Computer-Assisted Translation of Programs from 6502 to 6809

by Edgar Pass

The article discusses techniques of translating 6502 programs to run on a 6809-based machine. Tables, 6809 routines, and discussion of special problems are included.

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. Table 1 presents the programming models for each of the processors, to indicate the flavor of some of the changes and extensions.

Register Comparison

The similarities and differences in the register structures of the processors are apparent in table 1. Of the three processors, 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 are not issues here.

When matching up 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

Table 1: **Programming Models for the 6800, 6809, and 6502**

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 and 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

Table B-1 *(continued)*

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Inherent
Logical shift Left	LSLA					48
	LSLB					58
	LSL		Ø8	68*	78	
Logical Shift	LSRA					44
	LSRB					54
	LSR		Ø4	64*	74	
Multiply	MUL					3D
Complement,2's	NEGA					4Ø
	NEGB					5Ø
	NEG		ØØ	6Ø*	7Ø	
No Operation	NOP					12
Inclusive OR	ORA	8A	9A	AA*	BA	
	ORB	CA	DA	EA*	FA	
	ORCC	1A				
Push Reg's on Stack	PSHS**					34
	PSHU**					36
Pull Reg's from Stack	PULS**					35
	PULU**					37
Rotate Left	ROLA					49
	ROLB					59
	ROL		Ø9	69*	79	
Rotate Right	RORA					46
	RORB					56
	ROR		Ø6	66*	76	
Subtract with Carry	SBCA	82	92	A2*	B2	
	SBCB	C2	D2	E2*	F2	
Sign Extend	SEX					1D
Store	STA		97	A7*	B7	
	STB		D7	E7*	F7	
	STD		DD	ED*	FD	
	STS		1ØDF	1ØEF*	1ØFF	
	STU		DF	EF*	FF	
	STX		9F	AF*	BF	
	STY		1Ø9F	1ØAF*	1ØBF	
Subtract	SUBA	8Ø	9Ø	AØ*	BØ	
	SUBB	CØ	DØ	EØ*	FØ	
	SUBD	83	93	A3*	B3	
Software Interrupt	SWI					3F
	SWI2					1Ø3F
	SWI3					113F
Sync to Int.	SYNC					13

Table B1 *(continued)*

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Inherent
Transfer Reg's	TFR**					1F
Test, Zero or Minus	TSTA					4D
	TSTB					5D
	TST		ØD	6D*	7D	

* Post byte required (see indexed addressing chart)
** Post byte specifying registers to be used is required.

Table B-2: Branch and Long Branch Instructions

Operation	Mnemonic	Relative	Direct	Indexed	Extended
Branch if Carry Clear	BCC	24			
	LBCC	1Ø24			
Branch if Carry Set	BCS	25			
	LBCS	1Ø25			
Branch if = Zero	BEQ	27			
	LBEQ	1Ø27			
Branch if >= Zero	BGE	2C			
	LBGE	1Ø2C			
Branch if > Zero	BGT	2E			
	LBGT	1Ø2E			
Branch if Higher	BHI	22			
	LBHI	1Ø22			
Branch if Higher/Same	BHS	24			
	LBHS	1Ø24			
Branch if <= Zero	BLE	2F			
	LBLE	1Ø2F			
Branch if Lower	BLO	25			
	LBLO	1Ø25			
Branch if Lower/Same	BLS	23			
	LBLS	1Ø23			
Branch if < Zero	BLT	2D			
	LBLT	1Ø2D			
Branch if Minus	BMI	2B			
	LBMI	1Ø2B			
Branch if Not = Zero	BNE	26			
	LBNE	1Ø26			
Branch if Plus	BPL	2A			
	LBPL	1Ø2A			
Branch Always	BRA	2Ø			
	LBRA	16			
Branch Never	BRN	21			
	LBRN	1Ø21			
Branch if V Clear	BVC	28			
	LBVC	1Ø28			
Branch if V Set	BVS	29			
	LBVS	1Ø29			
Branch to Subroutine	BSR	8D			
	LBSR	17			
Jump	JMP		ØE	6E*	7E
Jump to Subroutine	JSR		9D	AD*	BD
Return from Interrupt	RTI	3B (Implied)			
Return from Subroutine	RTS	39 (Implied)			

* Post byte required (see indexed addressing chart)

Table C: 6502 Op-Codes and Mnemonics

Operation	Mnemonic	Code	Addressing
Add with Carry	ADC	61	INDIRECT,X
	ADC	65	ZERO PAGE
	ADC	69	IMMEDIATE
	ADC	6D	ABSOLUTE
	ADC	71	INDIRECT,Y
	ADC	75	ZERO PAGE,X
	ADC	79	ABSOLUTE,Y
	ADC	7D	ABSOLUTE,X
And	AND	21	INDIRECT,X
	AND	25	ZERO PAGE
	AND	29	IMMEDIATE
	AND	2D	ABSOLUTE
	AND	31	INDIRECT,Y
	AND	35	ZERO PAGE,X
	AND	39	ABSOLUTE,Y
	AND	3D	ABSOLUTE,X
Arithmetic Shift Left	ASL	Ø6	ZERO PAGE
	ASL	ØA	ACCUMULATOR
	ASL	ØE	ABSOLUTE
	ASL	16	ZERO PAGE,X
	ASL	1E	ABSOLUTE,X
Branch	BCC	9Ø	RELATIVE
	BCS	BØ	RELATIVE
	BEQ	FØ	RELATIVE
	BMI	3Ø	RELATIVE
	BNE	DØ	RELATIVE
	BPL	1Ø	RELATIVE
	BVC	5Ø	RELATIVE
	BVS	7Ø	RELATIVE
Bit Test	BIT	24	ZERO PAGE
	BIT	2C	ABSOLUTE
Break	BRK	ØØ	IMPLIED
Clr Carry	CLC	18	IMPLIED
Clr Dec Mode	CLD	D8	IMPLIED
Clr Int Mask	CLI	58	IMPLIED
Clr Overflow	CLV	B8	IMPLIED
Jump to SR	JSR	2Ø	RELATIVE
Load Accumulator	LDA	A1	INDIRECT,X
	LDA	A5	ZERO PAGE
	LDA	A9	IMMEDIATE
	LDA	AD	ABSOLUTE
	LDA	B1	INDIRECT,Y
	LDA	B5	ZERO PAGE,X
	LDA	B9	ABSOLUTE,Y
	LDA	BD	ABSOLUTE,X
Compare Accumulator	CMP	C1	INDIRECT,X
	CMP	C5	ZERO PAGE
	CMP	C9	IMMEDIATE
	CMP	CD	ABSOLUTE
	CMP	D1	INDIRECT,Y
	CMP	D5	ZERO PAGE,X
	CMP	D9	ABSOLUTE,Y
	CMP	DD	ABSOLUTE,X
Compare X	CPX	EØ	IMMEDIATE
	CPX	E4	ZERO PAGE
	CPX	EC	ABSOLUTE
Compare Y	CPY	CØ	IMMEDIATE
	CPY	C4	ZERO PAGE
	CPY	CC	ABSOLUTE
Decrement	DEC	C6	ZERO PAGE
	DEC	CE	ABSOLUTE
	DEC	D6	ZERO PAGE,X
	DEC	DE	ABSOLUTE,X
Decrement-X	DEX	CA	IMPLIED
Decrement-Y	DEY	88	IMPLIED
Exclusive Or	EOR	41	INDIRECT,X
	EOR	45	ZERO PAGE
	EOR	49	IMMEDIATE
	EOR	4D	ABSOLUTE
	EOR	51	INDIRECT,Y
	EOR	55	ZERO PAGE,X
	EOR	59	ABSOLUTE,Y
	EOR	5D	ABSOLUTE,X
Increment	INC	E6	ZERO PAGE
	INC	EE	ABSOLUTE
	INC	F6	ZERO PAGE,X
	INC	FE	ABSOLUTE,X
Increment-X	INX	E8	IMPLIED
Increment-Y	INY	C8	IMPLIED
Jump	JMP	4C	ABSOLUTE
	JMP	6C	INDIRECT
Rotate Left	ROL	26	ZERO PAGE
	ROL	2A	ACCUMULATOR
	ROL	2E	ABSOLUTE
	ROL	36	ZERO PAGE,X
	ROL	3E	ABSOLUTE,X
Rotate Right	ROR	66	ZERO PAGE
	ROR	6A	ACCUMULATOR
	ROR	6E	ABSOLUTE
	ROR	76	ZERO PAGE,X
	ROR	7E	ABSOLUTE,X

Table C *(continued)*

Operation	Mnemonic	Code	Addressing
Load X	LDX	A2	IMMEDIATE
	LDX	A6	ZERO PAGE
	LDX	AE	ABSOLUTE
	LDX	B6	ZERO PAGE,Y
	LDX	BE	ABSOLUTE,Y
Load Y	LDY	AØ	IMMEDIATE
	LDY	A4	ZERO PAGE
	LDY	AC	ABSOLUTE
	LDY	B4	ZERO PAGE,X
	LDY	BC	ABSOLUTE,X
Logical Shift Right	LSR	46	ZERO PAGE
	LSR	4A	ACCUMULATOR
	LSR	4E	ABSOLUTE
	LSR	56	ZERO PAGE,X
	LSR	5E	ABSOLUTE
No Oper.	NOP	EA	IMPLIED
Inclusive OR	ORA	Ø1	INDIRECT,X
	ORA	Ø5	ZERO PAGE
	ORA	Ø9	IMMEDIATE
	ORA	ØD	ABSOLUTE
	ORA	11	INDIRECT,Y
	ORA	15	ZERO PAGE,X
	ORA	19	ABSOLUTE,Y
	ORA	1D	ABSOLUTE,X
Push Data	PHA	48	IMPLIED
	PHP	Ø8	IMPLIED
Pull Data	PLA	68	IMPLIED
	PLP	28	IMPLIED
Transfer Registers	TAX	AA	IMPLIED
	TAY	A8	IMPLIED
	TSX	BA	IMPLIED
	TXA	8A	IMPLIED
	TXS	9A	IMPLIED
	TYA	98	IMPLIED
Ret. f/Int.	RTI	4Ø	IMPLIED
Ret. f/SR	RTS	6Ø	IMPLIED
Subtract with Carry	SBC	E1	INDIRECT,X
	SBC	E5	ZERO PAGE
	SBC	E9	IMMEDIATE
	SBC	ED	ABSOLUTE
	SBC	F1	INDIRECT,Y
	SBC	F5	ZERO PAGE,X
	SBC	F9	ABSOLUTE,Y
	SBC	FD	ABSOLUTE,X
Set Carry	SEC	38	IMPLIED
Set Decimal	SED	F8	IMPLIED
Set Int Msk	SEI	78	IMPLIED
Store Accumulator	STA	81	INDIRECT,X
	STA	85	ZERO PAGE
	STA	8D	ABSOLUTE
	STA	91	INDIRECT,Y
	STA	95	ZERO PAGE,X
	STA	99	ABSOLUTE,Y
	STA	9D	ABSOLUTE,X
Store X	STX	86	ZERO PAGE
	STX	8E	ABSOLUTE
	STX	96	ZERO PAGE,Y
Store Y	STY	84	ZERO PAGE
	STY	8C	ABSOLUTE
	STY	94	ZERO PAGE,X

Note that, on the 65Ø2, Absolute addresses appear in low-order-byte-first sequence.

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 6502 "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 ways.

The addressing modes supported by each of the processors are generally similar, although there are a few significant differences. Table 2 presents the addressing modes of interest in each of the processors of interest.

One significant difference between the 6502 and the other two processors lies in the storage format of a 16-bit address. Whereas the Motorola processors store 16-bit addresses as high-order 8-bits, then low-order 8-bits in successive locations, the 6502 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 6502, 6800, and 6809. The most obvious is that the 6800 and 6809 use a 16-bit S register, whereas the 6502 uses an 8-bit S register and prefixes these 8-bits with an 8-bit constant 01 to form a 16-bit address. Thus the 6502 stack is restricted to addresses $0100-$01FF. The 6800 and 6502 decrement the stack pointer after placing a new item into it, whereas the 6809 decrements it before. Thus the 6800 and 6502 stack pointers always point to one address below the current stack limit, whereas the 6809 stack pointer always points to the last item placed onto the stack (if any). The TSX and TXS instructions on the 6800 (but not on the 6502) take this into account by adding one to the X register after transferring the contents of the the S register to it and by subtracting one from the S register after transferring the X register to it.

This difference can cause a problem when you translate programs from the 6800 to the 6809. However, because of the highly restricted nature of the 6502 S register, it should cause little difficulty in translating programs from the 6502 to the 6809. The main problem stems from the 6800 trick of using the stack pointer as a second index register. However, the 6502 Y register functions as a second index register in many addressing modes, and the 6502 S register is restricted to page 01 in memory addresses, eliminating it as an effective third index register on the 6502.

Table 3 summarizes many of the differences and similarities already discussed concerning the 6502, 6800, and 6809, 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 instructions. Complete instruction sets for each of the 6502, 6800, and 6809 processors may be

***Table 2:* Addressing Modes**

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; prefixed 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 00 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 00 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 effective address

found at the end of this article. An asterisk in table 3 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.

Emulation Discussion

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 assembler time, no run-time instruction modification is assumed.

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 processors' instruction sets cause time and space problems in the emulation process:

- 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.

***Table 3:* Summary Table**

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
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..		Ø			X
TXA		N.....Z.	....NZ..					X
TXS		....	....		X+1			X
TYA		N.....Z.	....NZ..					Y

Reversed Address Bytes

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. You must 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

Table 4: Translation Analysis

6502 Opcode	6809 Code	Comments
ADC Operand	ADC Operand	Add with Carry
	TFR CC,DP	Save CC Register
	TFR CC,A	
	ANDA #$02	
	STA SEVFLG	Set V Flag Byte
	TST SEDFLG	Check D Flag
	BEQ * + 7	
	TFR DP,CC	Restore CC Register
	DAA	Convert to Decimal
	BRA * + 4	
	TFR DP,CC	Restore CC Register
AND Operand	AND Operand	AND Accumulator
ASL Operand	ASL Operand	Arithmetic Shift Left
BCC Operand	BCC Operand	Check C Flag
BCS Operand	BCS Operand	Check C Flag
BEQ Operand	BEQ Operand	Check Z Flag
BIT Operand	ANDA Operand	Bit Test
	* N and V Flags Not Set	
BMI Operand	BMI Operand	Check N Flag
BNE Operand	BNE Operand	Check Z Flag
BPL Operand	BPL Operand	Check N Flag
BRK	SWI	(Requires Vector)
	* Interrupt Handler May Convert CC Format	
BVC Operand	TFR CC,DP	Save CC Register
	TST SEVFLG	Check V Flag Byte
	BNE * + 6	Change 6 to 7 for LBRA
	TFR DP,CC	Restore CC Register
	BRA Operand	Branch if V Clear
	TFR DP, CC	Restore CC Register
BVS Operand	TFR CC,DP	Save CC Register
	TST SEVFLG	Check V Flag Byte
	BEQ * + 6	Change 6 to 7 for LBRA
	TFR DP,CC	Restore CC Register
	BRA Operand	Branch if V Set
	TFR DP,CC	Restore CC Register
CLC	ANDCC #$FE	Clear C Flag
CLD	TFR CC,DP	Save CC Register
	CLR SEDFLG	Clear D Flag Byte
	TFR DP,CC	Restore CC Register
CLI	ANDCC #$EF	Clear I Flag
CLV	TFR CC,DP	Save CC Register
	CLR SEVFLG	Clear V Flag Byte
	TFR DP,CC	Restore CC Register
CMP Operand	CMPB Operand	Compare Accumulator
CPX Operand	EXG D,X	Prepare for Compare
	CMPB Operand	Compare X Register
	EXG X,D	
CPY Operand	EXG D,Y	Prepare for Compare
	CMPB Operand	Compare Y Register
	EXG Y,D	
DEC	DECB	Bump Accumulator Down
DEX	EXG X,D	Prepare for DEX
	LDA #$00	Clear MS 8 Bits, Not C Flag
	DECB	Bump X Down
	EXG D,X	Correct D and X
DEY	EXG Y,D	Prepare for DEY
	LDA #$00	Clear MS 8 Bits, Not C Flag
DECB	Bump Y Down	
	EXG D,Y	Correct D and Y
EOR Operand	EORB Operand	EOR Accumulator
INC	INCB	Bump Accumulator
INX	EXG X,D	Prepare for INX
	LDA #$00	Clear MS 8 Bits, Not C Flag
	INCB	Bump X Up
	EXG D,X	Correct D and X
INY	EXG Y,D	Prepare for INY
	LDA #$00	Clear MS 8 Bits, Not C Flag
INCB	Bump Y Up	
	EXG D,Y	Correct D and Y
JMP Operand	JMP Operand	Jump
JSR Operand	JSR Operand	Subroutine Call
LDA Operand	LDA Operand	Load Accumulator
LDX Operand	EXG X,D	Prepare for LDX
	LDA #$00	Clear MS 8 Bits, Not C Flag
	LDB Operand	Load Value
	EXG D,X	Correct D and X
LDY Operand	EXG Y,D	Prepare for LDY
	LDA #$00	Clear MS 8 Bits, Not C Flag
	LDB Operand	Load Value
	EXG D,Y	Correct D and Y
LSR Operand	LSR Operand	Logical Shift Right
NOP	NOP	No Operation
ORA Operand	ORB Operand	Or Accumulator
PHA	PSHS B	Push Accumulator
PHP	* Execute Cond Code Translation from 6809	
	PSHS A	Push 6502 CC Register
PLA	PULS B	Pull Accumulator
	TSTB	Set CC Register
PLP	PULS A	Pull 6502 CC Register
	* Execute Cond Code Translation to 6809	
ROL Operand	ROL Operand	Roll Left
ROR Operand	ROR Operand	Roll Right
RTI	RTI	Return from Interrupt
	* Interrupt Handler May Convert CC Format	
RTS	RTS	Exit Subroutine
SBC Operand	SBC Operand	Subtract with Borrow
	TFR CC,DP	Save CC Register
	TFR CC,A	
	ANDA #$02	

(Continued)

Table 4 *(Continued)*

6502 Opcode	6809 Code	Comments
	STA SEVFLG	Set V Flag Byte
	* Warning: Decimal Flag Not Honored	
	TFR DP,CC	Restore CC Register
SEC	ORCC #$01	Set C Flag
SED	TFR CC,A	Save CC Register
	STA SEDFLG	Set D Flag Byte
	TFR A,CC	Restore CC Register
SEI	ORCC #$10	Set I Flag
STA Operand	TFR CC,DP	Save CC Register
	STB Operand	Store Accumulator
	TFR DP,CC	Restore CC Register
STX Operand	EXG X,D	Prepare for Store
	TFR CC,DP	Save CC Register
	STB Operand	Store X Register
	TFR DP,CC	Restore CC Register
	EXG D,X	Restore D and X
STY Operand	EXG Y,D	Prepare for Store
	TFR CC,DP	Save CC Register
	STB Operand	Store X Register
	TFR DP,CC	Restore CC Register
	EXG D,Y	Restore D and Y
TAX	LDA #$00	Clear MS 8 Bits, Not C Flag
	TSTB	Set CC Register
	TFR D,X	Set X to Accumulator
TAY	LDA #$00	Clear MS 8 Bits, Not C Flag
	TSTB	Set Condition Code
	TFR D,Y	Set Y to Accumulator
TSX	TFR D,U	Save D Register
	TFR S,D	Get S Register
	LDA #$00	Clear MS 8 Bits, Not C Flag
	DECB	Correct Value
	TFR D,X	Set X Register
	TFR U,D	Restore D Register
TXA	TFR X,D	Move X to Accumulator
	TSTB	Set CC Register
TXS	TFR D,U	Save D Register
	TFR X,D	Get X Register
	TFR CC,DP	Save CC Register
	INCB	Correct Value
	TFR DP,CC	Restore CC Register
	TFR D,S	Set S Register
	TFR U,D	Restore V Register
TYA	TFR Y,D	Move Y to Accumulator
	TSTB	Set CC Register

COLOR COMPUTER USERS

THE POWERFUL FLEX DISK OPERATING SYSTEM WITH HUNDREDS OF SOFTWARE PACKAGES IS NOW AVAILABLE!

Now you can run FLEX, OS-9 and Radio Shack disk software on your Color Computer. If you have a 32K Color Computer with the Radio Shack disk system, all you need to do is make a trivial modification to access the hidden 32K, as described in the Feb. issue of COLOR COMPUTER NEWS and the April issue of '68' Micro. You can get FLEX from us right now. OS-9 will be ready by summer. Please note that this will only work with the Radio Shack disk system and 32K/64K memory chips that RS calls 32K. Maybe they put 64K's in yours, too. If you don't have a copy of the article, send a legal size SASE (40¢ stamps) and we'll send it to you.

In case you don't understand how this works, I'll give you a brief explanation. The Color Computer was designed so that the roms in the system could be turned off under software control. In a normal Color Computer this would only make it go away. However, if you put a program in memory to do something first (like boot in FLEX or OS-9), when you turn off the roms, you will have a full *64K RAM* System with which to run your program.

Now, we need the other half of the 64K ram chips to work, and this seems to be the case most of the time, as the article states. Of course, you could also put 64K chips in.

Some neat utilities are included.

MOVEROM moves Color Basic from ROM to RAM. Because it's moved to RAM you can not only access it from FLEX, you can run it and even change it!! You can load Color Computer cassette software and save it to FLEX disk. Single Drive Copy, Format and Setup commands plus an online help system are included.

Installing FLEX is simple. Insert the disk and type:

RUN "FLEX"

That's all there is to it! You are now up and running in the most popular disk operating system for the 6809. There are hundreds of software packages now running under the FLEX system. Open your Color Computer to a whole new world of software with FLEX.

FLEX $99.00

INCLUDES OVER 25 UTILITIES!

FLEX Editor	$ 50.00
FLEX Assembler	$ 50.00
FLEX Standard BASIC	$ 65.00
FLEX Extended Business BASIC	$100.00

Other languages available include; FORTH, Pascal, Fortran77, 'C,' A/BASIC Compiler, plus more.

Application packages include; A/R, G/L, A/P, Inventory, Electronic Spreadsheets, Accounting, Database programs and more. SEND FOR LIST.

TRS-80 COLOR COMPUTER COMPLETE WITH 64K RAM, 24K ROM, SINGLE DISK DRIVE AND FLEX, SET UP AND READY TO RUN FOR ONLY $1,275. Includes 60 day extended warranty. If you have a Computer, call about RS disk controllers and drives.

FRANK HOGG LABORATORY, INC.

130 MIDTOWN PLAZA • SYRACUSE NEW YORK 13210 • (315)474-7856

(Continued from page 81)

The 6809 has more instructions that 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.

Condition Code Register Format

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 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 #$20
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

Page zero wraparound is another attribute of the 6502 which is not present on the 6809 and must be handled by the translator through 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 in 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 this 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 similar to that provided earlier, with the exception of the last line, which would use indirect addressing and would be in 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 8-Bit Limitation of X and Y

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 through the D register in order to properly extend or truncate the value. For instance, the translator-generated 6809 code for INX would be:

```
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 on the program being translated. However, they may be severe, and the emulation overhead will usually be small.

Translation Analysis

Table 4 presents a simplified representation of the required translator actions in the conversion of each 6502 instruction to 6809 instructions. The following assumptions are made implicitly in this table:

- address mode processing is handled separately but always presents a 16-bit effective address
- absolute addresses are stored in 6809 format (high, then low bytes)
- stack register is handled using 6809 16-bit format and is not restricted to $01XX range
- format conversion of the condition code register is not handled:
  no "B" flag handling is required
  "D" and "V" flags are handled as separate flag bytes
- X and Y registers are restricted to 8 bits
- situations such as "too-long" branches must be handled by the programmer after translation

Conversion Analysis

Most computer programs, even on microcomputers, do not run standalone 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 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 modifing 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 available 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 means of performing the conversion, though sometimes the most difficult, 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 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.

Summary

The preceding discussion has presented a method to convert 6502 source programs to 6809 source programs. This conversion is performed in two phases.

The first phase is a low-level (instruction-by-instruction) translation process which could be performed manually or by using a computer program. 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 phase is higher-level, and must generally be performed manually (although possibly with the assistance of an 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. This process 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.

Tables summarizing the instruction sets of the 6502, 6800, and 6809 processors follow.

Edgar Pass may be contacted at Computer Systems Consultants, Inc., 1454 Latta Lane, Conyers, GA 30207.

Table A-1: 6800,01,02,03,08 Op-Codes and Mnemonics

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Inherent
Add	ADDA	8B	9B	AB	BB	
	ADDB	CB	DB	EB	FB	
Add Double Acc	ADDD*	C3	D3	E3	F3	
Add Accum.	ABA					1B
Add With Carry	ADCA	89	99	A9	B9	
	ADCB	C9	D9	E9	F9	
And	ANDA	84	94	A4	B4	
	ANDB	C4	D4	E4	F4	
Bit Test	BITA	85	95	A5	B5	
	BITB	C5	D5	E5	F5	
Clear	CLR			6F	7F	
	CLRA					4F
	CLRB					5F
Compare	CMPA	81	91	A1	B1	
	CMPB	C1	D1	E1	F1	
Compare Accum.	CBA					11
Complement,1's	COM			63	73	
	COMA					43
	COMB					53
Complement,2's	NEG			6Ø	7Ø	
	NEGA					4Ø
	NEGB					5Ø
Dec Adj Acc.	DAA					19
Decrement	DEC			6A	7A	
	DECA					4A
	DECB					5A
Exclusive OR	EORA	88	98	A8	B8	
	EORB	C8	D8	E8	F8	
Increment	INC			6C	7C	
	INCA					4C
	INCB					5C
Load Accum.	LDAA	86	96	A6	B6	
	LDAB	C6	D6	E6	F6	
Load Doub Acc	LDAD*	CC	DC	EC	FC	
Multiply	MUL*					3D
Inclusive OR	ORAA	8A	9A	AA	BA	
	ORAB	CA	DA	EA	FA	
Push Data	PSHA					36
	PSHB					37
Pull Data	PULA					32
	PULB					33
Rotate Left	ROL			69	79	
	ROLA					49
	ROLB					59

* Not available in 68ØØ,68Ø2,or 68Ø8

Table A-1 *(continued)*

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Inherent
Shift Left	ASL			68	78	
Arithmetic	ASLA					48
	ASLB					58
Double	ASLD*					Ø5
Shift Right	ASR			67	77	
Arithmetic	ASRA					47
	ASRB					57
Shift Right	LSR			64	74	
Logical	LSRA					44
	LSRB					54
Double	LSRD*					Ø4
Store Accum	STAA		97	A7	B7	
	STAB		D7	E7	F7	
Doub. Accum.	STAD*		DD	ED	FD	
Subtract	SUBA	8Ø	9Ø	AØ	BØ	
	SUBB	CØ	DØ	EØ	FØ	
Double	SUBD*	83	93	A3	B3	
Subtract Acc.	SBA					1Ø
Subtract	SBCA	82	92	A2	B2	
With Carry	SBCB	C2	D2	E2	F2	
Transfer	TAB					16
Accumulators	TBA					17
Test Zero or	TST			6D	7D	
Minus	TSTA					4D
	TSTB					5D

* Not available in 68ØØ,68Ø2,or 68Ø8

Table A-2: Index Register and Stack Manipulation Instructions

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Implied
Compare IXR	CPX	8C	9C	AC	BC	
Decrement IXR	DEX					Ø9
Decrment SP	DES					34
Increment IXR	INX					Ø8
Increment SP	INS					31
Load IXR	LDX	CE	DE	EE	FE	
Load SP	LDS	8E	9E	AE	BE	
Store IXR	STX		DF	EF	FF	
Store SP	STS		9F	AF	BF	
IXR-->SP	TXS					35
SP-->IXR	TSX					3Ø
Add B to X	ABX*					3A
Push IXR	PSHX*					3C
Pull IXR	PULX*					38
Rotate Right	ROR			66	76	
	RORA					46
	RORB					56

* Not available in 68ØØ,68Ø2, or 68Ø8

Table A-3: 6800,01,02,03,08 Op-Codes and Mnemonics

CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

Operation	Mnemonic	Implied
Clear Carry	CLC	ØC
Clear Int Msk	CLI	ØE
Clr Overflow	CLV	ØA
Set Carry	SEC	ØD
Set Int Msk	SEI	ØF
Set Overflow	SEV	ØB
Acc A-->CCR	TAP	Ø6
CCR-->Acc A	TPA	Ø7

Table A-4: Jump and Branch Instructions

Operation	Mnemonic	Relative	Indexed	Extended	Implied
Branch Always	BRA	2Ø			
Branch if Carry Clear	BCC	24			
Branch if Carry Set	BCS	25			
Branch if = Zero	BEQ	27			
Branch if >= Zero	BGE	2C			
Branch if > Zero	BGT	2E			
Branch if Higher	BHI	22			
Branch if <= Zero	BLE	2F			
Branch if Lower/Same	BLS	23			
Branch if < Zero	BLT	2D			
Branch if Minus	BMI	2B			
Branch if Not = Zero	BNE	26			
Branch if V Clear	BVC	28			
Branch if V Set	BVS	29			
Branch if Plus	BPL	2A			
Branch to Subroutine	BSR	8D			
Jump	JMP		6E	7E	
Jump to Subroutine	JSR		AD	BD	
No Operation	NOP				Ø1
Return from Interrupt	RTI				3B
Return from Subroutine	RTS				39
Software Interrupt	SWI				3F
Wait for Interrupt	WAI				3E

Table B-1: 6809 Op-Codes and Mnemonics

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Inherent
Add B to X	ABX					3A
Add w/ carry	ADCA	89	99	A9*	B9	
	ADCB	C9	D9	E9*	F9	
Add	ADDA	8B	9B	AB*	BB	
	ADDB	CB	DB	EB*	FB	
	ADDD	C3	D3	E3*	F3	
And	ANDA	84	94	A4*	B4	
	ANDB	C4	D4	E4*	F4	
	ANDCC	1C				

Table B-1 *(continued)*

Operation	Mnemonic	Immediate	Direct	Indexed	Extended	Inherent
Arithmetic Shift Left	ASLA					48
	ASLB					58
	ASL		Ø8	68*	78	
Arithmetic Shift Right	ASRA					47
	ASRB					57
	ASR		Ø7	67*	77	
Bit Test	BITA	85	95	A5*	B5	
	BITB	C5	D5	E5*	F5	
Clear	CLRA					4F
	CLRB					5F
	CLR		ØF	6F*	7F	
Compare	CMPA	81	91	A1*	B1	
	CMPB	C1	D1	E1*	F1	
	CMPD	1Ø83	1Ø93	1ØA3*	1ØB3	
	CMPS	118C	119C	11AC*	11BC	
	CMPU	1183	1193	11A3*	11B3	
	CMPX	8C	9C	AC*	BC	
	CMPY	1Ø8C	1Ø9C	1ØAC*	1ØBC	
Complement,1's	COMA					43
	COMB					53
	COM		Ø3	63*	73	
Wait for int.	CWAI					3C
Dec. adj Acc.	DAA					19
Decrement	DECA					4A
	DECB					5A
	DEC		ØA	6A*	7A	
Exclusive OR	EORA	88	98	A8*	B8	
	EORB	C8	D8	E8*	F8	
Exchange Reg's	EXG**					1E
Increment	INCA					4C
	INCB					5C
	INC		ØC	6C*	7C	
Load	LDA	86	96	A6*	B6	
	LDB	C6	D6	E6*	F6	
	LDD	CC	DC	EC*	FC	
	LDS	1ØCE	1ØDE	1ØEE*	1ØFE	
	LDU	CE	DE	EE*	FE	
	LDX	8E	9E	AE*	BE	
	LDY	108E	1Ø9E	1ØAE*	1ØBE	
Load Effective Address	LEAS			32*		
	LEAU			33*		
	LEAX			3Ø*		
	LEAY			31*		

* Post byte required (see indexed addressing chart)
** Post byte specifying registers to be used is required.