Carleton University

Department of Systems and Computer Engineering

SYSC 3006 (Computer Organization) summer 2020

Lab / Assignment 3

Prerequisites

Lab / Assignment 2, series 5 and all related materials posted on cuLearn.

Goal

- Program a revised datapath and extend the design to include the microarchitecture feature of loading instructions from Main Memory (as discussed in lectures);
- Implement and test all designs using Logisim.

NOTE: In this lab description, most figure numbers refer to figures in "**Notes Towards Hardware (V8)**" posted in the course website on cuLearn.

Introduction

In lab 2 – part 1, you designed a simple processing system circuit. This circuit requires a unique Control FSM for each operation. Then in lab 2 –part 2, you generalized this simple circuit by adding the 16-bit encoded operation latch. The introduction of the latch enables a single Control FSM to execute every operation. In this lab, we are going to further develop the circuit into the microarchitecture. We will be renaming the memory words and encoded operation latch to registers and the Instruction Register (IR), respectively. In lab 2 – part 2, we introduced the FSM "Decode" state, at address 0, then E0 starts at address 1 (D, E0, E1, E2, or 00, 01, 10, 11 as ROM address). Then at the decode state, an external combinational logic circuit is used to decode the opcode and generates the next execution state. In this lab we will be introducing the 3 fetch states. Therefore, the decode state will be moved from address 0 to address 4. Then the three execution states become states 4, 5 and 6 (E0, E1 and E2 before).

In this lab we will be replacing the decoding combinational circuit (OneSourceOp...), by a Decode ROM (figure 1 below). If you completed Lab-2 Part 2, you should find the Decode ROM reasonably easy to understand. So when the decode state is detected (from the control FSM Output current state), and the opcode is fed as address to the Decode ROM. Then for each opcode address, the Decode ROM contains (as data word) the next state that follows the Decode state for each instruction.

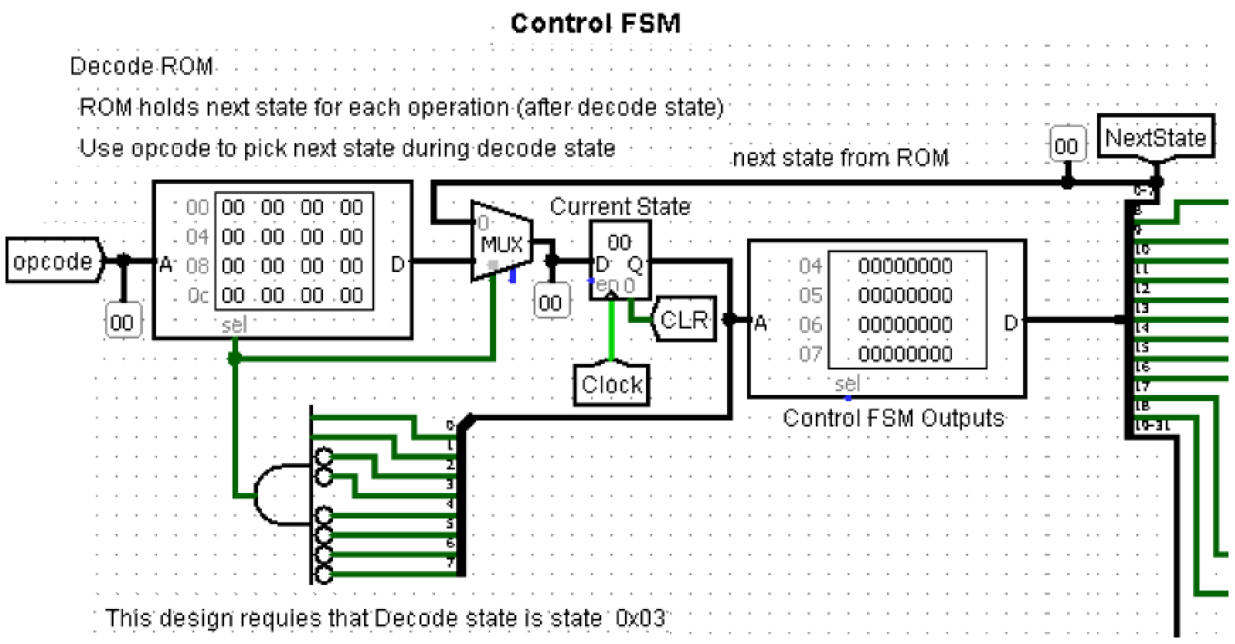
Control FSM
Decode ROM
ROM holds next state for each operation (after decode state)
Use opcode to pick next state during decode state
next state from ROM
NextState
Current State
MUX
CLR
Clock
Control FSM Outputs
opcode
sel
This design requies that Decode state is state 0x03

Figure 1: Revised circuit to include the Decode ROM

The FSM uses the Next State encoded in the FSM as the next state for all FSM transitions except for the transition out of the Decode state. When the state machine is in the Decode state (state 3), it does not use the Next State encoded in the FSM; instead, it uses a value encoded in the Decode ROM. The Decode ROM is addressed using the instruction opcode (from the IR). The Decode ROM must be programmed to contain the correct “Next State” that follows the Decode State for each instruction. For example, the opcode for the ADD instruction is 0x01, and therefore the contents of the Decode ROM data word at address 0x01 should be 0x04 (the correct state following the Decode State for the ADD instruction). Therefore, the decode ROM gives you the flexibility to decide what in the next state of each single opcode separately. This configuration, allows having n number of different execution possibilities (figure 59) for each different type of instructions. So at each time the machine get into the decode state, it will take only one path of a multiple (jump to the address of the first execution state of that specific path). Till now we have seen only two different type of instruction requiring different execution states (e.g., ADD and NOT), but in next segment of this course we will be introducing more. Here comes the advantage of introducing this method of decoding. Regardless of the execution path your machine takes after de decode state, it always returns to F0, to fetch the next instruction from the Main Memory. Then same cycle repeats again.

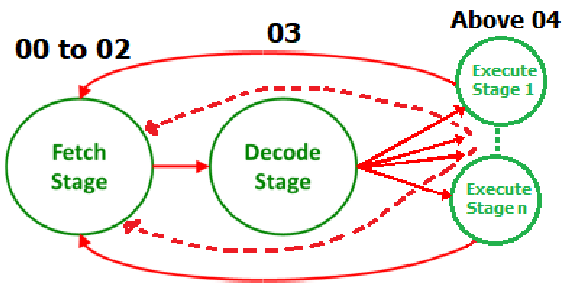

Figure 59: Instruction Cycle Stages

In this lab, the Instruction Register is extended from 16-bit to 32-bit wide (8 hex digits: $H_7 H_6 H_5 H_4 H_3 H_2 H_1 H_0$), as discussed in class. In the instruction encoding, each reference to a register requires 4 bits (since there are 16 registers) and the opcode is encoded in 8 bits. The 32-bit Instruction Register uses the bit layout from figure 43.

bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
use	Op								Rd				Rsx				Rsy				Not used (always 0) for now!											

Figure 43: Instruction Register Layout

Terminology note: *Instructions* are fetched from Main Memory and then decoded and executed over several states; while *operations* are performed by the ALU in one state.

Recall that the 8-bit opcodes for the instructions that use the ALU are obtained by extending the 4-bit ALU operation with leading 0's. For example, the ADD instruction will have the 8-bit opcode = 0x01, which is obtained by extending the 4-bit ALU ADD operation (0x1) with leading 0's. The complete 32-bit ADD instruction R3 $\leftarrow$ [R2] + [R1] would be encoded in hexadecimal as: 0x01321000.
The MOV (Move) instruction has been introduced as an instruction for copying a register to another register (i.e. using the RY ALU operation).
A NOP instruction is different from all of the other instructions that use the ALU since it does not generate any useful result. Think carefully about how to deal with a NOP instruction in the lab (hint: in this lab it can be resolved at the Decode state).

Logisim files setup and components information

A folder containing 2 Logisim circuits has been included. **You MUST UNZIP the folder** (i.e. extract all files) before trying to load these circuit files into Logisim. If you do not unzip the files, they may load into Logisim, but will not simulate properly!!!

Part 1

Load the supplied Lab-III_Part1 circuit into Logisim (figure 1 abouve). It is very similar to your solution to Lab II. Do not modify the components or wiring of this circuit in Part 1. The Decode state is state 3.

Part 2

A major difference from Lab II is that the FSM has been expanded to allow for fetching an instruction from Main Memory. When finishing part 1 save your circuit, then use "save as" and change it to Part 2. then follow the instructions in the assignment section below to complete the lab.

Your Assignment

Use the included editable .docx file (MS-Word) for your answers, then save it as PDF file before submitting it. Here we refer to tables to be filled in, they are not shown here but they do in the assignment .docx file.

Part 1 – [2.4-mark/5]

1-1

a) [0.50-mark] Fill in the Part 1 Instruction Encoding Table for the following instructions.

1- R5 ← NOT R4
2- R7 ← R6 – R5
3- NOP
4- R0 ← R7

b) [0.50-mark] Complete the FSM Output ROM for part 1.

c) [0.50-mark] Complete the FSM Decode ROM Tables (this table is same for Part 1 and 2 of this lab).

NOTE: In the supplied circuit, the Control FSM outputs the RegSEL and RegLD signals with the behaviour expected by the Registers RAM. The FSM Output ROM does not use RegR and RegW signals as done in Lab-II and in class; the RegSEL and RegLD signals (with Logisim RAM signaling behaviour) are used instead.

Do not change any of the values that have been pre-entered into the table, except for the empty cases in the hexadecimal encodings (in Part 1 FSM Output ROM Table). The first 3 states are there only as placeholders in Part 1 and have been designed to have no undesirable effects on the circuit. This allows you to always start the FSM in state 0.

The resulting FSM Output ROM and FSM Decode ROM values should work for any of the instructions discussed in class that use the ALU (i.e. NOP, ADD, SUB, MOV, AND, OR, XOR and NOT).

d) [0.50-mark] Save your circuit as Lab-3_Part1.circ and submit it with your assignment for verification and to get the marks for this section.

e) [0.40-mark] Same as you did in lab 2, execute the instructions in the table above in the given sequence with all registers initially containing 0x0. Log the execution of the sequence on your implementation to validate the execution of the required instructions and show the results here. (The simulation log should include: Current State, IR, PC, registers, and Next State. Set the log radix to hex).

Part 2 – [total of 2.6-mark/5]

Extend your solution to Part 1 to add the ability to load instructions from Main Memory (as discussed in lectures). The circuit in Part 1 has been set up to simplify the extension. The Decode ROM values used in Part 1 do not require any modification for Part 2. The Part 1 FSM ROM Output Table includes all of the new signals needed in this part (however, many of the values used in Part 1 for these signals won't work in Part 2). To get access to the new bits needed in Part 2, replace the "NotUsedInPart1" tunnel with a splitter. You might find it useful to break out the splitter outputs using tunnels with meaningful names (as done with the FSM outputs used in Part 1). The names of the new signals in the table are the same and have the same function as those used in the lecture slides.

You will need to add the "new" Processor hardware parts discussed in:

Figure 45: connect the IR to the Internal Data Bus, add the IRCE and clock signals (make sure you get the right clock signal), and add the PC address generator with the PCOE control signal.

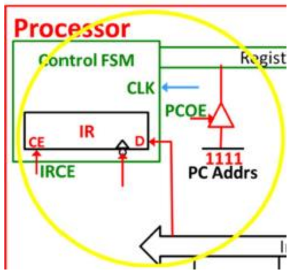

Figure 46: add the 32-bit constant 1 and the C1OE control signal.

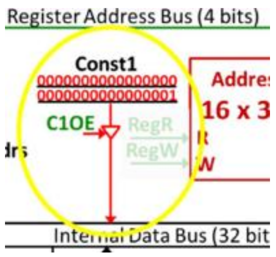

Figure 48: add the Interconnection Bus Interface as shown.

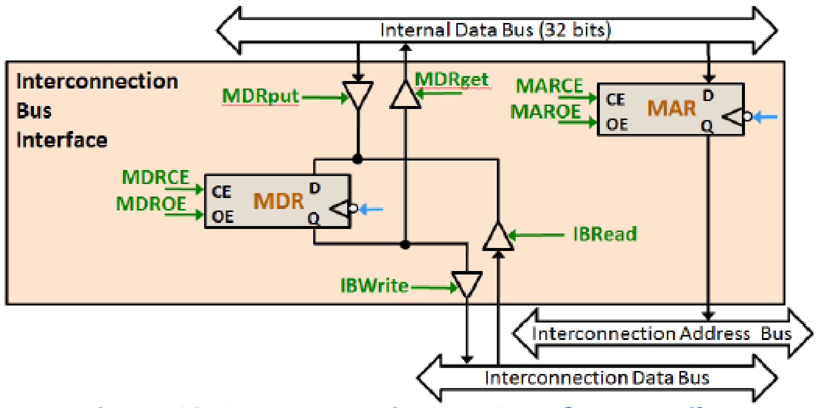
Internal Data Bus (32 bits)
Interconnection Bus Interface
MDRput
MDRget
MARCE
MAROE
CE
OE
MAR
D
Q
MDRCE
MDROE
MDR
IBRead
IBWrite
Interconnection Address Bus
Interconnection Data Bus
Figure 48: Interconnection Bus Interface Details

Figure 50: add the ADD operation and its AADD control signal.

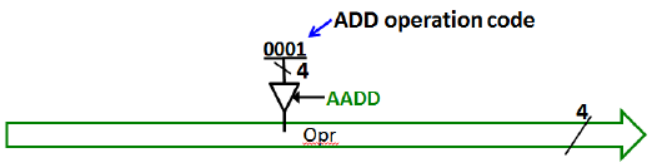

Figure 50 Generating the ALU ADD Operation in the Control FSM

You will need a Main Memory component select decoder, similar to that shown in Figure 41. This decoder should assert the SEL of the main memory to allow access to its words range (e.g, 0x000). This step is important in computer addressing e.g., in the event where we add another memory component with same addressing rage, this new component should have a different decoding to asserted the SEL (e.g.0x001) of the new component. You can see this as the telephone area code, two cities could have same telephone numbers range, but the area code for each city is unique.

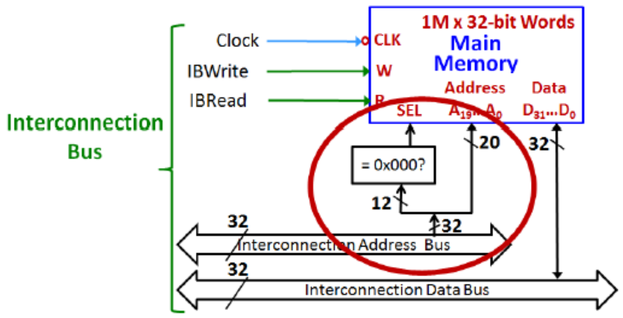

Figure 41: Using Address Signals to Select the Component

To simplify this part, a Main Memory component and surrounding circuit for use in Part 2 has already been added … do not modify the given Main Memory circuit and build your solution to be compatible with the names used in the tunnels. For simplicity, a 64 x 32-bit Main Memory has been included (i.e. the lower 6 address bits on the Interconnection Address Bus are used to select a word, while the upper 26 bits must all be 0 to be a valid memory address within this memory bank). Also note that Figure 41 shows the Bank of Memory Words that was developed in class, while Logisim's RAM component has a few differences (as you discovered in Lab-II !). The supplied circuit accounts for the differences!

2.1 - Tables

a) [0.50-mark] Complete the Part 2 FSM Output ROM Table.

b) [0.40-mark]Complete the Part 2 Main Memory Instructions Table for the following instructions.

1- R10 ← R1 OR R2
2- R11 ← R2 – R10
3- R12 ← NOT (R11)
4- R13 ← R0 + R12

2.2 - Circuit wiring

Save a copy of your part 1 (Save as) and name it "Lab3-Part-2.circ". Then extend your Processor solution to Part 1 to read and execute instructions from Main Memory. You will not need to modify the circuit beyond the description of modifications given above.

a) [0.30-mark] Show below a screenshot of the new control FMS outputs that you added in the Logisim circuits, and a short description about each output.

b) [0.40-mark] Show here a screenshot of the new hardware components that you added in the Logisim circuits. Include a short description about each component function.

c) [0.30-mark] Include a copy of your Lab3-Part-2.circ circuit with your submission. We must verify your circuit functionality in order to assign you marks for this part (c)

2.3 Execution test

a) [0.40-mark] The Part 2 Main Memory Instructions Table in 2.1 –b) contain same instruction sequence from your lab 2-part 2, but here they are encoded over 32-bit word width. Clear the 16 internal register bloc and the Main memory to 0x0. Then as you did in lab 2, initiate R0 to 0x00000001, R1 to 0x1000000F and R2 to 0xF0000000. Then insert the instructions from the Table in 2.1 –b) above into the Main Memory starting at address 0 and up (one instruction per address word as indicated in the table) in the given sequence. Now, execute all the instructions (repeat fetch-decode-execute cycle till all instruction are fully executed). To execute all instruction just poke the Toggle Switch to advance the FSM through the operation till all instructions are fully executed (while observing their execution). Make sure you PC (R15), holds the address of the first instruction to be fetched, then observe it increments… to fetch the next... Same as you did in part 1, log the execution of the sequence on your implementation to validate the execution of the required instructions and show the results here.

b) [0.30-mark] Compare the concept used here to your lab 2-part 2, briefly describe here what is the advantage of the concept here over lab2-part 2?

Submission deadline

Must be submitter on cuLearn, locate (Assignment 3 submission) and follow instructions. Submission exact deadline (date and time) is displayed clearly within the Assignment 3 submission on cuLearn.

Note: If you have any question please contact your respective group TA (see TA / group information posted on cuLearn) or use Discord class server.

Good Luck