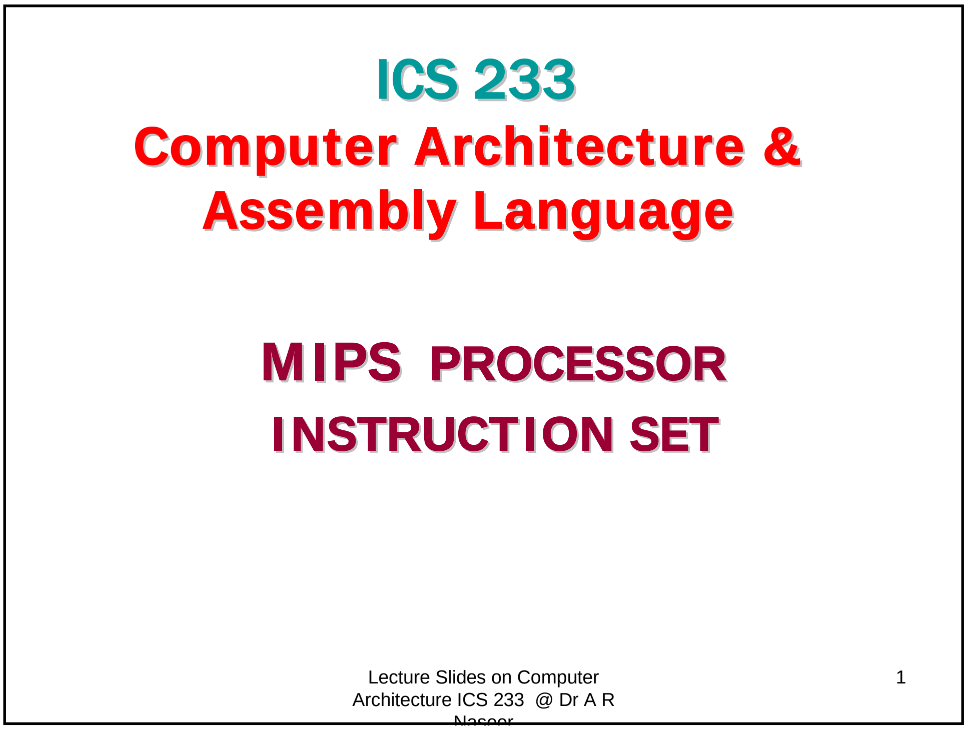
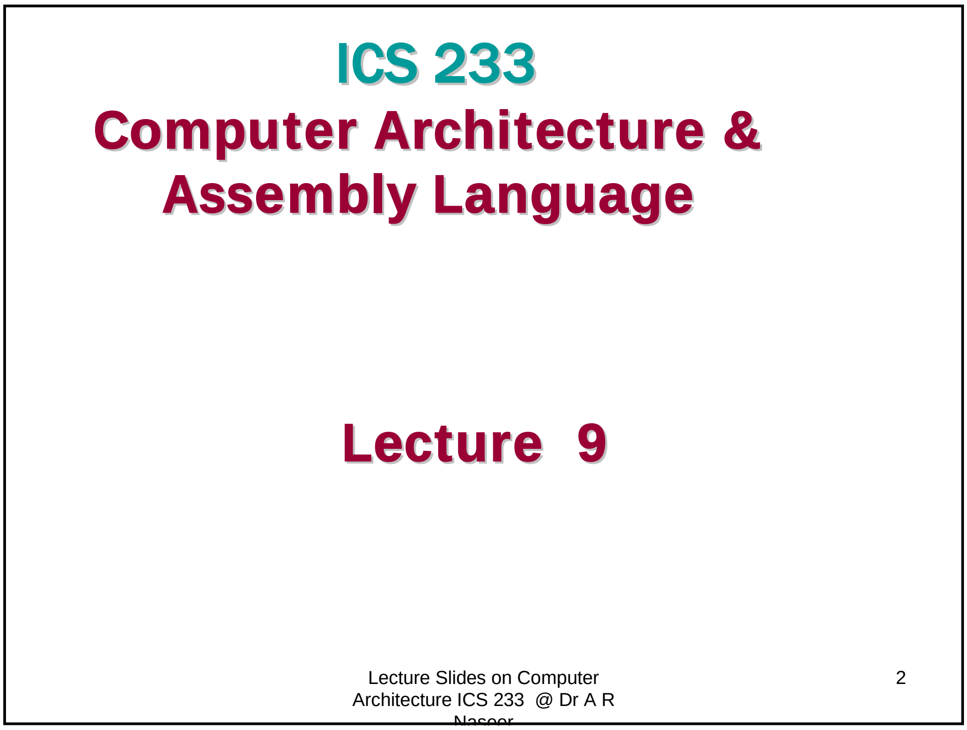


ICS 233
Computer Architecture & Assembly Language

MIPS PROCESSOR INSTRUCTION SET



ICS 233
Computer Architecture & Assembly Language

Lecture 9

Lecture Outline

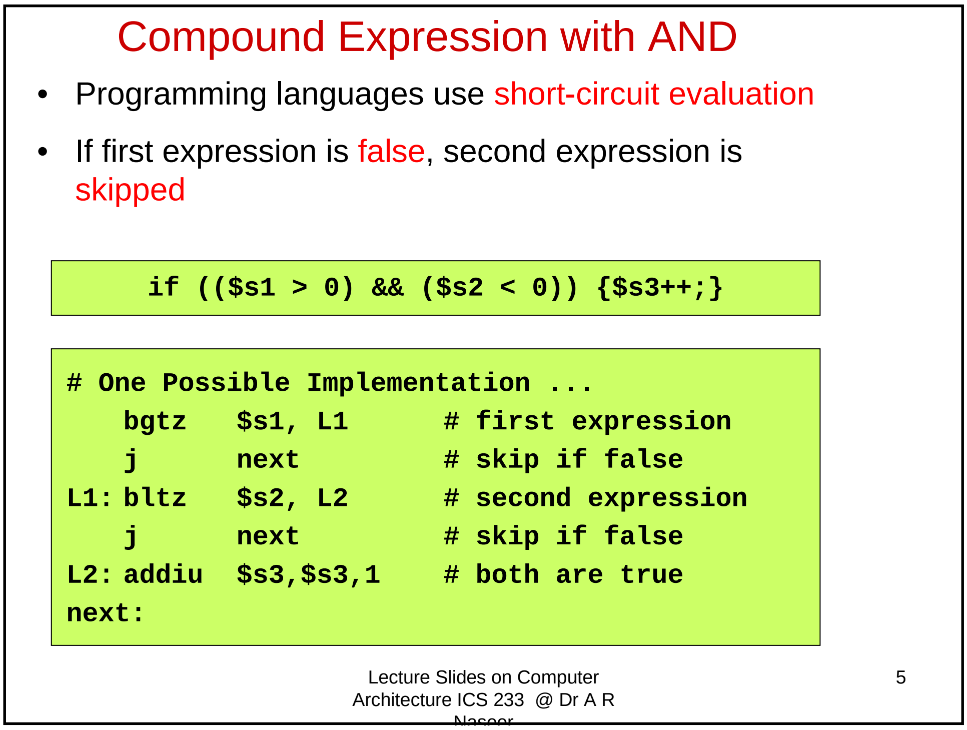
- ❑ **Translating IF Statement**
- ❑ **Translating WHILE loop**

Translating an IF Statement

- **Consider the following IF statement:**
  `if (a == b) c = d + e; else c = d – e;`
  Assume that a, b, c, d, e are in $s0, $s1, $s2, $s3, $s4 respectively

- **How to translate the above IF statement?**

```
        bne    $s0, $s1, else
        addu   $s2, $s3, $s4
        j      exit
else:   subu   $s2, $s3, $s4
exit:   . . .
```

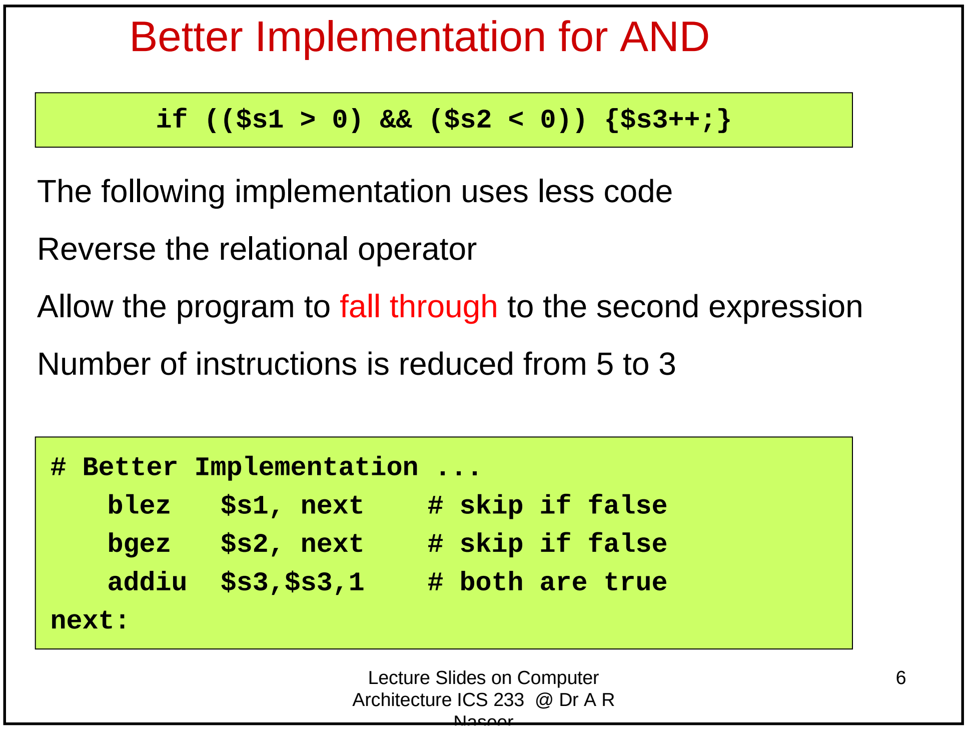


Compound Expression with AND

- Programming languages use short-circuit evaluation
- If first expression is false, second expression is skipped

```
if (($s1 > 0) && ($s2 < 0)) {$s3++;}
```

```
# One Possible Implementation ...
    bgtz   $s1, L1      # first expression
    j      next         # skip if false
L1: bltz   $s2, L2      # second expression
    j      next         # skip if false
L2: addiu  $s3,$s3,1    # both are true
next:
```



Better Implementation for AND

```
if (($s1 > 0) && ($s2 < 0)) {$s3++;}
```

The following implementation uses less code

Reverse the relational operator

Allow the program to fall through to the second expression

Number of instructions is reduced from 5 to 3

```
# Better Implementation ...
    blez   $s1, next    # skip if false
    bgez   $s2, next    # skip if false
    addiu  $s3,$s3,1    # both are true
next:
```

Compound Expression with OR

- Short-circuit evaluation for logical OR
- If first expression is true, second expression is skipped

```
    if (($sl > $s2) || ($s2 > $s3)) {$s4 = 1;}
```

- Use fall-through to keep the code as short as possible

```
    bgt $s1, $s2, L1    # yes, execute if part
    ble $s2, $s3, next  # no: skip if part
L1: ori  $s4, $0, 1     # set $s4 to 1
next:
```

- `bgt`, `ble` are pseudo-instructions
  - Translated by the assembler to real instructions

TRY THIS …..

- Translate the IF statement to assembly language
- $s1 and $s2 values are unsigned

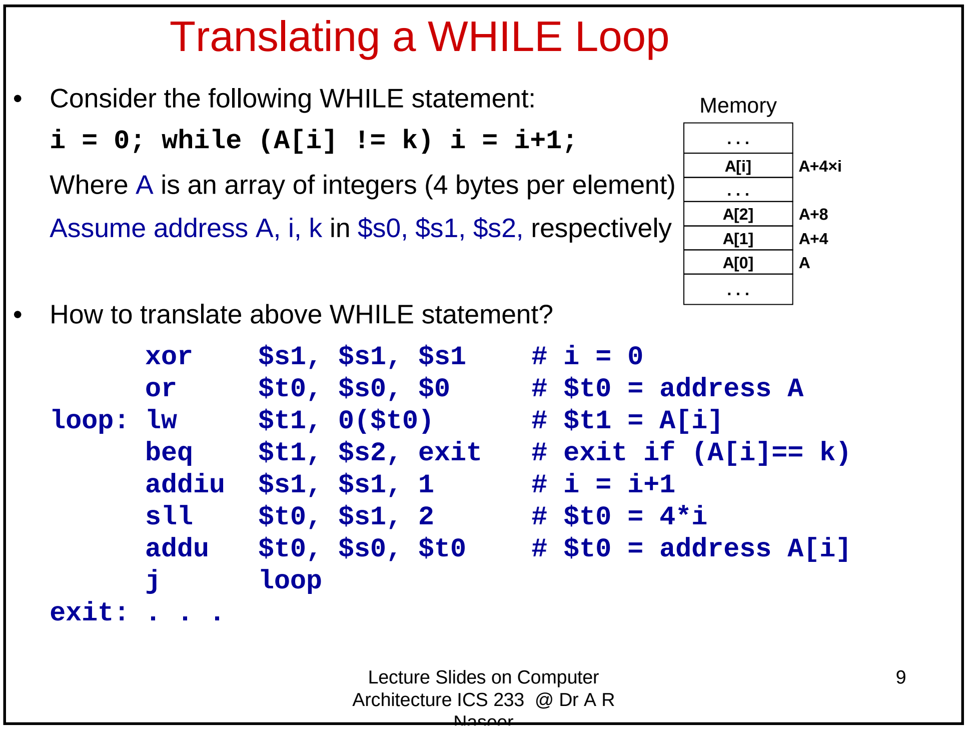
```
if( $s1 <= $s2 ) {
  $s3 = $s4
}
```

```
   bgtu $s1, $s2, next
   or   $s3, $s4, $0
next:
```

- $s3, $s4, and $s5 values are signed

```
if (($s3 <= $s4) &&
    ($s4 >  $s5)) {
  $s3 = $s4 + $s5
}
```

```
   bgt  $s3, $s4, next
   ble  $s4, $s5, next
   addu $s3, $s4, $s5
next:
```



Translating a WHILE Loop

- Consider the following WHILE statement:

  `i = 0; while (A[i] != k) i = i+1;`

  Where A is an array of integers (4 bytes per element)

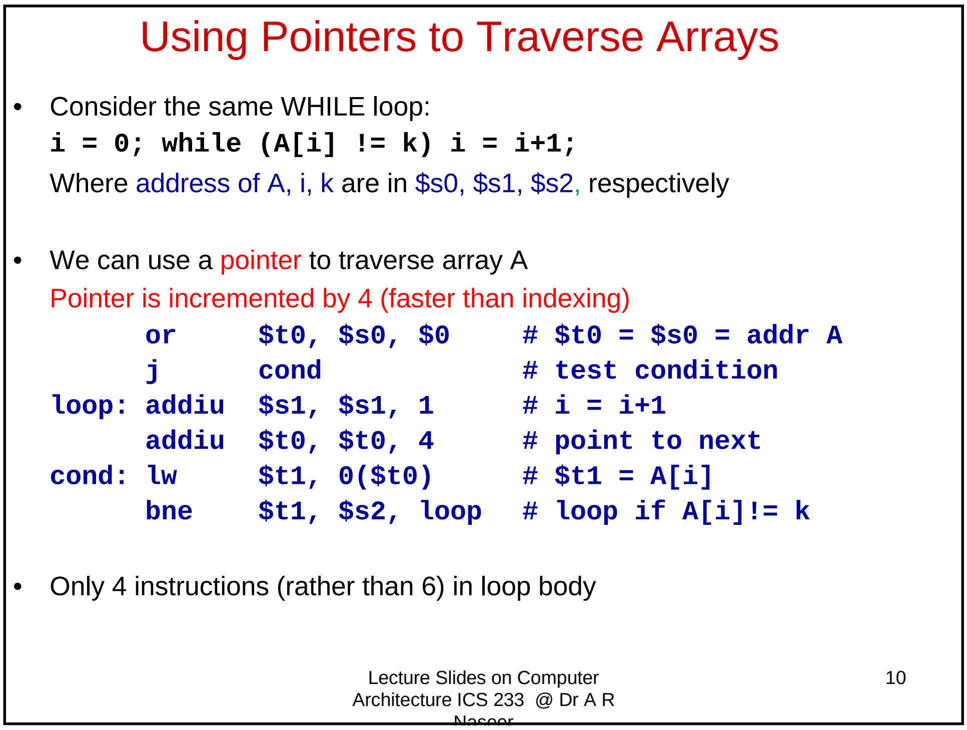
  Assume address A, i, k in $s0, $s1, $s2, respectively

Memory

. . .	
A[i]	A+4×i
. . .	
A[2]	A+8
A[1]	A+4
A[0]	A
. . .	

- How to translate above WHILE statement?

```
      xor    $s1, $s1, $s1    # i = 0
      or     $t0, $s0, $0     # $t0 = address A
loop: lw     $t1, 0($t0)      # $t1 = A[i]
      beq    $t1, $s2, exit   # exit if (A[i]== k)
      addiu  $s1, $s1, 1      # i = i+1
      sll    $t0, $s1, 2      # $t0 = 4*i
      addu   $t0, $s0, $t0    # $t0 = address A[i]
      j      loop
exit: . . .
```



Using Pointers to Traverse Arrays

- Consider the same WHILE loop:

  `i = 0; while (A[i] != k) i = i+1;`

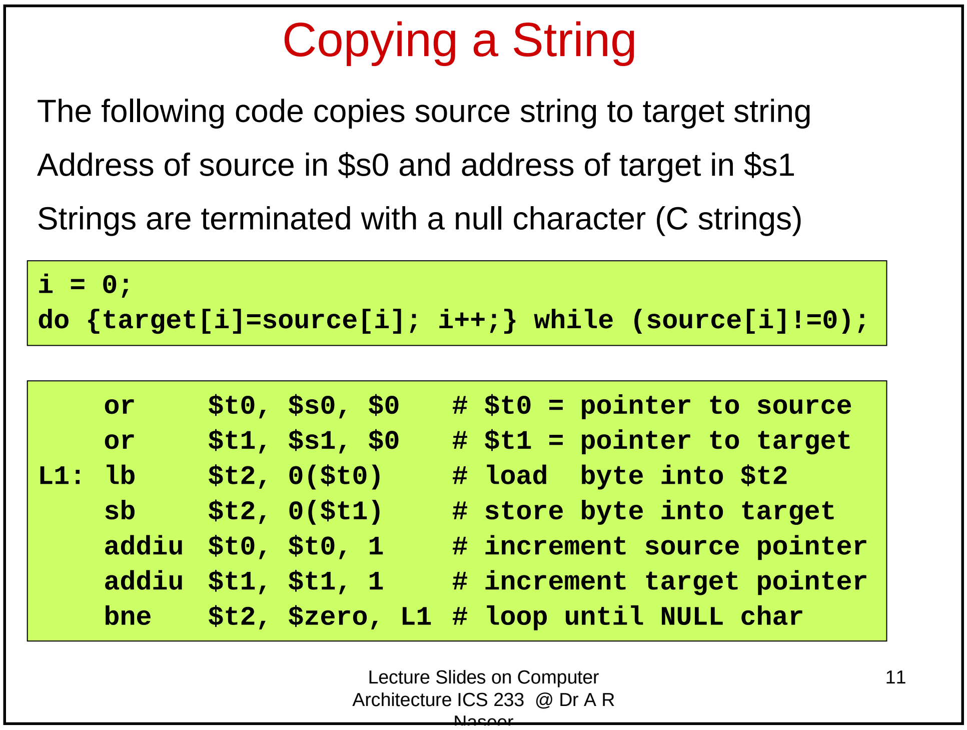
  Where address of A, i, k are in $s0, $s1, $s2, respectively

- We can use a pointer to traverse array A

  Pointer is incremented by 4 (faster than indexing)

```
      or     $t0, $s0, $0     # $t0 = $s0 = addr A
      j      cond             # test condition
loop: addiu  $s1, $s1, 1      # i = i+1
      addiu  $t0, $t0, 4      # point to next
cond: lw     $t1, 0($t0)      # $t1 = A[i]
      bne    $t1, $s2, loop   # loop if A[i]!= k
```

- Only 4 instructions (rather than 6) in loop body



Copying a String

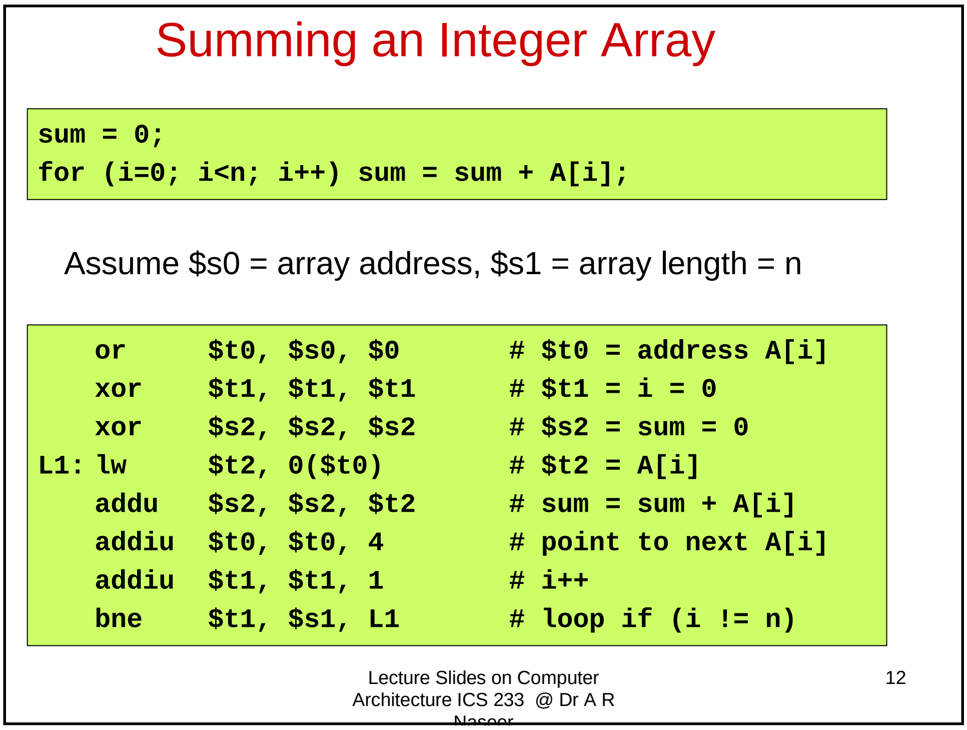
The following code copies source string to target string

Address of source in $s0 and address of target in $s1

Strings are terminated with a null character (C strings)

```
i = 0;
do {target[i]=source[i]; i++;} while (source[i]!=0);
```

```
    or     $t0, $s0, $0    # $t0 = pointer to source
    or     $t1, $s1, $0    # $t1 = pointer to target
L1: lb     $t2, 0($t0)     # load  byte into $t2
    sb     $t2, 0($t1)     # store byte into target
    addiu  $t0, $t0, 1     # increment source pointer
    addiu  $t1, $t1, 1     # increment target pointer
    bne    $t2, $zero, L1  # loop until NULL char
```



Summing an Integer Array

```
sum = 0;
for (i=0; i<n; i++) sum = sum + A[i];
```

Assume $s0 = array address, $s1 = array length = n

```
    or     $t0, $s0, $0       # $t0 = address A[i]
    xor    $t1, $t1, $t1      # $t1 = i = 0
    xor    $s2, $s2, $s2      # $s2 = sum = 0
L1: lw     $t2, 0($t0)        # $t2 = A[i]
    addu   $s2, $s2, $t2      # sum = sum + A[i]
    addiu  $t0, $t0, 4        # point to next A[i]
    addiu  $t1, $t1, 1        # i++
    bne    $t1, $s1, L1       # loop if (i != n)
```