A SIMULATION APPROACH BEFORE USING THE INDUSTRIAL MICROCOMPUTER CONTROLLER

INFORMATICA 3/1988

Anton Zorman
Maksimiljan Gerkeš
Viljem Žumer
Krista Rizman
Tehniška fakulteta Maribor

UDK 681.326.06:519.876.5

In this paper we describe a simulation packet that enables the verification of the software portion of the controller before implementating the controller on a real object. This program is unique of its kind, as far as we know, because it enables the examination of the controller's behavior in planner's workplace and so considerably reduces start expenses. The simulation packet enables controlled execution of the user's program, clearly arranged writing out of data on the screen or on the printer and gives a chance for data modification.

SIMULACIJSKI PRISTOP PRED UPORABO INDUSTRIJSKEGA MIKRORAČUNALNIŠKEGA KRMILNIKA.
V članku opisani **simulacijski paket** omogoča verifikacijo programskega dela krmilja pred vgraditvijo krmilnika na objekt. Po nam znanih podatkih je to edini tovrstni program, ki omogoča preizkus obnašanja krmilja na projektantovem delovnem mestu in s tem znatno znižanje zagonskih stroškov objekta. **Simulacijski paket omogoča** prikaz izvajanja uporabniškega programa, urejen in pregleden izpis vrednosti na zaslon ali na tiskalnik in možnost spreminjanja vrednosti podatkov.

1 Initial considerations about the industrial microcomputer controller and it's simulation

Requests for a system which upgrades standard programmable controller functions came from industry. Initial efforts were made by Metalna, Maribor. After it's definition phase the project was supported by the Research Society of Slovenia (Raziskovalna Skupnost Slovenije).

Intention and practical use of the industrial microcomputer controller

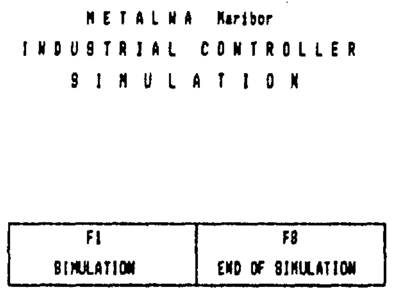
The controller is intended for control in capital equipment facilities plants, where heavy environment conditions and the immense equipment costs do not allow any compromise. A number of unique functions were built in the controller to obtain the required functionality. Special attention was dedicated to the user's program - control application development and verification tools.

The paper decribes a unique part of the user development and verification software which allows controlled application verification at the planner's workplace. This function reduces starting expenses when a capital object is put into work.

With convenient tools, software design methods and simulation, most of the mistakes, bugs and imperfections of the controller's software are discovered and removed before implementation on an object.

Thus, the simulation of the controller's software can be carried out in different ways. Module simulation is specially efficient. It allows that only verified software blocks are put together into larger structures. Typical software modules are functions and sobroutines like structures in high level language, and assembler like macros.

Simple use of the simulation packet is assured with it's **hierarchical tree structured menu**. The user selects from the menu on the screen the actions to be executed. Each action is determined with a function key on the keyboard. The user selects the desired action by pressing the adequate function key on the keyboard. Figure 1 shows an example of this principle.


METALNA Maribor
INDUSTRIAL CONTROLLER
SIMULATION

F1	F8
SIMULATION	END OF SIMULATION

Fig. 1: The initial simulation menu.

After selecting the adequate menu's window of a selected action, it lights up in yellow colour. This light is an advertisement for the user to notice which action he had selected. In the same way, the simulator advertises the user when he returns back over the menues. If there is no special, objective reasons, the selected action is executed immediately.

The user can present his program for the controller eitheir in a graphic form, the so-called **contact networks**, or in textual, **mnemonic form**.

The user must call the analyser before applying the simulation of his program. The analyser is a kind of compiler. It was designed for this special purpose: it translates the user's program together with the declaration module and possibly with some other files for functions or subroutines into the 'object form' which is 'understood' and executed by the simulation, when requested.

The simulation can be executed only when the analyser does not find and report any error(s) in the user's program file, nor in the declaration module's file. A example of both files is on Figure 2a and Figure 2b.

```
 MAIN
   SET   M1
   RESET M2
   BP    5
   ADD   CB12,     CB328,    B1
   SUB   CW7,      CW8,      W16
   MUL   CL323,    CL818,    L234
   W17 = CW17
   DIV   CW111,    CW71,     W12(W17), W20
   BP    9
   AND   W16,      W17,      W19
   CPL   CB12,     B12
   NEG   CB12,     B13
   BP    10
   RLC   CL328,    65,       L235
   SLC   CB115,    5,        B235
   TRANS CB115.3,  3,        L236.7
   TRANS M300,     100,      M250
   Q  =  M1
   JPQ   33
   NOP
33 L237  = L236
   M21   = N ( (M1 A M2) O (W16 LT CW87) )
   BP    20
   NOP
   BP    25
 END.
```

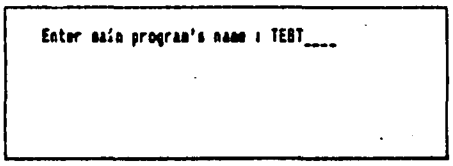
Fig. 2a: The user's program (file TEST.MPR)

```
 DECLAR
 CONST
   CB10    <- 10
   CB12    <- 12
   CB115   <- 115
   CB119   <- 119
   CB123   <- 123
   CB321   <- 30
   CB328   <- 32
   CB378   <- 37
   CW17    <- 17
   CW71    <- 71
   CW73    <- 7300
   CW78    <- 78
   CW87    <- 87
   CW111   <- 111
   CW200   <- 2000
   CW312   <- 312
   CW419   <- 8841
   CW788   <- 788
   CL0     <- 0
   CL5     <- 5
   CL46    <- 46
   CL49    <- 49
   CL64    <- 64
   CL175   <- 175
   CL200   <- 2000000
   CL215   <- 215
   CL287   <- 287
   CL300   <- 3000000
   CL328   <- 328000
   CL818   <- 818000000
 ENDCONST
  IOSPEC
    MOD1 is TY31
    MOD2 is TY31
    MOD3 is TY31
    MOD4 is TY31
 BLOCK
   CS    11, YES,     70
   C     11, M6,      M7,  M8, M61,M62, M63, M64
   CS    12, NO,      80
   C     12, M6,      M7,  M8, M9, M10, M11, M12
   CS    13, YES,     90
   C     13, M10,     M11, M5, M6, M16, M17, M88
   TS    12, 10ms,    YES, 999
   T     12, M2,      M29, M3,      -
   TS    11, 100ms,   NO,  50
   T     11, M1,      M2,  M3,     M4
   TS    13, 1ms,     NO,  100
   T     13, M3,      M4,  M2,     M1
   TXS   23, READ,    2,  "START"
   TX    23, M22,     M3
   TXS   24, WRITE,   2,  "O.K."
   TX    24, M21,     M3
   TXS   25, NUMWRITE, 2, B333
   TX    25, M20,     M3
   TXS   26, NUMREAD,  2, L338
   TX    26, M26,     M3
   MS    13, 100ms,   NO,  333
   M     13, M2,      M20
   MS    14, 10ms,    YES, 238
   M     14, M3,      M27
   MS    15, 100ms,   NO,  669
   M     15, M27,     M84
   DS     8, 1s,      11,  8,   B
   D      8, M2,      M32, M33
   DS    16, 1s,      18,  16,  W
   D     16, M25,     M26, M37
   DS    32, 1s,      13,  32,  L
   D     32, M2,      M21, M30
   RS    16, FIFO,    22,  L
   R     16, M2,      M24, M32, M41, M52
   RS    17, LIFO,    30,  B
   R     17, M2,      M25, M35, M43, M53
   RS    18, FIFO,    20,  W
   R     18, M3,      M55, M55, M53, M57
 ENDBLOCK
 ENDIO
 END.
```

Fig. 2b: The declaration module (file D2.DCM)

NOTE

The user does not need to write the file type (MPR for user's (or main) program or DCM for declaration module), because special purpose editors do this. File types are always hidden from the users!

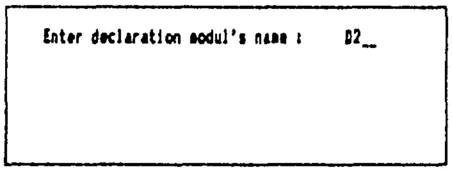


```
Enter main program's name : TEST____
```

Fig. 3a: Reading a name of the main program.

After all neccessary files were translated, the simulation establishes the presence of error(s) very simply. If the analyser finds any error(s), then it does not produce the object file, which is a direct input to the simulation. In case of an error the simulator writes a message on the screen. In this message it tells the user that he can not execute the simulation because the error(s) is(are) found and that the user can correct the error(s) in a corresponding editor.

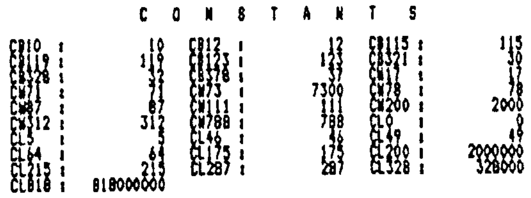
The program **SIMULATION** first of all reads the name of user program's file (or main program's file) and the name of the declaration module. Figures 3a and 3b show the user's answers to both questions.



```
Enter declaration modul's name :    D2_
```

Fig. 3b: Reading the name of the declaration module.

After successful return from the analyser and before the simulation execution, the simulator writes a detailed report about constants and function blocks used in the declaration module (Figures 4a and 4b).

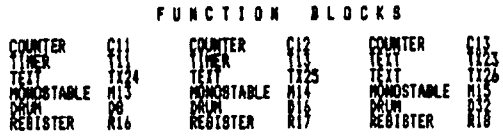


```
                  C  O  N  S  T  A  N  T  S

CB10  :         10  CB12  :        12  CB115 :       115
CB119 :        119  CB123 :       123  CB321 :        30
CB32  :         32  CB37  :        37  CW17  :        17
CW71  :         71  CW73  :      7300  CW78  :        78
CW87  :         87  CW111 :       111  CW200 :      2000
CW312 :        312  CW788 :       788  CL0   :         0
CL5   :          5  CL46  :        46  CL49  :        49
CL64  :         64  CL175 :       175  CL200 :   2000000
CL215 :        215  CL287 :       287  CL328 :    328000
CL818 :  818000000
```

Fig. 4a: The report about CONSTANTS.

If the value of a constant is too large for it's data type, then the simulator assigns the largest positive or negative value to that constant, depended on its sign!
The data type of a constant is appointed by the second letter of the constant's name: **B**, **W** or **L** for byte (8 bits), word (16 bits) or a longword (32 bits).

The data type of an instruction operand identifies how many bits of storage should be considered as a unit and what is the interpretation of that unit to be. The simulator only recognises **the integer, BCD and ASCII data**. An integer can be stored in a bit, byte, word or in longword. Some instructions interpret the integer data as a signed value, while others as a bit strings.



```
              F U N C T I O N   B L O C K S

COUNTER     C11     COUNTER     C12     COUNTER     C13
TIMER       T11     TIMER       T13     TEXT        TX23
TEXT        TX24    TEXT        TX25    TEXT        TX26
MONOSTABLE  M13     MONOSTABLE  M14     MONOSTABLE  M15
DRUM        D8      DRUM        D16     DRUM        D32
REGISTER    R16     REGISTER    R17     REGISTER    R18
```

Fig. 4b: The report about FUNCTION BLOCKS.

1.1 Representation of a types of data and function blocks

The user can use the following types of data:

(a) <u>CONSTANTS</u>
The user assigns initial values to constants in the declaration module. During the execution of the simulation the simulator has read only access to it's value. The user can assign a new value to a constant in the later described menues **SINGLE CHANGE** and **ADJUST EDITOR**.

(b) <u>SYSTEM DATA</u>
The controller's system data are set up by the operating system. The simulator simulates this function by assigning them values. It can read only system data. The same principle to modify the system data is in valid as for constants.

(c) <u>INTERNAL DATA</u>
Internal data are general purpose data. The simulator can use them similarly as variables in a high level language: their values can be modified by the simulator.

(d) <u>INPUT DATA</u>
Input data are external physical inputs into the controller.

(e) <u>OUTPUT DATA</u>
Output data are physical outputs out from the controller.

We use **'the single assignment rule'** for all types of data, because of the simulation of parallel execution mutualy exclusive events.

Overview of the controller's data types

The user can use almost all combinations of types of data with data types in simulation. This survey is shown on Figure 5.

TYPE OF DATA	TYPE's DATA NAME	TYPE's DATA MARK	DATA TYPE's NAME	DATA TYPE's MARK	A SAMPLE
CONSTANT DATA:					
BYTE	Constant	C	Byte	B	CB18
WORD	Constant	C	Word	W	CW555
LONGWORD	Constant	C	Longword	L	CL234
SYSTEM DATA:					
BIT	System	S	Bit	*	S12
BYTE	System	S	Byte	B	SB373
WORD	System	S	Word	W	SW383
LONGWORD	System	S	Longword	L	SL947
INTERNAL DATA:					
BIT	Internal	*	Bit	M	M991
BYTE	Internal	*	Byte	B	B3
WORD	Internal	*	Word	W	W495
LONGWORD	Internal	*	Longword	L	L952
OUTPUT DATA:					
BIT	Output	O	Bit	*	O12.13
BYTE	Output	O	Byte	B	OB10.4
WORD	Output	O	Word	W	OW4.10
LONGWORD	Output	O	Longword	L	OL2.2
INPUT DATA:					
BIT	Input	I	Bit	*	I5.13
BYTE	Input	I	Byte	B	IB8.10
WORD	Input	I	Word	W	IW10.0
LONGWORD	Input	I	Longword	L	IL1.3

NOTE

Asterisk "*" means that at this place there is no mark!

Fig. 5: Survey about types of data.

The user has six **types of function blocks** besides the types of data mentioned above. The function blocks are:

(a) <u>TIMER</u>
The timer enables temporal control over events in an object. After a certain time delay something can happen, the value of which is programmable.

(b) MONOSTABLE
The monostable enables temporal control, too. It generates a pulse of specific duration, the value of which is programmable.
The main difference between the monostable and the timer is the following: the user can programmably control the timer through its inputs. After the user had enabled the monostable to start running, he can no longer programmably influence the monostable. Only one exception is allowed: the user can repeatedly start the monostable from the beginning!

(c) COUNTER
The counter permits the upcounting and downcounting of events. These two operations can be performed simultaneousy or not, as required.

(d) DRUM CONTROLLER
The drum controller enables temporal or event-driven (through its inputs) control: values of output bits of current drum step are assigned to actual bits. The two mentioned modes of operation are mutually exclusive.

(e) REGISTER
The register enables storage of data in two different ways:
* **FIFO** stack or
* **LIFO** queue.

(f) TEXT
The text enables simple input/output operations (communication between the user and the controller).

1.2 Simulation of a user's program execution

The simulation receives the user's program merged together with other files in object code on a file. The file has sequential organization. It consists of records arranged in the sequence in which they are written in the file (the first record written is the first record in the file, ... and so on).

Particular instruction needs more records. Records of the same instruction are always arranged in this way: a first record contains an operand which will have a result (one or two for division), a following record is a second operand, if it indeed exists in syntax of an instruction. After operands, if the instruction has any, comes the operator. This is an instruction which will be executed.

The simulator reads the records in the described regular sequence, too. Simulation of execution is based on the principle of **stack computer**. The simulator reads a record from the object code's file. Records are already in correct sequence, in so-called reverse Polish notation. The content of a record is either an operator or an operand.

If it is **an operand** then the simulator pushes it on the stack.
If it is **an operator** then the simulator pulls the corresponding number of operands from the stack, executes the operator (instruction) and assigns a value to a result.

The **IMCL** (Industrial Microcomputer Controller Language) is a mnemonic programmable language for our controller. The user can simulate all of the instructions of the **IMCL**:

ARITHMETIC OPERATIONS:
(1) ADD - arithmetic addition
(2) SUB - arithmetic substraction
(3) DIV - arithmetic division
Divide by zero:
The simulator assigns the largest positive or negative value to the result, dependebly on the numerator sign, a zero to the remainder and reports the overflow of the result.
The simulator writes the values of condition flags (Negative, Zero, oVerflow and Carry) on the screen, then follow the messages about the mode of execution and about the current number of cycles - reiterations of execution of the user's program.
In case of dividing by zero the simulator always breaks execution and writes a message. After the message the user can continue with the execution of his program. He must press the key **RUN** - Figure 6!
(4) MUL - arithmetic multiplication

BIT OPERATIONS:
(5) A - logical AND operation over a bit's expressions
(6) O - logical OR operation over a bit's expressions
(7) N - negation of a bit's expression
(8) SET - bit set
(9) RESET - reset bit
(10) P - protection and assignement

BIT OPERATIONS BETWEEN TERMS
(8, 16 or 32 bit string's length):
(11) OR - logical OR operation
(12) XOR - logical XOR operation
(13) AND - logical AND operation
(14) NAND - logical NAND operation
(15) NOR - logical NOR operation

COMPLEMENTS:
(16) NEG - one's complement
(17) CPL - two's complement

TRANSFER OF BIT STRING:
(18) SLC - shift left
(19) SRC - shift right
(20) RLC - rotate left
(21) RRC - rotate right
(22) TRANS - general purpose transfer of bits between bit strings

RELATIONAL OPERATORS:
(23) NE - operands are not equal ?
(24) EQ - are both operands equal ?
(25) LT - first operand is less than second one
(26) LTE - first operand is less than or equals to the second one
(27) GT - first operand is greater than second one
(28) GTE - first operand is greater than or equals to the second one

CONVERSIONS:
(29) CBIN - conversion from BCD to two's complement
(30) CBCD - conversion from two's complement to BCD

CONTROL OPERATIONS:
(31) JPQ - jump if Q bit is equal 1
(32) JPnotQ - jump if Q bit is equal 0
(33) JPX - jump if X bit is equal 1

(34) JPnotX - jump if X bit is equal 0
(35) JP - unconditional jump
The simulator writes a message to the user that the next step will be to execute a labelled program statement corresponding to the label of the JUMP statement. Jump skips the statements between JUMP instruction and this statement!
The simulator writes this message only in step-wise mode of execution!

CALL OPERATIONS:
(36) CALLM - calling a module
(37) CALLQ - conditional calling a module

MISCELLANEOUS OPERATIONS:
(38) BP - break point
When the simulator reaches a break point that it is not cancelled out in the user's program, it breaks te execution of simulation. The simulator writes a message about the break point irrespective of the mode of execution: step-wise mode or continous mode.
(39) EQUAL - assignement an operand's value to a result
(40) NOP - no operation
In the step-wise mode of execution the simulator writes only a message that it reached the NOP instruction and reassigns all condition flags to zero.

Instructions are **orthogonal** which means that the user can use the same instruction with different data types. For example: once with a byte, some other time with a longword.

Internal types of data, which are longer than one bit, we can **address** also in **index mode** and **indirect (deferred mode)**;
for example: **ADD (W3), W4(W6), W3**.
In such cases the value of **indirectly addressed internal variable** tells the simulator on which internal datum (variable) the instruction will be actually executed, or in **index mode of addressing**, (indexed variable is within round brackets) a sum of both internal variables' values gives index (**address**) of actual internal datum.

1.3 Representation of a user's program execution

You can repeatedly execute the simulation of yours program as many times as you like. Every time you can choose one mode from the following **modes of execution**:
(a) more cycles,
(b) single cycle,
(c) step-wise mode,
(d) continous mode,
(e) with break point(s) or
(f) without break point(s).

Somê modes of execution are compatible with others. The user can, for example, execute the simulation in step-wise mode, with break points and has more cycles. One cycle is one iteration of the user's program execution.

The key **RUN** (Figure 6) enables a **commencement** or a **continuation** of user's program execution.

The **step-wise mode of execution** enables the user's program execution step by step, one instruction after another. For particular instructions a message is written to the screen. The message contains rudimentary informations: which instruction is executed, operand names and values and values of conditional flags (Negative, Zero, oVerflow and Carry).

The simulation can start with the following default modes of execution: a single cycle, continous mode and with break points! If the user does not choose the step-wise mode then the entire user's program is executed at least once (depended on the number of cycles - iterations of execution!) without the simulator writing out any message, except if there is a run time error or a break point instruction or a jump instruction!

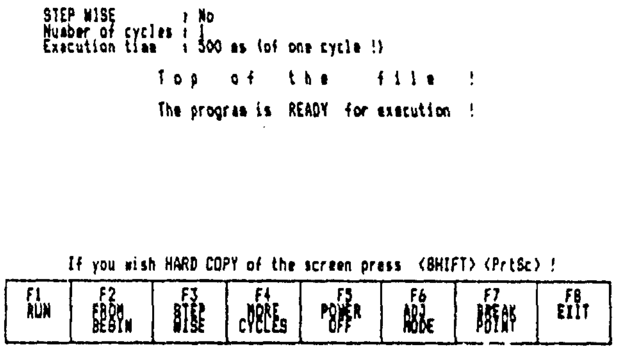

```
STEP WISE        : No
Number of cycles : 1
Execution time   : 300 ms (of one cycle !)

        T o p   o f   t h e   f i l e   !

        The program is  READY  for execution  !

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !
```

Fig. 6: A fundamental menu of sim' lation.

The user selects either the step-wise mode or the oposite continous mode, with tne key **STKP WISE** (Figure 6). The step-wise mode and the continous mode are mutually exclusive. A new state is oposite to a previous state. At commencement of execution **the default state** is the **continous mode**, so if the user wishes the step-wise mode, he must press the key **STEP WISE** (Figure 6).

The key **MORE CYCLES** permits to inscribe the value of the number of cycles (Figure 7).

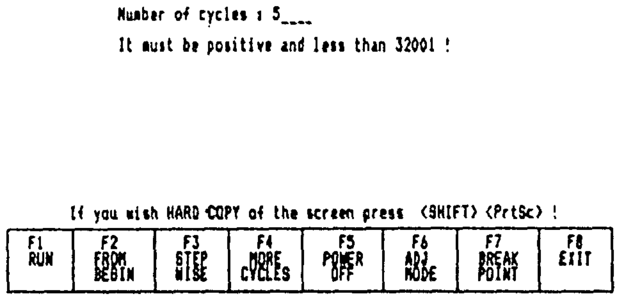

```
Number of cycles : 5____

It must be positive and less than 32001 !

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !
```

Fig. 7: The user inscribes the value of the number of cycles.

The user can cancel out all of the used **break points** or just some of them, or gives them active status, if he chooses a menu of break points (the key **BREAK POINT**), which is shown on Figure 8.

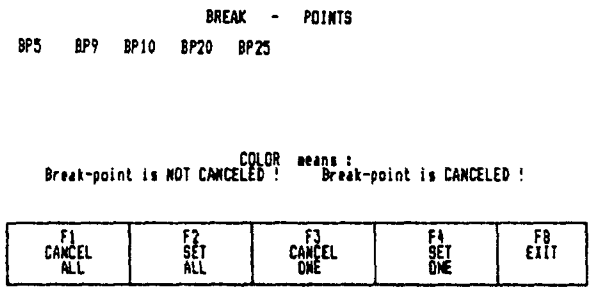

```
                 BREAK  -  POINTS

BP5   BP9   BP10   BP20   BP25

                    COLOR  means :
   Break-point is NOT CANCELED !    Break-point is CANCELED !
```

Fig. 8: A menu of break points.

If break point (BP) is cancelled out, then the execution of simulation is not broken!

Before executing an instruction, the simulator verifies if the neccessary operands have values. If they do not have, the simulator calls the user's attention to this fact. The user can choose either to inscribe an initial value to a datum or to confirm a simulation's proposal to assign a zero value to that datum.

The simulation enables all inscriptions of digits (numerical data) in:

(a) binary number system,
(b) octal number system,
(c) decimal number system and
(d) hexadecimal number system.

The user indicates the desired number system by inscribing the first character: **B**, **O** or **H** for binary, octal or hexadecimal number system. For decimal number system he does not inscribe any letter before digits!

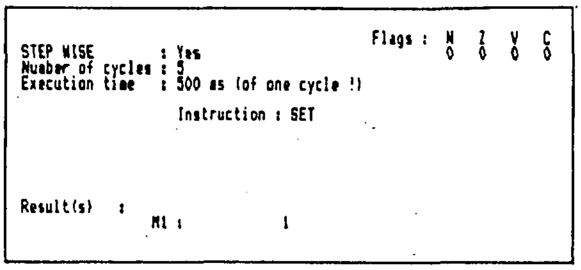
During the inscription of an integer number, the simulator ignores the prohibited characters. For example: letters that do not have sense for the <u>selected number system</u>, or letters the values of which are larger than the basis of the number system decremented by one. Likewise, the simulator reports an error if the numerical value exceeds the allowed integer interval of datum's data type (a bit, a byte, a word or a longword).

During execution (depending on the selected modes of execution and menues) the user can make hard-copy of the screen, use **ADJUST MODE** that permits an overview and adjustment of used data and function blocks, select different modes of execution, cancel out or not cancel out break point(s) and, if he wishes, he can terminate the simulation. This function is enabled by the key **EXIT**, <u>that always brings us into the previous menu</u>.

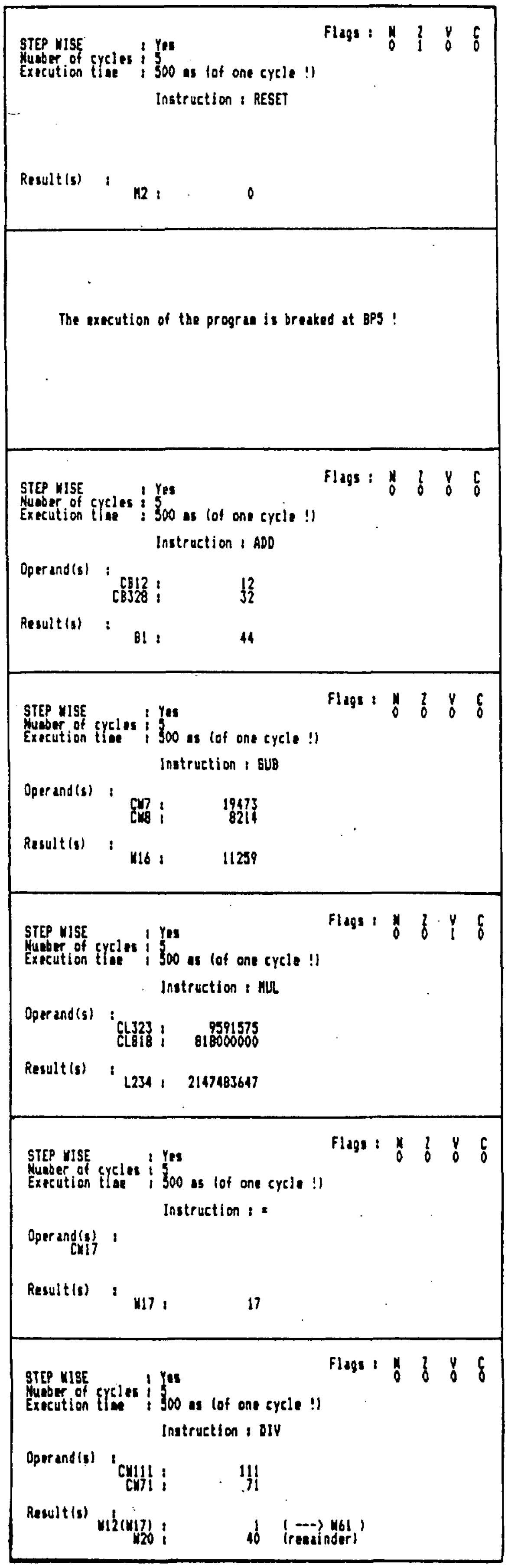
The simulation permits the user to repeatedly execute different user's programs or combinations of the same user's program with different declaration modules.

2 An instance of step-wise execution of a simple user's program

Figure 9 shows an example of step-wise execution of a simple user's program, the source mnemonic form of which was presented on Figure 2a. We cut off the fundamental menu (presented on Figure 6) in order to spare space. Each message is written above this menu. Temporal order of messages is from top to the bottom of a paper.



```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : SET

Result(s)    :
               M1 :           1
```



```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   1   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : RESET

Result(s)    :
               M2 :           0
```

```
   The execution of the program is breaked at BP5 !
```

```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : ADD

Operand(s)   :
             CB12 :           12
            CB328 :           32

Result(s)    :
               B1 :           44
```

```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : SUB

Operand(s)   :
              CW7 :        19473
              CW8 :         8214

Result(s)    :
              W16 :        11259
```

```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   0   1   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : MUL

Operand(s)   :
            CL323 :      9591575
            CL818 :    818000000

Result(s)    :
             L234 :   2147483647
```

```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : =

Operand(s)   :
      CW17

Result(s)    :
              W17 :           17
```

```
                                              Flags :  N   Z   V   C
STEP WISE         : Yes                                0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                    Instruction : DIV

Operand(s)   :
            CW111 :          111
             CW71 :           71

Result(s)    :
         W12(W17) :            1   ( ---> W61 )
              W20 :           40   (remainder)
```

```
The execution of the program is breaked at BP9 !
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : AND

Operand(s)   :
               W16 : 00101011 11111011
               W17 : 00000000 00010001

Result(s)    :
               W19 : 00000000 00010001
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              1   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : CPL

Operand(s)   :
               CB12 : 00001100

Result(s)    :
                B12 : 11110100
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              1   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : NEG

Operand(s)   :
               CB12 : 00001100

Result(s)    :
                B13 : 11110011
```

```
The execution of the program is breaked at BP10 !
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   1   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : RLC

Operand(s)   :
               CL328 : 00000000 00000101 00000001 01000000
          Rotate no.  : 65 (  => 1 !!! )

Result(s)    :
                L235 : 00000000 00001010 00000010 10000000
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : SLC

Operand(s)   :
               CB115 : 01110011
          Shift no.  : 5

Result(s)    :
                B235 : 01100000
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : TRANS

Operand(s)   :
               CB115 : 01110011
         No. of bits : 3

Result(s)    :
                L236 : 00000010 00010101 01000111 00111011
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : TRANS

Operand(s)   :
               M300 : 00000000 00000000 00000000 00000000  ...
        No. of bits : 100

Result(s)    :
               M250 : 00000000 00000000 00000000 00000000  ...
```

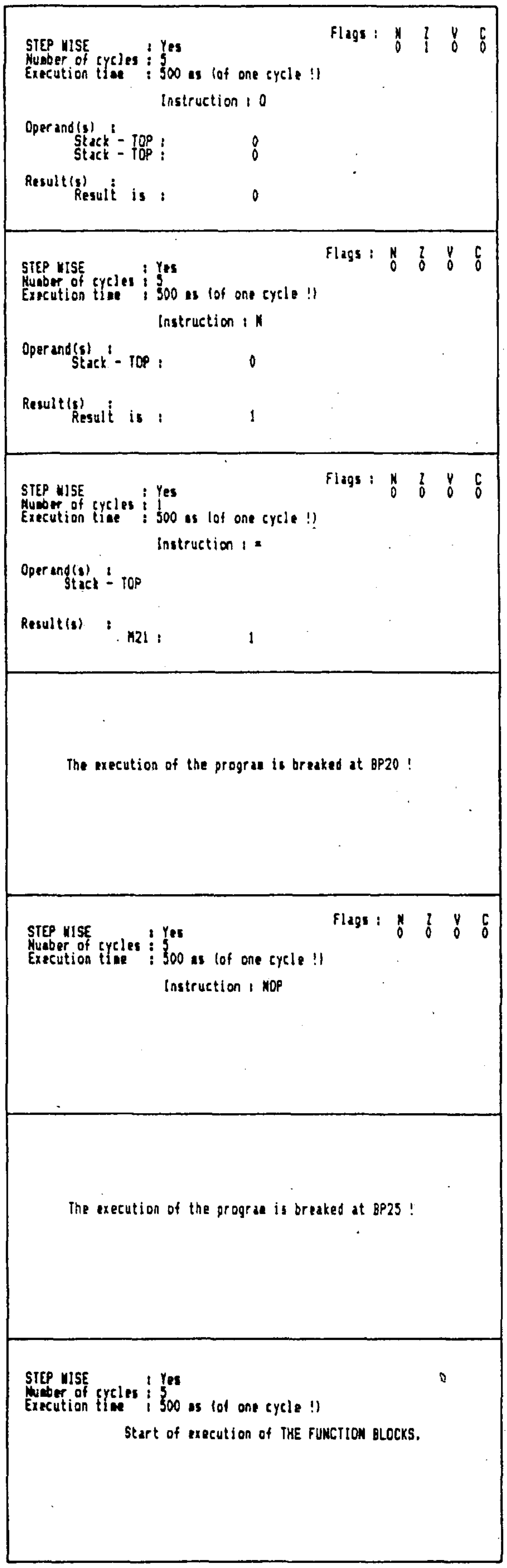
```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : =

Operand(s)   :
        M1

Result(s)    :
               Q :              1
```

```
The execution of the program is breaked at LABEL 33 !
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   0   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : =

Operand(s)   :
        L236

Result(s)    :
               L237 :       34948923
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   1   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : A

Operand(s)   :
               M2 :             0
               M1 :             1

Result(s)    :
        Result  is  :           0
```

```
                                            Flags :  N   Z   V   C
STEP WISE         : Yes                              0   1   0   0
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)

                   Instruction : LT

Operand(s)   :
               W16 :        11259
              CW87 :           87

Result(s)    :
        Result  is  :           0
```



```
STEP WISE         : Yes                        Flags :  N  Z  V  C
Number of cycles  : 5                                   0  1  0  0
Execution time    : 500 ms (of one cycle !)

                  Instruction : O

Operand(s)  :
        Stack - TOP :          0
        Stack - TOP :          0

Result(s)   :
        Result  is  :          0
```

```
STEP WISE         : Yes                        Flags :  N  Z  V  C
Number of cycles  : 5                                   0  0  0  0
Execution time    : 500 ms (of one cycle !)

                  Instruction : N

Operand(s)  :
        Stack - TOP :          0

Result(s)   :
        Result  is  :          1
```

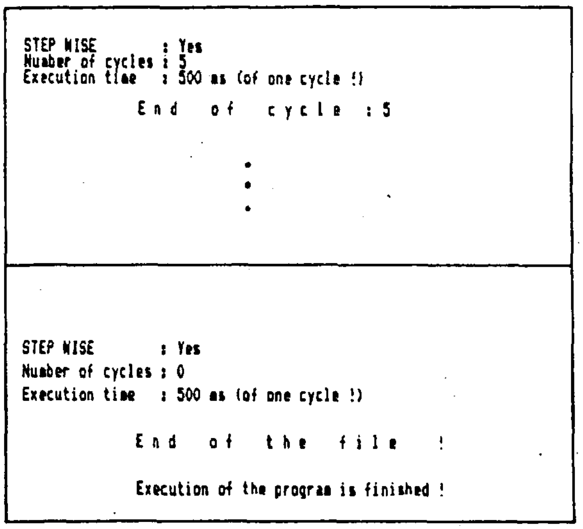
```
STEP WISE         : Yes                        Flags :  N  Z  V  C
Number of cycles  : 1                                   0  0  0  0
Execution time    : 500 ms (of one cycle !)

                  Instruction : =

Operand(s)  :
       Stack - TOP

Result(s)   :
           M21 :               1
```

```
The execution of the program is breaked at BP20 !
```

```
STEP WISE         : Yes                        Flags :  N  Z  V  C
Number of cycles  : 5                                   0  0  0  0
Execution time    : 500 ms (of one cycle !)

                  Instruction : NOP
```

```
The execution of the program is breaked at BP25 !
```

```
STEP WISE         : Yes
Number of cycles  : 5
Execution time    : 500 ms (of one cycle !)
             Start of execution of THE FUNCTION BLOCKS.
```



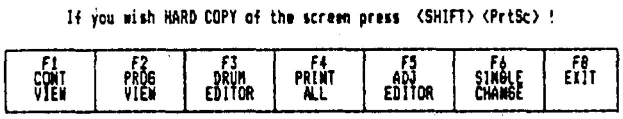
```
STEP WISE         : Yes
Number of cycles  : 5
Execution time    : 500 ms (of one cycle !)
            E n d   o f   c y c l e  : 5

                      .
                      .
                      .
```

```
STEP WISE         : Yes
Number of cycles  : 0
Execution time    : 500 ms (of one cycle !)

          E n d   o f   t h e   f i l e   !

          Execution of the program is finished !
```

Fig. 9: A step-wise mode of execution of a simple user's program.

3 Adjustment and writing out values

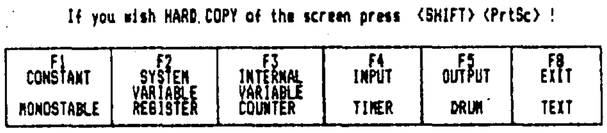
In the fundamental simulation menu (Figure 6), the user must select **ADJUST MODE** (Figure 10). The simulator permits two modes of adjusting and writing out data.



```
If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|  F1  |  F2  |  F3    |  F4   |  F5    |  F6    |  F8  |
| CONT | PROG | DRUM   | PRINT | ADJ    | SINGLE | EXIT |
| VIEW | VIEW | EDITOR | ALL   | EDITOR | CHANGE |      |
```

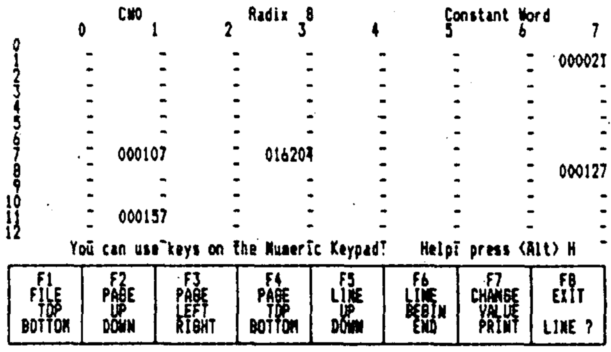
Fig. 10: A menu of ADJUST MODE.

The first mode is directed to a particular datum or function block and so we call it **SINGLE CHANGE** (Figure 11). The second mode is directed to all values of the same type of data - **ADJUST EDITOR** (Figure 12).



```
If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|     F1     |    F2    |    F3    |  F4   |  F5    |  F8  |
|  CONSTANT  |  SYSTEM  | INTERNAL | INPUT | OUTPUT | EXIT |
|            | VARIABLE | VARIABLE |       |        |      |
| MONOSTABLE | REGISTER | COUNTER  | TIMER | DRUM   | TEXT |
```

Fig. 11: A menu of SINGLE CHANGE.



```
          CW0              Radix  8             Constant Word
     0      1        2        3        4        5       6       7
 0
 1   -      -        -        -        -        -       -   000021
 2   -      -        -        -        -        -       -       -
 3   -      -        -        -        -        -       -       -
 4   -      -        -        -        -        -       -       -
 5   -      -        -        -        -        -       -       -
 6   -      -        -        -        -        -       -       -
 7   -   000107      -     016204      -        -       -       -
 8   -      -        -        -        -        -       -   000127
 9   -      -        -        -        -        -       -       -
10   -      -        -        -        -        -       -       -
11   -   000157      -        -        -        -       -       -
12   -      -        -        -        -        -       -       -
     You can use keys on the Numeric Keypad!    Help! press <Alt> H

|  F1    |  F2  |  F3   |  F4    |  F5  |  F6   |  F7    |  F8    |
|  FILE  | PAGE | PAGE  | PAGE   | LINE | LINE  | CHANGE | EXIT   |
|  TOP   | UP   | LEFT  | TOP    | UP   | BEGIN | VALUE  |        |
| BOTTOM | DOWN | RIGHT | BOTTOM | DOWN | END   | PRINT  | LINE ? |
```

Fig. 12: A menu of ADJUST EDITOR.

The way of selection is identical for both modes, because it runs through similar menues. The user selection starts on menu of types of data (Figure 13a). There are constants, internal data, system data, input and ouput data. The user chooses between a bit, a byte, a word and a longword in the next menu of data types (Figure 13b). The user can choose all data types for any selected type of data except a bit for constants, because the simulator does not have a constant bit!

```
If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|    F1    |    F2    |    F3    |   F4   |   F5   |  F8  |
| CONSTANT |  SYSTEM  | INTERNAL | INPUT  | OUTPUT | EXIT |
|          | VARIABLE | VARIABLE |        |        |      |
```

Fig. 13a: Menu of types of data.

```
If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|  F1  |  F2   |  F3   |    F4     |  F8  |
| BITS | BYTES | WORDS | LONGWORDS | EXIT |
```

Fig. 13b: Menu of data types.

3.1 Adjustment and writing out values of function blocks' data

The menu **SINGLE CHANGE** (Figure 11) enables adjustment and writing out values of function blocks' data.

After selecting the type of a function block, the user inscribes its successive number and then, if it actually exists, the simulator draws its picture on the screen. The simulator then writes important information about it: type of function block, its successive number, names and values of its input and output bits and current values of its typical variables. In a menu, under each drawing, the user can read which variables he can adjust and what actions he can use (Figures 14a, 14b, 14c, 14d, 14e and 14f).

```
                      ---------- M 13 ----------
S (M2)     = 0 -|  TB              :  0,100 s        |- R (M20)   = 0
                |  PRESET  VALUE   :  333            |
                |  MODIF           :  NO             |
                |  CURRENT VALUE   :  0              |
                 ------------------------------------

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|    F1     |      F2       |      F3      |  F8  |
| TIME BASE | CURRENT VALUE | PRESET VALUE | EXIT |
```

Fig. 14a: A menu of MONOSTABLE.

```
                      ---------- R 16 ----------
R (M2)     = 0 -|  Register type is FIFO - queue.     |- F (M41)   = 0
I (M24)    = 0 -|  INPUT  DATA     :  0               |- E (M52)   = 1
O (M32)    = 0 -|  OUTPUT DATA     :  0               |
                |  REGISTER LENGTH :  22              |
                |  NUMBER OF DATA  :  0               |
                |  Data type is longword.             |
                 -------------------------------------

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|      F1     |        F2       |    F8    |
| INPUT  DATA | REGISTER LENGTH |   EXIT   |
| OUTPUT DATA | NUMBER OF DATA  | CONTENTS |
```

Fig. 14b: A menu of REGISTER.

```
                      ---------- C 12 ----------
R (M6)     = 0 -|  PRESET  VALUE  :  80              |- E (M10)   = 0
P (M7)     = 0 -|  MODIF          :  NO              |- D (M11)   = 0
U (M8)     = 0 -|  CURRENT VALUE  :  0               |- F (M12)   = 0
D (M9)     = 0 -|                                    |
                 ------------------------------------

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|      F1       |      F2      |  F8  |
| CURRENT VALUE | PRESET VALUE | EXIT |
```

Fig. 14c: A menu of COUNTER.

```
                      ---------- T 11 ----------
E (M1)     = 0 -|  TB              :  0,100 s        |- D (M3)   = 0
C (M2)     = 0 -|  PRESET  VALUE   :  50             |- R (M4)   = 0
                |  MODIF           :  NO             |
                |  CURRENT VALUE   :  0              |
                 ------------------------------------

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|    F1     |      F2       |      F3      |  F8  |
| TIME BASE | CURRENT VALUE | PRESET VALUE | EXIT |
```

Fig. 14d: A menu of TIMER.

```
                      ---------- D 32 ----------
R (M2)     = 0 -|  TB                 :  1,000 s                 |- F (M30)   = 0
U (M21)    = 0 -|  PRESET  VALUE      :  13                      |
                |  CURRENT VALUE      :  0                       |
                |  NUMBER OF STEPS    :  32                      |
                |  CURRENT STEP       :  0                       |
                |  Step logic STATES  :                          |
                |  00000000 00000000 00000000 00100000           |
                 ------------------------------------------------

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|  F1  |    F2   |   F3   |    F4    |    F5   |  F8  |
| TIME | CURRENT | PRESET |  NUMBER  | CURRENT | EXIT |
| BASE |  VALUE  | VALUE  | OF STEPS |  STEP   |      |
```

Fig. 14e: A menu of DRUM.

```
                      ---------- TEXT 24 ----------
S (M21)    = 0 -|  INPUT/OUTPUT  :  WRITE STRING     |- D (M3)   = 0
                |                                    |
                 ------------------------------------

Your message is O.K. .

If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !

|        F1         |   F2    |  F8  |
| INTERNAL VARIABLE | MESSAGE | EXIT |
|      UNIT  1      | UNIT  2 |      |
```

Fig. 14f: A menu of TEXT.

3.2 Adjustment and writing out values of data

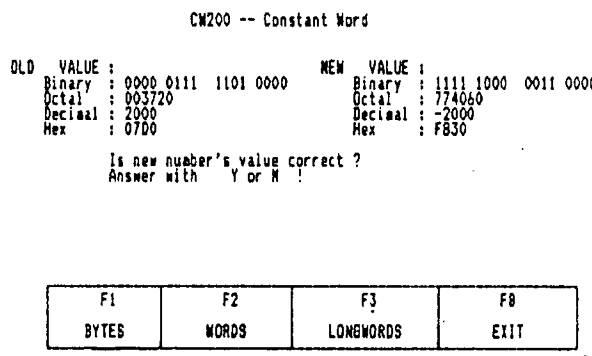
As mentioned above, the user can adjust and write out values of data in two places in our simulation.

3.3 SINGLE CHANGE

After selecting type of data and its data type, the user inscribes its successive number in menu **SINGLE CHANGE**. If the entered name is correct in the context of Figure 5 and has a value, then the simulator writes out its value in binary, octal, decimal and hexadecimal number system. Now, the user can adjust the value or return to the menu of data types. An instance of choosing and adjusting of a datum **CW200** in menu **SINGLE CHANGE** is shown on fugures 15a and 15b.

```
Constant Word NUMBER : 200__
It must be greater than -1 and less than 1000 !

|  F1   |  F2   |    F3     |  F8  |
| BYTES | WORDS | LONGWORDS | EXIT |
```

Fig. 15a: Choosing a datum.



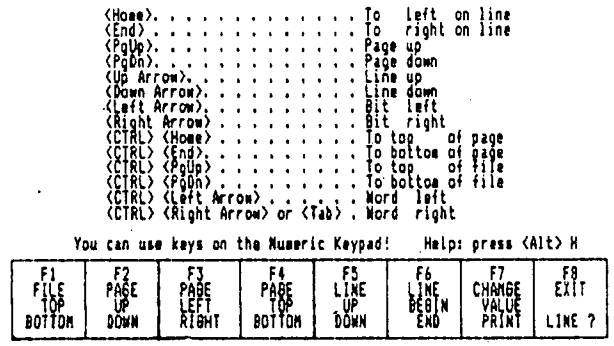
```
                    CW200 -- Constant Word

OLD   VALUE :                             NEW   VALUE :
   Binary  : 0000 0111  1101 0000            Binary  : 1111 1000  0011 0000
   Octal   : 003720                          Octal   : 774060
   Decimal : 2000                            Decimal : -2000
   Hex     : 07D0                            Hex     : F830

           Is new number's value correct ?
           Answer with    Y or N  !
```

F1	F2	F3	F8
BYTES	WORDS	LONGWORDS	EXIT

Fig. 15b: Adjusting the same datum.

3.3.1 ADJUST EDITOR

ADJUST EDITOR is a screen editor that enables overlooking of all values of a selected type of data. We can not usually write all the values on a screen at the same time. We can see those values which are not momentary on the screen by moving the window through a table of values (we called it a file sometimes). Function keys of the menu of **ADJUST EDITOR** enable this moving (Figure 16).



```
<Home). . . . . . . . . . . . . . To   Left  on line
<End> . . . . . . . . . . . . . . To   right on line
<PgUp>. . . . . . . . . . . . . . Page up
<PgDn>. . . . . . . . . . . . . . Page down
<Up Arrow>. . . . . . . . . . . . Line up
<Down Arrow>. . . . . . . . . . . Line down
<Left Arrow>. . . . . . . . . . . Bit  left
<Right Arrow> . . . . . . . . . . Bit  right
<CTRL> <Home> . . . . . . . . . . To top    of page
<CTRL> <End>. . . . . . . . . . . To bottom of page
<CTRL> <PgUp> . . . . . . . . . . To top    of file
<CTRL> <PgDn> . . . . . . . . . . To bottom of file
<CTRL> <Left Arrow> . . . . . . . Word  left
<CTRL> <Right Arrow> or <Tab> . Word  right

You can use keys on the Numeric Keypad!    Help: press <Alt> H
```

F1	F2	F3	F4	F5	F6	F7	F8
FILE TOP BOTTOM	PAGE UP DOWN	PAGE LEFT RIGHT	PAGE TOP BOTTOM	LINE UP DOWN	LINE BEGIN END	CHANGE VALUE PRINT	EXIT LINE ?

Fig. 16: The possibilities of ADJUST EDITOR.

We can move
* to top of a file and
* to bottom of a file.

We can move window
* up,
* down,
* left and
* right.

Within a window we can move
* to top of a page and
* to bottom of a page.

We can move a line
* up and
* down

and inside of the line
* to left on line (beginning of line),
* to right on line (end of line)

and from one datum to another
* a datum left and
* a datum right.

We can move out of a current window only by moving a window, because by moving within a window we can not come out of it.

Such moving is often time-consuming and as we wished to increase the efficincy of the **ADJUST EDITOR**, we added the above described possibilities still another one: the movement to a datum required by the user (the key **LINE ?** on Figure 16).

Besides these, function keys, we can also use the keypad's keys for the same purpose.

The key **CHANGE VALUE** permits adjustment of values and the key **PRINT** enables writing out all values of the selected type of data, clearly arranged on the printer.

4 Conclusion

The presented simulation packet is written in the programming language Turbo Pascal. The user can execute it on all personal computers **IBM XT/AT** type, or compatible, equiped with an operating system **MSDOS**.

The simulation packet is a composition of twelve independent programs, which are connected with standard procedure Execute. The number of source program lines exceeds 50000, the object code is more than 360k bytes of lenght.

The **SIMULATOR** is an integral part of the user's development tools. Consequently, the entire project offers the special purpose environments to the users.

5 References:

* The industrial microcomputer controller's architecture (in slovene), Metalna Maribor, 1987,

* V. Žumer et al., Workstation for software development, Technical report - in slovene, 1986

* V. Žumer et al., Workstation for software development, Technical report - in slovene, 1987

* Programmable Contròllers SIMATIC S5 (different variants), SIMENS, 1984

* Programmable Controllers TSX (different variants), Telemecanique, 1986

* R.E Fairley, Software Engineering Concepts, McGraw-Hill Book Company, 1985

* VAX Architecture Handbook, Digital Equipment Corporation, 1981