ADDRESSING MODES

• The different ways in which the location of an operand is specified in an instruction are referred to as **Addressing Modes** (Table 2.1).

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri) (LOC)	EA = [Ri] EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ri] + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	−(Ri)	Decrement Ri; EA = [Ri]

EA = effective address
Value = a signed number

IMPLEMENTATION OF VARIABLE AND CONSTANTS

• **Variable** is represented by allocating a memory-location to hold its value.
• Thus, the value can be changed as needed using appropriate instructions.
• There are 2 accessing modes to access the variables:

1) Register Mode
2) Absolute Mode

Register Mode

• The operand is the contents of a register.
• The name (or address

Click to add title

➢ **ADDITIONAL ADDRESSING MODES**

1) **Auto Increment Mode**

- Effective-address of operand is contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
- Implicitly, the increment amount is 1.
- This mode is denoted as
    - (Ri)+ ;where Ri=pointer-register.

1) **Auto Decrement Mode**

- The contents of a register specified in the instruction are first automatically decremented and are then used as the effective-address of the operand.
- This mode is denoted as
    - -(Ri) ;where Ri=pointer-register.
- These 2 modes can be used together to implement an important data structure called a stack.

Click to add title

Ex (1)

Add $(R_2)+, R_0$

Initially

R_2	1040

R_0	10

103C	
1040	10
1044	20
1048	

$(1040) = 10$

$$\text{Add} \quad \frac{(R_0) = 10}{R_0 = 20\text{H}}$$

After execution

R_2	1044

R_0	20H

(29)

Ex 27 Add $-(R_2), R_0$

Initially

R_2	1040

R_0	10

103C	08
1040	10
1044	:
1048	:

$(R_0) = 10$

$(103C) = 08$

Add

$R_0 = 18$

After execution

R_2	103C

R_0	18

Registers R1 and R2 of a computer contains the decimal values 1200 and 4600. What is the effective- address of the memory operand in each of the following instructions?

(a) Load 20(R1), R5

(b) Move #3000,R5

(c) Store R5,30(R1,R2)

(d) Add -(R2),R5

(e) Subtract (R1)+,R5

Rodrigo A. Obando

Basic Input/Output Operations

ASSEMBLY LANGUAGE

- We generally use symbolic-names to write a program.
- A complete set of symbolic-names and rules for their use constitute an **Assembly Language**.
- The set of rules for using the mnemonics in the specification of complete instructions and programs is called the **Syntax** of the language.
- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an **Assembler.**
- The user program in its original alphanumeric text formal is called a **Source Program**, and the assembled machine language program is called an **Object Program**.

For example:

MOVE R0,SUM ;The term MOVE represents OP code for operation performed by instruction.

ADD #5,R3 ;Adds number 5 to contents of register R3 & puts the result back into registerR3.

ASSEMBLER DIRECTIVES

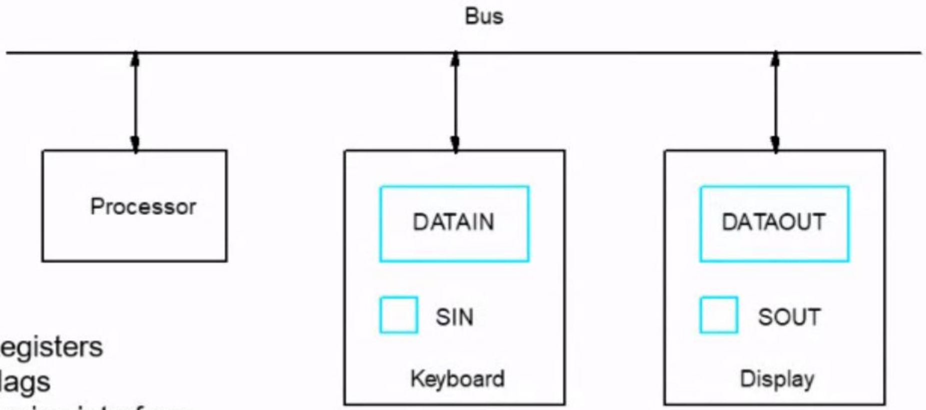
- **Directives** are the assembler commands to the assembler concerning the program being assembled.
- These commands are not translated into machine opcode in the object-program.

	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that	START	MOVE	N,R1

I/O

- The data on which the instructions operate are not necessarily already stored in memory.
- Data need to be transferred between processor and outside world (disk, keyboard, etc.)
- I/O operations are essential, the way they are performed can have a significant effect on the performance of the computer.

Program-Controlled I/O Example



- Registers
- Flags
- Device interface

Figure 2.19 Bus connection for processor, keyboard, and display

Program-Controlled I/O Example

- Machine instructions that can check the state of the status flags and transfer data:

```
READWAIT  Branch to READWAIT if SIN = 0
          Input from DATAIN to R1

WRITEWAIT Branch to WRITEWAIT if SOUT = 0
          Output from R1 to DATAOUT
```

ADDRESSING MODES

• The different ways in which the location of an operand is specified in an instruction are referred to as **Addressing Modes** (Table 2.1).

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

IMPLEMENTATION OF VARIABLE AND CONSTANTS

• **Variable** is represented by allocating a memory-location to hold its value.
• Thus, the value can be changed as needed using appropriate instructions.
• There are 2 accessing modes to access the variables:

1) Register Mode
2) Absolute Mode

Register Mode

• The operand is the contents of a register.
• The name (or address