CPU 3: Machine Code and Assembly Language

An assembly language is a low-level programming language. To use a **low-level language** a programmer must have a knowledge of the architecture of the computer.

Programs written in assembly languages are translated to machine code by an **assembler**.

An example of an assembly language instruction is: LDA #24.

- The meaning of this instruction is: load (write) 24 in the accumulator.
- LDA is called the **operator**
- #24 is called the **operand**
- LDA is called the **opcode** (short for 'operation code').
- LDA is also called a **mnemonic** i.e. it reminds you of the meaning of the operation.
- Example of an assembly language program:

```
rep:  LDA #14    ; load the accumulator with 14
      ADD #31    ; add 31 to the contents of the accumulator
      JZE rep    ; jump to 'rep' if accumulator contains zero
      HLT        ; stop program execution
```

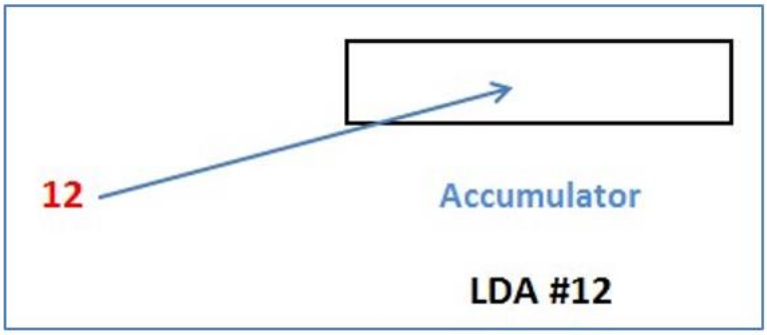
  In the above program 'rep' is called a **label**.

- The **instruction set** is the set of all instructions performed by a particular CPU.

Memory address modes tell you how the CPU accesses data.

Immediate Addressing Mode

Immediate addressing means that the data is found inside the instruction itself. Example: 'LDA #12' means: load 12 into A (the accumulator).



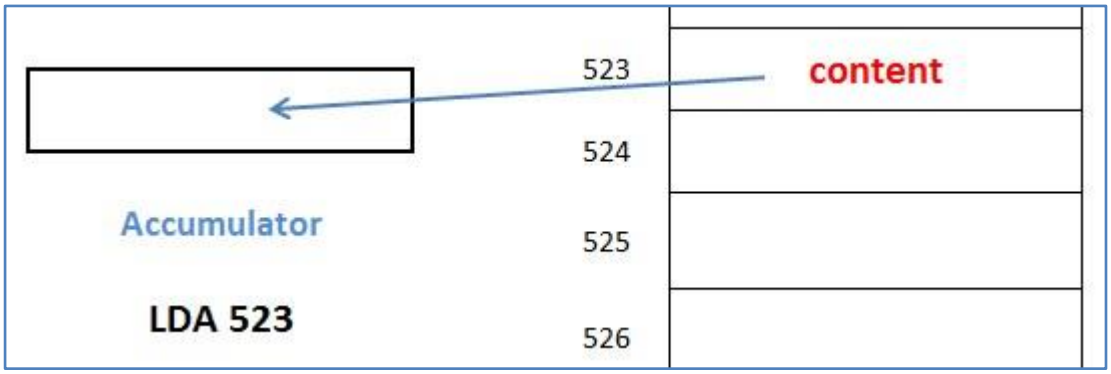
Immediate addressing

Advantage: very fast.
Disadvantage: the value in the instruction never changes.

Direct Addressing Mode

Direct addressing means the code refers directly to a location in memory.

Example: 'LDA 523' means: load the contents of address 523 into A.



Direct Addressing

Advantage: fast (but not as fast as immediate addressing).
Disadvantage: the program cannot be relocated (i.e. moved about in main memory).

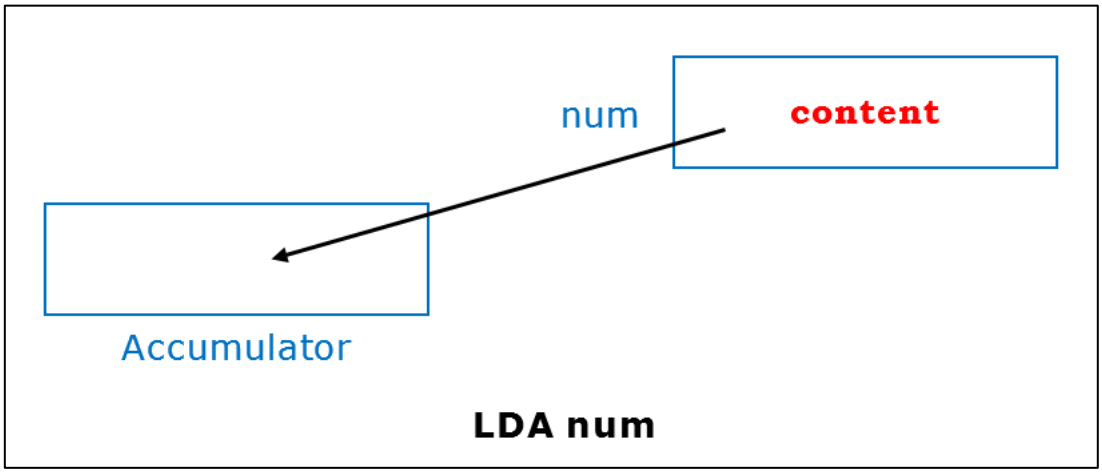
Symbolic Addressing Mode

Symbolic addressing means that the instruction instead of referring to an address refers to a name (symbol).

Example: 'LDA num' means: load the contents of num into A.

Advantages:

- The program is re-locatable in memory.
- Using symbols makes the software much more understandable.



Symbolic Addressing

Assembly Language Types of Instructions

Data transfer instructions:

- LDA x ; Load accumulator A with x
- STA x ; Store contents of accumulator A in x

Arithmetic, Logical and Shift instructions:

- ADD x ; Add contents of accumulator A with x
- SUB x ; Subtract contents of x from accumulator A (A minus x)
- MUL x ; Multiply contents of accumulator A with x
- DIV x ; Divide contents of accumulator A by x
- AND x ; Logical AND the contents of accumulator A with x (bitwise operation)

  Example AND 14 (let us assume that the accumulator has 01100110). Therefore 01100110 AND 00001110 (bitwise operation) gives 00000110 (work the AND operation on each set of corresponding bits).
- ORA x ; Logical OR the contents of accumulator A with x (bitwise operation)

For the same two numbers seen above, the calculation ORA will give 01101110.

- NOT ; Logical NOT the contents of accumulator A (no operand).
- SHL ; Logical shift contents of accumulator A to the left, introducing a 0 in the vacated bit (no operand and bitwise operation).
- SHR ; Logical shift contents of accumulator A to the right, introducing a 0 in the vacated bit (no operand and bitwise operation).

Transfer of control instructions:

- JMP x ; Jump to the instruction pointed to by the label x (**unconditional jump**)
- JZE x ; Jump to the instruction pointed to by the label x if contents of accumulator is zero (this is called a **conditional jump** i.e. a jump to another line of the program occurs only if a condition is true).
- JNZ x ; Jump to the instruction pointed to by the label x if contents of accumulator is not zero (a conditional jump).

Other instructions:

- HLT ; End of program

Example of an Assembly Language Program

(taken from the 2016 Secondary Education Certificate Computing paper 2A)

```
1                LDA #3      ; Load accumulator A with 3
2                STA C       ; Store contents of accumulator A in memory location C
3    LAB1        LDA Y       ; Load accumulator A with contents of memory location
                               Y
4                MUL Z       ; Multiply contents of the accumulator A by the contents
                               of memory location Z
5                STA Y       ; Store contents of accumulator A in memory location Y
6                LDA C       ; Load accumulator A with contents of memory location
                               C
7                SUB #1      ; Subtract 1 from accumulator A
8                STA C       ; Store contents of accumulator A in memory location C
9                JNZ LAB1    ; Jump to LAB1 if contents of accumulator is not 0
10               HLT         ; End of program
```

Given that initially 3 is stored in location Z and 1 is stored in location Y, what are the contents of Y in the end.

Solution:

1	A = 3		
2	C = A C = 3		
3	A = Y A = 1	A = 3	A = 9
4	A = A*Z A = 3	A = 3*3 A = 9	A = 9*3 A = 27
5	Y = A Y = 3	Y = 9	Y = 27
6	A = C A = 3	A = 2	A = 1
7	A = A-1 A = 2	A = 1	A = 0
8	C = A C = 2	C = 1	C = 0
9	Jump to 3	Jump to 3	No jump
10			End program

A Note

For the purpose of this section the size and type of accumulator is 8-bit unsigned.