Principles of Compiler Construction

（目标）代码生成

真正的挑战在于生成高质量代码--时间&空间

优化效果：源代码 ＞ 中间代码 ＞ 目标代码

Prof. Wen-jun LI

School of Computer Science and Engineering

lnslwj@mail.sysu.edu.cn

Lecture 11. Code Generation

1. Introduction
2. Abstraction of Target Machines
3. Implementation of Procedures
4. Optimization of Basic Blocks
5. A Simple Code Generator
6. Peephole Optimization

1. Introduction

- Code generators focus on
  1. Instruction selection
     - Choose appropriate target-machine instructions to implement IR statements.
  2. Register allocation and assignment
     - Make full use of registers, the fastest computational unit.
  3. Instruction ordering
     - Decide the order in which the execution of instructions is scheduled.
     - Also called **instruction scheduling**.

1

Instruction Selection

- Both the speed and the size cost of the generated code should be considered.
  - Trade-off and consequence.
- Example 1: **a = a + 1**
  - ```
    LD    R0,  a
    ADD   R0,  R0,  #1
    ST    a,   R0
    ```
  - `INC   a` 更高效的做法
- Example 2: set register R0 to 0
  - `LD    R0,  #0`
  - `XOR   R0,  R0` 更高效的做法
    ```

Register Allocation and Assignment

- Include two subproblems
  - Register allocation
    - Select the set of variables that will reside in registers at each point in the program.
  - Register assignment
    - Pick the specific register that a variable will reside in.
- A difficult problem in code generation.

3

Instruction Scheduling

- Important for modern pipelined processors
  - RISC: Reduced Instruction Set Computer
    - RISC vs. CISC
  - Data Hazards (抛锚 **stalls** in a pipeline): attempt to use a datum before it becomes available in register.
    - A value written to a register will not be available in its following cycles.
    - The following cycles should be utilized by instructions that does not depend on the value
      - Or they will be wasted.
    - Optimization: clustering instructions with no dependencies together, while preserving original semantics.

Instruction Scheduling (cont')

- An example: **(a + b) + c** on SGI's <u>MIPS 2000</u>

```
LD    R1, a
LD    R2, b
NOP
ADDi  R1, R1, R2
LD    R2, c
NOP
ADDi  R1, R1, R2
```

IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

- Optimal generated code

```
LD    R1, a
LD    R2, b
LD    R3, c
ADDi  R1, R1, R2
NOP
ADDi  R1, R1, R3
```

2. Abstraction of Target Machines

- Instruction set
- Addressing modes
- Program and instruction costs

Instruction Set

- Supports the following instructions
  - LD dst, addr // load
  - ST x, Ri // store
  - OP dst, src1, src2 // ADD, SUB, ...
    OP dst, src1 // unary operation
  - BR L // branch
  - Bcond r, L // conditional branch

Addressing Modes

- Addressing in instructions
  - #C // immediate constant, cost = 1
  - x // absolute, cost = 1
  - *x // indirect memory, cost = 1
  - R // direct register, cost = 0
  - *R // indirect register, cost = 0
  - a(Ri) // direct indexed, cost = 1
    - **a** is a variable or a constant
    - LD R1, a(R2)
      - R1 = contents(a + contents(R2))
    - LD R1, 100(R2)
      - R1 = contents(100 + contents(R2))
  - *a(Ri) // indirect indexed, cost = 1
    - LD R1, *100(R2)
      - R1 = contents(contents(100 + contents(R2)))

Program and Instruction Costs

- Cost of an instruction
  - = 1 + operand_addr_cost
- Cost of a program
  - = total of instruction costs
- Examples
  - `LD R0, R1 // cost = 1`
  - `LD R0, x // cost = 2`
  - `LD R1, *100(R2) // cost = 2`

3. Implementation of Procedures

- Static Allocation
- Stack Allocation

procedures
英 [prəˈsiːdʒəz] 美 [prəˈsiːdʒərz]
n. 程序；规程；外科手术（procedure
的复数）

Static Allocation

- Implementation of **call** ***callee***
  - ...
    ST *callee.staticArea*, #*here* + 20 // return address
    BR *callee.codeArea* // jump to Callee
    ...

跳转指令之后下一条指令的地址

- Implementation of **return**
  - ...
    BR **callee.staticArea* // return to Caller

Assume that the return address is saved at the **beginning** of the activation record. And **20 = 5 words * 4 bytes/word**

An Example: Three-Address Code

// code for **c**

$action_1$

call p

$action_2$

halt

// code for **p**

$action_3$

return

An Example: Implementation

```
// code for c
100    action_1
120    ST  364,  #140        // return address
132    BR  200               // call p
140    action_2
160    HALT                  // return to operating system
       ...
// code for p
200    action_3
220    BR  *364              // return to its caller
       ....
// activation record for c
300                          // return address to Caller
304                          // local data for c
       ....
// activation record for p
364    140                   // return address to Caller
368                          // local data for p
```

Stack Allocation 需要的时候才分配

- Implementation of initialization
  - ```
    LD    SP, #stackStart                // initialize the stack
    ...                                  // code for main()
    HALT                                 // terminate
    ```
    寄存器

- Implementation of **call** ***callee***
  - ```
    ...
    ADD   SP,  SP,  #caller.recordSize   // push an AR
    ST    *SP, #here + 16                // return address
    BR    callee.codeArea                // jump to Callee
    SUB   SP,  SP,  #caller.recordSize   // pop the AR
    ...
    ```

DBv2 must assume that addressing **0(SP)** has no additional cost

- Implementation of **return**
  - ```
    ...
    BR    *0(SP)                         // return to Caller
    ```
    ```

An Example: Three-Address Code

```
// code for m
action1
call q
action2
halt

// code for p
action3
return
```

```
// code for q
action4
call p
action5
call q
action6
call q
return
```

An Example: Implementation

100	LD	SP,	#600	
108	action_1			
128	ADD	SP,	SP,	#msize
136	ST	*SP,		#152
144	BR	300		
152	SUB	SP,	SP,	#msize
160	action_2			
180	HALT			
	...			
200	action_3			
220	BR	*0(SP)		
	....			
300	action_4			
320	ADD	SP,	SP,	#qsize
328	ST	*SP,		#344
336	BR	200		
344	SUB	SP,	SP,	#qsize
352	action_5			
372	ADD	SP,	SP,	#qsize
380	ST	*SP,		#396
388	BR	300		
396	SUB	SP,	SP,	#qsize
404	action_6			
424	ADD	SP,	SP,	#qsize
432	ST	*SP,		#448
440	BR	300		
448	SUB	SP,	SP,	#qsize
456	BR	*0(SP)		
	...			
600				

code for **p**

code for **q**

blue for call sequence

stack

4. Optimization of Basic Blocks

- Approach

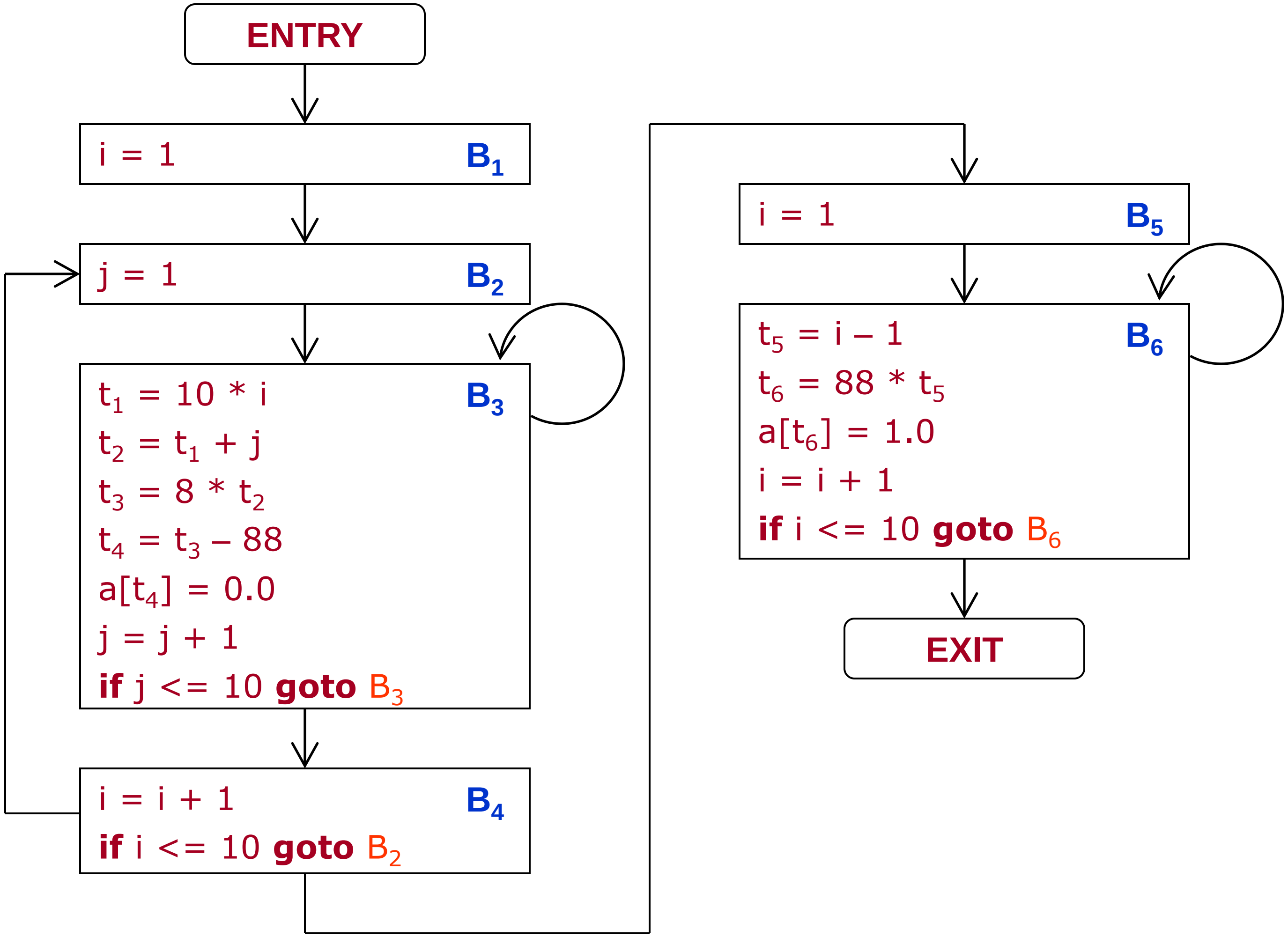
Basic Blocks

- Single entry and single exit
- Example

(1) $i = 1$	(10) $i = i + 1$
(2) $j = 1$	(11) **if** $i <= 10$ **goto** 2
(3) $t_1 = 10 * i$	(12) $i = 1$
(4) $t_2 = t_1 + j$	(13) $t_5 = i - 1$
(5) $t_3 = 8 * t_2$	(14) $t_6 = 88 * t_5$
(6) $t_4 = t_3 - 88$	(15) $a[t_6] = 1.0$
(7) $a[t_4] = 0.0$	(16) $i = i + 1$
(8) $j = j + 1$	(17) **if** $i <= 10$ **goto** 13
(9) **if** $j <= 10$ **goto** 3	

Flow Graphs

流程图



Construction of Flow Graphs

- Partition three-address instructions into basic blocks
  - **Leader**: the 1^{st} instruction in a basic block.
  - Algorithm to find all leaders
    1. The 1^{st} three-address instruction
    2. Target of a conditional or unconditional jump　跳转指令目标以及后头的那一条指令
    3. Instruction immediately follows a conditional or unconditional jump

3种 切割结果是上一页

Liveness and Next-Use Information

后续优化时使用

- Calculate liveness and next-use info in a basic block: backward scanning
  - Initialize: for each variable v,
    - v.nextUse = **none**;
      v.liveness = v is temporary ? **false** : **true**;
  - For each i: x = y + z
    - Attach information of x, y and z to instruction i;
    - x.liveness = **false**;基本块的出口是否还有再使用
      x.nextUse = **none**;只关注在当前基本块内的情况
    - y.liveness = z.liveness = **true**;
      y.nextUse = z.nextUse = i;

Conservative.
Global data-flow analysis in practice

An Example

- Information are stored at the entry of each variables in the symbol table

说明赋值必须保留

define use

(1) t = a – b
(2) u = a – c
(3) v = t + u
(4) d = v + u

nextuse liveness ?

(1) $t^{(3),T} = a^{(2),T} - b^{-,T}$
(2) $u^{(3),T} = a^{-,T} - c^{-,T}$
(3) $v^{(4),T} = t^{-,F} + u^{(4),T}$
(4) $d^{-,T} = v^{-,F} + u^{-,F}$

自底向上分析

Var.	Next-Use					Liveness				
	Init	(4)	(3)	(2)	(1)	Init	(4)	(3)	(2)	(1)
a	–			(2)	(1)	T			T	T
b	–				(1)	T				T
c	–			(2)		T			T	
d	–	–				T	F			
t	–		(3)		–	F		T		F
u	–	(4)	(3)	–		F	T	T	F	
v	–	(4)	–			F	T	F		

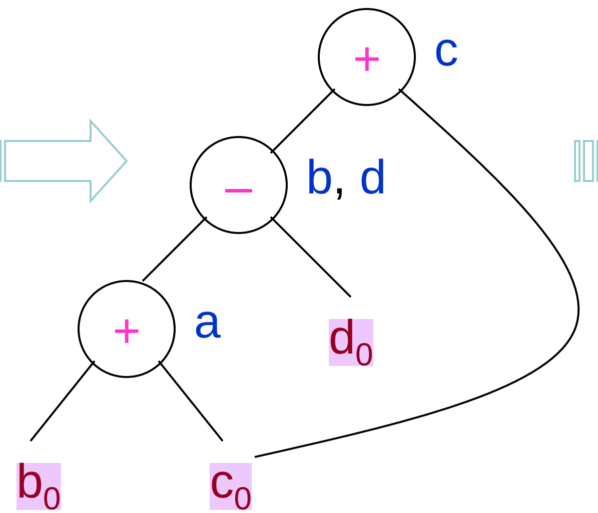
依赖数据流分析技术

Optimization Based on DAGs

- Perform several code-improving transformations
  - Elimination of local common subexpressions
  - Elimination of dead code
  - Reordering of independent statements
  - Application of algebraic laws

Local Common Subexpressions

```
a = b + c
b = a - d
c = b + c
d = a - d
```



来自外部

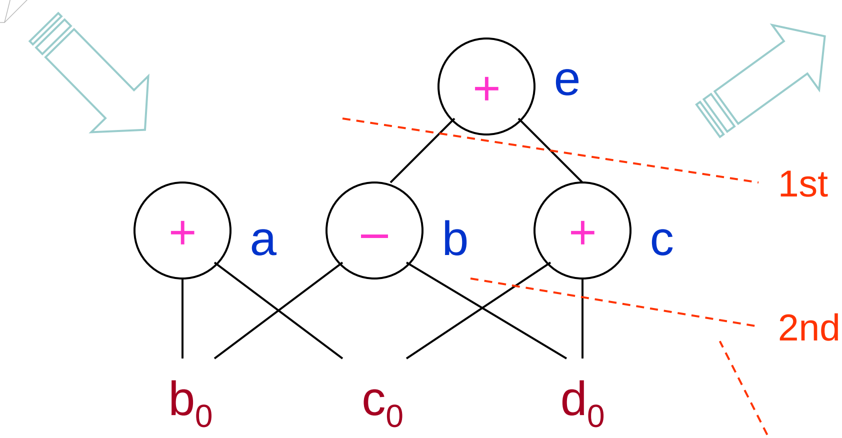
```
a = b + c
d = a - d
c = d + c
b = d
```

Only if b is live on exit

没有再使用就删去该语句

Dead Code

```
a = b + c
b = b - d
c = c + d
e = b + c
```




```
a = b + c
b = b - d
```

Suppose only **a** and **b** are live on exit

Algebraic Identities

部分可以直接替换的等价

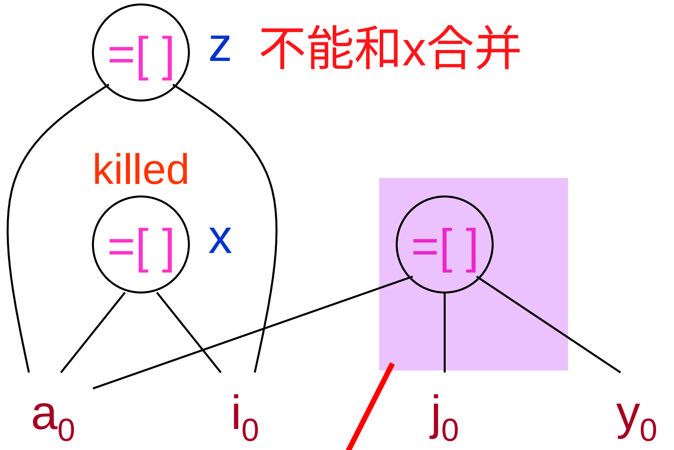
- Arithmetic identities
  - $x + 0 = 0 + x = x$
  - $x * 1 = 1 * x = x$
  - $x – 0 = x$
  - $x / 1 = x$
  - …
- Local reduction in strength
  - $2 * x \Rightarrow x + x$
  - $x / 2 \Rightarrow x * 0.5$
  - …

换成开销更小的等价语句

- Constant folding
  - $2 * 3.14 \Rightarrow 6.28$
  - …

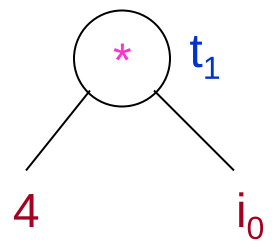
Array References

```
x = a[i]
a[j] = y
z = a[i]
```



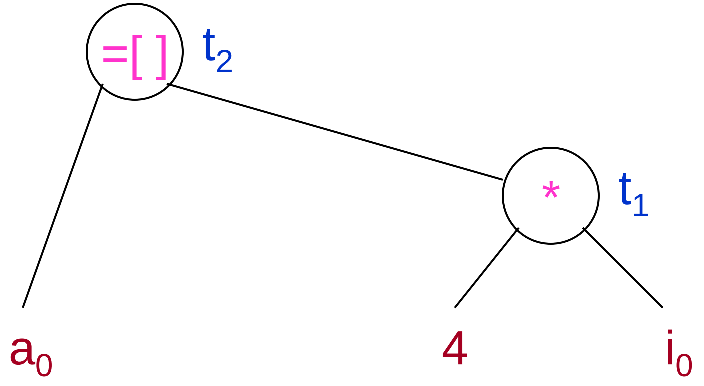
An Example:
Construction a DAG from a BB (1)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



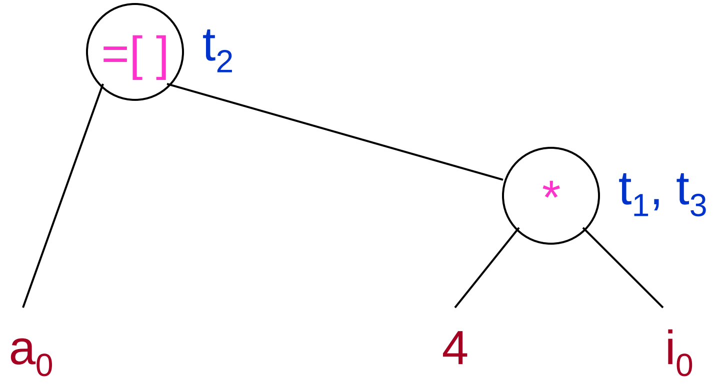
An Example: Construction a DAG from a BB (2)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



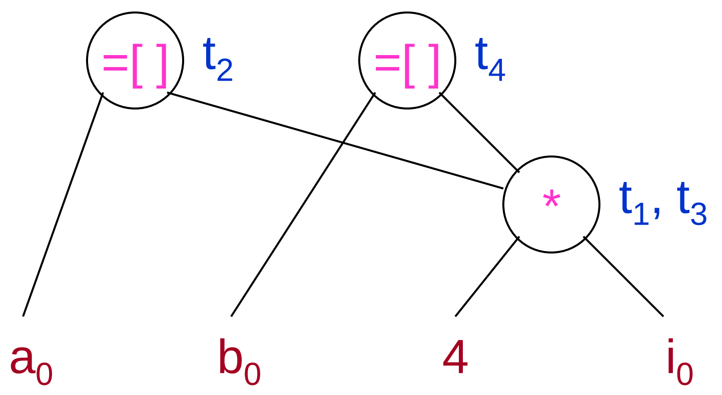
An Example: Construction a DAG from a BB (3)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



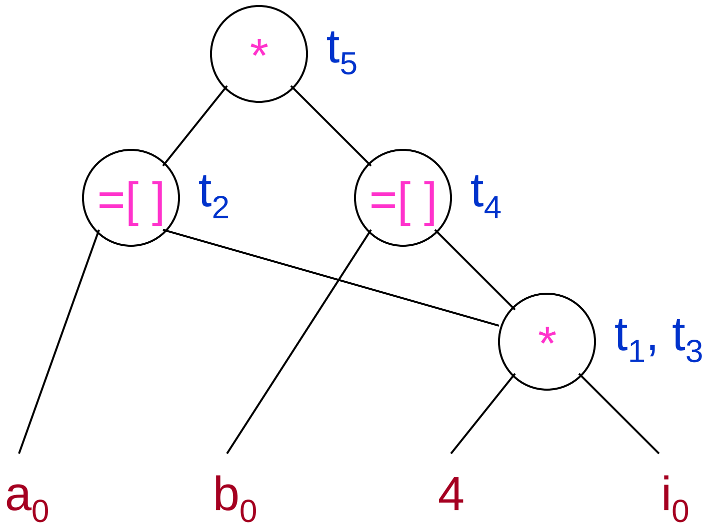
An Example: Construction a DAG from a BB (4)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



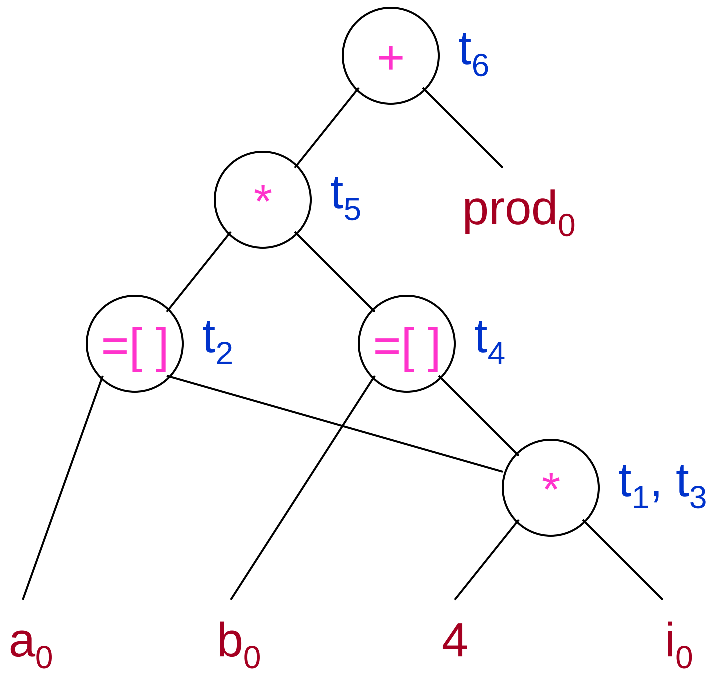
An Example: Construction a DAG from a BB (5)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



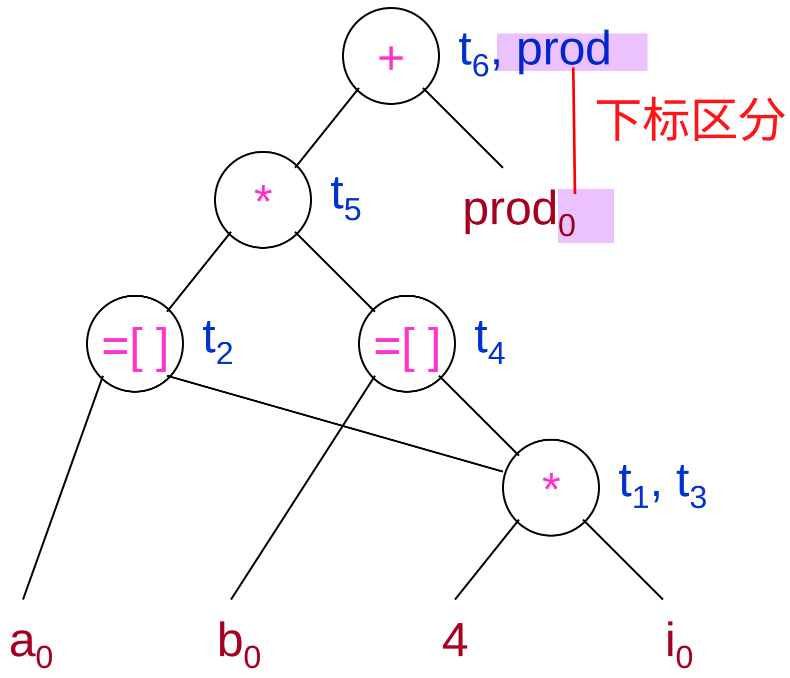
An Example:
Construction a DAG from a BB (6)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



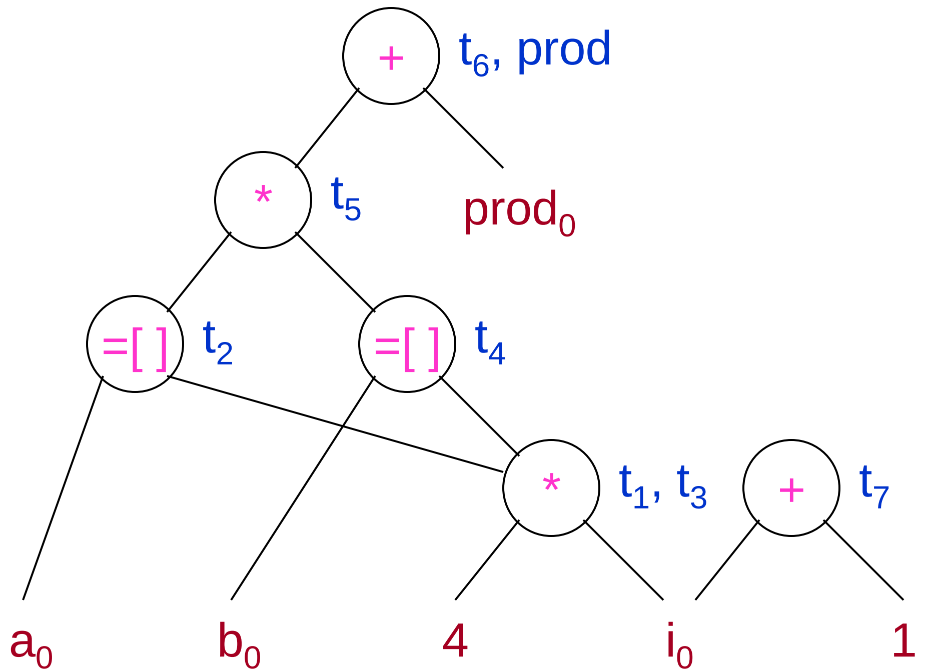
An Example:
Construction a DAG from a BB (7)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



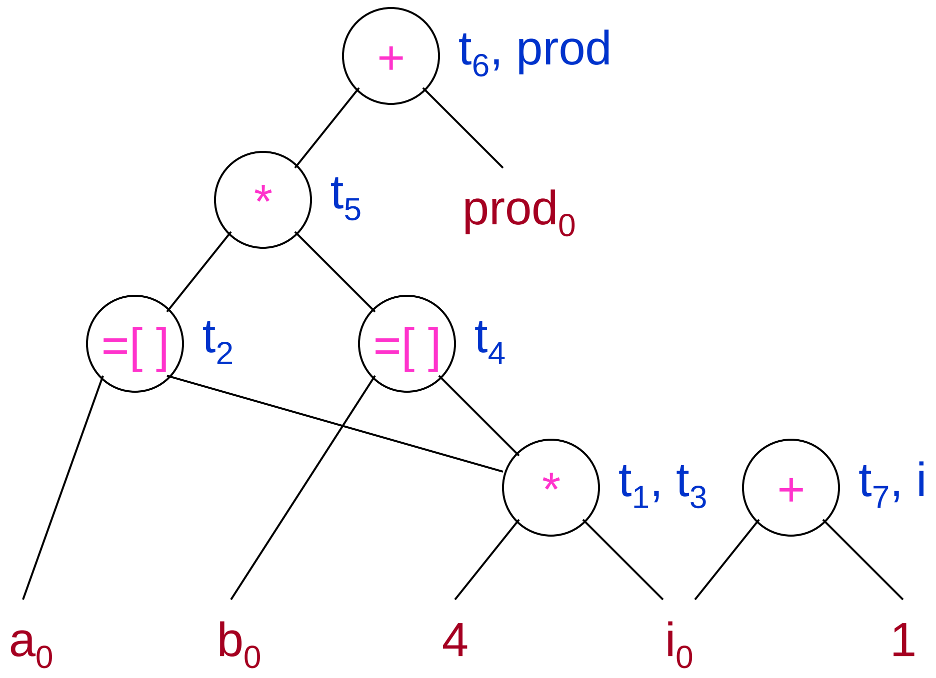
An Example:
Construction a DAG from a BB (8)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



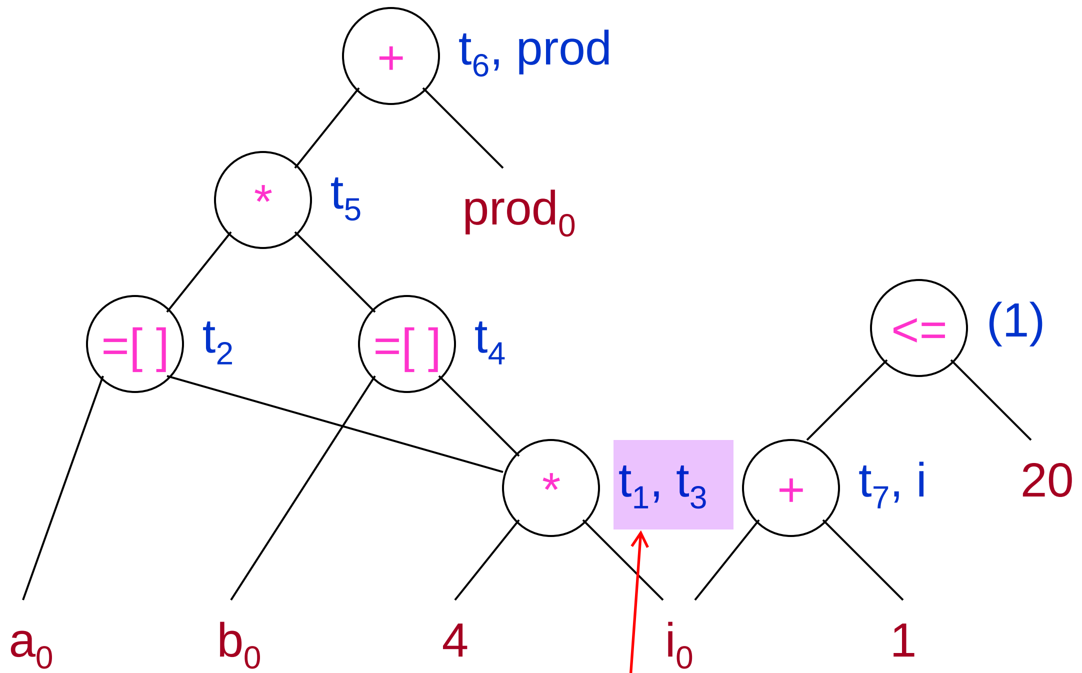
An Example: Construction a DAG from a BB (9)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



An Example:
Construction a DAG from a BB (10)

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = t5 + prod
prod = t6
t7 = i + 1
i = t7
if i <= 20 goto (1)
```



不创建新的变量，直接附着t1

能不能死掉就看变量还有没有用

5. A Simple Code Generator

- What does "simple" mean?
  - Generate code for a single basic block.
- Motivation
  - Make full use of registers to the best advantage.

Storage	Access Cycles	Capability
Registers	1	256-8000 Bytes
Cache	3	256 KB - 1 MB
Main memory	20-100	32 MB - 1 GB
Disk	0.5-5 Mega	10 GB - 1 TB

Compiler

OS and Hardware

Programmer

编程器

表示数据交流

Register and Address Descriptors

- Register descriptor 记录寄存器内记录的变量内容
  - **RegDesc: Register** $\rightarrow 2^{\textbf{Variable}}$
- Address descriptor 记录变量的地址信息
  - **AddrDesc: Variable** $\rightarrow 2^{\textbf{Location}}$
  - **Location = Register** $\cup$ **Memory**

Code-Generation Algorithm

- Based on function getReg($x = y \text{ op } z$)
  - Select registers for variables in three-address instruction $x = y \text{ op } z$.
    - That is R_x, R_y and R_z for x, y and z respectively.
  - Make decisions based on the register descriptor and the address descriptor.

Code-Generation Algorithm (cont')

- Machine instructions for $x = y + z$
  - Call getReg($x = y + z$) to select registers R_x, R_y and R_z for x, y and z.
  - y不再寄存器内 **If** $y \notin regDesc(R_y)$, get some $y' \in addrDesc(y)$ and issue an instruction: LD R_y, y'.
  - **If** $z \notin regDesc(R_z)$, get some $z' \in addrDesc(z)$ and issue an instruction: LD R_z, z'.
  - Issue an instruction: ADD R_x, R_y, R_z.
  - Adjust register and address descriptors.

Code-Generation Algorithm (cont')

- Machine instructions for $x = y$
  - Call getReg($x = y$) to select registers R_x and R_y.
    - getReg() will always choose the same register for both x and y.
  - **If** $y \notin$ regDesc(R_y), get some $y' \in$ addrDesc(y) and issue an instruction: LD R_y, y'.
  - Adjust register descriptor: regDesc(R_y) ∪= {x}

把x并入

spill
英 [spɪl] 美 [spɪl]
v. （使）洒出，泼出，溢出；（东西）倒出，洒落；涌出，蜂拥而出；（光线）透进，倾泻在……上；向（某人）说出秘密；使（某人）从马上（或自行车上）摔下来；（球赛中）掉球；（航海）减少（帆）的受风压力（多指把帆放松）
n. 洒出（量），溢出（量）；（从马、自行车上等的）摔下，跌下；<澳>职位空缺；（点火用的）纸捻，木片；小塞子

Design of getReg()

- Input: x = y + z
- Output: R_x, R_y and R_z
- Algorithm (use R_y as an example)
  - **if** y in some registers, pick R_y in them.
  - **elsif** there is empty registers, pick one as R_y.
  - **elsif** { let R be a candidate and R holds v, **foreach** v check:
    - **if** addrDesc(v) has other location, R is OK.
    - **elsif** v == x **and** x != z, R is OK.
    - **elsif** v is not used later, R is OK.
    - **else** spill, i.e. issue ST v, R.
  - choose the R with minimal spills }.

检查过程

计算每个候选R的“费用”（需要生成的保存指令的个数），选择费用最低的寄存器（或之一）

6. Peephole Optimization

- A simple but effective technique for locally code improvement
  - Examine a sliding window (peephole) of target instructions.
  - Replace instruction sequence within the peephole by a shorter or faster sequence.

Eliminating Redundant Loads and Stores

- Examples
  - LD R0, a
    ST a, R0
    // eliminated

Eliminating Unreachable Code

- Examples
  - ```
          if debug == 1 goto L1
          goto L2
    L1:   print debugging information
    L2:
          if debug != 1 goto L2
          print debugging information
    L2:
    ```
    ```

Flow-of-Control Optimizations

- Examples
  - ```
             goto L1
             ...
      L1:    goto L2
    ```
    ```
             goto L2
             ...
      L1:    goto L2
    ```
  - ```
             if a < b goto L1
             ...
      L1:    goto L2
    ```
    ```
             if a < b goto L2
             ...
      L1:    goto L2
    ```

Recall the translation scheme of flow-of-control statements!

Algebraic Simplification and Reduction in Strength

- Examples
  - x = x + 0
    // eliminated
  - x = x * 1
    // eliminated
  - y = x * 2
    y = x << 1
  - y = x * 4
    y = x << 2

Further Reading

- Dragon Book, 2^{nd} Edition (DBv2)
  - Comprehensive Reading:
    - Section 8.1-8.2 on introduction to code generation and abstraction of target machines.
    - Section 8.3 on implementation of procedures.
    - Section 8.4-8.5 on DAG-based block optimization.
    - Section 8.6 on a simple code generator.
  - Skip Reading:
    - Section 8.7 on peephole optimization.
    - Section 8.8-8.11 on more advanced topics.

Enjoy the Course!