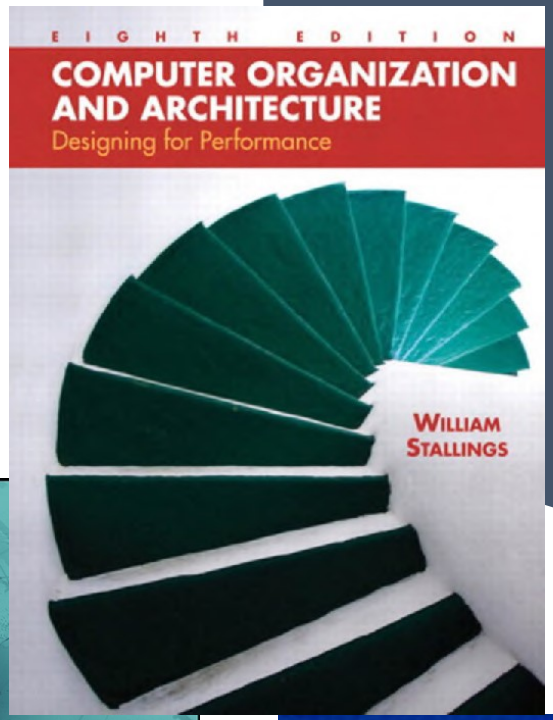
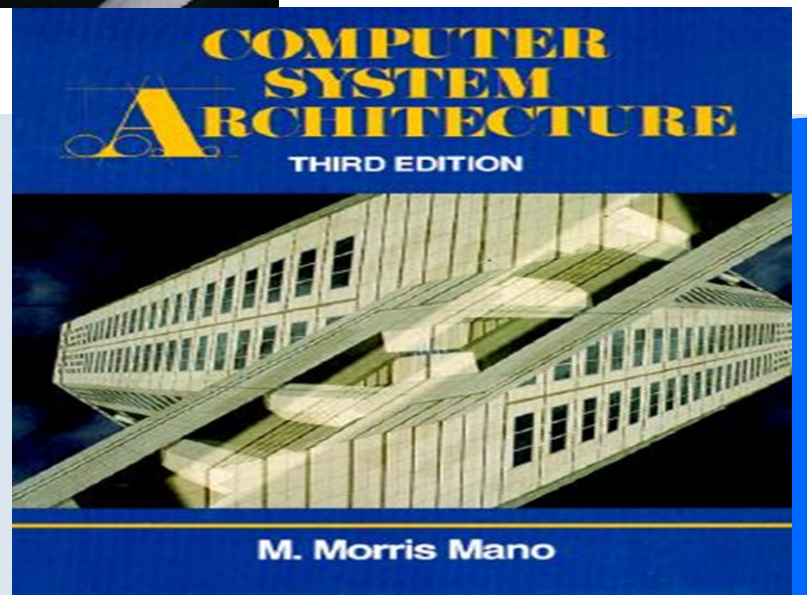
Computer Architecture

2nd Class, Computer Science Dept.

By Dr. Ahmed Al-Taie,

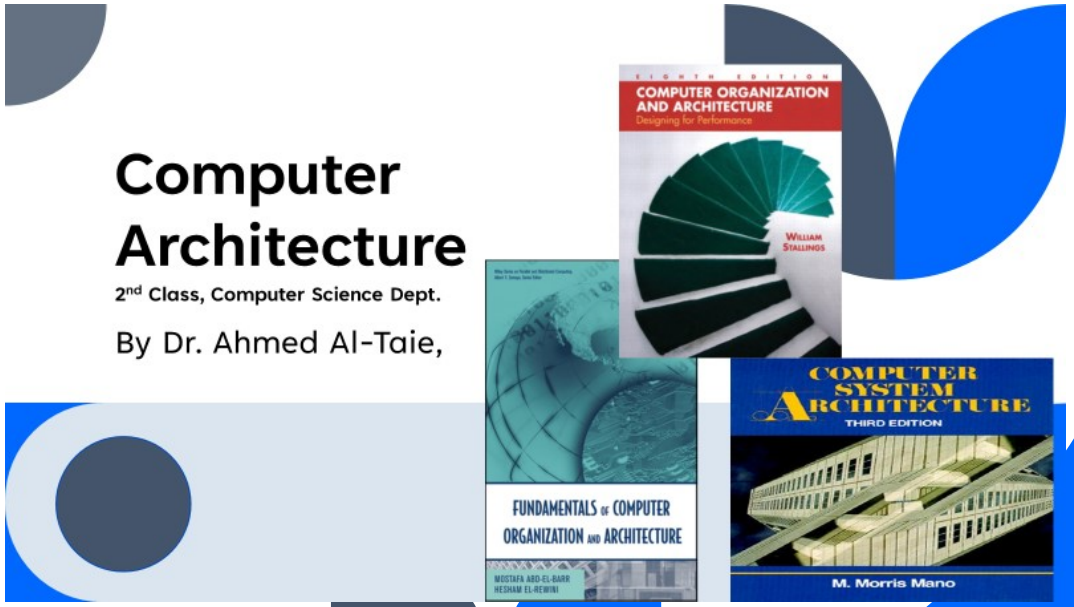




Chapter 2
Instruction Set Architecture and Design

Outline

- Memory Locations and Operations
- Addressing Modes
- Instruction Types
- Programming Examples



Instruction Set Architecture and Design

The basic principles involved in instruction set architecture and design.

- The discussion starts with a consideration of **memory locations and addresses.**
- We present an **abstract model** of the **main memory** in which it is considered as **a sequence of cells** each **capable of storing n bits**.

Storing and retrieving information into and from the memory.

- The information **stored and/or retrieved** from the **memory needs to be addressed**.
- A discussion on a **number of different ways to address memory locations** (**addressing modes**) is the next topic to be discussed in the Lecture.

A **program consists of** a **number of instructions** that have to be **accessed in a certain order.**

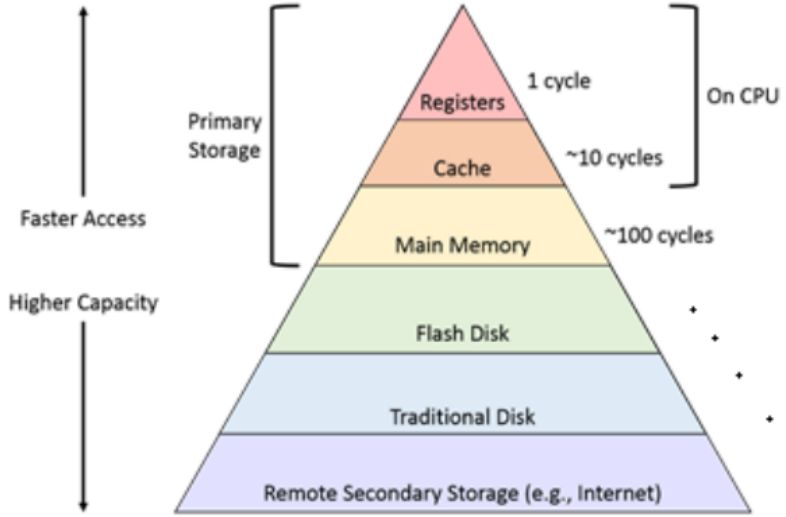
- That **motivates us to explain** the issue of **instruction execution and sequencing** in some **detail**.
- We then show **the application** of the **presented addressing modes** and **instruction characteristics** in writing sample segment codes for **performing a number of simple programming tasks**.

Instruction Set Architecture and Design

A unique characteristic of computer memory is that it should be **organized** in **a hierarchy**.

- In such **hierarchy**, **larger and slower memories** are used to **supplement smaller and faster** ones.
- A **typical memory hierarchy starts** with **a small, expensive**, and **relatively fast module**, called **the cache**.

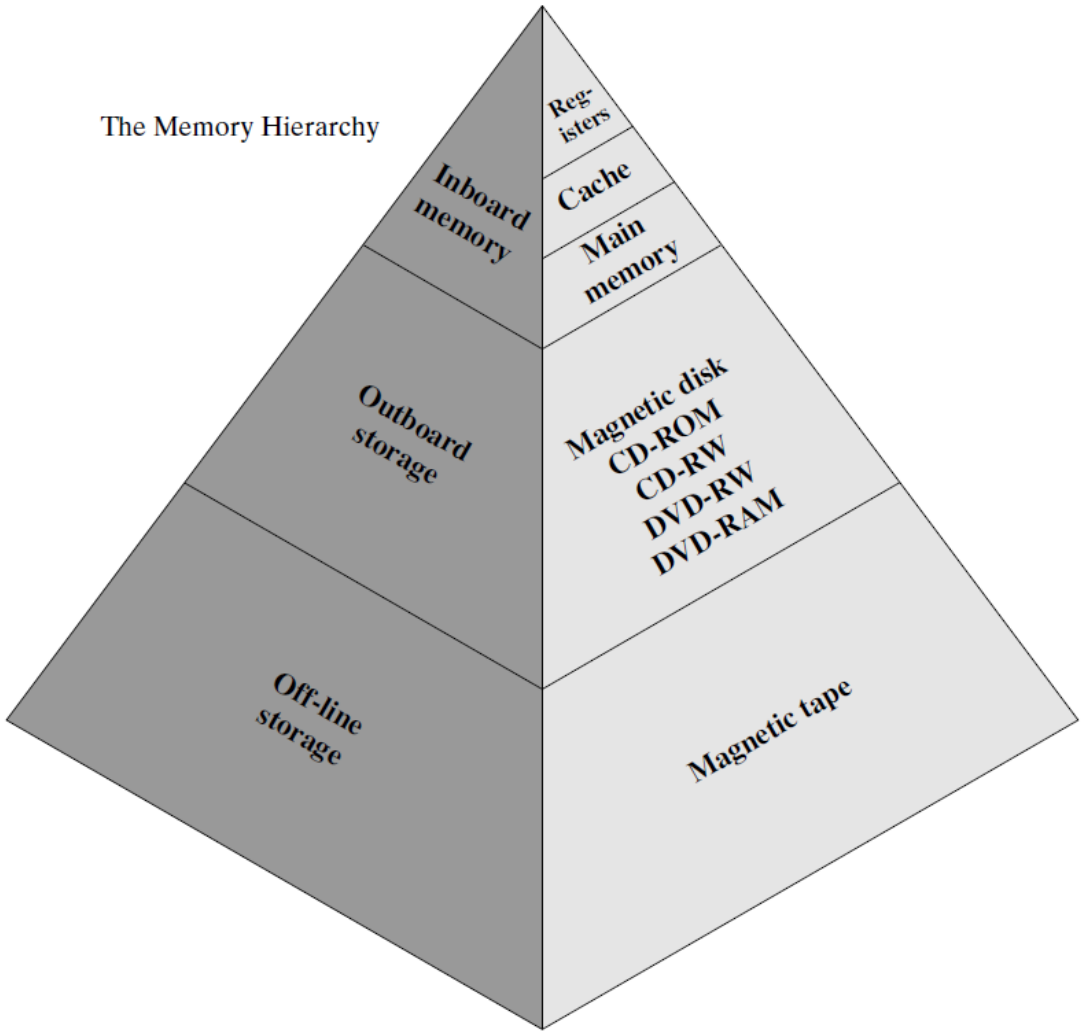
The **cache** is followed in the hierarchy by a **larger, less expensive,** and **relatively slow main memory part.**



Memory Hierarchy.

- **Cache and main memory** are built using **semiconductor** material.
- They are followed in the hierarchy by **larger**, **less expensive**, and **far slower magnetic memories** that consist of **the (hard) disk** and **the tape**

Instruction Set Architecture and Design



- **The characteristics and factors influencing the success of** the **memory hierarchy** of a computer are discussed in detail in Chapters 6 and 7.
- Our concentration in this Lecture is **on the (main) memory** from the programmer's point of view.
- In particular, we **focus** on the **way information is stored in and retrieved** out of the memory.

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 atomic 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.**
- For example, the size of a memory of a personal computer is **256 Mbytes**, **that is,**
- **$256 * 2^{20} = 2^{28}$ bytes.**

MEMORY LOCATIONS AND OPERATIONS

To **read a word in** and **write it 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 in which a given word is to be stored.
- **This is called a memory write operation.**

Similarly, the **address** will be **used** to **determine the memory location** from which a word is to be **retrieved from the memory**.

- **This is called a memory read operation**.
- The number of bits, *l*, needed to distinctly address M words in a memory is given by $l = \log_2 M$.

- **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 $\log_2 (64 \ 2^{20}) = \log_2 (2^{26}) = 26$ **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 $M = 2^l$.

MEMORY LOCATIONS AND OPERATIONS

- As mentioned above, there are **two basic memory operations**. These are the **memory write** and **memory read operations**.
- During **a memory write** operation **a word is stored** into **a memory location** whose **address is specified**.
- During **a memory read** operation **a word is read** from **a memory location** whose **address is specified**.
- Typically, **memory read and memory write operations** are **performed** by the **central processing unit (CPU)**.

Figure 2.1 illustrates the **concept of memory words** and **word address** as explained above.

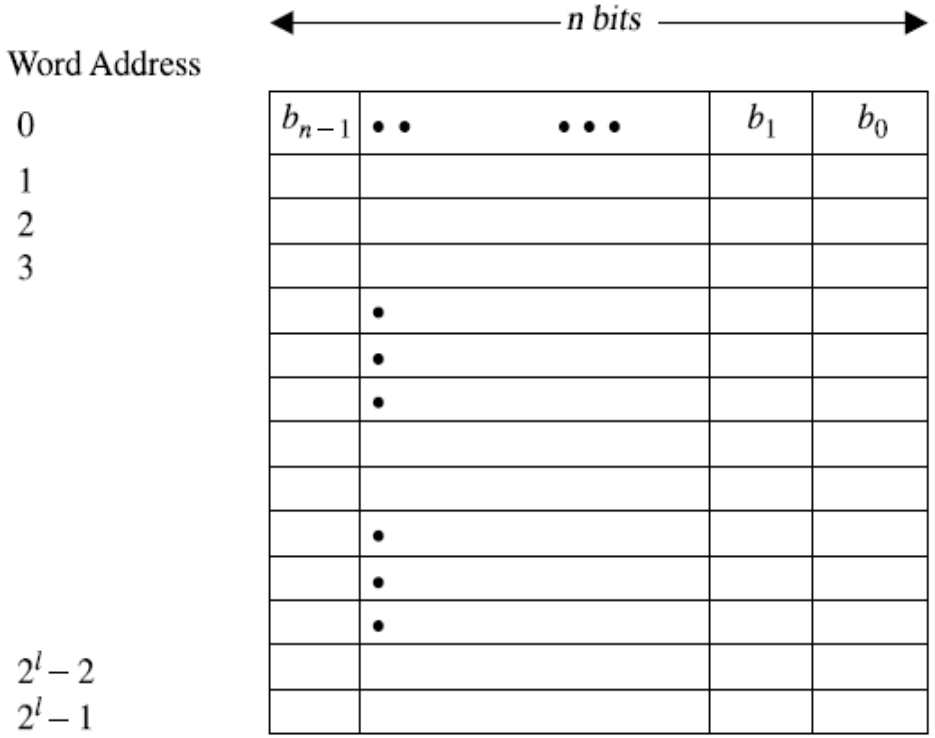


Figure 2.1 Illustration of the main memory addressing

MEMORY LOCATIONS AND OPERATIONS

Three basic steps are **needed** in order **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 register** (**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**.

MEMORY LOCATIONS AND OPERATIONS

Figure 2.2 illustrates the **operation of writing** the **word** given by **7E** (in **hex**) into the **memory location** whose **address** is **2005**.

- **Part a** of the figure **shows** the **status** of the **registers** and **memory locations involved** in the **write operation before** the **execution of the operation.**
- **Part b** of the figure **shows** the **status after** the **execution of the operation**.
- It is worth mentioning that the **MDR** and the **MAR** are **registers** used **exclusively** by the **CPU** and are **not accessible** to the **programmer**.

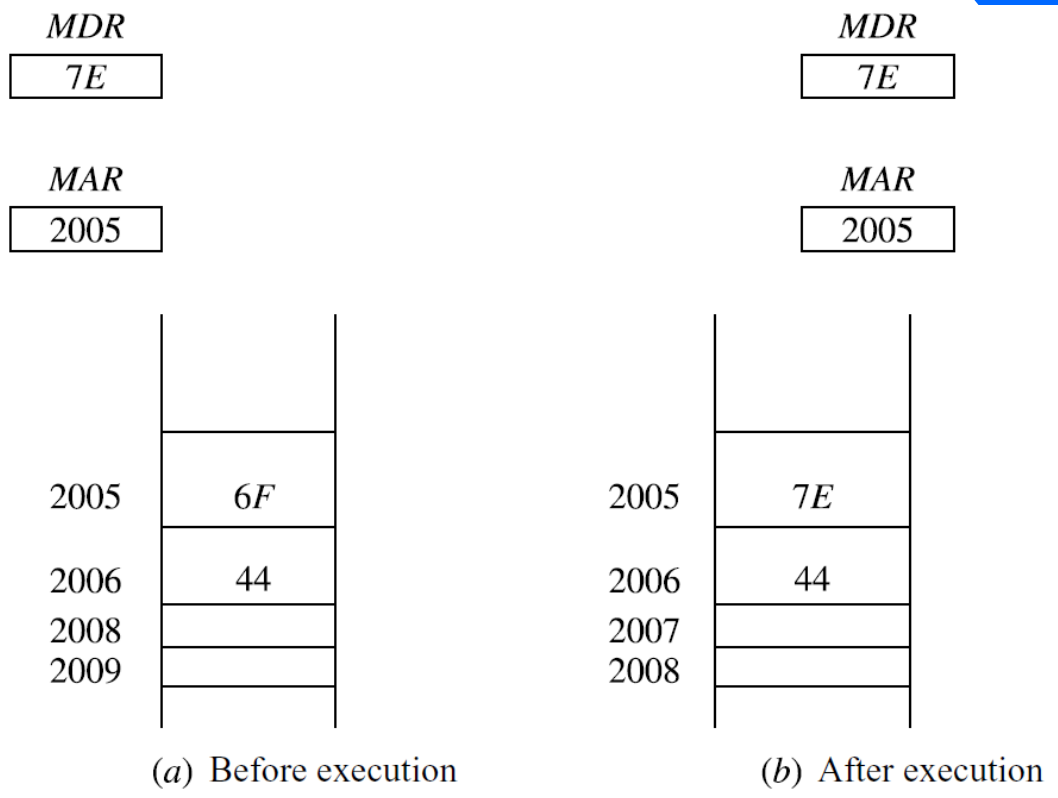


Figure 2.2 Illustration of the memory write operation

MEMORY LOCATIONS AND OPERATIONS

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. **After some time**, corresponding to the **memory delay** in **reading** the **specified word**, the **required word** will be **loaded** by the **memory** into the **MDR** **ready for use** by the **CPU**.

MEMORY LOCATIONS AND OPERATIONS

Figure 2.3 illustrates the **operation of reading** the **word stored** in the **memory location** whose **address** is **2010**.

- **Part a** of the figure **shows** the **status** of the **registers** and **memory locations** involved in the **read operation before** the **execution** of the **operation**.
- **Part b** of the figure **shows** the **status after** the **read operation**.

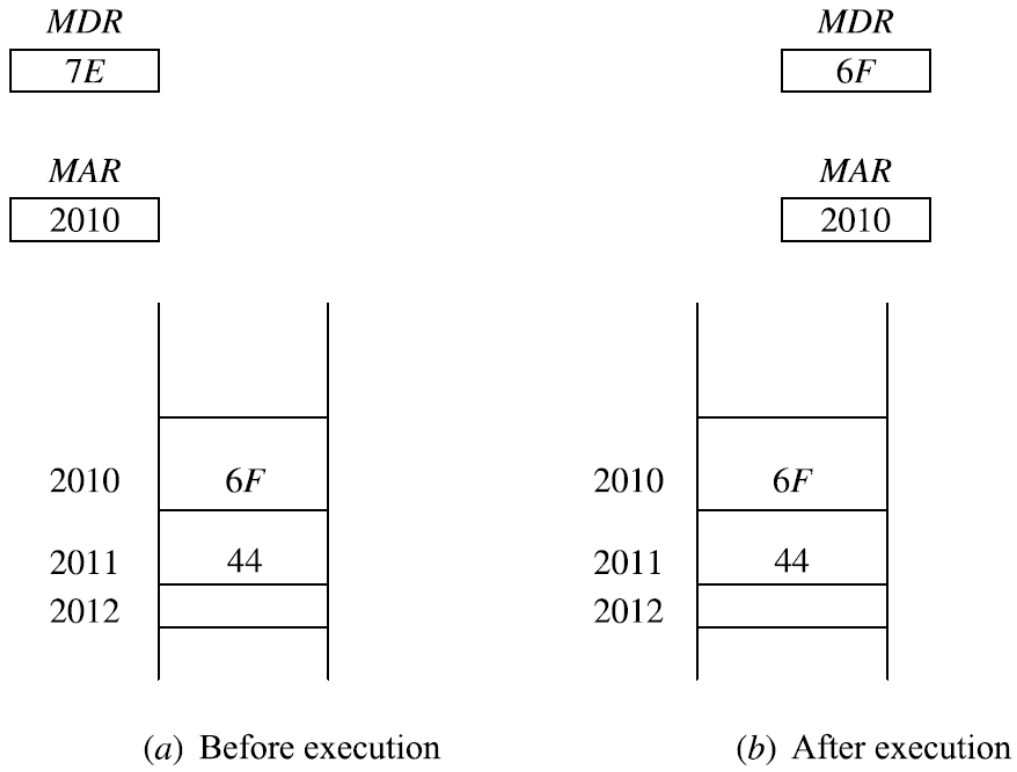


Figure 2.3 Illustration of the memory read operation

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 (1) the operation to be performed**, **encoded** in what is called **the op-code field**, and (2) **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: **three-address, two-address, one-and-half-address, one-address**, and **zero-address. We explain these classes together with simple examples in the following paragraphs**.

ADDRESSING MODES

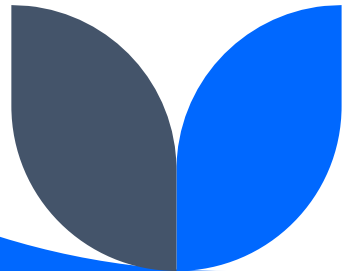
It should be noted that in **presenting these examples**, we would use **the convention operation, source, destination** to **express any instruction**.

In that convention, operation represents the **operation to be performed**, for example, **add, subtract, write, or read.**

The **source field** represents the **source operand(s)**.

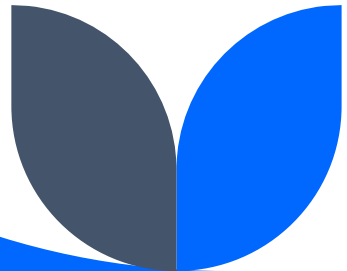
The **source operand** can **be a constant, a value stored in a register, or a value stored in the memory.**

The **destination field represents** the **place where the result of the operation is to be stored**, **for example**, **a register** or a **memory location**.



ADDRESSING MODES

- 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.



ADDRESSING MODES

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.

ADDRESSING MODES

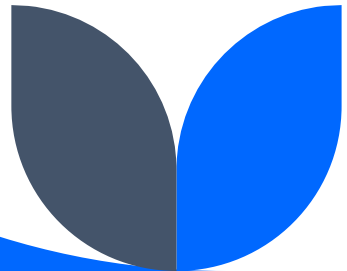
The operation performed by the three-address instruction

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.



ADDRESSING MODES

A one-address instruction takes the form **ADD R1**.

- In this case the instruction implicitly refers to a register, called the Accumulator $\mathbf{R_{acc}}$, 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 R_{acc}.
- 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,R_{acc}.

ADDRESSING MODES

- Between the two- and the one-address instruction, there can be a **one-and-half address** instruction.
- Consider, for example, the instruction **ADD B,R1**.
- In this case, the instruction adds **the contents** of **register R1** to the **contents of memory location B** and stores the result in register R1.
- Owing to the fact that the instruction **uses two types of addressing**, that is, a **register** and a **memory** location, it is **called** a **one-and-half-address** instruction. This is because register addressing needs a smaller number of bits than those needed by memory addressing.

ADDRESSING MODES

- 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**.

Figure 2.4 illustrates these two operations.

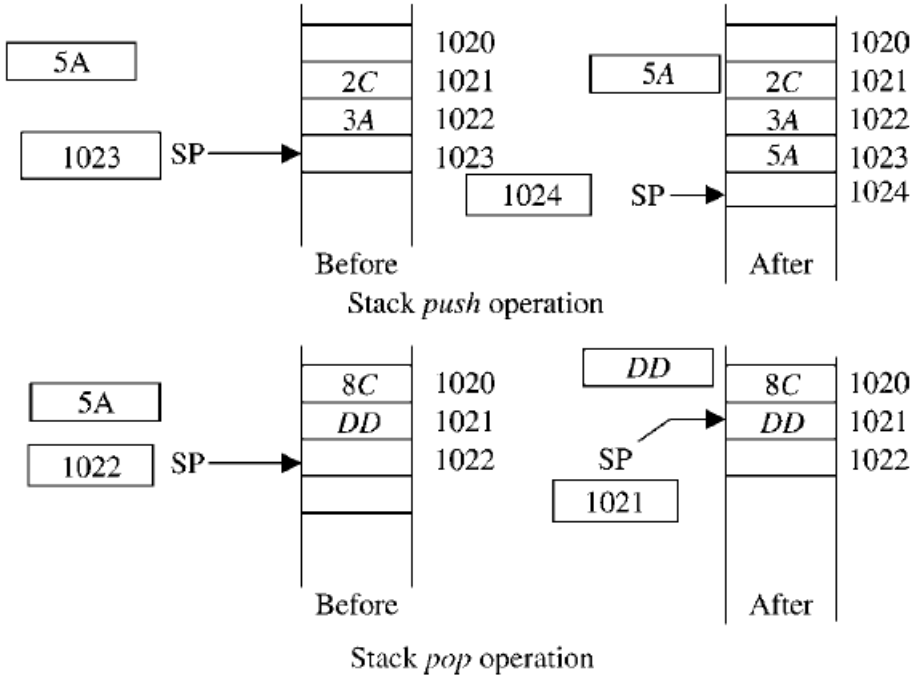


Figure 2.4 The stack push and pop operations

ADDRESSING MODES

As can be seen, a specific register, called the stack pointer (SP), is used to indicate the stack location that can be addressed.

In the stack push operation, the SP value is used to indicate the location (called the top of the stack) in which the value (5A) is to be stored (in this case it is location 1023). After storing (pushing) this value the SP is incremented to indicate to location 1024. In the stack pop operation, the SP is first decremented to become 1021. The value stored at this location (DD in this case) is retrieved (popped out) and stored in the shown register.

ADDRESSING MODES

Different **operations** can be **performed** using the **stack structure**.

Consider, for example, an instruction such as ADD (SP)+, (SP).

The instruction adds the contents of the stack location pointed to by the SP to those pointed to by the SP + 1 and stores the result on the stack in the location pointed to by the current value of the SP.

Figure 2.5 illustrates such an addition operation. Table 2.1 summarizes the instruction classification discussed above.

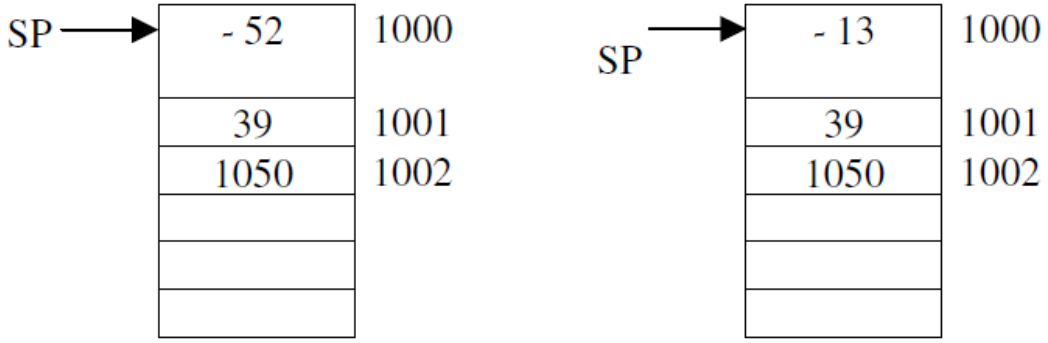


Figure 2.5 Addition using the stack

TABLE 2.1 Instruction Classification

Instruction class	Example
Three-address	*ADD* R_1,R_2,R_3
	ADD A,B,C
Two-address	*ADD* R_1,R_2
	ADD A,B
One-and-half-address	*ADD B*,R_1
One-address	*ADD* R_1
Zero-address	*ADD (SP)+, (SP)*

ADDRESSING MODES

- The **different ways** in which **operands can be addressed** are called the **addressing modes**.
- **Addressing modes differ in the way the address information of operands is specified**.
- The **simplest addressing mode** is to include the **operand itself** in the instruction, that is, ***no address information is*** needed. This is called immediate addressing.
- A more involved **addressing mode** is to compute the **address of the operand** by **adding a constant value** to the **content of a register**.
- **This is called indexed addressing.**
- Between these **two addressing modes there exist a number of other addressing modes** including **absolute addressing, direct addressing,** and **indirect addressing**.
- A number of different addressing modes are explained below.

ADDRESSING MODES 1- Immediate Mode

- According to this addressing mode, the **value of the operand** is (**immediately**) available **in the instruction itself**.
- Consider, for example, the case of loading the decimal value 1000 into a register Ri. This operation can be performed using an instruction such as the following:
- LOAD #1000, Ri.
- In this instruction, 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.
- It should be noted that in order to indicate that the value 1000 mentioned in the instruction is the operand itself and not its address (immediate mode), it is customary to prefix the operand by the special character (#).
- As can be seen the use of the immediate addressing mode is simple. The use of immediate addressing leads to poor programming practice.
- This is because a change in the value of an operand requires a change in every instruction that uses the immediate value of such an operand. A more flexible addressing mode is explained below.

ADDRESSING MODES

2- Direct (Absolute) Mode

- According to this addressing mode, the address of the memory location that holds the operand is included in the instruction.
- Consider, for example, the case of loading the value of the operand stored in memory location 1000 into register Ri.
- This operation can be performed using an instruction such as LOAD 1000, Ri. In this instruction, the source operand is the value stored in the memory location whose address is 1000, and
- the destination is the register Ri. Note that the value 1000 is not prefixed with any special characters, indicating that it is the (direct or absolute) address of the source operand.
- Figure 2.6 shows an illustration of the direct addressing mode.

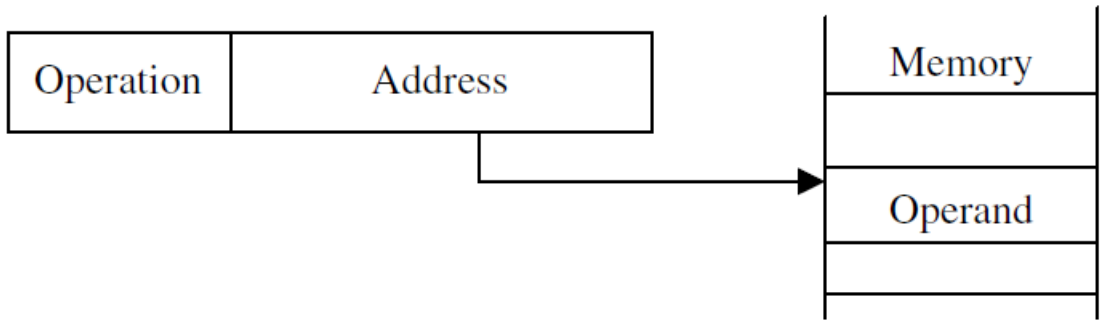


Figure 2.6 Illustration of the direct addressing mode

ADDRESSING MODES

2- Direct (Absolute) Mode

- For example, if the **content of the memory location whose address is 1000 was (2345) at the time when the instruction LOAD 1000, Ri is executed, then the result of executing such instruction is to load the value (2345) into register** Ri.
- **Direct (absolute) addressing mode** provides more **flexibility compared** to the **immediate mode**. However, it requires the explicit inclusion of the operand address in the instruction.
- A more **flexible addressing mechanism** is provided through the use of the **indirect addressing mode**. This is explained below.
- Figure 2.6 shows an illustration of the direct addressing mode.

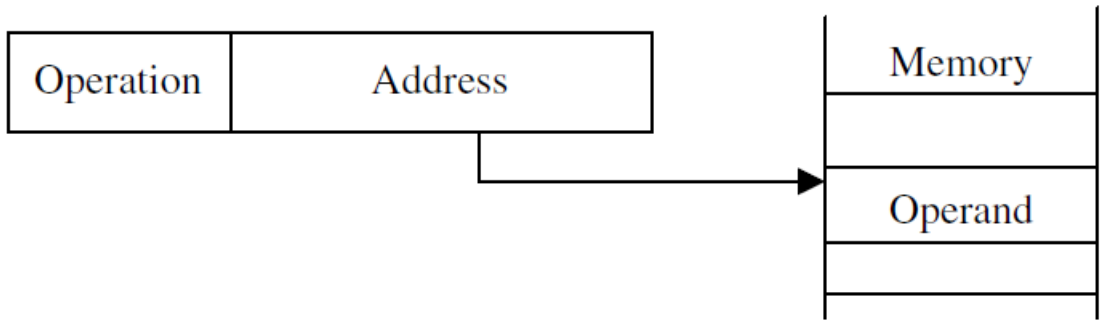


Figure 2.6 Illustration of the direct addressing mode

ADDRESSING MODES

3- Indirect Mode

- In the **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.
- In order to **indicate** the use of **indirection in the instruction,** it is customary to include the name of the register or **the memory location in parentheses.**
- Consider, for example, the instruction
- **LOAD (1000), Ri**
- This instruction has the memory location 1000 enclosed in parentheses, thus **indicating indirection.**
- 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.

- The two types are illustrated in Figure 2.7.

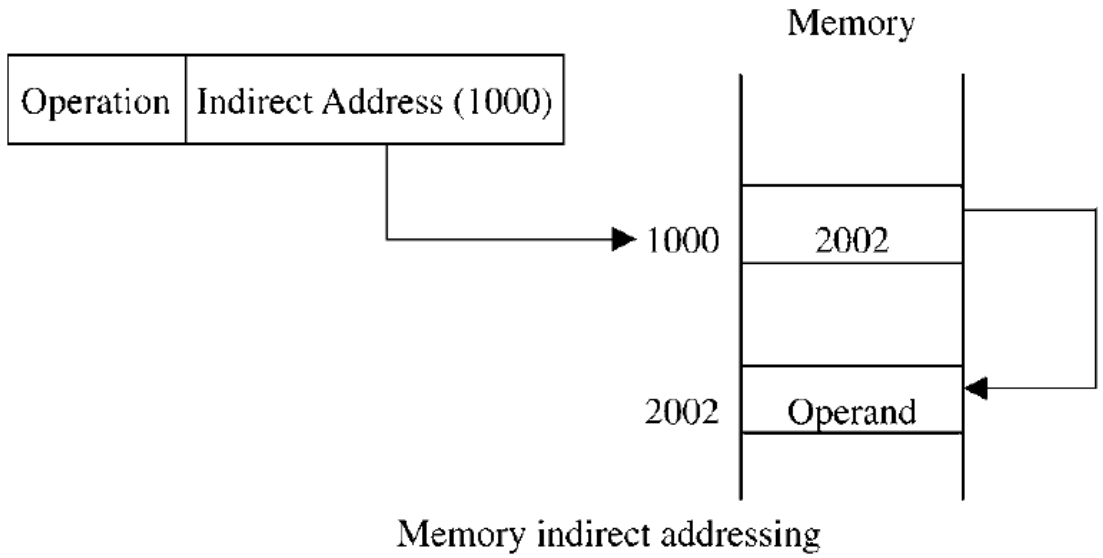


Figure 2.7 Illustration of the indirect addressing mode

ADDRESSING MODES

3- Indirect Mode

- Because **indirection** can be made through either a **register** or a **memory location**, therefore, we can identify two types of **indirect addressing**.
- These are **register indirect addressing**, if a **register** is **used to hold the address** of the **operand**, and **memory indirect addressing**, if **a memory location** is used to **hold the address of the operand**.

- The two types are illustrated in Figure 2.7.

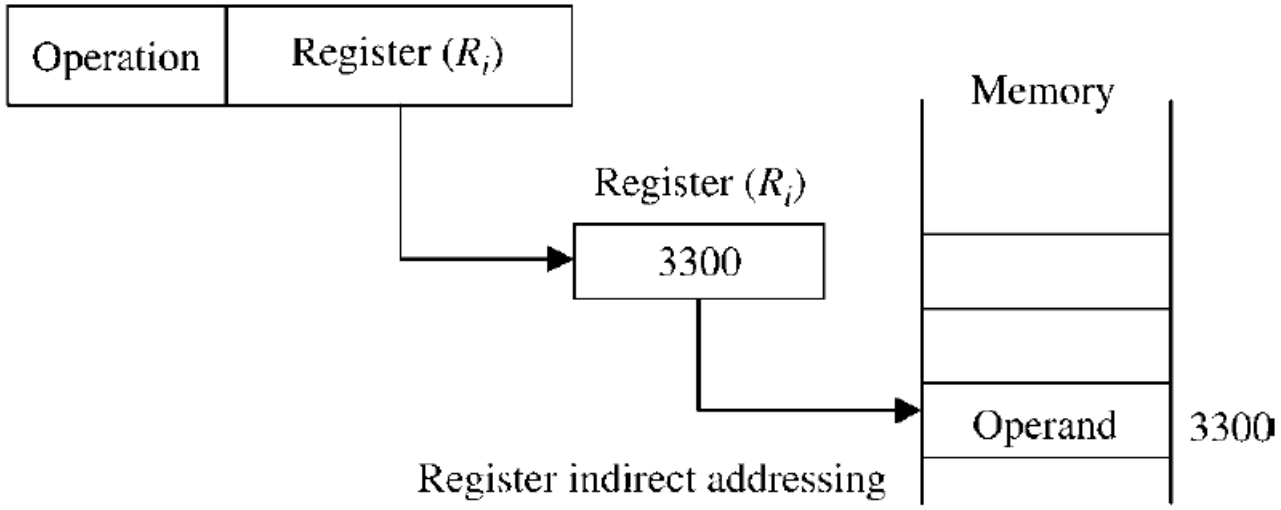


Figure 2.7 Illustration of the indirect addressing mode

ADDRESSING MODES

4- Indexed Mode

- In **this addressing mode**, the **address of the operand** is **obtained** by **adding a constant** to the **content** of a register, called the **index register**.
- Consider, for example, the instruction **LOAD X(R_{ind}), R_i.**
- 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**.
- **Index addressing** is **indicated** in **the instruction** by **including the name of the index register** in **parentheses** and using the symbol **X** to **indicate** the **constant to be added.**

- Figure 2.8 illustrates **indexed addressing**. As can be seen, **indexing requires** an **additional level of complexity over register indirect addressing**.

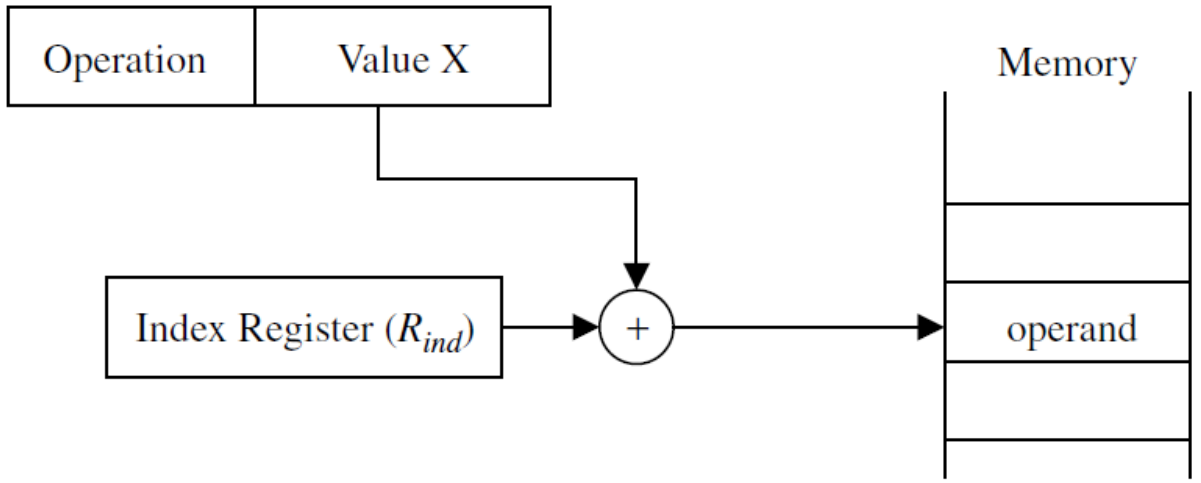


Figure 2.8 Illustration of the indexed addressing mode

ADDRESSING MODES

5- Other Modes

- The **addressing modes presented above represent** the **most commonly used modes** in **most processors**.
- They **provide the programmer** with **sufficient means** to **handle most general programming tasks.**
- However, a **number of other addressing modes** have been **used in a number of processors** to **facilitate execution** of **specific programming tasks**.
- These **additional addressing modes** are **more involved as compared to those presented** above.
- Among these **addressing modes** the **relative, autoincrement,** and the **autodecrement** modes **represent** the **most well-known ones**. These are explained below.

ADDRESSING MODES

5- Other Modes: 1- Relative Mode

- Recall that in indexed addressing, an **index register**, R_{ind}, is used.
- **Relative addressing** is the **same as indexed addressing** except that the **program counter (PC)** replaces the **index register.**
- For example, the instruction
- **LOAD X(PC), R_i**
- **loads** register R_i with the **contents** of the **memory location** whose **address** is the **sum** of the **contents** of the **program counter (PC)** and the **value X.**

- Figure 2.9 illustrates the **relative addressing mode.**

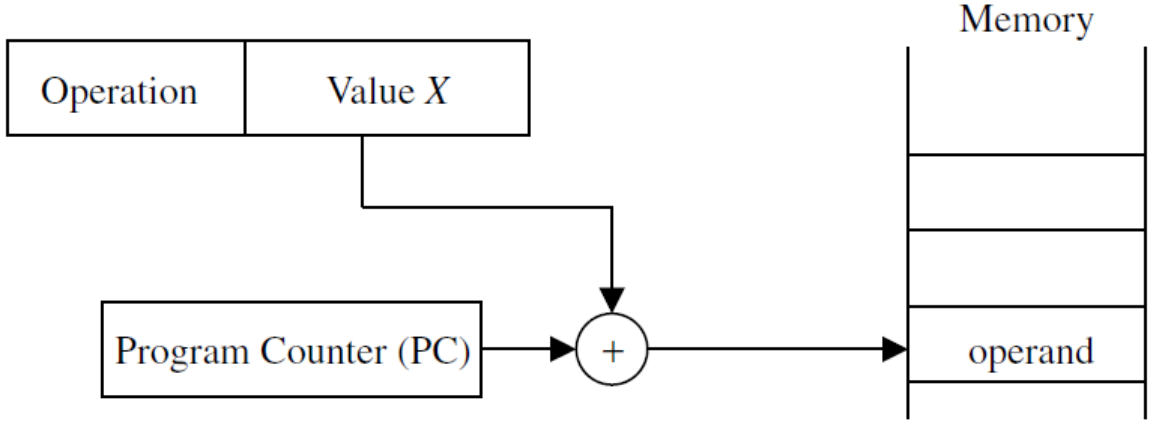


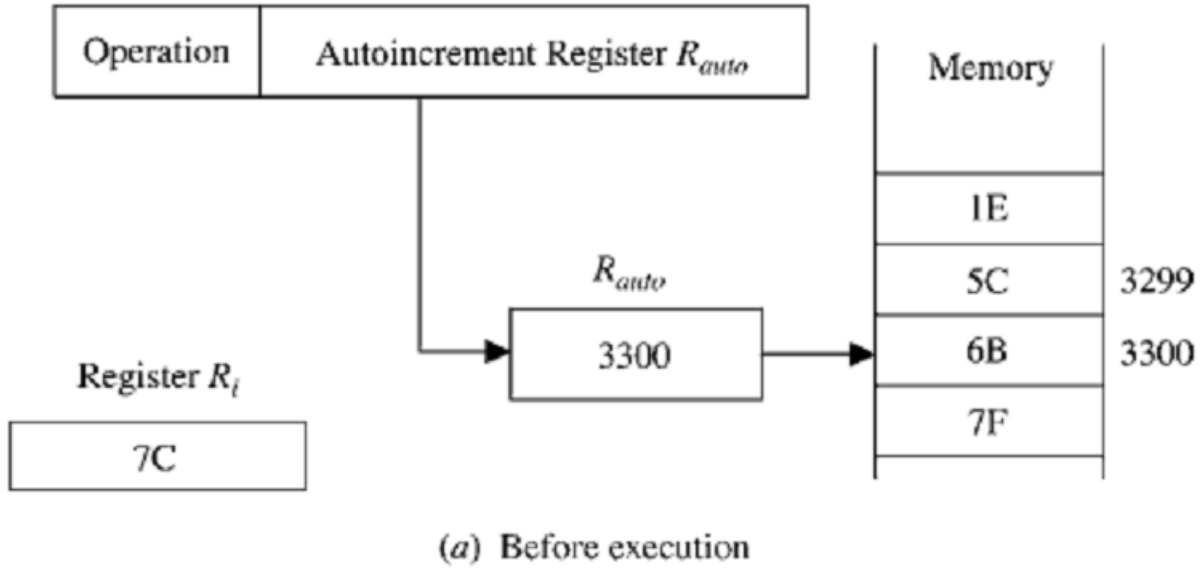
Figure 2.9 Illustration of relative addressing mode

ADDRESSING MODES

5- Other Modes: 2- Autoincrement Mode

- **This addressing mode** 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**.
- However, **with autoincrement**, the **content** of the **autoincrement register** is **automatically incremented after accessing** the **operand**.
- **As before**, **indirection** is **indicated** by including the **autoincrement register** in **parentheses**.

- Figure 2.10 illustrates the **autoincrement addressing mode**.



(a) Before execution

ADDRESSING MODES

5- Other Modes: 2- Autoincrement Mode

- The **automatic increment** of the **register's content** after **accessing** the **operand** is **indicated** by including a (+) after the **parentheses**.
- Consider, for **example, the instruction** **LOAD** $(R_{auto})+, R_i$.
- This instruction **loads register** R_i with the **operand** whose **address** is the **content of register** R_{auto}.
- **After loading** the **operand** into **register** R_i, the content of **register** R_{auto} is **incremented**, **pointing** for example to the **next item** in **a list** of **items**.

- Figure 2.10 illustrates the **autoincrement addressing mode.**

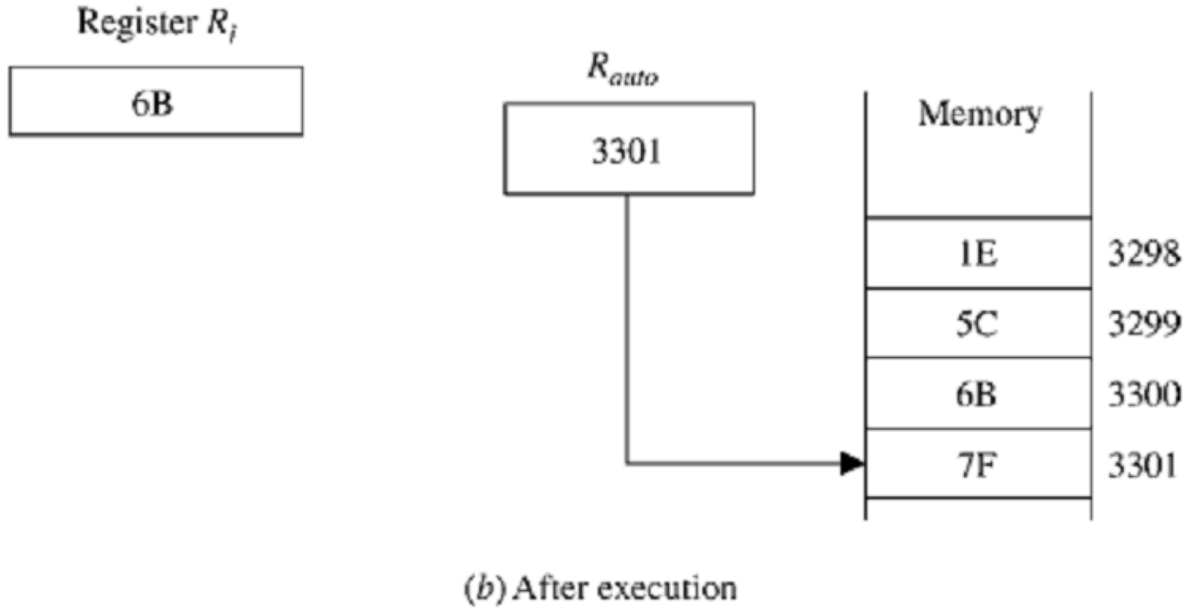


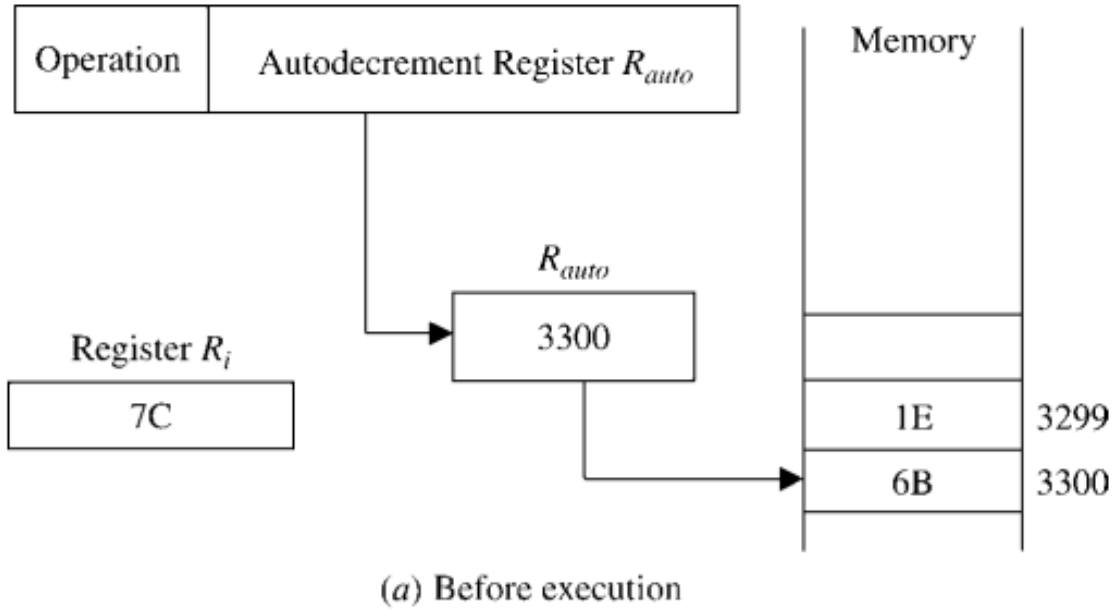
Figure 2.10 Illustration of the autoincrement addressing mode

ADDRESSING MODES

5- Other Modes: 3- Autodecrement Mode

- **Autodecrement Mode** **Similar** to the **autoincrement**, the **autodecrement mode** uses a **register** to **hold the address** of the **operand**.
- However, in this case the **content of the autodecrement register** is **first decremented** and the **new content** is used as the **effective address** of the **operand.**
- In order to reflect the **fact** that the **content** of the **autodecrement register** is **decremented** before **accessing the operand**, a (-) is included **before** the **indirection parentheses**.

- Figure 2.11 illustrates the **autodecrement addressing mode**.

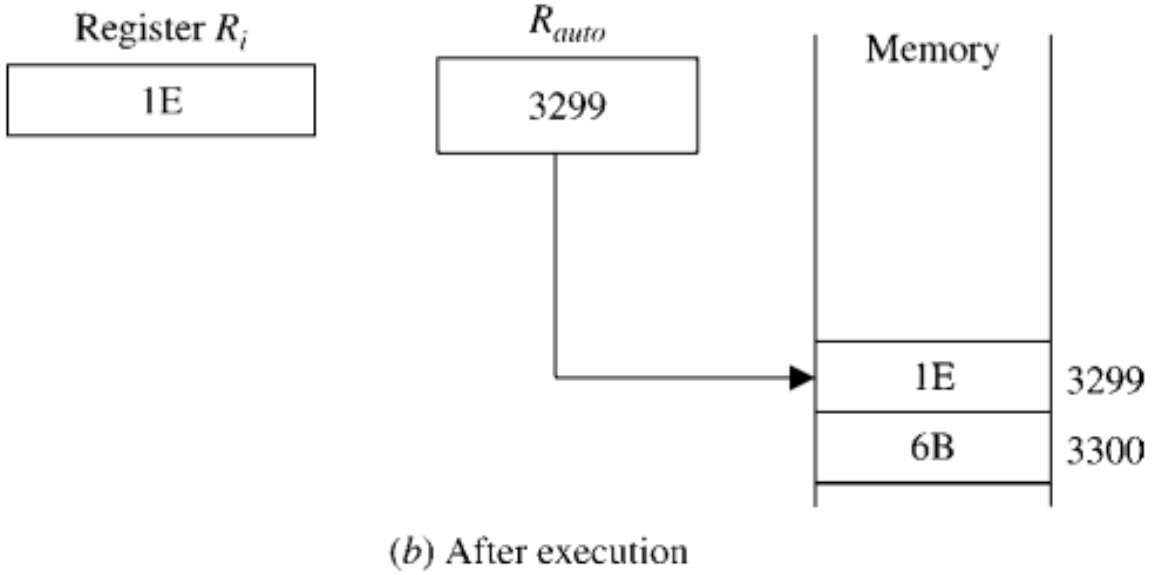


(a) Before execution

ADDRESSING MODES

5- Other Modes: 3- Autodecrement Mode

- Consider, for **example**, the instruction **LOAD – (R_{auto}), Ri.**
- This instruction **decrements** the content of the **register** R_{auto} and then uses the new content as the **effective address** of the **operand** that is to be **loaded** into **register Ri.**
- Figure 2.11 illustrates the **autodecrement addressing mode**.



(*b*) After execution

Figure 2.11 Illustration of the autodecrement addressing mode

ADDRESSING MODES

- The **seven addressing modes** presented above are **summarized in Table 2.2**.
- In each case, the table shows **the name of the addressing mode, its definition**, and **a generic example** illustrating the use of such mode.
- In **presenting** the **different addressing modes** we have used the **load instruction for illustration**.
- However, it should be **understood** that there are **other types** of **instructions** in a **given** machine.
- In the following section we elaborate on the **different types** of **instructions** that **typically constitute** the **instruction set of a given machine**.

TABLE 2.2 Summary of Addressing Modes

Addressing mode	Definition	Example	Operation
Immediate	Value of operand is included in the instruction	*load* #1000, R_i	$R_i \leftarrow 1000$
Direct (Absolute)	Address of operand is included in the instruction	*load* 1000, R_i	$R_i \leftarrow M[1000]$
Register indirect	Operand is in a memory location whose address is in the register specified in the instruction	*load* (R_j), R_i	$R_i \leftarrow M[R_j]$
Memory indirect	Operand is in a memory location whose address is in the memory location specified in the instruction	*load* (1000), R_i	$R_i \leftarrow M[1000]$
Indexed	Address of operand is the sum of an index value and the contents of an index register	*load* $X(R_{ind})$, R_i	$R_i \leftarrow M[R_{ind} + X]$
Relative	Address of operand is the sum of an index value and the contents of the program counter	*load* $X(PC)$, R_i	$R_i \leftarrow M[PC + X]$
Autoincrement	Address of operand is in a register whose value is incremented after fetching the operand	*load* $(R_{auto})+$, R_i	$R_i \leftarrow M[R_{auto}]$ $R_{auto} \leftarrow R_{auto} + 1$
Autodecrement	Address of operand is in a register whose value is decremented before fetching the operand	*load* $-(R_{auto})$, R_i	$R_{auto} \leftarrow R_{auto} - 1$ $R_i \leftarrow M[R_{auto}]$

Thank you