CprE 288 – Introduction to Embedded Systems
Exam 1 Review
• Exam 1: Tuesday 10/2, in class
  – Open textbook, datasheet, 1 page of notes, and calculator allowed
  – 75 minutes
  – Electronic textbook and electronic Datasheet is fine. Nothing else on your electronic device can be used or you will receive an F for CPRE 288

http://class.ece.iastate.edu/cpre288
Exam Coverage

• Focus
  – Embedded C programming
  – Memory Mapped I/O and Registers
    • General Concept of Memory mapped registers
    • GPIO
• Covers material in lectures
  – No UART questions
  – No ADC questions
  – No Interrupt questions
• Covers material in homework's (HW1 – HW4)
• Cover materials in Labs 1 - 4
• 60 points total (10% of course grade)

http://class.ece.iastate.edu/cpre288
Exam Preparation

Suggested preparation steps:

• Review homeworks
• Review lectures
• Review labs
• Make a nice notes sheet
• Note: this set of slides are *not comprehensive*
## Keywords

- `char`
- `short`
- `int`
- `long`
- `float`
- `double`
- `enum`
- `struct`
- `union`
- `break`
- `case`
- `continue`
- `default`
- `else`
- `for`
- `goto`
- `if`
- `return`
- `switch`
- `while`
- `auto`
- `const`
- `extern`
- `register`
- `signed`
- `static`
- `unsigned`
- `volatile`
- `sizeof`
- `typedef`
- `void`
Could you write the following statements by hand?

- Loops (for, while, do)
  - Write a **for loop** to sum elements of an array or count characters in a string
  - Do you know the syntax of a **do while loop**, **for loop**, and **while loop**?

- **typedef**
  - Could you write a typedef definition
  - Do you know what it means when you see a variable type like `uint8_t`?

- Switch statements
  - Do you know where the semi-colon and colons go in a switch/case statement?
  - Do you understand how the control flow can fall through a case?

- Control flow
  - Do you understand the keywords **break** and **continue** and their use?
Know your Operators

• Do you know the difference between these two sets of operators?

```
&
|
~
^  
^&
||
!
```
char a = 20, b = 10, r = 5;

// math operations
r = a + b;
r = a - b;
r = a * b;
r = a % b;

// bitwise operators
r = a & 3;
r = a | b;
r = a ^ 0xFF;
r = ~a;
r = a >> 3;
r = b << r;

// conditional
r = (r) ? a : b;

// boolean
r = a || b;
r = a && b;
r = !a;
r = a < 20;
r = b <= 15;
r = b > 10;
r = a >= b;

// post and prefix
a++;
++a;
b--;
--b;

// assignments
r = a = b = 42;
r |= a;
r &= b;
Know how to use Operator Precedence

- Can you use this table?

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>++ -- ()</td>
<td>Suffix/postfix increment and decrement Function call</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>[] . -&gt;</td>
<td>Array subscripting Element selection by reference Element selection through pointer</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>++ -- + - ! ~ (type) * &amp; sizeof</td>
<td>Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Type cast Indirection (dereference) Address-of Size-of</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>3</td>
<td>* / %</td>
<td>Multiplication, division, and modulus (remainder)</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>Addition and subtraction</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>For relational operators &lt; and ≤ respectively For relational operators &gt; and ≥ respectively</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>== !=</td>
<td>For relational = and ≠ respectively</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>^</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>&amp; &amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>? :</td>
<td>Ternary conditional</td>
<td>Right-to-Left</td>
</tr>
<tr>
<td>14</td>
<td>= + = * = /= %= &lt;&lt;= &gt;&gt;= &amp; = ^=</td>
<td>=</td>
<td>Direct assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR</td>
</tr>
<tr>
<td>15</td>
<td>,</td>
<td>Comma</td>
<td></td>
</tr>
</tbody>
</table>
Do these declarations make sense?

```c
void main() {
    char x = 5, y = 10;
    char z;
    char array1[10];
    char array2[] = {1, 2, 3};
    char array3[5] = {1, 2, 3};
    char *str = "Hello!";

    int i = 7;
    int *ptr = &i;
    int **pp = &ptr;
    char *p;
}
```
Know your Declarations

• Do these declarations make sense?

```c
struct House {
    unsigned long value;
    unsigned char baths;
    unsigned char bedrooms;
    unsigned char stories;
    unsigned long footage;
};

void main() {
    struct House my_home;
    struct House *bob_home = &my_home;

    my_home.baths = 1;
    my_home.value = 115000;
    bob_home->baths = 3;
    bob_home->value = 230000;
}
```
Data Structures

- Array access
- Pointers
  - Dereference
  - Address operator
- Access members of structs and unions
- Bit field defined
- Know the difference between a struct and union
Pointers

• What are pointers?
• Relationship between
  – pointers
  – array names
• Pointer arithmetic
• Be sure to review class examples and homework problems related to pointers

http://class.ece.iastate.edu/cpre288
C-strings

• Review the concept of C-strings
• Relationship between
  – C-strings
  – arrays
• Understand the importance of the NULL byte of a C-string
Bitwise operations

- Forcing bits to 1, Forcing bits to 0, Toggling (i.e. inverting) bits
- Testing if any of a set of bits is set to 1
  1) Decide which bits you want to test
  2) Isolate those bits (i.e. force all other bits to 0)
- Testing if all bits of a set of bits are set to 1
  1) Decide which bits you want to test
  2) Isolate those bits (i.e. force all other bits to 0)
  3) Compare for equality with the Mask
- For the case of testing for bits set to 0. Follow bit(s) set to 1 testing procedure, but invert the variable that you are testing.
- Generic systematic checking example

```c
if( (x & MASK_ALL1s) == MASK_ALL1 &&
    (~x & MASK_ALL0s) == MASK_ALL0s &&
    (x & MASK_ANY1s) &&
    (~x & MASK_ANY0s) )
```
Memory Mapped I/O

• Concept of Memory Mapped I/O
  – Device registers can be accessed from a program as if accessing memory (i.e. devices’ registers are “mapped” to memory addresses)
  – Allow sending commands and checking status of a devices by just reading and writing to memory mapped locations.

• Using GPIO

• Polling device status vs. device initiating an interrupt.
Pointer Problem

• There will be a pointer problem similar to the one in HW4.
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;  
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;  
    *p_ptr = 0xFEC0;
```c
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
```
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;  
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main()
{
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
```c
typedef struct coord{
    char x;
    char y;
} coord_t;

typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;

    coord_ptr = &my_coord[1];
    num_ptr = &num_array[1];
    p_ptr = &num_array[0];
    *p_ptr = 0xFEC0;
    p_ptr++;
    my_coord[1].x = 0x33;
    my_coord[1].y = 0x44;
    num_array[0] = 0x06;
    num_array[1] = 0x04;
}```
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main()
{
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x05040;
    p_ptr++;
    *p_ptr = 0xFEC0;
}

type| Address | Variable Name | Value
---|---------|--------------|------
0xFFFF_FFFF | coord_ptr | 0xFF
0xFFFF_FFFE | num_ptr | 0xFF
0xFFFF_FFFD | p_ptr | 0xFF
0xFFFF_FFFC | a | 0x07
0xFFFF_FFFB | my_coord[1].y | 0x44
0xFFFF_FFFA | my_coord[1].x | 0x33
0xFFFF_FFF9 | my_coord[0].y | 0x00
0xFFFF_FFF8 | my_coord[0].x | 0x00
0xFFFF_FFF7 | num_array[1] | 0x00
0xFFFF_FFF6 | num_array[0] | 0x00
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
typedef struct coord{
    char x;
    char y;
} coord_t;

coord_t *coord_ptr;
int *num_ptr;
int **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2] = {1, 4};

int main(){
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
Writing the Body of a function

• There will be 1 or 2 problems where you will be asked to write the body of a function to implement a defined computation.
Show your work

• Show intermediate steps. This will help us give partial credit

• Write down assumptions. In general Profs/TAs will not answer questions during the Exam.
  – They will just tell you to write down your assumptions
  – The exception will be if a major typo is found by a student.
QUESTIONS
Question 1

A. How many bytes are each of the following types (on the TMC123)?
   • char, short, int, long, float, double

B. What range of values can be stored in an unsigned char?

C. What range of values can be stored in a signed char?

D. What is the value stored in \( x \) after this code runs?
   
   ```c
   int x, y, z;
   x = y = z = 10;
   ```
### Question 1 (answer)

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes</th>
<th>sizeof()</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>-</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>signed char</td>
<td>1</td>
<td>-</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>-</td>
<td>0 to 255</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
<td>-</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int (on TMC4123)</td>
<td>4</td>
<td>-</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>unsigned int (on TMC4123)</td>
<td>4</td>
<td>-</td>
<td>0 to 4294967295</td>
</tr>
<tr>
<td>(pointer on TMC4123)</td>
<td>4</td>
<td>-</td>
<td>Address Space</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>-</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>signed long</td>
<td>4</td>
<td>-</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td>-</td>
<td>0 to 4294967295</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>-</td>
<td>±1.175e-38 to ±3.402e38</td>
</tr>
</tbody>
</table>
Question 2a

• When is the condition of the following if statement true?

```java
if ((x = 3) || (x & 1)) {
    // do something
}
```
• When is the condition of the following **if** statement true?

```c
if ((x = 3) || (x & 1)) {
    // do something
}
```

• **The statement is always true.** Know the difference between the assignment operator (=) and the equality operator (==).
  
  – The value on the left (x = 3) is always true, as the value of an assignment is the value that was assigned. This allows programmers to have compound assignments.
• When is the condition of the following `if` statement true?

```java
if ((x == 3) || (x & 1)) {
    // do something
}
```
• When is the condition of the following if statement true?

```java
if ((x == 3) || (x & 1)) {
    // do something
}
```

• The statement is true if x is either equal to 3 or bit 0 is set.
• When is the condition of the following `if` statement true?

```plaintext
define ((x & 0x08) == 0x08) {
    // do something
}
```
When is the condition of the following if statement true?

```java
if ((x & 0x08) == 0x08) {
    // do something
}
```

The statement is true if bit3 of x is set.

- x = 0b00001000; condition is TRUE
- x = 0b01001110; condition is TRUE
- x = 0b00101001; condition is TRUE
- x = 0b00000000; condition is FALSE
- x = 0b11100000; condition is FALSE