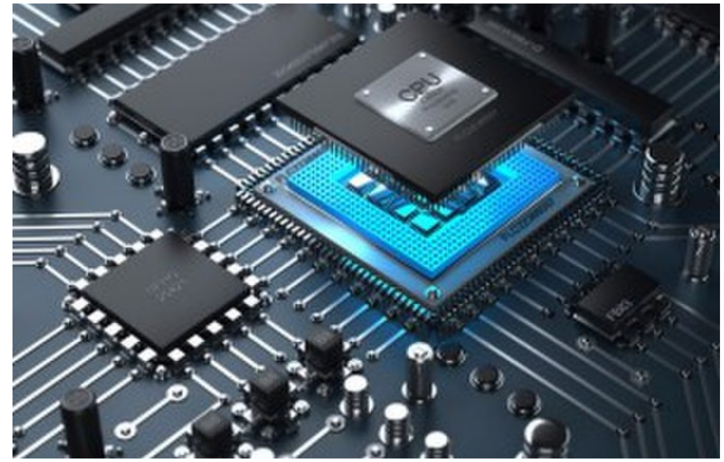


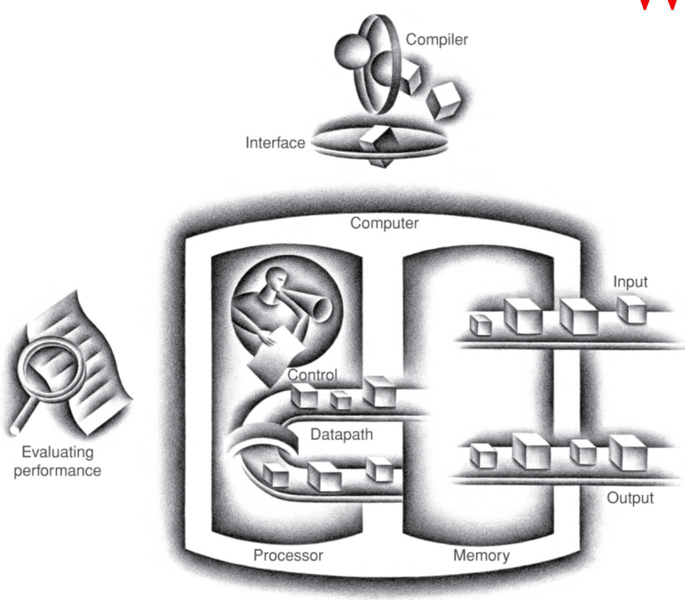
Computer Abstractions and Technology

CS 154: Computer Architecture

Lecture #2

Winter 2020

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB



A Word About Registration for CS154

FOR THOSE OF YOU NOT YET REGISTERED:

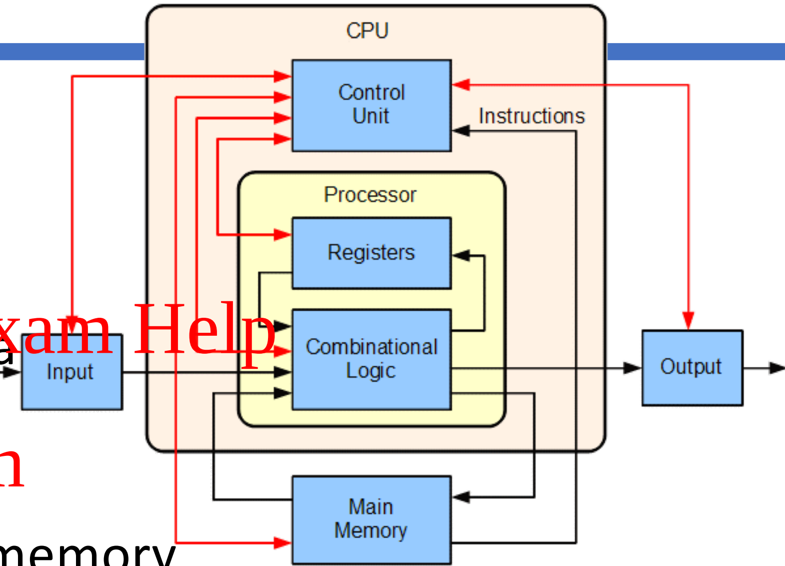
- This class is **FULL**

- **If you want to add this class AND you are on the waitlist, see me after lecture**

Lecture Outline

- Tech Details
  - Trends
  - Historical context
  - The manufacturing process of Ics
- Important Performance Measures
  - CPU time
  - CPI
  - Other factors (power, multiprocessors)
  - Pitfalls

Parts of the CPU



- The **Datapath**, which includes the **Arithmetic Logic Unit** (ALU) and other items that perform operations on data

- **Cache Memory**, which is small & fast RAM memory for immediate access to data. Resides inside the CPU. (other types of memory are outside the CPU, like DRAM, etc…)

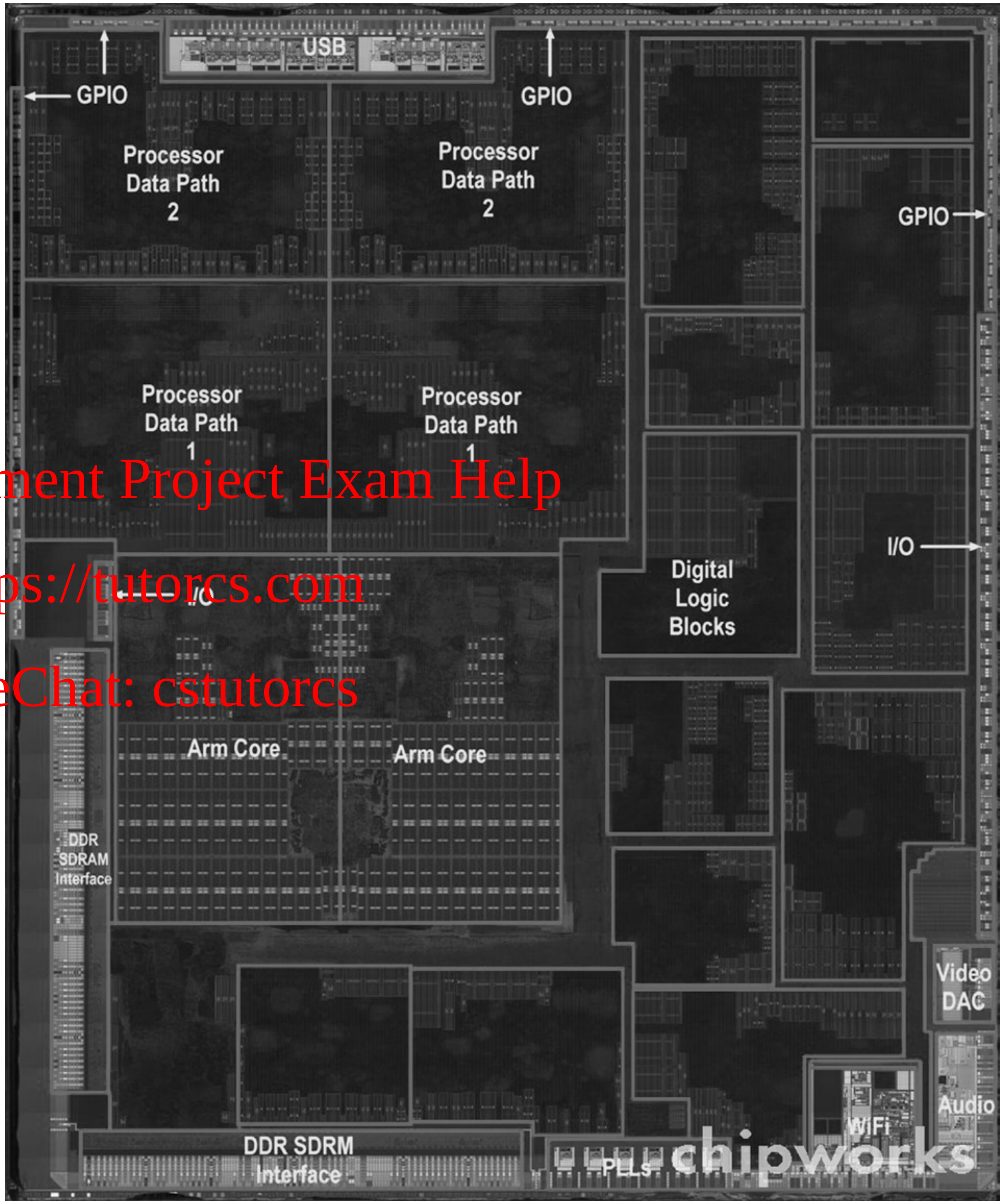
- The **Control Unit** (CU)

  which sequences how Datapath + Memory interact

Image from wikimedia.org

Inside the Apple A5 Processor

Manufactured in 2011 – 2013
32 nm technology
37.8 mm^2 die size



The CPU's Fetch-Execute Cycle

- **Fetch** the next instruction
- **Decode** the instruction
- **Get data** if needed
- **Execute** the instruction
  - Maybe access mem again and/or write back to reg.

This is what happens inside a computer interacting with a program at the "lowest" level

Pipelining (Parallelism) in CPUs

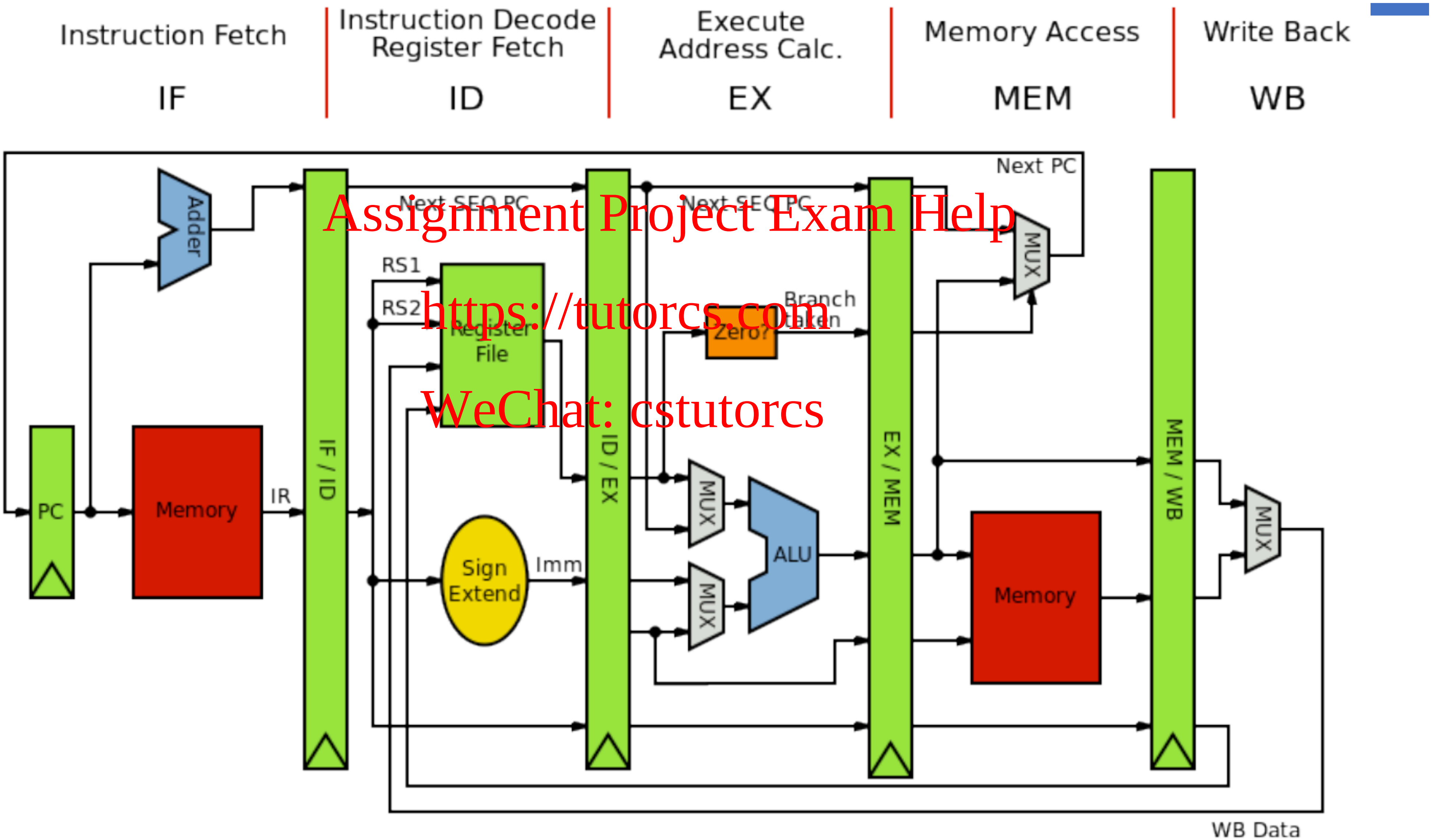
- Pipelining is a fundamental design in CPUs
- Allows multiple instructions to go on at once
  - a.k.a instruction-level parallelism

Basic five-stage pipeline

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

Digital Design of a CPU (Showing Pipelining)



Computer Languages and the F-E Cycle

- Instructions get executed in the CPU in machine language (i.e. all in “1”s and “0”s)

- Even *small* instructions, like

“add 2 to 3 then multiply by 4”,

need *multiple* cycles of the CPU to get fully executed

- But **THAT’S OK!** Because, typically,
  CPUs can run *many millions* of instructions per second

Computer Languages and the F-E Cycle

- But **THAT'S OK!** Because, typically,
  CPUs can run *many millions* of instructions per second

- In *low-level languages* (like assembly or machine lang.) you need to spell those parts of the cycles one at a time

- In *high-level languages* (like C, Python, Java, etc…) you don't
  - 1 HLL statement, like "*x = c*(a + b)*" is enough to get the job done
  - This would translate into multiple statements in LLLs
  - **What translates HLL to LLL?**
  - **What translates LLL to ML?**

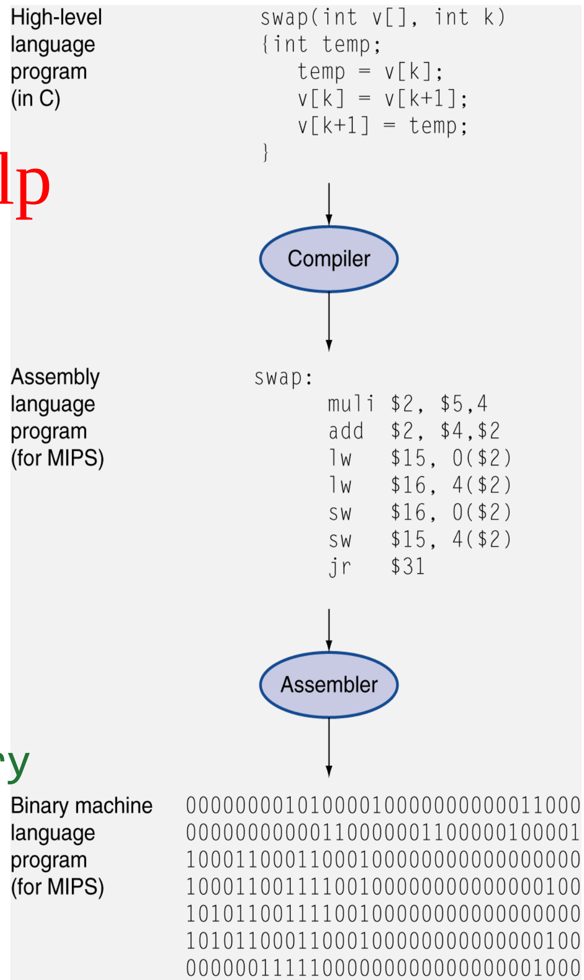
Machine vs. Assembly Language

- **Machine language (ML)** is the actual 1s and 0s

Example:

`10111101110111000001010101000`

- **Assembly language** is one step above ML
  - Instructions are given **mnemonic codes** but still displayed one step at a time
  - Advantage? Better human readability



Example:

```
lw   $t0, 4($sp)     # fetch N from someplace in memory
add  $t0, $t0, $t0   # add N to itself
                     # and store the result in N
```

Computer Memory

<u>Usually organized in two parts:</u>

- **Address**: ***Where*** can I find my data?
- **Data** (payload): ***What*** is my data?

<u>Recall:</u>

- A bit (b) is ______________________
- A byte (B) is ______________________
- MIPS CPUs operate ***instructions*** that are ____ bits long
- MIPS CPUs organize ***memory*** in units called _______ that are ___ bits long
- MIPS memory is addressable in _________ ***endian***

Reminder of some MIPS instructions

- **add** vs **addi** vs **addu** vs **addui**
- **mult** and **mflo**
- **sll**
- **srl** vs **sra**
- **la** vs **li** vs **lw** vs **sw**

Eight Great Ideas in Computer Architecture

- Design for Moore's Law
- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

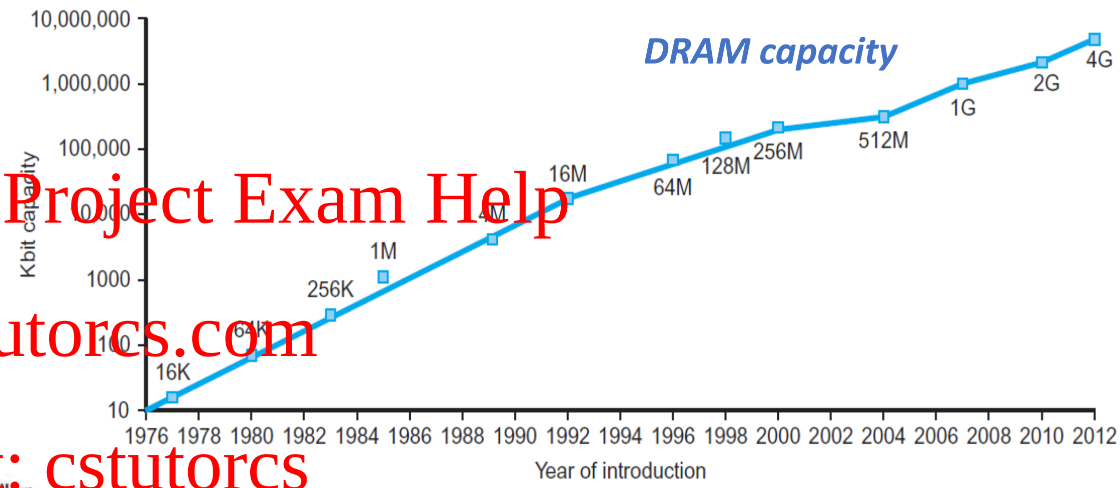
Electronic Circuitry Tech Trends

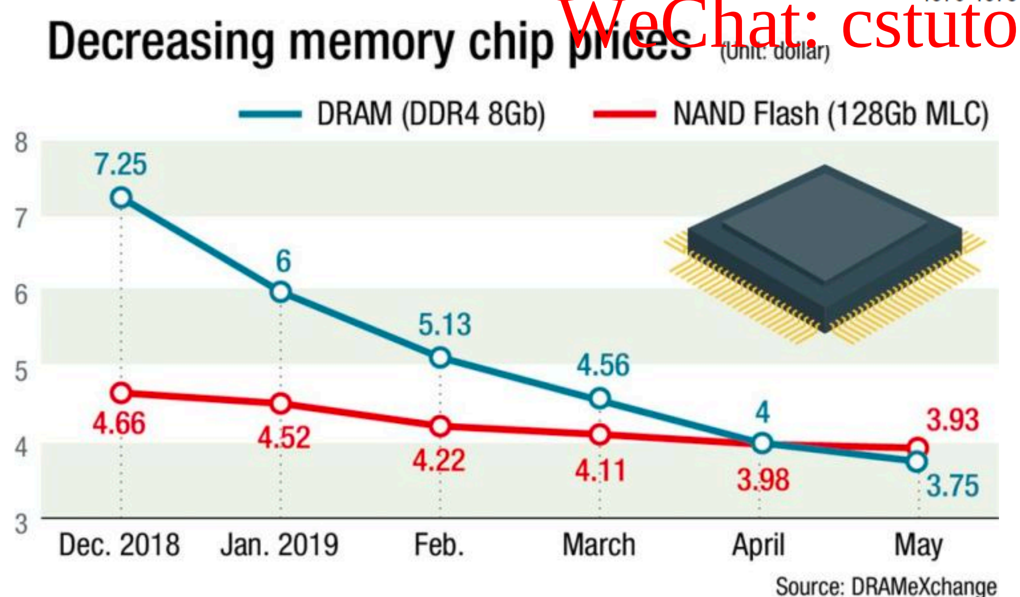
- Electronics technology continues to evolve
  - Increased memory capacity (at same price/size)
  - Increased CPU performance
  - Reduced costs overall

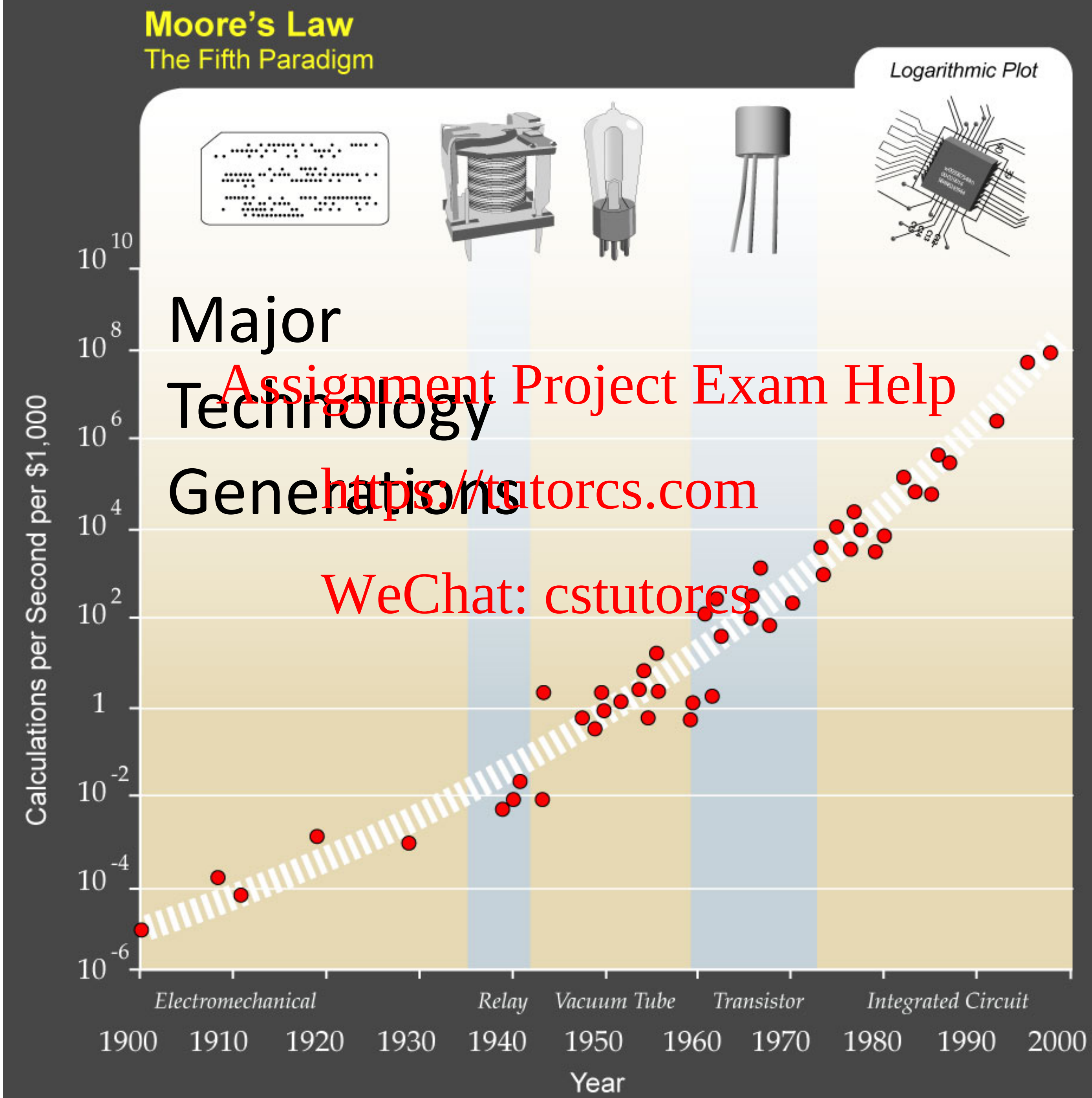
Year	Technology	Relative Performance
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2.4 million
2013	Application Specific IC or ASIC (ultra-large scale)	250 million

DRAM capacity goes up and the prices come down…

- DRAM = Dynamic RAM
- Very common tech used for computer memory





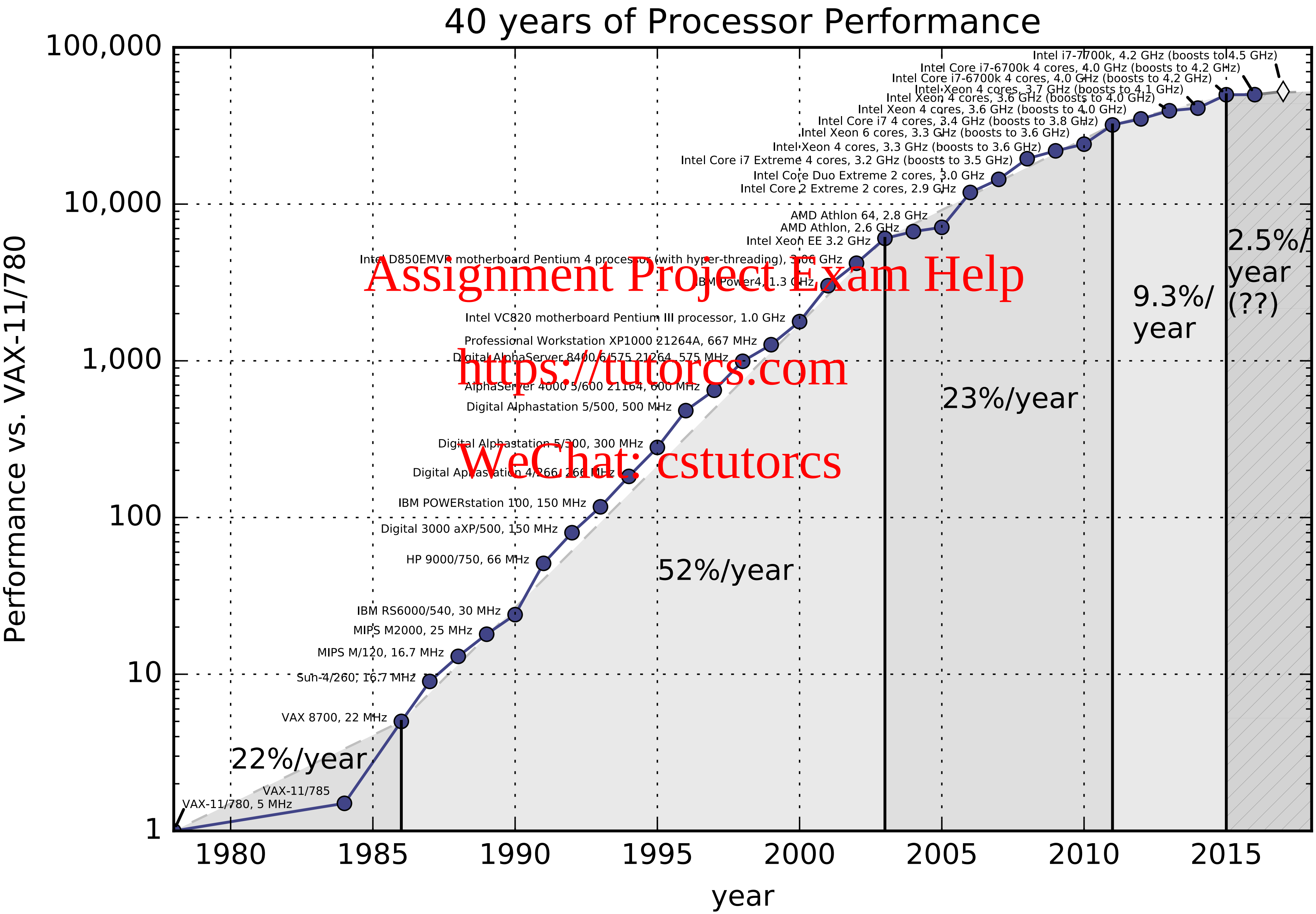


Major Technology Generations

?

Source: K.Asanovic, UCB

Single-Thread Processor Performance



[Hennessy & Patterson, 2017]

Computer Architecture: A Little History

Throughout the course we'll use a historical narrative to help understand why certain ideas arose

Why worry about old ideas?

- Helps to illustrate the design process, and explains why certain decisions were taken
- Because future technologies might be as constrained as older ones
- Those who ignore history are doomed to repeat it
  - Every mistake made in mainframe design was also made in minicomputers, then microcomputers, where next?

Digital Computers

- An improvement over Analog Computers…

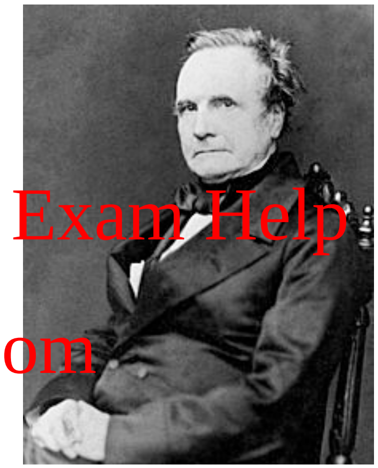
- Represent problem variables as numbers encoded using discrete steps
  - Discrete steps provide noise immunity

- Enables accurate and deterministic calculations
  - Same inputs give same outputs exactly

Computing Devices for General Purposes

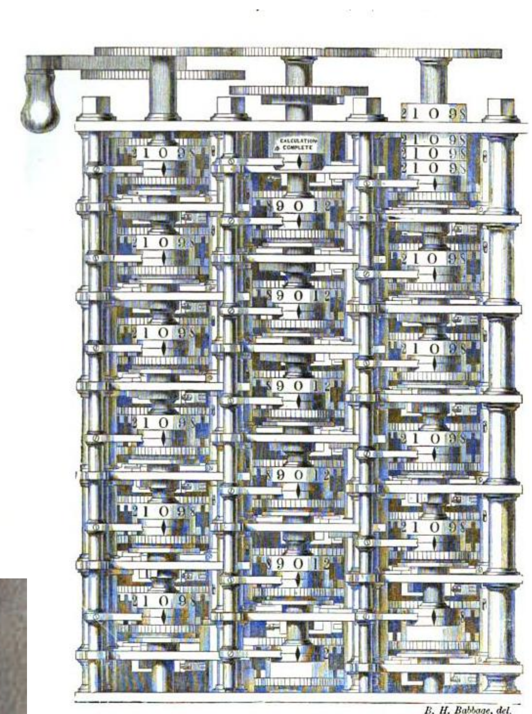
- **Charles Babbage (UK)**
  - *Analytical Engine* could calculate polynomial functions and differentials
  - Inspired by older generation of calculating machines made by Blaise Pascal (1623-1662, France)
  - Calculated results, but also *stored intermediate findings* (i.e. precursor to computer memory)
  - “Father of Computer Engineering”

- **Ada Byron Lovelace (UK)**
  - Worked with Babbage and foresaw computers doing much more than calculating numbers
  - Loops and Conditional Branching
  - “Mother of Computer Programming”



C. Babbage (1791 – 1871)

A. Byron Lovelace (1815 – 1852)



Part of Babbage’s Analytical Engine

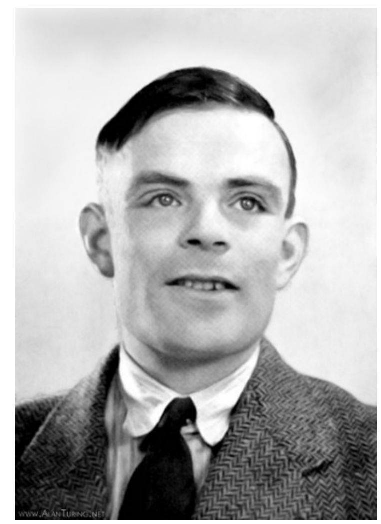
Images from Wikimedia.org

The Modern Digital Computer

- Calculating machines kept being produced in the early 20th century (IBM was established in the US in 1911)
- Instructions were very simple, which made hardware implementation easier, but this hindered the creation of complex programs.

Alan Turing (UK)

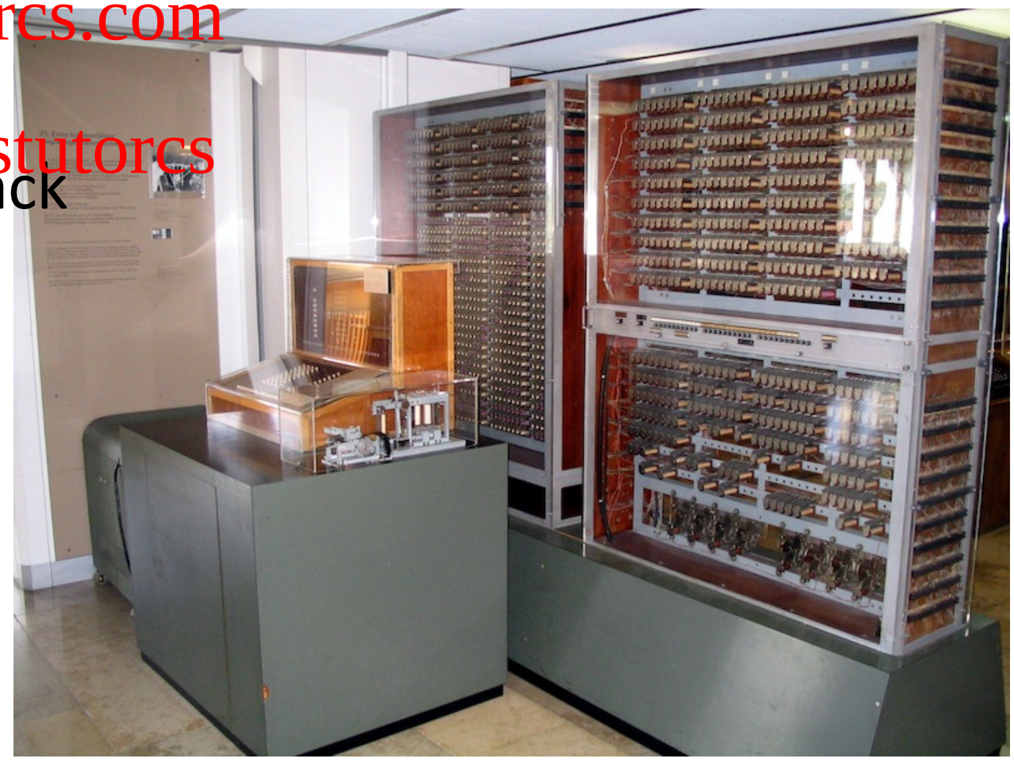
- Theorized the possibility of computing machines capable of performing *any* conceivable mathematical computation as long as this was representable as an *algorithm*
  - Called "*Turing Machines*" (1936) – ideas live on today…
  - Lead the effort to create a machine to successfully decipher the German "Enigma Code" during World War II



A. Turing (1912 – 1954)

Zuse Z3 (1941)

- Built by Konrad Zuse in wartime Germany using 2000 relays
- Could do *floating-point* arithmetic with hardware
- 22-bit word length; clock frequency of about 4–5 Hz!!
- 64 words of memory!!!
- Two-stage pipeline
  1) fetch & execute, 2) writeback
- No conditional branch
- Programmed via paper tape

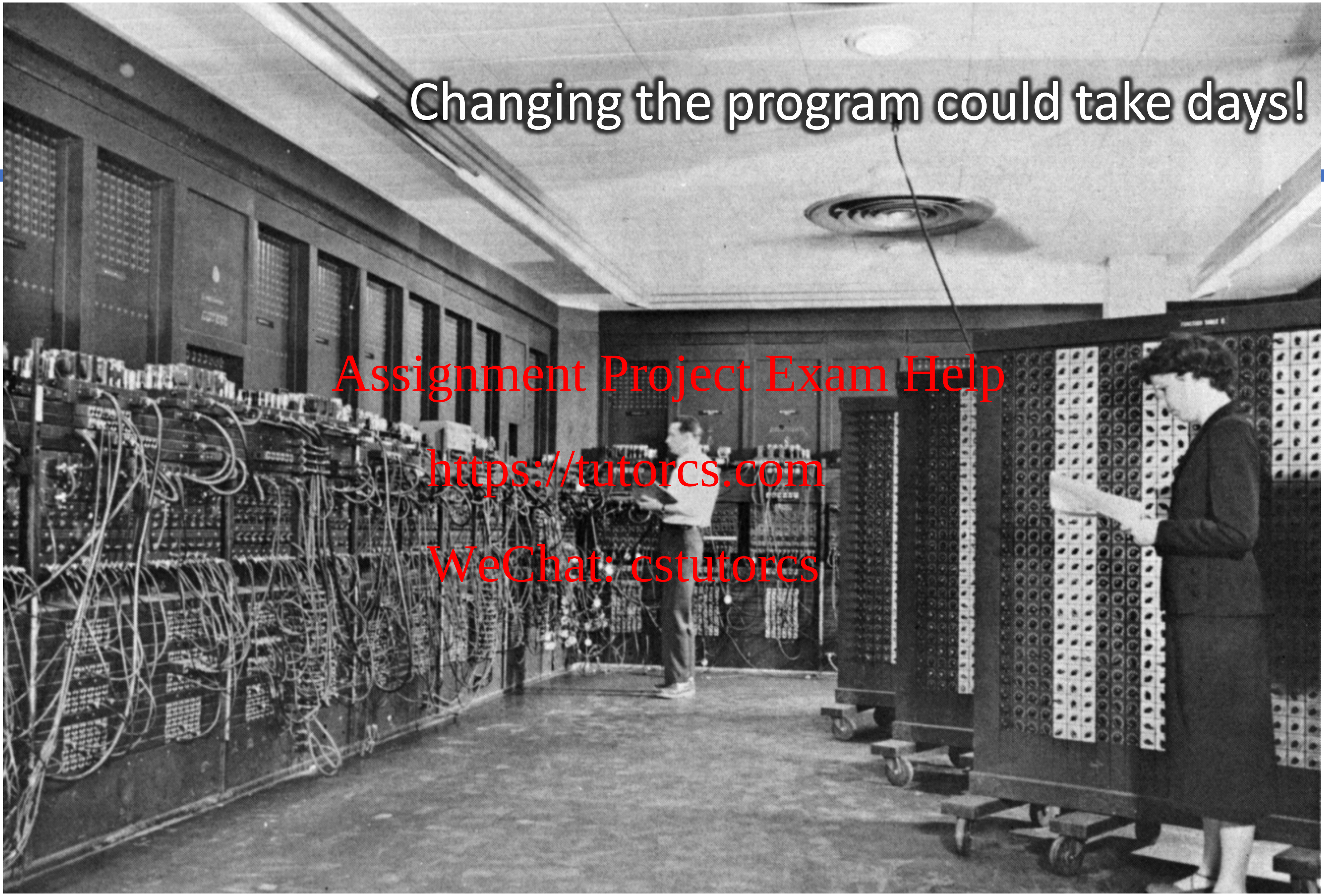


Replica of the Zuse Z3 in the Deutsches Museum, Munich

[Venusianer, Creative Commons BY-SA 3.0]

ENIAC (1946)

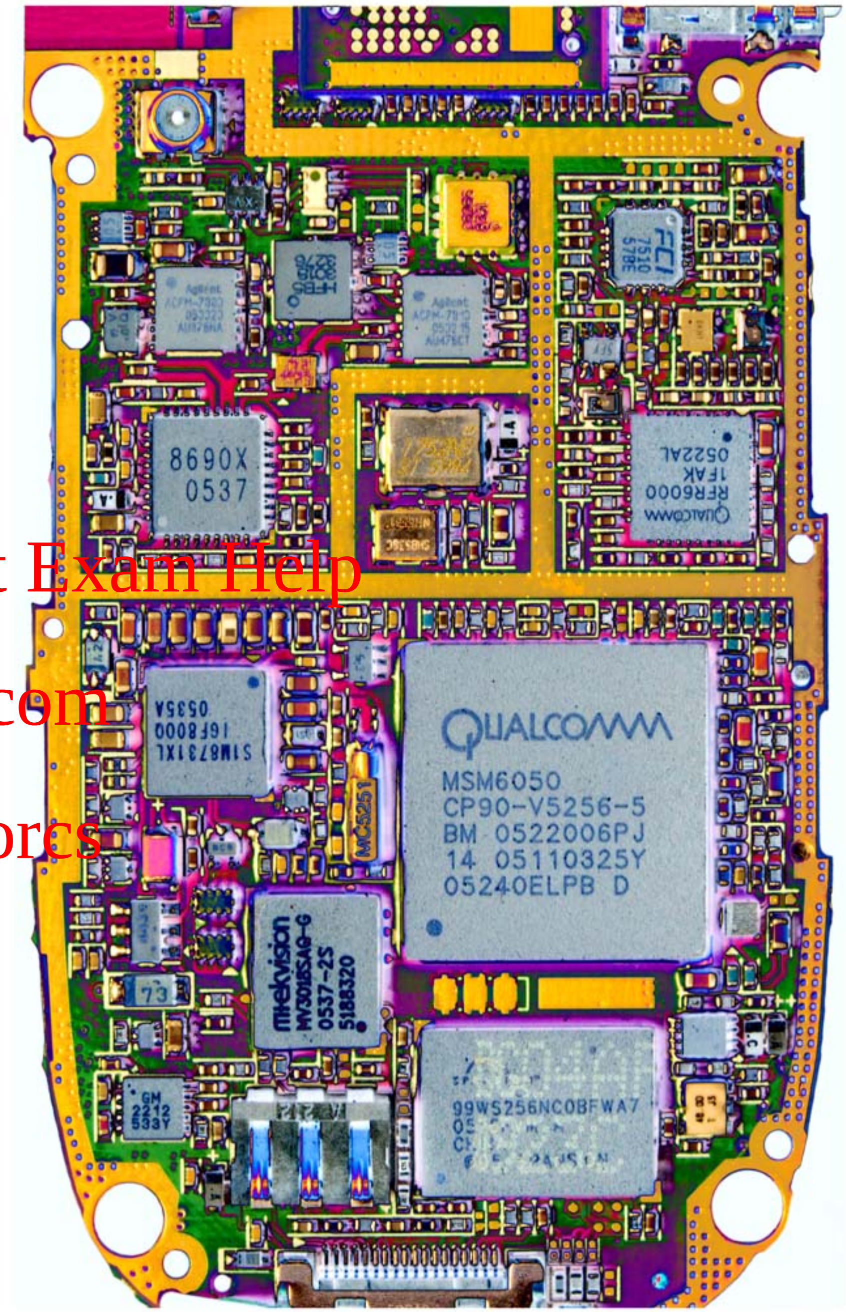
- First electronic general-purpose computer
- Constructed during WWII to calculate firing tables for US Army
  - Trajectories (for bombs) computed in 30 seconds instead of 40 hours
  - Was very fast for its time – started to replace human "computers"
- Used vacuum tubes (transistors hadn't been invented yet)
- Weighed **30 tons**, occupied **1800 sq ft**
- It used **160 kW** of power (about 3000 light bulbs worth)
- It cost **$6.3 million** in today's money to build.
- Programmed by plugboard and switches, time consuming!
- As a result of large number of tubes, it was often broken
  (5 days was longest time between failures!)



[Public Domain, US Army Photo]

Comparing today's cell phones (with dual CPUs), with ENIAC, we see they

cost 17,000X less
are 40,000,000X smaller
use 400,000X less power
are 120,000X lighter
AND…
are 1,300X more powerful.



EDVAC (1951)

- ENIAC team started discussing ***stored-program concept*** to speed up programming and simplify machine design

- Based on ideas by John von Nuemann & Herman Goldstine

- Still the basis for our general CPU architecture today

Commercial computers:
BINAC (1949) and UNIVAC (1951) at **EMC**

- Eckert and Mauchly left academia and formed the Eckert-Mauchly Computer Corporation (EMC)

- World's first commercial computer was BINAC which didn't work…

- Second commercial computer was UNIVAC
  - Famously used to predict presidential election in 1952
  - Eventually 46 units sold at >$1M each

IBM 650 (1953)

- The first mass-produced computer
- Low-end system aimed at businesses rather than scientific enterprises
- Almost 2,000 produced

[Cushing Memorial Library and Archives, Texas A&M, Creative Commons Attribution 2.0 Generic]

Improvements in C.A.

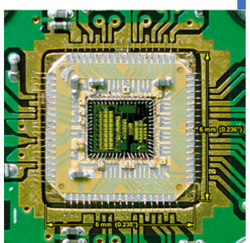
- IBM 650's instruction set architecture (ISA)
  - 44 instructions in base instruction set, expandable to 97 instructions

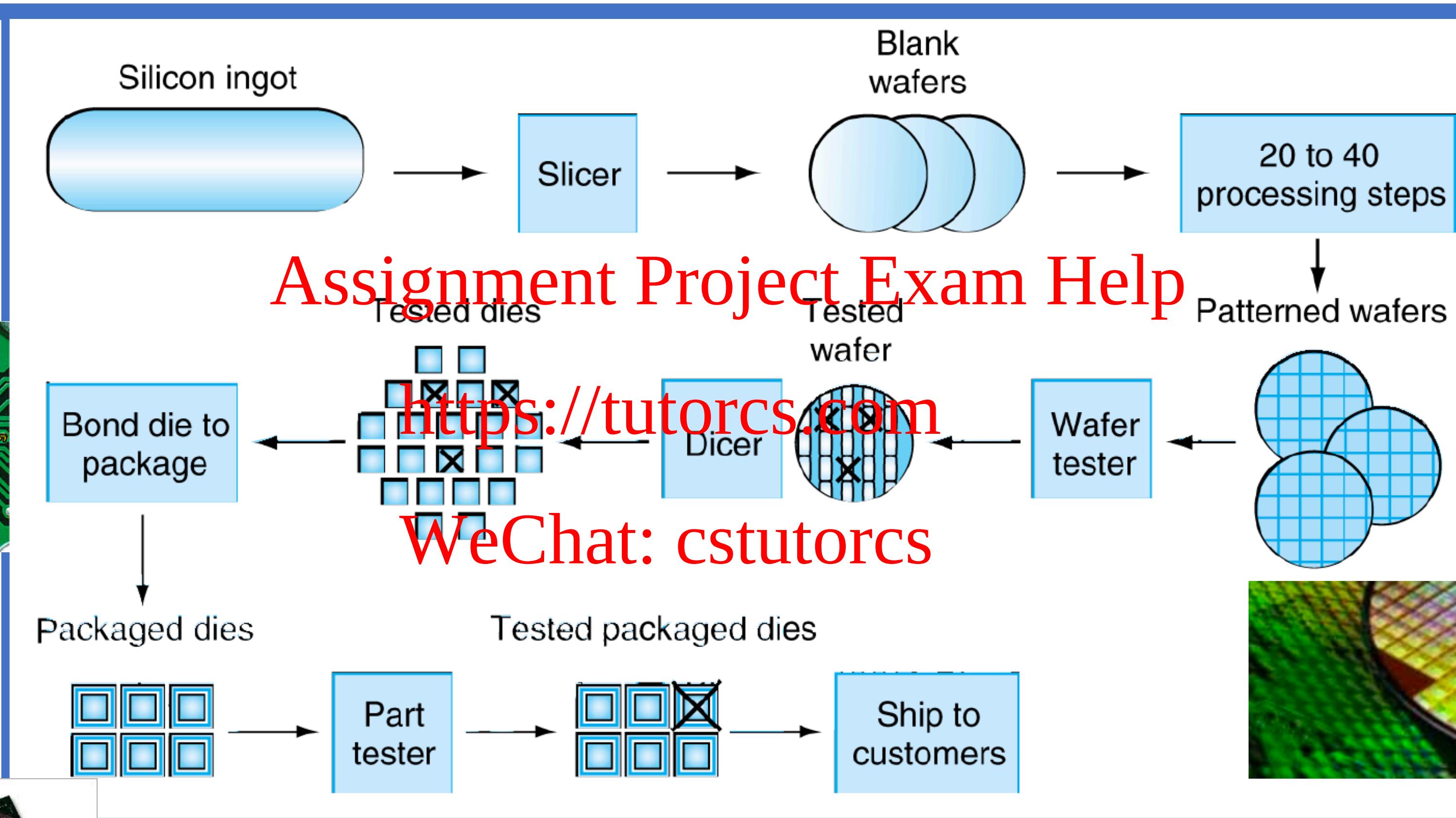
- Hiding instruction set completely from programmer using the concept of ***high-level languages*** like Fortran (1956), ALGOL (1958) and COBOL (1959)
  - Allowed the use of stack architecture, nested loops, recursive calls, interrupt handling, etc…

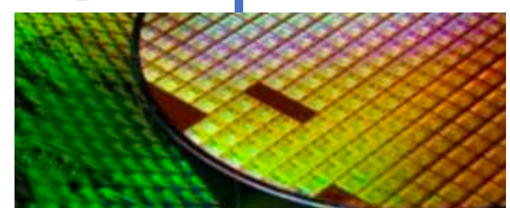
Adm. Grace Hopper (1906 – 1992),
inventor of several High-level language concepts

[Public Domain, wikimedia]

Manufacturing ICs



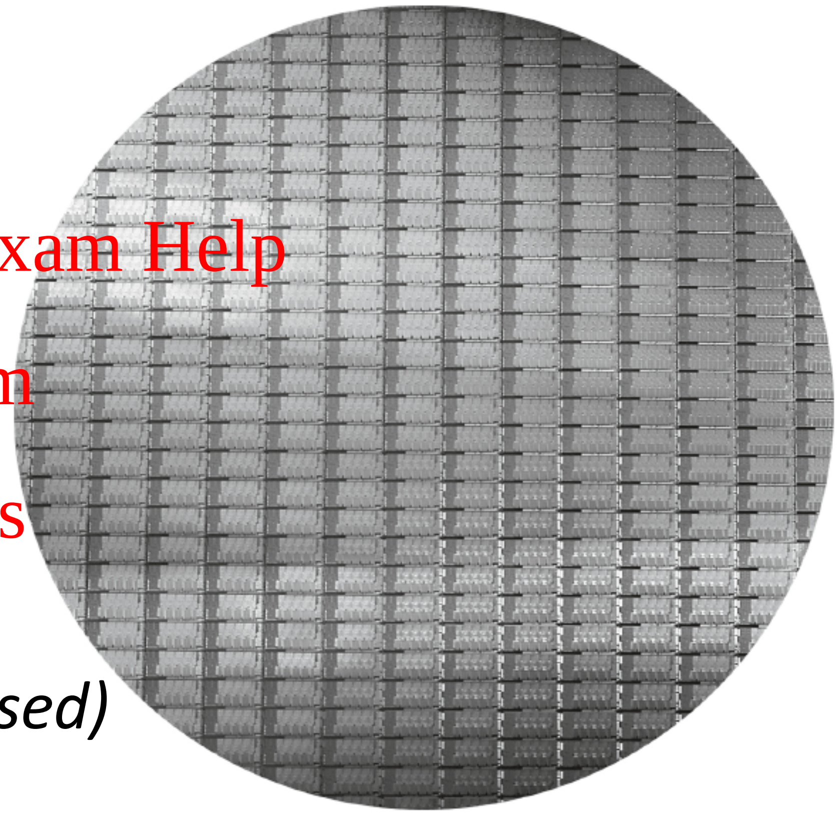




Yield: the proportion of working dies per wafer;
often expressed as a number between 0 and 1

Example: Intel Core i7 Wafer

- 300mm (diameter) wafer
- 280 chips
- Each chip is 20.7 mm x 10.5 mm
- 32nm CMOS technology
  (the size of the smallest piece of logic and the type of Silicon semiconductor used)



Costs of Manufacturing ICs

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- Wafer cost and area are fixed
- Defect rate determined by manufacturing process
- Die area determined by architecture and circuit design

YOUR TO-DOs for the Week

- Do your reading for next class (see syllabus)

- Work on Assignment #1 for lab (***lab01***)
  - Meet up in the lab this Friday
  - Do the lab assignment
  - You have to submit it as a **PDF** using ***Gradescope***
  - Due on **Wednesday, 1/15, by 11:59:59 PM**

</LECTURE>