A Minimalistic Introduction to MIPS Instruction

General Format

```
31     26    21    16    11    6      0
 ______ _____ _____ _____ _____ ______
|______|_____|_____|_____|_____|______|
   OP     RS    RT    RD  SHAMT  FUNC
```

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op
    Operation code

rs
    First source register operand

rt
    Second source register operand

rd
    Destination register operand

shamt
    Shift amount - used in shift instructions

funct
    Select the variant of the operation in the op code field

Specific Instruction Formats

Format	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Comments
R	op	rs	rt	rd	shamt	funct	Arithmetic
I	op	rs	rt	address/immediate			Transfer, branch,immediate
J	op	target address					Jump

MIPS Instruction Set

The MIPS instruction set illustrates four underlying principles of hardware design:

1. Simplicity favors regularity.
2. Smaller is faster.
3. Good design demands compromise.

4. Make the common case fast.

Simplicity favors regularity

Consider the following example:

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add a,b,c	a=b+c	Always 3 operands
Arithmetic	subtract	sub a,b,c	a=b-c	Always 3 operands

Note that each operand has *exactly* three operands.

Smaller is faster.

MIPS has 32 32-bit registers,*$v0,...$v31*, a very large number would increase the clock cycle time.

Good design demands compromise.

The compromise represented by the MIPS design, was to make all the instructions the *same* length, thereby requiring different instruction formats.

Make the common case fast.

The MIPS instruction set addresses this principal by making constants part of arithmetic instructions. Furthermore, by loading small constants into the upper 16-bits of a register.

MIPS Instruction Set Summary

Note the summary is **not** complete. Click here to see the full list

Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add $1,$2,$3	$1=$2+$3	Always 3 operands
subtract	sub $1,$2,$3	$1=$2-$3	Always 3 operands
add immediate	addi $1,$2,10	$1=$2+10	add constant
add unsigned	addu $1,$2,$3	$1=$2+$3	Always 3 operations
subtract unsigned	subu $1,$2,$3	$1=$2-$3	Always 3 operations
add immed.unsigned	addiu $1,$2,10	$1=$2+10	Always 3 operations

Logical

Instruction	Example	Meaning	Comments
and	and $1,$2,$3	$1=$2&$3	3 register operands
or	or $1,$2,$3	$1=$2|$3	3 register operands
and immediate	andi $1,$2,10	$1=$2&10	AND constant

or immediate	or $1,$2,10	$1=$2|10	OR constant
shift left logical	sll $1,$2,10	$1=$2<<10	Shift left by constant
shift right logical	srl $1,$2,10	$1=$2>>10	Shift right by constant

Data Transfer

Instruction	Example	Meaning	Comments
load word	lw $1,10($2)	$1=Memory[$2+10]	memory to register
store word	sw $1,10($2)	Memory[$2+10]=$1	register to memory
load upper immed.	lui $1,10	$1=10x2^16	load constant into upper 16 bits

Conditional Branch

Instruction	Example	Meaning	Comments
branch on equal	beq $1,$2,10	if($1==$2)go to PC+4+10	Equal test
branch on not equal	bne $1,$2,10	if($1!=$2)go to PC+4+10	Not equal test
set on less then	slt $1,$2,$3	if($2<$3)$1=1;else $1=0	Less than compare

Unconditional Jump

Instruction	Example	Meaning	Comments
jump	j 1000	go to 1000	Jump to target address
jump register	jr $31	go to $31	For switch, procedure return
jump and link	jal 1000	$31=PC+4;go to 1000	For procedure call

Simple Output using MIPS I/O

Let us examine this simple program to introduce MIPS Assembler.

```
# hello.asm
# Simple routine to demo MIPS output.
# Author: R.N. Ciminero
# See Patterson & Hennessy pg. A-46 for system services.

        .text
        .globl  main
main:
        li      $v0,4           # code for print_str
        la      $a0, msg        # point to string
```

```
        syscall
        li      $v0,10          # code for exit
        syscall

        .data
msg:    .asciiz "Hello World!\n"
```

Output

```
Hello world!
```

Comments on the program:

 # hello.asm
: A single line *comment*, comments may also appear to the to the right of regular assembly language statements.

.text
: Begin program text segment.

.globl main
: Define a global function main.

main:
: A *label* **main**.

li $v0,4 # code for print_str
: Load immediate the system register,*$v0*, with a value of four, which indicates that we intend to output an ASCII string.

la $a0, msg # point to string
: Load the system register,*$a0*, with the address of the string to be output.

syscall
: System routine that performs the string output based on the contents of the system registers.

li $v0,10 # code for exit
: Load the system code for exit.

.data
: Begin the data segment of the program.

msg: .asciiz "Hello World!\n"
: The label *msg:* is the location of the ASCII string.

Simple Input

Let us examine the following program for simple *input*.

```
# simpleio.asm
```

```
# Simple routine to demo SPIM input/output.
# Author: R.N. Ciminero
# Revision date: 10-05-93 Original def.
# See Patterson & Hennessy pg. A-46 for system services.

        .text
        .globl  main
main:
        li      $v0,4           # output msg1
        la      $a0, msg1
        syscall
        li      $v0,5           # input A and save
        syscall
        move    $t0,$v0
        li      $v0,4           # output msg2
        la      $a0, msg2
        syscall
        li      $v0,5           # input B and save
        syscall
        move    $t1,$v0
        add     $t0, $t0, $t1   # A = A + B
        li      $v0, 4          # output msg3
        la      $a0, msg3
        syscall
        li      $v0,1           # output sum
        move    $a0, $t0
        syscall
        li      $v0,4           # output lf
        la      $a0, cflf
        syscall

        li      $v0,10          # exit
        syscall
        .data
msg1:   .asciiz "\nEnter A:   "
msg2:   .asciiz "\nEnter B:   "
msg3:   .asciiz "\nA + B =  "
cflf:   .asciiz "\n"
```

Output

```
Enter A: 10
Enter B: 5
A + B = 15
```

Comments on the program:

li $v0,5 # input A and save

Load the system register,*$v0*, with the input code.

syscall

System call to accept an *integer* from the keyboard and store it in register *$v0*.

move $t0,$v0

Macro to transfer the contents of system register *$v0* to temporary register *$t0*.

add $t0, $t0, $t1 # A = A + B

Add the contents of *$t0* to the contents of *$y1* and place the sum in *$t0*.

li $v0,1 # output sum

Load the system register *$v0* with the output *integer* code.

Looping Structures

Let us examine the following program for simple *looping* structures.

```
# sumit.asm
# Simple routine to sum N integers to demo a loop.
# Author: R.N. Ciminero
# Revision date: 10-06-93 Original def.
# See Patterson & Hennessy pg. A-46 for system services.

        .text
        .globl  main
main:
        li      $v0,4           # output msg1
        la      $a0, msg1
        syscall
        li      $v0,5           # input N and save
        syscall
        move    $t0,$v0

        li      $t1, 0          # initialize counter (i)
        li      $t2, 0          # initialize sum

loop:   addi    $t1, $t1, 1     # i = i + 1
        add     $t2, $t2, $t1   # sum = sum + i
        beq     $t0, $t1, exit  # if i = N, continue
        j       loop

exit:   li      $v0, 4          # output msg2
        la      $a0, msg2
        syscall

        li      $v0,1           # output sum
        move    $a0, $t2
        syscall
        li      $v0,4           # output lf
        la      $a0, lf
        syscall
        li      $v0,10          # exit
        syscall
        .data
msg1:   .asciiz "\nNumber of integers (N)?  "
msg2:   .asciiz "\nSum  =   "
lf:     .asciiz "\n"
```

Output

```
Number of integers (N)? 5
Sum = 15
```

Comments on the program:

li $t1, 0 # initialize counter (i)

Temporary register *$t1* contains the count.

li $t2, 0 # initialize sum
Temporary register *$t2* contains the sum.

loop: addi $t1, $t1, 1 # i = i + 1
Increment the counter by one.

add $t2, $t2, $t1 # sum = sum + i
Add the counter to the sum.

beq $t0, $t1, exit # if i = N, continue
If the counter equals the number of integers, then exit the loop.

j loop
Else perform the summation again.

exit: li $v0, 4 # output msg2
Statement to execute upon leaving the loop.