CS-UY 2214 — Homework 5

Jeff Epstein

Introduction

Unless otherwise specified, put your answers in a plain text file named `hw5.txt`. Number each answer. Submit your work on Gradescope.

You may consult the E20 manual, which is available on Brightspace.

Problems

1. Write an E20 assembly language program that will store the value 1099 at memory cell 456, then halt. Use your E20 assembler to make sure that your program is correct and can be assembled into valid machine code. Your solution should consist of no more than seven instructions.

2. Examine the single-cycle circuit diagram and its description in the E20 manual. In particular, pay attention to the meaning of the various control signals, and the possible values they can carry.

   We want to add a new instruction to the single-cycle E20 processor, while preserving as much of the existing hardware as possible.

   Consider the following instruction specification:

   swi imm, $regAddr

15 14 13	12 11 10	9 8 7	6 5 4 3 2 1 0
3 bits	3 bits	3 bits	7 bits
???	regAddr	000	imm

   Mnemonic: *Store word immediate*

   Example: `swi 5, $2`

   Example: `swi foo, $4`

   Stores the value signed value `imm` to the memory address in `$regAddr`.

   The memory address is interpreted as an absolute address. All 16 bits of the value of `$regAddr` are used to index into memory.

   Symbolically: `Mem[R[regAddr]] <- imm`

   Note that the numeric opcode isn't specified, because it isn't relevant for this exercise.

   Based on the single-cycle E20 circuit diagram in the E20 manual, describe in detail any changes necessary to the single-cycle E20 design in order to implement the specified instruction. Explicitly mention any hardware to be added or changed. Include in your discussion any new control wires. In addition, specify the values of all control signals (old and new) that should be set when the given instruction is executing. That is, for each of the control wires (FUNC_{alu}, MUX_{alu}, etc), including any additional control wires that you add, give their value. You may, if you like, provide an updated circuit diagram. Explain and justify all proposed changes.

Your modifications to the single-cycle E20 must not interfere with the execution of any other instruction. You should add the minimal amount of hardware necessary to accomplish the goal of implementing the specified instruction.

3. Consider two computers:

    - Computer A has a 5 GHz clock frequency and an average CPI of 4.
    - Computer B has a 3 GHz clock frequency and an average CPI of 2.

    Assuming the two computers are otherwise equivalent, which computer will run your programs faster? By how much (as a percent)? Justify your answer.

4. Examine the multicycle circuit diagram and its description in the E20 manual. In particular, pay attention to the meaning of the various control signals, and the possible values they can carry.

    We want to add a new instruction to the multicycle E20 processor, while preserving as much of the existing hardware as possible.

    Consider the following instruction specification:

    jmpm imm($regAddr)

15 14 13	12 11 10	9 8 7	6 5 4 3 2 1 0
3 bits	3 bits	3 bits	7 bits
???	regAddr	000	imm

    Mnemonic: *Jump via memory*

    Example: `jmpm 5($2)`

    Calculates a memory pointer by summing the signed number `imm` and the value `$regAddr`, loads the value from that address, and jumps to the loaded address.

    The memory address is interpreted as an absolute address. The least significant 13 bits of the value of `$regAddr + imm` are used to index into memory.

    Symbolically: `pc <- Mem[R[regAddr] + imm]`

    Note that the numeric opcode isn't specified, because it isn't relevant for this exercise.

    Based on the multicycle E20 circuit diagram in the E20 manual, describe in detail any changes necessary to the multicycle E20 design in order to implement the specified instruction. Explicitly mention any hardware to be added or changed.

    In addition, For each of the five stages of execution of the instruction, specify the correct behavior of the processor in one of two ways: either give the number of an existing execution state (0 – 13), *or* give a new execution state by specifying the values of all relevant control signals. For control signals that are not relevant, write "don't care."

    You may, if you like, provide an updated circuit diagram. Explain and justify all proposed changes.

    Your modifications to the multicycle E20 must not interfere with the execution of any other instruction. You should add the minimal amount of hardware necessary to accomplish the goal of implementing the specified instruction.

5. We've previously built a single-cycle E15 implementation in Verilog. Now, we will build a multicycle implementation. The visible behavior of the multicycle implementation should be identical to that of the original. However, now each instructions executes in four serial phases: fetch, decode, exec, and store.

   Download and extract the file `e15_multicycle_incomplete.zip`, containing an incomplete implementation of a multicycle E15 processor in Verilog.

   Read `E15Process.v` completely. Note the areas marked with a "`TODO`" comment: your task is to replace these lines by providing appropriate statements. Do not modify any other code. A correct implementation will allow your `E15Process.v` to execute any E15 assembly language program.

   In particular, you must provide code at the following points:

   - Provide the complete decode phase. The decode phase is responsible for sending values to the ALU on the mBus and the dBus. Your code should assign appropriate values to the registers `mbEn`, `dbEn`, and `addNotSub`, based on current instruction, which is accessible in the `opCode`, `src`, `dst`, and `immData` registers.

     In order to determine the correct values of `mbEn` and `dbEn`, you'll need to examine the continuous assignment for `mBus` and `dstBus`, provided at the end of the module. Those wires provide inputs directly to the ALU.

     You'll also need to update `myState` with the next phase.
   - Provide code for setting `pcIncr` in the exec phase. This will be very similar to your code in the single-cycle version of the processor. `pcIncr` is passed as an input into the `pcALU`, which determines the address of the next instruction.
   - In the store stage, provide code for storing the ALU result into an appropriate register, if necessary. Not all instruction write a value to a register. Because `mbEn` was set to `bEn_ALU` in the previous phase, `mBus` will contain the ALU output.

   Test programs are included. You should verify the correct function of your E15 processor by comparing each program's actual output against its expected output.

   Submit only your complete `E15Process.v`.