

# MIPS R3000 Instruction Set Architecture

- Instruction Categories

- Computational
- Load/Store
- Jump and Branch
- Floating Point
  - coprocessor
- Memory Management
- Special

## Registers

**R0 - R31**

**PC**

**HI**

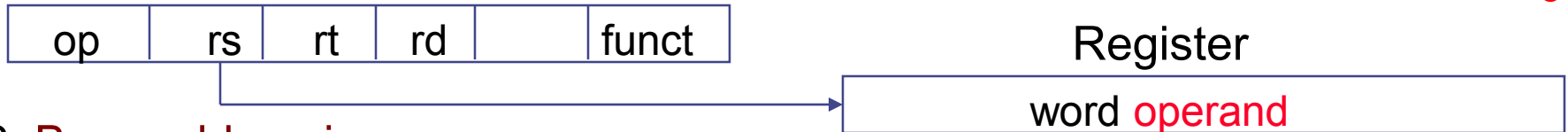
**LO**

## 3 Instruction Formats: **all 32 bits wide**

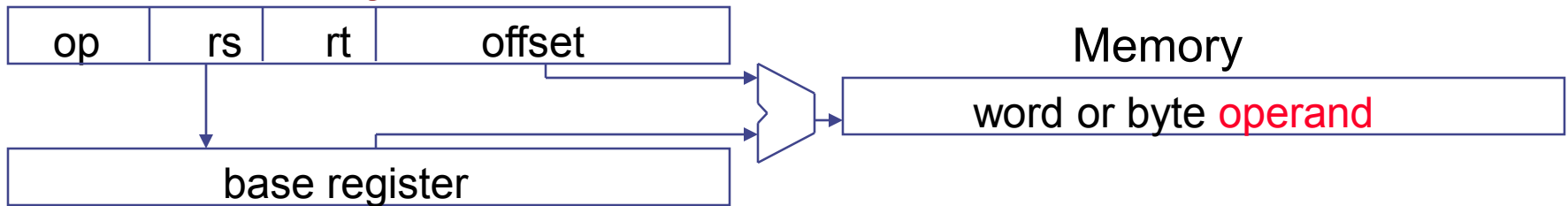
OP	rs	rt	rd	sa	funct	R format
OP	rs	rt	immediate			I format
OP	jump target					J format

# MIPS Addressing Modes

## 1. Register addressing



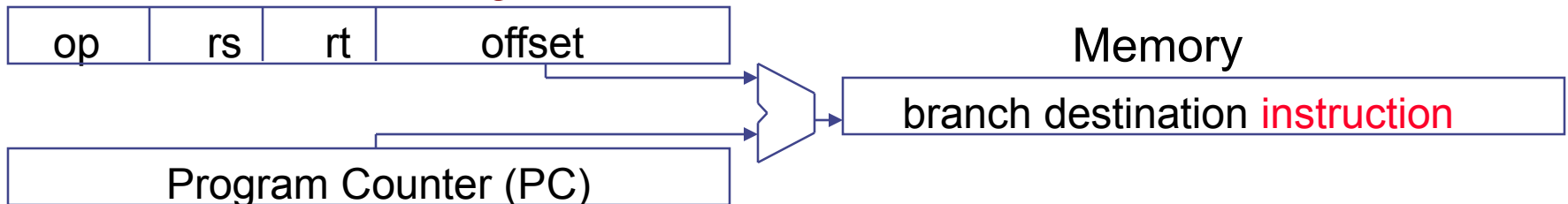
## 2. Base addressing



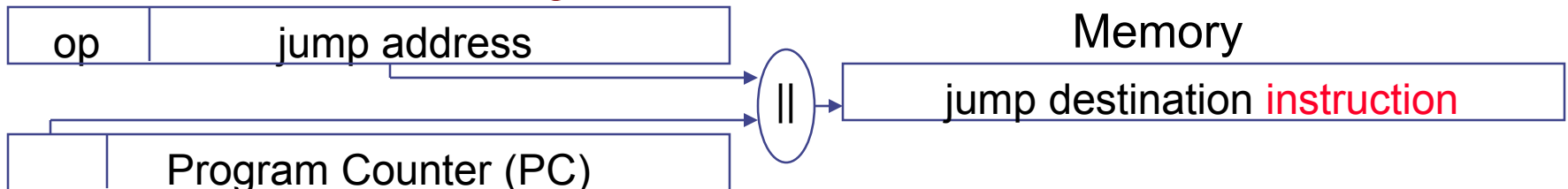
## 3. Immediate addressing



## 4. PC-relative addressing



## 5. Pseudo-direct addressing



# MIPS Register Convention

Name	Register Number	Usage	Preserve on call?
\$zero	0	constant 0 ( <b>hardware</b> )	n.a.
\$at	1	<b>reserved</b> for assembler	n.a.
\$v0 - \$v1	2-3	returned values	no
\$a0 - \$a3	4-7	arguments	<b>yes</b>
\$t0 - \$t7	8-15	temporaries	no
\$s0 - \$s7	16-23	saved values	<b>yes</b>
\$t8 - \$t9	24-25	temporaries	no
\$gp	28	global pointer	<b>yes</b>
\$sp	29	stack pointer	<b>yes</b>
\$fp	30	frame pointer	<b>yes</b>
\$ra	31	return addr ( <b>hardware</b> )	<b>yes</b>

# A MIPS Sample Program

C program

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;

    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

## MIPS Assy Program

```
.text
.align 2
.globl main

main:
    subu    $sp, $sp, 32
    sw      $ra, 20($sp)
    sd      $a0, 32($sp)
    sw      $0, 24($sp)
    sw      $0, 28($sp)

loop:
    lw      $t6, 28($sp)
    mul     $t7, $t6, $t6
    lw      $t8, 24($sp)
    addu    $t9, $t8, $t7
    sw      $t9, 24($sp)
    addu    $t0, $t6, 1
    sw      $t0, 28($sp)
    ble     $t0, 100, loop
    la      $a0, str
    lw      $a1, 24($sp)
    jal     printf
    move    $v0, $0
    lw      $ra, 20($sp)
    addu    $sp, $sp, 32
    jr      $ra

.data
.align 0

str:
    .asciiz "The sum from 0 .. 100 is %d\n"
```

## Machine code Memory Dump

```
001001111011110111111111111100000
101011111011111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
1010111110100000000000000011000
1010111110100000000000000011100
1000111110101110000000000011100
1000111110111000000000000011000
0000000111001110000000000011001
0010010111001000000000000000001
00101001000000010000000001100101
1010111110101000000000000011100
000000000000000000111100000010010
000000011000011111100100000100001
0001010000100000111111111110111
1010111110111001000000000011000
0011110000000100000100000000000
1000111110100101000000000011000
0000110000010000000000011101100
0010010010000100000001000110000
1000111110111111000000000010100
0010011110111101000000000100000
000000111110000000000000001000
00000000000000000001000000100001
```

## Reverse Engineered Code

```
addiu    $29, $29, -32
sw        $31, 20($29)
sw        $4, 32($29)
sw        $5, 36($29)
sw        $0, 24($29)
sw        $0, 28($29)
lw        $14, 28($29)
lw        $24, 24($29)
multu    $14, $14
addiu    $8, $14, 1
slti     $1, $8, 101
sw        $8, 28($29)
mflo     $15
addu     $25, $24, $15
bne      $1, $0, -9
sw        $25, 24($29)
lui      $4, 4096
lw        $5, 24($29)
jal      1048812
addiu    $4, $4, 1072
lw        $31, 20($29)
addiu    $29, $29, 32
jr        $31
move     $2, $0
```

# Supporting Procedures

---

## Process:

- Place parameters where procedure can access them
- Transfer control to the procedure
- Acquire storage resources for the procedure
- Perform the task
- Place result where calling program can access it
- Return control to calling program

## Support structure:

- \$a0-\$a3 argument passing registers
- \$v0-\$v1 return value registers
- \$ra return address register

# MIPS Arithmetic Instructions

- MIPS assembly language arithmetic statement

add \$t0, \$s1, \$s2

sub \$t0, \$s1, \$s2

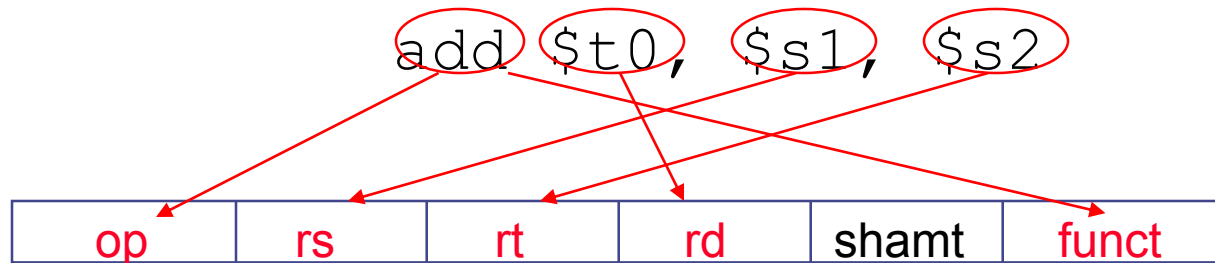
- ❑ Each arithmetic instruction performs only **one** operation
- ❑ Each arithmetic instruction fits in 32 bits and specifies exactly **three** operands

destination ← source1 **op** source2

- ❑ Operand order is fixed (destination first)
- ❑ Those operands are **all** contained in the datapath's **register file** (\$t0, \$s1, \$s2) – indicated by \$

# Machine Language - Add Instruction

- Instructions, like registers and words of data, are 32 bits long
- Arithmetic Instruction Format (**R** format):



op	6-bits	<b>o</b> pcode that specifies the operation
rs	5-bits	<b>r</b> egister file address of the first <b>s</b> ource operand
rt	5-bits	<b>r</b> egister file address of the second source operand
rd	5-bits	<b>r</b> egister file address of the result's <b>d</b> estination
shamt	5-bits	<b>s</b> hift <b>a</b> mount (for shift instructions)
funct	6-bits	<b>f</b> unction code augmenting the opcode

# MIPS Immediate Instructions

```
addi $sp, $sp, 4      # $sp = $sp + 4
slti $t0, $s2, 15     # $t0 = 1 if $s2 < 15
```

- Machine format (**I** format):

- ❑ Small constants are used often in typical code

- ❑ Possible approaches?

- put “typical constants” in memory and load them
- create hard-wired registers (like \$zero) for constants like 1
- have special instructions that contain constants !



- ❑ The constant is kept **inside** the instruction itself!

- Immediate format **limits** values to the range  $+2^{15}-1$  to  $-2^{15}$



# MIPS Memory Access Instructions

---

- MIPS has two basic **data transfer** instructions for accessing memory

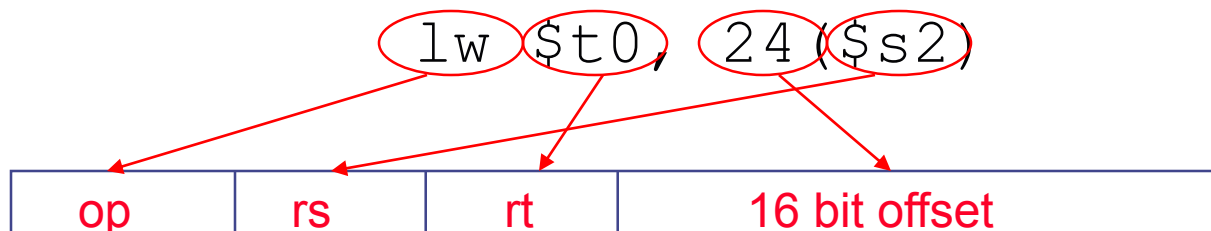
`lw     $t0, 4($s3)     #load word from memory`

`sw     $t0, 8($s3)     #store word to memory`

- The data is loaded into (**lw**) or stored from (**sw**) a register in the register file – a 5 bit address
- The memory address – a 32 bit address – is formed by adding the contents of the **base address register** to the **offset** value
  - A 16-bit field meaning access is limited to memory locations within a region of  $\pm 2^{13}$  or 8,192 words ( $\pm 2^{15}$  or 32,768 bytes) of the address in the base register
  - Note that the offset can be positive or negative

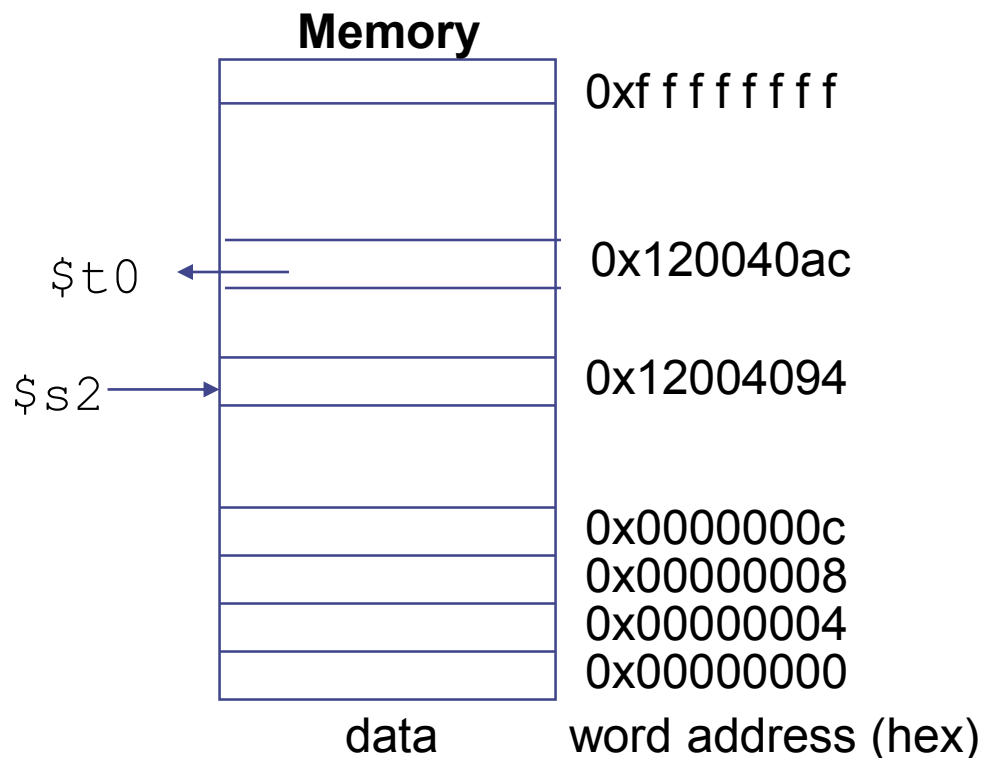
# Machine Language - Load Instruction

- Load/Store Instruction Format (**I** format):



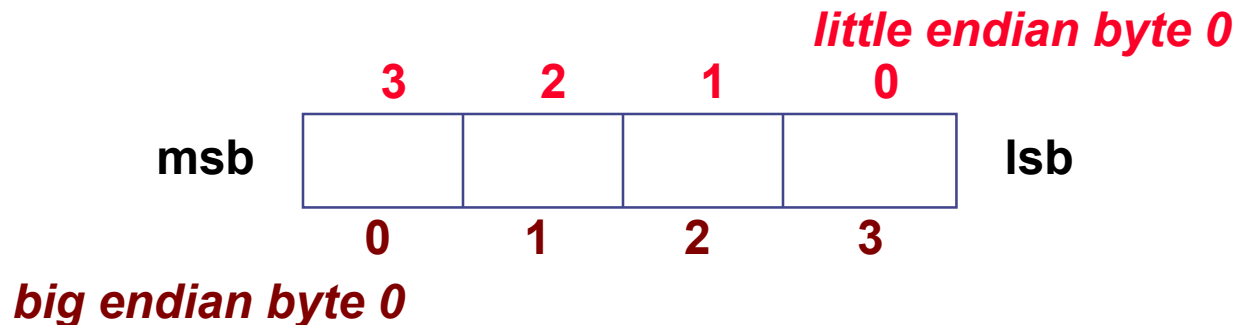
$$24_{10} + \$s2 =$$

$$\begin{array}{r} 0x00000018 \\ + 0x12004094 \\ \hline 0x120040ac \end{array}$$



# Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual **bytes** in memory
  - The memory address of a **word** must be a multiple of 4 (**alignment restriction**)
- **Big Endian**: leftmost byte is word address  
IBM 360/370, Motorola 68k, **MIPS**, Sparc, HP PA
- **Little Endian**: rightmost byte is word address  
Intel 80x86, DEC Vax, DEC Alpha (Windows NT)



# Loading and Storing Bytes

- MIPS provides special instructions to move bytes

lb      \$t0, 1(\$s3)    #load byte from memory

sb      \$t0, 6(\$s3)    #store byte to memory



## □ What 8 bits get loaded and stored?

- load byte places the byte from memory in the rightmost 8 bits of the destination register
  - what happens to the other bits in the register?
- store byte takes the byte from the rightmost 8 bits of a register and writes it to a byte in memory
  - what happens to the other bits in the memory word?

# MIPS Control Flow Instructions

- MIPS **conditional branch** instructions:

bne \$s0, \$s1, Lbl1 #go to Lbl1 if \$s0≠\$s1

beq \$s0, \$s1, Lbl1 #go to Lbl1 if \$s0=\$s1

- Ex:       if (i==j) h = i + j;

      bne \$s0, \$s1, Lbl1

      add \$s3, \$s0, \$s1

Lbl1:       ...

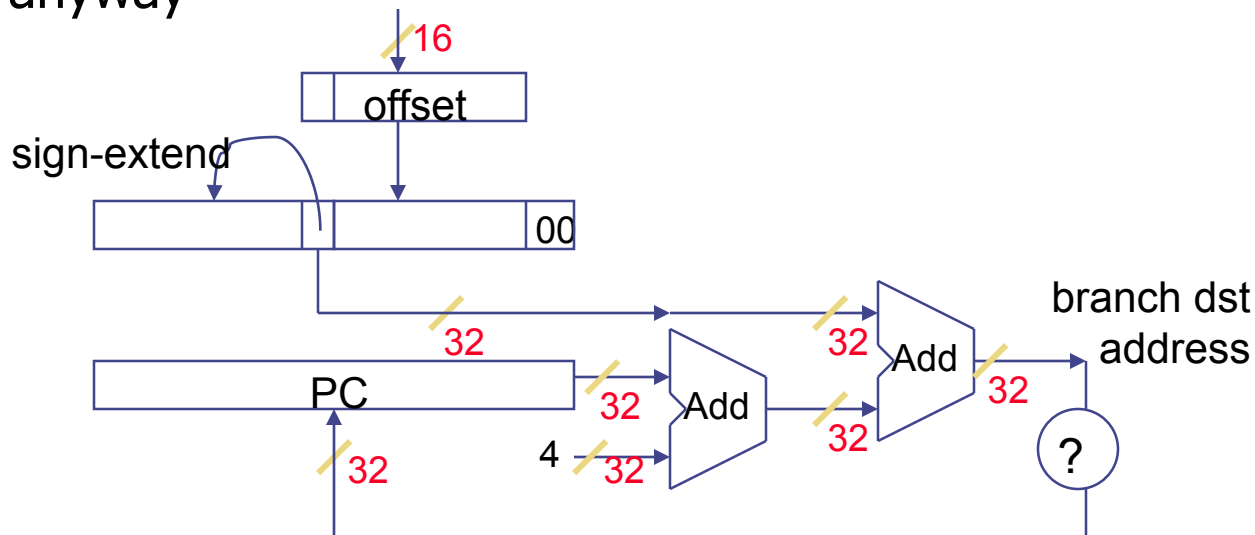
- Instruction Format (I format):



- How is the branch destination address specified?

# Specifying Branch Destinations

- Use a register (like in lw and sw) added to the 16-bit offset
  - which register? Instruction Address Register (the PC)
    - its use is automatically **implied** by instruction
    - PC gets updated (PC+4) during the **fetch** cycle so that it holds the address of the next instruction
  - limits the branch distance to  **$-2^{15}$  to  $+2^{15}-1$**  instructions from the (instruction after the) branch instruction, but most branches are local anyway



# Other Control Flow Instructions

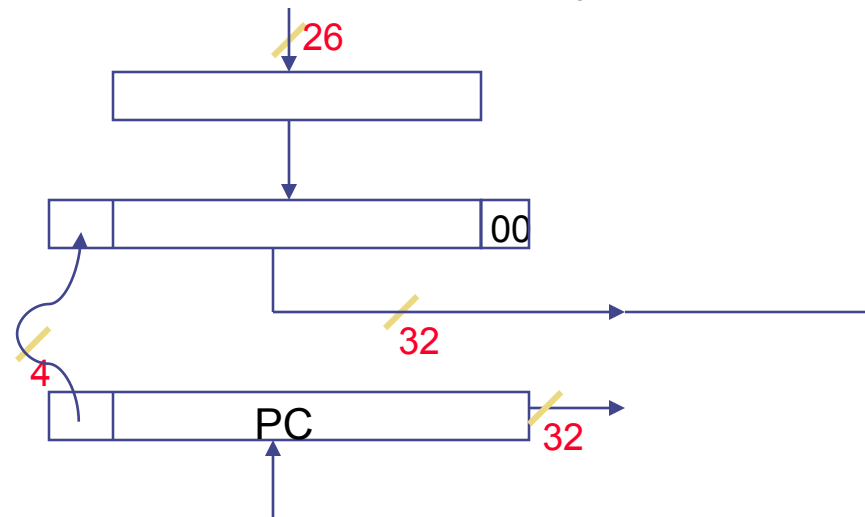
- MIPS also has an unconditional branch instruction or **jump** instruction:

`j label                      #go to label`

## ❑ Instruction Format (**J** Format):



from the low order 26 bits of the jump instruction



# Instructions for Accessing Procedures

- MIPS **procedure call** instruction:

`jal ProcedureAddress #jump and link`

- Saves PC+4 in register \$ra to have a link to the next instruction for the procedure return
- Machine format (**J** format):



- Then can do procedure **return** with a

`jr $ra #return`

- Instruction format (**R** format):





# MIPS ISA - First look

Category	Instr	Op Code	Example	Meaning
Arithmetic (R & I format)	add	0 and 32	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$
	subtract	0 and 34	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$
	add immediate	8	addi \$s1, \$s2, 6	$\$s1 = \$s2 + 6$
	or immediate	13	ori \$s1, \$s2, 6	$\$s1 = \$s2 \vee 6$
Data Transfer (I format)	load word	35	lw \$s1, 24(\$s2)	$\$s1 = \text{Memory}(\$s2+24)$
	store word	43	sw \$s1, 24(\$s2)	$\text{Memory}(\$s2+24) = \$s1$
	load byte	32	lb \$s1, 25(\$s2)	$\$s1 = \text{Memory}(\$s2+25)$
	store byte	40	sb \$s1, 25(\$s2)	$\text{Memory}(\$s2+25) = \$s1$
	load upper imm	15	lui \$s1, 6	$\$s1 = 6 * 2^{16}$
Cond. Branch & R format)	br on equal	4	beq \$s1, \$s2, L	if ( $\$s1 == \$s2$ ) go to L
	br on not equal	5	bne \$s1, \$s2, L	if ( $\$s1 != \$s2$ ) go to L
	set on less than	0 and 42	slt \$s1, \$s2, \$s3	if ( $\$s2 < \$s3$ ) $\$s1=1$ else $\$s1=0$
	set on less than immediate	10	slti \$s1, \$s2, 6	if ( $\$s2 < 6$ ) $\$s1=1$ else $\$s1=0$
Uncond. Jump (J & R format)	jump	2	j 2500	go to 10000
	jump register	0 and 8	jr \$t1	go to \$t1
	jump and link	3	jal 2500	go to 10000; \$ra=PC+4