

COMP1521 26T1 — MIPS Control

<https://www.cse.unsw.edu.au/~cs1521/26T1/>

Jump Instructions

assembler	meaning	bit pattern
j <i>label</i>	$pc = pc \& 0xF0000000 \mid (X \ll 2)$	000010XXXXXXXXXXXXXXXXXXXXXXXXXXXX
jal <i>label</i>	$ra = pc + 4;$ $pc = pc \& 0xF0000000 \mid (X \ll 2)$	000011XXXXXXXXXXXXXXXXXXXXXXXXXXXX
jr r_s	$pc = r_s$	000000sssss000000000000000001000
jalr r_s	$ra = pc + 4;$ $pc = r_s$	000000sssss000000000000000001001

- jump instructions **unconditionally** transfer execution to a new location
 - in other word, 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

b <i>label</i>	$pc += I \ll 2$	pseudo-instruction
beq $r_s, r_t, label$	if ($r_s == r_t$) $pc += I \ll 2$	000100ssssssttttIIIIIIIIIIIIIIIIII
bne $r_s, r_t, label$	if ($r_s != r_t$) $pc += I \ll 2$	000101ssssssttttIIIIIIIIIIIIIIIIII
ble $r_s, r_t, label$	if ($r_s \leq r_t$) $pc += I \ll 2$	pseudo-instruction
bgt $r_s, r_t, label$	if ($r_s > r_t$) $pc += I \ll 2$	pseudo-instruction
blt $r_s, r_t, label$	if ($r_s < r_t$) $pc += I \ll 2$	pseudo-instruction
bge $r_s, r_t, label$	if ($r_s \geq r_t$) $pc += I \ll 2$	pseudo-instruction
blez $r_s, label$	if ($r_s \leq 0$) $pc += I \ll 2$	000110sssss00000IIIIIIIIIIIIIIIIII
bgtz $r_s, label$	if ($r_s > 0$) $pc += I \ll 2$	000111sssss00000IIIIIIIIIIIIIIIIII
bltz $r_s, label$	if ($r_s < 0$) $pc += I \ll 2$	000001sssss00000IIIIIIIIIIIIIIIIII
bgez $r_s, label$	if ($r_s \geq 0$) $pc += I \ll 2$	000001sssss00001IIIIIIIIIIIIIIIIII
bnz $r_s, label$	if ($r_s != 0$) $pc += I \ll 2$	pseudo-instruction
beqz $r_s, label$	if ($r_s == 0$) $pc += I \ll 2$	pseudo-instruction

- branch instruction **conditionally** transfer execution to a new location (except **b** is unconditional)
- mipsy will calculate correct value for **I** 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

Pseudo-Instructions

bge \$t1, \$t2, label

blt \$t1, 42, label

beqz \$t3, label

bnez \$t4, label

b label

Real Instructions

slt \$at, \$t1, \$t2

beq \$at, \$0, label

addi \$at, \$zero, 42

slt \$at, \$t1, \$at

bne \$at, \$0, label

beq \$t3, \$0, label

bne \$t4, \$0, label

beq \$0, \$0, label

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 provides 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

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

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

can be written as:

```
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

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

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
b    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;  
}
```

[source code for print_if_even.c](#)

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;  
}
```

[source code for print_if_even.simple.c](#)

Print If Even: MIPS

```
# 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 #  
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, epilogue # != 0) goto epilogue;
```

Print If Even: MIPS

```
rem $t1, $t0, 2      # if ((n % 2)
bnez $t1, epilogue   #      != 0) goto epilogue;
li $v0, 4             # syscall 4: print_string
la $a0, even_msg      #
syscall              # printf("even\n");
epilogue:
li $v0, 0             #
jr $ra               # return 0;
.data
prompt_msg:
    .asciiz "Enter a number: "
even_msg:
    .asciiz "even\n"
```

[source code for print_if_even.s](#)

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;
}
```

[source code for odd_even.c](#)

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.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;
```


Odd or Even: MIPS

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

[source code for odd_even.s](#)

Loops — while from C to Simplified C

Standard C

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

Simplified 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

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

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

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

[source code for count_to_10.c](#)

Simplified C

```
loop_i_to_10__init::  
    int i = 1;  
loop_i_to_10__cond::  
    if (i > 10) goto loop_i_to_10__end;  
loop_i_to_10__body::  
    printf("%d", i);  
    putchar('\n');  
loop_i_to_10__step::  
    i++; // i = 1  
    goto loop_i_to_10__cond;  
loop_i_to_10__end::
```

[source code for count_to_10.simple.c](#)

Printing First 10 Integers: MIPS

```
loop_i_to_10__init:
    li    $t0, 1                # int i = 1;

loop_i_to_10__cond:
    bgt   $t0, 10, loop_i_to_10__end # if (i > 10) goto loop_i_to_10__end;

loop_i_to_10__body:
    li    $v0, 1                # syscall 1: print_int
    move  $a0, $t0              #
    syscall                                # printf("%d", i);
    li    $v0, 11               # syscall 11: print_char
    li    $a0, '\n'             #
    syscall                       # putchar('\n');

loop_i_to_10__step:
    addi  $t0, $t0, 1           # i = i + 1;
    b     loop_i_to_10__cond

loop_i_to_10__end:
```

source code for count_to_10.s

Sum 100 Squares: C to simplified C

C

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

[source code for sum_100_squares.c](#)

Simplified 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('\n');  
    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:
```

Sum 100 Squares: MIPS

```
loop_i_to_100__body:
    mul $t2, $t1, $t1           # sum = (i * i) +
    add $t0, $t0, $t2           # sum;
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                     # printf("%d", sum);
    li $v0, 11                  # syscall 11: print_char
    li $a0, '\n'                #
    syscall                     # putchar('\n');
    li $v0, 0
    jr $ra                      # return 0;
```

source code for sum_100_squares.s

Conditionals — `if` and `&&`: from C to Simplified C

Standard C

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

Simplified C

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

Simplified C

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

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:
```

Standard C

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

Simplified C

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

Simplified 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

```
# assume i in $t0
# assume n in $t1

blt $t0, 0, then1
bge $t1, 42, then1
j   else1

then1:
    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

```
// 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

```
// 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

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

```
while (_Condition_) {  
    some_code_1  
    if (! Condition1) continue;  
    some_code_2  
    if (! Condition2) continue;  
    some_code_3  
}
```

C

```
int main(void) {  
    int n;  
    printf("Enter a number: ");  
    scanf("%d", &n);  
    if (n % 2 == 0 && n % 3 == 0) {  
        printf("six\n");  
    }  
    return 0;  
}
```

[source code for six.c](#)

Simplified C

```
int main(void) {  
    int n;  
    printf("Enter a number: ");  
    scanf("%d", &n);  
    if (n % 2 != 0) goto epilogue;  
    if (n % 3 != 0) goto epilogue;  
    printf("six-ish\n");  
epilogue:  
    return 0;  
}
```

[source code for six.simple.c](#)

&& example (six.s) : MIPS (part 1)

```
main:
    # Locals:
    # - $t0: int n
    # - $t1: n % 2
    # - $t2: n % 3
    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, epilogue  #      != 0) goto epilogue;
```

[source code for six.s](#)

&& example (six.s) : MIPS (part 2)

```
bnez    $t1, epilogue    #    != 0) goto epilogue;
rem $t2, $t0, 3          # if ((n % 3)
bnez    $t2, epilogue    #    != 0) goto epilogue;
li  $v0, 4               # syscall 4: print_string
la  $a0, six_msg         #
syscall                 # printf("six-ish\n");
epilogue:
li  $v0, 0               #
jr  $ra                # return 0;
.data
prompt_msg:
    .asciiz "Enter a number: "
six_msg:
    .asciiz "six-ish\n"
```

source code for six.s

|| example (two_three.c): C to simplified C

C

```
int main(void) {  
    int n;  
    printf("Enter a number: ");  
    scanf("%d", &n);  
    if (n % 2 == 0 || n % 3 == 0) {  
        printf("two-three-ish\n");  
    }  
    return 0;  
}
```

[source code for two_three.c](#)

Simplified C

```
int main(void) {  
    int n;  
    printf("Enter a number: ");  
    scanf("%d", &n);  
    if (n % 2 == 0) goto two_three_print;  
    if (n % 3 == 0) goto two_three_print;  
    goto epilogue;  
two_three_print:  
    printf("two-three-ish\n");  
epilogue:  
    return 0;  
}
```

[source code for two_three.simple.c](#)

|| example (two_three.s) : MIPS (part 1)

```
main:
    # Locals:
    # - $t0: int n
    # - $t1: n % 2
    # - $t2: n % 3
    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)
    beqz $t1, two_three_print # == 0) goto two_three_print;
```

[source code for two_three.s](#)

|| example (two_three.s) : MIPS (part 2)

```
    beqz    $t1, two_three_print    #    == 0) goto two_three_print;
    rem    $t2, $t0, 3              # if ((n % 3)
    beqz    $t2, two_three_print    #    == 0) goto two_three_print;
    b       epilogue                # goto epilogue;

two_three_print:
    li     $v0, 4                   # syscall 4: print_string
    la     $a0, two_three_msg      #
    syscall                        # printf("two-three-ish\n");

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

.data

prompt_msg:
    .asciiz "Enter a number: "

two_three_msg:
    .asciiz "two-three-ish\n"
```

break/continue example (forever_23.c) : C to simplified C

C

```
int main(void) {
    for (int n = 0; n < 100; n++) {
        if (n % 3 == 0) {
            continue;
        }
        if (n % 23 == 0) {
            break;
        }
        printf("%d\n", n);
    }
    return 0;
}
```

[source code for forever_23.c](#)

Simplified C

```
int main(void) {
    int n;
    n = 0;
forever_23_loop_top:
    if (n > 100) goto forever_23_loop_end;
    if (n % 3 == 0) goto forever_23_loop_top;
    if (n % 23 == 0) goto forever_23_loop_top;
    printf("%d", n);
    putchar('\n');
forever_23_loop_next:
    n = n + 1;
    goto forever_23_loop_top;
forever_23_loop_end:
    return 0;
}
```

[source code for forever_23.simple.c](#)

break/continue example (forever_23.s) : MIPS (part 1)

```
main:
    # Locals:
    # - $t0: int n
    # - $t1: n % 2
    # - $t2: n % 23
forever_23_loop_init:
    li    $t0, 0           # int n = 0;
forever_23_loop_top:
    rem   $t2, $t0, 3       # if ((n % 3)
    beqz   $t2, forever_23_loop_next    # == 0) goto forever_23_loop_next;
    rem   $t1, $t0, 23      # if ((n % 23)
    beqz   $t1, forever_23_loop_end     # == 0) goto forever_23_loop_end;
```

[source code for forever_23.s](#)

break/continue example (forever_23.s) : MIPS (part 2)

```
beqz    $t1, forever_23_loop_end    #    == 0) goto forever_23_loop_end;
li      $v0, 1                      # syscall 1: print_int
move     $a0, $t0                    #
syscall                                # printf("%d", n);
li      $v0, 11                     # syscall 11: print_char
li      $a0, '\n'                   #
syscall                                # putchar('\n');

forever_23_loop_next:
    addi    $t0, $t0, 1              #    n++;
    b       forever_23_loop_top;     # goto forever_23_loop_top;

forever_23_loop_end:
epilogue:
    li      $v0, 0                  #
    jr     $ra                      # return 0;
```

[source code for forever_23.s](#)

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

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

can be written as:

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