Tutorial 8 – Week of Nov. 8

Question 4.16

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies: In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250 ps	350 ps	150 ps	300 ps	200 ps

Also, assume that instructions executed by the processor are broken down as follows:

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

4.16.1 What is the clock cycle time in a pipelined and non-pipelined processor?
4.16.2 What is the total latency of an ld instruction in a pipelined and non-pipelined processor?
4.16.3 If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?
4.16.4 Assuming there are no stalls or hazards, what is the utilization of the data memory?
4.16.5 Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

Solution

4.16

4.16.1 Pipelined: 350; non-pipelined: 1250

4.16.2 Pipelined: 1250; non-pipelined: 1250

4.16.3 Split the ID stage. This reduces the clock-cycle time to 300ps.

4.16.4 35%.

4.16.5 65%

Question 4.20

Add NOP instructions to the code below so that it will
run correctly on a pipeline that does not handle data hazards.

```
addi x11, x12, 5
add  x13, x11, x12
addi x14, x11, 15
add  x15, x13, x12
```

Solution

4.20

```
addi x11, x12, 5
NOP
NOP
add x13, x11, x12
addi x14, x11, 15
NOP
add x15, x13, x12
```

Question 4.22

Consider the fragment of RISC-V assembly below:

```
sd x29, 12(x16)
ld x29, 8(x16)
sub x17, x15, x14
beqz x17, label
add x15, x11, x14
sub x15, x30, x14
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

4.22.1 Draw a pipeline diagram to show were the code above will stall.

4.22.2 In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

4.22.3 Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding NOPs to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

4.22.4 Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Use the instruction mix from Exercise 4.8.)

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

Solution

4.22

4.22.1 Stalls are marked with **:

```
sd  x29, 12(x16)  IF ID EX ME WB
ld  x29, 8(x16)      IF ID EX ME WB
sub x17, x15, x14       IF ID EX ME WB
bez x17, label             ** ** IF ID EX ME WB
add x15, x11, x14                   IF ID EX ME WB
sub x15,x30,x14                        IF ID EX ME WB
```

4.22.2 Reordering code won't help. Every instruction must be fetched; thus, every data access causes a stall. Reordering code will just change the pair of instructions that are in conflict.

4.22.3 You can't solve this structural hazard with NOPs, because even the NOPs must be fetched from instruction memory.

4.22.4 35%. Every data access will cause a stall.

Question 4.25

Consider the following loop.

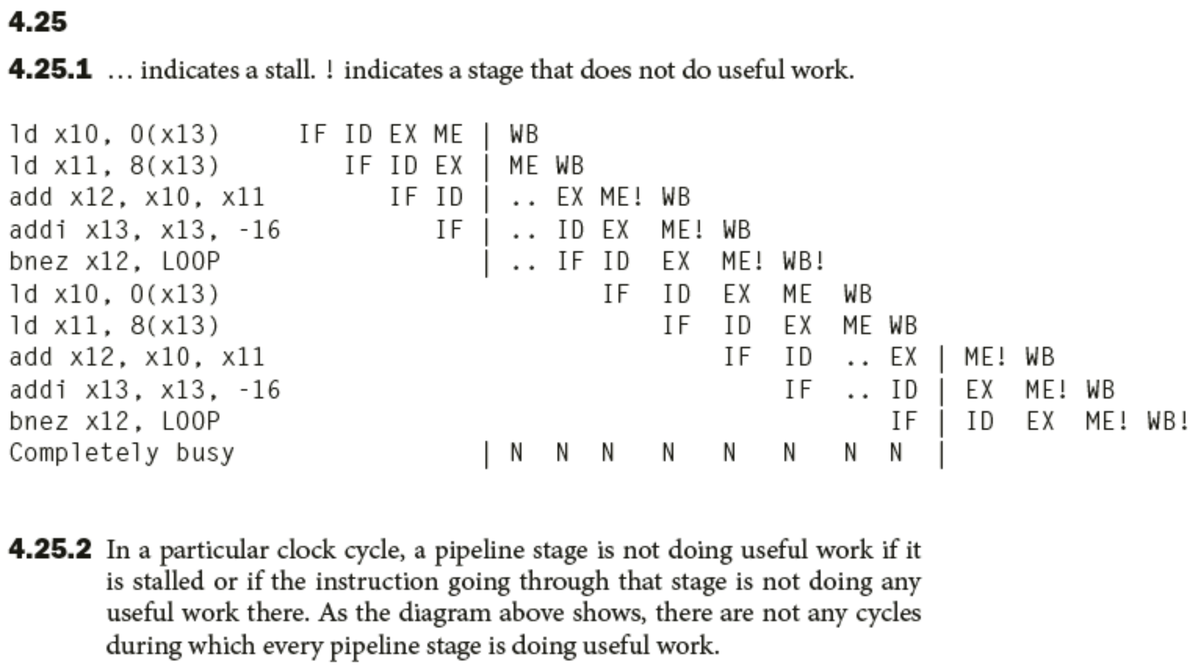
```
LOOP: ld   x10, 0(x13)
      ld   x11, 8(x13)
      add  x12, x10, x11
      subi x13, x13, 16
      bnez x12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

4.25.1 Show a pipeline execution diagram for the first
two iterations of this loop.

4.25.2 Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `subi` is in the IF stage. End with the cycle during which the `bnez` is in the IF stage.)

Solution



4.25

4.25.1 … indicates a stall. ! indicates a stage that does not do useful work.

```
ld x10, 0(x13)      IF ID EX ME | WB
ld x11, 8(x13)         IF ID EX | ME WB
add x12, x10, x11         IF ID | .. EX ME! WB
addi x13, x13, -16           IF | .. ID EX  ME! WB
bnez x12, LOOP                  | .. IF ID  EX  ME! WB!
ld x10, 0(x13)                          IF  ID  EX  ME  WB
ld x11, 8(x13)                              IF  ID  EX  ME WB
add x12, x10, x11                               IF  ID  .. EX | ME! WB
addi x13, x13, -16                                  IF  .. ID | EX  ME! WB
bnez x12, LOOP                                             IF | ID  EX  ME! WB!
Completely busy                 | N  N  N   N   N   N   N  N  |
```

4.25.2 In a particular clock cycle, a pipeline stage is not doing useful work if it is stalled or if the instruction going through that stage is not doing any useful work there. As the diagram above shows, there are not any cycles during which every pipeline stage is doing useful work.