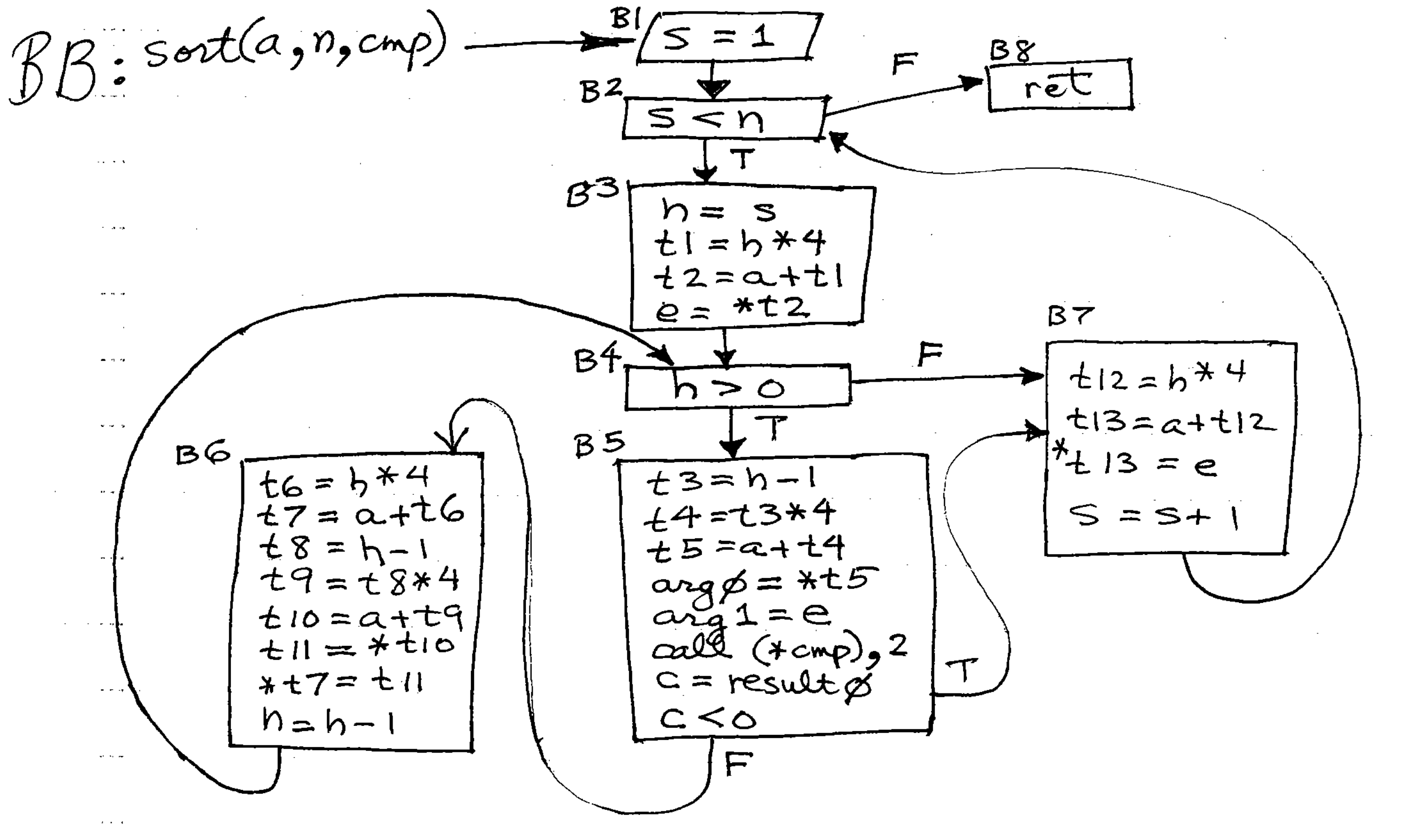
Optimization
Example (1)

Example: insertion sort

1. algorithmic optimization → qsort to 10?

```
void sort(ptr *a, int n, int (*cmp)(ptr,ptr)) {
    // a is an array, dim n
    for (s=1; s<n; s++) {       // s = #sorted
        h = s;                  // h = posn of hole
        e = a [h]               // e = elt @ hole
        while (h>0) {
            c = (*cmp)(a[h-1], e);
            if ( c<0) break ;
            a [h] = a [h-1];
            h = h-1 ;
        }
        a [h] = e;
    }
}
```



```
B1: s = 1
B2: s < n
B8: ret
B3: h = s
    t1 = h*4
    t2 = a+t1
    e = *t2
B4: h > 0
B5: t3 = h-1
    t4 = t3*4
    t5 = a+t4
    argø = *t5
    arg1 = e
    call (*cmp), 2
    c = resultø
    c<0
B6: t6 = h*4
    t7 = a+t6
    t8 = h-1
    t9 = t8*4
    t10 = a+t9
    t11 = *t10
    *t7 = t11
    h = h-1
B7: t12 = h*4
    t13 = a+t12
    *t13 = e
    s = s+1
```

Optimization Example (2)

1. Algebraic Opt ⇒ canonical form.

B5: t5 = a + t4
 = a + t3*4
 = a + (h−1)*4
 = a + h*4 − 4

B6: t10 = a + t9
 = a + t8*4
 = a + (h−1)*4
 $= \underbrace{a + \underbrace{h*4}_{t6}}_{t7} - 4$

2. Local common subex:

B6:

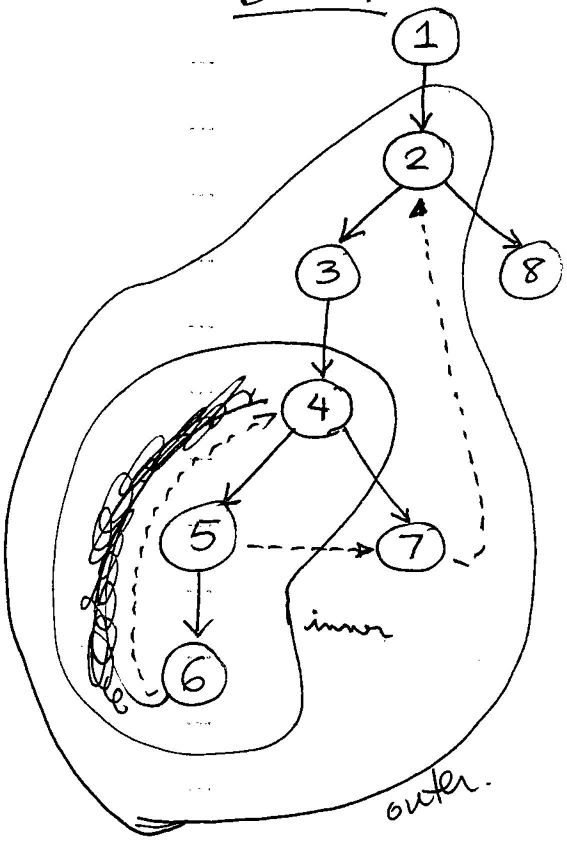
```
t6 = h*4
t7 = a + t6
t14 = t7 - 4
t11 = *t14
*t7 = t11
h = h - 1
```

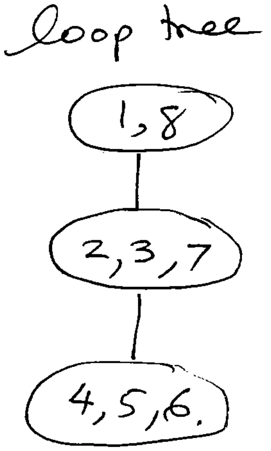
elim: t8, t9, t10
 ⇒ t14

```
B5: t15 = h*4
    t16 = a+t15
    t17 = t16-4
    argφ = *t17
    arg1 = e
    call (*cmp), 2
    c = resultφ
    c<0
```

3. Local const folding — no
copy prop — no
const prop — no.

4. Dom





Optimization
Example (3)

5. Reaching defs

B3 : h , B6 : h ⟶ reaches B4, B5, B6
Bϕ : a ⟶ reaches B5, B6 ∴ invariant

6. Inner loop : 4, 5, 6

induction vars: dead if input to other ind vars.

basic: $h = (h, 1, 0)$

derived:
$t6 = (h, 4, 0)$ dead.
$t7 = (h, 4, a)$ *
$t14 = (h, 4, a-4)$ * $\equiv (t7 - 4)$
$t15 = (h, 4, 0)$ dead
$t16 = (h, 4, a)$ * $\equiv$ t7 dead
$t17 = (h, 4, a-4)$ * $\equiv$ t14

almost useless: $h > \phi$
$t7 = h*4+a$ $h*4 > \phi*4$
$h*4+a > 0*4+a$
$\boxed{t7 > a}$

B4H: s1 = h*4
s7 = s1 + a
s14 = s7 − 4

B4: s7 > a

B5: argϕ = *s14
arg 1 = e ⟵ loop. inv.
call (*cmp), 2
c = result ϕ
c < 0

B6: t11 = *s14
*s7 = t11
s7 = s7 − 4
s14 = s14 − 4

Optimization Example (4)

7. Outer Loop (2,3,7)

```
basic:   s = (s,1,0)  → invariant in (4,5,6)
derived: h = (s,1,0)  → used in (4,5,6)
         t1 = (s,4,0)   dead
copy prop → t2 = (s,4,a)   * ——→ s3
         t12 = (s,4,0) ≡ t1  dead
         t13 = (s,4,a) ≡ t2 * ——→ s3
```

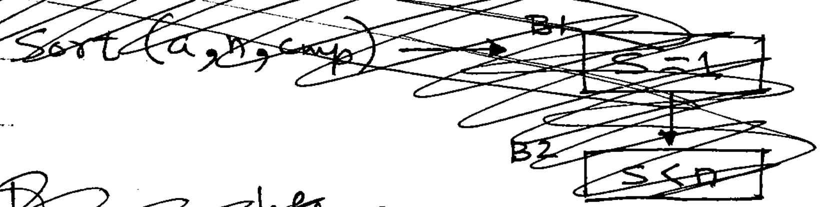
```
B2H:  s2 = s*4
      s3 = s2+a

B2:   s<n
```

~~B3: copy prop~~

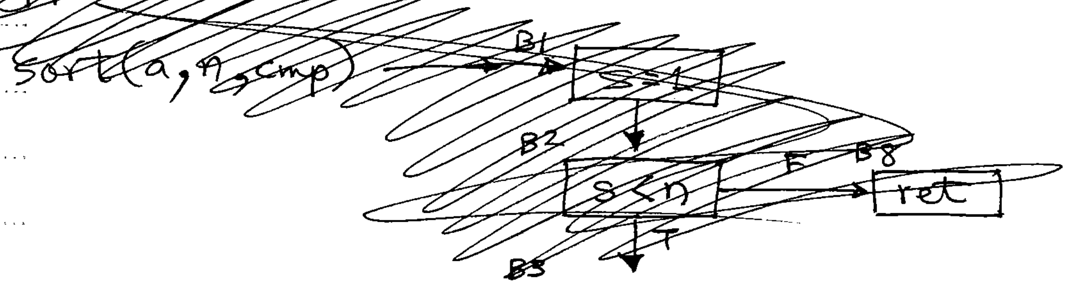
~~t1=~~ h=s

```
B3:  e = *s3

B7:  *s3 = e
     s3 = s3+4
     s = s+1
```

~~8. Re-assemble~~



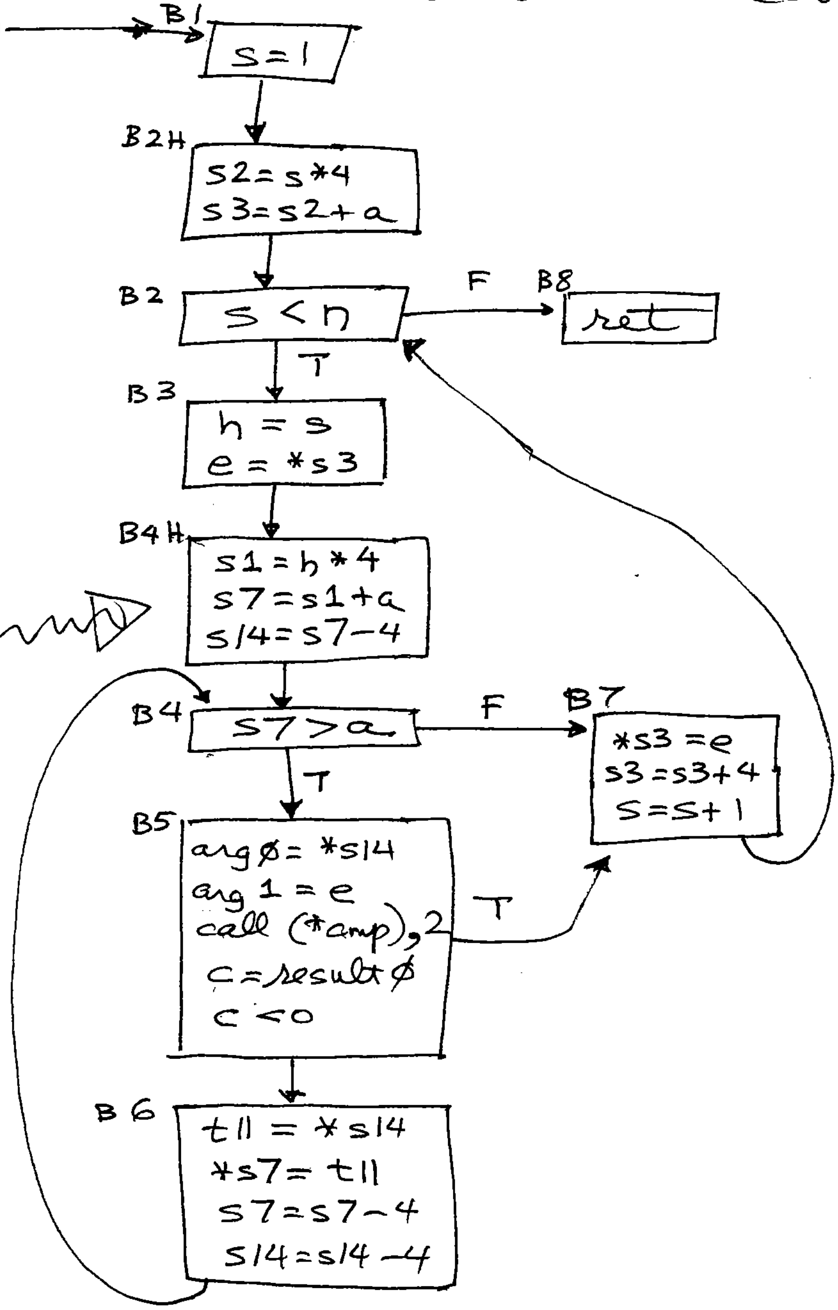
~~8. Reassemble~~



8. Summary (so far?)

- what do we have?
- what could we have done?

sort (a,n,cmp)



missed?

global copy prop

B3 : h = s

B4H : s1 = s*4
      s7 = s*4 + a

B2H: s3 = s*4 + a.

∴ s7 = s3

Machine idiom

inner loop

s14 ≡ s7 - 4

∴ *s14 ≡ *[s7-4].

alias analysis

*s3, *s7, *s14 - aliased.

9. Redo

global copy prop: $s7 \equiv s3$

machine idiom: $*s14 = *[s7-4]$

$\therefore$ s is almost useless.

$$\begin{aligned} s<n &\equiv s*4<n*4 \\ &\equiv s*4+a<n*4+a \\ &\equiv s3<n*4+a \end{aligned}$$

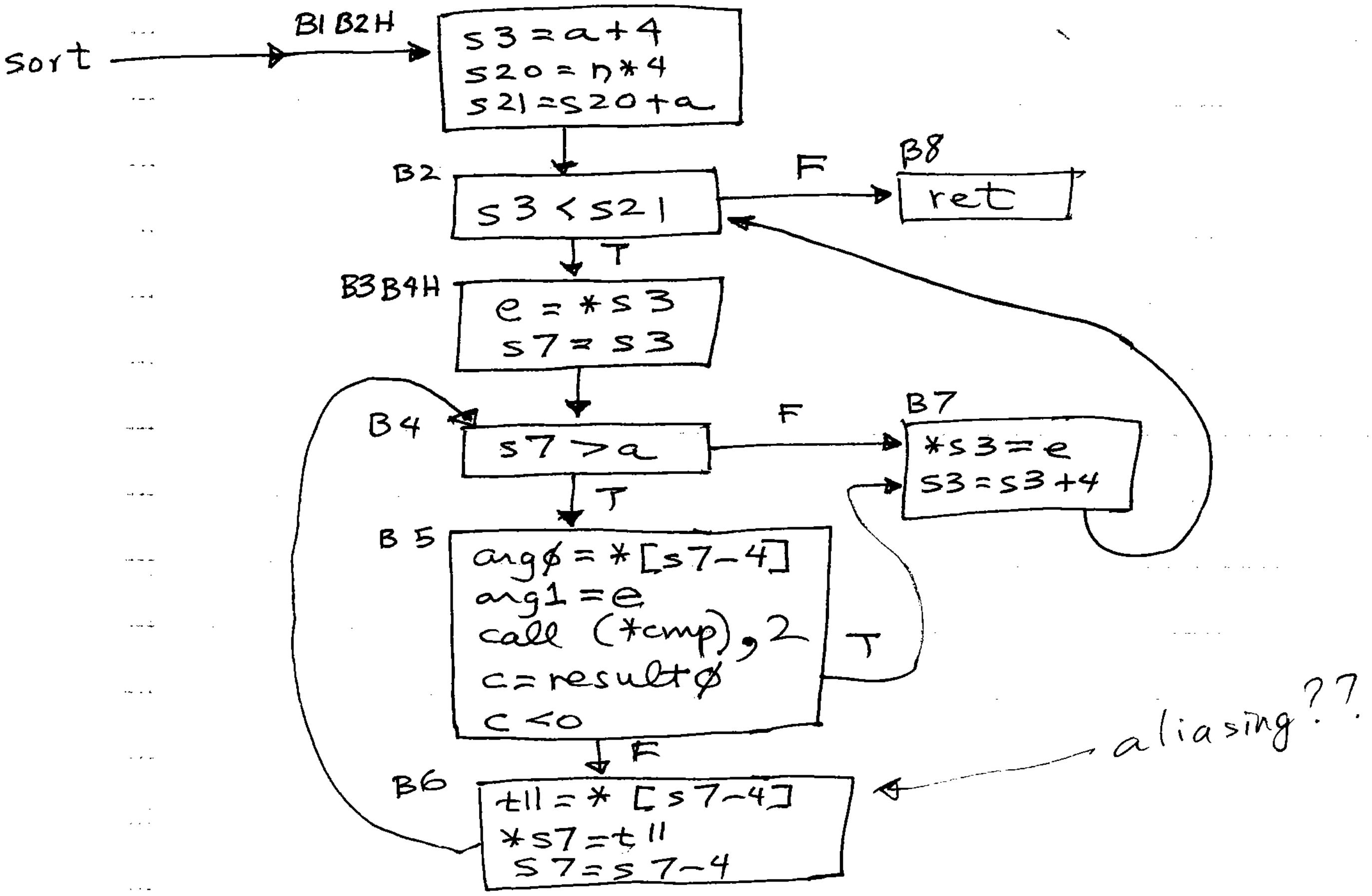
so hoist $n*4+a$ to B2H:

$$\left.\begin{aligned} s20 &= n*4 \\ s21 &= s20+a \end{aligned}\right\} \rightarrow \text{B2H}$$

B2: $\boxed{s3<s21}$

merge BB:

$$\rightarrow \begin{aligned} &\left.\begin{aligned} &s=1 \\ &s2=s*4 \end{aligned}\right\} \text{constant prop: } s2=4 \\ &\left.s3=s2+a\right\} \text{constant prop: } s3=a+4 \\ &s20=n*4 \\ &s21=s20+a \end{aligned}$$



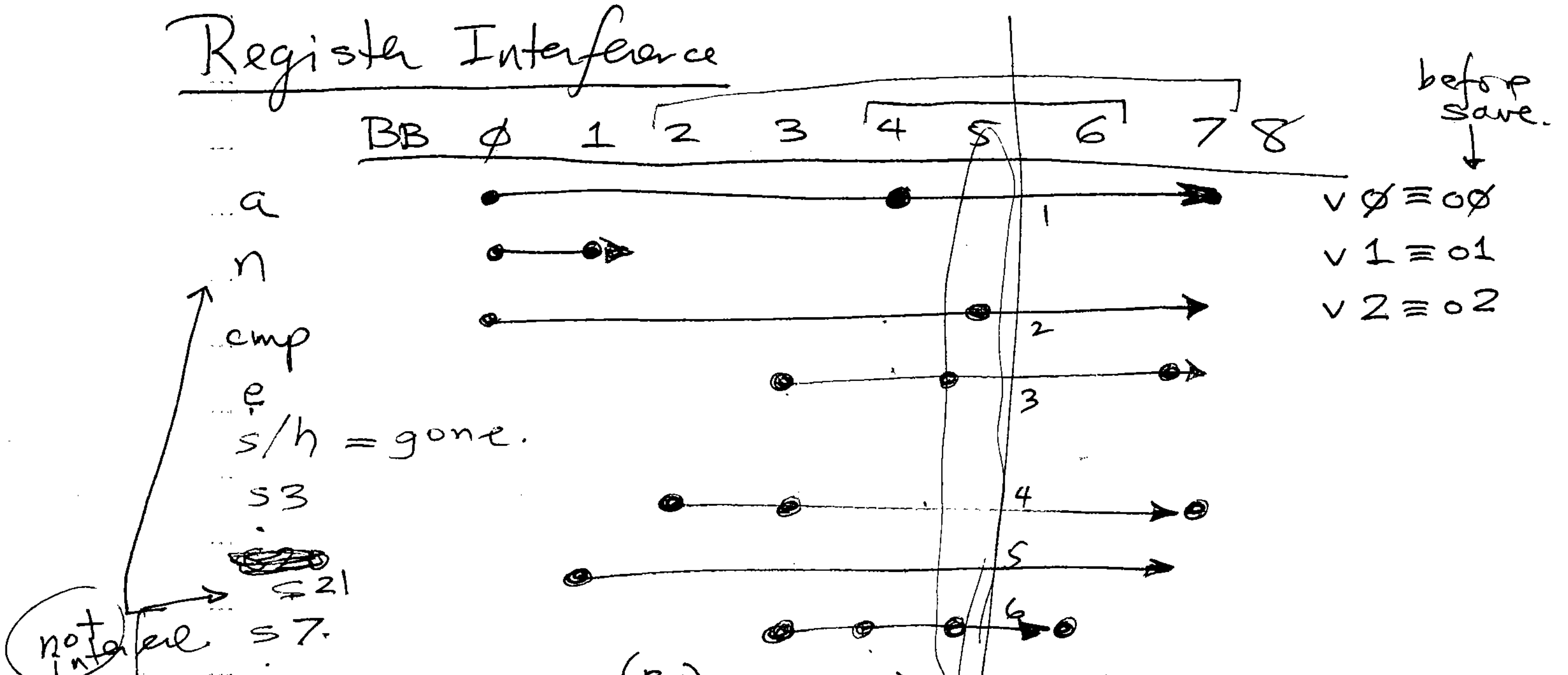
```
B1B2H:  s3=a+4
        s20=n*4
        s21=s20+a

B2:     s3<s21

B8:     ret

B3B4H:  e=*s3
        s7=s3

B4:     s7>a

B7:     *s3=e
        s3=s3+4

B5:     argφ=*[s7-4]
        arg1=e
        call (*cmp), 2
        c=resultφ
        c<0

B6:     t11=*[s7-4]
        *s7=t11
        s7=s7-4
```

Optimization Example (7)

Reaching & LiveVar

du chain
ud chain

Registe Interference



Local only : s20 (B1), t11 (B6)

→ not interfere s21

|| s20
t11

need 6 regs + 2 tmps.

~~no calls ∴ is leaf fn if~~
~~can alloc w/o i-regs/w/o~~ l regs.

~~∴ a = o0~~
~~n = o~~

~~∴ o0 = a~~
~~o1 = n, s20,~~ s21
~~o2 = cmp~~
~~o3 = s3.~~
~~o4 = e~~
~~o5 = s7~~

10. Sparc Reg alloc

- calls (*cmp) ∴ not leaf fn
- lots of regs; but alloc min.

~~a : i~~
~~n :~~
~~cmp :~~

i0 = a
i1 = n, s20, s21
i2 = cmp
i3 = s3
o0 = arg0, result0   i4 = e
o1 = arg1   i5 = s7

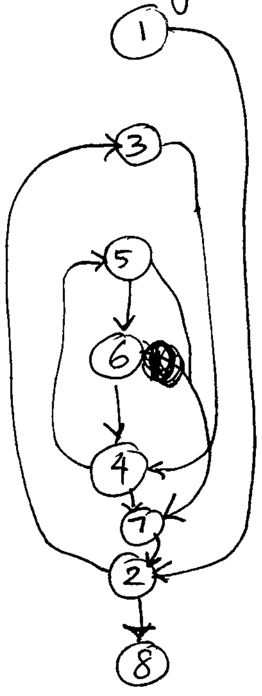
~~o~~ o0 = c, t11

use ireg if Reaches to cross call cwr.

11. Branch delay slots

1. sink insn past cti
2. bum insn from target.
~~3. (Load delay slots?)~~

Rearrange code → while test @ end.



12. Load delay slots (Sparc = optional)

B6 : incr in mid.

loads: B3, B5, B6.

13. call delay slot B5

14. branch delay B2, B4, B6.

Final Sparc Assembly

Optimization Example (9)

```
sort:        save   %sp,-96,%sp
             add    %i∅,4,%i3        ; s3=a+4
             sll    %i1,2,%i1        ; s20=n*4
             ba     B2               ; while @ End
             add    %i1,%i∅,%i1      ; DELAY: s21=s20+a
----------------------------------------------------------
    B3:      ld     [%i3],%i4        ; e = *s3
             ba     B4               ; while @ End
             or     %i3,%g∅,%i5      ; DELAY: s7=s3
----------------------------------------------------------
 *   B5:     ~~ld [%i5-4],%o∅ ; arg∅=*[s7-4]~~
             jmpl   %i2,%o7          ; call
             or     %i4,%g∅,%o1      ; DELAY: arg1=e
             [illegible]             ; after call.
             brlez,pn %o∅,B7
DELAY??      nop                     ~~DELAY??~~
----------------------------------------------------------
*ALIAS??  B6: ld    [%i5-4],%o∅      ; t11=*[s7-4]

? SWAP  →  { st     %o∅,[%i5]        ; *s7=t11
(load delay  add    %i5,-4,%i5
----------------------------------------------------------
    B4:      subcc  %i5,%i∅,%g∅
             bg,pt  B5
             nop ⇒ ld [%i5-4],%o∅    BUM (B5)
----------------------------------------------------------
    B7:      st     %i4,[%i3]
             add    %i3,4,%i3
----------------------------------------------------------
    B2:      subcc  %i3,%i1,%g∅
             bl,pt  B3
             nop ⇒ ld [%i3],%i4      BUM (B3)
----------------------------------------------------------
    B8:      ret
             restore    ; DELAY.
----------------------------------------------------------
```