The Assembly Advantage

by Randall Hyde

Trying Out the Tools—Some Basics

Once the beginner masters his Apple computer and develops a strong command of the Basic language, thoughts immediately turn to optimization. There's probably not a single program written that couldn't benefit from extra speed or additional features. So the beginner learns Basic tricks, like removing Rems, moving subroutines to the beginning of the program, and declaring often-used variables early in the program.

Ultimately, however, jerry-rigging fails to achieve the needed improvement and the programmer is forced to contemplate use of a faster language. While faster high-level languages, like Pascal and Forth, are available for the Apple II, these languages can't come close to 6502 assembly language. For the programmer writing time-critical code (like a hi-res arcade game) using 6502 machine code is an absolute necessity.

However, 6502 assembly language, the gateway to 6502 machine code, is difficult to learn. To ease the burden I have created a set of subroutines, collectively called Speed/Asm, to help minimize the effort. This series of articles will describe how you can easily create your own machine language programs using the Speed/Asm package.

Speed/Asm, like its companion, the LISA interactive assembler, is especially designed for the beginner at 6502 assembly language programming. I developed these packages as tools for use in the assembly language classes I teach in southern California. I've discovered that students who learn 6502 assembly language using the Speed/Asm package achieve competence much faster than those who learn using traditional methods of instruction.

Speed/Asm is a collection of subroutines you call from your program to perform certain tasks. In particular, the Speed/Asm subroutines emulate many of the statements found in Basic and Pascal. For example, the Basic program

```
10 FOR I = 1 TO 100
20 PRINT I;
30 NEXT I
```

is translated into Speed/Asm as

```
JSR FOR
ADR I,1,100
JSR PRTINT
ADR I
JSR NEXT
```

Persons using a macro assembler, like LISA version 3.0, could even code this as

```
.FOR I,1,100
.PRTINT I
.NEXT
```

As you can see, Speed/Asm looks quite a bit like Basic. But keep in mind that Speed/Asm *is* assembly language so you get a considerable performance boost compared to Basic.

Getting Started

Probably the best place to start is with the disk that comes in the Speed/Asm package. A quick catalog of the disk reveals that about 11 programs are included.

The file Speed/Asm is the boot program that does the cataloging. RELSA and RELFP are two programs used to generate a Speed/Asm program. Speed/Asm is relocatable to any page boundary; RELSA and RELFP are the programs that create an absolute version of the Speed/Asm program for actual use. RELFP generates a full Speed/Asm package, including the floating point operations. RELSA generates a copy of Speed/Asm without the floating point subroutines, hence it is much shorter.

SAFP.78, the next program on the disk, is a copy of Speed/Asm that has been located to address $7800. This file is provided for the convenience of those who want to write programs as quickly as possible without having to learn how to use the RELSA or RELFP program first. I will use SAFP.78 in all the examples I present, although only one line in your program will need to be changed if you wish to use Speed/Asm located at some other address.

The SA.EQUATES file contains equates for all the Speed/Asm subroutines. The file on this disk is provided in a LISA 2.5 compatible format (which can also be read in by LISA version 3.0). For those who are using an assembler other than the LISA interactive assembler, the SA.EQUATES file is reproduced in the program listing.

Incidentally, if you're a beginner and in the process of choosing an assembler, I would highly recommend a look at LISA. It is an interactive assembler that makes learning assembly language much easier. Unlike other assemblers, LISA catches errors on input, much like Basic.

I'm going to make the assumption, in this column, that you're using the LISA assembler. Attempting to describe every LISA feature to users of other assemblers would be too great a distraction. Attempting to write the code in a general fashion would make the programs overly large and confusing.

The remaining six programs provided on the Speed/Asm disk are test

Address correspondence to Randall Hyde, Lazer Microsystems, 1791 Capital, Corona, CA 91720.

programs that were used to help validate the Speed/Asm package. You may want to look at these files to get an idea of how Speed/Asm programs are written.

Preparing for Your First Program

Before we can jump in and run our first program there are several decisions that must be made. To begin with, we must decide where the program will run. Unlike Basic, which uses the same data in memory for the source code and run-time code, assembly language programs must be converted from a *source* (human-readable) format to an *object* code (computer-readable) format. This conversion is accomplished by an assembler like LISA. Once the source file is converted to object code you can run the program by executing the object code.

Most assemblers on the market (including LISA) operate in a *co-resident* mode. This means that the source text file and object code both reside in memory during the assembly of the program. Since both files are maintained in RAM at the same time, care must be taken to ensure that the source and run-time object code do not disturb one another. In Basic, memory was allocated automatically for you. While using 6502 assembly language, the memory management chore is left up to the programmer. So extra care must be taken when creating programs.

By referring to Figure 1 you can get an idea of what the Apple's memory space looks like while using LISA. In particular, locations $800 through $17FF are reserved for holding your object code, and locations $1800 and up are reserved for the source file. Four kilobytes of RAM should prove to be sufficient for most programs. Advanced programmers who require more RAM should consult the LISA documentation to find out how to adjust the default textfile/object code size settings.

Whenever you assemble a program, LISA automatically begins assembling and storing the code at location $800. Unless you want to assemble the code at some other location, no special action is necessary to assemble your code into the proper space in RAM.

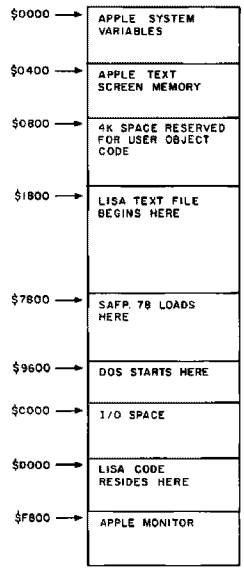


Figure 1. Apple memory map.

Writing Your First Assembly Language Program

The first program we'll write will do nothing at all! Actually it does do something: it immediately returns control to the Apple monitor. This program, as simple as it is, is important because it demonstrates how to terminate a 6502 assembly language program.

Consult the documentation that came with your assembler and learn how to use the editor to enter text into the source file buffer. Once you've mastered the editor, enter the following program into the text buffer:

```
EXIT  EQU  $FF59
      JMP  EXIT
      END
```

Once you've entered this program, assemble it using the assembler (the documentation should describe how to assemble a file). With LISA this is accomplished by typing ASM followed by a carriage return. Once the assembler has completed the task of assembling the code, you should access the Apple monitor program (LISA users type BREAK) and issue the Apple moni-

Listing. Speed/Asm Equates.

```
0800              1          TTL "Listing One: SPEED/ASM Equates"
0800              2  ;
0800              3  ; GENERAL PURPOSE EQUATES
0800              4  ;
0000              5  FORASAV  EPZ 0
0001              6  FORXSAV  EPZ FORASAV+1
0002              7  FORYSAV  EPZ FORXSAV+1
0003              8  FORZPG   EPZ FORYSAV+1
0005              9  DESTADR  EPZ FORZPG+2
0007             10  PTRADR   EPZ DESTADR+2
0009             11  ISIMMED  EPZ PTRADR+2
000A             12  OP       EPZ ISIMMED+1
000C             13  MAXLEN   EPZ OP+2
000D             14  VALUE    EPZ MAXLEN+1
000F             15  DIGIT    EPZ VALUE+2
0010             16  LEAD0    EPZ DIGIT+1
0011             17  JMPADR   EPZ LEAD0+1
0013             18  COUNT    EPZ JMPADR+2
0014             19  GOTLN    EPZ COUNT+1
0015             20  LINEINDX EPZ GOTLN+1
0016             21  SIGN     EPZ LINEINDX+1
0017             22  ACL      EPZ SIGN+1
0018             23  ACH      EPZ ACL+1
0019             24  XTNDL    EPZ ACH+1
001A             25  XTNDH    EPZ XTNDL+1
001B             26  AUXL     EPZ XTNDH+1
001C             27  AUXH     EPZ AUXL+1
0800             28  ;
0033             29  PROMPT   EPZ $33
004E             30  RNDL     EPZ $4E
004F             31  RNDH     EPZ $4F
0100             32  STACK    EQU $100
0200             33  INPUT    EQU $200
0800             34  ;
```

Listing continued.

Listing continued.

```
0000            35  FALSE    EQU 0
0001            36  TRUE     EQU 1
008D            37  CR       EQU $8D
0800            38  ;
0800            39  ;
0800            40  ; "IF" STATEMENT EQUATES
0800            41  ;
00ED            42  EQ       EQU "="
00A3            43  NE       EQU "#"
00BE            44  GT       EQU ">"
00BC            45  LT       EQU "<"
EDBE            46  GE       EQU ">"|"="*256
EDBC            47  LE       EQU "<"|"="*256
0800            48  ;
0800            49  ;
0800            50  ;
0800            51  ;
0800            52  ; SPEED/ASM ENTRY POINTS
0800            53  ;
0800            54  ;
0800            55  ;
0800            56  ; NOTE: THE EQUATE OF PUTC MUST
0800            57  ; BE CHANGED IF YOU RELOCATE
0800            58  ; SPEED/ASM TO SOME LOCATION
0800            59  ; OTHER THAN $7800
0800            60  ;
7800            61  PUTC     EQU $7800
7803            62  GETC     EQU PUTC+3
7806            63  SAGL     EQU GETC+3    ;FOR USE BY S/A ONLY- SEE DOC.
7809            64  SAPC     EQU SAGL+3    ; "   "  "  "  "    "  "
780C            65  HOME     EQU SAPC+3    ;HOME AND CLEAR
780F            66  READLN   EQU HOME+3
7812            67  INIT     EQU READLN+3
7815            68  FOR      EQU INIT+3
7818            69  FOR0     EQU FOR+3
781B            70  NEXT     EQU FOR0+3
781E            71  IFI      EQU NEXT+3
7821            72  IFI0     EQU IFI+3
7824            73  IFS      EQU IFI0+3
7827            74  IFS0     EQU IFS+3
782A            75  MOVE     EQU IFS0+3
782D            76  LOAD     EQU MOVE+3
7830            77  MOVS     EQU LOAD+3
7833            78  LDSTR    EQU MOVS+3
7836            79  PRINT    EQU LDSTR+3
7839            80  PRTSTR   EQU PRINT+3
783C            81  PRTINT   EQU PRTSTR+3
783F            82  RDSTR    EQU PRTINT+3
7842            83  RDINT    EQU RDSTR+3
7845            84  ONXGOTO  EQU RDINT+3
7848            85  CASE     EQU ONXGOTO+3
784B            86  CASEI    EQU CASE+3
784E            87  INSET    EQU CASEI+3
7851            88  NOTINSET EQU INSET+3
7854            89  ABS      EQU NOTINSET+3
7857            90  NEG      EQU ABS+3
785A            91  MUL      EQU NEG+3
785D            92  DIV      EQU MUL+3
7860            93  MOD      EQU DIV+3
7863            94  RND      EQU MOD+3
7866            95  SUBSTR   EQU RND+3
7869            96  INDEX    EQU SUBSTR+3
786C            97  LENGTH   EQU INDEX+3
786F            98  CONCAT   EQU LENGTH+3
7872            99  GETWZPG  EQU CONCAT+3  ;USED BY SPEED/ASM
7875           100  RDFP     EQU GETWZPG+3
7878           101  PRTE     EQU RDFP+3
787B           102  PRTF     EQU PRTE+3
787E           103  FADD     EQU PRTF+3
7881           104  FSUB     EQU FADD+3
7884           105  FMUL     EQU FSUB+3
7887           106  FDIV     EQU FMUL+3
788A           107  FLT      EQU FDIV+3
788D           108  FIX      EQU FLT+3
7890           109  FNEG     EQU FIX+3
7893           110  FADDTN   EQU FNEG+3
7896           111  FSUBTN   EQU FADDTN+3
7899           112  FTIMES   EQU FSUBTN+3
789C           113  FINTO    EQU FTIMES+3
789F           114  IFF      EQU FINTO+3
78A2           115  MOVFP    EQU IFF+3
0800           116  ;
0800           117          END
```

tor command 800G. The computer will beep and return you to the monitor mode. Congratulations! You've just run your first 6502 assembly language program.

A discussion of exactly what happened may help clear up any problems you have understanding the operation of this program. First of all, the statement

```
EXIT EQU $FF59
```

isn't a true 6502 assembly language statement at all. The EQU instruction is an example of a *pseudo opcode*, or *assembler directive*, provided by most assemblers. This instruction tells the assembler to replace every occurrence of EXIT with the value $FF59 during the assembly of the program. So, the next instruction (JMP EXIT) is converted to JMP $FF59.

The JMP (jump) instruction on the second line performs the same operation as a Goto in Basic. The JMP EXIT instruction directs the 6502 to jump to address $FF59 and begin executing code there. The entry point for the Apple monitor program just happens to be at address $FF59, so jumping to this location reactivates the Apple monitor. Since the program is executed from the Apple monitor and immediately returns control to the monitor, it will appear that nothing has happened.

The END pseudo opcode is required at the end of all LISA source files. It marks the physical end of the program. The END statement does *not* terminate the execution of your program. It is simply a marker for the assembler so it knows when to stop assembling the file. To terminate the program jump to location $FF59.

Other Methods of Terminating Your Programs

In addition to jumping to location $FF59, there are three other ways to terminate an assembly language program. Providing you are not within some nested subroutine and you haven't pushed any data onto the stack (advanced stuff), you can return control to the Apple's monitor

Congratulations! You've just run your first 6502 assembly language program.

with the simple command

```
RTS
```

This is a return-from-subroutine instruction. It's quite similar to Return in Basic. This program can be rewritten as:

```
RTS
END
```

If you assemble and run this program, control will once again be returned to the Apple monitor. This time, however, the speaker won't beep at you.

Another method for program termination is the BRK instruction. This instruction (used in a manner identical to RTS) beeps the speaker, prints the contents of the 6502's registers, and then transfers control to the Apple monitor. BRK is used mainly for debugging purposes, but you can use it anytime it's convenient to get a printout of the 6502 registers.

The last method for program termination I'll mention is very similar to the first example. A jump to location $FF69 also transfers control to the Apple monitor. Basic users are already familiar with this entry point to the Apple monitor. It's the CALL-151 instruction you use to get into the monitor in the first place.

Writing Your First Speed/Asm Program

Now that you can stop your program, the next step in learning how to use Speed/Asm is to begin writing programs that use it. Load the SA.EQUATES textfile into LISA and insert the following code just before the END statement in line 121:

```
EXIT  EQU  $FF69
;
;
      JSR  INIT
      JMP  EXIT
```

Assemble the program as before. But before you run it (and while you're still in LISA) type control-D BLOAD SAFP.78. Now break to the monitor and run the program by typing 800G. Note that the program will ask if your Apple can display upper- and lowercase. Once you type Y or N, control returns to the Apple monitor. If you answer Y, then all text sent to the screen is sent completely unmodified. If you answer N, then all lowercase characters are converted to uppercase before being output to the screen. This allows you to take advantage of lowercase add-on equipment, like the Lazer Microsystems' Lower Case + Plus and Keyboard + Plus units, without having to worry about compatibility problems with Apples not so equipped.

The JSR INIT instruction is new. JSR (which stands for jump to subroutine) is quite similar to GOSUB in Basic. Control is transferred to the specified address, where some routine is executed. The program then resumes execution at the next instruction after the JSR, whenever an RTS instruction is executed. The INIT subroutine entry address is defined in the Speed/Asm equates. *Init must be called before running any Speed/Asm program.* Failure to do so will cause the package to malfunction.

Once INIT is through doing its thing (asking you about lowercase and initializing the system), control is returned to the Apple monitor by the execution of the JMP EXIT instruction.

At this point we've mastered all the mechanics of running a Speed/Asm program: begin by JSRing to INIT, and end with a jump to location $FF69. Your Speed/Asm pro-

gram fits in between these two instructions. Consider the following program:

```
EXIT  EQU  $FF69
;
;
      JSR  INIT
      JSR  PRINT
      BYT  "This is a test",CR,0
      JMP  EXIT
```

It prints

This is a test

followed by a carriage return onto the video screen. This is but a short example of how to write your own Speed/Asm program. I will describe what this program does when I talk about the Print subroutine.

Declaring Variables in Speed/Asm: A Short Course in Data Structures

Speed/Asm programs can use any of four different data types: individual character, character string, integer number, and real (floating point) number. Unlike Basic, but similar to Pascal, storage space for all variables used in a Speed/Asm program must be allocated somewhere in the program. In order to define a variable for use by Speed/Asm you must understand the underlying representations for the character, string, integer, and real data types.

The Apple's memory space consists of 65,535 memory cells called bytes. Each byte holds one character, or other sub-unit of data. String, integer, and floating point variables are created by combining several bytes. For example, integer variables reside in two contiguous bytes of RAM; real variables (in Speed/Asm) require eight contiguous bytes. String variables need n + 2 bytes where n is the maximum number of characters you want the string to hold. You must explicitly reserve sufficient space for each variable you plan to use.

Most assemblers provide several pseudo opcodes to be used to reserve blocks of memory for multi-byte data types. Applicable pseudo opcodes provided in the LISA assembler include BYT, HBY, HEX, ADR, DBY, STR, ASC, INV, BLK, .DA, DCI and DFS. Most of them differ in how they allow you to *initialize* the variables you are defining. For our purposes the BYT, ADR, STR and DFS pseudo opcodes will be used to declare variables.

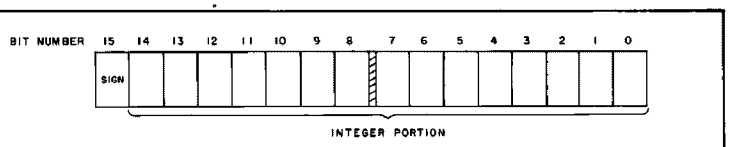


Figure 2. Two's complement representation.

BYT reserves one byte of storage for each operand present on the line after the BYT instruction. For example, the instruction

```
BYT 0,1,2,3,4
```

reserves five bytes of memory and *initializes* them to the values zero, one, two, three and four. Since a single byte can only represent numeric values in the range 0 to 255, any attempt to define an initial value outside this range will be futile.

If you attempt to define a value greater than 255 using the BYT pseudo opcode, LISA uses the low order (L.O.) byte of the value as the initial value for that location. For example, the statement

```
BYT EXIT
```

where EXIT is equated to $FF69 produces the same code as

```
BYT $69
```

The high order (H.O.) of EXIT's value is ignored by the BYT pseudo opcode.

To define a single-byte variable (possibly to hold a character value), place the variable's name in column 1 of the line containing the BYT pseudo opcode. If you want to define a single-byte variable called CHAR, do so using the statement

```
CHAR BYT 0
```

This statement defines CHAR, reserves space for it and initializes the variable to zero.

Another way to reserve space for a variable is with the DFS (define storage) pseudo opcode. Whereas BYT's operand specifies the initial value the space reserved occupies, DFS's operand specifies how many bytes are to be reserved. If you don't need to initialize CHAR to zero, an alternate method of defining it could be

```
CHAR DFS 1
```

This instruction tells LISA to reserve one byte of storage for the CHAR variable.

Integer variables in Speed/Asm are represented using the standard two-byte two's complement format (see Figure 2). Fifteen bits are used to hold the numeric value, and bit number 16 holds the sign. A two-byte integer variable can represent values in the range −32768 to +32767. Basic programmers should quickly recognize this range as the same supported by Basic integer variables. Basic uses this same format to store integer variables.

To reserve space for an integer variable in Speed/Asm you must reserve at least two bytes. One way to accomplish this is to follow the BYT pseudo opcode with two operands—that is,

```
INTGR BYT 0,0
```

But this method is inelegant because BYT should be used for declaring byte variables. The ADR pseudo opcode provides a much better alternative to the BYT pseudo opcode for declaring integer variables. The previous declaration could be rewritten as

```
INTGR ADR 0
```

Since *two* bytes are reserved for each operand, only one operand need follow the ADR pseudo opcode. Furthermore, if you follow the ADR pseudo opcode with a value that requires two bytes to hold (like EXIT, or some value greater than 255 or less than zero), ADR will properly store the two's complement repre-

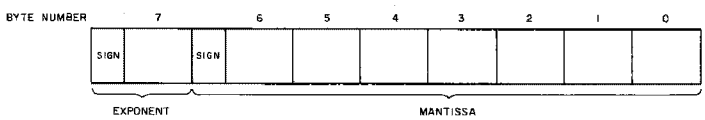
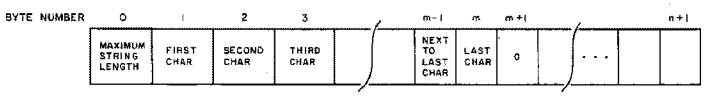


Figure 3. Floating point representation.



Notes:
1) Strings always begin with a byte containing the maximum allowable length of the string.
2) The maximum length is followed by m characters where m is the current dynamic length of the string.
3) All strings are terminated with a zero byte that marks the end of the string.
4) At least n + 2 bytes must be reserved for a string with maximum length n.

Figure 4. String representation.

sentation in memory for you.

Another way to reserve space for an integer variable is with the DFS statement. The instruction

```
INTGR DFS 2
```

reserves two bytes of storage for the integer variable INTGR.

Floating point variables in Speed/Asm require eight bytes. The only practical way to reserve space for a floating point variable is to use the DFS statement. Since eight bytes are required, the definition of a floating point variable takes the form

```
FLTPT DFS 8
```

Since LISA doesn't support a floating point pseudo opcode, you cannot initialize a floating point variable unless you know the hexadecimal representation for the value you're interested in. Figure 3 shows the floating point format used by the Speed/Asm package.

The last data type supported by the Speed/Asm package is the string data type. Strings differ from data of other types in that they vary in length. The amount of space you reserve for a string depends entirely on how big you wish to allow it to grow. A string in Speed/Asm supports two length attributes: a maximum string size and a dynamic (current length) size. Figure 4 shows the format of a string in Speed/Asm.

To declare a string you must reserve n + 2 bytes, where n is the maximum number of characters you wish to allow the string to hold. The first byte of the string must be initialized with the maximum length of the string. For an uninitialized string, the second byte should contain zero. The easiest way to define a string is to use the ADR and DFS pseudo opcodes in conjunction with one another as follows:

```
STRING  ADR  40
        DFS  40
```

The ADR pseudo opcode initializes the first byte of the string to 40 (the maximum length of the string) and the second byte of the string is initialized to zero. The DFS 40 instruction reserves the 40 bytes necessary to hold the string. Additional methods of reserving space for strings will be considered as the need arises.

Placement of Variables In Your Program

One last detail concerning the declaration of variables is the placement of variables within your Speed/Asm program.

The safest place is between the final JMP EXIT instruction and the END pseudo opcode. For technical reasons it is imperative that you do *not* place your variable declarations at the beginning or in the middle of the program.

In this article, the first installment in a series, I've discussed some features of the Speed/Asm package and how to reserve space for variables. Next time I'll describe how to use these variables and write some real Speed/Asm programs. ■

The Assembly Advantage

by Randy Hyde

Speedy Integers

Last time I discussed how variables are defined in a SPEED/ASM program. One byte must be reserved for a character variable, two bytes for an integer variable, eight for a floating point variable, and (n + 2) bytes must be reserved for a string variable. Now I'll describe how to perform integer arithmetic using the SPEED/ASM package.

Before describing how to deal with integer values in SPEED/ASM, I think a brief review concerning the declaration of integer variables is in order. As I mentioned last time, SPEED/ASM integers require two bytes of storage in RAM memory. While there are many ways to reserve two bytes of storage for an integer, I prefer to define an integer variable using the ADR pseudo-opcode thus:

<name> ADR 0

where <name> is the variable name with which I wish to reference the integer value. For example, to declare the integer variables I, J and K, I would use the statements:

```
I ADR 0
J ADR 0
K ADR 0
```

Integers on the 6502 consist of two bytes; the first eight bits (byte) comprise the low order byte (L.O. byte) and the second eight bits of the integer comprise the high order byte (H.O. byte). A single byte can hold any numeric value in the range 0. . . 255. Two bytes (taken as an unsigned integer) can be used to represent values in the range 0. . . 65535. The SPEED/ASM package uses a modified form of the binary numbering system called the *two's complement* numbering system. A pure binary numbering system cannot be used to represent negative values so the use of the binary number system is quite restrictive. The two's complement number system divides the unsigned range in half and uses half of the possible values to represent the numbers 0. . . 32767 and the other half of the available values to represent numbers in the range -32768. . . -1. Since SPEED/ASM uses the same two's complement format employed by Basic, SPEED/ASM's numeric range (-32768. . . 32767) is the same as Basic's.

Always remember that SPEED/ASM variables must be defined *outside* the range of your code. That is, during the execution of your program the 6502 must never jump to or fall through to a variable location. The 6502 would extract the data at that location and attempt to execute it as a valid 6502 instruction—usually with undesirable results. A well written SPEED/ASM program will have its variable declarations at the end of the program, after the JMP EXIT instruction (or whatever other method you use to terminate program execution—see Part One of this series). A good format for your SPEED/ASM programs is shown in Example 1.

Once you've defined an integer variable, the next step is to manipulate the data it holds. There are essentially ten integer operations available to the SPEED/ASM programmer: loading a variable with a value, copying the contents of one integer variable to another, the absolute val-

```
;
;
; (Put the SPEED/ASM equates here)
;
EXIT    EQU     $FF69
;
;
        JSR     INIT        ;Always call INIT first.
;
;
; (Your SPEED/ASM program goes here)
;
;
        JMP     EXIT        ;Used to terminate the program
;
;
; Variable declarations go here, eg:
;
I       ADR     0
J       ADR     0
;
; etc.
;
        END                 ;Required by LISA for end of program.
```

Example 1.

Address correspondence to Randy Hyde, Lazer MicroSystems, 1791 Capital, Corona, CA 91720.

```
CLC       ;Always before an addition
LDA I     ;Add L.O. byte of I to the
ADC J     ;L.O. byte of J and
STA K     ;store sum in L.O. byte of K.
LDA I+1   ;Add H.O. byte of I to the
ADC J+1   ;H.O. byte of J and store the
STA K+1   ;sum in the H.O. byte of K.
```

Example 2.

```
SEC       ;Carry must be set before a subtraction
LDA I     ;Subtract the L.O. byte of J from
SBC J     ;I and store the difference
STA K     ;into the L.O. byte of K.
LDA I+1   ;Subtract the H.O. byte of J from
SBC J+1   ;the H.O. byte of I and store the
STA K+1   ;difference into the H.O. byte of K.
```

Example 3.

ue function (ABS), negation, addition, subtraction, multiplication, division, modulo (remainder) and the random number function. Beyond these computational capabilities, the ability to input and output integers is also desirable.

Addition and subtraction are handled so easily in 6502 machine code that SPEED/ASM doesn't include addition and subtraction routines. If you wanted to add I and J and store the sum in K you would use the code in Example 2.

The CLC (clear carry) instruction *absolutely must* precede the addition sequence. Failure to clear the carry flag before performing the addition operation may result in an intermittent bug in your program. This addition sequence is almost identical to the Basic statement:

```
K=I+J
```

To perform a subtraction in 6502 assembly language (or SPEED/ASM), use the sequence in Example 3. Note that the carry flag must be *set* (using the SEC instruction) before performing the subtract sequence. Failure to set the carry before performing a subtraction may yield unpredictable results. The subtraction sequence above is roughly equivalent to the Basic statement:

```
K=I-J
```

If you need to add a *constant* to an integer variable (instead of adding two integer variables together) the # and / operators can be used to specify constants in the 6502 operand field. The # is used to specify the L.O. byte of an integer constant and the / is used to specify the H.O. byte of an integer constant. If you wanted to add the constant 4369 to the integer variable I and leave the result in K you would use the code in Example 4.

```
CLC
LDA   I
ADC   #4369
STA   K
LDA   I+1
ADC   /4369
STA   K+
```

Example 4.

```
CLC
LDA   #!-5639
ADC   J
STA   K
LDA   /!-5639
ADC   J+1
STA   K+1
```

Example 5.

This would produce the desired results. If you wanted to use a negative constant, LISA v2.5 requires that you preface the negative value with an exclamation mark. LISA v3.0 imposes no such restriction (see Example 5).

Testing for Overflow and Underflow

As I mentioned, the 6502 addition and subtraction operations are only rough approximations of the listed Basic statements. The difference between the assembly-language and Basic statements is in the way Basic checks for overflow or underflow. In Basic, if you attempt to add 32000 to 32000 you will get a ">32767" error. In assembly language you will end up with the value −1536 in variable K, and no error will be reported. When I was learning addition in grade school I was taught that 32000 + 32000 equals 64000, not −1536.

If you can live with a possible overflow or underflow, the above sequences should work just fine. If you need to report an error if overflow occurs, you must check the 6502 overflow flag after performing an addition or subtraction. After an addition or subtraction the 6502 overflow flag will be clear if the result is within range and set if it is out of range. The 6502 BVC (branch if overflow clear) and BVS branch if overflow set) instructions can be used to check for an overflow or underflow condition (see Example 6).

Initializing and Copying Integer Variables

The MOVE and LOAD routines are used to copy and initialize integer variables in a SPEED/ASM program. LOAD lets you initialize an integer variable with an integer constant and MOVE lets you copy the contents of one integer variable into another.

The LOAD command uses the calling sequence:

```
JSR   LOAD
ADR   <value>,<name>
```

This routine copies the two-byte integer <value> into the variable specified by <name>. For example, to load the value 3765 into the variable I you would use the statement:

```
JSR   LOAD
ADR   3765,I
```

To load a negative number into the variable LISA 2.5 users must preface the negative number with the exclamation point (!). To load −438 into the variable I you should use the statement(s):

```
JSR   LOAD
ADR   !-438,I
```

LISA 3.0 users should omit the exclamation mark. These two state-

```
EXIT        EQU   $FF69
;
            CLC
            LDA   I
            ADC   J
            STA   K
            LDA   I+1
            ADC   J+1
            STA   K+1
            BVC   GOODADD              ;If error then
            JSR   PRINT                ;print in error
            BYT   "Error >32767",CR,0  ;message and
            JMP   EXIT                 ;quit the program.

GOODADD     ---                        ;Continue here if
            ---                        ;no overflow
            SEC
            LDA   I
            SBC   J
            STA   K
            LDA   I+1
            SBC   J+1
            STA   K+1
            BVC   GOODSUB
            JSR   PRINT
            BYT   "Error< -32768",CR,0
            JMP   EXIT

GOODSUB     ---
            ---
```

Example 6.

Listing 1. SPEED/ASM equate file.

```
0800                1          TTL "SPEED/ASM Equates"
0800                2  ;
0800                3  ********************************
0800                4  *                              *
0800                5  *  LISTING ONE: SPEED/ASM equate *
0800                6  *               file.          *
0800                7  *                              *
0800                8  ********************************
0800                9  ;
0800               10  ;
0800               11  ; GENERAL PURPOSE EQUATES
0800               12  ;
0000               13  FORASAV   EPZ 0
0001               14  FORXSAV   EPZ FORASAV+1
0002               15  FORYSAV   EPZ FORXSAV+1
0003               16  FORZPG    EPZ FORYSAV+1
0005               17  DESTADR   EPZ FORZPG+2
0007               18  PTRADR    EPZ DESTADR+2
0009               19  ISIMMED   EPZ PTRADR+2
000A               20  OP        EPZ ISIMMED+1
000C               21  MAXLEN    EPZ OP+2
000D               22  VALUE     EPZ MAXLEN+1
000F               23  DIGIT     EPZ VALUE+2
0010               24  LEAD0     EPZ DIGIT+1
0011               25  JMPADR    EPZ LEAD0+1
0013               26  COUNT     EPZ JMPADR+2
0014               27  GOTLN     EPZ COUNT+1
0015               28  LINEINDX  EPZ GOTLN+1
0016               29  SIGN      EPZ LINEINDX+1
0017               30  ACL       EPZ SIGN+1
0018               31  ACH       EPZ ACL+1
0019               32  XTNDL     EPZ ACH+1
001A               33  XTNDH     EPZ XTNDL+1
001B               34  AUXL      EPZ XTNDH+1
001C               35  AUXH      EPZ AUXL+1
0800               36  ;
0033               37  PROMPT    EPZ $33
004E               38  RNDL      EPZ $4E
004F               39  RNDH      EPZ $4F
0100               40  STACK     EQU $100
0200               41  INPUT     EQU $200
0800               42  ;
0800               43  ;
```

Listing continued.

ments are comparable to the Basic statements:

I = 3765

and

I = −438

respectively. Please note that the # and / operators are not required before the constant values. This is an unfortunate inconsistency, so you should take extra care to avoid either placing the # or / symbols here, or leaving the # and / symbols out of the operand field of the 6502 LDA or other arithmetic instruction. Remember, the LOAD routine is used to load a *constant* value into an integer variable. If you use a variable name as the first operand to the LOAD routine, the *address* of that variable, not its current contents, will be loaded into the destination variable.

The MOVE routine copies the *contents* of one integer variable into another. The MOVE command uses the syntax:

```
JSR     MOVE
ADR     <name1>,<name2>
```

MOVE copies the contents of <name1> *into* <name2>. So if you wanted to copy the contents of variable J into variable I you would use the statement:

```
JSR     MOVE
ADR     J,I
```

This is comparable to the Basic statement:

I = J

Always remember that MOVE copies the *contents* of an integer variable into another variable. If you use a constant as the first operand (or second operand for that matter), MOVE will simply go to the address in memory specified by that constant, get the two bytes, and store them into the destination variable.

I should point out that SPEED/ASM does very little type and range checking. MOVE and LOAD simply move values around. They don't care if you're actually dealing with integer variables. They store two bytes into the address you specify regardless of whether the variable is a character, integer, floating point, string var-

Listing continued.

```
0000             44  FALSE    EQU 0
0001             45  TRUE     EQU 1
008D             46  CR       EQU $8D
0800             47  ;
0800             48  ;
0800             49  ; "IF" STATEMENT EQUATES
0800             50  ;
00BD             51  EQ       EQU "="
00A3             52  NE       EQU "#"
00BE             53  GT       EQU ">"
00BC             54  LT       EQU "<"
BDBE             55  GE       EQU ">"|"="*256
BDBC             56  LE       EQU "<"|"="*256
0800             57  ;
0800             58  ;
0800             59  ;
0800             60  ;
0800             61  ;
0800             62  ;
0800             63  ;
0800             64  ; SPEED/ASM ENTRY POINTS
0800             65  ;
0800             66  ;
0800             67  ;
0800             68  ; NOTE: THE EQUATE OF PUTC MUST
0800             69  ; BE CHANGED IF YOU RELOCATE
0800             70  ; SPEED/ASM TO SOME LOCATION
0800             71  ; OTHER THAN $7800
0800             72  ;
7800             73  PUTC     EQU $7800
7803             74  GETC     EQU PUTC+3
7806             75  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809             76  SAPC     EQU SAGL+3          ;  "   "  "  "   "     "  "
780C             77  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F             78  READLN   EQU HOME+3
7812             79  INIT     EQU READLN+3
7815             80  FOR      EQU INIT+3
7818             81  FOR0     EQU FOR+3
781B             82  NEXT     EQU FOR0+3
781E             83  IFI      EQU NEXT+3
7821             84  IFI0     EQU IFI+3
7824             85  IFS      EQU IFI0+3
7827             86  IFS0     EQU IFS+3
782A             87  MOVE     EQU IFS0+3
782D             88  LOAD     EQU MOVE+3
7830             89  MOVS     EQU LOAD+3
7833             90  LDSTR    EQU MOVS+3
7836             91  PRINT    EQU LDSTR+3
7839             92  PRTSTR   EQU PRINT+3
783C             93  PRTINT   EQU PRTSTR+3
783F             94  RDSTR    EQU PRTINT+3
7842             95  RDINT    EQU RDSTR+3
7845             96  ONXGOTO  EQU RDINT+3
7848             97  CASE     EQU ONXGOTO+3
784B             98  CASEI    EQU CASE+3
784E             99  INSET    EQU CASEI+3
7851            100  NOTINSET EQU INSET+3
7854            101  ABS      EQU NOTINSET+3
7857            102  NEG      EQU ABS+3
785A            103  MUL      EQU NEG+3
785D            104  DIV      EQU MUL+3
7860            105  MOD      EQU DIV+3
7863            106  RND      EQU MOD+3
7866            107  SUBSTR   EQU RND+3
7869            108  INDEX    EQU SUBSTR+3
786C            109  LENGTH   EQU INDEX+3
786F            110  CONCAT   EQU LENGTH+3
7872            111  GETWZPG  EQU CONCAT+3        ;USED BY SPEED/ASM
7875            112  RDFP     EQU GETWZPG+3
7878            113  PRTE     EQU RDFP+3
787B            114  PRTF     EQU PRTE+3
787E            115  FADD     EQU PRTF+3
7881            116  FSUB     EQU FADD+3
7884            117  FMUL     EQU FSUB+3
7887            118  FDIV     EQU FMUL+3
788A            119  FLT      EQU FDIV+3
788D            120  FIX      EQU FLT+3
7890            121  FNEG     EQU FIX+3
7893            122  FADDIN   EQU FNEG+3
7896            123  FSUBIN   EQU FADDIN+3
7899            124  FTIMES   EQU FSUBIN+3
789C            125  FINTO    EQU FTIMES+3
789F            126  IFF      EQU FINTO+3
78A2            127  MOVFP    EQU IFF+3
0800            128  ;
0800            129          END
```

Listing continued.

iable, or even a 6502 instruction. Therefore you should take care that the destination operand of the LOAD routine and both operands of the MOVE routine are the names of properly defined integer variables in your program.

The Absolute Value and Negation Routines

SPEED/ASM provides two routines for negating and calculating the absolute value of an integer variable. The ABS routine (see Listing 1 for the equate for ABS) is invoked using the calling sequence:

```
JSR     ABS
ADR     <name>
```

This routine will take the variable whose name appears after the ADR pseudo-opcode, compute its absolute value, and store the absolute value back into the variable. This routine performs the same function as the Basic statement:

```
I = ABS(I)
```

Upon return from the ABS routine the overflow flag will be clear if the absolute value function was performed properly. If the user attempted to take the absolute value of −32768 (an error condition) then the overflow flag will be returned set. You can use the BVC and BVS instructions to test for this error condition.

The SPEED/ASM negate routine is used like the ABS routine; the only difference is that the sign is inverted with the negate routine instead of always returning a positive value (as with the ABS function). If the integer variable was negative, the NEG routine will make it positive. If the variable was positive, NEG will make it negative. NEG uses the calling sequence:

```
JSR     NEG
ADR     <name>
```

and is equivalent to the Basic statement:

```
I = -I
```

Since ABS and NEG operate on the variable *in place*, you may want to use the MOVE routine to copy the variable into another location before

Listing continued.

```
SYMBOL TABLE SORTED ALPHABETICALLY

ABS      7854  ACH      0018  ACL      0017  AUXH     001C  AUXL     001B
CASE     7848  CASEI    784B  CONCAT   786F  COUNT    0013  CR       008D
DESTADR  0005  DIGIT    000F  DIV      785D  EQ       00BD  FADD     787E
FADDTN   7893  FALSE    0000  FDIV     7887  FINIO    789C  FIX      788D
FLT      788A  FMUL     7884  FNEG     7890  FOR      7815  FOR0     7818
FORASAV  0000  FORXSAV  0001  FORYSAV  0002  FORZPG   0003  FSUB     7881
FSUBTN   7896  FTIMES   7899  GE       BDBE  GETC     7803  GETWZPG  7872
GOTLN    0014  GT       00BE  HOME     780C  IFF      789F  IFI      781E
IFI0     7821  IFS      7824  IFS0     7827  INDEX    7869  INIT     7812
INPUT    0200  INSET    784E  ISIMMED  0009  JMPADR   0011  LDSTR    7833
LE       BDBC  LEAD0    0010  LENGTH   786C  LINEINDX 0015  LOAD     782D
LT       00BC  MAXLEN   000C  MOD      7860  MOVE     782A  MOVFP    78A2
MOVS     7830  MUL      785A  NE       00A3  NEG      7857  NEXT     781B
NOTINSET 7851  ONXGOTO  7845  OP       000A  PRINT    7836  PROMPT   0033
PRTE     7878  PRTF     787B  PRTINT   783C  PRTSTR   7839  PTRADR   0007
PUTC     7800  RDFP     7875  RDINT    7842  RDSTR    783F  READLN   780F
RND      7863  RNDH     004F  RNDL     004E  SAGL     7806  SAPC     7809
SIGN     0016  STACK    0100  SUBSTR   7866  TRUE     0001  VALUE    000D
XTNDH    001A  XTNDL    0019

SYMBOL TABLE SORTED BY ADDRESS

FORASAV  0000  FALSE    0000  TRUE     0001  FORXSAV  0001  FORYSAV  0002
FORZPG   0003  DESTADR  0005  PTRADR   0007  ISIMMED  0009  OP       000A
MAXLEN   000C  VALUE    000D  DIGIT    000F  LEAD0    0010  JMPADR   0011
COUNT    0013  GOTLN    0014  LINEINDX 0015  SIGN     0016  ACL      0017
ACH      0018  XTNDL    0019  XTNDH    001A  AUXL     001B  AUXH     001C
PROMPT   0033  RNDL     004E  RNDH     004F  CR       008D  NE       00A3
LT       00BC  EQ       00BD  GT       00BE  STACK    0100  INPUT    0200
PUTC     7800  GETC     7803  SAGL     7806  SAPC     7809  HOME     780C
READLN   780F  INIT     7812  FOR      7815  FOR0     7818  NEXT     781B
IFI      781E  IFI0     7821  IFS      7824  IFS0     7827  MOVE     782A
LOAD     782D  MOVS     7830  LDSTR    7833  PRINT    7836  PRTSTR   7839
PRTINT   783C  RDSTR    783F  RDINT    7842  ONXGOTO  7845  CASE     7848
CASEI    784B  INSET    784E  NOTINSET 7851  ABS      7854  NEG      7857
MUL      785A  DIV      785D  MOD      7860  RND      7863  SUBSTR   7866
INDEX    7869  LENGTH   786C  CONCAT   786F  GETWZPG  7872  RDFP     7875
PRTE     7878  PRTF     787B  FADD     787E  FSUB     7881  FMUL     7884
FDIV     7887  FLT      788A  FIX      788D  FNEG     7890  FADDTN   7893
FSUBTN   7896  FTIMES   7899  FINIO    789C  IFF      789F  MOVFP    78A2
LE       BDBC  GE       BDBE
```

```
TEMP = (Y MOD Z)
TEMP = TEMP * X
TEMP = TEMP + 2
TEMP = X/TEMP
TEMP1 = J +3
TEMP = TEMP1 * TEMP
I = TEMP - 55
```

Example 7.

calling the ABS or NEG routines. For instance, if you wanted to perform the Basic instruction,

I = ABS(J)

using the SPEED/ASM statements,

```
JSR     ABS
ADR     J
JSR     MOVE
ADR     J,I
```

does not perform the same operation. It leaves the absolute value of J in both I and J. While in this simple example I easily could have moved the data into I and then taken the absolute value of I, this would be impossible in more complex situations. To handle situations like this, simply move J into some temporary location, take the absolute value of that location, then operate on the data in this temporary location as you wish.

The Multiplication, Division and Modulo Functions

The 6502 doesn't support the multiplication, division and modulo (remainder) operations within its instruction set. To make up for the lack of these instructions in the 6502 instruction set, the SPEED/ASM package provides three routines to perform these operations for you: The MUL, DIV and MOD routines. All three routines use the same format and calling sequence. The calling sequence is:

```
JSR  MUL      ;Or DIV Or MOD
ADR  <IVAR1>,<IVAR2>,<IVAR3>
```

```
JSR     MOD
ADR     Y,Z,TEMP

JSR     MUL
ADR     TEMP,X,TEMP

CLC
LDA     TEMP
ADC     #2
STA     TEMP
LDA     TEMP + 1
ADC     /2
STA     TEMP + 1

JSR     DIV
ADR     X,TEMP,TEMP

CLC
LDA     J
ADC     #3
STA     TEMP1
LDA     J + 1
ADC     /3
STA     TEMP1 + 1

JSR     MUL
ADR     TEMP,TEMP1, TEMP

SEC
LDA     TEMP
SBC     #55
STA     I
LDA     TEMP + 1
SBC     /55
STA     I + 1
```

Example 8.

This performs the operation:

"<IVAR3> = <IVAR1> * <IVAR2>"

If the division or modulo operation is called, then the operation performed is

"<IVAR3> = <IVAR1>/<IVAR2>"

or

"<IVAR3> = <IVAR1>MOD<IVAR2>"

The 6502 overflow flag is returned set if overflow occurred while performing a multiplication or if a division by zero occurred during the execution of the DIV or MOD routines. Unless you are quite sure that overflow or underflow will not occur, you should always follow a call to MUL, DIV or MOD with a BVC or BVS instruction to test the validity of the result.

Converting Complex Equations to The SPEED/ASM Format

The arithmetic routines (with the exception of the ABS and NEG routines) all require exactly *three* parameters. Basic, on the other hand, allows a rich variety of operations within a single statement. In Basic you could type:

```
I = (J+3)*(X/(2+X*(Y MOD Z))) - 55
```

Such a statement *cannot* be translated to a single statement in SPEED/ASM. Rather, the statement is broken down into the sequence of binary operations that make up this equation and the individual operations are handled by calls to SPEED/ASM routines. The previous equation would be broken down to the operations given in Example 7. This code would be converted to the SPEED/ASM statements in Example 8. For purposes of clarity, the tests for overflow were omitted from this code. But it should help demonstrate how you translate a Basic expression into a sequence of SPEED/ASM routine calls.

The Random Number Function RND

The SPEED/ASM package provides a function that returns a random number every time it's called. The calling sequence is:

```
JSR   RND
ADR   <IVAR>
```

When ever RND is called it stuffs a pseudo-random number in the range 0...32767 and stores it in the variable that follows the JSR. If you wish to generate a random number in the range 0...n then call the random number generator and use the MOD routine. For example, to get a number in the range 0...25 you should use the code given in Example 9.

```
JSR   LOAD
ADR   26,TEMP
JSR   RND
ADR   RNDVAL
JSR   MOD
ADR   RNDVAL,TEMP,RNDVAL
```

Example 9.

Note that the mod of RNDVAL and 26 was taken. This produces a value in the range of 0...25.

Performing I/O in SPEED/ASM

Before discussing integer I/O in SPEED/ASM, I should first introduce character I/O, since numeric I/O is dependent upon character I/O. Five routines are associated with character I/O in SPEED/ASM: GETC, PUTC, READLN, HOME and INIT.

The INIT routine, as I've already mentioned, *must* be called before calling any SPEED/ASM routines. In

```
ENTRNUM     JSR     READLN
            JSR     RDINT
            ADR     J
            BVC     GOODNUM
;
            CMP     #0
            BEQ     BADNUM
            BMI     RANGERR
;
;Must be one at this point
;
BADNUM      JSR     PRINT
            BYT     "Bad character in number, re-enter",CR,0
            JMP     ENTRNUM
;
RANGERR     JSR     PRINT
            BYT     "Value out of range, re-enter",CR,0
            JMP     ENTRNUM
;
;
GOODNUM     ---     ;Continue processing here
```

Example 10.

Listing 2. SPEED/ASM demo program.

```
08U0              1          TTL "SPEED/ASM Demo"
08U0              2  ;
08U0              3  ***********************************
08U0              4  *                                 *
08U0              5  *  Listing Two: SPEED/ASM demo    *
08U0              6  *               program.          *
08U0              7  *                                 *
08U0              8  ***********************************
08U0              9  ;
08U0             10  ;
08U0             11  ;
0000             12  FALSE    EQU 0
0001             13  TRUE     EQU 1
008D             14  CR       EQU $8D
08U0             15  ;
08U0             16  ;
08U0             17  ;
08U0             18  ;
08U0             19  ;
08U0             20  ; SPEED/ASM ENTRY POINTS
08U0             21  ; (Only the equates necessary for
08U0             22  ;  this demo are included.)
08U0             23  ;
08U0             24  ;
08U0             25  ;
08U0             26  ; NOTE: THE EQUATE OF PUTC MUST
08U0             27  ; BE CHANGED IF YOU RELOCATE
08U0             28  ; SPEED/ASM TO SOME LOCATION
0800             29  ; OTHER THAN $7800
08U0             30  ;
7800             31  PUTC     EQU $7800
7803             32  GETC     EQU PUTC+3
7806             33  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809             34  SAPC     EQU SAGL+3          ;  "   "   "   "   "   "   "
780C             35  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F             36  READLN   EQU HOME+3
7812             37  INIT     EQU READLN+3
7815             38  FOR      EQU INIT+3
7818             39  FOR0     EQU FOR+3
781B             40  NEXT     EQU FOR0+3
781E             41  IFI      EQU NEXT+3
7821             42  IFI0     EQU IFI+3
7824             43  IFS      EQU IFI0+3
7827             44  IFS0     EQU IFS+3
782A             45  MOVE     EQU IFS0+3
782D             46  LOAD     EQU MOVE+3
7830             47  MOVS     EQU LOAD+3
7833             48  LDSTR    EQU MOVS+3
7836             49  PRINT    EQU LDSTR+3
7839             50  PRTSTR   EQU PRINT+3
783C             51  PRTINT   EQU PRTSTR+3
783F             52  RDSTR    EQU PRTINT+3
7842             53  RDINT    EQU RDSTR+3
7845             54  CNXGOTO  EQU RDINT+3
```

Listing continued.

particular it must be called before performing any I/O routines since several pointers and counters used by the I/O package are initialized by INIT. Failing to call INIT before performing an I/O operation may result in garbled data.

HOME is used to clear the screen and position the cursor in the upper left corner. This routine is included in the SPEED/ASM package to obtain a certain amount of machine independence. By placing this jump in the SPEED/ASM code (instead of the user program), it will have to be changed in only one location if you want to move the program to a computer other than the Apple II. Versions of SPEED/ASM will eventually be available for the Atari, PET, VIC and other 6502 computers, allowing you to easily move a program from one computer to another. HOME's purpose is to help minimize the machine dependent code.

All input from the system console is handled *line by line*. Any time you read a character, number or string from the keyboard, the SPEED/ASM routines will read the data from the current line input buffer. If the buffer is empty, the user is prompted to enter a new line from the keyboard. This works fine until you prompt the user for some input (expecting him to enter a new line from the keyboard) and the SPEED/ASM package uses the last few characters on the previous line as the input. To insure that the next input performed takes its data from the beginning of a new input line, you should call the READLN (read a line) routine to force the user to enter a new line of data. READLN will wait until the user types in a complete line of text and then it will continue execution with the next statement following the call to the READLN routine.

The GETC routine reads a single character from the current line buffer and returns it in the 6502 accumulator. If the line buffer is empty, a new line is read from the keyboard and GETC returns the first character on that line. I must point out that if there are characters in the input line buffer the keyboard *will not* be read.

Listing continued.

```
7848                    55  CASE     EQU ONXGOTO+3
784B                    56  CASEI    EQU CASE+3
784E                    57  INSET    EQU CASEI+3
7851                    58  NOTINSET EQU INSET+3
7854                    59  ABS      EQU NOTINSET+3
7857                    60  NEG      EQU ABS+3
785A                    61  MUL      EQU NEG+3
785D                    62  DIV      EQU MUL+3
7860                    63  MOD      EQU DIV+3
7863                    64  RND      EQU MOD+3
0800                    65  ;
0800                    66  ;
0800                    67  ;
0800                    68  ; Apple monitor equates:
0800                    69  ;
FF69                    70  EXIT     EQU $FF69          ;Address to quit S/A.
0800                    71  ;
0800                    72  ;
0800                    73  ;
0800                    74  ********************************
0800                    75  ;
0800                    76  ;
0800                    77  ; NOTE: INIT must be called before
0800                    78  ; any other SPEED/ASM routine.
0800                    79  ;
0800 20 12 78           80  START    JSR INIT
0803                    81  ;
0803                    82  ;
0803                    83  ; The following code loads 10 into
0803                    84  ; "I" and 54 into "J" then computes
0803                    85  ; their sum, difference, product,
0803                    86  ; quotient, and remainder.
0803                    87  ;
0803 20 2D 78           88          JSR LOAD
0806 0A 00 12           89          ADR 10,I
0809 0B
080A 20 2D 78           90          JSR LOAD
080D 36 00 14           91          ADR 54,J
0810 0B
0811                    92  ;
0811                    93  ; Compute the sum:
0811                    94  ;
0811 18                 95  LOOP     CLC                ;Always CLC before an addition.
0812 AD 12 0B           96          LDA I
0815 6D 14 0B           97          ADC J
0818 8D 16 0B           98          STA SUM
081B AD 13 0B           99          LDA I+1
081E 6D 15 0B          100          ADC J+1
0821 8D 17 0B          101          STA SUM+1
0824 70 69             102          BVS OVERFLOW       ;Check for >32767.
0826                   103  ;
0826                   104  ; Calculate the difference:
0826                   105  ;
0826 38                106          SEC                ;Always SEC before a subtraction
0827 AD 12 0B          107          LDA I
082A ED 14 0B          108          SBC J
082D 8D 1C 0B          109          STA DIFFRNCE
0830 AD 13 0B          110          LDA I+1
0833 ED 15 0B          111          SBC J+1
0836 8D 1D 0B          112          STA DIFFRNCE+1
0839 70 54             113          BVS OVERFLOW
083B                   114  ;
083B                   115  ; Calculate the product:
083B                   116  ;
083B 20 5A 78          117          JSR MUL
083E 12 0B 14          118          ADR I,J,PRODUCT
0841 0B 18 0B
0844 70 49             119          BVS OVERFLOW
0846                   120  ;
0846                   121  ; Calculate the quotient
0846                   122  ;
0846 20 5D 78          123          JSR DIV
0849 14 0B 12          124          ADR J,I,QUOTIENT   ;J/I, not I/J
084C 0B 1A 0B
084F 70 3E             125          BVS OVERFLOW
0851                   126  ;
0851                   127  ; Calculate the remainder
0851                   128  ;
0851 20 60 78          129          JSR MOD
0854 14 0B 12          130          ADR J,I,REMANDR
0857 0B 1E 0B
085A 70 33             131          BVS OVERFLOW
085C                   132  ;
085C                   133  ;
085C                   134  ; Generate a couple of random numbers
085C                   135  ;
085C 20 63 78          136          JSR RND
085F 20 0B             137          ADR RANDOM1
0861 20 54 78          138          JSR ABS
0864 20 0B             139          ADR RANDOM1
0866 70 27             140          BVS OVERFLOW
0868                   141  ;
0868 20 63 78          142          JSR RND
```

Listing continued.

Instead, the next available character in the input buffer will be returned in the 6502 accumulator. If you need to read the character from the keyboard, always call READLN before calling GETC.

The final character I/O routine is the PUTC routine. PUTC takes the character in the 6502 accumulator and outputs it to the console screen. One nice feature of the PUTC routine is that it will automatically convert lowercase to uppercase if the end user of your program cannot display lowercase on his Apple. If your system has a lowercase adapter, like the Lazer MicroSystems' Lower Case + Plus and Keyboard + Plus modules, then you can write your SPEED/ASM programs using easy-to-read lowercase without having to worry about incompatibility problems.

The READLN, GETC and PUTC routines are *primitive* routines. All other I/O routines can be synthesized from these three subprograms. When I talk about character operations we'll return to the discussion of the GETC and PUTC routines.

Using PRINT to Print String Literals

I've already used the SPEED/ASM PRINT routine in several examples. A formal definition of the PRINT routine will help explain its use in your SPEED/ASM programs.

The PRINT routine is used to print a sequence of ASCII characters to the Apple's video screen. This routine prints every character following the JSR PRINT instruction up to, but not including, a zero terminating byte. Upon encountering a zero byte, the PRINT routine terminates output, and control is returned to the 6502 instruction that follows the zero byte.

PRINT is useful for printing messages, prompts and other string literal output. PRINT does not automatically eject a carriage return after the string is printed. If you wish to output a carriage return you must explicitly include the ASCII code for the carriage return in your output string; i.e.,

```
JSR   PRINT
BYT   "STRING followed by Return",CR,0
```

Listing continued.

```
086B 22 0B      143           ADR RANDOM2
086D 20 54 78   144           JSR ABS
0870 22 0B      145           ADR RANDOM2
0872 70 1B      146           BVS OVERFLOW
0874 20 57 78   147 20 57 78  JSR NEG
0877 22 0B      148           ADR RANDOM2
0879 70 14      149           BVS OVERFLOW
087B            150 ;
087B 20 63 78   151           JSR RND
087E 24 0B      152           ADR RANDOM3
0880 20 57 78   153           JSR NEG
0883 24 0B      154           ADR RANDOM3
0885 70 08      155           BVS OVERFLOW
0887            156 ;
0887 20 63 78   157           JSR RND
088A 26 0B      158           ADR RANDOM4
088C 4C A9 08   159           JMP PRINUMS
088F            160 ;
088F 20 36 78   161 OVERFLOW JSR PRINT
0892 8D 8D CF   162           BYT CR,CR,"Overflow occured",CR,0
0895 F6 E5 F2
0898 E5 BC EF
089B F7 A0 EF
089E E3 E3 F5
08A1 F2 E5 E4
08A4 8D 00
08A6 4C 69 FF   163           JMP EXIT
08A9            164 ;
08A9            165 ;
08A9            166 ;
08A9            167 ; Print the sum:
08A9            168 ;
08A9 20 36 78   169 PRINUMS  JSR PRINT
08AC C9 BD A0   170           BYT "I= ",0
08AF 00
08B0 20 3C 78   171           JSR PRTINT
08B3 12 0B      172           ADR I
08B5 20 36 78   173           JSR PRINT
08B8 8D CA BD   174           BYT CR,"J= ",0
08BB A0 00
08BD 20 3C 78   175           JSR PRTINT
08C0 14 0B      176           ADR J
08C2            177 ;
08C2 20 36 78   178           JSR PRINT
08C5 8D D4 E8   179           BYT CR,"The sum is ",0
08C8 E5 A0 F3
08CB F5 ED A0
08CE E9 F3 A0
08D1 00
08D2 20 3C 78   180           JSR PRTINT
08D5 16 0B      181           ADR SUM
08D7            182 ;
08D7            183 ; Print the difference:
08D7            184 ;
08D7 20 36 78   185           JSR PRINT
08DA 8D D4 E8   186           BYT CR,"The difference (I-J) is ",0
08DD E5 A0 E4
08E0 E9 E6 E6
08E3 E5 F2 E5
08E6 EE E3 E5
08E9 A0 A8 C9
08EC AD CA A9
08EF A0 E9 F3
08F2 A0 00
08F4 20 3C 78   187           JSR PRTINT
08F7 1C 0B      188           ADR DIFFRNCE
08F9            189 ;
08F9            190 ; Print the product:
08F9            191 ;
08F9 20 36 78   192           JSR PRINT
08FC 8D D4 E8   193           BYT CR,"The product is ",0
08FF E5 A0 F0
0902 F2 EF E4
0905 F5 E3 F4
0908 A0 E9 F3
090B A0 00
090D 20 3C 78   194           JSR PRTINT
0910 18 0B      195           ADR PRODUCT
0912            196 ;
0912            197 ; Print the quotient
0912            198 ;
0912 20 36 78   199           JSR PRINT
0915 8D D4 E8   200           BYT CR,"The quotient (J/I) is ",0
```

Listing continued.

Listing continued.

```
0918 E5 A0 F1
091B F5 EF F4
091E E9 E5 EE
0921 F4 A0 A8
0924 CA AF C9
0927 A9 A0 E9
092A 00          092A:F3 A0 00
092D 20 3C 78   201          JSR PRTINT
0930 1A 0B      202          ADR QUOTIENT
0932            203  ;
0932            204  ; Print the remainder:
0932            205  ;
0932 20 36 78   206          JSR PRINT
0935 8D D4 E8   207          BYT CR,"The remainder (J mod I) is ",0
0938 E5 A0 F2
093B E5 ED E1
093E E9 EE E4
0941 E5 F2 A0
0944 A8 CA A0
0947 ED EF E4
094A A0 C9 A9
094D A0 E9 F3
0950 A0 00
0952 20 3C 78   208          JSR PRTINT
0955 1E 0B      209          ADR REMAINDR
0957            210  ;
0957            211  ; Print the random numbers:
0957            212  ;
0957 20 36 78   213          JSR PRINT
095A 8D D4 E8   214          BYT CR,"The random numbers are:"
095D E5 A0 F2
0960 E1 EE E4
0963 EF ED A0
0966 EE F5 ED
0969 E2 E5 F2
096C F3 A0 E1
096F F2 E5 BA
0972 8D D2 E1   215          BYT CR,"Random1: ",0
0975 EE E4 EF
0978 ED B1 BA
097B A0 00
097D 20 3C 78   216          JSR PRTINT
0980 20 0B      217          ADR RANDOM1
0982 20 36 78   218          JSR PRINT
0985 8D D2 E1   219          BYT CR,"Random2: ",0
0988 EE E4 EF
098B ED B2 BA
098E A0 00
0990 20 3C 78   220          JSR PRTINT
0993 22 0B      221          ADR RANDOM2
0995 20 36 78   222          JSR PRINT
0998 8D D2 E1   223          BYT CR,"Random3: ",0
099B EE E4 EF
099E ED B3 BA
09A1 A0 00
09A3 20 3C 78   224          JSR PRTINT
09A6 24 0B      225          ADR RANDOM3
09A8 20 36 78   226          JSR PRINT
09AB 8D D2 E1   227          BYT CR,"Random4: ",0
09AE EE E4 EF
09B1 ED B4 BA
09B4 A0 00
09B6 20 3C 78   228          JSR PRTINT
09B9 26 0B      229          ADR RANDOM4
09BB            230  ;
09BB            231  ;
09BB            232  ; Demonstrate the MOVE subroutine
09BB            233  ;
09BB 20 36 78   234          JSR PRINT
09BE 8D 8D C3   235          BYT CR,CR,"Current contents of I is ",0
09C1 F5 F2 F2
09C4 E5 EE F4
09C7 A0 E3 EF
09CA EE F4 E5
09CD EE F4 F3
09D0 A0 EF E6
09D3 A0 C9 A0
09D6 E9 F3 A0
09D9 00
09DA 20 3C 78   236          JSR PRTINT
09DD 12 0B      237          ADR I
09DF 20 36 78   238          JSR PRINT
09E2 8D C3 F5   239          BYT CR,"Current contents of J is ",0
09E5 F2 F2 E5
09E8 EE F4 A0
09EB E3 EF EE
09EE F4 E5 EE
09F1 F4 F3 A0
09F4 EF E6 A0
09F7 CA A0 E9
09FA F3 A0 00
09FD 20 3C 78   240          JSR PRTINT
```

Listing continued.

Note that CR was used instead of the actual code for carriage return ($8D). CR is a symbol, defined in the SPEED/ASM equates, which is replaced by the value $8D.

Since PRINT will print all characters up to the terminating zero byte, multiple lines can be output using a single call to the PRINT subroutine. Simply separate each line with a carriage return and PRINT will output the text on several lines:

```
JSR     PRINT
BYT     "This is the first line,
        and it is follwed by",CR
BYT     "this second line.",CR,0
```

Other than improving the readability of the program, the separate lines need not appear on separate source lines as in this example. The second string could have immediately followed the CR on the first line. This type of coding, however, is not recommended because it makes the source file much harder to read.

Performing Integer I/O In SPEED/ASM

Operating on integer values is one of the primary functions you will do in SPEED/ASM. However, these operations are almost useless unless you can communicate the results of these operations to the world outside the computer. Two routines are provided in the SPEED/ASM package to facilitate integer I/O: RDINT (read an integer) and PRTINT (print an integer).

Printing an integer using the PRTINT routine is easy—just follow the JSR PRTINT with the address of the integer you want to print. For example, if you wanted to print the contents of the integer variable I onto the Apple's video screen you would use the statement(s):

```
JSR     PRTINT
ADR     I
```

and the contents of I would be displayed for you. In the next installation of this series I will discuss how to format this output to create a pretty listing.

The RDINT routine is a little more complicated to use than the PRTINT routine because there is the possibility that an error condition might oc-

Listing continued.

```
0A00 14 0B      241           ADR J
0A02            242  ;
0A02 20 2A 78   243           JSR MOVE
0A05 12 0B 14   244           ADR I,J
0A08 0B
0A09            245  ;
0A09 20 36 78   246           JSR PRINT
0A0C 8D CE EF   247           BYT CR,"Now I contains ",0
0A0F F7 A0 C9
0A12 A0 E3 EF
0A15 EE F4 E1
0A18 E9 EE F3
0A1B A0 00
0A1D 20 3C 78   248           JSR PRTINT
0A20 12 0B      249           ADR I
0A22 20 36 78   250           JSR PRINT
0A25 8D E1 EE   251           BYT CR,"and J contains ",0
0A28 E4 A0 CA
0A2B A0 E3 EF
0A2E EE F4 E1
0A31 E9 EE F3
0A34 A0 00
0A36 20 3C 78   252           JSR PRTINT
0A39 14 0B      253           ADR J
0A3B            254  ;
0A3B            255  ;
0A3B            256  ; Ask the user if he wants to re-run
0A3B            257  ; the program with user input.
0A3B            258  ;
0A3B            259  ;
0A3B 20 36 78   260  RERUN    JSR PRINT
0A3E 8D 8D C4   261           BYT CR,CR,"Do you wish to re-run this",CR
0A41 EF A0 F9
0A44 EF F5 A0
0A47 F7 E9 F3
0A4A E8 A0 F4
0A4D EF A0 F2
0A50 E5 AD F2
0A53 F5 EE A0
0A56 F4 E8 E9
0A59 F3 8D
0A5B F0 F2 EF   262           BYT "program (Y/N)? ",0
0A5E E7 F2 E1
0A61 ED A0 A8
0A64 D9 AF CE
0A67 A9 BF A0
0A6A 00
0A6B 20 03 78   263           JSR GETC
0A6E 29 DF      264           AND #$DF            ;Convert Lower case to Upper case
0A70 20 00 78   265           JSR PUTC
0A73 C9 CE      266           CMP #"N"
0A75 D0 03      267           BNE >1
0A77 4C 69 FF   268           JMP EXIT
0A7A            269  ;
0A7A C9 D9      270  ^1       CMP #"Y"
0A7C D0 BD      271           BNE RERUN
0A7E            272  ;
0A7E            273  ;
0A7E            274  ; If so, get new values for I and J
0A7E            275  ;
0A7E            276  ;
0A7E 20 36 78   277  BADNUM1  JSR PRINT
0A81 8D         278           BYT CR
0A82 C5 EE F4   279           BYT "Enter a new value for I:",0
0A85 E5 F2 A0
0A88 E1 A0 EE
0A8B E5 F7 A0
0A8E F6 E1 EC
0A91 F5 E5 A0
0A94 E6 EF F2
0A97 A0 C9 BA
0A9A 00
0A9B 20 0F 78   280           JSR READLN
0A9E 20 42 78   281           JSR RDINT
0AA1 12 0B      282           ADR I
0AA3 50 21      283           BVC GOODNUM1
0AA5 20 36 78   284           JSR PRINT
0AA8 8D C5 F2   285           BYT CR,"Error in entry, re-enter",CR,0
0AAB F2 EF F2
0AAE A0 E9 EE
0AB1 A0 E5 EE
0AB4 F4 F2 F9
0AB7 AC A0 F2
0ABA E5 AD E5
0ABD EE F4 E5
0AC0 F2 8D 00
0AC3 4C 7E 0A   286           JMP BADNUM1
0AC6            287  ;
0AC6 20 36 78   288  GOODNUM1 JSR PRINT
0AC9 8D C5 EE   289           BYT CR,"Enter a new value for J:",0
0ACC F4 E5 F2
0ACF A0 E1 A0
```

Listing continued.

Listing continued.

```
0AD2 EE E5 F7
0AD5 A0 F6 E1
0AD8 EC F5 E5
0ADB A0 E6 EF
0ADE F2 A0 CA
0AE1 BA 00
0AE3 20 0F 78    290          JSR READLN
0AE6 20 42 78    291          JSR RDINT
0AE9 14 0B       292          ADR J
0AEB 50 22       293          BVC >0
0AED 20 36 78    294          JSR PRINT
0AF0 8D C2 E1    295          BYT CR,"Bad value for J, re-enter",CR,0
0AF3 E4 A0 F6
0AF6 E1 EC F5
0AF9 E5 A0 E6
0AFC EF F2 A0
0AFF CA AC A0
0B02 F2 E5 AD
0B05 E5 EE F4
0B08 E5 F2 8D
0B0B 00
0B0C 4C C6 0A    296          JMP GOODNUM1
0B0F             297  ;
0B0F 4C 11 08    298  ^0      JMP LOOP
0B12             299  ;
0B12             300  ;
0B12             301  ;
0B12             302  ;
0B12             303  ;
0B12             304  ;
0B12             305  ;
0B12             306  ;
0B12             307  **********************************
0B12             308  ;
0B12             309  ; Variable declarations:
0B12             310  ;
0B12             311  ; The following variables are all
0B12             312  ; integers.  So they are declared
0B12             313  ; with the ADR pseudo opcode to
0B12             314  ; reserve two bytes for each integer.
0B12             315  ;
0B12             316  ;
0B12 00 00       317  I       ADR 0
0B14 00 00       318  J       ADR 0
0B16             319  ;
0B16 00 00       320  SUM     ADR 0
0B18 00 00       321  PRODUCT ADR 0
0B1A 00 00       322  QUOTIENT ADR 0
0B1C 00 00       323  DIFFRNCE ADR 0
0B1E 00 00       324  REMAINDR ADR 0
0B20 00 00       325  RANDOM1 ADR 0
0B22 00 00       326  RANDOM2 ADR 0
0B24 00 00       327  RANDOM3 ADR 0
0B26 00 00       328  RANDOM4 ADR 0
0B28             329  ;
0B28             330          END

***** END OF ASSEMBLY

]brun sort

BRUN SORT
SYMBOL TABLE SORTED ALPHABETICALLY

  ABS      7854  BADNUM1  0A7E  CASE     7848  CASEI    784B  CR       008D
  DIFFRNCE 0B1C  DIV      785D  EXIT     FF69  FALSE    0000  FOR      7815
  FOR0     7818  GETC     7803  GOODNUM1 0AC6  HOME     780C  I        0B12
  IFI      781E  IFI0     7821  IFS      7824  IFS0     7827  INIT     7812
  INSET    784E  J        0B14  LDSTR    7833  LOAD     782D  LOOP     0811
  MOD      7860  MOVE     782A  MOVS     7830  MUL      785A  NEG      7857
  NEXT     781B  NOTINSET 7851  ONXGOTO  7845  OVERFLOW 088F  PRINT    7836
  PRODUCT  0B18  PRTINT   783C  PRTNUMS  08A9  PRTSTR   7839  PUTC     7800
  QUOTIENT 0B1A  RANDOM1  0B20  RANDOM2  0B22  RANDOM3  0B24  RANDOM4  0B26
  RDINT    7842  RDSTR    783F  READLN   780F  REMAINDR 0B1E  RERUN    0A3B
  RND      7863  SAGL     7806  SAPC     7809  START    0800  SUM      0B16
  TRUE     0001

SYMBOL TABLE SORTED BY ADDRESS

  FALSE    0000  TRUE     0001  CR       008D  START    0800  LOOP     0811
  OVERFLOW 088F  PRTNUMS  08A9  RERUN    0A3B  BADNUM1  0A7E  GOODNUM1 0AC6
  I        0B12  J        0B14  SUM      0B16  PRODUCT  0B18  QUOTIENT 0B1A
  DIFFRNCE 0B1C  REMAINDR 0B1E  RANDOM1  0B20  RANDOM2  0B22  RANDOM3  0B24
  RANDOM4  0B26  PUTC     7800  GETC     7803  SAGL     7806  SAPC     7809
  HOME     780C  READLN   780F  INIT     7812  FOR      7815  FOR0     7818
  NEXT     781B  IFI      781E  IFI0     7821  IFS      7824  IFS0     7827
  MOVE     782A  LOAD     782D  MOVS     7830  LDSTR    7833  PRINT    7836
  PRTSTR   7839  PRTINT   783C  RDSTR    783F  RDINT    7842  ONXGOTO  7845
  CASE     7848  CASEI    784B  INSET    784E  NOTINSET 7851  ABS      7854
  NEG      7857  MUL      785A  DIV      785D  MOD      7860  RND      7863
  EXIT     FF69
```

Listing continued.

cur. The RDINT routine expects the user to type a valid numeric integer which takes the form:

1) Any number of leading blanks, commas or carriage returns, followed by
2) An optional minus sign, followed by
3) One to five digits forming a value in the range 0...32767, followed by
4) A space, comma or carriage return.

If the numeric string is of the proper format then SPEED/ASM will store the value into the integer variable whose address follows the JSR; e.g.,

```
JSR     RDINT
ADR     J
```

will read an integer variable from the line input buffer (reading a new line if necessary) and store the numeric value into J.

If an input error occurs, then the V flag will be returned set so you can use the BVS or BVC instruction to test for the error condition. Three error conditions can be returned in the 6502 accumulator. If the overflow flag is set, then the accumulator contains zero if the last character of the number wasn't a space, comma or carriage return. This error condition can be considered optional. If you want to allow characters other than space, comma and carriage return at the end of a number, you can ignore this error.

If the overflow flag is set and the accumulator contains one, the first character of the number was not a valid digit or minus sign. All preceding spaces, commas and carriage returns were stripped before the failure to obtain a digit or minus sign was detected. This is a definite error and your program should prompt the user to re-enter the data.

If the overflow flag was set and the accumulator contained $8D, the value entered by the user was greater than 32767 or less than −32768. Obviously this number must be re-entered by the user. A program that would prompt the user to re-enter on an entry error is shown in Example 10.

Listing continued.

```
DOES YOUR APPLE SUPPORT LOWER CASE
DISPLAY? (Y/N):

I= 10
J= 54
The sum is 64
The difference (I-J) is -44
The product is 540
The quotient (J/I) is 5
The remainder (J mod I) is 4
The random numbers are:
Random1: 11979
Random2: -28681
Random3: 3539
Random4: 31519

Current contents of I is 10
Current contents of J is 54
Now I contains 10
and J contains 10

Do you wish to re-run this
program (Y/N)? Y

Enter a new value for I:25

Enter a new value for J:36
I= 25
J= 36
The sum is 61
The difference (I-J) is -11
The product is 900
The quotient (J/I) is 1
The remainder (J mod I) is 11
The random numbers are:
Random1: 30641
Random2: -6027
Random3: 25673
Random4: 14189

Current contents of I is 25
Current contents of J is 36
Now I contains 25
and J contains 25

Do you wish to re-run this
program (Y/N)? Y

Enter a new value for I:59

Enter a new value for J:22
I= 59
J= 22
The sum is 81
The difference (I-J) is 37
The product is 1298
The quotient (J/I) is 0
The remainder (J mod I) is 22
The random numbers are:
Random1: 26303
Random2: -20411
Random3: -5287
Random4: -1347

Current contents of I is 59
Current contents of J is 22
Now I contains 59
and J contains 59

Do you wish to re-run this
program (Y/N)? N
```

I have included additional examples in Listing 2.

Looking Forward

So far the examples have been rather trivial since the SPEED/ASM routines presented thus far haven't included the necessary looping, conditional, and transfer of control routines. Next time I'll start discussing program control structures so that you will be able to start writing fairly complex programs. See ya next time! ■

Note: SPEED/ASM and LISA v2.5 are available from Sierra On-Line, 209-683-6858. These programs are also available at your local dealer.

The Assembly Advantage

by Randy Hyde

Instruction Primer

In order to write programs in 6502 assembly language a considerable amount of seemingly unrelated background knowledge is required. The major purpose behind the SPEED/ASM package is to reduce the amount of that knowledge necessary to write useful programs. Mind you I said reduce, not eliminate. With this third installment of my column I will finish up most of the necessary background material so you can begin writing reasonable machine language programs. This article should also help tie up loose ends from the first two articles.

An Introduction to 6502 Assembly Language

You cannot properly program using SPEED/ASM without at least a little understanding of 6502 machine code. SPEED/ASM's purpose, as stated above, is not to eliminate the need for machine code, but to make the more difficult tasks easier. With this thought in mind it's time to introduce you to the simpler 6502 instructions.

The LDA Instruction

The LDA (load accumulator) instruction is easily the most used instruction on the 6502. It copies the contents of one of the 65,535 memory locations the 6502 can address into a special memory location called the accumulator. This memory location is found *inside* the 6502 chip. Storage cells located inside the processor chip are usually called registers.

The accumulator in the 6502 is where most of the action takes place. Numbers are added together in the accumulator, strings are compared using the accumulator, logical operations are performed by the accumulator, and more. Typically, a value is loaded into the accumulator using the LDA instruction and then that data is operated on using one of the 6502's arithmetic or logical operations. Consequently, the LDA instruction is usually the first instruction of any computational sequence that is executed.

There are several ways the accumulator can be loaded: with a constant, from a variable, from a string, or indirectly through a pointer. For the time being we will concern ourselves with the first two *addressing modes*, loading the accumulator with a constant and loading it from a variable.

Constants are specified by prefacing them with the pound sign, #. For example, to load the accumulator with 55 you would use the instruction:

```
LDA #55
```

There is one limitation on a constant you load into the 6502 accumulator—it cannot be greater than 255 or less than 0. The range limitation is due to the 8-bit size of the 6502 accumulator. Attempting to load a value larger than 255 will yield the result <value> MOD 256. For example:

```
START   LDA   #"C"
        STA   CHR
        JSR   LOAD
        ADR   0,INT
;
; Print the character onto the screen
; as an ASCII character
;
        LDA   CHR
        JSR   PUTC
;
; Transfer the character to INT and
; print its ASCII code as a decimal
; value.
;
        LDA   CHR
        STA   INT
        JSR   PRTINT
        ADR   INT
;
; Load CHR with the value 204
; and then print CHR on the screen
; as an ASCII character.
;
        LDA   #204
        STA   CHR
        LDA   CHR
        JSR   PUTC
```

Listing 1. SPEED/ASM program segment demonstrating the relationship between characters and their standard ASCII codes.

```
EXIT      EQU   $FF59
;
READLP    JSR   GETC
          CMP   #CR          ;Constant declared in SPEED/ASM.EQUATES.
          BEQ   ALLDONE
          JSR   PUTC
          JMP   READLP
;
ALLDONE   JMP   EXIT         ;Return to Apple monitor.
```

Listing 2. Sample application of BEQ branch instruction.

Address correspondence to Randy Hyde, 925 Lorna St., Corona, CA 91720.

```
;
; To emulate the statement
;
;    C$ = CHR$(I)
;
     LDA   I
     ORA   #$80     ;Set the H.O. bit to one.
     STA   C
;
;
; To emulate the statement
;
;    I = ASC(C$)
;
     LDA   C
     AND   #$7F     ;Set H.O. bit to zero.
     STA   I
     LDA   #0       ;Clear the H.O. byte of the
     STA   I+1      ;integer variable.
```

Listing 3. Code sequences to implement Applesoft CHR$ and ASC instructions. Applesoft utilizes a non-standard 0–127 range for its ASCII codes.

LDA #305

loads the accumulator with 49 (305 MOD 256 is 49). If you need to handle numbers too large for 8 bits you have to split the operation into two segments, the first handling the *low order byte* of the computation and the second handling computation of the *high order bytes*.

The integer values used by the SPEED/ASM package require 2 bytes to represent values in the range 0 to 65535 or (more commonly) signed values in the range −32768 to 32767. The pound sign operator lets you load the low order byte of a constant into the accumulator. To get the high order byte of a constant into the accumulator the slash, /, operator is used. The instruction:

LDA /305

loads the high order byte of 305 (which is 1) into the accumulator.

To load a negative decimal value into the 6502 accumulator preface the negative number with an exclamation point (LISA 2.5 users only). For example, to load the accumulator with the low order byte of −465 you would use the instruction:

LDA #!−465

To load the accumulator with the high order byte of −465 the instruction:

LDA /! - 465

is used.

To load a character constant into the 6502 accumulator follow the pound sign with the character you wish to load, enclosed by quotes. To load the character c into the accumulator you could use the instruction:

LDA #"c"

Do not use a slash—that will always put 0 into the accumulator. Normally a character is enclosed by quote marks. If you enclose it in apostrophes the high order bit of the character will be set to 0. Since the Apple normally likes its characters to have the high order bit set to 1 you should always enclose a character in quotes.

Next to loading a constant into the accumulator, loading the contents of a memory location is the most important function. To load the accumulator with the contents of a memory location specify the address of that location after the LDA instruction. To load the low order byte of the variable I into the accumulator you would use the instruction:

LDA I

If you inadvertently type a pound sign in front of I, the low order byte of the *address* of I will be loaded into the accumulator. Likewise, prefacing the I with a slash loads the high order byte of the address of I into the accumulator. If you want to load the high order byte of the variable I into the accumulator the instruction:

LDA I + 1

is used.

The STA Instruction

The second most popular instruction is STA. It stores a copy of the accumulator into a memory location. To use the STA instruction follow STA with the address of the variable you wish to store the accumulator into. For example the instruction:

STA I

stores the contents of the accumulator into the low order byte of variable I. If I is an integer variable, you can store the accumulator into the high order byte using the instruction:

```
STA I+1
```

The LDA and STA instructions can be combined to move data around in memory. The SPEED/ASM MOVE subroutine copies a 2-byte integer value from one variable to another. The calling sequence for MOVE is:

```
JSR  MOVE
ADR  VAR1,VAR2
```

MOVE transfers the 2 bytes at address VAR1 to the 2 bytes at address VAR2.

This action is easily simulated using the 6502 assembly sequence:

```
LDA  VAR1
STA  VAR2
LDA  VAR1+1
STA  VAR2+1
```

This short piece of 6502 code loads the 2 bytes at addresses VAR1 and VAR1+1 and stores them at addresses VAR2 and VAR2+1. Note that the accumulator contains the value in VAR1+1 (and VAR2+1) at the end of this code sequence.

Incidentally, the LDA/STA sequence above doesn't *exactly* duplicate the operation of the MOVE subroutine. A call to MOVE requires only 7 bytes of program memory space, while the LDA/STA sequence uses up 12 bytes. Furthermore, MOVE doesn't affect the contents of the 6502 accumulator, whereas the LDA/STA sequence does. Whatever value the accumulator contained before the execution of the sequence is lost, replaced by the same value as VAR1+1. Finally, MOVE executes quite a bit more slowly than the straight LDA/STA sequence.

If speed is the overriding consideration *and* the call to MOVE is deeply buried within a loop, you should recode the JSR MOVE instruction using the LDA/STA sequence. For most purposes, however, the call to MOVE is better since it is shorter and it doesn't affect the accumulator. The speed difference is usually insignificant, unless, as mentioned, MOVE is buried deep inside nested loops. After all, few humans can tell the difference between 16 and 100 microseconds.

Although the MOVE routine should be used to copy the contents of one integer variable to another, copying a character variable is best handled with the LDA/STA sequence. If you recall the discussion in the first part of this series, I mentioned that character variables only require 1 byte of storage. Since the 6502 accumulator is 8 bits wide it can easily accommodate a character value. If you want to copy the contents of character variable CH1 to character variable CH2 you should use the code:

```
LDA  CH1
STA  CH2
```

While on the subject of character and integer variables, I should mention two functions in Basic that everyone seems to love: CHR$ and ASC. SPEED/ASM has no equivalent for these functions because they aren't needed. The 6502 treats *everything* as an integer. Only the programmer distinguishes between character and numeric data. The SPEED/ASM program segment in Listing 1 demonstrates this relationship. It prints the string C195L on the screen. Since the high order bit of all character values is set, characters are represented by the decimal values 128–255. As you can see, the interpretation of data stored in the 6502's memory space is left to the user. At any one time the value in a memory cell could contain character, numeric, Boolean or some other data representation.

The CMP Instruction

The CMP instruction is used to *compare* the accumulator to a memory location or to a constant. It affects bits within another register inside the 6502 chip called the *processor status register* (or PSR). The exact definition of these bits is unimportant for now. What is important is that these bits can be tested with a set of 6502 *branch* instructions. The CMP instruction, combined with the branch

instructions, simulates the IF..THEN ..ELSE statements found in high level languages.

To compare the accumulator to a constant value, use the pound sign or slash operator after the CMP instruction. A code segment to compare the value in CHR to the character value X is:

```
LDA  CHR
CMP  #"X"
```

This fetches the value in CHR, loads it into the accumulator and then compares the accumulator to the character X.

To compare two character variables load the accumulator with the first and compare it to the second by specifying the address of the second after the CMP instruction—for example:

```
LDA  CHR1
CMP  CHR2
```

What you do after a comparison is next on the list...

The Branch and JMP Instructions

Once a comparison is performed, a program typically wants to execute one section of code if the condition was met, and execute a second section (or just skip over the first) if the condition was not met. Transfer of control after a CMP instruction is handled by the branch instructions on the 6502.

"JMP unconditionally transfers control to the address specified in the operand field—almost exactly like the GOTO statement in Basic."

The first instruction to consider, however, is not a branch at all, but the JMP (jump) instruction. JMP *unconditionally* transfers control to the address specified in the operand field—almost exactly like the GOTO statement in Basic. The only difference is that you must specify a statement label rather than a line number in the operand field. The following program segment continually writes the character A to the screen:

```
LOOP  LDA  #"A"
      JSR  PUTC
      JMP  LOOP
```

The branch instructions transfer control to a target label if and only if a certain condition exists. Although, under certain circumstances, a branch instruction may appear almost anywhere in a program, for now you should only place branch instructions *immediately* after a CMP instruction.

The BEQ/BNE Instructions

The BEQ (branch if equal) and BNE (branch if not equal) instructions are used to test for equality and inequality. If a BEQ instruction follows a compare instruction, then control is transferred to the target label if the accumulator equals the value it was compared to. If the value in the accumulator is not equal to the value specified after the CMP instruction, then the BEQ instruction is ignored and control is transferred to the next

Listing 4. Sample program demonstrating several character variable manipulations using SPEED/ASM.

```
0800                    TTL "SPEED/ASM Sample Program
0800                ;
0800                ;
0800                ;
0800                ***********************************
0800                *                                 *
0800                *       SPEED/ASM Equates         *
0800                *                                 *
0800                ***********************************
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                ; GENERAL PURPOSE EQUATES
0800                ;
0800                ; The following variables are used
0800                ; by the SPEED/ASM package and
0800                ; shouldn't be used by the SPEED/ASM
0800                ; programmer.
0800                ;
0800                ;
0800                ;
0800                ;
0000                FORASAV  EPZ 0
0001                FORXSAV  EPZ FORASAV+1
0002                FORYSAV  EPZ FORXSAV+1
0003                FORZPG   EPZ FORYSAV+1
0005                DESTADR  EPZ FORZPG+2
0007                PTRADR   EPZ DESTADR+2
0009                ISIMMED  EPZ PTRADR+2
000A                OP       EPZ ISIMMED+1
000C                MAXLEN   EPZ OP+2
000D                VALUE    EPZ MAXLEN+1
000F                DIGIT    EPZ VALUE+2
0010                LEAD0    EPZ DIGIT+1
0011                JMPADR   EPZ LEAD0+1
0013                COUNT    EPZ JMPADR+2
0014                GOTLN    EPZ COUNT+1
0015                LINEINDX EPZ GOTLN+1
0016                SIGN     EPZ LINEINDX+1
0017                ACL      EPZ SIGN+1
0018                ACH      EPZ ACL+1
0019                XTNDL    EPZ ACH+1
001A                XTNDH    EPZ XTNDL+1
001B                AUXL     EPZ XTNDH+1
001C                AUXH     EPZ AUXL+1
0800                ;
0033                PROMPT   EPZ $33
004E                RNDL     EPZ $4E
004F                RNDH     EPZ $4F
0100                STACK    EQU $100
0200                INPUT    EQU $200
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                ;
0800                *************
0800                * CONSTANTS *
0800                *************
0800                ;
0800                ;
0800                ;
```

Listing continued.

IF THIS BRANCH IS OUT OF RANGE...	USE THIS CODE INSTEAD.
BEQ DEST ;(BFL)	BNE L1 (BTR) JMP DEST L1:
BNE DEST ;(BTR)	BEQ L2 (BFL) JMP DEST L2:
BMI DEST	BPL L3 JMP DEST L3:
BPL DEST	BMI L4 JMP DEST L4:
BCS DEST ;(BGE)	BCC L5 JMP DEST L5:
BCC DEST ;(BLT)	BCS L6 JMP DEST L6:
BVS DEST	BVC L7 JMP DEST L7:
BVC DEST	BVS L8 JMP DEST L8:

Table 1. Long branch code sequences.

instruction sequentially following the BEQ instruction in the source file. The program in Listing 2 reads a character from the keyboard and prints it until a carriage return is read.

The BNE instruction branches to the target location if the contents of the accumulator is *not* equal to the value being compared to.

The BCS (BGE)/BCC (BLT) Instructions

The BCS (branch if carry set) and BCC (branch if carry clear) instructions are used after a comparison to see if the value in the accumulator is greater than or equal to the value in the CMP operand field, or if the accumulator is less than the CMP operand. As a mnemonic aid, the LISA assembler lets you substitute BGE and BLT (branch if greater than or equal and branch if less than) for BCS and BCC.

The BVS and BVC Instructions

The BVS instruction branches to the target label if the 6502 overflow

Listing continued.

```
0800            70  ; The following symbols are constants
0800            71  ; for the values "FALSE", "TRUE", and
0800            72  ; Carriage Return (respectively).
0800            73  ;
0800            74  ; These symbols should only appear
0800            75  ; as immediate operands to a 6502
0800            76  ; instruction or in the operand field
0800            77  ; of a pseudo-opcode like BYT.
0800            78  ;
0800            79  ;
0800            80  ;
0800            81  ;
0800            82  ;
0000            83  FALSE     EQU 0
0001            84  TRUE      EQU 1
008D            85  CR        EQU $8D
0800            86  ;
0800            87  ;
0800            88  ;
0800            89  ;
0800            90  ;
0800            91  ;
0800            92  ; "IF" STATEMENT EQUATES
0800            93  ;
0800            94  ; The following symbols should only
0800            95  ; be used in the ADR pseudo-opcode
0800            96  ; following a call to the SPEED/ASM
0800            97  ; IFx routines.
0800            98  ;
0800            99  ;
0800           100  ;
00ED           101  EQ        EQU "="
00A3           102  NE        EQU "#"
00BE           103  GT        EQU ">"
00BC           104  LT        EQU "<"
BDBE           105  GE        EQU ">"|"="*256
BDBC           106  LE        EQU "<"|"="*256
0800           107  ;
0800           108  ;
0800           109  ;
0800           110  ;
0800           111  ;
0800           112  ;
0800           113  ;
0800           114  ************************
0800           115  * SPEED/ASM ENTRY POINTS *
0800           116  ************************
0800           117  ;
0800           118  ;
0800           119  ;
0800           120  ;
0800           121  ; NOTE: THE EQUATE OF PUTC MUST
0800           122  ; BE CHANGED IF YOU RELOCATE
0800           123  ; SPEED/ASM TO SOME LOCATION
0800           124  ; OTHER THAN $7800
0800           125  ;
0800           126  ;
0800           127  ;
7800           128  PUTC      EQU $7800
7803           129  GETC      EQU PUTC+3
7806           130  SAGL      EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809           131  SAPC      EQU SAGL+3          ; "  "  "  "  "  "  "  "
780C           132  HOME      EQU SAPC+3          ;HOME AND CLEAR
780F           133  READLN    EQU HOME+3
7812           134  INIT      EQU READLN+3
7815           135  FOR       EQU INIT+3
7818           136  FOR0      EQU FOR+3
781B           137  NEXT      EQU FOR0+3
781E           138  IFI       EQU NEXT+3
7821           139  IFI0      EQU IFI+3
7824           140  IFS       EQU IFI0+3
7827           141  IFS0      EQU IFS+3
782A           142  MOVE      EQU IFS0+3
782D           143  LOAD      EQU MOVE+3
7830           144  MOVS      EQU LOAD+3
7833           145  LDSTR     EQU MOVS+3
7836           146  PRINT     EQU LDSTR+3
7839           147  PRTSTR    EQU PRINT+3
783C           148  PRTINT    EQU PRTSTR+3
783F           149  RDSTR     EQU PRTINT+3
7842           150  RDINT     EQU RDSTR+3
7845           151  ONXGOTO   EQU RDINT+3
7848           152  CASE      EQU ONXGOTO+3
784B           153  CASEI     EQU CASE+3
784E           154  INSET     EQU CASEI+3
```

Listing continued.

Listing continued.

```
7851            155  NOTINSET EQU INSET+3
7854            156  ABS      EQU NOTINSET+3
7857            157  NEG      EQU ABS+3
785A            158  MUL      EQU NEG+3
785D            159  DIV      EQU MUL+3
7860            160  MOD      EQU DIV+3
7863            161  RND      EQU MOD+3
7866            162  SUBSTR   EQU RND+3
7869            163  INDEX    EQU SUBSTR+3
786C            164  LENGTH   EQU INDEX+3
786F            165  CONCAT   EQU LENGTH+3
7872            166  GETWZPG  EQU CONCAT+3        ;USED BY SPEED/ASM
7875            167  RDFP     EQU GETWZPG+3
7878            168  PRTE     EQU RDFP+3
787B            169  PRTF     EQU PRTE+3
787E            170  FADD     EQU PRTF+3
7881            171  FSUB     EQU FADD+3
7884            172  FMUL     EQU FSUB+3
7887            173  FDIV     EQU FMUL+3
788A            174  FLT      EQU FDIV+3
788D            175  FIX      EQU FLT+3
7890            176  FNEG     EQU FIX+3
7893            177  FADDTN   EQU FNEG+3
7896            178  FSUBTN   EQU FADDTN+3
7899            179  FTIMES   EQU FSUBTN+3
789C            180  FINTO    EQU FTIMES+3
789F            181  IFF      EQU FINTO+3
78A2            182  MOVFP    EQU IFF+3
0800            183  ;
0800            184  ;
0800            185  ;
0800            186  ********************************
0800            187  *                              *
0800            188  * SPEED/ASM Sample program #3:  *
0800            189  *                              *
0800            190  *   This program demonstrates  *
0800            191  * some 6502 code that is neces- *
0800            192  * sary for writing SPEED/ASM    *
0800            193  * programs.  In particular it   *
0800            194  * demonstrates how one handles  *
0800            195  * character variables in a      *
0800            196  * SPEED/ASM program.            *
0800            197  *                              *
0800            198  * Assembler: LISA 2.5          *
0800            199  * Requires SPEED/ASM package.  *
0800            200  * Randall Hyde                 *
0800            201  *                              *
0800            202  ********************************
0800            203  ;
0800            204  ;
0800            205  ;
0800            206  ;
0800            207  ;
FF69            208  EXIT     EQU $FF69           ;Entry point to Apple monitor
0800            209  ;
0800            210  ;
0800            211  ;
0800 20 12 78   212           JSR INIT            ;Initialize SPEED/ASM
0803            213  ;
0803            214  ;
0803 20 0C 78   215           JSR HOME            ;Clear the screen
0806 20 36 78   216           JSR PRINT
0809 8D 8D B6   217           BYT CR,CR,"6502--SPEED/ASM test program"
080C B5 B0 B2
080F AD AD D3
0812 D0 C5 C5
0815 C4 AF C1
0818 D3 CD A0
081B F4 E5 F3
081E F4 A0 F0
0821 F2 EF E7
0824 F2 E1 ED
0827 8D 8D      218           BYT CR,CR
0829 C1 D3 C3   219           BYT "ASCII Character Set table",CR
082C C9 C9 A0
082F C3 E8 E1
0832 F2 E1 E3
0835 F4 E5 F2
0838 A0 D3 E5
083B F4 A0 F4
083E E1 E2 BC
0841 E5 8D
0843 AD AD AD   220           BYT "--------------------------"
0846 AD AD AD
0849 AD AD AD
084C AD AD AD
084F AD AD AD
0852 AD AD AD
0855 AD AD AD
0858 AD AD AD
```

Listing continued.

flag is set. The BVC instruction branches to the specified location if the 6502 overflow flag is clear.

The overflow flag is set and cleared by arithmetic operations in the 6502. Certain SPEED/ASM routines also pass an error status back in the overflow flag. The BVS or BVC instruction can be used after a call to such a routine to test for an error.

The addition and subtraction instructions set or clear the overflow flag depending on the status of the result produced. If an overflow occurred, then the overflow flag is set; otherwise it is cleared.

The BMI and BPL Instructions

Unlike the branches discussed up to this point, the BMI (branch if minus) and BPL (branch if plus) instructions aren't usually executed after a CMP instruction. They should be used after an LDA instruction. BMI specifies a branch if bit 7 of the value in the accumulator is 1; BPL branches if bit 7 of the accumulator is 0.

To test an integer quantity to see if it is positive or negative use the following code:

```
LDA   INT+1
BMI   ISNEG
```

or

```
LDA   INT+1
BPL   ISPOS
```

Note that you must test the high order byte of the integer quantity. Bit number 7 of the high order byte is the sign bit for an integer quantity in SPEED/ASM.

Suppose you wanted to set an integer variable negative. As discussed last time, SPEED/ASM supports two routines for changing the sign of an integer variable: ABS and NEG. ABS takes the absolute value of an integer variable. After execution the specified variable will contain a positive value.

The NEG routine negates (that is, changes the sign of) the specified variable. If the variable originally contained a positive value, the result will be negative; if the variable contained a negative value, the result will be positive.

Listing continued.

```
085B AD
085C 8D 8D      221          BYT CR,CR
085E C3 B8 E1   222          BYT "Character   ASCII Code",CR
0861 F2 E1 E3
0864 F4 E5 F2
0867 A0 A0 A0
086A C1 D3 C3
086D C9 C9 A0
0870 C3 EF E4
0873 E5 8D
0875 AD AD AD   223          BYT "---------   ---------",CR,CR,0
0878 AD AD AD
087B AD AD AD
087E A0 A0 A0
0881 AD AD AD
0884 AD AD AD
0887 AD AD AD
088A AD 8D 8D
088D 00
088E            224  ;
088E            225  ; Init CH to space
088E            226  ;
088E A9 A0      227          LDA #" "
0890 8D DA 09   228          STA CH
0893            229  ;
0893            230  ; Loop to print the ASCII values
0893            231  ;
0893 AD DA 09   232  PRTASC  LDA CH              ;Emulate the BASIC
0896 8D DB 09   233          STA I               ;statement "I=ASC(CH$)"
0899 A9 00      234          LDA #0
089B 8D DC 09   235          STA I+1
089E            236  ;
089E 20 36 78   237          JSR PRINT
08A1 A0 A0 A0   238          BYT "    ",0
08A4 A0 00
08A6 AD DA 09   239          LDA CH              ;Print the character
08A9 20 00 78   240          JSR PUTC
08AC 20 36 78   241          JSR PRINT
08AF A0 A0 A0   242          BYT "   ---   ",0
08B2 AD AD AD
08B5 AD AD A0
08B8 A0 00
08BA 20 3C 78   243          JSR PRTINT          ;Print the ASCII code
08BD DB 09      244          ADR I
08BF            245  ;
08BF            246  ;
08BF            247  ; Print a carriage return to
08BF            248  ; move to the next line.
08BF            249  ;
08BF A9 8D      250          LDA #CR
08C1 20 00 78   251          JSR PUTC
08C4            252  ;
08C4            253  ; Add one to the CH value
08C4            254  ;
08C4 EE DA 09   255          INC CH              ;6502 Instruction to increment a byte
08C7            256  ;
08C7            257  ; See if we're done
08C7            258  ;
08C7 AD DA 09   259          LDA CH
08CA C9 FF      260          CMP #$FF            ;Last ASCII value plus one.
08CC 90 C5      261          BLT PRTASC
08CE            262  ;
08CE            263  ;
08CE            264  ;
08CE            265  ; Read characters from the keyboard
08CE            266  ; until a "#" character is pressed.
08CE            267  ;
08CE 20 36 78   268          JSR PRINT
08D1 8D 8D C5   269          BYT CR,CR,"Enter any text terminated",CR
08D4 EE F4 E5
08D7 F2 A0 E1
08DA EE F9 A0
08DD F4 E5 F8
08E0 F4 A0 F4
08E3 E5 F2 ED
08E6 E9 EE E1
08E9 F4 E5 E4
08EC 8D
08ED E2 F9 A0   270          BYT "by a pound sign (""#"")",CR,CR
08F0 E1 A0 F0
08F3 EF F5 EE
08F6 E4 A0 F3
08F9 E9 E7 EE
08FC A0 A8 A2
08FF A3 A2 A9
0902 8D 8D
0904 BA 00      271          BYT ":",0
0906 20 03 78   272  READLOOP JSR GETC           ;Read the character
0909 20 00 78   273          JSR PUTC            ;and print it.
```

Listing continued.

What happens if you want to ensure that a variable contains a negative value regardless of its original sign? This problem could easily be handled using two SPEED/ASM calls:

```
JSR   ABS
ADR   VARIABLE
JSR   NEG
ADR   VARIABLE
```

The first SPEED/ASM call, to ABS, makes sure that VARIABLE contains a positive number. The second JSR, to NEG, negates this positive number to yield a negative number. Although this sequence is clever, a better way to accomplish the task is the following:

```
        LDA   VARIABLE+1
        BMI   ISNEGTV
        JSR   NEG
        ADR   VARIABLE
ISNEGTV:
```

This code checks VARIABLE to see if it is negative (by loading the high order byte of VARIABLE and branching if it is negative), and branches around the call to NEG if VARIABLE is already negative. If VARIABLE is positive NEG negates it.

The BTR and BFL Instructions

BTR (branch if true) and BFL (branch if false) are actually synonyms for the BNE and BEQ instructions (respectively). LISA 2.5 emits the same object code for BTR and BNE; likewise the same opcode is emitted for BFL as for BEQ.

The SPEED/ASM package uses the value 0 to represent false and 1 to represent true. After a value is loaded into the accumulator, the BTR and BFL instructions can test whether or not the value is zero (false). If so, BFL branches to the specified location. If not, BTR transfers control to another specified location. BFL and BTR are used extensively by SPEED/ASM IF routines (to be discussed in a future installment).

There is one problem with the branch instructions that I haven't mentioned yet: the branch range is somewhat limited. A 6502 branch instruction uses a special addressing

Listing continued.

```
090C C9 A3      274          CMP #"#"           ;Check for "#" character.
090E D0 F6      275          BNE READLOOP       ;If not "#", repeat.
0910            276  ;
0910            277  ;
0910            278  ; The following code prompts the
0910            279  ; person at the keyboard to enter
0910            280  ; "Y" or "N".  If something else
0910            281  ; is entered then the code is
0910            282  ; repeated.
0910            283  ;
0910 20 36 78   284  GETYORN  JSR PRINT
0913 8D 8D 8D   285          BYT CR,CR,CR
0916 C5 EE F4   286          BYT "Enter Yes or No (Y/N): ",0
0919 E5 F2 A0
091C D9 E5 F3
091F A0 EF F2
0922 A0 CE EF
0925 A0 A8 D9
0928 AF CE A9
092B 8D         092B:BA A0 00
092E 20 03 78   287          JSR GETC
0931 C9 D9      288          CMP #"Y"
0933 F0 04      289          BEQ GOODANS
0935 C9 CE      290          CMP #"N"
0937 D0 D7      291          BNE GETYORN
0939            292  ;
0939            293  GOODANS:
0939            294  ;
0939            295  ;
0939            296  ;
0939            297  ; This code prompts the user to enter
0939            298  ; a decimal value.  Once the value
0939            299  ; is entered the ASCII character
0939            300  ; corresponding to that code is
0939            301  ; printed on the screen.
0939            302  ;
0939            303  ;  Entering zero terminates the
0939            304  ; loop.
0939            305  ;
0939 20 36 78   306  PRTCHR   JSR PRINT
093C 8D         307          BYT CR
093D 8D C5 EE   308          BYT CR,"Enter a decimal value in the"
0940 F4 E5 F2
0943 A0 E1 A0
0946 E4 E5 E3
0949 E9 ED E1
094C EC A0 F6
094F E1 EC F5
0952 E5 A0 E9
0955 EE A0 F4
0958 E8 E5
095A 8D F2 E1   309          BYT CR,"range 0..255: ",0
095D EE E7 E5
0960 A0 B0 AE
0963 AE B2 B5
0966 B5 BA A0
0969 00
096A 20 42 78   310          JSR RDINT
096D DB 09      311          ADR I
096F 70 C8      312          BVS PRTCHR         ;If an error occured.
0971            313  ;
0971            314  ; Force integer value into the
0971            315  ; range 0..255 by zeroing the high
0971            316  ; order byte.
0971            317  ;
0971 A9 00      318          LDA #0
0973 8D DC 09   319          STA I+1
0976            320  ;
0976            321  ; Print the decimal value (in case
0976            322  ; they entered too large a number.
0976            323  ;
0976 20 36 78   324          JSR PRINT
0979 8D C4 E5   325          BYT CR,"Decimal value: ",0
097C E3 E9 ED
097F E1 EC A0
0982 F6 E1 EC
0985 F5 E5 BA
0988 E8         0988:A0 00
098A 20 3C 78   326          JSR PRTINT
098D DB 09      327          ADR I
098F 20 36 78   328          JSR PRINT
0992 A0 A0 C1   329          BYT "  ASCII Character: "",0
0995 D3 C3 C9
0998 C9 A0 C3
099B E8 E1 F2
099E E1 E3 F4
09A1 E5 F2 BA
09A4 A0 A2 00
09A7 AD DB 09   330          LDA I                          Listing continued.
```

mode called the *relative addressing mode*, in which the branch opcode is followed by a single address byte. Such a 2-byte instruction saves some memory (normally 3 bytes would be required) but branches can jump only to a location within a 256-byte range centered at the instruction following the branch instruction. Therefore, a branch instruction can branch 129 bytes forward (from the beginning of the branch instruction) or 126 bytes backwards.

Normally branches occur within this range and there's no problem. Occasionally, however, a program needs to branch to a location outside this +129/−126 byte range. To accomplish this use the opposite-type branch to jump around a 6502 JMP instruction to the intended address. For example, if you want to branch if true to location ISTRUE but the assembler gives you a branch-out-of-range error, substitute the following code:

```
        BFL   ISNTTRUE
        JMP   TRUE
ISNTTRUE:
```

Table 1 lists the instruction sequences to use if a branch-out-of-range error occurs.

Working with Character Variables Through SPEED/ASM

This subtitle may seem somewhat of a misnomer, since the SPEED/ASM package provides absolutely no character handling routines. All character handling must be performed by pure 6502 code. However, character variables are still, in essence, manipulated.

Declaring character variables was described in part 1 of this series. As a review, to reserve space for a character variable you specify the name of the variable followed by a pseudo opcode that reserves at least 1 byte of storage. I usually use the DFS instruction to reserve a single byte. For example, to reserve 1 byte for a variable named CHAR use the definition:

```
CHAR DFS 1
```

This statement instructs LISA to reserve 1 byte at the current location for the variable CHAR.

Listing continued.

```
09AA 20 00 78   331          JSR PUTC
09AD A9 A2      332          LDA #"""
09AF 20 00 78   333          JSR PUTC
09B2            334  ;
09B2            335  ; Check the L.O. byte of I to
09B2            336  ; see if it is zero.
09B2            337  ;
09B2            338  ; (On the 6502 BEQ and BNE can
09B2            339  ;  be used immediately after a
09B2            340  ;  LDA instruction to see if zero
09B2            341  ;  was loaded into the accumulator
09B2            342  ;  or not)
09B2            343  ;
09B2 AD DB 09   344          LDA I
09B5 D0 82      345          BNE PRTCHR
09B7            346  ;
09B7            347  ;
09B7            348  ; Quit and return to the Apple monitor
09B7            349  ;
09B7 20 36 78   350          JSR PRINT
09BA 8D 8D 8D   351          BYT CR,CR,CR
09BD C5 EE E4   352          BYT "End of sample program #3",CR,0
09C0 A0 EF E6
09C3 A0 F3 E1
09C6 ED F0 EC
09C9 E5 A0 F0
09CC F2 EF E7
09CF F2 E1 ED
09D2 A0 A3 B3
09D5 8D 00
09D7            353  ;
09D7            354  ;
09D7 4C 69 FF   355          JMP EXIT            ;Return to Apple Monitor
09DA            356  ;
09DA            357  CH      DFS 1               ;Char
09DB            358  I       DFS 2               ;Integer
09DD            359          END
***** END OF ASSEMBLY
!
SPEED/ASM  (C) 1981, LAZER SYSTEMS

DOES YOUR APPLE SUPPORT LOWER CASE
```

Listing continued.

As is the case with integer variables, character variables must be defined at a point in your code where they will not get executed *as* code. The best place to put variable definitions is after the JMP EXIT that terminates the program.

Initializing a character variable with a constant was briefly discussed earlier in this article. To do so load the accumulator with the desired constant (using the immediate addressing mode—that is, preface the character constant with a pound sign) and store the accumulator into the character variable. For example, to load CHAR with the character constant ? execute the code:

```
LDA   #"?"
STA   CHAR
```

Incidentally, any value can be loaded into CHAR. It doesn't have to be a character constant. You could initialize CHAR to contain the carriage

return character by using the code:

```
LDA   #CR      ;CR is defined in
               ;SPEED/ASM.EQUATES
STA   CHAR
```

Any other numeric or symbol value could be loaded into CHAR in a similar fashion.

Copying one character variable to another (also previously mentioned) is a trivial exercise. Load the accumulator with the source variable and then store the accumulator into the destination variable. To copy CHAR1 into CHAR2 you could use the code:

```
LDA   CHAR1
STA   CHAR2
```

I mentioned before that the CHR$ and ASC functions are handled by simple loads and stores. In certain cases this is true. But if you're translating Applesoft code into SPEED/ASM you will run into a few problems. Most noticeably, Applesoft uses a version of the ASCII character set wherein the ASCII codes occupy the range 0–127. To truly implement the CHR$ and ASC functions you should use the code sequences in Listing 3. The AND and ORA are special 6502 logical instructions that I'll describe in a future article. For now just copy this code sequence verbatim any time you want to simulate a CHR$ or ASC function.

Comparing character variables is simply a matter of loading one character value into the accumulator using the LDA instruction and comparing it to another using the CMP instruction. This month's sample program (Listing 4) demonstrates several character variable manipulations that should help answer any questions you have about character handling.

Notes of a Commercial Interest

The LISA assembler and the SPEED/ASM programming package are available from your local computer store and many mail order software houses. If you cannot locate a copy of either of these packages, you can order directly from Sierra On-Line Inc., 36575 Mudge Ranch Road, Coarsegold, CA 93614. The LISA assembler, SPEED/ASM, and Datamost's *Using 6502 Assembly Language* book are all available in a special LISA Educational Package available where LISA is sold. ■

Listing continued.

```
DISPLAY? (Y/N):
SPEED/ASM  (C) 1981, LAZER SYSTEMS

DOES YOUR APPLE SUPPORT LOWER CASE
DISPLAY? (Y/N):

   6502—SPEED/ASM test program

   ASCII Character Set table

Character   ASCII Code

        ——— 160
  !     ——— 161
  "     ——— 162
  #     ——— 163
  $     ——— 164
  %     ——— 165
  &     ——— 166
  '     ——— 167
  (     ——— 168
  )     ——— 169
  *     ——— 170
  +     ——— 171
  ,     ——— 172
  -     ——— 173
  .     ——— 174
  /     ——— 175
  0     ——— 176
  1     ——— 177
  2     ——— 178
  3     ——— 179
  4     ——— 180
  5     ——— 181
  6     ——— 182
  7     ——— 183
  8     ——— 184
  9     ——— 185
  :     ——— 186
  ;     ——— 187
  <     ——— 188
  =     ——— 189
  >     ——— 190
  ?     ——— 191
  @     ——— 192
  A     ——— 193
  B     ——— 194
  C     ——— 195
  D     ——— 196
  E     ——— 197
  F     ——— 198
  G     ——— 199
  H     ——— 200
  I     ——— 201
  J     ——— 202
  K     ——— 203
  L     ——— 204
  M     ——— 205
  N     ——— 206
  O     ——— 207
  P     ——— 208
  Q     ——— 209
  R     ——— 210
  S     ——— 211
  T     ——— 212
  U     ——— 213
  V     ——— 214
  W     ——— 215
  X     ——— 216
  Y     ——— 217
  Z     ——— 218
  [     ——— 219
  \     ——— 220
  ]     ——— 221
  ^     ——— 222
  _     ——— 223
  `     ——— 224
  a     ——— 225
  b     ——— 226
  c     ——— 227
  d     ——— 228
  e     ——— 229
  f     ——— 230
  g     ——— 231
  h     ——— 232
  i     ——— 233
  j     ——— 234
  k     ——— 235
  l     ——— 236
  m     ——— 237
  n     ——— 238
  o     ——— 239
  p     ——— 240
  q     ——— 241
  r     ——— 242
  s     ——— 243
  t     ——— 244
  u     ——— 245
  v     ——— 246
  w     ——— 247
  x     ——— 248
  y     ——— 249
  z     ——— 250
  {     ——— 251
  |     ——— 252
  }     ——— 253
  ~     ——— 254

Enter any text terminated
by a pound sign ("#")

:This is a test of SPEED/ASM
 #

Enter Yes or No (Y/N):

Enter a decimal value in the
range 0..255: 250

Decimal value: 250  ASCII Character: "z"

Enter a decimal value in the
range 0..255: 145

Decimal value: 145  ASCII Character: ""

Enter a decimal value in the
range 0..255: 174

Decimal value: 174  ASCII Character: "."

Enter a decimal value in the
range 0..255: 199

Decimal value: 199  ASCII Character: "G"

Enter a decimal value in the
range 0..255: 232

Decimal value: 232  ASCII Character: "h"

Enter a decimal value in the
range 0..255: 23456

Decimal value: 160  ASCII Character: " "

Enter a decimal value in the
range 0..255: 0

Decimal value: 0  ASCII Character: " "
```

The Assembly Advantage

by Randy Hyde

Control Structures

In the previous three installments of this column I've described how to define and use SPEED/ASM variables and how to perform the standard integer arithmetic operations, and I've discussed some 6502 code necessary to write complete SPEED/ASM programs. With most of the basics behind us, its' time to begin a consideration of SPEED/ASM's control structures.

The FOR...NEXT Loop

SPEED/ASM supports two variations on Basic's FOR...NEXT loop: FOR and FOR0. The FOR subroutine is a generalized FOR loop that allows variable initial/ending values and a stepsize. The FOR0 routine is a specialized version of FOR that assumes the initial and ending values are constants and that the stepsize is one. Since the FOR0 loop is used better than 90 percent of the time, I will describe it first.

The FOR0 loop uses the structure:

```
        JSR   FOR0
        ADR   IVAR,STARTVAL,ENDVAL

; Body of loop

        JSR   NEXT
```

IVAR must be the name of a SPEED/ASM integer variable; STARTVAL and ENDVAL *must* be integer constants. If you specify a SPEED/ASM variable name for STARTVAL or ENDVAL, SPEED/ASM will use the *address* of the variable, not the contents of the variable, as the initial or final value.

Since most FOR loops take the form:

```
FOR I = 1 TO 10
```

the FOR0 routine turns out to be quite adequate for most applications. See Listing 1 for a sample Basic FOR...NEXT routine and its SPEED/ASM equivalent.

As in Basic, it is illegal to jump into the middle of a FOR...NEXT loop in SPEED/ASM. Unlike Basic, SPEED/ASM isn't nice enough to tell you that you've executed a NEXT without a

```
10 FOR I = 1 TO 20
20 PRINT "I = ";I
30 NEXT I
40 END
```

Listing 1a. Sample Basic FOR...NEXT routine.

Address correspondence to Randy Hyde, 925 Lorna St., Corona, CA 91720.

```
EXIT    EQU   $FF69
;
        JSR   INIT      ;Always before running a SPEED/ASM PGM.
        JSR   FOR0
        ADR   I,1,20
;
;Body of loop
;
        JSR   PRINT
        BYT   "I = ",0
        JSR   PRINT
        ADR   I
        LDA   #CR       ;Print the return at the end
        JSR   PUTC      ;of the line.
;
        JSR   NEXT
        JMP   EXIT
;
;Variable declaration(s)
;
I       ADR   0
        END
```

Listing 1b. SPEED/ASM equivalent.

```
        JSR   FOR0
        ADR   I,1,10
;
;Body of loop
;
        JMP   EXITLP
;
        JSR   NEXT
        JMP   DIDNTXIT
;
;Exit condition at this point.
;
EXITLP  PLA
        PLA
        PLA
        PLA
        PLA
        PLA
        PLA
        PLA
;
DIDNTXIT:
```

Listing 2. Popping data off a stack.

matching FOR. Instead, the system simply hangs up (or begins doing bizarre things to the screen or accessing your peripheral devices). Therefore, you should always make sure that there are no jumps into the range of a FOR loop. For example, the following is definitely forbidden:

```
JMP   ENTERFOR ;Can't do!
      JSR   FOR0
      ADR   I,1,10
;
ENTERFOR:
;
      JSR   NEXT
;
```

It is in equally bad taste to jump outside the range of a FOR...NEXT loop from *within* a FOR loop. Basic also disallows this, but is much more forgiving. The latter won't catch the error until you overflow the FOR...NEXT stack (by doing it too many times). SPEED/ASM, on the other hand, explodes spectacularly the next time you execute an RTS or otherwise attempt to access the top of the stack.

The reason a jump into or out of a FOR...NEXT loop causes problems is that the FOR subroutine pushes data onto the top of the stack and leaves it there. This data is popped off and processed by the NEXT subroutine later on. Obviously, if you call the FOR routine from within a subroutine and then execute an RTS without first completing the loop by calling the NEXT routine, the 6502 will attempt to use the data pushed onto the stack by the FOR routine as the return address. Typically this will not return you to the spot you're interested in.

For those of you who regularly use the POP command in Applesoft, yes, you can pop this data off the stack and prematurely exit a FOR...NEXT loop. The SPEED/ASM FOR...NEXT loop pushes 8 bytes of data onto the stack so you can repair the stack by popping 8 bytes off. This is accomplished by executing eight PLA instructions in a row. See Listing 2.

The FOR subroutine is similar to the FOR0 subroutine. The major difference is that the FOR subroutine allows variable starting and ending values and a variable stepsize. The exact syntax for the FOR subroutine is:

```
JSR   FOR
ADR   <index var>,<start var>, <end var>,
<step var>
```

where index var is the name of the variable used as the loop index, start var is the name of the variable containing the starting value, end var is the name of the variable containing the ending value, and step var is the name of the variable containing the stepsize. To simulate the Basic statements:

```
10 FOR I = J TO K STEP STP
20 NEXT I
```

you would use the SPEED/ASM code:

```
JSR   FOR
ADR   I,J,K,STP

JSR   NEXT
```

Note that the FOR loop accepts only variable names; constants are not allowed. If either the starting or ending value must be a variable, or your loop requires a stepsize, then you must use a FOR loop and all the values must be specified as variables. For example, consider the Basic loop:

```
10 FOR I = J TO 100
20 NEXT I
```

Because the starting value (J) is specified as a variable, the FOR0 routine cannot be used. The FOR routine, however, requires that the starting, ending and stepsize values all be specified as variables. Therefore, to convert the statement above into SPEED/ASM code you will need to create two "dummy" variables: one to hold the ending value constant (100) and one to hold the stepsize constant (1). Use the code in Listing 3.

Note that the constants/variables C100 and C1 are sandwiched between a JMP instruction and the beginning of the loop. Since these values are static (they do not change) they should be incorporated into the code instead of standing at the end of the program with the variables. Although they are constant values they still must not be executed as 6502 instructions. Hence, a JMP instruction was executed to skip past the constants.

The FOR subroutine is very powerful, even though it may be somewhat cumbersome to use if the starting, ending and/or stepsize values are constants instead of variables. The inconvenience is actually minimal since most of the time the FOR0 subroutine is used instead of the FOR subroutine.

```
      JMP   STRTLP
C100  ADR   100
C1    ADR   1
;
STRTLP JSR  FOR
      ADR   I,J,C100,C1
;
;
;Body of loop
;
      JSR   NEXT
```

Listing 3. A FOR routine using "dummy" variables.

```
 1 J = 2
 5 I = 1
10 IF I> = 10 THEN GOTO 20
11 I = I*J
12 GOTO 10
20 END
```

Listing 4a. Sample Basic IF routine incorporating a GOTO statement.

```
EXIT  EQU   $FF69
;
      JSR   INIT
      JSR   LOAD
      ADR   2,J
      JSR   LOAD
      ADR   1,I
L10   JSR   IFI0
      ADR   I,GE,10
      BTR   L20
      JSR   MUL
      ADR   I,J,I
      JMP   L10
;
L20   JMP   EXIT
      END
```

Listing 4b. SPEED/ASM equivalent using BTR.

The SPEED/ASM IFI And IFI0 Routines

SPEED/ASM uses two routines to compare integer values: IFI and IFI0. IFI compares two integer variables; IFI0 compares an integer variable to an integer constant. Six types of comparison are possible. The IFI and IFI0 routines return "true" if a com-

Listing 6. IF and FOR Demo.

```
0800                1           TTL "Listing  6 : IF and FOR Demo"
0800                2  ;
0800                3  ;
0800                4  ;
0800                5  ***********************************
0800                6  *                                 *
0800                7  *       SPEED/ASM Equates         *
0800                8  *                                 *
0800                9  ***********************************
0800               10  ;
0800               11  ;
0800               12  ;
0800               13  ;
0800               14  ;
0800               15  ;
0800               16  ;
0800               17  ;
0800               18  *************
0800               19  * CONSTANTS *
0800               20  *************
0800               21  ;
0800               22  ;
0800               23  ;
0800               24  ; The following symbols are constants
0800               25  ; for the values "FALSE", "TRUE", and
0800               26  ; Carriage Return (respectively).
0800               27  ;
0800               28  ; These symbols should only appear
0800               29  ; as immediate operands to a 6502
0800               30  ; instruction or in the operand field
0800               31  ; of a pseudo-opcode like BYT.
0800               32  ;
0800               33  ;
0800               34  ;
0800               35  ;
0800               36  ;
0000               37  FALSE    EQU 0
0001               38  TRUE     EQU 1
008D               39  CR       EQU $8D
0087               40  BELL     EQU $87
0800               41  ;
0800               42  ;
0800               43  ;
0800               44  ;
0800               45  ;
0800               46  ;
0800               47  ; "IF" STATEMENT EQUATES
0800               48  ;
0800               49  ; The following symbols should only
0800               50  ; be used in the ADR pseudo-opcode
0800               51  ; following a call to the SPEED/ASM
0800               52  ; IFx routines.
0800               53  ;
0800               54  ;
0800               55  ;
00BD               56  EQ       EQU "="
00A3               57  NE       EQU "#"
00BE               58  GT       EQU ">"
00BC               59  LT       EQU "<"
BDBE               60  GE       EQU ">"|"="*256
BDBC               61  LE       EQU "<"|"="*256
0800               62  ;
0800               63  ;
0800               64  ;
0800               65  ;
0800               66  ;
0800               67  ;
0800               68  ;
0800               69  ***************************
0800               70  * SPEED/ASM ENTRY POINTS *
0800               71  ***************************
0800               72  ;
0800               73  ;
0800               74  ;
0800               75  ;
0800               76  ; NOTE: THE EQUATE OF PUTC MUST
0800               77  ; BE CHANGED IF YOU RELOCATE
0800               78  ; SPEED/ASM TO SOME LOCATION
0800               79  ; OTHER THAN $7800
0800               80  ;
0800               81  ;
0800               82  ;
7800               83  PUTC     EQU $7800
7803               84  GETC     EQU PUTC+3
7806               85  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809               86  SAPC     EQU SAGL+3          ; "  "  "  "  "  "  "  "
780C               87  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F               88  READLN   EQU HOME+3
7812               89  INIT     EQU READLN+3
7815               90  FOR      EQU INIT+3
7818               91  FOR0     EQU FOR+3
781B               92  NEXT     EQU FOR0+3
781E               93  IFI      EQU NEXT+3
7821               94  IFI0     EQU IFI+3
```

Listing continued.

parison holds, "false" if it does not.

You can call the IFI0 routine with the format:

```
JSR   IFI0
ADR   <var1>,<op>,<value>
```

where var1 is a SPEED/ASM integer variable, value is any integer constant, and op is the operator specifying which comparison to perform. The latter may be: EQ—test for equality; NE—test for inequality; LT—test for less than; LE—test for less than or equal to; GT—test for greater than; or GE—test for greater than or equal to. For example, to see if IVAR is less than 4376 use the statement:

```
JSR   IFI0
ADR   IVAR,LT,4376
```

The EQ, NE, GT, GE, LT and LE symbols are defined for you in SPEED/ASM Equates, Listing 7. Refer there for the values corresponding to these symbols.

The IFI0 and IFI routines return "true" or "false" in the 6502 accumulator (where true = 1 and false = 0), and set the 6502 Z bit in the P register so that the BTR and BFL (branch-if-true and branch-if-false) instructions can be used immediately after the call to IFI0 or IFI. To emulate the Basic IF statement you need only add a branch statement to the IFI0 or IFI statement to make it fully functional. Refer to Listing 4 for an example.

This relatively complete SPEED/ASM program (it still needs the

```
IF I < >10 THEN J = - I
```

Listing 5a. Sample Basic IF routine.

```
        JSR   IFI0
        ADR   I,NE,10
        BFL   INE10
        JSR   MOVE
        ADR   I,J
        JSR   NEG
        ADR   J
;
INE10:
```

Listing 5b. SPEED/ASM equivalent using BFL.

Listing continued.

```
7824              95  IFS       EQU IFI0+3
7827              96  IFS0      EQU IFS+3
782A              97  MOVE      EQU IFS0+3
782D              98  LOAD      EQU MOVE+3
7830              99  MOVS      EQU LOAD+3
7833             100  LDSTR     EQU MOVS+3
7836             101  PRINT     EQU LDSTR+3
7839             102  PRTSTR    EQU PRINT+3
783C             103  PRTINT    EQU PRTSTR+3
783F             104  RDSTR     EQU PRTINT+3
7842             105  RDINT     EQU RDSTR+3
7845             106  CNXGOTO   EQU RDINT+3
7848             107  CASE      EQU CNXGOTO+3
784B             108  CASEI     EQU CASE+3
784E             109  INSET     EQU CASEI+3
7851             110  NOTINSET  EQU INSET+3
7854             111  ABS       EQU NOTINSET+3
7857             112  NEG       EQU ABS+3
785A             113  MUL       EQU NEG+3
785D             114  DIV       EQU MUL+3
7860             115  MOD       EQU DIV+3
0800             116  ;
0800             117  ;
0800             118  ;
0800             119  ; Apple DOS equates:
0800             120  ;
03EA             121  INITDOS   EQU $3EA            ;DOS init routine.
0084             122  CTLD      EQU $84             ;Control-D character
0800             123  ;
0800             124  ;
0800             125  ; Apple Monitor equates
0800             126  ;
FF59             127  EXIT      EQU $FF59
0800             128  ;
0800             129  ;
0800             130  ;
0800             131  ;
0800             132  ;
0800             133  ;
0800             134  ***********************************
0800             135  *                                 *
0800             136  *  IF and FOR demonstration pgm.  *
0800             137  *                                 *
0800             138  *    This program creates a text  *
0800             139  *  file on the disk (it's named   *
0800             140  *  "NUMBERS") and writes out a    *
0800             141  *  series of numbers to the disk. *
0800             142  *  The second half of this pro-   *
0800             143  *  gram reads the data back in    *
0800             144  *  (in various formats) and dis-  *
0800             145  *  plays the input info on the    *
0800             146  *  screen.                        *
0800             147  *                                 *
0800             148  ***********************************
0800             149  ;
0800             150  ;
0800             151  ;
0800             152  ;
0800             153  ;
0800             154  ; Initialization section:
0800             155  ;
0800             156  ;
0800             157  ;
0800 20 12 78    158  SA.PGM.4 JSR INIT             ;Always do this first!
0803             159  ;
0803             160  ;
0803             161  ; The following call activates Apple DOS by
0803             162  ; "reconnecting" the input/output
0803             163  ; hooks at locations $36..$39  (See
0803             164  ; "Beneath Apple DOS" page 5-5).
0803             165  ;
0803 20 EA 03    166          JSR INITDOS
0806             167  ;
0806             168  ;
0806             169  ; Emulate the BASIC statement:
0806             170  ;
0806             171  ;   PRINT CHR$(4);"NOMON O,I,C"
0806             172  ;
0806 20 36 78    173          JSR PRINT
0809 8D 84 CE    174          BYT CR,CTLD,"NOMON O,I,C",CR,0
080C CF CD CF
080F CE A0 CF
0812 AC C9 AC
0815 C3 8D 00
0818             175  ;
0818             176  ;
0818             177  ;
0818             178  ;
0818             179  ***********************************
0818             180  ;
0818             181  ; File creation section:
0818             182  ;
0818             183  ;
0818             184  ;
0818             185  ; Open the file "NUMBERS" in the typical
0818             186  ; Apple DOS fashion.  Note that a
```

Listing continued.

SPEED/ASM equates at the top) exactly duplicates the Basic program. Note that the BTR (branch-if-true) instruction branches if the condition I> = 10 is true.

Sometimes you may want to execute an instruction other than a GOTO if an expression is true. For example, both Integer Basic and Applesoft Basic allow IF statements of the form IF <cond> THEN <statement>. This is easily simulated in SPEED/ASM by using the BFL (branch-if-false) instruction to branch around the statement you wish to execute. Listing 5 illustrates SPEED/ASM code of this type and a Basic equivalent.

In addition to setting the 6502 Z flag so that the BTR and BFL instructions can be used after an IFI0 instruction, the IFI0 routine returns "false" or "true" (0 or 1) in the 6502 accumulator register. This feature can be applied to several situations. Consider the Basic statement I = J< = 10. This assignment stores 0 into I if J is not less than or equal to ten; it stores 1 into I if J is less than or equal to ten. This action can easily be accomplished using the SPEED/ASM statements that follow:

```
JSR   IFI0
ADR   J,LE,10
STA   I
LDA   /0
STA   I+1
```

Don't forget that the high order byte of I must be set to 0. The LDA /0 and STA I+1 statements take care of this problem.

You use the IFI routine to compare two integer *variables*. IFI's syntax is almost identical to that of IFI0. The only difference is that you specify a second SPEED/ASM variable instead of a numeric constant. The format for the IFI routine is:

```
JSR   IFI
ADR   <var1>,<op>,<var2>
```

Interfacing to Apple DOS

While I could go on discussing how the FOR and IF subroutines work, the best way to explain their use is

through some concrete examples. Listing 6 uses FOR and IF to demonstrate how to create and access text files under Apple DOS.

As in Basic, in order to interface to Apple DOS you must print a control-D followed by a DOS command. Normally, Apple DOS only allows text files to be accessed from a running Basic program. Since SPEED/ASM is definitely not Basic, we must trick DOS into thinking that a Basic program is running. This is easily accomplished by storing the value $80 into locations $75 and $D9 in the Apple's 0 page memory space. This feat is accomplished using the 6502 code:

```
LDA   #$80
STA   $75
STA   $D9
```

Storing $80 into location $D9 tells DOS that an Integer Basic program is running; storing $80 into location $75 informs DOS that an Applesoft program is running. Whenever DOS receives a command to manipulate a text file it looks to see which Basic is currently active and then checks the appropriate 0 page location to determine whether or not the Basic is running. If location $75 contains $FF and Applesoft is active or if location $D9 is positive (less than $80) and Integer Basic is active, then a NOT DIRECT COMMAND error is issued and everything stops. Since many SPEED/ASM programs will be running under the control of Apple DOS, the SPEED/ASM INIT routine automatically stores $80 into locations $D9 and $75 for you. While this is ideal for Apple DOS users, if attempting to run your SPEED/ASM program under a different operating system (like ANIX, OS/A, or APEX) you should be aware of the fact that SPEED/ASM manipulates these two locations on power-up.

This month's demonstration program is quite simple. It writes out a sequence of numbers to a random access file and then reads them back, displaying them on the screen. Next month I will begin discussing string variables and expand this sample program into a mini database/mailing list program. ■

Listing continued.

```
0818                    187  ; carriage return is printed first
0818                    188  ; in order to insure that the DOS
0818                    189  ; command is recognized.
0818                    190  ;
0818                    191  ;
0818 20 36 78           192          JSR PRINT
081B 8D 84 CF           193          BYT CR,CTLD,"OPEN NUMBERS,S6,D1,L8",CR,0
081E D0 C5 CE
0821 A0 CE D5
0824 CD C2 C5
0827 D2 D3 AC
082A D3 B6 AC
082D C4 B1 AC
0830 CC B8 8D
0833 00
0834                    194  ;
0834 20 18 78           195          JSR FORO
0837 63 0D 01           196          ADR I,1,10
083A 00 0A 00
083D                    197  ;
083D                    198  ;
083D                    199  ; Emulate the BASIC statement:
083D                    200  ;
083D                    201  ;   PRINT CHR$(4);"WRITE NUMBERS, R";I;",B0"
083D                    202  ;
083D 20 36 78           203          JSR PRINT
0840 84 D7 D2           204          BYT CTLD,"WRITE NUMBERS,R",0
0843 C9 D4 C5
0846 A0 CE D5
0849 CD C2 C5
084C D2 D3 AC
084F D2 00
0851 20 3C 78           205          JSR PRTINT
0854 63 0D              206          ADR I
0856 20 36 78           207          JSR PRINT
0859 AC C2 B0           208          BYT ",B0",CR,0
085C 8D 00
085E                    209  ;
085E                    210  ;
085E                    211  ; Now output the value I to the textfile.
085E                    212  ;
085E                    213  ; This code emulates the BASIC statement:
085E                    214  ;
085E                    215  ;    PRINT I
085E                    216  ;
085E                    217  ;
085E 20 3C 78           218          JSR PRTINT
0861 63 0D              219          ADR I
0863 A9 8D              220          LDA #CR              ;Output a CR to finish
0865 20 00 78           221          JSR PUTC             ;off the line
0868                    222  ;
0868                    223  ;
0868                    224  ;  Repeat for 10 numbers
0868                    225  ;
0868 20 1B 78           226          JSR NEXT
086B                    227  ;
086B                    228  ;
086B                    229  ; At this point a random access file
086B                    230  ; has been created with 100 records.
086B                    231  ; Each record contains the record
086B                    232  ; number of that particular record.
086B                    233  ; To insure the integrity of the
086B                    234  ; file, it must be closed.  The following
086B                    235  ; SPEED/ASM code emulates the BASIC
086B                    236  ; code:
086B                    237  ;
086B                    238  ;   PRINT CHR$(4);"CLOSE NUMBERS"
086B                    239  ;
086B 20 36 78           240          JSR PRINT
086E 84 C3 CC           241          BYT CTLD,"CLOSE NUMBERS",CR,0
0871 CF D3 C5
0874 A0 CE D5
0877 CD C2 C5
087A D2 D3 8D
087D 00
087E                    242  ;
087E                    243  ;
087E                    244  **** End of file creation section.
087E                    245  ;
087E                    246  ;
087E                    247  ;
087E                    248  ;
087E                    249  **********************************
087E                    250  ;
087E                    251  ;
087E                    252  ; The following section re-opens
087E                    253  ; the text file just created and
087E                    254  ; prints the records onto the screen
087E                    255  ; This verifies proper operation
087E                    256  ; of the file creation process.
087E                    257  ;
087E                    258  ;
087E                    259  ;
087E 20 36 78           260          JSR PRINT
0881 8D D6 E5           261          BYT CR,"Verifying Records",CR,CR,0
0884 F2 E9 E6
```

Listing continued.

Listing continued.

```
0887 F9 E9 EE
088A E7 A0 D2
088D E5 E3 EF
0890 F2 E4 F3
0893 8D 8D 00
0896                262 ;
0896                263 ;
0896 20 36 78       264         JSR PRINT
0899 8D 84 CF       265         BYT CR,CTLD,"OPEN NUMBERS,L8",CR,0
089C D0 C5 CE
089F A0 CE D5
08A2 CD C2 C5
08A5 D2 D3 AC
08A8 CC B8 8D
08AB 00
08AC                266 ;
08AC 20 18 78       267         JSR FOR0
08AF 63 0D 01       268         ADR I,1,10
08B2 00 0A 00
08B5                269 ;
08B5 20 36 78       270         JSR PRINT
08B8 84 D2 C5       271         BYT CTLD,"READ NUMBERS,R",0
08BB C1 C4 A0
08BE CE D5 CD
08C1 C2 C5 D2
08C4 D3 AC D2
08C7 00
08C8 20 3C 78       272         JSR PRTINT
08CB 63 0D          273         ADR I
08CD 20 36 78       274         JSR PRINT
08D0 AC C2 B0       275         BYT ",B0",CR,0
08D3 8D 00
08D5                276 ;
08D5 20 0F 78       277         JSR READLN          ;Force a new line to be read.
08D8 20 42 78       278         JSR RDINT
08DB 65 0D          279         ADR J
08DD                280 ;
08DD                281 ; Print the record's value.
08DD                282 ;
08DD 20 36 78       283         JSR PRINT
08E0 D2 E5 E3       284         BYT "Record number ",0
08E3 EF F2 E4
08E6 A0 EE F5
08E9 ED E2 E5
08EC F2 A0 00
08EF 20 3C 78       285         JSR PRTINT
08F2 63 0D          286         ADR I
08F4 20 36 78       287         JSR PRINT
08F7 A0 E3 EF       288         BYT " contained '",0
08FA EE F4 E1
08FD E9 EE E5
0900 E4 A0 A7
0903 00
0904 20 3C 78       289         JSR PRTINT
0907 65 0D          290         ADR J
0909 20 36 78       291         JSR PRINT
090C A7 8D 00       292         BYT "'",CR,0
090F                293 ;
090F                294 ; Move on to the next record.
090F                295 ;
090F 20 1B 78       296         JSR NEXT
0912                297 ;
0912                298 ;
0912                299 ; ** End of record verification.
0912                300 ;
0912                301 ;
0912                302 ********************************
0912                303 ;
0912                304 ; Now print the file out backwards
0912                305 ;
0912                306 ;
0912                307 ;
0912 20 36 78       308         JSR PRINT
0915 8D D2 E5       309         BYT CR,"Reading records in reverse order",CR,CR,0
0918 E1 E4 E9
091B EE E7 A0
091E F2 E5 E3
0921 EF F2 E4
0924 F3 A0 E9
0927 EE A0 F2
092A E5 F6 E5
092D F2 F3 E5
0930 A0 EF F2
0933 E4 E5 F2
0936 8D 8D 00
0939                310 ;
0939 4C 42 09       311         JMP DOFOR           ;Hide the following constants
093C                312 ;
093C FF FF          313 CM1     ADR !-1  = ADR !-1
093E 01 00          314 C1      ADR 1
0940 0A 00          315 C10     ADR 10
0942                316 ;
0942                317 ;
0942 20 15 78       318 DOFOR   JSR FOR
0945 63 0D 40       319         ADR I,C10,C1,CM1    ;Stepsize of -1
0948 09 3E 09
094B 3C 09
094D                320 ;
094D 20 36 78       321         JSR PRINT
0950 84 D2 C5       322         BYT CTLD,"READ NUMBERS,R",0
0953 C1 C4 A0
0956 CE D5 CD
0959 C2 C5 D2
095C D3 AC D2
095F 00
0960 20 3C 78       323         JSR PRTINT
0963 63 0D          324         ADR I
0965 20 36 78       325         JSR PRINT
0968 AC C2 B0       326         BYT ",B0",CR,0
096B 8D 00
096D                327 ;
096D 20 0F 78       328         JSR READLN          ;Fetch a new line
0970 20 42 78       329         JSR RDINT
0973 65 0D          330         ADR J
0975                331 ;
0975 20 36 78       332         JSR PRINT
0978 D2 E5 E3       333         BYT "Record number ",0
097B EF F2 E4
097E A0 EE F5
0981 ED E2 E5
0984 F2 A0 00
0987 20 3C 78       334         JSR PRTINT
098A 63 0D          335         ADR I
098C 20 36 78       336         JSR PRINT
098F A0 E3 EF       337         BYT " contained '",0
0992 EE F4 E1
0995 E9 EE E5
0998 E4 A0 A7
099B 00
099C 20 3C 78       338         JSR PRTINT
099F 65 0D          339         ADR J
09A1 20 36 78       340         JSR PRINT
09A4 A7 8D 00       341         BYT "'",CR,0
09A7                342 ;
09A7 20 1B 78       343         JSR NEXT
09AA                344 ;
09AA                345 ;
09AA                346 ********************************
09AA                347 ;
09AA                348 ;
```

Listing continued.

Listing continued.

```
09AA                    349  ; Now get the bounds for output
09AA                    350  ; from the keyboard.
09AA                    351  ;
09AA 20 36 78           352          JSR PRINT          ;This turns off the DOS
09AD 8D 84 8D           353          BYT CR,CTLD,CR,0   ;READ mode.
09B0 00
09B1                    354  ;
09B1 20 36 78           355  REREAD0 JSR PRINT
09B4 C9 EE F0           356          BYT "Input a lower bounds:",0 ;Get first record #
09B7 F5 F4 A0
09BA E1 A0 EC
09BD EF F7 E5
09C0 F2 A0 E2
09C3 EF F5 EE
09C6 E4 F3 BA
09C9 00
09CA                    357  ;
09CA 20 0F 78           358          JSR READLN
09CD 20 42 78           359          JSR RDINT
09D0 67 0D              360          ADR FIRST
09D2 70 DD              361          BVS REREAD0
09D4                    362  ; FIRST must be greater than zero
09D4                    363  ; and less than or equal to 10.
09D4                    364  ;
09D4 20 21 78           365          JSR IFI0
09D7 67 0D BE           366          ADR FIRST,GT,0
09DA 00 00 00
09DD D0 29              367          BNR FIRSTOK0
09DF                    368  ;
09DF 20 36 78           369          JSR PRINT
09E2 8D 87 D6           370          BYT CR,BELL,"Value must be greater than zero",CR,0
09E5 E1 BC F5
09E8 E5 A0 ED
09EB F5 F3 F4
09EE A0 E2 E5
09F1 A0 E7 F2
09F4 E5 E1 F4
09F7 E5 F2 A0
09FA F4 E8 E1
09FD EE A0 FA
0A00 E5 F2 EF
0A03 8D 00
0A05 4C B1 09           371          JMP REREAD0
0A08                    372  ;
0A08 20 21 78           373  FIRSTOK0 JSR IFI0
0A0B 67 0D BC           374          ADR FIRST,LE,10
0A0E BD 0A 00
0A11 D0 24              375          BNR FIRSTOK1
0A13                    376  ;
0A13 20 36 78           377          JSR PRINT
0A16 8D 87 D6           378          BYT CR,BELL,"Value must be less than 11",CR,0
0A19 E1 BC F5
0A1C E5 A0 ED
0A1F F5 F3 F4
0A22 A0 E2 E5
0A25 A0 EC E5
0A28 F3 F3 A0
0A2B F4 E8 E1
0A2E EE A0 B1
0A31 B1 8D 00
0A34 4C B1 09           379          JMP REREAD0
0A37                    380  ;
0A37                    381  ;
0A37                    382  FIRSTOK1:
0A37 20 36 78           383  REREAD1 JSR PRINT
0A3A 8D C5 EE           384          BYT CR,"Enter the final value:",0
0A3D F4 E5 F2
0A40 A0 F4 E8
0A43 E5 A0 E6
0A46 E9 EE E1
0A49 EC A0 F6
```

```
0A4C E1 BC F5
0A4F E5 BA 00
0A52                    385  ;
0A52 20 0F 78           386          JSR READLN
0A55 20 42 78           387          JSR RDINT
0A58 69 0D              388          ADR ENDVAL
0A5A 70 DB              389          BVS REREAD1        ;In case of error
0A5C                    390  ;
0A5C                    391  ; Check to make sure that ENDVAL is
0A5C                    392  ; in the range 1..10.
0A5C                    393  ;
0A5C 20 21 78           394          JSR IFI0
0A5F 69 0D BE           395          ADR ENDVAL,GT,0
0A62 00 00 00
0A65 D0 30              396          BNR ENDOK0
0A67                    397  ;
0A67 20 36 78           398          JSR PRINT
0A6A 8D 87 C5           399          BYT CR,BELL,"Ending value must be greater than zero",CR,0
0A6D EE E4 E9
0A70 EE E7 A0
0A73 F6 E1 BC
0A76 F5 E5 A0
0A79 ED F5 F3
0A7C F4 A0 E2
0A7F E5 A0 E7
0A82 F2 E5 E1
0A85 F4 E5 F2
0A88 A0 F4 E8
0A8B E1 EE A0
0A8E FA E5 F2
0A91 EF 8D 00
0A94 4C 37 0A           400          JMP REREAD1
0A97                    401  ;
0A97 20 21 78           402  ENDOK0  JSR IFI0
0A9A 69 0D BC           403          ADR ENDVAL,LE,10
0A9D BD 0A 00
0AA0 D0 2A              404          BNR ENDOK1
0AA2                    405  ;
0AA2 20 36 78           406          JSR PRINT
0AA5 8D 87 C5           407          BYT CR,BELL,"Ending value must be less than 11",0
0AA8 EE E4 E9
0AAB EE E7 A0
0AAE F6 E1 BC
0AB1 F5 E5 A0
0AB4 ED F5 F3
0AB7 F4 A0 E2
0ABA E5 A0 EC
0ABD E5 F3 F3
0AC0 A0 F4 E8
0AC3 E1 EE A0
0AC6 B1 B1 00
0AC9 4C 37 0A           408          JMP REREAD1
0ACC                    409  ;
0ACC                    410  ;
0ACC                    411  ; Now get a stepsize value from the user.
0ACC                    412  ;
0ACC                    413  ENDOK1:
0ACC 20 36 78           414  REREAD2 JSR PRINT
0ACF 8D C5 EE           415          BYT CR,"Enter a stepsize value:",0
0AD2 F4 E5 F2
0AD5 A0 E1 A0
0AD8 F3 F4 E5
0ADB F0 F3 E9
0ADE FA E5 A0
0AE1 F6 E1 BC
0AE4 F5 E5 BA
0AE7 00
0AE8                    416  ;
0AE8 20 0F 78           417          JSR READLN
0AEB 20 42 78           418          JSR RDINT
```

Listing continued.

Listing continued.

```
0AEE 6B 0D      419          ADR STEPSIZE
0AF0 70 DA      420          BVS REREAD2
0AF2            421 ;
0AF2            422 ; The stepsize must be checked to
0AF2            423 ; make sure that it matches the
0AF2            424 ; following conditions:
0AF2            425 ;
0AF2            426 ;  It must be in the range 1..10
0AF2            427 ;     or -1..-10.
0AF2            428 ;
0AF2            429 ;  If ENDVAL is greater than FIRST,
0AF2            430 ;  STEPSIZE must be positive.
0AF2            431 ;
0AF2            432 ;  If ENDVAL is less than FIRST,
0AF2            433 ;  STEPSIZE must be negative.
0AF2            434 ;
0AF2            435 ;
0AF2 20 21 78   436          JSR IFI0
0AF5 6B 0D BD   437          ADR STEPSIZE,EQ,0
0AF8 00 00 00   *
0AFB F0 26      438          BFL SSOK0
0AFD            439 ;
0AFD 20 36 78   440          JSR PRINT
0B00 8D 87 D3   441          BYT CR,BELL,"Stepsize must not equal zero",CR,0
0B03 F4 E5 F0
0B06 F3 E9 FA
0B09 E5 A0 ED
0B0C F5 F3 F4
0B0F A0 EE EF
0B12 F4 A0 E5
0B15 F1 F5 E1
0B18 BC A0 FA
0B1B E5 F2 EF
0B1E 8D 00
0B20 4C CC 0A   442          JMP REREAD2
0B23            443 ;
0B23            444 ;
0B23 20 21 78   445 SSOK0    JSR IFI0
0B26 6B 0D BE   446          ADR STEPSIZE,GT,0
0B29 00 00 00
0B2C F0 78      447          BFL SSISNEG
0B2E            448 ;
0B2E            449 ; STEPSIZE > 0, make sure that ENDVAL >= FIRST.
0B2E            450 ;
0B2E 20 1E 78   451          JSR IFI
0B31 69 0D BE   452          ADR ENDVAL,GE,FIRST
0B34 BD 67 0D
0B37 D0 23      453          BTR CHKPOSSS
0B39            454 ;
0B39 20 36 78   455          JSR PRINT
0B3C 8D 87 D3   456          BYT CR,BELL,"STEPSIZE must be negative",CR,0
0B3F D4 C5 D0
0B42 D3 C9 DA
0B45 C5 A0 ED
0B48 F5 F3 F4
0B4B A0 E2 E5
0B4E A0 EE E5
0B51 E7 E1 F4
0B54 E9 F6 E5
0B57 8D 00
0B59 4C CC 0A   457          JMP REREAD2
0B5C            458 ;
0B5C            459 ; Make sure that STEPSIZE is in the range 1..10.
0B5C            460 ;
0B5C 20 21 78   461 CHKPOSSS JSR IFI0
0B5F 6B 0D BE   462          ADR STEPSIZE,GE,1
0B62 BD 01 00
0B65 8D 6D 0D   463          STA BOOLEAN         ;Save for compound test.
0B68 20 21 78   464          JSR IFI0
0B6B 6B 0D BC   465          ADR STEPSIZE,LE,10
0B6E BD 0A 00
0B71 2D 6D 0D   466          AND BOOLEAN
0B74 F0 03      467          BFL BADSS
0B76 4C 1E 0C   468          JMP GOODSS
0B79            469 ;
0B79            470 ;
0B79 20 36 78   471 BADSS    JSR PRINT
0B7C 8D 87 D3   472          BYT CR,BELL,"STEPSIZE must be in the range 1..10",CR,0
0B7F D4 C5 D0
0B82 D3 C9 DA
0B85 C5 A0 ED
0B88 F5 F3 F4
0B8B A0 E2 E5
0B8E A0 E9 EE
0B91 A0 F4 E8
0B94 E5 A0 F2
0B97 E1 EE E7
0B9A E5 A0 B1
0B9D AE AE B1
0BA0 B0 8D 00
0BA3 4C CC 0A   473          JMP REREAD2
0BA6            474 ;
0BA6            475 ;
0BA6            476 ;
0BA6            477 ; STEPSIZE < 0, make sure that ENDVAL <= FIRST.
0BA6            478 ;
0BA6 20 1E 78   479 SSISNEG  JSR IFI
0BA9 69 0D BC   480          ADR ENDVAL,LE,FIRST
0BAC BD 67 0D
0BAF D0 24      481          BTR CHKNEGSS
0BB1            482 ;
0BB1 20 36 78   483          JSR PRINT
0BB4 8D 87 D3   484          BYT CR,BELL,"STEPSIZE must be positive.",CR,0
0BB7 D4 C5 D0
0BBA D3 C9 DA
0BBD C5 A0 ED
0BC0 F5 F3 F4
0BC3 A0 E2 E5
0BC6 A0 F0 EF
0BC9 F3 E9 F4
0BCC E9 F6 E5
0BCF AE 8D 00
0BD2 4C CC 0A   485          JMP REREAD2
0BD5            486 ;
0BD5            487 ;
0BD5            488 ; Make sure that STEPSIZE is in the range -10..-1.
0BD5            489 ;
0BD5 20 21 78   490 CHKNEGSS JSR IFI0
0BD8 6B 0D BE   491          ADR STEPSIZE,GE,!-10   ADR STEPSIZE,GE,!-10
0BDB BD F6 FF
0BDE 8D 6D 0D   492          STA BOOLEAN         ;Save for compound test
0BE1 20 21 78   493          JSR IFI0
0BE4 6B 0D BC   494          ADR STEPSIZE,LE,!-1
0BE7 BD FF FF
0BEA 2D 6D 0D   495          AND BOOLEAN
0BED D0 2F      496          BTR GOODSS
0BEF            497 ;
0BEF 20 36 78   498          JSR PRINT
0BF2 8D 87 D3   499          BYT CR,BELL,"STEPSIZE must be in the range -1..-10",CR,0
0BF5 D4 C5 D0
0BF8 D3 C9 DA
0BFB C5 A0 ED
0BFE F5 F3 F4
0C01 A0 E2 E5
0C04 A0 E9 EE
0C07 A0 F4 E8
0C0A E5 A0 F2
0C0D E1 EE E7
0C10 E5 A0 AD
0C13 B1 AE AE
```

Listing continued.

Listing continued.

```
0C16 AD B1 B0
0C19 8D 00
0C1B 4C CC 0A   500          JMP REREAD2
0C1E            501  ;
0C1E            502  ;
0C1E            503  ; If STEPSIZE contains an appropriate value
0C1E            504  ; print the records as requested.
0C1E            505  ;
0C1E            506  ;
0C1E A9 8D      507  GOODSS   LDA #CR
0C20 20 00 78   508          JSR PUTC            ;Init for DOS.
0C23            509  ;
0C23 20 15 78   510          JSR FOR
0C26 63 0D 67   511          ADR I,FIRST,ENDVAL,STEPSIZE
0C29 0D 69 0D
0C2C 6B 0D
0C2E            512  ;
0C2E            513  ; Read the specified record
0C2E            514  ;
0C2E 20 36 78   515          JSR PRTINT
0C31 84 D2 C5   516          BYT CTLD,"READ NUMBERS,R",0
0C34 C1 C4 A0
0C37 CE D5 CD
0C3A C2 C5 D2
0C3D D3 AC D2
0C40 00
0C41 20 3C 78   517          JSR PRTINT
0C44 63 0D      518          ADR I
0C46 20 36 78   519          JSR PRINT
0C49 AC C2 B0   520          BYT ",B0",CR,0
0C4C 8D 00
0C4E            521  ;
0C4E 20 0F 78   522          JSR READLN
0C51 20 42 78   523          JSR RDINT
0C54 65 0D      524          ADR J
0C56            525  ;
0C56            526  ;
0C56 20 1E 78   527          JSR IFI
0C59 63 0D BD   528          ADR I,EQ,FIRST
0C5C 00 67 0D
0C5F F0 21      529          BFL >0
0C61            530  ;
0C61 20 36 78   531          JSR PRINT
0C64 D4 E8 E5   532          BYT "The first record contains ",0
0C67 A0 B6 E9
0C6A F2 F3 F4
0C6D A0 F2 E5
0C70 E3 EF F2
0C73 E4 A0 E3
0C76 EF EE F4
0C79 E1 E9 EE
0C7C F3 A0 00
0C7F 4C 25 0D   533          JMP PRTREC
0C82            534  ;
0C82 20 1E 78   535  ^0       JSR IFI
0C85 63 0D BD   536          ADR I,EQ,ENDVAL
0C88 00 69 0D
0C8B F0 20      537          BFL >1
0C8D            538  ;
0C8D 20 36 78   539          JSR PRINT
0C90 D4 E8 E5   540          BYT "The last record contains ",0
0C93 A0 EC E1
0C96 F3 F4 A0
0C99 F2 E5 E3
0C9C EF F2 E4
0C9F A0 E3 EF
0CA2 EE F4 E1
0CA5 E9 EE F3
0CA8 A0 00
0CAA 4C 25 0D   541          JMP PRTREC
0CAD            542  ;
0CAD            543  ;
0CAD 20 21 78   544  ^1       JSR IFI0
0CB0 63 0D BD   545          ADR I,EQ,2
0CB3 00 02 00
0CB6 F0 1F      546          BFL >2
0CB8            547  ;
0CB8 20 36 78   548          JSR PRINT
0CBB D4 E8 E5   549          BYT "The 2nd record contains ",0
0CBE A0 B2 EE
0CC1 E4 A0 F2
0CC4 E5 E3 EF
0CC7 F2 E4 A0
0CCA E3 EF EE
0CCD F4 E1 E9
0CD0 EE F3 A0
0CD3 00
0CD4 4C 25 0D   550          JMP PRTREC
0CD7            551  ;
0CD7 20 21 78   552  ^2       JSR IFI0
0CDA 63 0D BD   553          ADR I,EQ,3
0CDD 00 03 00
0CE0 F0 1F      554          BFL >3
0CE2            555  ;
0CE2 20 36 78   556          JSR PRINT
0CE5 D4 E8 E5   557          BYT "The 3rd record contains ",0
0CE8 A0 B3 F2
0CEB E4 A0 F2
0CEE E5 E3 EF
0CF1 F2 E4 A0
0CF4 E3 EF EE
0CF7 F4 E1 E9
0CFA EE F3 A0
0CFD 00
0CFE 4C 25 0D   558          JMP PRTREC
0D01            559  ;
0D01            560  ;
0D01 20 36 78   561  ^3       JSR PRINT
0D04 D4 E8 E5   562          BYT "The ",0
0D07 A0 00
0D09 20 3C 78   563          JSR PRTINT
0D0C 63 0D      564          ADR I
0D0E 20 36 78   565          JSR PRINT
0D11 F4 E8 A0   566          BYT "th record contains ",0
0D14 F2 E5 E3
0D17 EF F2 E4
0D1A A0 E3 EF
0D1D EE F4 E1
0D20 E9 EE F3
0D23 A0 00
0D25            567  ;
0D25            568  ;
0D25 20 36 78   569  PRTREC   JSR PRINT
0D28 BA A0 A0   570          BYT ":  '",0
0D2B A7 00
0D2D 20 3C 78   571          JSR PRTINT
0D30 65 0D      572          ADR J
0D32 20 36 78   573          JSR PRINT
0D35 A7 8D 00   574          BYT "'",CR,0
0D38            575  ;
0D38            576  ;
0D38 20 1B 78   577          JSR NEXT
0D3B            578  ;
0D3B            579  ;
0D3B            580  **********************************
0D3B            581  ;
0D3B            582  ;
0D3B            583  ; All Done...
0D3B            584  ;
0D3B 20 36 78   585          JSR PRINT
0D3E 8D 8D 8D   586          BYT CR,CR,CR,"That's all folks!",CR,0
0D41 D4 E8 E1
0D44 F4 A7 F3
0D47 A0 E1 EC
0D4A EC A0 B6
0D4D EF EC EB
0D50 F3 A1 8D
0D53 00
0D54            587  ;
0D54            588  ;
0D54            589  ;
0D54            590  ; Close all open files.
0D54            591  ;
0D54 20 36 78   592          JSR PRINT
0D57 8D 84 C3   593          BYT CR,CTLD,"CLOSE",CR,0
0D5A CC CF D3
0D5D C5 8D 00
0D60            594  ;
0D60            595  ;
0D60 4C 59 FF   596          JMP EXIT
0D63            597  ;
0D63            598  ;
0D63            599  ; Variable declarations
0D63            600  ;
0D63            601  ;
0D63 00 00      602  I        ADR 0
0D65 00 00      603  J        ADR 0
0D67 00 00      604  FIRST    ADR 0
0D69 00 00      605  ENDVAL   ADR 0
0D6B 00 00      606  STEPSIZE ADR 0
0D6D 00         607  BOOLEAN  BYT 0
0D6E            608  ;
0D6E            609  ;
0D6E            610          END

***** END OF ASSEMBLY


]

Verifying Records


Record number 1 contained '1'
Record number 2 contained '2'
Record number 3 contained '3'
Record number 4 contained '4'
Record number 5 contained '5'
Record number 6 contained '6'
Record number 7 contained '7'
Record number 8 contained '8'
Record number 9 contained '9'
Record number 10 contained '10'

Reading records in reverse order

Record number 10 contained '10'
Record number 9 contained '9'
Record number 8 contained '8'
Record number 7 contained '7'
Record number 6 contained '6'
Record number 5 contained '5'
Record number 4 contained '4'
Record number 3 contained '3'
Record number 2 contained '2'
Record number 1 contained '1'

Input a lower bounds:1

Enter the final value:7
```

Listing continued.

Listing continued.

```
Enter a stepsize value:2

The first record contains :  '1'
The 3rd record contains :  '3'
The 5th record contains :  '5'
The last record contains :  '7'


That's all folks!

Verifying Records


Record number 1 contained '1'
Record number 2 contained '2'
Record number 3 contained '3'
Record number 4 contained '4'
Record number 5 contained '5'
Record number 6 contained '6'
Record number 7 contained '7'
Record number 8 contained '8'
Record number 9 contained '9'
Record number 10 contained '10'

Reading records in reverse order

Record number 10 contained '10'
Record number 9 contained '9'
Record number 8 contained '8'
Record number 7 contained '7'
Record number 6 contained '6'
Record number 5 contained '5'
Record number 4 contained '4'
Record number 3 contained '3'
Record number 2 contained '2'
Record number 1 contained '1'

Input a lower bounds:9

Enter the final value:3

Enter a stepsize value:-2

The first record contains :  '9'
The 7th record contains :  '7'
The 5th record contains :  '5'
The last record contains :  '3'


That's all folks!

Verifying Records


Record number 1 contained '1'
Record number 2 contained '2'
Record number 3 contained '3'
Record number 4 contained '4'
Record number 5 contained '5'
Record number 6 contained '6'
Record number 7 contained '7'
Record number 8 contained '8'
Record number 9 contained '9'
Record number 10 contained '10'

Reading records in reverse order

Record number 10 contained '10'
Record number 9 contained '9'
Record number 8 contained '8'
Record number 7 contained '7'
Record number 6 contained '6'
Record number 5 contained '5'
Record number 4 contained '4'
Record number 3 contained '3'
Record number 2 contained '2'
Record number 1 contained '1'

Input a lower bounds:0

Value must be greater than zero
Input a lower bounds:11

Value must be less than 11
Input a lower bounds:1

Enter the final value:0

Ending value must be greater than zero

Enter the final value:12

Ending value must be less than 11
Enter the final value:4

Enter a stepsize value:-3

STEPSIZE must be positive.

Enter a stepsize value:0

Stepsize must not equal zero

Enter a stepsize value:4

The first record contains :  '1'


That's all folks!
```

Listing 7. SPEED/ASM Equates.

```
0800                    1          TTL "Listing 7 : SPEED/ASM Equates"
0800                    2  ;
0800                    3  ;
0800                    4  ;
0800                    5  *********************************
0800                    6  *                               *
0800                    7  *       SPEED/ASM Equates       *
0800                    8  *                               *
0800                    9  *********************************
0800                   10  ;
0800                   11  ;
0800                   12  ;
0800                   13  ;
0800                   14  ;
0800                   15  ;
0800                   16  ;
0800                   17  ;
0800                   18  ; GENERAL PURPOSE EQUATES
0800                   19  ;
0800                   20  ; The following variables are used
0800                   21  ; by the SPEED/ASM package and
0800                   22  ; shouldn't be used by the SPEED/ASM
0800                   23  ; programmer.
0800                   24  ;
0800                   25  ;
0800                   26  ;
0800                   27  ;
0000                   28  FORASAV  EPZ 0
0001                   29  FORXSAV  EPZ FORASAV+1
0002                   30  FORYSAV  EPZ FORXSAV+1
0003                   31  FORZPG   EPZ FORYSAV+1
0005                   32  DESTADR  EPZ FORZPG+2
0007                   33  PTRADR   EPZ DESTADR+2
0009                   34  ISIMMED  EPZ PTRADR+2
000A                   35  OP       EPZ ISIMMED+1
000C                   36  MAXLEN   EPZ OP+2
000D                   37  VALUE    EPZ MAXLEN+1
000F                   38  DIGIT    EPZ VALUE+2
0010                   39  LEAD0    EPZ DIGIT+1
0011                   40  JMPADR   EPZ LEAD0+1
0013                   41  COUNT    EPZ JMPADR+2
0014                   42  GOTLN    EPZ COUNT+1
0015                   43  LINEINDX EPZ GOTLN+1
0016                   44  SIGN     EPZ LINEINDX+1
0017                   45  ACL      EPZ SIGN+1
0018                   46  ACH      EPZ ACL+1
0019                   47  XINDL    EPZ ACH+1
001A                   48  XINDH    EPZ XINDL+1
001B                   49  AUXL     EPZ XINDH+1
001C                   50  AUXH     EPZ AUXL+1
0800                   51  ;
0033                   52  PROMPT   EPZ $33
004E                   53  RNDL     EPZ $4E
004F                   54  RNDH     EPZ $4F
0100                   55  STACK    EQU $100
0200                   56  INPUT    EQU $200
0800                   57  ;
0800                   58  ;
0800                   59  ;
0800                   60  ;
0800                   61  ;
0800                   62  ;
0800                   63  ;
0800                   64  *************
0800                   65  * CONSTANTS *
0800                   66  *************
0800                   67  ;
0800                   68  ;
0800                   69  ;
0800                   70  ; The following symbols are constants
0800                   71  ; for the values "FALSE", "TRUE", and
0800                   72  ; Carriage Return (respectively).
0800                   73  ;
0800                   74  ; These symbols should only appear
0800                   75  ; as immediate operands to a 6502
0800                   76  ; instruction or in the operand field
0800                   77  ; of a pseudo-opcode like BYT.
0800                   78  ;
0800                   79  ;
0800                   80  ;
0800                   81  ;
0800                   82  ;
0000                   83  FALSE    EQU 0
0001                   84  TRUE     EQU 1
008D                   85  CR       EQU $8D
0800                   86  ;
0800                   87  ;
0800                   88  ;
0800                   89  ;
```

Listing continued.

Listing continued.

```
0800                    90  ;
0800                    91  ;
0800                    92  ; "IF" STATEMENT EQUATES
0800                    93  ;
0800                    94  ; The following symbols should only
0800                    95  ; be used in the ADR pseudo-opcode
0800                    96  ; following a call to the SPEED/ASM
0800                    97  ; IFx routines.
0800                    98  ;
0800                    99  ;
0800                   100  ;
00ED                   101  EQ      EQU "="
00A3                   102  NE      EQU "#"
00BE                   103  GT      EQU ">"
00BC                   104  LT      EQU "<"
BDBE                   105  GE      EQU ">"|"="*256
BDBC                   106  LE      EQU "<"|"="*256
0800                   107  ;
0800                   108  ;
0800                   109  ;
0800                   110  ;
0800                   111  ;
0800                   112  ;
0800                   113  ;
0800                   114  ****************************
0800                   115  * SPEED/ASM ENTRY POINTS *
0800                   116  ****************************
0800                   117  ;
0800                   118  ;
0800                   119  ;
0800                   120  ;
0800                   121  ; NOTE: THE EQUATE OF PUTC MUST
0800                   122  ; BE CHANGED IF YOU RELOCATE
0800                   123  ; SPEED/ASM TO SOME LOCATION
0800                   124  ; OTHER THAN $7800
0800                   125  ;
0800                   126  ;
0800                   127  ;
7800                   128  PUTC    EQU $7800
7803                   129  GETC    EQU PUTC+3
7806                   130  SAGL    EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809                   131  SAPC    EQU SAGL+3          ; "  "  "  "  "   "  "
780C                   132  HOME    EQU SAPC+3          ;HOME AND CLEAR
780F                   133  READLN  EQU HOME+3
7812                   134  INIT    EQU READLN+3
7815                   135  FOR     EQU INIT+3
7818                   136  FOR0    EQU FOR+3
781B                   137  NEXT    EQU FOR0+3
781E                   138  IFI     EQU NEXT+3
7821                   139  IFI0    EQU IFI+3
7824                   140  IFS     EQU IFI0+3
7827                   141  IFS0    EQU IFS+3
782A                   142  MOVE    EQU IFS0+3
782D                   143  LOAD    EQU MOVE+3
7830                   144  MOVS    EQU LOAD+3
7833                   145  LDSTR   EQU MOVS+3
7836                   146  PRINT   EQU LDSTR+3
7839                   147  PRTSTR  EQU PRINT+3
783C                   148  PRTINT  EQU PRTSTR+3
783F                   149  RDSTR   EQU PRTINT+3
7842                   150  RDINT   EQU RDSTR+3
7845                   151  ONXGOTO EQU RDINT+3
7848                   152  CASE    EQU ONXGOTO+3
784B                   153  CASEI   EQU CASE+3
784E                   154  INSET   EQU CASEI+3
7851                   155  NOTINSET EQU INSET+3
7854                   156  ABS     EQU NOTINSET+3
7857                   157  NEG     EQU ABS+3
785A                   158  MUL     EQU NEG+3
785D                   159  DIV     EQU MUL+3
7860                   160  MOD     EQU DIV+3
7863                   161  RND     EQU MOD+3
7866                   162  SUBSTR  EQU RND+3
7869                   163  INDEX   EQU SUBSTR+3
786C                   164  LENGTH  EQU INDEX+3
786F                   165  CONCAT  EQU LENGTH+3
7872                   166  GETWZPG EQU CONCAT+3        ;USED BY SPEED/ASM
7875                   167  RDFP    EQU GETWZPG+3
7878                   168  PRTE    EQU RDFP+3
787B                   169  PRTF    EQU PRTE+3
787E                   170  FADD    EQU PRTF+3
7881                   171  FSUB    EQU FADD+3
7884                   172  FMUL    EQU FSUB+3
7887                   173  FDIV    EQU FMUL+3
788A                   174  FLT     EQU FDIV+3
788D                   175  FIX     EQU FLT+3
7890                   176  FNEG    EQU FIX+3
7893                   177  FADDIN  EQU FNEG+3
7896                   178  FSUBIN  EQU FADDIN+3
7899                   179  FTIMES  EQU FSUBIN+3
789C                   180  FINIO   EQU FTIMES+3
789F                   181  IFP     EQU FINIO+3
78A2                   182  MOVFP   EQU IFP+3
0800                   183  ;
0800                   184  ;
0800                   185  ;
0800                   186          END

***** END OF ASSEMBLY
```

The Assembly Advantage

by Randy Hyde

Character Strings

The previous installments of this column have described how to manipulate integer variables with the SPEED/ASM package. Now it's time to explore some additional data types supported by SPEED/ASM. For the next few months I'll discuss how you manipulate character strings, the most important non-numeric type of data.

String Variable Allocation

In part one of this series I described how to allocate storage for a string variable. For the benefit of those who don't have the April edition of *inCider*, I'll first review this procedure.

Unlike other data types in SPEED/ASM (integer, real and character), string variables require a *variable* amount of storage space. The amount of space required depends entirely on how big you wish to allow the string to grow. Strings in SPEED/ASM need n + 2 bytes, where n represents the *maximum* number of characters, up to 255, you will allow the string to contain. The first byte of a SPEED/ASM variable must contain this value. The next m bytes after that (where m < = n) must contain the characters that make up the string. Finally, the first byte following the characters making up the string must contain 0, which SPEED/ASM uses to mark the end of the character string. Figure 1 shows the SPEED/ASM format for a string variable.

Before defining a string variable you must determine its maximum possible size. If you try to store more characters into a string than you allow for, SPEED/ASM will truncate the string and return an error code. On the other hand, if you define a string using the maximum length (255 characters) when at most only 20 characters are ever required, you're wasting a lot of memory space. Obviously, some careful thought must go into deciding how much space to reserve.

The easiest way to do the reserving is to use LISA's ADR and DFS pseudo opcodes. The format for string declaration should be:

```
<label>  ADR   <maxlength>
         DFS   <maxlength>
```

where <label> is the name of the string variable you are declaring and <maxlength> is the maximum string length. <maxlength> must be in the range 1–255. This form of string declaration fulfills three functions: it stores the maximum string length into the first byte (as required), it stores 0 into the second byte (assuming <maxlength> is a value less than 255), and it reserves <maxlength> + 2 bytes as required for a SPEED/ASM string. (Two bytes are reserved by the ADR pseudo opcode, <maxlength> bytes are reserved by the DFS statement.) By storing 0 into the second byte of the string variable, this form of the string declaration initializes the variable to the empty string. The operation of this form

Randy Hyde is proprietor of Lazerwerks, creators of SPEED/ASM. Address correspondence to him at 925 Lorna St., Corona, CA 91720.

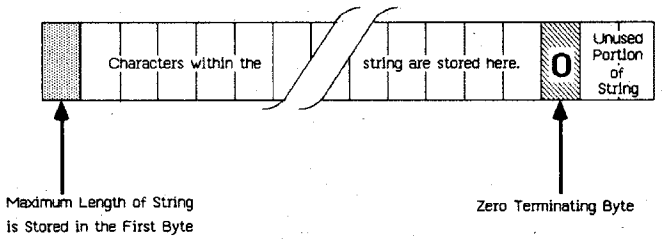
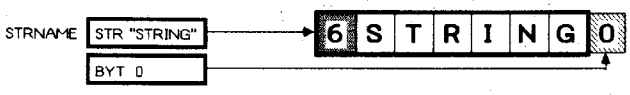


Figure 1. SPEED/ASM string format.

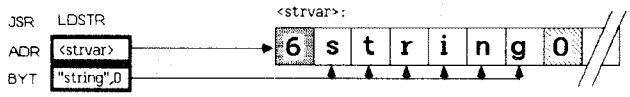
The ADR pseudo-opcode stores <strlen> in the first byte (this is the maximum length value required by SPEED/ASM strings) and then stores a zero into the second byte. This zero initializes the string to the empty string.
The DFS pseudo opcode reserves enough storage for the actual character string. SPEED/ASM strings require exactly <strlen>+2 bytes. By using both the ADR and DFS pseudo-ops in the fashion the required number of bytes are automatically allocated.

Figure 2. Operation of ADR/DFS pseudo opcodes for string declarations.



The STR pseudo-opcode stores a length byte followed by the characters in the string. In this case the STR pseudo-op emits a six followed by the six characters "STRING". The BYT pseudo-opcode emits the zero required at the end of all SPEED/ASM strings.

Figure 3. Using the STR and BYT pseudo opcodes to initialize a string variable.



The LDSTR routine copies the zero terminated string into the string variable whose address immediately follows the call to LDSTR. Note that in this example eight characters are copied: a length byte (which LDSTR computes), the six characters "string" and the zero terminating byte.

Figure 4. Operation of the LDSTR routine.

of the declaration is depicted in Figure 2.

Occasionally you may want to reserve space for a SPEED/ASM string variable and initialize the string to some fixed value. There are a couple of ways to accomplish this, depending on your requirements. If the string value will never change (i.e., you will never store another string value into the variable) then the following definition can be used:

```
<label>  STR   "<character string>"
         BYT   0
```

where <label> is the name of the string variable and <character string> is the data you wish to initialize the string to. The STR pseudo opcode emits the length of the specified string followed by the string itself. Since, in this case, the maximum possible length of the string *is* the length of the string, the STR pseudo opcode automatically emits the proper data for the maximum value. The BYT directive emits the required terminating 0 byte, since STR doesn't do so. This approach is pictured in Figure 3.

Using the STR method to declare an initial value for a string is great if the string value doesn't change during execution of the program, (or if you can guarantee that the initial string value is the largest string value). It is perfect for setting up such fixed strings as error messages and menus. If you need to initialize a string whose initial length *is not* the maximum length the string will grow to, you have to declare the string using the statements:

```
<label>  BYT   <maxlength>,"<string>"
         DFS   <maxlength> + 1 - <length
               of string>,0
```

Rather than explaining how this complicated affair works, space is better spent describing how to use SPEED/ASM routines to avoid such statements.

Initializing and Assigning Strings

The most basic string operation is the string assignment. SPEED/ASM supports two routines, LDSTR and MOVS, for this purpose. LDSTR copies a *string constant* into a string variable, and MOVS copies one *string variable* into another.

The LDSTR routine (load a string)

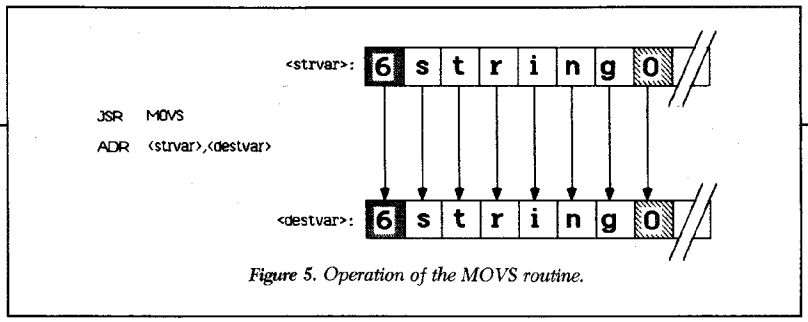


Figure 5. Operation of the MOVS routine.

replaces Basic string assignments of the form:

55 A$ = "STRING"

The syntax for the LDSTR is:

```
JSR   LDSTR
ADR   <deststr>
BYT   "<string constant>",0
```

where <deststr> is the name of the string variable into which you want to store the string <string constant>. LDSTR should be used to initialize a string variable with a string that is shorter than the maximum length. (See the problem in the last section.) The operation of LDSTR is shown in Figure 4.

The MOVS routine copies the contents of one string variable into another. This enables you to translate statements of the form:

100 A$ = B$

into SPEED/ASM. The calling sequence for the MOVS routine is:

```
JSR   MOVS
ADR   <source string>,
      <destination string>
```

where <source string> is the name of the source string and <destination string> is the name of the string you want the source string copied into. The operation of MOVS is shown in Figure 5.

The only problem with using LDSTR and MOVS occurs when you try to store a source string whose length is greater than the maximum length of the destination string. In this case LDSTR and MOVS truncate the string and store the largest string that will fit. Then LDSTR and MOVS return the 6502 overflow flag (V) set if such an error occurs. Likewise, if the error does not occur, the V flag is returned clear. So you can see if a string overflow occurred by checking the V flag (using the BVS and BVC instructions) after a JSR to LDSTR and MOVS.

String I/O

SPEED/ASM provides three routines for performing string input/output. These routines enable you to read a string from the keyboard and store it into a string variable, print

Listing 1. SPEED/ASM equates.

```
0800                1           TTL "Listing One: SPEED/ASM Equates"
0800                2  ;
0800                3  ;
0800                4  ;
0800                5  ***********************************
0800                6  *                                 *
0800                7  *       Listing One:              *
0800                8  *       SPEED/ASM Equates         *
0800                9  *                                 *
0800               10  ***********************************
0800               11  ;
0800               12  ;
0800               13  ;
0800               14  ;
0800               15  ;
0800               16  ;
0800               17  ;
0800               18  ;
0800               19  ; GENERAL PURPOSE EQUATES
0800               20  ;
0800               21  ; The following variables are used
0800               22  ; by the SPEED/ASM package and
0800               23  ; shouldn't be used by the SPEED/ASM
0800               24  ; programmer.
0800               25  ;
0800               26  ;
0800               27  ;
0800               28  ;
0000               29  FORASAV  EPZ 0
0001               30  FORXSAV  EPZ FORASAV+1
0002               31  FORYSAV  EPZ FORXSAV+1
0003               32  FORZPG   EPZ FORYSAV+1
0005               33  DESTADR  EPZ FORZPG+2
0007               34  PTRADR   EPZ DESTADR+2
0009               35  ISIMMED  EPZ PTRADR+2
000A               36  OP       EPZ ISIMMED+1
000C               37  MAXLEN   EPZ OP+2
000D               38  VALUE    EPZ MAXLEN+1
000F               39  DIGIT    EPZ VALUE+2
0010               40  LEAD0    EPZ DIGIT+1
0011               41  JMPADR   EPZ LEAD0+1
0013               42  COUNT    EPZ JMPADR+2
0014               43  GOTLN    EPZ COUNT+1
0015               44  LINEINDX EPZ GOTLN+1
0016               45  SIGN     EPZ LINEINDX+1
0017               46  ACL      EPZ SIGN+1
0018               47  ACH      EPZ ACL+1
0019               48  XTNDL    EPZ ACH+1
001A               49  XTNDH    EPZ XTNDL+1
001B               50  AUXL     EPZ XTNDH+1
001C               51  AUXH     EPZ AUXL+1
0800               52  ;
0033               53  PROMPT   EPZ $33
004E               54  RNDL     EPZ $4E
004F               55  RNDH     EPZ $4F
0100               56  STACK    EQU $100
0200               57  INPUT    EQU $200
0800               58  ;
0800               59  ;
0800               60  ;
0800               61  ;
0800               62  ;
0800               63  ;
0800               64  ;
0800               65  *************
0800               66  * CONSTANTS *
0800               67  *************
0800               68  ;
0800               69  ;
0800               70  ;
0800               71  ; The following symbols are constants
0800               72  ; for the values "FALSE", "TRUE", and
0800               73  ; Carriage Return (respectively).
0800               74  ;
0800               75  ; These symbols should only appear
0800               76  ; as immediate operands to a 6502
0800               77  ; instruction or in the operand field
0800               78  ; of a pseudo-opcode like BYT.
0800               79  ;
0800               80  ;
0800               81  ;
0800               82  ;
0800               83  ;
0000               84  FALSE    EQU 0
0001               85  TRUE     EQU 1
008D               86  CR       EQU $8D
0800               87  ;
0800               88  ;
0800               89  ;
0800               90  ;
0800               91  ;
0800               92  ;
0800               93  ; "IF" STATEMENT EQUATES
0800               94  ;
0800               95  ; The following symbols should only
0800               96  ; be used in the ADR pseudo-opcode
0800               97  ; following a call to the SPEED/ASM
```

Listing continued.

the contents of a string variable, and print a string constant.

To print a string constant use the PRINT routine, which should be quite familiar to you. We've been employing it all along to print prompts onto the Apple's video display. PRINT needs the calling sequence:

```
JSR    PRINT
BYT    "<string>",0
```

Note that the string constant must be terminated with a 0 byte.

To print the contents of a string variable you can use the PRTSTR routine. Type the statement:

```
JSR    PRTSTR
ADR    <string variable>
```

where <string variable> is the name of the string variable you wish to print.

Reading a string from the keyboard and storing it into a string variable is done by the RDSTR routine. RDSTR reads whatever data is present in the line input buffer up until a carriage return is detected. If the line buffer pointer is already pointing at a carriage return, a new line is read from the keyboard. The syntax for RDSTR is:

```
JSR    RDSTR
ADR    <string variable>
```

where <string variable> is the name of the string variable. If the string entered by the user is too large for the specified string the V flag is returned set; otherwise it is returned clear.

Additional Notes on Keyboard Input

The RDINT, RDSTR and READLN routines all work together in the SPEED/ASM environment. Some information on how they read data from the keyboard may be of help.

RDINT reads an integer from the current position in the line input buffer and leaves the line buffer pointer pointing at the first character beyond the integer read in. If the first character RDINT attempts to read is a carriage return, then RDINT first calls READLN to read a line of text from the keyboard.

RDSTR reads a string from the input buffer starting at the current index into the line buffer. The remainder of the line (up to the car-

Listing continued.

```
0800            98  ; IFx routines.
0800            99  ;
0800           100  ;
0800           101  ;
00BD           102  EQ       EQU "="
00A3           103  NE       EQU "#"
00BE           104  GT       EQU ">"
00BC           105  LT       EQU "<"
BDBE           106  GE       EQU ">"|"="*256
BDBC           107  LE       EQU "<"|"="*256
0800           108  ;
0800           109  ;
0800           110  ;
0800           111  ;
0800           112  ;
0800           113  ;
0800           114  ;
0800           115  ************************
0800           116  * SPEED/ASM ENTRY POINTS *
0800           117  ************************
0800           118  ;
0800           119  ;
0800           120  ;
0800           121  ;
0800           122  ; NOTE: THE EQUATE OF PUTC MUST
0800           123  ; BE CHANGED IF YOU RELOCATE
0800           124  ; SPEED/ASM TO SOME LOCATION
0800           125  ; OTHER THAN $7800
0800           126  ;
0800           127  ;
0800           128  ;
7800           129  PUTC     EQU $7800
7803           130  GETC     EQU PUTC+3
7806           131  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809           132  SAPC     EQU SAGL+3          ;  "   "  "  "   "    "  "
780C           133  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F           134  READLN   EQU HOME+3
7812           135  INIT     EQU READLN+3
7815           136  FOR      EQU INIT+3
7818           137  FOR0     EQU FOR+3
781B           138  NEXT     EQU FOR0+3
781E           139  IFI      EQU NEXT+3
7821           140  IFI0     EQU IFI+3
7824           141  IFS      EQU IFI0+3
7827           142  IFS0     EQU IFS+3
782A           143  MOVE     EQU IFS0+3
782D           144  LOAD     EQU MOVE+3
7830           145  MOVS     EQU LOAD+3
7833           146  LDSTR    EQU MOVS+3
7836           147  PRINT    EQU LDSTR+3
7839           148  PRTSTR   EQU PRINT+3
783C           149  PRTINT   EQU PRTSTR+3
783F           150  RDSTR    EQU PRTINT+3
7842           151  RDINT    EQU RDSTR+3
7845           152  ONXGOTO  EQU RDINT+3
7848           153  CASE     EQU ONXGOTO+3
784B           154  CASEI    EQU CASE+3
784E           155  INSET    EQU CASEI+3
7851           156  NOTINSET EQU INSET+3
7854           157  ABS      EQU NOTINSET+3
7857           158  NEG      EQU ABS+3
785A           159  MUL      EQU NEG+3
785D           160  DIV      EQU MUL+3
7860           161  MOD      EQU DIV+3
7863           162  RND      EQU MOD+3
7866           163  SUBSTR   EQU RND+3
7869           164  INDEX    EQU SUBSTR+3
786C           165  LENGTH   EQU INDEX+3
786F           166  CONCAT   EQU LENGTH+3
7872           167  GETWZPG  EQU CONCAT+3        ;USED BY SPEED/ASM
7875           168  RDFP     EQU GETWZPG+3
7878           169  PRTE     EQU RDFP+3
787B           170  PRTF     EQU PRTE+3
787E           171  FADD     EQU PRTF+3
7881           172  FSUB     EQU FADD+3
7884           173  FMUL     EQU FSUB+3
7887           174  FDIV     EQU FMUL+3
788A           175  FLT      EQU FDIV+3
788D           176  FIX      EQU FLT+3
7890           177  FNEG     EQU FIX+3
7893           178  FADDTN   EQU FNEG+3
7896           179  FSUBTN   EQU FADDTN+3
7899           180  FTIMES   EQU FSUBTN+3
789C           181  FINTO    EQU FTIMES+3
789F           182  IFF      EQU FINTO+3
78A2           183  MOVFP    EQU IFF+3
0800           184  ;
0800           185  ;
0800           186  ;
0800           187          END

***** END OF ASSEMBLY

]brun sort
BRUN SORT
SYMBOL TABLE SORTED ALPHABETICALLY

  ABS      7854  ACH      0018  ACL      0017  AUXH     001C  AUXL     001B
```

Listing continued.

riage return) is read in and stored into the specified string. If the line buffer pointer was pointing at a carriage return when RDSTR was called, then a line of text is first read from the keyboard.

If you want to ensure that a fresh line of text is read from the keyboard before reading an integer or a string, you should call READLN *immediately* before calling RDSTR or RDINT. For example:

```
JSR   READLN
JSR   RDINT
ADR   INTGR
JSR   READLN
JSR   RDSTR
ADR   STRVAR
```

If it is inconvenient to call READLN immediately before calling RDINT or RDSTR (an example appears in the sample program, Listing 2), then store the value 0 into the SPEED/ASM GOTLN variable. (This variable is predefined in the SPEED/ASM equates.) By storing 0 (false) into GOTLN you can force the RDSTR and RDINT routines to read a new line of text the next time they are executed, such as:

```
LDA   #FALSE
STA   GOTLN
  .     .
  .     .
  .     .
JSR   RDSTR    ;A new line of text
               ;will automatically
ADR   STRING   ;be read from the
               ;keyboard.
```

String Comparisons Using IFS0 and IFS

The IFS and IFS0 routines compare two strings in SPEED/ASM. These routines are very similar to the IFI and IFI0 integer routines discussed earlier in this series. IFS compares two string variables, and IFS0 compares a string variable to a string constant.

Since IFS0 is used most often, I'll describe it first. The calling sequence is:

```
JSR   IFS0
ADR   <string variable>,<op>
BYT   "string constant",0
```

where <string variable> is the name of a properly declared string variable, "string constant" is the string you wish to compare the string vari-

Listing continued.

```
CASE    7848  CASEI   784B  CONCAT  786F  COUNT   0013  CR      008D
DESTADR 0005  DIGIT   000F  DIV     785D  EQ      00BD  FADD    787E
FADDIN  7893  FALSE   0000  FDIV    7887  FINTO   789C  FIX     788D
FLT     788A  FMUL    7884  FNEG    7890  FOR     7815  FOR0    7818
FORASAV 0000  FORXSAV 0001  FORYSAV 0002  FORZPG  0003  FSUB    7881
FSUBIN  7896  FTIMES  7899  GE      BDBE  GETC    7803  GETWZPG 7872
GOTLN   0014  GT      00BE  HOME    780C  IFF     789F  IFI     781E
IFIO    7821  IFS     7824  IFS0    7827  INDEX   7869  INIT    7812
INPUT   0200  INSET   784E  ISIMMED 0009  JMPADR  0011  LDSTR   7833
LE      BDBC  LEAD0   0010  LENGTH  786C  LINEINDX 0015 LOAD    782D
LT      00BC  MAXLEN  000C  MOD     7860  MOVE    782A  MOVFP   78A2
MOVS    7830  MUL     785A  NE      00A3  NEG     7857  NEXT    781B
NOTINSET 7851 ONXGOTO 7845  OP      000A  PRINT   7836  PROMPT  0033
PRTE    7878  PRTF    787B  PRTINT  783C  PRTSTR  7839  PTRADR  0007
PUTC    7800  RDFP    7875  RDINT   7842  RDSTR   783F  READLN  780F
RND     7863  RNDH    004F  RNDL    004E  SAGL    7806  SAPC    7809
SIGN    0016  STACK   0100  SUBSTR  7866  TRUE    0001  VALUE   000D
XTNDH   001A  XTNDL   0019

SYMBOL TABLE SORTED BY ADDRESS

FORASAV 0000  FALSE   0000  TRUE    0001  FORXSAV 0001  FORYSAV 0002
FORZPG  0003  DESTADR 0005  PTRADR  0007  ISIMMED 0009  OP      000A
MAXLEN  000C  VALUE   000D  DIGIT   000F  LEAD0   0010  JMPADR  0011
COUNT   0013  GOTLN   0014  LINEINDX 0015 SIGN    0016  ACL     0017
ACH     0018  XTNDL   0019  XTNDH   001A  AUXL    001B  AUXH    001C
PROMPT  0033  RNDL    004E  RNDH    004F  CR      008D  NE      00A3
LT      00BC  EQ      00BD  GT      00BE  STACK   0100  INPUT   0200
PUTC    7800  GETC    7803  SAGL    7806  SAPC    7809  HOME    780C
READLN  780F  INIT    7812  FOR     7815  FOR0    7818  NEXT    781B
IFI     781E  IFIO    7821  IFS     7824  IFS0    7827  MOVE    782A
LOAD    782D  MOVS    7830  LDSTR   7833  PRINT   7836  PRTSTR  7839
PRTINT  783C  RDSTR   783F  RDINT   7842  ONXGOTO 7845  CASE    7848
CASEI   784B  INSET   784E  NOTINSET 7851 ABS     7854  NEG     7857
MUL     785A  DIV     785D  MOD     7860  RND     7863  SUBSTR  7866
INDEX   7869  LENGTH  786C  CONCAT  786F  GETWZPG 7872  RDFP    7875
PRTE    7878  PRTF    787B  FADD    787E  FSUB    7881  FMUL    7884
FDIV    7887  FLT     788A  FIX     788D  FNEG    7890  FADDIN  7893
FSUBIN  7896  FTIMES  7899  FINTO   789C  IFF     789F  MOVFP   78A2
LE      BDBC  GE      BDBE
```

able to, and <op> is any of the following SPEED/ASM comparisons:

```
EQ    ;Equal
NE    ;Not equal
LT    ;Less than
GT    ;Greater than
LE    ;Less than or equal
GE    ;Greater than or equal
```

These symbolic values are provided in the SPEED/ASM equates, Listing 1.

Upon return from the IFS0 routine the 6502 zero flag is set and the accumulator contains 0 if the comparison was *not* true. If the comparison *was* true, the zero flag is returned clear and the accumulator contains 1. This feature allows you to use LISA's BTR and BFL branches to test the comparison.

The IFS routine is used to compare two SPEED/ASM string variables. The calling sequence is:

```
JSR    IFS
ADR    <str1>,<op>,<str2>
```

where <str1> and <str2> are two SPEED/ASM variables that are to be compared to one another and <op> is any of the operators listed in the previous paragraph. Examples of IFS and IFS0 appear in Listing 2.

SPEED/ASM supports four string functions: SUBSTR, INDEX, LENGTH and CONCAT. These four functions provide the basic string manipulations required by most programs.

LENGTH determines the current *dynamic* length of a string—that is, the number of characters *currently* stored in the string variable. To call LENGTH use:

```
JSR    LENGTH
ADR    <string>
```

where <string> is the name of the string variable whose length you wish

Listing 2. String examples.

```
0800            1          TTL "Listing two: String Examples"
0800            2  ;
0800            3  ;
0800            4  ;
0800            5  ***********************************
0800            6  *                                 *
0800            7  *  LISTING 2: An example of      *
0800            8  *  the various SPEED/ASM string  *
0800            9  *  handling routines.            *
0800           10  *                                 *
0800           11  ***********************************
0800           12  ;
0800           13  ;
0800           14  ;
0800           15  ;
0800           16  ;
0800           17  ;
0800           18  ;
0800           19  ;
0800           20  ; GENERAL PURPOSE EQUATES
0800           21  ;
0800           22  ; The following variables are used
0800           23  ; by the SPEED/ASM package and
0800           24  ; shouldn't be used by the SPEED/ASM
0800           25  ; programmer.
0800           26  ;
0800           27  ;
0800           28  ;
0800           29  ;
0000           30  FORASAV   EPZ 0
0001           31  FORXSAV   EPZ FORASAV+1
0002           32  FORYSAV   EPZ FORXSAV+1
0003           33  FORZPG    EPZ FORYSAV+1
0005           34  DESTADR   EPZ FORZPG+2
0007           35  PTRADR    EPZ DESTADR+2
0009           36  ISIMMED   EPZ PTRADR+2
000A           37  OP        EPZ ISIMMED+1
000C           38  MAXLEN    EPZ OP+2
```

Listing continued.

Listing continued.

```
000D            39  VALUE    EPZ MAXLEN+1
000F            40  DIGIT    EPZ VALUE+2
0010            41  LEAD0    EPZ DIGIT+1
0011            42  JMPADR   EPZ LEAD0+1
0013            43  COUNT    EPZ JMPADR+2
0014            44  GOTLN    EPZ COUNT+1
0800            45  ;
0800            46  ;
0800            47  ;
0800            48  ;
0800            49  ;
0800            50  ;
0800            51  *************
0800            52  * CONSTANTS *
0800            53  *************
0800            54  ;
0800            55  ;
0800            56  ;
0800            57  ; The following symbols are constants
0800            58  ; for the values "FALSE", "TRUE", and
0800            59  ; Carriage Return (respectively).
0800            60  ;
0800            61  ; These symbols should only appear
0800            62  ; as immediate operands to a 6502
0800            63  ; instruction or in the operand field
0800            64  ; of a pseudo-opcode like BYT.
0800            65  ;
0800            66  ;
0800            67  ;
0800            68  ;
0800            69  ;
0000            70  FALSE    EQU 0
0001            71  TRUE     EQU 1
008D            72  CR       EQU $8D
0800            73  ;
0800            74  ;
0800            75  ;
0800            76  ;
0800            77  ;
0800            78  ;
0800            79  ; "IF" STATEMENT EQUATES
0800            80  ;
0800            81  ; The following symbols should only
0800            82  ; be used in the ADR pseudo-opcode
0800            83  ; following a call to the SPEED/ASM
0800            84  ; IFx routines.
0800            85  ;
0800            86  ;
0800            87  ;
00BD            88  EQ       EQU "="
00A3            89  NE       EQU "#"
00BE            90  GT       EQU ">"
00BC            91  LT       EQU "<"
BDBE            92  GE       EQU ">"|"="*256
BDBC            93  LE       EQU "<"|"="*256
0800            94  ;
0800            95  ;
0800            96  ;
0800            97  ;
0800            98  ;
0800            99  ;
0800           100  ;
0800           101  *************************
0800           102  * SPEED/ASM ENTRY POINTS *
0800           103  *************************
0800           104  ;
0800           105  ;
0800           106  ;
0800           107  ;
0800           108  ; NOTE: THE EQUATE OF PUTC MUST
0800           109  ; BE CHANGED IF YOU RELOCATE
0800           110  ; SPEED/ASM TO SOME LOCATION
0800           111  ; OTHER THAN $7800
0800           112  ;
0800           113  ;
0800           114  ;
7800           115  PUTC     EQU $7800
7803           116  GETC     EQU PUTC+3
7806           117  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809           118  SAPC     EQU SAGL+3          ;  "   "  "  "   "     "  "
780C           119  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F           120  READLN   EQU HOME+3
7812           121  INIT     EQU READLN+3
7815           122  FOR      EQU INIT+3
7818           123  FOR0     EQU FOR+3
781B           124  NEXT     EQU FOR0+3
781E           125  IFI      EQU NEXT+3
7821           126  IFI0     EQU IFI+3
7824           127  IFS      EQU IFI0+3
7827           128  IFS0     EQU IFS+3
782A           129  MOVE     EQU IFS0+3
782D           130  LOAD     EQU MOVE+3
7830           131  MOVS     EQU LOAD+3
7833           132  LDSTR    EQU MOVS+3
7836           133  PRINT    EQU LDSTR+3
7839           134  PRTSTR   EQU PRINT+3
783C           135  PRTINT   EQU PRTSTR+3
783F           136  RDSTR    EQU PRTINT+3
```

Listing continued.

to find. Since the maximum length of a string is 255 characters (which fits into 1 byte) the length is returned in the 6502 accumulator. This enables you to easily compare the length to an immediate value like:

```
JSR   LENGTH
ADR   STRING
CMP   #55
BGE   STRGE55
```

If you want to store this value into a SPEED/ASM variable, you should

"The SUBSTR routine extracts a portion of a string and stores this substring into another string variable."

store the accumulator into the low-order byte of the SPEED/ASM variable and store 0 into the high-order byte of the SPEED/ASM variable, as in this sequence:

```
JSR   LENGTH
ADR   STRING
STA   STRLEN
LDA   #0
STA   STRLEN+1
```

Note that the LENGTH routine returns the current dynamic length of the specified string. If you're interested in obtaining the maximum length of the string, simply load the accumulator with the first location of the string. For example:

```
LDA STRING ;Fetches maximum length of
           ;string.
```

The SUBSTR routine extracts a portion of a string and stores this substring into another string variable. The calling sequence for SUBSTR is:

```
JSR   SUBSTR
ADR   <source>,<index>,
      <length>,<destination>
```

where <source> is the name of the source string, <index> is the name of a SPEED/ASM integer variable containing an index into the string where the first character of the substring begins, <length> is the name

Listing continued.

```
7842            137  RDINT    EQU RDSTR+3
7845            138  ONXGOTO  EQU RDINT+3
7848            139  CASE     EQU ONXGOTO+3
784B            140  CASEI    EQU CASE+3
784E            141  INSET    EQU CASEI+3
7851            142  NOTINSET EQU INSET+3
7854            143  ABS      EQU NOTINSET+3
7857            144  NEG      EQU ABS+3
785A            145  MUL      EQU NEG+3
785D            146  DIV      EQU MUL+3
7860            147  MOD      EQU DIV+3
7863            148  RND      EQU MOD+3
7866            149  SUBSTR   EQU RND+3
7869            150  INDEX    EQU SUBSTR+3
786C            151  LENGTH   EQU INDEX+3
786F            152  CONCAT   EQU LENGTH+3
0800            153  ;
0800            154  ;
0800            155  ;
0800            156  ;
0800            157  ****************************
0800            158  ;
0800            159  ;
0800            160  ; Begin of program 1.2.
0800            161  ;
FF69            162  EXIT     EQU $FF69
0800            163  ;
0800            164  ;
0800 20 12 78   165  START    JSR INIT            ;Always do this first!
0803            166  ;
0803            167  ;
0803 20 0C 78   168           JSR HOME
0806 20 36 78   169           JSR PRINT
0809 8D 8D D3   170           BYT CR,CR,"SPEED/ASM String Routine Examples",CR,CR,0
080C D0 C5 C5
080F C4 AF C1
0812 D3 CD A0
0815 D3 F4 F2
0818 E9 EE E7
081B A0 D2 EF
081E F5 F4 E9
0821 EE E5 A0
0824 C5 F8 E1
0827 ED F0 EC
082A E5 F3 8D
082D 8D 00
082F            171  ;
082F            172  ;
082F            173  ***********************************
082F            174  *
082F            175  * The following call to PRTSTR
082F            176  * outputs a string that was pre-
082F            177  * initialized by LISA.
082F            178  *
082F            179  ***********************************
082F            180  ;
082F            181  ;
082F            182  ;
082F 20 39 78   183           JSR PRTSTR
0832 F2 0B      184           ADR STRVAR1
0834 A9 8D      185           LDA #CR
0836 20 00 78   186           JSR PUTC
0839            187  ;
0839            188  ;
0839            189  ***********************************
0839            190  *
0839            191  * The following section of code
0839            192  * reads a new line of text from
0839            193  * the keyboard and stuffs it into
0839            194  * the STRVAR2 variable.  If the
0839            195  * string was too long, the V flag
0839            196  * is returned set and the user is
0839            197  * prompted to re-enter the data.
0839            198  *
0839            199  ***********************************
0839            200  ;
0839            201  ;
0839 20 0F 78   202  ENTERSTR JSR READLN          ;Force a new line from the kbd
083C 20 3F 78   203           JSR RDSTR
083F 13 0C      204           ADR STRVAR2
0841 50 28      205           BVC GOODSTR         ;If no error.
0843            206  ;
0843 20 36 78   207           JSR PRINT
0846 8D C5 F2   208           BYT CR,"Error: String too long",CR
0849 F2 EF F2
084C BA A0 D3
084F F4 F2 E9
0852 EE E7 A0
0855 F4 EF EF
0858 A0 EC EF
085B EE E7 8D
085E D2 E5 E5   209           BYT "Reenter: ",0
0861 EE F4 E5
0864 F2 BA A0
0867 00
0868 4C 39 08   210           JMP ENTERSTR
086B            211  ;
086B            212  ;
086B            213  ; Print the string the user just entered.
```

Listing continued.

of a SPEED/ASM variable that contains the length of the substring to be extracted, and <destination> is the name of the string where the substring is to be stored. It is important to reiterate that <index> and <length> are the names of SPEED/ASM variables that contain the respective values, not simply the values themselves. If you try to place the value in the position for the index or length parameters, strange things will happen.

> **"If you try to place the value in the position for the index or length parameters, strange things will happen."**

If the substring you extract from the source string is too large to fit in the destination string, SUBSTR returns the V flag set; otherwise the V flag is returned clear. An example of the SUBSTR routine appears in Listing 2.

The INDEX routine lets you check if one string can be found within a larger one. That is, if one string is a substring of another, INDEX returns a value giving you the index of the source string within the second string. The syntax for using INDEX is:

```
JSR   INDEX
ADR   <key>,<source>
```

where <key> is the string you want to search for in the <source> string. If the string is found, the index into <source> is returned in the 6502 accumulator. If <key> is not present in <source>, then 0 is returned in the accumulator. Listing 2 includes some examples of how to use INDEX.

The final SPEED/ASM string manipulation routine is CONCAT. As you will surmise, it concatenates two strings. The calling sequence is:

```
JSR   CONCAT
ADR   <SRC1>,<SRC2>,<DEST>
```

where <SRC1> and <SRC2> are the two source strings to be concatenated

and <DEST> is the destination string where the result is to be stored. <DEST> should not be the same string as <SRC1> or <SRC2>. If the concatenated result is too large to fit into the destination string, it is truncated and CONCAT returns with the V flag set. If the concatenation was performed correctly the V flag is returned clear. Examples of CONCAT appear in Listing 2.

Putting It All Together

This month's example program (Listing 3) is the beginning of a simple database/mailing list program using all the SPEED/ASM constructs I've presented up to this point. I'll expand on this program next month, so if it seems rather incomplete that's only because it is.

I have now surveyed all the string handling routines provided by the SPEED/ASM package. While these routines provide most of the capabilities you'll require, some "pure" 6502 code is often necessary to make a program run smoothly. Next month, I'll describe the 6502 indexed and indirect addressing modes so you'll have all the string capabilities you could possibly want. ■

Listing continued.

```
086B                    214  ;
086B 20 39 78           215  GOODSTR  JSR PRISTR
086E 13 0C              216           ADR STRVAR2
0870                    217  ;
0870                    218  ;
0870                    219  ***********************************
0870                    220  *
0870                    221  * The following section of code
0870                    222  * loads STRVAR3 with a string
0870                    223  * constant ("This is a string")
0870                    224  * and then copies the STRVAR3
0870                    225  * variable into STRVAR4.  After
0870                    226  * all the loading and copying is
0870                    227  * performed, the contents of the
0870                    228  * strings are displayed.
0870                    229  *
0870                    230  ***********************************
0870                    231  ;
0870                    232  ;
0870                    233  ;
0870                    234  ; First demonstrate that the strings
0870                    235  ; contain nothing.
0870                    236  ;
0870 20 36 78           237           JSR PRINT
0873 8D D3 D4           238           BYT CR,"STRVAR3 contains """,0
0876 D2 D6 C1
0879 D2 B3 A0
087C E3 EF EE
087F F4 E1 E9
0882 EE F3 A0
0885 A2 00
0887 20 39 78           239           JSR PRISTR
088A 95 0C              240           ADR STRVAR3
088C 20 36 78           241           JSR PRINT
088F A2 8D D3           242           BYT """",CR,"STRVAR4 contains """,0
0892 D4 D2 D6
0895 C1 D2 B4
0898 A0 E3 EF
089B EE F4 E1
089E E9 EE F3
08A1 A0 A2 00
08A4 20 39 78           243           JSR PRISTR
08A7 B5 0C              244           ADR STRVAR4
08A9                    245  ;
08A9                    246  ; Now initialize them with LDSTR and MOVS
08A9                    247  ;
08A9                    248  ;
08A9 20 36 78           249           JSR PRINT
08AC 8D D5 F3           250           BYT CR,"Using LDSTR to initialize STRVAR3",CR,0
08AF E9 EE E7
08B2 A0 CC C4
08B5 D3 D4 D2
08B8 A0 F4 EF
08BB A0 E9 EE
08BE E9 F4 E9
08C1 E1 BC E9
08C4 FA E5 A0
08C7 D3 D4 D2
08CA D6 C1 D2
08CD D3 D4 D2     08CD: B3 8D 00
08D0 D6 C1 D2
08D3 E3 8D 00
08D0 20 33 78           251           JSR LDSTR
08D3 95 0C              252           ADR STRVAR3
08D5 D4 E8 E9           253           BYT "This is a string",0
08D8 F3 A0 E9
08DB F3 A0 E1
08DE A0 F3 F4
08E1 F2 E9 EE
08E4 E7 00
08E6                    254  ;
08E6                    255  ;
08E6                    256  ; Print STRVAR3 and STRVAR4 to show
08E6                    257  ; that they are indeed different.
08E6                    258  ;
08E6 20 36 78           259           JSR PRINT
08E9 8D D3 D4           260           BYT CR,"STRVAR3 contains """,0
08EC D2 D6 C1
08EF D2 B3 A0
08F2 E3 EF EE
08F5 F4 E1 E9
08F8 EE F3 A0
08FB A2 00
08FD 20 39 78           261           JSR PRISTR
0900 95 0C              262           ADR STRVAR3
0902 20 36 78           263           JSR PRINT
0905 A2 8D D3           264           BYT """",CR,"STRVAR4 contains """,0
0908 D4 D2 D6
090B C1 D2 B4
090E A0 E3 EF
0911 EE F4 E1
0914 E9 EE F3
0917 A0 A2 00
091A 20 39 78           265           JSR PRISTR
091D B5 0C              266           ADR STRVAR4
091F                    267  ;
```

Listing continued.

Listing continued.

```
091F                    268  ;
091F                    269  ; Now copy STRVAR3 into STRVAR4 and
091F                    270  ; print the two strings to show
091F                    271  ; that they are now the same.
091F                    272  ;
091F                    273  ;
091F 20 36 78           274          JSR PRINT
0922 8D 8D D5           275          BYT CR,CR,"Using MOVS to copy STRVAR3
                                     into STRVAR4",CR,0
0925 F3 E9 EE
0928 E7 A0 CD
092B CF D6 D3
092E A0 F4 EF
0931 A0 E3 EF
0934 F0 F9 A0
0937 D3 D4 D2
093A D6 C1 D2
093D B3 A0 E9
0940 EE F4 EF
0943 A0 D3 D4
0946 D2 D6 C1
0949 D2 B4 8D
094C B3 A0 E9           094C:00
094F EE F4 EF
0952 A0 D3 D4
0955 D2 D6 C1
0958 D2 B4 8D
095B 00
094D 20 30 78           276          JSR MOVS
0950 95 0C B5           277          ADR STRVAR3,STRVAR4
0953 0C
0954                    278  ;
0954                    279  ;
0954 20 36 78           280          JSR PRINT
0957 8D D3 D4           281          BYT CR,"STRVAR3 contains """,0
095A D2 D6 C1
095D D2 E3 A0
0960 E3 EF EE
0963 F4 E1 E9
0966 EE F3 A0
0969 A2 00
096B 20 39 78           282          JSR PRTSTR
096E 95 0C              283          ADR STRVAR3
0970 20 36 78           284          JSR PRINT
0973 A2 8D D3           285          BYT """",CR,"STRVAR4 contains """,0
0976 D4 D2 D6
0979 C1 D2 B4
097C A0 E3 EF
097F EE F4 E1
0982 E9 EE F3
0985 A0 A2 00
0988 20 39 78           286          JSR PRTSTR
098B B5 0C              287          ADR STRVAR4
098D                    288  ;
098D                    289  ;
098D                    290  ;
098D                    291  **********************************
098D                    292  *
098D                    293  * The following section of code
098D                    294  * demonstrates the IFS0 routine.
098D                    295  * It compares the contents of
098D                    296  * STRVAR3 to the literal "XXXXX"
098D                    297  *
098D                    298  **********************************
098D                    299  ;
098D                    300  ;
098D 20 36 78           301          JSR PRINT
0990 A2 8D 8D           302          BYT """",CR,CR,"STRVAR3 is ",0
0993 D3 D4 D2
0996 D6 C1 D2
0999 E3 A0 E9
099C F3 A0 00
099F 20 27 78           303          JSR IFS0
09A2 95 0C BD           304          ADR STRVAR3,EQ
09A5 00
09A6 D8 D8 D8           305          BYT "XXXXX",0
09A9 D8 D8 00
09AC F0 08              306          BFL >0
09AE A9 BD              307          LDA #"="
09B0 20 00 78           308          JSR PUTC
09B3 4C BC 09           309          JMP TESTDONE
09B6                    310  ;
09B6 20 36 78           311  ^0      JSR PRINT
09B8 BC BE 00           312          BYT "<>",0
09BC                    313  ;
09BC 20 36 78           314  TESTDONE JSR PRINT
09BF A0 A2 D8           315          BYT " ""XXXXX""",CR,0
09C2 D8 D8 D8
09C5 D8 A2 8D
09C8 00
09C9                    316  ;
09C9                    317  ;
09C9                    318  **********************************
09C9                    319  *
09C9                    320  * This section of code demon-
09C9                    321  * strates the IFS routine.  It
09C9                    322  * loads STRVAR4 with the string
09C9                    323  * literal "This is also a string"
09C9                    324  * and compares it to STRVAR3.
09C9                    325  *
09C9                    326  **********************************
09C9                    327  ;
09C9                    328  ;
09C9                    329  ; Load STRVAR4 with "This is also a string"
09C9                    330  ;
09C9                    331  ;
09C9 20 33 78           332          JSR LDSTR
09CC B5 0C              333          ADR STRVAR4
09CE D4 E8 E9           334          BYT "This is also a string",0
09D1 F3 A0 E9
09D4 F3 A0 E1
09D7 EC F3 EF
09DA A0 E1 A0
09DD F3 F4 F2
09E0 E9 EE E7
09E3 00
09E4                    335  ;
09E4                    336  ; Print the contents of the two
09E4                    337  ; strings.
09E4                    338  ;
09E4                    339  ;
09E4 20 36 78           340          JSR PRINT
09E7 8D 8D D3           341          BYT CR,CR,"STRVAR3 contains: """,0
09EA D4 D2 D6
09ED C1 D2 E3
09F0 A0 E3 EF
09F3 EE F4 E1
09F6 E9 EE F3
09F9 BA A0 A2
09FC 00
09FD 20 39 78           342          JSR PRTSTR
0A00 95 0C              343          ADR STRVAR3
0A02 20 36 78           344          JSR PRINT
0A05 A2 8D D3           345          BYT """",CR,"STRVAR4 contains: """,0
0A08 D4 D2 D6
0A0B C1 D2 B4
0A0E A0 E3 EF
0A11 EE F4 E1
0A14 E3                 0A14:E9 EE F3 BA A0 A2 00
0A1B 20 39 78           346          JSR PRTSTR
0A1E B5 0C              347          ADR STRVAR4
0A20                    348  ;
0A20 20 36 78           349          JSR PRINT
0A23 A2 8D 00           350          BYT """",CR,0
0A26                    351  ;
0A26                    352  ;
0A26                    353  ; Compare the two strings and see
0A26                    354  ; if they are equal.
0A26                    355  ;
0A26                    356  ;
0A26 20 24 78           357          JSR IFS
0A29 95 0C BD           358          ADR STRVAR3,EQ,STRVAR4
0A2C 00 B5 0C
0A2F F0 16              359          BFL >0
0A31 20 36 78           360          JSR PRINT
0A34 8D D3 D4           361          BYT CR,"STRVAR3 = STRVAR4",0
0A37 D2 D6 C1
0A3A D2 E3 A0
0A3D BD A0 D3
0A40 D4 D2 D6
0A43 C1 D2 B4
0A46 00
0A47                    362  ;
0A47                    363  ; Compare the two strings and see
0A47                    364  ; if they are not equal.
0A47                    365  ;
0A47                    366  ;
0A47 20 24 78           367  ^0      JSR IFS
0A4A 95 0C A3           368          ADR STRVAR3,NE,STRVAR4
0A4D 00 B5 0C
0A50 F0 17              369          BFL >1
0A52 20 36 78           370          JSR PRINT
0A55 8D D3 D4           371          BYT CR,"STRVAR3 <> STRVAR4",0
0A58 D2 D6 C1
0A5B D2 E3 A0
0A5E BC BE A0
0A61 D3 D4 D2
0A64 D6 C1 D2
0A67 B4 00
0A69                    372  ;
0A69                    373  ;
0A69                    374  ; Compare the two strings to see
0A69                    375  ; if STRVAR3 < STRVAR4.
0A69                    376  ;
0A69 20 24 78           377  ^1      JSR IFS
0A6C 95 0C BC           378          ADR STRVAR3,LT,STRVAR4
0A6F 00 B5 0C
0A72 F0 16              379          BFL >2
0A74 20 36 78           380          JSR PRINT
0A77 8D D3 D4           381          BYT CR,"STRVAR3 < STRVAR4",0
0A7A D2 D6 C1
0A7D D2 E3 A0
0A80 BC A0 D3
0A83 D4 D2 D6
0A86 C1 D2 B4
0A89 00
0A8A                    382  ;
0A8A                    383  ;
0A8A                    384  ; Check for STRVAR3 <= STRVAR4
0A8A                    385  ;
0A8A 20 24 78           386  ^2      JSR IFS
0A8D 95 0C BC           387          ADR STRVAR3,LE,STRVAR4
0A90 BD B5 0C
0A93 F0 17              388          BFL >3
0A95 20 36 78           389          JSR PRINT
```

Listing continued.

Listing continued.

```
0A98 8D D3 D4   390          BYT CR,"SIRVAR3 <= SIRVAR4",0
0A9B D2 D6 C1
0A9E D2 E3 A0
0AA1 BC BD A0
0AA4 D3 D4 D2
0AA7 D6 C1 D2
0AAA B4 00
0AAC            391  ;
0AAC            392  ;
0AAC            393  ;
0AAC            394  ; Check to see if SIRVAR3 > SIRVAR4.
0AAC            395  ;
0AAC 20 24 78   396  ^3       JSR IFS
0AAF 95 0C BE   397          ADR SIRVAR3,GT,SIRVAR4
0AB2 00 B5 0C
0AB5 F0 16      398          BFL >4
0AB7 20 36 78   399          JSR PRINT
0ABA 8D D3 D4   400          BYT CR,"SIRVAR3 > SIRVAR4",0
0ABD D2 D6 C1
0AC0 D2 E3 A0
0AC3 BE A0 D3
0AC6 D4 D2 D6
0AC9 C1 D2 B4
0ACC 00
0ACD            401  ;
0ACD            402  ;
0ACD            403  ;
0ACD            404  ; Is SIRVAR3 >= SIRVAR4?
0ACD            405  ;
0ACD 20 24 78   406  ^4       JSR IFS
0AD0 95 0C BE   407          ADR SIRVAR3,GE,SIRVAR4
0AD3 BD B5 0C
0AD6 F0 17      408          BFL >5
0AD8 20 36 78   409          JSR PRINT
0ADB 8D D3 D4   410          BYT CR,"SIRVAR3 >= SIRVAR4",0
0ADE D2 D6 C1
0AE1 D2 E3 A0
0AE4 BE BD A0
0AE7 D3 D4 D2
0AEA D6 C1 D2
0AED B4 00
0AEF            411  ;
0AEF A9 8D      412  ^5       LDA #CR
0AF1 20 00 78   413          JSR PUTC
0AF4            414  ;
0AF4            415  ;
0AF4            416  ;
0AF4            417  ;
0AF4            418  ***********************************
0AF4            419  *
0AF4            420  * Demonstrate the LENGTH routine
0AF4            421  *
0AF4            422  ***********************************
0AF4            423  ;
0AF4            424  ;
0AF4 20 6C 78   425          JSR LENGTH
0AF7 13 0C      426          ADR SIRVAR2
0AF9 8D 17 0D   427          STA LENSIR2
0AFC A9 00      428          LDA #0
0AFE 8D 18 0D   429          STA LENSIR2+1
0B01 20 36 78   430          JSR PRINT
0B04 D4 E8 E5   431          BYT "The length of SIRVAR2 is ",0
0B07 A0 EC E5
0B0A EE E7 F4
0B0D E8 A0 EF
0B10 E6 A0 D3
0B13 D4 D2 D6
0B16 C1 D2 E2
0B19 A0 E9 F3
0B1C A0 00
0B1E 20 3C 78   432          JSR PRTINT
0B21 17 0D      433          ADR LENSIR2
0B23            434  ;
0B23            435  ;
0B23            436  ;
0B23 A9 8D      437          LDA #CR
0B25 20 00 78   438          JSR PUTC
0B28            439  ;
0B28            440  ;
0B28            441  ;
0B28            442  ;
0B28            443  ***********************************
0B28            444  *
0B28            445  * The following code demonstrates
0B28            446  * the use of the SPEED/ASM SUBSTR
0B28            447  * routine.
0B28            448  *
0B28            449  *  This code emulates the BASIC
0B28            450  * statements:
0B28            451  *
0B28            452  *
0B28            453  * 10 FOR I = 1 TO 10
0B28            454  * 20 J = 6-I
0B28            455  * 30 PRINT MID$(A$,I,J)
0B28            456  * 40 PRINT MID$(A$,1,I)
0B28            457  * 50 NEXT I
0B28            458  *
0B28 20 18 78   459          JSR FOR0
0B2B 13 0D 01   460          ADR I,1,5
0B2E 00 05 00
0B31            461  ;
0B31 38         462          SEC
0B32 A9 06      463          LDA #6
0B34 ED 13 0D   464          SBC I
```

Listing continued.

Listing continued.

```
0B37 8D 15 0D   465          STA J
0B3A A9 00      466          LDA /6
0B3C ED 14 0D   467          SBC I+1
0B3F 8D 16 0D   468          STA J+1
0B42            469  ;
0B42 20 66 78   470          JSR SUBSTR
0B45 95 0C 13   471          ADR STRVAR3,I,J,STRVAR4
0B48 0D 15 0D
0B4B B5 0C
0B4D 20 39 78   472          JSR PRTSTR
0B50 B5 0C      473          ADR STRVAR4
0B52 A9 8D      474          LDA #CR
0B54 20 00 78   475          JSR PUTC
0B57            476  ;
0B57 20 66 78   477          JSR SUBSTR
0B5A 95 0C F0   478          ADR STRVAR3,ONE,I,STRVAR4
0B5D 0B 13 0D
0B60 B5 0C
0B62 20 39 78   479          JSR PRTSTR
0B65 B5 0C      480          ADR STRVAR4
0B67            481  ;
0B67 A9 8D      482          LDA #CR
0B69 20 00 78   483          JSR PUTC
0B6C 20 1B 78   484          JSR NEXT
0B6F            485  ;
0B6F            486  ;
0B6F            487  ;
0B6F            488  **********************************
0B6F            489  *
0B6F            490  * Test the INDEX routine:  This
0B6F            491  * code loads STRVAR4 with the
0B6F            492  * string "string" and searches for
0B6F            493  * "string" within STRVAR3.
0B6F            494  *
0B6F            495  **********************************
0B6F            496  ;
0B6F            497  ;
0B6F 20 33 78   498          JSR LDSTR
0B72 B5 0C      499          ADR STRVAR4
0B74 F3 F4 F2   500          BYT "string",0
0B77 E9 EE E7
0B7A 00
0B7B            501  ;
0B7B 20 69 78   502          JSR INDEX
0B7E B5 0C 95   503          ADR STRVAR4,STRVAR3
0B81 0C
0B82            504  ;
0B82 8D 19 0D   505          STA STRINDX          ;Store L.O. index
                                                  into STRINDX
0B85 A9 00      506          LDA #0               ;Set H.O. byte of
                                                  index to zero
0B87 8D 1A 0D   507          STA STRINDX+1
0B8A            508  ;
0B8A            509  ; Print the results:
0B8A            510  ;
0B8A 20 36 78   511          JSR PRINT
0B8D 8D 8D A2   512          BYT CR,CR,"""string"" was found at
                             position: ",0
0B90 F3 F4 F2
0B93 E9 EE E7
0B96 A2 A0 F7
0B99 E1 F3 A0
0B9C E6 EF F5
0B9F EE E4 A0
0BA2 E9 F4 E9   0BA2:E1 F4 A0 F0 EF F3 E9 F4 E9 EF EE BA A0 00
0BA5 EF EE BA
0BA8 A0 00
0BB0 20 3C 78   513          JSR PRTINT
0BB3 19 0D      514          ADR STRINDX
0BB5 A9 8D      515          LDA #CR
0BB7 20 00 78   516          JSR PUTC
0BBA            517  ;
0BBA            518  ;
0BBA            519  **********************************
0BBA            520  *
0BBA            521  * Demonstrate the CONCAT routine.
0BBA            522  *
0BBA            523  *  This code concatenates STRVAR4
0BBA            524  * (which contains "string") to
0BBA            525  * STRVAR3 and prints the result.
0BBA            526  *
0BBA            527  **********************************
0BBA            528  ;
0BBA            529  ;
0BBA 20 6F 78   530          JSR CONCAT
0BBD 95 0C B5   531          ADR STRVAR3,STRVAR4,STRVAR5
0BC0 0C D5 0C
0BC3            532  ;
0BC3 20 36 78   533          JSR PRINT
0BC6 8D C3 CF   534          BYT CR,"CONCAT(STRVAR3,STRVAR4)= """,0
0BC9 CE C3 C1
0BCC D4 A8 D3
0BCF D4 D2 D6
0BD2 C1 D2 E3
0BD5 AC D3 D4
0BD8 D2 D6 C1
0BDB D2 B4 A9
0BDE BD A0 A2
0BE1 00
0BE2 20 39 78   535          JSR PRTSTR
0BE5 D5 0C      536          ADR STRVAR5
0BE7            537  ;
0BE7 20 36 78   538          JSR PRINT
0BEA A2 8D 00   539          BYT """",CR,0
```

Listing continued.

Listing continued.

```
0BED                    540  ;
0BED                    541  ;
0BED                    542  ;
0BED 4C 69 FF           543          JMP EXIT
0BF0                    544  ;
0BF0 01 00              545  ONE      ADR 1
0BF2                    546  ;
0BF2 1F D4 E8           547  STRVAR1  STR "This string was pre-initialized"
0BF5 E9 F3 A0
0BF8 F3 F4 F2
0BFB E9 EE E7
0BFE A0 F7 E1
0C01 F3 A0 F0
0C04 F2 E5 AD
0C07 E9 EE E9
0C0A F4 E9 E1
0C0D EC E9 FA
0C10 E5 E4
0C12 00                 548          BYT 0
0C13                    549  ;
0C13 80 00              550  STRVAR2  ADR 128
0C15                    551          DFS 128
0C95                    552  ;
0C95                    553  ;
0C95 1E 00              554  STRVAR3  ADR 30
0C97                    555          DFS 30
0CB5                    556  ;
0CB5 1E 00              557  STRVAR4  ADR 30
0CB7                    558          DFS 30
0CD5                    559  ;
0CD5 3C 00              560  STRVAR5  ADR 60
0CD7                    561          DFS 60
0D13                    562  ;
0D13 00 00              563  I        ADR 0
0D15 00 00              564  J        ADR 0
0D17 00 00              565  LENSTR2  ADR 0
0D19 00 00              566  STRINDX  ADR 0
0D1B                    567  ;
0D1B                    568  ;
0D1B                    569  ;
0D1B                    570          END

***** END OF ASSEMBLY

Ibrun sort
BRUN SORT
SYMBOL TABLE SORTED ALPHABETICALLY

  ABS      7854  CASE     7848  CASEI    784B  CONCAT   786F  COUNT    0013
  CR       008D  DESTADR  0005  DIGIT    000F  DIV      785D  ENTERSTR 0839
  EQ       00BD  EXIT     FF69  FALSE    0000  FOR      7815  FOR0     7818
  FORASAV  0000  FORXSAV  0001  FORYSAV  0002  FORZPG   0003  GE       BDBE
  GETC     7803  GOODSTR  086B  GOTLN    0014  GT       00BE  HOME     780C
  I        0D13  IFI      781E  IFI0     7821  IFS      7824  IFS0     7827
  INDEX    7869  INIT     7812  INSET    784E  ISIMMED  0009  J        0D15
  JMPADR   0011  LDSTR    7833  LE       BDBC  LEAD0    0010  LENGTH   786C
  LENSTR2  0D17  LOAD     782D  LT       00BC  MAXLEN   000C  MOD      7860
  MOVE     782A  MOVS     7830  MUL      785A  NE       00A3  NEG      7857
  NEXT     781B  NOTINSET 7851  ONE      0BF0  ONXGOTO  7845  OP       000A
  PRINT    7836  PRTINT   783C  PRTSTR   7839  PTRADR   0007  PUTC     7800
  RDINT    7842  RDSTR    783F  READLN   780F  RND      7863  SAGL     7806
  SAPC     7809  START    0800  STRINDX  0D19  STRVAR1  0BF2  STRVAR2  0C13
  STRVAR3  0C95  STRVAR4  0CB5  STRVAR5  0CD5  SUBSTR   7866  TESTDONE 09BC
  TRUE     0001  VALUE    000D

SYMBOL TABLE SORTED BY ADDRESS

  FORASAV  0000  FALSE    0000  TRUE     0001  FORXSAV  0001  FORYSAV  0002
  FORZPG   0003  DESTADR  0005  PTRADR   0007  ISIMMED  0009  OP       000A
  MAXLEN   000C  VALUE    000D  DIGIT    000F  LEAD0    0010  JMPADR   0011
  COUNT    0013  GOTLN    0014  CR       008D  NE       00A3  LT       00BC
  EQ       00BD  GT       00BE  START    0800  ENTERSTR 0839  GOODSTR  086B
  TESTDONE 09BC  ONE      0BF0  STRVAR1  0BF2  STRVAR2  0C13  STRVAR3  0C95
  STRVAR4  0CB5  STRVAR5  0CD5  I        0D13  J        0D15  LENSTR2  0D17
  STRINDX  0D19  PUTC     7800  GETC     7803  SAGL     7806  SAPC     7809
  HOME     780C  READLN   780F  INIT     7812  FOR      7815  FOR0     7818
  NEXT     781B  IFI      781E  IFI0     7821  IFS      7824  IFS0     7827
  MOVE     782A  LOAD     782D  MOVS     7830  LDSTR    7833  PRINT    7836
  PRTSTR   7839  PRTINT   783C  RDSTR    783F  RDINT    7842  ONXGOTO  7845
  CASE     7848  CASEI    784B  INSET    784E  NOTINSET 7851  ABS      7854
  NEG      7857  MUL      785A  DIV      785D  MOD      7860  RND      7863
  SUBSTR   7866  INDEX    7869  LENGTH   786C  CONCAT   786F  LE       BDBC
  GE       BDBE  EXIT     FF69
```

Listing 3. SPEED/ASM Mini-Mailer.

```
0800                     1          TTL "Listing Three: S/A Mini-Mailer"
0800                     2  ;
0800                     3  ; Listing 3
0800                     4  ;
0800                     5  ;
0800                     6  ;
0800                     7  ; WARNING!!! This program is intended for demonstration
0800                     8  ; purposes only.  It has not been tested well enough
0800                     9  ; to qualify it as a commercial product.
0800                    10  ; Neither Lazerware nor Randy Hyde can assume any
0800                    11  ; responsibility for the use of this program.
0800                    12  ;
0800                    13  ;
0800                    14  ;
0800                    15  ;
0800                    16  ***********************************
0800                    17  *                                 *
0800                    18  *       SPEED/ASM Equates         *
0800                    19  *                                 *
0800                    20  ***********************************
0800                    21  ;
0800                    22  ;
0800                    23  ;
0800                    24  ;
0800                    25  ;
0800                    26  ;
0800                    27  *************
0800                    28  * CONSTANTS *
0800                    29  *************
0800                    30  ;
0000                    31  FALSE    EQU 0
0001                    32  TRUE     EQU 1
008D                    33  CR       EQU $8D
0087                    34  BELL     EQU $87
0084                    35  CTLD     EQU $84
0800                    36  ;
0800                    37  ;
00BD                    38  EQ       EQU "="
00A3                    39  NE       EQU "#"
00BE                    40  GT       EQU ">"
00BC                    41  LT       EQU "<"
BDBE                    42  GE       EQU ">"|"="*256
BDBC                    43  LE       EQU "<"|"="*256
0800                    44  ;
0800                    45  ;
0800                    46  ;
0800                    47  ;
0800                    48  ;
0800                    49  ;
0800                    50  ;
0800                    51  ***************************
0800                    52  * SPEED/ASM ENTRY POINTS *
0800                    53  ***************************
0800                    54  ;
7800                    55  PUTC     EQU $7800
7803                    56  GETC     EQU PUTC+3
7806                    57  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY-
                                                          SEE DOC.
7809                    58  SAPC     EQU SAGL+3          ;  "   "  "  "   "
                                                            "   "
780C                    59  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F                    60  READLN   EQU HOME+3
7812                    61  INIT     EQU READLN+3
7815                    62  FOR      EQU INIT+3
7818                    63  FOR0     EQU FOR+3
781B                    64  NEXT     EQU FOR0+3
781E                    65  IFI      EQU NEXT+3
7821                    66  IFI0     EQU IFI+3
7824                    67  IFS      EQU IFI0+3
7827                    68  IFS0     EQU IFS+3
782A                    69  MOVE     EQU IFS0+3
782D                    70  LOAD     EQU MOVE+3
7830                    71  MOVS     EQU LOAD+3
7833                    72  LDSTR    EQU MOVS+3
7836                    73  PRINT    EQU LDSTR+3
7839                    74  PRTSTR   EQU PRINT+3
783C                    75  PRTINT   EQU PRTSTR+3
783F                    76  RDSTR    EQU PRTINT+3
7842                    77  RDINT    EQU RDSTR+3
7845                    78  ONXGOTO  EQU RDINT+3
7848                    79  CASE     EQU ONXGOTO+3
784B                    80  CASEI    EQU CASE+3
784E                    81  INSET    EQU CASEI+3
7851                    82  NOTINSET EQU INSET+3
7854                    83  ABS      EQU NOTINSET+3
7857                    84  NEG      EQU ABS+3
785A                    85  MUL      EQU NEG+3
785D                    86  DIV      EQU MUL+3
7860                    87  MOD      EQU DIV+3
7863                    88  RND      EQU MOD+3
7866                    89  SUBSTR   EQU RND+3
7869                    90  INDEX    EQU SUBSTR+3
786C                    91  LENGTH   EQU INDEX+3
786F                    92  CONCAT   EQU LENGTH+3
0800                    93  ;
0800                    94  ; Apple Monitor equates.
0800                    95  ;
FF69                    96  EXIT     EQU $FF69
0800                    97  ;
0800                    98  ;
0800                    99  ;
0800                   100  ;
0800                   101  ***********************************
0800                   102  ;
0800                   103  ;
0800                   104  ; SPEED/ASM mini-mailing list pgm.
0800                   105  ;
0800                   106  ;
0800                   107  ;
0800                   108  ; This program demonstrates a practical
0800                   109  ; application of the SPEED/ASM routines
0800                   110  ; running under Apple DOS.
0800                   111  ;
0800                   112  ;
0800                   113  ;  This section of code provides the
0800                   114  ; input module.  The following information
0800                   115  ; is read in and written to the disk.
0800                   116  ;
0800                   117  ;
```

Listing continued.

Listing continued.

```
0800                    ;  First name: 13 characters.
0800                    ;  Last name:  13 characters.
0800                    ;  C/O:        25 characters.
0800                    ;  Street:     31 characters.
0800                    ;  City:       15 characters.
0800                    ;  State:       4 characters.
0800                    ;  Zipcode:    11 characters.
0800                    ;  Phone:      16 characters.
0800                    ;
0800                    ;
0800                    ;
0800                    **********************************
0800                    *
0800                    *
0800                    * SPEED/ASM initialization and
0800                    * title page.
0800                    *
0800                    **********************************
0800                    ;
0800                    ;
0800 20 12 78           BEGIN    JSR INIT            ;Always before using
                                                     SPEED/ASM
0803                    ;
0803                    ;
0803                    ;
0803                    **********************************
0803                    *
0803                    * Mini-Mailer Main Loop
0803                    *
0803                    **********************************
0803                    ;
0803                    ;
0803                    ;
0803                    ;
0803 20 36 78           JSR PRINT
0806 8D 8D C9           BYT CR,CR,"Insert disk with the MAIL.LIST",CR
0809 EE F3 E5
080C F2 F4 A0
080F E4 E9 F3
0812 E8 A0 F7
0815 E9 F4 E8
0818 A0 F4 E8
081B E5 A0 CD
081E C1 C9 CC
0821 AE CC C9
0824 D3 D4 8D
0827 A8 E9 E6           BYT "(if you have one) into the active drive"
                        ,CR
082A A0 F9 EF
082D F5 A0 E8
0830 E1 F6 E5
0833 A0 EF EE
0836 E5 A9 A0
0839 E9 EE F4
083C EF A0 F4
083F E8 E5 A0
0842 E1 E3 F4
0845 E9 F6 E5
0848 A0 E4 F2
084B E9 F6 E5
084E 8D
084F E1 EE E4           BYT "and press return: ",0
0852 A0 F0 F2
0855 E1                 0855: E5 F3 F3 A0 F2 E5 F4 F5 F2 EE BA A0 00
0862 20 0F 78           JSR READLN
0865                    ;
0865 20 36 78           JSR PRINT
0868 8D 84 CF           BYT CR,CTLD,"OPEN MAIL.LIST,L128",CR
086B D0 C5 CE
086E A0 CD C1
0871 C9 CC AE
0874 CC C9 D3
0877 D4 AC CC
087A B1 B2 B8
087D 8D
087E 84 CE CF           BYT CTLD,"NOMON O,I,C",CR,0
0881 CD CF CE
0884 A0 CF AC
0887 C9 AC C3
088A 8D 00
088C                    ;
088C                    ;
088C                    ;
088C                    ;
088C 20 0C 78           MAINLOOP JSR HOME
088F 20 36 78           JSR PRINT
0892 8D 8D D3           BYT CR,CR,"SPEED/ASM Mini-Mailing List
                        Program",CR,CR
0895 D0 C5 C5
0898 C4 AF C1
089B D3 CD A0
089E CD E9 EE
08A1 E9 AD CD
08A4 E1 E9 EC
08A7 E9 EE E7
08AA A0 CC E9
08AD F3 F4 A0
08B0 D0 F2 EF
08B3 E7 F2 E1
08B6 ED 8D 8D
08B9 C9 EE F0           BYT "Input Module:",CR,CR
08BC F5 F4 A0
08BF CD EF E4
08C2 F5 EC E5
08C5 BA 8D 8D
08C8 A0 A0 A0           BYT "    A) Add records",CR
08CB A0 C1 A9
08CE A0 C1 E4
08D1 E4 A0 F2
08D4 E5 E3 EF
08D7 F2 E4 F3
08DA 8D
08DB A0 A0 A0           BYT "    D) Delete a record",CR
08DE A0 C4 A9
08E1 A0 C4 E5
08E4 EC E5 F4
08E7 E5 A0 E1
08EA A0 F2 E5
08ED E3 EF F2
08F0 E4 8D
08F2 A0 A0 A0           BYT "    O) Open a new file",CR
08F5 A0 CF A9
08F8 F3                 08F8:A0 CD F0 E5 EE A0 E1 A0 EE E5 F7 A0 E6 E9 EC E5 8D
0909 A0 A0 A0           BYT "    P) Print file",CR
090C A0 D0 A9
090F A0 D0 F2
0912 E9 EE F4
0915 A0 E6 E9
0918 EC E5 8D
091B 8D                 BYT CR
091C A0 A0 A0           BYT "    Q) Quit",CR,CR
091F A0 D1 A9
0922 A0 D1 F5
0925 E9 F4 8D
0928 8D
0929 C3 E8 EF           BYT "Choice: ",0
092C E9 E3 E5
092F BA A0 00
0932                    ;
0932                    ;
0932                    ;
0932                    ;
0932                    ; READLN(INPUTSTR);
0932                    ;
0932 20 0F 78           JSR READLN
0935 20 3F 78           JSR RDSTR
0938 F6 11              ADR INPUTSTR
093A                    ;
093A                    ;
093A                    ; IF (INPUTSTR = "A") OR (INPUTSTR = "a") THEN
093A                    ;      GOTO ADDRECS;
093A                    ;
093A 20 27 78           JSR IFS0
093D F6 11 BD           ADR INPUTSTR,EQ
0940 00
0941 C1 00              BYT "A",0
0943 D0 0E              BTR ADDRECS
0945 20 27 78           JSR IFS0
0948 F6 11 BD           ADR INPUTSTR,EQ
094B 00
094C E1 00              BYT "a",0
094E D0 03              BTR ADDRECS
0950 4C 8B 0C           JMP TSTFORD
0953                    ;
0953                    ;
0953                    ;
0953                    ;
0953                    TTL "Mini-Mailing List: Add Module"
0953                    ;
0953                    ;
0953                    ;
0953                    ;
0953                    **********************************
0953                    ;
0953                    ;
0953                    ; Handle the "A" option here.
0953                    ;
0953 20 0C 78           ADDRECS  JSR HOME
0956 20 36 78           JSR PRINT
0959 8D 8D              BYT CR,CR
095B C1 E4 E4           BYT "Add records module:",CR,CR,0
095E A0 F2 E5
0961 E3 EF F2
0964 E4 F3 A0
0967 ED EF E4
096A F5 EC E5
096D BA 8D 8D
0970 00
0971                    ;
0971                    ;
0971                    ;
0971                    ;
0971 20 36 78           NOERROR  JSR PRINT
0974 8D 84 D2           BYT CR,CTLD,"READ MAIL.LIST,R0,B0",CR,0
0977 C5 C1 C4
097A A0 CD C1
097D C9 CC AE
0980 CC C9 D3
0983 D4 AC D2
0986 B0 AC C2
0989 B0 8D 00
098C                    ;
098C                    ; Read in the number of records in this
098C                    ; file into NUMRECS.
098C                    ;
098C 20 0F 78           JSR READLN
098F 20 42 78           JSR RDINT
0992 F2 11              ADR NUMRECS
```

Listing continued.

Listing continued.

```
0994 20 36 78   229           JSR PRINT           ;Turn off read cmd.
0997 8D 84 8D   230           BYT CR,CTLD,CR,0
099A 00
099B            231  ;
099B            232  ;
099B            233  ;
099B            234  ;
099B            235  ;
099B            236  ; Get the information to write to
099B            237  ; the file.
099B            238  ;
099B 20 36 78   239  GETFNAME JSR PRINT
099E 8D C5 EE   240           BYT CR,"Enter First name: ",0
09A1 F4 E5 F2
09A4 A0 C6 E9
09A7 F2 F3 F4
09AA A0 EE E1
09AD ED E5 BA
09B0 A0 00
09B2 20 0F 78   241           JSR READLN
09B5 20 3F 78   242           JSR RDSTR
09B8 78 12      243           ADR FIRSTNAM
09BA 50 1D      244           BVC GETLNAME        ;If ok.
09BC            245  ;
09BC 20 36 78   246           JSR PRINT
09BF 8D 87 D2   247           BYT CR,BELL,"Re-enter first name",CR,0
09C2 E5 AD E5
09C5 EE F4 E5
09C8 F2 A0 E6
09CB E9 F2 F3
09CE F4 A0 EE
09D1 E1 ED E5
09D4 8D 00
09D6 4C 9B 09   248           JMP GETFNAME
09D9            249  ;
09D9            250  ;
09D9            251  ;
09D9 20 36 78   252  GETLNAME JSR PRINT
09DC 8D C5 EE   253           BYT CR,"Enter Last name: ",0
09DF F4 E5 F2
09E2 A0 CC E1
09E5 F3 F4 A0
09E8 EE E1 ED
09EB E5 BA A0
09EE 00
09EF 20 0F 78   254           JSR READLN
09F2 20 3F 78   255           JSR RDSTR
09F5 85 12      256           ADR LASTNAME
09F7 50 1C      257           BVC GETAUX
09F9            258  ;
09F9 20 36 78   259           JSR PRINT
09FC 8D 87 D2   260           BYT CR,BELL,"Re-enter last name",CR,0
09FF E5 AD E5
0A02 EE F4 E5
0A05 F2 A0 BC
0A08 E1 F3 F4
0A0B A0 EE E1
0A0E ED E5 8D
0A11 00
0A12 4C D9 09   261           JMP GETLNAME
0A15            262  ;
0A15            263  ;
0A15            264  ;
0A15 20 36 78   265  GETAUX   JSR PRINT
0A18 C5 EE F4   266           BYT "Enter auxillary info: ",0
0A1B E5 F2 A0
0A1E E1 F5 F8
0A21 E9 EC EC
0A24 E1 F2 F9
0A27 A0 E9 EE
0A2A E6 EF BA
0A2D A0 00
0A2F 20 0F 78   267           JSR READLN
0A32 20 3F 78   268           JSR RDSTR
0A35 92 12      269           ADR AUXINFO
0A37 50 28      270           BVC GETSTRT
0A39            271  ;
0A39 20 36 78   272           JSR PRINT
0A3C 8D 87 D2   273           BYT CR,BELL,"Re-enter Auxillary information"
                              ,CR,0
0A3F E5 AD E5
0A42 EE F4 E5
0A45 F2 A0 C1
0A48 F5 F8 E9
0A4B EC EC E1
0A4E F2 F9 A0
0A51 E9 EE E6
0A54 EF F2 ED
0A57 E1 F4 E9
0A5A EF EE 8D
0A5D 00
0A5E 4C 15 0A   274           JMP GETAUX
0A61            275  ;
0A61            276  ;
0A61            277  ;
0A61 20 36 78   278  GETSTRT  JSR PRINT
0A64 8D C5 EE   279           BYT CR,"Enter street address: ",0
0A67 F4 E5 F2
0A6A A0 F3 F4
0A6D F2 E5 E5
0A70 F4 A0 E1
0A73 E4 E4 F2
0A76 E5 F3 F3
0A79 BA A0 00
```

```
0A18:C5 EE F4 E5 F2 A0 E1 F5 F8 E9 EC E9 E1
     F2 F9 A9 E9 EE E6 EF BA A0 00

0A3C:8D 87 D2 E5 AD E5 EE F4 E5 F2 A0 C1 F5 F8 E9 EC
E9 E1 F2 F9 A0 E9 EE E6 EF F2 ED E1 F4 E9 EF EE 8D 00
```

```
0A7C 20 0F 78   280           JSR READLN
0A7F 20 3F 78   281           JSR RDSTR
0A82 AB 12      282           ADR STRTADRS
0A84 50 21      283           BVC GETCITY
0A86            284  ;
0A86 20 36 78   285           JSR PRINT
0A89 8D 87 D2   286           BYT CR,BELL,"Re-enter Street Address",CR,0
0A8C E5 AD E5
0A8F EE F4 E5
0A92 F2 A0 D3
0A95 F4 F2 E5
0A98 E5 F4 A0
0A9B C1 E4 E4
0A9E F2 E5 F3
0AA1 F3 8D 00
0AA4 4C 61 0A   287           JMP GETSTRT
0AA7            288  ;
0AA7            289  ;
0AA7            290  ;
0AA7 20 36 78   291  GETCITY  JSR PRINT
0AAA 8D C5 EE   292           BYT CR,"Enter city: ",0
0AAD F4 E5 F2
0AB0 A0 E3 E9
0AB3 F4 F9 BA
0AB6 A0 00
0AB8 20 0F 78   293           JSR READLN
0ABB 20 3F 78   294           JSR RDSTR
0ABE CA 12      295           ADR CITY
0AC0 50 17      296           BVC GETSTATE
0AC2            297  ;
0AC2 20 36 78   298           JSR PRINT
0AC5 8D 87 D2   299           BYT CR,BELL,"Re-enter city",CR,0
0AC8 E5 AD E5
0ACB EE F4 E5
0ACE E1         0ACE:F2 A0 E3 E9 F4 F9 8D 00
0AD6 4C A7 0A   300           JMP GETCITY
0AD9            301  ;
0AD9            302  ;
0AD9            303  ;
0AD9 20 36 78   304  GETSTATE JSR PRINT
0ADC 8D C5 EE   305           BYT CR,"Enter state: ",0
0ADF F4 E5 F2
0AE2 A0 F3 F4
0AE5 E1 F4 E5
0AE8 BA A0 00
0AEB 20 0F 78   306           JSR READLN
0AEE 20 3F 78   307           JSR RDSTR
0AF1 D9 12      308           ADR STATE
0AF3 50 18      309           BVC GETZIP
0AF5            310  ;
0AF5 20 36 78   311           JSR PRINT
0AF8 8D 87 D2   312           BYT CR,BELL,"Re-enter State",CR,0
0AFB E5 AD E5
0AFE EE F4 E5
0B01 F2 A0 D3
0B04 F4 E1 F4
0B07 E5 8D 00
0B0A 4C D9 0A   313           JMP GETSTATE
0B0D            314  ;
0B0D            315  ;
0B0D            316  ;
0B0D 20 36 78   317  GETZIP   JSR PRINT
0B10 8D C5 EE   318           BYT CR,"Enter Zipcode: ",0
0B13 F4 E5 F2
0B16 A0 DA E9
0B19 F0 E3 EF
0B1C E4 E5 BA
0B1F A0 00
0B21 20 0F 78   319           JSR READLN
0B24 20 3F 78   320           JSR RDSTR
0B27 DD 12      321           ADR ZIPCODE
0B29 50 1A      322           BVC GETPHONE
0B2B            323  ;
0B2B 20 36 78   324           JSR PRINT
0B2E 8D 87 D2   325           BYT CR,BELL,"Re-enter zipcode",CR,0
0B31 E5 AD E5
0B34 EE F4 E5
0B37 F2 A0 FA
0B3A E9 F0 E3
0B3D EF E4 E5
0B40 8D 00
0B42 4C 0D 0B   326           JMP GETZIP
0B45            327  ;
0B45            328  ;
0B45            329  ;
0B45 20 36 78   330  GETPHONE JSR PRINT
0B48 8D C5 EE   331           BYT CR,"Enter Phone #: ",0
0B4B F4 E5 F2
0B4E A0 D0 E8
0B51 EF EE E5
0B54 A0 A3 BA
0B57 A0 00
0B59 20 0F 78   332           JSR READLN
0B5C 20 3F 78   333           JSR RDSTR
0B5F E8 12      334           ADR PHONENUM
0B61 50 1F      335           BVC WRTINFO
0B63            336  ;
0B63 20 36 78   337           JSR PRINT
0B66 8D 87 D2   338           BYT CR,BELL,"Re-enter Phone number",CR,0
0B69 E5 AD E5
0B6C EE F4 E5
0B6F F2 A0 D0
0B72 E8 EF EE
0B75 E5 A0 EE
```

Listing continued.

Listing continued.

```
0B78 F5 ED E2
0B7B E5 F2 8D
0B7E 00
0B7F 4C 45 0B   339          JMP GETPHONE
0B82            340  ;
0B82            341  ;
0B82            342  ;
0B82 18         343  WRTINFO  CLC
0B83 AD F2 11   344          LDA NUMRECS
0B86 69 01      345          ADC #1
0B88 8D F4 11   346          STA RECNUM
0B8B AD F3 11   347          LDA NUMRECS+1
0B8E 69 00      348          ADC /1
0B90 8D F5 11   349          STA RECNUM+1
0B93            350  ;
0B93            351  ;
0B93            352  ; PRINT CHR$(4);"WRITE MAIL.LIST,R";RECNUM;",B0"
0B93            353  ;
0B93 20 36 78   354          JSR PRINT
0B96 8D 84 D7   355          BYT CR,CTLD,"WRITE MAIL.LIST,R",0
0B99 D2 C9 D4
0B9C C5 A0 CD
0B9F C1 C9 CC
0BA2 AE CC C9
0BA5 D3 D4 AC
0BA8 D2 00
0BAA 20 3C 78   356          JSR PRTINT
0BAD F4 11      357          ADR RECNUM
0BAF 20 36 78   358          JSR PRINT
0BB2 AC C2 B0   359          BYT ",B0",CR,0
0BB5 8D 00
0BB7            360  ;
0BB7            361  ;
0BB7 20 39 78   362          JSR PRTSTR
0BBA 78 12      363          ADR FIRSTNAM
0BBC 20 E9 11   364          JSR WRITELN
0BBF            365  ;
0BBF 20 39 78   366          JSR PRTSTR
0BC2 85 12      367          ADR LASTNAME
0BC4 20 E9 11   368          JSR WRITELN
0BC7            369  ;
0BC7 20 39 78   370          JSR PRTSTR
0BCA 92 12      371          ADR AUXINFO
0BCC 20 E9 11   372          JSR WRITELN
0BCF            373  ;
0BCF 20 39 78   374          JSR PRTSTR
0BD2 AB 12      375          ADR STRTADRS
0BD4 20 E9 11   376          JSR WRITELN
0BD7            377  ;
0BD7 20 39 78   378          JSR PRTSTR
0BDA CA 12      379          ADR CITY
0BDC 20 E9 11   380          JSR WRITELN
0BDF            381  ;
0BDF 20 39 78   382          JSR PRTSTR
0BE2 D9 12      383          ADR STATE
0BE4 20 E9 11   384          JSR WRITELN
0BE7            385  ;
0BE7 20 39 78   386          JSR PRTSTR
0BEA DD 12      387          ADR ZIPCODE
0BEC 20 E9 11   388          JSR WRITELN
0BEF            389  ;
0BEF 20 39 78   390          JSR PRTSTR
0BF2 E8 12      391          ADR PHONENUM
0BF4 20 E9 11   392          JSR WRITELN
0BF7            393  ;
0BF7            394  ;
0BF7            395  ; Turn off output to the file.
0BF7            396  ;
0BF7 20 36 78   397          JSR PRINT
0BFA 84 8D 00   398          BYT CTLD,CR,0
0BFD            399  ;
0BFD            400  ; NUMRECS = RECNUM (Which contains NUMRECS+1)
0BFD            401  ;
0BFD 20 2A 78   402          JSR MOVE
0C00 F4 11 F2   403          ADR RECNUM,NUMRECS
0C03 11
0C04            404  ;
0C04            405  ;
0C04 20 36 78   406  GETYORNO JSR PRINT
0C07 8D 8D C1   407          BYT CR,CR,"Add another record? (Y/N): ",0
0C0A E4 E4 A0
0C0D E1 EE EF
0C10 F4 E8 E5
0C13 F2 A0 F2
0C16 E5 E3 EF
0C19 F2 E4 BF
0C1C A0 A8 D9
0C1F AF CE A9
0C22 BA A0 00
0C25 20 0F 78   408          JSR READLN
0C28 20 3F 78   409          JSR RDSTR
0C2B F6 11      410          ADR INPUTSTR
0C2D            411  ;
0C2D 20 27 78   412          JSR IFS0
0C30 F6 11 ED   413          ADR INPUTSTR,EQ
0C33 00
0C34 CE 00      414          BYT "N",0
0C36 D0 24      415          BNR EXITADD
0C38            416  ;
0C38 20 27 78   417          JSR IFS0
0C3B F6 11 ED   418          ADR INPUTSTR,EQ
0C3E 00
0C3F F9 00      419          BYT "y",0
0C41 D0 19      420          BNR EXITADD
```

Listing continued.

Listing continued.

```
0C43                    421  ;
0C43 20 27 78           422          JSR IFS0
0C46 F6 11 BD           423          ADR INPUTSTR,EQ
0C49 00
0C4A D9 00              424          BYT "Y",0
0C4C D0 0B              425          BNE DOAGAIN
0C4E                    426  ;
0C4E 20 27 78           427          JSR IFS0
0C51 F6 11 BD           428          ADR INPUTSTR,EQ
0C54 00
0C55 F9 00              429          BYT "y",0
0C57 F0 AB              430          BEQ GETYORNO
0C59                    431  ;
0C59 4C 9B 09           432  DOAGAIN JMP GETFNAME
0C5C                    433  ;
0C5C                    434  ;
0C5C                    435  ; Before exiting, update the record
0C5C                    436  ; count
0C5C                    437  ;
0C5C 20 36 78           438  EXITADD JSR PRINT
0C5F 8D 84 D7           439          BYT CR,CTLD,"WRITE MAIL.LIST,R0,B0",CR
0C62 D2 C9 D4
0C65 C5 A0 CC
0C68 C1 C9 CC
0C6B AE CC C9
0C6E D3 D4 AC
0C71 D2 B0 AC
0C74 C2 B0 8D
0C77 00                 440          BYT 0
0C78                    441  ;
0C78                    442  ;
0C78 20 3C 78           443          JSR PRTINT
0C7B F2 11              444          ADR NUMRECS
0C7D 20 36 78           445          JSR PRINT
0C80 A0 A0 A0           446          BYT "    ",CR,CTLD,CR,0
0C83 A0 8D 84
0C86 8D 00
0C88 4C 8C 08           447          JMP MAINLOOP
0C8B                    448  ;
0C8B                    449  ;
0C8B                    450  ;
0C8B                    451  ;
0C8B                    452  ;
0C8B                    453  ;
0C8B                    454          TTL "Mini-Mailer: Delete Module"
0C8B                    455  ;
0C8B                    456  **********************************
0C8B                    457  ;
0C8B                    458  ;
0C8B 20 27 78           459  TSTFORD JSR IFS0
0C8E F6 11 BD           460          ADR INPUTSTR,EQ
0C91 00
0C92 C4 00              461          BYT "D",0
0C94 D0 0E              462          BNE ISD
0C96                    463  ;
0C96 20 27 78           464          JSR IFS0
0C99 F6 11 BD           465          ADR INPUTSTR,EQ
0C9C 00
0C9D E4 00              466          BYT "d",0
0C9F D0 03              467          BNE ISD
0CA1 4C 3D 0E           468          JMP TSTFORO
0CA4                    469  ;
0CA4 20 36 78           470  ISD     JSR PRINT
0CA7 8D 8D C4           471          BYT CR,CR,"Delete which record? ",0
0CAA E5 EC E5
0CAD F4 E5 A0
0CB0 F7 E8 E9
0CB3 E3 E8 A0
0CB6 F2 E5 E3
0CB9 EF F2 E4
0CBC BF A0 00
0CBF 20 0F 78           472          JSR READLN
0CC2 20 42 78           473          JSR RDINT
0CC5 F4 11              474          ADR RECNUM
0CC7 50 29              475          BVC GOODNUM
0CC9                    476  ;
0CC9 20 36 78           477          JSR PRINT
0CCC 8D 87 C9           478          BYT CR,BELL,"Illegal record number,
                                     re-enter",CR,0
0CCF EC EC E5
0CD2 E7 E1 EC
0CD5 A0 F2 E5
0CD8 E3 EF F2
0CDB E4 A0 EE
0CDE F5 ED E2
0CE1 E5 F2 AC
0CE4 A0 F2 E5
0CE7 AD E5 EE
0CEA F4 E5 F2
0CED 8D 00
0CEF 4C 8B 0C           479          JMP TSTFORD
0CF2                    480  ;
0CF2                    481  ;
0CF2                    482  ; See if this is a valid record number.
0CF2                    483  ;
0CF2 20 36 78           484  GOODNUM JSR PRINT
0CF5 8D 84 D2           485          BYT CR,CTLD,"READ MAIL.LIST,R0,B0",CR,0
0CF8 C5 C1 C4
0CFB A0 CD C1
0CFE C9 CC AE
0D01 CC C9 D3
0D04 D4 AC D2
0D07 B0 AC C2
0D0A B0 8D 00
0D0D 20 0F 78           486          JSR READLN
```

Listing continued.

Listing continued.

```
0D10 20 42 78   487           JSR RDINT
0D13 F2 11      488           ADR NUMRECS
0D15            489  ;
0D15 20 1E 78   490           JSR IFI
0D18 F4 11 BC   491           ADR RECNUM,LE,NUMRECS
0D1B BD F2 11
0D1E D0 3F      492           BNE RECOK
0D20            493  ;
0D20 20 36 78   494           JSR PRINT
0D23 8D D4 E8   495           BYT CR,"That record doesn't exist",CR
0D26 E1 F4 A0
0D29 F2 E5 E3
0D2C EF F2 E4
0D2F A0 E4 EF
0D32 E5 F3 EE
0D35 A7 F4 A0
0D38 E5 F8 E9
0D3B F3 F4 8D
0D3E 8D D0 F2   496           BYT CR,"Press return to continue:",0
0D41 E5 F3 F3
0D44 A0 F2 E5
0D47 F4 F5 F2
0D4A EE A0 F4
0D4D EF A0 E3
0D50 EF EE F4
0D53 E9 EE F5
0D56 E5 BA 00
0D59 20 0F 78   497           JSR READLN
0D5C 4C 8C 08   498           JMP MAINLOOP
0D5F            499  ;
0D5F 20 36 78   500  RECOK    JSR PRINT
0D62 8D 84 D7   501           BYT CR,CTLD,"WRITE MAIL.LIST,R",0
0D65 D2 C9 D4
0D68 C5 A0 CD
0D6B C1 C9 CC
0D6E AE CC C9
0D71 D3 D4 AC
0D74 D2 00
0D76 20 3C 78   502           JSR PRTINT
0D79 F4 11      503           ADR RECNUM
0D7B 20 36 78   504           JSR PRINT
0D7E AC C2 B0   505           BYT ",B0",CR
0D81 8D
0D82 80 8D      506           BYT $80,CR
0D84 84 8D 00   507           BYT CTLD,CR,0
0D87            508  ;
0D87 20 36 78   509           JSR PRINT
0D8A 8D 8D C4   510           BYT CR,CR,"Deleted",CR
0D8D E5 EC E5
0D90 F4 E5 E4
0D93 8D
0D94 D0 F2 E5   511           BYT "Press return to continue:",0
0D97 F3 F3 A0
0D9A F2 E5 F4
0D9D F5 F2 EE
0DA0 A0 F4 EF
0DA3 A0 E3 EF
0DA6 EE F4 E9
0DA9 EE F5 E5
0DAC BA 00
0DAE 20 0F 78   512           JSR READLN
0DB1 4C 8C 08   513           JMP MAINLOOP
0DB4            514  ;
0DB4            515  ;
0DB4            516  ;
0DB4            517           TTL "Mini-Mailer: Status Module"
0DB4            518  ;
0DB4            519  ;
0DB4            520  ***********************************
0DB4            521  ;
0DB4            522  ;
0DB4 20 27 78   523  TSTFORS   JSR IFS0
0DB7 F6 11 BD   524           ADR INPUTSTR,EQ
0DBA 00
0DBB D3 00      525           BYT "S",0
0DBD D0 0E      526           BNE ISS
0DBF            527  ;
0DBF 20 27 78   528           JSR IFS0
0DC2 F6 11 BD   529           ADR INPUTSTR,EQ
0DC5 00
0DC6 F3 00      530           BYT "s",0
0DC8 D0 03      531           BNE ISS
0DCA 4C 3D 0E   532           JMP TSTFORO
0DCD            533  ;
0DCD            534  ;
0DCD 20 36 78   535  ISS      JSR PRINT
0DD0 8D 84 D2   536           BYT CR,CTLD,"READ MAIL.LIST,R0,B0",CR,0
0DD3 C5 C1 C4
0DD6 A0 CD C1
0DD9 C9 CC AE
0DDC CC C9 D3
0DDF D4 AC D2
0DE2 B0 AC C2
0DE5 B0 8D 00
0DE8 20 0F 78   537           JSR READLN
0DEB 20 42 78   538           JSR RDINT
0DEE F2 11      539           ADR NUMRECS
0DF0            540  ;
0DF0 20 36 78   541           JSR PRINT
0DF3 8D D4 E8   542           BYT CR,"There are ",0
0DF6 E5 F2 E5
0DF9 A0 E1 F2
0DFC E5 A0 00
0DFF 20 3C 78   543           JSR PRTINT
0E02 F2 11      544           ADR NUMRECS
0E04            545  ;
0E04 20 36 78   546           JSR PRINT
0E07 A0 F2 E5   547           BYT " records in the file",CR,CR
0E0A E3 EF F2
0E0D E4 F3 A0
0E10 E9 EE A0
0E13 F4 E8 E5
0E16 A0 E6 E9
0E19 EC E5 8D
0E1C 8D
0E1D D0 F2 E5   548           BYT "Press return to continue:",0
0E20 F3 F3 A0
0E23 F2 E5 F4
0E26 F5 F2 EE
0E29 A0 F4 EF
0E2C A0 E3 EF
0E2F EE F4 E9
0E32 EE F5 E5
0E35 BA 00
0E37 20 0F 78   549           JSR READLN
0E3A 4C 8C 08   550           JMP MAINLOOP
0E3D            554           TTL "Mini-Mailer: New File Module"
0E3D            555  ;
0E3D            556  ;    There should be ; for lines 551 ~ 553
0E3D            557  ***********************************
0E3D            558  ;
0E3D            559  ;
0E3D 20 27 78   560  TSTFORO  JSR IFS0
0E40 F6 11 BD   561           ADR INPUTSTR,EQ
0E43 00
0E44 CF 00      562           BYT "O",0
0E46 D0 0E      563           BNE ISO
0E48            564  ;
0E48 20 27 78   565           JSR IFS0
0E4B F6 11 BD   566           ADR INPUTSTR,EQ
0E4E 00
0E4F EF 00      567           BYT "o",0
0E51 D0 03      568           BNE ISO
0E53 4C D5 0F   569           JMP TSTFORP
0E56            570  ;
0E56            571  ;
0E56 20 36 78   572  ISO      JSR PRINT
0E59 8D C9 F3   573           BYT CR,"Is the previous disk still",CR
0E5C A0 F4 E8
0E5F E5 A0 F0
0E62 F2 E5 F6
0E65 E9 EF F5
0E68 F3 A0 E4
0E6B E9 F3 EB
0E6E A0 F3 F4
0E71 E9 EC EC
0E74 8D
0E75 E9 EE A0   574           BYT "in the drive? (Y/N): ",0
0E78 F4 E8 E5
0E7B A0 E4 F2
0E7E E9 F6 E5
0E81 BF A0 A8
0E84 D9 AF CE
0E87 A9 BA A0
0E8A 00
0E8B 20 0F 78   575           JSR READLN
0E8E 20 3F 78   576           JSR RDSTR
0E91 F6 11      577           ADR INPUTSTR
0E93            578  ;
0E93 20 27 78   579           JSR IFS0
0E96 F6 11 BD   580           ADR INPUTSTR,EQ
0E99 00
0E9A D9 00      581           BYT "Y",0
0E9C D0 57      582           BNE ISYES
0E9E 20 27 78   583           JSR IFS0
0EA1 F6 11 BD   584           ADR INPUTSTR,EQ
0EA4 00
0EA5 F9 00      585           BYT "y",0
0EA7 D0 4C      586           BNE ISYES
0EA9            587  ;
0EA9 20 36 78   588           JSR PRINT
0EAC 8D 87 D0   589           BYT CR,BELL,"Put the previous disk back
                              into",CR
0EAF F5 F4 A0
0EB2 F4 E8 E5       8D 87 D0 F5 F4 A0 F4 E8 E5 A0 F0
0EB5 A0 F0 F2       F2 E5 F6 E9 EF F5 F3 A0 E4 E9 F3
0EB8 E5 F6 E9       EB A0 E2 E1 E3 EB A0 E9 EE F4 EF
0EBB EF F5 F3                     8D
0EBE A0 E4 E9
0EC1 EB A0 E9       0EC1:F3 EB A0 E2 E1 E3 EB A0 E9 EE F4 EF 8D
0EC4 EE F4 EF
0EC7 8D
0ECE F4 E8 E5   590           BYT "the active drive before continuing",
                              CR,0
0ED1 A0 E1 E3
0ED4 F4 E9 F6
0ED7 E5 A0 E4
0EDA F2 E9 F6
0EDD E5 A0 E2
0EE0 E5 E6 EF
0EE3 F2 E5 A0
0EE6 E3 EF EE
0EE9 F4 E9 EE
```

Listing continued.

Listing continued.

```
0EEC F5 E9 EE
0EEF E7 8D 00
0EF2 4C 56 0E    591          JMP ISO
0EF5             592  ;
0EF5             593  ;
0EF5 20 36 78    594  ISYES   JSR PRINT
0EF8 8D 84 C3    595          BYT CR,CTLD,"CLOSE",CR,0
0EFB CC CF D3
0EFE C5 8D 00
0F01             596  ;
0F01 20 36 78    597          JSR PRINT
0F04 8D D0 BC    598          BYT CR,"Place the diskette you wish to
                              create",CR
0F07 E1 E3 E5
0F0A A0 F4 E8
0F0D E5 A0 E4
0F10 E9 F3 EB
0F13 E5 F4 F4
0F16 E5 A0 F9
0F19 EF F5 A0
0F1C F7 E9 F3
0F1F E8 A0 F4
0F22 EF A0 E3
0F25 F2 E5 E1
0F28 F4 E5 8D
0F2B E1 A0 A2    599          BYT "a ""MAIL.LIST"" file on into the active"
                              ,CR
0F2E CD C1 C9
0F31 CC AE CC
0F34 C9 D3 D4
0F37 A2 A0 B6
0F3A E9 BC E5
0F3D A0 EF EE
0F40 A0 E9 EE
0F43 F4 EF A0
0F46 F4 E8 E5
0F49 A0 E1 E3
0F4C F4 E9 F6
0F4F E5 8D
0F51 E4 E9 F3    600          BYT "disk drive",CR
0F54 EB A0 E4
0F57 F2 E9 F6
0F5A E5 8D
0F5C E1 EE E4    601          BYT "and press return: ",0
0F5F A0 F0 F2
                   E1 EE E4 A0 F0 F2 E5 F3 F3 A0
0F62 E5 F3 F3
                     F2 E5 F4 F5 F2 EE BA A0 00
0F65 A0 F2 E5
0F68 F4          0F68:F4 F5 F2 EE BA A0 00
0F6F             602  ;
0F6F 20 0F 78    603          JSR READLN
0F72 20 36 78    604          JSR PRINT
0F75 8D 84 CF    605          BYT CR,CTLD,"OPEN MAIL.LIST,L128",CR
0F78 D0 C5 CE
0F7B A0 CD C1
0F7E C9 CC AE
0F81 CC C9 D3
0F84 D4 AC CC
0F87 B1 B2 B8
0F8A 8D
0F8B 84 D7 D2    606          BYT CTLD,"WRITE MAIL.LIST",CR
0F8E C9 D4 C5
0F91 A0 CD C1
0F94 C9 CC AE
0F97 CC C9 D3
0F9A D4 8D
0F9C B0 A0 A0    607          BYT "0      ",CR
0F9F A0 A0 A0
0FA2 A0 8D
0FA4 84 8D 00    608          BYT CTLD,CR,0
0FA7             609  ;
0FA7             610  ;
0FA7 20 36 78    611          JSR PRINT
0FAA 8D C1 EC    612          BYT CR,"All done, press return to continue:"
                              ,0
0FAD EC A0 E4
0FB0 EF EE E5
0FB3 AC A0 F0
0FB6 F2 E5 F3
0FB9 F3 A0 F2
0FBC E5 F4 F5
0FBF F2 EE A0
0FC2 F4 EF A0
0FC5 E3 EF EE
0FC8 F4 E9 EE
0FCB F5 E5 BA
0FCE 00
0FCF 20 0F 78    613          JSR READLN
0FD2 4C 8C 08    614          JMP MAINLOOP
0FD5             615  ;
0FD5             616  ;
0FD5             617  ;
0FD5             618          TTL "Mini-Mailer: Print Module"
0FD5             619  ;
0FD5             620  ;
0FD5             621  **********************************
0FD5             622  ;
0FD5             623  ;
0FD5 20 27 78    624  TSTFORP  JSR IFS0
0FD8 F6 11 BD    625          ADR INPUTSTR,EQ
0FDB 00
0FDC D0 00       626          BYT "P",0
0FDE D0 0E       627          BTR ISP
0FE0             628  ;
0FE0 20 27 78    629          JSR IFS0
0FE3 F6 11 BD    630          ADR INPUTSTR,EQ
0FE6 00
0FE7 F0 00       631          BYT "p",0
0FE9 D0 03       632          BTR ISP
0FEB 4C 53 11    633          JMP TSTFORQ
0FEE             634  ;
0FEE             635  ;
0FEE 20 36 78    636  ISP     JSR PRINT
0FF1 8D D3 E5    637          BYT CR,"Send output to what slot?",CR
0FF4 EE E4 A0
0FF7 EF F5 F4
0FFA F0 F5 F4
0FFD A0 F4 EF
1000 A0 F7 E8
1003 E1 F4 A0
1006 F3 EC EF
1009 F4 BF 8D
100C A8 FA E5    638          BYT "(zero for screen): ",0
100F F2 EF A0
1012 B5 EF F2
1015 A0 F3 E3
1018 F2 E5 E5
101B EE A9 BA
101E A0 00
1020 20 0F 78    639          JSR READLN
1023 20 42 78    640          JSR RDINT
1026 F0 11       641          ADR I
1028             642  ;
1028             643  ;
1028 20 36 78    644          JSR PRINT
102B 8D 84 D2    645          BYT CR,CTLD,"READ MAIL.LIST,R0,B0",CR,0
102E C5 C1 C4
1031 A0 CD C1
1034 C9 CC AE
1037 CC C9 D3
103A D4 AC D2
103D B0 AC C2
1040 B0 8D 00
1043 20 0F 78    646          JSR READLN
1046 20 42 78    647          JSR RDINT
1049 F2 11       648          ADR NUMRECS
104B             649  ;
104B 20 36 78    650          JSR PRINT
104E 8D 84 D0    651          BYT CR,CTLD,"PR#",0
1051 D2 A3 00
1054 20 3C 78    652          JSR PRTINT
1057 F0 11       653          ADR I
1059 20 E9 11    654          JSR WRITELN
105C             655  ;
105C             656  ;
105C             657  ;
105C             658  ; FOR I = 1 TO NUMRECS STEP 1
105C             659  ;
105C 20 15 78    660          JSR FOR
105F F0 11 EE    661          ADR I,ONE,NUMRECS,ONE
1062 11 F2 11
1065 EE 11
1067             662  ;
1067 20 36 78    663          JSR PRINT
106A 84 D2 C5    664          BYT CTLD,"READ MAIL.LIST,B0,R",0
106D C1 C4 A0
1070 CD C1 C9
1073 CC AE CC
1076 C9 D3 D4
1079 AC C2 B0
107C AC D2 00
107F 20 3C 78    665          JSR PRTINT
1082 F0 11       666          ADR I
1084 20 E9 11    667          JSR WRITELN
1087             668  ;
1087 20 0F 78    669          JSR READLN
108A 20 3F 78    670          JSR RDSTR
108D 78 12       671          ADR FIRSTNAM
108F             672  ;
108F             673  ; Check to see if this is a deleted
108F             674  ; record or not.
108F             675  ;
108F 20 27 78    676          JSR IFS0
1092 78 12 BD    677          ADR FIRSTNAM,EQ
1095 00
1096 80 00       678          BYT $80,0
1098 F0 03       679          BFL NOTDLTD
109A 4C 24 11    680          JMP NEXTP
109D             681  ;
109D             682  ;
109D 20 0F 78    683  NOTDLTD  JSR READLN
10A0 20 3F 78    684          JSR RDSTR
10A3 85 12       685          ADR LASTNAME
10A5             686  ;
10A5 20 0F 78    687          JSR READLN
10A8 20 3F 78    688          JSR RDSTR
10AB 92 12       689          ADR AUXINFO
10AD             690  ;
10AD 20 0F 78    691          JSR READLN
10B0 20 3F 78    692          JSR RDSTR
10B3 AB 12       693          ADR STRTADRS
10B5             694  ;
10B5 20 0F 78    695          JSR READLN
10B8 20 3F 78    696          JSR RDSTR
10BB CA 12       697          ADR CITY
10BD             698  ;
10BD 20 0F 78    699          JSR READLN
10C0 20 3F 78    700          JSR RDSTR
10C3 D9 12       701          ADR STATE
```

Listing continued.

Listing continued.

```
10C5                    702  ;
10C5 20 0F 78           703          JSR READLN
10C8 20 3F 78           704          JSR RDSTR
10CB DD 12              705          ADR ZIPCODE
10CD                    706  ;
10CD 20 0F 78           707          JSR READLN
10D0 20 3F 78           708          JSR RDSTR
10D3 E8 12              709          ADR PHONENUM
10D5                    710  ;
10D5                    711  ;
10D5 20 39 78           712          JSR PRTSTR
10D8 85 12              713          ADR LASTNAME
10DA 20 36 78           714          JSR PRINT
10DD AC A0 00           715          BYT ", ",0
10E0 20 39 78           716          JSR PRTSTR
10E3 78 12              717          ADR FIRSTNAM
10E5 20 E9 11           718          JSR WRITELN
10E8                    719  ;
10E8 20 39 78           720          JSR PRTSTR
10EB 92 12              721          ADR AUXINFO
10ED 20 E9 11           722          JSR WRITELN
10F0                    723  ;
10F0 20 39 78           724          JSR PRTSTR
10F3 AB 12              725          ADR STRTADRS
10F5 20 E9 11           726          JSR WRITELN
10F8                    727  ;
10F8 20 39 78           728          JSR PRTSTR
10FB CA 12              729          ADR CITY
10FD 20 36 78           730          JSR PRINT
1100 AC A0 00           731          BYT ", ",0
1103 20 39 78           732          JSR PRTSTR
1106 D9 12              733          ADR STATE
1108 20 36 78           734          JSR PRINT
110B A0 A0 00           735          BYT "  ",0
110E 20 39 78           736          JSR PRTSTR
1111 DD 12              737          ADR ZIPCODE
1113 20 E9 11           738          JSR WRITELN
1116                    739  ;
1116 20 39 78           740          JSR PRTSTR
1119 E8 12              741          ADR PHONENUM
111B 20 E9 11           742          JSR WRITELN
111E 20 E9 11           743          JSR WRITELN
1121 20 E9 11           744          JSR WRITELN
1124                    745  ;
1124 20 1B 78           746  NEXTP   JSR NEXT
1127                    747  ;
1127                    748  ;
1127 20 36 78           749          JSR PRINT
112A 8D 84 D0           750          BYT CR,CTLD,"PR#0",CR
112D D2 A3 B0
1130 8D
```

```
1131 8D 8D              751          BYT CR,CR
1133 D0 F2 E5           752          BYT "Press return to continue:",0
1136 F3 F3 A0
1139 F2 E5 F4
113C F5 F2 EE
113F A0 F4 EF
1142 A0 E3 EF
1145 EE F4 E9
1148 EE F5 E5
114B BA 00
114D 20 0F 78           753          JSR READLN
1150 4C 8C 08           754          JMP MAINLOOP
1153                    755  ;
1153                    756  ;
1153                    757  ;
1153                    758  ;
1153                    759          TTL "Auxillary routines, etc."
1153                    760  ;
1153                    761  ;
1153 20 27 78           762  TSTFORQ JSR IFS0
1156 F6 11 BD           763          ADR INPUTSTR,EQ
1159 00
115A D1 00              764          BYT "Q",0
115C D0 16              765          BIR QUIT
115E                    766  ;
115E 20 27 78           767          JSR IFS0
1161 F6 11 BD           768          ADR INPUTSTR,EQ
1164 00
1165 F1 00              769          BYT "q",0
1167 D0 0B              770          BIR QUIT
1169                    771  ;
1169 20 36 78           772          JSR PRINT
116C 8D 87 87           773          BYT CR,BELL,BELL,BELL,0
116F 87 00
1171 4C 8C 08           774          JMP MAINLOOP
1174                    775  ;
1174                    776  ;
1174 20 36 78           777  QUIT    JSR PRINT
1177 8D CD E1           778          BYT CR,"Make sure the disk with the
                                     MAIL.LIST",CR
117A EB E5 A0
117D F3 F5 F2
1180 E5 A0 F4
1183 E8 E5 A0
1186 E4 E9 F3
1189 EB A0 F7
118C E9 F4 E8
118F A0 F4 E8
1192 E5 A0 CD
1195 C1 C9 CC
1198 AE CC C9
119B D3 D4 8D
119E B6 E9 EC           779          BYT "file is still in the active disk
                                     drive",CR
11A1 E5 A0 E9
11A4 F3 A0 F3
11A7 F4 E9 EC
11AA EC A0 E9
11AD EE A0 F4
11B0 E8 E5 A0
11B3 E1 E3 F4
11B6 E9 F6 E5
11B9 A0 E4 E9
11BC F3 E8 A0
11BF E4 F2 E9
11C2 F6 E5 8D
11C5 E1 EE E4           780          BYT "and press return:",0
11C8 A0 F0 F2
11CB E5 F3 F3
11CE A0 F2 E5
11D1 F4 F5 F2
11D4 EE BA 00
11D7 20 0F 78           781          JSR READLN
11DA                    782  ;
11DA 20 36 78           783          JSR PRINT
11DD 8D 84 C3           784          BYT CR,CTLD,"CLOSE",CR,0
11E0 CC CF D3
11E3 C5 8D 00
11E6 4C 69 FF           785          JMP EXIT
11E9                    786  ;
11E9                    787  ;
11E9                    788  ;
11E9                    789  ;
11E9                    790  ;
11E9                    791  ;
11E9                    792  ***********************************
11E9                    793  ;
11E9                    794  ; Auxillary routines.
11E9                    795  ;
11E9 A9 8D              796  WRITELN  LDA #CR
11EB 4C 00 78           797          JMP PUTC
11EE                    798  ;
11EE                    799  ;
11EE                    800  ;
11EE                    801  ***********************************
11EE                    802  ;
11EE                    803  ;
11EE                    804  ; Variable declarations.
11EE                    805  ;
11EE                    806  ;
11EE                    807  ;
11EE                    808  ;
```

Listing continued.

Listing continued.

```
11EE 01 00      809  ONE       ADR 1
11F0            810  ;
11F0            811  ;
11F0            812  ;
11F0 00 00      813  I         ADR 0
11F2 00 00      814  NUMRECS   ADR 0
11F4 00 00      815  RECNUM    ADR 0
11F6            816  ;
11F6            817  ;
11F6 80 00      818  INPUTSTR  ADR 128
11F8            819            DFS 128
1278 0B 00      820  FIRSTNAM  ADR 11
127A            821            DFS 11
1285            822  ;
1285 0B 00      823  LASTNAME  ADR 11
1287            824            DFS 11
1292            825  ;
1292 17 00      826  AUXINFO   ADR 23
1294            827            DFS 23
12AB            828  ;
12AB 1D 00      829  STRTADRS  ADR 29
12AD            830            DFS 29
12CA            831  ;
12CA 0D 00      832  CITY      ADR 13
12CC            833            DFS 13
12D9            834  ;
12D9 02 00      835  STATE     ADR 2
12DB            836            DFS 2
12DD            837  ;
12DD 09 00      838  ZIPCODE   ADR 9
12DF            839            DFS 9
12E8            840  ;
12E8 0E 00      841  PHONENUM  ADR 14
12EA            842            DFS 14
12F8            843  ;
12F8            844  ;
12F8            845            END

***** END OF ASSEMBLY
```

```
]brun sort
BRUN SORT
SYMBOL TABLE SORTED ALPHABETICALLY

  ABS       7854  ADDRECS   0953  AUXINFO   1292  BEGIN     0800  BELL      0087
  CASE      7848  CASEI     784B  CITY      12CA  CONCAT    786F  CR        008D
  CTLD      0084  DIV       785D  DOAGAIN   0C59  EQ        00ED  EXIT      FF69
  EXITADD   0C5C  FALSE     0000  FIRSTNAM  1278  FOR       7815  FOR0      7818
  GE        BDBE  GETAUX    0A15  GETC      7803  GETCITY   0AA7  GETFNAME  099B
  GETLNAME  09D9  GETPHONE  0B45  GETSTATE  0AD9  GETSTRT   0A61  GETYORNO  0C04
  GETZIP    0B0D  GOODNUM   0CF2  GT        00BE  HOME      780C  I         11F0
  IFI       781E  IFI0      7821  IFS       7824  IFS0      7827  INDEX     7869
  INIT      7812  INPUTSTR  11F6  INSET     784E  ISD       0CA4  ISO       0E56
  ISP       0FEE  ISS       0DCD  ISYES     0EF5  LASTNAME  1285  LDSTR     7833
  LE        BDBC  LENGTH    786C  LOAD      782D  LT        00BC  MAINLOOP  088C
  MOD       7860  MOVE      782A  MOVS      7830  MUL       785A  NE        00A3
  NEG       7857  NEXT      781B  NEXTP     1124  NOERROR   0971  NOTDLTD   109D
  NOTINSET  7851  NUMRECS   11F2  ONE       11EE  ONXGOTO   7845  PHONENUM  12E8
  PRINT     7836  PRTINT    783C  PRTSTR    7839  PUTC      7800  QUIT      1174
  RDINT     7842  RDSTR     783F  READLN    780F  RECNUM    11F4  RECOK     0D5F
  RND       7863  SAGL      7806  SAPC      7809  STATE     12D9  STRTADRS  12AB
  SUBSTR    7866  TRUE      0001  TSTFORD   0C8B  TSTFORO   0E3D  TSTFORP   0FD5
  TSTFORQ   1153  TSTFORS   0DB4  WRITELN   11E9  WRTINFO   0B82  ZIPCODE   12DD


SYMBOL TABLE SORTED BY ADDRESS

  FALSE     0000  TRUE      0001  CTLD      0084  BELL      0087  CR        008D
  NE        00A3  LT        00BC  EQ        00ED  GT        00BE  BEGIN     0800
  MAINLOOP  088C  ADDRECS   0953  NOERROR   0971  GETFNAME  099B  GETLNAME  09D9
  GETAUX    0A15  GETSTRT   0A61  GETCITY   0AA7  GETSTATE  0AD9  GETZIP    0B0D
  GETPHONE  0B45  WRTINFO   0B82  GETYORNO  0C04  DOAGAIN   0C59  EXITADD   0C5C
  TSTFORD   0C8B  ISD       0CA4  GOODNUM   0CF2  RECOK     0D5F  TSTFORS   0DB4
  ISS       0DCD  TSTFORO   0E3D  ISO       0E56  ISYES     0EF5  TSTFORP   0FD5
  ISP       0FEE  NOTDLTD   109D  NEXTP     1124  TSTFORQ   1153  QUIT      1174
  WRITELN   11E9  ONE       11EE  I         11F0  NUMRECS   11F2  RECNUM    11F4
  INPUTSTR  11F6  FIRSTNAM  1278  LASTNAME  1285  AUXINFO   1292  STRTADRS  12AB
  CITY      12CA  STATE     12D9  ZIPCODE   12DD  PHONENUM  12E8  PUTC      7800
  GETC      7803  SAGL      7806  SAPC      7809  HOME      780C  READLN    780F
  INIT      7812  FOR       7815  FOR0      7818  NEXT      781B  IFI       781E
  IFI0      7821  IFS       7824  IFS0      7827  MOVE      782A  LOAD      782D
  MOVS      7830  LDSTR     7833  PRINT     7836  PRTSTR    7839  PRTINT    783C
  RDSTR     783F  RDINT     7842  ONXGOTO   7845  CASE      7848  CASEI     784B
  INSET     784E  NOTINSET  7851  ABS       7854  NEG       7857  MUL       785A
  DIV       785D  MOD       7860  RND       7863  SUBSTR    7866  INDEX     7869
  LENGTH    786C  CONCAT    786F  LE        BDBC  GE        BDBE  EXIT      FF69
```

The Assembly Advantage

by Randy Hyde

6502 Addressing Modes

Last month I presented the SPEED/ASM string handling routines. While these routines provide all necessary string manipulation functions, it is sometimes more convenient to perform string operations directly in 6502 code. So once again I must resort to "pure" 6502 code.

So far I've discussed only simple variable types in SPEED/ASM. When dealing with simple variables two 6502 *addressing modes* are required—*immediate* and *absolute*. The immediate mode is for loading constants into one of the 6502 registers. To do so, precede the immediate data with a pound sign (#) or a slash (/). The pound sign specifies that the low order byte of the 16-bit value is to be loaded into the register; the slash operator specifies the high order byte. This addressing mode gets its name from the fact that the data to be loaded into the accumulator *immediately* follows the instruction opcode. See Figure 1.

The absolute addressing mode is used when you want to load (or store) a register from one of the 6502's 65,535 memory locations. In this case, follow the instruction with the address of the memory location you wish to access. The absolute addressing mode is so named because the address that follows the opcode is the "absolute" address—it is not modified by anything. See Figure 2.

Unfortunately, the absolute addressing mode won't let you access varying memory locations. The address you want to access must be specified at assembly time and cannot change while the program is running. Therefore, this mode cannot be used to access elements of an array using a specified (variable) index.

In order to set up and access arrays (and string variables are classified as arrays) additional knowledge of the 6502 is necessary. To support such multi-byte data structures the 6502 microprocessor supports several *indexed* and *indirect* addressing modes. An indexed addressing mode uses either the X or Y register as an *index register* to modify the address that follows the instruction. For example, the instruction:

```
LDA ADDRESS,X
```

loads the 6502 accumulator from location ADDRESS+X where X is the current content of the X register. So the instruction sequence:

```
LDX  #2
LDA  ADDRESS,X
```

performs the same action as:

```
LDA  ADDRESS+2
```

Don't confuse a statement of the form:

```
LDA  ADDRESS+I
```

with a statement of the form:

```
LDA  ADDRESS,X
```

In the former case the address calculation is performed *at assembly time*. That is, the assembly-time value of I (its address) is added to the assembly-time value of ADDRESS and the resulting sum is used as the address for the LDA absolute instruction.

In the latter case, the address calculation is performed at run time (when the instruction is executed). Here the address that follows the LDA instruction is simply the address of ADDRESS. At run time the X register's value is added to this address to obtain the true *effective address*.

The major advantage of the indexed addressing mode is the 6502's ability to modify the X register while the program is running. For example, consider the short program:

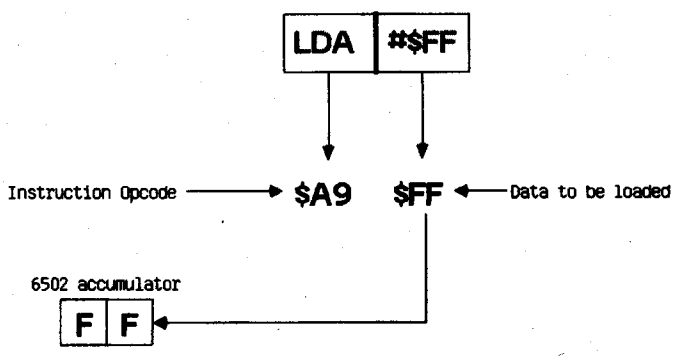


Figure 1: Immediate addressing mode format.

Randy Hyde is proprietor of Lazerware, creators of SPEED/ASM. Address correspondence to him at 925 Lorna St., Corona, CA 91720.

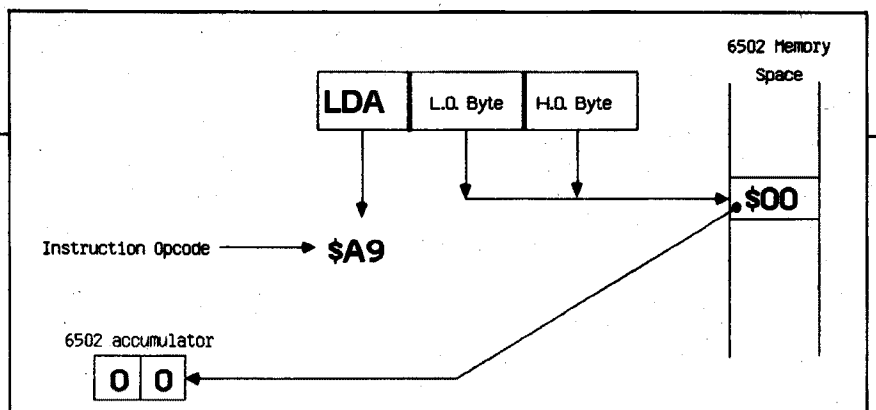


Figure 2. Absolute addressing mode format.

```
        LDX   #0
LOOP    TXA
        STA   ADDRESS,X
        INX
        BNE   LOOP
        BRK
```

This sequence initializes the 6502 X register to zero and then enters a short loop. Within the loop the value contained in the X register is copied into the accumulator with the TXA (transfer X to A) instruction. Next the value in the accumulator is stored at address ADDRESS+X. Since the X register contains zero, the current contents of the accumulator are stored at location ADDRESS+0. The next instruction, INX, increments the X register. Since the X register contains eight bits, it can only hold numbers in the range 0–255 ($00–$FF). Whenever you increment the value 255 ($FF) in the X register, it wraps back around to zero and the 6502 zero flag is set. In all other instances the zero flag is cleared. Therefore, the BNE (branch if not equal to zero) instruction can be used to continually branch back to location LOOP until the X register is incremented from $FF to $00. And that occurs only after the loop has executed 256 times. The loop above, incidentally, stores zero at location ADDRESS+0, one at location ADDRESS+1, two at location ADDRESS+2 ...., $FF at location ADDRESS+$FF. This is roughly equivalent to the Basic program:

```
10 DIM ADDRESS (255)
20 FOR X = 0 TO 255
40 A = X
50 ADDRESS (X) = A
60 NEXT X
70 STOP
```

Although I've used the 6502 X register in all the examples so far, the Y register can usually be used in a similar way. So, if you're already using the X register and you need to access some tabular data you could use the Y register instead.

Using the Indexed Modes

Now that I've described how the indexed addressing modes function, it would be a good idea to describe how to use these modes within SPEED/ASM programs. The most obvious application is to implement byte arrays. For example, an array of 16 characters could be defined as:

```
CHARARY DFS 16
```

Then you could access elements of the array using the indexed addressing modes.

It is important to note the difference between a character *string* and a character *array*. A character array is a collection of characters stored in contiguous memory locations. It is treated as a convenient collection of similar objects. A character string is a character array with two additional attributes: a maximum possible length (stored in the first byte of the string) and an actual length (denoted by storing a zero byte at the end of the current string data). A character string is typically treated as a single object.

Byte arrays are quite useful for storing tabular data such as a list of reserved words or a group of special characters. For example, consider the array:

```
VOWELS BYT "AEIOUYWaeiouyw"
```

You could use this to see if the current character in the accumulator contains a vowel. Consider the short program:

```
        LDX   #13
TSTVWL  CMP   VOWELS,X
        BEQ   ISAVWL
        DEX
        BPL   TSTVWL
; If you drop through to this point
; then the character
; in the accumulator
; isn't a vowel.
```

The new instruction here, DEX, decrements the 6502 X register by one. As long as the new value in the X register is positive (in the range 0–127 or $00–$7F) then this program branches back to the TSTVWL label. The instant you decrement $00 you get $FF, which is negative, and you fall through the loop. Notice that this loop starts at the last entry of the VOWEL array and works backwards. This short assembly language code is roughly equivalent to the Basic code:

```
10 FOR I = 13 TO 0 STEP -1
20 IF ACC=VOWELS(I) THEN GOTO
     nnnn
30 NEXT I
```

It should be noted that SPEED/ASM provides a special routine, INSET, for easily performing this type of test. I'll discuss INSET in a future article.

Emulating LENGTH and PRTSTR

Although the examples presented thus far have all dealt with character arrays, the indexed addressing modes can also be used with character strings. Keeping in mind the format for a string variable (see Figure 3), you can see that the following loop performs roughly the same operation as the SPEED/ASM LENGTH function:

```
        LDX   #0
LENGTHL LDA   STRING+1,X
        BEQ   FNDLEN
        INX
        JMP   LENGTHL ;Always
              ;taken.
;
FNDLEN
```

The only differences between this loop and the SPEED/ASM LENGTH routine are that here the length is returned in the 6502 X register (instead of the accumulator) and this loop only returns the length of STRING, not any arbitrary string. (One other,

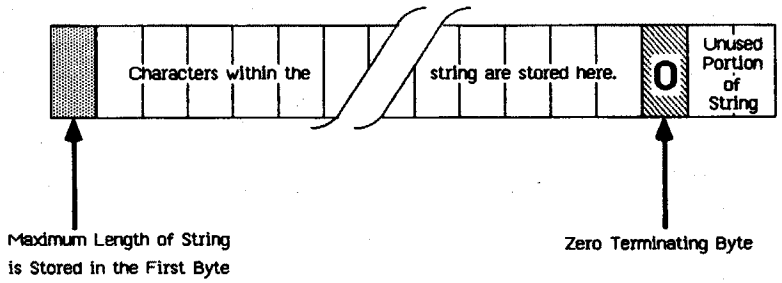


Figure 3. SPEED/ASM string format.

less obvious, difference is that this routine destroys both the X register and the accumulator, while the LENGTH routine modifies only the accumulator.)

The SPEED/ASM PRTSTR routine can be simulated using the loop:

```
        LDX   #0
PRTLOOP LDA   STRING+1,X
        BEQ   PRTDONE
        JSR   PUTC
        INX
        JMP   PRTLOOP
;
PRTDONE:
```

Note that in both cases I loaded from address STRING+1. This skips over the initial maximum length byte present in all SPEED/ASM strings. See Figure 3 again.

Program listing. Demonstration of indexed addressing modes.

```
0800              1          TTL "Listing One: Indexed Modes Ex."
0800              2  ;
0800              3  ;
0800              4  ;
0800              5  ***********************************
0800              6  *                                 *
0800              7  *      Listing One:               *
0800              8  *      Indexed Addressing Modes   *
0800              9  *      Example Program.           *
0800             10  *                                 *
0800             11  ***********************************
0800             12  ;
0800             13  ;
0800             14  ;
0800             15  *************
0800             16  * CONSTANTS *
0800             17  *************
0800             18  ;
0800             19  ;
0000             20  FALSE    EQU 0
0001             21  TRUE     EQU 1
008D             22  CR       EQU $8D
0800             23  ;
0800             24  ;
0800             25  ;
0800             26  ; "IF" STATEMENT EQUATES
0800             27  ;
0800             28  ;
00BD             29  EQ       EQU "="
00A3             30  NE       EQU "#"
00BE             31  GT       EQU ">"
00BC             32  LT       EQU "<"
BDBE             33  GE       EQU ">"|"="*256
BDBC             34  LE       EQU "<"|"="*256
0800             35  ;
0800             36  ;
0800             37  ;
0800             38  ;
0800             39  ;
0800             40  ;
0800             41  ;
0800             42  ***************************
0800             43  * SPEED/ASM ENTRY POINTS *
0800             44  ***************************
0800             45  ;
0800             46  ;
0800             47  ;
0800             48  ;
0800             49  ; NOTE: THE EQUATE OF PUTC MUST
0800             50  ; BE CHANGED IF YOU RELOCATE
0800             51  ; SPEED/ASM TO SOME LOCATION
0800             52  ; OTHER THAN $7800
0800             53  ;
0800             54  ;
0800             55  ;
7800             56  PUTC     EQU $7800
7803             57  GETC     EQU PUTC+3
7806             58  SAGL     EQU GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809             59  SAPC     EQU SAGL+3          ; "   "   "   "   "   "   "
780C             60  HOME     EQU SAPC+3          ;HOME AND CLEAR
780F             61  READLN   EQU HOME+3
```

Listing continued.

Setting Up Integer Arrays

Since integers require two bytes of storage each, an integer array must contain $2 \times n$ bytes, where n is the number of array elements. Consequently, you must define an array with twice the number of bytes as elements. The easiest way to reserve space for an integer array is to use the declaration:

```
<label> DFS 2*<numelmnts>
```

where <label> is the name of the array and <numelmnts> is the number of elements you desire in the array. When using the indexed addressing modes to access elements in an integer array, there is a limit of 128 elements. This is due to the fact that index registers can only access up to 256 different memory locations, which is just enough space for a 128-byte integer array.

One additional problem surfaces when using the index registers to access elements of an integer array: You must load the array index *times two* into the index register in order to access the proper element. There are several ways to do this. If you're loading an immediate value into the index register as an array index, you need only multiply the immediate value by two. For example:

```
LDX #25*2
```

for element 25 of an integer array. The multiplication is performed at assembly time so there is no run-time penalty.

If you need to access array elements using a variable index, you must multiply the index by two beforehand. There are two ways to easily accomplish this: actually multiply the index by two, or, if you're sequentially stepping through an array, increment the index by two for each element you access. For example, if you want to set each element of a 100-byte integer array equal to the index of that element you could use the code:

Listing continued.

```
7812                    62  INIT    EQU READLN+3
7815                    63  FOR     EQU INIT+3
7818                    64  FOR0    EQU FOR+3
781B                    65  NEXT    EQU FOR0+3
781E                    66  IFI     EQU NEXT+3
7821                    67  IFI0    EQU IFI+3
7824                    68  IFS     EQU IFI0+3
7827                    69  IFS0    EQU IFS+3
782A                    70  MOVE    EQU IFS0+3
782D                    71  LOAD    EQU MOVE+3
7830                    72  MOVS    EQU LOAD+3
7833                    73  LDSTR   EQU MOVS+3
7836                    74  PRINT   EQU LDSTR+3
7839                    75  PRTSTR  EQU PRINT+3
783C                    76  PRTINT  EQU PRTSTR+3
783F                    77  RDSTR   EQU PRTINT+3
7842                    78  RDINT   EQU RDSTR+3
0800                    79  ;
7854                    80  ABS     EQU RDINT+18
7857                    81  NEG     EQU ABS+3
785A                    82  MUL     EQU NEG+3
785D                    83  DIV     EQU MUL+3
7860                    84  MOD     EQU DIV+3
7863                    85  RND     EQU MOD+3
7866                    86  SUBSTR  EQU RND+3
7869                    87  INDEX   EQU SUBSTR+3
786C                    88  LENGTH  EQU INDEX+3
786F                    89  CONCAT  EQU LENGTH+3
0800                    90  ;
0800                    91  ;
0800                    92  ;
0800                    93  ;
0800                    94  ; Apple monitor equate
0800                    95  ;
FF69                    96  EXIT    EQU $FF69
0800                    97  ;
0800                    98  *********************************
0800                    99  *
0800                   100  *
0800                   101  * Sample program for Assembly
0800                   102  * Advantage Part six.
0800                   103  *
0800                   104  *   Demonstration of the 6502
0800                   105  * indexed addressing modes.
0800                   106  *
0800                   107  *********************************
0800                   108  ;
0800                   109  ;
0800 20 12 78          110  BEGIN   JSR INIT            ;Always call this first
0803                   111  ;
0803 20 0C 78          112          JSR HOME
0806                   113  ;
0806 20 36 78          114          JSR PRINT
0809 8D 8D 8D          115          BYT CR,CR,CR
080C D4 E8 E9          116          BYT "This program counts the number of",CR
080F F3 A0 F0
0812 F2 EF E7
0815 F2 E1 ED
0818 A0 E3 EF
081B E2 E5 F2
081E A0 EF E6
0821 8D
082E C1 EC F0          117          BYT "Alphabetic, Vowels, Numeric, and",CR
0831 E8 E1 E2
0834 E5 F4 E9
0837 E3 AC A0
083A D6 EF F7
083D E5 EC F3
0840 AC A0 CE
0843 F5 ED E5
0846 F2 E9 E3
0849 AC A0 E1
084C EE E4 8D
084F CE EF EE          118          BYT "Non-numeric characters found on an",CR
0852 AD EE F5
0855 ED E5 F2
0858 E9 E3 A0
085B E3 E8 E1
085E F2 E1 E3
0861 F4 E5 F2
0864 F3 A0 E6
0867 EF F5 EE
086A E4 A0 EF
086D EE A0 E1
0870 EE 8D
0872 E9 EE F0          119          BYT "input line.",CR
0875 F5 F4 A0
0878 EC E9 EE
087B E5 AE 8D
087E 8D                120          BYT CR
087F A0 A0 A0          121          BYT "   To terminate the program press"
0882 D4 EF A0
0885 F4 E5 F2
0888 ED E9 EE
088B E1 F4 E5
088E A0 F4 E8
0891 E5 A0 F0
0894 F2 EF E7
0897 F2 E1 ED
089A A0 F0 F2
089D E5 F3 F3
```

Listing continued.

```
        LDX  #0
SETLOOP TXA
        STA  ARRAY,X
        INX
        LDA  #0 ;Set H.O. byte
             ;to zero
        STA  ARRAY,X
        INX
        CPX  #200 ;100*2 elements
             ; in the array
        BLT  SETLOOP
```

If incrementing the index twice for each element isn't practical, multiplying the array index by two to obtain the byte offset is your only recourse. You should *not*, however, use the SPEED/ASM MUL routine. It is much too slow to use in this fashion. Luckily, there's a little trick you can pull to quickly and easily multiply a number by two—shift it to the left one location. You can do this with the 6502 accumulator by using the ASL (arithmetic shift left) instruction. Consider the following loop:

```
        JSR  FOR0
        ADR  I,1,100
;
        LDA  I
        ASL
        TAX
        LDA  I
        STA  ARRAY,X
        LDA  I+1
        STA  ARRAY+1,X
;
        JSR  NEXT
```

This loop performs the same function as the previous code. It loads the low order byte of I into the accumulator (the high order byte is always zero) and multiplies it by two by shifting it to the left. This data is transferred to the X register with the TAX instruction, and then the array elements are loaded from variable I (low order byte first, high order byte second).

The next program demonstrates one of SPEED/ASM's shortcomings—it has no ability to manipulate integer array elements directly. You must load an array element into an integer variable, manipulate that variable, and then store the integer variable back into the array element. For example, to multiply each element of the above array by 235 you should use the code:

```
        JSR  LOAD
        ADR  235,MULVAL
;
        JST  FOR0
        ADR  I,1,100
;
        LDA  I
        ASL
        TAX
        LDA  ARRAY,X
        STA  J
        LDA  ARRAY+1,X
        STA  J+1
```

Listing continued.

```
08A0 F3 F0 E1   122          BYT "space as the first character on",CR
08A3 E3 E5 A0
08A6 E1 F3 A0
08A9 F4 E8 E5
08AC A0 E6 E9
08AF F2 F3 F4
08B2 A0 E3 E8
08B5 E1 F2 E1
08B8 E3 F4 E5
08BB F2 A0 EF
08BE EE 8D
08C0 F4 E8 E5   123          BYT "the line.",CR,CR
08C3 A0 EC E9
08C6 EE E5 AE
08C9 8D 8D
08CB 00         124          BYT 0
08CC            125  ;
08CC            126  ;
08CC 20 36 78   127  READLOOP JSR PRINT
08CF 8D C5 EE   128          BYT CR,"Enter a line of text: ",0
08D2 F4 E5 F2
08D5 A0 E1 A0
08D8 BC E9 EE
08DB E5 A0 EF
08DE E6 A0 F4
08E1 E5 F8 F4
08E4 BA A0 00
08E7 20 0F 78   129          JSR READLN
08EA 20 3F 78   130          JSR RDSTR
08ED 23 0A      131          ADR INPUTSTR
08EF            132  ;
08EF            133  ; See if the first character is a space
08EF            134  ;
08EF AD 24 0A   135          LDA INPUTSTR+1
08F2 C9 A0      136          CMP #" "
08F4 D0 03      137          BNE NOEXIT
08F6 4C 69 FF   138          JMP EXIT
08F9            139  ;
08F9            140  ;
08F9            141  ; Now count the vowels, alphabetics
08F9            142  ; numerics, and other chars.
08F9            143  ;
08F9 20 2D 78   144  NOEXIT  JSR LOAD
08FC 00 00 1B   145          ADR 0,VOWEL
08FF 0A
0900 20 2D 78   146          JSR LOAD
0903 00 00 1D   147          ADR 0,ALPHAS
0906 0A
0907 20 2D 78   148          JSR LOAD
090A 00 00 1F   149          ADR 0,NUMERICS
090D 0A
090E 20 2D 78   150          JSR LOAD
0911 00 00 21   151          ADR 0,CHARS
0914 0A
0915            152  ;
0915 A2 00      153          LDX #0
0917 BD 24 0A   154  CNTLOOP LDA INPUTSTR+1,X    ;Get char
091A D0 7F      155          BNE TSTALPHA
091C            156  ;
091C            157  ;
091C            158  ; If at the end of the line, print
091C            159  ; the data out.
091C            160  ;
091C 20 36 78   161          JSR PRINT
091F 8D 8D CE   162          BYT CR,CR,"Number of vowels: ",0
0922 F5 ED E2
0925 E5 F2 A0
0928 EF F6 A0
092B F6 EF F7
092E E5 EC F3
0931 BA A0 00
0934 20 3C 78   163          JSR PRTINT
0937 1B 0A      164          ADR VOWEL
0939            165  ;
0939 20 36 78   166          JSR PRINT
093C 8D CE F5   167          BYT CR,"Number of alphabetics: ",0
093F ED E2 E5
0942 F2 A0 EF
0945 E6 A0 E1
0948 EC F0 E8
094B E1 E2 E5
094E F4 E9 E3
0951 F3 BA A0
0954 00
0955 20 3C 78   168          JSR PRTINT
0958 1D 0A      169          ADR ALPHAS
095A            170  ;
095A 20 36 78   171          JSR PRINT
095D 8D CE F5   172          BYT CR,"Number of numerics: ",0
0960 ED E2 E5
0963 F2 A0 EF
0966 E6 A0 EE
0969 F5 ED E5
096C F2 E9 E3
096F F3 BA A0
0972 00
```

Listing continued.

Listing continued.

```
0973 20 3C 78   173           JSR PRTINT
0976 1F 0A      174           ADR NUMERICS
0978            175  ;
0978 20 36 78   176           JSR PRINT
097B 8D CE F5   177           BYT CR,"Number of characters: ",0
097E ED E2 E5
0981 F2 A0 EF
0984 E6 A0 E3
0987 E8 E1 F2
098A E1 E3 F4
098D E5 F2 F3
0990 BA A0 00
0993 20 3C 78   178           JSR PRTINT
0996 21 0A      179           ADR CHARS
0998 4C CC 08   180           JMP READLOOP
099B            181  ;
099B            182  ;
099B            183  ; First, count the alphabetic chars
099B            184  ;
099B C9 C1      185  TSTALPHA CMP #"A"
099D 90 1D      186           BLT NOTALPHA
099F C9 DB      187           CMP #"Z"+1
09A1 90 08      188           BLT ISALPHA
09A3 C9 E1      189           CMP #"a"
09A5 90 15      190           BLT NOTALPHA
09A7 C9 FB      191           CMP #"z"+1
09A9 B0 11      192           BGE NOTALPHA
09AB            193  ;
09AB 18         194  ISALPHA  CLC
09AC A9 01      195           LDA #1
09AE 6D 1D 0A   196           ADC ALPHAS
09B1 8D 1D 0A   197           STA ALPHAS
09B4 A9 00      198           LDA /1
09B6 6D 1E 0A   199           ADC ALPHAS+1
09B9 8D 1E 0A   200           STA ALPHAS+1
09BC            201  ;
09BC            202  ;
09BC            203  ; Now count the vowels
09BC            204  ;
09BC BD 24 0A   205  NOTALPHA LDA INPUTSTR+1,X
09BF A0 0D      206           LDY #13
09C1 D9 0D 0A   207  VWLCMP   CMP VOWELS,Y
09C4 F0 05      208           BEQ ISAVWL
09C6 88         209           DEY
09C7 10 F8      210           BPL VWLCMP
09C9 30 11      211           BMI NOTAVWL
09CB            212  ;
09CB 18         213  ISAVWL   CLC
09CC A9 01      214           LDA #1
09CE 6D 1B 0A   215           ADC VOWEL
09D1 8D 1B 0A   216           STA VOWEL
09D4 A9 00      217           LDA /1
09D6 6D 1C 0A   218           ADC VOWEL+1
09D9 8D 1C 0A   219           STA VOWEL+1
09DC            220  ;
09DC            221  ;
09DC            222  ;
09DC            223  ; If numeric, increment that counter.
09DC            224  ;
09DC            225  ;
09DC BD 24 0A   226  NOTAVWL  LDA INPUTSTR+1,X
09DF C9 B0      227           CMP #"0"
09E1 90 15      228           BLT NOTANUM
09E3 C9 BA      229           CMP #"9"+1
09E5 B0 11      230           BGE NOTANUM
09E7            231  ;
09E7 18         232           CLC
09E8 AD 1F 0A   233           LDA NUMERICS
09EB 69 01      234           ADC #1
09ED 8D 1F 0A   235           STA NUMERICS
09F0 AD 20 0A   236           LDA NUMERICS+1
09F3 69 00      237           ADC /1
09F5 8D 20 0A   238           STA NUMERICS+1
09F8            239  ;
09F8            240  ;
09F8            241  ;
09F8            242  ; Increment the character count
09F8            243  ;
09F8 18         244  NOTANUM  CLC
09F9 AD 21 0A   245           LDA CHARS
09FC 69 01      246           ADC #1
09FE 8D 21 0A   247           STA CHARS
0A01 AD 22 0A   248           LDA CHARS+1
0A04 69 00      249           ADC /1
0A06 8D 22 0A   250           STA CHARS+1
0A09            251  ;
0A09            252  ; Move on to next character and repeat
0A09            253  ;
0A09 E8         254           INX
0A0A 4C 17 09   255           JMP CNTLOOP
0A0D            256  ;
0A0D            257  ;
0A0D            258  ;
0A0D            259  ;
0A0D C1 C5 C9   260  VOWELS   BYT "AEIOUYWaeiouyw"
0A10 CF D5 D9
0A13 D7 E1 E5
```

Listing continued.

Listing continued.

```
0A16 E9 EF F5
0A19 F9 F7
0A1B              261  ;
0A1B              262  ;
0A1B 00 00        263  VOWEL    ADR 0
0A1D 00 00        264  ALPHAS   ADR 0
0A1F 00 00        265  NUMERICS ADR 0
0A21 00 00        266  CHARS    ADR 0
0A23              267  ;
0A23              268  ;
0A23 80 00        269  INPUTSTR ADR 128
0A25              270           DFS 128
0AA5              271  ;
0AA5              272           END

***** END OF ASSEMBLY
!brun sort
BRUN SORT
SYMBOL TABLE SORTED ALPHABETICALLY
  ABS       7854  ALPHAS    0A1D  BEGIN     0800  CHARS     0A21  CNTLOOP   0917
  CONCAT    786F  CR        008D  DIV       785D  EQ        00ED  EXIT      FF69
  FALSE     0000  FOR       7815  FOR0      7818  GE        BDBE  GETC      7803
  GT        00BE  HOME      780C  IFI       781E  IFI0      7821  IFS       7824
  IFS0      7827  INDEX     7869  INIT      7812  INPUTSTR  0A23  ISALPHA   09AB
  ISAVWL    09CB  LDSTR     7833  LE        BDBC  LENGTH    786C  LOAD      782D
  LT        00BC  MOD       7860  MOVE      782A  MOVS      7830  MUL       785A
  NE        00A3  NEG       7857  NEXT      781B  NOEXIT    08F9  NOTALPHA  09BC
  NOTANUM   09F8  NOTAVWL   09DC  NUMERICS  0A1F  PRINT     7836  PRTINT    783C
  PRTSTR    7839  PUTC      7800  RDINT     7842  RDSTR     783F  READLN    780F
  READLOOP  08CC  RND       7863  SAGL      7806  SAPC      7809  SUBSTR    7866
  TRUE      0001  TSTALPHA  099B  VOWEL     0A1B  VOWELS    0A0D  VWLCMP    09C1
SYMBOL TABLE SORTED BY ADDRESS
  FALSE     0000  TRUE      0001  CR        008D  NE        00A3  LT        00BC
  EQ        00ED  GT        00BE  BEGIN     0800  READLOOP  08CC  NOEXIT    08F9
  CNTLOOP   0917  TSTALPHA  099B  ISALPHA   09AB  NOTALPHA  09BC  VWLCMP    09C1
  ISAVWL    09CB  NOTAVWL   09DC  NOTANUM   09F8  VOWELS    0A0D  VOWEL     0A1B
  ALPHAS    0A1D  NUMERICS  0A1F  CHARS     0A21  INPUTSTR  0A23  PUTC      7800
  GETC      7803  SAGL      7806  SAPC      7809  HOME      780C  READLN    780F
  INIT      7812  FOR       7815  FOR0      7818  NEXT      781B  IFI       781E
  IFI0      7821  IFS       7824  IFS0      7827  MOVE      782A  LOAD      782D
  MOVS      7830  LDSTR     7833  PRINT     7836  PRTSTR    7839  PRTINT    783C
  RDSTR     783F  RDINT     7842  ABS       7854  NEG       7857  MUL       785A
  DIV       785D  MOD       7860  RND       7863  SUBSTR    7866  INDEX     7869
  LENGTH    786C  CONCAT    786F  LE        BDBC  GE        BDBE  EXIT      FF69
```

```
;
   JSR   MUL
   ADR   MULVAL,J,J
;
   LDA   J
   STA   ARRAY,X
   LDA   J+1
   STA   ARRAY+1,X
;
   JSR   NEXT
```

This month's demonstration program (see the listing) shows various ways of using integer and character arrays in SPEED/ASM.

The indexed addressing modes on the 6502 suffer from one major disadvantage—they only allow access to 256 contiguous memory locations. Next month I'll discuss the 6502's indirect indexed addressing modes, which make broader access possible. These modes alleviate the major problem encountered when dealing with arrays on the 6502. ■

The Assembly Advantage

by Randy Hyde

6502 Indirect Addressing Modes

The 6502 indexed addressing modes, discussed in last month's column, are useful for implementing small arrays and for accessing elements of strings in known locations. For large arrays, and for arrays and strings whose address is not known at assembly time, the 6502 *indirect indexed* addressing modes are required.

So far you've met three basic types of addressing modes on the 6502 microprocessor: the immediate, the absolute, and the indexed. (For our purposes, the relative addressing mode, used by the branches, is identical to the absolute mode.) An addressing mode specifies where in memory data to be accessed is located. For example, the *immediate* mode tells the 6502 that the data is to be found *immediately after the instruction's opcode byte*. The *absolute* mode, rather than providing the data itself, follows the instruction opcode with the *address of the data* in memory.

The 6502 *indirect* addressing mode is a logical extension of this sequence. Rather than following the instruction opcode with the data to be accessed or the address of the data, it is followed with the *address of the address of the data*. Figure 1 pictures how the various addressing modes function; Figure 2 is a closer view of the indirect mode.

The indexed addressing modes on the 6502 add the contents of the X or Y register to the absolute address that

Randy Hyde is the proprietor of Lazerware, creators of SPEED/ASM and the LISA assembler. You can write to him at 925 Lorna St., Corona, CA 91720.

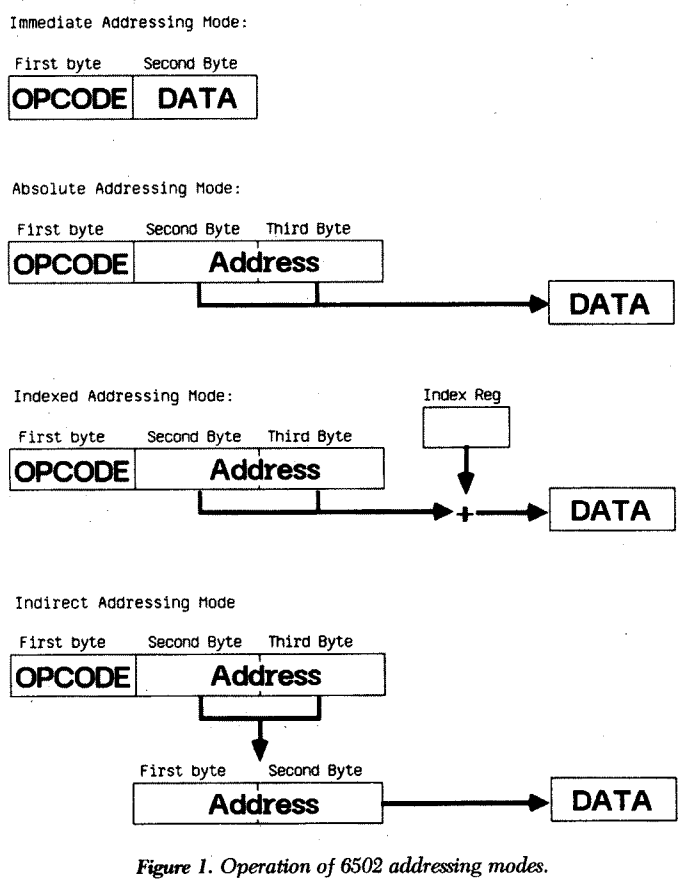


Figure 1. Operation of 6502 addressing modes.

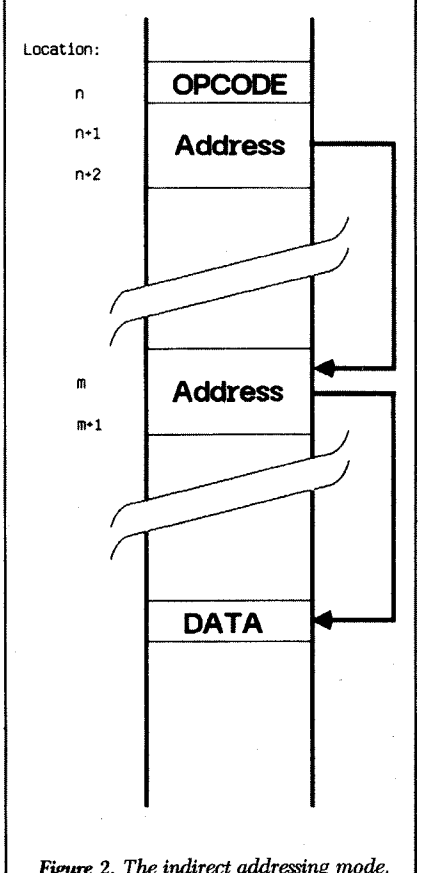


Figure 2. The indirect addressing mode.

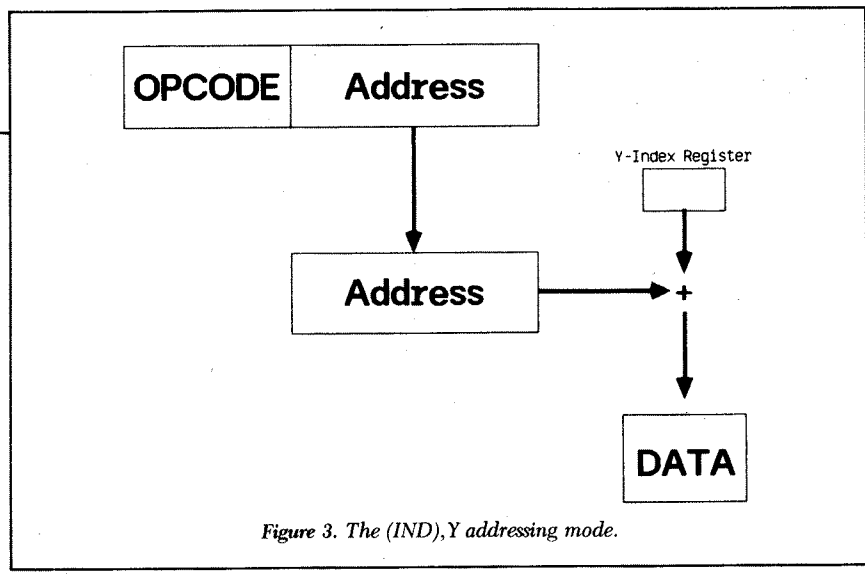


Figure 3. The (IND),Y addressing mode.

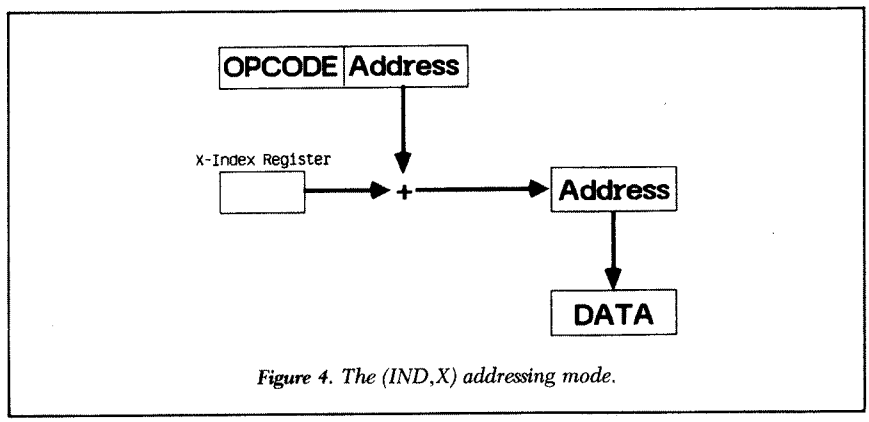


Figure 4. The (IND,X) addressing mode.

follows the instruction opcode to obtain the *effective address*. By applying this same technique to the indirect addressing modes you come up with the *indirect, indexed* mode. When using this mode, the value in the Y register is added to the value pointed at by the byte following the instruction opcode. This sum provides the *true effective address* for the instruction. See Figure 3.

A second form of indirect addressing—*indexed, indirect*—is also available. Here the contents of the X index register are added to the address immediately following the instruction opcode to obtain the address of the address of the data you're interested in. The operation of the indexed, indirect mode is shown in Figure 4.

Nasty Reality #1:
The Indirect Jump Instruction

There's only one problem with the 6502's indirect addressing modes: They can only be used with the JMP instruction. You can't load, add, subtract, or do anything else with them in their pure form.

The syntax for the indirect JMP instruction is:

JMP (ADRS)

where ADRS is the address of a two-byte pointer containing the address where you want to jump. ADRS must point at the low-order byte of the new address, and location ADRS+1 must contain the high-order byte.

There is a nasty little bug in the 6502 chip that can get you into a lot of trouble if you're unaware. The two-byte address pointed at by ADRS *must* be totally contained within a single page of memory. If ADRS holds the value xxFF (where xx is any single-byte value), then the 6502 fetches the low-order byte from location xxFF (as you'd expect) and the high-order byte from location xx00. Note that you really wanted the high-order byte fetched from location yy00, where yy = xx + 1.

Since the purpose of this month's column is to discuss implementing arrays, I will leave the discussion of the JMP indirect instruction for later. Right now let's worry about implementing large arrays with the indirect, indexed addressing modes.

Nasty Reality #2:
The Zero Page Addressing Mode

Unfortunately, I must introduce yet another 6502 addressing mode before continuing the discussion of the indirect mode. In reality, the indirect mode isn't just a combination of two modes (indirect and indexed), but rather *three* modes: indirect, indexed, and *zero page*.

The 64K address space of the 6502 is divided into 256 groups of 256 bytes each. Each block of 256 bytes is called a *page*, and the pages are numbered sequentially. Page zero is the first page of memory (addresses $0000–$00FF), page one is the second page (addresses $0100–$01FF), etc. The zero page addressing mode gets its name from the fact that it only allows you to access the first 256 bytes in the 64K address range, i.e., page zero.

There are two advantages to the zero page mode (compared to the absolute mode): An instruction using the zero page mode is one byte shorter and one microsecond faster than the equivalent instruction using the absolute mode. There are also two big disadvantages: You can only access the first 256 bytes of memory in the 6502's address space, and page zero is prime real estate—everyone else wants to use it too. In particular, DOS, ANIX, Basic, the Apple monitor, Pascal, CP/M (actually the BIOS drivers), SPEED/ASM, and many other programs that your SPEED/ASM program must co-exist with, all use some zero page memory. If you use the same location as SPEED/ASM or DOS, you can make the system crash. So use page zero only when you have to, and make sure you're not using any zero page locations occupied by a coresident system. Typically locations $50–$6F are available when operating SPEED/ASM under Apple DOS.

Declaring a zero page variable is

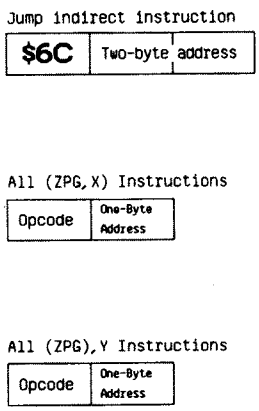


Figure 5. Indirect instruction formats.

done quite a bit differently than declaring normal SPEED/ASM variables. Rather than using the DFS or ADR pseudo-opcode to reserve space for the zero page variable, you must use the EPZ, "equate to page zero," pseudo-upcode. The syntax for EPZ is:

<varname> EPZ <address>

To explain, <address> is the actual address in memory where the variable is to be stored. Since this is a zero page variable, you must make sure that <address> is in the range $00–$FF, or LISA will signal an error when you attempt to assemble the program. Since SPEED/ASM variables (and 6502 pointers for that matter) require two bytes, make sure that both locations <address> and <address> + 1 in page zero are open, because both will be used by the SPEED/ASM routines and the indirect addressing modes.

Once you've "equated" a symbol to a zero page address using the EPZ pseudo-opcode, you can treat that label exactly like any other SPEED/ASM integer variable. To initialize a variable declared in page zero, you could use the LOAD or MOVE SPEED/ASM routine. To access it you could use any valid 6502 instruction (that works with the absolute addressing mode), or any of the SPEED/ASM integer functions like MUL, DIV, PRTINT, etc. When using a pure 6502 instruction, you will usually notice

```
STRPTR    EPZ   $50
;
          JSR   LOAD              ;Copy address of STRING
          ADR   STRING,STRPTR     ; into STRPTR
;
          LDY   #0
PRTLOOP   LDA   (STRPTR),Y
          BEQ   PRTDONE
          JSR   PUTC
          INY
          JMP   PRTLOOP
;
PRTDONE:
```

Listing 1a.

```
          LDY   #0
PRTLOOP   LDA   STRING,Y
          BEQ   PRTDONE
          JSR   PUTC
          INY
          JMP   PRTLOOP
;
PRTDONE:
```

Listing 1b.

```
          JSR   LOAD
          ADR   STRING1,STRPTR
          JMP   PRTIT
          .     .
          .     .
          .     .
          JSR   LOAD
          ADR   STRING2,STRPTR
          JMP   PRTIT
          .     .
          .     .
          .     .
          JSR   LOAD
          ADR   STRING3,STRPTR
          JMP   PRTIT
          .     .
          .     .
          .     .
PRTIT     LDY   #0
PRTLOOP   LDA   (STRPTR),Y
          BEQ   PRTDONE
          JSR   PUTC
          INY
          JMP   PRTLOOP
;
PRTDONE:
```

Listing 2.

that only two bytes of object code (instead of the normal three) are emitted. This is because the high-order byte of zero is implied by the use of the zero page addressing mode.

Back to the Indirect Addressing Modes

Before I digressed to a discussion of the zero page addressing mode, I mentioned that the indirect, indexed, and the indexed, indirect modes are actually combinations of three modes: indirect, indexed and zero page. All indirect instructions (except jump indirect) are two bytes long. The first byte is the 6502 opcode and the second byte is the zero page address of the pointer to the memory location you're interested in. See Figure 5.

Note that the program sequences in Listings 1a and 1b perform equivalent functions. Also note that in the case of the indirect, indexed-by-Y mode you are still limited to 256 bytes due to the Y index register's 8-bit limitation. So why use the indirect addressing mode? It doesn't appear to provide any additional features; in fact, it makes a somewhat complex process (indexed addressing) even more complex.

Its beauty lies in the fact that the pointer can be changed *under program control.* For example, the Listing 1a program segment (using the indexed mode) is forever limited to printing the string STRING. Listing 1b can be changed to print any string

```
;   Assume SPEED/ASM variable "I" contains the
;   index into the byte array "B".
;
            CLC
            LDA   I
            ADC   #B
            STA   ARRAYPTR
            LDA   I+1
            ADC   /B
            STA   ARRAYPTR+1
;
            LDY   #0
            LDA   (ARRAYPTR),Y      ;Loads B[I] into accumulator.
```

Listing 3.

by simply changing the LOAD instruction before the print loop. For example, consider the code in Listing 2. One of three different strings will be printed, depending upon the value loaded into STRPTR.

While this example certainly justifies the existence of the indirect, indexed-by-Y mode, it still doesn't show how to access more than 256 bytes using the indirect mode. And that's the whole purpose of this month's column—to describe how to access elements of an array containing more than 256 bytes.

The secret to accessing large blocks of data using the indirect addressing mode is to modify the two-byte pointer instead of the Y register. By setting the Y register to zero (which makes the indirect, indexed-by-Y mode behave *exactly* like a true indirect addressing mode) and then incrementing the two-byte zero page pointer, you can access up to 64K of data (the amount of memory accessible with a two-byte pointer). If that's not enough memory, you're using the wrong microprocessor!

To access an element of a byte array that contains more that 256 elements, the address of the desired element can be computed by the formula:

<adrs> = <base address> + <index>

where <base address> is the address of element zero of the array and <index> is the number of the desired array element. This calculation can be performed using the 6502 code in Listing 3.

When accessing elements of an integer array, don't forget that each element requires two bytes, so the index value must be multiplied by two before adding it to the base address. The quickest way to double the index is to shift it one position to the left. This is accomplished with the code in Listing 4.

The ASL instruction shifts the data in memory location I to the left one position. A zero is shifted into the low-order bit position, and the data in bit number seven is shifted into the carry flag. The ROL (rotate left) instruction shifts the high-order byte of I. The difference between the two instructions is that the contents of the carry flag (i.e., the bit shifted out of bit seven of the low-order byte) is shifted into the low-order bit of location I + 1.

In many cases it won't matter if you double the value of I. Sometimes you will reload it anyway. But in some instances, particularly if I is the index variable of a FOR loop or some other control variable, you can't leave the value doubled. To undouble a value (divide it by two) the LSR (logical shift right) and ROR (rotate right) instructions are used after the calculation to restore the value of I. *Note that the high-order byte of I is shifted to the right before the low-order byte.* This is exactly opposite to when the shift left function is used to double the value. I'll talk more about the shift instructions in a future column; for now, just duplicate these instructions verbatim.

```
        ASL   I              ;Multiply the index by two
        ROL   I+1            ; before adding it to the base address.
        CLC
        LDA   I
        ADC   BASE
        STA   ARRAYPTR
        LDA   I+1
        ADC   BASE+1
        STA   ARRAYPTR+1
;
        LDY   #0
        LDA   (ARRAYPTR),Y   ;Get L.O. byte of array element.
        STA   J              ;Save L.O. byte.
        INY
        LDA   (ARRAYPTR),Y   ;Get H.O. byte of array element.
        STA   J+1            ;Save H.O. byte.
;
        LSR   I+1            ;Divide the index by two to
        ROR   I              ;set it to its original value.
```

Listing 4.

Multi-Dimensional Arrays

Handling one-dimensional arrays is easy. The formula:

<adrs> = <base address> + (<index> *<element size>)

(where <element size> is the number of bytes required by each array element) is completely adequate. Multiple-dimension arrays are a little more difficult and a lot more time consuming.

For a two-dimensional array, the formula becomes:

<adrs> = BA + (NDX1 + NDX2*SIZE1)*WS

where BA is the base address of the array dimensioned as ARRAY[SIZE1, SIZE2] and accessed as ARRAY[NDX1, NDX2], and each element occupies WS bytes. The multiplication by WS is easy, assuming a character or integer array. Shift to the left if integer, and ignore the multiplication by WS if character. The other multiplication (NDX2*SIZE1), however, cannot be handled with a simple shift in most cases. This one requires a real multiply, and multiplies are very slow.

Three-dimensional arrays are even worse—two multiplies are required. The formula for calculating the address of an element of a three-dimensional array is:

<adrs> = BA + (NDX1 + (NDX2 + (NDX3*SIZE2))*SIZE1)*WS

If you need to use higher-dimensioned arrays, consider using a high level language capable of supporting your data structure needs.

Speeding Things Up

Normally when you access array elements you go for adjacent elements rather than random locations within the array. Once you've calculated the address of an element, obtaining the addresses of adjacent elements is easy. If you want to access the *next* location, simply add WS to the current address. If you want to access the *previous* location, subtract WS from the current address.

When dealing with byte (character) arrays, all you need do is increment or decrement the address by one. While you *could* use the ADC instruction sequence to add one to the pointer, there is a tricky way to add one quickly to a two-byte integer value. The code to accomplish this is:

```
        INC   VAR
        BNE   >0
        INC   VAR+1
^0:
```

where VAR is the name of the pointer (or any SPEED/ASM or two-byte integer value) you wish to increment.

How does it work? Well, the first increment instruction adds one to the low-order byte of the variable. Now, the only time overflow occurs when incrementing by one is with the value $FF because then you wind up with $00. This is the *only* time you wind up with zero, and likewise it's the only time the 6502 zero flag is set after the increment. Whenever overflow occurs, you must add one to the high-order byte of the variable, so the BNE instruction skips the INC VAR+1 instruction. If the Z flag is set, then the 6502 drops through and increments the high-order byte.

The sequence above only works for character or single-byte arrays. When dealing with an integer array you need the code:

```
CLC
LDA   VAR
ADC   #2
STA   VAR
BCC   >0
INC   VAR+1
^0:
```

I could have incremented VAR twice, but this code is faster and shorter. Incidentally, the trick here is to recognize that only the carry gets added to VAR+1, so rather than perform the explicit addition, I only incremented VAR+1 whenever there was a carry out of the low-order byte. This trick can be used when adding any 8-bit value to a 16-bit integer variable.

The Indexed-by-X, Indirect Addressing Mode

The indexed-by-X, indirect addressing mode on the 6502 is severely handicapped by the requirement of zero page. The way this mode works is, the X register is added to the zero page address that follows the instruction opcode. Then this sum points at the low-order byte of a pointer that points to the data to be accessed. This addressing mode lets you set up a table (or array, if you will) of pointers, and index into this array of pointers to find the data. Unfortunately, this mode presupposes lots of zero page at your disposal. Since this is probably not the case, the indexed-by-X, indirect addressing mode isn't very useful.

There is one exception, however. If you load the X register with zero, then the indexed-by-X, indirect mode degenerates to a pure indirect addressing mode. Since this is usually what you're interested in when accessing an element of a multi-dimensional array, the indexed-by-X, indirect addressing mode may prove useful on occasion. ■

The Assembly Advantage

by Randy Hyde

Control Structures: Review and New

So far I've discussed five control structures in SPEED/ASM: the JMP instruction (unconditional goto), the JSR instruction (jump to subroutine), the SPEED/ASM IFx and IFx0 routines, the FOR/FOR0 loop and the 6502 branch instructions. SPEED/ASM supports several additional control structures, many of them extremely powerful and easy to use. Included in this set of routines are ONXGOTO, CASE, CASEI, INSET, and NOTINSET.

A Quick Review of SPEED/ASM Control Structures

Before describing the new control structures, a quick review of the control routines I've presented thus far may help tie up any loose ends.

● *The JMP instruction* is actually a 6502 instruction. It is almost identical to the GOTO statement in Basic—it transfers control to a new statement which doesn't necessarily follow the JMP. The syntax for the JMP instruction is:

JMP <label>

where <label> is a valid statement label in your SPEED/ASM program.

● *The JSR instruction* is used to call a 6502 or SPEED/ASM subroutine. *The RTS instruction* is used to return from a user subroutine. The JSR/RTS combination is used identically to the GOSUB/RETURN statements in Basic (except, of course, you specify a statement label instead of a line number). The syntax for the JSR statement is:

JSR <label>

where <label> is the name of the user subroutine you wish to call. All user subroutines should be terminated with the RTS instruction. The RTS instruction does not allow any operands; its syntax is:

RTS

The JSR instruction also is used to call routines in the SPEED/ASM package, since all SPEED/ASM routines are nothing more than 6502 subroutines.

● *The SPEED/ASM IFx/IFx0 routines* come in two flavors (actually three, but I've only discussed two versions so far): the IFI/IFI0 routines and the IFS/IFS0 routines. The IFI/IFI0 routines are used to compare two integer values. IFI compares two in-

Listing. Demonstration SPEED/ASM program using routines ONXGOTO, CASE, CASEI, INSET and NOTINSET. These control structures will construct flexible menu programs.

```
0800                1           ttl "Speed/asm Sample pgm #8"
0800                2  ;
0800                3  ;
0800                4  ;
0800                5  ***********************************
0800                6  *                                 *
0800                7  *  Speed/asm sample program #8    *
0800                8  *                                 *
0800                9  *  Written by Randall Hyde        *
0800               10  *  Written on LISA v2.6           *
0800               11  *  7/22/83                        *
0800               12  *                                 *
0800               13  ***********************************
0800               14  ;
0800               15  ;
0800               16  ;
0800               17  ;
0800               18  ***********************************
0800               19  *                                 *
0800               20  *       SPEED/ASM Equates         *
0800               21  *                                 *
0800               22  ***********************************
0800               23  ;
0800               24  ;
0800               25  ;
0800               26  ;
0800               27  ;
0800               28  ;
0800               29  ;
0800               30  ;
0800               31  ; GENERAL PURPOSE EQUATES
0800               32  ;
0800               33  ; The following variables are used
0800               34  ; by the SPEED/ASM package and
0800               35  ; shouldn't be used by the SPEED/ASM
0800               36  ; programmer.
0800               37  ;
0800               38  ;
0800               39  ;
0800               40  ;
0000               41  FORASAV  epz 0
0001               42  FORXSAV  epz FORASAV+1
0002               43  FORYSAV  epz FORXSAV+1
0003               44  FORZPG   epz FORYSAV+1
```

Listing continued.

Randy Hyde is proprietor of Lazerware, creators of SPEED/ASM and LISA. Address correspondence to him at 925 Lorna St., Corona, CA 91720.

Listing continued.

```
0005              45  DESTADR  epz FOR2PG+2
0007              46  PTRADR   epz DESTADR+2
0009              47  ISIMMED  epz PTRADR+2
000A              48  OP       epz ISIMMED+1
000C              49  MAXLEN   epz OP+2
000D              50  VALUE    epz MAXLEN+1
000F              51  DIGIT    epz VALUE+2
0010              52  LEAD0    epz DIGIT+1
0011              53  JMPADR   epz LEAD0+1
0013              54  COUNT    epz JMPADR+2
0014              55  GOTLN    epz COUNT+1
0015              56  LINEINDX epz GOTLN+1
0016              57  SIGN     epz LINEINDX+1
0017              58  ACL      epz SIGN+1
0018              59  ACH      epz ACL+1
0019              60  XTNDL    epz ACH+1
001A              61  XTNDH    epz XTNDL+1
001B              62  AUXL     epz XTNDH+1
001C              63  AUXH     epz AUXL+1
0800              64  ;
0033              65  PROMPT   epz $33
004E              66  RNDL     epz $4E
004F              67  RNDH     epz $4F
0100              68  STACK    equ $100
0200              69  INPUT    equ $200
0800              70  ;
0800              71  ;
0800              72  ;
0800              73  ;
0800              74  ;
0800              75  ;
0800              76  ;
0800              77  *************
0800              78  * CONSTANTS *
0800              79  *************
0800              80  ;
0800              81  ;
0800              82  ;
0800              83  ; The following symbols are constants
0800              84  ; for the values "FALSE", "TRUE", and
0800              85  ; Carriage Return (respectively).
0800              86  ;
0800              87  ; These symbols should only appear
0800              88  ; as immediate operands to a 6502
0800              89  ; instruction or in the operand field
0800              90  ; of a pseudo-opcode like BYT.
0800              91  ;
0800              92  ;
0800              93  ;
0800              94  ;
0800              95  ;
0000              96  FALSE    equ 0
0001              97  TRUE     equ 1
008D              98  CR       equ $8D
0087              99  BELL     equ $87
0800             100  ;
0800             101  ;
0800             102  ;
0800             103  ;
0800             104  ;
0800             105  ;
0800             106  ; "IF" STATEMENT EQUATES
0800             107  ;
0800             108  ; The following symbols should only
0800             109  ; be used in the ADR pseudo-opcode
0800             110  ; following a call to the SPEED/ASM
0800             111  ; IFx routines.
0800             112  ;
0800             113  ;
0800             114  ;
00BD             115  EQ       equ "="
00A3             116  NE       equ "#"
00BE             117  GT       equ ">"
00BC             118  LT       equ "<"
BDBE             119  GE       equ ">"|"="*256
BDBC             120  LE       equ "<"|"="*256
```

Listing continued.

teger variables and IFI0 compares an integer variable to an integer constant. The IFS/IFS0 routines compare two SPEED/ASM strings. IFS compares two string variables and IFS0 compares a string variable to a string constant. The syntax for the IFx routines is:

```
JSR     IFI
ADR     <Ivar1>,<op>,<Ivar2>
```

and

```
JSR     IFS
ADR     <Svar1>,<op>,<Svar2>
```

where <Ivar1> and <Ivar2> are the names of properly defined SPEED/ASM integer variables and <Svar1> and <Svar2> are the names of SPEED/ASM string variables. <op> is any of the SPEED/ASM logical operators:

EQ
NE
LE
GE
LT
GT

as defined in the SPEED/ASM equates file.

"IFS compares two string variables and IFS0 compares a string variable to a string constant."

The IFx0 routines compare a variable to a constant; the syntax for these two instructions is:

```
JSR     IFI0
ADR     <Ivar>,<op>,<Iconst>
```

and

```
JSR     IFS0
ADR     <Svar>,<op>
BYT     "string constant",0
```

where <Iconst> is an integer constant and "string constant" is a *zero terminated* character string.

Immediately after the call to one of the IFx/IFx0 routines you should use the 6502/LISA BTR (branch if true) or BFL (branch if false) instruction to test the comparison for true or false.

Listing continued.

```
0800            121  ;
0800            122  ;
0800            123  ;
0800            124  ;
0800            125  ;
0800            126  ;
0800            127  ;
0800            128  ***************************
0800            129  * SPEED/ASM ENTRY POINTS *
0800            130  ***************************
0800            131  ;
0800            132  ;
0800            133  ;
0800            134  ;
0800            135  ; NOTE: THE EQUATE OF PUTC MUST
0800            136  ; BE CHANGED IF YOU RELOCATE
0800            137  ; SPEED/ASM TO SOME LOCATION
0800            138  ; OTHER THAN $7800
0800            139  ;
0800            140  ;
0800            141  ;
7800            142  PUTC     equ $7800
7803            143  GETC     equ PUTC+3
7806            144  SAGL     equ GETC+3          ;FOR USE BY S/A ONLY- SEE DOC.
7809            145  SAPC     equ SAGL+3          ; "   "   "   "   "     "   "
780C            146  HOME     equ SAPC+3          ;HOME AND CLEAR
780F            147  READLN   equ HOME+3
7812            148  INIT     equ READLN+3
7815            149  FOR      equ INIT+3
7818            150  FOR0     equ FOR+3
781B            151  NEXT     equ FOR0+3
781E            152  IFI      equ NEXT+3
7821            153  IFI0     equ IFI+3
7824            154  IFS      equ IFI0+3
7827            155  IFS0     equ IFS+3
782A            156  MOVE     equ IFS0+3
782D            157  LOAD     equ MOVE+3
```

Listing continued.

The SPEED/ASM FOR/FOR0 instructions emulate the Basic loops of the same name. The syntax for the FOR0 loop is:

```
JSR     FOR0
ADR     <Ivar>,<start>,<end>
  .
  .
  .
JSR     NEXT
```

where <Ivar> is the name of a SPEED/ASM integer variable and <start> and <end> are integer constants. This emulates Basic statements of the form:

FOR I = 1 TO 10

The SPEED/ASM FOR loop handles the case where integer variables are required for the starting, ending or stepsize variables. The syntax for the FOR loop is:

JSR FOR

```
ADR    <Ivar>,<vstrt>,<vend>,<vstep>
  .
  .
  .
JSR    NEXT
```

where <Ivar>, <vstrt>, <vend>, and <vstep> all are SPEED/ASM integer variable names.

To compare two single byte values pure 6502 code is used. The 6502 CMP instruction, along with the various branch instructions, lets you compare a value in memory to the value in the accumulator. After the instruction CMP<operand> where <operand> is any of:

<Ivar>
#<Ivar>
/<Ivar>
<Ivar>,X
<Ivar>,Y
(ZPG,X)
(ZPG),Y

the 6502 branch instructions can be

Listing continued.

```
7830            158  MOVS      equ LOAD+3
7833            159  LDSTR     equ MOVS+3
7836            160  PRINT     equ LDSTR+3
7839            161  PRTSTR    equ PRINT+3
783C            162  PRTINT    equ PRTSTR+3
783F            163  RDSTR     equ PRTINT+3
7842            164  RDINT     equ RDSTR+3
7845            165  ONXGOTO   equ RDINT+3
7848            166  CASE      equ ONXGOTO+3
784B            167  CASEI     equ CASE+3
784E            168  INSET     equ CASEI+3
7851            169  NOTINSET  equ INSET+3
7854            170  ABS       equ NOTINSET+3
7857            171  NEG       equ ABS+3
785A            172  MUL       equ NEG+3
785D            173  DIV       equ MUL+3
7860            174  MOD       equ DIV+3
7863            175  RND       equ MOD+3
7866            176  SUBSTR    equ RND+3
7869            177  INDEX     equ SUBSTR+3
786C            178  LENGTH    equ INDEX+3
786F            179  CONCAT    equ LENGTH+3
7872            180  GETWZPG   equ CONCAT+3      ;USED BY SPEED/ASM
7875            181  RDFP      equ GETWZPG+3
7878            182  PRTE      equ RDFP+3
787B            183  PRTF      equ PRTE+3
787E            184  FADD      equ PRTF+3
7881            185  FSUB      equ FADD+3
7884            186  FMUL      equ FSUB+3
7887            187  FDIV      equ FMUL+3
788A            188  FLT       equ FDIV+3
788D            189  FIX       equ FLT+3
7890            190  FNEG      equ FIX+3
7893            191  FADDIN    equ FNEG+3
7896            192  FSUBIN    equ FADDIN+3
7899            193  FTIMES    equ FSUBIN+3
789C            194  FINTO     equ FTIMES+3
```

Listing continued.

Listing continued.

```
789F                    195  IFF      equ FINTO+3
78A2                    196  MOVFP    equ IFF+3
0800                    197  ;
0800                    198  ;
0800                    199  ;
0800                    200  ;
0800                    201  ;
0800                    202  **********************************
0800                    203  **********************************
0800                    204  ;
0800                    205  ; Sample program #8 begins here.
0800                    206  ;
0800                    207  ;
0800 20 12 78           208  START    jsr INIT            ;Always do first
0803                    209  ;
0803 20 0C 78           210  MENULOOP jsr HOME
0806 20 36 78           211           jsr PRINT
0809 D3 F0 E5           212           byt "Speed/asm Sample Program #8",CR
080C E5 E4 AF
080F E1 F3 ED
0812 A0 D3 E1
0815 ED F0 BC
0818 E5 A0 D0
081B F2 EF E7
081E F2 E1 ED
0821 A0 A3 B8
0824 8D
0825 8D C5 F8           213           byt CR,"Examples of various"
0828 E1 ED F0
082B BC E5 F3
082E A0 EF E6
0831 A0 F6 E1
0834 F2 E9 EF
0837 F5 F3
0839 A0 ED E5           214           byt " menu programming",CR
083C EE F5 A0            A0 ED E5EE F5 A0 F0 F2 EF E7 F2 E1 ED ED E9 EE E7 8D
083F F0 F2 EF
0842 E7 F2 E1
0845 ED ED E9
0848 8D                  0848:EE E7 8D
084B F3 F4 F9           215           byt "styles.",CR
084E BC E5 F3
0851 AE 8D
0853 8D                 216           byt CR
0854 D3 E5 BC           217           byt "Select one:",CR,CR
0857 E5 E3 F4
085A A0 EF EE
085D E5 BA 8D
0860 8D
0861 A0 A0 C1           218           byt "  A) Straight menu selection",CR
0864 A9 A0 D3
0867 F4 F2 E1
086A E9 E7 E8
086D F4 A0 ED
0870 E5 EE F5
0873 A0 F3 E5
0876 BC E5 E3
0879 F4 E9 EF
087C EE 8D
087E A0 A0 C2           219           byt "  B) Select values in a set",CR
0881 A9 A0 D3
0884 E5 BC E5
0887 E3 F4 A0
088A F6 E1 BC
088D F5 E5 F3
0890 A0 E9 EE
0893 A0 E1 A0
0896 F3 E5 F4
0899 8D
089A A0 A0 C3           220           byt "  C) Checking for values"
089D A9 A0 C3
08A0 E8 E5 E3
08A3 E8 E9 EE
08A6 E7 A0 E6
08A9 EF F2 A0
08AC F6 E1 BC
08AF F5 E5 F3
08B2 A0 EE EF           221           byt " not in a set",CR
08B5 F4 A0 E9
08B8 EE A0 E1
08BB A0 F3 E5
08BE F4 8D
08C0 8D                 222           byt CR
08C1 C3 E8 EF           223           byt "Choice? :",0
08C4 E9 E3 E5
```

Listing continued.

used to determine how the variables compare. The applicable instructions are:

- BEQ—Branch if the accumulator equals the operand of CMP.
- BNE—Branch if the acc does not equal the operand of CMP.
- BLT—Branch if the acc is less than the operand of CMP.
- BGE—Branch if the acc is greater than or equal to the operand of the CMP instruction.

The CSP Instruction

The CSP instruction (Call SPEED/ASM Procedure) is a new pseudo-opcode/6502 instruction added to LISA's repertoire specifically for use by SPEED/ASM programmers. CSP combines LISA's JSR and .DA statements. This instruction may help make writing SPEED/ASM programs much easier.

The syntax for the CSP instruction is:

CSP <adrs> {<.da expressions>}

which is identical to the statements:

JRS <adrs>
.DA <.da expressions>

There are four different types of <.da expressions>: a full address, a string expression, a high order byte value, and a low order byte value.

Any time an address expression appears in the operand field of a .DA statement (or in the <.da expression> portion of the CSP statement) two bytes of object code are generated. If a string appears in the operand field of a .DA or CSP instruction, then a single byte of object code is emitted for each character in the string. For our purposes you should enclose the string with quotation marks. If an address expression is immediately preceded by a pound sign (#), then only one byte of object code is output; its value will be the low order byte of the specified address expression. If an address expression is prefaced with a slash (/), then the high order byte of the address expression's value is output.

With the CSP instruction you can type many SPEED/ASM statements on a single line. Some examples of the CSP statement in operation include:

Listing continued.

```
08C7 BF A0 BA
08CA 00
08CB                    224  ;
08CB                    225  ;
08CB                    226  ; Read a character, if it is lower
08CB                    227  ; case convert it to upper case,
08CB                    228  ; then jump to the appropriate routine.
08CB                    229  ;
08CB 20 03 78           230          jsr GETC
08CE C9 E1              231          cmp #"a"            ;See if lower case
08D0 90 02              232          blt >0
08D2 29 DF              233          and #$DF            ;Converts it to upper
08D4                    234  ;
08D4 20 48 78           235  ^0      csp CASE,#NUMCAS0/3
08D7 03
08D8 C1 0B 09           236  CASES0   .da "A",STDMENU
08DB C2 54 0A           237          .da "B",SETSELCT
08DE C3 5E 0B           238          .da "C",NOTSTEXP
08E1                    239  ;
0009                    240  NUMCAS0  =   *-CASES0
08E1                    241  ;
08E1                    242  ;
08E1                    243  ;
08E1                    244  ; An invalid value was entered at
08E1                    245  ; this point.
08E1                    246  ;
08E1 20 36 78           247          csp PRINT
08E4 8D 8D 87           248          byt CR,CR,BELL,BELL
08E7 87
08E8 C9 EE F6           249          byt "Invalid entry, press return",CR,0
08EB E1 EC E9
08EE E4 A0 E5
08F1 EE F4 F2
08F4 F9 AC A0
08F7 F0 F2 E5
08FA F3 F3 A0
08FD F2 E5 F4
0900 F5 F2 EE
0903 8D 00
0905                    250  ;
0905 20 0F 78           251          jsr READLN
0908 4C 03 08           252          jmp MENULOOP
090B                    253  ;
090B                    254  ;
090B                    255  ;
090B                    256  **********************************
090B                    257  ;
090B                    258  ;
090B                    259  ; A standard menu selection is
090B                    260  ; demonstrated here.  Actually,
090B                    261  ; the main menu is identical to
090B                    262  ; this type of code.
090B                    263  ;
090B 20 0C 78           264  STDMENU  jsr HOME
090E 20 36 78           265          csp PRINT
0911 8D 8D              266          byt CR,CR
0913 D3 F4 E1           267          byt "Standard Menu Selection",CR,CR
0916 EE E4 E1   D3 F4 E1 EE E4 E1 F2 E4 A0 CD E5 EE F5 A0 D3 E5 EC E5 E3 F4 E9 EF EE 8D 8D
0919 F2 E4 A0
091C 8D         091C:CD E5 EE F5 A0 D3 E5 EC E5 E3 F4 E9 EF EE 8D 8D
092C A0 A0 B0           268          byt "  0: print 0",CR
092F BA A0 F0
0932 F2 E9 EE
0935 F4 A0 B0
0938 8D
0939 A0 A0 B1           269          byt "  1: print 1",CR
093C BA A0 F0
093F F2 E9 EE
0942 F4 A0 B1
0945 8D
0946 A0 A0 B2           270          byt "  2: print 2",CR
0949 BA A0 F0
094C F2 E9 EE
094F F4 A0 B2
0952 8D
0953 A0 B5 B5           271          byt " 55: print 55",CR,CR
0956 BA A0 F0
0959 F2 E9 EE
095C F4 A0 B5
095F B5 8D 8D
0962 C3 E8 EF           272          byt "Choice? :",0
0965 E9 E3 E5
0968 BF A0 BA
096B 00
096C                    273  ;
```

Listing continued.

```
CSP PRINT,"Printing strings with CSP",#0
CSP IFI,I,LE,J
CSP IFS0,STRNG,EQ,"STRING COM-
    PARE",#0
CSP FOR0,I,1,100
CSP FOR,I,STRT,END,STEP
CSP PRTINT,I
```

Due to its convenience I will use CSP in many of the examples that follow.

The ONXGOTO Subroutine

The first new SPEED/ASM control routine I will discuss is the ONXGOTO routine. The ONXGOTO routine transfers control to a new statement depending upon the value in the X registers. The syntax for the ONXGOTO instruction is:

```
JSR     ONXGOTO
ADR     <numentries>
ADR     <label0>,<label1>,...,<labeln>
```

where <numentries> is the number of labels that follow the <numentries> value and <label1>..<labeln> are labels within your SPEED/ASM program. If the X register contains zero, control is transferred to <label0>. If the X register contains one, ONXGOTO will jump to location <label1>, etc. If the X register contains a value greater than <numentries>, then program execution continues with the 6502 statement immediately following the <labeln> entry. *Warning*! It is critical that the <numentries> value *exactly* represents the number of addresses that follow. If <numentries> is too small and the ONXGOTO routine falls through, the 6502 will attempt to execute one of the trailing addresses as valid 6502 code. This usually will cause the program to bomb. One way to guarantee that the <numentries> value is always correct is to use a call to ONXGOTO of the form:

```
        JSR   ONXGOTO
        ADR   ENTRY0/2
TABLE0  ADR   ADRS1,ADRS2,...,ADRSn
;
ENTRY0  EQU   *-TABLE0
```

The "*" operator in the operand field says "give me the current program address." By subtracting the address of the jump table's first entry from the address of the first byte after the address table, this equate calculates the number of bytes in the table.

Listing continued.

```
096C 20 42 78   274           csp RDINT,SELCTVAR
096F DC 0B
0971            275  ;
0971 20 4B 78   276           csp CASEI,NUMCS1/4,SELCTVAR
0974 04 00 DC
0977 0B
0978 00 00 AE   277  CASES1   adr 0,ZERO
097B 09
097C 01 00 C7   278           adr 1,ONE
097F 09
0980 02 00 DD   279           adr 2,TWO
0983 09
0984 37 00 F3   280           adr 55,FIFTYFIV
0987 09
0988            281  ;
0010            282  NUMCS1   =   *-CASES1
0988            283  ;
0988            284  ;
0988            285  ; If the program gets to this point
0988            286  ; then they've entered an invalid
0988            287  ; value.
0988            288  ;
0988 20 36 78   289           csp PRINT
098B 87 87 8D   290           byt BELL,BELL,CR
098E C9 EE F6   291           byt "Invalid entry, press return",CR,0
0991 E1 EC E9
0994 E4 A0 E5
0997 EE F4 F2
099A F9 AC A0
099D F0 F2 E5
09A0 F3 F3 A0
09A3 F2 E5 F4
09A6 F5 F2 EE
09A9 8D 00
09AB 4C 0B 09   292           jmp STDMENU
09AE            294  ;
09AE            295  ********
09AE            296  ;
09AE            297  ; Control is transferred here if
09AE            298  ; the user pressed 0.
09AE            299  ;
09AE 20 36 78   300  ZERO      csp PRINT,#CR,"You pressed zero",#CR,#0
09B1 8D D9 EF
09B4 F5 A0 F0
09B7 F2 E5 F3
09BA F3 E5 E4
09BD A0 FA E5
09C0 F2 EF 8D
09C3 00
09C4 4C 0F 0A   301           jmp QUITSTD
09C7            302  ;
09C7            303  ;
09C7            304  ********
09C7            305  ;
09C7            306  ; Control is transferred here if
09C7            307  ; the user pressed 1.
09C7            308  ;
09C7 20 36 78   309  ONE       csp PRINT,#CR,"You entered 1",#CR,#0
09CA 8D D9 EF
09CD F5 A0 E5
09D0 EE F4 E5
09D3 F2 E5 E4
09D6 A0 B1 8D
09D9 00
09DA 4C 0F 0A   310           jmp QUITSTD
09DD            311  ;
09DD            312  ;
09DD            313  ********
09DD            314  ;
09DD            315  ; Control is transferred here if
09DD            316  ; the user pressed 2.
09DD            317  ;
09DD 20 36 78   318  TWO       csp PRINT,#CR,"You entered 2",#CR,#0
09E0 8D D9 EF                   8D D9 EF F5 A0 E5 EE F4 E5 F2 E5 E4 A0 B2 8D 00
09E3 8D         09E3:F5 A0 E5 EE F4 E5 F2 E5 E4 A0 B2 8D 00
09F0 4C 0F 0A   319           jmp QUITSTD
09F3            320  ;
09F3            321  ;
09F3            322  ********
09F3            323  ;
09F3            324  ; Control is transferred here if
09F3            325  ; the user pressed 55.
09F3            326  ;
09F3 20 36 78   327  FIFTYFIV csp PRINT,#CR
09F6 8D
```

Listing continued.

Since we're interested in the number of entries, not the number of bytes in the table, you must divide the ENTRY0 label by two (since there are two bytes per table entry) to compute the proper value.

Using this method for specifying the number of entries in the ONX-GOTO routine lets you modify the number of entries in the address table and automatically update the <numentries> value. If you don't use this method, adding or deleting an entry from the address table forces you to increment or decrement the <numentries> to make up for the change. If you don't, disaster may strike the next time you run the program. Since it is so easy to forget to update the <numentries> value when modifying the address table, using the equate to automatically calculate the number of entries in the table is a smart thing to do.

The CASE Statement

SPEED/ASM supports a control structure very similar to the CASE statement found in high level languages like Pascal and "C". Two versions of the SPEED/ASM CASE statement are provided: CASE and CASE1. CASE is a single byte CASE statement; it compares the value in the 6502 accumulator against several values and branches if the accumulator equals one of those values. CASE1 compares the contents of a SPEED/ASM integer variable to one of several integer values and branches if a match is made.

The format for the CASE statement is:

```
CSP   CASE,#<numentries>
.DA   #<value1>,<adrs1>
.DA   #<value2>,<adrs2>
.DA   #<value3>,<adrs3>
  .       .       .
  .       .       .
  .       .       .
.DA   #<valuen>,<adrsn>
```

where <numentries> is the number of cases present, <valuei> (i = 0..n), are the single byte values you want to compare the 6502 accumulator against, and <adrsi> (i = 0..n) are labels where SPEED/ASM will jump to

Listing continued.

```
09F7 D9 EF F5   328          byt "You entered fifty-five",CR,0
09FA A0 E5 EE
09FD F4 E5 F2
0A00 E5 E4 A0
0A03 E6 E9 E6
0A06 F4 F9 AD
0A09 E6 E9 F6
0A0C E5 8D 00
0A0F            329  ;
0A0F            330  ;
0A0F 20 36 78   331  QUITSTD  csp PRINT,#CR
0A12 8D
0A13 D0 F2 E5   332          byt "Press return to continue:"
0A16 F3 F3 A0
0A19 F2 E5 F4
0A1C F5 F2 EE
0A1F A0 F4 EF
0A22 A0 E3 EF
0A25 EE F4 E9
0A28 EE F5 E5
0A2B BA
0A2C 00         333          byt 0
0A2D 20 0F 78   334          jsr READLN
0A30 4C 03 08   335          jmp MENULOOP
0A33            336  ;
0A33            337  ;
0A33            338  ;
0A33            339  ;
0A33            340  ;
0A33            341  **********************************
0A33            342  ;
0A33            343  ; Demonstrate set selection with
0A33            344  ; the INSET routine.
0A33            345  ;
0A33            346  ;
0A33 20 36 78   347  SETLOOP  jsr PRINT
0A36 8D D0 F2   348          byt CR,"Press return to continue:",0
```

Listing continued.

if a match is found.

During execution the CASE statement compares the value in the 6502 accumulator to <value0>. If the accumulator is equal to <value0>, control is transferred to location <adrs0>. If the accumulator does not equal <value0>, the accumulator is compared against <value1> and control is transferred to location <adrs1> if a match is made. If the accumulator doesn't equal <value1> it's compared to <value2>, etc. If the accumulator isn't equal to any of the values present in the CASE statement, control is resumed at the first statement after the <valuen> entry.

SPEED/ASM will bomb horribly if <numentries> doesn't properly reflect the number of cases in the CASE statement. The safest way to specify this value is to have LISA v2.6 calculate it for you. This can be accomplished using code of the form:

Listing continued.

```
0A39 E5 F3 F3      8D D0 F2 E5 F3 F3 A0 F2 E5 F4 F5 F2 EE A0
0A3C A0 F2 E5       F4 EF A0 E3 EF EE F4 E9 EE F5 E5 BA 00
0A3F F4 F5 F2
0A42 EE A0 F4
0A45 EF A0 E3
0A48 8D          0A48:EF EE F4 E9 EE F5 E5 BA 00
0A51 20 0F 78   349          jsr READLN
0A54            350  ;
0A54 20 0C 78   351  SETSELCT jsr HOME
0A57 20 36 78   352          jsr PRINT
0A5A C5 EE F4   353          byt "Enter an alphabetic or numeric",CR
0A5D E5 F2 A0
0A60 E1 EE A0
0A63 E1 EC F0
0A66 E8 E1 E2
0A69 E5 F4 E9
0A6C E3 A0 EF
0A6F F2 A0 EE
0A72 F5 ED E5
0A75 F2 E9 E3
0A78 8D
0A79 E3 E8 E1   354          byt "character (return quits):",0
0A7C F2 E1 E3
0A7F F4 E5 F2
0A82 A0 A8 F2
0A85 E5 F4 F5
0A88 F2 EE A0
0A8B F1 F5 E9
0A8E F4 F3 A9
0A91 BA 00
0A93            355  ;
0A93 20 03 78   356          jsr GETC
0A96 C9 8D      357          cmp #CR
0A98 D0 03      358          bne >1
0A9A 4C 03 08   359          jmp MENULOOP
0A9D            360  ;
0A9D            361  ; Convert lower case to upper case
0A9D            362  ;
```

Listing continued.

```
      CSP    CASE,#NUMCASES/3
CASETBL .DA    <value0>,<adrs0>
        .DA    <value1>,<adrs1>
         .       .       .
         .       .       .
         .       .       .
        .DA    <valuen>,<adrsn>
;
NUMCASES = *-CASETBL
```

This code automatically computes the number of cases present in the case list. Furthermore, you don't have to change anything if you add or delete cases later on.

The CASEI statement is similar to the CASE statement; the only difference is the CASEI routine lets you compare a SPEED/ASM integer variable to a series of integer values (CASE only performs byte comparisons). The syntax for the CASEI statement is:

```
CSP    CASE,<numentries>,<SAvariable>
ADR    <value0>,<adrs0>
ADR    <value1>,<adrs1>
```

```
ADR    <value2>,<adrs2>
  .        .        .
  .        .        .
  .        .        .
ADR    <valuen>,<adrsn>
```

The variable <numentries> is the number of cases (a two-byte value for CASEI and a single byte value for CASE); <SAvariable> is the name of a SPEED/ASM integer variable; <valuei> (i=0..n) are 16-bit integer values, and <adrsi> (i=0..n) are the names of statement labels in your SPEED/ASM program where a branch will be made to if <SAvariable> equals <valuei>.

Like the CASE and ONXGOTO statements, <numentries> must accurately describe the number of entries in the case table or SPEED/ASM may hang. To make sure you enter the proper value you should let LISA v2.6 compute the number of entries for you using code of the form:

```
        CSP   CASEI,NUMCASES/4,VAR
CASES   ADR   <value0>,<adrs0>
        ADR   <value1>,<adrs1>
        ADR   <value2>,<adrs2>
          .       .       .
          .       .       .
          .       .       .
        ADR   <valuen>,<adrsn>
;
NUMCASES = *-CASES
```

NUMCASES must be divided by four, since there are four bytes in each case entry.

The INSET and NOTINSET Routines

The INSET and NOTINSET routines compare the accumulator against a set of values and branch to a single location if the accumulator is in the specified set (INSET), or is not in the specified set (NOTINSET). The syntax for these two routines is identical:

```
CSP    INSET,<numentries>
BYT    <value0>,<value1>,...,<valuen>
ADR    <adrs>

or

CSP    NOTINSET,<numentries>
BYT    <value0>,<value1>,...,<valuen>
ADR    <adrs>
```

Listing continued.

```
0A9D C9 E1      363  ^1      cmp #"a"
0A9F 90 02      364          blt >0
0AA1 29 DF      365          and #$DF
0AA3            366  ;
0AA3            367  ^0:
0AA3            368  ;
0AA3            369  ;
0AA3            370  ;
0AA3            371  ;
0AA3            372  ; Note the sneaky way of specifying
0AA3            373  ; the number of elements in the sets.
0AA3            374  ; LISA's STR pseudo-opcode emits the
0AA3            375  ; length of the string that follows.
0AA3            376  ; This just happens to be the number
0AA3            377  ; of characters in the set.  This only
0AA3            378  ; works if the set consists entirely
0AA3            379  ; of printable characters.
0AA3            380  ;
0AA3 20 4E 78   381          jsr INSET
0AA6 1A C1 C2   382          str "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
0AA9 C3 C4 C5
0AAC C6 C7 C8
0AAF F2
0AC1 06 0B      383          adr ISALPHA
0AC3            384  ;
0AC3 20 4E 78   385          jsr INSET
0AC6 0A B0 B1   386          str "0123456789"
0AC9 B2 B3 B4
0ACC B5 B6 B7
0ACF B8 B9
0AD1 34 0B      387          adr ISNUMRIC
0AD3            388  ;
0AD3            389  ;
0AD3            390  ; Non-alphanumeric character at this point.
0AD3            391  ;
0AD3 20 36 78   392          csp PRINT,#BELL,#CR
0AD6 87 8D
0AD8 D9 EF F5   393          byt "You pressed a non-alphanumeric"
0ADB A0 F0 F2
0ADE E5 F3 F3
0AE1 E5 E4 A0
0AE4 E1 A0 EE
0AE7 EF EE AD
0AEA E1 EC F0
0AED E8 E1 EE
0AF0 F5 ED E5
0AF3 F2 E9 E3
0AF6 A0 E3 E8   394          byt " character"
0AF9 E1 F2 E1
0AFC E3 F4 E5
0AFF F2
0B00 8D 8D 00   395          byt CR,CR,0
0B03 4C 33 0A   396          jmp SETLOOP
0B06            397  ;
0B06            398  ;
0B06 20 36 78   399  ISALPHA csp PRINT,#CR,#CR
0B09 8D 8D
0B0B D9 EF F5   400          byt "You entered an alphabetic character"
0B0E A0 E5 EE
0B11 F4 E5 F2
0B14 E5 E4 A0
0B17 E1 EE A0
0B1A E1 EC F0
0B1D E8 E1 E2
0B20 E5 F4 E9
0B23 E3 A0 E3
0B26 E8 E1 F2
0B29 E1 E3 F4
0B2C E5 F2
0B2E 8D 8D 00   401          byt CR,CR,0
0B31 4C 33 0A   402          jmp SETLOOP
0B34            403  ;
0B34 20 36 78   404  ISNUMRIC csp PRINT,#CR,#CR
0B37 E8
0B39 D9 EF F5   405          byt "You entered a numeric character"
0B3C A0 E5 EE
0B3F F4 E5 F2
0B42 E5 E4 A0
```

Listing continued.

Listing continued.

```
0B45 E1 A0 EE
0B48 F5 ED E5
0B4B F2 E9 E3
0B4E A0 E3 E8
0B51 E1 F2 E1
0B54 E3 F4 E5
0B57 F2
0B58 8D 8D 00   406          byt CR,CR,0
0B5B 4C 33 0A   407          jmp SETLOOP
0B5E            408  ;
0B5E            409  ;
0B5E            410  ;
0B5E            411  ;
0B5E            412  ;
0B5E            413  ;
0B5E            414  ;
0B5E            415  ***********************************
0B5E            416  ;
0B5E            417  ;
0B5E            418  ; Demonstrate the NOTINSET routine
0B5E            419  ; here.
0B5E            420  ;
0B5E 20 0C 78   421  NOTSTEXP jsr HOME
0B61 20 36 78   422  NOTSTLP  csp PRINT
0B64 C2 F9 A0   423          byt "By typing a non-alphanumeric",CR
0B67 F4 F9 F0
0B6A E9 EE E7
0B6D A0 E1 A0
0B70 EE EF EE
0B73 AD E1 EC
0B76 F0 E8 E1
0B79 EE F5 ED
0B7C E5 F2 E9
0B7F E3 8D
0B81 E3 E8 E1   424          byt "character you can exit this routine"
0B84 F2 E1 E3
0B87 F4 E5 F2
0B8A A0 F9 EF
0B8D F5 A0 E3
0B90 E1 EE A0
0B93 E5 F8 E9
0B96 F4 A0 F4
0B99 E8 E9 F3
0B9C A0 F2 EF
0B9F F5 F4 E9
0BA2 EE E5
0BA4 8D 00      425          byt CR,0
0BA6 20 03 78   426          jsr GETC
0BA9 C9 E1      427          cmp #"a"
0BAB 90 02      428          blt >0
0BAD 29 DF      429          and #$DF
0BAF            430  ;
0BAF 20 51 78   431  ^0       csp NOTINSET
0BB2 24 B0 B1   432          str "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
0BB5 B2 B3 B4
0BB8 B5 B6 B7
0BBB B8 B9 C1
0BBE C2 C3 C4
0BC1 C5 C6 C7
0BC4 C8 C9 CA
0BC7 CB CC CD
0BCA CE CF D0
0BCD D1 D2 D3
0BD0 D4 D5 D6
0BD3 D7 D8 D9
0BD6 DA
0BD7 03 08      433          adr MENULOOP
0BD9 4C 61 0B   434          jmp NOTSTLP
0BDC            435  ;
0BDC            436  ;
0BDC            437  ;
0BDC            438  ***********************************
0BDC            439  ;
0BDC            440  ; Variable declaration
0BDC            441  ;
0BDC 00 00      442  SELCTVAR adr 0
0BDE            443          end
***** END OF ASSEMBLY
```

In the case of INSET the 6502 accumulator is compared to the values <value0>..<valuen>. If the accumulator is equal to any value in this list, control is transferred to location <adrs>. If the accumulator doesn't equal any of the values in the set, program execution continues with the first statement after the INSET statement.

The NOTINSET routine is used to ensure that the accumulator doesn't contain a value in a given set. Control is transferred to the branch address if and only if the value in the accumulator does not match any of the values in the set. If the accumulator matches one of the values in the set that follows the call to NOTINSET, control is transferred to the first statement after the NOTINSET jump address.

Conclusions

The program control transfer routines provided in the SPEED/ASM package are very powerful. This month's demonstration program shows how these control structures can be used to set up some very flexible menu programs.

"The program control transfer routines provided in the SPEED/ASM package are very powerful. Along with the power, however, comes responsibility."

Along with the power, however, comes responsibility. It is very important that you make sure all <numentries> values properly reflect the number of entries in the table following the SPEED/ASM routine call. SPEED/ASM uses this information to determine how many cases to check, where the first instruction following the case table can be found, etc. Failure to provide proper data in this parameter slot probably will cause your program to hang. ■