

SXCheck Version 1.01

General Notes

This program is donated to the public domain, it may be freely distributed without charge.

Neither the author nor the author's company make any warranties that the program is free of errors, or that it is suitable for a specific purpose.

The author would be thankful to receive any error reports, comments, and hints for improvements. Please send a piece of sample code together with your comments whenever possible.

Happy "SXing",

Guenther Daubach
MDA EDV-Beratung GmbH
Im Eulenflug 25
D-51399 Burscheid, Germany
Tel./Fax: +49 2174 – 785 931
email: g.daubach@mda-burscheid.de
Home: www.g-daubach.com

Why did I Write SXCheck?

When writing programs for the SX microcontroller in Assembly, a common mistake made by many programmers (including myself) is to forget the hash mark in front of a constant, e.g.:

	<code>mov</code>	<code>w, \$10</code>
instead of correctly	<code>mov</code>	<code>w, #\$10</code>
or	<code>mov</code>	<code>w, InitValue</code>
instead of correctly	<code>mov</code>	<code>w, #InitValue</code>

When the leading hash sign is missing, the Assembler assumes that the *contents* of the register at address \$10 or the *contents* of the register whose address is specified with the symbolic name "InitValue" shall be copied into w.

With the leading hash sign, the Assembler generates code that moves the *constant* value of \$10, or the *constant* value that is assigned to the symbolic name "InitValue" into w.

Another mistake (at least I make sometimes) is messing up bit and byte instructions, like typing

	<code>add</code>	<code>Counter, ra.0</code>
instead of correctly	<code>addb</code>	<code>Counter, ra.0</code>

Earlier versions of SX assemblers did not generate an error or warning for such typea of errors, where newer versions do. Nevertheless, I though it might be handy to add some checking for Bits and Bytes to SXCheck as well. More about this later in this document.

Setting up SXCheck

When developing this application, I only had a German version of MS Visual Basic available. Therefore, the Setup dialog language is German.

Here are the required steps for installation:

Open the folder to where you have unzipped the downloaded files, and double-click Setup.exe to start.

In the welcome screen that shows up after a short while, click the left Ok button to continue. I suggest that you use the defaults for the further install options. If you need to change the default installation path by some reason, click the "Verzeichnis wechseln" button to open the standard windows file select box and enter another path there.

Then click the large button showing a computer and a diskette box to accept the installation path (either the default or the path you have entered).

The next dialog is used to select a program group for the start menu. Click the left "Weiter" button to continue.

The installation process then begins, and after completion a final dialog box is shown. Click the OK button to finish.

In the Windows Start menu, you should find a new program group "SXCheck" now, and in this group there is a link to launch SXCheck.

Running SXCheck

Start the program from the Windows Start menu. First, a file select box will open that allows you to navigate to a folder, where an SX assembly source code (.SRC) file is located, and select a file name.

After you have selected a file, SXCheck immediately starts processing the file, and shows the results in a list box.

Click the "Select File" button in order to select another file, or the "Check Again" button in order to check the same file again after you have changed and saved it with the SX-Key IDE or another text editor program. Note that SXCheck opens the SRC file in non-shared, read-only mode. This means that it does not make any changes to the original SRC file. On the other hand, depending on what text editor you use, it might be necessary to close the editor before running SXCheck in case the text editor keeps the file open because SXCheck would terminate with an error (can't open the file).

Click the "Exit" button to leave SXCheck.

How SXCheck Works

SXCheck performs two passes through the code to be checked. During the first pass, it collects the information about variables, constants, and bit definitions, and saves them in a symbol table. Here are some examples:

InitVal = \$10

or

InitVal EQU \$10

cause that "InitVal" is stored in the table, as type "Constant"

```
Counter      DS      1
```

causes that "Counter" is stored in the table, as type "Variable"

```
TxTPin      =      ra.0
```

causes that "TxDPin" is stored in the table, as type "Bit"

When SXCheck finds code like this

```
InitVal1     =      $10
InitVal2     =      InitVal1
```

when processing "InitVal2", it first searches the symbol table for "InitVal1", and assigns the type of "InitVal1" to the type of "InitVal2" (Constant in this example) and saves it in the symbol table as well.

Warnings generated by SXCheck

When SXCheck detects "suspicious" code, it adds a warning like this to the displayed list box:

```
Line 77: 2nd OP is a constant w/o leading '#'
         cje      DOWN, 0, next_bit
```

The first line specifies the line number in the source code file, together with a warning message. Here the warning means that the second operand (the 0) seems to be a constant, but has no leading hash mark.

In the next line, the original code line is shown.

The following warnings may be generated:

2nd OP is a constant w/o leading '#'

This warning will be generated together with ADD, AND, OR, SUB, XOR, MOV, CJA, CJAE, CJB, CJBE, CJE, CJNE, CSA, CSAE, CSB, CSBE, CSE, or CSNE instructions when the second operand is either a numeric constant (decimal, binary, or hex), or when the second operand is a symbolic name that has been specified as a constant, e.g.

```
mov    w, 10
```

or

```
InitVal = $10
```

```
mov    w, InitVal
```

will cause such warnings.

2nd OP is a variable

This warning will be generated together with ADD, AND, OR, SUB, XOR, MOV, CJA, CJAE, CJB, CJBE, CJE, CJNE, CSA, CSAE, CSB, CSBE, CSE, or CSNE instructions when the second operand is a symbolic name that has been specified as a variable, e.g.

```
        org      $08
Temp DS      1
```

```
mov    w, #Temp
```

will cause such warning. In the first pass, SXCheck has detected that "Temp" is a symbolic name for the register at \$08, and saved it as type "Variable" in the symbol table.

In the mov instruction, a leading hash mark before the symbolic name is found which means that the Assembler would mov the *address* of "Temp" into w but not the contents of "Temp". Sometimes, this may be what the programmer wants to do, e.g. when setting up for indirect addressing. This is why SXCheck generates "Warnings" only, but no "Errors".

Important Note:

Some programmers use the following syntax to specify and use variables:

```
Temp EQU $08
```

```
mov w, Temp
```

This causes the Assembler to generate code which moves the contents of "Temp" into w, and this might be actually what the programmer intended to do. SXCheck, on the other hand, considers that "Temp" is a constant because it was defined together with an EQU directive, and therefore will generate a warning like "2nd OP is a constant w/o leading '#". Therefore, you better use the DS directive to define space for variables instead of "EQU" or "=" to define the addresses of variables, although this is not incorrect.

2nd OP is a bit

This warning will be generated together with ADD, AND, OR, SUB, XOR, MOV, CJA, CJAE, CJB, CJBE, CJE, CJNE, CSA, CSAE, CSB, CSBE, CSE, or CSNE instructions when the second operand is the symbolic name of a bit, like in

```
TxDPin = RA.0
```

```
mov w, TxDPin
```

1st OP is not a bit

This warning will be generated together with CLRB, SETB, JB, JNB, SB, or SNB instructions when the operand is not a bit, like in

```
ORG $08
Flags DS 1
Done = Flags.0
```

```
clrb Flags ; causes a warning
clrb Flags.0 ; is OK
clrb Done ; is OK
```

This warning will also be issued together with MOVB instructions, when the first (target) operand is not a bit.

2nd OP is not a bit

This warning will be generated together with ADDB, and SUBB instructions, when the second operand is not a bit like in

```
InPort = rb
CountPin = InPort.0
```

```
Counter ORG $08
DS 1
```

```
addb Counter, InPort      ; causes a warning
addb Counter, InPort.0    ; is OK
addb Counter, CountPin    ; is OK
```

This warning will also be issued together with MOVB instructions, when the second (source) operand is not a bit.

What SXCheck can and cannot do

As described before, SXCheck makes certain assumptions concerning what symbols name variables, constants, and bits, that base upon common SX coding style, but it may not always detect “suspicious” code on one hand, and on the other hand it may generate warning messages on code lines that perform exactly as intended.

Therefore, SXCheck is not the right tool to convert bad code into good one, it is just an aid to inspect your code, and I hope it will be helpful when you have trouble with some piece of code.

Of course, the best situation is when SXCheck’s list box only shows the message:

“No warnings to report”.

But be careful, this only means that SXCheck has no problems found, like the following story will tell you:

Once, two programming errors met in a program. The first one said: “My programmer is trying to fix me since a week”, and the second one said: “Ha, my programmer does not even know that I exist!”

Good luck, and have fun,

Guenther