Elektronite

presents

Intellivision White Papers

Paper #3

Programming the Intellivision II

Hello world

Version 1.1

[Authors]

Carlos Madruga
Valter Prette

Contents

Introduction

Learning any programming language will be always easier when done through examples.

From that perspective, let us examine a little program written by Merrick J. Stemen, which was submitted to the first SDK1600 programming “contest” proposed by the “Intellivision Library” web site in 2000.

Legal disclaimer:

You’re reading an Elektronite IntyLab White Paper.
The document is freely distributable but not modifiable. If you intend to modify or add contents to this document, please contact vprette@hotmail.com to let me integrate your contribution in a new release version for the interest of all the collectors worldwide.

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

Hello world

The output of the program itself is pretty simple: it will generate a standard presentation screen, and then return a message in case any buttons are pressed. You can learn many things from it.

We will present the source code below, along with some comments.

```
;;-----------------------------------------------------------------------

;;MICKEY.ASM

;;-----------------------------------------------------------------------

;; by Merrick J. Stemen

;; 18 JANUARY 2000

;; This program does use EXEC calls.

;; Sorry it is not really a game, but it is a start!

;;

;; Compiled with the AS1600 compiler.

;;

;;-----------------------------------------------------------------------

;; Portions of the ROMHDR adapted from Joe Zbiciak's PONG
```

Observe how the AS1600 compiler understands anything written after “;” or “;;” as

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

comments.
What did the author of the program mean by “EXEC calls”?
The EXEC (as in “executive”) is a 4Kb program located at Intellivision's ROM. The EXEC has routines relative to graphics manipulation, sounds, etc.
In case you want to use such routines on our own programs, you will need to have a file containing a copy of EXEC's code. Such file can be found for instance inside the “Intellivision Lives” CD, and it is called “Exec.bin”. Your emulator needs to be configured properly in order to be able to utilize that file, but that will not be covered by this tutorial.

For your information, Joe Zbiciak has made available his own version of the EXEC ("miniexec.bin"), which is free, as opposed to the EXEC which is copyrighted. You may choose to use his routines instead, but that is really up to you. Just make sure you read the licensing details.

```
        ROMW    10

        ORG     $5000
```

The ROMW label (as in “rom width”) indicates that we will be working with a 10-bit “cartridge”! That may sound odd, but it is in fact ok when it comes to the Intellivision, which can also use RAMS/ROMs that are 8, 12, 14 or 16-bits wide.

By the way, nomenclature-wise, 10 bits are called a “decle”.

The ORG instruction indicates the starting point of our program. Programs may occupy up to 8Kb inside the RAM, located between $5000 e $6FFF. The "$" character indicates the number immediately after it is in hex.
However, if we wanted to write a program larger than 8Kb, we could count on an extra 4Kb block located between $D000 and $DFFF, and yet another 4Kb block between $F000 and $FFFF.

```
;;-------------------------------------------------------------------------

ROMHDR:
```

Programs need to include a standard header, in this case indicated by the label ROMHDR. This header is used by the EXEC for initialization purposes.

Note: this header is also referred to as the "Universal Data Block" (UDB)

```
        WORD    $0001           ; Movable Object Data Pointer
```

The STIC (Standard Television Interface Chip) is the chip that takes care of graphics for the Intellivision. One of the things it does is to manage up to 8 “moving objects” on screen.
An “object” is some shape defined by the user that can be easily moved across the screen, usually on top of a background picture. If you are familiar with the term “sprite”, then that is exactly what we are talking about.

The code above indicates the location of the "sprite picture databases", meaning: where is the starting point of the decles that together represent the shape of a sprite. The EXEC copies this database to the GRAM during the initialization process.
Anyway, we do not use sprites at this program. That database will be empty.

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

```
        WORD    $0000           ; RTAB
```

This address points to a routine that is executed periodically. The frequency of execution is every 0.5 seconds. Not relevant to this program.

```
        WORD    START           ; START of program
```

Points to the real program starting point, labeled START.
That routine will only be called after executing the code that comes after the specification of the cartridge's title. Don't worry, you will understand it later.

```
        WORD    $0000       ; Background graphics pointer
```

Also sometimes labeled "BKGDATA". Just like the first line of the header, this address points to the place where shapes of background images will be defined. We do not use those at this program.

```
        WORD    ROMHDR          ; CARDTAB
```

Card Table. This is a list that specifies pictures from BKGDATA to be transferred to the RAM. Not relevant to this program.

```
        WORD    TITLE           ; Pointer to ASCIIZ Title String
```

Points to the cartridge title, or that name that comes after "Mattel Electronics presents...".

```
        BYTE    $80             ; Bit 7 SET -- Run code after title
```

Bits 0 through 4 from this address control whether there will be “click” sounds when disk/keys/action keys are pressed.
If bit 7 is set, the code immediately after the specification of the cartridge title will be executed.
The number 80 in hex is equivalent to 10000000 in binary, meaning: "clicks off" and will run the code immediately after the title.

```
        BYTE    $02             ; H/V offset
```

Whatever is typed at this line goes straight to address $32 at the STIC.
If bit 0 is set (meaning: equal to 1), the first column of cards is blocked, which creates a thicker border to the left side of the screen. Bit 1 does the same thing in relation to rows.

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

In the case of this program, the value being used is 02, which is equivalent to 00000010 in binary. That means that we are doing a 1-row screen offset.

But you might be wondering at this point what are "cards"...

The STIC puts the screen together by using 240 cards (20 width x 12 height), which are groups of 8x8 pixels.
That is equivalent to a resolution of 160x96 pixels but because in reality the STIC won't show you the last column located to the right, the real resolution is 159x96. Each pixel can have one of 16 colors (8 primaries, 8 pastels).

```
BYTE    $00             ; Color Stack mode (0=CS; 1=F/B)
```

This piece is important, because it indicates what color scheme will be used by the STIC. Each card may have 2 colors.

The first scheme is the so-called "Foreground/Background" (F/B). In this scheme, each card can have its 2 colors determined individually, but the catch is that one of those colors HAS TO BE a primary one. The other color may be a primary or pastel one.

The second scheme is called "Color Stack Mode" (CS). In this scheme, we define a circular stack of 4 colors. Those colors may be any primary or pastel ones, but now the catch is different: one of the colors of each card HAS TO BE either one color pulled from the stack, or the next one in sequence. That way it is possible to have 2 pastels on the same card, which is not possible through the F/B scheme.
There is a third scheme called “Colored Squares", but it is not used very often. For more information please refer to the document "De re
Intellivision, part 2".
The program being studied here utilizes the CS mode..

```
BYTE    1, 1, 1, 1      ; Color Stack data (All blue)
```

This is the color stack to which I referred before. Here we list the 4 colors that will go inside the stack. If we were using the F/B mode, this line could countain anything because it would actually be ignored.

The possible color codes are:

Primaries:
0=Black, 1=Blue, 2=Red, 3=Tan, 4=Dark Green, 5=Green, 6=Yellow, 7=White

Pastels:
8=Gray, 9=Cyan, 10=Orange, 11=Brown, 12=Pink, 13=Light Blue, 14=Yellow-Green, 15=Purple

In this program the author decided to put only Blue inside the stack

```
BYTE    $02             ; Red Border
```

Color of the screen border. In this case, red.

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

Ok, this is the end of the ROMHDR.

Certain programs may present an additional section of the header, but that would have to do with SPRITES - which we do not use in this case. So that section was not necessary.

```
;;-------------------------------------------------------------------------

TITLE:          BYTE    100, "MICKEY'S GAME",0
```

This is the cartridge title, indicated by the TITLE label.
The first value is the year, which in this case has been set to 100. In reality, one could say the Intellivision does suffer from the Y2K bug, since years at the presentation screen only go up to 1999. :-) Whatever value is specified here ends up being put side by side to a “19...” at the presentation screen.

One solution to this issue is to overwrite the year message at the presentation screen.

Finally, after the year comes the actual title ("MICKEY'S GAME"), and at the end there is always the number 0.

```
;;-------------------------------------------------------------------------

;; Print "2000" where the copyright date goes

;; (I realize that I only need two zeros)

POST_TITLE:         MVII    #$2D0, R4       ; Where the copyright date goes

                    MVII    #$97, R1        ; Character "2" - white

                    MVO@    R1, R4          ; Put the "2" on the screen
```

Here the author starts overwriting the presentation screen in order to avoid the year issue. He also is overwriting the name “Mattel”. You would not just hand out the rights to your project to them either, would you ? :-)

All this constitutes part of the code that runs immediately after the title definition, which is being executed just like specified at the header.

The MVII instruction points a register to an address.

We start pointing R4 to $2D0 (location of the character to be written on screen) and R1 to $97.
Following that, we use the MVO@ instruction, which is equivalent to a POKE Rx, Ry in BASIC.

But why do we point R4 and R1 to those values ?

One step at a time. First about the value of R4:

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

Addresses linked with background cards on screen (“BACKTAB”) start at $200 and go up to $2F0, which corresponds to 240 cards in sequence. If you remember, there are 20 cards per row on screen. So if you do some calculations, you will see that the address $2D0 corresponds to the 10th line, 8th column card.
If you can not arrive at this result, find a scientific calculator that can do hex operations, subtract $2D0 from $200 and convert the result to decimal. You should get the result 208, which is 10x20+8... there you go !

Now about the value of R1:

Because we are using the CS color mode (remember the header !), each card is defined the following way:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
value	?	?	a	cF	t	n	n	n	n	n	n	n	n	f	f	f
$97 =	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1

where:

a = if set, advances to the next color of the stack

Ffff = foreground color. Can be primary or pastel.

t = if equal to 0, indicates that the card is pointing to the GROM. Otherwise, it will point to the GRAM.

nnnnnnnn = position of the card on the GROM (0-212) or GRAM (0-63)

The GROM counts with more characters than the GRAM does. With the GROM we can use lower case characters, punctuation and graphic characters. In this program, we can see that $97 is equivalent to card 18 at the GROM, which is simply the number “2”.

With that, we place character "2" (which color is white – see bits 0,1,2 and 12) at the position $2D0. Remember all this is being done with the objective of writing the year “2000” over the presentation screen.

One characteristic of instruction MVO@ is that when it is used in conjunction with a so-called "Auto-Incrementing Data Counter" (registers R4 and R5), such data counters are automatically incremented. That means that we will be already at the right position to write the characters that come next in sequence. Neat.

```
MVII   #$87, R1        ; Character "0" - white
MVO@    R1, R4          ; "0"
MVO@    R1, R4          ; "0"
```

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

```
MVO@    R1, R4          ; "0"
```

We then load R1 with the “0” character. No need to point R4 to the next position at $2D1, because the instruction MVO@ executed before already did that.
So all we had to do was to execute 3 times the instruction MVO@ R1,R4 to be able to finish printing "2000" at the presentation screen.

```
MVO@    R5, R6          ; Push R5
INCR    R4
JSR     R5, $187B
BYTE    "Mickey", 0     ; Put my name over last Mattel
MVI@    R6, R7          ; Return from Title Screen
```

The next step was to put the name “Mickey” above the word “Mattel”. For that, the author decided to experiment with a subroutine from the EXEC dedicated to printing whole words at once, rather than printing each individual character as before.

Before that, we have to save the content of R5 to register R6 with the instruction MVO@ R5,R6, because that value will be lost when the printing subroutine is called. Instruction JSR is in charge of calling the subroutine, but it will write its return address to R5.

But why is saving the content of R5 on R6 important at all ?

In reality, when the code that comes after the title is executed, everything is seen as a big subroutine, whose return address stays stored in R5. It is as if the EXEC had done a JSR R5, POST_TITLE.

So if we save this return address in R6, we can later move it back to the Program Counter (R7) and let the program continue as normal.

Back to the piece of code above, we also increment R4 through the instruction INCR, just so there is a space between “2000” and “Mickey” on screen.

A few more details about the printing subroutine:

1- we use a JSR instruction to call it. As explained before, that instruction will store its return address in R5.

2- the string to be printed must be located after the JSR instruction, followed by a 0.

3- register R4 must point to the starting location of the string on screen.

4- register R3 can be used to store the color of the characters. Because the author of this program never did that, it will use white.

5- register R7 (Program Counter) is auto-incremented after execution of the subroutine.

After calling the subroutine, we execute the instruction MVI@ R6,R7. Remember that now R6 has the value of R5 when POST_TITLE got called. So R7 will now have the

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

return address of POST_TITLE.

So what happens next? Can you tell?

Yes, you got it right, POST_TITLE is over and the main program will now be called by the EXEC.

(So you DID NOT get it right? No problem, now you know :-)

```
;;-------------------------------------------------------------------------

START:          MVO@    R5, R6          ; Push R5

                MVII    #$0200, R4       ; Start of ScreenMem

                MVII    #240, R0         ; To clear entire screen

                JSR     R5, $1738        ; CLEARMEM
```

Here is the start of the real program, which is what comes after the title screen.
The first thing done was to save R5 to R6. Same logic as in POST_TITLE.
Next, we call another subroutine from the EXEC, which is used to clear the screen.
This routine works the following way:

1- R4 must point to the initial address of the range that will be cleared.
2- R0 must have the number of cards to be cleared. In our case, the number 240 is being used because we want to clear the whole screen (remember: screen is 20x12). Note that 240 does not have “$” in front of it, which means the value is in DECIMAL.

```
                MVII   #$027C, R4  ; Where to print the message

                MVII   #$0004, R3        ; Color to print the message

                JSR     R5, $187B        ; Print Data that follows

                BYTE    "HELLO,WORLD!", 0

                MVII    #$02DE, R4       ; Where to print the message

                MVII    #$0000, R3       ; Color to print the message

                JSR     R5, $187B        ; Print the data that follows

                BYTE    "PRESS F10 OR ESC", 0

                MVI@    R6, R7           ; IS THIS THE END??

;;-------------------------------------------------------------------------
```

Finally, after clearing the screen, we will print two messages.
For that, we use that old subroutine from the EXEC located at $187B.

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

The last instruction of the program transfers the value contained at the memory address pointed by R6 to the Program Counter R7. That ends the program.

So what did you think about this first tutorial ?
Too easy? Too scary? Did not understand a thing?
Let us know your opinion by contacting Elektronite at intylab@elektronite.com

Here are some suggested exercises for you:

1- Change the message at the title screen from "Mattel Electronics presents" to "<your name> presents".

2- Change the background color from blue to black.

3- Eliminate the offset from the borders.

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

APPENDIX: EXERCISE SOURCE CODE

```
;;-------------------------------------------------------------------

;;MICKEY.ASM

;;-------------------------------------------------------------------

;; by Merrick J. Stemen

;; 18 JANUARY 2000

;; This program does use EXEC calls.

;; Sorry it is not really a game, but it is a start!

;;

;; Compiled with the AS1600 compiler.

;;

;;-------------------------------------------------------------------

;; Portions of the ROMHDR adapted from Joe Zbiciak's PONG

ROMW 10

ORG     $5000

;;-------------------------------------------------------------------

ROMHDR:

WORD  $0001             ; Movable Object Data Pointer
WORD  $0000             ; RTAB
WORD  START             ; START of program
WORD  $0000             ; Background graphics pointer
WORD  ROMHDR                    ; CARDTAB
WORD  TITLE             ; Pointer to ASCIIZ Title String

BYTE    $80             ; Bit 7 SET -- Run code after title
BYTE    $02             ; H/V offset
BYTE    $00             ; Color Stack mode (0=CS; 1=F/B)
BYTE    1, 1, 1, 1      ; Color Stack data (All blue)
BYTE    $02             ; Red Border

;;-------------------------------------------------------------------

TITLE:  BYTE    100, "MICKEY'S GAME",0

;;-------------------------------------------------------------------
;; Print "2000" where the copyright date goes
;; (I realize that I only need two zeros)
```

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.

```
POST_TITLE:   MVII    #$2D0, R4       ; Where the copyright date goes
MVII    #$97, R1                      ; Character "2" - white
MVO@ R1, R4                           ; Put the "2" on the screen
MVII    #$87, R1        ; Character "0" - white
MVO@ R1, R4             ; "0"
MVO@ R1, R4             ; "0"
MVO@ R1, R4             ; "0"
MVO@ R5, R6             ; Push R5
INCR    R4
JSR     R5, $187B
BYTE    "Mickey", 0     ; Put my name over last Mattel :-)
MVI@    R6, R7          ; Return from Title Screen
;;-------------------------------------------------------------------------
START:MVO@ R5, R6  ; Push R5
MVII    #$0200, R4      ; Start of ScreenMem
MVII    #240, R0        ; To clear entire screen
JSR     R5, $1738       ; CLEARMEM
MVII    #$027C, R4      ; Where to print the message
MVII    #$0004, R3      ; Color to print the message
JSR     R5, $187B       ; Print Data that follows
BYTE    "HELLO,WORLD!", 0
MVII    #$02DE, R4      ; Where to print the message
MVII    #$0000, R3      ; Color to print the message
JSR     R5, $187B       ; Print the data that follows
BYTE    "PRESS F10 OR ESC", 0
MVI@    R6, R7  ; IS THIS THE END??

;;-------------------------------------------------------------------------
```

This document is an Elektronite White Paper. The document is freely distributable but not modificable. If you intend to modify or add contents to this document, please contact Elektronite at www.intellivisionworld.com to let us integrate your contribution in a new release version for the interest of all the collectors worldwide.