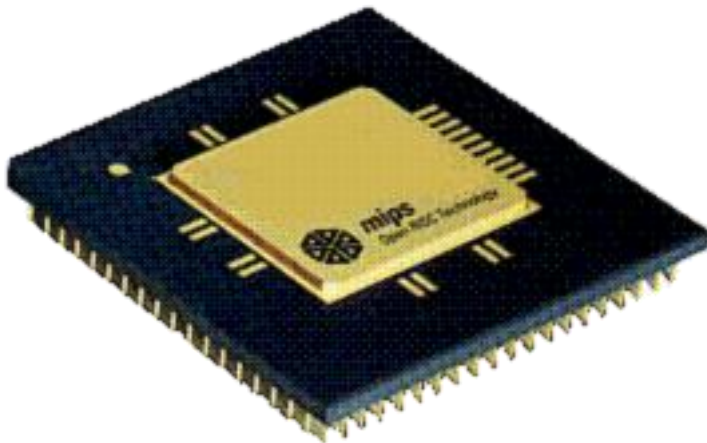


Machine Architecture - Lecture 4

Ioannis Ivrissimtzis

ioannis.ivrissimtzis@durham.ac.uk



MIPS - Assembly and Machine Language

MIPS

Microprocessor without Interlocked Pipeline Stages:

Designed in early 1980s at Stanford University; team led by John Hennessy.

Powered SGI workstations until mid-1990s.

Still used in embedded systems, consumer electronics.

Design principles

Underlying design principles, as articulated in Hennessy and Patterson in their book [Computer Architecture: A Quantitative Approach](#), (1989).

Simplicity favours regularity

Make the common case fast

Smaller is faster

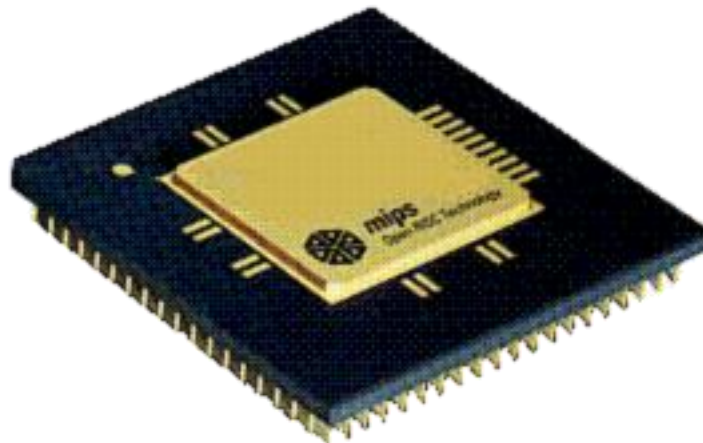
Good design demands good compromises

MIPS

32-bit RISC processor:

~110 instructions in the instruction set

32 general purpose registers



Registers

| Name | Number | Use |
|-------------|--------|-------------------------|
| \$0 | 0 | the constant value 0 |
| \$at | 1 | assembler temporary |
| \$v0 - \$v1 | 2-3 | function return values |
| \$a0 - \$a3 | 4-7 | function arguments |
| \$t0 - \$t7 | 8-15 | temporary variables |
| \$s0 - \$s7 | 16-23 | saved variables |
| \$t8 - \$t9 | 24-25 | temporary variables |
| \$k0 - \$k1 | 26-27 | OS temporaries |
| \$gp | 28 | global pointer |
| \$sp | 29 | stack pointer |
| \$fp | 30 | frame pointer |
| \$ra | 31 | function return address |

MIPS assembly

Not very different from the LMC assembly.

| High-level code | MIPS assembly code | Comments |
|-----------------|----------------------|--|
| a = b + c | add \$s0, \$s1, \$s2 | # \$s0 = a # \$s1 = b # \$s2 = c |

MIPS assembly

Not very different from the LMC assembly.

| High-level code | MIPS assembly code | Comments |
|-----------------|--|--|
| a = b + c - d | sub \$t0, \$s2, \$s3 add \$s0, \$s1, \$t0 | # \$s0 = a # \$s1 = b # \$s2 = c # \$s3 = d # t = c - d # a = b + t |

Instruction types

There are three types of instructions:

R-type: register operands

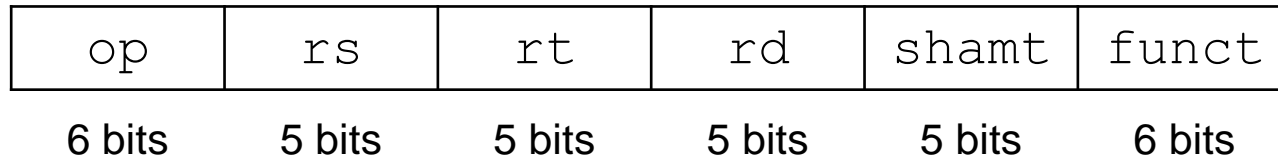
I-type: immediate operand

J-type: for jumping

add.s0, s1, s2
add.s0, \$3

Next, we will see how assembly code is translated to machine language.

R-type instruction format



Three register operands:

rs, rt: source registers
rd: destination register

10000
2 << 3 = 16

The other fields:

op: the operation code or **opcode** (always 0 for R-type instructions)
funct: the **function**, tells computer what operation to perform
shamt: the **shift amount** for shift instructions, otherwise it is 0

R-type instruction example 1

The assembly code for adding the values in registers 17 and 18 and putting the answer in register 16

```
add $s0, $s1, $s2
```

is translated to machine language (decimal field values are shown above the binary machine code and hexadecimals next to it)

| op | rs | rt | rd | shamt | funct | |
|--------|--------|--------|--------|--------|--------|------------|
| 0 | 17 | 18 | 16 | 0 | 32 | |
| 000000 | 10001 | 10010 | 10000 | 00000 | 100000 | 0x02328020 |
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits | |

Note: assembler and machine code order the operands differently.

R-type instruction example 2

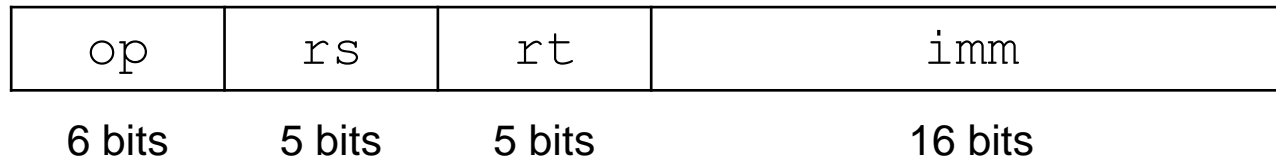
The assembly code for adding the values in registers 17 and 18 and putting the answer in register 16

```
sub $t0, $t3, $t5
```

is translated to machine language (decimal field values are shown above the binary machine code)

| op | rs | rt | rd | shamt | funct |
|--------|--------|--------|--------|--------|--------|
| 0 | 11 | 13 | 8 | 0 | 34 |
| 000000 | 01011 | 01101 | 01000 | 00000 | 100010 |
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

I-type instruction format



Three operands:

rs, rt: register operands

imm: 16-bit immediate

rs and imm are used as source operands. rt is used as source in some instructions and as destination in some other.

The other field:

op: the **opcode**. The operation is completely determined by the opcode.

I-type instruction example 1

The assembly code for adding the number 5 to the value of register 17 and 18 and putting the answer in register 16

```
addi $s0, $s1, 5
```

is translated to machine language (decimal field values are shown above the binary machine code)

| op | rs | rt | imm |
|--------|--------|--------|---------------------|
| 8 | 17 | 16 | 5 |
| 001000 | 10001 | 10000 | 0000 0000 0000 0101 |
| 6 bits | 5 bits | 5 bits | 16 bits |

I-type instruction example 2

The assembly code for loading the number 5 to the register 16

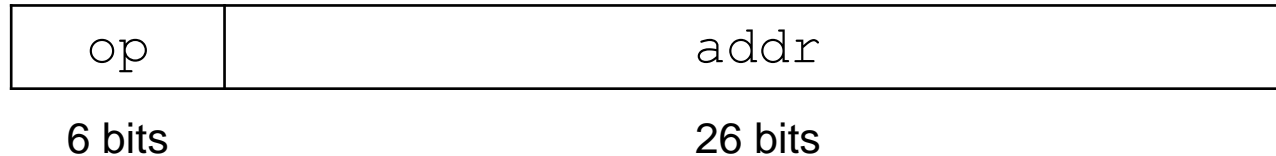
```
li $s0, 5
```

is translated to machine language (decimal field values are shown above the binary machine code)

| op | rs | rt | imm |
|--------|--------|--------|---------------------|
| 13 | 0 | 16 | 5 |
| 001101 | 00000 | 10000 | 0000 0000 0000 0101 |
| 6 bits | 5 bits | 5 bits | 16 bits |

Implemented by the **logical OR** I-type instruction: `ori $s0, $0, 5`

J-type instruction format



addr: 26-bit address operand

op: the **opcode**, the operation is completely determined by the opcode

Used for jump instructions such as **j**.

The compiler will create a 32-bit address from the 26 bits of `addr`.

Next ... addressing modes

Register Only

Immediate

Base Addressing

address of memory operand is given by
base address + signed immediate

PC-Relative (for writing the Program Counter)

jump so far from current position

Pseudo-direct (for writing the Program Counter)