vector, the address to which control is transferred on power-up or a hardware reset; this is usually the start address of the ROM monitor. Furthermore, the two bytes at $FFF0 and $FFF1 are reserved by Motorola for possible future use.

Information about interrupts is contained in three bits of the condition code register (CC): bit 4 (I), bit 6 (F) and bit 7 (E). Setting the I bit to one masks the $\overline{\text{IRQ}}$ interrupt, setting the F bit masks $\overline{\text{FIRQ}}$. The E bit is used by the processor to differentiate between $\overline{\text{IRQ}}$ or $\overline{\text{NMI}}$, and $\overline{\text{FIRQ}}$: if E gets set to one then there has been an $\overline{\text{IRQ}}$ or $\overline{\text{NMI}}$, if E gets set to zero then an $\overline{\text{FIRQ}}$ has occurred.

When an interrupt is received, it is treated in a similar way to a subroutine call: the contents of some or all of the registers are stacked so that control can be returned to the same point in the program currently being executed. The interrupt service routine ends with an RTI instruction (similar to an RTS), which unstacks the registers and returns control to the original program.

The actual difference between the $\overline{\text{FIRQ}}$ and the other two interrupts is that $\overline{\text{FIRQ}}$ stacks the program counter (PC) and the condition code (CC) registers only, and is therefore much faster in operation than the other two. The interrupt service routine, however, must restore any registers that it uses, so this type of interrupt should not be used for routines that use more than just one or two registers. We can see now where the E bit is used because the same RTI instruction is used to terminate $\overline{\text{IRQ}}$ and $\overline{\text{FIRQ}}$ routines, but the processor must determine which registers need unstacking. The sequence of events when an interrupt occurs is:

1) The current instruction is executed.
2) The I bit is set, disabling other $\overline{\text{IRQ}}$ interrupts. If the interrupt was a $\overline{\text{FIRQ}}$ or a $\overline{\text{NMI}}$ then bit F is also set to disable $\overline{\text{FIRQ}}$. SW12 and SW13 do not mask other interrupts, but SW1 does.
3) On an $\overline{\text{FIRQ}}$ bit E is cleared to zero, otherwise it is set to one.
4) The vector in the appropriate memory locations is loaded into PC and execution continues from that address.

Our first program looks at another good use for interrupts; namely maintaining a real-time clock. We shall assume that some timing device, which could be a special purpose chip like the 6840 interval timer or a division of the system clock or a modification of the 50Hz mains, is connected to a PIA at $5000. The first subroutine will enable the interrupts and set up a 16-bit counter at $50. The interrupt service routine will simply increment the counter so that at any time inspection of $50 will give the number of timing signals that have been received, from which the time can be calculated if the start-up time and the frequency of the timing signals are known.

The second example program assumes a printer is connected to the same PIA at $E000. We shall employ a buffer, of indeterminate length, at $100 to store one line of output to be printed by the service routine. A flag at $50 is set to zero while the line is being printed, and to one when the line is finished. This will enable some other routine (which we shall not be concerned with here) to refill the buffer. Locations $51 and $52 contain a pointer into the buffer giving the address of the next character to be printed. The first subroutine sets up the PIA, flag and buffer pointer for a new line.

Program One

```
PIACR   EQU   $E001          PIA control register
PIADR   EQU   $E000          PIA data register
INTRP   EQU   $2000
CLK1    EQU   $50
CLK2    EQU   $51
        ORG   $1000          Subroutine to initialise clock
INITCK  CLR   CLK1           Clear clock locations
        CLR   CLK2
        LDA   #%00000101     Enable PIA interrupts
        STA   PIACR
        ANDCC #%11101111     Enable IRQ
WAITCK  TST   CLK2           Wait for first increment
        BEQ   WAITCK
        RTS
        ORG   INTRP          Interrupt service routine
        LDA   PIADA          Clear interrupt
        LDD   CLK1           Get count
        ADDD  #1             Increment count
        STD   CLK1
        RT1
```

Program Two

```
PIACR   EQU   $E001          PIA control register
PIADR   EQU   $E000          PIA data register
INTRP   EQU   $2000
CR      EQU   13             Carriage return
BUFFER  EQU   $100           Address of buffer
BUFPTR  EQU   $51            Buffer pointer
FLAG    EQU   $50            End-of-line flag
        ORG   $1000          Subroutine to set everything up
        CLR   FLAG           Clear end-of-line flag
        LDX   #BUFFER        Initialise buffer pointer
                             to start of buffer
        STX   BUFPTR
        CLR   PIACR          Address data direction register
        LDA   $FF            Set all lines to output
        STA   PIADR
        LDA   #%00000101     Enable PIA interrupts
        STA   PIACR
        ANDCC #%11101111     Enable IRQ
        RTS
        ORG   INTRP          Interrupt service routine
        LDX   BUFPTR         Buffer pointer
        LDA   ,X+            Get next character from buffer
        STA   PIADR          Print it
        LDB   PIADR          Clear interrupt
        STX   BUFPTR         Incremented buffer pointer
        CMPA  #CR            Was it end-of-line?
        BNE   FINISH         Skip if not end-of-line
        INC   FLAG           Else set flag
FINISH  RTI
```