Design of an 8-bit Processor

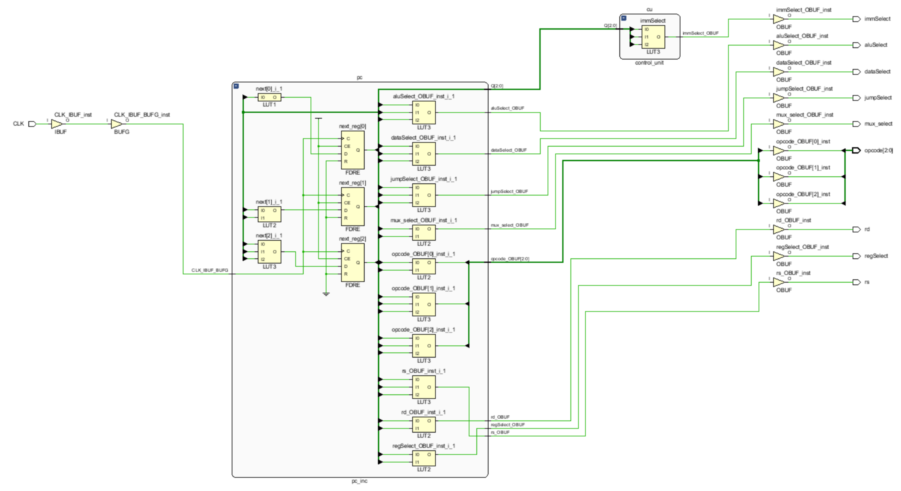
Basu, Bodhiswattwa
Miller, Ben
Both, Jesse
Zhou, Andrew

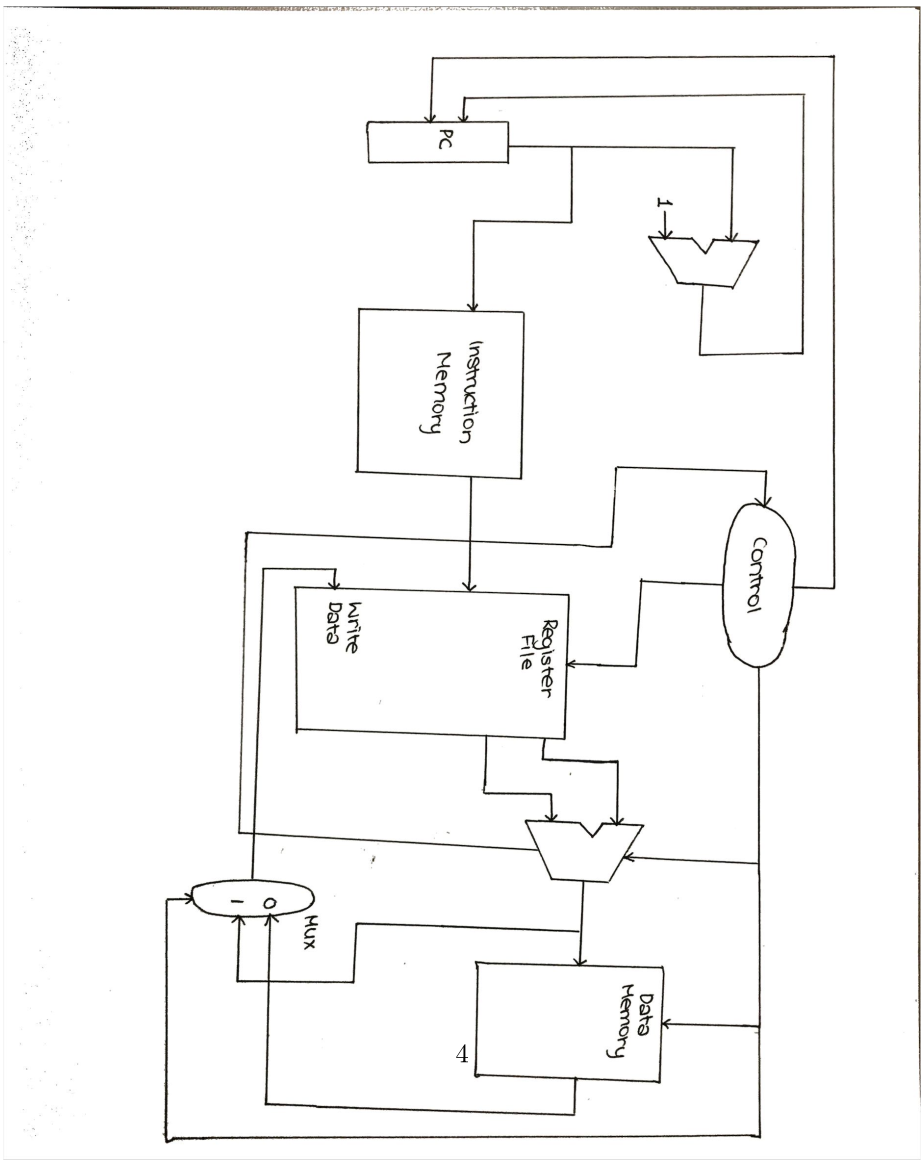
April 7, 2022

Contents

1 Schematics

1.1 Datapath and Control path





2 Components

2.1 Program Counter

Perhaps the simplest component in the entire datapath, the program counter is used to store the address of the current instruction.

At the datapath's initialization, the program counter begins at 0. The address of the first instruction is 1. After each instruction is completed, the program counter is incremented by 1.

```
sw $s0, 0($s1)
lw $s1, 0($s1)
add $s1, $s1, $s1
addi $s0, $s0, 2
j L1
L1: sub $s1, $s1, $s0
```

If this instruction sequence consisted of the entire sequence executed by the datapath, then the `sw` would have an address of 1, `lw` would have an address of 2, etc.

2.2 Instruction Memory

The instruction memory has two primary purposes. First, it is where all instructions are stored. Second, it is used to decode instructions and transmit this information to the other pieces of the datapath.

An instruction can be composed of up to four different parts out of a pool of five: the opcode, a source register `rs`, a destination register `rd`, an three bit immediate value, or a five bit address.

Instruction memory isn't capable of and has no intention of distinguishing what type of instruction (R/I/J) something is. This logic is handled by other components. The instruction memory is only used to receive an instruction and break it apart into information the other components can use more easily.

2.3 Control Unit

The control unit is arguably the most important component in the entire datapath and is responsible for coordinating the remainder of the datapath on a per-instruction basis.

A normal datapath's control unit consists of eight flags determined by the opcode of an instruction to set the control signals other components use for execution. This 8-bit processor uses six. Each control signal is 1-bit, having a value of either 0 or 1.

1. aluSelect - Specifies which operation the ALU will be performing. If 0, the ALU will be doing addition. If 1, the ALU will be doing subtraction.

2. regSelect - Specifies whether or not a register will be written to. If 0, a register will be written to, as in the case of an `add` or a `lw` instruction. If 1, a register will not be written to, as in the case of an `jump` or a `sw` instruction.

3. immSelect - Specifies whether this is an R-type or I-type instruction. Primarily used to differentiate between `add` and `addi`. If 0, the instruction is not an I-type. If 1, the instruction is an I-type.

4. dataSelect - Specifies whether or not data memory will be written to. Essentially used to differentiate `sw` from other I-type instructions. If 0, the instruction will not be writing into data memory. If 1, the instruction will be writing into data memory.

5. muxSelect - Used as a selector bit for a multiplexor. The multiplexor in question is used to choose between the outputs of the ALU and data memory. The output of this multiplexor is what is placed in the destination register of the instruction. If 0, the instruction is an R-type instruction and accordingly the ALU's output should be used. If 1, the instruction is `lw` and the loaded data memory should be used.

6. jumpSelect - Specifies whether the instruction is a J-type or not. If 0, the instruction is not a J-type. If 1, the instruction is a J-type. This information is used by the program counter to determine whether or not it should be moving the program counter to an address or simply incrementing it by 1 like normal.

2.4 Register Bank

The register bank serves two purposes, with the two at different stages of the datapath.

The first instance is where the register bank gets read. This is done by all R-type and I-type instructions. The register bank takes the addresses of the registers that need to be read and stores their values in a format readable by other components.

The second purpose is at the end of the datapath, where the register bank is written into. This is done by all R-type instructions in addition to `lw` and `sw`.

The register bank itself runs on a clock. Depending on where on the clock cycle the program is the register bank performs different actions. On both the positive and negative edge, the register bank reads registers and outputs their values. On exclusively the positive edge, the register bank checks if data needs to be written and writes the data if requested. This is done to ensure data isn't written twice.

2.5 Sign Extend

The sign extension unit serves a simple purpose. The processor itself is an 8-bit processor, but due to this restriction the program can only process 3-bit wide immediate values. The sign extension unit extends these values to 8-bits.

2.6 ALU

The Arithmetic Logic Unit, or ALU, accepts the values of registers and a selector bit. Depending on the selector bit, either addition or subtraction is performed.

2.7 Data Memory

Data memory is similar to instruction memory in that it is a component wherein information is stored. Data memory can be read from and written to, by `lw` or `sw` respectively. A selector bit is used to differentiate between the two instructions.

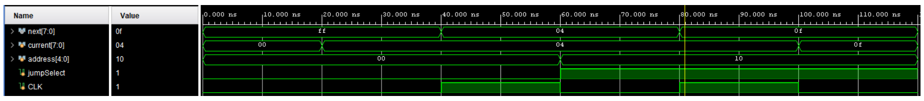
The logic for the two instructions itself is quite simple. If information is being loaded, the program accesses data memory, finds what it needs, and returns it. If information is being stored, the memory location is found and written to.

2.8 Multiplexors

A small device used to select between a combination of inputs. This datapath uses a single mux: a 16 to 8 mux which itself instantiates several 2 to 1 muxes.

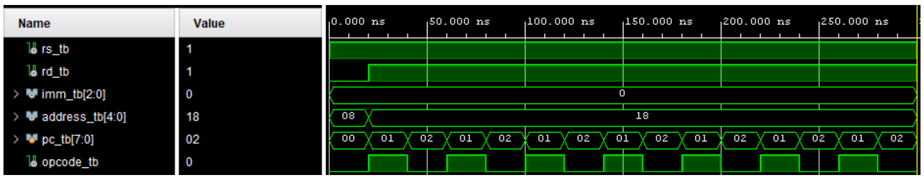
3 Simulation Results

3.1 Program Counter



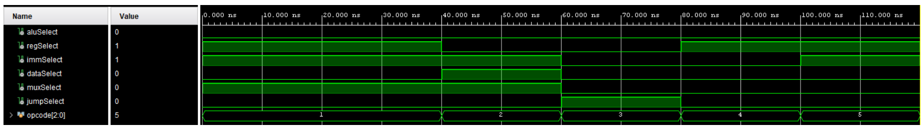
```
Time      next     current     jumpSelect  address     CLK
-----------------------------------------------------------
0000 11111111 00000000 0 00000 0
0014 11111111 00000100 0 00000 0
0028 00000100 00000100 0 00000 1
003c 00000100 00000100 1 10000 0
0050 00001111 00000100 1 10000 1
0064 00001111 00001111 1 10000 0
```

3.2 Instruction Memory



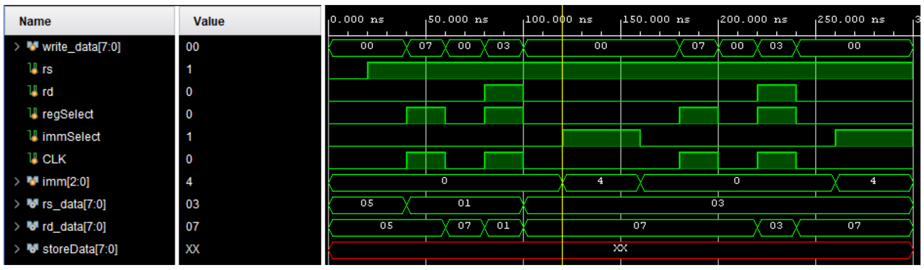
```
Time      opcode   rd  rs  imm      address      PC
-----------------------------------------------------
0000 00000000 00000000 1 0 08 0
0014 00000001 00000001 1 0 18 1
0028 00000000 00000001 1 0 18 2
003c 00000001 00000001 1 0 18 1
0050 00000000 00000001 1 0 18 2
0064 00000001 00000001 1 0 18 1
0078 00000000 00000001 1 0 18 2
008c 00000001 00000001 1 0 18 1
00a0 00000000 00000001 1 0 18 2
00b4 00000001 00000001 1 0 18 1
00c8 00000000 00000001 1 0 18 2
00dc 00000001 00000001 1 0 18 1
00f0 00000000 00000001 1 0 18 2
0104 00000001 00000001 1 0 18 1
0118 00000000 00000001 1 0 18 2
```

3.3 Control Unit



```
Time     aluSelect   regSelect   immSelect   dataSelect  muxSelect   jumpSelect  opcode
----------------------------------------------------
0000  0  1  1  0  1  0  001
0028  0  0  1  1  1  0  010
003c  0  0  0  0  0  1  011
0050  0  1  0  0  0  0  100
0064  0  1  1  0  0  0  101
```

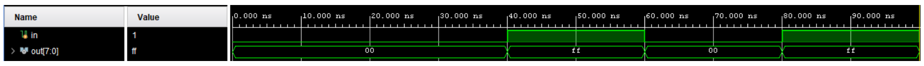
3.4 Register Bank



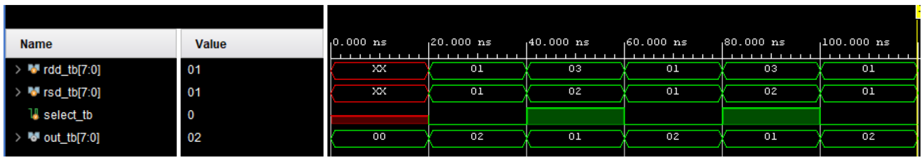
```
Time      write_data  rd   rs   rd_data       rs_data       CLK
----------------------------------------------------
0000  00000000  0  0  00000101  00000101  0
0014  00000000  0  1  00000101  00000101  0
0028  00000111  0  1  00000101  00000001  1
003c  00000000  0  1  00000111  00000001  0
0050  00000011  1  1  00000001  00000001  1
0064  00000000  0  1  00000111  00000011  0
00b4  00000111  0  1  00000111  00000011  1
00c8  00000000  0  1  00000111  00000011  0
00dc  00000011  1  1  00000011  00000011  1
00f0  00000000  0  1  00000111  00000011  0
```

3.5 Sign Extend

```
Time      out      in
------------------------------------------------------
0000  00000000  0
0028  11111111  1
003c  00000000  0
0050  11111111  1
```



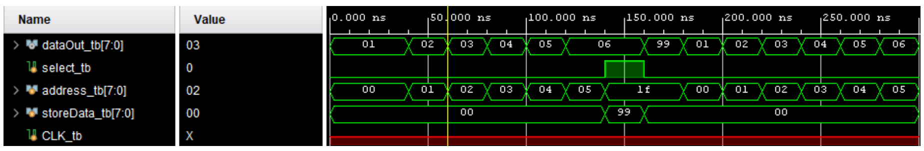
3.6 ALU



Name	Value
rdd_tb[7:0]	01
rsd_tb[7:0]	01
select_tb	0
out_tb[7:0]	02

```
Time        rs_data     rd_data     select  out
------------------------------------------------------
0000 xxxxxxxx xxxxxxxx x 00000000
0014 00000001 00000001 0 00000010
0028 00000010 00000011 1 00000001
003c 00000001 00000001 0 00000010
0050 00000010 00000011 1 00000001
0064 00000001 00000001 0 00000010
```

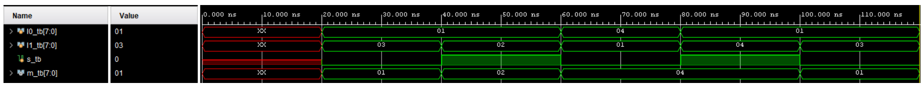
3.7 Data Memory

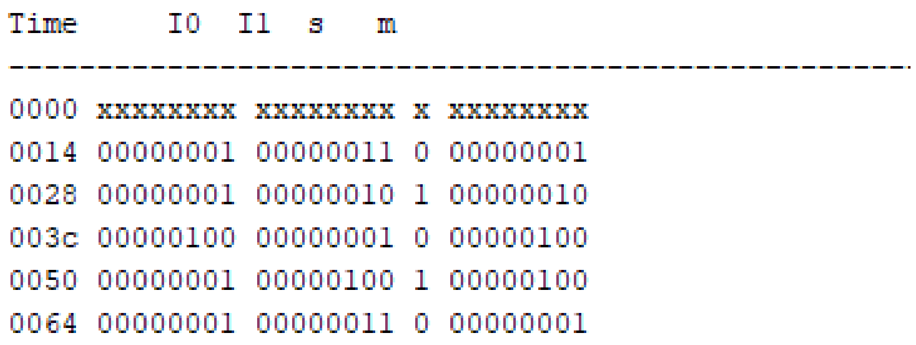


Name	Value
dataOut_tb[7:0]	03
select_tb	0
address_tb[7:0]	02
storeData_tb[7:0]	00
CLK_tb	X

```
Time        dataOut     select  address     storeData
------------------------------------------------------
0000 01, 0, 00000000, 00
0028 02, 0, 00000001, 00
003c 03, 0, 00000010, 00
0050 04, 0, 00000011, 00
0064 05, 0, 00000100, 00
0078 06, 0, 00000101, 00
008c 06, 1, 00011111, 99
00a0 99, 0, 00011111, 00
00b4 01, 0, 00000000, 00
00c8 02, 0, 00000001, 00
00dc 03, 0, 00000010, 00
00f0 04, 0, 00000011, 00
0104 05, 0, 00000100, 00
0118 06, 0, 00000101, 00
```

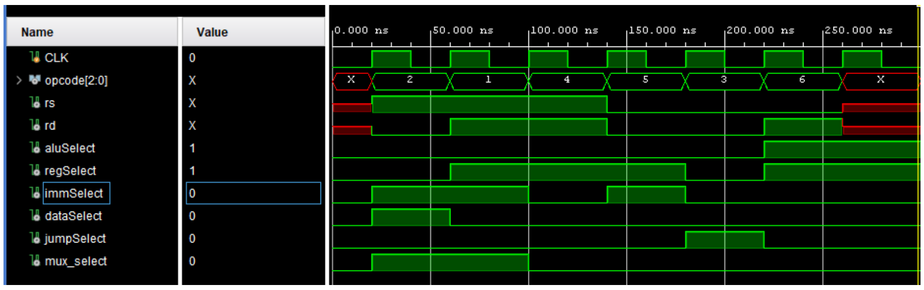
3.8 Multiplexors





```
Time       I0    I1    s    m
-------------------------------------------------------
0000 xxxxxxxx xxxxxxxx x xxxxxxxx
0014 00000001 00000011 0 00000001
0028 00000001 00000010 1 00000010
003c 00000100 00000001 0 00000100
0050 00000001 00000100 1 00000100
0064 00000001 00000011 0 00000001
```

3.9 Main



```
Time      rs  rd  aluSelect   regSelect   immSelect   dataSelect  mux_select  jumpSelect  opcode  CLK
---------------------------------------------------

0000  x  x  0  0  0  0  0  0  xxx  0
0014  1  0  0  0  1  1  1  0  010  1
0028  1  0  0  0  1  1  1  0  010  0
003c  1  1  0  1  1  0  1  0  001  1
0050  1  1  0  1  1  0  1  0  001  0
0064  1  1  0  1  0  0  0  0  100  1
0078  1  1  0  1  0  0  0  0  100  0
008c  0  0  0  1  1  0  0  0  101  1
00a0  0  0  0  1  1  0  0  0  101  0
00b4  0  0  0  0  0  0  0  1  011  1
00c8  0  0  0  0  0  0  0  1  011  0
00dc  0  1  1  1  0  0  0  0  110  1
00f0  0  1  1  1  0  0  0  0  110  0
0104  x  x  1  1  0  0  0  0  xxx  1
0118  x  x  1  1  0  0  0  0  xxx  0
```

4 Work Distribution

- Bodhiswattwa was responsible for testing the datapath on hardware and designing detailed test suites for the majority of the components. She also helped assist with the development of every component in the system.
- Jesse was responsible for designing `design.v`, `sign_ext.v`, `data_memory.v`. Jesse also assisted Bodhiswattwa in designing the test suites.
- Ben was responsible for writing the project report, writing the `muxes.v` file, and assisting with the development of the remaining components.
- Andrew was responsible for designing `ALU.v`, `control_unit.v`, `instruction_memory.v`, and `prog_counter`.

5 References

1. `https://www.cise.ufl.edu/~mssz/CompOrg/CDA-proc.html`
2. `https://gustavus.edu/mcs/max/CODfigs/f04-18.pdf`
3. `https://cse.buffalo.edu/~rsridhar/cse490-590/hw/project1_spring2022.pdf`
4. `https://cse.buffalo.edu/~rsridhar/cse490-590/hw/CSE490_590_Project1_Pointers_final.pdf`