Administrivia

- Midterm 1 will be on February 14th.
- Midterm content...
**Introduction**

- Interrupts provide guarantee on response time.
- Interrupts allow response to rare but important events.
- Periodic interrupts used for data acquisition and control.
- Interrupts can provide a way to buffer I/O data.

---

**What are Interrupts?**

- An automatic transfer of software execution in response to hardware that is asynchronous with current software.
- Hardware can be external I/O device or internal event.
- When hardware needs service, it requests an interrupt.
- Calls *interrupt service routine* as a *background thread*.
- Thread is terminated with *rti* instruction.
- Threads may communicate using *FIFO queues* and synchronize using *semaphores*.
- Threads share global variables while *processes* do not.
- Each potential interrupt has separate *arm* bit.
- *Interrupt enable bit*, I, found in condition code.
Wire- or negative-logic interrupt requests:
- Can add additional I/O devices w/o redesigning H/W.
- No limit to number of interrupting I/O devices.
- Microcomputer hardware is simple.

Dedicated edge-triggered interrupt requests:
- Software is simpler, easier to debug, and faster.
- Less coupling between software modules.
- Easier to implement priority.
Interrupt Service Routines (ISR)

- Software executed when hardware requests an interrupt.
- Polled interrupts - one large ISR handles all requests.
- Vectored interrupts - many small, specific ISRs.
- When the device is armed, the I bit is zero, and an interrupt is requested, it is serviced as follows:
  - Execution of main program is suspended.
  - All registers are pushed onto the stack.
  - The ISR, or background thread, is executed.
  - The ISR executes rti instruction.
  - All registers are restored from the stack.
  - The main program is resumed.

Interrupt Execution

![Diagram showing the flow of execution during an interrupt]

- Hardware
- Main thread
- Interrupt thread
- Time
- Hardware needs service
- Saves execution state
- ISR provides service
- Restores execution state
When to Use Interrupts

<table>
<thead>
<tr>
<th>Gadfly</th>
<th>Interrupts</th>
<th>DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictable</td>
<td>Variable arrival times</td>
<td>Low latency</td>
</tr>
<tr>
<td>Simple I/O</td>
<td>Complex I/O</td>
<td>High bandwidth</td>
</tr>
<tr>
<td>Fixed load</td>
<td>Variable load</td>
<td></td>
</tr>
<tr>
<td>Single thread</td>
<td>Multithread</td>
<td></td>
</tr>
<tr>
<td>Nothing else to do</td>
<td>Infrequent alarms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Program errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Overflow, illegal op</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Illegal memory access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Machine/memory errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Power failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real-time clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data acquisition/control</td>
<td></td>
</tr>
</tbody>
</table>

Scott R. Little  (Lecture 8: Interrupts)   ECE/CS 5780/6780   9 / 22

Interthread Communication

- Interrupt threads have logically separate registers/stack, so communication must occur through global memory.

Scott R. Little  (Lecture 8: Interrupts)   ECE/CS 5780/6780   10 / 22
Input Device Interrupts

Input device
Interrupt service routine
Main program
Elements in FIFO

Busy | Done | Busy | Done
a | c | d | a
Empty | Empty | Empty | Empty

a) main program waits because FIFO is empty
b) ISR reads data and puts into FIFO
c) main program gets data from FIFO
d) main program processes data

Output Device Interrupts

Output device
Interrupt service routine
Main program
Elements in FIFO

Disarmed | Done | Busy | Done | Disarmed | Done | Busy
a | a | b | a | b | a | b
Empty | 1 | Empty | 1 | Empty | 1 | Empty

a) main program generates data
b) main program puts data into FIFO, arms
b) ISR gets from FIFO and writes to port
c) ISR disarms itself because FIFO is empty
Other Interrupt Issues

- Periodic interrupts are essential for implementing data acquisition and control systems.
- ISR should only occur when needed, come in clean, perform function, and return right away.
- Gadfly loops and iterations should be avoided in ISRs.
- Percent of time in ISRs should be minimized.
- Interface latency is time between new input available and when software reads the input data.
- Device latency is response time of external I/O device.
- A real-time system guarantees a bound on interface latency.

Reentrant Programming

- A program segment is reentrant if it can be concurrently executed by two (or more) threads.
- Reentrant software must place local variables on stack.
- A nonreentrant subroutine has a section of code called a vulnerable window or critical section, and error occurs if:
  - One thread calls a nonreentrant subroutine,
  - A thread is executing in the critical section when interrupted.
  - Second thread calls same sub or shared variable.
- Mutual exclusion is often implemented by disabling interrupts.
Nonrentrant Subroutine in C

```c
int Result; /* Temporary global variable */
int Ave(int x,y){
    Result = y; /* Save second number */
    Result = (Result + x) >> 1; /* (1st+2nd)/2 */
    return(Result);
}
```

Atomic Operations

- Atomic operation is one that once started is guaranteed to finish.
- In most computers, machine instructions are atomic.
- The following is atomic:
  ```
  inc counter       where counter is global variable
  ```
- The following is nonatomic:
  ```
  ldaa counter     where counter is global variable
  inca
  staa counter
  ```
Read-Modify-Write Example

- Software reads global variable, producing a copy of the data.
- Software modifies the copy.
- Software writes modification back into global variable.

```c
unsigned int Money; /* bank balance (global) */
/* add 100 dollars */
void more(void){
    Money += 100;
}
```

Money rmb 2 bank balance implemented as a global
* add 100 dollars to the account
more ldd Money where Money is a global variable
    addd #100
    std Money Money=Money+100
rts

Write Followed by Read Example

- Software writes to a global variable.
- Software reads from global variable expecting original data.

```c
int temp; /* global temporary */
/* calculate x+2*d */
int mac(int x, int d){
    temp = x+2*d; /* write to a global variable */
    return (temp); /* read from global */
}
```

temp rmb 2 global temporary result
* calculate RegX=RegX+2*RegD
mac stx temp Save X so that it can be added
    lsld RegD=2*RegD
    addd temp RegD=RegX+2*RegD
    xgdx RegX=RegX+2*RegD
rts
Nonatomic Multistep Write

- Software write part of new value to a global variable.
- Software write rest of new value to a global variable.

```c
int info[2]; /* 32-bit global */
void set(int x, int y){
    info[0]=x;
    info[1]=y;}
```

Info rmb 4 32-bit data implemented as a global
* set the variable using RegX and RegY
set stx Info Info is a 32 bit global variable
sty Info+2
rts

Disabling Interrupts in C

```c
int Empty; /* -1 means empty, 0 means it contains someth
int Message; /* data to be communicated */
int SEND(int data){ int OK;
    char SaveSP;
    asm tpa
    asm staa SaveSP
    asm sei   /* make atomic, entering critical */
    OK=0;    /* Assume it is not OK */
    if(Empty){
        Message=data;
        Empty=0;  /* signify it is now contains a message*.
        OK=-1;}   /* Successfull */
    asm ldaa SaveSP
    asm tap    /* end critical section */
    return(OK);}
```
A Binary Semaphore

* Global parameter: Semi4 is the mem loc to test and set
* If the location is zero, it will set it (make it -1)
  and return Reg CC (Z bit) is 1 for OK
* If location is nonzero, return Reg CC (Z bit) = 0
Semi4 fcb 0 Semaphore is initially free
Tas tst Semi4 check if already set
  bne Out busy, operation failed, return Z=0
  dec Semi4 signify it is now busy
  bita #0 operation successful, return Z=1
Out rts

Reentrant or Not?

- Must be able to recognize potential sources of bugs due to nonreentrant code in high-level languages.
- Is the following atomic?
  time++;
- Yes, if the compiler generates:
  inc time
- No, if the compiler generates:
  ldd time
  addd #1
  std time