Trabalho de Geração de Código BIP

Disciplina de Compiladores
2016/I

Trabalho de Geração de Código BIP

Os slides seguintes apresentam as instruções para o trabalho de geração de código que será realizado em 3 etapas:

Etapa 1 (24/05 a 07/06/16)	Etapa 2 (07/06 a 21/06/16)	Etapa 3 (21/06 a 28/06/16)
1.1 Declarações (1 ponto)	2.1 Desvio simples (2,5 pontos)	3.1 Sub-rotinas (3 pontos)
1.2 Entrada de dados (2)	2.2 Desvio composto (2,5)	3.2 Passagem de parâm. (4)
1.3 Saída de dados (1)	2.3 Laço *while* (2,5)	3.3 Retorno de funções (3)
1.4 Atribuições (2)	2.4 Laço *for* (2,5)	
1.5 Operações aritméticas (3)		
(+1 ponto pela interface)		

Serão utilizados os códigos de exemplo para a validação de cada item. Caso a gramática não seja Portugol, os códigos deverão ser adaptados para a sintaxe da linguagem escolhida.

1.1 Declarações
(1 ponto)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro vet1[5]
  inteiro vet2[2]
  inteiro a
  inteiro b
Inicio
  leia(a)
  escreva(b)
  escreva(88)
fim
	No BIP vamos trabalhar somente com inteiros!
	
	Para cada inteiro declarado, deve ser gerada uma linha assembly na seção .data.
	
	A seção .data pode ser gerada com a varredura da tabela de símbolos.
	
	O BIP possui somente um contexto de declarações. Assim, variáveis de mesmo id podem ser tratadas adicionando-se o nome do contexto como prefixo, por ex.:
	
	principal_a : 0
	
	se1_a: 0
		.data
		  vet1 : 0,0,0,0,0
		  vet2 : 0,0
		  a : 0
		  b : 0
		.text
		_PRINCIPAL:
		  LD $in_port
		  STO a
		  LD b
		  STO $out_port
		  LDI 88
		  STO $out_port
		  HLT 0

1.2 Entrada de dados
(2 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro vet1[5]
  inteiro vet2[2]
  inteiro a
  inteiro b
Inicio
  leia(a)
  escreva(b)
  escreva(88)
fim
	Sempre que o comando de entrada for invocado, deverão ser geradas duas instruções na seção .text do assembly:
	
	A instrução LD $in_port espera que o usuário forneça um valor como entrada. O valor fornecido é carregado (LD) no acumulador (ACC).
	
	A instrução STO a armazena o valor do acumulador (ACC) no endereço de memória identificado pela variável a.
		.data
		  vet1 : 0,0,0,0,0
		  vet2 : 0,0
		  a : 0
		  b : 0
		.text
		_PRINCIPAL:
		  LD $in_port
		  STO a
		  LD b
		  STO $out_port
		  LDI 88
		  STO $out_port
		  HLT 0

1.3 Saída de dados
(1 ponto)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro vet1[5]
  inteiro vet2[2]
  inteiro a
  inteiro b
Inicio
  leia(a)
  escreva(b)
  escreva(88)
fim
	Sempre que o comando de saída for invocado, deverão ser geradas duas instruções na seção .text do assembly.
	
	A instrução LD b carrega o valor do endereço de memória identificado pela variável b no acumulador (ACC).
	
	A instrução LDI 88 carrega o valor 88 no acumulador (ACC).
	
	A instrução STO $out_port fornece o valor registrado no acumulador (ACC) na saída do BIP.
		.data
		  vet1 : 0,0,0,0,0
		  vet2 : 0,0
		  a : 0
		  b : 0
		.text
		_PRINCIPAL:
		  LD $in_port
		  STO a
		  LD b
		  STO $out_port
		  LDI 88
		  STO $out_port
		  HLT 0

1.4 Atribuições simples
(1 de 2 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro vet1[5]
  inteiro vet2[2]
  inteiro a
  inteiro b
inicio
  a <- 5
  a <- b
fim
	Uma ação semântica no ID que recebe o valor deve guardar o lexema (a) em memória.
	
	A próxima ação semântica deverá gerar a instrução que carrega o valor de atribuição no acumulador. A instrução será LDI no caso de uma constante e LD no caso de uma variável (ID).
	
	Uma ação semântica no final da linha deverá gerar a instrução STO, para que valor do acumulador seja armazenado no endereço de memória identificado pelo lexema (a), que foi guardado no primeiro passo.
		.data
		  vet1 : 0,0,0,0,0
		  vet2 : 0,0
		  a : 0
		  b : 0
		.text
		_PRINCIPAL:
		  LDI 5
		  STO a
		  LD b
		  STO a
		  HLT 0

1.4 Atribuições com vetores
(1 de 2 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro vet1[5]
  inteiro vet2[2]
  inteiro a
  inteiro b
inicio
  a <- vet1[0]
  vet2[0] <- a
fim
	Na atribuição com vetores é necessário utilizar o registrador $indr para armazenar o valor do índice do vetor.
	
	O comando STO $indr armazena o valor do acumulador (ACC) no registrador $indr.
	
	O comando LDV vet1 carrega o valor da variável vet1[$indr] no acumulador.
	
	O comando STOV vet2 armazena o valor do acumulador na variável vet2[$indr].
		.data
		  vet1 : 0,0,0,0,0
		  vet2 : 0,0
		  a : 0
		  b : 0
		.text
		_PRINCIPAL:
		  LDI 0
		  STO $indr
		  LDV vet1
		  STO a
		  LDI 0
		  STO 1000
		  LD a
		  STO 1001
		  LD 1000
		  STO $indr
		  LD 1001
		  STOV vet2
		  HLT 0

1.5 Operações aritméticas simples
(1 de 3 pontos)

Portugol

```
procedimento principal()
declaracoes
   inteiro vet1[5]
   inteiro vet2[2]
   inteiro a
   inteiro b
inicio
   a <- 5 + 5
   a <- 5 - 5
   a <- a + 5
   a <- a + b
fim
```

Conversão (GALS)

Uma ação semântica no operador deverá guardar o valor do lexema (+ ou -) em memória e setar uma flag como TRUE, para que as ações semânticas do ID e da constante saibam em qual lado do operador eles estão.

Se lado esquerdo:

- Gerar LD se o operando for uma variável.
- Gerar LDI se o operando for uma constante.

Se lado direito:

- Gerar ADD se a operação for soma e o operando variável.
- Gerar ADDI se a operação for soma e o operando constante.
- Gerar SUB se a operação for subtração e o operando variável.
- Gerar SUBI se a operação for subtração e o operando constante.

Assembly BIP (.asm)

```
.data
   vet1 : 0,0,0,0,0
   vet2 : 0,0
   a : 0
   b : 0
.text
_PRINCIPAL:
   LDI    5
   ADDI   5
   STO    a
   LDI    5
   SUBI   5
   STO    a
   LD     a
   ADDI   5
   STO    a
   LD     a
   ADD    b
   STO    a
   HLT    0
```

1.5 Operações aritméticas com vetores
(1 de 3 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro vet1[5]
  inteiro vet2[2]
  inteiro a
  inteiro b
inicio
  vet1[0] <- a + 5
fim
	Vetor recebendo valor
	
	Neste caso, além de guardar o lexema (vet1) em memória, deverá também ser gerada uma instrução para guardar o valor do índice em temp1 (1000).
	
	Ao final, antes de ser gerada a instrução STOV, o valor do acumulador (ACC) deverá ser armazenado em temp2 (1001), para que o valor do índice possa ser transferido de temp1 (1000) para $indr.
		.data
		  vet1 : 0,0,0,0,0
		  vet2 : 0,0
		  a : 0
		  b : 0
		.text
		_PRINCIPAL:
		  LDI 0
		  STO 1000
		  LD a
		  ADDI 5
		  STO 1001
		  LD 1000
		  STO $indr
		  LD 1001
		  STOV vet1
		  HLT 0

1.5 Operações aritméticas com vetores
(1 de 3 pontos)

Portugol

```
procedimento principal()
declaracoes
   inteiro vet1[5]
   inteiro vet2[2]
   inteiro a
   inteiro b
inicio
   a <- vet1[0] + a
   a <- 5 + vet2[0]
fim
```

Conversão (GALS)

Vetor como operando

1. Se lado esquerdo:

Basta setar o valor de $indr antes de gerar a instrução LDV.

2. Se lado direito:

O valor do acumulador deverá ser armazenado em temp1 (1000) para que o valor do índice possa ser carregado no acumulador e transferido para $indr.

Em seguida o valor do vetor dever ser carregado no acumulador e armazenado em temp2 (1001).

A operação é realizada com os valores armazenados em temp1 e temp2.

Assembly BIP (.asm)

```
.data
   vet1 : 0,0,0,0,0
   vet2 : 0,0
   a : 0
   b : 0
.text
_PRINCIPAL:
   LDI     0
   STO     $indr
   LDV     vet1
   ADD     a
   STO     a
   LDI     5
   STO     1000
   LDI     0
   STO     $indr
   LDV     vet2
   STO     1001
   LD      1000
   ADD     1001
   STO     a
   HLT     0
```

2.1 Desvio condicional simples
(2,5 pontos)

Portugol

```
procedimento principal()
declaracoes
   inteiro a
inicio
   se (a = 2) entao
      a <- 30
   fimse
fim
```

Conversão (GALS)

1. Gerar o código da expressão. Será necessário utilizar temp1 (1000) e temp2 (1001) para realizar a comparação.

2. A ação semântica de "entao" deverá gerar uma instrução de desvio para "FIM" + <ID do escopo>. O tipo de instrução pode ser gravado em memória quando o uso de um operador relacional é sinalizado:

```
BEQ  se  !=
BNE  se  =
BGT  se  <=
BGE  se  <
BLT  se  >=
BLE  se  >
```

4. O código de "entao" será gerado pelas ações já definidas no slide 1.4.

5. A ação semântica de "fimse" deverá gerar a instrução "FIM" + <ID do escopo> + ":".

Assembly BIP (.asm)

```
.data
   a : 0
.text
_PRINCIPAL:
   LD      a
   STO     1000
   LDI     2
   STO     1001
   LD      1000
   SUB     1001
   BNE     FIMSE1
   LDI     30
   STO     a
FIMSE1:
   HLT     0
```

2.2 Desvio condicional composto
(2,5 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro a
inicio
  se (a = 2) entao
  a <- 30
  senao
  a <- 0
  fimse
fim
	1. O código da expressão será gerado pelas ações já definidas no slide 2.1.
	
	2. A ação semântica de "entao" deverá gerar uma instrução de desvio para "EL" + <ID do escopo>. Usar a mesma estratégia do slide 2.1 para definir o tipo de instrução.
	
	3. O código de "entao" será gerado pelas ações já definidas no slide 1.4.
	
	4. A ação semântica de "senao" deverá gerar as instruções "JMP FIM" + <ID do escopo> e "EL" + <ID do escopo> + ":";
	
	5. O código de "senao" será gerado pelas ações já definidas no slide 1.4.
	
	6. A ação semântica de "fimse" deverá gerar a instrução "FIM" + <ID do escopo> + ":".
		.data
		  a : 0
		.text
		_PRINCIPAL:
		  LD a
		  STO 1000
		  LDI 2
		  STO 1001
		  LD 1000
		  SUB 1001
		  BNE ELSE1
		  LDI 30
		  STO a
		  JMP FIMSE1
		ELSE1:
		  LDI 0
		  STO a
		FIMSE1:
		  HLT 0

2.3 Laço *while*
(2,5 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro a
inicio
  enquanto (a < 5) faca
  a <- a + 1
  fimenquanto
fim
	1. A ação semântica de "enquanto" deve gerar uma nova instrução "INI_" + <ID do escopo> + ":".
	
	2. O código da expressão será gerado pelas ações já definidas no slide 2.1.
	
	3. A ação semântica de "faca" deverá gerar uma instrução de desvio para "FIM" + <ID do escopo>. Usar a mesma estratégia do slide 2.1 para definir o tipo de instrução.
	
	4. O código de "faca" será gerado pelas ações já definidas no slide 1.4.
	
	5. A ação semântica de "fimenquanto" deverá gerar as instruções "JMP INI_" + <ID do escopo> e "FIM" + <ID do escopo> + ":"
		.data
		  a : 0
		.text
		_PRINCIPAL:
		INI_ENQ1:
		  LD a
		  STO 1000
		  LDI 5
		  STO 1001
		  LD 1000
		  SUB 1001
		  BGE FIMFACA1
		  LD a
		  ADDI 1
		  STO a
		  JMP INI_ENQ1
		FIMFACA1:
		  HLT 0

2.4 Laço *for*
(2,5 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

procedimento principal()
declaracoes
  inteiro a
inicio
  para a <- 1 ate 50 passo 1
  escreva(a)
  fimpara
fim
	1. Setar o valor inicial da variável (a).
	
	2. Guardar o valor final (50) em temp1 (1000) e o valor do passo (1) em temp2 (1001).
	
	3. Carregar o valor inicial (a) no acumulador (ACC).
	
	EM PARAx:
	
	4. Realizar o teste e, se for o caso, ir para FIMPARAx:.
	
	5. As instruções do laço serão geradas pelas ações já definidas anteriormente.
	
	6. Gerar FIMPARAx:.
		.data
		  a : 0
		.text
		_PRINCIPAL:
		  LDI 1
		  STO a
		  LDI 50
		  STO 1000
		  LDI 1
		  STO 1001
		  LD a
		PARA1:
		  SUB 1000
		  BGT FIMPARA1
		  LD a
		  STO $out_port
		  LD a
		  ADD 1001
		  STO a
		  JMP PARA1
		FIMPARA1:
		  HLT 0

3.1 Sub-rotinas
(3 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

inteiro dobro(inteiro valor)
declaracoes
inicio
  retornar valor + valor
fim
procedimento principal()
declaracoes
  inteiro a
inicio
  escreva(dobro(a))
fim
	A ação semântica que sinaliza a declaração de uma sub-rotina deverá gerar uma nova seção assembly com o nome da sub-rotina (_DOBRO:).
	
	A ação semântica que sinaliza o uso de uma sub-rotina deverá gerar a instrução CALL, tendo como operando o nome da sub-rotina (_DOBRO).
		.data
		  dobro_valor : 0
		  a : 0
		.text
		  JMP _PRINCIPAL
		_DOBRO:
		  LD DOBRO_valor
		  ADD DOBRO_valor
		  RETURN 0
		_PRINCIPAL:
		  LD a
		  STO DOBRO_valor
		  CALL _DOBRO
		  STO $out_port
		  HLT 0

3.2 Passagem de parâmetros
(4 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

inteiro dobro(inteiro valor)
declaracoes
inicio
  retornar valor + valor
fim
procedimento principal()
declaracoes
  inteiro a
inicio
  escreva(dobro(a))
fim
	Se a lógica do item 1.1 foi desenvolvida corretamente, todos os IDs utilizados como parâmetros devem constar na tabela de símbolos com os atributos parâmetro=TRUE e posição=X.
	
	Sempre que uma passagem de parâmetro for sinalizada, deve-se carregar o valor no acumulador (ACC) e transferi-lo para o endereço de memória identificado pelo nome da variável, que pode ser obtido na tabela de símbolos (consultar escopo e posição).
		.data
		  dobro_valor : 0
		  a : 0
		.text
		  JMP _PRINCIPAL
		_DOBRO:
		  LD DOBRO_valor
		  ADD DOBRO_valor
		  RETURN 0
		_PRINCIPAL:
		  LD a
		  STO DOBRO_valor
		  CALL _DOBRO
		  STO $out_port
		  HLT 0

3.3 Retorno de funções
(3 pontos)

Portugol	Conversão (GALS)	Assembly BIP (.asm)

inteiro dobro(inteiro valor)
declaracoes
inicio
  retornar valor + valor
fim
procedimento principal()
declaracoes
  inteiro a
inicio
  escreva(dobro(a))
fim
	No BIP, as sub-rotinas retornam valores através do acumulador (ACC).	.data
		  dobro_valor : 0
		  a : 0
		.text
		  JMP _PRINCIPAL
		_DOBRO:
		  LD DOBRO_valor
		  ADD DOBRO_valor
		  RETURN 0
		_PRINCIPAL:
		  LD a
		  STO DOBRO_valor
		  CALL _DOBRO
		  STO $out_port
		  HLT 0