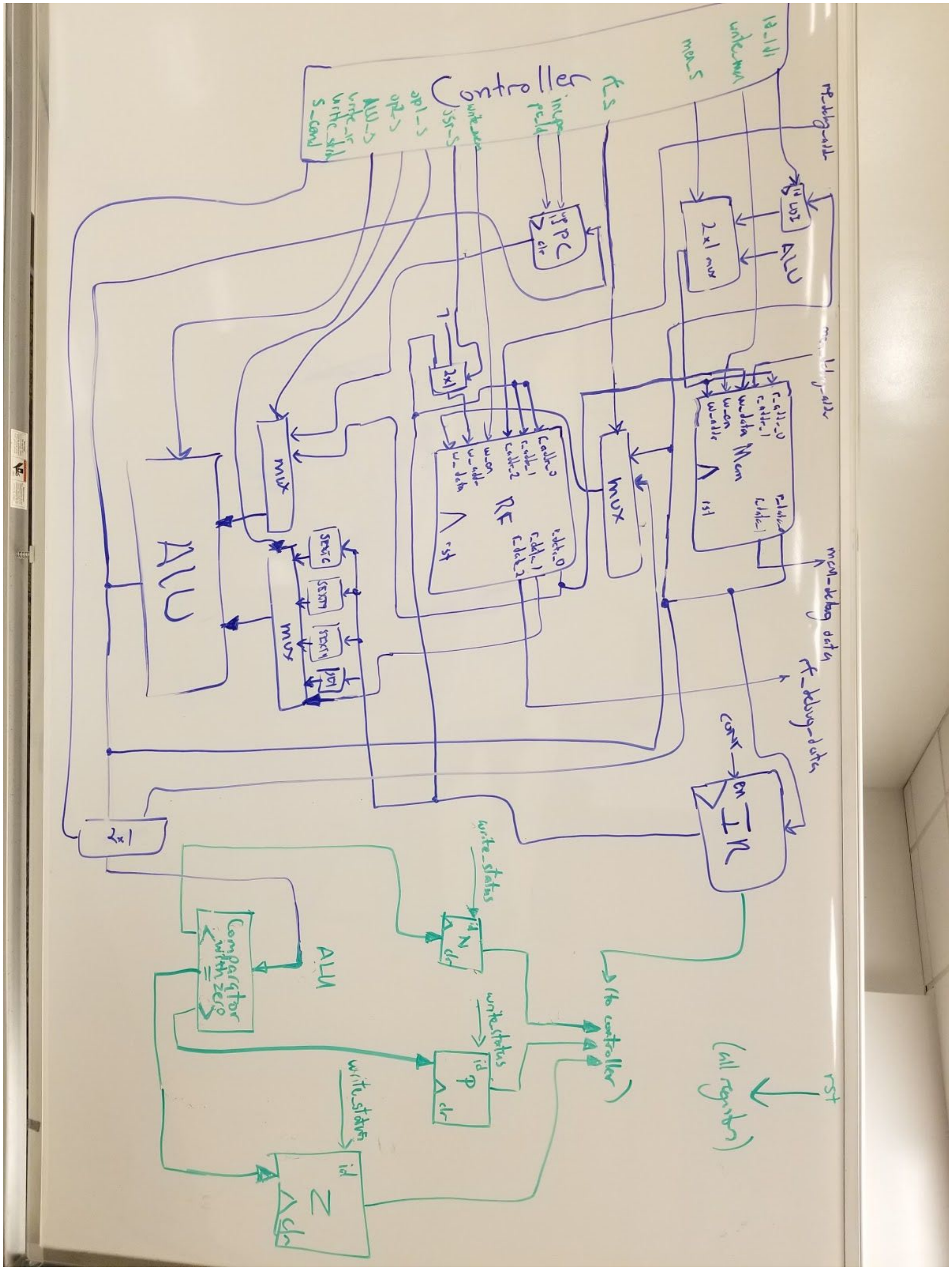
Anthony Hein
Andrew Paul
ELE 206
Final Project -- PUnC
1/11/2020

Datapath:

Unfortunately, our datapath design was drawn on a whiteboard that was since erased. We only had to make minimal changes to this design in the second checkpoint, and none in the third checkpoint. These changes are enumerated below. Our original design is included as an image below.

- A 2x1 mux was added in front of the r_addr_0 port of the register file, it muxed between bits [11:9] of the IR (this is the SR in store instructions) and bits [8:6] of the IR (this is the SR1/BaseR/SR in all other instructions).
- A 2x1 mux was added in front of the r_addr_1 port of the register file, it muxed between bits [8:6] of the IR (this is BaseR in the store instruction in which this register appears) and bits [2:0] of the IR (this is SR2 in all instructions where SR2 appears).
- The output of the IR was sent to the controller (all bits).
- The r_data_1 of the register file was sent to the mux preceding the leftmost operator of the ALU (this is also for the store instruction in which BaseR appears)
- The bitwidths of wires were adjusted appropriately and mux selects added.


Controller
ir_ld
write_mem
mem_s
rf_s
incpc
pc_ld
write_en
jsr_s
op1_s
op2_s
ALU_s
write_ir
write_stat
mem_debug_addr
rf_debug_addr
mem_debug data
rf_debug_data
PC
clr
2x1 mux
ALU
2x1
7
raddr_0
raddr_1
raddr_2
w_en
w_addr
w_data
rdata_0
rdata_1
rdata_2
RF
rst
Mem
mux
SEXT6
SEXT9
SEXT11
IR
cont
en
write_status
N
ld
clr
Comparator with zero
<
=
>
P
Z
(to controller)
rst
(all registers)

Assembly Program:

```
// Computes the nth Fibonacci number, where 0 is the 1st Fibonacci number and n is stored at
location 18 in memory.

// Initialize memory and first 2 fib numbers 0 and 1
/*0:*/ 2011   // LD R0,  #17            Load the counter (or n) into R0
/*1:*/ E210   // LEA R1, #16            Load the address at which data (not instructions) start.
/*2:*/ 14A3   // ADD R2, R2, #3         i = 3
/*3:*/ 4800   // JSR #0                 Link instruction address of start of for loop (w/o branch).

// Start of for loop which adds the most recent numbers in memory to generate next Fib number.
/*4:*/ 96BF   // NOT R3, R2             This line and the next load -i into R3.
/*5:*/ 16E1   // ADD R3, R3, #1
/*6:*/ 16C0   // ADD R3, R3, R0         This line computes counter - i
/*7:*/ 0809   // BRn #9                 Iteration condition for loop (if i<n) stay in loop.

/*8:*/ 1681   // ADD R3, R2, R1         Address of ith Fib number in R3 (start of data + i).
/*9:*/ 18FF   // ADD R4, R3, #-1        Address of (i-1)th Fib number in R4
/*10:*/ 6900   // LDR R4, R4, #0        (i-1)th Fib number in R4
/*11:*/ 1AFE   // ADD R5, R3, #-2       Address of (i-2)th Fib number in R5
/*12:*/ 6B40   // LDR R5, R5, #0        (i-2)th Fib number in R5
/*13:*/ 1D44   // ADD R6 R5, R4         ith Fib = (i-1)th + (i-2)th number in R6
/*14:*/ 7CC0   // STR R6, R3, #0        Store ith Fib number (R6) at address of ith Fib (R3)
/*15:*/ 14A1   // ADD R2, R2, #1        Increment i
/*16:*/ C1C0   // JMP R7                To go back to check iteration condition line 7
// End for loop

/*17:*/ F000   // HALT                  Program end.

// Begin data section (not instructions).
/*18:*/ 0019   // 0019                  Counter (or n) number of Fib numbers to compute
/*19:*/ 0000   // 0000                  1st Fib number.
/*20:*/ 0001   // 0001                  2nd Fib number.
/*21:*/ 0000   // 0000                  (Memory location will be overwritten with 3rd Fib number).
```

This program computes the nth Fibonacci number, where 0 is the 1st Fibonacci number, 1 is the 2nd fibonacci number, etc. The number of Fibonacci numbers to compute, n, is specified in data memory location 18 and can be changed as long as the program is recompiled. Each Fibonacci number is computed by summing the two most recently computed Fibonacci numbers (instead of making this a recursive algorithm, it is linear). Each Fibonacci number computed gets stored in memory, more specifically memory location 18 + i stores the ith Fibonacci number. This is important because these numbers will later be fetched from memory to computer the next

Fibonacci number. At the end of the program, R6 holds the desired nth Fibonacci number and memory 19 through 18 + n store all Fibonacci numbers computed. NOTE, technically, this is only accurate when n is greater than or equal to 2, but this is acceptable because the first two Fibonacci numbers are uninteresting.

Feedback:
Andrew and I spent around 20 hours on this project. The datapath was a daunting task that provided us with the most difficulty, but after that was completed, the project felt very sequential and formulaic. It is amazing to look back and see what this is capable of (it is a complete general purpose processor!). All in all, it is a great way to end this class, and definitely the most rewarding project.