VE370 Mid Review

2018-10-28

VE370 TA Group

UMJI-SJTU

Content

HW5 Sample Solution

HW 5. 1

```
sw R16,-100(R6)
```

4.16.1 (a) As this instruction executes, what is kept in each register located between two pipeline stages?

HW 5. 1

```
sw R16,-100(R6)
```

4.16.1 (a) As this instruction executes, what is kept in each register located between two pipeline stages?

- IF/ID: PC+4, instruction

HW 5. 1

```
sw R16,-100(R6)
```

4.16.1 (a) As this instruction executes, what is kept in each register located between two pipeline stages?

- IF/ID: PC+4, instruction
- ID/EX: PC+4, ReadData1($6), ReadData2($16), offset: -100(32b), Rs Rt field in this instruction, all control signals

HW 5. 1

```
sw R16,-100(R6)
```

4.16.1 (a) As this instruction executes, what is kept in each register located between two pipeline stages?

- IF/ID: PC+4, instruction
- ID/EX: PC+4, ReadData1($6), ReadData2($16), offset: -100(32b), Rs Rt field in this instruction, all control signals
- EX/MEM: PC+4+offset, ReadData2($16), control signals for WB and MEM, Rs Rt field in this instruction, ALU result and ALU zero

HW 5. 1

```
sw R16,-100(R6)
```

4.16.1 (a) As this instruction executes, what is kept in each register located between two pipeline stages?

- IF/ID: PC+4, instruction
- ID/EX: PC+4, ReadData1($6), ReadData2($16), offset: -100(32b), Rs Rt field in this instruction, all control signals
- EX/MEM: PC+4+offset, ReadData2($16), control signals for WB and MEM, Rs Rt field in this instruction, ALU result and ALU zero
- MEM/WB: Control sinals for WB, ReadData from MEM, ALUresult, Selected Reg Destination

HW 5. 2 & 5. 3

4.16.2 (a) Which registers need to be read, and which registers are actually read?

4.16.3 (a) What does this instruction do in the EX and MEM stages?

HW 5. 2 & 5. 3

4.16.2 (a) Which registers need to be read, and which registers are actually read?

---

need to be read: R16, R6; actually read: R16, R6

4.16.3 (a) What does this instruction do in the EX and MEM stages?

HW 5. 2 & 5. 3

4.16.2 (a) Which registers need to be read, and which registers are actually read?

---

need to be read: R16, R6; actually read: R16, R6

4.16.3 (a) What does this instruction do in the EX and MEM stages?

EX:

- Add the offset with the base address from R6
- Add PC+4 with the lower 16bits of instruction (useless)
- One reg destination is selected by mux (useless)

MEM:

- Write the value of R16 into the target address in data memory

HW 5. 4

```
LOOP:
  add $1, $2, $1
  lw  $2, 0($1)
  lw  $2, 16($2)
  slt $1, $2, $5
  beq $1, $9, LOOP
```

4.16.4(b) Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration).

HW 5. 4

Solution:

	add $1, $1, $2	WB					
	lw $1, 0($1)	MEM	WB				
	lw $1, 0($1)	EX	MEM	WB			
	beq $1, $0, LOOP	ID	EX	MEM	WB		
LOOP: (3rd)	lw $1, 0($1)	IF	ID	EX	MEM	WB	
	add $1, $1, $2		EX	ID	EX	MEM	WB
	lw $1, 0($1)			IF	ID	EX	MEM
	lw $1, 0($1)				IF	ID	EX
	beq $1, $0, LOOP					IF	ID
LOOP: (4th)	lw $1, 0($1)						IF

HW 5. 5 & 5. 6

4.16.5(b) How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

4.16.6(b) At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

HW 5. 5 & 5. 6

4.16.5(b) How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

`lw` fully takes use of all stages; `add` does no useful work in MEM; `beq` does no useful work in WB

4.16.6(b) At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

HW 5. 5 & 5. 6

4.16.5(b) How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

`lw` fully takes use of all stages; `add` does no useful work in MEM; `beq` does no useful work in WB
Only the first three clocks out of five are fully used.

4.16.6(b) At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

HW 5. 5 & 5. 6

4.16.5(b) How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

`lw` fully takes use of all stages; `add` does no useful work in MEM; `beq` does no useful work in WB
Only the first three clocks out of five are fully used.

4.16.6(b) At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

- Fetched instruction from 2nd iteration `beq $1, $0, LOOP`

HW 5. 5 & 5. 6

4.16.5(b) How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

`lw` fully takes use of all stages; `add` does no useful work in MEM; `beq` does no useful work in WB
Only the first three clocks out of five are fully used.

4.16.6(b) At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

- Fetched instruction from 2nd iteration `beq $1, $0, LOOP`
- PC+4

HW 5. 7

4.17.3 Problems in this exercise assume that instructions executed by a pipelined processor are broken down as follows:

Assuming there are no stalls, how often (percentage of all cycles) do we use the data memory?

HW 5. 7

4.17.3 Problems in this exercise assume that instructions executed by a pipelined processor are broken down as follows:

Assuming there are no stalls, how often (percentage of all cycles) do we use the data memory?

The percentage we use DMEM is just the sum of percentages of `sw` and `lw`.

a. 25%+5% = 30%
b. 20%+10% = 30%

HW 5. 8

4.18.1 (b)

HW 5. 8

4.18.1 (b)

- Ex: ALUop: 00; RegDst: 0; ALUSrc: 1

HW 5. 8

4.18.1 (b)

- Ex: ALUop: 00; RegDst: 0; ALUSrc: 1
- MEM: Branch: 0; MEMWrite: 0; MEMRead: 1

HW 5. 8

4.18.1 (b)

- Ex: ALUop: 00; RegDst: 0; ALUSrc: 1
- MEM: Branch: 0; MEMWrite: 0; MEMRead: 1
- MemtoReg: 0; RegWrite: 1

HW 5. 9

4.18.3 (b) What is the value of the PCSrc signal for this instruction? This signal is generated early in the MEM stage (only a single AND gate). What would be a reason in favor of doing this in the EX stage? What is the reason against doing it in the EX stage?

PCsrc $= 0$.

- Reason for in EX: one clock cycle earlier to select the correct result of pc (useful for branch instructions)
- Reason for in MEM: PCsrc is dependent on ALUzero and need further AND calculation. May exceed the cycle length.

HW 5. 10

Problems in this exercise refer to the following instruction sequences (b)

```
I1:  add $1, $2, $1
I2:  lw  $2, 0($1)
I3:  lw  $1, 4($1)
I4:  or  $3, $1, $2
```

4.20.1(a) Find all data dependences in this instruction sequence.

HW 5. 10

Problems in this exercise refer to the following instruction sequences (b)

```
I1:  add $1, $2, $1
I2:  lw  $2, 0($1)
I3:  lw  $1, 4($1)
I4:  or  $3, $1, $2
```

4.20.1(a) Find all data dependences in this instruction sequence.

- ReadAfterWrite: I1 - I2 I3 I4($1), I2 - I4($2), I3 - I4($1)

HW 5. 10

Problems in this exercise refer to the following instruction sequences (b)

```
I1:  add $1, $2, $1
I2:  lw  $2, 0($1)
I3:  lw  $1, 4($1)
I4:  or  $3, $1, $2
```

4.20.1(a) Find all data dependences in this instruction sequence.

- ReadAfterWrite: I1 - I2 I3 I4($1), I2 - I4($2), I3 - I4($1)
- WriteAfterRead: I1 - I2($2), I1 - I3($1)

HW 5. 10

Problems in this exercise refer to the following instruction sequences (b)

```
I1:  add $1, $2, $1
I2:  lw  $2, 0($1)
I3:  lw  $1, 4($1)
I4:  or  $3, $1, $2
```

4.20.1(a) Find all data dependences in this instruction sequence.

- ReadAfterWrite: I1 - I2 I3 I4($1), I2 - I4($2), I3 - I4($1)
- WriteAfterRead: I1 - I2($2), I1 - I3($1)
- WriteAfterWrite: I1 - I3($1)

HW 5. 11

```
add $1, $2, $1
lw  $2, 0($1)
lw  $1, 4($1)
or  $3, $1, $2
```

4.20.3(a) To reduce clock cycle time, we are considering a split of the MEM stage into two stages. Find all hazards in this instruction sequence for this 6-stage pipeline with and then without forwarding.

The only difference is that originally load-use harzard only affects the very next instruction with RAW dependency while in this 6-stage version, it will affect next 2 intructions.

- With Forwarding: I2 - I4($2) (not exists in 5-stage), I3 - I4($2)
- Without Forwarding: I1 - I2 I3($1), I2 - I4($2), I3 - I4($1)

HW 5. 12

We assume that all values in data memory are zeroes and that registers have the following initial values:

	R0	R1	R2	R3
a.	0	–1	31	1500
b.	0	4	63	3000

```
add $1, $2, $1
lw  $2, 0($1)
lw  $1, 4($1)
or  $3, $1, $2
```

4.20.5(a) If we assume forwarding will be implemented when we design the hazard detection unit, but then we forget to actually implement forwarding what are the final register values after this instruction sequence?

Without forwarding, the written reg can only be updated after two cycles of the execution.

$$\$1 = 0;\ \$2 = 0;\ \$3 = 30 \vee 31 = 31_{dec}$$

HW 5. 13

	R0	R1	R2	R3
a.	0	–1	31	1500
b.	0	4	63	3000

```
add $1, $2, $1
lw  $2, 0($1)
lw  $1, 4($1)
or  $3, $1, $2
```

4.20.6(a) Add NOPs sequence to ensure correct execution in spite of missing support for forwarding.

```
add $1, $2, $1
NOP
NOP
lw  $2, 0($1)
lw  $1, 4($1)
NOP
NOP
```

Review

A few more exercise