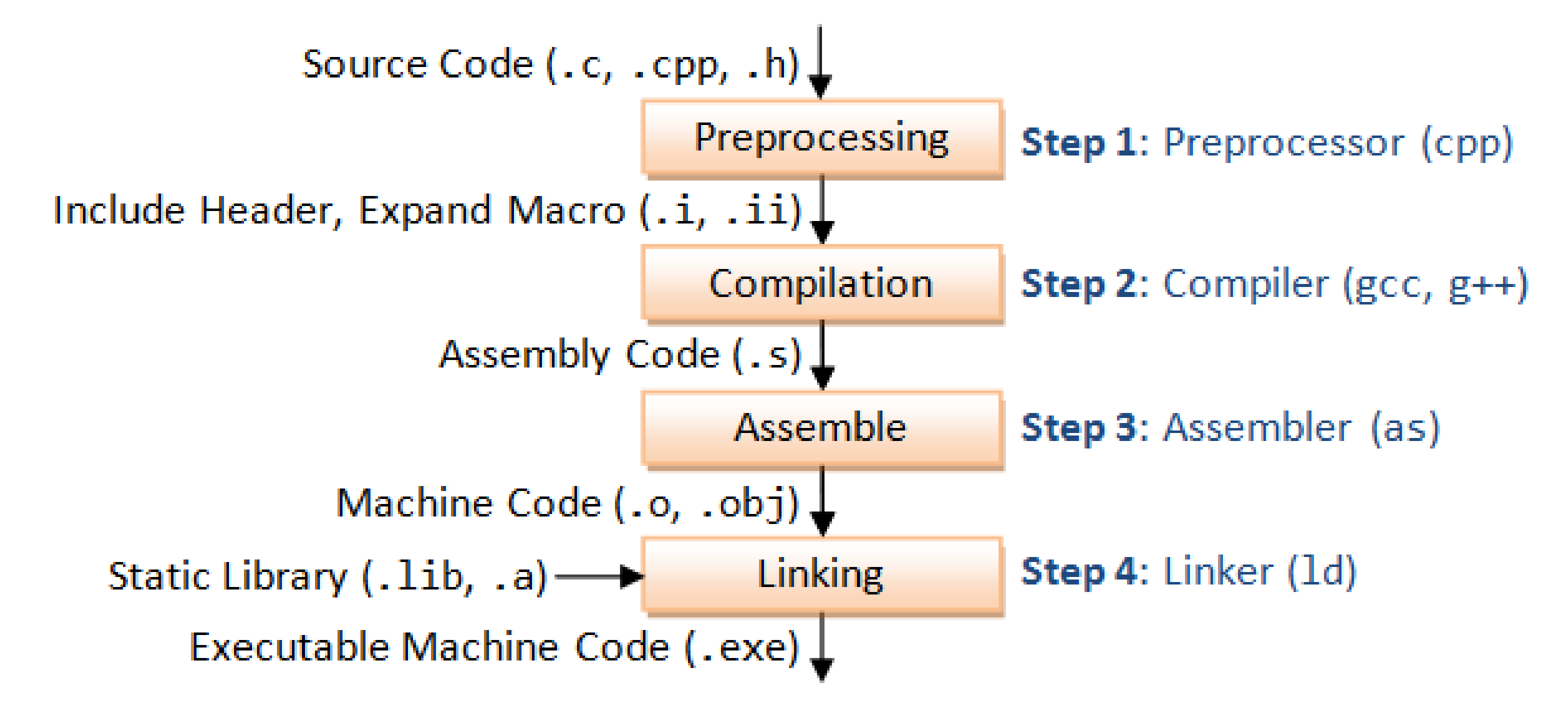
Embedded Systems

Introduction to Embedded Programming

UNC CHARLOTTE

The Compilation Process



UNC CHARLOTTE

Source File

```
#include <msp430.h>
#define SHIFT_ME 3
#define LOOP_FOREVER() while(1);

void delay(unsigned int x) {
	while(x--);
}
void main (void) {
	unsigned int xx = 100%2 << SHIFT_ME;
	delay(xx);
	LOOP_FOREVER();
}
```

UNC CHARLOTTE

The Preprocessor

- The pre-processor processes the source code before it continues with the compilation stage.
- The pre-processor
  - Resolves #define statements (constants, variable types, macros)
  - Concatenates #include files and source file into one large file
  - Processes #ifdef - #endif statements
  - Processes #if - #endif statements
- Specifically for embedded systems the pre-processor also processes vendor-specific directives (non-ANSI)
  - #pragma

UNC CHARLOTTE

Source After Preprocessing

… the msp430.h file …

```
void delay(unsigned int x) {
	while(x--);
}
void main (void) {
	unsigned int xx = 100%2 << 3;
	delay(xx);
	while(1);
}
```

UNC CHARLOTTE

The Compiler

- The compiler turns source code into machine code packaged in object files.
  - Common file format are object file format (COFF) or the extended linker format (ELF).
- A cross-compiler produces object files that will then be linked for the target instead of the computer running the compiler (compare Linux, embedded Linux, MSP430)
- Practical approach in embedded systems: Turn off compiler optimization!
  - Should be off by default

UNC CHARLOTTE

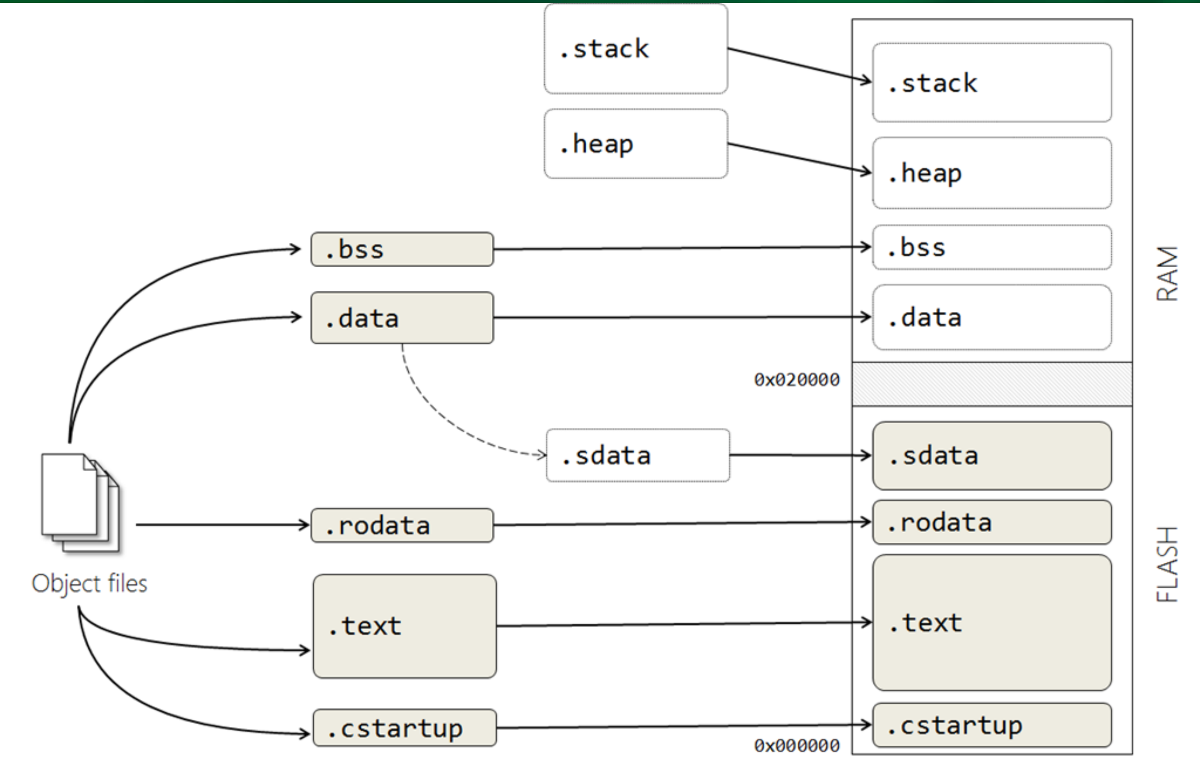
The Assembler

- It takes the assembly source code and produces an assembly listing with offsets
- The assembler output is stored in an object file

UNC CHARLOTTE

The Linker

- Linker scripts are text files. The main purpose of the linker script is to describe how the sections in the input files should be mapped into the output file, and to control the memory layout of the output file



The Linker

- The linker performs the following
  - It combines object files by merging object file sections.
    - .text section for code
    - .data section for initialized global variables
    - .bss section for uninitialized global variables
  - It resolves all unresolved symbols.
    - External variables
    - Function calls
  - Reports errors about unresolved symbols.
  - Appends the start-up code (see next slide)
    - Provide symbolic debug information

The Linker - Startup Code

- Startup is a small fragment of assembly code that prepares the machine for executing a program written in a high-level language.
  - For embedded C it is typically also an object file specified in the linker script.
- Tasks of the startup code
  - Disable all interrupts
  - Initialize stack pointers for software stack
  - Initialize idata sections
  - Zero all uninitialized data areas in data memory (ANSI standard)
  - Call loop: main(); goto loop;

UNC CHARLOTTE

Example

- While writing a C-program its a basic rule to write a main() in it as its said that the execution begins at the main()
- But how the control comes to main()?
- Example Source Code:

```
int i=10;
int j;
main()
{
}
```

- How do we get to main?

UNC CHARLOTTE

Example – Object File

```
./a.out:     file format elf32-msp430


Disassembly of section .text:

0000f800 <__init_stack>:
    f800:       31 40 80 02     mov     #640,   r1      ;#0x0280

0000f804 <__low_level_init>:
    f804:       b2 40 80 5a     mov     #23168, &0x0120 ;#0x5a80
    f808:       20 01

0000f80a <__do_copy_data>:
    f80a:       3f 40 02 00     mov     #2,     r15     ;#0x0002
    f80e:       0f 93           tst     r15
    f810:       05 24           jz      $+12            ;abs 0xf81c
    f812:       2f 83           decd    r15
    f814:       9f 4f 42 f8     mov     -1982(r15),512(r15);0xf842(r15), 0x0200(r15)
    f818:       00 02
    f81a:       fb 23           jnz     $-8             ;abs 0xf812

0000f81c <__do_clear_bss>:
    f81c:       3f 40 02 00     mov     #2,     r15     ;#0x0002
    f820:       0f 93           tst     r15
    f822:       04 24           jz      $+10            ;abs 0xf82c
    f824:       1f 83           dec     r15
    f826:       cf 43 02 02     mov.b   #0,     514(r15);r3 As==00, 0x0202(r15)
    f82a:       fc 23           jnz     $-6             ;abs 0xf824

0000f82c <__jump_to_main>:
    f82c:       30 40 34 f8     br      #0xf834
```

UNC CHARLOTTE

Example – Object File

```
0000f830 <__ctors_end>:
    f830:       30 40 3e f8     br      #0xf83e

0000f834 <main>:
    f834:       31 40 80 02     mov     #640,   r1          ;#0x0280
    f838:       04 41           mov     r1,     r4
    f83a:       30 40 40 f8     br      #0xf840

0000f83e <_unexpected_>:
    f83e:       00 13           reti

0000f840 <__stop_progExec__>:
    f840:       ff 3f           jmp     $+0                 ;abs 0xf840

Disassembly of section .data:

00000200 <__data_start>:
 200:   0a 00           .word   0x000a; ????

Disassembly of section .bss:

00000202 <__bss_start>:
        ...
```

UNC CHARLOTTE

Example – Object File

```
Disassembly of section .vectors:

0000ffe0 <InterruptVectors>:
    ffe0:       30 f8           interrupt service routine at 0xf830
    ffe2:       30 f8           interrupt service routine at 0xf830
    ffe4:       30 f8           interrupt service routine at 0xf830
    ffe6:       30 f8           interrupt service routine at 0xf830
    ffe8:       30 f8           interrupt service routine at 0xf830
    ffea:       30 f8           interrupt service routine at 0xf830
    ffec:       30 f8           interrupt service routine at 0xf830
    ffee:       30 f8           interrupt service routine at 0xf830
    fff0:       30 f8           interrupt service routine at 0xf830
    fff2:       30 f8           interrupt service routine at 0xf830
    fff4:       30 f8           interrupt service routine at 0xf830
    fff6:       30 f8           interrupt service routine at 0xf830
    fff8:       30 f8           interrupt service routine at 0xf830
    fffa:       30 f8           interrupt service routine at 0xf830
    fffc:       30 f8           interrupt service routine at 0xf830
    fffe:       00 f8           interrupt service routine at 0xf800
```

UNC CHARLOTTE

Example – Object File Explained

- The CPU starts program execution at the word address stored in the reset vector, 0xFFFEh
- _init_stack : which initialize the stack pointer to 0x280
  - Notice that RAM address space starts at 0x27F in memory map from previous slides
- _low_level_init : Here the WDT(Watch Dog Timer) is initialized and holded
- _do_copy_data : Here the value of the initialized global variable is copied to the RAM from 0x200(_data_start Section)
- **_do_clear_bss** : Here the uninitialized global variables are initialized to zero and stored in the RAM. This is stored in the _bss_start Section

MATERIAL SOURCE: http://remyaraj89.wordpress.com/2010/11/25/a-c-startup-code-of-msp430-and-a-linker-script-to-link-it/

UNC CHARLOTTE

Writing a Basic Program

- C programs begin with a main() function
- Generally, embedded system programs will run inside of an infinite loop
  - While (1); for(;;);

```
void main(void) {
    // code here
    while(1){
            // code here
    }
}
```

Variables and Data Types

- Variables are used for storing data
- Declared with type first, then name, then value (optional)
  - int var = 1;
    - Reserves psace for an integer and fills it with 0x0001
  - int var;
    - Reserves space for an integer, but does not assign value
    - Could contain junk information
- Variable types have different sizes in memory
- Variable size is different for different machines

UNC CHARLOTTE

Variable Types

Table 5-1. MSP430 C/C++ Data Types

Type	Size	Representation	Range Minimum	Range Maximum
char, signed char	8 bits	ASCII	-128	-127
unsigned char, bool	8 bits	ASCII	0	255
short, signed short	16 bits	2s complement	-32 768	32 767
unsigned short, wchar_t	16 bits	Binary	0	65 535
int, signed int	16 bits	2s complement	-32 768	32 767
unsigned int	16 bits	Binary	0	65 535
long, signed long	32 bits	2s complement	-2 147 483 648	2 147 483 647
unsigned long	32 bits	Binary	0	4 294 967 295
enum	16 bits	2s complement	-32 768	32 767
float	32 bits	IEEE 32-bit	1.175 495e-38[1]	3.40 282 35e+38
double	32 bits	IEEE 32-bit	1.175 495e-38[1]	3.40 282 35e+38
long double	32 bits	IEEE 32-bit	1.175 495e-308[1]	3.40 282 35e+38
pointers, references, pointer to data members	16 bits	Binary	0	0xFFFF

UNC CHARLOTTE

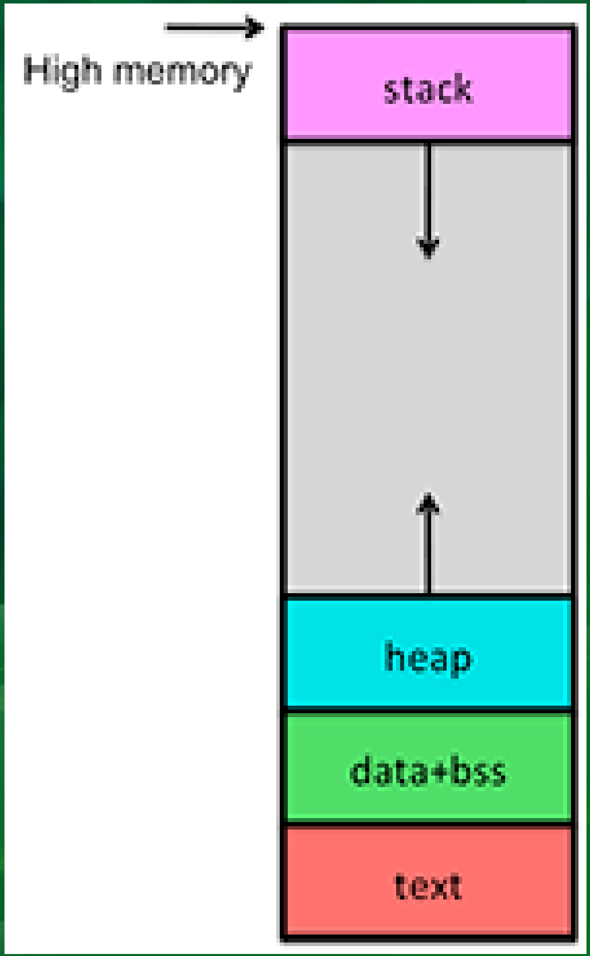
Declaring Variables

- Variable names are lowercase by convention
- Variable names cannot begin with numbers or special characters except for underscore
  - var
  - _var
- Variable names cannot be keywords
  - while
  - do

UNC CHARLOTTE

Declaring Variables

- Where you declare variables in your code will determine where they are stored in memory
- In embedded systems, it is important to keep in mind how much space your variables are occupying, where they are located, and what you are actually writing to that space



UNC CHARLOTTE

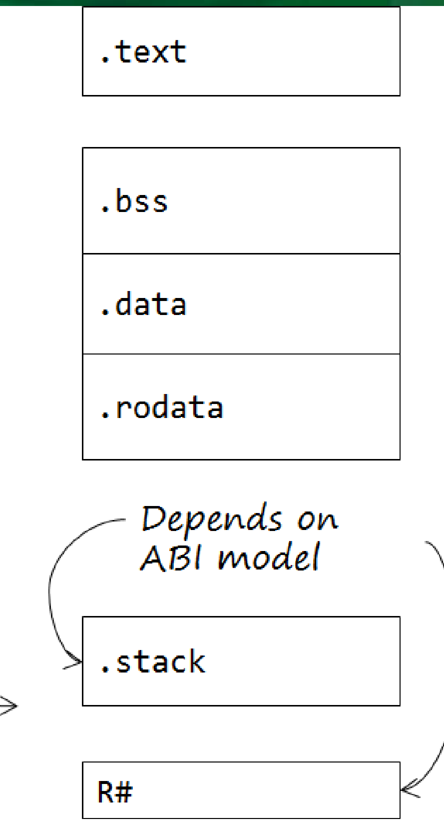
Declaring Variables

- Where you declare variables determines the “scope of the variable”
- “Global” variables are declared outside of functions and are available to all functions inside the same .c file
- “Local” variables are declared inside of functions, and are only available within that function
  - If a function containing a local variable returns (finishes), then that variable disappears

Local Variables

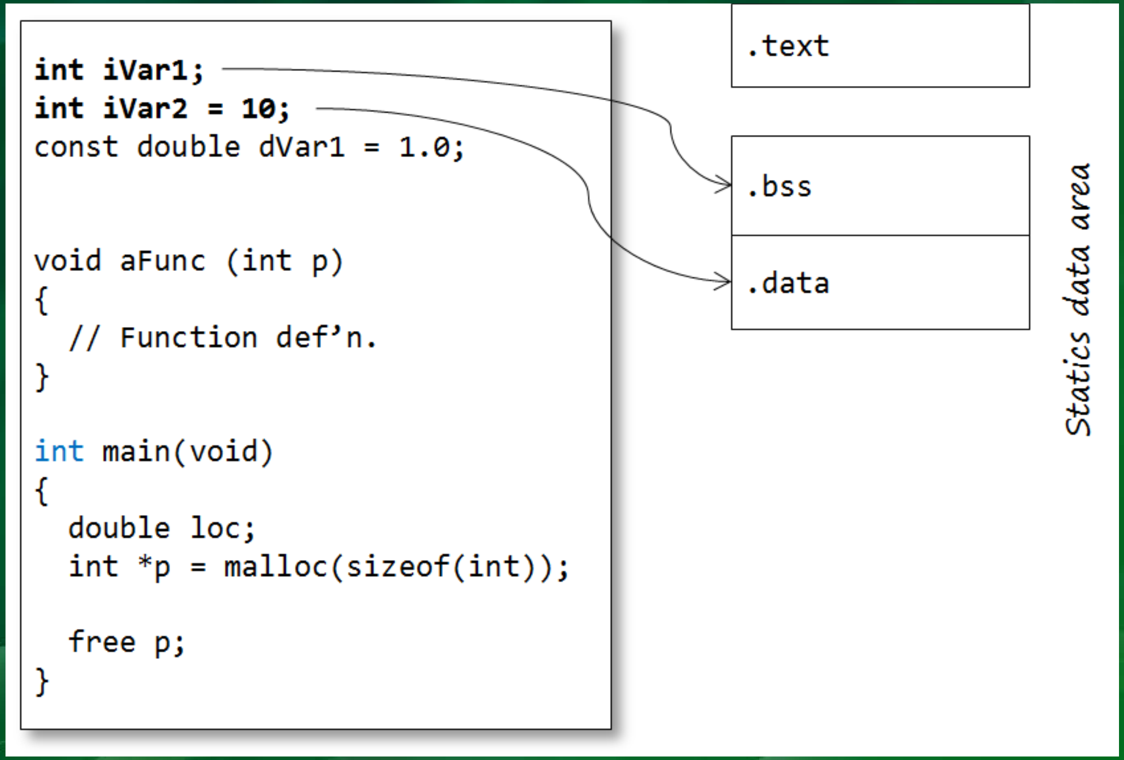
- Stored on the stack
- Only accessible within that function

```c
int iVar1;
int iVar2 = 10;
const double dVar1 = 1.0;


void aFunc (int p)
{
  // Function def'n.
}

int main(void)
{
  double loc;
  int *p = malloc(sizeof(int));

  free p;
}
```



Global Variables

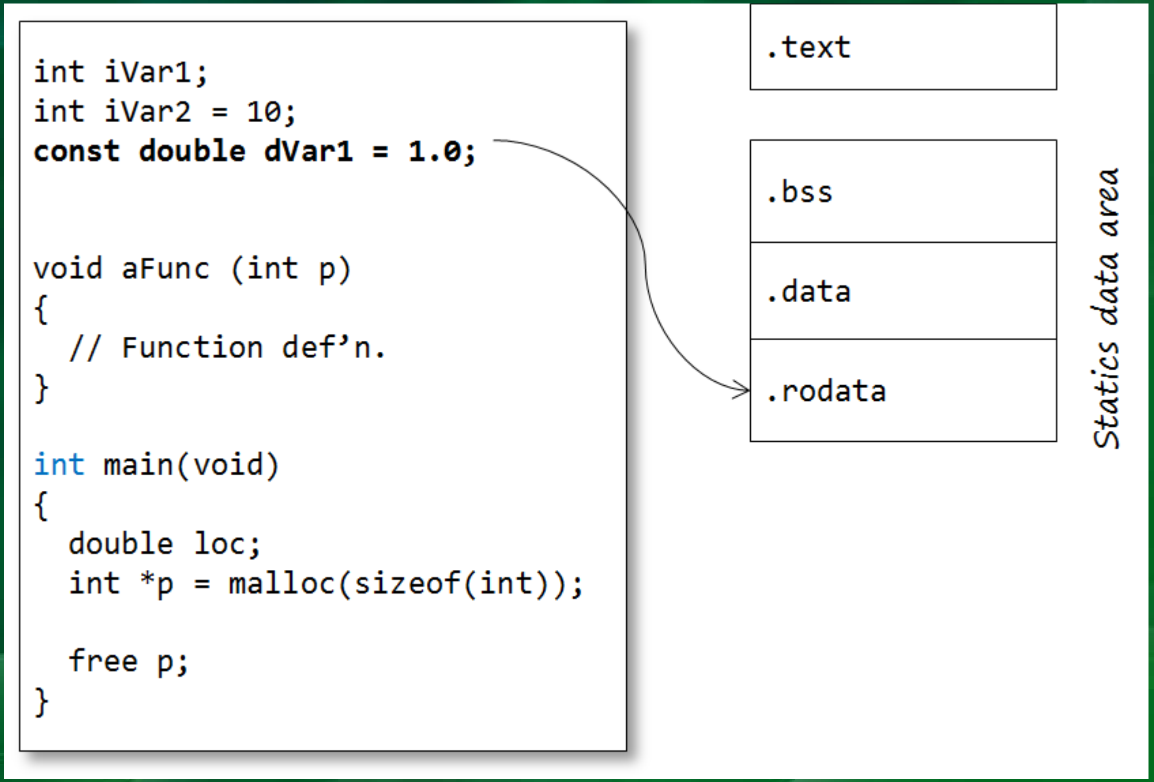
- Global variables are stored in .bss
- Accessible throughout the entire .c file



UNC CHARLOTTE

Constant Variables

- Constant variables are stored in the .data or .rodata section



SOURCE MATERIAL : http://blog.feabhas.com/2012/06/the-c-build-process/

UNC CHARLOTTE

Reading and Setting I/O Pins

- GPIO have 4 registers
  - Data Direction Registers (PXDIR)
    - Determine input or output
    - 1 is output, 0 is input
  - PXOUT
    - Sets output pins high or low
  - PXIN
    - Reads input pin values
  - PXREN
    - Enables pull-up resistors on input pins

UNC CHARLOTTE

Example Code

```
#include <msp430.h>

int main(void) {
        WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer
        P1DIR |= 0x01;                      // Set P1.0 to output direction
        P1REN |= 0x08;                      // Enable P1.3 pull-up resistors

        for(;;) {
                if( !(P1IN & 0x08) )     // Mask P1.3 to get button state
                        P1OUT |= 0x01;      // Set P1.0 high
                else
                        P1OUT &= 0xFE;      // Set P1.0 low

        }

        return 0;
}
```

UNC CHARLOTTE

Bit Masking

- Used to isolate individual bits from a register
- Example: reading pin 3 from port 1
  - P1IN [01001001] <= Port to read from
  - And 00001000
  - 00001000
  - (P1IN & 0x08) <= bit mask for pin 3
  - (P1IN & (1<<3)) <= bit mask by shifting 1 up 3 positions

UNC CHARLOTTE

Bit Masking by Shifting

- (1 << 3)
  - 0000 0001 => 0000 1000
  - Used to make masking more readable
  - (1 << PIN3)
    - Sometimes individual bits in a register are #defined in a header file to make masking easier

UNC CHARLOTTE

Using Or-Equals and And-Equals

- Use |= and &= to mask without disturbing the other bits in a register
- To "turn on" pin 0 without overwriting the other pins, use: P1OUT |= 0x01
- To "turn off" pin 0 without clearing other pins use: P1OUT &= 0xFE
  - PORT1: 0010 01011
  - & 1111 11110
  - 0010 01010

UNC CHARLOTTE