The Sourceror's Apprentice

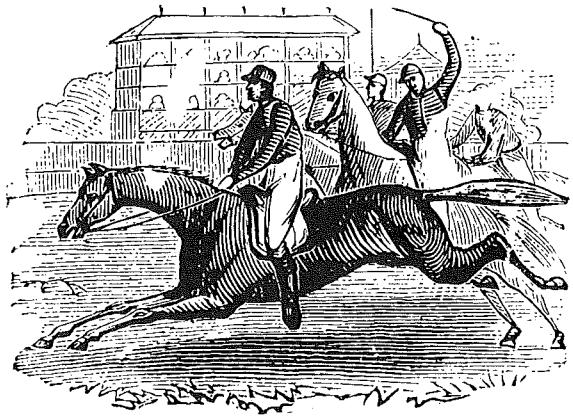
The Assembly Language Journal of Merlin Programmers Vol. 1 No. 7 July, 1989

Apple, Inc. Left in KC Dust

Normally, the mission of *The Sourceror's Apprentice* is to provide source code examples to y'all. It's that simple.

Due to the vast importance and significance of the A2-Central Developer's Conference in Kansas City, however, I'm going to deviate from the norm (more than usual) and delve into the world of editorial. Bear with me. We have source code for you this month, too, but I know that most of you are intensely interested in the II, and I think you'll want the straight skinny as reported and interpreted by another developer.

So here we go.



Some of the events of most obvious import to me were not on the schedule. No one scheduled Barney Stone's passionate rebuke of Apple, Inc. marketing strategy. No one scheduled Apple employee Eric Soldan's delivery of App.Builder, his macro-based assembly language environment for 8 bit Apples. No one scheduled the formation of the Apple II Developers Association, an event of historical importance. And no one scheduled Bill Mensch's radical support of his 65XXX series of chips - an impromptu pledge of megabuck venture capital support for language development on the IIgs.

It was a *very* interesting conference.

Contrary to what many of you predicted, Apple, Inc. sent lots of people, over 20 if I remember correctly. The only unfortunate event of the weekend was the vitriolic outbursts directed at Apple, Inc. marketing that had to be fielded by the Apple employees present. It was a little like killing the messenger. The Apple folks present were there because they love the II. I found them to be quite supportive, incredibly knowledgeable, and very patient. Apple done good by hiring Dave Lyons, Matt Deatherage, Eric Soldan, Jim Mensch, Tim Swihart, and the rest of the gang.

I think Apple, Inc. was left in the dust and outclassed by its own employees, Laser Computers, Beagle Bros., and of course, the A2-Central people (thanks, Tom Weishaar). Yes, there is a little momentum building for the II at Apple (especially on the technical side of things - System 5.0 is GREAT, the overlay card is hot, App.Builder will save us tons of development time, etc.), but until the company can figure out and execute a viable marketing strategy, Apple II sales will shrink (at worst) or remain static (at best). I am pleased that Apple seems to be making good on John Sculley's promises at the last San Francisco AppleFest. Apple's Apple II employees are doing wonderful things. But what is it going to take to get the marketing department in line? I cannot believe that John Sculley is willingly allowing a billion dollar profit center to dribble away because the marketing boys are feeding him a distorted view of the market. I could get excited about *any* serious ad campaign for the II, but I want ads revealing the II series as the fantastic general purpose computer that it is. I want dealers who support the II in the manner it deserves. I want a coherent strategy which pits the II and AppleWorks against the IBM clones, not the Mac. Since Apple saddled the IIgs with the desktop interface, I would very much like to see an ad touting that interface on the II! I want a strategy which understands the importance of color *and* cost effectiveness in educational computing. I want some marketing that is based on reality, not prejudice.

Well, I'm grateful to get that out of my system. But I have one last word: Will someone (anyone?) in Apple marketing stand up and be counted? Explain to us your rationale, your plan. As a student of marketing myself (I gave a seminar on it at the conference), I can only conclude that Apple II marketing is wandering in the wilderness.

Here were some of the highlights of the conference that are most likely to be of interest or concern to *Apprentice* subscribers:

• The Beagle Boys really did tell all about AppleWorks and TimeOut internal routines and structures. They gave us the tools (and the encouragement) to develop TimeOut compatible applications for them to publish or for us to publish ourselves (for a 4% royalty, a most reasonable fee, I'd say). You won't read any details within our pages however; everyone present had to sign a non-disclosure agreement. I doubt the opportunity to get at this data will come around again (at least until next conference). If the Beagle Bros. offer this again, I cannot over-encourage you to be involved.

• Laser Computers wowed the crowd with some fantastic hardware (the EX2 will be shipping in mid-August) and fantastic support for developers. Not only do they have a hardware purchase program for commercial developers (there is no fee for the developer program, either), but they also announced a developer co-marketing project. This venture is simply wonderful; if you can produce a four color ad and submit it to them by September 11th, they will reproduce the thing free in a catalog that goes out with every single Laser computer. Since they ship 12,000 per month, that is nothing to sneeze at. As the Laser reps said, "We're young and we're hungry." Contact Mike Wagner at (312) 540-8086, extension 744. Oh yeah, your software must be Laser compatible, but that is not an obstacle. The Laser boys have a semi-confidential list of ROM entry points they support which is free for the asking.

• Apple announced an updated BASIC.SYSTEM 1.3 which should be immediately ditched. They have had unfortunate luck with any software with those numbers (remember ProDOS 1.3?). They forgot an RTS in a routine this time, allowing the code to fall through a READ routine into a WRITE routine. DON'T USE BASIC.SYSTEM 1.3!!! An immediate update is forthcoming, of course. I guess I don't feel quite so bad about my own silly boo boos now!

• Chris Haun was showing off a version of DesignMaster compatible with System Version 5.0. It was a ridiculously wonderful package, and vastly underpriced. This is a full featured piece of prototyping software that produces your entire desktop interface including windows, text, buttons, lists, pop-up menus, and much, much more. This package is so nice that we here at the Ariel Cabin are going to buy copies in bulk and sell them to you like a dealer. I plan to sell the 5.0 version for $28 (including shipping), so you should be able to get your hands on a powerful productivity tool at a remarkable price. Prototyping software is all the rage on other computers because it helps you to program faster and better. If you do desktop programming of ANY kind on the GS, this package is a MUST HAVE. And don't forget that I said those things even before we started selling the software. I hope our price convinces you of our commitment to getting the product into your hands.

There were many, many other valuable and enjoyable aspects of the conference. Meeting Bob Sander-Cederlof in person was a joy, as was a chance to really talk with Brian Fitzgerald and John Brooks (two of the cleverest programmers in this plane of existence). Their contributions to Eric Mueller's animation seminar were terrific .

And that certainly wasn't all. I have mentioned App.Builder twice now and not explained it thoroughly. The long and the short of it is that Eric of DTS put together an incredibly powerful set of macros that make 8-bit assembly language programming a lot like working in a higher level language. Since the macros are VERY intelligent, the code they generate is quite clean and lean. The only drawback is that, due to a bug in APW macro parsing routines, the only working version is for the MPW cross-development system. Eric cut me a deal, though, and this is what we gonna do. I'm going to translate his work to Merlin format, hence I think it'll be operative on Merlin before APW!!! There is something ironic about that... I don't know how long the translation will take, but when we are finished we'll make it available to you absolutely free of charge (assuming you provide a self-addressed, stamped envelope and a disk). Let me stress that App.Builder is a productivity tool, too,

and it will make your work faster and easier.

I am exuberant and excited, but I have one last prophetic warning. Momentum is building for the II again, thankfully, but Apple has yet to reap all the consequences of years of neglect and exploitation. Computer markets turn slowly nowadays, and Apple's moderate amount of support at present will not stem the tide in the short term. The die has already been cast, if you'll forgive mixed metaphors, and some very nasty things will happen in the next year to 18 months. I think there will be profits for those able to persevere. But we must lobby Apple like maniacs to insure that they persevere as well.

Oh yeah, the KC Royals game was fun, too. Bo went 3 for 4, but the good guys lost.

On a non-conference note, we here at the Ariel Cabin are so impressed with Applied Ingenuity's hard drives (combined with their competitive pricing), we have asked to become a dealer. We've included a price list. And you thought you couldn't afford a hard drive!

Making a List (and checking it twice!)

By Ross W. Lambert, Editor

I was tempted to hold this article until Christmas just so the headline would be timely. Fortunately, I have a little bit of a Christmas present for you, anyway: the Apple IIgs™ List Manager is easier to use than I imagined.

Although the *Toolbox References* are cryptic and ambiguous regarding the List Manager, and although Gary Little calls it rarely used (*Exploring the Apple IIgs, p. 194)*, and although the *Apple IIGS Source Code Sampler* confounds the issue with extraneous code (take a lesson from educators, DTS, one major point per lesson), I was pleasantly surprised to discover that the List Manager is a friendly sort of beastie.

The List Manager's utility stems from its flexibility. Imagine a situation wherein users are required to make selections from a constantly and widely varying list (maybe based on product availability or some such changeable thing). It's a nightmare to program and format if you attempt to convert the items to buttons or other controls, and a nightmare to use if you don't ("What was the name of that item, again?").

The List Manager, however, handles these situations with ease. In fact, every time you use SFGetFile, the scrolling list of files is coordinated by the List Manager and its co-authority, the Control Manager.

Using the List Manager in its wildest permutations can get a little hairy (it can handle lists of graphical objects as well as textual lists), but in its plain vanilla form the animal is quite tame. We shall examine but one simple application of lists. Fortunately, the simplest is also the most common.

As is the usual case with toolbox routines, one of the most difficult tasks is setting up the associated data records. There are three discrete structures required by the List Manager, the list record, the item records (also called member records), and the item strings themselves.

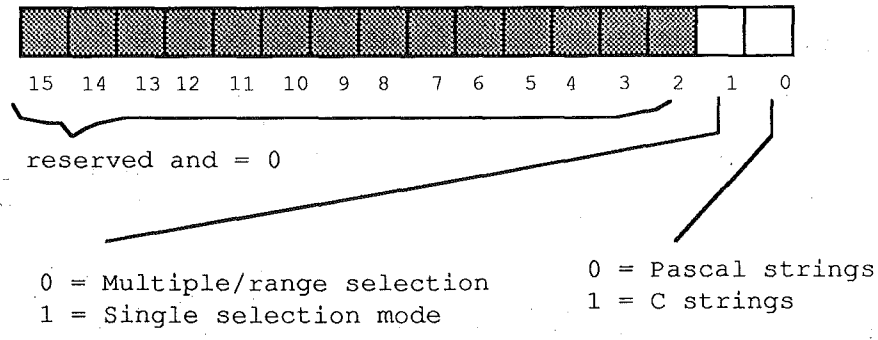
Here's what the "List Record" looks like for one of my lists:

```
ListRecord
            DA    30,30,102,230          ;RECT in which list should fit
            DA    listSize               ;total number of items in list
            DA    listView               ;total items in sight at once
            DA    listType               ;bits 0 & 1 flag parms
            DA    listStart              ;first member to highlight
            ADRL  $0000                  ;handle to list control
            ADRL  $0000                  ;drawing routine (0 = default)
            DA    listMemHeight          ;height of list members in pixels
            DA    listMemSize            ;size of member's data record
            ADRL  MyList                 ;pointer to member data
            ADRL  $0000                  ;four bytes for application use
            ADRL  $0000                  ;scroll bar color table (0 = default)
```

Most of the data is pretty straightforward, but a few items beg explanation. The item `listType` has two bit-sized flags, each of which controls something about the list. Bit zero flags the type of string in use. A zero tells Mr. List Manager that you want to use Pascal strings (which have a leading length byte). A one says, "Hey, I want C strings!", which, of course, end with a $00 terminator.

Bit one flags the type of selection allowed. Unlike SFGetFile, programmer designed lists may permit selection of a range of items and/or multiple unconnected items. Setting bit one to a one permits single item selection only, while a zero permits the range type of selection rules. All of the other 14 bits *must* be set to zero. See p. 11-12 of *Toolbox Reference Manual, Volume One* for more information.

Figure 1. listType selection flags



The `listStart` field merely determines the number of the item that you want to appear at the top of the list. This is normally item one, although if you want the user's last choice at the top, you could do it that way, too.

The item commented as the "handle to the list control" is really just space provided for the List Manager to fill in. I do not make any use of this handle at present.

`listMemHeight` is simply the height of the font used to display the list. If you're using the system font, plop in a 10. On a related note, you'll notice in my RECT dimensions that the height of my list is defined to be equal to `listView * listMemHeight+2`, or 7 * 10 + 2 which is 72. The plus two snuck in there so that the last item in the window does not rest on the line at the bottom of the list; it is purely aesthetic.

The `listMemSize` parameter refers to size of the item record you have provided for each item in the list. More on that in a minute.

`MyList` is a pointer to the item records I just referred to. Let's look at it:

```
*  Item Records

MyList

            ADRL Item1                  ;contains address of string
            DFB  0                      ;required flag byte
            ADRL Item2
            DFB  0
            ADRL Item3
            DFB  0                      ;and so on up to listSize...

 * String data

 Item1      str  'Item number 1'
 Item2      str  'Item number 2'
 Item3      str  'Item number 3'
```

`listMemSize` in our case would be five, this because each item record has a long pointer (4 bytes) to the associated string, and a required one byte flag for use by Mr. List Manager. Five is therefore the smallest possible item record. You may make each item record larger if you want. Some folks tuck in little flags for their own use.

As you can see, setting up a list 'tain't no big deal. You can even read in your data from disk or slide it in from a resource (a new feature in System 5.0 - Don't fret, we'll get to them soon enough).

Creating a list is also pretty tame stuff. Here's a fragment from a program I just finished:

```
 585 ************************************************
 586 *                                              *
 587 *    Setup List Manager Stuff                  *
 588 *                                              *
 589 ************************************************
 590
 591 SetupList
 592
 593 listSize =     16
 594 listView =     7
 595 listType =     %00000010            ;multiple and range selection
allowed, Pascal strings (leading len byte)
 596 listStart =    1
 597 listMemHeight = 10                   ;height of system font
 598 listMemSize =  5                    ;pointer to string + memFlag
byte
 599 listYtop =     30
 600 listYbot =     102
 601 listXtop =     30
 602 listXbot =     230
 603
 604          pha                        ;result space
 605          pha
 606          PushLong W1Ptr
 607          PushLong #ListRecord
 608          _CreateList
 609          PullLong ListHandle
```

```
610
611           RTS
612
613 ListHandle
614           ADRL  $0000
```

Creating a list is really just a matter of pushing a long result space, passing a window pointer and the address of your list record, and then calling `_CreateList`. Keep in mind, however, that `_CreateList` just arranges memory. Nothing is displayed on screen yet.

Which brings me to my next point: the List Manager works hand in hand with the Control Manager. In fact, a list is really a complicated set of controls. Therefore the mechanism to reveal an existing list is simply to call `_DrawControls`.

The question remains, though, *when* do you call `_DrawControls`.? The answer is: it depends. Since I use TaskMaster most of the time, I draw the controls in the routine TaskMaster calls to fill in the content of my window. Note that if you try to draw a list that does not exist your program will die!

Here's a snippet:

```
755 DrawContent
756
757           phb               ;push TM's data bank on stack
758           phk               ;push current program bank
759           plb               ;pull back to make data bank = prog
bank
760
761           PushLong WlPtr    ;refresh controls
762           _DrawControls
763
764           lda   ListFlag    ;first pass?
765           bne   :cont       ;nope
766           dec   ListFlag    ;yep, so set flag
767           jmp   :cont2
768
769 :cont     jsr   PrintSelection
770
771 :cont2    ~MoveTo #15;#15
772           ~DrawString #MsgAddr
773
774 drawout   plb                        ;and program bank
775           rtl                        ;must end with RTL!
776
777 MsgAddr   str   'Here is a list for you to play with:'
778
```

You probably noticed the `JSR PrintSelection` in line 769. I'll tell you more about that routine in a few paragraphs.

Now that the Control Manager has displayed our list, we must figure out how to operate the thing. This is our Christmas present, however; the Control Manager already knows what to do!

Using TaskMaster's event code, I branch to my routine that handles a mouse down in the content area of a window. Once there we find and track a control (which we have to do whenever there's a control in a window), and if the user selects an item, handle the

selection (you don't even have to do *that* much). By the way, the fragment below assumes that our list is the only control in the window. If you need to check for a hit in the list (as opposed to a button or some other control), the Control Manager "part code" for a list is $88.

```
158 InContent                   ;find out if we clicked on a control
159
160          pha                ;result space
161          PushLong #FControlHandle
162          PushLong EventWhere
163          PushLong W1Ptr
164          _FindControl
165
166          PLA                ;A has code for control clicked in if any
167          BNE   onControl
168
169 exit     rts
170
171 onControl PHA                ;push it back on stack
172          PushLong EventWhere ;and position-via TaskMaster's records
173          PushLong #-1        ;default action proc
174          PushLong FControlHandle ;and handle to found control
175          _TrackControl       ;track the bugger
176          PLA                 ;still on a control?
177
178          beq   exit          ;if not, leave
179                              ;CMP $88 here if you need to distinguish
list from other controls
180          phb                 ;save TM's databank
181          phk                 ;push prog bank
182          plb                 ;pull down into data bank register
183
184          jsr   PrintSelection    ;if so, tell us what we picked
185
186          plb
187          rts
188
```

As I mentioned earlier, you don't really need to handle item selections until the user is all done with the list. The dynamic duo (the List Manager and the Control Manager) will keep track of what is selected and what is not. In my examples, I wanted to let the user know what item has just been selected (redundant, since it is highlighted in the list, but it is to prove a point).

The point I'm talking about is the technique for determining which item is selected.

The `_NextMember` call does this job. The only tricks involved are to push a long word result space, the number of the item to start searching with, and the address of the list record (see lines 797-801 below). `_NextMember` searches the list from the item number you provide to the end of the list. If it finds a selected item, it returns a pointer to the item record (also called a member record) of the next selected member found.

```
790 PrintSelection
791
792
793          phd                ;save TM's direct page
794          lda   DPAddr       ;get our DP
795          tcd                ;put into DP register
```

```
796
797          pha
798          pha               ;result space
799          PushLong #0       ;search from beginning
800          PushLong #ListRecord  ;pass list record location
801          _NextMember       ;scan list for selected item
802
803          pla   MemberHandle
804          sta   MyDP
805          pla   MemberHandle+2
806          sta   MyDP+2
807
808 :cont     ldy   #2
809          lda   [MyDP],y    ;get the pointer the handle points to
810          tax
811          lda   [MyDP]
812          sta   MemberHandle
813          stx   MemberHandle+2
814
815          pha
816          pha               ;long word result space
817          _GetPort
818          PullLong OldGrafPort
819
820          PushLong W1Ptr
821          _SetPort
822
823          ~MoveTo #125;#125
824          ~DrawString #ClearLine ;yes, I'm lazy - these are spaces
825          ~MoveTo #20;#125
826          ~DrawString #SelectStr
827          ~DrawString MemberHandle
828
829          PushLong OldGrafPort
830          _SetPort
831
832          pld                    ;restore DP for TaskMaster
833
834          rts
```

In lines 801 to the end I indulge in some weirdness because I want to find and display the actual text of the selected item. If you go back to the item record section, you'll see that an item record contains a pointer to the actual string data. Hmmm... if `_NextMember` returns a pointer which points to a pointer, doesn't that make the value it returned a handle? Yes, in a manner of speaking. And no - you only need to consider it a handle if you're looking for the string data like I did above. In that case, I dereferenced it and used the result to print some text.

A reasonable alternative might be to make your item records 6 bytes long and use the extra byte to hold the number of the item. The value returned by `_NextMember` plus 5 would yield the item number. Hence the new structure:

```
*  Item Records

MyList
           ADRL Item1                ;contains address of string
           DFB  0                    ;required flag byte
           DFB  1                    ;item number
```

```
ADRL  Item2
DFB   0
DFB   2                          ;item number
ADRL  Item3
DFB   0                          ;and so on up to listSize...
DFB   3                          ; item number
```

And the new code:

```
797          pha
798          pha                 ;result space
799          PushLong #0         ;search from beginning
800          PushLong #ListRecord  ;pass list record location
801          _NextMember         ;scan list for selected item
802
803          PullLong MemberPointer
804          lda   MemberPointer+5   ;retrieves item number selected
805          sta   ItemSelected
```

Remember that if you change the size of the item records, you also need to update your list record such that `listMemSize` is set equal to the correct size.

GS assembly code seems to be self-replicating at times. A simple procedure can easily expand into 500+ lines, or so it goes for me. I found it refreshing to discover that I could make use of the list display capabilities in the toolbox with such a small expenditure of time and energy. I hope you can make use of this, too.

- Ross

An ORiGinal Letter

By Steve Stephenson

Editor: It is probably hard to appreciate Steve's idea here on nested ORGs unless you've ever had to write code that required it. I had a contract job once that required some custom quit code for ProDOS 8. The tricky part was that I had to install different versions based on the type of host machine (yuk), hence I had to write code that would be loaded at $2000+, relocated to the dispatcher space, and then pulled down and run at $1000. Since there were several versions (all of which had to be ORGed at $1000), it got messy. Steve's ideas would have helped me a lot way back when.

Dear Ross,

The other day one of my friends asked me about nesting orgs with Merlin. I have had some experience in that area and whipped up this little code fragment and uploaded it to our local BBS for him. Later I realized that this is an area of Merlin that leaves a little to be desired and that it is not adequately treated in the documentation; just the kind of thing to share with others who may run into the same situation. The end result is this letter and the code that might interest you...

First of all, the situation: You are writing a program that will sit on the disk in a single file: it has several sections that will be moved to other areas of memory (other than where it is loaded) at runtime; some of these sections need to be modified for some reason (perhaps for different computer models); you are therefore trying to reference those areas at runtime.

Now the problem: Although Merlin supports the concept of ORG ADDR to let the various sections attain their respective runtime addresses, the only pseudo op that Merlin has to return to the previous address is ORG. And actually that is a misnomer; ORG does NOT return the PC address back to the previous address (prior to ORG ADDR), instead it returns the PC to an address relative to the INITIAL ORG. This is very similar conceptually to the Basic.System GETBUFR and FREEBUFR in that you may incrementally go deeper with the GETBUFR call, but FREEBUFR pops ALL. Merlin's method is entirely satisfactory if you only want to change the address once and come right back, but you need another method to nest more than one level with any degree of fluidity.

The solution: Don't use ORG; instead always use ORG ADDR. How simple! Gee, wish I had thought of that... but no so fast. How do you know what address to use? Well, the method I use is to calculate and reset back to the previous level's relative address when I finish a nested section. To calculate the address, you will need three labels. I always name the section (PRIOR to the re-ORG) so I have an address that is relative to the previous section. The re-ORG it wherever you need (using absolutes or constants). Then insert another label for the start of the section; this one will be relative to the nested section's addresses. After the code of the section, add another label as an end marker (also relative to this section). To calculate for the reset, subtract the end marker from the beginning marker to determine the number of bytes used and add that to the address that existed prior to entering this section. What you will have is the address that you would have had if you'd not changed the ORG.

Referring to the source code will make this MUCH clearer...

On a totally different subject... I have enclosed a disk with a program I put together for fellow Merlin-16 programmers. I thought you and your readership might be interested. Feel free to do whatever you wish with it.

Sincerely,

Steve Stephenson

Full-time, Independent, Free-lance Apple II Assembly Language Programmer (Currently specializing in the IIGS)

Editor: The program Steve was referring to is called Dialog Maker. It is an exceptional public domain dialog creation utility which generates Merlin 16 source. I'll put it on the quarterly disk and copy it for anyone that sends me a disk and SASE.

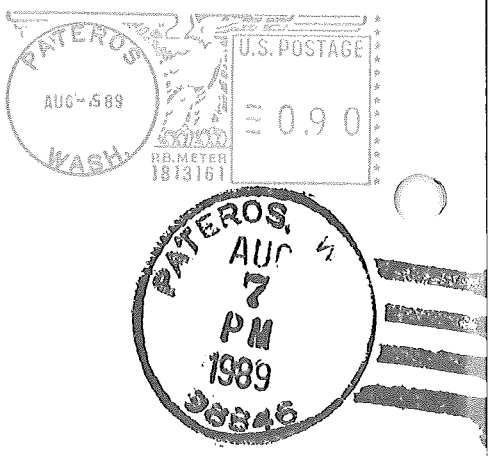
```
                        2   *================================================
                        3   * How to run nested orgs with Merlin
                        4   *================================================
                        5            tr    adr
                        6            org   $1000
                        7   Start
1000: EA                8   :a       nop              ;$1000
1001: EA                9   :b       nop              ;$1001
                       10   *------------------------------------------------
                       11   z1                        ;$1002
                       12            org   $2000
                       13   z1:beg                    ;$2000
2000: EA               14   :a       nop              ;$2000 ($1002)
2001: EA               15   :b       nop              ;$2001 ($1003)
                       16   *------------------------------------------------
                       17   nest1                     ;$2002 ($1004)
                       18            org   $3000
                       19   nest1:beg                 ;$3000
3000: EA               20   :a       nop              ;$3000 ($2002) ($1004)
3001: EA               21   :b       nop              ;$3001 ($2003) ($1005)
```

```
                        22  *-----------------------------------------------
                        23  nest2                       ;$3002 ($2004) ($1006)
                        24           org   $4000
                        25  nest2:beg                   ;$4000
4000: EA                26  :a       nop               ;$4000 ($3002) ($2004) ($1006)
4001: EA                27  :b       nop               ;$4001 ($3003) ($2005) ($1007)
                        28  nest2:end                   ;$4002 ($3004) ($2006) ($1008)
                        29  * now to "un" org back one level, count up the number
                        30  * of bytes at this nested level and add the address
                        31  * of the beginning of the nested area (using address
                        32  * relative to previous level).  In this case,
                        33  * nest2:end - nest:2beg ($4002-$4000) = 2 bytes,
                        34  * nest2's address is $3002, so the next pc address
                        35  * will be $3004
                        36           org   nest2:end-nest2:beg+nest2
                        37  *-----------------------------------------------
3004: EA                38  :a       nop               ;$3004 ($2006) ($1008)
3005: EA                39  :b       nop               ;$3005 ($2007) ($1009)
                        40  nest1:end                   ;$3006 ($2008) ($100a)
                        41  * $3006-$3000 = 6 + $2002 = $2008
                        42           org   nest1:end-nest1:beg+nest1
                        43  *-----------------------------------------------
2008: EA                44  :a       nop               ;$2008 ($100a)
2009: EA                45  :b       nop               ;$2009 ($100b)
                        46  z1:end
                        47  * $200a-$2000 = $a + $1002 = $100c
                        48           org   z1:end-z1:beg+z1
                        49  *-----------------------------------------------
                        50  Reality
100C: EA                51  :a       nop               ;$100c
100D: EA                52  :b       nop               ;$100d
                        53  *-----------------------------------------------

End Merlin-16 assembly, 14 bytes, 0 errors, 53 lines, 11 symbols.

Elapsed time = 14 seconds.

Symbol table, alphabetical order:

   NEST1    =$2002      NEST1:BEG=$3000     NEST1:END=$3006     NEST2    =$3002
   NEST2:BEG=$4000      NEST2:END=$4002     Z1       =$1002     Z1:BEG   =$2000
   Z1:END   =$200A

Symbol table, numerical order:

   Z1       =$1002      Z1:BEG   =$2000     NEST1    =$2002     Z1:END   =$200A
   NEST1:BEG=$3000      NEST2    =$3002     NEST1:END=$3006     NEST2:BEG=$4000
   NEST2:END=$4002
```

Ariel Publishing

Box 398
Pateros, WA 98846

509/ 923-2025



The Sourceror's Apprentice

Subscription prices in US dollars *(Canadian and Mexican subscribers add $5 per year, all other non-North American subscribers add $15 per year)*:

1 year ...$28 2 years...$52

Ross W. Lambert.... Editor & Publisher
Tamara Lambert..... Subscriptions
Jay Jennings, Eric Mueller, Robert Moore..... Tech Editors
Rebecca Lambert... Stamp licking

WARRANTY and LIMITATION of LIABILITY

I warrant that the information in *The Apprentice* is correct and somewhat 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
P.O. Box 398
Pateros, WA 98846

509/ 923-2025

The Sourceror's Apprentice is a product of the United States of America.

We here at Ariel Publishing freely admit our shortcomings, but nevertheless we strive to bring glory to the Lord Jesus Christ.