

The Sourceror's Apprentice

The Assembly Language Journal of Merlin Programmers

Vol. 0 No. 0 November, 1988

New Apple II ML Newsletter Premieres!

Welcome to the premiere edition of "The Sourceror's Apprentice"! Even though this is the very first issue, we've had scores of anxious questions over the last few weeks. Allow us to anticipate a few of yours and provide some answers.

Q. What kind of material is The Apprentice going to cover?

We plan to cover the material our audience wants and needs. I think that those desires will change with time, so the question is somewhat difficult to answer. Nevertheless, here are some of the directions we want to go:

First, and probably foremost, we want to examine IIGS programming. Although the GS is a marvelous machine, creating an entirely 16-bit application program can be a little intimidating at first, especially to programmers fairly new to assembly language. We'll call upon several experts to walk you through the steps. We'll also be looking at developing desk accessories, taking advantage of GS/OS's power, etc.

Second, we'll have a regular column entitled, "Eight is Enough" which will cover 8-bit assembly language techniques. Although 8-bit computers are "old technology", many of the examples we'll tear apart will teach skills applicable to any programming environment. Data structures, graphics, Prodos 8, BASIC.SYSTEM added commands, interfacing with AppleSoft(tm), etc. If you give us enough time, we'll hit it all.

We will also have a monthly question and answer column, but we'll steer our features towards your needs. So give us a call. Drop us a line. Send us some code. Keep in touch.

Q. Do you fancy yourselves as successors to our beloved "Apple Assembly Lines"?

Yes - and no. The Sourceror's Apprentice will definitely be different than AAL, but I hope to maintain Bob S-C's high standards. One thing about the newsletter industry that is rather unique is that subscribers are really able to dig into the personality and brain of newsletter editors. I am definitely not Bob Sander-Cederlof. I'll be quite ecstatic the day that I am half the programmer he is. I am a teacher, though, both by profession and personality. I think readers will find my articles to be clear-headed explanations of the innards of their machines.

In addition, I am not afraid to call upon outside expertise. Programmers seem to develop "pockets" of expertise based on the programming projects they've been involved with. Some of the expertise we're gonna tap belongs to distinguished gentlemen like Glen Bredon, author of the Merlin series of assemblers, ProSEL, ProCMD, etc.; Bob Sander-Cederlof - yes, Uncle Bob promised me a few lines of code, although he did say I'd have to translate it from the S-C Assembler format!; MohawkMan, a radio DJ and newscaster who has written radio station management software, a ZBASIC programmer's utility package called ZTOOLS, and several freelance pieces; Karl Bunker, author of the public domain masterpiece DOGPAW; and Eric Mueller, the boy-wonder author of NibbleTerm, the comm program featured in the August issue of Nibble magazine.

I also hope to continue Bob S-C's tradition of open and lively exchanges of ideas. I would consider it a compliment if a reader sent improvements on code we published. And if you send us a full length article which we use, we'll send some cash your way, usually something in the \$50 - \$75 range. I know that is not world class cash, but it is significantly nicer than a pat on the back. Be sure that you ask us for our writers guidelines or download 'em from GENIE's A2Pro round table library.

Our mission, if you could call it that, is to provide a forum for the discussion of Apple II assembly language. In this respect I hope to equal or perhaps even exceed the success of AAL.

Q. The Apprentice is self-described as "The Assembly Language Journal of Merlin Programmers". Why is that and why should anybody with a different assembler subscribe?

Let me answer that with a question. Why did many of us without the SC Assembler subscribe to "Apple Assembly Lines"? There will be a lot of information in "The Apprentice". Source code will be presented in the Merlin format, but it is really pretty easy to translate such stuff, especially if you're doing it as you type the thing in.

In case you're wondering why I chose to be Merlin-specific, it is mostly because of the fine performance of the assembler itself and the company distributing it. I am NOT knocking any other company or product, but I have had several opportunities to work with Roger Wagner and found him and his employees quite agreeable. They're a great bunch, and Merlin is

More Premiere...

a great assembler. I am quite comfortable working with both.

Q. What is the exact relationship between you and Roger Wagner Publishing?

I call it "cooperative". We are two distinct entities, with absolutely no management in common. I have even told Roger that I reserve the right to disagree with RWP policies. I haven't had occasion to, but it is something I feel I should do if it might benefit my subscribers.

Q. How many pages will "The Apprentice" have and how much will it cost, etc.?

"The Apprentice" will always have at least twelve 8.5 X 11 inch pages. It will occasionally have more. We will not be carrying any advertising in the newsletter itself, although I may occasionally permit "inserts". You'll never lose any editorial space for an ad.

Because our pages are nearly twice the size of the old AAL pages, and because we won't have any advertising in the newsletter itself, the amount of information presented each month should be comparable to AAL's.

"The Apprentice" will appear in your mailbox once each month, beginning with January, 1989. The regular cost per year will be \$28, \$52 for two years. We are currently offering an introductory special which gives you one year for \$25 or two years for \$48. This special will expire December 31, 1988.

Q. Where is Unalakleet, Alaska and why would anybody want to live there?

Unalakleet is an Inupiaq eskimo village on the western coast of Alaska. Strategically located at the mouth of the Unalakleet River on the Bering Sea, Unalakleet offers the absolute best fishing in the world, in my humble opinion.

Willie Nelson, Mel Tillis, Merv Griffin, and Japanese businessmen too numerous to mention - funny names, though, Honda, Yamaha, Isuzu - you'd think they were all named after cars... - all pay tons of money to fish here. The Iditarod Sled Dog race passes right in front of my house. In fact, Bryant Gumbell walked out on the sea ice to film a feature from Unalakleet.

My wife, Tamara, works as a full time teacher for the Bering Strait School District; that is why we moved here initially. I taught, too, for six years. But the call of the Apple proved too strong for me. Besides, it gives me something to do during the long Alaskan winter night. To say you work nights in Alaska is to say nothing at all. Everybody does.

Incidentally, this newsletter was composed and typeset using my own custom PostScript routines on an Apple IIGS, AppleWorks, and an Apple LaserWriter NT.



Avoiding a False Start

I know that most of you bonzai byte bashers out there have got the Merlin 8/16 to GS connection all figured out. But stop for a moment and consider the landscape as it appears to newcomers - or some folks I know who are converting to Merlin 8/16 from other assemblers, even APW. First of all, although we do have large chunks of sample code available in the Programmer's Introduction to the Apple IIGS, aptly named HodgePodge, the examples in the text itself are in C, while the appended assembly language versions are written for the "other" assembler.

To further befog and baffle, there are not only pure toolcalls and underscored macro toolcalls, but also the new, more powerful tilde (~) macro toolcalls. And don't forget that the tilde macros are not implemented for each toolcall, but only the ones that require manipulating the stack.

And although the toolboxes are "simple" to use, they are definitely not straightforward. In fact, there is virtually NO published material I've run across that has all of the toolbox interdependencies listed correctly. The issue is so complicated that Apple tech support will be distributing a very long and painfully detailed technical note on the subject next month. At least they are finally sorting it out.

If that were not enough, the proper startup order for the tools is a bit tricky at times, as is the issue of NDA support.

So, newcomers, if you are feeling a bit overwhelmed, you have a right to such emotion. Oldsters and pros, have patience. The Apprentice will definitely try to print "serious" 65816/GS articles. But first we must lay some groundwork.

As for the issue of NDA support, I am reminded of a music teacher I had in college. As I chafed against the multitudinous rules constraining arrangers of music, I reminded her that Johann Sebastian Bach frequently "broke the rules." She reminded me, of course, that J.S.B. knew which rules he was breaking.

Although I may not agree with Apple's position on NDA support for GS applications programs, I nevertheless feel that we ought to abide by and propagate their "rules". That is not to say that they ought NEVER to be broken, but we must certainly know the consequences when we do.

I fear that the present situation might get out of hand. According to chapter 5 of the Apple II GS Toolbox Reference Manual, Volume 1,

1. All applications wishing to support NDAs should start up the following tool sets and check to make sure that they are of the minimum version number required:

Tool	Minimum Version

\$01 Tool Locator	1.2
\$02 Memory Manager	1.2
\$03 Miscellaneous Tool Set	1.2
\$04 Quick Draw II	1.2
\$06 Event Manager	1.0
\$0E Window Manager	1.3
\$0F Control Manager	1.3
\$14 LineEdit Tool Set	1.0
\$15 Dialog Manager	1.0
\$16 Scrap Manager	1.0



2. Applications not complying with #1 above should not even start up the Desk Manager.

Thems fightin' words to some programmers.

At present, though, we have a recipe for disaster. According to Apple, NDAs can assume that the ten toolsets above are loaded and of the proper age. Apple's HodgePodge program, major sections of which are emulated - read "copied" - by many plebiscite GS programmers, does NOT comply with the NDA PROTOCOL. It does indeed load the proper versions of RAM based tools, but it does not make any version checks for the ROM tools, i.e. the ToolLocator, Memory Manager, etc.

The upshot of it all is that an NDA could crash our applications if we don't follow the protocol exactly.

On the other hand, it has always been a good practice in Appledom to assume that "the other guy" has not followed Apple's protocols or cleaned up after themselves. How many times have we watched 8-bit software choke because it assumed the softswitches were set a certain way? The point here is that NDA authors would probably be wise to assume nothing. Make sure of the environment. Do status and version calls.

More Mohawk...

For these reasons, Mohawk Man and Co. have elected to try to adhere to all of Apple's protocols. They have done numerous things you won't see elsewhere just to play by the rules. How many folks start up the ADB Toolset or the Miscellaneous Tools? Even though those are "pre-started" at present, Apple, Inc. says to start them up for future compatibility reasons.

And just in case your predecessor in the GS's memory banks didn't clean up after themselves, our Generic Startup routine traps the duplicate startup error which the Event Manager can return.

This article and program goes the extra mile. You don't necessarily need to. But make certain that you know the potential consequences of your actions.
'Nuff said.

A TIP: Set up Merlin so that it is not case sensitive. You can easily change the PARMS file to take care of this - see p.170 of the manual.

-- The Editor

A Generic StartUp/Shutdown

by Mohawk Man



This article and program serves two purposes. First, it will introduce you to the idea of programming the GS by using the built-in Toolbox routines. Second, it will provide a base on which to build future Apple II GS programs. One thing you'll notice when you look at GS source code is that it's generally much longer than IIe/c source code. One reason for this is that before using any of the Toolbox routines in the GS, you must start them up, and in some cases, load them from disk to RAM before they can be used. Thus, we come to the main reason for this generic startup routine.

When GS programs are presented in the future issues of *The Apprentice*, they won't include a startup or shutdown segment. Instead, the program/article will give you a list of tools needed and you can modify a copy of the Generic Startup routine for that program. This will save you typing, and save room in *The Apprentice* for the "meat" of the program. So, if you don't already have a standard startup routine available, it would be a good idea to type this one in.

Before I get into describing the program listing, let's just touch base on what Toolboxes actually are. Every 8-bit Apple programmer worth his salt knows that to clear the text screen you do a JSR \$FC58. The routine for clearing the screen is built into the Apple II ROM. The Toolbox on the GS works in much the same way, but includes routines that are far more complicated (and useful) than just clearing the screen. There are individual Toolsets for the GS with individual tools in each set. For example, in the Sound Toolset there are 25 different tools that do everything from starting the Sound Manager to playing a digitized sound.

Using these tool sets are very easy. The basic routine is to push any needed result space onto the stack, push any parameters, load the X-register with the tool number, and do a JSL to \$E10000. For example, to start up QuickDraw II, you would do the following:

PushWord DPAddr	;address of direct page for QD
PushWord #\$0	;0 means 320 mode - \$80 means 640
PushWord #160	;number of bytes per line
PushWord ProgID	;ID as returned by the Memory Manager
LDX #\$0204	;QuickDraw startup call
JSL \$E10000	;and do the call

There are macros available that cut down on a lot of this typing. Instead of doing the last two lines as shown (LDX and JSL), you would just type `_QDStartUp` and the macro would take over from there. Also, there are "smart" macros for Merlin that make tool calls much easier. If you used the new macros, the Quick Draw startup segment would look like this:

```
~QDStartUP DPAddr;#0;#160;ProgID
```

Because there is less typing, there's less chance of making a mistake. These macros will be used in Generic StartUp and just about all other GS programs in *The Apprentice*. If you don't have these super macros, they are available on the \$15 Extras Disk from Roger Wagner Publishing. Until you get them, you can still use the programs published here. Just dig out your Toolbox Reference Manuals - a "must have" for the GS programmer - and push all the result space and parameters yourself.

The listing of Generic StartUp is very straight forward. The first thing we have to do is to start the Tool Locator. Without that, we wouldn't be able to start the rest of the needed tools. The Memory Manager is then started and returns the Program ID. This ID will be used throughout the program when calling different tools. Next is a NewHandle call that allocates several pages to be used as Direct Pages for the following Toolsets. That's one nice thing about the Apple II GS; you can have as many Direct Pages as you need, formerly known as "zero pages". You're not stuck with just one, as you are with the rest of the II family.

After QuickDraw II and the Event Manager are started, we shift gears. So far all the tools we've started have been built into the ROM of the GS. Now, we have to load tools that are disk based. To do that, we make a LoadTools call. The parameter passed to that call is the address of a table of the tools we'd like loaded. These tools are located in the SYSTEM/TOOLS subdirectory of the boot volume. If that volume cannot be found, Generic StartUp will ask the user to put that disk online and then try again to load the tools.

When the tools have been loaded, we then go through them one by one and start them up. Some of them need direct page space. This space has already been allocated with the NewHandle call.

Editor: which means it is probably easiest to plan ahead and figure out the total number of direct pages you'll need before you even begin.

We just have to tell the tool where their direct page starts. This is accomplished with our IncDP macro, which loads the accumulator with the current address of the direct page space, DPAddr, and adds to that the number of pages that the last Toolset needed. That will give us the address of the start of direct page space for the current Toolset.

Generic Startup starts all of the tools needed for the average IIGS desktop application. I planned at first to start ALL the tools and then let you weed out the ones that you didn't need. But most applications won't need the MIDI tools or ACE, Audio Compression and Expansion, tools or other specialized tools so it would be needless typing on your part - and on mine! Adding a tool to Generic Startup is very easy. It takes just a few simple steps. First, add \$100 to the variable TotalDP for each toolset that needs a direct page. Some do, some don't.

Editor: This is yet another reason to purchase the Toolbox References - both volumes. If you want to use the toolsets, these two humongous books are indispensable.

If the tool is disk based, you'll have to add it to the list in the ToolTable so it'll be loaded from disk. Then add the lines to actually start the tool up. Make sure that you place it in the correct location in the listing as some tools need to be started in a specific order. Check the interdependencies and startup order described in the Apple IIGS ToolBox Reference, pp. 2-4, and check out the pending tech note from Apple.

The second part of the program is just as important as the first. It's the ShutDown routine. This simply shuts down all the tools that were started in the first part of the program. Be sure that the same tools that are being shutdown match the tools that were started!

More Mohawk...

Editor: We uncovered a bit of a controversy regarding the shutdown sequence. It seems that Mohawk Man discovered that the IIGS releases all of a program's memory when the memory manager is shutdown. Using ~DisposeHandle to release memory acquired by your program seems to be redundant. In Exploring the Apple IIGS, Gary Little promotes the same line of thought since his own startup/shutdown sequence does not use _DisposeHandle to get rid of acquired zero page memory. My own research indicates that the memory manager does indeed release all memory associated with your program when it shuts down.

In order to be sure of ourselves, we put in a call to Developer Technical Services. They responded with the usual song and dance about it being "...good programming practice to clean up after yourself." They also stressed that, since this is one of Apple's "rules", a change in the system could render software incompatible.

To me, their latter argument is the only one with substance. Good programming practices are not carved in stone. They depend on the environment. Though it certainly is important to clean up after yourself, there is no reason to if the system is doing it for you - that is, if you are CERTAIN that the system is doing it.

But since the system may well change at a later date, it is probably best to follow Apple's guidelines. At the very least, be forewarned that there are always potential problems when you don't do things Apple's way. Of course, that has never stopped Apple's own programmers...

As a final note, be sure and scrutinize Mohawk Man's IncDP macro. It is fairly unique, I think.

```
1 *****
2 *
3 * Generic Start
4 * By Mohawk Man, Ross Lambert,*
5 * and Eric Mueller
6 * This program is public
7 * domain...
8 *
9 *****
10
11
12
13 * Stuff for Merlin
14
15
16 mx %00 ;full 16-bit mode
17 rel ;generate relocatable output
18 dsk generic.start.1 ;name of output file (link file)
19 use gen.strt.macs ;run MACGEN to generate macro
file
20
21
22
23 *
24 * Our equates:
25 *
26
```



```
79      CMP    #TLminv      ;meet or beat minimum?
80      BCS    :1           ;if not, we're history
81      JMP    ShutDown
82
83 :1    ~MMStartUp        ;start the Memory manager
84      ldx    #2
85      jsr    Check4Error
86
87      pla                    ;retrieve our program ID
88      sta    ProgID
89
90      ~MMVersion          ;check for minimum version #
91      PLA
92      CMP    #MMminv
93      BCS    :2
94      JMP    ShutDown
95
96                      ;next, get direct page memory for tools
97
98 :2    ~NewHandle #TotalDP;ProgID;#Attr;#0
99
100     ldx    #3
101     jsr    Check4Error
102
103     PullLong DPptr      ;get handle and store in our direct
page
104     LDA    [DPptr]      ;long indirect load gets new dp
address
105                      ;for tools
106     STA    DPAddr       ;store it
107
108     _MTStartUp          ;start Miscellaneous Tools
109                      ;not absolutely necessary as of this
110                      ;writing, it's recommended by Apple,
Inc. to
111                      ;insure future compatibility
112
113     ~MTVersion
114     PLA
115     CMP    #MTminv
116     BCS    :3
117     JMP    ShutDown
118
119
120 :3    ~QDStartUp DPAddr;#scrnsize;#0;ProgID
121     ldx    #4
122     jsr    Check4Error
123
124     IncDP $300          ;QD uses three pages
125     ~QDVersion
126     PLA
127     CMP    #QDminv
128     BCS    :4
129     JMP    ShutDown
130
131
132 * startup Event Manager
133
134 :4    ~EMStartUp DPAddr;#20;#minx;#maxx;#miny;#maxy;ProgID
```

```
135          BCC    cont1
136          CMP    #$0601          ;ignore already started error
137          BEQ    cont1
138
139          ldx     #5
140          jsr     Check4Error
141
142 cont1      IncDP    $100          ;event manager used one page
143
144          ;NOTE: Event Mgr. v 1.0 is
sufficient
145          ;      for NDA support, so we'll
not
146          ;      waste time cking the
version #
147
148
149 * Use QuickDraw to tell user what's Up
150
151          ~SetBackColor #0          ;background text color
152          ~SetForeColor #15        ;text color
153
154          ~MoveTo #20;#90          ;position text to x=20,y=90
155          ~DrawString #MSGA
156
157
158 *
159 * load the RAM based tools
160 *
161
162
163 LoadTools ~LoadTools #ToolTable
164          bcc     StartTools        ;if the tools loaded okay, start them
165
166          cmp     #$45                ;was error VOLUME NOT FOUND?
167          bne     AbortStart          ;if not, we be dead
168          jsr     GetBootVol          ;get the boot volume for the tools
169          cmp     #1                  ;'Okay' picked?
170          beq     LoadTools          ;if so, try to load the tools again
171
172 AbortStart pla                      ;get rid of current return address
173
174          ~MoveTo #50;#90
175          ~DrawString #MSGC          ;tell user we crashed
176
177
178          jmp     EarlyShutDown      ;we're outta here
179
180
181
182
183 *
184 * start the RAM based tools
185 *
186
187
188 StartTools
189
190
```

```
191      ~WindStartUp ProgID          ;start the Window Manager
192      ldx    #6
193      jsr    Check4Error
194
195      ~RefreshDeskTop #0            ;draw the screen/ 0 = whole scrn
196
197      ~CtlStartUp ProgID;DPAddr      ;start the Control Manager
198      ldx    #7
199      jsr    Check4Error
200
201      IncDP $100                    ;Control Mgr. needs a direct
page
202
203
204
205      ~MenuStartUp ProgID;DPAddr     ;start the Menu Manager
206      ldx    #8
207      jsr    Check4Error
208
209      IncDP $100
210
211      ~LEStartUp ProgID;DPAddr       ;start Up line edit tools
212      ldx    #9
213      jsr    Check4Error
214
215      IncDP $100                    ;uses a direct page
216
217      ~DialogStartUp ProgID          ;start the Dialog manager
218      ldx    #10
219      jsr    Check4Error
220
221
222      _DeskStartUp                  ;start the Desk Manager
223      ldx    #11
224      jsr    Check4Error
225
226      _ADBStartUp                  ;recommended by Apple, Inc.
227      ldx    #12
228      jsr    Check4Error
229
230      _ScrapStartUp                ;start scrap manager for NDA
support
231      ldx    #13
232      jsr    Check4Error
233
234 *
235 * StartUp procedure is finished.
236 *
237 * Your program would continue here.
238 * This trivial example is mostly to prove
239 * that it all works!
240 *
241
242
243      ~SetBackColor #13             ;background color of txt block
244      ~MoveTo #20;#90               ;X & Y coordinates
245      ~DrawString #MSGB             ;print our second msg - press a
key
```



```

246
247 RDKEY      LDAL    $C000          ;read keyboard for a keypress
248           AND     #$00FF          ;clear high word
249           CMP     #$0080          ;keypress?
250           BCC     RDKEY          ;if not, loop back
251           STAL    $C010          ;clear strobe
252
253           JMP     ShutDown        ;if a key pressed, shut it all
down
254
255
256
257 MSGA      STR     'Loading RAM-based tools...'
258 MSGB      STR     'Press any key for shutdown...'
259 MSGC      STR     'Premature shutdown: LoadTools failure...'
260
261
262 *
263 *   Check For Errors After
264 *   Making A Tool Call
265 *
266
267 Check4Error bcs YepError          ;carry set if there was an error
268           rts                   ;otherwise, get outta here
269
270 YepError phx                      ;err on stack for _SysFailMgr
call
271           sta     ErrCode          ;save our error number for a sec
272           ~HexIt ErrCode          ;change code to a hex number
273           PullLong ErrCode        ;save and store error code
274           PushLong #DeathMsg      ;use customized death message
275           _SysFailMgr            ;oh! we're dead!
276
277 DeathMsg
278           dfb     #EndDMsg-StartDMsg
279 StartDMsg
280           asc     'Error $'
281
282
283 ErrCode     ds      4
284           asc     ' at location $'
285 EndDMsg
286
287
288 *
289 *   Get the boot volume
290 *
291
292
293 GetBootVol
294
295           _Get_Boot_Vol GBVParms    ;get the name of the boot
296
297           ~TLMountVolume #150;#50;#Prompt1;#VolName;#OKMsg;#CancelMsg
298                               ;ask the user for the boot volume
299
300           pla
301           rts                   ;retrieve the result
                               ;and get outta here

```

```
302
303 GBVParms adrl VolName          ;ptr to space for the volume name
304
305 Prompt1 str 'Insert the volume: '
306 VolName ds 18                  ;space for volume name
307
308 OKMsg str 'OK'
309 CancelMsg str 'Cancel'
310
311
312
313 *
314 * A Generic ShutDown Routine
315 *
316
317 ShutDown ent                   ;global label so other modules can die here.
318
319 _DeskShutDown                  ;kill first so that NDAs get rid of
themselves
320 _DialogShutDown
321 _MenuShutDown
322 _CtlShutDown
323 _WindShutDown
324 _LEShutDown
325 _ScrapShutDown
326
327 EarlyShutDown                  ;start here if LoadTools fails
328
329 _EMShutDown
330 _QDShutDown
331 _MTShutDown
332 ~DisposeHandle ProgID          ;gets rid of zero page memory
333 ~MMShutDown ProgID
334 _TLShutDown
335
336 :1 _Quit QuitParms
337 bra :1                          ;keep quitting if GS/OS is busy
338
339 QuitParms adrl $0
340 ds 2
341
342 *
343 * some data for the modules
344 *
345
346
347 ProgID ds 2                     ;space for the program ID
348
349 DPAddr ds 4                     ;pointer to the direct pages
350
351
352
353 *****
354 *
355 * Meaning of the double numbers in the table:
355 *
```

} Why not reverse order to Start Up?

```

357 * The first number is the tool number and
358 * the second number is the minimum version
359 * needed for our application.
360 *
361 * editor: you may need to change these if your
362 * application requires certain functions/bug fixes
363 * only available in updated versions of the tools.
364 *
365 * The version number is hexadecimalized, i.e. v 1.2
366 * is turned into $0102, v 1.3 = $0103, etc.
367 * A zero means we'll take anything.
368
369 *****
370
371
372 ToolTable dw 8 ;number of tools to load
373
374 dw 5,0 ;Desk Manager
375 dw 6,0 ;ADB Tool Set
376 dw 14,$0103 ;Window Manager
377 dw 15,$0103 ;Menu Manager
378 dw 16,$0103 ;Control Manager
379 dw 20,0 ;LineEdit tools
380 dw 21,0 ;Dialog Manager
381 dw 22,0 ;Scrap manager

```

Eight is Enough

When I first tackled assembly language, the task I detested the most was printing text to the screen. Every routine I looked at was clunky compared to the simplicity of BASIC's PRINT statement.

One day, though, I discovered a pearl beyond price - an assembly language printing routine that was simple to use and required no deformations of program flow. An embedded string printer, this routine required only two lines, a JSR and the text data immediately thereafter. The routine returned program flow to the first instruction following the text data. At last, I could print easily and quickly and not have to relegate my text to a data table - and compute a way to find it!

I found the original code in Don Lancaster's *Assembly Cookbook for the Apple II/IIe*, p. 265. Even though Mr. Lancaster's code is a masterpiece of sneakily efficient programming, I wanted more.

I wanted my subroutine to position text as well as print it. I wanted my subroutine to use ProDOS-style text data preceded by a length byte, and I wanted to exit the tiniest bit "cleaner" by saving and restoring the zero page locations used. This last requirement helped ease the conflicts I've had with the already

overcrowded zero page on 8-bit Apples. In so doing, however, I sacrificed preservation of the registers. If that is a priority for you, I suggest looking up Don's original routine.

Using my subroutine is a breeze:

```

LDY #vertical_line
;put vertical line in the Y register
LDX #horizontal_line
;put horizontal location in X.
JSR IMPRINT
;go do it
STR "This text will print!"
;text data

```

... <- The next instruction here will be executed upon the RTS!

The routine's magical quality stems from the fact that the 6502 and its brethren leave a return address on the stack whenever they encounter a JSR. As you'll see in the source listing, the IMPRINT subroutine merely adds the length of the string to the return address on top of the stack. The program then jumps to the next instruction immediately following your string data.

The byte squeezers among you will immediately see that I've wasted at least one byte - the TYA in line 49 is really unnecessary if the caller loads the vertical line into the accumulator instead of the Y register. That, I'm afraid, is a concession to clarity for my oft-confused brain. Specifying the X and Y coordinates of a point by loading the X and Y registers is something I can easily remember. You can easily change this to suit yourself.

One other nicety of the routine is that it increments its "saved" return address as it is printing your text - see lines 60 - 67. This means that you don't have to mess with additions afterward. The method used, INCs instead of ADC's, is dependent upon the fact that the 6502 series "rolls over" a byte to zero if you INC it

when it equals 255, \$FF in hex. If the INCed byte ever equals zero, we know it must have "rolled" and we should carry a 256 over to the highbyte, i.e. INC the highbyte once.

Be aware that the VTAB routine we used jumps to the value in the accumulator, NOT to the value in CV, \$25 hex or 37 decimal. Since CV is not updated by Imprint, subsequent character output to the screen by a different printing routine might not end up where you think it will. If you want to fix that, just change the equate for VTAB to \$FC22 and the TYA to STY CV.

One other item to note is that I set the variable HTAB equal to 1403 decimal, \$057B hex. You'll need to change that to 36 decimal, \$24 hex if you want to work in 40 columns.

```

1 *****
2 *
3 * Embedded String Printer
4 * by Ross W. Lambert, Editor
5 * The Sourceror's Apprenctice
6 *
7 * This subroutine is in the
8 * public domain
9 *
10 *****
11
12
13 * Stuff for Merlin
14
15         mx %11                ;8 bit emulation mode for Merlin 8/16
16         dsk EMBED.STR.OBJ
17
18
19 * A very few equates
20
21 COUT      = $FDED                ;montior char out rtn using hooks
22 VTAB      = $FC24                ;monitor VTAB rtn using contents of A
23 Ptr       = $06                  ;our zero page pointer
24 HTAB      = $057B                ;keeps horizontal pos, $24 for 40 cols
25
26
27
28 Demo      LDY #5                  ;VTAB 5
29           LDX #5                  ;HTAB 5
30           JSR Imprint              ;call subroutine
31           STR "This is my first string"
32
33           LDY #10                  ;VTAB 10 - exectution resumes here!
34           LDX #10                  ;HTAB 10
35           JSR Imprint              ;call subroutine
36           STR "This is my second string"
37           RTS                      ;end of demo
38

```

```

39
40 Imprint      * LDA  Ptr              ;get and save current zero page values
41              * STA  PtrSave
42              * LDA  Ptr+1
43              * STA  PtrSave+1
44
45              * PLA                    ;pull down return address - 1
46              * STA  Ptr              ;store in zero page pointer
47              * PLA
48              * STA  Ptr+1
49
50              * STX  HTAB              ;HTAB over
51              * TYA                    ;move vertical line into accumulator
52              * JSR  VTAB              ;and call monitor VTAB
53
54              * LDY  #0                ;init offset
55              * INC  Ptr              ;so that it points to first byte of
data
56              * BNE  cont
57              * INC  Ptr+1
58 cont          * LDA  (Ptr),Y          ;gets leading length byte
59              * TAX                    ;move length into counter
60
61 nextchar      * INC  Ptr              ;moves up to next character in string
62              * BNE  cont2
63              * INC  Ptr+1            ;increment highbyte if low rolled over
64 cont2         * LDA  (Ptr),Y          ;gets a character
65              * JSR  COUT             ;output via hooks
66              * DEX                    ;decrement counter
67              * BEQ  exit             ;all done?
68              * JMP  nextchar         ;if not, go back for more
69
70 exit          * LDA  Ptr+1            ;push return address back onto stack
71              * PHA
72              * LDA  Ptr
73              * PHA
74
75              * LDA  PtrSave+1        ;restore original zero page contents
76              * STA  Ptr+1
77              * LDA  PtrSave
78              * STA  Ptr
79
80              * RTS                    ;and go home
81
82 PtrSave       * DS    2              ;create two byte data stash
83
84

```



See p.21

Coming Attractions

- Karl Bunker Debunks Relocation Myths... DOGPAW's Karl Bunker explains and explores three different methods for generating relocatable code for 8-bit Apple II's.
- Mohawk Man Mows Down Viruses... our very own _MM_ will create code to protect your GS from nasty silicon sicknesses.
- Window Pain: Creating Custom Windows on the GS... Editor Ross Lambert will give us a PEEK into the murky depths of GS windowing.
- Eight-Bit Editor Explicated... Our fair editor will expound on the creation of an 8-bit text editor with many word processor-like functions.



Keynote address at an IBM-sponsored standards conference

The Sourceror's Apprentice

Copyright (C) 1988 by Ross W. Lambert
and Ariel Publishing, Inc.
All Rights Reserved

All programs in THE APPRENTICE are in the public domain and may be freely copied and distributed. Apple User Groups and other important folks may reprint articles upon request. Just gimme a call at 907/624-3161 or drop me a line at the address below.

Worldwide prices in US dollars effective January 1, 1989:
1 year...\$28, 2 years...\$52, 3 years...\$72

Back issues are available at \$3.00 each.

WARRANTY AND LIMITATION OF LIABILITY

I warrant that the information in THE APPRENTICE is correct and useful to somebody somewhere. Any subscriber may ask for a full refund of their last subscription payment at any time. MY LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for ANY damages in excess of the fees paid by a subscriber.

Please direct all correspondence to:

Ariel Publishing, Inc.
P.O. Box 266
Unalakleet, Alaska 99684 USA

THE APPRENTICE is a product of the United States of America.