Unit 4. Inside Software Programs

Objectives of the unit

At the unit end the student should know

- Contents of a program
- Instructions and their hierarchies
- Control structures
- Fields

Two Parts

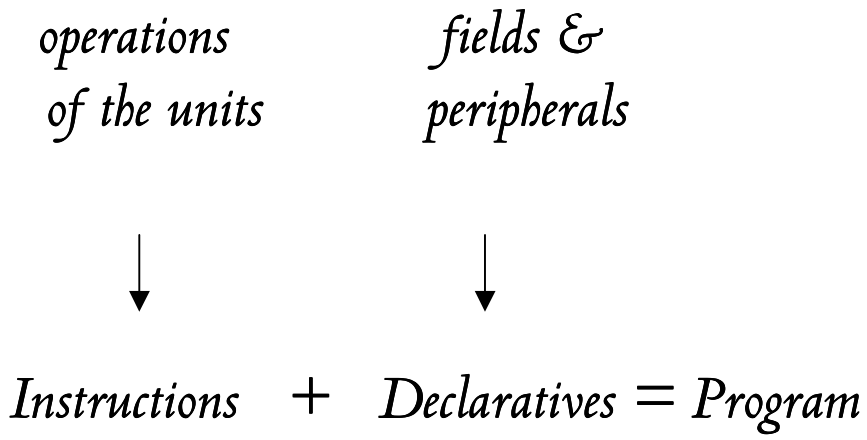

Figure 4-1

The hardware equipment is unfinished and unable to perform any task. Electronic engineers leave the computer system such as and software specialists complete it. They define the precise job which the computer will execute and assign this program-work to the machine.

Programmers bring the construction of the computer system to an end.

Which sides of the machine does the programmer finish? Which are the main contents of the program?

The undefined parts of the computer system are precisely the following two:

- **The instructions in MC1**
- **The data in MC2 and the peripherals to connect.**

Consequently a program encompasses the ensuing sections

- ***INSTRUCTIONS*** command directly the computer to work and constitute the so-called **algorithm**.

- ***DECLARATIVES*** define the data that the instructions manipulate in MC2 and select the peripherals necessary for the work program.

Both the parts are essential and set right the machine.

Linguistic Remark: The term *algorithm* derives from the mathematical field where it means “rules of calculus”. Computers conversely process information that is text, sound, picture and very rarely is an abstract number !

Programming

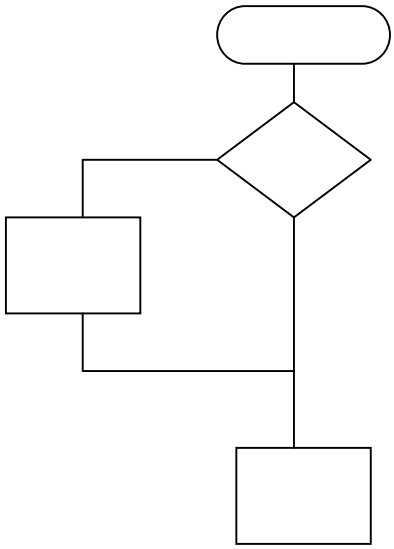

Figure 4-2

A program cannot be improvised and the developer prepares an appropriate design in advance of the writing phase. In short the program development includes the ensuing steps.

- **DESIGN :** The programmer outlines the algorithm and the declaratives on paper.
- **CODING :** The programmer translates the scheme drawn in the previous step using the programming language and releases the software program.

The programmer selects the formalism of design, which is the most suitable for the program coding that will come next. He has two schemes at disposal:

- Pseudocoding : is oriented to the symbolic programming languages (e.g. Cobol, PL/1, C, Java).

- Block Diagram : conforms to the Assembler language.

This concise graph (see Figure 4-2) suits the Assembler language that reflects the bare operations of the hardware units. Logicians laud the properties of the block diagram; instead today we could say that it is somewhat out of action. Once again abstract studies sidetrack people from the reality.
Pseudocoding is much more popular in respect to the block diagram, in fact it looks like the natural language and is very easy to write and to read. We shall illustrate the nest pages with this tool, so that you can realize the inner structure of a program without any specific study.

Instructions

- *Read*
- *Write*

→ ***Input/Output Units***

- *Movement instructions (Move, Assign, Shift,Store)*
- *Arithmetic instructions (* / + -)*
- *Logical instructions (And, Or, Nand, Xor etc.)*
- *Various instructions*

→ ***Arithmetic/Logic Unit***

- *Goto*
- *Compare*

→ ***Control Unit***

Figure 4-3

A software instruction commands the execution of one operation of CU, ALU and TR, thus they fall into three separate groups.

i) **External instructions** enable TR and are essentially

- Read
- Write

They activate a variety of peripherals such as the keyboard, the printer and the disk drive.

ii) **Internal instructions** enable ALU and are essentially

- Movement operations
- Arithmetic operations
- Logical operations
- Various operations.

They make the largest menu.

iii) **Control instructions** enable CU and are

- Goto
- Compare

They make the smallest and the most influential menu, as we shall see.

Instructions (contd.)

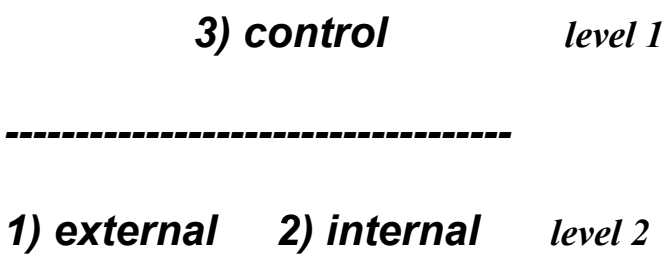

Figure 4-4

The Arithmetic and Logic Unit (ALU) and the peripherals (TR) rely on CU. In particular, the hardware hierarchy in Figure 2-5 implies that the software instructions rank two levels. Menu 3) settles on top and the others lie bottom. This means that the control instructions have influence on the external instructions and those internal. What does mean, in the practice, that the control instructions "influence" all the others?

Goto does not perform any task instead it establishes the execution of a function. For example, ***Goto X*** triggers the instructions X, namely Goto rules over X.

Compare examines two data in MC2. E.g. ***Compare A, B*** confronts A with B, then Goto launches the appropriate instructions according to the outcome of Compare. In short, Compare takes a leading role with respect the common instructions.

The control instructions do not achieve mathematical results, instead they supervise the other units. They modify the behaviors of the remaining operations, namely they guide the execution of the instructions 1) and 2).

We conclude that the instructions 3) provide the appropriate order of execution to the sections of the program, and in practice the programmer organizes the plan of work for the machine through the control instructions.

Macro Instructions

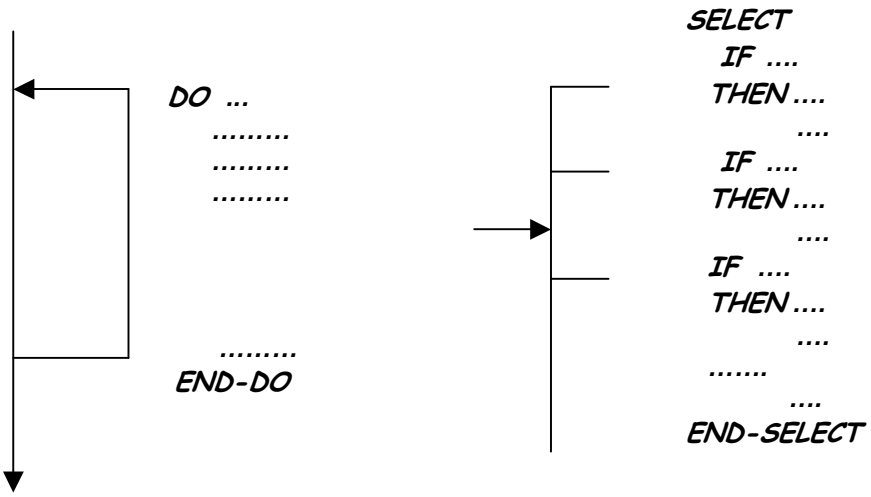

Figure 4-5

When a programmer writes Goto and Compare out of any rule, he gives a casual organization to the program.

In order to avoid disorganized processes, the CU instructions are rigidly composed and make two main **control macro-instructions**. In practice programmers must write the control macros instead of Goto and Compare that could be used improperly.
Which are the control macros? What do they?

The control macros are chiefly the following two

- **DO**
- **SELECT**

The former triggers repeatedly a group of instructions; the latter executes two, three or more functions that are alternative. In particular DO repeats the internal/external instructions (A) written inside.

```
DO
  :::
  :::     (A)
  :::
  :::
END-DO
```

The second macro executes the blocks X, Y, Z according to the conditions placed aside.

```
SELECT
   IF...
      :::   (X)
   IF...
      :::   (Y)
   IF...
      :::   (Z)
END-SELECT
```

For example, the loop prints five lines with the message “Cheer Up!”

```
DO 5 TIMES
     Write "Cheer Up !"
END-DO
```

The next SELECT prints the explicit meaning of the symbols S, M and D

```
SELECT
     IF State = S
     THEN Write "Single"
     IF State = M
     THEN Write "Married"
     IF State = D
     THEN Write "Divorced"
 END-SELECT
```

Macro Instructions (contd.)

```
BEGIN       DO ....        SELECT
  ....        ....           IF ....
  ....        ....           THEN....
  ....        ....              ....
  ....        ....           IF  ....
  ....        ....           THEN....
  ....        ....              ....
  ....        ....           ....
  ....        ....           ....
END         END-DO         END-SELECT
```

Figure 4-6

The Control Unit normally fires the operations in sequence. It executes the internal and external instructions one after the other. The programmer modifies this monotonous order by means of DO and SELECT.

In short the programmer arrange the operations of the program in three shapes, called **structures** from now onward:

1] **sequences** (automatically)

2] **loops** (using the macro DO)

3] **selections** (using the macro SELECT)

They constitute the basic forms of a software algorithm.

In the pioneer age programs were designed out of any rule with remarkable burdens. Software maintenance was almost impossible. The Böhm-Jacopini theorem, published in 1966, proved the control structures theoretically and programmers got effective guidelines.

Macro Instructions (contd.)

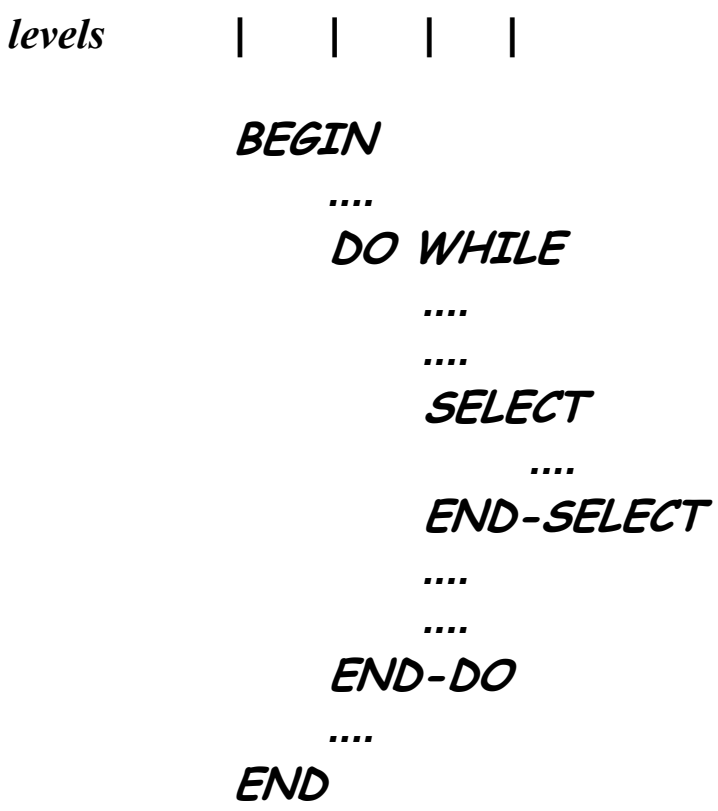

Figure 4-7

Developers identify the structures using keywords written in capital letters. They open with a keyword and close with END

BEGIN... **END**-BEGIN
DO... **END**-DO
SELECT... **END**-SELECT

The first structure, rather trivial, encloses the entire algorithm.
Programmers indent the controlled instructions inside the structure. The keywords and the internal/external instructions lie on distant vertical lines that mark the hierarchy.

A macro can control another structure. When A controls the structure B, then B is indented inside A. In Figure 4-7 BEGIN includes DO and in turn DO controls SELECT.

The macros subdivide the program in blocks and aid the programmer to handle a complex algorithm. In fact he designs a program more easily if this is sectioned into parts.

Cells

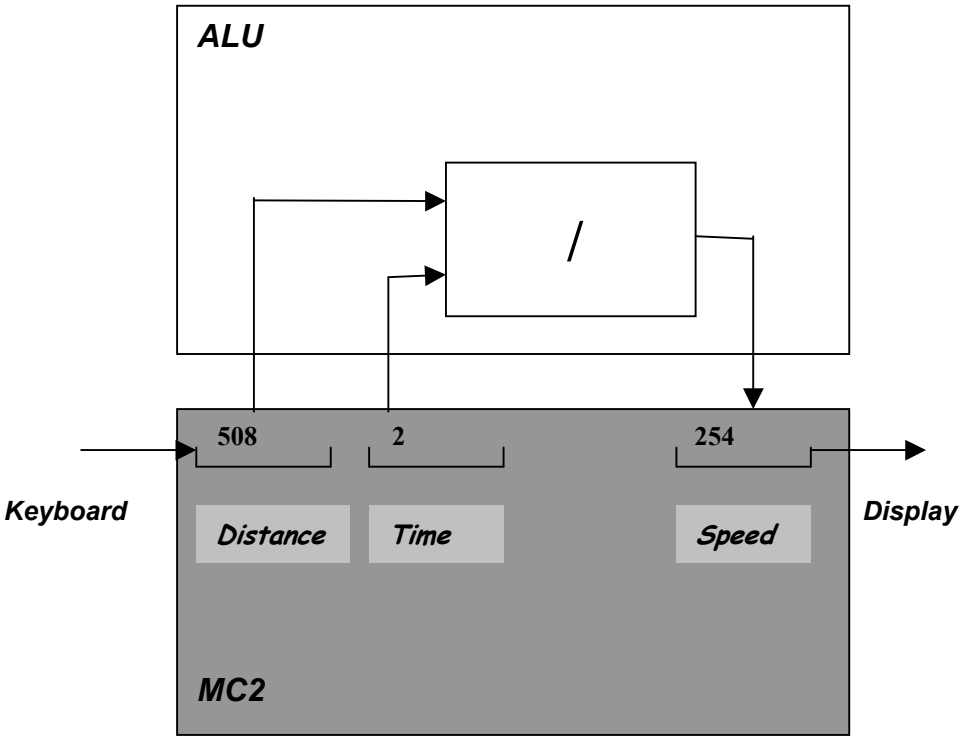

Figure 4-8

Both external and internal instructions use cells of memory that we call **fields** and also "area", "elements", "spaces" etc.
The storage fields are the essential tools for the mechanical treatment of information because any field has assigned a distinguished meaning. Take for example the program that calculates the speed from the distance and time. (Figure 4-8) The user enters two numbers and the keyboard puts 508 into “Distance”, and 2 into “Time”. ALU divides the values and places the result unto “Speed”. Lastly this one is displayed on the screen. Note how the fields can operate with any values of distance, time and speed on condition that the binary word is not longer than the container.

The fields clarify how the processor is capable of producing new meanings, in accordance with the definition given in Chapter 2, although it is “stupid”.
Logicians and theoreticians neglect the relevance of these physical items and lose themselves in long discourses. Instead the practical stance opens the door to the most simple and realistic explanation. The fields elucidate how the machine provides extraordinary answers although it is "unconscious" of what it does, as somebody still does not believe, due to the fields. These tiny components conceal a great secret of ICT that we are able to reveal in the simplest manner.

Linguistic Remark: Fields are also told as *operands* and *variables*. Do not mistake the computer operands and variables, which are material cells in storage and have physical properties, with the mathematical operands and variables which are abstract.

Cells of Memory (contd.)

```
BEGIN -
  Read Distance, Time
  Speed = Distance / Time
  Write Speed
END
```

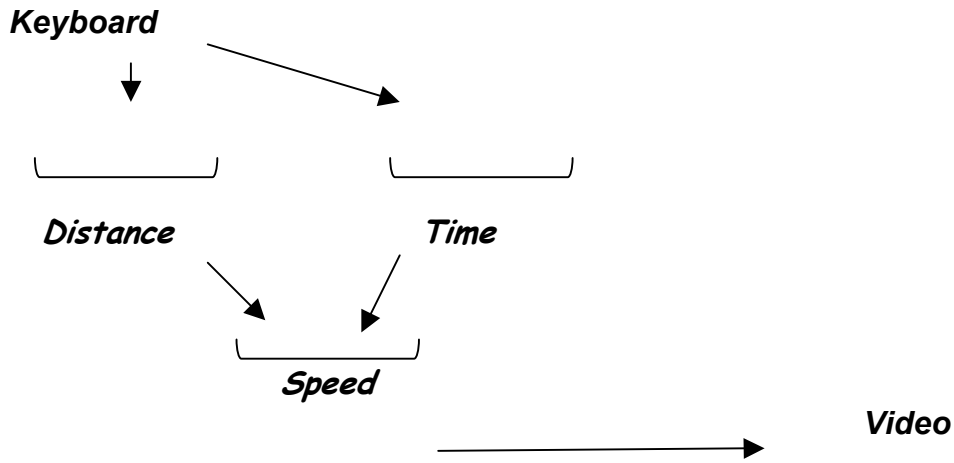

Figure 4-9

We take a couple of cases in order to clarify how software programs arrange the instructions, the fields and the peripherals.

I) We reopen and expand the example cited in the previous pages. The user presses Distance and Time. The program computes and displays the Speed on the video screen. The pseudocoding and the scheme with the fields and the peripherals illustrate the whole program step by step (Figure 4-9).

II) The user introduces the initial of his civil state. If he/she presses S(ingle), M(arried) or D(ivorced), the program displays the state otherwise it notices the input is wrong. Figure 4-9bis illustrates how the program runs.

```
BEGIN –
   Write "Press the initial of your state"
   Read Initial
   SELECT
      IF State = S
      THEN Write "Single"
      IF State = M
      THEN Write "Married"
      IF State = D
      THEN Write "Divorced"
      ELSE Write "Wrong Choice !"
   END-SELECT
END
```

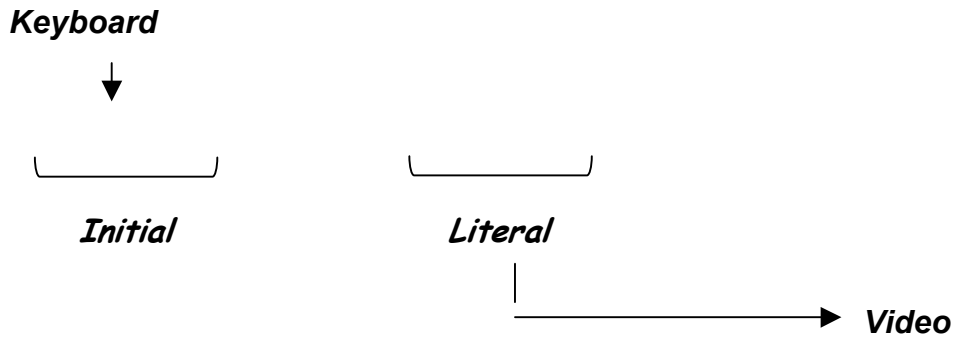

Figure 4-9bis

Call

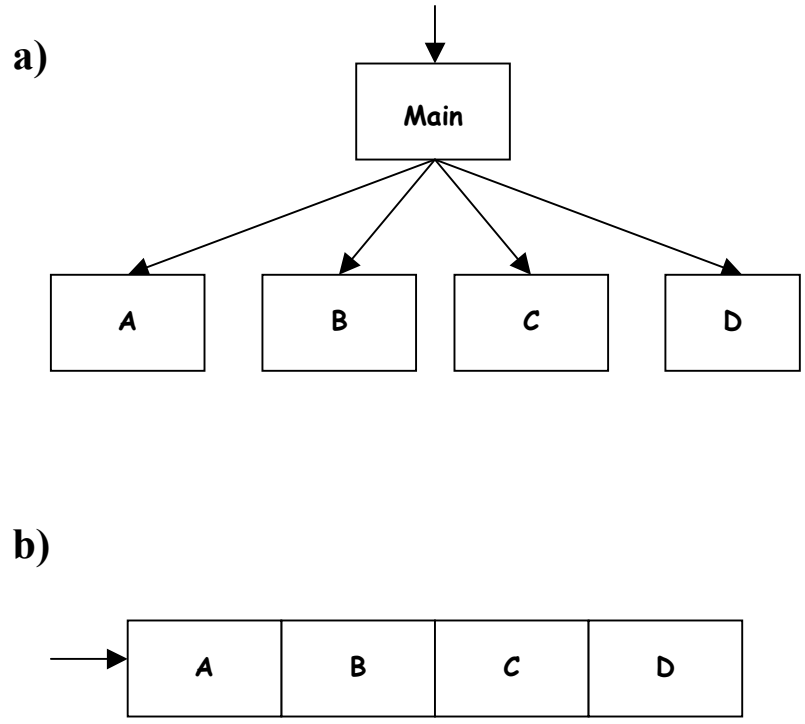

Figure 4-10

If the program is large (e.g. it has more than hundred lines), practitioners subdivide it for more agile handling. Each module executes a function and may be entrusted to a specialist who works independently from the colleagues.

The programming structures make this partitioning easier and guide the organization of the software package.
At the end the parts are put back together in two ways.
The parts of the package may be potentially arranged in two ways.

b) - The merged modules make a rigid package. The parts are steadily combined through physical links. For example, the program executes the functions A, B, C and D, and they are united as in Figure 4-10b.

a) - Specialists dislike a huge body and prefer to have an agile structure that links the modules through dynamical connections. They append a special module on top, named **Main,** that calls the functions of the algorithm. For example, the Main encompasses the ensuing instructions that trigger A, B, C and D in sequence by means of CALL A, CALL B, CALL C and CALL D.

```
BEGIN
      CALL A
      CALL B
      CALL C
      CALL D
END
```

In such a way the software product fulfils its duties. Figure 4-10a exhibits the whole package.

CALL is a special control macro since it a combination of control instructions. By definition it establishes hierarchical ranks. The scheme in Figure 4-10a exhibits the hierarchy of blocks and will be quoted in the last chapter of the present book.

Unit 4 SUMMARY

This chapter has deduced a number of properties of the software programming from the hierarchy property of the hardware units.

We have discovered the essential role of fields to the information processing.

Hierarchy of units

↓

Hierarchy of instructions

↓

Control Macros

↓

Programming Structures