Bilkent University, Computer Science, CS224 Computer Organization

“Design Report”, Lab #05, Section #06, Barış Tan Ünal, 22003617

15.04.2022

PART B & C: List of All Data Hazards & Their Solutions

Abbreviations:

- A: Arithmetic operations (add, addi, sub, mul etc.).
- L: Logical operations (and, or, andi, ori, sll, srl etc.).
- C: Comparison operations (slt, slti).
- lw: Load word.
- sw: Store word.
- fec: Fetch stage.
- dec: Decode stage.
- exe: Execution stage.
- mem: Memory stage.
- wb: Writeback stage.
- df: Data forward.
- fwdAE / fwdBE: Control hazard forward signals.

RAW Data Hazards:

lw → A & L & C	1 stall in *dec* (stallD) + 1 df from *wb* to *exe* (fwdAE or fwdBE): load-use
lw → Branch	2 stalls in *dec* (stallD) + 1 df from *wb* to *dec* (fwdAD or fwdBD): load-branch
lw → lw	1 stall in *dec* (stallD) + 1 df from *wb* to *exe* (fwdAE or fwdBE): load-load
lw → sw	1 stall in *dec* (stallD) + 1 df from *wb* to *exe* (fwdAE or fwdBE): load-store

A & L & C → A & L & C	1 df from *mem* to *exe* (fwdAE or fwdBE): compute-use
A & L & C → Branch	1 stall in *dec* (stallD) + 1 df from *mem* to *dec* (fwdAD or fwdBD): compute-branch
A & L & C → lw	1 df from *mem* to *exe* (fwdAE or fwdBE): compute-load (use)
A & L & C → sw	1 df from *mem* to *exe* (fwdAE or fwdBE): compute-store (use)

Control Hazards:

Branch → A & L & C	1 flush in *exe* (flushE) OR 1 stall (stallF) in *fec:* branch
Branch → Branch	1 flush in *exe* (flushE) OR 1 stall (stallF) in *fec:* branch
Branch → lw	1 flush in *exe* (flushE) OR 1 stall (stallF) in *fec:* branch
Branch → sw	1 flush in *exe* (flushE) OR 1 stall (stallF) in *fec:* branch

General Template:

i1 → i2 solution (stalls, forwards, flushes), (the required hazard unit signal): type

Remarks:

The reason of all of RAW data hazards listed below are that we are trying to read the data which has not been written yet.

- *lw → non-branch instr:* One clock cycle of stalling (by stallD) and data forwarding from wb to exe stage are needed since we cannot forward the resultW to SrcAE or SrcBE through a 3:1 mux with the help of forwardAE & forwardBE hazard unit signals.

- *lw → branch*: We need two stalls (by stallD) instead of one because the data (resultW) should be forwarded to *dec* stage with the help of forwardAD & forwardBD hazard unit signals since branch decision is made in the *dec* stage.

- *Arithmetic & Logical & Comparison instr → non-branch instr*: No stalling is needed but a data forwarding is needed from mem to exe. The forwarded data is ALUOutM which is forwarded to the place of SrcAE or SrcBE, depending on the mux selector forwardAE & forwardBE hazard unit signals.

- *Arithmetic & Logical & Comparison instr → branch:* One clock cycle of stalling is needed because the data (resultW) should be forwarded to *dec* stage with the help of forwardAD & forwardBD hazard unit signals since branch decision is made in the *dec* stage.

The reason of all control hazards listed below is that we don't know to take the branch or not. As a solution, we can predict the branch as 'not taken' and flush if our prediction was wrong.

- *Branch → all instr:* Either stalling for one clock cycle without making an early-prediction (by stallF) or flushing after an early-prediction is needed.

Note: There are no WAR or WAW hazards in the given pipelined microprocessor setup.

PART D: Forwarding Logic

lwStall = ((rsD == rtE) OR (rtD == rtE)) AND MemtoRegE

branchStall = (BranchD AND RegWriteE AND ((WriteRegE == rsD) OR (WriteRegE == rtD)))
OR
(BranchD AND MemtoRegM AND ((RegWriteM == rsD) OR (WriteRegM == rtD))

StallF = StallD = FlushE = lwStall OR branchStall

ForwardAD = (rsD != 0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD != 0) AND (rtD == WriteRegM) AND RegWriteM

```
if (rsE != 0) AND (rsE == WriteRegM) AND (RegWriteM)
        ForwardAE = 10
else if (rsE != 0) AND (rsE == WriteRegW) AND (RegWriteW)
        ForwardAE = 01
else
        ForwardAE = 00

if (rtE != 0) AND (rtE == WriteRegM) AND (RegWriteM)
        ForwardBE = 10
else if (rtE != 0) AND (rtE == WriteRegW) AND (RegWriteW)
        ForwardBE = 01
else
        ForwardBE = 00
```

PART E: Hazards of *sracc* & Solutions

- *sracc* → *non-branch instr:* (RAW data hazard). It is a compute-use type of hazard. It occurs when a sracc intruction is followed by any instructions besides branch type and j-type instructions. It can be solved dynamically with a data forwarding through fwdAE or fwdBE, which is from *mem* to *exe*. Another solution is stalling for one clock cycle. If it is wanted to be solved statically, then two *nop* instructions should be inserted in between those instructions such that it would wait for the *wb* stage of *sracc* to be completed just before *add* instruction reads the reqiured registers.

Example test program **with** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
sracc $t0, $t1, $t2
add $t3, $t0, $0
```

Example test program **without** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
sracc $t0, $t1, $t2
nop
nop
add $t3, $t0, $0
```

- *sracc* → *branch:* (RAW data hazard). It is a compute-branch type of hazard. It occurs when a sracc intruction is followed by a branch instruction. It can be solved dynamically by stalling for one clock cycle followed by a data forwarding through fwdAD or fwdBD, which is from *mem* to *dec*. If it is wanted to be solved statically, then two *nop* instructions should be inserted in between those instructions such that it would wait for the *wb* stage of *sracc* to be completed just before branch instruction reads the reqiured registers.

Example test program **with** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
sracc $t0, $t1, $t2
beq $t0, $t1, skip
```

Example test program **without** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
sracc $t0, $t1, $t2
nop
nop
beq $t0, $t1, skip
```

- non-branch/non-lw instr → sracc: (RAW data hazard). It is a compute-use type of hazard. It occurs when any instruction besides branch type, j-type or *lw* instruction is followed by *sracc* instruction. It can be solved dynamically with a data forwarding through fwdAE or fwdBE, which is from *mem* to *exe*. Another solution is stalling for one clock cycle. If it is wanted to be solved statically, then two *nop* instructions should be inserted in between those instructions such that it would wait for the *wb* stage of *add* instruction to be completed just before *sracc* instruction reads the reqiured registers.

Example test program **with** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
add $t0, $t1, $t2
sracc $t3, $t0, $t1
```

Example test program **without** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
add $t0, $t1, $t2
nop
nop
sracc $t2, $t0, $t1
```

- branch → sracc: (Control hazard). It is a branch type of hazard. It occurs when branch instruction is followed by *sracc* instruction. It can be solved dynamically with flushing the branch prediction. Another solution is stalling for one clock cycle. If it is wanted to be solved statically, then one *nop* instructions should be inserted in between those instructions such that it would wait for the *dec* stage of *beq* instruction to be completed just before taking the branch or not.

Example test program **with** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
beq $t0, $t1, skip
sracc $t2, $t0, $t1
```

Example test program **without** hazards:

```
addi $t0, 2
addi $t1, 3
addi $t2, 4
beq $t0, $t1, skip
nop
sracc $t2, $t0, $t1
```

- lw –> sracc: (RAW data hazard). It is a load-use type of hazard. It occurs when *lw* instruction is followed by *sracc* instruction. It can be solved dynamically with stalling for two clock cycles and a data forwarding through fwdAE or fwdBE, which is from *wb* to *exe*. If it is wanted to be solved statically, then two *nop* instructions should be inserted in between those instructions such that it would wait for the *wb* stage of *lw* instruction to be completed just before *sracc* instruction reads the reqiured registers.

Example test program **with** hazards:

```
addi $t0, 2
addi $t1, 3
lw $t2, 4($t1)
sracc $t1, $t2, $t0
```

Example test program **without** hazards:

```
addi $t0, 2
addi $t1, 3
lw $t2, 4($t1)
nop
nop
sracc $t1, $t2, $t0
```