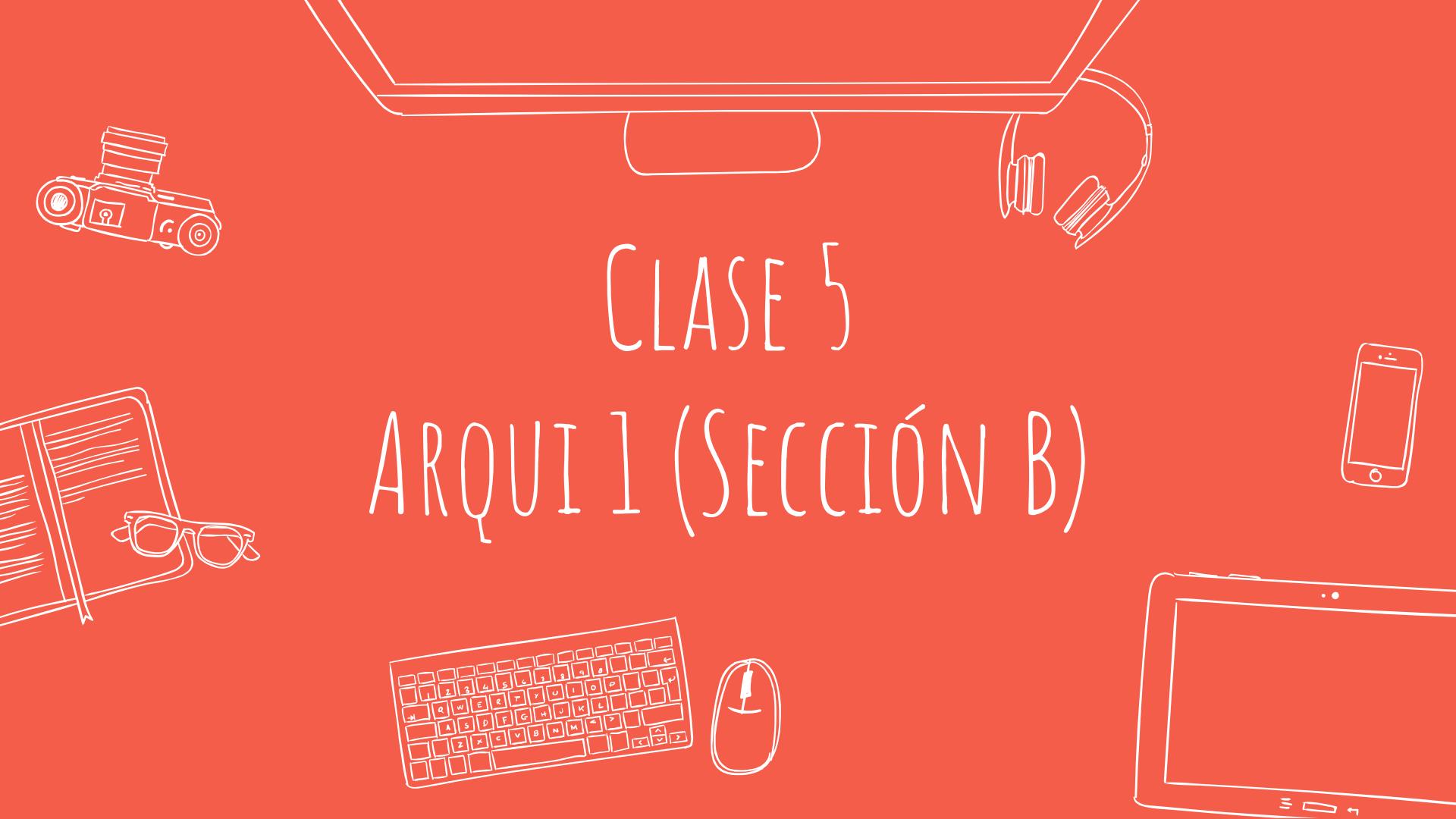


CLASE 5

ARQUI 1 (SECCIÓN B)



AGENDA 26/02

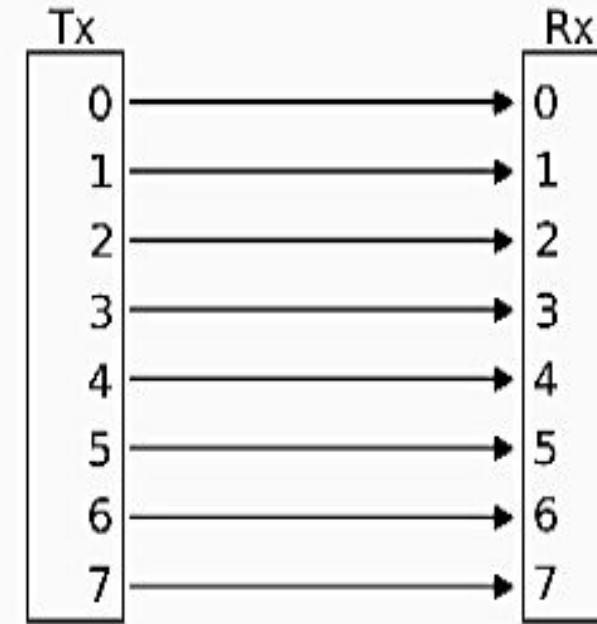
1. Dudas Proyecto 1
2. Tema Clase 5
3. Hoja de trabajo 2

TIPOS DE COMUNICACIÓN

- Comunicación Paralela
- Comunicación Serial

COMUNICACIÓN PARALELA

La comunicación en paralelo es un método de transmitir múltiples dígitos binarios simultáneamente.



COMUNICACIÓN SERIE

Transmiten sus datos un bit a la vez.

Estas interfaces pueden operar con tan solo un cable, por lo general nunca más de cuatro.

Parámetros:

- ✗ Velocidad de transmisión
- ✗ Bits de datos
- ✗ Bit de parada
- ✗ Bit de paridad

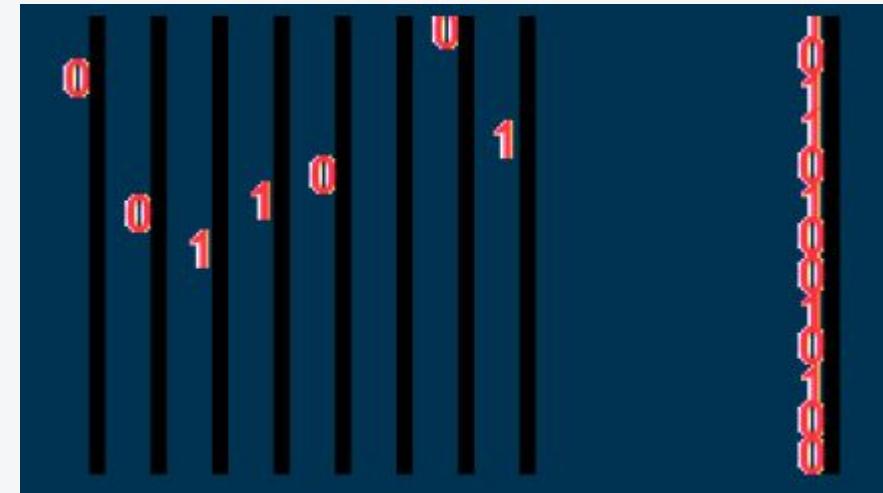
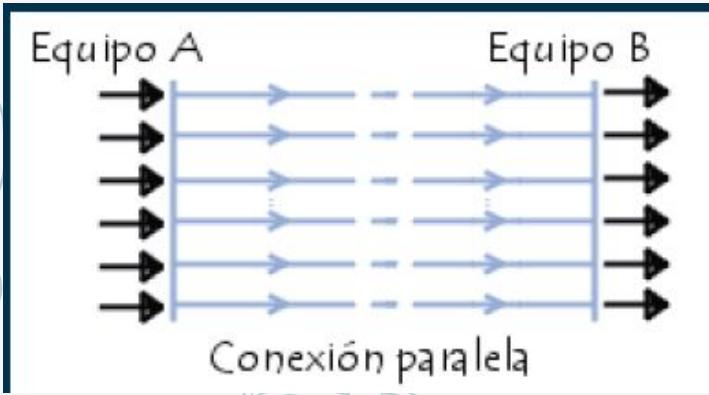
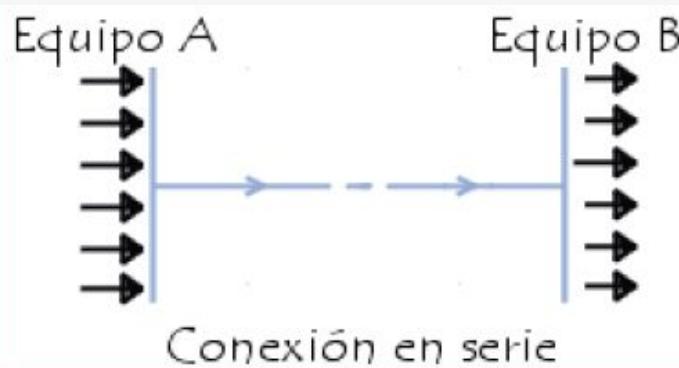
ESTRUCTURA Tamaño (bits)	Inicio	Datos	Paridad	Parada
	1	5-9	0-1	1-2

EJEMPLO TRANSMISIÓN SERIE

0	1	1	1	1	0	0	1	0	1	0	1	1	0	1	0	0	1	0	1	0	1	0	1	1
Inicio	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7	Parada	Inicio	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7	Parada	Inicio	Bit0	Bit1	Bit2	Bit3



COMUNICACIÓN SERIE VS PARALELA

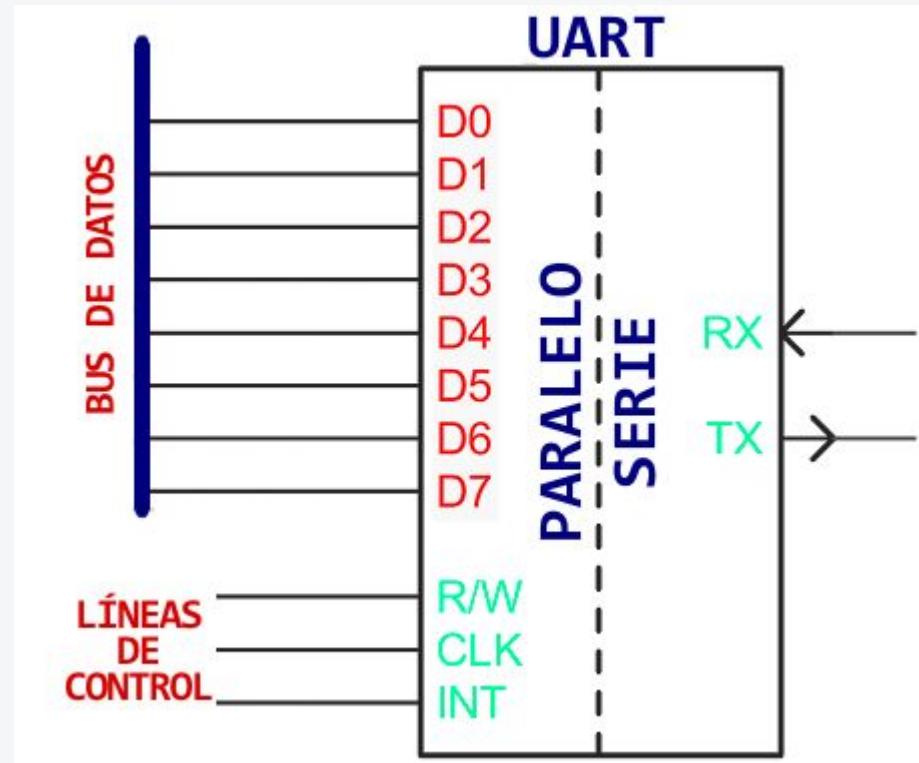




UART



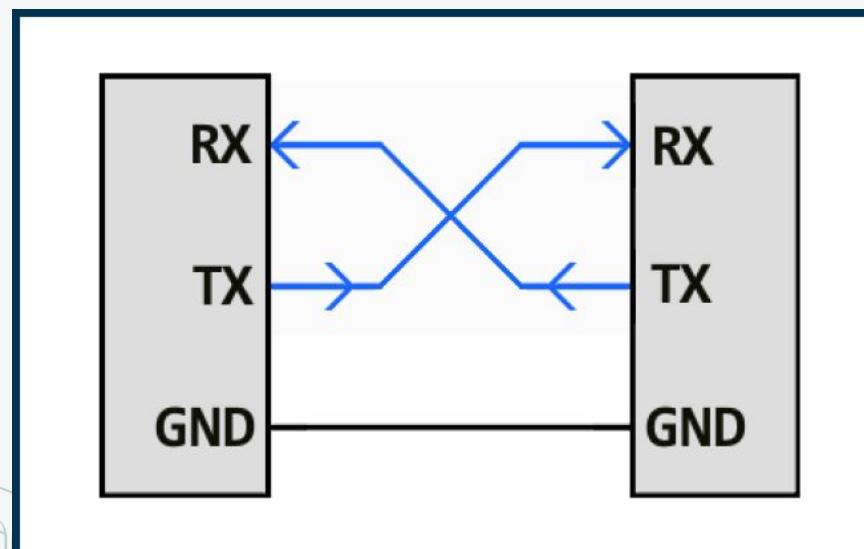
La última pieza de este armado en serie es encontrar algo para crear los paquetes en serie y controlar las líneas de hardware físico. Esto se concreta con un módulo llamado **UART** (Universal Asynchronous Receiver/Transmitter = Receptor/Transmisor Asíncrono Universal).



CONEXIÓN EN SERIE

Un bus serie consta sólo de 2 cables, uno para enviar datos y otro para recibir.

Entonces, los dispositivos serie deben tener dos pines serie: Rx, receptor y Tx, transmisor.



ARDUINO Y LA COMUNICACIÓN SERIAL

Nombre Función	Función
begin()	establece la velocidad de la UART en baudios para la transmisión serie, también es posible configurar el número de bits de datos, la paridad y los bits de stop, por defecto es 8 bits de datos, sin paridad y un bit de stop.
read()	lee el primer byte entrante del buffer serie.
write()	escribe datos en binario sobre el puerto serie. El dato es enviado como un byte o serie de bytes.
print()	imprime datos al puerto serie como texto ASCII, también permite imprimir en otros formatos.
available()	da el número de bytes (caracteres) disponibles para leer en el puerto serie, son datos que han llegado y se almacenan en el buffer serie que tiene un tamaño de 64 bytes.
end()	deshabilita la comunicación serie permitiendo a los pines RX y TX ser usado como pines digitales.
if(Serial)	especifica si el puerto serie está listo.
find()	lee datos del buffer serie hasta encontrar el string buscado.

Nombre Función	Función
parseInt()	busca el siguiente entero válido en el stream de datos del puerto serie.
readBytes()	lee datos del buffer serie y lo guarda en una variable buffer.
setTimeout()	configura el máximo de milisegundos de espera para la lectura del puerto serie. Por defecto es un segundo.
readBytesUntil()	lee caracteres del buffer serie y los guarda en un array de caracteres, la función termina si el carácter terminado es encontrado o la longitud determinada se a leido o ha alcanzado el timeout.
serialEvent()	llamado cuando hay datos disponibles.
flush()	espera hasta la transmisión completa de los datos salientes.
peek()	devuelve el siguiente carácter del buffer serie pero sin borrarlo de él.
readString()	lee caracteres del buffer serie y los guarda en un string. La función termina cuando se produce un timeout.

¿CÓMO MANEJA LOS DATOS ARDUINO?

Memoria SRAM, EPROM Y FLASH

MEMORIA FLASH

Memoria NO VOLÁTL

Memoria de programa. Usualmente desde 1 Kb a 4 Mb (controladores de familias grandes). Es donde se guarda el sketch ya compilado. Sería el equivalente al disco duro de un ordenador. En la memoria flash también se almacena del bootloader. Se puede ejecutar un programa desde la memoria flash, pero no es posible modificar los datos, sino que es necesario copiar los datos en la SRAM para modificarlos.

MEMORIA EEPROM

MEMORIA NO VOLÁTIL

- ✗ Para mantener datos después de un reset. Se puede grabar desde el programa del microcontrolador, usualmente, constantes de programa
- ✗ Tienen un número limitado de lecturas/escrituras
 - tener en cuenta a la hora de usarla.
- ✗ Esta memoria solo puede leerse byte a byte
Tiene una vida útil de unos 100.000 ciclos de escritura.

MEMORIA SRAM

Memoria de Arduino UNO:
Flash 32k bytes
SRAM 2k bytes
EEPROM 1k byte

Memoria de Arduino MEGA:
Flash 256k bytes
SRAM 8k bytes
EEPROM 4k byte

STATIC RANDOM ACCESS MEMORY

- X Variables locales, datos parciales. Usualmente se trata como banco de registros y memoria volátil. Es la zona de memoria donde el sketch crea y manipula las variables cuando se ejecuta.
- X Es un recurso limitado y debemos supervisar su uso para evitar agotarlo.
- X Es de tipo volátil

REGISTROS DE ARDUINO

Donde X hace referencia al número de puerto.

✗ REGISTROS PORTX

Es el que controla si el PIN está en un nivel alto o bajo.

✗ REGISTROS PINX

Este nos sirve para leer el estado de un PIN que es un input. Únicamente es de lectura no se puede escribir un valor.

✗ REGISTROS DDRX

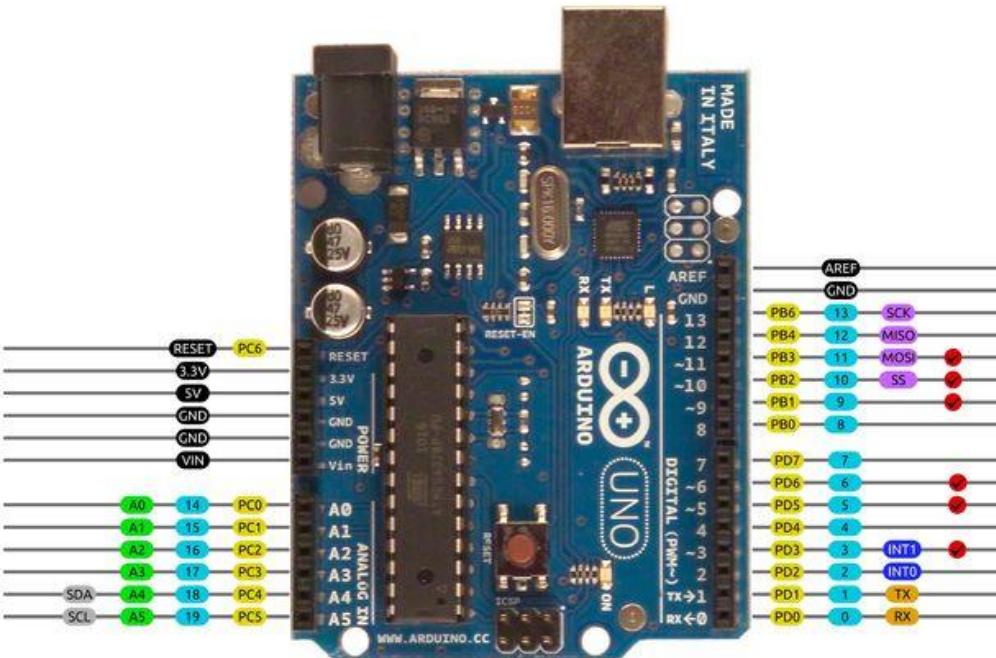
Determina o establece si el PIN es una entrada o es una salida, se puede usar para leer o escribir, cómo actuará el PIN si de entrada o salida.

Ejemplo: Arduino UNO

Configurar los pines digitales del 2 al 7 como Salida.

```
void setup(){
    DDRD= B11111100; //Utilizamos D porque es del 0 al 7 y B porque es
    //En este ejemplo no utilicé los pines 0 y 1 del Arduino.
    /*Esta instrucción es lo mismo que hacer esto:
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    pinMode(4,OUTPUT);
    pinMode(5,OUTPUT);
    pinMode(6,OUTPUT);
    pinMode(7,OUTPUT); */
}
```

Arduino Uno Pinout

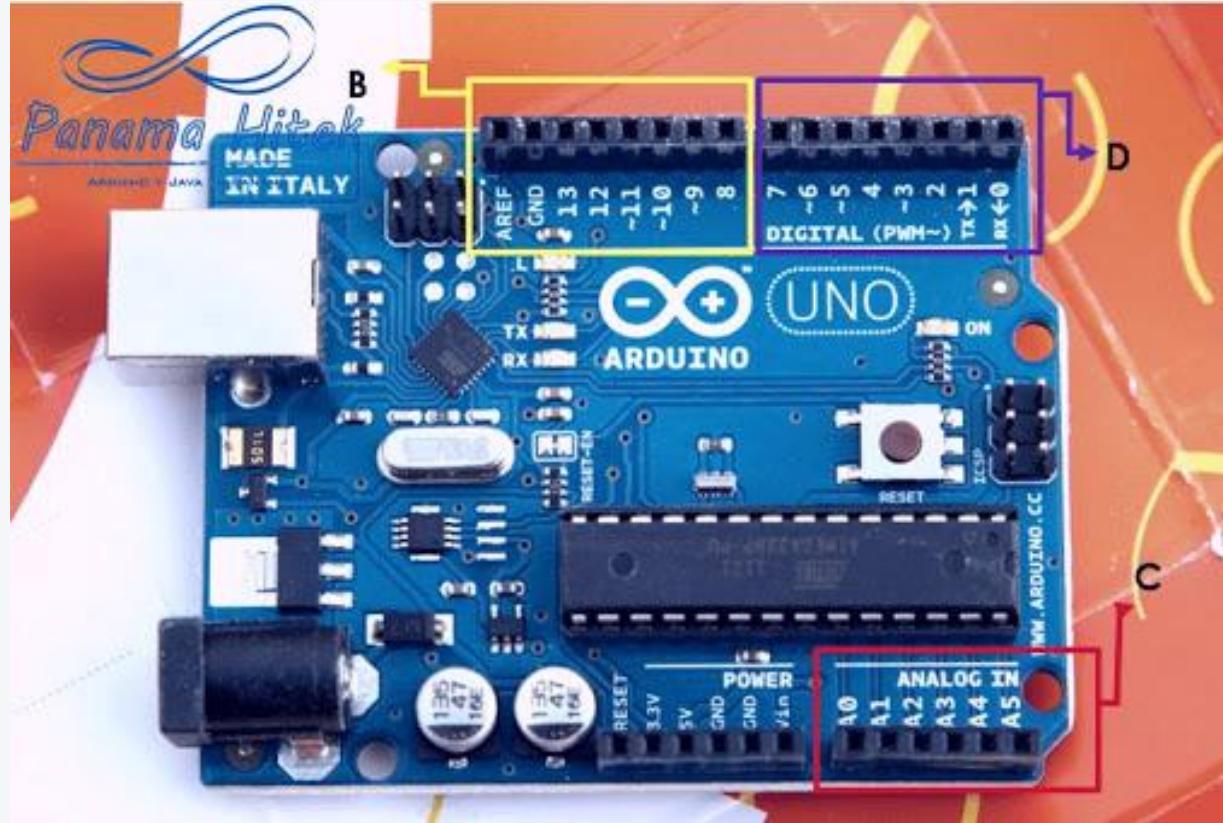


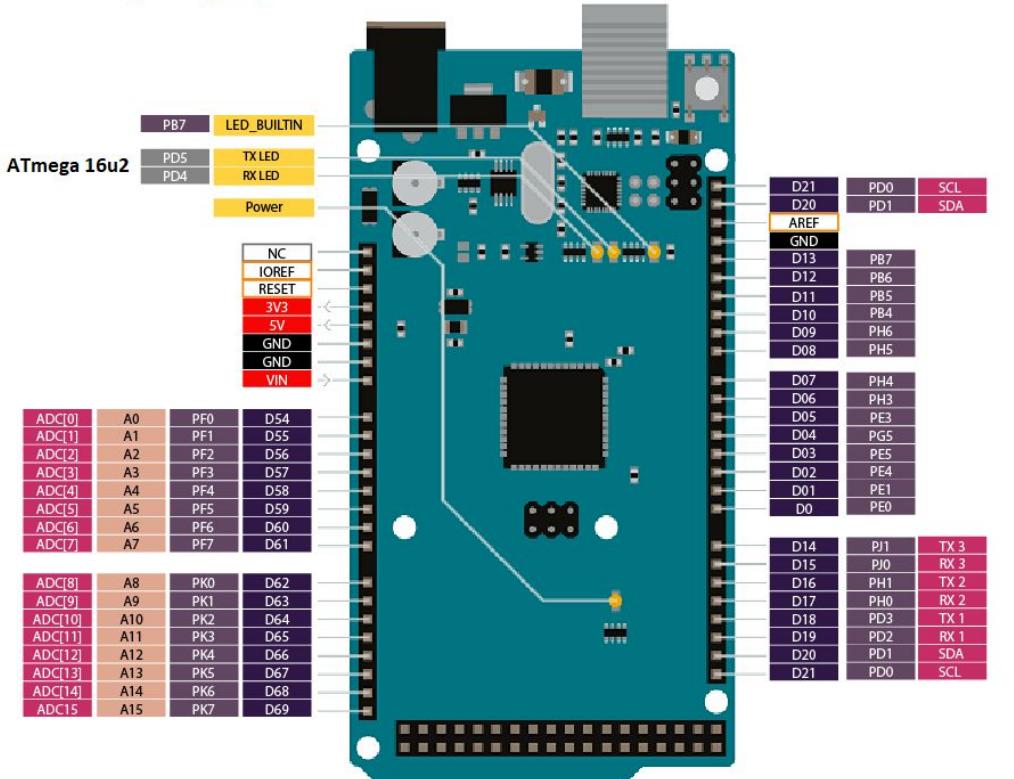
AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT



2014 by Bouni
Photo by Arduino.cc

MAPEO DE PUERTOS ARDUINO 1





Arduino Mega 2560 Rev3 Pinout

VENTAJAS

- ✗ Se pueden cambiar los pines de estado muy rápido, en fracciones de microsegundos. Las funciones `digitalRead()` y `digitalWrite()` son cada una cerca de una docena de líneas de código, lo cual al ser compilado se convierte en unas cuantas instrucciones máquina. Cada instrucción máquina necesita un ciclo de reloj a 16 MHz, lo cual puede sumar mucho tiempo en aplicaciones muy dependientes del tiempo.
- ✗ El Registro PORT (Puerto) puede hacer el mismo trabajo en muchos menos ciclos de trabajo.
- ✗ Algunas veces necesitamos configurar muchos pines exactamente al mismo tiempo.

Por lo que usar las funciones `digitalWrite (10,HIGH)`, seguida de la función `digitalWrite (11,HIGH)`, causará que el pin 10 se ponga en nivel alto, varios microsegundos después el pin 11, lo cual puede confundir circuitos digitales conectados al Arduino, cuyo funcionamiento dependa del tiempo preciso del cambio de esos bits.

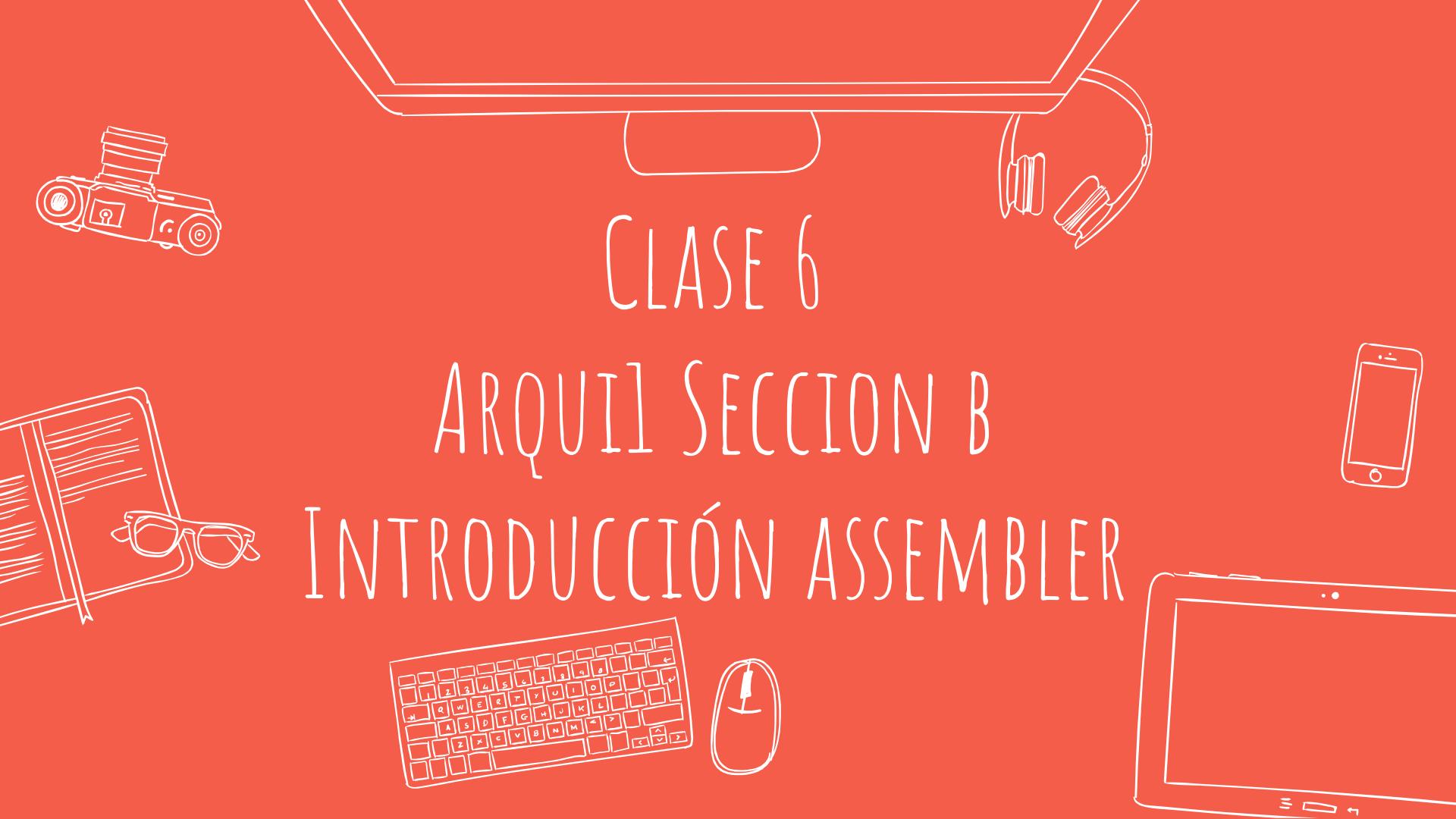
HOJA DE TRABAJO # 2

Realizar de manera individual (Utilizando únicamente registros PORT, DDRy PIN). Restringido el uso de sentencias digitalRead, digitalWrite, pinMode, analogRead, analogWrite.

Conectar dos display de 7 segmentos al arduino, directamente (sin uso de drivers). Mostrar su número de carnet en el primero, y en el segundo los siguientes números: 20210226

Entregar el código arduino y el diagrama en proteus en un archivo .zip con nombre [ACYE1]HT2_#carnet

Subirla a uedi antes de las 23:59 del 03/03



CLASE 6

ARQUI SECCION B

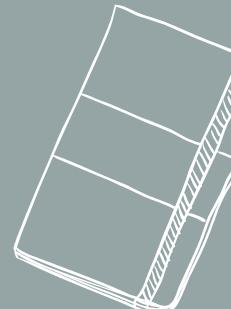
INTRODUCCIÓN ASSEMBLER



AGENDA 05/03

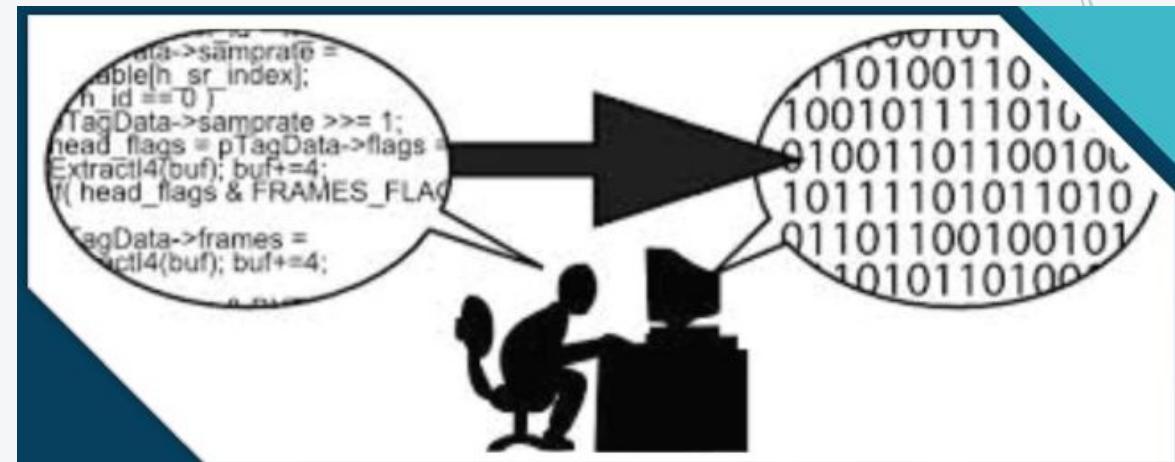


1. Dudas Proyecto 1
2. Tema Clase 6
3. Ejercicio Proyecto 1
4. Tarea 4



El ordenador sólo entiende el lenguaje de código binario ó mejor conocido como código máquina sólo utiliza 0 y 1 para decodificar cualquier acción.

¿CÓMO NOS ENTIENDE NUESTRO ORDENADOR?



LENGUAJES DE BAJO NIVEL

Un lenguaje de programación de características de bajo nivel o de primera generación, es aquel en el que sus instrucciones ejercen un control directo sobre el hardware y están condicionados por la estructura física de las computadoras que lo soportan. El uso de la palabra bajo en su denominación no implica que el lenguaje sea menos potente que un lenguaje de alto nivel, sino que se refiere a la reducida abstracción entre el lenguaje y el hardware.

T
I
P
O
S

D
E
L
E
N
G
U
A
J
E
S

LENGUAJE DE ALTO NIVEL
(JAVA, C#, PHP, PYTHON, ETC)

LENGUAJE DE BAJO NIVEL
ENSAMBLADOR

CÓDIGO MÁQUINA
(BINARIO [0-1])

HARDWARE
(PARTE FISICA DE LA COMPUTADORA)



ESTRUCTURA DE LOS LENGUAJES

Lenguaje Máquina:

Es el lenguaje básico, sólo admite todo 1 o nada o Todo sistema informático está basado en este código, ya que el 1 quiere decir que se permite el paso de la electricidad y el 0 no lo permite.

Lenguaje Bajo Nivel:

También denominados nemotécnicos o nemónicos, no son ya programas ejecutables directamente por el ordenador, sino textos de código fuente que necesitan de alguna herramienta para su conversión a lenguaje máquina, son los programas llamados ensambladores. Sus instrucciones suelen ser una denominación abreviada de la instrucción máquina que simbolizan, y tienen una correspondencia casi directa a las instrucciones máquina que representan.

Lenguaje Alto Nivel:

Son aquellos que permiten una máxima flexibilidad al programador a la hora de abstraerse o de ser literal

Permiten un camino bidireccional entre el lenguaje máquina y una expresión casi oral entre la escritura del programa y su posterior compilación Estos lenguajes están orientados a objetos.

¿QUÉ ES EL LENGUAJE ENSAMBLADOR?

- ✗ Derivado del lenguaje máquina, formado por abreviaturas de letras y números llamadas mnemotécnicos.
- ✗ Cada enunciado produce exactamente una instrucción máquina.
- ✗ Tienen acceso a todas las características e instrucciones disponibles en la máquina.

REGISTROS DE SEGMENTOS

- ✖ **DS, Data Segment (Segmento de Datos):** La dirección inicial de un segmento de datos de programa es almacenada en el registro DS.
- ✖ **CS,Code Segment (Segmento de Código):** DOS almacena la dirección inicial del segmento de código de un programa en el registro CS.
- ✖ **SS,Stack Segment (Segmento de Pila):** Permite la colocación en memoria de una pila, para almacenamiento temporal de direcciones y datos.
- ✖ **ES, Extra Segment (Segmento Extra):** Algunas operaciones con cadenas de caracteres (datos de caracteres) utilizan el registro extra de segmento para manejar el direccionamiento de memoria.

REGISTROS DE PROPÓSITO GENERAL



Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética.



Los registros son direccionables por medio de un nombre.



Los bits por convención, se numeran de derecha a izquierda, como en:

... 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Los **registros de propósito general** son los encargados de almacenar temporalmente los datos con los que el procesador realiza las operaciones, y de guardar los resultados de dichas operaciones.

REGISTROS DE PROPÓSITO GENERAL

Registros de Propósito General

64 bits	32 bits	16 bits	8 bits	8 bits
RAX	EAX	AX	AH	AL
RBX	EBX	BX	BH	BL
RCX	ECX	CX	CH	CL
RDX	EDX	DX	DH	DL
RSI	ESI	SI	No disponible	
RDI	EDI	DI	No disponible	
RSP	ESP	SP	No disponible	
RBP	EBP	BP	No disponible	

REGISTROS DE PROPÓSITO GENERAL

- ✗ AX, AH, AL Acumulador a menudo conserva el resultado temporal después de una operación aritmética o lógica.
- ✗ BX, BH, BL Base Se utiliza para guardar la dirección base de listas de datos en la memoria.
- ✗ CX, CH, CL Contador Contiene el conteo para ciertas instrucciones de corrimientos y rotaciones, de iteraciones en el ciclo loop y operaciones repetidas de cadenas
- ✗ DX, DH, DL Datos Contiene la parte más significativa de un producto después de una multiplicación la parte más significativa del dividendo antes de la división

REGISTROS ÍNDICES Y APUNTADORES

Los registros SP (apuntador de la pila) Y BP (apuntador de base) están asociados con el registro SS y permiten al sistema accesar datos en el segmento de la pila.

Índices trabajan con memorias aleatorias

Apuntadores trabajan con memorias estáticas

Índice:
SI - Source index
DI - Destination index

Apuntador:
BP - Base Pointer
SP - Stack Pointer

REGISTROS ÍNDICES Y APUNTADORES

No pueden conectarse directamente con los registros de segmentos.

No pueden verse como dos registros de 8 bits (parte alta y parte baja).

- ✗ SI: Source Index, o Apuntador a la Fuente
- ✗ DI: Destination Index, Apuntador al Destino

- ✗ BP: Base Pointer, o apuntador a la Base de la Pila.
- ✗ SP: Stack Pointer, o apuntador al tope de la Pila.

DOS

```
flat assembler 1.67.21 | 00:03:45

; FASM example of writing 32-bit program using DPMI
; requires DPMI host ( HDPMI32, CWSDPMI, etc. ) installed in system

format MZ
heap 0 ; no additional memory
stack $8000

segment loader use16

    push cs
    pop ds
    jmp @f

text: db 'Hello',$0D,$0A,$24

@@:   mov ax,$1687
      int $2F
      or ax,ax ; DPMI installed?
      jnz error
      test bl,1 ; 32-bit programs supported?
      jz error
      mov word [mode_switch],di

1/1 - USEDPMI.ASM | Row 11/113, Column 46/80
```

DOS

DOS (sigla de Disk Operating System, "Sistema Operativo de Disco" y "Sistema Operativo en Disco") es una familia de sistemas operativos para computadoras personales (PC). Creado originalmente para computadoras de la familia IBM PC, que utilizaban los procesadores Intel 8086 y 8088, de 16 bits, siendo el primer sistema operativo popular para esta plataforma.

DOS

Contaba con una interfaz de línea de comando en modo texto o alfanumérico, vía su propio intérprete de órdenes, command.com. Probablemente la más popular de sus variantes sea la perteneciente a la familia MS-DOS, de Microsoft, suministrada con buena parte de los ordenadores compatibles con IBM PC, en especial aquellos de la familia Intel, como sistema operativo independiente o nativo, hasta la versión 6.22, frecuentemente adjunto a una versión de la interfaz gráfica de Windows de 16 bits

DOSBox

DOSBox es un emulador que recrea un entorno similar al sistema DOS con el objetivo de poder ejecutar programas y videojuegos originalmente escritos para el sistema operativo MS-DOS de Microsoft en computadoras más modernas o en diferentes arquitecturas (como Power PC). También permite que estos juegos funcionen en otros sistemas operativos como GNU/Linux. Fue hecho porque Windows XP ya no se basa en MS-DOS y paso a basarse a Windows NT.

DOSBOX

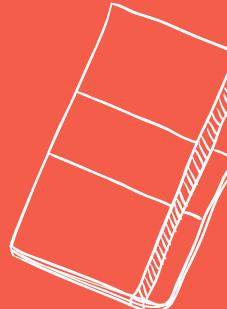
En la actualidad es un programa que resulta muy útil para los amantes de aquellos lenguajes de programación como el IDE de Turbo C, famoso en la programación a la vieja escuela; es importante resaltar, que este emulador también es hoy posible usarlo en todos los entornos de linux.

LENGUAJE ENSAMBLADOR

El lenguaje ensamblador, o assembler (en inglés assembly language y la abreviación asm), es un lenguaje de programación de bajo nivel. Consiste en un conjunto de mnemónicos que representan instrucciones básicas para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura de procesador y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador.



Cada arquitectura de procesador tiene su propio lenguaje ensamblador que usualmente es definida por el fabricante de hardware



LENGUAJE ENSAMBLADOR

Está basada en los mnemónicos que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria y otras características del lenguaje. Un lenguaje ensamblador es por lo tanto específico de cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que idealmente son portables.

MNEMÓNICO

En informática, un mnemónico o nemónico es una palabra que sustituye a un código de operación (lenguaje de máquina), con lo cual resulta más fácil la programación, es de aquí de donde se aplica el concepto de lenguaje ensamblador.

MNEMÓNICO

Un ejemplo común de mnemónico es la instrucción MOV (mover), que le indica al microprocesador que debe asignar datos de un lugar a otro. El microprocesador no entiende palabras, sino números binarios, por lo que es necesaria la traducción del término mnemónico a código objeto.

EJEMPLO

La sentencia: **MOV AL, 61h**

Asigna el valor hexadecimal 61 (97 decimal) al registro "AL". El programa ensamblador lee la sentencia de arriba y produce su equivalente binario en lenguaje de:

Binario: 10110000 01100001 (hexadecimal: B61) El mnemónico MOV es un código de operación u "opcode". El opcode es seguido por una lista de argumentos o parámetros, completando una típica instrucción de ensamblador. En el ejemplo, AL es un registro de 8 bits del procesador, al cual se le asignará el valor hexadecimal 61 especificado.

El código de máquina generado por el ensamblador consiste de 2 bytes. El primer byte contiene empaquetado la instrucción MOV y el código del registro hacia donde se va a mover el dato:

1011 0000 01100001

+---- Número 61h en binario

+---- Registro AL

+----- Instrucción MOV

EJEMPLOS DE INTERRUPCIONES

int 01h-->un solo paso

int 02h-->interrupcion no enmascarable

int 03h--> punto de interrupcion

int 04h-->desbordamiento

int 05h-->impresion de pantalla

int 08h-->Cronometro

int 15h-->Servicios del sistema

int 16h-->Funciones de entrada del teclado

int 18h-->Entrada con el Basic de Rom

int 19h-->Cargador ed arranque

int 1Ah-->Leer y establecer la hora

int 1Bh-->Obtener el control con una interrupcion de teclado.

int 20h-->Terminar un programa

int 33h->Funciones del Raton

TAREA 4

La Tarea consiste en:

- ✗ Descargar e Instalar DosBox
- ✗ Descargar e Instalar un Lenguaje Ensamblador (MASM ó NASM)
- ✗ Investigar sobre los ensambladores Masm, Nasm Tasm y Fasm.

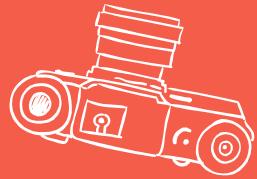
Entregar en UEDI un pdf con la investigación. Con el nombre:
[ACYE1]Tarea4_#carnet.

Tienen hasta el miércoles 10 antes de las 23:59

CLASE 7

ARQUI SECCION B

ASSEMBLER

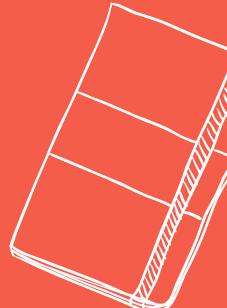


AGENDA 12/03

1. Dudas Proyecto 1
2. Hoja Calificación Proyecto 1
3. Práctica 3 (Assembler)
4. Tema Clase 7
5. HT3

BREVE REPASO

CLASE 6



Los **registros de propósito general** son los encargados de almacenar temporalmente los datos con los que el procesador realiza las operaciones, y de guardar los resultados de dichas operaciones.

REGISTROS DE PROPÓSITO GENERAL

Registros de Propósito General

64 bits	32 bits	16 bits	8 bits	8 bits
RAX	EAX	AX	AH	AL
RBX	EBX	BX	BH	BL
RCX	ECX	CX	CH	CL
RDX	EDX	DX	DH	DL
RSI	ESI	SI	No disponible	
RDI	EDI	DI	No disponible	
RSP	ESP	SP	No disponible	
RBP	EBP	BP	No disponible	

REGISTROS DE PROPÓSITO GENERAL

- ✗ AX, AH, AL Acumulador a menudo conserva el resultado temporal después de una operación aritmética o lógica.
- ✗ BX, BH, BL Base Se utiliza para guardar la dirección base de listas de datos en la memoria.
- ✗ CX, CH, CL Contador Contiene el conteo para ciertas instrucciones de corrimientos y rotaciones, de iteraciones en el ciclo loop y operaciones repetidas de cadenas
- ✗ DX, DH, DL Datos Contiene la parte más significativa de un producto después de una multiplicación la parte más significativa del dividendo antes de la división

INTERRUPCIONES

Una interrupción es una situación especial que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción para tratarla. Tal situación se da, por ejemplo, cuando un periférico requiere la atención del procesador para realizar una operación de E/S.

EJEMPLOS DE INTERRUPCIONES

int 01h-->un solo paso

int 02h-->interrupcion no enmascarable

int 03h--> punto de interrupcion

int 04h-->desbordamiento

int 05h-->impresion de pantalla

int 08h-->Cronometro

int 15h-->Servicios del sistema

int 16h-->Funciones de entrada del teclado

int 18h-->Entrada con el Basic de Rom

int 19h-->Cargador ed arranque

int 1Ah-->Leer y establecer la hora

int 1Bh-->Obtener el control con una interrupcion de teclado.

int 20h-->Terminar un programa

int 33h->Funciones del Raton

ESTRUCTURA DE UN PROGRAMA EN ENSAMBLADOR MASM

Un programa ensamblador escrito con sintaxis MASM
está formado por:



[Definición de macros] [Includes]
[Declaración del programa]
[Segmento de pila]
[Segmento de datos]
[Segmento

de

código]

DECLARACIÓN DEL PROGRAMA

.MODEL [Modelo]

Los modelos pueden ser:

- ✗ Tiny: Los datos y el código juntos ocupan menos de 64 KB por lo que entran en el mismo segmento.
- ✗ Small: Los datos caben en un segmento de 64KB y el código cabe en otro segmento de 64KB.
- ✗ Medium: Los datos entran en un sólo segmento de 64KB, pero el código puede ser mayor de 64KB.
- ✗ Large: Tanto el código como los datos pueden ocupar más de 64KB.

SEGMENTO DE PILA:

.STACK[size]: Define el segmento de pila de la longitud especificada.

Si no se especifica, se pone un tamaño por defecto.

SEGMENTO DE DATOS

.DATA: define un segmento de datos NEAR con valores iniciales.

SEGMENTO DE CÓDIGO

.CODE: Define un segmento de código.

END

COMENTARIOS:

; comentario

PROCEDIMIENTOS

Secciones de código que se pueden llamar para su ejecución desde distintas partes del programa. Estos deben de ir dentro de un segmento de código.

identificador PROC {NEAR|FAR} (depende del modelo, en small por defecto es NEAR)

sentencias

identificador ENDP

Llamada: CALL identificador

MACROS

Al igual que los procedimientos, pueden ser llamados desde cualquier parte del segmento de código, pero a diferencia de ellos, estos pueden tener parámetros y se declaran antes de cualquier otra directiva, no en el segmento de código.

```
nombre_macro MACRO [parametro [,parámetro...]]  
[LOCAL nombre_local[,nombre_local]]  
sentencias  
ENDM
```

ETIQUETAS

Las etiquetas no son más que identificadores que nos permiten dividir o definir segmentos de código a los cuales queramos acceder desde determinado salto.

Identificador:

INSTRUCCIONES DE TRANSFERENCIA DE DATOS (NO AFECTAN FLAGS)

MOV destino, fuente -> Copia el contenido del operando fuente en el destino.

Operación: dest <- src

INSTRUCCIONES DE TRANSFERENCIA DE CONTROL (NO AFECTAN FLAGS)

SALTOS CONDICIONALES

JMP Etiqueta -> Saltar hacia la dirección de la etiqueta

CALL Procedimiento -> Ir al procedimiento cuyo inicio es label.

RET -> retorno de procedimiento

SALTOS CONDICIONALES

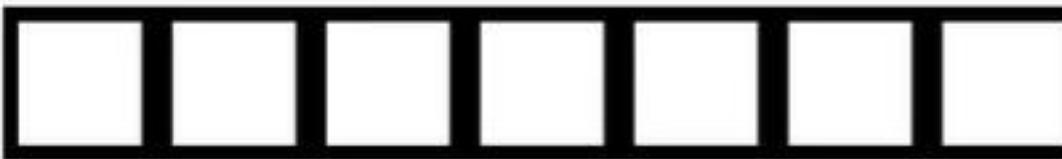
- JB etiqueta / JNAE etiqueta -> Saltar a etiqueta si es menor
- JBE etiqueta / JNA etiqueta -> Saltar a etiqueta si es menor o igual
- JE etiqueta -> Saltar a etiqueta si es igual
- JNE etiqueta -> Saltar a etiqueta si es distinto
- JAE etiqueta / JNB etiqueta -> Saltar a etiqueta si es mayor o igual
- JA etiqueta / JNBE etiqueta -> Saltar a etiqueta si es mayor

TAMAÑO DE LOS OPERANDO

En sintaxis NASM el tamaño de los datos especificados por un operando puede ser de byte, word, double word y quad word. Cabe recordar que lo hace en formato *little-endian*.

- BYTE: indica que el tamaño del operando es de un byte (8 bits).
- WORD: indica que el tamaño del operando es de una palabra (word) o dos bytes (16 bits).
- DWORD: indica que el tamaño del operando es de una doble palabra (double word) o cuatro bytes (32 bits).
- QWORD: indica que el tamaño del operando es de una cuádruple palabra (quad word) u ocho bytes (64 bits).

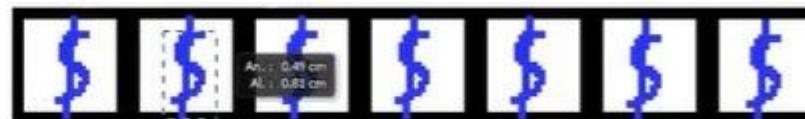
ARREGLO EN ASSEMBLER



Arreglo db 7 dup('\$'), '\$'

↑ ↑ ↑ ↑ ↑ ↑
Nombre Tipo Numero Llenas Finalizan
variable de caracter con con
 casillas caracter caracter

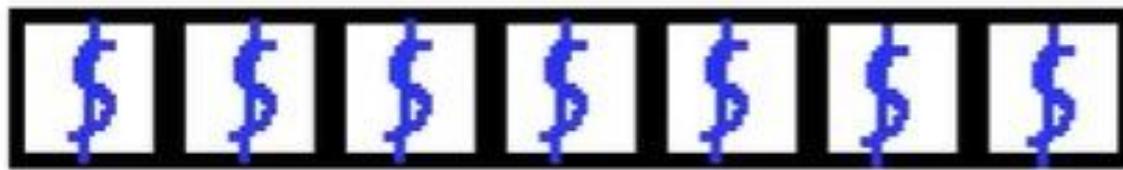
ARREGLO



Arreglo db 7 dup('\$'), '\$'

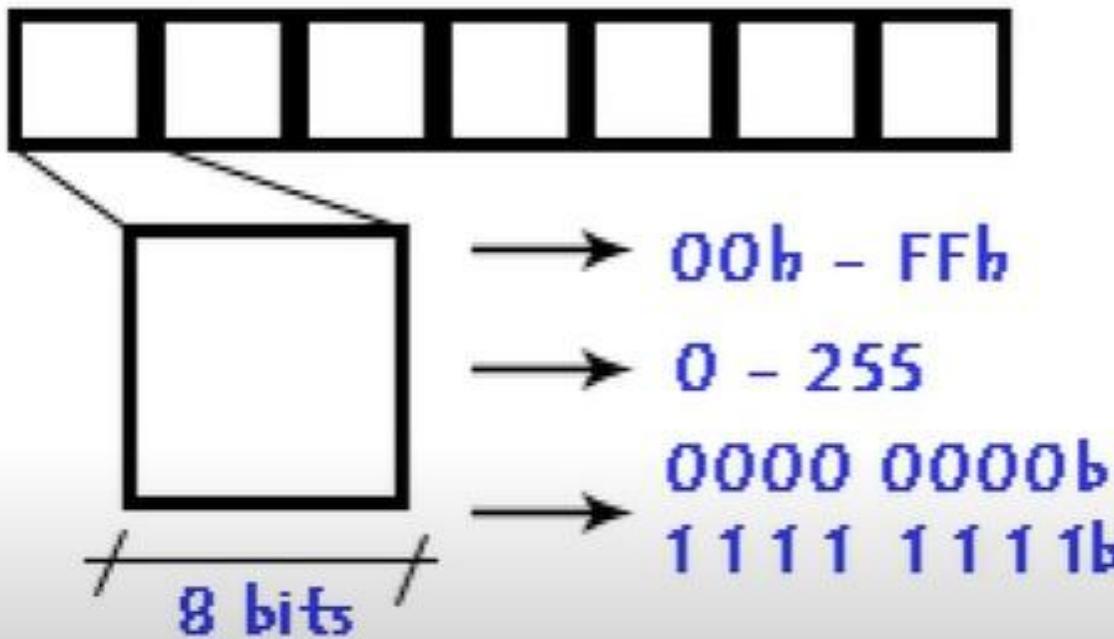
↑ ↑ ↑ ↑ ↑
Nombre Tipo Numero Llenas Finalizan
variable de con con con
 Casillas caracter caracter caracter

NOS DESPLAZAMOS CON :



Registros de Desplazamiento
`si, di, bp, bx`

CADA POSICIÓN DEL ARREGLO TIENE 8 BITS



Handler: El Handle es un número mediante el cual se referencia a un fichero dado.

HANDLER

Handler = # DE TIPO WORD

fichero: c:\entrada.arq



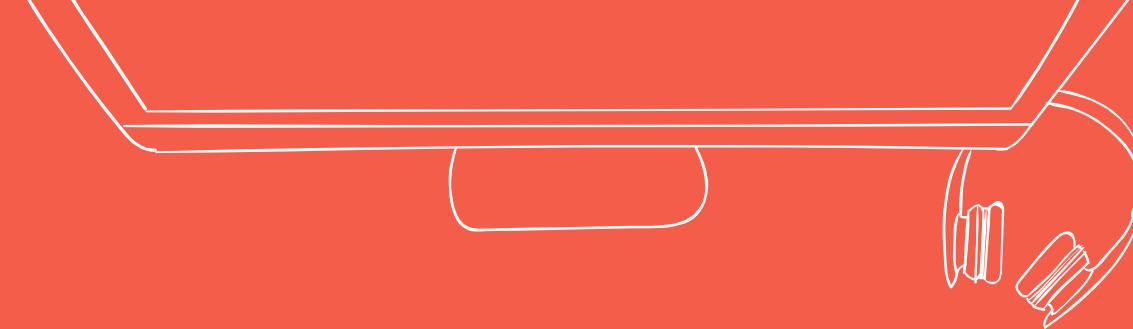
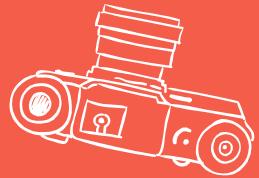
handler = 0001h

fichero: c:\entrada2.arq



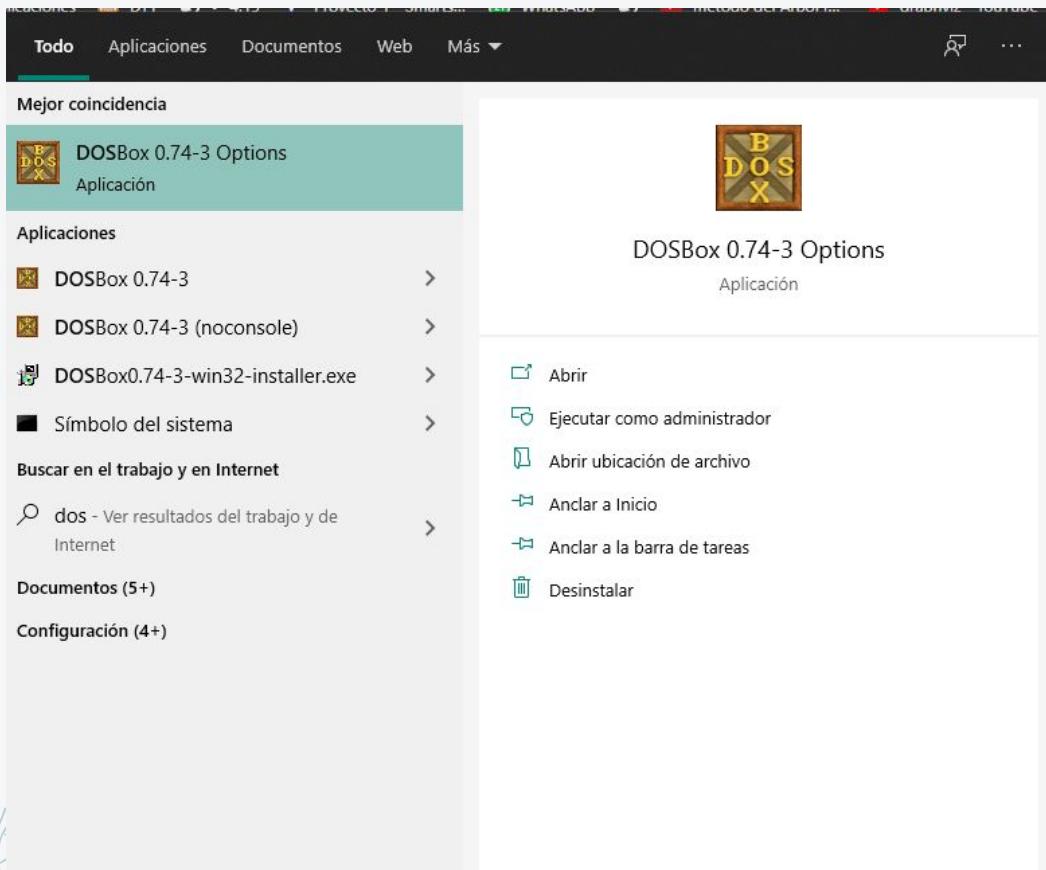
handler = 0002h

INSTALACIÓN DOSBOX Y MASM



https://www.youtube.com/watch?v=pIRd79UsHXA&t=7s&ab_channel=Christian

CONFIGURACIÓN OPCIONAL



dosbox-0.74-3.conf: Bloc de notas

Archivo Edición Formato Ver Ayuda

serial4=disabled

[dos]

```
#           xms: Enable XMS support.  
#           ems: Enable EMS support.  
#           umb: Enable UMB support.  
# keyboardlayout: Language code of the keyboard layout (or none).
```

xms=true

ems=true

umb=true

keyboardlayout=auto

[ipx]

ipx: Enable ipx over UDP/IP emulation.

ipx=false

[autoexec]

Lines in this section will be run at startup.

You can put your MOUNT lines here.

mount c c:\MASM

c:

cd masm611

cd bin

PROCEDIMIENTO PARA CREAR EL EJECUTABLE (EXE)

MASM



EN DOSBOX DEBEN SEGUIR LOS SIGUIENTES PASOS:

```
DOSBox Status Window
OSBox version 0.74-3
Copyright 2002-2019 DOSBox Team
-- 
ONFIG:Loading primary setting
IDI:Opened device:win32

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c c:\MASM           Montar la carpeta MASM
Drive C is mounted as local directory c:\MASM
Z:\>c:                         Entrar al directorio c:
C:\>cd masm611                  entrar a la carpeta masm611
C:\MASM611\cd bin                y a la carpeta bin, donde realizaremos los ejecutables
C:\MASM611\BIN>_
```

DOSBox Status Window

DOSBox version 0.74-3

Copyright 2002-2019 DOSBox Te

CONFIG:Loading primary settin

MIDI:Opened device:win32

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO

For supported shell commands type: HELP

To adjust the emulated CPU speed, use **ctrl-F11** and **ctrl-F12**.

To activate the keymapper **ctrl-F1**.

For more information read the **README** file in the DOSBox directory.

HAVE FUN!

The DOSBox Team <http://www.dosbox.com>

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c:\MASM

Drive C is mounted as local directory c:\MASM

Z:\>c:

C:\>cd masm611

C:\MASM611\BIN>cd bin

C:\MASM611\BIN>ml hola.asm

← Compilar el programa

DOSBox version 0.74-3

Copyright 2002-2019 DOSBox Te

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

-

□

X

CONFIG:Loading primary settin

Z:\>mount c c:\MASM
Drive C is mounted as local directory c:\MASM

MIDI:Opened device:win32

DOSBox switched to max cycles

in DOSBox's options.

Z:\>c:

les amount

C:\>cd masm611

C:\MASM611>cd bin

C:\MASM611\BIN>ml hola.asm

Microsoft (R) Macro Assembler Version 6.11

Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: hola.asm

Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992

Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

Object Modules [.obj]: hola.obj

Run File [hola.exe]: "hola.exe" ← Crea el ejecutable

List File [nul.map]: NUL

Libraries [.lib]:

Definitions File [nul.def]:

C:\MASM611\BIN>_



DOSBox Status Window

DOSBox version 0.74-3

Copyright 2002-2019 DOSBox Te

CONFIG:Loading primary settin
MIDI:Opened device:win32
DOSBox switched to max cycles
in DOSBox's options.

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

Z:\>c:

C:\>cd masm611

C:\MASM611>cd bin

C:\MASM611\BIN>ml hola.asm

Microsoft (R) Macro Assembler Version 6.11

Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: hola.asm

Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992

Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

Object Modules [.obj]: hola.obj

Run File [hola.exe]: "hola.exe"

List File [nul.map]: NUL

Libraries [.lib]:

Definitions File [nul.def]:

C:\MASM611\BIN>hola ← Ejecutamos

Hola mundo!!!

C:\MASM611\BIN>_

LINKS APOYO

i8086 and DOS interrupts.pdf

INT 21H

Código ASCII

x86 and amd64 instruction reference

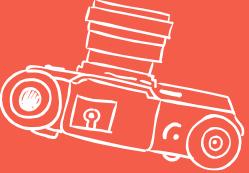
INT 21H

Tabla de interrupciones

HOJA DE TRABAJO # 3

1. Investigar como se realizan las operaciones aritmeticas (+,-,*,/) en ensamblador.

Deben entregar un documento pdf con nombre [ACEY1]HT3_#Carnet, entregar en uedi antes del 19 de marzo a las 23:59

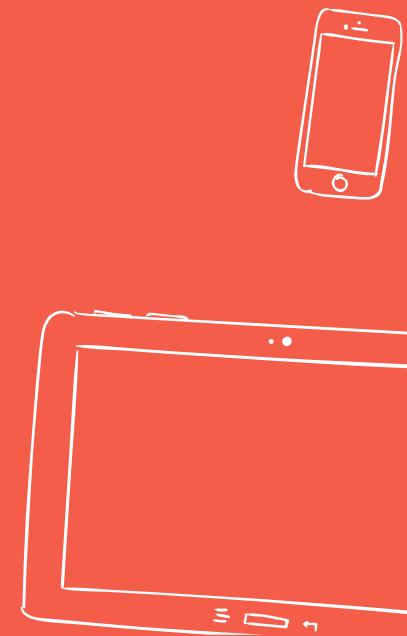


CLASE 8



ARQUII SECCION B

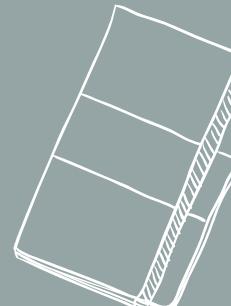
ASSEMBLER





AGENDA 19/03

1. Dudas Práctica 3
2. Tema Clase 8 (Archivos y operaciones)
3. TAREA 5
4. CORTO 2 (20/03) 9:00 PM
5. FOTOS DTT
6. RECORDATORIO: **Hoy es ÚLTIMO DÍA DE ASIGNACIONES en el portal de ingeniería**



SUMA

La operación de suma o adicción (ADD) permite realizar la suma de dos números enteros a nivel de bits 8, los datos a operar pueden estar almacenados en registros de 8, 16, 32 o 64 bits.

ADD destino, fuente

En alto nivel: destino = destino + fuente

RESTA

La operación de sustracción (SUB) permite realizar la resta de dos números enteros a nivel de bits 8, los datos a operar pueden estar almacenados en registros de 8, 16, 32 o 64 bits.

SUB destino, fuente

En alto nivel: destino = destino - fuente

MULTIPLICACIÓN

La instrucción MUL (multiplicación) permite realizar la operación de multiplicación entre un registro implícito y un operando fuente, siendo estos dos datos enteros sin signo. En tanto la instrucción IMUL se encarga de los datos que presentan signo. Los registros A y D participan de forma implícita en la operación, ambos con el mismo tamaño que el registro fuente. (A siempre se utiliza, D se puede cambiar por otro registro)

MUL fuente

IMULT fuente

Registro fuente de 8 bits
 $AL \times \text{fuente} = AH : AL$

Registro fuente de 16 bits
 $AX \times \text{fuente} = DX : AX$

Registro fuente de 32 bits
 $EAX \times \text{fuente} = EDX : EAX$

Registro fuente de 64 bits
 $RAX \times \text{fuente} = RDX : RAX$

```
mov al,numero          ;copiamos a al el primer numero  
mov bl, numero2        ;copiamos a bl el segundo numero  
mul bl                ;multiplica al por el destino(bl)  
mov resultado, al       ;el resultado lo guarda en al
```

DIVISIÓN

La instrucción DIV (división) realiza la operación de división entre un registro implícito y un operando fuente, siendo ambos datos números enteros sin signo. Mientras que IDIV es utilizado para dividir dos números enteros con signo.

DIV fuente

IDIV fuente

Implícitamente intervienen en la división con y sin signo los registros A y D, los cuales manejaran en la operación el mismo número de bits que el registro fuente.

Registro fuente de 8 bits

AH : AL
fuente 8 bits

Resultado -> AL
Residuo -> AH

Registro fuente de 16 bits

DX : AX
fuente 16 bits

Resultado -> AX
Residuo -> DX



DIVISIÓN

```
mov al,numero  
mov bl, numero2  
div bl  
mov resultado, al
```

```
;copiamos a al el primer numero  
;copiamos a bl el segundo numero  
;divide al / bl(fuente)  
;el resultado se guarda en al
```

TAREA # 5

1. Realizar un programa que pueda sumar y restar números de -99 a +99 , en modo calculadora

Deben entregar un archivo .exe y el archivo.asm dentro de una carpeta con nombre [ACEY1]T5_#Carnet, entregar en uedi antes del 25 de marzo a las 23:59

LINKS APOYO



[manual_basico_ensamblador](#)
[Suma y Resta en ensamblador](#)
[Multiplicación en ensamblador](#)
[División en ensamblador](#)
[INT 21H](#)