Writing Assembly code for 8-bit devices

Jan 7, 2020 • Knowledge

Title

Writing Assembly code for 8-bit devices

Article URL

https://microchipsupport.force.com/s/article/Writing-Assembly-code-for-8-bit-devices (https://microchipsupport.force.com/s/article/Writing-Assembly-code-for-8-bit-devices)

Question

What things must I manage when writing assembly code?

Tags:

Writing Assembly code

Inlined assembly code

Best practices for writing assembly code

MPLAB XC8 compiler

Answer

When writing assembly code by hand, you assume responsibility for managing certain features of the device and formatting your assembly instructions and operands. The following list describes some of the actions you must take.

• Whenever you access a RAM variable, you must ensure that the bank of the variable is selected before you read or write the location. This is done by one or more assembly instructions. The exact code is based on the device you are using and the location of the variable. Bank selection is not be required if the object is in common memory, (which is called the access bank on PIC18 devices) or if you are using an instruction that takes a full address (such as the movff instruction on PIC18 devices). Check your device data sheet to see the memory architecture of your device, as well as the instructions and registers which control bank selection. Failure to select the correct bank will lead to code failure.

The BANKSEL pseudo instruction can be used to simplify this process

For example:

BANKSEL PORTA

MOVWF PORTA

• You must ensure that the address of the RAM variable you are accessing has been masked so that only the bank offset is being used as the instruction's file register operand. This should not be done if you are using an instruction that takes a full address (such as the movff instruction on PIC18 devices). Check your device data sheet to see what address operand instructions requires. Failure to mask an address can lead to a fixup error or code failure.

The BANKMASK macro can truncate the address for you.

```
BANKSEL PORTA
movwf BANKMASK(PORTA)
```

To use the BANKMASK macro inside inlined assembly code:

```
asm ("BANKSEL PORTA");
asm ("movwf " ___mkstr(BANKMASK(PORTA)));
```

• Before you call or jump to any routine, you must ensure that you have selected the program memory page of this routine using the appropriate instructions. You can either use the PAGESEL pseudo instruction, or the fcall or LJMP pseudo instructions (not required on PIC18 devices) which will automatically add page selection instructions, if required.

```
fcall _foo
PAGESEL $ ; select this page
```

• You must ensure that any RAM used for storage has memory reserved . If you are only accessing variables defined in C code, then reservation is already done by the compiler. You must reserve memory for any variables you only use in the assembly code using an appropriate directive such as DS or DABS.
It is often easier to define objects in C code rather than in assembly code.

```
DABS 1,0x100,4,foo
```

that is identical to the following directives:

```
GLOBAL foo
foo EQU 0x100
DABS 1,0x100,4
```

• You must place any assembly code you write in a psect .
A psect you define may need flags (options) to be specified. Take particular notice of the delta, space, reloc and class flags. If these are not set correctly, compile errors or code failure will almost certainly result. If the psect specifies a class and you are happy with it being placed anywhere in the memory range defined by that class, it does not need any additional options to be linked; otherwise, you will need to link the psect using a linker option.
Assembly code that is placed in-line with C code will be placed in the same psect as the compiler-generated assembly and you should not place this into a separate psect.
Some examples of the use of the PSECT directive follow:

```
PSECT foo
PSECT bar,size=100h,global
PSECT foobar,abs,ovrld,class=CODE,delta=2
```

• You must ensure that any registers you write to in assembly code are not already in used by compiler-generated code. If you write assembly in a separate module, then this is less of an issue because the compiler will, by default, assume that all

registers are used by these routines. No assumptions are made for in-line assembly (although the compiler will assume that the selected bank was changed by the assembly) and you must be careful to save and restore any resources that you use (modify) and which are already in use by the surrounding compiler-generated code.

URL Name

Writing-Assembly-code-for-8-bit-devices

Rate This Article :

(https://www.microchip.com)