BBM 101 – Introduction to Programming I

Fall 2014, Lecture 2

Aykut Erdem, Erkut Erdem, Fuat Akal

Today

- **Stored program computers**
  - Components of a computer
  - Von Neumann architecture
- **Programs and Data**
  - Binary data representations
  - Bit operations
- **Programming languages (PLs)**
  - Syntax and semantics
  - Dimensions of a PL
  - Programming paradigms
  - A partial history of PLs

Today

- **Stored program computers**
  - Components of a computer
  - Von Neumann architecture
- **Programs and Data**
  - Binary data representations
  - Bit operations
- **Programming languages**
  - Syntax and semantics
  - Dimensions of a PL
  - Programming paradigms
  - A brief history of PLs

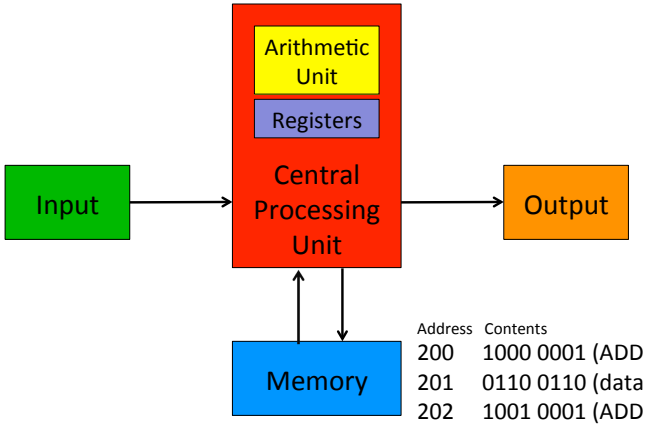
Recall: Stored Program Concept

- Stored-program concept is the fundamental principle of the ENIAC's successor, the EDVAC (Electronic Discrete Variable Automatic Computer)
- Instructions were stored in memory sequentially with their data
- Instructions were executed sequentially except where a conditional instruction would cause a jump to an instruction someplace other than the next instruction

Slide credit: G. Kesden

Stored Program Concept

- "Fetch-Decode-Execute" cycle



Address	Contents
200	1000 0001 (ADD to R1)
201	0110 0110 (data value 102)
202	1001 0001 (ADD to R1)
203	0110 0110 (data at address 102)
204	1111 0111 (JUMP 7 bytes)

Stored Program Concept

- Mauchly and Eckert are generally credited with the idea of the stored-program
- BUT: John von Neumann publishes a draft report that describes the concept and earns the recognition as the inventor of the concept
  - "von Neumann architecture"
  - A First Draft of a Report of the EDVAC published in 1945
  - http://www.wps.com/projects/EDVAC/

von Neumann, Member of the Navy Bureau of Ordinance 1941-1955

The Integrated Circuit

- Robert Noyce and Jack Kilby are credited with the invention of the integrated circuit (IC) or microchip.
  - Kilby wins Nobel Prize in Physics in 2000.
  - Robert Noyce co-founded Intel in 1968.
- By the mid 1970s, ICs contained tens of thousands of transistors per chip.
  - In 1970, Intel created the 1103--the first generally available DRAM chip.
  - Today, you would need more than 65,000 of them to put 8 MB of memory into a PC.

Units of Memory

Byte	B	8 bits (8b)		
Kilobyte	KB	1024 B	= 2^{10} bytes	≈ 10^3 bytes
Megabyte	MB	1024 KB	= 2^{20} bytes	≈ 10^6 bytes
Gigabyte	GB	1024 MB	= 2^{30} bytes	≈ 10^9 bytes
Terabyte	TB	1024 GB	= 2^{40} bytes	≈ 10^{12} bytes
Petabyte	PB	1024 TB	= 2^{50} bytes	≈ 10^{15} bytes

- How many bytes can be stored in a 4GB flash drive?
- How many bytes/second is a 16 Mbps cable modem connection?

How Time Flies

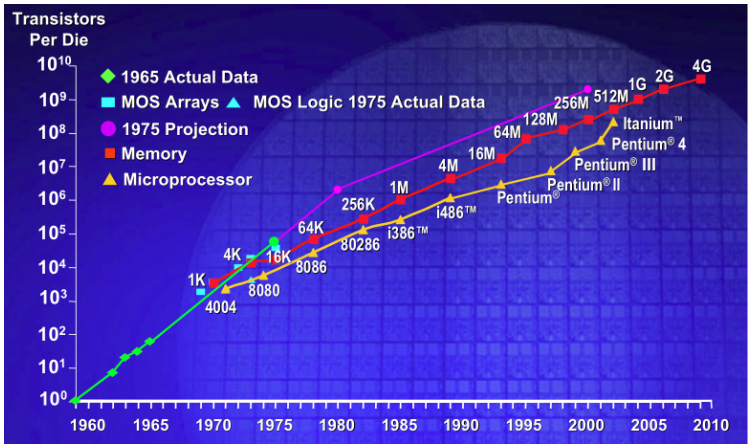
Commodore 64 (1982)
40 cm X 22 cm X 8 cm
64 KB of IC memory
$595

Apple iShuffle (2008)
3 cm X 3 cm X 1 cm
2 GB of flash memory
$49

Moore's Law

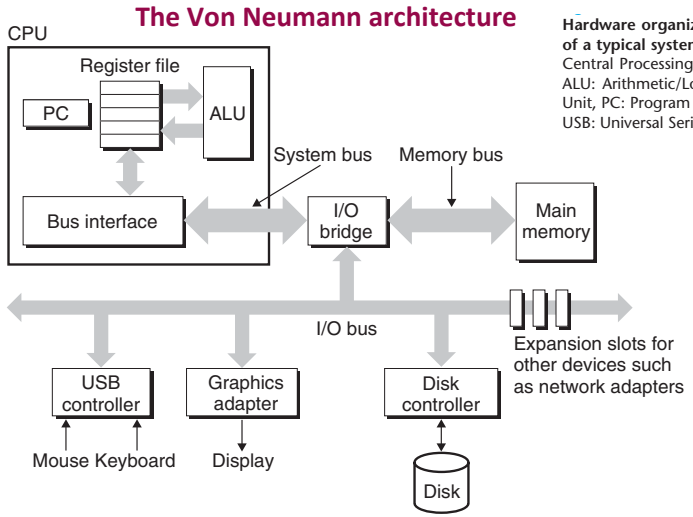
- Gordon Moore co-founded Intel Corporation in 1968.
- Famous for his prediction on the growth of the semiconductor industry: Moore's Law
  - ftp://download.intel.com/research/silicon/moorespaper.pdf
  - An empirical observation stating in effect that the complexity of integrated circuits doubles every 18 months. ("complexity" generally means number of transistors on a chip)

Moore's Law



Source: Intel Corporation

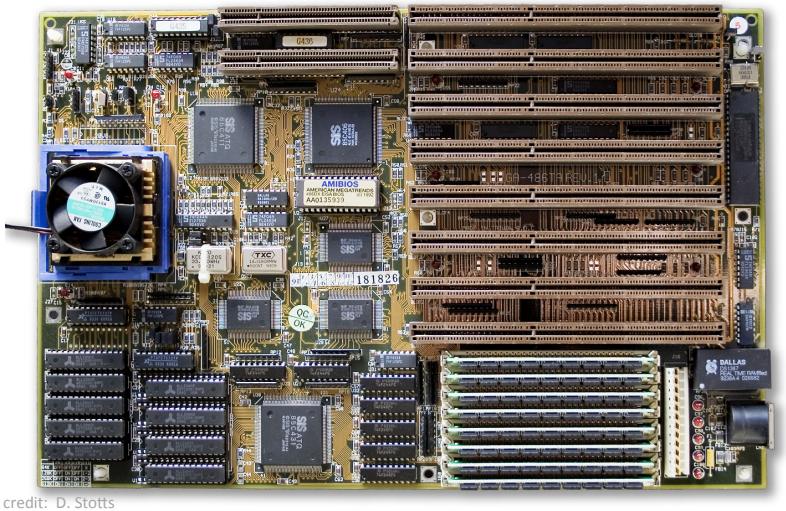
A detailed view



Hardware organization of a typical system. CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program counter, USB: Universal Serial Bus.

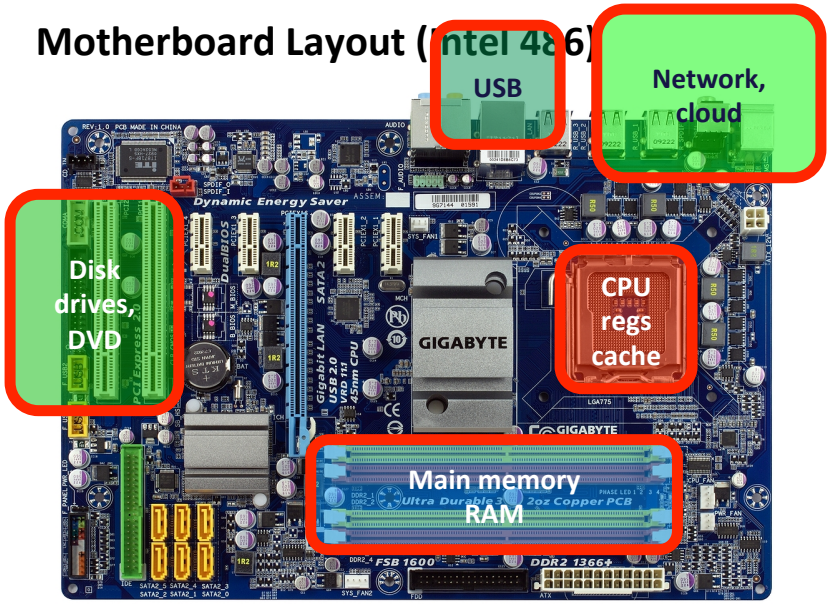
Image: R.E. Bryant, D.R. O'Hallaron, G. Kesden

Motherboard Layout (Intel 486)



Slide credit: D. Stotts

Motherboard Layout (Intel 486)

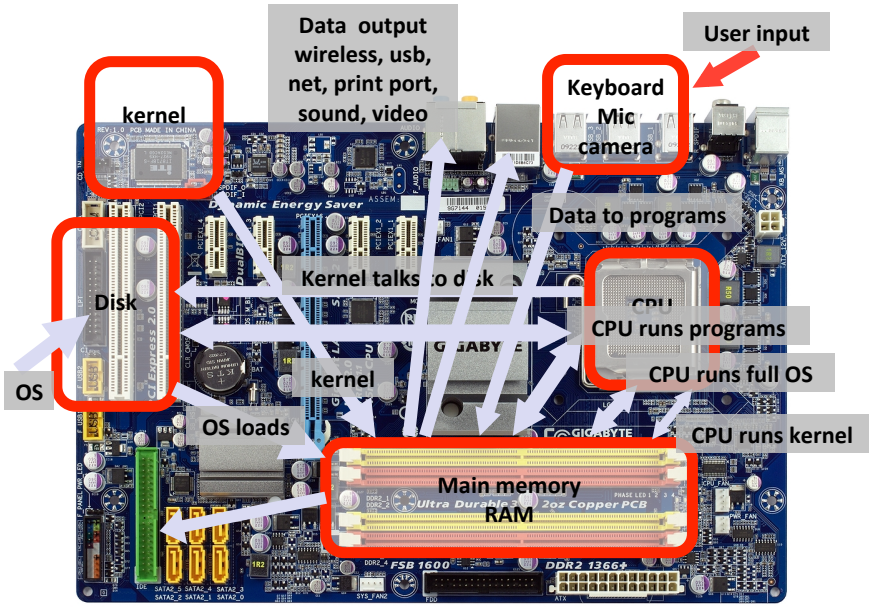


Slide credit: D. Stotts

What happens when you power on your computer?

- A minimum amount of information is read from secondary memory into main memory
- Control is transferred to that area of main memory; this code reads the core of the OS, called the ***kernel***
- The kernel executes the initial process
- This process loads a full OS off disk (or cloud)
- Called ***bootstrapping*** (pulling oneself up by one's bootstraps)… the computer ***"boots up"***
- OS then runs all the other programs you write and use…

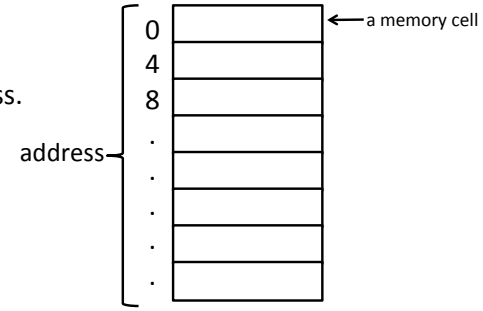
Slide credit: D. Stotts



Slide credit: D. Stotts

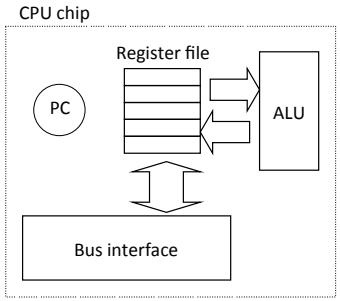
Memory

- Fixed-size units (cells) to store data
- Each cell has its own address.
- Volatile vs. non-volatile
  - Volatile memories lose information if powered off (e.g. RAM)
  - Nonvolatile memories retain value even if powered off (e.g. ROM, cache memory, SSDs)



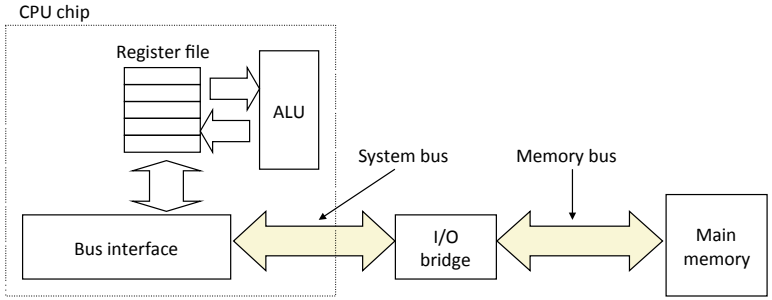
Arithmetic-Logic Unit (ALU)

- ALU + Control = Processor
- **Registers.** Storage cells that holds heavily used program data
  - Without address, specific purpose
  - **e.g.** the operands of an arithmetic operation, the result of an operation, etc.



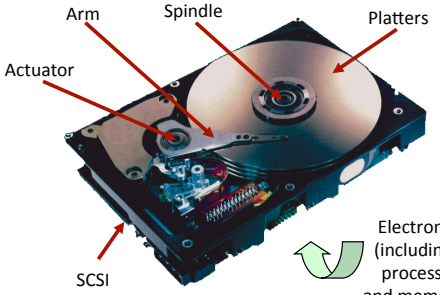
Traditional Bus Structure

- A bus is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.



Disk Drives

Magnetic disk drives
- Rotating disks



A solid-state drives (SSDs)
- No moving parts

Image courtesy of Seagate Technology

The Memory Hierarchy

- Storage devices form a hierarchy
- Storage at one level serves as a cache for storage at the next lower level.

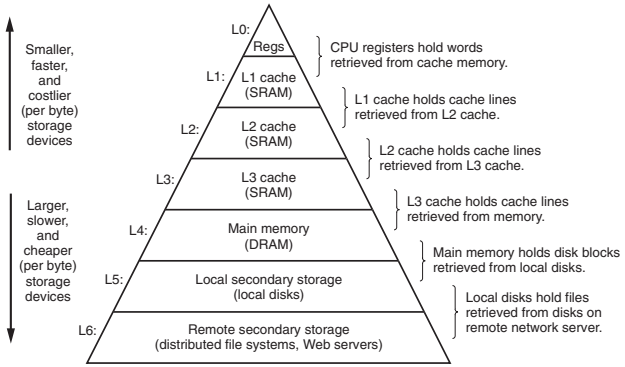
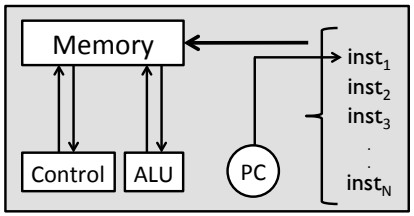


Image: R.E. Bryant, D.R. O'Hallaron, G. Kesden

Components of a Computer (cont'd)

- Sequential execution of machine instructions
  - The sequence of instructions are stored in the memory.
  - One instruction at a time is fetched from the memory to the control unit.
  - They are read in and treated just like data.



- PC (program counter) is responsible from the flow of control.
- PC points a memory location containing an instruction on the sequence.

- Early programmers (coders) write programs via machine instructions.

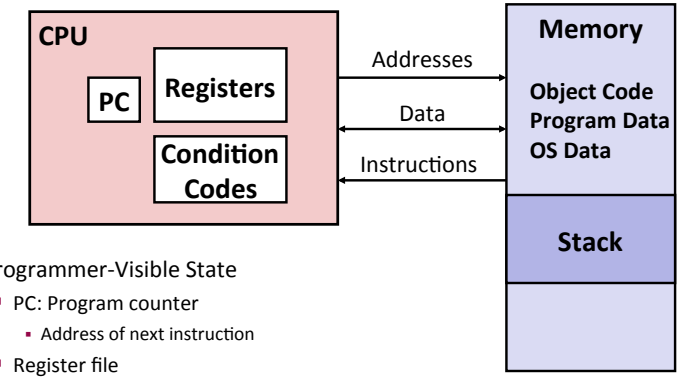
Assembly Language

- A low-level programming language for computers
- More readable, English-like abbreviations for instructions
- Architecture-specific

- Example:

```
MOV AL, 61h
MOV AX, BX
ADD EAX, 10
XOR EAX, EAX
```


Assembly Programmer's View



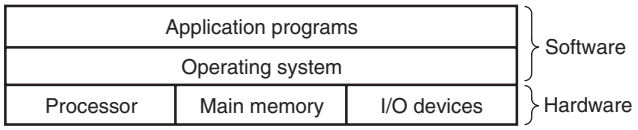
- Programmer-Visible State
  - PC: Program counter
    - Address of next instruction
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code, user data, (some) OS data
  - Includes stack used to support procedures

Adopted from: R.E. Bryant, D.R. O'Hallaron, G. Kesden

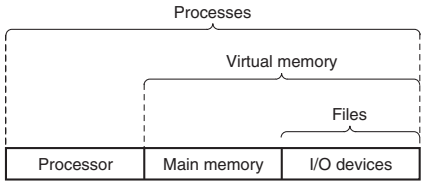
The Operating System (OS)

- The operating system is a layer of software interposed between the application program and the hardware.
  - Unix, Linux, Mac OS, Windows, etc.
- All attempts by an application program to manipulate the hardware must go through the OS.
- The operating system has two primary purposes:
  1. To protect the hardware from misuse by runaway applications, and
  2. To provide applications with simple and uniform mechanisms for manipulating complicated and often wildly different low-level hardware devices.



Abstractions provided by an OS

- The operating system achieves both goals via the fundamental "abstractions"
  - **Processes:** The operating system's abstraction for running programs.
  - **Virtual Memory:** An abstraction that provides each process with the illusion that it has exclusive use of the main memory.
  - **Files:** Abstractions for I/O devices

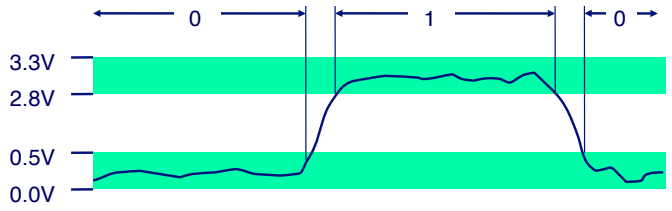


Today

- Stored program computers
  - Components of a computer
  - Von Neumann architecture
- **Programs and Data**
  - Binary data representations
  - Bit operations
- Programming languages
  - Syntax and semantics
  - Dimensions of a PL
  - Programming paradigms
  - A partial history of PLs

Binary Data Representations

- **Binary** - taking two values
- **bit = 0 or 1**
  - False or True
  - Off or On
  - Low voltage or High voltage



Encoding Byte Values

- **Byte = 8 bits**
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters '0' to '9' and 'A' to 'F'

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

Byte-Oriented Memory Organization

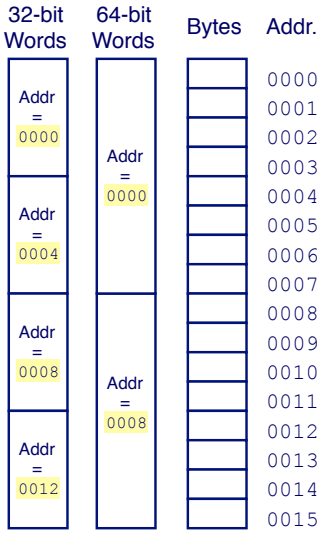
- Programs Refer to Virtual Addresses
  - Conceptually very large array of bytes
  - System provides address space private to particular "process"
    - Program being executed
    - Program can clobber its own data, but not that of others
- Compiler + Run-Time System Control Allocation
  - Where different program objects should be stored
  - All allocation within single virtual address space

Machine Words

- Machine Has "Word Size"
  - Nominal size of integer-valued data
    - Including addresses
  - Most current machines use 32 bits (4 bytes) words
    - Limits addresses to 4GB
    - Becoming too small for memory-intensive applications
  - High-end systems use 64 bits (8 bytes) words
    - Potential address space ≈ 1.8 X 10^{19} bytes
    - x86-64 machines support 48-bit addresses: 256 Terabytes
  - Machines support multiple data formats
    - Fractions or multiples of word size
    - Always integral number of bytes

Word-Oriented Memory Organization

- Addresses Specify Byte Locations
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)



von Neumann revisited

Programs and their data all stored together in memory!

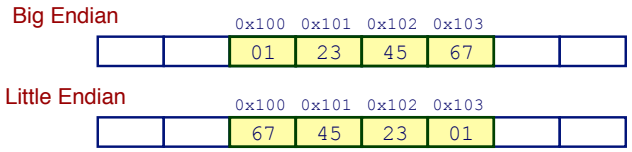
- Some 0's 1's chunks stand for numbers
- Some stand for characters, text
- Some stand for images, videos, etc.
- Some stand for memory locations
- Some stand for program instructions like "add 2 numbers" or "save register 5 to memory location 2145768"
- Computer sorts it all out during the fetch-execute cycle

Byte Ordering

- How should bytes within a multi-byte word be ordered in memory?
- Conventions
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
  - Little Endian: x86
    - Least significant byte has lowest address

Byte Ordering Example

- Big Endian
  - Least significant byte has highest address
- Little Endian
  - Least significant byte has lowest address
- Example
  - Assume we have a 4-byte data representation 0x01234567 stored at the address 0x100



The origin of "endian"

"Gulliver finds out that there is a law, proclaimed by the grandfather of the present ruler, requiring all citizens of Lilliput to break their eggs only at the little ends. Of course, all those citizens who broke their eggs at the big ends were angered by the proclamation. Civil war broke out between the Little-Endians and the Big-Endians, resulting in the Big-Endians taking refuge on a nearby island, the kingdom of Blefuscu."

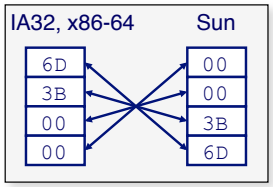
Illustration by Chris Beatrice

– Danny Cohen, *On Holy Wars and A Plea For Peace,* 1980

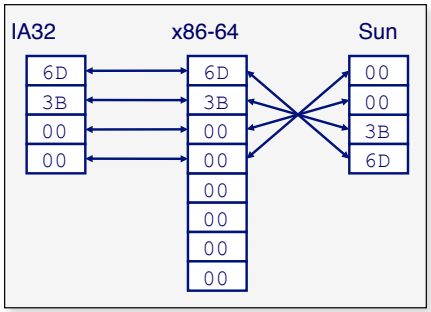
Representing Integers

Decimal:	15213
Binary:	0011 1011 0110 1101
Hex:	3 B 6 D

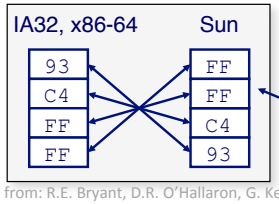
```
int A = 15213;
```




```
long int C = 15213;
```

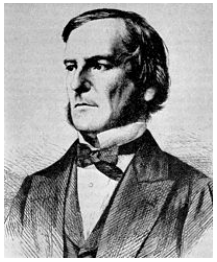



```
int B = -15213;
```



Two's complement representation
(will be covered in BBM 231)

Boolean Algebra



- Developed by George Boole in 19th Century
  - Algebraic representation of logic
    - Encode "True" as 1 and "False" as 0

And

- A&B = 1 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or

- A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not

- ~A = 1 when A=0

~	
0	1
1	0

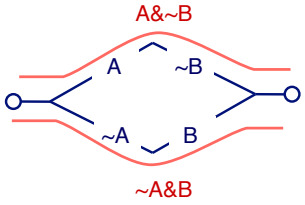
Exclusive-Or (Xor)

- A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

Application of Boolean Algebra

- Applied to Digital Systems by Claude Shannon
  - 1937 MIT Master's Thesis
  - Reason about networks of relay switches
    - Encode closed switch as 1, open switch as 0



Connection when

A&~B | ~A&B

= A^B

General Boolean Algebras

- Operate on Bit Vectors
  - Operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```

- All of the Properties of Boolean Algebra Apply

Representing & Manipulating Sets

- Representation
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$
    - 01101001 { 0, 3, 5, 6 }
    - *76543210*
    - 01010101 { 0, 2, 4, 6 }
    - *76543210*
- Operations

&	Intersection	01000001	{ 0, 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
~	Complement	10101010	{ 1, 3, 5, 7 }

Bit-Level Operations

- Operations &, |, ~, ^
- Examples (Char data type)
  - `~0x41` → `0xBE`
    - $\sim 01000001_2 \rightarrow 10111110_2$
  - `~0x00` → `0xFF`
    - $\sim 00000000_2 \rightarrow 11111111_2$
  - `0x69 & 0x55` → `0x41`
    - $01101001_2 \;\&\; 01010101_2 \rightarrow 01000001_2$
  - `0x69 | 0x55` → `0x7D`
    - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

Shift Operations

- Left Shift: x << y
  - Shift bit-vector x left y positions
    - Throw away extra bits on left
  - Fill with 0's on right
- Right Shift: x >> y
  - Shift bit-vector x right y positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate most significant bit on right
- Undefined Behavior
  - Shift amount < 0 or ≥ word size

Argument x	01100010
<< 3	00010000
Log. >> 2	00011000
Arith. >> 2	00011000

Argument x	10100010
<< 3	00010000
Log. >> 2	00101000
Arith. >> 2	11101000

Today

- **Stored program computers**
  - Components of a computer
  - Von Neumann architecture
- **Programs and Data**
  - Binary data representations
  - Bit operations
- **Programming languages (PLs)**
  - Syntax and semantics
  - Dimensions of a PL
  - Programming paradigms
  - A partial history of PLs

Programming Languages

- an artificial language designed to express computations that can be performed by a machine, particularly a computer.
- can be used to create programs that control the behavior of a machine, to express algorithms precisely, or as a mode of human communication.
- **e.g.**, C, C++, Java, Python, Prolog, Haskell, Scala, etc..

Creating computer programs

- Each programming language provides a set of primitive operations
- Each programming language provides mechanisms for combining primitives to form more complex, but legal, expressions
- Each programming language provides mechanisms for deducing meanings or values associated with computations or expressions

Aspects of languages

- Primitive constructs
  - Programming language – numbers, strings, simple operators
  - English – words
- Syntax – which strings of characters and symbols are well-formed
  - Programming language –we'll get to specifics shortly, but for example 3.2 + 3.2 is a valid C expression
  - English – "cat dog boy" is not syntactically valid, as not in form of acceptable sentence

Aspects of languages

- Static semantics – which syntactically valid strings have a meaning
  - English – "I are big" has form <noun> <intransitive verb> <noun>, so syntactically valid, but is not valid English because "I" is singular, "are" is plural
  - Programming language – for example, <literal> <operator> <literal> is a valid syntactic form, but 2.3/'abc' is a static semantic error

Aspects of languages

- Semantics – what is the meaning associated with a syntactically correct string of symbols with no static semantic errors
  - English – can be ambiguous
    - "They saw the man with the telescope."
  - Programming languages – always has exactly one meaning
    - But meaning (or value) may not be what programmer intended

Where can things go wrong?

- Syntactic errors
  - Common but easily caught by computer
- Static semantic errors
  - Some languages check carefully before running, others check while interpreting the program
  - If not caught, behavior of program unpredictable
- Programs don't have semantic errors, but meaning may not be what was intended
  - Crashes (stops running)
  - Runs forever
  - Produces an answer, but not programmer's intent

Our goal

- Learn the syntax and semantics of a programming language
- Learn how to use those elements to translate "recipes" for solving a problem into a form that the computer can use to do the work for us
- Computational modes of thought enable us to use a suite of methods to solve problems

Dimensions of a Programming Language

- **Low-level vs. High-level**
  - Distinction according to the level of abstraction
  - In low-level programming languages (e.g. Assembly), the set of instructions used in computations are very simple (nearly at machine level)
  - A high-level programming language (e.g. C, Java) has a much richer and more complex set of primitives.

Dimensions of a Programming Language

- **General vs. Targeted**
  - Distinction according to the range of applications
  - In a general programming language, the set of primitives support a broad range of applications.
  - A targeted programming language aims at a very specific set of applications.
    - **e.g.**, MATLAB (matrix laboratory) is a programming language specifically designed for numerical computing (matrix and vector operations)

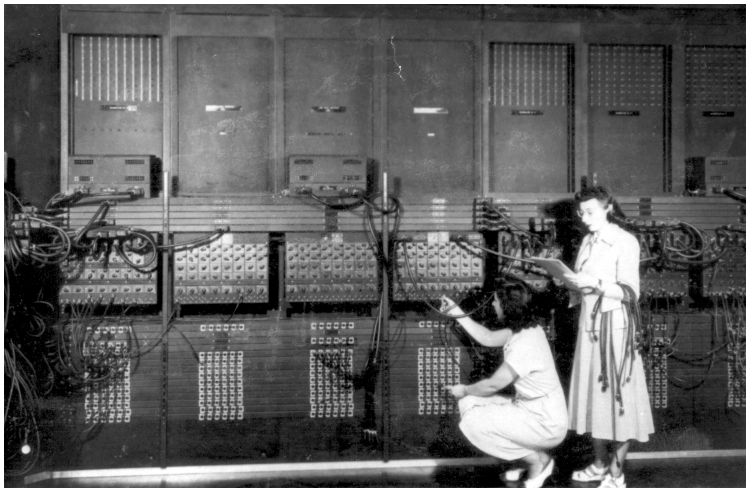
Dimensions of a Programming Language

- **Interpreted vs. Compiled**
  - Distinction according to how the source code is executed
  - In interpreted languages (e.g. LISP), the source code is executed directly at runtime (by the interpreter).
    - Interpreter control the the flow of the program by going through each one of the instructions.
  - In compiled languages (e.g. C), the source code first needs to be translated to an object code (by the compiler) before the execution.

Programming Language Paradigms

- **Functional**
  - Treats computation as the evaluation of mathematical functions (e.g. Lisp, Scheme, Haskell, etc.)
- **Imperative**
  - describes computation in terms of statements that change a program state (e.g. FORTRAN, BASIC, Pascal, C, etc.)
- **Logical (declarative)**
  - expresses the logic of a computation without describing its control flow (e.g. Prolog)
- **Object oriented**
  - uses "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs (e.g. C++, Java, C#, Python, etc.)

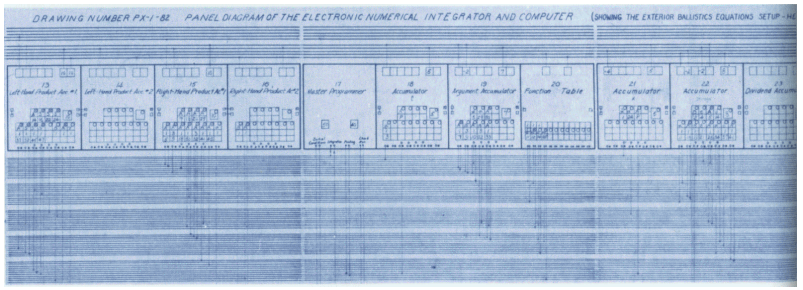
Programming the ENIAC



Slide credit: Ras Bodik

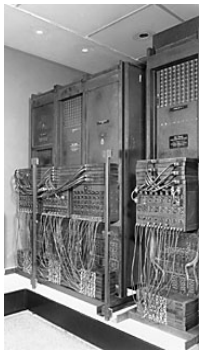
Programming the ENIAC

ENIAC program for external ballistic equations:



Programming the ENIAC

- Programming done by
  - rewiring the interconnections
  - to set up desired formulas, etc
- Problem (what's the tedious part?)
  - programming = rewiring
  - slow, error-prone
- Solution:
  - store the program in memory!
  - birth of *von Neumann* paradigm



The first assembler

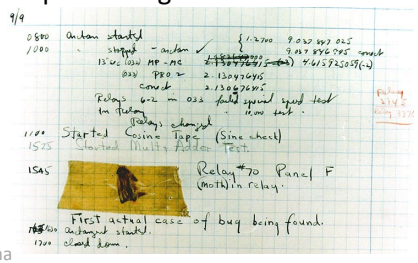
- Assembler - a computer program for translating assembly language into executable machine code
  - Example: `ADD R1, R2, R3 0110000100100011`
- The EDSAC programming system was based on a subroutine library
  - commonly used functions that could be used to build all sorts of more complex programs
  - the first version, Initial Orders 1, was devised by David Wheeler, then a research student, in 1949
- Team published "The Preparation of Programs for an Electronic Digital Computer"
  - the only programming textbook then available
  - computers today still use Cambridge model for subroutines library

The first compiler

- A compiler is a computer program that translates a computer program written in one computer language (the *source* language) into a program written in another computer language (the *target* language).
  - Typically, the target language is assembly language
  - Assembler may then translate assembly language into machine code
- A-0 is a programming language for the UNIVAC I or II, using three-address code instructions for solving mathematical problems.
- A-0 was the first language for which a compiler was developed.

The first compiler

- A-0 was produced by Grace Hopper's team at Remington Rand in 1952
  - Grace Hopper had previously been a programmer for the Harvard Mark machines
  - One of U.S.'s first programmers
  - She found a moth in the Mark I, which was causing errors, and called it a computer "bug"



FORTRAN (1957)

- First successful high-level programming language
  - Code more readable and understandable by humans
  - Developed by John Bachus at IBM
  - Stands for: FORmula TRANslation
  - Started development in 1954
- A key goal of FORTRAN was efficiency, although portability was also a key issue
  - automatic programming that would be as good as human programming of assembly code
- Programs that took weeks to write could now take hours
- 1961 – First FORTRAN programming textbook
  - Universities began teaching it in undergrad programs
- Provided standard exchange of programs despite different computers
- Became the standard for scientific applications

FORTRAN

```
REAL SUM6,SUM7,SUM8,DIF6,DIF7,DIF8,SUMINF
OPEN(6,FILE='PRN')
SUM6=.9*(1.-0.1**6)/0.9
SUM7=.9*(1.-0.1**7)/0.9
SUM8=.9*(1.-0.1**8)/0.9
******COMPUTER SUM OF INFINITE TERMS
SUMINF=0.9/(1.0-0.1)
******COMPUTE DIFFERENCES BETWEEN FINITE & INFINITE SUMS
DIF6 = SUMINF - SUM6
DIF7 = SUMINF - SUM7
DIF8 = SUMINF - SUM8
WRITE(6,*) 'INFINITE SUM = ', SUMINF
WRITE(6,*) 'SUM6 = ', SUM6, ' INFINITE SUM - SUM6 = ', DIF6
WRITE(6,*) 'SUM7 = ', SUM7, ' INFINITE SUM - SUM7 = ', DIF7
WRITE(6,*) 'SUM8 = ', SUM8, ' INFINITE SUM - SUM8 = ', DIF8
STOP
```


COBOL (1960)

- Stands for: Common Business-Oriented Language
- COBOL was initially created in 1959 (and released in 1960 as Cobol 60) by a group of computer manufacturers and government agencies
- One goal of COBOL's design was for it to be readable by managers, so the syntax had very much of an English-like flavor.
  - The specifications were to a great extent inspired by the FLOW-MATIC language invented by Grace Hopper
- Became the standard for business applications
  - Still used in business applications today.
- 90% of applications over next 20 years were written in either COBOL or FORTRAN
  - Old programmers came out of hiding for Y2K

COBOL

```
000100 ID DIVISION.
000200 PROGRAM-ID. ACCEPT1.
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500 01 WS-FIRST-NUMBER PIC 9(3).
000600 01 WS-SECOND-NUMBER PIC 9(3).
000700 01 WS-TOTAL PIC ZZZ9.
000800*
000900 PROCEDURE DIVISION.
001000 0000-MAINLINE.
001100 DISPLAY 'ENTER A NUMBER: '.
001200 ACCEPT WS-FIRST-NUMBER.
001300*
001400 DISPLAY 'ANOTHER NUMBER: '.
001500 ACCEPT WS-SECOND-NUMBER.
001600*
001700 COMPUTE WS-TOTAL = WS-FIRST-NUMBER + WS-SECOND-NUMBER.
001800 DISPLAY 'THE TOTAL IS: ', WS-TOTAL.
001900 STOP RUN.
```


Living & Dead Languages

- Hundreds of programming languages popped up in the 1960s, most quickly disappeared
- Some dead:
  - JOVIAL, SNOBOL, Simula-67, RPG, ALGOL, PL/1, and many, many more
- Some still kicking:
  - LISP (1957)
  - BASIC (1964)
  - Pascal (1970)
  - Prolog (1972)
  - And of course, C (1973)

ALGOL-60 (1960)

- Created mainly in Europe by a committee of computer scientists
  - John Backus and Peter Naur both served on the committee which created it
  - Desired an IBM-independent standard
- Stands for: ALGOrithmic Language
- Primarily intended to provide a mechanism for expressing algorithms uniformly regardless of hardware
- The first report on Algol was issued in 1958,
- The language itself was not a success, but it was an influence on other successful languages
  - A primary ancestor of Pascal and C.
- It introduced block structure, compound statements, recursive procedure calls, nested if statements, loops, and arbitrary length identifiers

LISP (1958)

- Developed by John McCarthy at MIT
- Stands for: LISt Processing
  - Designed for symbolic processing
  - Introduced symbolic computation and automatic memory management
- Used extensively for Artificial Intelligence applications

BASIC (1964)

- Created by John Kemeny and Thomas Kurtz at Dartmouth College
- Stands for: Beginner's All-purpose Symbolic Instruction Code
  - one of the first languages designed for use on a time-sharing system
  - one of the first languages designed for beginners
- Variants like Visual BASIC still used today by Microsoft.

Prolog (1972)

- Created by Alain Colmerauer and Phillipe Roussel of the University of Aix-Marseille and Robert Kowalski of the University of Edinburgh
- Stands for: PROgramming in LOGic.
- Prolog is the leading *logical* programming language.
  - used in artificial intelligence programs, computer linguistics, and theorem proving.

Prolog

```
parents(william, diana, charles).
parents(henry, diana, charles).
parents(charles, elizabeth, philip).
parents(diana, frances, edward).
parents(anne, elizabeth, philip).
parents(andrew, elizabeth, philip).
parents(edwardW, elizabeth, philip).
married(diana, charles).
married(elizabeth, philip).
married(frances, edward).
married(anne, mark).
parent(C,M) <= parents(C,M,D).
parent(C,D) <= parents(C,M,D).
sibling(X,Y) <= parents(X,M,D) and parents(Y,M,D).
```


C (1973)

- Developed by Ken Thompson and Dennis Ritchie at AT&T Bell Labs for use on the UNIX operating system.
  - now used on practically every operating system
  - popular language for writing system software
- Features:
  - An extremely simple core language, with non-essential functionality provided by a standardized set of library routines.
  - Low-level access to computer memory via the use of pointers.
- C ancestors: C++, C#, Java

C++ (C with Classes)

- Bjarne Stroustrup began work on C with Classes in 1979, renamed C++ in 1982.
  - Developed at AT&T Bell Laboratories.
  - Added features of Simula to C.
  - Contained basic object-oriented features:
    - classes (with data encapsulation), derived classes, virtual functions and operator overloading
- In 1989, release 2.0 added more features:
  - multiple inheritance, abstract classes, static member functions, and protected members
- Standard Template Library (STL) official in 1995

Slide credit: Thomas J. Cortina

Java

- Created by Patrick Naughton and James Gosling at Sun Microsystems
  - Originally designed for small consumer devices
  - Original project code name: Green
  - Main feature: Code is generated for a virtual machine that can run on any computer with an appropriate interpreter
  - Original name of the language: Oak

Slide credit: Thomas J. Cortina

Python

- Created by Guido van Rossum in the late 1980s
- Allows programming in multiple paradigms: object-oriented, structured, functional
- Uses dynamic typing and garbage collection

Slide credit: Thomas J. Cortina

Summary

- **Stored program computers**
  - Components of a computer
  - Von Neumann architecture
- **Programs and Data**
  - Binary data representations
  - Bit operations
- **Programming languages (PLs)**
  - Syntax and semantics
  - Dimensions of a PL
  - Programming paradigms
  - A partial history of PLs

Next week

- **Introduction to Programming**
  - Basic Concepts
  - Developing Algorithms
  - Creating Flowcharts
- **The C Programming Language**
  - Your first C Program
  - Programming Process
  - Lexical Elements of a C Program
    - Keywords, Identifiers, Constants, Data Types, Operators
  - Standard Input and Output
  - Type Conversion and Casting

The Strange Birth and Long Life of Unix

Photo: Alcatel-Lucent

- http://spectrum.ieee.org/computing/software/the-strange-birth-and-long-life-of-unix