RELATÓRIO EA876 - TRABALHO 1

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

I. INTRODUÇÃO

O objetivo deste trabalho é realizar a compilação de linguagem matemática para ARM *Assembly*, por meio das ferramentas *lex/flex yacc/Bison* e a linguagem C, sendo entrada uma expressão matemática e gerando uma saída correspondente ao código assembly que calcula o resultado dessa expressão.

II. MÉTODO

Utilizamos como base as regras de formação e expressões regulares do exemplo de calculadora apresentado em sala de aula. Às expressões regulares, acrescentamos a regra de identificação do caractere "-" que gera o token SUB, utilizado para a implementação da operação de subtração, bem como a operação com valores negativos. Para as regras de formação, projetamos uma estrutura denominada NUMERO, cujas regras podem ser denotadas por:
NUMERO $\rightarrow$ INT
NUMERO $\rightarrow$ SUB NUMERO
NUMERO $\rightarrow$ SOMA NUMERO
Assim é possível fazer a identificação de sinais sucessivos (Exemplo 1: " - - 2"). A estratégia que utilizamos para que o código *Assembly* gerado execute as operações respeitando as regras de precedência foi utilizar a pilha do *ARM* através das instruções *STMFD* e *LDMFD*, dessa forma, cada NUMERO encontrado é empilhado, para o caso do Exemplo 1, o seguinte código será gerado:
mov r0, #2
stmfd sp!, r0
Dessa forma, todos os operandos das operações implementadas ficam empilhados, ou seja, ao implementarmos o código gerado pela identificação de uma operação, desempilhamos seus operandos para $r1$ e $r2$, operamos seus valores e empilhamos o resultado ao final. Por utilizarmos a pilha, inicializamos a mesma no início do arquivo *Assembly* da seguinte forma:
mov sp, #0xF00
Ao final da identificação das operações, desempilhamos um valor da pilha, que corresponde ao resultado da expressão de entrada, carregando seu valor em r0 da seguinte forma:
ldmfd sp!, r0
Para a implementação da multiplicação, o programa gerencia os rótulos criados para os laços com um contador, de forma que a primeira multiplicação encontrada (caso exista) irá gerar os rótulos de início e fim: $m0\_b$ e $m0\_e$ respectivamente, a segunda $m1\_b$ e $m1\_e$ e assim sucessivamente. 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.