Implementation of Pipeline : *An Introduction*

Appendix A - Computer Architecture : *A Quantitative Approach* - Hennessy & Patterson

MIPS R4000 Pipeline

- The **integer pipeline** has **eight (8) stages.**
- The **clock rate** can be made **higher** due to the **deeper pipeline.**
- The **time critical cache access** is divided into **stages.**
- The **instruction cache access** is divided in **two** and the **data cache access** is divided in **three (3)** stages.

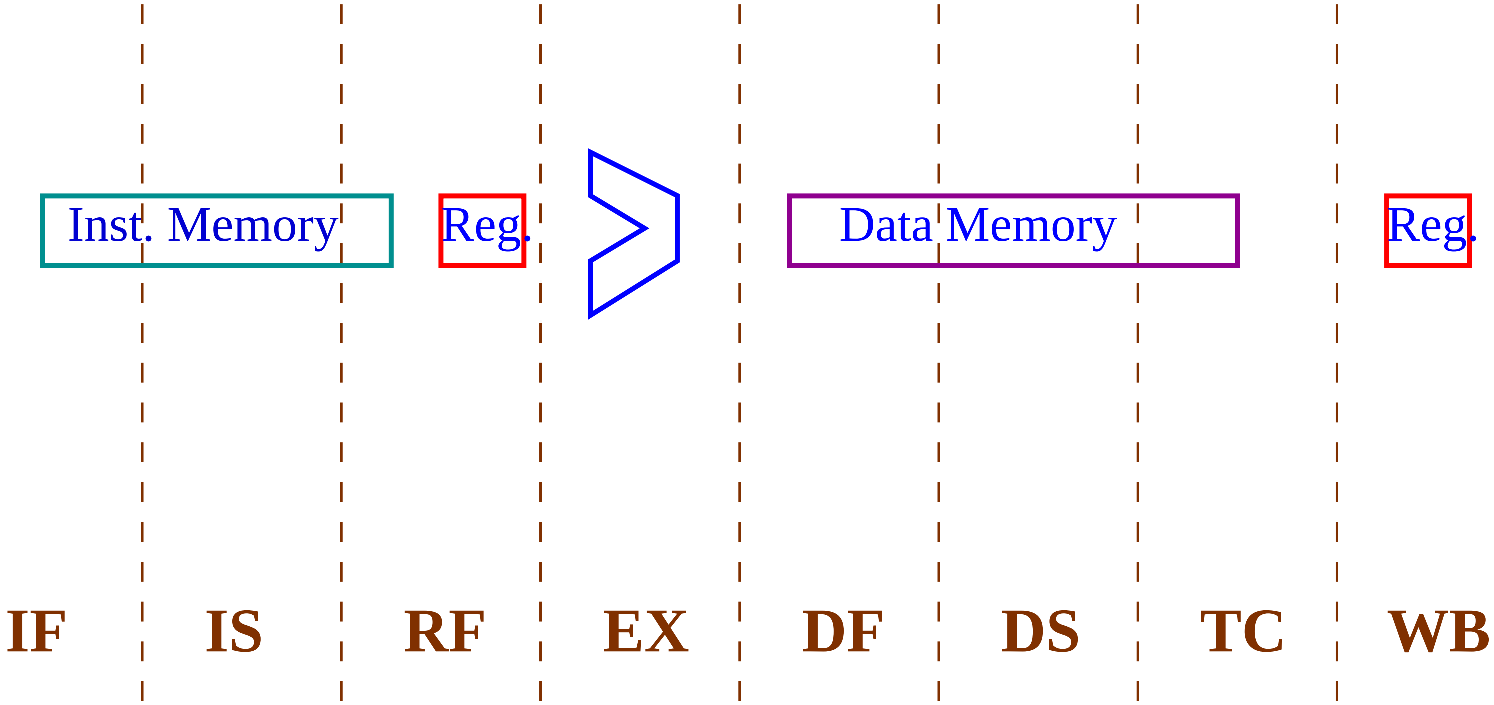

Figure 1: **R4000 8-stage Pipeline**

IF

- **Instruction cache access initiated for instruction fetch.**
- **The Next PC selection takes place.**

IS

- **Instruction cache access completed, the instruction is fetched.**
- **Still the hit detection is not complete!**

RF

- **Instruction cache hit complete.**
- **Instruction decode.**
- **Register fetch.**
- **Hazerd checking.**

EX

- **Memory address calculation or,**
- **ALU operations or,**
- **Branch address computation and branch condition checking.**

DF

- **First half of the data cache access for data fetch.**

DS

- **Completion of the data cache access.**
- **The test for cache hit is not complete[a].**

[a] The pipeline uses the data even before the cache hit detection is complete!

TC

- **Tag check for data cache hit.**

WB

- **Register write back for the load and ALU op.**

Effect of Deeper Pipeline

- Higher clock rate.
- Data forwarding is required after several cycles of load.
- Increase in load and branch delay.

Load Delay

Inst.	Clock								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
LD R1,0(R2)	**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**	
DADD R3,R1,R4		**IF**	**IS**	**RF**	**st**	**st**	**EX**	**DF**	**DS**
DSUB R5,R1,R6			**IF**	**IS**	**st**	**st**	**RF**	**EX**	**DF**
OR R7,R1,R8				**IF**	**st**	**st**	**IS**	**RF**	**EX**

OR does not require data forwarding in this case.

But ···

Load Delay

	Clock								
Inst.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
LD R1,0(R2)	**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**	
DADD R3,R5,R4		**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**
DSUB R7,R2,R6			**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**
OR R9,R1,R8				**IF**	**IS**	**RF**	**EX**	**DF**	**DS**
XOR R10,R1,R8					**IF**	**IS**	**RF**	**EX**	**DF**

OR and XOR require data forwarding in this case.

Branch in MIPS4000

- Branch target address and branch condition are computed at EX stage.
- The basic branch delay is of three (3) cycles.
- Single cycle branch delay slot.
- Branch strategy is predicted-not-taken.

Branch Delay: Not Taken

Inst.	Clock								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Branch (i)	**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**	
Delay		**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**
Inst - $i+1$			**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**
Inst - $i+2$				**IF**	**IS**	**RF**	**EX**	**DF**	**DS**

Branch Delay: Taken

Inst.	Clock 1	2	3	4	5	6	7	8	9
Branch (i)	**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**	
Delay		**IF**	**IS**	**RF**	**EX**	**DF**	**DS**	**TC**	**WB**
Inst - $i+1$			**IF**	**IS**			**no-op**		
Inst - $i+2$				**IF**		**no-op**			
Target					**IF**	**IS**	**RF**	**EX**	**DF**

Two (2) cycle branch stall for taken branch.

ALUOut Forwarding

- **The ALU output can be forwarded from four different pipeline registers: EX/DF, DF/DS, DS/TC and TC/WB.**

R4000 Floating-point Unit

- **There are three functional units: floating-point divider, floating-point multiplier, and floating-point adder (used also by multiply and divide units).**
- **A floating-point operation may take 2 to 112 cycles.**
- **The functional unit can be thought of having eight (8) different stages.**

R4000 Floating-point Unit

- **Each stage has a single copy.**
- **A floating-point instruction may use a stage zero (0) or more number of times in different order.**
- **The usage of differnet stages by different instructions gives rise to different latency and initiation times.**

8-stages of R4000 Floating-point Unit

A	FP adder	Mantissa/Significand add
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First multiplier-stage
N	FP multiplier	Second multiplier-stage
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP number

Latency and Initiation Intervals

Instr.	L	I	Stages Used
add/sub	4	3	U,S+A,A+R,R+S
multiply	8	4	U,E+M,M,M,M,N,N+A,R
divide	36	35	U,A,R,D^{27},D+A,D+R, D+A,D+R,A,R
sqrt	112	111	U,E,$(A+R)^{108}$,A,R

L - Latency and I - Initiation Intervals. The latency is 1-cycle less for store.

Stall with Multiply

If a floating-point multiply instruction is issued in the in the 0th cycle. In the next diagram we study how even an independent floating-point addition instruction interacts with it at different cycles.

Stall Add after Multiply

Clock									
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
U	**E+M**	**M**	**M**	**M**	**N**	**N+A**	**R**		
	U	**S+A**	**A+R**	**R+S**					
		U	**S+A**	**A+R**	**R+S**				
			U	**S+A**	**A+R**	**R+S**			
				U	**S+A**	**st**	**st**	**A+R**	**R+S**
					U	**st**	**S+A**	**A+R**	**R+S**
						U	**S+A**	**A+R**	**R+S**

Pipeline Scheduling

- **A simple pipeline fetchs an instruction and issues it if the source data does not depend on the output of some instruction already in the pipeline.**
- **Data forwarding can resolve some of the dependences; otherwise the hazard detection circuit of the pipeline stalls the instruction.**
- **Compiler can statically (at compile time) schedule instructions to reduce stalls.**

Dynamic Scheduling

- Some processors rearranges instructions to reduce stalls.
- So far the instruction issue was in-order. A stalled instruction stalls the following stream of instructions.

Modification of ID Stage

- In the original MIPS both structural and data hazards are tested in the ID phase.
- But to allow an instruction to begin execution as soon as its operands are available (even if a predecessor is stalled), the issue process is splitted in two phases.
- Checking the structural hazard and waiting for the absence of data hazard.
- Instructions are executed and completed out-of-order.

Modification of ID Stage

- **Issue: decode the instruction and check for structural hazard.**
- **Read Operands: wait until there is no data hazards, then read operands.**
- **The IF is before the issue and EX is after the read operand.**

Out-of-Order Completion and WAR Hazard

Consider the following sequence of code.

```
DIV.D   F0,F2,F4
ADD.D   F10,F0,F8
SUB.D   F8,F8,F14
```

The antidependence between SUB.D and ADD.D will give wrong result if SUB.D is completed before ADD.D.

A Scoreboard for Dynamic Scheduling

- **A scoreboard tries to maintain an execution of one instruction per clock cycle by executing an instruction as early possible.**
- **An instruction following a stalled instruction may be issued and executed provided it does not depend on any stalled or active instruction.**
- **A scoreboard takes care of proper issue, execution and hazard detection of instructions.**

A Scoreboard for MIPS

- **There is one integer unit, one floating point adder, one divider and two multipliers.**
- **In CDC 6600 there were sixteen (16) functional units - four (4) floating-point, seven (7) integer and five (5) memory references.**
- **The view of the processor is as follows.**

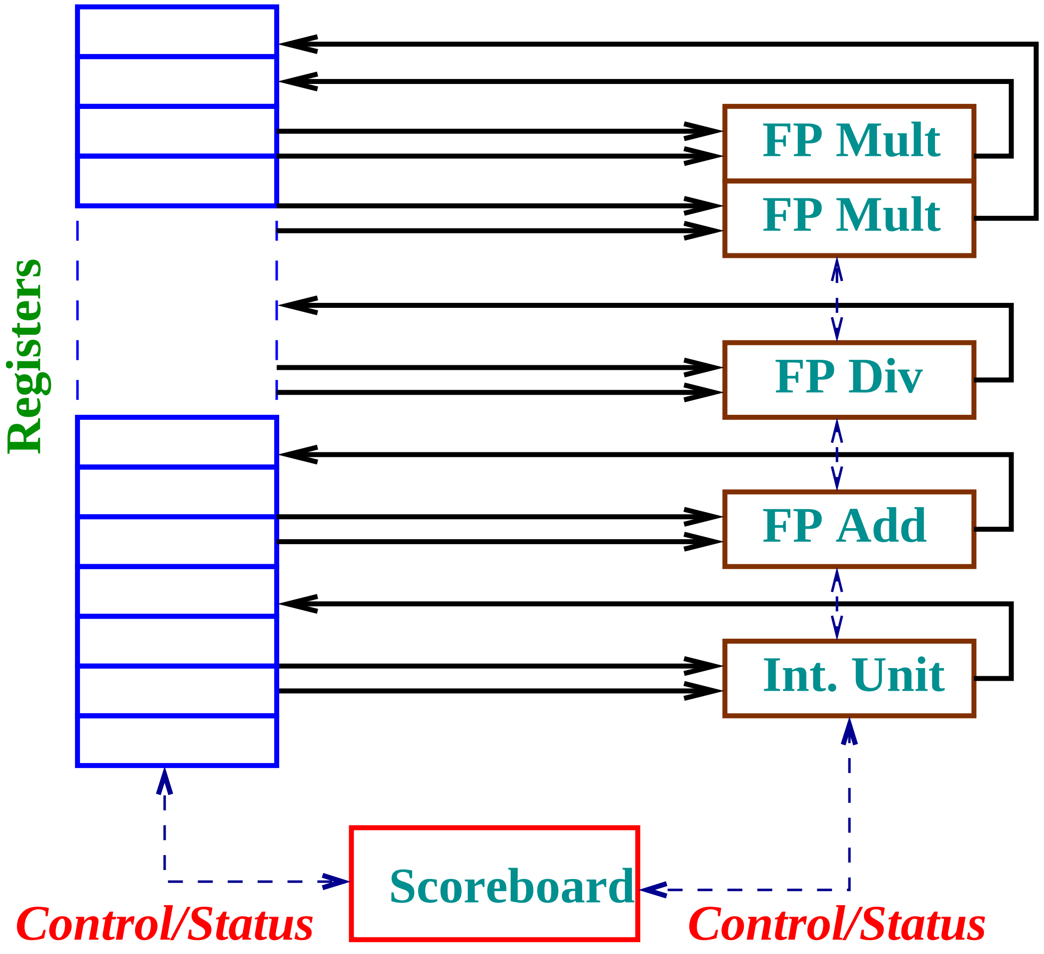

Figure 2: **Scoreboard**

MIPS Scoreboard: An Example

- Every instruction goes through the scoreboard where the data dependence is recorded.
- This phase corresponds to instruction issue. The scoreboard decides when an instruction can read operands and start execution.
- The scoreboard also controls write to a destination register.
- We consider the floating-point instructions only.

MIPS Scoreboard: An Example

- There are **four (4) steps** for execution.
- These steps **replace ID**, **EX**, and **WB** stages of **MIPS pipeline**.

MIPS Scoreboard - Issue

- **The instruction is issued if the corresponding functional unit is free and no active instruction in any other functional unit has the same destination register (no WAW hazard).**
- **The scoreboard updates its data structure.**
- **In presence of WAW hazard, the instruction issue is stalled as long as the hazard is not cleared (in-order issue).**

MIPS Scoreboard - Issue

- **Even if the instruction is stalled due to WAW, the instruction buffer (FIFO queue) between the fetch and the issue is filled. It stalls when the queue is full.**

MIPS Scoreboard - Read Operands

- The scoreboard keeps track of the availability of operands. A register value is available if it is not the destination of an active instruction.
- Once the required register values are available, the scoreboard signals the functional unit to fetch the operand and start execution. RAW is resolved dynamically and an instruction may start execution out-of-rder.
- This is the end of ID phase of pipeline.

MIPS Scoreboard - Execution

- **On receiving data, a functional unit starts execution and informs the scoreboard on completion.**
- **It may take multiple cycle for a floating-point operation.**

MIPS Scoreboard - Write Result

- **After receiving a completion signal from a execution unit the scoreboard checks for WAR hazard.**
- **It may stall the completing instruction.**

WAR Stall

```
DIV.D   F0,F2,F4
ADD.D   F10,F0,F8
SUB.D   F8,F8,F14
```

The ADD.D cannot be issued until DIV.D is complete. The instruction SUB.D can start its execution earlier but it will be stalled before the WB stage until ADD.D reads the operand from F8.

To Write or Not To Write

- A **register cannot be written when there is a preceding instruction that is suppose to read it.**
- **The destination and one of the operand registers are the same.**
- **If there is no WAR hazard, the register will be writen.**

  This is equivalent to the WB stage.

MIPS Scoreboard

- **Operands corresponding to an instruction are read when both are available in the registers. There is no data forwarding!**
- **The penalty is actualy not very large.**
- **Registers are normally written, unlike the original pipeline, immediately after the completion of the operation.**
- **This reduces the latency and gives the benefit of forwarding. But then there will be one cycle delay for read (after write).**

Scoreboard Data Structures

- **Instruction Status:** Indicates the state (there are four (4)) of the instruction.
- **Functional Unit Status:** Indicates the state of the functional units.
- **Register Result Status:** Indicates the functional unit that writes a register.

Functional Unit Status

- **Busy: indicates whether the functional unit is busy.**
- **Op: the operation to be performed by the unit.**
- F_i **(destination),** F_j **and** F_k **(sources): registers used.**
- Q_j **and** Q_k**: source registers modified by the functional units.**
- R_j **and** R_k**: boolean flags to indicate whether sources are ready and not yet read.**
  Set to 'no' after operand read.

A Sequence of Instructions

L.D	F6, 34(R2)
L.D	F2, 45(R3)
MUL.D	F0, F2, F4
SUB.D	F8, F6, F2
DIV.D	F10, F0, F6
ADD.D	F6, F8, F2

Instruction Status

Inst.	**Issue**	**Rd-Op.**	**Ex-Comp.**	**Wr-Reg.**
L.D F6, 34(R2)	**1**	**1**	**1**	**1**
L.D F2, 45(R3)	**1**	**1**	**1**	
MUL.D F0, F2, F4	**1**			
SUB.D F8, F6, F2	**1**			
DIV.D F10, F0, F6	**1**			
ADD.D F6, F8, F2				

The first load is complete and the 2nd load is going to write data.

Functional Unit Status

Nm	**Bsy**	**Op**	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
Int	**Y**	**LD**	F_2	R_3				**N**	
Mul1	**Y**	**Mul**	F_0	F_2	F_4	**Int**		**N**	**Y**
Mul2	**N**								
Add	**Y**	**Sub**	F_8	F_6	F_2		**Int**	**Y**	**N**
Div	**Y**	**Div**	F_{10}	F_0	F_6	**MUl1**		**N**	**Y**

Register Result Status

	F0	F2	F4	F6	F8	F10	···	F30
FU	Mul1	Int			Add	Div		

Status Just before MUL.D Writes Result

- **Add latency - 2 cycles, multiply latency - 10 cycles and Division latency - 40 cycles.**

Instruction Status

Inst.	Issue	Rd-Op.	Ex-Comp.	Wr-Reg.
L.D F6, 34(R2)	1	1	1	1
L.D F2, 45(R3)	1	1	1	1
MUL.D F0, F2, F4	1	1	1	
SUB.D F8, F6, F2	1	1	1	1
DIV.D F10, F0, F6	1			
ADD.D F6, F8, F2	1	1	1	

The 2nd ADD cannot write, there is a WAR hazard.

Instruction Status

Nm	Bsy	Op	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
Int	N								
Mul1	Y	Mul	F_0	F_2	F_4			N	N
Mul2	N								
Add	Y	Add	F_6	F_8	F_2			N	N
Div	Y	Div	F_{10}	F_0	F_6	MUl1		N	Y

Register Result Status

	F0	F2	F4	F6	F8	F10	···	F30
FU	Mul1			Add		Div		

Status Just before DIV.D Writes Result

Instruction Status

Inst.	Issue	Rd-Op.	Ex-Comp.	Wr-Reg.
L.D F6, 34(R2)	1	1	1	1
L.D F2, 45(R3)	1	1	1	1
MUL.D F0, F2, F4	1	1	1	1
SUB.D F8, F6, F2	1	1	1	1
DIV.D F10, F0, F6	1	1	1	
ADD.D F6, F8, F2	1	1	1	1

Instruction Status

Nm	Bsy	Op	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
Int	N								
Mul1	N								
Mul2	N								
Add	N								
Div	Y	Div	F_{10}	F_0	F_6			N	N

Register Result Status

	F0	F2	F4	F6	F8	F10	$\cdots$	F30
FU						**Div**		

Control and Bookkeeping

- **Wait Until: not Busy[FU] and not Result[D] - the functional unit is not busy and there is no previous instruction writing in the destinition register.**
- **Bookkeeping: Busy[FU] $\leftarrow$ Y, OP[FU] $\leftarrow$ op, F_i[FU] $\leftarrow$ D, F_j[FU] $\leftarrow$ S_1, F_k[FU] $\leftarrow$ S_2, Q_j[FU] $\leftarrow$ Result[S_1], Q_k[FU] $\leftarrow$ Result[S_2], R_j $\leftarrow$ not Q_j, R_k $\leftarrow$ not Q_k, Result[D] $\leftarrow$ FU**

Read Operands

- **Wait Until**: **R_j and R_k - both input operands are readable.**
- **Bookkeeping**: **$R_j \leftarrow$ No, $R_k \leftarrow$ No, $Q_j \leftarrow 0$, $Q_k \leftarrow 0$,**

Write Result

- **Wait Until:** $\forall$ f((F_j[f] $\neq$ F_i[FU] or R_j[f] == No) and (F_k[f] $\neq$ F_i[FU] or R_k[f] == No)) - the **source** of any instruction **already issued** is same as the **destination** of the **current instruction** and the source has not yet been read.

Write Result

- **Bookkeeping**: $\forall$ **f ((if Q$_j$[f] == FU, R$_j$[f] $\leftarrow$ Yes), (if Q$_k$[f] == FU, R$_k$[f] $\leftarrow$ Yes)), Result[F$_i$[FU]] $\leftarrow$ 0, Busy[FU] $\leftarrow$ No.**