### Assignment Instructions

Convert the circuits of Project #3 into subcircuits in Logisim.

Then construct in Logisim the steps of the von Neumann model with your subcircuits. Verify that each step functions as you intended.

Write a Word document that describes your subcircuits and how your design creates each step of a von Neumann model.

Submit all of your .cir files, including the subcircuits, along with the Word document in a zip file named p4-LastnameFirstname.zip.

EXTRA CREDIT (Up to 50 points): Tie the steps of the von Neumann model into a complete circuit. Verify that it functions as a von Neumann machine.

### Subcircuits Created

* Main
  + Contains all the created subcircuits and applies them as a complete computer.
  + Computer is started then used with some initiation steps
    - Enable then disable INITIAL INST ADDR > PP input which primes all initial memory with data for use
    - Enable Switch input to turn on the Control Unit in preparation for clock cycling
    - Either manually clock cycle using the Manual Clock input (click once, it’s a button) or Select the Simulate dropdown in the top of the window and activate Tick Enabled. This will allow the automatic clock to cycle through by itself. The computer will cycle through the steps of the Von Neumann Model (below) at this point.
    - Once done with instructions Disable the clock and Switch inputs (either through the Simulate menu or by toggling the inputs off in the circuit)
    - Clear the initial memory with the Clear Ctr input (this does not clear the general purpose registry at this time).
* 8bit Registry [1:1]
  + A 1 address, 8bit addressable register.
* 16bit Registry [1:1]
  + A 1 address, 16bit addressable register.
* 16bit Registry Selector
  + A 1bin in, 16bit out decoder used to pause data transfer at certain points.
* Initial Instruction Address
  + A 1 address, 16bit addressable register initially used to prime the Program Pointer in Main.
* Incrementer
  + A circuit that receives 16bits, and increments them by one.
  + Allows input for data incrementation or the use of a data loop that will continually increment the same data.
  + Has three steps
    - Mem Load: Loads data from input into the embedded temp memory.
    - Load: Loads embedded memory into incrementer.
    - Increment: Increments the data and outputs it to either Program Pointer or back to embedded memory. The latter was for testing. Typically this will always get its data from the Program Pointer.
* Temp OR (Unused)
  + A 16bit OR logic gate that was originally for routing data back up to earlier registries, but was not used in the end.
* Memory
  + A Decoded 2^16bit memory, 16bit addressable register with five instructions card-wired in, to simulate external memory. I’ve grounded the decoder for any addresses beyond four bits as that would be 2^16 address spaces (65,536 address spaces in total. Not doing that right now...)
    - LDR 1: Tells to load the value from memory registry 3 (DATA 10) into registry 1 in the General Purpose Registry.
    - LDR 2: Tells to load the value from memory registry 4 (DATA 8) into registry 2 in the General Purpose Registry.
    - ADD: Will be used to add the data from the General Purpose Registries 1 and 2.
    - DATA 10: 16bit data containing the value of ten.
    - DATA 8: 16bit data containing the value of eight.
    - Continued with space for additional instructions.
* Memory Data (Unused)
  + A 16bit memory, 16bit addressable registry with the 16bit Registry Selector applied. This allows the data of each address space to only output if the write enabling is active.
* General Purpose Registries
  + A 2^3bit address, 16bit addressable general purpose register used for and operations or temporary data storage for the computer. This is where data pulled from external memory will be stored for use.
* Program Pointer (PP)
  + The Program Pointer, also called Program Counter, will store the next instruction to be sent to the Memory Address Registry. This is a 1 address, 16bit addressable registry that requires Write Enabling for overwriting and also includes a Read Enabling input in place of the selector decoder. This enabler is used to create a step-by-step data transfer with the next circuit activating it when it needs data.
* Memory Address Registry (MAR)
  + The circuit for the Memory Address Registry is the same as the PP circuit, and receives its data from the PP before the data stored in the MAR is used for external memory access.
* Memory Data Registry (MDR)
  + The circuit for the Memory Data Register is the same as the PP and MAR, save the Read Enable input. If data is stored in the MDR it is automatically outputted to the next step.
* Instruction Registry (IR)
  + The Instruction Registry is where the current instructions to be processed are kept. This circuit is the same as the MDR circuit and will automatically send its data to the next step.
* Instruction Decoder
  + The Instruction Decoder receives data from the IR, then breaks it apart based on standard LC3 instruction layouts. When complete, this will decode any of the 20 instructions and continue through the Von Neumann Model processes as necessary.
* Operation Multiplexer
  + This circuit decodes the 4bit operation opcode so the necessary operation circuit path can be enabled, allowing the processing of multiple types of instructions. To this point, only the LDR instruction is designed. The ADD circuitry is next to be set up.
* Evaluate Address
  + This circuit, which will only be used for certain instructions, creates the new address using the source registry and the 2’s comp offset value. The value is outputted to an embedded memory space that uses a selector to only output data when ready for evaluation.
* 16bit Multiplexer
  + A multiplexer that was used as a substitute for the previously stated Temp OR circuit. This allows the computer to output one 16bit value, or another depending on an activation toggle input. This is used to allow the evaluated address to be sent back up to earlier steps in the process so the necessary data can be retrieved from memory for storage in the General Purpose Registry.
* 16bit Decoder
  + This decoder is used between the MDR and the IR. The purpose is to send the data pulled from the external memory either to the IR if To IR is enabled, or to the General Purpose Registry if To Reg is enabled. This is for when the evaluated address data in memory is being cycled back to the General Purpose Registry and only activates at that point.
* N (Unused)
  + A 1 address, 1bit addressable memory that was created with the intention of using it as the LC3 uses it. To this point it is not in use and is merely a placeholder for further work.
* Z (Unused)
  + A 1 address, 1bit addressable memory that was created with the intention of using it as the LC3 uses it. To this point it is not in use and is merely a placeholder for further work.
* P (Unused)
  + A 1 address, 1bit addressable memory that was created with the intention of using it as the LC3 uses it. To this point it is not in use and is merely a placeholder for further work.
* Control Unit
  + A sequential logic circuit designed to activate various states/steps in the computer’s runtime.
  + A final step is added that leaves all states disabled for testing/clearing or data evaluation. Uses a decoder and embedded memory for functionality.
  + Circuit must be primed before use by cycling the clock once. Then the Activate input may be enabled, and the output cycled by toggling the Clock input on and off. A previous step primes the memory in Main, when using the computer.

### 

### Von Neumann Model Process

Once the computer memory is primed with the INITIAL INST ADDR > PP button, and the control unit enabled with the Switch input, the steps of the Von Neumann Model are as so:

1. Fetch Instructions
   1. Accomplished with the first [0:3] steps of the control unit states.
      1. MAR <- PP (increments PP address)
      2. MEM ADDR <- MAR
      3. MDR <- MEM INST
      4. IR <- MDR
2. Decode
   1. Accomplished automatically when the data is applied to the IR
      1. Instruction Decoder <- IR
3. Evaluate Address
   1. Accomplished during the [4:4] state of the CU.
      1. MAR <- EVAL ADDR (Instruction Decoder)
4. Fetch Operands
   1. Accomplished during the [5:6] states of the CU.
      1. MEM ADDR <- MAR
      2. MDR <- MEM INST/DATA
5. Execute
   1. Skipped as unnecessary for process
6. Store
   1. Accomplished during the [7:7] state of the CU.
      1. GEN REG <- 16bit DEC <- MDR
7. Return to Step 1. FETCH