Design Journal - Ethan Bennett

M1

1/2/2023 (~1 HR 30 M)

I would've liked to meet sooner, but with a teammate unable to access a computer all the way in Singapore it was quite difficult until now. The primary goal of this meeting was to figure out

A) Register assignments
B) Instruction types and format
C) Processor architecture

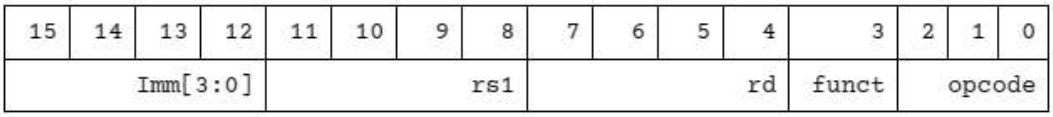
So I guess I'll discuss those next.

First, we decided to do a Load/Store architecture, since we were the most confident in it. Had a lot of ideas for possible instruction formats right off the bat. We briefly considered an accumulator architecture, but some of the instruction types seemed really annoying, so we pivoted to Load/Store.

We agreed that we should probably have 16 registers, since that would give us more than enough. Something to consider is that having excess registers is quite expensive, but for now it's ok. Allocating them was easy enough, many similarities with RISC-V register allocation. Zero register, stack pointer, etc. We have significantly less save/temp/argument registers, though. One of the perks of 16 bit architecture is that having 32 registers is hard since that means any register-register operation is nearly impossible. That's like, 15 of 16 bits dedicated to just registers. Enjoy a one bit opcode. Luckily, we aren't psychopaths, and went with 16 registers. That leaves us room for a 3 bit opcode and a 1 bit funct.

On the subject of opcodes, our next priority was instruction types. Many of these share a lot of similarity with RISC-V, so it wasn't terribly difficult. The design document covers these quite nicely, so I won't bother repeating myself here. One thing we considered and had to scrap was merging the destination and source register in I-Type instructions (to allow for a larger immediate) but this meant we would need a whole new type for lw instructions. I didn't want to give lw the satisfaction of its own type. Thus, we went with a more traditional I-Type, as seen below.

I-Type (Immediate)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm[3:0]				rs1				rd				funct	opcode		

Allows for two immediate operations (adding a register value with an immediate, and loading a value to a register from memory).

Instruction	Opcode	Funct	Example	Meaning	Description
addi	001	0	addi x5, x6, 7	x5 = x6 + 7	R[rd] = R[rs1] + SE(imm)
lw	001	1	lw x5, 0(x6)	Load a value from memory address x6 with an offset of 0 bytes into x5.	R[rd] = M[R[rs1]]+SE(imm)

That's about all there is to say about this meeting.

1/9/2023 (4 HR)

Probably should have met sooner, but once again one of our teammates was still in Singapore until yesterday. Its fine, how hard could finishing the design document be-

4 HOURS LATER

OK, the design document is finished. Definitely will start work on M2 sooner. Geez. Had to do some rethinking on branch instructions, and realized we could just implement bne using beq, so we replaced bne with bgt. That's all really.

M2

Not a particularly interesting milestone to talk about in terms of big decisions. We cleaned up the design document a bit, and drew up the RDL. My main contribution was all of the component specifications - when doing that part, we had not yet considered multicycle, but that is something we should absolutely include since we are measuring performance based on execution time. We will need to add a few registers, but it shouldn't be excessively hard.

M3
The main concerns with this milestone were how to best transition to multicycle. Honestly? It's not that bad. We essentially just added a few registers in between the major components. The

RTL for the processor changed as a result, but nothing was unmanageable. We also drew up some prospective data paths, but that aspect is not yet finalized.