Chapter 4

" The Processor "

Exercise Solve.

→4.1:
Ans: The control signal ALUMax can control Mux. The output of the register file is selected at the ALU input and the immediate from the instruction word is selected by 1(Imm) as the second input to ALU ~~as~~ AN x Rd, Rs, Rt.

RegWrite	MemRead	ALUMax	MemWrite	ALUop	RegMax	Branch
0	0	1(Imm)	1	ADD	X	0

Reg Muxe is the control signal that controls the mox of the Data input to the register file.

→4.13:
Ans: The branch add produces the data memory as an input. This output is not used where the branch add, second the readpure register produces no output.

→ 4.12:
Ans: The Branch add ~~produces the data~~ unit doesnt perform a useful function for this instruction. It writes the part of the register.

→ 4.2.1:

Ans: The given instruction memory, both registers read ports, the ALU to, add Rd, and Rs, data memory, and write port in Register.

→ 4.2.2:

Ans: The instruction can be implemented using existing blocks.

→ 4.2.3:

Ans: The instruction can be implemented without adding new control signal. It only required control logic.

→ 4.3.1:

Ans: The latency of this path is 400ps + 200ps + 30ps + 120 ps + 350ps + 30ps = 1130 ps.

The original clock cycle time without improvement is 1130.

New critical path = 100 + 100 + 30 + 420 + 200 + 350 + 100 = 1600. The new clock cycle time with the improvement is 1600 ps.

→4.3.2:

Ans: Speed up = $\frac{\text{old clock cycle time}}{\text{new clock cycle time}} = \frac{1130}{1160} = 0.83$

which means it's actually have a slowdown.

→4.3.3:

Ans: Without improvement the total cost,

1000 + 30 + 10 + 120 + 200 + 2000 + 500 = 3860

with improvement the total cost,

1000 + 30 + 10 + 700 + 200 + 2000 + 500 = 440

Instruction executed with improvement = 95%
(5% fewer instruction).

$\frac{\text{original cost}}{\text{Performance ratio}} = \frac{\text{total Cost}}{(\text{Instr. executed} \times \text{clock cycle time})}$

$= \frac{3860}{(1 \times 1160)} = 3.27$

$\frac{\text{Improved cost}}{\text{Performance ratio}} = \frac{4440}{0.95 \times 1600} = 2.929$

Relative cost : $\frac{4440}{3890} = 1.15$.

Cost/Performance = $1.15/0.83 = 1.39$.

→4.5.1:

Ans: The data memory is used by lw and sw instructions, so 25% + 10% = 35%

→4.5.2:

Ans: The sign extend circuit is actually computing a ~~result~~ in every cycle, but its output is ignored for ADD and not operations.

Now, 20% + 25% + 25% + 10% = 80%.

→4.7.1:

Ans:

Sign extend	Jump shift ~~left~~ 2
00000000000000000010100	00011000100000000000001010000

→4.7.2:

Ans:

ALUop [1-0]	Instruction [50-0]
00	010100

→4.7.4:

Ans: for each mux, the rule of its data output during the ~~execut~~ execution of this instruction and these register values.

Wr Reg Mux	ALU Max	Mem/ALU Muge	Branch mux	Jump Mux
2 or 0	20	x	PC+4	PC+4

→4.7.5:
Ans: for the ALU and the two add units, so their data input values,

ALU	add (PC+4)	Add (Branch)
-3 and 20	PC and 4	PC+4, 20×4

→4.9.1:
Ans: sequence of instruction,

or r1, r2, r3
or r2, r1, r4
or r1, r1, r2

Here, RAW on R1 from i1 to i2 and i3 RAW on R2 from i2 to i3, WAR on R2 from i1 to i2. WAR on R1 from i2 to i3. WAR on R1 from i1 to i3.

→ 4.9.4:

Ans: The sequence of instruction takes 7 cycles for execution, (7+4) × 180 ps = 11 × 180 = 1980 ps

The total execution time with forwarding is 7×240 = 1680

The speed up because forwarding will be 1.18 as per the total exe.

→ 4.10.1:

Ans:

sw	r16, 12 (r6)	IF - ID - Ex - MEM - WB
lw	r16, 8 (r6)	IF - ID - Ex - MEM - WB
beq	r5, r4, lb1	IF - ID - Exe - MEM - WB
add	r5, r1, r4	.... IF - ID -
slt	r5, r15, r4	IF - I

Total cycle 11. we cannot NOPs to the code to eliminate this hazard - NOPs need to be fetch just like any other instruction. So, structural hazard and data hazard must be addressed.

→ 4.10.2:

Ans: This change only saves one cycle in an entire execution without data hazard

Instruction executed = 5.

cycles with 5 stages, 4+5 = 9

cycles with 4 stages, 3+5 = 8

$$\therefore \text{speed up} = \frac{\text{cycles with 5 stages}}{\text{Cycles with 4 stages}} = \frac{9}{8} = 1.13$$

→ 4.10.3:

Ans: The speed up achieved on the code if branch outcomes are determined in the ID stage, relative to the execution. where the branch outcomes are determined in the ID stage, each branch only causes one stall cycle

Instr. executed = 5, branches executed = 4 cycles with branches in Exe = 4+5+(1×2) = 9+2 = 11.

cycle with branch in ID, 4+5+(1×1) = 10

$$\text{Speed up} = \frac{11}{10} = 1.10$$

→ 4.10.4:

Ans: The no of cycles for the 5 stage and the 4 stage pipeline is already computed. The clock cycle time is equal to the latency of the longest latency stage,

IF = 200 ps, ID = 120 ps, EXE = 150 ps, MEM = 190 ps

WB = 100 ps.

cycle with 5 stages = 200 ps
cycle with 4 stages = 210 ps

$$\text{speed up} = \frac{9 \times 200}{8 \times 210} = 1.0781$$

→ 4.10.5:

Ans: Assuming that the latency ID stage increased by 50% and the latency of the EXE stage decreased by 10ps.

New ID latency = 180ps; New Exe latency = 140ps
New cycle time = 200ps; Old cycle time = 200 ps

$$\text{Speed up} = \frac{11 \times 200}{10 \times 200} = 1.1$$

→ 4.10.6:

Ans: For each branch, the change does not affect execution time because it adds one additional stall.

cycle with branch in execution = $4+5+(10\times2)$
$= 11$

Execution time = 11×200 ps
$= 2200$ps.

cycle with branch in MEM = $4+5+(1\times 3)$

$= 12$

Execution time (branch in MEM) = 12×200 ps

$= 2400$ ps.

$\therefore$ speed up = $\frac{\text{exe time (In Ex)}}{\text{exe time (in MEM)}} = \frac{2200 \times 10^{12}}{2400 \times 10^{12}} = 0.9167$

$= 0.92$.

→ 4.11.1:
Ans: Pipeline execution diagram for the third iteration of the loop:

lw r1, 0(r1)	WB							
lw r1, 0(r1)	EX	MEM	WB					
beq r1, r0, loop	IF	....	EXE	MEM	WB			
lw r1, 0(r1)	IF	......	ID	EXE	MEM	WB		
and r1, r1, r2			IF	ID	....	EXE	WB	
lw r1, 0(r1)				IF	....	ID	EXE	
lw r1, 0(r1)						IF	ID---	
beq r1, r0, loop							IF ---	

→ 4.11.2:
Ans: We get from 4.11.1, the stalled stages will not doing useful work particular cycle. So, the total cycle

per iteration is 8. cycles in which all stages do useful work in none. So, the percentage of cycles in which all stages do useful work in 0%.

→ 4.12.18

Ans: Dependences to the 1st next instruction result in 2 stall cycle, and ~~&~~ the stall is also 2 cycles if the dependence is to both 1st and 2nd next iteration

$CPI = 1 + 0.35 \times 2 + 0.15 \times 1 = 1.85$.

stall cycles, $\frac{0.85}{1.85} = 0.459$

$\therefore 0.459 \times 100 = 45.9 = 46\%$

→ 4.12.2:

Ans. Considering the full forwarding, the MEM stage of one instruction to the 1st next instruction cause RAW dependences

$CPI = 1 + 20\% = 1 + 0.20$
$= 1.20$

stall cycle $= \frac{0.20}{1.20} = 1.67 \times 100$
$= 16.7$ or 17%

→ 4.12.3:

Ans: Stall cycles of EXE/MEM, 0.2+0.05+0.1+0.1
= 0.45

~~Exe~~ to 2nd has no stall.

MEM/WB is better than EXE/MEM

→ 4.12.4:

Ans: Clock cycle time without forwarding,

1.85 × 150 = 277.5 ps

Clock cycle time with forwarding, 1.2 × 150 = 180 ps

$$\text{Speed up} = \frac{277.5}{180} = 1.54167$$

Ay: