CS240
Lecturer: Dr. Simina Fluture

Week 1

Computer System Architecture, M. Morris Mano

Computer Architecture: Computer architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction set, and techniques for addressing memory.

Computer Organization: Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.

Computer Organization and Architecture, William Stallings

Computer Architecture: Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.

Computer Organization: Computer organization refers to the operational units and their interconnections that realize the architectural specifications.

Architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.

Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

Computer Architecture, Patterson & Hennessy

The architecture of the computer is the interface between the hardware and the lowest-level software. It includes anything programmers need to know to make a binary machine language program work correctly.

CS240
Lecturer: Dr. Simina Fluture

Microprocessor without Interlocked Pipeline Stages

<u>About MIPS and the Author of the Textbook</u>

In 1981, a team led **by John L. Hennessy** at Stanford University started work on what would become the first MIPS processor. **The basic concept was to dramatically increase performance through the use of deep instruction pipelines, a technique that was well known, but difficult to implement.** Generally in pipeline architecture, successive instructions in a program sequence will overlap in execution. Instead of waiting for the instruction to complete, each unit inside the CPU will fetch and start executing an instruction before the preceding instruction is complete.

A difference between the MIPS design and the competing Stanford RISC involved the handling of subroutine calls. RISC used a technique called register windows to improve performance of these very common tasks, but in using hardware to do this they locked in the number of calls that could be supported. Hennessy felt that a careful compiler could find free registers without resorting to a hardware implementation, and that simply increasing the number of registers would not only make this simple, but increase the performance of all tasks.

In other ways the MIPS design was very much in keeping with the overall RISC design philosophy. To improve overall performance, RISC designs reduce the number of instructions in order to use fewer bits to encode them - in the MIPS design the instructions normally require only 5 bits of the 32-bit word.

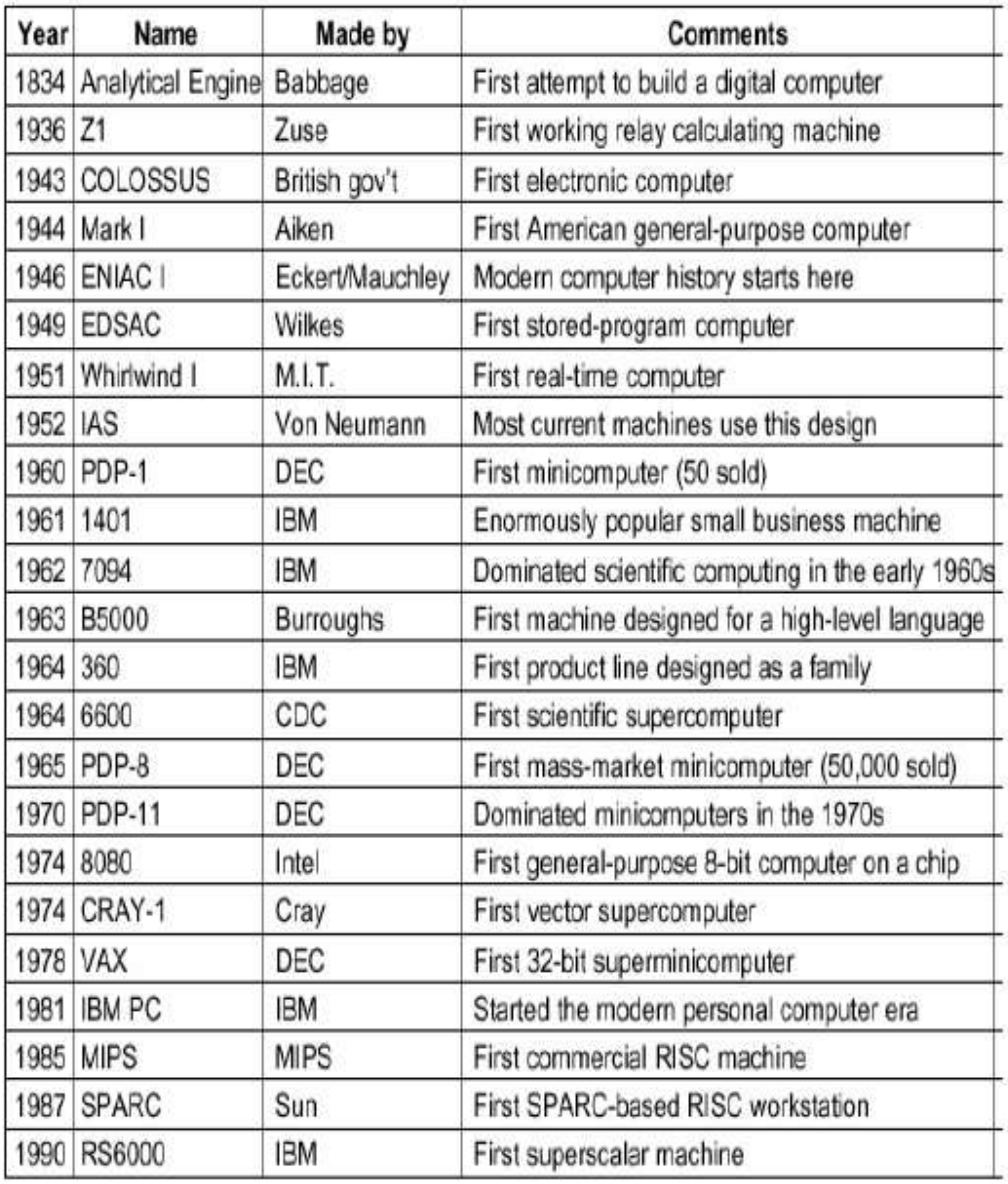
In 1984 Hennessy was convinced of the future commercial potential of the design, and left Stanford to form MIPS Computer Systems. They released their first design, the **R2000**, in 1985, improving the design as the **R3000** in 1988. These 32-bit CPUs formed the basis of their company through the 1980s, used primarily in SGI's series of workstations.

In 1991 MIPS released the first 64-bit microprocessor, the **R4000**. However, MIPS had financial difficulties while bringing it to market. The design was so important to SGI, at the time one of MIPS' few major customers that SGI bought the company outright in 1992 in order to guarantee the design would not be lost. As a subsidiary of SGI, the company became known as MIPS Technologies.

CS240
Lecturer: Dr. Simina Fluture

Topics: **Computer History**
Von Neumann Architecture
The Main Components of a Computer

Computer History (up to 2000):



Year	Name	Made by	Comments
1834	Analytical Engine	Babbage	First attempt to build a digital computer
1936	Z1	Zuse	First working relay calculating machine
1943	COLOSSUS	British gov't	First electronic computer
1944	Mark I	Aiken	First American general-purpose computer
1946	ENIAC I	Eckert/Mauchley	Modern computer history starts here
1949	EDSAC	Wilkes	First stored-program computer
1951	Whirlwind I	M.I.T.	First real-time computer
1952	IAS	Von Neumann	Most current machines use this design
1960	PDP-1	DEC	First minicomputer (50 sold)
1961	1401	IBM	Enormously popular small business machine
1962	7094	IBM	Dominated scientific computing in the early 1960s
1963	B5000	Burroughs	First machine designed for a high-level language
1964	360	IBM	First product line designed as a family
1964	6600	CDC	First scientific supercomputer
1965	PDP-8	DEC	First mass-market minicomputer (50,000 sold)
1970	PDP-11	DEC	Dominated minicomputers in the 1970s
1974	8080	Intel	First general-purpose 8-bit computer on a chip
1974	CRAY-1	Cray	First vector supercomputer
1978	VAX	DEC	First 32-bit superminicomputer
1981	IBM PC	IBM	Started the modern personal computer era
1985	MIPS	MIPS	First commercial RISC machine
1987	SPARC	Sun	First SPARC-based RISC workstation
1990	RS6000	IBM	First superscalar machine

CS240
Lecturer: Dr. Simina Fluture

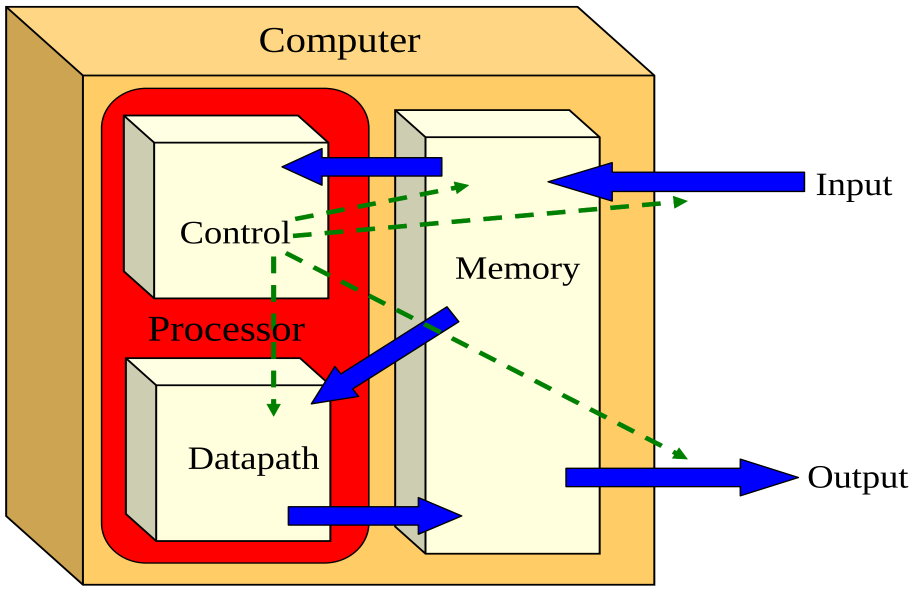
Von Neumann Architecture

Von –Neumann Architecture: was introduced during the first generation of computers; it is a design model for digital computers that has been followed since then. The Von Neumann bottleneck consists in the separation between the CPU and memory; this will limit the data transfer rate.

Main Computer Components

The hardware consists of five classic components:

- Input
- Output
- Memory
- Datapath (arithmetic operations) and control – processor



The CPU fetches an instruction from memory (one at a time) and executes the instruction.

CPU: controls the operation of the computer and performs data processing.

- **Control Unit:** controls the operation of the CPU and computer
- **Arithmetic and logic unit:** performs the computer's data processing

Memory: Main Memory

- mostly RAM (Random Access Memory) – read and write
- ROM (Read Only Memory)

Other types of main memory that the CPU can directly access: Registers, Cache

I/O devices: moves data between the computer and external devices

CS240
Lecturer: Dr. Simina Fluture

Topics: System Bus Architecture
- **Language Levels**

Instruction Set
Architecture Moore's Law

Fetch (Decode) & Execute Cycle

Fetch the next instruction from memory into the instruction register
Change the program counter to point to the following instruction
Decode the instruction
Execute the instruction Repeat

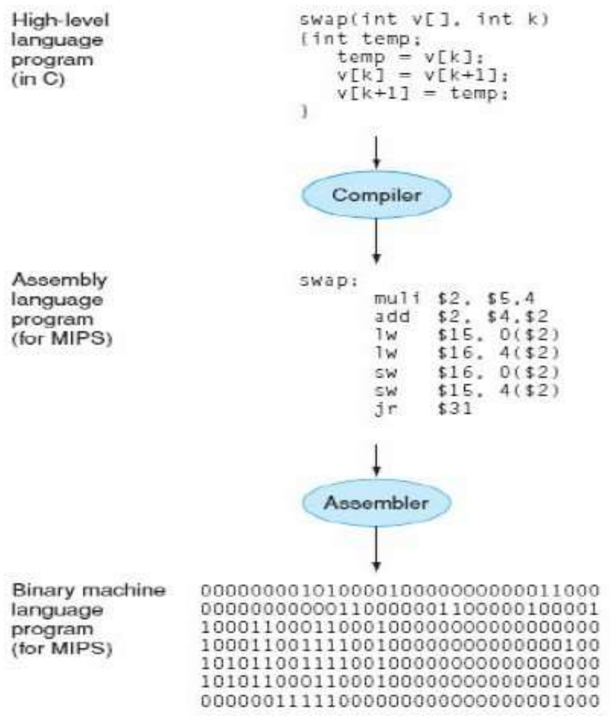
The System Bus Model

A refinement of the von Neumann model, the system bus model has a CPU (ALU and control), memory, and an input/output unit.
System Bus: parallel electrical conductors
System bus has 50 to hundreds of separate lines.

Data lines (data bus): to transfer data
Address lines: for specifying the destination of data; CPU might want to read from or write to memory.
Control lines: control the access of data (read or write)

High-Level Language to Binary Language

Programs can be translated to machine language, which can be executed directly in the computer. The language that a compiler translates is called the **source code**.
Assembly language is an interface to higher-level software.


High-level language program (in C)
Compiler
Assembly language program (for MIPS)
Assembler
Binary machine language program (for MIPS)

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

```
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
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instruction Set Architecture (ISA): interface between the hardware and the low-level software; it enables many implementations of varying cost and performance to run identical software.

It represents anything a programmer needs to know to make an assembly-language program work correctly.

- Instruction formats:
- What the instructions do
- number and types of registers
- addressing modes, exceptional conditions

Modern Instruction Set Architectures:
IA-32, PowerPC, MIPS, SPARC, ARM, and others

Moore's Law

Computing power doubles every 18 months for the same price.
Project planning needs to take this observation seriously: an architectural innovation that is being developed for a projected benefit that quadruples performance in three years may no longer be relevant: the architectures that exist by then may already offer quadrupled performance and may look entirely different from what the innovation needs to be effective.

CS240
Lecturer: Dr. Simina Fluture

DATA REPRESENTATION (INTRODUCTION)

A digital computer manipulates information expressed in the binary number system.

Memorize the first powers of 2:

2^0 ……………………………………… 2^{10}

The smallest memory unit is a bit: 0, 1; Byte**:** 8 bits

1KB =
1MB =
1GB =
1TB =
1PB =

Fixed Point Numbers: each number has exactly the same number of digits and the point is always fixed, in the same place.

Decimal Numbers:
$7341_{10} = 7 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 1 \times 10^0$

The coefficients of a number system define the range of possible values that a digit may have: 0 – 9 for decimal; 0,1 for binary.

Binary Numbers:
Most Significant Bit: is the leftmost bit
Least Significant Bit: the rightmost bit
In general a number expressed in base *b* system has coefficients (k) multiplied by the power of *b*:

Coefficients range from 0 to *b-1*. To distinguish between numbers of different bases, we enclose the coefficients in parentheses and write a subscript equal to the base used.

Examples:

Letters of the alphabet are used to supplement the 10 decimal digits when the base of the number is greater than 10 – hexadecimal (base 16).

Assembly language frequently uses the **hexadecimal number system**. We will see that converting between binary and hexadecimal doesn't require computation.

To specify that the number is in hexadecimal we use the notation: 0x
For example: 10 = 0xA

2	8	10	16

CS240
Lecturer: Dr. Simina Fluture

Number (base) Conversion:

Decimal to other Bases (Binary, octal, hexadecimal)

Conversion of a decimal number into a base *b* is done by successive divisions by *b* and accumulations of the remainders. Process continues until the integer quotient becomes 0.

Or by expressing the decimal number as a sum of powers of ***b***

To binary:

To octal:

To hexadecimal:

Decimal Fraction to Binary

Multiplication is used and the integers are accumulated instead of remainders.

$(0.6875)_{10} = ?_{2}$

The conversion of decimal numbers with both integer and fractional parts is done by converting the integer and fraction separately and then combining the two answers.

$(41.6875)_{10} = ?_{2}$

Binary to other bases (decimal, octal, hexadecimal)

To decimal:
We already covered the conversion of a number in base b to decimal.

To octal:
Partition the binary number in groups of **three** digits each starting from the binary point and preceding to the right and left.

$10110001101011.111100000110 = ?_{8}$
To hexadecimal:
Partition the binary number in groups of **four** digits each starting from the binary point and preceding to the right and left.

$10110001101011.11110010 = ?_{16}$

Detecting whether a binary number is ODD or EVEN

Multiplication by constants that are a power of 2

CS240
Lecturer: Dr. Simina Fluture

SIGNED FIXED POINT NUMBERS

There are different ways of representing signed fixed point numbers:

- Signed Magnitude
- One's Complement
- Two's Complement
- Biased Representation

Signed Magnitude:
Leftmost bit is the sign bit: 0 for positive, 1 for negative
If the number is stored in a byte, 1 bit is for the sign and 7 bits for the number.

For arithmetic operation in the computer, negative numbers are referred to as **SIGNED-COMPLEMENT SYSTEM.** We do not use a signed magnitude representation. A negative number is represented by its complement.

One's Complement:
The **formal way** to determine the 1's complement in binary of a negative integer uses the following rule:

Rule: The 1's complement in binary of a negative integer is obtained by subtracting its magnitude from 2^n **-1** where n is the number of bits used to store the integer in binary.

There is in fact, then, an easy way to determine the 1's complement of a negative integer. Starting from the signed representation of the positive value, invert all the bits of the positive representation including the sign bit.

Note: There are two different representations for +0 and -0.

Two's Complement:
To negate: compute the one's complement and add 1
Note: the carry from the leftmost bit is thrown away.

2's complement has only one representation of 0.
All negative numbers have a 1 in the most significant bit.

There is in fact, then, an easier way to determine the 2's complement of an integer.
Starting from the signed representation, leave all the rightmost zero's and the first rightmost one unchanged, invert all the bits of the positive representation including the sign bit.

4bits combination	**its value (in base 10)**
0000	0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	
1010	-6
1011	
1100	-4
1101	
1110	
1111	

Positive numbers start at 0 but the negative numbers start at -1. The magnitude of the most negative number is one greater than the magnitude of the most positive number.

Sign Extension (for 2's complement representation) – used in arithmetic operations:
Copy the most significant bit into all the additional significant digits.

Zero Extension – used in logical operations:
Copy the zero bit into all the additional significant digits.

Biased Representation
A bias value is subtracted from the number. Floating point representation uses a biased exponent. The number is treated as unsigned but it is "shifted' in value by subtracting the bias from it.

Fixed Point Addition and Subtraction
We consider the 2's complement number system
The sign of the result will be generated correctly as long as overflow doesn't occur.

Subtraction can be seen as a special addition. The carry at the highest bit position is disregarded.
a – b = a + (-b)

Examples (of 8-bit 2's complement addition)

(+10) + (23) = 33	**(+6) + (+13)**	**(+29) + (+14)**
(+5) + (-2)	**(-6)+(13)**	**(-1) + (-4)**

CS240
Lecturer: Dr. Simina Fluture

Overflow
Overflow occurs when adding two positive numbers produces a negative result or when adding two negative numbers produces a positive result.

Adding operands of unlike signs never produces an overflow. Notice that discarding the carry out of the most significant bit during two's complement addition is a normal occurrence, and does not by itself indicate overflow.
Also check Figure 3.2 – Overflow Conditions for addition and subtraction (page 226)

As an example of overflow, consider adding $(80 + 80 = 160)_{10}$, which produces a result of -9610 in an 8-bit two's complement format:

```
   01010000 = 80
+ 01010000 = 80
----------
   10100000 = -96 (not 160 because the sign bit is 1.)
```

Or $(80 + 50)_{10}$ gives a wrong result of -126_{10}

Multiplication

Unsigned Multiplication
When 2 unsigned n-bit numbers are multiplied, the result can be as large as 2n bits.

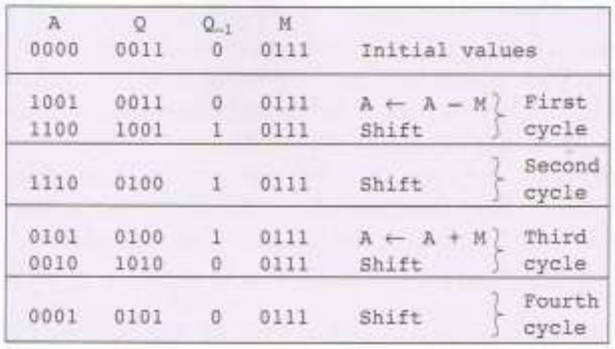
```
         1101  (13)      Multiplicand M
       X 1011   (11)     Multiplier Q
   ----------------
         1101
        1101
    1101
------------------
   10001111  (143)      Product P
```

Multiplication Hardware uses shifts (left) and add. MIPS multiply instructions ignore overflow, so it is up to the software to check for them.

CS240
Lecturer: Dr. Simina Fluture

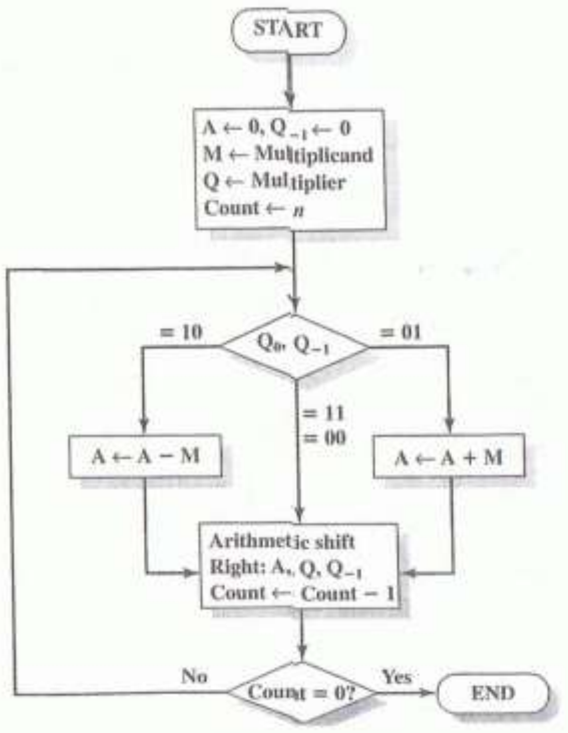
Signed Multiplication

Fixed Point Number Multiplication – Booth Algorithm



A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth cycle

Figure 9.13 Example of Booth's Algorithm (7 × 3)



00 or 11:	no arithmetic operation
01:	add the multiplicand to A (left half of the product)
10:	subtract the multiplicand from A (left part of the product)

CS240
Lecturer: Dr. Simina Fluture

Topics: **Floating Point Number Representation**
Combinational circuits - Introduction

Floating Point Representation

Fixed-point notation has its limitations. Very large numbers or very small numbers cannot be represented.

Alternative notation is the scientific notation and has a single digit to the left of the decimal point.

For example:

fixed-point notation	0.000000001
scientific notation	1.0×10^{-9}

A number in scientific notation that has no leading 0s is called a **normalized number**.

1.0×10^{-9} is in scientific normalized form

0.1×10^{-8} or 10.0×10^{-10} are not

A general form for floating point numbers is: $\mathbf{(-1)^S \times F \times B^E}$
S: sign bit
F: Fraction, Significand Field, Mantissa
E: Exponent
B: Base

B (base) doesn't need to be stored, for binary numbers it is implicitly 2.

Overflow: if the exponent is too large to be represented in the exponent field
Underflow: if the fraction is so small that it cannot be represented

To minimize the possibility of an overflow or underflow double-precision floating point is used instead of single precision floating point.

Single precision: floating point is represented in a 32-bit word.
Double Precision: floating point is represented in 64 bits (2 x 32-bit word).

CS240
Lecturer: Dr. Simina Fluture

IEEE Standard 754

- base 2
- normalized form, the leading 1 bit of the normalized binary number is not represented
  significand is 24 bits long (1 + 23) or 53 bits long (1 + 52)

Floating Point number will be of the form: **$(-1)^S$ x (1+Fraction)x 2^E**

- some bit patterns represent special values
  - 0 has the exponent value of 0
  - exponent of all ones with a nonzero fraction is *not a number* **NaN** value
- it is useful to use *biased notation*

Usually bias = $2^{k-1}-1$ *k* is the number of bits in the binary exponent.

Floating point number: **$(-1)^S$ x (1+Fraction)x $2^{\text{BiasedExponent - Bias}}$**

Range of numbers that can be represented in a 32-bit word:

+/-1.0000 0000 0000 0000 0000 000_2x 2^{-126} to a large as

+/-1.1111 1111 1111 1111 1111 111_2x 2^{+127}

<u>Figure:</u> Expressible Numbers in 32-bit formats
Numbers represented in floating point notation are not spaced evenly along the number line.

Converting Binary to Decimal Floating Point

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1 1 0 0 0 0 0 0 1 0 1 0 0 …….

Converting Decimal to Binary Floating Point

CS240
Lecturer: Dr. Simina Fluture

FP addition and subtraction

Check for zeros
Align the significands
Add or subtract the significands
Normalize the result

FP multiplication

Check for zeros
Add exponents (if stored in biased form, subtract the bias from the sum)
If overflow or underflow report and stop
Multiply the significands considering their signs
Normalize
Round

FP Division

Test for zeros
Subtract exponents (if stored in biased form, add the bias)
Check for exponent underflow or overflow
Divide significands
Normalize
Round

CS240
Lecturer: Dr. Simina Fluture

COMBINATIONAL CIRCUITS

Combinational logic: a digital logic circuit in which logical decisions are made based only on combinations of the inputs.

Sequential logic: a circuit in which decisions are made based on combinations of the current inputs as well as the past history of inputs. e.g. a memory unit.

Finite state machine: a circuit which has an internal state, and whose outputs are functions of both current inputs and its internal state. e.g. a vending machine controller.

COMBINATIONAL CIRCUITS
A combinational circuit can be defined in three ways:

- **Truth Table** (n inputs, 2^n outputs)
- **Graphical Symbols – Logic Gates**
- **Boolean functions**

A Truth Table
• Developed in 1854 by George Boole.
• Outputs are computed for all possible input combinations (how many input combinations are there?)

The output of a truth table represents a **Boolean function**.
Handout: truth tables showing all possible functions of two binary variables

Logic Gates – Handout posted

Boolean Algebra
A two-valued Boolean algebra is defined on a set of two elements, B={0,1}
All the variables have the values 0 or 1. Most important operators are NOT (‘), AND (*), OR (+).

Boolean function is an expression formed with binary variables, the two binary operators OR and AND and unary operator NOT.

Principle of duality: The dual of a Boolean function is obtained by replacing AND with OR and OR with AND, 1s with 0s, and 0s with 1s.

Identity law	**A+0 = A**	**A*1 = A**
Zero and One laws	**A + 1 = 1**	**A*0 = 0**
Inverse Laws	**A + A’ = 1**	**A * A’ = 0**
Commutative Laws	**A+B = B+A**	**A*B = B*A**
Associative Laws	**A + (B+C) = (A+B) +C**	**A * (B*C) + (A*B)*C**
Distributive laws	**A*(B+C) = (A*B) + (A*C)**	**A+(B*C) = (A+B) * (A+C)**
Absorption	**A + AB = A**	**A * (A + B) = A**

CS240
Lecturer: Dr. Simina Fluture

Complement of a Function

1. Take the dual of the function
2. Complement each literal

Use De Morgan Laws: (a+b)’ = a’*b’   (a*b)’ = a’ + b’

Minterms and Maxterms

Minterm: AND term of *n* variables
Maxterm: OR term of *n* variables

n = total number of variables; $\rightarrow$ 2^n minterms or 2^n maxterms