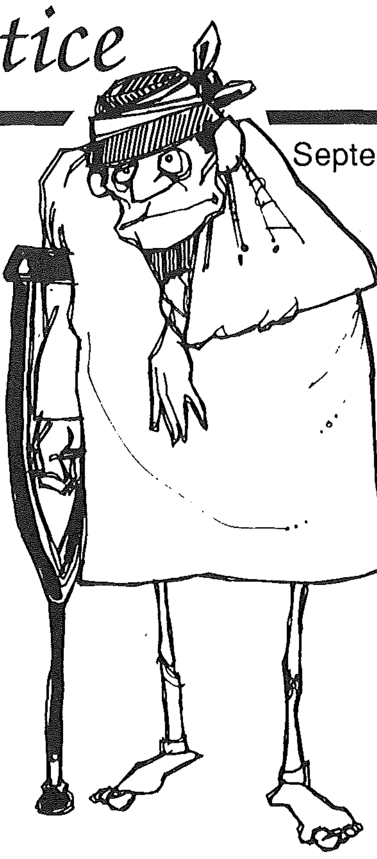
The Sourceror's Apprentice

The Assembly Language Journal of Merlin Programmers

September, 1989



Split Stitches and Other Emergencies

I didn't really know the meaning of the word "behind" until I went into business for myself. No sooner did I get a business license than I became like the butcher who backed into the hamburger machine - he got a little behind in his work.

All yuks aside, I got waylaid in September by some surgery and a trade show, hence we are even behinder than normal. My apologies - and don't fear. We've not evaporated or gone belly up (boy, some of you are nervous, although I understand the concern. This is a difficult time in the computer industry, a real dog eat dog business to begin with.) Did any of you see us at AppleFest? It was a profitable adventure, but extremely tiring. Not only that, but two of Apple's finest, Greg Branche and Tim Swihart, got me laughing so hard that I literally split my stitches!

Enough drivel. Time for a little hard news...

1) We are no longer carrying Applied Ingenuity hard drives. They are still a fine product and a good buy at the current (higher) price, but AI itself is the best place to get the drives, both in terms of order fulfillment speed and price.

2) DesignMaster is now sold exclusively by The ByteWorks (4700 Irving Blvd. NW Suite 207, Albuquerque, NM 87114, (505) 898-8183). Before you choke on the price, the program is greatly expanded, supports System 5.0, and is still the best productivity buy for yer bucks. Note that it won't be shipping for a few weeks yet. Considering that the programmer used to sell it for $30, you may be wondering why the jump in price. Chris Haun is probably making *less* on each unit now than he was - but he doesn't have to fill orders and do bookkeeping (onerous chores, let me tell ya).

On top of that, two other parties need to cover their costs and make a profit, the publisher (who now pays for manuals, etc.) and the distributors (Egghead Software, Roger Coats, Programs Plus and the like).

The sorry lesson herein is that anybody selling a product too cheaply is not building in enough margin to make it worth while for someone else to distribute!

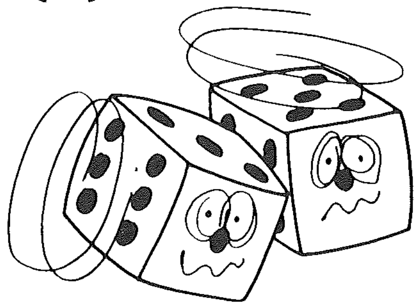
3) The Merlin translation of APP.BUILDER (the 8 bit macro language for assembly language programmers) is on hold. Apple's Eric Soldan re-wrote major portions of it and asked me to wait until he was done.

In APP.BUILDER's stead he submitted a fine article on using/reading the paddle circuits in your Apple. Ol' Eric had some neat tricks up his sleeve for coercing two byte values out of the port.

One of the main themes I heard from *Apprentice* subscribers is that you would like to see "The Gentleman's GS" become a monthly feature. That would be hard to do since we're so limited by space. But I will endeavor to put one in at least every *other* month... starting *next* month.

Getting More Value(s) From Your Game Port

By Eric Soldan, Apple II DTS



Editor: Eric accomplishes something herein that I did not even believe was possible - coercing the game paddles to return two byte values. I can see why Apple snatched him up for DTS! The concepts he delivers are not difficult conceptually, but they are a might heavy for the casual reader, especially in the last half of the article. For that reason, get a pop/fruit juice/milk, kick the kids out for about 20 minutes, sit back, and read on...

You probably haven't given your game port a lot of thought lately. You know what it is capable of, right? Just plug a paddle, joystick, Koala Pad, or whatever in, and start reading the values from it. Nothing could be simpler. Do a PDL(n) from APPLESOFT, or a LDX #n, JSR PREAD in assembly language. Either way, you are returned a value from 0 to 255, and then you do the appropriate thing based on this number.

What if you want to use the value to address some pixel on the hi-res screen? The width of the hi-res screen is 280 pixels and the paddle read routine just returns one byte as a result. I had this problem when trying to interface a Koala Pad to an Artwork Editor I wrote. I didn't want to access the screen in just bit pairs (values 0-139). I wanted to access any pixel on the screen instead.

Quickly looking at the paddle read routine, my first thought was that it can't be improved. The code is very tight, of course, and it only involves one byte being incremented. Counting with two bytes would supposedly take even longer per count, and what we need to do is to count faster than the old routine, not slower. We want to be able to count at least 280 counts in the same amount or less time than the old routine took to do 256 counts.

Let's look at the original paddle read routine:

```
PREAD    LDA PTRIG         ;Start timer going.
         LDY #0
         NOP               ;Adjust timing for first count.
         NOP
PREAD2   LDA PADDL0,X      ;4 cycles.  See how the paddle port is doing.
         BPL DONE          ;2 cycles when branch fails.
         INY               ;2 cycles.  Paddle not done yet, so add 1 to count.
         BNE PREAD2        ;3 cycles when branch succeeds.
         DEY               ;Change the 0 to the maximum count of 255.
DONE     RTS
```

Again, this looks unbeatable in terms of speed. There are four instructions in the loop, none of which is many clock cycles. To do much better, we will have to get the number of instructions per count down to three. Three instructions per count is the minimum number possible. We need to do the following steps, but not necessarily in this order:

1: Load the paddle signal value.
2: Branch if we are done (or not done)
3: Count the loop.

After some work, I came up with the following:

```
         BIT PTRIG          ;Start timer going.
         LDA #0             ;Load ACC and Y with 0.
         TAY
         CLC
LOOP     ADC #1             ;3 cycles.  Add odd counts into ACC.
         LDX PADDL0         ;4 cycles.  Check paddle 0.
         BPL DONE           ;2 cycles when branch fails.
         INY                ;2 cycles.
         LDX PADDL0         ;4 cycles.
         BMI LOOP           ;3 cycles when branch succeeds.
DONE     STY YVAL
         ADC YVAL           ;Add odd and even counts together.
         LDY #0             ;Make Y register hi-byte of result.
         BCC RTS0
         INY
RTS0     RTS
```

The above loop counts odd counts in the accumulator and even counts in the Y register. Each count takes a total of 9 cycles, compared to 11 cycles for the original paddle read routine. The increase in speed will allow 312 counts in the same time that it took for 256 counts before. The only problem with the new read routine is that there is no way out of the loop if there is no device plugged into the game port. The original routine would have terminated anyway with a value of 255. I can't think of a way to write a routine faster than the original that handles counts greater than one byte that will terminate if there is no device plugged in. To prevent this possible infinite loop, a verify paddle port device routine can be called prior to calling this new routine to make sure that there is a device plugged in.

Well, using this technique, a device such as a Koala Pad can read X-coordinates from 0 to 279, thereby addressing the entire hi-res screen.

But wait! What about double hi-res? For double hi-res graphics, you need a read routine that returns a value from 0 to 559. This is possible with the Apple IIGS in fast mode. The load from C0xx memory doesn't slow down the processor for very long. The net effect is that it is possible to count fast enough to get values from 0 to 559. And with a faster processor speed (via TransWarp), even higher count rates will be possible. Of course, with the IIGS, it is possible to just put it in 16 bit mode and use the Y register alone for counts greater than 1 byte. Just using a 16 bit Y register does not make it faster than this 8 bit method. The only way to improve the speed on the IIGS is to take advantage of the directPage register and set it so that the soft switches for the paddle port is a one-byte address. Using a one-byte address saves a cycle. Of course, this trick can be done using either the 8 bit or 16 bit method.

The problem with counting beyond 512 counts in 8 bit is that both the accumulator and the Y register wrap. There would seem to be no way to detect this wrap. Fortunately, however, when the accumulator wraps, the carry becomes set for a count pair, and then the carry is added into the accumulator. So, considering the accumulator and the carry together, the accumulator period is 255, whereas the Y register period is 256. Every time the registers wrap, the accumulator gets one more ahead of the Y register. The accumulator and Y register are out of phase, so to speak. So, we can let the count loop wrap and then figure out how many times it wrapped when it finally finishes.

This technique is useful for any type of timing where fast counting is critical. It allows values from 1 to 65534 to be returned before it fails because of wrapping problems. Devices could be designed whose maximum resistance is greater than that of the standard devices for any range of input values needed. Having a faster counting method allows that maximum resistance to be lessened.

The following code demonstrates this technique. Just POKE the paddle number in byte location 6 and then call 768. The result is returned in bytes 6 and 7. You can try it very simply from APPLESOFT using the simple APPLESOFT example.

Have fun.

Applesoft Example

```
10  HOME
20  POKE 6,0:CALL 768:PRINT PEEK(6)+PEEK(7)*256;
30  POKE 6,1:CALL 768:PRINT ","PEEK(6)+PEEK(7)*256"         "
40  VTAB 1:GOTO 20
```

Two Byte Paddle Read Source

```
        ORG $300

PDLNUM  EQU $06
AV      EQU $06
PDLLO   EQU $06
YV      EQU $07
PDLHI   EQU $07
ODD     EQU $08
WRAP    EQU $08
PDL0    EQU $C064
PDL1    EQU $C065
PTRIG   EQU $C070


PDLREAD LDA PDLNUM      ;Get paddle number from APPLESOFT example
        CLC             ;and use it to modify the address of the LDX PDLn
        ADC #PDL0       ;instructions in the fast loop so it will be
        STA FASTLP0+1   ;testing the correct paddle.
        STA FASTLP1+1
        LDY #1          ;Prepare everything in advance for the fast loop.
        STY ODD         ;Will be decremented to 0 for even counts.
        DEY             ;Set accumulator and Y-register to 0
        TYA
        CLC             ;Carry starts cleared.
        PHP             ;Remember interrupt status.
        SEI             ;Disable interrupts.
        LDX PTRIG       ;Trigger the paddles.
```

```
FASTLOOP ADC #1            ;3 cycles
FASTLP0  LDX PDL0          ;4 cycles
         BPL PDLADD        ;2 cycles for no branch
         INY               ;2 cycles
FASTLP1  LDX PDL0          ;4 cycles
         BMI FASTLOOP      ;3 cycles for branch
         DEC ODD           ;ODD=1 if exited from ADC half.
                           ;ODD=0 if exited from INY half.

PDLADD  PLP               ;Restore interrupt status to original status.

* The table below indicates the entering values, and the process of generating
* the actual count value from the beginning data.  Since the carry would have
* been added into the accumulator next odd count, the carry and the accumulator
* are added together first in the code.
*   C=Carry value    A=Accumulator value    Y=Y register value
*
*    C:  ..   0    0    1    1    0    0  ..   0    0    1    1    0    0  ..
*    A:  .. 255  255    0    0    2    2  .. 255  255    0    0    2    2  ..
*    Y:  .. 254  255  255    0    0    1  .. 253  254  254  255  255    0  ..
*
-------------------------------------------------------------------------------
* A+C=   .. 255  255    1    1    2    2  .. 255  255    1    1    2    2  ..
* A+C+Y= .. 509  510  256    1    2    3  .. 508  509  255  256    1    2  ..
* A+C-Y= ..   1    0    2    1    2    1  ..   2    1    3    2    3    2  ..
*
-------------------------------------------------------------------------------
*   ODD= ..   1    0    1    0    1    0  ..   1    0    1    0    1    0  ..
*  WRAP= ..   0    0    1    1    1    1  ..   1    1    2    2    2    2  ..
*    (WRAP = A+C-Y-ODD)
*
-------------------------------------------------------------------------------
*ACTUAL= ..  509  510  511  512  513  514  .. 1019 1020 1021 1022 1023 1024  ..
*ACTUAL= A+C+Y+511*WRAP (-256 if Y>127 and A<128)

         ADC #0            ;Add carry value.
         STA AV            ;AV = A+C
         STY YV
         SEC
         SBC YV            ;ACC = A+C - Y
         SEC
         SBC ODD           ;ACC = A+C - Y - ODD
         STA WRAP
         LDA AV
         CLC
         ADC YV            ;ACC = A+C + Y
         PHA               ;Low byte so far.
         LDA WRAP          ;Add 2*WRAP (512*WRAP) + C to start hi-byte.
         ADC WRAP          ;1 will be subtracted later to make 511*WRAP.
         PHA               ;High byte so far.
         LDA YV
         EOR #$80          ;Bit 7 on if Y<128.
         ORA AV            ;NOT(Boolean(Y>127 and A<128))
```

```
        ASL                 ;Place logical in carry.
        PLA                 ;High byte so far.
        SBC #0              ;Carry CLEAR when Y>127 and A<128.
        TAY                 ;High byte so far.
        PLA                 ;Low byte so far.
        SEC                 ;512*WRAP was added before, so subtract WRAP
        SBC WRAP            ;to make it +511*WRAP.
        BCS PADD0
        DEY
PADD0   STA PDLLO           ;OPTIONAL store result.
        STY PDLHI           ;Count value range is 1 to 65534.
        RTS
```

Super Slick Stuff From Synesis Systems

More Goodies From Steve

Dear Ross,

Here's another tidbit (or three) you might wish to share with the GS programmers in the group to get more power out of Merlin 16. I was inspired by your July issue to show another way to do direct page addressing and some macros and a pseudo op that I find useful.

First of all, a couple of little macros that I think really ought to be built-in instructions: TKB and TSD. I use these little guys constantly; they make sense, they're obvious, and they help make source code readable (William D. Mensch, are you listening?).

Now, how about a good use for the pseudo opcodes DUM and DEND? Many times in a program you need to access a variable. And most of us tend to group related variables in a stash area. But when that stash is hot inside of your own code space (like in a direct page or a Memory Manager acquired memory block), it seems difficult to come up with an easily modifiable method. The usual method is to equate the first variable and then define each of the others relative to the previous one. But you don't want to be around when you need to insert another in the middle of the list or change the amount of space one of the early ones uses; it ain't a pretty sight! On the other hand, using DUM and DEND makes it totally painless! See lines 87-91 in my example; the usual method is shown in the comments area.

I've found yet another valuable way of using DUM and DEND when it comes to direct page indirect long addressing (see lines 99-103) Warning! This method is not for the feint of heart! To see the traditional, conservative method, you may refer to page 8 of the July issue (*Editor: Steve must have faith in my objectivity. He is referring to your fair editor's piece on the List Manager. His comment is okay, though, because I am a traditional conservative in every sense of the term - I just didn't know it spilled over* *into my programming!)* In the course of writing a Desk Accessory (with no direct page), I found the need to create one's own dpage as required. The idea is not new (look through any Apple reference materials), but typically you see it done with hard coded addressing such as LDA [3],Y. This is very difficult to read, modify, or maintain. What I espouse is meaningful labels, wherever possible.

So, to put these concepts together into an example, I selected pages 8 and 9 f rom the July issue as the guinea pig (sorry Ross).

This first toolbox call (_NextMember) returns the address of the selected member. Since the result is returned on the stack, is there any need to pull it off and store it in your dpage just so you can dereference it? By leaving it on the stack (for awhile) and temporarily making the stack into the dpage, you can deref it in place. To make a 'mini' dpage, all you need is TSD. But you must also remember the stack pointer has already bumped to the NEXT location, so you may use TSC INC TCD and thereafter refer to stuff in the stack/dpage as zero-relative. OR, you can use DUM 1 and give the stuff in the stack some meaningful labels!

When you are through accessing the stuff in the stack, just reset D and the stack pointer.

There are several ways to reset the stack pointer but simply pulling it into an unused register is usually most efficient; I came up with a simple macro that does this for me (it assumes a 16-bit Acc, the mini dpage starts with DUM 1, and ends with the label ':dpage'). In this example, the 'fix:stk' macro (line 121) pops 4 words after restoring D. The macro decides how many words to pop (line 22) by looking up the value of ':dpage' (in this example, it equals $B) and subtracting the initial offset (1) and width of D(2) and dividing the resulting number of bytes (8) by 2 (resulting in 4). This value becomes the LUP value in the macro. Using this DUM/DEND method and the transparent 'fix:stk' macro it is very easy to modify the routine. For instance, I give myself another long variable in my mini dpage by simply pushing more space (lines 95-96) and naming it (line 103). Note in lines 119) that you may do all sorts of other stuff with this mini dpage in place; however, be careful not to branch out of this routine without exiting though 'fix:stk' (or equivalent).

You may go back over your code and find (as I did) many places that you pulled a handle off the stack from a tool call, stored it in an absolute location within your program, copied it to you dpage just to deref it, and never used it again! What a waste of program space and dpage, not to mention all of the extra steps in copying the value.

Sincerely,

Steve Stephenson

P.S. With an Apple IIGS, Merlin 16+, and an understanding wife, life doesn't get much better!

Steve's Sample Code

```
1
2              xc
3              xc
4              mx      %00
```

```
                           rel
                           tr
                           tr    adr
                  *--------------------------------------------------
                  * some of my useful macros
                           do    0
                  tkb      mac                ;Transfer K to B
                           phk                ; reset B = K using stack
                           plb
                           <<<
                  tsd      mac                ;Transfer S to D
                           tsc                ; reset D = S using Acc (C)
                           tcd
                           <<<
                  fix:stk  mac                ;undo mini dpage
                  * requires use of 'dum 1' & ':dpage dend'
                           pld                ; restore d
                           lup   #:dpage-3/2 ;calc number of words to pop
                           pla                ;(just pop and throw away)
                           <<<
                           fin
                  *--------------------------------------------------
                  * some constants for this example
                           dum   0
0000: 00 00 00 00 itemPtr  adrl  0            ;(itemPtr equ 0)
0004: 00          selected dfb   0            ;(selected equ itemPtr+4)
0005: 00          itemNum  dfb   0            ;(itemNum equ selecetd+1)
                           dend
                  *--------------------------------------------------
                  Example
8000:                      ~NextMember #0;#ListRecord
                           PHS   2
                           DO    1
8000: 48                   PHA
8001: 48                   PHA
                           ELSE
                           PHA
                           FIN
                           <<<
                           PxL   #0;#ListRecord
                           DO    2/1
                           PHL   #0
                           IF    #=#0
8002: F4 00 00             PEA   #^0
                           ELSE
                           PHW   ]1+2
                           FIN
                           PHW   #0
                           IF    #=#0
8005: F4 00 00             PEA   #0
                           ELSE
                           IF    MX/2
                           LDA   ]1+1
                           PHA
```

```
                    94         FIN
                    94         LDA    ]1
                    94         PHA
                    94         FIN
                    94         <<<
                    94         <<<
                    94         DO     2/2
                    94         PHL    #ListRecord
                    94         IF     #=#ListRecord
8008: F4 00 00      94         PEA    #^ListRecord
                    94         ELSE
                    94         PHW    ]1+2
                    94         FIN
                    94         PHW    #ListRecord
                    94         IF     #=#ListRecord
800B: F4 4B 80      94         PEA    #ListRecord
                    94         ELSE
                    94         IF     MX/2
                    94         LDA    ]1+1
                    94         PHA
                    94         FIN
                    94         LDA    ]1
                    94         PHA
                    94         FIN
                    94         <<<
                    94         <<<
                    94         DO     2/3
                    94         PHL    ]3
                    94         DO     ]0/4
                    94         PHL    ]4
                    94         FIN
                    94         FIN
                    94         FIN
                    94         FIN
                    94         <<<
                    94         Tool   $0B1C
800E: A2 1C 0B      94         LDX    #$0B1C
8011: 22 00 00 E1   94         JSL    $E10000
                    94         <<<
                    94         <<<
8015: 48            95         pha              ;push long space
8016: 48            96         pha              ; for deref use
8017: 0B            97         phd              ;save current d
8018:               98         tsd              ;reset dpage
8018: 3B            98         tsc              ; reset D = S using Acc (C)
8019: 5B            98         tcd
                    98         <<<
                    99         dum    1         ;(offset for stk ptr)
0001: 00 00         100 :d       dw     0         ;the saved d
0003: 00 00 00 00   101 :entry   adrl   0         ;the extra space
0007: 00 00 00 00   102 :listptr adrl   0         ;the selected member ptr
                    103 :dpage   dend
```

```

801A: A7 07          lda   [:listptr] ;deref the list ptr
801C: 85 03          sta   :entry     ; to get the entry's ptr
801E: A0 02 00       ldy   #2
8021: B7 07          lda   [:listptr],y
8023: 85 05          sta   :entry+2

8025: A0 05 00       ldy   #itemNum   ;offset to item number
8028: B7 07          lda   [:listptr],y ; get the number
802A: 8D 49 80       sta   ItemSelected

802D: A0 06 00       ldy   #6
8030: B7 03          lda   [:entry],y ;get 6th char in the entry
8032: 8D 47 80       sta   keep       ; (for something to do)

8035:                ~DrawString :entry ;display the string

                     PHL   :entry
                     IF    #=:entry
                     PEA   ^]1
                     ELSE
                     PHW   :entry+2
                     IF    #=:entry+2
                     PEA   ]1
                     ELSE
                     IF    MX/2
                     LDA   ]1+1
                     PHA
                     FIN
8035: A5 05          LDA   :entry+2
8037: 48             PHA
                     FIN
                     <<<
                     FIN
                     PHW   :entry
                     IF    #=:entry
                     PEA   ]1
                     ELSE
                     IF    MX/2
                     LDA   ]1+1
                     PHA
                     FIN
8038: A5 03          LDA   :entry
803A: 48             PHA
                     FIN
                     <<<
                     <<<
                     Tool  $A504
803B: A2 04 A5       LDX   #$A504
803E: 22 00 00 E1    JSL   $E10000
                     <<<
```

```
                           <<<
               :done
8042:                      fix:stk          ;undo the mini dpage
               * requires use of 'dum 1' & ':dpage dend'
8042: 2B                   pld              ; restore d
8043: 68                   pla              ;(just pop and throw away)
8044: 68                   pla              ;(just pop and throw away)
8045: 68                   pla              ;(just pop and throw away)
8046: 68                   pla              ;(just pop and throw away)
                           <<<
```

The Return of the Source Code Monster

Generic Start II: The Sequel

By Jay Jennings, A2-Central

Editorial Preface: One of the most enjoyable things about being an editor is that I ***always*** *get the last word. Just watch... but don't forget to pay attention to Jay 'cuz the new startup and shutdown calls for System 5.0 are mucho easier.*

If you gaze back through the mists of time (or dig out your back issues) you'll see that one of the first programs published in the Sourceror's Apprentice was GENERIC STARTUP. That program was designed to make programming the IIgs much easier by leaving the worry of loading and starting tools to someone else. That "someone else" consisted of Ross Lambert, Eric Mueller, and myself. Even after all the shouting and discussions ended, we remained friends (although we all moved to different states). We felt the GENERIC STARTUP routine was needed because there was so much confusion about which tools had to be started and in what order. *(Editor: Fortunately, after having a good laugh or two, Apple II DTS issued Apple IIGS Tech Note #12 which straightened everyone out. At least until things changed again...)*

It's time to update the GENERIC STARTUP routine. But this time I'm going it alone. *(Editor: Because no one else would work with him?)* The reason for a new routine is to take advantage of some of the new features provided for us in System Disk 5.0 for the IIgs. The new startup routine takes advantage of two new tool calls, STARTUPTOOLS and SHUTDOWNTOOLS.

The StartUpTools call loads and starts all the tools that we need. It knows which ones to use because we pass it a pointer to a table of tools and version numbers. The call looks in ROM for the tool and if it can't find it (or the version number isn't high enough), it pulls the tool from the System Disk. One of the nice things about this tool call is that we no longer have to allocate direct page space for our tools. The call takes care of that for us.

The first thing we'll look at is the format of the StartStop record.

First is a flag word that must be set to zero. The System Disk 5.0 pre-release docs don't say what this flag is for and I have no idea, so just trust me. Set it to zero. *(Editor: Would you trust a guy who makes Joe Isuzu look like George Washington>?)*

Next is a word the specifies what video mode we want QD II to start up in; 320x200 or 640x200. This also set some parameters for the Event Manager (like clamping values).

Third and fourth in the list are a couple empty places that will be filled in when you actually make the StartUpTools call. Don't worry about them as the ShutDownTools call will use those.

The fifth parameter in the table is the total number of tools that you want to start up.

Finally, we have a list of tools to start. We also specify the minimum version number that we need to work with.

Listing 1 - New System 5.0 StartUp Procedure

```
StartStopRec
     dw   0      ;Flag word - must be set to zero
     dw   $80    ;video mode for QDII - 0=320 / $80=640
     ds   2      ;resFileID - used by ShutDownTools
     ds   4      ;dPageHandle - used by ShutDownTools
     dw   12     ;number of tools to start up

     dw   3,$0300       ;Misc Tools
     dw   4,$0300       ;QuickDraw II
     dw   5,0           ;Desk Manager
     dw   6,$0300       ;Event Manager
     dw   14,$0300      ;Window Manager
     dw   16,$0300      ;Control Manager
     dw   15,$0300      ;Menu Manager
     dw   18,$0206      ;QD Aux
     dw   20,0          ;LineEdit
     dw   21,0          ;Dialog Manager
     dw   22,$0104      ;Scrap Manager
     dw   23,0          ;Standard File Tool set
     dw   27,$0204      ;Font Manager
     dw   28,0          ;List Manager
     dw   34,0          ;TextEdit Tool Set
```

Notice that there are two very important tool sets that aren't on the list. The first is the Tool Locator and the second is the Memory Manager. We still have to start those manually because the new StartUpTool call needs the program's UserID to work. And we need the Tool Locator to start the Memory Manager. This makes the startup routine very simple.

Listing 1b - What Should Have Come Before Listing 1

```
StartUp    ent
           _TLStartUp          ;start the Tool Locator
                               ;check for a tool error here
           pha                 ;result space
           _MMStartUp          ;start the Memory Manager
                               ;check for tool error here
           PullWord  ProgID    ;get our user ID

           PushLong  #0        ;result space

           PushWord  ProgID    ;push our program ID
           pea       #0        ;define the next reference
           PushLong  #StartStopRec  ;address of tool table
           ldx       #$1801    ;_StartUpTools number
           jsl       $E10000
                               ;check for a tool error here
           PullLong  SSRec     ;reference to StartStop record

           rts

SSRec      ds        4         ;space for StartStop reference
```

Attention
puisque QDA ne
est dans la liste
il faut.
remettre
le curseur
normal par
_InitCursor.

That's all there is to it! This one tool call allocates direct page space for the tools, loads, and starts them. Because of the magic of System Disk 5.0 we trade pages of code for just a few lines.

Before we move on to the shut down routine, there's one line above that may need a little more explanation. The line PEA #0 defines the type of reference that we use in the next line (PushLong #StartStopRec) of the program. By pushing a zero we're stating the reference is a pointer (we pushed the address of the table next). If we used a one as a reference that means the next line would be referring to a handle. And by using a two we state the reference is a resource ID. Thus we can put our tool table in the resource fork of our program if we so desire (and can find the info to do that).

Shutting down the tools is even easier than starting them. It takes just a few lines of code.

Listing 2 - System 5.0 Shutdown Procedure

maintenant
PushLong #SSRec

```
ShutDown   ent
           pea       #0        ;define the next reference
           ~~PushLong  #StartStopRec~~  ;address of tool table
           ldx       $#1901    ;_ShutDownTools number
           jsl       $E10000
```

```
          PushWord  ProgID
          _MMShutDown          ;shut down the Memory Manager
          _TLShutDown          ;...and the Tool Locator
     {Quit code goes here}
```

Our first line of code defines the StartStop record reference just like in the startup code. A zero means we'll use a pointer, a one means a handle, and a two means look for the tool table in the resource fork.

You can get more information on this (and all the new tool calls) by getting the IIgs Toolbox

Reference Volume 3 from APDA. I believe they're still selling the beta docs but that's much better than having no docs at all.

Ask Mike Rochip

Dear Mike,

Jeff at Roger Wagner Publishing thought I should write to you. We've spent the last hour figuring out how to get Merlin 16+ to print slashed zeros on an Imagewriter. Perhpas there are other readers who might like to do this.

In the Parms file on line 252 there is a define string of 15 bytes (DS 15) which is the printer init string. This string needs to be changed. One way is to tuse the Ctl-O command. This allows you to type in the code characters directly. Here's how:

1) Change the Define String command (DS) to ASC and then delete the 15.

2) The type a quotation mark (", indicating high bit set); then type CTRL-O (your cursor will disappear) and push the escape key (your cursor will return).

3) Now push the D (it is uppercase) key.

4) Press CTRL-O (again your cursor will disappear) and push CTRL-@ (that's CTRL, SHIFT, @; and your cursor will reappear).

5) Press CTRL-O (again your cursor will disappear) and push CTRL-A (it will reappear).

6) Type CTRL-O (cursor gone) and push the escape key (cursor back).

7) Next push the Z (uppercase) key.

8) Press CTRL-O (cursor gone) and push CTRL-@ (CTRL, SHIFT, @; cursor will reappear).

9) Press CTRL-O (again your cursor will disappear) and push CTRL-@ (CTRL, SHIFT, @; cursor reappears) .

10) Type closing quotation mark. (")

11) Then type in a comma, and 00 (that's two zeros seven times. This fills out the string to 15 bytes.

That's it. Save the modified parms file and do an OA-A on the source file to assemble it. It will automatically replace the binary file in the main directory. You can then reboot Merlin to activate the new parms.

Now I have another question. I would like to order back issues Vol 1 No 1 through Vol 1 No 3. How do I order these back issues and how much do they cost?...

Thanks for your time and help,

Sincerely,

Gerald D. Schultz II
Playa Del Rey, CA 90392

Gerald -

*You've provided all the help - thank **you.***

Back issues are now $3.00 each, which includes postage for USA subscribers. Non-USA subscribers please add $1.50 US per back issue (we've gotten stung on foreign postage, folks - sorry to raise the price for non-USA people, but we really have to watch our checkbook around here.)

Dear Mike,

I am the Kansas math teacher who called the other night...

Specifically, my needs are as follows:

1) How to write floating point routines to do the basic math functions, addition, subtraction, multiplication, division, logs, trig functions, exponential functions, etc.

2) Are there algorithms for exact (digit by digit) results of the above functions? If you know of a source please tell me.

T.L. Warkentin
Lakin, KS

Dear T.L.,

I apologize for taking so long to keep my promise to write back to you. I had to dig through a bunch of boxes (we've just moved and will be moving yet again!) so it took much longer than I thought.

The book you need is called "6502 Assembly Language Subroutines" by Leventhal and Saville. The good folks at A2-Central carry it. (913/469-6502 or write P.O. Box 11250, OverLand Park, Kansas 66207). It contains numerous routines for all kinds of things, including floating point arithmetic.

Your question is more subtle than it might appear on the surface - floating point math is not very easy in assembly language. Many, if not most programmers who use the Apple II make use of the Applesoft floating point routines in ROM. I strongly recommend this for most purposes. Another book available from A2-Central, "Assembly Language for the Applesoft Programmer" by Finley and Myers, provides a ton of worthwhile information, including an entire chapter on floating point arithmetic. That chapter (#8) is entitled "Using Applesoft Floating Point Subroutines". I think that it is just what the doctor ordered for your situation.

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, Jerry Kindall ...Contributing Editors
Rebecca Lambert... Stamp licking

Back issues are availabe for $3.00 each. Non-North American subscribers add $1.50 each.

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.