End-semester Assignment

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

August 21, 2020

1 Introduction

A 32-bit 6-staged pipelined processor is designed in Aa language to execute a custom instruction set.

1.1 Instruction set

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

2 Architectural overview

This processor includes following stages:

1. Fetch and Instruction cache stage
2. Decode
3. Register read
4. Execute
5. Memory read/write
6. Write-back

3 Design Decisions

3.1 Forwarding Decisions

1. Data cache state to execute stage:
   Condition 1: $dcache.reg\_write == 1\ dcache.rd == iexec.rs1$
   Action: Forward dcache.exec_result to rs2 input.
   Condition 2: $dcache.reg\_write == 1\ dcache.rd == iexec.rs2$
   Action: Forward dcache.exec_result to rs2 input.
   Example:
   ADD r1, r2, r4
   SUB r4, r7, r9

2. Instruction retire stage to execute stage:
   Condition 1: $iretire.reg\_write == 1\ iretire.rd == iexec.rs1$
   Action: Forward iretire.mem_regData to rs1 input.
   Condition 2: $iretire.reg\_write == 1\ iretire.rd == iexec.rs2$
   Action: Forward iretire.mem_regData to rs2 input.
   Example:
   OR r5, r2, r4
   AND r5, r6, r7
   SUB r4, r7, r9

3. Instruction retire stage to Data cache stage:
   Condition 1: $iretire.reg\_write == 1\ iretire.rd == dcache.memAddr$
   Action: Forward iretire.mem_regData to dcache.memAddr
   Condition 2: $iretire.reg\_write == 1\ iretire.rd == dcache.rs1$
   Action: Forward iretire.mem_regData to dcache.dataIn
   Example:
   LOAD r2, r0, r5
   STORE r5, r2, r8

4. Instruction retire stage to Register file stage:
   Condition 1: $iretire.reg\_write == 1\ iretire.rd == regFile.rs1$
   Action: Forward iretire.mem_regData to rd1 output of register file stage.

Condition 2: $iretire.reg\_write == 1\ iretire.rd == regFile.rs2$
Action: Forward iretire.mem_regData to rd2 output of register file stage.
Example:
OR r5, r2, r4
AND r5, r6, r7
SUB r8, r7, r9
SLL r4, r11, r9

3.2 Branch hazard Decisions

1. BN, BZ instructions are detected at the output of EX/DCACHE pipelined register. Five stages (Instruction Fetch, Instruction cache, Decode, Register file, Execute) are flushed for a conditional branch instruction.
   Condition: $dcache.is\_Branch == 1$
   Action: Flush the stages previous to Data cache stage.

2. Unconditional jumps are detected at the output of REG/EX pipelined register. Four stages (Instruction Fetch, Instruction cache, Decode, Register file) are flushed for a unconditional branch instruction.
   Condition: $iexec.is\_unconditional\_Branch == 1$
   Action: Flush the stages previous to Execute stage.

3.3 Stalling Decisions

1. LOAD preceded by some dependent instruction.
   Condition: $iexec.rd == regFile.rs1 \,|\, iexec.rd == regFile.rs2$
   Action: Stall first four stages (Instruction Fetch, Instruction cache, Decode, Register file) and flush REG/EX pipelined register in next iteration.
   Example:
   LOAD r1, r0, r4
   ADD r4, r6, r9

4 Diagram

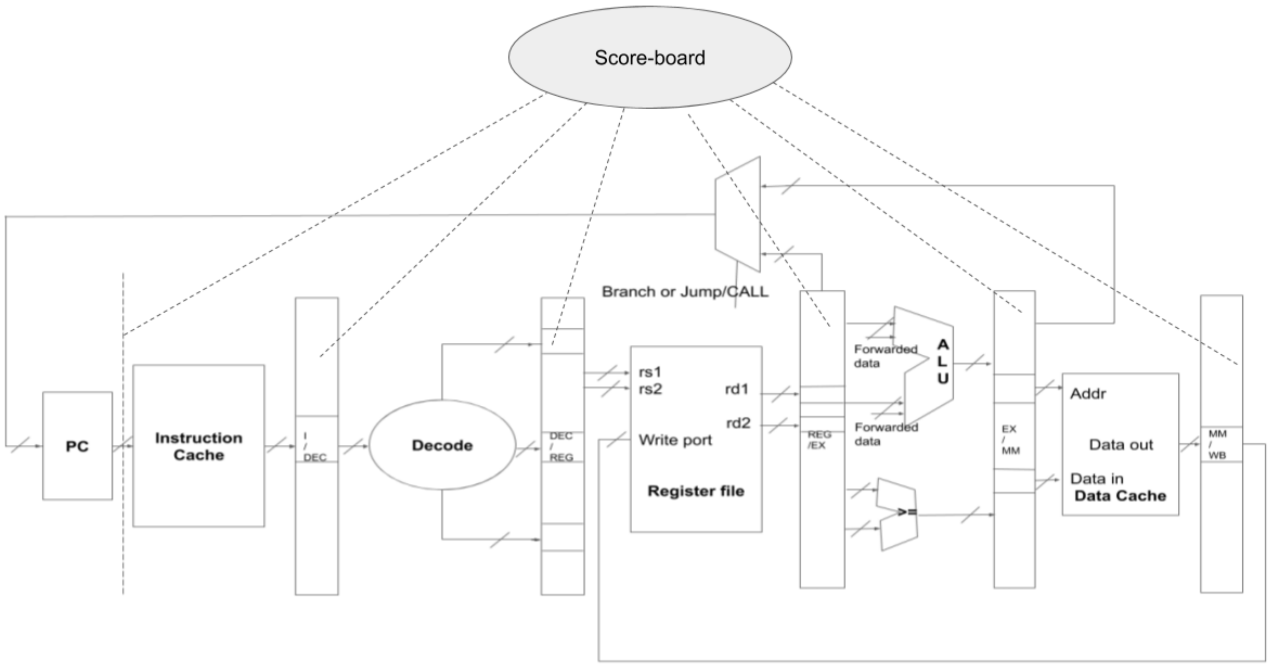


Figure 1: Pipelined architecture

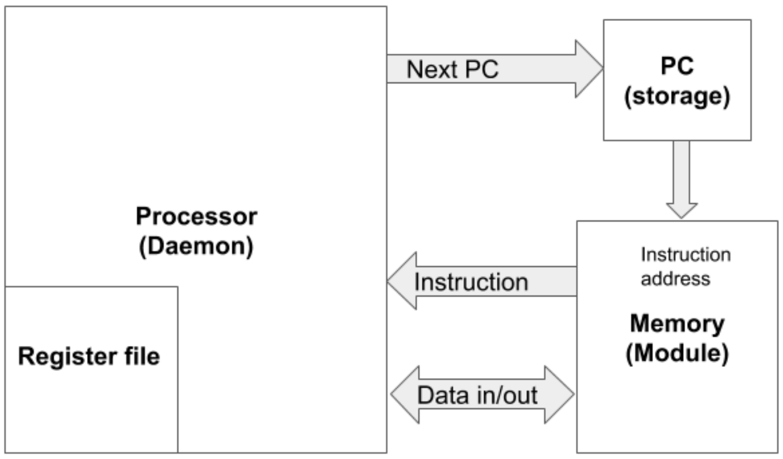


Figure 2: Overall architecture

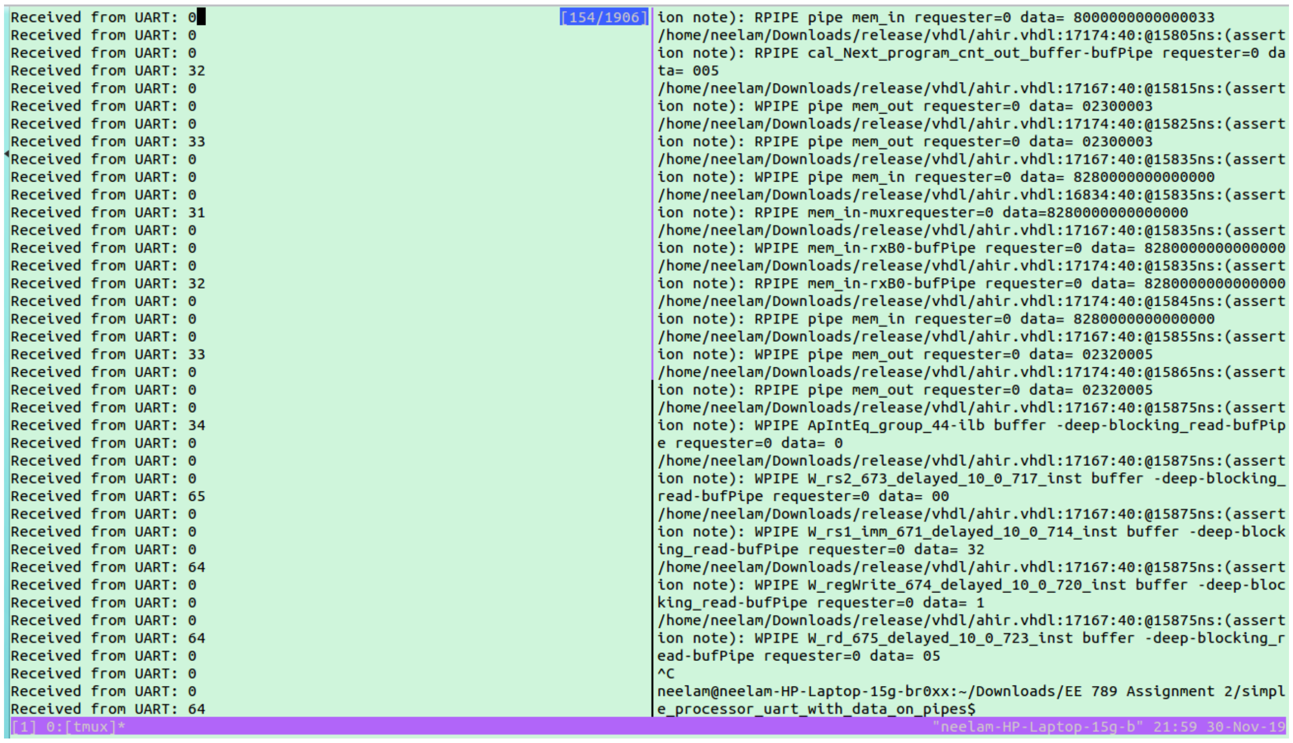
5 Results

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

Table 1: Memory map and corresponding action

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

5.1 Simulation



```
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: 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: 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: 0
Received from UART: 64
```

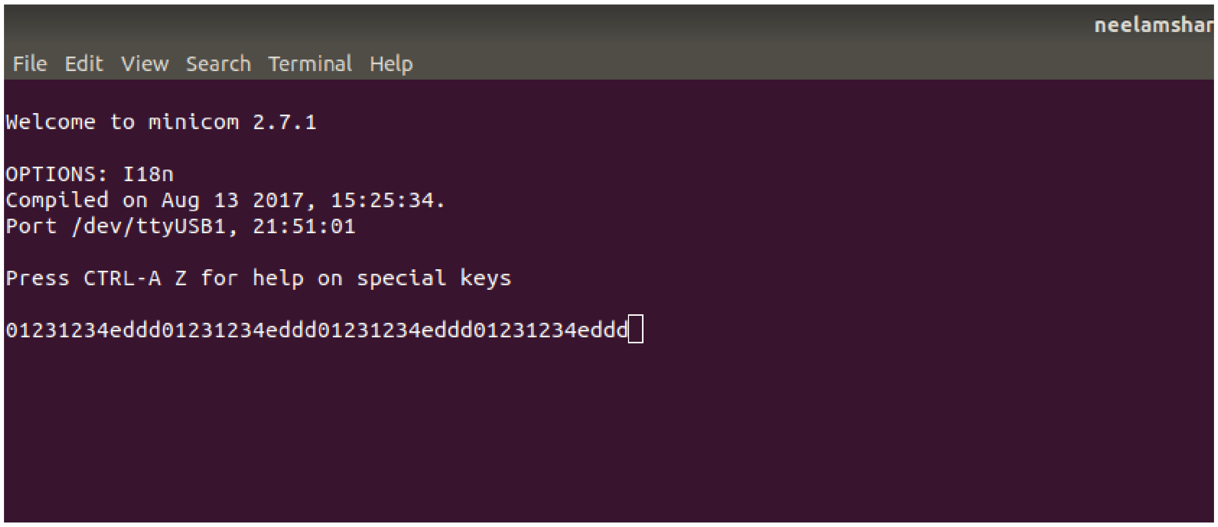
```
[154/1906]
ion note): RPIPE pipe mem_in requester=0 data= 8000000000000033
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17174:40:@15805ns:(assert
ion note): RPIPE cal_Next_program_cnt_out_buffer-bufPipe requester=0 da
ta= 005
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15815ns:(assert
ion note): WPIPE pipe mem_out requester=0 data= 02300003
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17174:40:@15825ns:(assert
ion note): RPIPE pipe mem_out requester=0 data= 02300003
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15835ns:(assert
ion note): WPIPE pipe mem_in requester=0 data= 8280000000000000
/home/neelam/Downloads/release/vhdl/ahir.vhdl:16834:40:@15835ns:(assert
ion note): RPIPE mem_in-muxrequester=0 data=8280000000000000
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15835ns:(assert
ion note): WPIPE mem_in-rxB0-bufPipe requester=0 data= 8280000000000000
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17174:40:@15835ns:(assert
ion note): RPIPE mem_in-rxB0-bufPipe requester=0 data= 8280000000000000
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17174:40:@15845ns:(assert
ion note): RPIPE pipe mem_in requester=0 data= 8280000000000000
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15855ns:(assert
ion note): WPIPE pipe mem_out requester=0 data= 02320005
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17174:40:@15865ns:(assert
ion note): RPIPE pipe mem_out requester=0 data= 02320005
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15875ns:(assert
ion note): WPIPE ApIntEq_group_44-ilb buffer -deep-blocking_read-bufPip
e requester=0 data= 0
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15875ns:(assert
ion note): WPIPE W_rs2_673_delayed_10_0_717_inst buffer -deep-blocking_
read-bufPipe requester=0 data= 00
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15875ns:(assert
ion note): WPIPE W_rs1_imm_671_delayed_10_0_714_inst buffer -deep-block
ing_read-bufPipe requester=0 data= 32
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15875ns:(assert
ion note): WPIPE W_regWrite_674_delayed_10_0_720_inst buffer -deep-bloc
king_read-bufPipe requester=0 data= 1
/home/neelam/Downloads/release/vhdl/ahir.vhdl:17167:40:@15875ns:(assert
ion note): WPIPE W_rd_675_delayed_10_0_723_inst buffer -deep-blocking_r
ead-bufPipe requester=0 data= 05
^C
neelam@neelam-HP-Laptop-15g-br0xx:~/Downloads/EE 789 Assignment 2/simpl
e_processor_uart_with_data_on_pipes$
```

```
[1] 0:[tmux]*                                  "neelam-HP-Laptop-15g-b" 21:59 30-Nov-19
```

Figure 3: Simulation Output

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.



```
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: Simulation 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).