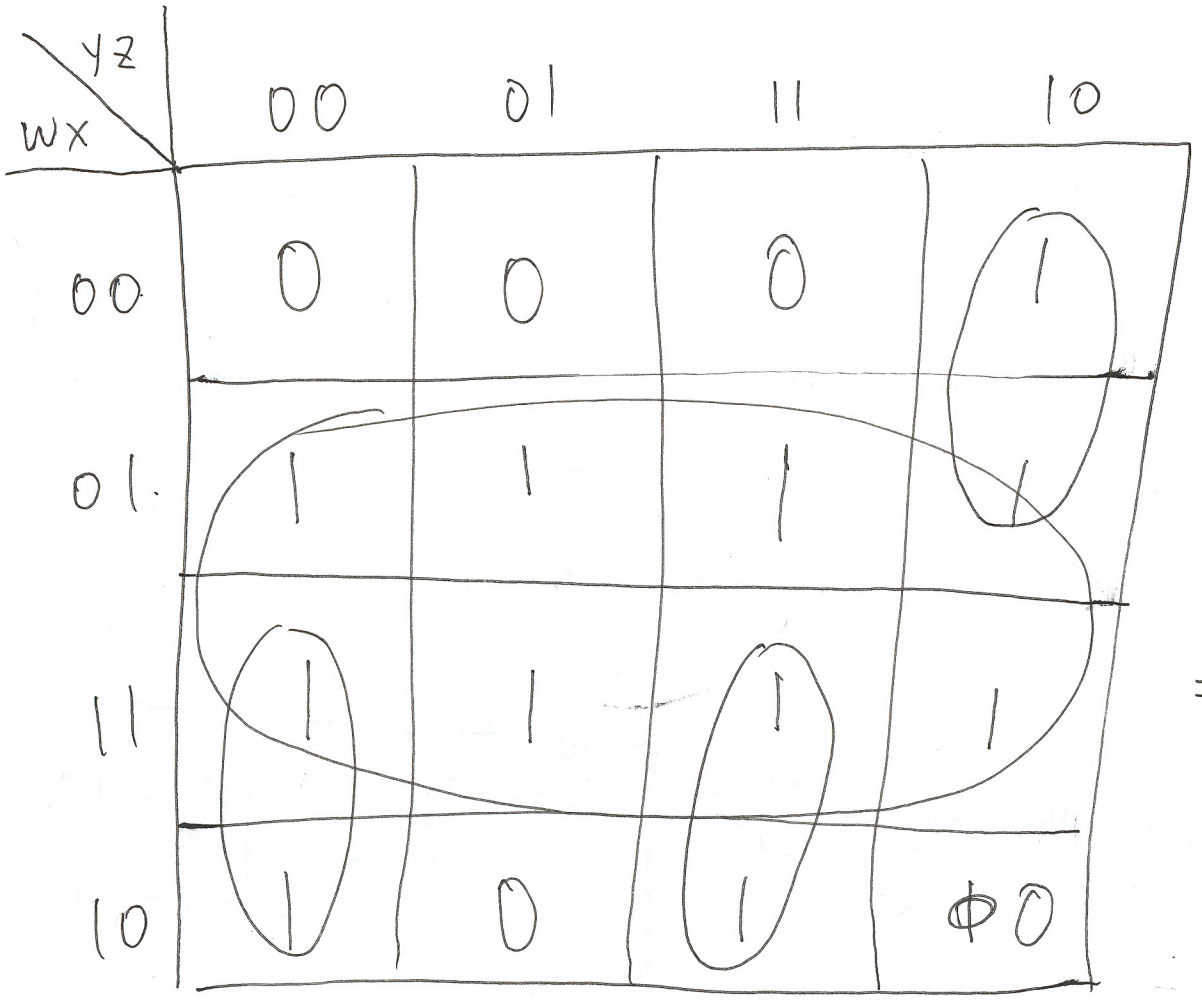
1 a)

w	x	y	z	b	
T	T	T	T	1	0
T	T	T	F	1	1
T	T	F	T	1	2
T	T	F	F	1	3
T	F	T	T	1	4
T	F	T	F	0	5
T	F	F	T	0	6
T	F	F	F	1	7
F	T	T	T	1	8
F	T	T	F	1	9
F	T	F	T	1	A
F	T	F	F	1	B
F	F	T	T	0	C
F	F	T	F	1	D
F	F	F	T	0	E
F	F	F	F	0	F

b)



c)

$x + w'yz' + wy'z' + wyz$

$= (w \text{ NAND } y \text{ NAND } z) \text{ NAND } (x')$

(b)

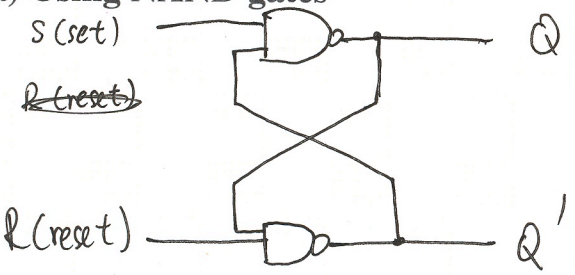
c) Using the Karnaugh maps, write the boolean expressions for segment a and b using a minimum "sum of products" boolean expression.

b =

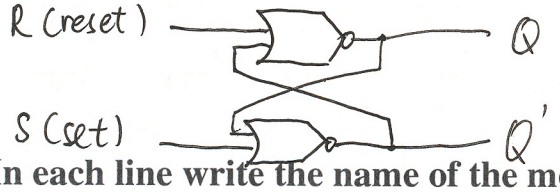
d) Draw the schematic of your implementation using NAND gates.

2. Draw a Flip Flop

a) Using NAND gates



b) Using NOR gates



3. In each line write the name of the memory section where the variables are stored:

```
int a = 5;  // a is stored in ____data____

int b[20];  // b is stored in ____bss____

int main() {  // main is stored in ____text____
{
    int x;    // x is stored in ____stack____

    int *p = (int *)malloc(sizeof(int)); // p points to memory stored in
____heap____
}
```

4. Enumerate the steps needed to load a program.

① The loader allocates space for all the sections of the executable file (text, data, bss etc)
② It loads into memory the executable and shared libraries (if not loaded yet)
③. It also write (resolves) any value in the executable to point to the functions/variable in the shared libraries @ (continued to the next page).

④. Once ~~the~~ memory image is ready, the loader jumps to the _start entry point that calls init() of all libraries and initializes static constructors. Then it calls main() and the program begins.

⑤ _start also calls exit() when main() returns.

5. What is a "Dynamic Linker" and a "Static Linker".

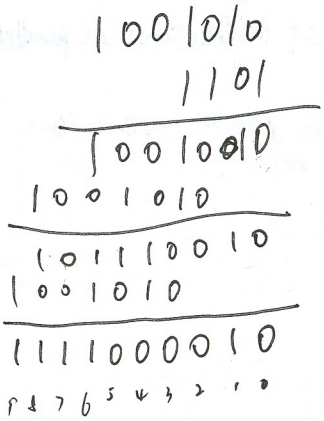
Static linker ~~links~~ copies all library routines used in the program into executable image. It required more space and memory but is faster and more portable. (.a file)

Dynamic linker places the name of a sharable library in the executable image. Actual linking with library routines does not occur until the image is run, when both the executable and the library are placed in memory. Multiple program can share one copy of the library

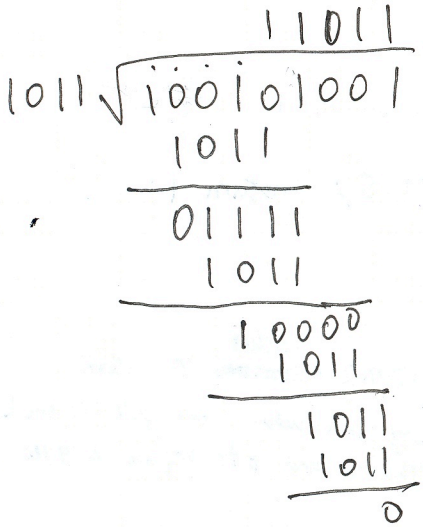
6. Perform the following multiplication in binary. Also, show convert to decimal the operands and the result.

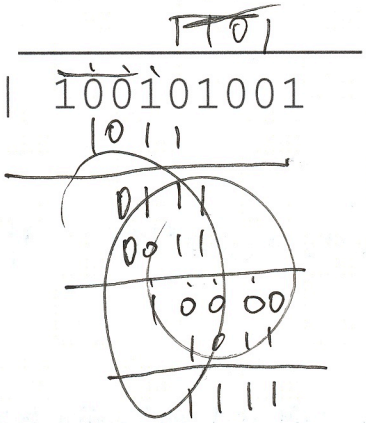
1001010 x 1101

$2^1+2^6+2^7+2^8+2^9 = 962$



7. Perform the following division in binary, Also, convert to decimal the operands and the result.





$11011_b = 27$

8. Represent the number1.25 in binary using the IEEE 754 double representation. The formula is given:

$$\text{Val in decimal} = (-1)^{s} \times (1.m) \times 2^{(e-bias)} \text{ where:}$$

$(2 + 2^{8-m}) \times 2^{x} = 1.25$

bias = 1023

s = bit 63

e = bits 52 to 62

m= bits 0 to 51

$m = -1$ → 1000

$X = 1022$

Val in binary: 0011 1111 1110 1000 0000 0000 0000 0000 0000

9. Explain what is a "Pipe Stall" and how can be avoided.

Pipeline is a hardware optimization technique. It allows the execution of instructions in parallel

Pipe stall occurs when the instruction depends on the result of the previous instruction

Avoid: ① reorder the instructions in order to delay the use of result.

② Avoid introducing unnecessary branches

③ Delay reference to result register(s)

10. Obtain the representation of number -58 using 8 bits. Verify that 78 + (-58) is equal to 20 using binary addition and complements of 2.

Signed magnitude: the first bit 0-+ 1--

One's complement: switch all 0&1s. the left most bit indicates 0-+ 1--

two's complement: (one's complement + 1.)

58: 00111010

-58: 11000101+1

= 11000110

78: 01001110

+(-58) + 11000110

00010100

proved!

$= 2^2 + 2^4 = 4 + 16 = 20$

11. Write a function "int isBigEndian()" that will return 1 if the computer is big endian or 0 if it is little endian.

little: 0x50 0x00

big: 0x00 0x50

```
int isBigEndian() {
    int i=5; char* p = (char*)&i; if (p*==5) return 0; return 1;
}
```

12. What is the difference between Harvard Architecture and Von ~~Newman~~ Architecture

Harvard Archi. has seperate data and instruction buses, Von Neumann has a single cache for storing data and program. It is possible to implement pipeline to do both simultaneously and achieve one cycle for one instruction. when Von ... has a single ~~memory~~ cach for storing data and program. pipelining is impossible and it require 2 cycles to execute one line.

Futhur more, Typically, code (text section) memory is read only, ~~so for Harvard.~~ and data is write-read, so for Harvard it is impossible to modify the program by itself. where in Von. There ~~buses can~~ is only one bus so it has to be write-read, it is possible for the program to modify itself.

13. What do CISC and RISC stand for and enumerate the characteristics of each.

CISC stands for complex Instruction Set computer. RISC stands for reduced instruction set computer

CISC chips are relatively slower per instruction, but use little instructions. CISC uses more complicated, ~~target~~ large, slower instructions but use less instructions ~~to~~ for a task.

14. Write the following characteristics of the ATMEGA328:

Program Memory (KB): 32
RAM (KB): 2
Clock Speed (Hz): 20M
Number of I/O ports: ~~22~~ 23
Number of A/D ports: 6

14. Rewrite the following program in AVR assembly language. Remember that int's in AVR are 2 bytes long.

```
int a = 6;
int b = 3;
int c;

void setup() {
 c = a + b;
}

loop()
{
}
```

```
lds  r24,b
lds  r25, (b)+1
lds  r18, a
lds  r19, (a)+1
add  r24, r18
abc  r25, r19
sts  (c)+1, r25
sts  C, r24      ret
```

15. Rewrite the following program in AVR assembly language. Remember that int's in AVR are 2 bytes long.

```
int a = 6;
int b = 4;
int c;

void setup() {
 c = a * b;
}

loop()
{
}
```

```
lds r20,b
lds r21, (b)+1
lds r18, a
lds r19,(a)+1
mul r20, r18
movw r24, r0
mul  r20, r19
add  r25, r0
mul r21, r18
add r25, r0
clr r1
sts (c)+1, r25
sts  C, r24
ret
```

16. In x86-64 Assembly language implement the function **int addarray(int n, int * array)** that add all the elements of the array
passed as parameter. n is the length of the array.

17. In x86-64 Assembly language implement a program "maxmin" that prompts two integer numbers fom stdin and then displays the maximum, minimum and average of both: numbers. Here is an example of the usage:

```
        .text
.globl addarray
        .type   addarray, @function
addarray:
        movl (%esi),%eax        #store the first number into eax
        movl %edi,%ecx          #store the pointer pointing to the first int from the array
while:  cmpl $0,%ecx            #compare the current n and 0
        jne loop                #jump to loop if current n does not equal to 0
        ret                     #return as eax stores the value
loop:   addl $-1,%ecx           #decrease current n by 1
        addl $4,%esi            #move the pointer 4 bytes ahead to get the next int
        addl (%esi),%eax        #add the new int with the eax
        jmp while               #go back to while loop
```

17.

```
.data
        a:      .int 0
        b:      .int 0
scanS:  .string "%d %d"
print1: .string "a=%d\n"
print2: .string "b=%d\n"
print3: .string "max=%d\n"
print4: .string "min=%d\n"
print5: .string "avg=%d\n"
.text
        .globl main
main:
        movq $scanS,%rdi
        movq $a,%rsi
        movq $b,%rdx
        movq $0,%rax
        call scanf

        movq $print1,%rdi
        movq a,%rsi
        movq $0,%rax
        call printf

        movq $print2,%rdi
        movq b,%rsi
        movq $0,%rax
        call printf

        movq a,%rbx
        movq b,%rcx
        cmpq %rbx,%rcx
        jge min
        jmp max

max:
        movq $print3,%rdi
        movq a,%rsi
        movq $0,%rax
        call printf

        movq $print4,%rdi
        movq b,%rsi
        movq $0,%rax
        call printf

        movq a,%rbx
        movq b,%rax
        addq %rbx,%rax
        movq $2,%rbx
        idivq %rbx


        movq $print5,%rdi
        movq %rax,%rsi
        movq $0,%rax
        call printf
        ret

min:
```

```
movq $print3,%rdi
movq b,%rsi
movq $0,%rax
call printf

movq $print4,%rdi
movq a,%rsi
movq $0,%rax
call printf

movq a,%rbx
movq b,%rax
addq %rbx,%rax
~~movq $2,%rbx~~ shrq $1, %rax
id~~ivq %rbx~~

movq $print5,%rdi
movq %rax,%rsi
movq $0,%rax
call printf
ret
```