Especificação da Etapa 6 do Projeto de Compilador

Suporte à Execução

A sexta etapa do trabalho de implementação de um compilador para a **Linguagem Amarela** consiste na geração de código intermediário para o suporte à execução. A parte principal consiste no código para gerenciar o fluxo de controle através da pilha de ativação e os seus registros de ativação. Utilizaremos como representação intermediária a **Linguagem ILOC**, descrita em detalhes no apêndice A de *Engineering a Compiler* [**?**], mas com o essencial descrito na definição da quinta etapa do projeto de compiladores. Os registros de ativação deverão conter pelo menos as seguintes informações (em ordem arbitrária):

- Temporários (caso sejam necessários)
- Variáveis locais
- Estado da Máquina
- Vínculo estático
- Vínculo dinâmico
- Valor retornado
- Parâmetros formais (argumentos)
- Endereço de retorno

1 Funcionalidades Necessárias

Os resultados compreendem *a correção dos problemas encontrados na etapa anterior* e as funcionalidades seguintes:

1.1 Registros de Ativação

Deve ser definida uma ordem dos dados dentro de um registro de ativação. Esta ordem deve ser obedecida no momento de criar o registro de ativação de cada função da Linguagem **Amarela**. Cada função deve ter o seu padrão de registro de ativação, de acordo com seus parâmetros formais e variáveis locais.

1.2 Gerenciamento da Pilha

O código a ser embutido no programa para o suporte ao ambiente de execução deve ser capaz de gerenciar o fluxo de controle do programa através da pilha de ativação. Assuma que existam registradores específicos para tal: como o *stack pointer* (sp) e o *frame pointer* (fp). Os endereços de vínculo estático e dinâmico de cada registro de ativação devem ser corretamente calculados.

1.3 Sequência de Ativação

O programa deve ter código específico para a chamada e o retorno de cada subprograma da Linguagem **Amarela**. O código de chamada deve ser capaz de criar corretamente, na pilha, uma instância do registro de ativação de um subprograma apontando o fluxo de execução para o código da função chamada no segmento de código. De maneira equivalente, o código de retorno deve ser capaz de atualizar a pilha e retornar o fluxo de execução para a função chamadora, no ponto imediatamente após a instrução de chamada do subprograma. Cada subprograma deve ter um rótulo específico indicando a sua primeira instrução.

1.4 Passagem de Parâmetros

A linguagem **Amarela** possui apenas passagem de parâmetro por valor com semântica de entrada, que pode ser implementado realizando uma cópia do parâmetro real para o parâmetro formal. Lembre-se que a Linguagem **Amarela** não aceita passagem de arranjos como parâmetros, somente variáveis de tipo simples.

Entrada e Saída Padrão

Organize a sua solução para que o compilador leia o programa em **Amarela** da entrada padrão e gere o programa em ILOC na saída padrão. Dessa forma, pode-se realizar o seguinte comando (considerando que `main` é o binário do compilador):

```
./main < entrada.iks > saida.iloc
```

Onde `entrada.iks` contém um programa em **Amarela**, e `saida.iloc` contém o programa em ILOC correspondente.

2 Notas importantes

É interessante enfatizar que a execução do programa começa sempre pela função cujo nome é `main`, que pode eventualmente não ser a primeira função do código de entrada na linguagem **Amarela**. Outro ponto bastante relevante é que tanto `fp`, `sp` e `rbss` devem ser obrigatoriamente inicializados antes do início da execução da função principal do programa. Portanto as primeiras instruções no código de saída devem ser algo como, em ILOC:

```
loadI 0 => fp
loadI 0 => sp
loadI 0 => rbss
```

Onde os endereços fp e sp são normalmente 0, mas o sp deve obrigatoriamente ser depois definido com o tamanho do registro de ativação da função principal. Quanto ao rbss, pode-se assumir que o segmento de dados de variáveis globais começa em 0 também, mas em um outro espaço de memória diferente da memória usada pela pilha.

3 Requisitos Obrigatórios

Os requisitos obrigatórios são os mesmos da seção equivalente na especificação das etapas anteriores do projeto de compilador. O ambiente de compilação e execução desta etapa podem ser definidos da seguinte forma:

```
$ git pull origin master
$ cmake -DETAPA_1=OFF -DETAPA_2=OFF -DETAPA_3=OFF -DETAPA_4=OFF -DETAPA_5=OFF -DETAPA_6=ON ..
$ make
```

Não existem testes de avaliação automática. O grupo pode utilizar o simulador `ilocsim` para verificar o bom funcionamento do código gerado de suporte à execução. Um programa simples que pode ser utilizado para testes desta etapa é o seguinte:

```
int mult (int z, int w)
{
  int x;
  x = z * w;
  return x;
}

int main()
{
  int x;
  int y;
  x = 2;
  y = mult(x, x);
}
```

4 Dúvidas

Crie um novo tópico no fórum de discussão do projeto de compilador no moodle.

Bom trabalho!