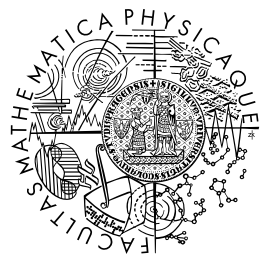

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

High Level Assembler Plugin

Detailed Specification

Authors: Bc. Michal Bali, Bc. Marcel Hruška, Bc. Peter Polák, Bc. Adam Šmelko, Bc. Lucia Tódová

Supervisor: RNDr. Miroslav Kratochvíl

Consultant: Ing. Slavomír Kučera

Prague 2019

Contents

Introduction

Related Work

1. IBM High Level Assembler Language overview

In general, high-level assemblers provide for their assembly languages features that are commonly found in high-level programming languages. Hence, in addition to ordinary machine instructions they also contain control statements similar to *if, while, for* as well as custom callable macros.

IBM High Level Assembler (HLASM) comforts this definition and adds other features which will be described in this chapter.

1.1 Syntax

HLASM has somehow complicated syntax.

1.1.1 Statement

HLASM program is sequence of *statements*. Statement consists of four fields. Those are:

- **Name field** — Serves as place for named constants that are to be used in code.
- **Operation field** — Instruction that is executed.
- **Operands field** — Field for instruction operands separated by comma.
- **Remark field** — Serves as line commentary.

```
 label    instruction     operands             remarks
.NOMOV        AGO      (&WH).L1,.L2,.L3     SEQUENTIAL BRANCH
```

1.1.2 Continuation

One line in HLASM source code can contain only up to 80 characters. However, sometimes statement is too long to be written in one line. Therefore, special handling is introduced called **continuation**.

For indication that statement continues on the next line a character other than space is placed in **continuation column** (default is 72). Then the remainder of the statement must start on **continue column** (default is 15) to finally create a well formed statement (see 1.1).

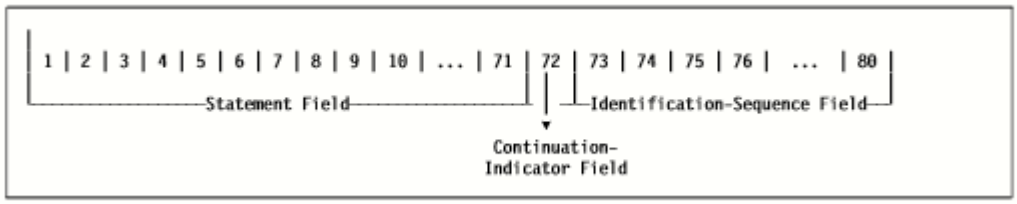

Figure 1.1: Description of line columns.

1.2 Semantics

1.2.1 Conditional assembly

1.2.2 Ordinary assembly

2. Requirements

This chapters outlines the main features that the project should implement to be a working product. All the features below have been discussed with professional HLASM developers and have been agreed on by the company.

The final product is a VS Code extension, downloadable from the Market Place. The extension contains all executables/libraries that are needed for the project to work correctly on the most popular platforms. No other prerequisites should be needed.

Modularity of the project is another important requirement.

The parsing library provides 2 kinds of API: complex one that mimicks the LSP specification and a simple one that accepts text and dependency-resolver object and returns diagnostics.

The language server implements the LSP standard, hence it can be easily reintegrated within other IDEs such as Eclipse Che.

2.1 Language features

This part provides a brief overview of the parts of HLASM that will be implemented in the project.

First of all, the project recognizes the syntax of HLASM (including continuations) and parses it into predefined structures.

It is able to interpret high-level parts of the assembler such as the conditional assembly instructions for the code generation, assembler instruction for the code layout determination and the macro expansion.

For the dependency search, a configuration file is necessary to provide locations of depending files. This configuration procedure is similar to the one used on real mainframes.

On top of that, all instructions (including machine instructions) are being semantically and syntactically checked for correct operand format usage. However, the operand contents are not being checked as there is no machine instruction interpretation.

2.2 LSP features

This section demonstrates the possible uses of the extension on the client side. LSP provides a list of well-defined features. The project implements the following:

- Go to definition for all symbols, macro definitions and copy members
- Find all references for all symbols, macro definitions and copy members
- Completion for instructions, defined symbols and macros
- Hover for symbol attributes, their locations, contents and other useful information depending on the symbol type
- Diagnostics for syntax and semantic errors and warnings
- (Extra to LSP) Server-side Highlighting for all symbols

The highlighting is not a standard part of the LSP, nonetheless it is a needed addition. Due to the complexity of HLASM, a typical syntax highlighting is not sufficient.

3. Architecture

-JNI? asi by som nespominal
mirko:
a je fajn rozepsat vsechny API a takovy veci co sou po ceste

3.1 Parser library

3.1.1 Workspace manager

3.1.2 Analyzer

Parser

Processing

Checking

3.1.3 Debugger

3.2 Language server

3.3 VS code client

4. Technologies

mirko: soupis konkretnich technologii a verzi

5. Project execution

mirko:
milestony
gantt
prirazeni lidi k projektum
udelejte si cas na psani dokumentace
je fajn mit contingency plan, co delat kdyz se to dojebe nebo ltery ficury jsou jak prioritni

Conclusion

Future work