①

Loop: LD R4, 0(R2) // Load in R4 the content of R2, with an offset of 0

1 LD R5, 0(R3)

2 ADD R6, R4, R5

3 SD R6, 0(R3) // Store contents of R6 to the memory addrees in 3 with offset

4 ADD R6, R6, R4

5 ADDI R3, R3, #8

6 SUB R20, R4, R3

7 BNZ R20, Loop

Dependencies

RAW

WAR

WAW

a) R4 depends on R2 — 0

R5 " R3 — 1

R6 " R5, R4 — 2

R3 " R6, R3 — 3

R6 " R6, R4 — 4

R20 " R4, R3 — 6

writes	reads	instruction
R3	R3	5
R20	R20	7
-		

R4: $I_1 \rightarrow I_3$ RAW

R5: $I_2 \rightarrow I_3$ RAW

R6: $I_3 \rightarrow I_4$ WAW

~~X R3: $I_1 \rightarrow I_3$~~ ~~WAR~~ X (It only writes in the main memory)

R6: $I_4 \rightarrow I_5$ RAW, WAW

R3: $I_5 \rightarrow I_6$ RAW

R20: $I_7 \rightarrow I_8$ RAW

Source → Target

b) No forwarding

Branch → flush

All memory accesses need two cycles

Timing sequence square

5 sequence machine

IF ← 2 cycles

ID

EX

M ← 2 cycles (LD SD), 1 cycle rest

WB

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	IF	IF	ID	EX	M	M	WB																		
1			IF	IF	ID	EX	M	M	WB																
2					IF	IF	ID	-	-	EX	M	WB													
3							IF	IF	-	ID	-	-	EX	M	M	WB									
4										IF	IF	-	ID	-	-	-	EX	M	WB						
5													IF	IF	-	-	ID	EX	M	WB					
6																	IF	IF	ID	-	EX	M	WB		
7																			IF	IF	ID	-	-	EX	M

(2)

```
i0   lw   $r3, 0($r0)
i1   lw   $r1, 0($r3)
i2   addi $r1, $r1, 1
i3   sub  $r4, $r3, $r2
i4   sw   $r1, 0($r3)
i5   bnz  $r4, i0
```

5 stages pipelined architecture { F, D, Ex, M, WB

load & store need 3 cycles (M)

branch need 2 cycles. (Ex)

↳ evaluation of the condition will be known after decoding.

a)

instruction	Read	Writes
0	$r0	$r3
1	$r3	$r1
2	$r1	$r1
3	$r3, $r2	$r4
4	$r1, $r3	-
5	$r4	-

Source	Target	Register	Type
0	1	$r3	RAW
1	2	$r1	RAW
0	3	$r3	RAW
2	4	$r1	RAW
0	4	$r3	RAW
3	5	$r4	

b)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	F	D	E	M	M	M	WB														
1		F	D	-	-	-	-	E	M	M	M	WB									
2			F	-	-	-	-	D	-	-	-	-	E	M	WB						
3								F	-	-	-	-	D	E	M	WB					
4													F	D	-	E	M	M	M	WB	
5														F	-	D	E	E	-	M	WB

c)

1 iteration takes 16 cycles as the fetch instruction will be on the 17th cycle (and not on the 18th as in the statement says that the result of the branch is known after decoding).

d)

(3)

```
0  LD   R4 0(R1)
1  DIV  R2 R2 R4
2  ADD  R1 R1 #1
3  SUB  R5 R5 #1
4  SD   R4 0(R5)
5  SUB  R6 R1 R5
6  BNEZ R6 I0
```

Source	Target	Register	Type
0	1	R4	RAW
2	5	R1	RAW
3	4	R5	RAW
5	6	R5	RAW

Instruction	Read	Write
0	R1	R4
1	R2, R4	R2
2	R1	R1
3	R5	R5
4	R4, R5	-
5	R1, R5	R6
6	R6	-

a) Not fowarding, Pipelined 5 steps
- Adwitional M for LD
- Pipline freezing for branching
- Branch evaluation is dow during decode

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	F	D	E	M	M	WB														
1		F	D	-	-	-	E	M	WB											
2			F	-	-	-	D	E	M	WB										
3							F	D	E	M	WB									
4							F	D	-	-	E	M	WB							
5								F	-	-	D	E	M	WB						
6											F	D	-	-	E	M	WB			
													F	Flush	-	-	D	E	M	M	WB

(4)

		0	1	2	3	4	5	6	7	8	9	10	11	12
Loop:	LD R4 0(R2)	F	D	E	M	WB								
1	LD R5 0(R3)		F	D	E	M	WB							
2	ADD R6, R4, R5			F	D	-	E	M	WB					
3	SD R6, 0(R3)				F	X	D	E	M	WB	WB			
4	BNZ R6, loop					F	-	D	E	M	WB	WB		

a)
Memory access instruction 3 cycles
Branch instruction 1 cycle
} In addition to the fetch/issue cycle

$C_{loop} = C_{I1} + C_{I2} + C_{I3} + C_{I4} + C_{I5} = (1+3) + (1+3) + (1) + (1+3) + (1+1) = 15$

b)

Target	Source	Register	Type
3	1	R4	RAW
3	2	R5	RAW
4	3	R6	RAW
5	3	R6	RAW

c) Fowarding technique
Branch handling technique is freezing

(6)

a)

```
1  loop: lw   $f0, 0($r1)
2        lw   $f2, 0($r2)
3        mul.f $f4, $f0, $f2
4        add.d $f6, $f6, $f4
5        addi  $r1, $r1, 4
6        addi  $r2, $r2, 4
7        sub   $r3, $r3, 1
8        bnez  $r3, loop
```

Instruction	Read	Write
1	$r1	$f0
2	$r2	$f2
3	$f0, $f2	$f4
4	$f6, $f4	$f6
5	$r1	$r1
6	$r2	$r2
7	$r3	$r3
8	$r3	-

Source	Target	Register	Type
1	3	$f0	RAW
2	3	$f2	RAW
3	4	$f4	RAW
7	8	$r3	RAW

b) Read, write on the same register can be done without conflicts. (write back & decode in the same cycle)
- Branches are flushed the pipeline
- Memory requires 2 cycle
- Branch is known in execution

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	F	D	E	M	WB																	
2		F	D	E	M	WB																
3			F	D	-	-	E	M	WB													
4				F	-	-	D	-	-	E	M	WB										
5							F	-	-	D	-	-	E	M	WB							
6										F	-	-	D	-	-	E	M	WB				
7													F	-	-	D	E	M	WB			
8																F	D	-	-	E	M	WB
																	F	flush			F	

c) 16 · N

d) There is forwarding & branches are predicted as taken
↳ (from memory before write back)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	F	D	E	M	WB												
2		F	D	E	M	WB											
3			F	D	-	E	M	WB									
4				F	-	D	-	E	M	WB							
5						F	-	D	E	M	WB						
6								F	D	E	M	WB					
7									F	D	-	E	M	WB			
8										F	D	-	E	M	WB		

(5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	19
ADDI R3,R3,1	F	D	E	E	M	WB												
Loop: LD R4, 0(R6)		F	D	-	E	M	M	WB										
MULT R5,R4,R4			F	-	D	-	-	E	E	M	WB							
ADD R5,R3,R5					F	-	-	D	E	E	M	WB						
SD R5,0(R6)								F	D	E	-	M	M	WB				
DADDI R1,R1,1									F	D	-	E	E	M	WB			
DADDI R1,R1,1										F	-	D	-	-	E	M	WB	
BNE R1,R2,Loop												F	-	-	D	E	M	WB

* Fowarding * SD & LD take an extra cycle memory * ADD & Mult take an extra execution cycle Not ADDI
* No branch prediction

a) In none, as we are using a pipelined 5 step processor.

R5 ADD → MULT WAW
R1 DADDI → DADDI WAR

b)

Target	source	Register	Type
3	2	R4	RAW
4	3	R5	RAW
5	4	R5	RAW
7	6	R1	RAW

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	F	D	E	M	WB													
2		F	D	E	M	WB	WB											
3			F	D	-	E	E	M	WB									
4				F	-	D	-	E	E	M	WB	WB	WB	WB				
5						F	-	D	-	E	M	WB	WB	WB	WB			
6								F	-	D	E	M	-	WB	X	WB		
7										F	D	E	-	M	WB	X	WB	
8												F	D	E	-	M	WB	X

(7)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	Loop lw $f0, 0($r1)	F	D	E	M	WB																	
2	lw $f2, 0($r2)		F	D	E	M	WB																
3	sub.f $f4, $f0, $f2			F	D	-	-	E	M	WB													
4	mult.d $f4, $f4, $f4				F	-	-	D	-	-	E	M	WB										
5	addd $f6, $f6, $f4						F	-	-	D	E	M	WB										
6	addi $r1, $r1, 4								F	D	E	M	WB										
7	addi $r2, $r2, 4									F	D	E	M	WB									
8	sub $r3, $r3, 1										F	D	E	M	WB								
9	bnez $r3, loop											F	D	E	M	WB							

a)

Reads	writes	Instruction
$r1	$f0	1
$r2	$f2	2
$f0, $f2	$f4	3
$f4	$f4	4
$f4, $f6	$f6	5
$r1	$r1	6
$r2	$r2	7
$r3	$r3	8
$r3	-	9

Source	Target	Register	type
1	3	$f0	RAW
2	3	$f2	RAW
3	4	$f4	RAW
3	4	$f4	WAW
4	5	$f4	RAW
1	6	$r1	WAR
2	7	$r2	WAR
8	9	$r3	RAW

b)
* No fowarding
* Read and write in the same cycle (WB is affected)
* Branches flush the pipeline
* Additional cycle for memory ref-
* Branch evaluation is done after execution

* 5 step procesor w/pipeli

c)

N = 20

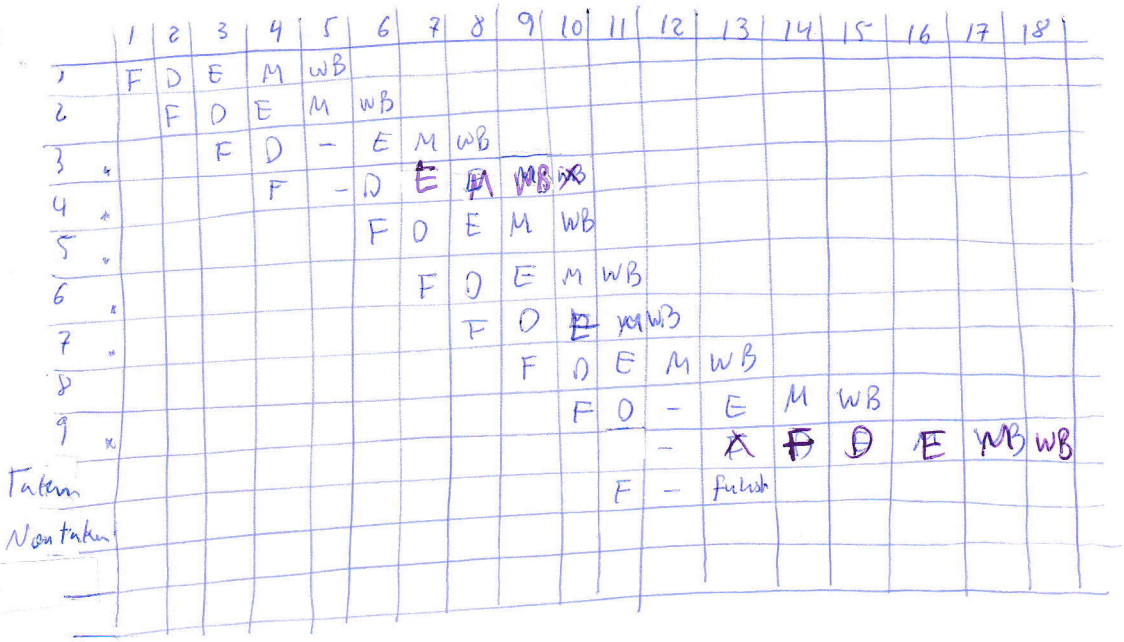
19

* Full fowarding
* Branches are taken.

e)

N = 13

d)



(5)

```
1     DADDUI R3 R1 #40
Loop: L.D. F0, 0(R1)
2     L.D  F2, 0(R2)
3     ADD.D F4, F0, F2
4     S.D F4 0(R1)
5     DADDUI R1, R1, #8
6     DADDUI R2, R2, #8
7     BLE R1, R3, Loop
8
```

Producer	Consumer	Latency
FP ALU	FP ALU	5
FP ALU	Load/store double	4
FP ALU	Branch inst.	4
Load double	FP ALU	2
Load double	Load double	1
Store double	FP ALU	2

* One instruction per cycle (except stalls)
* Pipeline with a single data path

1) F0 : I2 → I4, RAW
F2 : I3 → I4, RAW
F4 : I4 → I5, RAW
R1 : I2 → I6, WAR
R1 : I5 → I6, WAR
R2 : I3 → I7, WAR
R3 : I1 → I8, RAW
R1 : I6 → I8, RAW

2) # number of iterations is 5.
1 + 5 · 13 = 66 cycles
↳ cycles + stalls
1 + 5 · 11 = 56 cycles, 17%

3)

```
1) DADDUI R3, R1, #40
Loop: LD  F0, 0(R1)
      LD  F2, 0(R2)
      <stall> DADDUI R1, R1, #8 }
      <stall> DADDUI R2, R2, #8
      ADD.D F4, F0, F2
      <stall>
      <stall>
      <stall>
      <stall>
      SD F4, 0(R1)  SD F4, -8(R1)
      DADDUI R1, R1, #8 }
      DADDUI R2, R2, #8
      BLE R1, R3, Loop
```

4) Only unrolling (no scheduling)

```
DADDUI R3, R1, #32
Loop: LD F0, 0(R1)
LD F2, 0(R2)
<stall>
<stall>
ADD.D F4, F0, F2
<stall>
<stall>
<stall>
<stall>
SD F4, 0(R1)

LD F6, 8(R1)
LD F8, 8(R2)
<stall>
<stall>
ADD.D F10, F6, F8
<stall>
<stall>
<stall>
<stall>
SD F10, 8(R1)
ADDUI R1, R1, #16
ADDUI R2, R2, #16
BLE R1, R3, Loop.

LD F0, 0(R1)
LD F2, 0(R2)
<stall>
<stall>
ADD.D F4, F0, F2
<stall>
<stall>
<stall>
<stall>
SD F4, 0(R1)
```

1 + 2·23 + 10 = 57 iterations.

```
DADDUI R3, R1, #32
Loop: LD F0, 0(R1)
LD F2, 0(R2)
LD F6, 8(R1)
LD F8, 8(R2)
ADD.D F4, F0, F2
<stall>
ADD.D F10, F6, F8
DADDUI R1, R1, #16
DADDUI R2, R2, #16
SD F4, -16(R1)
<stall>
SD F10, -8(R1)
BLE R1, R3 Loop

LD F0, 0(R1)
LD F2, 0(R2)
<stall>
<stall>
ADD.D F4, F0, F2
<stall>
<stall>
<stall>
<stall>
SD F4, 0(R1)
```

1 + 2·13 + 10 = 37

① * Each instruction 1 cycle * Processor is pipelined * R1 = 0, R2 = 24, R3 = 16

```
Loop 1: LD F2, 0(R1)      1+3
        ADDD F2, F0, F2   1+2
Loop 2: LD F4, 0(R3)      1+3
        MULTD F4, F0, F4  1+4
        DIVD F10, F4, F0  1+10
        ADDD F12, F10, F4 1+4
        ADDI R1, R1, #8   1+0
        SUB R18, R2, R1   1+0
        BNZ R18 loop2     1+1
        SD F2, 0(R3)      1+1
        ADDI R3, R3, #8   1+0
        SUB R20, R2, R3   1+0
        BNZ R20, loop1    1+1
```

Instruction	Additional cycle cost
LD	3
SD	1
ADD	2
MULTD	4
DIVD	10
ADDI	0
SUB	0
BNZ	1

a) 1 iteration of loop 1
3 iterations of loop 2

Loop 2 cycles: 4 + 5 + 11 + 5 + 1 + 1 + 2 = 29

Loop 1 – loop 2 cycles: 4 + 3 + 2 + 1 + 1 + 2 = 13

13 + 29 · 3 = 13 + 87 = 100

b)

```
Loop 1: LD F2, 0(R1)      1+3
        ADDD F2, F0, F2   1+2
Loop 2: LD F4, 0(R3)      1+3
        MULTD F4, F0, F4  1+4
        DIVD F10, F4, F0  1+10
        ADDD F12, F10, F4 1+2

        LD F4 0(R3)       1+3
        MULTD F4, F0, F4  1+4
        DIVD F10, F4, F0  1+10
        ADDD F12, F10, F4 1+2

        LD F4 0(R3)       1+3
        MULTD F0, F0, F4  1+4
        DIVD F10, F4, F10 1+10
        ADD F12, F10, F4  1+2

        ADDI R1, R1, #24  1+0
        SUB R18, R2, R1   1+0
        BNZ R18 loop2     1+1

        SD F2, 0(R3)      1+1
        ADDI R3, R3, #8   1+0
        BNZ R20, loop1    1+1
```

b) Iterations with loop 2 unrolled

85 iterations

c) $\frac{100}{85} = 1{,}17\ \%$ improvement

---

②

Source	Target	Latency
FP ALU	FP ALU	6
FP ALU	Store double	3
Load double	FP ALU	2
Load double	Store double	0

```
Loop: LD F0 0(R1)
      LD F2 0(R2)
      stall x2
      ADDD F4, F2, F0
      SD F4 0(R3)
      stall x3
      DADDUI R1, R1, -8
      DADDUI R2, R2, -8
      DADDUI R3, R3, -8
      BNE R1, R4, Loop
```

a) 13 · N iterations

Arrays of 40 000 bytes

× 40 000 iterations · 13 cycles / iterations = 519 987 ≈ 520 000

10 · 40 000 = 400 000 cycles

b)

```
Loop: LD F0 0(R1)
      LD F2 0(R2)   <stall x2>
      ADDD F4 F2 F0
      DADDUI R1 R1 #8
      DADDUI R2 R2 #8
      DADDUI R3 R3 #8
      SD F4 0 R3
      BNE R1 R4 loop
```

c)

```
Loop: LD F0 0R1
      LD F2 0R2
      <stall> x2
      ADD F4 F2 F0
      LD F4 -8R1
      LD F2 -8R2
      DADDUI R1 R1 -16
      DADDUI R2 R2 -16
      DADDUI R3 R3 -16
      SD F4 0R3
      BNE R1 R4 loop
```

12 · 20k = 240 000 cycles

```
Loop: LD F0 0(R1)
      LD F2 0(R2)
      <stall>
      <stall>
      ADDD F4,F2,F0
      LD F0 -8(R1)
      LD F2 -8(R2)
      <stall>
      SD F4,0(R3)
      ADDD F4,F2,F0
      LD F0 -16(R1)
      LD F2 -16(R2)
      <stall>
      SD F4 -8(R3)
      ADDD F4 F0,F2
      LD F0 -24(R1)
      LD F2 -24(R2)
      <stall>
      SD F4 -16(R3)
      ADDD F4,F0,F2
      DADDUI R1 -32
      DADDUI R2 -32
      DADDUI R3 -32
      SD F4 8(R3)
      BNE R1,R4,Loop
```

10 000 iterations · 25 cycles/iteration = 250 000

```
LD F0 0(R1)
LD F2 0(R2)
LD F6 -8(R1)
LD F8 -8(R2)
LD F12 -16(R2)
LD F14 -16(R2)
LD F18 -24(R1)
LD F20 -24(R2)
ADDD F4, F0, F2
ADDD F10, F6, F8
ADDD F16, F14, F12
ADDD F22, F20, F18
SD F4 0(R3)
SD F10 -8(R3)
SD F6 -16(R3)
SD F22 -24(R3)
DADDIU R1 R1 #-32
DADDIU R2 R2 #-32
DADDIU R3 R3 #-32
BNE R1, R4, Loop
```

200 000 cycles

③

* One instruction per cycle
* Stalls when data dependency

Instruction	Latency (additional)
ldc1	2
sdc1	2
add.d	4
mul.d	6
addi	0
subi	0
bneq	1

b)

```
1  Loop  lcd1 $F0  0($to)
2        lcd1 $f2  0($t1)
3        add.d $f4, $f0, $f2     <- stall x 2
4        mul.d $f4, $f4, $f6     <- stall x 4
5        sdc1 $f4, 0($t2)        <- stall x 6
6        addi $to, $to, 8
7        addi $t1, $t1, 8
8        subi $t3, $t3, 1
9        bneq $t3 Loop
10       addi $t2 $t2
```

8

a)

	Instruction	Read	Writes
1		$to	$f0
2		$t1	$f2
3		$f0 $f2	$f4
4		$f4 $f6	$f4
5		$f4 $t2	-
6		$to	$to
7		$t1	$t1
8		$t3	$t3
9		$t3	-
10		$t2	$t2

Source	target	Instruction	type
1	3	$f0	RAW
2	3	$f0	RAW
4	5	$f4	RAW
8	9	$t3	RAW
3	4	$f4	RAW

b) 21 + 1 || 21 cycles per iteration

c)

```
Loop: $f0  0($to) ldc1
      ldc1 $f2  0($t1)
      addi $to $to 8
      addi $t1 $t1 8
      add.d mul.d $f4 $f0 $f2
      Subi $t3 $t3 1
      <stall> x 3
      mult.d $f4 $f4 $f6
      <stall> x 6
      sdc1 $f4 0($t2)
      bneq $t3 loop
      addi $t2 $t2 8
```

18 cycles per iteration

d)

```
loop: ldc1  $f0  ($t0)        25n + 1
      ldc1  $f2  ($t1)
      ldc1  $f6  8($t0)
      ldc1  $f8  8($t1)
      ldc1  $f10 16($t0)
      ldc1  $f12 16($t1)
      ldc1  $f20 24($t0)
      ldc1  $f22 24($t1)
      add.d $f4  $f0  $f2
      add.d $f8  $f6  $f8
      add.d $f16 $f10 $f12
      add.d $f22 $f20 $f18
      subi  $t3  $t3  4
      mult  $f4  $f4  $f6
      mult  $f10 $f10 $f6
      mult  $f16 $f16 $f6
      mult  $f22 $f22 $f6
      addi  $t0  $t0  32
      addi  $t1  $t1  32
      <stall>
      sdc1  $f4  -32($t2)
      sdc1  $f10 -24($t2)
      sdc1  $f16 -16($t2)
      sdc1  $f10 -8($t2)
      bnez  $t3  loop
      addi  $t2  $t2  8
```