Essentials of Computer Systems - Exercises #4

1 Performance Equations

Execution time $= T = IC \times t_{\text{CLK}} \times CPI = \frac{\#insn}{program} \cdot \frac{\#seconds}{cycle} \cdot \frac{\#cycpes}{insn}$
Performance $= \frac{1}{T}$

Speedup$= \frac{Told}{Tnew}$

Amdahl's law: the total speedup depends on

- fraction of X being used in original system $\Rightarrow f$
- enhancement/speedup of X $\Rightarrow S$

$S_{\text{total}} = \frac{1}{\frac{f}{S}+(1-f)}$

Exercise 1.1 *Average CPI*

When running a given benchmark we found the frequency and CPIs of instructions given in table 1. Afterwards, two separate modifications were made and benchmarking repeated:

a) hardware modifications resulting in changed CPIs of the instructions (see table 2)

b) modifications to the compiler resulting in changed frequencies if the instructions (see table 3)

Compare all three options and identify the best one! → average CPI of the different versions

Table 1		
insn	f_i	CPI_i
ALU	*50%*	*1*
load	*20%*	*5*
store	*10%*	*3*
other	*20%*	*2*

Table 2		
insn	f_i	CPI_i
ALU	*50%*	*1*
load	*20%*	*4*
store	*10%*	*4*
other	*20%*	*2*

Table 3		
insn	f_i	CPI_i
ALU	*55%*	*1*
load	*10%*	*5*
store	*10%*	*3*
other	*25%*	*2*

$\frac{1\cdot 50+5\cdot 20+3\cdot 10+2\cdot 20}{100} = 2.2$ $\frac{1\cdot 50+4\cdot 20+4\cdot 10+2\cdot 20}{100} = 2.10$ $\frac{1\cdot 55+5\cdot 10+3\cdot 10+2\cdot 25}{100} = 1.85$

$Speedup = \frac{IC \cdot t_{clock} \cdot CPI_{old}}{IC \cdot t_{clock} \cdot CPI_{new}} = \frac{CPI_{old}}{CPI_{new}}$ same benchmark → same IC, same hardware → same t_{clock}

$S = \frac{2.2}{2.1} = 1.05$ $S = \frac{2.2}{1.85} = 1.12$ → higher speed-up

Exercise 1.2 *Amdahl's law*

The old CPU takes 5 seconds to execute a given program. A new CPU with four times faster adder[1] *is available. The adder is used in 25% of the instructions in the given program. Compute the new execution time for the given program when using the new CPU.*

$S = \frac{1}{\frac{0.25}{4} + (1-0.25)} = 1.23$ → speed-up of 23%

old time → 5 s, speed-up → 1.23 $= 4.0625$ s ← new time

[1] adder is a part of the ALU

2 Execution on the Pipelined CPU

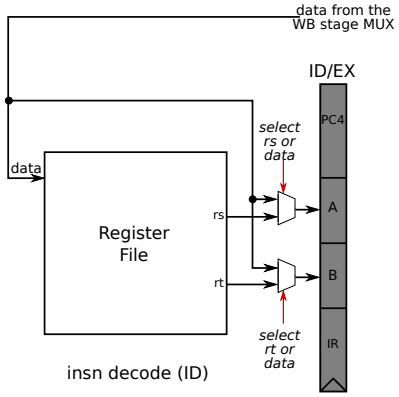
Exercise 2.1 *For the following code snippet show:*

a) analysis of data dependencies. Identify the data hazards.

b) show the timing diagram of execution on pipelined CPU without bypassing logic (see lecture slides)

c) show the timing diagram of execution on pipelined CPU with bypassing logic from EX/MEM and MEM/WB interstage registers

d) show the timing diagram of execution on pipelined CPU with bypassing logic from EX/MEM and MEM/WB interstage registers and with RegFile bypassing. RegFile bypassing allows the "overlap" of ID and WB stage by adding MUXes on the inputs to the interstage registers ID/EX.A and ID/EX.B, allowing to select either the RegFile outputs rs and rt or the data value from WB stage - please see schematic below.



recap:

insn syntax	*operation*
and Rd, Rs, Rt	*Rd ← Rs AND Rt*
or Rd, Rs, Rt	*Rd ← Rs OR Rt*
lw Rt, offset(Rs)	*Rt ← MEM[Rs+offset]* (W, R)

code snippet:

lw R3, 0(R1)
and R4, R3, R2
or R5, R3, R2
and R3, R4, R5

a) Analysis

code snippet:

lw R3, 0(R1)
and R4, R3, R2
or R5, R3, R2
and R3, R4, R5

insn 2 reads R3 → insn 3 reads R3 } RAR — not a data dependency
insn 2 reads R2 → insn 3 reads R2 }

insn 2 reads R3 → insn 4 writes R3 } WAR
insn 3 reads R3 → insn 4 writes R3 }
insn 1 writes R3 → insn 4 writes R3 } WAW
} not a problem, as long as the insn are completed in the original order

insn 1 writes R3 → insn 2 reads R3
insn 1 writes R3 → insn 3 reads R3
insn 2 writes R4 → insn 4 reads R4
insn 3 writes R5 → insn 4 reads R5
} RAW — Hazards!

b) Stalling the CPU waits (stall) until the needed value is available in RegFile (after WB)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw R3, 0(R1)	IF	ID	EX	MEM	WB									
and R4, R3, R2		IF	o	o	o	ID	EX	MEM	WB					
or R5, R3, R2						IF	ID	EX	MEM	WB				
and R3, R4, R5							IF	o	o	o	ID	EX	MEM	WB

r3 value is retrievable in RegFile from cycle 6

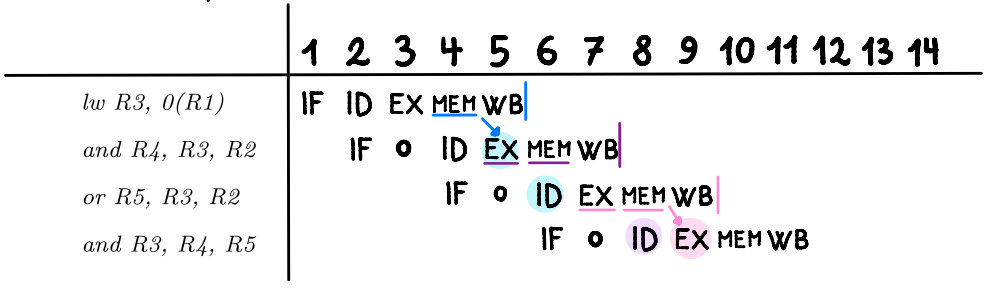
r4 value is retrievable in RegFile from cycle 10

r5 value is retrievable in RegFile from cycle 11

c) Interstage registers bypassing

request the values from the interstage registers as soon as they are available (bypass):

- load (lw): value available in MEM/WB (after MEM) → only temporarily available !
- arithmetic/logic operations: value available in EX/MEM, then in MEM/WB (after EX and MEM)



r3 value is directly available in cycle 5, and retrievable in RegFile from cycle 6

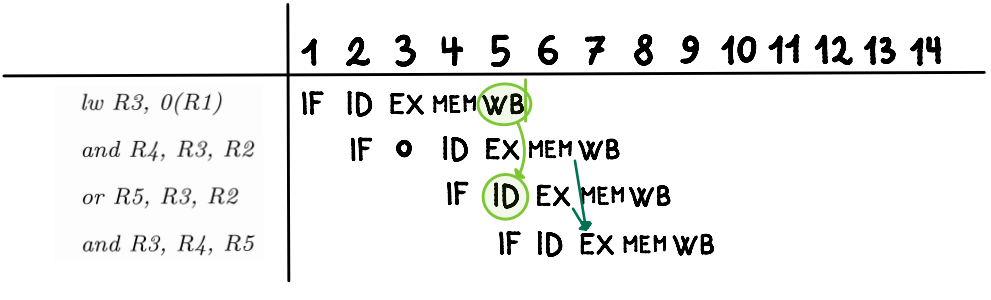
r4 value is directly available in cycles 6 and 7 and retrievable in RegFile from cycle 8

r5 value is directly available in cycles 8 and 9 and retrievable in RegFile from cycle 10

DECODE accepts addresses, EXECUTE accepts data. Only DECODE (ID) can access the RegFile, only EXECUTE can take data directly from the interstage registers

d) RegFile bypassing

data that are being saved in the RegFile are available to DECODE in the same cycle.



r3 is directly available in cycle 5 and retreivable in RegFile from cycle 6

both r4 and r5 use bypassing