RELATÓRIO EA876 - TRABALHO 1

Eduardo Parducci[1] e Henrique Roberto da Cunha[2]

I. Introdução

O objetivo deste trabalho é, usando lex/yacc, transformar uma expressão matemática, que é a entrada, em um código assembly que calcule o resultado dessa expressão.

II. Método

Com o exemplo de calculadora fornecido pelo professor, já temos as leis de prioridade básicas de expressões mátemáticas, como por exemplo a prioridade que os parentesis tem sobre as operações e a prioridade que a multiplicação tem sobre soma, subtração, etc.

Para a identificação de números, além de definir a regra lex [0-9]+ INT, utilizamos uma regra de linguagem livre de contexto para tratar os números negativos da seguinte forma:

```
NUMERO:
 INT {
  $$ = $1;
}
 | SUB NUMERO {
  $$ = $2*(-1);
}
 | SOMA NUMERO {
  $$ =$2;
}
```

Outra regra básica definida foi baseado em como a entrada pode ser também uma resposta final, sempre carregamos ela em r0.

Ao encontrar as expressões, sejam elas quais forem executamos um comando de printf, de maneira a construir o código assembly que executa essa operaçào. O resultado dessa operação sempre será armazenado na pilha.

As expressões de soma e subtração são, no geral, bem parecidas, de forma que ao encontrarmos dois números A e B que já estão empilhados executamos o comando de printf gerando o código:

```
ldmfd sp!, {r1}
ldmfd sp!, {r2}
add/sub r0, r1, r2
stmfd sp!, {r0}
ldmfd sp!, {r0}
```

Para o caso da multiplicação é um pouco diferente, para conseguirmos armazear todos os rótulos de multiplicação de forma a ter todas as operações executáveis, criamos um rótudo diferente para cada multiplicação, de forma que no printf é representado por m%d_4 de forma que o %d é o parâmetro que irá diferenciar cada rótulo de multiplicação. Isso é feito incrementando uma variável toda vez que encontramos essa operação.

Outro problema encontrado, é a multiplicação de um operando A>0 por um B<0, pois este é executado de forma diferente de uma operação A<0 e B>0. Isso foi resolvido criando apenas uma condição para verificar se esse algarismo B é menor que 0 e o tratar de forma diferente, novamente com um rótulo específio de mesmo valor da multiplicação.

III. Conclusão

Os resultados foram atingidos com sucesso, ou seja, as expressões matemáticas que foram colocadas geram os trechos de código esperado que calcula corretamente a expressão.

Por exemplo a entrada (2+1)*(-3+2) gera o código:

```
  mov sp, #0x200
  mov r0, #2
  stmfd sp!, {r0}
  mov r0, #1
  stmfd sp!, {r0}
  ldmfd sp!, {r1}
  ldmfd sp!, {r2}
  add r0, r1, r2
  stmfd sp!, {r0}
  mov r0, #-3
  stmfd sp!, {r0}
  mov r0, #2
  stmfd sp!, {r0}
  ldmfd sp!, {r1}
  ldmfd sp!, {r2}
  add r0, r1, r2
  stmfd sp!, {r0}
  mov r2, #0
  mov r0, #0
  ldmfd sp!, {r1}
  ldmfd sp!, {r3}
m0_b
  cmp r2, r1
  beq m0_e
  cmp r1, #0
  blt sub0
  add r2, r2, #1
  add r0, r0, r3
  b m0_b
sub0
  sub r2 ,r2,#1
  sub r0,r0,r3
  b m0_b
m0_e
  stmfd sp!, {r0}
  ldmfd sp!, {r0}
```

De forma que colocando em um simulador assembly, temos o resultado esperado da operação.