ECE 2049 LECTURE 9

TODAY

- FLOATING POINT
- EXAM 1 REVIEW
- NW 4
- CLOCKS + TIMERS

ADMINISTRIVIA

- EXAM 1: OUT AFTER CLASS
  - TAKE HOME FORMAT
  - OPEN BOOK/NOTES/INTERNET/ETC
  - EXAMS ARE INDIVIDUAL
  - WHEN DONE, UPLOAD TO CANVAS
  - DUE: THURSDAY, JUNE 15 2022 BY NOON EDT (IE, START OF CLASS)
    - IF THIS DEADLINE WOULD BE PROBLEMATIC FOR YOU, LET ME KNOW ASAP

- TOPICS: NW 1-4, LECTURES 1-8
  - NUMBER REPRESENTATIONS ← DEC, BIN, 2'S COMP, S-M
  - MSP430 ARCHITECTURE
    - BYTE ORDER
    - RAM VS FLASH
    - MEMORY ORG
  - DIGITAL I/O
    - SIMILAR TO NW
  - GENERAL CONCEPTS FROM LAB

OFFICE HOURS

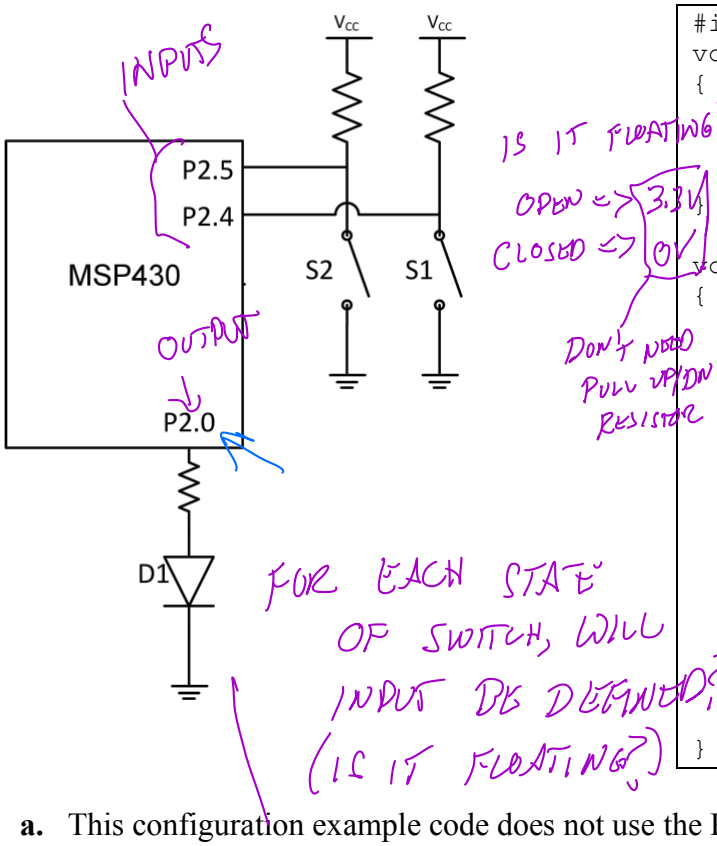
- TODAY 4-6PM (NICK)
- WED NOON-2PM (NICK) REMOTE?
- WED 3-5PM (RUSNI)

(CHECK CALENDAR ON CANVAS FOR LATEST SCHEDULE!)

Other Administrivia

- Lab 2: - Signoff / Submit Prelab when you start
  - We will cover remaining parts in lecture this week
  - Signoff due next Thurs, June 23

1. (5 pts) In this example, two switches and a blue LED are connected to an MSP430F5529 running the code shown below. Answer the following questions about the configuration.

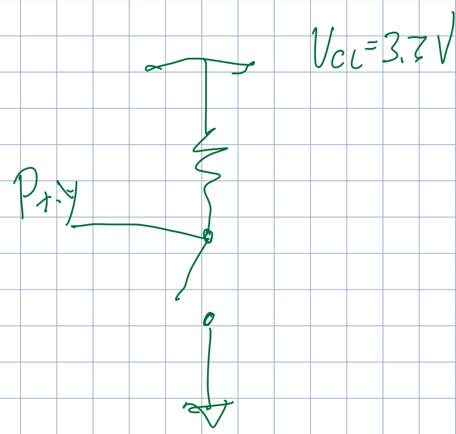


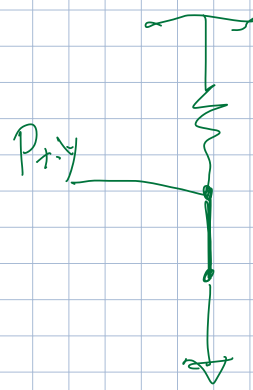

```
#include <msp430.h>
void configure_io(void)
{
    P2SEL &= ~(BIT5|BIT4|BIT0);
    P2DIR &= ~(BIT5|BIT4);
    P2DIR |=  (BIT0);
}

void main(void)
{
    char val, s;

    configure_io();

    while(1) {
        val = P2IN;
        s = (val & 0x60) >> 5;

        if(s > 1) {
            P2OUT &= ~BIT0;
        } else {
            P2OUT |= BIT0;
        }
    }
}
```

a. This configuration example code does not use the P2REN register to enable internal pull-up/pull-down resistors. Does it need to be configured in this case? Explain why or why not.

b. In the example main(), what happens to the LED (on or off) if `val = 0x7A`? What happens if `val = 0x1C`?

(Continued on the next page)

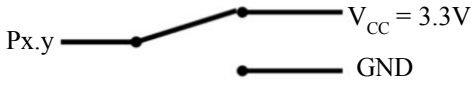




	7	6	5	4	3	2	1	0
P4IN	x	x	x	x	x	x	x	x
P4IN & BIT6	0	1	0	0	0	0	0	0

7	6	5	4	3	2	1	0
0	0	0	0	P7.2	P1.6	P4.5	P4.4

2. (5 pts) Say you want to connect four slide-switches (also known as "single-pole, double-throw" switches) like the one shown at the right to P7.2 and P4.6-4. The switches operate such that when the switch is slid to the right, the pin is connected to GND (0V), and when slid to the left is connects to V_{CC} (3.3V).



a. Try to complete the two functions `config_switches()` and `read_switches()` based on the skeleton and comments below. Your code must be typed—you can find this example as a C file on the course website as switches.c. We will review and discuss how to do this in detail in class.

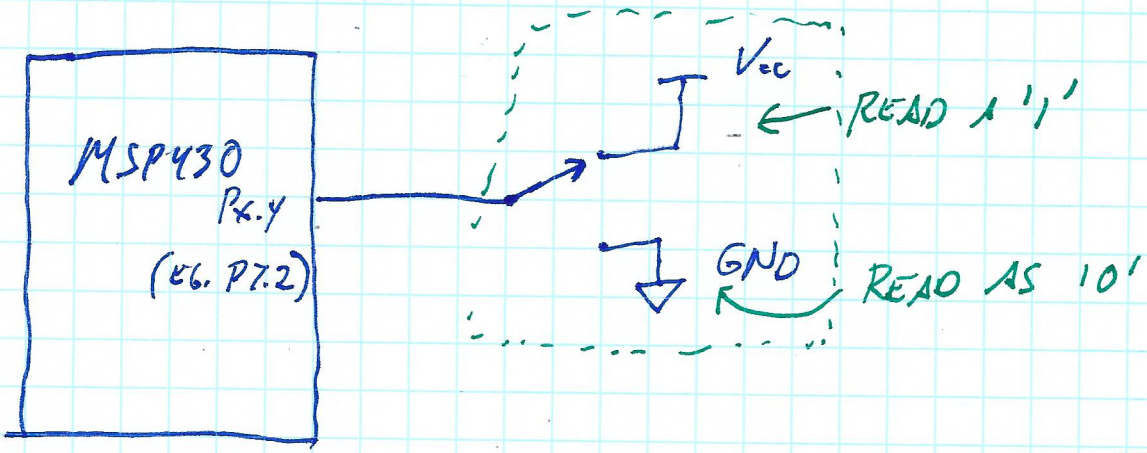
```
#include <msp430.h>

// Function prototypes
void config_switches(void);
char read_switches(void);

// Example main() to demonstrate how the functions are used--no need to modify it.
void main(void)
{
      char val;
      WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
      config_switches();

      while(1)
      {
            val = read_switches();
            // Assume something with val happens here...
      }
}

void config_switches(void)
{
      // Configure switches here!
}

// Return a value between 0-Fh corresponding
// to the value of the switches, with the values
// of each switch in the following bit positions:
// MSB                              LSB
// Bits 7-4    Bit 3    2.    1       0
// 0           P7.2  P4.6  P4.5    P4.4
char read_switches(void)
{
      char ret_val = 0;

      // Read switches and place the output
      // into the appropriate bit here!

      return ret_val;
}
```

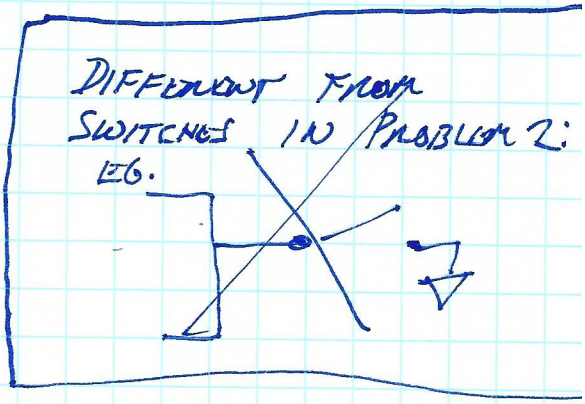
b. Assuming that your program has properly configured the slide switches, what should the function `read_switches()` return given the following register values? (Note, not all of these may be useful!)

`P7IN = 0x55, P7OUT = 0x44, P4IN = 0xCF, P4OUT = 0xD8`

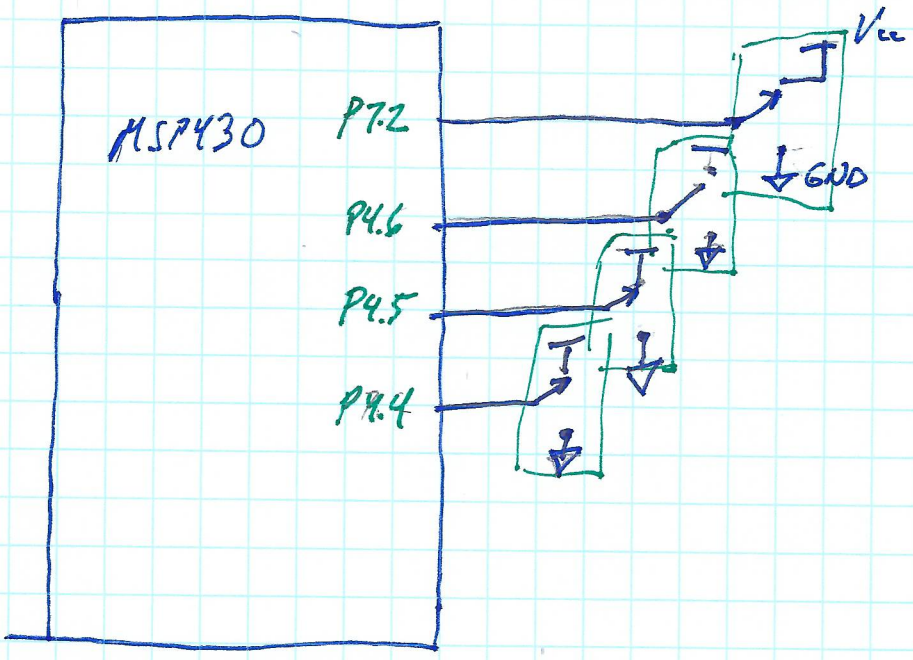
HW3.
PROBLEM 3: SWITCH HARDWARE

OB





FOR PROBLEM 3: 4 INDEPENDENT SWITCHES



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;          //. 2 BYTES
   float           b;          // 4 BYTES
   char            c;          // 1 BYTE
   unsigned int    d;          // 2 BYTES
   long int        e;          // 4 "
   double          f;          // 8 "
   int          arr[5];        //          (2 BYTES/INT) * 5 = 10 BYTES.
```

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.

ARR[i] IS SAME AS

$*(ARR_{START} + (i * SIZEOF(INT)))$

↑ DEREFERENCE

(GET DATA AT THIS ADDRESS)

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;
```

0000 0001 0000 0001

0000 0010 0000 0010

01 02 02

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

1111 1111 + 1 = 0000 0000

3) What value is assigned to z?

a) 2 b) 4 c) 4.2 d) 10

TRUNCATION.

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()
```

100 100 0 ... 99 FIND MAX NUMBER IN ARRAY

Quick Questions:

1) How many times does the while loop execute? i = 0..99

a) 99 b) 100 c) 101

2) To what value is big[47] assigned? 47 % 10 = 7

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

Data Representations (HW #1):

>> ***Integer representations***:
--Unsigned, sign-magnitude, two's complement and BCD

>> ***Expect Conversion Between Bases and Formats!***

Unsigned integers = 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

Sign Magnitude integers = *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!

Two's Complement integers = 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

Binary Coded Decimal = Each decimal digit expressed in binary nibble

367 = 0000 0011 0110 0111b

- TO DO FP IN A FAST WAY, NEED SPECIAL HARDWARE (FLOATING POINT UNIT, FPU)
- WE DON'T HAVE AN FPU, SO EMULATE BEHAVIOR.

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)

2.5+10?

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

Little Endian: The MSP430, like Intel processors, is "Little Endian" (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

Ex: 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
....	....

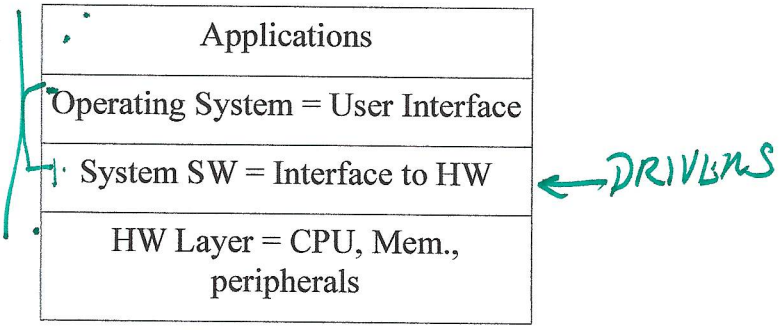
A memory dump from a big endian processor (also left to right)

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

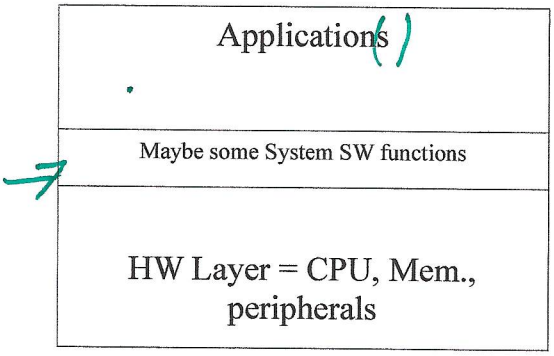
Network Byte Order = BIG ENDIAN!!!

Microprocessor Systems Architecture:

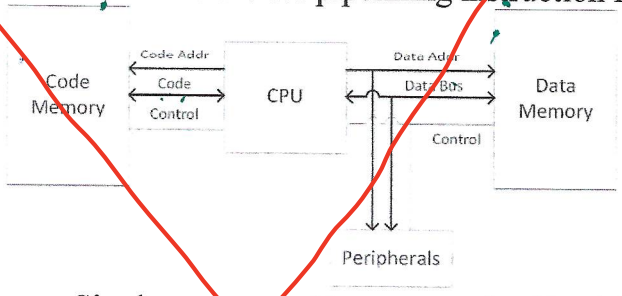
>> General Computing Hardware/Software Hierarchy



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

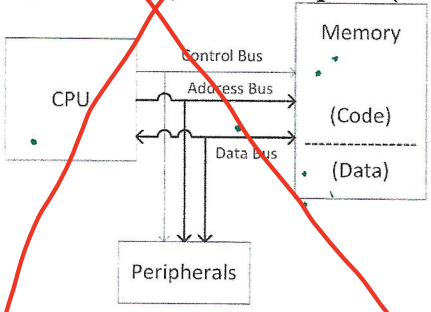


Harvard Architecture – Separate memory address spaces (and busses) for code and data ("Better" architecture for pipelining instruction fetches)



DON'T NEED FOR EXAM

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



MSP430

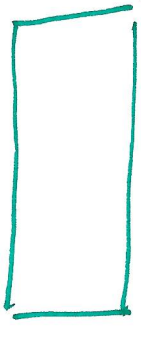
IN PRACTICE: HYBRID OF TWO FORMS (x86)

>> 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)

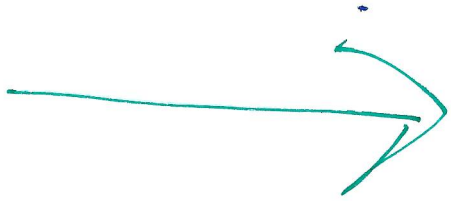
>> Flash is NOT byte writable!
-- Must be erased in multi-byte (e.g. 512 byte) segments
>> A flash write cycle takes much longer than read cycle

MSP430 is 16 bit Microcontroller

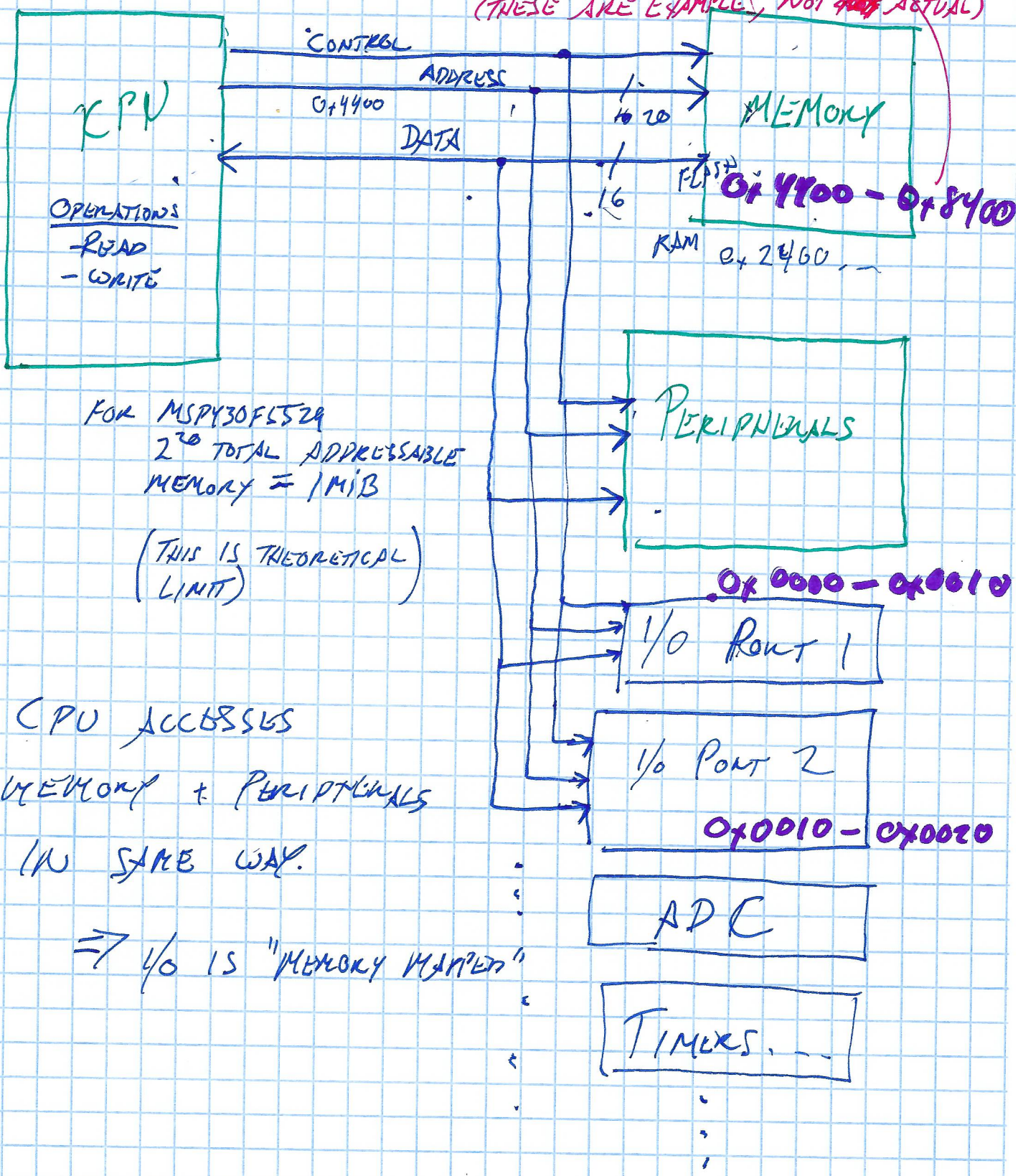
>> 16 bit word size = 16 bit internal registers
>> Also has 20 bit address bus (can access up to 1 MB = 2^{20} addresses)
>> Know Memory Map for MSP430x5529x Processors (from HW)
-- Addresses for RAM & FLASH, (good thing to have in notes!)
>> Know how to figure memory addresses

Memory Mapped I/O

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



ALL MEMORY AND PERIPHERALS ARE ASSIGNED ADDRESSES IN THE MEMORY MAP.
(THESE ARE EXAMPLES, NOT ACTUAL)



FOR MSP430F5529
2^{20} TOTAL ADDRESSABLE MEMORY = 1MiB

(THIS IS THEORETICAL LIMIT)

CPU ACCESSES MEMORY & PERIPHERALS IN SAME WAY.

⇒ I/O IS "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):

>> 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

CONFIG

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 Register: 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 ← OUTPUT

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) ← INPUT

>> 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...)

P3OUT =>

>> ***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;
   }
}
```

7 6 5 4 3 2 1 0
P5DIR 1 0 1 0 1 0 1 0

a) Which port(s) and which pins are being used as digital inputs? (SET TO 0)

PORT 5, PINS 6, 4, 2, 0

b) Which port(s) and which pins are being used as digital outputs? (SET TO 1)

PORT 5, PINS 7, 5, 3, 1

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

6Dh 0110 1101

0x55 0101 0101

0100 0101 <<

=> 1 0 0 0 1 0 1 0 b

BITS THAT CHANGE OUTPUT PINS

EXTRA CONTENT WE MISSED FROM LECTURE 3

How do you represent fractional numbers in binary form?

So far, we have only expressed integer values in binary. There are two conventions for representing fractions: fixed point and floating point.

$2^{-1} = 1/2 = 0.5$
$2^{-2} = 1/4 = 0.25$
$2^{-3} = 1/8 = 0.125$

Fixed Point

INTEGER 4.52 FRACTIONAL

In a given data type, we can define a binary "radix point", which is a fixed point that denotes fractional bits.

$2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$
0 0 0 0 | 1 0 1 0
INT | FRAC

$0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 0\times2^{-4}$
$= 0.5 + 0.125$
$= \boxed{0.625}$

In this format, the precision of the number is defined by the number of fractional bits.

For example, 4 fractional bits = $2^{(-4)} = 0.0625$ is the smallest fraction you can represent

Often, fixed-point representations are stored in scaled form as integers–it's up to you (the programmer) to treat them as fixed-point values.

INT X = 16;
0001 0000

IF YOU LIKE THIS: ECE4703

Floating Point

IEEE 754 -2.3×10^{-5}

Floating point is an IEEE standard used to approximate real-valued numbers to a certain number of decimal places.

FLOAT DOUBLE

There are two forms, *single precision* (32 bits) and *double precision* (64 bits). Each representation has three components:

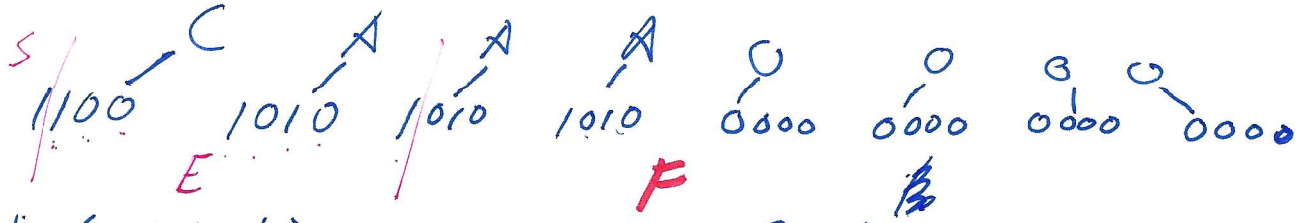
- A sign bit (S)
- An exponent (E)
- A fractional part (F), which is also called the "mantissa" or "significand"

Format for single precision: S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
Exponent is 8 bits, fractional part is the remainder (23 bits)

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

Example: Floating point to Decimal
What is the decimal equivalent of the floating point variable CAAA0000h?

1. FIND COMPONENTS



S = 1 (NEGATIVE)
E = 10010101b = 149
F = 0101010...

2. WRITE FRACTIONAL PART AS .1.F, FIND DECIMAL

1.F = 1.010101
$= 1 + 2^{-2} + 2^{-4} + 2^{-6}$
$= 1 + 0.25 + 0.625 + 0.00125$
$= 1.38125$ (NOT DONE)

3. USE FORMULA

$V = (-1)^1 (2^{149-127})(1.38125)$

$= \boxed{-557056.0}$

WE NEED SPECIAL HARDWARE TO DO THIS FAST.

Some features of floating point:

- Effective range: approximately +/- 10^{38}
- Single precision has ~7 decimal digits of precision
  - Double precision and others have more
- Special representations for +/- infinity, NaN
- Standard has conventions for rounding, normalization, etc.

Example: Decimal to floating point $VALUE = (-1)^S (2^{E-127})(1.F)$

Represent -5.375 as a single-precision floating-point number.

1. WRITE AS BINARY

-101.011

MUST MOVE BY 2

$.375 = 0.25 + 0.125$

$2^{-2} + 2^{-3}$

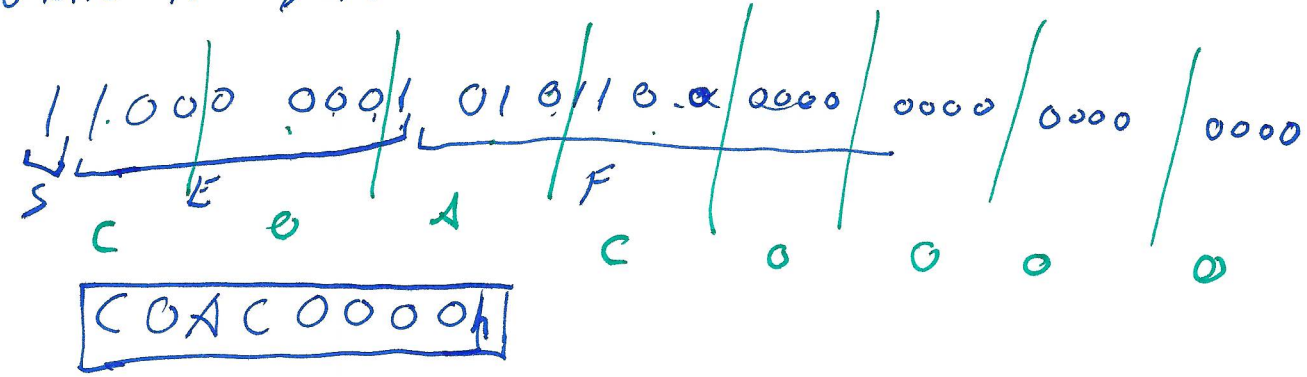
2. WRITE $1.F$ TO FIND E

$S = 1$

$E = 127 + 2 = 129 = 1000\ 0001b$

$F = 01011$

3. WRITE IN DEFINED FORMAT



C0AC0000h