

ECE260: Fundamentals of Computer Engineering

MIPS Instruction Set

James Moscola
Dept. of Engineering & Computer Science
York College of Pennsylvania



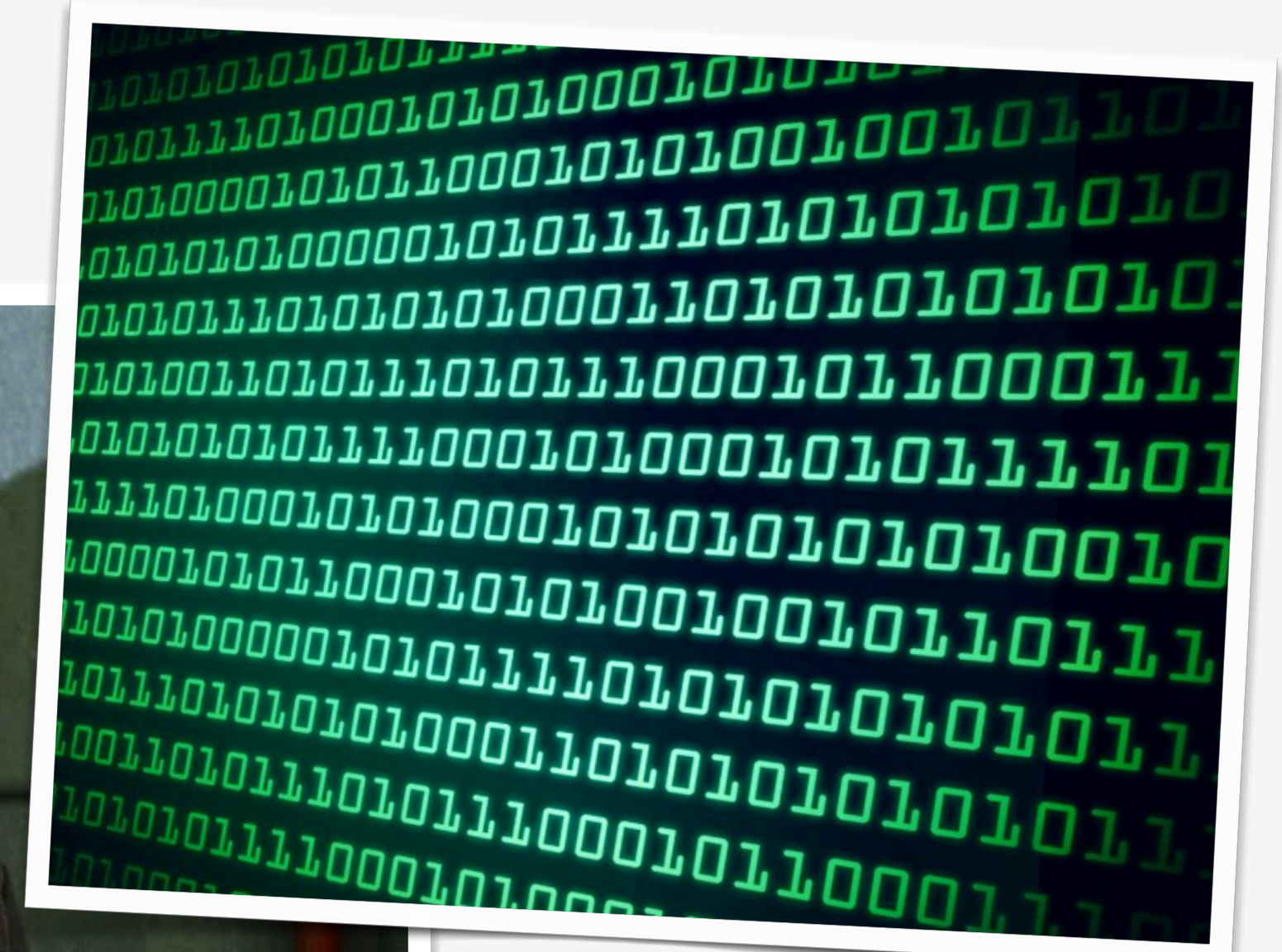
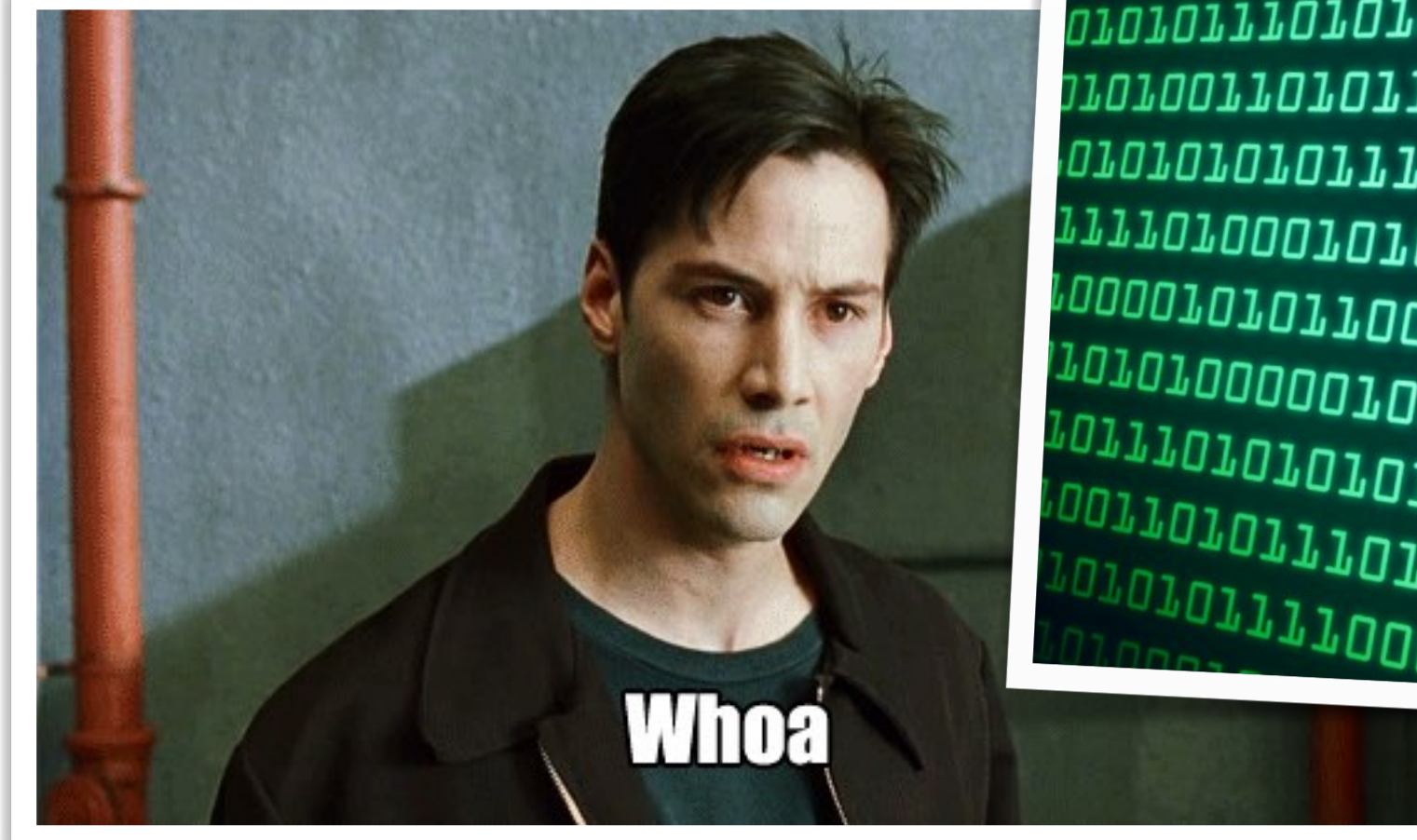
MIPS Registers

- MIPS architecture has a 32×32 -bit register file (e.g. it has 32 32-bit registers)
 - Registers are numbered 0 to 31
 - Registers can be referenced in assembly code using various names

Register Number	Register Name	Use
0	\$zero	Constant value 0
1 - 7	Coming soon ...	Coming soon ...
8 - 15	\$t0 - \$t7	Temporary values
16 - 23	\$s0 - \$s7	Saved temporary values
24 - 25	\$t8 - \$t9	Temporary values
26 - 31	Coming soon ...	Coming soon ...

Representing Instructions

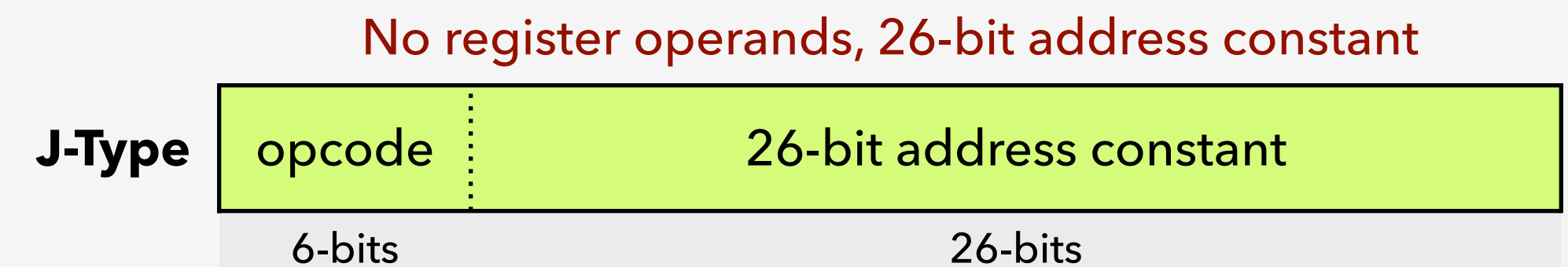
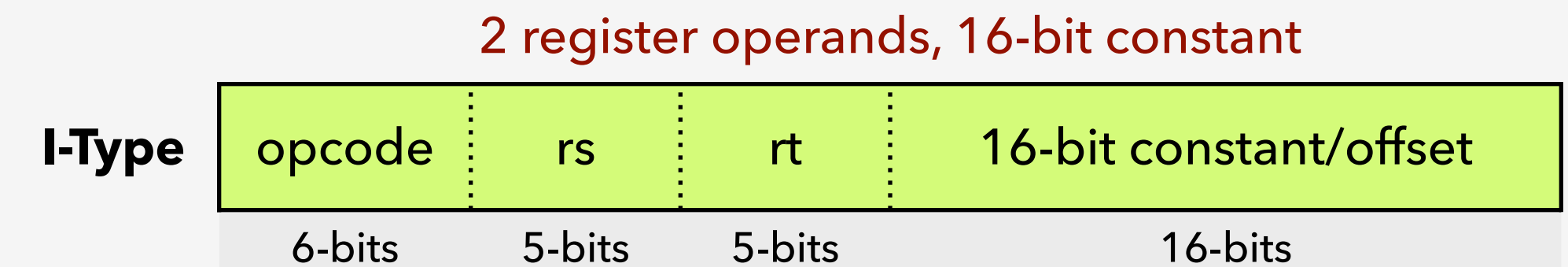
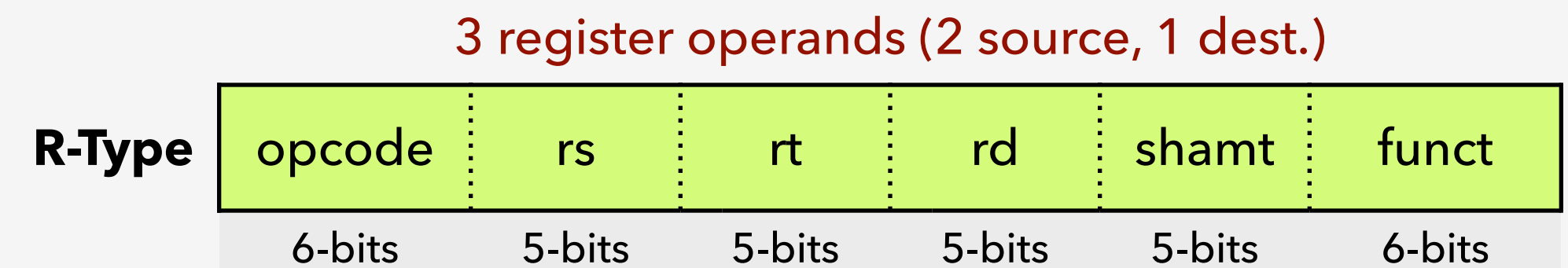
- All instructions are encoded in binary
 - Binary values represent type of instruction, source and destination registers, etc.
- Sequence of binary instructions is referred to as **machine code**
 - Machine code is the output of an assembler
 - Contents of executable application files is machine code
 - Loaded into **TEXT** section of memory when application runs
 - Interpreted by CPU as program executes



MIPS Instruction Formats

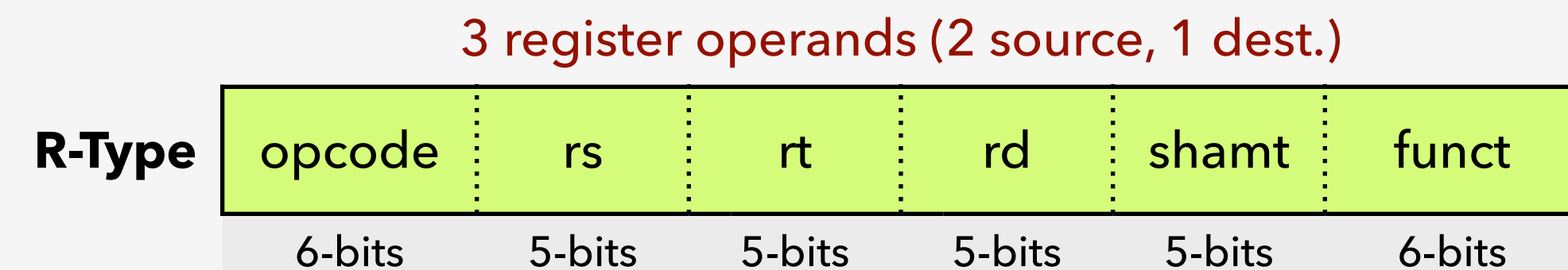
- MIPS instructions are encoded as 32-bit words
- MIPS instructions include multiple "fields":
 - A 6-bit operation or "OPCODE"
 - Specifies which operation to execute (fewer than 64)
 - Up to three 5-bit OPERAND fields
 - Each specifies a register (one of 32) as source/destination
 - May contain embedded constants or immediate values
 - 5-bits, 16-bits, or 26-bits long
 - Can be treated as signed or unsigned

- Three basic MIPS instruction formats



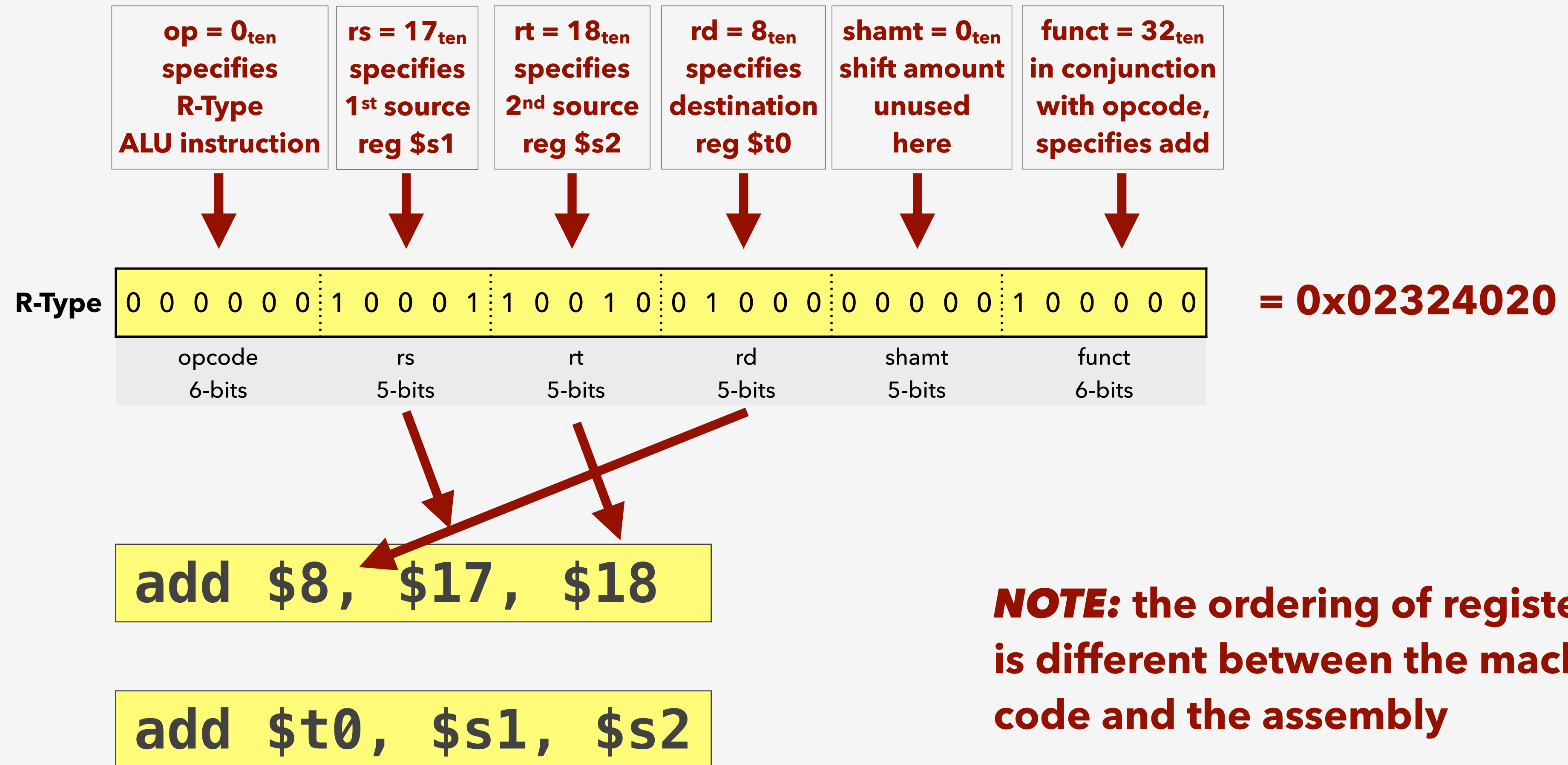
MIPS R-Type Instructions

- Used for operations that require two source operands and for shift operations
 - add, addu, sub, subu, div, divu, mult, multu, and, or, nor, xor, etc.
 - sll, srl, sra
- Instruction fields
 - op: operation code (opcode)
 - rs: first source register number
 - rt: second source register number
 - rd: destination register number
 - shamt: shift amount
 - funct: function code (extends opcode)



MIPS R-Type Instruction Example

- An example **add** instruction:



NOTE: the ordering of registers is different between the machine code and the assembly

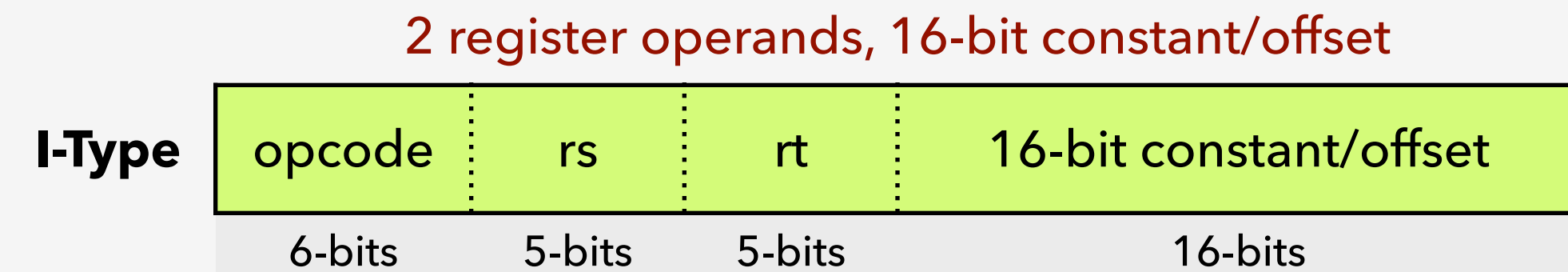
MIPS I-Type Instructions

- Used for immediate arithmetic, load/store, comparisons, branching instructions

- addi, addiu, andi, ori, etc.
- lw, sw, bne, beq, etc.

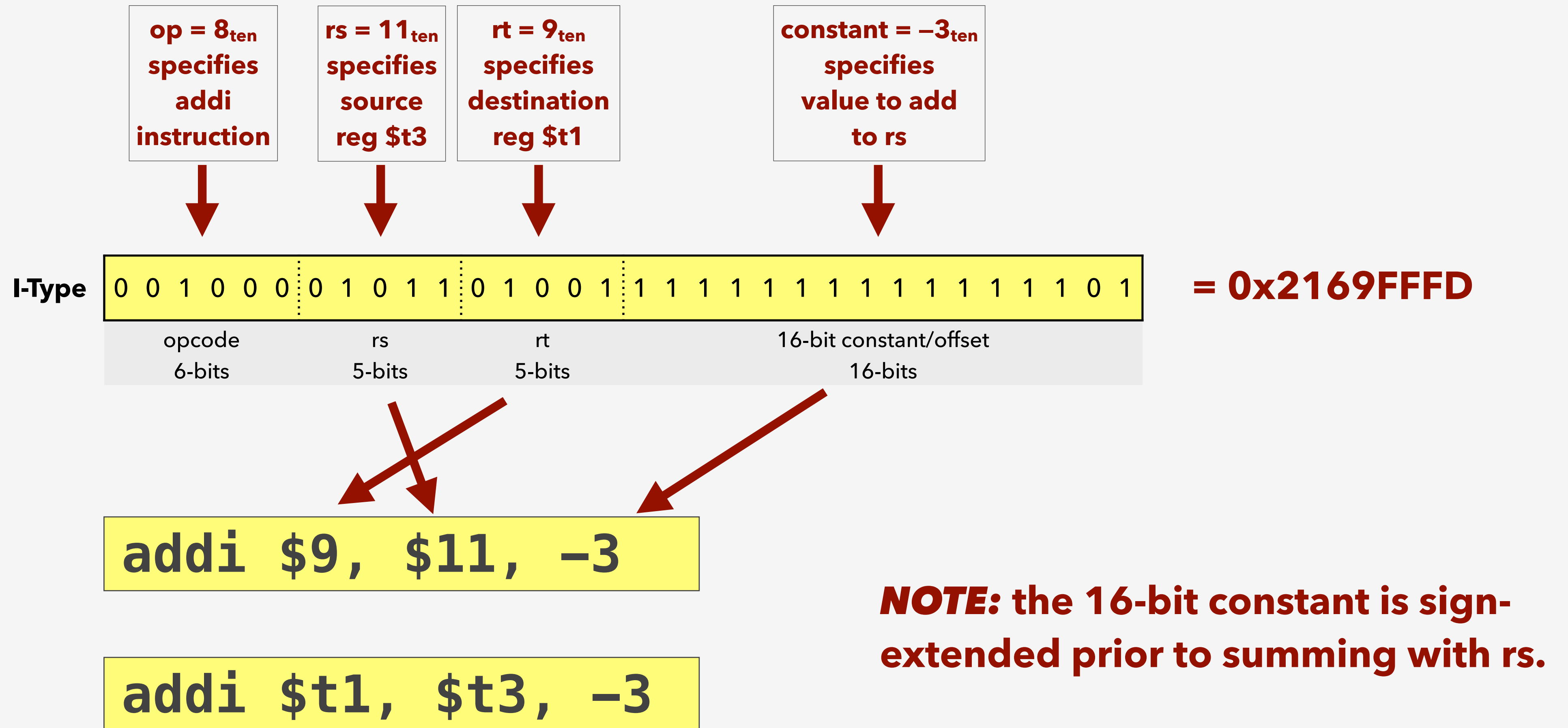
- Instruction fields

- op: operation code (opcode)
- rs: base address for load/store operations
- rt: destination or source register number
- 16-bit constant/offset range: -2^{15} to $+2^{15} - 1$ (-32,768 to 32,767)
 - Constant for instructions that use immediate values
 - Offset for load and store operations (added to base address)
 - Offset for branching operations



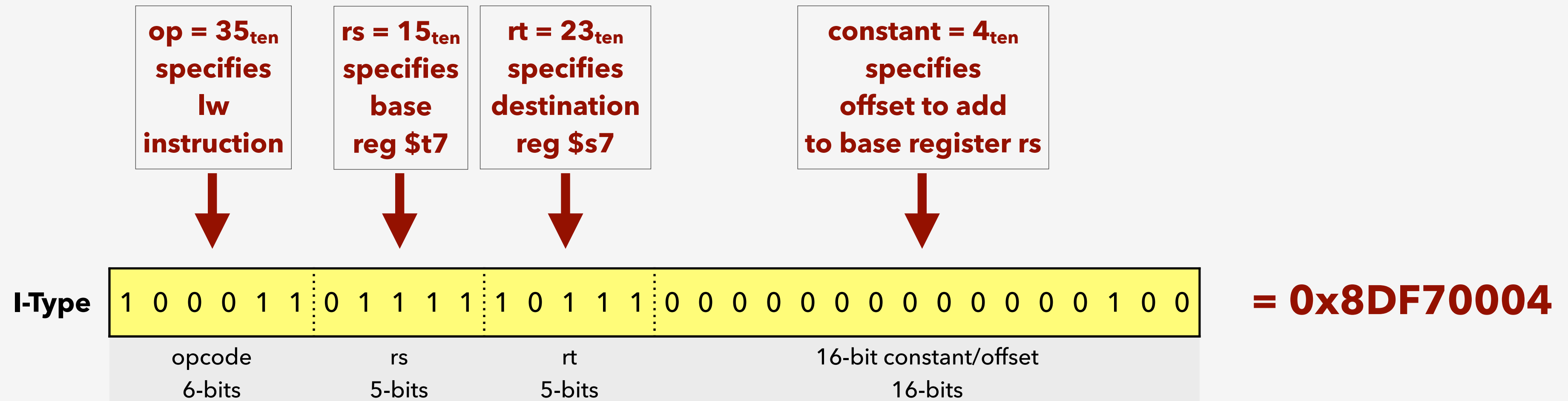
MIPS I-Type Instruction Example (immediate)

- An example **addi** instruction:



MIPS I-Type Instruction Example (load)

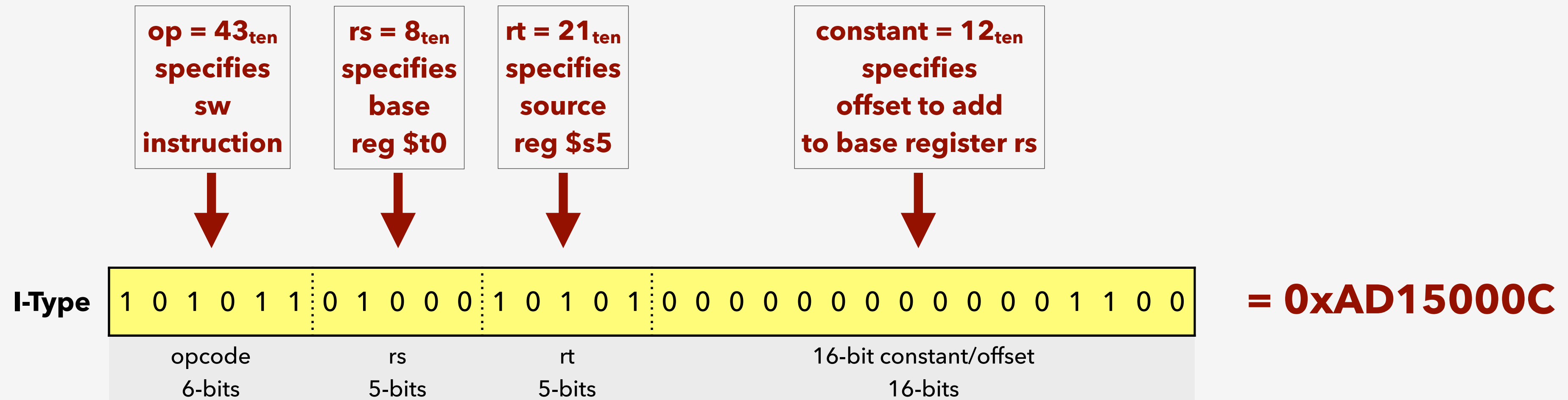
- An example **lw** instruction:



NOTE: the 16-bit constant is sign-extended prior to summing with rs.

MIPS I-Type Instruction Example (store)

- An example **sw** instruction:



sw \$21, 12(\$8)

sw \$s5, 12(\$t0)


NOTE: the 16-bit constant is sign-extended prior to summing with rs.

Logical Operations

- Instructions for bitwise manipulation
- Useful for extracting and inserting groups of bits in a word

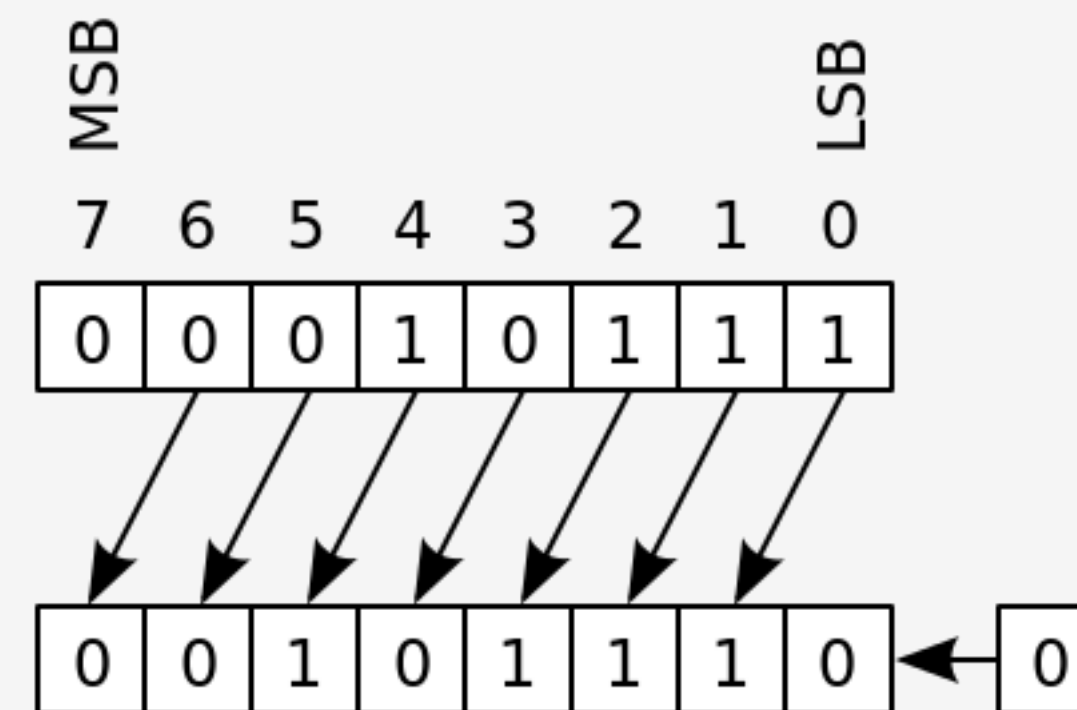
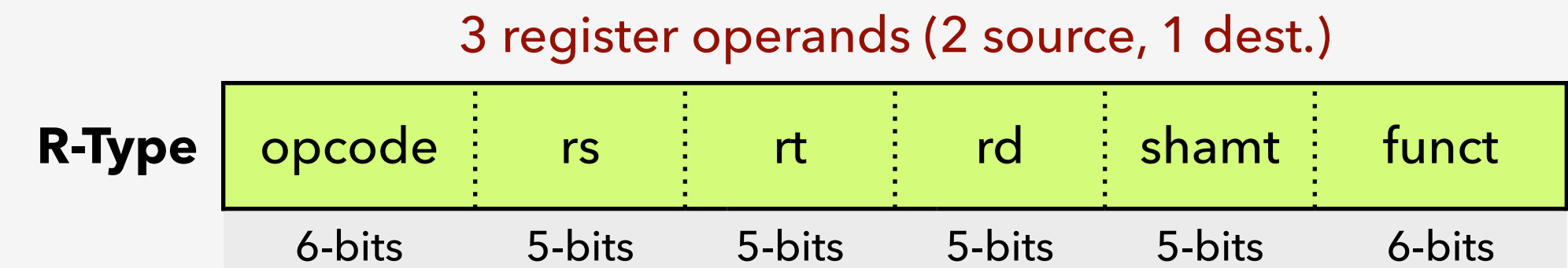
Operation	C / C++	MIPS
Shift Left	<<	sll
Shift Right	>>	srl, sra
Bitwise AND	&	and, andi
Bitwise OR		or, ori
Bitwise XOR	^	xor
Bitwise NOT	~	nor

```
int x = 3;  int y = 4;  int z;  
z = x << 1;  // z is now 6  
z = y >> 1;  // z is now 2  
z = x & y;    // z is now 0  
z = x | y;    // z is now 7  
z = x ^ y;    // z is now 7  
z = ~x;       // z is now 0xffffffffc
```

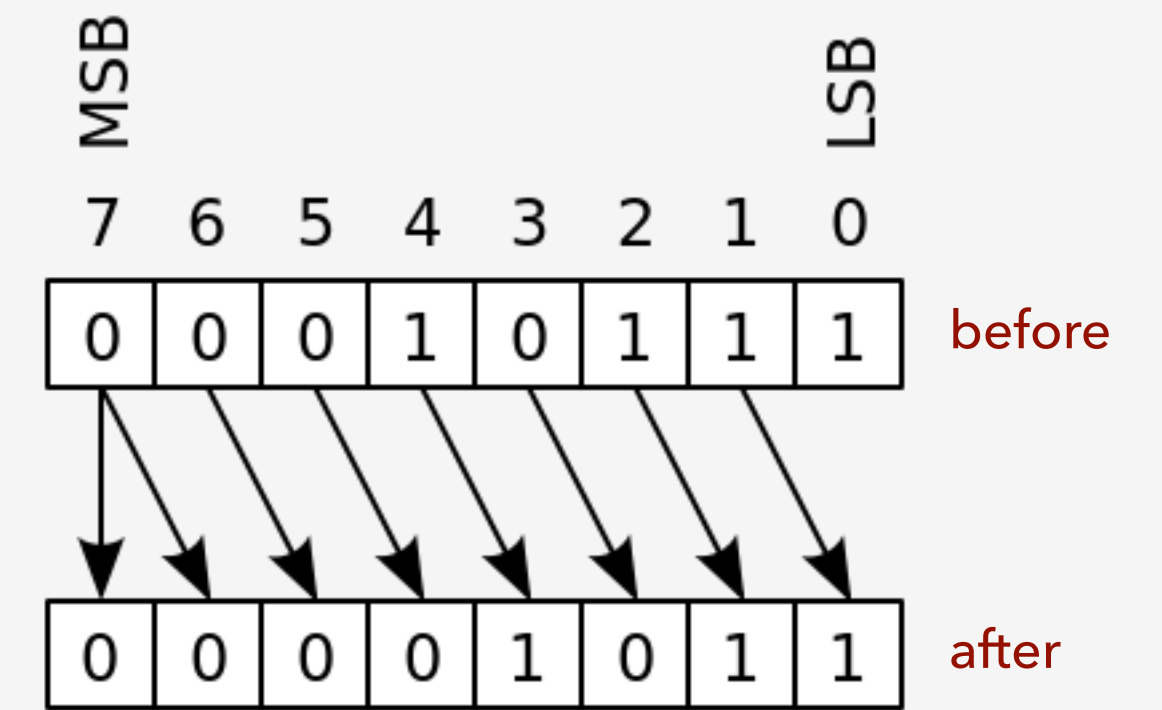

either -4 or
4,294,967,292

Shift Operations

- Uses R-Type instruction
 - shamt field specifies how many positions to shift
- Shift left logical
 - Shift left and fill with 0 bits
 - sll by i bits multiplies by 2^i
- Shift right logical
 - Shift right and fill with 0 bits
 - srl by i bits divides by 2^i (unsigned only)
- Shift right arithmetic
 - Shift right and fill with copy of MSB



Shift Left Logical: The empty position in the least significant bit is filled with a zero.

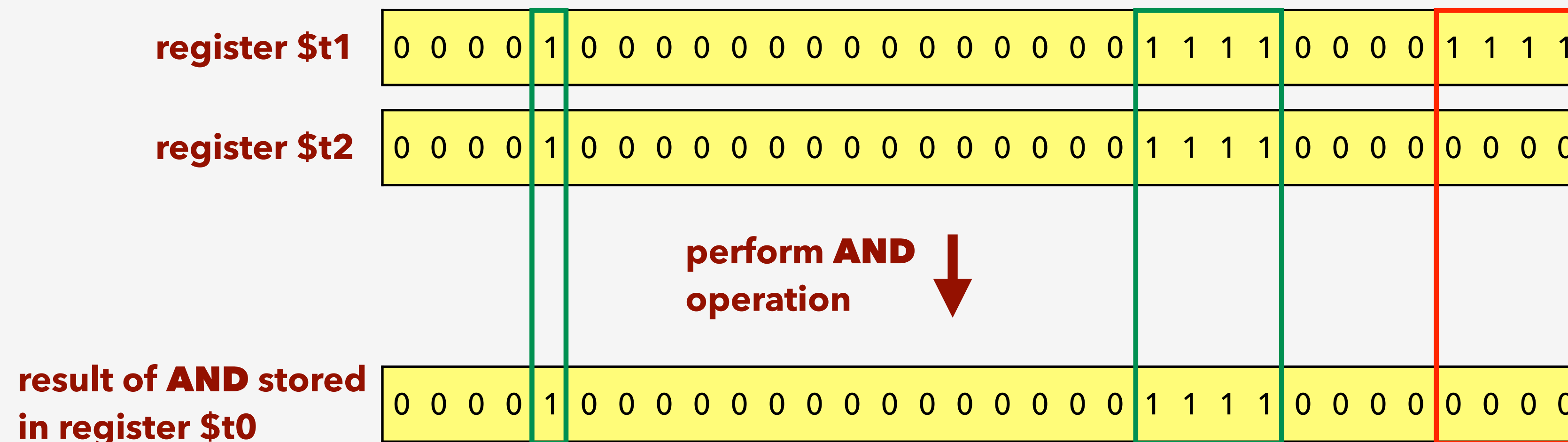


Shift Right Arithmetic: The empty position in the most significant bit is filled with a copy of the original MSB.

AND Operations

- Uses R-Type instruction (opcode = 0x00, funct = 0x24)
- Useful to mask bits in a word
 - Select some bits, clear others to 0

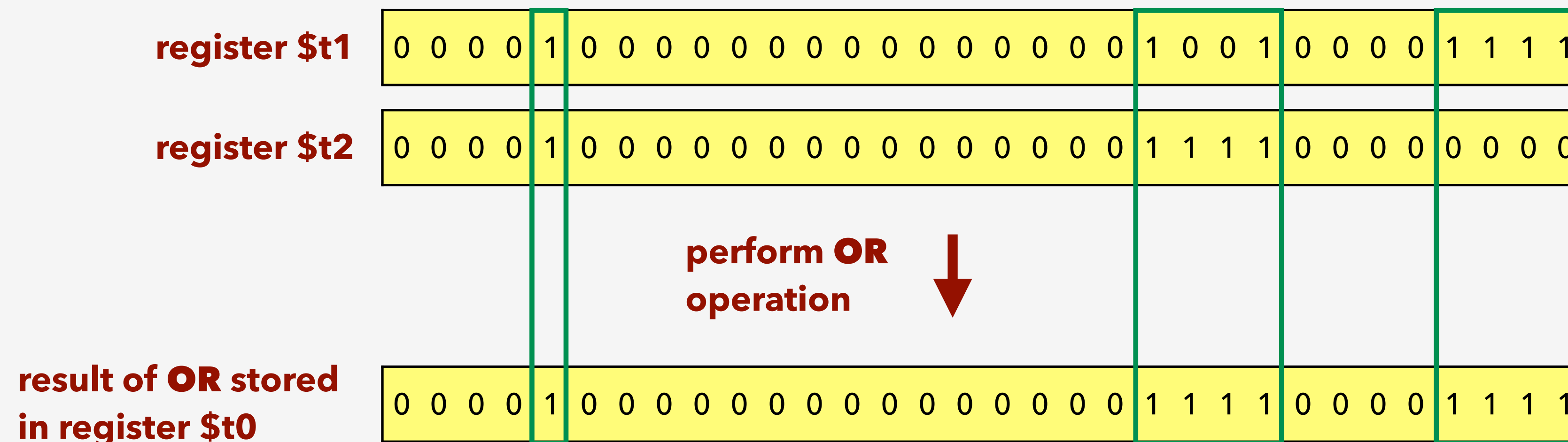
and \$t0, \$t1, \$t2



OR Operations

- Uses R-Type instruction (opcode = 0x00, funct = 0x25)
- Useful to include bits in a word
 - Set some bits to 1, leave others unchanged

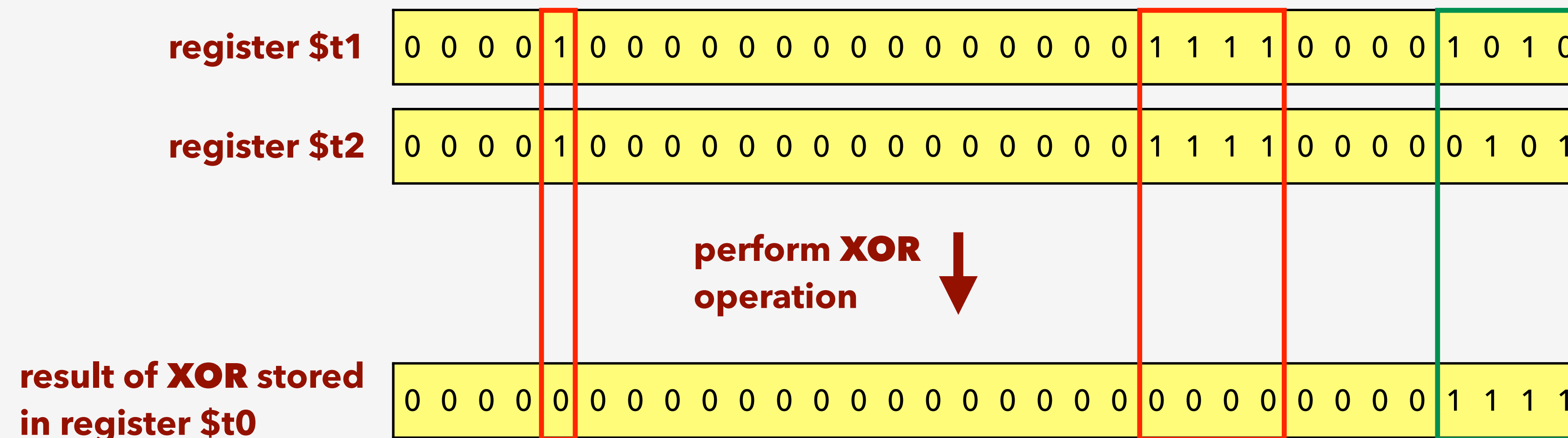
```
or $t0, $t1, $t2
```



XOR Operations

- Uses R-Type instruction (opcode = 0x00, funct = 0x26)
- Useful to find bits that differ between words
 - Set output bit to 1 if input bits differ, others 0

```
xor $t0, $t1, $t2
```



NOT Operations

- MIPS has no **NOT** instruction
 - Instead, use a **NOR** instruction (R-Type) with register \$zero

```
a NOR b == NOT(a OR b) // to get a NOT, just set b=0
```

```
nor $t0, $t1, $zero # negates $t1, stores result in $t0
```

- Useful to invert bits in a word
 - Changes 0's to 1's, and 1's to 0's

register \$t1

0 1 1 0 1 1 0 1

perform **NOR**
operation



result of **NOR** stored
in register \$t0

1 0 0 1 0 0 1 0