PR003说明

提纲

- 作业时间表
  - PR003，今天~学期末
- 实验任务
  - 中间代码生成
  - 汇编代码生成
  - 代码优化
- 模拟器
- 优化简介

实验任务

- 任务:
  - 从AST进行生成中间代码
  - 由中间代码翻译为RISC-V汇编
  - 代码优化
- 难点:
  - 中间代码的设计
  - AST到中间代码的翻译
  - 中间代码到RISC-V汇编的翻译

实验任务——中间代码生成

- 中间代码的设计
  - 三地址码
  - SSA（不要求）
- 到中间代码的翻译
  - if-else如何翻译？
  - while如何翻译？
  - 变量声明、定义如何翻译？

实验任务——中间代码设计

- 定义函数
- 赋值操作
- 加、减、乘、除
- 取地址操作
  - 取数组元素

实验任务——中间代码设计

- 取某地址的内存单元的内容
- 向某地址的内存单元存内容
- 定义标号
  - if-else、while跳转
- 无条件跳转到标号
- 有条件跳转到标号
- 返回

实验任务——中间代码设计

- 传形参
- 调用函数
- 传实参

实验任务——中间代码生成

- 翻译模式
  - 基本表达式
  - 语句
  - 条件表达式
  - 函数调用
  - 函数参数
  - 数组
- 龙书上有

实验任务——汇编代码生成

- 目标机器：RISC-V 64GC
  - 大概长什么样子
- 有哪些指令
- ABI
  - 调用约定
  - 寄存器使用
- 翻译的细节
  - float怎么表示？全局变量怎么表示？

实验任务——汇编代码生成

- 目标机器： RISC-V 64GC

```c
int func(int a)
{
    int b = 3;
    a = a + b;
    return a;
}

int main(void)
{
    int a = 0;
    int b = a;
    func(a);
    return 0;
}
```

```
    .file   "testcode.c"
    .option nopic
    .attribute arch, "rv64i2p0_m2p0_a2p0_f2p0_d2p0_c2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  1
    .globl  func
    .type   func, @function
func:
    addi    sp,sp,-48
    sd  s0,40(sp)
    addi    s0,sp,48
    mv  a5,a0
    sw  a5,-36(s0)
    li  a5,3
    sw  a5,-20(s0)
    lw  a4,-36(s0)
    lw  a5,-20(s0)
    addw    a5,a4,a5
    sw  a5,-36(s0)
    lw  a5,-36(s0)
    mv  a0,a5
    ld  s0,40(sp)
    addi    sp,sp,48
    jr  ra
    .size   func, .-func
    .align  1
    .globl  main
    .type   main, @function
```

实验任务——汇编代码生成

- 目标机器： RISC-V 64GC

```c
int func(int a)
{
    int b = 3;
    a = a + b;
    return a;
}

int main(void)
{
    int a = 0;
    int b = a;
    func(a);
    return 0;
}
```

```
    .align  1
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-32
    sd  ra,24(sp)
    sd  s0,16(sp)
    addi    s0,sp,32
    sw  zero,-20(s0)
    lw  a5,-20(s0)
    sw  a5,-24(s0)
    lw  a5,-20(s0)
    mv  a0,a5
    call    func
    li  a5,0
    mv  a0,a5
    ld  ra,24(sp)
    ld  s0,16(sp)
    addi    sp,sp,32
    jr  ra
    .size   main, .-main
    .ident  "GCC: (GNU) 10.2.0"
```

实验任务——汇编代码生成

- 目标机器：RISC-V 64GC

```c
float func(float a, float b)
{
    float c = a;
    c = a + b;
    return c;
}

int main(void)
{
    float a = 3.2;
    float b = 4.3;
    int c = 4;
    float d;
    d = func(a, b);
    return 0;
}
```

```
    .attribute stack_align, 16
    .text
    .align  1
    .globl  func
    .type   func, @function
func:
    addi    sp,sp,-48
    sd  s0,40(sp)
    addi    s0,sp,48
    fsw fa0,-36(s0)
    fsw fa1,-40(s0)
    flw fa5,-36(s0)
    fsw fa5,-20(s0)
    flw fa4,-36(s0)
    flw fa5,-40(s0)
    fadd.s  fa5,fa4,fa5
    fsw fa5,-20(s0)
    flw fa5,-20(s0)
    fmv.s   fa0,fa5
    ld  s0,40(sp)
    addi    sp,sp,48
    jr  ra
    .size   func, .-func
```

实验任务——汇编代码生成

- 目标机器：RISC-V 64GC

```c
float func(float a, float b)
{
    float c = a;
    c = a + b;
    return c;
}

int main(void)
{
    float a = 3.2;
    float b = 4.3;
    int c = 4;
    float d;
    d = func(a, b);
    return 0;
}
```

```
    .align  1
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-32
    sd  ra,24(sp)
    sd  s0,16(sp)
    addi    s0,sp,32
    lui a5,%hi(.LC0)
    flw fa5,%lo(.LC0)(a5)
    fsw fa5,-20(s0)
    lui a5,%hi(.LC1)
    flw fa5,%lo(.LC1)(a5)
    fsw fa5,-24(s0)
    li  a5,4
    sw  a5,-28(s0)
    flw fa1,-24(s0)
    flw fa0,-20(s0)
    call    func
    fsw fa0,-32(s0)
    li  a5,0
    mv  a0,a5
    ld  ra,24(sp)
    ld  s0,16(sp)
    addi    sp,sp,32
    jr  ra
    .size   main, .-main
    .section    .rodata
    .align  2
.LC0:
    .word   1078774989
    .align  2
.LC1:
    .word   1082759578
    .ident  "GCC: (GNU) 10.2.0"
```

实验任务——汇编代码生成

- 目标机器：RISC-V 64GC

```
39  int aa = 0;
40  const int bb = 2;
41  int a_array[5];
42  int b_array[5] = {1, 2, 3, 4, 5};
43  int c_array[5] = {1, 2, 3};
44  float fa = 1.2;
45  double da = 2.3;
46
47  float func(float a, float b)
48  {
49      float c = a;
50      c = a + b;
51      return c;
52  }
53
54  int main(void)
55  {
56      float a = 3.2;
57      float b = 4.3;
58      int c = 4;
59      float d;
60      d = func(a, b);
61      return 0;
62  }
```

```
 5      .attribute stack_align, 16
 6      .text
 7      .globl  aa
 8      .section    .sbss,"aw",@nobits
 9      .align  2
10      .type   aa, @object
11      .size   aa, 4
12  aa:
13      .zero   4
14      .globl  bb
15      .section    .srodata,"a"
16      .align  2
17      .type   bb, @object
18      .size   bb, 4
19  bb:
20      .word   2
21      .globl  a_array
22      .bss
23      .align  3
24      .type   a_array, @object
25      .size   a_array, 20
```

实验任务——汇编代码生成

- 目标机器：RISC-V 64GC

```
39  int aa = 0;
40  const int bb = 2;
41  int a_array[5];
42  int b_array[5] = {1, 2, 3, 4, 5};
43  int c_array[5] = {1, 2, 3};
44  float fa = 1.2;
45  double da = 2.3;
46
47  float func(float a, float b)
48  {
49      float c = a;
50      c = a + b;
51      return c;
52  }
53
54  int main(void)
55  {
56      float a = 3.2;
57      float b = 4.3;
58      int c = 4;
59      float d;
60      d = func(a, b);
61      return 0;
62  }
```

```
21      .globl  a_array
22      .bss
23      .align  3
24      .type   a_array, @object
25      .size   a_array, 20
26  a_array:
27      .zero   20
28      .globl  b_array
29      .data
30      .align  3
31      .type   b_array, @object
32      .size   b_array, 20
33  b_array:
34      .word   1
35      .word   2
36      .word   3
37      .word   4
38      .word   5
39      .globl  c_array
40      .align  3
41      .type   c_array, @object
42      .size   c_array, 20
43  c_array:
44      .word   1
45      .word   2
46      .word   3
47      .zero   8
```

实验任务——汇编代码生成

- 目标机器：RISC-V 64GC

```
39  int aa = 0;
40  const int bb = 2;
41  int a_array[5];
42  int b_array[5] = {1, 2, 3, 4, 5};
43  int c_array[5] = {1, 2, 3};
44  float fa = 1.2;
45  double da = 2.3;
46
47  float func(float a, float b)
48  {
49      float c = a;
50      c = a + b;
51      return c;
52  }
53
54  int main(void)
55  {
56      float a = 3.2;
57      float b = 4.3;
58      int c = 4;
59      float d;
60      d = func(a, b);
61      return 0;
62  }
```

```
48      .globl  fa
49      .section    .sdata,"aw"
50      .align  2
51      .type   fa, @object
52      .size   fa, 4
53  fa:
54      .word   1067030938
55      .globl  da
56      .align  3
57      .type   da, @object
58      .size   da, 8
59  da:
60      .word   1717986918
61      .word   1073899110
```

实验任务——汇编代码生成

- RISC-V 64GC指令
- Chapter 24

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

RV64I Base Instruction Set (in addition to RV32I)

imm[11:0]		rs1	110	rd	0000011	LWU
imm[11:0]		rs1	011	rd	0000011	LD
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	SD
000000	shamt	rs1	001	rd	0010011	SLLI
000000	shamt	rs1	101	rd	0010011	SRLI
010000	shamt	rs1	101	rd	0010011	SRAI
imm[11:0]		rs1	000	rd	0011011	ADDIW

实验任务——汇编代码生成

- RISC-V 64GC ABI
  - 寄存器约定
  - 调用约定
- Chapter 25

Integer Register Convention

Name	ABI Mnemonic	Meaning	Preserved across calls?
x0	zero	Zero	-- (Immutable)
x1	ra	Return address	No
x2	sp	Stack pointer	Yes
x3	gp	Global pointer	-- (Unallocatable)
x4	tp	Thread pointer	-- (Unallocatable)
x5-x7	t0-t2	Temporary registers	No
x8-x9	s0-s1	Callee-saved registers	Yes
x10-x17	a0-a7	Argument registers	No
x18-x27	s2-s11	Callee-saved registers	Yes
x28-x31	t3-t6	Temporary registers	No

实验任务——汇编代码生成

- RISC-V 64GC ABI
  - 寄存器约定
  - 调用约定
- Chapter 25

Floating-point Register Convention

Name	ABI Mnemonic	Meaning	Preserved across calls?
f0-f7	ft0-ft7	Temporary registers	No
f8-f9	fs0-fs1	Callee-saved registers	Yes*
f10-f17	fa0-fa7	Argument registers	No
f18-f27	fs2-fs11	Callee-saved registers	Yes*
f28-f31	ft8-ft11	Temporary registers	No

gnu工具链

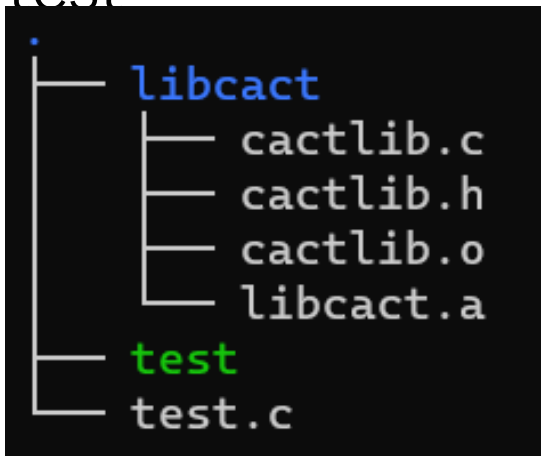
- $riscv64-unknown-elf-gcc -S test.s test.c

```c
int main(void)
{
    int a = 2;
    int b = 3;
    int c = a + b;
    print_int(c);
    return 0;
}
```

```
    .file   "test.c"
    .option nopic
    .attribute arch, "rv64i2p0_m2p0_a2p0_f2p0_d2p0_c2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  1
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-32
    sd  ra,24(sp)
    sd  s0,16(sp)
    addi    s0,sp,32
    li  a5,2
    sw  a5,-20(s0)
    li  a5,3
    sw  a5,-24(s0)
    lw  a4,-20(s0)
    lw  a5,-24(s0)
    addw    a5,a4,a5
    sw  a5,-28(s0)
    lw  a5,-28(s0)
    mv  a0,a5
    call    print_int
    li  a5,0
    mv  a0,a5
    ld  ra,24(sp)
    ld  s0,16(sp)
    addi    sp,sp,32
    jr  ra
    .size   main, .-main
    .ident  "GCC: (GNU) 10.2.0"
```

模拟器

- $riscv64-unknown-elf-gcc -o test test.c -L./libcact –lcact
- $spike pk test




```c
int main(void)
{
    int a = 2;
    int b = 3;
    int c = a + b;
    print_int(c);
    return 0;
}
```

```
lishuaijiang@ACT100:~/workspace/demo$ spike pk test
bbl loader
5
```

优化

- 基本块内（龙书8.5）
  - 窥孔优化（龙书8.7）
    - 中间代码、汇编代码
    - 消除冗余的load/store
    - 不可达代码消除
    - 控制流优化
    - 代数化简/强度削减
  - 常数表达式计算（常数折叠）
    - 在编译时计算操作数为常数的表达式。

优化 —— 窥孔优化

- 消除冗余的load/store

```c
int main(void)
{
  int a = 2;
  int b = 3;
  int c = a + b;
  print_int(c);
  return 0;
}
```

test.c

```
10 main:
11          addi    sp,sp,-32
12          sd      ra,24(sp)
13          sd      s0,16(sp)
14          addi    s0,sp,32
15          li      a5,2
16          sw      a5,-20(s0)
17          li      a5,3
18          sw      a5,-24(s0)
19          lw      a4,-20(s0)
20          lw      a5,-24(s0)
21          addw    a5,a4,a5
22          sw      a5,-28(s0)
23          lw      a5,-28(s0)
24          mv      a0,a5
25          call    print_int
26          li      a5,0
27          mv      a0,a5
28          ld      ra,24(sp)
29          ld      s0,16(sp)
30          addi    sp,sp,32
31          jr      ra
```

-O0：情况1

```
10 main:
11          addi    sp,sp,-32
12          sd      ra,24(sp)
13          sd      s0,16(sp)
14          addi    s0,sp,32
15          li      a5,2
16          sw      a5,-20(s0)
17          li      a5,3
18          sw      a5,-24(s0)
19          lw      a4,-20(s0)
20          lw      a5,-24(s0)
21          addw    a5,a4,a5
22          sw      a5,-28(s0)
23          sw      a4,-28(s0)
24          lw      a5,-28(s0)
25          mv      a0,a5
26          call    print_int
27          li      a5,0
28          mv      a0,a5
29          ld      ra,24(sp)
30          ld      s0,16(sp)
31          addi    sp,sp,32
32          jr      ra
```

情况2

```
18          li      a5,3
19          sw      a5,-24(s0)
20          lw      a4,-24(s0)
```

情况3

优化 —— 窥孔优化

- 保证优化的正确性

```c
int main(void)
{
  int a = 2;
  int b = 3;
  int c = a + b;
  print_int(c);
  return 0;
}
```

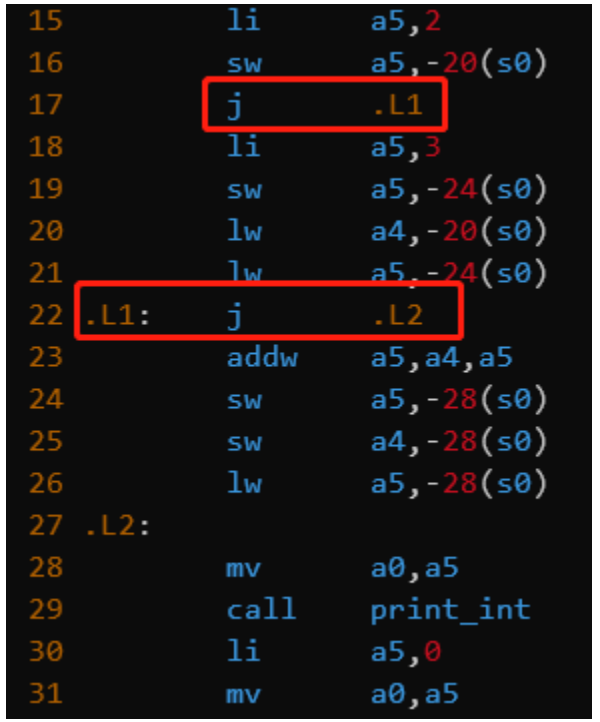
test.c

```
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        li      a5,2
        sw      a5,-20(s0)
        li      a5,3
        sw      a5,-24(s0)
        lw      a4,-20(s0)
        lw      a5,-24(s0)
        addw    a5,a4,a5
        sw      a5,-28(s0)
        lw      a5,-28(s0)
        mv      a0,a5
        call    print_int
        li      a5,0
        mv      a0,a5
        ld      ra,24(sp)
        ld      s0,16(sp)
        addi    sp,sp,32
        jr      ra
```

```
        addw    a5,a4,a5
        sw      a5,-28(s0)
.L1
        lw      a5,-28(s0)
```

优化 —— 窥孔优化

- 消除冗余的load/store
- 不可达代码消除
- 代数化简/强度削减
  - 用移位运算代替乘/除运算
  - 乘/除数为2的幂



优化 —— 常数表达式计算

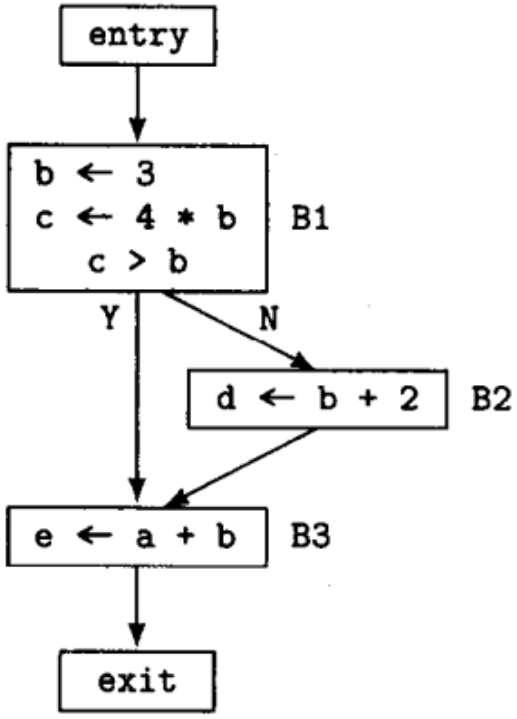
- 在编译时计算操作数为常数的表达式。
  - 判别表达式的所有操作数是否为常数值
  - 在编译时计算此表达式
  - 及用计算结果替代该表达式。
- 正确性：
  - 整型常数表达式：0作除数
  - 浮点常数表达式

优化

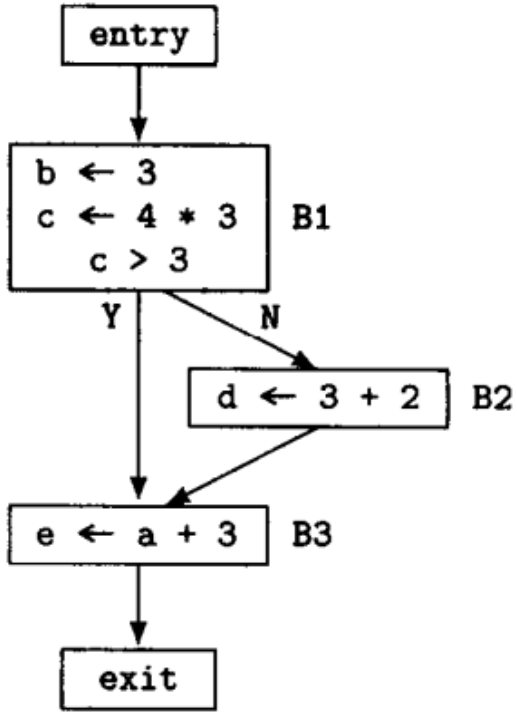
- 基本块内
  - 常数传播
  - 复写传播
  - 公共子表达式删除
    - 如果一个表达式不止被计算了一次，删除多余的计算（只留一个)
  - 死代码删除
    - 删除其结果从未使用的计算

优化 —— 常数传播

- 减少寄存器使用
- 增加其他优化的机会和效果



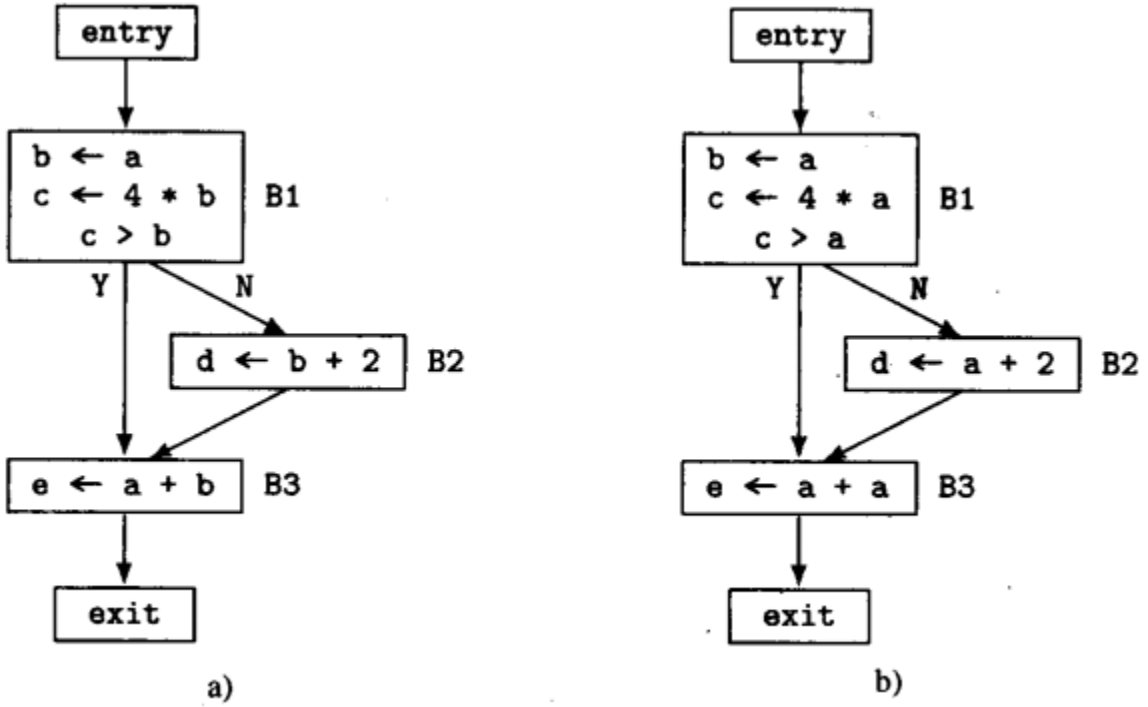
a)



b)

优化 —— 复写传播

- 增加其他优化的机会和效果
  - 死代码删除
  - a是整数时，e可以用移位实现



优化——局部公共子表达式删除

- 局部公共子表达式删除

位置	指令
1	c ← a + b
2	d ← m & n
3	e ← b + d
4	f ← a + b
5	g ← -b
6	h ← b + a
7	a ← j + a
8	k ← m & n
9	j ← b + d
10	a ← -b
11	if m & n goto L2

位置	指令
1	t1 ← a + b
2	c ← t1
3	d ← m & n
4	e ← b + d
5	f ← t1
6	g ← -b
7	h ← t1
8	a ← j + a
9	k ← m & n
10	j ← b + d
11	a ← -b
12	if m & n goto L2

位置	指令
1	t1 ← a + b
2	c ← t1
3	t2 ← m & n
4	d ← t2
5	e ← b + d
6	f ← t1
7	g ← -b
8	h ← t1
9	a ← j + a
10	k ← t2
11	j ← b + d
12	a ← -b
13	if m & n goto L2

位置	指令
1	t1 ← a + b
2	c ← t1
3	t2 ← m & n
4	d ← t2
5	t3 ← b + d
6	e ← t3
7	f ← t1
8	g ← -b
9	h ← t1
10	a ← j + a
11	k ← t2
12	j ← t3
13	a ← -b
14	if t2 goto L2

优化

- 基本块间（龙书9）
  - 常数传播
  - 全局公共子表达式删除
  - 循环不变量外提
- 后端
  - 寄存器分配
    - 使两个不重叠的临时变量存放在同一个寄存器中

优化

- Pass
  - 其实可能就是一个函数
- 怎么表示IR
  - 字符串？
  - class？
- 实验报告
  - 写明实现了哪几个优化，以及具体是怎么实现的？
  - 用户可以以什么方式使用？比如，通过命令行的参数：-O0, -O1

其他

- 一步一步来
  - 设计IR，生成IR
  - 汇编代码生成，通过基本case
  - 寄存器分配
    - 线性扫描
  - 在IR上进行优化
- 时间是充足的

Thanks!

Q&A