PRÁCTICA 0a

ESTRUCTURA INTERNA DE UN PROCESADOR CON MODELO VON NEUMANN

ARQUITECTURA DE COMPUTADORES. 2º CURSO

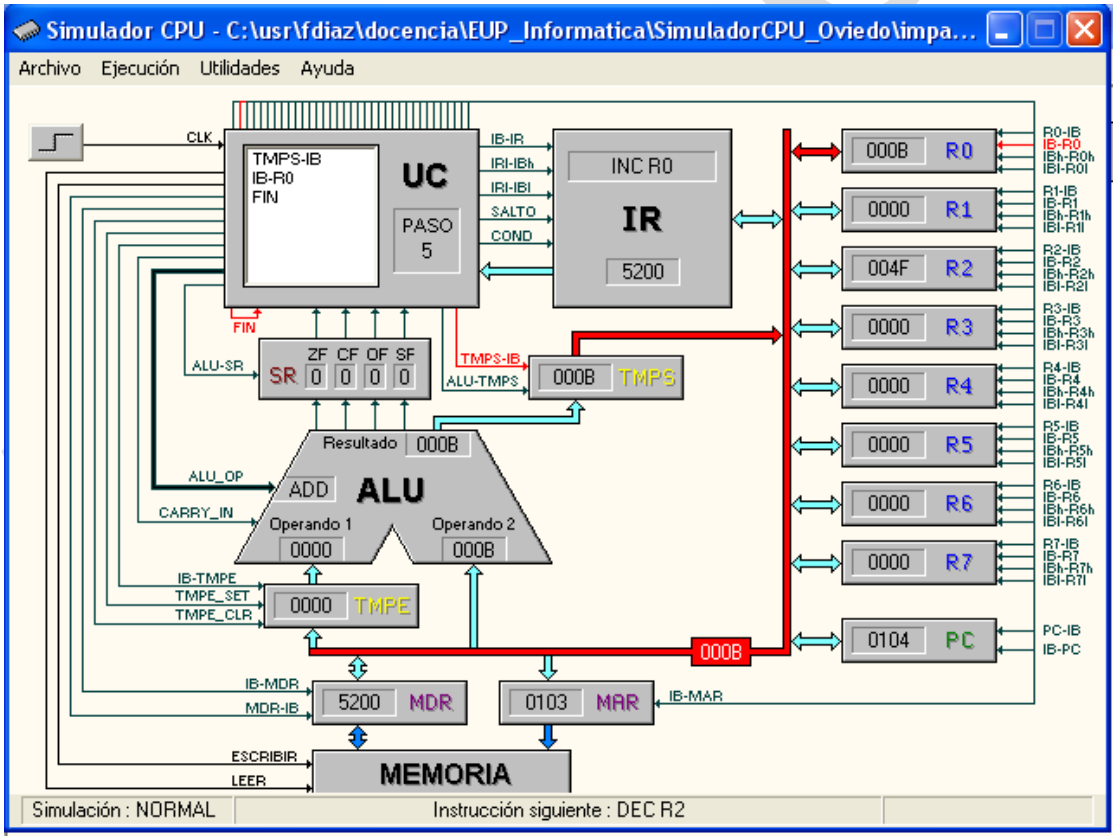
OBJETIVOS:

En práctica sirve como repaso y complemento práctico de lo estudiado en "Estructura de Computadores" de 1º de grado. Se pretende que el alumno repase la estructura y funcionamiento de un procesador sencillo pero basado en el modelo o arquitectura *Von Neumann* (no confundir con el modelo o arquitectura *Harvard*). El modelo *Von Neumann* (memoria de datos e instrucciones común) es el modelo dominante en todos los procesadores comerciales de hoy en día (x86, ARM, MIPS, etc). Su conocimiento y dominio es esencial para luego entender aspectos avanzados de los procesadores tales como la segmentación, la jerarquía de memoria y el paralelismo a nivel de procesadores en general.

El alumno deberá entender la estructura básica de este procesador o CPU, conocer como se ejecutan las instrucciones de lenguaje o código máquina de este simulador, y extraer las características principales del procesador estudiado.

PREPARACIÓN:

En el aula de ordenadores se ejecutarán un par de sencillos programas (en código máquina) en un simulador de una CPU simple desarrollado en la Universidad de Oviedo. La CPU se compone de una ALU, una Unidad de Control (UC), y los registros habituales (PC, IR, MAR, MDR, etc.), como se ve en la siguiente figura.



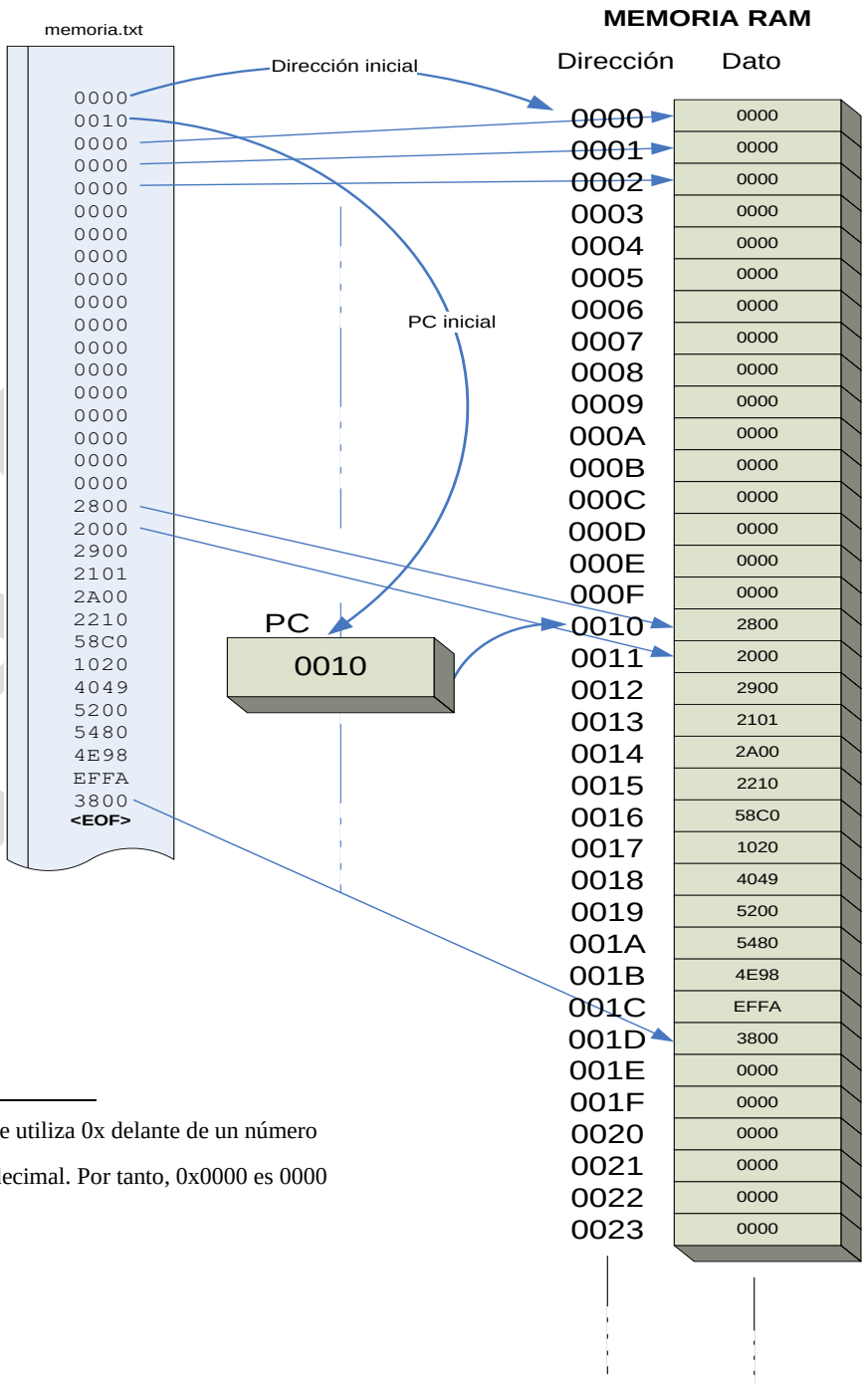
En tal figura superior se observan los 8 registros de la CPU (con el valor que contienen en **hexadecimal**), los buses (todos tienen un nombre, pero no son útiles para esta práctica). Entre los buses, destacar el *gran bus* (contiene el valor 000B en la figura) que conecta los registros con la ALU, el IR y los MAR, MDR. Además hay un registro especial encima de la ALU: es el "registro de estado" (SR, State Register), que indica que ha ocurrido tras hacer una operación en la ALU. Concretamente tiene 4 bits, que señalan:

- si la operación dio como resultado cero (se pondría a 1 el bit ZF, Zero Flag).
- si la operación generó un acarreo (se pondría a 1 el bit CF, Carry Flag).

- si se produjo desbordamiento (se pondría a 1 el bit OF, Overflow Flag).
- si el signo del resultado es negativo (se pondría a 1 el bit SF, Sign Flag).

Los programas que se quieren ejecutar (en simulación) deben estar en ficheros de texto (pero con la extensión .prg). Tales ficheros deben contener una línea por cada instrucción o dato que se almacenará en la memoria (del ordenador simulado). Además, existe una o dos primeras líneas con dos números (**hexadecimales**), que son dos direcciones: la primera es donde se almacenará el código en memoria, y la segunda, el valor inicial del PC (*Program Counter* o Contador de Programa), o sea, será la dirección de donde se empiecen a ejecutar las instrucciones. De esta manera, el espacio que existe entre la dirección de carga del programa en memoria y la dirección de la primera instrucción e ejecutar (valor del PC) se utiliza como espacio para guardar el equivalente a las variables de un lenguaje de alto nivel.

En la siguiente figura se muestra el contenido de un fichero con un programa de ejemplo. Este fichero se llama "memoria.txt" aunque para que el simulador lo pudiese abrir debería llamarse "memoria.prg". Se observa cómo la dirección inicial de carga es la 0x0000[1] y la dirección de la primera instrucción es la 0x0010. Luego tras cargar tal fichero para simular su ejecución la memoria tendrá el contenido dado por la parte derecha de tal figura. El espacio de memoria que hay entre el inicio del programa 0x100 y el valor del PC (comienzo de la primera instrucción) se suele utilizar para almacenar datos del programa. Estos datos se corresponderían con variables o constantes de un lenguaje de alto nivel como C.



[1] Se sigue la notación en C que utiliza 0x delante de un número para indicar que está en hexadecimal. Por tanto, 0x0000 es 0000 en hexadecimal.

Evidentemente, el programa está en código máquina, y no podemos saber que hará si no sabemos el formato del juego de instrucciones. Afortunadamente, el simulador de CPU puede "desensamblar" el código máquina, esto es, traducirlo a lenguaje ensamblador, más fácil de entender por los humanos. Una especificación completa del juego de instrucciones está disponible en el anexo de este boletín.

En el siguiente ejemplo se muestra un programa tal y como se cargaría en un fichero de texto "pot.prg" y el mapa de memoria. En la última columna de la derecha se pone la traducción a lenguaje ensamblador.

Fichero pot.prg
0100
0106
0000
0000
0000
0000
0000
0000
5800
2801
5840
2101
1020
5200
4049
1020
5200
4049
1020

Direcciones de Memoria	Datos (o instrucciones)	Traducción de las instrucciones a lenguaje ensamblador
0x0100	0000	
0x0101	0000	
0x0102	0000	
0x0103	0000	
0x0104	0000	
0x0005	0000	
0x0106 (PC)	5800	CLR R0
0x107	2801	MOVH R0, 1
0x108	5840	CLR R1
0x109	2101	MOVL R1, 1
0x10A	1020	MOV [R0], R1
0x10B	5200	INC R0
0x10C	4049	ADD R1,R1,R1
0x10D	1020	MOV [R0], R1
0x10E	5200	INC R0
0x10F	4049	ADD R1,R1,R1
0x110	1020	MOV [R0], R1

Algunas aclaraciones sobre el juego de instrucciones de esta CPU y el ejemplo anterior:

- El registro (u operando en general) destino es el de la izquierda, y los de la derecha son los fuentes.
- CLR significa borrar, es decir, poner a cero (del inglés CLEAR).
- INC significa incrementar (en una unidad).
- ADD viene del inglés SUMAR.
- MOV es mover, y sirve para escribir, leer de memoria (aquí se usa el símbolo corchete [ ] para indicar acceso a memoria).
- Hay dos variantes de MOV: MOVH (del inglés HIGH) carga un número de 8 bits en los 8 bits más significativos de un registro, y MOVL (del inglés LOW) carga un número de 8 bits en los 8 bits menos significativos de un registro. Por ejemplo, las dos instrucciones CLR R1 y MOVL R1,1, consiguen primero poner a cero el registro R1 y luego ponerlo a 1.
- La **unidad direccionable de la memoria no es el byte**, sino que coincide con el tamaño de una instrucción. Lo normal es que la unidad mínima direccionable de la memoria en un procesador sea el *byte*. Por simplicidad en este simulador coincide con el tamaño de las instrucciones.
- Cada vez que se carga un programa o se reinicia (resetea) el ordenador, toda la memoria se pone a 0 (0x0000). La instrucción cuyo código máquina es el 0x0000 es la MOV R0, R0, es decir una instrucción que no hace nada útil, ni provoca ningún cambio en el registro R0.

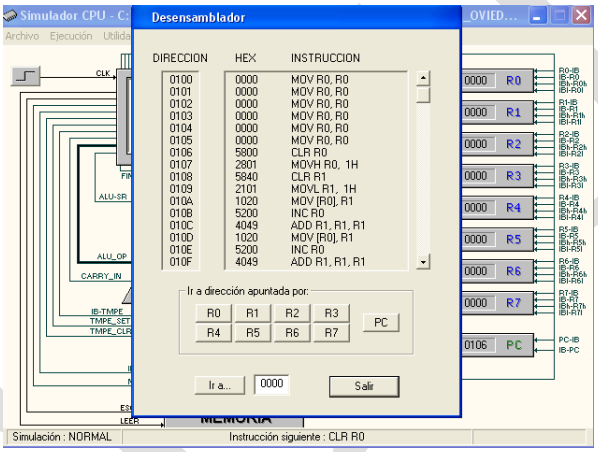
Puede descargar el software de este simulador de CPU desde la plataforma de enseñanza virtual.
Descomprima el archivo y haga doble click sobre el ejecutable. Para ejecutar y leer en Windows 64 bits el fichero de ayuda puede que necesite descargar e instalar algún programa auxiliar extra.

PASOS A SEGUIR PARA EJECUTAR LOS PROGRAMAS:

Para simular que se está ejecutando un programa hay que hacer:

1. Ejecutar el programa simulador (que se llama *smldrcpu.exe*).
2. Cargar en el simulador el programa en código máquina. Para ello usar Archivo->Abrir, y luego seleccionar en "Mostrar archivos de tipo" los *.PRG. El programa de ejemplo a cargar se llama *pot.prg*.
3. A continuación, comprobar que la carga del programa se hizo correctamente. Para ello el simulador dispone de una opción para ver el contenido de la memoria del ordenador simulado. Se aconseja usar Utilidades->Memoria->Desensamblador. Se verá una pantalla como la de la siguiente figura. Obsérvese como está cargado el programa y los datos del fichero *pot.prg* (el de la preparación de esta práctica). Pulsando sobre un registro se mostrará la zona de memoria "apuntada" por tal registro, es decir, las direcciones del valor del registro. También se puede especificar una dirección numérica en la caja "*Ir a...*".

   Puede ver el código de su programa si hace clic en "PC", que le direccionará a la posición de inicio de su programa en memoria.



4. Tras salir del desensamblador, se puede simular la ejecución del programa. Esto se puede hacer "instrucción a instrucción" (pulsando F8), o ciclo a ciclo (periodo) de reloj (pulsando F7, o haciendo clic sobre el símbolo que representa un flanco de subida de una señal cuadrada ⎍, que está en la esquina superior izquierda, bajo "Archivo"). Si se ejecuta ciclo a ciclo se observará como una instrucción tarda varios ciclos en ejecutarse (lo que se corresponde con las fases IF, ID, EX de una instrucción). Se empezará ejecutando por la dirección que indica el PC y se seguirán ejecutando instrucciones, mientras el PC se va incrementando.
5. También puede abrirse la ayuda del simulador, que es otro fichero llamado *usuario.hlp*. En la ayuda se explica también como simular un programa, y también se describe el juego de instrucciones que tiene el procesador simulado y que aparece en el anexo. Para abrir el fichero haga doble *clic* sobre él (puede que en Windows 64 bits necesite descargar un programa extra para abrir este fichero).

ALGUNAS ACLARACIONES DEL SIMULADOR:

- Para incrementar el PC, le suma cero, pero añadiéndole un acarreo (*carry*, señal CARRY_IN activada) de 1, en definitiva le suma 1.
- Lo que se muestra en el simulador en cada ciclo de reloj, ya se ha ejecutado.

ANEXO: Juego de Instrucciones

Nemotécnico	*Formato de la instrucción (binario)*	*Descripción*	*Excepciones*	*Ejemplo de uso*
Instrucciones de movimiento (código de clase de instrucción 00) Todas estas instrucciones están encaminadas a dar valores a los registros, bien leyendo el valor desde memoria, desde otro registro, o desde la propia instrucción. También hay instrucciones que escriben datos de un registro a memoria. Un caso especial en esta clase de instrucción es la instrucción STOP, ya que no es una instrucción de movimiento propiamente dicha.				
MOV Rd,Rs	00 000 Rd Rs 00000	Copia el valor que esté almacenado en el registro Rs, al registro Rd.	No tiene.	MOV R3,R5 Binario: 00 000 011 101 00000b Hexadecimal: 03A0h Antes de esta instrucción: R3 = 1C40h R5 = 9B23h. Después de esta instrucción: R3 = 9B23h
MOV Rd,[Ri]	00 001 Rd Ri 00000	Lee un valor de 16 bits de la memoria, en la dirección indicada por el valor del registro Ri, y lo guarda en el registro Rd.	Excepción de memoria, si la dirección desde la que se quiere leer está fuera de rango. En tal caso, el registro Rd no se altera.	MOV R7,[R1] Binario: 00 001 111 001 00000b Hexadecimal: 0F20h Antes de esta instrucción: R1 = 0300h R7 = 1234h La posición 0300h de memoria contiene el valor 8D01h. Después de esta instrucción: R7 = 8D01h
MOV [Ri],Rs	00 010 Ri Rs 00000	Escribe el valor de 16 bits que está en el registro Rs, a la dirección de memoria indicada por el valor del registro Ri	Excepción de memoria, si la dirección a la que se pretende escribir está fuera de rango.	MOV [R4],R0 Binario: 00 010 100 000 00000b Hexadecimal: 1400h Antes de esta instrucción: R0 = FF12h R4 = 12ABh La posición 12ABh de memoria contiene el valor 2855h. Después de esta instrucción: La posición 12ABh de memoria contiene el valor FF12h
MOVL Rd,inm8	00 100 Rd inm8	Guarda en el byte menos significativo del registro Rd el valor inm8.	No tiene.	MOVL R5,5Ah Binario: 00 100 101 01011010b Hexadecimal: 255Ah Antes de esta instrucción: R5 = 1812h Después de esta instrucción: R5 = 185Ah
MOVH Rd,inm8	00 101 Rd inm8	Guarda en el byte más significativo del registro Rd el valor inm8	No tiene.	MOVH R0,EFh Binario: 00 101 000 11101111b Hexadecimal: 28EFh Antes de esta instrucción: R0 = 6B03h Después de esta instrucción: R0 = EF03h
Instrucciones lógico-aritméticas (código de clase de instrucción 01). Todas estas instrucciones actualizan las banderas de condición después de realizar la operación. - Si el resultado de la operación es 0, ZF vale 1, en otro caso, 0. - El valor de SF será el valor del bit más significativo del resultado. - Si al realizar la operación hubo desbordamiento (el resultado no cabe en un entero de 16 bits en complemento a 2) entonces OF vale 1, en otro caso, vale 0. - Si al realizar la operación hubo acarreo (el resultado no cabe en un entero de 16 bits en formato de binario natural) entonces CF vale 1, en otro caso, vale 0.				
ADD Rd,Rs1,Rs2	01 00000 Rd Rs1 Rs2	Suma el contenido del registro Rs1 con el de Rs2, y el resultado lo guarda en el registro Rd.	No tiene.	ADD R2,R6,R7 Binario: 01 00000 010 110 111b Hexadecimal: 40B7h Antes de esta instrucción: R2 = 883Eh R6 = 12FFh R7 = A003h Después de esta instrucción: R2 = B302h ZF=0, SF=1, CF=0, OF=0
SUB Rd,Rs1,Rs2	01 00001 Rd Rs1 Rs2	Resta el contenido del registro Rs1 menos el de Rs2, y el resultado lo guarda en el registro Rd.	No tiene.	SUB R0,R1,R4 Binario: 01 00001 000 001 100b Hexadecimal: 420Ch Antes de esta instrucción: R0 = 0A1Ch, R1 = 5982h, R4 =9BB0h Después de esta instrucción: R0 = BDD2h ZF=0, SF=1, CF=0, OF=0
OR Rd,Rs1,Rs2	01 00010 Rd Rs1 Rs2	Realiza la operación OR (suma lógica, en C es el operador |) entre el contenido del registro Rs1 y Rs2. El resultado se guarda en el registro Rd.	No tiene.	OR R2,R3,R4 Binario: 01 00010 010 011 100b Hexadecimal: 449Ch Antes de esta instrucción: R2 = 9B10h R3=E004h R4=38C8h Después de esta instrucción: R2=F8CCh ZF=0, SF=1, CF=0, OF=0

Nemotécnico	*Formato de la instrucción (binario)*	*Descripción*	*Excepciones*	*Ejemplo de uso*
AND Rd,Rs1,Rs2	01 00011 Rd Rs1 Rs2	Realiza la operación AND (producto lógico, en C es el operador &) entre el contenido del registro Rs1 y Rs2. El resultado se guarda en el registro Rd.	No tiene.	AND R2,R3,R4 Binario: 01 00011 010 011 100b Hexadecimal: 469Ch Antes de esta instrucción: R2 = 9B10h R3=E004h R4=38C8h Después de esta instrucción: R2=2000h ZF=0, SF=0, CF=0, OF=0
XOR Rd,Rs1,Rs2	01 00100 Rd Rs1 Rs2	Realiza la operación XOR (o exclusivo lógico, en C es el operador ^) entre el contenido del registro Rs1 y Rs2. El resultado se guarda en el registro Rd.	No tiene.	XOR R2,R3,R4 Binario: 01 00100 010 011 100b Hexadecimal: 489Ch Antes de esta instrucción: R2 = 9B10h R3=E004h R4=38C8h Después de esta instrucción: R2=D8CCh ZF=0, SF=1, CF=0, OF=0
COMP Rs1,Rs2	01 00111 Rs1 Rs2 000	Compara el contenido del registro Rs1 con el del registro Rs2. En realidad lo que hace es restar Rs1 menos Rs2, pero no guarda el resultado en ningún registro de destino, solamente actualiza las banderas de condición.	No tiene.	COMP R2,R7 Binario: 01 00111 010 111 000b Hexadecimal: 4EB8h Antes de esta instrucción: R2 = AA15h R7 = 79F0h Después de esta instrucción: ZF=0, SF=0, CF=1, OF=1
NOT Rds	01 01000 Rds 000000	Realiza la operación NOT (complemento a 1) del valor del registro Rds, y lo almacena en el mismo registro.	No tiene.	NOT R4 Binario: 01 01000 0100 000000b Hexadecimal: 5080h Antes de esta instrucción: R4 = 1234h Después de esta instrucción: R4 = EDCBh ZF=0, SF=1, CF=0, OF=0
INC Rds	01 01001 Rds 000000	Incrementa el valor del registro Rds	No tiene	INC R1 Binario: 01 01001 001 000000b Hexadecimal: 5240h Antes de esta instrucción: R1 = 90BFh Después de esta instrucción: R1 = 90C0h ZF=0, SF=1, CF=0, OF=0
DEC Rds	01 01010 Rds 000000	Decrementa el valor del registro Rds	No tiene	DEC R7 Binario: 01 01010 111 000000b Hexadecimal: 55C0h Antes de esta instrucción: R7 = 4098h Después de esta instrucción: R7 = 4097h ZF=0, SF=0, CF=0, OF=0
NEG Rds	01 01011 Rds 000000	Realiza el complemento a 2 del valor del registro Rds	No tiene	NEG R6 Binario: 01 01011 110 000000b Hexadecimal: 5780h Antes de esta instrucción: R6 = 80B2h (-32590 en decimal) Después de esta instrucción: R6 = 7F4Eh (32590 en decimal) ZF=0, SF=0, CF=0, OF=0
CLR Rds	01 01100 Rds 000000	Pone a 0 el registro Rds	No tiene	CLR R3 Binario: 01 01100 011 000000b Hexadecimal: 58C0h Antes de esta instrucción: R3 = F12Fh Después de esta instrucción: R3 = 0000h ZF=1, SF=0, CF=0, OF=0
Instrucciones de salto incondicional (código de clase de instrucción 10)				
JMP desplaz	10 desplaz	El valor de 14 bits *desplaz* es extendido con signo a 16 bits, y se le suma al valor actual del registro PC, obteniendo la dirección de la siguiente instrucción a ejecutar.	Excepción de memoria si la dirección destino de salto está fuera del rango de la memoria disponible.	JMP -23 Binario: 10 11111111101001b Hexadecimal: BFE9h Antes de esta instrucción: PC = 0120h Después de esta instrucción: PC = 0109h

Nemotécnico	*Formato de la instrucción (binario)*	*Descripción*	*Excepciones*	*Ejemplo de uso*
Instrucciones de salto condicional (código de clase de instrucción 11)				
BR cond desplaz	11 cond desplaz	*cond* codifica la condición que ha de cumplirse: 000 C, Salto si CF=1 001 NC, Salto si CF=0 010 O, Salto si OF=1 011 NO, Salto si OF=0 100 Z, Salto si ZF=1 101 NZ, Salto si ZF=0 110 S, Salto si SF=1 111 NS, Salto si SF=0 Si la condición se cumple, el valor de 11 bits *desplaz* se extiende con signo a 16 bits, y se le suma al valor actual del registro PC, obteniendo la dirección de la siguiente instrucción a ejecutar. Si la condición no se cumple, no se modifica PC.	Excepción de memoria si la dirección destino de salto está fuera del rango de la memoria disponible.	BR S,+42 Binario: 11 110 00001000010b Hexadecimal: F042h Antes de esta instrucción: PC = 3200h SF=0 Después de esta instrucción: PC = 3200h (no se modifica PC, al no cumplirse la condición) BR NZ,+42 Binario: 11 101 00001000010b Hexadecimal: E842h Antes de esta instrucción: PC = A004h ZF=0 Después de esta instrucción: PC = A02Eh (se modifica PC, al cumplirse la condición)

EJERCICIOS PARA PRACTICAS CON EL SIMULADOR

Estos ejercicios son orientativos: en realidad, con cargar el simulador y probar paso a paso los programas de ejemplo (comprendiéndolos) sería más que suficiente (ejercicio 0). El resto de los ejercicios se corresponden a los solicitados en el laboratorio cuando esta sesión práctica era presencial.

Ejercicio 0: Cargar los diversos programas y ejecutarlos paso a paso comprendiendo lo que sucede en cada ciclo de reloj

Ejercicio1: Rellene la siguiente tabla. Ayúdese del software/programa del simulador

PROGRAMA "POT.PRG"

	Explique que hace en cada ciclo (periodo) la instrucción INC R0	**Explique que hace en cada ciclo (periodo) la instrucción MOV [R0], R1.**
Ciclo 1		
Ciclo 2		
Ciclo 3		
Ciclo 4		
Ciclo 5		
Ciclo 6		
Ciclo 7		

Ejercicio2: rellene cuántos ciclos de reloj duraría cada fase (o etapa) de ejecución de las dos instrucciones anteriores, si se definiera que la instrucción tiene 3 fases: IF, ID, EX

	instrucción INC R0	instrucción MOV [R0], R1.
IF		
ID		
EX		

Ejercicio3: escriba un programa en código ensamblador, ejecútelo y compruebe su funcionamiento viendo el estado de la memoria al final de su ejecución (menú "Utilidades" + "Memoria"+"Editor hexadecimal"). Para ello ayúdese del programa "ensamblador.exe" (y su manual de instrucciones) que traduce un programa escrito en lenguaje ensamblador contenido en un fichero de texto, al código máquina del simulador de la CPU. Se obtiene un fichero "memoria.txt" que habrá de renombrar a "memoria.prg" para probarlo en el simulador. El programa a realizar se cotejará en función de la primera letra del primer apellido del alumno/a:

1. Alumnos/as con primer apellido que comience entre las letras A-F.

   Realice un programa en ensamblador que calcule los 16 primeros números de la serie de Fibonacci y los escriba a partir de la posición cero de memoria.
2. Alumnos/as con primer apellido que comience entre las letras G-L

   Realice un programa en ensamblador que escriba los 20 primeros múltiplos de 3 y los escriba a partir de la posición cero de memoria.
3. Alumnos/as con primer apellido que comience entre las letras G-P.

   Realice un programa en ensamblador que escriba a partir de la posición 1 de memoria la tabla de multiplicar del número que se haya escrito previamente en la posición 0 de memoria. Ese número se habrá definido como una constante previamente en su programa en ensamblador y guardado a continuación en esa posición 0 de memoria.
4. Alumnos/as con primer apellido que comience entre las letras Q-Z.

   Realice un programa en ensamblador que escriba a partir de la posición 0 de memoria los números del 1 al 25. A continuación, y a partir de la posición 26 de memoria, la misma serie de números del 1 al 25 pero haciéndole a cada número una operación XOR con 0xAAAA.