do this at the point where the required value appears, since it is difficult to ensure that the flags are not changed by any intervening instructions.

The flags are tested by means of 'branch' instructions, which are the low-level equivalent of the BASIC GOTO command. The 6809 uses relative (rather than absolute) branches almost exclusively. The difference is that a *relative* branch transfers control by so many bytes forward (or back), while an *absolute* branch transfers control to a specified address. The effect, however, is the same. It distinguishes between *short* branches, where the range is expressed in a single byte (-128 to 127), and *long* branches, which can go anywhere in memory. We will be using short branches only.

The 6809 has a large set of branch instructions, and we will introduce these as we need them. The following examples illustrate the instructions used to test and compare the values held in the accumulators and the use of the branch instructions to select and repeat procedures.

- ANDCC: It is not possible to load values directly into the condition code register, but it is good practice to set all the flags you need to zero before you start using them. The easiest way of doing this is by using the ANDCC instruction, which operates just like an AND command, using zeros as masks in the bit positions we want to use.

- SUB (SUBtract): The operand is subtracted from the accumulator, which sets the C, V, Z and N flags on the result. (The H flag is also set if the subtraction is eight-bit).

- CMP (CoMPare): This works in exactly the same way as SUB, except that the contents of the register are left unchanged. As in SUB, the C, V, Z, N (and possibly H) flags are set.

- BRA (the unconditional BRAnch): This is just like the BASIC GOTO command.

- BGT (Branch if Greater Than zero): This is a test for the signed numbers. The branch takes place if Z is zero (the number is non-zero). To allow for the fact that the sign bit may be incorrectly set if overflow has occurred, either N must be zero and V also zero (straighforward non-negative) or N must be one and V also one (incorrectly negative due to overflow). Other similar tests for signed numbers are BGE, BLT and BLE.

- BLO (Branch if LOwer than zero): This is an unsigned test, since it is pointless inspecting N with unsigned numbers. The branch occurs if the C flag is set, indicating a borrow after a subtraction. Similar unsigned tests are BLS, BHI and BHS.

- A program to find the larger of two signed eight-bit numbers stored in $3000 and $3001. The larger of the two numbers to be placed in $3002. First label the numbers:

```
NUM1    EQU     $3000
NUM2    EQU     $3001
ANS     EQU     $3002
        ORG     $1000
```

- The code begins: the condition code flags are set to zero and the first number is loaded. This is compared with the other number:

```
        ANDCC   #%11110000
        LDA     NUM1
        CMPA    NUM2
```

- If NUM1 is the larger, then the program branches to FINISH. Otherwise it loads the second number into the A register. Whichever number is in the register when FINISH is reached is then stored in ANS, and the program returns to the operating system and ENDs:

```
        BGT     FINISH
        LDA     NUM2
FINISH  STA     ANS
        SWI
        END
```

Original Directives
The differing effects that assembler directives and Assembly language statements have on the assembler's location counter and on the contents of memory can be seen in this example

Original Directives

```
LABEL   OP-CODE  OPERAND      LOCATION   MEMORY
FIELD   FIELD    FIELD        COUNTER    CONTENTS
*----------DEMONSTRATION-------------
RESET   EQU      $F100
INDEX   EQU      16
MASK1   EQU      %01101010    $0400      ??
                              $0400      ??
                              $0400      ??

                 ORG     $1000
                              $1000      ??

CR      FCB      16          $1000      $10

MEMTOP  FDB      $7FFF       $1001      $7F
                             $1002      $FF

TABLE1  RMB      7           $1003      ??
                             $1004      ??
                             $1005      ??
                             $1006      ??
                             $1007      ??
                             $1008      ??
                             $1009      ??

ERRMSG  FCC      'ERROR      $100A      $45
                             $100B      $52
                             $100C      $52
                             $100D      $4F
                             $100E      $52

                 CLRA        $100F      $4F

                 END         $100F      ??

*-------SYMBOL TABLE----------------
RESET   F100   INDEX   0010   MASK1   006A
CR      1000   MEMTOP  1001   TABLE1  1003
ERRMSG  100A
```

No ORG has been issued, so the location address is the assembler's default setting. Note that location address is not affected by EQU, and the contents of memory are as yet undefined

This sets the location as specified, but memory contents remain undefined

FCB causes the operand to be stored in the byte addressed by the location counter

FDB causes two bytes to be initialised

RMB reserves 7 bytes (contents undefined) by incrementing the location counter by that number

The ASCII codes of the operand string are placed in memory by the FCC directive

At last — an Assembly language operation! There is no operand, so we have only one byte for the op-code

Another directive that does not affect the location counter

This is how the symbols used in the program would be stored in the assembler's workspace for its own reference during the Assembly