PIC ASM

Here you can find tips, tricks, tutorials and documentation concerning the Microchip PIC assembler language.

Tips & Tricks, Questions and Answers

How to locate two lables at the same memory position?

If you want to export a label for C applications, and you want to use the same name for the variable in C and ASM, you can do this in the way shown here (for example in module_xxx.asm):

```
global VAR1
global VAR2
global _VAR1
global _VAR2

MY_VARS UDATA
_VAR1 RES 0
VAR1 RES 1
_VAR2 RES 0
VAR2 RES 1
```

This will put _VAR1 / VAR1 at the same memory position. Never put the _VAR1 RES 0 **after** VAR1 RES 1! This would cause the assembler to give _VAR1 an address, but would not reserve any memory for it!

Also refer this discussion in the forum: http://www.midibox.org/forum/index.php/topic,12846.0.html

How does the SDCC compiler push function parameters to the stack?

```
unsigned char FRAM_WriteBuf(unsigned char device_addr,unsigned int
address,unsigned char byte_count, unsigned char * buffer) __wparam;
```

- If the function is defined with the __WPARAM directive, the first byte of the first parameter will be moved to WREG.
- All the other paramteres will be pushed to the stack in reverse direction (last parameter first).
- If a parameter is more than one byte long, always the most significant byte will be pushed first.
- Pointer types can have different sizes (depends on the target device, maybe also configuarble). When I compiled the code with the above function call for a PIC18F452, the pointer parameter was 3 bytes long. (**Update this info if you know more**)
- If the first parameter is more than one byte long, the MSB goes to WREG, the other bytes will be

pushed to the stack **last** (also MSB to LSB).

The reverse direction makes sense when popping the parameters from the stack. The first byte popped will be the first parameter's LSB, the last byte popped is the last parameters MSB. So if the pointer parameter is the last parameter, you will not have to care if it is a 2 or 3 byte-type. You just read LSB / MSB, if there's a byte more (and you don't need it for the target's device address-range), you just forget about it.