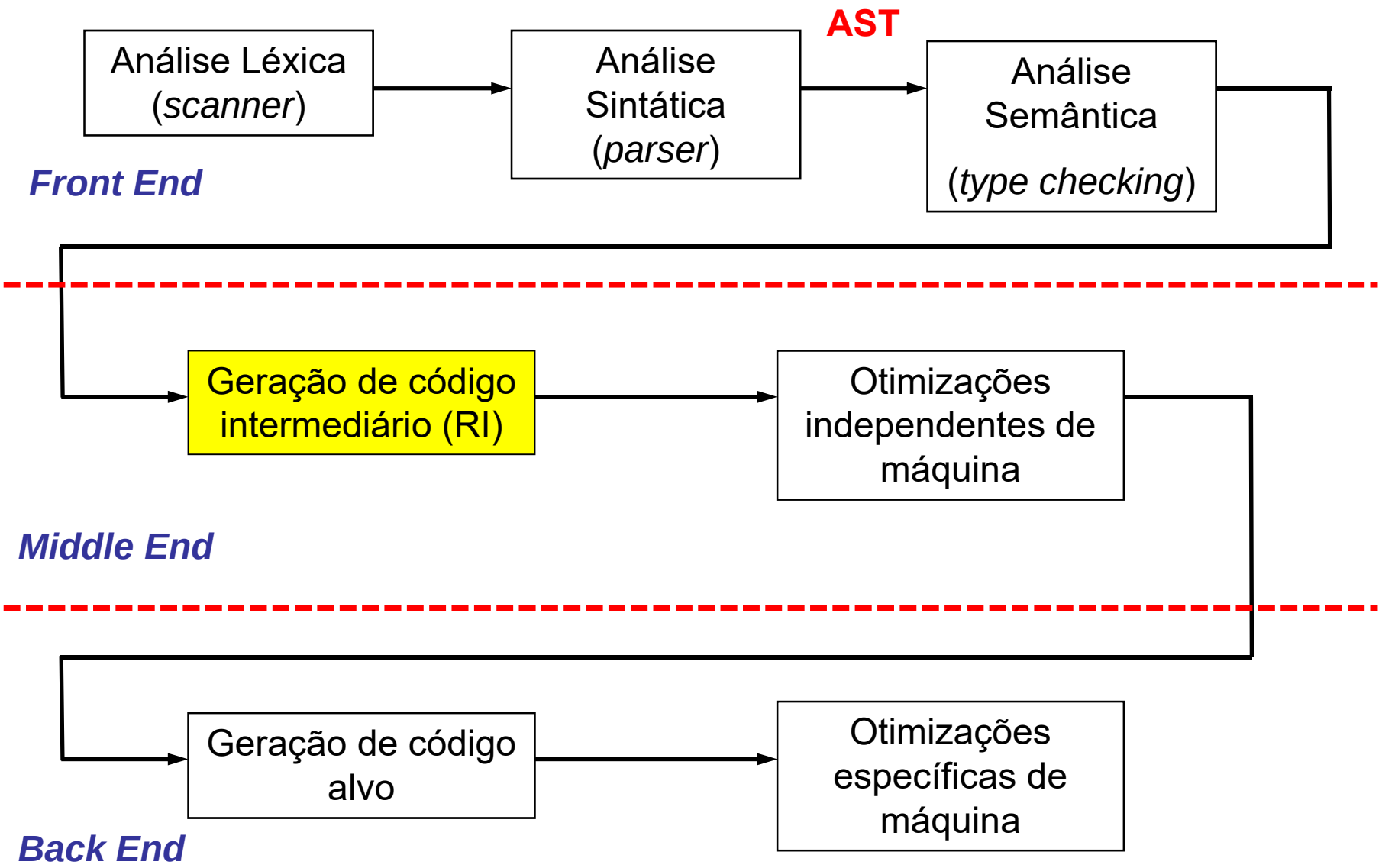
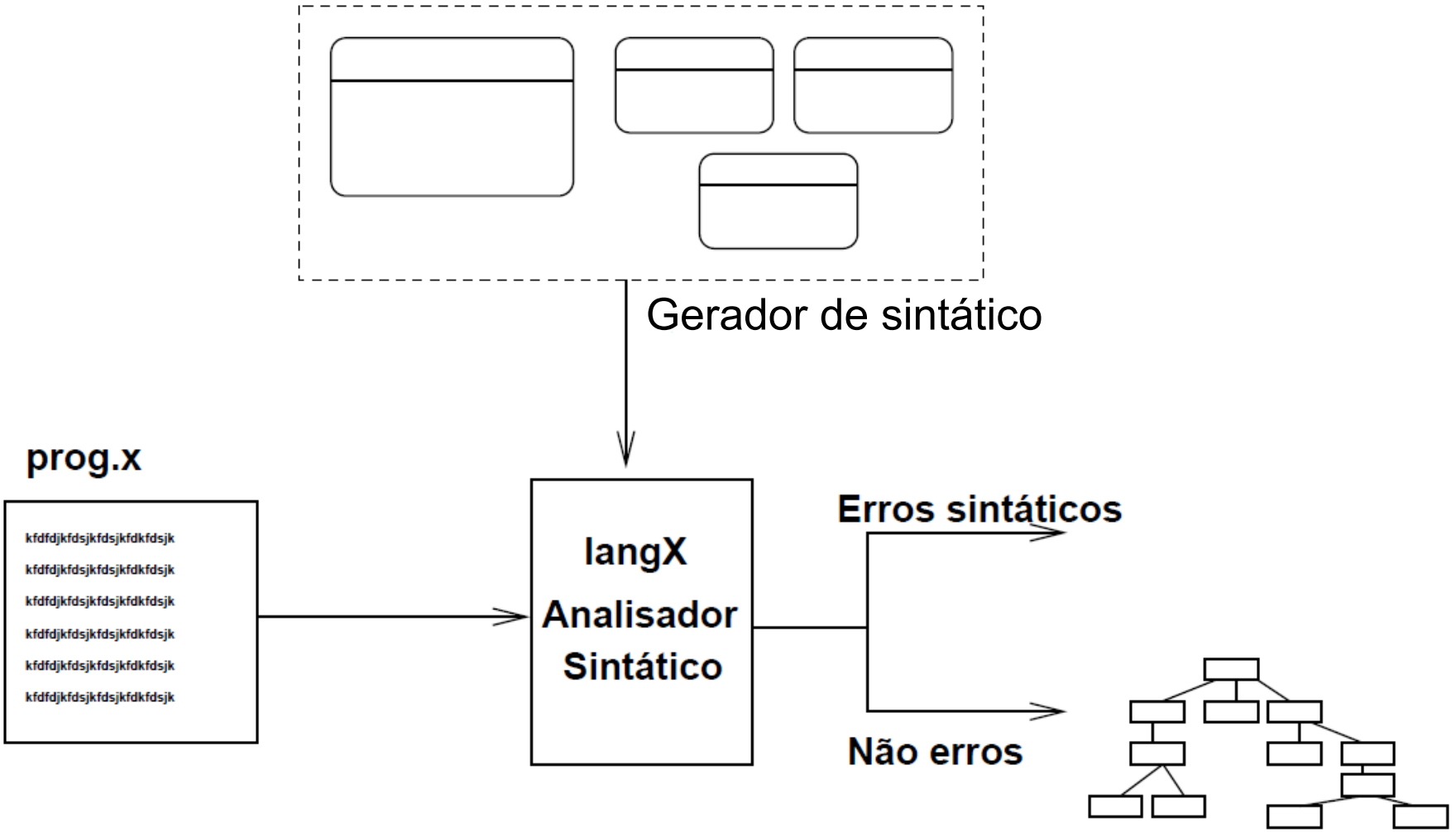
Representação Intermediária

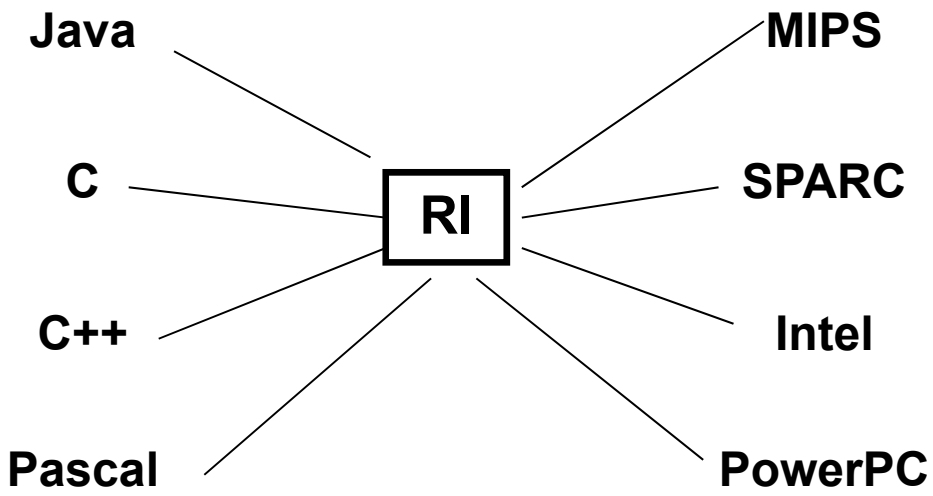
Fluxo do Compilador



Representação Intermediária (RI)



Representação Intermediária (RI)



Queremos compiladores para ***N*** linguagens, direcionados para ***M*** máquinas diferentes.

RI nos possibilita elaborar ***N*** front-ends e ***M*** back-ends, ao invés de ***N.M*** compiladores.

Escolha de uma RI

- Reuso: adequação à linguagem e à arquitetura alvo, custos.
- Projeto: nível, estrutura, expressividade
- “O projeto de uma representação intermediária é uma arte e não uma ciência.” - Steven S. Muchnick
- Pode-se adotar mais de uma RI

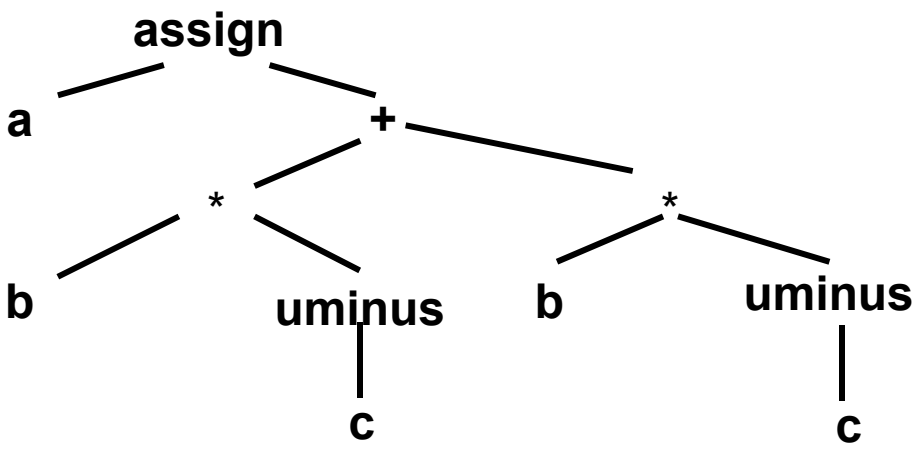
Tipos de RI

- Representação gráfica:
  - Árvores Sintáticas ou DAGs
  - Manipulação pode ser feita através de gramáticas
  - pode ser tanto de alto-nível como nível médio

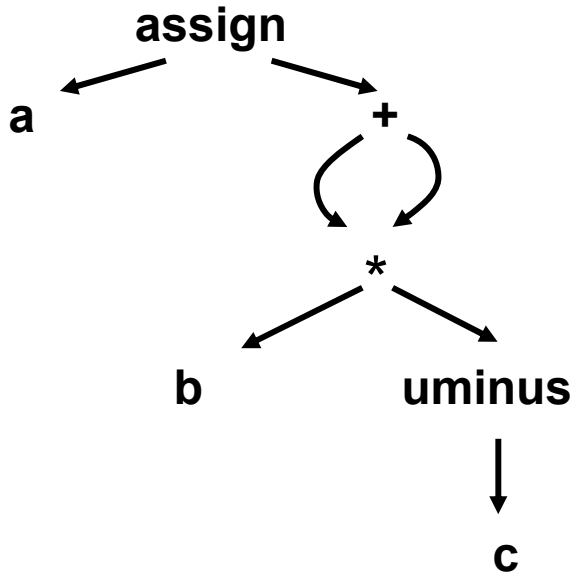
- Representação Linear
  - RI's se assemelham a um pseudo-código para alguma máquina abstrata
  - Java: Bytecode (interpretado ou traduzido)
  - Código de três endereços

Representação em Árvores vs DAG

a := b * -c + b * - c



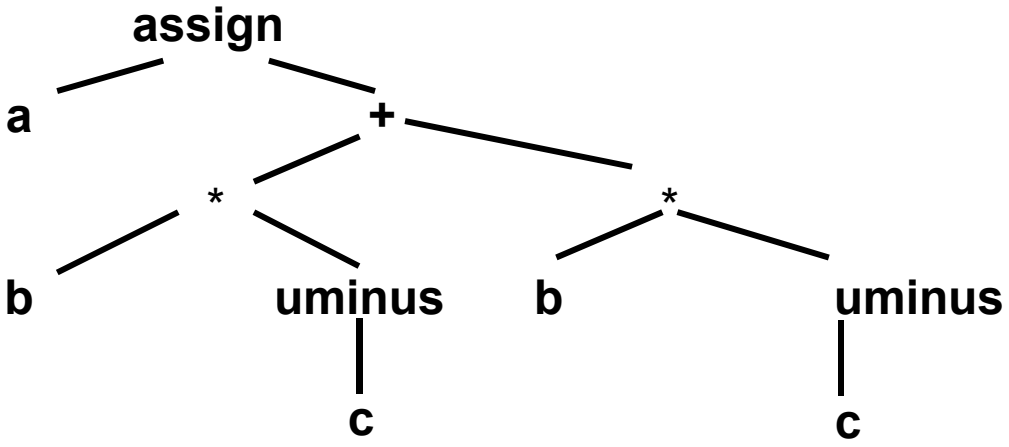
Árvore



DAG

Código de três endereços

- Forma geral: **x** := **y** *op* **z**
- Representação linearizada de uma árvore sintática, ou DAG



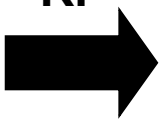

```
t1 := - c
t2 := b * t1
t3 := -c
t4 := b * t3
t5 := t2 + t4
a := t5
```

a := b * -c + b * - c

Exemplo: Produto Interno

```
{
  ...
  prod = 0;
  i = 1;
  while (i <= 20) {
          prod = prod + a[i] * b[i];
          i = i +1;
   }
  ...
}
```

RI

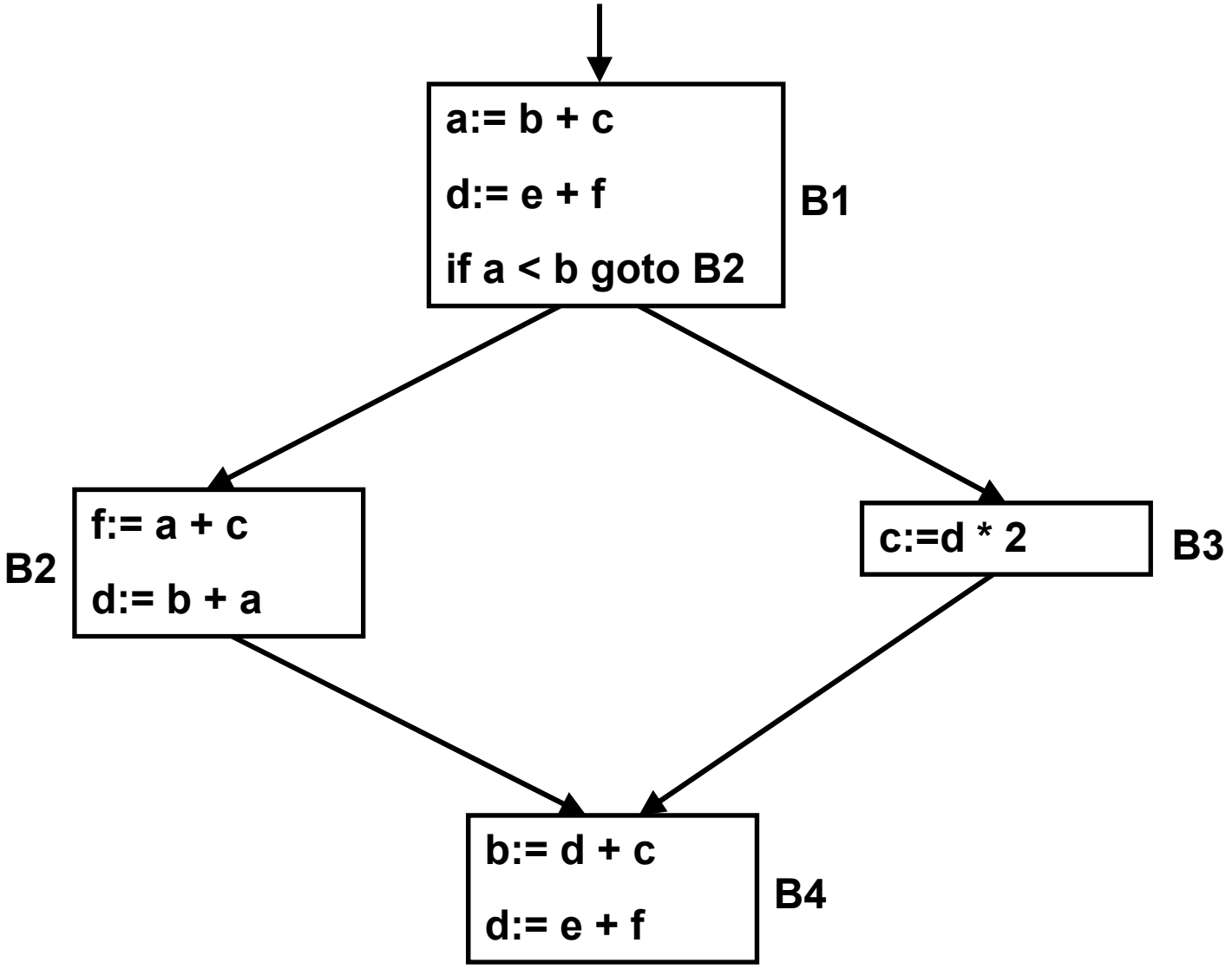


```
(1) prod := 0
(2) i : = 1
(3) t1 := 4 * i
(4) t2 := a [ t1]
(5) t3 := 4 * i
(6) t4 := b [t3]
(7) t5 := t2 * t4
(8) t6 := prod + t5
(9) prod := t6
(10) t7 := i + 1
(11) i := t7
(12) if i <= 20 goto (3)
```

Representação Híbrida

- Combina-se elementos tanto das RI's gráficas (estrutura) como das lineares.
  - Usar RI linear para blocos de código seqüencial e uma representação gráfica (CFG: *control flow graph*) para representar o fluxo de controle entre esses blocos

Exemplo: CFG com código de 3 endereços



Caso Real - GCC

- Várias linguagens: pascal, fortran, C, C++
- Várias arquiteturas alvo: MIPS, SPARC, Intel, Motorola, PowerPC, etc
- Utiliza mais de uma representação intermediária
  - árvore sintática: construída por ações na gramática
  - RTL: tradução de trechos da árvore

Caso Real - GCC

```
d := (a+b)*c
```

```
(insn 8 6 10 (set (reg:SI 2)
            (mem:SI (symbol_ref:SI (“a”)))))
(insn 10 8 12 (set (reg:SI 3)
            (mem:SI (symbol_ref:SI (“b”)))))
(insn 12 10 14 (set (reg:SI 2)
            (plus:SI (reg:SI 2) (reg:SI 3))))
(insn 14 12 15 (set (reg:SI 3)
            (mem:SI (symbol_ref:SI (“c”)))))
(insn 15 14 17 (set (reg:SI 2)
            (mult:SI (reg:SI 2) (reg:SI 3))))
(insn 17 15 19 (set (mem:SI (symbol_ref:SI (“d”)))
            (reg:SI 2)))
```

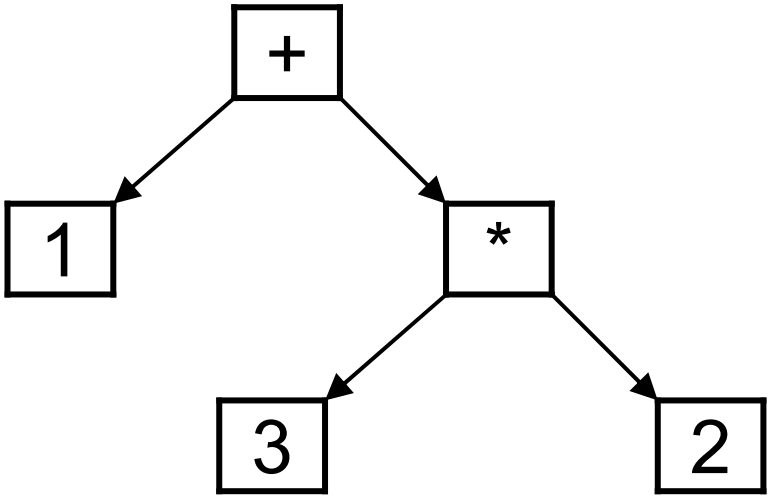
Tradução para RI

- A RI deveria ter componentes descrevendo operações simples
  - MOVE, um simples acesso, um JUMP, etc
- A idéia é quebrar pedaços complicados da AST em um conjunto de operações de máquina
- Cada operação ainda pode gerar mais de uma instrução *assembly* no final

Tradução de AST para código de 3 endereços

Expressão Matemática: 1+3*2

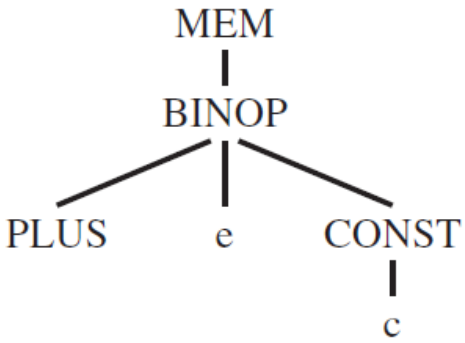
Como seria o código de 3 endereços para a AST?



Seleção de Instruções

Introdução

- A árvore da RI expressa uma operação “simples” em cada nó, como por exempo:
  - Acesso à memória
  - Operador Binário
  - Salto condicional
- Instruções da máquina podem realizar uma ou mais dessas operações
- Que instrução seria essa?



- Encontrar o conjunto de instruções de máquina que implementa uma dada árvore da RI é o objetivo da ***Seleção de Instruções***

Padrões de Árvores

- Expressam as instruções da máquina
- Seleção de instruções:
    - Cubra a árvore da RI com um número de padrões existentes para a máquina alvo, segundo alguma métrica.
- Exemplo:
  - Máquina Jouette
    - r0 contém sempre zero

Padrão 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

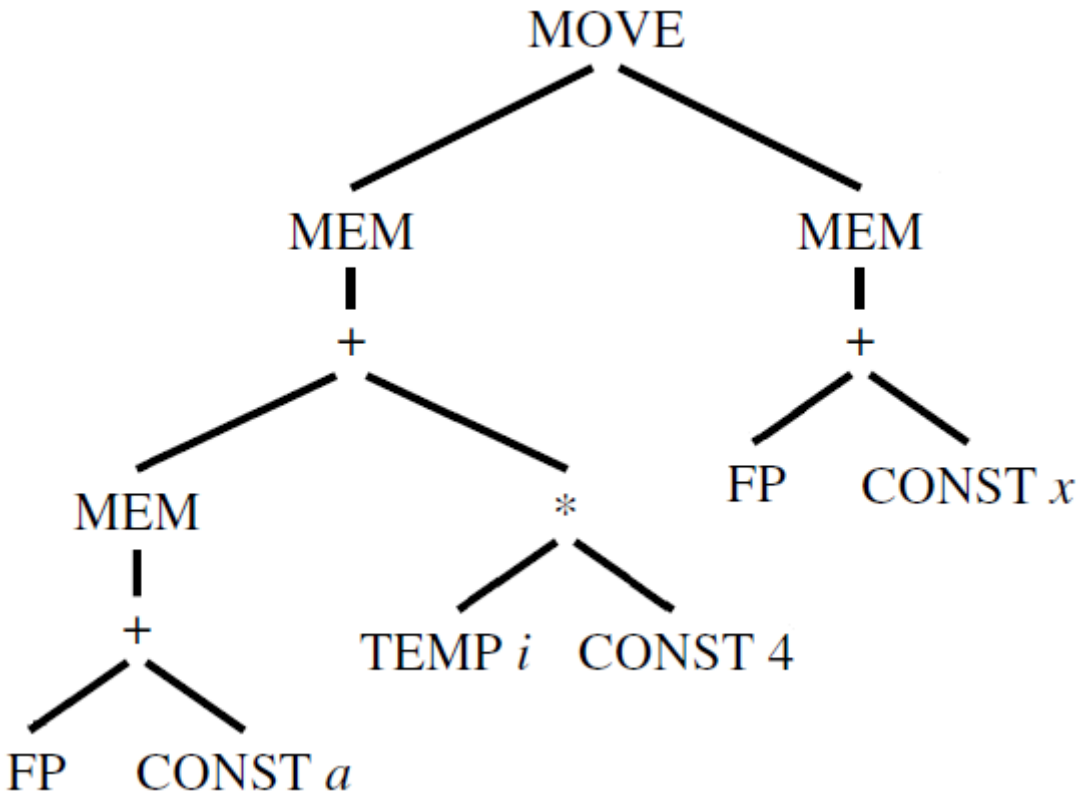
Padrão Jouette

- Primeira linha não gera instrução
  - TEMP é implementado como registrador
- Duas últimas instruções não geram resultado em registrador
  - Alterações na memória
- Uma instrução pode ter mais de um padrão associado
- Objetivo é cobrir a árvore toda, sem sobreposição entre padrões

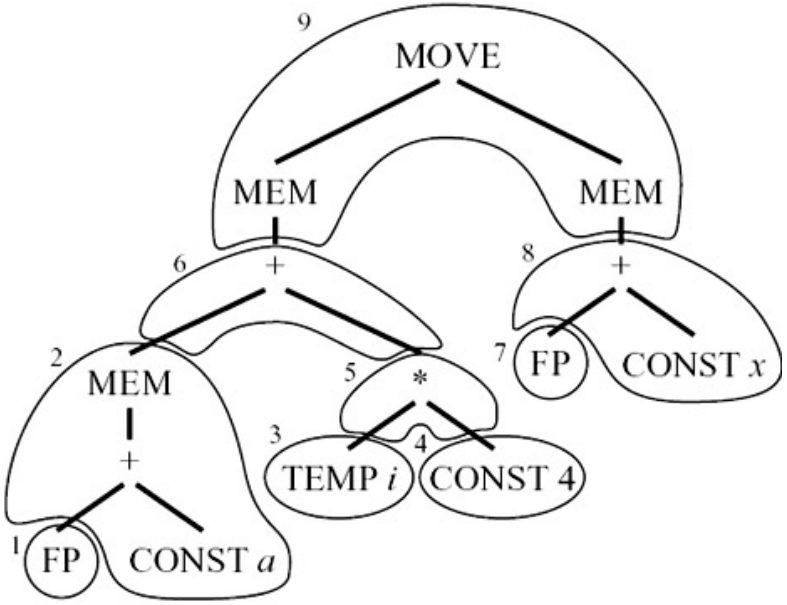
Seleção de Instruções: Maximal Munch

- 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.
- Bastante simples
  - Inicie na raiz
  - Encontre o maior padrão que possa ser encaixado nesse nó
    - Cubra o raiz e provavelmente outros nós
  - Repita o processo para cada sub-árvore a ser coberta
- A cada padrão selecionado, uma instrução é gerada
- Ordem inversa da execução! A raiz é a última a ser executada

Seleção de Instruções: Maximal Munch



Seleção de Instruções: 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	$M[r_1] \leftarrow M[r_2]$

Seleção de Instruções: Minimal Munch

- 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 $M[r1 + 0] \leftarrow r2$

Cobertura da árvore utilizando Minimal Munch