MIPS Architecture

MIPS ISA Design Principles

1. **Simplicity favors regularity**
    - Consistent instruction size, instruction formats, data formats
    - Eases implementation by simplifying hardware, leading to higher performance
2. **Smaller is faster**
    - Fewer bits to access and modify
    - Use the register file instead of slower memory
3. **Make the common case fast**
    - e.g. Small constants are common, thus small immediate fields should be used.
4. **Good design demands good compromises**
    - Compromise with special formats for important exceptions
    - e.g. A long jump (beyond a small constant)

MIPS - Introduction

word - 16 bits
- 2 byks

- Microprocessor without Interlocked Pipelined Stages (MIPS)
- family of reduced instruction set computer (RISC)
- Developed by MIPS Technologies
- Initially 32 bit instruction set but now, MIPS 32/64 bit
- 5 versions - MIPS I, II, III, IV, and V
- Byte addressing - Byte addressing refers to hardware architectures which support accessing individual bytes.
- MIPS follows to **Big Endian** for storing

word addressing, byte addressing

What is Microprocessor without Interlocked Pipelined Stages ?

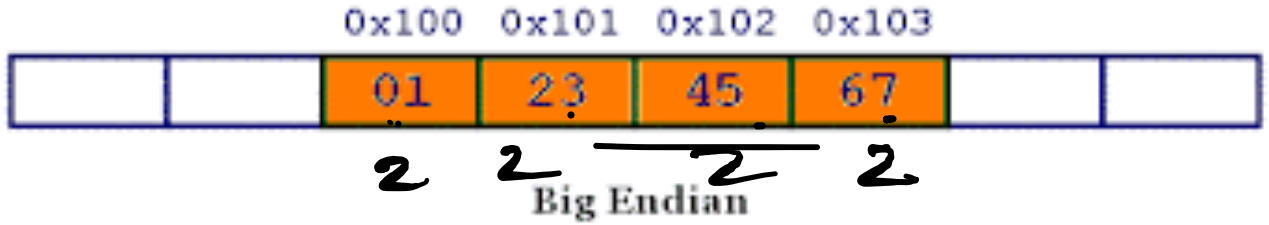
- A project at Stanford University intended to simplify processor design by eliminating hardware interlocks between the five pipeline stages.
- This means that only single execution cycle instructions can access the thirty two 32bit general registers, so that the compiler can schedule the m to avoid conflicts.

- **Big endian and Little Endian**

Little and big endian are two ways of storing multibyte data-types (int, float, etc).

In little endian machines, last byte of binary representation of the multibyte data-type is stored first.

On the other hand, in big endian machines, first byte of binary representation of the multibyte data-type is stored first.



Big Endian

Little Endian

Registers in MIPS

- There are only 32 (programmer visible) 32-bit registers in a MIPS processor.
  - Intel x86 has 8 registers, x86-64 has 16 registers
- HI/LO registers separately other than GPR to store results of multiplication and division.
- Registers also use much less power than memory accesses.
- MIPS convention is to use two-character names following a dollar sign.
  - Register 0: $zero – stores the constant value 0.
  - Registers 16-23: $s0-$s7 – saved temporaries (variables in C code).
  - Registers 8-15: $t0-$t7 – temporaries.

Registers

Name	Number	Use
$zero	0	Constant value 0.
$at	1	Assembler temporary. For resolving pseudo-instructions.
$v0-$v1	2-3	Function results and expression evaluation.
$a0-$a3	4-7	Arguments.
$t0-$t9	8-15, 24-25	Temporaries.
$s0-$s7	16-23	Saved temporaries.
$k0-$k1	26-27	Reserved for OS kernel.
$gp	28	Global pointer.
$sp	29	Stack pointer.
$fp	30	Frame pointer.
$ra	31	Return address.

MIPS Instructions

- Arithmetic
  - add, subtract, multiply, divide
- Logical
  - and, or, nor, not, shift
- Data transfer (load and store)
  - load from or store to memory
- Control transfer (branches)
  - jumps, branches, calls, returns

MIPS Instruction format

There are three different formats for MIPS instructions.

- R format
  - Used for shifts and instructions that reference only registers.

- I format
  - Used for loads, stores, branches, and immediate instructions.

- J format
  - Used for jump and call instructions

Representing instructions in MIPS

Name	Fields					
Field Size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R format	op	rs	rt	rd	shamt	funct
I format	op	rs	rt	immed		
J format	op	targaddr				

op – instruction opcode.
rs – first register source operand.
rt – second register source operand.
rd – register destination operand.

shamt – shift amount.
funct – additional opcodes.
immed – offsets/constants.
targaddr – jump/call target.

opcode dst soc soc

16001

MIPS R Format

- Used for shift operations and instructions that only reference registers.
- The op field has a value of 0 for all R format instructions.
- The funct field indicates the type of R format instruction to be performed.
- The shamt field is used only for the shift instructions (sll and srl, sra)

Name	Fields					
Field Size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R format	op	rs	rt	rd	shamt	funct

op – instruction opcode.
rs – first register source operand.
rt – second register source operand.
rd – register destination operand.
shamt – shift amount.
funct – additional opcodes.

R Format Instruction Encoding Example

Consider the following R format instruction:

```
addu $t2, $t3, $t4
```

— register reference inst.

Fields	op	rs	rt	rd	shamt	funct
Size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Decimal	0	11	12	10	0	33
Binary	000000	01011	01100	01010	00000	100001
Hexadecimal			0x016c5021			

MIPS I Format

- Used for arithmetic/logical immediate instructions, loads, stores, and conditional branches.
- The op field is used to identify the type of instruction.
- The rs field is the source register.
- The rt field is either the source or destination register, depending on the instruction.
- The immed field is zero-extended if it is a logical operation. Otherwise, it is sign extended.

Name	Fields					
Field Size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I format	op	rs	rt	immed		

I Format Instruction Encoding Example

Arithmetic example:

```
addiu $t0,$t0,1
```

Fields	op	rs	rt	immed
Size	6 bits	5 bits	5 bits	16 bits
Decimal	9	8	8	1
Binary	001001	01000	01000	0000000000000001
Hexadecimal	0x25080001			

I Format Instruction Encoding Example

Memory access example:

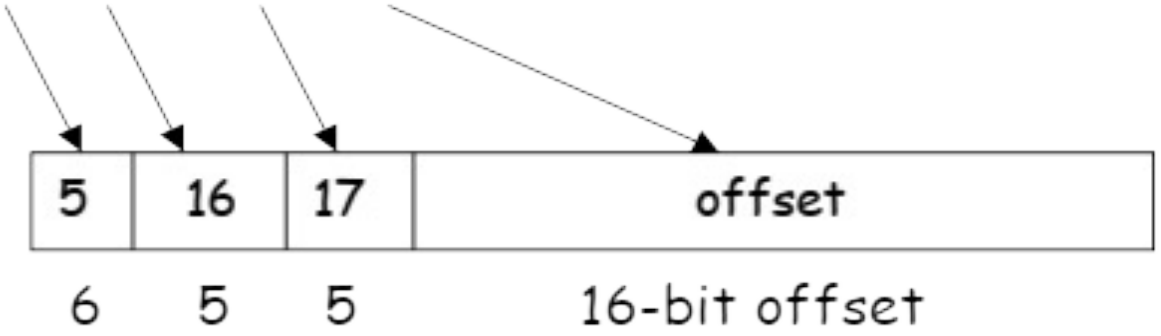
```
lw $s1,100($s2)
```

Fields	op	rs	rt	immed
Size	6 bits	5 bits	5 bits	16 bits
Decimal	35	18	17	100
Binary	100011	10010	10001	0000000001100100
Hexadecimal	0x8e510064			

I Format Instruction Encoding Example

Conditional branch is represented using I-type format:

bne $s0, $s1, 1234 is represented as



PC + offset determines the branch target.

This is called **PC-relative addressing**.

MIPS J Format

- Used for unconditional jumps and function calls.
- The op field is used to identify the type of instruction.
- The targaddr field is used to indicate an absolute target address.

Name	**Fields**					
Field Size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
J format	op	targaddr				

Memory Operands in MIPS

- Memory contains both data and instructions.

  - von Neumann architecture v.s. Harvard architecture

- Memory can be viewed as a large array of bytes.

  - The beginning of a variable or instruction is associated with a specific element of this array.

  - The address of a variable or instruction is its offset from the beginning of memory.

J Format Instruction Encoding Example

Jump example: `j L1`

Assume L1 is at the address 4194340 in decimal, which is 400024 in hexadecimal. We fill the target field as an address in instructions (0x100009) rather than bytes (0x400024). Jump uses **pseudo-direct** addressing to create a 32-bit address.

Fields	**op**	**target address**
Size	6 bits	26 bits
Decimal	2	1048585
Binary	000010	00000100000000000000001001
Hexadecimal		0x08100009

MIPS Instructions

General format:

```
<optional label> <operation> <operands>
```

Example:

```
loop:    addu $t2,$t3,$t4 # instruction with a label
         subu $t2,$t3,$t4 # instruction without a label
L2:      # a label can appear on a line by itself
# a comment can appear on a line by itself
```