Machine language

"In Paris they just simply opened their eyes and stared when we spoke to them in French!
We never did succeed in making those idiots understand their own language."
***Innocents Abroad*, Mark Twain**

Assembly language

High-level language

- **a = b + c;**

Machine language

- **000000 01000 01001 01010 00000 100001**

Assembly language is between high-level and machine

- **Each statement defines one machine operation**
- **Directly represents architecture**
- **Assembler program translates to machine language**

ISA: Instruction Set Architecture

- **Machine structure as seen by the programmer**
- **Each kind of machine has its own ISA**
  - **Sun (Labs): SPARC**
  - **DEC (Class cluster): Alpha**
  - **HP: PA (Precision Architecture)**
  - **IBM Classic: S360/370/390/zSeries**
  - **PC: Intel x86**
  - **MAC: Motorola 680x0**

ISA: Types

Types of architectures

- CISC: complex instruction set computer
    - Traditional computer architecture
    - Unique instructions for as many operations as possible
    - Advantages
        - Each instruction can do more work
        - Programs use less memory
        - Easier to program directly or to write compilers
    - Disadvantages
        - More complex hardware circuits
        - More expensive to develop and build
        - Usually slower
- RISC: reduced instruction set computer
    - Developed from research in late '70's/early '80's
        - at IBM, Stanford, and UC-Berkeley
    - Look at actual instruction use, focus on most frequent ones
    - Advantages
        - Easier to learn
        - Simpler circuits
        - Cheaper and more reliable to design and build
        - Faster

Disadvantages

- **Larger, more complex programs**
- **Harder to program**
- **Depends on compiler for optimization**

Stored program

Stored program concept

- **Instructions and data are stored in the same memory**
- **Instructions are simply another kind of data**
- **Instructions are executed sequentially unless branch elsewhere or stop**

Fetch-execute cycle

- **Instruction fetch**
  - **Get the next instruction from memory**
- **Decode**
  - **Figure out what operation to perform on which operands**
- **Operand fetch**
  - **Get the operand values**
- **Execute**
  - **Perform the operation**
- **Store result**

Repeat until done

Instructions

Any instruction set must perform a basic set of operations

- May have more complex combinations or special operations as well

Types of operations

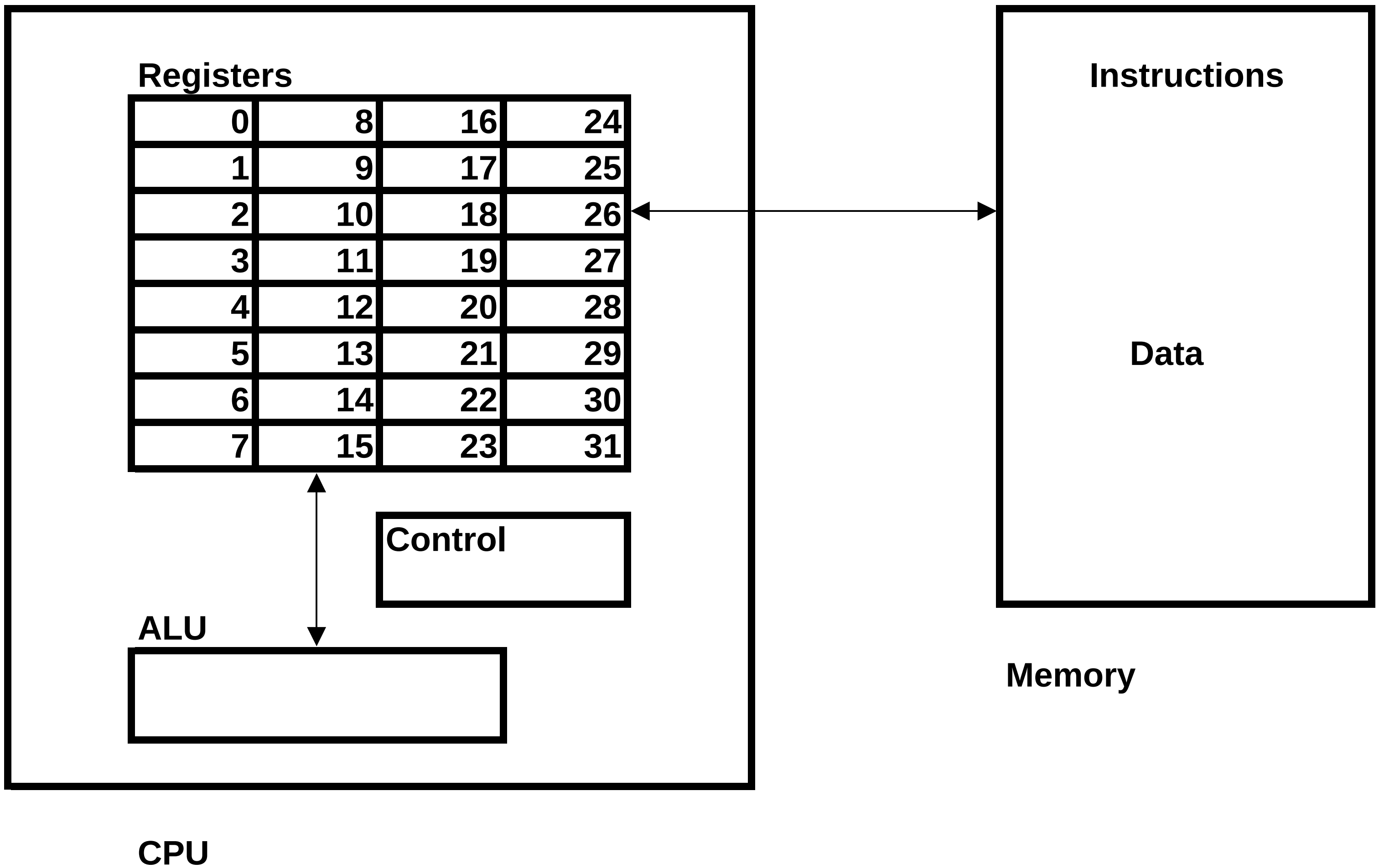
- **Data transfer**: load, store
- **Arithmetic**: add, subtract, multiply, divide
- **Logic**: and, or, xor, complement
- **Compare**: equal, not equal, greater than, less than
- **Branch/jump**: change execution order

MIPS

MIPS

- **"Microcomputer without interlocked pipeline stages"**
- **Name is pun on acronym for "millions of instructions per second"**
- **RISC architecture developed in middle '80's**
- **Extended through several versions**
  - **current: MIPS IV**
- **Used in many "embedded" applications**
  - **Game machines: Sony, Nintendo**
  - **TV set top boxes: LSI Logic shipped 7 million in 2001**
  - **Routers: Cisco**
  - **Laser printers**
  - **PDAs**
- **High-performance workstations: Silicon Graphics (Lord of the Rings, other films)**
- **"Over 100 million sold"**

MIPS: machine model



MIPS: machine model

Main memory

- data: 32-bit address: range from 0x00000000 to 0xFFFFFFFF
- upper half of range reserved (see fig. 3-22)

Processor

- **registers**: store data to perform operations
  - faster than main memory
  - **load-store** architecture
    - access memory only through load, store instructions
      - **load**: register <- - - data from memory
      - **store**: register - - -> data to memory
      - amount of data in bytes (1, 2, 4, 8) depends on instruction
    - all other operations use only registers or immediate values
      - immediate: contained in instruction
    - CISC: may use register and memory to perform operation
  - **32-bit** registers
    - **32 general-purpose** registers
      - $r0-$r31
      - Design Principle #2: **"Smaller is faster."**
    - 16 floating point registers
- **ALU**: arithmetic-logic unit
  - performs operations on values in registers
- **control**: determines how operations executed ("computer within computer")

MIPS: instructions

ALU performs arithmetic and logical operations (instructions)
Instruction specifies

1. **The operation to perform.**
2. **The first operand (usually in a register).**
3. **The second operand (usually in a register).**
4. **The register that receives the result.**

MIPS has about 111 different instructions
all 32 bits, 3 different formats

MIPS: instruction example

Example: add unsigned

```
addu    $r10,$r8,$r9  # add 2 numbers
```

Syntax

- **3-operand instructions: all arithmetic/logical operations**
  - **operands separated by commas**
  - **Design principle #1: "Simplicity favors regularity."**
- **operation:** `addu`
  - **one operation per instruction**
  - **one instruction per line**
- **registers**
  - **source:** `$r8, $r9`
  - **target:** `$r10`
- **comment:** `# add 2 numbers`
  - **starts with #, ends with end of line**

Semantics

```
$r10 = $r8 + $r9;
```

alternative notation:

```
R[10] <-- R[8] + R[9]
```

Machine code

hex: `0x01095021`

MIPS: instruction fields

`addu`	`$r10,$r8,$r9  # add 2 numbers`							
hex:	`0x01095021`							
	`0`	`1`	`0`	`9`	`5`	`0`	`2`	`1`
binary:	`0000`	`0001`	`0000`	`1001`	`0101`	`0000`	`0010`	`0001`

fields:	`000000`	`01000`	`01001`	`01010`	`00000`	`100001`
	b_{31-26}	b_{25-21}	b_{20-16}	b_{15-11}	b_{10-6}	b_{5-0}
	opcode	$rs	$rt	$rd	shamt	function

\# bits

- 6 opcode: operation code
- 5 $rs: first source register
- 5 $rt: second source register
- 5 $rd: destination register
- 5 shamt: shift amount
- 6 function: modifies opcode

Why function field?

Notice that the form of the machine instruction is very close to assembler,
but the order of the source and target is reversed

Example of R-type (register) instruction
1 of 3 formats