

Luke McNeil

Milestone 1

1) Wednesday, September 30, 2020

[10 min]

Talked with team through chat and decided to do a stack architecture.

2) Saturday, October 3, 2020

[2 hours]

Wrote a first draft of the GCD and RELPRIME functions This included making up instructions as I went along. I used instructions presented in Sid's lecture on the stack architecture, as well as borrowing ideas for stack manipulation from Forth.

3) Sunday, October 4, 2020

[1 hour]

Spent some time trying to figure out what actually uses a stack architecture in the real world, and get a better idea of how it is implemented in hardware. Did some googling and wikipedia reading. Watched youtube videos on Forth and Java Bytecode.

4) Monday, October 5, 2020

[2 hours]

Met with team to discuss various requirements for M1. These include coming up with what additional registers we would need (none). We designed the format types O and A. After the meeting and having talked to Sid for some advice I decided that we could have two stacks of registers. One is the main stack which is for computations, and the other is a return address stack. This way nested functions can always remember where they should return to. This also allowed me to describe the function calling conventions. I wrote a code fragment where main calls f1 which calls f2 which calls f3 showing these conventions in use.

5) Tuesday, October 6, 2020

[1 hour]

I converted my RELPRIME and GCD to the table format that is in the design doc. I then refined the descriptions of our format types. I then listed out all of our instructions so far in a table providing their format type, argument if any, and a description. I also then created a table explaining how to convert O types to machine code which is pretty easy.

[1.5 hours]

Met as a group discussed how to convert all instructions to machine code. Figured out the addressing modes we are going to use. I wrote some additional code fragments explaining simple addition and getting input.

This is what we decided on was our plan for Milestone 2

M2 task assignment:

- a) 1st meeting: break instructions into small steps and move data from one register to another, determine single-cycle or multi-cycle

- b) Luke & Austin: RTL Description of each instruction
- c) Jinhao & Yiju : A list of generic components specifications needed for RTL
- d) 2nd meeting: debug and test the processor through Xilinx ISE and fix existing problems

Milestone 2

1) Thursday, Friday October 8-9, 2020

[4 hours]

Created a simulator for our stack language written in Chez Scheme. The simulator can take a program such as our relprime example and then run it, simulating the stack as a list.

2) Tuesday, October 13, 2020

[3.5 hours]

Wrote the RTL descriptions for add, sub, or, dup, swap, drop, and over. While doing this I had to figure out how we would refer to the stack of registers in RTL. The way I decided to do it was with Reg.push(value) which pushes onto the stack a value and Reg.pop() which pops off the top thing on the stack.

I also edited our design document in response to Sid's feedback of M1. This includes making places for a title page, table of contents, executive summary, and additional sections. I then combined the table showing instruction description with the table showing opcode and funct into 1 table. I also replaced exit with halt, an instruction which always jumps to itself.

I then wrote an assembler for our stack language in Chez Scheme. I used several procedures from the simulator to read in the program. I then used a hashtable to match opcodes and functs, and then had to do some annoying stuff with converting a number to binary. This seems like it should have been easier since the number is actually being stored in binary, I just don't know how to get to that.

3) Wednesday, October 14, 2020

[3.5 hours]

I recreated the RTL for the ones I had previously done and added it for all of the instructions. I changed from using Reg.pop() and Reg.push() to treating the stack as an array of size 64 where stack[0] is the top of the stack.

I had to think about what would happen when our stack machine runs out of registers on the stack. I decided there would be 64 registers in the stack and 64 registers in the return address stack. For this round of RTL I just assumed that data would be lost when the user tries to add something to an already full stack. This is not ideal, but it will make the implementation easier. This might change in the future.

I then made a factorial program and put it in implementation/example-programs/fact.asm. I used this to see if limit of 64 registers would be enough. In this example it is. This is because of the fact that factorial(8) is already bigger than a 16 bit number. Using updates to the simulator I found that factorial(8) only had a max-stack-size of 11 and a

max-return-stack-size of 10. This is plenty less than 64. This might not be true though in other examples where the result does not grow to be more than 16 bits so quickly. A good one to try would be fibonacci.

I also refactored the Design document to contain many different sections to make it more readable.

For M3 I will

- a) Get up to speed in Xilinx
- b) Start on datapath
- c) Write tests for small components

Milestone 3

1) Saturday, October 17, 2020

[1.5 hours]

I created a xilinx project and pushed it to git. It has a mux and a test file from the course website.

I added a note in the design doc that specifies what will happen if a programmer tries to use more than 64 registers. (some information will be lost)

2) Sunday, October 18, 2020

[2.5 hours]

We met as a teams and spent a while verifying RTL. This consisted of drawing out what would happen in various parts and seeing if it was what we wanted.

We then started drawing out a datapath on the whiteboard. We ended up just keeping to add things until we basically had a datapath that could work for all of our instructions.

I then started playing around with what our stack of register component would need as input and output and what it should actually do. I started trying to draw out how it would work with a numPush and numPop input as well as 3 write wires. The output was the top of the stack and second from the top. I quickly realized that this was too much. I realized that when doing an add rather than doing two pops and then a push, it is simply a pop and then a replace.

3) Monday, October 19, 2020

[4 hours]

Jinhao and I met with Sid during office hours. This helped a lot in our design of the datapath and register stack. He gave us the inspiration to simplify the register stack into having two inputs: a stackOP, and a w which is write data. The outputs remain the same just being the top and next value. This makes the design of the register stack much simpler.

I met with Jinhao to further discuss the datapath. We drew a neater version of what we had on to the whiteboard. We ended up having to make some design decisions. For one we added another memory block separate from instruction memory. This is similar to what was done in single cycle in the book. Next we decided to push back some functionality into the ALU. We needed for example only to get out the second input given to the ALU. Rather than add a multiplexor to

the first input giving an option for it just to be zero and telling the ALU to add, we decided that we could just make a special OP to give to the ALU to just give back the second input.

I then implemented the register stack in Verilog. Most of the time spent here was figuring out how to get everything set up, not necessarily working on the logic. Once I got to the logic part it was pretty easy to just use for loops to shift the stack up or down. One thing I need to think about is when the write is done. Right now I have it set to do it on the posedge. I think we might want to switch that so that the ALU can read from the register stack, put a result on w, then on the negative edge of the clock w will be put on top of the stack. From my first implementation I had the for loop iteration for pushing and popping backwards which meant that I was overriding some data. Testing helped me find this issue.

3) Tuesday, October 20, 2020

[2 hours]

I spent a lot of time messing with the register stack to make it faster and smaller. It seems to be making a bunch of flip-flops for some reason that has to do with me doing something wrong. I'm not really sure what that is, but I will probably return to this component later to make it more efficient.

I then implemented the merger component which was a verilog one-liner.

4) Wednesday, October 21, 2020

[2 hours]

We met as a group and ironed out the component list. I then added a unit testing plan description for each component. Jinhao and I then created several integration plans for testing to be done going forward.

Things for Milestone 4

- Continue implementing necessary components and adding unit tests
 - Jinhao - sign-extender
 - Austin - adder
 - Jinhao - left-shifter
 - Luke - register
 - Someone - memory blocks
- Start on integration testing plans
 - Group Meetings (might split out individual work later)
- Implement Control
 - Luke

#####