中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

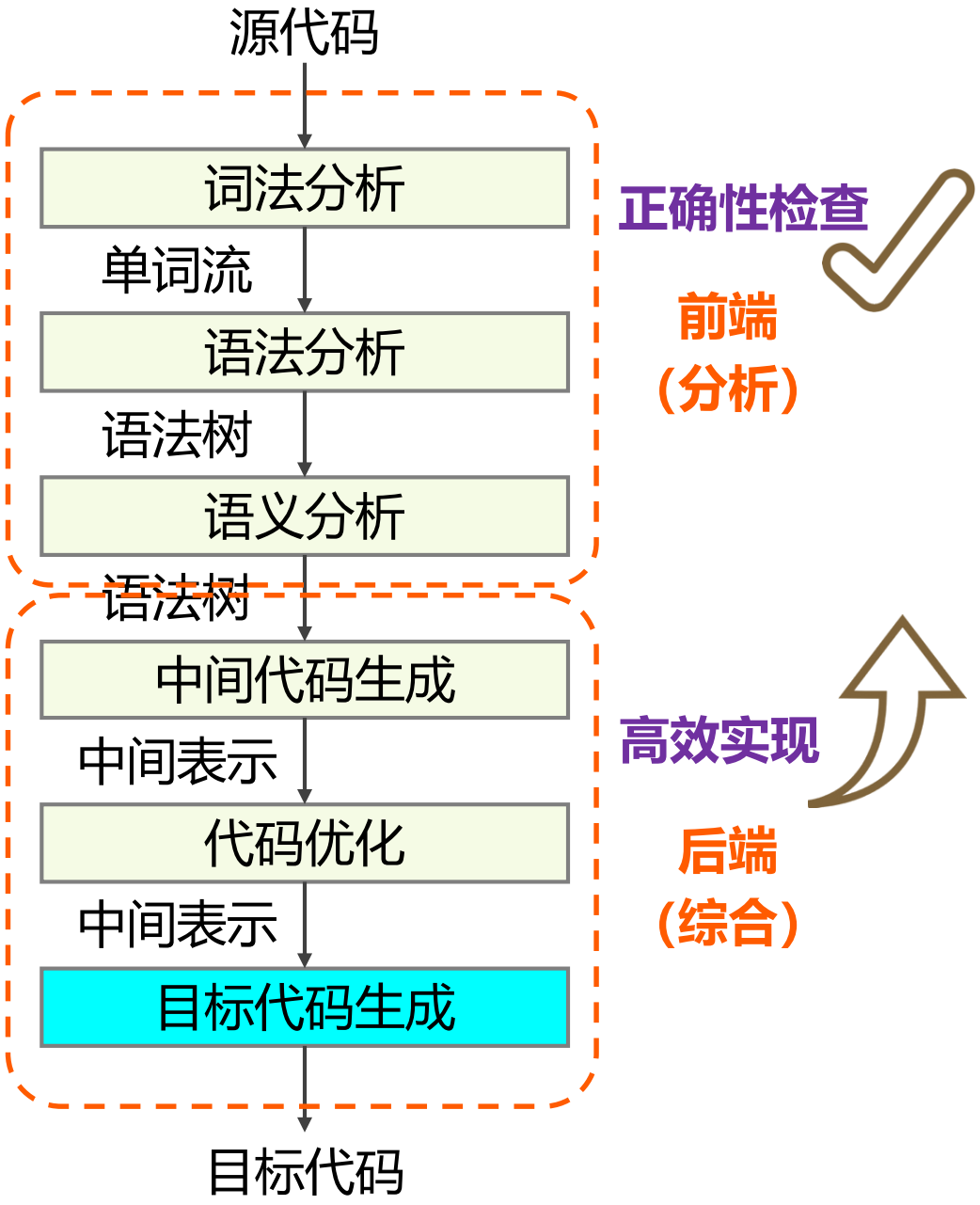
DCS290

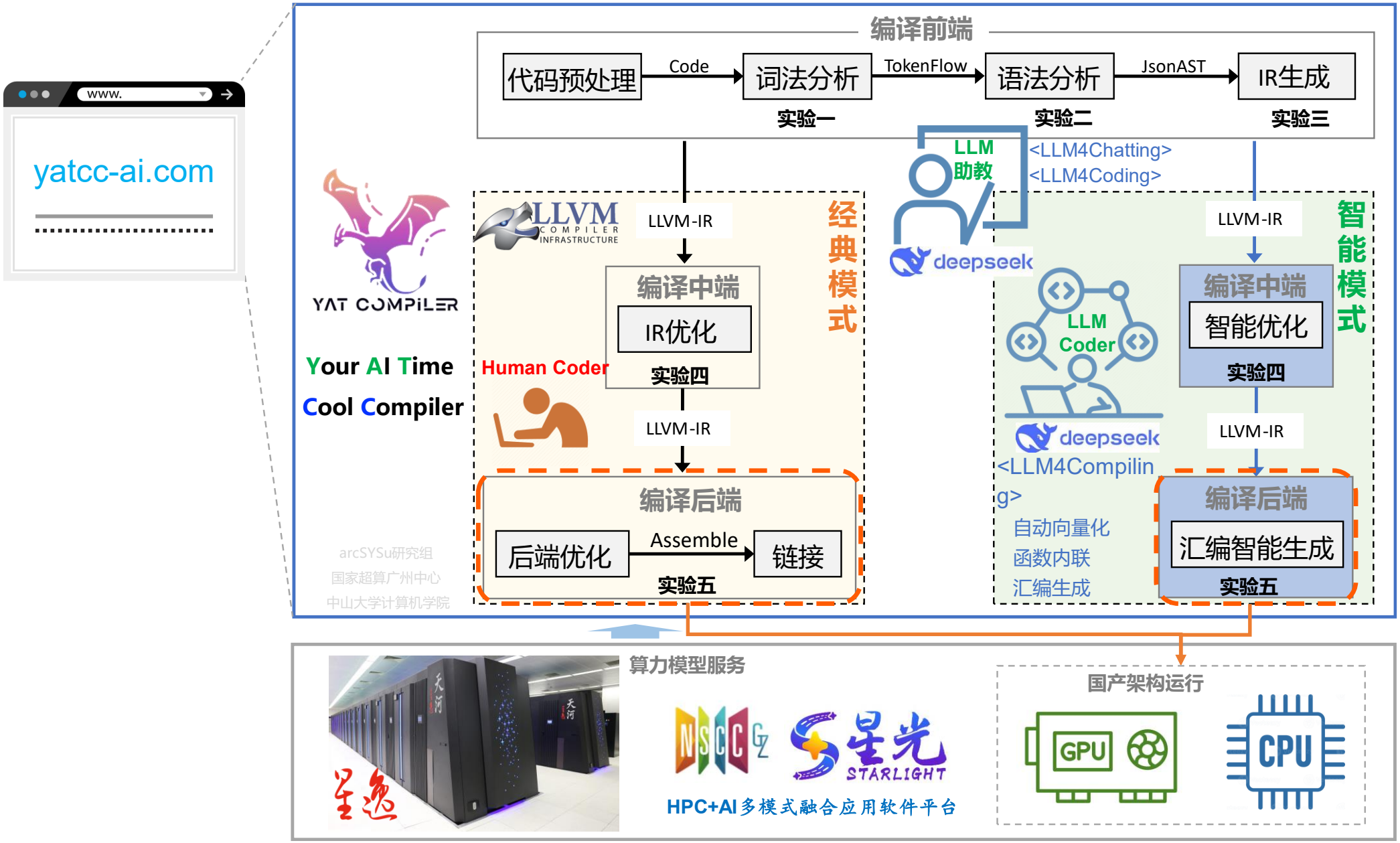
Compilation Principle

编 译 原 理

第八章 目标代码生成

郑馥丹
zhengfd5@mail.sysu.edu.cn


源代码
词法分析
单词流
语法分析
语法树
语义分析
语法树
中间代码生成
中间表示
代码优化
中间表示
目标代码生成
目标代码
正确性检查
前端
（分析）
高效实现
后端
（综合）


www.
yatcc-ai.com
编译前端
代码预处理
Code
词法分析
TokenFlow
语法分析
JsonAST
IR生成
实验一
实验二
实验三
LLM
助教
<LLM4Chatting>
<LLM4Coding>
deepseek
YAT COMPILER
Your AI Time
Cool Compiler
LLVM
COMPILER
INFRASTRUCTURE
LLVM-IR
经典模式
编译中端
IR优化
实验四
Human Coder
LLVM-IR
编译后端
后端优化
Assemble
链接
实验五
arcSYSu研究组
国家超算广州中心
中山大学计算机学院
LLVM-IR
智能模式
编译中端
智能优化
实验四
LLM
Coder
deepseek
LLVM-IR
<LLM4Compiling>
自动向量化
函数内联
汇编生成
编译后端
汇编智能生成
实验五
算力模型服务
NSCC GZ
星光
STARLIGHT
HPC+AI多模式融合应用软件平台
国产架构运行
GPU
CPU

CONTENTS

目 录

1. 代码生成器

- 一个基于基本块的代码生成器
  - 它的输入是四元式中间代码，输出是M机器的汇编代码
  - 着重讨论在基本块内**如何充分利用寄存器**

2. 寄存器分配原则

- 总原则：
  - 在指令的执行代价中，**寄存器的代价是最小的**，因此总是**希望将尽可能多的运算对象放在寄存器中**；
  - 由于任何一个计算机模型中的**寄存器个数都是有限的**，因而需要根据一些原则，对寄存器进行分配。

2. 寄存器分配原则

- 基于基本块的寄存器分配的一般原则：
  - 当生成某变量的目标代码时，**尽量让变量的值或中间结果保留在寄存器中**，直到寄存器不够分配为止，这样引用变量值时可减少对内存的存取次数，提高运行速度
  - 进入基本块时所有寄存器是空闲的，**当到基本块出口时，应将变量的有用值存回内存，释放所有寄存器**
  - 在基本块内，**后面不再被引用的变量占用的寄存器应尽早释放**，以提高寄存器的利用效率

3. 待用信息链表法

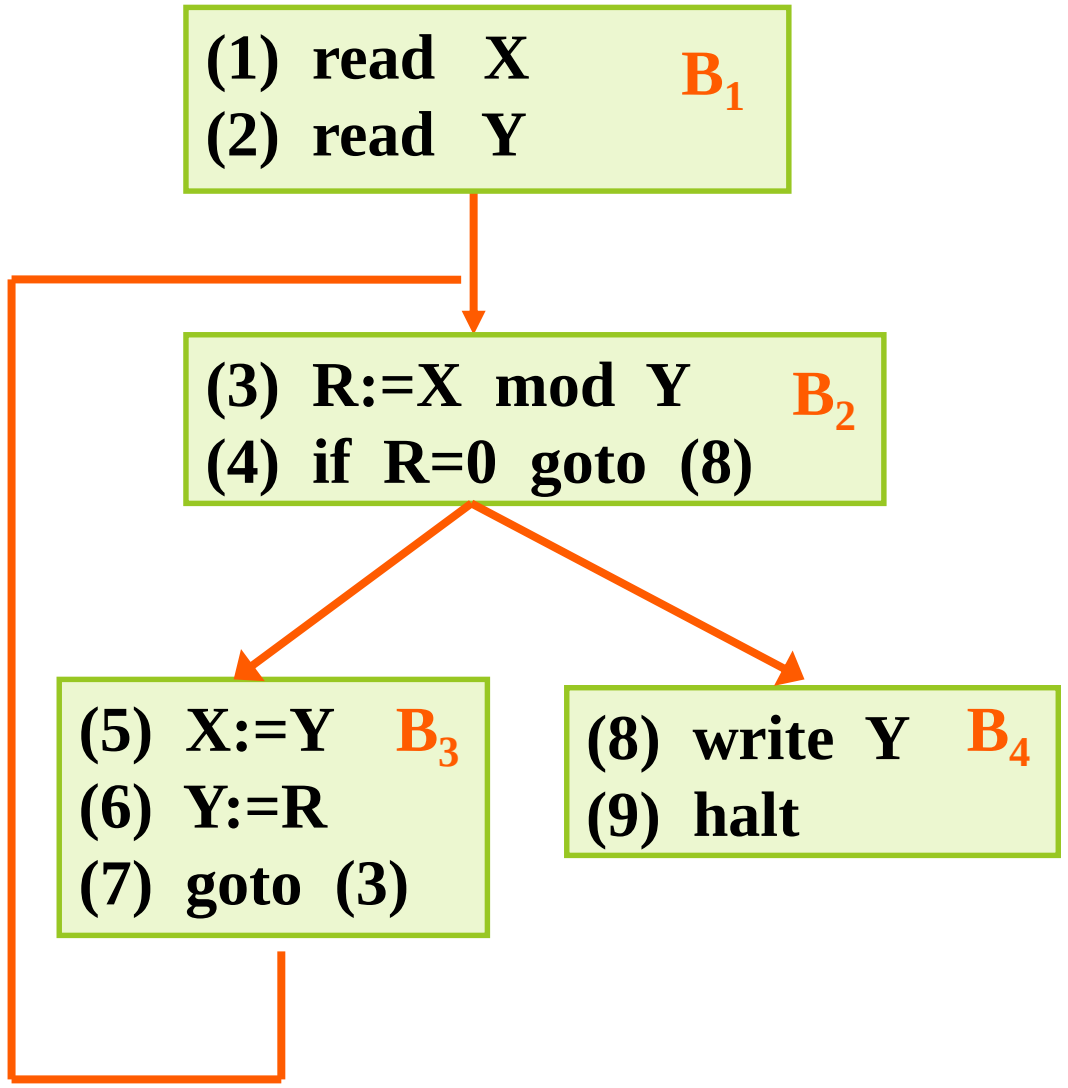
- 为了把在基本块内还要被引用的变量值尽可能保存在寄存器中，不再被引用的变量所占的寄存器尽早释放，在翻译四元式 i：A:=B op C时，必须知道变量A，B，C在基本块内其后的引用情况，即所谓**变量的待用信息**。

3. 待用信息链表法

- 基本定义（回顾）：
  - 定值、引用、活跃
    - ✓ 在形如 i: A:=B+C 的代码中，出现在“:=”左边和右边的变量，分别被称为**对变量的定值和引用**（对变量A定值，对变量B和C引用），i被称为变量的**定值点或引用点**。若定值变量的值在**i之后的代码序列中被引用，**则称**变量在i点是活跃的**。
  - 待用信息
    - ✓ 在基本块中，变量A在四元式i中被定值，在i后面的四元式j中引用A值，且从i到j之间没有其他对A的定值点，称j是i中对变量A的**待用信息**（下次引用信息）。所有这样的待用信息j_k（k=1,2,…）构成**待用信息链**。

3. 待用信息链表法

- 基本定义（回顾）：
  - 在基本块B2内：R在(3)处定值，在(4)处被引用，所以(4)是(3)中R的待用信息
  - 流程B3->B2：X在(5)处被定值，在(3)处被引用，所以X在(5)处是活跃的



3. 待用信息链表法

- **只在基本块内考虑待用信息**，一个变量在基本块的后继中是否被引用，可从活跃变量信息得知（出基本块后，变量是否活跃需要进行全局的数据流分析才能确定）

3. 待用信息链表法

- 基本块内求待用信息的算法
  - 假设**符号表**中含有变量的**待用信息**和**活跃信息**栏；**四元式表**上也有关于结果变量、左右操作数变量的**待用**和**活跃信息**栏。

3. 待用信息链表法

- 例：有如下基本块

  (1) T:=A-B

  (2) U:=A-C

  (3) V:=T+U

  (4) D:=V+U

  – A, B, C, D是用户变量（在基本块出口活跃）

  – T, U, V是临时变量（在基本块出口不活跃）

  – 用“F”表示“非待用”、“非活跃”，“L”表示“活跃”，用一个二元组表示待用和活跃情况，第一元表示待不待用，第二元表示活不活跃。如(F, L)表示(非待用，活跃)，(4, L)表示在第4条四元式被用，活跃。

  – 则该基本块的待用信息及活跃信息的计算结果如下：

附加在四元式上的待用及活跃信息

序号	四元式	结 果	左操作数	右操作数
1	T:=A-B			
2	U:=A-C			
3	V:=T+U			
4	D:=V+U			

符号表中的待用及活跃信息

变量名	初值	(4)	(3)	(2)	(1)
A					
B					
C					
D					
T					
U					
V					

3. 待用信息链表法

- 基本块内求待用信息的算法

① 把基本块中各变量在符号表的登记项中的待用信息栏置为“非待用”，活跃信息栏按变量在基本块出口是否活跃而置为“活跃”或“非活跃”（**假定用户变量都是活跃的，临时变量都是非活跃的**）。

附加在四元式上的待用及活跃信息

序号	四元式	结 果	左操作数	右操作数
1	T:=A-B			
2	U:=A-C			
3	V:=T+U			
4	D:=V+U			

符号表中的待用及活跃信息

变量名	初值	(4)	(3)	(2)	(1)
A	(F,L)				
B	(F,L)				
C	(F,L)				
D	(F,L)				
T	(F,F)				
U	(F,F)				
V	(F,F)				

3. 待用信息链表法

- 基本块内求待用信息的算法

② 从基本块出口的四元式开始**由后向前**依次处理各个四元式 i: A:=B op C ，直到处理完为止：

I. 把符号表中变量A的待用信息和活跃信息附加到四元式i上。

II. 把符号表中变量A的待用信息栏和活跃信息栏分别置为“非待用”和“非活跃”。**（由于在i中对A的定值只能在i以后引用，而对i以前的四元式来说A是非活跃和非待用的。）**

附加在四元式上的待用及活跃信息

序号	四元式	结 果	左操作数	右操作数
1	T:=A-B			
2	U:=A-C			
3	V:=T+U			
4	D:=V+U	(F,L)		

符号表中的待用及活跃信息

变量名	初值	(4)	(3)	(2)	(1)
A	(F,L)				
B	(F,L)				
C	(F,L)				
D	(F,L)	(F,F)			
T	(F,F)				
U	(F,F)				
V	(F,F)				

3. 待用信息链表法

- 基本块内求待用信息的算法

② 从基本块出口的四元式开始**由后向前**依次处理各个四元式 i: A:=B op C ，直到处理完为止：

III. 把符号表中变量B和C的待用信息和活跃信息附加到四元式i上。

IV. 把符号表中变量B和C的待用信息栏置为“i”，活跃信息栏置为“活跃”。

注意：i，ii，iii，iv的次序不能颠倒，因为B和C也可能是A（对同一个变量定值或引用）

附加在四元式上的待用及活跃信息

序号	四元式	结 果	左操作数	右操作数
1	T:=A-B	(3,L)	(2,L)	(F,L)
2	U:=A-C	(3,L)	(F,L)	(F,L)
3	V:=T+U	(4,L)	(F,F)	(4,L)
4	D:=V+U	(F,L)	(F,F)	(F,F)

符号表中的待用及活跃信息

变量名	初值	(4)	(3)	(2)	(1)
A	(F,L)			(2,L)	(1,L)
B	(F,L)				(1,L)
C	(F,L)			(2,L)	
D	(F,L)	(F,F)			
T	(F,F)		(3,L)		(F,F)
U	(F,F)	(4,L)	(3,L)	(F,F)	
V	(F,F)	(4,L)	(F,F)		

4. 代码生成算法

(1) 寄存器描述数组RVALUE

– 为了掌握各寄存器的使用情况，用数组RVALUE记录每个寄存器的当前情况：

✓ RVALUE[Ri]={A}：表示Ri里的值是变量A的值，或说A独占Ri。

✓ RVALUE[Ri]={ }：表示Ri是空闲的。

✓ RVALUE[Ri]={B，C}：表示Ri里的值是变量B和C的值，或说B，C共占Ri。

在复写（B:＝C）时存在多个变量共占一个寄存器。

4. 代码生成算法

(2) 变量地址描述数组AVALUE

– 生成目标代码要用到变量的地址，它可能是某寄存器Ri，也可能是内存单元（仍用变量名表示）。若变量值同时在寄存器Ri和内存，则取Ri为其地址。

✓ AVALUE[A]={Ri}：表示变量A的值在Ri中，不在内存。

✓ AVALUE[B]={Ri, B}：表示变量B的值在Ri中，又在内存。

✓ AVALUE[C]={C}：表示变量C的值只在内存。

4. 代码生成算法

(3) 寄存器分配函数GETREG

– 以形如 i: A:=B op C 的四元式为输入，返回一个寄存器R，用以存放A的结果值，其算法为：

① 如果B独占Ri，且B与A是同一标识符或B值不再引用，则选Ri为R并转④

② 否则，如果有空闲Ri，则选Ri为R并转④

③ 否则，释放一个Ri作为R，最好占用**Ri(要被释放的寄存器)**的变量值同时在内存中，或在基本块中引用的位置最远。对Ri中**不是A的变量**M，且其值不在内存，则做如下处理：

- 生成目标代码**ST Ri，M**，把Ri的值存回变量M在内存的地址
- 修改变量地址描述：如**M不是B**则AVALUE[M]={M}，否则AVALUE[M]={Ri，M}。
- 修改寄存器描述：删除RVALUE{Ri}中的M。

④ 给出R，返回

通过GETREG（i: A:=B op C ）返回的寄存器R用于进行B op C的运算，并把结果（A值）保存在R中。

5. 基本块的代码生成算法

- 开始时所有寄存器都空闲，顺序取出四元式i: A:=B op C，按下述算法生成目标代码。
  - ① 分配寄存器 R=GETREG(i: A:=B op C);
  - ② 利用地址描述数组AVALUE[B]和AVALUE[C]，确定出B和C现行值的存放位置B′和C′，如果其现行值在寄存器中，则把寄存器取作B′和C′；
  - ③ 如果B′≠R则生成目标代码**LD R，B′；op R，C′**；否则生成**op R，C′**，如果B′或C′为R，则删除AVALUE[B]或AVALUE[C]中的R。
  - ④ 令AVALUE[A]={R}，RVALUE[R]={A}。
  - ⑤ 如果B和C的现行值是“非引用”和“非活跃”，且其值在某个R_k中，则删除RVALUE[R_k]中的B或C，删除AVALUE[B]和AVALUE[C]中的R_k。
- **到达基本块出口时，若有活跃变量占用寄存器且其值不在内存，则生成存数指令ST，保存其值并释放寄存器。**

例：假设只有R_1和R_2两个寄存器

利用基本块代码生成算法生成目标代码

四元式	选取R ⇦	目标代码 ⇦	RVALUE	AVALUE
(1)T:=A-B	空闲的R_1	LD R_1, A SUB R_1, B	R_1含有T	T在R_1中
(2)U:=A-C	空闲的R_2	LD R_2, A SUB R_2, C	R_1含有T R_2含有U	T在R_1中 U在R_2中
(3)V:=T+U	T独占的R_1	ADD R_1, R_2	R_1含有V R_2含有U	V在R_1中 U在R_2中
(4)D:=V+U	V独占的R_1	ADD R_1, R_2	R_1含有D	D在R_1中
		ST R_1, D		

在基本块出口处，D占用R_1，且D值不在内存，因此生成ST R_1, D 指令保存D值，然后释放R_1。

这里，把右操作数占用的R自动释放掉了

例：假设只有R_1和R_2两个寄存器
利用基本块代码生成算法生成目标代码

四元式	选取R	目标代码	RVALUE	AVALUE
T1:=A+B	空闲的R1	LD R1,A ADD R1,B	R1中有T1	T1在R1中
T2:=C-T1	空闲的R2	LD R2,C SUB R2,R1	R2中有T2	T2在R2中
T3:=D*E	空闲的R1	LD R1, D MUL R1,E	R1中有T3 R2中有T2	T3在R1中 T2在R2中

T4:=F+G	释放R2	ST R2, T2
		LD R2, F
		ADD R2,G
			R1中有T3
			R2中有T4
				T3在R1中
				T4在R2中

T5:=T3-T4	T3独占的R1	SUB R1,R2	R1中有T5	T5在R1中
W:=T2/T5	空闲的R2	LD R2,T2 DIV R2, R1	R2中有W	W在R2中
	释放R2	ST R2,W		

右操作数如果在以后都不会再用到，则将右操作数所占的寄存器自动释放掉