High Level Assembler Plugin

User documentation

Michal Bali, Marcel Hruška, Peter Polák,
Adam Šmelko, Lucia Tódová

Supervisor: Miroslav Kratochvíl

Consultant: Slavomír Kučera (Broadcom)

Contents

HLASM Language Support is an extension that supports the High Level Assembler language. It provides code completion, highlighting and navigation features, shows mistakes in the source, and lets you trace how the conditional assembly is evaluated with a modern debugging experience.

This extension is a part of the Che4z open-source project.

1. Getting Started

To start using the HLASM Language Support extension, follow these steps:

1. Install the extension either from Visual Studio Marketplace[1] or from source as described in chapter 10 of the project documentation.
2. In **File** - **Open Folder...**, select the folder where your HLASM project is located.
3. Open your HLASM source code (no file extension is needed) or create a new file.
4. If the extension fails to auto-detect HLASM language, set it manually in the bottom-right corner of the VS Code window.
5. The extension is now enabled on the opened file. If you have macro definitions in separate files or use the COPY instruction, proceed with the steps below to configure the extension to search for external files in the correct directories:
6. After opening the HLASM file, two popups display. Select `Create pgm_conf.json with current program` and `Create empty proc_grps.json`. The two configuration files are created in the *.hlasmplugin* subfolder.
7. In the *proc_grps.json* file, fill the *libs* array with paths to folders with macro definitions and COPY files. For example, if you have your macro files in the *ASMMAC/* folder, type the string *"ASMMAC"* into the libs array.

There is an example workspace in the folder `example_workspace` that can be used to test out the extension.

For a full explanation of the configuration, see the chapter 4.

[1] `https://marketplace.visualstudio.com/items?itemName=broadcomMFD.hlasm-language-support`

2. Functionality of extension

The HLASM Language Support extension parses and analyzes all parts of a HLASM program. It resolves all ordinary symbols, variable symbols and checks the validity of most instructions. The extension supports conditional and unconditional branching and can define global and local variable symbols. It can also expand macros and COPY instructions.

2.1 Highlighting

The HLASM Language Support extension highlights statements with different colors for labels, instructions, operands, remarks and variables. Statements containing instructions that can have operands are highlighted differently to statements that do not expect operands. Code that is skipped by branching AIF, AGO or conditional assembly is not colored.

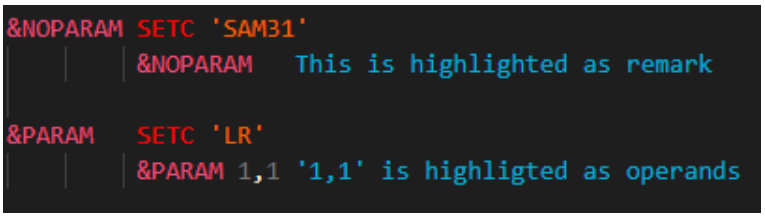

Figure 2.1: An example of highlighting.

2.2 Autocomplete

Autocomplete is enabled for the instruction field. While typing, a list of instructions starting with the typed characters displays. Selecting an instruction from the list completes it and inserts the default operands. Variables and sequence symbols are also filled with a value from their scope.

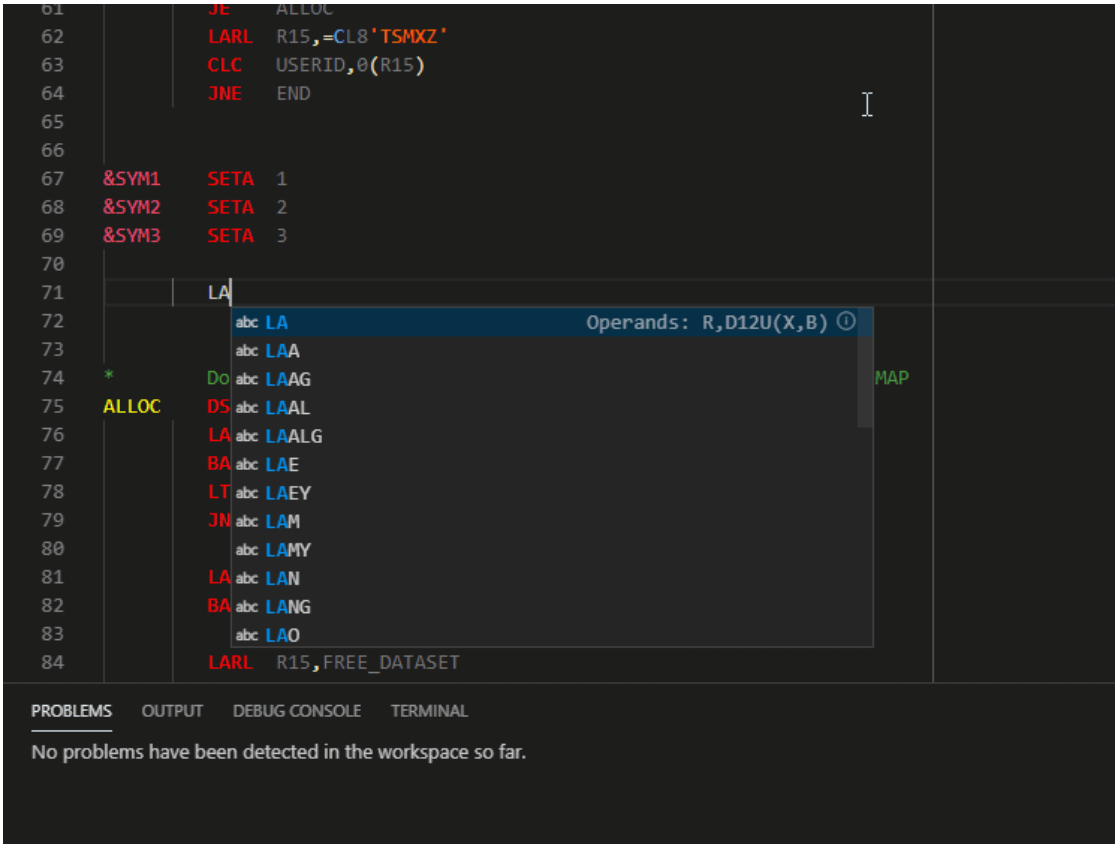

Figure 2.2: Autocomplete usage example.

2.3 Go To Definition and Find All References

The extension adds the functionality of `go to definition` and `find all references`. Use the `go to definition` functionality to show definitions of variable symbols, ordinary symbols and macros, or open COPY files directly. Use the `find all references` functionality to show all places where a symbol is used.

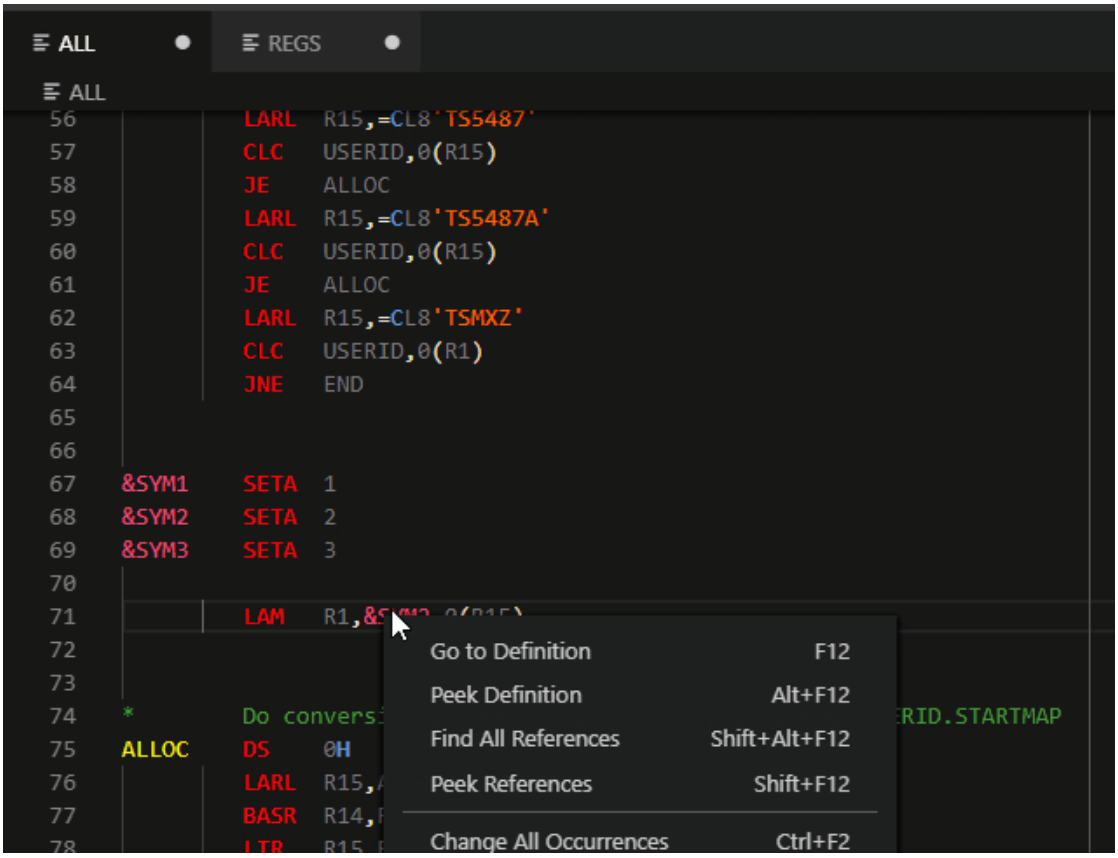

Figure 2.3: Go To Definition usage example.

3. Macro Tracer

The macro tracer functionality allows you to track the process of assembling HLASM code. It lets you see step-by-step how macros are expanded and displays values of variable symbols at different points during the assembly process. You can also set breakpoints in problematic sections of your conditional assembly code.

The macro tracer is not a debugger. It cannot debug running executables, only track the compilation process.

3.1 Configuring the Macro Tracer

1. Open your workspace.
2. In the left sidebar, click the bug icon to open the debugging panel (Ctrl + Shift + D).
3. Click create a launch.json file. A `select environment` prompt displays.
4. Enter **HLASM Macro tracer**. Your workspace is now configured for macro tracing.

3.2 Using the Macro Tracer

To run the macro tracer, open the file that you want to trace. Then press **F5** to open the debugging panel and start the debugging session.

When the tracer stops at a macro or COPY instruction, you can select **step into** to open the macro or COPY file, or **step over** to skip to the next line.

Breakpoints can be set before or during the debugging session.

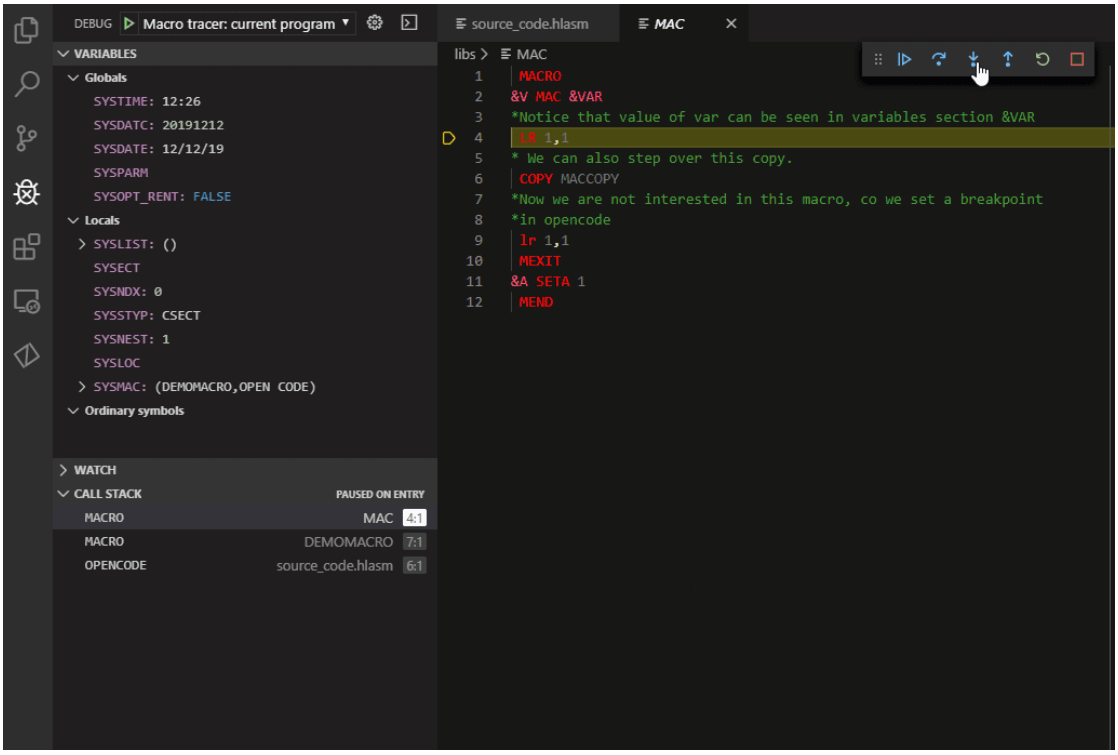

Figure 3.1: Macro Tracer usage example.

4. External Macro Libraries and COPY Members

The HLASM Language Support extension looks for locally stored members when a macro or COPY instruction is evaluated. The paths of these members are specified in two configuration files in the *.hlasmplugin* folder of the currently open workspace. Ensure that you configure these files before using macros from separate files or the COPY instruction.

When you open a HLASM file or manually set the HLASM language for a file, you can choose to automatically create these files for the current program.

The structure of the configuration is based on CA Endevor® SCM. *proc_grps.json* defines processor groups by assigning a group name to a list of directories which are searched in the order they are listed. *pgm_conf.json* provides mapping between source files (open code files) and processor groups. It specifies which list of directories is used with which source file. If a relative source file path is specified, it is relative to the current workspace.

Listing 1 This example defines two processor groups, GROUP1 and GROUP2, and a list of directories to search for macros and COPY files.

```
{
  "pgroups": [
    {
      "name":"GROUP1",
      "libs": [
        "ASMMAC/",
        "C:/SYS.ASMMAC"
      ]
    },
    {
      "name":"GROUP2",
      "libs": [
        "G2MAC/",
        "C:/SYS.ASMMAC"
      ]
    }
  ]
}
```

If you have the two configuration files configured as above and invoke the MAC1 macro from *source_code.hlasm*, the folder *ASMMAC/* in the current workspace is searched for a file with the exact name MAC1. If that search is unsuccessful the folder *C:/SYS.ASMMAC* is searched. If that search is unsuccessful an error displays that the macro does not exist.

Listing 2 The following example specifies that GROUP1 is used when working with *source_code.hlasm* and GROUP2 is used when working with *second_file.hlasm*

```
{
  "pgms": [
    {
      "program": "source_code",
      "pgroup": "GROUP1"
    },
    {
      "program": "second_file",
      "pgroup": "GROUP2"
    },
  ]
}
```

There is also the option *alwaysRecognize* which takes an array of wildcards. It allows you to configure two things:

- All files matching these wildcards will always be recognized as HLASM files.
- If an extension wildcard is defined, all macro and copy files with such extension may be used in the source code. For example, with the extension wildcard **.hlasm*, a user may add macro MAC to his source code even if it is in a file called *Mac.hlasm*.

Listing 3 In this example, GROUP1 is used for all open code programs.

```
{
  "pgms": [
    {
      "program": "*",
      "pgroup": "GROUP1"
    }
  ]
}
```