Jump Instructions

<table>
<thead>
<tr>
<th>assembler</th>
<th>meaning</th>
<th>bit pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>j label</td>
<td>pc = pc &amp; 0xF0000000</td>
<td>000010XXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>jal label</td>
<td>ra = pc + 4; pc = pc &amp; 0xF0000000</td>
<td>000011XXXXXXXXXXXXXXXXXXXXXXXXXX</td>
</tr>
<tr>
<td>jr r_s</td>
<td>pc = r_s</td>
<td>000000ssss0000000000000000001000</td>
</tr>
<tr>
<td>jalr r_s</td>
<td>ra = pc + 4; pc = r_s</td>
<td>000000ssss0000000000000000001001</td>
</tr>
</tbody>
</table>

- Jump instructions **unconditionally** transfer execution to a new location
  - In other words, jump instructions change the PC (program counter)
- For `j label` and `jal label` MIPSy calculates correct value for `X` from location of `**label` in code
- `jal` & `jalr` set `$ra` ($31) to address of the next instruction
  - Call to function `f` implemented by `jal f`
  - Return can then be implemented with `jr $ra`
- `jr` & `jalr` can be used with any register
  - Used to implement function pointer derefencing in C, and methods in object-oriented languages

Branch Instructions

<table>
<thead>
<tr>
<th>b label</th>
<th>pc += I«2</th>
<th>pseudo-instruction</th>
</tr>
</thead>
</table>
| beq r_s, r_t, label | if \((r_s == r_t)\) pc += I«2 | 000100ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss ss s

- Branch instructions **conditionally** transfer execution to a new location (except `b` is unconditional)
- **Mipsy** will calculate correct value for `f` from location of `label` in code
- **Mipsy** allows second operand (`r_t`) to be replaced by a constant (fine to use in COMP1521)
Example Translation of Branch Pseudo-instructions

<table>
<thead>
<tr>
<th>Pseudo-Instructions</th>
<th>Real Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bge $t1, $t2, label</code></td>
<td><code>slt $at, $t1, $t2</code></td>
</tr>
<tr>
<td></td>
<td><code>beq $at, $0, label</code></td>
</tr>
<tr>
<td><code>blt $t1, 42, label</code></td>
<td><code>addi $at, $zero, 42</code></td>
</tr>
<tr>
<td></td>
<td><code>slt $at, $t1, $at</code></td>
</tr>
<tr>
<td><code>beqz $t3, label</code></td>
<td><code>bne $at, $0, label</code></td>
</tr>
<tr>
<td><code>bnez $t4, label</code></td>
<td><code>bne $t4, $0, label</code></td>
</tr>
<tr>
<td><code>b label</code></td>
<td><code>beq $0, $0, label</code></td>
</tr>
</tbody>
</table>

Branch versus Jump

- jump instructions are unconditional
- branch instructions are conditional and can implement if and while
  - except `b label` which has same effect as `j label`
  - you can use either
- `jal` and `jr` instructions provide a simple function call & return implementations
- no equivalent branch instructions
- branch instruction encode a 16-bit relative offset
  - target (label) must be within -32768..32767 instructions
  - not a problem in COMP1521 - we write small programs
- jump instruction encode a 28-bit value
  - allows jumps to be used for targets (labels) further away

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 a solution in C
- map down to "simplified" C
- translate simplified C statements 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

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 most control structures by goto
Conditionals — if from C to Simplified C

Standard C
if (i < 0) {
    n = n - i;
} else {
    n = n + i;
}

Simplified C
if (i >= 0) goto else1;
n = n - i;
goto end1;
else1:
n = n + i;
end1:

note: else is not a valid label name in C

Conditionals — if from Simplified C to MIPS

Simplified C
if (i >= 0) goto else1;
n = n - i;
goto end1;
else1:
n = n + i;
end1:

MIPS
# assuming i in $t0,
# assuming n in $t1...
bge $t0, 0, else1
sub $t1, $t1, $t0
goto end1
else1:
add $t1, $t1, $t0
end1:

Print If Even: C to simplified C

C
int main(void) {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (n % 2 == 0) {
        printf("even\n");
    }
    return 0;
}

Simplified C
int main(void) {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (n % 2 != 0) goto epilogue;
    printf("even\n");
    epilogue:
    return 0;
}
# Print a message only if a number is even.
# Written by: Abiram Nadarajah <abiramn@cse.unsw.edu.au>
# Written as a COMP1521 lecture example

.text
main:
    # Locals:
    # - $t0: int n
    # - $t1: n % 2
    li $v0, 4  # syscall 4: print_string
    la $a0, prompt_msg  # printf("Enter a number: ");
    syscall  # syscall 4: print_string
    li $v0, 5  # syscall 5: read_int
    syscall  # scanf("%d", &n);
    rem $t1, $t0, 2  # if ((n % 2)
    bnez $t1, epilogue  # != 0) goto epilogue;
    li $v0, 4  # syscall 4: print_string
    la $a0, even_msg  # printf("even\n");
    syscall  # syscall 4: print_string

epilogue:
    li $v0, 0  # return 0;
    jr $ra  # return 0;

.data
prompt_msg:
    .asciiz "Enter a number: ";

even_msg:
    .asciiz "even\n"

Odd or Even: C to simplified C

C
int main(void) {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (n % 2 == 0) {
        printf("even\n");
    } else {
        printf("odd\n");
    }
    return 0;
}

Simplified C
int main(void) {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (n % 2 != 0) goto n_mod_2_ne_0;
    printf("even\n");
    goto epilogue;
    n_mod_2_ne_0:
    printf("odd\n");
    epilogue:
    return 0;
}

source code for odd_even.c
source code for odd_even.simple.c
Odd or Even: MIPS

# Print out whether a value is odd or even.
# Written by: Abiram Nadarajah <abiramn@cse.unsw.edu.au>
# Written as a COMP1521 lecture example

.text
main:
    # Locals:
    # - $t0: int n
    # - $t1: n % 2
    li $v0, 4       # syscall 4: print_string
    la $a0, prompt_msg   #
    syscall           # printf("Enter a number: ");
    li $v0, 5       # syscall 5: read_int
    syscall         #
    move $t0, $v0    # scanf("%d", &n);
    rem $t1, $t0, 2  # if ((n % 2)
    bnez $t1, n_mod_2_ne_0   # != 0) goto n_mod_2_ne_0;

.n_mod_2_ne_0:
    li $v0, 4       # syscall 4: print_string
    la $a0, even_msg #
    syscall         # printf("even\n");

n_mod_2_ne_0:   # goto epilogue
    li $v0, 4       # syscall 4: print_string
    la $a0, odd_msg #
    syscall        # printf("odd\n");

.epilogue:
    li $v0, 0       #
    jr $ra           # return 0;

.data
prompt_msg:
    .asciiz "Enter a number: ">

even_msg:
    .asciiz "even\n"

odd_msg:
    .asciiz "odd\n"
Loops — while from Simplified C to MIPS

**Simplified C**

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

**MIPS**

```mips
li $t0, 0  # i in $t0
li $t1, 0  # n in $t1
loop:
bge $t0, 5, end
add $t1, $t1, $t0
addi $t0, $t0, 1
j loop
end:
```

---

**Printing First 10 Integers: C to simplified C**

**C**

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

**Simplified C**

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

---

**Printing First 10 Integers: MIPS**

```mips
# print integers 1..10 one per line
main:  # int main(void) {
    li $t0, 1  # i = 1;
loop:  # loop:
    bgt $t0, 10, end  # if (i > 10) goto end;
    move $a0, $t0  # printf("%d" i);
    li $v0, 1
    syscall
    li $a0, '\n'  # printf("\n");
    li $v0, 11
    syscall
    addi $t0, $t0, 1  # i++;
    b loop  # goto loop;
end:  # end:
i $v0, 0  # return 0
jr $ra
```

---

https://www.cse.unsw.edu.au/~cs1521/23T3/ COMP1521 23T3 — MIPS Control 19 / 32

---

https://www.cse.unsw.edu.au/~cs1521/23T3/ COMP1521 23T3 — MIPS Control 20 / 32

---

https://www.cse.unsw.edu.au/~cs1521/23T3/ COMP1521 23T3 — MIPS Control 21 / 32

---

https://www.cse.unsw.edu.au/~cs1521/23T3/ COMP1521 23T3 — MIPS Control 22 / 32
### Sum 100 Squares: C to simplified C

**C**

```c
int main(void) {
    int sum = 0;
    for (int i = 1; i <= 100; i++) {
        sum += i * i;
    }
    printf("%d\n", sum);
    return 0;
}
```

**Simplified C**

```c
int main(void) {
    int sum = 0;
    loop_i_to_100__init:
        int i = 0;
    loop_i_to_100__cond:
        if (i > UPPER_BOUND) goto loop_i_to_100__end;
    loop_i_to_100__body:
        sum += i * i;
    loop_i_to_100__step:
        i++;
        goto loop_i_to_100__cond;
    loop_i_to_100__end:
        printf("%d", sum);
        putchar('
');
        return 0;
}
```

### Sum 100 Squares: MIPS

```
# Calculate 1*1 + 2*2 + ... + 99*99 + 100*100
# Written by: Abiram Nadarajah <abiramn@cse.unsw.edu.au>
# Written as a COMP1521 lecture example
UPPER_BOUND = 100

.text
main:
    # Locals:
    # - $t0: int sum
    # - $t1: int i
    # - $t2: temporary value
    li $t0, 0  # int sum = 0;
loop_i_to_100__init:
    li $t1, 1  # int i = 0;
loop_i_to_100__cond:
    bgt $t1, UPPER_BOUND, loop_i_to_100__end  # while (i < UPPER_BOUND) {
loop_i_to_100__body:
    mul $t2, $t1, $t1  # sum = (i * i) +
    add $t0, $t0, $t2  # ... 0;
    loop_i_to_100__step:
        addi $t0, $t0, 1  # i++;
        b loop_i_to_100__cond  # }
loop_i_to_100__end:
    li $v0, 1  # syscall 1: print_int
    move $a0, $t0
    syscall  #
    syscall  # printf("%d", sum);
    li $v0, 11  # syscall 11: print_char
    li $a0, '\n'
    syscall  #
    li $v0, 0  # putchar('\n');
    jr $ra  # return 0;
```
**Conditionals — if and &&: from 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;
else1:
    n = n - i;
goto end1;
else1:
    n = n + i;
end1:
```

**Conditionals — if and ||: from 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;
else1:
    n = n - i;
goto end1;
then1:
    n = n + i;
goto end1;
else1:
    n = n + i;
end1:
```

**Conditionals — if and ||: from Simplified C to MIPS**

**Simplified C**

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

**Conditionals — if and ||: from 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;
else1:
    n = n - i;
goto end1;
then1:
    n = n + i;
goto end1;
else1:
    n = n + i;
end1:
```
### Conditionals — if and ||: from Simplified C to MIPS

#### 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:
```

#### MIPS

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

---

#### The break statement

Sometimes it is useful to exit from the middle of a loop

- **break** allows you to check a condition mid-loop and quit

```c
// read up to 100 characters
// stop if the next character is '!
while (i <= 100) {
    int ch = getchar();
    if (ch == '!') break;
    putchar(ch);
}
```

---

#### The continue statement

Sometimes it is useful to go to next iteration and skip rest of loop

- **continue** allows you to go to next iteration from mid-loop

```c
// iterate over integers 1..100
// skip every multiple of three
for (i = 1; i <= 100; i++) {
    if (i % 3 == 0) continue;
    printf("%d\n", i);
}
```
continue can simplify loops

```c
while (Condition) {
    some_code_1
    if (Condition1) {
        some_code_2
        if (Condition2) {
            some_code_3
        }
    }
}
```

Side Topic: C do/while

C has a different while loop - do/while (post-test).

- loop condition checked at bottom of loop - always executed once
- many programmers do not use it

```c
do {
    printf("%d\n", i);
    i++;
} while (i < 10);
```

can be written as:

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