Implementation of Pipeline : *An Introduction*

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

Exception Handling in a Pipeline

- **An exception changes the order of instruction execution in an unexpected way.**
- **Exceptions are more difficult to handle in a pipeline due to overlapping of instruction execution.**

Different Types of Exception

- IO device request (**interrupt**).
- Request for OS service (**system call/software interrupt**).
- Tracing instruction execution.
- Breakpoint.
- Integer arithmetic overflow.
- Floating-point arithmetic anomaly.

Different Types of Exception

- **Page fault.**
- **Misaligned memory access.**
- **Invalid instruction.**
- **Memory protection violation.**
- **Hardware error.**
- **Power failure.**

Exception Classification

- **Synchronous - asynchronous: System call - IO request.**
- **Requested - coerced: System call - IO request.**
- **User-muskable - user-nonmaskable: Integer overflow - memory protection violation.**
- **Within instruction - between instruction: Page fault - IO request.**
- **Resume execution - terminate execution: IO request - undefined instruction.**

Stopping and Restarting Execution

- **In a unpipelined implementation an execption like a page-fault is one of the most difficult to handle.**
- **It is generated in the middle of an instruction.**
- **The instruction is restarted after processing the fault.**
- **It is essential to remember the state of the CPU to correctly restart the instruction.**

Page-fault in MIPS Pipeline

- **A data page-fault may occur only at the MM cycle.**
- **There are two other instructions in execution in the pipe following the offending instruction.**
- **The pipeline states are to be saved properly for a correct restart of the instruction.**
- **Saving the PC of the offending instruction is necessary to restart it.**
- **The pipeline register contains the Next PC .**

Page-fault in MIPS Pipeline

- A trap/system call is forced in the IF/ID pipeline register.
- The offending instruction and the following instructions are changed to no-ops so that they cannot write anything.
- Trap transfers control to the exception-handling routine of the OS which saves the PC.
- Saving only one PC is not enough if there is delayed branch.

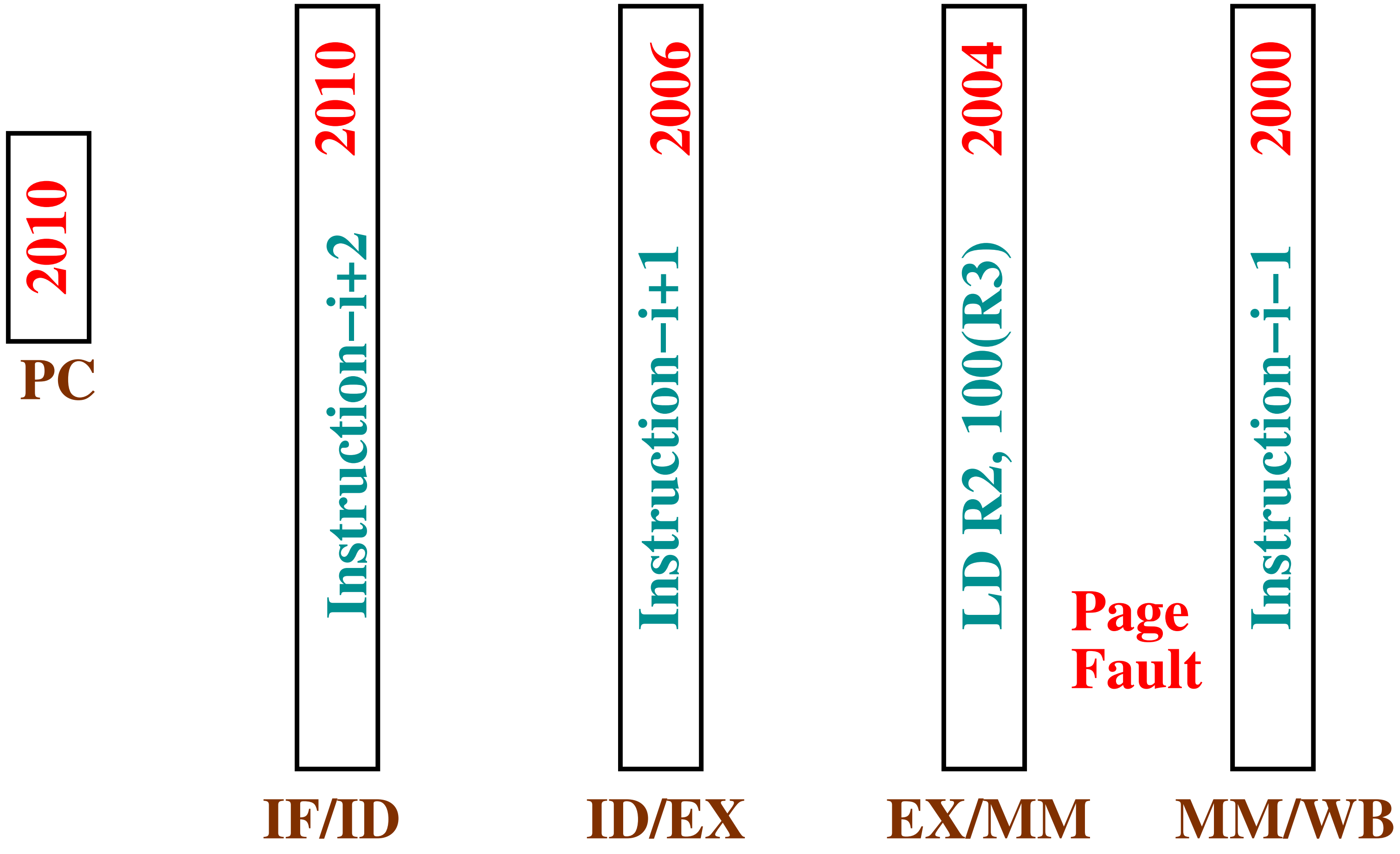

Figure 1: Page Fault

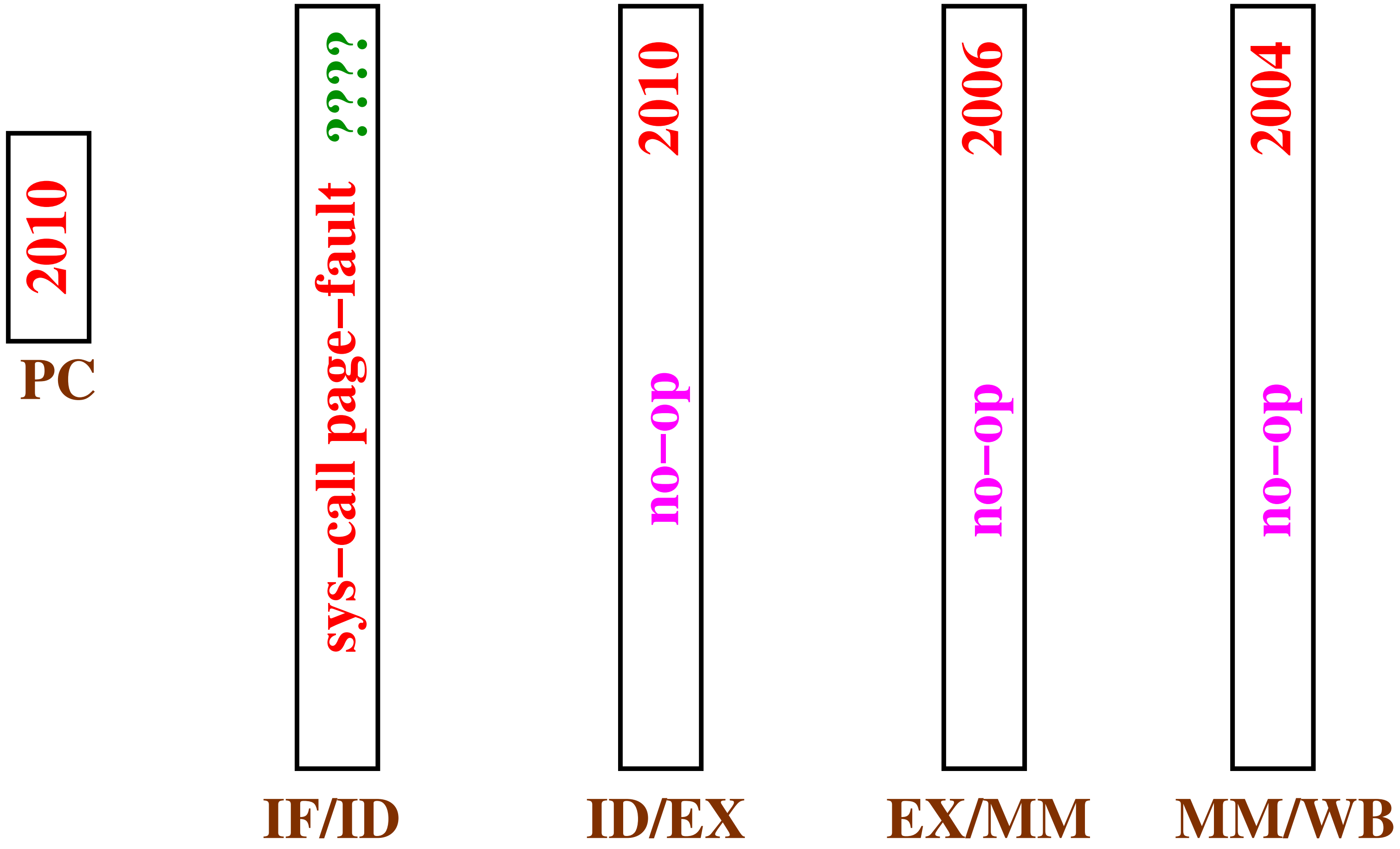

Figure 2: **Page Fault**

Page-fault with One Branch Delaye Slot

Branch	**BEQZ R2, target**
Delay Slot	**LD R3, offset(R1)**

- **We assume that the branch is taken and**
- **There is a page-fault at load-data access.**

Page-fault with Delay Slot

Inst.	Clock							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Branch (i)	**IF**	**ID**	**EX**	**MM**	**WB**			
Load (i+1)		**IF**	**ID**	**EX**	**PF**	**WB**		
Inst. (target)			**IF**	**ID**	**EX**	**MM**	**WB**	

It is not enough to save the address of load (in the delay slot), the address of the target is also to be saved.

Page-fault with Delay Slot

- Even if the page fault is at the fetch of the delay slot instruction, we have to allow the branch instruction to be completed.
- The address of the offending instruction and the address of the target instruction are to be saved.

Exception Handling

- **After handling the exception the processor status is reinstated and the instruction stream is restarted.**
- **A pipeline is said to have precise exceptions if it can be stopped after an execption and all instructions before the offending instruction can be completed and all instructions including and following the offending one can be restarted.**

Exception Handling

- Ideally faulting instruction should not have a side-effect.
- But in some cases e.g. on floating-point exception on some processor writes the result before exception handling.
- There should be hardware support to retrive the original operands in such case. The destinition may be one of the sources!

Exception Handling

- **A floating-point operation e.g. divide, runs for many cycles. Some other instruction may overwrite the source of the offending floating-point instruction** [**these operations often complete out of order**]**.**
- **Precise exceptions is costly to maintain.**

Exception Handling

- **Two modes of operations in high-performance CPU - precise exception mode and high-performance mode.**
- **The precise exception mode is naturally slower and is mainly used for debugging on these machines - it allows less overlap among floating-point instructions.**

Precise Exceptions

- Supporting precise exception is a necessity for many systems.
- It is essential to make exceptions precise in a system with demand paging.
- It is easier to support precise exception for integer pipeline.

Exceptions in MIPS Pipe Stages

- **IF: page fault (instruction fetch), misaligned memory access, memory protection violation.**
- **ID: Illegal opcode (opcodes not available in the user mode).**
- **EX: Arithmetic exceptions.**
- **MM: page fault (data fetch), misaligned data access, memory protection violation.**

Exceptions in MIPS Pipe Stages

- Two different exceptions may occur simultaneously (data page fault (PF) and arithmetic exception (AE)) in two different instructions in the pipeline.

Inst.	Clock					
	1	2	3	4	5	6
LD	IF	ID	EX	PF	WB	
DDIV		IF	ID	AE	MM	WB

Exceptions in MIPS Pipe Stages

- The first exception will be handled first and it will be restarted. The second exception will occur again and will be handled.
- But the exceptions may occur out of order.

Out of Order Exceptions in MIPS Pipe Stages

	Clock					
Inst.	**1**	**2**	**3**	**4**	**5**	**6**
LD	**IF**	**ID**				
DDIV		**PF (I)**				

Out of Order Exceptions in MIPS Pipe Stages

	Clock					
Inst.	**1**	**2**	**3**	**4**	**5**	**6**
LD	**IF**	**ID**	**EX**	**PF (D)**	**WB**	
DDIV		**PF (I)**	**ID**	**AE**	**MM**	**WB**

Exceptions in MIPS Pipe Stages

- The instruction page fault of DDIV cannot be handled before the data page fault of LD.
- The precise exceptions of pipeline requirs that all instructions before the fault can be completed. It is not possible in this case due to the data page fault of LD.

Exceptions in MIPS Pipe Stages

- An exception in a pipeline cannot be handled in the order of its occurrence.
- Exceptions corresponding to every instruction in the pipe are saved in a status vector.
- Once an exception is registered in the status vector, all data write by the instruction are turned off.
- The status vector is checked at the end of MM cycle and handled (naturaly in proper order).

Instruction Committed in MIPS

- **Every MIPS instruction writes at most one result.**
- **A MIPS instruction is committed (guaranteed to complete) at the end of the MM cycle.**
- **No instruction updates the state before this phase.**
- **Precise exception is easier to implement.**

Instruction Set Complication

- **Some CPU has instructions that changes state before it is guarranteed to complete.**
- **Also in a pipeline the $(i+1)$th instruction may change state before the ith instruction is guarranteed to complete.**

State Change Before Exception

Inst.	Clock					
	1	**2**	**3**	**4**	**5**	**6**
LD R2,-(R3)	**IF**	**ID**	**ALU1**	**MM (PF)**	**ALU2**	

The register R3 will be written before the page fault.

$(i+1)^{th}$ Instruction Changes State

Inst.	Clock					
	1	**2**	**3**	**4**	**5**	**6**
LD R2,0(R3)	**IF**	**ID**	**ALU1**	**MM (PF)**	**ALU2**	
DADD R4, -(R1)		**IF**	**ID**	**ALU1**	**MM**	

The register R1 will be written before the page fault.

Instruction Set Complication

- **If there is pre-decrement in register indirect addressing, then the register content will be changed before the memory access is successful.**
- **In such a case if there is an execption e.g. page-fault, the exception is imprecise (as the instruction is partially complete).**
- **Restarting from imprecise exception is difficult as the processor state is to be reinstated.**

Instruction Set Complication

- **In a unpipelined implementation we may defer the state change, but in a pipelined implementation it will be costly as there may be dependency on the updated state.**
- **As an example, a VAX instruction may bl4 autoinc/dec a register more than once [addl2 -(r2), -(r2)].**
- **The precise exception model demands that such processors should have the ability to back out to the state before the offending instruction.**

Instruction Set Complication

- **Similar problem arises if an instruction updates memory in the middle of execution.**
- **Example of such instructions are VAX or IBM 360 string copy instructions.**
- **Such instructions uses GPRs as working storage - saving and restoring them can restart the instruction.**
- **VAX can also restart an instruction from the middle.**

Instruction Set Complication

- **Many processors set condition codes as part of the ALU instructions.**
- **This saves extra register and also delinks condition evaluation and branch.**
- **But it makes it difficult to schedule other instructions between the condition evaluation instruction and the branch instruction as the other instructions may affect the condition codes.**

Instruction Set Complication

- In a complex ISA like VAX or IBM 360, some instruction may take only one clock cycle and some other may use tens or even hundreds of cycles.
- The number of memory access in such processors may also varry from zero to hundred (e.g. string copy instructions - rep movs %esi, %edi)

Instruction Set Complication

- **Making all instructions execute for same number of clock cycles for such an ISA is not possible due to large number of complex data hazards and corresponding data forward conditions.**
- **Instruction level pipeline would have been very difficult.**

Instruction Set Complication

- **These processors were implemented as microinstruction pipeline. An instruction is converted to a set of simple microinstruction and are executed in a pipeline.**
- **IA-86 after 1995 also uses the same technique.**
- **These microinstructions look very similar to MIPS instructions.**

Multicycle Operations in MIPS Pipeline

- **Floating-point operations normally takes many cycles to be completed.**
- **The modified pipeline has more number of EX cycles depending on the need of the operations.**
- **There are multiple functional units for the EX phase.**

Four (4) Functional Units in MIPS

- The main integer unit for load, store, ALU operation and branch.
- Floating-point and integer multiplier.
- Floating-point adder for addition, subtraction and conversion.
- Floating-point and integer division.

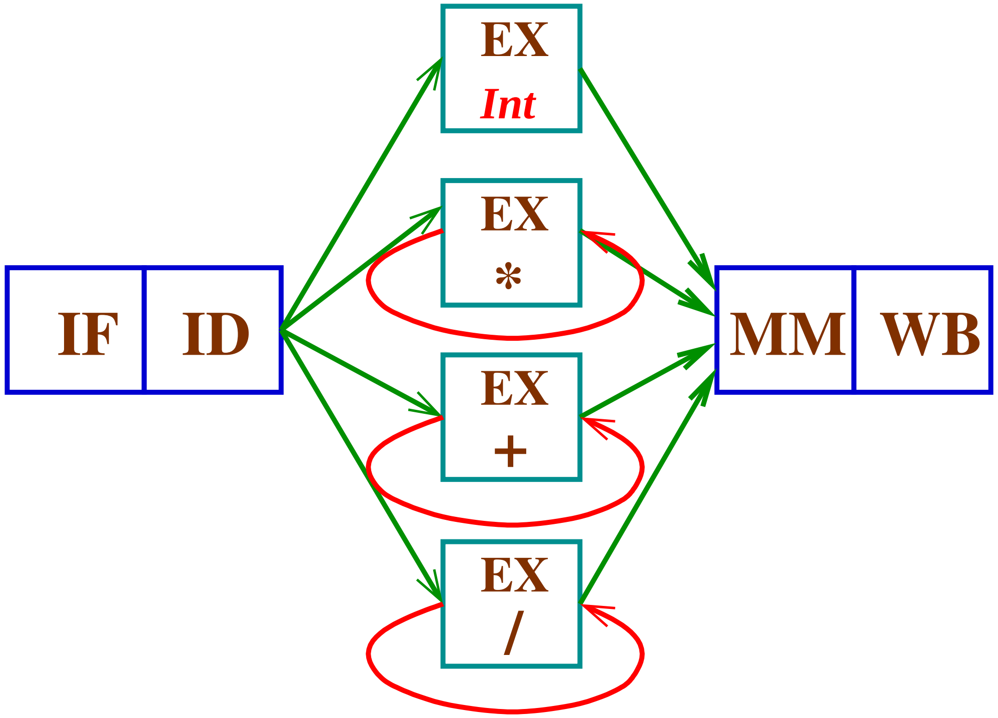

Figure 3: **Four Functional Units**

Four (4) Functional Units in MIPS

- **Execution of stages of these functional units are not pipelined i.e. no two instructions using the same execution unit can be scheduled parallaly.**
- **If an instruction is stalled before the EX stage, all following instructions are also stalled.**

Pipelined EX Stage

- In practice the intermediate result of a longer EX stage is not feed-back, but it takes more than one cycle delay.
- These EX stages can also be pipelined to execute more than one operations at a time.
- Two parameters, the latency and the initiation interval (repeat interval) of different operations are important to design such a pipeline.

Two Parameters

- **Latency: the number of clock cycles after which the result of an instruction can be used by the next instruction.**
- **Repeat Interval: The number of clock cycles that are essential to elapse between the scheduling of two instructions in the same functional unit.**

Two Parameters

EX Unit	Latency	Initiation Interval
Integer ALU	0	1
Load FP/Int	1	1
FP $+$	3	1
FP/Int $\times$	6	1
FP/Int $\div$	24	25

Pipeline Latency

- **Integer ALU: the latency is zero (0). An instruction produces the result at the end of EX stage and can be used in the EX stage of the next instruction.**
- **Load FP/Int: the latency is one (1). The data is available at the end of MM stage and can be used in the EX stage only after one (1) clock cycle delay.**

Pipeline Latency

- **For other cases the latency is $n_{EX} - 1$, where n_{EX} is the number of pipe stages in the EX unit (when the result is ready).**
- **There are four (4), seven (7) and twenty-five (25) stages in the FP adder, FP/Int multiplier and the FP/Int. divider respectively.**

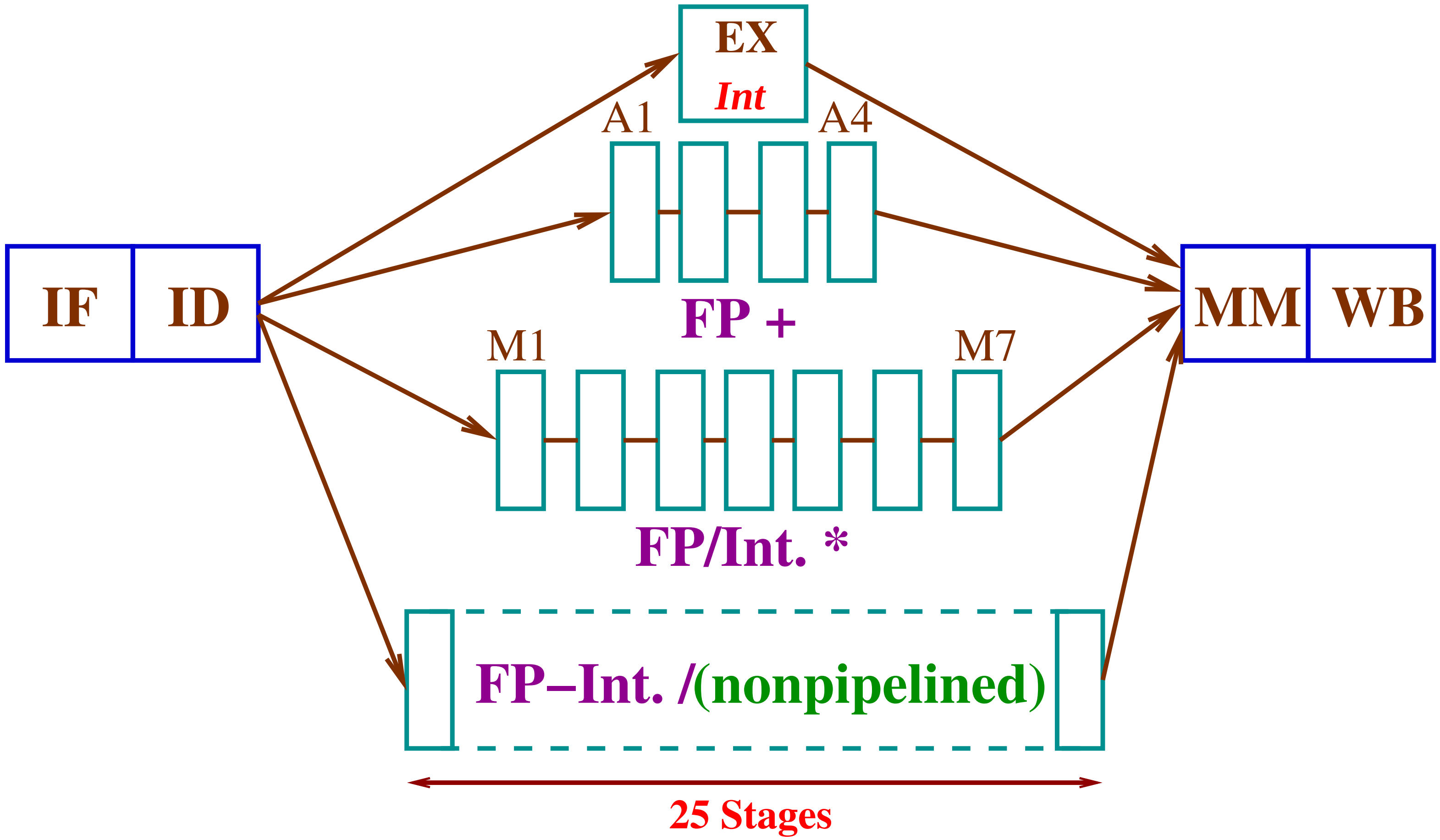

Figure 4: **Pipelined Functional Units**

Pipeline in EX

The shown pipeline has

- **four (4)** stages for **floating point addition**, and
- **seven (7)** stages for integer or floating-point **multiplication**.
- The floating-point/integer divider is **not pipelined**, but uses 25 clock cycles.
- These **additional pipeline stages** (**named differently**) are also **separated by pipeline registers**.

Pipeline in EX

- **four (4) floating point addition, and seven (7) integer or floating-point multiplication can be present in the EX stage.**
- **But only one division operation can be present in EX.**

Pipeline Diagram: An Example

MUL.D	IF	ID	M1	M2	M3	M4	M5	M6	M7	MM	WB
ADD.D		IF	ID	A1	A2	A3	A4	MM	WB		
LD.D			IF	ID	EX	MM	WB				
SD.D				IF	ID	EX	MM	WB			

Longer latency of floating-point operations increases the RAW hazards.

New Pipeline Registers

- **The new pipeline registers in FP adder are A1/A2, A2/A3, A3/A4. Similarly there are six (6) new pipeline registers in the multiplication pipeline.**
- **The original ID/EX register is extended to feed four (4) different EX units. We call them as ID/EX, ID/DIV, ID/M1 and ID/A1.**

Hazards in Longer Latency Pipeline

- Due to different execution time of different instructions, there may be more than one write in a cycle.
- Different instructions may reach WB stage out of order and there may be WAW hazard.
- Due to out of order completion there will be problem of exception handling.
- RAW hazards will be more frequent due to long latency operations.

RAW Hazards

	Clock Cycles								
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
L.D F4,0(R2)	**IF**	**ID**	**EX**	**MM**	**WB**				
MUL.D F0,F4,F6		**IF**	**ID**	**st**	**M1**	**M2**	**M3**	**M4**	**M5**
ADD.D F2,F0,F8			**IF**	**st**	**ID**	**st**	**st**	**st**	**st**
S.D F2,0(R2)					**IF**	**st**	**st**	**st**	**st**

RAW Hazards

	Clock Cycles								
Instruction	10	11	12	13	14	15	16	17	18
L.D F4, 0(R2)									
MUL.D F0, F4, F6	M6	M7	MM	WB					
ADD.D F2,F0,F8	st	st	A1	A2	A3	A4	MM	WB	
S.D F2,0(R2)	st	st	ID	EX	st	st	MM	WB	

Single Register Write Port

Structural Hazard

	Clock Cycles						
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**
MUL.D F0,F4,F6	**IF**	**ID**	**M1**	**M2**	**M3**	**M4**	**M5**
no-op		**IF**	**ID**	**EX**	**MM**	**WB**	
no-op			**IF**	**ID**	**EX**	**MM**	**WB**
ADD.D F2,F4,F6				**IF**	**ID**	**A1**	**A2**
no-op					**IF**	**ID**	**EX**
no-op						**IF**	**ID**
L.D F2,0(R2)							**IF**

Single Register Write Port

Structural Hazard

	Clock Cycles					
Instruction	**8**	**9**	**10**	**11**	**12**	**13**
MUL.D F0, F4, F6	**M6**	**M7**	**MM**	**WB**		
no-op						
no-op						
ADD.D F2,F4,F6	**A3**	**A4**	**MM**	**WB**		
no-op	**MM**	**WB**				
no-op	**EX**	**MM**	**WB**			
L.D F2,0(R2)	**ID**	**EX**	**MM**	**WB**		

Write Port of Register: Structural Hazard

- **Three instructions are trying to write in the floating-point register file in the same clock.**
- **If the register file has only one write port, the processor control must serialize the writes and so the completion of instructions.**
- **Increasing the number of write ports may not be a good solution as this type of situation is rare.**

Interlocking by Reservation Register

- Detect the possibility of use of the write port in the ID stage and stall the instruction.
- The use of write port by the previous instructions can be tracked using a shift register (reservation register).
- The current instruction can be stalled if there is a clash.

Reservation Register

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

MUL.D F0,F4,F6

-	-	-	-	-	-	-	-	1	-	-	-	-	-	-

no-op

no-op

ADD.D F2,F4,F6

-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
						1								

The add instruction can be stalled at the ID stage.

Interlocking before MM or WB

- An alternative solution is to stall the conflicting instruction before the MM or WB stage.
- Any one of the conflicting instructions may be stalled.
- One heuristic may be to give priority to the operation with longer latency, which might have caused a RAW hazard, to write.

WAW Hazard

	Clock Cycles							
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ADD.D F2,F4,F6	**IF**	**ID**	**A1**	**A2**	**A3**	**A4**	**MM**	**WB**
no-op		**IF**	**ID**	**EX**	**MM**	**WB**		
L.D F2,0(R2)			**IF**	**ID**	**EX**	**MM**	**WB**	

This code has the redundant ADD.D instruction. But still we don't want F2 to hold the value of ADD.D.

WAW Hazard

- **WAW hazard is rare and can be detected at the ID stage. There are two different ways to handle it.**
- **Issue of the L.D instruction is delayed until ADD.D enters the MM stage, or**
- **The WB of ADD.D is disabled.**

Hazard Detection at ID

- Check for structural hazard: wait for the availability of the functional unit (division) and wait for the availability of write ports of the register.
- Check for RAW hazard: wait while the source register of the instruction is listed as the destination register in some pipe stage.

Hazard Detection at ID

- **Check for WAW hazard: check whether any instruction at any stage of A1, A2, A3, A4, D, M1, $\cdots$ M7 has the same destination register of the ID stage instruction. In such a situation stall the instruction.**

Data Forwarding

- **Different pipeline registers are to be checked for floating-point data forwarding source. These registers are:**
  EX/MM, A4/MM, M7/MM, D/MM and MM/WB.

Data Forwarding: An Example

L.D F1, 100(R2)

ADD.D F2,F1,F3

- **There will be data forwarding from MM/WB to the A1 stage.**

Out-of-Order Completion

Consider the following sequence of code.

```
DIV.D  F0, F1, F2
ADD.D  F3, F3, F4
```

Out-of-Order Completion

.

- The DIV.D instruction will be scheduled in the floating-point division unit and the ADD.D instruction will be scheduled in the floating-point add/subtract unit.
- The ADD.D will be complete earlier.
- If there is an exception in DIV.D after the completion of ADD.D, it is imprecise.

Different Approaches

- The first approach taken by the earlier machines was to accept the imprecise exception.
- The exception may be handled without stopping the pipeline.
- This is not acceptable in a demand paging VM system (or with IEEE floating-point standard) where restarting an instruction may be necessary.
- Precise and imprecise exception modes of mordern processors.

Different Approaches

- The second approach is to buffer the results as long as the earlier instructions are not complete.
- If the differences of running times among the operations are too large, the number of buffer required will also become large.
- Forwarding circuit and control will be more complicated.

Viable Variation

- **A history file keeps the original values of the registers.**
- **The original values of the registeres modified by the instructions completed out-of-order can be restored from the history file.**
- **VAX used similar technique for autoincerment, autodecrement addressing.**

Viable Variation

- **Another approach is to use future file to store the results of instructions completed out-of-order.**
- **After completion of earlier instructions original registers are updated from the future files.**

Different Approaches

- **In the third approach the exception may be imprecise but the exception-handler gets sufficient information (e.g. PC values of all the instruction in the pipeline) so that after exception handling, it (the software) simulates all the incomplete instructions before the last completed one.**

3rd Approach: An Example

Consider the following sequence of instructions.

- **I_1: a long running instruction which will generate an exception.**
- **$I_2 \cdots I_{n-1}$: the sequence of incomplete instructions.**
- **I_n: the completed instruction.**

The I_{n+1} instruction is to be started after the exception handling

3rd Approach: An Example

- **The exception-handler get the PC's of $I_1, \cdots, I_{n-1}$ and I_{n+1}.**
- **The software is suppose to handle the exception, then finish the partially completed instructions and**
- **Finally return the control back to I_{n+1}.**
- **It is difficult to simulate the completion of execution of these instructions.**

Different Approaches

- **In the fourth scheme an instruction is issued if it is guaranteed that the previous instructions will be completed without exception. Otherwise the pipeline will be stalled.**
- **The floating-point functional unit should test early in the EX stage whether there can be an exception.**