Memory System

For this stage of the project, the memory modules are the same as in Phase 1. The processor will have separate single-cycle instruction and data memory, which are both byte-addressable. The instruction memory has a 16-bit address input and a 16-bit data output. The data memory has a 16-bit address input, a 16-bit data input, a 16-bit data output, and a write enable signal. If the write signal is asserted, the memory will write the data input bits to the location specified by the input address.

Verilog modules are provided for both memories.

The instruction memory contains the binary machine code instructions to be executed on your processor.

3. Implementation

3.1 Pipelined Design

Your design must use **a five-stage pipeline** (IF, ID, EX, MEM, WB) similar to that in the class material. The design will make use of Verilog modules that you developed as part of the homework assignments along with the modules developed for Phase 1 of the project.

You must implement hazard detection so that your pipeline correctly handles all dependences. You are required to implement full data forwarding and register bypassing, **including MEM-to-MEM forwarding as described in HW5**. You need to handle both data hazards and control hazards. You must implement predict-not-taken for branch instructions, which flushes instructions when branches are taken. You are not required to implement other optimizations to reduce branch delays such as dynamic branch prediction and branch delay slots. Branches should be resolved in the ID stage.

It is highly recommended to make a table that lists (for each instruction) the control signals needed at each pipeline stage. Also, make a table listing the input signals and output signals for the data and control hazard detection units (data hazard stalls and control hazard flushes). The data hazard detection unit should assume that data forwarding and register bypassing is available.

3.2 Stall/Flush

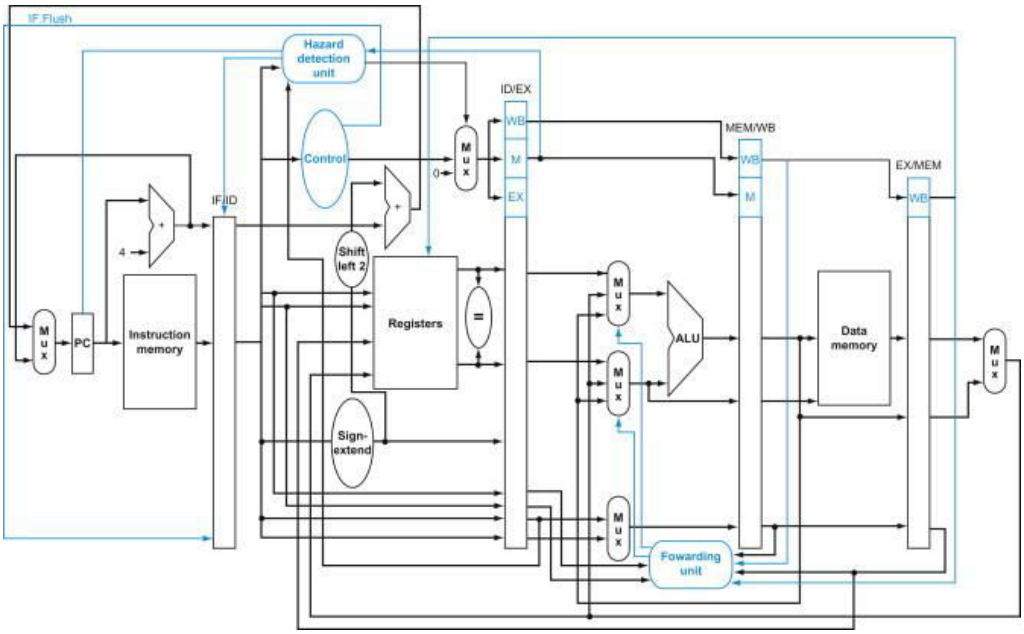
A stall is performed by using a global stall signal, which disables the write-enable signals to the D-FFs of the upstream pipeline registers (i.e., upstream from the stalling stage).
Flushes can be performed by converting the flushed operations into NOPs (i.e., preventing them from performing register/memory writes and from making changes to any other processor state, such as flags).

3.3 HLT Instruction

The HLT instruction should raise the ‘hlt’ signal only when it reaches the writeback stage. In the IF stage, you need to check for HLT instructions. If you fetch a HLT instruction, you must prevent the PC from being updated, thus stopping the program from fetching beyond the HLT instruction. The only exception is when the HLT instruction is fetched immediately after a taken branch: in this case, the HLT instruction is part of the wrong predict-not-taken branch path and will be flushed, thus the PC should be updated to the branch target address as usual.

3.4 Schematic

A summary diagram of the 5-stage pipeline is shown below (COD Figure 4.65):



Note: Not all hardware components and signals are shown here. There are more detailed diagrams of each stage of the pipeline in your textbook (along with details of how hazards are detected and resolved). You can refer to them for building the pipeline stages of the project.

3.5 Reset Sequence

WISC-F18 has an active low reset input (rst_n). Instructions are executed when rst_n is high. If rst_n goes low for one clock cycle, the contents of the state of the machine is reset and starts execution at address 0x0000.

4. Interface

Your top level Verilog code should be in file named ***cpu.v***. It should have a simple 4-signal interface: ***clk***, ***rst_n***, ***hlt*** and ***pc[15:0]***.

Signal Interface of ***cpu.v***		
Signal:	**Direction:**	**Description:**
clk	in	System clock
rst_n	in	Active low reset. A low on this signal resets the processor and causes execution to start at address 0x0000
hlt	out	When your processor encounters the HLT instruction, it will assert this signal once it has finished processing the last instruction before the HLT
pc[15:0]	out	PC value over the course of program execution

5. Submission Requirements

1. You are provided with an assembler to convert your text-level test cases into machine level instructions. You will also be provided with a global testbench and test case. The test case should be run with the testbench and the output (as a .txt file) should be submitted for Phase 2 evaluation.

2. You are also required to submit a zipped file containing: all the Verilog files of your design, all testbenches used and any other support files.