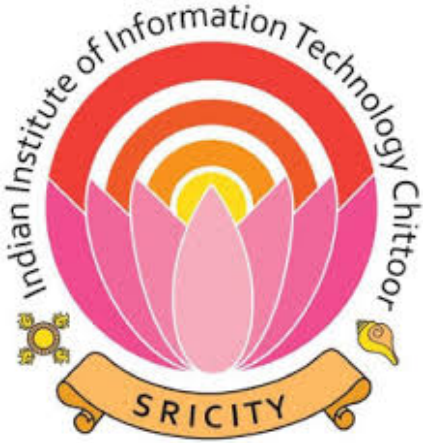
CO: Computer Organization

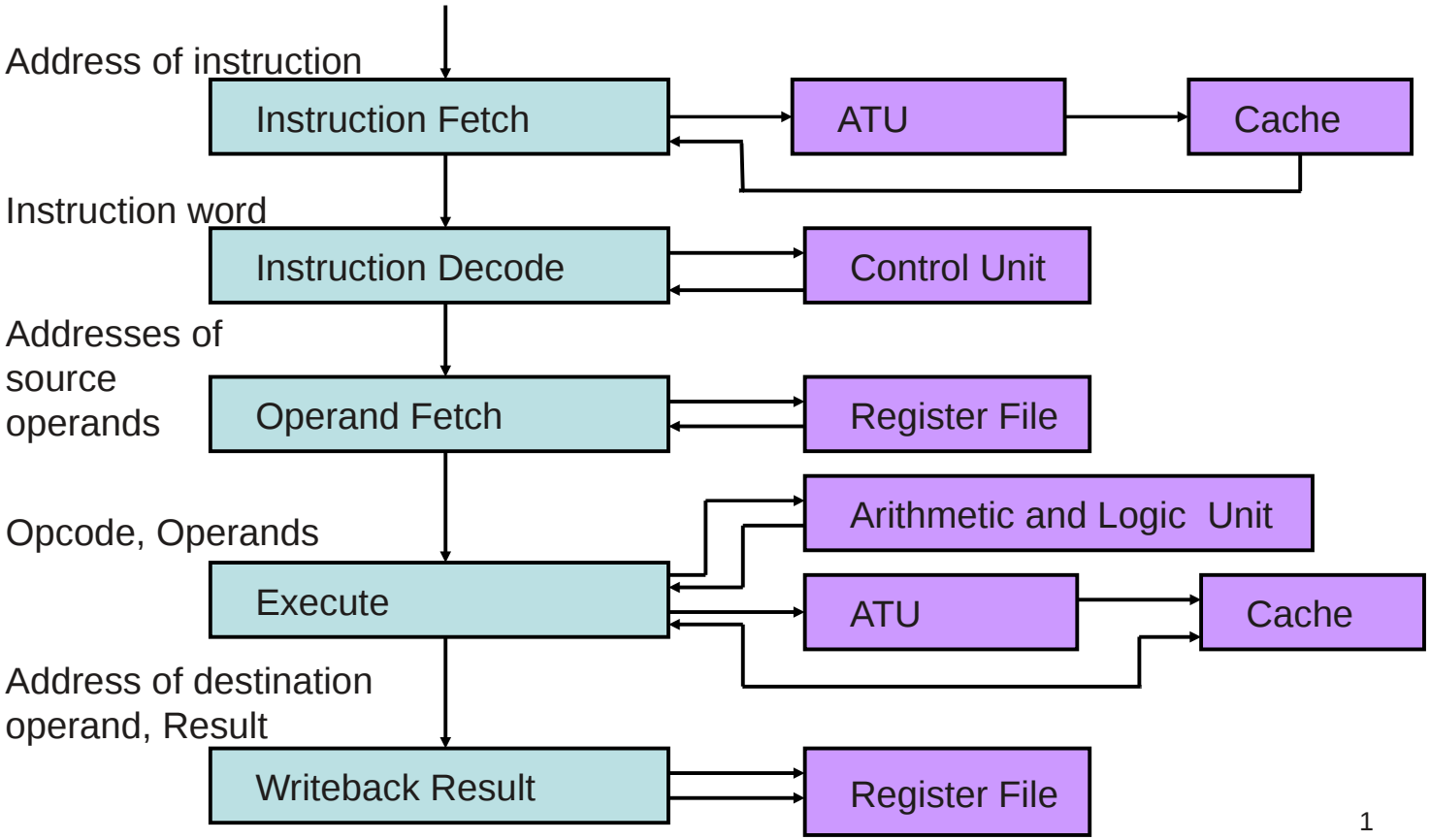
Pipelining

Indian Institute of Information Technology, Sri City

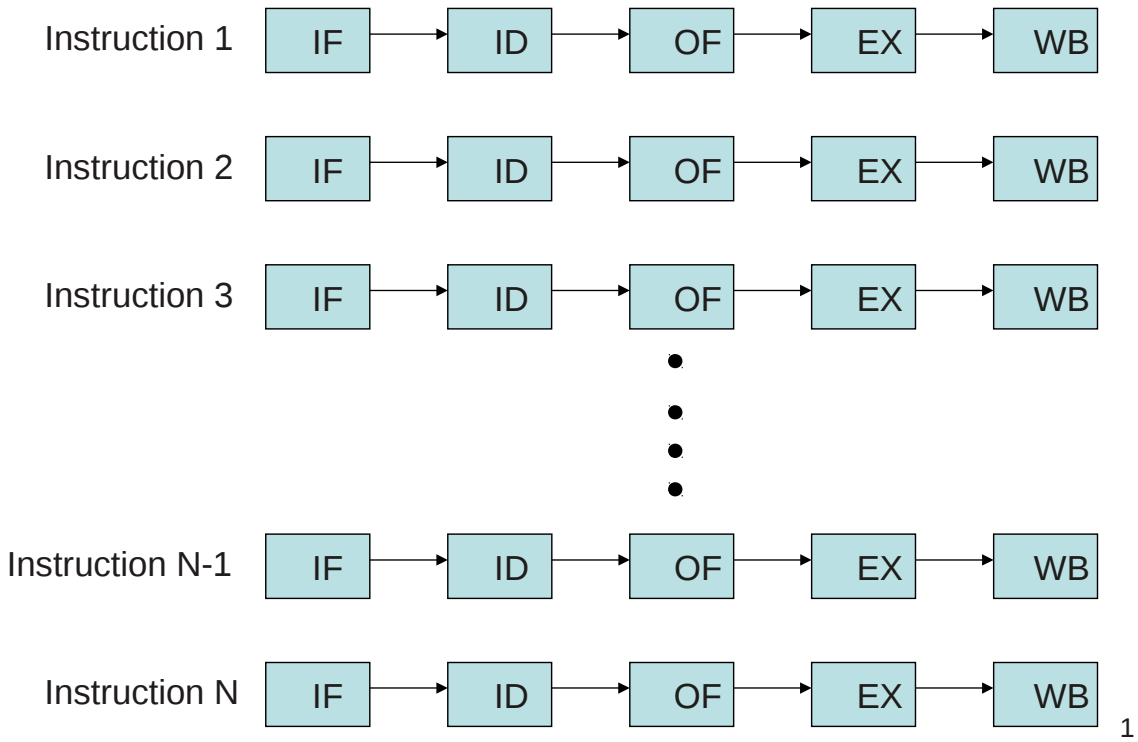
Jan - May - 2018



Instruction Cycle in RISC Processors



Non-pipelined Execution of Instructions



1

Pipelined Execution of Instructions

Pipelining is an implementation technique where multiple instructions are overlapped in execution.

Instruction	1	2	3	4	5	6	7	8	9	10
I1	IF	ID	OF	EX	WB					
I2		IF	ID	OF	EX	WB				
I3			IF	ID	OF	EX	WB			
I4				IF	ID	OF	EX	WB		
I5					IF	ID	OF	EX	WB	
I6						IF	ID	OF	EX	WB

- Non-pipelined takes: 5N cycles
- Pipeline takes : 5+N−1 cycles

1

ADD R1, R2, R3
SUB R4, R5, R6
ADD R7, R8, R9
MUL R11, R1, R10
MUL R12, R4, R1
MUL R7, R1, R4
MUL R4, R1, R11

Table 1: Sample Assembly Code

Assume that each stage takes 1 clock cycle.

Instruction	1	2	3	4	5	6	7	8	9	10	11	12
ADD R1, R2, R3	IF	ID	OF	EX	WB							
SUB R4, R5, R6		IF	ID	OF	EX	WB						
ADD R7, R8, R9			IF	ID	OF	EX	WB					
MUL R11, R1, R10				IF	ID	OF	EX	WB				
MUL R12, R4, R1					IF	ID	OF	EX	WB			
MUL R7, R1, R4						IF	ID	OF	EX	WB		
MUL R4, R1, R1							IF	ID	OF	EX	WB	

Table 2: Pipe-lined Execution

Assume that each stage takes 1 clock cycle.

ADD R1, R2, R3
SUB R4, R5, R6
MUL R12, R4, R1
ADD R7, R8, R9
MUL R11, R1, R10
MUL R7, R1, R4
MUL R4, R1, R11

Table 3: Sample Assembly Code

Stall: stop or delay in execution

ADD R1, R2, R3
SUB R4, R5, R6
MUL R12, R4, R1
ADD R7, R8, R9
MUL R11, R1, R10
MUL R7, R1, R4
MUL R4, R1, R11

Table 3: Sample Assembly Code

Stall: stop or delay in execution

Operand Forwarding Logic (or) Pipeline Interlock logic

Hazard

(It prevents the next instruction in the instruction stream from being executing during its designated clock cycle.)

- Data Hazards: They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
- Control Hazards: They arise from the pipelining of branches and other instructions that change the contents of PC.
- Structural Hazards: They arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.

Control Hazard

Instructions that causes for Control Hazards.

- JMP NEXT (unconditional jump)
- JCC NEXT (jump on conditional code Eg. : JZ, JNZ, JGE, JC, ... etc.)
- INT (Interrupt: It is signal to a processor that needs immediate attention.)

FOR(i=0;**i**<**N**; i++)
{**FOR-BODY**}

```
SUB R0,R0,R0
FOR-BEGIN: CMP R0, R1
JGE  END-FOR
FOR-BODY
INC R0
JMP FOR-BEGIN
END-FOR:
```

Predicting that branch will not takes place.

FOR(i=0;**i**<**N**; i++)
{**FOR-BODY**}

```
SUB R0,R0,R0
FOR-BEGIN: CMP R0, R1
JGE  END-FOR
FOR-BODY
INC R0
JMP FOR-BEGIN
END-FOR:
```

Predicting that branch will not takes place.

DO-WHILE

i=0;
DO {
DO-BODY
i++;
} WHILE(**i**<**N**)

SUB R0,R0,R0
DO: **DO-BODY**
INC R0
CMP R0,R1
JLT DO:

Predicting that branch will takes place.

DO-WHILE

i=0;
DO {
DO-BODY
i++;
} WHILE(**i**<**N**)

SUB R0,R0,R0
DO: **DO-BODY**
INC R0
CMP R0,R1
JLT DO:

Predicting that branch will takes place.

IF(**i**<**N**) THEN {**BODY**}

```
CMP R0, R1
JGE END-IF
BODY
END-IF:
```

IF(**i**<**N**) THEN {**IF-BODY**} ELSE {**ELSE-BODY** }

```
CMP R0, R1
JGE ELSE-BEGIN
IF-BODY
JMP END-IF
ELSE-BEGIN:  ELSE-BODY
END-IF:
```

Control Transfer Instructions (JMP, JCC)

Uses two types of branch predictions techniques:

- Static Branch Prediction: Loop Statements
- Dynamic Branch Prediction: IF-ELSE Statements
  (It uses history of behaviour for an instruction)

JCC	PV	FTA/BTA

JCC: Jump on Conditional Code
PV: Prediction is Valid/Not
FTA: Fall-Through Address
BTA: Branch Target Address

Control Transfer Instructions (JMP, JCC)

Uses two types of branch predictions techniques:

- Static Branch Prediction: Loop Statements
- Dynamic Branch Prediction: IF-ELSE Statements
  (It uses history of behaviour for an instruction)

JCC	PV	FTA/BTA

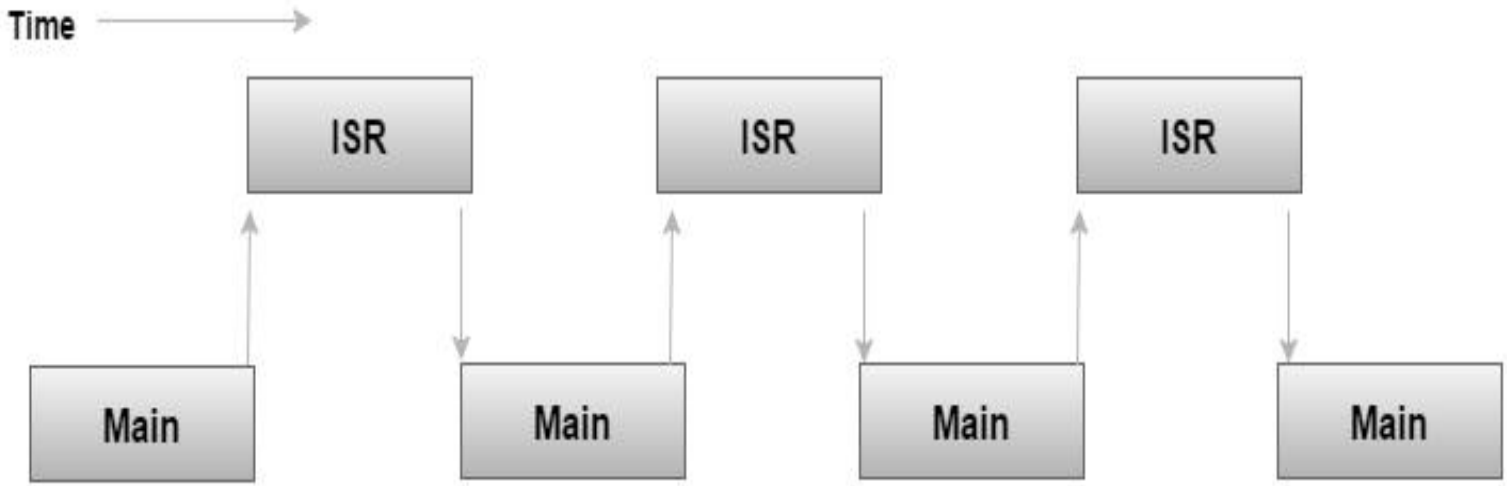
JCC: Jump on Conditional Code
PV: Prediction is Valid/Not
FTA: Fall-Through Address
BTA: Branch Target Address

```
FOR(i=0;i<N; i++)
{
FOR-BODY1
IF(i%3=0){ FOR-BODY2 }
}
SUB R0,R0,R0
FOR-BEGIN: CMP R0, R1
JGE  END-FOR
FOR-BODY1
MOD R2,R0,R3
JNZ X
FOR-BODY2
X: INC R0
JMP FOR-BEGIN
END-FOR:
```

Difference b/n Program Execution without/with Interrupts

Program Execution without Interrupts

Program Execution with Interrupts



ISR : Interrupt Service Routine

How an interrupt will be handled by a Processor?

The processor responds

- By suspending its current activities
- Saving its state using PCB (Process Control Block)
- Executing a function called an interrupt service routine (ISR)
- Resuming its normal activities

Classification of Interrupts:

- Hardware Interrupt
- Software interrupt

How an interrupt will be handled by a Processor?

The processor responds

- By suspending its current activities
- Saving its state using PCB (Process Control Block)
- Executing a function called an interrupt service routine (ISR)
- Resuming its normal activities

Classification of Interrupts:

- Hardware Interrupt
- Software interrupt

Interrupts due to Peripheral Devices

- Complete the execution of all instructions in the pipeline.
- Save the process state
- Interrupt Handling
  - Identify the device
  - Execute ISR
  - Return from ISR
- Resume the interrupted process

Process Switch

- Complete the execution of all instructions in the pipeline
- Save the process state
- Suspend the current process and move it to Ready-Queue
- Bring a new process from the Ready-Queue

Exception

(An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions)

Example: Arithmetic Exceptions - Divide-By-Zero, Over-Flow

- Complete the execution of all instructions up to the excepting instruction
- Save the contents of registers for debugging
- Terminate the process

Super Scalar Execution

No.of Cycles	Assembly Code
3	FADD F2,F0,F1
5	FMUL F5,F3,F4
3	FSUB F6,F2,F5
3	LOAD F7,X
5	FMUL F8,F6,F7
2	STORE Y,F8
1	ADD R4,R2,R3
3	LOAD R5,M
6	MUL R7,R5,R6
1	ADD R8,R4,R7
2	STORE N,R8

Register Renaming

Renaming the destination registers in the trailing instructions

Example:
ADD R1,R2,R3
SUB R3,R2,R1
MUL R1,R2,R3
DIV R2,R1,R3

Free Registers	Instruction
R4,R5,R6,R7	
R5,R6,R7	ADD **R4**,R2,R3
R6,R7	SUB **R5**,R2,**R4**
R7	MUL **R6**,R2,**R5**
	DIV **R7**,**R6**,**R5**

Table 4: Register Renaming

Register Renaming

Renaming the destination registers in the trailing instructions

Example:
ADD R1,R2,R3
SUB R3,R2,R1
MUL R1,R2,R3
DIV R2,R1,R3

Free Registers	Instruction
R4,R5,R6,R7	
R5,R6,R7	ADD **R4**,R2,R3
R6,R7	SUB **R5**,R2,**R4**
R7	MUL **R6**,R2,**R5**
	DIV **R7**,**R6**,**R5**

Table 4: Register Renaming

Register Renaming

- Destination Allocate
- Register Update
- Source Read

Destination Allocate

- Find a register in Register Rename File (RRF) with RRF.Busy=0
- Set RRF.Busy=1 and RRF.Valid=0
- Update the map table with rename register tag
- Set ARF.Busy=1

Register Renaming

Register Update

- Whenever a functional unit finishes its operation, update the RRF and set RRF.Valid=1
- Any dependent instruction can use the data from RRF
- When the instruction is completed, first update the ARF with data and later set ARF.Busy=RRF.Busy=0

Source Read

- If ARF.Busy=0, Fetch the operand from ARF.
- If ARF.Busy=1, the content is stale.
  - Access the map table to retrieve the rename reg. tag
  - If RRF.Valid=1, instruction execution is finished
  - If RRF.Valid=0, instruction is not executed. Forward the rename register tag from the map table to the reservation station