PRELAB for Lab 4

Go to page 76 of the C compiler User's Guide to complete the following table. For the type, fill in data type that produces a variable the the size give. For max/min write in the maximum and minimum values that can be represented with the data type in that row.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Size | signed/unsigned | Type | min | max |
| 8-bit | unsigned | Unsigned char | 0 | 258 |
| 8-bit | signed | Signed char | -128 | 127 |
| 16-bit | unsigned | unsigned short | 0 | 65535 |
| 16-bit | signed | int | -32768 | 32767 |
| 32-bit | unsigned | long | 0 | 4 294 967 295 |
| 32-bit | signed | long | -2,147,483,648 | 2 147 483 647 |
| 64-bit | unsigned | Long long | 0 | 18 446 744 073 709 551 615 |
| 64-bit | signed | Long long | -9 223 372 036 854 775 808 | 9 223 372 036 854 775 807 |

When writing embedded C code, its always a good idea to separate your code from the architecture as much as possible because it is easier to change... This is why its better to

* use the peripheral register names in your code (e.g. P2IN) rather than their address (e.g. 0x28).
* use peripheral register field names in your code (e.g.

Because space is limited on microcontrollers, its a common practice to use variables with a range suitable for the task at hand. Unfortunately, there is no standard among C compilers between the basic data types like char, short, long and the number of bits in the underlying data representation. Furthermore, when writing and reading code, it is not readily apparent how may bits are in a short or long variable. Consequently, we will write our programs using typed definitions that provide an obvious connection between the data type and the number of bits in the representation.   
  
Start by consulting the [Typedef wiki page](http://en.wikipedia.org/wiki/Typedef). Next, write the C code definitions for the following

|  |  |  |
| --- | --- | --- |
| type | Meaning | C typedef declaration |
| int8 | unsigned 8-bit value | typedef unsigned char int8; |
| sint8 | signed 8-bit value | typedef signed char sint8; |
| int16 | unsigned 16-bit value | typedef unsigned short int16; |
| sint16 | signed 16-bit value | typedef signed short sint16; |
| int32 | unsigned 32-bit value | typedef unsigned long int32; |
| sint32 | signed 32-bit value | typedef signed long sint32; |
| int64 | unsigned 64-bit value | typedef unsigned long long int64; |
| sint64 | signed 64-bit value | typedef signed long long sint64; |

**Discover calling/return convention**

Make a project around [simpleLab4.c](http://ecse.bd.psu.edu/cmpen352/lab/lab4/code/simpleLab4.c). While the functioning of the program is not really that important, lets first take some time and understand what is going on in this program before we look at the underlying assembly language. Use CCS to step through the program and examine the a,b,c,d,e variables in main, just after the call to the function func in line 16.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | a | b | c | d | e |
| 1st | 1 | 2 | 3 | 4 | 2 |
| 2nd | 10 | 9 | 8 | 7 | 8 |
| 3rd | 16 | 15 | 14 | 13 | 14 |
| 4th | 22 | 21 | 20 | 19 | 20 |
| 5th | 28 | 27 | 26 | 25 | 26 |

Now examine the assembly code generated by the compiler by selecting the **View -> Disassembly** menu item. You should see the disassembly window as a selectable tab in the xxx subwindow. Your first task is to find the code for the function **func** and write down the starting and ending address in the table below. Next, identify which registers are used to pass the input parameters from main to the function. Write their identities below. If its not clear which register hold which input parameter, go ahead a change the code, so that **func** only has one input parameter, recompile the code, and then examine the assembly. Finally, determine which register is used to return the value from func to main.

|  |  |
| --- | --- |
| Object | Value |
| starting address of func | C0BA |
| ending address of func | C0C6 |
| Register holding w | R12 |
| Register holding x | R13 |
| Register holding y | R14 |
| Register holding z | R15 |
| Register holding return value | R12 |