

## RE: Robby, \_\_int8 limited to: -128 to 127.

---

*Source:* <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2007-12/msg00369.html>

---

- *From:* Robby <[Robby@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:Robby@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Wed, 12 Dec 2007 17:07:01 -0800
- 

Hello all!

I deeply appologize if I have caused such a confusion! But its really not my fault nor is it yours!

The integer is 8 bits unsigned. In this compiler, if I want a 16 bit I have to declare it as a 16 bit long variable.

And yes an 8bit int cannot hold a value of 256. CCS is currently working on the issue.

My ininitial code works fine without pointers. But with pointers the logic should be the same.

As for the fellows that believe that my code is wrong, well, let me explain myself.

Here's the original code. Now skip down to the explanations part under this code.

=====*Calling routine!*

```
int spiA0=1,spiA1=0,spiA2=0;
```

```
//INCREMENT_ADDRESS BY AMOUNT OF INCREMENTATIONS REQUIRED  
INC_ADDR(16,&spiA2, &spiA1,&spiA0);
```

=====*Function!*

```
void INC_ADDR( /***/INCREMENT_ADDRESS***/  
int INCBY, //AMOUNT OF INCREMENTATIONS REQUIRED  
int *A2, //MSB OF MESSAGE ADDRESS IN PAR FLASH  
int *A1, //MIDSB OF MESSAGE ADDRESS IN PAR FLASH  
int *A0) //LSB OF MESSAGE ADDRESS IN PAR FLASH  
{  
int i;  
  
for(i=0;i<INCBY;i++)  
{  
if((*A0)==255) //Before A0 turns to 0, statement is true  
{(*A1)++; //Increment central byte of flash adress  
if((*A1)==255) //Before A1 to 0, statement is true
```

RE: Robby, \_\_int8 limited to: -128 to 127.

```
{ (*A2)++; //Increment MSBYTE of flash adress  
} }
```

```
(*A0)++;  
}}
```

=====

Look, its like this:

In order to access the data from my external memory I need a three byte address coming out of my Micro controller. This is because I am creating my own electronic computer hardware architecture. Therefore I have a microcontroller connected to an external flash memory.

So suppose I need to access the data at address 000001 Hex from my flash.

Therefore byte1,byte2 and byte3 must look like this:

BYTE3 0000 0000 BYTE2 0000 0000 BYTE1 0000 0001

In my code spiA0 is BYTE1, spiA1 is BYTE2 and spiA2 is BYTE3

So if my code starts and fetches data from the memory at location 000001 Hex as initialized in my code above (see spiA0, spiA1,spiA2 above!) The three bytes are outputted to the memory address of the flash and my controller retrieves the pertinent 8 bit data via another 8 bit bus.

Then later say I want to access location 000002HEX of the memory. I call INC\_ADDR as so:

```
INC_ADDR(1,&spiA2, &spiA1,&spiA0);
```

Note this time the first parameter is a 1 instead of a 16! This mean we will increment the 3 byte address by 1.

Once in the INC\_ADDR function, the contents of the for loop is executed. Lets look at what will happen when the first command in the code will be executed.

```
if((*A0)==255) //Before A0 turns to 0 (at255), statement is true
```

Here, A0 is 1 right. So logic will skip the logic in the "if" statement and will go straight to the:

```
(*A0)++;
```

and increment the A0 value from 1 to 2.

When the INC\_ADDR function returns, then I can access data from location 000002HEX of the memory.

As this scenario goes on, and I keep calling the INC\_ADDR function in order

RE: Robby, \_\_int8 limited to: -128 to 127.

RE: Robby, \_\_int8 limited to: -128 to 127.

to increment my 3 byte address, eventually, there will be a call where A0 will equal to 255, right! We do agree that when A0 is 255 and we call upon INC\_ADDR, its because we were at 0000FFHEX and we now require require 000100H. Right!

Once in the INC\_ADDR function, the contents of the for loop is executed. This time around lets look at what will happen when the first command in the code will be executed.

In this instance, A0 is 255 right. So logic within the "if" statement will be executed. As shown below: (Please view comments)!

```
for(i=0;i<INCBY;i++)
{
if((*A0)==255) //A0 is 255, so lets go increment A1!
{(*A1)++; //Increment A1!
if((*A1)==255) //Did A1 reach 255, if no then leave A2 alone!
{ (*A2)++; //Increment A2
} }
}

(*A0)++; //Increment A0 or wrap A0 back to 0
```

A1 will be incremented, A2 will be left alone and as the last command, A0 is incremented, but since it was at 255, it will be reset to 0, Hence the required address:

BYTE3 0000 0000 BYTE2 0000 0001 BYTE1 0000 0000

or

000100HEX

Once the function returns I access data at location 000100HEX of flash! The same logic is applied for A2.

This logic works for this particular requirement and is probably not the most efficient way to do this. I will later look into other possibilities as suggested by Norbert to avoid unnecessary looping in the case I would want to increase my address from say 000001 to 00FFFF.

However in this case, I don't understand why I would worry about postfix and prefixes on the variables. I always thought that prefix and postfix were important when using lines like this:

```
int x;
x = ++(*A0);
```

or

```
x = (*A0)++;
```

RE: Robby, \_\_int8 limited to: -128 to 127.

RE: Robby, \_\_int8 limited to: -128 to 127.

I don't know why suggestions are made where I should use "++\*A1;" as opposed to "\*A1++;" . I mean what difference would this make in my current logic described above.

I have been using this fragment of code and aquired amazing results in this microcomputer system.

Okay, its getting late, I have to run... sorry for the long post, I don't mean to bore anyone with my silly memories and logic.

All feedback appreciated from all.

I thankyou all very much for your post, I sincerely appreciate it.

Cheers and happy holidays!

—

Best regards

Robert

"Jeff Relf" wrote:

Are you ( Robby ) asking about the contents of A1, the \_\_int8 ?  
or A1 the pointer ? What does sizeof( int \* ) return ?

If it's the \_\_int8 ( i.e. sizeof( int ) == 1 )  
then it can't be 256, as it's limited to: -128 to 127.

If it's the pointer you're talking about,  
then you have to tell us what it's value was before entering the loop.

.

RE: Robby, \_\_int8 limited to: -128 to 127.