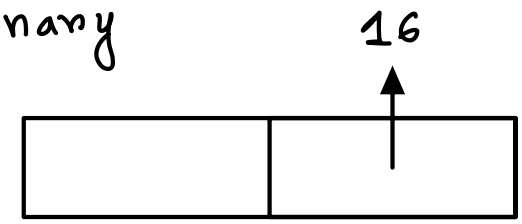
Branching far away (Slide 54)

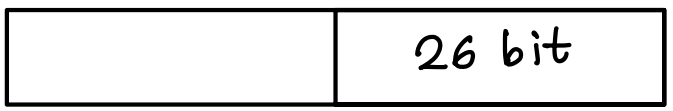
If branch target is too far to encode with 16-bit offset, assembler rewrites the code. Example:

```
beq $s0, $s1, L1
```

16 bit binary number




```
bne $s0, $s1, L2
j L1
L2: ...
```



j instruction

Slide 58 (important) (must know):

Register Usage

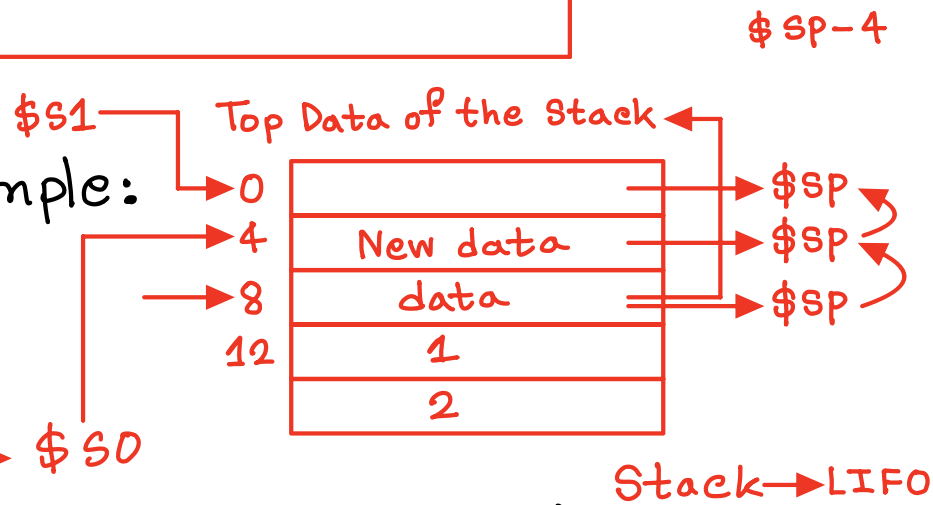
- $a0 – $a3: arguments (reg's 4 – 7)
- $v0, $v1: result values (reg's 2 and 3)
- $t0 – $t9: temporaries
  - Can be overwritten by callee
- $s0 – $s7: saved
  - Must be saved/restored by callee
- $gp: global pointer for static data (reg 28)
- $sp: stack pointer (reg 29)
- $fp: frame pointer (reg 30)
- $ra: return address (reg 31)

Functions:

Caller function

Called function

```
Main (){
      int x = 0;
      int y = 5;
      int z = addition(x, y);
}
int addition (int a, int b){
      int c = a+b;
      return c;
}
```



Leaf Procedure Example:

```
Main (){
      int f = 0;
      f = f + 1;
      int z = leaf_example(1,2,3,4);
      int y = f + z;
}
int leaf_example(int g, h, i, j) {
      int f;
      f = (g+h) - (i+j);
      return f;
}
```

Stack → LIFO

$SP holds address of top data.

jal → jump and link

Arguments g → $a0
h → $a1
i → $a2
j → $a3

f → $s0 (hence, need to save $s0 on stack)

Result in $v0

z, y in $s1, $s2

Answer: Mips Code:

```
add $s0, $zero, $zero
addi $s0, $s0, 1
jal leaf_example
add $s1, $v0, $zero
add $s2, $s0, $s1
leaf_example:
addi $sp, $sp, -4
sw $s0, 0($sp)
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
add $v0, $s0, $zero
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

sw $s0, 0($sp) → sw $s0, 0($sp)

$t0 → g+h

$t1 → i+j

$s0 → (g+h)-(i+j)

f = $s0

g = $s2

Top এ Data রাখার জন্য

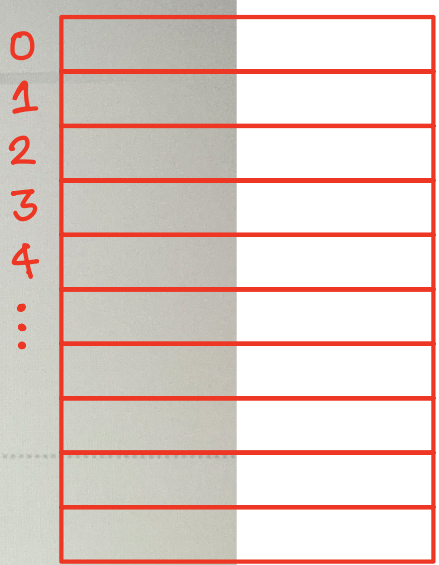
```
addi  $sp, $sp, -4
sw  $s0, 0($sp)
addi  $sp, $sp, -4
sw  $s2, 0($sp)
lw  $s2, 0($sp)
addi $sp, $sp, 4
lw $s2, 0($sp)
addi $sp, $sp, 4
```

slides → 68, 69, 70

Slide 68:

Character Data

- Byte-encoded character sets
  - ASCII: 128 characters
    - 95 graphic, 33 control
  - Latin-1: 256 characters
    - ASCII, +96 more graphic characters
- Unicode: 32-bit character set
  - Used in Java, C++ wide characters, …
  - Most of the world's alphabets, plus symbols
  - UTF-8, UTF-16: variable-length encodings



Slide 69:

Byte/Halfword Operations

- Could use bitwise operations
- MIPS byte/halfword load/store
  - String processing is a common case

```
lb rt, offset(rs)      lh rt, offset(rs)
```

  - Sign extend to 32 bits in rt

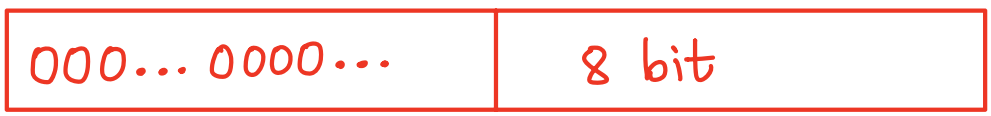
```
lbu rt, offset(rs)     lhu rt, offset(rs)
```

  - Zero extend to 32 bits in rt

```
sb rt, offset(rs)      sh rt, offset(rs)
```

  - Store just rightmost byte/halfword

lb rt, offset(rs) → Fetching 8 bit data



lh rt, offset(rs)

ASCII characters are always unsigned.

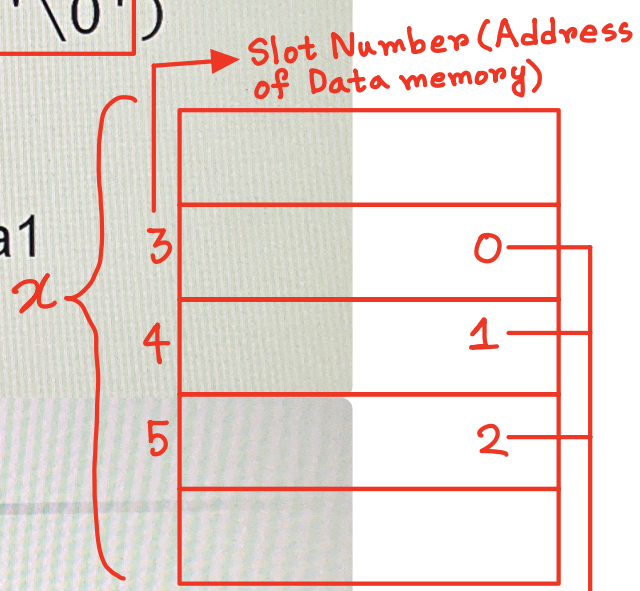
Slide 70:

String Copy Example

- C code (naïve):
  - Null-terminated string

```
void strcpy (char x[], char y[])
{ int i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i += 1;
}
```

  - Addresses of x, y in $a0, $a1
  - i in $s0



String Copy Example

- MIPS code:

```
strcpy:
    addi $sp, $sp, -4          # adjust stack for 1 item
    sw   $s0, 0($sp)           # save $s0
    add  $s0, $zero, $zero     # i = 0
L1: add  $t1, $s0, $a1         # addr of y[i] in $t1
    lbu  $t2, 0($t1)           # $t2 = y[i]
    add  $t3, $s0, $a0         # addr of x[i] in $t3
    sb   $t2, 0($t3)           # x[i] = y[i]
    beq  $t2, $zero, L2        # exit loop if y[i] == 0
    addi $s0, $s0, 1           # i = i + 1
    j    L1                    # next iteration of loop
L2: lw   $s0, 0($sp)           # restore saved $s0
    addi $sp, $sp, 4           # pop 1 item from stack
    jr   $ra                   # and return
```

Indexing of x[] array.

sw $s0, 0($sp)

while((x[i]=y[i])!='\0')

L1: add $t1, $s0, $a1

lbu $t2, 0($t1)

A[B[i]]=x

➢ A[B[i]]=x

Base address of A and B are in $s1 and $s2, x and i are in $s3 and $s4

Ans. sll $t0,$s4,2
add $t1, $s2,$t0
lw $t2,0($t1)
sll $t0,$t2,2
add $t1, $s1,$t2
sw $s3,0($t1)

A[B[i]]=x

B[i]=5

A[5]=x

32 bit architecture = 32/8 = 4 slots

64 bit architecture = 64/8 = 8 slots

256 bit architecture = 256/8 = 32 slots

A[6] in 64 bit architecture

lw $t0, 48($s1)