Chapter 2

Instruction Set Architecture & Design

2.1 Memory Locations and Operations

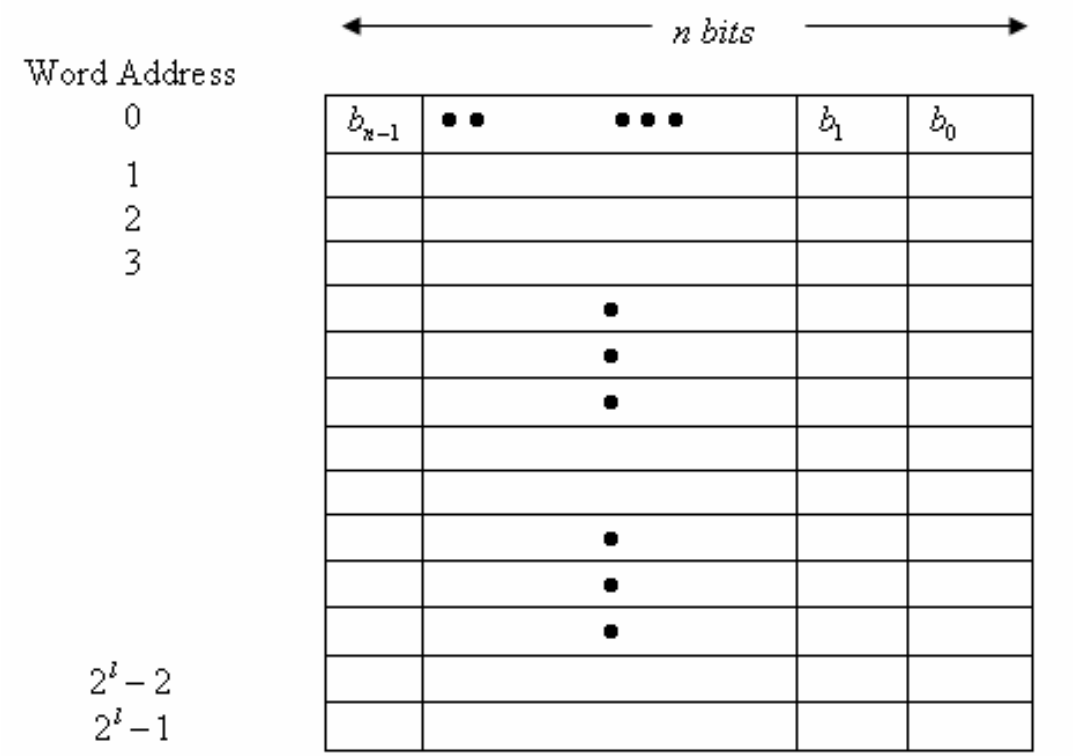
- The (main) memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary *digit* (*bit*), having value of 1 or 0.
- These cells are organized in the form of groups of fixed number, say *n*, of cells that can be dealt with as an atomic entity. An entity consisting of 8-bit is called a *byte*.
- the entity consisting of *n*-bit 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.

2.1 Memory Locations and Operations

- 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 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: $l = \log_2 M$.

2.1 Memory Locations and Operations

- 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 $M = 2^l$.



2.1 Memory Locations and Operations

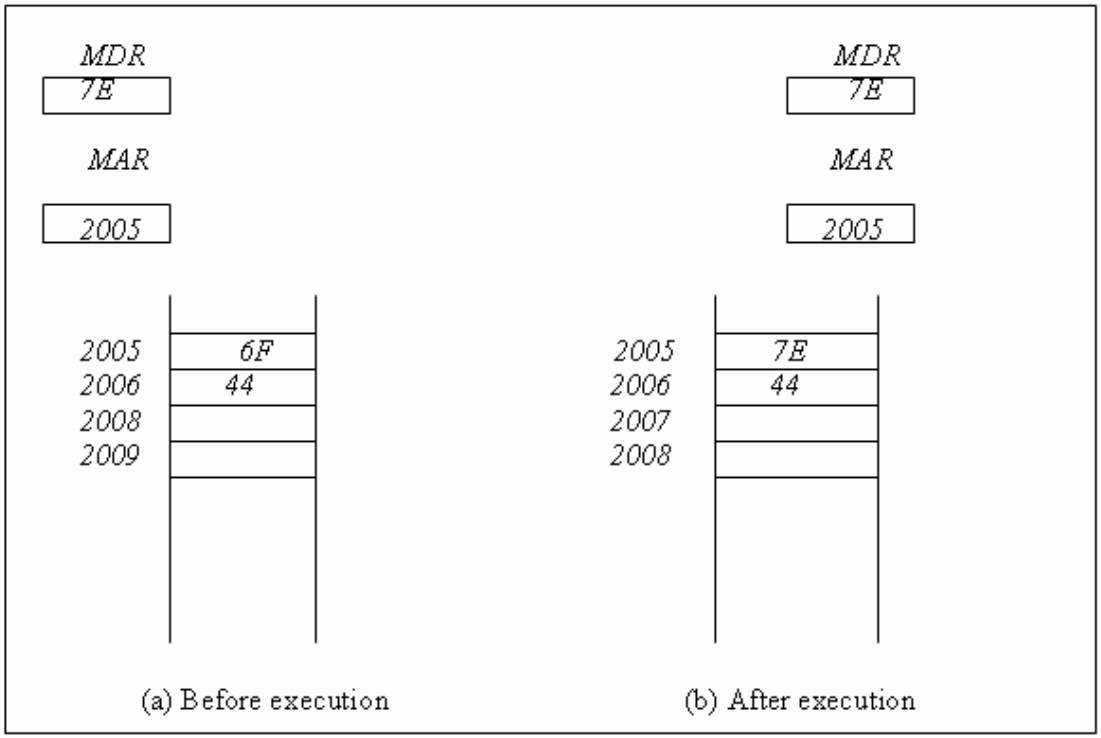
- During a memory write operation a word is stored into a memory location whose address is specified.
- During a memory read operation a word is read from a memory location whose address is specified.
- Typically, memory read and memory write operations are performed by the *central processing unit*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

2.1 Memory Locations and Operations

- Three basic steps are needed in order for the CPU to perform a write operation into a specified memory location:
  - 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*).
  - 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*).
  - 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.

2.1 Memory Locations and Operations

- This example illustrates the operation of writing the word given by 7E (in hex) into the memory location whose address is 2005.

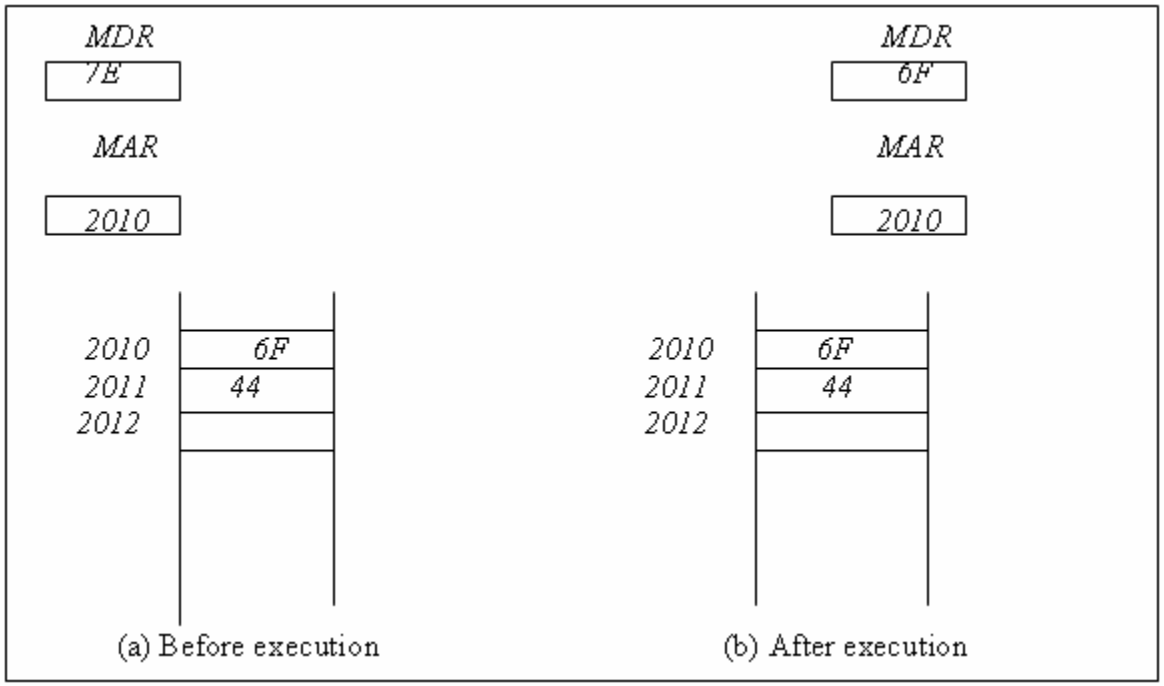


2.1 Memory Locations and Operations

- Three basic steps are needed in order to perform a memory read operation:
  - The address of the location from which the word is to be read is loaded into the MAR.
  - 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.
  - 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.

2.1 Memory Locations and Operations

- This example illustrates the operation of reading the word stored in memory location whose address is 2010.

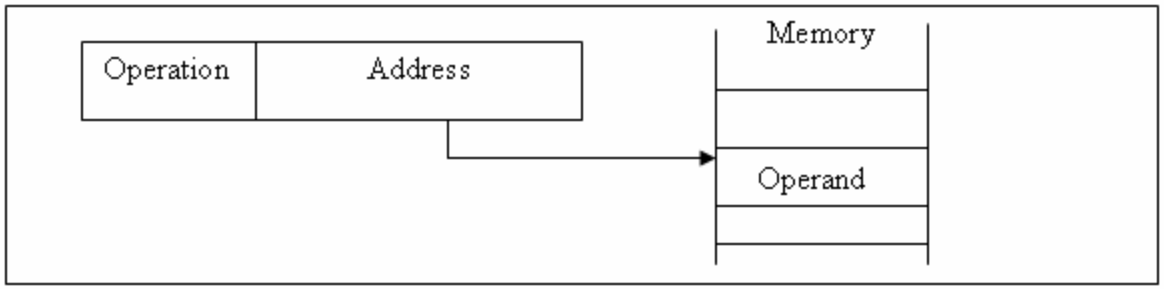


2.2 Addressing Modes

- Immediate Mode
  - The value of the operand is (immediately) available in the instruction itself: $LOAD\ \#1000, R_i$
  - 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), its customary to prefix the operand by the special character (#).
  - 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 operand.

2.2 Addressing Modes

- Direct (Absolute) Mode
  - The address of the memory location that holds the operand is included in the instruction: $LOAD\ 1000, R_i$
  - The source operand is the value stored in memory location whose address is 1000, and the destination is the register R_i .



2.2 Addressing Modes

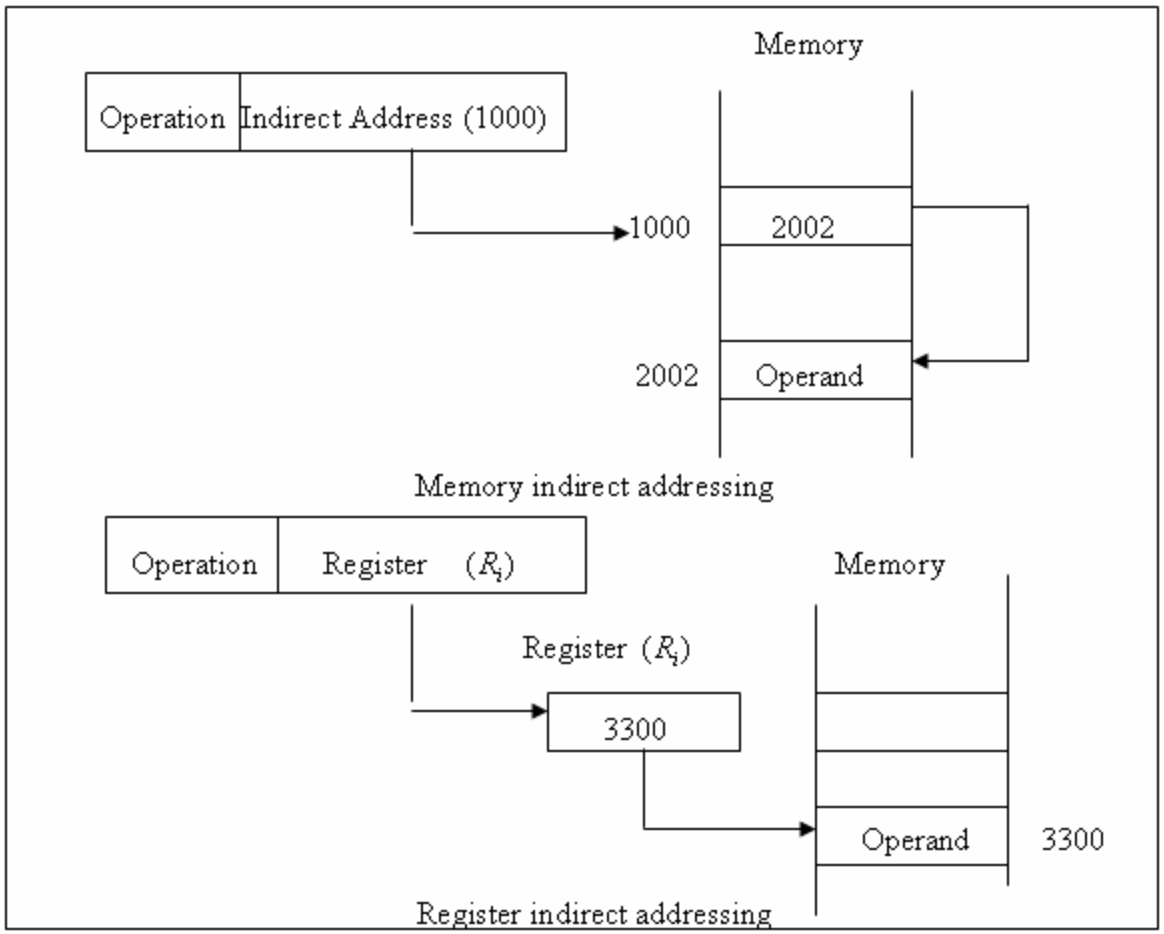
- Indirect Mode
  - What is included in the instruction is not the address of the operand, but rather a name of a register or a memory location that holds the (effective) address of the operand.
  - In order to indicate the use of indirection in the instruction, it is customary to include the name of the register or the memory location in parentheses: $LOAD\ (1000), R_i$

2.2 Addressing Modes

- Indirect Mode
  - we can identify two types of indirect addressing:
    - *register indirect addressing*: if a register is used to hold the address of the operand.
    - *memory indirect addressing*: if a memory location is used to hold the address of the operand.

2.2 Addressing Modes

- Indirect Mode



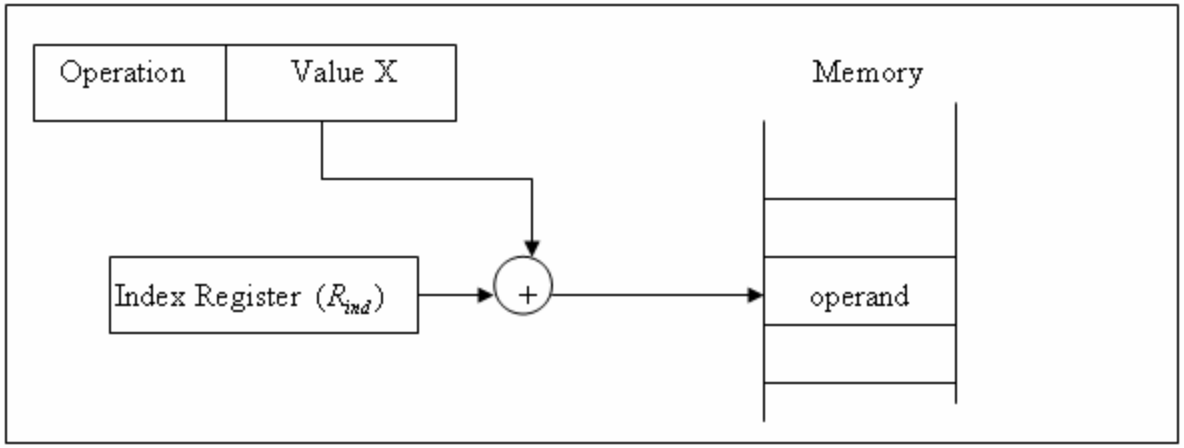
2.2 Addressing Modes

- Indexed Mode
  - The address of the operand is obtained by adding a constant to the content of a register, called the *index register*:

    $LOAD\ \ X(R_{ind}), R_i$
  - 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.
  - Indexing requires an additional level of complexity over register indirect addressing.

2.2 Addressing Modes

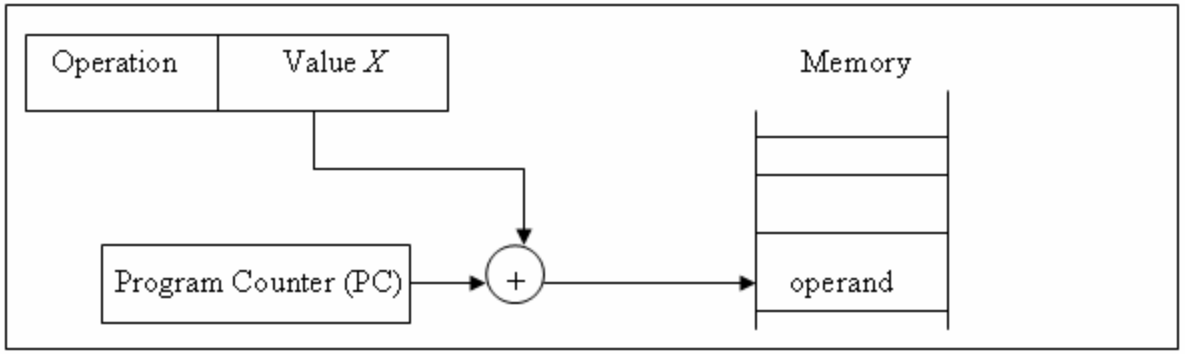
- Indexed Mode



2.2 Addressing Modes

- Other Modes
  - Relative mode
    - Relative addressing is the same as indexed addressing except that the program counter (PC) replaces the index register:

      $LOAD\ X(PC), R_i$



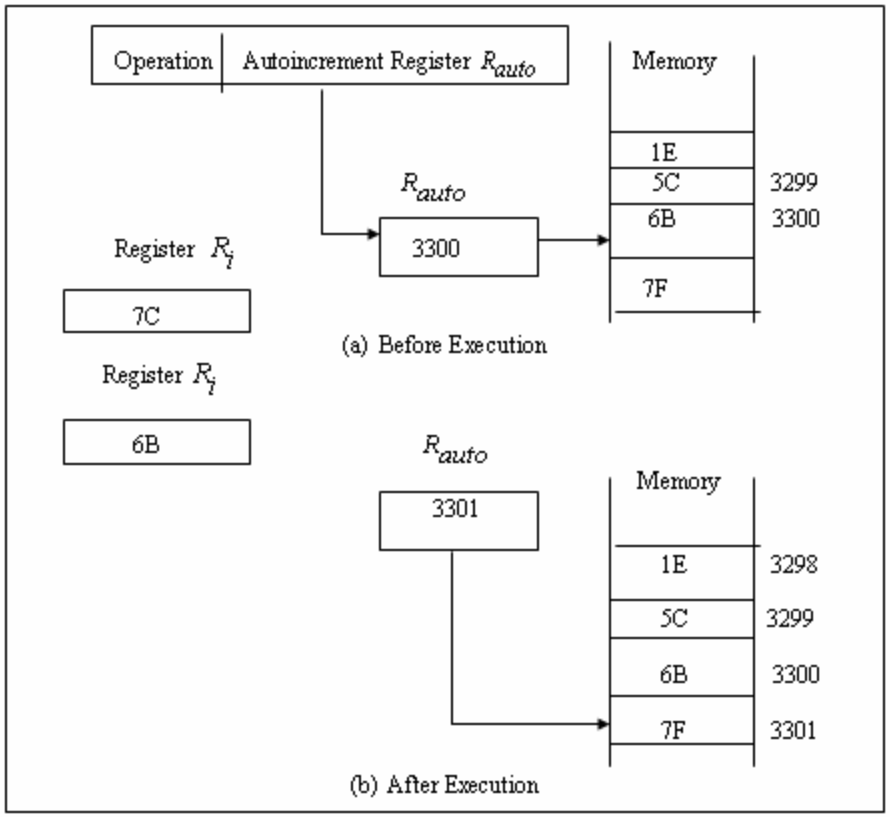
2.2 Addressing Modes

- Other Modes
  - 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.
    - However, with autoincrement, the content of the autoincrement register is automatically incremented after accessing the operand:

$$LOAD \ \ (R_{auto})+, R_i$$

2.2 Addressing Modes

- Other Modes
  - Autoincrement mode

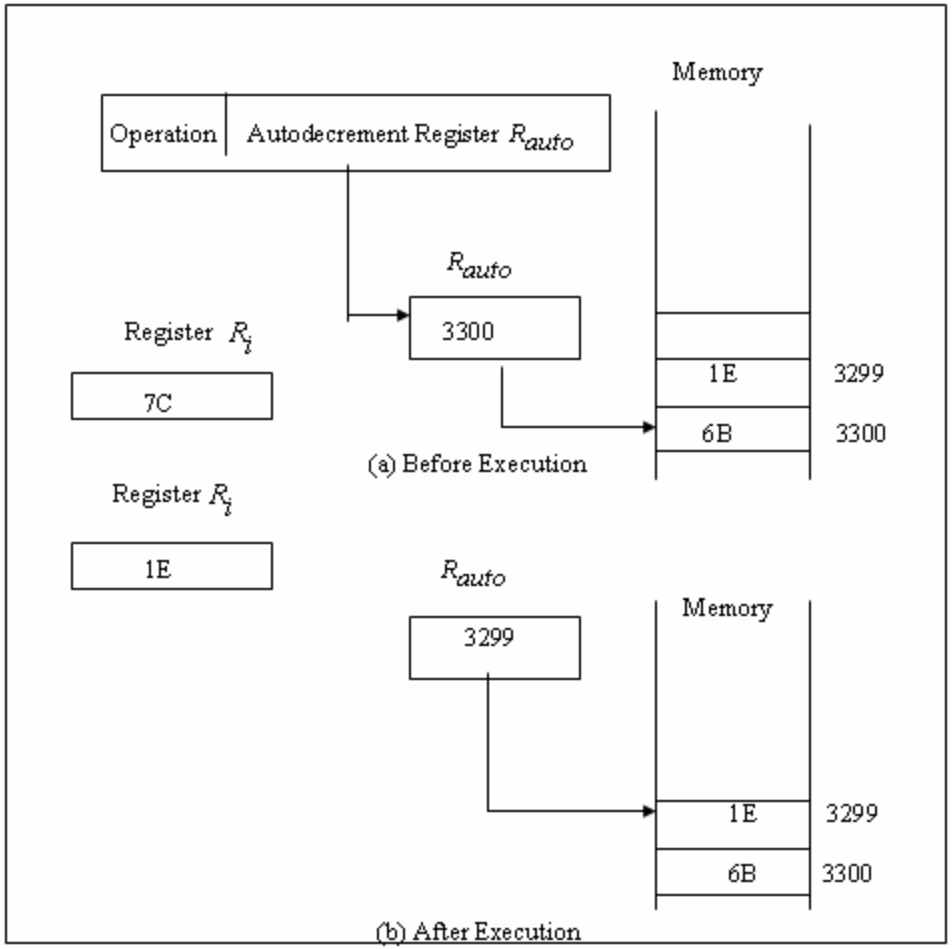


2.2 Addressing Modes

- Other Modes
  - Autodecrement mode
    - The autodecrement mode uses a register to hold the address of the operand.
    - However, the content of the autodecrement register is first decremented and the new content is used as the effective address of the operand: $LOAD \quad -(R_{auto}), R_i$

2.2 Addressing Modes

- Other Modes
  - Autodecrement mode



2.2 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	$load\ -(R_{auto}), R_i$	$R_{auto} \leftarrow R_{auto} - 1$ $R_i \leftarrow M[R_{auto}]$

2.3 Instruction Types

- Data Movement Instructions
  - Data movement instructions are used to move data among the different registers in the CPU.
  - A simple register to register movement of data can be made through the instruction *MOVE* R_i, R_j
  - Date movement instructions include those used to move data to (from) registers from (to) memory.
  - These instructions are usually referred to as the load and store instructions, respectively.
  - Examples of the two instructions are:
    - LOAD 25838, R_j
    - STORE R_i ,1024

2.3 Instruction Types

- Data Movement Instructions

Data Movement Operation	Meaning
MOVE	Move data (a word or a block) from a given source (a register or a memory) to a given destination
LOAD	Load data from memory to a register
STORE	Store data into memory from a register
PUSH	Store data from a register to stack
POP	Retrieve data from stack into a register

2.3 Instruction Types

- Arithmetic and Logical Instructions
  - Arithmetic and logical instructions are those used to perform arithmetic and logical manipulation of registers and memory contents.
  - Examples of arithmetic instructions include the *ADD* and *SUBTRACT* instructions:

$$\begin{aligned} ADD \quad & R_1, R_2, R_0 \\ SUBTRACT \quad & R_1, R_2, R_0 \end{aligned}$$

  - In addition, some machines have MULTIPLY and DIVIDE instructions.
  - These two instructions are expensive to implement and could be substituted by the use of repeated addition or repeated subtraction.

2.3 Instruction Types

- Arithmetic and Logical Instructions:

Arithmetic operations	Meaning
ADD	Perform the arithmetic sum of two operands
SUBTRACT	Perform the arithmetic difference of two operands
MULTIPLY	Perform the product of two operands
DIVIDE	Perform the division of two operands
INCREMENT	Add one to the contents of a register
DECREMENT	Subtract one from the contents of a register

2.3 Instruction Types

- Arithmetic and Logical Instructions

Logical Operation	Meaning
AND	Perform the logical ANDing of two operands
OR	Perform the logical ORing of two operands
EXOR	Perform the XORing of two operands
NOT	Perform the complement of an operand
COMPARE	Perform logical comparison of two operands and set flag accordingly
SHIFT	Perform logical shift (right or left) of the content of a register
ROTATE	Perform logical shift (right or left) with wraparound of the content of a register

2.3 Instruction Types

- Sequencing Instructions
  - Control (Sequencing) instructions are used to change the sequence in which instructions are executed.
  - A common characteristic among these instructions is that their execution changes the program counter (*PC*) value.
  - The change made in the *PC* value can be unconditional in the *unconditional* branching or the jump instructions.
    - The earlier value of the *PC* is lost and execution of the program starts at a new value specified by the instruction.
  - The change made in the *PC* by the branching instruction can be *conditional* based on the value of a specific *flag.*
    - These flags represent the individual bits of a specific register, called the *CONDITION CODE (CC) REGISTER.*
    - The values of flags are set based on the results of executing different instructions.

2.3 Instruction Types

- Sequencing Instructions
  - Examples of condition flags

Flag name	The meaning
Negative (N)	Set to 1 if the result of the most recent operation is negative, it is 0 otherwise
Zero (Z)	Set to 1 if the result of the most recent operation is 0, it is 0 otherwise
Overflow (V)	Set to 1 if the result of the most recent operation causes an overflow, it is 0 otherwise
Carry (C)	Set to 1 if the most recent operation results in a carry, it is 0 otherwise

2.3 Instruction Types

- Sequencing Instructions
  - Some transfer of control operations

Transfer of control operation	Meaning
BRANCH-IF-CONDITION	Transfer of control to a new address if condition is true
JUMP	Unconditional transfer of control
CALL	Transfer of control to a subroutine
RETURN	Transfer of control to the caller routine

2.3 Instruction Types

- Input/Output Instructions
  - Input and output instructions (I/O instructions) are used to transfer data between the computer and peripheral devices.
  - The two basic I/O instructions used are the *INPUT* and *OUTPUT* instructions.
  - For example: the *INPUT* instruction is used to transfer data from an input device to the processor.

2.4 Summary

- The main issues related to instruction set design and its characteristics were considered.
- A model of the main memory was presented in which the memory is abstracted as a sequence of cells, each is capable of storing *n* bits.
- A number of addressing modes were presented.
  - These include immediate, direct, indirect, indexed, autoincrement, and autodecrement.
- A discussion on instruction types was also presented, which include data movement, arithmetic/logical, instruction sequencing, and Input/Output.