代码生成
Part1：概述与简单机器模型

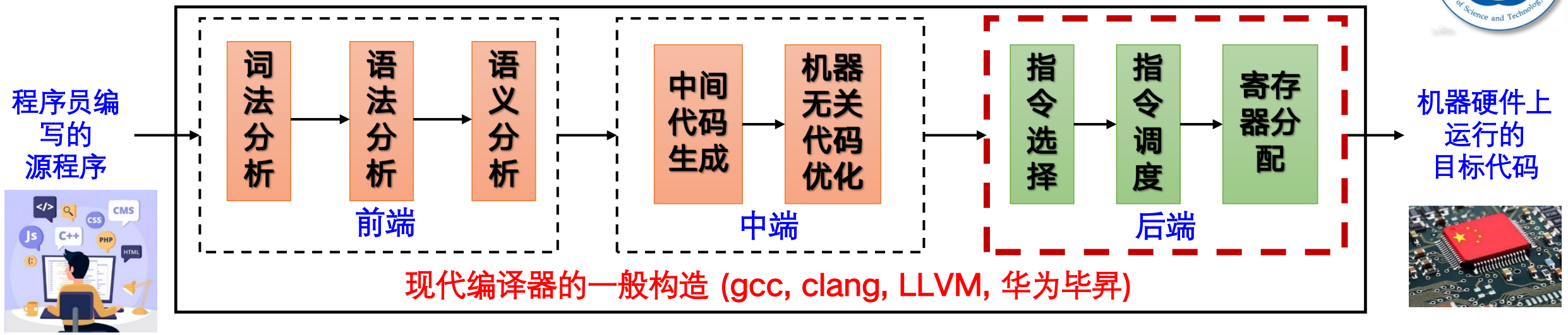
李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2024年11月11日

本节提纲



- 代码生成器任务概述
- 一个简单的目标机器模型
- 指令选择
- 寄存器选择

目标：语义正确+资源有效利用

代码生成器的主要任务及问题

- **指令选择**
  - 为中间表示(IR)语句选择适当的目标机器指令
  - 如果不考虑效率，则十分简单
  - 例：三地址语句 x = y + z 对应的目标代码为

```
LD R0, y /*将y的值加载到寄存器R0中*/
ADD R0, R0, z /*将z加到R0上*/
ST x, R0   /*将R0的值保存到x中*/
```

代码生成器的主要任务及问题

- **指令选择**
  - 为中间表示(IR)语句选择适当的目标机器指令
  - 例： 三地址语句x = y + z; m = x + n对应的目标代码为

```
LD R0, y /*将y的值加载到寄存器R0中*/
ADD R0, R0, z /*将z加到R0上*/
ST x, R0   /*将R0的值保存到x中*/
LD R0, x /*将x的值加载到寄存器R0中*/
ADD R0, R0, n /*将n加到R0上*/
ST m, R0   /*将R0的值保存到m中*/
```

代码生成器的主要任务及问题

- **指令选择**
  - 为中间表示(IR)语句选择适当的目标机器指令
  - 例： 三地址语句x = y + z; m = x + n对应的目标代码为

```
LD R0, y /*将y的值加载到寄存器R0中*/
ADD R0, R0, z /*将z加到R0上*/
ST x, R0   /*将R0的值保存到x中*/
LD R0, x /*将x的值加载到寄存器R0中*/
ADD R0, R0, n /*将n加到R0上*/
ST m, R0   /*将R0的值保存到m中*/
```

冗余指令

代码生成器的主要任务及问题

- 同一中间表示代码可以由多组指令序列来实现，但不同实现之间的效率差别是很大的
  - 例：语句a＝a＋1可以有两种实现方式

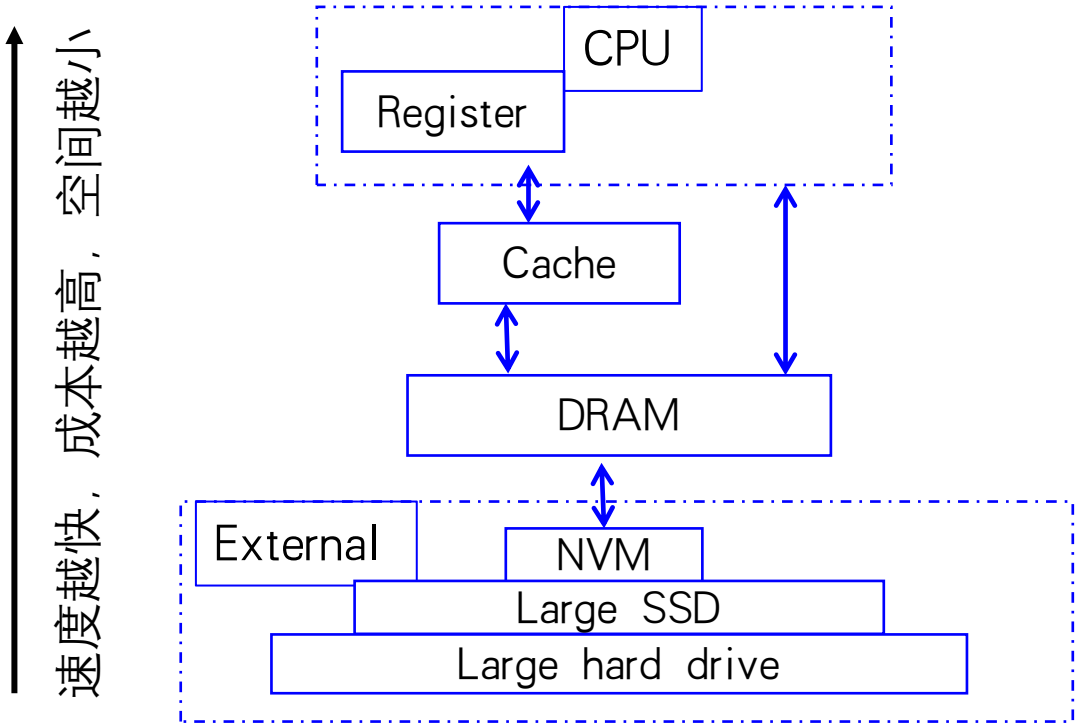
```
LD      R0, a                    // R0 = a
ADD   R0, R0,    #1    //   R0 = R0 + 1
ST      a, R0                    //   a = R0
```

```
INC a
```

- 因此，生成高质量代码需要知道指令代价。

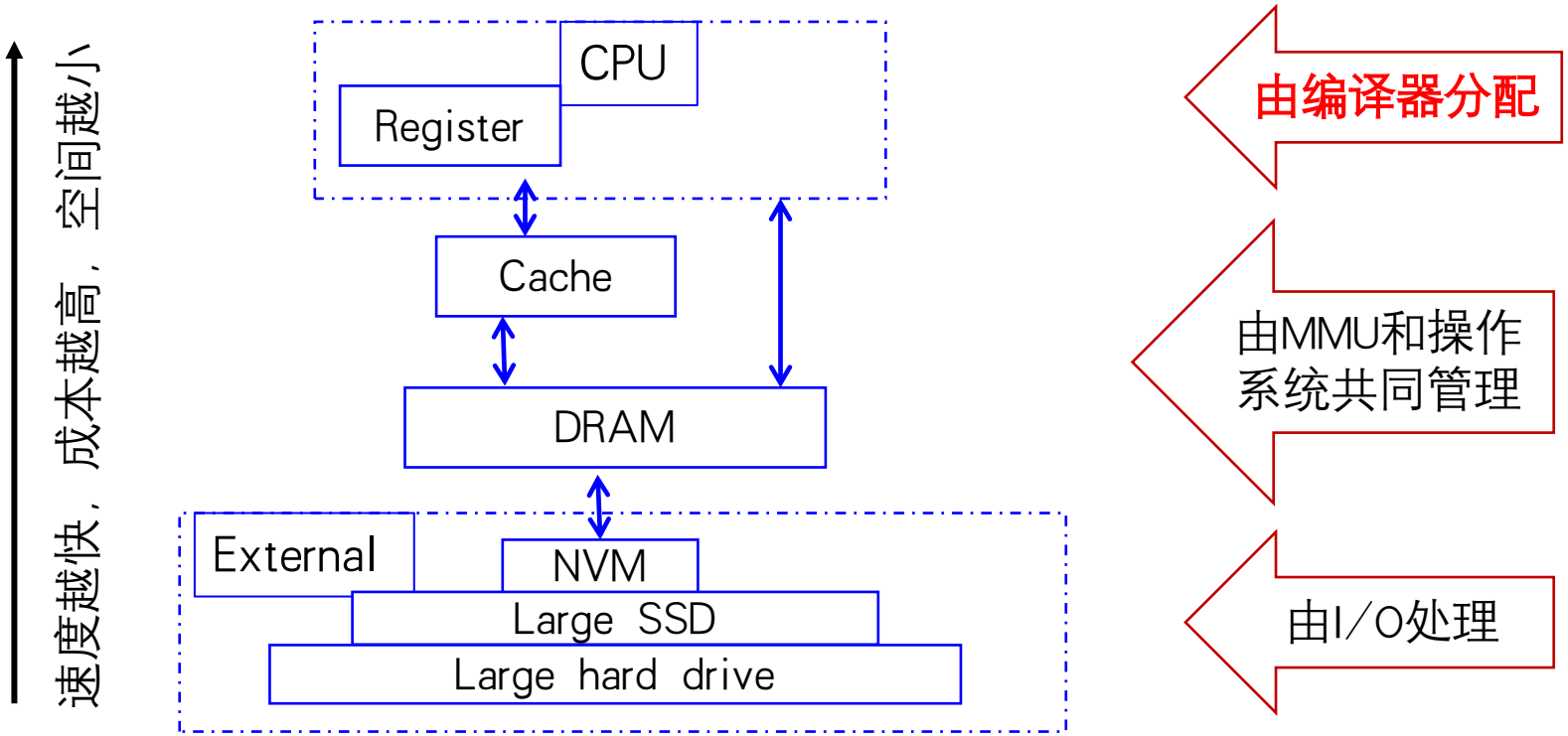
代码生成器的主要任务及问题

- 除了考虑指令的代价和序列长度外，我们还需要考虑运算对象和结果如何存储的问题。



代码生成器的主要任务及问题

- 除了考虑指令的代价和序列长度外，我们还需要考虑运算对象和结果如何存储的问题。



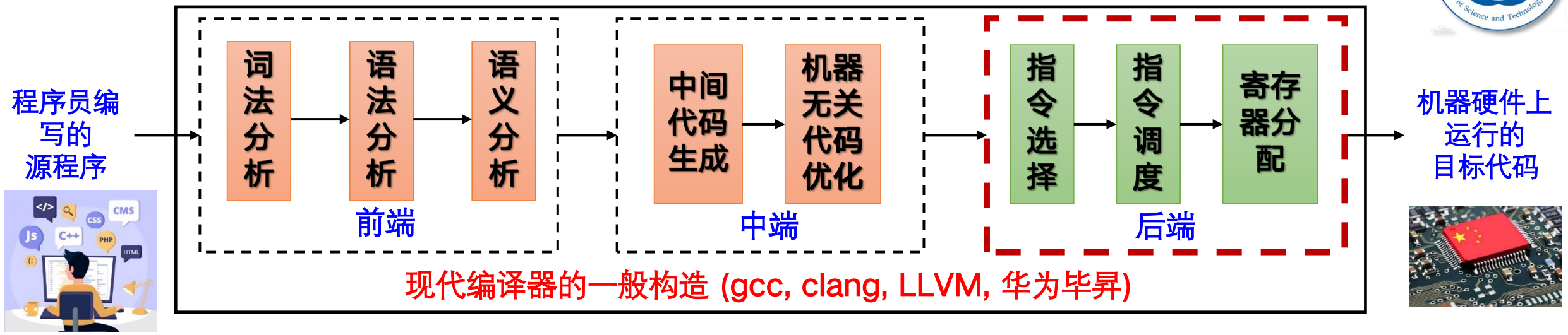
代码生成器的主要任务及问题

- **寄存器分配和指派**
  - 在每个程序点上决定将哪些值放在哪些寄存器中
  - 指定一个变量被存放在哪个寄存器中
  - 在寄存器中获得运算分量比内存要快，但是寄存器数量十分有限
  - 高效利用寄存器可以减少CPU等待的时间

代码生成器的主要任务及问题

- **求值顺序**
  - 计算执行的顺序会影响目标代码的效率
  - 好的执行顺利可以使寄存器需求降低

本节提纲



- 代码生成器任务概述
- **一个简单的目标机器模型**
- **指令选择**
- **寄存器选择**

目标：语义正确+资源有效利用

一个简单的目标机器模型

- **三地址机器模型**
  - 目标机器指令集(也可以称为目标语言)包含LD、ST、运算、跳转等指令
  - 内存按照字节寻址
  - 假设有n个通用寄存器R0, R1, …, Rn-1
  - 假设所有运算分量都是整数
  - 指令之前可能有一个标号

目标机器指令集

- **加载指令　LD dst, addr**
  - LD R0, x
  - LD R1, R2
- **保存指令 ST x, R**
- **运算指令　OP dst, src1, src2**
- **跳转指令**
  - 无条件跳转　　BR L
  - 条件跳转　　Bcond r, L
    - 例：　BLTZ r, L (r是寄存器，LTZ 是 less than zero的缩写)

寻址模式

- **变量名 a**
  - 例： LD R1, a

    R1 = contents(a) 其中contents(a) 表示a位置中存放的内容
- **a(r)：数组访问**
  - a是一个变量，r是一个寄存器
  - 例： LD R1, a(R2)

    R1 = contents(a + contents(R2))
- **c(r)：沿指针取值**
  - c是一个整数，r是一个寄存器
  - 例： LD R1, 100(R2)

    R1 = contents(100 + contents(R2))

寻址模式——间接寻址

- ***r**
  - 在寄存器r的内容所表示的位置上存放的内存位置
  - 例： LD R1, *R2

    R1 = contents(contents(contents(R2)))
- ***c(r)**
  - 在寄存器r中内容加上c后所表示的位置上存放的内存位置
  - 例： LD R1, *100(R2)

    R1 = contents(contents(100 + contents(R2)))

寻址模式——直接数寻址

- **#c**
  - c是一个常数
  - 例： LD R1, #100

    R1 = 100

指令的代价

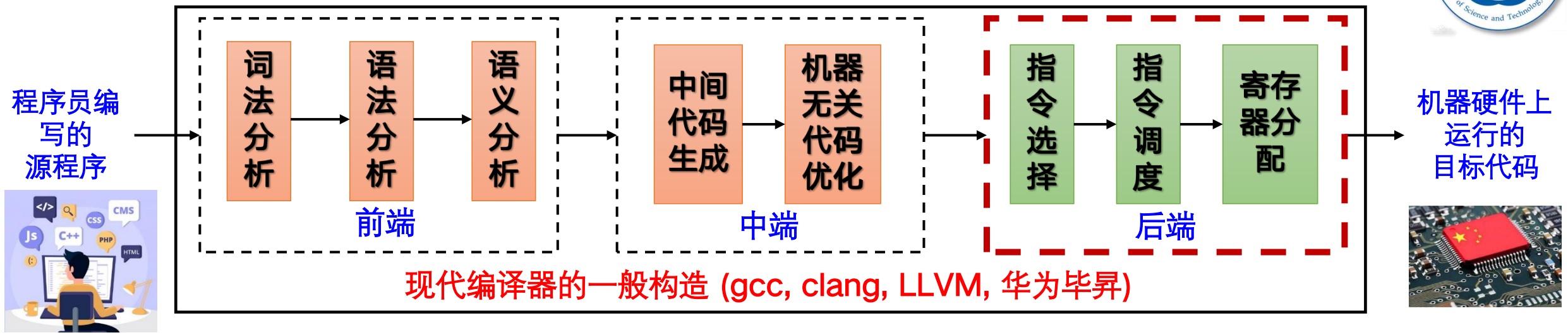
- 在上述简单的目标机器上，指令代价简化为

1 + 指令的源和目的寻址模式(addressing mode)的附加代价

- 寄存器寻址模式附加代价为0
- 涉及内存位置或者常数的寻址方式代价为1

指令	代价	
LD R0，R1	1	寄存器
LD R0，M	2	寄存器+内存
LD R1，*100(R2)	2	寄存器+内存

本节提纲



- 代码生成器任务概述
- 一个简单的目标机器模型
- **指令选择**
- **寄存器选择**

目标：语义正确+资源有效利用

运算语句的目标代码

- 三地址指令
  - x = y – z
- 目标代码
  - LD R1, y　　// R1 = y
  - LD R2, z　　// R2 = z
  - SUB R1, R1, R2　　// R1 = R1 – R2
  - ST　x, R1　　// x = R1

数组寻址语句的目标代码

- **三地址指令**
  - b = a[i]
  - a是一个实数数组，每个实数占8个字节
- **目标代码**
  - LD R1, i　　// R1 = i
  - MUL R1, R1, 8　　// R1 = R1 * 8
  - LD R2, a(R1)　　// R2 = contents(a + contents(R1))
  - ST b, R2　　// b = R2

数组寻址语句的目标代码

- **三地址指令**
  - a[j] = c
  - a是一个实数数组，每个实数占8个字节
- **目标代码**
  - LD R1, c　　// R1 = c
  - LD R2, j　　// R2 = j
  - MUL R2, R2, 8　　// R2 = R2 * 8
  - ST a(R2), R1　　// contents(a + contents(R2)) = R1

数组寻址语句的目标代码

- 三地址指令
  - x = *p
- 目标代码
  - LD R1, p　　// R1 = p
  - LD R2, 0(R1)　　// R2 = contents(0 + contents(R1))
  - ST x, R2　　// x = R2

数组寻址语句的目标代码

- 三地址指令
  - *p = y
- 目标代码
  - LD R1, p　　// R1 = p
  - LD R2, y　　// R2 = y
  - ST 0(R1), R2　　// contents(0 + contents(R1)) = R2

条件跳转语句的目标代码

- **三地址指令**
  - if x < y goto L
- **目标代码**
  - LD R1, x　　// R1 = x
  - LD R2, y　　// R2 = y
  - SUB R1, R1, R2　　// R1 = R1 – R2
  - BLTZ R1, M　　// if R1 < 0 jump to M

M是标号为L的三地址指令所产生的目标代码中的第一条指令的标号

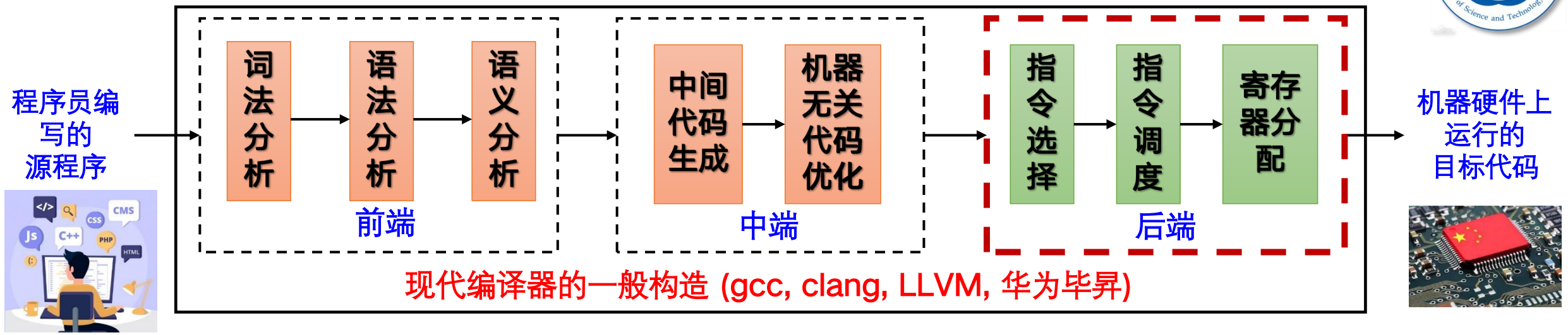
指令的代价——举例

三地址代码	x = y – z	b = a[i]	if x < y goto L

目标指令序列	LD r1 , y
	LD r2 , z
	SUB r1,r1,r2
	ST x, r1
		LD r1,i
		MUL r1,r1,8
		LD r2, a(r1)
		ST b, r2
			LD r1, x
			LD r2, y
			SUB r1,r1,r2
			BLTZ r1, M

代价	2+2+1+2 = 7	2+2+2+2=8	2+2+1+2=7

本节提纲



- 代码生成器任务概述
- 一个简单的目标机器模型
- 指令选择
- **寄存器选择**

目标：语义正确+资源有效利用

寄存器及地址描述符

- **寄存器描述符 (register descriptor)**
  - 记录每个寄存器当前存放的是哪些变量的值
- **地址描述符 (address descriptor)**
  - 记录运行时每个名字的当前值存放在哪个或者哪些位置
  - 该位置可能是寄存器、栈单元、内存地址或者是它们的某个集合
  - 这些信息可以存放在该变量名对应的符号表条目中

三地址语句的目标代码生成

- **对每个形如x = y op z的三地址指令I**
  - 调用寄存器选择函数getReg(I)来为x、y、z选择寄存器Rx, Ry, Rz
  - 如果Ry中存放的不是y，则生成指令“LD Ry, y’”，y’是存放y的内存位置之一
  - 对于Rz和z的处理与上一步骤类似
  - 最后，生成目标指令“OP Rx, Ry, Rz”

基本块的收尾处理

- 对于一个在基本块出口处可能活跃的变量x，如果它的地址描述符表明它的值没有存放在x的内存位置上，则生成指令“ST x, R”
  - R是在基本块结尾处存放x值的寄存器

管理寄存器和地址描述符

- **当生成加载、保存和其他指令时，必须同时更新寄存器和地址描述符**
  - 对于LD R, x指令
    - 修改R的寄存器描述符，使之只包含x
    - 修改x的地址描述符，把R作为新增位置加入到x的位置集合中
    - 从任何不同于x的地址描述符中删除R

管理寄存器和地址描述符

- **当生成加载、保存和其他指令时，必须同时更新寄存器和地址描述符**
  - 对于OP Rx, Ry, Rz指令
    - 修改Rx的寄存器描述符，使之只包含x
    - 从任何不同于Rx的寄存器描述符中删除x
    - 修改x的地址描述符，使之只包含位置Rx
    - 从任何不同于x的地址描述符中删除Rx

管理寄存器和地址描述符

- 当生成加载、保存和其他指令时，必须同时更新寄存器和地址描述符
  - 对于ST x, R指令
    - 修改x的地址描述符，使之包含自己的内存位置

管理寄存器和地址描述符

- 当生成加载、保存和其他指令时，必须同时更新寄存器和地址描述符
  - 对于x = y指令，假设总是为x和y分配同一个寄存器，如果需要生成“LD Ry, y’”，则
    - 修改Ry的寄存器描述符，使之只包含y
    - 修改y的地址描述符，把Ry作为新增位置加入y的位置集合中
    - 从任何不同于y的地址描述符中删除Ry
    - 修改Ry的寄存器描述符，使之也包含x
    - 修改x的地址描述符，使之只包含Ry

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

R1	R2	R3	a	b	c	d	t	u	v
			a	b	c	d			

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3
a	b	

a	b	c	d	t	u	v
a，R1	b，R2	c	d			

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

```
LD R1, a
LD R2, b
SUB R2, R1, R2
```

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a，R1	b	c	d	R2		

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

LD R3, c

SUB R1, R1, R3

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a，R1	b	c	d	R2		

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

LD R3, c

SUB R1, R1, R3

R1	R2	R3
a	t	c

a	b	c	d	t	u	v
a，R1	b	c，R3	d	R2		

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

LD R3, c

SUB R1, R1, R3

R1	R2	R3
u	t	c

a	b	c	d	t	u	v
a	b	c，R3	d	R2	R1	

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

ADD R3, R2, R1

R1	R2	R3
u	t	c

a	b	c	d	t	u	v
a	b	c，R3	d	R2	R1	

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

ADD R3, R2, R1

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

LD R2, d

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

LD R2, d

R1	R2	R3
u	a，d	v

a	b	c	d	t	u	v
R2	b	c	d，R2		R1	R3

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

t、u、v为临时变量

a、b、c、d在出口处活跃

ADD R1, R3, R1

R1	R2	R3
u	a，d	v

a	b	c	d	t	u	v
R2	b	c	d，R2		R1	R3

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

exit

t、u、v为临时变量

a、b、c、d在出口处活跃

ST a, R2

ST d, R1

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
R2	b	c	R1			R3

寄存器分配选择——举例

基本块三地址代码如下：

t = a – b

u = a – c

v = t + u

a = d

d = v + u

exit

t、u、v为临时变量

a、b、c、d在出口处活跃

ST a, R2

ST d, R1

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
a，R2	b	c	d，R1			R3

寄存器选择函数的设计

- 函数*getReg*返回保存x = y *op* z的x值的场所L
  - 如果名字y在R中，这个R不含其它名字的值,并且在执行x = y *op* z后y不再有下次引用，那么返回这个R作为L
  - 否则，如果有的话，返回一个空闲寄存器
  - 否则，如果x在块中有下次引用，或者*op*是必须用寄存器的算符，那么找一个已被占用的寄存器R(可能产ST M, R 指令，并修改 M的描述)
  - 否则，如果x在基本块中不再引用，或者找不到适当的被占用寄存器，选择x的内存单元作为L

一起努力
打造国产基础软硬件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2024年11月11日