# Signed and unsigned numbers

- Binary digits or bits
  - 0 or 1
  - The basic component of information in computer systems.

- Binary numbers are represented as a sequence of binary bits.
  - A number of base 2

- Example

$$1011_{two}$$

$$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)_{ten}$$

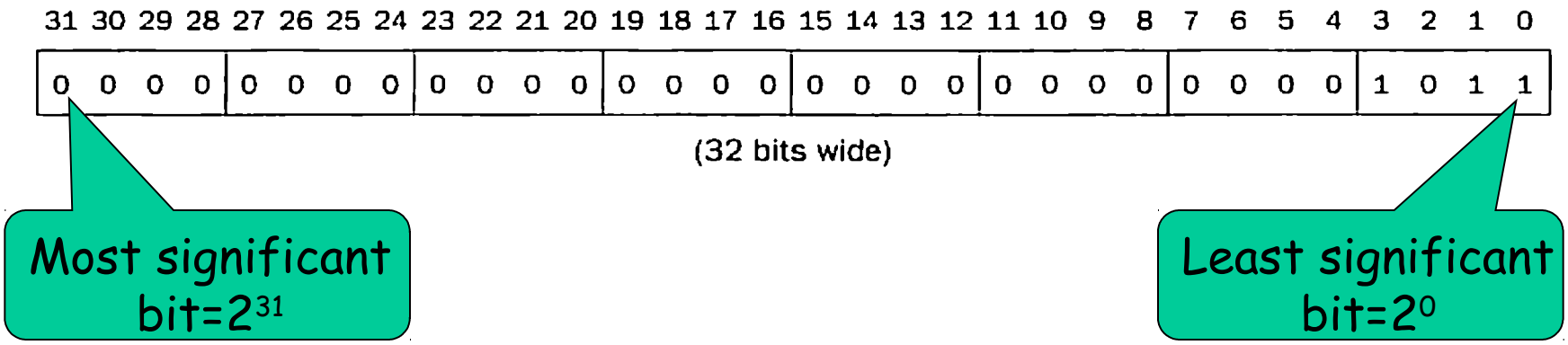
$$= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)_{ten}$$

$$= 8 + 0 + 2 + 1_{ten}$$

$$= 11_{ten}$$

# Representation of binary numbers

- An integer is usually represented as a MIPS word



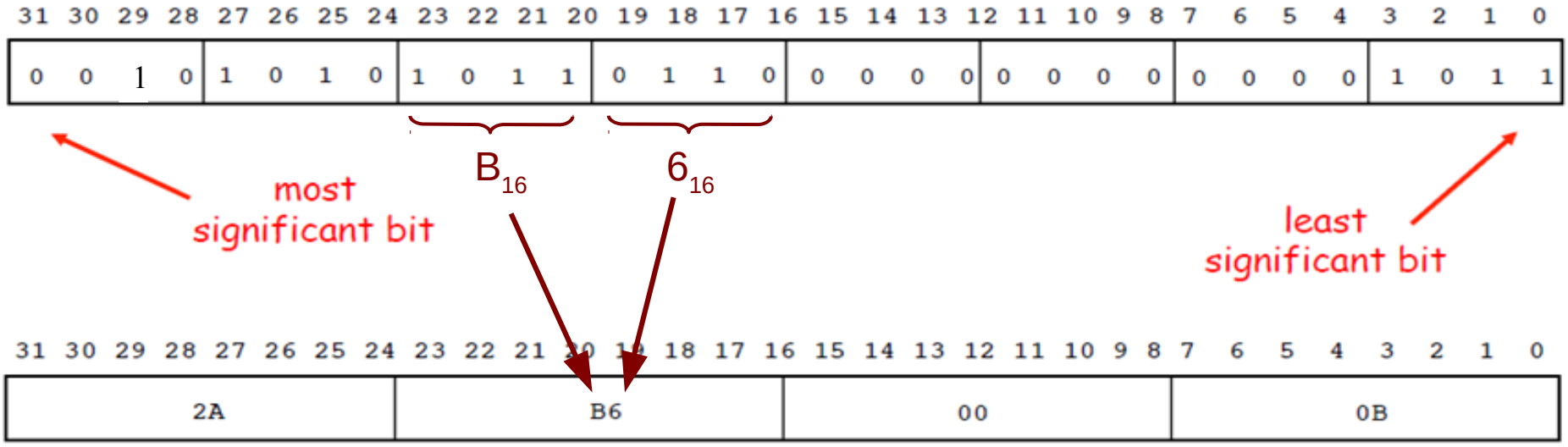

- Question: how many different integers can be represented by a word?

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$
$$\ldots \qquad\qquad \ldots$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$$

# Hexadecimal representation


- Sixteen different digits
  - 0,1,2,3,4,5,6,7,8,9,A(10),B(11),C(12),D(13),E(14),F(15)

- Each hexadecimal digit corresponds to four consecutive binary digits.



# Hexadecimal conversion table

| Hexadecimal | Binary | Hexadecimal | Binary | Hexadecimal | Binary | Hexadecimal | Binary |
|---|---|---|---|---|---|---|---|
| $0_{hex}$ | $0000_{two}$ | $4_{hex}$ | $0100_{two}$ | $8_{hex}$ | $1000_{two}$ | $c_{hex}$ | $1100_{two}$ |
| $1_{hex}$ | $0001_{two}$ | $5_{hex}$ | $0101_{two}$ | $9_{hex}$ | $1001_{two}$ | $d_{hex}$ | $1101_{two}$ |
| $2_{hex}$ | $0010_{two}$ | $6_{hex}$ | $0110_{two}$ | $a_{hex}$ | $1010_{two}$ | $e_{hex}$ | $1110_{two}$ |
| $3_{hex}$ | $0011_{two}$ | $7_{hex}$ | $0111_{two}$ | $b_{hex}$ | $1011_{two}$ | $f_{hex}$ | $1111_{two}$ |

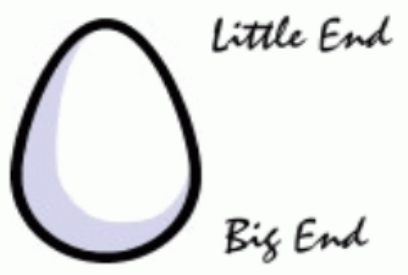
- Which hexadecimal digit has been converted wrongly?

1101 0011 1000 0111 1010 0101 0010 1011

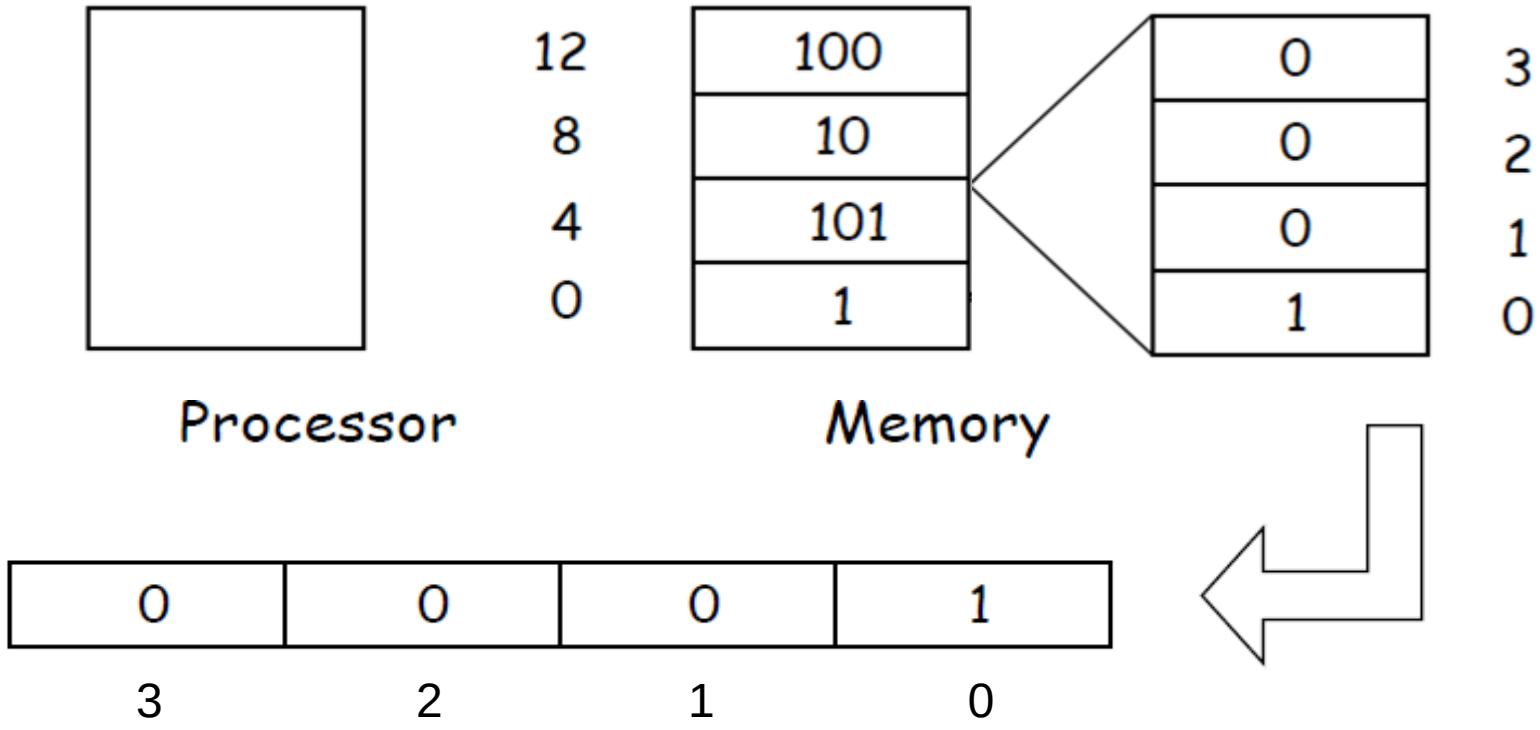
D 3 8 7 9 5 2 B

- A. 3 -> 4
- B. 7-> 8
- C. 9 -> A
- D. 5 -> 4

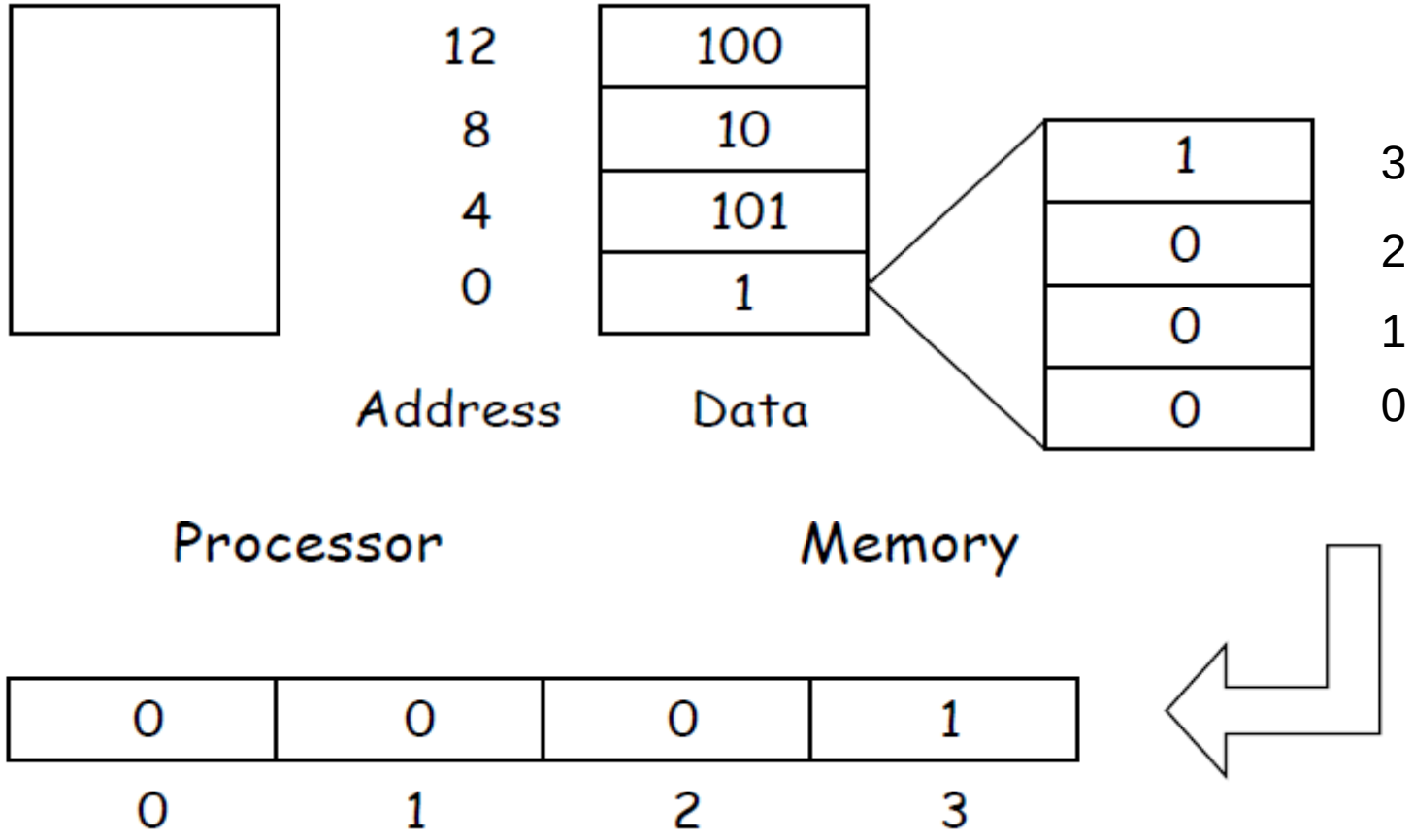
# Big-endian or little-endian?



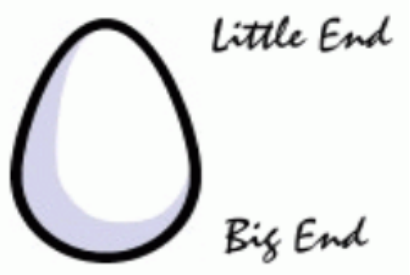

- Big-endian
  - The "big end" byte as the word address
- Little-endian
  - The "little end" byte as the word address



# Byte order on big-endian machine



# Big-endian or little-endian?



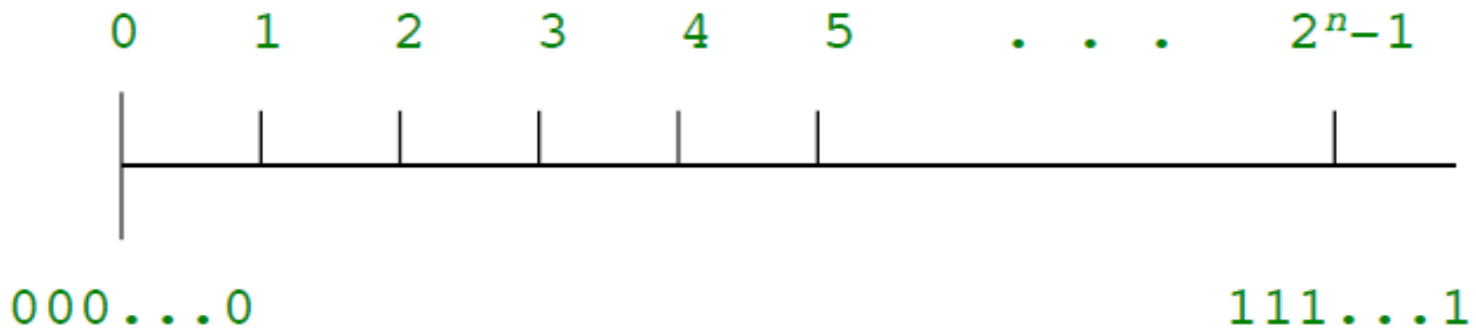

- Big-endian
  - The "big end" byte as the word address
- Little-endian
  - The "little end" byte as the word address

No one answer.

- For MIPS, all data is stored in little-endian byte order (each word consists of byte 3 followed by byte 2 then 1 then 0).
- For example, each word can hold 4 characters of a string and those 4 characters will appear in the reverse order from that of the string literal.
- When sending data between computers, this can mess things up… **Why?**

# Signed magnitude representation

- Using n bits, we could represent the natural numbers up to $2^n-1$.




- Or we could reserve for one bit, usually the msb, to represent a number's sign.

# Some problems with signed magnitude

- Ambiguous representation of zero
  - Positive zero 00…00
  - Negative zero 10…00
- How to add positive numbers to negative numbers?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | represents | +1101 |
| 1 | 1 | 1 | 0 | 1 | represents | -1101 |

$$\begin{array}{r} 0\ 1\ 1\ 0\ 1 \\ +\ 1\ 1\ 1\ 0\ 1 \\ \hline =0\ 1\ 0\ 1\ 0 \end{array}$$

# Two's complement representation

- Representation range

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
. . . . . .

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2,147,483,645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2,147,483,646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2,147,483,647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2,147,483,648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2,147,483,647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2,147,483,646_{ten}$
. . . . . .

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

# Converting two's complement to decimal

- Conversion formula

$$(x31 \times -2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^{1}) + (x0 \times 2^{0})$$

- Calculate the decimal value of the 32-bit number below:

$$1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1111\ \ 1100_{two}$$

  - Conversion using the formula

$$\begin{aligned}
&(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{2}) + (0 \times 2^{1}) + (0 \times 2^{0}) \\
&= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0 \\
&= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten} \\
&= -4_{ten}
\end{aligned}$$

# A useful shortcut

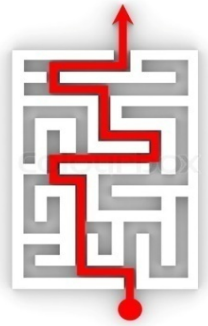
- Quick way to negate a two's complement binary number
  - Invert every 0 to 1 and every 1 to 0
  - Then add one to the result

- Example

$$2_{ten} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$$

  - Find two's complement representation of -2:

$$\begin{array}{rl} & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} \\ + & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1_{two} \\ \hline = & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} \\ = & -2_{ten} \end{array}$$



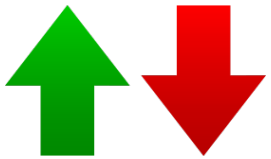
# Going the other direction

- Negate two's complement representation below, which is

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000_{two}$$

- What is the corresponding decimal number
  - A. 10
  - B. 4
  - C. 16
  - D. 8

$$\begin{array}{r} 0000,0000,0000,0000,0000,0000,0000,0111 \\ +\ 0000,0000,0000,0000,0000,0000,0000,0001 \\ \hline 0000,0000,0000,0000,0000,0000,0000,1000 \end{array}$$



# Sign extension

- Convert a binary number represented in n bits to a number represented with more than n bits.

- How to extend?
  - Take the most the significant bit from the smaller quantity and replicate it to fill the new bits of the larger quantity.

- Example:
  - 16-bit binary

$$0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

  - 32-bit binary

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

# Negate and extension

- 16-bit negative number

$$\begin{array}{r} 0000\ 0000\ 0000\ 0010_{two} \\ 1111\ 1111\ 1111\ 1101_{two} \\ +\qquad\qquad\qquad\qquad 1_{two} \\ \hline = \ 1111\ 1111\ 1111\ 1110_{two} \end{array}$$

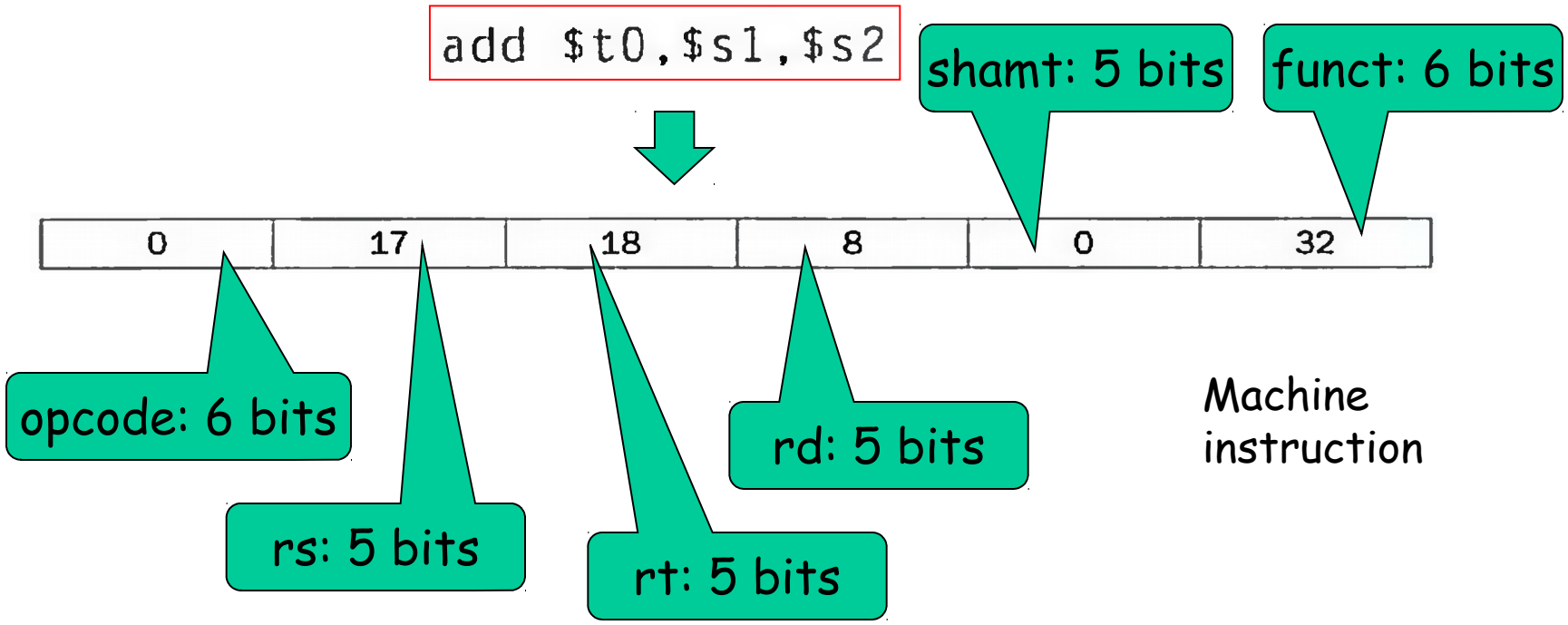
- 32-bit negative number

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$$

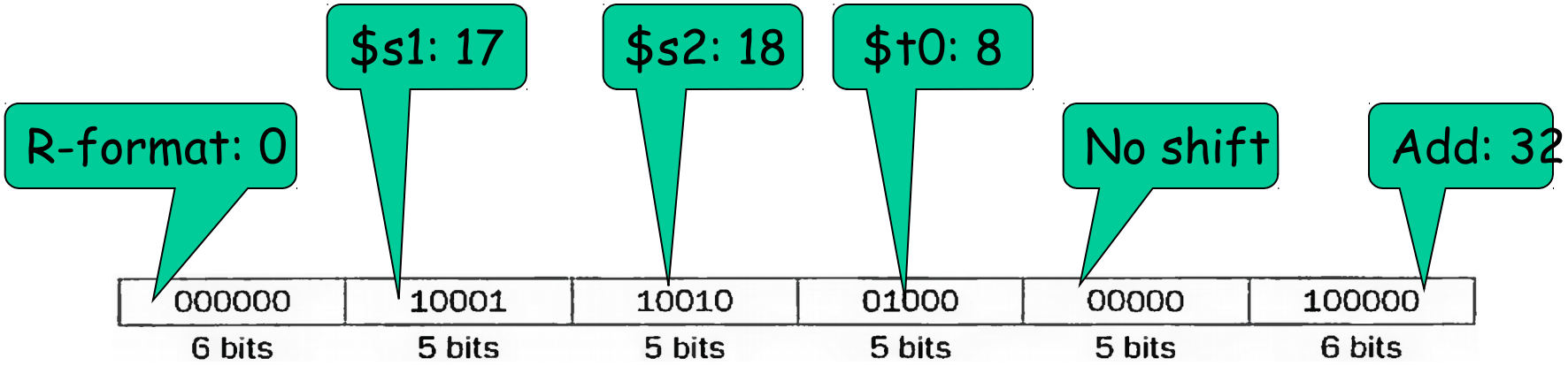
# Representing instructions in computers

## R-Format

- Translate MIPS assembly instruction into a machine instruction



# Meaning of the instruction




- op: basic operation of the instruction (operation code)
- rs: the first register source operand
- rt: the second register operand
- rd: the register destination operand
- shamt: shift amount, only used in shifting operations
- funct: function code, select the specific variant of the operation in the op field.

# Important concept

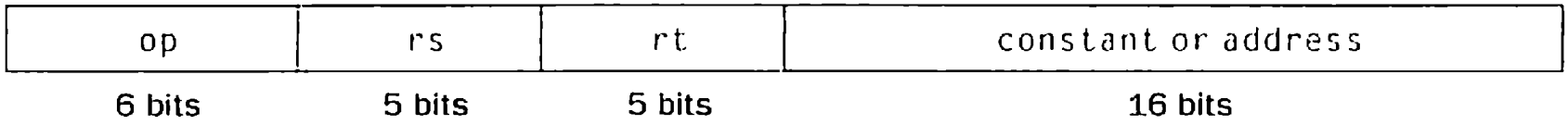
- Machine language
  - Binary representation used for communication within a computer system
- Instruction format
  - A form of representation of an instruction composed of fields of binary numbers
- In MIPS, all instructions have the same length.

- What is the maximum offset achievable with 5 bits?
  - A. from -16 to 15
  - B. from -32 to 31
  - C. from -8 to 7
  - D. from 0 to 31

Design principle 4: Good design demands good compromises.

# I-format

- I-format is used by immediate and data transfer instructions.




  - The base register is rs and the destination register is rt
  - 16-bit address means a load instruction can load any word within a region of $-2^{15}$ (-32768) to $2^{15}-1$ (+32767) bytes of the address

- Example:

| | | | |
|---|---|---|---|
| op=100011<br>(35, lw) | rs=10011<br>(19, $s3) | rt=01000<br>(8, $t0) | 100000<br>(32) |

lw $t0, 32($s3)

# Summary of MIPS instruction encoding

- MIPS instruction encoding discussed so far.

| Instruction | Format | op | rs | rt | rd | shamt | funct | address |
|---|---|---|---|---|---|---|---|---|
| add | R | 0 | reg | reg | reg | 0 | $32_{ten}$ | n.a. |
| sub (subtract) | R | 0 | reg | reg | reg | 0 | $34_{ten}$ | n.a. |
| add immediate | I | $8_{ten}$ | reg | reg | n.a. | n.a. | n.a. | constant |
| lw (load word) | I | $35_{ten}$ | reg | reg | n.a. | n.a. | n.a. | address |
| sw (store word) | I | $43_{ten}$ | reg | reg | n.a. | n.a. | n.a. | address |

# Example

- Translate MIPS assembly language into machine language
  - High-level language
    ```
    A[300] = h + A[300];
    ```
  - Assembly language
    ```
    lw   $t0,1200($t1)# Temporary reg $t0 gets A[300]
    add  $t0,$s2,$t0  # Temporary reg $t0 gets h + A[300]
    sw   $t0,1200($t1)# Stores h + A[300] back into A[300]
    ```
- How are registers associated with variables in the high level language?
  - Base of array - $t1
  - Variable h - $s2

# Answer

- Decimal representation of machine instructions

| op | rs | rt | rd | address/ shamt | funct |
|---|---|---|---|---|---|
| 35 | 9 | 8 | 1200 | | |
| 0 | 18 | 8 | 8 | 0 | 32 |
| 43 | 9 | 8 | 1200 | | |

- Binary representation of machine instructions

| | | | | | |
|---|---|---|---|---|---|
| 100011 | 01001 | 01000 | 0000 0100 1011 0000 | | |
| 000000 | 10010 | 01000 | 01000 | 00000 | 100000 |
| 101011 | 01001 | 01000 | 0000 0100 1011 0000 | | |

```
lw   $t0,1200($t1)# Temporary reg $t0 gets A[300]
add  $t0,$s2,$t0  # Temporary reg $t0 gets h + A[300]
sw   $t0,1200($t1)# Stores h + A[300] back into A[300]
```

# Division algorithm

**Convert 67 to its binary equivalent:**

$67_{10} = x_2$

Step 1: 67 / 2 = 33 R 1 — *Divide 67 by 2. Record quotient in next row*

**Step 2:** 33 / 2 = 16 R 1 — *Again divide by 2; record quotient in next row*

**Step 3:** 16 / 2 = 8 R 0 — *Repeat again*

**Step 4:** 8 / 2 = 4 R 0 — *Repeat again*

**Step 5:** 4 / 2 = 2 R 0 — *Repeat again*

**Step 6:** 2 / 2 = 1 R 0 — *Repeat again*

**Step 7:** 1 / 2 = 0 R 1 — **STOP** *when quotient equals 0*

$1\,0\,0\,0\,0\,1\,1_2$

# The stored-program concept

- Instructions represented as numbers
- Programs are stored in memory to be read or written, just like numbers
- Programs are shipped as files of binary numbers
  - Binary compatibility
  - A small number of instruction set architectures