INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONES: 224281/2/3/4/5/6
CABLE: CENTRATOM - TELEX 460392 - I

H4.SMR/159-25

THIRD COLLEGE ON MICROPROCESSORS:
TECHNOLOGY AND APPLICATIONS IN PHYSICS

7 October - 1 November 1985

INTERRUPTS

W. VON RUDEN
EF DIVISION, CERN
Geneva 23, Switzerland

Interrupts

The three classical methods of input/output are

- programmed
- interrupt driven
- direct memory access

They can be classified as follows:

	programmed	interrupt	DMA
hardware	low	low	high
software	simple,limited	medium	complex
speed	low	medium	high

The actual transfer speed in both programmed and interrupt mode is not different, but the programmed mode will be much slower and cumbersome to program, if many I/O devices have to be serviced. Both interrupts and DMA allow for 'multi-tasking', programmed I/O needs complex 'polling'.

Basic principle of interrupts

When an interrupt occurs, the context of the running program must be preserved, the control is passed to the 'interrupt service', which does the necessary work to handle the interrupt. At the end of this 'service', the previous status must be restored and control is returned to the interrupted program.

This means, that an interrupted program does not realize that it was interrupted, except that its execution is slowed down.

Interrupts with the M6809

The M6809 has the **hardware** interrupts Reset, NMI, IRQ and FIRQ and the **software** interrupts SWI, SWI2 and SWI3.

The CPU handles all hardware interrupts in a very similar way, the differences will be explained as we go along. When an interrupt occurs, the M6809 finishes the execution of the present instruction, saves all the registers on the S stack (only the PC and the CC for FIRQ), updates the interrupt mask bits I and F and the E bit. It then fetches the interrupt vector from the table located at the top of the memory map and puts it in the PC, which means that instruction execution is passed to the 'interrupt service' routine. This routine should clear the interrupt source and finally execute the 'RTI' instruction, whereby the CPU will recover its status from the stack and resume instruction execution in the interrupted program.

The **Reset** is not really an interrupt, because it does not save the CPU status, but it is usually included in the discussion of the interrupts due to the way it is done in the CPU.

The **NMI** or **N**on-**M**askable-**I**nterrupt is an edge triggered input to the CPU. As the name says, it cannot be masked and whenever a high to low transition is detected on the NMI input, the CPU will execute the NMI sequence. The only exeption is after Reset, where the NMI is blocked until the S stack pointer is loaded for the first time. **Caution**: If more than one device is connected to the NMI line, special care has to be taken to avoid dead-locks, which may arrise when the second device pulls the NMI line down while the first one is being treated and the line was low anyway; no further transition can occur and the system may hang.

The **IRQ** or **I**nterrupt **R**e**Q**uest is a level sensitive interrupt input of the M6809 and is the most commonly used interrupt, because it is the easiest to use.

The **FIRQ** or **F**ast **I**nterrupt **R**e**Q**uest is foreseen for devices which need a fast service and where only a small number of CPU registers are needed to service it. It is the programmers responsability to save such registers and to restore them before the RTI is executed.

The **SWI**s or **S**oft**W**are **I**nterrupts are under program control and they are very useful to implement operating system features such as breakpoints and monitor calls. Monitor calls are independent of changes in the system software, as long as the

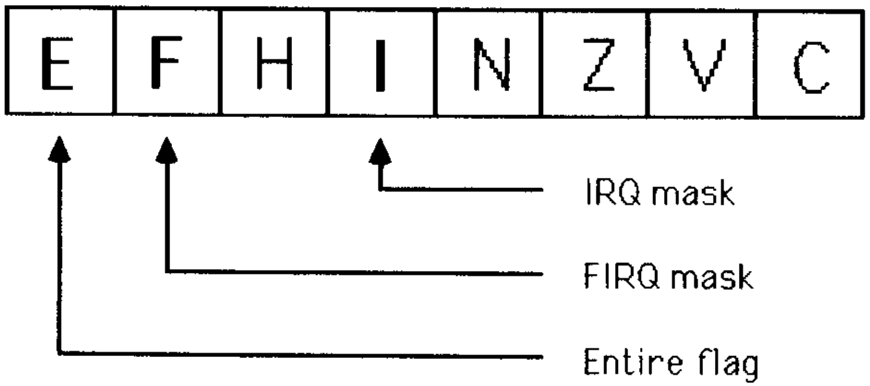
numbers are fixed, and they also allow the system to use all registers, because the CPU saves them automatically. The interrupt sequence for SWIs is identical to the hardware interrupt sequence, with the only the difference that they are synchronized to the flow of the program.

Interrupt vector table

Reset	FFFE/F	highest priority
NMI	FFFC/D	
SWI	FFFA/B	
IRQ	FFF8/9	
FIRQ	FFF6/7	has priority over IRQ
SWI2	FFF4/5	
SWI3	FFF2/3	lowest priority
reserved	FFF0/1	

The priorities are determined through the way the interrupt mask bits are set in the condition code register (CC) by the interrupts. The next figure shows the position of the bits related to interrupts. The 'Entire' bit is needed, because there is only one RTI instruction to be used both with normal interrupts and the FIRQ. E=1 means that all CPU registers have been saved on the stack, for E=0 only the PC and the CC registers are saved.

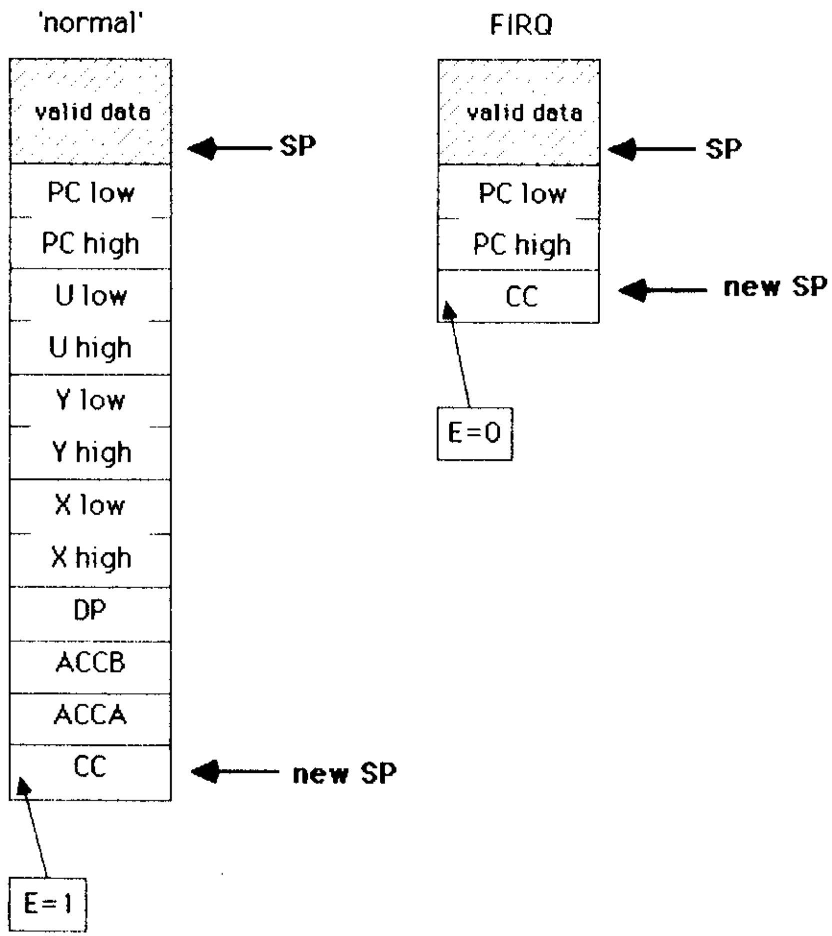
Note that this bit is set/cleared by the CPU before CC is saved on the stack.



Interrupt	bits set	bits not set	possible
Reset	I, F		NMI after 'LDS'
NMI	E, I, F		NMI
SWI	E, I, F		NMI
IRQ	E, I	F	NMI, FIRQ
FIRQ	I, F	E	NMI
SWI2	E	I, F	NMI, FIRQ, IRQ
SWI3	E	I, F	NMI, FIRQ, IRQ

Stacking of registers

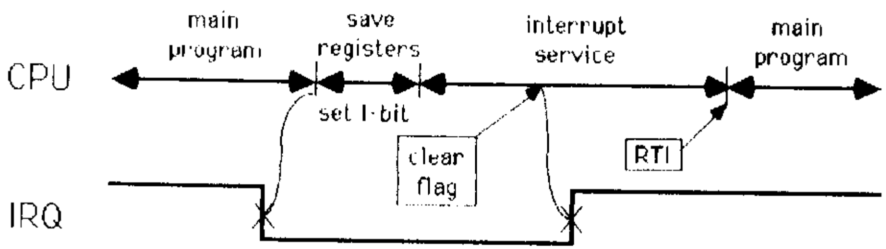
The CPU saves the registers automatically when an interrupt has been accepted. We distinguish two cases:



When 'RTI' is executed, CC is pulled first and therefore the CPU will know, if all registers have to be recovered or only the PC in the case of the FIRQ.

Timing example for IRQ

The following example shows a typical IRQ sequence. Note that the IRQ line goes up as soon as the interrupt condition in the peripheral chip has been cleared, but the interrupt service routine is only finished when RTI is executed. Should another interrupt arrive on the same line before the RTI, the IRQ line goes low again or stays low, but the interrupt will only be serviced after the RTI, because the I-bit will be up during the complete service time.



Case study

In the following, we give an example of using the IRQ with a PIA on the ROSY station. We assume, that a push-button has been connected to CA1 and that we like to see the high-to-low transition. Whenever the button is pressed, we increment a counter. If the counter reaches a predefined maximum value, the main program is informed by setting a flag and a message is printed.

The general strategy is as follows:

Main program:

1. define the interrupt vector
2. initialize the hardware
3. enable the IRQ
4. wait for flag and do other things

Interrupt service:

1. determine interrupt source
2. clear interrupt flag and increment counter
3. if count=max, set flag and clear counter
4. return from interrupt

```
        NAM     COUNT
*
*       This program uses a push-button connected to CA1
*       of the PIA at address $EF08 to count pulses via
*       interrupts.
*       If a maximum count is reached, an event flag is set
*       for the main program, which will print a message.
*
        LIB     MONCALLS        include ROSY definitions
*
EOT     EQU     4
PIA     EQU     $EF08           define base address
ORA     EQU     PIA
CRA     EQU     PIA+1
*
*       reserve variables needed
*
CNT     RMB     1               counter
EVENT   RMB     1               event flag
MAXCNT  EQU     100             define max count
*
*       Main program
*
Start   EQU     *
        CLR     COUNT           init all stuff
        CLR     EVENT
        LEAX    PUSHB,PCR       vector for push-button
        LDB     #3              vector code for IRQ
        MON     VECTOR          set vector in ROSY
        TSTB                    any error?
        BNE     WRONG           yes, in deed
*
*       init PIA
*
        LDA     #5              set access ORA and int. enable
        STA     CRA
```



```
*
*       now enable IRQs
*
        ANDCC #$EF
*
*       Main loop starts here
*
MAINLP  EQU     *
        TST     EVENT           any event?
        BEQ     NONE            no !
        LEAX    MAXMSG,PCR      send message
        MON     PRINT
        CLR     EVENT           reset flag
*
NONE    EQU     *
        •
        •
        here we can do other things
        •
        •
        BRA     MAINLP
*
*
MAXMSG  FCC     /Maximum count reached/
        FCB     EOT



*
*       Error handling
*
WRONG   EQU     *
        MON     ERROR
        MON     RETURN          give up if error
```

```
*
*       Interrupt service routine
*
PUSHB   EQU   *
        LDA   CRA              was it PIA ?
        BMI   FOUND            yes
        LDB   #14              send error message
        MON   ERROR            "Undefined IRQ"
        BRA   P_OUT
*
FOUND   EQU   *
        LDA   ORA              clear CRA-7 flag
        LDA   CNT              increment count
        INCA
        CMPA  #MAXCNT          maximum reached ?
        BLO   NEXT             not yet
        CLRA                   reset cnt, set flag
        INC   EVENT
NEXT    EQU   *
        STA   CNT              save new count
P_OUT   RTI                    That's all folks !
*
        END
```