

# End-semester Assignment

Neelam Sharma (18307R030)  
EE 789: Algorithmic Design of Digital Systems

April 28, 2020

## 1 Introduction

A 32-bit non-pipelined processor is designed in Aa language to execute a custom instruction set and tested for addition of two 128-bit numbers on Basys3 board.

### 1.1 Instruction set

1. HALT
2. SBIR imm rd  
Loads byte imm into lowest byte of rd.
3. LOAD rs1 rd  
 $\langle rd \rangle \leftarrow mem[\langle rs1 \rangle]$
4. STORE rs1 rs2  
 $mem[\langle rs1 \rangle] \leftarrow \langle rs2 \rangle$
5. OP rs1 rs2 rd  
 $\langle rd \rangle \leftarrow \langle rs1 \rangle \text{ OP } \langle rs2 \rangle$ , where OP = AND, OR, XNOR, XOR, ADD, SUB, SLL, SRL, SRA
6. BZ rs1 rs2  
*if  $\langle rs1 \rangle == 0$  jump to  $\langle rs2 \rangle$*
7. BN rs1 rs2  
*if  $\langle rs1 \rangle$  is negative, jump to  $\langle rs2 \rangle$*
8. CALL rs1 rd  
*jump to  $\langle rs1 \rangle$ , store pc in rd.*
9. JMP rs1 *jump to  $\langle rs1 \rangle$*
10. CMP rs1 rs2 rd  
 $\langle rd \rangle \leftarrow (excmux$   
 $(\langle rs1 \rangle == \langle rs2 \rangle) 0$   
 $(\langle rs1 \rangle < \langle rs2 \rangle) -1$   
 $(\langle rs1 \rangle > \langle rs2 \rangle) 1)$

## 2 Algorithm implemented

---

**Algorithm 1** processor's algorithm

---

```
1: procedure EXECUTEINSTRUCTIONS(carry)           ▷ Runs till a HALT instruction is reached
2:   pc  $\leftarrow$  0
3:   currentInstruction  $\leftarrow$  mem[pc]
4:   opcode  $\leftarrow$  currentInstruction[31 – 24]
5:   rs1  $\leftarrow$  currentInstruction[23 – 16]
6:   rs2  $\leftarrow$  currentInstruction[15 – 8]
7:   rd  $\leftarrow$  currentInstruction[7 – 0]
8:   while (opcode  $\neq$  HALT) do           ▷ While current instrctions's opcode is not HALT keep
executing
9:     regWrite, memWrite  $\leftarrow$  decode(opcode)
10:    rd1  $\leftarrow$  regFile[rs1]
11:    rd2  $\leftarrow$  regFile[rs2]
12:    execResult  $\leftarrow$  calALUResult(opcode, rd1, rd2)
13:    memOutData  $\leftarrow$  calBridge(memWrite, memAddr, memInData)
14:    if opcode == LOAD then
15:      regFile[rd]  $\leftarrow$  memOutData
16:    else if regWrite == True then
17:      regFile[rd]  $\leftarrow$  execResult
18:    pc  $\leftarrow$  callController(opcode, rd1, rd2)
19:    currentInstruction  $\leftarrow$  mem[pc]
20:    opcode  $\leftarrow$  currentInstruction[31 – 24]
21:    rs1  $\leftarrow$  currentInstruction[23 – 16]
22:    rs2  $\leftarrow$  currentInstruction[15 – 8]
23:    rd  $\leftarrow$  currentInstruction[7 – 0]
24:  return None
```

---

## 3 Implementation Details

- **Memory organization:** A 32-bit 256 location memory which consists of both instructions and data.
- Above memory is implemented as an external memory module in VHDL using pipes.
- A Bridge daemon is designed which forwards the request and receives response from either memory or serial device depending upon address.
- For now, result of any instruction is sent to serial device as well.

## 4 Diagram

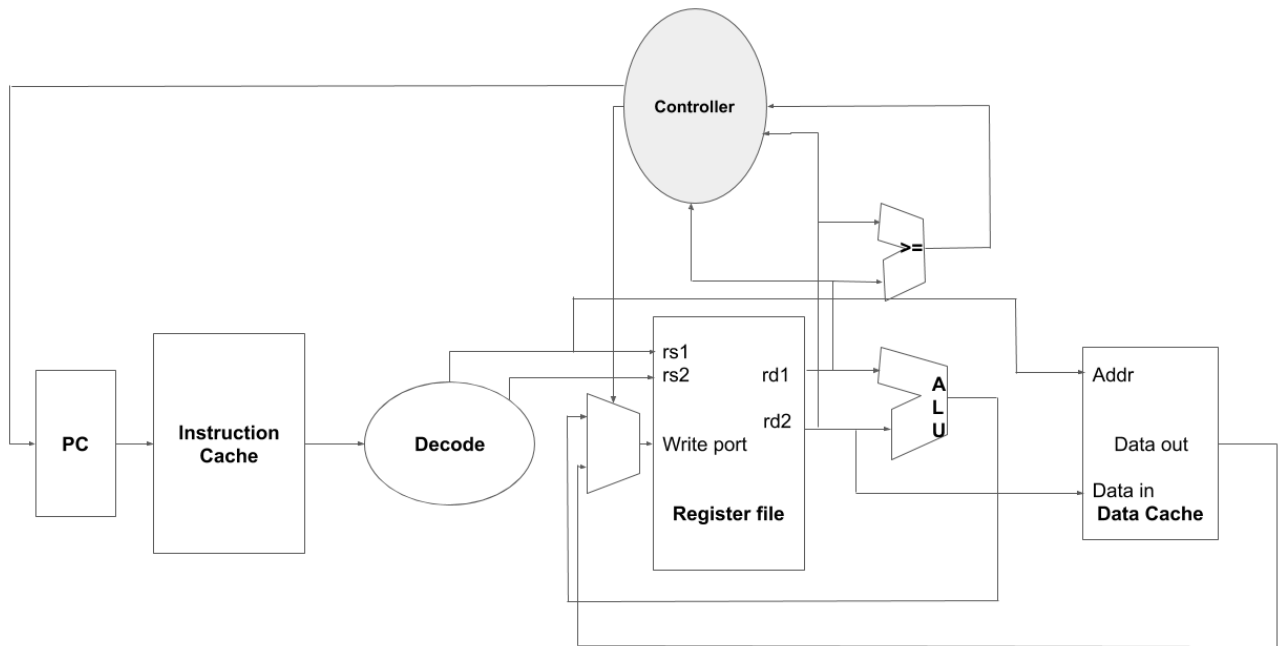


Figure 1: Processor architecture

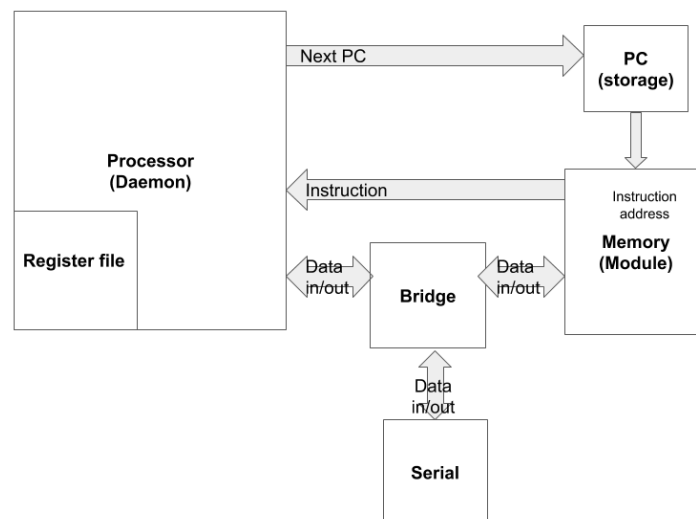


Figure 2: Overall architecture

Consider the data stored in memory with carry = 0x00000001 set initially:

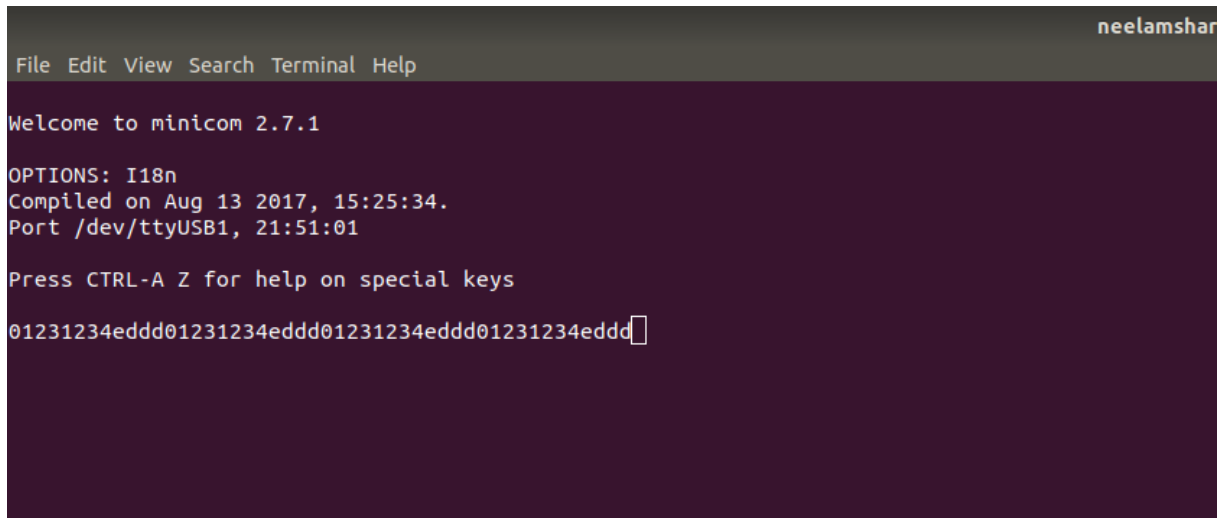
Location	Data stored	Result
0	SBIR 30 00 03	[r3] = x00000030 (ASCII value= 0)
1	SBIR 31 00 02	[r2] = x00000031 (ASCII value= 1)
2	SBIR 32 00 01	[r1] = x00000032 (ASCII value= 2)
3	SBIR 33 00 00	[r0] = 0x00000033 (ASCII value= 3)
4	SBIR 31 00 04	[r4] = 0x00000031 (ASCII value= 1)
5	SBIR 32 00 05	[r5] = 0x00000032 (ASCII value= 2)
6	SBIR 33 00 06	[r6] = 0x00000033 (ASCII value= 3)
7	SBIR 34 00 07	[r7] = 0x00000034 (ASCII value= 4)
8	ADD r0, r4, r8	[r8] = 0x00000065 (ASCII value= e )
9	ADD r1, r5, r9	[r9] = 0x00000064 (ASCII value = d )
10	ADD r2, r6, r10	[r10] = 0x00000064 (ASCII value = d )
11	ADD r3, r7, r11	[r11] = = 0x00000064 (ASCII value = d )
12	HALT	None

```
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 32
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 33
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 31
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 32
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 33
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 34
Received from UART: 0
Received from UART: 0
Received from UART: 65
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 64
Received from UART: 0
Received from UART: 0
Received from UART: 0
Received from UART: 64
Received from UART: 0
Received from UART: 0
Received from UART: 64
[1] 0:[mux]#
```

4

## 5.2 Serial Port results

The button (btnU) on Basys3 is used to start the processor, pressing it resulted in multiple times processor to be triggered that's why same result is repeated 4 times.



```
neelamshar
File Edit View Search Terminal Help
Welcome to minicom 2.7.1
OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port /dev/ttyUSB1, 21:51:01
Press CTRL-A Z for help on special keys
01231234eddd01231234eddd01231234eddd01231234eddd█
```

Figure 4: Serial terminal's Output

### Explanation

- SBIR 30 00 03 sends output of data to be written on register r3 i.e., x00000030 (ASCII value=0).
- SBIR 31 00 02 sends output of data to be written on register r2 i.e., x00000031 (ASCII value=1).
- SBIR 32 00 01 sends output of data to be written on register r1 i.e., x00000032 (ASCII value=2).
- SBIR 33 00 00 sends output of data to be written on register r0 i.e., x00000033 (ASCII value=3).
- SBIR 31 00 04 sends output of data to be written on register r4 i.e., x00000031 (ASCII value=1).
- SBIR 32 00 05 sends output of data to be written on register r5 i.e., x00000032 (ASCII value=2).
- SBIR 33 00 06 sends output of data to be written on register r6 i.e., x00000033 (ASCII value=3).
- SBIR 34 00 07 sends output of data to be written on register r7 i.e., x00000034 (ASCII value=4).
- ADD r0, r4, r8 sends output of data to be written on register r8 i.e., x00000065 (ASCII value=e).

- ADD r1, r5, r9 sends output of data to be written on register r9 i.e., x00000064 (ASCII value=d).
- ADD r2, r6, r10 sends output of data to be written on register r10 i.e., x00000064 (ASCII value=d).
- ADD r3, r7, r11 sends output of data to be written on register r11 i.e., x00000064 (ASCII value=d).