ICSI404 – Assignment 6 – build on the computer

This assignment builds on the previous assignment.

Part 1 – The PC

Programs are stored in memory. As you know, memory requires an address to know where to fetch from. We need to create a "tracker" in the computer so that it knows where to load the next part of the program from. Create a longword in your computer called "PC" (program counter); it should default to 0. With this complete, we can start loading instructions.

In your fetch() method, read the next longword from memory using the PC as the address. We will need this value later, so make a "currentInstruction" longword and store the value there. Use a rippleAdder to increment the PC by 2 (the size of an instruction, for now).

Part 2 – The Registers

As we have discussed in class, accessing memory is expensive. We want to store our "local variables" of our program in memory that is closer to the CPU and quicker to work with. Of course, that memory is expensive, so we can't have a lot. Create an array of 16 longwords; these will be our registers.

Part 3 – Understand instructions

Our instructions will (eventually) look like this:

add R1 R2 R3 // Add R1 to R2 and put the result in R3

This breaks down:

4 bits for the instruction (add) (1110 – this comes from our ALU project)

4 bits for each of the registers (1,2,3). A total of 16 bits (2 bytes – note the increment in PC in part 1). In order, these would be: 0001 0010 0011

Our whole instruction in bits is: 1110 0001 0010 0011

This allows us to add (or perform any of the ALU operations) on the registers.

Part 4 – Decode, Execute and Store

In the decode process, we get the values from the registers and prepare for the ALU. This seems odd to us as programmers – register[index] is very simple. Why not just put that into the execute function?

Two reasons – one is that it isn't as simple in hardware – it takes time that execute() needs to do the work. The second is that a register can only do one thing at a time and we will be updating the register in store(). Consider an instruction like add R1 R2 R1 (the same as R1 += R2). Store needs to have clear access to R1.

In decode, examine currentInstruction to determine which 2 registers are our source. This will require shifting and masking and is a little complex. Create two new longwords in our CPU (op1 and op2). Copy the data from the registers indicated by the instruction into op1 and op2.

In execute, pass the control bits (the opcode) into the ALU along with op1 and op2. Create a new longword called result and put the result from the ALU into the result longword.

In store, copy the value from the result longword into the register indicated by the instruction.

This code is not testable yet – we have no output and no way to get values INTO the registers. Every operation will return 0 right now.

You must submit buildable .java files for credit.

Rubric	Poor	OK	Good	Great
Comments	None/Excessive (0)	"What" not "Why", few (5)	Some "what" comments or missing some (7)	Anything not obvious has reasoning (10)
Variable/Function naming	Single letters everywhere (0)	Lots of abbreviations (5)	Full words most of the time (8)	Full words, descriptive (10)
PC	None (0)	Exists (7)	Exists and is used in fetch (13)	Exists, is used in fetch and is incremented (20)
Registers	None (0)			Exist (10)
Decode	None (0)		Extracts wrong bits or doesn't copy into OP1/2(10)	Copies both registers into OP1/OP2 (20)
Execute	None (0)		Extracts wrong bits or doesn't call ALU (10)	Extracts the right bits and calls ALU (20)
Store	None (0)		Extracts wrong bits or doesn't copy into register(5)	Copies into register (10)