Exercise 1 – DLX ISA

The Figure shows a portion of the DLX memory including a sequence of instruction written in machine language (only the instruction opcodes are written in symbolic language):

		address
BNEZ	00011 00000 1111 1111 1101 1100	CFFF 002CH
ADDI	00011 00011 1111 1111 1111 1111	CFFF 0028H
ADDI	00010 00010 0000 0000 0000 0100	CFFF 0024H
ADDI	00001 00001 0000 0000 0000 0100	CFFF 0020H
SW	00001 00101 0000 0000 0000 0000	CFFF 001CH
BEQZ	00100 00000 0000 0000 0000 0100	CFFF 0018H
SGT	00101 00100 00100 00000000000	CFFF 0014H
LW	00010 00101 0000 0000 0000 0000	CFFF 0010H
LW	00001 00100 0000 0000 0000 0000	CFFF 000CH
ADDI	00000 00011 0000 0100 0000 0000	CFFF 0008H
ADDI	00000 00010 1000 0000 0000 0000	CFFF 0004H
ADDUI	00000 00001 1000 0000 0000 0000	CFFF 0000H

- «Disassemble» the DLX machine language included in the memory portion shown in the figure, so to obtain the corresponding sequence of assembly instructions
- Explain what is the operation performed by the obtained instruction sequence, also pointing out clearly what is the memory location and the size of the data structures involved in this operation.

a) "Disassembling"

Instructions in binary format

		indirizzi
BNEZ	00011 00000 1111 1111 1101 1100	CFFF 002CH
ADDI	00011 00011 1111 1111 1111 1111	CFFF 0028H
ADDI	00010 00010 0000 0000 0000 0100	CFFF 0024H
ADDI	00001 00001 0000 0000 0000 0100	CFFF 0020H
SW	00001 00101 0000 0000 0000 0000	CFFF 001CH
BEQZ	00100 00000 0000 0000 0000 0100	CFFF 0018H
SGT	00101 00100 00100 00000000000	CFFF 0014H
LW	00010 00101 0000 0000 0000 0000	CFFF 0010H
LW	00001 00100 0000 0000 0000 0000	CFFF 000CH
ADDI	00000 00011 0000 0100 0000 0000	CFFF 0008H
ADDI	00000 00010 1000 0000 0000 0000	CFFF 0004H
ADDUI	00000 00001 1000 0000 0000 0000	CFFF 0000H

Corresponding assembly instructions

```
        ADDUI   R1,R0,8000H
        ADDI    R2,R0,8000H (*)
        ADDI    R3,R0,400H
LOOP:   LW      R4,0(R1)
        LW      R5,0(R2) (**)
        SGT     R4,R5,R4
        BEQZ    R4,SKIP
        SW      0(R1),R5
SKIP:   ADDI    R1,R1,4
        ADDI    R2,R2,4
        ADDI    R3,R3,-1
        BNEZ    R3,LOOP
```

(*) in this case 8000H is signed, so the offset is -32K, and R2 = 1^{16}##8000H

(**) Since MDR = R2, the starting address of V2 is the unsigned interpretation of R2's initial value, i.e. FFFF 8000H

b) The obtained instruction sequence processes two vectors, each made out of 1024 words (32 bits) located respectively at address 8000H and FFFF 8000H. By referring to these two vectors as, respectively, V1 and V2, the performed data processing can be expressed as follows:

for i=0…1023: V1[i]=max(V1[i], V2[i]) .

Exercise 2 – sequential DLX

Suppose one has to extend the Instruction Set of the DLX with the following instruction:

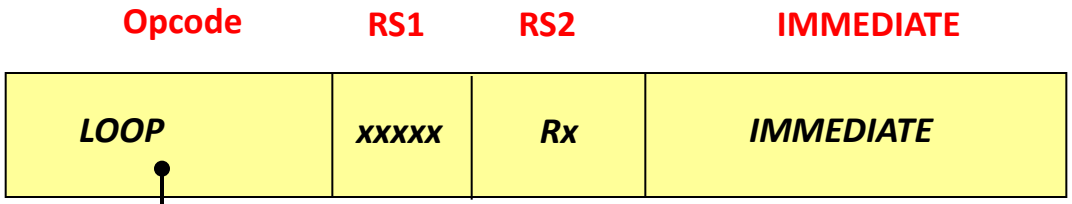
LOOP Rx, IMMEDIATE

where Rx is a general-purpose register and IMMEDIATE is a 16-bit signed immediate. The instruction has to decrement the register Rx, then transfer the control to the instruction specified by means of IMMEDIATE only if the value of Rx is not equal to 0. A possible description of the operations executed by the new instruction is:

Rx ← Rx –1;
if (Rx ≠ 0) then PC ← PC + IMMEDIATE;

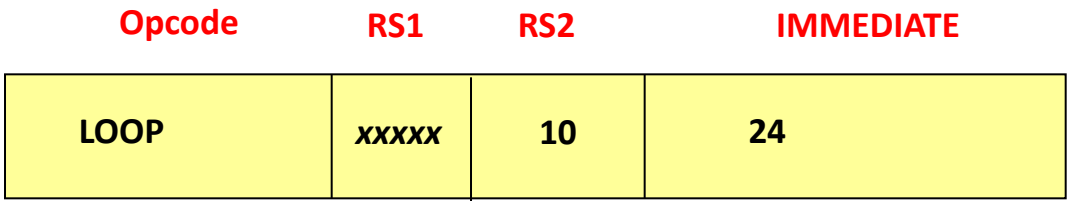
a) Which of the 3 instruction formats of the DLX is more suitable to encode this new instruction? How can the instruction LOOP R10, 24 be encoded? *(1 point).*
b) Referring to the datapath of the sequential DLX (i.e., non-pipelined), determine the state diagram that controls the execution of the new instruction, including also the states needed to the fetch stage and decode stage *(3 points).*
c) *S*tarting from this state diagram and assuming that each memory access requires 4 clocks, compute the CPI (clock-per-instruction) of the new instruction (*1 point)*.
d) Show how the new instruction could be deployed within a loop, written in DLX assembly, summing up the corresponding elements of 2 vectors, V1 and V2, and storing the result in a third vector, V3. Assume that the size of the 3 vectors is 1K and that the elements are words (32 bits) *(2 points)*.
e) *Compute the CPI_{mean} of the sequence of instructions written at the previous point.*

a) To encode the new instruction, we need to use the I format since this format allows specifying a register operand and a 16-bit immediate operand. Hence we choose to encode the LOOP Rx, IMMEDIATE in the I format as shown in the following figure:

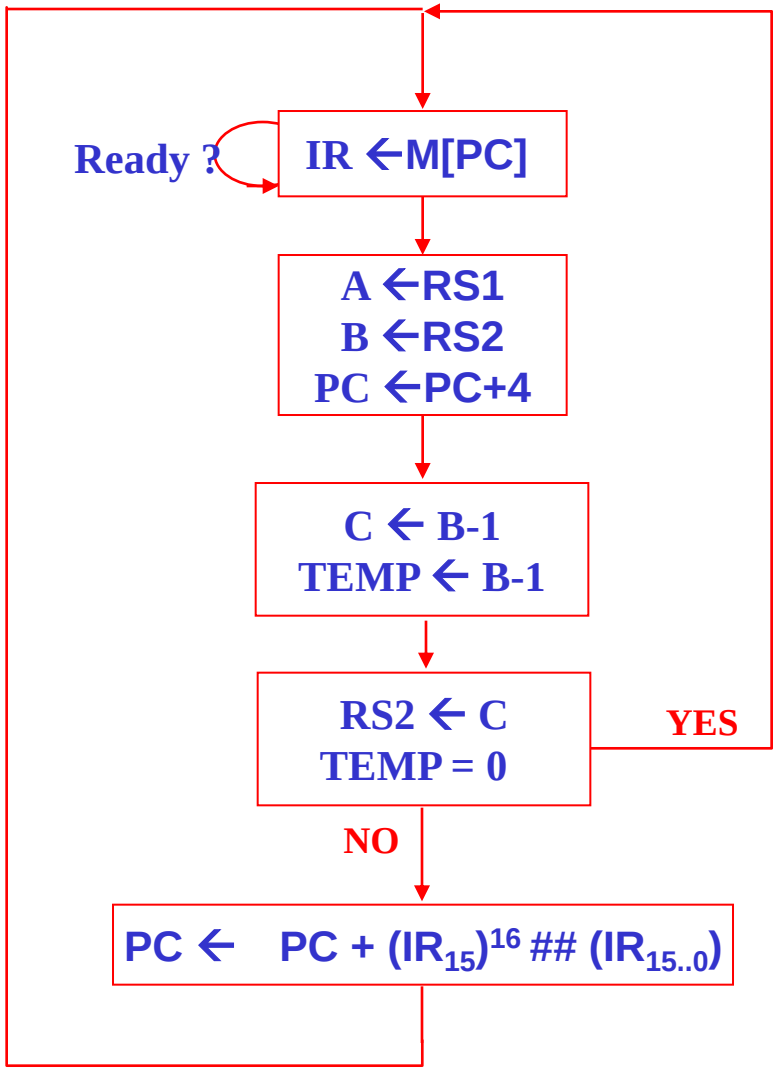


With this notation we mean to represent the binary configuration that encodes the opcode of the new instruction
LOOP Rx IMMEDIATE

LOOP R10, 25



b) State diagram



c) CPI

- *Case LT (Loop-Taken)*

5 states and 1 memory access (every memory access requires 4 clocks):
CPI=8 (5 states + 3 additional clocks needed by memory access).

- *Case LNT (Loop-Not-Taken)*

CPI=7 (4 states + 3 additional clocks needed by memory access).

d)DLX code that exploits the LOOP instruction

```
        ADDI R1,R0,R0          ; index initialization
        ADDI R2,R0,1024        ; iteration counter initialization
START:  LW R3, V1(R1)          ; current element of V1 in R3
        LW R4, V2(R1)          ; current element of V2 in R4
        ADD R3,R3,R4           ; sum between the current elements in R3
        SW V3(R1), R3          ; storing the sum in V3
        ADDI R1,R1,4           ; incrementing the index
        LOOP R2, START         ; jump to label START if the iteration counter is
                               ; not equal to zero
```

e) CPI_{mean} of the code written for the previous point

CPI*mean* = NUM_CLOCKS / NUM_INSTRUCTIONS

NUM_INSTRUCTIONS= 2 + 6*1024 = 6146

NUM_CLOCKS = NUM_ALU*CPI_ALU+NUM_LW*CPI_LW +
NUM_SW*CPI_SW+NUM_LT*CPI_LT+NUM_LNT*CPI_LNT

Instruction	States	Wait	CPI	NUM
Load	6	6	12	2048
Store	5	6	11	1024
ALU	5	3	8	2050
LOOP (taken)	5	3	8	1023
LOOP (not taken)	4	3	7	1

NUM_CLOCKS = …….=60431

CPI*mean* = 60431 /6146 = 9,83

Memory accesses are performed in 4 clocks (i.e. each memory access requires 3 wait states).

Exercise 3 – pipelined DLX (2)

Consider the following sequence of DLX Assembly code:

```
ADDUI R1,R0,8000H
LOOP: SUBI R1,R1,1
LBU R2,V1(R1)
LBU R3,V2(R2)
SB V3(R1),R2
SB V1(R1),R3
BNEZ R1,LOOP
```

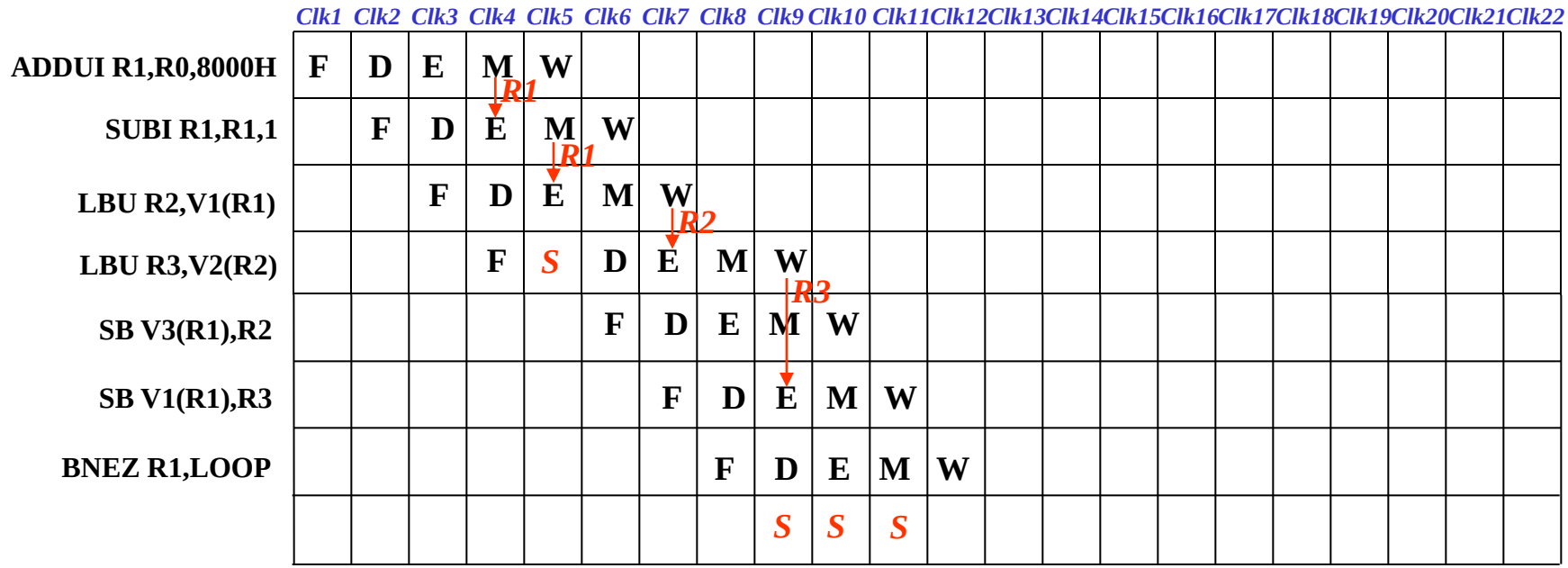
Assuming that:

- The pipelined structure of the DLX
- The Forwarding Unit (FU) minimizing all data hazards is present
- Register File performs *split-cycle* (able to read and write the same register within the same clock cycle)
- Branches are handled using a *predict-not-taken* policy

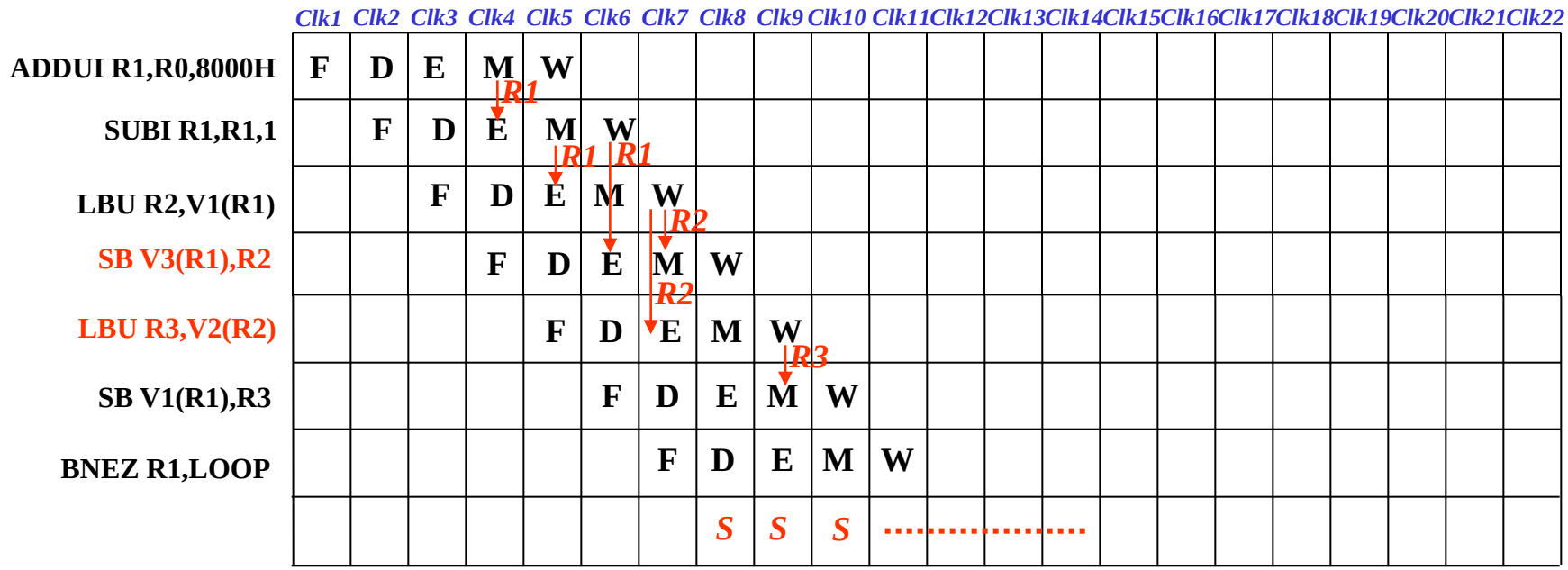
a) Considering just the first instruction and the first loop iteration, draw the flow diagram showing the instruction flow inside the pipeline. In this diagram, mark clearly: a) the presence of stalls, if any; b) the clock cycles where the FU is active; c) for each clock cycle where the FU is active, the two instructions involved in data forwarding and the register on which the hazard has been eliminated thanks to forwarding.

b) Show how to modify the execution order of the instructions so to avoid all stalls without changing the final results produced by the code, and draw the pipeline diagram related to the modified code.

a) **Pipeline diagram**



b) **New instruction sequence avoiding stalls**



Exercise 4

A benchmark program includes the following instruction «mix»:

- ALU: 45 %
- LOAD: 25% (10% of these instructions characterized by the destination register being the same as the source register of a ALU instruction immediately following it)
- STORE: 10%
- BRANCH: 15% (85% taken, 15% not taken)
- JUMP: 5%

a) Considering the **DLX sequential structure**, and assuming that all memory accesses are completed within 3 clocks, compute the mean CPI with regards to the execution of the benchmark program

b) Consider the **DLX pipelined structure**, with Forwarding Unit and Register File with split-cycle, and assuming that branches are handled with a «predict-not-taken» policy. Using the formula:

CPImean = (1+s)

compute the mean CPI with regards to the execution of the benchmark program.

c) Assuming that the clock frequency is the same in both cases, compute the speed-up yielded by the pipelined architecture with respect to that of the sequential architecture with regards to the execution of the benchmark program.

a) CPI_{mean} – sequential DLX sequenziale

$$CPI_{mean} = 0.45*CPI_{ALU} + 0.25*CPI_{LOAD} + 0.1*CPI_{STORE} + 0.85*0.15*CPI_{BRANCH\ TAKEN} + 0.15*0.15*CPI_{BRANCH\ NOT\ TAKEN} + 0.05*CPI_{JUMP}$$

Since in the evaluated situation we have that:

$CPI_{ALU} = 5+2=7$

$CPI_{LOAD} = 6+2*2=10$

$CPI_{STORE} = 5+2*2=9$

$CPI_{BRANCH\ TAKEN} = 4+2=6$

$CPI_{BRANCH\ NOT\ TAKEN} = 3+2=5$

$CPI_{JUMP} = 3+2=5$

the CPI_{mean} can be compute as:

$$CPI_{mean} = 0.45*7 + 0.25*10 + 0.1*9 + 0.85*0.15*6 + 0.15*0.15*5 + 0.05*5 =$$
$$3.15 + 2.5 + 0.9 + 0.765 + 0.1125 + 0.25 = 7.6775$$

b) CPI_{mean} – pipelined DLX

$CPI_{mean} = 1 + s$

Stalls due to «branch taken»

with s = number of stalls per instruction = $\underbrace{1*0.1*0.25}_{\text{Stalls due to load instr.}} + \overbrace{3*0.85*0.15}^{\text{Stalls due to «branch taken»}} + \underbrace{3*0.05}_{\text{Stalls due to jump instr.}} = 0.025 + 0.3825 + 0.15 = 0.5575$

Hence:

$CPI_{mean} = 1 + 0.5575 = 1.5575$

c) Speed-up:

$Speed\ Up_{pipelined/sequential} = CPU_{time}(sequential) / CPU_{time}(pipelined)$

Expressing CPU_{time} as $CPU_{time} = N_{instructions} * CPI_{mean} * T_{ck}$ we get:

$$Speed\ Up_{pipelined/sequential} = (N_{instructions} * CPI_{mean}(sequential) * T_{ck}) / (N_{instructions} * CPI_{mean}(pipelined) * T_{ck})$$

$$= CPI_{mean}(sequential) / CPI_{mean}(pipelined)$$

$$= 7.6775 / 1.5575 = 4.9293$$

Exercise 5 - pipelined DLX

a) Write the DLX assembly code that calculates the following expression:

$$Z = A + B - C$$

where A, B, C and Z are variables whose address is smaller than 64K.

b) By analysing this code, determine how many stalls are required for each instruction in absence of Forwarding Unit

1) in the case of Register File **without split-cycle**
2) in the case of Register File **with split-cycle**

c) Determine how many stalls are required for each instruction in presence of Forwarding Unit (assuming RF with split-cycling), and show in which circumstances (instruction and clock period) the FU is activated, assuming the Register File handles split-cycling

a) DLX assembly code

```
LW R1, A(R0)     ; operand A is loaded from memory
LW R2, B(R0)     ; operand B is loaded from memory
LW R3, C(R0)     ; operand C is loaded from memory
ADD R1, R2, R1   ; A+B
SUB R1, R1, R3   ;  (A+B) - C
SW Z(R0), R1     ; result is written in memory
```

B-1) Stalls in absence of Forwarding Unit, assuming the Register File does not handle split-cycling

	Clk1	Clk2	Clk3	Clk4	Clk5	Clk6	Clk7	Clk8	Clk9	Clk10	Clk11	Clk12	Clk13	Clk14	Clk15	Clk16	Clk17	Clk18	Clk19
LW R1, A(R0)	F	D	E	M	W														
LW R2, B(R0)		F	D	E	M	W													
LW R3, C(R0)			F	D	E	M	W												
ADD R1, R2, R1				F	S	S	D	E	M	W									
SUB R1, R1, R3							F	S	S	S	D	E	M	W					
SW Z(R0), R1											F	S	S	S	D	E	M	W	

The number of stalls for the instructions shown above is detailed as follows:

LW R1, A(R0) : no stalls
LW R2, B(R0) : no stalls
LW R3, C(R0) : no stalls
ADD R1, R2, R1 : 2 stalls,
SUB R1, R1, R3 : 3 stalls
SW Z(R0), R1 : 3 stalls

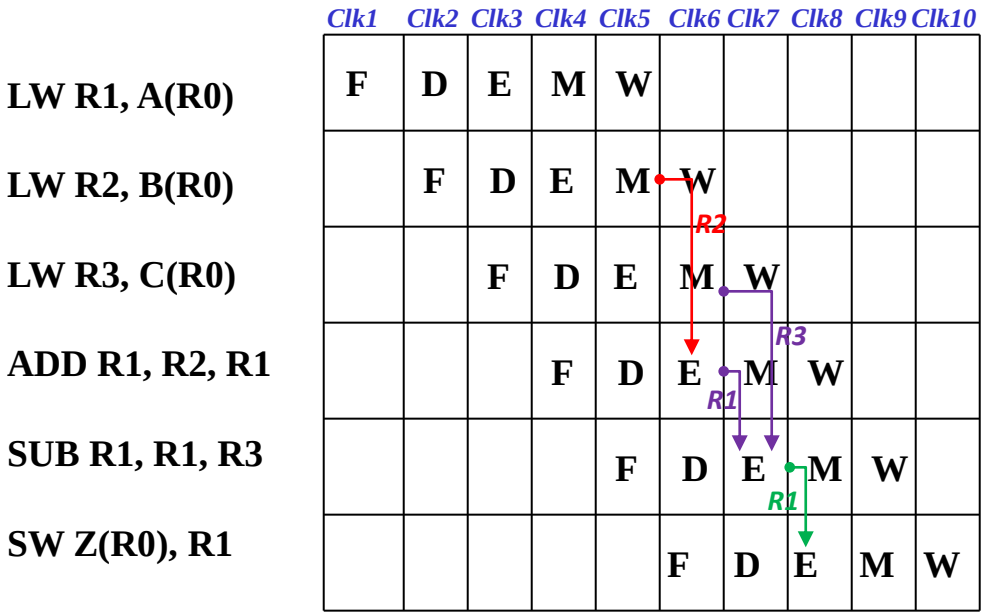
B-2) Stalls in absence of Forwarding Unit, assuming the Register File handles split-cycling

	Clk1	Clk2	Clk3	Clk4	Clk5	Clk6	Clk7	Clk8	Clk9	Clk10	Clk11	Clk12	Clk13	Clk14	Clk15	Clk16	Clk17	Clk18	Clk19
LW R1, A(R0)	F	D	E	M	W														
LW R2, B(R0)		F	D	E	M	W													
LW R3, C(R0)			F	D	E	M	W												
ADD R1, R2, R1				F	S	D	E	M	W										
SUB R1, R1, R3						F	S	S	D	E	M	W							
SW Z(R0), R1									F	S	S	D	E	M	W				

The number of stalls for the instructions shown above is detailed as follows:

LW R1, A(R0) : no stalls
LW R2, B(R0) : no stalls
LW R3, C(R0) : no stalls
ADD R1, R2, R1 : 1 stalls,
SUB R1, R1, R3 : 2 stalls
SW Z(R0), R1 : 2 stalls

c) Stalls in presence of Forwarding Unit:



In presence of FU **no instruction requires stalling.** If the RF handles split-cycling, the FU is active in the following clock cycles:

1) Clock 6 for instruction ADD R1, R2,R1 (R2 produced by LW R2, B(R0) is forwarded to the ALU from the MEM/WB pipeline register).

2) Clock 7 for instruction SUB R1, R1,R3 (R3 ed R1, produced respectively by instructions LW R3, C(R0) and ADD R1, R2,R1, are forwarded to the ALU from the MEM/WB and EX/MEM pipeline registers).

3) Clock 8 for instruction SW Z(R0), R1 (R1 produced by instruction SUB R1, R1, R3, is forwarded to the register SMDR on EX/MEM pipeline register from the EX/MEM pipeline register). Alternatively, it can be forwarded from MEM/WB directly to the memory during clock 9.