HAD2026 Tutorial 6

Concept Questions

Q1: Pipelining takes advantage of

A: Locality

B: Parallelism

C: Both A and B

D: Legal tax loopholes

E: Not sure

Q2: How does the five-stage pipeline affect instruction throughput?

A: throughput increases

B: throughput decreases

C: throughput stays the same

D: depends on the program

E: uncertain, blame Heisenberg

Q3: How does the five-stage pipeline affect instruction latency?

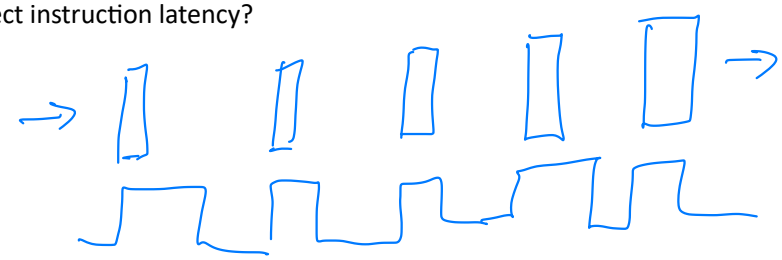
A: latency increases

B: latency decreases

C: latency stays the same

D: depends on the program

E: latency turns negative



Q4: Assume we are building a pipelined MIPS64 processor. Considering only integer registers (32 general purpose register), how many bits are used to represent each of RS, RT, RD (the source and destination register specifiers)?

A: 8

B: 64

C: 5

D: 32

E: Not sure

$2^5 = 32$ $\log_2(32) = 5$

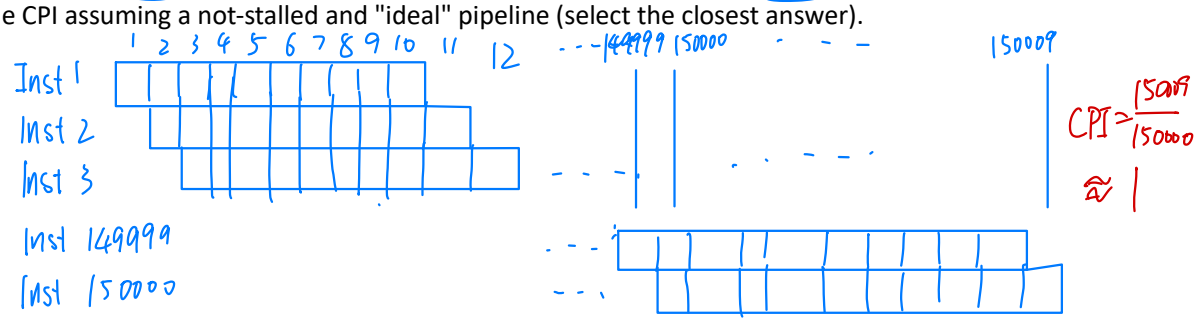
Q5: A 10-stage pipeline executes 150000 instructions. Each stage in the pipeline takes 1 cycle and all instructions are independent. What is the CPI assuming a not-stalled and "ideal" pipeline (select the closest answer).

A. 1.5

B. 15000

C. 1

D. 10



Pipelining Problem

Consider the following procedure, adapted from *Hacker's Delight*, 2nd ed. page 81:

```
popcount:
  LDI     r2, DATA        ; r2 gets pointer to data
```

```
  ADDI    r5, r2, 0x10    ; r5 is our loop-ending value
loop:
  LDR     r3, r2, 0x00    ; r3 holds mark from memory
  AND     r4, r1, r3      ; r4 = x & mem
  LSR     r1, r1, 0x01    ; r1 = r1 >> 1
  AND     r3, r1, r3      ; r3 = ((x>>1)&mem)
  ADD     r1, r3, r4      ; r1 = (x&mem) + ((x>>1)&mem)
  ADDI    r2, r2, 0x04    ; increment the pointer
  BNE     r2, r5, loop    ; are we done?
  JR                     ; return to caller

.DATA                    ;offset
.DWORD    0x55555555      ;0x00
          0x33333333      ;0x04
          0x0F0F0F0F      ;0x08
          0x00FF00FF      ;0x0C
          0x0000FFFF      ;0x10
```

(a)Identify data dependences in the code above and use those dependences to identify the (structural, data, and control) hazards the code above might encounter.

```
A  LDI    r2,     DATA
B  ADDI   r5,  r2,  0x10
C  LDR    r3,  r2,  0x00
D  AND    r4,  r1,  r3
E  LSR    r1,  r1,  0x01
F  AND    r3,  r1,  r3
G  ADD    r1,  r3,  r4
H  ADDI   r2,  r2,  0x04
I  BNE    r2,  r5,  loop
J  JR
```

RAW Hazard

```
A  LD    r2,  $data          # ← procedural call
B  DADDI r5,  r2,  $0x10     # ← depend on r2 in inst A
C  LD    r3,  0(r2)          # ← r2 in inst A, r2 in inst H
D  AND   r4,  r1,  r3        # ← depend on r3, r1 in inst C
E  SHRI  r1,  r1,  #1        # ← r1 in inst C, after 1st loop
F  AND   r3,  r1,  r3        # ← r1 in inst E, r3 in inst C
G  DADD  r1,  r3,  r4        # ← r3 in inst F, r4 in inst ID
H  DADDI r2,  r2,  #4        # ← r2 in inst A, itself
I  BNE   r2,  r5,  $loop     # ← r2 in inst H
H  J     r31                 # ← procedural call
```

(b)Show the timing for this instruction sequence for the RISC pipeline with no forwarding path except bypass register file (i.e. your written register can be read at the same cycle). Use a pipeline timing chart like Table 1. Assume that the branch is handled by stalling. Assume that the last branch is a taken branch. The branch target is resolved at the end of the decode stage

(c) Show again the timing of this instruction sequence for the RISC pipeline, but now with normal forwarding and bypassing hardware. Draw an (->) to indicate forwarding. Additionally, the branch is handled by being predicted non-taken. Use Table 2.

(d) Show again the RISC pipeline timing of the slightly-modified instruction sequence in Table 3, again with forwarding and bypassing hardware. The branch is now handled with a delay slot, which we handle by putting the DADD instruction in the slot. Bonus: Can you come up with a more-performant instruction order that has an equivalent result?

Instruction	Clock Number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
No Forwarding																
LDI r2, DATA																
ADDI r5, r2, 0x10																
LDR r3, r2, 0x00																
AND r4, r1, r3																
LSR r1, r1, 0x01																
AND r3, r1, r3																
ADD r1, r3, r4																
ADDI r2, r2, 0x04																
BNE r2, r5, loop																
JR																
LDR r3, r2, 0x00																
AND r4, r1, r3																
LSR r1, r1, 0x01																
AND r3, r1, r3																
and so on...																
	W→D															
Bypass regfile and reading in X																
LDI r2, DATA	F	D	X	M	W											
ADDI r5, r2, 0x10		F	D	D$_s$	D$_s$	X	M	W								
LDR r3, r2, 0x00			F	F$_s$	F$_s$	D	X	M	W							
AND r4, r1, r3						F	D$_s$	D$_s$	D$_s$	X	M	W				
LSR r1, r1, 0x01							F	F$_s$	F$_s$	D	X	M	W			
AND r3, r1, r3										F	D	D$_s$	D$_s$	X	M	W
ADD r1, r3, r4	X	M	W								F	F$_s$	F$_s$	D	D$_s$	D$_s$
ADDI r2, r2, 0x04	D	X	M	W										F	F$_s$	F$_s$
BNE r2, r5, loop	F	D	D$_s$	D$_s$	X	M	W									
JR	Not	fetched														
LDR r3, r2, 0x00					F	D	X	M	W							
AND r4, r1, r3						F	D	D	D	X	M	W				
LSR r1, r1, 0x01							F	F	F	D	X	M	W			
AND r3, r1, r3										F	D	D	D	X	M	W
and so on...																

Table 1: RISC pipeline. On each clock cycle, another instruction is fetched and begins its five- cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: F = instruction fetch, D =instruction decode, X = execution, M = memory access, and W = write-back.

Instruction	Clock Number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LDI r2, DATA																
ADDI r5, r2, 0x10																
LDR r3, r2, 0x00																
AND r4, r1, r3																
LSR r1, r1, 0x01																
AND r3, r1, r3																
ADD r1, r3, r4																
ADDI r2, r2, 0x04																
BNE r2, r5, loop																
JR																
LDR r3, r2, 0x00																
AND r4, r1, r3																
LSR r1, r1, 0x01																
AND r3, r1, r3																
and so on...																

Table 2: RISC pipeline. On each clock cycle, another instruction is fetched and begins its five- cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: F = instruction fetch, D =instruction decode, X = execution, M = memory access, and W = write-back.

Instruction	Clock Number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
No Forwarding																
LDI r2, DATA																
ADDI r5, r2, 0x10																
LDR r3, r2, 0x00																
AND r4, r1, r3																
LSR r1, r1, 0x01																
AND r3, r1, r3																
ADD r1, r3, r4																
ADDI r2, r2, 0x04																
BNE r2, r5, loop																
JR not fetched																
LDR r3, r2, 0x00																
AND r4, r1, r3																
LSR r1, r1, 0x01																
AND r3, r1, r3																
and so on...																

Table 3: RISC pipeline. On each clock cycle, another instruction is fetched and begins its five- cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: F = instruction fetch, D =instruction decode, X = execution, M = memory access, and W = write-back.