

---

# The MIPS architecture

M. Sonza Reorda

Politecnico di Torino  
Dipartimento di Automatica e Informatica

1

M. Sonza Reorda – Politecnico di Torino

## MIPS

- MIPS is a family of RISC processors, which have been very successful for embedded applications
- The first processor in the family was introduced in 1985
- Several versions have been introduced since then
- What is described here is a simplified version of the so called MIPS64.

2

M. Sonza Reorda – Politecnico di Torino

---

## General characteristics

- Simple load-store instruction set
- Design for pipelining efficiency
  - Fixed instruction set encoding
- Efficiency as a compiler target.

3

M. Sonza Reorda – Politecnico di Torino

## Registers

- MIPS includes
  - 32 64-bit *general purpose registers* (GPRs) named R0, R1, ..., R31 known as integer registers
  - 32 64-bit *floating point registers* (FPRs) named F0, F1, ..., F31; they can store either a single-precision (32-bit) or a double-precision (64-bit) floating-point value
- The value of R0 is always 0.

4

M. Sonza Reorda – Politecnico di Torino

---

# Data types

- Supported data types
  - For integer data
    - 8-bit bytes
    - 16-bit half-words
    - 32-bit words
    - 64-bit double words
  - For floating-point data
    - 32-bit single precision
    - 64-bit double precision
- MIPS64 operations work on
  - 64-bit integers
  - 32- or 64-bit floating point values.

5

M. Sonza Reorda – Politecnico di Torino

# Addressing modes

Only two addressing modes are supported

- Immediate
- Displacement (with 16-bit displacement field)
  - May evolve into
    - register indirect (if displacement is null)
    - absolute addressing (if register is R0).

Memory is byte addressable with 64-bit addresses.

Either the Big endian or the Little endian representation can be selected via software.

6

M. Sonza Reorda – Politecnico di Torino

---

# Instruction set

- It is composed of 4 groups
  - Load and store instructions
  - ALU instructions
  - Branches and jumps
  - Floating-point instructions.

7

M. Sonza Reorda – Politecnico di Torino

## Load and store

---

<i>Data transfers</i>	<i>Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR</i>
LB, LBU, SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH, LHU, SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW, LWU, SW	Load word, load word unsigned, store word (to/from integer registers)
LD, SD	Load double word, store double word (to/from integer registers)
L.S, L.D, S.S, S.D	Load SP float, load DP float, store SP float, store DP float
MFC0, MTC0	Copy from/to GPR to/from a special register
MOV.S, MOV.D	Copy one SP or DP FP register to another FP register
MFC1, MTC1	Copy 32 bits to/from FP registers from/to integer registers

---

8

M. Sonza Reorda – Politecnico di Torino

## Load and store: examples

Example instruction	Instruction name	Meaning
LD R1,30(R2)	Load double word	$\text{Regs}[R1] \leftarrow_{64} \text{Mem}[30+\text{Regs}[R2]]$
LD R1,1000(R0)	Load double word	$\text{Regs}[R1] \leftarrow_{64} \text{Mem}[1000+0]$
LW R1,60(R2)	Load word	$\text{Regs}[R1] \leftarrow_{64} (\text{Mem}[60+\text{Regs}[R2]])_0^{32} \text{ ## Mem}[60+\text{Regs}[R2]]$
LB R1,40(R3)	Load byte	$\text{Regs}[R1] \leftarrow_{64} (\text{Mem}[40+\text{Regs}[R3]])_0^{56} \text{ ## Mem}[40+\text{Regs}[R3]]$
LBU R1,40(R3)	Load byte unsigned	$\text{Regs}[R1] \leftarrow_{64} 0^{56} \text{ ## Mem}[40+\text{Regs}[R3]]$
LH R1,40(R3)	Load half word	$\text{Regs}[R1] \leftarrow_{64} (\text{Mem}[40+\text{Regs}[R3]])_0^{48} \text{ ## Mem}[40+\text{Regs}[R3]] \text{ ## Mem}[41+\text{Regs}[R3]]$
L.S F0,50(R3)	Load FP single	$\text{Regs}[F0] \leftarrow_{64} \text{Mem}[50+\text{Regs}[R3]] \text{ ## } 0^{32}$
L.D F0,50(R2)	Load FP double	$\text{Regs}[F0] \leftarrow_{64} \text{Mem}[50+\text{Regs}[R2]]$
SD R3,500(R4)	Store double word	$\text{Mem}[500+\text{Regs}[R4]] \leftarrow_{64} \text{Regs}[R3]$
SW R3,500(R4)	Store word	$\text{Mem}[500+\text{Regs}[R4]] \leftarrow_{32} \text{Regs}[R3]_{32..63}$
S.S F0,40(R3)	Store FP single	$\text{Mem}[40+\text{Regs}[R3]] \leftarrow_{32} \text{Regs}[F0]_{0..31}$
S.D F0,40(R3)	Store FP double	$\text{Mem}[40+\text{Regs}[R3]] \leftarrow_{64} \text{Regs}[F0]$
SH R3,502(R2)	Store half	$\text{Mem}[502+\text{Regs}[R2]] \leftarrow_{16} \text{Regs}[R3]_{48..63}$
SB R2,41(R3)	Store byte	$\text{Mem}[41+\text{Regs}[R3]] \leftarrow_8 \text{Regs}[R2]_{56..63}$

§

## ALU instructions

<i>Arithmetic/logical</i>	<i>Operations on integer or logical data in GPRs; signed arithmetic trap on overflow</i>
DADD, DADDI, DADDU, DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB, DSUBU	Subtract; signed and unsigned
DMUL, DMULU, DDIV, DDIVU, MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND, ANDI	And, and immediate
OR, ORI, XOR, XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV	Shifts: both immediate (DS_) and variable form (DS_V); shifts are shift left logical, right logical, right arithmetic
SLT, SLTI, SLTU, SLTIU	Set less than, set less than immediate; signed and unsigned

---

## ALU instructions: examples

Example instruction	Instruction name	Meaning
DADDU R1,R2,R3	Add unsigned	$\text{Regs}[R1] \leftarrow \text{Regs}[R2] + \text{Regs}[R3]$
DADDIU R1,R2,#3	Add immediate unsigned	$\text{Regs}[R1] \leftarrow \text{Regs}[R2] + 3$
LUI R1,#42	Load upper immediate	$\text{Regs}[R1] \leftarrow 0^{32} \# \# 42 \# 0^{16}$
DSLL R1,R2,#5	Shift left logical	$\text{Regs}[R1] \leftarrow \text{Regs}[R2] \ll 5$
SLT R1,R2,R3	Set less than	$\text{if } (\text{Regs}[R2] < \text{Regs}[R3])$ $\text{Regs}[R1] \leftarrow 1 \text{ else } \text{Regs}[R1] \leftarrow 0$

11

M. Sonza Reorda – Politecnico di Torino

## ALU instructions: R0 usage

- ADD immediate with R0 as source operand → loading a constant
- ADD with R0 as source operand → moving from register to register.

12

M. Sonza Reorda – Politecnico di Torino

# Control flow instructions

<i>Control</i>	<i>Conditional branches and jumps; PC-relative or through register</i>
BEQZ, BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ, BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T, BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN, MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J, JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL, JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode

13

M. Sonza Reorda – Politecnico di Torino

## Control flow instructions: examples

Example instruction	Instruction name	Meaning
J name	Jump	$PC_{36..63} \leftarrow \text{name}$
JAL name	Jump and link	$\text{Regs}[R31] \leftarrow PC+8$ ; $PC_{36..63} \leftarrow \text{name}$ ; $((PC+4)-2^{17}) \leq \text{name} < ((PC+4)+2^{17})$
JALR R2	Jump and link register	$\text{Regs}[R31] \leftarrow PC+8$ ; $PC \leftarrow \text{Regs}[R2]$
JR R3	Jump register	$PC \leftarrow \text{Regs}[R3]$
BEQZ R4, name	Branch equal zero	if $(\text{Regs}[R4] == 0)$ $PC \leftarrow \text{name}$ ; $((PC+4)-2^{17}) \leq \text{name} < ((PC+4)+2^{17})$
BNE R3, R4, name	Branch not equal zero	if $(\text{Regs}[R3] \neq \text{Regs}[R4])$ $PC \leftarrow \text{name}$ ; $((PC+4)-2^{17}) \leq \text{name} < ((PC+4)+2^{17})$
MOVZ R1, R2, R3	Conditional move if zero	if $(\text{Regs}[R3] == 0)$ $\text{Regs}[R1] \leftarrow \text{Regs}[R2]$

14

M. Sonza Reorda – Politecnico di Torino

---

# Floating-point instructions

---

<i>Floating point</i>	<i>FP operations on DP and SP formats</i>
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D,SUB.S,SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D,MADD.S,MADD.PS	Multiply-add DP, SP numbers and pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT. __. __	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C. __.D,C. __.S	DP and SP compares: “__” = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

---

15

M. Sonza Reorda – Politecnico di Torino

## Instruction format

- All instructions
  - are encoded on 32 bits
  - Include a 6-bit primary opcode
- Format changes depending on the instruction type:
  - I-type: load, store, conditional jumps
  - R-type: ALU instructions
  - J-type: jump, jump and link, trap, return from exceptions.

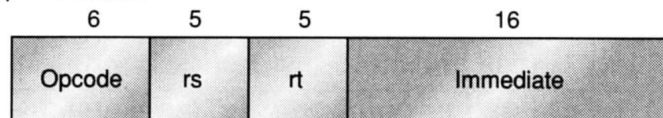
16

M. Sonza Reorda – Politecnico di Torino

---

## I-type instruction format

I-type instruction



Encodes: Loads and stores of bytes, half words, words, double words. All immediates ( $rt \leftarrow rs \text{ op immediate}$ )

Conditional branch instructions (rs is register, rd unused)

Jump register, jump and link register

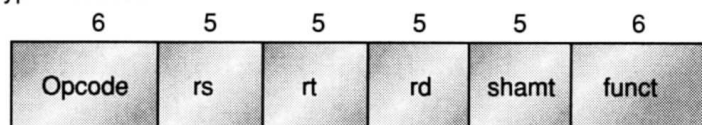
(rd = 0, rs = destination, immediate = 0)

17

M. Sonza Reorda – Politecnico di Torino

## R-type instruction format

R-type instruction



Register-register ALU operations:  $rd \leftarrow rs \text{ funct } rt$

Function encodes the data path operation: Add, Sub, . . .

Read/write special registers and moves

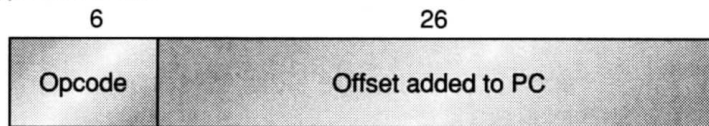
18

M. Sonza Reorda – Politecnico di Torino

---

# J-type instruction format

J-type instruction



Jump and jump and link  
Trap and return from exception