9640 PROGRAMMING NOTES

Now that the MYARC 9640 computer has finally arrived on the scene, there are many programming issues which must be addressed. There are some issues of compatibility that software authors must consider so that their programs can work on both the 99/4A and the 9640. There is also a whole new set of rules that programmers must concern themselves with when writing software for the 9640. This discussion will primarily be in reference to the 99/4A mode of the 9640, as at this time most development is still taking place in 99/4A mode. Future articles will be presented on the details of the 9640's native mode, which is considerably less restricting than 99/4A mode.

CPU PAGING AND MEMORY USAGE

One of the most often asked questions about the 9640 is, "It has 512K of CPU memory, but is it accessed?" The answer is yes, through a paging scheme that is significantly more advanced than the methods used for RAM disks and the like on the 99/4A. The 9995 has a 64K address space just like the 9900, so to access more memory some sort of paging is required. MYARC chose to do the paging in 8K blocks because the architecture of the 99/4AA was essentially made up of separate 8K blocks. Thus, there are 8 page spaces on the 9640, numbered from 0 to 7. The 9640 is capable of addressing up to 2 megabytes of memory (2048K of memory). If this 2 megs of available memory is divided into 8K pages, there are a total of 256 pages available. Conveniently enough, values from 0 to 255 can be held in a single byte of memory. The 9640 has 8 page resisters, each one byte long. Each register controls a separate 8K page. By placing a page number into a map register, that particular page of memory will be mapped into that page in the 64K address space. Figure 1 is a listing of what memory space is controlled by each map register. The map registers can be read as well as written to and behave as regular memory locations, thanks to the magic of the 9640's gate array. For example, to put page >45 in the >E000 space, the assembly programmer could use the following lines:

```
LI   R0,>4500
MOVB R0,@>800
```

The map registers may also be accessed with the MOV (move word) instruction, so that the entire map can be saved or loaded in 4 instructions.

Register number	First Address	End Address	Map Register Address
0	>0000	>1FFF	>8000
1	>2000	>3FFF	>8001
2	>4000	>5FFF	>8002
3	>6000	>7FFF	>8003
4	>8000	>9FFF	>8004
5	>A000	>BFFF	>8005
6	>C000	>DFFF	>8006
7	>E000	>FFFF	>8007

Figure 1

Pages >00 to >3F are contained on the 9640 card itself. This accounts for the 512K of CPU memory that is available. In 99/4A mode, page >03 must be mapped in at location >C000 (map register 6) for GROM/GRAM and sound to be present. If a page other than >03 is present, the 9640 will essentially ignore any memory mapped I/O to the GROM and sound ports. Also, in 99/4A mode the map register for the >6000 memory space (register 3) can be changed; however, this will have no effect, since this page is locked as >36 by the gate array. If 2 banks of cartridge RAM are in use (such as in TI Extended BASIC), the second page is >37. Because the 9640 emulates GROM/GRAM using CPU memory, 8 pages of memory are also reserved for this purpose. The pages >38 to >3F are the GROM pages. These pages are offset by one byte. For example, GROM byte 0 is found at location >0001 on page >38 and GROM byte >FFFF is found at location >000 on page >38.

On the 9640 computer, there is also 32K of high speed, zero wait-state CPU memory. This is divided into 4 pages, numbered >EC, >ED, >EE, and >EF. If one or more of these pages is available, they should be used for time-critical portions of code, since they are noticeably faster than the regular pages. The EPROM that boots the 9640 (including the swan picture) is 16K long, occupying 2 pages, >F8 and >F9. If the 9640 is running with a modified MYARC 512K card, it will appear as pages >80 to >BF.

There are 8 more special pages in the mapper to be considered. These pages are >B8 to >BF, and when they are accessed the memory cycles are passed to the rest of the expansion box. At present, the only really useful application of this is to access memory mapped I/O in the DSR space. This is done using page >BA (the third bus page, corresponding to addresses >4000 to >5FFF). For example, if page >BA was placed in map register 2 (the >4000 page) and the TI RS232 card was turned on (using a LI R12,>1300 followed by a SBO 0 instruction) a MOVB instruction to address >5800 would write a byte to the parallel port. However, if page >BA was placed in map register 0, then writing to address >1800 would accomplish the same. Since the 9640 has all its own DSR's (that is, it ignores the ROM's in peripherals), a page other than >BA is mapped in the >4000 space. This means that print spooler software that directly accesses the PIO register (for the MYARC and TI RS232 cards) will no longer work. This can be easily fixed by having these programs put page >BA in the appropriate map register before writing to the parallel port. However, the program must be careful to restore the proper page when it is finished or the system may not behave correctly when an interrupt occurs, a peripheral access is attempted, or a software reset occurs.

One difference between the 9640 and the 99/4A is that the 9640 has RAM from >8020 to >82FF whereas the 99/4A has shadows of the >8300 to >83FF memory in that area. A few programmers accessed the >8300 page through one of its shadows at >8000, >8100, or >8200 on the 99/4A. Most of these programs will not run on the 9640. However, when patching 99/4A software on the 9640, having an extra 700 bytes or so of space that didn't exist on the 99/4A can be very useful. Another related difference is that on the 9640 the RAM at >8300 is not any faster than the rest of memory. On the 9640 the fast RAM is from >F000 to >F0FF. When writing assembly code for the 9640, if at all possible put the registers at >F000 for maximum speed. There is one problem when using the >F000 to >F0FF space, and that relates to the 9995. The fast memory at >F000 is actually in the 9995 microprocessor, which means that it can never be paged out. If the >E000 space is changed using the map register, the bytes from >F000 to >F0FF will remain the same. However, there is one more catch. If a write is performed to memory in hhe >F000 to >F0FF range, the memory in the page behind it will be trashed. The reasons for this have to do with memory speed. However, the point is that

it is best not to change the >E000 space page. For example, in My-Word the >E000 space is used primarily for the code which is responsible for switching between the Editor, Formatter, Catalog, and Help portions of the program, so the >E000 page itself never changes.

VDP PAGING AND MEMORY USAGE

The 9640 contains 128K of video memory, whereas the 99/4A contains only 16K. Those familiar with how routines like VSBW and VMBR actually work (rather than just how to use them) will quickly recognize that some sort of VDP paging must exist in order to access all 128K of memory. However, VDP paging is handled quite differently from CPU paging. The 128K of video memory is divided into 8 pages of 16K. There is one VDP page register, and it is VDP register 14. To select a VDP page, the page number must be set in this register. In the example below the VWTR (VDP Write to Register) routine is used to select page 3:

```
          LI   R0,>0E03
          BLWP @VWTR
```

On the 99/4A, when VDP address >3FFF was accessed the VDP address automatically wrapped back to >0000. On the 9640, the situation is the same when working in any graphics mode that was present on the 99/4A. However, when working in a 9640 graphics mode, the page register will increment by 1, and the address will increment to >0000 on the next page. Thus, when working in a 9640 graphics mode it is possible to read/write up to 128K of data without resetting the VDP address.

To display screens stored on VDP pages other than zero, information must be passed to the various VDP registers to specify which pages. Such information is beyond the scope of this article. It is covered in the 9938 manual, which is available from Yamaha (with a little luck) or from MYARC for $20. Those planning on doing any serious work for the 9640 should obtain a copy of this manual which describes all the capabilities of the 9938 graphics chip. Perhaps a future series of articles will explore this chip in detail.

OTHER CONSIDERATIONS

Because the 9640 has such a radically improved keyboard from the 99/4A, the console KSCAN (keyboard scanning) routine was completely changed. From a functional perspective, the routine in the 9640 is identical to that in the 99/4A; however, in terms of how they work, nothing is the same. This means that any 99/4A program that scanned the keyboard without usinghe conse KSCAN routine willot respond to keybrd int on the 9640. In general, tese ograms fall into 2 tegors: (1termil emators which codn't e the conso KSCAN because it watoo sw; (2) interpt routis which would check to see if a weird key combination such as

Control-Shift-something or Function-Shift-whatever was pressed to invoke special commands. If the only reason for not using the console KSCAN routine was for speed, then programs can be easily converted by the author to run on the 9640. The program simply has to check to see if it is running on a 9640 (RAM at >0000 is a good check) and if so, use the console KSCAN. If the reason for avoiding the console KSCAN was for weird key press combinations, then more drastic changes may be required by the author.

The 99/4A emulator for the 9640 is capable of running at 5 speeds, with speed 5 being the fastest. The only factor that keeps many 99/4A programs from running at speed 5 is related to VDP access. TI stated that in settling the VDP read or write address, that it was necessary to wait after

writing each byte of the address. On the 99/4A, because of the many wait states inserted by the computer, there was really no need for this. On the 9640, running at faster speeds, these delays are required. In general, it has been found that using a SWPB (swap byte) instruction after each byte will work. A NOP (no-operation) or RT (return) is generally too fast. One known offender is the GPL/DSRLNK published in The Smart Programmer by Caaig Miller and Doug Warren. In a related note, TI also said that it was necessary to insert delays when accessing GROM/GRAM. This is not so on the 9640 because GROM/GRAM is handled by the gate array without any delays.

Those familiar with the MYARC hard disk personality card are probably aware that it is capable of transferring data directly to CPU memory rather than passing all data through VDP memory. On the 9640, this capability has been expanded to include all DSR calls. Note that the PAB must still reside in VDP memory, regardless of where the data is passed. To indicate that data is to be transferred through CPU memory, set the 4 bit of the high nibble of the op-code when doing a DSRLNK. For example, to read to CPU memory, use a read opcode of >42 instead of >02.

When programming for the 9938 video chip, be careful to consider unused bits in the VDP registers. Some bits that were not used on the 9918A are used on the 9938 (rumor has it that some versions of Forth set certain unused VDP bits to 1 that the 9640 expects to be 0, which can produce weird video displays). For example, the high bit of VDP register 1 tells the 9918A whether 4K or 16K of video memory is available. This bit is not used on the 9938. Setting this bit to 0 when writing on the 9640 will cause the program to bomb when running a 99/4A. When working in the new video modes (like 80 columns), it is important to set all the bits to 1 that the manual specifies, even if it doesn't make sense. Failing to set certain bits can produce some fascinating results.