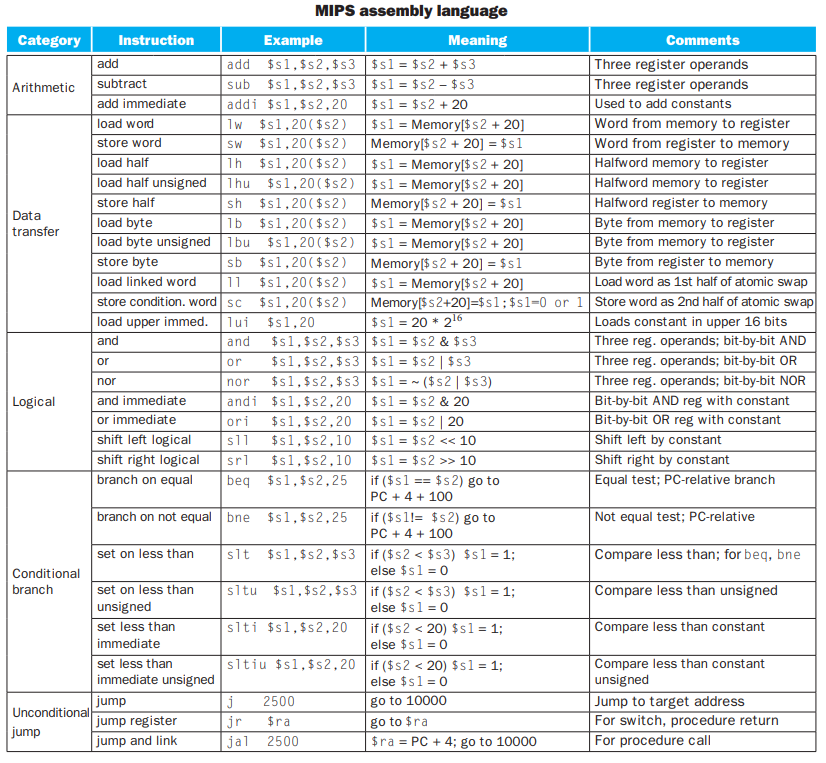
**Instruction Set**: The vocabulary of commands understood by a given architecture.

**Stored-Program Concept**: The idea that instructions and data of many types can be stored in memory as numbers, leading to the stored-program computer.



**Registers**: The operands of arithmetic instructions are restricted; they must be from a limited number of special locations built directly in hardware called registers. Registers are primitives used in hardware design that are also visible to the programmer when the computer is completed.

**Word**: The natural unit of access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.

**Data Transfer Instruction**: A command that moves data between memory and registers.

**Address**: A value used to delineate the location of a specific data element within a memory array.

**Load**: The data transfer instruction that copies data from memory to a register is traditionally called load. The actual MIPS name for this instruction is lw, standing for load word.

**Alignment Restriction**: A requirement that data be aligned in memory on natural boundaries.

Computers divide into those that use the address of the left most or “big end” byte as the word address versus those that use the rightmost or “little end” byte. MIPS is in the big-endian camp.

**Store**: The instruction complementary to load is traditionally called store; it copies data from a register to memory. The actual MIPS name is sw, standing for store word.

**Spilling Registers**: The process of putting less commonly used variables (or those needed later) into memory.

Registers take less time to access and have higher throughput than memory, making data in registers both faster to access and simpler to use. Accessing registers also uses less energy than accessing memory.

**Binary Digit**: Also called binary bit. One of the two numbers in base 2, 0 or 1, that are the components of information.

**Least Significant Bit**: The rightmost bit in a MIPS word.

**Most Significant Bit**: The left most bit in a MIPS word.

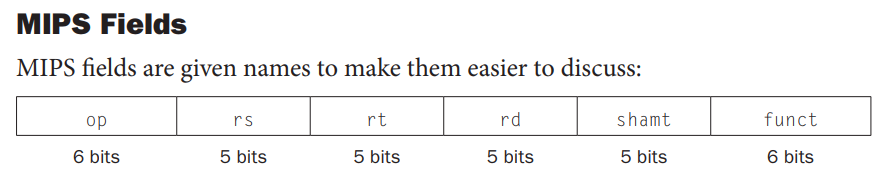
**Two’s Complement**: Two’s complement gets its name from the rule that the unsigned sum of an n-bit number and its n-bit negative is 2n; hence, the negation or complement of a number x is 2n - x, or its “two’s complement.”

**One’s Complement**: A notation that represents the most negative value by 10 . . . 000two and the most positive value by 01 . . . 11two, leaving an equal number of negatives and positives but ending up with two zeros, one positive (00 . . . 00two) and one negative (11 . . . 11two). Th e term is also used to mean the inversion of every bit in a pattern: 0 to 1 and 1 to 0.

**Biased Notation**: A notation that represents the most negative value by 00 . . . 000two and the most positive value by 11 . . . 11two, with 0 typically having the value 10 . . . 00two, thereby biasing the number such that the number plus the bias has a non-negative representation.

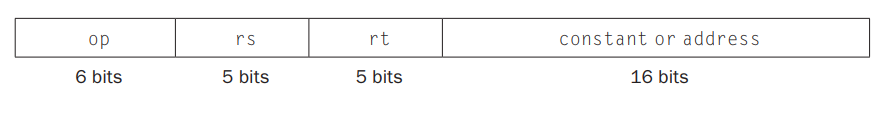
**Instruction Format**: A form of representation of an instruction composed of fields of binary numbers.

**Machine language**: Binary representation used for communication within a computer system.



(These are for R-type, that is, Register-Type Instructions)

1. **Op**: Basic operation of the instruction, traditionally called the opcode. It is the field that denotes the operation and format of an instruction.
2. **Rs**: The first register source operand.
3. **Rt**: The second register source operand.
4. **Rd**: The register destination operand. It gets the result of the operation.
5. **Shamt**: Shift amount.
6. **Funct**: This field, often called the function code, selects the specific variant of the operation in the op field.



(These are for I-type, that is immediate-type instructions)

In a load word instruction, the **rt** field specifies the destination register, which receives the result of the load.

**AND**: A logical bit-by-bit operation with two operands that calculates a 1 only if there is a 1 in both operands.

**OR**: A logical bit-by-bit operation with two operands that calculates a 1 if there is a 1 in either operand.

**NOT**: A logical bit-by-bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.

**NOR**: A logical bit-by-bit operation with two operands that calculates the NOT of the OR of the two operands. That is, it calculates a 1 only if there is a 0 in both operands.

**Conditional Branch**: An instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.

**Basic Block**: A sequence of instructions without branches (except possibly at the end) and without branch targets or branch labels (except possibly at the beginning).

**Jump address table**: Also called jump table. A table of addresses of alternative instruction sequences.

**PC-relative addressing**: An addressing regime in which the address is the sum of the program counter (PC) and a constant in the instruction.

