MIPS Assembly Programming
Branch and Jump

• Branch and Jump instructions Alter the order of program execution…
  – High Level Languages (HLL)
  – Assembly Language (AL).
In HLL (Java, C/C++)

- Constructs for altering the order of program execution within a procedure:
  - if (...) ... else ...
  - switch (...) { case ... case ... case ... default ... }
  - while (...) ...
  - do ... while (...)
  - for (...) ...
  - goto ... // C/C++ only - not Java
In Assembly Language

We only have two groups of similar instructions:

1. Unconditional Branch / Jump instruction
   - puts a new value into the program counter, causing the next instruction to be fetched from that location

2. Conditional Branch instructions
   - puts a new value into the program counter if and only if some condition is true

- Program Counter (PC) register - holds the address of the NEXT instruction to be fetched from memory and executed.

- In MIPS, a 4 is added to the pc after each fetched instruction (all MIPS instructions are one word = 4-bytes)
Branch and Jump instructions

- Unconditional branch or Jump instruction: \texttt{goto label (b)}

- Conditional branch instructions:
  \texttt{if condition is true \ldots then goto label (beq, bne)}

<table>
<thead>
<tr>
<th>Op</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>\textit{lab}</td>
<td>Unconditional branch to \textit{lab}.</td>
</tr>
<tr>
<td>beq</td>
<td>src1, src2, \textit{lab}</td>
<td>Branch to \textit{lab} if src1 \equiv src2.</td>
</tr>
<tr>
<td>bne</td>
<td>src1, src2, \textit{lab}</td>
<td>Branch to \textit{lab} if src1 \neq src2.</td>
</tr>
</tbody>
</table>
# Branch and Jump instructions

<table>
<thead>
<tr>
<th>Op</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>lab</td>
<td>Unconditional branch to lab.</td>
</tr>
<tr>
<td>beq</td>
<td>src1, src2, lab</td>
<td>Branch to lab if src1 ≡ src2.</td>
</tr>
<tr>
<td>bne</td>
<td>src1, src2, lab</td>
<td>Branch to lab if src1 ≠ src2.</td>
</tr>
<tr>
<td>bge(u)</td>
<td>src1, src2, lab</td>
<td>Branch to lab if src1 ≥ src2.</td>
</tr>
<tr>
<td>bgt(u)</td>
<td>src1, src2, lab</td>
<td>Branch to lab if src1 &gt; src2.</td>
</tr>
<tr>
<td>ble(u)</td>
<td>src1, src2, lab</td>
<td>Branch to lab if src1 ≤ src2.</td>
</tr>
<tr>
<td>blt(u)</td>
<td>src1, src2, lab</td>
<td>Branch to lab if src1 &lt; src2.</td>
</tr>
<tr>
<td>beqz</td>
<td>src1, lab</td>
<td>Branch to lab if src1 ≡ 0.</td>
</tr>
<tr>
<td>bnez</td>
<td>src1, lab</td>
<td>Branch to lab if src1 ≠ 0.</td>
</tr>
<tr>
<td>bgez</td>
<td>src1, lab</td>
<td>Branch to lab if src1 ≥ 0.</td>
</tr>
<tr>
<td>bgtz</td>
<td>src1, lab</td>
<td>Branch to lab if src1 &gt; 0.</td>
</tr>
<tr>
<td>blez</td>
<td>src1, lab</td>
<td>Branch to lab if src1 ≤ 0.</td>
</tr>
<tr>
<td>bltz</td>
<td>src1, lab</td>
<td>Branch to lab if src1 &lt; 0.</td>
</tr>
<tr>
<td>bgezal</td>
<td>src1, lab</td>
<td>If src1 ≥ 0, then put the address of the next instruction into $ra and branch to lab.</td>
</tr>
<tr>
<td>bgtzal</td>
<td>src1, lab</td>
<td>If src1 &gt; 0, then put the address of the next instruction into $ra and branch to lab.</td>
</tr>
<tr>
<td>bltzal</td>
<td>src1, lab</td>
<td>If src1 &lt; 0, then put the address of the next instruction into $ra and branch to lab.</td>
</tr>
</tbody>
</table>

*bge, bgt, ble, blt* are pseudo-instructions
Branch if Equal (beq) (1)

Branches to target if $t0 = t1$
Branch if Equal ($beq$) (1)

Branches to target if $t0 = t1$

```
.text
.globl main
main:
    li $t0, 1
    li $t1, 1
    beq $t0, $t1, Branch
    la $a0, BranchNo  # prints for no match
    li $v0, 4
    syscall
    li $v0, 10        # system call code for exit = 10
    syscall
Branch:
    la $a0, BranchYes # prints for match
    li $v0, 4
    syscall
    li $v0, 10        # system call code for exit = 10
    syscall
.data
BranchYes: .asciiz "Successful Branch  \n"
BranchNo:  .asciiz "No Branch       \n"
```
Assemble ... GO

Successful Branch
-- program is finished running --

Successful Branch
-- program is finished running --
Branch if Equal; (**beq**) (2)

```assembly
# Folder: 13\BEQ.asm
# Ryan Preidel, 2011S

.text
li $t0, 1
li $t1, 2
beq $t0, $t1, Branch  # checks if values match
la $a0, BranchNo      # prints for no match
li $v0, 4
syscall
li $v0, 10            # system call code for exit = 10
syscall                # call operating sys

Branch:
la $a0, BranchYes     # prints for match
li $v0, 4
syscall
li $v0, 10            # system call code for exit = 10
syscall                # call operating sys

.data
BranchYes: .asciiz "Successful Branch \n"
BranchNo:  .asciiz "No Branch \n"
```
Assemble ... GO

No Branch

-- program is finished running --

No Branch

-- program is finished running --
Branch if Greater Than (\texttt{bgt})

Branches to target if $t0 > t1$

```assembly
# Folder: 13\BQT.asm
# Ryan Preidel, 2011S

.text
li $t0, 2
li $t1, 1
bgt $t0, $t1, Branch  # checks if values match
la $a0, BranchNo  # prints for no match
li $v0, 4
 syscall
li $v0, 10  # system call code for exit = 10
 syscall  # call operating sys

Branch:
la $a0, BranchYes  # prints for match
li $v0, 4
 syscall
li $v0, 10  # system call code for exit = 10
 syscall  # call operating sys

.data
BranchYes: .asciiz "Successful Branch \n"
BranchNo: .asciiz "No Branch \n"
```
Assemble … GO

Successful Branch

-- program is finished running --

Successful Branch

-- program is finished running --
Branch if Greater Than (bgt)

Branches to target if $t0 > $t1

```
.text
li $t0, 1
li $t1, 2
bgt $t0, $t1, Branch  # checks if values match
la $a0, BranchNo      # prints for no match
li $v0, 4
syscall
li $v0, 10            # system call code for exit = 10
syscall                # call operating sys
Branch:
la $a0, BranchYes     # prints for match
li $v0, 4
syscall
li $v0, 10            # system call code for exit = 10
syscall                # call operating sys
.data
BranchYes: .asciiz "Successful Branch \n"
BranchNo:  .asciiz "No Branch \n"
```
Assemble ...GO

No Branch

-- program is finished running --

No Branch

-- program is finished running --
Branch if Less Than or Equal To (ble)

Branches to target if $t0 <= $t1

```asm
.text
li $t0, 1
li $t1, 2
ble $t0, $t1, Branch  # checks if values match
la $a0, BranchNo     # prints for no match
li $v0, 4
syscall
li $v0, 10           # system call code for exit = 10
syscall
Branch:
la $a0, BranchYes    # prints for match
li $v0, 4
syscall
li $v0, 10           # system call code for exit = 10
syscall
.data
BranchYes: .asciiz  "Successful Branch\n"
BranchNo:  .asciiz  "No Branch\n"
```

Assemble ... GO

Successful Branch
-- program is finished running --

Successful Branch
-- program is finished running --
Branch if Less Than or Equal To (ble)

Branches to target if $t0 <= $t1

```assembly
.text
li $t0, 2
li $t1, 1
ble $t0, $t1, Branch    # checks if values match
la $a0, BranchNo       # prints for no match
li $v0, 4
syscall
li $v0, 10            # system call code for exit = 10
syscall
Branch:
la $a0, BranchYes     # prints for match
li $v0, 4
syscall
li $v0, 10            # system call code for exit = 10
syscall
.data
BranchYes: .asciiz "Successful Branch\n"
BranchNo: .asciiz "No Branch\n"
```
Assemble …GO

No Branch

-- program is finished running --
Branch if Less Than (blt)

```assembly
# Folder: 13\BLT.asm
# Ryan Preidel, 2011S

.text
li $t0, 0
li $t1, 1
blt $t0, $t1, Branch # checks if values match
la $a0, BranchNo # prints for no match
li $v0, 4
syscall
li $v0, 10 # system call code for exit = 10
syscall
Branch:
la $a0, BranchYes # prints for match
li $v0, 4
syscall
li $v0, 10 # system call code for exit = 10
syscall
.data
BranchYes: .asciiz "Successful Branch \n"
BranchNo: .asciiz "No Branch \n"
```
Assemble …GO

Successful Branch

-- program is finished running --

Successful Branch

-- program is finished running --
Branch if Less Than (**blt**)

```assembly
# Folders: 13\BLT.asm
# Ryan Preidel, 2011S

.text
li $t0, 1
li $t1, 1
blt $t0, $t1, Branch   # checks if values match
la $a0, BranchNo       # prints for no match
li $v0, 4
syscall
li $v0, 10             # system call code for exit = 10
syscall
Branch:
la $a0, BranchYes      # prints for match
li $v0, 4
syscall
li $v0, 10             # system call code for exit = 10
syscall
.data
BranchYes: .asciiz "Successful Branch \n"
BranchNo:  .asciiz "No Branch \n"
```
Assemble …GO

No Branch

-- program is finished running --

No Branch

-- program is finished running --
Branch if Greater Than or Equal To (bge)

Branches to target if $t0 >= $t1

```assembly
.text
li $t0, 1
li $t1, 1
bge $t0, $t1, Branch  # checks if values match
la $a0, BranchNo     # prints for no match
li $v0, 4
syscall
li $v0, 10
syscall             # system call code for exit = 10
Branch:
la $a0, BranchYes  # prints for match
li $v0, 4
syscall
li $v0, 10          # system call code for exit = 10
        syscall
.data
BranchYes: .asciiz "Successful Branch\n"
BranchNo:  .asciiz "No Branch\n"
```
Assemble …GO

```
Successful Branch

-- program is finished running --
```

```
Successful Branch

-- program is finished running --
```
# Folder: 13\BGE.asm
# Ryan Preidel, 2011

.text
li $t0, 1
li $t1, 2
bge $t0, $t1, Branch  # checks if values match
la $a0, BranchNo  # prints for no match
li $v0, 4
syscall
li $v0, 10  # system call code for exit = 10
syscall
Branch:
la $a0, BranchYes  # prints for match
li $v0, 4
syscall
li $v0, 10  # system call code for exit = 10
syscall
.data
BranchYes: .asciiz "Successful Branch\n"
BranchNo: .asciiz "No Branch\n"
Assemble …GO

No Branch

-- program is finished running --
Branch if Not Equal (bne)

Branches to target if $t0 \neq t1$

```assembly
.data
BranchYes: .asciiz "Successful Branch  \n"
BranchNo: .asciiz "No Branch  \n"

.text
li $t0, 1
li $t1, 2
bne $t0, $t1, Branch  # checks if values match
la $a0, BranchNo      # prints for no match
li $v0, 4
syscall
li $v0, 10
syscall
Branch:               # system call code for exit = 10
la $a0, BranchYes     # prints for match
li $v0, 4
syscall
li $v0, 10
syscall              # call operating sys

Branch:               # system call code for exit = 10
```

# Folder: l3\BNE.asm
# Ryan Preidel, 2011S
Assemble ...GO

Successful Branch

-- program is finished running --

Successful Branch

-- program is finished running --
Branch if Not Equal (bne)

Branches to target if $t0 \neq \$t1

```asm
.text
li \$t0, 1
li \$t1, 1
bne \$t0, \$t1, Branch  # checks if values match
la \$a0, BranchNo      # prints for no match
li \$v0, 4
syscall
li \$v0, 10            # system call code for exit = 10
syscall
Branch:
la \$a0, BranchYes     # prints for match
li \$v0, 4
syscall
li \$v0, 10            # system call code for exit = 10
syscall
.data
BranchYes: .asciiz "Successful Branch  \n"
BranchNo:  .asciiz "No Branch  \n"
```
Assemble ...GO

No Branch

-- program is finished running --

No Branch

-- program is finished running --
## Jump

<table>
<thead>
<tr>
<th>Op</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>label</td>
<td>Jump to label <code>lab</code>.</td>
</tr>
</tbody>
</table>
Example: Using `beq` and `j`

```assembly
.globl main
main:
    li $t0, 1
    li $t1, 1
    li $t2, 0

loop:
    beq $t0, 5 exit
    mul $t1, $t1, 2
    add $t2, $t2, $t1
    addi $t0, $t0, 1
    j loop

exit:
    move $a0, $t2
    li $v0, 1
    syscall
    li $v0, 10
    syscall
```

- Trace the example.
- What is the implemented expression by the loop?
- What is the result \[ $t2 \]?
Using **beq** and **j**

```
.main:
    li    $t0, 1    # loads 1 to t0
    li    $t1, 1    # loads 1 to t1
    li    $t2, 0    # loads 0 to t2

loop:
    beq    $t0, 5 exit  # if the value of 5 is in t0, go to exit
    mul    $t1, $t1, 2  # multiply t1 by 2 and store it to t1
    add    $t2, $t2, $t1  # adds t1 and t2 and stores it to t2
    addi   $t0, $t0, 1  # adds t0 and 1 and stores it to t0 -- uses the loop to increments from 1 to 5
    j      loop  # jumps to loop

exit:
    move   $a0, $t2  # moves the value in t2 into a0
    li     $v0, 1  # prints the value in a0
    syscall
    li     $v0, 10  #exit
    syscall
```
Trace ...

<table>
<thead>
<tr>
<th></th>
<th>$t0$</th>
<th>$t1$</th>
<th>$t2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2*2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2<em>2</em>2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2<em>2</em>2*2</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

\[2 + 2 \times 2 + 2 \times 2 \times 2 + 2 \times 2 \times 2 \times 2 = 30\]
Formula

\[ \sum_{i=1}^{4} 2^i \]

\[ 2^1 + 2^2 + 2^3 + 2^4 = 30 \]

\[ 2 + 2 \times 2 + 2 \times 2 \times 2 + 2 \times 2 \times 2 \times 2 = 30 \]
# Jump

<table>
<thead>
<tr>
<th>Op</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>label</td>
<td>Jump to label <em>lab</em>.</td>
</tr>
<tr>
<td>jr</td>
<td>src1</td>
<td>Jump to location <em>src1</em>.</td>
</tr>
<tr>
<td>jal</td>
<td>label</td>
<td>Jump to label <em>lab</em>, and store the address of the next instruction in $ra$.</td>
</tr>
<tr>
<td>jalr</td>
<td>src1</td>
<td>Jump to location <em>src1</em>, and store the address of the next instruction in $ra$.</td>
</tr>
</tbody>
</table>
Example: Using: **jal ... jr**

**Jal x = Jump and link to x**

**Jal subroutine = Jump to label subroutine, and store the address of the next instruction in the return address register: $ra.**

**jr src1 = Jump to location src1**
The jal instruction saves the return address in register $ra.
Assemble ... GO

This is from main ...
This is a subroutine call
-- program is finished running --

This is from main ...
This is a subroutine call
-- program is finished running --
Notes …

- **Jal** stands for **Jump and Link**, it is the way of jumping to another location while retaining the memory location of where you jumped from …

- In our example “subroutine” is the location where the program counter (pc) should jump to …

- The method under “subroutine” prints out a short message

- **jr $ra** stands for **jump return** and is used to jump back to the register $ra (return address)

- **nop** stands for **no-operation**, it is used to prevent the "processor" from beginning another operation (**used in pipelining**).