Relocatable Jumps#

General Information

Author: Peter Meyer
Assembler: generic
Published: APPLE Assembly Line 12/82
Download: Apple Assembly Line Archive

A machine language routine is said to be relocatable if it can function properly regardless of its absolute location in memory. If a routine contains a JMP or a JSR to an INTERNAL address then it is not relocatable; if it is run in another part of memory then the internal JSR or JMP will still reference the former region of memory. JMPs and JSRs to subroutines at absolute locations (e.g. in the Monitor) do not impair the relocatability of a routine.

I will explain here a technique whereby you can, in effect, perform internal JSRs and JMPs without impairing the relocatability of your routine. There is a small price to pay for this: namely, an increase in the length of your routine. Your routine must be preceded by a 2-part Header Routine which is 43 bytes long. In addition, each internal JSR requires 8 bytes of code, and each internal JMP requires 11 bytes of code.

No tables or other data storage are required, except that three bytes must be reserved for a JMP instruction. These three bytes can be anywhere in memory, but must be at an absolute location. There are three bytes that normally are used only by Applesoft, namely, the ampersand JMP vector at $3F5 to $3F7. Since we are here concerned only with machine language routines in their own right, we can use the locations $3F5 to $3F7 for our own purposes. However, other locations would do just as well.

The technique is fully illustrated in the accompanying assembly language program. This routine consists of three parts:

1. Header Part 1 (SETUP), which sets up a JMP instruction at VECTOR (at $3F5-$3F7, but could be different, as explained above) to point to Header Part 2.
2. Header Part 2 (HANDLER), which is a 15-byte section of code whose task is to handle requests to perform internal JSRs and JMPs (more on this below).
3. The main part of the routine, in which internal JSRs and JMPs (in effect) are performed using macro instructions.

When your routine (including the Header) is executed, the first thing that happens is that Header Part 1 locates itself (using the well-known JSR $FF58 technique), then places a JMP HANDLER at VECTOR. Thereafter a JMP VECTOR is equivalent to JMP HANDLER, and a JSR VECTOR is equivalent to a JSR HANDLER. The HANDLER routine handles requests from your routine for internal JSRs and JMPs. To perform a JSR to an internal subroutine labelled SUBR simply include the following code:

```
HERE    LDA #SUBR-HERE-7 low byte of offset
     LDY /SUBR-HERE-7 high byte of offset
     TSX
     JSR VECTOR
```

As explained above, the JSR VECTOR is in effect a JSR HANDLER. The Header Part 2 code takes the values in the A and Y registers and adds them to an address which it obtains from the stack to obtain the address of SUBR. It then places this address in INDEX ($5E,5F) and executes "JMP (INDEX)".

An internal JMP, from one part of your routine to another, is performed in a similar manner. Suppose you wish to JMP from HERE to THERE. It is done as follows:

```
HERE    LDA #THERE-HERE-7 low byte of offset
     LDY /THERE-HERE-7 high byte of offset
     TSX
     JSR $FF58
     JMP VECTOR
```

Since we are (in effect) performing a JMP, rather than a JSR, we do a JMP VECTOR rather than a JSR VECTOR. The other difference is that we have a JSR $FF58 following the TSX.

Clearly the sequence of instructions which allows you to perform a JMP or a JSR could be coded as a macro. The macros to use are shown in the accompanying program listing. By using macros an internal JMP or JSR can be performed with a single macro instruction bearing a very close resemblance to a real JSR or JMP instruction.

The following program, which consists of the Header Routine plus a demonstration routine, can be assembled to disk using the .TF directive. It can then be BRUN at any address and it will function properly. Thus it is relocatable, despite the fact that there are (in effect) an internal JMP and two internal JSRs performed.

When performing an internal JSR or JMP using my techniques, it is not possible to pass values in the registers, since these are required to pass information to the HANDLER routine. Nor is it advisable to try to pass parameters on the stack, even though the HANDLER routine does not change the value of the stack pointer. Better is to deposit values in memory and retrieve them after the transition has been made.

The HANDLER routine passes control to the requested part of your routine using a JMP indirect. (INDEX at $5E,5F, has been used in the accompanying program, but any other address would do as well, provided that it does not cross a page boundary.) This means that the section of your routine to which control is passed (whether or not it is a subroutine) may find its own location by inspecting the contents of the location used for the JMP indirect. This feature of this technique is also illustrated in the accompanying program, in the PRINT.MESSAGE subroutine.

The use of internal data blocks is something not normally possible in a relocatable routine, but it can be done if the techniques shown here are used.

This method of performing internal JSRs and JMPs in a relocatable routine may be simplified if the routine is intended to function as a subroutine appended to an Applesoft program. If the subroutine is appended using my utility the Routine Machine (available from Southwestern Data Systems), then it is not necessary to include the 47-byte Header Routine. Internal JMPs and JSRs can still be performed exactly as described above, except that the address of VECTOR must be $3F5-$3F7. This technique is not described in the documentation to the Routine Machine. A full explanation may be found in the Appendix to the documentation accompanying Ampersoft Program Library, Volume 4 (also available from Southwestern Data Systems).

```
 1010   .TF B.MEYER.1
 1020  *SAVE S.MEYER.1
 1030  *-------------------------------
 1040  *    SETUP AND HANDLER ROUTINES
 1050  *    TO ALLOW INTERNAL JSRS AND
 1060  *    JMPS IN A RELOCATABLE MACHINE
 1070  *    LANGUAGE ROUTINE
 1080
 1090  *    BY PETER MEYER, 11/3/82
 1100  *-------------------------------
```

```
1110  *    LOCATIONS
1120
1130  INDEX        .EQ $5E,5F
1140  STACK        .EQ $100 - $1FF
1150  VECTOR     .EQ $3F5 - $3F7
1160  *-------------------------------
1170  *    MACRO DEFINITIONS
1180
1190           .MA JSR
1200  :1     LDA #]1-:1-7
1210           LDY /]1-:1-7
1220           TSX
1230           JSR VECTOR
1240           .EM
1250
1260           .MA JMP
1270  :1     LDA #]1-:1-7
1280           LDY /]1-:1-7
1290           TSX
1300           JSR $FF58
1310           JMP VECTOR
1320           .EM
1330  *-------------------------------
1340  *    HEADER PART 1
1350
1360  SETUP  JSR $FF58  FIND OURSELVES
1370           TSX
1380           CLC
1390           LDA #HANDLER-SETUP-2
1400           .DA #$7D,STACK-1    FORCE ABS,X MODE
1410           STA VECTOR+1
1420
1430           LDA /HANDLER-SETUP-2
1440           ADC STACK,X
1450           STA VECTOR+2
1460
1470           LDA #$4C      "JMP"
1480           STA VECTOR
1490           BNE MAIN.ROUTINE    ALWAYS
1500  *-------------------------------
1510  *    HEADER PART 2
1520
1530  HANDLER
1540
1550  *    ON ENTRY A,Y HOLD OFFSET
1560  *    FOR JMP OR JSR FROM ROUTINE
1570  *    X IS STACK POINTER FROM BEFORE LAST JSR
1580
1590           CLC
1600           .DA #$7D,STACK-1    FORCE ABS,X MODE
1610           STA INDEX
1620           TYA
1630           ADC STACK,X
1640           STA INDEX+1
1650           JMP (INDEX)
1660  *-------------------------------
1670  *        MAIN ROUTINE, FOR EXAMPLE
1680  *-------------------------------
1690  MSG   .EQ $06 AND $07
```

```
1700  CH     .EQ $24
1710  CV     .EQ $25
1720  INVFLG .EQ $32
1730  COUNT  .EQ $3C
1740  SETTXT .EQ $FB39
1750  VTABZ  .EQ $FC24
1760  HOME .EQ $FC58
1770  COUT .EQ $FDED
1780  *-------------------------------
1790  MAIN.ROUTINE
1800           JSR SETTXT
1810           JSR HOME
1820  MAIN.LOOP
1830           LDA #190
1840           STA COUNT
1850  .1     LDA #AALQT-PRINT.MESSAGE
1860           STA MSG
1870           LDA /AALQT-PRINT.MESSAGE
1880           STA MSG+1
1890           >JSR PRINT.MESSAGE
1900           DEC COUNT
1910           BNE .1
1920           LDA #LONGQT-PRINT.MESSAGE
1930           STA MSG
1940           LDA /LONGQT-PRINT.MESSAGE
1950           STA MSG+1
1960           >JSR PRINT.MESSAGE
1970           >JMP FORWRD
1980
1990  *-------------------------------
2000  PRINT.MESSAGE
2010           CLC
2020           LDA MSG     CHANGE RELATIVE ADDRESS TO
2030           ADC INDEX    AN ABSOLUTE ADDRESS, BY
2040           STA MSG     ADDING THE OFFSET
2050           LDA MSG+1
2060           ADC INDEX+1
2070           STA MSG+1
2080           LDY #0       POINT AT FIRST CHAR OF MSG
2090  .1     LDA (MSG),Y  GET NEXT CHAR OF MSG
2100           BMI .2       IT IS LAST CHAR
2110           ORA #$80      MAKE APPLE VIDEO FORM
2120           JSR COUT      PRINT IT
2130           INY          ADVANCE POINTER
2140           BNE .1       ...ALWAYS
2150  .2     JMP COUT    PRINT AND RETURN
2160  *-------------------------------
2170  *    256 BYTES TO JUMP OVER, JUST FOR ILLUSTRATION
2180
2190           .BS $100
2200  *-------------------------------
2210  *    TOGGLE INVERSE FLAG, AND HOME CURSOR
2220
2230  FORWRD LDA INVFLG
2240           EOR #$C0
2250           STA INVFLG
2260           LDA #0
2270           STA CH
2280           STA CV
```

```
2290           JSR VTABZ
2300           >JMP MAIN.LOOP
2310  *-------------------------------
2320  AALQT   .AT /AAL /
2330  LONGQT .HS 0D0D
2340           .AS / A P P L E A S S E M B L Y L I N E /
2350           .HS 0D02
2360           .AT /  S - C    S O F T W A R E C O R P .  /
```