Report

On

32 BIT PIPELINED RISC PROCESSOR

Apoorv Dethe

JUNE ‘20

In this project, 16 bit pipelined RISC processor is designed in verilog language and is simulated in ModelSim.

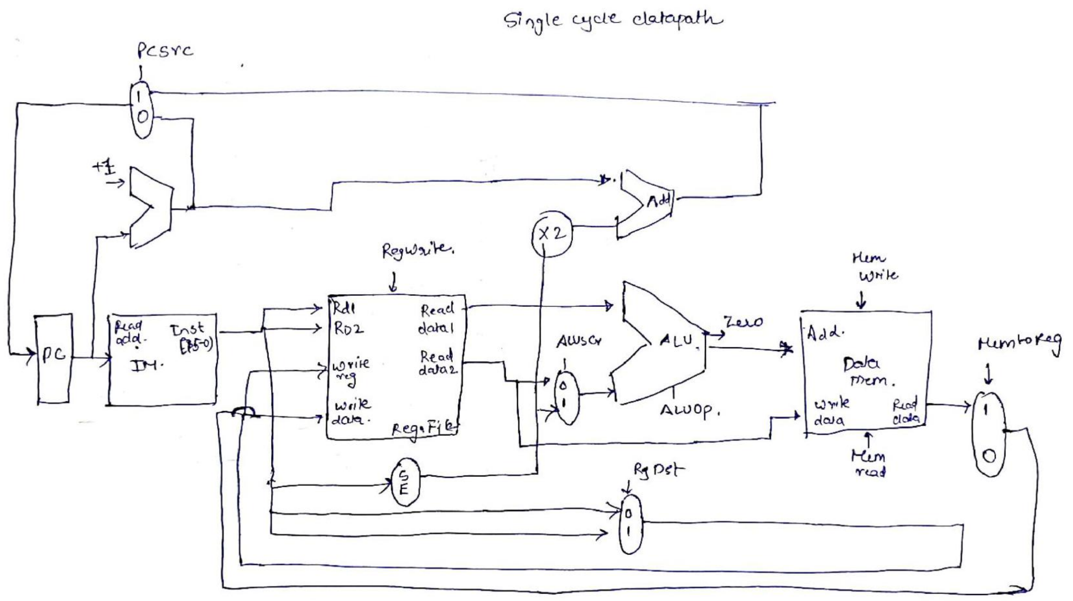
The instruction set of RISC processor consist of following 3 types of instruction format

1. R Type format.
2. I type format.
3. J type format.

The R type instruction format is followed by all arithmetic instructions whose operands are in registers. I type format is followed by lw, sw, adi etc. J type format is followed by j, jal type instructions.

In this project the following things are assumed:
Instruction length is 16 bit.
Registers are of 8 bit length.
There are 8 registers.
Opcode is of 4 bits.

First of all single cycle data path is designed and the block diagram of single cycle datapath is shown below. This datapath does not include hardware for jump instruction

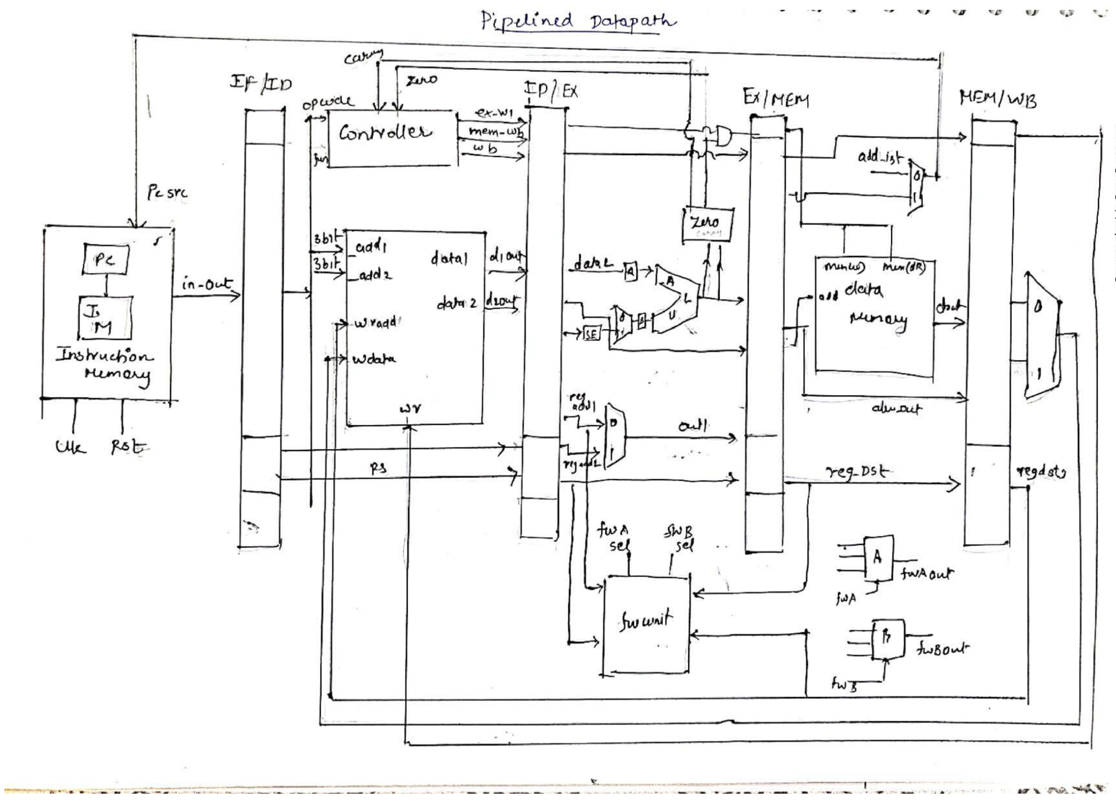


Single cycle datapath

.
Now to make it 5 staged PIPELINED architecture we add 4 registers.
These are :

1. Instruction fetch - decode
2. Decode - execution
3. Execution - memory
4. Memory - write back

The new block diagram is as shown below:



Everything works fine but when there are two instructions back to back and in 1st instruction data is written into register and in second instruction data is used then there creates a problem. This is call data hazard and this hazard can be can be mitigated by data forwarding unit.
In this forwarding unit data is directly fed into ALU input when its calculated.

In this project , arithmetic, logical, branch, load and store instructions are implemented.

The simulations of instructions is shown below:

<u>For arithmetic/logical instructions:</u>

The content of register file is

R0 = 6

R1 = 5

R2 = 3

R3 = 6

R4 = 2

R5 = FF

R6 = FF

R7 = PC

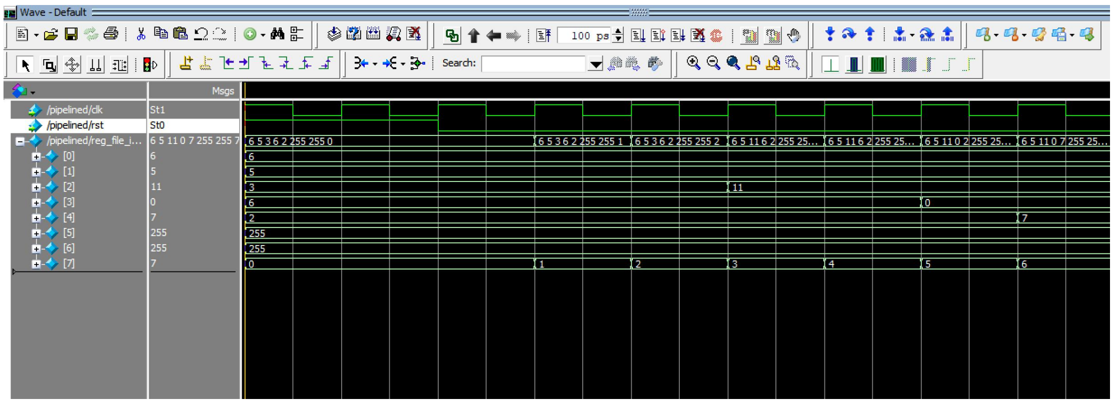
We will execute following instructions:

ADD R2,R0,R1; /////////R2=11;

SUB R3,R3,R0;/////////R3=0;

ADZ R4,R4,R1;//////////R4=7;

From the simulations we also get R4 =7;

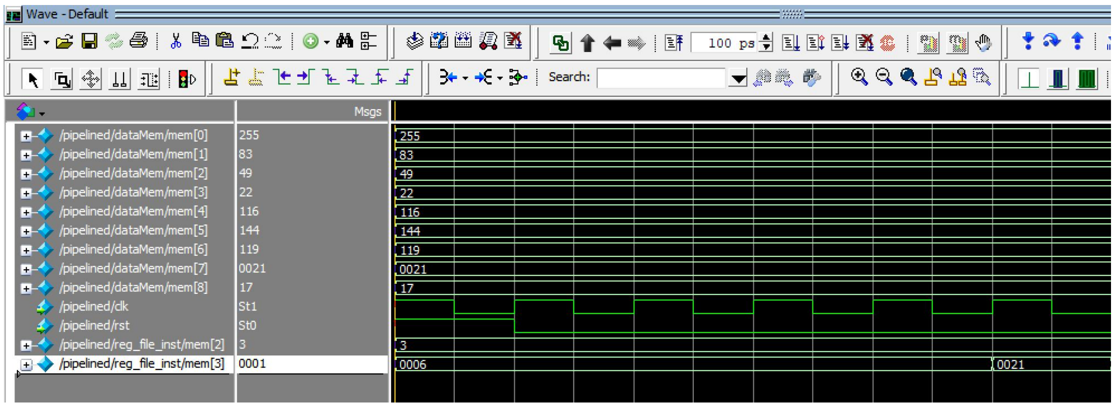


```
/pipelined/clk         St1
/pipelined/rst         St0
/pipelined/reg_file_i... 6 5 11 0 7 255 255 7   6 5 3 6 2 255 255 0 | 6 5 3 6 2 255 255 1 | 6 5 3 6 2 255 255 2 | 6 5 11 6 2 255 25... | 6 5 11 6 2 255 25... | 6 5 11 0 2 255 25... | 6 5 11 0 7 255 25...
  [0]   6     6
  [1]   5     5
  [2]   11    3 | 11
  [3]   0     6 | 0
  [4]   7     2 | 7
  [5]   255   255
  [6]   255   255
  [7]   7     0 | 1 | 2 | 3 | 4 | 5 | 6
```

Arithmetic

<u>For lw :</u>

We will execute the following instruction
lw R3,R2,4; ////which means address = (content(R2) +4)
R3 = mem(address)
We can calculate address = 7
So data at mem(7) should be stored at R3
0021 which is stored in mem[7] is loaded into R3



```
/pipelined/dataMem/mem[0]        255     255
/pipelined/dataMem/mem[1]        83      83
/pipelined/dataMem/mem[2]        49      49
/pipelined/dataMem/mem[3]        22      22
/pipelined/dataMem/mem[4]        116     116
/pipelined/dataMem/mem[5]        144     144
/pipelined/dataMem/mem[6]        119     119
/pipelined/dataMem/mem[7]        0021    0021
/pipelined/dataMem/mem[8]        17      17
/pipelined/clk                   St1
/pipelined/rst                   St0
/pipelined/reg_file_inst/mem[2]  3       3
/pipelined/reg_file_inst/mem[3]  0001    0006    0021
```

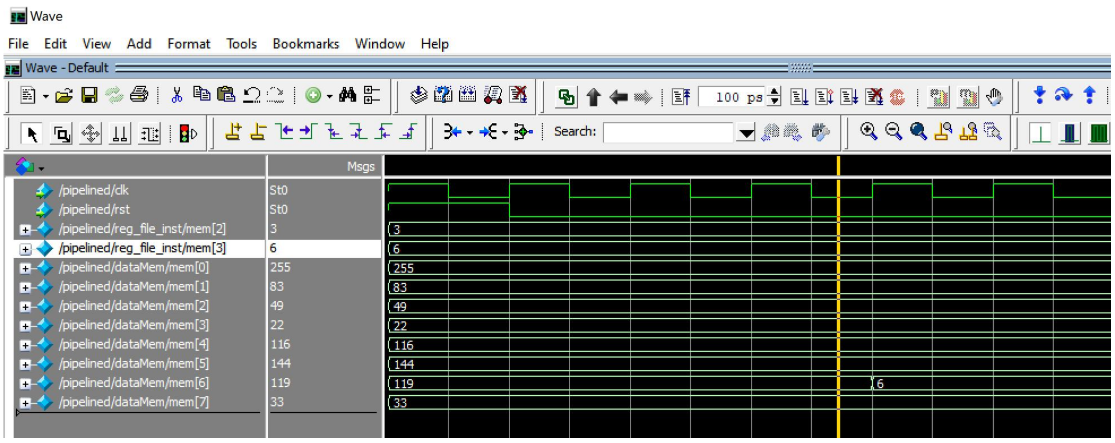
LW

<u>For sw :</u>

We will execute the following instruction to store word from register to memory

sw R3,R2,3; /////it means the data of register R3 will be stored at (content(R2)+3)

Content of R3 will be stored at mem(6) which is aso confirmed from simulations



```
/pipelined/clk                   St0
/pipelined/rst                   St0
/pipelined/reg_file_inst/mem[2]  3       3
/pipelined/reg_file_inst/mem[3]  6       6
/pipelined/dataMem/mem[0]        255     255
/pipelined/dataMem/mem[1]        83      83
/pipelined/dataMem/mem[2]        49      49
/pipelined/dataMem/mem[3]        22      22
/pipelined/dataMem/mem[4]        116     116
/pipelined/dataMem/mem[5]        144     144
/pipelined/dataMem/mem[6]        119     119     6
/pipelined/dataMem/mem[7]        33      33
```

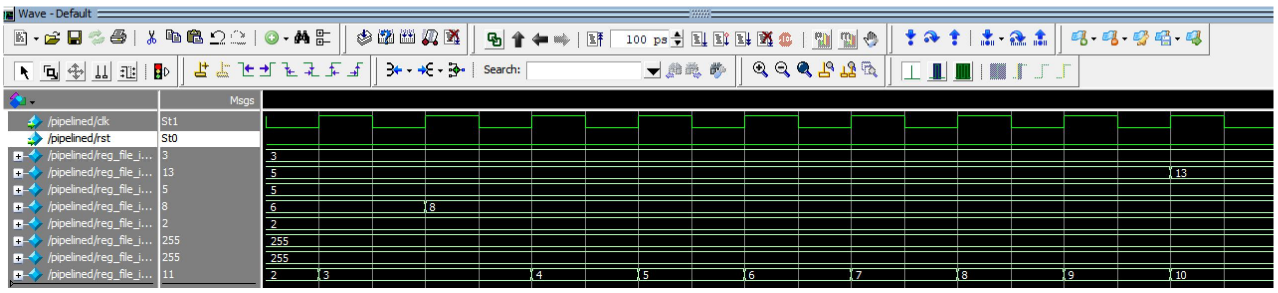
SW

<u>For branch instruction:</u>

We will consider the following instruction set;
The content of register file is
R0 = 6
R1 = 5
R2 = 3
R3 = 6
R4 = 2
R5 = FF
R6 = FF
R7 = PC

```
ADD R3,R0,R1
BEQ R2,R1,1; //////If contents of R2 and R1 are equal then branch
ADD R3,R2,R0;
ADD R3,R1,R0;
ADD R3,R2,R0;
ADD R3,R3,R1;     ////Branch here
ADD R1,R0,R1;
```

After this instruction the content of R3 = 13;



```
/pipelined/clk           St1
/pipelined/rst           St0
/pipelined/reg_file_i... 3     3
/pipelined/reg_file_i... 13    5 13
/pipelined/reg_file_i... 5     5
/pipelined/reg_file_i... 8     6 8
/pipelined/reg_file_i... 2     2
/pipelined/reg_file_i... 255   255
/pipelined/reg_file_i... 255   255
/pipelined/reg_file_i... 11    2 3 4 5 6 7 8 9 10
```

BEQ