# Multiply/Divide

```
mult   rs, rt  # start signed multiply
multu  rs, rt  # start unsigned multiply
               # lo, bottom 32 bits
div    rs, rt  # start signed divide
divu   rs, rt  # start unsigned divide
               # lo = quotient, hi = remainder
mfhi   rd      # rd = hi
mflo   rd      # rd = lo

mthi   rs      # hi = rs
mtlo   rs      # lo = rs
```

Special registers used to handle 64-bit result.

# Arithmetic Instructions

## C Code

```
int a, b, c, d;
a = b - (2*c + 7);
c = (a < 0) ?  1 :  0;
```

## Assembly

```
# a in reg 16, b in reg 17, c in reg 18
 addu   $8, $18, $18  # temp = 2*c
 addiu  $8, $8, 7     # temp = temp + 7
 subu   $16, $17, $8  # a = b - temp = b - (2*c+7)
 slt    $18, $16, $0  # c = (a < 0)
```

# Logical Operations

AND: both bits must be 1 (C operator &)

```
  0  1  0  1
  0  0  1  1
-------------
  0  0  0  1
```

OR: either bit is 1 (C operator |)

```
  0  1  0  1
  0  0  1  1
-------------
  0  1  1  1
```

# Logical Operations

eXclusive OR: bits must be different (C operator ^)

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |

NOR: OR followed by NOT

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |

# Logical Operations

```
and   rd, rs, rt      # rd = rs & rt
andi  rt, rs, imm     # rt = rs & imm
nor   rd, rs, rt      # rd = ~(rs | rt)
lui   rt, imm         # rt = imm << 16
or    rd, rs, rt      # rd = rs | rt
ori   rt, rs, imm     # rt = rs | imm
sll   rd, rt, shamt   # rd = rt << shamt
sllv  rd, rt, rs      # rd = rt << (rs&0x1f)
sra   rd, rt, shamt   # rd = rt >>s shamt
srav  rd, rt, rs      # rd = rt >>s (rs&0x1f)
srl   rd, rt, shamt   # rd = rt >> shamt
srlv  rd, rt, rs      # rd = rt >> (rs&0x1f)
xor   rd, rs, rt      # rd = rs ^ rt
xori  rt, rs, imm     # rt = rs ^ imm
```

# How Do I ...

## Load a 16-bit constant?

```
addiu $8, $0, value
```

## Load a 32-bit constant?

```
lui $8, (value >> 16)
ori $8, $8, (value & 0xffff)
```

## Why no subiu?

```
addiu $8, $9, (-value)
```

## Move from one register to another?

```
or $8, $9, $0
```

# How Do I ...

## Negate a register?

```
subu $8, $0, $8
```

## Complement a register?

```
nor $8, $8, $0
```

## Check for equality?

```
xor $8, $16, $17
sltiu $8, $8, 1
```

# How Do I ...

## Calculate absolute value of a register?

```
sra $9,$8,31
xor $8,$9,$8
andi $9,$9,1
addu $8,$9,$8
```

# Control Flow

Instructions that modify the pc. They specify:

- New pc value
  - pc-relative address
  - absolute address
- When is pc modified?
  - unconditional, equality between registers, etc.
- Save current pc?
  - Used for function calls
  - More later

# Control Flow: Branches

Branch instructions:

```
beq     rs, rt, imm  # if (rs==rt) goto L
bgez    rs, imm      # if (rs>=0) goto L
bgezal  rs, imm      # link; if (rs>=0) goto L
bgtz    rs, imm      # if (rs>0) goto L
blez    rs, imm      # if (rs<=0) goto L
bltz    rs, imm      # if (rs<0) goto L
bltzal  rs, imm      # link; if (rs<0) goto L
bne     rs, rt, imm  # if (rs!=rt) goto L
```

pc-calculation: $\mathtt{L} = \mathtt{pc} + 4 + (\mathtt{s\_ext}(\mathtt{imm}) << 2)$

link: $31=pc+8

# Control Flow: Jumps

Jump instructions:

```
j     tgt      # goto target
jal   tgt      # link; goto target
jalr  rs, rd   # rd=pc+8; goto rs
jr    rs       # goto rs
```

pc-calculation:

$$\texttt{target} = ((\texttt{pc} + 4)\&\texttt{0xf0000000})|(\texttt{tgt} << 2)$$


Jumps: absolute; Branches: pc-relative

# How Do I...

## Branch if register is zero?

```
beq $9,$0, L
```

## Unconditional branch?

```
beq $0, $0, L
```

## Branch if one register is smaller than another?

```
slt $8, $16,$17
bne $8, $0, L
```

## Jump to a 32-bit address?

# How Do I...

## Jump to an offset > 16 bits?

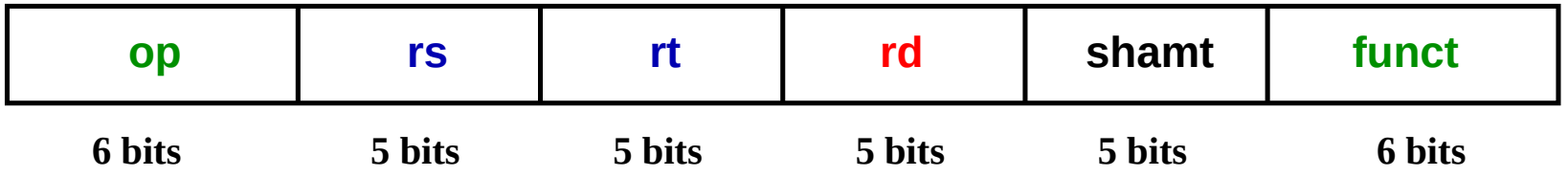
```
        bltzal $0, next
next:   li $8, adjoffset
        addu $8,$8,$31
        jr $8
```

## Wait, what is `li`?

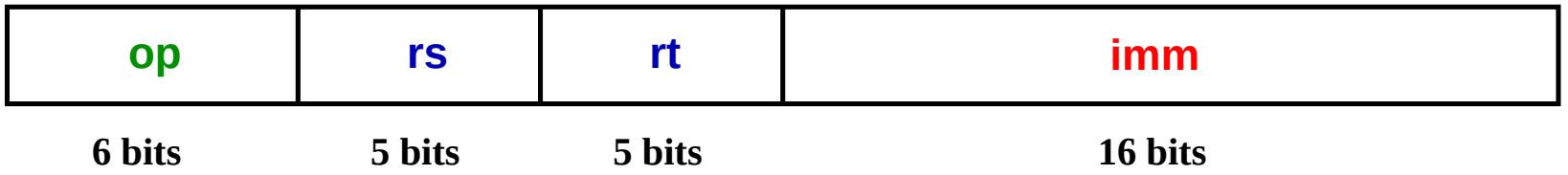
A pseudo-operation. Gets translated to either `addiu`, or `lui` and `ori`

# Summary: Instruction Formats

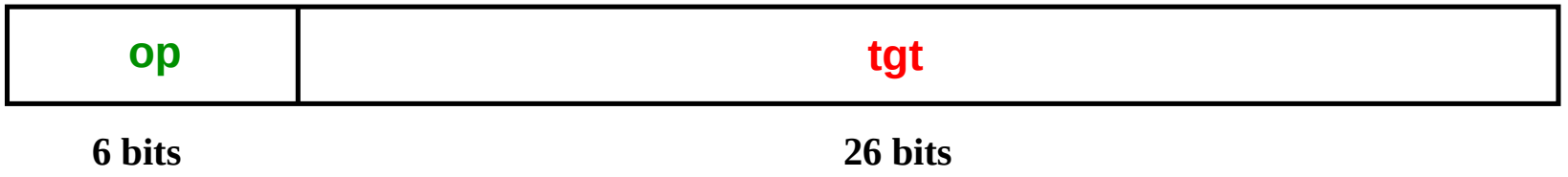
R-format:




I-format:




J-format:




# Translating Conditional Branches

## C code

```
int a, b;
if (a >= b) { a = 0; }
b += a;
```

## Assembly

```
# assume $16=a, $17=b
        slt $8,$16,$17    # set if a < b
        bne $8,$0,$skip   # if 1, then branch
        li $16,0          # set a to zero
$skip:  addu $17,$17,$16  # b += a
```

CSL

# Translating Loops

## C code

```
int i, j;
for (i=0; i < 10; i++)
   j += i;
j++;
```

First step toward assembly: remove structure (yikes!)

# Translating Loops

## Modified C code

```
            i = 0;
    loop:   if (!(i < 10)) goto finished;
            j += i;
            i++;
            goto loop;
finished:   j++;
```

# Translating Loops

## Minor Optimization

```
        i = 0;
loop:   j += i;
        i++;
        if (i < 10) goto loop;
        j++;
```

## Assembly (i in $16, j in $17)

```
        li $16,0            # set i = 0
$loop:  addu $17,$17,$16    # j += i
        addiu $16,$16,1     # i++
        slti $8,$16,10      # compare i < 10
        bne $8,$0, $loop    # and branch
        addiu $17,$17,1     # j++
```

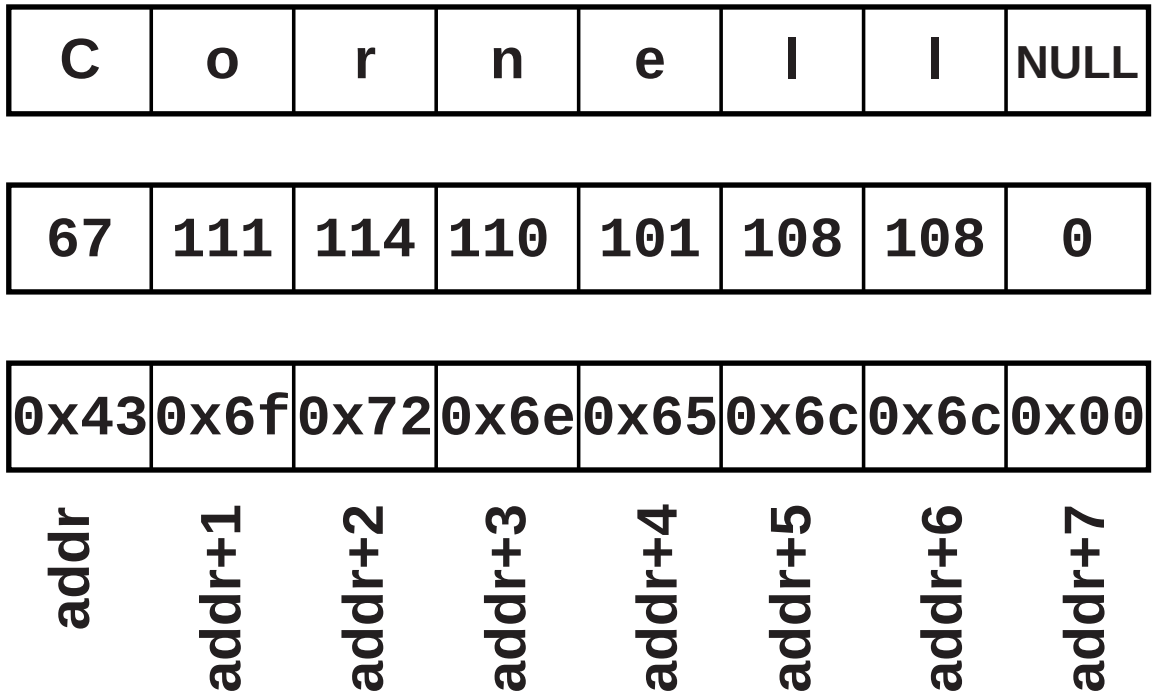
# Characters And Strings

Characters are stored as bytes (8 bits, ASCII)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | SPC | 48 | 0 | 64 | @ | 80 | P | 96 | ‘ | 112 | p |
| 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| ... | | ... | | ... | | ... | | ... | | ... | |
| 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | DEL |

# Characters And Strings

String: variable-length character array, terminated by NULL (byte 0)




# String Search

Problem: find the number of spaces in a string.

C code

```
/* s contains the address of the string */
count = 0;
while (*s) {
   if (*s++ == ' ') count++;
}
/* count contains the number of spaces */
```

First step: remove structure.

# String Search

## Modified C code

```
            count = 0;
   start:   if (!(*s)) goto done;
            if (!(*s == ' ')) goto skipinc;
            count++;
 skipinc:   s++;
            goto start;
    done:   ...
```

# String Search

Assembly count in $16, s in $17

```
            li $16,0              # count = 0
  $start:   lbu $8, 0($17)        # temp = *s
            beq $8,$0,$done       # if *s == 0 goto done
            li $9,32              # temp2 = ' '
            bne $8,$9,$skipinc    # if *s != ' '
                                  #  goto skipinc
            addiu $16,$16,1       # count++
 $skipinc:  addiu $17,$17,1       # s++
            beq $0,$0,$start      # goto start
    $done:  ...
```