Trabajo Practico Nº1

Nombre: Santiago vietto

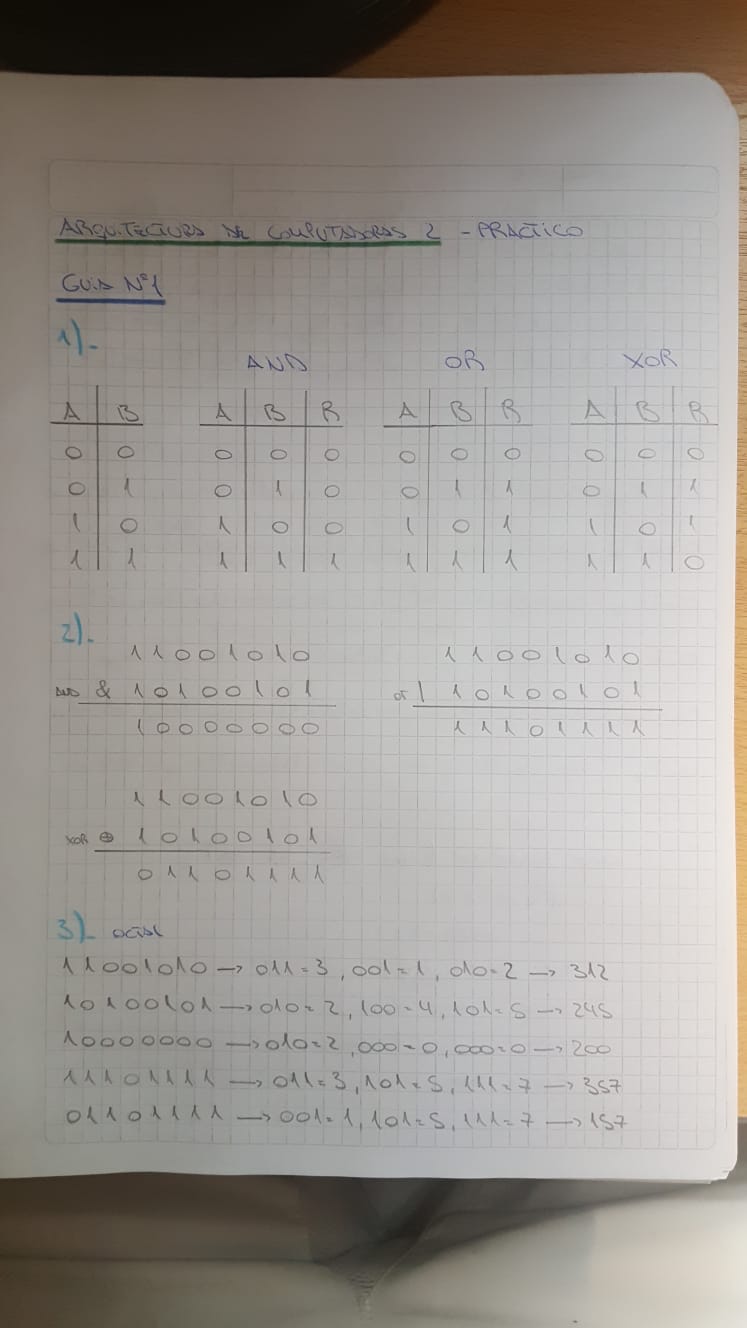
Materia: Arquitectura de computadoras 2

Docente: Ignacio Fazio

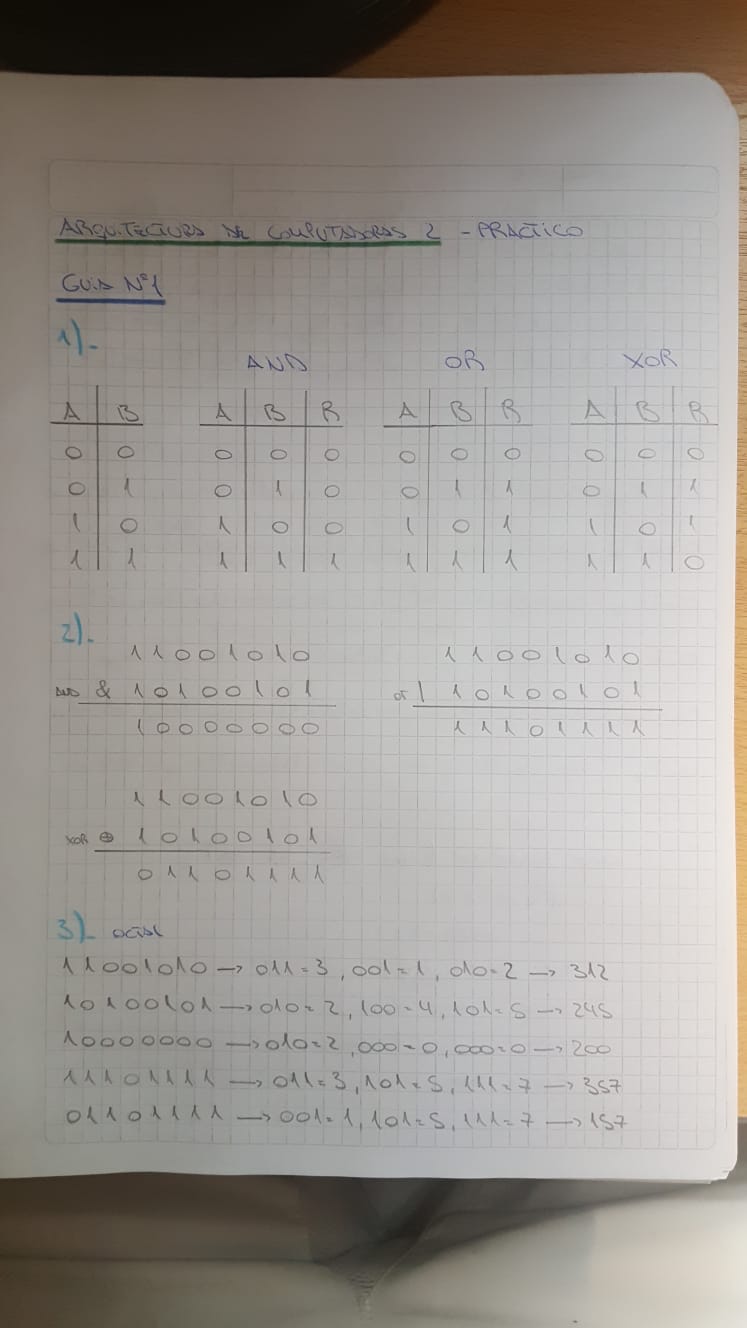
Clave: 1802890

Año: 2020

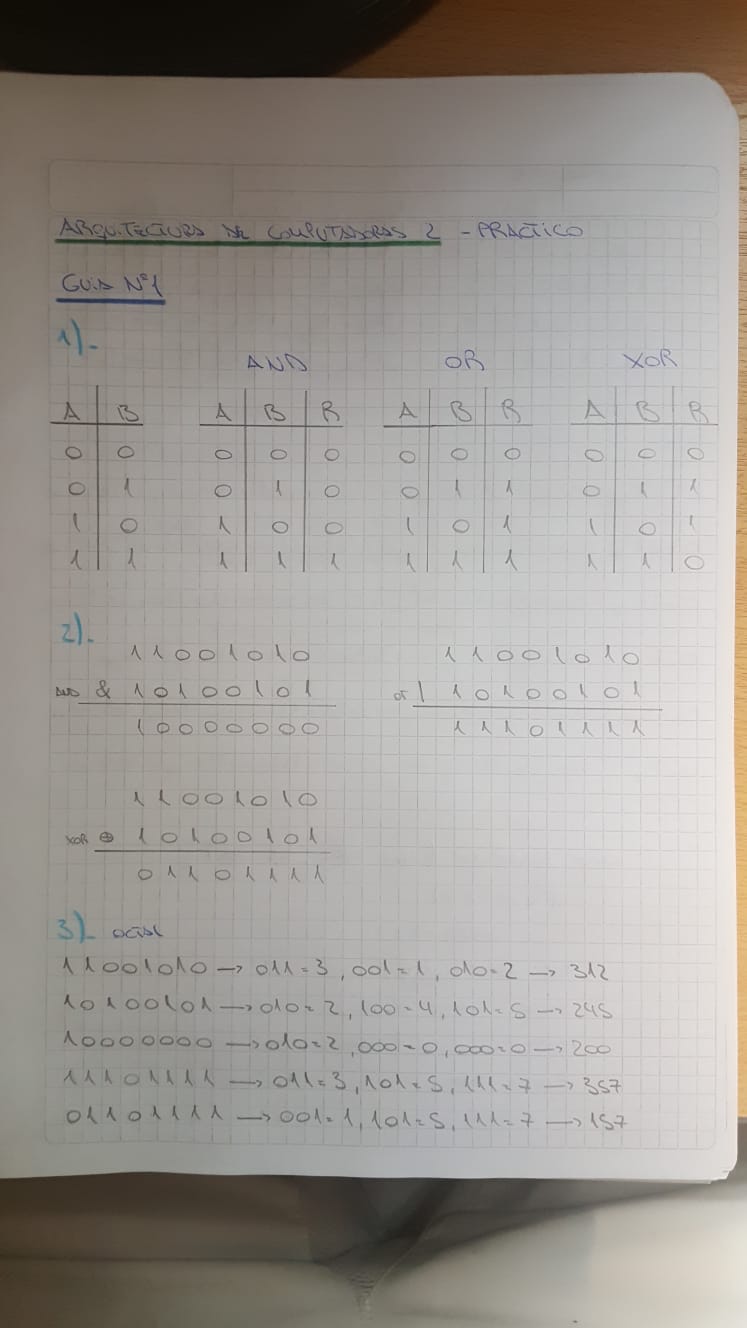
Ejercicio 1:



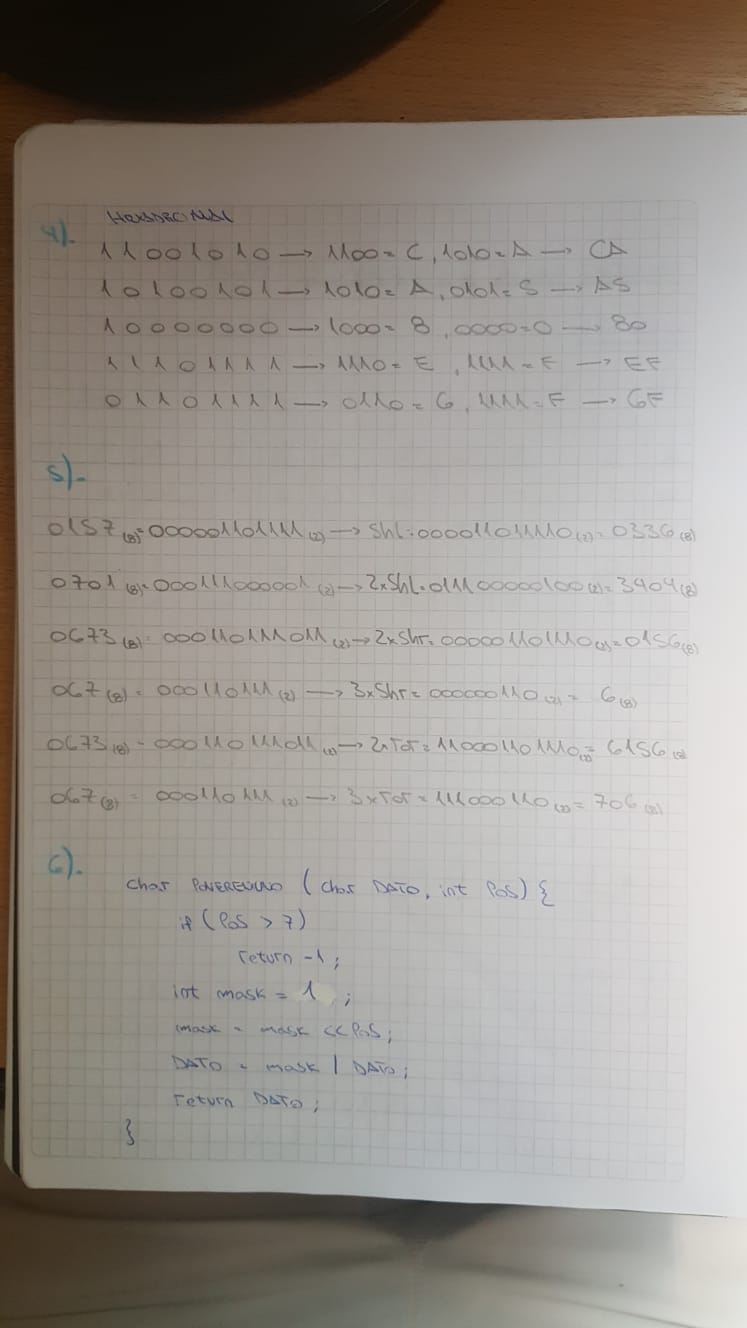
Ejercicio 2:

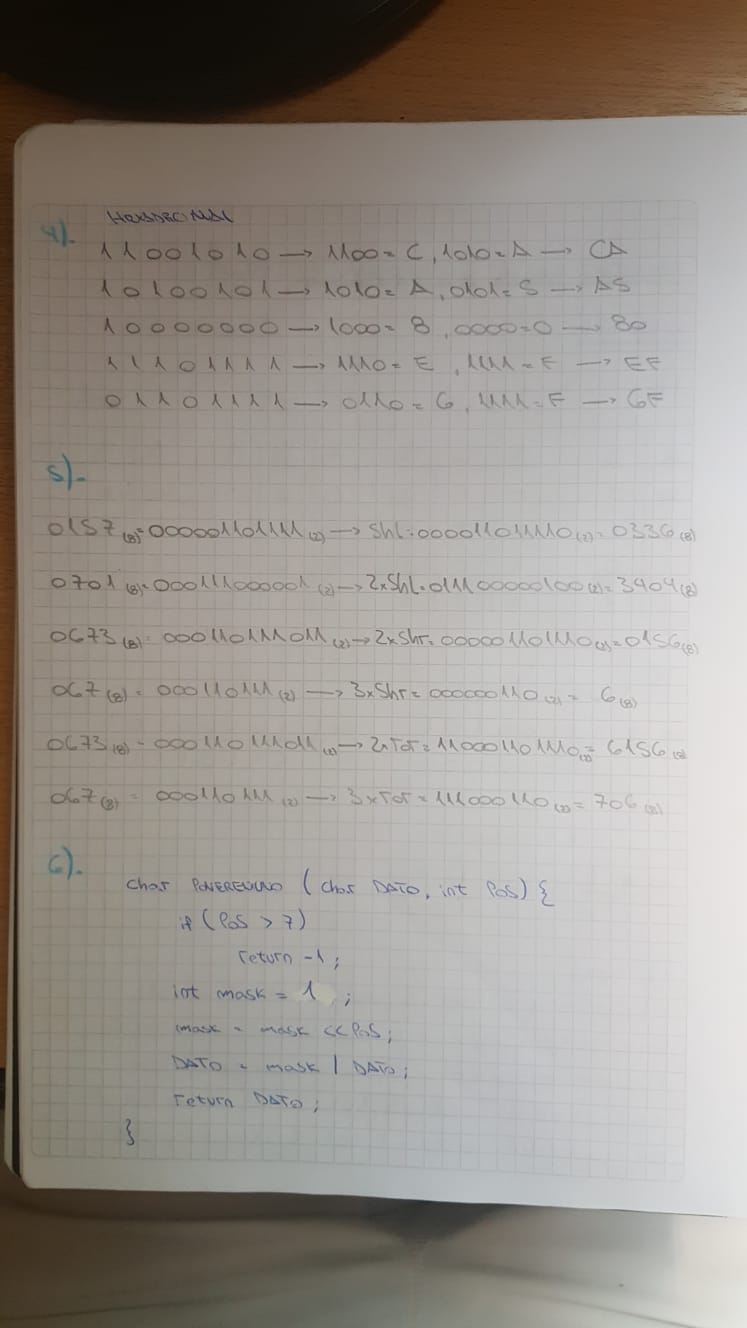


Ejercicio 3:



Ejercicio 4:



Ejercicio 5:

Ejercicio 6:

//Funcion poner en uno

char Set\_uno(char dato, int pos){

if (pos > 7){ //si la posicion es mayor a 7 no nos deja

return -1; }

else{

int mask = 0x1; //ponemos la mascara en 00000001

mask = mask << pos; //rotamos a la izquierda el numero de veces de la posicion

dato = mask | dato; //la barra | es un or, si es 0 o 1, lo pone en 1

return dato; }

}

//Funcion poner en cero

char Set\_cero(char dato, int pos){

if (pos > 7){ //si la posicion es mayor a 7 no nos deja

return -1; }

else{

int mask = 0xFE; //ponemos la mascara en 11111110

mask = mask << pos; // rotamos a la izquierda el numero de veces de la posicion

dato = mask & dato; //la barra & es un and, si es 0 o 1, lo pone en 0

return dato; } }

Ejercicio 7:

**A)\_**

**/\* Tomamos el RXCIE (bit 7), para definir LOW y HIGH \*/**

//En la funcion HI, la funcion debe poner en el bit 7 un 1 sin modificar los otros 7 bits

void RXCIE\_HI(){

int mask = 0x1;

mask = mask << 7; //rotamos a la izquierda hasta la posicion 7

puerto = puerto | mask; //ponemos el puerto en 1

}

//En este caso ponemos el bit 7 en 0 sin modificar el resto

void RXCIE\_LO(){

int mask = 0xFE;

mask = mask << 7; //rotamos a la izquierda hasta la posicion 7

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el TXCIE (bit 6), para definir LOW y HIGH \*/**

void TXCIE\_HI(){

int mask = 0x1;

mask = mask << 6; //rotamos a la izquierda hasta la posicion 6

puerto = puerto | mask; //ponemos el puerto en 1

}

void TXCIE\_LO(){

int mask = 0xFE;

mask = mask << 6; //rotamos a la izquierda hasta la posicion 6

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el UDRIE (bit 5), para definir LOW y HIGH \*/**

void UDRIE\_HI(){

int mask = 0x1;

mask = mask << 5; //rotamos a la izquierda hasta la posicion 5

puerto = puerto | mask; //ponemos el puerto en 1

}

void UDRIE\_LO(){

int mask = 0xFE;

mask = mask << 5; //rotamos a la izquierda hasta la posicion 5

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el RXEN (bit 4), para definir LOW y HIGH \*/**

void RXEN\_HI(){

int mask = 0x1;

mask = mask << 4; //rotamos a la izquierda hasta la posicion 4

puerto = puerto | mask; //ponemos el puerto en 1

}

void RXEN\_LO(){

int mask = 0xFE;

mask = mask << 4; //rotamos a la izquierda hasta la posicion 4

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el TXEN (bit 3), para definir LOW y HIGH \*/**

void TXEN\_HI(){

int mask = 0x1;

mask = mask << 3; //rotamos a la izquierda hasta la posicion 3

puerto = puerto | mask; //ponemos el puerto en 1

}

void TXEN\_LO(){

int mask = 0xFE;

mask = mask << 3; //rotamos a la izquierda hasta la posicion 3

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el CHR9 (bit 2), para definir LOW y HIGH \*/**

void CHR9\_HI(){

int mask = 0x1;

mask = mask << 2; //rotamos a la izquierda hasta la posicion 2

puerto = puerto | mask; //ponemos el puerto en 1

}

void CHR9\_LO(){

int mask = 0xFE;

mask = mask << 2; //rotamos a la izquierda hasta la posicion 2

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el RXB8 (bit 1), para definir LOW y HIGH \*/**

void RXB8\_HI(){

int mask = 0x1;

mask = mask << 1; //rotamos a la izquierda hasta la posicion 1

puerto = puerto | mask; //ponemos el puerto en 1

}

void RXB8\_LO(){

int mask = 0xFE;

mask = mask << 1; //rotamos a la izquierda hasta la posicion 1

puerto = puerto & mask; //ponemos el puerto en 0

}

**/\* Tomamos el TXB8 (bit 0), para definir LOW y HIGH \*/**

void TXB8\_HI(){

int mask = 0x1;

puerto = puerto | mask; //ponemos el puerto en 1

}

void TXB8\_LO(){

int mask = 0xFE;

puerto = puerto & mask; //ponemos el puerto en 0

}

**B)\_**

**/\* Creamos una funcion que nos lea en que estado esta el bit, para este caso TXCIE \*/**

//Esta se puede usar para todos los otros registros

bit TXCIE(){

puerto = puerto >> 6; //Tomamos el puerto que es el registro del puerto serie

puerto = puerto & 0x01; //Le aplicamos un and

if( puerto == 1){ //Preguntamos si el puerto es igual a uno

return true;

}else{

return false; }

}

**/\* Aplicando la parte A en TXEN lo ponemos en HIGH (bit 3) y devuelve return a la funcion TXCIE() que creamos antes \*/**

bool transmision\_completa() {

TXEN\_HI(); //TXEN debe estar en high para que TXCIE este en high

return TXCIE(); //Por defecto manda true (1), si el TXCIE es 0 manda false

}

**C)\_**

bool completa ;

completa = transmision\_completa(unsigned char TXCIE);

**D)\_**

bool transmision\_completa{

TXEN\_HI();

return TXCIE();

}

string transmision\_completa() {

TXEN\_HI();

if (TXCIE){

return “Transmicion completa”;

} else{

return “Transmicion fallida”;

}

Ejercicio 8:

**A)\_**

**/\* Aplicamos la misma logica del ejercicio 7 (A) \*/**

//Reemplazamos el mask por el puerto y lo rotamos a la derecha

bool getSPIE (unsigned char puerto) {

int mask = 0x1; //Le damos a una variable mask el valor de 1

puerto = puerto >> 7; //Rotamos el puerto 7 lugares

puerto = puerto & mask; //Aplicamos el and con la mascara en el primer bit

return puerto;

}

bool getCPOL (unsigned char puerto) {

int mask = 0x1; //Le damos a una variable mask el valor de 1

puerto = puerto >> 3; //Rotamos el puerto 3 lugares

puerto = puerto & mask; ; //Aplicamos el and con la mascara en el primer bit

return puerto;

}

**B)\_**

//Usamos la tabla para guiarnos

//Las funciones getSRP1 y getSRP2 son la misma logica que el punto A

**/\* Primer forma de resolver \*/**

int getPrescalingFactor(){ //Utilizamos if según el valor de cada funcion

if(!getSRP1()){

if(!getSRP0()){

return 2;

}else{

return 4;

}

}

if(getSRP1()){

if(!getSRP0()){

return 16;

}else{

return 32;

}

}

}

**/\* Segunda forma de resolver \*/**

int getPrescalingFactor2(){ //Multiplicamos según el valor de cada funcion

return ( (7\*getSRP1()) + 1) \* (2 + (2\*getSRP0()) );

}

**/\* Tercer forma de resolver \*/**

int getPrescalingFactor3(){ //Leemos el puerto y no entramos a funciones como los otros

array fd[4] = {2,4,16,32}; //Defino un array

puerto = puerto >> 1; //Hacemos una rotacion

puerto = puerto & 0x03;

return fd [puerto];

}

**/\* Cuarta forma de resolver \*/**

int getPrescalingFactor4(){ //Igual que la tercera pero uso las velocidades Hz

array fd[4] = {“1 Mhz”, ”500 Khz”, “125Khz”, “62,5 Khz”};

puerto = puerto >> 1;

puerto = puerto & 0x03;

return fd [puerto];

}

**C)\_**

void getCOMControl(){

int a[3] ={2, 4, 16, 32};

int Sp0=0 , Sp1=0;

Sp0 = SPR0;

Sp1 = SPR1;

cout<< “SPR0 es: ”<<Sp0<<endl;

cout<< “SPR1 es: ”<<Sp1<<endl;

cout<< “El factor de division es es: ”<< Sp0 + Sp1\*2 <<endl; }

**D)\_**

void setCPOL(unsigned char \*control, bool bit) {

if (bit) {

unsigned char mask = bit;

mask = mask << 3;

\*control = \*control | mask;

} else {

unsigned char mask = 0xFE;

mask = mask << 3;

\*control = \*control & mask;

}

}