GRADO EN INGENIERÍA INFORMÁTICA

ESTRUCTURA DE COMPUTADORES

Práctica de microprogramación

Documentación del simulador P8080E

Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Facultad de Informática - UPM

Curso 2009/2010

8/10/2009

P8080E: MICROPROCESADOR MICROPROGRAMABLE

1 INTRODUCCIÓN.

La práctica consiste, como su nombre indica, en microprogramar un microprocesador, y esperamos que nos ayude a distinguir entre los conceptos, radicalmente diferentes, de programar un microprocesador y microprogramar un procesador. La coincidencia de términos lleva frecuentemente a hablar de microprogramación de forma ambigua.

La elección del microprocesador 8080 de Intel es arbitraria. Se eligió por su simplicidad, por la abundancia de información y porque, de alguna forma, pese a estar ya anticuado, ha pasado a la historia de la informática. Conviene destacar, no obstante, que la práctica no pretende hacer un 8080, ni en ningún modo repetir la labor del personal de diseño de Intel. El 8080 es una orientación. Aquí se describe un pseudo 8080 (escribiremos P8080E) que se inspira en aquél.

Las razones que nos han llevado a crear un pseudo 8080 son fundamentalmente de dos tipos:

1.- El microprocesador 8080 no se pensó para un diseño microprogramado y, por tanto, no se adapta naturalmente a nuestro objetivo. Su diseño fue hecho a base de puertas lógicas de forma más convencional.

2.- Es nuestro interés presentar al alumno un diseño, según las explicaciones que se dan en las clases teóricas, prefiriendo sacrificar algún detalle real ante un concepto lógico.

Concretamente, P8080E difiere del 8080 en los siguientes puntos:

- No se respetan los diagramas de tiempo, variando ligeramente la correlación de señales.
- No se respeta el número de ciclos de reloj (estados T) que lleva cada instrucción.
- Se han olvidado completamente todas las señales relacionadas con el control de los buses, tales como HOLD y HLDA.
- Tampoco se considera el manejo de interrupciones, señales INT, INTE e INTA.
- Señales como DBIN, SYNC, WAIT y WR se han considerado innecesarias.
- El pin RESET queda enmascarado. P8080E ejecuta un RESET único al arrancar (power-up) y nunca más.
- Los pines de alimentación (GND, +5v, -5v y +12v) y las fases de reloj (ϕ1 y ϕ2) se dan por sobreentendidas.
- Se ha modificado la organización interna respecto de los diagramas que se incluyen en los manuales estándar del 8080.
- Se ha añadido un biestable de desbordamiento (overflow) a la ALU y un biestable N para implementar dos niveles de ejecución: nivel de usuario y nivel privilegiado.
- No se genera la información de STATUS típica de cada ciclo de máquina M del 8080.
- P8080E genera directamente las señales MEMR, MEMW, IOR e IOW que, en el caso del 8080, exigen emplear un circuito tipo 8228.

Todas estas divergencias deseamos aprovecharlas para una arquitectura más educativa. Creemos que sólo le sorprenderán a quien esté acostumbrado a trabajar con el microprocesador 8080, construyendo circuitos con él.

Debe quedar claro que lo que sigue es la descripción del P8080E. En ningún momento pretendemos hacer un manual del 8080. En la Biblioteca de la Facultad pueden encontrarse numerosos textos sobre microprocesadores que se extienden en la descripción del 8080 de Intel. A ellos nos remitimos para un mejor conocimiento de éste y para completar la información sobre P8080E.

2 SISTEMA MICROCOMPUTADOR.

El sistema completo simulado (fig. 1) incluye, además del P8080E, una memoria de 32kx8 (32768 palabras de 8 bits, bytes) y un periférico.

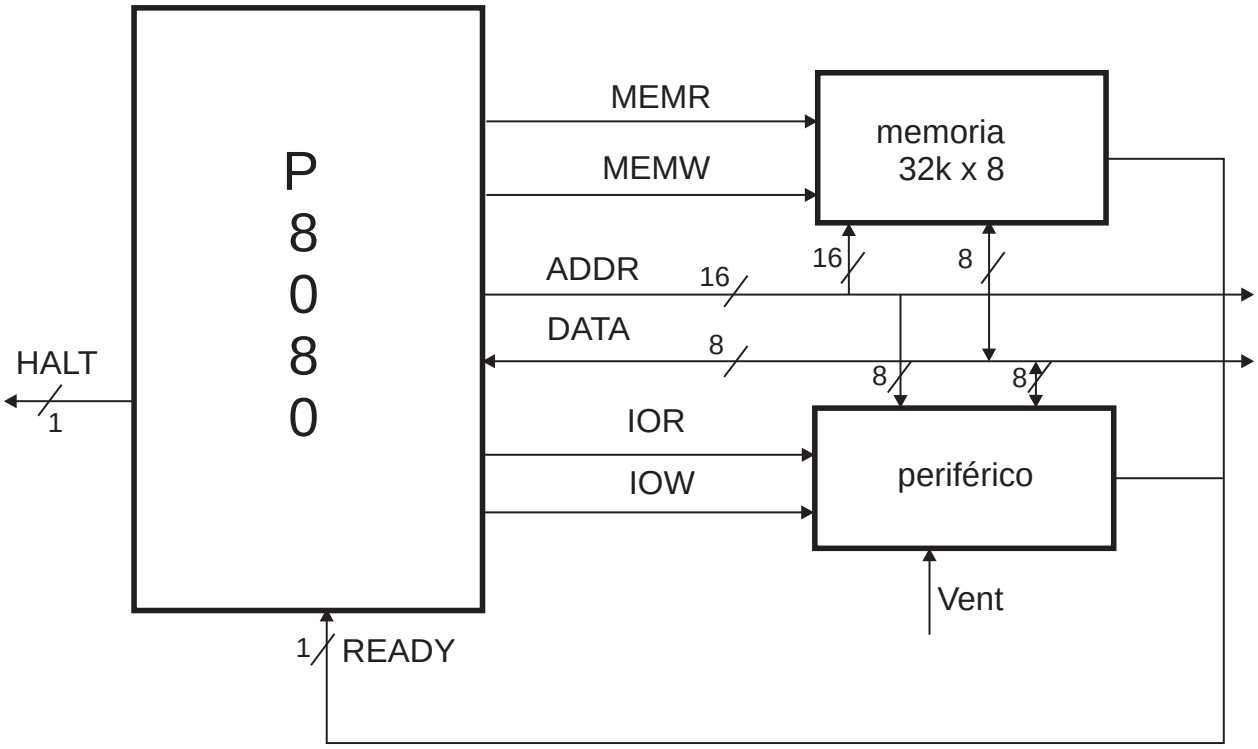


Figura 1.- Sistema completo. Señales de interconexión.

P8080E es capaz de direccionar 64k palabras de 8 bits a través de su bus de direcciones (ADDR) de 16 bits. Los datos van y vienen de memoria por el bus de datos (DATA) de 8 bits. P8080E pide ciclos de lectura en memoria (memory read) por medio de la señal MEMR, y ciclos de escritura por medio de MEMW (memory write). La memoria puede responder inmediatamente o pedir un tiempo extra a través de la línea READY (lista). De las 64 k palabras direccionables, sólo se han simulado las 32k primeras. El resto no existe. Escribir en una celda que no existe es como gritar en el desierto. Leer de una celda que no existe es como leer sin libro. No se provoca error alguno, pero el resultado es impredecible.

P8080E es capaz de relacionarse con 256 puertas (ports) de entrada y/o salida. En nuestro sistema sólo existe una de tales puertas que comunica con un periférico, es la puerta 0. El periférico es un convertidor A/D que nos permite convertir una tensión externa Vent entre 0 y 10 voltios en un código binario 0 a 255. Se puede escribir con ayuda de la señal IOW (input output write) y se puede leer con IOR (input output read), pudiendo responder el periférico inmediatamente o pedir un plazo por medio de READY. Como se ve, es simétrico de lo que teníamos en memoria.

Concretamente, en la puerta 0 podemos escribir 8 bits a través del bus de datos (DATA) y leer un bit a través de la línea inferior de dicho bus. Los detalles internos del periférico se verán más adelante.

Para terminar, P8080E genera una señal HALT (parado) cuando está parado, o más precisamente, al ejecutar la instrucción HLT P8080E se queda en un bucle sin salida, situación que indica al exterior por medio de la señal HALT. 8080 sale de este bucle por medio de interrupciones. Como P8080E carece de este mecanismo, es imposible sacarlo del bucle. De hecho se para.

3 PROTOCOLOS DE RELACIONES EXTERIORES.

P8080E utiliza dos fases de reloj no solapadas ϕ1 y ϕ2, a semejanza de su objeto de imitación 8080. Respecto a estas dos fases se dan todos los diagramas de tiempo.

Comenzaremos por los ciclos de lectura de memoria, que se atienen al esquema de la figura 2.

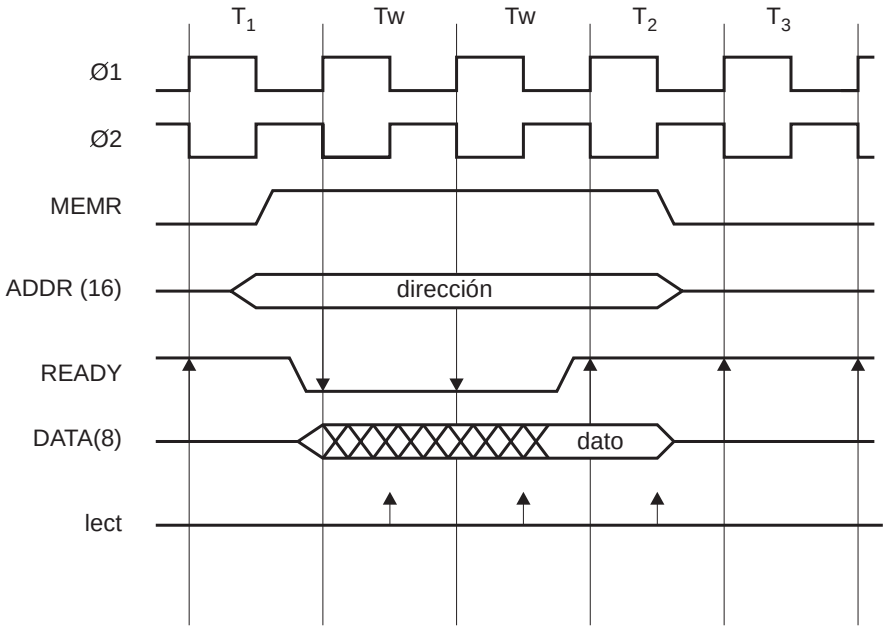


Figura 2.- Ciclo de lectura de memoria.

ϕ1 y ϕ2 son las dos fases de reloj. Hablaremos de los estados T (semejantes a los del 8080) definidos por los flancos ascendentes de ϕ1.

La señal MEMR sale de P8080E hacia la memoria. El 8080 genera algo parecido a través de la información de STATUS; pero P8080E es más simple. MEMR se lee “memory read” y marca un ciclo de memoria para realizar una lectura. MEMR se activa y desactiva con los flancos ascendentes de ϕ2.

ADDR es el bus de direcciones, 16 bits, que carga la dirección de memoria cuyo contenido deseamos leer. ADDR debe ir sincronizado con MEMR. Antes de activar MEMR nos es indiferente. La dirección debe permanecer estable durante el tiempo que dure MEMR. Tras T2 vuelve a sernos indiferente.

La memoria puede tener problemas personales de ella para localizar el dato. Si desea un desahogo de tiempo, lo indica mediante la señal READY, usualmente a nivel TRUE (lista), que pasa a FALSE (espera). P8080E estudia esta señal en los flancos ascendentes de ϕ1, instantes que se han marcado con pequeñas flechas en la figura 2.

Si READY = FALSE, nos quedamos en estados de espera Tw. En cuanto READY = TRUE, continuamos. Puede suceder que no haya ningún estado Tw. Basta para ello, que la memoria deje READY = TRUE. Puede suceder que haya 1, 2, 3, ... estados Tw. Depende de la rapidez de respuesta de la memoria.

La fila marcada **lect** no tiene realidad física. Simplemente nos indica que en los puntos indicados, el contenido del bus de datos DATA, entra en el P8080E hacia el destino deseado. Como se ve, durante los ciclos Tw se carga un dato que no está garantizado. No importa, la carga en T2 escribe el dato correcto sobre lo que se pueda haber cargado anteriormente.

Por último, DATA es el bus de datos, que debe contener el dato correcto, al menos el tiempo indicado en la figura 2. Es responsabilidad de la memoria cumplir este requisito. No olvidemos que la memoria puede pedir tiempos de espera Tw a través de READY.

Los ciclos de escritura (fig. 3) son muy parecidos. Las únicas diferencias afectan (1) a la señal MEMW, que sustituye a la MEMR de antes, y que indica escritura, y (2) al bus de datos, DATA. En efecto, DATA es ahora responsabilidad de P8080E, que es quien dice qué quiere escribir en la dirección de memoria indicada. El diagrama nos muestra el tiempo que P8080E se compromete a tener estable el dato. Es responsabilidad de la memoria tomarlo a tiempo. No olvidemos que la memoria siempre puede pedir tiempo extra a través de READY, forzando estados de espera Tw.

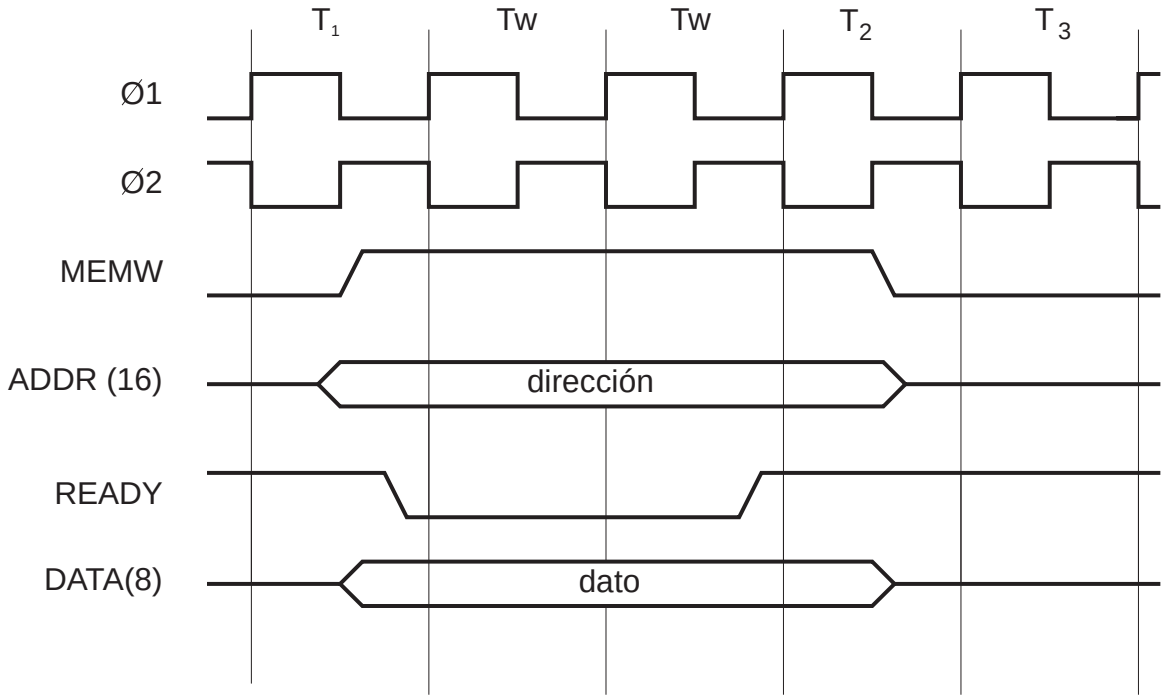


Figura 3.- Ciclo de escritura en memoria.

La entrada de datos es equivalente a la lectura (fig. 2), diferenciándose de aquella en que ahora se activa IOR en lugar de MEMR y que la dirección es sólo de 8 bits, que ocupan los 8 bits menos significativos del bus de direcciones ADDR.

Análogamente, la salida de datos se reconoce por la señal IOW, que sustituye a la MEMW característica de los ciclos de escritura (fig. 3). La dirección consta de 8 bits.

4 ORGANIZACIÓN INTERNA.

P8080E, por dentro, se parece mucho a su primo 8080. La figura 4 es fotocopia del manual de Intel.

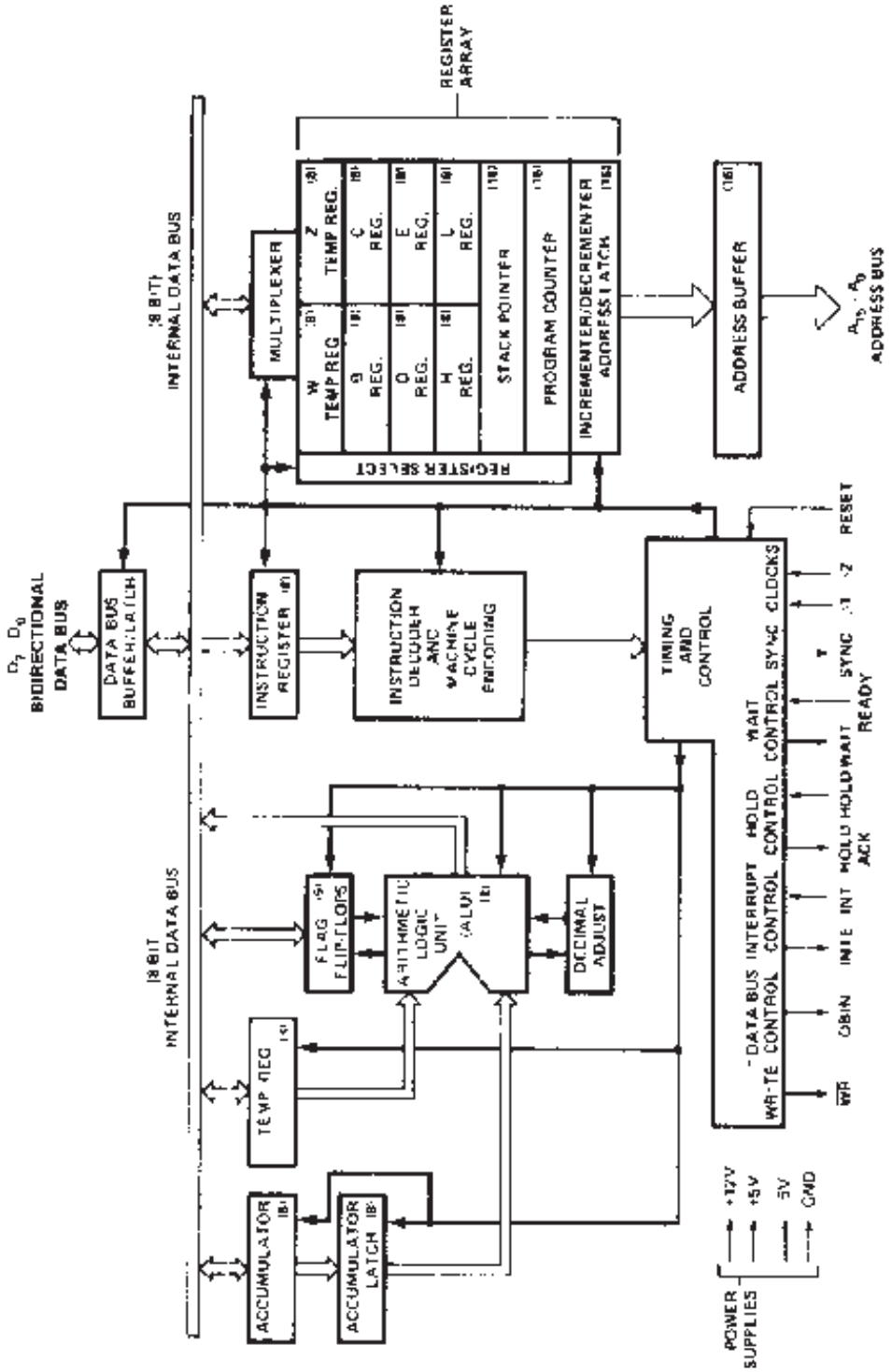


Figura 4.- Arquitectura y organización del Intel 8080

La figura 5 muestra los bloques en que se subdivide P8080E. El paralelismo es considerable. Las diferencias se han introducido para que P8080E sea microprogramable, sencillo y didáctico. La figura 5 también indica en qué otras figuras se descompone cada bloque. Creemos que una figura vale por mil palabras y con esta idea hemos organizado el índice de figuras de forma gráfica.

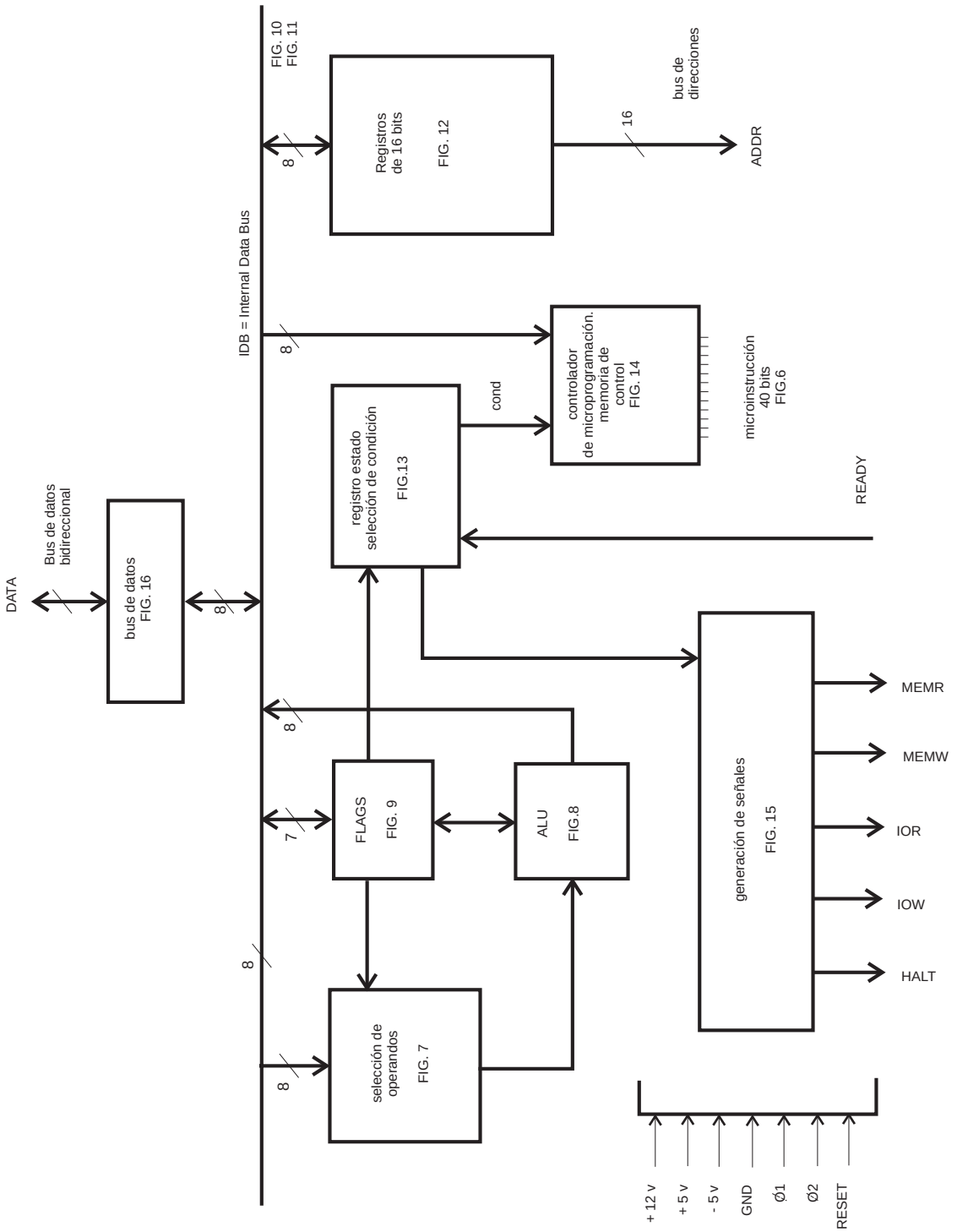


Figura 5.- Arquitectura y organización del P8080E

P8080E es microprogramable. Su memoria de control consta de 2048 micropalabras de 40 bits cada una. Para direccionar la memoria de control, necesitamos direcciones de 11 bits. Las micropalabras contienen microinstrucciones.

Cada microinstrucción controla lo que ocurre en todas y cada una de las partes de P8080E. Como orden de presentación de la organización interna de P8080E hemos preferido seguir el de los campos de la microinstrucción. Los aspectos de sincronización, fases de reloj, se tratarán al final.

La figura 6 presenta el formato de las microinstrucciones. En ella aparecen los distintos campos que pasamos a estudiar. Por cada campo se cita su nombre y el número de bits que incluye. Algunos campos están muy estrechamente relacionados entre sí, tanto que se comentan agrupados.

	dalt							
	selA, CY_A, TMP_0	invTMP, selCin, invCin, opALU	ldDI, selCY, modF	_IDB, IDB_	selrp, op16	H_L, ldADDR, selN, dalt_R	selcond, invcond, ssmi	save, actext, mem_IO, R_W, HALT
defecto	X X X X X	0 X 0 X X	0 1 1 0 0	X X 0 0 0	1 1 1 0 0	X 0 1 0 0	0 0 0 1 0	0 0 X X 0
bit nº	1 2 3 4 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31 32 33 34 35	36 37 38 39 40

Figura 6.- Formato de microinstrucción.

selA 3 bits fig.7

Nos permite seleccionar la entrada **A1** de la ALU (figs. 7 y 8).

SelA	A1
0	A
1	A/2
2	A*2
3	DAA
4	ACT
5	TMP
6	DI
7	-

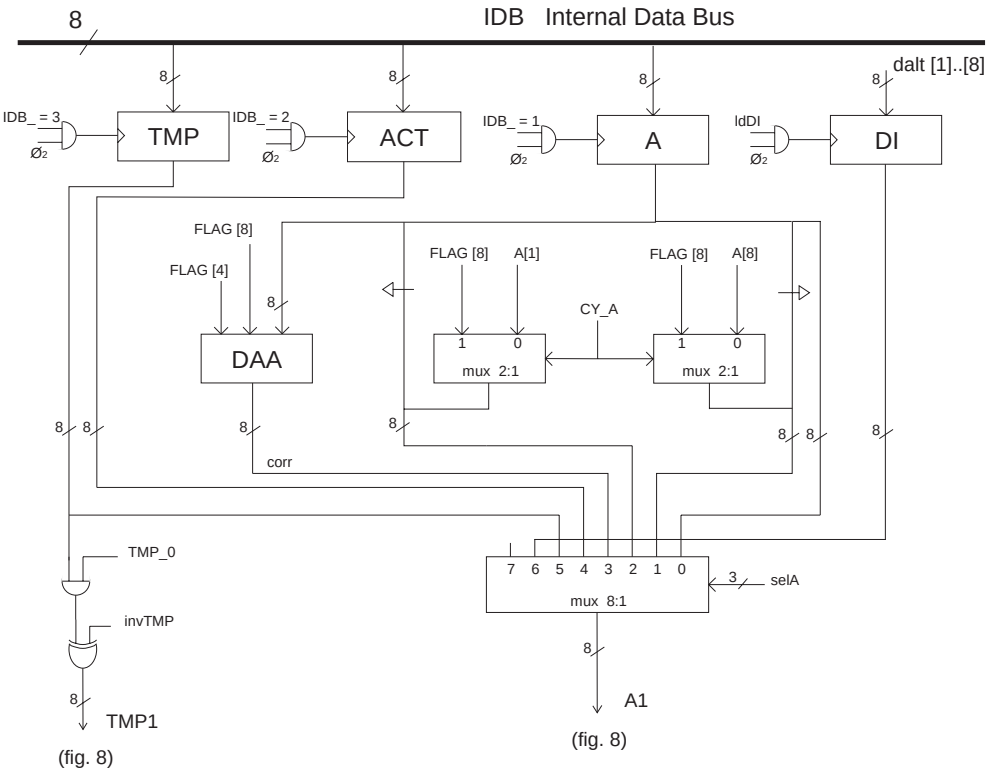


Figura 7.- Selección de operandos para la ALU

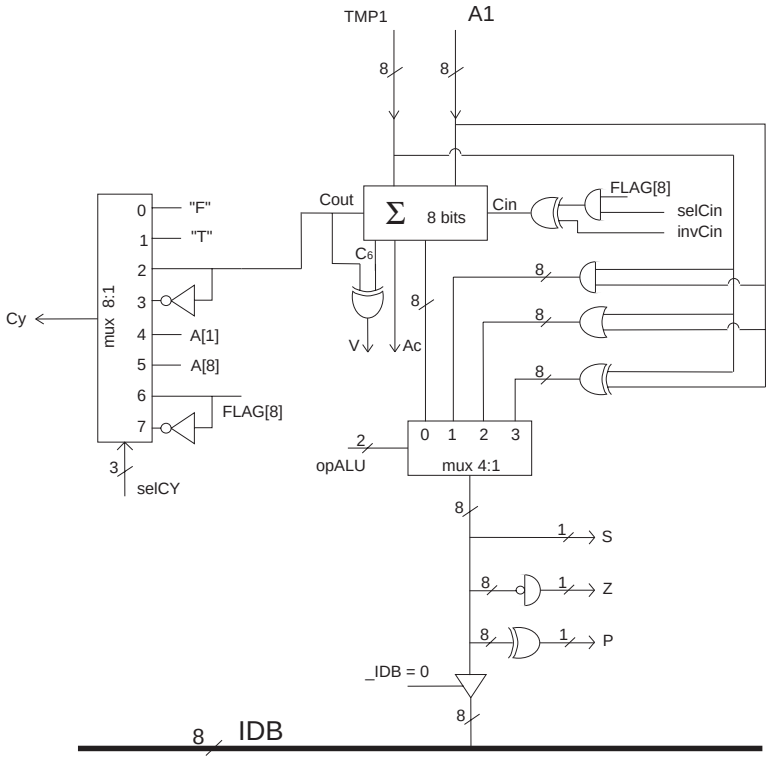


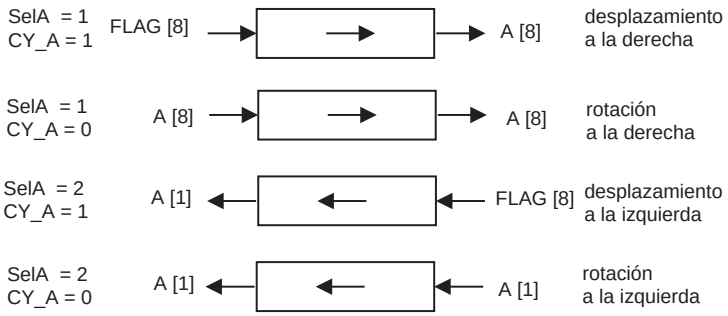
Figura 8.- ALU: Unidad Aritmético-Lógica

DAA es un circuito combinacional que, tras sumar de golpe dos números BCD de dos dígitos cada uno, nos dice si a alguno de ellos hay que aplicarle el factor de corrección 6. DAA es un circuito combinacional con la siguiente tabla de verdad:

CY	A[1..4]	AC	A[5..8]	corr
0	0..9	0	0..9	00
0	0..8	0	A..F	06
0	0..9	1	0..3	06
0	A..F	0	0..9	60
0	9..F	0	A..F	66
0	A..F	1	0..3	66
1	0..2	0	0..9	60
1	0..2	0	A..F	66
1	0..3	1	0..3	66

CY_A 1 bit fig. 7

Nos selecciona el bit que falta para completar un desplazamiento. Combinado con **selA** tenemos 4 variantes para **A1**. Nos serán útiles para instrucciones de desplazamiento y rotación:



TMP_0 1 bit fig. 7
invTMP 1 bit fig. 7

Estos dos campos nos seleccionan la entrada **TMP1** del sumador (fig. 8). Entre ambos disponemos de:

TMP_0	invTMP	TMP1
0	0	00000000 = cero
0	1	11111111 = -1 (comp. a 2)
1	0	TMP para sumar
1	1	**not** TMP para restar

selCin 1 bit fig. 8
invCin 1 bit fig. 8

Estos dos campos deciden la entrada **Cin** al sumador de la fig. 8:

selCin	invCin	Cin	
0	0	0	para ADD
0	1	1	para SUB
1	0	CY	para ADC
1	1	**not** CY	para SBB

opALU 2 bits fig. 8

Selecciona la operación que se realiza en la ALU:

opALU	ALU	
0	A1 + TMP1 + Cin	sumas y restas
1	A1 and TMP1	
2	A1 or TMP1	
3	A1 xor TMP1	

ldDI 1 bit fig. 7

Carga los primeros 8 bits del campo **dalt** en el registro **DI** (Dato inmediato). Este bit forma parte del campo **dalt**, por lo que el dato inmediato sólo estará en el registro **DI** hasta se realice algún salto.

selCY 3 bits fig. 8

Selecciona un nuevo candidato a ocupar la octava posición del registro FLAG. Véase la figura 9.

selCY	CY	
0	F	puesta a 0
1	T	puesta a 1
2	C7	acarreo de salida del sumador
3	not C7	idem negado, para restas
4	A[1]	bit más a la izquierda del acumulador A
5	A[8]	bit más a la derecha del acumulador A
6	FLAG[8]	antiguo CY
7	not FLAG[8]	antiguo CY negado

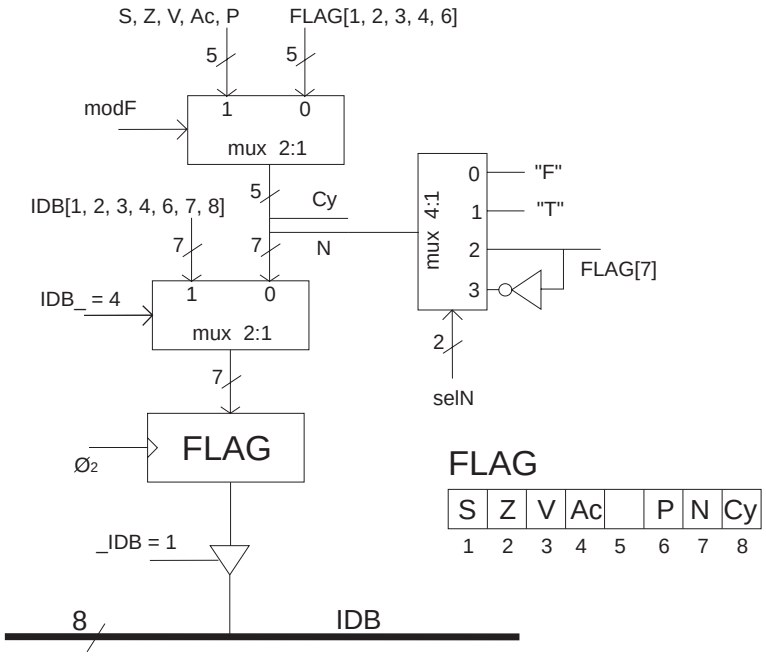


Figura 9.- La palabra FLAG

modF	1 bit	fig. 9

Controla si se modifican o no los FLAGS S, Z, V, AC y P. Muchas instrucciones no alteran estos FLAGS.

_IDB	2 bits	fig. 10 y 11
IDB_	3 bits	fig. 10 y 11

Estos dos campos controlan el uso del bus de datos interno, IDB, para transmitir datos de una parte a otra (fig. 10). _IDB controla quién sale a IDB, mientras que IDB_ especifica el destino. La figura 10 contiene los decodificadores de estos campos.

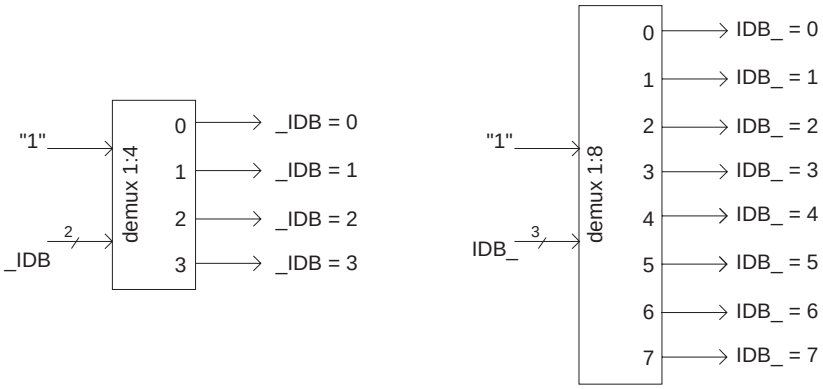


Figura 10.- Decodificadores que afectan al IDB.

La figura 11 explica la función de cada caso.

_IDB	fuente
0	ALU
1	FLAG
2	REGS
3	DATA

IDB_	destino
0	a ninguna parte
1	A
2	ACT
3	TMP
4	FLAG
5	REGS
6	DATA
7	IR

Figura 11.- Codificación de las entradas y salidas a/de IDB.

Algunos ejemplos de transferencias posibles pueden ser:

ALU→A	_IDB = 0	IDB_ = 1
DATA→ IR	_IDB = 3	IDB_ = 7
FLAG→ DATA	_IDB = 1	IDB_ = 6
etc.		

selrp	3 bits	fig. 12
op16	2 bits	fig. 12
H_L	1 bit	fig. 12
ldADDR	1 bit	fig. 12

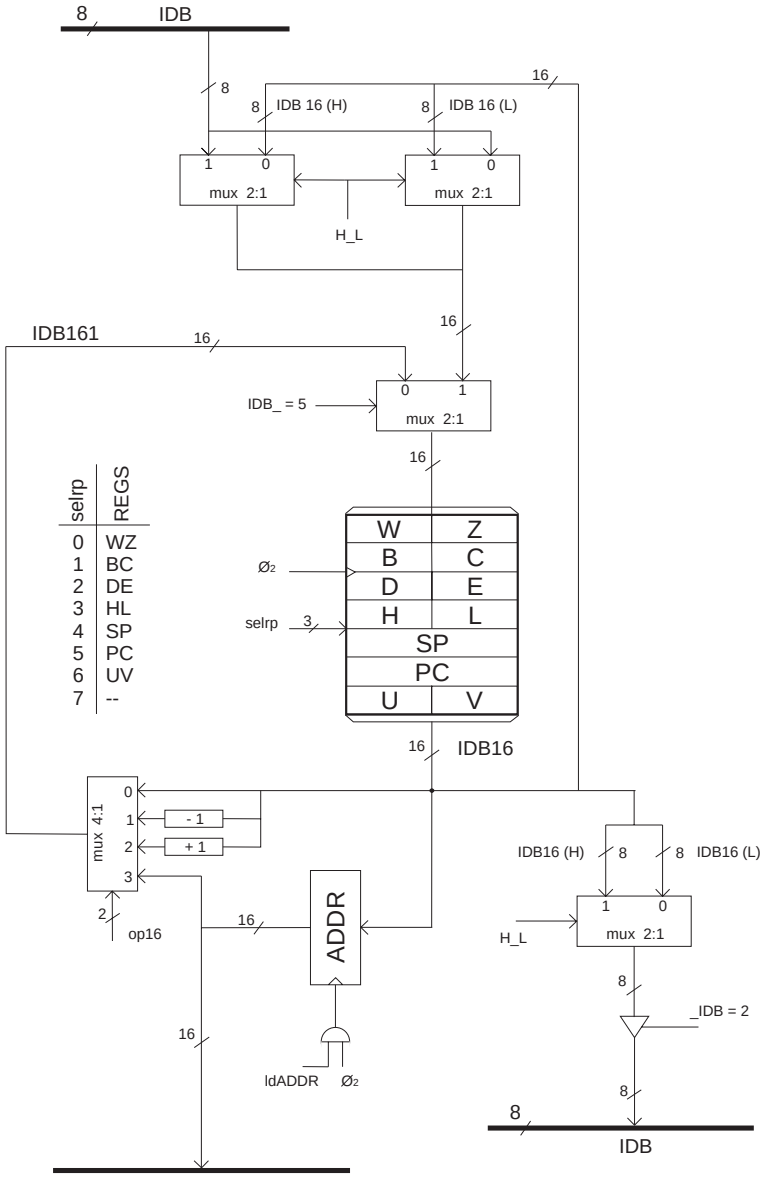


Figura 12.- Registros de 16 bits.

Estos cuatro campos controlan la parte de 16 bits del P8080E.

selrp - selecciona un par de registros, 16 bits.

op16 - decide la realimentación de los registros.

H_L - selecciona entre los 8 bits superiores o los 8 inferiores.

ldADDR - carga 16 bits en el registro de direcciones ADDR.

El par seleccionado (ver tabla de clave en la propia figura 12) sale a IDB16.

IDB sólo cubre 8 bits. De los 16 de IDB16, el campo H_L selecciona los 8 bits superiores (H_L = 1) o los 8 bits inferiores (H_L = 0). Sólo sale a IDB si el campo _IDB = 2 (REGS).

IDB16 se carga en el registro ADDR si y sólo si ldADDR = 1.

A efectos de entrada, carga de registros, tenemos las siguientes posibilidades:

1.- Cargar desde IDB. Esto es así si IDB_ = 5. Se cargan los 8 bits superiores (H_L = 1) o los 8 inferiores (H_L = 0). Los registros de la figura 12 siempre se gestionan por pares, es decir, como registros de 16 bits. Si queremos modificar sólo los 8 bits desde el bus IDB, habrá que reciclar los otros 8 bits desde IDB16. Más concretamente, si tenemos en BC un contenido bc y queremos alterar B escribiendo b' en él, tendremos que combinar b' con c para obtener b'c, que se carga en BC. Así queda modificado B y repite C. De estos cambalaches se encargan los dos multiplexores de la figura 12, parte superior.

2.- Cargar desde IDB16, cuando IDB_≠ 5. IDB16 lo controla el campo op16 que permite elegir entre 4 opciones:

op16	IDB16
0	se conserva el contenido de rp
1	se decrementa en 1 - decrp
2	se incrementa en 1 - incrp
3	se toma el contenido del registro ADDR

selN 2 bits fig. 9

Este campo permite definir el nuevo valor del flag N (Nivel de ejecución). Hay cuatro posibilidades:

selN		
0	F	Puesta a 0
1	T	Puesta a 1
2	Flag [7]	Antiguo N
3	**not** Flag[7]	Antiguo N negado

dalt_R 1 bit fig. 14

Este campo indica si los saltos a la dirección especificada por **dalt** se realizan de forma absoluta (**dalt_R** = 0) o relativa al contador de microprograma (**dalt_R** = 1).

selcond 3 bits fig. 13

Nos permite seleccionar entre 8 condiciones para controlar la dirección de la siguiente microinstrucción a ejecutar.

selcond		
0	TRUE	incondicional
1	FLAG[1]	signo (S)
2	FLAG[2]	cero (Z)
3	FLAG[6]	paridad (P)
4	FLAG[8]	acarreo (CY)
5	READY	señal exterior
6	FLAG[3]	overflow (V)
7	FLAG[7]	nivel de ejecución (N)

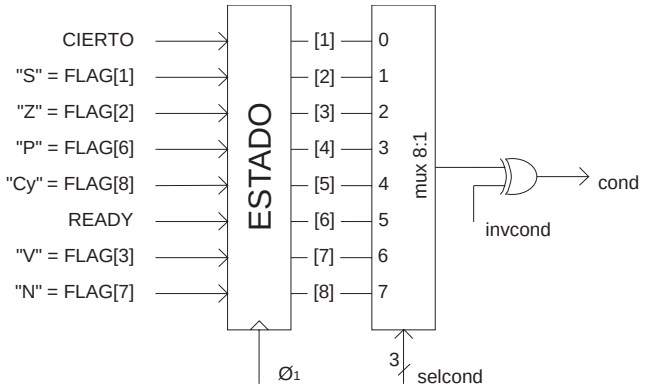


Figura 13.- Registro de estado. Selección de condición.

invcond 1 bit fig. 13

Niega la condición seleccionada por selcond.

ssmi 1 bit fig. 14

Selección de la siguiente microinstrucción. Junto con la condición **cond**, determinada por **selcond** e **invcond**, tenemos las siguientes posibilidades:

cond	ssmi	microdirección
0	0	la siguiente, mpc+1
0	1	la reservada en el registro μP(RET)
1	0	la indicada en el campo dalt (bits 1 .. 11)
1	1	IR * 8

mpc es el contador de microprograma. En cada momento, apunta a la microinstrucción que estamos ejecutando. Lo más usual es ejecutar microinstrucciones consecutivas. Para ello utilizaremos la fila 1 de la tabla anterior.

El registro μ**P** son en realidad tres registros organizados en forma de pila. Constituyen la micropila del P8080E y permiten realizar tres niveles de microsubrutinas. Cuando se selecciona esta opción se realiza implícitamente una operación **POP** de la micropila. Se carga con una operación **PUSH** cuando lo ordena el campo **save** que veremos a continuación.

dalt son 11 bits de la microinstrucción que nos permiten ir explícitamente a una determinada dirección. Podemos saltar a cualquier sitio de la memoria de control. El campo **dalt** especifica una dirección absoluta o un desplazamiento en complemento a 2 con respecto al contador de microprograma dependiendo del campo **dalt_R**. Si **save = 1**, **dalt** especifica la dirección de una microsubrutina.

IR*8 significa multiplicar el contenido de este registro por 8, es decir, añadir 3 ceros a la derecha. De esta forma seleccionamos la microrrutina que implementa cada instrucción. Esta posibilidad de elección es básica para que P8080E interprete el juego de instrucciones del 8080.

Eventualmente, tras la fase de FETCH, tendremos en el registro de instrucción IR un código de operación. Sea el 21H, que corresponde a la instrucción LXI H, d16. ¿Dónde están las microinstrucciones para interpretar esta instrucción? Sencillo: a partir de la dirección 21H•8 = 108H. P8080E debe saltar a la dirección 108H. Este salto se hace utilizando la última fila de la tabla anterior.

save 1 bit fig. 14

Si se selecciona el campo **dalt** como la dirección de la siguiente microinstrucción a ejecutar (**cond** = 1 y **ssmi** = 0) y **save = 1**, se salva en la micropila la dirección de la microinstrucción siguiente a la que estamos ejecutando, es decir, mpc + 1. De esta forma luego podremos retornar a ella saltando a μP. Las direcciones almacenadas avanzan un lugar en la micropila, es decir, se realiza una operación **PUSH**. La dirección almacenada en el último registro de la micropila **se pierde**. Dado que μP tiene una profundidad de tres registros, **sólo se podrán anidar tres saltos a microsubrutina**.

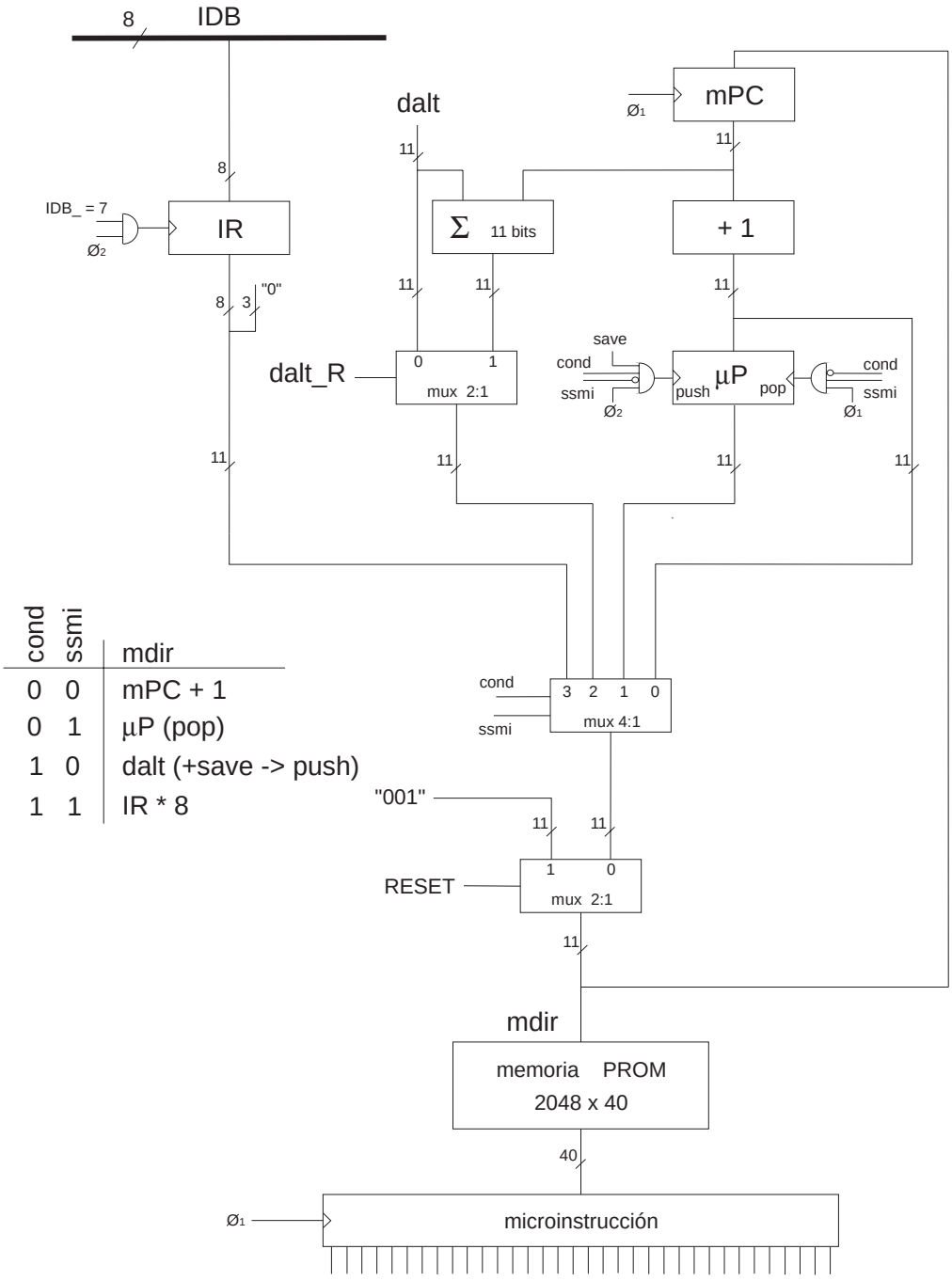


Figura 14.- Control del microprograma. Memoria de control.

actext	1 bit	fig. 15
mem_io	1 bit	fig. 15
R_W	1 bit	fig. 15

Estos campos controlan la generación de la señales para acceder a memoria y a los periféricos.

actext	mem_io	R_W	se activa:
1	1	1	MEMR
1	1	0	MEMW
1	0	1	IOR
1	0	0	IOW
0	x	x	ninguna

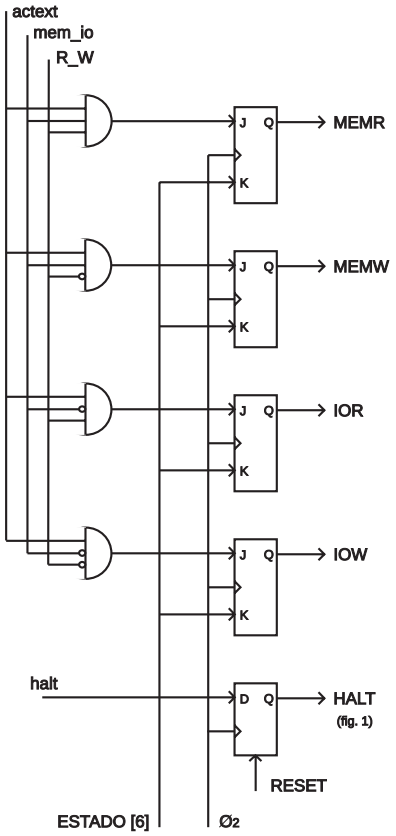


Figura 15.- Generación de señales externas.

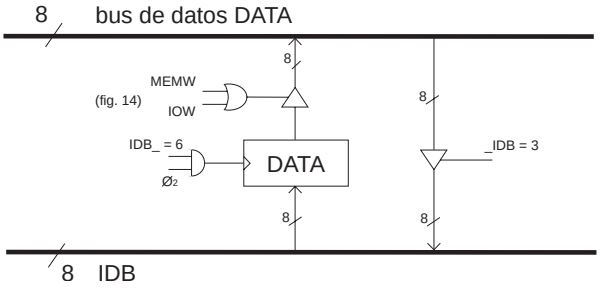


Figura 16.- Bus de datos.

HALT 1 bit fig. 1

Si TRUE, P8080E se para.

dalt 11 bits fig. 14

Campo solapado con otros que nos da una dirección alternativa de salto para seleccionar la siguiente dirección. Además de esto, sus 8 primeros bits pueden ser cargados en el registro DI para realizar operaciones en la ALU sobre datos inmediatos. La carga de este registro se gobierna con el campo **ldDI**. Obsérvese que el campo **dalt** está solapado con los campos que controlan la ALU y que estas tres interpretaciones de los 11 primeros bits del registro de control son mutuamente excluyentes.

5 FLAGS.

Quizás merecen atención especial los FLAGS, conjunto de 7 bits de información auxiliar de P8080E a disposición del programador. Esta información se denomina a veces de estado, término que evitaremos para no confundirlo con el estado de la máquina. Hay unos FLAGS que aparecen en la figura 9 y un registro de estado que aparece en la figura 13. En esta sección tratamos exclusivamente de los primeros.

Tras una operación en la ALU, P8080E se reserva alguna información sobre aquella. Concretamente, evalúa seis valores lógicos que son (fig. 8):

S	TRUE si el resultado es negativo. S: = bit de “signo”.
Z	TRUE si el resultado es cero. Z:= todos los bits son FALSE.
V	TRUE si se produce overflow en complemento a 2, V:= C_6 XOR C_7
AC	TRUE si hubo acarreo de la posición 3 en el sumador de la ALU. AC:= C_3.
P	TRUE si el número de 1’s es par.
CY	varias posibilidades, ver fig. 8.

Por otro lado hay otro biestable de estado que no está asociado a la ALU y que sirve para almacenar el nivel o modo de ejecución del procesador:

N varias posibilidades, ver fig. 9.

P8080E guarda estos valores en el registro FLAG. La figura 9 muestra a qué valor se dedica cada una de las 8 posiciones de este registro. Un bit se desperdicia; además, puede observarse que no está conectado a ningún elemento, por lo que el registro FLAG no puede utilizarse para almacenar valores temporalmente. Este registro puede cargarse directamente desde IDB, si IDB_= 4, o puede cargarse con los valores evaluados en la figura 8 . En este segundo caso van por separado S, Z, V, AC y P, de una parte, y N y CY de otra.

S, Z, V, AC y P se evalúan en la figura 8 siempre de la misma forma. Si **modF** entonces se actualizan en FLAG. Si no, se repite el valor anterior, esto es, las posiciones de FLAG a ellos dedicadas no se modifican.

CY se selecciona con **selCY** (fig. 8). Si no queremos modificarlo, se repite su valor con selCY = 6.

N se selecciona con **selN** (fig. 9). Si no queremos modificarlo, se repite su valor con selN = 3.

El contenido del registro FLAG puede afectar diversas zonas de P8080E, a saber:

- sale a IDB	si _IDB = 1	fig. 9
- desplazamientos	si CY_A = 1	fig. 7
- DAA	si selA = 3	fig. 7

- sumador	si selCin = 1	fig. 8
- reciclaje de S, Z, V, AC y P	si modF = 0	fig. 9
- reciclaje de CY	si selCin = 6, 7	fig. 8
- reciclaje de N	si selN = 2, 3	fig. 9
- estado de la máquina	si selcond = 1, 2, 3, 4,6, 7	fig. 13

6 PIPELINE.

P8080E está organizado en forma de pipeline con dos segmentos que se detallan en la figura 17. El segmento I se encarga de la selección de la siguiente microinstrucción, mientras que el segmento II gestiona las operaciones sobre datos y registros.

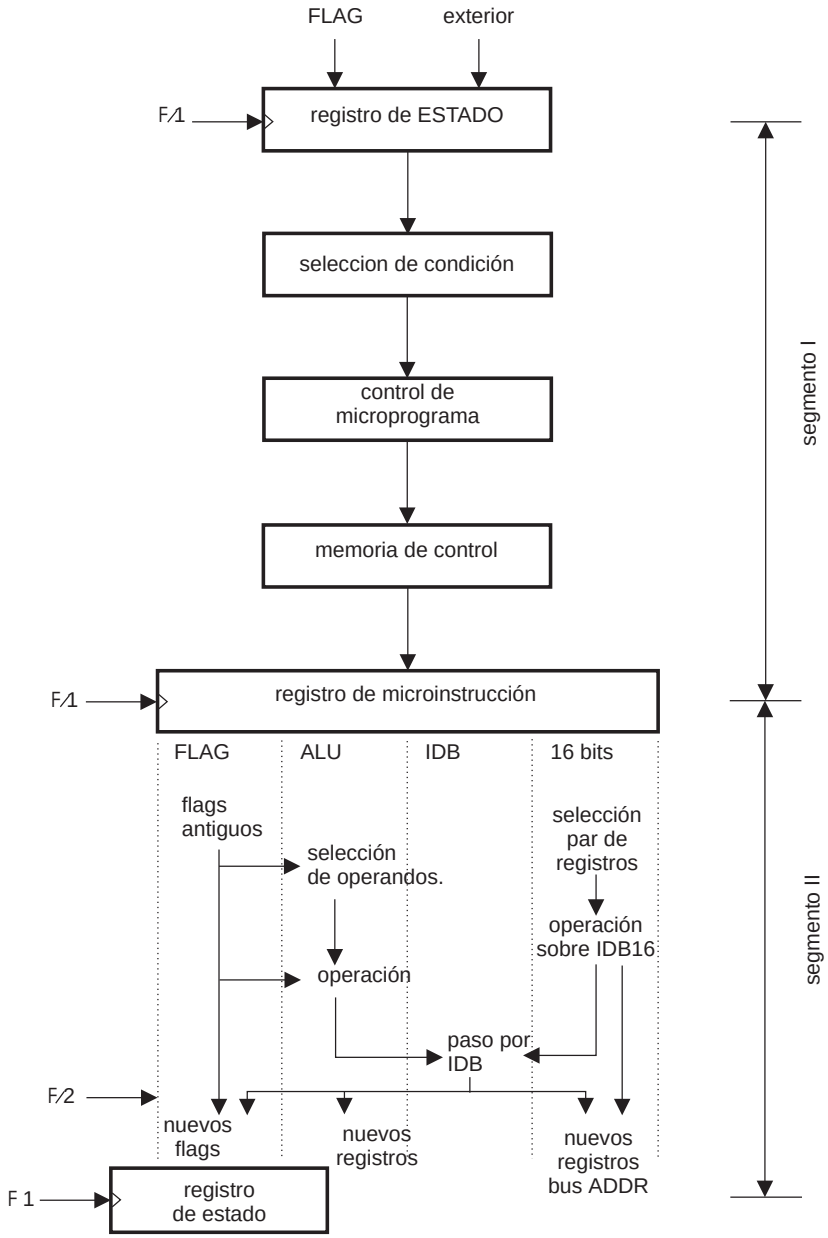


Figura 17.- Organización del pipeline.

El reloj $\phi 1$ controla ambos segmentos que se sincronizan a través del registro de estado (fig. 13) y del registro de microinstrucción (fig. 14).

La sección I arranca en el registro de estado, pasa por la lógica de condiciones, el controlador de microprograma que decide cuál será la siguiente microinstrucción, y la memoria de control que nos la da. Termina depositando la próxima microinstrucción a las puertas del registro de microinstrucción.

En este último arranca la sección II que distribuye microinstrucciones para la sección I, para la ALU, los registros, los buses, etc. En particular genera los FLAGS que pueden controlar la actividad de la sección I en el siguiente microciclo.

Es un tipo de segmentación (pipeline) muy frecuente. Se intenta que dos procesos lentos, la búsqueda de microinstrucciones en memoria, y la gestión de los datos, ocurran en paralelo, acelerando la máquina. El problema o característica de este tipo de organización reside en que un salto condicional de microprograma se realiza sobre el estado de la máquina al final de la microinstrucción anterior, no sobre el estado final del ciclo actual. Este aspecto debe cuidarse al escribir los microprogramas.

μdir {i+1} = **función** (μinst {i}, ESTADO {i-1}).

La figura 17 nos detalla la sincronización de las varias actividades de P8080E. Nótese que el segmento II se parte en dos subfases, la que arranca con el flanco ascendente de $\phi 1$ y la que arranca con el flanco ascendente de $\phi 2$.

El registro de estado, fig. 13, cumple una función auxiliar, cual es sincronizar la señal asíncrona READY, procedente de memoria o de periféricos.

7 MICROPROGRAMACIÓN.

P8080E arranca con un RESET, fig. 14, que le lleva a ejecutar la microinstrucción 1. Aquí debe microprogramarse el conjunto de operaciones que constituye el arranque del procesador: debe preverse la inicialización del PC, contador de programa, con la dirección 0000 de memoria. La primera instrucción que se ejecuta es pues la que hay en esa posición de memoria principal.

Se pasa a la fase de búsqueda o FETCH. Para ello se lee de memoria el contenido de la posición indicada por PC. Se hace un ciclo de lectura en memoria. El dato, que es un código de operación, se trae a IR, el registro de instrucción. Este se utilizará para saltar a la microrrutina correspondiente para ejecutar la instrucción. Al final de ésta, regresamos a hacer el FETCH de la siguiente instrucción, etc.

La instrucción HALT es un poco especial. Su interpretación habitual es quedarse en un bucle por tiempo indefinido, sin saltar a FETCH. Para P8080E, HALT implica pararse definitivamente.

Es misión del microprogramador, el alumno, prever microinstrucciones para la fase de RESET, de FETCH y cada una de las instrucciones del 8080.

La microprogramación de P8080E es laboriosa, pero bastante simple. En esta sección presentamos un posible lenguaje microensamblador para microprogramar P8080E. Este lenguaje no está implementado, por lo que el alumno tendrá que traducir personalmente a los bits correspondientes de la micropalabra. De todas formas creemos interesante distinguir dos fases:(1) escribir el microprograma en el lenguaje fuente y (2) microensamblarlo a mano. Otra opción sería codificar a base de ceros y unos directamente sobre una hoja de codificación.

Cada microinstrucción es una serie de campos separados por comas, terminando en punto y coma. Podemos anteponer una etiqueta numérica que fuerza la dirección donde se carga la micropalabra en la memoria de control, la dirección real.

La figura 18 presenta un ejemplo de microprograma. La primera línea es un comentario. Todas las líneas que comienzan por "." son líneas de comentario.

```
. RESET

.       Carga el valor 0000H en el contador de programa. Primero el byte bajo y después el alto.
1:      TMP_0 = 0, opALU = AND, ALU -> REGS, rp = PC(L);
        TMP_0 = 0, opALU = AND, ALU -> REGS, rp = PC(H);


. FETCH

.       Lee el contenido de la dirección de memoria apuntada por el PC y lo guarda en el registro
.       de instrucción (IR).
3:      rp = PC, incrp, ldADDR, MEMR;
        DATA -> IR, jmp_r $ if not READY;
.       Salta a la microrrutina correspondiente al código de operación almacenado en el registro IR.
        jmp inst;


. LXI H, d16  ... 21H

.       Lee el operando de 16 bits y lo almacena en el registro HL. Primero la parte baja.
108:    rp = PC, incrp, ldADDR, MEMR;
        DATA -> REGS, rp = HL(L), jmp $ if not READY;
.       A continuación la parte alta:
        rp = PC, incrp, ldADDR, MEMR;
        DATA -> REGS, rp = HL(H), jmp $ if not READY;
.       Salta a la rutina de fetch
        jmp 003H;


. MOV A, M  .... 7EH

.       Lee el contenido de la dirección de memoria apuntada por el registro HL y lo guarda en el
.       Acumulador.
3F0:    rp = HL, ldADDR, MEMR;
        DATA -> A, jmp $ if not READY;
.       Salta a la rutina de fetch
        jmp 003H;


. INR A       .... 3CH

.       Realiza la operación A <- A + 00H + 1 modificando los flags.
1E0:    selA = A, TMP_0 = 0, selCin = 0, invCin, opALU = ADD, modF, selCy = Cout, ALU -> A;
.       Salta a la rutina de fetch.
        jmp 003H;

. HALT        ... 76H

3B0:    HALT, jmp $;
```

Figura 18.- Ejemplo de microprograma.

Comenzamos en la posición 001 de la memoria de control que llamaremos RESET. Es aquí por donde comienza a ejecutar P8080E, ya que RESET se activa al arrancar (véase la figura 14) y nos lee la micropalabra de la dirección 1.

Esta primera microinstrucción nos dice que:

1.- El campo TMP_0 vale 0, fig. 7.

2.- La ALU realiza una operación AND, es decir, opALU = 1, ver fig. 8.

3.- De la ALU pasamos a los registros. Esto se hace a través de IDB. Para ello hacemos _IDB = 0 e IDB_= 5, ver fig. 11.

4.- Se selecciona el par de registros PC, selrp = 5. De los 16 bits de PC, la parte baja, H_L = 0, ver figura 12.

Los demás campos toman los valores por defecto que se indica en la figura 6.

Total, que hemos seleccionado un 00 por TMP1 y hemos hecho un AND con otra cosa cualquiera. Con este resultado, que vale 00, salimos a IDB, para terminar en el registro PC, parte inferior (fig. 12).

La siguiente microinstrucción hace lo mismo cargando 00 en los 8 bits superiores de PC.

Con esto termina la labor de RESET que ha cargado ceros en los 16 bits de PC y ha seleccionado ejecución en modo usuario. Esto es, pone el contador de programa a cero. Por ello la primera instrucción que se ejecutará será la residente en la posición 0000 de memoria principal.

La tercera microinstrucción es la primera de una serie dedicada a localizar en memoria principal el código de operación de la instrucción que hay que ejecutar. Se conoce por el nombre propio de FETCH (traída). Hace lo siguiente:

(1) selrp = 5,	fig. 12
(2) OP16 = 2,	fig. 12
(3) IdADDR = 1,	fig. 12
(4) actext = 1, mem_io = 1, R_W = 1;	fig. 15

Es decir, que sacamos el registro PC al bus de direcciones, al paso que lo incrementamos y lo guardaremos incrementado. Además arrancamos un ciclo de lectura en memoria principal.

La microinstrucción 4 nos dice que:

(1) _IDB = 3, IDB_ = 7,	fig. 11
(2) selcond = 5, invcond = 1, ssmi = 0, dalt = 000, dalt_R = 1;	fig. 14

Es decir se carga el dato que esta en bus externo (DATA) en el registro de instrucción (IR) y comprobamos si la memoria ha terminado con la lectura del dato (READY = 1), en cuyo caso pasamos a ejecutar la microinstrucción siguiente. Si no ha terminado (READY = 0), el dato que hemos leído no es correcto (es basura), por lo que debemos repetir la operación y, por tanto, saltamos a la misma microinstrucción. El salto lo hacemos relativo al contador de microprograma (dalt_R = 1) con desplazamiento cero (dalt = 000).

La microinstrucción 5 dice que:

(1) selcond = 0, invcond = 0, ssmi = 1;	fig. 14

Saltando incondicionalmente a IR•8.

Se han añadido varias microinstrucciones más, a modo de ejemplo. Concretamente las microrrutinas necesarias para interpretar las siguientes instrucciones del 8080:

LXI	H,d16	código: 21H más 2 bytes para el dato
MOV	A,M	código: 7EH
INR	A	código: 3CH
HLT		código: 76H

Al no existir un traductor automático o microensamblador, no insistiremos en la definición sintáctica del lenguaje. El alumno es muy libre de utilizar sus propios convenios para los demás campos. Simplemente le agradeceremos que sean claros y útiles.

El usuario deberá microensamblar a mano las microinstrucciones que desee que ejecute P8080E. Para ello se utiliza una plantilla de codificación como la mostrada en la figura 6. Esta plantilla incluye un valor por defecto para cada uno de los campos. La mecánica de traducción es la siguiente:

para cada campo de la microinstrucción:
  si se especifica su valor en el microprograma
    entonces llenarlo con el valor especificado
  si no, llenarlo con el valor por defecto.

Por ejemplo, la primera microinstrucción de la figura 18 nos da la siguiente información:

dirección = 001H	especificada	
selA = xxx	por defecto	da igual
CY_A = x	por defecto	da igual
TMP_0 = 0	especificada	
invTMP = 0	especificada	TMP1=0
selCin = x	por defecto	
invCin = 0	por defecto	da igual
opALU = 01	especificada	AND
ldDI = 0	por defecto	no se carga DI
selCY = 110	por defecto	no se modifica FLAG[8]
modF = 0	por defecto	no se modifica FLAG[1, 2, 3, 4, 6]
_IDB = 00	especificada	ALU →
IDB_ = 101	especificada	→ REGS
selrp = 101	especificada	PC
op16 = 00	por defecto	da igual
H_L = 0	especificada	parte baja, L
ldADDR = 0	por defecto	no se carga el bus ADDR
selN = 10	por defecto	no se modifica
dalt_R = 0	por defecto	
selcond = 000	por defecto	CIERTO
invcond = 1	por defecto	FALSO
ssmi = 0	por defecto	siguiente
save = 0	por defecto	no se carga μP
actext = 0	por defecto	no se activa nada
mem_io = x	por defecto	
R _W = x	por defecto	
HALT = 0	por defecto	P8080E activo

La figura 19 muestra el microprograma fuente de la figura 18 y las micropalabras resultantes del proceso de microensamblado manual.

```
. RESET
.       Carga el valor 0000H en el contador de programa. Primero el byte bajo y después el alto.
. 1:    TMP_0 = 0, opALU = AND, ALU -> REGS, rp = PC(L);
001 XXXX0 0X001 01100 00101 10100 00100 00010 00XX0
.       TMP_0 = 0, opALU = AND, ALU -> REGS, rp = PC(H);
002 XXXX0 0X001 01100 00101 10100 10100 00010 00XX0
.
.
. FETCH
.       Lee el contenido de la dirección de memoria apuntada por el PC y lo guarda en el registro de instrucción (IR).
. 3:    rp = PC, incrp, ldADDR, MEMR;
003 XXXXX 0X0XX 01100 XX000 10110 X1100 00010 01110
.       DATA -> IR, jmp_r $ if not READY;
004 00000 00000 01100 11111 11100 X0101 10110 00XX0
.       Salta a la microrrutina correspondiente al código de operación almacenado en el registro IR.
.       jmp inst;
005 XXXXX 0X0XX 01100 XX000 11100 X0100 00001 00XX0
.
.
. LXI H, d16  ... 21H
.       Lee el operando de 16 bits y lo almacena en el registro HL. Primero la parte baja.
. 108: rp = PC, incrp, ldADDR, MEMR;
108 XXXXX 0X0XX 01100 XX000 10110 X1100 00010 01110
.       DATA -> REGS, rp = HL(L), jmp $ if not READY;
109 00100 00100 11100 11101 01100 00100 10110 00XX0
.       A continuación la parte alta:
.       rp = PC, incrp, ldADDR, MEMR;
10A XXXXX 0X0XX 01100 XX000 10110 X1100 00010 01110
.       DATA -> REGS, rp = HL(H), jmp $ if not READY;
10B 00100 00101 11100 11101 01100 10100 10110 00XX0
.       Salta a la rutina de fetch
.       jmp 003H;
10C 00000 00001 11100 XX000 11100 X0100 00000 00XX0
.
. MOV A, M  .... 7EH
.       Lee el contenido de la dirección de memoria apuntada por el registro HL y lo guarda en el Acumulador.
. 3F0: rp = HL, ldADDR, MEMR;
3F0 XXXXX 0X0XX 01100 XX000 01100 X1100 00010 01110
.       DATA -> A, jmp $ if not READY;
3F1 01111 11000 11100 11001 11100 X0100 10110 00XX0
.       Salta a la rutina de fetch
.       jmp 003H;
3F2 00000 00001 11100 XX000 11100 X0100 00000 00XX0
.
. INR A       .... 3CH
.       Realiza la operación A <- A + 00H + 1 modificando los flags.
. 1E0: selA = A, TMP_0 = 0, selCin = 0, invCin, opALU = ADD, modF, selCy = Cout, ALU -> A;
1E0 000X0 00100 00101 00001 11100 X0100 00010 00XX0
.       Salta a la rutina de fetch.
.       jump 003H;
1E1 00000 00001 11100 XX000 11100 X0100 00000 00XX0
.
. HALT        ... 76H
. 3B0: HALT, jmp $;
3B0 01110 11000 01100 XX000 11100 X0100 00000 00XX1
.
/
0000 21             LXI     H, 0010
0001 10
0002 00
0003 7E             MOV     A, M
0004 3C             INR     A
0005 76             HLT
0010 99
/
```

Figura 19.- Microprograma ejemplo: simbólico y microensamblado.

8 EL PERIFÉRICO.

El periférico simulado se detalla en la figura 20. Consta de un registro que se carga al escribir en la puerta 0, un conversor de digital a analógico y un comparador de tensiones analógicas que devuelve un valor digital que puede leerse por la puerta 0.

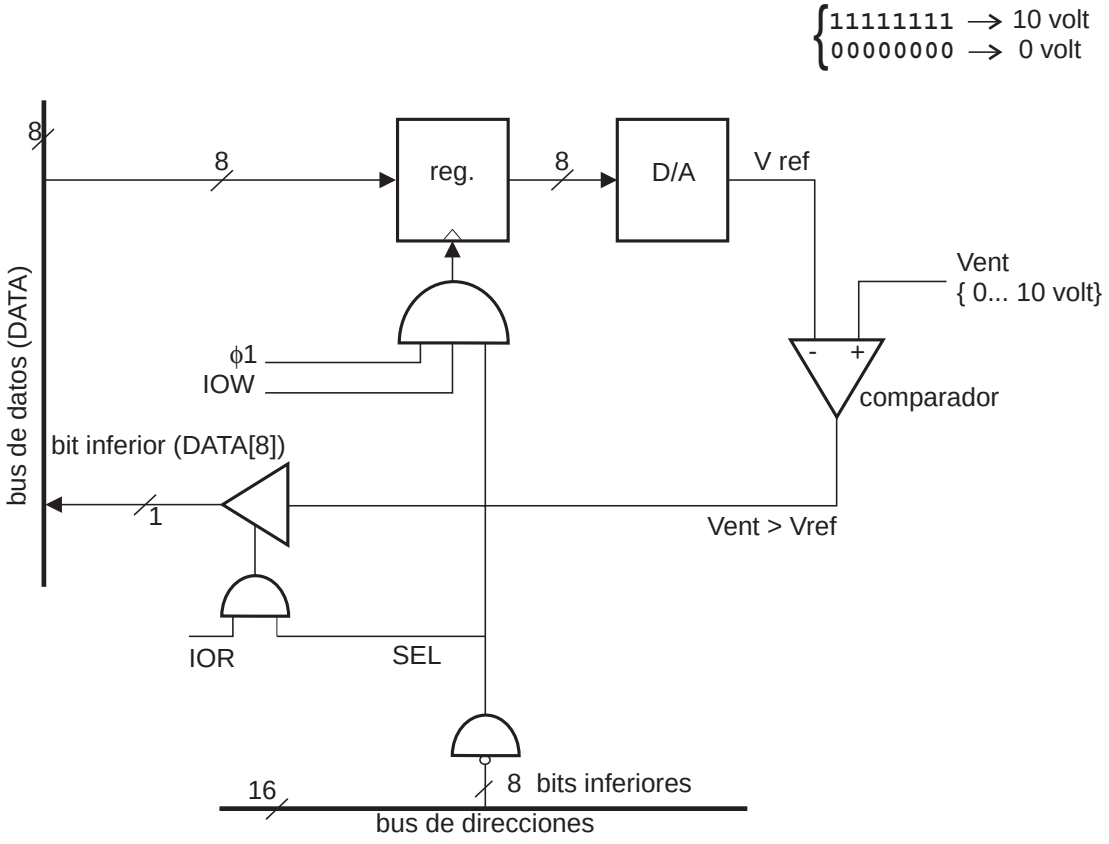


Figura 20.- Periférico. Conversor de tensión.

Al escribir, instrucciones OUT se carga un dato en el registro. Este dato se convierte a una tensión analógica con la codificación:

00→0 voltios FF→10 voltios

Esta tensión se compara con una entrada externa Vent, generándose un valor lógico que dice si Vent > Vref.

Al ejecutar instrucciones IN de lectura de la puerta 0, esta señal pasa al bit menos significativo del bus de datos. El resto del bus puede contener cualquier cosa.

Con este periférico se quiere leer una tensión Vent. Es un caso típico de toma de muestras en procesos industriales.

9 LA PRÁCTICA

El juego de instrucciones a desarrollar por el alumno se proporciona en documento aparte. En él se incluye también otra información importante sobre el enunciado de la práctica.

10 TRABAJO A REALIZAR

Consiste en microprogramar el juego de instrucciones planteado en el apartado anterior y comprobar su funcionamiento mediante un programa de prueba.

En primer lugar se microprogramarán las instrucciones solicitadas a **nivel simbólico** (véase el ejemplo de la figura 18), también llamado **nivel RT** (**R**egister **T**ransfer), incluyendo todos los comentarios que sean necesarios para facilitar la comprensión del código.

Los microprogramas realizados se traducirán manualmente a micropalabras, con ayuda de la tabla de la figura 6 (véase la figura 19). A continuación, se pasarán por el simulador del P8080E (**P8080E.EXE**), residente en los PC's del Centro de Cálculo de la Facultad. Este simulador genera opcionalmente un listado de salida con dos partes:

```
mpc   SR  A AC  T DI  I R SZVA-PNC DR  UV   WZ   BC   DE   HL   SP   PC   ADDR DT  MR MW I R I W RY HT  Vent     Vr ef

001 000 00 00 00 00 00 00000000 00 0000 0000 0000 0000 0000 0000 0000 0000 FF  0  0  0  0  1  0  2.100  0.000
002 000 00 00 00 00 00 00000000 00 0000 0000 0000 0000 0000 0000 0000 0000 FF  0  0  0  0  1  0  2.100  0.000
003 000 00 00 00 00 00 00000000 00 0000 0000 0000 0000 0000 0000 0001 0000 FF  1  0  0  0  0  0  2.100  0.000
004 000 00 00 00 00 FF 00000000 00 0000 0000 0000 0000 0000 0000 0001 0000 3D  1  0  0  0  0  0  2.100  0.000
004 000 00 00 00 00 3D 00000000 00 0000 0000 0000 0000 0000 0000 0001 0000 21  1  0  0  0  1  0  2.100  0.000
004 000 00 00 00 00 21 00000000 00 0000 0000 0000 0000 0000 0000 0001 0000 21  0  0  0  0  1  0  2.100  0.000
005 000 00 00 00 00 21 00000000 00 0000 0000 0000 0000 0000 0000 0001 0000 FF  0  0  0  0  1  0  2.100  0.000

108 000 00 00 00 00 21 00000000 00 0000 0000 0000 0000 0000 0000 0002 0001 FF  1  0  0  0  0  0  2.100  0.000
109 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 00FF 0000 0002 0001 36  1  0  0  0  0  0  2.100  0.000
109 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 0036 0000 0002 0001 10  1  0  0  0  1  0  2.100  0.000
109 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 0010 0000 0002 0001 10  0  0  0  0  1  0  2.100  0.000
10A 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 0010 0000 0003 0002 FF  1  0  0  0  0  0  2.100  0.000
10B 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 FF10 0000 0003 0002 FF  1  0  0  0  0  0  2.100  0.000
10B 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 FF10 0000 0003 0002 30  1  0  0  0  0  0  2.100  0.000
10B 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 3010 0000 0003 0002 00  1  0  0  0  1  0  2.100  0.000
10B 000 00 00 00 21 21 00000000 00 0000 0000 0000 0000 0010 0000 0003 0002 00  0  0  0  0  1  0  2.100  0.000
10C 000 00 00 00 00 21 00000000 00 0000 0000 0000 0000 0010 0000 0003 0002 FF  0  0  0  0  1  0  2.100  0.000

003 000 00 00 00 00 21 00000000 00 0000 0000 0000 0000 0010 0000 0004 0003 FF  1  0  0  0  0  0  2.100  0.000

mpc   SR  A AC  T DI  I R SZVA-PNC DR  UV   WZ   BC   DE   HL   SP   PC   ADDR DT  MR MW I R I W RY HT  Vent     Vr ef

004 000 00 00 00 00 FF 00000000 00 0000 0000 0000 0000 0010 0000 0004 0003 7F  1  0  0  0  0  0  2.100  0.000
004 000 00 00 00 00 7F 00000000 00 0000 0000 0000 0000 0010 0000 0004 0003 7E  1  0  0  0  1  0  2.100  0.000
004 000 00 00 00 00 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0003 7E  0  0  0  0  1  0  2.100  0.000
005 000 00 00 00 00 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0003 FF  0  0  0  0  1  0  2.100  0.000

3F0 000 00 00 00 00 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0010 FF  1  0  0  0  0  0  2.100  0.000
3F1 000 FF 00 00 7E 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0010 FF  1  0  0  0  0  0  2.100  0.000
3F1 000 FF 00 00 7E 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0010 DB  1  0  0  0  0  0  2.100  0.000
3F1 000 DB 00 00 7E 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0010 99  1  0  0  0  1  0  2.100  0.000
3F1 000 99 00 00 7E 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0010 99  0  0  0  0  1  0  2.100  0.000
3F2 000 99 00 00 00 7E 00000000 00 0000 0000 0000 0000 0010 0000 0004 0010 FF  0  0  0  0  1  0  2.100  0.000

003 000 99 00 00 00 7E 00000000 00 0000 0000 0000 0000 0010 0000 0005 0004 FF  1  0  0  0  0  0  2.100  0.000
004 000 99 00 00 00 FF 00000000 00 0000 0000 0000 0000 0010 0000 0005 0004 FF  1  0  0  0  0  0  2.100  0.000
004 000 99 00 00 00 FF 00000000 00 0000 0000 0000 0000 0010 0000 0005 0004 BE  1  0  0  0  0  0  2.100  0.000
004 000 99 00 00 00 BE 00000000 00 0000 0000 0000 0000 0010 0000 0005 0004 3C  1  0  0  0  1  0  2.100  0.000
004 000 99 00 00 00 3C 00000000 00 0000 0000 0000 0000 0010 0000 0005 0004 3C  0  0  0  0  1  0  2.100  0.000
005 000 99 00 00 00 3C 00000000 00 0000 0000 0000 0000 0010 0000 0005 0004 FF  0  0  0  0  1  0  2.100  0.000

1E0 000 9A 00 00 00 3C 10000100 00 0000 0000 0000 0000 0010 0000 0005 0004 FF  0  0  0  0  1  0  2.100  0.000

mpc   SR  A AC  T DI  I R SZVA-PNC DR  UV   WZ   BC   DE   HL   SP   PC   ADDR DT  MR MW I R I W RY HT  Vent     Vr ef

1E1 000 9A 00 00 00 3C 10000100 00 0000 0000 0000 0000 0010 0000 0005 0004 FF  0  0  0  0  1  0  2.100  0.000
003 000 9A 00 00 00 3C 10000100 00 0000 0000 0000 0000 0010 0000 0006 0005 FF  1  0  0  0  0  0  2.100  0.000
004 000 9A 00 00 00 FF 10000100 00 0000 0000 0000 0000 0010 0000 0006 0005 7F  1  0  0  0  0  0  2.100  0.000
004 000 9A 00 00 00 7F 10000100 00 0000 0000 0000 0000 0010 0000 0006 0005 76  1  0  0  0  1  0  2.100  0.000
004 000 9A 00 00 00 76 10000100 00 0000 0000 0000 0000 0010 0000 0006 0005 76  0  0  0  0  1  0  2.100  0.000
005 000 9A 00 00 00 76 10000100 00 0000 0000 0000 0000 0010 0000 0006 0005 FF  0  0  0  0  1  0  2.100  0.000
3B0 000 9A 00 00 00 76 10000100 00 0000 0000 0000 0000 0010 0000 0006 0005 FF  0  0  0  0  1  1  2.100  0.000
```

Figura 21.- Simulación: traza del p8080e.

1.- Copia de las líneas de entrada numeradas.
2.- Traza de ejecución (véase el ejemplo de la figura 21).

Para comprobar el funcionamiento de los microprogramas realizados se debe crear un fichero cuyo contenido tenga el formato que se puede ver en la figura 19, esto es:

- El contenido de la memoria de control: las **microinstrucciones.**
- Una línea con "/" en la primera posición.
- El contenido de la memoria principal: las **instrucciones y los datos.**
- Otra línea con "/" en la primera posición.

Las microinstrucciones son micropalabras para la memoria de control y su formato de entrada es el siguiente:

- 3 dígitos hexadecimales que especifican la dirección de memoria de control.
- 40 bits (1, 0 o X) que son el contenido de esta dirección.

Este último campo puede contener blancos en cualquier posición para facilitar la lectura.

En el ejemplo de la figura 19, aparte de las líneas de comentarios (cuyo primer carácter es "."), se han introducido microinstrucciones en las siguientes direcciones de la memoria de control:

```
001 002                 RESET
003 004 005             FETCH
108 109 10A 10B 10C     LXI H, d16
3F0 3F1 3F2             MOV A, M
1E0 1E1                 INR A
3B0                     HLT
```

La memoria principal contendrá el programa de prueba, realizado con el juego de instrucciones microprogramado, y los datos de entrada. En el ejemplo anterior, el programa está a partir de la dirección 0000H de memoria principal y es:

```
fuente          código máquina

LXI H, 0010H        21 10 00
MOV A, M            7E
INR A               3C
HLT                 76
```

y la posición 0010H contiene el valor 99H (dato de entrada para el programa principal).

El formato del programa de entrada al simulador es el siguiente:

- 4 dígitos hexadecimales que son la dirección de memoria principal.
- 2 dígitos hexadecimales, que son el contenido de esta dirección.
- Comentarios (en este campo se debe escribir el código ensamblador).

El P8080E arranca, según se dijo anteriormente, ejecutando la microinstrucción 001 después de haber inicializado a 00H todos los registros. En esta dirección debe comenzar la microrrutina de RESET que deberá inicializar el contador de programa (PC=0000H).

Seguidamente iría la microrrutina de FETCH (exactamente igual a la del ejemplo de la figura 19). La primera instrucción que se ejecuta estará en la dirección 0 de memoria principal.

Con toda esta información, P8080E empieza a funcionar ciclo a ciclo, generando una línea de traza con cada flanco ascendente de $\phi 1$, justo antes del flanco.

El juego de instrucciones planteado implica un uso muy frecuente de zonas de microcódigo común a varias instrucciones. Para compartir microcódigo hay dos alternativas:

- Saltos incondicionales a la zona de microcódigo común.
- Saltos a microsubrutina (tres niveles como máximo).

El paso de parámetros se realizará mediante los registros transparentes al programador.

10.1 VERSIÓN NO INTERACTIVA DEL SIMULADOR

Además de la versión interactiva del simulador, está disponible otra versión no interactiva que permite limitar el tiempo de ejecución y genera como resultado dos ficheros: 1) la traza de ejecución y 2) el valor de ciertos registros y posiciones de memoria al final de la ejecución. El simulador se invoca del siguiente modo:

P8080ENI.EXE fich.ent fich.sal

donde *fich.ent* es el fichero de entrada que contiene la memoria de control y la memoria principal, y *fich.sal* es la traza de ejecución (NUL si no se desea generar el fichero de traza).

Este simulador acepta cuatro opciones, /v, /m, /a y /w, que permiten controlar su ejecución.

La opción **/v** permite conocer la versión del simulador que se está ejecutando.

La opción **/m "microciclos"** permite especificar el máximo número de microciclos que se deben simular.

La opción **/a "alarma"** permite especificar el tiempo máximo de ejecución en segundos. Si se invoca el simulador con las opciones /m y /a de forma simultánea, sólo se tiene en cuenta la opción /m.

Por ejemplo:

G:\DATSI\P8080E\P8080ENI /a15 mcont_pp.p80 traza

Además de la traza de ejecución, este simulador puede generar un **fichero de resultados** en el que se incluyen los valores que tienen ciertos registros y posiciones de memoria al final de la ejecución. La especificación de qué registros y posiciones de memoria se deben incluir en este fichero de resultados se lee de otro **fichero de configuración** cuya sintaxis se explica en un fichero ejemplo llamado *p8080.cfg* que se encuentra en el directorio *G:\DATSI\P8080E.*

El nombre de ambos ficheros de resultados y de configuración se toma del fichero de entrada, el que contiene la memoria de control, al que se le sustituye su extensión por las extensiones **'.sal'** y **'.cfg'** respectivamente. En el ejemplo anterior el nombre de estos ficheros sería **mcont_pp.sal** y **mcont_pp.cfg**.

Esta versión no interactiva está también disponible para ejecutar en un computador con sistema operativo **Linux**.

Todos las versiones del simulador, así como un ejemplo del fichero de configuración y una versión del programa de entrega de la práctica para windows 95 con acceso a internet están disponibles a través de la página WEB de la asignatura:

http://datsi.fi.upm.es/docencia/Estructura_09.html

10.2 DURACIÓN DE LOS CICLOS DE ESPERA

Existe una última opción, **/w**, que, si se especifica en la línea de comando mediante la que se invoca el simulador (ya sea el denominado "interactivo" o el "no interactivo"), sirve para controlar el número de

ciclos de espera que se producirá en los accesos a memoria. El empleo de esta opción es muy útil para depurar el microcódigo correspondiente a los accesos a memoria.

Si **no** se incluye la opción /w, el número de ciclos de espera de cada ciclo de memoria será aleatorio, variando entre 0 y 3, por lo que se generarán trazas de ejecución distintas para cada simulación del mismo programa. Si se incluye la opción **/wf**, el número de ciclos de espera será también aleatorio, pero siguiendo una secuencia fija que se repite en cada ejecución del microprograma en el simulador; por lo tanto, varias simulaciones del mismo programa generarán trazas de ejecución idénticas.

Si se incluye la opción /w<n> con n comprendido entre 0 y 9 (ej.: /w6), todos los accesos a memoria tendrán el mismo numero n de ciclos de espera. Mediante esta opción se podrá comprobar el funcionamiento de todos los accesos a memoria en los casos extremos: cuando la memoria responde inmediatamente (/w0) y cuando lo hace empleando un elevado número de ciclos de espera.

11 NORMAS DE ENTREGA DE LA PRÁCTICA

Para evaluar el trabajo realizado por los alumnos en esta práctica de microprogramación, se ejecutará, utilizando la memoria de control proporcionada por los alumnos, una serie de programas de prueba. Para que ello sea posible, cada grupo de alumnos deberá entregar dos ficheros cuyo formato se especifica más adelante.

11.1 FORMATO DE LOS FICHEROS

Se deberán entregarse dos ficheros para permitir la corrección de la práctica:

- **mcont_pp.p80**: Este fichero deberá incluir el contenido de la memoria de control y de la memoria principal tal como se describe en el apartado 10 y en la figura 19. La memoria de control deberá incluir los microprogramas, comentados, de las microrrutinas de reset, fetch y de las instrucciones propuestas. La memoria principal deberá incluir un programa de prueba, que funcione correctamente, con los datos de entrada que sean pertinentes.
- **autores.p80**: Este fichero **se entrega solamente una vez para dar de alta al grupo de autores**. Deberá contener, única y exclusivamente, una línea por cada autor de la práctica, con su identificación. El formato debe ser el siguiente (atención: los signos de puntuación son significativos):

Nº Matrícula; DNI; apellido apellido, nombre;

El número de matrícula que se debe indicar, en el fichero de autores, es el qu**e asigna la secretaría de la Facultad** (por ejemplo 990999) y no el que se utiliza como identificador para abrir cuentas en el Centro de Cálculo (por ejemplo a990999).

11.2 PROGRAMA DE ENTREGA DE LOS FICHEROS DE LA PRÁCTICA

Para entregar los dos ficheros cuyo formato se describe en el apartado anterior, se deberá ejecutar el programa de entrega **"cliente -p p80"** que está disponible en la **red de PCs** y en **zipi.fi.upm.es**. En la red de PCs el programa se encuentra en el directorio "**G:\DATSI\ENTREGAS**", mientras que en **zipi.fi.upm.es** está disponible como si fuera un comando más del sistema operativo.

11.2.1 Programa de entrega para computadores PC-Windows con acceso a Internet

Aquellos alumnos que tengan acceso a un computador personal con Windows 95/98 y conexión Internet, pueden instalar un programa de entrega en dicho computador. Este programa les permitirá realizar las mismas operaciones que el programa **"cliente"** de la red de PCs o de zipi.fi.upm.es y se puede recoger mediante un "navegador web" accediendo a la página de la asignatura:

http://www.datsi.fi.upm.es/docencia/Estructura_09/U_Control/

Una vez obtenido, se ha de desempaquetar con *pkunzip* ó con *winzip*, lo que generará los ficheros mencionados más arriba (más varios .BAT adicionales, para entregar otras prácticas, que no tienen utilidad para la de microprogramación).

11.2.2 Uso del programa de entrega de la práctica

El programa permite, además de entregar los ficheros indicados anteriormente, consultar los resultados de la evaluación de la práctica y noticias que vayan surgiendo durante el periodo de realización de la práctica.

Al entrar en el programa, en primer lugar se identifica la práctica y la convocatoria. El mismo programa se utiliza para entregar prácticas de varias asignaturas del departamento DATSI y, por otro lado, pueden solaparse los periodos de realización de una misma práctica para dos convocatorias, p.e. febrero y junio. Para evitar confusiones, **es importante comprobar que se está usando el programa de entrega adecuado**.

Estructura de Computadores. Practica de microprogramacion (P8080E)
Febrero 2005
Servidor de Practicas. Version 1.9.1

A continuación el programa pide la identificación del usuario. Tomaremos como identificación de usuario el número de matrícula de uno de los integrantes del grupo. El programa mostrará el mensaje:

Introduzca su identificador (Num. matricula): *990999*

El usuario deberá introducir el número de matrícula de uno de los integrantes del grupo (en el ejemplo anterior 990999).

Si es la primera vez que el usuario entra en el sistema de entrega, el programa le invitará a que introduzca una palabra clave (“password”) mostrando el siguiente mensaje:

Se va a establecer password.
Password:

El usuario deberá introducir una palabra clave (no se mostrará en pantalla). Para confirmar que no se ha producido ningún error al introducir el *password*, se vuelve a pedir:

Repita el password tecleado anteriormente:

Si se ha producido algún error se mostrara el mensaje:

Passwords no coinciden

El programa solicita el password hasta tres veces. Si persisten los errores muestra el siguiente mensaje y termina.

Conexión cerrada

Si el comando se ha ejecutado con éxito, se intentará acceder al fichero ‘’autores.p80’’, que se encontrará en el directorio desde el que se está ejecutando el programa de entrega. Si dicho fichero existe y la información que contiene es correcta, se mostrara el mensaje:

MANDANDO EL FICHERO autores.p80 ... OK.

SE HAN DADO DE ALTA LOS SIGUIENTES ALUMNOS:
990999 012345678 Español Español Juan

Se ha asignado password al usuario 990999.

- NO LO OLVIDE
- NO LO APUNTE
- NO LO DIVULGUE

A continuación se mostrara una identificación de la práctica y la convocatoria, junto con el siguiente menú:

OPCIONES:

1. Mandar Ficheros.
2. Consultar Resultados.
3. Cancelar Entregas
4. Bloquear la Entrega
5. Ayuda !!!!
6. Noticias. (05/11/04/ 18:45)
q . Abandonar
>>

Más adelante se explica cada una de las opciones del menú.

Sin embargo, es posible que se detecte algún problema relacionado con el fichero de autores, por ejemplo que no se encuentre en el directorio de trabajo, o que alguno de los números de DNI que en él se encuentran no está registrado en la base de datos de alumnos. En el primero de estos casos, se mostraría el siguiente mensaje:

No se puede abrir el fichero autores.p80
Entrega cancelada.

En el segundo caso (Nº MATRICULA no existente en la base de datos de alumnos), se mostraría el siguiente mensaje:

****
EL Nº MATRICULA 990999 NO ESTA EN LA
LISTA DE LA ASIGNATURA
COMPRUEBE EL Nº MATRICULA O CONSULTE A SU
PROFESOR DE PRACTICAS
****

No se puede poner password

Otros errores que se detectan y señalan al usuario, relacionados con el fichero de autores, son los correspondientes a uno o varios autores cuya descripción es incorrecta (p.e. el DNI contiene caracteres alfabéticos) o que ya están dados de alta en otro grupo de prácticas.

DESCRIPCIÓN DE LAS OPCIONES DEL MENÚ DEL PROGRAMA DE ENTREGA

1. MANDAR FICHEROS

Esta opción permite mandar los ficheros de una práctica, que deberán estar en el directorio de trabajo del usuario. Si el comando se ejecuta correctamente se mostrarán los siguientes mensajes:

MANDANDO EL FICHERO mcont_pp.p80 ...OK.

Si este fichero no se encuentra, el programa lo comunicará al usuario:

MANDANDO EL FICHERO mcont_pp.p80 ...
No se puede abrir el fichero mcont_pp.p80
Entrega cancelada.

El servidor de entregas intenta asegurar que cada uno de los ficheros enviados (en este caso solamente el fichero mcont_pp.p80) tiene el formato correcto. Para ello trata de ejecutar un programa princi-

pal con la memoria de control entregada. Si el simulador del P8080E no puede arrancar la simulación debido a errores de formato, se muestra el siguiente mensaje:

EL FICHERO mcont_pp.p80 NO TIENE EL FORMATO CORRECTO
ENTREGA NO REALIZADA

En este caso el alumno deberá obtener la traza de ejecución que genera su fichero mcont_pp.p80 (mediante la versión no interactiva del simulador o ejecutando *en fichero*). En ella podrá localizar fácilmente mensajes de error correspondientes a errores de formato, duplicación de microinstrucciones, direcciones incorrectas, etc. Una vez corregidos todos los errores y depurada totalmente la práctica, deberá proceder a una nueva entrega.

Cada vez que un alumno realiza **la entrega de una práctica**, **se cancelan todas las entregas pendientes de corrección** correspondientes a dicho alumno. Si dicha entrega es **errónea** por algún motivo, también **se anula esa entrega**.

2. CANCELAR ENTREGAS

Esta opción permite cancelar todas las entregas realizadas correctamente desde la última corrección. El grupo de prácticas será eliminado de la lista de prácticas pendientes de corrección. Si se han realizado varias entregas, correctamente o no, quedarán canceladas todas ellas.

3. CONSULTAR RESULTADOS

Esta opción permite consultar los resultados de la corrección de una práctica entregada. El programa pide el nombre de un fichero en el que se copiarán los resultados de la ejecución del conjunto de *test*s de pruebas que componen el corrector. Se mostrará el siguiente mensaje:

La salida será redirigida a un fichero.

Nombre del fichero (ENTER para salida por pantalla) ?? *result.txt*

En este caso se grabarán los resultados de las pruebas en el fichero **result.txt**. Si como respuesta al mensaje se pulsa la tecla ENTER, los resultados serán mostrados por pantalla.

4. BLOQUEO DE LA ENTREGA

Si el usuario no desea entregar más veces la práctica, por haber superado todas las pruebas, puede bloquear la entrega para mayor seguridad. Para ello deberá seleccionar esta opción, en cuyo caso no se podrá volver a realizar una nueva entrega de los ficheros asociados a la práctica. Si la opción se realiza satisfactoriamente, tras solicitar al usuario confirmación para bloquear la entrega, se mostrará el mensaje:

ENTREGA BLOQUEADA.

5. AYUDA

Esta opción mostrará en pantalla una breve descripción de cada una de las opciones del programa de entrega.

6. NOTICIAS

Esta opción permite notificar al alumno modificaciones en la especificación de la práctica o, en general, noticias de interés de la práctica. En el menú general del programa de entrega aparece la fecha y hora de la última actualización. El programa pide el nombre de fichero en el que se copiarán las noticias.

La salida será redirigida a un fichero.

Nombre del fichero (ENTER para salida por pantalla) ?? *noticias.txt*

En este caso se grabará la información relativa a la asignatura en el fichero **noticias.txt**. Si como respuesta al mensaje se pulsa la tecla ENTER, la información será mostrada por pantalla.

Se ha establecido una clave general de acceso (restringido) al programa de entregas, de forma que el alumno interesado en acceder únicamente al fichero de noticias pueda hacerlo a través de esta clave, incluso si no está dado de alta en ningún grupo de prácticas. Se trata del identificador **p80**, al que se accede con la clave (password) **p80**.

q. ABANDONAR

Termina la ejecución del programa de entrega. Si se realiza con éxito se mostrará el mensaje:

Cerrando la conexión

y a continuación aparecerá el “prompt” del sistema operativo.

NOTA IMPORTANTE: Para evitar la difusión o copia de la práctica, se recomienda encarecidamente trabajar sobre discos flexibles o formatear la unidad de disco D (FORMAT /Q D:) y apagar el ordenador.

Cualquier fichero que no haya sido borrado mediante este procedimiento, u otro semejante, quedará a merced del uso indebido que terceras personas puedan hacer de él. Se recuerda a los alumnos que, en caso de detección de copia, se aplicarán las normas de la asignatura sobre copia de exámenes a todos los alumnos implicados.

Aquellos alumnos que trabajen en zipi o en batman deben proteger sus directorios y ficheros adecuadamente mediante los comandos que ambos sistemas proporcionan.

11.2.3 Procedimiento de entrega vía Web.

Se ha desarrollado una aplicación Web que permite realizar las mismas operaciones que la aplicación descrita en el apartado anterior. La URL en la que se encuentra es:

http://www.datsi.fi.upm.es/Practicas

y es también accesible desde la página de la asignatura:

http://www.datsi.fi.upm.es/docencia/Estructura_09/U_Control/

11.3 INTERPRETACIÓN DEL FICHERO DE RESULTADOS

Para poder interpretar correctamente el fichero de resultados que proporciona el corrector automático de la práctica, se han de tener en cuenta los siguientes aspectos:

- Los resultados de cada prueba se generan una vez que el simulador ha ejecutado el código máquina correspondiente a dicha prueba con la memoria de control proporcionada por el alumno (o grupo).
- La ejecución de dicho código finaliza cuando se cumple una de las siguientes condiciones:
  - Terminación normal: Se lee y ejecuta una instrucción que determine la finalización del programa (HALT)
  - Timeout: Ha transcurrido un tiempo excesivo desde que el programa se puso en funcionamiento, lo que se interpreta (habitualmente es así) que se está ejecutando un bucle infinito de instrucciones o de microinstrucciones, por lo que se cancela la ejecución del programa, independientemente del punto de ejecución en que se encuentre.
  - Microinstrucción no definida: La unidad de control trata de ejecutar una microinstrucción que no está definida en la memoria de control.

- Una vez finalizada la ejecución se obtiene la siguiente información:
  - el contenido de los registros y de algunas posiciones de memoria que han debido quedar con ciertos valores conocidos de antemano
  - las últimas líneas de la traza de ejecución de microinstrucciones
  - la primera diferencia detectada entre dos trazas de ejecución de instrucciones: la obtenida con la memoria de control proporcionada por los alumnos y la considerada como traza oficial de ejecución de instrucciones.
- Se proporciona al alumno el contenido real de los registros y posiciones de memoria involucrados, así como el valor que deberían tener en caso de que haya diferencias. Téngase en cuenta que si el programa se ha detenido por “timeout”, el contenido de la mayoría de esas posiciones y registros no es significativo.

EJEMPLO

```
REGISTRO B: 00
REGISTRO C: 00
REGISTRO D: 00
REGISTRO E: 00
REGISTRO H: 00
REGISTRO L: 00
REGISTRO SP: 1FFC (valor correcto: 0FF8)
REGISTRO PC: 0059 (valor correcto: 005E)
REGISTRO F: 02 (valor correcto: 56)
DIRECCION 0000: 0F (valor correcto: 1F)
DIRECCION 0001: FF
```

Se proporcionan también las últimas líneas de la traza de ejecución de microinstrucciones, debido a que en algunos casos sirven para detectar errores en la microprogramación de las instrucciones. Esto ocurre, por ejemplo, cuando desde una cierta microinstrucción se realiza un salto a una microdirección cuyo contenido no está definido en la memoria de control o cuando una microinstrucción realiza un salto incondicional a sí misma, etc. La información proporcionada por estas últimas líneas de la traza de ejecución de microinstrucciones no siempre es significativa.

EJEMPLO

Últimas lineas de la traza:

```
mpc SR  A  AC T  DI  IR SZVA-PNC DR UV   WZ   BC   DE   HL   SP   PC   ADDR DT MR MW IR IW RY HT Vent  Vref
029 000 00 00 00 05  FF 00000010 01 0000 0114 0000 0000 0000 0FEF 0F02 0F01 4D  1  0  0  0  0  0  4.840 0.000
029 000 00 00 00 05  4D 00000010 01 0000 0114 0000 0000 0000 0FEF 0F02 0F01 00  1  0  0  0  1  0  4.840 0.000
029 000 00 00 00 05  00 00000010 01 0000 0114 0000 0000 0000 0FEF 0F02 0F01 00 0  0  0  0  1  0  4.840 0.000
02A 000 00 00 00 05  00 00000010 01 0000 0114 0000 0000 0000 0FEF 0F02 0F01 FF 0  0  0  0  1  0  4.840 0.000
000 microinstruccion no definida.
```

- Tal como se ha indicado anteriormente, se comparan dos trazas de ejecución de instrucciones: la obtenida con la memoria de control proporcionada por los alumnos y la considerada como oficial. Si existen diferencias entre dichas trazas, se proporciona la primera de ellas, de modo que se pueda identificar qué registros son los que han alcanzado un valor que no es el correcto y cuál es la instrucción en que esto ha ocurrido. Téngase en cuenta que la ejecución del programa no finaliza al detectarse esta diferencia (de hecho la diferencia se detecta *a-posteriori*, al analizar las trazas completas de microinstrucciones), por lo que esta instrucción no necesariamente ha de tener relación alguna con las últimas microinstrucciones proporcionadas. Por otro lado, es posible que, aún siendo idénticas la traza de ejecución de instrucciones oficial y la obtenida con la memoria de control de los alumnos, el resultado de esta prueba no sea satisfactorio: puesto que en la

traza de ejecución de instrucciones se proporciona el contenido de todos los registros, incluido el de Flags, el error estaría producido en este caso por diferencias en algunas posiciones de memoria, lo que indicaría que la implementación de alguna instrucción de escritura sería incorrecta.

EJEMPLO

```
Comparacion de trazas (de instrucciones):
    IR   SZVA-PNC    BC     DE     HL     SP     PC
Alumnos:
016 68| 00000011 | 0000 | 0000 | 0000 | 0508 | 011E NEG.W
017 4C| 01010110 | 0000 | 0000 | 0000 | 0508 | 011F STM #m,[.A2--]
Oficial:
016 68| 00000011 | 0000 | 0000 | 0000 | 0508 | 011E NEG.W
017 4C| 01000010 | 0000 | 0000 | 0000 | 0508 | 011F STM #m,[.A2--]
Diferencias en: FLAGS.
```

Obsérvese en este ejemplo que la primera línea de las trazas de instrucciones encabezadas como “Alumnos” y “Oficial” son idénticas, mientras que las diferencias (en este caso en los Flags) se encuentran en la segunda línea, que es la que contiene el estado en que quedan los registros tras ejecutar la instrucción de la primera línea (en este caso NEG.W).

11.4 SERVICIOS DE ENSAMBLADO Y EXTRACCION DE TRAZAS

Existe un servicio que permite ensamblar programas de prueba y extraer la traza de instrucciones que se han ejecutado en una simulación. Para usarlo se debe indicar un fichero de entrada. Dependiendo del contenido de dicho fichero se realizará un servicio u otro.

Si el fichero de entrada contiene en su primera línea un caracter "#", se supondrá que el fichero contiene un programa de prueba escrito en ensamblador y usando los mnemónicos del juego de instrucciones correspondiente a la práctica. En tal caso, se ensambla el programa de prueba y se genera como resultado el contenido de la memoria principal que hay que incluir como parte del fichero de entrada, para que el simulador P8080E ejecute dicho programa de prueba.

Si el fichero de entrada es el fichero de salida correspondiente a la ejecución de un programa de prueba en el simulador P8080E, de la traza de microinstrucciones generada por el simulador se extrae la traza de instrucciones que se han ejecutado, incluyendo los valores que los registros visibles por el programador contenían antes de la ejecución de cada instrucción. Dicha traza se genera como resultado del servicio.

El servicio está disponible mediante el gestor de entrega de prácticas de la página web de la asignatura y también, usando el programa de entrega de prácticas del Centro de cálculo. En el primer caso hay que seleccionar "Servicios P8080E" y en el segundo hay que ejecutar el siguiente mandato:

G:\DATSI\ENTREGAS\cliente -p 7015

En ambos casos se deben usar los mismos Usuario y Clave (Identificador y Password) que para usar los sistemas de entrega de la práctica.

La información detallada de estos servicios, puede encontrarse en la página de la práctica.

12 APÉNDICE

A continuación se incluyen algunas figuras de interés para la realización de la práctica. La figura 23 representa el cronograma correspondiente a un ciclo de lectura detallado. Las figuras 24 y 25 muestran de forma compacta la estructura interna de la máquina simulada. Por último, se proporciona una hoja que, a modo de resumen, contiene las tablas más importantes para realizar la codificación.

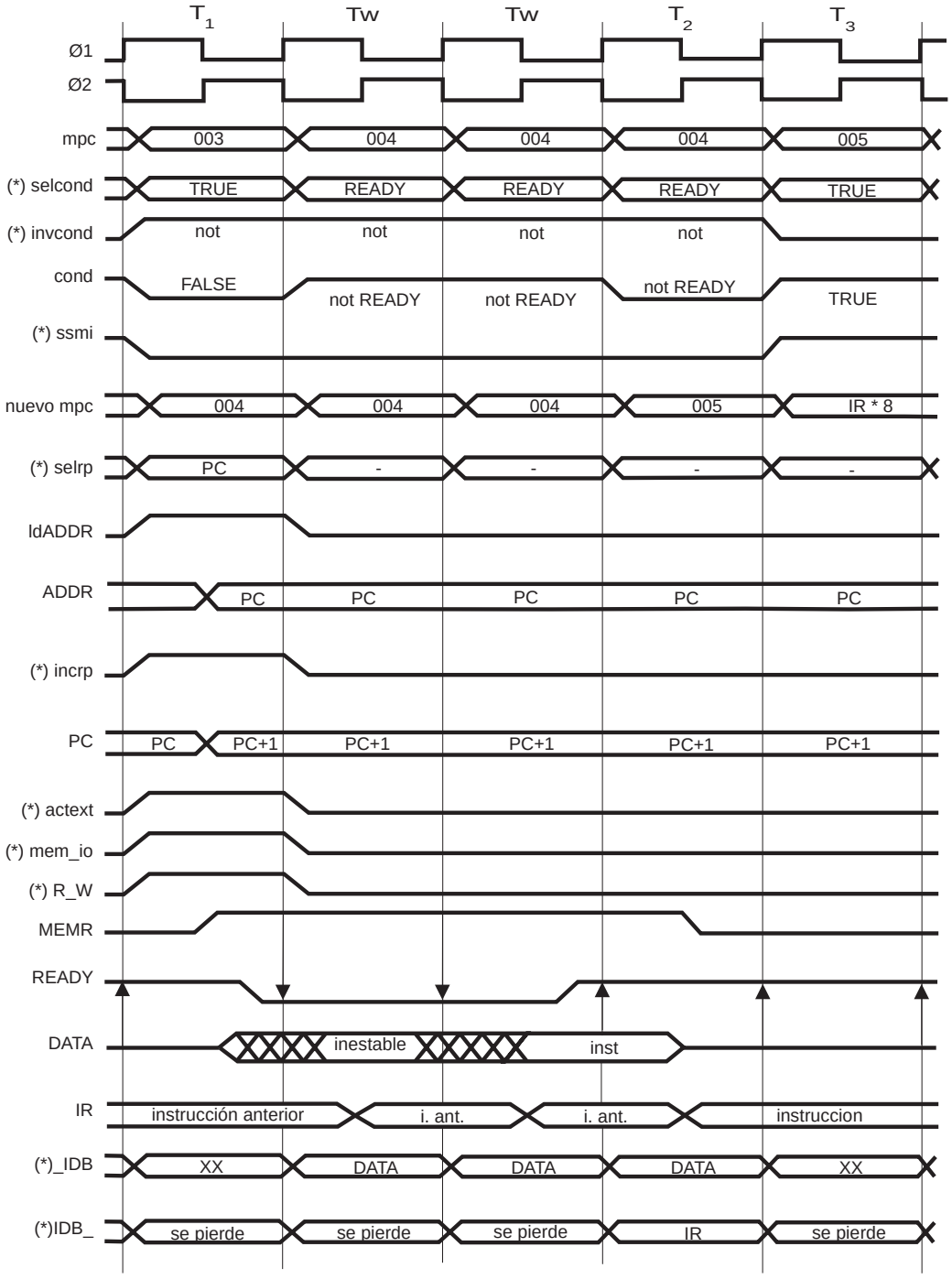


Figura 22.- Ciclo de lectura detallado.

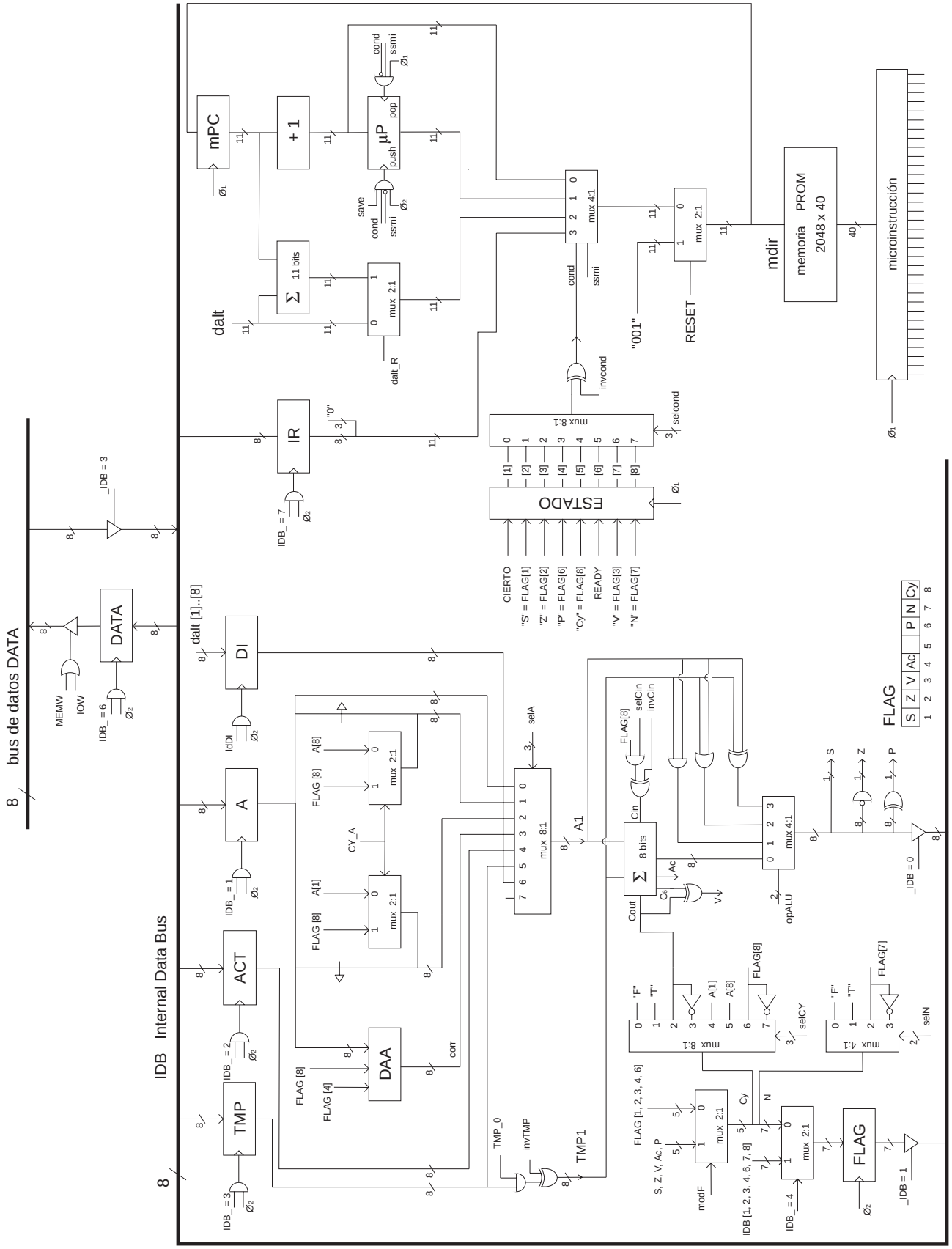
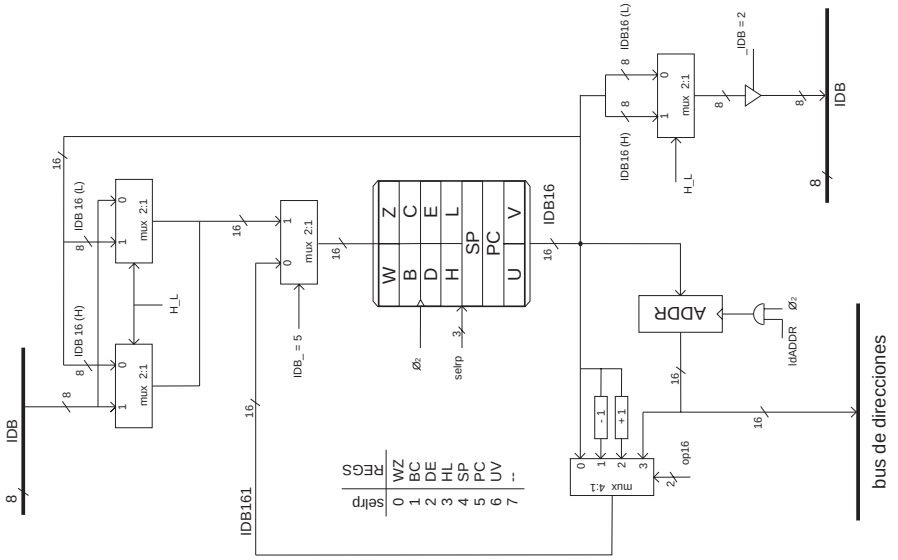


Figura 23.- P8080E (1 de 2).



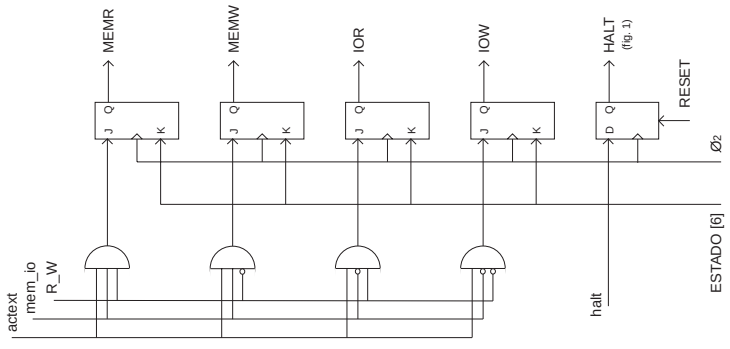


Figura 24.- P8080E (2 de 2).

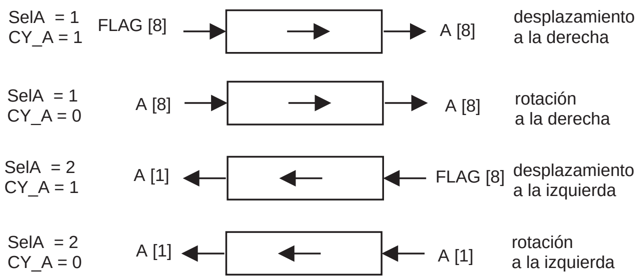
bit nº	campo	defecto
1	selA (dalt)	X
2	selA (dalt)	X
3	selA (dalt)	X
4	CY_A (dalt)	X
5	TMP_0 (dalt)	X
6	invTMP (dalt)	0
7	selCin (dalt)	X
8	invCin (dalt)	0
9	opALU (dalt)	X
10	opALU (dalt)	X
11	ldDI (dalt)	0
12	selCY	1
13	selCY	1
14	selCY	0
15	modF	0
16	_IDB	X
17	_IDB	X
18	IDB_	0
19	IDB_	0
20	IDB_	0
21	selrp	1
22	selrp	1
23	selrp	1
24	op16	0
25	op16	0
26	H_L	X
27	ldADDR	0
28	selN	1
29	selN	0
30	dalt_R	0
31	selcond	0
32	selcond	0
33	selcond	0
34	invcond	1
35	ssmi	0
36	save	0
37	actext	0
38	mem_IO	X
39	R_W	X
40	HALT	0

SelA	A1
0	A
1	A/2
2	A*2
3	DAA
4	ACT
5	TMP
6	DI
7	-

TMP_0	invTMP	TMP1	
0	0	00000000 = cero	
0	1	11111111 = -1 (comp. a 2)	
1	0	TMP	para sumar
1	1	**not** TMP	para restar

selCin	invCin	Cin	
0	0	0	para ADD
0	1	1	para SUB
1	0	CY	para ADC
1	1	**not** CY	para SBB

opALU	ALU	
0	A1 + TMP1 + Cin	sumas y restas
1	A1 and TMP1	
2	A1 or TMP1	
3	A1 xor TMP1	



selCY	CY	
0	F	puesta a 0
1	T	puesta a 1
2	C7	acarreo de salida del sumador
3	not C7	idem negado, para restas
4	A[1]	bit más a la izquierda del acumulador A
5	A[8]	bit más a la derecha del acumulador A
6	FLAG[8]	antiguo CY
7	not FLAG[8]	antiguo CY negado

_IDB	fuente
0	ALU
1	FLAG
2	REGS
3	DATA

IDB_	destino
0	a ninguna parte
1	A
2	ACT
3	TMP
4	FLAG
5	REGS
6	DATA
7	IR

selrp	REGS
0	WZ
1	BC
2	DE
3	HL
4	SP
5	PC
6	UV
7	--

selcond		
0	TRUE	incondicional
1	FLAG[1]	signo (S)
2	FLAG[2]	cero (Z)
3	FLAG[6]	paridad (P)
4	FLAG[8]	acarreo (CY)
5	READY	señal exterior
6	FLAG[3]	overflow (V)
7	FLAG[7]	nivel de ejecución (N)

op16	IDB16	
0	se conserva el contenido de rp	
1	se decrementa en 1	- decrp
2	se incrementa en 1	- incrp
3	se toma el contenido del registro ADDR	

cond	ssmi	microdirección
0	0	la siguiente, mpc+1
0	1	la reservada en el registro μP(RET)
1	0	la indicada en el campo dalt (bits 1 .. 11)
1	1	IR * 8

selN		
0	F	Puesta a 0
1	T	Puesta a 1
2	Flag [7]	Antiguo N
3	**not** Flag[7]	Antiguo N negado

actext	mem_io	R_W	se activa:
1	1	1	MEMR
1	1	0	MEMW
1	0	1	IOR
1	0	0	IOW
0	x	x	ninguna