Pathri Vidya Praveen
CS24BTECH11047

Homework 4

1. A six-stage pipelined processor has delays of 130, 125, 110, 132, 135, and 145 picoseconds for each pipeline stage. The registers used between pipeline stages have a delay of 12 picoseconds each. [3 + 2 marks]

Sol:

T = clock cycle time = slowest stage delay + pipeline register delay
slowest stage delay = max(130,125,110,132,135,145) = 145 ps
pipeline register delay = 12 ps
T = 157 ps = clock cycle time

a. Find the total time to execute 1000 independent instructions on this pipelined processor, assuming there are no pipeline stalls/hazards.

Sol:

Total cycles = Number of stages + Number of instructions - 1 = 6 + 1000 - 1 = 1005
Total time = Total cycles x Clock cycle time = 1005 x 157 = 157785 ps

b. Find the total time to execute 1000 independent instructions on this pipelined processor, assuming that 20% of instructions incur a 1-cycle stall.

Sol:

Given that 20% of instructions incur a 1-cycle stall
Number of stalls = 20% of number of instructions = 0.2 x 1000 = 200
Total cycles = 1005 + number of stalls = 1005 + 200 = 1205
Total time = total cycles x clock cycle time = 1205 x 157 = 189185 ps

2. Consider a 5-stage pipelined processor P1 with a total latency of 800 ps, equally divided between the pipeline stages. P2 is an improved version of P1, which improves the ALU by supporting MULT instruction. Including the MULT instruction increases the ALU latency by 250 ps, with the latency of other stages being unchanged. Presence of MULT instruction also reduces the total number of instructions by 10% (as we can directly use MULT instead of realizing it through repeated ADD). Answer the following: [2 + 2 + 1 marks]

a. What is the minimum required clock period for processors P1 and P2?

Sol:

P1 has 5 stages with total latency of 800 ps equally divided with stage latency of 160 ps.
Clock period determined by slowest stage
Minimum required clock period for processor P1 = 160 ps

P2 improves ALU by adding a MULT instruction increasing its latency by 250 ps keeping others unchanged.
ALU latency = 160 ps + 250 ps = 410 ps
Minimum required clock period for processor P2 = max(160,410,160,160,160) = 410 ps

b. Which processor (P1 or P2) would execute a given code in a smaller time?

Sol:

n = number of instructions in given code for P1
90% of n = 0.9n = number of instructions in given code for P2
Total time = (number of stages + number of instructions - 1) x clock cycle time
Total time for P1 = (5 + n - 1) x 160 = (160n + 640) ps
Total time for P2 = (5 + 0.9n - 1) x 410 = (369n + 1640) ps

Total time for P1 < Total time for P2 for all n >= 1
P1 processor executes a given code in a smaller time

c. If a piece of code has 5000 instructions on P1, find the total execution time of this code on P1 and P2.

Sol:

Given n = 5000 .

Total execution time of code on P1 = (160 x 5000 + 640) ps = 800640 ps
Total execution time of code on P2 = (369 x 5000 + 1640) ps = 1846640 ps

---

3. Consider the code given below:
(Added instruction numbers for each instructions)

1 → add x14, x12, x11
2 → add x15, x14, x12
3 → ld x13, 8(x13)
4 → ld x12, 0(x14)
5 → and x13, x15, x13
6 → ld x11, 4(x13)
7 → sd x13, 0(x15)

a. Insert NOPs for correct execution (assuming no forwarding or hazard detection is implemented). [3 marks]

Sol:

Without forwarding , RAW hazards require stalling until the producer instruction writes back to the register file. The pipeline stages are IF, ID, EX, MEM, WB. Total of 5 NOPs are inserted.

The below NOPs are inserted for correct execution.
2 NOPs between instructions 1 and 2 due to dependency on x14.
1 NOP between instructions 4 and 5 due to dependency on x13 from instruction 3
2 NOPs between instructions 5 and 6 due to dependency on x13.

Final code after NOPs insertion:

```
add x14 , x12 , x11
NOP
NOP
add x15 , x14 , x12
ld x13 , 8(x13)
ld x12 , 0(x14)
NOP
and x13 , x15 , x13
NOP
NOP
ld x11 , 4(x13)
sd x13 , 0(x15)
```

b. Insert NOPs for correct execution (assuming data forwarding without hazard detection is implemented). [2 marks]

Sol:

With data forwarding , results from EX, MEM,WB stages can be forwarded to the EX stage of dependent instructions. This eliminates the need for stalls in most cases ,except for load-use hazards where the consumer immediately follows the load. But in this code , no such consecutive load use hazards occur. So , no NOPs are required.

Final code after NOPs insertion:

```
add  x14 , x12 , x11
add x15 , x14 , x12
ld x13 , 8(x13)
ld x12 , 0(x14)
and x13 , x15 , x13
ld x11 , 4(x13)
sd x13, 0(x15)
```

---

4. Consider the code given below:

1 → add x14, x12, x11
2 → L2: beq x15, x14, L1
3 → ld x13, 8(x13)
4 → beq x12, x13, L2
5 → L1: and x13, x15, x13
6 → ld x11, 4(x13)
7 → sd x13, 0(x15)

a. Insert NOPs for correct execution (assuming no forwarding or hazard detection is implemented). [3 marks]

Sol:

A result must be written to the register file in WB stage before a subsequent instruction can read it. Branch decisions are made later in the pipeline stage (EX stage). Total of 10 NOPs are inserted.

Final code:

```
add x14,x12,x11 // data hazard as beq needs x14
NOP
NOP
L2: beq x15,x14,L1 // control hazard as branch decision is not known for 2 cycles
NOP
NOP
ld x13,8(x13) // load-use hazard
NOP
NOP
beq x12,x13,L2 // control hazard
NOP
NOP
L1: and x13,x15,x13 // data hazard as ld needs x13 for its address
NOP
NOP
ld x11,4(x13)
sd x13,0(x15)
```

b. Insert NOPs for correct execution (assuming data forwarding without hazard detection is implemented, and branch comparison happens in 2nd stage). [2 marks]

Sol:

Total of 5 NOPs are inserted.

```
add x14,x12,x11
NOP // stall for data hazard
L2: beq x15,x14,L1
NOP // stall for beq control hazard
ld x13, 8(x13)
NOP // 2 stalls for ld to beq data hazard
NOP
beq x12,x13,L2
NOP // stall for beq control hazard
L1: and x13,x15,x13
ld x11 , 4(x13)
sd x13,0(x15)
```