Indian Institute of Technology (IIT-Bombay)

AUTUMN Semester, 2025
COMPUTER SCIENCE AND ENGINEERING

CS230: Digital Logic Design and Computer Architecture

Tutorial - IV

Full Marks: 0

Time allowed: ∞ hours

1. Consider a processor with 128 registers and an instruction set of size 20. Each instruction has five distinct fields, namely, opcode, two source register identifiers, one destination register identifier, and a sixteen-bit immediate value. Each instruction must be stored in memory in a byte-aligned fashion. If a program has 250 instructions, the amount of memory (in bytes) consumed by the program text is ___________.

Solution:

Registers $= 128 \Rightarrow$ need 7 bits to uniquely identify each (since $2^7 = 128$).

Instruction set size $= 20 \Rightarrow$ need 5 bits for the opcode (since $2^5 = 32 > 20$).

Each instruction has 5 fields:

$$\begin{aligned}\text{Opcode} &= 5 \text{ bits},\\ \text{Source register 1} &= 7 \text{ bits},\\ \text{Source register 2} &= 7 \text{ bits},\\ \text{Destination register} &= 7 \text{ bits},\\ \text{Immediate value} &= 16 \text{ bits}.\end{aligned}$$

Total bits per instruction:

$$5 + 7 + 7 + 7 + 16 = 42 \text{ bits}.$$

Each instruction = 42 bits.
Since it needs to be stored in a byte-aligned fashion, we must round up to the nearest byte:

$$\left\lceil \frac{42}{8} \right\rceil = \lceil 5.25 \rceil = 6 \text{ bytes per instruction.}$$

Total instructions = 250.

Hence, total memory used:

$$250 \times 6 = \boxed{1500 \text{ bytes.}}$$

2. Suppose the functions H and K can be computed in 5 and 3 nanoseconds by functional units U_H and U_K, respectively. Given two instances of U_H and two instances of U_K, it is required to implement the computation

$$H(K(X_i)) \quad \text{for } 1 \leq i \leq 10.$$

Ignoring all other delays, the minimum time required to complete this computation is ___________ nanoseconds.

Solution

The same concept is used in pipelining. Bottleneck here is U_H as it takes 5 ns while U_K takes 3 ns only. We have to do 10 such calculations and we have 2 instances of U_H and U_K respectively.

So, U_H can be done in

$$\frac{50}{2} = 25 \text{ nano seconds.}$$

For the start, U_H needs to wait for U_K output for 3 ns and rest all are pipelined and hence no more waiting is needed.

So, the answer is

$$3 + 25 = 28 \text{ ns.}$$

3. Consider a 5-stage instruction pipeline consisting of the following stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Each stage takes exactly one clock cycle to complete. The processor does not support forwarding, and all data hazards are resolved by stalling. Branch instructions are resolved during the ID stage, and each branch incurs a one-cycle stall due to control hazards. The pipeline handles all types of hazards by stalling.

You are given the following sequence of instructions:

```
I1:  LW R1, 0(R2)
I2:  ADD R3, R1, R4
I3:  SUB R5, R1, R6
I4:  BEQ R3, R7, LABEL
I5:  AND R8, R5, R9
```

(a) Draw the pipeline diagram cycle-by-cycle (up to when the last instruction finishes).

(b) Indicate where stalls are inserted due to data or control hazards.

(c) Compute the total number of clock cycles required for execution.

(d) Calculate the speedup compared to non-pipelined execution, where each instruction takes 5 cycles and executes sequentially.

Solution

We have a 5-stage pipeline with stages IF, ID, EX, MEM, WB (one cycle each). There is *no forwarding*; all data hazards are resolved by stalling. Branches are resolved in ID and incur one extra cycle of control-stall. The instruction sequence is:

I1: LW R1, 0(R2)

I2: ADD R3, R1, R4

I3: SUB R5, R1, R6

I4: BEQ R3, R7, LABEL

I5: AND R8, R5, R9

Key observations (no forwarding):

- A load (I1) produces its register value only after its WB stage completes. Any dependent instruction must not enter EX until the producer's WB has finished.
- If a consumer would have reached EX earlier, we must insert stall cycles so that its EX is delayed until the producer's WB is done.
- Branch (I4) is resolved in ID and additionally causes one-cycle control stall.

(a) Pipeline diagram (cycle-by-cycle).

Below is a cycle-by-cycle pipeline diagram. Columns are cycles (1..14). Each row is one instruction. "S" denotes a stall (the instruction remains in ID and prevents IF from fetching a new instruction).

Cycle	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
I1	IF	ID	EX	MEM	WB									
I2		IF	ID	S	EX	MEM	WB							
I3					IF	ID	EX	MEM	WB					
I4							IF	ID	S	EX	MEM	WB		
I5										IF	ID	EX	MEM	WB

(b) Stalls due to hazards

- Data hazard (I1 $\to$ I2): 2-cycle stall in ID (C4–C5).
- Control hazard (I4): 1-cycle stall after ID stage (C9).

(c) Total number of clock cycles

Last instruction (I5) completes WB in cycle **14**.

$$\boxed{\text{Total cycles} = 14}$$

(d) Speedup over non-pipelined execution

Non-pipelined: each instruction = 5 cycles $\Rightarrow 5 \times 5 = 25$ cycles.

$$\text{Speedup} = \frac{25}{14} \approx 1.79$$

$$\boxed{\text{Speedup} \approx 1.79}$$

4. An instruction pipeline has five stages, namely, Instruction Fetch (*IF*), Instruction Decode and Register Fetch (*ID/RF*), Instruction Execution (*EX*), Memory Access (*MEM*), and Register Writeback (*WB*) with stage latencies of: *IF*: 1 ns, *ID/RF*: 2.4 ns, *EX*: 2.0 ns, *MEM*: 1.1 ns, and *WB*: 0.8 ns. To improve clock frequency, the designers split the *ID/RF* stage into three stages: *ID*, *RF1*, and *RF2*, each with latency $\frac{2.4}{3}$ ns. Similarly, the *EX* stage is split into two stages: *EX1* and *EX2*, each with latency 1 ns. Thus, the new design has a total of 8 pipeline stages. A program has 25% branch instructions, which execute in the *EX* stage (old design) and produce the next instruction pointer at the end of the stage. In the new design, the branch instruction computes the next instruction pointer only after the *EX2* stage. The *IF* stage stalls after fetching a branch instruction until the next instruction pointer is computed. All non-branch instructions have an average CPI of 1 in both designs. Let P and Q be the total execution times of the program on the old and new designs, respectively. The value of $\dfrac{P}{Q}$ is ___________.

Solution:

Execution time = cycle time × (branch instructions × $\text{CPI}_{\text{branch}}$ + non-branch instructions × $\text{CPI}_{\text{non-branch}}$).

CPI (cycles per instruction) = number of stall cycles + 1.

Given data:

Stage latencies (ns):

$$\text{IF} = 1.0, \quad \text{ID/RF} = 2.4, \quad \text{EX} = 2.0, \quad \text{MEM} = 1.1, \quad \text{WB} = 0.8.$$

The ID/RF stage is split into three stages: ID, RF1, RF2 each of latency 2.4/3 = 0.8 ns. The EX stage is split into EX1 and EX2 each of latency 1.0 ns. Branch fraction $b = 25\% = 0.25$. Branches produce the next-PC at the end of EX (old) and EX2 (new). IF stalls from fetch of a branch until the next-PC is available.

Explanation & Cycle time:

Old design clock period:

$$T_{\text{clk,old}} = \max\{1.0,\ 2.4,\ 2.0,\ 1.1,\ 0.8\} = 2.4 \text{ ns}.$$

New design stage latencies:

$$\text{IF} = 1.0,\ \text{ID} = 0.8,\ \text{RF1} = 0.8,\ \text{RF2} = 0.8,\ \text{EX1} = 1.0,\ \text{EX2} = 1.0,\ \text{MEM} = 1.1,\ \text{WB} = 0.8,$$

so

$$T_{\text{clk,new}} = \max\{1.0,\ 0.8,\ 0.8,\ 0.8,\ 1.0,\ 1.0,\ 1.1,\ 0.8\} = 1.1 \text{ ns}.$$

CPI:

Indexing stages starting at IF = 1:

- In the old design the branch resolves at the end of EX (stage index 3). IF is stage 1, so stall cycles per branch = 3 − 1 = 2. Thus
$$\text{CPI}_{\text{branch,old}} = 1 + 2 = 3.$$
- In the new design the branch resolves at the end of EX2 (stage index 6). IF is stage 1, so stall cycles per branch = 6 − 1 = 5. Thus
$$\text{CPI}_{\text{branch,new}} = 1 + 5 = 6.$$

Non-branch instructions have CPI = 1 in both designs.

The average CPI in each design can be written as

$$\mathrm{CPI}_{\mathrm{avg}} = (1-b)\cdot 1 \;+\; b\cdot(\mathrm{CPI}_{\mathrm{branch}}) = 1 + b\cdot(\text{stalls}).$$

Hence

$$\mathrm{CPI}_{\mathrm{old}} = 1 + 0.25\times 2 = 1.5, \qquad \mathrm{CPI}_{\mathrm{new}} = 1 + 0.25\times 5 = 2.25.$$

Calculation of execution times:

Let P and Q be total execution times on old and new designs respectively. Since execution time $\propto$ CPI $\times$ clock period,

$$P = \mathrm{CPI}_{\mathrm{old}}\, T_{\mathrm{clk,old}} = 1.5\times 2.4 = 3.6 \text{ (relative ns)},$$

$$Q = \mathrm{CPI}_{\mathrm{new}}\, T_{\mathrm{clk,new}} = 2.25\times 1.1 = 2.475 \text{ (relative ns)}.$$

Thus the ratio is

$$\frac{P}{Q} = \frac{1.5\times 2.4}{2.25\times 1.1} = \frac{3.6}{2.475} = \frac{18/5}{99/40} = \frac{16}{11} \approx 1.4545.$$

Result:

$$\boxed{\frac{P}{Q} = \frac{16}{11} \approx 1.45}$$

5. Suppose we have a deeply pipelined processor, for which we implement a branch-target buffer for the conditional branches only. Assume that the misprediction penalty is always four cycles and the buffer miss penalty is always three cycles. Assume a 90% hit rate, 90% accuracy, and 15% branch frequency. How much faster is the processor with the branch-target buffer versus a processor that has a fixed two-cycle branch penalty? Assume a base clock cycle per instruction (CPI) without branch stalls of one.

Solution:-

For this problem we are given the base CPI without branch stalls. From this we can compute the number of stalls given by no BTB and with the BTB: $\text{CPI}_{\text{noBTB}}$ and CPI_{BTB} and the resulting speedup given by the BTB:

$$\text{Speedup} = \frac{\text{CPI}_{\text{noBTB}}}{\text{CPI}_{\text{BTB}}} = \frac{\text{CPI}_{\text{base}} + \text{Stalls}_{\text{base}}}{\text{CPI}_{\text{base}} + \text{Stalls}_{\text{BTB}}}$$

To compute $\text{Stalls}_{\text{BTB}}$, consider the following table:

BTB result	**BTB prediction**	**Frequency (per instruction)**	**Penalty (cycles)**
Miss		$15\% \times 10\% = 1.5\%$	3
Hit	Correct	$15\% \times 90\% \times 90\% = 12.1\%$	0
Hit	Incorrect	$15\% \times 90\% \times 10\% = 1.3\%$	4

Therefore:

$$\begin{aligned}
\text{Stalls}_{\text{noBTB}} &= 15\% \times 2 = 0.30 \\
\text{Stalls}_{\text{BTB}} &= (1.5\% \times 3) + (12.1\% \times 0) + (1.3\% \times 4) = 0.097 \\
\text{Speed up} &= \frac{1.0 + 0.30}{1.0 + 0.097} = 1.2
\end{aligned}$$

6. Consider a branch-target buffer that has penalties of zero, two, and two clock cycles for correct conditional branch prediction, incorrect prediction, and a buffer miss, respectively. Consider a branch-target buffer design that distinguishes conditional and unconditional branches, storing the target address for a conditional branch and the target instruction for an unconditional branch. What is the penalty in clock cycles when an unconditional branch is found in the buffer?

Solution:-
Storing the target instruction of an unconditional branch effectively removes one instruction. If there is a BTB hit in instruction fetch and the target instruction is available, then that instruction is fed into decode in place of the branch instruction. The penalty is -1 cycle. In other words, it is a performance gain of 1 cycle.

7. Use the following code fragment:

```
Loop:   LD      R1,0(R2)       ; load R1 from address 0+R2
        DADDI   R1,R1,#1       ; R1 = R1 + 1
        SD      R1,0(R2)       ; store R1 at address 0+R2
        DADDI   R2,R2,#4       ; R2 = R2 + 4
        DSUB    R4,R3,R2       ; R4 = R3 - R2
        BNEZ    R4,Loop        ; branch to Loop if R4 != 0
```

Data hazards are caused by data dependences in the code. Whether a dependency causes a hazard depends on the machine implementation (i.e., number of pipeline stages). List all of the data dependences in the code above. Record the register, source instruction, and destination instruction; for example, there is a data dependency for register R1 from the LD to the DADDI.

Solution:-

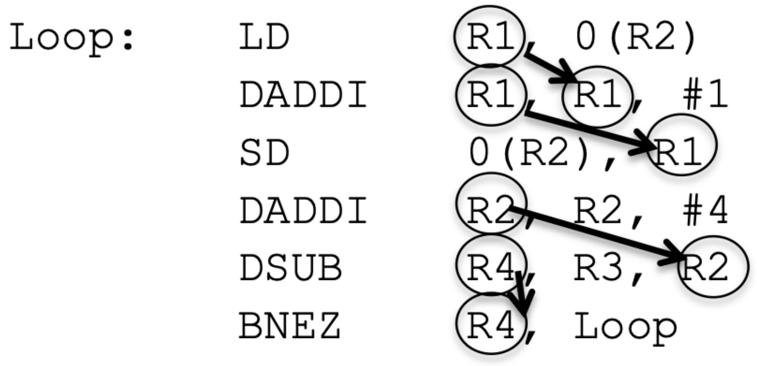

Figure 1: Data Dependencies

8. A 5-stage pipelined processor (IF, ID, EX, MEM, WB) has full data forwarding. How many stall cycles, if any, are required to execute the following two-instruction sequence?

```
I1: LW   R1, 0(R2)     ; Load R1 from memory
I2: ADD  R3, R1, R4    ; Use R1 in an addition
```

Solution:
This sequence contains a **load-use data hazard**.

(a) The LW instruction fetches the value for $R1$ from memory during its **MEM stage**. The data is available at the end of this stage.

(b) The ADD instruction needs the value of $R1$ for its **EX stage**.

(c) Even with forwarding, the data from I1's MEM stage is not ready in time for I2's EX stage, which immediately follows. The pipeline must insert a "bubble" to wait for the data.

This results in a single **1-cycle stall**.

9. A processor resolves branches in the EX stage, incurring a **3-cycle penalty** for a misprediction. A loop runs exactly 5 times. What is the total penalty (in cycles) for the loop's conditional branch instruction if the processor uses a static **"Predict Always Taken"** scheme?

Solution:
The branch instruction is executed 5 times.

(a) **Executions 1-4:** The loop continues, so the branch is **Taken**. The "Predict Always Taken" scheme correctly predicts these. (Penalty = 0 cycles)

(b) **Execution 5:** The loop terminates, so the branch is **Not Taken**. The scheme incorrectly predicts "Taken". This is a **misprediction**.

There is only **one misprediction** (on the final exit).

Total Penalty = 1 misprediction$\times$3 cycles/misprediction=3 cycles.