Computer Architecture

Second Year- Electrical Engineering

By: Aum_Alhuda Gani

Instruction Set Architecture and Design

A program consists of a number of instructions that have to be accessed in a certain order. Programs need to deal with main memory. The main memory is considered as a sequence of cells each capable of storing **n** bits. Information stored and/or retrieved from the memory needs to be addressed. There are a number of different ways to address memory locations (**addressing modes**).

Memory Locations and Operations

The (main) memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0. These cells are organized in the form of groups of fixed number, say **n**, of cells that can be dealt with as an entity. An entity consisting of 8 bits is called a **byte**. In many systems, the entity consisting of n bits that can be stored and retrieved in and out of the memory using one basic memory operation is called a **word** (the smallest addressable entity). Typical size of a word ranges from 16 to 64 bits. It is, however, customary to express the size of the memory in terms of bytes.

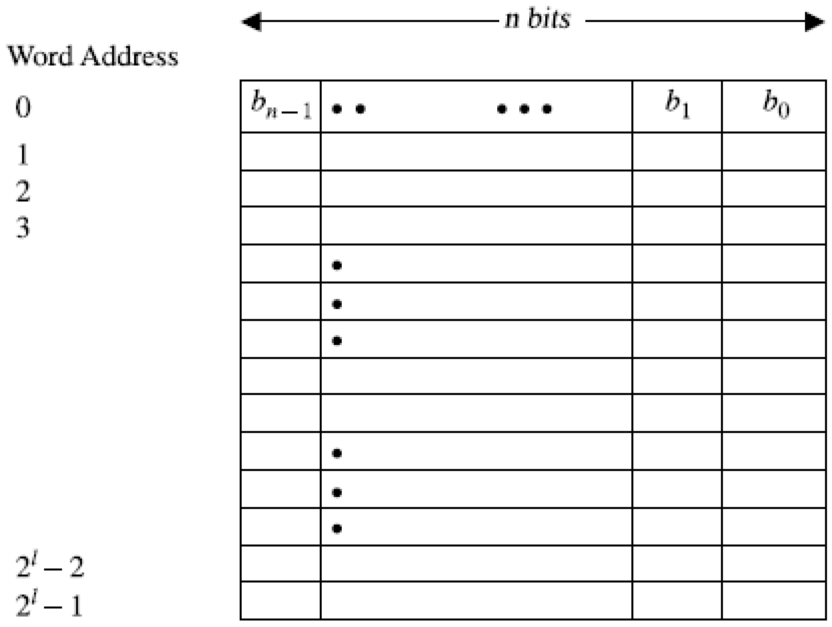
In order to be able to move a word in and out of the memory, a distinct address has to be assigned to each word. This address will be used to determine the location in the memory.

The number of bits, **L**, needed to distinctly address M words in a memory is given by **L= log2 M**. For example, if the size of the memory is 64 M (read as 64 mega words), then the number of bits in the address is log2 ($64 * 2^{20}$) = log2(2^{26}) =26 bits. Alternatively, if the number of bits in the address is L, then the maximum memory size (in terms of the number of words that can be addressed using these L bits) is $\mathbf{M = 2^L}$.

There are two basic memory operations:

Memory write: during a memory write operation a word is stored into a memory location whose address is specified.

Memory read: a word is read from a specified (addressed) memory location.



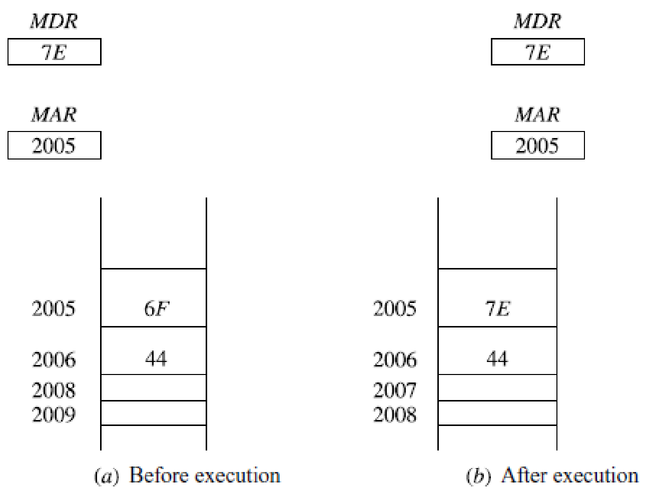
Three basic steps are needed for the CPU to perform a **write operation** into a specified memory location:

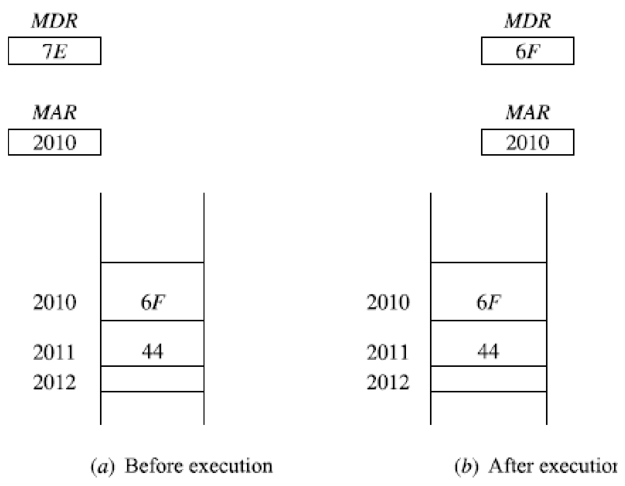
1. The word to be stored into the memory location is first loaded by the CPU into a specified register, called the memory data register (**MDR**).
2. The address of the location into which the word is to be stored is loaded by the CPU into a specified register, called the memory address registers (**MAR**).
3. A signal, called write, is issued by the CPU indicating that the word stored in the MDR is to be stored in the memory location whose address in loaded in the MAR.

Similar to the write operation, three basic steps are needed in order to perform a memory **read operation**:

1. The address of the location from which the word is to be read is loaded into the MAR.
2. A signal, called read, is issued by the CPU indicating that the word whose address is in the MAR is to be read into the MDR.
3. The required word will be loaded by the memory into the MDR by the CPU.

Write Operation Read Operation





ADDRESSING MODES

Information involved in any operation performed by the CPU needs to be addressed. In computer terminology, such information is called the **operand**. Therefore, any instruction issued by the processor must carry at least two types of information. These are the operation to be performed, encoded in what is called the **op-code** field, and the address information of the operand on which the operation is to be performed, encoded in what is called the address field.

Instructions can be classified based on the number of operands as:

Instruction Class	Example
Three-address	ADD R1,R2,R3
	ADD A,B,C
Two-address	ADD R1,R2
	ADD A,B
One-address	ADD R1
Zero-address	ADD (SP), (SP)

Description:

A three-address instruction takes the form **operation add-1, add-2, add-3**. In this form, each of add-1, add-2, and add-3 refers to a register or to a memory location. Consider, for example, the instruction **ADD R1,R2,R3**. This instruction indicates that the operation to be performed is addition. It also indicates that the values to be added are those stored in registers R1 and R2 that the results should be stored in register R3.

An example of a three-address instruction that refers to memory locations may take the form **ADD A,B,C**. The instruction adds the contents of memory location A to the contents of memory location B and stores the result in memory location C.

A two-address instruction takes the form **operation add-1, add-2**. In this form, each of add-1 and add-2 refers to a register or to a memory location. Consider, for example, the instruction **ADD R1,R2**. This instruction adds the contents of register R1 to the contents of register R2 and stores the results in register R2. The original contents of register R2 are lost due to this operation while the original contents of register R1 remain intact. This instruction is equivalent to a three-address instruction of the form ADD R1,R2,R2. A similar instruction that uses memory locations instead of registers can take the form: **ADD A,B**. In this case, the contents of memory location A are added to the contents of memory location B and the result is used to override the original contents of memory location B.

The operation performed by the three-address instruction ADD A,B,C can be performed by the two two-address instructions MOVE B,C and ADD A,C. This is because the first instruction moves the contents of location B into location C and the second instruction adds the contents of location A to those of location C (the contents of location B) and stores the result in location C.

A one-address instruction takes the form **ADD R1**. In this case the instruction implicitly refers to a register, called the **Accumulator Racc**, such that the contents of the accumulator is added to the contents of the register R1 and the results are stored back into the accumulator **Racc**. If a memory location is used instead of a register then an instruction of the form ADD B is used. In this case, the instruction adds the content of the accumulator Racc to the content of memory location B and stores the result back into the accumulator Racc. The instruction ADD R1 is equivalent to the three-address instruction ADD R1, Racc, Racc or to the two-address instruction ADD R1, Racc.

It is interesting to indicate that there exist zero-address instructions. These are the instructions that use stack operation. A stack is a data organization mechanism in which the last data item stored is the first data item retrieved. Two specific operations can be performed on a stack. These are the push and the pop operations.

When CPU executes an instruction, it performs the specified function on data (operands). The operands may be:

i. Part of the instruction.
ii. Reside in one of the internal registers.
iii. Stored at an address on memory.
iv. Held at an input/output (I/O) port.

The different ways in which operands can be addressed are called the **Addressing Modes** that include:

1. Immediate Mode: The value of the operand is (immediately) available in the instruction itself.

Example: **LOAD #1000, Ri** ; $Ri \leftarrow 1000$

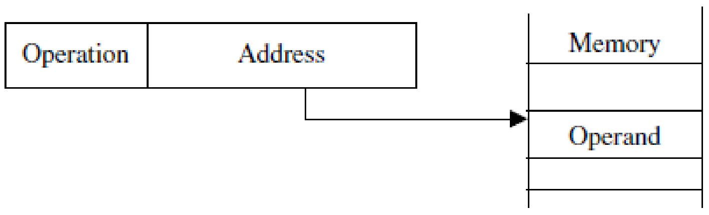
The operation to be performed is to load a value into a register. The source operand is (immediately) given as 1000, and the destination is the register Ri.

Value 1000 mentioned in the instruction is the operand itself and not its address. It is customary to prefix the operand by the special character (#).

2. Direct (Absolute) Mode: the address of the memory location that holds the operand is included in the instruction.

Example: **LOAD 1000, Ri** ; Ri ← M[1000]

That loading the value of the operand stored in memory location 1000 into register Ri.



3. Indirect Mode: what is included in the instruction is not the address of the operand, but rather a name of a register or a memory location that holds the (effective) address of the operand.

Example:

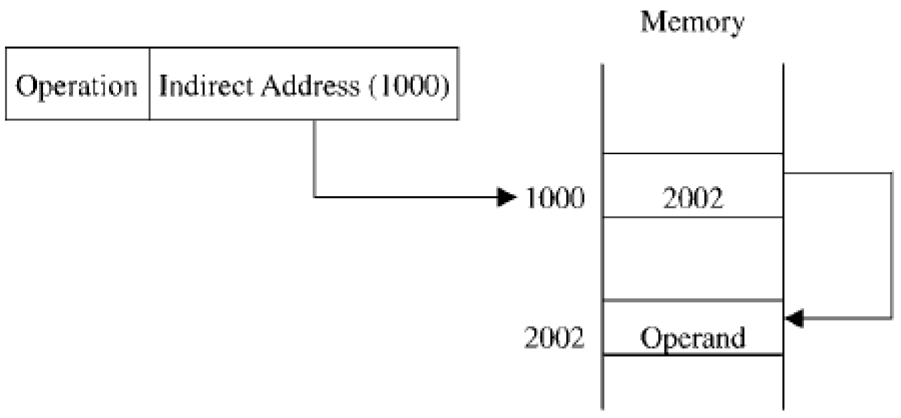
LOAD (1000), Ri. ; Ri ← M[[1000]]

LOAD (Rj), Ri ; Ri ← M[Rj]

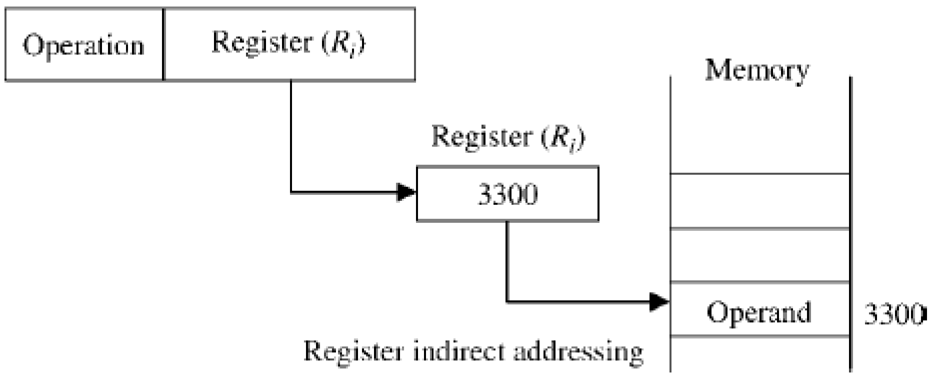
The meaning of this instruction is to load register Ri with the contents of the memory location whose address is stored at memory address 1000. Because indirection

Two types of indirect addressing:

i. Memory indirect addressing (a memory location hold the address of the operand.
ii. Register indirect addressing (a register holds the address of the operand).



Memory indirect addressing

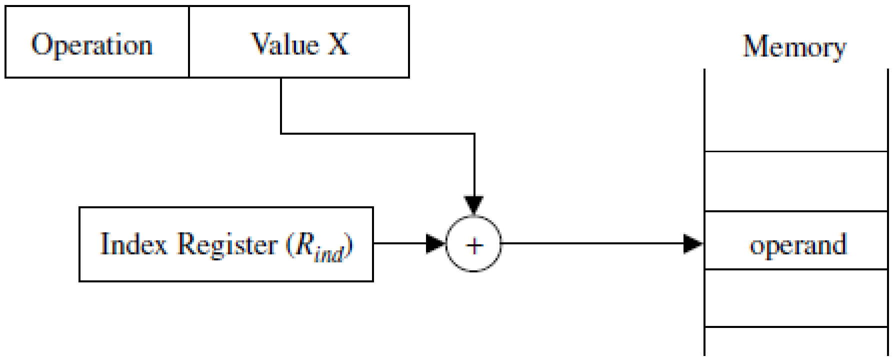


Register indirect addressing

4. Indexed Mode: the address of the operand is obtained by adding a constant of the content of a register, called the index register.

Example: **LOAD X(R_{ind}), Ri** ; Ri ← M[R_{ind} + X]

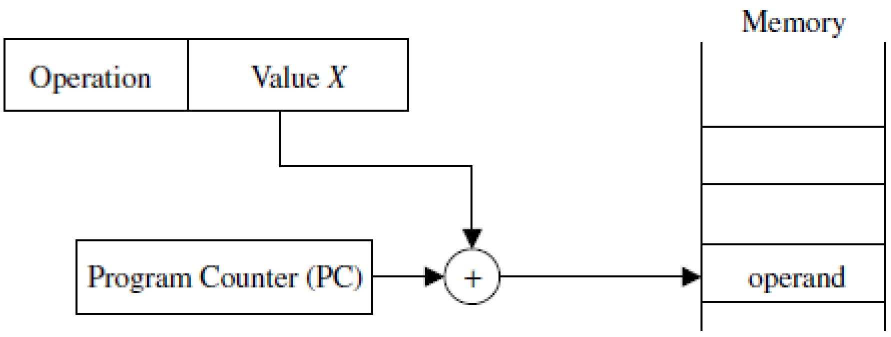
This instruction loads register Ri with the contents of the memory location whose address is the sum of the contents of register **R_{ind}** and the value X.



5. Relative Mode: it is the same as indexed addressing **except** that the program counter (PC) replaces the index register.

Example: **LOAD X(PC), Ri** ; Ri ← M[PC + X]

Loads register Ri with the contents of the memory location whose address is the sum of the contents of the program counter (PC) and the value X.



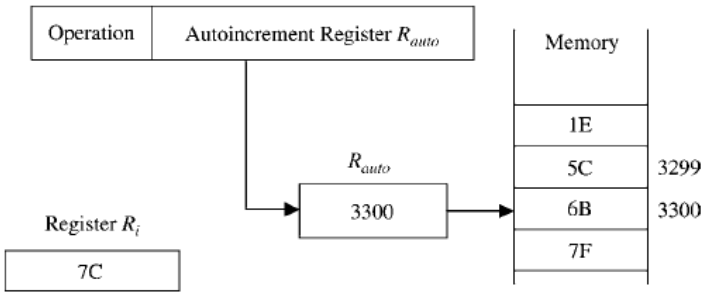
6. Autoincrement Mode: it is similar to the register indirect addressing mode in the sense that the effective address of the operand is the content of a register; call it the autoincrement register that is included in the instruction. The content of the autoincrement register is automatically incremented after accessing the operand.

Example: **LOAD (R_{auto}) +, Ri** ; Ri ← M[R_{auto}]

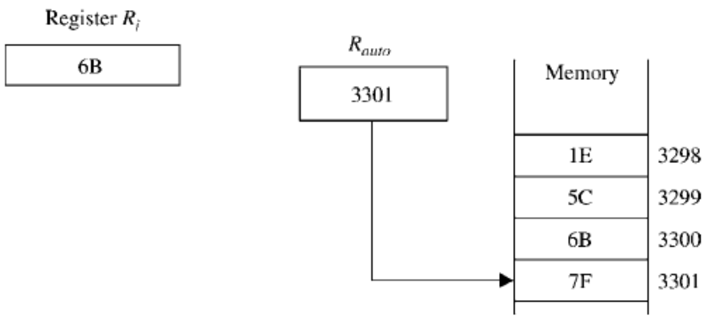
R_{auto} ← R_{auto} +1

7. Autodecrement Mode: the content of the autodecrement register is first decremented and the new content is used as the effective address of the operand.
Example: **LOAD _ (Rauto), Ri** ; R_{auto} ← R_{auto} - 1

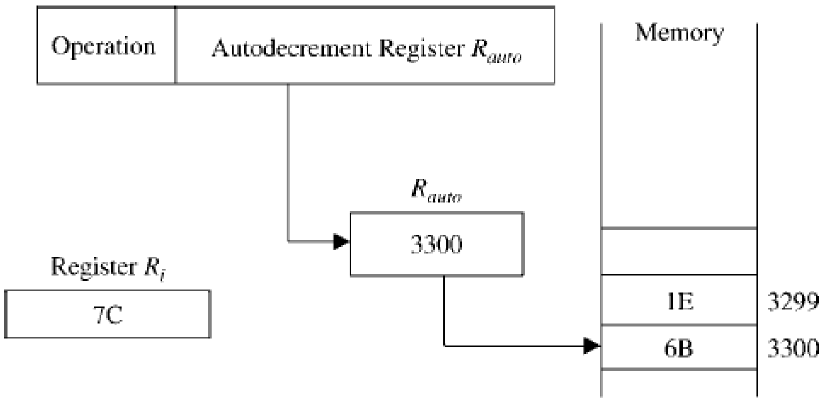
Ri ← M[R_{auto}]



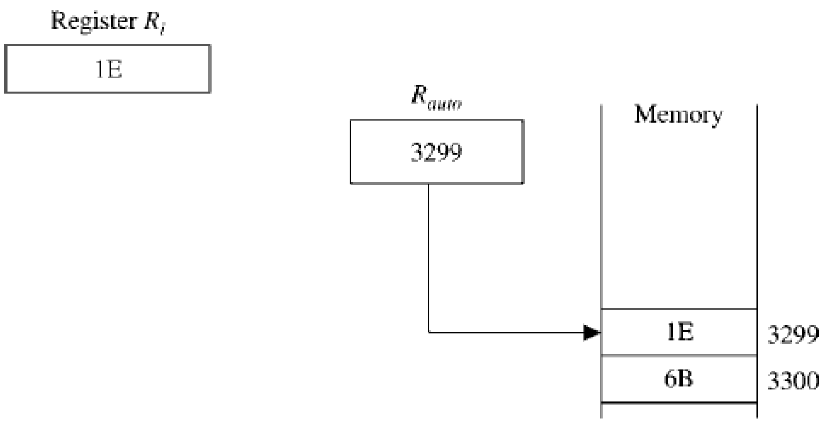
(a) Before execution



(b) After execution



(a) Before execution



(b) After execution