Lecture Notes for Lecture 1 of CS 5600 (Computer Systems) for the Fall 2019 session at the Northeastern University Silicon Valley Campus.

Computer Systems Organization

Philip Gust,
Clinical Instructor
Department of Computer Science

http://www.ccis.northeastern.edu/home/pgust/classes/cs5600/2019/Fall/index.html

Computer Systems Organization

Introduction

- In this first lecture we will begin to explore the physical characteristics of a computer systems.
- We will examine the hardware components, and how they are organized, how information is represented, how the computer operates on information, and input/output.
- Finally, we will learn about how the computer is programmed, including the instructions it can carry out, and how instructions are executed by the computer.

Computer Systems Organization

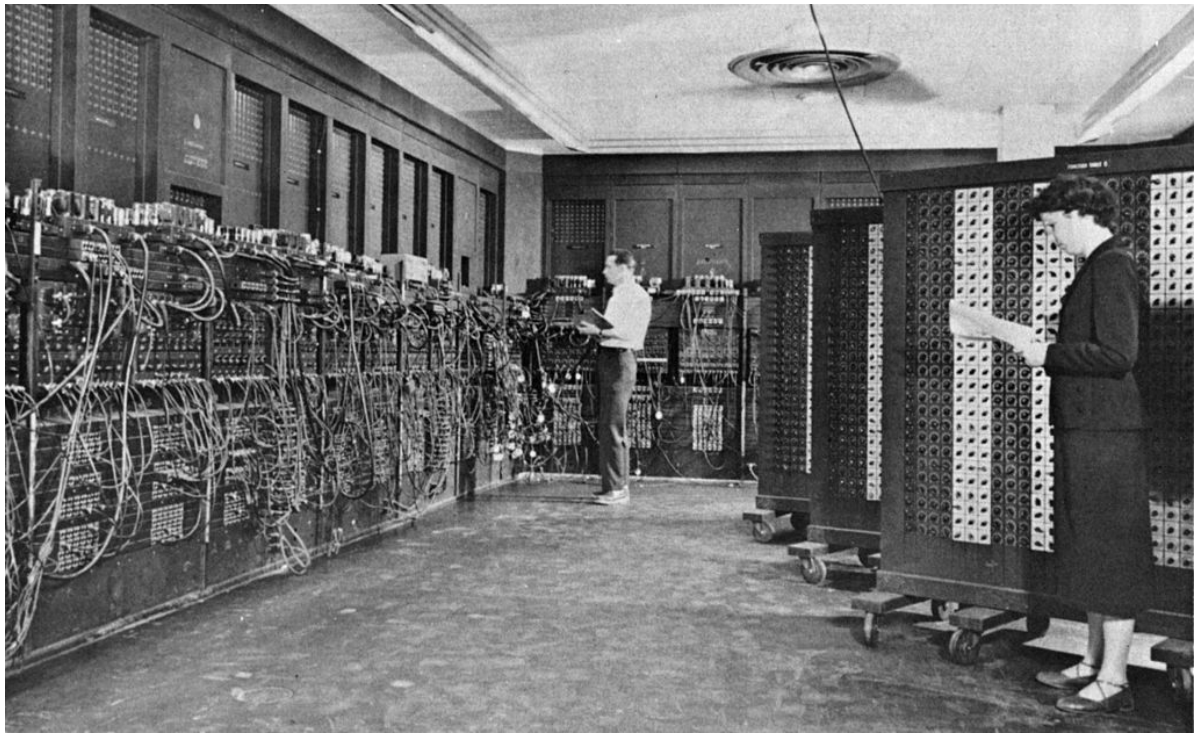
Introduction

- Modern computers are called *stored-program* computers because the program can be stored within the computer along with the information they process.
- Programs are not fixed: they can be changed over time by modifying the instructions stored within the computer as a result of computations.
- Among the earliest general-purpose digital computers, ENIAC, did not have this property. It was built in the U.S. in the mid-1940s.
- Programming was done by making physical connections with switches and wires and plugs to set a program. Modifying these programs was a very time-consuming process.

Computer Systems Organization

Introduction

- ENIAC, one of the earliest general-purpose digital computer, was programmed with switches, and cables and plugs.



Computer Systems Organization

Introduction

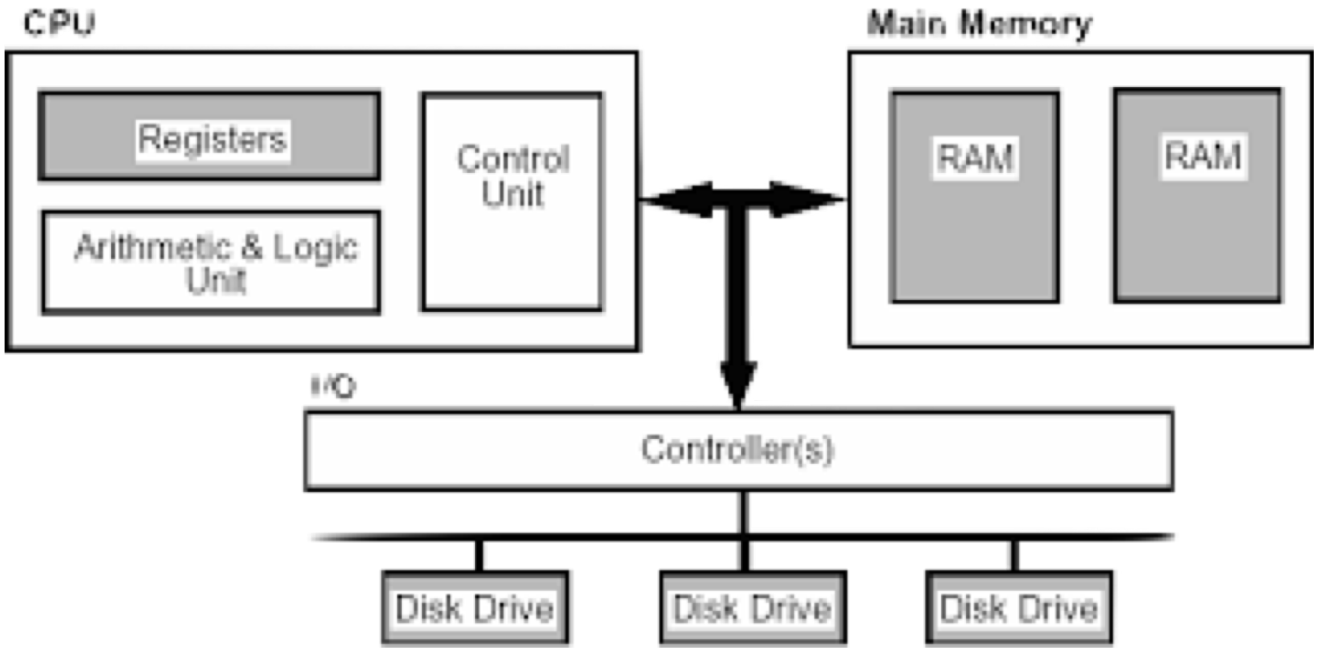
- Around the same time, mathematician John von Neumann wrote a paper describing a new computer, EDVAC, that had all the elements of a modern computer system:
  - A central processing unit that contains an arithmetic logic unit and processor registers
  - A control unit that contains an instruction register and program counter
  - Memory that stores data and instructions
  - External mass storage
  - Input and output mechanisms

https://web.archive.org/web/20110503181603/http://systemcomputing.org/turing%20award/Maurice_1967/TheFirstDraft.pdf

Computer Systems Organization

Introduction

- What he proposed became know as the *von Neumann architecture*, a stored program computer that fetches data and instructions from memory.



Computer Systems Organization

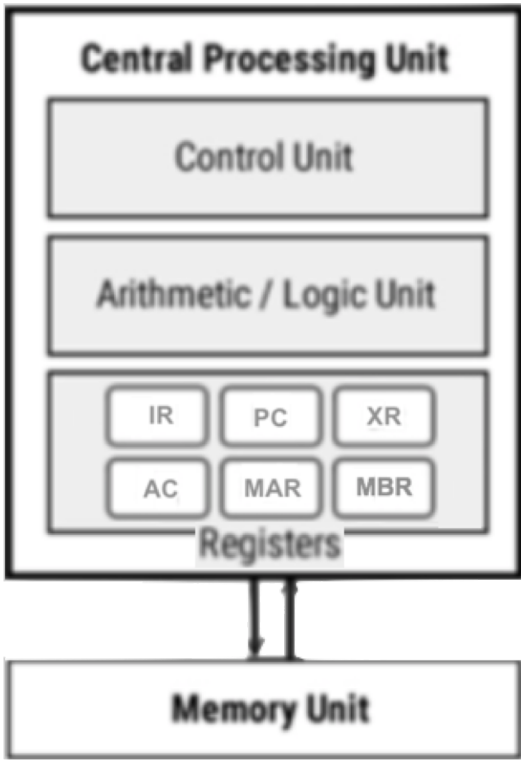
Introduction

- All modern digital computer systems are based on the von Neumann architecture, and have the same set of elements that von Neumann identified in his 1945 paper.
- We will look at the components in more detail to understand how this architecture works in principal.

Computer Systems Organization

Central Processing Unit (CPU)

- The central processing unit (CPU) is responsible for executing instructions that operate on data and instructions stored in memory.
- It consists of a control unit (CU) and high-speed memory locations known as *registers*. The CPU has a limited number of these registers, and some of them have special purposes.



- The CPU also has an arithmetic/logic unit (ALU) that performs arithmetic and decision-making operations under control of program instructions.

Computer Systems Organization

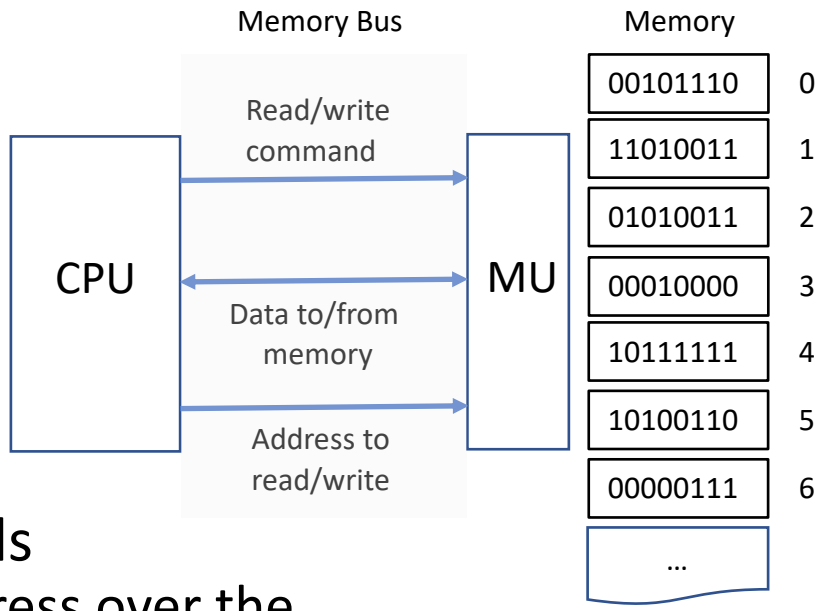
Memory Unit (CPU)

- The memory unit (MU) organizes groups of binary digits (*bits*) into units that can be addressed by location or address.
- Depending on the memory, unit can be as few as 4 bits or as many as 32 or 64 bits. The most typical size is 8-bits, which is known as a *byte*.
- The bits in the byte can represent a numeric value or a purely binary quantity, such as a text character or an element of an image.

Computer Systems Organization

Memory Unit (CPU)

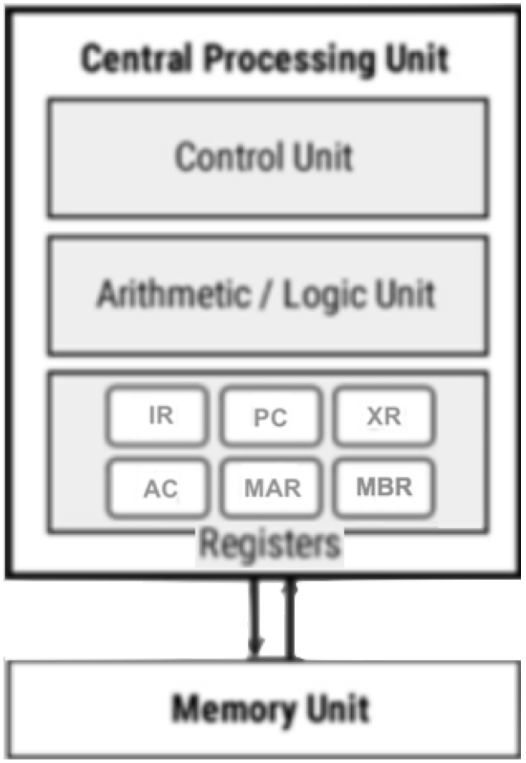
- The CPU fetches and stores values in memory using a set of lines connecting it with the memory, known as the *memory bus*.
- To read a value from memory, the CPU sends a “read” command and the address over the address line. The MU sends the value over the data line.



- To write a value, the CPU sends a “write” command and the address over the address line, and the value to write over the data line.

Computer Systems Organization

Memory Unit (CPU)

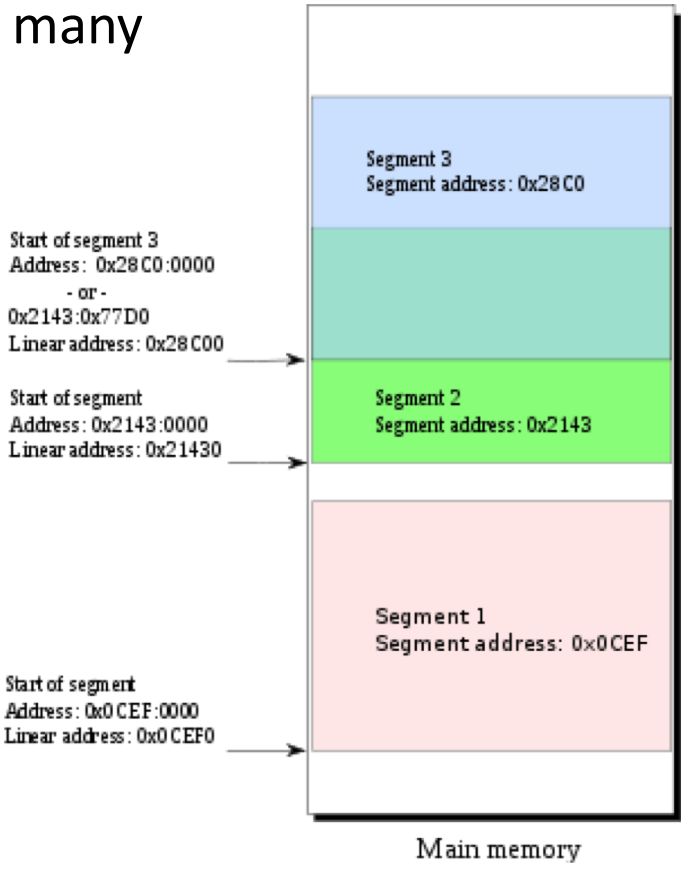


- The CU sends the address by storing it in a special register called the *memory address register* (MAR), connected to the address line.
- When writing a value, the CU places the value in another special register called the *memory buffer register* (MBR) before sending the “write command.
- When reading a value, the CU sets the MAR with the address, sends the “read” command. The MBR is connected to the data line, where the value from the data line is placed.

Computer Systems Organization

Memory Unit (CPU)

- The size of the address determines how many memory locations can be addressed.
- If the size of the MAR and memory addresses supported by the CPU matches the size of the memory address line, the CPU can directly address every memory location: a *flat address space*.
- However if the MU has a wider address line, the CPU must make special provisions to address it. A simple scheme is to *segment* the memory into addressable regions.



Computer Systems Organization

Memory Unit (CPU)

- The simplest version of this scheme is to sets a special segment register in the CPU to the base address of the segment.

0000 0110 1110 1111	segment register
0001 0010 0011 0100	MAR

- Special hardware combines the segment register with the MAR by shifting the segment register left, then adding the two and sending the longer address to the MU over its address line.

0000 0110 1110 1111 0000	segment register shifted left
0001 0010 0011 0100	MAR
0000 1000 0001 0010 0100	extended address

Computer Systems Organization

Data Representation

- If the smallest addressable value is a byte, then It is necessary for the CPU to combine a sequence of bytes to create a quantity large enough to represent common numeric types:
  - byte/char: 1 byte
  - short: 2 bytes
  - int: 4 bytes
  - long: 8 bytes
  - float: 4 bytes
  - double 8 bytes
- It is the role of the CU to fetch the appropriate number of bytes required for an operation and make them available to the ALU.
- The ALU performs arithmetic or logical operations on this group of bytes interpreted as the appropriate data type.

Computer Systems Organization

Character Data Representation

- A character can be represented by designating a value to represented it. When the byte value is sent to a character output device, that device will interpret the value as a character.
- There are several character representations in use:
  - ASCII: this 7-bit character encoding represents non-printing characters like newline or backspace with values between 0 and 31. Printable characters have values between 32 and 127.
  - Extended ASCII: this 8-bit character encoding maps values between 128 and 255 to additional characters from primarily European alphabets plus symbols like currency symbols, Greek characters, and special symbols
  - EBCDIC: an 8-bit proprietary encoding used primarily by IBM mainframes.
  - UTF-8: international character encoding corresponds to ASCII for the first 128 characters, and up to 4 bytes for other characters – part of Unicode standard.

Computer Systems Organization

Integer Data Representation

- A 32-bit integer can be represented as a sequence of, most commonly, 4 bytes.

  00001010 00001111 11110000 10100000 32-bit int

- The question arises, does the byte with the lowest address or the byte with the highest address contain the most significant byte?
- In *big-endian* format, the the byte containing the most significant bit is stored first (has the lowest address), then the following bytes are stored in decreasing significance order.

  0 1 2 3
  00001010 00001111 11110000 10100000 big-endian (msb first)

- In *little-endian* format, the least-significant byte comes first

  0 1 2 3
  10100000 11110000 00001111 00001010 little-endian (lsb first)

Computer Systems Organization

Integer Data Representation

- Both big and little forms of endianness are widely used in digital electronics. The choice of endianness for a new design is often arbitrary. Many modern CPU can actually work in either mode.
- However, later technology revisions and updates perpetuate the existing endianness and many other design attributes to maintain backward compatibility.
- Big-endian is the most common format in data networking; fields in the protocols of the Internet protocol suite, such as IPv4, IPv6, TCP, and UDP, are transmitted in big-endian order, also referred to as *network byte order*.
- Little-endian storage is popular for microprocessors, in part due to significant influence on microprocessor designs by Intel Corporation.

Computer Systems Organization

Integer Data Representation

- A negative 32-bit int can be represented by setting the most-significant bit to 0 if it is positive and 1 if it is negative.
- There are two common ways to represent a negative number in this way. Conceptually the simplest is 1's compliment: simply change all 0s to 1 and all 1s to 0.
- For example, here is the number earlier, with its 1s complement negative form

00001010 00001111 11110000 10100000	32-bit positive int
11110101 11110000 00001111 01011111	32-bit negative of positive int

Computer Systems Organization

Integer Data Representation

- While this is simple, it does have a few issues. First, there are two representations of 0:

  00000000 00000000 00000000 00000000 32-bit +0

  11111111 11111111 11111111 11111111 32-bit -0

- The other issue is that the algorithms for one's compliment arithmetic are somewhat difficult to implement.

Computer Systems Organization

Integer Data Representation

- Another way to represent a negative number is called 2's complement. To form a 2's complement, you first take the 1's complement, then add 1 as an unsigned operation.
- For example, here is the number earlier, with its 1s complement negative form

00001010 00001111 11110000 10100000	32-bit positive int
11110101 11110000 00001111 01011111	32-bit negative 1's complement
11110101 11110000 00001111 01100000	32-bit negative 2's complement

- One advantage is that there is only one representation of 0. Another is that the algorithms for 2's complement arithmetic are simpler than for 1's complement – important for hardware.

Computer Systems Organization

Integer Data Representation

- There is actually a third way to form the negative: simply change the sign bit. This is called sign-magnitude representation
- For example, here is the number earlier, with its sign-magnitude representation.

00001010 00001111 11110000 10100000	32-bit positive int
10001010 00001111 11110000 10100000	32-bit positive sign-magnitude

- This representation also has two representations of 0, and it also the algorithms for int math are also considerably more difficult than either 2's or 1's complement. This form is not often used.

Computer Systems Organization

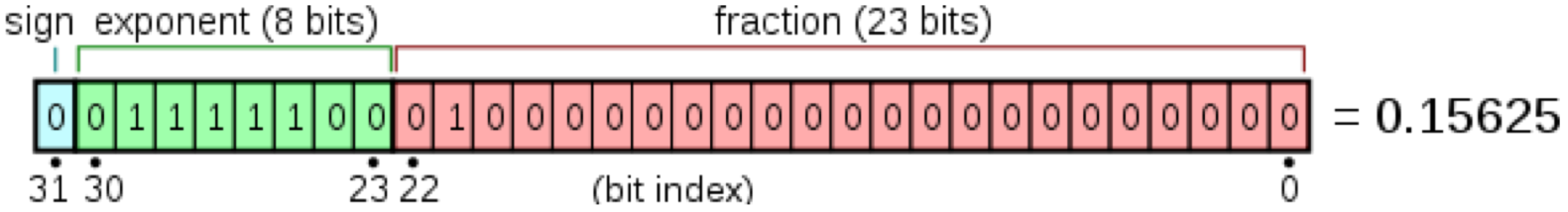
Floating Point Data Representation

- Now we will consider how to represent a real number using a floating-point approximation.
- To do this, we will represent the number in scientific notation. For example, 0.00531 can be written as 5.31×10^{-3}. The -3 is the number of places the decimal is shifted from its original position.
- We can do the same for a binary floating point number. For example, 0.00101 can be written as 1.01×2^{-3}.

Computer Systems Organization

Floating Point Data Representation

- The industry standard for representing binary floating point numbers is the IEEE 754 floating point standard. Here is an example of a single-precision 32-bit floating point number.



- The IEEE 754 standard standard specifies a binary32 as having:
  - Sign bit: 1 bit
  - Exponent width: 8 bits
  - Significand precision 24 bits (23 explicitly stored).

Computer Systems Organization

Floating Point Data Representation

- This gives from 6 to 9 significant decimal digits precision.
- If a decimal string with at most 6 significant digits is converted to IEEE 754 single-precision representation, and then converted back to a decimal string with the same number of digits, the final result should match the original string.
- If an IEEE 754 single-precision number is converted to a decimal string with at least 9 significant digits, and then converted back to single-precision representation, the final result must match the original number

Computer Systems Organization

Floating Point Data Representation

- Sign bit determines the sign of the number, which is the sign of the significand as well.
- Exponent is either an 8-bit signed integer from −128 to 127 (2's complement) or an 8-bit unsigned integer from 0 to 255, which is the accepted biased form in IEEE 754 binary32 definition.
- If the unsigned integer format is used, the exponent value used in the arithmetic is the exponent shifted by a bias – for the IEEE 754 binary32 case, an exponent value of 127 represents the actual zero (i.e. for 2e − 127 to be one, e must be 127).
- Exponents range from −126 to +127 because exponents of −127 (all 0s) and +128 (all 1s) are reserved for special numbers.

Computer Systems Organization

Floating Point Data Representation

- The true significand includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1, unless the exponent is stored with all zeros.
- Thus only 23 fraction bits of the significand appear in the memory format, but the total precision is 24 bits (equivalent to $\log_{10}(224) \approx 7.225$ decimal digits).

Computer Systems Organization

Floating Point Data Representation

- The single-precision binary floating-point exponent is encoded using an offset-binary representation, with the zero offset being 127; also known as exponent bias in the IEEE 754 standard.
  - Emin = $01_H - 7F_H = -126$
  - Emax = $FE_H - 7F_H = 127$
  - Exponent bias = $7F_H = 127$
- Thus, in order to get the true exponent as defined by the offset-binary representation, the offset of 127 has to be subtracted from the stored exponent.

Computer Systems Organization

Floating Point Data Representation

- The stored exponents 00H and FFH are interpreted specially.

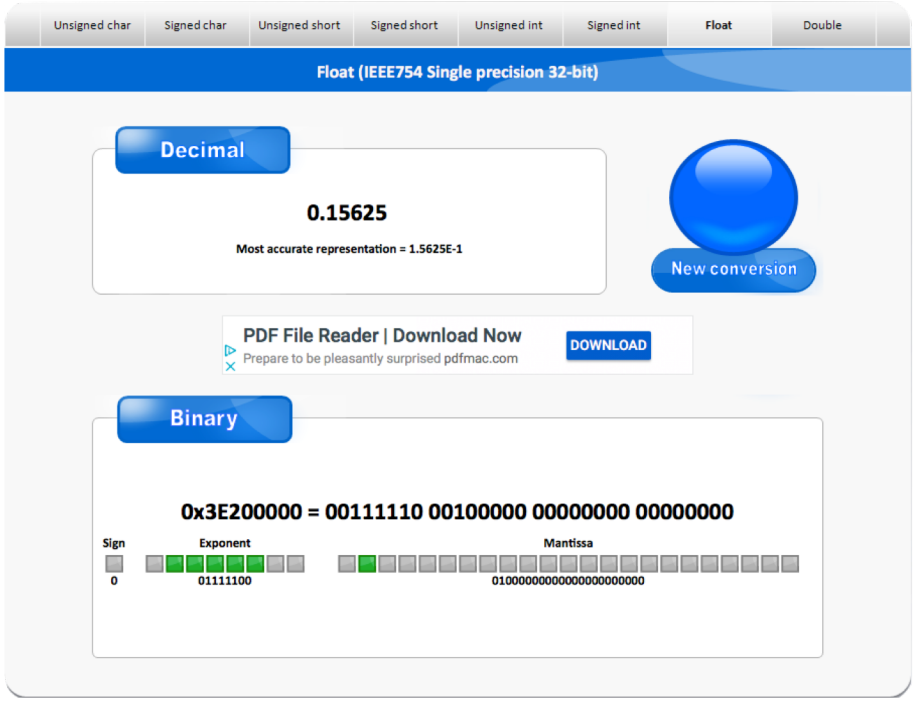
Exponent	Significand zero	Significand non-zero	Equation
00_H	zero, –0	denormal numbers	$(-1)^{signbit} \times 2^{-126} \times 0.significandbits$
01_H, ..., FE_H	normalized value		$(-1)^{signbit} \times 2^{exponentbits-127} \times 1.significandbits$
FF_H	±infinity	NaN (quiet, signalling)	

- The minimum positive normal value is $2^{-126} \approx 1.18 \times 10^{-38}$ and the minimum positive (denormal) value is $2^{-149} \approx 1.4 \times 10^{-45}$.

Computer Systems Organization

Floating Point Data Representation

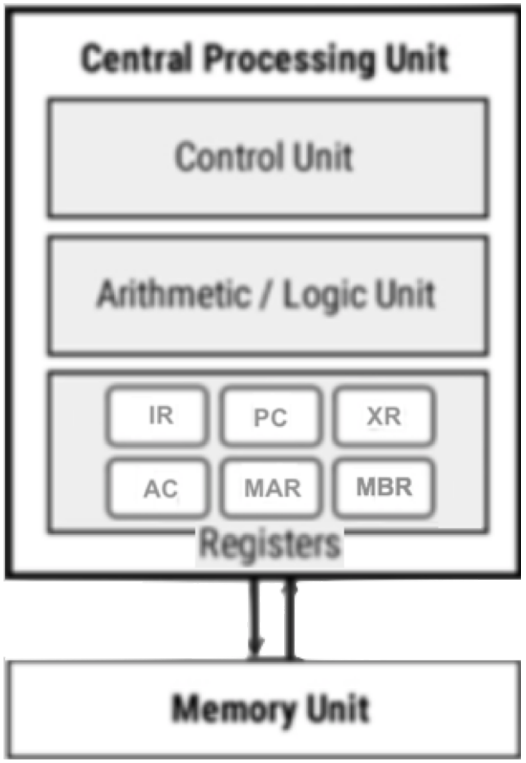
- There are a number of IEEE 753 single-precision 32-bit calculators on the web. It is much easier to use one of these than doing it by hand. (ex: http://www.binaryconvert.com/result_float.html)



Computer Systems Organization

Representing CPU Instructions

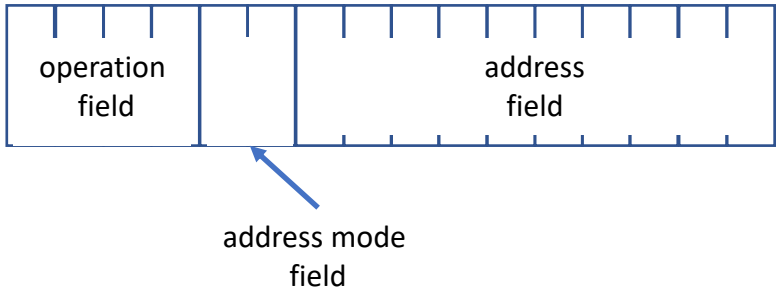
- We noted earlier that the central processing unit (CPU) is responsible for executing instructions that operate on data and instructions stored in memory.
- We will now look briefly at the instruction set for a simple computer that has just one general-purpose register called an accumulator, denoted by AC, and a special-purpose register denoted by XR. These registers are mapped to memory locations 0 and 1 respectively.



Computer Systems Organization

Representing CPU Instructions

- Each instruction is represented by a number called an *operation code* or “op-code”. The op-code is embedded as a field of a 16-bit instruction format.
- The other fields of the instruction are the address of a value or the actual value, and an *address mode* that determines how the address field is interpreted.
- This is called a *one-address* instruction format. The other value is the accumulator (AC).



Computer Systems Organization

Representing CPU Instructions

- This simple computer supports twelve instructions specified by the 4-bit operation field that fit into these categories:
  - **Memory operations** allow a program to fetch data from memory or store data in memory. Memory operations initiate memory read or write between the MU and the accumulator (AC) in the CPU.
  - **Test/branch operations** change the order in which instructions are executed. The control unit “branches” to a new location depending on the results of testing the AC.
  - **Arithmetic operations** perform arithmetic between data in the AC and whatever data are indicated in the rest of the instruction.

Computer Systems Organization

Representing CPU Instructions

- Here are the instructions including their op-codes, symbolic names for notational convenience, and descriptions

Op-code	Symbolic	Description
0000	halt	halt execution
0001	load	load from memory into the AC
0010	store	store the AC to memory
0011	call	branch to procedure
0100	br	unconditional branch
0101	breq	branch if AC = 0
0110	brge	branch if AC ≥ 0
0111	brlt	branch if AC < 0
1000	add	integer add to AC
1001	sub	integer subtraction from AC
1010	mul	integer multiplication of AC
1011	div	integer division of AC

Computer Systems Organization

Representing CPU Instructions

- The address mode field specifies how to interpret the data in the address field. This is know as the *effective address*.
  - **Direct mode**: Use the number in the address field as a memory address when executing the command
  - **Indirect mode**: The number in the address field is a memory address. Stored at that location is the address to use.
  - **Indexed mode**: use the sum of the index register (XR) and the number in the address field as a memory address when executing the command.
  - **Immediate mode**: Use the number in the address as a constant.

Computer Systems Organization

Representing CPU Instructions

- Here are the address modes the address field

Code	Symbolic	Description
00	(none)	direct mode
01	=	immediate mode
10	$	indexed mode
11	@	indirect mode

- Instructions can be written in symbolic for for notational convenience.

```
load    =1      set accumulator (AC) to value 1
store   0x177   store the value at memory location 0x177.
```

Computer Systems Organization

Executing CPU Instructions

- Once the program is loaded into sequential memory locations, and the program counter register (PC) is set to the first address to execute, the computer is started.
- The control unit performs a fixed sequence of steps, known as the instruction cycle, to execute a program.
- Control unit loads the 16-bit instruction from the address specified by the PC into the instruction register (IR), decodes its fields, and increments the address in the PC register.

Computer Systems Organization

Executing CPU Instructions

- Next, effective addressing is performed by placing the value of the IR address field into the memory address register (MAR).
- If the address mode is immediate, the value is copied to the memory buffer register (MBR).
- If the address mode is indirect, the content of the memory location is loaded into the MBR and then transferred to the MAR as the address to use.
- If the address mode is indexed, the value of the index register (XR) is added to the MAR as the address to use.

Computer Systems Organization

Executing CPU Instructions

- As an example, here are the instructions that implement the C conditional expression z = (y > 0) ? x/y : 0;
- We load the instructions starting at location 0x100, and reserve locations 0x50 for x, 0x51 for y and 0x52 for z:

```
0x100   load    0x51    load y
0x101   brlt    0x106   if y ≤ 0
0x102   breq    0x106   … branch to else code
0x103   load    0x50    then x/y
0x104   div     0x51
0x105   br      0x107
0x106   load    =0      else 0
0x107   store   0x52    store z
0x110   halt
```

Computer Systems Organization

Executing CPU Instructions

- Here are instructions to implement a C loop that sums an array of 10 integers:

```
int sum = 0;
for (int i = 9; i >=0; i--) sum += array[i];
```

- The array starts at location 0x50 and the sum is location 0x60.

```
0x100   load    =0
0x101   store   0x60    initialize sum
0x102   load    =10     number of elements to count
0x103   sub     =1      decrement number to count
0x104   brlt    0x10B   if count expired, terminate loop
0x105   store   1       store in XR as array index
0x106   load    $0x50   add next array element to sum
0x107   add     0x60
0x108   store   0x60
0x109   load    1       get index
0x10A   br      0x103   repeat for next element
0x10B   halt
```