SYSC 5704 - ELEMENTS OF COMPUTER SYSTEMS

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGG.

ASSIGNMENT - 4

NAME : AARTHI THIRUMAVALAVAN

STUDENT NO. : 100958800

DATE : 13-11-2014

Question 4.3:

	COST
1. Instruction memory	1000
Registers	200
ALU	100
Data memory	2000
ADD	60
MUX	30
control	500
Total Cost	3890

a) Clock cycle time is determined by the critical path. For the given condition, the critical path is

PC → Instruction memory → Registers → MUX → ALU → D Mem → Mux.

The latency of this path is 400+200+30+120+350+30 = 1130 ps

After improvement, the latency is 1130 + 300 = 1430 ps

b) while no direct speed up occurs in the critical path, and in fact, the cycle time is lengthened since it adds MUL to the instruction set, 5% fewer instructions can be performed. So, if we assume 1000 instructions at 1130 ps, we have run time of 1,130,000 ps. After improvement, we have 950 instructions at 1430 ps. we have a speed up of $\left(\frac{1}{0.95}\right) \times \left(\frac{1130}{1140}\right) = 0.83$ and we have a run time of 1,358,500. Consequently, we have a perform decrease. of 225800 ps, instead of ~~decrease~~ increase. The increase in cycle time is not recovered in decrease in instruction count.

c) The cost is always the total cost of all the components (not just the components on critical path) so the original processor has a cost of I-Mem, Regs, Control, ALU, D-Mem, 2 Add Units and 3 Mux Units, for a total cost of $1000 + 200 + 500 + 100 + 2000 + 2 \times 30 + 3 \times 10 = 3890$.

We will compute cost relative to this baseline. The performance relative to this baseline is the speed up computed in part (b) ie., 0.83 and our cost / performance relative to the baseline is as follows:

new cost => 3890 + 600 = 4490

relative cost => 4490 / 3890 = 1.15

1.15 / 0.83 = 1.39

We are paying significantly more for worser performance. So the cost/performance is a lot worse than the modified, so the cost / performance improves.

Question : 4.8

a) Pipelining reduces the cycle time to the length of the longest stage plus the register delay.

CT = 300 + 50 = 350 ps ⟶ CT for pipelined processor

Because there is no pipelining, the cycle time must allow an instruction to go through all stages in one cycle.

CT = 250 + 250 + 150 + 300 + 200 = 1150 ps ⟶ CT for non-pipelined processor.

b) Total latency for pipelined processor:

Latency becomes CT * N where N is the number of stages as one instruction will need to go through each of the stages and each stage takes on cycle.

Latency = 5 * 320 = 1500 ps.

Total latency for non pipelined processor.

The latency is the same as cycle time since it takes the instruction one cycle to go from the beginning of fetch to the end of write back.

Latency = 1150 ps

c) We would want to choose the longest stage to split in half. The new cycle time becomes the originally and longest stage length.
We will split the memory stage. So the new cycle time will be CT = 300 ps = 300 ps.

d) Assume the distribution ~~function~~ of instructions that run on the processor is :

45% : ALU
20% : BEQ
20% : LW
15% : SW

LW and SW instructions uses the data memory. As a result, the utilization of the data memory is
20% + 15% = 35%

e) Similarly, ALU and LW instructions use the register blocks write port. As a result, the utilization of the registers blocks write port is 45% + 20% = 65%.

(f) Instead of a single-cycle organization, we can use a multicycle organization where each instruction takes multiple cycles but one instruction finishes before another is fetched. In this organization, an ~~int~~ instruction only goes through stages it actually needs (eg., ST only takes 4 cycles because it doesn't need the WB stage). Compare clock times and execution times with single-cycle, multi-cycle

Solution :

Clock time for pipelined = 300 ps.

Clock time for non-pipelined : 1150 ps.

Multi cycle has the same clock cycle time as the pipelined organization and it is = 300 ps.

In single-cycle, every instruction takes one clock cycle and in pipelined a long running program with no pipeline stalls completes one instruction in every cycle.

Therefore, a multicycle organization completes a lw in 5 cycles, a sw in 4 cycles, an alu in 4 cycles and a beq in 4 cycles.

Multi cycle execution time is x times pipelined execution time where x = 0.20 x 5 + 0.80 x 4 = ~~4.00~~ 4.20

Single cycle execution time is x times pipelined execution time where value of x = 1150 ps / 300 ps = 3.83

Question 4.9:

a)

Instruction sequence	Dependences
I1: OR R1, R2, R3	RAW on R1 from I1 to I2 & I3
I2: OR R2, R1, R4	RAW on R2 from I2 to I3
I3: OR R1, R1, R2	WAR on R2 from I1 to I2
	WAR on R1 from I2 to I3
	WAW on R1 from I1 to I3

(b) In basic 5 stage pipeline WAR and WAW, dependences do not cause any hazards. Without forwarding, any RAW dependance between an instruction and the next two instructions (if register read happens in the first half). The code that eliminates these hazards by inserting NOP instruction is:

Instruction seq	Delay
OR R1, R2, R3	
NOP	Delay I2 to avoid RAW hazard on R1 from I1
NOP	
OR R2, R1, R4	
NOP	Delay I3 to avoid RAW hazard on R2 from I2.
NOP	
OR R1, R1, R2	

(c) With full forwarding, an ALU instruction can forward a value to the EX stage of the next instruction without a hazard. However, a load cannot forward to the EX stage of the next instruction (but can to the instruction after that).

The code that eliminates these hazards by inserting NOP instruction is:

Instruction sequence

```
OR  R1, R2, R3
OR  R2, R1, R4
OR  R1, R1, R2
```

No rawhazard on R1 from I1
No rawhazaid on R2 from I2 (forwarded)

(d) The total execution time is the clock cycle time times the number of cycles. Without any stalls a 3-instruction sequence executes in 7 cycles (5 to complete first ~~equation~~ instruction, then one per instruction). The execution without forwarding must add a stall for every NOP we had in (b) and execution forwarding must add a stall cycle for every NOP we had in (c). overall we get:

without forwarding => (7+2) x 250 ps = 2250 ps

with forwarding => (7+1) x 300 ps = 2400 ps

Speedup due to forwarding = 0.94 (which is a slowdown)

(e) with ALU-ALU only forwarding, an ALU instruction can forward to the next instruction but not to the second next instruction. A load can forward at all, because it determines the data value in MEM stage, when its too late for ALU-ALU forwarding. we have

Instruction sequence

```
OR  R1, R2, R3
OR  R2, R1, R4      ALU-ALU forwarding R2 from I1
OR  R1, R1, R2      ALU-ALU forwarding R2 from I2
```

f) no forwarding ⇒ (7+2) × 250 ps = 2250 ps

with ALU-ALU forwarding ⇒ (7 × 290 ps) = 2030 ps

Question H.18:

Question H.18

(b)

```
ADD R5,R0,R0      IF ID EX ME  WB
BEQ R5,R6,End     IF ID ** EX  ME WB
ADD R10,R5,R1        IF ** ID  EX ME WB
LW  R11,0(R10)       IF ** ID  ** EX ME WB
LW  R10,1(R10)          IF ** ID ** EX ME WB
SUB R10, R11,R10           IF ** ID ** ** EX  ME WB
ADD R11,R5,R2                 IF ** ** * ID  EX ME WB
SW  R10,0(R11)                IF ** ** * ID  ** EX ME WB
ADDI R5, R5, #2                          IF ** ID EX ME WB
BEW  R0, R0, again                       IF ** ID ** EX ME WB
BEQ  R5, R6, End                            IF ** ID EX ME WB
```

(c) The only way to execute two instructions fully in parallel is for a load/store to execute together with another instruction. To achieve this around each L/S instruction, we will try to put on non-L/S instruction that have no dependencies with L/S.

```
        ADD     R5, R0, R0
Again:  ADD     R10, R5, R1
        BEQ     R5, R6, End
        LW      R11, 0(R10)
        ADD     R12, R5, R2
        LW      R10, 1(R10)
        ADDI    R5, R5, 2
        SUB     R10, R11, R10
        SW      R10, 0(12)
        BEQ     R0, R0, Againd
End:
```

(a) CPI for issue 1 =) 1.11 (10 cycles per instructions). There is 1 stall cycle in each iteration due to data hazard between the second LW and the next instruction (SUB).

CPI for ISSUE 2 =) 1.06 (19 cycles per 18 instructions). Neither of the 2 LW instructions can execute in parallel with another instruction and SUB stalls because it depends on the second LW. The SW instruction executes in parallel with ADDI in even-numbered iterations.

Speedup = 1.05

(e) CPI for 1-issue = 1.11

CPI for 2-issue = 0.83 (15 cycles per 18 instructions)

In all iteration, SUB is stalled because it depends on the second LW. The only instructions that execute in odd numbered iterations as a pair are ADDI and BEQ. In even numbered iterations, only the two LW instructions cannot execute as a pair.

Speed up = 1.34
