SIMPLE ARITHMETIC

It is now possible for us to take a detailed look at several machine code programs that show how some simple arithmetic is performed using the 6809 instruction set. We pay particular attention to signed arithmetic, and the use of the condition code register.

At this stage in the course, we can put some instructions together into a working program, although we will need to examine some new instructions and ways of representing data first of all. We will begin by devising a simple program that converts a binary coded decimal (BCD) number into its binary representation.

A binary coded decimal number (see page 168) is a way of representing a decimal number in binary form that is particularly useful when dealing with eight-bit processors. Using this representation, each digit in a decimal number is translated into its binary equivalent. The decimal number 69, for example, is equivalent to the BCD representation %01101001: the leftmost four bits (0110) are the binary equivalent of 6, and the rightmost four bits (1001) are equal to the decimal 9. Thus, using BCD, we get an entirely different decimal equivalent than we would if we were converting the *binary* number %01101001 (it is equivalent to 105 decimal).

Our conversion program will need a number of new instructions; let's consider these in turn:

- LSR (Logical Shift Right): This shifts every bit of the operand one place to the right. The rightmost bit is shifted into the carry bit of the condition code register of the processor, and a zero is shifted into the leftmost bit of the operand.

- AND: This logically ANDs each bit of a register with the corresponding bits of the operand, leaving the result in the register. This instruction is most often used to *mask* certain bits: if a register contains a one in a bit, then ANDing it with another bit will copy that second bit into the register; if the register bit contains a zero, then ANDing it will always result in a zero. For example, the effect of ANDing a register value of %00001111 with a given memory location is to copy the rightmost four bits only of the location into the register. Thus:

```
%00001111 Register value
%10110110 AND memory location value
%00000110 Result in register
```

- MUL: This MULtiplies the contents of the A and B registers, leaving the result in the D register (the 16-bit register formed from A and B together). Very few other eight-bit processors support multiplication as an op-code.

- SWI (SoftWare Interrupt): This is a convenient way of terminating a machine code program, returning control to the operating system. We shall examine this instruction in more detail when we consider the interrupt system later in the course. Here is the BCD-to-binary program:

- Specify value in location counter:

```
ORG $1000
```

- Store BCD 58 in BCDNUM and reserve byte at BINNUM:

```
BCDNUM FCB %01011000
BINNUM RMB 1
```

- Load BCD 58 into the A register and mask the lower digit. Store that digit in BINNUM:

```
START LDA  BCDNUM
      ANDA #%00001111
      STA  BINNUM
```

- Load BCD 58 into A accumulator and shift upper digit (leftmost four bits) rightwards:

```
SHIFT LSRA
      LSRA
      LSRA
      LSRA
```

- Load 10 (decimal) into the B register and multiply by the contents of A:

```
MULT LDB  #10
     MUL
```

- The result is 16 bits in the D register, but as this result cannot be greater than 90 ($10 \times 9 = 90$), only the lower byte of D is needed. The lower byte is in the B register — so add its contents to BINNUM and store the result:

```
ADDIT ADDB BINNUM
      STB  BINNUM
```

- Thus, we have the BCD number in BCDNUM and the binary equivalent stored in BINNUM. We can finally return to the operating system and end the source code:

```
RETURN SWI
END
```

TWO'S COMPLEMENT

The programming examples we have given so far in the course have all involved simple arithmetic, and we shall continue in this vein for a little while longer. Let's now look at the problem of *sign* — by which we mean positive and negative numbers.