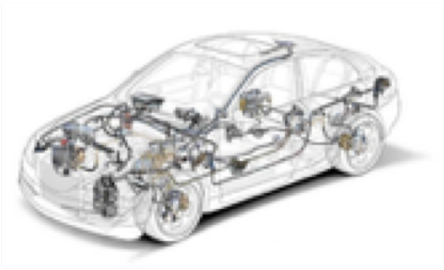
Instruction Set Architecture

Course: Introduction to Processor Architecture

Spring 2020

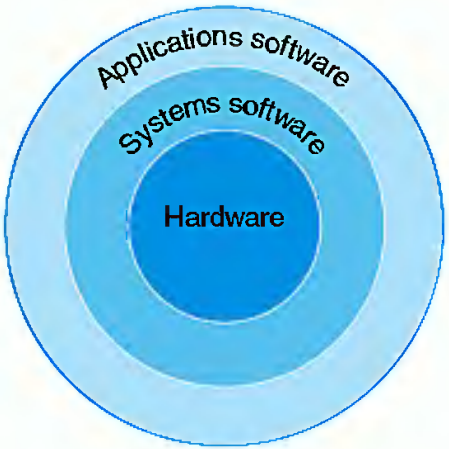
Suresh Purini, IIIT Hyderabad

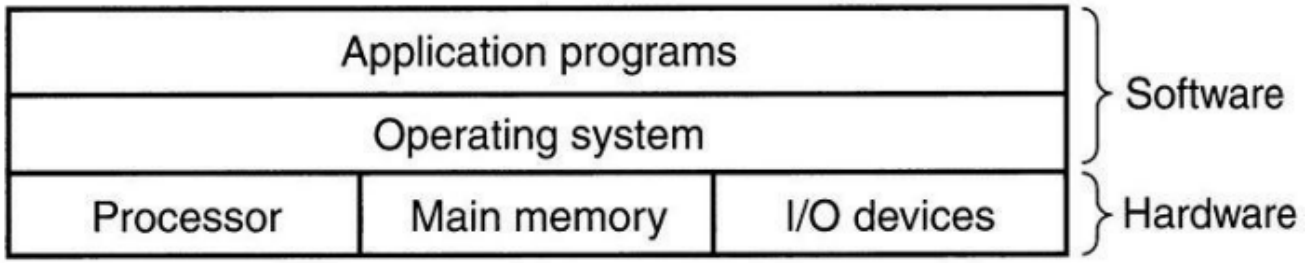
Ubiquitous Computing



Computer System = Hardware + System Software + Application Software

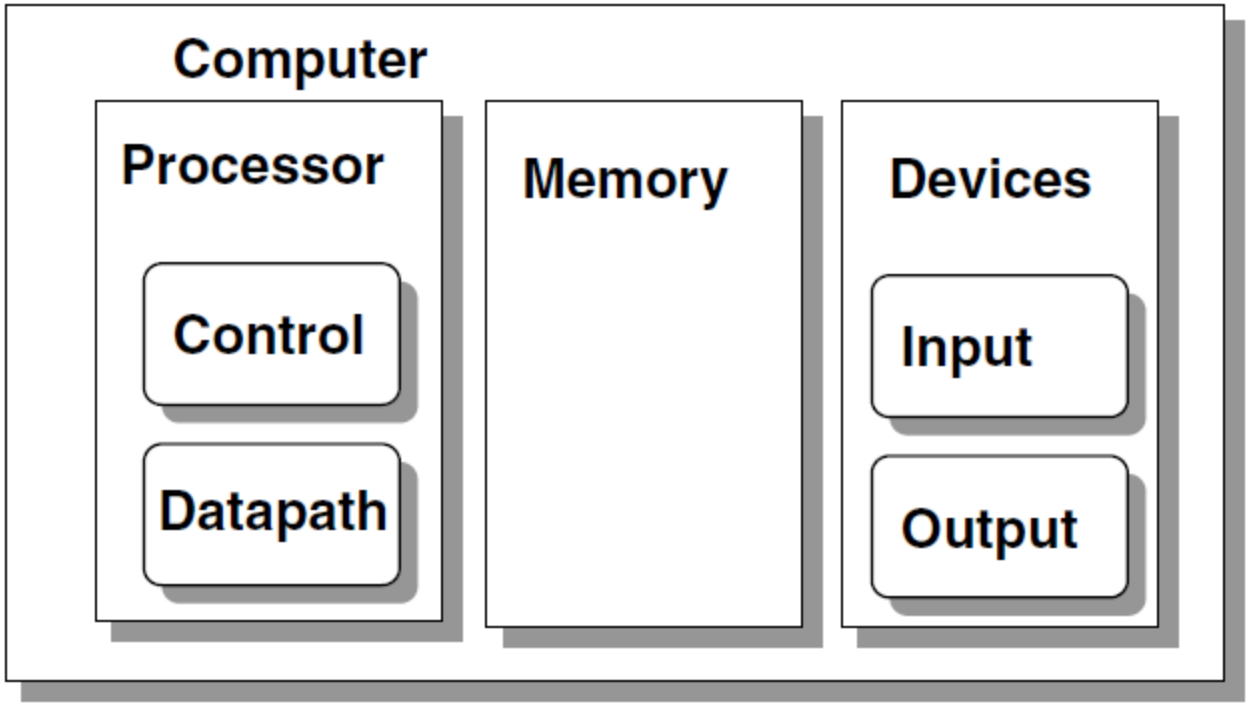
Source: H&P-3 (Hennesy & Patterson, 3rd Edition)





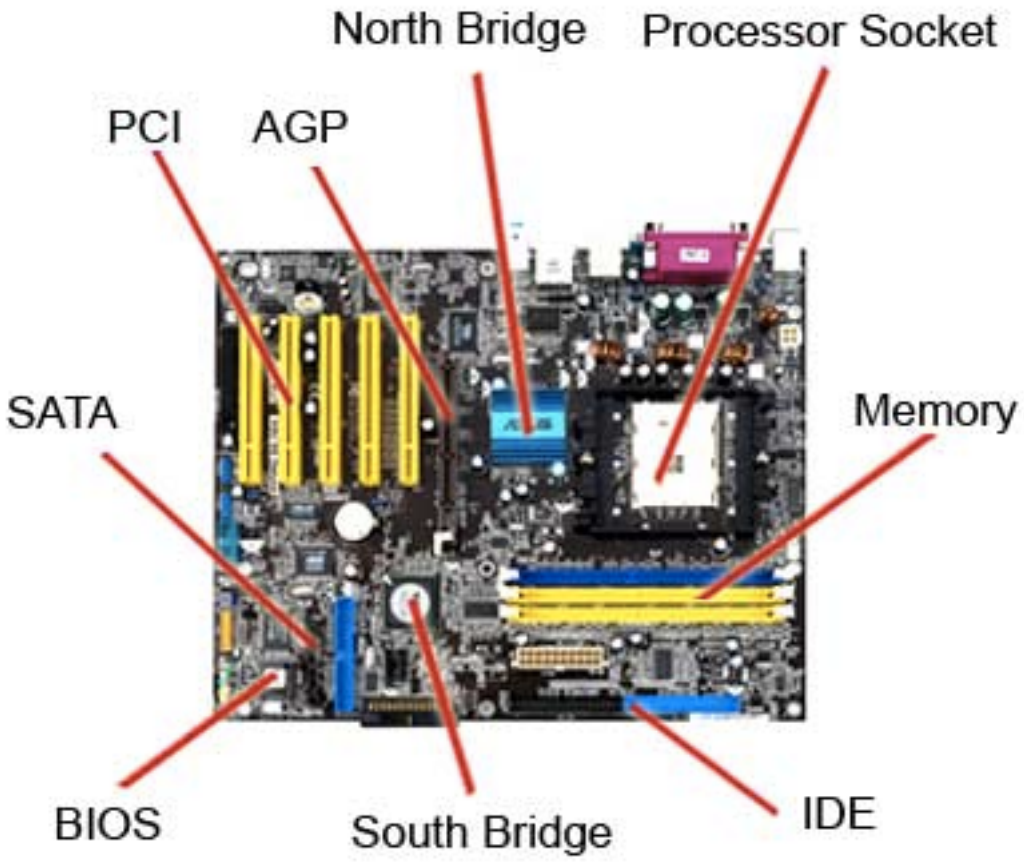
- **System Software:** Operating System, Device Drivers, Loaders, Linkers, Compilers, Assemblers, Editors, ….
- **Application Software:** Web browsers, user-specific applications, …..

Major Function Units



Source: Prof. Cheung's Course Notes (Imperial College, London)

PC Motherboard



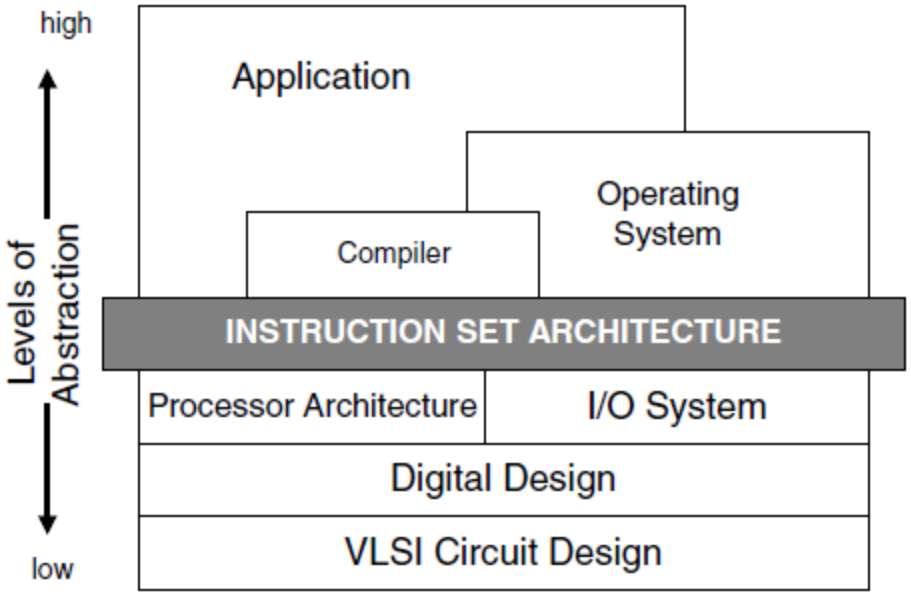
ISA: Hardware – Software Interface

Instruction Set Architecture (ISA)

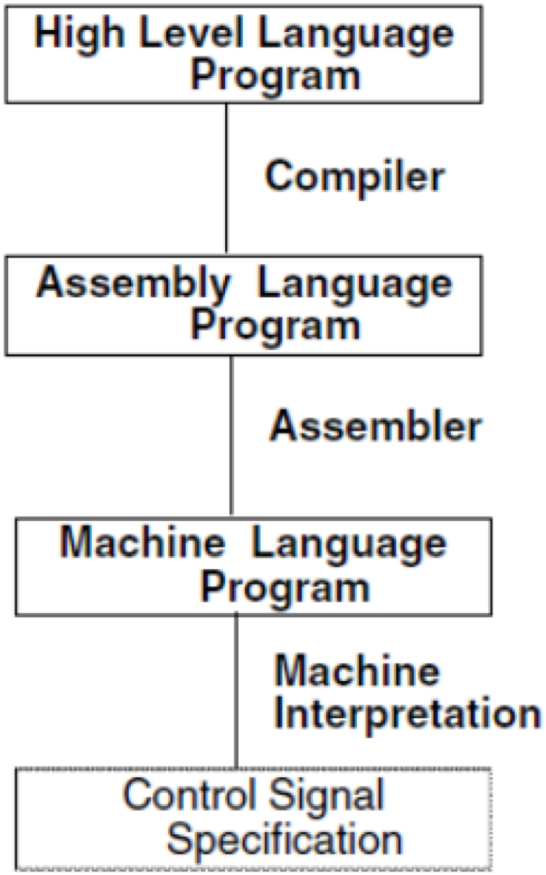
- ISA is an abstraction for the Software to interface with the Hardware.
- Advantage: Multiple implementations for the same ISA.
- AMD Opteron 64 and Intel Pentium 4 are different Implementations of the ISA.

"... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation."

➢Amdahl, Blaaw, and Brooks, 1964



ISA: Hardware – Software Interface



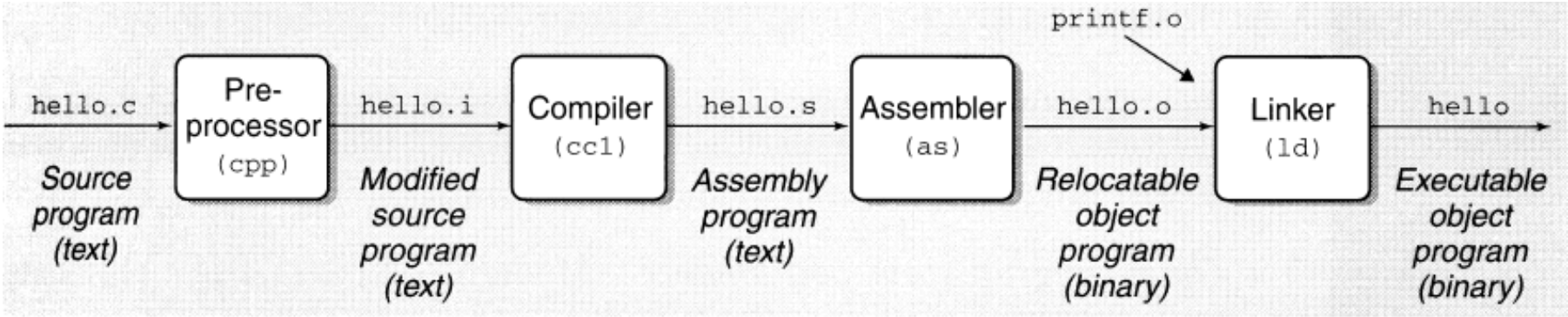

```
temp := v[k];
v[k] := v[k+1];
v[k+1] := temp;
```

```
lw   $15, 0($2)
lw   $16, 4($2)
sw   $16, 0($2)
sw   $15, 4($2)
```

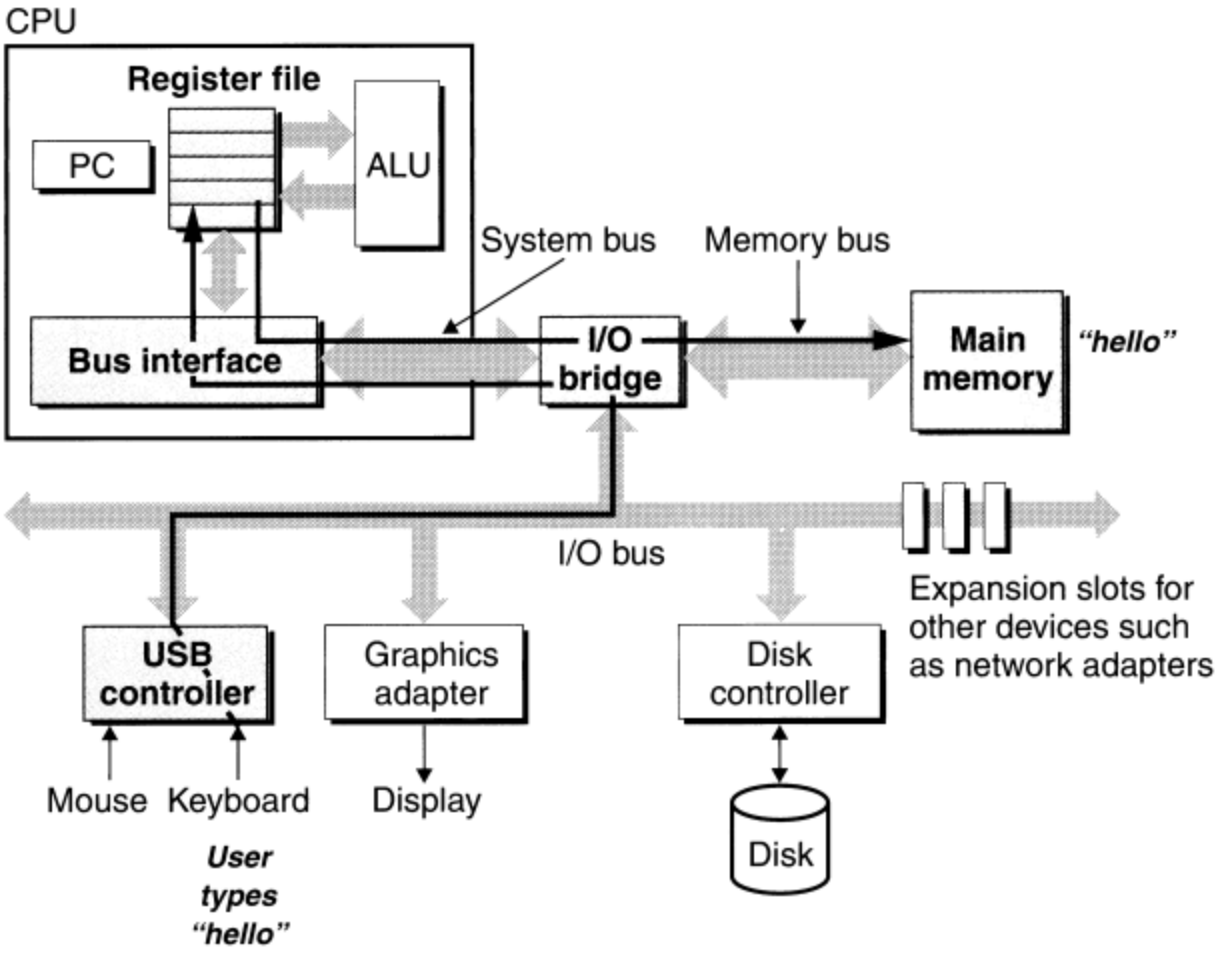
MIPS Assemble Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ISA: Hardware – Software Interface

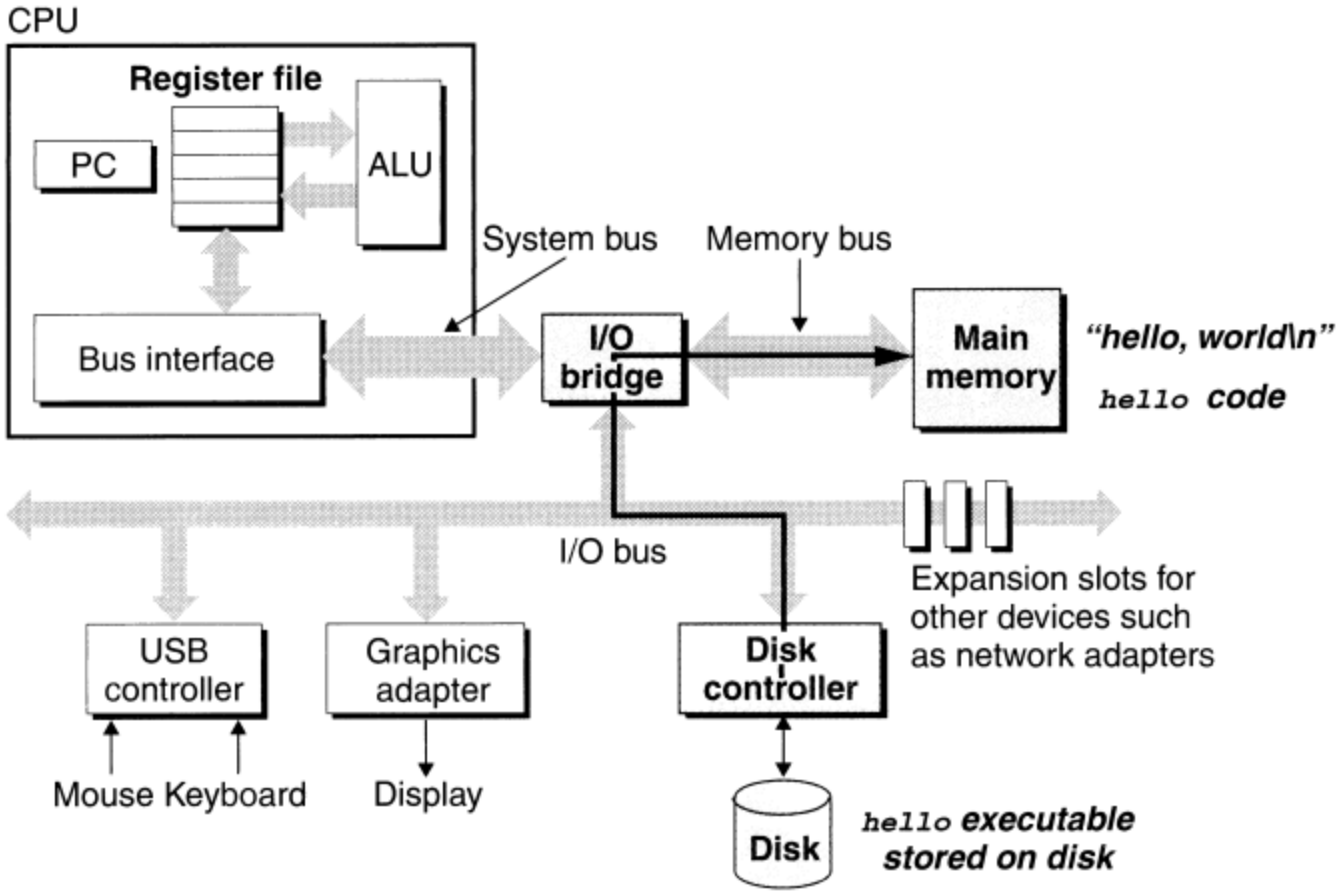


Running the "Hello World" Program



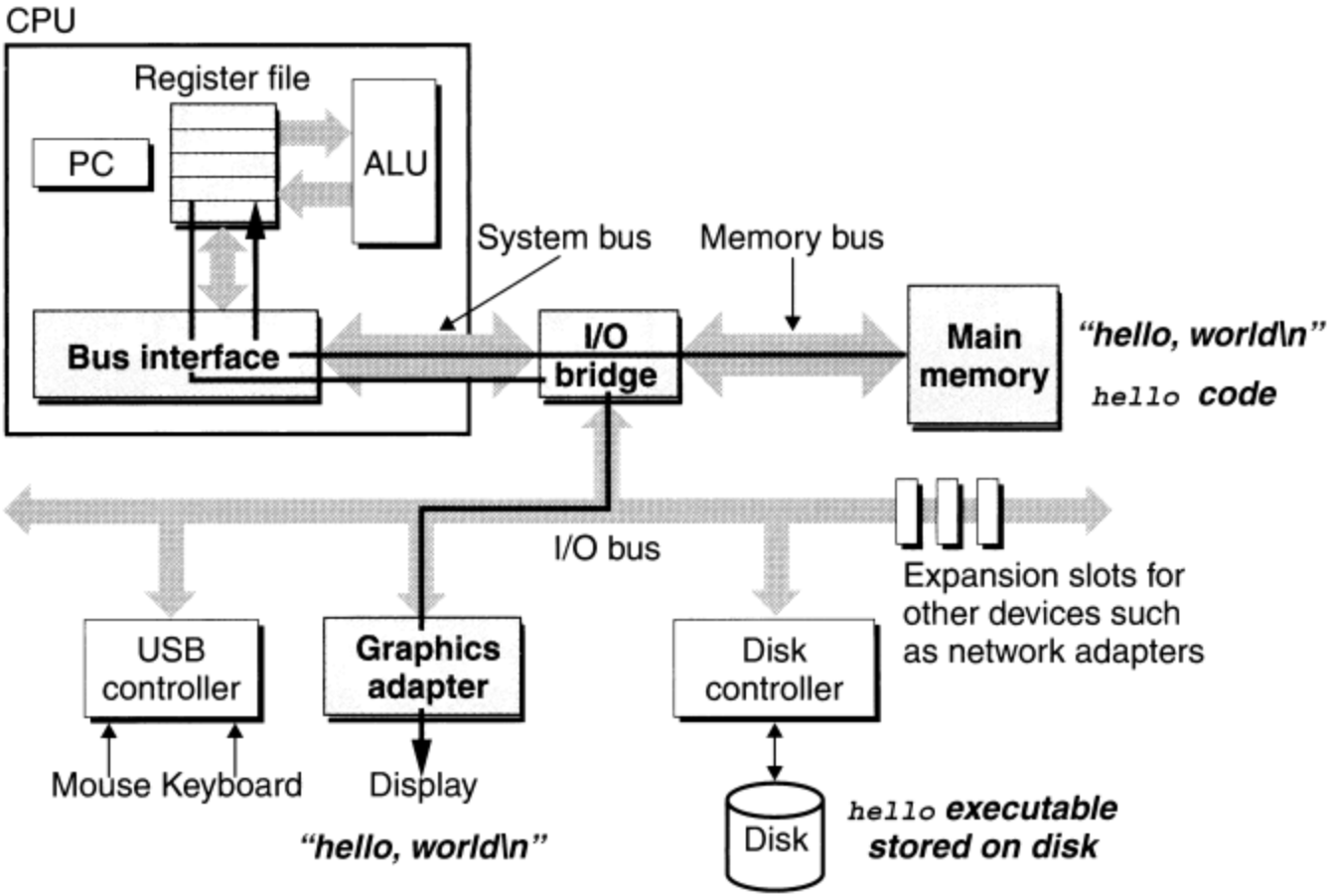
Source:

Running the "Hello World" Program



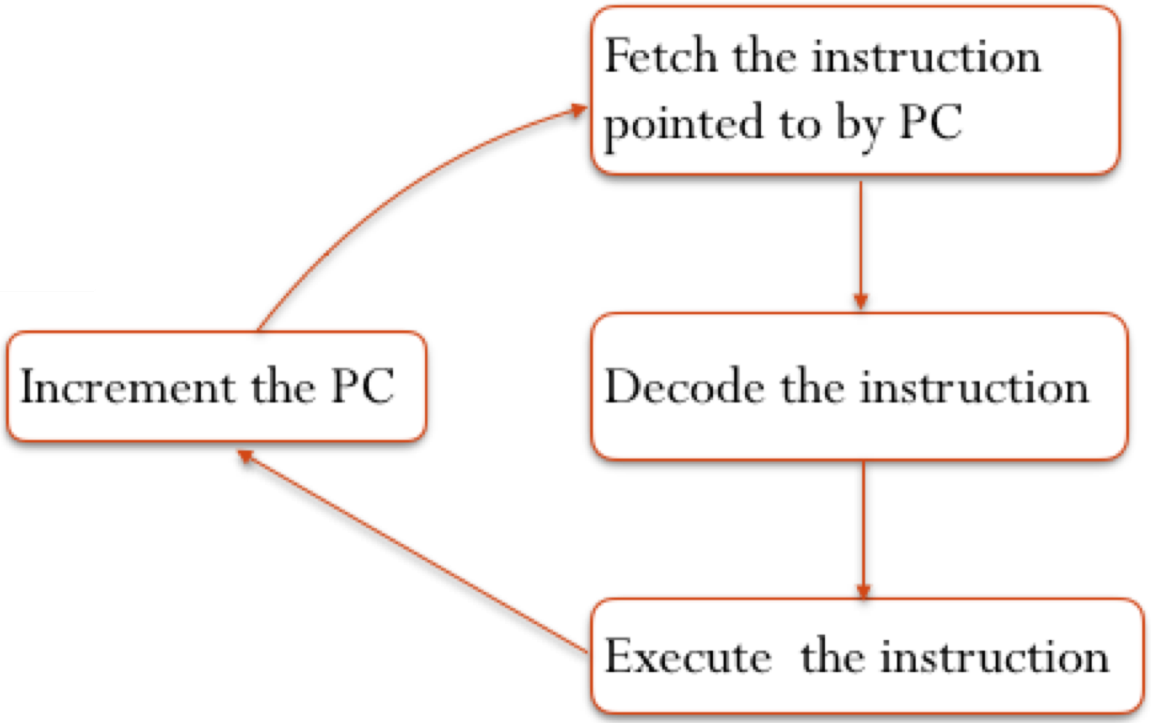
Source:

Running the "Hello World" Program



Source: RB&DO-1

Von Neumann Model of Computing



MIPS ISA

ISA

- Data Transfer Instructions
- Arithmetic Instructions
- Load Store Instructions

ISA has to be Complete. But what does Completeness mean?

C Example

```
1: int sum_pow2(int b, int c)
2: {
3:     int pow2 [8] = {1, 2, 4, 8, 16, 32, 64, 128};
4:     int  a, ret;
5:     a = b + c;
6:     if (a < 8)
7:         ret = pow2[a];
8:     else
9:         ret = 0;
10:    return(ret);
11:}
```

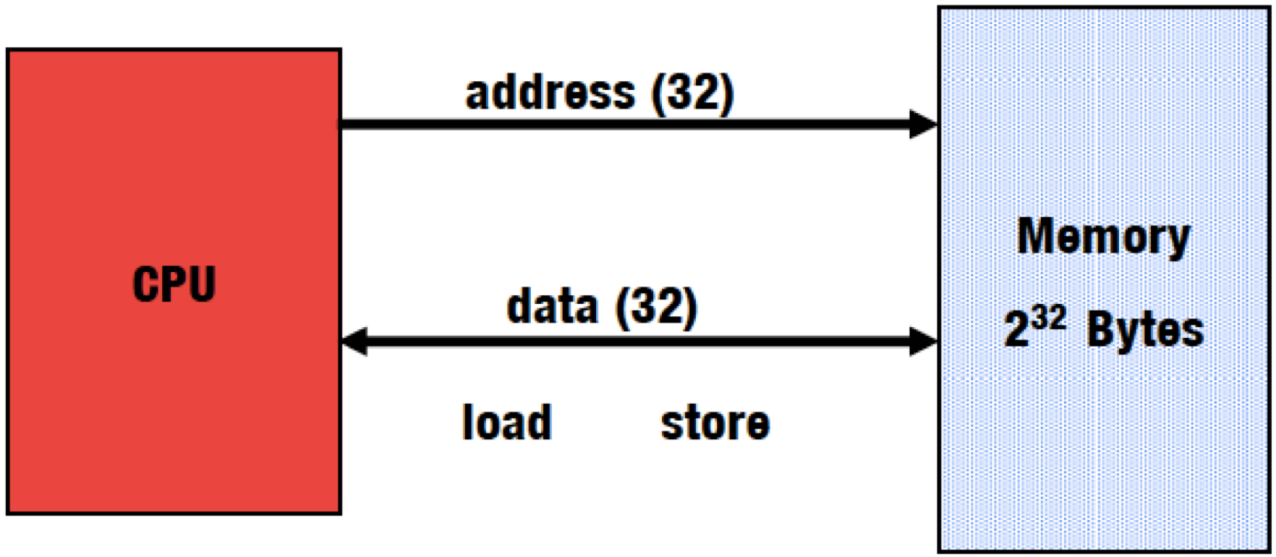
Simple Arithmetic Operations

Instruction	Example	Meaning	Comments
add	add $1,$2,$3	$1 = $2 + $3	3 operands; Overflow
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; Overflow
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; Overflow
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; No overflow
subtract unsign	subu $1,$2,$3	$1 = $2 – $3	3 operands; No overflow
add imm unsign	addiu $1,$2,100	$1 = $2 + 100	+ constant; No overflow

Memory Data Transfer

Data transfer instructions are used to move data to and from memory

A load operation moves data from a memory location to a register and a store operation moves data from a register to a memory location



Data Transfer Instructions: Loads

Data transfer instructions have three parts

- Operator name (transfer size)
- Destination register
- Base register address and constant offset

```
lw dst, offset(base)
```

- Offset value is a signed constant

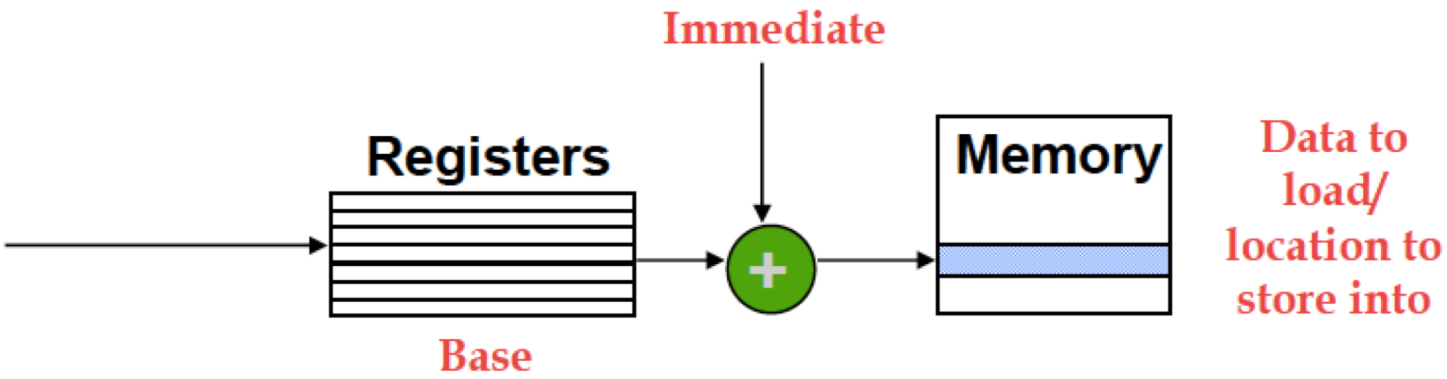
Memory Accesss

All memory access happens through loads and stores

Aligned words, half-words, and bytes

Floating Point loads and stores for accessing FP registers

Displacement based addressing mode



MIPS Integer Load/Store

Instruction	Example	Meaning	Comments
store word	sw $1, 8($2)	Mem[8+$2]=$1	Store word
store half	sh $1, 6($2)	Mem[6+$2]=$1	Stores only lower 16 bits
store byte	sb $1, 5($2)	Mem[5+$2]=$1	Stores only lowest byte
store float	sf $f1, 4($2)	Mem[4+$2]=$f1	Store FP word
load word	lw $1, 8($2)	$1=Mem[8+$2]	Load word
load halfword	lh $1, 6($2)	$1=Mem[6+$2]	Load half; sign extend
load half unsign	lhu $1, 6($2)	$1=Mem[8+$2]	Load half; zero extend
load byte	lb $1, 5($2)	$1=Mem[5+$2]	Load byte; sign extend
load byte unsign	lbu $1, 5($2)	$1=Mem[5+$2]	Load byte; zero extend

Control Flow

The simplest conditional test is the `beq` instruction for equality

```
beq reg1, reg2, label
```

Consider the code

```
        if (a == b) goto L1;
        // Do something
L1:     // Continue
```

Use the `beq` instruction

```
        beq $s0, $s1, L1
        # Do something
L1:     # Continue
```

Control Flow

The `bne` instruction for not equal

```
bne reg1, reg2, label
```

Consider the code

```
        if (a != b) goto L1;
        // Do something
L1:     // Continue
```

Use the `bne` instruction

```
        bne $s0, $s1, L1
        # Do something
L1:     # Continue
```

Unconditional Jumps

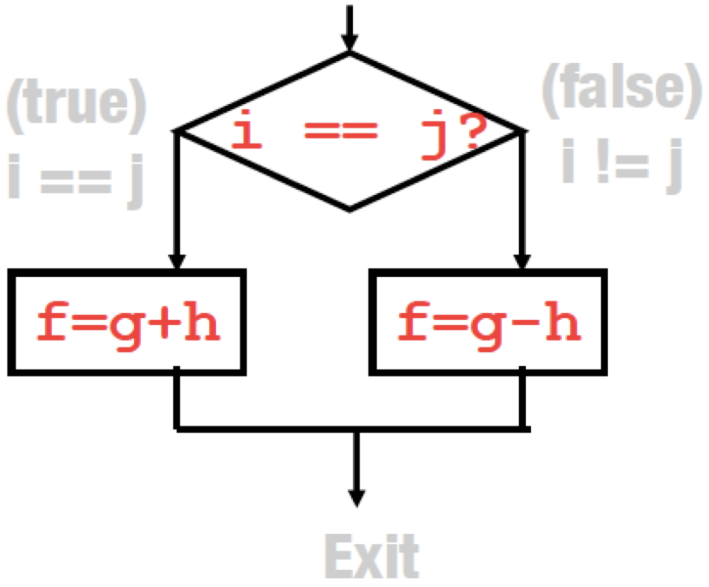
The `j` instruction jumps to a label

```
j label
```

Unconditional Jump

Consider the code

```
if (i == j) f = g + h;
else f = g - h;
```




```
        beq $s3, $s4, True   # Branch if i == j
        sub $s0, $s1, $s2    # f = g - h
        j Exit               # Go to Exit
True:   add $s0, $s1, $s2    # f = g + h
Exit:
```

MIPS Comparisons

Instruction	Example	Meaning	Comments
set less than	`slt $1, $2, $3`	$1 = ($2 < $3)	comp less than signed
set less than imm	`slti $1, $2, 100`	$1 = ($2 < 100)	comp w/const signed
set less than uns	`sltu $1, $2, $3`	$1 = ($2 < $3)	comp < unsigned
set l.t. imm. uns	`sltiu $1, $2, 100`	$1 = ($2 < 100)	comp < const unsigned