Module 6. Review: The Story So Far

Writing Numbers

Be comfortable with standard conventions for writing numbers used in class and in C:

- Decimal: 42
- Hex: 0x2A or 2Ah
- Binary: 0010 1010b

You should be able to convert from binary to hex easily (and vice versa)!

Basic C Instructions and Syntax:

>> Know layout of C source file (Lecture 2)
>> Some Data types (as they are defined in CSS for the MSP430)

```
// What are the sizes for each datatype?
    int             a;        //
    float           b;        //
    char            c;        //
    unsigned int    d;        //
    long int        e;        //
    double          f;        //
    int         arr[5];       //
```

Arrays: Are blocks of memory where multiple values are stored contiguously. Storing elements successively (in order) makes it easy to access each element given its index.

Standard C Operators:

Math: `+` `-` `*` `/` `=` `%` (modulo)
Unary: `++` `--` (also `|=` `&=` `+=` etc.)
Relational and Logical: `>` `>=` `<` `<=` `==` `!=` `&&` `||`
Bitwise: `&` (AND) `|` (OR) `^` (XOR) `>>` (R shift) `<<` (L shift) `~` (NOT)

Quick Questions:

```
int a = 0x0101;
int w = a + 12;
int x = a << 1;

unsigned char b = 0xff;
unsigned char y = b + 2;

int d = 42;
int z = d / 10;
```

1) What value is assigned to x?
a) 0x0202 b) 0x1010 c) 0x2020 c) 0x0080

2) What value is assigned to y?
a) -1 b) 0 c) 1 d) 256

3) What value is assigned to z?
a) 2 b) 4 c) 4.2 d) 10

Decisions, looping, etc:

```
if  (kk > 100) {
    kk = 0;
 } else {
    z = 2*z+kk;
    kk++;
}

while  (j < 100) {
    /* Body of  loop  */
    j++;
 }

for (i = strt; i < end_pt; i++) {
  /* Body of loop. Do something  */
}
```

--> The ***"Forever Loop"***

```
while  (1) {
    /* Body of loop. Do something */
}
```

Basic Structure of a C program

```
#define  MAX_SZ  100;

 // Determines max value of an array
 unsigned int arrayMax(unsigned int* in_arr, int num_pts);

 void main()
 {
     unsigned int     big[MAX_SIZ];
     unsigned int     maximum=0;
     unsigned int     i, other_val;

     /* Do some stuff */
    i = 0;
     while (i < MAX_SZ)
    {
        big[i] = (i % 10);
        i++;
    }
     maximum = arrayMax(big, MAX_SZ);
    /* Do more stuff */

 }  // end of main()
```

Quick Questions:

1) How many times does the while loop execute?

a) 99 b) 100 c) 101

2) To what value is big[47] assigned?

a) 40 b) 0.47 c) 7 d) 470

3) What is the range of *valid* indices for the big array?

a) big[1] to big[100] b) big[0] to big[99]
c) big[0] to big[100] d) big[0] to big[9]

4) To what value is maximum assigned?

a) 99 b) 100 c) 10 d) 9

<u>Data Representations (HW #1):</u>

>> Integer representations:

--Unsigned, sign-magnitude, two's complement and BCD

>> Expect Conversion Between Bases and Formats!

<u>Unsigned integers</u> = all bits used to convey magnitude (whole numbers) – For *n* bits, values run from 0 to $2^n - 1$ (i.e. *N*=16, 0 to 65535)

1026 = 00000100 00000010b = 0402h

<u>Sign Magnitude integers</u> = *n*-1 bits used to convey magnitude with "most significant bit" or MSB used for sign (0 = +, 1 = -). For *n* bits, values run from $-2^{(n-1)}-1$ to $2^{(n-1)}-1$

1026 = 0000 0100 0000 0010b = 0402h

-1026 = 1000 0100 0000 0010b = 8402h

** Has 2 representations of 0 >>> +0 and -0!

<u>Two's Complement integers</u> = Common format for signed integers (int). For *n* bits, values run from $-2^{(n-1)}$ to $2^{(n-1)}-1$. (i.e. *n*=16, -32768 to 32767). Used by C.

Positive numbers: Same as Unsigned

1026 = 0000 0100 000 00010b = 0402h

Negative numbers (ONLY!!): ***Encode*** magnitude, ***Complement*** each bit, ***Add*** 1

```
-15 = 0000 0000 0000 1111  = 15
      1111 1111 1111 0000     complement
                       +1
      -------------------
      1111 1111 1111 0001  =  0FFF1h  =  -15 in two's complement
```

<u>Binary Coded Decimal</u> = Each decimal digit expressed in binary nibble

367 = 0000 0011 0110 0111b

Fractional Number representations:

Fixed point: Binary radix point assigned a fixed location in byte (or word)

0101.1010 = $5 + 2^{-1} + 2^{-3}$ = 5.625

Precision is function of number of fractional bits assigned
--> 4 fractional bits = 2^(-4) = 0.0625 = smallest fraction

Floating Point (IEEE Standard) : Used to better approximate real valued decimal values to a prescribed number of decimal places

Single Precision (32 bits): S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF

Value = $(-1)^S \; 2^{(E-127)} * (1.F)$

Why are floating point operations computationally expensive?

For the exam, you do not need to remember how to convert to/from floating-point, but you should understand what it is and how it differs from fixed-point.

Character Representations

ASCII: Standard for representing characters in Roman alphabet and some control characters

- You will have an ASCII table on the exam. Know how to read one and when you need it!

Quick Questions:

1) The decimal equivalent of unsigned integer 8002h is
   a) 32770 b) 65538 c) -2 d) 16386

2) The decimal equivalent of two's complement integer 8002h is
   a) -2 b) 32770 c) -32766 d) -65538

3) The decimal equivalent of two's complement integer 0002h is
   a) -2 b) 32770 c) 2 d) -65538

4) The decimal equivalent of BCD integer 8002h is
   a) -2 b) 32770 c) 8002 d) 2008

<u>Little Endian: The MSP430, like Intel processors, is "Little Endian"</u> (HW1)

-- The lower byte of each 16 bit word is stored first then the higher byte
"*Low Byte, High Byte*"

-- For double words the lower word is stored first then the upper word

<u>*Ex*</u>: How 65340 decimal = 00 01 00 04h is stored in memory at address 0400h

Little Endian

Address	***Byte Value***
02403h	00h
02402h	01h
02401h	00h
02400h	04h
....	....

A memory dump from CCS shows contents of addresses from left to right starting at 02400h

`02400h = 04 00 01 00` ... <= Bytes appear "out of order" when read left to right

Big Endian: Many other RISC processors

-- The higher byte (big end) of each 16 bit word is stored first then the lower byte

BIG Endian

Address	***Byte Value***
02403h	04h
02402h	00h
02401h	01h
02400h	00h
....	....

A memory dump from a big endian processor (also left to right)
`02400h= 00 01 00 04`... <= Bytes appear "in order" when read left to right

<u>*Network Byte Order*</u> = BIG ENDIAN!!!

<u>Microprocessor Systems Architecture</u>:

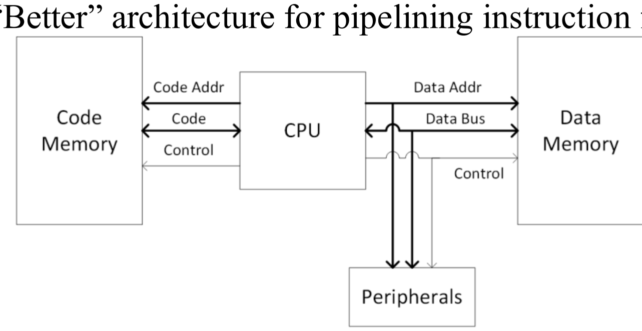
>> General Computing Hardware/Software Hierarchy

Applications
Operating System = User Interface
System SW = Interface to HW
HW Layer = CPU, Mem., peripherals

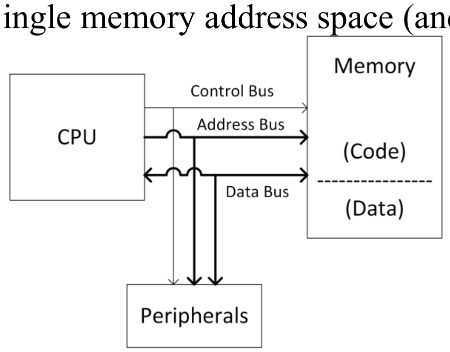
>> Gets "squashed" in an embedded system...

Applications
Maybe some System SW functions
HW Layer = CPU, Mem., peripherals

Harvard Architecture – Separate memory address spaces (and busses) for code and data
("Better" architecture for pipelining instruction fetches)



Von Neumann Architecture – Single memory address space (and bus) for code & data



>> MSP430x55xx uses ***Von Neumann*** architecture

>> We're using ***MSP430F5529***
-- 128 KB Flash memory (code)
-- 8 KB RAM (data) + 2 kB USB RAM
-- LCD controller
-- Hardware multiply, UART, and a slew of other peripherals
(Timers, ADC, comparator, general digital IO ports...)

Memory Organization:

>> Memory = group of sequential locations where binary data is stored

-- In MSP430, a memory location holds 1 byte

-- Each byte has unique address which CPU uses to read to and write from that location
-- Multibyte data is stored Little Endian!

-- *2 types of memory: Volatile and Non-volatile*
RAM = 8KB = DATA memory = Volatile
FLASH = 128KB = CODE memory (primarily!) = Non-volatile

Memory Operations

- Read and Write: retrieving or writing DATA to/from RAM (under programmer control)
- Fetch: retrieving of instruction from CODE (Flash) memory
(automatic CPU function)
>> Flash is NOT byte writable!
-- Must be erased in multi-byte (e.g. 512 byte) segments
>> A flash write cycle takes much longer than read cycle

MSP430 is 16 bit Microcontroller
>> 16 bit word size = 16 bit internal registers
>> Also has 20 bit address bus (can access up to 1 MB = 2^{20} addresses)
>> Know Memory Map for MSP430x5529x Processors (from HW)
-- Addresses for RAM & FLASH, (good thing to have in notes!)
>> Know how to figure memory addresses

Memory Mapped I/O

What does it mean for I/O to be *memory-mapped*?

Quick Questions:

1) The long int i = 0x00081230 is stored in memory by a microprocessor as

Address	*Contents*
0213h	30h
0212h	12h
0211h	08h
0210h	00h

The microprocessor must be

a) Little Endian b) Big Endian c) Running Linux d) Running Windows 10

2) In the MSP430F5529, the RAM is

a) non-volatile system memory b) volatile data memory

c) non-volatile code memory d) consists only of the 16 CPU registers

3) In the MSP430F5529, the FLASH memory is

a) non-volatile code memory b) volatile data memory

c) volatile code memory d) not available in this model

MSP430F5529 Basic Digital I/O (HW3-4):

>> Eight independent, individually configurable digital I/O ports
-- Ports 1-7 are 8-bit wide and Port 8 is 3 bits wide

>> Each pin of each port can be configured individually as an input or an output

>> Each pin of each port can be individually read or written to

Function Select Register: Sets function of each pin in the port (i.e. P4SEL)
-- Bit = 0 = Selected for Digital I/O
-- Bit = 1 = Not selected for digital I/O (multiplexed pin functions)

*Direction Registe*r: Sets direction of each pin in the port (i.e. P2DIR)
-- Bit = 0 = Corresponding pin is an ***Input***
-- Bit = 1 = Corresponding pin is an ***Output***

Input Register: Where input to the port is read from (i.e. P2IN)
-- Bit = 0 = Logic low
-- Bit = 1 = Logic high

Output Register: Where data to be output from the port is written (i.e. P5OUT)
-- Bit = 0 = Logic low
-- Bit = 1 = Logic high

Drive Strength: Sets drive strength of port (we will usually leave as default)
--Bit = 0 = reduced drive strength (default)
--Bit = 1 = full drive strength

Pull-up/down Resistor Enable: Enable internal pull-up resistors (can be used for inputs)
--Bit = 0 = Not enabled (default)
--Bit = 1 = Enabled (see User's Guide)

>> All I/O port registers are ***memory mapped.*** Register names defined in ***msp430x4xx.h***
(Read from and Write to defined names as if writing to C variables...)

>> ***Polling***: Repeated checking of IO ports to see if they have data or need servicing (usually inside main loop)

```
#include "msp430.h"
#include <stdlib.h>

void  configPort()
{
   P5SEL = 0x00;
   P5DIR = (BIT7|BIT5|BIT3|BIT1);
}

void main()
{
   configPort();

   while (1)
   {
     char in = P5IN;
     P5OUT = (in & 0x55) << 1;
   }
}
```

a) Which port(s) and which pins are being used as digital inputs?

b) Which port(s) and which pins are being used as digital outputs?

c) Assume that the port 5 input register holds the value 6Dh. What value is written to the port 5 output register?