HW10 – Simplified MIPS Datapath

This assignment helps you gain a deeper understanding of the single-cycle MIPS datapath.

Deliverables: Your completed datapath circuit in a file named user1_user2.circ.

Simplified MIPS

You will implement the simulated circuitry to execute a simplified version of MIPS using Logisim. A starting circuit is provided that executes a single specific instruction, you will add the logic to implement several more. The specs for the modified MIPS are described below:

MEMORY

- The word size will be 12 bits instead of 32 bits, with the following effects:
  - All registers including the PC will hold 12-bit values.
  - Instructions will be 12 bits.
  - Data values in memory will be 12 bits.

- 4 registers will be available for standard program use (labeled t0-t3), plus the PC will be a separate register.

- One black-box RAM unit will be used for the instruction memory cache, and a separate RAM unit for Main Memory.

- RAM will use sequential addresses for each 12-bit value, unlike MIPS which addresses each byte thus using increments of 4 to access each 32-bit value.

  Below is an example block of 6 values taken from RAM starting at address 20 (an arbitrary choice).
  The table on the **left** shows the block in one column with **decimal** values.
  The table on the **right** shows the same memory using the LOGISIM display for a RAM unit, with 2 data fields per row and all values in **hexadecimal** (addresses and data), with leading 0's shown explicitly:

	+0
20	2
21	259
22	20
23	7
24	0
25	33

	+0	+1
014	002	103
016	014	007
018	000	021

  *** Note: you can right click on a RAM unit in Logisim and choose "save image" to save the current state of memory, or "load image" to load a saved state, this is very useful for testing! ***

ARITHMETIC

- The only module you may use under "Arithmetic" in LOGISM is the adder, though you may use more than 1 of them.
- All values will be unsigned, there will simply be no negative numbers in this world.
- Overflow will be ignored in all cases (including subtraction that results in a negative value, the result will simply be treated as a positive value).

- Remember that subtraction can be done easily using the adder by complementing the 2nd input (use a not gate) and setting the carry-in value to 1.
- For branch instructions, perform subtraction and then come up with the logic to test whether the result is 0 and use that information to choose to branch or not.

INSTRUCTIONS

- There will be 2 types of instructions, R and I, with fields as follows

  - R-type

	Op	R1	R2	R3	----
# bits	3	2	2	2	3

  - I-type

	Op	R1	R2	Imm
# bits	3	2	2	5

- The 8 instructions with op codes and arguments will be
  - Op 000: add (r3 = r1+r2)
  - Op 001: sub (r3 = r1 – r2)
  - Op 010: addi (r2 = r1 + imm)
  - Op 011: subi (r2 = r1 – imm)
  - Op 100: beq (PC = PC+imm iff r1 == r2)
  - Op 101: bne (PC = PC+imm iff r1 != r2)
  - Op 110: sw (write r2 to address r1+imm)
  - Op 111: lw (r2 = read data from address r1+imm)

- The 1st two instructions are R-type and the rest I-type, this means an op code that starts with 00 indicates an R-type instruction, and anything else an I-type.
- R-type instructions and I-type instructions use different fields of the instruction itself as the destination register id, make sure you get the correct value for the dest reg based on the op code!
- For the branch instructions, you will not do any shifting of the immediate, as we are putting one 12-bit instruction at each address, not using bytes or anything like that.

PROGRAM COUNTER

- In a normal instruction execution, you will add just 1 to the PC
- For the branch instructions, you will NOT add 1 to the PC before adding the immediate. In each instruction execution you will EITHER add 1 to the PC to continue to the next instruction, OR you will add the immediate, never both. This makes the path slightly simpler.

Your Task

I have provided the file HW10start.circ with an implementation only of the 1st instruction (add, op code 000, r-type). Note there is no control yet except for the registers, the op code is unused, it's just that the default behavior is to add. If an instruction is all 0's, it adds register 0 to register 0 and stores the result in register 0.

FIRST, TEST!

- Make sure you understand the various black-box units used in the datapath. Follow where the bits of the instruction are sent after being split by the bit splitter unit. Click on each unit and check what variables are set in the details of that unit. Why do the register file MUX's have a value set to 12 data bits but no data bits field even exists in the decoder?
- You can start right away by putting a value of 1 in register 0 and then tick the clock several times. You will see the PC incrementing, the instruction being executed highlighted in instruction memory (always just 000 for now), and that the value in register 0 doubles each full clock cycle.
- Now try setting the PC back to 0 and adding some different add instructions to the first few memory slots. Remember that the memory is displayed in hex, so the instruction for "t3 = t0+t1" (in binary 000 00 01 11 xxx) would be 038 (assuming don't-care last 3 bits are 0). Tick the clock and make sure your instructions execute the way you think they should.

ADD THE REST OF THE INSTRUCTION IMPLEMENTATION

- Don't forget to include control values such as read/write enables; they are currently unconnected as they are not needed for just the single instruction.
- If you need more area in Logisim for your circuit, take any component and drag it down and right off the current Logisim screen then let go to place it there. Logisim will add scroll bars so you can access the larger space. You can do this as much as you want to get more and more room.

INSTRUCTIONS TO ADD

- Sub: you will only need to add some control based on the op code
- Addi/subi: you will need to implement the path for basic I-type instructions, remember the destination register is taken from a different field in the instruction than it was for r-type instructions.
    - Since you only have to deal with unsigned values, you can extend the 5-bit immediate into a 12-bit value by concatenating with a 7-bit 0 value. You can use the Splitter in reverse to concatenate. Change the bit width in to 12, fan out to 2, and then you can change the individual bits to come from the top or bottom. Use a 7-bit pin as input for the 7 most significant bits into the Splitter, and your 5-bit immediate as the least significant bit input.
- Branch: you will need to modify the PC logic to add the option of changing the PC address to the branch address based on the equality or inequality of the inputs. As discussed above, use the adder to perform subtraction and then logic to decide the input to PC. Also remember do NOT add 1 to the PC before adding the branch value.
- Store/load: you will need to use the main memory unit included on the start datapath but not yet connected. It takes one address, and a separate read data and write data.