4.8 Assembly Language

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**
- if 10110 is the binary code for LDA, then 10110 is called the **opcode** (sometimes LDA itself is called the opcode)
- LDA is also called a **mnemonic**
- 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**

Memory address modes tell you how the CPU accesses data.

Three commonly-used memory address modes are the following:

- Immediate
- Direct
- Symbolic

Immediate addressing means that the data is found inside the instruction itself.

Example:
'LDA #12' means: load 12 into A (the accumulator).

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

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

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

Advantage: fast (but not as fast as immediate addressing).
Disadvantage: the program cannot be relocated.

Re-locatable code refers to code that can be put anywhere in RAM.

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.

Data transfer instructions:

Note that what is written after the semicolon is a **comment**.

```
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)
ORA x  ; Logical OR the contents of accumulator A with x (bitwise operation)
NOT    ; Logical NOT the contents of accumulator A (no operand).
SHL    ; Logical shift contents of accumulator A to the left and introduce a 0 in the vacated
         bit (no operand and bitwise operation).
SHR    ; Logical shift contents of accumulator A to the right and introduce 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 0
         (zero)
JNZ x  ; Jump to the instruction pointed to by the label x if contents of accumulator is not 0
         (zero)
```

Other instructions:

```
HLT   ;End of program
```