Instruction Formats

- 3 types: R-format (register format), I-format (Immediate format), and J-format (Jump format)
- Each 32 bits long
- OPCODE always in bits 26-31 (6 bits wide)
- These 6 bits are sufficient to tell the hardware what we want it to do
- The remaining 26 bits are used for operands for those commands
- The type of format that we use is largely dependent on the operands that we have to use for that instruction

R-format (Register format)

- Contains:
  - Operation code (6 bits) → 000000 for R-type instructions
  - 3 register fields (5 bits each) → rs: first source register; rt: second source register; rd: destination register
  - shift amount field (5 bits)
  - function code (6 bits)

OP	RS	RT	RD	ShAmt	FCT
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

* eg: add, sub, and, or, slt, sll, srl

- Only format that can handle having 3 registers
- Will frequently have 2 inputs coming from registers, and a third register as a destination
- Useful for a lot of arithmetic instructions
- ShAmnt will only be used for shift instructions, otherwise it will be set to 0
- FCT gives us a way to control what the ALU is doing

I-format (Immediate format)

- Contains:
  - OPCODE (6 bits)
  - 2 register fields (5 bits each) → rs: source register; rt: destination register
  - Immediate (16 bits)

OP	RS	RT	Immediate
6 bits	5bits	5bits	16 bits

* eg: addi, lw, sw, beq, bne, slt

- Used for Add Immediate (addi) instruction where we add some constant value to the contents of a register and store the result in a different register
- Used for load and store instructions, where we have a base address in a register, a constant offset that we put in the instruction, and then some other register that we want to write to
- Branch instructions use the immediate field for a PC-relative offset:
  ↳ the immediate field tells us how far backwards or forwards we want to go from our current position

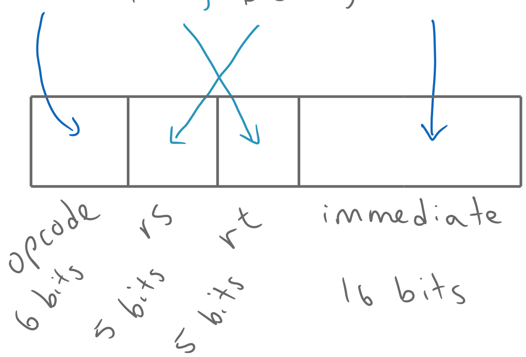
Addressing Modes

A : contents of an address field in the instruction

Immediate Addressing (MIPS uses it)

eg: addi (add immediate)

addi $s0, $t1, -24 # $s0 ← $t1 - 24



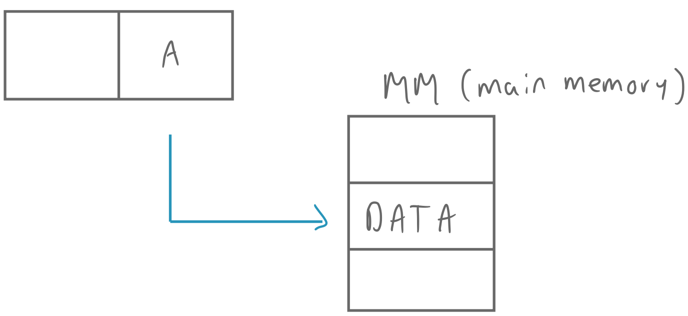
Other I-type instruction

ori $s0, $t1, 0xAB05

$s0 ← $t1 | 0xAB05

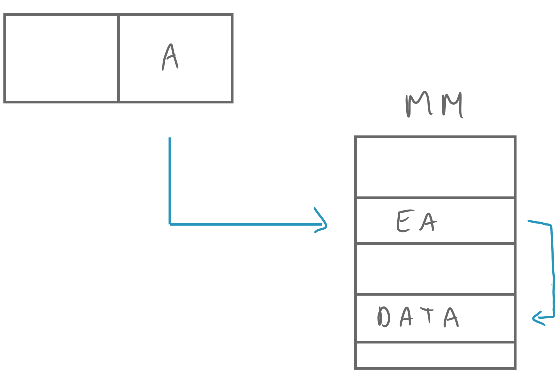
bitwise OR immediate

Direct Addressing (MIPS doesn't use it)



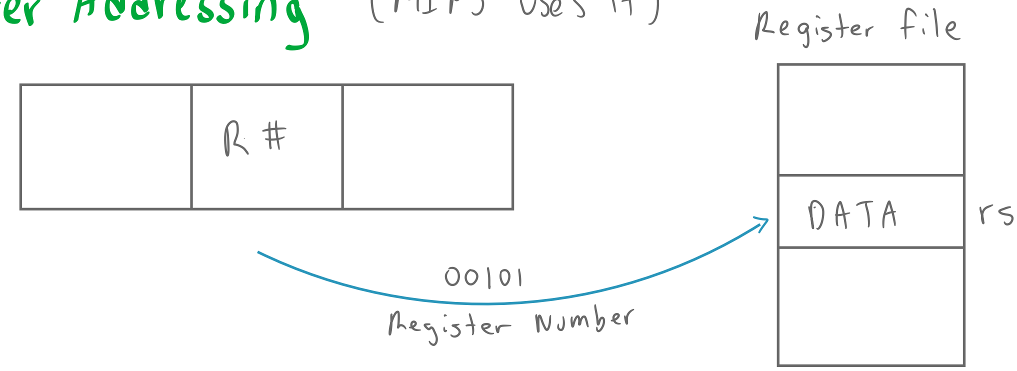
The address field A in the instruction directly specifies the memory location where the data is stored

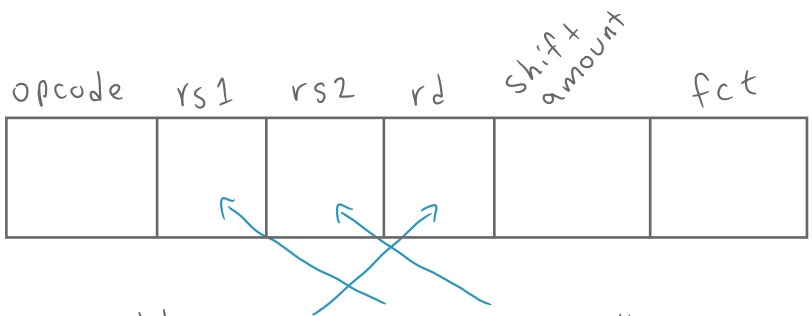
Indirect Addressing (MIPS doesn't use it)



The address field A in the instruction does not contain the actual data's memory location. Instead, A holds a memory address where the effective address (EA) is stored. The processor first retrieves the EA from this memory location, then uses it to access the actual data in main memory.

Register Addressing (MIPS uses it)

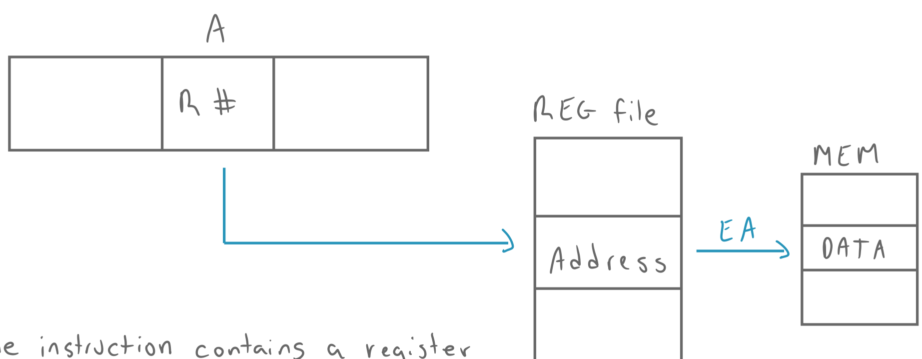




add $s1, $t1, $t0   # $s1 ← $t1 + $t0

- the instruction contains a Register Number
- the processor retrieves the data from the specified register in the register file
- this eliminates the need to access memory, improving speed

Register Indirect Addressing (MIPS doesn't use it)



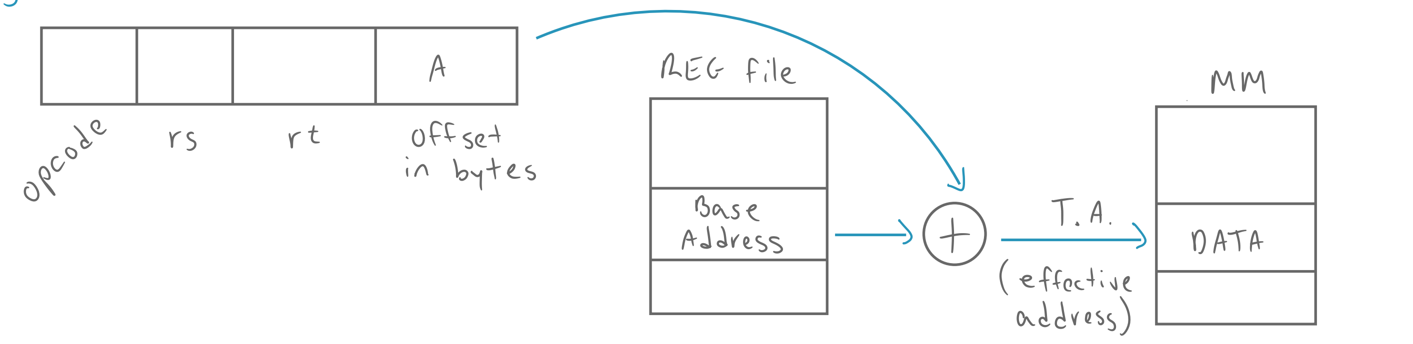
- The instruction contains a register number that holds a memory address.
- The processor fetches the effective address from the specified address in the register file.
- The processor uses this EA to access memory and retrieve data

Base-Register Addressing (Displacement Addressing) (MIPS uses it)

Used for lw and sw

T.A. = Base Address + offset (target address)

eg: lw $t1, 24($s0) # load



- The base register holds a memory address
- The offset is a signed immediate included in the instruction
- The processor computes the target address (TA) by adding the offset to the value in the base register
- The TA is then used to access memory

Example

Consider an instruction. The address field of the instruction contains the value 2000.

When needed, register #18 is used. Register 18 contains the value 1600.

The list below shows a few addresses and the memory content of each of those addresses.

Address (bytes)	Memory Content
48	844
2000	3000
1600	400
2500	800
3000	1200
3600	500

=>

Addressing Modes	Effective Address (bytes)	Value
IMMEDIATE	—	2000
DIRECT	1200	3000
INDIRECT	3000	1200
REGISTER	REG # 18	1600
REGISTER INDIRECT	1600	400
DISPLACEMENT	3600	500

We'll refer to the content of the address field of the instruction as A

- Immediate: A is 2000, grab it immediately
- Direct: A is 2000, an address of a mem location that contains our data => 3000
- Indirect: A is 2000, an address of a mem location that contains an address (3000) of a mem location that contains our data => 1200
- Register: Uses the register number given in the instruction (18) which contains our data => 1600
- Register Indirect: Uses the register number given in the instruction (18) which contains an address (1600) of a mem location that contains our data => 400
- Displacement: EA = (value in base register) + (offset from instruction)
  = Value in register # 18 + offset of 2000
  = 1600 + 2000 = 3600 => ADDR 3600 contains our data
  => 500