Abstractions, Trends and Performance

@carlosjaimebh

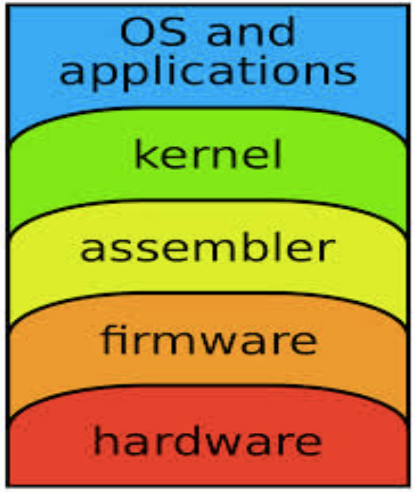
Image from: http://home.deib.polimi.it/mottola/

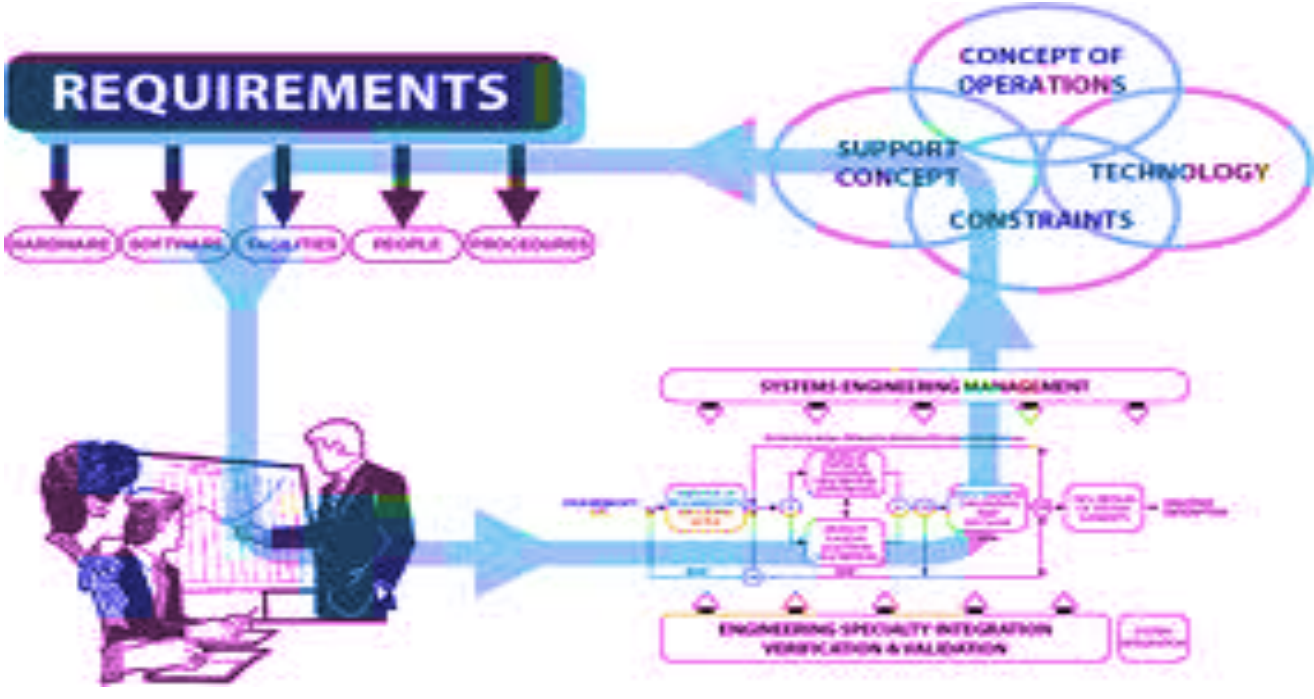
Designer/Architect of Computer Systems

- The task is a complex one: Determinate what attributes are important for a new computer:
  - **MAXIMIZE PERFORMANCE**
  - Energy Eficiency
  - Low Cost and Power
  - Availability

And the Role for a Systems Engineer?

- Take decisions, suggest to acquire or use new techology in accordance with the previous attributes and requirements.





Abstraction

Abstraction by Alisa Burke

- **Abstraction** is a process by which concepts are derived from the usage and classification of literal ("real" or "concrete") concepts, first principles, or other methods.
- Abstractions may be formed by reducing the information content of a concept or an observable phenomenon, typically to retain only information which is relevant for a particular purpose.

WIKIPEDIA
The Free Encyclopedia

Levels of Abstraction

Abstraction	Implemented With
Computer	Circuit board(s)
Circuit board	Processor, memory, and bus adapter chips
Processor	VLSI chip
VLSI chip	Many gates
Gate	Many transistors
Transistor	Semiconductor implemented in silicon

Figure 2.26 An example of levels of abstraction in digital logic. An item at one level is implemented using items at the next lower level.

From https://eca.cs.purdue.edu/index.html

Transistor Representation (Review)

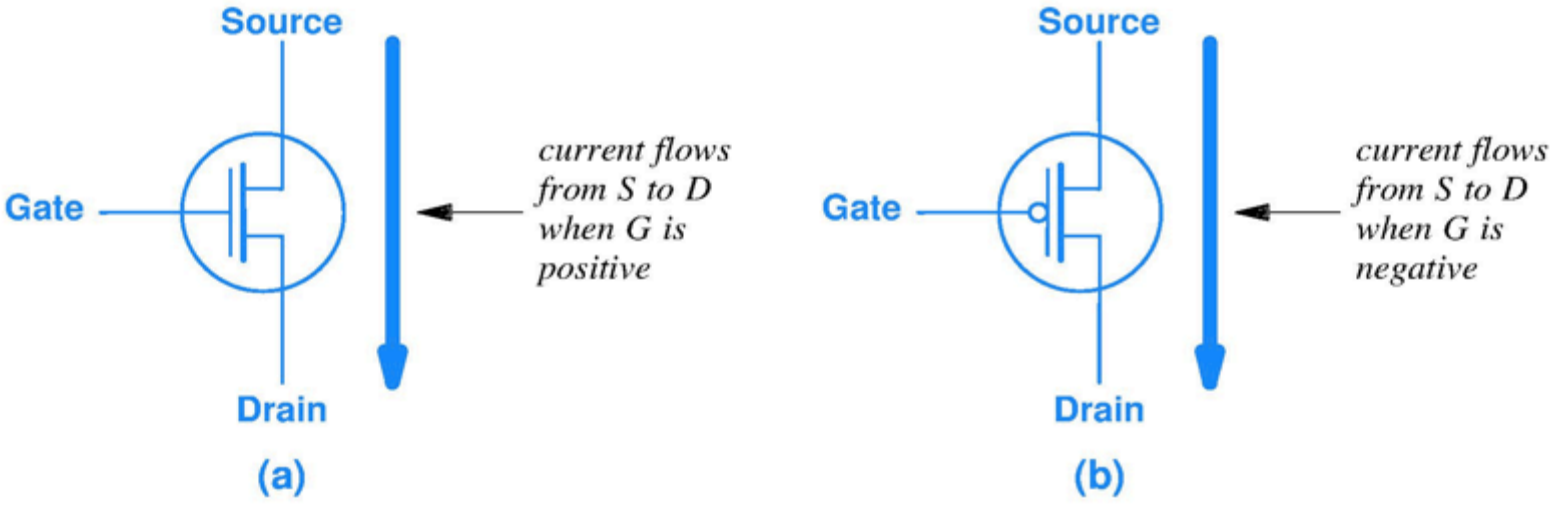


Figure 2.1 The two types of transistors used in logic circuits. The type labeled (a) turns on when the gate voltage is positive; transistor labeled (b) turns on when the gate voltage is zero (or negative).

From https://eca.cs.purdue.edu/index.html

Logical Gates Representation (Review)

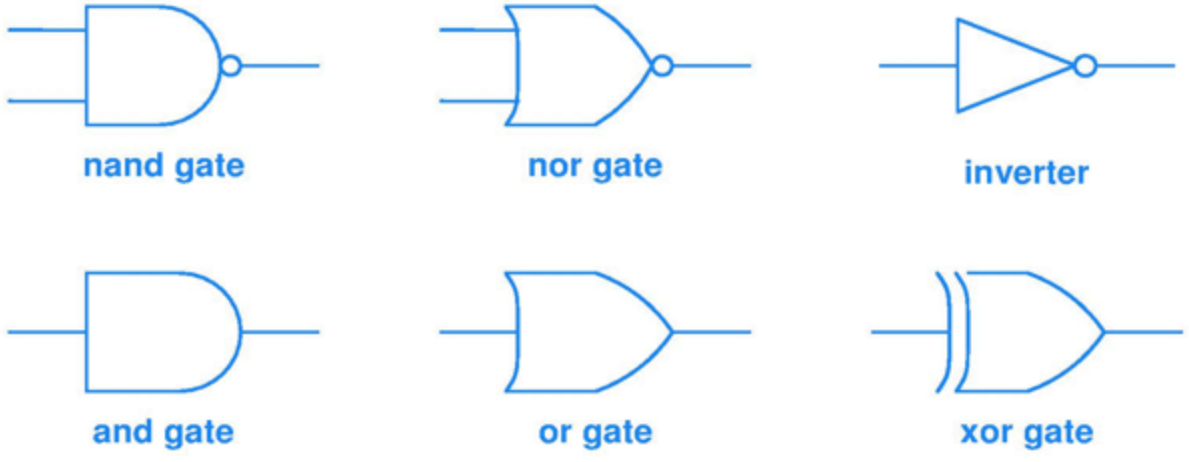


Figure 2.6 The symbols for commonly used gates. Inputs for each gate are shown on the left, and the output of the gate is shown on the right.

From https://eca.cs.purdue.edu/index.html

Binary Counters Representation (Review)

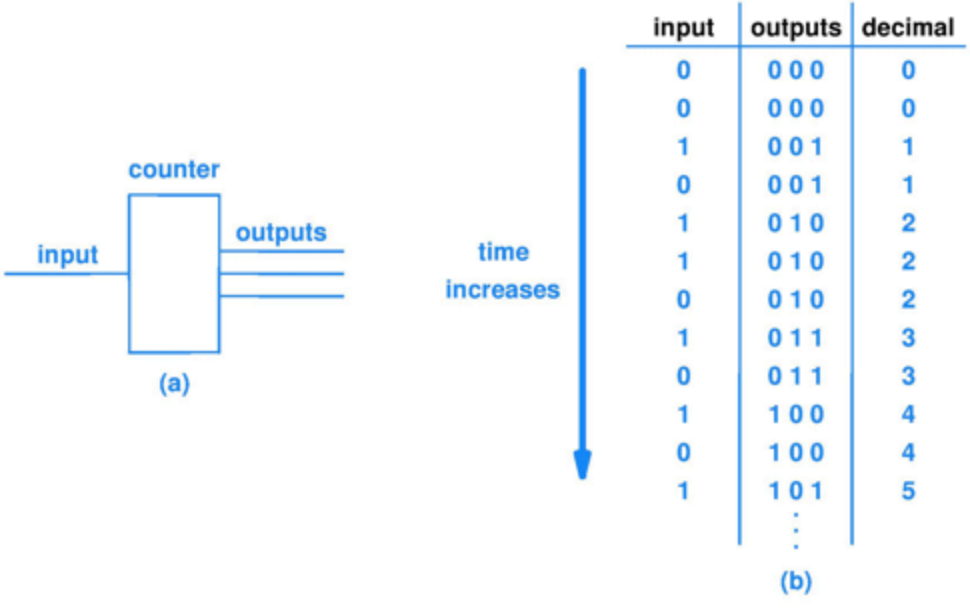


Figure 2.19 Illustration of (a) a binary counter, and (b) a sequence of input values and the corresponding outputs. The column labeled *decimal* gives the decimal equivalent of the outputs.

From https://eca.cs.purdue.edu/index.html

Clocks and Sequences Representation

Traditional Computer Sequence

1. Test the battery
2. Power on and test the memory
3. Start the disk spinning
4. Power up the screen
5. Read the boot sector from disk into memory
6. Start the CPU

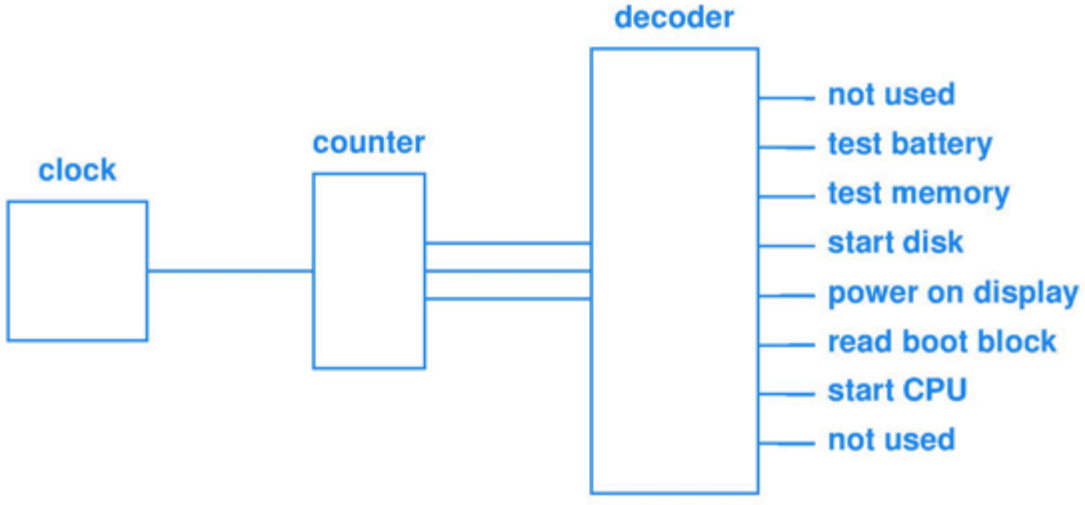
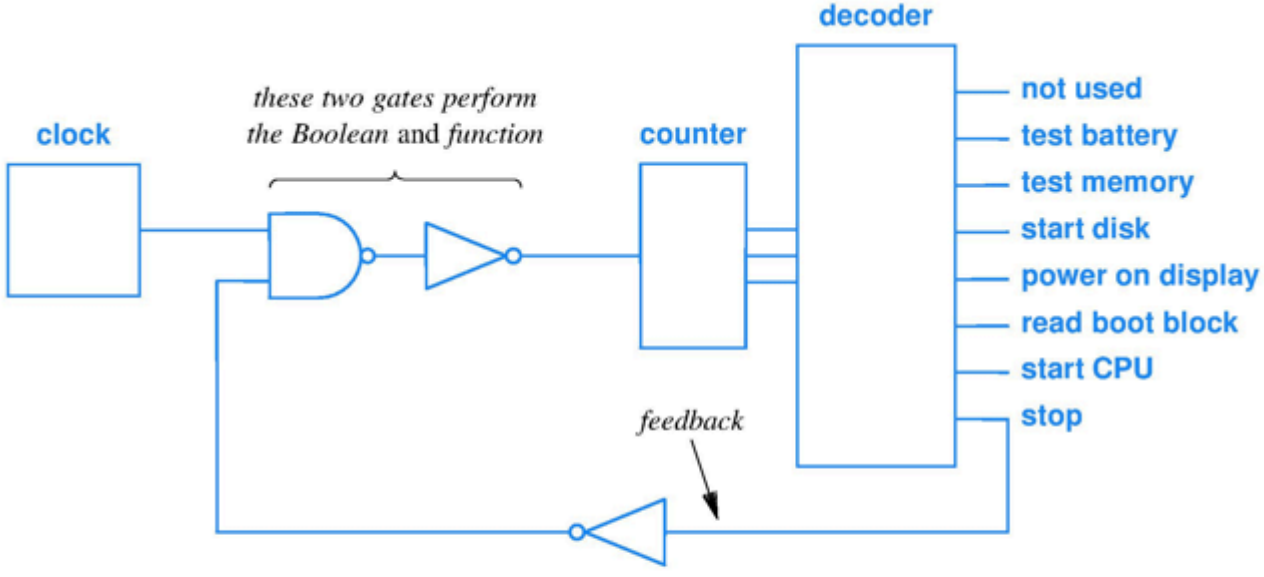


Figure 2.21 An illustration of how a clock can be used to create a circuit that performs a sequence of six steps. Output lines from the counter connect directly to input lines of the decoder.

From https://eca.cs.purdue.edu/index.html

The Feedback Concept



CLOCK and (not F)

Figure 2.22 A modification of the circuit in Figure 2.21 that includes feedback to stop processing after one pass through each output.

From https://eca.cs.purdue.edu/index.html

Architectural Representation

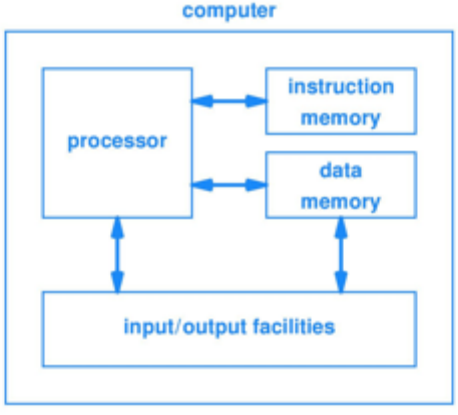


Figure 4.1 Illustration of the Harvard Architecture that uses two memories, one to hold programs and another to store data.

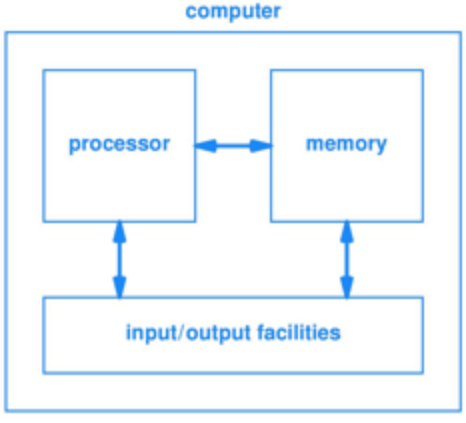


Figure 4.2 Illustration of the Von Neumann Architecture. Both programs and data can be stored in the same memory.

From https://eca.cs.purdue.edu/index.html

Processors Representation

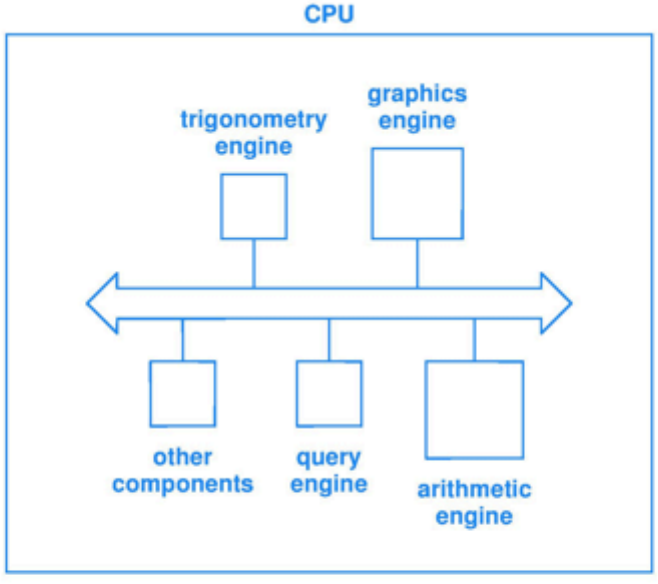


Figure 4.3 An example of a CPU that includes multiple components. The large arrow in the center of the figure indicates a central interconnect mechanism that the components use to coordinate.

From https://eca.cs.purdue.edu/index.html

Processor Structure Representation

Conceptual Units:

- Controller
- Arithmetic Logic Unit (ALU)
- Local Data Storage (Registers)
- Internal Interconections
- External Interfaces (I/Os)

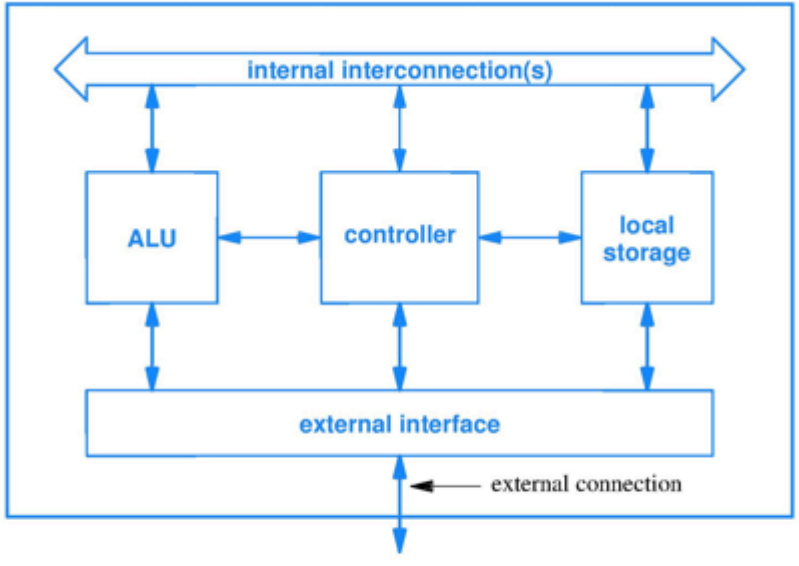
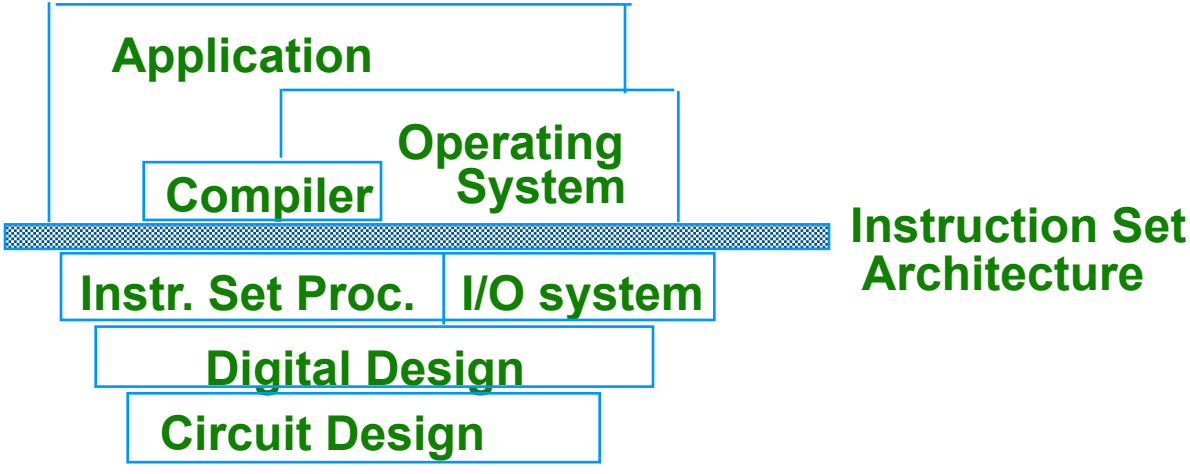


Figure 4.4 The five major units found in a conventional processor. The external interface connects to the rest of the computer system.

From https://eca.cs.purdue.edu/index.html

Instruction Set Architecture (ISA)

° Co-ordination of *levels of abstraction*



° Under a set of rapidly changing *Forces*

From https://eca.cs.purdue.edu/index.html

Binary Weighted Positional Representation

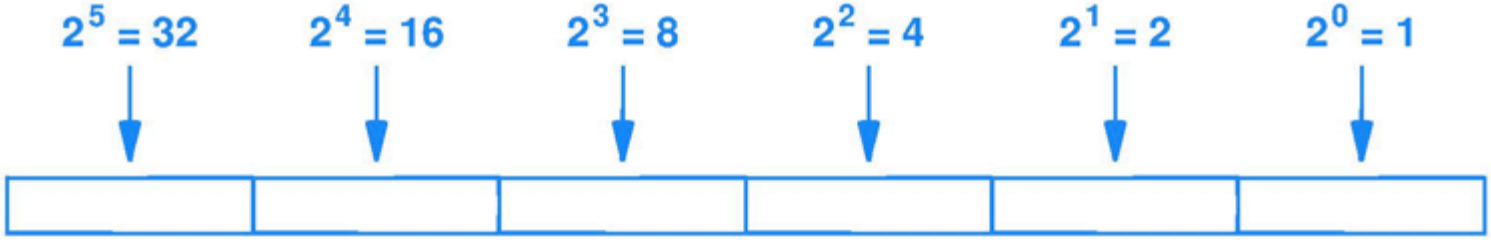


Figure 3.2 The value associated with each of the first six bit positions when using a positional interpretation in base two.

From https://eca.cs.purdue.edu/index.html

Processors Categories and Roles

- Coprocessors
  - Accelerators
- Microcontrollers
- Embedded System Processors
- General-purpose Processors
- Specific-purpose Processors

Program Translation

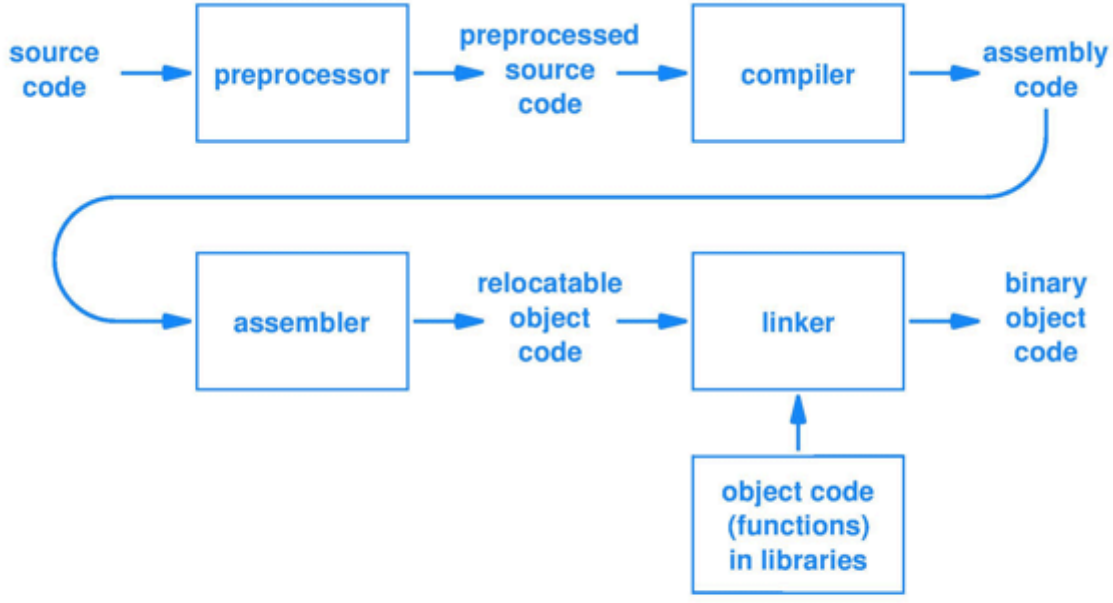
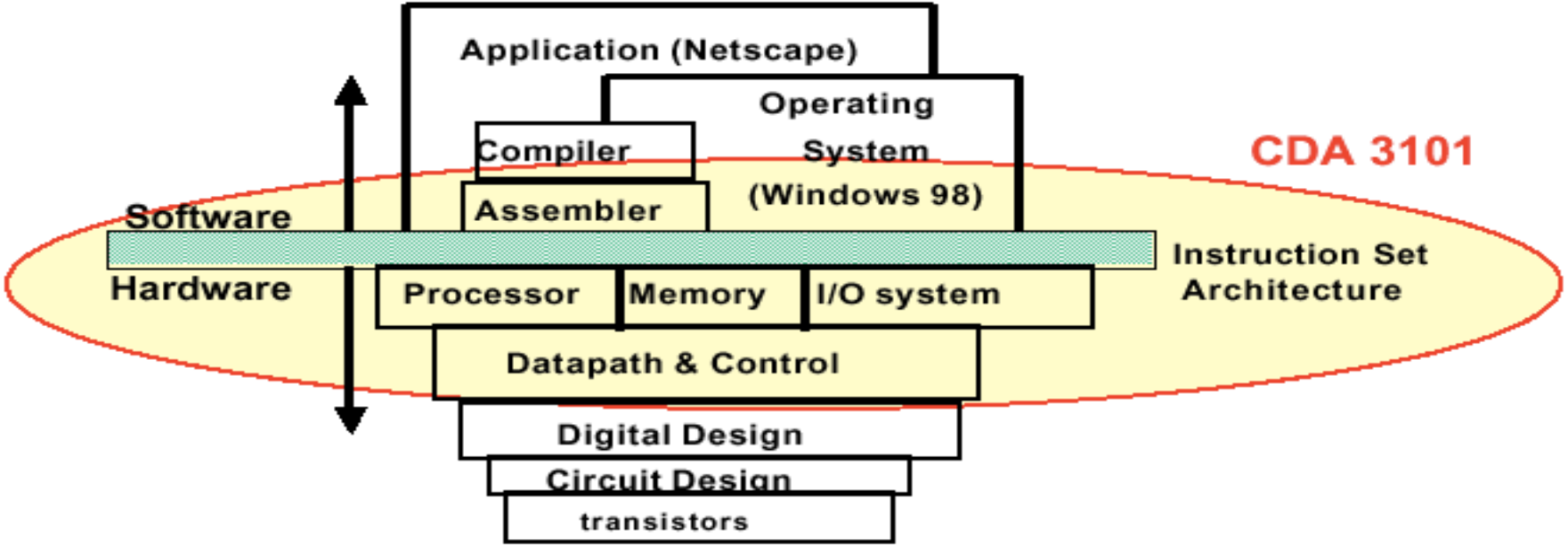


Figure 4.6 The steps used to translate a source program to the binary object code representation used by a processor.

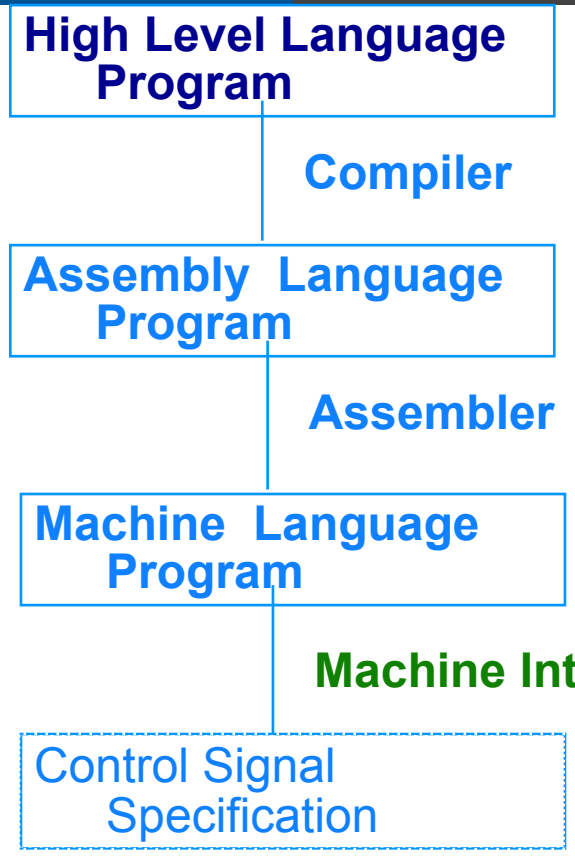
From https://eca.cs.purdue.edu/index.html

ISA Levels



http://www.cise.ufl.edu/~mssz/CompOrg/CDAintro.html

Review: Levels of Representation




```
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)
```

```
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
```

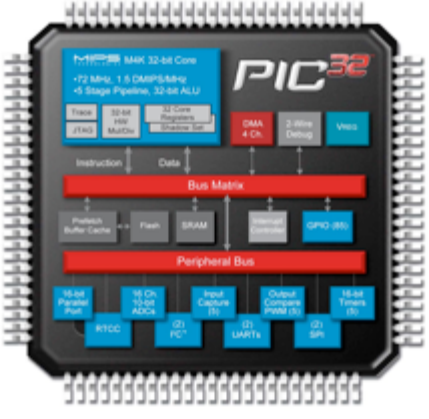
80X86 Architecture

- X86 denotes a family of ISAs based on the Intel® 8086CPU.

MIPS Architecture

- Microprocessor without Interlocked Pipeline Stages is a RISC_ISA computer developed by MIPS Technologies, formerly used in Embedded Systems or video game consoles (Sony® PlayStation®) .

MIPS TECHNOLOGIES



Some Aspects of ISA

1. Class of ISA
2. Memory Addressing
3. Addresing Modes
4. Types of Sizes of Operands
5. Operations
6. Control Flow Instructions
7. Encoding and ISA

1. Class of ISA

- All ISA are classified as general-purpose register architecture (Operands are either registers or memory locations)
  - 80x86 has a 16 general-purpose registers (16 floating-point data)
  - MIPS has 32 general-purpose registers (32 floating-point reigisters)
- Two Popular versions:
  - Register Memory ISAs (80x86)
  - Load Store ISAs (ARM, MIPS)

1. Class of ISA: An Example

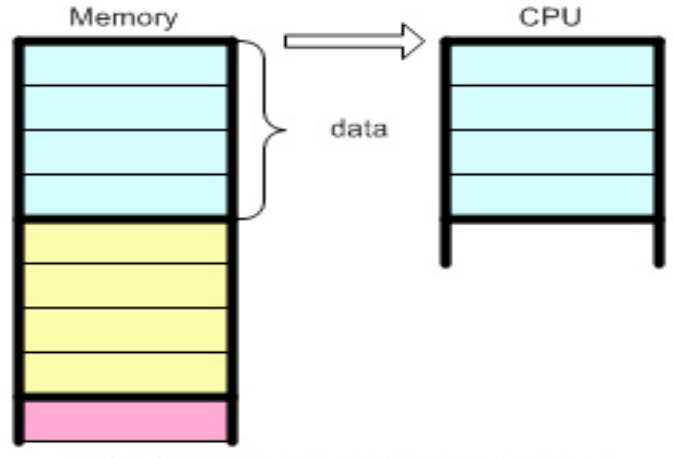
MIPS registers and usage conventions

Name	Register Number	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	value for results and expressions	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

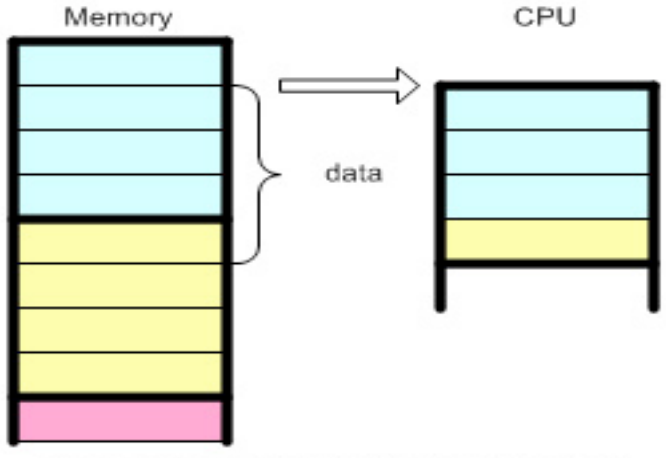
2. Memory Addressing

- All desktop and servers computers use byte addressing to access memory operands.
  - Some architectures (ARMS, MIPS) require that objects must be aligned.
  - 80x86 does not require alignement, but accesses are generally faster if operands are aligned

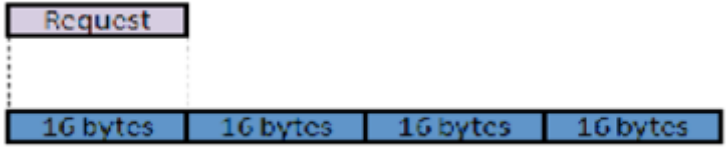
2. Memory Addressing
Aligned and Missalinged examples



4-byte memory access for aligned data



4-byte memory access for misaligned data



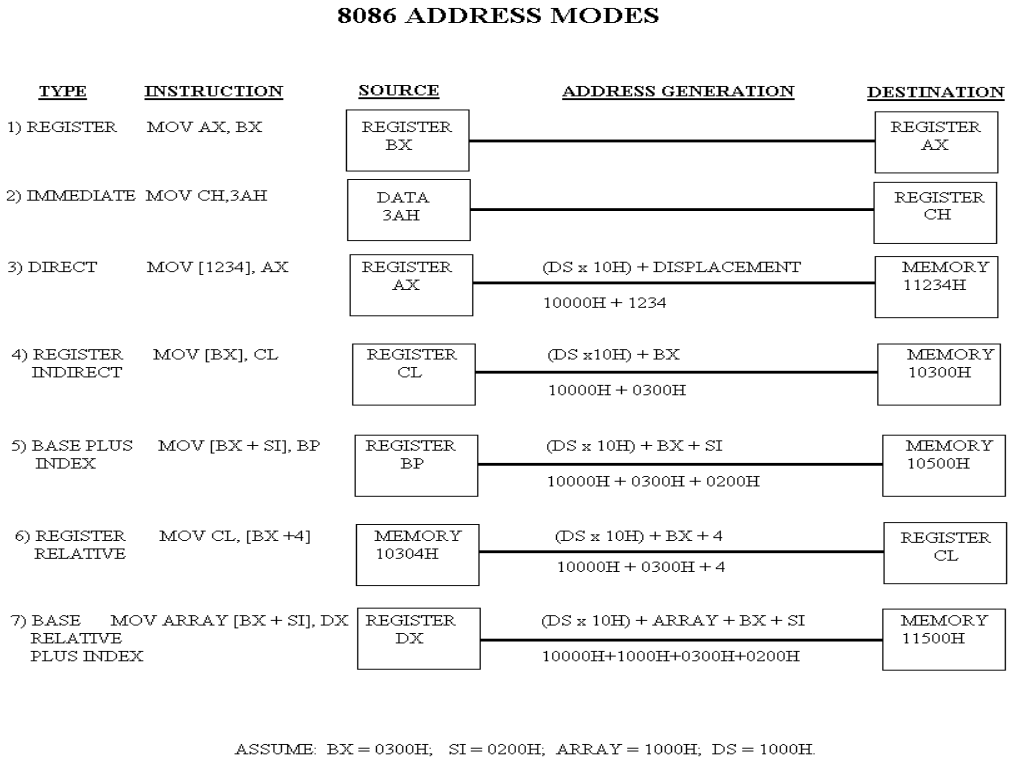
Aligned Request

Unaligned (misaligned) Request

3. Addressing Modes

- Addressing Modes specify the address of a memory object, registers and constant operands.
  - MIPS: Register, Immediate (Constants), Displacement.
  - 80x86: Support the previous three + three variations of displacement:
    - no register (absolute)
    - two registers (based indexed with displacement)
    - two registers (based with scaled index and displacement)
  - ARM: The three MIPS registers + PC-Relative addressing, the sum of two registers and the sum of two registers multiplied by the size of the operand in bytes.

3. Addressing Modes: An Example



4. Types and Sizes of Operands

- 8-bit (ASCII character)
- 16-bit (Unicode Character or half word)
- 32-bit (Integer or word)
- 64-bit (Double word or long integer)
- IEE 754 Floating Point in 32-bit (Single Precision) and 64-bit (Double precision)
- 80-bit Floating Point (Extended Double Precision Only for 80x86)

5. Operations

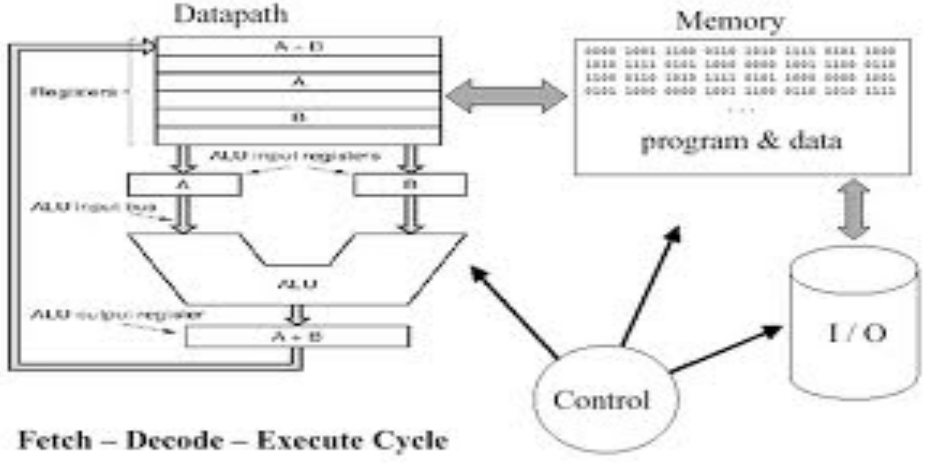
- Data Transfer
  - Moves data between registers and memory, or between the integer and special registers.
- Arithmetic Logical
  - Operations on Integer or Logical data in GPRs (General Purpose Registers)
- Control
  - Conditional branches and jumps
- Floating Point
  - Floating Point Operations on Double Precision and Single Precisions Formats

5. Operations:
Subset of the Instructions in MIPS64

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR*
LB,LBU,SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH,LHU,SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW,LWU,SW	Load word, load word unsigned, store word (to/from integer registers)
LD,SD	Load double word, store double word (to/from integer registers)
L.S,L.D,S.S,S.D	Load SP float, load DP float, store SP float, store DP float
MFC0,MTC0	Copy from/to GPR to/from a special register
MOV.S,MOV.D	Copy one SP or DP FP register to another FP register
MFC1,MTC1	Copy 32 bits to/from FP registers from/to integer registers
Arithmetic/logical	*Operations on integer or logical data in GPRs; signed arithmetic trap on overflow*
DADD,DADDI,DADDU,DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB,DSUBU	Subtract; signed and unsigned
DMUL,DMULU,DDIV, DDIVU,MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL,DSRL,DSRA,DSLLV, DSRLV,DSRAV	Shifts: both immediate (DS__) and variable form (DS__V); shifts are shift left logical, right logical, right arithmetic
SLT,SLTI,SLTU,SLTIU	Set less than, set less than immediate; signed and unsigned
Control	*Conditional branches and jumps; PC-relative or through register*
BEQZ,BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ,BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T,BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN,MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J,JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL,JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode
Floating point	*FP operations on DP and SP formats*
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D,SUB.S,SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D,MADD.S,MADD.PS	Multiply-add DP, SP numbers, and pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C.__.D,C.__.S	DP and SP compares: "__" = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

6. Control Flow Instructions

- Conditional Branches
- Unconditional Jumps
- Procedure Calls
- Returns



Fetch – Decode – Execute Cycle

7. Encoding an ISA

- Fixed Length
  - ARM and MIPS (32-bits long)
- Variable Length
  - 80x86 (Ranking from 1 to 18 bytes)

- I.E. MIPS instruction encoding formats:
  - R-type (6-bit opcode, 5-bit rs, 5-bit rt, 5-bit rd, 5-bit shamt, 6-bit function code)

31-26	25-21	20-16	15-11	10-6	5-0
opcode	*rs*	*rt*	*rd*	shamt	function

  - I-type (6-bit opcode, 5-bit rs, 5-bit rt, 16-bit immediate)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*imm*

  - J-type (6-bit opcode, 26-bit pseudo-direct address)

31-26	25-0
opcode	*pseudodirect jump address*

Designing and Organization

- Implementation of a computer has two components: organization and hardware.
  - Organization: High level aspects (memory system, memory interconection, design of the CPU)
    - Microarchitecture
      - Example: two processors with the same ISA but different organization are AMD Opteron adn Intel Core i7.
  - Hardware: Specifics of a Computer (Detailed logic design and the packaging technology of the computer)

Summary of Some Functional Requerimients an Architect Faces

Functional requirements	Typical features required or supported
Application area	*Target of computer*
Personal mobile device	Real-time performance for a range of tasks, including interactive performance for graphics, video, and audio; energy efficiency (Ch. 2, 3, 4, 5; App. A)
General-purpose desktop	Balanced performance for a range of tasks, including interactive performance for graphics, video, and audio (Ch. 2, 3, 4, 5; App. A)
Servers	Support for databases and transaction processing; enhancements for reliability and availability; support for scalability (Ch. 2, 5; App. A, D, F)
Clusters/warehouse-scale computers	Throughput performance for many independent tasks; error correction for memory; energy proportionality (Ch 2, 6; App. F)
Embedded computing	Often requires special support for graphics or video (or other application-specific extension); power limitations and power control may be required; real-time constraints (Ch. 2, 3, 5; App. A, E)
Level of software compatibility	*Determines amount of existing software for computer*
At programming language	Most flexible for designer; need new compiler (Ch. 3, 5; App. A)
Object code or binary compatible	Instruction set architecture is completely defined—little flexibility—but no investment needed in software or porting programs (App. A)
Operating system requirements	*Necessary features to support chosen OS (Ch. 2; App. B)*
Size of address space	Very important feature (Ch. 2); may limit applications
Memory management	Required for modern OS; may be paged or segmented (Ch. 2)
Protection	Different OS and application needs: page vs. segment; virtual machines (Ch. 2)
Standards	*Certain standards may be required by marketplace*
Floating point	Format and arithmetic: IEEE 754 standard (App. J), special arithmetic for graphics or signal processing
I/O interfaces	For I/O devices: Serial ATA, Serial Attached SCSI, PCI Express (App. D, F)
Operating systems	UNIX, Windows, Linux, CISCO IOS
Networks	Support required for different networks: Ethernet, Infiniband (App. F)
Programming languages	Languages (ANSI C, C++, Java, Fortran) affect instruction set (App. A)

From Hennesy and Patterson

Trends in Technology

- Integrated Circuit Logic Technology
  - Transistor density increases by about 35%/year, quadrupling somewhat over four years (Moore's Law)

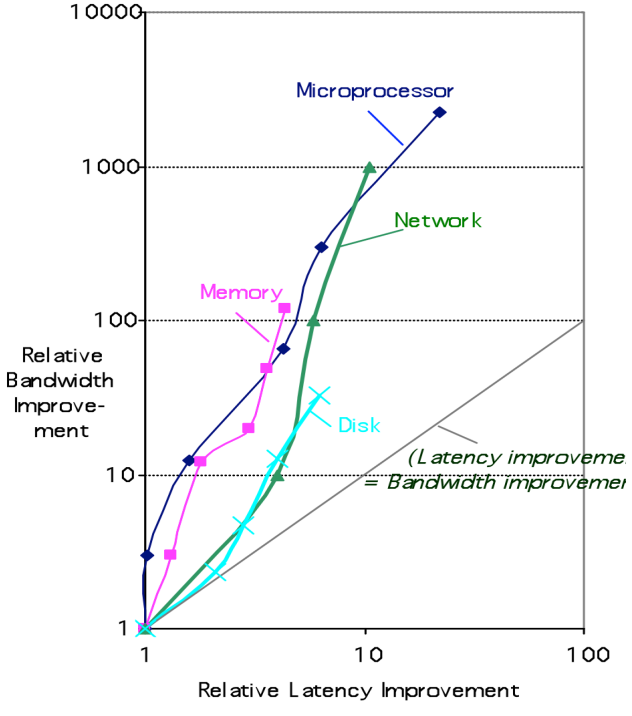
CA:AQA Edition	Year	DRAM growth rate	Characterization of impact on DRAM capacity
1	1990	60%/year	Quadrupling every 3 years
2	1996	60%/year	Quadrupling every 3 years
3	2003	40%–60%/year	Quadrupling every 3 to 4 years
4	2007	40%/year	Doubling every 2 years
5	2011	25%–40%/year	Doubling every 2 to 3 years

Trends in Technology

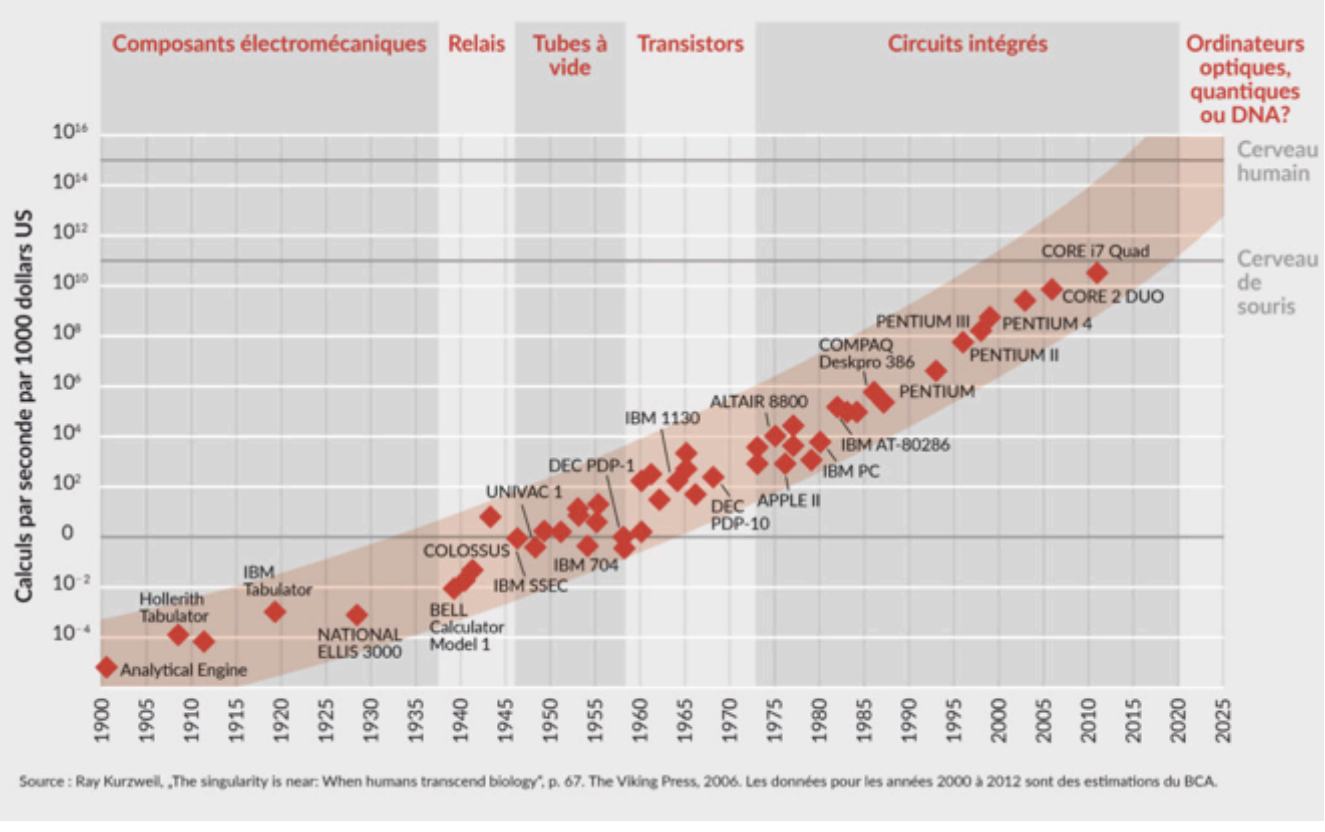
- Semiconductor Flash
  - Electrically erasable programmable read-only memory (Flash Memory)
- Magnetic Disk Technology
  - Disks are 15 to 25 times cheaper per bit than flash.
- Network Technology
  - Network performance depends both on the performance of switches and on the performance of the transmission system.
- Organization Trends
  - RISC Technology (ARM, RISC-V)
  - Tensor and Accelerated Unit Processors (GPUs, TPUs)
- New Materials
  - Graphene
  - DNA and BioMaterials
  - Optics and Quantum Materials

Performance Trends:
Bandwidth over Latency

Microprocessor	16-bit address/ bus, microcoded	32-bit address/ bus, microcoded	5-stage pipeline, on-chip I & D caches, FPU	2-way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip L2 cache	Multicore OOO 4-way on chip L3 cache, Turbo
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Year	1982	1985	1989	1993	1997	2001	2010
Die size (mm^2)	47	43	81	90	308	217	240
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000	1,170,000,000
Processors/chip	1	1	1	1	1	1	4
Pins	68	132	168	273	387	423	1366
Latency (clocks)	6	5	5	5	10	22	14
Bus width (bits)	16	32	32	64	64	64	196
Clock rate (MHz)	12.5	16	25	66	200	1500	3333
Bandwidth (MIPS)	2	6	25	132	600	4500	50,000
Latency (ns)	320	313	200	76	50	15	4
Memory module	DRAM	Page mode DRAM	Fast page mode DRAM	Fast page mode DRAM	Synchronous DRAM	Double data rate SDRAM	DDR3 SDRAM
Module width (bits)	16	16	32	64	64	64	64
Year	1980	1983	1986	1993	1997	2000	2010
Mbits/DRAM chip	0.06	0.25	1	16	64	256	2048
Die size (mm^2)	35	45	70	130	170	204	50
Pins/DRAM chip	16	16	18	20	54	66	134
Bandwidth (MBytes/s)	13	40	160	267	640	1600	16,000
Latency (ns)	225	170	125	75	62	52	37
Local area network	Ethernet	Fast Ethernet	Gigabit Ethernet	10 Gigabit Ethernet	100 Gigabit Ethernet		
IEEE standard	802.3	803.3u	802.3ab	802.3ac	802.3ba		
Year	1978	1995	1999	2003	2010		
Bandwidth (Mbits/sec)	10	100	1000	10,000	100,000		
Latency (µsec)	3000	500	340	190	100		
Hard disk	3600 RPM	5400 RPM	7200 RPM	10,000 RPM	15,000 RPM	15,000 RPM	
Product	CDC Wren I 94145-36	Seagate ST41600	Seagate ST15150	Seagate ST39102	Seagate ST373453	Seagate ST3600057	
Year	1983	1990	1994	1998	2003	2010	
Capacity (GB)	0.03	1.4	4.3	9.1	73.4	600	
Disk form factor	5.25 inch	5.25 inch	3.5 inch	3.5 inch	3.5 inch	3.5 inch	
Media diameter	5.25 inch	5.25 inch	3.5 inch	3.0 inch	2.5 inch	2.5 inch	
Interface	ST-412	SCSI	SCSI	SCSI	SCSI	SAS	
Bandwidth (MBytes/s)	0.6	4	9	24	86	204	
Latency (ms)	48.3	17.1	12.7	8.8	5.7	3.6	

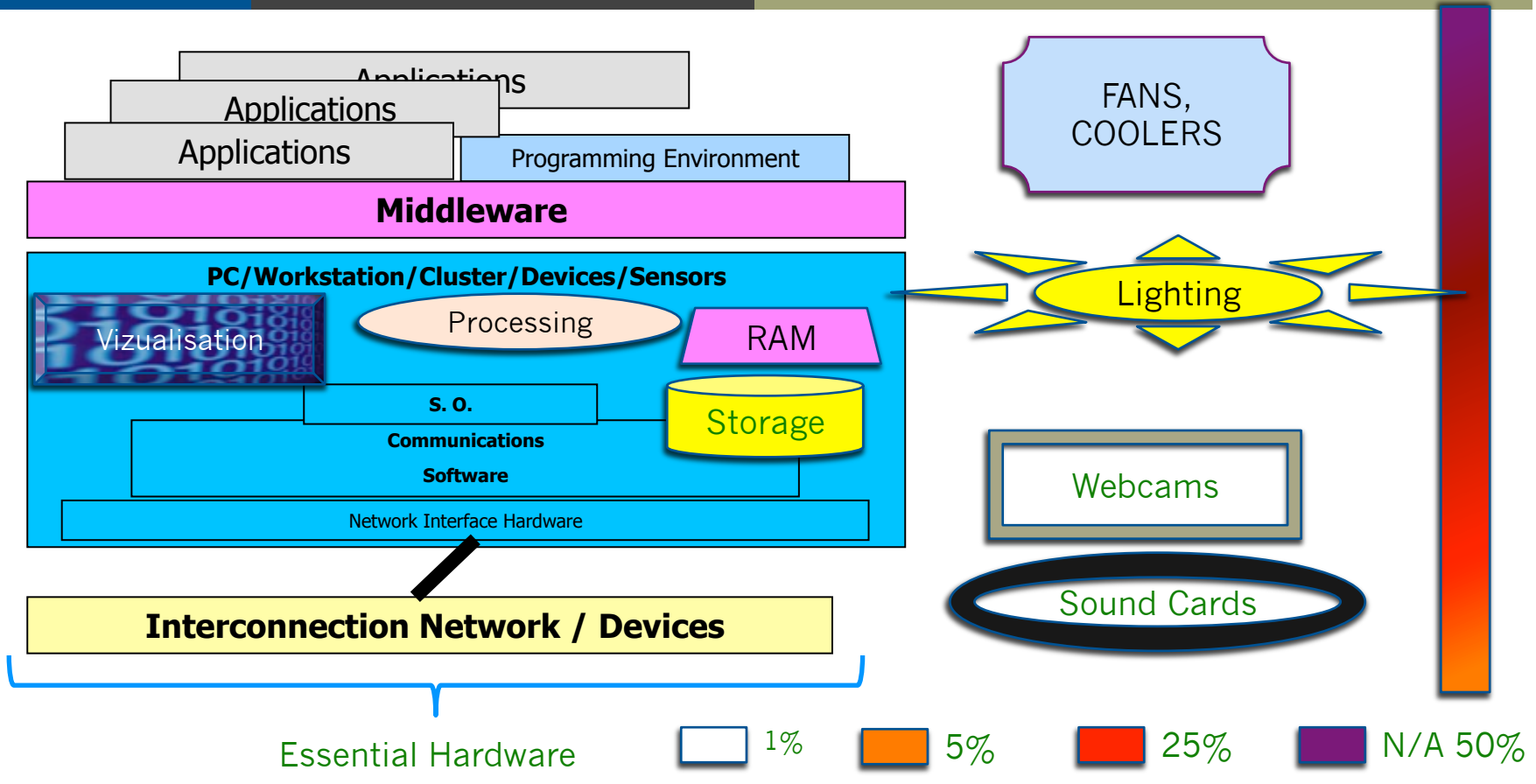


Trends Following Moore's Law

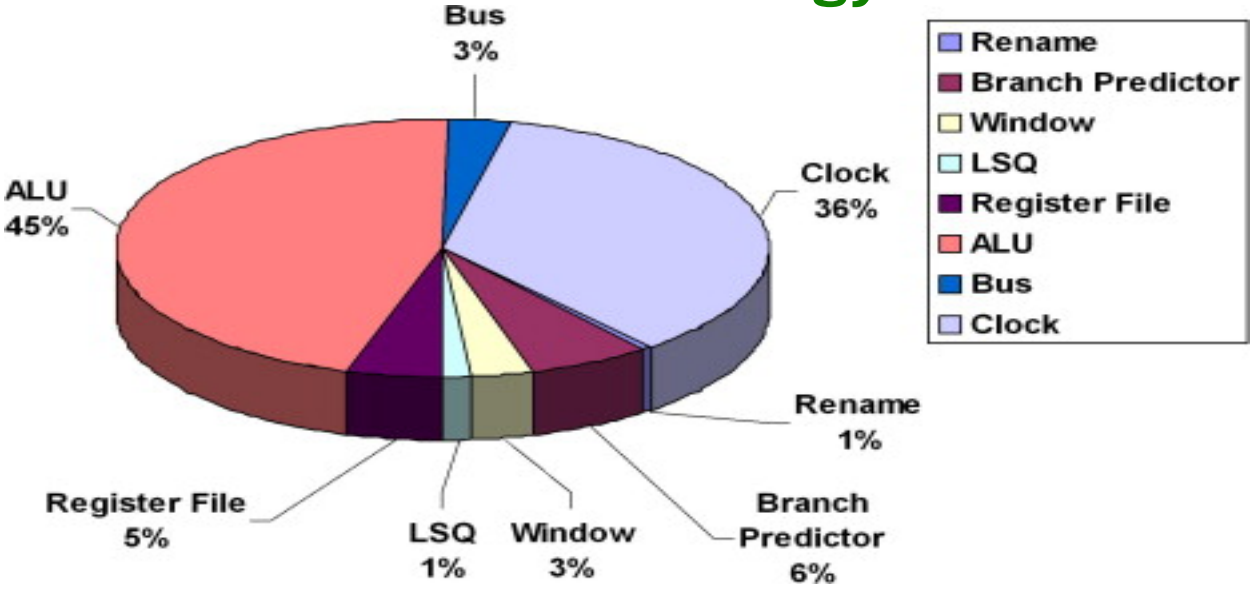


Source : Ray Kurzweil, „The singularity is near: When humans transcend biology", p. 67. The Viking Press, 2006. Les données pour les années 2000 à 2012 sont des estimations du BCA.

Performance Trends Power and Energy



Performance Trends
Power and Energy



The distribution of the energy consumed in the processor components.

From Nikolaos Kroupis , Dimitrios Soudris, **FILESPPA: Fast Instruction Level Embedded System Power and Performance Analyzer**

Microprocessors and Microsystems Volume 35, Issue 3 2011 329 - 342

Performance Trends
Power and Energy:
Techniques to improve energy efficiency

1. Do nothing well: Turn off the clock of inactive modules to save energy and dynamic power.
2. Dynamic Voltage-Frequency Scaling (DVFS)
3. Design for Typical Case (Scheduling of Activity)
4. Overclocking (Turbo Mode)

Trends in Cost

- Impact of Time, Volume and Commoditization
  - Learning Curve
  - Manufacturing Costs
  - Transport
  - Market (Cost vs Price)
  - Operation Costs
  - **Dependability**
    - **Service Level Agreements (Infrastructure)**
    - **Service Level Objects (Networking, Power)**

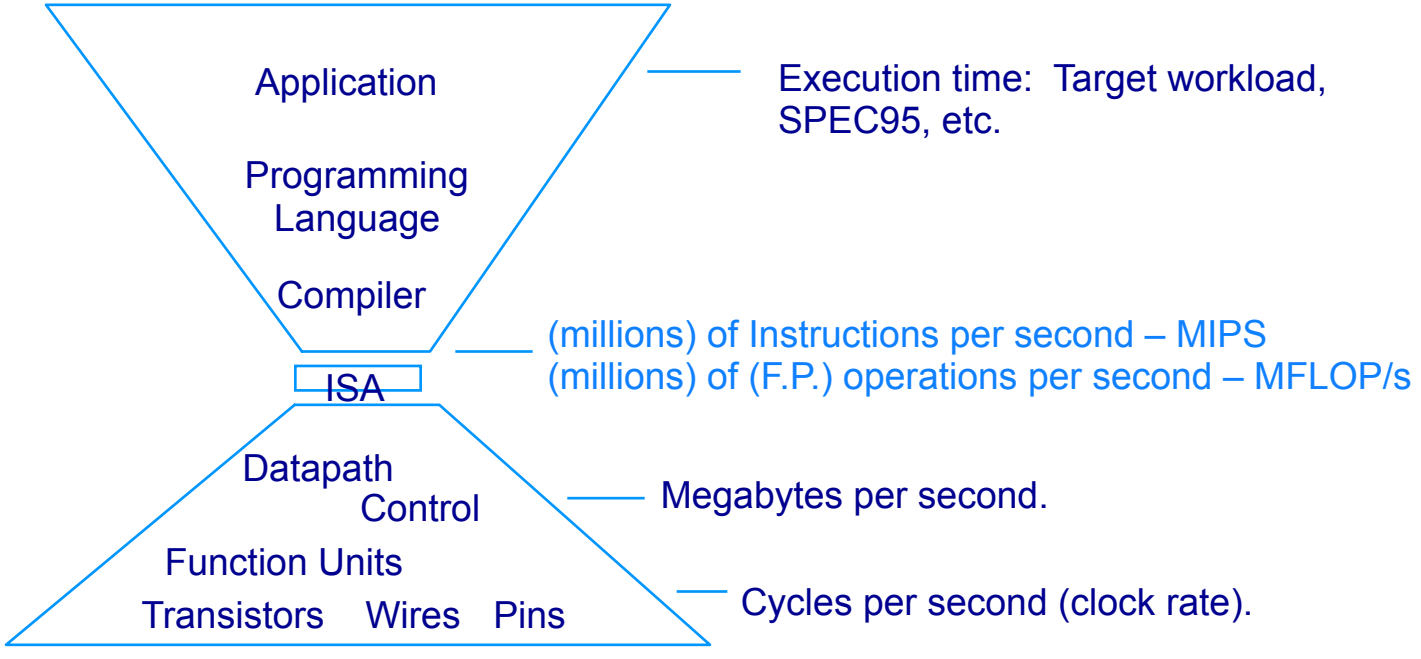
Performance

- A Computer System exists to IMPROVE PERFORMANCE
  - High Speed
  - Data Treatment
  - Availability
  - Capacity
  - Low Latency
  - High Bandwdith

Measuring Performance

- In order to compare how fast computers can process data, we have to measure their performance.
- There are a number of measurements of performance.
- Clock speed, MIPS, FLOPS & Benchmark tests are all used. Some are a better measure than others.

Metrics of Computer Performance



Each metric has a purpose, and each can be misused.

Computer Performance

- *Response Time* (*elapsed time*, *latency*):
  - how long does it take for *my* job to run?
  - how long does it take to execute (start to finish) *my* job?
  - how long must *I* wait for the database query?
- *Throughput*:
  - how *many* jobs can the machine run at once?
  - what is the *average* execution rate?
  - how *much* work is getting done?
- *If we upgrade a machine with a new processor what do we increase?*
- *If we add a new machine to the lab what do we increase?*

Individual user concerns...

Systems manager concerns...

Execution Time

- *Elapsed Time*
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - a useful number, but often not good for comparison purposes

elapsed time = CPU time + wait time (I/O, other programs, etc.)

- *CPU time*
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

CPU time = user CPU time + system CPU time

⇒ elapsed time = user CPU time + system CPU time + wait time

- Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*)
  - time spent executing the lines of code that are *in our program*

Definition of Performance

- For some program running on machine X:

  $$\text{Performance}_X = 1 / \text{Execution time}_X$$

- *X is n times faster than Y* means:

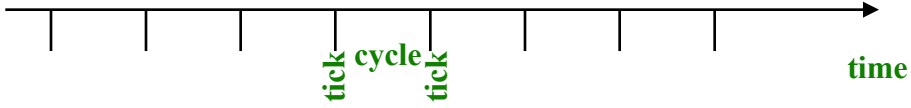
  $$\text{Performance}_X \ / \ \text{Performance}_Y = n$$

Clock Cycles

- Instead of reporting execution time in seconds, we often use *cycles*. In modern computers hardware events progress cycle by cycle: in other words, each event, e.g., multiplication, addition, etc., is a sequence of cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- *Clock ticks* indicate start and end of cycles:



- *cycle time* = time between ticks = seconds per cycle

- *clock rate* (*frequency*) = cycles per second (1 Hz. = 1 cycle/sec, 1 MHz. = 10^6 cycles/sec)

- *Example*: A 200 Mhz. clock has a $\frac{1}{200 \times 10^6} \times 10^9 = 5$ nanoseconds cycle time

Clock Speed

- The clock signal is carried by one of the lines on the control bus.
- One single pulse is called a ‘clock cycle’.
- Measured in Megahertz (MHz) & Gigahertz (GHz). 1 MHz = 1 million pulses per second. 1 GHZ = 1000 MHz.

Processor Clock Speed

- CPU clock speeds are compared at http://www.cpubenchmark.net/common_cpus.html

Performance Equation I

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

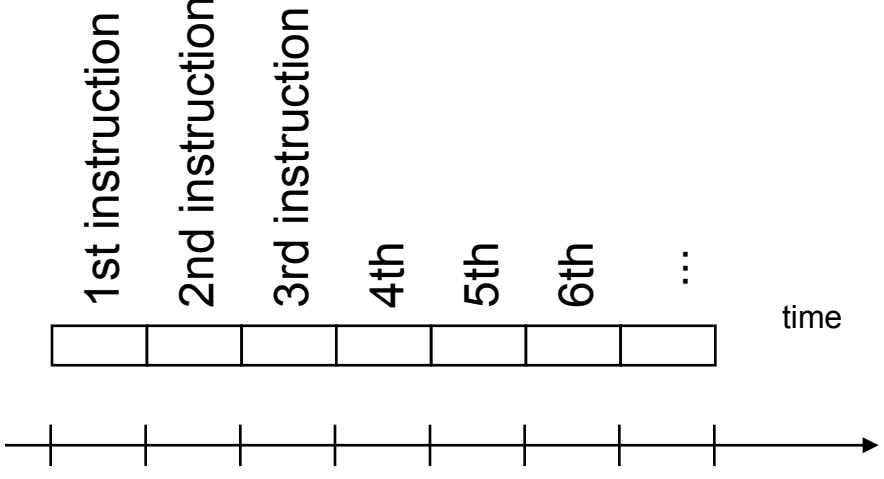
equivalently

CPU execution time for a program = CPU clock cycles for a program × Clock cycle time

- So, to improve performance one can either:
  - reduce the number of cycles for a program, or
  - reduce the clock cycle time, or, equivalently,
  - increase the clock rate

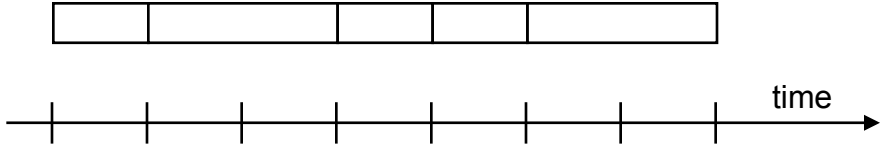
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



  - *This assumption is incorrect!* Because:
    - Different instructions take different amounts of time (cycles)
    - Why…?

How many cycles are required for a program?



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- *Important point*: changing the cycle time often changes the number of cycles required for various instructions because it means changing the hardware design. More later…

Example

- Our favorite program runs in 10 seconds on computer A, which has a 400Mhz. clock.
- We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.
- *What clock rate should we tell the designer to target?*

Terminology

- A given program will require:
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - *cycle time* (seconds per cycle)
  - *clock rate* (cycles per second)
  - (*average*) *CPI* (cycles per instruction)
    - a floating point intensive application might have a higher average CPI
  - *MIPS* (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance Measure

- *Performance is determined by execution time*

- Do any of these other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

- *Common pitfall* : thinking one of the variables is indicative of performance when it really isn't

Performance Equation II

CPU execution time for a program = Instruction count for a program × average CPI × Clock cycle time

- *Derive the above equation from Performance Equation I*

Questions?

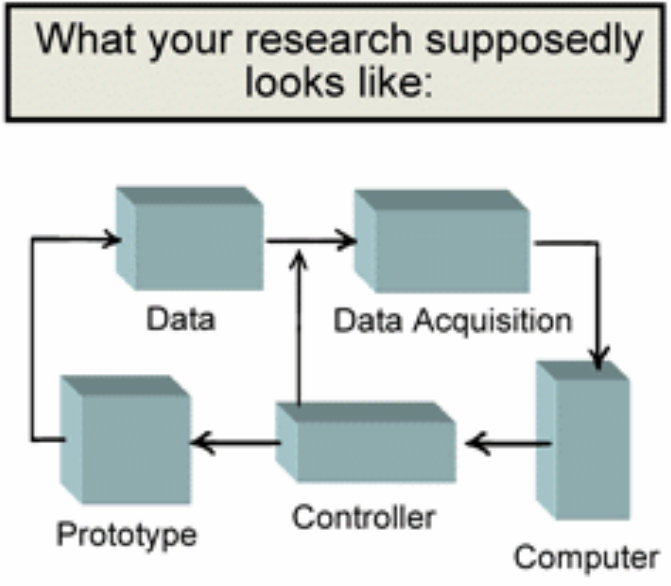


Figure 1. Experimental Diagram

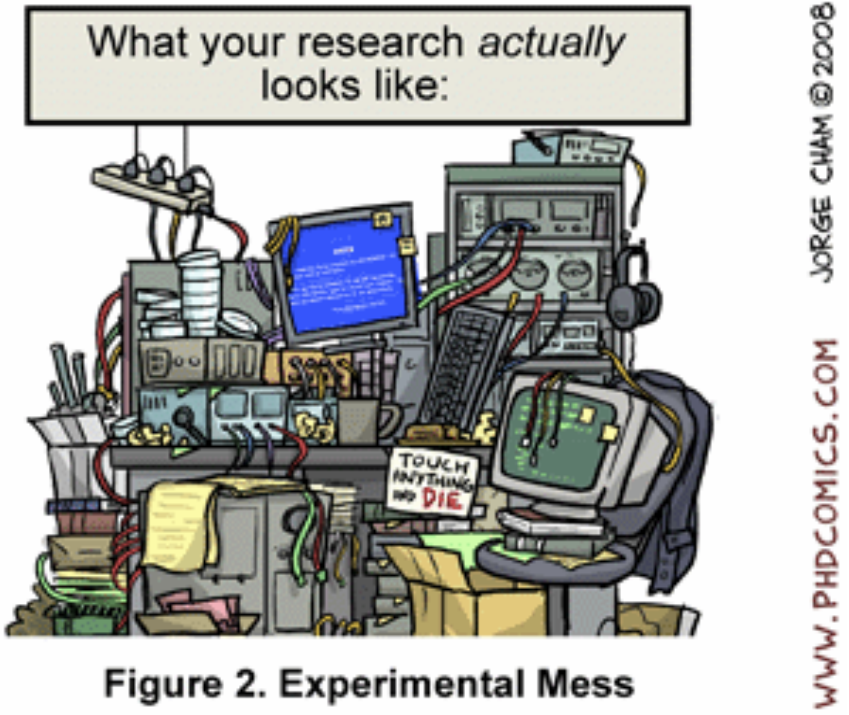


Figure 2. Experimental Mess

@carlosjaimebh