Sequence of Events During Interrupt
1. Hardware needs service (busy-to-done) transition.
2. Flag is set in one of the I/O status registers.
   (a) Interrupting event sets the flag (ex., STAF=1).
   (b) The device is armed (ex., STA1=1).
   (c) Microcomputer interrupts are enabled (ex., I=0).
3. Thread switch.
   (a) Microcomputer finishes current instruction.
   (b) All registers are pushed onto the stack.
   (c) Vector address is obtained and put into the PC.
   (d) Microcomputer sets I=1.
4. Execution of the ISR.
5. Return control back to the thread that was running.

General Features of Interrupts
- All interrupting systems must have the:
  1. Ability for hardware to request action from computer.
  2. Ability for computer to determine the source.
  3. Ability for computer to acknowledge the interrupt.
- To *arm* (*disarm*) a device means to enable (shut off) the source of interrupts.
- To *enable* (*disable*) means to allow (postpone) interrupts at this time.

6811 Stack Before and After an Interrupt
(See Figure 4.18)
6811 Interrupts

- 6811 has two external requests TRQ and XTRQ.
- Other interrupt sources include:
  - A STRA interrupt
  - Three input capture interrupts
  - Five output capture interrupts
  - Three timer interrupts (timer overflow, RTI, pulse accumulator)
  - Two serial port interrupts (SCI and SPI)

6811 Interrupts

- Interrupts have a fixed priority, but can elevate one to highest priority using hardware priority interrupt (HPRI) register.
- XIRQ is highest-priority device and has separate vector and enable bit (X).
- Once X bit is cleared, software cannot disable it.
- XIRQ handler sets X and I, and restores with rti.

6811 Interrupt Vectors and Priority

<table>
<thead>
<tr>
<th>Vector</th>
<th>Interrupt Source</th>
<th>Enable</th>
<th>Arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFFF</td>
<td>Power on reset</td>
<td>Always</td>
<td>Always highest</td>
</tr>
<tr>
<td>$FFFF</td>
<td>Hardware reset</td>
<td>Always</td>
<td>Always</td>
</tr>
<tr>
<td>$FFFC</td>
<td>COP clk monitor fail</td>
<td>Always</td>
<td>OPTION.CME=1</td>
</tr>
<tr>
<td>$FFFA</td>
<td>COP failure</td>
<td>Always</td>
<td>CONFIG.NOCP=0</td>
</tr>
<tr>
<td>$FFF4</td>
<td>Nonmaskable XIRQ</td>
<td>X=0</td>
<td>External hardware</td>
</tr>
<tr>
<td>$FFF2</td>
<td>External IRQ</td>
<td>I=0</td>
<td>External hardware</td>
</tr>
<tr>
<td>$FFF0</td>
<td>Real time int., RTIF</td>
<td>I=0</td>
<td>TMSK2.RTH=1</td>
</tr>
<tr>
<td>$FFEE</td>
<td>Int capture 1, IC1F</td>
<td>I=0</td>
<td>TMSK1.IC1=1</td>
</tr>
<tr>
<td>$FFEC</td>
<td>Int capture 2, IC2F</td>
<td>I=0</td>
<td>TMSK1.IC2=1</td>
</tr>
<tr>
<td>$FFEA</td>
<td>Int capture 3, IC3F</td>
<td>I=0</td>
<td>TMSK1.IC3=1</td>
</tr>
<tr>
<td>$FFE8</td>
<td>Outp capture 1, OC1F</td>
<td>I=0</td>
<td>TMSK1.OC1I=1</td>
</tr>
<tr>
<td>$FFE6</td>
<td>Outp capture 2, OC2F</td>
<td>I=0</td>
<td>TMSK1.OC2I=1</td>
</tr>
<tr>
<td>$FFE4</td>
<td>Outp capture 3, OC3F</td>
<td>I=0</td>
<td>TMSK1.OC3I=1</td>
</tr>
</tbody>
</table>

6811 Interrupt Vectors and Priority (cont)

<table>
<thead>
<tr>
<th>Vector</th>
<th>Interrupt Source</th>
<th>Enable</th>
<th>Arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFE2</td>
<td>Outp capture 4, OC4F</td>
<td>I=0</td>
<td>TMSK1.OC4I=1</td>
</tr>
<tr>
<td>$FFE0</td>
<td>Outp capture 5, OC5F</td>
<td>I=0</td>
<td>TMSK1.OC5I=1</td>
</tr>
<tr>
<td>$FFDE</td>
<td>Timer overflow, TOF</td>
<td>I=0</td>
<td>TMSK2.TOF=1</td>
</tr>
<tr>
<td>$FFDC</td>
<td>Pulse accum overflow</td>
<td>I=0</td>
<td>TMSK2.PAOV=1</td>
</tr>
<tr>
<td>$FFDA</td>
<td>Pulse accum inp edge</td>
<td>I=0</td>
<td>TMSK2.PAH=1</td>
</tr>
<tr>
<td>$FFD8</td>
<td>SPI complete, SPIF</td>
<td>I=0</td>
<td>SPCR.SPIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Rx data reg full, RDRF</td>
<td>I=0</td>
<td>SCCR2.RIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Tx overrun, OVRN</td>
<td>I=0</td>
<td>SCCR2.RIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Tx data reg full, TDRE</td>
<td>I=0</td>
<td>SCCR2.TIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Tx complete, TC</td>
<td>I=0</td>
<td>SCCR2.TCIE=1</td>
</tr>
<tr>
<td>$FFD6</td>
<td>Idle line detect, IDLE</td>
<td>I=0</td>
<td>SCCR2.ILIE=1</td>
</tr>
<tr>
<td>$FFF8</td>
<td>Illegal opcode trap</td>
<td>Always</td>
<td>Always</td>
</tr>
<tr>
<td>$FFF6</td>
<td>Software interrupt SWI</td>
<td>Always</td>
<td>Always lowest</td>
</tr>
</tbody>
</table>
Setting Interrupt Vectors

<table>
<thead>
<tr>
<th>Vector</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>org $FFF0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fdb RTIHAN</td>
<td>Ptr to real time interrupt handler</td>
<td></td>
</tr>
<tr>
<td>org $FFF2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fdb IRQHAN</td>
<td>Ptr to external IRQ and STRA handler</td>
<td></td>
</tr>
<tr>
<td>org $FFF4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fdb XIRQHAN</td>
<td>Ptr to external XIRQ handler</td>
<td></td>
</tr>
<tr>
<td>fdb RESETHAN</td>
<td>Ptr to reset handler</td>
<td></td>
</tr>
</tbody>
</table>

6811 Pseudo-Vectors

<table>
<thead>
<tr>
<th>Vector</th>
<th>Vector Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFD6</td>
<td>SCI</td>
<td>$00C4-$00C6</td>
</tr>
<tr>
<td>$FFD8</td>
<td>SPI</td>
<td>$00C7-$00C9</td>
</tr>
<tr>
<td>$FFDA</td>
<td>Pulse accum inp edge</td>
<td>$00CA-$00CC</td>
</tr>
<tr>
<td>$FFDC</td>
<td>Pulse accum overflow</td>
<td>$00CD-$00CF</td>
</tr>
<tr>
<td>$FFDE</td>
<td>Timer overflow, TOF</td>
<td>$00D0-$00D2</td>
</tr>
<tr>
<td>$FFE0</td>
<td>Outp capture 5, OC5F</td>
<td>$00D3-$00D5</td>
</tr>
<tr>
<td>$FFE2</td>
<td>Outp capture 4, OC4F</td>
<td>$00D6-$00D8</td>
</tr>
<tr>
<td>$FFE4</td>
<td>Outp capture 3, OC3F</td>
<td>$00D9-$00DB</td>
</tr>
<tr>
<td>$FFE6</td>
<td>Outp capture 2, OC2F</td>
<td>$00DC-$00DE</td>
</tr>
<tr>
<td>$FFE8</td>
<td>Outp capture 1, OC1F</td>
<td>$00DF-$00E1</td>
</tr>
</tbody>
</table>

6811 Pseudo-Vectors (cont)

<table>
<thead>
<tr>
<th>Vector</th>
<th>Vector Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFEA</td>
<td>Inp capture 3, IC3F</td>
<td>$00E2-$00E4</td>
</tr>
<tr>
<td>$FFEC</td>
<td>Inp capture 2, IC2F</td>
<td>$00E5-$00E7</td>
</tr>
<tr>
<td>$FFEE</td>
<td>Inp capture 1, IC1F</td>
<td>$00E8-$00EA</td>
</tr>
<tr>
<td>$FFF0</td>
<td>Real time int., RTIF</td>
<td>$00EB-$00ED</td>
</tr>
<tr>
<td>$FFF2</td>
<td>External IRQ, STAF</td>
<td>$00EE-$00F0</td>
</tr>
<tr>
<td>$FFF4</td>
<td>Nonmaskable XIRQ</td>
<td>$00F1-$00F3</td>
</tr>
<tr>
<td>$FFF6</td>
<td>Software interrupt SWI</td>
<td>$00F4-$00F6</td>
</tr>
<tr>
<td>$FFF8</td>
<td>Illegal opcode trap</td>
<td>$00F7-$00F9</td>
</tr>
<tr>
<td>$FFF9</td>
<td>COP failure</td>
<td>$00FA-$00FC</td>
</tr>
<tr>
<td>$FFFC</td>
<td>COP clk monitor fail</td>
<td>$00FD-$00FF</td>
</tr>
</tbody>
</table>

Setting Interrupt Pseudo-Vectors

<table>
<thead>
<tr>
<th>Vector</th>
<th>Opcode for JMP</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa #$7E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>staa $00EB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldx #RTIHAN</td>
<td>Ptr to real time interrupt handler</td>
<td></td>
</tr>
<tr>
<td>stx $00EC</td>
<td>JMP RTIHAN</td>
<td></td>
</tr>
<tr>
<td>ldaa #$7E</td>
<td>Opcode for JMP</td>
<td></td>
</tr>
<tr>
<td>staa $00EE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldx #IRQHAN</td>
<td>Ptr to external IRQ and STRA handler</td>
<td></td>
</tr>
<tr>
<td>stx $00EF</td>
<td>JMP IRQHAN</td>
<td></td>
</tr>
<tr>
<td>ldaa #$7E</td>
<td>Opcode for JMP</td>
<td></td>
</tr>
<tr>
<td>staa $00F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldx #XIRQHAN</td>
<td>Ptr to external IRQ and STRA handler</td>
<td></td>
</tr>
<tr>
<td>stx $00F2</td>
<td>JMP XIRQHAN</td>
<td></td>
</tr>
</tbody>
</table>
External Interrupt Design Approach

- First, identify status signal that indicates the busy-to-done state transition.
- Next, connect the I/O status signal to a microcomputer input that can generate interrupts.

(See Figure 4.21)

Interrupting Software

1. Ritual - executed once, disable interrupts during, initialize globals, set port dir, set port interrupt ctrl reg, clear interrupt flag, arm device, and enable interrupts.
2. Main program - initialize SP, execute ritual, interacts with ISRs via global data (ex. FIFO queue).
3. ISR(s) - determine interrupt source, implement priority, acknowledge (clear the flag) or disarm, exchange info w/main program via globals, execute rti to exit.
4. Interrupt vectors - in general purpose processors vectors in RAM, in embedded systems usually in ROM.

Polled Versus Vectored Interrupts

- Vectored interrupts - each interrupt source has a unique interrupt vector address.
- Polled interrupts - multiple interrupt sources share the same interrupt vector address.
  - Minimal polling - check flag bit that caused interrupt.
  - Polling for 0s and 1s - verify entire status register.

External I/O Device(s) Connected to the Microcomputer

(See Figures 4.22 and 4.23)
Example of a Vectored Interrupt

```assembly
Example of a Vectored Interrupt

TimeHan ldaa #$80 ; TOF is bit 7
staa TFLG2 ; clear TOF
; Timer interrupt calculations
rti

ExtHan ldaa PIOC
    lda PORTCL ; clear STAF
; External interrupt calculations
rti
    org $FFDE ; timer overflow
    fdb TimeHan
    org $FFF2 ; IRQ external
    fdb ExtHan
```

Example of a Polled Interrupt

```assembly
Example of a Polled Interrupt

ExtHan ldaa PIOC ; which one
    bita #$80 ; STAF?
    bne STAFHan
    ldaa OtherStatus1
    bita #$80 ; External?
    bne OtherHan
    swi ; error
STAFHan ldaa PORTC ; clear STAF
; STAF interrupt calculations
rti

OtherHan ldaa OtherData
; Other interrupt calculations
rti
    org $FFF2 ; IRQ external
    fdb ExtHan
```

Keyboard Interface Using Interrupts

(See Figures 4.24, 4.25, and 4.26)

Interrupting Keyboard Software in C

```c
// PC6-PC0 inputs = key DATA, STRA=STROBE interrupt
void Init(void){
    unsigned char dummy;
    asm("sei");
    PIOC=0x42; // EGA=1, STAI
    DDRC=0x80; // STRA=STROBE
    PORTC=0x00; // PC7=0
    dummy=PIOC; dummy=PORTCL;
    InitFifo();
    asm("cli");}
#pragma interrupt_handler ExtHan()
void ExtHan(void){
    if((PIOC & STAF)==0)asm("swi");
    PutFifo(PORTCL);} // ack
```
Interrupting Keyboard Software in Assembly
; PC6-PC0 inputs = keyboard DATA
; STRA=STROBE interrupt on rise
; 6 STAI 1 Interrupts armed
; 5 CWM 0 Normal outputs
; 4 HNDS 0 No handshake
; 3 GIN 0
; 2 PLS 0 STRB not used
; 1 EGA 1 STAF set on rise of READY
; 0 INVBO STRB not used
Init sei ; Make this atomic
ldaa #$80 ;PC7 is an output
staa DDRC ;PC6-0 inputs
ldaa #$42
staa PIOC

Printer Interface Using IRQ Interrupts
(See Figures 4.27, 4.28, 4.29, and 4.30)

Printer Interface Helper Routines in C
// PC6-PC0 outputs = printer DATA
// STRA=READY interrupt on rise
// STRB=START pulse out
unsigned char OK; // 0=busy, 1=done
unsigned char Line[20]; //ASCII data
unsigned char *Pt; // pointer to line
void Fill(unsigned char *p){
    Pt=&Line[0];
    while((*Pt++)=(*p++)); // copy
    Pt=&Line[0]; // initialize pointer
    OK=0;
}
unsigned char Get(void){
    return(*Pt++);}

Interrupting Keyboard Software in Assembly
ldaa PIOC ; clears STAF
ldaa PORTCL
clr PORTC ;Make PC7=0
jsr InitFifo
cli ; Enable IRQ
rts
ExtHan ldaa PIOC ; poll STAF
bmi KeyHan
swi ; error
KeyHan ldaa PORTCL ; clear STAF
jsr PutFifo
rti
org $FFF2 ; IRQ external
fdb ExtHan

Printer Interface Helper Routines in C
Printer Interface Helper Routines in Assembly

; ***** goes in RAM ***************
OK rmb 1 ; 0 = busy, 1 = done
Line rmb 20 ; ASCII, end with 0
Pt rmb 2 ; pointer to Line
; ***** goes in ROM ***************
; Input RegX => string
Fill ld y # Line ; RegX => string
sty Pt ; initialize pointer
Floop ld aa 0, X ; copy data
staa 0, Y
inx
iny

Printer Interface in C

void Init(unsigned char *thePt){
 asm(“sei”); // make atomic
 Fill(thePt); // copy data into global
 DDRC=0xFF; // Port C outputs
 PIOC=0x5E; // arm out handshake
 PORTCL=Get(); // start first
 asm(“cli”);}
#pragma interrupt_handler ExtHan()
void ExtHan(void){unsigned char data;
 if((PIOC & STAF)==0)asm(“swi”);
 if(data=Get())
 PORTCL=data; // start next
 else{
 PIOC=0x1E; // disarm
 OK=1;}} // line complete

Printer Interface Initialization Routines in Assembly

; PC6-PC0 outputs = printer DATA
; STRA=READY interrupt on rise
; Input RegX => string
Init sei ; Make this atomic
bsr Fill ; Init global
ld aa #$FF ; PC7 is an output
staa DDRC ; PC6-0 outputs
ld aa #$5E
staa PIOC
bsr Get ; start first
staa PORTCL
cli ; Enable IRQ
rts
**Printer Interface ISR in Assembly**

ExtHan ldax PIOC ;poll STAF?
bmi PrtHan
swi ;error
PrtHan bsr Get
    tsta
    beq Disarm
    staa PORTCL ;start next
    bra Done
Disarm ldx $1E ;STAI=0
    staa PIOC
    inc OK ;line complete
Done rti
    org $FFFA ;IRQ external
    fdb ExtHan

**Power System Interface Using XIRQ**

(See Figure 4.31)

**Power System Interface in C**

/* Power System interface
XIRQ requested on a rise of TooLow
PB0, negative logic pulse, will acknowledge XIRQ
PB1=1 will activate backup power */

`#pragma interrupt_handler PowerLow()
void PowerLow(void){ PORTB=2; PORTB=3; }
void Ritual(void){
    // Port B outputs
    PORTB=0; PORTB=1; // Make XIRQ=1
    asm(" ldx #0x10\n" " tap");
}`

**Power System Interface in Assembly**

RITUAL ldx #0       Backup power initially off
    staa PORTB    Set the flip flop, make XIRQ=1
    ldx #1
    staa PORTB    Ready to receive rising edge
    ldx $10       Enable XIRQ, Disable IRQ
    tap
    rts           Back to main thread
* Software can only enable XIRQ, not disable it.
XIRQHAN ldx #2
    staa PORTB    Enable BackUp, ack XIRQ
    ldx #3
    staa PORTB
    rti
    org $FFFF4
    fdb XIRQHAN XIRQ interrupt vector