SHIFT838 Newsletter

NOVEMBER 21, 2015 VOLUME 1, NO 11

This newsletter is dedicated to the ongoing support for the Texas Instruments TI-99/4A and Myarc Geneve 9640 user community and is published by SHIFT838.

In this edition I wanted to re-visit emulation, specifically around the new version of MAME (Multiple Arcade Machine Emulator) since the MESS (Multi Emulator Super System) functionality has not been incorporated into MAME. I was able to pose some questions to one of the developers of MAME/MESS, Michael Zapf.

And next month's teaser to close out 2015, an interview with **John Behnke**, the coder of the original Tunnels of Doom editor as well as many other cool programs.

Thanks to all that have subscribed!

Michael Zapf Interview

Michael is one of the many developers on MAME/MESS project and has been for years. He has been a key coder in getting the many different types of peripheral devices to work within the emulator exactly as they would work on real TI and Geneve hardware. He has been very helpful to me for working with the emulator when I was coding my BBS and recently when I was coding the **Ooey GUI MESS Launcher**.

I plan on covering Michaels ***TIIMAGETOOL*** for serial port mapping in depth in a later edition of the newsletter.

Below are a couple of links for anyone that wants to read up on what this emulator can do.

http://mamedev.org/

http://www.mess.org/

Chris: When did you first get involved in the TI?

Michael: Back in June 1982, when I was still 12 years old, I started to become interested in the upcoming home computer technology. I remember that I discussed with my father which model to buy; my first idea was the comparably cheap ZX-81, but he opted for the much more expensive TI-99/4A (at that time about 1000 DM or $600). We got it with an English Training Course (audio cassettes plus data cassette and cartridge) and the usual manuals. My first game cartridge, by the way, was TI Invaders.

While my father lost interest again (was too much typing for him) I picked up speed, and in the following year (1983) I got the P-Box and floppy drive. This was just three weeks before „Black Friday"; if we had been just a few weeks later, I doubt we would have continued that way. Instead, I got a memory expansion and soon started Assembly programming.

Chris: When was MAME first released and what was the deciding factor to create this program?

Michael: MAME was first released in February 1997 by Nicola Salmoria, and its name is the acronym for "Multiple Arcade Machine Emulator". This name suggests the original objective, that is, to emulate arcade machines – those single-game cabinets with a screen, some buttons, and a joystick, found in arcade halls since the late seventies.

There are two main aspects of the MAME philosophy. First, the emulation of the arcade machines allows us to enjoy those games that are almost lost in oblivion. Second, and for most developers the more important point, MAME strives to preserve this technology as an "executable documentation". This means that you can, programming skills presumed, have a look at the code and find out how the computer actually worked. This is carried down to the single components.

Well, to be honest, I have to admit that we actually cannot reach this ultimate level and represent the internals of the real chips, not least because they are not programmed in a language like C or C++. Nevertheless, our aim is to get down as deeply as possible to the level of single signals, so you may, for example, understand how speech is produced from LPC, when the board stops the CPU with wait states, or how the video processor creates its output.

Chris: How come that there is no more MESS, only MAME? What does this mean for the users?

Michael: We should probably first sort out what are the basic parts of this huge project. The MAME project, as I said, originally targeted at emulating arcade machines. For this purpose, a generic emulation engine was built, together with a big deal of emulated circuits. To emulate a specific arcade machine you have to implement a "driver" (in MAME terminology) which includes all required circuits and which describes the lines running between them.

Later, the idea came up to use that same core engine and circuit set to create emulations of home computers and other computing systems. So in 1998, the project MESS was kicked off, its name standing for "Multi-Emulator Super System". The target of MESS is to emulate computer systems, in particular those from the Home Computer era, but also reaching back to mainframe computers with punch card input, and forward up to consoles like the Sony PlayStation. As of November 2015, MESS emulates a total of 996 computer systems, and over 2100 variants, with the TI family being a notable, but comparably small subset.

Accordingly the MESS project "borrowed" the MAME core and added more circuits as needed. Thus, every MESS computer emulation has been actually driven by the MAME core from the very start.

Until May 2012, the MESS developers had to clone the MAME code base, and then added their specific emulations to it. With more and more systems added to MESS, the situation became increasingly difficult. On one hand, there were the inevitable bugs that had to be fixed, on the other hand, the MESS maintainers also wanted to contribute some improvements to the core. However, all those changes had to be passed "upstream" to the MAME developers to be applied to the core, then it had to be cloned again, and so on.

On May 21st, 2012, the repositories of MAME and MESS were finally united, and all MESS contributors, including me, were granted access to the MAME core.

Together with this organizational issue, MAME and MESS underwent radical changes in their codebase, the most important of which was the switch from C to C++, which, consequently, required us to re-write all our implementations. As you can see on the ninerpedia website, where I recorded all TI-related changes to MESS, this happened shortly before the repository merge.

As for the recent “MAME takeover”, this is probably less significant than it sounds at first. The actual changes were to incorporate the subdirectories of the “mess” sub-tree into the MAME project structure. For that reason, when MAME is built, the MESS part is now included in the executable file. If you build it by yourself, you can still choose to create the “sub-target” MESS only.

Of course, there are ongoing discussions on the developer’s mailing list and on the forums about the public presentation. There is a general agreement that the trademark MAME has a considerable recognition, far more than MESS ever achieved, and that it could be advantageous for MESS to participate in that recognition as being viewed as an integral part of MAME. We are still looking for a better expansion of the acronym MAME, which refers to the arcade machines only, without having to change the acronym, of course.

By the way, the developers are still calling it MESS when they refer to the parts that are concerned with the computer systems. So that name is not gone yet.

Chris: What do you consider more important for emulation – user experience or precision?

Michael: Well, ultimately, highest precision should imply the best user experience. Ideally. But the answer highly depends on the rationale of the emulation as seen by the author, and also on the environment where it is run. It is neither necessary nor does it make sense to impose a “right way” on all emulations.

The MAME approach, in particular trying to achieve an emulation of circuits as close to the signal level as possible, implies some considerable host performance requirements. We already tried to compile and run MESS on a Raspberry Pi, and failed with just 5% of running speed. To make MESS run on that platform, we would have to revert all the efforts that dealt with the low level emulation, and instead, we would have to apply some tricks to squeeze out as much performance as possible. However, this would undermine the whole MAME concept.

Chris: What happened to the MESS GUI? Is the command line the suggested way of using MAME now?

Michael: I suppose you refer to the GUI that was included in MESS in earlier releases, which provided a common pull-down menu look. This menu was only contained in the Windows build of MAME/MESS, not in the SDL build (MacOS and Linux). In that sense, these were undesirable platform dependencies.

Of course, you'd say, the proper way would have been to add that support to SDL instead of letting all people suffer equally. The problem was pretty simple: The GUI maintainer left the team a while ago, and nobody else tried to continue on it.
With no more maintenance, and MAME being a continuously developed project, the GUI started to degrade, up to the point where it completely broke the whole emulation. Also, new features inside the MAME core were not accessible to the GUI. Finally someone pulled the plug.

Later, one of the developers actually tried to repair the GUI and offered a MESS version with the GUI, which was not directly supported by the main development team, however.

It is not true that the command line is the only remaining means of control. Instead, changes to the running emulation cannot be committed by the command line at all. For this purpose we have the OSD menu ("on-screen display") which is brought up by pressing the TAB key in "partial keyboard mode"; this mode is entered and left by pressing Scroll Lock.

The OSD menu has always been there, even when there was still the MESS GUI. It is definitely recommended to have a close look at the OSD and all its selections.

Chris: When and how did you get started to code the TI-99/4x, TI-99/8 and Geneve 9640 modules for MESS?

Michael: I started to work on MESS in 2007 with release 0.113. I already got to know MESS some years earlier, but obviously I was not really interested at that time, maybe because of lacking features or instability. In 2007 I found the WinUAE emulator for the Amiga and was outright impressed what could be done by emulation! So I started again to searched for TI emulations and found MESS. What was particularly interesting about MESS was that it ran on Linux, and it did not only have a TI-99/4A but also a Geneve emulation!

Sadly, I had to discover that all TI emulations were unusable, because the floppy emulation was broken. As the emulation worked with the 0.97 release I suspected that there must have been some changes that broke it. So I downloaded the source code and started my search. After some time I found the problem and reported it to the developers so that it could be fixed.

From there, I continued working on the TI emulations, part by part, adding more and more features, until today.

Chris: How many developers including yourself work on the MAME project in general, and on the TI family emulation in particular?

Michael: Different sources talk about hundreds of developers with thousands of external contributors, which is hard to prove.

In fact, I had to have a look at Github's statistics to find out. You just don't get in contact with every single person on the mailing list, and there is about a dozen people who are visibly active, with a majority of developers who stay in the background.

So if we consider the activities in the last five years, and if we only count people who at least contributed 10 times to the repository, we have 62 people. My rank is #24 with 196 commits since October 2010, about 80000 added and 56000 removed source file lines.

By the way, the developers near the top of the list show some 2500 commits and more, and the ultimate number of modified lines by a single developers is 12 million. However, these are often bulk changes, for instance, when method signatures have changed. This quickly causes thousands of lines to change in the repository in a single go.

As for the TI family – which currently includes the TI-99/4, /4A, /8, Geneve, and SGCPU – it seems as if I am indeed the sole person in charge since 2007 when I joined the team. The only exception is the CC-40 which is taken care of by another developer. This is not so uncommon; many systems are maintained by only one or a few people. Nevertheless, I still hope for some companions who would like to pick up some of the work still waiting for completion.

Chris: Did you ever contribute to other parts of MAME?

Michael: MAME contains of a lot of circuit emulations, most of which are used in numerous different systems. Apart from my work in the TI family, I rewrote the complete TMS99xx processor family. Quite some arcade cabinets make use of the TMS9980A, some of them even of the TMS9995. While I worked on these processor emulations, I learned a bit about a few arcade machines.

I also provided a little assistance to the implementation of the voice synthesis processor, the v9938, and the sound chip. The HDC9234 disk controller chip (on the HFDC) was probably the hardest piece of work for me; it is, however, only used on the HFDC and on one or two other controller boards. Also, recently I completed my works on the emulation of MFM hard disks, known from the Seagate ST-255 and similar drives that can be attached to the HFDC.

Chris: Do you have real TI/Geneve hardware in order to make comparison test between MAME emulation and the real deal? If so, please detail your setup.

Michael: My setup is an almost vanilla Geneve, only modified with the SRAM expansion that became necessary for later MDOS releases. I do not have any expansions like Genmod or PFM. In addition, my P-Box contains a serial/parallel interface card (from the Wiesbaden club), the Speech Synthesizer plugged on my self-made adapter card, an ASCSI card with a SCSI hard drive, and a DDCC-1 floppy controller from Myarc with one 3.5‟ and one 5.25‟ drive. There is a 14‟ CRT monitor attached via SCART connector; at first I used the Amiga monitor, but later I had to replace it with another one of a similar type.

I actually did some tests, especially when I rewrote the CPU emulation, to verify the wait state generation. Since I do not have a running TI console anymore (there was basically no more use since I had my Geneve), it is somewhat more tedious to do checks against a real TI console. In some cases I sent test programs to other people and had them report their results. In particular, Ciro Barile helped me to reveal the secrets of the TI-99/8 by running my test programs on his console.

Chris: Where do you see MAME going in the next few years?

Michael: MAME is going to cover more and more systems, in different directions. We see console emulations for Sony Playstation at one end, but also 50s or 60s era computing systems on the other end. Recently, emulations for physical devices are showing up, for example matrix printers, and as I heard, people are interested to adopt my proposal for floppy sounds for printer heads.

I believe we will soon see some big improvements in the user interface. Currently there are some efforts in creating the Lua interface (Lua is a scripting langauge), which allows an external program to send commands to the emulator. Some of the features of the OSD are already accessible to the Lua interface, and there is no reason why the rest won't become available. I imagine that we could write a TI-specific user interface connecting to this Lua interface, making it possible to insert cartridges by drag and drop, showing dip switches, picking disk images as desired and so on.

Chris: Will there be a F18A emulation?

Michael: There is currently no prospect for the F18A emulation. This is not because I did not like it; quite the contrary, I believe it is one of the most important developments in our community. Rather, I'm running into limitations of MAME/MESS and the hardware to run it.

As I see it, the F18A has two important features:

1. It enables VGA output. This is something that MAME/MESS is doing for free, of course, because it is running on PCs. This does not need to be emulated.

2. It allows for micro-programming and thus enhancing the video capabilities. This is the actual challenge. The FPGA itself emulates a fictitious processor, a TMS9900-compatible CPU running at 100 MHz.

Supposed that I added a proper F18A emulation; in this case a user could expect to be able to reprogram it like the real F18A. Unfortunately, the TMS9900 emulation in MAME is pretty hungry on resources because of its emulation depth. With our experiences from running the TI emulation on a Raspberry Pi, I simply cannot imagine that this implementation will ever allow to emulate a 100 MHz core.

This means that we'll have to write a completely new TMS9900 emulation just for the F18A. But will this suffice? What if there are changes to that core, for example, in the instruction set? These can be as easily reprogrammed as the rest of the FPGA.

As I see it, this would become a major project with an uncertain time scale and outcome.

Chris: What are your current and future plans for the TI emulations in MAME?

Michael: I have still got a long list to be worked down; here are some jobs that come to my mind (no priorities implied, and not guaranteed to be complete):

- TI-99/8 needs some re-write for its chipset (current work)
- UberGROM Board (needs completion)
- PGRAM
- SGCPU keyboard
- Add missing cartridges to the softlist
- State save (suspend the emulation)
- SCSI card (WH or SNUG)
- IDE card
- Corcomp controller
- MBX interface
- Direct access to the serial interface (without serial bridge)
- Hexbus floppy (so that 99/8 is fully usable)
- TI-99/4B and TI-99/5
- TI-specific User interface (via Lua)
- TMS99000 processor family
- Fix TI-99/2
- Fix Powertran Cortex

October Highlights

- Check out this month's Atariage "TI-99 Hi Score Contest" game "**CrossFire**". The winner takes home a new and sealed Defender cart in box and a copy of Panzer Strike with a full size manual.
- Ooey GUI Mess Launcher V1.1RC1 released for everyone who loves emulation!
- **9640 Menu System** for the TI-99/4A (True 80 columns) and F18A support too!
- **Floppy Days Podcast #49** released!

Software

Ooey GUI MESS Launcher

Version 1.1 RC1 released for use and feedback! Complete read-up in last months newsletter.

Topic:
http://atariage.com/forums/topic/243360-new-mess-gui-launcher/

Video:
https://www.dropbox.com/s/9poph5k2txgwjye/OoeyGUI.mp4?dl=0

9640 Menu System for the TI-99/4A

This is the best menu system I have seen for the 99/4A and with its release there was a nice little nugget for all of us 80 column users out here. The new version of Mass Transfer with 80 column and ANSI support. Now we can call Heatwave and The Hidden Reef BBS and see them in all their ANSI glory! Not to mention other ANSI based BBS' as well.

If you are an F18A user then this is a must and you need to go download it. You will love it. A full read up on this menu system can be reviewed in Volume 1 Issue #9 of the SHIFT838 newsletter.

Link to the original release thread is:

http://atariage.com/forums/topic/245321-9640-menu-system-timxt-beta-releases/?hl=%2Btimxt#entry3366775

Calling All GAMERS!

Owen Brand (***Opry99er***) has started a TI Gaming competition on AtariAge where a TI-99/4A game is chosen every month and TI'ers can compete to see who can get the highest score. At the months end the person with the highest scores receives some type of prize.

If you want to read the message thread in its entirety and possibly participate in the friendly completion then click below:

http://atariage.com/forums/topic/241547-official-ti-994a-hi-score-competition/page-1

Last month's game was : **MicroPinball**
Winner was:Vorticon with a score of **147,200** points

I did not think it would have achieved as much play as it did. But I believe MicroPinball on the 99/4a took the top slot for the GameTracker on AtariAge!

This month's game is : **CROSSFIRE**

GOOD LUCK!

Coding

Assembly Tutorial #3 – Key Input

In this program we are introducing a fair amount of new information. We will be writing small subroutines that we will branch to with a return address provided by the BL (Branch and Link) instruction. We already have learned a bit about the BLWP (Branch and Load Workspace Pointer) instruction, which we have used so far to execute TI-provided subroutines for reading and writing VRAM (VDP RAM) among other things. We will be using it again to process keyboard input in this program via a console subroutine called KSCAN, which, among other things, scans the keyboard for a keystroke and stores information in scratchpad RAM.

The last branch instruction we have yet to learn is B (Branch), which has no provision for returning to the caller as with BLWP and BL. We will use it four times in this program—once, explicitly and three times implicitly through the RT directive. The RT directive is a proxy for B *R11, which introduces a new kind of addressing, viz., workspace register indirect addressing. B *R11 means to branch to the address contained in register 11. Register 11 is special to the BL instruction. When the TMS9900 executes BL, it will save the address following the BL instruction in register 11 so we can return to it with RT (B *R11).

The unfortunate fact about the BL instruction is that it is limited to one level of branching unless we take pains to save the contents of Rll before branching to a second level. We actually do this in one of the subroutines (BSPACE) because we need to Branch and Link to a second subroutine (CURBL) from within BSPACE.

Before going any further, let's list the subroutines in the program with very brief functional descriptions:

1. **CLRSCR**—Clears the screen by filling the SIT (Screen Image Table) with blanks.
2. **DSPMSG**—Displays a message at the designated screen location. This subroutine expects the zero-based screen row in R0, the zero-based column in R1 and the address of the length word (2 bytes) that precedes the message.
3. **CURDSP**—Displays an underscore at the current cursor position, which is kept in R7 throughout the program.
4. **CURBL**—Blanks the current cursor position.
5. **BSPACE**—Reacts to entry of backspace (ASCII 8) by blanking the cursor, backing up one space (unless at the first position) displaying the cursor in the new position and awaiting a new keystroke.

The PROMPT and GREET strings are stored in the same manner as MSG in the last tutorial, i.e., a length word followed by the message. Space for the user-entered string is provided by the BSS (Block Starting with Symbol) directive, BSS 20, which reserves 20 bytes for FNAME, the label preceding BSS.

This program also is the first in which we provide a program exit. When the break key (FCTN+4) is pressed during key entry or in the infinite loop at the end of the program, the program will exit to the TI-99/4A color-bar screen.

There are a couple of things we could do with this program that would improve it. One would be including a string-length check on the input. As it stands, you can enter a string as long as you like. This will cause a problem if it is long enough to overwrite the GREET string (39 characters). Overwriting the PROMPT string doesn't matter because we've already used it.

Another improvement would be a blinking cursor during keyboard input. We will add this embellishment in the next program dealing with keyboard input.

```
*=== KEYINPUT
=====================================================
* This program does the following:
*     1. Changes screen to TEXT mode
*     2. Clears screen
*     3. Changes text and screen colors to white on dark blue
*     4. Asks for keyboard input with "Your First Name?"
*     5. Echoes what is typed, allowing
*        a. Correction with backspace
*        b. Reset to TI title screen with <break> (FCTN+4)
*     6. Displays the typed input as a greeting
*     7. Greeting remains on screen until <break> detected
*
* The following registers are used to track the indicated values throughout
*     the program. We could have used addresses, but this is not a very in-
*     volved program and registers are faster, though speed is not really
*     an issue here:
*
*     R6 = first character position on the screen of keyboard input.
*     R7 = cursor position on screen.
*====================================================
       REF  VWTR,VSBW,VMBW,KSCAN    reference E/A utilities
       DEF  START                   declare program's entry point for E/A loader

SCSTRT EQU  0            start of SIT (Screen Image Table)
SCWID  EQU  40           screen width
SCEND  EQU  960          screen size and address just past SIT
BREAK  DATA >0200        ASCII code for <break> (FCTN+4)

* Program entry point
START  LI   R0,>01B0     load R0 with TEXT mode and screen blank settings
*                        for VR01 (VDP register #1) while we change stuff
       BLWP @VWTR        Write to VR01
       LI   R0,>07F4     load R0 with screen and text color settings for VR07
       BLWP @VWTR        Write to VR07

*: BL = Branch and Link. We use it here to branch to the CLRSCR subroutine
*:    because it will store the address of the instruction following the BL
*:    instruction, allowing our program to continue.

       BL   @CLRSCR      clear screen

* Turn display back on.  We are using the console's KSCAN routine, which writes
*    the contents of >83D4 to VR01--so, first, we need to copy to >83D4 what
*    we will then put in VR01
       LI   R0,>01F0

*: SWPB = SWaP Bytes.  It is used to swap the bytes of the contents of an
*:    address or register.

       SWPB R0           get >F0 to high byte
       MOVB R0,@>83D4    copy byte (>F0) to >83D4 for KSCAN's use
       SWPB R0           restore integrity of R0
       BLWP @VWTR        write VR01

* Display prompt. To do this we need to pass 3 values to DSPMSG. There is
*     more than one way to do this. Here, we'll do it through registers.
*     We have set up DSPMSG to expect these values in R0-R2.
       LI   R0,11        load screen row
```

```
*: CLR = CLeaR contents of address or register, i.e., replace with 16 zeros.

       CLR  R1              load screen column
       LI   R2,PROMPT       RAM location of prompt text length word
       BL   @DSPMSG         display prompt and update cursor position

* Get keyboard input
       LI   R5,FNAME+2      load FNAME buffer address
       MOV  R5,R6           use R6 to track start of input
GETNAM BL   @CURDSP         call cursor display routine

*: We will use the console's KSCAN subroutine to get keyboard input. When
*:     KSCAN returns, we need to check bit >20 of the GPL status byte at
*:     >837C to see whether a key was pressed. We can then get the ASCII
*:     value of the key from >8375.  If no key was pressed, this value will
*:     be >FF.

       BLWP @KSCAN          get keyboard input
       MOVB @>837C,R0       get status byte

*: ANDI = AND Immediate. ANDI performs a bitwise AND of the register contents
*:     and the immediate value, storing the result in the register.

       ANDI R0,>2000        check status byte for key press

*: JEQ = Jump if EQual. It performs the jump if the ST (Status Register)
*:     equal bit is set.

       JEQ  GETNAM          check key input again if not
       CLR  R1              clear R1 to allow easier byte comparisons
       MOVB @>8375,R1       get character typed

*: CI = Compare Immediate. It compares the register contents with the imme-
*:     iate value and sets the ST equal bit to 1 if they are the same or
*:     resets it to 0 (clears), otherwise.

       CI   R1,>FF00        really a character?
       JEQ  GETNAM          check key input again if not
       CI   R1,>0200        <break>?
       JEQ  EXIT            if so, exit program
       CI   R1,>0800        <backspace>?
       JNE  NOTBSP          jump if not
       BL   @BSPACE         erase previous input
       JMP  GETNAM          check key input again
NOTBSP CI   R1,>0D00        <enter>?
       JEQ  GETNMX          if so, we're outta here!
       MOV  R7,R0           cursor position to R0

*: INC = INCrement by one the contents of the operand.

       INC  R7              increment cursor position
       BLWP @VSBW           echo character to display, replacing cursor

*: The addressing mode, represented by *R5+ below, is new to us.  The '*'
*:     makes the addressing indirect and the '+' auto-increments the contents
*:     of the register, i.e., the referenced, indirect address, by 1 or 2,
*:     depending on the nature of the instruction.  A byte instruction, such
*:     as MOVB, will increment by 1.  A word instruction, such as MOV, will
*:     increment by 2.
```

```
      MOVB R1,*R5+        char to FNAME and inc cursor pos & FNAME pointer
      JMP  GETNAM         check key input again
GETNMX BL   @CURBL         blank cursor
      LI   R1,FNAME+2     load start address of FNAME entry

*: S = Subtract contents of source (first) operand from contents of des-
*:    tination (second) operand and store in destination operand.

      S    R1,R5          calculate name length
      MOV  R5,@FNAME      store character count
      BL   @CLRSCR        clear screen
      LI   R0,5           load screen row
      LI   R1,10          load screen column
      LI   R2,GREET       RAM location of greeting text length word
      BL   @DSPMSG        display greeting and update cursor position

* Display First Name stored in FNAME
      MOV  R7,R0          get cursor position to R0 for VRAM destination
      LI   R1,FNAME       load address of length word
      MOV  *R1,R2         get length

*: A = Add contents of source operand to contents of destination operand
*:    and store in destination operand.

      A    R2,R7          update cursor position

*: INCT = INCrement by Two the contents of the operand.

      INCT R1             correct RAM source address of message
      BLWP @VMBW          write message to screen display

* display '!' after First Name
      MOV  R7,R0          get cursor position to R0 for VRAM destination
      LI   R1,>2100       load '!'
      BLWP @VSBW          write '!' to screen display

SPIN   BLWP @KSCAN         check keyboard

*: CB = Compare Bytes of the contents of the source operand with those of
*:    the destination operand.

      CB   @>8375,@BREAK  <break>?
      JEQ  EXIT           exit program if so
      JMP  SPIN           loop forever

* Exit program
EXIT   CLR  @>83C4         first, zero ISR (Interrupt Service Routine) hook--
*                          probably not necessary here, but it's a good habit
      BLWP @>0000         exit to console power-up routine

*== Clear Screen Routine
=================================================================
* Clear the screen by writing spaces to SIT
*
CLRSCR LI   R0,SCEND-1     we're gonna start at the end of the SIT and work down
      LI   R1,>2000       writing blank (ASCII code=[>20] must be in MSB for VSBW)
* Loop writing 1 blank at a time to the SIT
BLANKS BLWP @VSBW          write one blank to next lower SIT location
      DEC  R0             decrement by 1 the value in R0
      JNE  BLANKS         if R0<>0, write another blank

*: RT = ReTurn to caller. Actually, this is a Pseudo-instruction that trans-
*:    lates to B *R11, which uses register indirect addressing to branch to
*:    the address contained in R11 (the return address).

      RT
```

```
*== Display Message Routine
===================================================
* This routine expects the following information in R0-R2:
*     R0 = zero-based screen row
*     R1 = zero-based screen column
*     R2 = address of length word preceding message
*
DSPMSG LI   R3,SCWID       load screen width

*: MPY = MultiPlY the contents of the destination register by the contents
*:    of the source operand (register or address) and place the 32-bit,
*:    right-justified product in two consecutive registers starting with
*:    the destination register.

       MPY  R0,R3          cursor position of start of row

*: The 16-bit product we expect from the above multiplication is in R4.

       A    R1,R4          add column to cursor position
       MOV  R4,R7          we'll use R7 for the cursor position
       MOV  R4,R0          copy for display via VMBW
       MOV  R2,R1          get address of message text length word
       INCT R1             correct to address of message text
       MOV  *R2,R2         message text length to R2
       A    R2,R7          adjust cursor position to end of message
       BLWP @VMBW          display message
       RT                  return to caller

*== Cursor Display Routine
====================================================
* Display '_' (ASCII 95) at cursor position
*
CURDSP MOV  R7,R0          cursor position to R0
       LI   R1,>5F00       load '_' to R1
       BLWP @VSBW          display cursor
       RT                  return to caller

*== Cursor Blanking Routine
===================================================
* Write blank (ASCII 32) at cursor position
*
CURBL  MOV  R7,R0          cursor position to R0
       LI   R1,>2000       load ASCII blank to R1
       BLWP @VSBW          blank cursor
       RT                  return to caller

*== Backspace Routine
=====================================================
====
* Erase cursor. Adjust cursor position if not at start of input
*

*: C = Compare 16-bit contents of source and destination operands, setting
*:    or resetting the ST equal bit.

BSPACE C    R5,R6          first character?
       JEQ  BSPXIT         return to caller if so

*: At this point, we need to save R11 because we're about to call another sub-
*:    routine with BL, which will destroy our return to the main program. We'd
*:    be stuck in this subroutine forever!
```

```
       MOV  R11,R4         save return
       BL   @CURBL         call cursor blanking routine
       LI   R0,>2000       load blank
       MOVB R0,*R5          blank last-written FNAME buffer position
       DEC  R5            decrement FNAME buffer pointer
       DEC  R7            decrement cursor pointer

*: B = Branch to address represented by the operand.  In this case, *R4 means
*:     the address contained in R4.

BSPXIT B    *R4            return to caller (to saved address)

*== Various strings
==========================================================
======

*: BSS = Block Starting with Symbol and will reserve a buffer with the number
*:     of bytes that follow BSS.

FNAME  BSS  20             First Name storage--first word will be char count

PROMPT DATA 17             length of prompt text
       TEXT 'Your First Name? '
       EVEN

GREET  DATA 11             length of greeting text
       TEXT 'GREETINGS, '
       EVEN

       END  START
```

Resources

Contact information

To contact me please feel free to visit my website and click on the ‘Contact’ tab.

http://shift838.wix.com/shift838

Newsletter Topics

If you would like to participate in the writing of this newsletter or provide any topics for this newsletter please contact me via my web site.

Sites

There are a few of sites that I think should get their own list below. These are for the TI Hall of Fame and TI-99ers Unsung website. Please visit these below sites as both have great information.

http://www.ti99hof.org/index.html

http://www.ti99ers.org/unsung/

Floppy Days

Randall Kindig's Floppy Days: A great resource for PODCASTERS to listen about information about old computer systems!

These are just a few of the links available for '**Floppy Days Podcast**':

https://www.facebook.com/floppydayspodcast

https://twitter.com/floppydays

Remembrance

Also the below site has a list of all the TI-99ers that have passed. Please be sure to check them out.

http://ti99ers.org/modules/Inspire/remember.htm

Below resources are just a handful of sites that support the TI-99/4A and/or Geneve 9640 computers. It is in no way a full list. This section will be included in all future newsletters. If there is a site that you think should be mentioned then please contact me.

Web sites / FTP Sites

http://www.99er.net

http://www.ninerpedia.org/

ftp://ftp.whtech.com

http://shift838.wix.com/shift838

http://www.ti99-geek.nl/

http://www.mainbyte.com

http://www.atariage.com

http://www.harmlesslion.com

http://www.ti99iuc.it

http://www.turboforth.net

http://www.ninerpedia.org/

Yahoo List Groups:

https://groups.yahoo.com/neo/groups/TI99-4A/info

https://groups.yahoo.com/neo/groups/TI994A/info

https://groups.yahoo.com/neo/groups/Geneve9640/info

https://groups.yahoo.com/neo/groups/turboforth/info

Active BBS’

HeatWave BBS

Access: Dial-Up and Telnet
System: Geneve 9640
Software: S&T BBS Software
Location: Arizona
Content: TI and Geneve file libraries, message bases, door games and e-mail.
Telnet to: www.heatwavebbs.com port 9640 Dialup : **602-955-4491 @ 8-N-1**

The Hidden Reef

Access: Dial-Up
System: TI-99/4a Modified
Software: S&T BBS Software
Location: New York
Content: TI and Geneve file libraries, message bases, door games and e-mail.
Dialup : **718-448-9402 @ 8-N-1**

The Keep

Access: HTTP and Telnet
System: Pentium 4 running Windows 2000
Software: Worldgroup BBS Software (up to 256 user connections)
Location: Tigard, Oregon
Content: TI and Geneve file libraries, message bases, door games, multi-user and multiplayer games and e-mail.
Telnet : www.thekeep.net port **23** Web browser to http://www.thekeep.net

The Keep has TI File libraries, Message bases, e-mail, door games, multi-user and multiplayer games. The keep also has a modem line connected for anyone that would like to contact The Hidden Reef BBS from the internet through The Keep.

Simply telnet to www.thekeep.net on port 23, login to The KEEP and then type /**GO DIALOUT** at the main menu, then D1 to dial out to The Hidden Reef. It’s that simple.

Vendors

SHIFT838 – Provides used TI equipment as acquired. Check with me often. A lot of the items need rehoming from other TI Users.

Arcade Shopper – Provides old and new TI equipment, upgrades and new runs of PCBs at www.arcadeshopper.com

Repair Centers

Richard Bell

Repairs available on limited basis, please contact Richard at swim4home@verizon.net for wait-time before sending any repairs

Tim

Myarc-related hardware repairs on a limited, as-available basis. Contact Tim at insane_m@hotmail.com for wait times or to request service.