NOTE: This is a draft and does not represent the official opinion of the parser team!

Kürzen!

1 Parser-Modul

Bitte berücksichtigen: Es kann noch Änderungen hieran geben, da dies nur eine Präsentation des aktuellen Standes ist...

1.1 Funktionalität

Das Parser-Modul liest den gegebenen Assembler-Code, führt alle gegebenen Compiler-Direktiven aus und generiert für jedes Argument einen Syntax-Baum, der an die Architektur übergeben wird. Außerdem reserviert er den über die Reservierungs- und Definierungsdirektiven festgelegten Speicher.

Somit entspricht der Parser dem Assemblierer. Die Trennung von Architektur und Parser wurde vorgenommen, um Code-Duplikation zu vermeiden, die bei Architekturen mit mehreren Dialekten auftreten könnten (bei X86 z.B. AT&T- und Intel-Syntax).

1.2 Umsetzung (Referenzmodul)

Das Referenzmodul (welches im Rahmen dieses Großpraktikums konstruiert werden soll) wird als bekannter 2-Pass-Assembler realisiert.

1.2.1 0. Pass (eventuell)

Zuvor werden alle Kommentare entfernt. Dieser Schritt ist nur notwendig, falls der RISC-V-Assembler mehrzeilige Kommentare unterstützen soll.

1.2.2 1. Pass

Im ersten Schritt wird der rohe Assemblertext gelesen und in Zeilenbereiche unterteilt, die jeweils einen Befehl mit allen zugehörigen Marken beinhalten. Diese Befehle werden anschließend in Objekte gepackt (mit Zeilenintervall und Datei des Auftretens), wobei nach Direktiven und den eigentlichen Befehlen unterschieden wird. Besonders bei ersteren wird bereits hier genauer unterschieden. Alle Labels und eventuell auch Konstantennamen (die wie Labels behandelt werden) werden die Symboltabelle gepackt mit ihrem korrespondierenden (Text-)Wert.

Wir sollten uns überlegen, ob wir die Zeilennummern auch (wg. Einheitlichkeit) zu Strings konvertieren und dann wieder zurücklesen... Sonst könnte man auch zwischen 1. und 2. Pass einfach die Symboltabelle parsen...

Wenn von außen gewünscht (also wenn das Programm gerade nicht läuft), wird hier der Speicher reserviert, sonst werden nur die entsprechenden Positionen berechnet. Bei allen Makros wird Anfang und Ende erfasst und diese in eine separate Makroliste eingetragen.

1.2.3 2. Pass

Wir haben nun eine Befehlsfolge von Objekten. In diesem Schritt werden nun die Direktiven ausgeführt und die Labels mit ihrem Wert eingesetzt. Aus jedem der Argumente wird ein Syntaxbaum gebildet. Wegen der Ambivalenz der Architekturen bezüglich Datentypen usw. wird hier eine Factory von Knoten von der Architektur bereitgestellt. Als Ausgabe dient schließlich eine Liste von Objekten **ohne** jegliche Direktiven und mit Argumenten als Syntaxbäumen. Diese wird dann dem Core zur Verfügung gestellt.

1.3 Abhängigkeiten

Der Parser kommuniziert mit allen anderen Modulen über den Core und ist somit alleine von diesem abhängig. Die Architektur erfragt vom Parser die einzelnen assemblierten Zeilen, um diese auszuführen. Die GUI erhält vom Parser die Fehlermeldungen und stellt die Anfragen, um Code zu übersetzen.

Wir haben uns dazu entschieden, vorerst die boost::spirit-Bibliothek zu verwenden, um Ausdrücke zu parsen. Eigentlich ist es auch möglich, selber einen Parser zu schreiben (z.B. mit dem Packrat-Algorithmus für Ausdrücke, die sicherlich am schwierigsten zu parsen wären, weil kontextfrei), besonders da wir komplizierte Präprozessoren und bedingte Assemblerkompilierung ausschließen und Assemblertext allgemein nicht sonderlich komplex ist. Dennoch erachten wir es als notwendig, gerade im Bezug auf die Stärke des Parser-Teams (zwei Personen) und der verfügbaren Zeit.

1.4 Verworfen: Allgemeiner Parser

Im Laufe unserer Entscheidungsfindung stand auch kurz auf dem Plan, einen allgemeinen Parser mit einem kleinen dialektabhängigen Modul zu entwerfen. Dies hätte den Vorteil, dass die Dialektmodule wesentlich kleiner und einfacher zu programmieren ausgefallen wären, da die Hauptarbeit ja sowieso der große allgemeine Parser übernimmt. Jedoch haben wir diese Idee für dieses Großpraktikum verworfen, da hierbei zu viele Komplikationen entstanden (wie will man fast alle Assemblersprachen verallgemeinern?). Sie steht jedoch für spätere Projekte frei, verwendet zu werden.