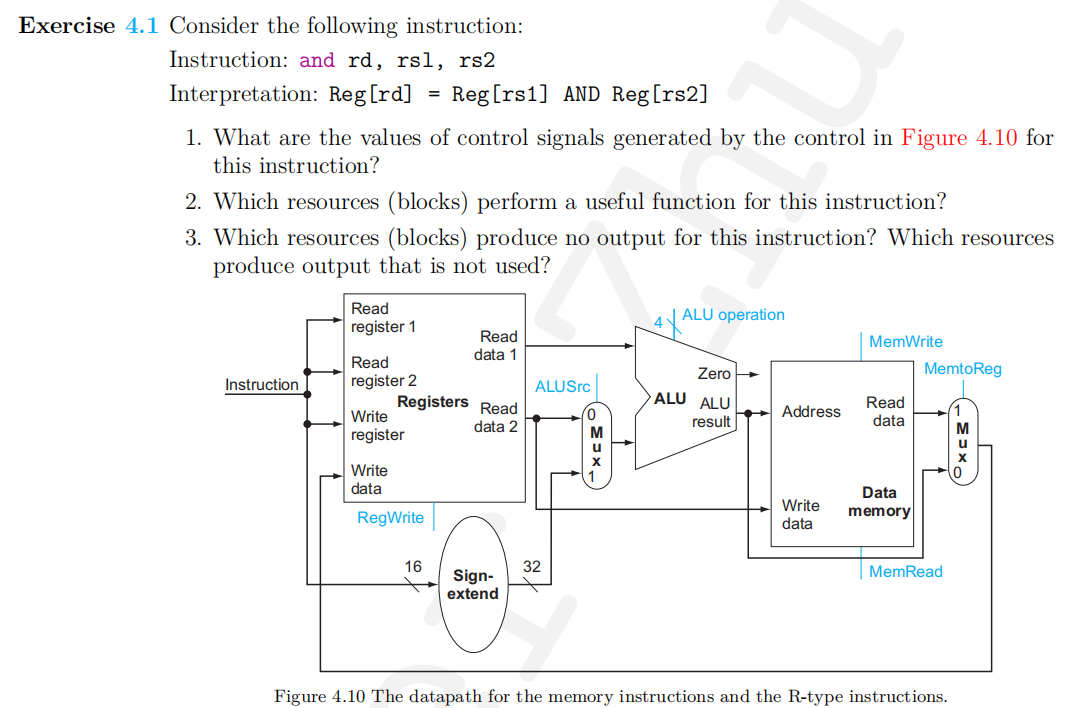
# Homework 03

|  |  |  |
| --- | --- | --- |
| 黄家睿 | 202283890036 | IOT |

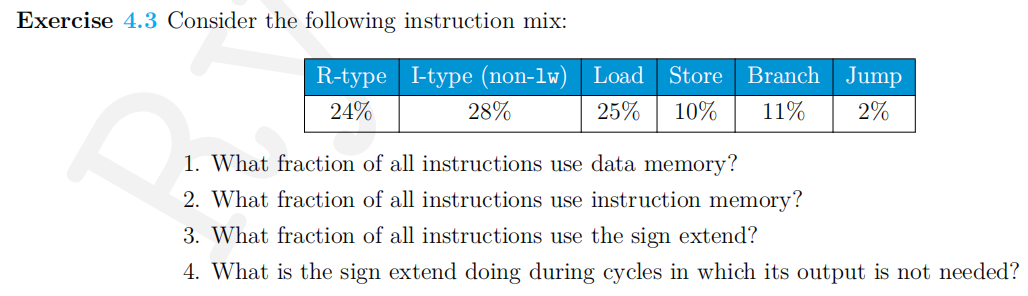


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RegWrite | ALUSrc | ALU operation | MemWrite | MemRead | MemtoReg |
| True | 0000 | And | False | False | 0 |

**Solution:**  1:

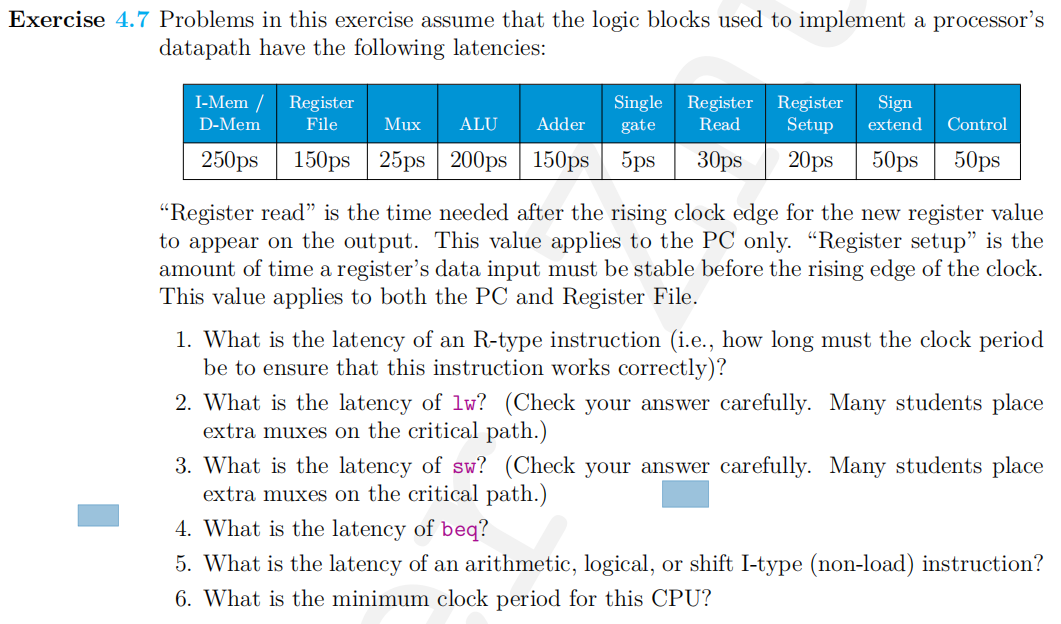
2: Register, the ALUSrc Mux, ALU, MemtoReg

3: All blocks produce some output, but data memory is not used.



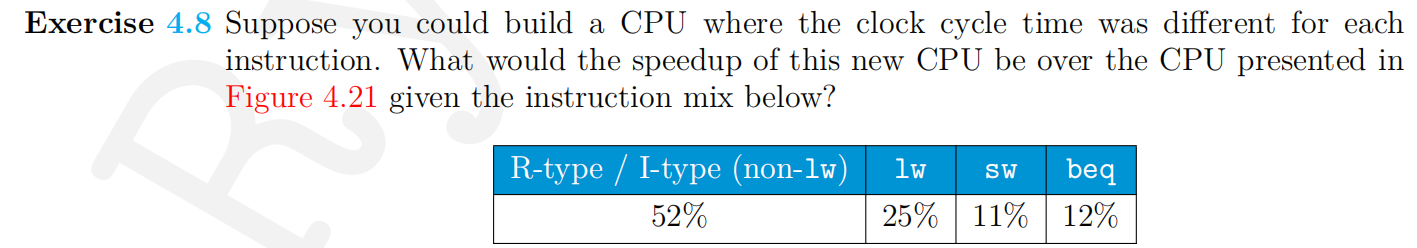
**Solution**  1. Only Load and Store use data memory. Fraction=25%+10%=35%

1. All kinds of instruction must be fetched and then decoded from instruction memory before its execution, fraction=100%
2. All kinds of instruction except r-type one does not need sign extend, fraction=100%-24%=76%
3. If output does not need sign extend, it will be ignored.



**Solution:**  1. R-type: 30ps+250ps+150ps+25ps+200ps+25ps+20ps=700ps

1. lw: 30ps+250ps+150ps+25ps+200ps+250ps+25ps+20ps=950ps
2. sw: 30ps+250ps+150ps+25ps+200ps+250ps=905ps
3. beq: 30ps+250ps+150ps+25ps+200ps+5ps+25ps+20ps=705ps
4. I-type: 30ps+250ps+150ps+25ps+200ps+25ps+20ps=700ps
5. The minimum clock cycle required is the largest time slot that a instruction takes, the amount of time of accessing memory via lw instruction ,950ps.

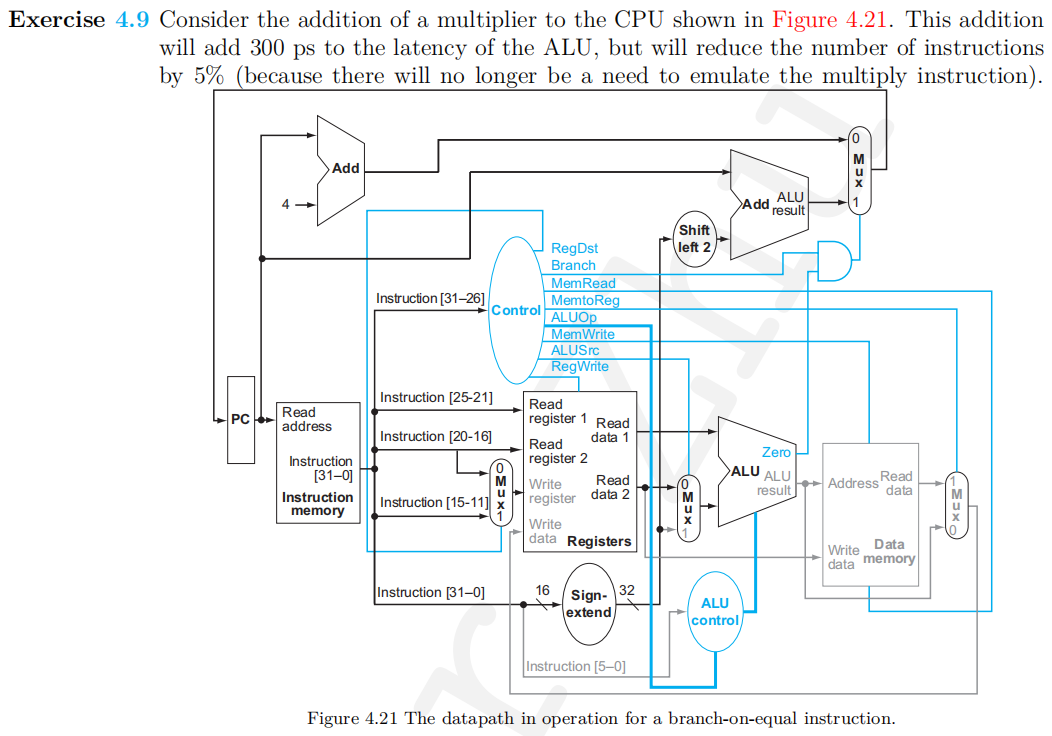


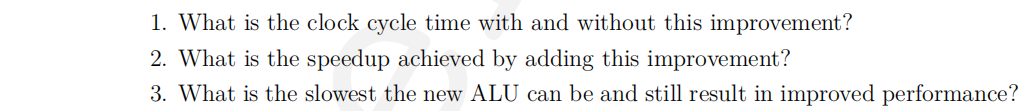
**Solution:** The average time per instruction for the instruction mix is：

00ps+25950ps+11%705ps+12%705=785.65ps

While a CPUC executing different instruction for the same clock cycle time is 950ps.

The speedup factor: 

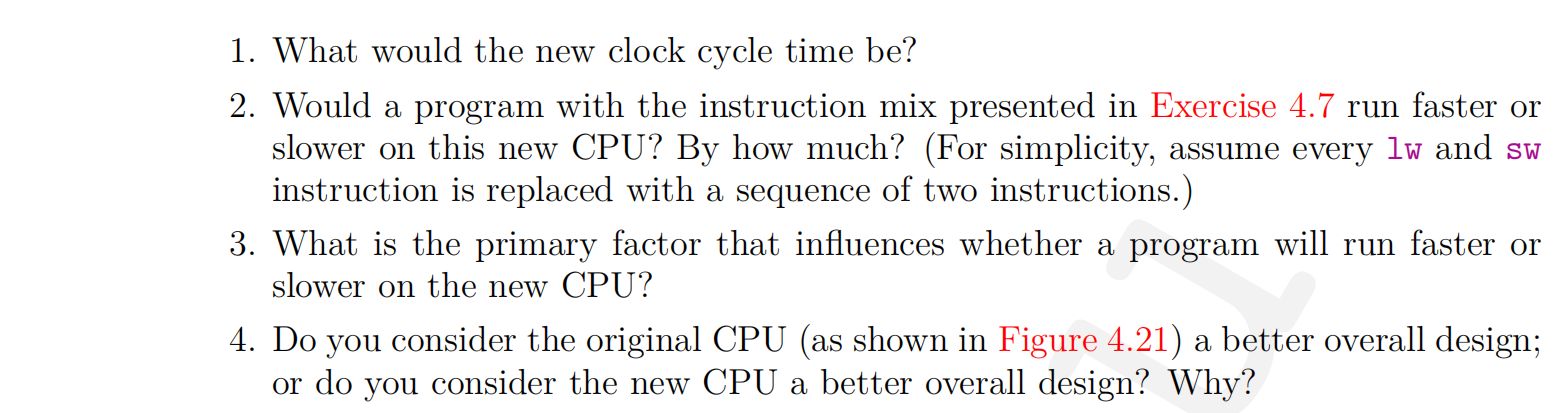
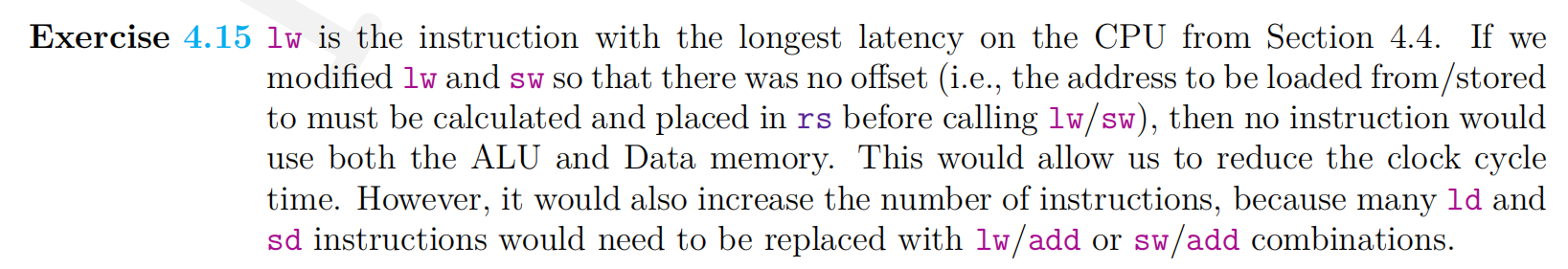




**Solution:** 1. Without the improvement the cycle time is 950ps and after the improvement is 1250, which the cycle time would 300ps longer.

1. The speedup factor is :
2. The worst case is the new one takes as much cycle time as the previous one, i.e. 950ps and given that the improvement would reduce the number of instructions by 5% so the maximum cycle time is

In other words, the time the new ALU takes should only increase at most 50ps, which means a slowest improved ALU should take 250ps to finish computation

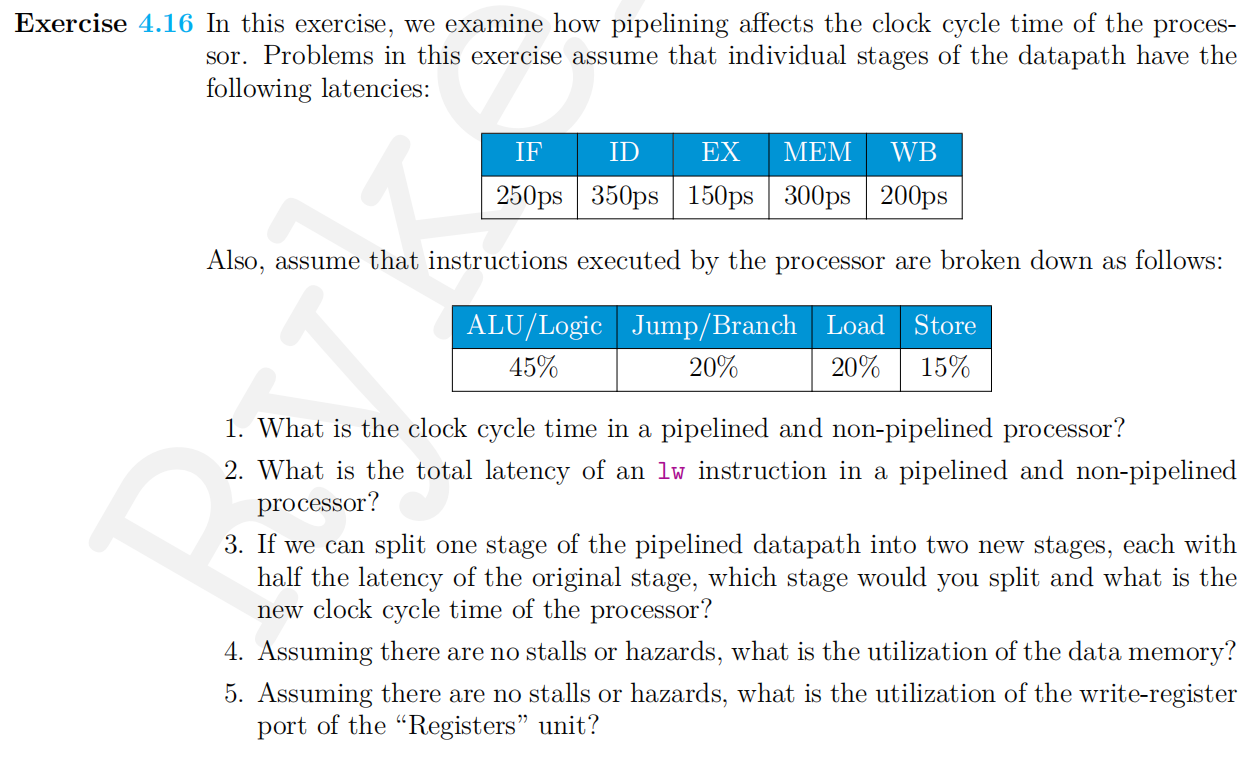
 **Solution: 1.** The clock time is :30ps+250ps+150ps+max{25ps+200ps,25ps+250ps+ 25ps+20}=750ps

1. The original version of the CPU takes 950ps to run cycle while th e revised one take 750ps. Beside since Id and sd instructions are equivalent to two lws and sws, there will be an extra 25%+1 1%=36% load/store instruction, resulting in 1.36n750%=n1012. 5ps and the speedup factor

.

Hence, the program run slower on this new CPU

1. The number and effect of load/store instructions are the primary factors
2. The new one is better. Despite increasing instruction count, this design streamlines memory access, enhances scalability, and facilitates future optimizations, masking it a superior choice for systems prioritizing memory latency reduction and overall performance.

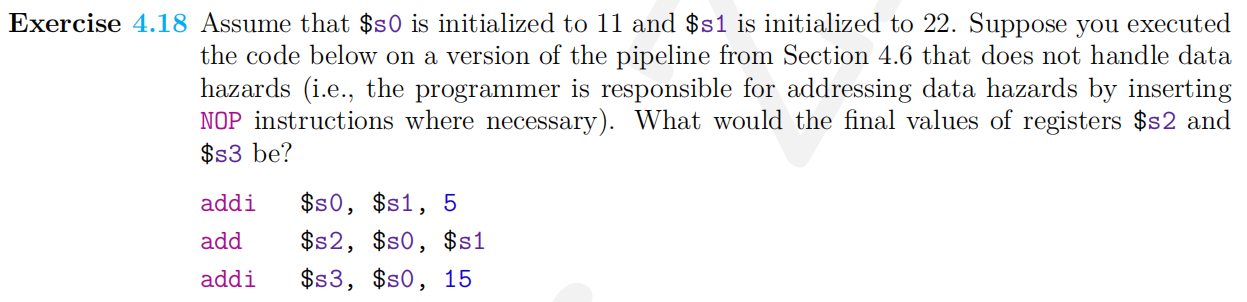
 solution: 1. For non-pipelined process 250ps+350ps+150ps+300ps+200ps=1250ps, for pipelined t=350ps

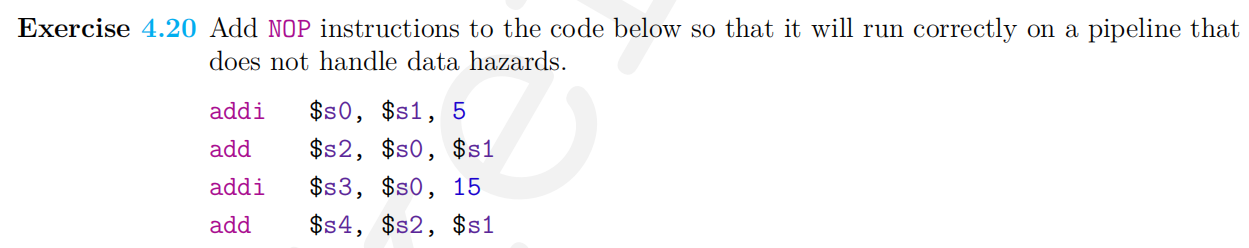
1. The time cost for a non-pipelined is 1250ps, for the pipelines one

It is equal to the number of stages times the cycle time:

3350=1750ps

1. Split the ID stage since it is the stage with the longest running ti me and then the cycle, time becomes the format second longest running : 300ps
2. 35%.
3. 65%

 **solution:** $s2=22+5+22=49 and $s3=22+5+15=42



**solution:** *addi $s0,$s1,5*

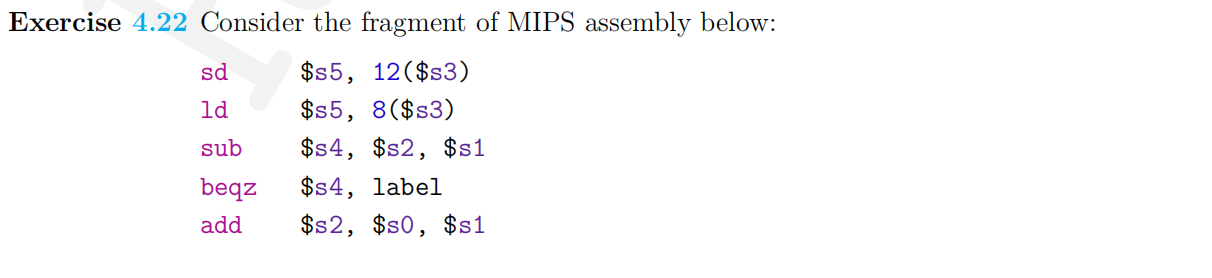
*nop*

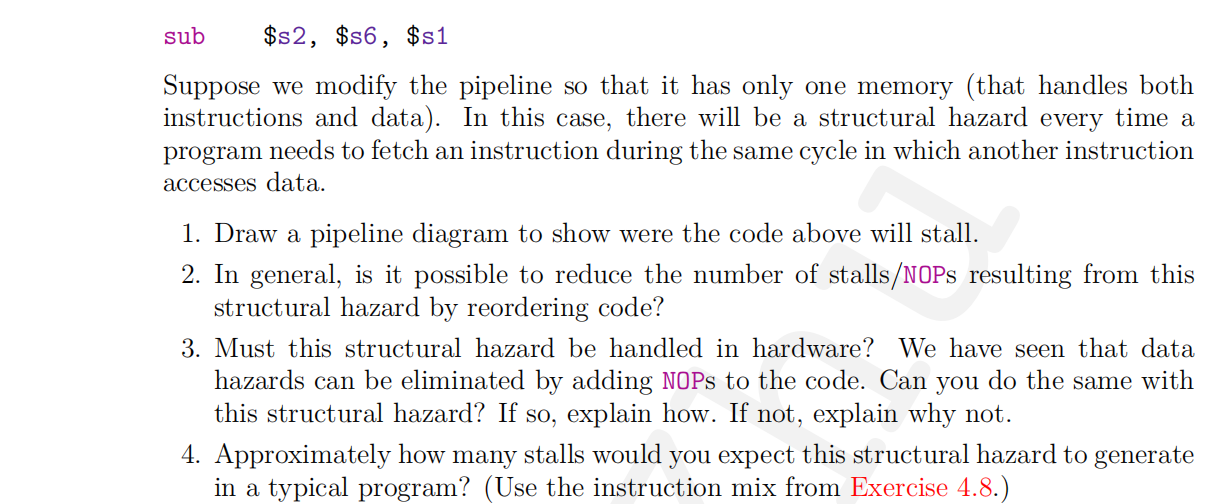
*nop*

*add $s2,$s0,$s1*

*nop*

*add $s4,$S2,$s1*





**Solution:** 1. sd $s5,12($s3) IF ID EX ME WB

Ld $s5,8($s3) IF ID EX ME WB

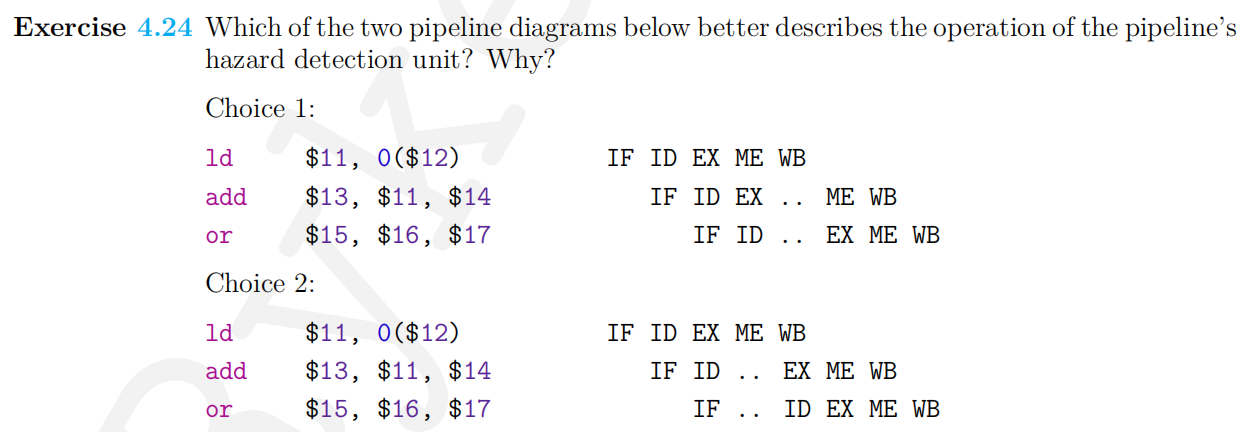
Sub $s4,$s2,$s1 IF ID EX ME WB

Beqz $s4,label \*\* \*\* IF ID EX ME WB

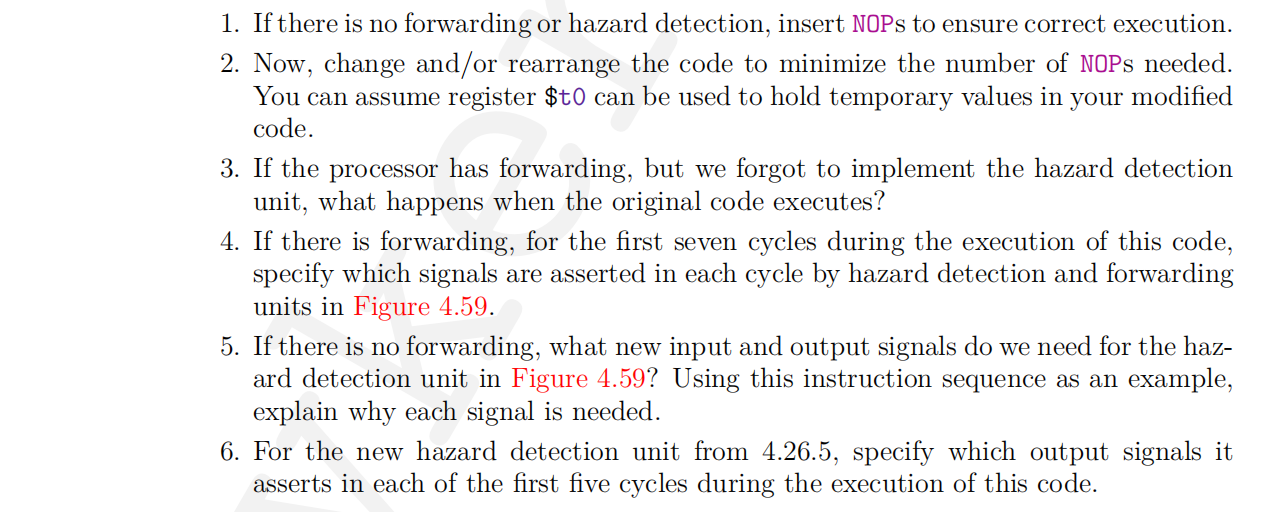
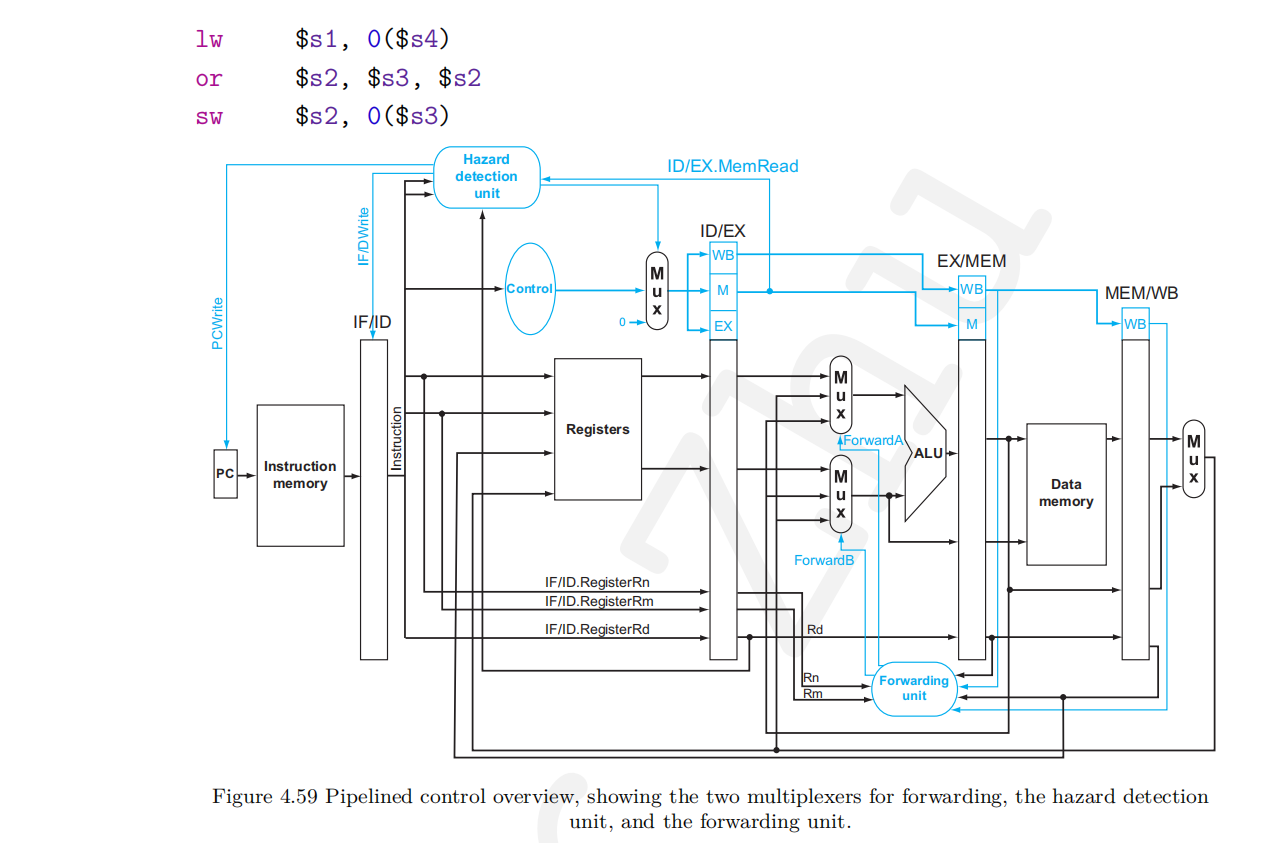
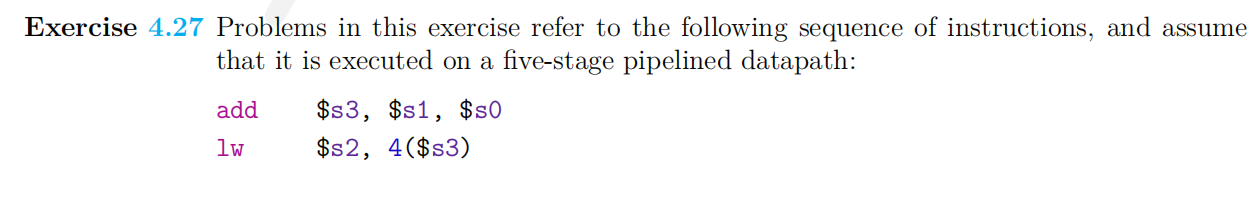
Add $s2,$s0,$S1 IF ID EX ME WB

Sub $s2,$s6,$s1 IF ID EX ME WB

1. There is no possibility in reordering the code to reduce the number of stalls since every instruction must be fetched before any stages later from the same memory where data comes from
2. It cannot be solved by inserting NOPs since NOP itself shall be fetched from instruction memory as well
3. 25%+11%=36% since every data access will lead to a stall



**Solution**: choice 2 since stall demands are detected in the stage of instruction fetching



**Solution**: 1. add $s3,$s1,$s0

Nop

Nop

Lw $s1,4($s3)

Lw $s1,0($s4)

Nop

Or $s2,$s3,$s2

Nop

Nop

Sw $s2,0($s3)

1. There is no way to reduce the number of NOPs by rearranging the code
2. It executes correctly. Only when a instruction uses the result from the preceding load instruction would hazard detection be needed, which is not the case.
3. The first seven cycle:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
| add | IF | ID | EX | ME | WB |  |  |  |  |
| ld |  | IF | ID | EX | ME | WB |  |  |  |
| ld |  |  | IF | ID | EX | ME | WB |  |  |
| or |  |  |  | IF | ID | EX | ME | WB |  |
| sd |  |  |  |  | IF | ID | EX | ME | WB |

(1) Forward A = X; Forward B = X (no instruction in EX stage yet)

(2) Forward A = X; Forward B = X (no instruction in EX stage yet)

(3) Forward A = 0; Forward B = 0 (no forwarding; values taken from

registers)

(4) Forward A = 2; Forward B = 0 (base register taken from result of

previous instruction)

(5) Forward A = 1; Forward B = 1 (base register taken from result of two

instructions previous )

(6) Forward A = 0; Forward B = 2 (rs1 = x15 taken from register;

rs2 = x13 taken from result of 1st ld—two instructions ago)

1. Forward A = 0; Forward B = 2 (base register taken from register file.

Data to be written taken from previous instruction)

1. The hazard detection unit requires the values of $rd from the MEM/WB register and there is no extra need for output signals. To detect potential data hazard between add and ld, the value $rd from EX/MEM should be employed. Besides, the one between the first ld and the or should be detected by using the value of $rd from MEM/WB
2. The first five cycle:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cycle | 1 | 2 | 3 | 4 | 5 |
| Add $s3,$s1,$s0 | IF | ID | EX | ME | WB |
| Lw $s2,4($S3) |  | IF | ID | \*\* | \*\* |
| Lw $s1,0($S4) |  |  | IF | \*\* | \*\* |

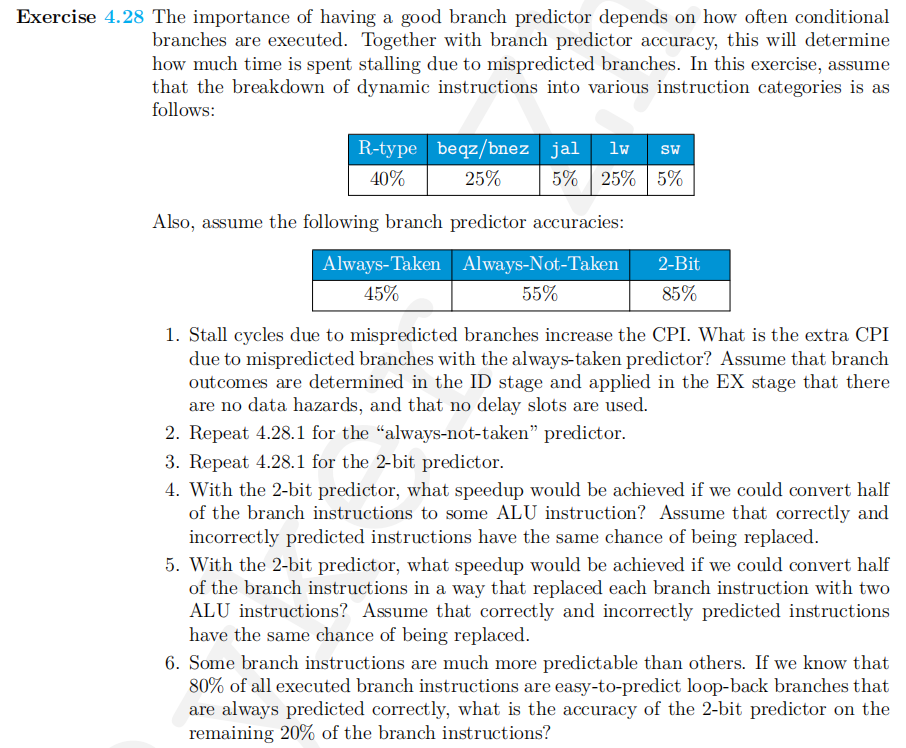
(1) PC Write = 1; IF/ID Write = 1; control mux = 0

(2) PC Write = 1; IF/ID Write = 1; control mux = 0

(3) PC Write = 1; IF/ID Write = 1; control mux = 0

(4) PC Write = 0; IF/ID Write = 0; control mux = 1

(5) PC Write = 0; IF/ID Write = 0; control mux = 1



Solution: 1. Since a mispredicted branch, which makes up 25% of the instruction mi, leas to a three-instruction penalty ,i.e. the instruction in the IF, ID, and EX stage and given that there are 100%-45%=55% iof instruction triggers the flush for the always-taken predictor, the CPI would reach

1+25%3(100%-55%)=1.1.4125

1. 1+25%3(100%-45%)=1.3375
2. 1+25%3(100%-85%)=1.1125
3. Given that half of the branch instruction are converted into ALU instruction, the proportion of branch becomes 12.5%, while leads to a lower CPI : 1+12.5%3(100%-85%)=1.05625

and subsequently the speedup factor:

1.11251.0533

1. Given that half of the branch instructions are converted into two ALU instructions, 12.5% of the branch instructions will have an extra cycle since it would be replaced with ALU instruction and another 12,5% that is not replaced would then have a 100%-85%=15% chance of penalty so the CPI becomes 1+12.5%1+12.5%15%1=1.14375 and thus we can obtain the speedup factor: 1.11251.143750.97
2. Since 80% of the branch instruction are always predicted correctly, only the remaining 20% ave to predict depending on the 2-bit predictor accuracy. The overall accuracy, however, does not change, i.e. 85%. Thus the accuracy of the remaining ones is obtained by solving the equation 80%+20%,which is 25%.