COMP1521 23T1 — MIPS Control

https://www.cse.unsw.edu.au/~cs1521/23T1/

Jump Instructions

assembler	meaning	bit pattern
j *label*	pc = pc & 0xF0000000 | (X«2)	000010XXXXXXXXXXXXXXXXXXXXXXXXXX
jal *label*	ra = pc + 4; pc = pc & 0xF0000000 | (X«2)	000011XXXXXXXXXXXXXXXXXXXXXXXXXX
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 *label*	pc += I«2	pseudo-instruction
beq r_s, r_t, *label*	if (r_s == r_t) pc += I«2	000100ssssstttttIIIIIIIIIIIIIIII
bne r_s, r_t, *label*	if (r_s != r_t) pc += I«2	000101ssssstttttIIIIIIIIIIIIIIII
ble r_s, r_t, *label*	if (r_s <= r_t) pc += I«2	pseudo-instruction
bgt r_s, r_t, *label*	if (r_s > r_t) pc += I«2	pseudo-instruction
blt r_s, r_t, *label*	if (r_s < r_t) pc += I«2	pseudo-instruction
bge r_s, r_t, *label*	if (r_s >= r_t) pc += I«2	pseudo-instruction
blez r_s, *label*	if (r_s <= 0) pc += I«2	000110sssss00000IIIIIIIIIIIIIIII
bgtz r_s ,*label*	if (r_s > 0) pc += I«2	000111sssss00000IIIIIIIIIIIIIIII
bltz r_s ,*label*	if (r_s < 0) pc += I«2	000001sssss00000IIIIIIIIIIIIIIII
bgez r_s ,*label*	if (r_s >= 0) pc += I«2	000001sssss00001IIIIIIIIIIIIIIII
bnez r_s, *label*	if (r_s >= 0) pc += I«2	pseudo-instruction
beqz r_s, *label*	if (r_s >= 0) pc += I«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	Real Instructions
`bge  $t1, $t2, label`	`slt  $at, $t1, $t2` `beq  $at, $0, label`

`blt  $t1, 42, label`	`addi $at, $zero, 42`
	`slt  $at, $t1, $at`
	`bne  $at, $0, label`

`beqz $t3, label`	`beq  $t3, $0, label`
`bnez $t4, label`	`bne  $t4, $0, label`
`b 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

goto in C

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:
```

Printing First 10 Integers: C to simplified C

C

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

source code for print10.c

Simplified C

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

source code for print10.simple.c

Printing First 10 Integers: MIPS

```
# 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("%c", '\n');
    li   $v0, 11
    syscall
    addi $t0, $t0, 1    #   i++;
    b    loop           # goto loop;
end:
    li   $v0, 0         # return 0
    jr   $ra
```

source code for print10.s

Sum 100 Squares: C to simplified 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

```
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

Sum 100 Squares: MIPS

```
# 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.s

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.s

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;
}
```

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:
```

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:
```


Odd or Even: C to simplified 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;
}
```

source code for odd_even.c

Simplified 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;
}
```

source code for odd_even.simple.c

Odd or Even: MIPS

```
# 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           # 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
```

source code for odd_even.s

Odd or Even: MIPS

```
odd:                        # else
    la   $a0, string2       # printf("Odd\n");
    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"
```

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

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

source code for print10.c

Simplified C

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

source code for print10.simple.c

Printing First 10 Integers: MIPS

```
# 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("%c", '\n');
    li   $v0, 11
    syscall
    addi $t0, $t0, 1  #   i++;
    b    loop           # goto loop;
end:
    li   $v0, 0         # return 0
    jr   $ra
```

source code for print10.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

Sum 100 Squares: MIPS

```
# 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.s

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.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:
```

Conditionals — if and &&: from 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

```
    # 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;
    if (n >= 42) goto then1;
    goto else1;
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

```
    # 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:
```

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

```
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;
end:
```