Isaac Ahn
ECE M116C
Prof Nader Sehatbakhsh

ECE M116C CA1 Report

Datapath Description

For my sanity's sake, my datapath simply follows the single-cycle processor that we "developed" in class, with additional components as needed, but also including any possibly unnecessary components.

After instantiating `myCPU`, I created a `regFile` array of 32 `int` objects. Within the `while` loop, there are 7 `bitsets` of varying sizes for the seven possible instruction components: the opcode, read register 1, read register 2, write register, funct3, funct7, and the immediate. Using `for` loops to iterate through the instructions, each of the `bitset` objects received a value, regardless of whether the instruction actually used it or not. The unnecessary `bitsets` would simply not be used.

The `immGen` is a `switch` with different cases based on `opcode`, and the 12 bits of the immediate are gathered and reordered from the instruction `bitset`. The immediate is converted into an `int`.

The controller is also built into a `switch`, with different cases based on `opcode` and `funct3`. The control signals are `bool` or `bitset` values.

In preparation for the next step, output signals from the `regFile` are initialized as `int` objects.

The ALUSrc Mux is an `if/else` statement that sets the second ALU input as the immediate or read register 2 based on the `ALUSrc` control signal.

The ALU Control and ALU are a nested `switch` statement. The `ALUOp` is a `bitset` of two bits: `00` signals an add, `01` signals a subtract, `10` is split into another `switch` case where the funct3 determines whether to $\otimes$, >>, or `&`.

The next two segments are short scripts for JALR and BLT. JALR is an `if` statement dependent on the `jump` control signal. Within the `if` statement, `ulong` objects `jumpPC` (set to the ALU output) and `nextPC` (set to `myCPU.readPC()`) are initialized. For the purposes of control flow instructions, a `CPU::writePC()` function has been created. The `jumpPC` value is written to the `PC` object and the `nextPC` value is written to the write register. The BLT is also an `if` statement, this time the condition being that the ALU output (rs1 – rs2) is negative and the `branch` signal is `true`. If so, the `PC` is set to the current PC plus the immediate minus 4, as the `PC` is incremented by 4 at the fetch stage.

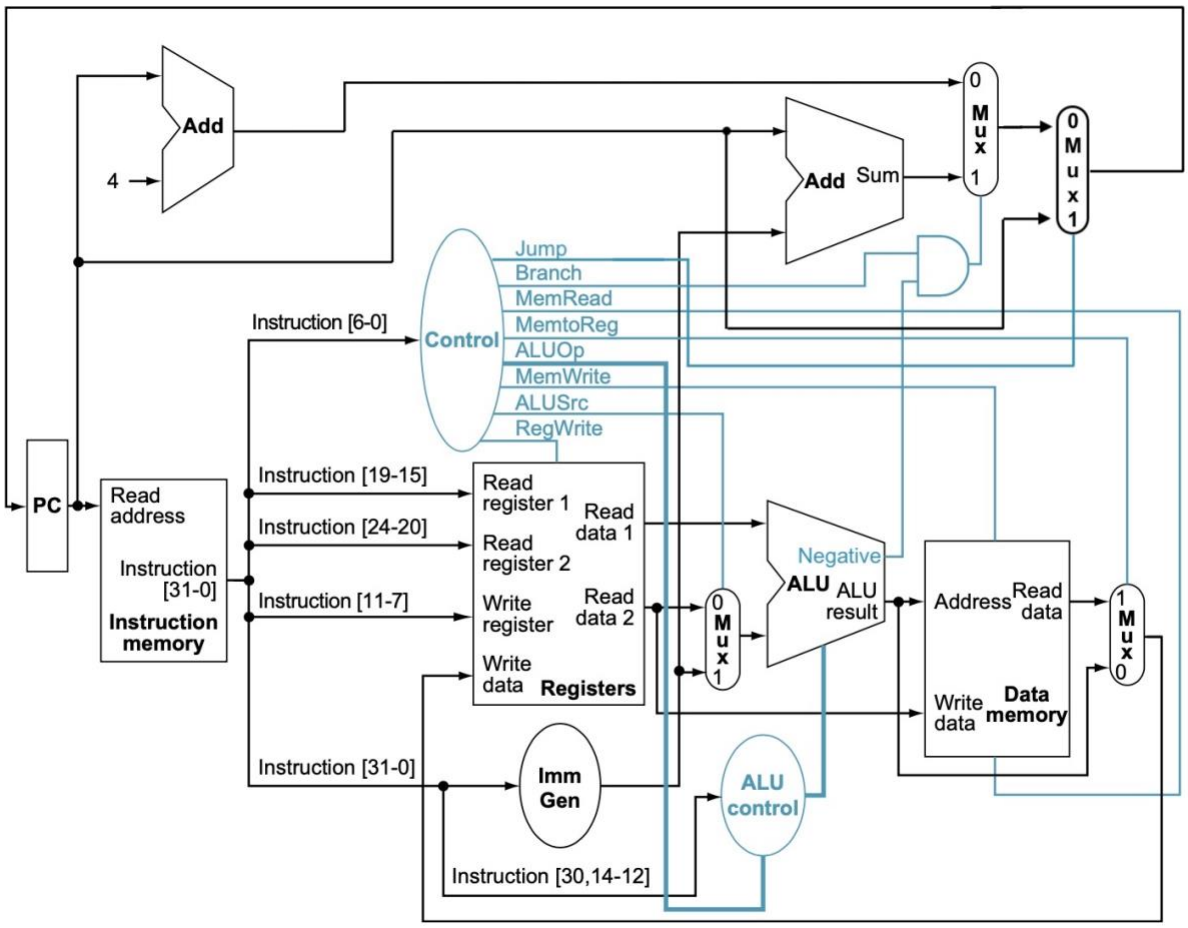
The memory stage begins with the ALU output, which is renamed `addrIn`. A `bitset<32>` `writeDataIn` is set to the bit value of the output register 2. If the `memRead` signal is true, a temporary `int` array of 4 is created and set to the values of `dmemory[addrIn:addrIn+3]`. A `readData` integer is set to the integer value of the properly ordered and concatenated `dmem` elements. If the `memWrite` signal is true, `dmemory[addrIn:addrIn+3]` is written to the value of `writeDataIn` via a `CPU::writeMem` function created in the `cpu.cpp` file.

The writeback stage is an `if/else` statement followed by an `if` statement, the conditions set as the `memToReg` signal and the `regWrite` signal, respectively. If `memToReg` is `true`, an `int` object `writeback` is set to `readData`. If `false`, `writeback` is set to

ALU output. If `regWrite` is `true`, the `regFile` array is modified at the proper index (found by casting the `writeReg` `bitset` to an `int` value).

`a0` is set to `regFile[10]` and `a1` is set to `regFile[11]`.

Datapath Diagram



Control Signals

	branch	memRead	memToReg	aluOp	memWrite	aluSrc	regWrite	jump
ADD	0	0	0	00	0	1	1	0
SUB	0	0	0	01	0	1	1	0
ADDI	0	0	0	00	0	1	1	0
XOR	0	0	0	10	0	1	1	0
ANDI	0	0	0	10	0	1	1	0
SRA	0	0	0	10	0	1	1	0
LW	0	1	1	00	0	1	1	0
SW	0	0	0	00	1	1	0	0
BLT	1	0	0	01	0	0	0	0
JALR	0	0	0	00	0	1	1	1

Questions

1. What is the total number of cycles for running “all” trace (ZERO instruction included)?

   13 cycles

2. How many r-type instructions does this program (“all”) have?

   2 r-type instructions

3. What is the IPC of this processor (for “all” trace)?

   As a single-cycle processor, the instructions per cycle is 1.