Relatório EA876 - Trabalho 1

Larissa Medeiros (178014) e Rafael Gonçalves (186062)

Introdução

O objetivo do trabalho foi fazer um compilador de expressões matemáticas para código assembly executável em processador de arquitetura ARM usando as linguagens Lex e Yacc.

A nossa abordagem consistiu em criar uma sintaxe livre de contexto que cobrisse todos os casos de expressões matemáticas a serem tratados e então resolver os casos base.

Método

Selecionamos quais elementos precisariam ser identificados pelo Lex para a criação de tokens que seriam usados no Yacc. As regras no Yacc foram definidas segundo a seguinte gramática livre de contexto:

1. S → E
2. E → N
3. E → (E)
4. E → E Op E
5. Op → +
6. Op → -
7. Op → *
8. N → [0-9]*
9. N → \(\-[0-9]*\)

As principais expressões tratadas foram as que se resolvem para o símbolo E, já que isso significa efetivamente a resolução de expressões de forma recursiva.

A primeira expressão da gramática apenas define que o programa se resume a um símbolo intermediário de expressão, necessariamente. As rotinas de resolução dos casos 3, 5, 6, 7, 8 e 9 são simplesmente atribuições. Os demais casos imprimem instruções assembly, como mostradosa seguir:

Caso 2:

```
mov r0, #N
str r0, [SP], $4
```

Caso 4:

```
ldr r2, [SP, #-4]!
ldr r1, [SP, #-4]!
OP r0, r1, r2
str r0, [SP], #4
```

Em que OP pode ser: add, mul ou sub, N é um número inteiro e SP é o stack pointer, no nosso caso o r3.

Tratamos o problema do armazenamento dos operandos na própria linguagem assembly. Utilizamos os comandos STR para armazenar o resultado de cada expressão intermediária na pilha e LDR para carregar resultados da pilha nos registradores utilizados nas operações.

Resultados

Como proposto, nosso programa não resolve as expressões em C, mas sim imprime o código assembly que resolve a expressão.

Não conseguimos resolver o problema de identificar quaisquer números negativos. A implementação atual admite que números negativos estarão sempre entre parênteses.

Segue o código produzido pelo exemplo:

$$(-5) * (5 + 2) - 7$$

```
mov r0, #-5
push r0
mov r0, #5
push r0
mov r0,#2
push r0
pop r2
pop r1
add r0, r1, r2
push r0
pop r2
pop r1
mul r0, r1, r2
push r0
mov r0, #7
push r0
pop r2
pop r1
sub r0, r1, r2
push r0
```