ECS 154A: Homework 4

# Logisim

1. Use the [version from the class Google Drive of Logisim Evolution](https://drive.google.com/file/d/14uDb89VUd5yopoxduOR2lzd34VD3auQH/view?usp=sharing). Other versions may not work correctly.
2. Do not rename the files you receive. If you do so you will automatically fail the tester when you submit.
3. Put your solution for each problem into implementation subcircuit
4. Do not rename the implementation subcircuit anything else. If you do so you will automatically fail the tester when you submit.
5. Do not change the appearance of the implementation subcircuit from what it is set as. Doing so will cause you to automatically fail the tester when you submit.
   1. That is this field right here 
6. Do not move the pins inside of the implementation subcircuit as that affects the appearance of the circuit on the outside as you saw in discussion. Doing so will cause you to automatically fail the tester when you submit.
   1. If you want to “move the pins” instead connect tunnels to the pins and move the tunnels around.
7. Do not name any of the subcirucits in your solution main. Doing so will cause you to automatically fail the tester when you submit.
8. You **can** create as many other subcircuits as you want in your solution. Just make sure your solution ends up in the implementation subcircuit

# 

# Restrictions

For all problems in this homework, you may only use

* All of the components under Wiring
* All of the components under Gates **EXCEPT** for Controlled Buffer, Controlled Inverter, PLA
* All of the components under Plexers
* All of the components under Arithmetic
* All of the components under Memory **EXCEPT** for **RAM, ROM,** and **Random** Generator

Unless a problem specifies otherwise.

You have been provided a Register File circuit in the starting circuit. The **only outputs** of the register file that you are allowed to use are A\_Out and B\_Out. The rest are for testing purposes and **should not be used.** Using them will result in a 50% penalty in your grade.

# Problem 1: CPU.circ (100 points)

Build a 4-bit single cycle CPU that can implement the given instructions.

## Instruction Format

Our CPU will be using fixed length instructions. Our CPU will also have two types of instruction formats: R-type and I-type. In R-type instructions both operands come from registers. In I-type instructions, the first operand comes from a register and the second will be contained within the instruction.

### R-Type

| Name | Bits | Description |
| --- | --- | --- |
| OpCode | 15 - 12 | Determines what operation should be performed |
| C | 11 - 8 | The destination register. The C in RegC = RegA OP RegB |
| A | 7 - 4 | The first source register. The A in RegC = RegA OP RegB |
| B | 3 - 0 | The second source register. The B in RegC = RegA OP RegB |

### 

### I-Type

| Name | Bits | Description |
| --- | --- | --- |
| OpCode | 15 - 12 | Determines what operation should be performed |
| C | 11 - 8 | The destination register. The C in RegC = RegA OP Imm |
| A | 7 - 4 | The first source register. The A in RegC = RegA OP Imm |
| Immediate | 3 - 0 | The second source register. The Imm in RegC = RegA OP Imm |

## 

## Instructions

| Operation | Encoding (The value in the OpCodeField) | Description |
| --- | --- | --- |
| STOP | 0000 | The CPU ceases execution |
| NOP | 0001 | Do nothing |
| LOAD | 0010 | RegC = Immediate |
| MOVE | 0011 | RegC = RegA |
| ANDR | 0100 | RegC = RegA AND RegB |
| ANDI | 0101 | RegC = RegA AND Immediate |
| ORR | 0110 | RegC = RegA OR RegB |
| ORI | 0111 | RegC = RegA OR Immediate |
| XORR | 1000 | RegC = RegA XOR RegB |
| XORI | 1001 | RegC = RegA XOR Immediate |
| NOT | 1010 | RegC = NOT RegA |
| NEGATE | 1011 | RegC = -RegA |
| ADDR | 1100 | RegC = RegA + RegB |
| ADDI | 1101 | RegC = RegA +Immediate |
| SUBR | 1110 | RegC = RegA - RegB |
| SUBI | 1111 | RegC = RegA - Immediate |

## 

## Inputs

| Pin | Size (in bits) | Explanation |
| --- | --- | --- |
| Instruction | 16 | The instruction located at Instruction\_Address |
| ClkIn | 1 | The Clock. Connect this to the clock ports of your registers/flip-flops. Do nothing else with this. |

## Outputs

| Pin | Size (in bits) | Explanation |
| --- | --- | --- |
| Instruction\_Address\_Out | 5 | The address of the instruction you want to execute |
| Reg0-15 | 4 | The values in the register file. This has already been connected for you |

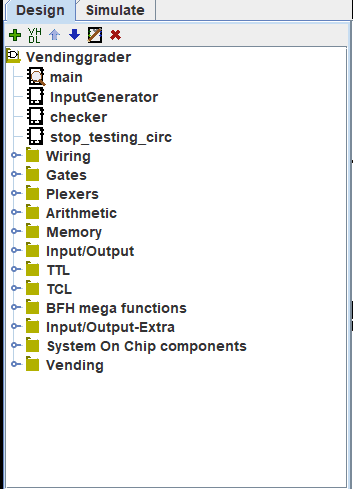
## CPU Components

Your CPU should have

* A Program Counter (PC)
  + This stores and keeps track of what instruction you are on
* Instruction Decoder
  + This is a bunch of combinational logic that sets the control signals inside of your CPU
* Register File
  + A bunch of registers as well as ways to specify which ones you want. This has already been created for you.
  + The **only outputs** of the register file that you are allowed to use are A\_Out and B\_Out. The rest are for testing purposes and **should not be used.** Using them will result in a 50% penalty in your grade.

# Testing

Testing for this problem is different than for previous assignments but is more similar to sequential circuits than combinational circuits. After you finish building your circuit and are ready to test it

1. Open the associated grader circuit
2. Scroll down on the left and side until you find your circuit. Right-click on it and select Reload Library
   1. 
3. Use cntrl+t to tick the clock and check that the outputs of the registers are what they are supposed to be as based on the test program below.

## 

## The Test Program

| Instruction | Meaning | Result |
| --- | --- | --- |
| LOAD REG0, 3 | Reg0 = 3 | Reg0 = **3** |
| LOAD REG1,6 | Reg1 = 6 | Reg1 = **6** |
| NOP | Do Nothing | No Change |
| MOVE REG2, Reg1 | Reg2 = Reg1 | Reg2 = **6** |
| ANDR REG3, REG0, REG1 | REG3 = REG0 AND REG1 | REG3 = 3 & 6 = **2** |
| ANDI REG4, REG3,3 | REG4 = REG3 AND 3 | REG4 = 2 & 3= **2** |
| ORR REG5, Reg2, REG0 | REG5 = Reg2 OR REG0 | REG5 = 6 |3 = **7** |
| ORI REG6, Reg3, 12 | REG6 = Reg3 OR 12 | REG6 = 2 OR 12 = **14** |
| XORR REG7, Reg2, REG0 | REG7 = Reg2 XOR REG0 | REG7 = 6 ^ 3 = **5** |
| XORI REG8, Reg6, 15 | REG8 = Reg6 XOR 15 | REG8 =14 ^15 = **1** |
| NEG REG9, REG3 | REG9 = -REG3 | REG9 = **-2** |
| ADDR REG10, Reg7, REG7 | REG10 = Reg7 + REG7 | REG10 = 5 + 5 = **10** |
| ADDI REG11, Reg1, 3 | REG11 = Reg1 + 3 | REG11 = 6 + 3 = **9** |
| SUBR REG12, Reg6, REG2 | REG12 = Reg6 - REG2 | REG12 = 14 - 6 = **8** |
| SUBI REG13, Reg8, 5 | REG13 = Reg8 - 5 | REG13 = 1 - 5 = **-4** |
| NOT REG15, REG8 | REG15 = NOT REG8 | REG15 = ~1 = **14** |
| ANDR REG1, REG12, REG13 | REG1 = REG12 AND REG13 | REG1 = 8 & 12 = **8** |
| NEG REG5, REG5 | REG5 = -REG5 | REG5 = **-7** |
| NOP | Do Nothing | No Change |
| ADDR REG14, REG5, REG2 | REG14 = Reg5 + REG2 | REG14 = -7 + 6 = **-1** |
| XORR REG7, REG7, REG14 | REG7 = Reg7 ^ REG14 | REG7 = 5 ^ -1 = **10** |
| ORI REG2, REG2, 3 | REG2 = Reg2 | 3 | REG2 = 6 | 3 = **7** |
| SUBI REG13, REG13, 12 | REG13 = REG13 - 12 | REG13 = 12 - 12 = **0** |
| STOP | CPU should cease execution | CPU should cease execution |
| LOAD REG15, 13 | This line should not be executed because the CPU should have STOPped already | This line should not be executed because the CPU should have STOPped already |

The test program is just an **example program.** Your CPU should function on **any program** given to it.

# Making Fixes to Your Solution

After you make changes to your solution you will need to reload your circuit in the grader circuit. If you don’t it won’t see the updates. To reload your circuit, select your circuit in the grader, right-click it and select Reload Library.

# Submitting

## Submit to

Logisim Homework 3 on GradeScope.

## What to Submit

Submit a zip file that contains the following .circ files

1. CPU.circ

Inside of each .circ file leave a comment with you and your partner’s names.

Make sure that you submit a zip that contains the files and **NOT** the **folder containing the files.** Check out the animation below for what to submit.

