Architecture name: Definitely not our ISA

**Overall Philosophy:**  
 Our group wanted to make a version of our ISA from project 2 that would utilize a set of instructions that were simpler to realize through hardware

**Specific Goals:**   
 Using the basic ISA from Project 2, our group stuck with the same commands and functions but switched around the format of the machine code to make it more normalized. Specifically, we aimed to reduce the size of the MUX leading into our instruction memory, as before it required a 4 bit MUX, but now only requires 3.

**Instruction list and details:**

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | OP code | Format | Example |
| lw | 000 01 x y | lw Rx, Ry | Rx = M[Ry]  Rx = $0, $1; Ry = $0, $2 |
| sw | 001  x iii | sw Rx, imm | M[imm] = Rx  Rx = $0, $1; imm = [0:7] |
| bgtR0 | 000 1  x ii | bgtR0 Rx, imm | Pc = pc + imm if Rx > $0  Rx = $1, $2; imm = [1:4] |
| bltR0 | 100 1 x  ii | bltR0 Rx, imm | Pc = pc + imm if Rx < $0  Rx = $1, $2; imm = [1:4] |
| shiftL | 010 10 xx | shiftL Rx | Rx << 1  Rx = [$0-$3] |
| shiftR | 010 11 xx | shiftR Rx | Rx >> 1  Rx = [$0-$3] |
| j | 11   iiiii | J Label | PC = imm  Imm = [0:31] |
| add | 100 0 xx y | add Rx, Ry | Rx = Rx + Ry  Rx = [$0-$3]; Ry = [$0&$2] |
| sub | 010 0 xx y | sub Rx, Ry | Rx = Rx + Ry  Rx = [$0-$3]; Ry = [$0&$2] |
| addi | 101 xx ii | addi Rx, imm | Rx = Rx + imm  Rx = [$0-$3]; imm = [-2:1] |
| init | 011  xx ii | init Rx, imm | Rx = imm  Rx = [$0-$3]; imm = [0:3] |

**Register design:**

Our ISA uses registers 0, 1, 2, and 3. None of these registers have special properties.

**Control flow and branch options:**

When branching, to keep the immediate as short as possible, it is only possible to branch forward, and in order to branch backwards a jump must be used instead. The target addresses for our branches is simple adding onto the PC, and for our jump it overwrites our PC value.   
 Branches can jump forward up to 4 commands, and jump can set PC to any value under 32.  
By limiting branches to jumping forward, they were able to skip 4 commands ahead using 1 less bit for the immediate value than we otherwise would have needed.

**Data memory addressing and storing:**

Our load and store words work very differently based on the different need of load ranges. Because our programs only need to store values into relatively low values in memory, our store function places either register 0 or register 1’s value into the memory address of an unsigned 3 bit immediate (0:7)

However, because loading is required on a much wider range of memory, our load function loads an address of memory defined by either register 0 or register 1 into either register 0 or register 2.

**Questions:**

1) The most significant advantages of our ISA is the versatility and range of commands. Our ISA can do almost all simple things and it starts to become limited by its shortage of registers. This shortage caused many problems when attempting to implement level 2 completion of program 2, but the wide spread of commands makes most programming under this ISA very possible / easy. A small limitation that could easily be changed if the requirements needed it to is the 3 bit range in storing memory. This could easily be switched to a register’s value in the event widespread storing was required. Rather than perfecting anything, our ISA is a good do-it-all ISA, with specifications for the needed tasks that make it more effective implemented where possible.

2) Not a lot of time had been spent on minimizing DIC, as the timeframe did not allow for it. However, in project 3 our ISA had some formatting changes made to make our hardware require shorter, or even less, MUX gates. If this project was restarted from the beginning a total overhaul on the ISA would be required. Ours is probably too generalized and can be totally reworked to form an ISA with more specific commands that would require a lower DIC, however our current ISA does not cause too much complication in our hardware, so any additional effort put forward would focus on DIC reduction rather than HW simplification, at least at first.

3)

A: My favorite part of this project was its reliance on previous projects, and being forced to deal with the limitations you set yourself earlier in the semester. This forces you to think outside the box and to do what you can with what you’ve already committed yourself to using. This directly leads into my least favorite part of the project, which is after realizing how many different ways you could approach your ISA design to make the code simpler, DIC lower, and hardware lighter, the time allotted for project 3 does not leave time for reworking a new, significantly better ISA.

B: If I was giving advice to somebody starting ECE 366 today, it would be to keep track of their algorithms from project 1, and to try to implement them with as little unnecessary capabilities as possible, because if you design yourself into a corner during the early stages of project 2, the end of project 2 and project 3 will be much harder than they need to be.

C: Most of the value of this project came from being forced into a situation where you need to make something very specific with a shortage of resources (bits, time) to work with. It forces consideration of what is absolutely necessary and what is not, and emphasizes the logical thinking skills required to do so. On top of this it also is in a team setting, where different members often had to make totally different parts to the same whole, and have them match up perfectly. All told, this project may not be a resume builder by itself, but the problem solving and teamwork skills it forced us to utilize are valuable to any workplace in any field.