Computer Architecture

MIPS instruction: Examples and Demonstrations

Dr. Anirban Sengupta, CSE, Prof
Indian Institute of Technology Indore
Web: http://www.anirban-sengupta.com/

Registers Vs Memory

What is spilling registers?

✓ When there are more variables than registers, then compiler tries to keep most frequently used variable in registers

Why not keep all variables in memory?

✓ Smaller is faster: Registers are faster than memory

✓ MIPS arithmetic instructions can read two registers, operate on them, and write one register per instruction

✓ MIPS data transfer only read or write one operand per instruction, and no operation

Ref. Computer Architecture: A Quantitative Approach 5th Edition, The Morgan Kaufmann Series by John L. Hennessy , David A. Patterson

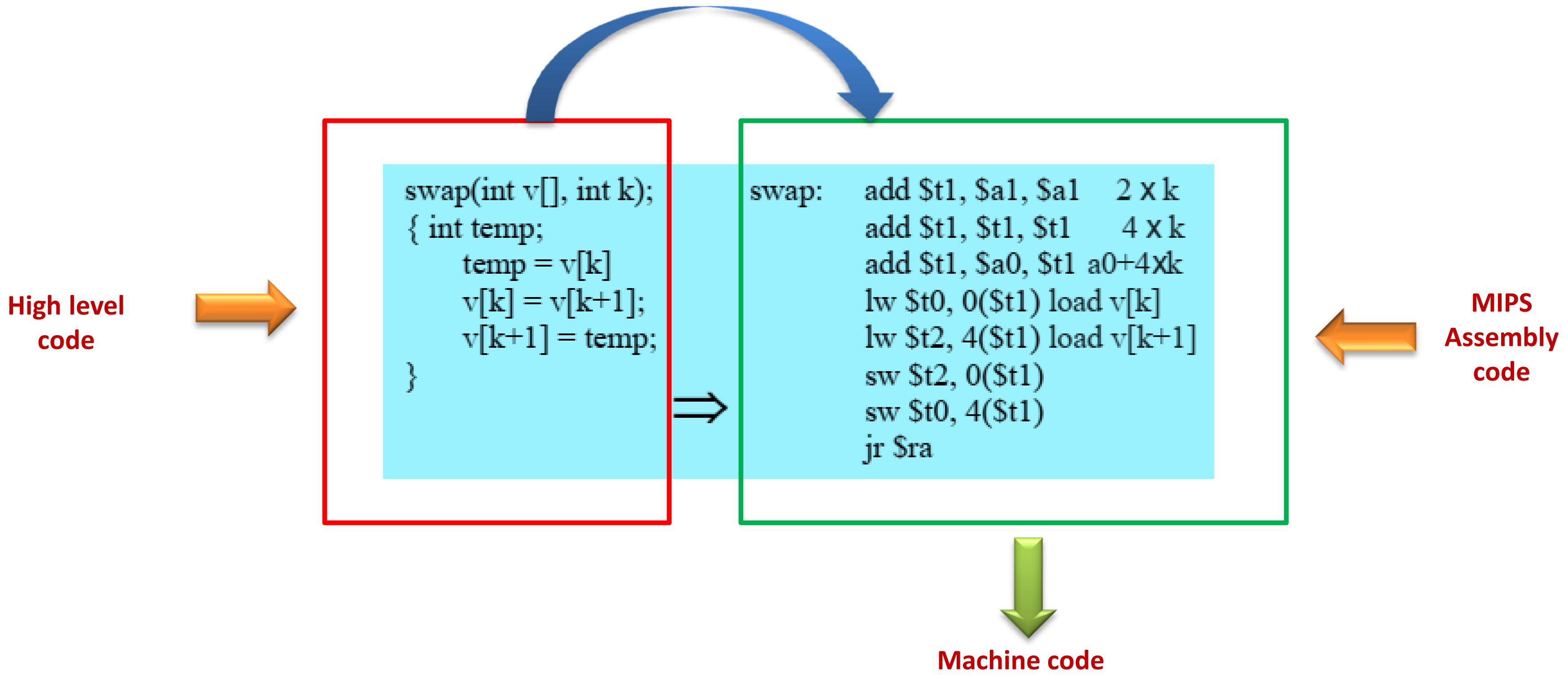
MIPS Instructions: Examples

A[8] = h + A[8];

```
MIPS code:
    lw      $t0, 32($s3)     A[8]
    add     $t0, $s2, $t0   h + A[8]
    sw      $t0, 32($s3)
```

- ❖ h is associated with register $s2
- ❖ base address for A[i] is in register $s3

MIPS Instructions: Examples



```
swap(int v[], int k);
{ int temp;
      temp = v[k]
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

```
swap:    add $t1, $a1, $a1     2 x k
         add $t1, $t1, $t1      4 x k
         add $t1, $a0, $t1  a0+4xk
         lw $t0, 0($t1) load v[k]
         lw $t2, 4($t1) load v[k+1]
         sw $t2, 0($t1)
         sw $t0, 4($t1)
         jr $ra
```

Ref. Computer Architecture: A Quantitative Approach 5th Edition, The Morgan Kaufmann Series
by John L. Hennessy , David A. Patterson

MIPS Branching: Examples

- MIPS processor conditional instructions:
- Branch on not equal
- Branch on equal
- I-type instruction

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

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

bne $s0, $s1, Label
add $s3, $s0, $s1

Label:  ....
```

MIPS Branching: Examples

C-code

```
if (i ! = j)
    h = i + j;
else
    h = i - j;
```

```
$s4 <= i, $s5 <= j, $s3 <= h
      beq $s4, $s5, Lab1
      add $s3, $s4, $s5
      j Label2
Label1:     sub $s3, $s4, $s5
Label2:     ...
```

MIPS code

Ref. Computer Architecture: A Quantitative Approach 5th Edition, The Morgan Kaufmann Series by John L. Hennessy , David A. Patterson

Loading 32-bits constants into a register

Lui $t0, 1010101010101010 **16-bit value**

ori $t0, $t0, 1010101010101011

1010101010101010 | 0000000000000000

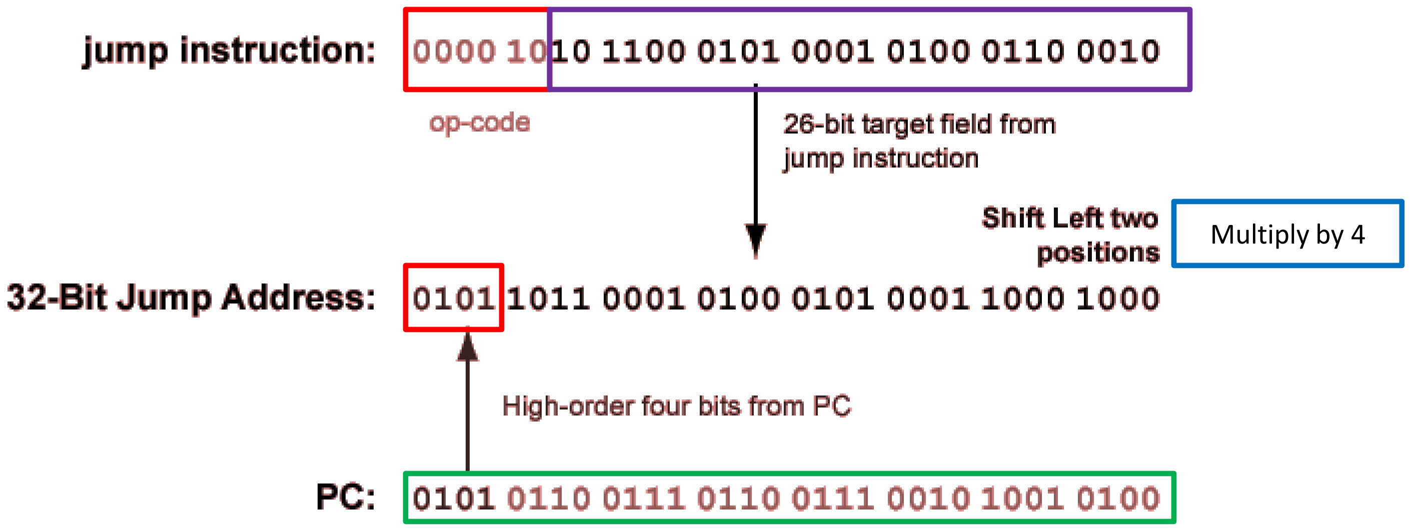
0000000000000000 | 1010101010101011

1010101010101010 | 1010101010101011

32-bit value loaded

Ref. Computer Architecture: A Quantitative Approach 5th Edition, The Morgan Kaufmann Series by John L. Hennessy , David A. Patterson

Calculate Jump Target Address



1. Upper 4-bit (31:28) of current PC+4
2. 26-bit of immediate field of jump instruction
3. Lower order bits 00

Opcode of Jump instruction: 000010

Ref. Computer Architecture: A Quantitative Approach 5th Edition, The Morgan Kaufmann Series by John L. Hennessy , David A. Patterson

Calculating target address of Branch

I-type formats for branches

Immediate (target label) = [target address – (PC+4)] / 4

4 * Immediate + (PC+4) = target address

e.g. $s1, $s2, 25

Here '25' is the target label and not target address

So target address is calculated as:

TA = (4*25) +(PC+4)

Calculating target address of Jump

J-type formats for jump

Immediate (target label) = [target address] / 4

4 * Immediate = target address

Note: this is not 32-bit address. This only forms the lower 28-bits of the address. To form 32-bit address updated PC must be concatenated with the upper 4-bits.

e.g. j 2500

Here ‘2500’ is the target label and not target address.
So target address is calculated as:
TA = (4*2500) = 10000

e.g. jal 2500

Here ‘2500’ is the target label and not target address.

$ra = PC+4 goto 10000 TA

Ref. Computer Architecture: A Quantitative Approach 5th Edition, The Morgan Kaufmann Series by John L. Hennessy , David A. Patterson

Example of MIPS code to machine code

C-code

```
While (save [i] = = k)
i = i + 1;
```

Assuming:

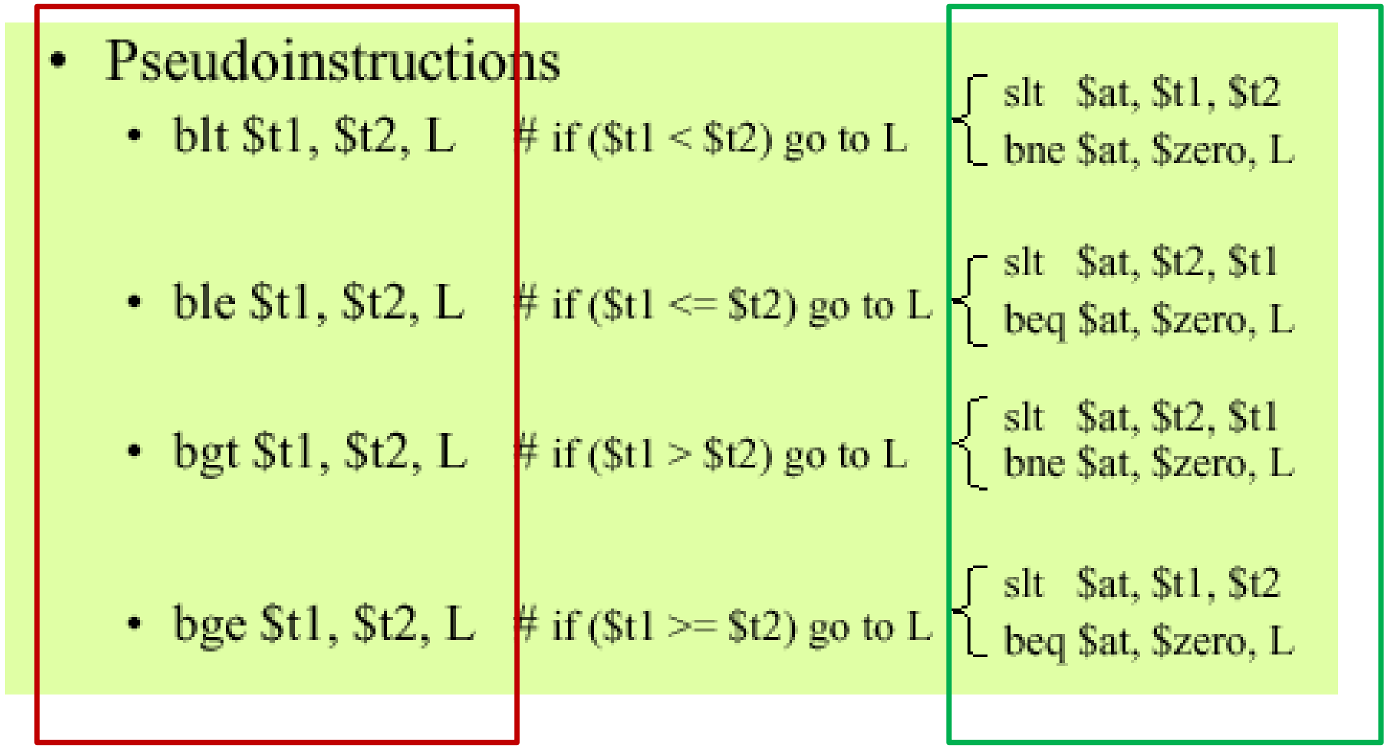
- reg $s6 stores base address of save [ ]
- i → $s3
- k → $ s5

MIPS code

```
L: sll    $t1, $s3, 2          # temp = reg $t1 = 4 * i
          add  $t1, $t1, $s6   # $t1 = address of save [i]
          lw $t0, 0($t1)       # temp reg $t0 = save [i]
          bne $t0, $s5 Exit    # goto Exit if save [i] ≠ k
          addi $s3, $s3, 1     # i++
          j L                  # goto L
Exit:
```

Explanation of each instruction

Pseudo MIPS instruction



- Pseudoinstructions
  - blt \$t1, \$t2, L # if (\$t1 < \$t2) go to L
  - ble \$t1, \$t2, L # if (\$t1 <= \$t2) go to L
  - bgt \$t1, \$t2, L # if (\$t1 > \$t2) go to L
  - bge \$t1, \$t2, L # if (\$t1 >= \$t2) go to L

```
slt  $at, $t1, $t2
bne  $at, $zero, L

slt  $at, $t2, $t1
beq  $at, $zero, L

slt  $at, $t2, $t1
bne  $at, $zero, L

slt  $at, $t1, $t2
beq  $at, $zero, L
```

Translation

Pseudo instruction

Pseudo MIPS instruction

Pseudo instruction

Translation

mov $rt, $rs	addi $rt, $rs, 0
li $rs, small	addi $rs, $0, small
li $rs, big	lui $rs, upper (big) ori $rs, $rs, lower (big)
la $rs, big	lui $rs, upper (big) ori $rs, $rs, lower (big)
lw $rt, big($rs)	lui $t0, upper (big) ori $t0, $t0, lower (big) add $t0, $rs, $t0 lw $rt, 0($t0)