Computer Architecture ECS-355

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

(Week 15) Lecture 29-30

Learning objectives:

- Review of the last lecture
- RISC Instruction Pipeline
- Delayed Load and Delayed Branch
- Vector Processing
- Vector Operations

Resources: Beside these lecture handouts, this lesson will draw from the following

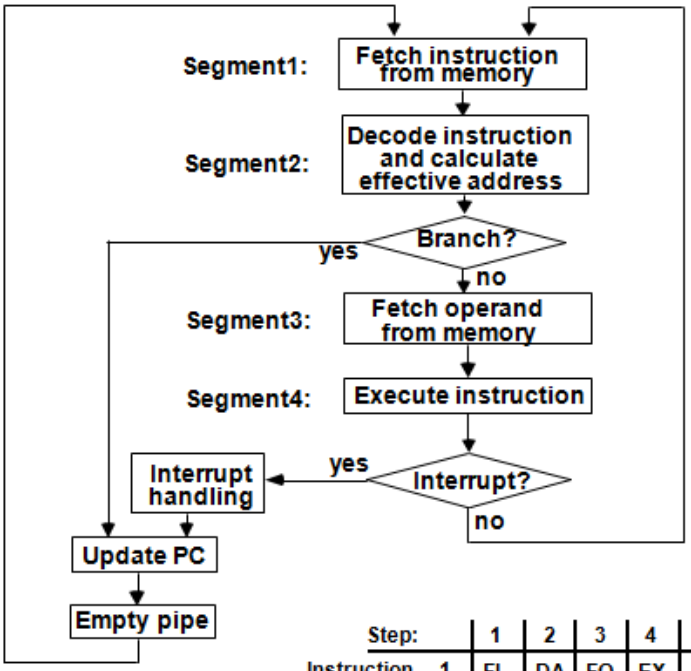
Text Book: Computer System Architecture by Morris Mano (3rd Edition) and **Reference book: Computer Architecture, by William Stallings (4th Edition)**.

Lecture:

Four-Segment Instruction Pipeline

Pipeline Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment. Assume further that most of the instructions place the result into a processor register so that the instruction execution and storing of the result can be combined into one segment. This reduces the instruction pipeline into four segments.

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**



	Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	-	-	FI	DA	FO	EX			
	5					-	-	-	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Three-Segment Instruction Pipeline

The control section fetches the instruction from program memory into an instruction register. The instruction is decoded at the same time that the registers needed for the execution of the instruction are selected. The processor unit consists of a number of registers and an arithmetic logic unit (ALU) that performs the necessary arithmetic, logic, and shift operations. A data memory is used to load or store the data from a selected register in the register file. The instruction cycle can be divided into three sub-operations and implemented in three segments:

I: Instruction fetch

A: ALU operation

E: Execute instruction

Delayed Load

Consider now the operation of the following four instructions:

1. LOAD: Rl ← M[address 1]
2. LOAD: R2← M[address 2]
3. ADD: R3← Rl + R2
4. STORE: M[address 3] ←R3

If the three-segment pipeline proceeds without interruptions, there will be a data conflict in instruction 3 because the operand in R2 is not yet available in the A segment. This can be seen from the timing of the pipeline shown in Fig. (a). The E segment in clock cycle 4 is in a process of placing the memory data into R2. The A segment in clock cycle 4 is using the data from R2, but the value in R2 will not be the correct value since it has not yet been transferred from memory. It is up to the compiler to make sure that the instruction following the load instruction uses the data fetched from memory. If the compiler cannot find a useful instruction to put after the load, it inserts a no-op (no-operation) instruction. This is a type of instruction that is fetched from

Clock cycles:	1	2	3	4	5	6
1. Load *R*1	I	A	E			
2. Load *R*2		I	A	E		
3. Add *R*1 + *R*2			I	A	E	
4. Store *R*3				I	A	E

(a) Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load *R*1	I	A	E				
2. Load *R*2		I	A	E			
3. No-operation			I	A	E		
4. Add *R*1 + *R*2				I	A	E	
5. Store *R*3					I	A	E

(b) Pipeline timing with delayed load

Delayed Branch

It was shown in Fig. that a branch instruction delays the pipeline operation until the instruction at the branch address is fetched. Several techniques for reducing branch penalties were discussed in the preceding section. The method used in most RISC processors is to rely on the compiler to redefine the branches so that they take effect at the proper time in the pipeline. This method is referred to as delayed branch.

The compiler for a processor that uses delayed branches is designed to analyze the instructions before and after the branch and rearrange the program sequence by inserting useful instructions in the delay steps.

An example of delayed branch is shown in Fig. The program for this example consists of five instructions:

1. Load from memory to R1

2. Increment R2
3. Add R3 to R4
4. Subtract R5 from R6
5. Branch to address X

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to *X*					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in *X*								I	A	E

(a) Using no-operation instructions

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to *X*			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instruction in *X*						I	A	E

(b) Rearranging the instructions

Vector Processing

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer days or even weeks to complete.

Computers with vector processing capabilities are in demand in specialized applications.

- Long-range weather forecasting
- Petroleum explorations
- Seismic data analysis
- Medical diagnosis

Vector Operations

Many scientific problems require arithmetic operations on large arrays of numbers. These numbers are usually formulated as vectors and matrices of floating-point numbers. A vector is an ordered set of a one-dimensional array of data items. A vector V of length n is represented as a row vector by V = [V1, V2, V3, · · · Vn].

To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

DO I= 1, 100

C (I) = B (I) +A (I)

This is a program for adding two vectors A and B of length 100 to produce a vector C. This is implemented in machine language by the following sequence of operations initialize I= 0

Read A (I)

Read B (I)

Store C (I) =A (I)+ B(I)

Increment I= I+ 1

If I <= 100 go to 20

Continue

This constitutes a program loop that reads a pair of operands from arrays A and B and performs a floating-point addition. The loop control variable is then updated and the steps repeat 100 times.

A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop. It allows operations to be specified with a single vector instruction of the form

C (1: 100) = A (1: 100) + B (1: 100)

The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.