Using the Keyboard and Graphics Adapter Simulator

November 2014

Contents

1 Introduction

This document is the only literature available describing the Keyboard and Display Adapter and must therefore fill several roles.

- For those who want to get started quickly printing text to the screen, the appendix contains useful functions for common tasks (such as writing a string to the screen) that you can copy and paste into your MIPS program.
- If you want a better feel for how basic string drawing works, skip to the heavily commented examples in section 3.2.
- Should you run into trouble, want to write more efficient code than is possible by using the provided functions, or desire a deeper understanding of the adapter and its commands, the rest of the document details all possible commands and states of the adapter and can be used as a reference.

The Keyboard and Graphics Adapter Simulator (KGAS) is a modified version of the Keyboard and Display Simulator bundled with MARS. Using the same four words of memory as the KDS, the display component of the KGAS trades the ability to display non-Unicode characters for the ability to write multicolored characters to arbitrary locations on the screen. The KGAS, moreover, supports several graphical "commands" to facilitate common operations like clearing the screen, with potential for future expansion.

The keyboard simulated by the KGAS is mostly unchanged, but now includes the ability to process non-printing characters like the arrow keys. Printing characters' ASCII values are sent, as before: for example, pressing Shift-A will send 'A' through the keyboard, not the keycodes for Shift or A. Furthermore, the text field does not retain all the characters the user has typed, ending the annoying side effect of the user's input building up in an illegible mass–but forcing the MIPS program to be responsible for showing users what they have typed.

2 The Keyboard

Use of the keyboard simulated by Keyboard and Graphics Adapter Simulator is identical to the Keyboard and Display Simulator, and is described by Pete Sanderson in the Help section for that tool:

> While the tool is connected to MIPS, each keystroke in the text area causes the corresponding ASCII code to be placed in the Receiver Data register (low-order byte of memory word 0xffff0004), and the Ready bit to be set to 1 in the Receiver Control register (low-order bit of 0xffff0000). The Ready bit is automatically reset to 0 when the MIPS program reads the Receiver Data using an 'lw' instruction.
>
> ...
>
> Upon setting the Receiver Controller's Ready bit to 1, its Interrupt-Enable bit (bit position 1) is tested. If 1, then an External Interrupt will be generated. Before executing the next MIPS instruction, the runtime simulator will detect the interrupt, place the interrupt code (0) into bits 2-6 of Coprocessor 0's Cause register ($13), set bit 8 to 1 to identify the source as keyboard, place the program counter value (address of the NEXT instruction to be executed) into its EPC register ($14), and check to see if an interrupt/trap handler is present (looks for instruction code at address 0x80000180). If so, the program counter is set to that address. If not, program execution is terminated with a message to the Run I/O tab. The Interrupt-Enable bit is 0 by default and has to be set by the MIPS program if interrupt-driven input is desired. Interrupt-driven input permits the program to perform useful tasks instead of idling in a loop polling the Receiver Ready bit! Very event-oriented. The Ready bit is supposed to be read-only but in MARS it is not.

As described in the introduction, the keyboard is different in only two relatively minor ways. First, pressing a non-printing character (like the arrow keys or control, but not Enter or Backspace) will sent a one-byte keycode to the second-lowest byte of the Receiver Data Register. (The second-lowest byte was chosen so that there would be no ambiguity between UTF-8 characters and key codes.) *There is one exception:* since Shift is used to send capital letters and other valid UTF-8 characters, its keycode is not sent. Also, the user's input is no longer displayed in the text area except for the last character entered.

3 The Graphics Adapter

3.1 Registers

The graphics adapter is controlled and monitored by MARS using two 32-bit registers: the Status and the Data Register. Both are partitioned into four 8-bit subregisters.

3.1.1 The Status Register

The Status Register, address 0xFFFF0008, is used by the display adapter to convey information back to the running MIPS program. Usually, this information is about the current state of the adapter. The four subregisters of the Status Register are:

Enquiry 1	Enquiry 2	Error Level	Flags

Fig. 1. The bytes of the Status Register, in big-endian order.

The first three registers are currently unused. The Enquiry registers are intended to hold the adapter's response to an enquiry made of it by the user. The Error Level register is intended to be used by more complex drawing commands for which the flags devoted to error handling would be either insufficient or ambiguous.

The Flags register is a bit set containing boolean values related to the current status of the adapter which are most likely to be immediately relevant, including whether the adapter is ready for more input. The flags currently supported are:

INVALID_ARGS	INVALID_CMD	INVISIBLE_CHAR	OFFSCREEN	3	2	INTERRUPTS	READY
—Error codes: commands—		—Error codes: text—		—Status—			

Fig. 2. The bits of the Flags Register, organized into categories by type.

The high four bits represent error conditions and are set whenever the corresponding error occurs: INVALID_ARGS is set when a command's arguments are out of bounds, INVALID_CMD is set when an unidentified command is sent, INVISIBLE_CHAR is set when a character's background and foreground color is the same, and OFFSCREEN is set when a character is drawn at invalid coordinates.

The low four bits represent normal conditions: only the lower two are used. If INTERRUPTS is set, an external interrupt will be generated after the display *becomes* ready. External interrupts will not be generated after a character is drawn or after a no-delay command is executed, because the adapter executes these "infinitely" quickly relative to the MIPS program. If the adapter is READY for a command or character, that bit is set.

3.1.2 The Data Register

The Data Register is used to send either characters or graphical commands to the adapter. From least to greatest address, the subregisters are

1. Attribute Register [Argument Register 2] (0xFFFF000C)
2. Vertical Position Register [Argument Register 1] (0xFFFF000D)
3. Horizontal Position Register [Command Register] (0xFFFF000E)

4. Character Register (0xFFFF000F)

The two names for each register correspond to the two ordinary functions of the adapter: printing a character, or processing a command. The registers' order is, perhaps, the opposite of what would be expected. This was done so that, in the ordinary little-endian configuration of MARS, a character or command and its arguments could be written as a hexadecimal word from left to right. A command is "executed" by the graphics adapter when the device is ready and when the Character Register is written to.

3.2 Displaying text

Whenever a value between 0x01 and 0xFE is written to the Character Register, the corresponding character is written to the screen. Additional data, however, is necessary; each character must have a position on screen and a set of attributes defining how it is to be drawn. This information is provided in the other three registers, as their names suggest:

- The Horizontal and Vertical Position Registers hold the X and Y coordinates of the character to be drawn. These coordinates are not in pixels, but in character units: a character at (0,0) is at the upper-left hand corner, and a character at (1,0) is immediately adjacent to it.
- The Attribute Register is used for color. Each character has a foreground and background color, each of which is represented by four bits:

Intensity	Red	Green	Blue	Intensity	Red	Green	Blue
—Foreground color—				—Background color—			

Fig. 3. The bits of the Attribute Register

This usage is identical to color support on the IBM CGA in text mode. A more immediately useful table of colors is shown below:

0x0	0x1	0x2	0x3
0x4	0x5	0x6	0x7
0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF

Fig. 4. The sixteen possible colors generated by the adapter

- The Character Register, as previously mentioned, contains characters between 0x01 and 0xFE inclusive. These values, by default, represent characters in the IBM PC's code page 437: this character set was chosen to make vivid text user interfaces possible, with support for box drawing and shading.

3.2.1 Examples

Note: before running any example, ensure that the KGAS is running and connected to MARS.

```
li  $t0 ,  0x0F          #color: black with white background
sb  $t0 ,  0xFFFF000C    #store in Attribute Register
li  $t0 ,  0x00          #y position: 0
```

```
sb $t0, 0xFFFF000D      #store in Vertical Position Register
sb $t0, 0xFFFF000E      #x position is also 0; store in Horizontal Position Register
li $t0, 'H'             #character to display: 'A'
sb $t0, 0xFFFF000F      #store in Character Display Register. Character is displayed
#After a command is executed, the registers are unchanged.
#This is useful when writing a string of characters that
#share the same Y position and color.
li $t0, 0x01            #Only the X position needs be changed, to 1.
sb $t0, 0xFFFF000E      #Store in Horizontal Position Register
li $t0, 'i'             #character to display: 'i'
sb $t0, 0xFFFF000F      #store (and display) character
```

Example 1. Writing "Hi" to the screen at position (0,0).

```
li $t0, 0x4800000F      #character: 0x48 ('H'), x, y: 0x00, color: 0x0F
sw $t0, 0xFFFF000C      #write this data to screen
li $t0, 0x6901000F      #character: 0x69 ('i'), x: 0x01, y: 0x00, color: 0x0F
sw $t0, 0xFFFF000C      #these four lines are equivalent to the first example
```

Example 2. Using word literals to write "Hi" to the screen.

3.3 Display commands

In the code page 437 character set, 0x00 signifies the null character and does not print. Consequentially, if 0x00 is written to the Character Register, no character is written; the other three registers, however, are interpreted as a graphical command as follows:

1. Argument Register 2 (Y) (0xFFFF000C)
2. Argument Register 1 (X) (0xFFFF000D)
3. Command Register (0xFFFF000E)

The graphical commands are diverse in function, with ample capacity for future expansion. Each, for convenience and for standardization of representation, is associated with a short, assembly-language-like mnemonic. Since different commands naturally use different amounts of data, the argument registers might be combined to represent one 16-bit argument, or both might be ignored. The syntax this document will use to represent each type of command is as follows:

- no arguments (cf. NOP): both argument registers are ignored.
- one 8-bit argument (cf. FNT X): only the first argument register is used.
- two 8-bit arguments (cf. RES X,Y)
- one 16-bit argument (cf. CLR XY)

Each command will be presented in a table in the following pages. The upper left corner of each table will contain the byte representing the command that must be written to the Command Register.

3.3.1 List of commands

0x00:	NOP *(no operation)*
Description:	Nothing happens.
Arguments:	none.
Delay:	none.
Example:	`li $t0, 0x00000000` `sw $t0, 0xFFFF000C #nothing happens`

0x01:	CLR XY *(clear screen)*
Description:	Erases all characters on screen and sets the screen background color to the 16-bit value in arguments 1 and 2. The 16-bit value is in the 5-6-5 "highcolor" format.
Arguments:	XY: new 16-bit background color for terminal
Delay:	125 instructions.
Example:	`li $t0, 0x00017BE0` `sw $t0, 0xFFFF000C #screen becomes olive-colored`

0x02:	RES X,Y *(resize screen)*
Description:	Changes the size of the display to X characters by Y characters. The window of the KGAS is resized to accommodate the change. The screen is erased by this operation. Note: the minimum possible terminal size is 40x25 and the maximum possible size is 255x128.
Arguments:	X: new width Y: new height
Delay:	7,500 instructions.

Example:	`li $t0, 0x0002601A`
	
	`sw $t0, 0xFFFF000C #expands console to width 60h (96), height 1Ah (26)`

0x03:	FNT X *(set font)*
Description:	Changes the adapter's font, and changes the size of the display (and window) to accommodate the new font. The screen is erased by this operation.
Arguments:	X: if 0, reset font to default; otherwise, use Apple II font.
Delay:	7,500 instructions.

Example:	`li $t0, 0x0003FFFF`
	
	`sw $t0, 0xFFFF000C #window shrinks and goes blank; Apple II font used`

4 Implementations of Common Display Functions

The following functions are those which I think would be most frequently used. Feel free to copy and paste them and use them in your own work.

```
#Writes string at $a0 to ($a1, $a2) with color $a3
KGASPrintString:
li $t1, 0xFFFF000C
sb $a1, 2($t1)
sb $a2, 1($t1)
sb $a3, ($t1)
KGASPSLoop:
lb $t0, ($a0)
beqz $t0, KGASPSLoopEnd
sb $t0, 3($t1)
addi $a0, $a0, 1
addi $a1, $a1, 1
sb $a1, 2($t1)
b KGASPSLoop
KGASPSLoopEnd:
jr $ra
```

Function 1. Prints a null-terminated string to the adapter.

```
.data
string: .asciiz "Hello world!"
.text

la $a0, string
li $a1, 1          #x = 1
li $a2, 1          #y = 1
li $a3, 0xF0       #color = white character (F), black background (0)
                   #See section 3.2 for all colors available.
jal KGASPrintString

li $v0, 10
syscall
#function omitted
```

Example 1. Full program demonstrating use of above.

```
#Changes the screen size to (width = $a0, height = $a1).
#Waits for screen to resize before returning.
KGASResizeScreen:
lui $t0, 0x0002
andi $a0, 0x0FF
sll $a0, $a0, 8
andi $a1, 0x0FF
addu $t0, $t0, $a0
addu $t0, $t0, $a1
sw $t0, 0xFFFF000C
KGASRSWaitloop:
lw $t0, 0xFFFF0008
bnez $t0, KGASRSWaitloopdone
b KGASRSWaitloop
KGASRSWaitloopdone:
jr $ra
```

Function 2. Resizes the screen.

```
.data
.text

li $a0, 52 #width = 52 characters
li $a1, 52 #height = 52 chars.
jal KGASResizeScreen
#note: each character is 8x16 px, so
#window becomes half as wide as tall.

li $v0, 10
syscall
#function omitted
```

Example 2. Full program demonstrating use of above.

```
#Changes the font to default (IBM) if $a0 = 0, and to the Apple II font otherwise.
#The screen will resize to show the same number of characters as before.
KGASSetFont:
andi $a0, 0x0FF
sll $a0, $a0, 8
ori $a0, 0x00030000
sw $a0, 0xFFFF000C
KGASSFWaitloop:
lw $t0, 0xFFFF0008
bnez $t0, KGASSFWaitloopdone
b KGASSFWaitloop
KGASSFWaitloopdone:
jr $ra
```

Function 3. Changes the adapter's font (and clears the screen in doing so.)

```
.data
apple_ii: .asciiz "Apple II font! 0123456789"
ibm: .asciiz "IBM font! 0123456789"
.text

li $a0, 1
jal KGASSetFont
la $a0, apple_ii
li $a1, 1
li $a2, 1
li $a3, 0xF0
jal KGASPrintString

li $a0, 2000
li $v0, 32
syscall

li $a0, 0
jal KGASSetFont
la $a0, ibm
li $a1, 12
li $a2, 1
li $a3, 0xF0
jal KGASPrintString

li $v0, 10
syscall
#functions omitted (uses KGASPrintString too)
```

Example 3. Full program demonstrating both fonts.

5 Licensing

The Keyboard and Display Adapter is released under the same MIT license as the Keyboard and Display simulator, which was written by Pete Sanderson and Kenneth Volmar. All trademarks, trade names, and product names appearing within the software or this documentation are the property of their respective owners. The default font is "Perfect DOS VGA 437", created by Zeh Fernando. It is available for download here: http://www.dafont.com/perfect-dos-vga-437.font. The Apple II font is PrintChar21.ttf, created by Kreative Korporation, and freely available at http://www.kreativekorp.com/software/fonts/apple2.shtml, under the following license:
KREATIVE SOFTWARE RELAY FONTS FREE USE LICENSE
version 1.2f

Permission is hereby granted, free of charge, to any person or entity (the "User") obtaining a copy of the included font files (the "Software") produced by Kreative Software, to utilize, display, embed, or redistribute the Software, subject to the following conditions:

1. The User may not sell copies of the Software for a fee.

1a. The User may give away copies of the Software free of charge provided this license and any documentation is included verbatim and credit is given to Kreative Korporation or Kreative Software.

2. The User may not modify, reverse-engineer, or create any derivative works of the Software.

3. Any Software carrying the following font names or variations thereof is not covered by this license and may not be used under the terms of this license: Jewel Hill, Miss Diode n Friends, This is Beckie's font!

3a. Any Software carrying a font name ending with the string "Pro CE" is not covered by this license and may not be used under the terms of this license.

4. This license becomes null and void if any of the above conditions are not met.

5. Kreative Software reserves the right to change this license at any time without notice.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR FROM OTHER DEALINGS IN THE SOFTWARE.