Introduction to Computer Organization – Fall 2025

Homework 2

***Due: Oct. 6th, 2025 @11:55pm on Gradescope***

Name: \_\_\_\_\_\_\_\_\_\_\_\_Kowei Huang\_\_\_\_\_\_\_\_\_\_\_ Uniqname: \_\_\_\_\_\_\_\_\_\_hkowei\_\_\_\_\_\_\_\_\_\_\_

1. Submit a pdf of your typed or handwritten homework on Gradescope.
2. Your answers should be neat, clearly marked, and concise. Typed work is recommended, but not required unless otherwise stated. Show all your work where requested, and state any special or non­-obvious assumptions you make.
3. You may discuss your solution methods with other students, but the solutions you submit must be your own.
4. **Late Homework Policy:**  Submissions turned in by 1:00am the following day will be accepted but with a 5% penalty. Assignments turned in between 1:00am and 11:55pm (about 24 hours after the due date) will get a 30% penalty, and any submissions made after this time will not be accepted.
5. When submitting your answers to Gradescope you need to indicate what page(s) each problem is on to receive credit. The grader may choose not to grade the homework if answer locations are not correctly indicated.
6. After each question (or in some cases question part), we’ve indicated which lecture number we expect to cover the relevant material. So “**(L7)**” indicates that we expect to cover the material in lecture 7.

### **Problem** 1**: Linking** [7 points]

If a function is called from different locations, do the function’s accesses of its (non-static) local variables need to be put in the relocation table and adjusted by the linker for every call? Why or why not? *Briefly* explain your answer. **(L7)**

**No. Because the non-static local variables are only alive across the life of the function. So we don’t need to assign an address for it in the data section. What actually happens is that these variables are in the stack frame of the function during its execution. The stack dynamically allocates space for these variables when the function is called and de-allocates it upon return.**

### **Problem** 2**:** Wires Encoded [10 points]

Say you have a 4-bit unsigned number X[3:0] (so X3 is the most significant bit, X2 is the next most significant, etc.). Using only standard gates (AND, OR, NOT), and constant 0s and 1s, multiply X[3:0] by 2, add 1, and then multiply it by 4, presenting the output as an 8-bit unsigned number Y[7:0], i.e., Y[7:0] = ((X[3:0] \* 2) + 1) \* 4. Use as few gates as possible, even if it means you use no gates. **(L8)**

X3

X2

X1

X0

Y7

Y6

Y5

Y4

Y3

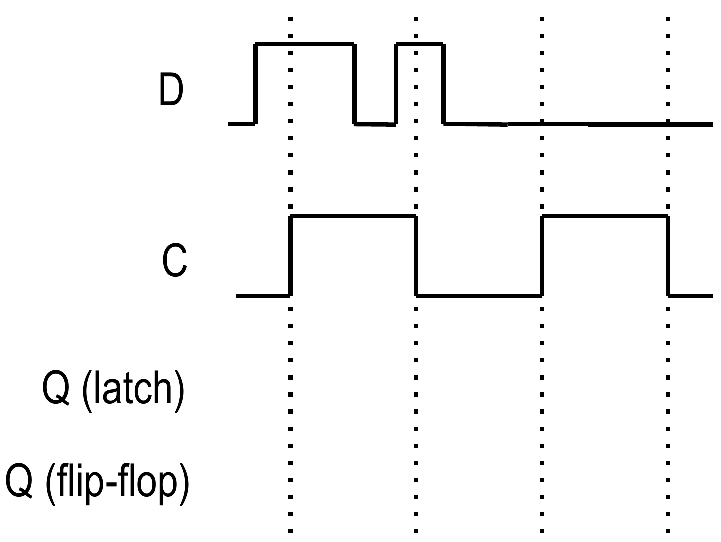
Y2

Y1

Y0

### **Problem** 3**: Latches and flip-flops** [10 points]

Complete the timing diagrams for a D latch and D flip-flop given the provided inputs. The flip-flop is positive-edge triggered. For the latch we are using “C” for the “G” input (which is fairly common). Assume both the latch and the flip-flop have an initial value of zero. **(L9)**



### **Problem** 4**: State machines** [10 points]

Answer the following questions about state machines. **(L9)**

1. Consider a state machine implementation with 9 state bits. What is the maximum number of states you can have with such an implementation? **[4]**

1. Say you have a state machine with 3 bits of input, 6 bits of output, and 64 states. How large would your control ROM need to be? Fill in the blanks below. **[6]**

*\_\_\_\_\_ bits of address  
  
 \_\_\_\_\_ bits of data per address*

*A total of \_\_\_\_\_\_\_\_\_ bits in the ROM.*

### **Problem** 5**: Incremental Improvement** [33 points] **(L11)**

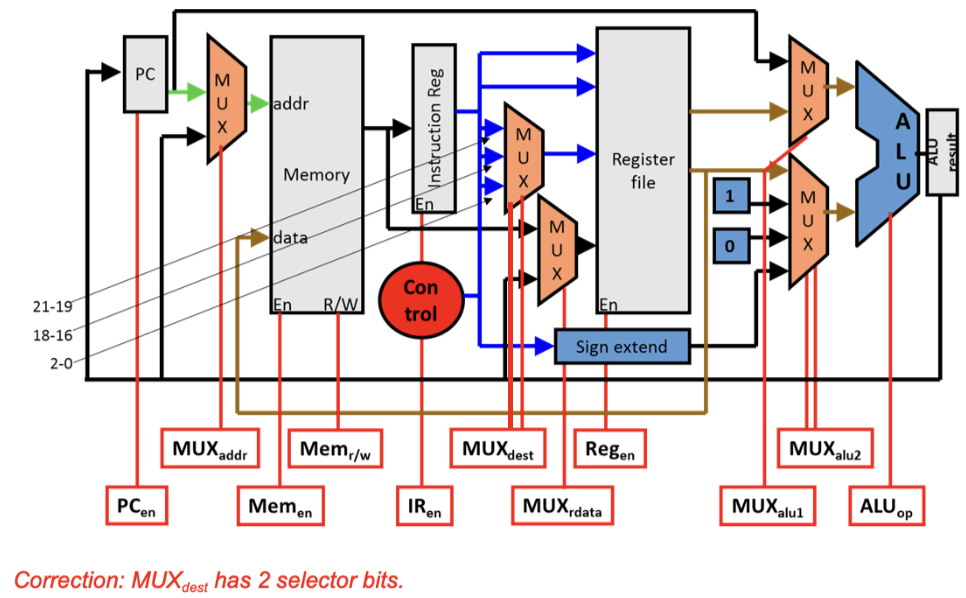
In a moment of clarity, the EECS 370 staff realizes that the LC2K instruction set is rather limited when compared to its greatest marketplace rival, ARM. In order to add convenience and attract users, the EECS 370 staff decides to implement two new instructions: push and pop. **(L11)**

|  |  |
| --- | --- |
| **push regA regB**  **//MEM[REG[regA]] = REG[regB];**  **// REG[regA]++;** | **pop regA regB**  **// REG[regA]--;**  **// REG[regB] = MEM[REG[regA]];** |

Using the modified multi-cycle LC2K datapath below, fill out the control ROM table for the **push instruction**, using as few cycles as possible. Write “X” if the setting doesn’t matter. Multiplexers’ inputs are numbered 0, 1, 2… where 0 is the topmost input. You are to use as few cycles as possible. You will not be modifying the datapath given. (*Note: for this problem you are not doing anything with the pop instruction*). EDIT: Assume the data register from lecture is "inside" the memory unit, so that it still takes 1 cycle to read data for a load, and 1 separate cycle to write the value into the register.

Notes:

* regA is intended to hold the stack pointer
* regB holds the data to be pushed or popped the stack
* The first two cycles, FETCH and DECODE, have been given to you
* Several entries in the table require multiple control bits
* There may be unused rows in the table
* Recall:  
   0 = Read and 1 = Write for Mem r/w  
   0 = ADD and 1 = NOR for ALUop

****

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cycle** | **PC**  **(en)** | **MUX**  **(addr)** | **Mem**  **(en)** | **Mem**  **(r/w)** | **IR**  **(en)** | **MUX**  **(dest)** | **MUX**  **(rdata)** | **Reg**  **(en)** | **MUX**  **(alu1)** | **MUX**  **(alu2)** | **ALU**  **(op)** |
| **1** | **0** | **0** | **1** | **0** | **1** | **XX** | **X** | **0** | **0** | **01** | **0** |
| **2** | **1** | **X** | **0** | **X** | **0** | **XX** | **X** | **0** | **X** | **XX** | **X** |
| **3** |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |  |  |  |  |  |
| **8** |  |  |  |  |  |  |  |  |  |  |  |

### **Problem** 6**: SC/MC Datapath Performance** [30 points] **(L12)**

Consider the multi-cycle and single-cycle datapath from lecture but with the delays shown for each operation:

Read memory 7 ns

Write memory 9 ns

Read register file 1 ns

Write register file 2 ns

ALU 1 ns

All other operations 0 ns

1. What is the clock period for the single-cycle processor if we only had to support add, halt, beq, lw, and noop? **[5]**

1. What is the clock period for the single-cycle processor if we support all LC2K instructions except jalr? **[6]**
2. If we can decrease the delay for one of the operations by 30%, which operation should it be for our **single-cycle** datapath which supports all LC2K instructions except jalr? What is the clock period after improvement? **[9]**
3. If we can decrease the delay for one of the operations by 20%, which operation should it be for our **multi-cycle** datapath which supports all LC2K instructions except jalr? What is the clock period after improvement? **[10]**