Design of Pipelined MIPS Processor with Hazards

Submitted in the partial fulfillment of the course
Computer Architecture (CS F342)

Prepared By:

Nagesh Samane	2017A8PS0612P
Ateeksha Mittal	2017A8PS0431P

Submitted To:
Mr. Abheek Gupta
Ms. Kanika Monga
Mr. Mohit Vyas

Date:
5th July, 2020

A. Problem Statement:

Design a 5-stage pipelined MIPS processor that detects and remedies hazards for the following set of instructions

1. sub **\$2**, \$1, \$3
2. and \$8, **\$2**, \$5
3. or \$9, \$6, **\$2**
4. add **\$5**, \$5, \$6
5. sub \$4, \$3, \$6
6. beq **\$5**, \$6

Above set of instructions consist of 5 R-type instructions and 1 conditional branch instruction. From the analysis of instructions we concluded that there are three data hazards (of RAW -Read After Write type) that need to be solved. Forwarding units were designed to detect and solve the conflicts raised due to data dependencies.

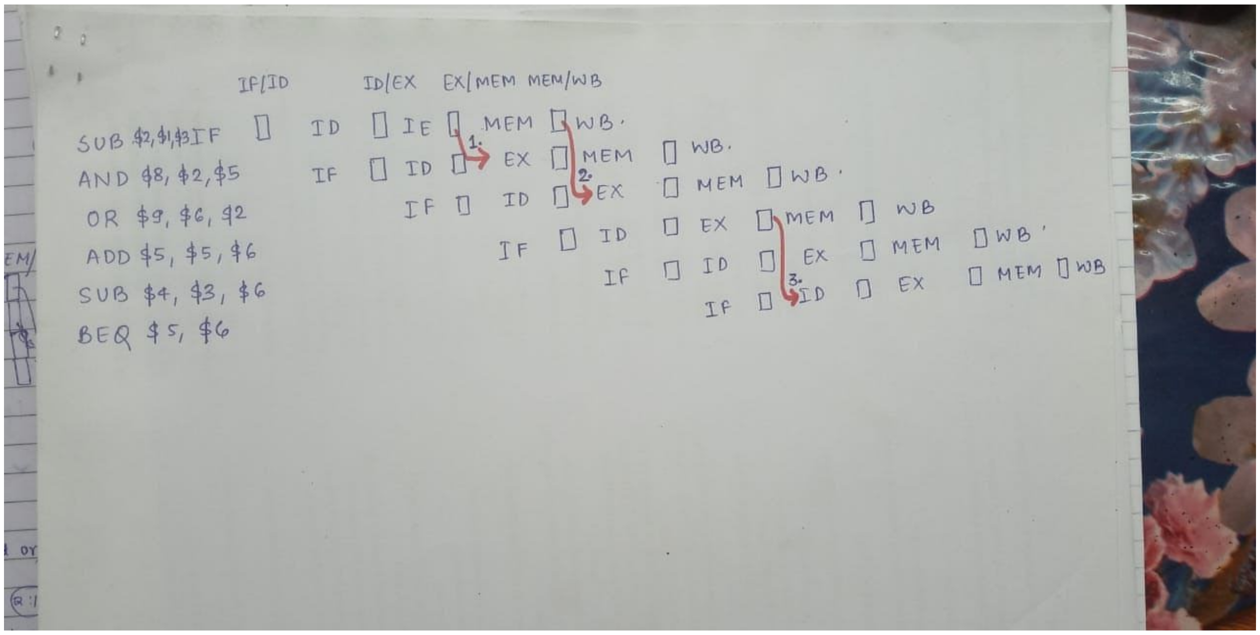


Figure 1. Solving data hazards by **forwarding units**

Data hazards 1 and 2 (for R-type instructions) are handled by forward_unit module and data hazard 3(for Branch instruction) handled by forward_unit_B module

B. Implementation Details:

1. Register file is of 32 registers numbered from **0 to 31** (each 32 bit) and initialized with value equal to its register number.
    a. e.g. r0 = 32’d0, r1 = 32’d1, r2 = 32’d2 and so on.

2. Instruction register is initialized with **six instructions** (as mentioned in the question) in 32 bit MIPS format from a text file present in the same directory.

3. Branch offset for Beq instruction is assumed to be 0xFFFE.
    a. Although with a given set of instructions the branch condition fails and no branch is taken. We have **verified the functionality branch taken** case by varying the set of instruction sizes to 9.

4. The size of pipeline registers is kept minimum and calculated by hand calculations based on the amount of data to be forwarded to the next stage.

5. Opcode for ALU (four operations encoded in 2 bits)

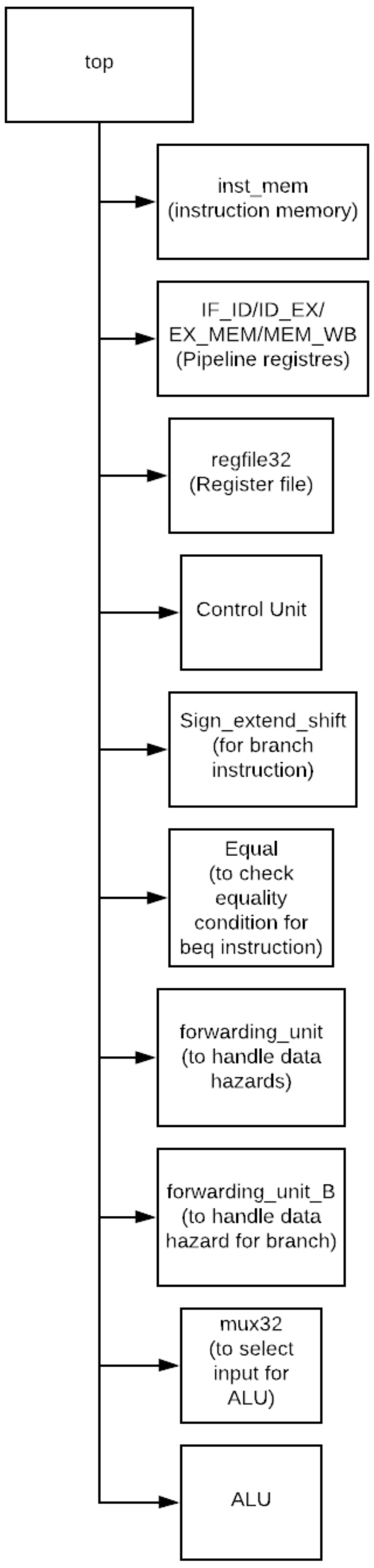
a.	Add	00
b.	And	01
c.	Or	10
d.	Sub	11

6. Clocking of modules
    a. All the operations in different stages of pipeline at particular time step and Next state calculation @negedge clock
    b. Next state assignment @posedge clock
    c. Instruction memory @negedge clock

d. Regfile registers @posedge clock
e. Pipeline registers @negedge clock

7. Description of text files
    a. *Instructions_final.txt:* contains instruction encoded in MIPS format stored in byte organized way
        i. **sub r2 r1 r3** `0x00231022`
        ii. **and r8 r2 r5** `0x00454024`
        iii. **or r9 r6 r2** `0x00C24825`
        iv. **add r5 r5 r6** `0x00A62820`
        v. **sub r4 r3 r6** `0x00662022`
        vi. **beq r5 r6 0xfffe** `0x10A6FFFE`
    b. Offset for branch is not mentioned in the question hence it is set to 0xFFFE to take branch to instruction 4 (0x10) if the condition is satisfied.
    c. *register.txt :* initial values of register
    d. *output_reg.txt:* final updated register file is read out

C. Modules Included in the Verilog Code:



D. State Diagram:

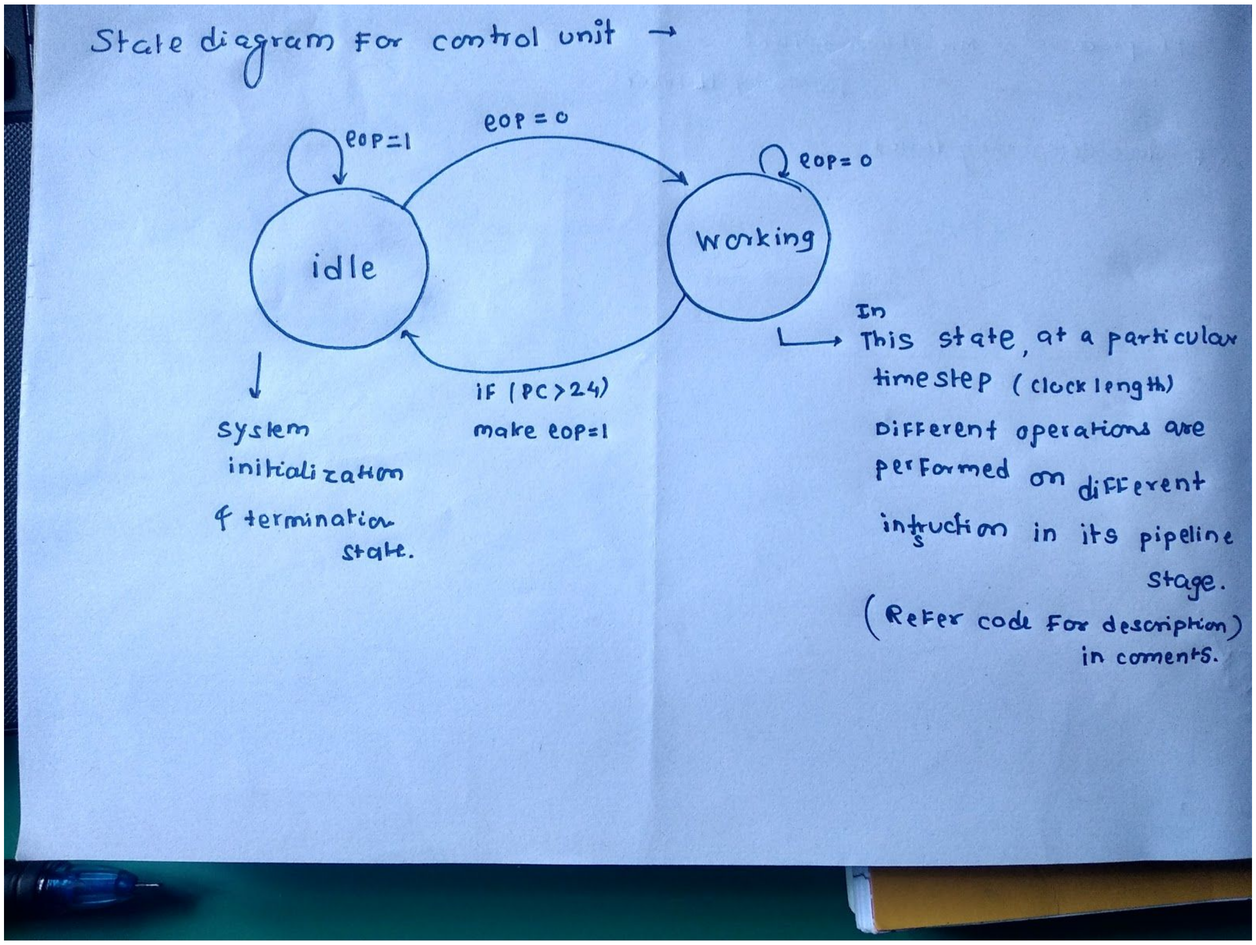
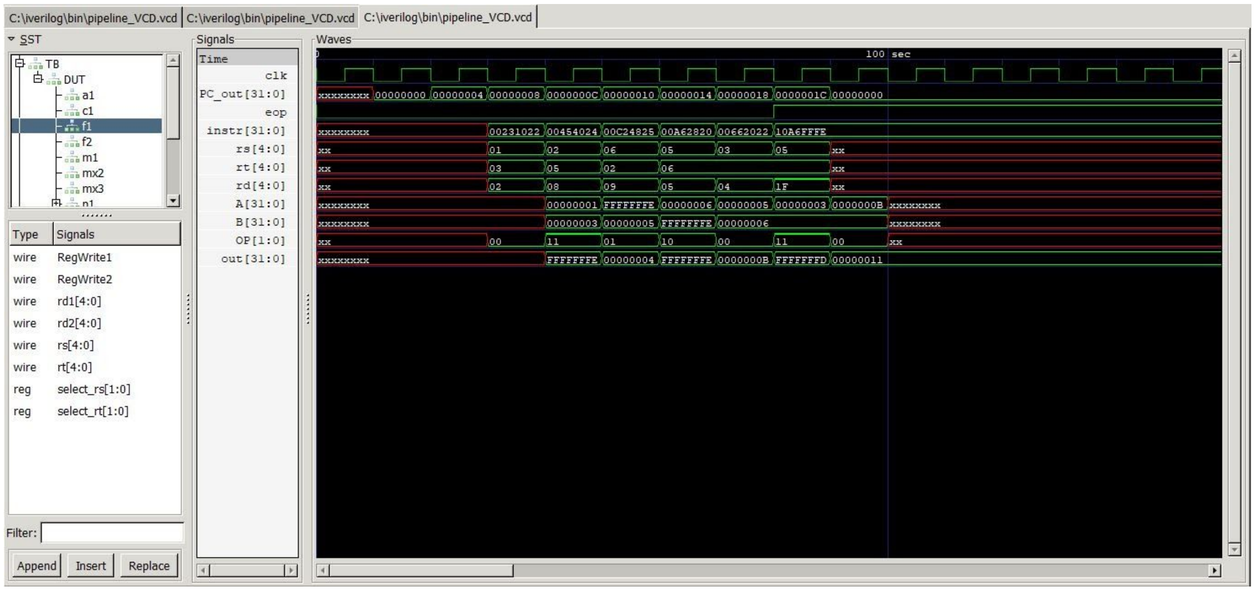

State diagram for control unit →
eop=1
eop = 0
eop= 0
idle
working
if (PC>24)
make eop=1
system initialization & termination state.
In This state, at a particular timestep (clock length) Different operations are performed on different instructions in its pipeline stage.
(Refer code For description) in coments.

Figure 2. State diagram for **Control Unit**

E. Analysis of result:


C:\iverilog\bin\pipeline_VCD.vcd
C:\iverilog\bin\pipeline_VCD.vcd
C:\iverilog\bin\pipeline_VCD.vcd
SST
TB
DUT
a1
c1
f1
f2
m1
mx2
mx3
Signals
Waves
Filter:
Append
Insert
Replace

Type	Signals
wire	RegWrite1
wire	RegWrite2
wire	rd1[4:0]
wire	rd2[4:0]
wire	rs[4:0]
wire	rt[4:0]
reg	select_rs[1:0]
reg	select_rt[1:0]

```
Time                                                                          100 sec
clk
PC_out[31:0]  xxxxxxxx 00000000 00000004 00000008 0000000C 00000010 00000014 00000018 0000001C 00000000
eop
instr[31:0]   xxxxxxxx 00231022 00454024 00C24825 00A62820 00662022 10A6FFFE
rs[4:0]       xx 01 02 06 05 03 05 xx
rt[4:0]       xx 03 05 02 06 xx
rd[4:0]       xx 02 08 09 05 04 1F xx
A[31:0]       xxxxxxxx 00000001 FFFFFFFE 00000006 00000005 00000003 0000000B xxxxxxxx
B[31:0]       xxxxxxxx 00000003 00000005 FFFFFFFE 00000006 xxxxxxxx
OP[1:0]       xx 00 11 01 10 00 11 00 xx
out[31:0]     xxxxxxxx FFFFFFFE 00000004 FFFFFFFE 0000000B FFFFFFFD 00000011
```

Figure 3. Timing diagram for implemented MIPS pipelined processor

Signals attached in the diagram:

1. Clock
2. PC_out: pointer to the next instruction in instruction in memory
3. eop: end of operation (here limited for 6 instructions)
4. instr: 32 bit instruction read from instruction memory
5. rs: source 1 (for R-type) register number extracted from current instruction
6. rt: source 2 (for R-type) register number extracted from current instruction
7. rd: destination register number extracted from current instruction
8. A: input 1 of ALU (which is output of mux32 module)
9. B: input 2 of ALU (which is output of mux32 module)
10. OP: opcode for ALU (extracted for instruction in EX stage)
11. out: output of ALU

Other signals can be verified by running ***pipeline_VCD.vcd*** file present in the project directory.

F. Conclusion:

From **output_reg.txt and timing diagram** it is clear that we have designed a five stage (IF, ID, EX, MEM, WB) MIPS pipelined processor capable of handling RAW data hazards for R-type and branch instruction.