23.12.2020

BILKENT UNIVERSITY

COMPUTER ENGINEERING DEPT.

CS224 LAB 5 SECTION 2

DESIGN REPORT

ZÜBEYİR BODUR

21702382

1. List of All Possible Hazards

Assuming no Hazad Unit existed in *PipelinedDatapath.png*, we would have four types of hazards that require three different solutions.

a) Compute-Use

- Type: Data hazard
- Affected Stages: ID (decode stage) and all stages after ID are affected.
- Description: All writebacks, except lw, whose subsequent insctructions include reading from the same register creates a data hazard since, in the subsequent insctructions, the register read is not updated.

b) Load-Use

- Type: Data hazard
- Affected Stages: ID (decode stage) and all stages after ID are affected.
- Description: If the lw instruciton is executed and it is followed up immediately by an instruction who reads from the rt register in lw, it creates a data hazard that can't be eliminated by forwarding.

c) Load-Store

- Type: Data hazard
- Affected Stages: ID (decode stage) and all stages after ID are affected.
- Description: Similar to load-use, if the lw instruction is executed and it is followed up by a sw instruction that has the same rt register, another data hazard occurs.

d) Branch Hazard

- Type: Control hazard
- Affected Stages: IF (fetch stage) and all stages after IF are affected (simply, all stages are affected).
- Description: If a branch instruction, beq for this processor, is executed, due to pipelining, a few subsequent instructions will use the old PC as PC is not updated yet,

creating a control hazard.

2. Hardware Solutions for each Hazard

a) Compute-use

- Solution: Data forwarding
- Data forwarding should be done when rs/rt register adresses in the IE stage are equal to rd register in WB stage. However, data forwarding shouldn't be done when the target register to forward is the zero register (rsE != 0 for SrcA and rtE != 0 for SrcB)
- Data forwarding should be done by forwarding the ALU output from two different stages (WB or MEM) to the SrcA or SrcB in the IE stage.

b) Load-use

- Solution: Stalling and flushing
- Stalling and flushing should be done when rs/rt register address in the ID stage is equal to rt register address in the IE stage and the instruction in IE stage includes writing data from memory to register (MemtoRegE == 1).
- Stalling should be done by disabling the pipes between WB-IF and IF-ID while flushing should be done by clearing the pipe between ID-IE.

c) Load-store

- Solution: Stalling and flushing
- Stalling and flushing should be done when rt register address in the ID stage is equal to rt register address in the IE stage and the instruction in IE stage includes writing data to memory (MemWriteE == 1).
- Stalling should be done by disabling the pipe between WB-IF and IF-ID while flushing should be done by clearing the pipe between ID-IE.

d) Branch

- Solution: Flushing

- Flushing should be done when branch prediction (that branch won't be taken) turns out to be false, when PCSrcM is 1.
- IE, ID and IF stages needs to be flushed so that PCSrcM can have the right PC to be fetched in WB stage without any hazard. Flushing should be done by clearing the pipes between ID-IE, IF-ID, and WB-IF.

3. Logic Eqns. for Hazard Unit

Introduce the variable load_use_stall and load_store_stall below:

load_use_stall = ((rsD == rtE) | (rtD == rtE)) & MemtoRegE

load_store_stall = (rtD == rtE) & MemWriteE

Then, the logic equations for the outputs of the Hazard Unit will be the following:

a) StallF & StallD

StallF = StallD = load_use_stall | load_store_stall

b) FlushF & FlushD

FlushF = FlushD = PCSrcM

c) FlushE

FlushE = load_use_stall | load_store_stall | PCSrcM

d) ForwardAE

```
if ((rsE != 0) & (rsE == WriteRegM) & RegWriteM)
    ForwardAE = 2'b10
else if ((rsE != 0) & (rsE == WriteRegW) & RegWriteW)
    ForwardAE = 2'b01
else
    ForwardAE = 2'b00
```

e) ForwardBE

if ((rtE != 0) & (rtE == WriteRegM) & RegWriteM)

ForwardBE = 2'b10

else if ((rtE != 0) & (rtE == WriteRegW) & RegWriteW)

ForwardBE = 2'b01

else

ForwardBE = 2'b00

4. Test Programs

The following set of instructions could be given once at a time to the IMEM to check if the pipelined processor will work.

a) MIPS code

```
# no hazards
addi    $t1, $t1, 1
addi    $t2, $t2, 2
addi    $t3, $t3, 3
addi    $t4, $t4, 4
addi    $t5, $t5, 5
addi    $t6, $t6, 7
addi    $s1, $s1, 6
addi    $s2, $s2, 19
addi    $s5, $s5, 8
add     $s3, $t1, $t2
sw      $s2, 40($zero)
sub     $s4, $s1, $s5
and     $s5, $t5, $t6
lw      $s6, 40($zero)
or      $s7, $t3, $t4
```

```
#compute-use hazard
add     $s0, $s2, $s3
and     $t0, $s0, $s1
or      $t1, $s4, $s0
sub     $t2, $s0, $s5
#load-use hazard
lw      $s0, 40($zero)
and     $t0, $s0, $s1
or      $t1, $s4, $s0
sub     $t2, $s0, $s5
#load-store hazard
lw      $s2, 50($zero)
sw      $s2, 60($zero)
sw      $s2, 70($zero)
sw      $s2, 80($zero)
#branch hazard
beq     $t1, $t2, label
and     $t0, $s0, $s1
or      $t1, $s4, $s0
sub     $t2, $s0, $s5
lw      $s0, 40($zero)
label:
slt     $t3, $s2, $s3
```

b) Machine Code in Hex

```
8'h00: instr = 32'h21290001;
8'h04: instr = 32'h214a0002;
8'h08: instr = 32'h216b0003;
8'h0c: instr = 32'h218c0004;
8'h10: instr = 32'h21ad0005;
```

```
8'h14: instr = 32'h21ce0007;
8'h18: instr = 32'h22310006;
8'h1c: instr = 32'h22520013;
8'h20: instr = 32'h22b50008;
8'h24: instr = 32'h012a9820;
8'h28: instr = 32'hac120028;
8'h2c: instr = 32'h0235a022;
8'h30: instr = 32'h01aea824;
8'h34: instr = 32'h8c160028;
8'h38: instr = 32'h016cb825;
8'h3c: instr = 32'h02538020;
8'h40: instr = 32'h02114024;
8'h44: instr = 32'h02904825;
8'h48: instr = 32'h02155022;
8'h4c: instr = 32'h8c100028;
8'h50: instr = 32'h02114024;
8'h54: instr = 32'h02904825;
8'h58: instr = 32'h02155022;
8'h5c: instr = 32'h8c120032;
8'h60: instr = 32'hac12003c;
8'h64: instr = 32'hac120046;
8'h68: instr = 32'hac120050;
8'h6c: instr = 32'h112affff;
8'h70: instr = 32'h02114024;
8'h74: instr = 32'h02904825;
8'h78: instr = 32'h02155022;
8'h7c: instr = 32'h8c100028;
8'h80: instr = 32'h0253582a;
```