CAS CS 210 : COMPUTER SYSTEMS

Professor: Appavoo
CS:APP2e : Chapter 1

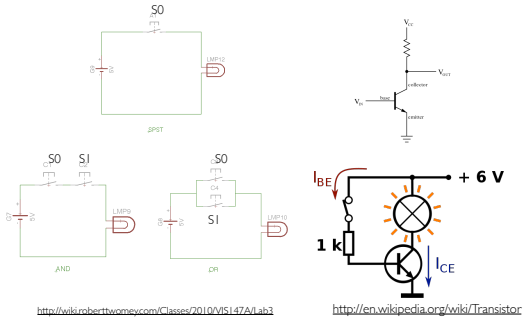
HARDWARE & SOFTWARE

OVERVIEW: HODGE-PODGE

INFORMATION IS BIT + CONTEXT

.....-.. .-.. ---.-----.-. .-.. -..

ELECTRICITY & TRANSISTORS



http://wiki.roberttwomey.com/Classes/2010/VIS147A/Lab3

http://en.wikipedia.org/wiki/Transistor

INTRODUCTION

- Binary Digit : Bit
- Groups of Bits + An Interpretation = Encoding
- Computers Store and manipulate information in ONLY in binary.
- Despite the fact that we want to work with numbers, letters, symbols, pictures, sound, audio, etc.
- We must represent the information by an encoding

Representations are not the same as the Information. You must know what the encoding/context is to tell what a binary value "mean"

THE BYTE — 8 BITS

8 switches

8 binary digits

7	6	5	4	3	2	1	0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

2 HEX(16 valued) digits

1	0
0-F	0-F

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

OFF,ON,OFF,OFF,ON,ON,OFF,ON

01001101

4D

EXAMPLES

A PROGRAM IS JUST BINARY TOO!!

THE VON NEUMANN MODEL

The archetype of the general-purpose digital computer

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

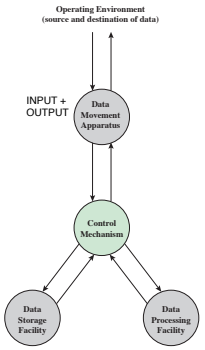
Between the

United States Army Ordnance Department

and the

University of Pennsylvania

https://sites.google.com/site/michaeldgodfrey/vonneumann/vnedvac.pdf?attredirects=0&d=1



THE VON NEUMANN MODEL

The archetype of the general-purpose digital computer

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

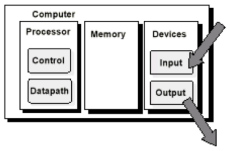
United States Army Ordnance Department

and the

University of Pennsylvania

https://sites.google.com/site/michaeldgodfrey/vonneumann/vnedvac.pdf?attredirects=0&d=1

Five Components



- Input
- Output
- Memory
- Datapath
- Control

THE COMPUTER

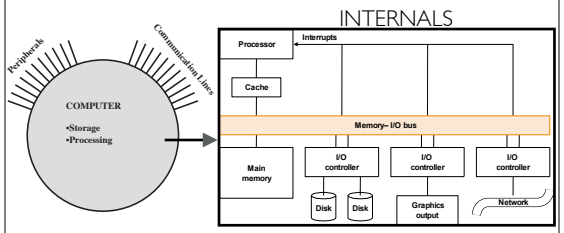


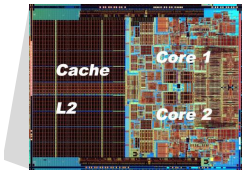
Figure 1.3 The Computer

Bus = control lines + data lines
Control lines carry requests, acknowledgements, type of information
Data lines carry data, complex commands, or addresses

THE PROCESSOR

Datapath and control
Datapath: ALU + registers
Implemented using millions of transistors
Impossible to understand by looking at each transistor

- **Machine Evolution**
  - 486 1989 1.9M
  - Pentium 1993 3.1M
  - Pentium/MMX 1997 4.5M
  - PentiumPro 1995 6.5M
  - Pentium III 1999 8.2M
  - Pentium 4 2001 42M
  - Core 2 Duo 2006 291M



THE MANUALS

http://www.cs.bu.edu/~jappavoo/Resources/210/extra/IA32_basic_architecture(24547007)-v1.pdf

http://www.cs.bu.edu/~jappavoo/Resources/210/extra/IA32_instruction_set_reference(24547107)-v2.pdf

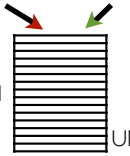
http://www.cs.bu.edu/~jappavoo/Resources/210/extra/IA32_system_programming_guide(24547207)-v3.pdf

THE STORED PROGRAM & THE LOOP!

STORED PROGRAM CONCEPT

PROGRAMS (INSTRUCTIONS)

DATA



BOTH STORED IN MEMORY

UNIFIED PROCESSING MODEL EMBODIED BY CPU -- INSTRUCTIONS JUST ANOTHER FORM OF DATA

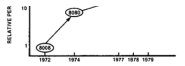


Figure 2-4. Relative Performance of the 8086 and 8088

2.2 Processor Architecture

Microprocessors generally execute a program by repeatedly cycling through the steps shown below (this description is somewhat simplified):

1. Fetch the next instruction from memory.
2. Read an operand (if required by the instruction).

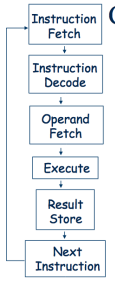
3. Execute the instruction.
4. Write the result (if required by the instruction).

In previous CPUs, most of these steps have been performed serially, or with only a single bus cycle fetch overlap. The architecture of the 8086 and 8088 CPUs, while performing the same steps, allocates them to two separate processing units within the CPU. The execution unit (EU) executes instructions; the bus interface unit (BIU) fetches instructions, reads operands and writes results.

The two units can operate independently of one another and are able, under most circumstances, to extensively overlap instruction fetch with execution. The result is that, in most cases, the time normally required to fetch instructions "disappears" because the EU executes instructions that have already been fetched by the BIU. Figure 2-5 illustrates this overlap and compares it with traditional microprocessor operation. In the example, overlapping reduces the elapsed time required to execute three instructions, and allows two additional instructions to be prefetched as well.

The processor causes the system to perform the desired operations by reading the first instruction in the program, and performing the very simple task dictated by the specific pattern of bits in this instruction (referred to as "executing" that instruction). It then goes on to the next instruction in the program and executes it. This simple operation of fetching an instruction and executing it is performed over and over, each time on the next instruction in sequence. In this way the program instructs the processor to bring about the desired system operation.

THE LOOP!
THE HEART OF MACHINE



CPU Execution Cycle

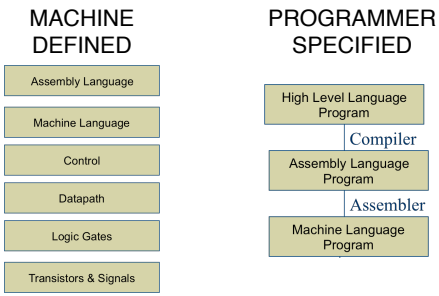
- Obtain instruction from program storage
- Determine required actions and instruction size
- Locate and obtain operand data
- Compute result value
- Deposit results in storage for later use
- Determine successor instruction

IMAGINE IF ALL SOFTWARE HAD TO BE WRITTEN AS MACHINE CODE

Effectively we would be mainly writing low-level control programs that shuffle bits between hardware devices and represent all data an code as raw binary numbers

WE NEED SOME ABSTRACTION LAYERS!

PROGRAM TRANSLATION



WHY "C"

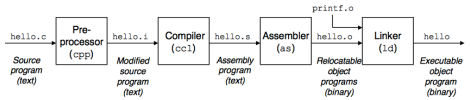
"In computing, **C** (/ˈsiː/, like the letter C) is a general-purpose programming language initially developed by Dennis Ritchie between 1969 and 1973 at Bell Labs.[4] Its design provides constructs that map efficiently to typical machine instructions, and therefore it found lasting use in applications that had formerly been coded in assembly language, most notably system software like the Unix computer operating system"

- Very little between you and the hardware.
- Power and Control: Allows you to more directly program/ control all aspects of the hardware.
- You can create new environments/machines for other programmers
- But what can happen when you play with fire?

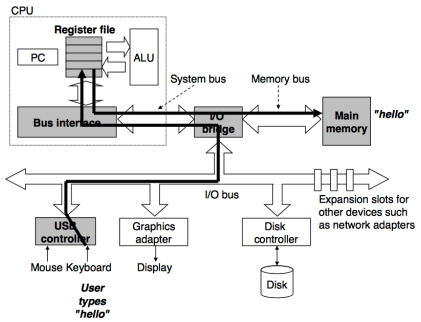
BEHIND THE CURTAINS

- What exactly is a program?
- How are they really constructed?
- With "C" we can more directly explore these things.

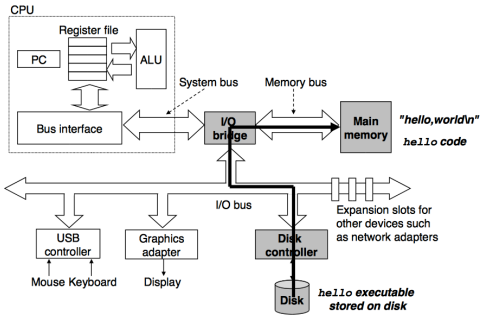
```
#include <stdio.h>

int main(void)
{
  printf("hello world!!!\n");
  return 1;
}
```



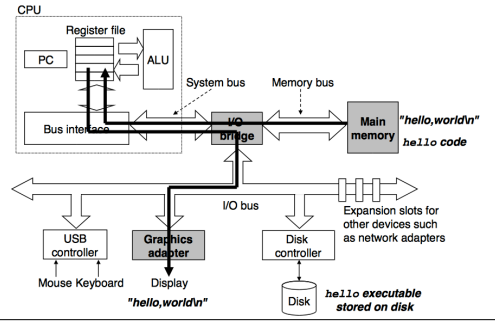
./HELLO: STARTS WITH IO



LOAD : MORE IO



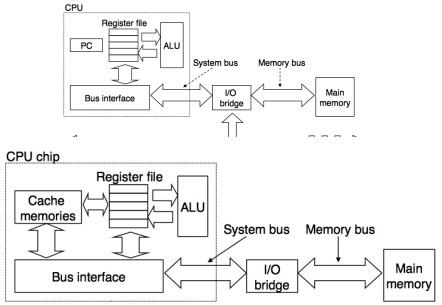
EXECUTE: AND MORE IO

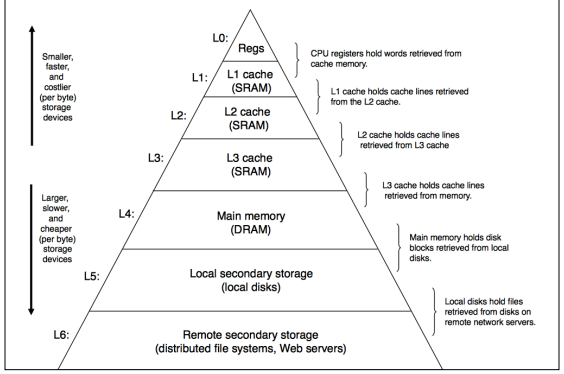


CACHES MATTER

MOVE IT!

COMMUNICATION TIME MATTERS : CACHES



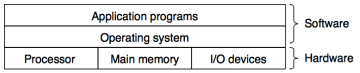


NEED MORE ABSTRACTIONS

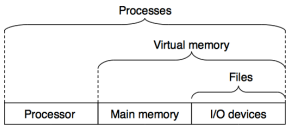
I/O AND RESOURCE MANAGEMENT

RUNTIME SOFTWARE LAYERS

OPERATING SYSTEMS (SYSTEMS SOFTWARE)

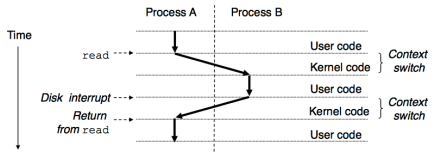


Layered Model

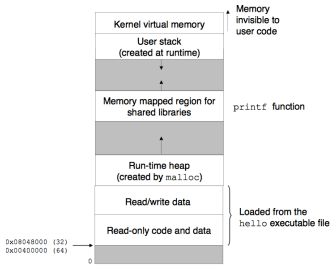


SYSTEM SOFTWARE CREATES ABSTRACTIONS

PROCESSES



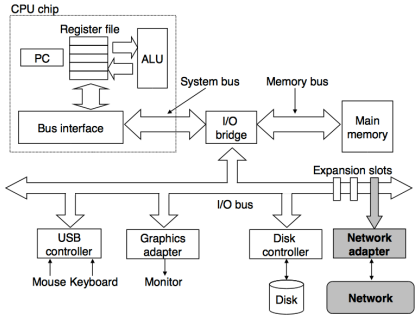
VIRTUAL MEMORY



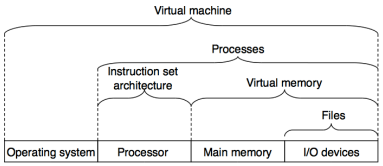
FILES

```
$ ls
hello   hello.c   hello.i  hello.o   hello.s
$ hexdump -C hello.c
00000000  23 69 6e 63 6c 75 64 65  20 3c 73 74 64 69 6f 2e  |#include <stdio.|
00000010  68 3e 0a 0a 69 6e 74 0a  6d 61 69 6e 28 69 6e 74  |h>..int.main(int|
00000020  20 61 72 67 63 2c 20 63  68 61 72 20 2a 2a 61 72  | argc, char **ar|
00000030  67 76 29 0a 7b 0a 20 20  20 70 72 69 6e 74 66 28  |gv).{.   printf(|
00000040  22 48 65 6c 6c 6f 20 57  6f 72 6c 64 21 21 21 5c  |"Hello World!!!\|
00000050  6e 22 29 3b 0a 20 20 20  72 65 74 75 72 6e 20 30  |n");.   return 0|
00000060  3b 0a 7d 0a                                       |;.}.|
00000064
```

NETWORKS AS IO DEVICE

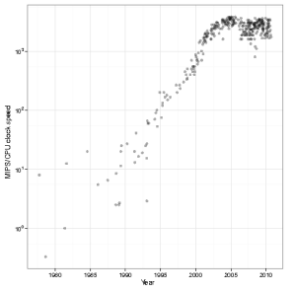


ABSTRACTIONS ON TOP OF ABSTRACTIONS



THE FREE LUNCH ENDED AWHILE AGO

MUST GO FASTER!!!



BUT HOW?

WHAT WAS THE MEGAHERTZ WAR?

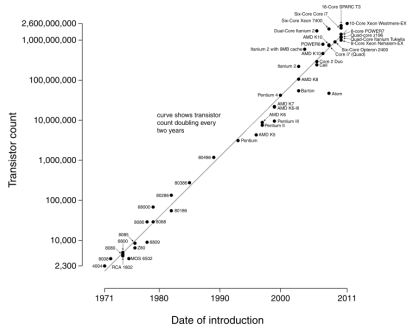
WHO WON?

MULTI-CORES & GPU's WHAT'S THE BIG DEAL?

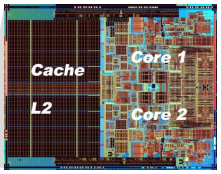
WHO LOOSES?

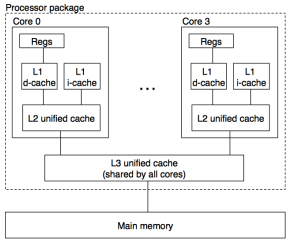
HMMMM

Microprocessor Transistor Counts 1971-2011 & Moore's Law



MUTLI-CORES

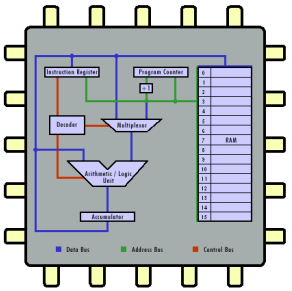




Intel Corei7 organization

A SIMPLE EXAMPLE OF THE UNDERLYING VIEW OF A COMPUTER

A COMPUTER



RAM : Memory locations

Registers: Instruction Register (IR), the Program Counter (PC), and the Accumulator.

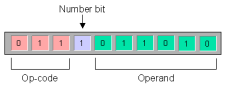
Buses: Address, Data, Control

ALU: Math and Logic operations

Control Unit: Decoder and the Multiplexor compose the Control Unit.

http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/Lesson.html

INSTRUCTION SET ARCHITECTURE



Op-code	*Mnemonic*	*Function*	*Example*
1	LOAD	Load the value of the operand into the Accumulator	LOAD 10
10	STORE	Store the value of the Accumulator at the address specified by the operand	STORE 8
11	ADD	Add the value of the operand to the Accumulator	ADD #5
100	SUB	Subtract the value of the operand from the Accumulator	SUB #1
101	EQUAL	If the value of the operand equals the value of the Accumulator, skip the next instruction	EQUAL #20
110	JUMP	Jump to a specified instruction by setting the Program Counter to the value of the operand	JUMP 6
111	HALT	Stop execution	HALT

A simple machine language
http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/Lesson.html

A PROGRAM — SUM

This program represents the formulas $x = 2$, $y = 5$, $x + y = z$ where the variables x, y, and z correspond with the memory locations 13, 14, and 15 respectively.

#	*Machine code*	*Assembly code*	*Description*
0	001 1 000010	LOAD #2	Load the value 2 into the Accumulator
1	010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
2	001 1 000101	LOAD #5	Load the value 5 into the Accumulator
3	010 0 001110	STORE 14	Store the value of the Accumulator in memory location 14
4	001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator
5	011 0 001110	ADD 14	Add the value of memory location 14 to the Accumulator
6	010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15
7	111 0 000000	HALT	Stop execution

Sum program
http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/Lesson.html

SEE SUM ANIMATION

http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/Lesson.html

NEXT CLASS

TUESDAY: Systems Programming and 'C' and Fundamentals of Boolean Logic and Gates

In addition to the 'C' readings assigned in the Syllabus:

The first one is just to help you recall basic knowledge and the second should be very short and I am only looking for basic comprehension

1. Review Truth Tables (eg. http://en.wikipedia.org/wiki/Truth_table you might find it informative/fun to play with the app here http://www.brian-borowski.com/software/truth/) — recall how to prove logical equivalence
2. Skim http://en.wikipedia.org/wiki/Logic_gate focus on the table in the "Symbols section"