CAO HW4

Remy Jaspers - 4499336

March 18, 2017

1a

The lower 16 bits are put into the sign extender, which in this instruction is:
0000 1000 0010 0101.
The sign bit is copied to fill the remaining 16 bits, thus the output of the sign extend unit will be:
0000 0000 0000 0000 0000 1000 0010 0101.
At the input of the Adder, this value will be shifted left two bits, thus the value will become:
0000 0000 0000 0000 0010 0000 1001 0100

The shift left 2 unit will take the lower 26 bits of the instruction and shift these left two bits. The input for this unit is:
1 1000 0010 0000 1000 0010 0101
Shifted left twice to produce:
1 1000 0010 0000 1000 0010 0101 00

This will be the output of the shift left 2 unit. The Jump address will be calculated from concatenating PC+4 to this address. The current PC value is not given so we cannot calculate the upper four bits.

1b

From figure 4.22, p. 269:

RegDst	1
ALUSRc	0
MemtoReg	0
RegWrite	1
MemRead	0
MemWrite	0
Branch	0
ALUOp1	1
ALUOp2	0

1c

As this is an R-format instruction, the new PC value will just be PC+4.

1d

The instruction fetched from the instruction memory is:

```
or $at, $t4, $v0
```

The values in these registers are provided. $t4 = R12 = 16 and $v0 = R2 = -128. These are the inputs to the main ALU.
16 | -128 = -112 = 0xFFFFFF90 = 1111 1111 1111 1111 1111 111 1001 0000.

The inputs for the Adder are (as given in 1a) 0x00002094, from the shift left 2 unit. The other input comes from the PC+4. Thus the adder performs the calculation 0x00002094 + (PC + 4).

2a

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
LW R2, 0(R1)	IF	ID	EX	MEM	WB								
NOP													
NOP													
AND R1, R2, R1				IF	ID	EX	MEM	WB					
LW R3, 0(R2)					IF	ID	EX	MEM	WB				
LW R1, 0(R1)						IF	ID	EX	MEM	WB			
NOP													
NOP													
SW R1, 0(R2)									IF	ID	EX	MEM	WB

The AND instruction needs to wait for the write back stage to occur before it can compute the result using the ALU. Two NOPS are necessary here, because we assume no forwarding unit is available. Same for the store word instruction. We need to wait for the previous instructions write back to finish. Again two nops are inserted. The total number of cycles needed is 13.

2b

It is not possible to reorder this code to reduce the number of stalls. There are too many dependencies.

2c

Only 9, instead of 13 cycles will be needed because there won't be NOPS inserted, as there's no hazard detection unit. In this way however, data integrity cannot be guaranteed for the dependency on the R1 register in the AND instruction and the preceding LW instruction. As this is a load-use hazard, we can only forward this from MEM to EX but this is impossible to do in the same cycle.

The third instruction (LW R3, 0(R2)) will be correct as there's no dependency on registers used in the previous instruction. There is a hazard again between the fourth LW and SW instruction. Again, the data in R1 cannot be guaranteed to be correct when the SW instruction is executed, because we can only forward from MEM, as in the previous case.

2d

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
LW R2, 0(R1)	IF	ID	EX	MEM	WB					
AND=>NOP		IF	ID	X	X	X				
AND R1, R2, R1			IF	ID	EX	MEM	WB			
LW R3, 0(R2)					IF	ID	EX	MEM	WB	
LW R1, 0(R1)						IF	ID	EX	MEM	WB
ForwardA	00	00	00	01	00					
ForwardB	00	00	00	00	00					
Hazard Detection Unit	0	0	1	0	0					

In cycle 3, the hazard detection unit finds that the ID/EX stage needs the output of MEM/WB from the previous instruction. In that cycle, ID/EX.registerRS == MEM/WB.registerRT and it enables the multiplexer for forwarding from MEM/WB to ID/EX so that the value will be available.The AND then becomes a NOP allowing for R2 of the LW to settle before use in the EX stage of the AND in cycle 5.

10 cycles are necessary to complete this code, as one NOP is needed to forward data from MEM of LW to the EX stage of the AND instruction.