Backend

Nijat Aghayev

Einleitung

Die Hauptaufgabe der Backend war die vom Team Transformation erstellte LLVM Bitcode Datei (mit .bc Dateiendung) in die MSP430 Assembler Datei zu transformieren (mit .asm Dateiendung).

Programmverlauf

Das Program fordert eine Eingabedatei (als LLVM Bitcode Datei) , liest diese Datei und setzt voraus , dass die LLVM IR Instruktionen in dieser Datei mit entsprechenden Assembler Codes ausgestattet sind. Es iteriert über die Funktionen und Basic Blocks der Funktionen. In jedem Iterationsschritt werden die Metadaten aus LLVM IR Instruktionen gelesen und in die Ausgabedatei geschrieben.

Wenn die Metadaten „NOP“ sind , wird eine Folge der zufälligen Assembler Codes generiert, die Anzahl der Zeile dieser Folge entspricht dem zweiten Operand der Metadaten.

Wenn die LLVM IR Instruktion „BR“ ist , wird eine „JMP“ Assembler Instruktion erstellt , diese „JMP“ Instruktion spring zum nächsten Basic Block über.

Alle „JMP“ Instrutionen in der LLVM Bitcode Datei werden komplett ignoriert.

Details über Funktionen

Es gibt insgesamt 7 Funktionen in Backend.cpp . In folgenden werden auf die Details dieser Funktionen eingegangen.

1) **bool generateASM(const std::string &FileName);**

Eingabeparameters:

- FileName als konstante string Referenz

Rückgabetyp:

- bool

Die wichtigste Logik des Programms ist in dieser Funktion zu finden. Diese Funktion liest die Eingabedatei (LLVM Bitcode Datei) , iteriert über die Funktionen und die Basic Blocks der Funktionen , liest die Metadaten der Basic Blocks und schreibt sie in die Ausgabedatei. Wenn kein Fehler auftritt , wird true zurückgegeben, sonst false wird zurückgegeben.

2) **void replaceAll(std::string &s , const std::string &search, const std::string &replace);**

Eingabeparameters:

- s als string Referenz
- search als konstante string Referenz
- replace als konstante string Referenz

Rückgabetyp:

- void

Diese Funktion findet und ersetzt alle Vorkommen des strings „search“ mit dem string „replace“ im string „s“.

3) **int getBBCount(std::unique_ptr<Module> &m);**

Eingabeparameters:

- m als std::unique_ptr<Module> Referenz

Rückgabetyp:

- int

Diese Funktion zählt die Anzahl der Basic Blocks im LLVM Module „m“ durch und gibt das Ergebnis zurück.

4) **void generateRandomCodeForNOP(int numberOfIRForInstr, raw_fd_ostream &asmOutput, std::string &indent);**

Eingabeparameters:

- numberOfIRForInstr als int
- asmOutput als raw_fd_ostream Referenz
- indent als string Referenz

Rückgabetyp:

- void

Diese Funktion generiert eine Folge der zufälligen Assembler Instruktionen für „NOP“. Die Anzahl der Zeilen der Folge entspricht dem Eingabeparameter „numberOfIRForInstr“. Die erstellte Folge wird in die „asmOutput“ geschrieben.

5) **std::pair<std::string,int> generateRandomRegAndValue();**

Eingabeparameters:

Rückgabetyp:

- std::pair<std::string,int>

Diese Funktion erstellt ein zufälliges Register von „r0“ durch „r15“ , eine zufällige Integer Zahl zwischen 1 und 255 und packt sie zusammen als std::pair<std::string,int> und gibt dieses pair zurück.

6) **int randNum(int min, int max);**

Eingabeparameters:

- min als int
- max als int

Rückgabetyp:

- int

Diese Funktion erstellt eine zufällige Integer Zahl zwischen „min“ und „max“ (beide einschließlich) und gibt das Ergebnis zurück.

7) **int main(int argc, char **argv);**

Eingabeparameters:

- argc als int
- argv als char**

Rückgabetyp:

- int

Diese Funktion ist Einsprungpunkt des Programms, sie ruft die Funktion generateASM() auf. Wenn etwas schief geht , wird Rückgabewert -1 zurückgegeben, sonst 0 wird zurückgegeben.

Ausführung des Programms

```
mkdir build
cd build
cmake .. -Gninja
ninja
bin/backend ../test/test1.out.bc
```

Die obige Ausführung wird eine Datei ../test/test1.out.asm generieren.