Etapa 6: Geração de Código Assembly

Aluno: Victor dos Santos Melo

Matrícula: 00285640

Resumo

O código foi implementado para o processador Intel Core i7, disponível no MacBook Pro. Parece ser a mesma linguagem assembly usada no macbook do professor.

O código desta etapa foi implementado no arquivo `assembler.c`. Segue abaixo uma lista de informações que julguei relevantes sobre o desenvolvimento dessa etapa.

1. A função `print` está imprimindo floats como inteiros. Para corrigir, bastaria alterar para usar as instruções corretas do assembly. Porém, irei deixar isso para a próxima etapa, por conta da falta de tempo.
2. Tive que resolver erros de etapas anteriores, porém houve um problema com minha definição da TAC que atrapalhou muito o desenvolvimento dessa etapa, tomando muito tempo pra eu identificar o problema. No caso, eu não estava, na hora de chamar a função, armazenando na instrução intermediária qual o label intermediário que armazena a resposta da função. Dessa forma, não era possível obter a resposta da função na hora da chama.
3. Alguns trechos do código foram implementados, porém, durante o desenvolvimento, acabei seguindo outros caminhos e por falta de tempo não consegui remover todos os códigos deixados. Um exemplo é o trecho `printstringstr`, que foi substuído pelo `L_.str.N`, mas não foi removido ainda. Na próxima etapa realizarei essa remoção.

Abaixo seguem quatro casos de uso da linguagem de programação, seu resultado em assembly e o resultado obtido pela sua execução. No início de cada caso descrevo as funcionalidades que quero demonstrar com ele.

Caso 1:

- Demonstração de um caso simples, para visualizar a estrutura geral do assembly.

Código **ere2020:**

```
main () = int {
    print 1
};
```

Código **assembly:**

```
## FIXED INIT
  .section  __TEXT,__cstring,cstring_literals
```

```
  printintstr: .asciz "%d\n"
  printfloatstr: .asciz "%f\n"
  printstringstr: .asciz  "%s\n"

  .section  __TEXT,__text,regular,pure_instructions
.globl _main
# TAC_FBEGIN
_main:
  .cfi_startproc
  pushq %rbp
  movq  %rsp, %rbp
# TAC_PRINT
  movl  _1(%rip), %esi
  leaq  L_.str.0(%rip), %rdi
  callq _printf
# TAC_FEND
  popq  %rbp
  retq
  .cfi_endproc
# DATA SECTION
  .section  __DATA,__data

_0: .long 0
_1: .long 1
L_.str.0: .asciz  "%d\n"
```

Resultado da execução:

```
$ ./a.out
1
```

Caso 2:

- Operações implementadas.
- Números sendo tratados como hexadecimais.

Código **ere2020:**

```
var_10 = int: A;
var_14 = int: E;
var_2 = int: 2;

main () = int {
    print "\nOperações aritméticas:\n"

    print var_14 + var_10
    print var_14 - var_10
```

```
    print var_14 * var_2
    print var_14 / var_2

    print "\nOperações booleanas: \n"

    print TRUE ^ TRUE
    print TRUE ^ FALSE
    print TRUE | FALSE
    print FALSE | FALSE
    print ~TRUE
    print ~FALSE

    print "\n Operações relacionais com números: \n"
    print 25 >= 3
    print 25 > 3
    print 25 <= 3
    print 25 < 3
    print 25 == 3
    print 25 != 3
    print 25 == 25
    print 25 != 25
};
```

Código **assembly:**

```
## FIXED INIT
  .section  __TEXT,__cstring,cstring_literals
  printintstr: .asciz "%d\n"
  printfloatstr: .asciz "%f\n"
  printstringstr: .asciz  "%s\n"

  .section  __TEXT,__text,regular,pure_instructions
.globl _main
# TAC_FBEGIN
_main:
  .cfi_startproc
  pushq %rbp
  movq  %rsp, %rbp
# TAC_PRINT
  leaq  _string_9(%rip), %rsi
  leaq  L_.str.0(%rip), %rdi
  callq _printf
# TAC_ADD
  movl  _var_14(%rip), %eax
  addl  _var_10(%rip), %eax
  movl  %eax, _1temp_0(%rip)
# TAC_PRINT
  movl  _1temp_0(%rip), %esi
```

```
  leaq  L_.str.1(%rip), %rdi
  callq _printf
# TAC_SUB
  movl  _var_14(%rip), %eax
  subl  _var_10(%rip), %eax
  movl  %eax, _1temp_1(%rip)
# TAC_PRINT
  movl  _1temp_1(%rip), %esi
  leaq  L_.str.2(%rip), %rdi
  callq _printf
# TAC_MULT
  movl  _var_14(%rip), %eax
  imull _var_2(%rip), %eax
  movl  %eax, _1temp_2(%rip)
# TAC_PRINT
  movl  _1temp_2(%rip), %esi
  leaq  L_.str.3(%rip), %rdi
  callq _printf
# TAC_DIV_INT
  movl  _var_14(%rip), %eax
  cltd
  idivl _var_2(%rip)
  movl  %eax, _1temp_3(%rip)
# TAC_PRINT
  movl  _1temp_3(%rip), %esi
  leaq  L_.str.4(%rip), %rdi
  callq _printf
# TAC_PRINT
  leaq  _string_10(%rip), %rsi
  leaq  L_.str.5(%rip), %rdi
  callq _printf
# TAC_AND
  movl  _TRUE(%rip), %eax
  movl  _TRUE(%rip), %edx
  andl  %edx, %eax
  movl  %eax, _1temp_4(%rip)
# TAC_PRINT
  movl  _1temp_4(%rip), %esi
  leaq  L_.str.6(%rip), %rdi
  callq _printf
# TAC_AND
  movl  _TRUE(%rip), %eax
  movl  _FALSE(%rip), %edx
  andl  %edx, %eax
  movl  %eax, _1temp_5(%rip)
# TAC_PRINT
  movl  _1temp_5(%rip), %esi
  leaq  L_.str.7(%rip), %rdi
  callq _printf
```

```
# TAC_OR
  movl  _TRUE(%rip), %eax
  movl  _FALSE(%rip), %edx
  orl    %edx, %eax
  movl  %eax, _1temp_6(%rip)
# TAC_PRINT
  movl  _1temp_6(%rip), %esi
  leaq  L_.str.8(%rip), %rdi
  callq _printf
# TAC_OR
  movl  _FALSE(%rip), %eax
  movl  _FALSE(%rip), %edx
  orl    %edx, %eax
  movl  %eax, _1temp_7(%rip)
# TAC_PRINT
  movl  _1temp_7(%rip), %esi
  leaq  L_.str.9(%rip), %rdi
  callq _printf
# TAC_NOT
  movb  _TRUE(%rip), %cl
  xorb  $-1, %cl
  andb  $1, %cl
  movb  %cl, _1temp_8(%rip)
# TAC_PRINT
  movl  _1temp_8(%rip), %esi
  leaq  L_.str.10(%rip), %rdi
  callq _printf
# TAC_NOT
  movb  _FALSE(%rip), %cl
  xorb  $-1, %cl
  andb  $1, %cl
  movb  %cl, _1temp_9(%rip)
# TAC_PRINT
  movl  _1temp_9(%rip), %esi
  leaq  L_.str.11(%rip), %rdi
  callq _printf
# TAC_PRINT
  leaq  _string_13(%rip), %rsi
  leaq  L_.str.12(%rip), %rdi
  callq _printf
# TAC_GE
  movl  _25(%rip), %edx
  cmpl  _3(%rip), %edx
  jge   _1label_0
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_10(%rip)
# TAC_JUMP
  jmp   _1label_1
```

```
# TAC_LABEL
_1label_0:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_10(%rip)
# TAC_LABEL
_1label_1:
# TAC_PRINT
  movl  _1temp_10(%rip), %esi
  leaq  L_.str.13(%rip), %rdi
  callq _printf
# TAC_GT
  movl  _25(%rip), %edx
  cmpl  _3(%rip), %edx
  jg    _1label_2
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_11(%rip)
# TAC_JUMP
  jmp   _1label_3
# TAC_LABEL
_1label_2:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_11(%rip)
# TAC_LABEL
_1label_3:
# TAC_PRINT
  movl  _1temp_11(%rip), %esi
  leaq  L_.str.14(%rip), %rdi
  callq _printf
# TAC_LE
  movl  _25(%rip), %edx
  cmpl  _3(%rip), %edx
  jle   _1label_4
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_12(%rip)
# TAC_JUMP
  jmp   _1label_5
# TAC_LABEL
_1label_4:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_12(%rip)
# TAC_LABEL
_1label_5:
# TAC_PRINT
  movl  _1temp_12(%rip), %esi
```

```
  leaq  L_.str.15(%rip), %rdi
  callq _printf
# TAC_LT
  movl  _25(%rip), %edx
  cmpl  _3(%rip), %edx
  jl    _1label_6
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_13(%rip)
# TAC_JUMP
  jmp   _1label_7
# TAC_LABEL
_1label_6:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_13(%rip)
# TAC_LABEL
_1label_7:
# TAC_PRINT
  movl  _1temp_13(%rip), %esi
  leaq  L_.str.16(%rip), %rdi
  callq _printf
# TAC_EQ
  movl  _25(%rip), %edx
  cmpl  _3(%rip), %edx
  je    _1label_8
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_14(%rip)
# TAC_JUMP
  jmp   _1label_9
# TAC_LABEL
_1label_8:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_14(%rip)
# TAC_LABEL
_1label_9:
# TAC_PRINT
  movl  _1temp_14(%rip), %esi
  leaq  L_.str.17(%rip), %rdi
  callq _printf
# TAC_DIF
  movl  _25(%rip), %edx
  cmpl  _3(%rip), %edx
  jne   _1label_10
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_15(%rip)
```

```
# TAC_JUMP
  jmp   _1label_11
# TAC_LABEL
_1label_10:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_15(%rip)
# TAC_LABEL
_1label_11:
# TAC_PRINT
  movl  _1temp_15(%rip), %esi
  leaq  L_.str.18(%rip), %rdi
  callq _printf
# TAC_EQ
  movl  _25(%rip), %edx
  cmpl  _25(%rip), %edx
  je    _1label_12
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_16(%rip)
# TAC_JUMP
  jmp   _1label_13
# TAC_LABEL
_1label_12:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_16(%rip)
# TAC_LABEL
_1label_13:
# TAC_PRINT
  movl  _1temp_16(%rip), %esi
  leaq  L_.str.19(%rip), %rdi
  callq _printf
# TAC_DIF
  movl  _25(%rip), %edx
  cmpl  _25(%rip), %edx
  jne   _1label_14
# TAC_MOVE
  movl  _0(%rip), %eax
  movl  %eax, _1temp_17(%rip)
# TAC_JUMP
  jmp   _1label_15
# TAC_LABEL
_1label_14:
# TAC_MOVE
  movl  _1(%rip), %eax
  movl  %eax, _1temp_17(%rip)
# TAC_LABEL
_1label_15:
```

```
# TAC_PRINT
  movl  _1temp_17(%rip), %esi
  leaq  L_.str.20(%rip), %rdi
  callq _printf
# TAC_FEND
  popq  %rbp
  retq
  .cfi_endproc
# DATA SECTION
  .section  __DATA,__data

_3: .long 3
_2: .long 2
_E: .long 14
_A: .long 10
_0: .long 0
_1: .long 1
_25:  .long 37
_1temp_10:  .long 0
_1temp_11:  .long 0
_1temp_12:  .long 0
_1temp_13:  .long 0
_1temp_14:  .long 0
_1temp_15:  .long 0
_1temp_16:  .long 0
_1temp_17:  .long 0
_TRUE:  .long 1
_string_9: .asciz "\nOperações aritméticas:\n"
_FALSE: .long 0
_string_13: .asciz  "\n Operações relacionais com números: \n"
_1temp_0: .long 0
_1temp_1: .long 0
_1temp_2: .long 0
_1temp_3: .long 0
_1temp_4: .long 0
_1temp_5: .long 0
_1temp_6: .long 0
_1temp_7: .long 0
_1temp_8: .long 0
_1temp_9: .long 0
_string_10: .asciz  "\nOperações booleanas: \n"
_var_10:  .long 10
_var_14:  .long 14
_var_2: .long 2
L_.str.0: .asciz  "%s\n"
L_.str.1: .asciz  "%d\n"
L_.str.2: .asciz  "%d\n"
L_.str.3: .asciz  "%d\n"
L_.str.4: .asciz  "%d\n"
```

```
L_.str.5: .asciz  "%s\n"
L_.str.6: .asciz  "%d\n"
L_.str.7: .asciz  "%d\n"
L_.str.8: .asciz  "%d\n"
L_.str.9: .asciz  "%d\n"
L_.str.10: .asciz "%d\n"
L_.str.11: .asciz "%d\n"
L_.str.12: .asciz "%s\n"
L_.str.13: .asciz "%d\n"
L_.str.14: .asciz "%d\n"
L_.str.15: .asciz "%d\n"
L_.str.16: .asciz "%d\n"
L_.str.17: .asciz "%d\n"
L_.str.18: .asciz "%d\n"
L_.str.19: .asciz "%d\n"
L_.str.20: .asciz "%d\n"
```

Resultado da execução:

```
Operações aritméticas:

24
4
28
7

Operações booleanas:

1
0
1
0
0
1

 Operações relacionais com números:

1
1
0
0
0
1
1
0
```

Observações

- Este monte de `L.str.` existe no assembly pois foi um teste durante o desenvolvimento. Porém, como o tempo do trabalho ficou curto, acabei não ajustando. Na próxima etapa resolverei isso.

Caso 3:

- Chamada de funções.
- Passagem de parâmetros.
- Print de resultado de funções.

Código **ere2020:**

```
main () = int {
    print funcao_custom(2,4)
};

funcao_custom (a = int, b = int) = int {
    return a * b
};
```

Código **assembly:**

```
## FIXED INIT
  .section  __TEXT,__cstring,cstring_literals
  printintstr: .asciz "%d\n"
  printfloatstr: .asciz "%f\n"
  printstringstr: .asciz  "%s\n"

  .section  __TEXT,__text,regular,pure_instructions
.globl _main
# TAC_FBEGIN
_main:
  .cfi_startproc
  pushq %rbp
  movq  %rsp, %rbp
# TAC_ARG
  movl  _2.0(%rip), %eax
  movl  %eax, _b(%rip)
# TAC_ARG
  movl  _6.0(%rip), %eax
  movl  %eax, _a(%rip)
# TAC_FCALL
  callq _funcao_custom
# TAC_PRINT
  movl  _1temp_0(%rip), %esi
  leaq  L_.str.0(%rip), %rdi
```

```
  callq _printf
# TAC_FEND
  popq  %rbp
  retq
  .cfi_endproc
# TAC_FBEGIN
_funcao_custom:
  .cfi_startproc
  pushq %rbp
  movq  %rsp, %rbp
# TAC_MULT
  movss _a(%rip), %xmm0
  mulss _b(%rip), %xmm0
  movss %xmm0, _1temp_0(%rip)
# TAC_RET
  movss _1temp_0(%rip), %xmm0
# TAC_FEND
  popq  %rbp
  retq
  .cfi_endproc
# DATA SECTION
  .section  __DATA,__data

_0: .long 0
_1: .long 1
_6.0: .long 1086324736
_2.0: .long 1073741824
_1temp_0: .long 0
L_.str.0: .asciz  "%d\n"
_a: .long 0
_b: .long 0
```

Resultado da execução:

```
$ ./a.out
8
```

Caso 4:

- Vetores.
- Print de caracteres.
- Print de string.

Código **ere2020:**

```
caracteres = char[2]: 'a' 'b';

main () = int {
    print caracteres[0]
    print caracteres[1]
    print "Caracteres acima estão sendo exibidos\n"
};
```

Código **assembly:**

```
## FIXED INIT
  .section  __TEXT,__cstring,cstring_literals
  printintstr: .asciz "%d\n"
  printfloatstr: .asciz "%f\n"
  printstringstr: .asciz  "%s\n"

  .section  __TEXT,__text,regular,pure_instructions
.globl _main
# TAC_AATTR
  leaq  _caracteres(%rip), %rcx
  movl  _97(%rip), %edx
  movslq  _0(%rip), %rsi
  movl  %edx, (%rcx,%rsi,4)
# TAC_AATTR
  leaq  _caracteres(%rip), %rcx
  movl  _98(%rip), %edx
  movslq  _1(%rip), %rsi
  movl  %edx, (%rcx,%rsi,4)
# TAC_FBEGIN
_main:
  .cfi_startproc
  pushq %rbp
  movq  %rsp, %rbp
# TAC_ACALL
  leaq  _caracteres(%rip), %rcx
  movslq  _0(%rip), %rdx
  movl  (%rcx,%rdx,4), %esi
  movl  %esi, _1temp_0(%rip)
# TAC_PRINT
  movl  _1temp_0(%rip), %esi
  leaq  L_.str.0(%rip), %rdi
  callq _printf
# TAC_ACALL
  leaq  _caracteres(%rip), %rcx
  movslq  _1(%rip), %rdx
  movl  (%rcx,%rdx,4), %esi
  movl  %esi, _1temp_1(%rip)
# TAC_PRINT
```

```
  movl  _1temp_1(%rip), %esi
  leaq  L_.str.1(%rip), %rdi
  callq _printf
# TAC_PRINT
  leaq  _string_7(%rip), %rsi
  leaq  L_.str.2(%rip), %rdi
  callq _printf
# TAC_FEND
  popq  %rbp
  retq
  .cfi_endproc
# DATA SECTION
  .section  __DATA,__data

_2: .long 2
_0: .long 0
_1: .long 1
_97:  .long 97
_98:  .long 98
_string_7: .asciz "Caracteres acima estão sendo exibidos\n"
_1temp_0: .long 0
_1temp_1: .long 0
_caracteres:
  .long 97
  .long 98
L_.str.0: .asciz  "%d\n"
L_.str.1: .asciz  "%d\n"
L_.str.2: .asciz  "%s\n"
```

Resultado da execução:

```
$ ./a.out
97
98
Caracteres acima estão sendo exibidos
```

Observações

- Caracteres são printados de acordo com seu código ASCII.
- Foi definido um **id** para cada elemento da hash. É esse id que é utilizado para definir o número da string no `#data_section`. Por exemplo, acima temos `_string_7`, o que significa que a string é o sétimo elemento armazenado na hash.