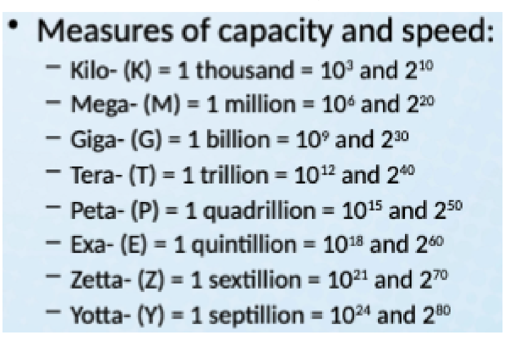
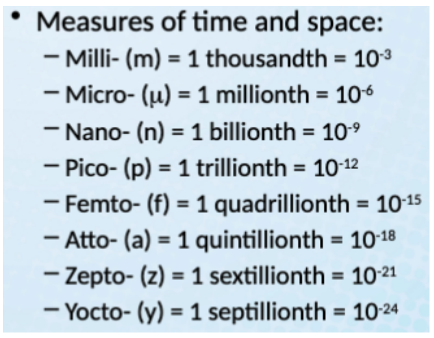
Chapter 1: Introduction

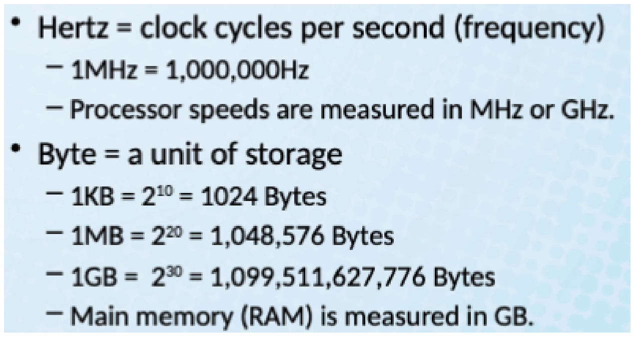
Computer organization:physical aspects of computer systems (e.g., circuit design, control signals, memory types).
Computer architecture: logical aspects of system implementations seen by programmer(instruction set/format)
Principle of Equivalence of Hardware and Software: any tasks done by hardware can also be done with software, and vise versa
Computer device: processor(interpret/execute programs), memory(store data&programs), mechanism for transferring data to/from the outside world



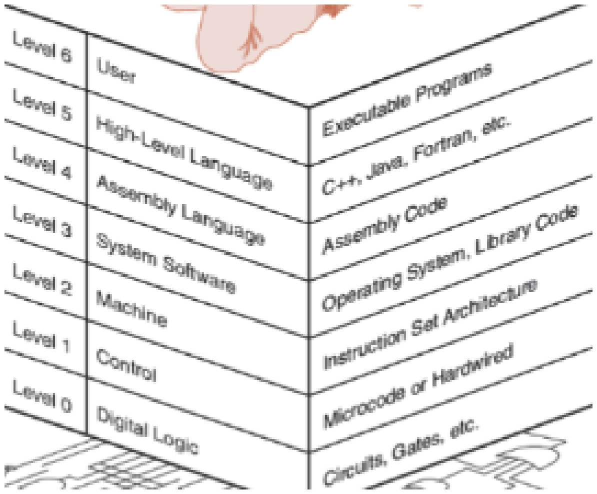
- Measures of capacity and speed:
  - Kilo- (K) = 1 thousand = 10^3 and 2^{10}
  - Mega- (M) = 1 million = 10^6 and 2^{20}
  - Giga- (G) = 1 billion = 10^9 and 2^{30}
  - Tera- (T) = 1 trillion = 10^{12} and 2^{40}
  - Peta- (P) = 1 quadrillion = 10^{15} and 2^{50}
  - Exa- (E) = 1 quintillion = 10^{18} and 2^{60}
  - Zetta- (Z) = 1 sextillion = 10^{21} and 2^{70}
  - Yotta- (Y) = 1 septillion = 10^{24} and 2^{80}

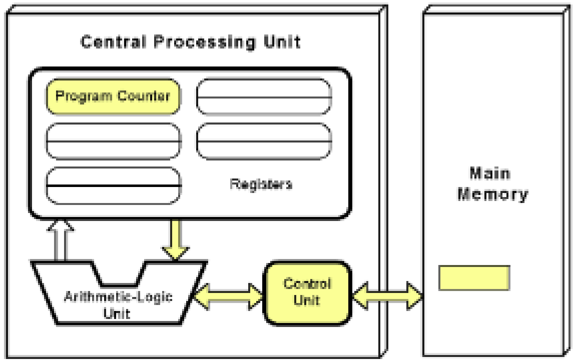


- Measures of time and space:
  - Milli- (m) = 1 thousandth = 10^{-3}
  - Micro- (μ) = 1 millionth = 10^{-6}
  - Nano- (n) = 1 billionth = 10^{-9}
  - Pico- (p) = 1 trillionth = 10^{-12}
  - Femto- (f) = 1 quadrillionth = 10^{-15}
  - Atto- (a) = 1 quintillionth = 10^{-18}
  - Zepto- (z) = 1 sextillionth = 10^{-21}
  - Yocto- (y) = 1 septillionth = 10^{-24}



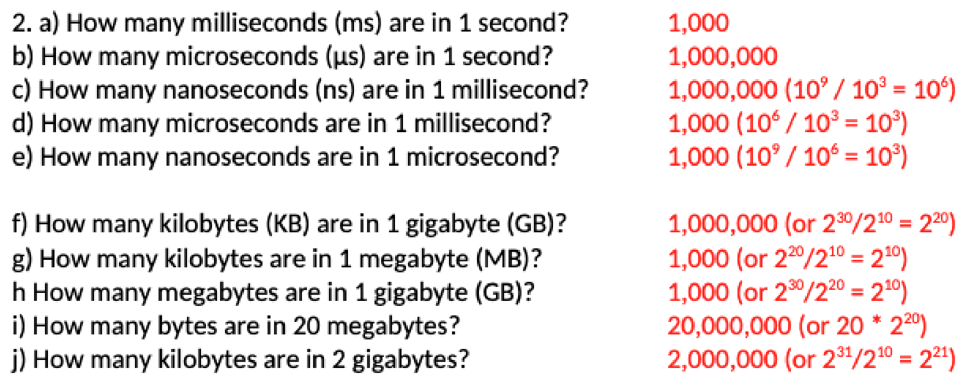
- Hertz = clock cycles per second (frequency)
  - 1MHz = 1,000,000Hz
  - Processor speeds are measured in MHz or GHz.
- Byte = a unit of storage
  - 1KB = 2^{10} = 1024 Bytes
  - 1MB = 2^{20} = 1,048,576 Bytes
  - 1GB = 2^{30} = 1,099,511,627,776 Bytes
  - Main memory (RAM) is measured in GB.





What is an ISA? Instruction Set Architecture, interface between a computer's hardware and software
Name the three basic components of every computer? CPU, memory, I/O devices
How does the fetch-decode-execute cycle work? The CPU fetches next instruction from memory, decode the instruction, and execute the instruction
What is a multicore processor? A multicore processor contains multiple CPUs each capable of executing a different program at essentially the same time.
In what ways are hardware and software different? In what ways are they the same? Between hardware and software, hardware provides more speed, software provides more flexibility. Hardware and software are related through the Principle of Equivalence of Hardware and Software. They can solve problems equally, although solutions are often easier in one versus the other.
Briefly explain two breakthroughs in the history of computing. vacuum tubes, transistors, integrated circuits,VLSI, binary arithmetic, quantum computing, and parallel computing
In the von Neumann model, explain the purpose of the:
a) **processing unit** The processing unit performs all of the arithmetic and logic functions.

b) **program counter** The program counter is responsible for keeping track of the next instruction to fetch.



2. a) How many milliseconds (ms) are in 1 second?	1,000
b) How many microseconds (µs) are in 1 second?	1,000,000
c) How many nanoseconds (ns) are in 1 millisecond?	1,000,000 (10^9 / 10^3 = 10^6)
d) How many microseconds are in 1 millisecond?	1,000 (10^6 / 10^3 = 10^3)
e) How many nanoseconds are in 1 microsecond?	1,000 (10^9 / 10^6 = 10^3)
f) How many kilobytes (KB) are in 1 gigabyte (GB)?	1,000,000 (or $2^{30}/2^{10} = 2^{20}$)
g) How many kilobytes are in 1 megabyte (MB)?	1,000 (or $2^{20}/2^{10} = 2^{10}$)
h How many megabytes are in 1 gigabyte (GB)?	1,000 (or $2^{30}/2^{20} = 2^{10}$)
i) How many bytes are in 20 megabytes?	20,000,000 (or 20 * 2^{20})
j) How many kilobytes are in 2 gigabytes?	2,000,000 (or $2^{31}/2^{10} = 2^{21}$)

Under the von Neumann architecture, a program and its data are both stored in memory. It is therefore possible for a program, thinking a memory location holds piece of data when it actually holds a program instruction, to accidentally (or on purpose) modify itself. What implications does this present to you as a programmer?
Care must be taken when programming to make sure the code doesn't modify itself in some way. For example, if a memory location holds an instruction (which is represented by a binary number), and a value is added to that instruction, the result could be a valid instruction that is later executed, resulting in an error that is very difficult to track down. The modification of an instruction could also cause a program to crash.
Explain what it means to "fetch" an instruction.
The program counter holds the memory address of the next instruction to be executed. The control unit retrieves that instruction from memory so it can be decoded and executed.
What are the limitations of Moore's Law? Why can't this law hold forever? Explain.
There are technical limitations, including heat dissipation and power leakage, not to mention the physical limitations of space and the fact that transistors can only get so small (we can't go smaller than the size of an atom).

Chapter 2: Data Representation in Computer Systems

Bit: most basic unit of information in a computer, on/off in a digital circuit, high/low in voltage
Byte: a group of 8 bits
Word: contiguous group of bytes, can be any size, 16/32/64 are most common
Nibble: group of 4 bits
What does overflow mean in unsigned numbers? The result is larger than the number. = carry out
What are the three components of a floating-point number?Sign, exponent, significant
Explain the difference between ASCII and Unicode. ASCII uses 8 bits per character and 128 assigned upper & lower on locale. Unicode uses 16 bits per character with codes by standard
How many bits does a EBCDIC, ASCII and Unicode require? ASCII:8 EBCDIC:8 Unicode:16

Chapter 3: Boolean Algebra and Digital Logic

Boolean algebra: mathematical system for the manipulation of variables that have ½ values, these values are true/false, or on/off, or 1/0, or high/low. Boolean operators include AND, OR, and NOT

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	X′
0	1
1	0

The NOT operator has highest priority, followed by AND and then OR.

F(x,y,z) = xz′ + y

x	y	z	z′	xz′	xz′ + y
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

F(x,y,z) = xy + x′z + yz

- We can use Boolean identities to simplify:

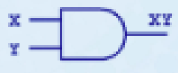
$F(x,y,z) = xy + x'z + yz$	
$= xy + x'z + yz(1)$	(Identity)
$= xy + x'z + yz(x + x')$	(Inverse)
$= xy + x'z + (yz)x + (yz)x'$	(Distributive)
$= xy + x'z + x(yz) + x'(zy)$	(Commutative)
$= xy + x'z + (xy)z + (x'z)y$	(Associative twice)
$= xy + (xy)z + x'z + (x'z)y$	(Commutative)
$= xy(1 + z) + x'z(1 + y)$	(Distributive)
$= xy(1) + x'z(1)$	(Null)
$= xy + x'z$	(Identity)

Identity Name	AND Form	OR Form
Identity Law	1x = x	0 + x = x
Null Law	0x = 0	1 + x = 1
Idempotent Law	xx = x	x + x = x
Inverse Law	xx′ = 0	x + x′ = 1

Identity Name	AND Form	OR Form
Absorption Law	x(x+y) = x	x + xy = x
DeMorgan's Law	(xy)′ = x′+ y′	(x+y)′ = x′y′
Double Complement Law	(x)″ = x	

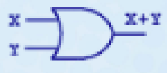
Identity Name	AND Form	OR Form
Commutative Law	xy = yx	x+y = y+x
Associative Law	(xy) z = x (yz)	(x+y) +z = x + (y+z)
Distributive Law	x+yz = (x+y) (x+z)	x (y+z) = xy+xz

- In the sum-of-products form, ANDed variables are ORed together.
  - For example: F(x,y,z) = xy + xz + yz
- In the product-of-sums form, ORed variables are ANDed together.
  - For example: F(x,y,z) = (x+y) (x+z) (y+z)



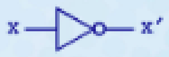
X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1



X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



NOT X

X	X′
0	1
1	0

X XOR Y

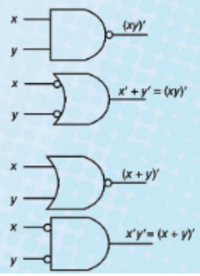
X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

X NAND Y

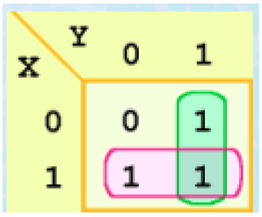
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

X NOR Y

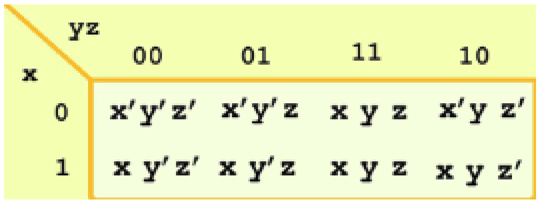
X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



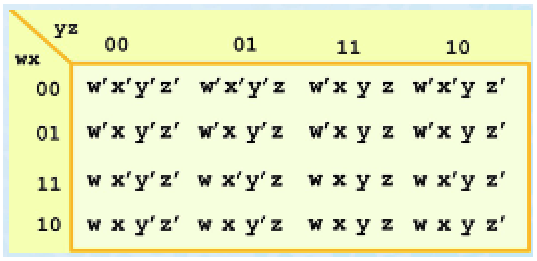
Gate: digital computer circuits that implements boolean functions. Gate is an electronic device that produces results based on 2/2+ input values
Kmap: a matrix consisting of rows and columns that represent the output values of a boolean function



x \ Y	0	1
0	0	1
1	1	1



x \ yz	00	01	11	10
0	x′y′z′	x′y′z	x y z	x′y z′
1	x y′z′	x y′z	x y z	x y z′



wx \ yz	00	01	11	10
00	w′x′y′z′	w′x′y′z	w′x y z	w′x′y z′
01	w′x y′z′	w′x y′z	w′x y z	w′x y z′
11	w x′y′z′	w x′y′z	w x y z	w x′y z′
10	w x y′z′	w x y′z	w x y z	w x y z′

Which Boolean operation is referred to as a Boolean product? AND
Which Boolean operation is referred to as a Boolean sum? OR
Describe the operation of a ripple-carry adder. Why are ripple-carry adders not used in most computers today?Multiple adder as used together with the carry-out of one adder used as an input to the next adder. They are not used because they are too slow
How are sequential circuits different from combinational circuits? Combinational circuits base their output only on the inputs. Sequential circuits base their output based on current state and input and are usually clock based.
What do we mean when we say that a sequential circuit is edge triggered rather than level triggered? Edge triggered mean that the clock transitions from high to low or low to high. Level is based on the low or high state of the clock.
Which flip-flop give a true representation of computer memory? D flip-flop. It stores one bit of information
Midterm Review Posted information:
Registers: data holding place of the computer processor. Holds instruction, storage address, data
L1 cache: small enough to provide ½ cycles of access time. Processor first looks in here
L2: larger, slower than the L1, Processor looks in here second
RAM: random-access memory, laptop's short-term memory, where data is stored that the processor needs to run your application and open your files
Disk: computer data storage device, contains sensitive information
Ranges of numbers in signed and unsigned:

- Signed int 32-bit [-2147483648 to 2147483647]
- Unsigned int 32-bit [0 to 4294967295]

RISC vs CISC differences
Fixed vs Variable Length Instructions. RISC has large number of registers, usually load/store architecture,has smaller number of instructions
Hardware interrupt: interrupt request signal from peripheral circuits(external devices) -> processor stops, device of the request line informs, interrupt program resumed, POlling, vectored, interrupt, and interrupt nesting

Software interrupt: occurs by executing dedicated instruction(internal devices) -> process stops and switches to interrupt handler code, completes required work/handle errors before returning

```
a) x(y + z)(x' + z')
   = x(x'y + yz' + x'z + zz') Distributive/Commutative
   = xx'y + xyz' + xx'z + xzz' Distributive
   = 0 + xyz' + 0 + 0          Inverse/Null
   = xyz'                      Identity

b) xy + xyz + xy'z + x'y'z
   = xy(1 + z) + (x+ x')y'z    Distributive
   = xy(1) + (1)y'z            Idempotent
   = xy + y'z                  Identity

c) xy'z + x(y + z')' + xy'z'
   = xy'z + xy'z + xy'z        DeMorgan
   = xy'z + xy'(z + z')        Distributive
   = xy'z + xy'(1)             Inverse
```

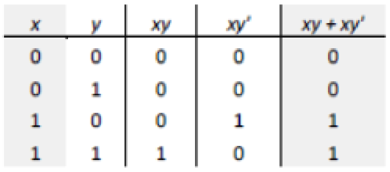
Using DeMorgan's Law, write an expression for the complement of F if

```
F(x,y,z) = (x'+y)(x+z)(y'+z)'

F(x,y,z)  = (x'+y)(x+z)(y'+z)'
F'(x,y,z) = ((x'+y)(x+z)(y'+z)')'
          = (x'+y)'+(x+z)'+(y'+z)''
          = xy'+x'z'+(y'+z) (not simplified)
```

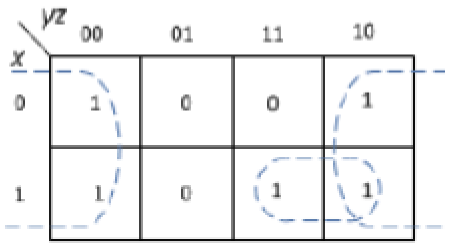
Show that x = xy + xy'
a) Using truth tables
b) Using Boolean identities

a)



x	y	xy	xy'	xy + xy'
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	0	1

$z' + xy$



x \ yz	00	01	11	10
0	1	0	0	1
1	1	0	1	1

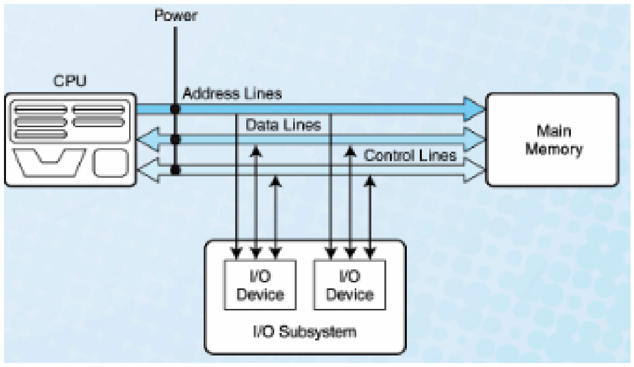
```
b) xy + xy' = x(y + y') Distributive
            = x(1) Inverse
            = x Identity
```

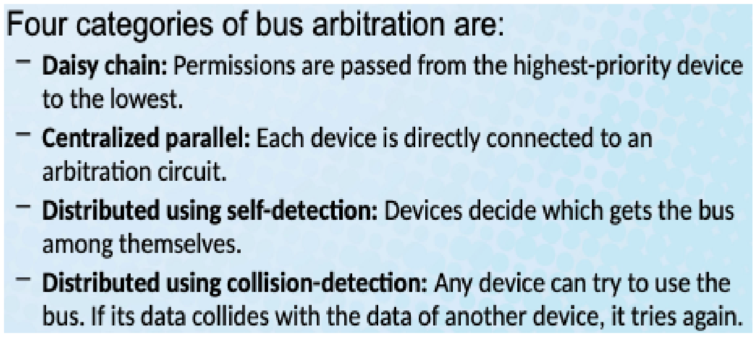
How many inputs does a decoder have if it has 64 outputs? 6 inputs.
How many control lines does a multiplexer have if it has 32 inputs? 5 control lines

---

Chapter 4: MARIE: An introduction to a simple computer

Bus: a set of wires that simultaneously convey a single bit along each line. Consists of data lines, control lines, and address lines.
Control lines: determine the direction of data flow, and when each device can access the bus
Address lines: determine the location of the source/destination of the data





Four categories of bus arbitration are:
- **Daisy chain:** Permissions are passed from the highest-priority device to the lowest.
- **Centralized parallel:** Each device is directly connected to an arbitration circuit.
- **Distributed using self-detection:** Devices decide which gets the bus among themselves.
- **Distributed using collision-detection:** Any device can try to use the bus. If its data collides with the data of another device, it tries again.

- The CPU time required to run a program is given by the general performance equation:

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- I/O can be memory-mapped—where the I/O device behaves like main memory from the CPU's point of view.
- Or I/O can be instruction-based, where the CPU has a specialized I/O instruction set.

High order: the high-order 4 bits select the chip
Low order: the low-order 4 bits select the chip

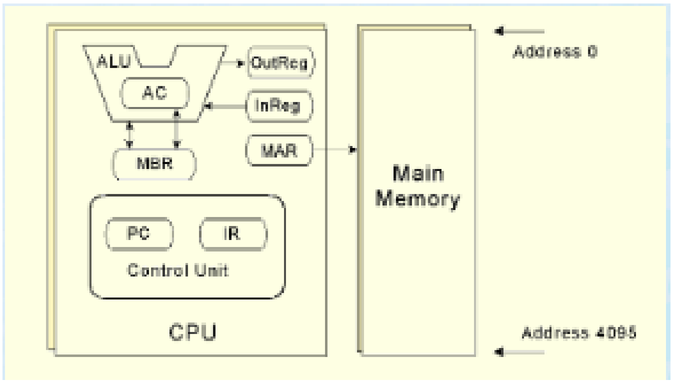
MARIE:

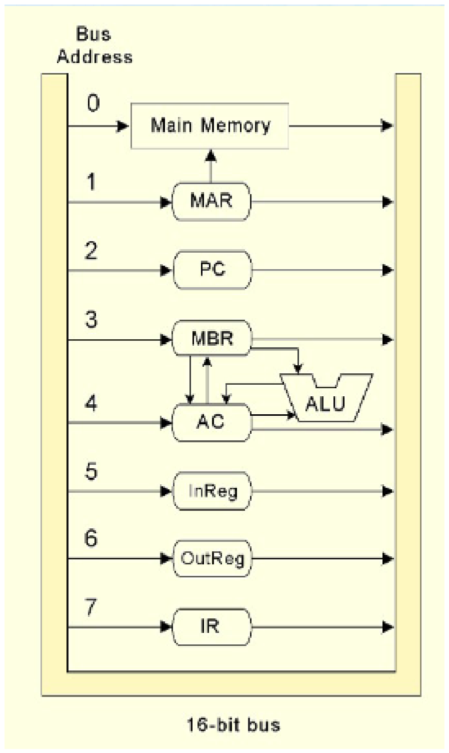
The MARIE architecture has the following characteristics:

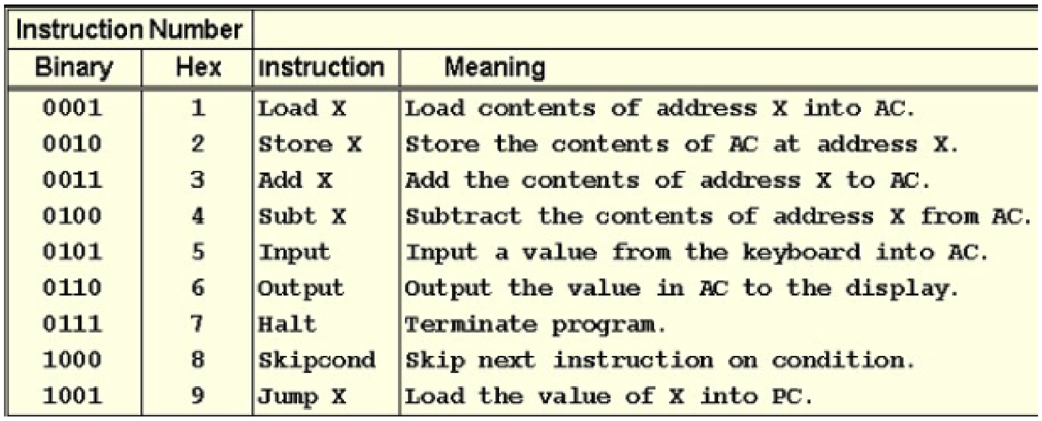
– Binary, two's complement data representation.
– Stored program, fixed word length data and instructions.
– 4K words of word-addressable main memory.
– 16-bit data words.
– 16-bit instructions, 4 for the opcode and 12 for the address.
– A 16-bit arithmetic logic unit (ALU).
– Seven registers for control and data movement.

MARIE's seven registers are:

– (1) Accumulator, AC, a 16-bit register that holds a conditional operator (e.g., "less than") or one operand of a two-operand instruction.
– (2) Memory address register, MAR, a 12-bit register that holds the memory address of an instruction or the operand of an instruction.
– (3) Memory buffer register, MBR, a 16-bit register that holds the data after its retrieval from, or before its placement in memory.
– (4) Program counter, PC, a 12-bit register that holds the address of the next program instruction to be executed.
– (5) Instruction register, IR, which holds an instruction immediately preceding its execution.
– (6) Input register, InREG, an 8-bit register that holds data read from an input device.
– (7) Output register, OutREG, an 8-bit register, that holds data that is ready for the output device.







Instruction Number			
Binary	Hex	Instruction	Meaning
0001	1	Load X	Load contents of address X into AC.
0010	2	Store X	Store the contents of AC at address X.
0011	3	Add X	Add the contents of address X to AC.
0100	4	Subt X	Subtract the contents of address X from AC.
0101	5	Input	Input a value from the keyboard into AC.
0110	6	Output	Output the value in AC to the display.
0111	7	Halt	Terminate program.
1000	8	Skipcond	Skip next instruction on condition.
1001	9	Jump X	Load the value of X into PC.

Where are registers located and what are the different types?CPU chip. hold number and control registers to maintain CPU status.

What is a bus cycle?The states needed to transfer data over the bus. The time between two clicks of the clock.

Explain the difference between memory-mapped I/O and instruction-based I/O.

In memory-mapped I/O, the device registers appear as memory locations. In instruction-based I/O CPU instructions transfer data to the device registers.

Why is address alignment important?

If data can be read from memory on aligned boundaries, performance is improved.

List and explain the two types of memory interleaving and the differences between them.

High-order interleaving means sequential addresses are on the same chip. Low-order interleaving means sequential addresses are spread over chips. Low-order can result in performance improvements.

How does interrupt driven I/O work?An interrupt causes the current state of the CPU to be saved, the interrupt is handled and then the CPU state is restored. Interrupts are used by devices to signal completion of a task.

What is a stack? Why is it important for programming? A stack is used to store local variables for a function and the return address of a function. It makes modular programming and recursive functions feasible.

What are the main functions of the CPU? The CPU is responsible for fetching program instructions, decoding each instruction that is fetched and performing the indicated sequence of operations on the correct data.

How is the ALU related to the CPU? What are its main functions?

The ALU is part of the CPU. It carries out arithmetic operations (typically only integer arithmetic) and can carry out logical operations such as AND, OR, and XOR, as well as shift operations.

How many bits are required to address a 4M × 16 bits main memory if

a) Main memory is byte-addressable? 4Mx2= 2^2 x 2^20 x2 = 2^23. So 23 bits

b) Main memory is word-addressable? 4Mx2= 2^2 x 2^20 x2 = 2^22. So 22 bits

A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (opcode) and an address part (allowing for only one address). Each instruction is stored in one word of memory.

a) How many bits are needed for the opcode? 8

b) How many bits are left for the address part of the instruction? 16

c) What is the maximum allowable size for memory? 2^{16}

d) What is the largest unsigned binary number that can be accommodated in one word of memory? $2^{24} - 1$

Explain why, in MARIE, the MAR is only 12 bits wide while the AC is 16 bits wide. MARIE can handle 16-bit data, so the AC must be 16 bits wide. However, MARIE's memory is limited to 4096 address locations, so the MAR only needs to be 12 bits wide to hold the largest address.

Write the assembly language equivalent of the following MARIE machine language instructions:

a) 0111 0000 0000 0000 Halt

b) 1011 0011 0011 0000 AddI 330

c) 0100 1111 0100 1111 Subt F4F

Write the following code segment in MARIE's assembly language:

```
if X > 1 then
Y = X + X;
X = 0;
endif;
Y = Y + 1;
```

```
         ORG      100
If,      Load     X      /Load X
         Subt     One    /Subtract 1, store result in AC
         Skipcond 800    /If AC>0 (X>1), skip the next
         Jump     Endif  /Jump to Endif if X is not >= 1
Then,    Load     X      /Reload X so it can be doubled
         Add      X      /Double X
         Store    Y      /Y= X + X
         Clear           /Move 0 into AC
         Store    X      /Set X to 0

Endif, Load     Y      /Load Y into AC
       Add      One    /Add 1 to Y
       Store    Y      /Y = Y + 1
       Halt            /Terminate program
X,     Dec      5      /X has starting value
Y,     Dec      6      /Y has starting value
One,   Dec      1      /Use as a constant
```

```
X = 1;
while X < 10 do
   X = X + 1;
endwhile;
```

```
          ORG       100
          Load      One
          Store     X     /Initialize X
Loop,     Load      X     /Load loop constant
          Subt      Ten   /Compare X to 10
          SkipCond  000   /If X is less than 10, loop
          Jump      Endloop /terminate loop
          Load      X     /Begin body of loop
          Add       One   /Add 1 to X
          Store     X     /Store new value in X
          Jump      Loop  /Continue loop
Endloop,  Halt            /Terminate program
X,        Dec 0           /Storage for X
One,      Dec 1           /The constant value 1
Ten,      Dec 10          /The loop constant
```

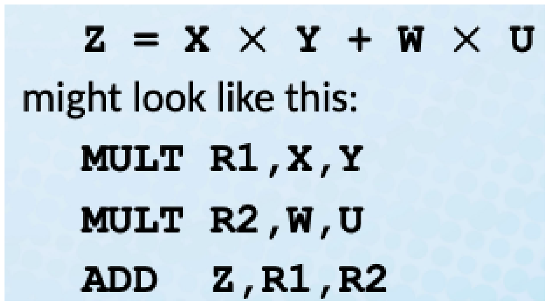
Chapter 5: A closer look at Instruction Set Architectures

Infix notation: Z = X + Y Postfix notation: Z = XY+

Infix notation: Z = (X x Y) + (W x U) Postfix notation: Z = X Y x W U x +

Infix notation: (2+3) - 6/3 Postfix notation: 2 3+ 6 3/ -

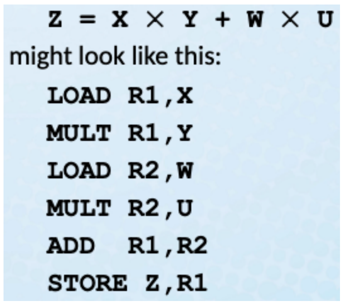
Infix notation: A + B / C - 4 Postfix notation: A B C / + 4 -



Z = X × Y + W × U

might look like this:

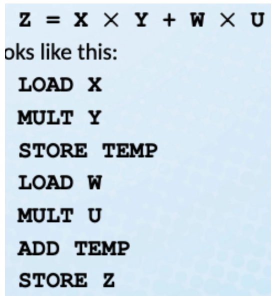
```
MULT R1,X,Y
MULT R2,W,U
ADD  Z,R1,R2
```



Z = X × Y + W × U

might look like this:

```
LOAD  R1,X
MULT  R1,Y
LOAD  R2,W
MULT  R2,U
ADD   R1,R2
STORE Z,R1
```



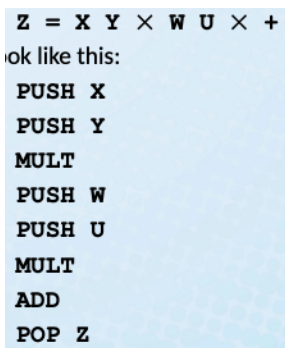
Z = X × Y + W × U

oks like this:

```
LOAD X
MULT Y
STORE TEMP
LOAD W
MULT U
ADD TEMP
STORE Z
```

three-address ISA -> infix two-address ISA -> infix one-address ISA -> infix

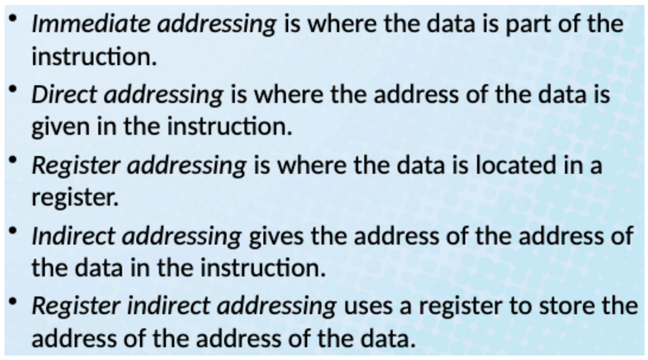
Stack ISA -> postfix



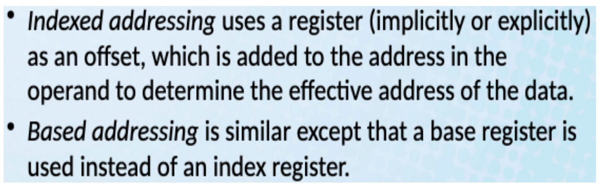
Z = X Y × W U × +

ok like this:

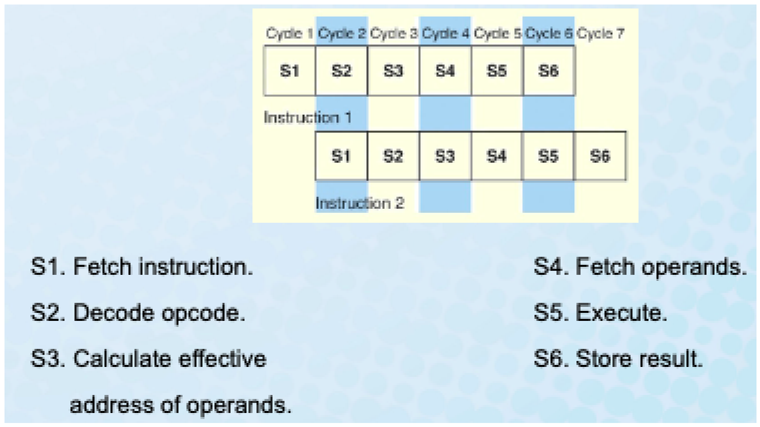
```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
ADD
POP Z
```



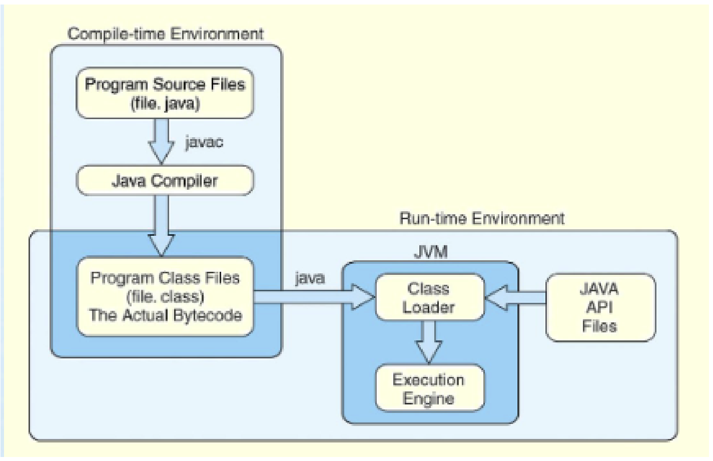
- *Immediate addressing* is where the data is part of the instruction.
- *Direct addressing* is where the address of the data is given in the instruction.
- *Register addressing* is where the data is located in a register.
- *Indirect addressing* gives the address of the address of the data in the instruction.
- *Register indirect addressing* uses a register to store the address of the address of the data.



- *Indexed addressing* uses a register (implicitly or explicitly) as an offset, which is added to the address in the operand to determine the effective address of the data.
- *Based addressing* is similar except that a base register is used instead of an index register.



S1. Fetch instruction.
S2. Decode opcode.
S3. Calculate effective address of operands.
S4. Fetch operands.
S5. Execute.
S6. Store result.



How does a microprogram operation differ from a regular assembly language instruction? A regular assembly language instruction are the instructions implemented by the CPU. In order to execute each assembly language instruction several execution steps are usually required. The microprogram instructions implement these steps by controlling the control signals needed to perform the operation.

Compare CISC machines to RISC machines?In CISC machines instructions are of variable length allowing much more complex and compact instructions. In RISC machines all instructions are the same length. CISC instructions are more complex to decode. RISC instructions are easy to decode.

MARIE saves the return address for a subroutine in memory, at a location designated by the jump-and-store instruction. In some architectures, this address is stored in a register, and in many it is stored on a stack. Which of these methods would best handle recursion?

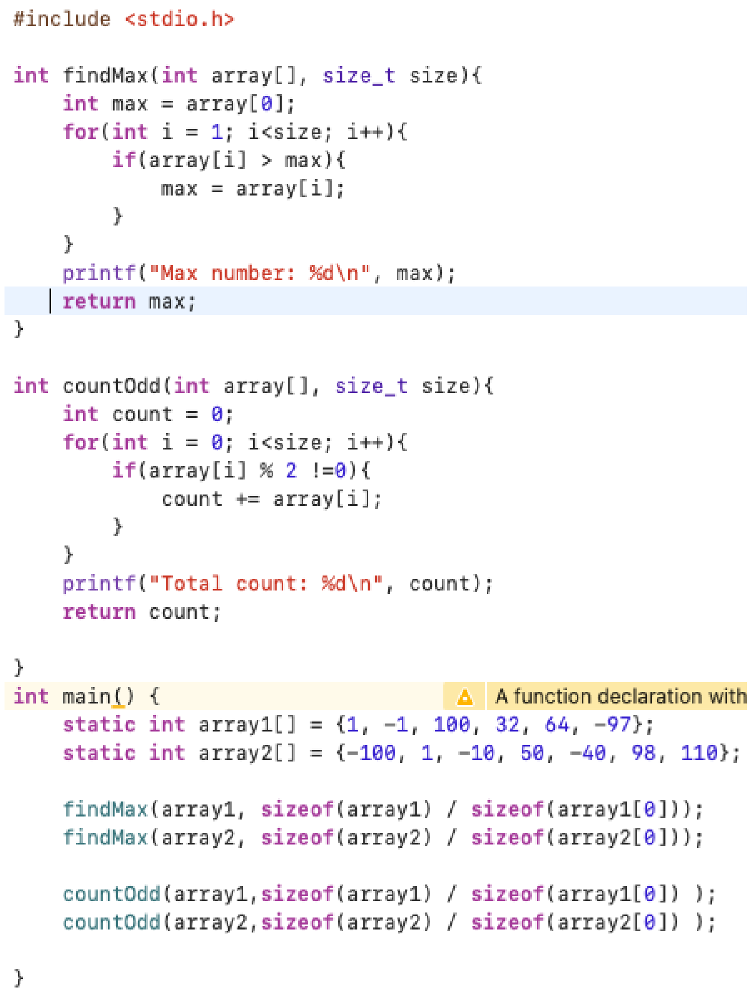
Explain your answer. (Hint: Recursion implies many subroutine calls.)

A stack would handle recursion more efficiently. The stack could grow as large as necessary to accommodate multiple calls to the subroutine. If there were only one register or one memory location, multiple calls to the subroutine from within the subroutine (i.e. recursion) would not be possible.

34. Write the following code segment in MARIE assembly language. (Hint: Turn the for loop into a while loop):

```
     Sum = 0;
     for X = 1 to 10 do
       Sum = Sum + X;

      Load     One      /Load constant
      Store    X        /Initialize loop control variable X
      Clear
      Store    Sum
Loop, Load     X        /Load X
      Subt     Ten      /Compare X to 10
      SkipCond 800     /If AC> 0 (X less than 10)
      Jump     Go
      Jump     Endloop /Terminate loop
Go,   Load     Sum
      Add      X       /Add X to Sum
      Store    Sum     /Store result in Sum
      Load     X
      Add      One     /Increment X
      Store    X
      Jump     Loop
Endloop, Load  Sum
      Output           /Print Sum
      Halt             /terminate program
Sum,  Dec 0
X,    Dec 0            /Storage for X
One,  Dec 1            /The constant value 1
Ten,  Dec 10           /The loop constant
      END
```



```c
#include <stdio.h>

int findMax(int array[], size_t size){
    int max = array[0];
    for(int i = 1; i<size; i++){
        if(array[i] > max){
            max = array[i];
        }
    }
    printf("Max number: %d\n", max);
    return max;
}

int countOdd(int array[], size_t size){
    int count = 0;
    for(int i = 0; i<size; i++){
        if(array[i] % 2 !=0){
            count += array[i];
        }
    }
    printf("Total count: %d\n", count);
    return count;

}
int main() {                                ⚠ A function declaration with
    static int array1[] = {1, -1, 100, 32, 64, -97};
    static int array2[] = {-100, 1, -10, 50, -40, 98, 110};

    findMax(array1, sizeof(array1) / sizeof(array1[0]));
    findMax(array2, sizeof(array2) / sizeof(array2[0]));

    countOdd(array1,sizeof(array1) / sizeof(array1[0]) );
    countOdd(array2,sizeof(array2) / sizeof(array2[0]) );

}
```