**UNIT - I**

Chapter - 2 Instruction Set Architecture

2.1 Memory Locations and Addresses

* Memory consists of large array of words or bytes, each one having its own address.
* Each cell can store a bit of information i.e. 0 or 1. Each group of n bits is referred to as a word of information, and n is called the word length. The word length can vary from 8 to 64 bits. A unit of 8 bits is called a byte.
* Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers

k

from 0 through 2 -1 as the addresses of successive-locations in the memory).

k k

If 2 = no. of addressable locations; then 2 addresses constitute the address-space

24

of the computer. For example, a 24-bit address generates an address-space of 2

locations (16 MB).

**2.1.1 Byte-Addressability**

* It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to successive byte locations in the memory. Such a memory is called a byte- addressable memory.
* In byte-addressable memory, successive addresses refer to successive byte locations in the memory. Byte locations have addresses 0, 1, 2.... If the word-length is 32 bits, successive words are located at addresses 0, 4, 8... with each word having 4 bytes.

**2.1.2 Big-Endian & Little-Endian Assignments**

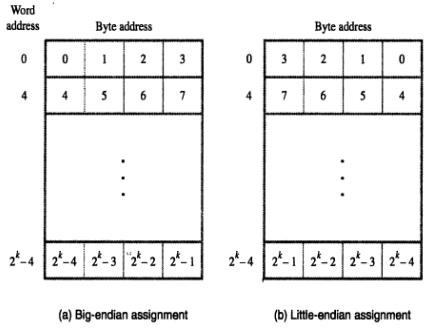
There are two ways in which byte-addresses are arranged

1. Big-Endian: Lower byte-addresses are used for the more significant bytes of the word.
2. Little-Endian: Lower byte-addresses are used for the less significant bytes of the word. In both cases, byte-addresses 0, 4, 8. are taken as the addresses of successive word

in the memory.

Consider a 32-bit integer (in hex): 0x12345678 which consists of 4 bytes: 12, 34, 56, and

78. Hence this integer will occupy 4 bytes in memory.



Assume, we store it at memory address starting 1000. On little-endian, memory will look like

|  |  |
| --- | --- |
| Address | Value |
| 1000 | 78 |
| 1001 | 56 |
| 1002 | 34 |
| 1003 | 12 |

On big-endian, memory will look like

|  |  |
| --- | --- |
| Address | Value |
| 1000 | 12 |
| 1001 | 34 |
| 1002 | 56 |
| 100 | 78 |
|  |  |

**2.1.3 Word Alignment**

In the case of a 32-bit word length, natural word boundaries occur at addresses 0, 4, 8, . . . ,the number of bytes in a word is a power of 2. Hence, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4, . . . , and for a word length of 64 (23 bytes), aligned words begin at byte addresses 0, 8, 16, . . . . There is no fundamental reason why words cannot begin at an arbitrary byte address.

2.2 Memory Operations

Two memory operations are: 1) Load (Read/Fetch) & 2) Store (Write).

The Load operation transfers a copy of the contents of a specific memory-location to the processor. The memory contents remain unchanged.

Steps for Load operation:

1. Processor sends the address of the desired location to the memory.
2. Processor issues read signal to memory to fetch the data.
3. Memory reads the data stored at that address.
4. Memory sends the read data to the processor.

The Store operation transfers the information from the register to the specified memory- location. This will destroy the original contents of that memory-location.

Steps for Store operation are:

1. Processor sends the address of the memory-location where it wants to store data.
2. Processor issues write signal to memory to store the data.
3. Content of register(MDR) is written into the specified memory-location.

2.3 Instructions and Instruction Sequencing

A computer must have instructions capable of performing the following basic operations. They are:

* + Data transfer between memory and processor register.
  + Arithmetic and logical operations on data.
  + Program sequencing and control.
  + I/O transfer.

**2.3.1 Register Transfer Notation:** The possible locations that may be involved during data transfer are:

* Memory Location
* Processor register

the expression

R2 ← [LOC] means that the contents of memory location LOC are transferred into processor register R2.

As another example, consider the operation that adds the contents of registers R2 and R3, and places their sum into register R4. This action is indicated as

R4 ← [R2] + [R3]

This type of notation is known as Register Transfer Notation (RTN). Note that the right-hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location

**2.3.2 Assembly Language Notation**

we use assembly language. For example, an instruction that causes the transfer, from memory location LOC to processor register R2, is specified by the statement

Load R2, LOC

The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R2 are overwritten.

The second example of adding two numbers contained in processor registers R2 and

R3 and placing their sum in R4 can be specified by the assembly-language statement

Add R4, R2, R3

In this case, registers R2 and R3 hold the source operands, while R4 is the destination.

we used the English words Load and Add to denote the required operations.

In the assembly-language such operations are defined by using *mnemonics*, which are typically abbreviations of the words describing the operations.

**2.3.3 RISC and CISC Instruction sets**

|  |  |
| --- | --- |
| **RISC** | **CISC** |
| Simple instructions taking one cycle. | Complex instructions taking multiple cycle. |
| Instructions are executed by hardwired control unit. | Instructions are executed by microprogrammed control unit. |
| Few instructions. | Many instructions. |
| Fixed format instructions. | Variable format instructions. |
| Few addressing modes, and most instructions have register to register addressing mode. | Many addressing modes. |
| Multiple register set. | Single register set. |
| Highly pipelined. | No pipelined or less pipelined. |

**2.3.4 Introduction to RISC Instruction Sets**

Two key characteristics of RISC instruction sets are:

• Each instruction fits in a single word.

• A *load/store architecture* is used, in which

– Memory operands are accessed only using Load and Store instructions.

– All operands involved in an arithmetic or logic operation must either be in processor

registers, or one of the operands may be given explicitly within the instruction word.

At the start of execution of a program, all instructions and data used in the program are

stored in the memory of a computer. If operands are expected to be in processor register, use Load instructions which copy the contents of a memory location into a processor register. Load instructions are of the form

**Load processor\_register, memory\_location**

The memory location can be specified in several ways. The term *addressing modes* is used

to refer the memory location.

Consider a typical arithmetic operation i.e statement C = A + B in a high-level language program instructs the computer to add the current values of the two variables called A and B, and to assign the sum to a third variable, C.

For above operation the sequence of simple machine instructions.

**Load R2, A**

**Load R3, B**

**Add R4, R2, R3**

**Store R4, C**

We say that Add is a ***three-operand*, or a *three-address***, instruction of the form

**Add destination, source1, source2**

The Store instruction is of the form

**Store source, destination**

where the source is a processor register and the destination is a memory location. Observe

that in the Store instruction the source and destination are specified in the reverse order from the Load instruction; this is a commonly used convention.

**2.3.5 Instruction Execution and Straight Line Sequencing:**

**Instruction Execution:** There are 2 phases for executing an instruction. They are,

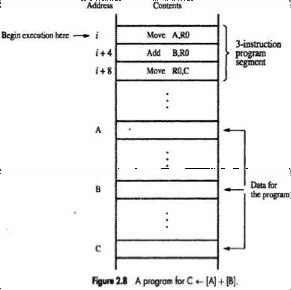
* Instruction Fetch
* Instruction Execution

**Instruction Fetch:** The instruction is fetched from the memory location whose address is in PC. This is then placed in IR.

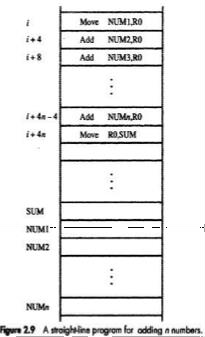
**Instruction Execution:** Instruction in IR is examined and decoded to determine which operation is to be performed.

**Program execution Steps: [** figure is copied from 5-edition, but you draw 6-edition fig]

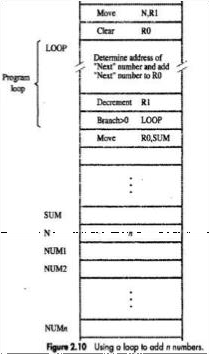
* + To begin executing a program, the address of first instruction must be placed in PC.
  + The processor control circuits use the information in the PC to fetch & execute instructions one at a time in the order of increasing order.
  + This is called Straight line sequencing.
  + During the execution of each instruction, the PC is incremented by 4 to point to the address of next instruction.



**2.3.6 Branching:** The Address of the memory locations containing the n numbers are symbolically given as NUM1, NUM2.....NUMn. Separate Add instruction is used to add each number to the contents of register R0. After all the numbers have been added, the result is placed in memory location SUM. Instead of using a long list of Load and Add instructions, it is possible to implement a program loop in which the instructions read the next number in the list and add it to the current sum. To add all numbers, the loop has to be executed as many times as there are numbers in the list.



**Using loop to add ‘n’ numbers:**



* Number of entries in the list, n is stored in memory location M. Register R1 is used as a counter to determine the number of times the loop is executed.
* Content location M are loaded into register R1 at the beginning of the program.
* It starts at location Loop and ends at the instruction, Branch>0.During each pass, the address of the next list entry is determined and the entry is fetched and added to R0.
* Decrement R1; It reduces the contents of R1 by 1 each time through the loop.

***branch* instruction**: This type of instruction loads a new address into the program counter. As a result, the processor fetches and executes the instruction at this new address, called the *branch target*.

**A *conditional branch***instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

In the above Figure, the instruction

Branch\_if\_[R2]*>*0 LOOP is a conditional branch instruction that causes a branch to location LOOP if the contents of register R2 are greater than zero.

**2.4 Addressing Modes [ Follow the class notes]**

The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.

**Generic Addressing Modes:**

* Immediate mode
* Register mode
* Absolute mode
* Register Indirect mode
* Index mode
* Base with index

**Register Mode:** The operand is the contents of the processor register. The name (address) of the register is given in the instruction.

**Absolute Mode (Direct Mode):** The operand is in a memory location. The address of this location is given explicitly in the instruction.

**Immediate Addressing Mode:** The operand is specified in the instruction itself. For eg. Move R2, #200

**Direct Addressing Mode:** Operand resides in Memory and its address is given explicitly in the address field of an instruction.

**Register Addressing Mode:** name of the register (address code of a specific general purpose register) appears in the address field of an instruction i.e.

**Index Mode:** The effective address of an operand is generated by adding a constant value to the contents of a register. The constant value uses either special purpose or general purpose register.

**X (RI)** where **X** – denotes the constant value contained in the instruction **Ri** – name of the register involved. The Effective Address of the operand EA**=X + [Ri]** The index register R1 contains the address of a new location and the value of X defines an offset (also called a displacement).

To find operand first go to Reg R1 (using address)-read the content from R1 i.e. 1000 Add the content 1000 with offset 20 to get the result. Here the constant X refers to the new address and the contents of index register that defines the offset to the operand.

The sum of two values is given explicitly in the instruction and the other is stored in register. Add 20(R1), R2 (or) EA=>1000+20=1020

**Relative Addressing:** It is same as index mode. The difference is, instead of general purpose register, here we can use program counter (PC).

**2.6 Stacks**

In modern computers, a stack is implemented by using a portion of the main memory for this purpose.

One processor register, called the stack pointer (SP), is used to point to the stack.

Data can be stored in a stack with successive elements occupying successive memory locations.

We use a stack that grows in the direction of decreasing memory.

If we assume a byte-addressable memory with a 32-bit word length, the **push** operation can be implemented as

**Subtract SP, SP, #4**

**Store Rj, (SP)**

where the Subtract instruction subtracts 4 from the contents of SP and places the result in SP.

The pop operation can be implemented as

**Load Rj, (SP)**

**Add SP, SP, #4**

These two instructions load (pop) the top value from the stack

(Write Diagram in text book )

**2.7 Subroutines**

**(Refer Text Book)**

**2.8 Additional Instructions**

**(Refer Text Book)**

**Logical instruction, Shift and Rotate Instruction, Mul,Div instructions**