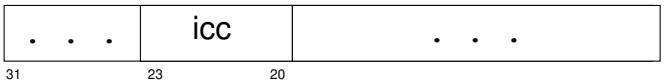
Branching

CS 217

Condition Codes

- Processor State Register (**PSR**)



- Integer condition codes (icc)
  - N **set if the last ALU result was negative**
  - Z **set if the last ALU result was zero**
  - V **set if the last ALU result was overflowed**
  - C **set if the last ALU instruction that modified the icc caused a carry out of, or a borrow into, bit 31**

Condition Codes (cont)

- **cc** versions of the integer arithmetic instructions set all the codes

```
addcc src1, src2, dst
subcc src1, src2, dst
```

- **cc** versions of the logical instructions set only **N** and **Z** bits

```
andcc src1, src2, dst
orc   src1, src2, dst
```

Compare and Test Instructions

- Synthetic instructions can set condition codes

Synthetic	Implementation
`tst reg`	`orcc reg,%g0,%g0`
`cmp src1,src2`	`subcc src1,src2,%g0`
`cmp src,value`	`subcc src,value,%g0`

Using `%g0` as the destination discards the result

Carry and Overflow

- If the carry bit is set
  - the last addition resulted in a carry, or
  - the last subtraction resulted in a borrow
  - Used for multi-word addition

```
addcc %g3,%g5,%g7
addxcc %g2,%g4,%g6
```

the most significant word is in the even register

(%g6,%g7) = (%g2,%g3) + (%g4,%g5)

- If the overflow bit is set
  - result of subtraction (or signed-addition) doesn't fit

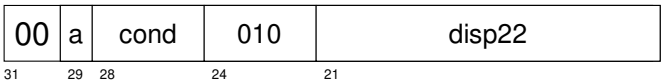
Branches

- Tests on the condition codes implement conditional branches and loops

```
If (a == 0)
   a = 1;
else
   a = 2;
```

Branch Instructions

- Transfer control based on **icc**
  - b [**a** | **n** | . . | **z**] {,a} ***label***



  - target is a PC-relative address: PC + 4 x disp22
  - where PC is the address of the branch instruction

Branch Instructions (cont)

- Unconditional branches (and synonyms)
  - ba jmp **branch always**
  - bn nop **branch never**
- Raw condition-code branches
  - bnz !Z
  - bz Z
  - bpos !N
  - bneg N
  - bcc !C
  - bcs C
  - bvc !V
  - bvs V

Branching Instructions (cont)

- Comparisons

instruction	signed	unsigned
be	Z	Z
bne	!Z	!Z
bg bgu	!(Z | (N^V))	!(C | Z))
ble bleu	Z | (N^V)	C | Z
bge bgeu	!(N^V)	!C
bl blu	N^V	C

Control Transfer

- Instructions normally fetched and executed from sequential memory locations
- PC is the address of the current instruction, and nPC is the address of the next instruction (nPC = PC + 4)
- Branches and control transfer instructions change nPC to something else

Control Transfer (cont)

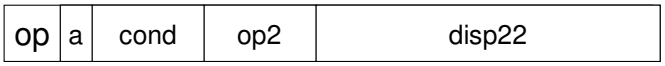
- Control transfer instructions

instruction	type	addressing mode
b*icc*	conditional branch	PC-relataive
jmpl	jump and link	register indirect
rett	return from trap	register indirect
call	procedure call	PC-relative
t*icc*	traps	register indirect (vectored)

PC-relative addressing is like register displacement addressing that uses the PC as the base register

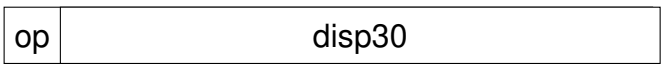
Control Transfer (cont)

- Branch instructions



nPC = PC + signextend(disp22) << 2

- Calls



nPC = PC + signextend(disp30) << 2

position-independent code does not depend on where it's loaded; uses PC-relative addressing

Branching Examples

- if-then-else

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

```
        #define a %l0
        #define b %l1
        #define c %l2
        cmp a,b
        ble L1; nop
        mov a,c
        ba L2; nop
L1:     mov b,c
L2:     ...
```

Branching Examples (cont)

- Loops

```
for (i=0; i<n; i++)
    . . .
```

```
        #define i %l0
        #define n %l1
        clr i
L1:     cmp i,n
        bge L2; nop
        . . .
        inc i
        ba L1; nop
L2:
```

Branching Examples (cont)

- Alternative implementation

for (i=0; i<n; i++)

```
    . . .
```

```
        #define i %l0
        #define n %l1
        clr i
        ba L2; nop
L1:     . . .
        inc i
L2:     cmp i,n
        bl L1; nop
```