

# 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. $S \rightarrow E$      | 6. $Op \rightarrow +$      |
| 2. $E \rightarrow N$      | 7. $Op \rightarrow -$      |
| 3. $E \rightarrow (E)$    | 8. $Op \rightarrow *$      |
| 4. $E \rightarrow -E$     | 9. $N \rightarrow [0-9]^*$ |
| 5. $E \rightarrow E Op E$ |                            |

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, 4, 6, 7, 8 e 9 são simplesmente atribuições. Os demais casos imprimem instruções assembly, como mostrados a seguir:

### Caso 2:

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

### Caso 5:

```
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 r10.

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.

Segue o código produzido pelo exemplo:

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

```
mov r10, #3200      bl mul
mov r0, #5           str r0, [r10], #4
str r0, [r10], #4    end
ldr r1, [r10, #-4]!
sub r0, #1, r1        mul
str r0, [r10], #4     ldr r2, [r10, #-4]!
mov r0, #5            ldr r1, [r10, #-4]!
str r0, [r10], #4     mov r0, #0
mov r0, #2
str r0, [r10], #4     loop
ldr r2, [r10, #-4]!   cmp r1, #0
ldr r1, [r10, #-4]!   ble endmul
add r0, r1, r2         add r0, r0, r2
str r0, [r10], #4     sub r1, r1, #1
mov r0, #7            b loop
str r0, [r10], #4
ldr r2, [r10, #-4]!   endmul
ldr r1, [r10, #-4]!   mov pc, lr
sub r0, r1, r2
str r0, [r10], #4
```