HANDY RELEASE 0.7.2
15 February 1989

CONFIDENTIAL and PROPRIETARY

This document describes the V 0.7.2 release of Handy development environment. It includes these sections:

- Handebug V1.47
- HPRINT
- Display Fix?
- Interrupt Handler
- Miscellaneous
- What You Must Do To Start Using This Release

Handebug V1.47

The major feature added to Handebug for this release is the ability to save and restore the current configuration (known and promised for long as "settings" and here she comes at last!). Immediately after being loaded Handebug looks in HANDY: for a file named Handebug.config. It will process this file to establish colors, the default path for .bin, .sym and other .config files, and other things.

RJ sez: by this time tomorrow, no one's Handebug will have the old colors!

Currently supported settings are:

HOME path specification
BINFILE filename
SYMFILE filename
COLORS rgb rgb rgb rgb
FOLLOWPC ON/OFF
HIDEWINDOW left/top/width/height
SCREENISHIDDEN
DISKNAME VOLUME/DEVICE

BREAKPOINT n $xxxx ON/OFF/CLEAR
MEMWATCH n $xxxx 8/8x2/16/OFF
GO $xxxx
REGISTER X xx
REGISTER Y xx
REGISTER A xx
REGISTER PC xxxx
REGISTER P xx
REGISTER S xx

The current configuration can be saved at any time and reloaded later. Shift-F6 can be used to load or save configurations. When saving a configuration Handebug will save: the current download directory as the HOME directory; the last size and position of the Bring-Back-Handebug window as the HIDEWINDOW values; the last requested download and symbol files as BINFILE and SYMFILE . For that matter, everything else named above is saved too.

GO, BREAKPOINT and MEMWATCH can be either hex addresses or symbolic references. If they are symbols then a SYMFILE must be specified in the config file before the labels. When saving these values Handebug will check for a matching label and if found will use the symbol instead of the hex value.

By specifying addresses of $0000, CLEAR for BREAKPOINTS and OFF for MEMWATCH fields you can reset Handebug (not stated very clearly, no?) In other words, turn everything off before saving your settings and they'll be turned off next time you run Handebug.

Just to keep people on their toes John's moved stuff around. Downloading a file now *never* loads symbols; all symbol operations are now completely removed from download activity. If you have old symbols around when you download a new file, it's now your responsibility to get rid of them if you want them gone. LoadSymbols and CLEAR SYMBOLS (Shift-F10) do just what they say without anything fancy. The ?? command gadget has been changed to LoadSymbols. Shift-F10 now clears the symbol table (used to be Shift-F9).

Here's a summary of the symbols functions:

LoadSymbols: Always allows you to load symbols
CLEAR SYMBOLS (Shift-F10): Always clears out all the symbols
Symbols: Displays current symbols. If there are no symbols, this loads symbols

Handebug now gets Handebug.defs out of HANDY: rather than S: (this release copies S:Handebug.defs to HANDY: and deletes the one in S:). Note, however, that .config loaded via Shift-F6 can exist anywhere.

This is probably the last feature-release for a while. There will be ongoing bug fixes, but future enhancement requests will be noted and added to the Real-Soon-Now list.

In case you haven't noticed, there's a new template with this release. Hope the text matches the functions!

HPRINT

Finally released: the HPRINT text generation routines. This collection of code, macros and font data declarations allow you to get easy, low-cost text into your programs. See the Programmer's Guide chapter HPRINT - HANDY PRINT ROUTINES for a description, and see 6502:examples/testhprint.src for an example of text at work.

Display Fix?

There's a new bunch of minor changes to the display macros. All of it is transparent to you, whether you are an EOF_USER, EOL_USER, both or neither. The difference is that the display address hardware registers are now hit with new addresses during "vertical retrace" after EOF. This change just may cause the occasional screen glitches that we've seen to go away, as it's suspected that the glitches come from being halfway through updating the two-byte display address while the hardware is also accessing that two-byte display address, resulting in the occasional screenful of junk.

If we still see the glitches after this release, then it's back to the drawing board.

Though these changes are small and should be transparent to most of you, the system is significantly different now: there is always some level of EOF handler created by the system,

whether you've specified EOF_USER or not. However, the EOF handler is small and does less work if you haven't specified EOF_USER.

Interrupt Handler

Several of you have inquired into what it takes to add your own interrupt handler to the system. Actually, it's quite easy. There's two parts to system interrupt handlers: what rules you must follow, and how to link into the system.

Connecting your interrupt handler into the system is easy. At the memory location IntTable is an array of 8 vectors, one for each of the 8 interrupts. When any of the 8 hardware interrupt sources generates an interrupt, the system interrupt router vectors to the appropriate routine through this table. In order to have the system vector to your routine for a given interrupt, you need only install the address of your routine into IntTable like this:

```
        ;--- Let's install our routine into interrupt 5
        SEI
        LDA   #<MyInt5Routine
        STA   IntTable+(5*2)
        LDA   #>MyInt5Routine
        STA   IntTable+(5*2)+1
        CLI
        ...
MyInt5Routine
        STZ   Int5Happens     ; do some interrupt stuff
        RTS
```

If you think you might be sharing an interrupt with other code (for instance, if you want to link into the end-of-frame handler chain), you should provide a way for your code to vector off to the old handler routine after you've done your work. An indirect jump or a bit of self-modifying code does the trick. Here's an example of the indirect jump technique:

```
        ;--- Let's install our routine into interrupt 5
        LDA   IntTable+(5*2)
        STA   MyInt5JMPVector
        LDA   IntTable+(5*2)+1
        STA   MyInt5JMPVector+1
        SEI
        LDA   #<MyInt5Routine
        STA   IntTable+(5*2)
        LDA   #>MyInt5Routine
        STA   IntTable+(5*2)+1
        CLI
        ...
MyInt5Routine
        STZ   Int5Happens     ; do some interrupt stuff
        JMP   (MyInt5JMPVector)
```

Here's an example of the self-modifying code technique:

```
        ;--- Let's install our routine into interrupt 5
        LDA   IntTable+(5*2)
        STA   MyInt5RoutineExit+1
```

```
        LDA   IntTable+(5*2)+1
        STA   MyInt5RoutineExit+2
        SEI
        LDA   #<MyInt5Routine
        STA   IntTable+(5*2)
        LDA   #>MyInt5Routine
        STA   IntTable+(5*2)+1
        CLI
        ...
MyInt5Routine
        STZ   Int5Happens      ; do some interrupt stuff
MyInt5RoutineExit
        JMP   MyInt5RTS
MyInt5RTS
        RTS
```

The rules for your handler are simple. If you touch any of the registers A, X, Y or S you must save and restore them. You don't need to bother saving and restoring the P register. Lastly, your routine should end with either an RTS or a JMP to the previous handler as described above.

Miscellaneous

New S:SavePrefs script file, does what the old S:sysconfig did (which is to save your preferences to your boot disk after you've run Preferences). There's a new S:sysconfig too, while we're at it.

New HANDY:asm release, 1.02, which handles .CHARSET files correctly.

What You Must Do To Start Using This Release

Nothing.