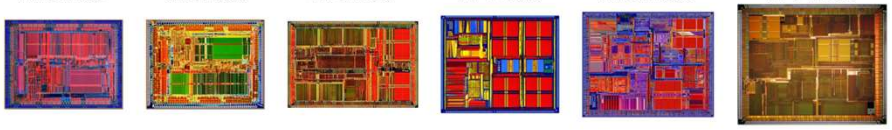
Computer Structure and Language

The MIPS Assembly Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

History



	MIPS R2000	MIPS R3000	MIPS R4000	MIPS R5000	MIPS R10000	MIPS R12000
Year	1985	1988	1992	1996	1995	1998
MIPS ISA	MIPS I (32-bit)	MIPS I (32-bit)	MIPS III (64-bit)	MIPS IV (64-bit)	MIPS IV (64-bit)	MIPS IV (64-bit)
Transistor count	110k	110k	2.3 – 4.6m	3.7m	6.8m	7.15m
Process node	2 µm	1.2 µm	0.35 µm	0.32 µm	0.35 µm	0.25 µm
Die size	80 mm²	40 mm²	84 – 100 mm²	84 mm²	350 mm²	229 mm²
Speed	12 – 33 MHz	20 – 40 MHz	50 - 250 MHz	150 – 266 MHz	180 – 360 MHz	270 – 400 MHz
Flagship devices	DECstation 2100 and 3100 workstations	***Sony PlayStation game console*** SGI IRIS and Indigo workstations ***NASA New Horizons space probe***	***Nintendo N64 game console*** Carrera Computers and DeskStation Technology PCs (Windows NT) SGI Onyx, Indigo, Indigo2, and Indy workstations	SGI O2 and Indy workstations Cobalt Qube servers HP LJ4000 laser printers	SGI Indigo2 and Octane workstations ***SGI Onyx and Onyx2 supercomputers*** NEC Cenju-4 supercomputers Siemens Nixdorf servers	SGI Octane 2, Onyx 2, and Origin workstations

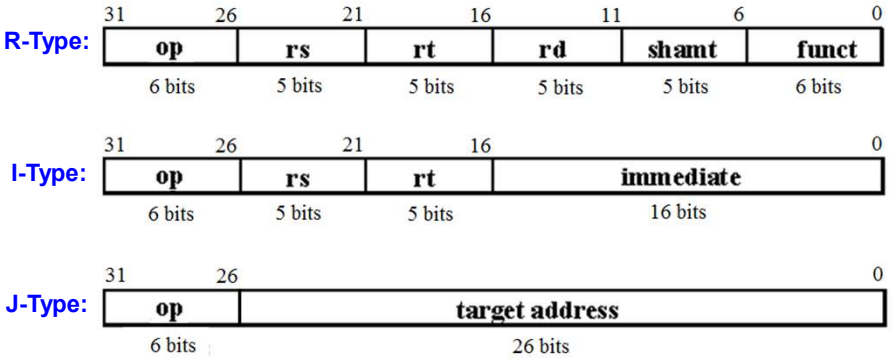
Features

- RISC architecture
  - Very simple
- 32 word-size registers named as $0, $1, …, $31
- 2^{32} bytes addressable main memory (Bi-Endianness)
- Addressing modes:
  - Data access
    - Immediate addressing (16-bit value)
    - Register direct
    - Base-displacement
  - Instruction access
    - PC-relative addressing as PC ← (PC) + 18-bit address
    - Pseudo-direct addressing as PC ← $(PC)_{31:28}$: 28-bit address
- Data type
  - 8/16/32 bits (byte, half-word, word) binary numbers

MIPS Registers

Name	Register Number	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	value for results and expressions	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

Instruction Format



op: *operation code* or *opcode* (0 for R-Type instructions)
funct: *function* (with opcode specifies the operation to perform)
shamt: *shift amount* for shift instructions (0 otherwise)
rs, rt: source registers
rd: destination register
immediate: 16-bit immediate data
target address: 26-bit address

MIPS supports 4 types of instructions:

1. **Data Transfer Instructions**
   load (lw,lh,lb), store (sw,sh,sb), load upper immediate (lui)
2. **Arithmetic and logical Instructions**
   add, sub, addi, subi, mult, div, and, or, andi, ori, sll, srl, …
3. **Conditional Branch**
   beq, bgez, bgezal, bgtz, blez, bltz, bltzal, bne,
4. **Unconditional Branch**
   j, jr, jal, syscall

Data Transfer Instructions

Load Instructions: lw, lh, lb **I-Format**

lw reg1,offset(reg2) ≡ reg1 ← $(M_{(reg2)+offset})_{word}$;
(reg2)+offset should be divisible by 4
Opcode: 100011

lh reg1, offset(reg2) ≡ reg1 ← $(M_{(reg2)+offset})_{16\text{-}bit}$;
(reg2)+offset should be divisible by 2
Opcode: 100001

lb reg1, offset(reg2) ≡ reg1 ← $(M_{(reg2)+offset})_{byte}$;
Opcode: 100000

Data Transfer Instructions

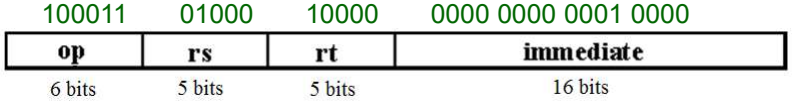
Load Instructions: lw, lh, lb

Example 1:

lw $s0,16($t0) ≡ $s0 ← $(M_{(\$t0)+16})_{word}$;

Opcode: 100011
rs: $t0 → 01000
rt: $s0 → 10000
immediate: 0x0010

Machine Code : 0x8d100010



Data Transfer Instructions

Store Instructions: sw, sh, sb **I-Format**

sw reg1,offset(reg2) ≡ $M_{(reg2)+offset} \leftarrow (reg1)$;
(reg2)+offset should be divisible by 4
Opcode: 101011

sh reg1, offset(reg2) ≡ $M_{(reg2)+offset} \leftarrow (reg1)_{15..0}$;
(reg2)+offset should be divisible by 2
Opcode: 101001

sb reg1, offset(reg2) ≡ $M_{(reg2)+offset} \leftarrow (reg1)_{7..0}$;
Opcode: 101000

Data Transfer Instructions

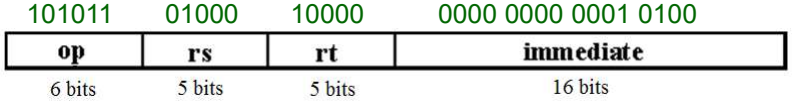
Store Instructions: sw, sh, sb

Example 1:

sw $s0,20($t0) ≡ $M_{(\$t0)+20} \leftarrow (\$s0)$;

Opcode: 101011
rs: $t0 → 01000
rt: $s0 → 10000
Immediate: 0x0014

Machine code : 0xAD100014



Data Transfer Instructions

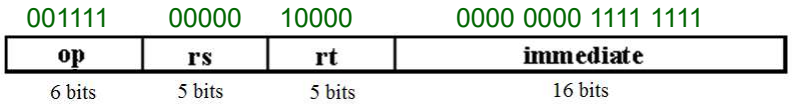
Load upper immediate: LUI **I-Format**

lui reg1,immediate ≡ reg1 ← immediate << 16;
Opcode: 001111

Example 1:
lui $s0,255 ≡ $s0 ← 0x00FF0000;

rs: 00000
rt: $s0 → 10000
Immediate: 0x00FF

Machine Code: 0x3c1000FF



Data Transfer Instructions

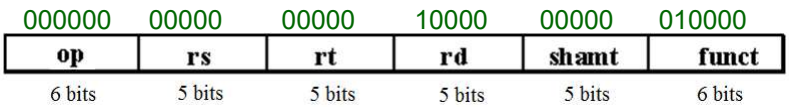
Move from HI/LO: mfhi,mflo (Useful for MUL and DIV) R-Format

mfhi reg1 ≡ reg1 ← (HI); op: 000000 funct: 010000
mflo reg1 ≡ reg1 ← (LO); op: 000000 funct: 010010

Example 1:
mfhi $s0 ≡ $s0 ← (HI);

rs, rt: 00000 rd: $s0: 10000 shamt: 00000 funct: 010000

Machine Code: 0x00008010



Arithmetic and logical Instructions

Without immediate: add, sub, slt, mult, div R-Format

add reg1,reg2,reg3 ≡ reg1 ← (reg2)+(reg3);
op: 000000 funct: 100000

sub reg1,reg2,reg3 ≡ reg1 ← (reg2) – (reg3);
op: 000000 funct: 100010

slt reg1,reg2,reg3 ≡ reg1 ← ((reg2)<(reg3))?1:0;
op: 000000 funct: 101010

mult reg1,reg2 ≡ HI:LO ← (reg1)*(reg2);
op: 000000 funct: 011000

div reg1,reg2 ≡ HI ← (reg1)%(reg2), LO ← (reg1)/(reg2)
op: 000000 funct: 011010

Arithmetic and logical Instructions

Without immediate: and, or, nor, xor R-Format

and reg1,reg2,reg3 ≡ reg1 ← (reg2) & (reg3)
op: 000000 funct: 100100

or reg1,reg2,reg3 ≡ reg1 ← (reg2) | (reg3)
op: 000000 funct: 100101

nor reg1,reg2,reg3 ≡ reg1 ← ~((reg2) | (reg3))
op: 000000 funct: 100111

xor reg1,reg2,reg3 ≡ reg1 ← (reg2) ^ (reg3)
op: 000000 funct: 100110

Arithmetic and logical Instructions

Without immediate: sll, sllv, sra, srav, srl, srlv R-Format

sll reg1,reg2,shamt ≡ reg1 ← (reg2) << shamt (logical)
op: 000000 funct: 000000

sllv reg1,reg2,reg3 ≡ reg1 ← (reg2) << (reg3) (logical)
op: 000000 funct: 000100

sra reg1,reg2,shamt ≡ reg1 ← (reg2) >> shamt (arithmetic)
op: 000000 funct: 000011

srav reg1,reg2,reg3 ≡ reg1 ← (reg2) >> (reg3) (arithmetic)
op: 000000 funct: 000111

srl reg1,reg2,shamt ≡ reg1 ← (reg2) >> shamt (logical)
op: 000000 funct: 000010

srlv reg1,reg2,reg3 ≡ reg1 ← (reg2) >> (reg3) (logical)
op: 000000 funct: 000110

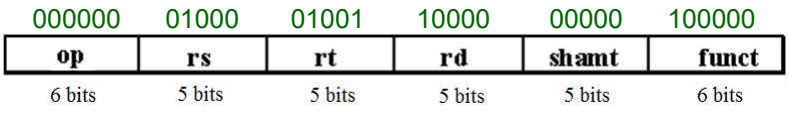
Arithmetic and logical Instructions

Example 1:

add $s0,$t0,$t1 ≡ $s0 ← ($t0)+($t1)

rs: $t0 → 01000
rt: $t1 → 01001
rd: $s0 → 10000
shamt: 00000
Funct: 100000

Machine Code: 0x01098020



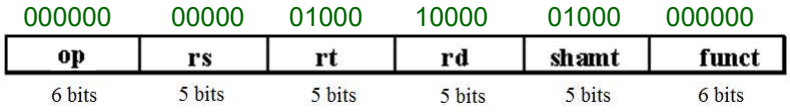
Arithmetic and logical Instructions

Example 2:

sll $s0,$t0,8 ≡ $s0 ← ($t0) << 8

rs: 00000
rt: $t0 → 01000
rd: $s0 → 10000
shamt: 01000
Funct: 000000

Machine Code: 0x00088200



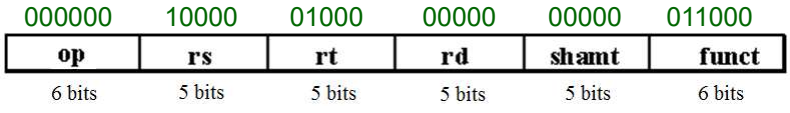
Arithmetic and logical Instructions

Example 3:

mult $s0,$t0 ≡ HI,LO ← ($s0) * ($t0)

rs: $s0 → 10000
rt: $t0 → 01000
rd: 00000
shamt: 00000
funct: 011000

Machine Code: 0x02080018



Arithmetic and logical Instructions

With immediate: addi, slti, andi, ori, xori **I-Format**

addi	reg1,reg2,imm	≡	reg1 ← (reg2)+imm;	op: 001000
slti	reg1,reg2,imm	≡	reg1 ← ((reg2)<imm)?1:0;	op: 001010
andi	reg1,reg2,imm	≡	reg1 ← (reg2) & imm	op: 001100
ori	reg1,reg2,imm	≡	reg1 ← (reg2) | imm	op: 001101
xori	reg1,reg2,imm	≡	reg1 ← (reg2) ^ imm	op: 001110

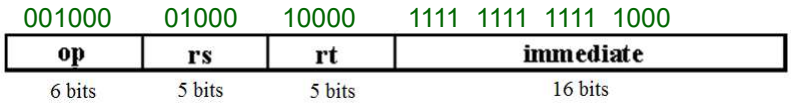
Arithmetic and logical Instructions

Example 1:

addi $s0,$t0,-8 ≡ $s0 ← ($t0)-8

op: 001000
rs: $t0 → 01000
rt: $s0 → 10000
immediate: 0xFFF8

Machine Code: 0x2110FFF8



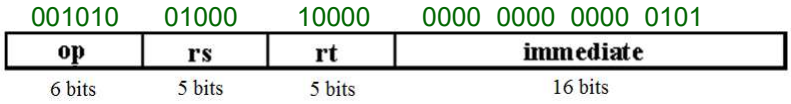
Arithmetic and logical Instructions

Example 2:

slti $s0,$t0,5 ≡ $s0 ← (($t0)<5)?1:0

op: 001010
rs: $t0 → 01000
rt: $s0 → 10000
immediate: 0x0005

Machine Code: 0x29100005



Conditional Branch Instructions

beq, bgez, bgezal, bgtz **I-Format**

beq reg1,reg2,offset ≡ pc ←((reg1)==(reg2))?(pc)+4+(offset*4):(pc)+4;
op: 000100

bgez reg1,offset ≡ pc ←((reg1)>=0)?(pc)+4+(offset*4):(pc)+4;
op: 000001, rt: 00001

bgezal reg1,offset ≡ $ra ← (pc)+4;
pc ←((reg1)>=0)?(pc)+4+(offset*4):(pc)+4;
op: 000001, rt: 10001

bgtz reg1,offset ≡ pc ←((reg1)>0)?(pc)+4+(offset*4):(pc)+4;
op: 000111, rt: 00000

Conditional Branch Instructions

blez, bltz, bltzal, bne **I-Format**

blez reg1,offset ≡ pc ←((reg1)<=0)?(pc)+4+(offset*4):(pc)+4;
op: 000110, rt: 00000

bltz reg1,offset ≡ pc ←((reg1)<0)?(pc)+4+(offset*4):(pc)+4;
op: 000001, rt: 00000

bltzal reg1,offset ≡ $ra ← (pc)+4;
pc ←((reg1)<0)?(pc)+4+(offset*4):(pc)+4;
op: 000001, rt: 10000

bne reg1,reg2,offset ≡ pc ←((reg1)!=(reg2))?(pc)+4+(offset*4):(pc)+4;
op: 000101

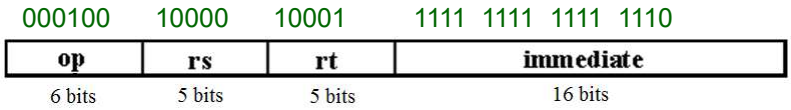
Conditional Branch Instructions

Example 1:

```
L1: Inst1
    beq    $s0,$s1,L1
```

op: 000100
rs: $s0 → 10000
rt: $s1 → 10001
offset: -2

Machine Code: 0x1211FFFE



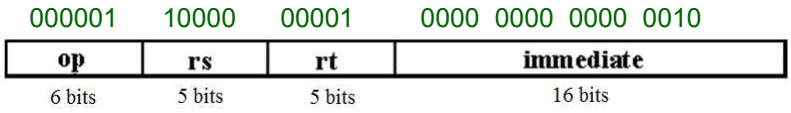
Conditional Branch Instructions

Example 2:

```
        bgez  $s0,L1
        Inst1
        Inst2
   L1:  Inst3
```

op: 000001
rs: $s0 → 10000
rt: 00001
offset: 2

Machine Code: 0x06010002



Unconditional Branch Instructions

j, jal, **J-Format**

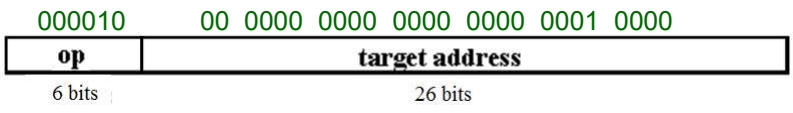
j target ≡ pc ← {pc[31:28],target*4}; op: 000010

jal target ≡ $ra ← (pc)+4;
pc ← {pc[31:28],target*4}; op: 000011

Example 1: j 0x10;

op: 000010
target: 0x10

Machine Code: 0x08000010



Unconditional Branch Instructions

jr, jalr, syscall **R-Format**

jr reg1 ≡ pc ← (reg1);
op: 000000 funct: 001000

jalr reg1, reg2 ≡ reg1 ←(pc)+4;
pc ← (reg2);
op: 000000 funct: 001001

syscall ≡ epc ← (pc)+4;
pc ← 0x3c;
Using $v0 to select services
Using $a0-$a3 for arguments
op: 000000 funct: 001100

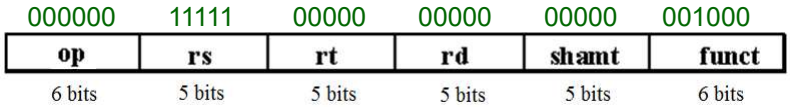
Unconditional Branch Instructions

jr, jalr, syscall **R-Format**

Example 1:
jr $ra;

op: 000000
rs: 11111
rt: 00000
rd: 00000
shamt: 00000
funct: 001000

Machine Code: 0x03E00008



Working with Syscall (Input/Ouput)

Example 1: Printing an integer number in $t0

```
addi    $v0, $zero, 1  #selecting the corresponding service
add     $a0, $zero, $t0  #moving the integer number to $a0
syscall
```

Example 2: Reading an integer number and moving it to $t0

```
addi    $v0, $zero, 5  #selecting the corresponding service
syscall
add     $t0, $zero, $v0  #moving the integer number to $t0
```

Example 3: Printing a string; $a0 is the address of string

```
addi    $v0, $zero, 4  #selecting the corresponding service
syscall
```


Working with Stack Memory

No push/pop instruction in MIPS

Example 1: Push two words ($t0 & $t1) on stack

```
addi    $sp, $sp, -8  #making space on stack
sw      $t1, 4($sp)
sw      $t0, 0($sp)
```

Example 2: Pop two words

```
lw      $t0, 0($sp)
lw      $t1, 4($sp)
addi    $sp, $sp, 8  #restoring stack pointer
```

How to Write a MIPS program:

```
        .data
        #add your data here

        .text
main:
        #add your code here


        addi    $v0, $zero, 10
        syscall #Terminating the program
```


Data Segment

.word → storing 32-bit numbers

.half → storing 16-bit numbers

.byte → storing 8-bit numbers

.space → reserving a number of bytes

.asciiz →. storing a string with null terminator

Example:

```
        .data
array_word:      .word    1, 10
array_halfword:  .half    16, 20, -10, 65
array_byte:      .byte    -3, 10
free_byte:       .space   100
string:          .asciiz  "Enter a number: "
```

Calculate MIN and MAX in a Array:

```
        .text
main:
        la      $t0, count          #A pseudo instruction:  lui $t0, count[31:16]
                                                            ori $t0, $t0, count[15:0]
        la      $s0, array
        lw      $t0, 0($t0)
        li      $t5, 1              #A pseudo instruction:  lui $t5, 0
                                                            ori $t5, $t5, 1
        add     $t1, $zero, $s0
        lw      $s1, 0($t1)
        lw      $s2, 0($t1)
loop:
        beq     $t5, $t0, exit
        addi    $t1, $t1, 4
        lw      $t2, 0($t1)
        slt     $t3, $s1, $t2
        slt     $t4, $t2, $s2
        beq     $t3, $zero, change
```


Calculate MIN and MAX in a Array:

```
continue:
        beq     $t4, $zero, change2
continue2:
        addi    $t5, $t5, 1
        j       loop
change:
        add     $s1, $zero, $t2
        j       continue
change2:
        add     $s2, $zero, $t2
        j       continue2
exit:
        li      $v0,10
        syscall

        .data
count:
        .word 15
array:
        .word 3,4,2,6,12,7,18,26,2,14,19,7,8,12,13
```

Calculate Fibonacci Sequence:

```
        .text
main:
        la      $s0, n
        add     $t1, $zero, $s0
        lw      $s1, 0($s0)
        addi    $t1, $t1, 4
        li      $t0, 1
        li      $t2, 1
        sw      $t0, 0($t1)
        addi    $t1, $t1, 4
        sw      $t0, 0($t1)
        li      $t4, 1
loop:   addi    $t4, $t4, 1
        beq     $t4, $s1, exit
        add     $t3, $zero, $t0
        add     $t0, $zero, $t2
        add     $t2, $t3, $t2
        addi    $t1, $t1, 4
        sw      $t2, 0($t1)
        j       loop
exit:
        li      $v0,10
        syscall

        .data
n:
        .word 15
```


Reverse every word in a string:

```
        .text
main:
        la      $s0, str
        li      $t4, 1
        li      $t1, 32
        add     $t6, $zero, $s0
l0:
        lb      $t7, 0($t6)
        beq     $t7, $zero, exit1
        addi    $t6, $t6, 1
        j       l0
exit1:
        sb      $t1, 0($t6)
        sb      $zero, 1($t6)
```

Reverse every word in a string:

```
l1:
        lb      $s1, 0($s0)
        beq     $s1, $t1, l2
        j       l3
l2:
        addi    $s0, $s0, 1
        j       l1
l3:
        add     $t2, $zero, $s0
l4:
        lb      $s1, 0($s0)
        beq     $s1, $zero, exit
        beq     $s1, $t1, l5
        addi    $s0, $s0, 1
        j       l4
l5:
        add     $t5, $zero, $s0
        addi    $t5, $t5, -1
```


Reverse every word in a string:

```
l6:     sub     $t3, $t5, $t2
        slti    $t3, $t3, 2
        beq     $t3, $t4, l7
        lb      $s2, 0($t2)
        lb      $s3, 0($t5)
        sb      $s2, 0($t5)
        sb      $s3, 0($t2)
        addi    $t2, $t2, 1
        addi    $t5, $t5, -1
        j       l6
l7:
        lb      $s2, 0($t2)
        lb      $s3, 0($t5)
        sb      $s2, 0($t5)
        sb      $s3, 0($t2)
l8:
        lb      $s1, 0($s0)
        beq     $s1, $t1, l9
        j       l3
l9:     addi    $s0, $s0, 1
        j       l8
```

```
exit:
        sb      $zero, -1($s0)
        li      $v0, 4
        la      $a0, str
        syscall

        li      $v0,10
        syscall

        .data
str:
    .asciiz "Computer structure and language"
```

Questions?