Important Architectural Issues

1. General purpose or Special purpose

2. Programmability or Semantic gap

Gap between what the high-level language software needs and what the hardware provides. Determined by the Instruction Set Architecture.

3. Operating Systems Support

Support for Virtual memory, Protection, Interrupt processing etc.

4. Speeding Up Applications

Pipelining, instruction scheduling, branch processing code morphing, hardware accelerators etc. are dealt at the Microarchitecture level.

5. Cost-Performance, Power Consumption etc

What price will you pay for performance? How much power will the processor consume?

Measuring the Speed

MIPS = Million Instructions Per Second

MFLOPS = Million FLOating point ops Per Sec

GFLOPS = Billion (Giga) FLOating point ops Per Sec

TERAFLOPS = Trillion FLOating point ops Per Sec

PETAFLOPS = 10^{15} FLOating point ops Per Sec

Moore's Law.

The packaging density of transistors on an integrated circuit increases 2x every 18 months.

Gates Law.

The speed of software halves every 18 months (Microsoft is the worst offender. Software bloat almost compensates for hardware improvement due to Moore's law).

What do we do with a PETAFLOP machine? Do we have enough work for them?

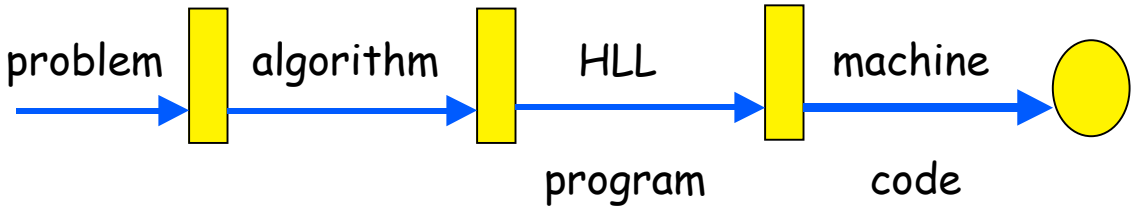
Laws of Architecture

- Signals cannot travel faster than the speed of light.
- Memory is always slower than the CPU.
- Software is slower than hardware.
- Moore's Law
- Amdahl's law

Techniques for enhancing speed

- Faster circuits
- Pipelining
- Instruction Level Parallelism (ILP)
- Single Instruction Multiple Data
- Multiple Instruction Multiple Data
- Better Algorithms

- **Factors influencing computer performance**



How fast can you solve a problem on a given machine?

Depends on

- The algorithm used
- The HLL program code
- The efficiency of the compiler

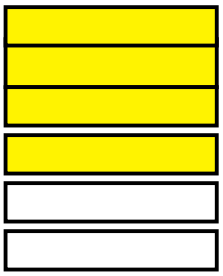
And, of course, the target machine

If the algorithm is lousy, then do not blame the computer!

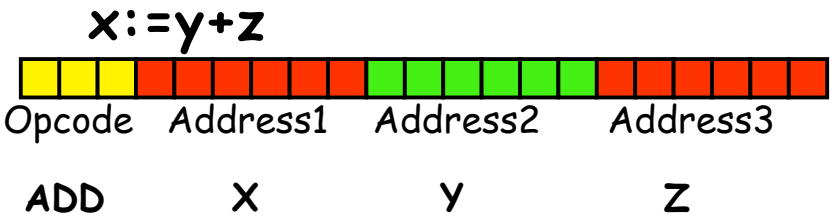
Classical Von Neumann machine

- Stored program computer
- Sequential execution of instructions
- Indistinguishability of data and instructions: instructions can be manipulated as data.

Memory Unit



Sample Instruction Formats



Data Formats

- One or more bytes.
- No special representation for typed data. The data type is determined by the opcode.

00000011 00000101

Can represent 773 if the type is binary

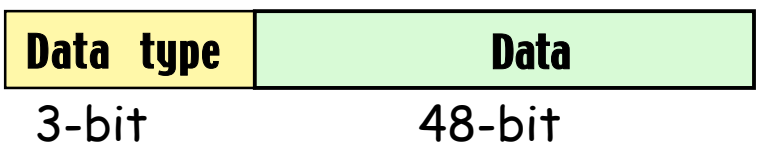
Can represent 305 if the type is BCD

Can represent 3.05 if the type is fixed point BCD

Can even represent an instruction!

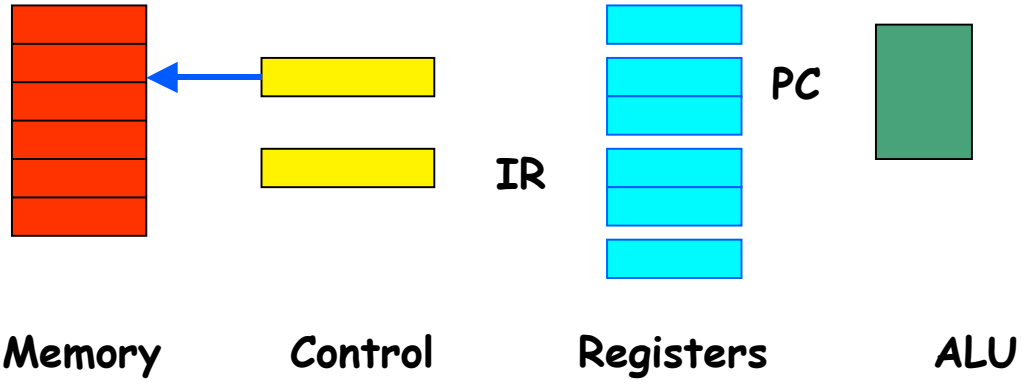
Exception

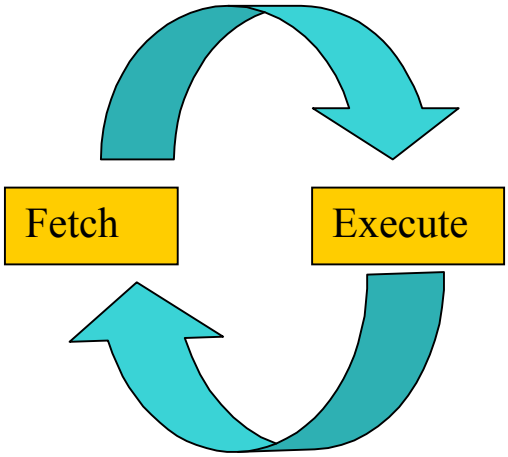
Tagged Architecture (like Burroughs B6700)



Advantages? Disadvantages?

Basic Instruction Cycle





Fetch

IR := M[PC]

Increment PC

Go to execute phase

Execute

Decode instruction

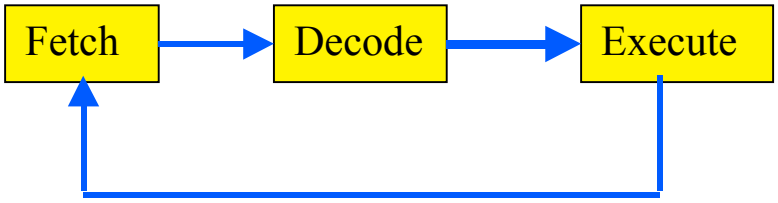
Compute operand addresses

Fetch operands

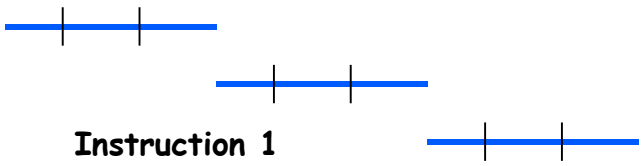
Complete the operation

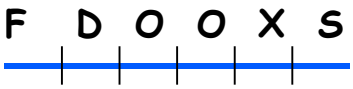
Go to fetch phase

Other Views of Instruction Cycle



If each phase takes one clock cycle (this may not always be true), then each instruction will take 3 clock cycles.

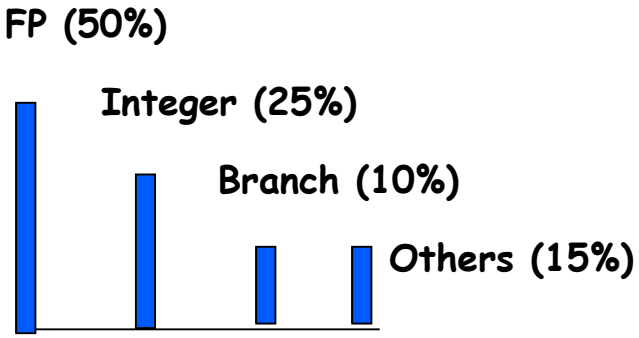




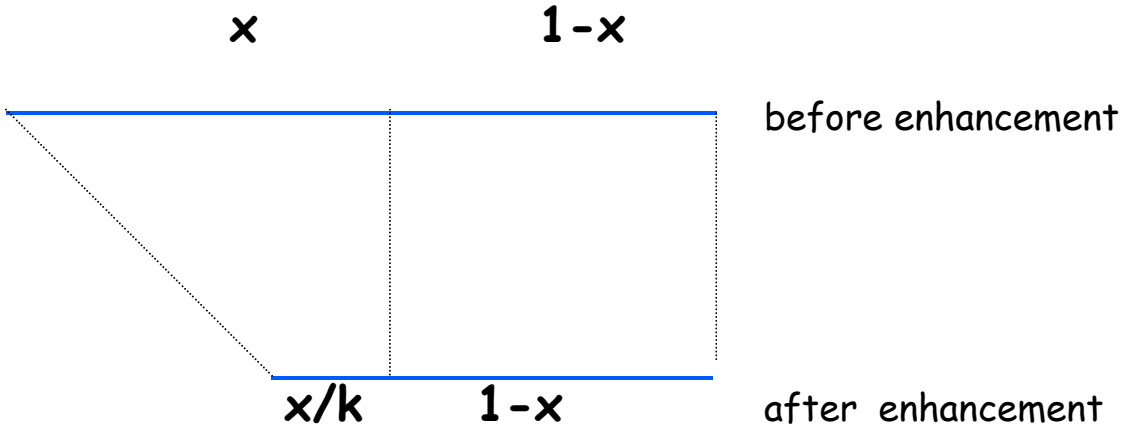
X:= Y + Z

Instruction cycle with six phases: Fetch, Decode, Op fetch, Op fetch, Execute, Store (result). CPI (Cycles Per Instruction) is an important metric for a processor.

Amdahl's Law



How to invest your $$ to speedup the above machine?



x = fraction to be enhanced **k** = enhancement factor

Overall speedup = $1/(x/k + (1-x))$

Maximum possible speedup = $1/(1-x)$

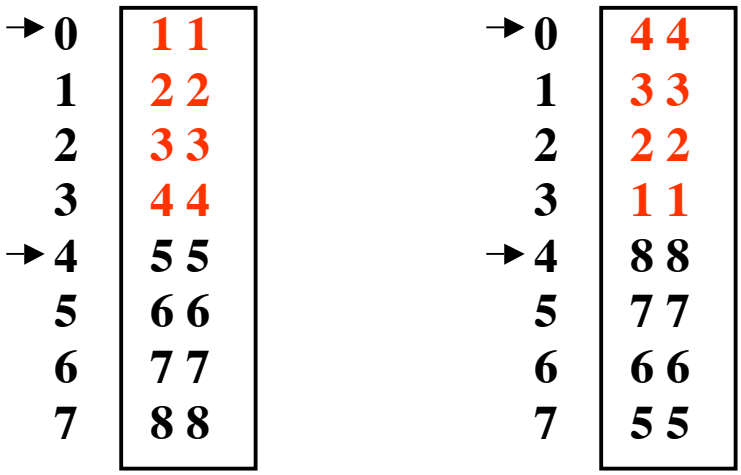
Law of diminishing return.

Little Endian vs. Big Endian

How will you store the following two 32-bit words in the memory?

X = 11223344 (hex)

Y = 55667788 (hex)



Big Endian Little Endian

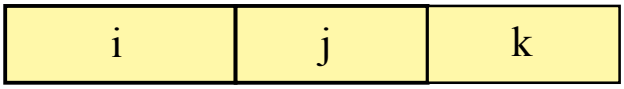
Motorola 680x0 **Intel 80x86**

(Matter of convention)

Instruction Set Design

What is the minimum number of instructions required in a processor to write any program?

The answer is ONE!



Address Address Address

This instruction does the following operation:

M[j] := M[i] – M[j]

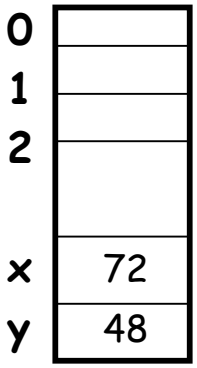
If the result is negative, then jump to address k

A processor with only one instruction will be cheap!

Why don't we design such processors?

Large Semantic Gap …

Exercise 1. Write a program to compute y:= x+y




```
19  0, 0, 20      *M[0] = 0*
20  0, y, 21      *y := 0-y*
21  x, y, 22      *y := x-(-y)*
```

Non-trivial program writing is an exercise in frustration.

Semantic Gap

Gap between what the application needs, and what the processor provides

Large semantic gap leads to explosive code sizes, and increased chance of errors.

Semantic gap decreases, when the processor supports more instructions and more data types.