CSE 220: Systems Fundamentals I Unit 4: MIPS Assembly: Branches and Loops

Branches

- There are no if-statements or loops in MIPS
- Instead there are different kind of **branch** statements that direct the CPU to execute instructions out of sequential order
- In addition to the 32 registers we can use as programmers, there is also the **program counter** (**PC**), which holds the address of the next instruction to execute
- After an instruction is fetched (the address of which is in the PC), the value in the PC is incremented by 4 (i.e., 4 bytes)
- The assumption is that the next instruction to execute is in the neighboring memory cell
- Branch instructions provide a different value to the PC

Types of Branches

- **Conditional** branches
  - The PC is updated if a condition is true
    - *branch on equal* (`beq`)
    - *branch on not equal* (`bne`)
    - *branch on less than* zero (`bltz`)
    - many others...
- **Unconditional** branches
  - The PC is changed directly
    - *jump* (`j`)
    - *jump register* (`jr`)
    - *jump and link* (`jal`)

Conditional Branching

- Used for implementing if-statements, switch-statements and loops
- `beq`: if two registers have the same data, jump to the instruction at a provided memory address
- `bne`: if two registers have different data, jump to the instruction at a provided memory address
- Example usage:
  - `beq $a0, $s1, Equal_Case`

`beq` Example

```
  addi $s0, $0, 4         # $s0 = 0 + 4 = 4
  addi $s1, $0, 1         # $s1 = 0 + 1 = 1
  sll  $s1, $s1, 2        # $s1 = 1 << 2 = 4
  beq  $s0, $s1, target   # branch is taken
  addi $s1, $s1, 1        # not executed
  sub  $s1, $s1, $s0      # not executed

target:
  add  $s1, $s1, $s0      # $s1 = 4 + 4 = 8
```


`bne` Example

```
  addi $s0, $0, 4         # $s0 = 0 + 4 = 4
  addi $s1, $0, 1         # $s1 = 0 + 1 = 1
  sll  $s1, $s1, 2        # $s1 = 1 << 2 = 4
  bne  $s0, $s1, target   # branch not taken
  addi $s1, $s1, 1        # $s1 = 4 + 1 = 5
  sub  $s1, $s1, $s0      # $s1 = 5 – 4 = 1

target:
  add  $s1, $s1, $s0      # $s1 = 1 + 4 = 5
```


Conditional Branching

- Other conditional branching instructions:
  - **`bgez`**: branch to label if register contains a value greater than or equal to zero
    - Example: **`bgez $a0, target`**
  - **`bgtz`**: branch on greater than zero
  - **`blez`**: branch on less than or equal to zero
  - **`bltz`**: branch on less than zero
- **`bge`**: branch on greater than or equal to
  - Example: **`bge rs, rt, label`**
  - Branch to **`label`** if **`rs`** $\geq$ **`rt`**

Conditional Branching

- The four relational operators $<, >, \leq, \geq$ are actually pseudoinstructions
- They can be implemented with the help of the R-type **`slt`** instruction: *set on less than*
  - **`slt rd, rs, rt`**
  - Set **`rd`** to 1 if **`rs`** < **`rt`**; otherwise, set **`rd`** to 0

If-statement Example

- Java code:

```
if (i == j)
    f = g + h;
f = f - i;
```

MIPS code:

```
# $s0 = f, $s1 = g,
# $s2 = h, $s3 = i,
# $s4 = j
    bne $s3, $s4, L1
    add $s0, $s1, $s2

L1: sub $s0, $s0, $s3
```

- Note that the MIPS assembly code tests the opposite case ($i \neq j$). You will see this convention used a lot.

Unconditional Branching

- An unconditional branch is akin to a "go to" statement
- I'll show you how to use one in a loop in a few minutes

```
addi $s0, $0, 4         # $s0 = 4
addi $s1, $0, 1         # $s1 = 1
j target                # jump to target
sra $s1, $s1, 2         # not executed
addi $s1, $s1, 1        # not executed
sub $s1, $s1, $s0       # not executed

target:
  add $s1, $s1, $s0     # $s1 = 1 + 4 = 5
```


Unconditional Branching

- The **`j`** instruction simply takes an immediate value that gives part of the address (26 bits) to jump to
  - The 32-bit target address is formed by concatenating the first 4 bits of the PC to the 26-bit immediate after shifting them 2 bits to the left
- The **`jr`** instruction is an R-type instruction that jumps to the address given in a register
  - Example: **`jr $s0`**
  - Used when returning from function calls
- The **`jal`** instruction is used when making a function call
- More on **`jr`** and **`jal`** in a later Unit

MIPS Program: Find max(a, b, c)

- Given a in **`$s0`**, b in **`$s1`** and c in **`$s2`**, find the maximum of the three and store the maximum in **`$s3`**
- Java code:

```
if (a > b)
   if (a > c)
      max = a;
   else
      max = c;
else
   if (b > c)
      max = b;
   else
      max = c;
```

MIPS Program: Find max(a, b, c)

```
   # s0 = a, $s1 = b, $s2 = c, $s3 = max
   li $s0, 255           # a
   li $s1, 11            # b
   li $s2, 9             # c

   ble $s0, $s1, a_LTE_b  # a <= b, so either b or c is max'm
   ble $s0, $s2, maxC     # a > b but a <= c, so max = c
   move $s3, $s0          # a > b and a > c, so max = a
   j done
a_LTE_b:
   ble $s1, $s2,  maxC    # a <= b and b <= c, so max = c
   move $s3, $s1          # a <= b and b > c, so max = b
   j done

maxC:
   move $s3, $s2          # max = c

done:
```


while-loop Example

- Let' see how to write a while-loop in MIPS
- Java code:

```
// determines the power
// of n such that 2^n = 128
int pow = 1;
int n   = 0;

while (pow != 128) {
  pow = pow * 2;
  n = n + 1;
}
```

MIPS code:

```
# $s0 = pow, $s1 = n

   addi $s0, $0, 1
   add  $s1, $0, $0
   addi $t0, $0, 128
while:
   beq  $s0, $t0, done
   sll  $s0, $s0, 1
   addi $s1, $s1, 1
   j    while
done:
```


for-loop Example #1

- Recall that we typically use for-loops when we know the exact number of iterations
- Java code:

```
// add the numbers
// from 0 to 9
int sum = 0;
int i;
for (i=0; i!=10; i++) {
  sum = sum + i;
}
```

MIPS code:

```
# $s0 = i, $s1 = sum
       add  $s1, $0, $0
       add  $s0, $0, $0
       addi $t0, $0, 10
for:   beq  $s0, $t0, done
       add  $s1, $s1, $s0
       addi $s0, $s0, 1
       j    for
done:
```


for-loop Example #2

Java code:

```
// sums the powers of
// 2 from 1 to 256
int sum = 0;
int i;

for (i=1; i < 257; i=i*2) {
  sum = sum + i;
}
```

MIPS code:

```
# $s0 = i, $s1 = sum
      add  $s1, $0, $0
      addi $s0, $0, 1
      addi $t0, $0, 257
loop: slt  $t1, $s0, $t0
      beq  $t1, $0, done
      add  $s1, $s1, $s0
      sll  $s0, $s0, 1
      j    loop
done:
```

switch-statement Example

- Java code:

```
switch (amount) {
  case 20:
    fee = 2;
    break;
  case 50:
    fee = 3;
    break;
  case 100:
    fee = 5;
    break;
  default:
    fee = 7;
}
```

MIPS code:

```
case20:
      li $t0, 20
      bne $s0, $t0, case50
      li $s1, 2
      j done
case50:
      li $t0, 50
      bne $s0, $t0, case100
      li $s1, 3
      j done
case100:
      li $t0, 100
      bne $s0, $t0, default
      li $s1, 5
      j done
default:
      li $s1, 7
done:
```


Example: Count # of Ones

- Let's write a MIPS program that counts the number of binary 1s in a 32-bit word **num**
- Java code:

```
counter = 0;
position = 1;
for (i = 0; i < 32; i++) {
    bit = num & position;
    if (bit != 0)
        counter++;
    position = position << 1;
}
```


Example: Print First *N* Primes

- Let's write a program to print the first *N* prime numbers, where *N* is hard-coded
- It's helpful first to look a Java implementation and then turn it into MIPS

Example: Leap Year

- A year after 1582 is a leap year if it is divisible by 4 with the exception of centenary years (years ending in 00) that are not divisible by 400
- 2015 was not a leap year because 2015 is not divisible by 4
- 1900 was not a leap year because although 1900 is divisible by 100, it is not divisible by 400
- 2000 was a leap year because 2000 is divisible by 400

```
if (year % 4 != 0) then
   ordinary_year
else if (year%100 == 0) and (year%400 != 0) then
   ordinary_year
else
   leap_year
```