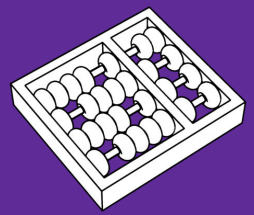
Seleção de Instruções

Geração de Código

Hervé Yviquel
herve@ic.unicamp.br

Universidade Estadual de Campinas (Unicamp)
Instituto de Computação (IC)
Laboratório de Sistemas de Computação (LSC)

MC921 • Projeto e Construção de Compiladores • 2023 S2



UNICAMP

Aula Anterior

Resumo

- Procedimentos
- Ativações
- Registro de Ativação
- Registradores

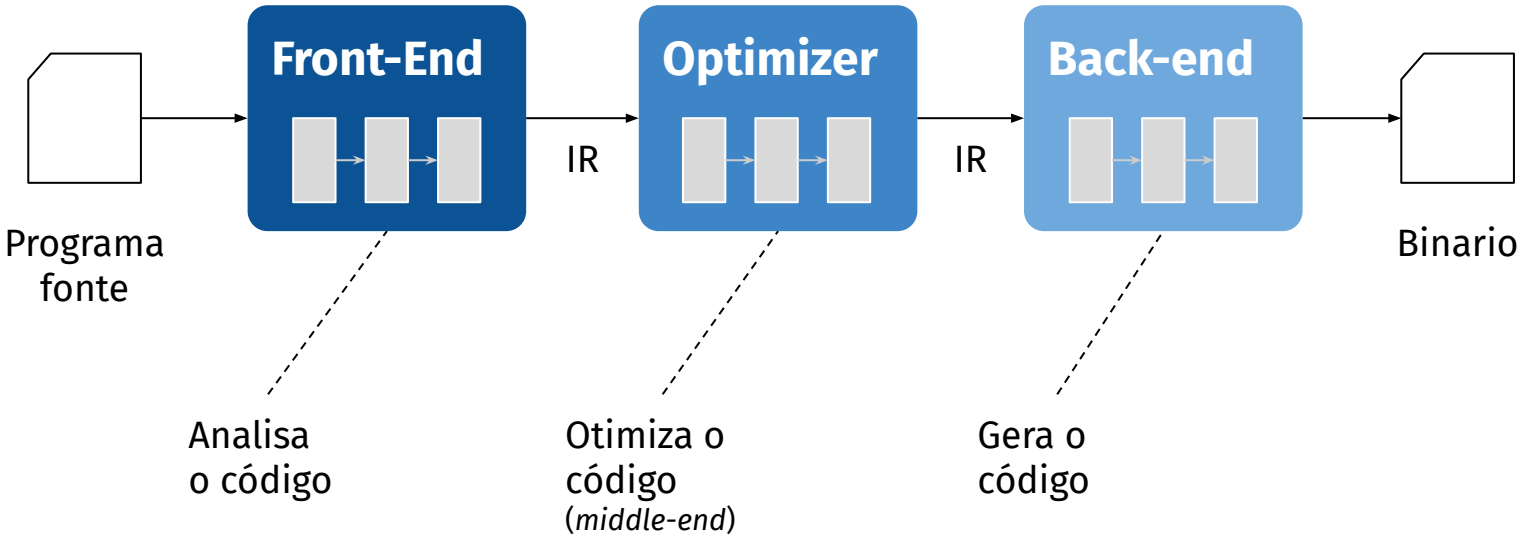
Aula de Hoje

Plano

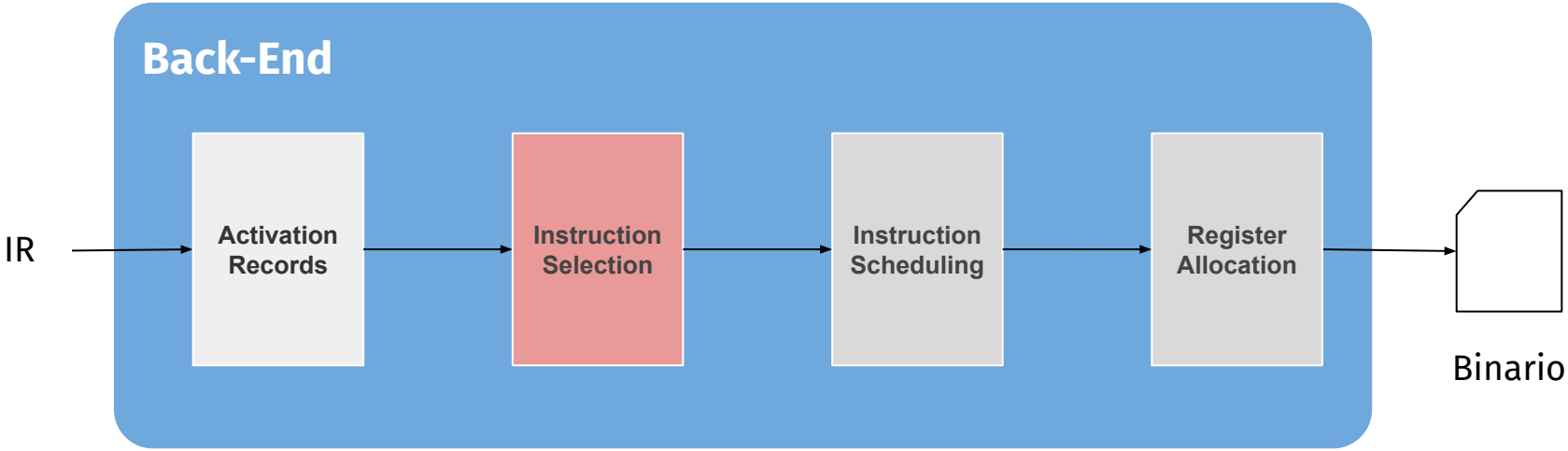
- Seleção de Instruções
- Arquitetura
- Primeiro exemplo
- Maximal Munch
- Custo da Cobertura
- Programação Dinâmica
- Geração de Código
- Eficiência

Seleção de Instruções

Estrutura Moderna do Compilador



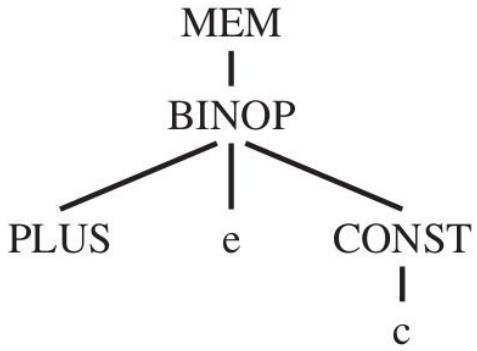
Back-End do Compilador



Representação Intermediária

- Vamos supor que a IR é representada com árvores
- Uma árvore da IR expressa uma operação “simples” em cada nó
  - Acesso à memória
  - Operador Binário
  - Salto condicional
- Instruções da máquina podem realizar uma ou mais dessas operações

Introdução



- Que instrução seria essa?
- Encontrar o conjunto de instruções de máquina que implementa uma dada árvore da IR é o objetivo da Seleção de Instruções

Padrões de Árvores

- Podemos expressar instruções da máquina como um fragmento da árvore da IR
  - Padrões de árvore
- Seleção de instruções
  - Cubra a árvore da IR com o “menor“ número de padrões existentes para a máquina alvo
- Exemplo:
  - Arquitetura Jouette

Arquitetura *Jouette*

Arquitetura *Jouette*

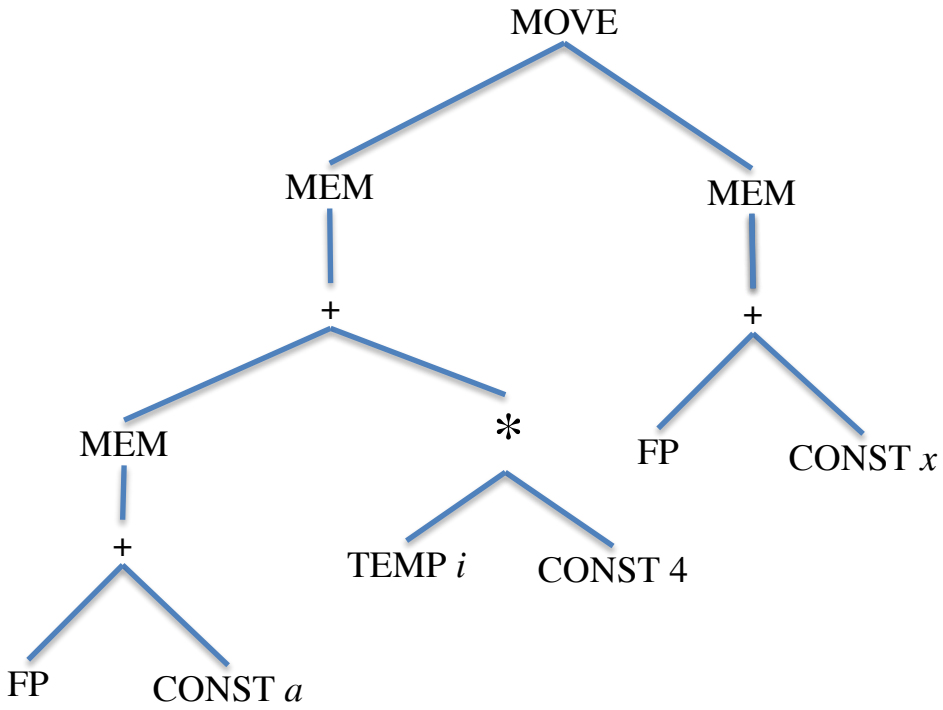
- Arquitetura Load/Store
- Qualquer instrução pode acessar qualquer registrador
- Registradores podem armazenar dado ou endereço
- r0 sempre contém zero
- Cada instrução gasta 1 ciclo
- Executa apenas uma instrução por ciclo

Padrões de Árvore - *Jouette*

Name	*Effect*	*Trees*
—	r_i	TEMP
ADD	$r_i \leftarrow r_j + r_k$	+
MUL	$r_i \leftarrow r_j \times r_k$	*
SUB	$r_i \leftarrow r_j - r_k$	-
DIV	$r_i \leftarrow r_j / r_k$	/
ADDI	$r_i \leftarrow r_j + c$	+ CONST; + CONST; CONST
SUBI	$r_i \leftarrow r_j - c$	- CONST
LOAD	$r_i \leftarrow M[r_j + c]$	MEM + CONST; MEM + CONST; MEM CONST; MEM
STORE	$M[r_j + c] \leftarrow r_i$	MOVE MEM + CONST CONST; MOVE MEM + CONST; MOVE MEM CONST; MOVE MEM
MOVEM	$M[r_j] \leftarrow M[r_i]$	MOVE MEM MEM

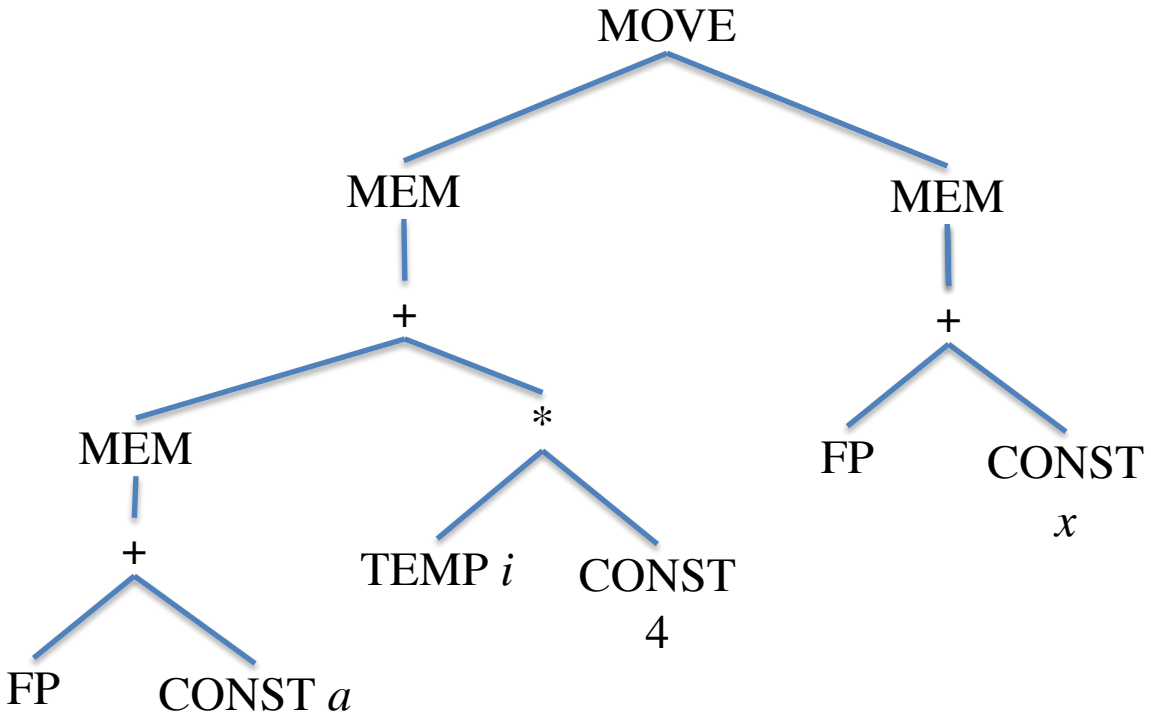
Primeiro Exemplo

Primeiro Exemplo

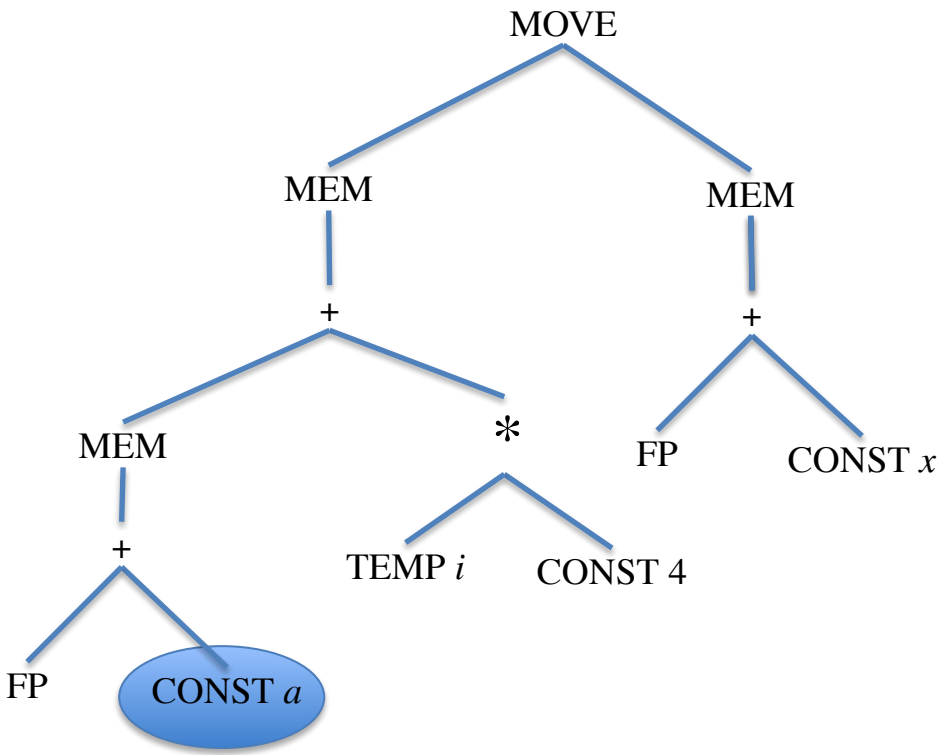


- Objetivo é cobrir a árvore com padrões
  - sem sobreposição entre padrões
- Exemplo
  - a[i] := x
  - Supondo que i está em um registrador e as variáveis a e x estão na pilha (indexada por FP)

Primeiro Exemplo

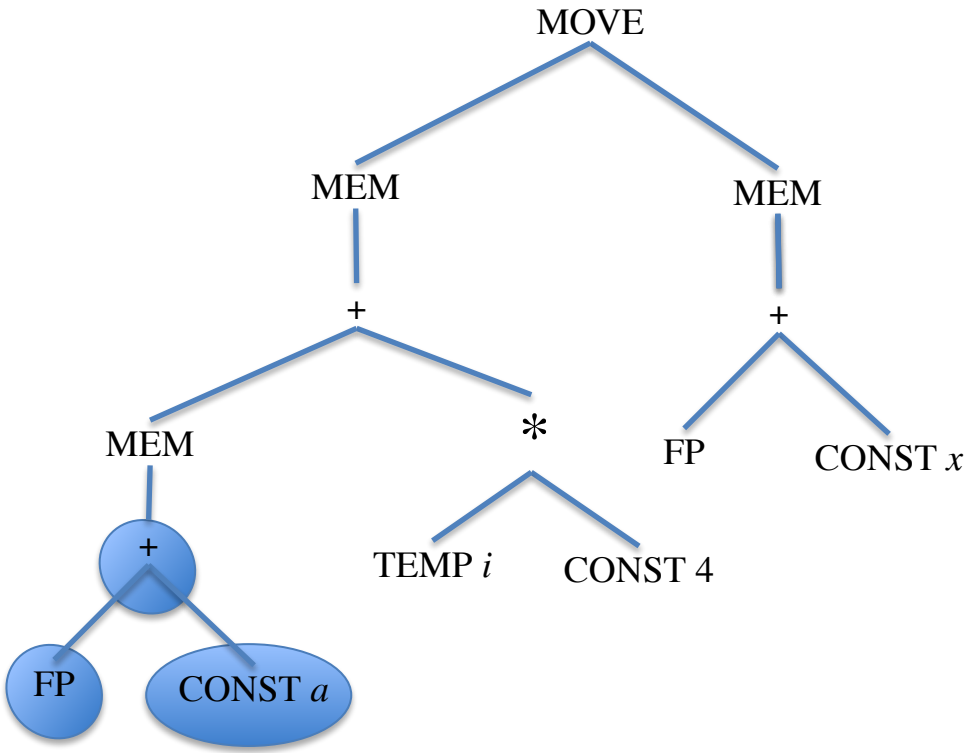


Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

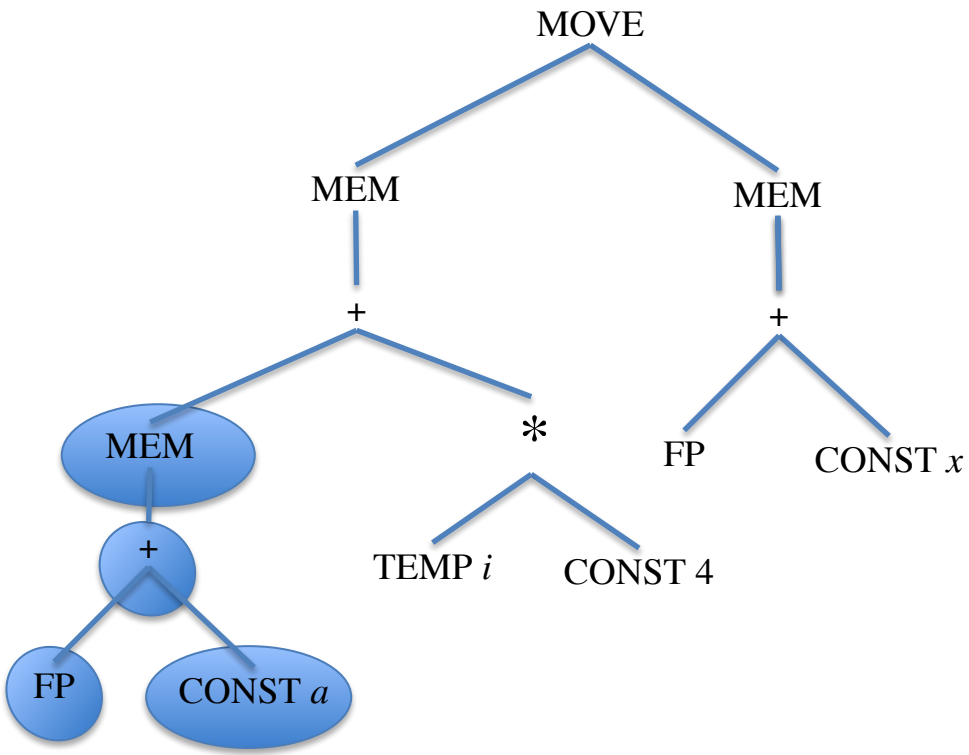
Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

Primeiro Exemplo

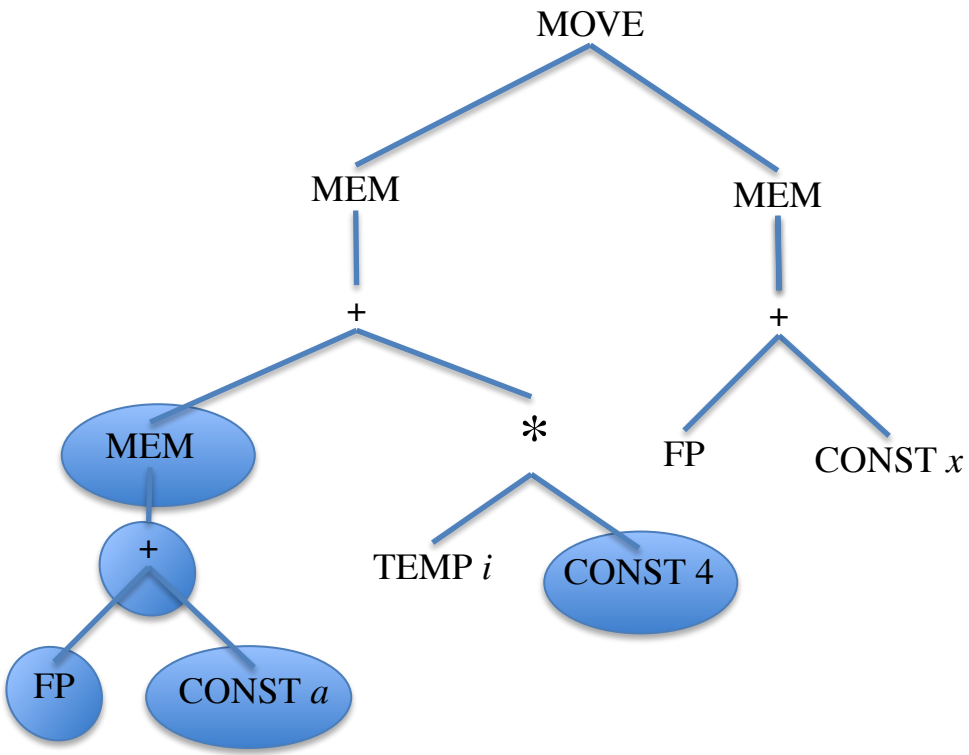


ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

LOAD $r1 \leftarrow M[r1 + 0]$

Primeiro Exemplo



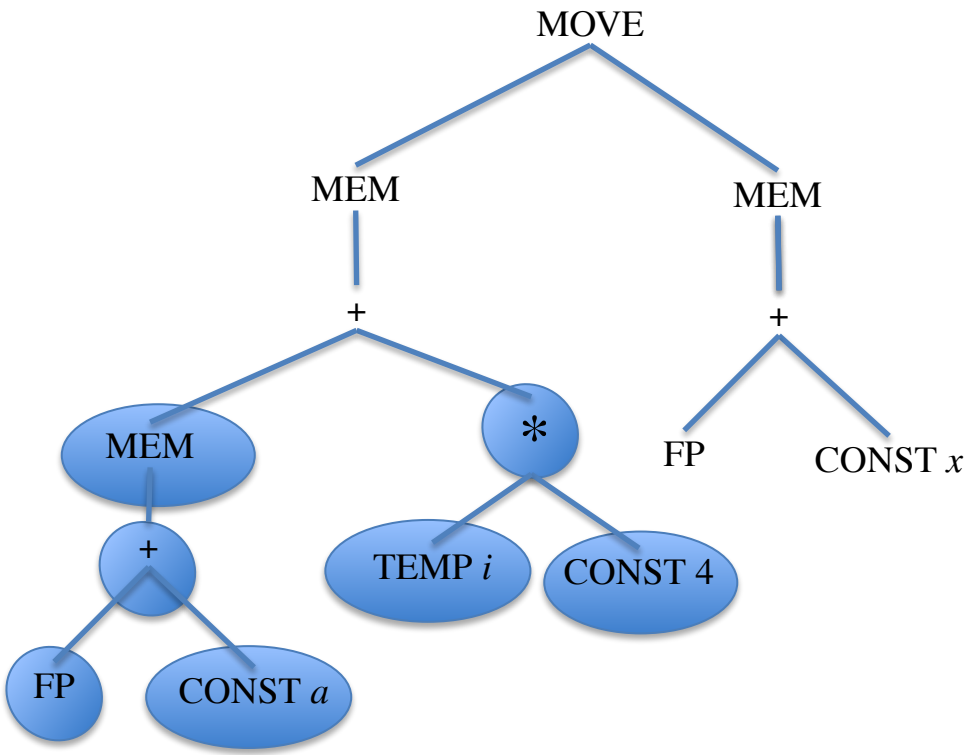
ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

LOAD $r1 \leftarrow M[r1 + 0]$

ADDI $r2 \leftarrow r0 + 4$

Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

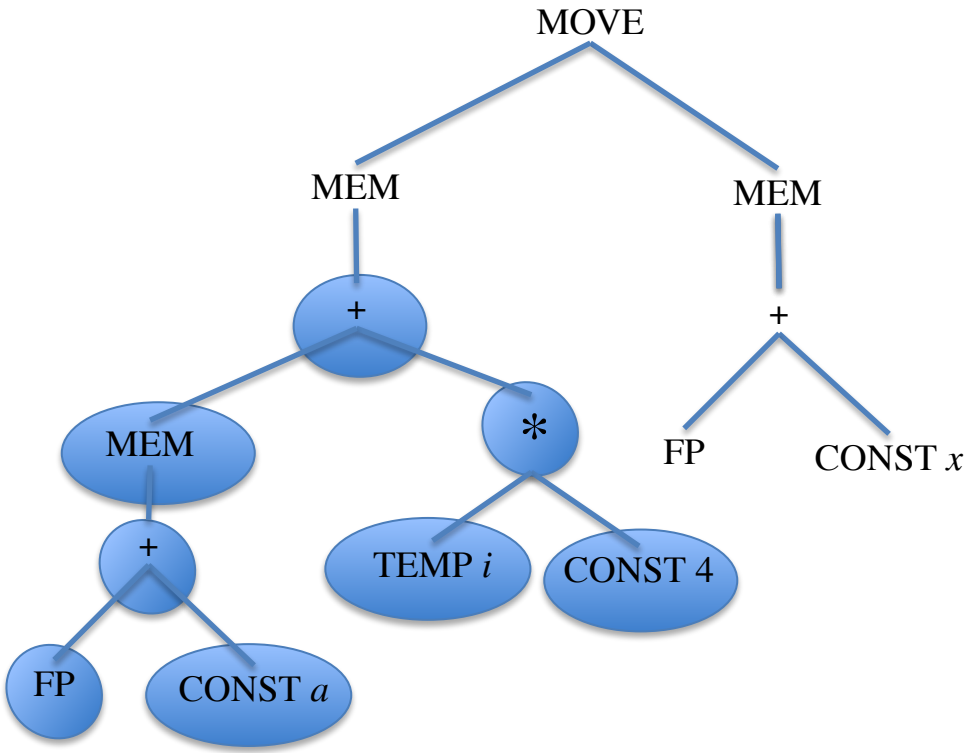
ADD $r1 \leftarrow \mathbf{fp} + r1$

LOAD $r1 \leftarrow M[r1 + 0]$

ADDI $r2 \leftarrow r0 + 4$

MUL $r2 \leftarrow ri \times r2$

Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

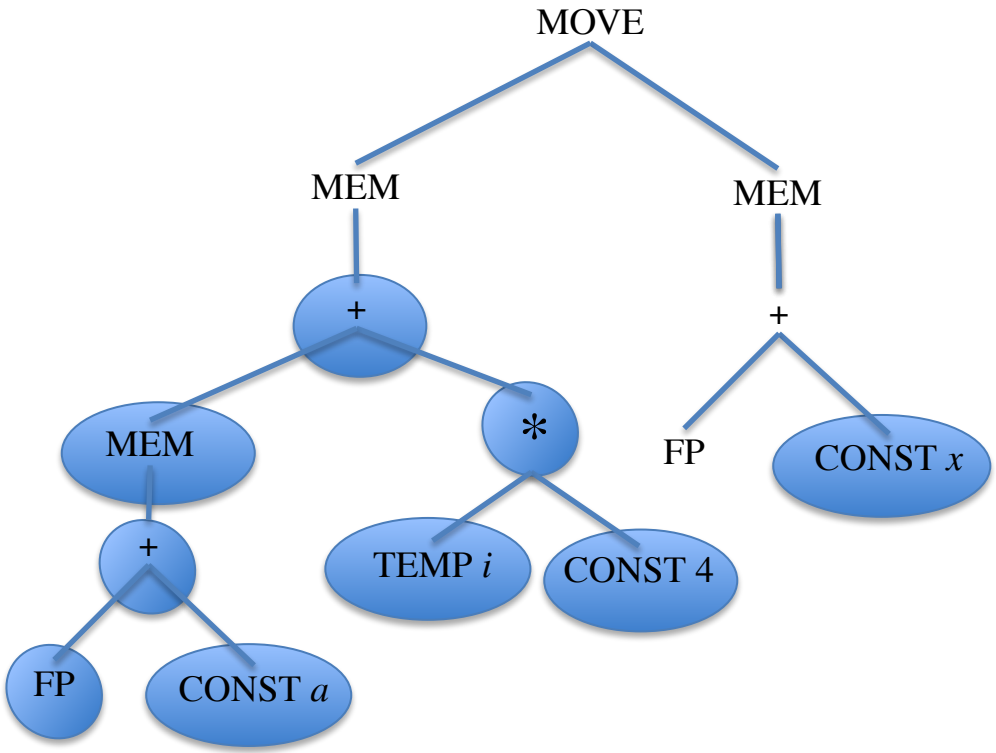
LOAD $r1 \leftarrow M[r1 + 0]$

ADDI $r2 \leftarrow r0 + 4$

MUL $r2 \leftarrow ri \times r2$

ADD $r1 \leftarrow r1 + r2$

Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

LOAD $r1 \leftarrow M[r1 + 0]$

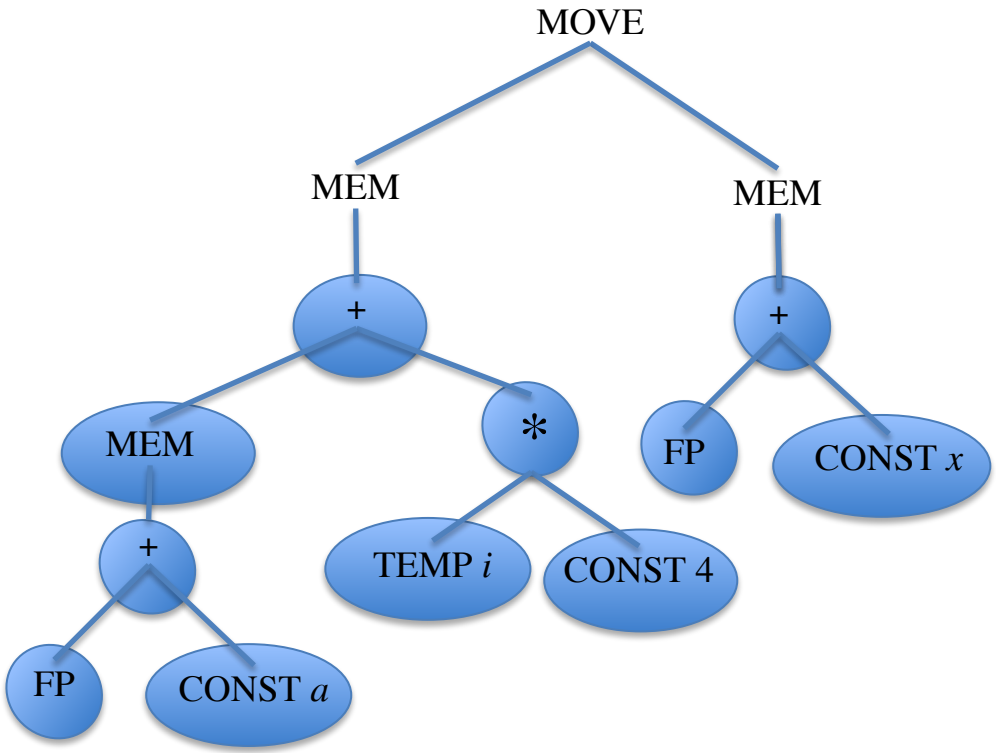
ADDI $r2 \leftarrow r0 + 4$

MUL $r2 \leftarrow ri \times r2$

ADD $r1 \leftarrow r1 + r2$

ADDI $r2 \leftarrow r0 + x$

Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

LOAD $r1 \leftarrow M[r1 + 0]$

ADDI $r2 \leftarrow r0 + 4$

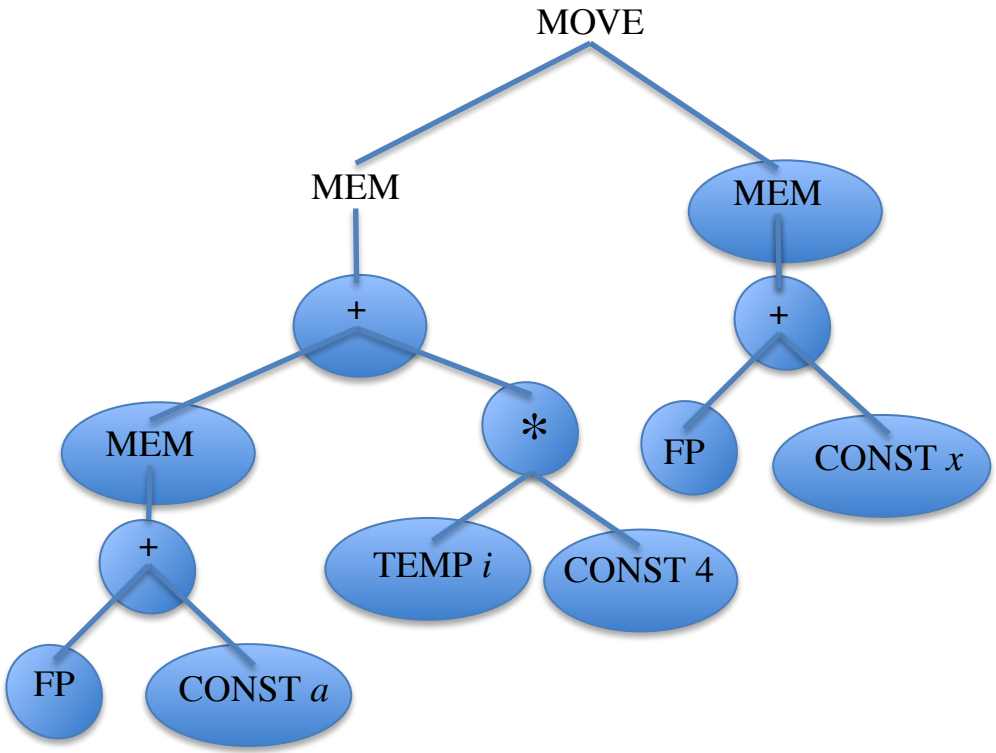
MUL $r2 \leftarrow ri \times r2$

ADD $r1 \leftarrow r1 + r2$

ADDI $r2 \leftarrow r0 + x$

ADD $r2 \leftarrow \mathbf{fp} + r2$

Primeiro Exemplo



ADDI $r1 \leftarrow r0 + a$

ADD $r1 \leftarrow \mathbf{fp} + r1$

LOAD $r1 \leftarrow M[r1 + 0]$

ADDI $r2 \leftarrow r0 + 4$

MUL $r2 \leftarrow ri \times r2$

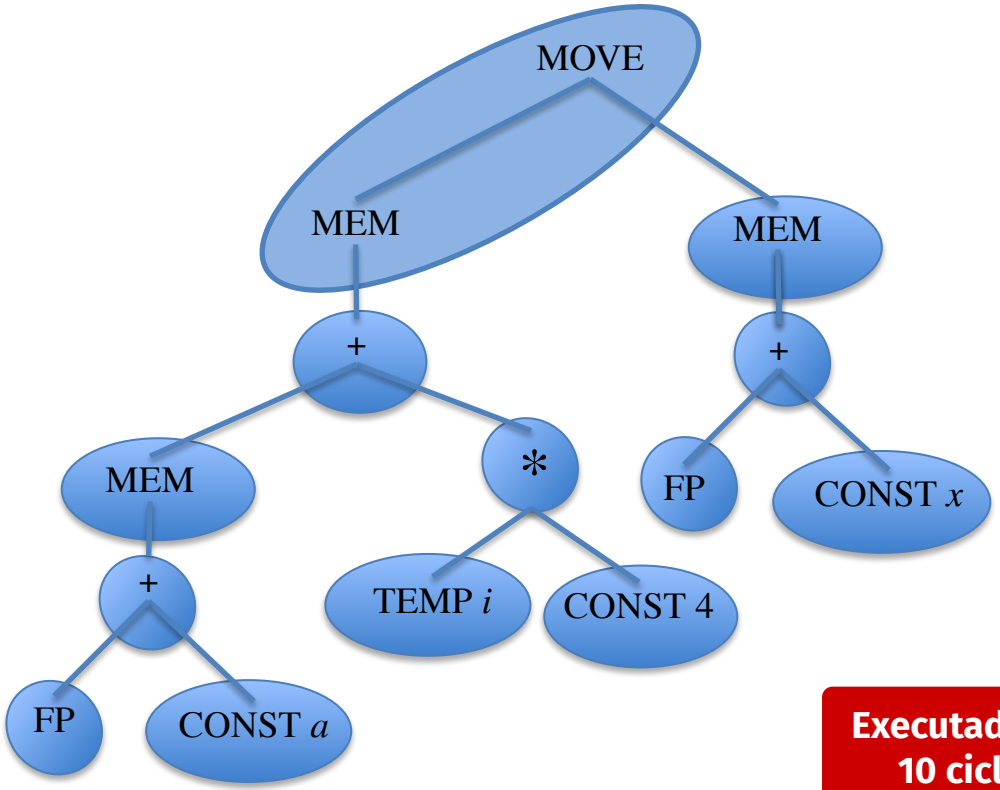
ADD $r1 \leftarrow r1 + r2$

ADDI $r2 \leftarrow r0 + x$

ADD $r2 \leftarrow \mathbf{fp} + r2$

LOAD $r2 \leftarrow M[r2 + 0]$

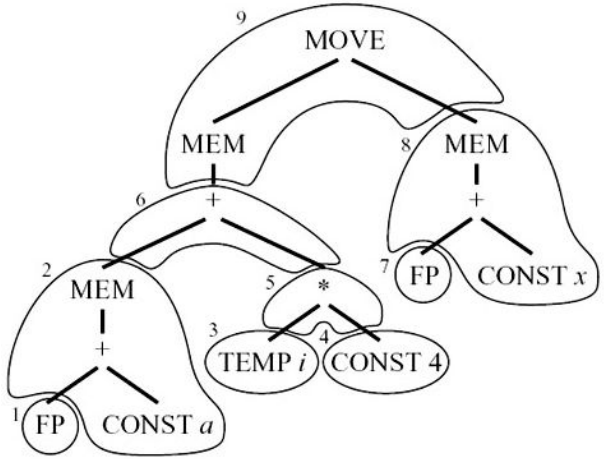
Primeiro Exemplo



Executado em 10 ciclos

ADDI	$r1 \leftarrow r0 + a$
ADD	$r1 \leftarrow \mathbf{fp} + r1$
LOAD	$r1 \leftarrow M[r1 + 0]$
ADDI	$r2 \leftarrow r0 + 4$
MUL	$r2 \leftarrow ri \times r2$
ADD	$r1 \leftarrow r1 + r2$
ADDI	$r2 \leftarrow r0 + x$
ADD	$r2 \leftarrow \mathbf{fp} + r2$
LOAD	$r2 \leftarrow M[r2 + 0]$
STORE	$M[r1 + 0] \leftarrow r2$

Seleção de Instruções



2	LOAD	$r_1 \leftarrow M[\mathbf{fp} + a]$
4	ADDI	$r_2 \leftarrow r_0 + 4$
5	MUL	$r_2 \leftarrow r_i \times r_2$
6	ADD	$r_1 \leftarrow r_1 + r_2$
8	LOAD	$r_2 \leftarrow M[\mathbf{fp} + x]$
9	STORE	$M[r_1 + 0] \leftarrow r_2$

- Uma possível solução
- A solução é única?

Executado em 6 ciclos

Maximal Munch

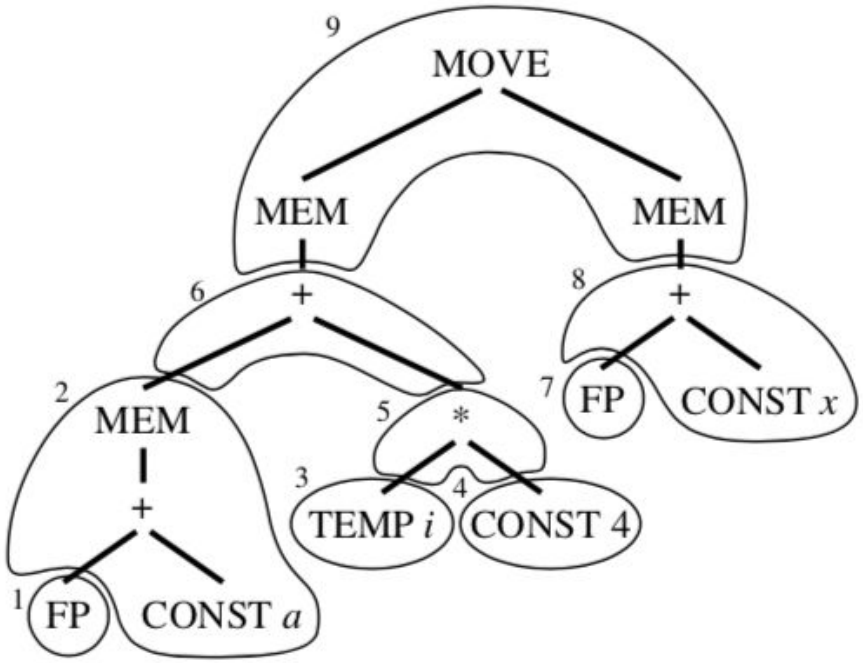
Maximal Munch

- Algoritmo guloso
  - Encontra cobertura *optimal*
- Bastante simples
  - Inicie no nó raiz
  - Encontre o maior padrão que possa ser encaixado naquele nó
    - Cubra o raiz e provavelmente outros nós
  - Repita o processo para os próximos nós da árvore IR a ser coberta
- A cada padrão selecionado, uma instrução é gerada
- Ordem inversa da execução!
  - A raiz é a última a ser executada

Maximal Munch

- Algumas observações
  - O maior padrão é aquele com maior número de nós
  - Se dois padrões do mesmo tamanho encaixam, a escolha é arbitrária
- Facilmente implementado através de funções recursivas
  - Ordene as cláusulas com a prioridade de tamanho dos padrões
  - Se para cada tipo de nó da árvore existir um padrão de cobertura de um nó, nunca pode ficar travado

Usando Maximal Munch



2	LOAD	$r_1 \leftarrow M[\mathbf{fp} + a]$
4	ADDI	$r_2 \leftarrow r_0 + 4$
5	MUL	$r_2 \leftarrow r_i \times r_2$
6	ADD	$r_1 \leftarrow r_1 + r_2$
8	ADDI	$r_2 \leftarrow \mathbf{fp} + x$
9	MOVEM	$M[r_1] \leftarrow M[r_2]$

Name	*Effect*	*Trees*
—	r_i	TEMP
ADD	$r_i \leftarrow r_j + r_k$	+
MUL	$r_i \leftarrow r_j \times r_k$	*
SUB	$r_i \leftarrow r_j - r_k$	-
DIV	$r_i \leftarrow r_j / r_k$	/
ADDI	$r_i \leftarrow r_j + c$	+ CONST; + CONST; CONST
SUBI	$r_i \leftarrow r_j - c$	- CONST
LOAD	$r_i \leftarrow M[r_j + c]$	MEM + CONST; MEM + CONST; MEM CONST; MEM
STORE	$M[r_j + c] \leftarrow r_i$	MOVE MEM + CONST CONST; MOVE MEM + CONST; MOVE MEM CONST; MOVE MEM
MOVEM	$M[r_j] \leftarrow M[r_i]$	MOVE MEM MEM

Exercícios

- Construa as árvores representando o seguinte código
  - MOVE(MEM(+(+(CONST 1000, MEM(TEMP x)), TEMP fp)), CONST 0)
  - *(CONST 5, MEM(CONST 100))
- Faça a cobertura Maximal Munch

Name	*Effect*	*Trees*
—	r_i	TEMP
ADD	$r_i \leftarrow r_j + r_k$	+
MUL	$r_i \leftarrow r_j \times r_k$	*
SUB	$r_i \leftarrow r_j - r_k$	-
DIV	$r_i \leftarrow r_j / r_k$	/
ADDI	$r_i \leftarrow r_j + c$	+ CONST; + CONST; CONST
SUBI	$r_i \leftarrow r_j - c$	- CONST
LOAD	$r_i \leftarrow M[r_j + c]$	MEM + CONST; MEM + CONST; MEM CONST; MEM
STORE	$M[r_j + c] \leftarrow r_i$	MOVE MEM + CONST CONST; MOVE MEM + CONST; MOVE MEM CONST; MOVE MEM
MOVEM	$M[r_j] \leftarrow M[r_i]$	MOVE MEM MEM

Custo da Cobertura

Custo da Cobertura

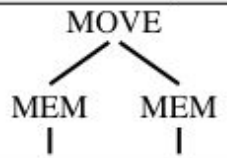
- Queremos a cobertura que nos traga o menor custo
  - Caso as instruções tenham latências diferentes
    - A cobertura de menor tempo total
- Cada instrução recebe um custo
  - A melhor cobertura da árvore é a que a soma dos custos dos padrões utilizados é a menor possível
  - Este é o **optimum**

Optimal e Optimum

- Uma cobertura onde nenhum par de padrões adjacentes possa ser combinado em um par de menor custo é optimal
- Caso haja um padrão que possa ser quebrado e diminua o custo total, ele deve ser descartado
  - não é optimum
- Optimum é sempre optimal
- Optimal nem sempre é optimum

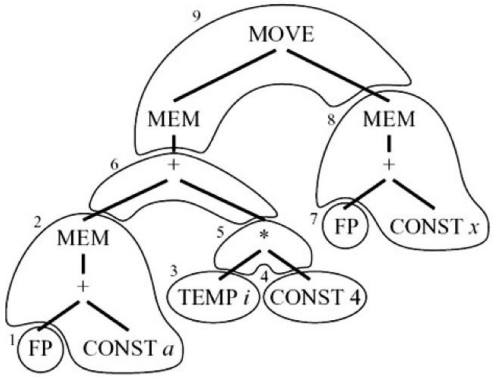
Instrução MOVE

- Na tabela de padrões assuma:
  - MOVEM tem custo m
  - Todas as outras têm custo 1
  - O que acontece com as duas coberturas se
    - m = 0, 1 ou 2?

MOVEM	$M[r_j] \leftarrow M[r_i]$	

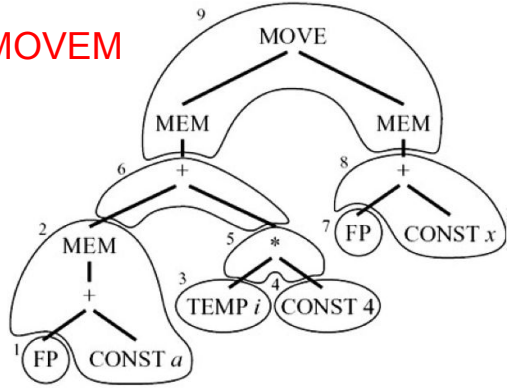
Variação do custo do MOVE

m	A	B
0	6	5
1	6	6
2	6	7



A

MOVEM



B

Maximal Munch não resulta em *optimum*!!

Algoritmos

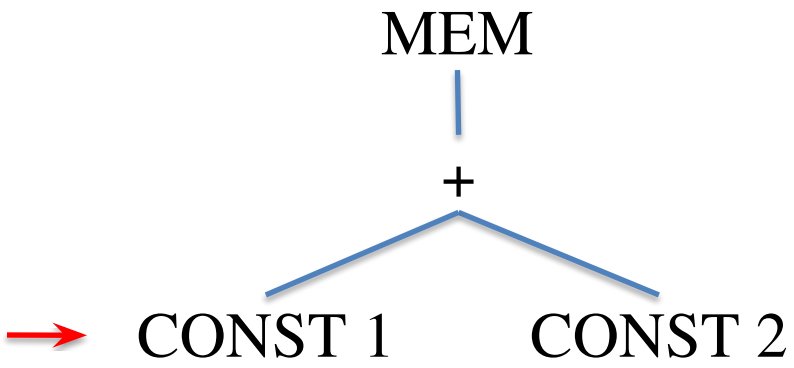
- Achar coberturas “optimais” é mais fácil
- RISC
  - Instruções simples levam a padrões pequenos
  - Custo costuma ser mais uniforme
  - A diferença entre optimal e optimum praticamente desaparece
- CISC
  - Dada a complexidade das instruções, os padrões costumam ser grandes
  - A diferença entre optimal e optimum se torna mais considerável

Programação Dinâmica

Programação Dinâmica

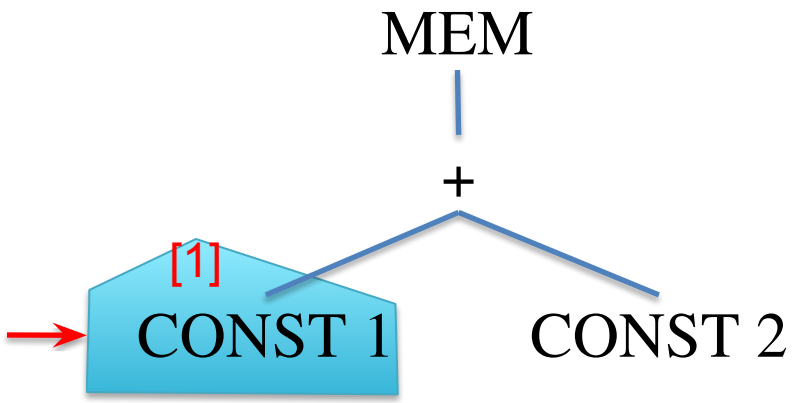
- Encontra um cobertura ótima (optimum)
- PD monta uma solução ótima baseada em soluções ótimas de sub-problemas
- O algoritmo atribui um vetor de custo a cada nó da árvore
  - A soma do custo de todas as instruções da melhor cobertura da sub-árvore com raiz no respectivo nó
  - Para um dado nó n
    - Encontra o melhor custo para suas sub-árvores
    - Analisa os padrões que podem cobrir n
    - Algoritmo bottom-up

Exemplo



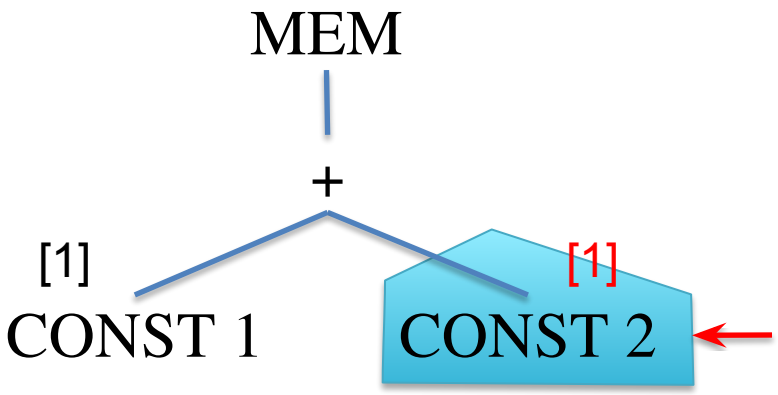
Tile	Instruction
CONST	ADDI

Exemplo



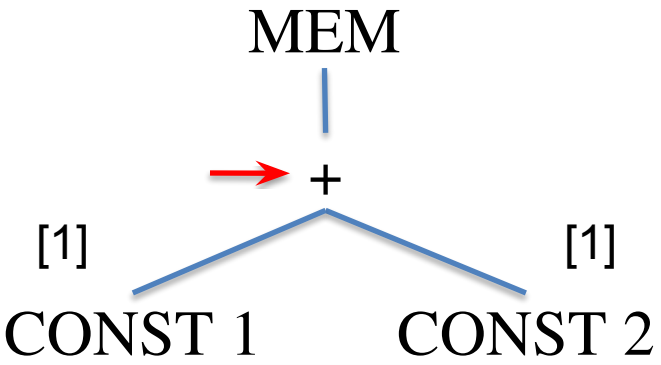
	Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
→	CONST	ADDI	1	0	1

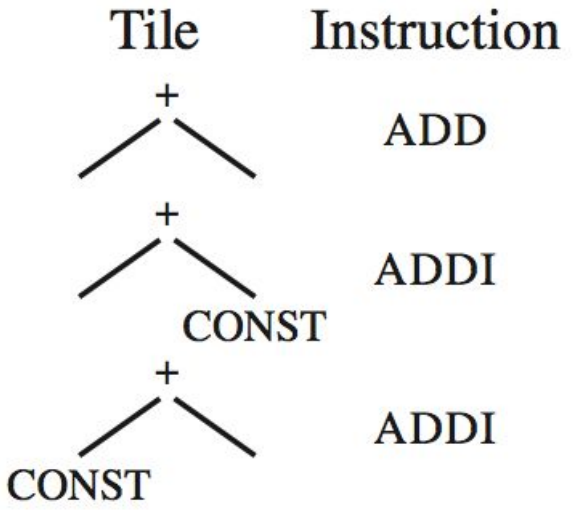
Exemplo



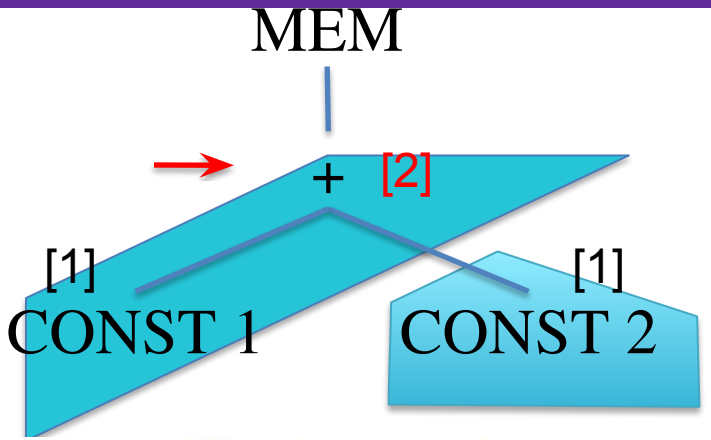
	Tile	Instruction	Tile Cost	Leaves Cost	Total Cost	
→	CONST	ADDI	1	0	1	SAME

Exemplo


MEM
+
[1]
[1]
CONST 1
CONST 2

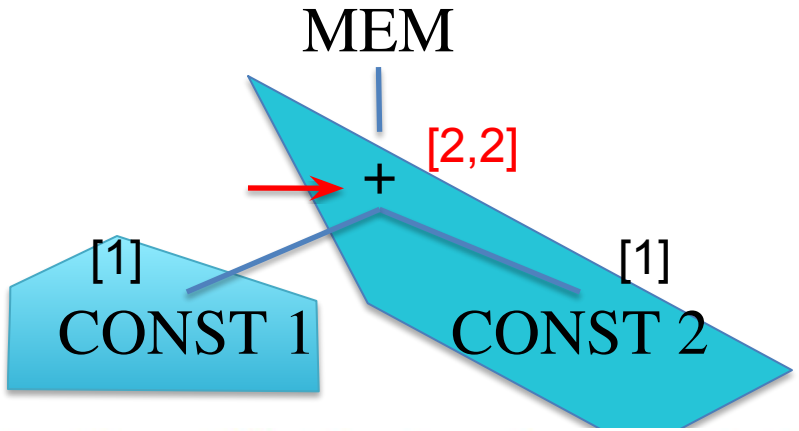

Tile
Instruction
+
ADD
+
ADDI
CONST
+
ADDI
CONST

Exemplo



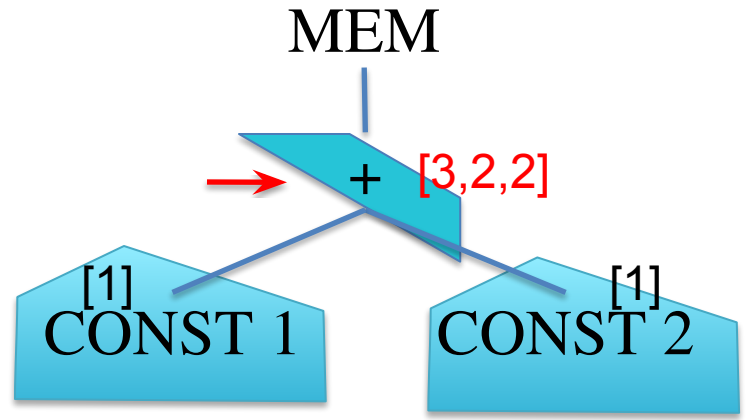
Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
+	ADD	1	1+1	3
+ CONST	ADDI	1	1	2
+ CONST	ADDI	1	1	2

Exemplo



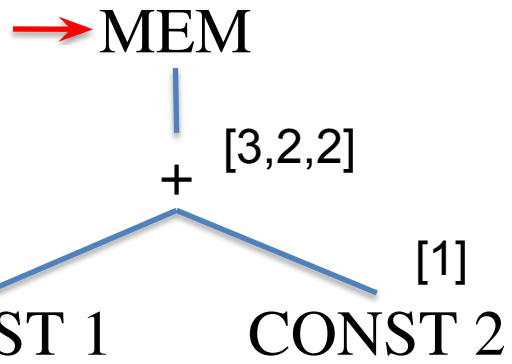
Tile	Instruction	Tile Cost	Leaves Cost	Total Cost	
+	ADD	1	1+1	3	
→ + CONST	ADDI	1	1	2	SAME
+ CONST	ADDI	1	1	2	

Exemplo



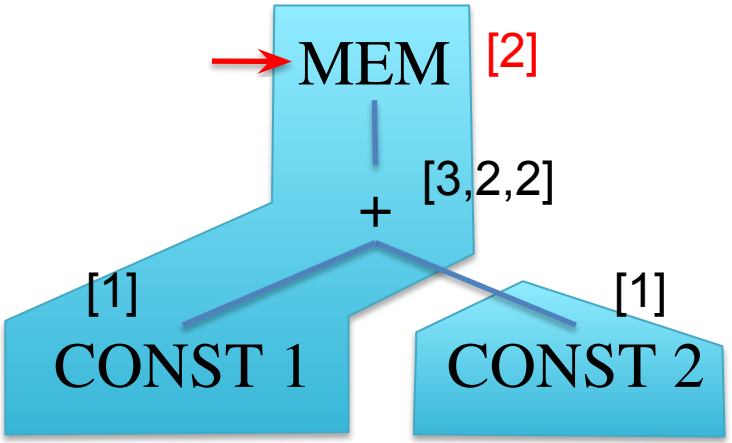
Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
+	ADD	1	1+1	3
+ CONST	ADDI	1	1	2
+ CONST	ADDI	1	1	2

Exemplo



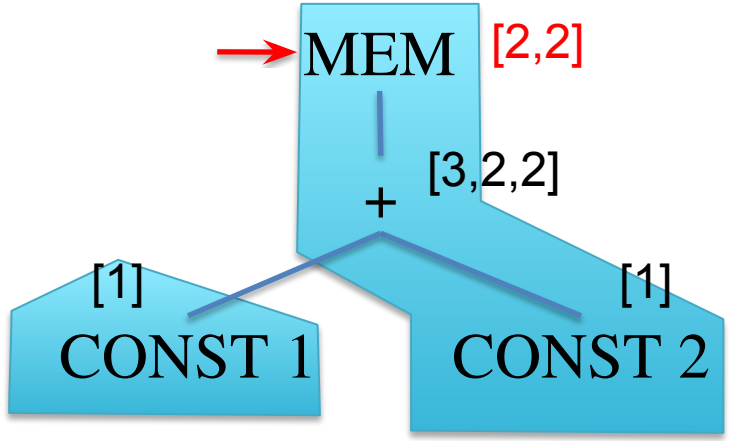
Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
MEM | MEM | + / \ CONST				
MEM | + / \ CONST				

Exemplo



Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
MEM	LOAD	1	2	3
MEM + CONST	LOAD	1	1	2
MEM + CONST	LOAD	1	1	2

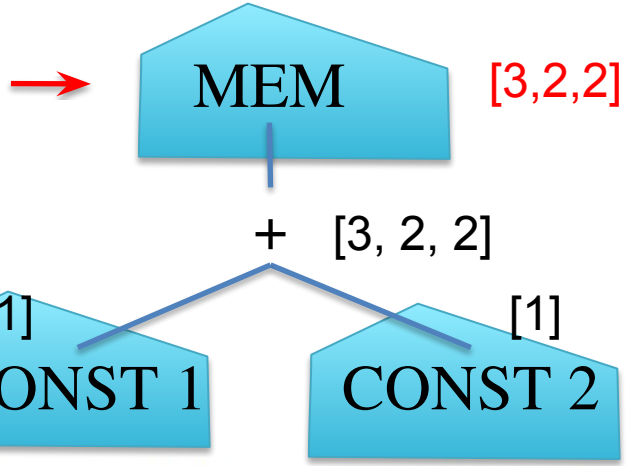
Exemplo



Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
MEM |	LOAD	1	2	3
MEM | + / \ CONST	LOAD	1	1	2
MEM | + / \ CONST	LOAD	1	1	2

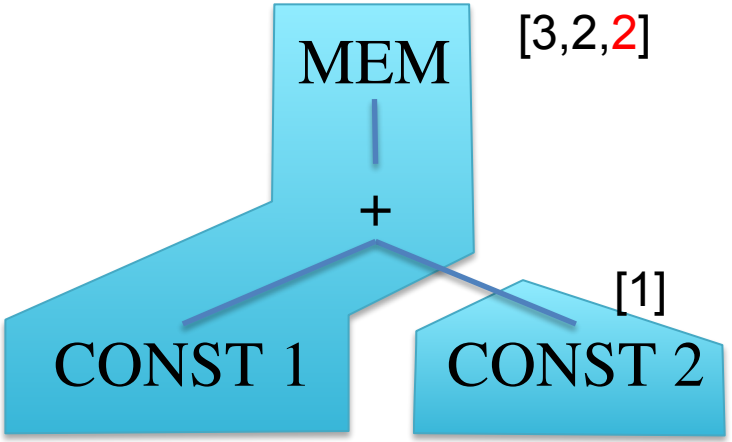
SAME

Exemplo



Tile	Instruction	Tile Cost	Leaves Cost	Total Cost
MEM	LOAD	1	2	3
MEM + CONST	LOAD	1	1	2
MEM + CONST	LOAD	1	1	2

Exemplo



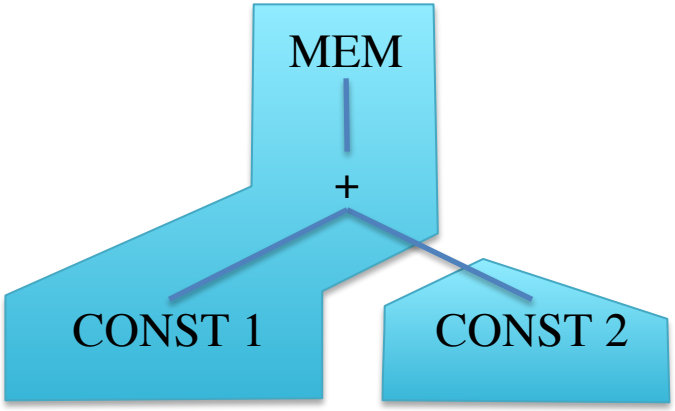
Melhor custo: 2

Geração de Código

Geração de Código

- Após computar o custo da raiz, emitir as instruções das folhas para a raiz
- Emissão(n)
  - Para cada folha f do padrão de árvore selecionado para o nó n, execute emissão(f) recursivamente.
  - Emita a instrução do padrão de n
- A emissão de código é feita através da chamada
  - Emissão(raiz)

Geração de Código



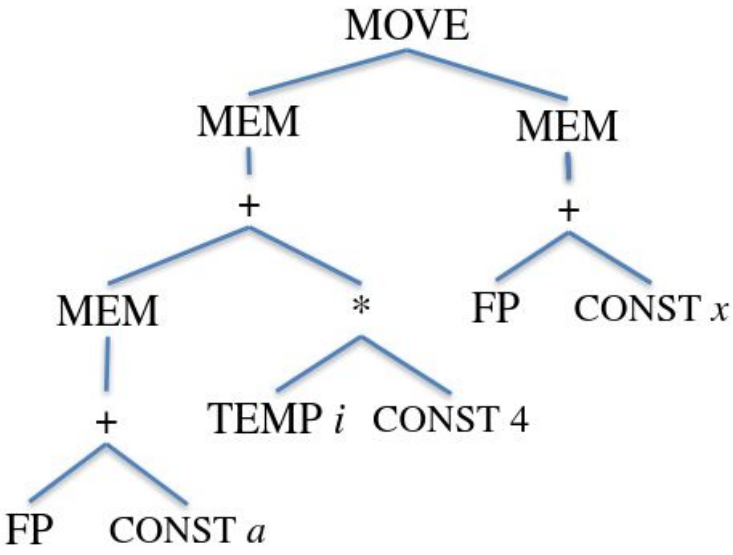

```
Emissão(MEM)
  Emissão(CONST 2)
    Emite: ADDI r1 := r0 + 2
  Emite: LOAD r1 := M[r1 + 1]
```

Ferramentas

- Muitas ferramentas foram desenvolvidas para gerar seletores de instruções automaticamente
  - TWIG [Aho, Ganapathi, Tjiang, ’86]
  - BURG [Fraser, Henry, Proebsting, ’92]
  - BEG [Emmelmann, Schroer, Landwehr, ’89]
  - ...
- Geram seletores de instruções de baixo para cima de especificações de regras de correspondência de árvores
  - Usam a abordagem de programação dinâmica
  - Muito trabalho foi feito para eles ficarem altamente eficientes

Exercício

Faça você mesmo considerando custo 1 para todos padrões



Name	*Effect*	*Trees*
—	r_i	TEMP 1
ADD	$r_i \leftarrow r_j + r_k$	+ 2
MUL	$r_i \leftarrow r_j \times r_k$	* 3
SUB	$r_i \leftarrow r_j - r_k$	- 4
DIV	$r_i \leftarrow r_j / r_k$	/ 5
ADDI	$r_i \leftarrow r_j + c$	+ (CONST) 6 + (CONST) 7 CONST 8
SUBI	$r_i \leftarrow r_j - c$	- (CONST) 9
LOAD	$r_i \leftarrow M[r_j + c]$	MEM + (CONST) 10 MEM + (CONST) 11 MEM CONST 12 MEM 13
STORE	$M[r_j + c] \leftarrow r_i$	MOVE MEM + (CONST CONST) 14 MOVE MEM + (CONST) 15 MOVE MEM CONST 16 MOVE MEM 17
MOVEM	$M[r_j] \leftarrow M[r_i]$	MOVE (MEM MEM) 18

Eficiência

Comentários sobre Eficiência

- Seja
  - T: número de padrões diferentes
  - K: no. médio de nós não-folhas dos padrões casados
  - K': maior # de nós a serem olhados para identificar quais padrões casam em uma dada sub-árvore. Aprox. o tamanho do maior padrão
  - T': média de padrões diferentes que casam em cada nó
  - N: # nós da árvore
- RISC típico:
  - T=50, K=2, K'=4, T'=5
- Maximal Munch: (K' + T') * N/K
- Programação Dinâmica: (K'+ T') * N
  - Requer duas passadas na árvore

Comentários sobre Eficiência

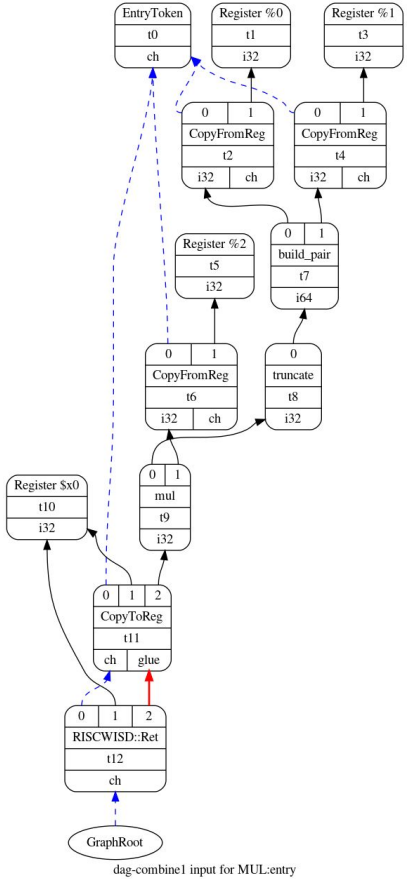
- Ambos são lineares
- Seleção de instruções é bastante rápida comparada com outras fases da compilação
- Até análise léxica pode ser mais demorada

Instruction Selection in LLVM

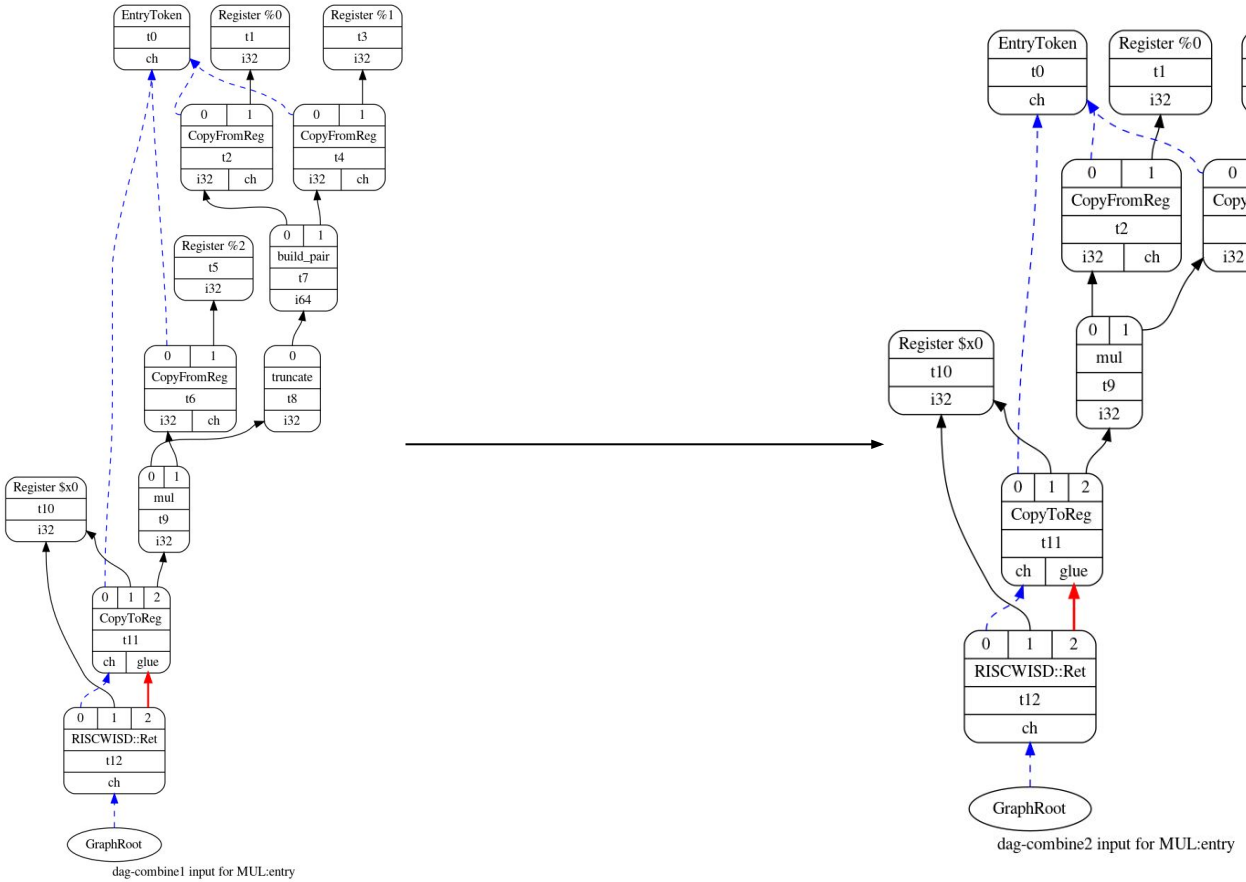
Etapa de Construção do Selection DAG


```
unsigned int MUL(unsigned long long int x, unsigned int y)
{
    return x * y;
}
```

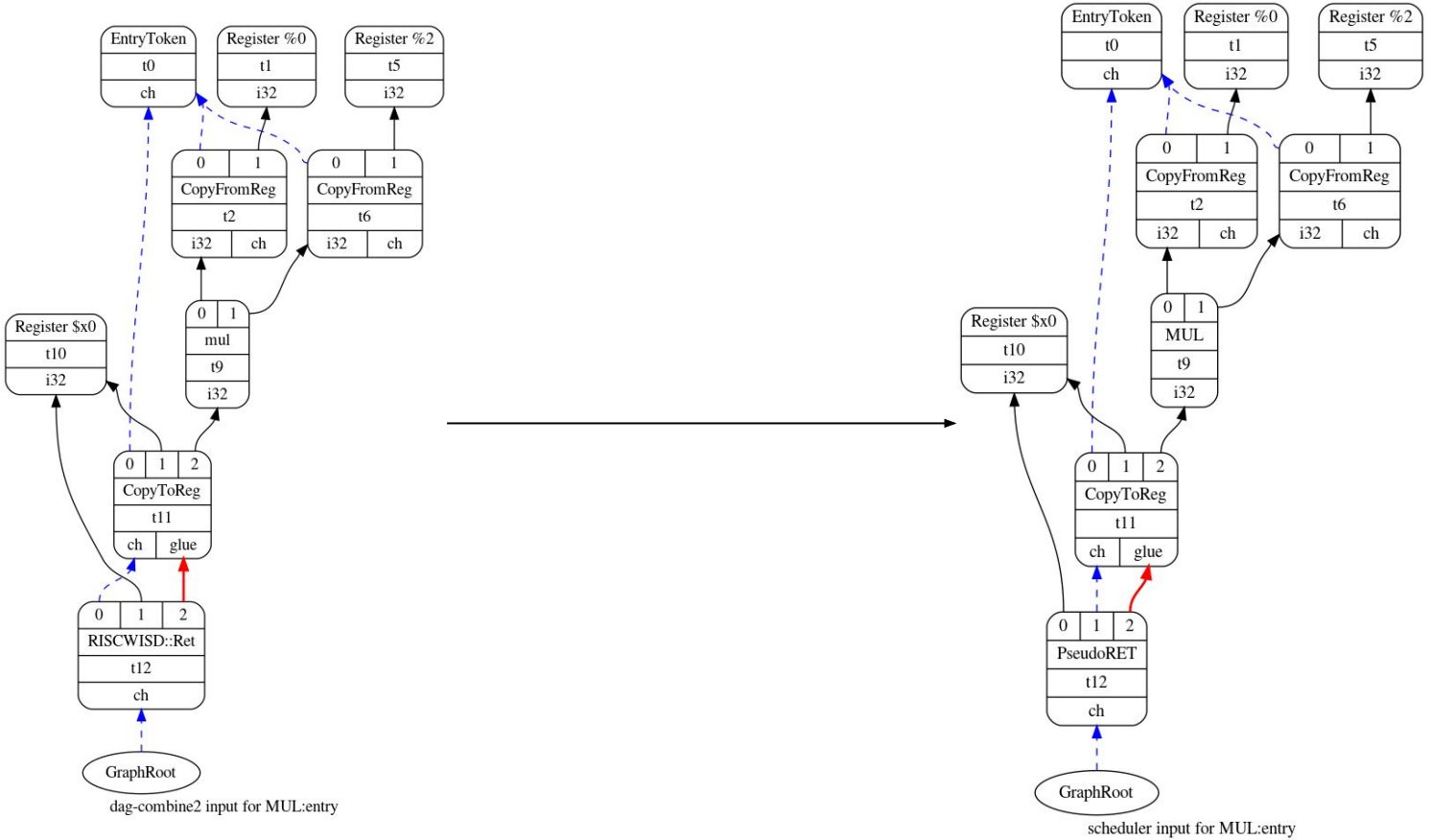
```
define dso_local i32 @MUL(i64 %x, i32 %y) local_unnamed_addr #0 {
entry:
  %0 = trunc i64 %x to i32
  %conv1 = mul i32 %0, %y
  ret i32 %conv1
}
```



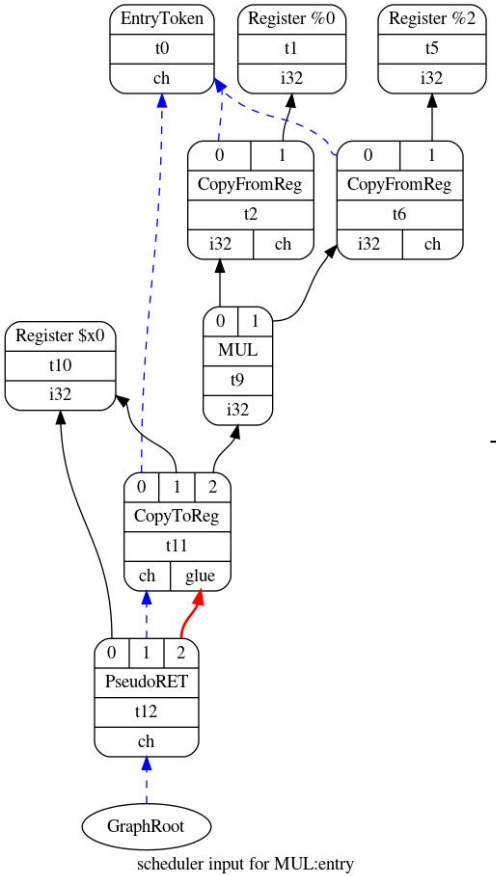
Etapa de Legalização e Otimização



Etapa de Seleção de Instrução



Etapa de Escalonamento



scheduler input for MUL:entry

```
bb.0.entry:
  liveins: $x0, $x2
  %2:gpr = COPY $x2
  %0:gpr = COPY $x0
  %3:gpr = MUL %0:gpr, %2:gpr
  $x0 = COPY %3:gpr
  PseudoRET implicit $x0
```

Cuidado
Nesse ponto, consideramos ainda um número infinito de registradores virtuais

Leitura Recomendada

- LLVM Documentation
  - The LLVM Target-Independent Code Generator
- Andres Amaya Garcia's blog
  - How to Write an LLVM Backend #4: Instruction Selection
- Shiva Chen presentation
  - LLVM Instruction Selection

Resumo

- Seleção de Instruções
- Maximal Munch
- Programação Dinâmica
- Geração de Código

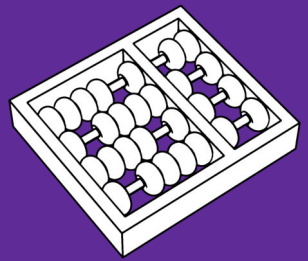
Leitura Recomendada

- Capítulo 9 do livro do Appel
- Capítulo 11 do livro do Cooper

Próxima Aula

- Alocação de Registro

Obrigado!
Merci!



UNICAMP

Pallete

Bubbles

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

BUBBLE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

DRACULA

Tables

Table Title	
Column 1	**Column 2**
One	Two
Three	Four

Table Title	
Column 1	**Column 2**
One	Two
Three	Four

Table Title	
Column 1	**Column 2**
One	Two
Three	Four

Table Title	
Column 1	**Column 2**
One	Two
Three	Four

Emojis