

# APPLICATION NOTE

OCTOBER 1982

NA-023A

## **Programming the EF9365 / EF9366 graphic display processor using the MPL language**

Philippe THOMAS  
Laboratoire d'Applications



**THOMSON-EGCIS**  
Integrated Circuits

Programming the EF9365/9366 graphic display processor (GDP) in assembler language may be, in some applications very long and dull. Thus the possibility of using a high-level language is of interest for more than one reason, leading to reduced programming, maintenance and debugging times. Various high-level languages may be used with the EF6800/EF6809 microprocessors, including MPL which is derived from PL/1 and is perfectly adapted to programming the peripheral circuits of this family.

The programming examples in this Application Note show the use of this circuit with the MPL compiler. The reader should be familiar with the EF9635/EF9366 Technical Manual and with the MPL Reference Manual.

### THE EF9365/EF9366 GRAPHIC DISPLAY PROCESSOR

The EF9365/EF9366 is a true graphic display processor offering a high degree of flexibility in use through the possibility for direct interfacing to any 8-bit microprocessor and the provision of 11 internal registers. The two versions available cover various display resolutions :

EF9365 : 512 x 512 (interlaced scanning)  
          256 x 256  
          128 x 128 (non-interlaced scanning)  
          64 x 64

EF9366 : 512 x 256 (non-interlaced scanning).

A programming-oriented functional schematic of the graphic display processor, subdivided into five subsystems, is given in Figure 1. The distribution of the 11 registers within the addressable memory space of the microprocessor is defined in Table 1.

#### DECODING AND CONTROL

The decoding and control subsystem comprises four registers. The command register CMD (see Table 2) transmits all commands to the graphic display processor : plot vector, write character, verify processor status, etc... The status register is accessible in read mode only and contains processor status and interrupt flags (see Table 3).

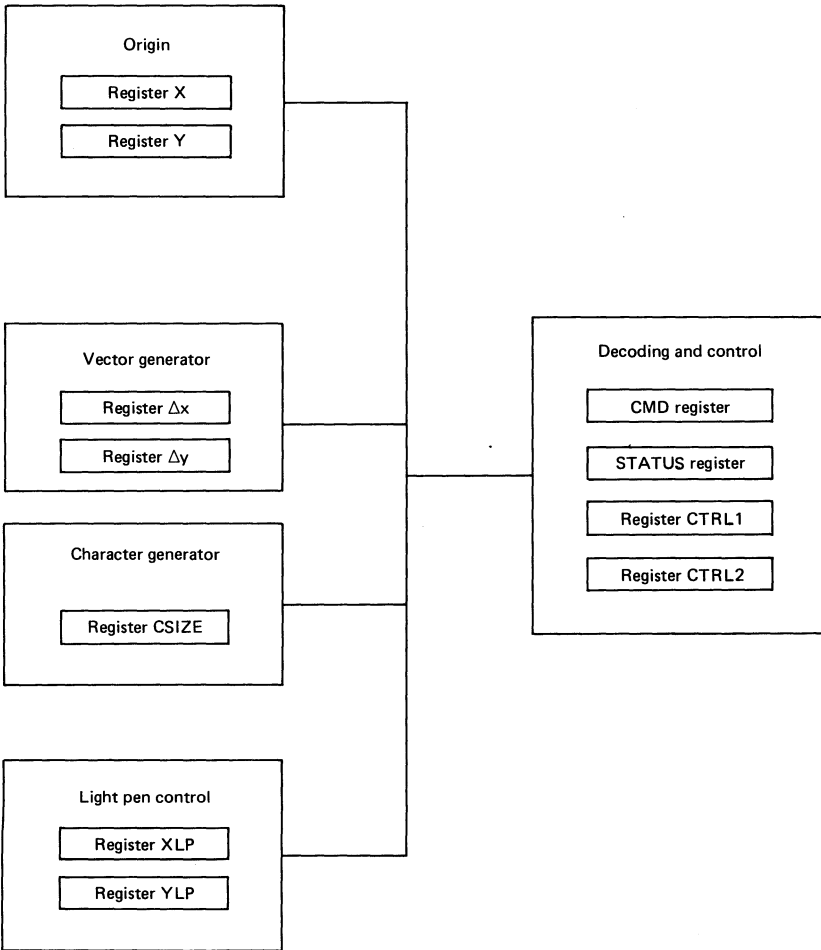


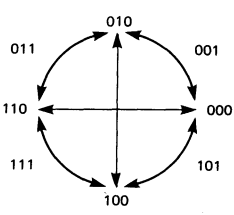
FIGURE 1 - FUNCTIONAL SCHEMATIC OF EF9365/EF9366 INTERNAL REGISTERS

TABLE 1 - REGISTER ADDRESS

ADDRESS REGISTER					REGISTER FUNCTIONS		Number of bits
Binary				Hexa	Read R/W = 1	Write R/W = 0	
A3	A2	A1	A0				
0	0	0	0	0	STATUS	CMD	8
0	0	0	1	1	CTRL 1 (Write control and interrupt control)		7
0	0	1	0	2	CTRL 2 (Vector and symbol type control)		4
0	0	1	1	3	CSIZE (Character size)		8
0	1	0	0	4	Reserved		—
0	1	0	1	5	DELTA X		8
0	1	1	0	6	Reserved		—
0	1	1	1	7	DELTA Y		8
1	0	0	0	8	X MSBs		4
1	0	0	1	9	X LSBs		8
1	0	1	0	A	Y MSBs		4
1	0	1	1	B	Y LSBs		8
1	1	0	0	C	XLP (Light-pen)	Reserved	7
1	1	0	1	D	YLP (Light-pen)	Reserved	8
1	1	1	0	E	Reserved		—
1	1	1	1	F	Reserved		—

Reserved : These addresses are reserved for future versions of the circuit. In read mode, output buffers D0-D7 force a high state on the data bus.

TABLE 2 - COMMAND REGISTER

b7 b6 b5 b4	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1																																									
	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0																																									
b3 b2 b1 b0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F																										
0 0 0 0	0	Set bit 1 of CTRL 1 : Pen selection	Vector generation (for b2, b1, b0 see small vector definition)	SPACE	0	@	P	`	p	<b>SMALL VECTOR DEFINITION :</b>  <table border="1" style="margin: 10px auto;"> <tr> <td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td> </tr> <tr> <td>1</td><td> \Delta X </td><td> \Delta Y </td><td colspan="5">Direction</td> </tr> </table> <b>Dimension</b>  <table border="1" style="margin: 10px auto;"> <tr> <td>\Delta X or \Delta Y</td><td>Vector length</td> </tr> <tr> <td>0 0</td><td>0 step</td> </tr> <tr> <td>0 1</td><td>1 step</td> </tr> <tr> <td>1 0</td><td>2 steps</td> </tr> <tr> <td>1 1</td><td>3 steps</td> </tr> </table> <b>Direction</b>  							b7	b6	b5	b4	b3	b2	b1	b0	1	\Delta X	\Delta Y	Direction					\Delta X or \Delta Y	Vector length	0 0	0 step	0 1	1 step	1 0	2 steps	1 1	3 steps
b7	b6	b5		b4	b3	b2	b1	b0																																		
1	\Delta X	\Delta Y		Direction																																						
\Delta X or \Delta Y	Vector length																																									
0 0	0 step																																									
0 1	1 step																																									
1 0	2 steps																																									
1 1	3 steps																																									
0 0 0 1	1	Clear bit 1 of CTRL 1 : Eraser selection	!	1	A	Q	a	q																																		
0 0 1 0	2	Set bit 0 of CTRL 1 : Pen/Eraser down selection	"	2	B	R	b	r																																		
0 0 1 1	3	Clear bit 0 of CTRL 1 : Pen/Eraser up selection	#	3	C	S	c	s																																		
0 1 0 0	4	Clear screen	\$	4	D	T	d	t																																		
0 1 0 1	5	X and Y registers reset to 0	%	5	E	U	e	u																																		
0 1 1 0	6	X and Y reset to 0 and clear screen	&	6	F	V	f	v																																		
0 1 1 1	7	Clear screen, set CSIZE to code "minsize" All other registers reset to 0 (except XLP, YLP)	7	G	W	g	w																																			
1 0 0 0	8	Light-pen Initialization (WHITE forced low)	(	8	H	X	h	x																																		
1 0 0 1	9	Light-pen initialization	)	9	I	Y	i	y																																		
1 0 1 0	A	5 x 8 block drawing (size according to CSIZE)	*	:	J	Z	j	z																																		
1 0 1 1	B	4 x 4 block drawing (size according to CSIZE)	+	:	K	[	k	{																																		
1 1 0 0	C	Screen scanning : Pen or Eraser as defined by CTRL1	,	<	L	\	l	!																																		
1 1 0 1	D	X register reset to 0	-	=	M	]	m	}																																		
1 1 1 0	E	Y register reset to 0	.	>	N	\	n	—																																		
1 1 1 1	F	Direct image memory access request for the next free cycle.	/	?	O	←	o	☒																																		

## OTHER REGISTERS

### STATUS REGISTER (Read only)

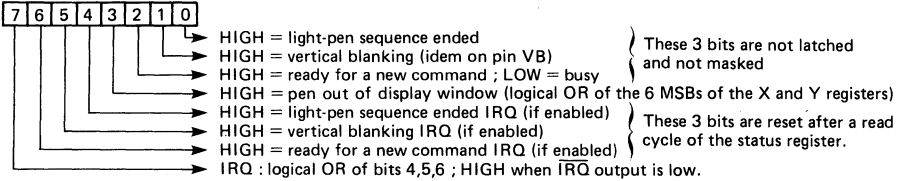
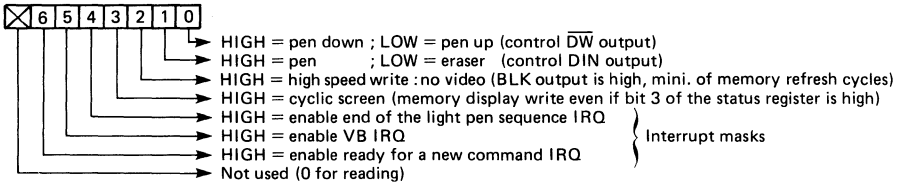
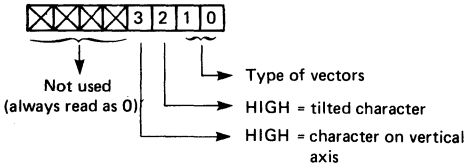


TABLE 3

### CONTROL REGISTER 1 (Read/Write)



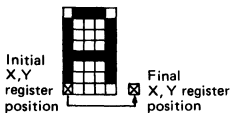
### CONTROL REGISTER 2 (Read/Write)



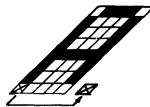
b1	b0	Type of vectors
0	0	continuous
0	1	dotted
1	0	dashed
1	1	dotted-dashed

2 dots on, 2 dots off  
 4 dots on, 4 dots off  
 10 dots on, 2 dots off,  
 2 dots on, 2 dots off.

### Types of character orientations



$b_3 = 0, b_2 = 0$



$b_3 = 0, b_2 = 1$



$b_3 = 1, b_2 = 0$



$b_3 = 1, b_2 = 1$

TABLE 4

Control registers CTRL1 and CTRL2 define the writing mode selected ("pen" and "erase" modes are used in a way analogous to their use on a plotter), interrupt modes and line types (see Table 4).

### ORIGIN

The origin subsystem comprises two registers X and Y defining the origin point relative to which calculations by the graphic display processor are defined. These two registers are modified by the processor when executing certain commands. For example, at the end of plotting a vector, registers X and Y point to the final coordinates of the vector.

### VECTOR GENERATOR

A standard vector is plotted in several steps :

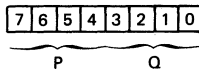
- Position origin : write registers X and Y,
- Position projections along X and Y axes : write registers  $\Delta X$  and  $\Delta Y$ ,
- Write command into corresponding register.

The GDP has a complete set of commands for plotting vectors (see Table 2) allowing the reduction of this sequence (non-standard vectors). These cover, for example, the plotting of vectors less than 4 dots long and vectors parallel to one axis.

### CHARACTER GENERATOR

Writing the ASCII code for a character into the command register automatically plots that character on the screen. The character size is programmed in register CSIZE with a scaling factor up to 16 in the X or Y direction. The smallest scale corresponds to code 0001 and the largest to code 0000 (see Table 5).

#### C-SIZE REGISTER (Read/Write)



P : Scaling factor on X axis  
Q : Scaling factor on Y axis

P and Q may take any value between 1 and 16. This value is given by the leftmost or rightmost 4 bits for P and Q respectively. Binary value (0) means 16.

TABLE 5

## LIGHT PEN CONTROL

Two 8-bit registers specify the address and status of the light pen following a light pen initialization command (see Table 6).

### XLP and YLP REGISTERS

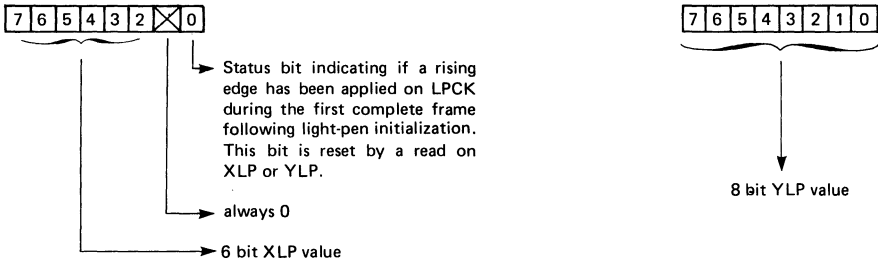


TABLE 6

This outline description of the internal registers of the EF9365/EF9366 shows the possibilities offered by the circuit. It also shows that in many applications programming this device reverts to writing an appropriate code into each of the appropriate registers.

## THE MPL LANGUAGE

MPL is a language derived from PL/1 and developed specifically for the EF6800/EF6809 microprocessor. It is well adapted to programming this type of device and its associated peripherals. The compiler generates assembler source language for easy debugging, attachment to other modules written in assembler language for easy debugging, attachment to other modules written in assembler language, for example, and storage in ROM. As compared with assembler language, MPL offers greater flexibility in terms of variable manipulation, arithmetic calculations and structured programming, without losing the visibility of the generated code and the microprocessor.

Figure 2 shows the sequencing of operations required in using the MPL compiler and the following outline specifications give an idea of the possibilities of this language.

Variable type :	binary on one or two bytes, signed or unsigned, bit decimal table in up to 3 dimensions.
Arithmetic operator :	addition, subtraction multiplication, division.
Logic operator :	shift AND, OR
Sequencing instructions :	IF ... THEN ... ELSE DO ... WHILE DO ... TO ... ("FOR" loop) Procedure call with entry and return of arguments

## EXAMPLES OF MPL USE

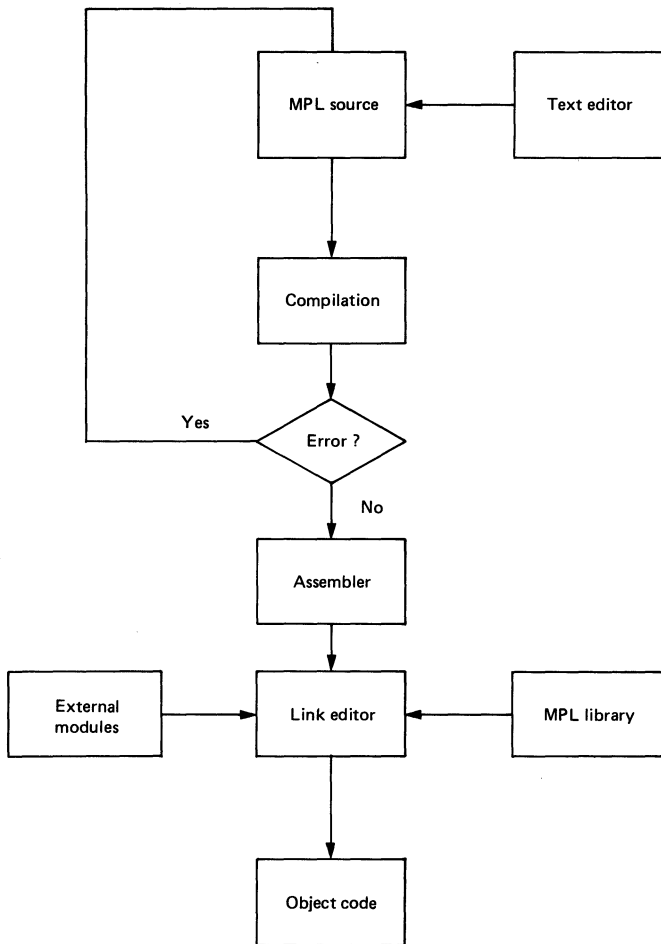


FIGURE 2

The following program examples relate to a hardware configuration based on EFCIS Eurocard family boards connected to the G64 bus :

- CPU board                   EFS-MPU1 (EF6800) or  
                                  EFS-MPU2 (EF6809).
- Graphics board             EFS-VIG1 (EF9366 + one memory plane)  
                                  EFS-VIE1 (8-colour extension).
- Memory board               EFS-32U2 (RAM/REPROM).

This configuration is connected to a colour video monitor to display a 512 x 256 dot image in eight colours (non-interlaced scanning).



The registers to be programmed are those of the EF9366 and an additional colour register defined as shown in the table below (for simplicity, only those bits of the register relevant to this application are defined) :

Bit								Colour
b7	b6	b5	b4	b3	b2	b1	b0	
Not used in this application					0	0	0	White
					0	0	1	Cyan
					0	1	0	Magenta
					0	1	1	Blue
					1	0	0	Yellow
					1	0	1	Green
					1	1	0	Red
					1	1	1	Black

Colour register

#### REGISTER AND VARIABLE DECLARATION

The internal registers of the EF9366 are declared using a structure which permits various types of variables to be mixed (the binary and bit types, for example). Not all bits of the control registers are declared. Only those used individually in the programs are explicitly declared :

- READY            Bit 2 of the STATUS register, indicating whether the circuit is ready to accept a further command.
- PLGOM           Bit 1 of register CTRL1, selecting the pen (PENER = 1) or eraser (PENER = 0).
- ONOFF           Bit 0 of register CTRL1, indicating pen-eraser down (ONOFF = 1) or up (ONOFF = 0).

Further declarations correspond to variables representing constants used in the colour register and registers CSIZE and CTRL2.

The declaration instructions shown in Figure 3 use 3 types of variable :

- BIN            Unsigned 1-byte variable
- BIN (2)        Unsigned 2-byte variable
- BIT (n)        n-bit variable

The n-bit variable type is used, when declared in a structure, to define an n-bit field in a specified byte.

Pseudo-instructions DEF and CONST specify a physical variable address and a symbol value, respectively (see Figure 3).

```

/* *****/
/*
/*          EF 9366 DECLARATIONS
/*
/* *****/

DECLARE 1 GDP DEF $F820,          ! F820=GDP address
      2 STATUS,                  ! status register
      3 NUS1 BIT(5),             ! not used
      3 READY BIT(1),           ! GDP ready
      3 NUS2 BIT(2),            ! not used
      2 COMAND BIN DEF STATUS,   ! command register
      ! same address as STATUS
      2 CTRL1,                   ! CTRL1 register
      3 NUS3 BIT(6),             ! not used
      3 PENER BIT(1),           ! pen,eraser selection
      3 ONOFF BIT(1),           ! up,down
      2 CTRL2 BIN,              ! CTRL2 register
      2 CSIZE BIN,              ! CSIZE register
      2 NUS4,                    ! byte not used
      2 DELTAX BIN,             ! DELTAX register
      2 NUS5,                    ! byte not used
      2 DELTAY BIN,            ! DELTAY register
      2 X366 BIN(2),            ! X register
      2 Y366 BIN(2),            ! Y register

```

```

/* *****/
/*
/* EXTERNAL REGISTERS AND
/* GLOBAL VARIABLES
/*
/* *****/

```

```

DECLARE COLOR DEF $F831          ! color register

```

```

/* GLOBAL VARIABLES
/*

```

```

/* variables for CSIZE register
/*

```

```

DCL   ZERO      CONST(0),
      ONE       CONST(1),
      TWO       CONST(2),
      THREE     CONST(3),
      FOUR      CONST(4),
      FIVE      CONST(5),
      SIX       CONST(6),
      SEVEN     CONST(7),
      EIGHT     CONST(8),
      NINE      CONST(9),
      TEN       CONST($A),
      ELEVEN    CONST($B),
      TWELVE    CONST($C),
      THIRT     CONST($D),
      FOURT     CONST($E),
      FIFT      CONST($F),
      SIXT      CONST($0)

```

```

/* variables for color register */
DCL    WHITE    CONST(0),
        CYAN    CONST(1),
        MAGEN   CONST(2),
        BLUE    CONST(3),
        YELLOW  CONST(4),
        GREEN   CONST(5),
        RED     CONST(6),
        BLACK   CONST(7)

/* variables for CTRL2 register */
DCL    VERTI    CONST(1),
        HORIZ   CONST(0),
        SLANT   CONST(1),
        NORMA   CONST(0),
        CONTI   CONST(0),
        DOTTED  CONST(1),
        DASHED  CONST(2),
        MIXED   CONST(3)

```

FIGURE 3

#### WRITING COMMON PROCEDURES

Like many other languages, MPL authorises the writing of procedures (subroutines) providing for the entry of parameters in two forms. The most general form supports parameters of any non-subscript type. The non-limiting list of parameters is then delimited by parentheses :

```
CALL { procedure name } (p1, p2, ..., pn)
```

The second form is limited to three parameters transmitted by accumulators A and B and register X of the microprocessor, but generates a shorter code. Parameters in this case are delimited by the symbols "< >" :

```
CALL { procedure name } < p1, p2, p3 >
```

#### Setting of register CSIZE

Procedure SETSIZ initialises the X and Y scaling factors of register CSIZE. This procedure may be called using the instruction :

```
CALL SETSIZ < TWO, TWO, >
```

which writes register CSIZE with the code 00100010.

#### Setting of register CTRL2

This register defines the character orientation and line type. The call may be effected as follows, for example :

```
CALL SETCR2 (HORIZ, NORMA, CONTI)
```

which specifies non-italic characters along the horizontal axis and continuous lines. The corresponding code is xxxx0000.

```

/* ***** */
/*
/* PROCEDURE SETSIZ:
/* SET CSIZE REGISTER
/* CALLING: CALL SETSIZ<P1,P2,>
/* P1: X SCALE
/* P2: Y SCALE
/*
/* ***** */

```

SETSIZ:

PROC<ECHX,ECHY,>

DECLARE ECHX BIN, ! for parameters  
ECHY BIN

ECHX=ECHX SHIFT 4 ! ECHX shifted in the 4 MSB  
CSIZE=ECHX IOR ECHY ! write in CSIZE

RETURN  
END SETSIZ

```

/* ***** */
/*
/* PROCEDURE SETCR2
/* SET CNTRL2
/* CALLING:CALL SETCR2(P1,P2,P3)
/* P1: CHARACTERS
/* ORIENTATION
/* P2: CHARACTERS TYPE
/* P3: TYPE OF VECTORS
/*
/* ***** */

```

SETCR2:

PROC(PR1,PR2,PR3)

DECLARE PR1 BIN, ! for parameters  
PR2 BIN,  
PR3 BIN

PR1=PR1 SHIFT 3 ! justifies parameters in byte  
PR2=PR2 SHIFT 2

CTRL2=PR1 IOR PR2 IOR PR3

RETURN  
END SETCR2

FIGURE 4

### APPLICATION EXAMPLE N° 1

This example shows the procedures described above and causes a text to be displayed on the screen in blinking mode (see Figure 6).

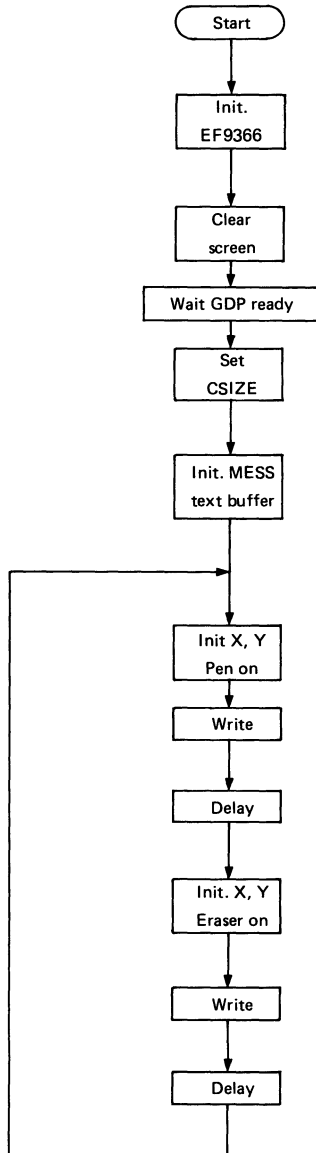


FIGURE 5

```

/*****
/*
/* WRITE AND FLICKER OF A TEXT
/*
/*****

EX1:
  PROCEDURE OPTIONS(MAIN)

  DECLARE MESS CHAR(30), ! text buffer
    WAIT BIN(2) ! for delay

/* GDP initialization */

  COMAND=$6 ! clear screen

  DO WHILE READY=0
    END ! GDP ready ?

  CTRL1=$0 ! inhibition of interruptions
  ONOFF=1 ! pen down

  COLOR=RED
  CALL SETSIZ<TWO,TWO,>

  MESS='THOMSON-EFCIS' ! initialization of text

/* writing loop */

LOOP:
  PENER=1 ! pen selection
  X366=100 ! initialize X and Y
  Y366=20
  CALL WRITE<13,> ! write 13 characters

  WAIT=20000
  CALL DELAY(WAIT) ! delay

  PENER=0 ! selection of eraser
  X366=100
  Y366=20
  CALL WRITE<13,>

  WAIT=10000
  CALL DELAY(WAIT)

  GOTO LOOP
END EX1

```

```

/*****
/*
/* PROCEDURE DELAY
/* CALLING: CALL DELAY(value)
/* value<65000
/*
/*
/*****

```

```

DELAY:
    PROC(TIME)

    DECLARE TIME BIN(2)

    DO WHILE TIME NE 0
        TIME=TIME-1
    END
RETURN

END DELAY

```

```

/*****
/*
/* PROCEDURE WRITE
/* CALLING: CALL WRITE<value,,>
/* value=NB. OF CHARACTERS
/* value< 30
/* EXIT: GDP READY
/*
/*
/*****

```

```

WRITE:
    PROC<NBCAR,,>

```

```

/*    local variables    */

```

```

DECLARE TBMESS(30) BIN DEF MESS /* for ASCII conversion */
/* same address as MESS */
DECLARE NBCAR BIN
DECLARE N BIN

    DO N=1 TO NBCAR
        COMAND=TBMESS(N) /* write the caractere */
        DO WHILE READY=0 /* wait GDP ready */
            END
        END

RETURN

END WRITE

```

FIGURE 6

## WRITING VECTORS

The vector generator of the EF9366 simplifies the plotting of vectors, the programmer needing only to position the origin and projections of the vector. The appropriate command then runs the algorithm of the graphic display processor which causes the vector to be plotted.

The instruction set comprises 144 codes. This optimises the plotting function but may sometimes result in the programmer having to search for the codes corresponding to the required vector. With the standard the end coordinates of the vector, this procedure computes the projections along the X and Y axes and the code of the command and then sets the X and Y registers (see Figures 7 and 8). Plotting the vector is thus reduced to calling the procedure :

CALL WRVECT (X origin, Y origin, X end, Y end).

For example, the following instruction :

CALL WRVECT (0, 0, 50, 70) writes the registers with the following value :

X : \$0000  
Y : \$0000  
DELTA X : \$32  
DELTA Y : \$46  
Command : \$11



WRVECT

Write  
X and Y

Calculate  
projections  
DTX, DTY

Calculate bits  
b3, b2, b1  
of command  
(Direct)

Calculate  
bit 0 of  
command  
(SPECL)

NO YES  
Small  
vector

b7,b6,b5,b4  
= § 1 (GTVECT)

b7 = 1  
(SMVECT)

b6,b5 ← DTX  
b4,b3 ← DTY

Command =  
GTVECT +  
SMVECT +  
DIRECT +  
SPECL

End

FIGURE 7

```

/* ***** */
/*
/*      PROCEDURE WRVECT:
/*      INPUT:  ORIGIN;X0,Y0 (START)
/*              ;X1,Y1 (END)
/*      EXIT:  SET DELTAX,DELTAY,COMAND
/*
/*      THE VECTOR MUST HAVE DELTAX AND
/*      DELTAY <= 255
/*
/* ***** */
WRVECT:
  PROC (X0,Y0,X1,Y1)

    DCL X0 BIN(2) ,Y0 BIN (2), /* start of vector      */
        X1 BIN(2) ,Y1 BIN(2) /* end of vector        */
    DCL SPECL BIN, /* special direction    */
        DIRECT BIN, /* direction            */
        GTVECT BIN, /* great vector         */
        SMVECT BIN /* small vector         */

/* set X and Y of GDP */

X366=X0
Y366=Y0

/* declarations for computing with two bytes */

    DCL DTX SIGNED BIN(2),
        DTY SIGNED BIN(2)

DTX=X1-X0
DTY=Y1-Y0

/* direction scanning */

IF ( DTX GT 0 AND DTY GE 0 ) THEN DIRECT=0
IF ( DTX LE 0 AND DTY GT 0 ) THEN DIRECT=2
IF ( DTX LT 0 AND DTY LE 0 ) THEN DIRECT=6
IF ( DTX GE 0 AND DTY LT 0 ) THEN DIRECT=4

/* special vectors scaanning */

IF (DTX =0 OR DTY =0)
  THEN SPECL = 0
  ELSE SPECL = 1

/* conversion DTX,DTY in positive */

IF DTX LT 0 THEN DELTAX=-DTX ELSE DELTAX=DTX
IF DTY LT 0 THEN DELTAY=-DTY ELSE DELTAY=DTY

```

```
/* small or great vectors scanning */  
IF( DELTAX > 3 OR DELTAY > 3 )  
  THEN  
    DO  
      GTVECT = $10  
      SMVECT = 0  
    END  
  ELSE  
    DO  
      GTVECT=0  
      SMVECT=$80  
      DELTAX=DELTAX SHIFT 5  
      DELTAY=DELTAY SHIFT 3  
      SMVECT=SMVECT IOR DELTAX IOR DELTAY  
    END  
  COMAND=SMVECT IOR GTVECT IOR DIRECT IOR SPECL  
  RETURN  
  END WRVECT
```

FIGURE 8

## APPLICATION EXAMPLE N° 2

The program shown in Figure 10 moves a solid rectangle along the horizontal axis. The flowchart is the following (Figure 9).

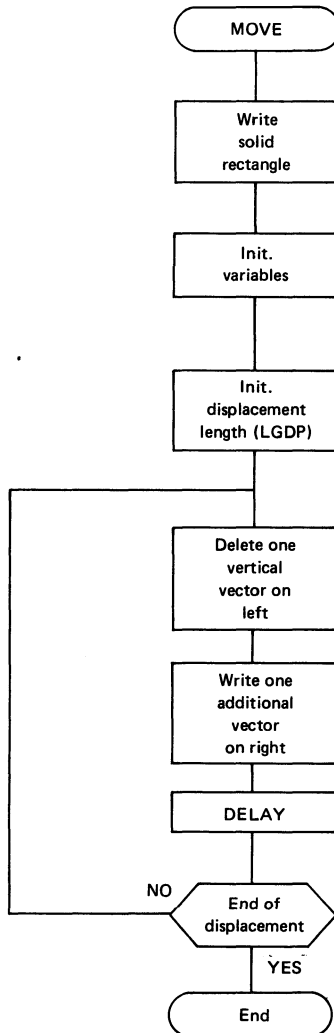


FIGURE 9

```
/*  
/*  
/* MOVING A FULL RECTANGLE  
/*  
/*  
/*  
/*  
/*  
*/
```

```
EX2:  
  PROCEDURE OPTIONS(MAIN)
```

```
  DCL DPX SIGNED BIN(2), ! start in X  
    DPY SIGNED BIN(2), ! start in Y  
    DAX SIGNED BIN(2), ! end in X  
    DAY SIGNED BIN(2), ! end in Y  
    WIDT SIGNED BIN(2), ! width of rectangle  
    EIGH SIGNED BIN(2) ! height of rectangle
```

```
  DCL J BIN(2) ! subscript for loop
```

```
  /* initialization of GDP */
```

```
  COMAND=$6  
    DO WHILE READY=0  
      END
```

```
  CTRL1=$0  
  PENER=1  
  ONOFF=1
```

```
  /* co-ordinates initialization of rectangle */
```

```
  DPX=30  
  DPY=30  
  WIDT=100  
  EIGH=60
```

```
  DAX=DPX+WIDT  
  DAY=DPY
```

```
  /* write first rectangle */
```

```
  DO J=1 TO EIGH  
    CALL WRVECT(DPX,DPY,DAX,DAY) ! call in library  
    DO WHILE READY=0  
      END  
    DPY=DPY+1 ! incrementation of origins  
    DAY=DPY
```

```
  END
```

```
  /* moving of rectangle */
```

```
    DCL LGDP BIN ! width of moving
```

```

/* initialization of parameters */

LGDP=40

DPX=30
DPY=30
WIDT=100
EIGH=60

DAY=DPY+EIGH-1
DAX=DPX

/* loop for moving */

DO WHILE LGDP NE 0

/* clear l vector on left */
PENER=0
CALL WRVECT(DPX,DPY,DAX,DAY)
DO WHILE READY=0
END

/* write l vector on right */
PENER=1
DPX=DPX+WIDT+1
DAX=DPX
CALL WRVECT(DPX,DPY,DAX,DAY)
DO WHILE READY=0
END
WAIT=300
CALL DELAY(WAIT) ! call in library
DPX=DPX-WIDT
DAX=DPX

LGDP=LGDP-1 ! decrementation of subscript

END

END EX2

```

FIGURE 10

## USE OF MPL COMPILER

The compiler is supplied on floppy disk and comprises the operating system (MDOS or EFDOS), the compiler (file MPL.CM) and the library (file MPLSLIB.RO).

The compilation commands for a source file are as follows (under MDOS or EFDOS) :

```
MPL TOTO ; LSO = TOTO
```

This compiles file TOTO with listing of the source at the printer and creation of a compiled file TOTO.AI (assembler source containing as commentary the corresponding MPL source instructions).

The procedure starting with file TOTO.AI is the same as that used for an assembler program :

```
ASBL TOTO.AI
```

This creates object file TOTO.RO.

Link editing may be carried out as indicated below, calling library MPLSLIB.RO and creating runnable object file TOTO.LO :

```
LINK  
?IF = F1  
? BASE  
? LOAD = TOTO  
? LIB = MPLSLIB  
? OBJA = TOTO  
? EXIT
```

## USE OF USER LIBRARY

Preceding examples have shown how procedures are used in various applications. It is beneficial to group procedures in a library which is called at the link editing stage. This means that the programmer need not insert these modules into all his applications, and reduces compilation time.

Procedures SETSIZ, SETCR2, TEMPO, ECRIT and WRVECT combined to constitute a single source file GDPLIB.SA once compiled and assembled give the library file GDPLIB.RO. At link editing time the operator then calls the two files GDPLIB.RO and MPLSLIB.RO in order to insert the procedures used into the object file.

**NOTES**

---

Printed in France