High Level Assembler Plugin
Project specification

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

Supervisor: Miroslav Kratochvíl

Contents

Introduction

Related Work

1. HLASM 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 fig. 1.1).

1.2 Semantics

1.2.1 Conditional assembly

1.2.2 Ordinary assembly

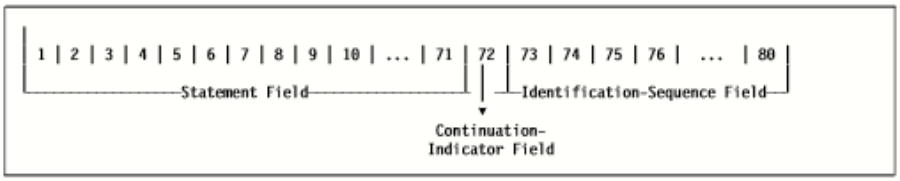

Figure 1.1: Description of line columns.

2. Requirements

-co ten nas produkt ma byt vseobecne zhrnutie
...je to extension ... doda support pre ... -cela tato sekcia uz je popisana niekde na CA wiki, mozno dobry zaklad

2.1 Language features

-zoznam veci jazyka co podporujeme

2.2 LSP features

-working plugin for vs code

- Go to definition for all symbols, macro definitions and copy members.
- Find all references
- Completion for instructions, defined symbols and macros
- Highlighting
- Hover

-non functional requirement - api kniznice??

3. Architecture

-JNI? asi by som nespominal
mirko:
a je fajn rozepsat vsechny API a takovy veci co sou po ceste
–velky graf vsetkych komponent –ku kazdemu odstavcek

3.1 Parser library

3.1.1 Workspace manager

3.1.2 Analyzer

Lexer

Parser

Processing

Checking

3.1.3 Debugger

3.2 Language server

3.3 VS code client

4. Technologies

mirko: soupis konkretnich technologii a verzi antlr cmake jenkins json lib boost asio? docker

vscode theia che produkcne zdrojaky poskytnute broadcom google test –jenkins sa opytat ako s tym ze to nie je nase

jazyky typescript c++ cmake

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
1. mesiac research jazyk 8t research zvolit parser 4t research zvolit ide 4t
2. mesiac implementacia LSP POC VSCode klient POC lexer 6 parser 8t cmake 1t
3. mesiac do klienta semanticky highlighting assembler checker conditional assembly instructions expressions debugger POC
4. mesiac CA LSP features machine instruction checker 12t macro expansion
5. copy 4t machine expression 4t client-server continuation handling
6. DC ordinary 8t diagnostikz
7. ORDINARY LSP features code coverage 8t
8. benchmark testing 8t
9. dokumentacia