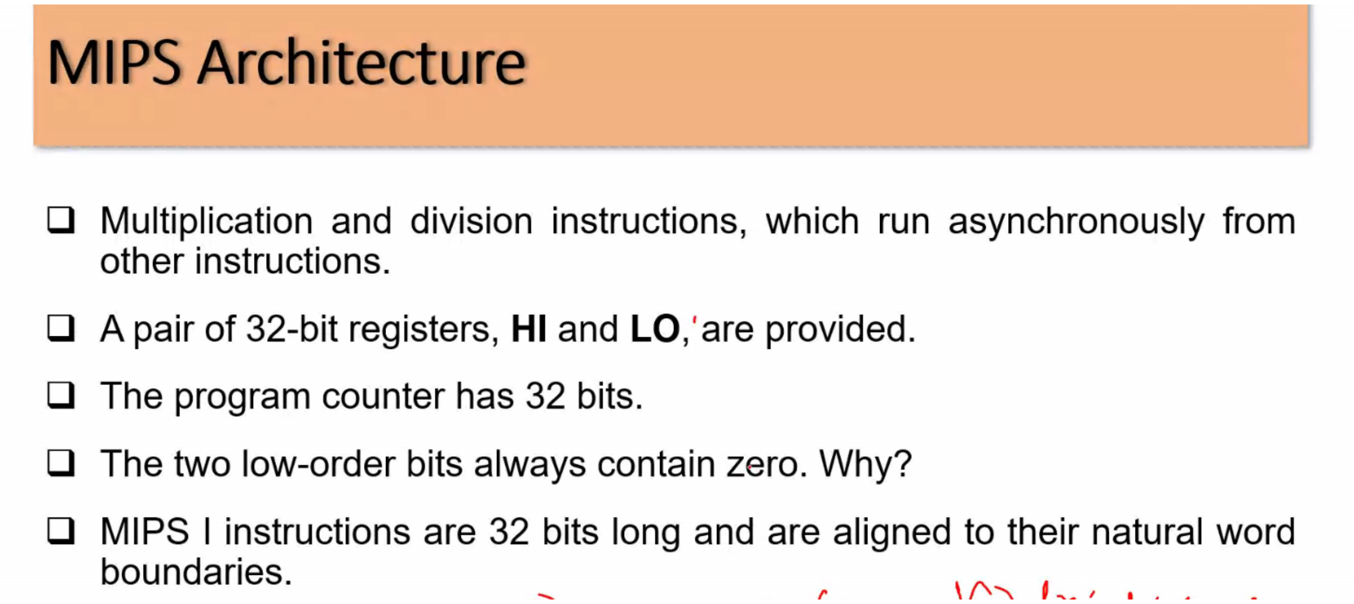


MIPS Architecture

❑ Multiplication and division instructions, which run asynchronously from other instructions.

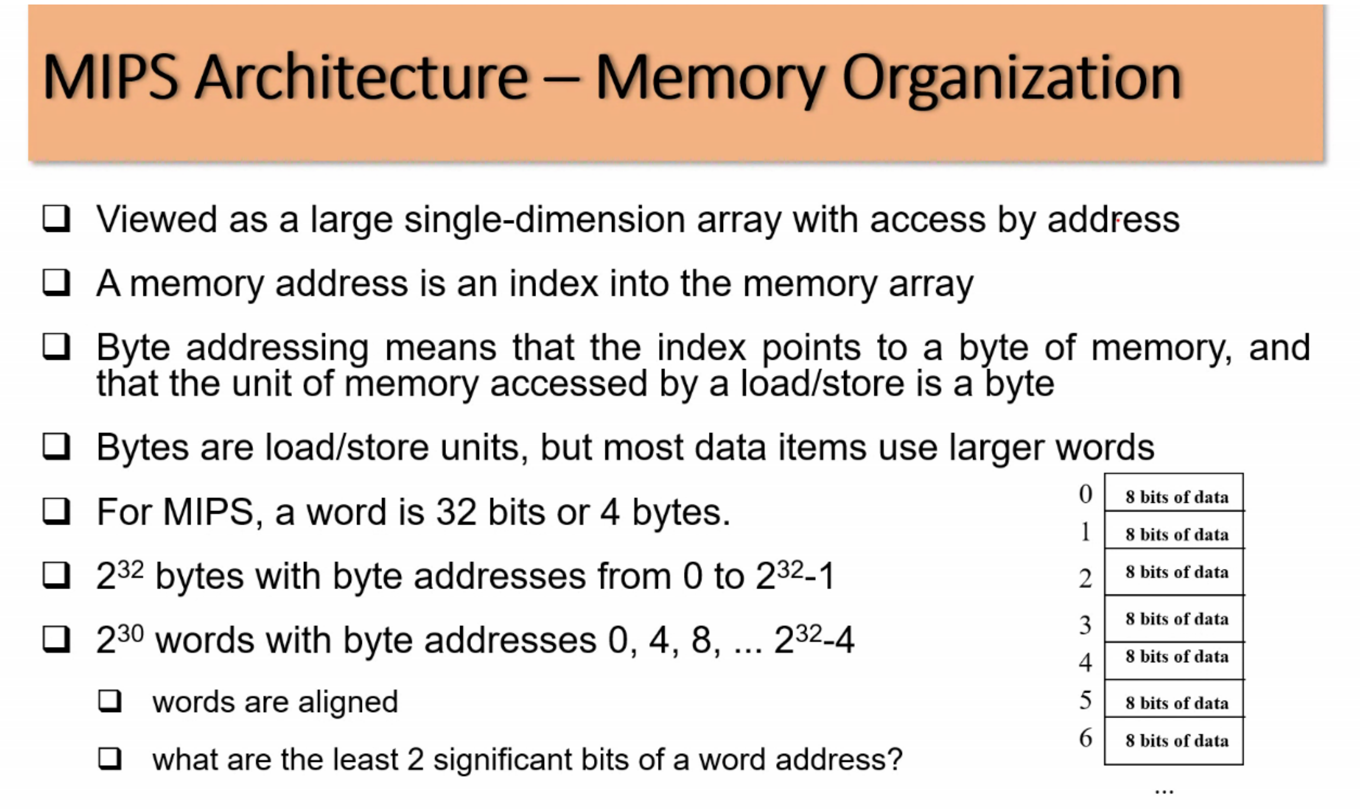
❑ A pair of 32-bit registers, **HI** and **LO**, are provided.

❑ The program counter has 32 bits.

❑ The two low-order bits always contain zero. Why?

❑ MIPS I instructions are 32 bits long and are aligned to their natural word boundaries.

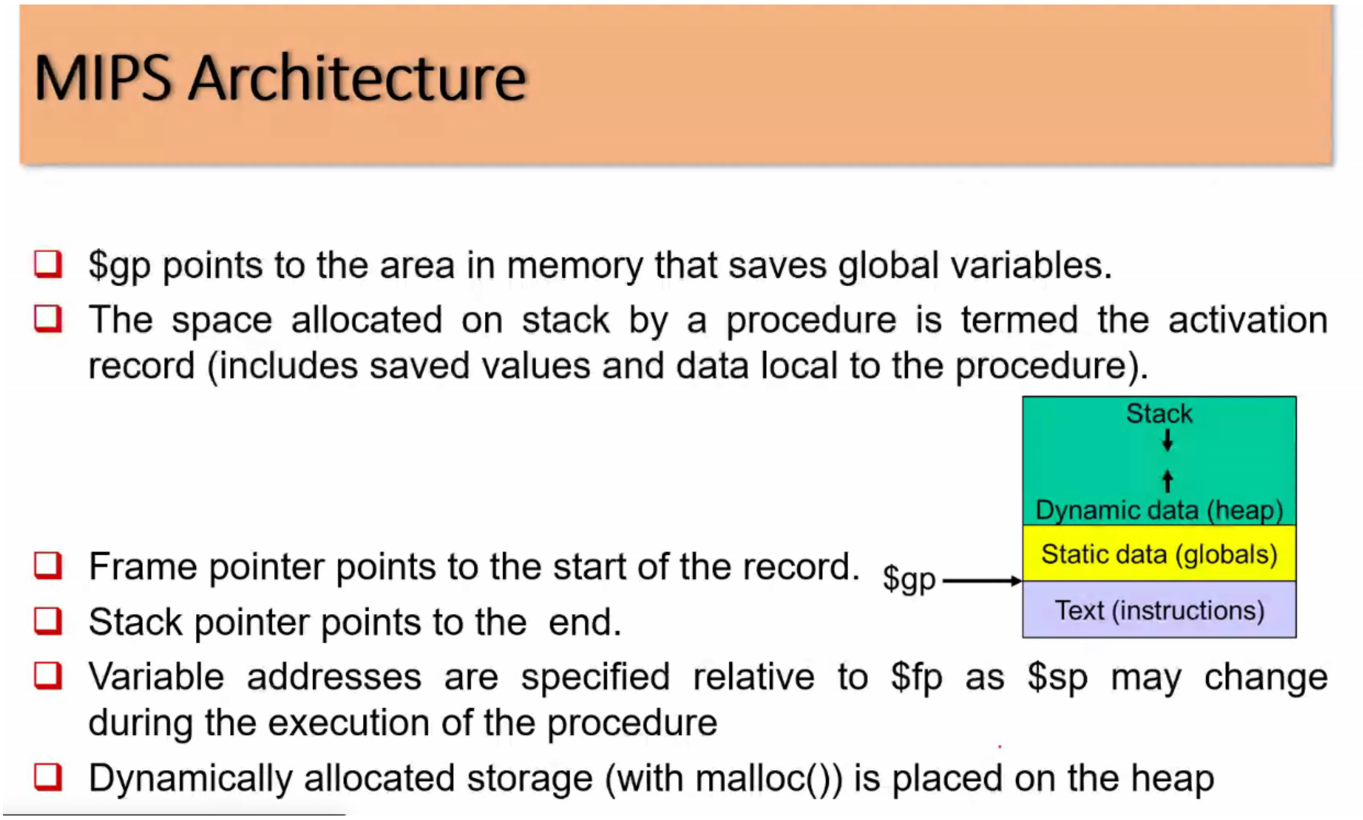
- In mips each instr is 4 bytes(32 Bits). Program counter is always incremented by 4 every time
- Values must be fetched from memory before any (add, sub) instructions are carried out on them.



MIPS Architecture – Memory Organization

- ❑ Viewed as a large single-dimension array with access by address
- ❑ A memory address is an index into the memory array
- ❑ Byte addressing means that the index points to a byte of memory, and that the unit of memory accessed by a load/store is a byte
- ❑ Bytes are load/store units, but most data items use larger words
- ❑ For MIPS, a word is 32 bits or 4 bytes.
- ❑ 2^{32} bytes with byte addresses from 0 to 2^{32}-1
- ❑ 2^{30} words with byte addresses 0, 4, 8, ... 2^{32}-4
  - ❑ words are aligned
  - ❑ what are the least 2 significant bits of a word address?

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data
	...



MIPS Architecture

- ❑ $gp points to the area in memory that saves global variables.
- ❑ The space allocated on stack by a procedure is termed the activation record (includes saved values and data local to the procedure).
- ❑ Frame pointer points to the start of the record.
- ❑ Stack pointer points to the end.
- ❑ Variable addresses are specified relative to $fp as $sp may change during the execution of the procedure
- ❑ Dynamically allocated storage (with malloc()) is placed on the heap

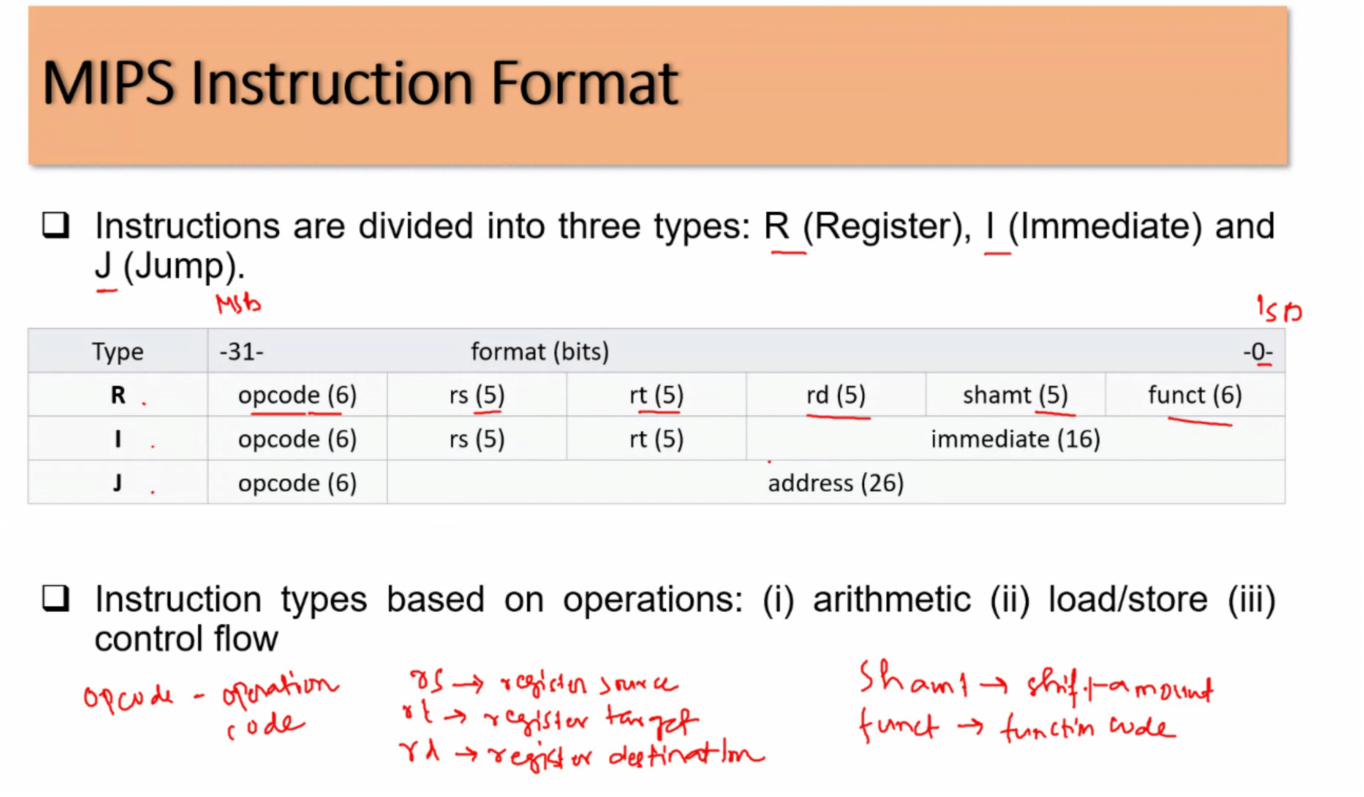
- Each Frame pointer is 64k
- Frame pointer is fixed. Stack pointer moves.
- stack pointer moves downwards/stack grows downwards.

Mips instruction format

- Instructions of 3 types: R(register) I(Immediate), J(Jump)

Type				**Format**		
R	opcode(6)	rs(5)	rt(5)	rd(5)	shamt(5)	funct
I	opcode(6)	rs(5)	rt(5)		immediate(16)	
J	opcode(6)				address(26)	

- opcode - operation code
- rs - register source
- rt - register target
- rd - register destination
- shamt - shift amount
- funct - function code



MIPS Instruction Format

- ❑ Instructions are divided into three types: R (Register), I (Immediate) and J (Jump).

MSb lsb

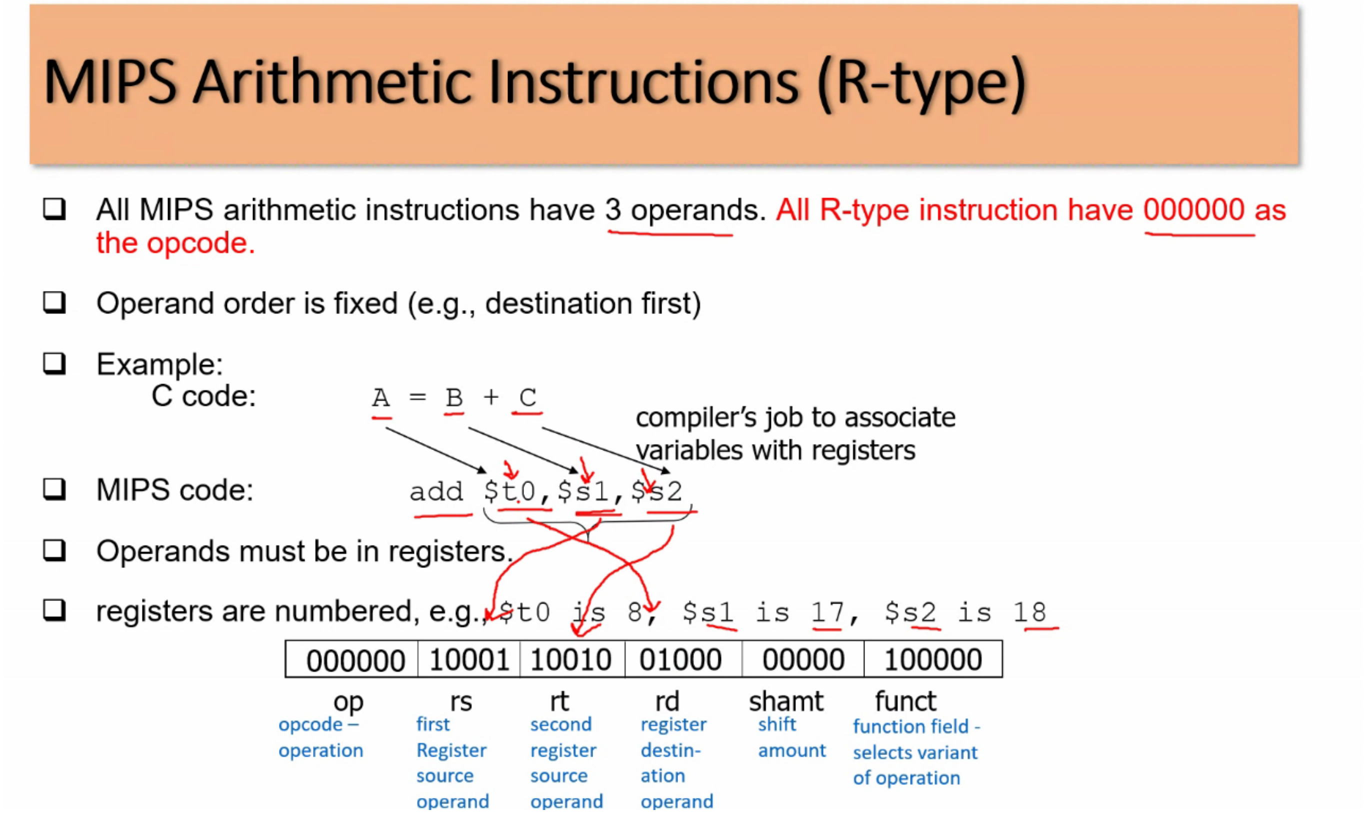
Type	-31-	format (bits)				-0-
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

- ❑ Instruction types based on operations: (i) arithmetic (ii) load/store (iii) control flow

opcode - operation code

rs → register source
rt → register target
rd → register destination

shamt → shift-amount
funct → function code



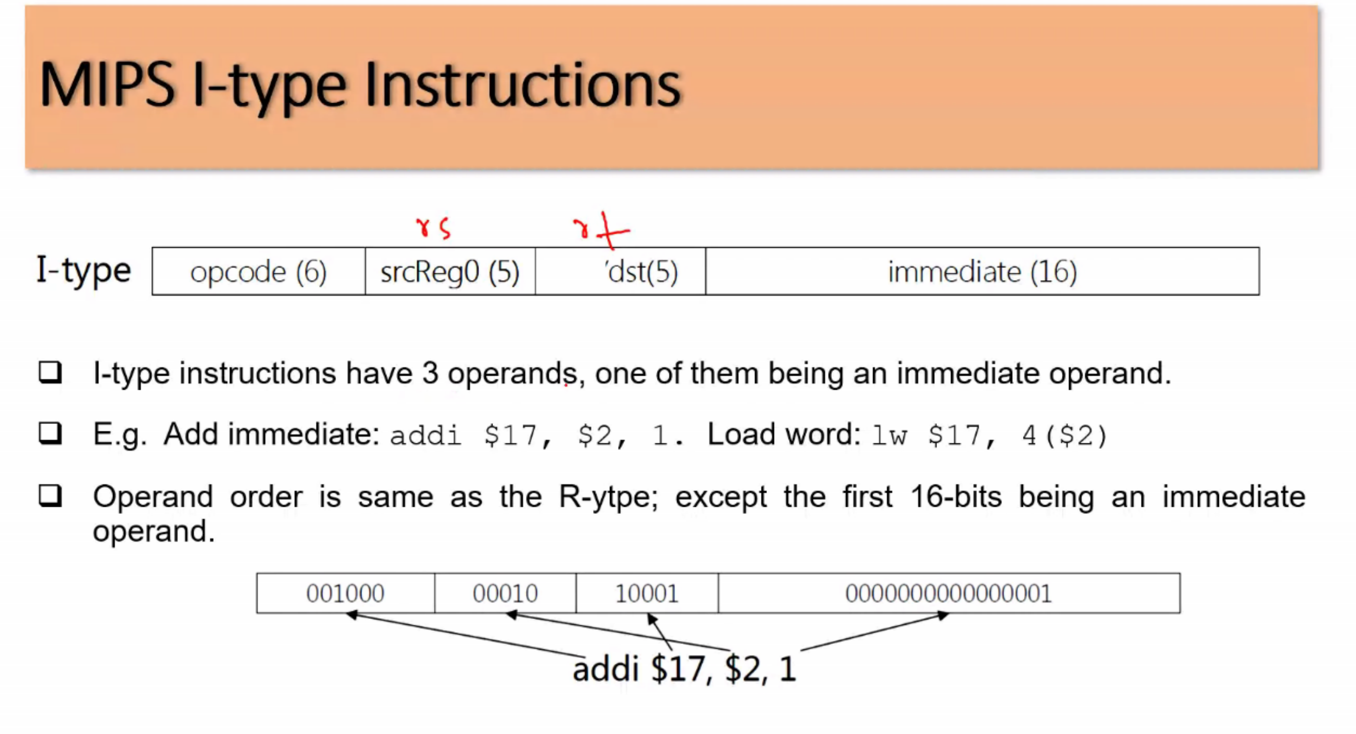
MIPS Arithmetic Instructions (R-type)

- ❑ All MIPS arithmetic instructions have 3 operands. All R-type instruction have 000000 as the opcode.
- ❑ Operand order is fixed (e.g., destination first)
- ❑ Example:
  C code: `A = B + C`

compiler's job to associate variables with registers

- ❑ MIPS code: `add $t0,$s1,$s2`
- ❑ Operands must be in registers.
- ❑ registers are numbered, e.g., `$t0 is 8, $s1 is 17, $s2 is 18`

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct
opcode – operation	first Register source operand	second register source operand	register destin-ation operand	shift amount	function field - selects variant of operation



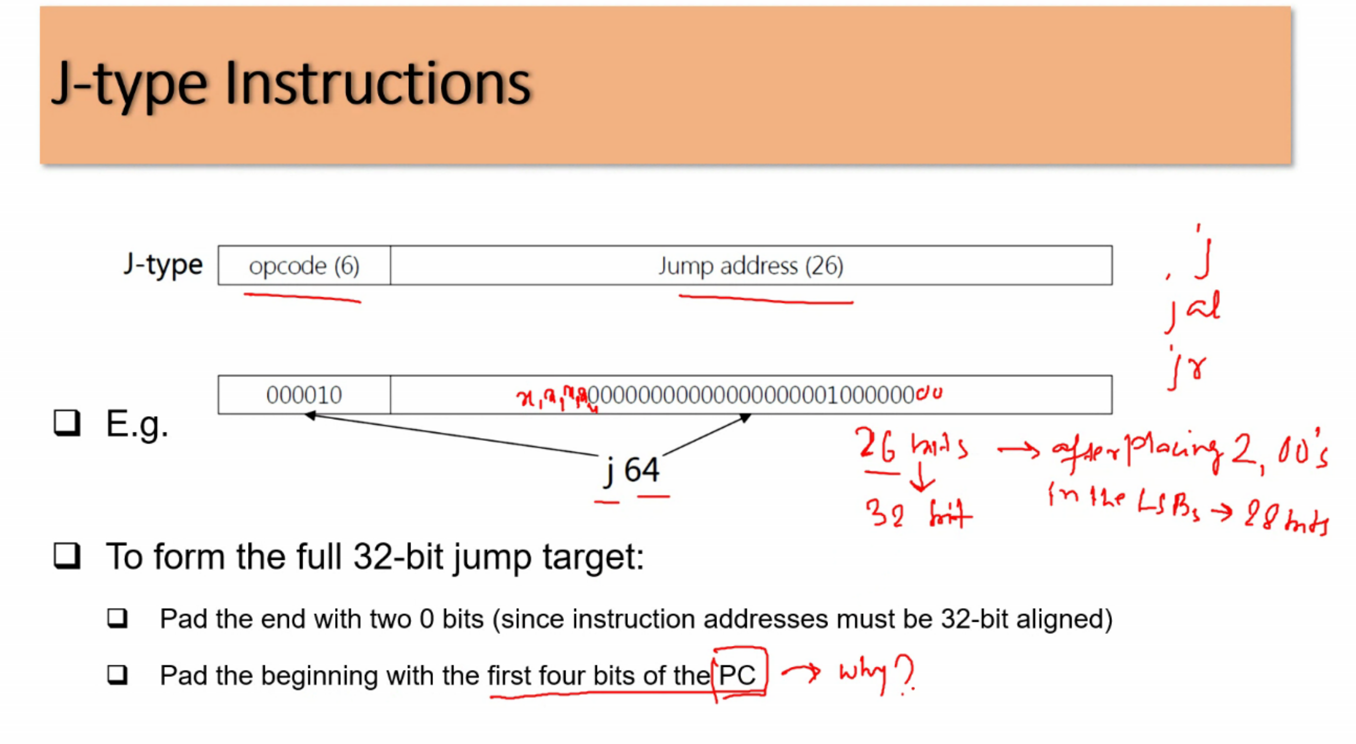
MIPS I-type Instructions

I-type	opcode (6)	srcReg0 (5)	'dst(5)	immediate (16)
		rs	rt	

- ❑ I-type instructions have 3 operands, one of them being an immediate operand.
- ❑ E.g. Add immediate: `addi $17, $2, 1`. Load word: `lw $17, 4($2)`
- ❑ Operand order is same as the R-ytpe; except the first 16-bits being an immediate operand.

001000	00010	10001	0000000000000001

addi $17, $2, 1



J-type Instructions

J-type	opcode (6)	Jump address (26)

j
jal
jr

❑ E.g.

000010	x1x2x3x400000000000000000001000000 00

j 64

26 bits → after placing 2, 00's in the LSBs → 28 bits
↓
32 bit

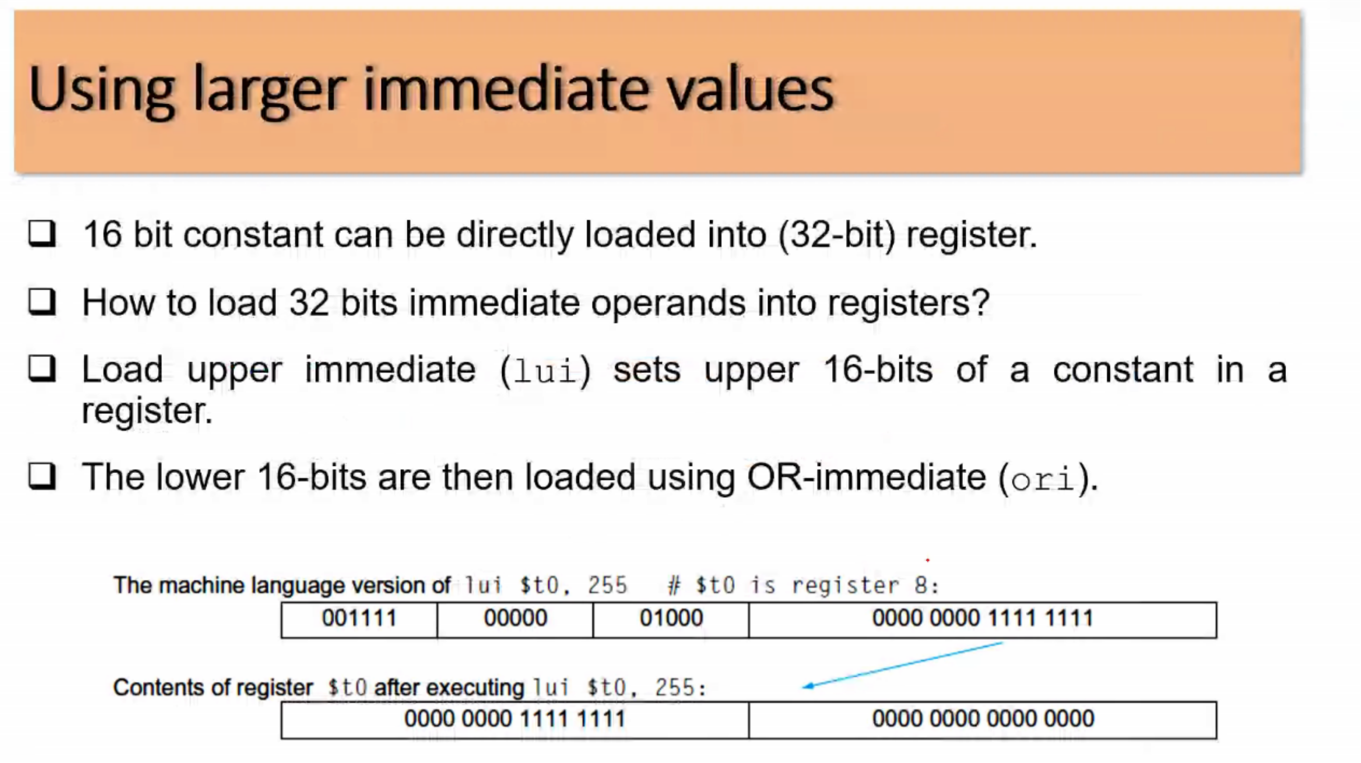
❑ To form the full 32-bit jump target:

- ❑ Pad the end with two 0 bits (since instruction addresses must be 32-bit aligned)
- ❑ Pad the beginning with the first four bits of the PC → why?

How to convert 26 bit address to 32 bit address.

- place 2 00's in the LSB => 28 bits
- pad the beginning with the first four bits of program counter.

To load >16 Bit values in immediate



Using larger immediate values

- ❑ 16 bit constant can be directly loaded into (32-bit) register.
- ❑ How to load 32 bits immediate operands into registers?
- ❑ Load upper immediate (`lui`) sets upper 16-bits of a constant in a register.
- ❑ The lower 16-bits are then loaded using OR-immediate (`ori`).

The machine language version of `lui $t0, 255   # $t0 is register 8:`

001111	00000	01000	0000 0000 1111 1111

Contents of register `$t0` after executing `lui $t0, 255:`

0000 0000 1111 1111	0000 0000 0000 0000