RIPHAH INTERNATIONAL UNIVERSITY, LAHORE

Computer Organization and Assembly Language

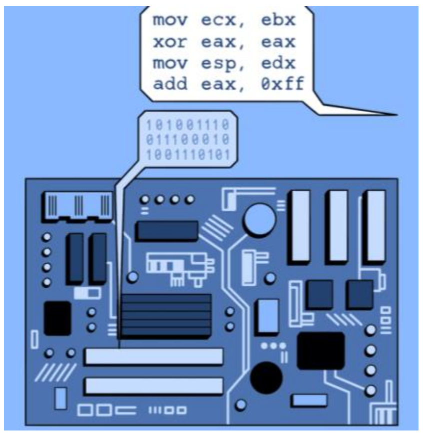
Lab

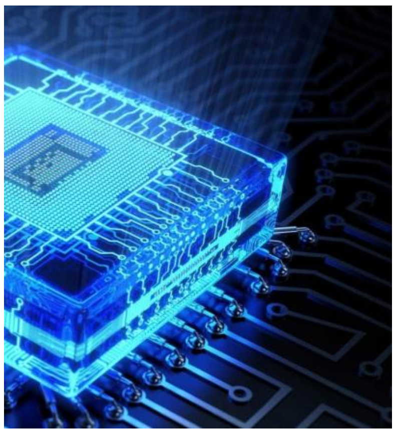
Fall-2024

Lab 2 Manual

VVM Programming

Instructor: Engr. Amna Bibi
Office: Faculty Offices Physics & Math 3rd Floor, Block A
Office Hours: Tuesday 10:00-12:00





<u>Lab # 2</u>

<u>INTRODUCTION TO VVM PROGRAMMING</u>

Objective:

Learn VVM Programming and simulate VVM program.

VVM Programming

VVM has its own simple Programming Language which supports such operations as conditional and unconditional branching, addition and subtraction, and input and output, among others. The language allows the student to create reasonably complex programs, and yet the language is quite easy to learn and to understand -- only eleven unique operations are provided. When VVM programs go awry, as in the case of endless loops or data overflows, VVM (virtual) system errors are triggered before the user's eyes.

VVM programs can be written in Machine Language, in Assembly Language, or in a combination of both.

The Machine Language format is represented in decimal values, so there is no need for the student user to interpret long binary machine codes.

- In the **Machine Language format**, each instruction is a **three-digit integer where the first digit** specifies the **operation code (op code),** and the **remaining two digits** represent the **operand.**
- In the **Assembly Language format**, the operation code is replaced by a **three-character mnemonic code.** The two-digit operand usually represents a memory address.

The sample program below is shown in both formats.

Following the automatic syntax validation process, VVM programs are converted to machine language format and loaded into the 100 data-word virtual RAM which is fully visible to the user during program execution.

The Language Instructions

The eleven operations of the VVM Language are described below. The Machine Language codes are shown in parentheses, while the Assembly Language version is in square brackets.

1. Load Accumulator (5nn) [LDA nn]

The content of RAM address nn is copied to the Accumulator Register, replacing the current content of the register. The content of RAM address nn remains unchanged. The Program Counter Register is incremented by one.

2. Store Accumulator (3nn) [STO nn] (or [STA nn])

The content of the Accumulator Register is copied to RAM address nn, replacing the current content of the address. The content of the Accumulator Register remains unchanged. The Program Counter Register is incremented by one.

3. Add (1nn) [ADD nn]

The content of RAM address nn is added to the content of the Accumulator Register, replacing the current content of the register. The content of RAM address nn remains unchanged. The Program Counter Register is incremented by one.

4. Subtract (2nn) [SUB nn]

The content of RAM address nn is subtracted from the content of the Accumulator Register, replacing the current content of the register. The content of RAM address nn remains unchanged. The Program Counter Register is incremented by one.

5. Input (901) [IN] (or [INP])

A value input by the user is stored in the Accumulator Register, replacing the current content of the register. The Program Counter Register is incremented by one.

6. Output (902) [OUT] (or [PRN])

The content of the Accumulator Register is output to the user. The current content of the register remains unchanged. The Program Counter Register is incremented by one.

7. Halt (0nn) [HLT] (or [COB])

Program execution is terminated. The operand value nn is ignored in this instruction and can be omitted in the Assembly Language format.

8. Branch if Zero (7nn) [BRZ nn]

This is a conditional branch instruction. If the value in the Accumulator Register is zero, then the Program Counter Register is replaced by the operand value nn. The result is that the next instruction to be executed will be taken from address nn rather than from the next sequential address. Otherwise (Accumulator <> 0), the Program Counter Register is incremented by one, and the next sequential instruction is executed.

9. Branch if Positive or Zero (8nn) [BRP nn]

This is a conditional branch instruction. If the value in the Accumulator Register is positive or zero, then the Program Counter Register is replaced by the operand value nn. The result is that the next instruction to be executed will be taken from address nn rather than from the next sequential address. Otherwise (Accumulator < 0), the Program Counter Register is incremented by one, and the next sequential instruction is executed.

10. Branch (6nn) [BR nn] (or [BRU nn] or [JMP nn])

This is an unconditional branch instruction. The current value of the Program Counter Register is replaced by the operand value nn. The result is that the next instruction to be executed will be taken from address nn rather than from the next sequential address. The value of the Program Counter Register is not incremented with this instruction.

11. No Operation (4nn) [NOP] (or [NUL])

This instruction does nothing other than increment the Program Counter Register by one. The operand value nn is ignored in this instruction and can be omitted in the Assembly Language format. (This instruction is unique to the VVM and is not part of the original Little Man Model.)

Embedding Data in Programs

Data values used by a program can be loaded into memory along with the program. In Machine or Assembly Language form simply use the format "snnn" where s is an optional sign, and nnn is the three-digit data value. In Assembly Language, you can specify "DAT snnn" for clarity.

The VVM Load Directive

By default, VVM programs are loaded into sequential memory addresses starting with address 00. VVM programs can include an additional load directive which overrides this default, indicating the location in which certain instructions and data should be loaded in memory. The syntax of the Load Directive is "*nn" where nn represents an address in memory. When this directive is encountered in a program, subsequent program elements are loaded in sequential addresses beginning with address nn.

Program#1

Simple conditional structure using “brp” & “br” instruction.

```
in      Input A

sto 98   Store A

in      Input B

sto 99   Store B

lda 98   Load value of A

sub 99   Subtract B from A

brp 11   If A >= B, branch to 11 A is < B Find difference

lda 98   Load value of A

sub 99   Subtract value of B

sto 97   Store C

br 14    Jump to 14

lda 98   [11] Load A (A is >= B)

add 99   Add B
```

Equivalent BASIC program:

```
INPUT A
INPUT B
IF A >= B THEN
C = A + B
```

```
sta 97   Store C

out [14] Print result

hlt      Done
```

LAB TASKS

1. Take any integer as input, if the number is greater than 5 print it

   If a>5, print a

   Else if a=0,then Halt

   Else if a<5,then halt

2. Take two numbers as input and print the larger number.