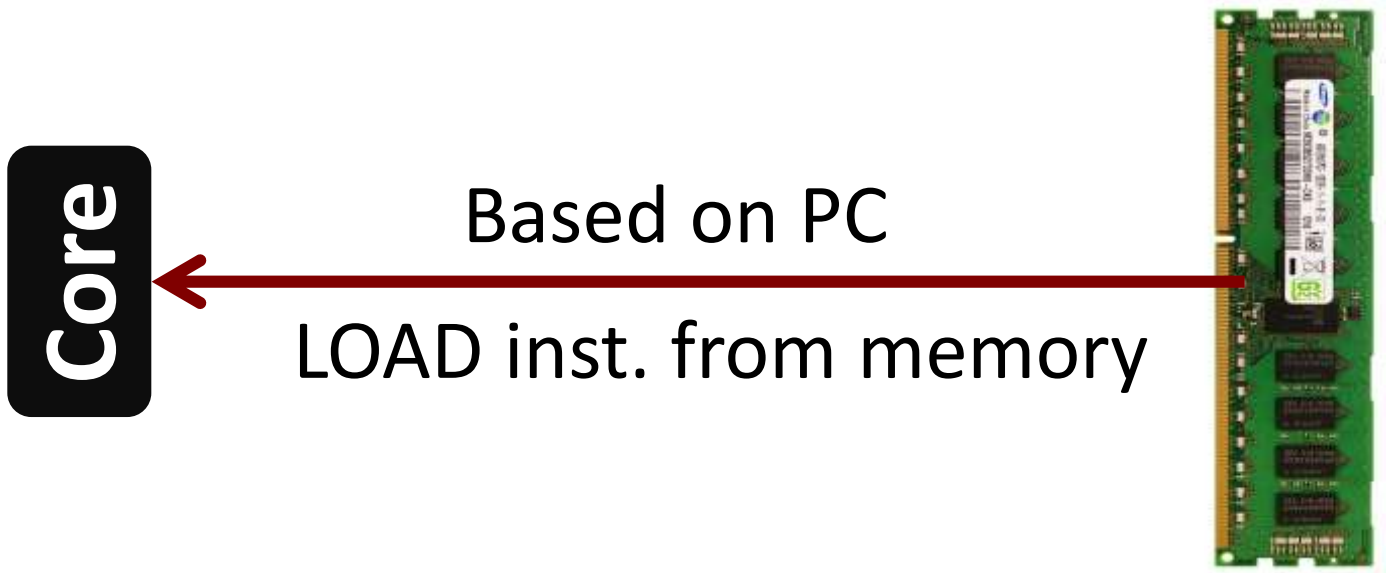
CS230: Digital Logic Design and Computer Architecture

Lecture 9: Instruction Decoding and addressing modes

https://www.cse.iitb.ac.in/~biswa/courses/CS230/autumn23/main.html

Why instruction decoding?

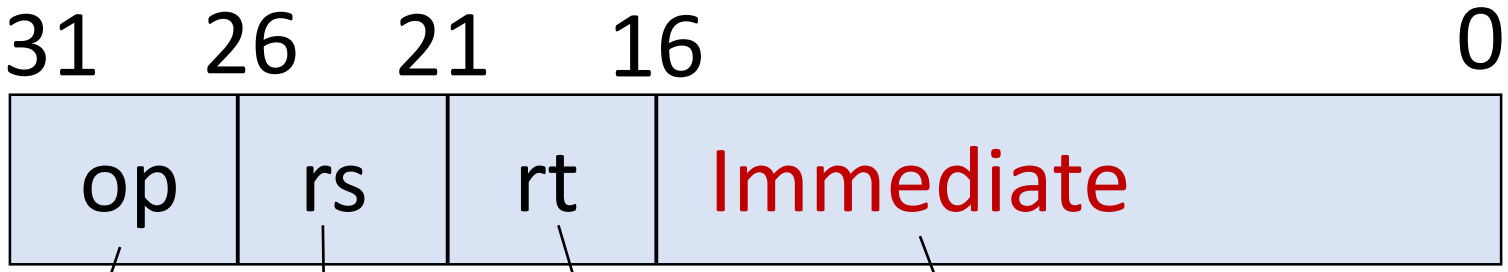


Instruction received then what?

Remember instructions are of 32-bit size (in MIPS), so PC+4

How will the processor know what to infer from these 32 bits?
Simple: Have a decoder ☺

Instruction Decoding



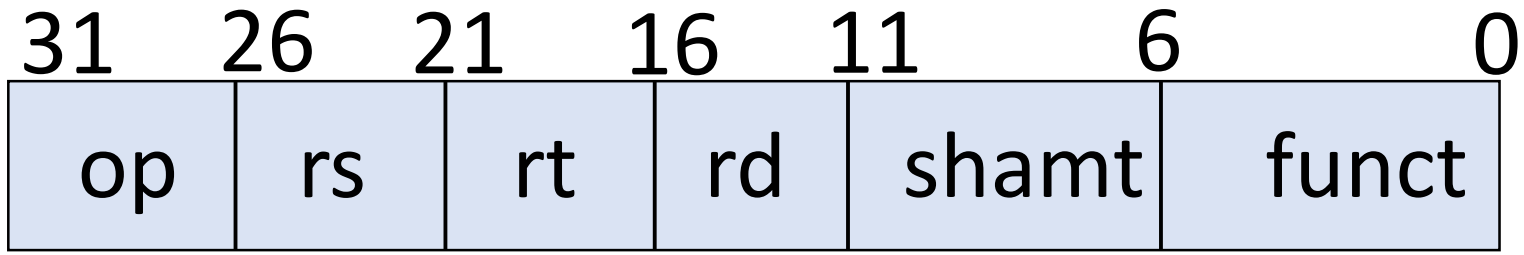
6 bits, maximum
64 operations
add
sub
mul
lw
bne …….

5 bits, 32 registers
(R0 to R31)

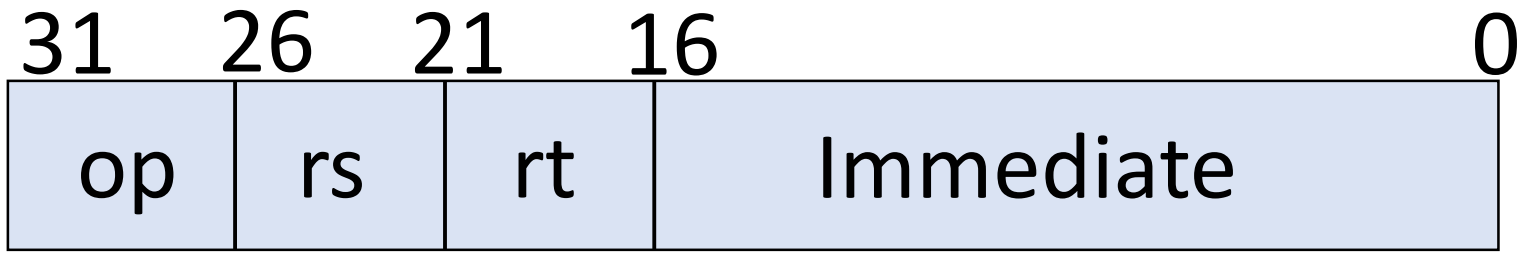
5 bits, 32 registers (R0 to R31)

16 bit constant

Let’s have a look



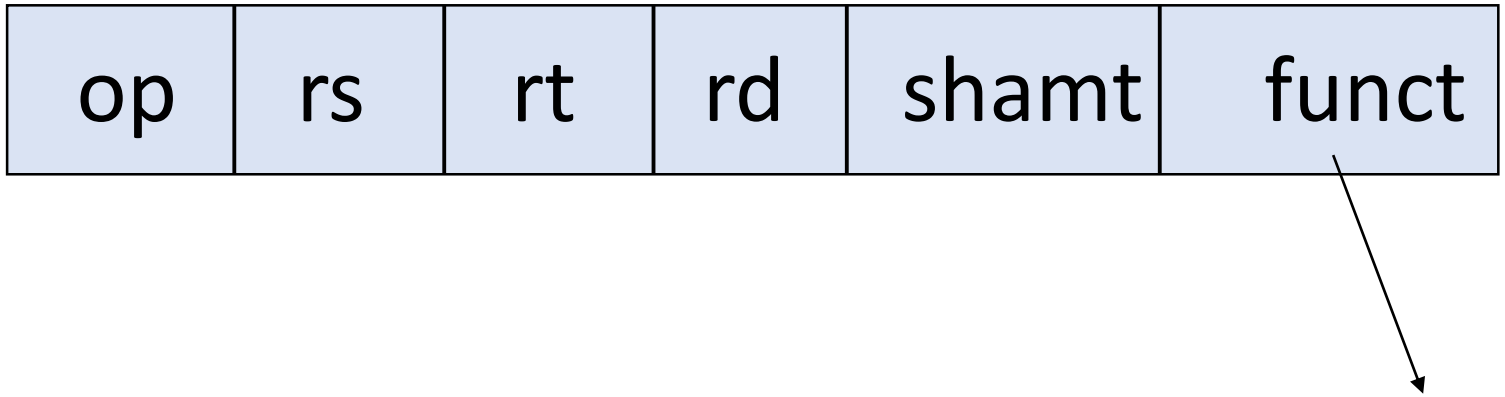
R-type



I-type

J-type

10K Feet View of MIPS encoding



Why this field?

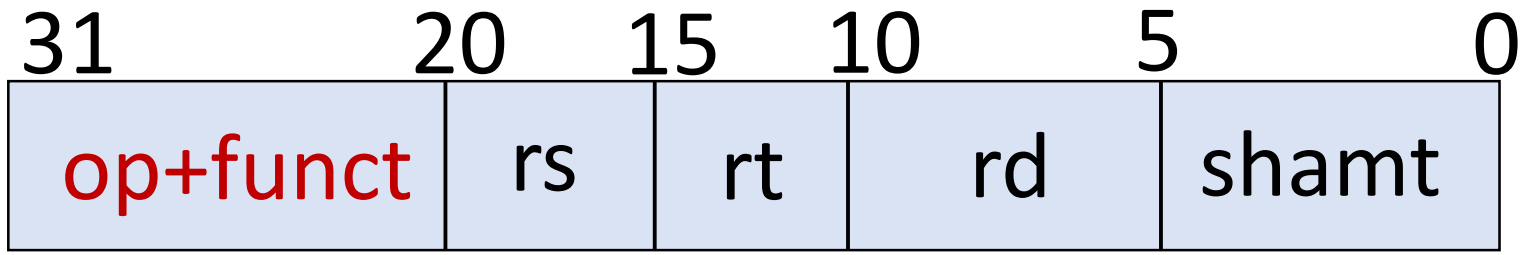
Wastage of space ☹

Good design demands good compromises

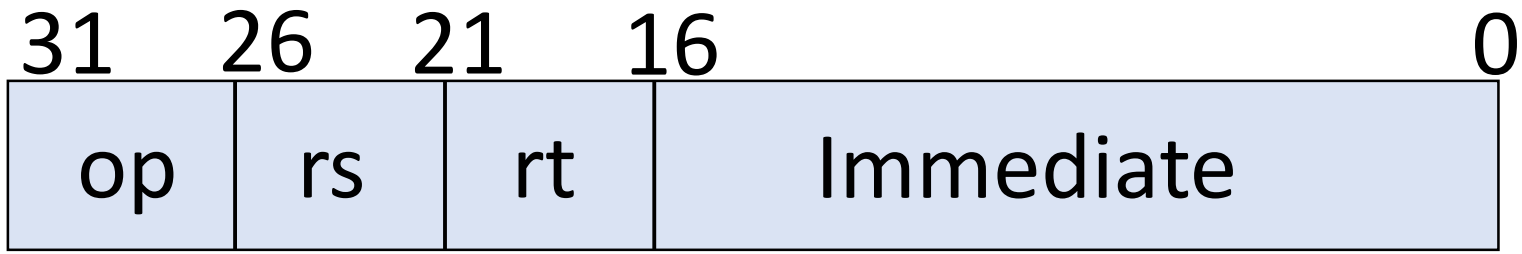
Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32	n.a.
sub	R	0	reg	reg	reg	0	34	n.a.
addi	I	8	reg	reg	n.a.	n.a.	n.a.	constant
lw	I	35	reg	reg	n.a.	n.a.	n.a.	address
sw	I	43	reg	reg	n.a.	n.a.	n.a.	address

tells how to treat the last set of fields:
three fields or one field, still why funct ☹

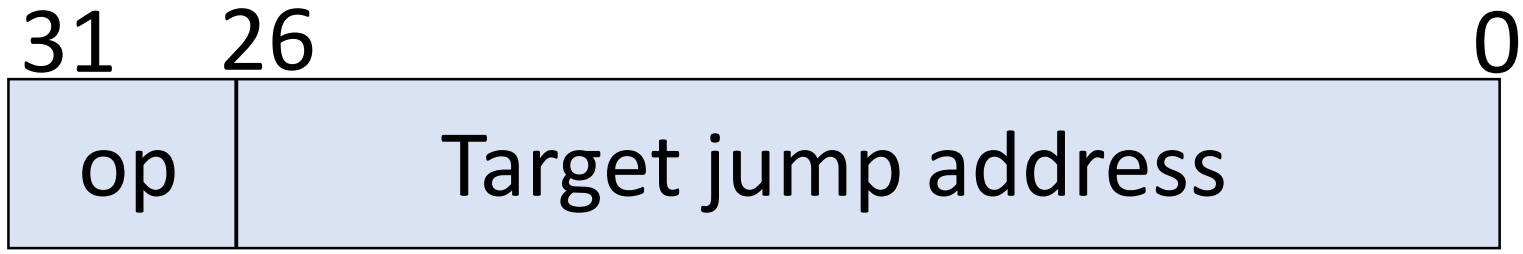
Why not?



R-type



I-type



J-type

What is a good compromise?

- Fixed length instructions ☺ 32-bit irrespective of ops
- Fields are at the *same* or almost same location
- All formats look *similar*

Addressing Modes

(How and where to find the data)

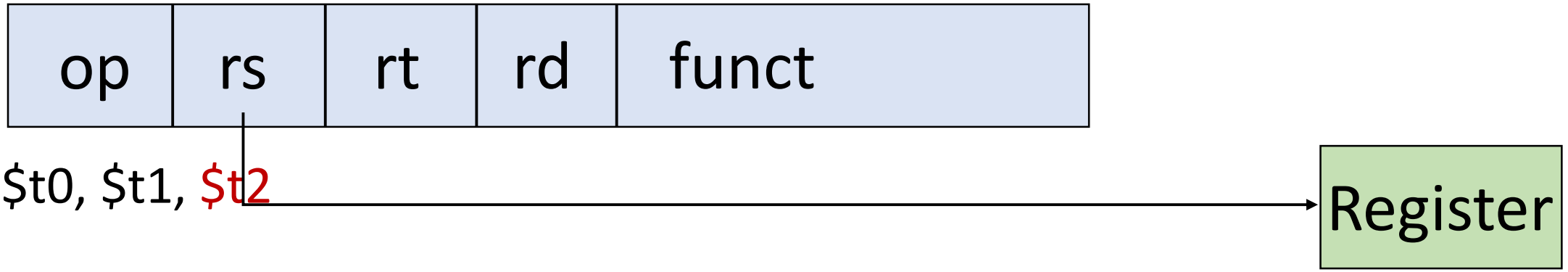
Five Common Addressing Modes

Immediate

addi $t0, $t1, 5

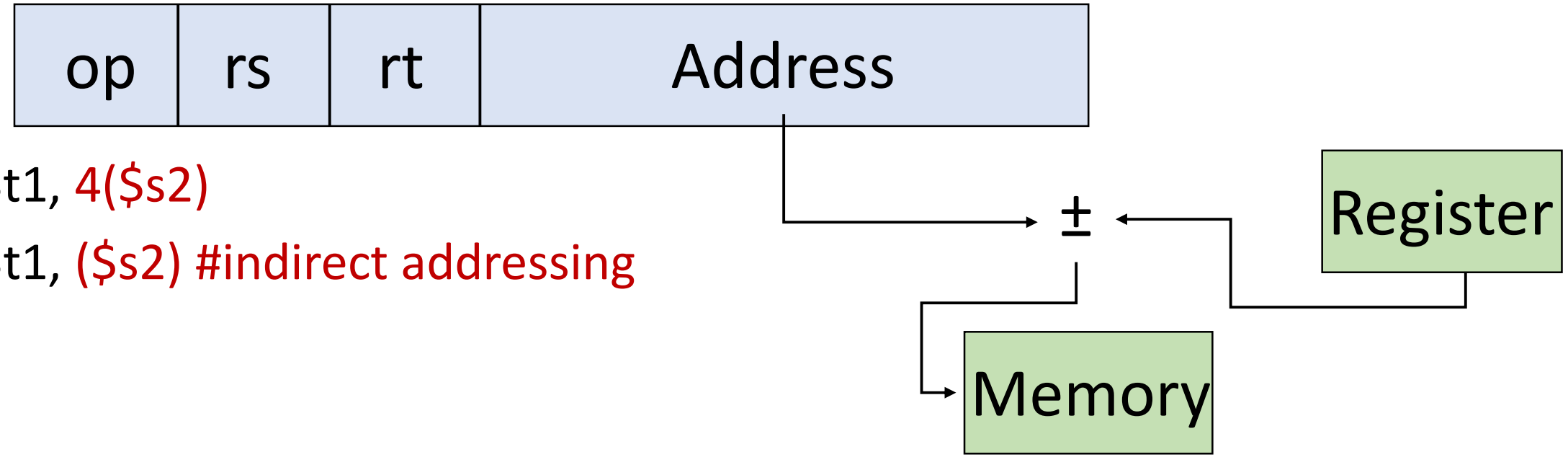
Five Common Addressing Modes

Register



Five Common Addressing Modes

Base (Arrays, structures, pointers)

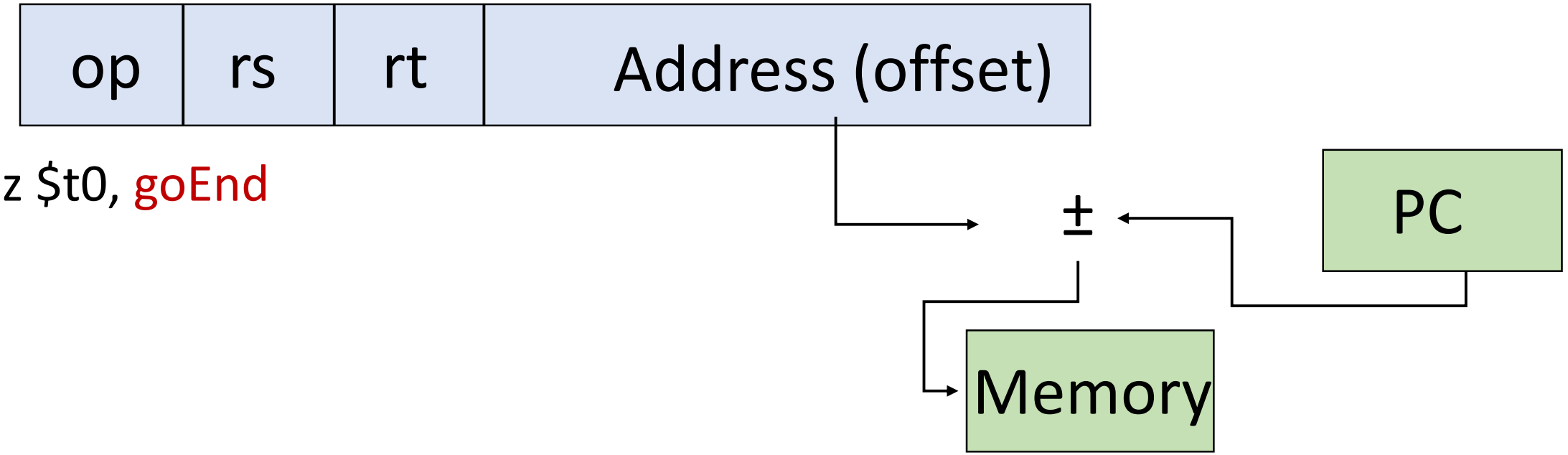


lw $t1, 4($s2)

lw $t1, ($s2) #indirect addressing

Five Common Addressing Modes

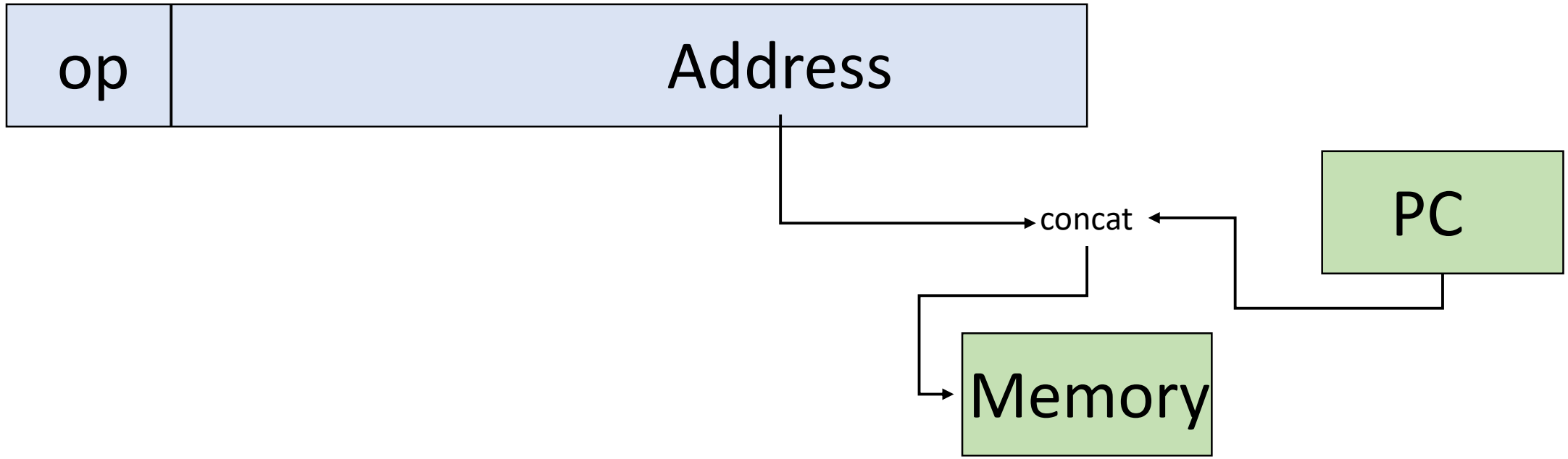
PC-relative (e.g., conditional branches, need an offset)



beqz $t0, goEnd

Five Common Addressing Modes

Pseudodirect



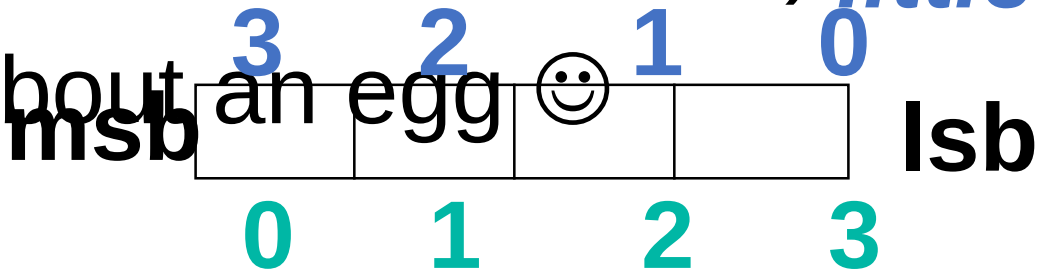
jal

Takes upper four bits of PC and concatenate 26 bits from address field and then add two zeros at the lsb. PS: It is for unconditional jumps.

Endianness (Byte ordering within a word)

- Big Endian: address of most significant byte = word address
  (xx00 = Big end of word), MIPS
- Little Endian: address of least significant byte = word address
  (xx00 = Little end of word), x86

Think about an egg



little endian byte 0

big endian byte 0

Just for an example, do not take it for granted …

```
unsigned int i = 1;
char *c = (char*)&i;  // reading the LSB
Printf (“%d”, *c);
```

```
unsigned int i = 12345678;
char *c = (char*)&i;
Printf (“%d”, *c);
```

```
unsigned int i = 1;
char *c = (char*)&i;  // reading the LSB
Printf (“%d”, *c);
```

Little endian: 1

Big endian: 0

```
unsigned int i = 12345678;
char *c = (char*)&i;
Printf (“%d”, *c);
```

Little endian: 78

Big endian: 12

Coffee credits

Hari: +1

নমস্কার