

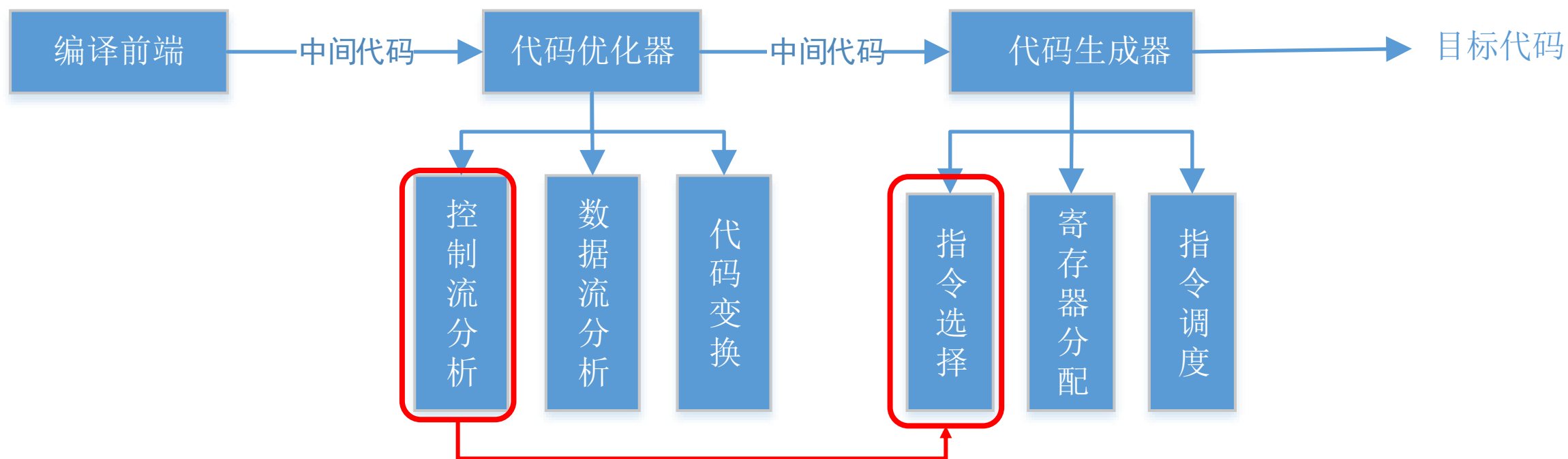
并行编译与优化 Parallel Compiler and Optimization

计算机研究所编译系统室

Lecture 5: Instruction Selection

第五课：指令选择

编译流程



考虑中间表示代码 $rj \leftarrow ri$ ，将寄存器ri的值(整型)复制到rj，将其翻译成目标指令

一种翻译方式是使用寄存器到寄存器mov指令：

`mov rj, ri`

还有其他方式吗？ [填空1]

作答

■ 考虑中间表示代码 $rj \leftarrow ri$, 将寄存器ri的值(整型)复制到rj, 将其翻译成目标指令

■ 一种翻译方式是使用寄存器到寄存器mov指令:

mov rj, ri

■ 其他方式

⊕ add rj, ri, 0

⊕ sub rj, ri, 0

⊕ mul rj, ri, 1

⊕ and rj, ri, ri

⊕ or rj, ri, 0

⊕ xor rj, ri, 0

⊕ store M, ri

load rj, M

指令选择: 从备选指令中选择最佳指令

5.1 指令选择概述

5.2 低层次中间表示

5.3 指令选择方法

- 掌握低层次中间表示的基本形式、基于树模式匹配的指令选择方法和基于窥孔优化的指令选择方法
- 理解指令选择的基本概念和目标

■ 将中间表示(IR)翻译成等价的目标机指令集(ISA)指令序列的过程

- ⊕ 编译中端代码优化器是运行在代码的IR形式上
- ⊕ IR代码必须翻译成ISA指令序列，才能在目标机上执行
- ⊕ 指令选择实现IR到目标机指令的翻译



■ 生成代价(成本)最小的指令序列

- ⊕ 执行时间最短
- ⊕ 代码长度最短
- ⊕ 能耗最低
- ⊕ ...



飞腾编译器
(Aquila)



■ DSP芯片

应用于国防、航空领域
对程序的实时性、代码大小、功耗均有要求
天河编译团队研发面向DSP芯片的编译系统
突破多尺度编译优化和代码生成技术

■ 指令选择的搜索空间巨大

⊕ 目标机ISA有大量备选指令序列可达到同一语义效果

- x86上设置eax寄存器为0

```
mov eax,0    xor eax,eax    sub eax,eax    imul eax,0
```

- RISCv上计算 $r=r*2$

```
mul x5,x5,#2    add x5,x5,x5    sll x5,x5,#1
```

- RISC机器：每个IR操作对应1~2条指令

- CISC机器：可能需要将几个IR操作汇聚为一条指令

⊕ 灵活的寻址模式使搜索空间变大

- 一条指令可以同时完成地址计算、访存操作和寄存器算术运算

```
str fp, [sp, #-4]!
```

⊕ 某些ISA对特定操作增加了附加约束

- 如多字访存指令需要连续的寄存器对

■ 获得全局最优的指令选择是NP完全问题

5.1 指令选择概述

5.2 低层次中间表示

5.3 指令选择方法

■ 高层次中间表示(HIR)

- ⊕ 靠近源语言，更多上下文信息用于进行高层次优化

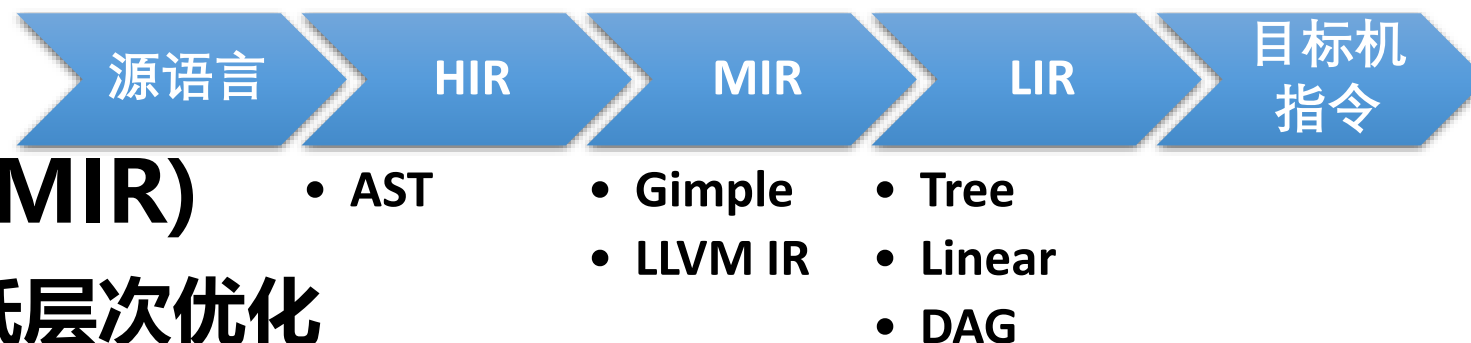
■ 中层次中间表示 (MIR/IR)

- ⊕ 中端编译优化

■ 低层次中间表示(LIR/MIR)

- ⊕ 靠近机器，用于进行低层次优化
- ⊕ 更容易翻译为目标机指令

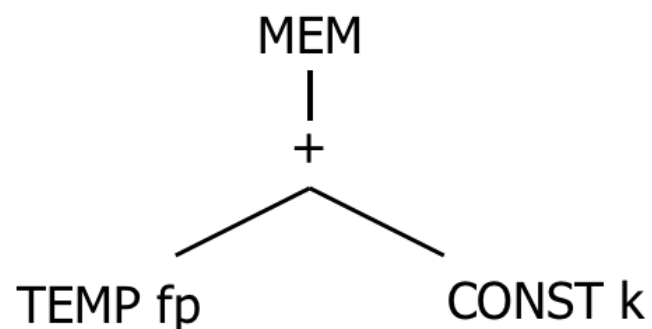
■ 在指令选择之前，可以将中间代码转换为更底层的表示



- **LLVM在指令选择之前，将LLVM IR转换为SelectionDAG**
 - ⊕ **与目标机无关的低层次IR**
 - ⊕ **每个基本块对应一个DAG，DAG中的结点对应指令或者一个操作数，DAG中的边描述了指令间存在数据依赖关系**
 - ⊕ **基于SelectionDAG采用模式匹配进行指令选择**
- **针对不同形式的低层次中间表示，有不同的指令选择方法**

■ 一个化简的LIR示例

- ⊕ 假设目标机为RV32IM指令集架构（RISC机器）
- ⊕ 该目标机包含寄存器-寄存器指令 + load/store指令
- ⊕ 两种形式的LIR：树结构IR和线性IR



MEM(BINOP(PLUS, TEMP fp, CONST k))

■支持的表达式 (Expressions)

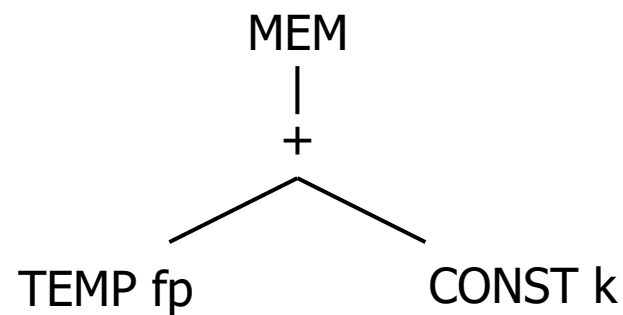
- ⊕ **CONST(i)** – 整型常量*i*
- ⊕ **TEMP(t)** – 临时变量*t* (符号寄存器)
- ⊕ **BINOP(op,e1,e2)** – 执行二元操作*e1 op e2*
- ⊕ **MEM(e)** – 地址*e*处的内存内容
- ⊕ **CALL(f,args)** – 调用函数*f*, 参数列表为*args*

■支持的语句 (Statements)

- ⊕ **MOVE(TEMP t, e)** – 将表达式e的值存储到临时变量t中
- ⊕ **MOVE(MEM(e1), e2)** – 将表达式e2的值存储到表达式e1指示的内存中
- ⊕ **EXP(e)** – 执行表达式e, 忽略结果
- ⊕ **SEQ(s1,s2)** – 先执行语句s1, 再执行语句s2
- ⊕ **NAME(n)** – 汇编指令的标号n
- ⊕ **JUMP(e)** – 无条件跳转到e处, e可以是一个NAME标号或更复杂的情况(如switch)
- ⊕ **CJUMP(op,e1,e2,t,f)** – 条件跳转, 执行e1 op e2; 若结果为真则跳转到标号t, 否则跳转到标号f
- ⊕ **LABEL(n)** – 确定标号n在代码处的位置

■ 读取栈上，栈帧指针fp+偏移k处的变量

⊕ 树IR

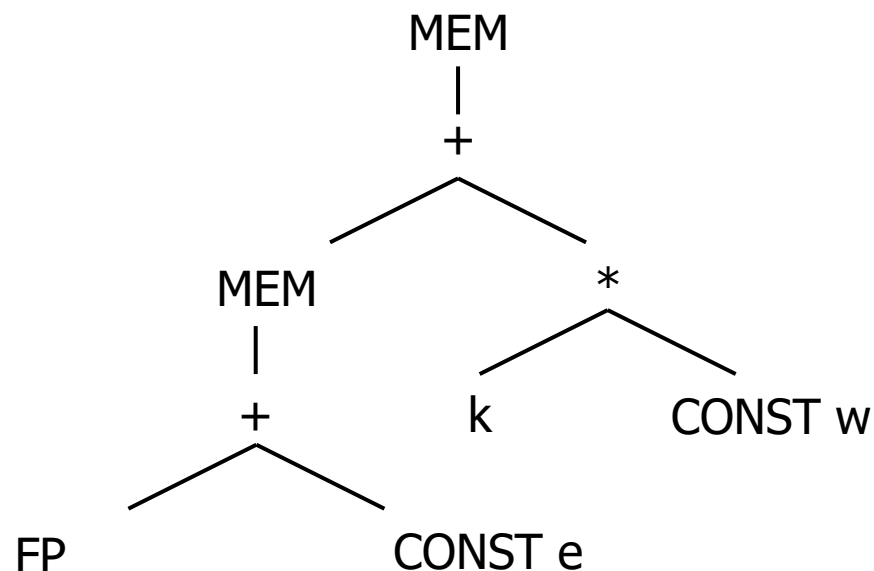


⊕ 线性IR

MEM(BINOP(PLUS, TEMP fp, CONST k))

- 读取栈上，数组e下标为k的元素e[k]，假设存储每个元素需要w个字节，e是栈上存放数组首地址的栈帧偏移

⊕ 树IR



⊕ 线性IR

MEM(BINOP(PLUS, MEM(BINOP(PLUS, TEMP fp, CONST e)), BINOP(MUL, k, CONST w)))

5.1 指令选择概述

5.2 低层次中间表示

5.3 指令选择方法

■ 首届图灵奖获得者 (1966)



"For his influence in the area of advanced programming techniques and compiler construction." --Turing Award Citation

Any noun can be verbed (任何名词都可以变为动词)
--by Alan J. Perlis

■ 基于宏扩展(Macro-expansion)的指令选择

- ⊕ 自顶向下将LIR逐一翻译为指令序列 (one-by-one translation)
- ⊕ ☺ 简单，易于实现
- ⊕ ☹ 难以考虑代码整体质量，不能够利用指令集强大的寻址模式

■ 基于模式匹配 (Pattern-matching) 的指令选择

- ⊕ 利用模式匹配技术选择与一段LIR匹配的指令

- ⊕ 直到得到覆盖全部LIR的指令序列

- ⊕ 基于模式匹配的指令选择方法包括

 - 基于树模式匹配的指令选择方法

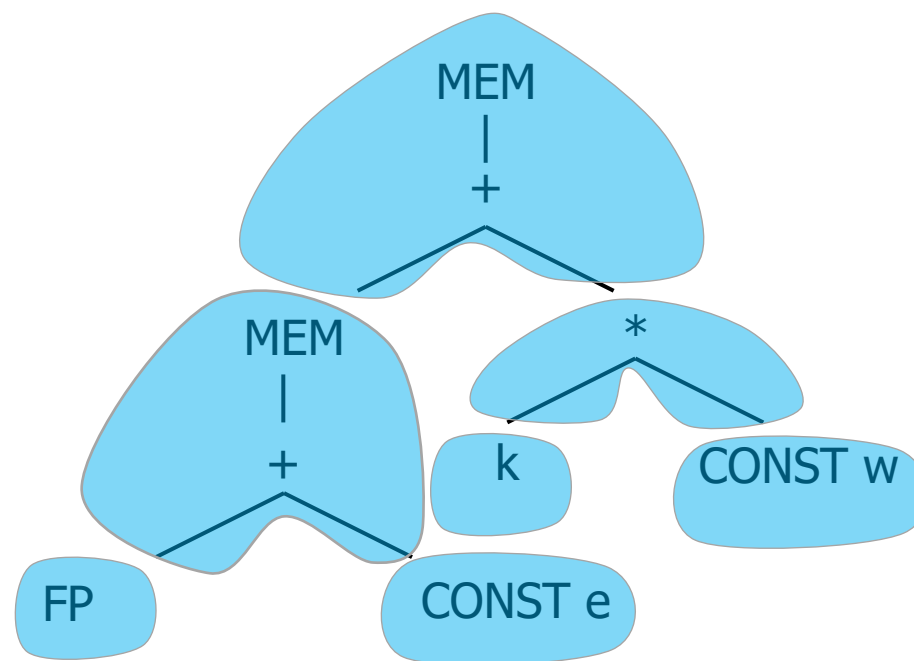
 - 基于窥孔优化的指令选择方法

■ 指令选择阶段假设有足够多的符号寄存器

- ⊕ 到寄存器分配阶段再考虑符号寄存器到物理寄存器的分配

■ IR树

- ⊕ IR树中的一段树枝，称之为**树型** (tree pattern)
- ⊕ 树型可以看成是树中的一个节点及其子节点



■ 操作树

- ⊕ 为了方便进行模式匹配，把目标机指令也表示为树，称之为**操作树**或**瓦片** (tile)

■ 基于树模式匹配的指令选择

- ⊕ 给定一个IR树和一组操作树，将操作树**平铺**(tiling)到IR树上
 - 为每个树型找到一个操作树来覆盖它，操作树与操作树相连接，不重叠
 - 用操作树的最小集合来平铺一个IR树，以提升代码质量

■ 为将指令选择转换为树模式匹配问题，目标机指令表示为操作树

■ 寄存器

⊕ ri

TEMP

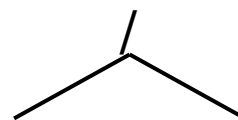
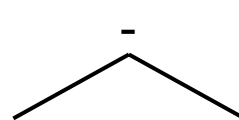
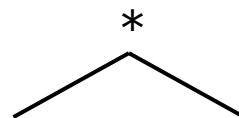
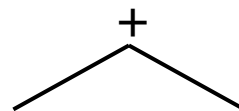
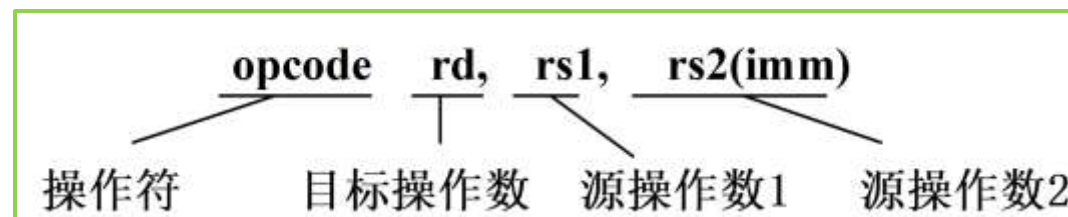
TEMP结点作为寄存器ri使用

■ 算术运算指令

⊕ add rd1, rs1, rs2

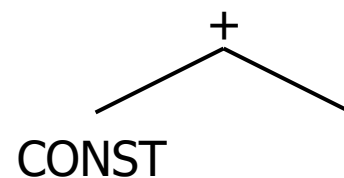
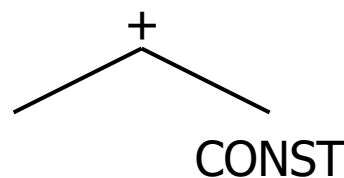
⊕ mul rd1, rs2, rs2

⊕ SUB和DIV类似

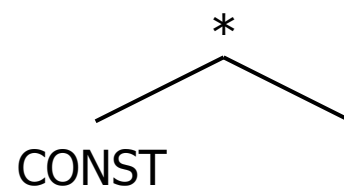
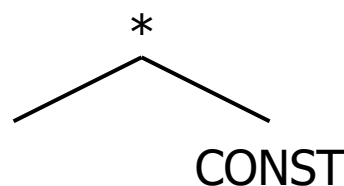


■ 立即数指令

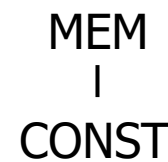
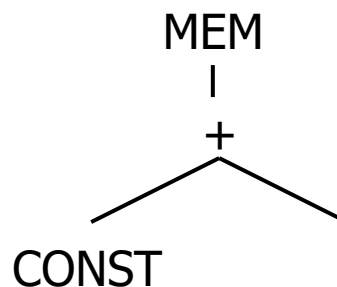
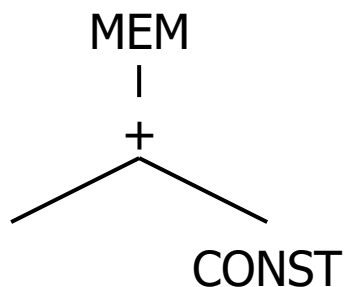
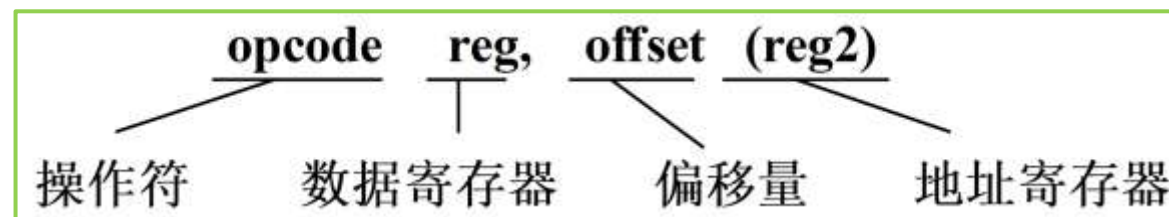
⊕ **addi rd1, rs1, imm**



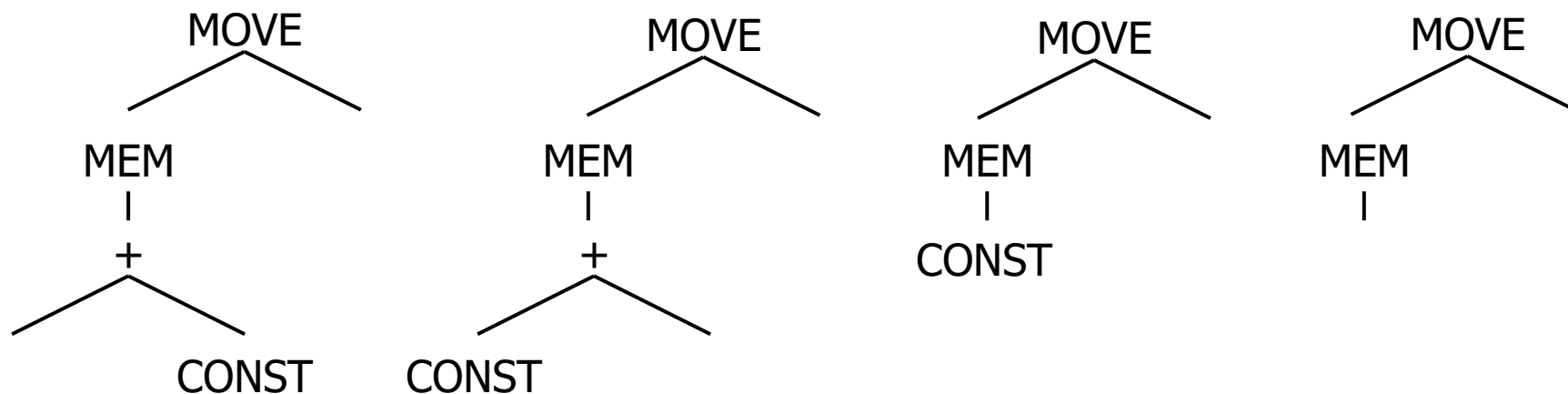
⊕ **muli rd1, rs1, imm**



■ 访存指令(包含寻址)

⊕ **ld r1, offset(r2)** **$r1 = \text{MEM}(r2 + \text{offset})$** 

■ 访存指令(包含寻址)

⊕ **st r1, offset(r2)**➤ **MEM[r2 + offset] = r1**

■ 平铺方案是一组 $\langle \text{node}, \text{op} \rangle$ 对集合

- ⊕ node 是 IR 树中的一个结点

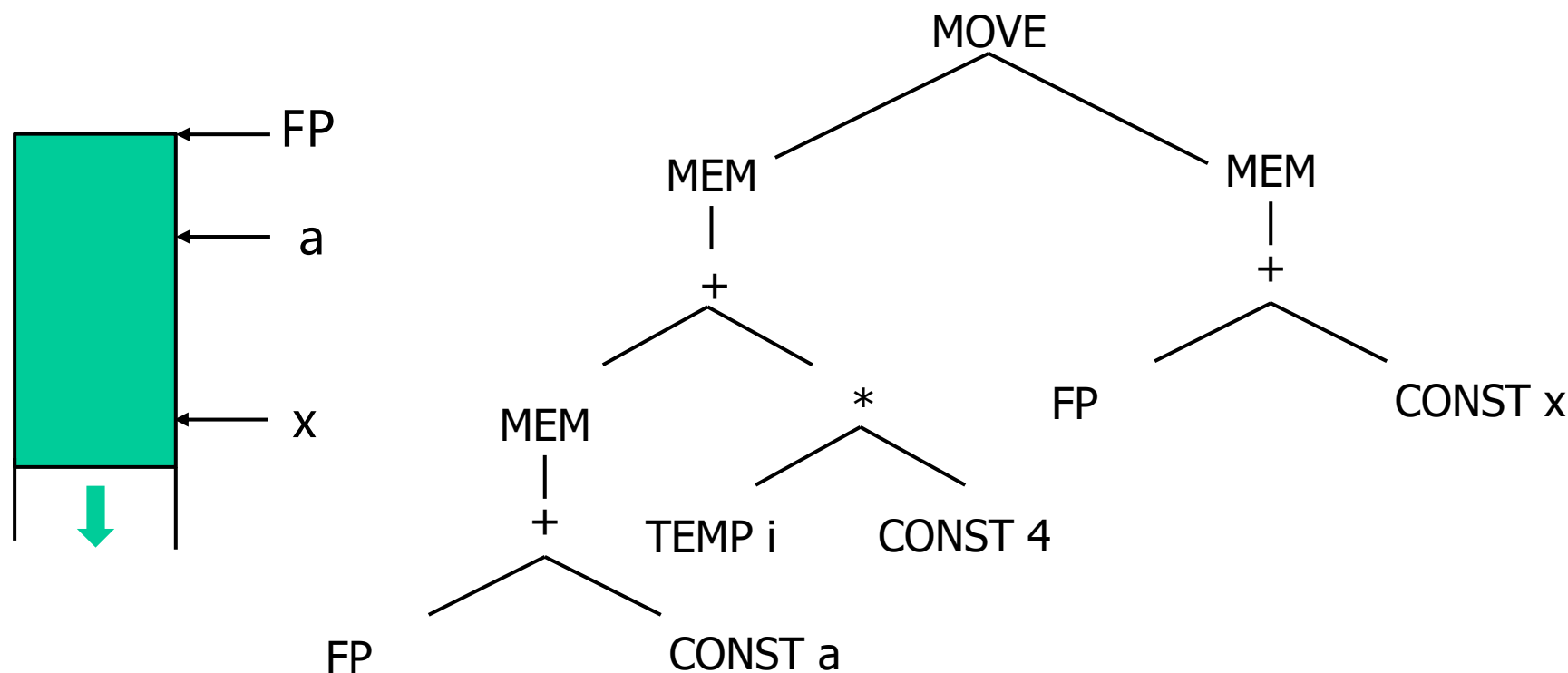
- ⊕ op 是一个操作树 (瓦片)

- ⊕ 一个 $\langle \text{node}, \text{op} \rangle$ 对表示, op 对应的目标机指令可以覆盖以 node 为根的子树

■ 如果一个平铺方案覆盖了一个 IR 树中的每一个 node, 并且每个 op 树 (瓦片) 都与其邻居 op 树 **相连接**, 则该平铺方案实现对该 IR 树的 **覆盖**

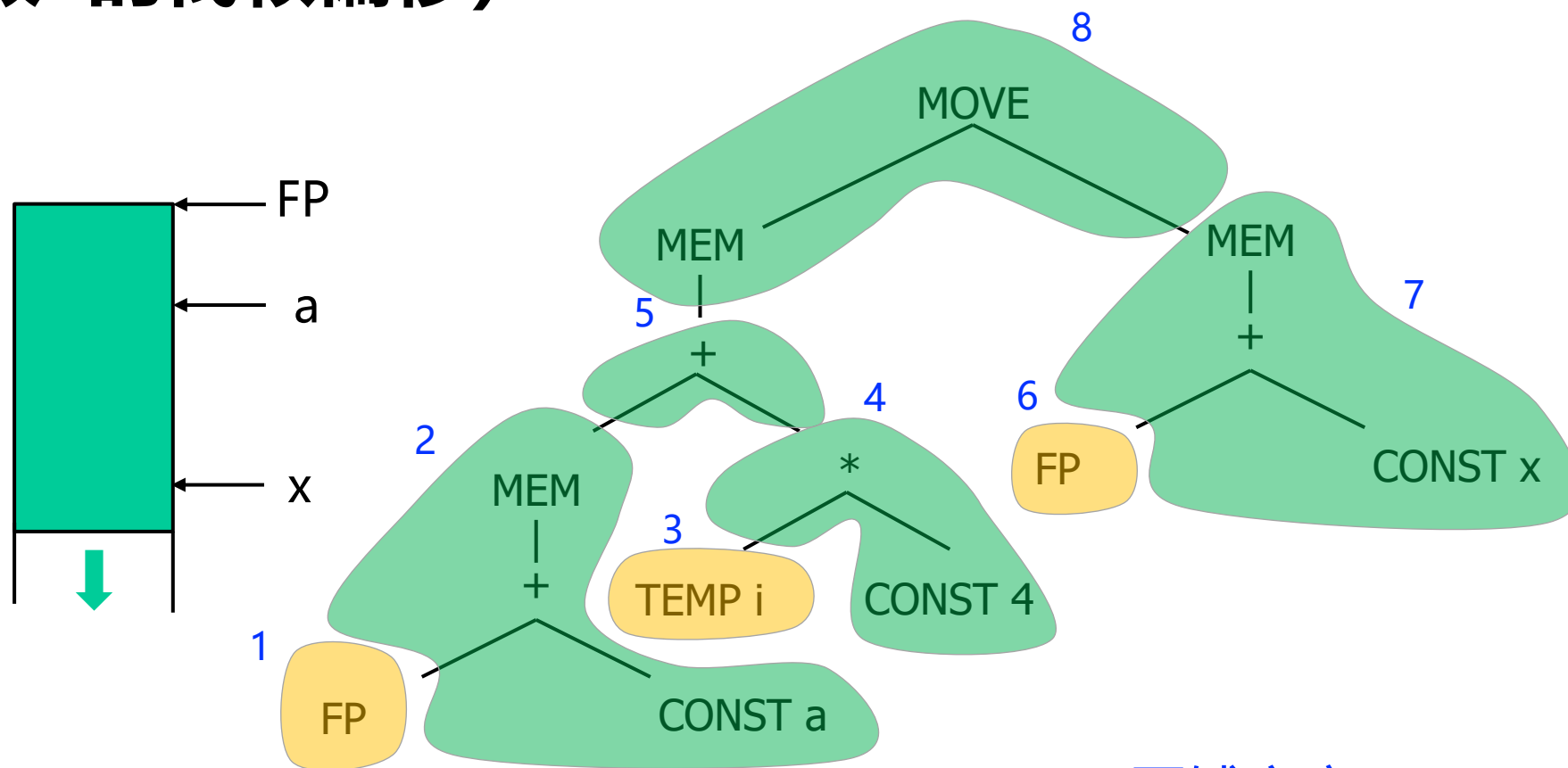
- **两个瓦片相连接**：对于一个 $\langle \text{node}, \text{op} \rangle$ 对，如果其node被平铺方案中另一个op树的一个叶结点涵盖，那么称该op树与其邻居op树相连接
- **当两个op树相连接时，两者公共结点的存储类别必须一致，否则无法将正确的值从较低的树传到较高的树**
 - ⊕ 例如都存储于寄存器中

■ 示例: $a[i] = x$, 假设 a 是栈上的整型数组, i 是寄存器变量, x 是栈上的变量 (a 是栈上存放数组首地址的栈帧偏移, x 是栈上存放 x 的栈帧偏移)



IR树

■ 示例: $a[i] = x$, 假设 a 是栈上的整型数组, i 是寄存器变量, x 是栈上的变量 (a 是栈上存放数组首地址的栈帧偏移, x 是栈上存放 x 的栈帧偏移)



平铺方案

■ 获得基于IR树的平铺方案后，代码生成器可以进行代码生成

- ⊕ 后根次序的自底向上遍历瓦片

- ⊕ 按瓦片顺序，基于**重写规则**，生成目标机指令序列

- ⊕ 使用**相同寄存器名**将相连瓦片在重叠位置连接在一起

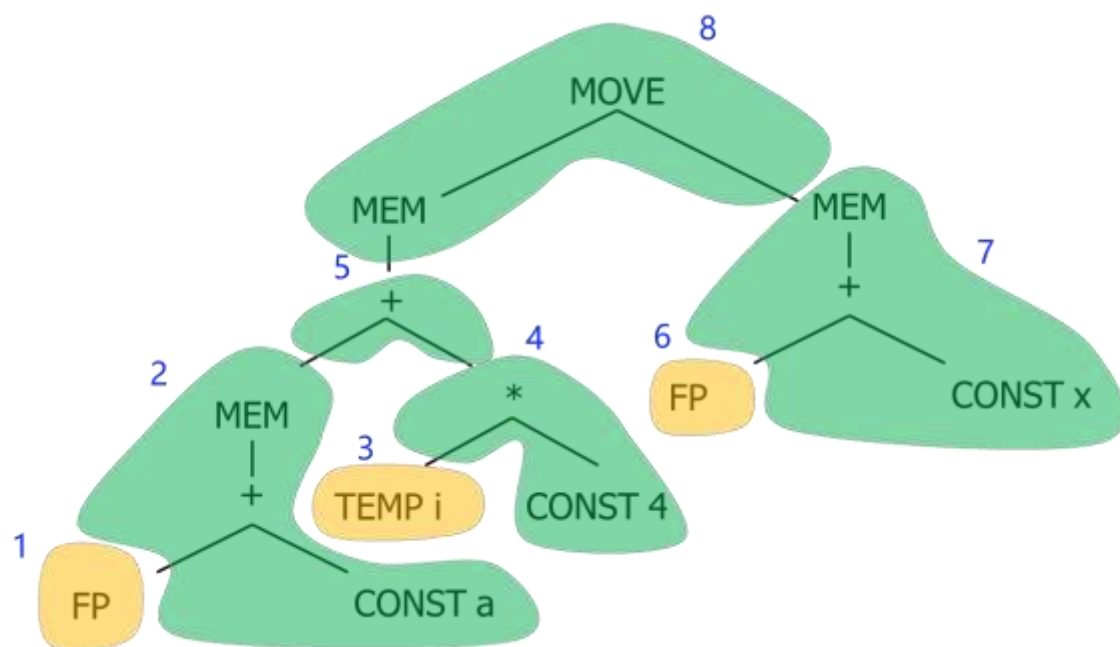
■ 重写规则

- ⊕ 一条重写规则由瓦片，指令模板和相关成本组成

| 规则 | 瓦片 | 指令模板 | 成本 |
|-----|--|---|----|
| (1) | $\text{Reg} \rightarrow +(\text{Reg1}, \text{Reg2})$ | <code>add r_{new}, r1, r2</code> | 1 |
| (2) | $\text{Reg} \rightarrow \text{MEM}(+(\text{Reg1}, \text{offset}))$ | <code>ld r_{new}, offset(r1)</code> | 2 |

■ 基于重写规则生成指令序列 (通过相同寄存器名连接瓦片)

⊕ 子树1、3、6不对应任何指令，因为已经含有正确值的寄存器



```
2 ld    r1, a(fp)  //读取数组a的地址
4 muli  r2, ri, 4   //临时变量i的值在ri中
5 add   r3, r1, r2  //计算a[i]的地址
7 ld    r4, x(fp)  //读取x
8 st    r4, (r3)    //a[i] = x
```

- IR树的每个结点可能有多个与之匹配的瓦片，可以生成多个平铺方案
- 关键是如何快速找到一个良好的平铺方案，满足指令选择的目标 (如程序执行时间最短)
- 从两个层次看这个问题
 - ⊕ 通过树遍历，找到所有的平铺方案
 - ⊕ 将瓦片与成本关联，通过累加成本，找到在每个结点处最低成本的匹配

■ Tile(node n)算法

- ⊕ 为IR树中以n为根结点的子树找到平铺方案
- ⊕ Label(n)标注与结点n匹配的所有瓦片的集合(即与结点n匹配的所有重写规则的集合)
- ⊕ 假设IR树中每个结点至多有两个子结点，且一个重写规则的右侧至多有一个操作
- ⊕ 按后跟次序遍历IR树，确保标注结点n之前先标注其子结点

■ Tile(node n)算法

```
Label(n) <- empty; //按后根次序遍历IR树, Label集合初始为空
if n has two children then //二元操作
    Tile(left(n)) //遍历左子树
    Tile(right(n)) //遍历右子树
    for each rule r that implements n //对能够实现结点n指定的操作的每个规则r
        if left(r) ∈ Label(left(n)) and right(r) ∈ Label(right(n))
            Label(n) <- Label(n) ∪ {r} //如左右子树均匹配, 则r属于Label(n)
else if n has one child then //一元操作原理同二元操作
    Tile(left(n))
    for each rule r that implements n
        if left(r) ∈ Label(left(n))
            Label(n) <- Label(n) ∪ {r}
else //n是叶子结点
    Label(n) <- { all rules that implement n }
```

(4) 找出平铺方案

- **Tile算法可以从整个操作树集合(重写规则集合)中找到所有可能的平铺方案**
- **如何快速找到最低成本的平铺方案?**
 - ⊕ **从所有匹配的平铺方案中找到最低成本匹配 (低效)**
 - ⊕ **利用动态规划方法找到最低成本匹配 (高效)**

■ 利用动态规划方法找到最低成本匹配

- ⊕ 在Tile算法的基础上考虑每个瓦片(重写规则)的成本
- ⊕ 在自底向上遍历中，计算每个结点的成本，选择最低成本的匹配
 - 成本 = 与结点n匹配的规则的成本 + n的所有子树的成本之和
 - 要考虑操作数的存放位置（如寄存器，内存或常量立即数）
- ⊕ 通过累加成本并在每个结点处选择最低成本的匹配，保证生成**局部最优，全局低成本**的平铺方案

- 是基于线性IR的指令选择优化方法
- 窥孔优化(Peehole Matching)基本思想
 - ⊕ 编译器使用**滑动窗口**(也称为**窥孔**)**在代码上移动**
 - ⊕ **每次仅考察窥孔中的指令序列(一小段相邻指令序列)**
 - ⊕ **寻找可以改进的特定模式**
 - ⊕ **识别出一个模式时，使用更好的指令序列重写该模式**
- **快速高效： 有限的模式集合 + 有限的关注区域**

利用窥孔优化技术，将下列代码改写为更有的指令 [填空1]

```
str r1, [fp, #-4]  
ldr r1, [fp, #-4]
```

以ARMv7指令集为例

作答

(1) 窥孔优化示例

■调用store指令后调用load指令, 或调用push调用pop

original

```
str r1, [fp, #-4]  
ldr r1, [fp, #-4]
```

```
push {fp}  
pop {fp}
```

improved

```
str r1, [fp, #-4]
```

```
---
```

以ARMv7指令集为例

■ 简单的代数恒等式

original

`add r1, r1, #0``mul r2, r1, #2``mul r2, r1, #4`

improved

`---``add r2, r1, r1``lsl r2, r1, #2`

以ARMv7指令集为例

■ 跳转指令

original

```
.L1:      B .L1  
          B .L2
```

improved

```
B .L2
```

以ARMv7指令集为例

■ 早期实现

- ⊕ 一组有限的手工编码模式
- ⊕ 窗口很小（通常只有2、3条指令）
- ⊕ 通过穷举搜索进行模式匹配
- ⊕ 有限的模式+小窗口，保证了算法的高效运行

■ 现代实现

- ⊕ 日益复杂的指令集驱动更系统化的处理方法
- ⊕ 处理过程划分为三个任务：展开、简化和匹配
- ⊕ 使用符号解释(symbolic interpretation)和简化的系统化应用

■展开程序

- ⊕ 识别IR形式的输入代码，重写为一系列底层IR(LLIR, Low-Level IR)操作
- ⊕ 该LLIR需表示原IR操作的所有直接影响
 - 如设置add操作的条件码
- ⊕ 基于模板进行重写，各操作逐一展开，无需考虑上下文

■ 简化程序

- ⊕ 通过一个小滑动窗口对LLIR进行一趟处理
- ⊕ 以系统化方式进行局部优化
 - 前向替换
 - 代数化简 ($x+0 \Rightarrow x$)
 - 常量传播
 - 死代码消除
- ⊕ 简化程序是处理过程的核心

■ 匹配程序

⊕ 对着模式库比较简化过的LLIR，寻找能够以最低成本保留住LLIR中所有效应的模式

- 保证正确性的效应必须保留
- 可以有新的不影响正确性的效应

⊕ 输出目标机指令(通常是汇编码)



■ 指令选择概述

- ⊕ 指令选择实现**从IR到指令的映射**
- ⊕ 目标是生成**成本最小**的指令序列
- ⊕ 指令选择的复杂性

■ 低层次中间表示

- ⊕ 在指令选择之前，可以将中间代码转换为更底层的中间表示
- ⊕ 不同形式的低层次中间表示：**树IR和线性IR**

■ 指令选择方法

⊕ 基于宏扩展的指令选择方法

⊕ 基于树模式匹配的指令选择方法

- 用操作树表示目标机指令
- 用操作树去平铺IR树，找到平铺方案
- 基于重写规则进行代码生成

⊕ 基于窥孔优化的指令选择方法

- 快速高效：小滑动窗口+有限的匹配模式

- 《现代编译原理C语言描述》(虎书) 第9章
- 《编译器设计》 第二版 第11章

- 写出表达式 $x \leftarrow a[i] * b * t$ 对应的低层次IR树，基于课件中给出的目标机操作树(瓦片)，进行指令选择，包括生成一种平铺方案(画出所有的瓦片覆盖)，以及给出对应的目标机指令序列。
 - ⊕ 其中， a 是栈上存放数组首地址的栈帧偏移， x 是栈上存放 x 的栈帧偏移， b 是栈上存放 b 的栈帧偏移， t 和 i 是临时变量(其值已位于寄存器中)