CONTENTS

INTRODUCTION TO COMPUTER ARCHITECTURE

SYLLABUS

UNIT I

Introduction – Performance and cost Instructions set Architecture – Qualitative analysis of ISA – Addressing modes – Quantitative analysis of ISA - Reduced instruction set – Computer Architecture – Pipe line architecture – MIPS series – Motorola 88000 – SPARC Microchannel architecture – I/o subsystem architecture – architecture of I/o Bus – PCI bus – Microchannel architecture – Data Flow architecture – Parallel Architecture for control Driven Machine – Pipe line Hazards – The cross bar – switched systems – Multiprocessor with single and multi-stage-inter connections Network – Switch lattice Architecture.

UNIT II

Programming Languages – Assembly Language – Assembler – Subroutines – Input/Output Programming – Register transfer language Inter – Register transfer – Data transfer and manipulation – program control – Microprocessor organisation of 8086.

UNIT III

Microprogramming – Arithmetic Micro – Operations – Logic Micro operations – Shift Micro operations – Control functions – Control Memory – Address sequencing – Micro Program sequencer – Micro instruction formats – Advantages and Applications of Micro Programming.

UNIT IV

Arithmetic processor design – comparison and subtraction of unsigned binary numbers – Additions and subtraction, multiplication and division algorithms – Floating point arithmetic operations – Decimal Arithmetic operations.

UNIT V

Peripheral Devices – I/O interface – Asynchronous data transfer – Direct Memory Access – Priority interrupts – Input/Output processor – Memory hierarchy – Associative memory – Virtual memory – Cashe memory – Memory management hardware.

UNIT I

LESSON

1

INTRODUCTION TO COMPUTER ARCHITECTURE

CONTENTS

1.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain performance and cost instruction set architecture
- Discuss the qualitative and quantitative analysis of ISA
- Describe the addressing modes
- Identify reduced instruction set computer

1.1 INTRODUCTION

Computers vary greatly in terms of physical size, speed of operation, storage capacity, application, cost, ease of maintenance and various other parameters. The hardware of a computer consists of physical parts that are connected in some way so that the overall structure achieves the pre-assigned functions. Each hardware unit can be viewed at different levels of abstraction.

At the highest level of abstraction one can imagine the entire computer as a single piece of hardware. Going a level deeper, however, reveals that there are simpler hardware units that comprise the computer. As one goes deeper and deeper into the hardware abstraction level the view goes on expanding until the point where it cannot be expanded any further. Here is an illustration of the abstraction levels of typical personal computer hardware.

Continuing in this way, the hardware (at least the electronic part) breaks down to the following simple digital components.

- Registers
- Counters
- Adders
- Multiplexers
- Demultiplexers
- Coders
- Decoders
- I/O Controllers

The simplification can go on to still deeper levels. For instance, a transistors may be broken down into PNP or NPN semiconductors. These transistors and components at this level are usually packed into an integrated circuit implemented on a single silicon chip called microchip or just a chip.

Many technologies exist for manufacturing microchips. SSI (Small Scale Integration) packs 10 to 20 transistors in a single chip; MSI (Medium Scale Integration) packs as many as 100 transistors; LSI (Large Scale Integration) can have up to 1000 transistors while VLSI (Very Large Scale Integration) can pack more than a million transistors on a single IC chip. The complexity of integration is likely to go on increasing with time as a consequence smaller and more powerful computers will go on appearing.

Evidently, which components are used and how they are interconnected, dictates what the resulting computer will be good at doing. Thus, in a faster computer, there will be special components connected in a special way that enhances the speed of operation of the designed computer.

Different computer designs can have different components. Moreover, the same components can be interconnected in variety of ways. Each design will provide a different performance to the users. Exactly what components interconnected in what ways will produce what performance is the subject of Computer Architecture.

1.2 COMPUTER ARCHITECTURE

Computer Architecture deals with the issue of selection of hardware components and interconnecting them to create computers that achieve specified functional, performance and cost goals.

Some of the designs are identified by special names. Some of them are listed in the following one-liners.

1.2.1 BUS Architecture

Bus is another name for a communication line. The components in bus architecture are interconnected via a bus. Furthermore, the same bus may be used for the flow of both data and control or there may be separate buses for the purpose.

1.2.1 Instruction Set Architecture

The components are chosen and interconnected according to the set of instructions that the computer must be able to execute.

1.2.3 CISC (Performance and Cost Instruction Set Computer) Architecture

In this architecture the operands of instructions may be in memory (as well as in *registers*), in which many addressing modes are available, and in which some instructions may perform highly specialized tasks. As a rule, implementing CISC architecture in hardware requires the use of microcode. CISC instructions may require 10 or more clock periods to execute. CISC is an acronym for Complex Instruction Set Computer. The CISC machines are easy to program and make efficient use of memory. Since the earliest machines were programmed in assembly language and memory was slow and expensive, the CISC philosophy was commonly implemented in large computers such as PDP-11. Most common microprocessor designs such as the Intel 80x86 and Motorola 68K series have followed the CISC philosophy. The CISC instructions sets have the following main characteristics:

- A 2-operand format, where instructions have a source and a destination.
- Register to register, register to memory, and memory to register commands.
- Multiple addressing modes for memory, including specialised modes for indexing through arrays.
- Variable length instructions where the length often varies according to the addressing mode.
- Instructions requiring multiple clock cycles to execute.

A CISC machine, Intel 80486 uses 5-stage pipeline, which allows CPU to sustain close to one instruction executed per clock cycle. However, this architecture does not provide maximum potential performance improvement due to the following reasons:

- Sub-cycles may occur between initial fetch and instruction execution.
- An instruction may need result from previous instruction to execute.
- A branch instruction may occur.

1.2.4 RISC (Reduced Instruction Set Computer) Architecture

This is a class of computer architectures featuring a small number of primitive instructions, a constant hardware instruction length (in bits), and no memory accesses except by "load" and "store" instructions. The objective of a RISC is to reduce the number of clock periods per (integer) instruction to 1 for most executions, in order to maximize performance. The essential goal of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer. The major characteristics of a RISC processor are:

- Relatively few instructions.
- Relatively few addressing modes.
- Memory access limited to load and store instructions.
- All operations done within the registers of the CPU.
- Fixed length easily decoded instruction format.
- Single-cycle instruction execution.
- Hardwired rather than microprogrammed control.

A typical RISC processor architecture includes register-to-register operations, with only simple load and store operations for memory access. Thus the operand is code into a processor register with a load instruction. All computational tasks are performed among the data stored in processor registers and with the help of store instructions results are transferred to the memory. This architectural feature simplifies the instruction set and encourages the optimization of register manipulation. Almost all instructions have simple register addressing so only a few addressing modes are utilized. Other addressing modes may be included, such as immediate operands and relative mode. An advantage of RISC instruction format is that it is easy to decode.

An important feature of RISC processor is its ability to execute one instruction per clock cycle. This is done by a procedure referred to as pipelining. A load or store instruction may need two clock cycles because access to memory consumes more time than register operations. Other characteristics attributed to RISC architecture are:

- A relatively large number of register in the processor unit.
- Use of overlapped register windows to speed-up procedure call and return.
- Efficient instruction pipeline.
- Compiler support for efficient translation of high-level language programs into machine language programs.

1.2.5 Von Newmann Architecture

This is a stored-program architecture in which there is a single processor that operates sequentially on data that is stored in the same physical memory and in the same format as the instructions.

1.2.6 Parallel Processor Architecture

This architecture breaks down the data and instruction streams into parallel streams. This way performance can be enhanced manifolds. Following are different type of parallel architecture.

- SIMD (Single-Instruction Stream Multiple-Data Stream)
- MIMD (Multiple-Instruction Stream Multiple-Data Stream)

1.3 ADDRESSING MODES

As described the earlier, the operation to be performed is specified by the operation field of the instruction. The execution of the operation is performed on some data stored in computer registers or

memory words. The way the operands are chosen during program. Selection of operands during program execution depends on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referred. The purpose of using address mode techniques by the computer is to accommodate one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

Usage of addressing modes lends programming versatility to the user and helps to write program data mode affection in terms of instruction and implementation. The basic operation cycle of the computer must be understood to have a thorough knowledge of addressing modes. The instruction cycle is executive in the control unit of the computer and is divided into three principal phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Execute the instruction.

The program counter or PC register in the computer hold the instruction in the program stored memory. Each time when instruction is fetched from memory the PC is incremented, for it holds the address of the instruction to be executed next. Decoding involves determination of the operation to be performed, the addressing mode of the instruction, and the returns of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.

Figure 1.1 shows the distinct addressing mode field of in instruction format. The operation code (opcode) specifies the operation to be performed. The mode field issue to locate the operands needed for the operation. An address field in an instruction may or may not be present. If its there it may designate a memory address and if not, then a processor register. It is noticeable that each address field may be associated with its on specific addressing mode.

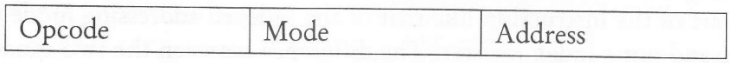

Figure 1.1: Instruction Format with Mode Field

Implied Mode: This mode specify the operands implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register references instructions that use an accumulator are implied mode instructions. Zero-address introductions are implied mode instructions.

Immediate Mode: The operand is specified in the instruction itself in this mode i.e. the immediate mode instruction has an operand field rather than an address field. The actual operand to be used in conjunction with the operation specified in the instruction is contained in the operand field.

Register Mode: In this mode the operands are in registers that reside within the CPU. The register required is chosen from a register field in the instruction.

Register Indirect Mode: In this mode the instruction specifies a register in the CPU that contains the address of the operand and not the operand itself. Usage of register indirect mode instruction necessitates the placing of memory address of the operand in the processor register with a previous instruction.

Autoincrement or Autodecrement Mode: After execution of every instruction from the data in memory it is necessary to increment or decrement the register. This is done by using the increment or decrement instruction. Given upon its sheer necessity some computers use special mode that increments or decrements the content of the registers automatically.

Direct Address Mode: In this mode the operand resides in memory and its address is given directly by the address field of the instruction such that the affective address is equal to the address part of the instruction.

Indirect Address Mode: Unlike direct address mode, in this mode give the address field give the address where the effective address is stored in memory. The instruction from memory is fetched through control to read is address part to access memory again to read the effective address. A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following equation:

Effective address = Address part of instruction + Context of CPU register

The CPU Register used in the computation may be the program counter, Index Register or a base Register.

Relative Address Mode: This mode is applied often with branch type instruction where the branch address position is relative to the address of the instruction word itself. As such in the mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address whose position in memory is relative to the address of the next instruction. Since the relative address can be specified with the smaller number of bits then those required the design the entire memory address, it results in a shorter address field in the instruction format.

Indexed Addressing Mode: In this mode the effective address is obtained by adding the content of an index register to the address part of the instruction. The index register is a special CPU register that contains an index value and can be incremented after its value is used to access the memory.

Base Register Addressing Mode: In this mode the affective address is obtained by adding the content of a base register to the part of the instruction like that of the indexed addressing mode though the register here is a base register and not a index register. The difference between the two modes is based on their usage rather than their computation. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction, and gives a displacement relative to this base address. The base register addressing mode is handy for relocation of programs in memory to another as required in multi programming systems. The address values of instruction must reflect this change of position with a base register, the displacement values of instructions do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

Numerical Example

Figure 1.2 explicitly shows the effect of addressing mode on the instruction through a numerical example. The two-word instruction at address 200 and 201 is a "load to AC" instruction when an address field equal to 500. The first word of the instruction specifies the operation code and mode, and the second word specifies the address part. PC has the value 200 for fetching this instruction. The content of processor register R1 is 400, and t content of an index register XR is 100. AC receives the operand after the instruction is executed. The figure lists a few pertinent addresses and shows the memory content at each of these addresses.

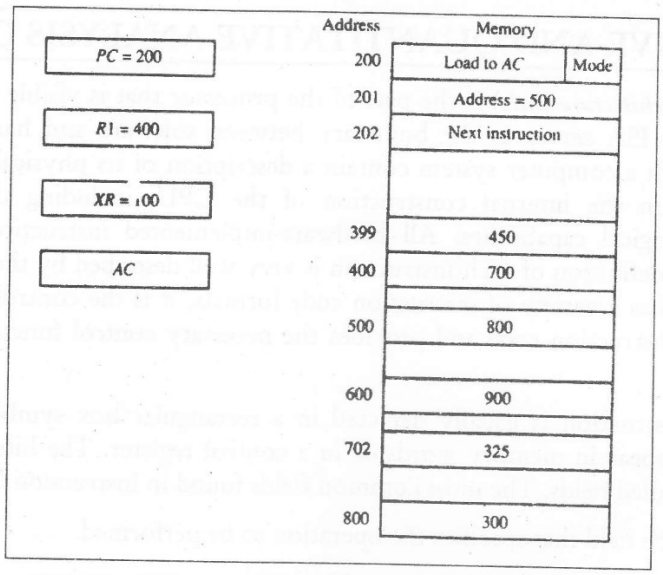

Figure 1.2: Numerical Example for Addressing Modes

Any one of the number of modes can be specific by the mode field of the instruction. For each possible mode we calculate the effective address and the operand that must be loaded into AC. In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800. In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC. (The effective address in this case is 201.) In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300. In the relative mode the effective address is 500 + 202 = 702 and the operand is 325. (Note that the value in PC after the fetch phase and during the execute phase is 202.) In the index mode the effective address is XR + 500 = 100 + 500 = 600 and the operand is 900. In the register mode the operand is in Rl and 400 is loaded into AC. (There is no effective address in this case.) In the register indirect mode the effective address is 400, equal to the content of R1 and the

perand loaded into AC is 700. The autoincrement mode is the same as the register indirect mode except that R1 is incremented to 401 after the execution of the instruction. The autodecrement mode decrements R1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450. Table 6.4 the values of the effective address and the operand loaded into AC for the nine addressing modes.

Table 1.1: Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

1.4 QUALITATIVE AND QUANTITATIVE ANALYSIS OF ISA

The ***Instruction Set Architecture*** (ISA) is the part of the processor that is visible to the programmer or compiler writer. The ISA serves as the boundary between software and hardware. The reference manuals provided with a computer system contain a description of its physical and logical structure. These manuals explain the internal construction of the CPU, including the processor registers available and their logical capabilities. All hardware-implemented instructions, their binary code format, and a precise definition of each instruction is very well described by these manuals. Given the fact that a computer has a variety of instruction code formats, it is the control unit within the CPU that interprets each instruction code and provides the necessary control functions needed to process the instruction.

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

At times other special fields are also employed sometimes employed as for example a field that gives the number of shifts in a shift-type instruction.

The operation code field of an instruction is referred to a group of bits that define various processor operations, such as add, subtract, complement, and shift. A variety of alternatives for choosing the operands from the given address is specified by the bits that define the mode field of the instruction code. Further ahead is the description about the address field on the instruction format and affect at including of multiple address field in instruction in the digital circuits In this section we are concerned with the address field of an instruction format and consider the effect of including multiple address fields in an instruction.

Execution of operations done by some data stored in memory or processor registers through specification received by computer instructions. A memory address specifies operands residing in memory whereas those residing in processor registers are specified by a register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers R0 through R15 will have a register address field of four bits. The binary number 0101, for example, will designate register R5.

Instructions in computers can be of different lengths containing varying number of addresses. The internal organization of the registers of a computer determines the number of address fields in the instruction formats. Most computers fall into one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

An example of an accumulator-type organization is the basic computer. All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address

field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as:

ADD X

where X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

An example of a general register type of organization was presented in Figure 3.2. The instruction format in this type of computer needs three register address fields. Thus, the instruction for an arithmetic addition may be written in an assembly language as:

ADD Rl, R2, R3

to denote the operation Rl ← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same and one of the source registers. Thus, the instruction:

ADD Rl, R2

would denote the operational R1 ← R1 + R2. Only register addresses for R1 and R2 need to be specified in this instruction.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction:

MOV R1, R2

denotes the transfer R1 ← R2 (or R2 ← R1, depending on the particular computer). Thus, transfer-type instructions need two address fields to specify the source and the destination.

Usually only two or three address fields are employed general register-type computers in their instruction format. Each address field may specify a processor register or a memory word. An instruction symbolized by:

ADD R1, X

would specify the operation R1 ←R1+M[X]. It has two address fields, one for register R1 and the other for the memory address X.

Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction:

PUSH X

will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction:

ADD

in a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. Since all operands are implied to be in the stack , the need for specifying operands with the address field does not arise.

Most of the computers comprise one of the three types of organizations that have just been described. Some computers combine features from more than one organizational structure. For instance, the Intel 8080 microprocessor has seven CPU registers, one of which is an accumulator register. Consequently, the processor exhibits some of the characteristics of a general register type and some of the characteristics of an accumulator type. All arithmetic and logic instructions, as well as the load and store instructions, use the accumulator register, so these instructions have only one address field. On the other hand, instructions that transfer data among the seven processor registers have a format that contains two register address fields. Moreover, the Intel 8080 processor has a stack pointer and instructions to push and pop from a memory stack. The processor, however, does not have the zero-address-type instructions which are characteristic of a stack-organized CPU.

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

X = (A + B)*(C + D)

using zero, one, two, or three address instructions. We will use the symbols ADD, SUB, MUL and DIV for the four arithmetic operations; MOV for the transfer-type operation; and LOAD and STORE for transfers to and from memory and AC register. We will assume that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X.

1.4.1 Three-Address Instructions

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) * (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

ADD R1,A,B	R1←M[A]+M[B]
ADD R2,C,D	R2←M[C]+M[D]
MUL X,R1,R2	M[X] ←R1*R2

It is assumed that the computer has two processor registers, Rl and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses. An example of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

1.4.2 Two-Address Instructions

Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or memory word. The program to evaluate X = (A + B)* (C + D) is as follows:

MOV R1, A	R1←M[A]
ADD R1, B	R1←RI+M[B]

MOV R2, C R2←M[C]

ADD R2,D R2 ← R2+M[D]

MUL R1,R2 R1 ← R1*RP

MOV X, R1 M[X] ← R1

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation transferred.

1.4.3 One-Address Instructions

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate X=(A+B)*(C+D) is:

LOAD A AC ← M[A]

ADD B AC ← AC+M[B]

STORE T M[T] ← AC

LOAD C AC ← M[C]

ADD D,A AC ← AC+M[D]

MUL T AC ← AC * M[T]

STORE X M[X] ← AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

1.4.4 Zero-Address Instructions

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

PUSH A TOS←A

PUSH B TOS←B

ADD T TOS← (A+B)

PUSH C TOS←C

PUSH D TOS←D

ADD T TOS← (C+D)

MDL T TOS← (C+D)*(A+B)

POP X M[X] ←TOS

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

1.4.5 RISC Instructions

The instruction set of a typical RISC processor is restricted/to the use of load and store instructions when communicating between memory and CPU. All other instructions are executed within the registers of the CPU without referring to memory. A program for a RISC-type CPU consists of LOAD and STORE instructions that have one memory and one. Register address, and computational-type instructions that have three addresses with all three specifying processor registers. The following is a program to evaluate X = (A + B)*(C + D).

LOAD	R1, A	R1← M[A]
LOAD	R2, B	R2 ← M[B]
LOAD	R3, C	R3 ← M[C]
LOAD	R4, C	R4 ← M[D]
ADD	R1, R1, R2	R1 ← R1 + R2
ADD	R3, R3, R2	R3 ← R3 + R4
MUL	R1, R1, R3	R1 ← R1*R3
STORE	X, R1	M[X] ← R1

The load instructions transfer the operands from memory to CPU registers. The add and multiply operations are executed with data in the registers without accessing memory. The result of the computations is then stored in memory with a store instruction.

1.4.6 Instruction Cycle

A program stays in the memory unit of the computer and has a sequence of instructions. The program is executed by going through a cycle for each instruction. Each instruction cycle is now subdivided into a sequence of sub cycles or phases. In the basic computer each instruction cycle has the following parts:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

After the completion of step 4, the control goes back to step 1 to fetch, decode and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Check Your Progress

Fill in the blanks:

1. CISC architecture in hardware requires the use of .....
2. The essential goal of RISC architecture involves an attempt to ..... execution time by simplifying the instruction set of the computer.
3. The immediate mode instruction has an ..... field rather than an address field.
4. In Indexed Addressing Mode the effective address is obtained by ..... the content of an index register to the address part of the instruction.
5. A memory address specifies operands residing in memory whereas those residing in ..... registers are specified by a register address.

1.5 LET US SUM UP

Computer Architecture deals with the issue of selection of hardware components and interconnecting them to create computers that achieve specified functional, performance and cost goals. The execution of the operation is performed on some data stored in computer registers or memory words. The way the operands are chosen during program. Selection of operands during program execution depends on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referred. The ***Instruction Set Architecture*** (ISA) is the part of the processor that is visible to the programmer or compiler writer. The ISA serves as the boundary between software and hardware. The reference manuals provided with a computer system contain a description of its physical and logical structure.

1.6 KEYWORDS

ISA: Instruction Set Architecture

MSI: Medium Scale Integration

LSI: Large Scale Integration

VLSI: Very Large Scale Integration

CISC: Cost Instruction Set Architecture

RISC: Reduced Instruction Set Architecture

1.7 QUESTIONS FOR DISCUSSION

1. What is the difference between CISC and RISC Architecture?
2. Differentiate between parallel processor architecture and BUS architecture.
3. Explain the various Addressing Modes:
   (a) Indexed Addressing Mode
   (b) Base Register Addressing Mode.
4. Discuss three and one address instruction.

Check Your Progress: Model Answers

1. Microcode
2. Reduce
3. Operand
4. Adding
5. Processor

1.8 SUGGESTED READINGS

Sajjan G. Shiva; *Computer Design and Architecture*; Marcel Dekker

Silvia Melitta Mueller, Wolfgang J. Paul; *Computer Architecture*; Springer

Joseph D. Dumas II; *Computer Architecture*; CRC Press

Nicholas P. Carter; *Schaum's Outline of Computer Architecture*; Mc. Graw-Hill Professional

LESSON

2

PIPELINE ARCHITECTURE

CONTENTS

2.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain pipeline architecture
- Describe MIPS series
- Understand Motorola 88000
- Identify SPARC microchannel architecture