COMP1521 22T2 — MIPS Control

https://www.cse.unsw.edu.au/~cs1521/22T1/
### Jump Instructions

<table>
<thead>
<tr>
<th>assembler</th>
<th>meaning</th>
<th>bit pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>j <em>label</em></td>
<td>pc = pc &amp; 0xF0000000</td>
<td>(X«2)</td>
</tr>
<tr>
<td>jal <em>label</em></td>
<td>( r_{31} = pc + 4; )</td>
<td>pc = pc &amp; 0xF0000000</td>
</tr>
<tr>
<td>jr ( r_s )</td>
<td>pc = ( r_s )</td>
<td>000000ssssss00000000000000000001000</td>
</tr>
<tr>
<td>jalr ( r_s )</td>
<td>( r_{31} = pc + 4; )</td>
<td>pc = ( r_s )</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 \( r_{31} \) ($ra) to address of the next instruction
  - Used for function calls
  - Return can then be implemented with jr $ra
<table>
<thead>
<tr>
<th>assembler</th>
<th>meaning</th>
<th>bit pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b label</code></td>
<td><code>pc += I«2</code></td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td><code>beq rs,rt,label</code></td>
<td><code>if (rs == rt) pc += I«2</code></td>
<td><code>000100sssssttttttIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII</code></td>
</tr>
<tr>
<td><code>bne rs,rt,label</code></td>
<td><code>if (rs != rt) pc += I«2</code></td>
<td><code>000101sssssttttttIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII</code></td>
</tr>
<tr>
<td><code>ble rs,rt,label</code></td>
<td><code>if (rs &lt;= rt) pc += I«2</code></td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td><code>bgt rs,rt,label</code></td>
<td><code>if (rs &gt; rt) pc += I«2</code></td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td><code>blt rs,rt,label</code></td>
<td><code>if (rs &lt; rt) pc += I«2</code></td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td><code>bge rs,rt,label</code></td>
<td><code>if (rs &gt;= rt) pc += I«2</code></td>
<td>pseudo-instruction</td>
</tr>
<tr>
<td><code>blez rs,label</code></td>
<td><code>if (rs &lt;= 0) pc += I«2</code></td>
<td><code>000110ssssss00000I II II II II II II II II II II II II II II II</code></td>
</tr>
<tr>
<td><code>bgtz rs,label</code></td>
<td><code>if (rs &gt; 0) pc += I«2</code></td>
<td><code>000111ssssss00000I II II II II II II II II II II II II II II II</code></td>
</tr>
<tr>
<td><code>bltz rs,label</code></td>
<td><code>if (rs &lt; 0) pc += I«2</code></td>
<td><code>000001ssssss00000I II II II II II II II II II II II II II II II</code></td>
</tr>
<tr>
<td><code>bgez rs,label</code></td>
<td><code>if (rs &gt;= 0) pc += I«2</code></td>
<td><code>000001ssssss00001I II II II II II II II II II II II II II II II</code></td>
</tr>
</tbody>
</table>

- A **branch instruction** **conditionally** transfers execution to a new location.
- **spim** will calculate the correct value for `I` from the location of `label` in code.
- **spim** allows the second operand (`rt`) to be replaced by a constant.
- Also, **`bnez`**, **`beqz`** pseudo-instructions.
# 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><code>blt  $t1, $t2, label</code></td>
<td><code>beq  $at, $0, label</code></td>
</tr>
<tr>
<td><code>b  label</code></td>
<td><code>slt  $at, $0, $0</code></td>
</tr>
<tr>
<td><code>beqz  $t3, label</code></td>
<td><code>beq  $0, $0, label</code></td>
</tr>
<tr>
<td><code>bnez  $t4, label</code></td>
<td><code>beq  $t3, $0, label</code></td>
</tr>
<tr>
<td></td>
<td><code>bne  $t4, $0, label</code></td>
</tr>
</tbody>
</table>
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 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.
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
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
Adding Two Numbers — C to Simple C

C

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

Simplified C

```c
int main(void) {
    int x, y, z;
    x = 17;
    y = 25;
    z = x + y;
    printf("%d", z);
    printf("\n");
    return 0;
}
```
Adding Two Numbers — Simple C to MIPS

Simplified C

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

MIPS

```mips
# add 17 and 25 and print result
main:
   li $t0, 17       # x = 17;
   li $t1, 25       # y = 25;
   add $t2, $t1, $t0 # z = x + y
   move $a0, $t2    # printf("%d", z);
   li $v0, 1
   syscall
   li $a0, '\n'     # printf("%c", '\n');
   li $v0, 11
   syscall
   li $v0, 0        # return 0
   jr $ra
```

source code: add.s
Loops — while from 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;
    i++; 
    goto loop;
end:
```
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:
```
Conditionals — *if* from 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` is not a valid label name in C
Conditionals — if from Simplified C to MIPS

**Simplified C**

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

**MIPS**

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

**Simplified 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: 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;
if (n >= 42) goto then1;
    goto else1;
then1:
    n = n - i;
    goto end1;
else1:
    n = n + i;
end1:
```
Printing First 10 Integers: C to simplified C

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 = 1;
    loop:
        if (i > 10) goto end;
        i++;
        printf("%d", i);
        printf("\n");
        goto loop;
    end:
        return 0;
}
```
# print integers 1..10 one per line

main:       # int main(void) {
    # int i;  // in register $t0
    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:        # return 0
    li   $v0, 0
    jr   $ra
Odd or Even: C to simplified C

C

```c
int main(void) {
    int x;
    printf("Enter a number: ");
    scanf("%d", &x);
    if (x % 2 == 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 % 2;
    if (v0 == 1) goto odd;
    printf("Even\n");
    goto end;
    odd:
    printf("Odd\n");
    end:
    return 0;
}
```
# read a number and print whether its odd or even

```mips
main:
    la $a0, string0       # printf("Enter a number: ");
    li $v0, 4
    syscall
    li $v0, 5            # scanf("%d", x);
    syscall
    rem $t0, $v0, 2      # if (x % 2 == 0) {
    beq $t0, 1, odd
    la $a0, string1      # printf("Even\n");
    li $v0, 4
    syscall
    b end
```
odd:
   la  $a0, string2  # else
   li  $v0, 4
   syscall
end:
   li  $v0, 0     # return 0
   jr  $ra
.data
string0:
   .asciiz "Enter a number: "
string1:
   .asciiz "Even\n"
string2:
   .asciiz "Odd\n"
int main(void) {
    int sum = 0;
    for (int i = 0; i <= 100; i++) {
        sum += i * i;
    }
    printf("%d\n", sum);
    return 0;
}

Simplified 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;
}
# calculate 1*1 + 2*2 + ... + 99 * 99 + 100 * 100
# sum in $t0, i in $t1, square in $t2

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;
    b loop

end:

source code for sum_100_squares
Sum 100 Squares: MIPS

end:
move $a0, $t0  # printf("%d", sum);
li   $v0, 1
syscall
li   $a0, '\n'  # printf("%c", '\n');
li   $v0, 11
syscall
li   $v0, 0   # return 0
jr    $ra

source code for sum_100_squares
Side Topic: C do/while

- C has a different while loop - do/while.
- Loop condition checked at bottom of loop executed - 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;
end:
```