ECE 131 – Programming Fundamentals

Dr. Daryl Lee

University of New Mexico

- Until now, we have been treating data as numbers or characters.
- Until now, all data has been contained in memory.
- One of C's great applications is in the control of hardware. Consider the following excerpt from the data sheet of the PIC12F629, a product in the Microchip family of microprocessors:

Bit Manipulation

- This says that the "oscillator calibration" value is a 6-bit number, but it is stored in bits 2-7, not bits 0-5

REGISTER 2-7: OSCCAL: OSCILLATOR CALIBRATION REGISTER (ADDRESS: 90h)

R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0
CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	—	—
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-2 **CAL5:CAL0:** 6-bit Signed Oscillator Calibration bits
`111111` = Maximum frequency
`100000` = Center frequency
`000000` = Minimum frequency

bit 1-0 **Unimplemented:** Read as '0'

- Or, consider this excerpt from the same document:

THE UNIVERSITY of NEW MEXICO

Bit Manipulation

- This says that the “GPIO” consists of individual bits, not a “number”

REGISTER 3-1: GPIO: GPIO REGISTER (ADDRESS: 05h)

U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 7-6	**Unimplemented**: Read as ‘0’
bit 5-0	**GPIO<5:0>**: General Purpose I/O pin 1 = Port pin is $>V_{IH}$ 0 = Port pin is $<V_{IL}$

- C does not have single-bit capability; how do we handle such data?

Bitwise combinations

- Bitwise combinatorial operators

```
char x = 0x15; // 00010101
char y = 0x17; // 00010111
char z;
z = x & y; // AND: 00010101
z = x | y; // OR: 00010111
z = x ^ y; // XOR: 00000010
z = ~x; // NOT: 11101010
```

THE UNIVERSITY of NEW MEXICO

2's Complement arithmetic

- MSB $= 0$ for values $>= 0$, 1 for values < 0
- To negate a number, negate each bit and add 1 to result:

```
char x = 0x15; // 00010101
char z;
z = -x; // 11101010 + 1 = 11101011
```

- There is such a thing as 1's complement arithmetic, where negation of a number is just flipping of all bits. The result is that there are two representations of zero.

Shift

- Signed and unsigned

```
char sr, x = 0x85; // 10000101
unsigned char ur, y = 0x85; // 10000101
sr = x << 2; // SHIFT LEFT: 00010100
ur = x >> 2; // SHIFT RIGHT, UNSIGNED 00100001
sr = x >> 2; // SHIFT RIGHT, SIGNED 11100001
```

THE UNIVERSITY of NEW MEXICO

Usage Example

Suppose we want to set a value into the OSCCAL register given earlier:

```
// OSCCAL register is at address 0x90
char *OSCCAL = 0x90;
char OSCCALValue = 13;
char tmp = *OSCCAL; // Get current register
tmp &= 0x03; // Save two LSBs
// Shift value and OR into place
tmp |= (OSCCALValue << 2);
*OSCCAL = tmp;
```

THE UNIVERSITY of NEW MEXICO

Using Bitfields

- If that's such a common thing to do, might there be a better way?

```
typedef struct {
   int unused :  2;
   int value :  6;
} OSCCALReg;

// OSSCAL register is at address 0x90
OSCCALReg *OSCCAL = 0x90;
OSCCAL->value = 13;
```

- Bits are allocated starting from the LSB.

Unions

Wouldn't it be nice to view the same data different ways?

```
typedef struct {
   int unused :  2;
   int value :  6;
} OSCCALReg;

typedef union {
   char asByte;
   OSCCALReg asReg;
} OSCCALVar;

// OSSCAL register is at address 0x90
OSCCALVar *OSCCAL = 0x90;
OSCCAL->asReg.value = 13;
printf("Full register = %02x", OSCCAL->asByte);
```