Branch instructions

- Types of branch instructions
  - ✓ conditional branch
  - ✓ unconditional branch, jump
  - ✓ **call**

- Conditional branch
  - ✓ Use condition code (CC: Z, N, V, C)
  - ✓ Instructions that update condition code
    - ➢ addcc, subcc, …

Branch instructions (signed number)

opcode	branch condition
ba	goto, branch always
bn	branch never
bl	branch on less than 0
ble	branch on less than or equal to 0
be	branch on equal to 0
bne	branch on not equal to 0
bge	branch on greater than or equal to 0
bg	branch on greater than 0

- Branch instruction format
  - ✓ op-code   label
- Example

← which type of instruction?

```
    bl   t1
    nop
      :
t1:   :
```

- ✓ What if condition test result is‘true’?
- ✓ What if condition test result is‘false’?

```
if (a > b)
    c = a - b;
```

→ Suppose a, b, c is assigned to %l0, %l1, %l2, respectively

```
        subcc %l0, %l1, %g0    !   a -  b
        ble      next          !   If a -  b <= 0 then  branch
        nop
        sub %l0, %l1, %l2      !  when a -  b  > 0
next:     ...                  !  when a – b  <= 0
```

Do-while example 1

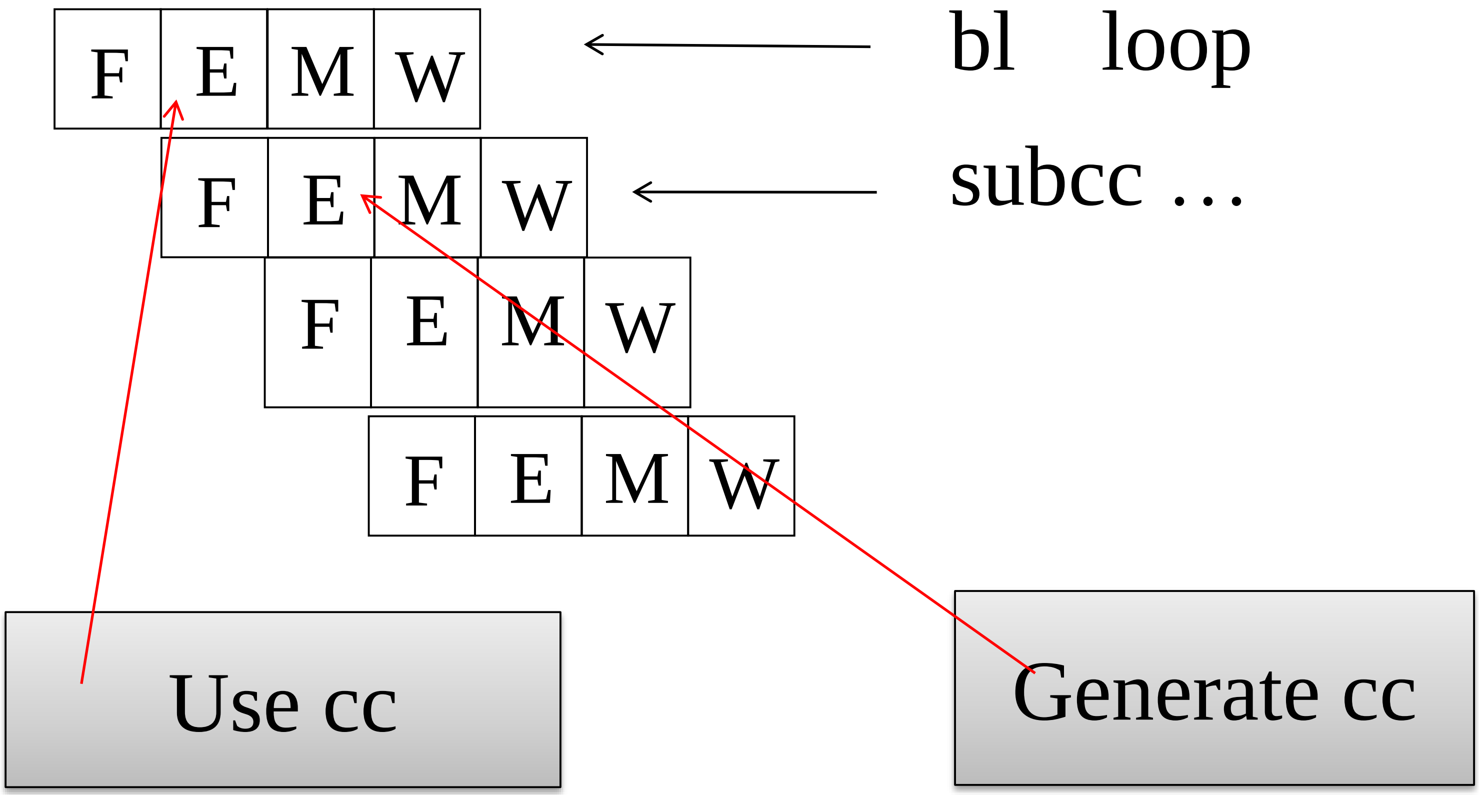
```
main() {
  int x, y;

  x = 0;
  do {
    y = ((x - 1) * (x - 7)) / (x - 11);
    x++;
  } while (x < 11);
}
```

Var.	Register
x	%l0
y	%l1

```
        .global  main
main:
        save     %sp, -96, %sp
         clr      %l0
loop:
        sub       %l0, 1,     %o0
        call      .mul
        sub
        call      .div
        sub
        mov       %o0, %l1
        add       %l0, 1,     %l0
        subcc     %l0, 11,   %g0
        bl        loop
        nop
        mov     1,         %g1
        ta      0
```

CC update

- Using delay slot (eliminate nop)
  - ✓ Move mov instruction to the delay slot
- What if we move subcc %l0, 11, %g0 to DS?



F E M W

F E M W

F E M W

F E M W

bl   loop

subcc …

Use cc

Generate cc

Updating CC

- cmp   rs1, reg_or_imm

  ↔ subcc   rs1, reg_or_imm, %g0

-
  ```
  loop: sub    %l0, 1, %o0
        call   .mul
        sub    %l0, 7, %o1
        call   .div
        sub    %l0, 11, %o1
        add    %l0,  1, %l0
        cmp    %l0,  11
        bl     loop
        mov    %o0,  %l1
        mov     1,   %g1
        ta    0
  ```

Implementing while statement

✓ Sum b/w 0 & 9

```
s = 0;
i = 0;
while (i < 10){
  s = s + i;
  i++;
}
```

□ ςαριαβλεσ – **σ: %ο0**

ι: %λ0

```
      clr %o0         ! s = 0
      clr %l0         ! i = 0
test: cmp  %l0, 10
                   ! If i ≥ 10, exit loop
      nop          ! Delay slot
      add %o0,%l0,%o0  ! s =s + i
      add %l0, 1, %l0  ! i++
                   ! Loop
      nop          ! Delay slot
next: ...
```

Reorganizing code

```
      clr %o0     !s=0
      clr %l0     !i=0
test: cmp %l0, 10
      bge next
      nop
      add %o0,%l0,%o0
      add %l0, 1, %l0
      ba test
      nop
next: ...
```

```
      clr %o0
      clr %l0
      ba test
      nop   ! Delay slot
loop: add %o0, %l0, %o0 !s=s + i
      add %l0, 1, %l0   ! i++
test: cmp %l0, 10
            ! To loop or not to?
      nop   ! Delay slot
νεξτ: □
```

Optimization (1)

```
     clr %o0                 ! s = 0
     ba test
     clr %l0                 ! i = 0
Loop:add %o0, %l0, %o0       ! s = s + i
     add %l0, 1, %l0         ! i++
Test:cmp %l0, 10             ! subcc %l0, 10, %g0
                          ! Jump to while-loop-start
     nop                     ! Delay cycle
```

Optimization (2)

```
     clr %o0               ! s = 0
     clr %l0               ! i = 0
     ba test
     cmp %l0, 10           ! subcc %l0, 10, %g0
Loop:add %o0, %l0, %o0     ! s = s + i
     add %l0, 1, %l0       ! i++
     cmp %l0, 10           ! subcc %l0, 10, %g0
Test:bl loop
     nop
```

Optimization (3)

```
     clr %o0        ! s = 0
     clr %l0        ! i = 0
     ba test
     cmp %l0, 10    ! subcc %l0, 10, %g0
Loop:add %l0, 1, %l0     ! i++
     cmp %l0, 10         ! subcc %l0, 10, %g0
Test:bl,a loop
     add %o0, %l0, %o0   ! s = s + i
```

Annulled branch

- Notation
  - ✓ op-code**, a** label

- Execution
  - ✓ If condition is **true**: Instruction in delay slot is executed normally
  - ✓ If condition is **false**: Execution of instruction in delay slot is annulled

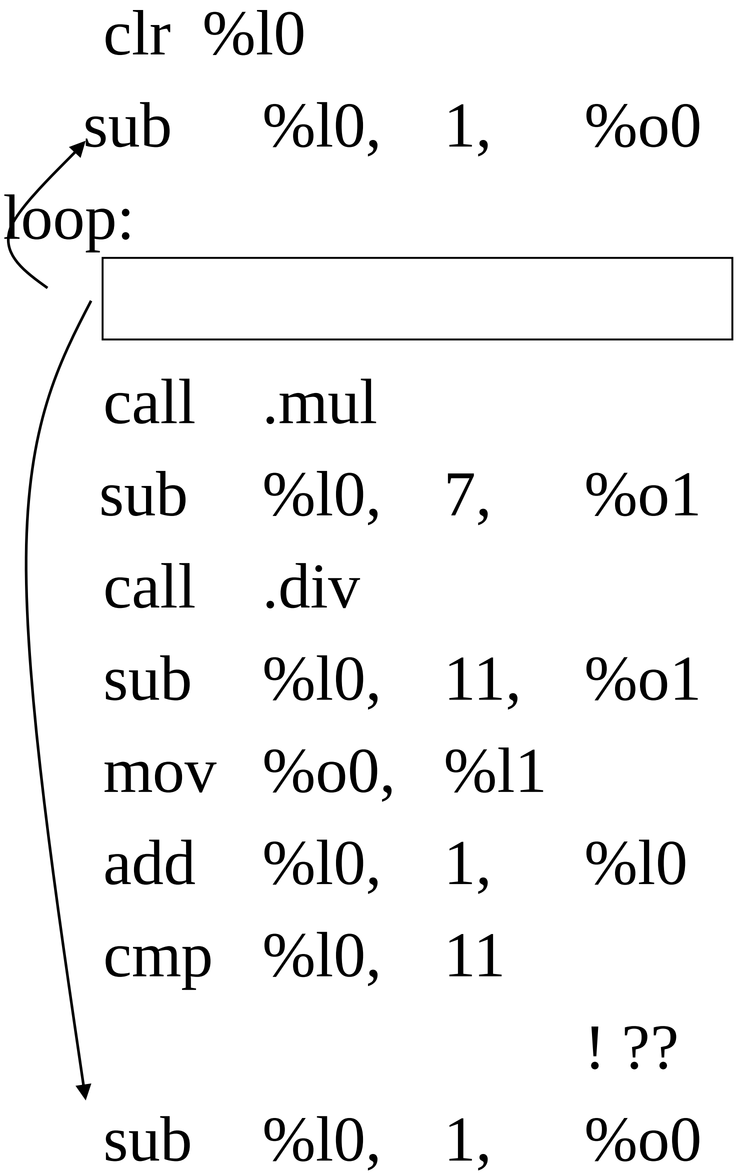
Back to do-while example (on Page 5)

```
      clr     %l0
loop:
      sub     %l0,    1,      %o0
      call    .mul
      sub     %l0,    7,      %o1
      call    .div
      sub     %l0,    11,     %o1
      mov     %o0,    %l1
      add     %l0,    1,      %l0
      cmp     %l0,    11
                              ! ??
      nop
```

```
      clr     %l0
loop:
      sub     %l0,    1,      %o0
      call    .mul
      sub     %l0,    7,      %o1
      call    .div
      sub     %l0,    11,     %o1

      add     %l0,    1,      %l0
      cmp     %l0,    11
                              ! ??
      mov     %o0,    %l1
```

Do-while example with annulled branch

```
     clr     %l0
loop:
     sub     %l0,    1,      %o0
     call    .mul
     sub     %l0,    7,      %o1
     call    .div
     sub     %l0,    11,     %o1
     mov     %o0,    %l1
     add     %l0,    1,      %l0
     cmp     %l0,    11
                             ! ??

     nop
```



```
     clr  %l0
    sub     %l0,    1,      %o0
loop:
     [                          ]
     call    .mul
     sub     %l0,    7,      %o1
     call    .div
     sub     %l0,    11,     %o1
     mov     %o0,    %l1
     add     %l0,    1,      %l0
     cmp     %l0,    11
                             ! ??
     sub     %l0,    1,      %o0
```

For loop

```
for (a = 1; a <= b; a++) c *= a;
        mov   1,      %l0
        ba    test
        nop
loop:   mov   %l0,    %o0
        call  .mul
        mov   %l2,    %o1
        mov   %o0,    %l2
        add   %l0, 1, %l0
test:
        cmp   %l0,    %l1
                                 ! ??
        nop
```

```
for(ex1;ex2;ex3) st;
→
ex1;
while(ex2){
   st;
   ex3;
}
```

a,b,c allocated to

%l0, %l1, %l2

For loop

```
for (a = 1; a <= b; a++) c *= a;

        ba      test
        mov     1,      %l0
loop:
        call    .mul
        mov     %l2,    %o1
        mov     %o0,    %l2
        add     %l0, 1, %l0
test:
        cmp     %l0,    %l1
                                ! ??
        mov     %l0,    %o0
```

```
for(ex1;ex2;ex3) st;
→
ex1;
while(ex2){
   st;
   ex3;
}
```

a,b,c allocated to

%l0, %l1, %l2

If-Then (1)

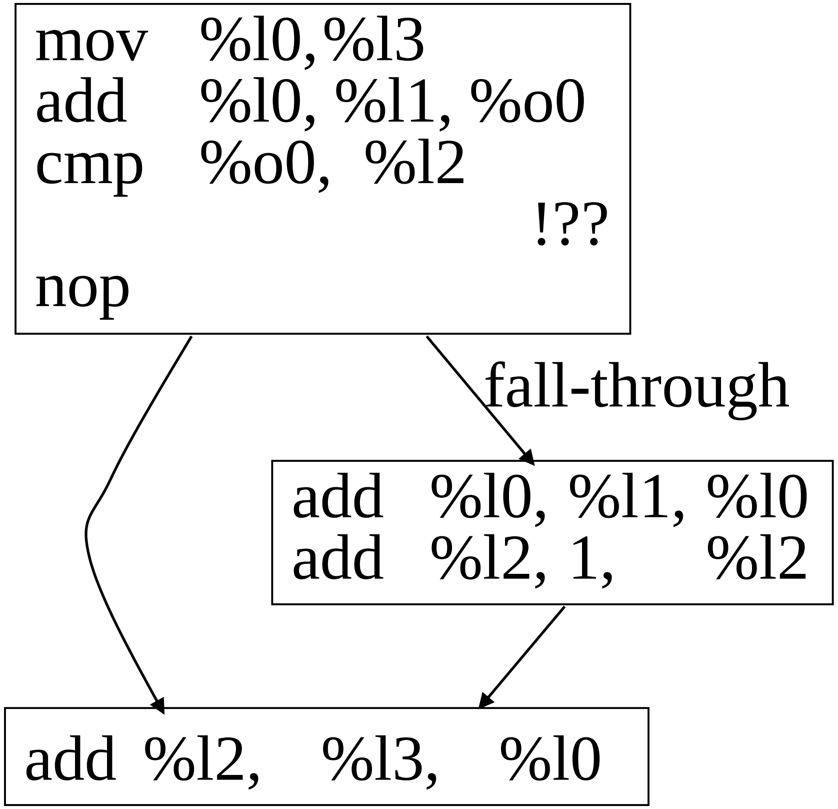
```
d = a;
if ((a + b) > c) {
  a += b;
  c++;
}
a = c + d;
```

```
        mov   %l0,   %l3
        add   %l0,   %l1,   %o0
        cmp   %o0,   %l2
                            ! ??
        nop
        add   %l0,   %l1,   %l0
        add   %l2,   1,     %l2
then:
        add   %l2,   %l3,   %l0
```

Var.	Register
a	%l0
b	%l1
c	%l2
d	%l3

If-Then (2)

```
       mov   %l0,    %l3
       add   %l0,    %l1,    %o0
       cmp   %o0,    %l2
                             ! ??
       nop
       add   %l0,    %l1,    %l0
       add   %l2,    1,      %l2
then:
       add   %l2,    %l3,    %l0
```



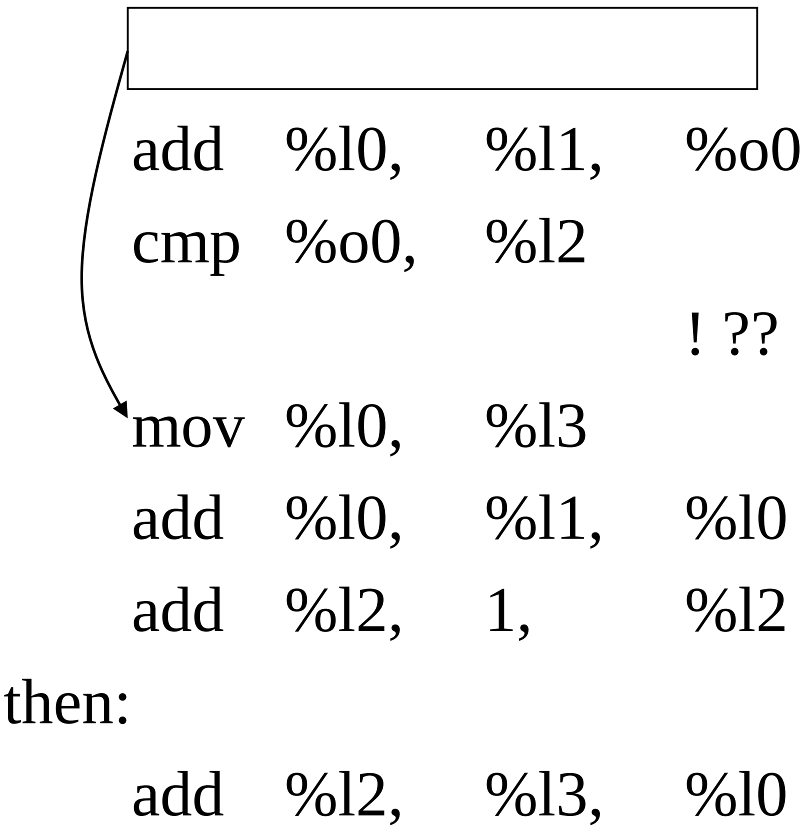
```
mov   %l0,%l3
add   %l0, %l1, %o0
cmp   %o0,  %l2
                  !??
nop
```

fall-through

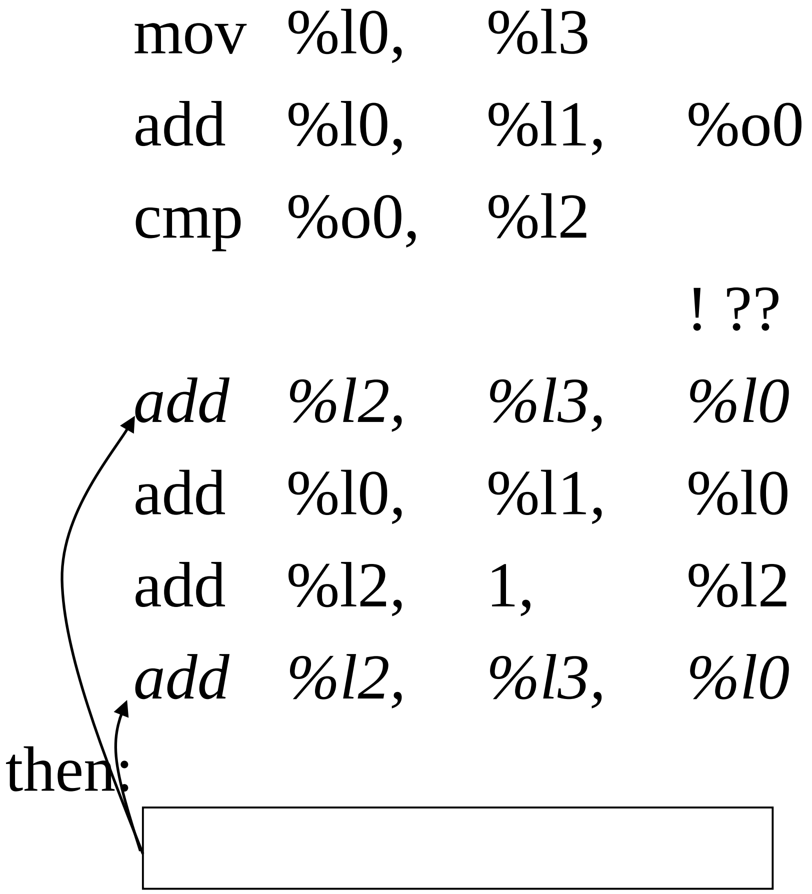
```
add   %l0, %l1, %l0
add   %l2, 1,     %l2
```

```
add  %l2,   %l3,   %l0
```

Optimized schedule



```
        add   %l0,   %l1,   %o0
        cmp   %o0,   %l2
                            ! ??
        mov   %l0,   %l3
        add   %l0,   %l1,   %l0
        add   %l2,   1,     %l2
then:
        add   %l2,   %l3,   %l0
```



```
        mov   %l0,   %l3
        add   %l0,   %l1,   %o0
        cmp   %o0,   %l2
                            ! ??
        add   %l2,   %l3,   %l0
        add   %l0,   %l1,   %l0
        add   %l2,   1,     %l2
        add   %l2,   %l3,   %l0
then:
```

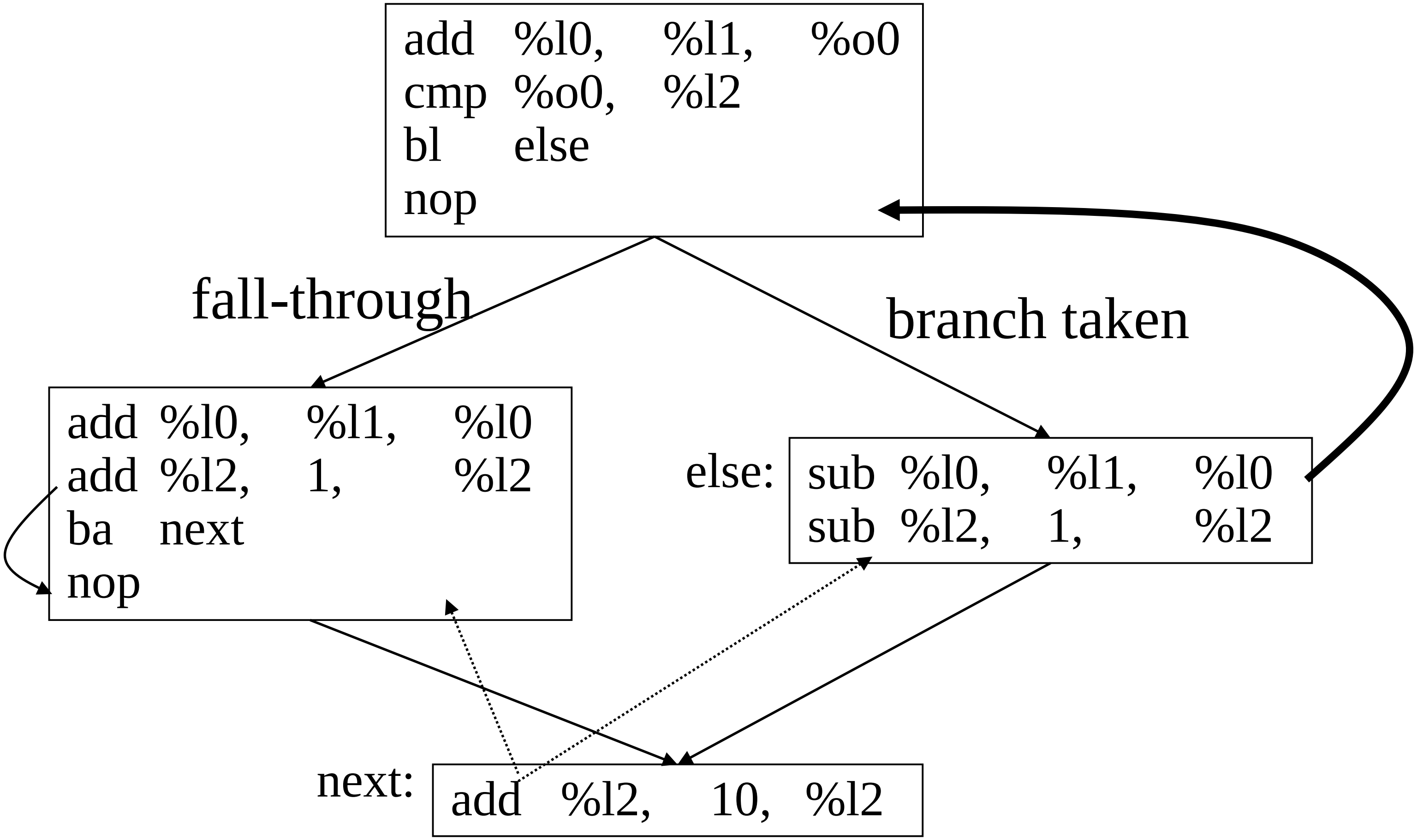
If-then-else

```
if ((a + b) >= c) {
  a += b;
  c++;
} else {
  a -= b;
  c--;
}
c += 10;
```

Var	Reg
a	%l0
b	%l1
c	%l2

```
        add     %l0,    %l1,    %o0
        cmp     %o0,    %l2
        bl      else
        nop
        add     %l0,    %l1,    %l0
        add     %l2,    1,      %l2
        ba      next
        nop
else:
        sub     %l0,    %l1,    %l0
        sub     %l2,    1,      %l2
next:
        add     %l2,    10,     %l2
```

If-then-else code schedule



```
add   %l0,    %l1,    %o0
cmp   %o0,    %l2
bl    else
nop
```

fall-through

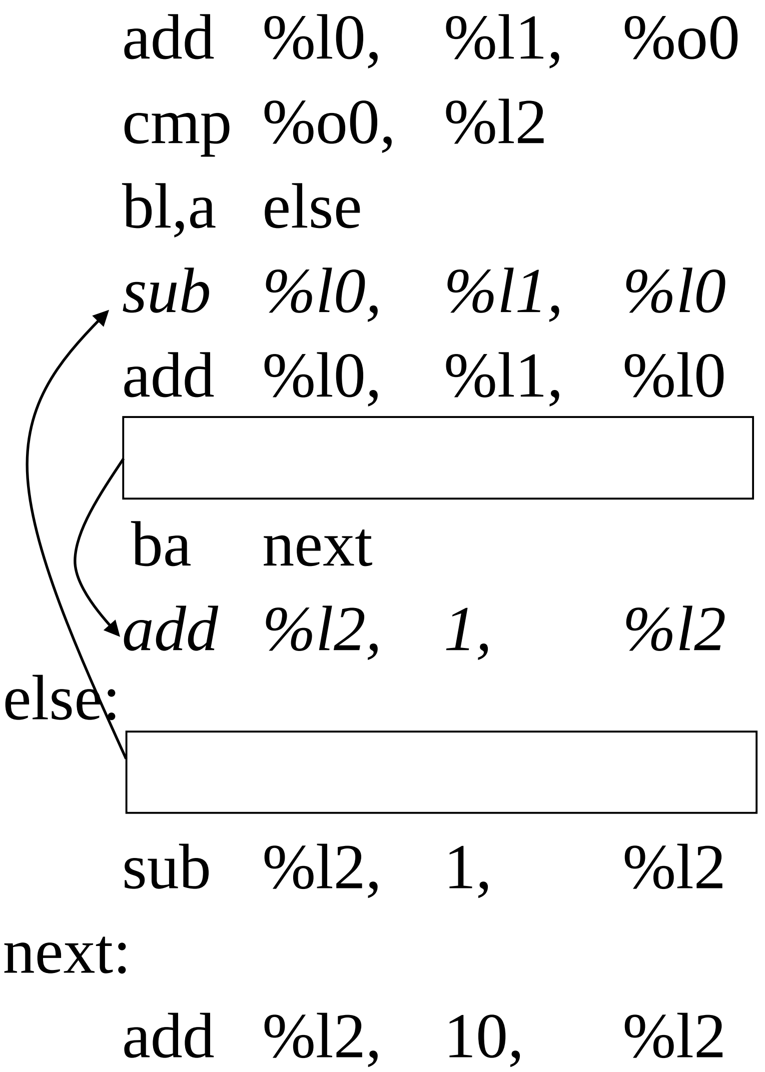
branch taken

```
add  %l0,    %l1,    %l0
add  %l2,    1,      %l2
ba   next
nop
```

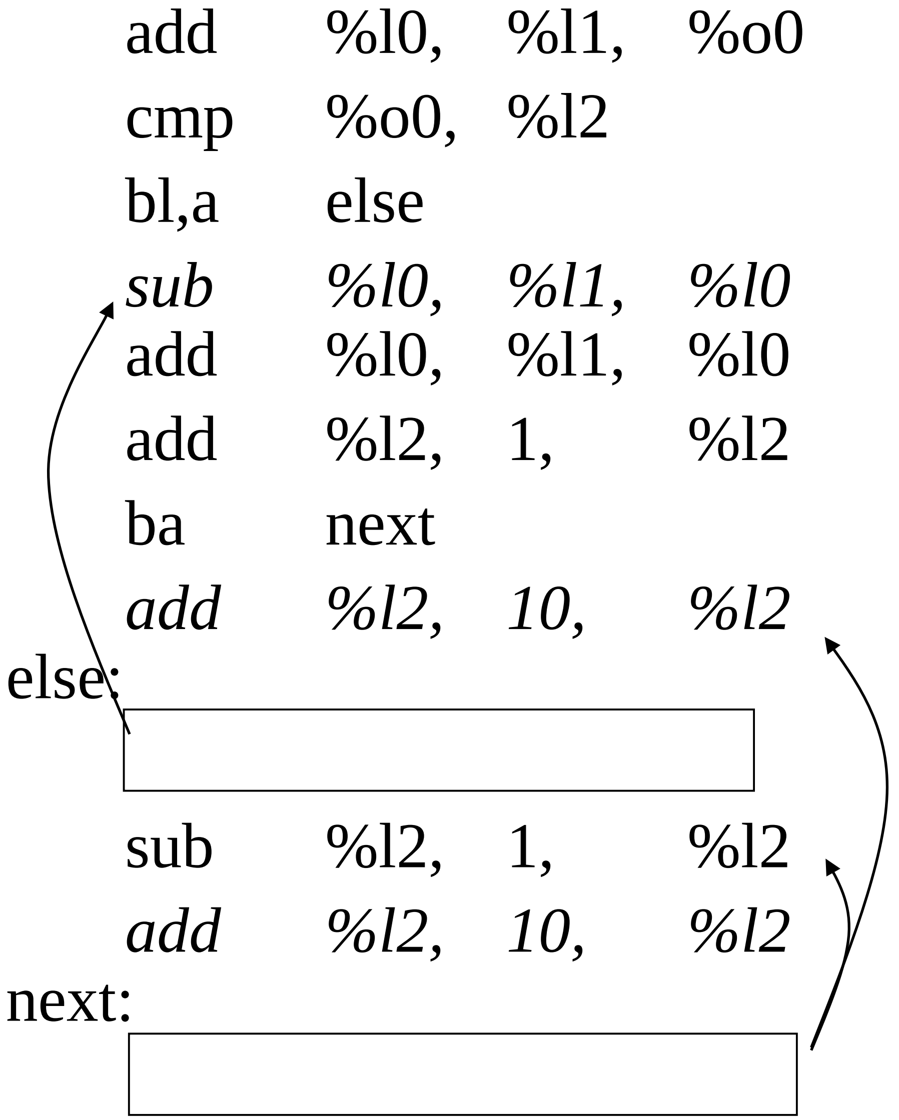
```
else:  sub  %l0,    %l1,    %l0
       sub  %l2,    1,      %l2
```

```
next:  add   %l2,    10,   %l2
```

After optimization



```
        add    %l0,    %l1,    %o0
        cmp    %o0,    %l2
        bl,a   else
        sub    %l0,    %l1,    %l0
        add    %l0,    %l1,    %l0

         ba    next
        add    %l2,    1,      %l2
else:

        sub    %l2,    1,      %l2
next:
        add    %l2,    10,     %l2
```



```
        add    %l0,    %l1,    %o0
        cmp    %o0,    %l2
        bl,a   else
        sub    %l0,    %l1,    %l0
        add    %l0,    %l1,    %l0
        add    %l2,    1,      %l2
        ba     next
        add    %l2,    10,     %l2
else:

        sub    %l2,    1,      %l2
        add    %l2,    10,     %l2
next:

```

Branch instructions and CC

- Branch instructions with CC test
  - ✓**signed** number case

OP code	CC
bl	N xor V= 1
ble	Z or (N xor V) = 1
be	Z = 1
bne	Z = 0
bge	N xor V = 0
bg	Z or (N xor V) = 0 | ¬ Z and ¬ (N xor V)

Subtraction of signed/unsigned number

- Signed Numbers
  - Negative numbers in 2’s complement representation
  - If sign bit (MSB) is 0(1) then positive(negative)
- n-bit numbers can represent:
  - signed: $-2^{n-1} \sim 2^{n-1} - 1$
  - unsigned: $0 \sim 2^{n} - 1$
- Same hardware/instructions for signed/unsigned numbers
- Subtraction is implemented as addition
  - ✓ $x - y = x + (-y) = x + (2^{n}-1-y) + 1$

Signed number example (n=8)

✓ Performing addition ignoring carry

```
      6-3 = 6+(-3)            3-6=3+(-6)

   +6: 00000110            +3: 00000011
+(-3): 11111101         +(-6): 11111010
       00000011                11111101
```

✓ What is sign of results?

Overflow example

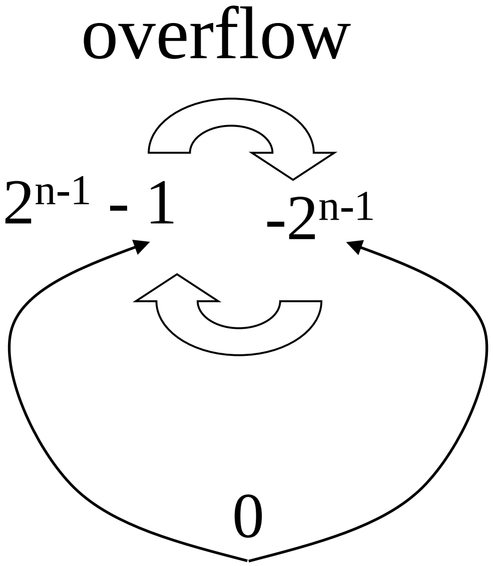
```
  127-(-1)=127+1          -128-1=-128+(-1)

 127: 01111111           -128:    10000000
+  1: 00000001        + (-1):     11111111
________________________________________________
      10000000                    01111111
```

✓ Are results reliable?

✓ How about sign?

Detection of overflow with addition (V bit)

(+) + (+) → (+)	V = 0
(+) + (-) → (+,-)	V = 0
(-) + (-) → (-)	V = 0
(+) + (+) → (-)	V = 1
(-) + (-) → (+)	V = 1



Detection of overflow with subtraction

(+) - (+) → (+,-)	$V = 0$,	(≥, <)
(-) - (-) → (+,-)	$V = 0$,	(≥, <)
(+) - (-) → (+)	$V = 0$	
(-) - (+) → (-)	$V = 0$	
(+) - (-) → (-)	$V = 1$	
(-) - (+) → (+)	$V = 1$	

- $V = C_{out} \oplus C_{out-1}$

Unsigned number example (n=4)

```
      12-3=12+(-3)      3-12=3+(-3)

   +12: 1100              +3: 0011
 +(-3): 1101          +(-12): 0100
 ----------------------------------
        1001                  0111
    carry occurs      no carry
        + 수 (+9)           -수 (-9)
```

- ✓ How to detect overflow?
- ✓ How to detect sign of results?

Branch instructions (unsigned number)

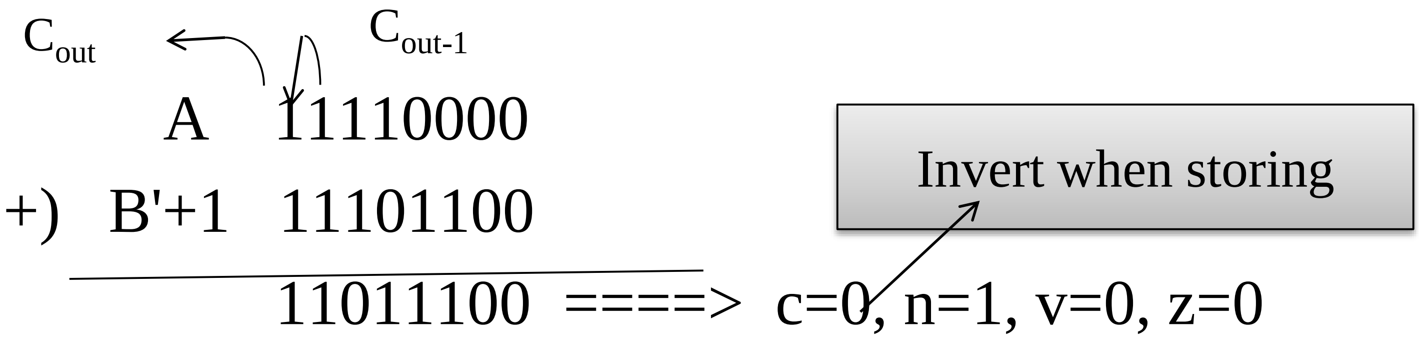
op	Behavior	CC
blu	branch on less than 0	C = 1
bleu	branch on less than or equal to 0	C = 1 or Z = 1
be	branch on equal to 0	
bne	branch on not equal to 0	
bgeu	branch on greater than or equal to 0	C = 0
bgu	branch on greater than 0	C = 0 and Z = 0

CC interpretation example (8-bit arithemetic)

- To generate CC, perform A - B = A + B' + 1 (i.e., using subcc)

A: 11110000 => 240_{10}, -16_{10}

B: 00010100 => 20_{10}



C_{out} C_{out-1}

```
      A    11110000
+)  B'+1   11101100
-------------------
           11011100  ====>  c=0, n=1, v=0, z=0
```

Invert when storing

CC interpretation

1. **unsigned** number

- If $A<B$, then $C=1$ $\therefore$ If $A\geq B$, then $C=0$
- If $A=B$, then $Z=1$ $\therefore$ If $A\neq B$, then $Z=0$
- If $A>B$ ($A\geq B$ and $A\neq B$), then $C=0$ and $Z=0$
- If $A\leq B$ ($A<B$ or $A=B$), then $C=1$ or $Z=1$

2. **signed** number

- If $A\geq B$ (i.e., $A-B \geq 0$), then
  - ✓ no overflow($v=0$) and $N=0$
    overflow($v=1$) and $N=1$

  $\therefore$ $N'V' + NV = 1$, i.e., $N\oplus V=0$

2. **signed** number (cont.)

- A >= B case

1) Without overflow (V = 0)

A B result

(+) - (+) → (+)

(-) - (-) → (+) (=> N = 0)

2) With overflow (V = 1)

- Among overflow cases list below, results are positive when N = 1

(+) + (+) → (-)

(-) + (-) → (+)

(+) - (-) → (-)

(-) - (+) → (+)

2. **signed** number (cont.)

- $A<B$ → inverse of $A \geq B$

  no overflow and $N=1$

  overflow and $N=0$

  $\therefore$ $N'V + NV' = 1$, i,e., $N \oplus V = 1$

- $A>B$ → $A \geq B$ and $A \neq B$

  $\therefore$ $N \oplus V = 0$ and $Z=0$

- $A \leq B$ → $A<B$ or $A=B$

  $\therefore$ $N \oplus V = 1$ or $Z=1$

Example

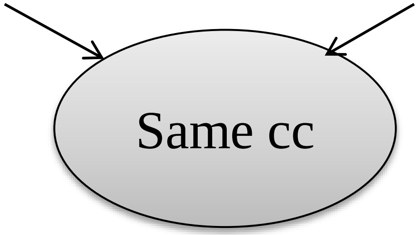
- Branching two different pieces of code

1)

```
mov -3, %l1
cmp %l1, 9
blu    less
nop
```

2)

```
mov -3, %l1
cmp %l1, 9
bl     less
nop
```



Same cc

- Why?

%l1: 11 … … 111101

1) If signed number, -3 $<$ **9**

2) If unsigned number, $2^{31} + 2^{30} + \ldots + 2^{2} + 2^{0}$ $>$ **9**

Branch inst. using individual CC bits

opcode	CC	equivalent inst.
bneg	N = 1	
bpos	N = 0	
bz	Z = 1	be
bnz	Z = 0	bne
bvs	V = 1	
bvc	V = 0	
bcs	C = 1	blu
bcc	C = 0	bgeu

Why bz == be?

✓ (bne, bnz) (blu, bcs) (bgeu, bcc)

Branch instruction format

- ✓ op-code label

Bit index	31 30	29	28 25	24 22	21 0
Field	OP	annul	cond	OP-2	displacement

- ✓ OP : 00 OP-2: Table 4.6

- ✓ Example

```
loop: subcc  %l3, 1, %l3
      bg,a   loop
```

→

00 1 1010 010 1111111111111111111111