Today's plan:

- Announcements
- General Strategy
- Microcontroller programming concepts/last bits of assembly
- Activity 2

- Intro to programming in C – time permitting

Announcements:

Turn in Lab notes and annotated programs on Connect before Start of Lab 3.

Lab 1&2 Marking scheme:

Successful completion of required lab activities.	2
Completeness and quality of procedure/circuits/code etc. used to perform the activities	10
General Report Structure: objectives defined, clear enough statements of what is being done and why, what the results were. General understandability	2
Above and beyond. Evidence of exploration above and beyond the specific tasks requested in the manual.	1

The remaining labs will have similar schemes. Labs 1&2 will be weighted less heavily than the others.

General Strategy

Documentation: read the manual!
- not always so easy. Which manual?

There are two major documents relevant for the microcontroller:
- Family reference guide (slau144) [eg cpu instructions]
- chip data sheet (slas735) [eg what pin can do what]

Table of contents
Keyword searching

Also:
- Course lab manual – general instruction, what tasks are required
- OS specific set-up/user guide – how to set up computers, what command to type to compile/assemble programs and load on to Launchpad board.

Stack and Stack Pointer

- A part of the RAM usually starting at the top (highest address 0x400) of the available RAM used to store values which we will use later (PUSH, PUSH.b and POP, POP.b commands) or for storing values of registers during subroutines and interrupts.

- The size of the RAM limits the number of nested subroutines and interrupts – we can not allow the area where we keep the variables to overlap with the stack. This is a common cause of “crashing” the computer or microprocessor.
- Stack pointer (SP register) indicates the lowest occupied position in the stack

```
#include "msp430g2553.inc"
org 0xf800
RESET:
        mov.w #0x0400, sp
        mov.w #WDTPW|WDTHOLD,&WDTCTL
        mov.b #11110111b, &P1DIR
        mov.b #01000001b, &P1OUT
        mov.b #00001000b, &P1IE
        mov.w #0x0041, R7
        mov.b R7, &P1OUT
        EINT
        bis.w #CPUOFF,SR
PUSH:
        xor.w #0000000001000001b, R7
        mov.b R7,&P1OUT
        bic.b #00001000b, &P1IFG
        reti
org 0xffe4
dw PUSH
org 0xfffe
dw RESET
```

Interrupts

- Are triggered by an external event (eg an input going low, a timer overflowing, number of counts exceeding some preset value etc.)

- PC and SR are saved on the stack so that regular flow can resume when the interrupt finishes.

Interrupts

- When interrupt occurs the current microprocessor's activity stops and the interrupt service routine (ISR) is started
- The address of the ISR has to be stored in the specific location in the memory. This address is called an interrupt vector. They are located in the flash memory area 0xFFE0 to 0xFFFF. The addresses of the interrupt vectors are listed in the msp430g2553 manual.

```
PUSH:
        xor.w #0000000001000001b, R7
        mov.b R7,&P1OUT
        bic.b #00001000b, &P1IFG
        reti
org 0xffe4
dw PUSH
org 0xfffe
dw RESET
```

Interrupts

- The event setting an interrupt is in fact setting a bit in a specific register. This bit is called an interrupt flag. For example a PORT1.3 interrupt, sets bit 3 in the P1IFG register at the address 0x026. The ISR must clear this flag!

- Each interrupt service routine has to end with RETI command.

```
PUSH:
        xor.w #0000000001000001b, R7
        mov.b R7,&P1OUT
        bic.b #00001000b, &P1IFG
        reti
org 0xffe4
dw PUSH
org 0xfffe
dw RESET
```

Interrupts

- When the interrupt occurs:
  - status register and program counter are pushed onto the stack.
  - the program counter is loaded from the interrupt vector
  - the ISR (pointed to by the interrupt vector) executes
  - the ISR must clear the interrupt flag [some clear themselves]
  - reti pops the status register and program counter off the stack so the program can continue

  Save/restore registers? In assembly you need to worry about this yourself. In a C program, this is handled for you.

```
#include "msp430g2553.inc"
org 0xf800
RESET:
        mov.w #0x400, sp
        mov.w #WDTPW|WDTHOLD,&WDTCTL
        mov.b #11110111b, &P1DIR
        mov.b #01000001b, &P1OUT
        mov.b #00001000b, &P1IE
        mov.w #0x0041, R7
        mov.b R7, &P1OUT
        EINT
        bis.w #CPUOFF,SR
PUSH:
        xor.w #0000000001000001b, R7
        mov.b R7,&P1OUT
        bic.b #00001000b, &P1IFG
        reti
org 0xffe4
dw PUSH
org 0xfffe
dw RESET
```

Enable pin P1.3 to produce interrupts

Global enable interrupts (DINT will disable)

Why program in Assembly?

Full control of every detail of program flow and memory organization

Speed!

Smallest, most compact programs

Why program in Assembly?

Full control of every detail of program flow and memory organization

Speed!

Smallest, most compact programs

Why not?

Need to take care of every detail

Hard to debug

Tedious

Instruction set is CPU specific