### Jump Instructions

<table>
<thead>
<tr>
<th>assem.</th>
<th>meaning</th>
<th>bit pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>j label</td>
<td>pc = pc &amp; 0xF0000000</td>
<td>00010xxxxxxxxxxxxxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>jal label</td>
<td>rs1 = pc + 4; pc = pc &amp; 0xF0000000</td>
<td>000011xxxxxxxxxxxxxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>jr rs</td>
<td>pc = rs</td>
<td>000000asas000000000000000001000</td>
</tr>
<tr>
<td>jalr rs</td>
<td>rs1 = pc + 4; pc = rs</td>
<td>000000asas000000000000000001001</td>
</tr>
</tbody>
</table>

- Jump instruction **unconditionally** transfer execution to a new location.
- **spim** will calculate correct value for \( X \) from location of label in code.
- jal & jalr set \( rs1 (\$ra) \) to address of the next instruction.
  - Used for function calls.
  - Return can then be implemented with jr \( \$ra \).

### Branch Instructions

<table>
<thead>
<tr>
<th>assem.</th>
<th>meaning</th>
<th>bit pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>b label</td>
<td>pc += I &lt;&lt; 2</td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td>beq rs,rt,label</td>
<td>if ( (rs == rt) ) pc += I &lt;&lt; 2</td>
<td>000100asas00000000000000001000</td>
</tr>
<tr>
<td>bne rs,rt,label</td>
<td>if ( (rs != rt) ) pc += I &lt;&lt; 2</td>
<td>000101asas00000000000000001000</td>
</tr>
<tr>
<td>ble rs,rt,label</td>
<td>if ( (rs &lt;= rt) ) pc += I &lt;&lt; 2</td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td>bgt rs,rt,label</td>
<td>if ( (rs &gt; rt) ) pc += I &lt;&lt; 2</td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td>blt rs,rt,label</td>
<td>if ( (rs &lt; rt) ) pc += I &lt;&lt; 2</td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td>bge rs,rt,label</td>
<td>if ( (rs &gt;= rt) ) pc += I &lt;&lt; 2</td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td>blez rs,label</td>
<td>if ( (rs &lt;= 0) ) pc += I &lt;&lt; 2</td>
<td>000010asas00000000000000001000</td>
</tr>
<tr>
<td>bgtz rs,label</td>
<td>if ( (rs &gt; 0) ) pc += I &lt;&lt; 2</td>
<td>000011asas00000000000000001000</td>
</tr>
<tr>
<td>bltz rs,label</td>
<td>if ( (rs &lt; 0) ) pc += I &lt;&lt; 2</td>
<td>000000asas00000000000000001000</td>
</tr>
<tr>
<td>bgez rs,label</td>
<td>if ( (rs &gt;= 0) ) pc += I &lt;&lt; 2</td>
<td>000001asas00000000000000001000</td>
</tr>
</tbody>
</table>

- Branch instruction **conditionally** transfer execution to a new location.
- spim will calculate correct value for \( I \) from location of label in code.
- spim allows second operand (\( rt \)) to be replaced by a constant.

### Example Translation of Branch Pseudo-instructions

<table>
<thead>
<tr>
<th>Pseudo-Instructions</th>
<th>Real Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>bge $t1, $t2, label</td>
<td>sll $at, $t1, $t2</td>
</tr>
<tr>
<td>blt $t1, $t2, label</td>
<td>sll $at, $t1, $t2</td>
</tr>
</tbody>
</table>

### goto in C

The goto statement allows transfer of control to any labelled point with a function. For example, this code:

```c
for (int i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
```

Can be written as:

```c
int i = 1;
loop:
    if (i > 10) goto end;
    i++;
    printf("%d", i);
    printf("\n");
goto loop;
end:
```


goto in C

- goto statements can result in very difficult to read programs.
- goto statements can also result in slower programs.
- In general, use of goto is considered bad programming style.
- Do not use goto without very good reason.
- kernel & embedded programmers sometimes use goto.

MIPS Programming

Writing correct assembler directly is hard.

Recommended strategy:
- develop the solution in C
- map to “simplified” C
- translate each simplified C statement to MIPS instructions

Simplified C

- does not have while, compound if, complex expressions
- does have simple if, goto, one-operator expressions

Simplified C makes extensive use of
- labels ... symbolic name for C statement
- goto ... transfer control to labelled statement

Example:

```
C
int main(void) {
    int x = 17;
    int y = 25;
    printf("%d\n", x + y);
    return 0;
}
```

Simplified C

```
int main(void) {
    int x, y, z;
    x = 17;
    y = 25;
    z = x + y;
    printf("%d", z);
    printf("\n");
    return 0;
}
```

Mapping C into MIPS

Things to do:
- allocate variables to registers/memory
- place literals in data segment
- transform C program to:
  - break expression evaluation into steps
  - replace control structures by goto

```
adding 2 numbers: C to simplified C

C
int main(void) {
    int x = 17;
    int y = 25;
    printf("%d\n", x + y);
    return 0;
}
```

Simplified C

```
int main(void) {
    int x, y, z;
    x = 17;
    y = 25;
    z = x + y;
    printf("%d", z);
    printf("\n");
    return 0;
}
```
adding 2 numbers: simplified C to MIPS

Simplified C

```c
int x, y, z;
x = 17;
y = 25;
z = x + y;
printf("%d", z);
```

MIPS

```mips
# add 17 and 25 and print result
main:
    # x,y,z in $t0,$t1,$t2,
    li $t0, 17
    # x = 17;
    li $t1, 25
    # y = 25;
    add $t2, $t1, $t0
    move $a0, $t2
    move $a0, $t2
    # z = x + y
    add $t1, $t0
    # printf("%d", z);
    li $v0, 1
    syscall
    li $v0, 0
    # printf("\n");
    syscall
    jr $ra
```

while loop - converting C to simplified C

Standard C

```c
i = 0;
n = 0;
while (i < 5) {
    n = n + i;
i++;
}
```

Simplified C

```c
i = 0;
n = 0;
loop:
if (i >= 5) goto end;
n = n + i;
goto loop;
end:
```

if - converting C to simplified C

Standard C

```c
if (i < 0) {
n = n - i;
} else {
n = n + i;
}
```

Simplified C

```c
if (i >= 0) goto else1;
n = n - i;
goto end1;
else1:
n = n + i;
end1:
```

• note else can’t be used a a label in C
### if - converting simplified C to MIPS

**Simplified C**

```c
if (i >= 0) goto else1;
n = n - i;
goto end1;
else1:
n = n + i;
end1:
```

**MIPS**

```mips
# assume i in $t0
# assume n in $t1
bge $t0, 0, else1
sub $t1, $t1, $t0
goto end1
else1:
    add $t1, $t1, $t0
end1:
```

### if/and: C to simplified C

**Standard C**

```c
if (i < 0 && n >= 42) {
n = n - i;
} else {
n = n + i;
}
```

**Simplified C**

```c
if (i >= 0) goto else1;
if (n < 42) goto else1;
n = n - i;
goto end1;
else1:
n = n + i;
end1:
```

### if/and: simplified C to MIPS

**Simplified C**

```c
if (i >= 0) goto else1;
if (n < 42) goto else1;
n = n - i;
goto end1;
else1:
n = n + i;
end1:
```

**MIPS**

```mips
# assume i in $t0
# assume n in $t1
bge $t0, 0, else1
blt $t1, 42, else1
sub $t1, $t1, $t0
j end1
else1:
    add $t1, $t1, $t0
end1:
```

### odd-even: C to simplified C

**Standard C**

```c
if (i < 0 || n >= 42) {
n = n - i;
} else {
n = n + i;
}
```

**Simplified C**

```c
if (i < 0) goto then1;
if (n >= 42) goto then1;
goto else1;
then1:
n = n - i;
goto end1;
else1:
n = n + i;
end1:
```
C

```c
int main(void) {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

Simplified C

```c
int main(void) {
    int i;
    i = 1;
    loop:
        if (i > 10) goto end;
        i++;
        printf("%d\n", i);
        goto loop;
    end:
    return 0;
}
```

Odd or Even: C to simplified C

C

```c
int main(void) {
    int x;
    printf("Enter a number: ");
    scanf("%d", &x);
    if ((x & 1) == 0) {
        printf("Even\n");
    } else {
        printf("Odd\n");
    }
    return 0;
}
```

Simplified C

```c
int main(void) {
    int x, v0;
    printf("Enter a number: ");
    scanf("%d", &x);
    v0 = x & 1;
    if (v0 == 1) goto odd;
    printf("Odd\n");
    goto end;
    odd:
        printf("Odd\n");
    end:
    return 0;
}
```

Odd or Even: MIPS

```mips
main:
    li $t0, 1
    $i = 1;
loop:
    bgt $t0, 10, end
    move $a0, $t0
    syscall
    li $a0, 0
    printf("Even\n");
    j loop
end:
    li $v0, 0
    return 0
jr $ra
```

# print integers 1..10 one per line
main:
    li $t0, 1
    $i = 1;
loop:
    bgt $t0, 10, end
    move $a0, $t0
    syscall
    li $v0, 0
    printf("Even\n");
    j loop
end:
    li $v0, 0
    return 0
jr $ra
```

# read a number and print whether its odd or even
main:
    la $a0, string0
    printf("Enter a number: ");
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    and $t0, $v0, 1
    beq $t0, 1, odd
    la $a0, string1
    printf("Even\n");
    li $v0, 4
    syscall
    j end
```

Odd or Even: MIPS
Odd or Even: MIPS

```mips
odd:
    # else
    la $a0, string2  # printf("Odd\n");
    li $v0, 4
    syscall
end:
    li $v0, 0  # return 0
    jr $ra
.string0:
    .asciiz "Enter a number: "
.string1:
    .asciiz "Even\n"
.string2:
    .asciiz "Odd\n"
```

Source code for `odd_even.s`

---

Sum 100 Squares: MIPS

```mips
main:
    li $t0, 0  # sum = 0;
    li $t1, 0  # i = 0
loop:
    bgt $t1, 100, end  # if (i > 100) goto end;
    mul $t2, $t1, $t1  # square = i * i;
    add $t0, $t0, $t2  # sum = sum + square;
    addi $t1, $t1, 1  # i = i + 1;
    j loop
end:
    move $a0, $t0  # printf("%d", sum);
    li $v0, 1
    syscall
    li $a0, /
    li $v0, 11
    syscall
    li $v0, 0  # return 0
    jr $ra
```

Source code for `sum_100_squares.s`

---

Sum 100 Squares: C to simplified C

**C**

```c
int main(void) {
    int sum = 0;
    for (int i = 0; i <= 100; i++) {
        sum += i * i;
    }
    printf("%d\n", sum);
    return 0;
}
```

Source code for `sum_100_squares.c`

**Simplified C**

```c
int main(void) {
    int i, sum, square;
    sum = 0;
    i = 0;
    loop:
        if (i > 100) goto end;
        square = i * i;
        sum = sum + square;
        i = i + 1;
        goto loop;
    end:
        printf("%d", sum);
        printf("\n");
        return 0;
}
```

Source code for `sum_100_squares.simple.c`