6502-6809 TRANSLATOR

by Edgar M. (Bud) Pass, Ph.D.

Computer Systems Consultants, Inc.
1454 Latta Lane, Conyers, GA 30207
Telephone Number 404-483-1717/4570

Copyright Notice

Limited Warranty Statement

Computer Systems Consultants, Inc., and its agents, makes no express or implied warranties concerning the applicability of 6502-6809 TRANSLATOR to a particular purpose. Liability is limited to the original license cost. This warranty is presented expressly in lieu of all other warranties, expressed or implied, including those of merchantability and fitness for use and of all other obligations on the part of Computer Systems Consultants, Inc. and its agents.

Problems and Improvements

Users are encouraged to submit problems and to suggest or to provide improvements for 6502-6809 TRANSLATOR. Such input will be processed on a best effort basis. Computer Systems Consultants reserves the right to make program corrections or improvements on an individual or wholesale basis, as required. The company is under no obligation to provide corrections or improvements to all users of 6502-6809 TRANSLATOR. In the case of specific situations requiring extensions to 6502-6809 TRANSLATOR or major assistance in its use, consulting is available on a pre-arranged, for-fee basis.

Computer-Assisted Translation of Programs From the MOS 6502 to the Motorola 6809

Edgar M. Pass, Ph.D.
Computer Systems Consultants, Inc.
1454 Latta Lane
Conyers, Georgia, 30207
Telephone 404-483-1717/4570

GENERAL

The CSC 6502 Translator provides a facility intended to assist the user in converting 6502 source programs into 6809 source programs. This manual first explains the theory of this translation and then describes the application of the theory in terms of the CSC 6502 Translator program. This program should be useful to those users needing to convert programs from a 6502-based system to a 6809-based system. The user is responsible for the media conversion required to make the 6502 program available to the translator program in FLEX, UNIFLEX, or OS/9 format (t.m. Technical Systems Consultants and Microware).

INITIAL COMPARISON

From a review of the Motorola 6800 and 6809 and MOS 6502, the instruction sets of the 6809 and 6502 are both seen to be derivatives of the (older) 6800 instruction set. However, the extensions and changes made in the 6809 and 6502 instruction sets have been in quite different directions. The following table presents the programming models for each of the processors, to indicate the flavor of some of the changes and extensions.

Processor	Register	Bits	Description
6800	A	8	Accumulator
	B	8	Accumulator
	CC	8	Condition Code Register (11HINZVC)
	PC	16	Program Counter
	S	16	Stack Pointer
	X	16	Index Register
6809	A	8	Accumulator
	B	8	Accumulator
	CC	8	Condition Code Register (EFHINZVC)
	D	16	A&B Registers (Concatenated)
	DP	8	Direct Page Register
	PC	16	Program Counter
	S	16	Stack Pointer
	U	16	User Stack Pointer
	X	16	Index Register
	Y	16	Index Register

6502			
	A	8	Accumulator
	CC	8	Condition Code Register (NV0BDIZC)
	PC	16	Program Counter
	S	8	Stack Pointer (First 8 bits=01)
	X	8	Index Register
	Y	8	Index Register

where Condition Code Register bits are defined as follows:

B	BRK command (6502)
C	carry/borrow
D	decimal mode (6502)
E	entire state on stack (6809)
F	fast interrupt (6809)
H	half carry (6800/6809)
I	interrupt mask
N	negative
V	overflow
Z	zero

REGISTER COMPARISON

The similarities and differences in the register structures of the processors are apparent from the table above. Of the three, the 6809 has the most versatile register structure with its two 8-bit accumulators, 8-bit direct page register, two 16-bit index registers, and two 16-bit stack pointers. The 6502 has a less-versatile register structure than either of the other two processors, its only highlight being a second 8-bit index register. The relative speed of the processors or relative compactness of the code is not an issue here.

In corresponding the register structures from the 6502 to the 6809, most registers map to the similarly-named register. The exception is the 6502 A register which corresponds more closely to the 6809 B register than the A register because of the manner in which the 6809 TFR and EXG instructions function.

The condition code registers of the three processors all differ in format and content, with the 6800 and 6809 being the most similar and the 6502 the most unlike. All three condition code registers contain carry/borrow, interrupt mask, negative, overflow, and zero bits, although the interpretation and setting of bits may vary considerably among the three.

The 6502 "V" flag is modified by far fewer instructions than the "V" flags on the 6800 and 6809 processors. The 6802 "B" flag allows an interrupt processing routine to determine the difference between an external interrupt and an internal interrupt generated by a BRK command. The 6502 "D" flag determines whether the ADC and SBC commands will operate in decimal or binary mode. There are no directly corresponding flags for "B" and "D" on the 6800 or 6809 processors. The (nearly) equivalent functions are performed in quite different manners.

ADDRESSING MODE COMPARISON

The addressing modes supported by each of the processors are generally similar, although there are a few significant differences. The following table presents the addressing modes of interest in each of the processors (6800, 6502, 6809).

Mode	Description
Inherent (Accumulator, Implied)	Changes registers or processor states without explicit regard for memory addressing
Direct (Zero-Page)	Prefixes 8-bit address in instruction with 8-bit 00 (DP on 6809) to provide 16-bit effective address
Extended (Absolute)	Uses 16-bit address in instruction directly as effective address
Immediate	Uses 8-bit or 16-bit value in instruction directly, and not as a memory address
Relative	Adds 8-bit offset in instruction to address of next sequential instruction to provide effective address of next instruction to be executed
Indexed (6800)	Adds 8-bit offset in instruction to value in X register to provide 16-bit effective address
Indexed (6809)	Uses one or more post-byte values in instruction to indicate an entire range of register and direct, indirect or non-indirect addressing schemes
Zero Page Indexed (6502)	Adds 8-bit offset in instruction to value in X or Y register to compute 8-bit value; prefixes this value with 8-bit 00 to provide 16-bit effective address
Absolute Indexed (6502)	Adds 16-bit offset in instruction to value in X or Y register to provide a 16-bit effective address
Indirect (6502)	Uses the 16-bit address in instruction to provide a 16-bit effective address; uses the contents of the locations at that

address and at the next address to provide a 16-bit memory address

Indexed Indirect (6502)
Adds the 8-bit offset in instruction to value in X or Y register to provide an 8-bit value, which is prefixed by an 8-bit ØØ to form a 16-bit effective address; the locations at that address and at the next address to provide a 16-bit effective address

Indirect Indexed (6502)
Prefixes 8-bit address in instruction with 8-bit ØØ to provide an 16-bit effective address; adds the contents of the locations at that address and at the next address to the contents of the Y or X register to provide a 16-bit effective address

One significant difference between the 65Ø2 and the other two processors lies in the storage format of 16-bit address. Whereas the Motorola processors store 16-bit addresses as high-order bits then low-order 8-bits in successive locations, the 65Ø2 stores 16-bit addresses as low-order 8-bits then high-order 8-bits in successive locations. This difference appears in the format of instructions containing 16-bit addresses and offsets, return addresses in the stack, 16-bit indirect addresses, interrupt vectors, jump tables, etc.

There are several differences in the use of the S registers on the 65Ø2, 68ØØ, and 68Ø9. The most obvious is that the 68ØØ and 68Ø9 use a 16-bit S register, whereas the 65Ø2 uses an 8-bit S register and prefixes these 8-bits with an 8-bit constant Ø1 to form a 16-bit address. Thus the 65Ø2 stack is restricted to addresses $Ø1ØØ-$Ø1FF. The 68ØØ and 65Ø2 decrement the stack pointer after placing a new item into it, whereas the 68Ø9 decrements it before. Thus the 68ØØ and 65Ø2 stack pointers always point to one address below the current stack limit, whereas the 68Ø9 stack pointer always points to the last item placed onto the stack (if any). The TSX and TXS instructions on the 68ØØ (but not on the 65Ø2) take this into account by adding one to the X register after transferring the contents of the S register to it and by subtracting one from the S register after transferring the X register to it.

This difference can cause difficulty in translating programs from the 68ØØ to the 68Ø9 but, because of the highly restricted nature of the 65Ø2 S register, should cause little difficulty in translating programs from the 65Ø2 to the 68Ø9. The main problem stems from the 68ØØ trick of using the stack pointer as a second index register. However, the 65Ø2 Y register functions as a second index register in many addressing modes, and the 65Ø2 S register is restricted to page Ø1 in memory addresses, eliminating it as an effective third index register on the 65Ø2.

The table below summarizes many of the differences and similarities already discussed concerning the 6502, 6800, and 6502, in terms of the 6502 instruction set. This set has 56 members, as opposed to 97 members for the 6800 and 58 members for the 6809. However, counting address mode and register variations, the 6502 can execute approximately 100 instructions, the 6800 can execute approximately 200 instructions, and the 6809 can execute approximately 750. In the table below, an asterisk indicates that the instruction has the indicated address mode, an entry under Condition-Code-Reg Form indicates the conversion of the Condition-Code format, an entry under Stack indicates stack manipulation, and an entry under X/Y indicates X or Y register modification. The entries under 6809 Condition-Code-Reg indicate the results provided by the translation suggested later in this article.

6502 Opcode	Absolute/ Zero-Page	Condition-Code-Reg 6502 NVØBDIZC	6809 EFHINZVC	Form	Stack	Zero Wrap	Indirect Wrap	X/Y
ADC	*	NV....ZC	..H.NZVC			*	*	
AND	*	N.....Z.	....NZ..			*	*	
ASL	*	N.....ZC	....NZ.C			*		
BCC								
BCS								
BEQ								
BIT	*	NV....Z.	....NZV.					
BMI								
BNE								
BPL								
BRK		...1.1..	...1....		-3			
BVC								
BVS								
CLC		.......Ø	.......Ø					
CLD		....Ø...	RESET D					
CLI		.....Ø..	...Ø....					
CLV		.Ø......	......Ø.					
CMP	*	N.....ZC	....NZ.C			*	*	
CPX	*	N.....ZC	....NZ.C					
CPY	*	N.....ZC	....NZ.C					
DEC	*	N.....Z.	....NZ..			*		
DEX		N.....Z.	....NZ..					X
DEY		N.....Z.	....NZ..					Y

Opcode	Absolute/ Zero-Page	Condition-Code-Reg 6502 NVØBDIZC	6809 EFHINZVC	Form	Stack	Zero Wrap	Indirect Wrap	X/Y
EOR	*	N.....Z.	....NZ..			*	*	
INC	*	N.....Z.	....NZ..			*		
INX		N.....Z.	....NZ..					X
INY		N.....Z.	....NZ..					Y
JMP	*							
JSR	*				-2			
LDA	*	N.....Z.	....NZ..			*	*	
LDX	*	N.....Z.	....NZ..			*		X
LDY	*	N.....Z.	....NZ..			*		Y
LSR	*	Ø.....ZC	....ØZ.C			*		
NOP								
ORA	*	N.....Z.	....NZ..			*	*	
PHA					-1			
PHP				TO	-1			
PLA		N.....Z.	....NZ..		+1			
PLP		NVØBDIZC	EFHINZVC	FROM	+1			
ROL	*	N.....ZC	....NZVC			*		
ROR	*	N.....ZC	....NZ.C			*		
RTI		NVØBDIZC	EFHINZVC		+3			
RTS					+2			
SBC	*	NV....ZC	....NZVC			*	*	
SEC		1	1					
SED		....1...	SET D					
SEI		.....1..	...1....					
STA	*					*	*	
STX	*					*		X
STY	*					*		Y
TAX		N.....Z.	....NZ..					X
TAY		N.....Z.	....NZ..					Y
TSX		N.....Z.	....NZ..		S-1			X
TXA		N.....Z.	....NZ..					X
TXS		....	....		X+1			X
TYA		N.....Z.	....NZ..					Y

The additional registers and instructions on the 6809 make possible an almost exact emulation of the 6502. The 6809 code will not generally have the same length as the 6502 code, nor will it require the same amount of time to execute. Because the translation is being done before assembly time, no run-time instruction modification is assumed.

It has been noted that certain features of the two processors are similar but not identical. If the incremental cost of the exact emulation of a 6502 instruction or feature exceeds its incremental utility in a specific program or subroutine, it would be highly desirable to be able to trade-off the exact emulation for a speed and space reduction in the 6809 code. For instance, the format and contents of the 6502 and 6809 condition code registers are different. Assuming that the "B" and "D" flags of the 6502 are handled separately, many 6502 programs would run correctly with no or minor changes (after translation) on the 6809, even with the 6809 format of condition code register.

The following differences in the instruction sets and features are considered to require a sufficient time and space penalty on the 6809 in exact emulation of the 6502 to be considered for less than complete emulation in the following cases:

-reversed order of absolute address high and low bytes,
-stack restriction to $01XX address range,
-"B", "D", and "V" flag handling in many instructions,
-format of condition code register,
-page-zero wraparound in several addressing modes,
-8-bit X and Y register limitations.

Other major tradeoffs will be discussed in relation to the individual instructions.

In order to reverse the order of high and low address bytes on the 6809 from the 6502, several approaches are possible. the most direct method which still maintains an exact emulation is to assume that all extended address bytes, except within instructions, are reversed and to include 6809 code of the following form to actively flip the address before use:

```
TFR CC,DP          SAVE CC REGISTER
LDU address        LOAD ADDRESS
EXG U,D            MOVE ADDRESS
EXG A,B            REVERSE BYTES
EXG D,U            PUT ADDRESS IN U REGISTER
TFR DP,CC          RESTORE CC REGISTER
```

Executing this code is time-consuming and wasteful if it is not needed. The definition of the 6502 .WORD (or equivalent) assembler pseudo-op code will require defining in such a manner as to reverse the bytes of its address operands. The TFR instructions used above are included to avoid disturbing the condition code register; most such sections of code will require protection of the condition code register.

In many cases, the programmer may decide to use the 6809, rather than 6502, form of extended addressing, and modify the translated

program as necessary to accomplish this. Then the reversal of address bytes as described above will not be required and the 6502 .WORD (or equivalent) assembler pseudo-op code will be translated to the 6809 FDB. The programmer will be required to correspondingly modify references to the bytes in the program representing reversed extended addresses. However, as most other tradeoffs do, this one preserves more of the flavor of the 6809 and less of the 6502 and is hence more efficient.

The 6502 stack restriction to the $01XX address range causes translation problems generally as far-reaching as the reversed address bytes situation. Every operation involving items placed onto the stack or pulled from the stack or the setting of the 'S register must be done thru special inline code. The translator may not directly insert any operation, such as a subroutine call, which uses the stack. The 6502 S register always points to the next available location, whereas the 6809 S register always points to the last item pushed into the stack. Whether 6502 stack emulation is used or not, the translated program must initialize the S register. Interrupt processing may not be supported with the emulated stack. The 6502 instructions which directly place information on the stack are as follows:

BRK, JSR, PHA, PHP;

those which directly gather information from the stack are as follows:

PLA, PLP, RTI, RTS;

and those which directly use or modify the stack pointer directly are as follows:

TSX, TXS.

The inserted 6809 code to emulate the placing of an item onto a 6502 stack is of the following form:

```
STB ,S          STORE B REGISTER IN STACK
TFR CC,DP       SAVE CC REGISTER
TFR D,U         SAVE D REGISTER
TFR S,D
DECB            BUMP S REGISTER DOWN
TFR D,S         SET S REGISTER
TFR U,D         RESTORE D REGISTER
TFR DP,CC       RESTORE CC REGISTER
```

and that of removing of an item from a 6502 stack is of the following form:

```
TFR CC,DP       SAVE CC REGISTER
TFR D,U         SAVE D REGISTER
TFR S,D
INCB            BUMP S REGISTER UP
TFR D,S         SET S REGISTER
TFR U,D         RESTORE D REGISTER
LDB ,S          GET B REGISTER FROM STACK
TFR DP,CC       RESTORE CC REGISTER
```

Instructions (such as BRK, JSR, RTI, and RTS) which require multiple stack operations will require multiple copies of these stack push and pull operations for exact emulation. Even with the pull and push routines, exact 6502 stack emulation must be done with interrupts turned off. The 6502 TXS and TSX instructions will require review if either the S or the X

unless such exact stack emulation is required in a given situation (which it seldom is), most 6502 programs will run after translation using 6809 stack handling with little or no change, and with a great increase in efficiency and functionality for stack-related operations.

The content differences in the condition code registers of the 6502 and 6809 are apparent primarily in the cases of interrupt processing and the ADC, BIT, BRK, CLD, CLV, PHP, PLP, RTI, SBC, and SED instructions.

The 6502 BRK instruction has no exact 6809 counterpart with respect to the "B" flag in the condition code register. However, if 6502 stack emulation and condition code register format are not required, the 6502 BRK instruction may be translated to the 6809 SWI instruction, which has a different vector address in high memory from the IRQ interrupt.

The 6809 has no direct counterpoint to the use of the 6502 "D" flag; however, it is modified only by the CLD, PLP, RTI, and SED instructions and is used only by the ADC and SBC instructions. Thus the 6502 "D" flag is easily emulated using a separate byte, the only difficulties being with the 6809 SBC instruction, which does not interface with the DAA instruction, and properly separating and combining multiple "D" flag bytes during interrupt processing.

The 6809 has many more instructions which modify the "V" flag than does the 6502, in which only the ADC, BIT, CLV, PLP, RTI, and SBC instructions modify the "V" flag. The 6502 "V" flag is thus easily emulated in the same manner as the "D" flag, with the same potential problems during interrupt processing.

Since the 6809 condition code register has format "EFHINZVC' and the 6502 condition code register has format "NV0BDIZC" two routines must be defined for the 6502 emulation, one to reformat condition codes in each direction. The routines are very similar; the following one reformats the 6809 condition code register into 6502 format:

```
     TFR  CC,DP          SAVE CC REGISTER
     TFR  D,U            SAVE D REGISTER
     TFR  CC,A
     CLRB                ZERO 6502 REGISTER
     BITA #$10           I FLAG
     BEQ  *+4
     ORAB #$04
     BITA #$08           N FLAG
     BEQ  *+4
     ORAB #$80
     BITA #$04           Z FLAG
     BEQ  *+4
     ORAB #$02
     TST  SEVFLG         V FLAG
     BEQ  *+4
     ORAB #$40
     BITA #$01           C FLAG
```

```
        BEQ   *+4
        ORAB  #$01
        TST   SEDFLG          D FLAG
        BEQ   *+4
        ORAB  #$80
        TFR   DP,CC           RESTORE CC REGISTER
        TFR   B,DP
        TFR   U,D             RESTORE D REGISTER
        TFR   DP,A            6502 CC IN A REGISTER
```

Again, since most programs never (or seldom) require the particular format of the 6502 condition code register, a programmer may decide to use the 6809-format condition code register and manually change the translated program, as required.

Page zero wraparound is another attribute of the 6502 which is not present on the 6809 and must be handled by the translator thru additional code if exact emulation is required. This problem occurs in the 6502 zero-page-indexed and indexed-indirect address modes. In the zero-page-indexed mode, the 8-bit offset in the 6502 instruction is added to the 8-bit value in the X or Y register to provide an 8-bit value, which is prefixed with 8-bit 00 to provide a 16-bit effective address. The 6809 code inserted by the translator would be of the following form:

```
        TFR   CC,DP               SAVE CC REGISTER
        LEAU ((address) AND $FF),X   COMPUTE ADDRESS
        EXG   U,D
        CLRA                      TRUNCATE TO 8 BITS
        EXG   D,U                 ADDRESS IN U REGISTER
        TFR   DP,CC               RESTORE CC REGISTER
        OPC   ,U                  PERFORM ORIGINAL OPERATION
```

The alternative to emulation would be to treat zero-page-indexed address mode as if it were absolute-indexed address mode, in which case the programmer would be responsible for ensuring that the correct effective address is calculated in each case. In the indexed-indirect mode, the 8-bit offset in the instruction is added to the 8-bit value in the X or Y register to form an 8-bit result, which is prefixed by an 8-bit 00 to form a 16-bit effective address. The contents of the locations at that address and at the next address are used to provide a 16-bit effective address. The 6809 code inserted by the translator would be identical to that provided earlier, with the exception of the last line, which would use indirect addressing and would be of the following form:

```
        OPC   [,U]            PERFORM ORIGINAL OPERATION
```

Assuming that no indirect addresses are placed at $00FF and $0000. An alternative to emulation would be to directly use the 6809 indirect address facility, manually correcting any cases in which the contents of the X or Y register plus the offset exceeds $00FE.

The 6502 8-bit X and Y register limitations affect the following 6502 instructions:

DEX, DEY, INX, INY, LDX, LDY, STX,
STY, TAX, TAY, TSX, TXA, TXS, TYA.

In virtually every case, the 8-bit value being processed must be moved thru the D register in order to properly extend or truncate

the value. For instance, the translator-generated 6809 code for INX would be the following:

```
EXG X,D         MOVE X REGISTER FOR TRUNCATION
LDA #$00        CLEAR MS 8 BITS, NOT C FLAG
INCB            BUMP LAST 8 BITS OF X
EXG D,X         RESTORE NEW X REGISTER
```

The magnitude of the problems associated with the conversion of the translated program to fully use the 16-bit X and Y registers of the 6809 would depend upon the program being translated. However, they may be severe, and the emulation overhead will usually be small.

CONVERSION ANALYSIS

Most computer programs, even on microcomputers, do not run stand-alone but run under control of an operating system or use external I/O, math, or service subroutines. Thus, even if the translation from 6502 to 6809 is exactly correct on an instruction-by-instruction basis, many 6502 programs would not run after translation without modification. The portions of programs requiring change in a practical environment will generally be in the following areas:

- -monitor, operating system, and subroutine library entry points,
- -I/O addresses and hardware,
- -memory-mapped video facilities,
- -miscellaneous tradeoffs made in translation.

Entry points may cause difficulties in terms of addresses, parameters, and functions. The address problems are usually the simplest to solve, since these generally involve merely changing addresses in .EQU statements. The parameter-passing problem encompasses addresses and values passed to and from subroutines, monitor entry points, and operating system routines, and may be far more complex. The number of variations in table and control block format and usage, control value interpretation, data structure representation, method of returning results, etc. is astronomical.

The best plan of attack on these problems varies with the nature of the effort. In the case of a well-defined subroutine library or set of operating system routines being referenced, it may be possible and advantageous to code a set of 6809 routines to interface to a similar-functional library or routines. Then this interface may be used in any program with few other changes in logic required.

I/O address and hardware differences may cause from minor to severe problems in conversion. Simply changing the EQU statements will probably not affect the complete conversion because of the differences in handling of the various I/O devices, such as VIO's, VIA's, PIA's, ACIA's, etc. These differences may be handled by coding interface subroutines, by modifying the code to handle the new I/O device in native mode, by using similar functional routines already available in the

6809 operating system, etc. In the worst case, the 6502 hardware facility may not even be available on the 6809, requiring extensive modifications.

Memory-mapped video facilities are avilable on many of the "appliance" computers as standard features but are not generally directly available on 6809 systems, with the notable exception of the Radio Shack Color Computer. If a 6502 program makes extensive use of memory-mapped video hardware, but the facility is not available on the 6809 or is available but is handled differently, several methods of translating the running 6502 program to become a running 6809 program are possible. The obvious, though sometimes most difficult, means of performing the conversion would be to rewrite the 6502 code after translation to drive the video board or terminal used on the 6809 directly. Another method would be to write a terminal emulation routine which would make the same output appear on an output device on a 6809 as on a video monitor on a 6502. The method used in a given case will depend upon the situation.

The other primary reason for manual intervention in the conversion process involves the tradeoffs made in the translation. The changes required by this have been previously discussed, but may benefit from some of the same organized attacks as suggested for the I/O and hardware problems. Other changes may be desirable to take advantage of the additional instructions and addressing modes of the 6809 versus the 6502.

THEORY SUMMARY

The preceding discussion has presented, in detail, the theory by which 6502 source programs are converted to 6809 source programs by the CSC 6502 Translator. This conversion is performed in two phases.

The first is a low-level (instruction-by-instruction) translation process which could be performed manually or by using a computer program, namely the CSC 6502 Translator. The instruction emulation level may be varied to cause the translated program to have certain attributes closer to the 6502 or to the 6809 architectures, as desired.

The second is a higher-level process which must generally be performed manually (although possibly with the assistance of a editing or special-purpose computer program) since it usually involves creativity and cleverness on a level not yet found in the most advanced computer programs, but sometimes found in human intelligence. This involves the resolution of the remaining differences between the translated 6502 program and the 6809 environment in which the 6809 program will run, and the final debugging and checkout.

As noted above, it is the user's responsibility to convert the 6502 source program into a 6809 FLEX, UNIFLEX or OS/9 compatible format. CSC markets a 6502 disassembler which runs on the 6809 and may be used to assist the user in converting 6502 object programs into 6502 source programs, compatible with the 6502 Translator.

The CSC 6502 Translator is delivered on a diskette containing the following files:

X02XLATS.TXT source file for translator
X02XLATS.CMD binary file for translator
X02MACRO.TXT translator macro library
X02MACLB.TXT assembler macro library
X02SUBLB.TXT assembler subroutine library
X02TEST.TXT 6502 test program

The only customization required for a particular terminal is the specification of the home-up-clear-screen sequence. This is accomplished by modifying the values of the symbols CLRSC1, CLRSC2, CLRSC3 in X02XLATS.TXT and re-assembling the module. Unused locations may be set to hex 00 (for no delay time) or hex 80 (for timing delays).

The translator program functions as a macro generator. The translation process is largely based upon the macro library X02MACRO.TXT, although certain assumptions are made in the translator program pertaining to pseudo-opcodes and address modes. There are currently three option groups (ABC,DEF,HIJ) which partially control the macro generation process. Each macro is composed of two parts.

The first portion provides the macro name and suffix. The macro name itself represents the 6502 opcode name (three characters), starting in the first position. There are currently six suffixes (A,X,Y,ZX,ZY,null), representing instruction address modes accumulator, indexed-indirect, indirect-indexed, zero page plus X, zero page plus Y, and other modes, respectively.

The second portion provides the macro expansion body. The lines are in a fixed format. The first position must be blank. The second thru fourth positions contain zero to three option names. The values of the multiple names, if any, are logically ANDed together. If any of the options are false, the line is not selected. Lines with all blank option fields are always selected. The fifth thru end positions contain the expansion text. Other than in a comment line (starting with an asterisk), an underline character is replaced with the instruction operand; the surrounding parentheses and trailing ",X" or ",Y" strings are dropped for indexed-indirect and indirect-indexed address modes.

The three option groups represent decisions to be made pertaining to condition-code format, ADC/SBC special code, and address format translations. These options were generally described in the theory section. 6809 stack handling is assumed, with the

exception of the TSX and TXS instructions, which will require review. Eight-bit X and Y registers are assumed.

The assembler macro library X02MACLB.TXT contains assembler macros used during the post-translation assembly process. Primarily, they reverse the order of address bytes from lo-hi in the 6502 to hi-lo in the 6809. This could be done more readily at assembly time than at translation time, when required.

The assembler subroutine library X02SUBLB.TXT contains the initialization and special code subroutines generally required for a translated program. If a 6502 program is translated in several portions, the extra library calls for X02SUBLB.TXT should be deleted when the program is re-combined for assembly.

Since the three libraries and translator are provided in editable source format on diskette, they may be readily modified to change or extend the translation process. In a situation as complex as the 6502-6809 translation, errors are certain to occur. The easiest manner in which to correct errors in or to customize the translation process will be to make changes in one or more of the libraries. Changes in the translator should be required more rarely, except to add new options or to adapt to a new 6502 assembler format. Since the assembler format changes may be complex, it may be simpler to modify the 6502 source program with a text editor before translation than to modify the translator. The translator currently assumes MOS assembler format, which uses pseudo-opcodes =, *=, .BYTE, .CHAR, .DBYTE, and .WORD and requires strings in .CHAR statements to be surrounded by single or double quotes.

The test file X02TEST.TXT contains at least one usage of every translation macro and exercises most of the translator program logic, when used with the appropriate options. It may also be used to help test modifications to the translator or libraries.

FINAL NOTES

The following items will almost certainly cause problems in the translation process and must be investigated before successful translation:

- -instruction modification,
- -instruction-counter or label relative code references,
- -external address references,
- -BRK, TSX, TXS usage.

Non-position-independent code will not run under OS/9, although non-position-independent data will run in single-user mode. CSC markets a program to assist in conversion of programs to PIC/PID format for OS/9 or for other purposes. The translator cannot logically output 6809 PIC/PID code from 6502 code.