COMPUTER ORGANIZATION (CSE 2021)

HUGH CHESSER
LAS 1012U

W1-W

Agenda

- Introduction to course
- Context
    - Hardware - Integrated Circuits (IC's)
    - Software – Assembly Language

Reading: Patterson, Sections 1.1 – 1.3.

CSE 2021: Computer Organization
Section E

Course URL:	https://wiki.cse.yorku.ca/course_archive/2012-13/F/2021/
Text:	D. A. Patterson and J. L. Hennessey, Computer Organization and Design, San Francisco, CA: Morgan Kaufmann Publishers, Inc., 4th edition (2009)
Class Schedule:	MW 17:30 – 19:00, LAS B
Office Hours:	Instructor: LAS 1012U, T, R 10 – 12 or by appointment Teaching Assistants: Mohammad Sajjadieh, Hong Ming Huai
Laboratory:	LAS 1006
Lab Schedule:	Lab 01 M, Lab 02 T, 19:00 – 22:00 – NOT every week – see course calendar
Lab Tools:	SPIM (QtSpim), Icarus Verilog, Crimson editor – all may be downloaded for free - see course web site for links to download
Assessment:	Quizzes: 12% (3 Quizzes @4% each) Lab Exercises: 32% (8 Labs A-D, K-N @4% each) Mid-term Exam: 20% Final Exam: 36%

Rough Course Schedule

- Two halves to the course:
- Software
- Hardware

WEEK #	WEEK OF	Mon	Wed	Lab	Approximate Lecture Schedule
1	Sep 03	-	□	-	Overview of the course
2	Sep 10	□	□	-	Performance and Data Translation
3	Sep 17	□	□	A	Code Translation
4	Sep 24	□	Quiz #1	B	Translating Utility Classes
5	Oct 01	□	□	C	Translating Objects
6	Oct 08	-[1)]	Mid-term[2)]	-	-
7	Oct 15	□	□	D	Introduction to Hardware
8	Oct 22	□	□	Make-up Labs	Machine Language + Floating-Point
9	Oct 29	□	-[3)]	K	The CPU Datapath
10	Nov 05	□	Quiz #2	L	The Single-Cycle Control
11	Nov 12	□	□	M	Pipelining
12	Nov 19	□	□	N	Caches
13	Nov 26	□	Quiz #3	Make-up Labs	
14	Dec 03	□	-	-	No lecture on Wednesday
Exam Period	Dec 5 - 21	-	-	-	There will be a final exam held during the exam period - TBA.

[1)] Oct 8 - No class - Thanksgiving

[2)] Mid-term is Wed Oct 10 in LAS B.

[3)] Co-curricular days - Oct. 31 - Nov. 4

Labs

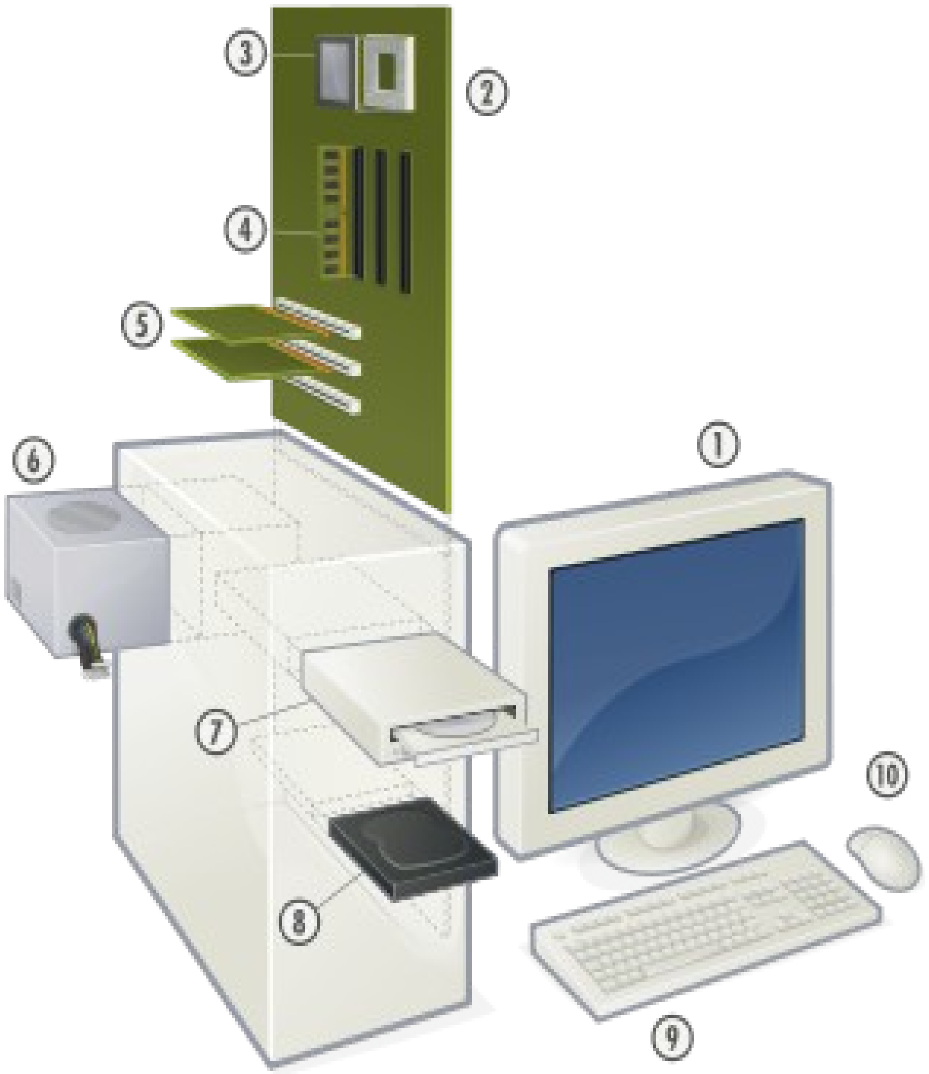
- Ensure you have a CSE account!
- M, T 19:00 – 22:00 LAS 1006
- Each lab contains useful practice exercises that are to be submitted (see Lab A instructions for details)
- At around 20:30 – a lab exercise will be given to each student to be completed INDEPENDENTLY and submitted within 75 minutes. Only aids are the submitted practice material.
- More details next week

Computer Hardware Architecture – High Level

Hardware Elements: Computer, Monitor, Keyboard, Mouse, Network, …

The components and how they interconnect can be said to constitute a level of abstraction

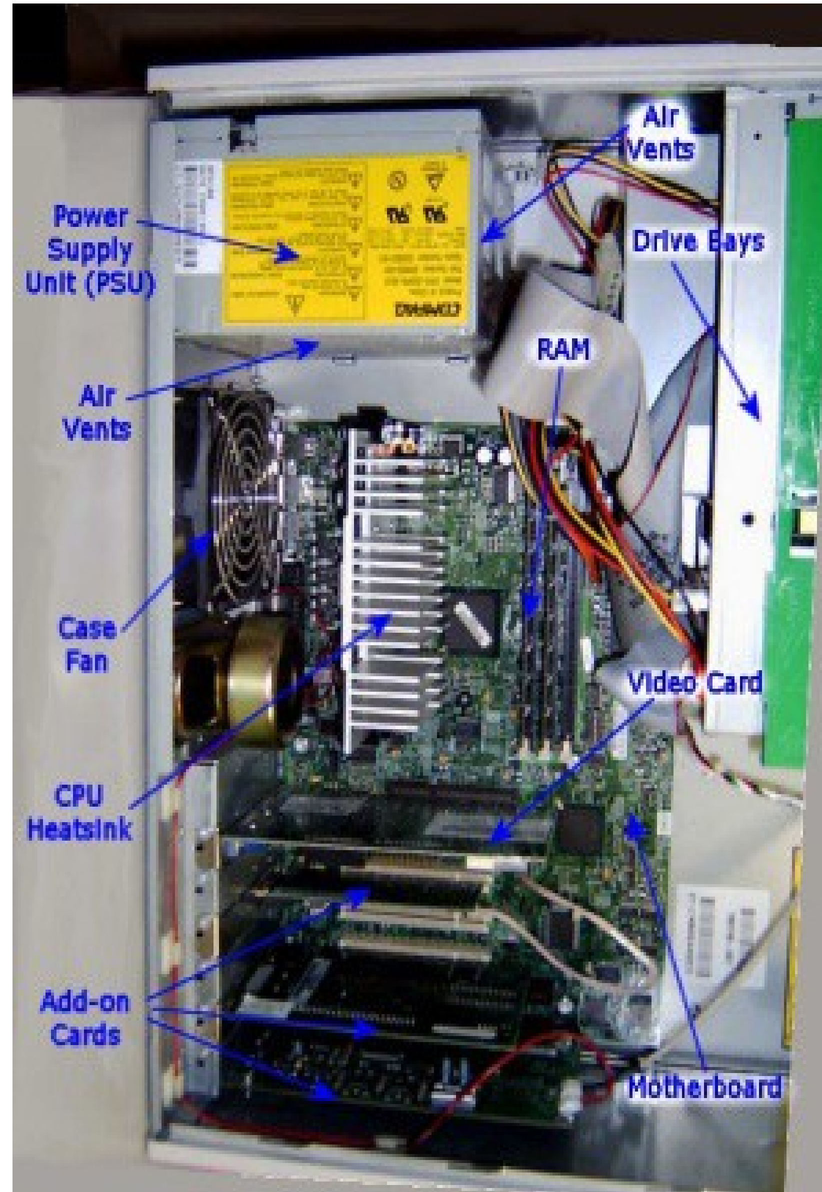
Our course will deal with a lower level of abstraction of the computer than shown here



Inside a PC – High Level Computer Architecture

PC consists of a motherboard (CPU, onboard memory, i/o devices), hard disk, floppy drives, power supply, and connectors.

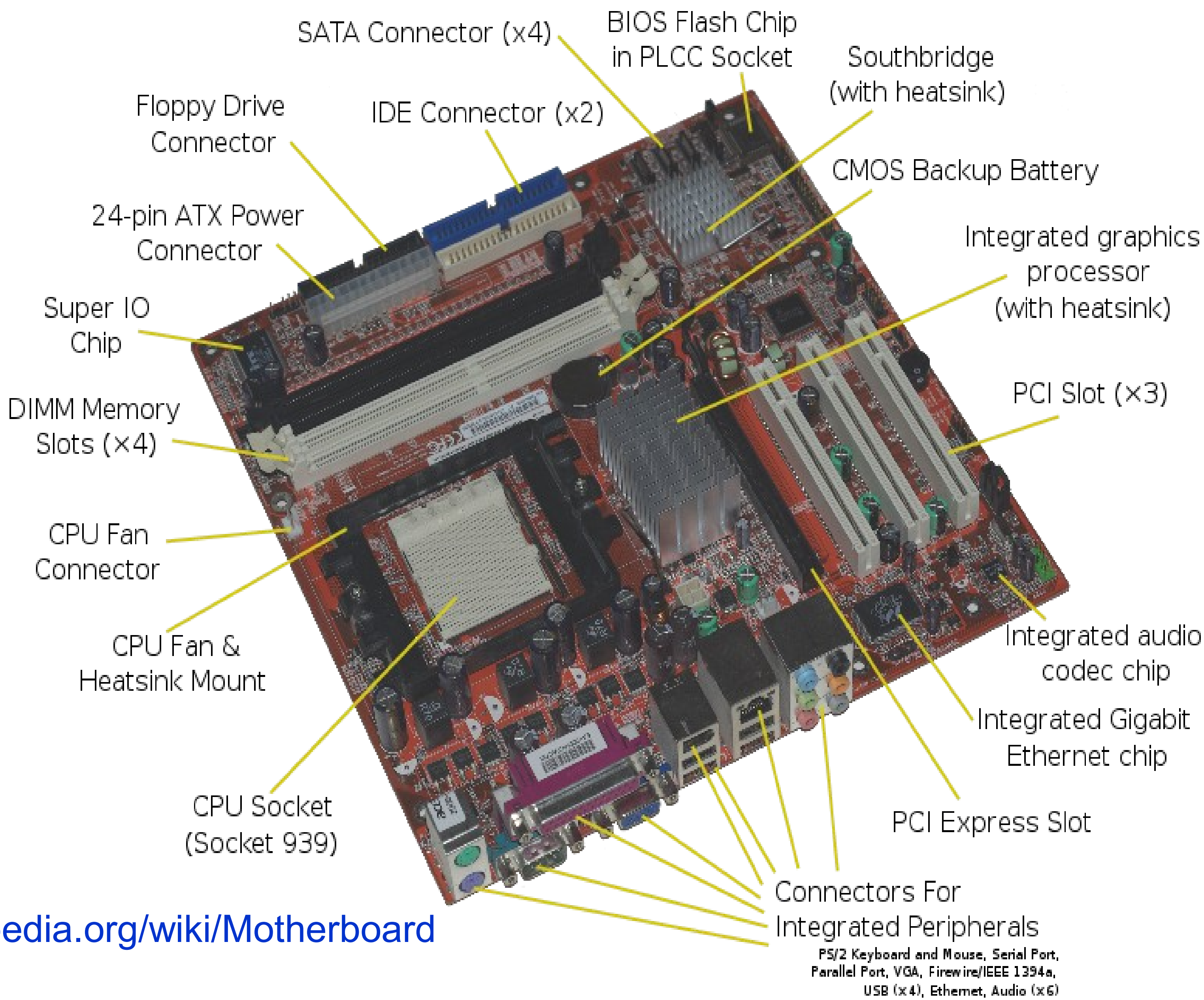
...again we'll be talking about a lower level of abstraction in the course...



Inside a PC: Motherboard

PC Motherboard consists of a central processing unit (CPU), typically PCI card slots, Dynamic random access memory (DRAM), and connectors for I/O devices

..even a lower level of abstraction...

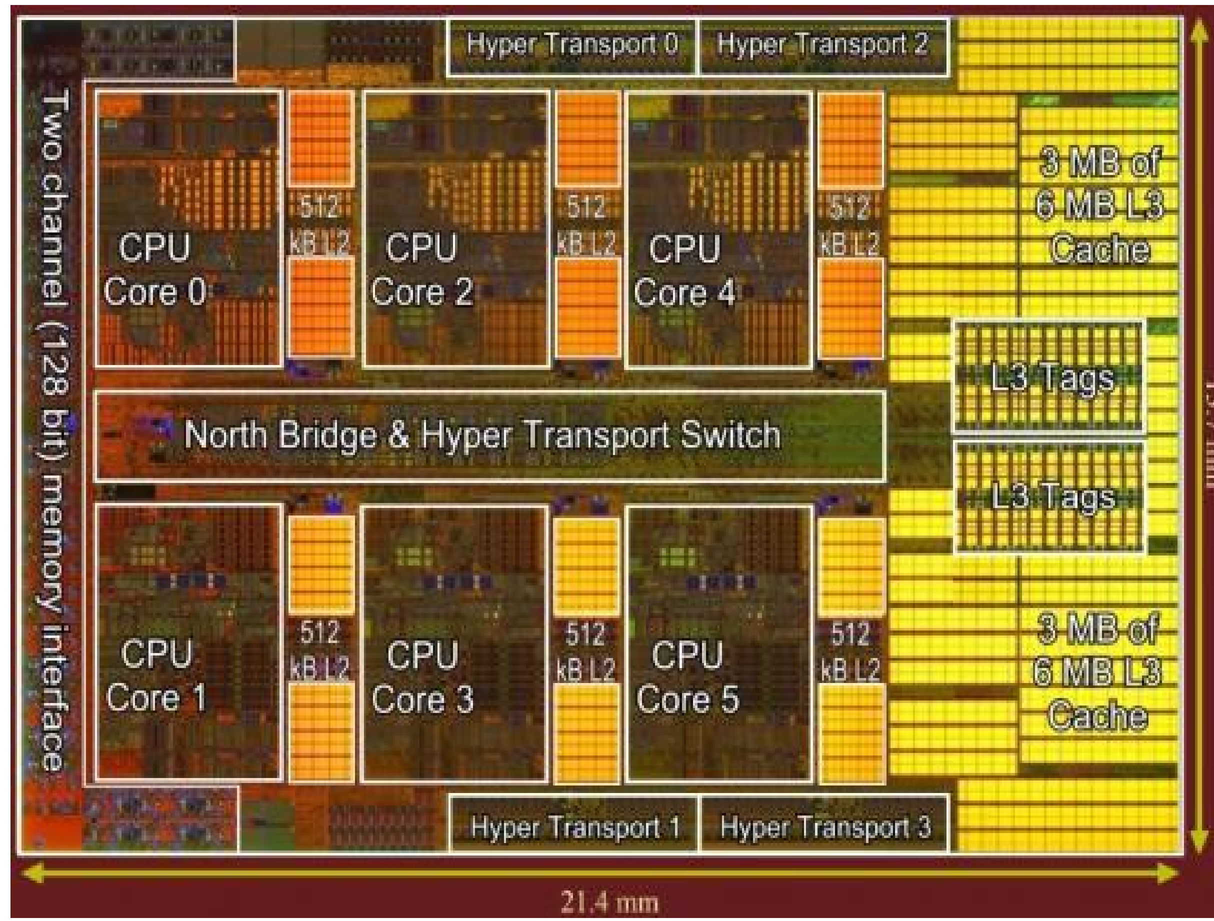


http://en.wikipedia.org/wiki/Motherboard

Inside a PC: CPU

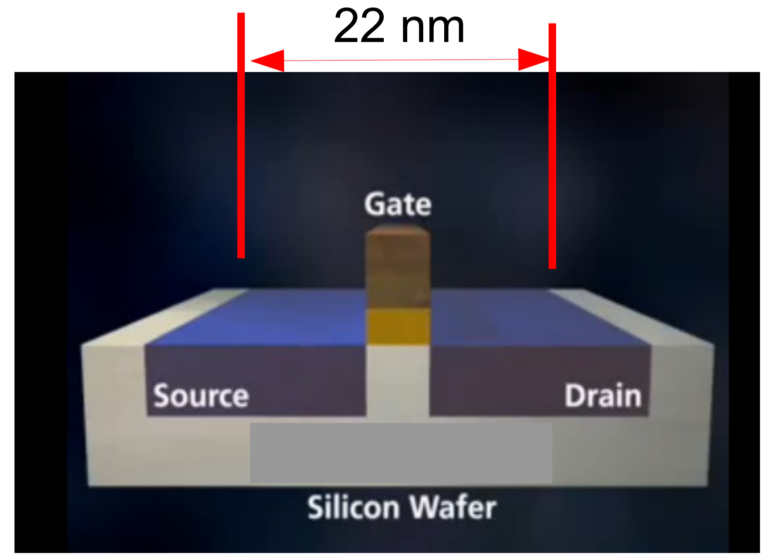
CPU comprises of two main components:

1. Datapath: consists of Data and instruction cache, Bus, and integer and floating point data path. The latter performs integer and floating point arithmetic operations
2. Control: tells the datapath memory and I/O devices what to do based on the program



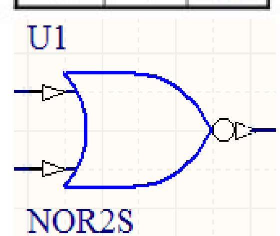
FET Transistor

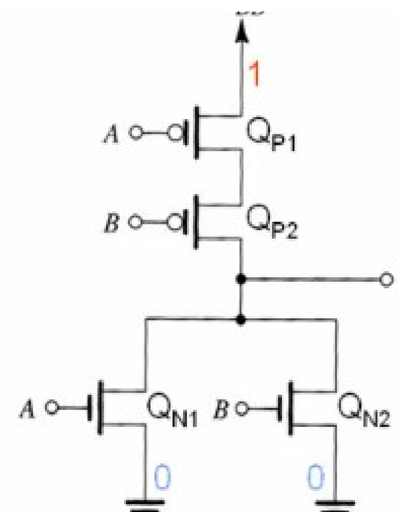
- Basic element of any digital circuit
- Logic gates (and, or, not, etc) are made from FETs
- Memory locations
- Our course deals with hardware at a higher level of abstraction – gates vs. transistors



Truth Table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0





Source: http://www.youtube.com/watch?v=J8ZPIDNaijs&list=PL46840EB0E725FB91&index=1&feature=plpp

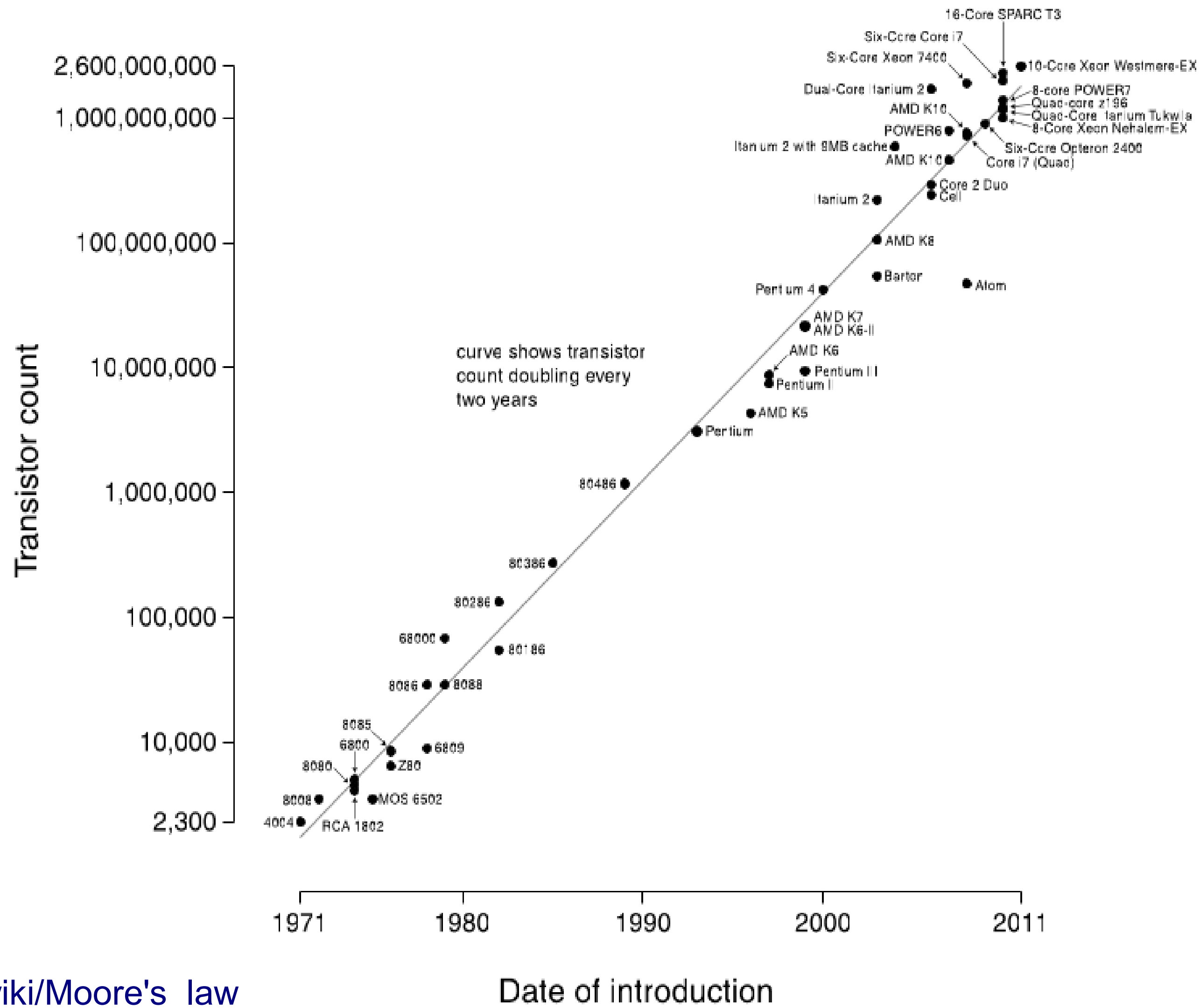
wl-w

Innovation (2)

Microprocessor Transistor Counts 1971-2011 & Moore's Law

Moore's Law:

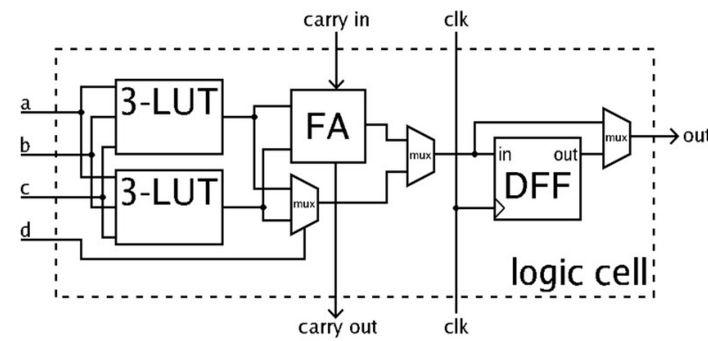
“The transistor density on integrated circuits doubles every couple of years.” This exponential growth and ever-shrinking transistor size has resulted in increased performance with decreased cost.



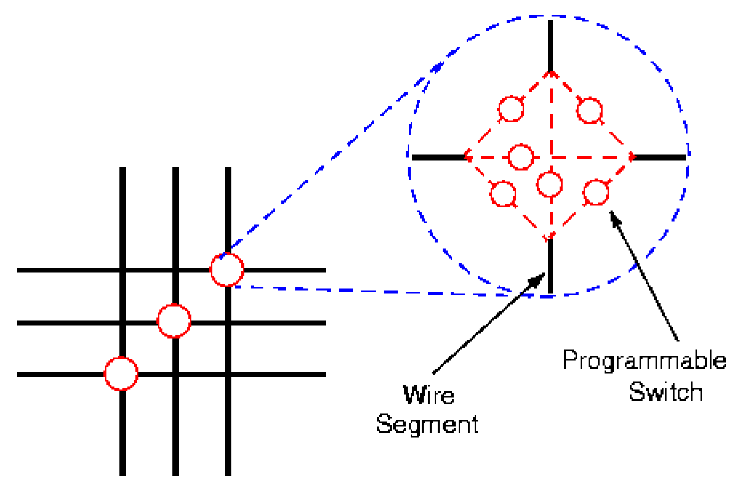
Source:
http://en.wikipedia.org/wiki/Moore's_law

Field Programmable Gate Arrays

- We will be learning “Verilog” to describe the behaviour of a desired logic circuit – an example of a Hardware Description Language (HDL)
- Technology behind “Soft processing” and “System on a Chip” (SOC)
- A type of Integrated Circuit that the user can configure or define
- Subtly different than traditional programming - more on this later



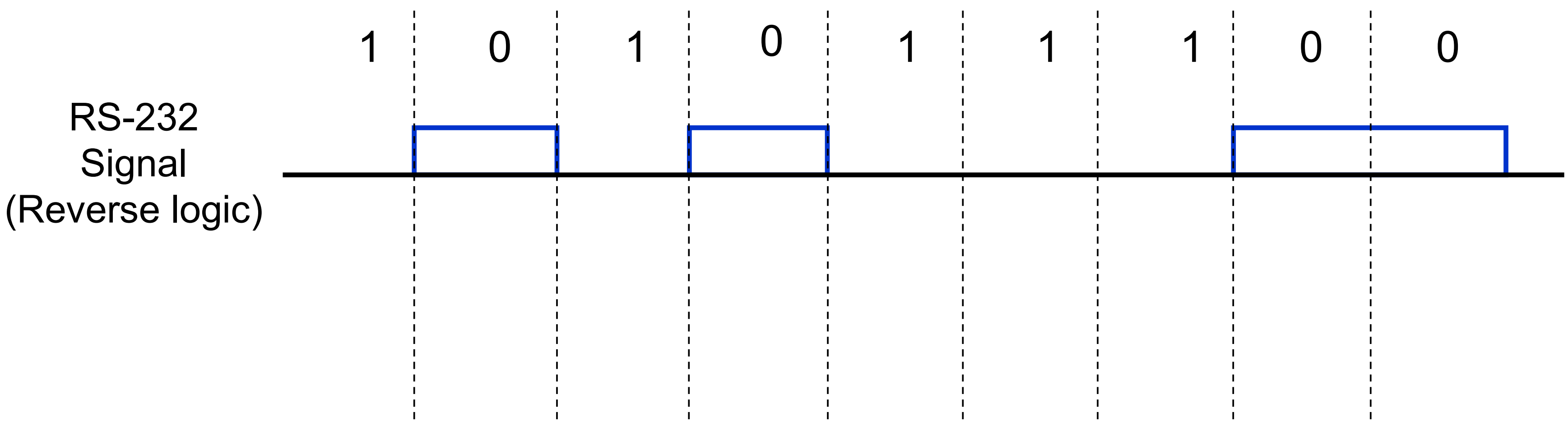
Logic Block



Interconnects

Binary Digits (Bits)

- Communication between different components of a computer takes place in terms of *on* and *off* electrical signals
- Symbols used to represent these electrical states are the numbers 1 and 0; binary digit 1 corresponding to high voltage and binary digit 0 corresponding to low voltage



- All operations and data inside a computer are expressed in terms of the binary digits or bits

Instructions

- Instructions: are commands given to a computer to perform a particular task.

  Example: *Addition of variables A and B*

  High Level Language: `(A + B)`

  Binary notation for the add operation: `100011001010000`
- Binary machine language program: is a one-to-one binary representation of a program written in a high level language.
- Clearly, binary machine language programs are tedious to write and debug.
- Instead a symbolic notation is used as an intermediate step between the high level language and its binary representation. This symbolic notation is referred to as the assembly language.

  Example: *Addition of variables A and B*

  High Level Language: `(A + B)`

  Assembly Language: `add A,B`

  Binary notation for the add operation: `100011001010000`

Levels of Programming

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp,
}
```

C compiler

Why use High-level Language?

1. Ease in writing & debugging
2. Improved productivity
3. HW independence

Assembly language program (for MIPS)

```
swap:
     muli $2, $5,4
     add  $2, $4,$2
     lw   $15, 0($2)
     lw   $16, 4($2)
     sw   $16, 0($2)
       sw   $15, 4($2)
       jr   $31
```

Compiler: converts a program written in high-level language into its equivalent symbolic assembly language representation.

Assembler

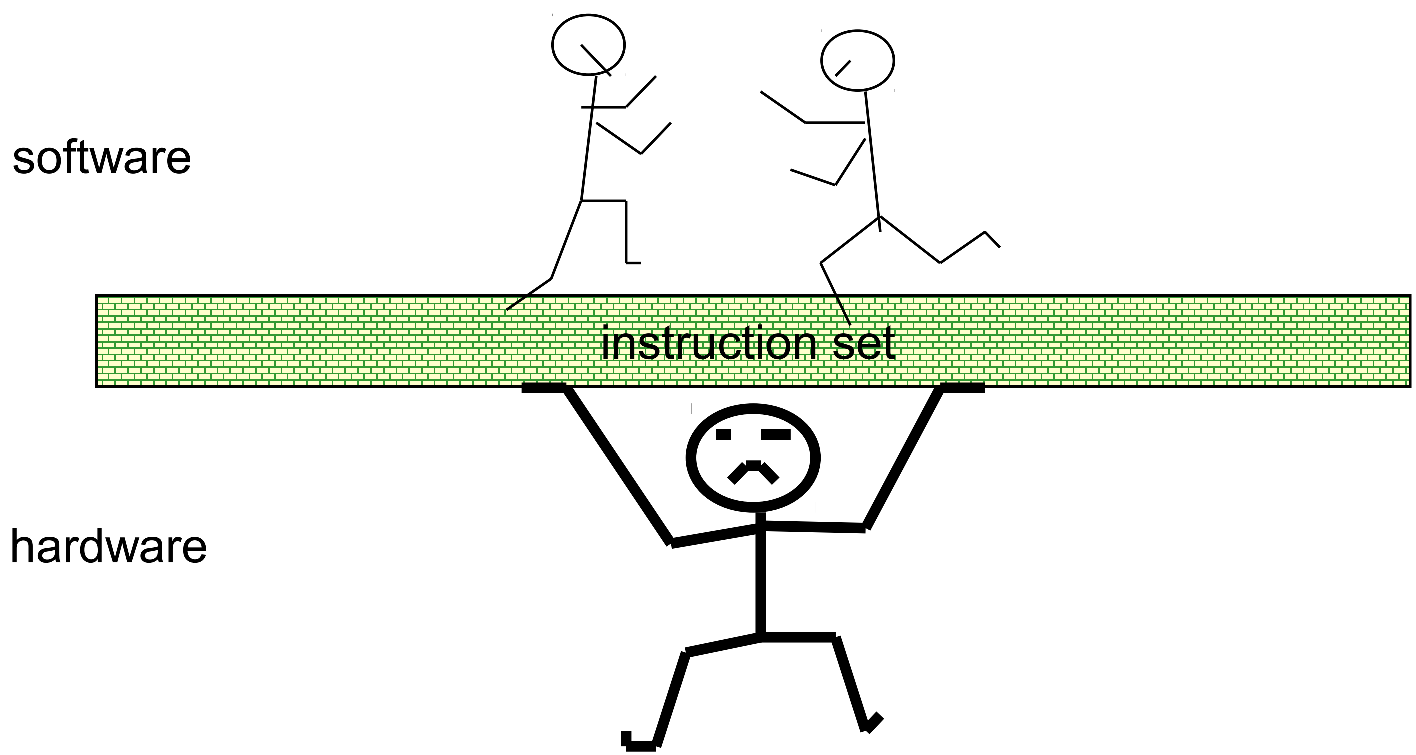
Assembler: translates assembly language into the binary machine language.

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instruction Set (1)

- Computer Architecture = Instruction Set Architecture + Machine Organization
- Machine Organization: Ways in which different computer components (Registers, ALU, Shifters, Logic Units, ...) are interconnected.
- Recall instructions are commands given to a computer to perform a particular task.
- Instruction Set: is a collection / library of instructions that a computer can execute.



Instruction Set (2)

- Programs written for a computer can only use the instructions provided in its instruction set.
- Examples of modern instruction set architectures (ISA's):
  1. 80x86/Pentium/K6/MMX (Intel, 1978-96)
  2. Motorola 68K (Motorola, late 1980,s, early 1990's)
  3. MIPS (SGI, 1986-96) I, II, III, IV, V
  4. SPARC (Sun, 1987-95) v8, v9
  5. ARM (ARM Ltd, 1992-96) ARMv6, ARMv7TDMI
- Instructions in the MIPS instruction set can be divided in five categories:
  1. Arithmetic operations: `add,sub(subtract),mult(multiply),div (division),etc.`
  2. Logical operations: `and, or, sll (shift left logical), etc.`
  3. Data Transfer: `lw (load), sw (save), etc.`
  4. Conditional branch: `beq(branch if equal), slt set if less than), etc.`
  5. Unconditional branch: `j (jump), etc.`

Question: Will the ISA developed on one machine be compatible with another machine of a different manufacturer?

Where are we headed?

- Performance issues (Chapter 1.4 – 1.8)   *vocabulary and motivation*
- MIPS instruction set architecture (Chapter 2)
- Arithmetic and how to build an ALU (Chapter 3)
- Constructing a processor to execute our instructions (Chapter 4)
- Pipelining to improve performance (Chapter 4)
- Memory: caches and virtual memory (Chapter 5)