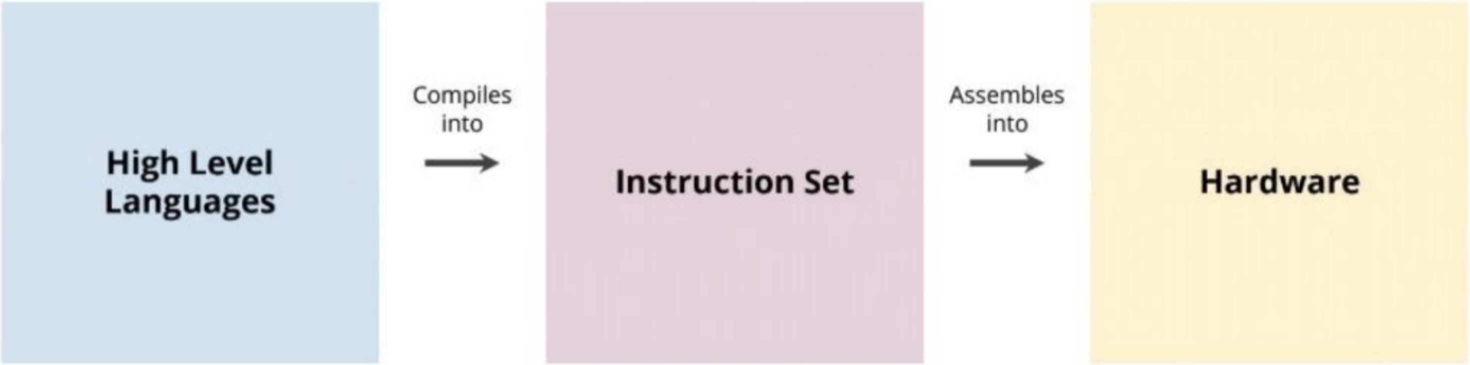
Instruction Set

Purpose: A middleman between the software and the hardware

- Instruction set can be converted to machine code through an assembler
- Different sets of hardware require different sets of assembly instructions



Types of Instructions

- Expressions
  - ADD, NAND
- Assignments
- Conditionals and Loops
  - BEQ
- Functions
  - Jalr

Addressing Modes

- How operands are specified
- Addressability: smallest space in memory that can be addressed
  - Our ISA will be word-adddressable
  - Other types may be byte addressable
- Three types
  - Register
  - Base + Offset
  - PC Relative

R-type Instructions

- Operands that we manipulate are within the register file itself
- Examples: add, nand

I-type Instructions

- Contains two operands for registers and one for immediate value
- Examples: addi, lw, sw
  - Add immediate, load word, store word
  - addi $v0, $a0, 25
    - Translates to `x = y + 25` where `$v0` is register x and `$a0` is register y

J-type Instructions

- Two registers, the rest is unused
- Examples: jalr, beq

O-type Instructions

- Opcode only, doesn't specify any operands
- Example: halt
  - halt
  - opcode 0b111

Data Types

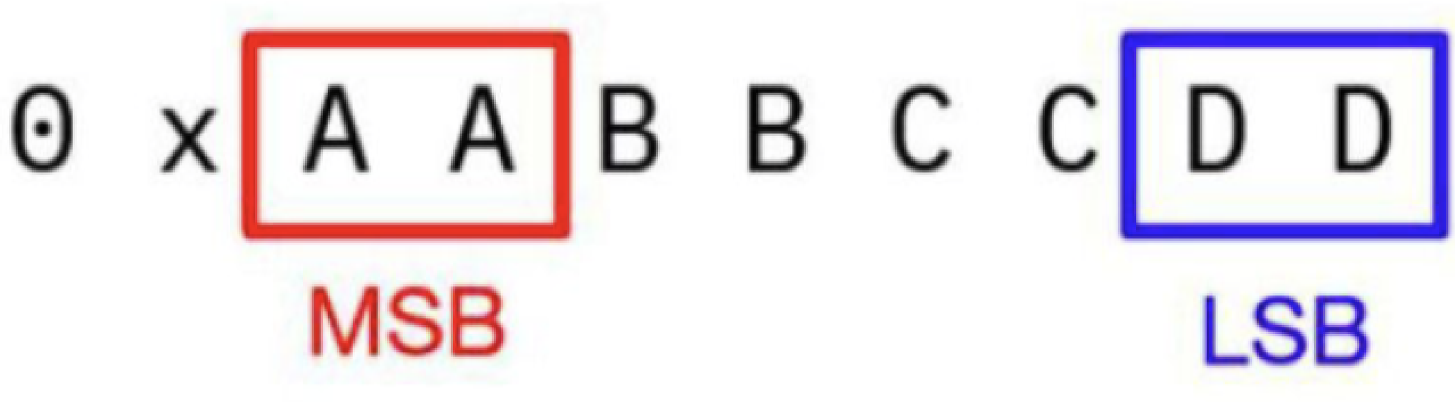
There is variation in what kinds of data we can load, store, and manipulate

Word size = max. precision supported in an architecture

- For our datapath, this is 4 bytes = 32 bits

Endianness

Endianness deals with how this data is placed at a specific location in memory



- Quick recap:
  - Rightmost two hex values = least significant byte
  - Leftmost two hex values = most significant byte

Big Endian

- Most significant bit is stored first
- Example: `0xABCDEFGH`

0x100	0x101	0x102	0x103
0xAB	0xCD	0xEF	0xGH

Little Endian

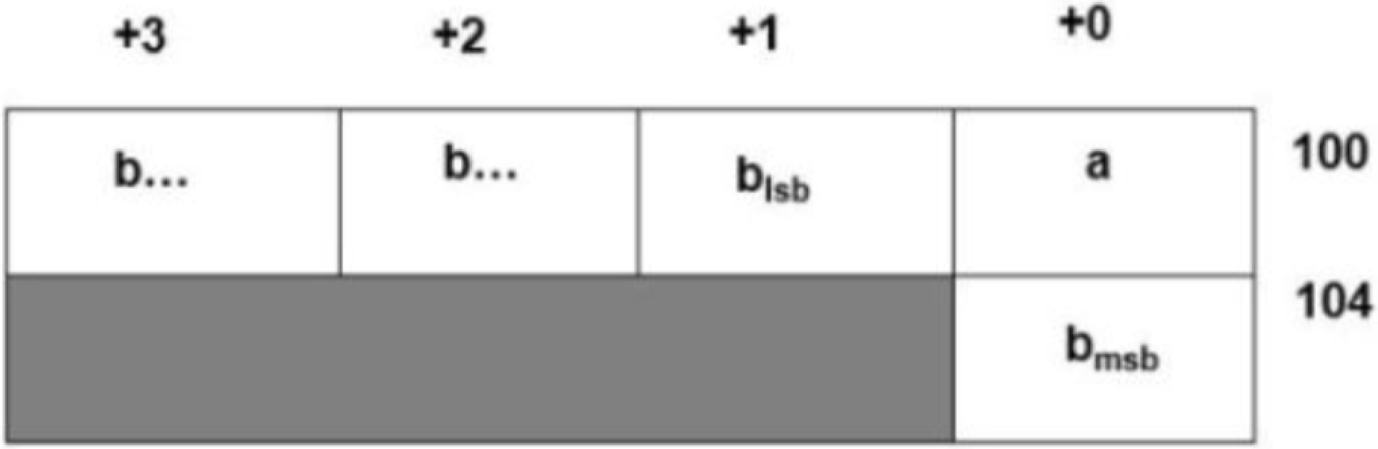
- Least significant bit is stored first
- Example: `0xABCDEFGH`

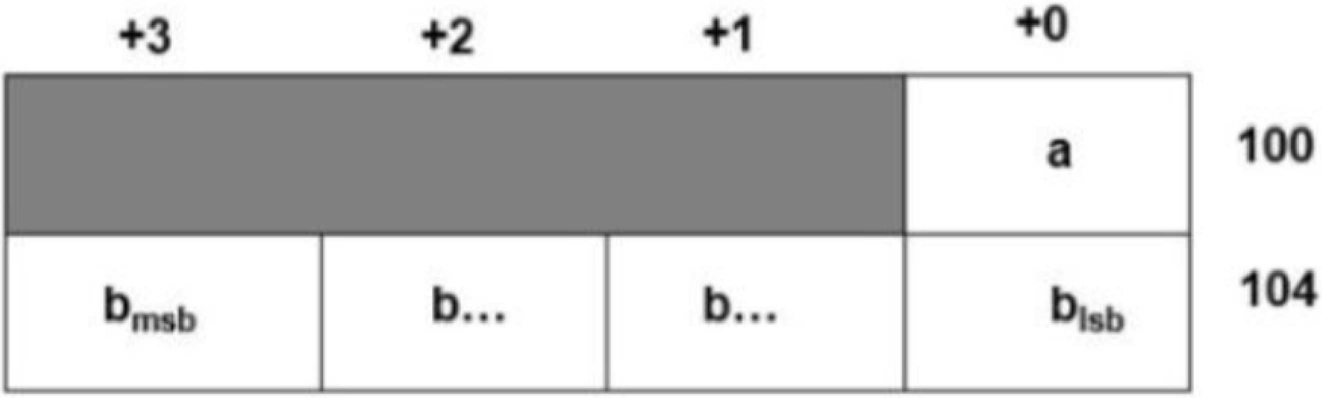
0x100	0x101	0x102	0x103
0xGH	0xER	0xCD	0xAB

Endianness only affects the order of bytes within a single value, not the value itself

Packing

- Data alignment matters
- Want the least amount of memory access (memory slow)





- Bottom one is Little Endian

Unaligned memory access is very expensive

Example:

```
struct {
    int a;
    char b;
    short d;
    short e;
}
```

+3	+2	+1	+0
a4	a3	a2	a1
d2	d1		b
		e2	e1

Note that we leave the space between b and d, so it becomes clearer to the compiler what is what