Computer Science 221

Computer Architecture

Fall 2001

Who Am I?

Professor Roel Vertegaal, PhD

Office: Goodwin 634

Phone: 33070

Email: **roel@acm.org**

Course Home Page:

`www.cs.queensu.ca/home/cisc221/`

Adjunct Dave Dove

Office: Goodwin 5

Dave Dove will hold office hours

Monday 2:30 - 3:30 and Wednesday 1:30-2:30.

Goals for the Course

Knowledge of:

- Basic concepts of computer architecture
  - what hardware looks like to the software.
- Information representation;
- Machine and assembly language programming;
- High level languages and execution on a machine
  - C, C++, Java;
- Interrupts and basic concurrent processes;
- Memory structure and management.
- Logic gates.

More About the Course

Background assumed for CISC 121

- that you have taken CISC 101 or have its high school equivalent.
- that you have taken CISC 121
- that you have taken CISC 104
- that you know how to use high-level languages.

We will assume this material!

Textbook

Computer Systems -- J. Stanley Warford

Other Materials

- **David Patterson and John Hennessy, Computer Organization and Design, Second Edition, 1998.**
- **V. Carl Hamacher, Zvonko G. Vranesic, and Safwat G. Zaky, Computer Organization, Fourth Edition, 1996.**
- **Andrew S. Tanenbaum, Structured Computer Organization, Fourth Edition, 1999. A very well-written book that provides background information on real machines that Warford lacks.**

Administrivia

Marking Scheme

assignments	25%
midterm	25%
final	50%

Assignments

- 4-5 individual assignments (25% of mark)

  4 programming, 1 non-programming
- you can work on assignments on your own
- assignments posted on web
- hand in assignments in hard copy form
- no late assignments

Tutorials

- No organized tutorials
- TAs will be present in Goodwin 230 at set times for consultancy with assignments
- Details of dates will be posted on web

Teaching Assistants

- **Teaching Assistants will help you with questions**
- **Teaching Assistants will help you with assignments**
- **TAs will be present in Goodwin 230 at set times**
  - Details of dates will be posted on web
- **TAs are:**
  - Jeff Shell (shell@cs.queensu.ca) Goodwin 636
  - Yaping Ding (ding@cs.queensu.ca) Goodwin 636
  - Ayse Karaman (karaman@cs.queensu.ca) 756
  - Wenzhong Chen (chenw@cs.queensu.ca) 633

Computing Facilities

- **The first assignment will be in high-level language.**
- **Other assignments will use the Pep/6 simulator**
- **Use Unix, either on Unix workstations or through PCs running XWin32, in the CISC/ECE labs in Goodwin and Walter Light Halls.**
- **To enter the labs you will need to purchase special buttons called iButtons. The previously used number pads have been replaced by the special button readers. The iButtons can be purchased in the bookstore for $2.79. They have to be encoded: you do that at the Jeffrey lab.**

Computing Facilities (2)

- **If you are registered in the course, you should already have an account. If you don't have an account, please check QCARD to ensure you are registered before you ask me about your account. The userid will be the same as your userid on qlink, and your initial password the same as your original qlink password (check getid again if you've forgotten it).**
- **Rooms with workstations are in Walter Light 214 and Goodwin Hall 230, 235, 248.**

Other Notes…

Reading:

Don't rely on lectures only
Next week: Chapter 3

Lecture Slides:

on the web after class, no not before class.

Messages:

web home page www.cs.queensu.ca/home/cisc221
you are responsible for checking here often!
you not be contacted about announcements

Lectures

- **There is only one section will meet in BIO**

  Slot 15:
  - Tuesday 12:30-1:30 BIO 1102
  - Thursday 11:30-12:30 BIO 1102
  - Friday 1:30-2:30 BIO 1103

- **Some classes marked * will be taught Dave Dove**

Course Schedule

W1	Sep 11- Sep 14	Introduction	
W2	Sep 18- Sep 21	Information Representation (1)	(Chapter 3)
W3	Sep 25- Sep 28	Information Representation (2)	(Chapter 3)
W4	Oct 2- Oct 5	Computer Architecture Pep/6	(Chapter 4)
W5	Oct 9- Oct 12	Assembly Language	(Chapter 5)
W6*	**Oct 16- Oct 19**	**High-Level Languages (1)**	**(Chapter 6)**
W7*	**Oct 23- Oct 26**	**High-Level Languages (2)**	**(Chapter 6)**
	Midterm*		
W8	Oct 30- Nov 2	Instruction Sets	(Tanenbaum)
W9	Nov 6 - Nov 9	Devices & Interrupts	(Chapter 8)
W10*	**Nov 13- Nov 16**	**Storage Management**	**(Chapter 9)**
W11	Nov 20- Nov 23	Combinational Logic	(Chapter 10)
W12	Nov 27- Nov 30	Sequential Logic & Review	(Chapter 11)

**Marked classes will be taught by Dave Dove, same place same slot*

Week 1: Plan

- **Introduction & Administratrivia**
- **Layers of abstraction between people and the machine**
- **History of their implementation**

Computer

- **A digital computer is a machine that can solve problems for people by carrying out instructions given to it.Ó Tanenbaum**
- **In its simplest form, a contemporary computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored information, and produces the resulting output information. The list of instructions is called a computer program, and the internal storage is called computer memory.Ó Hamacher et. al.**

Architecture

- **Vague notion of a description of the overall organization of something, without the details of how the something works. e.g. house design with lines representing walls, windows, roof, but not how they are really attached together and what materials are they built with.**

Computer Architecture

- **The design of those attributes of a computer system which are visible to the programmer.**
- **Not Computer Organization, which is a high-level description of hardware implementation that provides that view to the programmer.**
- **Computer Architecture components**
  - instruction set
  - bits used to represent various data types
  - I/O mechanism
  - memory addressing techniques

Levels of Abstraction

- **A digital computer is a machine that can solve problems by carrying out instructions given to in in a form of a program.**
- **The electronic circuits of a computer can recognize and execute a limited set of simple instructions.**
  - add 2 numbers, check if a number is zero or copy a number from one part of comp. memory to another.
  - Such a language is called Machine language. The instructions are primitive as possible so that the computer can be produced inexpensively and reliably.
  - Very tedious and error prone, as you will see this term.
- **There is a great gap between executing machine instructions and a user pressing a button to obtain some complicated report printed out. How to bridge that gap?**

Levels of Abstraction (2)

- **The solution evolved as levels of abstraction.**
  - suppression of detail to show the essence
  - an outline structure
  - subdivision of a system into smaller subsystems
- **We can imagine each level of abstraction as a consecutive virtual machine not too dissimilar from the previous level.**
- **Lets call the language that the physical machine M0 is 'wired' to execute - L0. Then let us image a machine M1 (virtual) with a language L1 that is more convinient to use by people. How can a program P1 written in language L1 be executed on our physical machine M0?**

Levels of Abstraction (3)

- **There are two ways:**
  - Translation: The whole of P1 will need to be translated into a program P0 that does the same job as P1, but uses only instructions from L0 language. Then P0 can be directly executed on M0 and we no longer need P1.
  - Interpretation: Each instruction from P1 is interpreted into some instructions from language L0 and they are directly executed on M0 machine. This method does not create a whole P0 and does not discard P1. This technique is called interpretation.
- **To generalize the above, we can have more than 2 different languages, each one more and more convenient for the people to use and express their problem solutions with.**

Levels of Abstraction (4)

- **There are seven levels in contemporary machines:**
- **(7) Applications level**
  - familiar to all users of computers: word processor
- **(6) High level language**
  - C, C++, Pascal , Turing, Java, Fortran, Cobol
- **(5) Assembly Level**
  - a representation of machine level instructions in a form that human beings can deal with more easily
- **(4) Operating System Level**
  - The operating system is software that stands between the hardware and other software, providing services beyond the bare machine

Levels of Abstraction (5)

- **(3) Machine Level -** software/hardware interface.
  - Computer as a device which is able to store representations of numbers,
  - has some method to get these numbers from the outside world and to output them again,
  - able to interpret some of the numbers as instructions to do very simple steps, like adding the number at one location in memory to that in another
- **(2) Microprogramming Level**
- **(1) Logic Gate Level**
  - We don't get to transistors, but will talk about how a computer can be built out of logic gates, circuits made from transistors and resistors

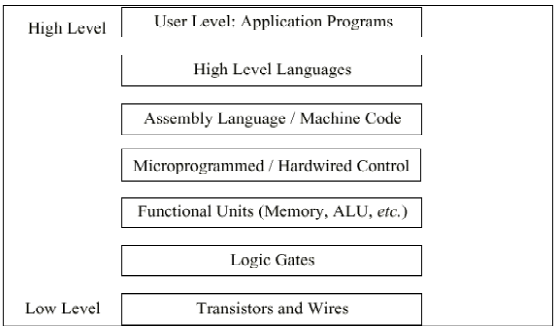
Definitions

- ***Computer Architecture* deals with functional behavior of a computer system as viewed by programmer.**
  - Size of data types
  - Types of instructions (implemented in machine language)
- ***Computer organization* deals with physical relations between components not visible to the programmer.**
  - The CPU
  - Clock frequency
  - SDRAM
- **Often these cannot be seen as completely separate.**
  - Type of CPU may determine type of instructions

II. Levels of Abstraction

Levels of Machines

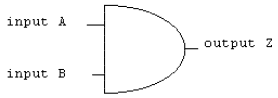
- **As we have seen, there are a number of levels in a computer, from user level down to the transistors.**
- **Progressing from the top level downward, levels become less abstract as more of the internal structure of the computer becomes visible.**



(1) Digital Logic Level

- **Hardware viewed as collection of logic gates**
  - Two more levels down: circuits and physics
  - Gates are built from analog components but modeled as digital devices
- **Gates can compute something**
  - Each gate has one or more inputs (0 or 1) and computes as output to some simple function such as AND or OR or ADD.
- **Gates can store numbers**
  - Gates are combined to form 1-bit memory, which is combined to form x-bit registers (x= 16/32/64)

Logic Gate: AND



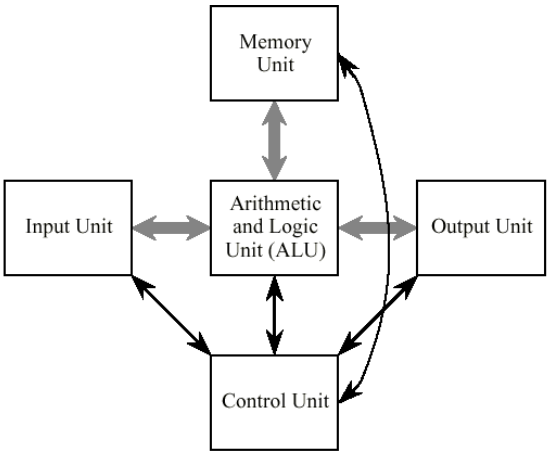
A	B	Z
0	0	0
1	0	0
0	1	0
1	1	1

- The state of the output (high or low) depends upon the combination of the input states.
- In the case of the gate shown, the output will only be high if both inputs are high. If either one input or both inputs are low then the output will be low.
- These characteristics can be shown using the above TRUTH TABLE. 1 indicates a high and 0 indicates a low.
- Note that Z is only a 1 when A AND B are both at 1.
- There is a form of mathematics associated with logic gates called BOOLEAN ALGEBRA. Invented a few hundred years ago by Mr Boole, to solve problems in logic.
- The most frequently used gates are AND, OR, NAND, NOR, NOT and XOR.

(2) Micro-Programming Level

- **A hybrid between hardware and software: CPU**
- **A hardware interpreter that implements machine language**
  - Interpreter performs small programs using lower-level operations on lower-level hardware.
- **Arithmetical Logical Unit is a circuit capable of performing such lower-level logic and arithmetic.**
- **Registers form a local memory.**
  - Data path connect registers with the ALU.
- **Control Unit operates the data paths**
  - Selects registers and let ALU operate on them
  - governs the series of steps taken by the data path during the execution of a user-visible instruction

(2) Micro-Programming Level: CPU



(2) Micro-Programming Level

- **Control Unit uses a *micro-program* for this purpose**
  - A sequence of micro-operations needed to operate the data paths
  - Since the micro-programs stand between the logic design (hardware) and the program being executed (software), they are sometimes referred to as *firmware*.
  - As such, the micro-program is the *interpreter* for machine level instructions from the next level
  - The trends is towards implementing micro-programs in hardware as this is faster.
- **We will later see that this kind of CPU architecture is called a Von Neumann machine.**

(3) Machine Language Level

- **Also called *ISA*: Instruction Set Architecture.**
- **This is the exact boundary of hardware and software: a set of instructions executed by the CPU.**
- **At this level, a computer is able to:**
  - Store representations of numbers
  - Input and output these to and from outside world
  - Interpret some of these as instructions to do simple things, such as adding the number at a location in store to another in another store.
- **Instructions and data are represented as numbers in binary form:**

  1010 1100

(3) Machine Language Level

- **CISC: *Complex Instruction Set Computers***
  - Implement a large array of instructions
  - Easier to write compilers: no longer problem
  - Faster to compile: no longer problem
  - Smaller programs: no longer problem
  - Require less memory: no longer expensive
  - VAX: had instruction for calculating value of polynomial given array of coefficients!
- **Examples:**
  - 680x0 Motorola family
  - x86 Intel family

(3) Machine Language Level

- ***RISC: Reduced Instruction Set Computers***
  - Implement small array of instructions
  - Primitive instructions have become instruction set
  - Execute code faster because interpretation is easier
  - Each instruction is executed in 1 cycle
  - Can therefore do some parallel processing
  - This is the current trend
- **Examples:**
  - PowerPC
  - SPARC
  - Intel calls Pentium II & III hybrid: CRISC

CPU used for Assignments

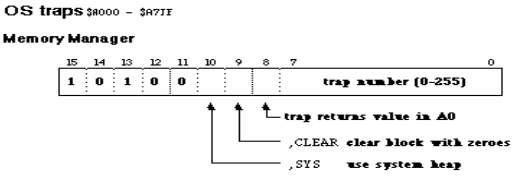
- **Practical difficulties of programming bare hardware:**
  - need special privileges to do it under a multi-user OS, and can crash the OS if you make mistakes
  - need special downloading, cross-assembly facilities to do it on a separate machine
  - limited development environment
  - Warford says that hardware changes; don't waste time learning details of one system to learn the concepts
- **So we use a simulator, a program that pretends to be a computer. Pep/6**

(4) Operating System Level

- **Operating System stands between the hardware and other software**
- **Interpreter provides services beyond the bare CPU's capabilities**
  - Interrupt handling
  - Memory Management
  - Process Control: running a program
  - Device Control
  - Provides extra functionality through *Application Programming Interface* (API) that is sometimes implemented as an extra instruction set
- **After this level, levels tend to be translated**

(4) Operating System Level

- **Example: A-Trap mechanism in 68K Macintoshes**
- **Motorola 68K did not recognize instructions starting with 1010 or in hexadecimals: A**
  - The CPU would stop if it encountered an instruction starting with A
  - It indicated this by signaling an interrupt on a pin.
  - This would activate system software that parsed the A-trap instruction: a trap door mechanism



(5) Assembly Language Level

- **A representation of machine language that is easier to read.**
- **Essentially the same level as machine level.**
- **However, it includes calls to the operating system.**
  - E.g., as instructions through a trap mechanism
- **So it is what stands in between programmers and the machine.**

```
LOADA  h#0011,d ;load 3 into accumulator
ADDA   h#0013,d ;add 5
```


The Top Levels

- **(6) High-Level Languages**
  - Even easier to use
  - Familiar from first-year programming courses
  - Languages such as C, C++, Java
  - Mostly translated using compiler
  - Java is compiled and then interpreted
  - This is the level of compilers that translate to assembly
- **(7) Applications Level**
- **(8) User**

So How Did The Levels Develop?

- **A program written in true machine language (level 2) can be directly executed on circuits (level 1)**
- **First digital computers had only 2 levels**
  - Instruction Set Architecture (ISA) was directly executed at the digital logic level
- **1951: Maurice Wilkes (U of Cambridge) suggest 3 level approach to simplify hardware**
  - Machine used interpreter to execute ISA
  - Hardware would only need to execute the interpreter and therefore less instructions
  - Hardware reduction meant less vacuum tubes: decrease cost, size and increase reliability

So How Did The Levels Develop?

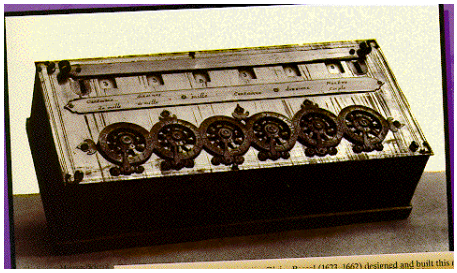
- **1960:Development of primitive Operating System.**
  - In the early days, a human operator executed the reading of punch cards, the compilation of programs into new punch cards, the reading of machine language cards, the execution etc.
  - If there was a bug (literally), all this had to be repeated again, resulting in considerable down-time
  - Job control languages were developed to speed up this process and reduce need for human intervention.
- **1960: IBM compatibility in a family of machines!**
  - For this, a unified leveled architecture was required
- **But before all that could happen...**

III. A Brief History of Computing

Calculation: La Pascaline (1642)

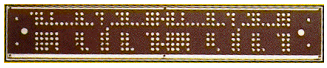
- **Blaise Pascal is credited for the invention of the first mechanical calculator (if we ignore abacus).**
- **Could add and subtract numbers recorded on a set of 8 dials connect to a drum.**
- **Big Innovation: Dials do not rotate until a carry is produced from a dial in a lower position.**
- **1673 Von Leibniz improves:**

  Binary calculus.

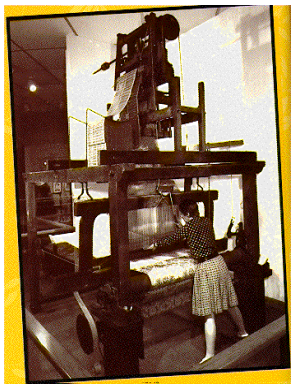


Automation: Jacquard Looms (1800)

- **Another important development was sequencing, required for automation.**



- **These weave machines used punch card loops with leather straps recording instructions for a colored pattern.**



Charles Babbage (1791 - 1871)

Grandfather of Computing

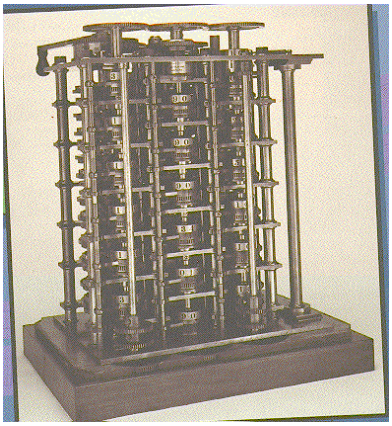
– Built *Difference Engine* to allow easier calculation of polynomials using finite differences (1836)

– Heavy reliance on manually computed mathematical tables in navigation & science

(Photograph of Charles Babbage in 1847.)

Difference Engine (1822)

- **Computed tables by setting and turning mechanical gears.**
  - Store numbers
  - Store 1 algorithm
  - Calculate
  - Read manual input
  - Print output
- **Principles found in all modern computers!**



(Babbage's Difference Engine in Science Museum, London)

Difference Engine: Compute 7^3

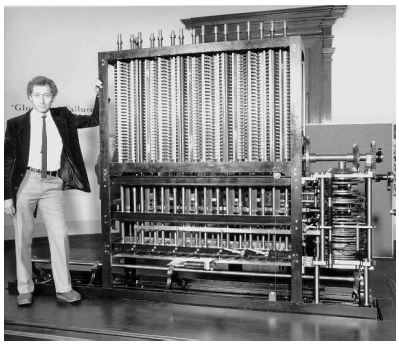
N	N*N*N	First order difference	Second order difference	Third order difference
1	1			
2	8	7		
3	27	19	12	
4	64	37	18	6
5	125	61	24	6
6	216	91	30	6

(=2rd) (=1nd order)

To calculate $7^3 = 6 + 30 + 91 + 216 = 343$

Analytical Engine (1836)

- **Computed more than one algorithm.**
  - Store numbers
  - Calculate
  - Read data card
  - Output data card
  - Read program card
  - Control operation
- **Principles found in all modern computers!**

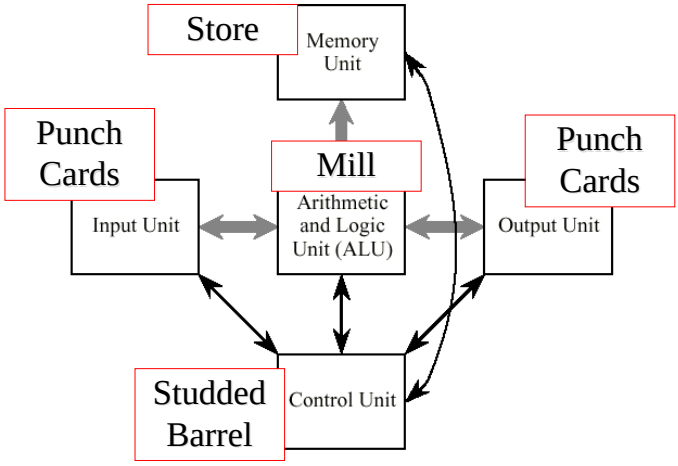


(Babbage's Reconstructed Engine 2 in Science Museum, London)

Analytical Engine (1836)

- **Babbage never got around to building the device.**
  - Mechanical hardware was not up to standards.
- **Input data and instruction data was read from punched cards, like Jacquard loom.**
- **Output data was printed or written to punch cards.**
- **A store recorded 1000 50-digit decimal values by the positions of geared wheels (like the Pascaline)**
- **A mill could operate on numbers**
  - retrieved from and return to the store via axes
  - could add, subtract, multiply and divide
- **Mill was instructed by a studded barrel control unit**
  - Bumps on barrel selected operation using gears

Sounds Familiar?



Ada, Lady Lovelace (1815-1852)

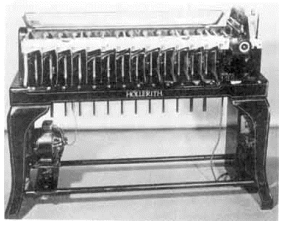
- **Translated French articles about the Analytical Engine into English.**
  - Spread knowledge about computers
- **Wrote the first computer programs**
  - For Babbage's Analytical Engine
  - To calculate Bernoulli numbers

QUOTE from Ada, The Enchantress of Numbers by Betty Alexandra Toole

Once Ada had made the distinction between numbers and the operations to be performed, it was not difficult for her to project further how the Analytical Engine would then be capable of giving two types of results; numerical and symbolic, (eg algebraic). In effect, an Analytical Engine could generate new programs as well as numbers. As a result the Analytical Engine opened up a vast new territory for the analysis of information. Here again, the Ada software language contains somewhat unique facilities corresponding in a sense to Ada's insight. One such Ada facility is the generic subprogram, a template for future software generation. Having defined a generic subprogram for data of one type, the Ada software developer can create new copies automatically tailored to data of other types.

Hollerith's Census (1890)

- **Hollerith encoded 1890 U.S. Census data on punched cards.**
- **Electric Tabulating System used electro-mechanical sensors to tabulate data.**
  - The punch was similar way to a typewriter
  - Counting was completed in 3 months instead of 2 years if counting had been done by hand: 62,622,250.
- **His Tabulating Machine Company (1896) became IBM.**



Hollerith's Electric Tabulator (1890)

The Digital Revolution

- **Data in physical variables rather than digits**
  - Position of a wheel, voltage
- **Digital: abacus, CD, digital clock**

- **Analogue: slide rule, face clock, vinyl album**
- **Digital is more:**
  - Versatile
  - Accurate
  - Simple: 1's and 0's
- **By the beginning of the 20th century, computers shifted to digital**

Electronic Digital Computation

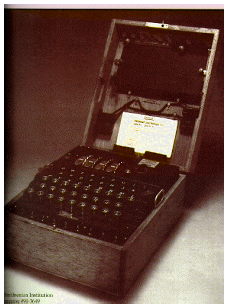
- **World War II accelerated the development of digital computers.**
  - Military Intelligence: breaking codes
  - Computing Physics of Grenade Trajectories
  - Computing Physic of Atomic Bomb (Los Alamos)

- **Early 1940's Konrad Zuse (Germany), John Atanasoff (USA) and George Stibitz (USA) had developed various electronic calculators.**

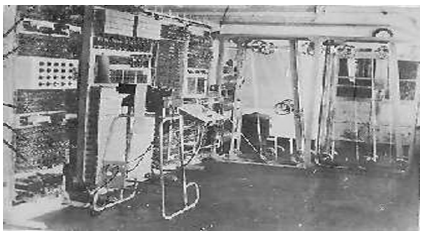
- **1944 George Aiken built an electronic version of the analytical engine.**

Alan Turing (1912-1954)

- **1936 First Theory of Computation.**
  - Turing Machine.
- **1941-1944 Colossus.**
  - Designed to Break Enigma Code
  - First Digital Electronic Computer
  - Used Vacuum Tubes as Gates



Enigma

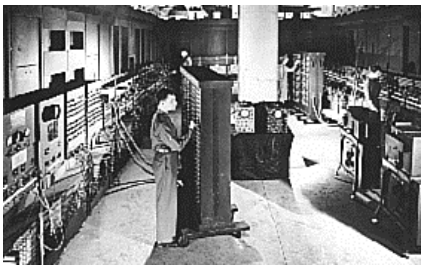


Colossus

AlanTuring

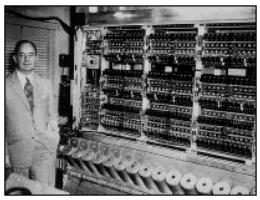
ENIAC (1946)

- **First general-purpose electronic digital computer.**
  - Weight 30 tons, 140 kW power, 17,468 vacuum tubes
  - Demonstrated versatility of digital computing
  - Demonstrated reliability of digital vacuum tubes
- **Built to calculate artillery range tables for US Army**
  - By John Mauchley and J. Presper Eckert: UNIVAC



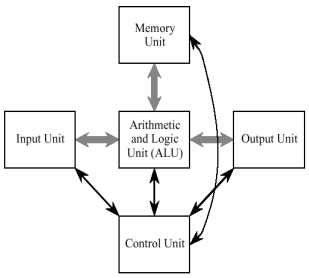
Von Neumann (1946)

- **Paper "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument".**
- **Added the last important feature:**
  - Instruction for the machine coded as numbers in same memory as is used to store results
  - Reprogramming matter of readin a new set of instruction in from the input device, instead of rewiring the hardware
- **Von Neumann machine**
  - A single instruction sequence, stored program computer



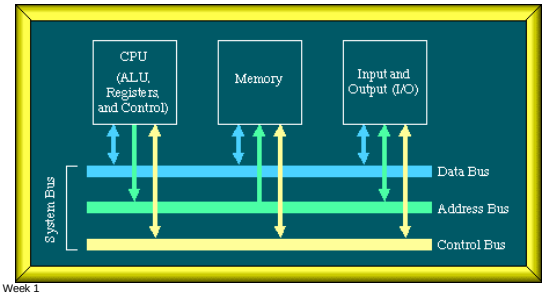
Von Neumann Architecture

- **Input and Output Devices**
- **Memory Organ**
  - Primary (selectrons) or registers (fast)
  - Secondary (tape) or hard drive (slow)
  - Data & Instructions
- **Arithmetic Organ**
  - Accumulator
  - > ALU
- **Control unit decodes instructions sequentially**
  - Order code: 6-bit opcode & 12-bit operand address



System Bus Model Refinement

- **ALU & CU form CPU, input & output form I/O unit and these are separate units (3rd unit is Memory)**
- **Units communicate using System Bus: Data, Address & Control busses.**
- **32 bit bus = 32 Physical Wires.**



Modern Computing

- **A lot has happened since then.**
  - On April 14, 1954, Gordon Teal showed Texas Instruments' Vice President, Pat Haggerty, the first working silicon transistor
  - According to Moore's law (founder of Intel), a lot more is going to happen.
  - Number of transistors doubles every 1.5 years
- **But essentially, all computers are still based on a classic Von Neumann Architecture.**