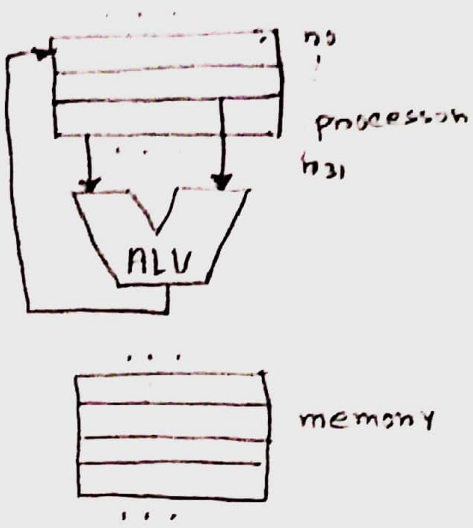
Hanipriya Das Aspita
2502310 62

Computer Architecture

Instruction Pipelining

MIPS:

- Microprocessor without Interlocked Pipelined Stages (MIPS)
- 32 registers, 32 bit each (4 byte)
- Uniform length instructions
- RISC load-store architecture
- MIPS architecture use এর goal-
  - performance maximize করা
  - cost & time reduce করা.



Every instruction in MIPS is represented in 32 bits. MIPS are basically three types of instruction:

① R-type ② I-type ③ J-type

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R:	op	rs	rt	rd	shamt	funct

I:	op	rs	rt	address / immediate

J:	op	target address

op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode
address: offset for load/store instruction
immediate: constants for immediate instructions

What is pipelining?

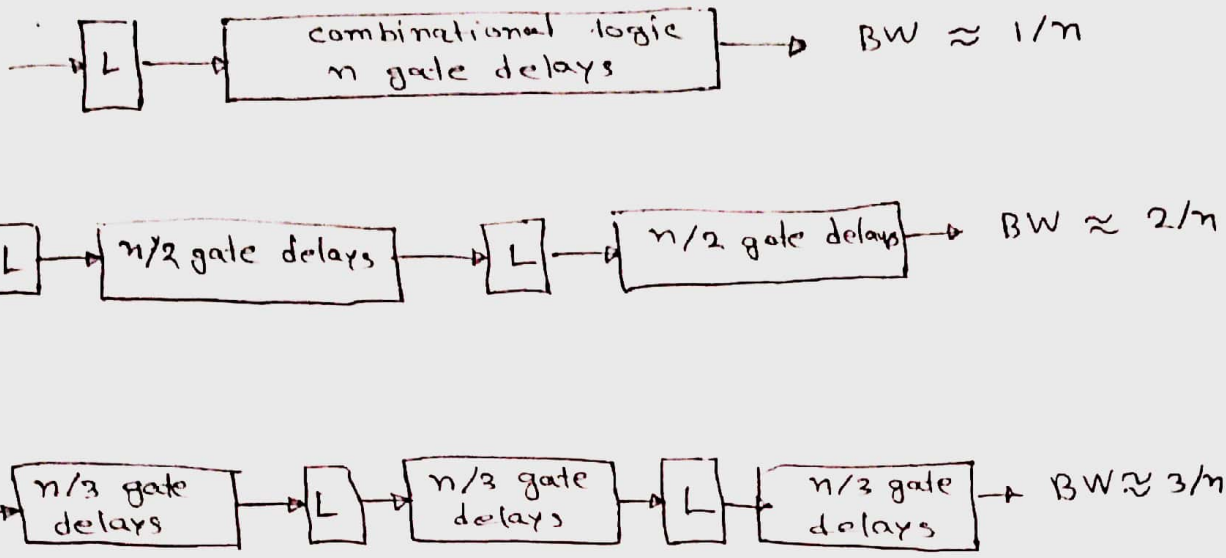
- Pipelining is an implementation technique whereby multiple instructions are overlapped in execution.
- It takes advantages of parallelism that exists among the actions needed to execute an instruction.
- Today, pipelining is the key implementation technique used to make fast CPUs.

Characteristics:

- Pipeline does not reduce the latency of a single task, it improves throughput (transactions / unit time) of entire workload.
- Pipeline rate is limited by the slowest pipeline stage.
- Potential speedup = No. of pipe stages.
- Unbalanced length of pipe stages reduces speed up.
- Time to fill and time to drain pipeline reduces speedup

Pipelining in Circuits :

- Pipelining partitions the system into multiple independent stages with added buffers between the stages.
- Pipelining can increase the throughput of a system.



① pipelining হলো অনেকগুলো stage এর একটা series, সেখানে প্রত্যেকটা series এ কোনো না কোনো কাজ হতেই থাকে।

② এই technique use করে একটা process কে অনেকগুলো subprocess এ ভাগ করা যায়।

③ stage গুলো এমনভাবে সাজানো থাকে যেন আগেরটার output পরেরটার input হিসেবে কাজ করে।

④ প্রত্যেকটা subprocess এর আবাস্থানে Register buffer থাকে যা পরের sub task এ input হিসেবে যায়।

⑤ এই stage গুলো একসাথে pipe এর মতো structure create করে।

Instruction Execution cycle :

Each instruction can take at most 5 clock cycles.

① Instruction Fetch cycle (IF):

- Based on PC, fetch the instruction from memory
- Update / increment PC.
- এই stage এর কাজ হলো memory থেকে instruction কে microprocessor এ আনা।
- current instruction এর 32 bit (4 byte) পরে next instruction থাকে, সেজন্য current instruction এর সাথে 4 byte যোগ করে পরের instruction count করে IF stage.

② Instruction Decode / Register fetch cycle (ID):

- Decode the instruction + register read operation
- Fixed field decoding
- Equality check of registers.
- Computation of branch target address if any.
- opcode বিশ্লেষন করে বুঝতে হবে instruction এ কি কাজ করতে বলা হয়েছে।
- Register file থেকে operand গুলো ID stage এ এনে ALU তে পাঠাতে হবে।

  add R_1, R_2, R_3 এই Register গুলো থেকে value গুলো ALU তে পাঠাতে হবে।
- যদি কোনো branch instruction থাকে, তবে ঐ branch instruction এর target address এই stage এ count হয়।

③ Execution / Effective address cycle (EX):

- Memory reference instruction : Calculate the effective address
- Register register ALU instruction (add, sub, mul)
- Register immediate ALU instruction
- Example: LW R1, 8(R2), Effective address = [R2] + 8
  Add R1, R2, R3
- এই stage এ normal ALU এর কাজ (add, sub, mul) গুলো হয়।

④ Memory access cycle (MEM):

- Load instruction : Read from memory using effective address.
- Store instruction: Write the data in the register to memory using effective address.
- Load / Store instruction এর ক্ষেত্রে MEM stage এ memory access করার প্রয়োজন পড়ে।
- Load : memory থেকে microprocessor এ data আসে
- Store : register থেকে memory তে data যায়।

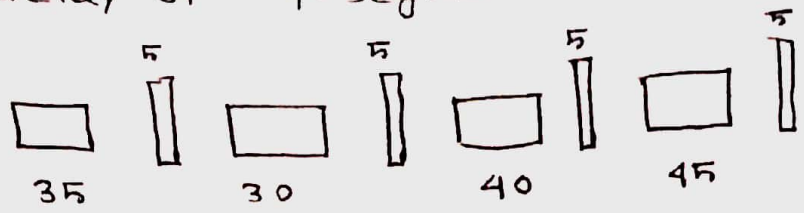
⑤ Write-back cycle (WB):

- Register - register ALU instruction / Load instruction
- Write the result into the register file
- এই stage এ যে data গুলো আসবে সেগুলো আবার Register file এ write হবে।

Answer to the q. no - 4

Here, the time delay of 4-segment pipeline:

$t_1 = 35$ ns
$t_2 = 30$ ns
$t_3 = 40$ ns
$t_4 = 45$ ns



But maximum delay time = 45 ns.
Interface register delay time = 5 ns

∴ Total maximum delay time = (45 + 5) ns
= 50 ns.

For 1st instruction, it will end after 4 stages.
But next 99 intruction will take 1 clock cycle to finish their execution.

So, time to complete 100 instruction is

$= (1 \times 4 \times 50) + (99 \times 50)$ ns

$= 200 + 4950$ ns

$= 5150$ ns.

Cycles required to implement instructions :

- Branch instruction — 4 cycles
- Store instruction — 4 cycles
- All other instruction — 5 cycles

Visualization of Pipeline :

instruction	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	ME	WB				
i+1		IF	ID	EX	ME	WB			
i+2			IF	ID	EX	ME	WB		
i+3				IF	ID	EX	ME	WB	
i+4					IF	ID	EX	ME	WB

Instruction Encoding :

Answer to the q. no - 5

I	0	XXXXX (5 bits) 32 combinations	5 bit Reg name	5 bit Reg name
II	1	00/01	5 bit Reg name	8 bit memory address
III	1	10/11	XXXXX XXX (8 bits) 512 combinations	5 bit Reg name

4) Pipeline Hazards :

Hazards : circumstances that would cause incorrect execution if next instruction is fetched and executed.

There are 3 different types of hazards :

(1) Structural hazards : Different instruction, at different stages, in the pipeline want to use the same hardware resources.

(2) Data hazards : An instruction in the pipeline requires data to be computed by a previous instruction still in the pipeline.

(3) Control hazards : Succeding instruction, to put into pipeline, depends on the outcome of a previous branch instruction already in pipeline.

Structural Hazard :

if one read port in memory.

	c1	c2	c3	c4	c5	c6	c7	c8	
Load	IF	ID	EX	ME	WB				
ins 1		IF	ID	EX	ME	WB			
ins 2			IF	ID	EX	ME	WB		
ins 3				IF	ID	EX	ME	WB	

This happens when we are using uniport memory. The problem happens in the 4th instruction. this 4th instruction also trying to access memory in the IF stage. But 1st Load instruction is also trying to access the memory.

Resolving Structural Hazard :

Eliminate the use of same hardware for two different things at the same time.

Solution 1: Wait

- must detect the hazard
- must have mechanism to stall

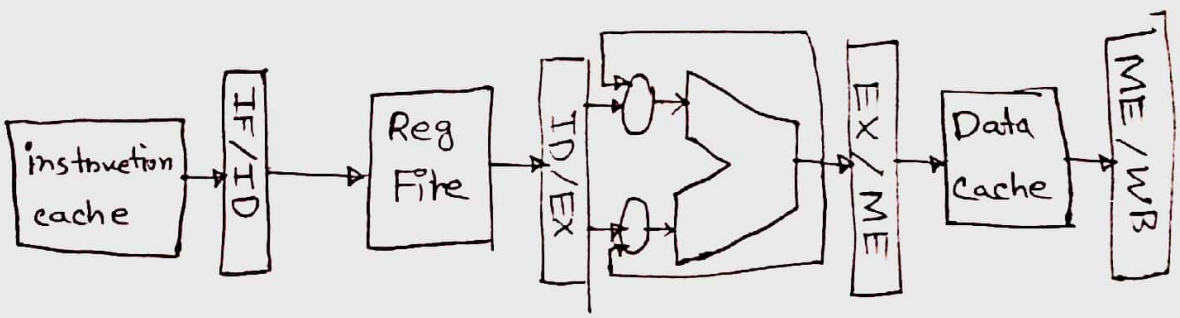
- But it has some problem also.

– instruction - 2 এর পরে একটা cycle stall দেওয়া হলো, instruction - 4 (IF) stage এ যাবে clock cycle ৫ এ. But ঐ time এ instruction - 1 (MEM) stage এ আছে। এখন, ins - 1 ALU হলে কোনো সমস্যা নাই, কিন্তু যদি Load/Store হয় তবে, আরেকটা clock cycle stall হতে হবে।

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Load	IF	ID	EX	ME	WB					
ins 1 L/S		IF	ID	EX	ME	WB				
ins 2			IF	ID	EX	ME	WB			
stall				○	○	○	○	○		
ins 3					IF	ID	EX	ME	WB	

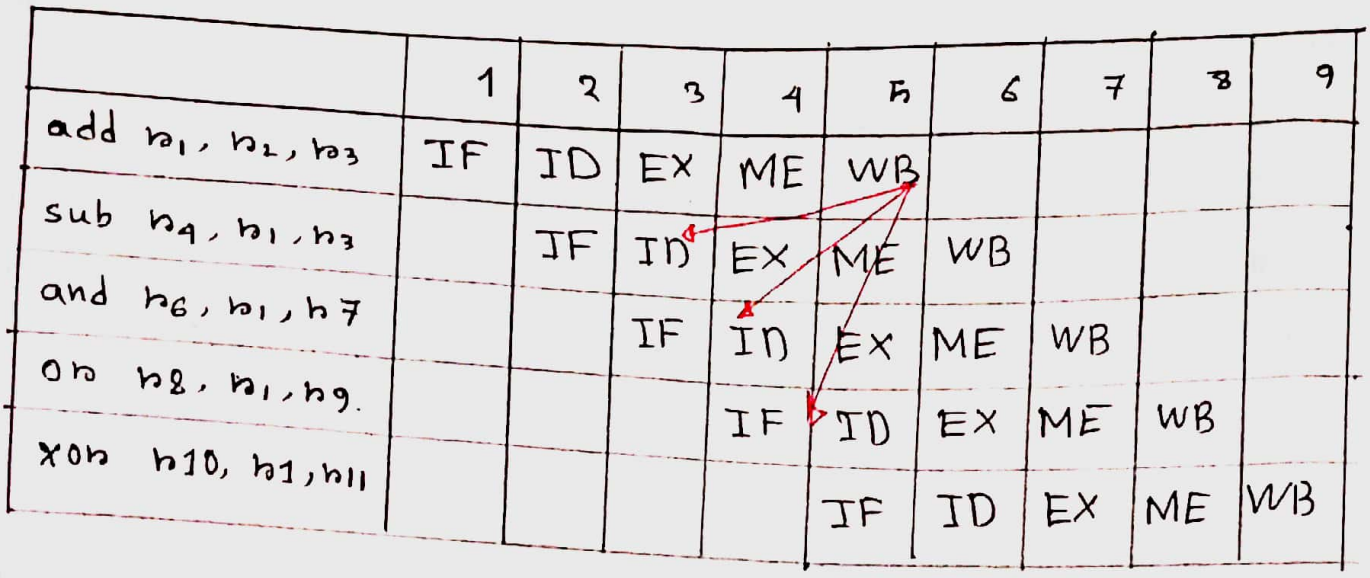
Solution 2: Duplicate handware

- Multiple such units will help both instruction to progress.
- If we increase the number of read port memory such that one read-port can be used always by IF unit and other read-port can be used always by the data access unit.
- We can separate memory, one for exclusively storing instruction and other for exclusively storing the data. We use data cache and instruction cache.



Data Hazard :

	1	2	3	4	5	6	7	8	9
add r_1, r_2, r_3	IF	ID	EX	ME	WB				
sub r_4, r_1, r_3		IF	ID	EX	ME	WB			
and r_6, r_1, r_7			IF	ID	EX	ME	WB		
or r_8, r_1, r_9				IF	ID	EX	ME	WB	
xor r_{10}, r_1, r_{11}					IF	ID	EX	ME	WB



এখানে add ins এর 5th cycle এর পরে আমরা r1 এর value টা পাব। But নিচের সবগুলো ins তাদের ID stage এ r1 এর value access করতে চাচ্ছে. যার ফলে এখানে data hazard create হচ্ছে। basically এইটে (RAW) hazard, Read after Write, আগের instruction Write করার আগেই যদি পরের instruction Read করতে যায়, তাহলে RAW Hazard হয়

এখানে. add এর পরের 3টা instruction sub, and, or এর ক্ষেত্রে data Hazard হবে। but xor এ কোনো data hazard হবে না, যেহেতু xor এর ID stage add এর WB এর পরে আছে।

Types of data hazards :

① Read after Write (RAW):

Instruction j tries to read operand before Instruction I writes it.

i : add r1, r2, r3

j : sub r4, r1, r3

- caused by a data dependence
- This ~~haz~~ hazard results from an actual need for communication.
- যখন আগের instruction write করার আগেই পরের instruction Read করতে যায়, তখন Raw hazard হয়।
- ফলে calculation ও ভুল আসে। যেহেতু add এর ans টা sub এ input হিসেবে লাগছে।

② Write after Read (WAR):

Instruction j writes operand before instruction i reads it.

- co

i : sub r4, r1, r3

j : add r1, r2, r3

K : mul r6, r1, r7

- called an antidependence by compiler writers.
- This results from the reuse of the name r1.
- Can't happen in MIPS 5 stage pipeline bez:
  - All instructions take 5 stages.

• Reads are always in stage 2, and
• Writes are always in stage 5.

- আগের ins read করার আগেই যদি পরের instruction write করে ফেলে, তখন WAR hazard হয়।
- এটা হয় out of order execution এ। প্রথানে যদি ins_i r1 কে read করতে গিয়ে যদি দেখে ins_j already r1 এ write করে ফেলেছে, তাহলে ins_i যে input টা নিবে তা ভুল হবে।

(3) Write after Write (WAW):

Instruction j writes operand before instruction i writes it.

i: sub r1, r4, r3
j: add r1, r2, r3
k: mul r6, r1, r7

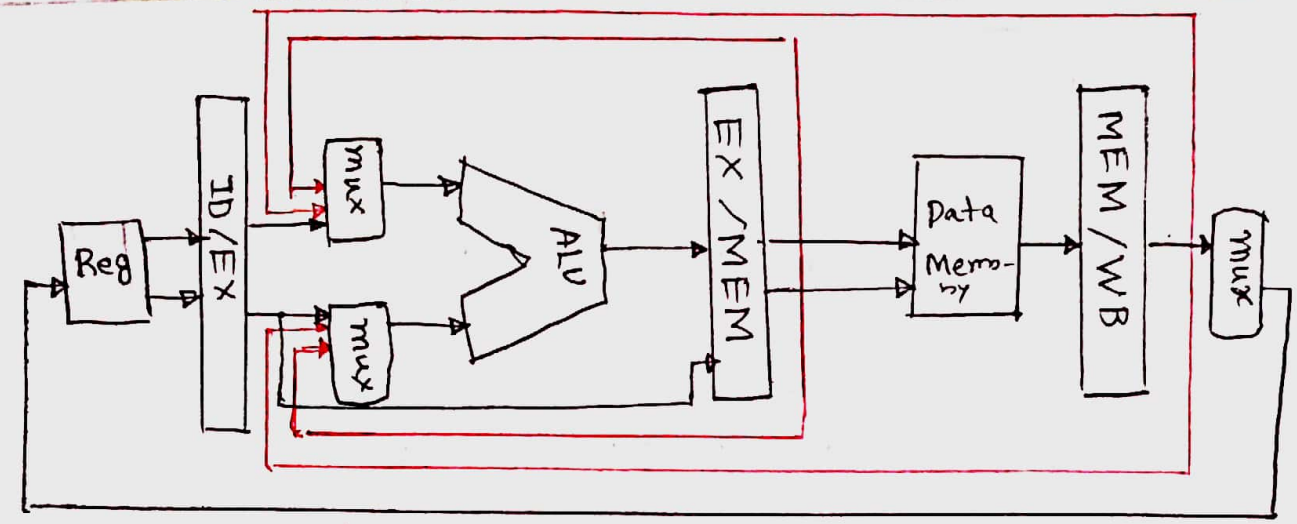
- Called an output dependence.
- This also results from the reuse of name r1.
- Can't happen in MIPS 5 stage pipeline bcz:
  - All instructions take 5 stages.
  - Writes are always in stage 5.
- আগের ins write করার আগেই যদি পরের ins write করে ফেলে, তখন WAW hazard হয়।
- এটাও হয় out of order execution এ, ins_j, ins_i এর পরে r1 এ write করার কথা, তখন যদি ins_j আগে write করে এবং পরে ins_i write করে তাহলে, যখন ins_K r1 read করবে, তা ভুল হবে।

Resolving Data Hazard :

Solution 1 : Operand forwarding

- Forward data if possible.

	1	2	3	4	5	6	7	8	9
add r_1, r_2, r_3	IF	ID	EX	ME	WB				
sub r_4, r_1, r_3		IF	ID	EX	ME	WB			
and r_6, r_1, r_7			IF	ID	EX	ME	WB		
or r_8, r_1, r_9				IF	ID	EX	ME	WB	
xor r_{10}, r_1, r_{11}					IF	ID	EX	ME	WB



- Data hazard even with operand forwarding.
যদি এরকম Load ~~store~~ instruction এর পরে কোনো ALU instruction থাকে, তখন operand forwarding এ কাজ হয় না, কারণ Load ins এ value টা MEM stage এর পরে পাওয়া যায়, তাই Load এর পর ALU থাকলে একটা ~~sta~~ cycle stall দিতেই হয়।

Solution – 2 : Softwan Re-.Scheduling

- instruction গুলো যদি Re-scheduling করা যায়, যাতে কোনো Load instruction এর পরে কোনো ALU instruction না থাকে, তাহলে stall কমানো যায়।
- একটা cycle stall দিয়েও solve করা যায়।

*Data hazard even after operand forwarding :

	1	2	3	4	5	6	7	8	9
Lw r_1, 0(r_2)	IF	ID	EX	ME	WB				
sub r_4, r_1, r_6		IF	ID	EX	ME	WB			
and r_6, r_1, r_7			IF	ID	EX	ME	WB		
Or r_8, r_1, r_9				IF	ID	EX	ME	WB	

* Relolving Load – ALU hazard with stall :

	1	2	3	4	5	6	7	8	9
Lw r_1, (0(r_2)	IF	ID	EX	ME	WB				
sub r_4, r_1, r_6		IF	ID	*	EX	ME	WB		
and r_6, r_1, r_7			IF	*	ID	EX	ME	WB	
Or, r_8, r_1, r_9				*	IF	ID	EX	ME	WB

Answer to the q. no - 6 :

Here, a, b, c, d, e and f in memory.

$a = b + c$

$d = e - f$

LW	Rb, b	5
LW	Rc, c	6
ADD	Ra, Rb, Rc	8
SW	a, Ra	9
LW	Re, e	10
LW	Rf, f	11
SUB	Rd, Re, Rf	13
SW	d, Rd	14

After Rescheduling ;

LW	Rb, b	5
LW	Rc, c	6
LW	Re, e	7
ADD	Ra, Rb, Rc	8
LW	Rf, f	9
SW	a, Ra	10
SUB	Rd, Re, Rf	11
SW	d, Rd	12

Answer: In the q. no – 7

For unpipelined architecture;

$CCT_u = 1/1.5 = 0.67$ ns

$CPI_u = 5$

∴ Ex-T_u per instruction = $CCT_u \times CPI_u$

$= 0.67 \times 5$

$= 3.33$ ns / instruction

For pipelined architecture;

$CCT_p = 1/1 = 1$ ns.

CPI_p = Base CPI + stall CPI.

= 1 + (Memory stall + Branch stall + Load-ALU st

= $1 + (0.3 \times 0.05 \times 50) + (0.2 \times 0.3 \times 2) + (0.1 \times 1)$

= $1 + 0.75 + 0.12 + 0.1$

= 1.97

∴ Ex-T_p per instruction = $CCT_p \times CPI_p$

$= 1 \times 1.97$

$= 1.97$ ns / instruction

∴ Speedup = $\frac{Ex\text{-}T_u}{Ex\text{-}T_p} = \frac{3.33}{1.97} = 1.69$

Answer to the q. no - 5

Without operand forwarding:

ID of nth instruction can be only after WB of (n-1)th instruction

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L.D	F	D	X	M	W											
ADD		F	*	*	*	D	X	M	W							
L D						F	*	*	*	D	X	M	W			
ADD										F	*	*	*	D	X	M

Instruction reach WB at clock cycles 5, 9, 13, 17, 21 ...

Last ADD instruction reaches WB in

$5 + (1999 \times 4) = 8001$ cycles

CPI = 8001 / 2000 = 4.0005

With operand forwarding;

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L.D	F	D	X	M	W											
ADD		F	*	D	X	M	W									
L.D				F	D	X	M	W								
ADD					F	*	D	X	M	W						

Every ADD after LD has a stall,

But LD after ADD don not have a stall,

Instruction reach WB at clock cycles 5, 7, 9, 10, 11, 13

Last ADD instruction reaches WB in

$7 + (999 \times 3) = 3004$ cycles

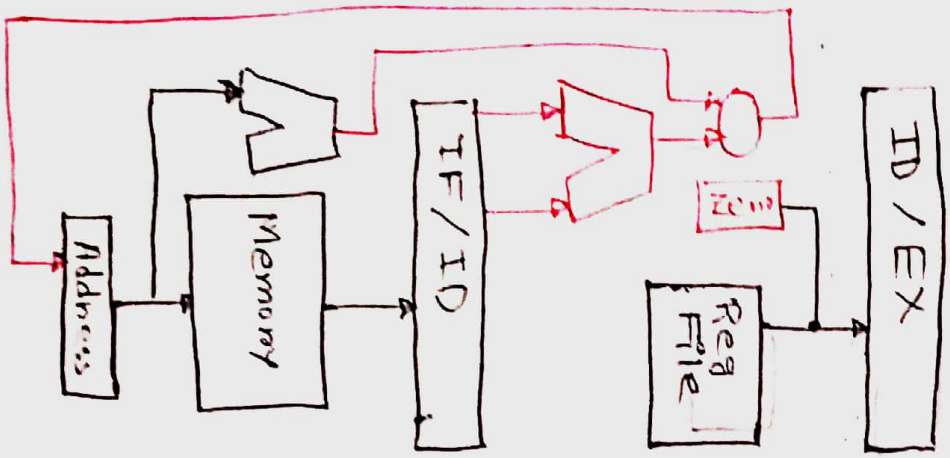
∴ CPI = 3004 / 2000 = 1.502

Control Hazard :

Control hazards on branches : Three stage stall

	1	2	3	4	5	6	7	8	9
10: beq $r1, $r4, 86	IF	ID	EX	ME	WB				
14: and $r2, $r3, $r5		IF	ID						
15: or $r6, $r1, $r7			IF	ID					
22: add $r8, $r1, $r9				IF	ID				
36: xor $r10, $r1, $r11					IF				

Branch Optimized MIPS Pipeline :



- এখানে, ins 10 এ branch instruction r1 ও r3 এর মান equal হলে এটি 36 no ins এ jump করবে। এখন r1 ও r3 এর মান equal হবে কিনা এইটা আমরা জানতে পারব ins_10 এর MEM stage এর শেষে অর্থাৎ 4th cycle এর শেষে, তাহলে 5-th cycle এ 36 no instruction IF stage এ যাবে। কিন্তু এর মাঝে আরো 3 টা instruction pipe line এ ঢুকে গেছে। তাহলে এখানে 3 টা stage stall হল।
- এখন branch pipeline optimize করে instruction টা jump করবে কিনা এইটা আমরা ID stage এই check করে ফেলতে পারি।

Four Branch Hazard Alternatives :

▷ Stall untill branch direction is clear :

- The moment you come to know that the instruction that is fetched is a branch, then you may have to stall until the branch direction is clear.

▷ Predict Branch not taken :

- Execute succesor instruction in sequence.
- "squash" instructions in pipeline if branch actually taken

	1	2	3	4	5	6	7	8	9
Untaken branch ins	F	D	X	M	W				
ins i+1		F	D	X	M	W			
ins i+2			F	D	X	M	W		
ins i+3				F	D	X	M	W	
ins i+4					F	D	X	M	W

	1	2	3	4	5	6	7	8	9
Taken branch ins	F	D	X	M	W				
ins i+1		F	*	*	*	*			
Branch target			F	D	X	M	W		
Branch target +1				F	D	X	M	W	
Branch target +2					F	D	X	M	W

▷ Predict Branch Taken :

- But branch target address in is not known by IF stage.
- Target is known at same time as branch outcome (ID stage)
- MIPS still incurs 1 cycle branch penalty.

▷ Delayed Branch :

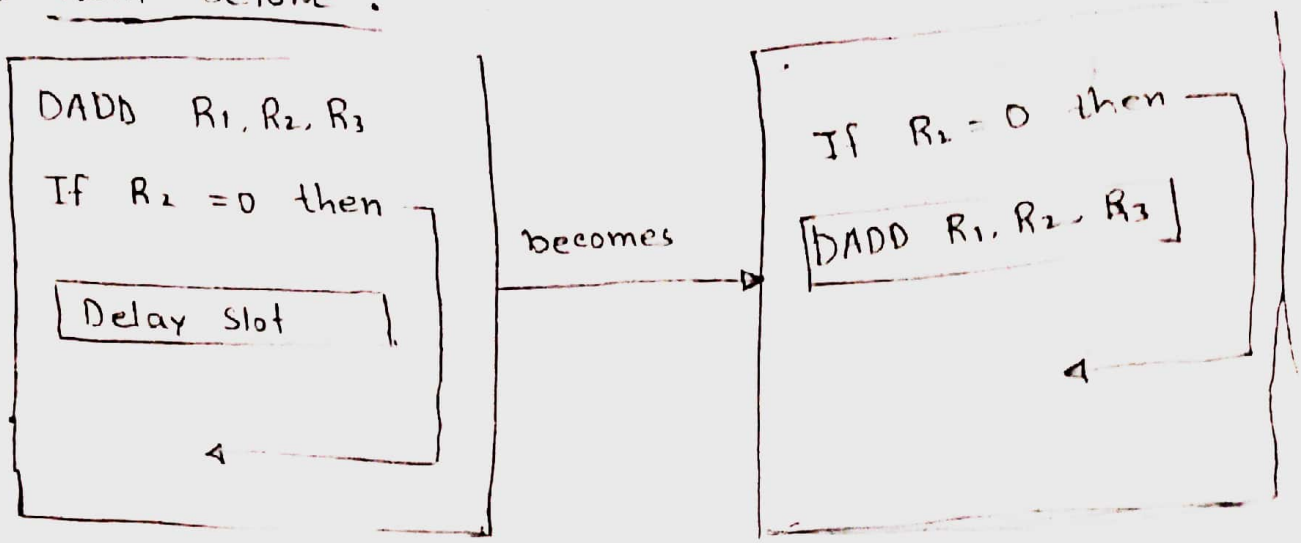
- Define branch to take place 'after' one instruction following the branch instruction.
- 1 slot delay allows proper decision and branch target address in 5 stage pipeline.
- Branch instruction এর পরে prediction ভুল হলে একটা cycle loss হয়, তাই এই cycle এ এমন একটা instruction use করা হয়, যার সাথে branch ins এর outcome এর কোনো dependency না থাকে।
- Branch ins এর পরের এই জায়গাটাকে delay slot বলে।
- Where to get instruction to fill branch delay slot?

	1	2	3	4	5	6	7	8	9
Untaken branch ins	F	D	X	M	W				
Branch delay ins (i+1)		F	D	X	M	W			
ins (i+2)			F	D	X	M	W		
ins (i+3)				F	D	X	M	W	
ins (i+4)					F	D	X	M	W

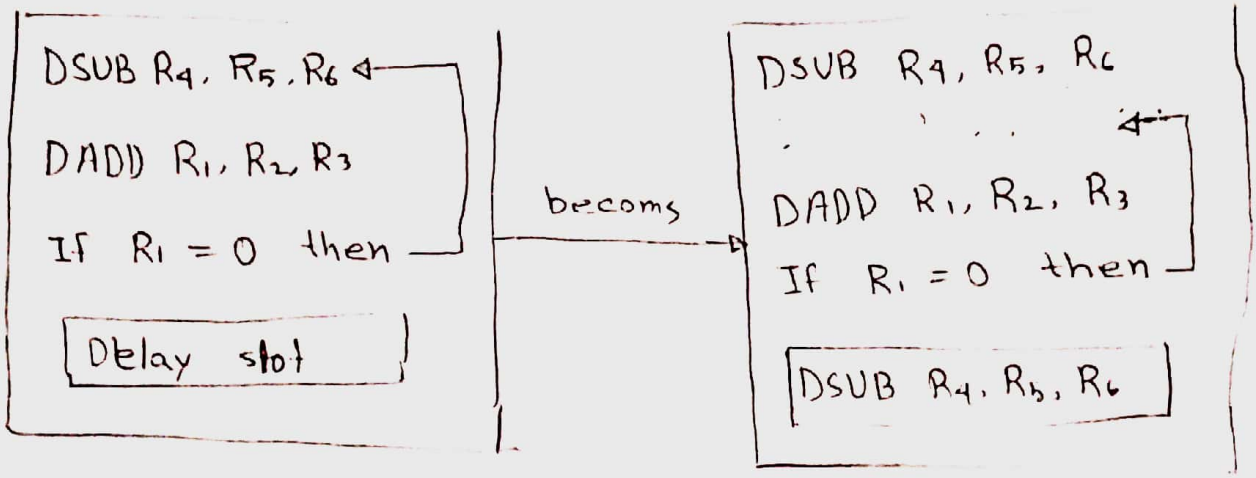
	1	2	3	4	5	6	7	8	9
Taken branch ins	F	D	X	M	W				
Branch delay in (i+1)		F	D	X	M	W			
Branch target			F	D	X	M	W		
Branch target +1				F	D	X	M	W	
Branch target +2					F	D	X	M	W

* Filling Branch Delay slot :

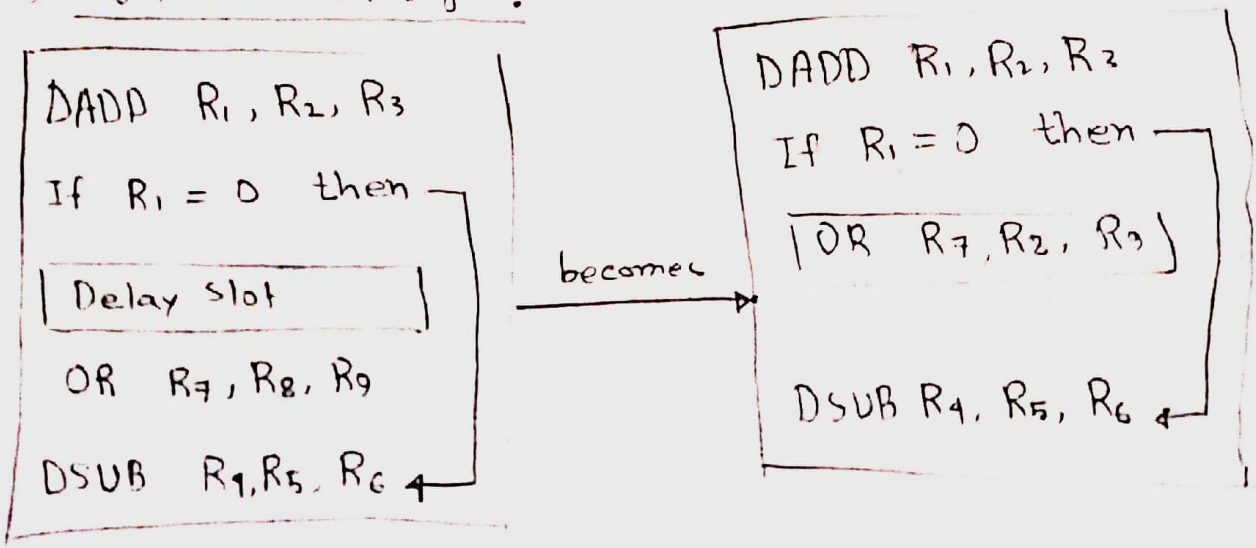
(a) From before :



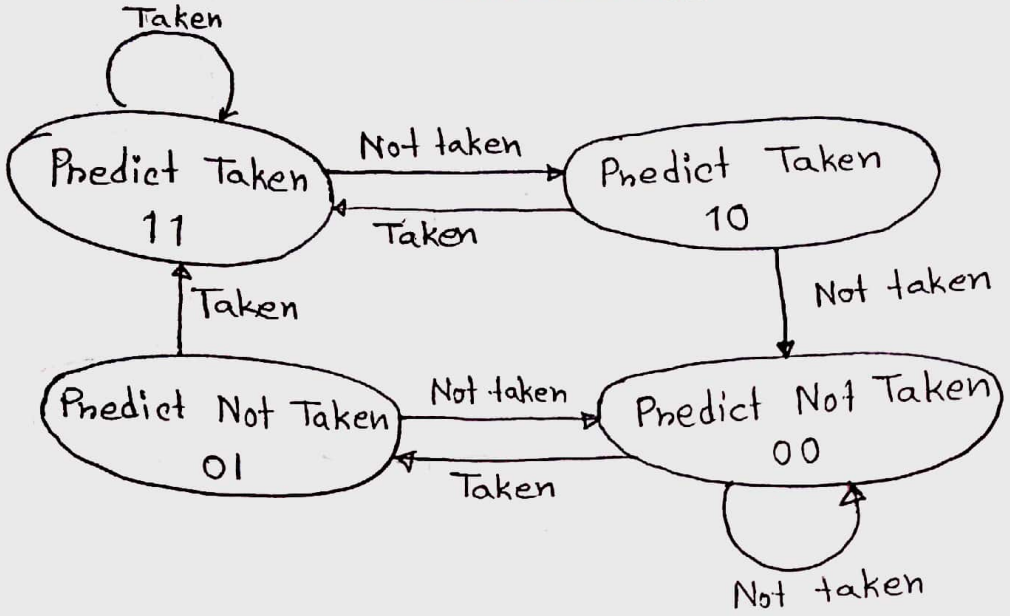
(b) From target :



(c) From fall through :



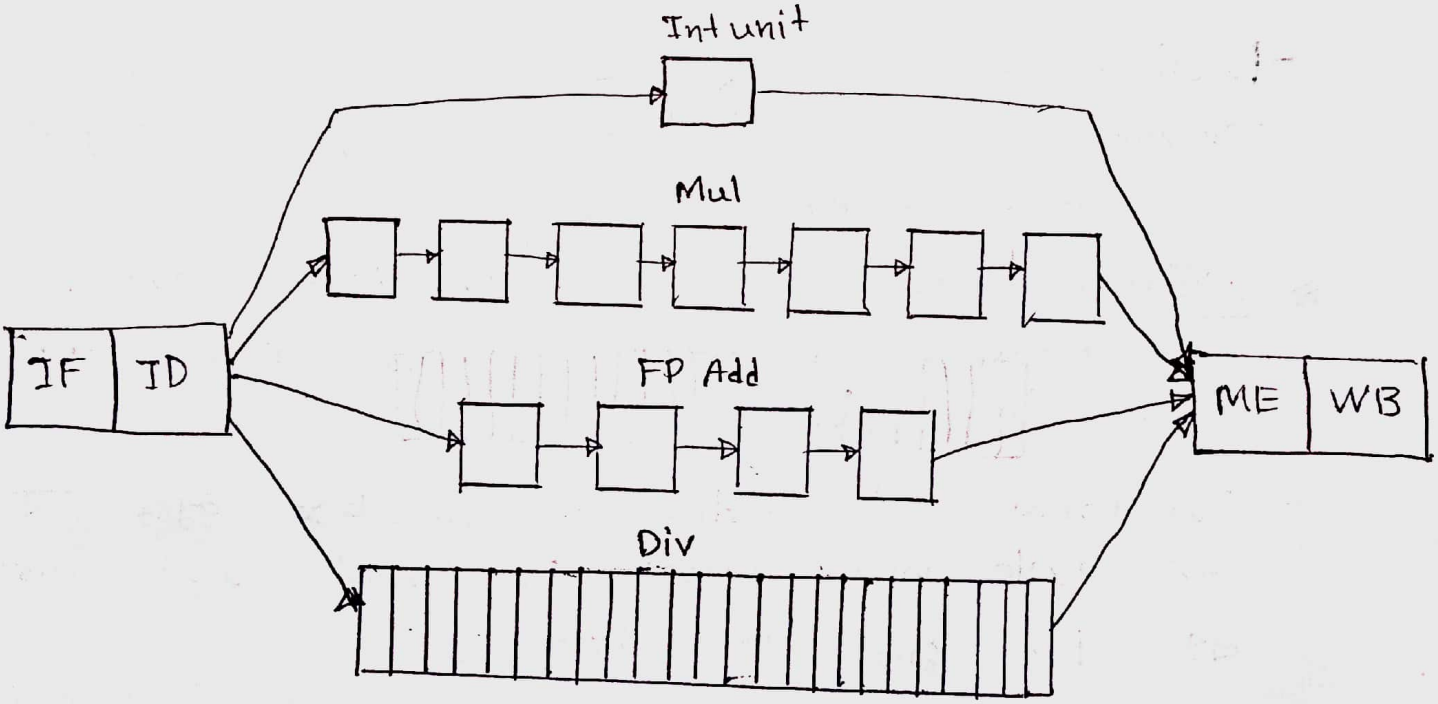
Answer to the q. no - 9



SI No	Last Outcome	BHT N/T	Prediction	Outcome	Mis-Prediction Y/N ?
1	N (initial	(00)/ 11	N	T	Yes
2	T	01 /(11)	T	T	No =
3	T	01 /(11)	T	N	Yes
4	N	(01)/10	N	N	No =
5	N	(00)/10	N	T	Yes
6	T	01/(10)	T	N	Yes
7	N	(01)/ 00	N	T	Yes
8	T	11 /(00)	N	T	Yes
9	T	11 /(01)	N	T	Yes
10	T	11 /(11)	T	N	Yes
11	N	(11)/ 10	T	T	No =
12	T	11 /(10)	T	N	Yes
13	N	(11)/00	T	T	No =
14	T	11 /(00)	N	T	Yes
15	T	11/ (01)	N	N	No =
16	N	(11)/ 00	T	T	No =

Multi-cycle Operations :

- Can EXE stage always complete the operation in 1 cycle?
- Some operations require more than 1 cycle to complete.
  - Floating Point / Integer Multiply
  - Floating Point / Integer Divide
  - Floating Point Add / Sub
- Dedicated hardware units are available on the processor for performing these operations.



Functional Unit	Latency	Initiation interval
1. Integer ALU	0	1
2. Data memory (FP/int Load)	1	1
3. FP add/sub	3	1
4. FP/int multiply	6	1
5. Fp/int divide	24	25

* Latency: The number of intervening cycles between an instruction that produces a result and an instruction that uses the result.

- Functional unit instruction এর Ex শুরু হওয়ার পরে আরেকটা instruction শুরু হওয়ার জন্য কতগুলো clock cycle wait করতে

* Initiation interval: The number of cycles that must elapse between issuing two operations of a given type.

- Functional unit instruction এর Ex শুরু হওয়ার কত cycle পরে... Same Functional unit instruction এর Ex শুরু হতে পারবে।

(1) Integer ALU:

add: IF ID EX ME WB
ins: IF ID EX ME WB

Latency - 0 ; wait করতে হয় না.
Initiation interval - 1 ; 1 টা clock cycle পরেই শুরু হতে পারে

② Data memory (FP / int Load) :

Lw : IF ID EX ME WB

Ins : — IF ID — EX ME WB .

Latency – 1 ; 1 টা clock cycle wait করতে হয়, কারণ Load ins এর value টা ME stage এর পরে পাওয়া যায়.

LW : IF ID EX ME WB

LW : IF ID EX ME WB

Initiation interval – 1 ; 1 টা clock cycle পরের পরের Load execute হতে পারে.

③ FP add / sub :

FP add : IF ID A_1 A_2 A_3 A_4 ME WB .

Ins : IF ID — — — Ex ME WB

Latency – 3 ; 3টা clock cycle wait করতে হয়, কারণ FP add ins এর A_4 এর পরে value টা ready হয়।

FP add : IF ID A_1 A_2 A_3 A_4 ME WB

FP add : IF ID A_1 A_2 A_3 A_4 ME WB

Initiation interval – 1 ; 1 টা clock cycle পরেই পরের FP add ins execute হতে পারে। Fp Add are divided into smaller box. they are internally pipelined. A_1 এর output A_2 use করে, A_2 এর output A_3 ... এজন্য 1st FP add যখন A_2 stage এ 2nd FPadd তখন A_1 stage এ কাজ করতে পারে।

④ FP/Int Multiply :

FP Mul : IF ID M_1 M_2 M_3 M_4 M_5 M_6 M_7 ME WB

Ins : IF ID — — — — — — EX ME WB

Latency – 6 ; ৬ টা clock cycle wait করতে হয়; কারণ Mul ins এর M_7 এর পর value টা ready হয়।

FP Mul : IF ID M_1 M_2 M_3 M_4 M_5 M_6 M_7 ME WB

FP Mul : IF ID M_1 M_2 M_3 M_4 M_5 M_6 M_7 ME WB

Initiation interval – 1 ; ১ টা clock cycle পরেই FP Mul instruction execute হতে পারে। কারণ, FP Mul এর ALU stage এর sub-stage গুলো আবার internally pipelined. তাই যখন 1st ins M_2 এ, 2nd ins তখন M_1 stage এ থাকতে পারে।

⑤ FP/Int Divide :

Div : IF ID D_1 D_2 D_3 .... D_{24} D_{25} ME WB

Ins : IF ID — — ..... — — Ex ME WB.

Latency – 24 ; পরের ins এর Execution শুরু করতে 24 টা clock cycle wait করতে হয়। কারণ Div ins এর D_{25} এর পর value টা ready হয়।

Div : IF ID D_1 D_2 D_3 -- D_{24} D_{25} ME WB

Div : IF ID D_1 D_2 .. D_{25}

Initiation interval – 25 ; 1st Div ins শুরু হওয়ার 25 টা clock cycle পরে 2nd Div ins শুরু হতে পারে। কারণ, Division stage টা internall pipelined না। পুরো 25 টা stage শেষ না হলে এই dW stage পরের ins use করতে পারে না।

Issues in longen latency Pipelines :

- Since divide unit is not-pipelined, structural hazard occured.

Div 1 : IF ID D_1 D_2 ---- D_{25} ME WB

Div 2 : IF ID ---------- Di D_2 ---- D_{25}

(S)→ To eliminate structural hazard more stall is required.

- Instruction have varying runtimes — more registers writes in the same cycle.
অনেক ক্ষেত্রে ~~WB~~ বিভিন্ন instruction এর WB same clock cycle এ থাকে।

	1	2	3	4	5	6	7	8	9	10	11
MULD F_0, F_4, F_6	F	D	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W
---		F	D	X	M	W					
---			F	D	X	M	W				
ADD.D F_2, F_4, F_6				F	D	A_1	A_2	A_3	A_4	M	W
---					F	D	X	M	W		
---						F	D	X	M	W	
L.D F_2, 0(R_2)							F	D	X	M	W

(S)→ Single ~~port~~ write port (serialize completion) vs multiple write ports.

- Resolve write port conflicts in ID stage and stall issue by 1
- stall either of the instruction (priority basis) at MEM/WB stage

▷ ● WAW hazards possible.

পরের ins যদি আগের ins এর আগেই same register এ write করতে চলে।

	1	2	3	4	5	6	7	8	9	10	11
MULD F_0, F_4, F_6	F	D	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W
...		F	D	X	M	W					
...			F	D	X	M	W				
ADD.D F_2, F_4, F_6				F	D	A_1	A_2	A_3	A_4	M	W
...					F	D	X	M	W		
L.D F_2, 0(R_2)						F	D	X	M	W	

(S)→ – WAW hazard at register F2

– Delay issue (ID → EX) of L.D. untill ADD.D enters MEM stage.

ADD.D				F	D	A_1	A_2	A_3	A_4	M	W	
...					F	D	X	M	W			
L.D						F	D	–	–	X	M	W

– Keep the result of ADD.D and give it to needed instruction.

– Hence only L.D will write on F2.

– If any instruction in A1,..A_4, M_1,..M_7 has the same destination as the instruction in ID and the time at which they reach WB is same, delay issue by 1 cycle and repeat.

▷ ⓪ RAW hazards possible.

· Stall for RAW hazards will be more.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D $F_4, 0(R_2)$	F	D	X	M	W												
MUL.D F_0, F_4, F_6		F	D	*	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W				
ADD D F_2, F_0, F_8			F	*	D	*	*	*	*	*	*	A_1	A_2	A_3	A_4	M	W
S.D $F_2, 0(R_2)$					F	*	*	*	*	*	*	D	X	*	*	*	M

Ⓢ → – If the source of an instruction in ID is F_i then F_i should be there as the name of destination of instruction in ID/A1, A1/A2, A2/A3 and ID/M1, M1/M2, ... M6/M7.

– RAW hazard আটকাতে গেলে stall দিতে হবে।

Answers to the q. no – 10

LOAD F_4, 8(R_2); $F_4 = X$

FMUL F_0, F_4, F_2 $F_0 = AX$ [$\because F_2 = A$]

FADD F_3, F_0, F_2; $F_3 = AX + A$

STOR F_3, 16(R_3);

So, AX+A will be stored in 16(R_3)

	1	2	3	4	5	6	7	8	9	10	11	13	13	14	15	16	17	18
LOAD	F	D	X	M	W													
FMUL		F	D	*	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W					
FADD			F	*	D	*	*	*	*	*	*	A_1	A_2	A_3	A_4	M	W	
STOR					F	*	*	*	*	*	*	D	*	*	*	X	M	W