COMP 273

Pipeline CPU Project

Project Due: December 3 to 5, 2012 at 23:55 on My Courses

Assemble a team of 2 or 3 students, 3 are preferable, who will work together to construct the solution to this project. Only your team leader will submit the project to the Project assignment box on My Courses. Make sure to include a text file identifying your team member's by name and ID number, so that everyone can get the grade. Include your own name and ID number in this list. The what-to-hand-in section details how this submission should be constructed.

The reason there are 3 days for the submission is because you will demo the program to the TA or Professor during one of those three days (December 3, 4, or 5). You will have to make an appointment, so the earlier you do this the better. You can even start now! (do this by email)

PROJECT (15 points + 3 bonus points)

This project is segmented into two parts: the basic question and the bonus question. All teams must do the basic question. The bonus question is available if you found the basic question too easy.

You must construct your project using LOGISIM.

BASIC QUESTION

Build an operational Pipeline CPU circuit diagram for a 4 stage pipeline CPU using LOGISIM. Your CPU is contained within a "system board". This system board contains the following additional elements: a 1-digit digital display with it's companion 8-bit output data register, and an 8-bit input buffer connected to a keypad (simulated by Logisim's input pins where each pin is one of the numeric digits from 0 to 9 from a keypad).

The description below is organized by first describing the pipeline CPU. Then it describes the digital display and input keypad buffer.

Your Pipeline CPU includes the following NON-parallel-executed stages:

Your CPU's macro-structure:

Your pipeline CPU will have the following elements: PC, 16-byte Instruction Cache, IR + CU, only 2 GP Registers, ALU + Status register, 16-byte Data Cache. There will be two private buses, one will connect to the display the other to the keypad. Data exiting the ALU can be directed either to the Data Cache, the GP Registers, the PC, and the display output register. Instructions can request for input data from either the Data Cache, the GP registers, and the keypad. Notice there is no RAM, only the two caches.

It is important to note that this pipeline CPU does not execute the instruction in parallel. Instructions are executed to completion, as in a classical CPU, before the next instruction is fetched.

Your CPU's execution cycle:

Stage 1: Retrieve one instruction from cache
Stage 2: Get instruction's arguments (from Data Cache, GP Register, or Keypad data register)
Stage 3: Execute the instruction (eg. ALU operation, JUMP, or MOVE)
Stage 4: Save the result (if any, into Data Cache, GP Registers, or display buffer register)

In order to make the drawings easier you may use black boxes in the following cases:

- Only after you have designed a full sample circuit in detail

- You also do not need to design an entire circuit if the rest of the circuit is identical to the part you just drew.
- You can reuse black-boxed components in other areas of your diagram.
- The only LOGISIM elements you can use are: flip-flop, or-gate, and-gate, not-gate, and the "off-the-shelf" circuits we covered in class.
- You can reuse components from your assignments, specifically (and optionally):
  - Register
  - Full-RAM
  - Adder

Your CPU must have the following elements:

1. Addresses are 4-bits
2. All data and instructions are 8-bits
3. Your program can only store unsigned integer numbers (max 1 byte in length)
4. The PC is 4 bits long
5. The IR is 8 bits long
6. The Sequencer (CU) controls the CPU and understands the following machine language instructions:
   - 6.1. ADD → ADD Rx, Ry (the result is stored in Rx)
   - 6.2. STORE → STR Rx, Address (the contents of Rx is put into Data Cache at Address)
   - 6.3. LOAD → LD Ry, Address (the contents at Address from Data Cache is put into Ry)
   - 6.4. Branch if Zero → BZ Rx, Address (if Rx is zero then go to Instruction Cache Address)
   - 6.5. PRINT → PRT Rx (output contents of Rx to 1-digit digital display buffer)
   - 6.6. READ → INP Rx (store data to Rx from keypad input buffer)
   - 6.7. STOP → RETURN (mysteriously, the program stops running)
     - You must supply your own binary definition for these instructions (i.e. its op-code, parameters, etc.)
     - Rx and Ry can be: R0, R1
     - R0 and R1 can be used for indirect addressing
7. The CPU Sequencer must be able to implement the above 7 instructions.
8. You have a single clock
9. Two general-purpose registers: R0 and R1.
10. An ALU System, consisting of:
    - 10.1. LEFT and RIGHT input registers
    - 10.2. A-OUT resultant register
    - 10.3. A 4-bit STATUS register with only two flags: overflow? and is-zero? (two extra spots for bonus)
    - 10.4. The ALU can only perform 8-bit unsigned integer addition
11. You do not need to concern yourself with how the first instruction got into the PC. You do not need to concern yourself with how information arrived in the cache in the first place. Just assume that the PC is initialized to zero and the cache has a program with data already at address zero
12. Please provide the following program written in your language that can fit in your memory and be executed by your CPU. We will execute that program when we test your design. Write a program that multiplies two numbers. One number comes from the data cache and the other comes from the input buffer. The solution is displayed on your 1-digit digital display.

Other than the items listed above, you are free to design your CPU in any way you like.

<u>1-digit Digital Display</u>

Your 1-digit digital display should be patterned along the slides presented in class. Specifically this would be Lecture 6/7 the slides covering the topic Binary Code Decimal Example. You will build a simple digital calculator-like 1-digit display. LOGISIM has an additional black-box that you are permitted to use. It is part of the built-in library extensions called Input/Output. Within that extension is a component called 7-Segment Display. You will need one of these. The CPU instruction PRT will convert the integer number stored in a register and display that to this digital display.

8-bit Input Buffer

Your circuit will also contain an input device. We will imagine that it is connected to a 10-digit keypad, but for our purposes it will simply be 10 LOGISIM input-pin connectors sitting to one side of your circuit diagram representing a byte of data coming from the keyboard. The user can "press" one "button" and a digit will be stored in the buffer. Your CPU can read these bits when the instruction INP is present. The instruction does not wait for the user.

Please provide a name for your CPU (something cool would be nice).

BONUS QUESTION

Listed below are three possible upgrades to your CPU. Each upgrade is worth 1 bonus point. You do not need to answer this in any particular order. Select 1, 2 or all three features to implement. There are no part marks for the bonus questions. You either get 0 or 1 point.

- Implement a 2's complement feature for your integer numbers in the ALU by providing for these two additional CPU programming instructions:
  - COMP Rx → Rx is converted to 2's complement, stored back into Rx (uses ALU)
  - SUB Rx, Ry → Rx = Rx – Ry, using 2's complement addition (use your own implementation)
  - Note: by default your memory and registers will now store signed integer numbers (in unsigned or 2's complement formats)

- ON/OFF Switch
  - Provide circuitry that combines the clock with an ON/OFF switch for the computer. When the switch is on and the clock ticks the PC is set to zero and the computer starts executing the program in memory. How the program got there, we don't know. But you must initialize the PC to zero. You MUST use a LOGISIM black-box for this option as well. The Input/Output library has a Button box. The user must press this button to turn on/off your computer.
  - Implement an additional computer instruction: HALT. This command turns off the computer.

- Multiplication
  - Upgrade the ALU so that it can multiply! There is an easy way and a hard way to do this…
  - Provide an additional instruction: MULT Rx, Ry. The result it stored in Rx.

WHAT TO HAND IN

Everything should be handed in electronically on My Courses.

Students must work in teams of 2 or 3 people. Only the team leader hands in the project to My Courses. In addition to his or her solution, please also submit a text file identifying each team member by name and student ID number. Each person in the team will receive the same grade. This text file will help us during grading.

Hand in all the LOGISIM files (including the black boxes) with a README.TXT file if the TA needs special instructions for running your circuit. Only those portions of your design that actually run will be graded.

Make sure to make an appointment with either the TA or the Professor to demo your project. You will have about 5 to 10 minutes to show that your project works (or which parts work).

Also submit a simple document that provides the following information: A black box summary of your entire CPU. This should fit on a single printed page. It would be a high-level view of the entire CPU. This includes an additional page of text describing how it functions. Your document should give some information about each LOGISIM file you uploaded, what it is and how it should be used. The very last page of your document should be your assembler-like program and a description of your assembler language.

Submit your document as a WORD, EXCEL, PDF or JPG file. Submit your design as LOGISIM file(s). We will execute these files to confirm that your design runs. Copied files will lose all grades. Make sure to submit this in the box called PROJECT. There are no late days.

Name your CPU. Provide this information in the document.

Project Document

Your project document should have the following elements:

1. A cover page
   1.1. Project title/CPU name
   1.2. Name of each team member with ID number
2. Section 1: Overview of CPU
   2.1. High level diagram of the entire CPU in one picture (one page)
   2.2. Overview of its functionality and flow (one page)
3. Section 2: LOGISIM File Breakdown
   3.1. Using a top-down approach, describe each part of the CPU (one or two pages)
4. Section 3: The Assembly Language (two to four pages)
   4.1. Define your instructions in assembly
   4.2. Define your instruction bit formatting
   4.3. Describe how your instructions execute over the CPU
   4.4. Provide the requested program
5. Section 4: Appendix (optional)
   5.1. This place is for miscellaneous other things important for TA to have

HOW IT WILL BE GRADED

The mini-project is worth 15 points plus 3 bonus points.

- The basic project is worth 15 points
  - Graded proportionally
  - Design 10 points
    - CPU – 7 points
      - Data cache – 1 point
      - Instruction cache – 1 point
      - Registers – 1 point
      - ALU – 1 point
      - PC + IR + CU – 2 points
      - Execution cycle – 1 point
    - DISPLAY – 1 points
    - INPUT – 1 point
    - Document – 1 point (but can loose points in other places if design is hard to read)
  - Running program 5 points
    - Branching – 1 point
    - Instruction bit format – 1 point
    - Addressing – 1 point
    - Program runs – 2 points
- Bonus points (3 in total) for mini-project
  - 1 point for each bonus implemented correctly, 0 points otherwise (not graded proportionally, graded binary = a full yes or a zero)