10320 CS410001 - Computer Architecture 2015

Appendix C2 - Sample Output for Project 2

First of all, we have the following format for you to generate your output report. **Be very careful for the format of the report! You must precisely follow the format, or your result may be evaluated as incorrect!**

Format:

cycle (cycle index in decimal representation)
$00: 0x(content in hexadecimal digits) # note that there is 1 space between ":" and "0x"
$01: 0x(content in hexadecimal digits)
…
… (other registers' contents)
…
$31: 0x(content in hexadecimal digits)
PC: 0x(content in hexadecimal digits)
IF: 0x(bit stream fetched) [to_be_stalled/to_be_flushed]
ID: (mnemonic) [to_be_stalled/fwd_EX-DM_rs/t_$x]
EX: (mnemonic) [fwd_EX-DM/DM-WB_rs/t_$x]
DM: (mnemonic)
WB: (mnemonic)
(2 lines here)

Note that

1. fwd_EX-DM/DM-WB_rs/t_$x means "forwarding from EX-DM/DM-WB for rs/t $x"
2. Comments are there for your understanding; they are not allowed in formal output files.
3. If both source operands (r_s, r_t) demand forwarding, then that for r_s goes first.

Knowing the format, you should also pay attention to these two requirements:

1. Output the contents of registers, fetched bit stream in IF stage and the instruction types (mnemonics) in other pipeline stages **before** executing instructions at each cycle.
2. Report all occurring hazards, i.e. stall, forwarding and flush, **right after** executing the instructions at each cycle.

The following are 5 examples to help you be familiar with format requirement.

Example 1:

Suppose that the iimage.bin describes the following program:

lw $2, 0($3)
or $3, $1, $4
beq $2, $3, some_line

```
and $1, $1, $0
…
```

We first observe the instructions executed in the pipeline for the first few cycles:

	IF	ID	EX	DM	WB	
cycle 0	lw	nop	nop	nop	nop	
cycle 1	or	lw	nop	nop	nop	
cycle 2	beq	or	lw	nop	nop	
cycle 3	and	beq	or	lw	nop	=> stall detected!
cycle 4	and	beq	nop	or	lw	=> forwarding detected!

The following is the content of snapshot.rpt after executing the above example.
(The content of registers are omitted)

snapshot.rpt of Example 1:

```
…

cycle 3
…(content of registers)
IF: 0x00200824 to_be_stalled
ID: BEQ to_be_stalled
EX: OR
DM: LW
WB: NOP


cycle 4
…(content of registers)
IF: 0x00200824
ID: BEQ fwd_EX-DM_rt_$3
EX: NOP
DM: OR
WB: LW


…
```

Example 2:

Suppose that the iimage.bin describes the following program:

```
lw $3, 0($2)
bne $1, $3, anyway
and $2, $5, $0
…
```

We first observe the instructions executed in the pipeline for the first few cycles:

	IF	ID	EX	DM	WB	
cycle 0	lw	nop	nop	nop	nop	
cycle 1	bne	lw	nop	nop	nop	
cycle 2	and	bne	lw	nop	nop	=> stall detected!
cycle 3	and	bne	nop	lw	nop	=> stall detected!
cycle 4	and	bne	nop	nop	lw	

The following is the content of snapshot.rpt after executing the above example.
(The content of registers are omitted)

snapshot.rpt of Example 2:

```
…

cycle 2
…(content of registers)
IF: 0x00A01024 to_be_stalled
ID: BNE to_be_stalled
EX: LW
DM: NOP
WB: NOP


cycle 3
…(content of registers)
IF: 0x00A01024 to_be_stalled
ID: BNE to_be_stalled
EX: NOP
DM: LW
WB: NOP
```

cycle 4
…(content of registers)
IF: 0x00A01024
ID: BNE
EX: NOP
DM: NOP
WB: LW

…

Example 3:

Suppose that the iimage.bin describes the following program:

```
lw $3, 0($2)
or $1, $3, $4
and $2, $5, $0
xor $7, $8, $9
…
```

We first observe the instructions executed in the pipeline for the first few cycles:

	IF	ID	EX	DM	WB	
cycle 0	lw	nop	nop	nop	nop	
cycle 1	or	lw	nop	nop	nop	
cycle 2	and	or	lw	nop	nop	=> stall detected!
cycle 3	and	or	nop	lw	nop	
cycle 4	xor	and	or	nop	lw	=> forwarding detected!

The following is the content of snapshot.rpt after executing the above example.
(The content of registers are omitted)

snapshot.rpt of Example 3:

…

cycle 2
…(content of registers)
IF: 0x00A01024 to_be_stalled
ID: OR to_be_stalled
EX: LW

DM: NOP
WB: NOP

cycle 3
…(content of registers)
IF: 0x00A01024
ID: OR
EX: NOP
DM: LW
WB: NOP

cycle 4
…(content of registers)
IF: 0x01093826
ID: AND
EX: OR fwd_DM-WB_rs_$3
DM: NOP
WB: LW

…

Example 4:

Suppose that the iimage.bin describes the following program:

```
addi $1, $2, 1
or $4, $2, $3
and $5, $1, $4
beq $1, $4, anywhere
sub $6, $7, $8
…
```

We first observe the instructions executed in the pipeline for the first few cycles:

	IF	ID	EX	DM	WB
cycle 0	addi	nop	nop	nop	nop
cycle 1	or	addi	nop	nop	nop
cycle 2	and	or	addi	nop	nop
cycle 3	beq	and	or	addi	nop

cycle 4	sub	beq	and	or	addi	=> forwarding detected!

The following is the content of snapshot.rpt after executing the above example.
(The content of registers are omitted)

snapshot.rpt of Example 4:

…

cycle 4
…(content of registers)
IF: 0x00E83022
ID: BEQ fwd_EX-DM_rt_$4
EX: AND fwd_DM-WB_rs_$1 fwd_EX-DM_rt_$4
DM: OR
WB: ADDI

…

Example 5:
Suppose that the iimage.bin describes the following program:

```
        beq $0, $0, target
        sub $6, $7, $8
target: and $2, $5, $0
        …
```

We first observe the instructions executed in the pipeline for the first few cycles:

	IF	ID	EX	DM	WB	
cycle 0	beq	nop	nop	nop	nop	
cycle 1	sub	beq	nop	nop	nop	=> flush detected!
cycle 2	and	nop	beq	nop	nop	

The following is the content of snapshot.rpt after executing the above example.
(The content of registers are omitted)

snapshot.rpt of Example 5:

…

cycle 1
…(content of registers)
IF: 0x00E83022 to_be_flushed
ID: BEQ
EX: NOP
DM: NOP
WB: NOP

cycle 2
…(content of registers)
IF: 0x00A01024
ID: NOP
EX: BEQ
DM: NOP
WB: NOP

…