# Unit 3:Assembly Branches & Loops

CSE 220: System Fundamental I

Stony Brook University

Joydeep Mitra

# Branching

- MIPS sequentially executes instructions. How?
  - Fetches from memory by incrementing Program Counter (PC) by 4.
- To implement branching, we need to control the PC.
- How to control the PC?
  - Perform test with a *conditional branch instruction.*
  - A conditional branch instruction performs a test and branches to a label/address only if the condition is TRUE.
  - *Unconditional branch (jump)* instructions always branch to some label/address.

# Conditional Branch Instructions

- Two types of branch instructions
  - branch is equal (`beq`)
    - `beq rs, rt, <label>`
      if value in `rs` == `rt` then execute the instruction in address `<label>`
  - branch not equal (`bne`)
    - `bne rs, rt, imm`
      if value in `rs` *!*= `rt` then execute the instruction in address `<label>`

# Conditional Branch Instructions

**MIPS Assembly Code**

```
  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   # $s0 == $s1, so branch is taken
  addi  $s1, $s1, 1        # not executed
  sub   $s1, $s1, $s0      # not executed

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

**Conditional branching using *beq***

- Branch to label *target* if \$s0 == \$s1
- else continue sequentially

**MIPS Assembly Code**

```
  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   # $s0 == $s1, so branch is 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 using *bne***

- Branch to label *target* if \$s0 ≠ \$s1
- else continue sequentially

# Unconditional Branch Instructions

- You can move the PC unconditionally using:
  - *jump* (`j`)
    - `j <label>`
      Execution jumps to the instruction at address `<label>`
  - *jump register* (`jr`)
    - `j <register>`
      Execution jumps to the instruction at address in `<register>`
  - *jump and link* (`jal`)
    - More on this later.

# Unconditional Branch Instructions

**MIPS Assembly Code**

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

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

**Unconditional branch using *j***

**MIPS Assembly Code**

```
0x00002000  addi  $s0, $0, 0x2010   # $s0 = 0x2010
0x00002004  jr    $s0               # jump to 0x00002010
0x00002008  addi  $s1, $0, 1        # not executed
0x0000200c  sra   $s1, $s1, 2       # not executed
0x00002010  lw    $s3, 44($s1)      # executed after jr instruction
```

**Unconditional branch using *jr***

# Conditional Statements

- Higher-level languages have conditional statements
  - *if* statements
  - *if/else* statements
  - *switch/case* statements
- How can we simulate them in MIPS?

# *if* Statement in MIPS

- Test the **inverted condition** of the *if condition* in the high-level code
- Skip to statement after if block if **inverted condition** is TRUE

**High-Level Code**

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

f = f - i;
```

**MIPS Assembly Code**

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
  bne $s3, $s4, L1     # if i != j, skip if block
  add $s0, $s1, $s2    # if block: f = g + h
L1:
  sub $s0, $s0, $s3    # f = f - i
```

# *if/else* Statement in MIPS

- Like in *if*, test the **inverted condition**
- If inverted condition TRUE, then jump to *else label*
- Otherwise, continue sequentially and skip past *else*

**High-Level Code**

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

else
  f = f - i;
```

**MIPS Assembly Code**

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
  bne $s3, $s4, else   # if i != j, branch to else
  add $s0, $s1, $s2    # if block: f = g + h
  j   L2               # skip past the else block
else:
  sub $s0, $s0, $s3    # else block: f = f - i
L2:
```

# *Switch/case* in MIPS

- *Switch/case* is like a series of *if/else* statements
- If 1st condition is not met jump to *next case*; otherwise continue sequentially and jump to *default case*
- Example:
  Calculate the fee for an ATM based on an input *amount*
  Possible input amounts – \$20, \$50, and \$100

# *Switch/case* Example

**High-Level Code**

```
switch (amount) {

  case 20:   fee = 2; break;

  case 50:   fee = 3; break;

  case 100:  fee = 5; break;

  default: fee = 0;

}

// equivalent function using if/else statements
  if      (amount == 20)  fee = 2;
  else if (amount == 50)  fee = 3;
  else if (amount == 100) fee = 5;
  else                    fee = 0;
```

**MIPS Assembly Code**

```
# $s0 = amount, $s1 = fee

case20:
  addi $t0, $0, 20        # $t0 = 20
  bne  $s0, $t0, case50   # amount == 20? if not,
                          # skip to case50
  addi $s1, $0, 2         # if so, fee = 2
  j    done               # and break out of case

case50:
  addi $t0, $0, 50        # $t0 = 50
  bne  $s0, $t0, case100  # amount == 50? if not,
                          # skip to case100
  addi $s1, $0, 3         # if so, fee = 3
  j    done               # and break out of case

case100:
  addi $t0, $0, 100       # $t0 = 100
  bne  $s0, $t0, default  # amount == 100? if not,
                          # skip to default
  addi $s1, $0, 5         # if so, fee = 5
  j    done               # and break out of case

default:
  add  $s1, $0, $0        # fee = 0

done:
```

# While Loops in MIPS

- Repeatedly test the **inverted condition**
  - If inverted condition is met, then exit loop
  - Otherwise, continue sequentially and *unconditionally jump* to start of loop

**High-Level Code**

```
int pow = 1;
int x = 0;



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

**MIPS Assembly Code**

```
# $s0 = pow, $s1 = x
  addi $s0, $0, 1         # pow = 1
  addi $s1, $0, 0         # x = 0

  addi $t0, $0, 128       # t0 = 128 for comparison
while:
  beq  $s0, $t0, done     # if pow == 128, exit while loop
  sll  $s0, $s0, 1        # pow = pow * 2
  addi $s1, $s1, 1        # x = x + 1
  j    while
done:
```

# For Loops

- Like *while*, repeatedly test the **inverted condition**
- *for* loops often have a loop variable to keep track of no. of loop executions

```
for (initialization; condition; loop operation)
    statement
```

  - `initialization` executes before loop starts to set initial loop variable.
  - `condition` is tested repeatedly at loop beginning.
  - If `condition` is not met, exit loop.
  - Execute loop operation to update loop variable at end of a loop.

# For Loop Example

| **High-Level Code** | **MIPS Assembly Code** |
| --- | --- |

**High-Level Code**

```
int sum = 0;



for (i = 0; i != 10; i = i + 1) {
  sum = sum + i ;


}
// equivalent to the following while loop
int sum = 0;
int i = 0;
while (i != 10) {
  sum = sum + i;
  i = i + 1;
}
```

**MIPS Assembly Code**

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

# Magnitude Comparison

- So far, we have only tested for equality.
- MIPS provides the *set less than instruction (*`slt`*)* to perform other comparisons (e.g., `<, >, <=, >=`)

```
slt rd, rs, rt
```

  - Sets `rd` to 1 if `rs < rt`. Otherwise `rd` is 0.

# Magnitude Comparison Example

- Add the powers of 2 from 1 to 100

```
// high-level code

int sum = 0;
for (i = 1; i < 101; i = i * 2)
  sum = sum + i;
```

```
# MIPS assembly code

# $s0 = i, $s1 = sum
  addi  $s1, $0, 0     # sum = 0
  addi  $s0, $0, 1     # i = 1
  addi  $t0, $0, 101   # $t0 = 101

loop:
  slt   $t1, $s0, $t0     # if (i < 101) $t1 = 1, else $t1 = 0
  beq   $t1, $0, done     # if $t1 == 0 (i >= 101), branch to done
  add   $s1, $s1, $s0     # sum = sum + i
  sll   $s0, $s0, 1       # i = i * 2
  j     loop

done:
```

# Magnitude Comparison Example

- Let's try out a few more comparisons with `slt`

```
if (g > h)
    g = g + h;
else
    g = g − h;
```

```
start:
  lw $s1, g
  lw $s2, h

  slt $t0, $s1, $s2           # if g < h, then $t0 = 1 else $t0 = 0
  bne $t0, $0, g_lte_h        # if $t0 != 0, then goto label g_lte_h
  beq $s1, $s2, g_lte_h       # if g == h, then goto label g_lte_h
  add $s1, $s1, $s2           # g = g + h
  j done                      # terminate program
g_lte_h:
  sub $s1, $s1, $s2           # g = g - h

done:
  # terminate program
  li $v0, 10
  syscall
```

# Magnitude Comparison Example

- More comparisons with `slt`

```
if (g >= h)
    g = g + 1;
else
    h = h – 1;
```

```
start:
  lw $s1, g
  lw $s2, h

  slt $t0, $s1, $s2          # if g < h, then $t0 = 1 else $t0 = 0
  bne $t0, $0, g_lt_h        # if $t0 != 0, then goto label g_lt_h
  addi $s1, $s1, 1           # g = g + 1
  j done                     # terminate program
g_lt_h:
  addi $s2, $s2, -1          # h = h - 1

done:
  # terminate program
  li $v0, 10
  syscall
```

# Magnitude Comparison Example

- One last example with `slt`

```
if (g <= h)
    g = 0;
else
    h = 0;
```

```
start:
  lw $s1, g
  lw $s2, h

  slt $t0, $s1, $s2          # if g < h, then $t0 = 1 else $t0 = 0
  beq $t0, $0, g_gte_h       # if $t0 == 0, then goto label g_gte_h
  add $s1, $0, $0            # g = 0
  j done                     # terminate program
g_gte_h:
  bne $s1, $s2, g_gt_h       # if g == h then goto label g_gt_h
  add $s1, $0, $0            # g = 0
  j done                     # terminate program
g_gt_h:
  add $s2, $0, $0            # h = 0

done:
  # terminate program
  li $v0, 10
  syscall
```

# Pseudoinstructions For Comparisons

- Because comparisons are very common, MIPS provides *pseudoinstructions* based on **slt**
  - **`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`**

# What About Strings?

- Strings are an ordered collection of characters
- Every character is a byte and has a unique encoding
- MIPS uses ASCII encoding – **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- Every character is mapped to a unique ASCII code; there are 128 of them
- The extended ASCII has 256 codes, but we will be using the 128 code ASCII

# ASCII Table

| Decimal | Binary | Octal | Hex | ASCII | Decimal | Binary | Octal | Hex | ASCII | Decimal | Binary | Octal | Hex | ASCII | Decimal | Binary | Octal | Hex | ASCII |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00000000 | 000 | 00 | NUL | 32 | 00100000 | 040 | 20 | SP | 64 | 01000000 | 100 | 40 | @ | 96 | 01100000 | 140 | 60 | ` |
| 1 | 00000001 | 001 | 01 | SOH | 33 | 00100001 | 041 | 21 | ! | 65 | 01000001 | 101 | 41 | A | 97 | 01100001 | 141 | 61 | a |
| 2 | 00000010 | 002 | 02 | STX | 34 | 00100010 | 042 | 22 | “ | 66 | 01000010 | 102 | 42 | B | 98 | 01100010 | 142 | 62 | b |
| 3 | 00000011 | 003 | 03 | ETX | 35 | 00100011 | 043 | 23 | # | 67 | 01000011 | 103 | 43 | C | 99 | 01100011 | 143 | 63 | c |
| 4 | 00000100 | 004 | 04 | EOT | 36 | 00100100 | 044 | 24 | $ | 68 | 01000100 | 104 | 44 | D | 100 | 01100100 | 144 | 64 | d |
| 5 | 00000101 | 005 | 05 | ENQ | 37 | 00100101 | 045 | 25 | % | 69 | 01000101 | 105 | 45 | E | 101 | 01100101 | 145 | 65 | e |
| 6 | 00000110 | 006 | 06 | ACK | 38 | 00100110 | 046 | 26 | & | 70 | 01000110 | 106 | 46 | F | 102 | 01100110 | 146 | 66 | f |
| 7 | 00000111 | 007 | 07 | BEL | 39 | 00100111 | 047 | 27 | ‘ | 71 | 01000111 | 107 | 47 | G | 103 | 01100111 | 147 | 67 | g |
| 8 | 00001000 | 010 | 08 | BS | 40 | 00101000 | 050 | 28 | ( | 72 | 01001000 | 110 | 48 | H | 104 | 01101000 | 150 | 68 | h |
| 9 | 00001001 | 011 | 09 | HT | 41 | 00101001 | 051 | 29 | ) | 73 | 01001001 | 111 | 49 | I | 105 | 01101001 | 151 | 69 | i |
| 10 | 00001010 | 012 | 0A | LF | 42 | 00101010 | 052 | 2A | * | 74 | 01001010 | 112 | 4A | J | 106 | 01101010 | 152 | 6A | j |
| 11 | 00001011 | 013 | 0B | VT | 43 | 00101011 | 053 | 2B | + | 75 | 01001011 | 113 | 4B | K | 107 | 01101011 | 153 | 6B | k |
| 12 | 00001100 | 014 | 0C | FF | 44 | 00101100 | 054 | 2C | , | 76 | 01001100 | 114 | 4C | L | 108 | 01101100 | 154 | 6C | l |
| 13 | 00001101 | 015 | 0D | CR | 45 | 00101101 | 055 | 2D | - | 77 | 01001101 | 115 | 4D | M | 109 | 01101101 | 155 | 6D | m |
| 14 | 00001110 | 016 | 0E | SO | 46 | 00101110 | 056 | 2E | . | 78 | 01001110 | 116 | 4E | N | 110 | 01101110 | 156 | 6E | n |
| 15 | 00001111 | 017 | 0F | SI | 47 | 00101111 | 057 | 2F | / | 79 | 01001111 | 117 | 4F | O | 111 | 01101111 | 157 | 6F | o |
| 16 | 00010000 | 020 | 10 | DLE | 48 | 00110000 | 060 | 30 | 0 | 80 | 01010000 | 120 | 50 | P | 112 | 01110000 | 160 | 70 | p |
| 17 | 00010001 | 021 | 11 | DC1 | 49 | 00110001 | 061 | 31 | 1 | 81 | 01010001 | 121 | 51 | Q | 113 | 01110001 | 161 | 71 | q |
| 18 | 00010010 | 022 | 12 | DC2 | 50 | 00110010 | 062 | 32 | 2 | 82 | 01010010 | 122 | 52 | R | 114 | 01110010 | 162 | 72 | r |
| 19 | 00010011 | 023 | 13 | DC3 | 51 | 00110011 | 063 | 33 | 3 | 83 | 01010011 | 123 | 53 | S | 115 | 01110011 | 163 | 73 | s |
| 20 | 00010100 | 024 | 14 | DC4 | 52 | 00110100 | 064 | 34 | 4 | 84 | 01010100 | 124 | 54 | T | 116 | 01110100 | 164 | 74 | t |
| 21 | 00010101 | 025 | 15 | NAK | 53 | 00110101 | 065 | 35 | 5 | 85 | 01010101 | 125 | 55 | U | 117 | 01110101 | 165 | 75 | u |
| 22 | 00010110 | 026 | 16 | SYN | 54 | 00110110 | 066 | 36 | 6 | 86 | 01010110 | 126 | 56 | V | 118 | 01110110 | 166 | 76 | v |
| 23 | 00010111 | 027 | 17 | ETB | 55 | 00110111 | 067 | 37 | 7 | 87 | 01010111 | 127 | 57 | W | 119 | 01110111 | 167 | 77 | w |
| 24 | 00011000 | 030 | 18 | CAN | 56 | 00111000 | 070 | 38 | 8 | 88 | 01011000 | 130 | 58 | X | 120 | 01111000 | 170 | 78 | x |
| 25 | 00011001 | 031 | 19 | EM | 57 | 00111001 | 071 | 39 | 9 | 89 | 01011001 | 131 | 59 | Y | 121 | 01111001 | 171 | 79 | y |
| 26 | 00011010 | 032 | 1A | SUB | 58 | 00111010 | 072 | 3A | : | 90 | 01011010 | 132 | 5A | Z | 122 | 01111010 | 172 | 7A | z |
| 27 | 00011011 | 033 | 1B | ESC | 59 | 00111011 | 073 | 3B | ; | 91 | 01011011 | 133 | 5B | [ | 123 | 01111011 | 173 | 7B | { |
| 28 | 00011100 | 034 | 1C | FS | 60 | 00111100 | 074 | 3C | < | 92 | 01011100 | 134 | 5C | \ | 124 | 01111100 | 174 | 7C | \| |
| 29 | 00011101 | 035 | 1D | GS | 61 | 00111101 | 075 | 3D | = | 93 | 01011101 | 135 | 5D | ] | 125 | 01111101 | 175 | 7D | } |
| 30 | 00011110 | 036 | 1E | RS | 62 | 00111110 | 076 | 3E | > | 94 | 01011110 | 136 | 5E | ^ | 126 | 01111110 | 176 | 7E | ~ |
| 31 | 00011111 | 037 | 1F | US | 63 | 00111111 | 077 | 3F | ? | 95 | 01011111 | 137 | 5F | _ | 127 | 01111111 | 177 | 7F | DEL |

# Strings in MARS

- Use the .data directive to declare global strings

**`person: .asciiz “Hopper”`**

- **`person`** is the string label akin to variable name
- **`.asciiz`** is the *directive* that indicates an *array of characters* or string *ending with the null character*
- What is the point of the *null character*?
  - Indicates end of string

# Manipulating Strings

- The characters are stored sequentially in memory; each character occupying one byte.
- The string has a base address.
- The first character is in offset 0.
- A character in the string can be accessed by adjusting the offset.
- Example:

```
.data
 person: .asciiz “Grace Hopper”
 .text
 la $t0, person   # get starting address of string
 lbu $t1, 3($t0)  # load person[3] into $t1 (‘c’)
```

# Loop Over A String

- Just like in high level languages, we can loop/iterate over a string to process every character

```
.data
person: .asciiz “Grace Hopper”
.text
la $t0, person   # get starting address of string
li $t2, 0        # counter
li $t3, 12        # number of characters (iterations)
loop:
   lbu $t1, 0($t0)     # offset is always zero
   # ...process character in $t1...
   addi $t0, $t0, 1    # get address of next character
   addi $t2, $t2, 1    # counter++
   blt $t2, $t3, loop
```