RV32I Single Cycle implementation

Computer Architecture

Fall 2024

The American University in Cairo

Aly Elaswad 900225517

Ismail Sabry 900222002

Supervised by Dr. Cherif Salama

Introduction

This report presents our implementation of the RV32I instruction set architecture in a single-cycle processor model. The RV32I architecture is designed to support a variety of operations including arithmetic, logic, control flow, and memory access. Our implementation allows for the execution of basic instructions efficiently within a single clock cycle.

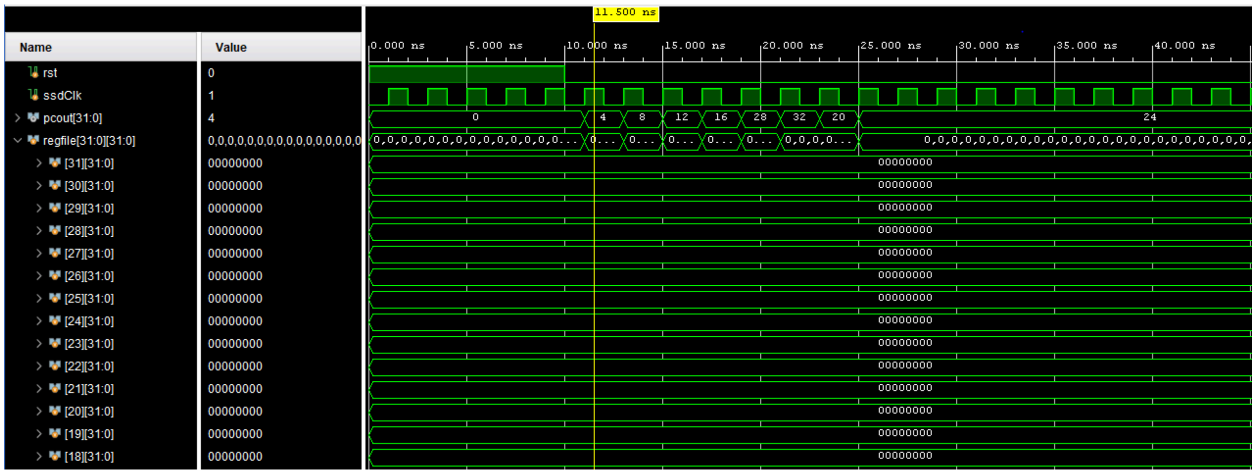
These are some of the test cases:

Test case 1: JALR, AUIPC,ECALL,I & R

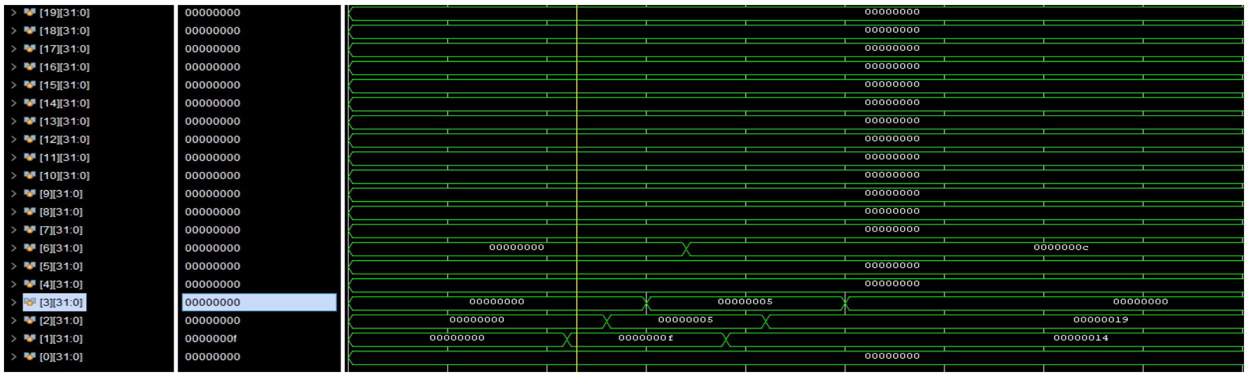
Instructions:

addi x1,x0,15
addi x2,x0,5
addi x3,x0,5
call func
addi x3,x3,-5
ecall
func:
add x2,x2,x1
ret

Result:



```
Name                 Value
rst                  0
ssdClk               1
pcout[31:0]          4
regfile[31:0][31:0]  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
[31][31:0]           00000000
[30][31:0]           00000000
[29][31:0]           00000000
[28][31:0]           00000000
[27][31:0]           00000000
[26][31:0]           00000000
[25][31:0]           00000000
[24][31:0]           00000000
[23][31:0]           00000000
[22][31:0]           00000000
[21][31:0]           00000000
[20][31:0]           00000000
[19][31:0]           00000000
[18][31:0]           00000000
```



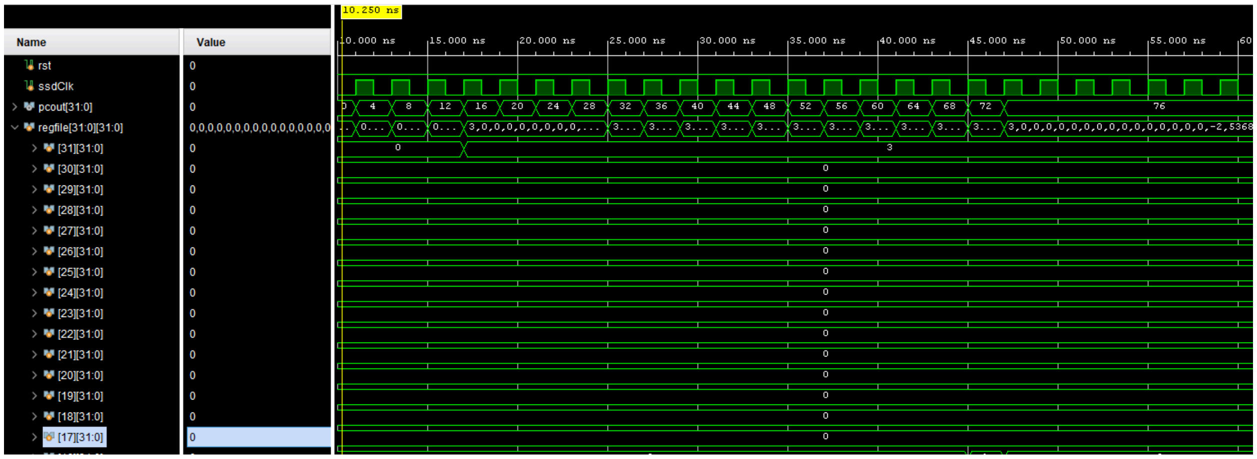
```
[19][31:0]  00000000
[18][31:0]  00000000
[17][31:0]  00000000
[16][31:0]  00000000
[15][31:0]  00000000
[14][31:0]  00000000
[13][31:0]  00000000
[12][31:0]  00000000
[11][31:0]  00000000
[10][31:0]  00000000
[9][31:0]   00000000
[8][31:0]   00000000
[7][31:0]   00000000
[6][31:0]   00000000
[5][31:0]   00000000
[4][31:0]   00000000
[3][31:0]   00000000
[2][31:0]   00000000
[1][31:0]   0000000f
[0][31:0]   00000000
```

Test case 2: LUI and shift

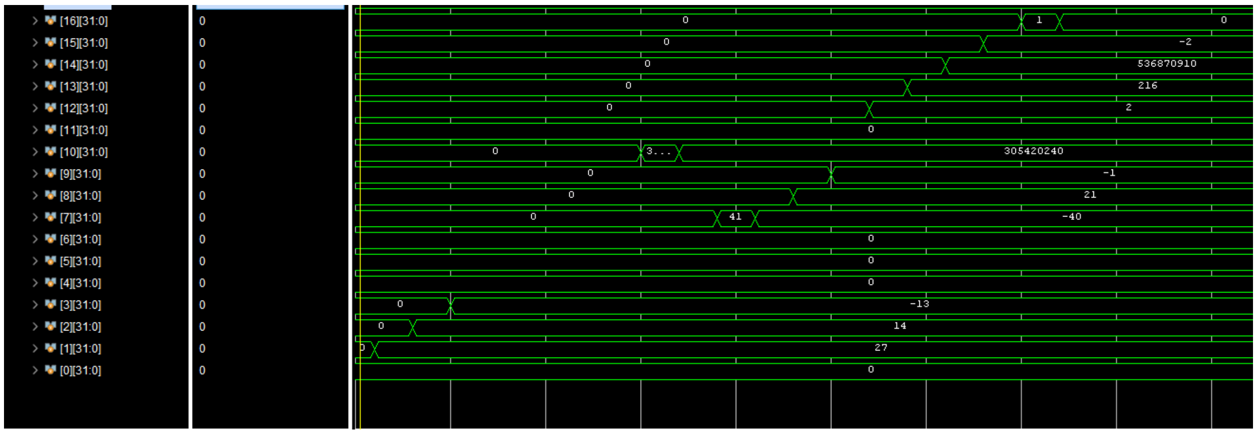
Instructions:

```
li x1,27
li x2,14
li x3,-13
li x31,3
sw x2,0(x0)
sh x3,4(x0)
sb x1,8(x0)
lui x10,0x12345
addi x10,x10,2000
add x7,x1,x2
sub x7,x3,x1
xor x8,x1,x2
or x9,x2,x3
and x12,x2,x3
sll x13,x1,x31
srl x14,x3,x31
sra x15,x3,x31
slt x16,x3,x2
sltu x16,x3,x2
```

Result:



Name	Value
rst	0
ssdClk	0
pcout[31:0]	0
regfile[31:0][31:0]	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
[31][31:0]	0
[30][31:0]	0
[29][31:0]	0
[28][31:0]	0
[27][31:0]	0
[26][31:0]	0
[25][31:0]	0
[24][31:0]	0
[23][31:0]	0
[22][31:0]	0
[21][31:0]	0
[20][31:0]	0
[19][31:0]	0
[18][31:0]	0
[17][31:0]	0

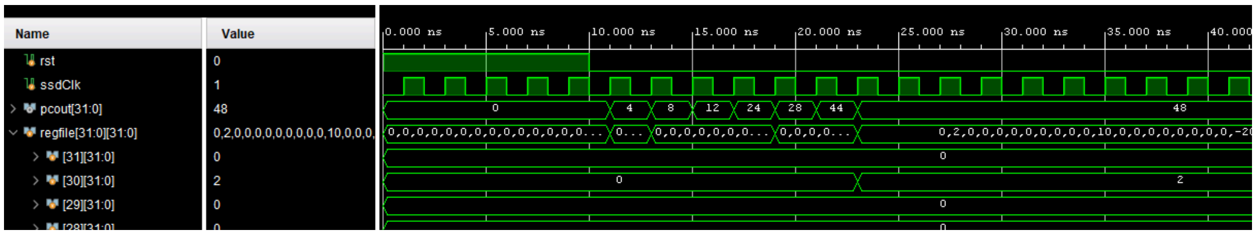


Name	Value
[16][31:0]	0
[15][31:0]	0
[14][31:0]	0
[13][31:0]	0
[12][31:0]	0
[11][31:0]	0
[10][31:0]	0
[9][31:0]	0
[8][31:0]	0
[7][31:0]	0
[6][31:0]	0
[5][31:0]	0
[4][31:0]	0
[3][31:0]	0
[2][31:0]	0
[1][31:0]	0
[0][31:0]	0

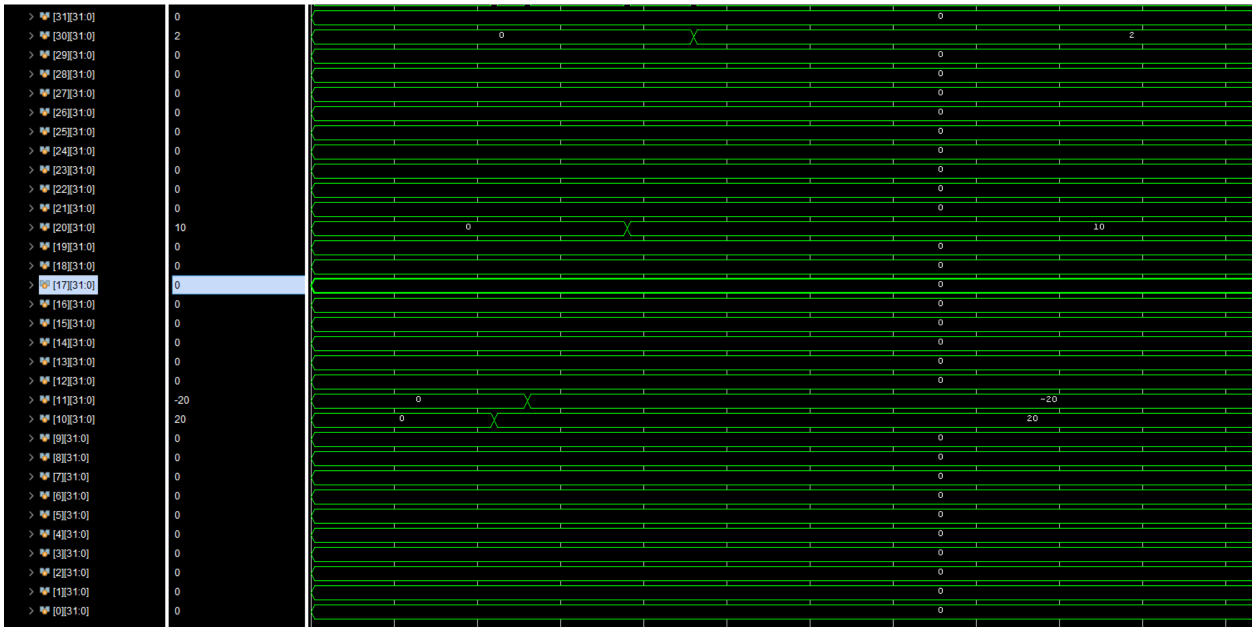
Test case 3:
Instructions:

```
li x10,20
li x11,-20
beq x10,x11,WRONGBEQ
bne x10,x11,RightBNE
li x22,5
ecall
RightBNE:
li x20,10
bltu x20,x11,CorrectBLT
li x23,10
li a7,10
ecall
WRONGBEQ:
li x31,1
CorrectBLT:
li x30,2
```

Result:



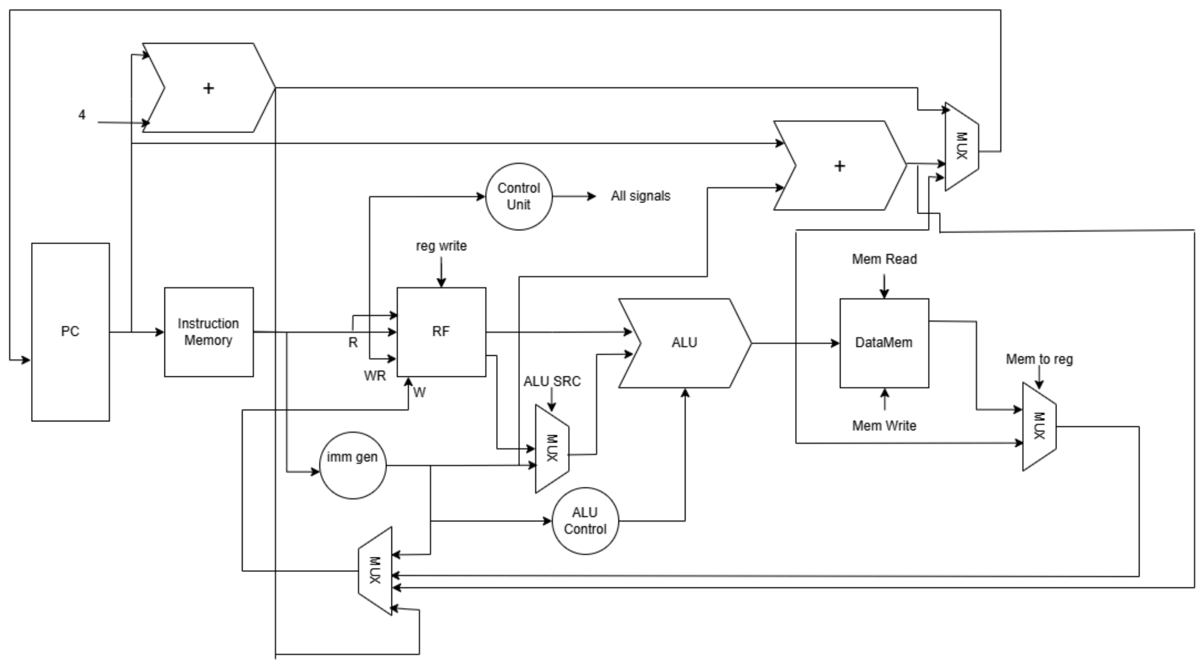
Name	Value
rst	0
ssdClk	1
pcout[31:0]	48
regfile[31:0][31:0]	0,2,0,0,0,0,0,0,0,0,0,10,0,0,0,
[31][31:0]	0
[30][31:0]	2
[29][31:0]	0
[28][31:0]	0



[31][31:0]	0
[30][31:0]	2
[29][31:0]	0
[28][31:0]	0
[27][31:0]	0
[26][31:0]	0
[25][31:0]	0
[24][31:0]	0
[23][31:0]	0
[22][31:0]	0
[21][31:0]	0
[20][31:0]	10
[19][31:0]	0
[18][31:0]	0
[17][31:0]	0
[16][31:0]	0
[15][31:0]	0
[14][31:0]	0
[13][31:0]	0
[12][31:0]	0
[11][31:0]	-20
[10][31:0]	20
[9][31:0]	0
[8][31:0]	0
[7][31:0]	0
[6][31:0]	0
[5][31:0]	0
[4][31:0]	0
[3][31:0]	0
[2][31:0]	0
[1][31:0]	0
[0][31:0]	0

Datapath:

Notes:
All necessary shifts have been made in the imm gen



The datapath for our RV32I single-cycle implementation consists of several key components:

1. **Registers:** 32 general-purpose registers.
2. **ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations.
3. **Control Unit:** Decodes instructions and generates control signals.
4. **Memory:** Data memory for loading and storing data.
5. **Instruction Memory:** Holds the instructions to be executed.
6. **MUXes:** Used for selecting between different data sources (e.g., for branch targets, ALU inputs).

Key Features:

- Single-cycle execution allows for simplicity in design but limits clock speed due to the longest path delay.
- Each instruction goes through a fetch, decode, execute, and memory stage within one cycle.
- The use of control signals facilitates the operation of the ALU and memory based on the type of instruction being executed.

Challenges in Implementation of RV32I Single Cycle Architecture

Implementing the RV32I instruction set architecture in a single-cycle processor model presented several challenges, particularly concerning memory operations, immediate generation, and the adaptation of existing ALU code. Below, we discuss these challenges in detail.

1. Memory Addressing: Word vs. Byte Addressable Memory

One of the primary challenges we faced was related to the memory model used in our implementation. The RV32I architecture is designed for byte-addressable memory, while our lab setup utilized word-addressable memory. This discrepancy required careful consideration in handling memory instructions, particularly for loading and storing data.

a. Load and Store Instructions

- **LB (Load Byte) and LH (Load Halfword):** These instructions are critical for working with byte and halfword data types. Since our memory was word-addressable, implementing LB and LH required adjustments to ensure that we correctly accessed the appropriate byte or halfword within a word.
- **Implementation Strategy:** To address this, we needed to calculate the correct address offsets based on the byte or halfword requested. This involved implementing logic to mask and shift bits appropriately when accessing specific bytes or halfwords from the word-aligned memory. For instance, using bitwise operations helped isolate the correct bytes from the loaded word.

b. Storing Data

Similarly, for store instructions (SW, SB, SH), we had to ensure that we were writing to the correct location in memory. This meant adapting our store operations to take into account the byte offset when storing byte or halfword values, requiring careful calculations to maintain data integrity.

2. Immediate Generation: IMMGEN

The immediate generation unit (IMMGEN) in the RISC-V architecture is responsible for generating the immediate values used in various instructions. Implementing this feature posed challenges related to ensuring proper sign extension and handling different instruction formats.

a. Sign Extension

Different instructions have varying immediate formats, such as 12-bit immediates for I-type instructions and 20-bit immediates for J-type instructions. We had to ensure that when extracting these immediates from the instruction binary, they were correctly sign-extended to 32 bits.