Dec 30, 2023

Produced this pseudocode from ChatGPT as a starting point to consider required core instructions:

```
LOAD num1   ; Load the first number into the accumulator
STORE dividend ; Store it as the dividend
LOAD num2   ; Load the second number into the accumulator
STORE divisor ; Store it as the divisor

LOOP:
  COMPARE 0   ; Compare if the divisor is equal to 0
  JUMP_IF_ZERO END   ; If it is, end the loop

  COMPARE dividend  ; Compare the dividend and divisor
  BRANCH_IF_NOT_EQUAL UPDATE_DIVIDEND ; If they are not equal, update the dividend

  HALT  ; If they are equal, halt (numbers are not relatively prime)

UPDATE_DIVIDEND:
  STORE temp      ; Store the divisor as a temporary value
  LOAD dividend   ; Load the value of the dividend into the accumulator
  STORE divisor   ; Store the value of the divisor as the new dividend
  LOAD temp       ; Load the temporary value (original divisor) into the accumulator
  DIVIDE          ; Divide the new dividend by the original divisor
  STORE remainder  ; Store the remainder as the new divisor
  JUMP LOOP       ; Repeat the loop

END:
  HALT   ; End of the program (numbers are relatively prime)
```

I made a google document, a shared folder and google form file to work with my team to produce our core instructions. Jada was traveling and so unable to meet during this. Dan had made the Git group that we joined. Dan, Thomas, and I together developed our instruction set and formatted it to appear like the green sheet we use in class. We met for about 90 minutes.

Notes: “sr” is the most unique register we use, its syntax is critical when writing code to make sure the program is understanding user intentions

Jan 08, 2024

I deleted the starting pseudocode from earlier, and I wrote the assembly code for relPrime using our instructions. Thomas and I met right after class for about 80 minutes. Thomas had already updated and formatted the document into Git. We reviewed our work and added our journals and the design document as PDFs in our Git repo. I updated the design document’s descriptions, and added a new value to sr for “always true” booleans.

Jan 11, 2024

I wrote the RTL’s for 7 of the instructions while leaving indices as “X”. We had our milestone 1 meeting, and then I started and completed the addressing modes. I transcribed everything from our .md “green sheet” to our design document.

Jan 12, 2024

I corrected and completed the RTLs. I started working on the generic components. I filled out the columns for register, ALU, immediate generator, memory and control. I also helped Jada with the memory map.

Jan 14, 2024

We met for 50 minutes as a group to review RTLs, each other’s individual work, discuss design changes, and delegate next tasks. We changed the status register to be 2 bits, so I went back and made all the appropriate changes with that, and then worked on finishing the table for Generic components. I also wrote the initial naming conventions and had the rest of the team approve it.

Week of Jan 15 - Every day was similar work

I confirmed with the team that we wanted a multicycle datapath. Then I started to update RTL's to be multi-cycle. I set the fetch and decode cycles to be the same for all instructions. At first, I organized ADD with ADDI, but looking at the single cycle RTLs, I decided it made more sense to load immediate vs load memory separately. So then I organized the RTLs by loading immediate values vs loading memory values. Following the structure from in class example, I made loading reggie its own cycle, and then loading special registers. Since each register is a distinct hardware piece, each can be acted on concurrently with reggie, and thus all belongs in a single cycle.

Looking at it made me think about the datapath a lot more so I started drawing that while writing the RTLs. Since we're using an accumulator with only 1 register being user accessible, it didn't really make sense to have a register file, and I added control signals as I realized they were needed. The most unique aspect of our datapath was how the ALU is set up. I prioritized avoiding a second ALU because that would incur the highest cost.

I used MUX's for each ALU input since they're a faster piece of hardware and will help with input selection. I went instruction by instruction to add each input to the MUX source A or MUX source B and then double checked to make sure all those ALU operations were doable. (eg. since Reggie's value is added to value in memory, those 2 values must come from separate MUX's)

I also added control signals and formatted that into a table, and met with Thomas on the 21st to review and verify the final datapath.

I spent more time later on developing and re-formatting the control signals and other information in the CTRL/MUX operation tables.

Jan 25-28

After our M3 meeting, I corrected the multicycle table. In Verilog, I wrote the initialization for memory, the ALU, and the control units. We delegated components for M4, and I worked on the ALU and control.

Afterwards, I debugged both those until they compiled correctly, and then I wrote generic test benches for both units. The way the test bench is written makes it easy (for me) to update and add more later depending on what I'm trying to look at specifically.

Jan 29

Tasks were reallocated again so Jada was tasked with the Finite State Machine. I helped Jada on the FSM by writing the states/flow based on the control unit I had already written.

Jan 30

I added the specifications for Control and ALU and test bench descriptions for both.

Feb 1

Since Jada never actually ended up making more progress on the FSM, I decided to just restart the work I had originally given her and remake it from scratch. I did this to also proof-check the rest of my work with datapath, RTLs, and control. While making it, I realized we were missing the following pieces in our datapath:

1. PC needed a mux and 2 bit src since its input could come from either ALU, Memory or Immediate.
2. IorD needed 2 more inputs for MDR and Immediate. Ideally, this means "IorD" should be renamed to just "MemSrc" but renaming this would cause other issues. This also requires IorD to be a 2 bit input instead of 1.
3. Since we're using the same ALU to increment PC, AluSrcA needs PC as a potential input.

I copied and pasted the above into the design doc to account for changes to datapath. I then updated the visual and notified my groupmates.

After completing the FSM, I started to work on rewriting the Control unit to reflect a FSM rather than a truth table.

Feb 3-4

The Control unit took much longer than expected.

More changes:

- I added a Mux for ALUout because in the multi-cycle RTLs, I wrote it so ALUout would be loaded with data from IR in case it was a branch instruction. In the case of a branch instruction, this would be faster and save it a clock cycle. However, in order to do this, I needed a MUX to allow ALUOut register to have more than 1 possible input. This meant it also needed an additional control bit. Since it is only reading form the IR in the decode cycle, I added the control bit: isDecode.
- I added 2 control bits to Reggie for "SLLI" and "LLI" so that logic can just happen in Reggie rather wait on the ALU to operate.

I also realized I had more questions regarding the multicycle implementation.
For comparison checks, I wasn't sure if that comparison was happening in the ALU or just as a line of code written in Control.v in Verilog. I decided to save this question for Dr. Williamson later.

I also felt like my understanding was fuzzy regarding how the clock cycles worked with the control and datapath.

On Feb 4, Thomas and I met to work on the datapath implementation. During this time, I also added implementations of components for Comp Code register, mux, and a unique register for Reggie (named Accumulator Register in Verilog). We took turns on writing on the Datapath.v file itself.

For the mux, it takes in 4 possible inputs, but for those muxes that use less than 4 possible options, the extra inputs default to zero.

I then implemented and added IR component as well. If it's not in the fetch cycle, its value is nonsense.

Throughout this period, I constantly updated the datapath visual and the design doc to track the changes and additional controls.

Feb 5-6

Most of the components were missing output connections in the Datapath so I added these, most especially for muxes and registers. I instantiated the CompCode as well in Datapath, and made corrections to some. I added the additional mux for aluoutsrc (with the isDecode control).

I realized a couple things on the datapath were also just wrong, like an AND gate for a write into output register, and outputting a wire that wasn't necessary from control (it was outputting the address read from IR). So I corrected these mistakes as well.

Feb 8

I worked on compiling and cleaning up errors--mostly syntactical with commas vs semicolons.

Feb 12

I helped Jada debug her work for an hour immediately after class. I helped her debug around 6:30 PM when our group met as well. I wrote StagedDatapath.v, instantiated all the stages, and the input and output connections for each. During this, I noticed some mistakes made in the wiring for Stages 1 and 3, and so I updated those stages. Stage 3 was missing outputs completely, and Stage 1 was missing inputs for the mux IorD. Our group met during this time for 4.5 hours.

I also went to EIT this day to fix my ModelSim. They said they tried every resource available to them and was unable to resolve the issue. So I continued compiling in Quartus.

Feb 13

I compiled everything from the previous day. It spat out 10 errors and so I then cleaned those up. Most of them were syntax issues from Stage 3, and a few others were syntax/typos from StagedDatapath. During class, we realized an issue with BMEM not having enough bits for the immediate address in addition to the comp code and opcode. I came up with the idea of just making the address immediate RA relative to avoid any significant changes to our design/implementation, and because RA would likely be closer to the target address than PC. I checked with my group mates and asked them for their input/alternative ideas, and we decided as a team to stick with this solution.