CS151: Computer Programming Fundamentals

- Today (6/9)

  Course Information

  Algorithms and Computers

  Brief History of Computers

  Anatomy of a Computer

  Language Hierarchy

Course Information

- website: www.unm.edu/~mfricke
- mailing list: cs151@cs.unm.edu
- Use CIRT AIX machines (xwin32)
- Emacs with g++ will be the official development environment
- Summer courses are very intensive

Current Assignments

- Homework 1 has been posted.

  Variables, mathematical and logical operators, input/output, and the "if" operator.

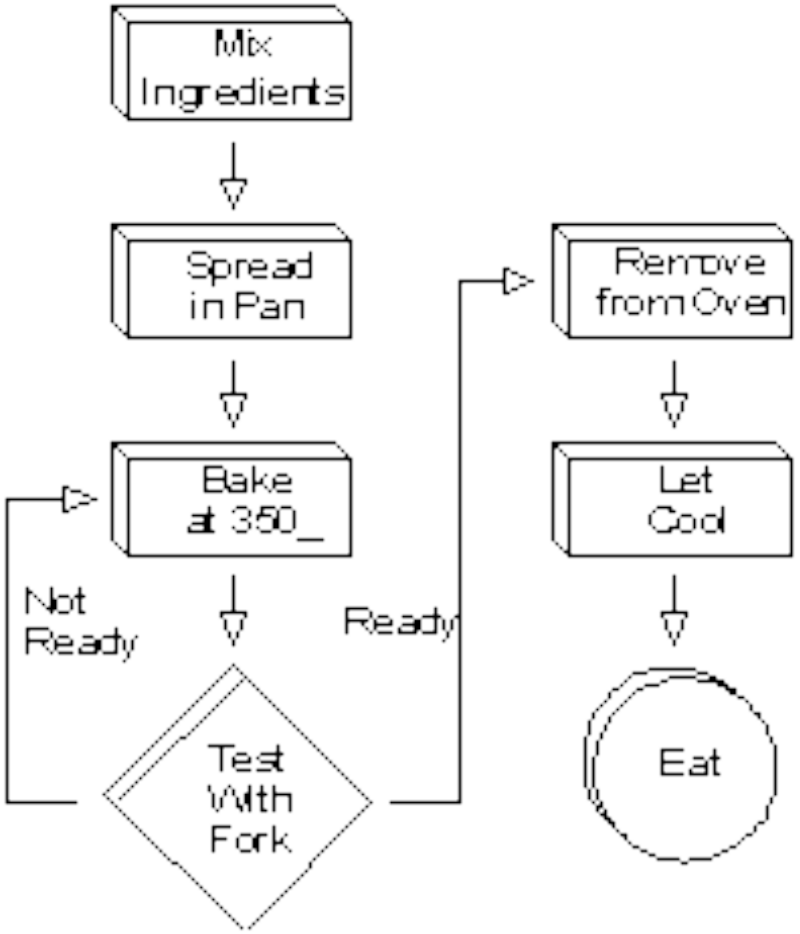
  Due in one week (June 16th) at midnight.
- Project 1 has been posted.

  Write a binomial root solver using the quadratic equation.

  Due in two weeks (June 23rd) at midnight.

Programs and Algorithms

- Algorithm: a series of abstract steps that solve a particular problem
  - Mathematics
  - Food Recipes
  - Textile Weaving

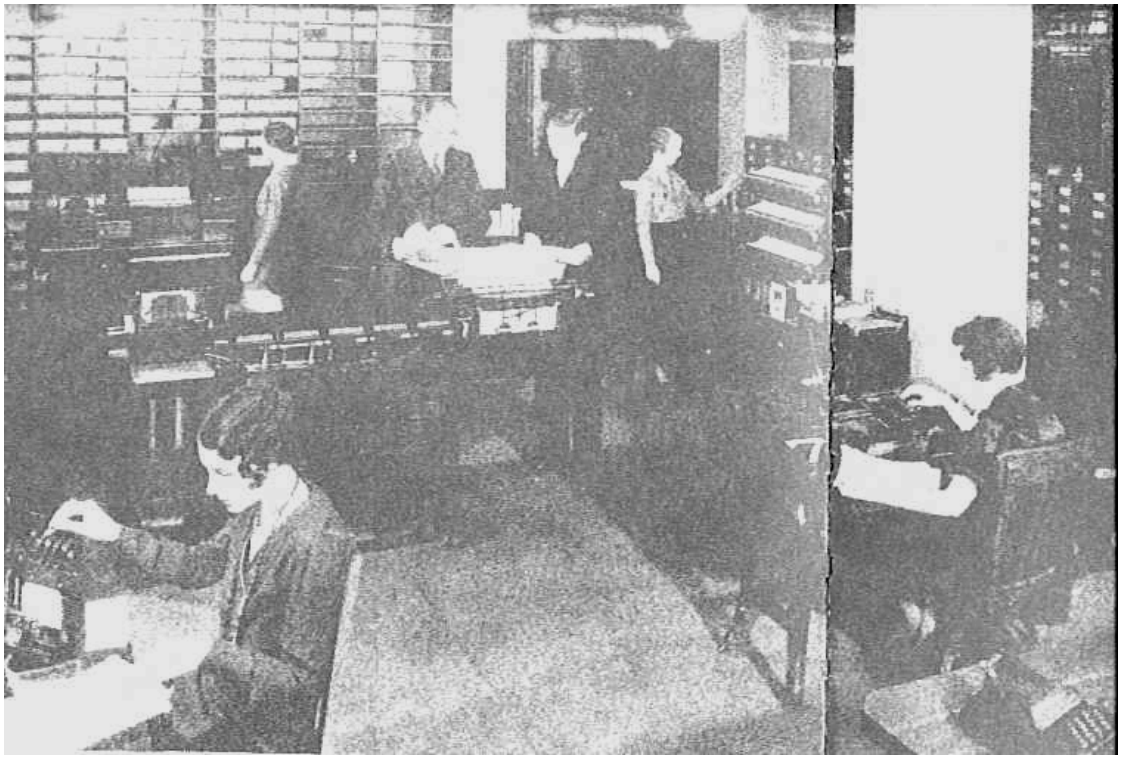


Programs and Algorithms

- Computer (idealized definition): anything capable of following the steps of an algorithm
- Universal computer: a computer capable of following the steps of all possible algorithms
- Program: the encoding of an algorithm so that a computer can follow the steps

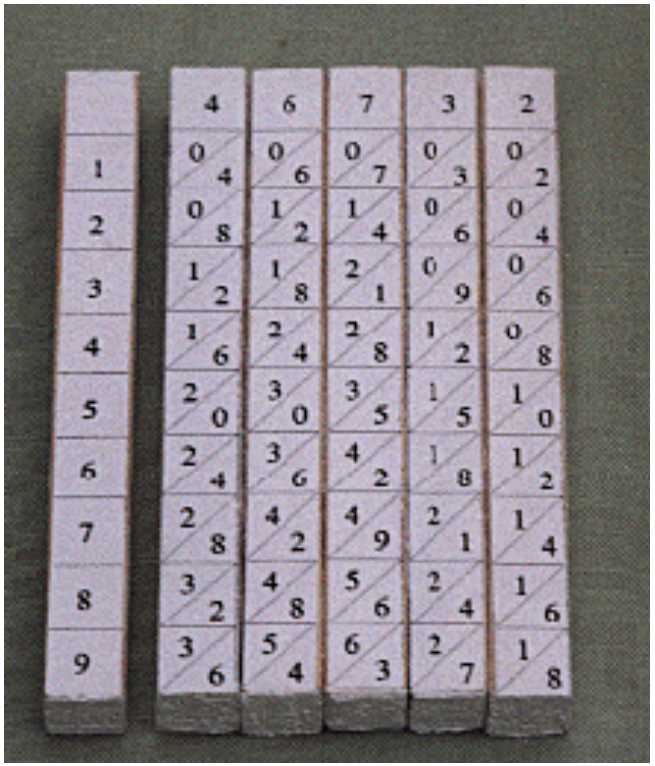
The First Computers

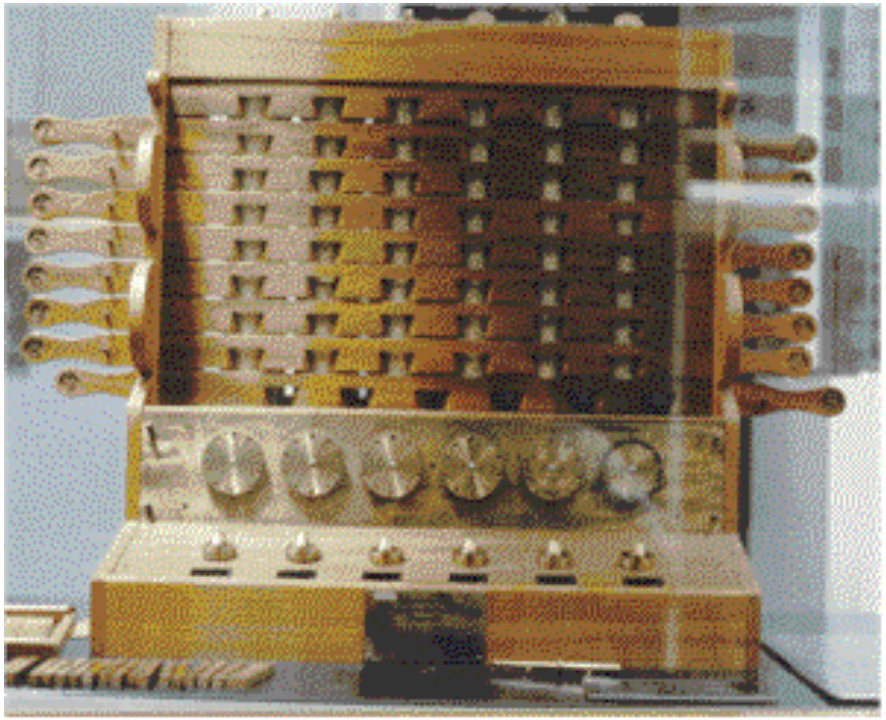
14th Century
Counting Tables

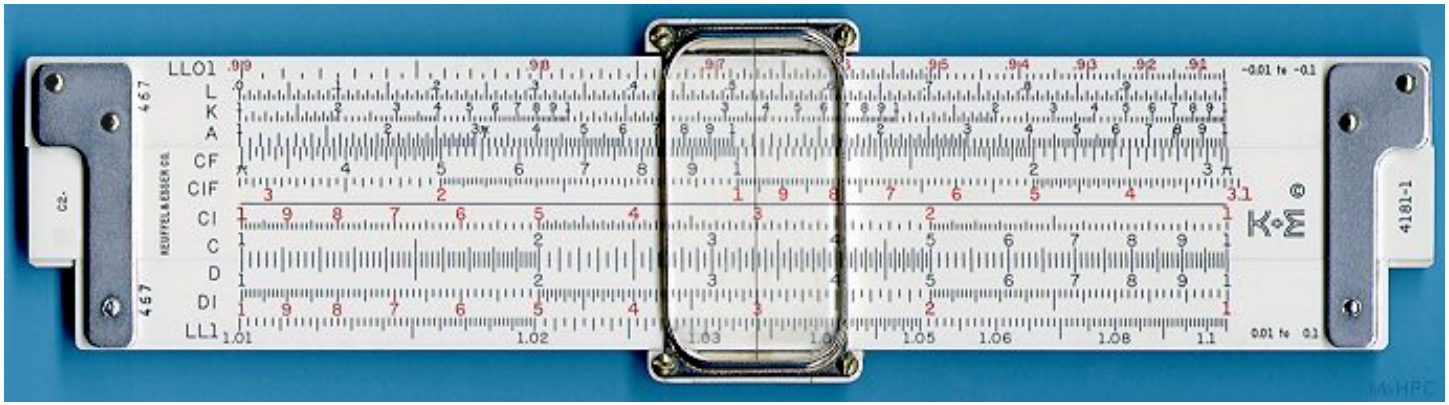


A 20th Century
Computer Center

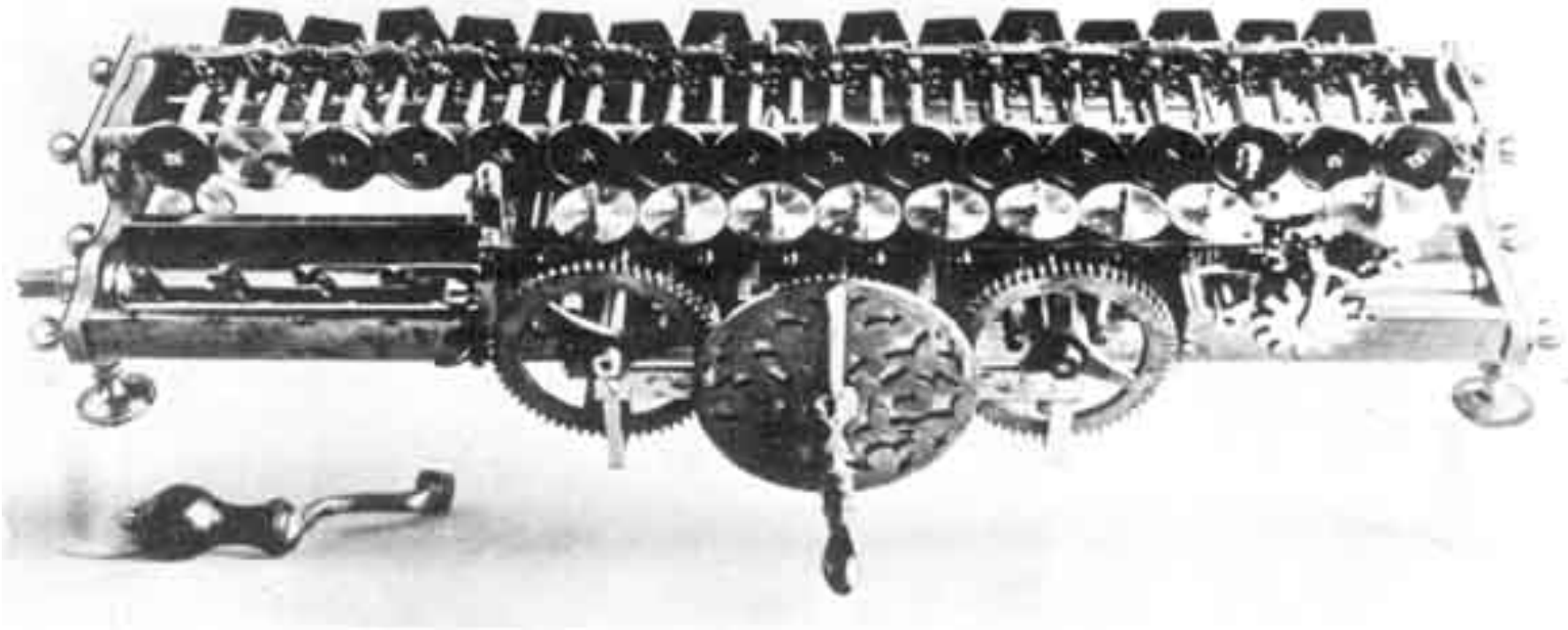
Napier's Bones and the Calculating Clock



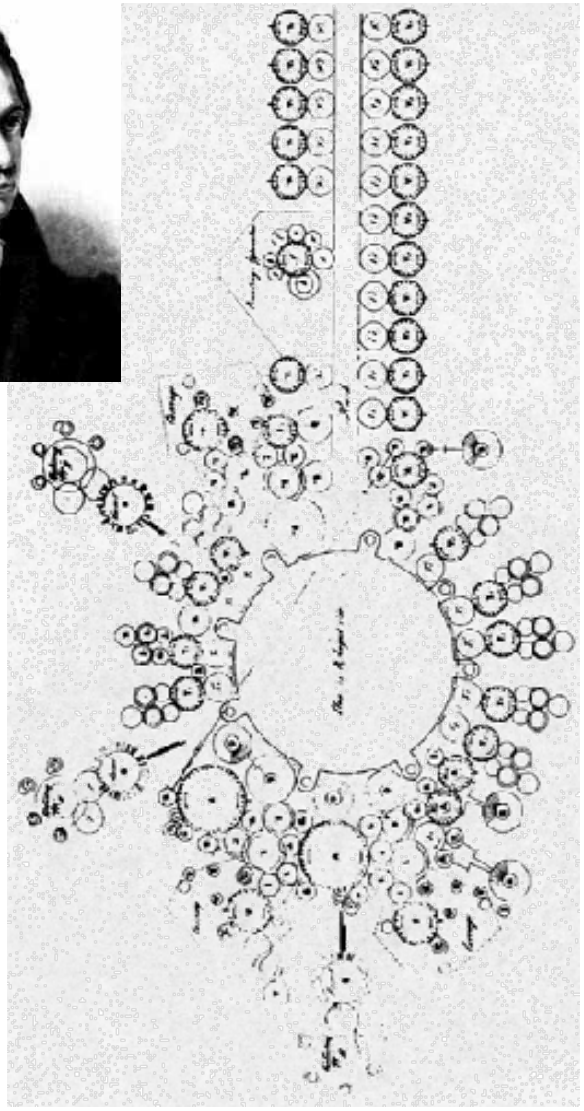


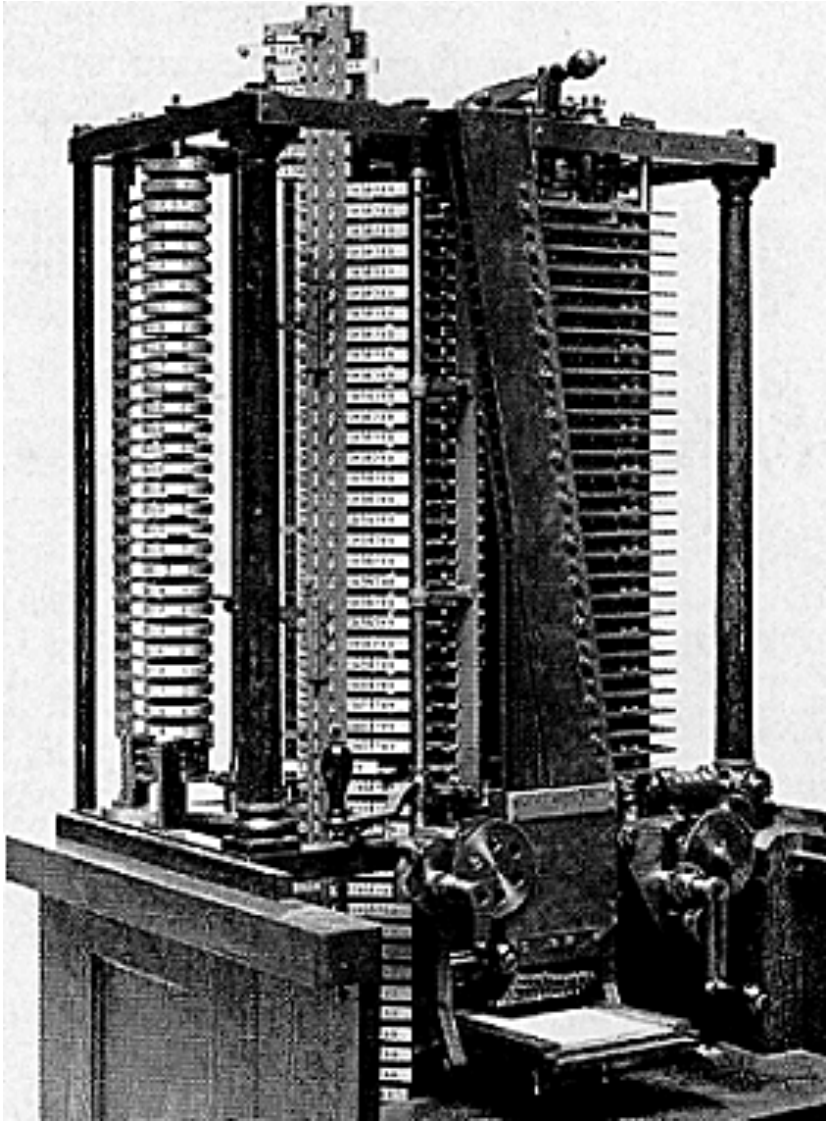


Baron Gottlieb Leibniz
The "Step Reckoner," 1671



Charles Babbage's Analytical Steam Engine
Designed in the 1830s, not built until 1906



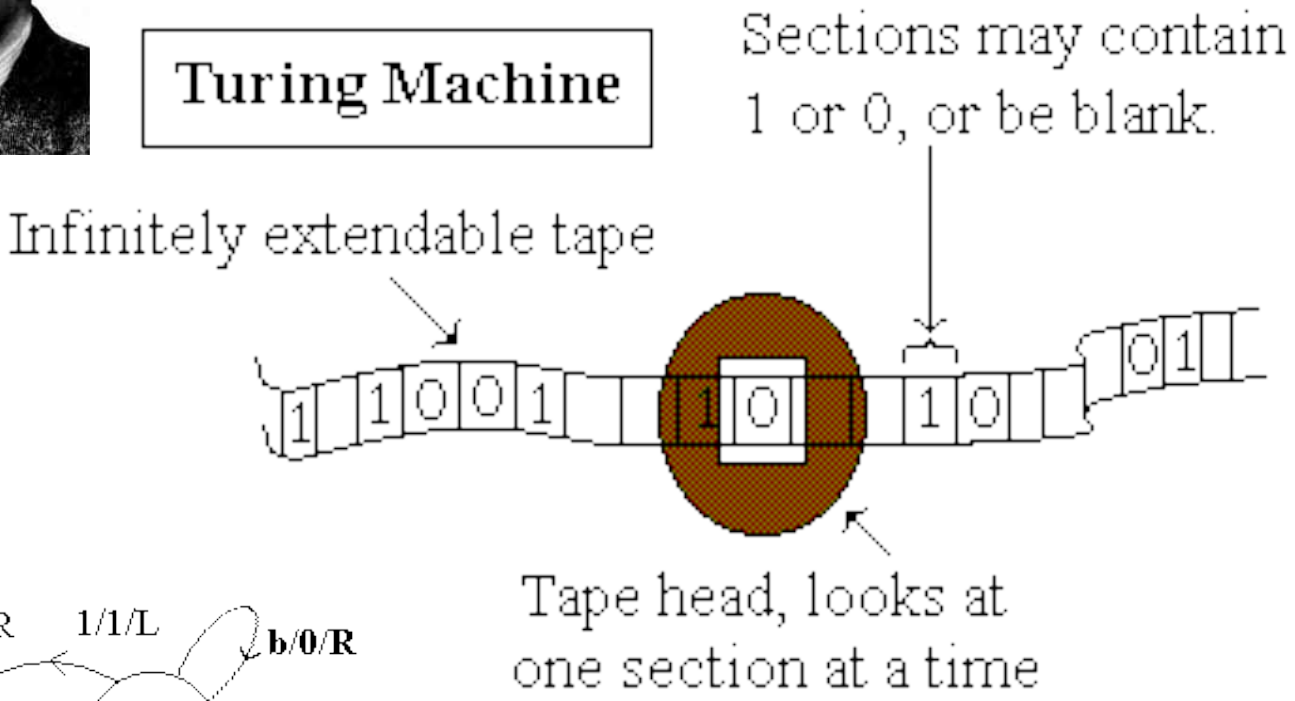


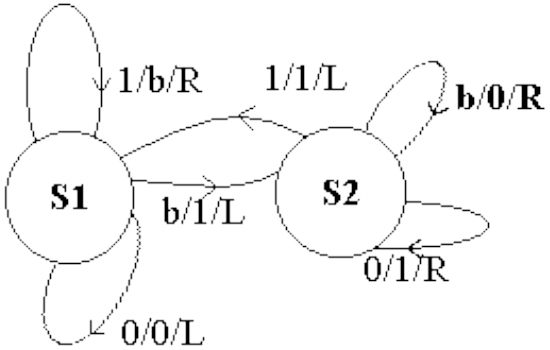
Countess Ada Lovelace
Program for generating Bernoulli Numbers (1842)

12		−	${}^{1}V_{10} - {}^{1}V_{1}$	${}^{2}V_{10}$	${}^{1}V_{10} = {}^{2}V_{10}$ ${}^{1}V_{1} = {}^{1}V_{1}$	$= n - 2 (= 2)$ .....
13		−	${}^{1}V_{6} - {}^{1}V_{1}$	${}^{2}V_{5}$	${}^{1}V_{6} = {}^{2}V_{6}$ ${}^{1}V_{1} = {}^{1}V_{1}$	$= 2n - 1$ .....
14		+	${}^{1}V_{1} + {}^{1}V_{7}$	${}^{2}V_{7}$	${}^{1}V_{1} = {}^{1}V_{1}$ ${}^{1}V_{7} = {}^{2}V_{7}$	$= 2 + 1 = 3$ .....
15		÷	${}^{2}V_{6} \div {}^{2}V_{7}$	${}^{1}V_{5}$	${}^{2}V_{6} = {}^{2}V_{6}$ ${}^{2}V_{7} = {}^{2}V_{7}$	$= \frac{2n-1}{3}$ .....
16		×	${}^{1}V_{5} \times {}^{3}V_{11}$	${}^{4}V_{11}$	${}^{1}V_{5} = {}^{0}V_{5}$ ${}^{3}V_{11} = {}^{4}V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$ .....
17		−	${}^{2}V_{6} - {}^{1}V_{1}$	${}^{3}V_{5}$	${}^{2}V_{6} = {}^{3}V_{6}$ ${}^{1}V_{1} = {}^{1}V_{1}$	$= 2n - 2$ .....
18		+	${}^{1}V_{1} + {}^{2}V_{7}$	${}^{3}V_{7}$	${}^{2}V_{7} = {}^{3}V_{7}$ ${}^{1}V_{1} = {}^{1}V_{1}$	$= 3 + 1 = 4$ .....
19		÷	${}^{3}V_{6} \div {}^{3}V_{7}$	${}^{1}V_{2}$	${}^{3}V_{6} = {}^{3}V_{6}$ ${}^{3}V_{7} = {}^{3}V_{7}$	$= \frac{2n-2}{4}$ .....
20		×	${}^{1}V_{2} \times {}^{4}V_{11}$	${}^{5}V_{11}$	${}^{1}V_{2} = {}^{0}V_{2}$ ${}^{4}V_{11} = {}^{5}V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} =$
21		×	${}^{1}V_{22} \times {}^{5}V_{11}$	${}^{6}V_{12}$	${}^{1}V_{22} = {}^{1}V_{22}$ ${}^{6}V_{12} = {}^{2}V_{12}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n}{}$

Alan Turing (1936)

Turing Machine



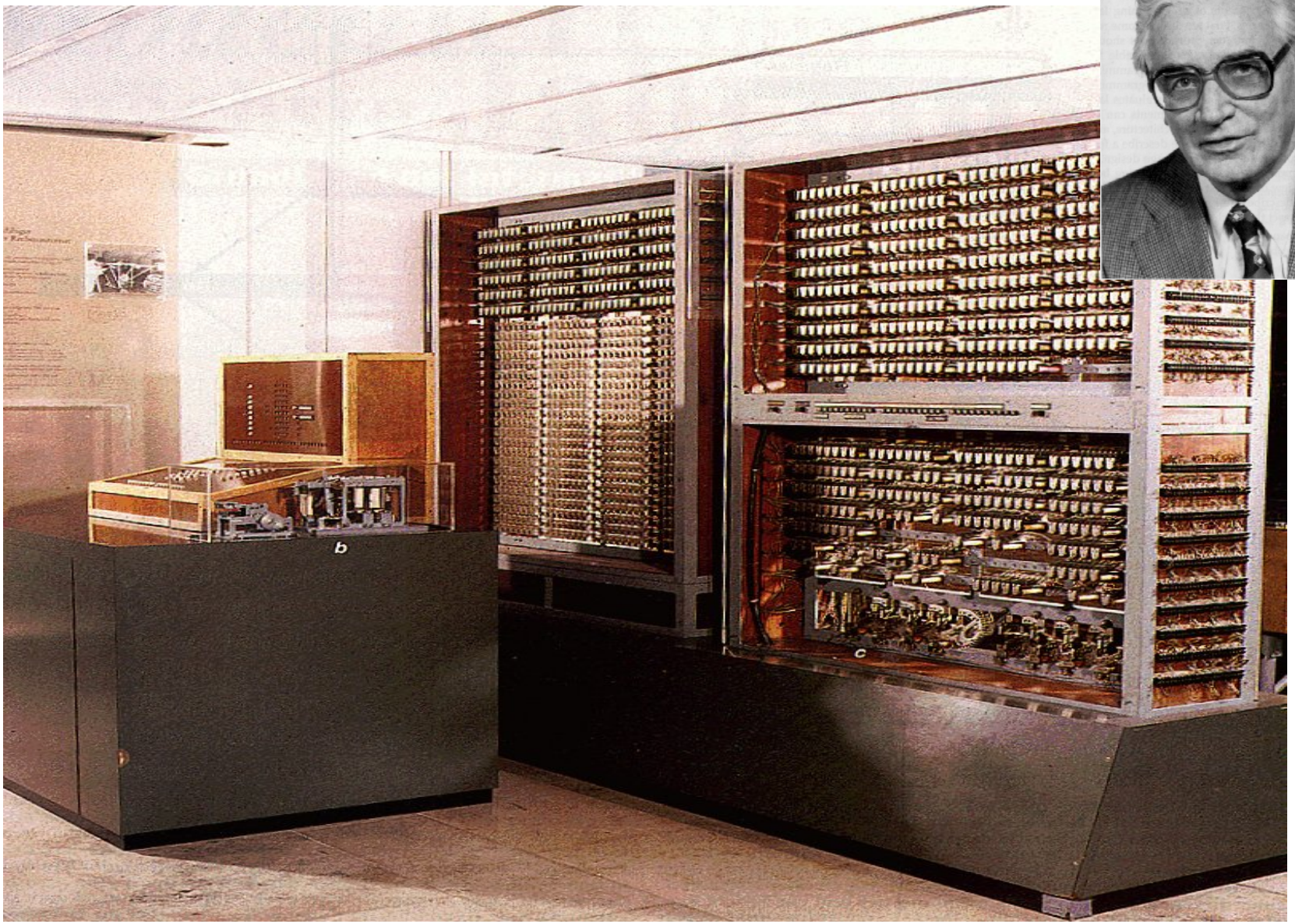


Transition State Diagram of Turing Machine (corresponds to transition state table)

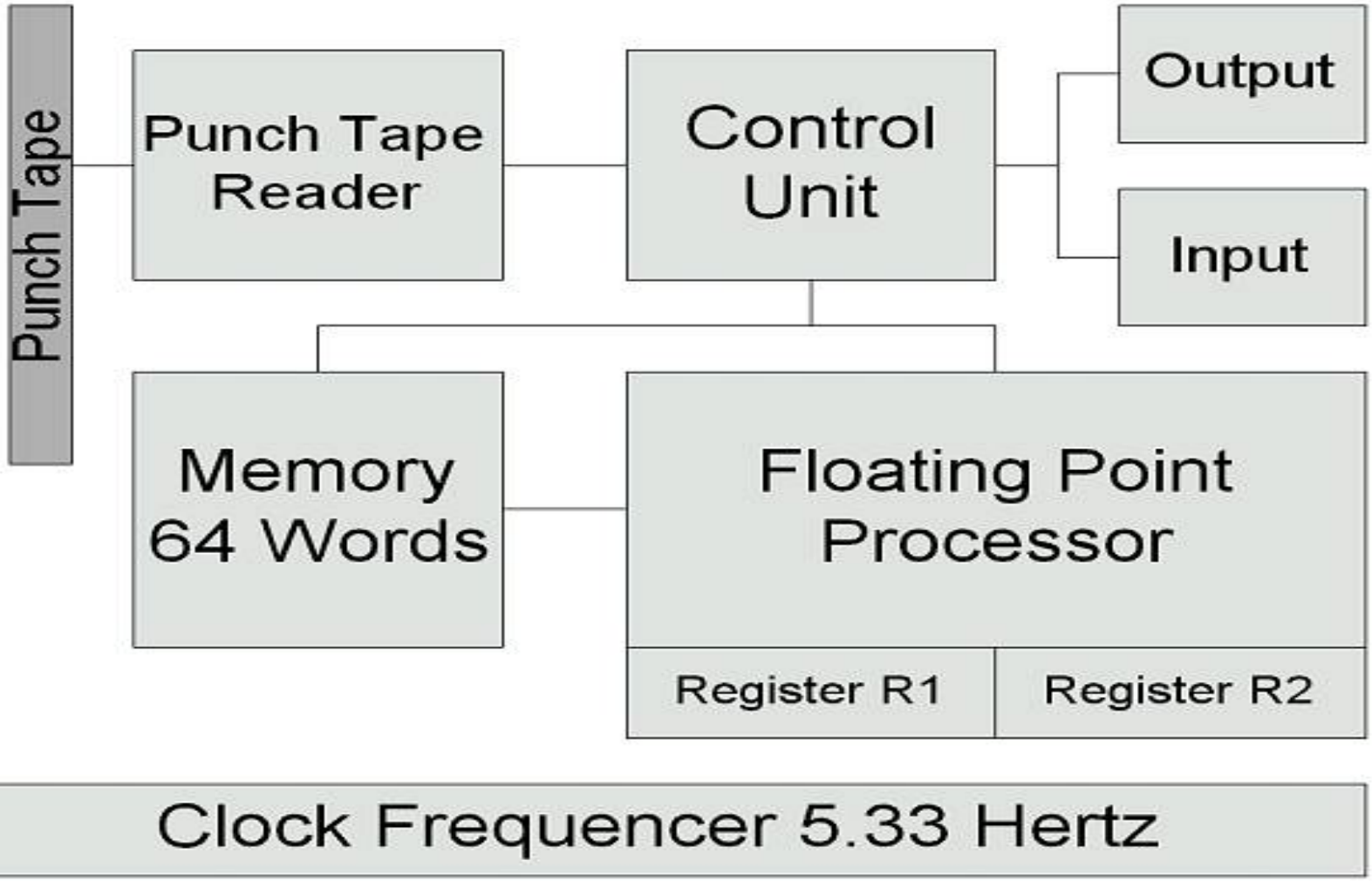
State	Read	Write	Move	Next State
S1	0	0	L	S1
	blank	1	L	S2
	1	blank	R	S1
S2	0	1	R	S2
	blank	0	R	S2
	1	1	L	S1

State Transition Table for a Turing Machine

The Z3 built by Konrad Zuze in Berlin 1941



Architecture of the Z3



Von Neumann Architecture

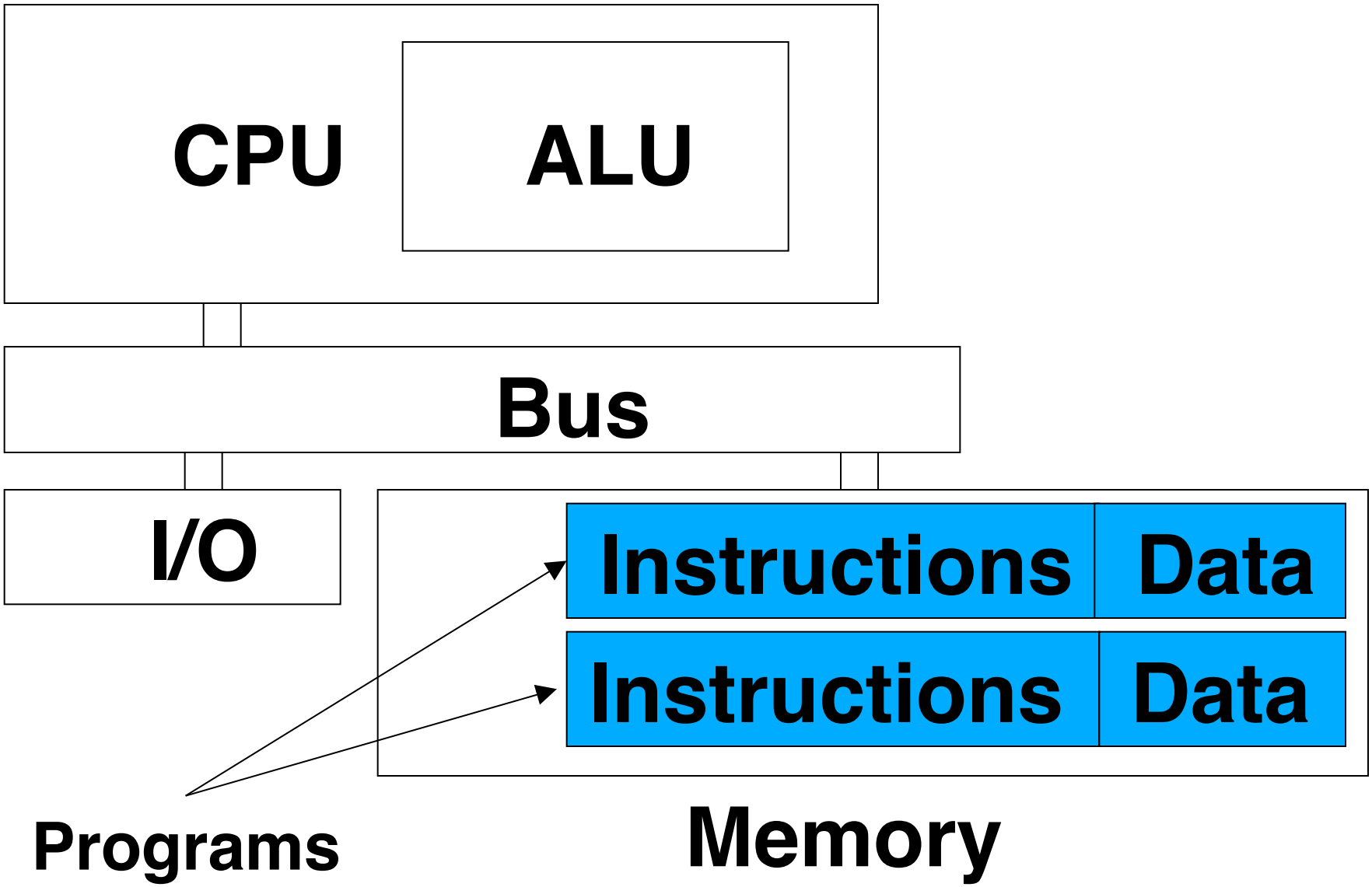
- Johann Von Neumann popularized the Stored Program computer.
- Previous computers (like the COLLOSUS and ENIAC) had the program literally wired in.
- The Z3 had its program on punch tape.
- Stored program computers made changing the algorithm a computer was to execute trivial. (compared to rewiring it!)
- It also meant that the sequence of execution could be changed while the program was running.

Anatomy of a Computer

- Six logical units of modern computers
  - Input (Keyboard, mouse, etc)
  - Output (Monitor, Printer, etc)
  - Volatile Memory (RAM or main memory)
  - Long term Memory (Disk, punch tape, etc)
  - Central Processing Unit (Processor)
  - Arithmetic and Logic Unit (Processor)

  (+ Bus, pipeline so all the other units can communicate)

Basic Organization of a Modern Computer



Anatomy of a Computer - Volatile Memory

- Volatile memory (RAM) consists of numerous (typically millions or billions) of binary digits ("bits").
- A bit can hold the value 1 (the bit is set) or zero (the bit is unset).
- A collection of eight bits can have 256 different states and so can be used to represent different things. For example 256 different characters (A,B, C, … a, b, c, … &, *, $) can be represented with a single byte by defining each different sequence of 1s and 0s to represent a different character.

Anatomy of a Computer - Volatile Memory

- In memory all these bits have to be organized in such a way that we can find them when we want them.
- Memory is divided up into "words," each word is typically two bytes (16 bits).

Anatomy of a Computer - Volatile Memory

- There are other terms for various numbers of bits – for example, half a byte (4 bits) is a nibble, four bytes are called a “long” or a “dword” (double word), 1024 bytes is a kilobyte.
- The more bits a division has the more states it can represent but the more memory it uses.

Anatomy of a Computer - CPU

- The CPU has several local "registers." Registers are memory locations just like RAM.
- The CPU is able to "fetch" values from RAM and place them into its registers.
- Data in the registers can then be operated on by the ALU and CPU.

Anatomy of a Computer - CPU

- The job then of the CPU is to load words into its registers from RAM, perform some operation on the registers and then store the result back into RAM.
- Simply, the list of instructions that tell the processor what data to load and what operations to perform is a computer program.

Anatomy of a Computer – ALU

- The ALU contains arithmetic and logic circuits which allow it to perform simple operations on data bits such as addition, multiplication, equality, etc.

Anatomy of a Computer

- Secondary storage is mapped out in much the same way as main memory (RAM). The media for storing the bits is non-volatile and so does not need constant power. This is where we save our programs when they are not running.
- Input devices can be as diverse as keyboards, scanners, and temperature sensors.
- Output devices are typically monitors, sound cards, and printers.
- Many devices are used for both input and output, such as network cards.

Languages – Machine Language

- Every CPU has a list of actions that it is capable of performing (the instruction set).
- These instructions must be given to the CPU in binary code for it to understand them.
- A typical instruction might be:

  00000001 01100101 10010010 01

Languages – Machine Language, cont

A typical instruction might be:

00000001 01100101 10010010 01

Where 00000001 means “load” (copy), “01100101 10010010” is the address of a word in memory, and 01 is the address of a register.

So this instruction tells the computer to load the value held at a certain address into the first register.

Languages - Assembly Language

Machine language was too difficult to work with so programmers added a layer of abstraction.

They wrote programs to translate keywords into the appropriate machine language.

Now they could write “load 77E814F1 esi” and the “assembler” would translate the code into the 1s and 0s of machine language.

Typical commands are things like load, add, jump, add1, poke, and, xor, etc.

Languages – Assembly Language

Typical snapshot of assembly language on a Pentium IV

Address	Instruction	Register or RAM Address
77E814EE	mov	esi,dword ptr [edi+8]
77E814F1	mov	dword ptr [ebp+64h],0Ah
77E814F8	add	esi,4Ah
77E814FB	jmp	77E7E91A
77E81500	push	ebx
77E81501	xor	ebx,ebx
77E81503	cmp	ecx,ebx
77E81505	push	esi
77E81506	push	edi

Languages – High Level Languages

- A lot of sophisticated software was written using machine and assembly language but it was still too difficult to understand.
- High level languages have added a further level of abstraction such that a single command in Fortran, Java, or C ++ might be translated into hundreds or thousands of assembly language instructions.

Languages – High Level Languages

- High level languages also allow the programmer to define new commands in terms of old ones.
- This means that most high level languages are unlimited in expressive power and in their potential to abstract away complexity.
- Once you have one high level language it become easy to use it to write other high level languages.

Languages - High Level Languages

- The first high level language was created by John Backus at IBM, in 1954.
- It was originally called SpeedCoding but later the name was changed to the Formula Translation Language or Fortran.
- In 1958 John McCarthy developed the List Processing Language or Lisp. Algol, Fortran III, Flow-Matic which became Cobol all came out the same year.
- There are now well over 200 major high level programming languages in 26 different groups.

A Brief History of High Level Languages

- C
  - Evolved from two other programming languages
    - BCPL and B
      - "Typeless" languages
  - Dennis Ritchie (Bell Laboratories)
    - Added data typing, other features
  - Development language of UNIX
  - Hardware independent
    - Portable programs
  - 1989: ANSI standard
  - 1990: ANSI and ISO standard published
    - ANSI/ISO 9899: 1990

A Brief History of High Level Languages

- History of C++
  - Extension of C
  - Early 1980s: Bjarne Stroustrup (Bell Laboratories)
  - "Spruces up" C
  - Provides capabilities for object-oriented programming
    - Objects: reusable software components
      - Model items in real world
    - Object-oriented programs
      - Easy to understand, correct and modify
  - Hybrid language
    - C-like style
    - Object-oriented style
    - Both