UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO

Disciplina: INF0334 – Software Básico

Wendel Marques de Jesus Souza (201702793)*

RELATÓRIO DO TRABALHO FINAL

1 – Objetivo do programa

Criação de um programa em C capaz de não só ler um arquivo contendo um programa na linguagem simples como também imprimir a tradução desse programa em Assembly na tela.

2 – Funcionamento do tradutor

A função ***tradutor*** é responsável por, efetivamente, realizar a tradução para Assembly. Inicialmente, todo o arquivo que contém o programa em linguagem simples é transcrito para um vetor (*linhas[]*) e, posteriormente, a função ***estabelece_espaco_pilha_para_params*** calcula o espaço necessário para salvar os parâmetros. O bloco seguinte (identificação: //while_var**) calcula o espaço necessário para armazenar as variáveis; essa etapa é finalizada com o alinhamento a 16. O bloco seguinte (//for_enderecos) calcula o endereço de cada variável e o armazena no vetor *pilha_enderecos[]*.

Exemplo do funcionamento do bloco *for_enderecos* – Seja o seguinte programa:

```
function pi1
def
var vi1
var vi2
vet va3 size ci5
var vi4
enddef
return ci1
end
```

*Único autor.
** Comentários no código-fonte para facilitar a busca do bloco.

- Tamanho da pilha após o alinhamento: 48;
- O valor do tamanho da pilha, após o alinhamento, sempre é o valor do endereço da primeira variável inteira – ou do array de inteiros;
- O endereço de vi1 é -48(%rbp). Ele é armazenado na posição 1 do vetor pilha_enderecos;
- Subtrai-se 4 de tamanho_pilha. Assim, tamanho_pilha = 44;
- O endereço de vi2 é, então, -44(%rbp). Ele é armazenado na posição 2 do vetor pilha_enderecos;
- Subtrai-se 4 de tamanho_pilha. Assim, tamanho_pilha = 40;
- Seguindo a mesma lógica, o endereço de va3 é -40(%rbp), o qual corresponde a va3[0]. Ele é armazenado em pilha_enderecos[3].
- Subtrai-se (4 * tam_vetor) de tamanho_pilha. Assim, nesse exemplo, tamanho_pilha = tamanho_pilha - 4 * 5 = 20;
- Por fim, pela mesmo lógica: a posição pilha_enderecos[4] armazena o valor -20, que corresponde ao endereço de vi4;
- Ou seja, o endereço da variável **x** é armazenado em pilha_enderecos[**x**].

O *while* após o *for* (id.: //while_traducao) vare o vetor linhas[] até que encontre “end”. Durante a varredura, uma linha é analisada por vez. Em cada análise, o código em Assembly correspondente é, por fim, impresso na tela.

Existe um contador de funções que permite numerar cada uma delas. Sendo assim, a primeira função do arquivo será impressa como *function1*, a segunda como *function2* e assim por diante. Essa manipulação facilita a identificação não do bloco da função em si, como também das chamadas de função. De modo semelhante, os condicionais “*if*” também são numerados.

Resumo do funcionamento de cada função:

- **descobre_reg_param:** Verifica pi<num>. O retorno depende do valor de <num>; retorna 1 para %edi; ou 2 para %esi; ou 3 para %edx;
- **estabelece_espaco_pilha_para_params:** Verifica quantos parâmetros cada função possui e então retorna a quantidade necessária para que seja possível salvá-los na pilha, caso necessário. Se possuir 1 parâmetro, tamanho = 8; se 2, 16; se 3, 24;
- **if_cases:** Cada retorno corresponde a uma possível traducao presente na função tradutor;
- **chamada_funcao:** Traduz chamadas de função;
- **conta_qtd_linhas_arq:** Conta a quantidade de linhas que o arquivo “prog.slp” possui.

4 – Salvamento de registradores

Os registradores usados são: %r8, %r9 e %r10. Tendo em vista que, nesse programa, eles desempenham papeis de variáveis “auxiliares”, não há a necessidade de salvá-los antes de chamar uma função, uma vez que armazenam valores que podem ser recuperados facilmente. Porém, o registradores %rdi, %rsi e %rdx sempre são salvos da seguinte forma:

%rdi → -8(%rbp)
%rsi → -16(%rbp)
%rdx → -24(%rbp)

5 – Testes realizados

O programa foi capaz de traduzir corretamente todos os programas testados. Aparentemente, traduz de forma correta qualquer linha que o programa em linguagem simples possa possuir. Vale ressaltar que o segundo

“exemplo 1” presente nas diretrizes do trabalho possui um erro na antepenúltima linha da última function. Está “endfi” no lugar de “endif”.

Ambiente de testes: gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)