Dr. Robert J. Cotter
The Johns Hopkins
School of Medicine
725 North Wolfe Street
Baltimore MD 21205

Programming the 1802

Dr. Cotter's last 1802 article appeared in the December 1978 issue. With this "welcome back" 1802 article, he explains how to input and output data, add subtract and multiply.

In "The Amazing 1802" (*Kilobaud*, No. 20, p. 102) and "Interfacing the Elf II" (*Kilobaud*, No. 24, p. 40), I described some hardware additions for expanding the Elf II. This time I'd like to share some programming techniques that will enable you to perform some simple arithmetic calculations on the Elf. This article will also introduce you to some methods for writing and calling subroutines you may wish to incorporate into your own programs.

If you did build the address decoder described in the latter article, you will find it easier to write and debug the programs and subroutines described below, since you can step up to the correct memory locations for entering the subroutines. If you did not, then have no fear, since all of the programs in this article have been written on the original 256 memory locations that come with the basic Elf II.

In Table 1 I have listed a subset of the 1802 instructions necessary for writing all of the programs in this article. Only one- and two-byte instructions are used, since the three-byte instructions are used for branching onto additonal (256 byte) pages of memory. I've also included the "level 1" mnemonics so you may translate the programs into the appropriate op code if you are using a different computer.

Before we begin programming, recall also that, in addition to the 256 8-bit memory locations, the Elf II has sixteen 16-bit registers and the 8-bit special purpose registers N, P, X and D. The 16-bit registers point to memory locations; the N, P and X registers point to the 16-bit registers; and the D register is used for arithemetic and logic operations.

decimal	hexadecimal	binary
75	4B	0100 1011
+58	+3A	+0011 1010
133	85	1000 0101

Example 1.

location	bytes	comments
00	F8 4B	4B→D
02	FC 3A	3A + D→D,DF
04	52	D→M(R2)
05	E2	R2 = RX
06	64	M(RX)→hex displays

Example 2.

How to Add Two Numbers

After I finished soldering my Elf II, my first inclination was to try to add two numbers together, even though it is a lot easier on a calculator. It takes more steps on a microcomputer, and you have to think in hexadecimal and binary to understand what is going on. Example 1 shows the addition of two numbers, 75 and 58.

The simplest way is to put the first number, 4B, into the D register using the LOAD IMMEDIATE instruction F8. We enter:

```
F8 4B
```

The D register then acts as an *accumulator* so that when you add the number 3A to the contents of D using the ADD IMMEDIATE:

```
FC 3A
```

the D register will then contain the answer, which must now be read out on the displays.

The Elf II will output numbers only from memory locations (not registers). The 52 instruction will transfer our answer to the memory location pointed to by R2, one of the 16-bit registers. Output instructions, however, will only fetch data pointed to by the register designated by the 8-bit X register. An E2 instruction places a 2 in register X, so that R2 = RX. A 64 instruction puts the answer on the displays. The whole program is shown in Example 2. If you enter the program and press the RUN switch, the hex displays will show the answer: 85.

So far, so good! But what happens if you add two numbers whose sum is greater than 256 (requiring more than 8 bits)? See Example 3. The binary addition produces a "carry" into the ninth bit. Fortunately, this bit is retained in a special register, the DF register, whenever the ADD IMMEDIATE command, FC, is used. A rewrite of our program for this addition is shown in Example 4. We have added three new steps that test DF for a "carry" bit and turn on the LED to indicate the carry.

Improving the Program

We need to improve the program. In step 04 we momentarily placed our answer in memory, and in steps 05 and 06 we designated this location as the source of our output display. This location is determined by R2, which may be pointing to any arbitrary location when the Elf is turned on. It may, in fact, write the answer in one of the program locations and erase

Op Code	Mnemonic	Name	Operation
1N	INC	INCREMENT REG N	RN + 1
2N	DEC	DECREMENT REG N	RN – 1
8N	GLO	GET LOW REG N	RN.0→D
9N	GHI	GET HIGH REG N	RN.1→D
AN	PLO	PUT LOW REG N	D→RN.0
BN	PHI	PUT HIGH REG N	D→RN.1
7A	REQ	RESET Q	0→Q (light off)
7B	SEQ	SET Q	1→Q (light on)
DN	SEP	SET P	N→P
EN	SEX	SET X	N→X
4N	LDA	LOAD ADVANCE	MN→D,RN + 1
5N	STR	STORE VIA N	D→MN
F0	LDX	LOAD VIA X	MX→D
F1	OR	OR	MX OR D→D
F2	AND	AND	MX AND D→D
F3	XOR	EXCLUSIVE-OR	MX XOR D→D
F6	SHR	SHIFT RIGHT	shift D right LSB→DF
76	SHRC	SHIFT RIGHT WITH CARRY	rotate D right LSB→DF, DF→MSB
FE	SHL	SHIFT LEFT	shift D left MSB→DF
7E	SHLC	SHIFT LEFT WITH CARRY	rotate D left MSB→DF, DF→LSB
F4	ADD	ADD	MX + D→DF,D
74	ADC	ADD WITH CARRY	MX + D + DF→DF,D
F5	SD	SUBTRACT D	MX – D→DF,D
75	SDB	SUBTRACT WITH BORROW	MX – D – $\overline{DF}$→DF,D
C4	NOP	NO OPERATION	continue
6N	OUT	OUTPUT (N = 1 – 7)	MX→BUS, RX + 1
64		(N = 4)	MX→hex display
6N	INP	INPUT (N = 9 – F)	BUS→D,MX
6C		(N = C)	keyboard→D,MX
30 MM	BR	UNCOND SHORT BRANCH	GO TO MM
31 MM	BQ	SHORT BRANCH IF Q = 1	GO TO MM if Q = 1
39 MM	BNQ	SHORT BRANCH IF Q = 0	GO TO MM if Q = 0
32 MM	BZ	SHORT BRANCH IF D = 0	GO TO MM if D = 00
3A MM	BNZ	SHORT BRANCH IF D≠0	GO TO MM if D≠00
33 MM	BDF	SHORT BRANCH IF DF = 1	GO TO MM if DF = 1
3B MM	BNF	SHORT BRANCH IF DF = 0	GO TO MM if DF = 0
37 MM	B4	SHORT BRANCH IF EF4 = 1	GO TO MM if INPUT switch is down
3F MM	BN4	SHORT BRANCH IF EF4 = 0	GO TO MM if INPUT switch is up
F8 KK	LDI	LOAD IMMEDIATE	KK→D
F9 KK	ORI	OR IMMEDIATE	KK OR D→D
FA KK	ANI	AND IMMEDIATE	KK AND D→D
FB KK	XRI	XOR IMMEDIATE	KK XOR D→D
FD KK	SDI	SUBTRACT D IMMEDIATE	KK – D – $\overline{DF}$→DF,D
FC KK	ADI	ADD IMMEDIATE	KK + D→DF,D

Table 1. COSMAC 1802 instruction sheet.

an instruction! Therefore, we will need to set R2 to point to some location well beyond the program.

Also, it is inconvenient, to say the least, to have to write a new program for each set of additions. Therefore, we will rewrite our program so that after execution is begun it awaits two operands from the keyboard, performs the computation, displays the answer and resets the program for the next computation. Program A gives the listing for such a program.

Step 1 sets R2 to point to location A0. R2 is called the "stack pointer," which we will set at A0 for all the programs in this article. All variables will then be located in memory beginning at this location and may be fetched as they are used in computations. Step 2 resets the carry register, DF, by loading 00 into the D register and then shifting 0 into DF. This is necessary since DF may contain a logical "1" when the computer is turned on. Step 3 designates our stack pointer, R2, as the output register.

Steps 4 to 6 illustrate the technique for accepting variables from the keyboard during execution. There are two loops (at steps 4 and 5) that wait for the INPUT switch to be depressed and released. If a number has been entered on the keyboard before pressing the INPUT switch, the instruction 6C will place that number into D and in the first position on the stack, in this case, A0. The 7A instruction resets Q, since it may be ON from a previous

decimal	hexadecimal	binary
58	3A	0011 1010
+242	+F2	+1111 0010
300	12C	1 0010 1100

Example 3.

location	bytes	comments
00	F8 3A	3A→D
02	FC F2	F2 + D→D,DF
04	52	D→M(R2)
05	E2	X→2
06	64	M(RX)→hex displays
07	33 0A	GO TO 0A if DF = 1
09	00	STOP
0A	7B	Q→ON

Example 4.

computation. The 64 instruction displays the number we have just entered so that we can verify that it was entered correctly.

Steps 9 to 11 are used to accept the second operand and copy it into D and onto A1, the second position on the stack. The 64 instruction again verifies this number on the displays.

Addition will not take place until the INPUT switch has been depressed and released one more time (steps 13, 14). In step 15 the stack pointer returns to the top of the stack and adds (instruction F4) the number in location A0 with the contents of D (which still holds the second operand). The instructions 52 and 64 read out the answer, and the remaining steps test the carry register, DF, before returning the program for the next computation.

Try the program out using the previous examples. Press: RUN; 4B INPUT; 3A INPUT; INPUT. The display will read "85" and the LED will be off. Next, enter: 3A INPUT; F2 INPUT; INPUT. The displays will show "2C" and the LED will be on, indicating a carry.

Double-Precision Arithmetic

It is common in computer computations to work in "double precision." For an 8-bit computer this means working

Location	Bytes	Step	Comments
0000	F8 A0 A2	1	A0→D, D→R2.0
03	F8 00 F6	2	00→D, shift D right
06	E2	3	2→X
07	3F 07	4	GO TO 07 if INPUT switch is up
09	37 09	5	GO TO 09 if INPUT switch is down
0B	6C	6	keyboard bytes→D, M2
0C	7A	7	reset Q
0D	64	8	M2→hex display
0E	3F 0E	9	GO TO 0E if INPUT switch is up
10	37 10	10	GO TO 10 if INPUT switch is down
12	6C	11	keyboard bytes→D, M2
13	64	12	M2→hex display
14	3F 14	13	GO TO 14 if INPUT switch is up
16	37 16	14	GO TO 16 if INPUT switch is down
18	22 22	15	R2 – 1, R2 – 1
1A	F4	16	MX + D→DF,D
1B	52	17	D→M2
1C	64	18	M2→hex displays
1D	33 21	19	GO TO 21 if DF = 1
1F	30 00	20	GO TO 00
21	7B	21	Q→ON
22	30 00	22	GO TO 00

Program A. Addition program.

Memory Allocation

0000-003F	MAIN PROGRAM used for reading in variables, determining the type of computation and displaying answers
0040-0049	ADD subroutine
0050-0059	SUBT subroutine
0060-0080	MULT subroutine
00A0-00A4	stack pointer storage of operands, answers and subroutine locations A0 operand 1, high-order byte A1 operand 1, low-order byte A2 operand 2, high-order byte A3 operand 2, low-order byte A4 subroutine to be called

Register Allocation

R2 = R(SP)	the stack pointer
R3 = R(PC)	the program counter, used to call subroutines
R5 = R(RET)	stores return location for main program
RF = R(ACC)	accumulator register; contains one of the operands and the answer.

Table 2.

with 16 bits, which gives an effective computation range from 0 to 65,535. Because the D register is only 8 bits long, it can no longer be used as the accumulator. Instead, the 16-bit register, RF, will be used as the accumulator, R(ACC), while R2 will still serve as the stack pointer, R(SP).

Also, at this stage we want to think ahead a little. We will want to write programs for subtraction and multiplication, as well as addition. We can cut down on our programming if we write all of these operations as subroutines. Therefore, we will first turn our attention to writing a program that will input the variables, store them on a stack, call the appropriate subroutine and display the answers. This will be the MAIN program.

Table 2 shows how we will organize the memory locations to accommodate the MAIN program, the subroutines and the variables stored via the stack pointer. We will also designate two new registers for subroutining. Register R3 will be used to point to a subroutine, while register R5 will store the return location when the subroutine has been executed.

The MAIN Program

The program to input and output the variables is listed as Program B. Step 1 sets the stack pointer, R(SP), while steps 2 and 3 clear DF and Q. Steps 4 to 6 should now be familiar as the loop that awaits the variables that will be copied onto the stack beginning at location A0. The program will accept two double-precision operands and store these in locations A0 to A3. It also accepts a fifth number (40, 50 or 60) to indicate which subroutine (ADD, SUBT or MULT) is to be executed.

Steps 7 to 9 test to see if the stack pointer has reached A5. It then stops accepting variables and continues execution. Instruction 22 returns the stack pointer to A4, which contains the address of the subroutine. In step 11 this is loaded into the pointer register, R3. Step 12 sets the return location into R5.

In step 13, the stack pointer is returned to the top of the stack at A0 and in the next two steps loads the high and low bytes into the 16-bit accumulator register. The stack pointer is pointing to A2 when the D3 instruction sends the program pointer to the appropriate subroutine location.

Step 17 is the return location from the subroutine. The instruction D0 restores control of the program pointer by the register R0. Steps 18 and 19 place

decimal	hexadecimal
39,730	9B32
+ 49,905	+ C2F1
89,635	15E23

Example 5.

✓ Reader Service—see page 179

the high-order byte of the answer (from the accumulator register) into memory and onto the output displays, while steps 24 and 25 do the same for the low-order byte after the INPUT switch has been depressed and released.

The 7B instruction in step 21 lights the LED if there has been a carry, and steps 26 to 28 return the program for a new computation if the INPUT switch is depressed one more time. The MAIN program can be used with any of the subroutines that follow.

The ADD Subroutine

Addition is accomplished in the D register by adding the low-order bits from memory and the accumulator register first, returning the 8-bit result to the accumulator and then repeating the process for the high-order bits. The first addition uses the F4 instruction (ADD), but since this addition may result in a "carry" into the ninth bit (register DF), the instruction 74 (ADD WITH CARRY) is used in the addition of the high-order bits. A carry after the second addition is tested in the MAIN program.

The ADD subroutine is listed as Program C and begins at location 40. Remember that upon entering the subroutine the stack pointer is aimed at location A2. The first step in the subroutine increments R2 to fetch the low-order bits first. At

Location	Bytes	Step	Comments
0000	F8 A0 A2	1	set R(SP)
03	F8 00 F6	2	clear DF
06	7A	3	reset Q
07	3F 07	4	wait for INPUT to be depressed
09	37 09	5	wait for INPUT to be released
0B	E2 6C 64	6	accept inputs from keyboard, store, display and increment R(SP)
0E	82	7	R2.0→D
0F	FB A5	8	D XOR A5
11	3A 06	9	GO TO 06 if D≠0
13	22	10	decrement R(SP) to location A4
14	F0 A3	11	M2→D; D→R3.0 (subroutine pointer)
16	F8 23 A5	12	set return location in R5
19	F8 A0 A2	13	reset R(SP) to A0
1C	F0 BF 12	14	M2→RF.1; R2 + 1
1F	F0 AF 12	15	M2→RF.0; R2 +
22	D3	16	call subroutine; 3→P
23	D0	17	0→P
24	9F 52	18	RF.1→D, D→M2
26	64	19	display high-order bytes of answer
27	3B 2A	20	GO TO 2A if DF = 0
29	7B	21	light Q
2A	3F 2A	22	wait for INPUT to be depressed
2C	37 2C	23	wait for INPUT to be released
2E	8F 52	24	RF.0→D, D→M2
30	64	25	display low-order byte of answer
31	3F 31	26	wait for INPUT to be depressed
33	37 33	27	wait for INPUT to be released
35	30 00	28	return

Program B. MAIN program.

Location	Bytes	Step	Comments
0040	12	1	R2 + 1
41	E2	2	x→2
42	8F	3	RF.0→D; fetch R(ACC) low-order bits
43	F4	4	M2 + D→D,DF; add low-order bits
44	AF	5	D→RF.0; store result in R(ACC)
45	22	6	R2 – 1
46	9F	7	RF.1→D; fetch R(ACC) high-order bits
47	74	8	M2 + D + DF→D,DF; add high-order bits and carry
48	BF	9	D→RF.1; store result in R(ACC)
49	D5	10	return to MAIN program

Program C. Add subroutine.

✓ Reader Service—see page 179

the end of the subroutine, the instruction D5 makes register R5 the program counter. Its initial location was set by the MAIN program as the return address.

Let's add two double precision numbers, represented in decimal and hexadecimal, to see how the program works (see Example 5). Enter both the MAIN program and the ADD subroutine into the computer and press the RUN switch. Then enter: 9B INPUT, 32 INPUT, C2 INPUT, F1 INPUT, 40 INPUT. The last entry is the location of the subroutine and indicates that you wish to add the two numbers.

The displays will show the number 5E and the LED will be lit, indicating the carry. If you press the INPUT switch again, the displays will read 23. One more depression of the INPUT switch readies the program for the next computation.

The SUBT Subroutine

The subroutine for subtraction is listed as Program D and is similar to the ADD subroutine. In this case, however, subtraction of the low-order bits is accomplished with instruction F5 (SUBTRACT), while the high-order bits use 75 (SUBTRACT WITH BORROW). Whenever a borrow occurs, DF will be set to "0." When it does not, DF = "1." Again, let's try an example:

```
 A217
-3C85
 6592
```

Enter the SUBT subroutine, press the RUN switch and then enter: 3C INPUT, 85 INPUT, A2 INPUT, 17 INPUT, 50 INPUT. Note that the subtrahend is entered first and that the last input calls the SUBT subroutine. The display will show 65, and depressing the INPUT switch will give the low-order byte 92. The LED will be lit, indicating that DF = 1, that no borrow has occurred and that the answer is positive.

Suppose then, that we subtract a larger number from a smaller one:

```
 3C85
-A217
-6592
```

Location	Bytes	Step	Comments
0050	12	1	R2 + 1
51	E2	2	x→2
52	8F	3	RF.0→D; fetch R(ACC) low-order bits
53	F5	4	M2 – D→D,DF; subtract low bits
54	AF	5	D→RF.0; store result in R(ACC)
55	22	6	R2 – 1
56	9F	7	RF.1→D; fetch R(ACC) high-order bits
57	75	8	M2 – D – $\overline{DF}$→DF,D; subtract with borrow
58	BF	9	D→RF.1; store result in R(ACC)
59	D5	10	return to MAIN program

Program D. Subt subroutine.

Location	Bytes	Step	Comments
0060	F8 00 AF	1	clear accumulator
63	22	2	R2 – 1
64	E2	3	x→2
65	F0	4	M2→D
66	3A 69	5	GO TO 69 if D≠00
68	D0	6	return to MAIN program
69	F6	7	shift D right
6A	52	8	D→M2
6B	12	9	R2 + 1
6C	3B 77	10	GO TO 77 if DF = 0
6E	F8 73 A5	11	set return location for MULT
71	30 40	12	GO TO ADD subroutine
73	F8 77 A3	13	reset R3
76	D3	14	3→P
77	12	15	R2 + 1
78	F0	16	M2→D
79	FE	17	shift left; MSB→DF
7A	52	18	D→M2
7B	22	19	R2 – 1
7C	F0	20	M2→D
7D	7E	21	rotate D left; DF→LSB; MSB→DF
7E	52	22	D→M2
7F	30 63	23	GO TO 63

Program E. Mult subroutine.

The answer will appear as 9A6E, and the LED will be off. This indicates that the answer is negative and that we have the "2's complement" of the correct answer. To get the negative result, subtract that number from 0000, and the display will read 6592 with the LED off (negative).

The MULT Subroutine

Multiplication in binary is done by shifting the multiplicand left and adding. Therefore, the MULT subroutine, listed as Program E, uses ADD as a subroutine. Also, since the multiplication of two 8-bit numbers produces a 16-bit answer, the subroutine used here will accept only 8-bit operands from the MAIN program.

The subroutine works as follows. The multiplier is placed in the D register and shifted right. DF is then tested. If DF = 1, then the multiplicand is entered into the accumulator register. The multiplicand is then shifted left. The multiplier is shifted again and DF is tested. If DF = 1, then the ADD routine is called to add the new multiplicand to the accumulator, and the muliplicand is again shifted left. If DF = 0, no addition takes place, but the multiplicand is still shifted left.

The ADD subroutine is called by a GO TO statement rather than a program pointer, since, unlike the MAIN program, MULT will always call this subroutine. This means that it enters the ADD subroutine still under the control of R3. Before entering the ADD subroutine, however, MULT changes the R5 location so that the ADD subroutine will return to location 73 and not the MAIN program.

Return to the MAIN program occurs when the multiplier equals zero. Return is accomplished by placing control of the program in the R0 register, which is pointing to the next location in the MAIN program from where we left it.

Let's try an example. Suppose we wish to perform the following multiplication:

```
  85
x AB
58D7
```

Enter the MULT subroutine, press the RUN switch and enter: 00 INPUT, 85 INPUT, 00 INPUT, AB INPUT, 60 INPUT. 58 and D7 will appear on the displays.

Where Do We Go from Here?

All of this seems fairly complicated compared to what you can do on a simple, inexpensive calculator. However, there are several distinct advantages. First, the subroutines could be used to perform calculations on data that is being continuously fed to the computer through an A/D converter—or two separate inputs could be compared, with the sum or difference plotted using a D/A converter.

Using the techniques described here, you may also wish to write a division subroutine or expand the multiplication subroutine for double precision. You also may rewrite the MAIN program to solve an equation like y = ax + b. The main advantage that the microcomputer has over the programmable calculator is its ability to accept data from something other than a keyboard.

In future articles I plan to discuss several other things you can do with your Elf II: for example, memory expansion in 256, 1K or 4K steps will allow you to expand your computer at a rate you can afford; an autoranging A/D converter will automatically give you the most siginificant 8 bits, for many different signal levels. I have also recently purchased the COSMAC Evaluation Kit and plan to do a comparison of that system with the Elf II. I also plan an article on timing for generating waveforms and one on a Teletype interface. ■