4/11/20

PIPELINE AND ITS HAZARDS

-19PW13
Madhumitha.S

Pipeline Hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles. Any condition that causes a stall in pipeline can be called a Hazard.

There are primarily three types of Hazards:

i) Data Hazard
ii) Control Hazards / instruction Hazards
iii) Structural Hazards

1. Data Hazard / Data Dependency

Data Hazard comes as a result of overlapping the execution of data-dependent instructions

Consider instruction S such that-

S: ADD R1, R2, R3

Addresses read by S = {R2, R3}

Address written by S = {R1}

Consider instruction R such that

R: SUB R4, R1, R5.

Here, instruction S writes value into R1 while the value is needed by 2nd instruction R.

This issue can be solved by a simple hardware technique called OPERAND FORWARDING. (or by-passing or short circuiting)

The insight in forwarding is that result is not really needed by SUB until ADD executes completely. We use the interface registers present between stages to hold intermediate output so that dependent instruction can access new value from it.

INS/CYCLE	1	2	3	4
S	F	D	E	M
R		F	D	E

* HAZARDS classification

i) RAW : Read After Write
ii) WAR : Write After read
iii) WAW : Write After write

1. RAW (True dependency)

A read after write data Hazard refers to situation where an instruction refers to a result that has not yet been calculated.

This can occur because even though an instruction is executed after a previous instructions, the previous instruction is not completely processed through pipeline.

eg: i1 : $R_2 \leftarrow R_1 + R_3$

i2 : $R_4 \leftarrow R_2 + R_3$

2. WAR (Anti-dependency)

A write after read data dependency represents a problem with concurrent execution of instructions.

eg: $I_1 \leftarrow R_2$ $i_1 : R_2 \leftarrow R_1 + R_5$

$i_2 : R_5 \leftarrow R_1 + R_4$

3. WAW (output dependency)

A write after write data Hazard may occur in a concurrent execution environment It occurs when instruction i2 tries to write output before instruction i2 writes it

$i_1 : R_2 \leftarrow R_1 + R_3$

$i_2 : R_2 \leftarrow R_4 + R_5$

WAR and WAW occur during out-of-order execution of the instructions.

2. CONTROL HAZARDS

The presence of a conditional branch alters the sequential flow of instructions and it is not known where to continue until the branch outcome is resolved, They occur during instructions like BRANCH, CALL, JMP, etc.

Consider the following:

100 : I1
101 : I2 (JMP 250)
102 : I3
:
:
250 : BL1

Expected output : I1 → I2 → BL1

But target address of JMP is known after decode stage only.

INS/CYLE	1	2	3	4	5	6
I1	F	D	E	M	WB	
I2		F	D	E	M	WB
I3			F	D	E	M
BL1				F	D	E

output sequence - I1 → I2 → I3 → BL1

* SOLUTIONS

1. stay until branch is resolved (stall)
2. Delayed branch : Redefine runtime behaviour of branches to take only after partially fetched/executed instructions through pipeline
3. Branch prediction : predict outcome of branch and fetch there.

Static - continue fetching instructions following the branch and design pipeline so that followings instructions run safely and flush pipeline if branch is taken

dynamic - Track branch instruction behaviour using branch prediction buffer, use to predict direction of branch, continue fetching at the predicted location

Here, in case of stalls; Total no: of stalls introduced in pipeline due to branch instructions

= branch frequency * branch penalty,

3. STRUCTURAL HAZARDS

A structural dependency arises due to the resource conflict in pipeline. A resource conflict is a situation where more than one instruction tries to access same resource in same cycle. A resource can be a register, memory or ALU

Example of conflict:

INS/CYCLE	1	2	3	4	5
I1	F	D	E	M	WB
I2		F	D	E	M
I3			F	D	E
I4				F (Mem)	D

In above scenario, I2, I4 try to access memory which introduces resource conflict.

To avoid this issues we introduce stalls in the pipeline.

INS/CYCLE	1	2	3	4	5	6	7
I 1	F	D	E	M	WB		
I 2		F	D	E	M	WB	
I 3			F	D	E	M	WB
I 4				-	-	-	F

* SOLUTIONS :

To minimize structural dependency stall in pipeline, we add more hardware, also use hardware Mechanism called renaming.

RENAMING - We divide memory into 2 independent modules used to store the instruction and data separately called code memory (CM) & Data Memory (DM)

CM - contain instructions

DM - contain operands / data

INS/CYCLE	1	2	3	4	5	6	7
I 1	F(CM)	D	E	M(DM)	W		
I 2		CM	D	E	DM	W	
I 3			CM	D	E	DM	W
I 4				CM	D	E	DM
I 5					CM	D	E

Thus, This solves Resource conflicts.