Int. J. Engng Ed. Vol. 11, No. 2, pp. 111–118, 1995
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Functional Simulation in Microprocessors Applications Teaching*

JACEK MAJEWSKI
Cybernetics Institute, Technical University of Wroclaw, ul. Janiszewskiego 11/17, 50-370 *Wroclaw, Poland*

In microcomputers and microprocessors instruction the student's job is to write programs for controlling equipment connected to a microcomputer. Instead of real devices we propose control devices simulated on the computer screen. Of course, it is important that the simulated environment is invisible for the students. Control programs, written by students, should work in the same way in the simulated environment as in a real environment. The paper considers as an example the preparation of a control program for the model of a plotter. The control plotter programs are written in C and 8086 assembler and compiled by real compilers: Borland C and TASM.

INTRODUCTION

FROM THE BEGINNING when microcomputers were invented, the most common microprocessor communication with the outside world has been through input/output ports that are located in the I/O or memory space. This facilitates constructing more advanced and more complicated microcomputer peripherals such as parallel or serial ports, timers or counters, DMA controllers and so on. Generally, internal ports of these peripherals are designed for the following functions:

- DATA ports;
- CONTROL ports;
- STATUS ports;

Communication through input/output ports is also applied for devices such as printers, plotters, floppy or hard discs, etc. More complicated devices, such as display graphic controllers, contain ports in addition to memories. Complex external devices have complicated electrical schemes. From the programmer's point of view, electrical details of controlled devices are not essential. For proper control it is enough to know what information to send to the CONTROL/DATA ports. The situation is the same even if external devices are causing interrupts.

In the teaching of microprocessor programming, the above idea is very common: students control external real devices, connected to the microcomputers, by writing programs in a specified language (assembler, C, BASIC). These programs are sending sequences of controls through input/output ports. Instead of real devices we consider the situation of controlling simulated devices displayed on the computer screen. This aspect is presented below. For better understanding we consider a case study of controlling a microprocessor external device. A *Plotter* as an example of an exercise given to a student.

* Paper accepted 25 November 1994.

STUDENT EXERCISE: PLOTTER CONTROL

Before writing a program it is important to understand how a controlled device works. Figure 1 shows a plotter connected to an IBM computer. The model of the plotter is similar to a real teaching device designed by ELWE of Germany. By controlling the movement of the plotter arm, equipped with a pen, students should draw a simple figure (e.g. a rhombus in Fig. 1). The plotter is controlled by the IBM PC computer through the output port located at the I/O position with the 300 Hex address. States of the plotter are read by two input ports, placed at addresses 300Hex and 301Hex. The bit map of I/O ports is shown in Fig. 2. The kinematics scheme of the plotter is shown in Fig. 3. The movement of the plotter arm (and the pen) is controlled by two motors. The motor M1 shifts the arm to the left or right (LF, RT bits). The movement of the pen forward and backward (FW, BW bits) are controlled by motor M2. The active state for all bits is 1. The output 300Hex port consists also of a PEN bit that puts the plotter pen to the down position and makes a point on the drawing sheet (bit PEN=1) and the D1 bit that turns the LED diode on or off.

The position of the plotter arm is indicated by sensors: left sensor (S3), right sensor (S4), forward sensor (S5) and backward sensor (S6). Additionally, when the pen is located exactly at the bottom left hand corner of the sheet the SP bit is equal to 1. The plotter has also three free push-buttons (S0, S1, S2) which can be programmed by the user.

The movement in a given direction can be

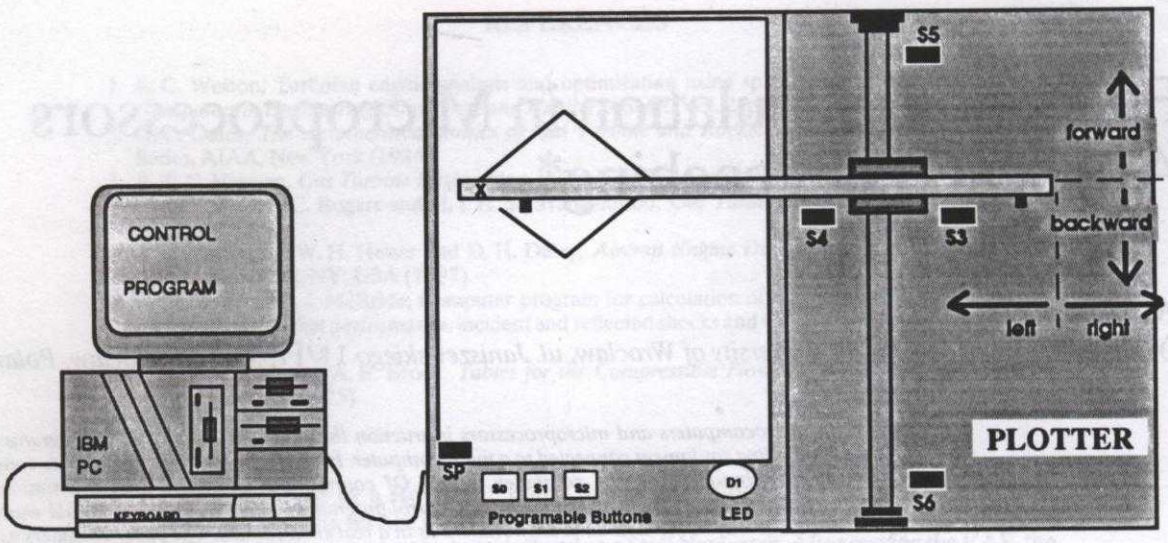

Fig. 1. Real situation: IBM computer controls a plotter.

Output Port Address: 300H:

7	6	5	4	3	2	1	0
X	X	PEN	BW	FW	LF	RT	D1
No meaning		Pen on/off	Backward	Forward	Left	Right	LED on/off
			Controlled by M2 Motor		Controlled by M1 Motor		

Input Port Address: 300H:

7	6	5	4	3	2	1	0
S6	S5	S4	S3	SP	S2	S1	S0
Backward Sensor	Forward Sensor	Right Sensor	Left Sensor	Pen Position Status	Programmable Push Buttons		

Input Port Address: 301H:

7	6	5	4	3	2	1	0
X	X	X	X	X	X	PG2	PG1
No meaning						Pulse Generator of M2 Motor	Pulse Generator of M1 Motor

Fig. 2. Bit assignment of I/O plotter ports (active state is 1).

precisely measured by counting the number of pulses produced by pulse generators: each motor (M1, M2) has one pulse generator. Rotation of the motor M1, when the pen is moved to the left/right direction, causes pulses that can be observed on bit PG1, for example.

For the device described, we write two simple programs: one in Borland C [3] and one in 8086 TASM assembler [4]. The programs are shown in Figs 4A and 5A. Both programs draw a rhomboidal spiral with the assumption that for drawing a line it is enough to activate a motor and wait (DELAY subroutine). The shape of the drawn spiral is shown in Fig. 6.

FUNCTIONAL SIMULATION OF THE PLOTTER

As mentioned in the Introduction and as is visible in the programs demonstrated in Figs 4 and 5, control of any external microprocessor devices

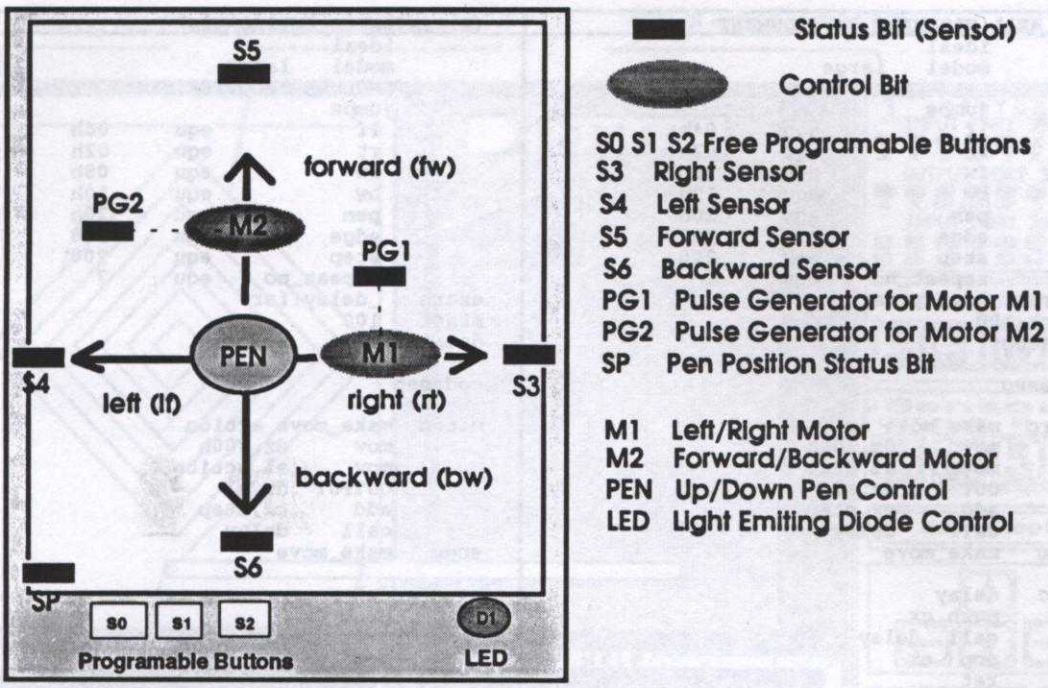

Fig. 3. Kinematics scheme of the plotter.

(A) REAL HARDWARE ENVIRONMENT

```
#INCLUDE <DOS.H>

#define fw              0x08
#define bw              0x10
#define lf              0x04
#define rt              0x02
#define pen             0x20
#define edge            300
#define step            200
#define repeat_no       7
void main(void)
{
  long int j,i,k=edge;

  for (j=0;j<repeat_no;j++)
  {
       OUTPORTB(0x300,fw+lf+pen);
       delay(k+=step);
       OUTPORTB(0x300,fw+rt+pen);
       delay(k+=step);
       OUTPORTB(0x300,bw+rt+pen);
       delay(k+=step);
       OUTPORTB(0x300,bw+lf+pen);
       delay(k+=step);
  };
       OUTPORTB(0x300,0x00);
}
```

(B) SIMULATED ENVIRONMENT

```
#INCLUDE "PLOTTER.h"

#define fw              0x08
#define bw              0x10
#define lf              0x04
#define rt              0x02
#define pen             0x20
#define edge            300
#define step            200
#define repeat_no       7
void main(void)
{
  long int j,i,k=edge;
  START();
  for (j=0;j<repeat_no;j++)
  {
       OUTPUT(0x300,fw+lf+pen);
       delay(k+=step);
       OUTPUT(0x300,fw+rt+pen);
       delay(k+=step);
       OUTPUT(0x300,bw+rt+pen);
       delay(k+=step);
       OUTPUT(0x300,bw+lf+pen);
       delay(k+=step);
  };
     OUTPUT(0x300,0x00);
STOP();
}
```

Fig. 4. Exemplary programs, written in C language, for drawing a rhomboidal spiral.

is a sequence of *input* or *output* instructions. From the user's point of view, electrical or mechanical details of controlled environments are not important. It is enough to understand that after setting a proper bit in a port there will be a proper reaction. This level of understanding means that only functions of controlled devices are important for the user.

With this assumption, we construct an artificial environment, simulated graphically on the computer screen. The situation when the described plotter is simulated on the IBM PC computer screen is shown in Fig. 6. Control programs are almost the same for the real plotter and the plotter simulated on the screen. Figures 4B and 5B present the same programs as Figs 4A and 5A. The

(A) REAL HARDWARE ENVIRONMENT

```
        ideal
        model   large

        jumps
        lf              equ     04h
        rt              equ     02h
        fw              equ     08h
        bw              equ     10h
        pen             equ     20h
        edge            equ     300
        step            equ     200
        repeat_no       equ     7
extrn   _delay:far
stack 100
dataseg

codeseg

macro   make_move action
        mov     dx,300h
        mov     al,action
        OUT     dx,al
        add     cx,step
        call    delay
endm    make_move

proc    delay
        push cx
        call _delay
        pop  cx
        ret
endp

proc    _main   far
        PUSH    DS
        SUB     AX,AX
        push    ax
        mov     cx,edge
        mov     bx,repeat_no
repeat:
        push    bx
        make_move fw+lf+pen
        make_move fw+rt+pen
        make_move bw+rt+pen
        make_move bw+lf+pen
        pop     bx
        dec     bx
        jnz     repeat
        mov     al,00h
        OUT  dx,al


        ret
endp

end     _MAIN
```

(B) SIMULATED ENVIRONMENT

```
        ideal
        model   large
        INCLUDE "PLOTTER.MAC"
        jumps
        lf              equ     04h
        rt              equ     02h
        fw              equ     08h
        bw              equ     10h
        pen             equ     20h
        edge            equ     300
        step            equ     200
        repeat_no       equ     7
extrn   _delay:far
stack   100
dataseg

codeseg

macro   make_move action
        mov     dx,300h
        mov     al,action
        OUTPUT  dx,al
        add     cx,step
        call    delay
endm    make_move

proc    delay
        push cx
        call _delay
        pop  cx
        ret
endp

proc _main far
        CALL    _START

        mov     cx,edge
        mov     bx,repeat_no

repeat:
        push    bx
        make_move fw+lf+pen
        make_movefw+rt+pen
        make_move bw+rt+pen
        make_move bw+lf+pen
        pop     bx
        dec     bx
        jnz     repeat
        mov     al,00h
        OUTPUT  dx,al

        CALL    _STOP
        ret
 endp

end
```

Fig. 5. Exemplary progam, written in 8086 assembler language, for drawing a rhomboidal spiral.

differences between the programs are shown in capital letters. How does the simulated plotter work? a C sample program will be considered. Considerations for assembler are similar. The general structure of user C programs is as follows:

include „plotter.h"
■ ■ ■
Start()
■ ■ ■
output(port, value)
■ ■ ■
input(port)
■ ■ ■
Stop()

The statement *include „plotter.h"* includes definitions for the simulated environment.

Start(), *Stop()* functions are for creating (removing) the plotter picture on the screen as a background.

The function *output* moves the plotter arm with the pen in the output port direction. When the arm reaches the edge of the sheet a sound is produced. The sound means that the moving mechanics are being destroyed.

The function *input* gives the state of the sensors. There are two situations when the state of a sensor is changed. First, the state of sensors is changed as a result of movement when a user program is executed. Second, at any moment when a user

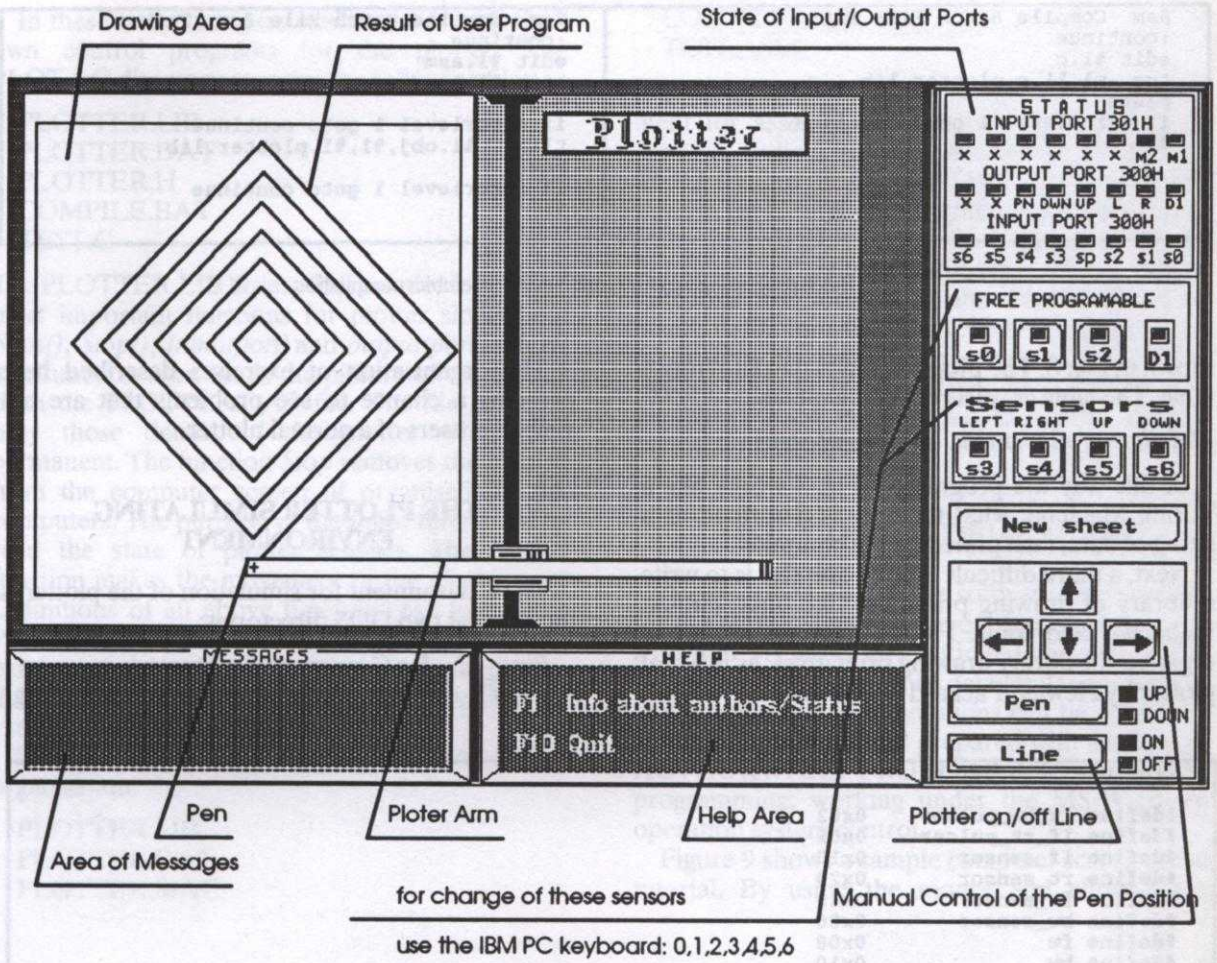

Fig. 6. The plotter simulated on the IBM PC screen: all details are the same as for the real plotter.

program is executed the user has the possibility to change the states of sensors "manually" by pressing keys on the IBM PC keyboard, for example, "4" for changing the state of sensor *S4*. In this way the user has a chance to "cheat"—it is possible to make the left sensor active (lf) before the plotter arm reaches the left edge of the drawing sheet.

States of all sensors and control bits are visible on the computer screen (see Fig. 6).

The description above shows that only functions of the plotter are simulated, not electrical or mechanical details. This degree of simulation we call functional simulation [1].

LABORATORY EXPERIMENT ORGANISATION

The process of a control program preparation by a student (a user) can be considered as consisting of two steps:

- tutorial: understanding of controlled device, for example the plotter;
- programming: cycle of editing, compiling, linking and program execution.

Tutorials are set by teachers who with their voice, chalk and hands describe how the device works. A better solution is of course to give students a written (boring) description on paper, similar to that presented at the beginning of this paper.

The process of programming can be simplified by using batch processing files shown in Fig. 7: one for the C compiler and one for the TASM assembler. Both files are executing the endless loop: edit–compile–link and execute. It is worth mentioning that the C and assembler programs use the same library *plotter.lib*.

LABORATORY EXPERIMENTS PROPOSALS

Our model of the plotter allows students to write programs of different levels of difficulty. It is easy to see that examples shown in Figs 4 and 5 do not exhaust the real possibilities of a plotter. The real possibilities are to draw a line in a given direction with a certain length and a given number of steps. An example of how to draw lines in the left/right or forward/backward direction, with a given length, is

```
Rem  Compile Batch File for   C
:continue
edit %1.c
tcc -ml %1.c plotter.lib
pause
if errorlevel 1 goto continue
%1
```

```
Rem  Compile Batch File for  TASM
:continue
edit %1.asm
tasm %1.asm
pause
if errorlevel 1 goto continue
tlink  %1.obj,%1,%1,plotter.lib
pause
if errorlevel 1 goto continue
%1
```

Fig. 7. Batch processing files for C and TASM assembler compilation.

shown in Fig. 8. The program reads the data from a table. The table consists of two elements:

direction-of-movement, number-of-steps

In this way the program of Fig. 8 works like a Turing Machine: what to draw is described as a string of data, interpreted by the computer.

Next, a more difficult level of exercise is to write a library of drawing primitives like lines, circles, rectangles and fonts. Finally there is a need to organise the library drawing primatives, as in a real plotter by Hewlett Packard or Rolland.

The organisation of exercises described here presents a chance to see problems that are not visible to users of a normal plotter.

THE PLOTTER SIMULATING ENVIRONMENT

The environment for simulation of the plotter is localised at two DOS directories:

Plot_C—for C language programming;
PLOT_A—for TASM language programming.

```
#include "plotter.h"

#define fw_bw_pulser     0x02
#define lf_rt_pulser     0x01
#define lf_sensor        0x10
#define rt_sensor        0x20
#define fw_sensor        0x40
#define bw_sensor        0x80
#define fw               0x08
#define bw               0x10
#define lf               0x04
#define rt               0x02
#define pen              0x20

unsigned char tab[]={fw+pen+lf, 2,lf+pen+bw, 4,bw+pen+rt, 6,rt+pen+fw, 8,
                     fw+pen+lf,10,lf+pen+bw,12,bw+pen+rt,14,rt+pen+fw,16,0,0};
#define move(pulser, end_sensor, action, steps_no)                    \
        if( !(input(0x300) & (end_sensor)) )                          \
          { output(0x300,(action) );                                  \
            counter=0;                                                \
            old= (input(0x301) & (pulser) );                          \
            while( counter < (steps_no) )                             \
                 {                                                    \
                  if( input(0x300) & (end_sensor) ) break;            \
                  new = (input(0x301) & (pulser) );                   \
                  if ( old != new ) { old= new; counter++; }          \
                 }                                                    \
          }                                                           \
      output(0x300,00)

void make_move(unsigned char direction, unsigned char steps_no)
{
  unsigned char old,new,counter;
  if(direction & lf)  move(lf_rt_pulser,lf_sensor,direction,steps_no);
  if(direction & rt)  move(lf_rt_pulser,rt_sensor,direction,steps_no);
  if(direction & fw)  move(fw_bw_pulser,fw_sensor,direction,steps_no);
  if(direction & bw)  move(fw_bw_pulser,bw_sensor,direction,steps_no);

}
void main(void)
{
 int i=-2;
 Start();
 while(tab[i+=2])  make_move(tab[i],tab[i+1]);
 while(1);
 Stop();
}
```

Fig. 8. Example of a more advanced C language program for precisely measured drawings.

In these two catalogues students can write their own control programs for the plotter. The PLOT_C directory contains the following files:

PLOTTER.LIB
PLOTTER.DAT
PLOTTER.H
COMPILE.BAT
TEST.C

The PLOTTER.LIB library file includes four of the most important functions for plotter simulation: *Start()*, *Stop()*, *Input(port)* and *output(port, value)*. The function *Start* loads the picture of the plotter from the PLOTTER.DAT file. The file contains only those details of the plotter which are permanent. The function *Stop* removes the picture from the computer screen of organised on the computers. The purpose of the *input* function is to read the state of plotter sensors. The *output* function makes the movement of the plotter arm. Definitions of all above functions are located at PLOTTER.H header file. The COMPILE.BAT batch processing file simplifies the edit–compile–link–execute process. The file TEST.C contains an exemplary control C language file.

The PLOT_A directory has the following files organisation:

PLOTTER.LIB
PLOTTER.DAT
PLOTTER.MAC
COMPILE.BAT
TEST.ASM

It is worth mentioning that the simulated environment for assembler programming is a result of creation of the C simulation environment. PLOTTER.LIB and PLOTTER.DAT are exactly the same as for C language programming. It means that from the assembler level the C language library functions are called. Definitions of all plotter functions are located at the PLOTTER.MAC macro definitions file. The COMPILE.BAT batch processing file simplifies the edit–compile–link–execute process for the assembler language. The file TEST.ASM contains an exemplary control assembler language file.

ABOUT THE TUTORIAL AGAIN

As described above, the software environment for the plotter simulation and programming needs a tutorial. As has been mentioned, the tutorial can be prepared by a teacher or delivered to students as written material. Both solutions can be combined into a computer tutorial prepared with the help of AUTHORWARE PROFESSIONAL [2] system programming, working under the MS-Windows operation system control.

Figure 9 shows a sample computer screen of the tutorial. By using the mouse and clicking the

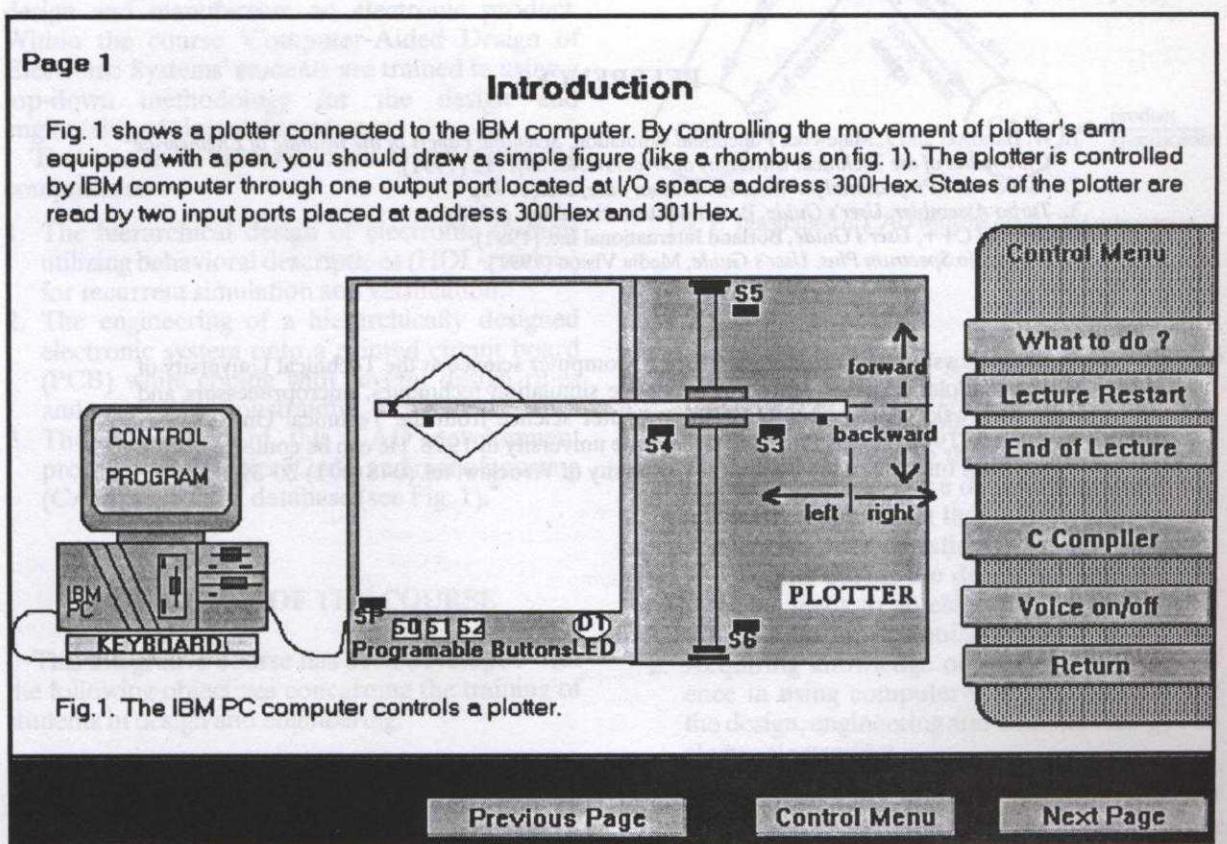

Fig. 9. Screen display example of the tutorial written with the help of the AUTHORWARE program.

buttons on the screen, students can continue the lesson (*Next Page* button), return to the previous page (*Previous Page* button) or call for help (*Control Menu* button). By pressing the *Control Menu* button on the screen, extra buttons appear.

What to do?
Lecture Begin
End of Lecture
Return
Voice on/off
C Compiler

The *What to do?* button opens the window with instructions on how to navigate the tutorial program.

The *Lecture Begin* and *End of Lecture* buttons are for starting the tutorial from the beginning or for quitting the tutorial.

The *Return* button is for closing the window with the help and with extra buttons.

The *Voice on/off* button is for activation of the human voice of a speaker. If the IBM PC computer has a Sound Blaster Card [5] it is possible to hear the text instead of reading it.

The *C Compiler* button causes exit to the DOS operating system and executes the COMPILE. BAT batch processing file for the student's program preparation.

The tutorial has the following elements:

- the plotter text description;
- kinematics scheme of the plotter;
- bit maps of input/output ports;
- sample C control plotter program;
- execution of the sample program;
- formulation of student's exercise.

SUMMARY

Functional simulation was used for the creation of a laboratory for the subject of *Microprocessor Applications*. The following equipment connected to microprocessors was simulated:

- traffic lights board;
- stepper motor;
- industrial conveyor belts and lifts;
- dynamic LED display/keyboard;
- plotter.

This list of exercises does not include a series of logically connected experiments from the teaching point of view. Exercises were chosen from the point of view of simulation difficulties: the first exercise is very simple to simulate but the last two are associated with multitasking problems. All exercises are available for programming in C and TASM languages. For all exercises, the tutorials, written in the AUTHORWARE language, are under preparation.

Acknowledgement—Parts of this work were supported by the European Commission TEMPUS JEP 1087 and COMETT Cb 6717 programme grants.

REFERENCES

1. W. Baranski and J. Majewski, Functional simulation, *Scientific Papers of the Institute of Engineering Cybernetics of the Technical University of Wroclaw No. 89*,7–12 (1991).
2. *Authorware Professional for Windows* Authorware, Inc. (1991).
3. *Turbo Assembler, User's Guide*, Borland International Inc. (1988).
4. *Borland C++, User's Guide*, Borland International Inc. (1991).
5. *Pro Audio Spectrum Plus, User's Guide*, Media Vision (1991).

Jacek Majewski is an assistant professor of computer science at the Technical University of Wroclaw (Poland). His research interests are simulation techniques, microprocessors and DSP. Majewski received his M.Sc. in computer science from the Technical University of Wroclaw in 1972 and his Ph.D. from the same university in 1978. He can be contacted at the Cybernetics Institute of the Technical University of Wroclaw, tel. (048/071) 20-39-96.