

Tutorial - 10

Problem-I

Filter out the instructions that may lead to control hazards and data hazards in a five stage pipelined processor following RISC-V ISA.

Instructions					
JAL	LW	ADD	SW	BEQ	SUB

Solution

Data hazards: There is no single instruction which is responsible for a data hazard. The data hazards are always a cause due to two or more instructions being dependent on each other.

(a) Processor has no data forwarding capability

(i)

ADD x2,x1,x0	ADD x2,x1,x0	LW x2,8(x0)	ADD x3,x2,x1
ADD x3,x2,x0	SUB x3,x2,x0	ADD x3,x2,x1	SW x3,12(x0)

(b) Processor has full hardware forwarding capability

(i) In this only Load-delay slot is able to produce a data hazard.

LW x2,8(x0)
ADD x3,x2,x1

Control Hazards: The instructions which lead to change in unintentional control flow are responsible for control hazards. These are individual instructions, no two instructions are required to create a control hazard. For example jump to an unpredicted address, unpredictable branch condition.

Instructions	
JAL	BEQ

Problem-II

Consider two different machines. The first has a single-stage, non-pipelined machine with a cycle time of 15 ns. The second is a pipelined machine with 5 pipeline stages and a cycle time of 3ns.

- What is the speedup of the pipelined machine versus the single-cycle machine, assuming there are no stalls?
- What is the speedup of the pipelined machine versus the single cycle machine if the pipeline stalls 1 cycle for 25% of the instructions?
- Now consider a 4-stage pipeline machine with a cycle time of 3.1 ns. Again, assuming no stalls, is this implementation faster or slower than the original 5-stage pipeline? Explain your answer.

Solution:

- a. The speedup is $15\text{ ns} / 3\text{ ns} = 5$
- b. New CPI = $1 + .25 \times 1 = 1.25$

Since the number of instructions is the same, the speedup is

$$\frac{CPI(\text{old}) \times \text{Cycletime}(\text{old})}{CPI(\text{new}) \times \text{Cycletime}(\text{new})} = \frac{1 \times 15}{1.25 \times 3} = 4$$

- c. The 5-stage machine is faster. This is because it has a shorter cycle time, which results in a faster overall execution time (since there are no stalls, they both have the same CPI).

Problem-III

You need to reorganize or optimize the below-mentioned assembly program such that the functionality of the program remains unchanged and the total execution time is minimum. Assume the processor chosen for the execution is 5-stage and follows RISC-V ISA and has full data forwarding capability.

Assembly Program

```
0x00: Addi x4,x0,#4
0x04: Sub x5,x3,x4
0x08: Lw x6,8(x5)
0x0C: Add x7,x6,x0
0x10: Lw x8,12(x5)
0x14: Add x9,x8,x0
0x18: Addi x3,x0,#3
0x1C: Add x9,x9,x3
```

Solution

The thumb rule to optimize any program or to reduce the execution time is to fill the pipeline such that there are minimum stalls. This can be done by reducing the dependency among subsequent or closely placed instructions (which can cause the subsequent instruction to stall). In the above example we have two load delay slots. If we are able to fill them with independent instructions the program will be optimized.

One possible solution after instruction reorganization can be:

```
0x00: Addi x4,x0,#4
0x04: Sub x5,x3,x4
0x08: Lw x6,8(x5)
0x0C: Lw x8,12(x5)
0x10: Addi x3,x0,#3
0x14: Add x7,x6,x0
0x18: Add x9,x8,x0
0x1C: Add x9,x9,x3
```

Problem-IV

Execute the below mentioned assembly program and explain the use of stack memory in this program.

Assembly Program

```
Addi x4,x0,#4
Addi x3,x1,x0
```

```

Add x5,x3,x1
Sub sp,sp,x4
Sw x3,0(sp)
Add x3,x5,x5
Add x5,x5,x0
Lw x3,0(sp)
Add sp,sp,s4
Beq x0,x0,#0

```

Solution

The stack memory is used at run-time to store the variables or content of CPU registers into a specific memory called stack. The stack memory is different from Heap Memory. In the above assembly program stack is being used to temporarily store the content of register x3 and then to retrieve its content when the operation on register x5,x3 is completed.

Problem-V

Execute the below-mentioned assembly program on a processor having standard five-stages and following RISC-V ISA. And find the difference in total number of execution cycles among the two cases

- (a) All branches are predicted correctly
- (b) All branches are predicted incorrectly

Assembly Program

```

Addi x1,x0,#1
Addi x10,x0,#10

```

DO_AGAIN:

```

Sub x10,x10,x1
Beq x10, x0, DO_AGAIN
Beq x0,x0,#0

```

Solution:

- (a) All branches are predicted correctly:

In this case there will be no stalling and the pipeline will be completely filled. Let x be the total number of cycles taken by the program to complete the execution.

- (b) All branches are predicted incorrectly:

Here we have total (10+1) branches to be predicted. In our standard five-stage processor one branch miss prediction will lead to a branch penalty of two cycles. So, the total penalty will be $11 * 2 = 22$ cycles. The total number of cycles taken for complete program execution is $x+22$ cycles.