

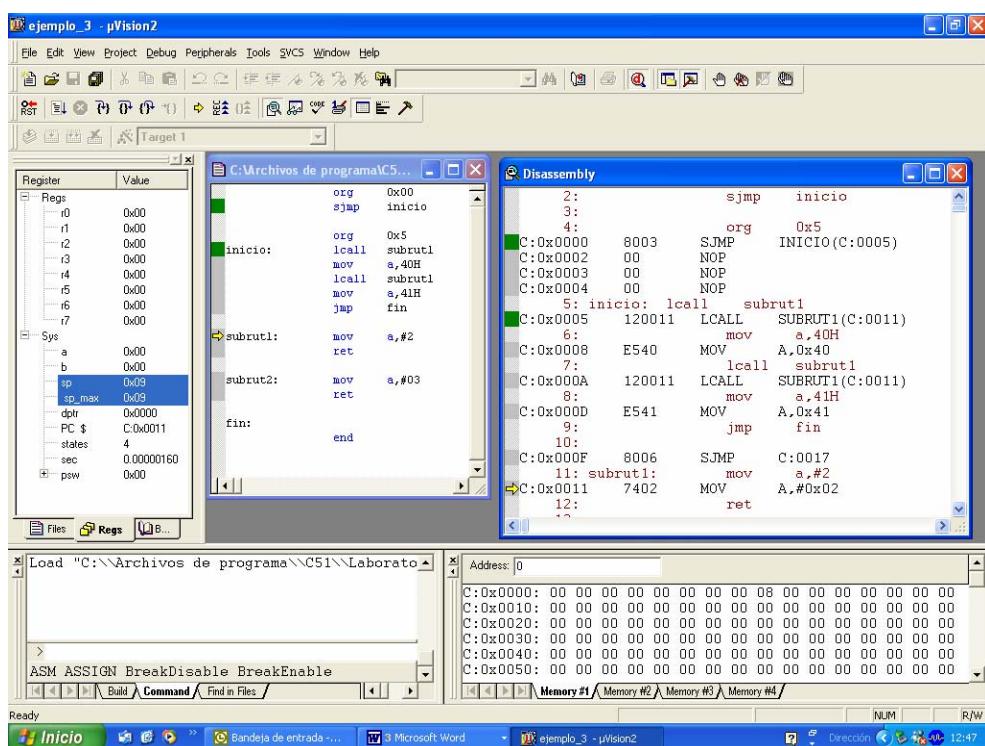


Ejercicios de Laboratorio

de la asignatura

Estructura de Computadores II

Versión 1.1



SEGUNDO CURSO DE INGENIERIA TÉCNICA INDUSTRIAL
ESPECIALIDAD EN
INFORMÁTICA DE GESTIÓN



E.U.I.T.I. de Bilbao
Universidad del País Vasco / Euskal Herriko Unibertsitatea



Dpto. Electrónica y Telecomunicaciones

Autor: Guillermo Bosque Perez

INDICE:

Introducción	2
Guía de Prácticas EC_II	3
Introducción al Entorno de Desarrollo Keil	4
Ejercicio N° 1	6
Ejercicio N° 2	7
Ejercicio N° 3	8
Ejercicio N° 4	9
Ejercicio N° 5	10
Apéndice 1. Diagrama de Bloques de un µC 8xC552.....	11
Apéndice 2. Registros de Funciones Especiales SFR	13
Apéndice 3. Juego de Instrucciones del µC 8051	16
Apéndice 4. Exámenes de Laboratorio Realizados hasta la Fecha de Publicación	48

Introducción

La asignatura Estructura de Computadores II, en su apartado teórico, presenta una gran heterogeneidad de temas, tal diversidad no puede llegar a plantearse en el laboratorio y por ello nos centraremos en uno de los temas para hacer especial hincapié en aspectos que, el alumno de Informática Técnica de Gestión, no volverá a encontrar en el resto de asignaturas. Este tema es el relacionado con el Lenguaje Ensamblador y el Ensamblador.

Las asignaturas EC I y EC II son las únicas que acercan al alumno a aspectos relacionados con el lenguaje próximo a la máquina (lenguajes de bajo nivel). Consideramos que es de gran importancia que el alumno perciba y “visualice” como se mueve la información a nivel del Procesador, Memorias y Periféricos ya que, cuando estos procesos se realizan en lenguajes de alto nivel, queda borroso como se realiza el flujo de datos e instrucciones.

La asignatura EC I realiza este acercamiento a nivel de Laboratorio y la asignatura EC II lo realiza a través del tema **LENGUAJE MÁQUINA Y ENSAMBLADOR** y de diversos ejercicios planteados en el Laboratorio.

Los ejercicios se plantean con una dificultad gradual, acercándonos a instrucciones que nos permitan conocer y manejar mejor la arquitectura del procesador. Este acercamiento gradual permite acometer un último ejercicio de una cierta complejidad funcionalidad.

Para el desarrollo de esta actividad nos apoyaremos en una herramienta de desarrollo software de amplia implantación industrial como es el entorno de desarrollo Keil. Este entorno de desarrollo está disponible en el servidor del departamento en su versión libre de estudiante.

La primera sesión, y como clase magistral, consistirá en una introducción al entorno de desarrollo Keil, para a continuación, describir la arquitectura del microcontrolador escogido para las prácticas.

Las sesiones posteriores serán en Laboratorio, donde continuaremos con el desarrollo de los cinco ejercicios propuestos. Al final de las sesiones dedicadas a la realización de los ejercicios, se dedicarán dos sesiones de laboratorio a evaluar el trabajo realizado durante el curso.

Dado el número de alumnos matriculados en esta asignatura, se realizarán tres grandes grupos (1-2-3) y estos se subdividirán en 10 subgrupos (1...-10) formados por dos o tres personas. Las sesiones en el Laboratorio se plantearán por rotación de grupos (1→2→3→1...) de 1h30' de duración. En cada puesto de trabajo (PC) se ubicará un subgrupo de manera fija. Teniendo en cuenta que el Laboratorio presenta una duración de tres horas, en cada jornada entrarán dos grupos, uno primero y otro al finalizar la sesión del anterior.

La bibliografía recomendada es la siguiente:

- M. Barrón, J. Martínez. *“Aplicaciones prácticas con el μC-8051”*. Ed. M. Barron - McGraw-Hill. 1999.

En este cuaderno se suministra el material de consulta necesario para la realización de las prácticas que se completará con las anotaciones del alumno. Este material ha sido extraído de la referencia anteriormente citada. Además se citan páginas de Philips y Keil donde podrá el alumno encontrar este material en inglés.

Como apunte final precisaremos que este cuaderno no substituye a las explicaciones, comentarios, matices aportados por el profesor en todas y cada una de las sesiones de Laboratorio. Es labor del alumno el completar esta documentación con toda suerte de comentarios y aclaraciones.

Guía de Prácticas EC II

1. INTRODUCCIÓN AL ENTORNO DE DESARROLLO KEIL

- a. Programa en el servidor del departamento en la escuela (<http://det.bp.ehu.es>), en Programas, Laboratorio EC2
- b. Crear CD
- c. Creación de un Proyecto

2. INTRODUCCIÓN AL MICROCONTROLADOR 80C552

- a. Bibliografía básica. Libro de biblioteca
- b. Estructura interna

3. REPERTORIO DE INSTRUCCIONES DEL µC80C552

- a. Copia del repertorio en este cuaderno

4. RELACIÓN DE EJERCICIOS

a. Ejercicio Nº 1:

- Creación de un proyecto concreto
- Ejemplo de programa
- Ejecución

b. Ejercicio Nº 2:

- Mapa de memoria de la RAM interna
- Acceso Directo e Indirecto (Bancos BK0 – BK3) a la RAM interna
- **Ejercicio 2:** Suma de dos valores en RAM
 - Almacenamiento de dos valores en la RAM interna alta
 - Sumar los dos valores y dejar el resultado en la RAM baja
 - Llevar el resultado a la RAM alta

c. Ejercicio Nº 3:

- Registros PSW - DPTR
- Actuación sobre bits – Actuación sobre bits del acumulador A
- Acceso a puertos P1[], P4[] y P5[]
- **Ejercicio 3.1:** Mover bits
 - Mover bits de un área de memoria (20H – 2FH)
- **Ejercicio 3.2:** Actuar sobre bits de P1
 - Actuar sobre bits de P1 y leer su estado
- **Ejercicio 3.3:** Trabajar con R0 de todos los bancos (amplía el ejercicio 2)
 - R0 de BK0, R0 de BK1, R0 de BK2 y R0 de BK3
 - Cargar direcciones (80H .. 83H) y diferentes valores en las direcciones

d. Ejercicio Nº 4:

- **Ejercicio 4.1:** Rotación de bits (mover bits)
 - Cambiar 55H por AAH con dos instrucciones, utilizándolas cuantas veces sea necesario.
- **Ejercicio 4.2:** Trabajo con cuatro bancos (amplía el ejercicio 3.3)
 - Trabajar con cuatro áreas de memoria (A, B, C, D) estando su dirección de origen almacenada en el registro R0 de los bancos BK0, BK1, BK2 y BK3. Sumar A+B, dejando el resultado en C. Dividir por 2 el resultado y dejarlo en D.

e. Ejercicio Nº 5:

- Tratamiento de las subrutinas (LCALL - ACALL)
- **Ejercicio 5.1:** Tratamiento simple de LCALL
 - Realizar una operación de una llamada a subrutina
- **Ejercicio 5.2:** Tratamiento complejo de LCALL
 - Control de nivel de un depósito de agua

5. EVALUACIÓN DE TRABAJOS

Se dedicarán dos sesiones de laboratorio a evaluar el trabajo realizado durante el curso.

Introducción al Entorno de Desarrollo Keil

1. Lanzamiento del Programa

Se pincha dos veces el ícono “Keil uVision”, en este momento se lanza el entorno de desarrollo.

2. Creación de un nuevo proyecto

- Sobre el Menú de la barra superior: *Project → New Project*
- En la opción “*Guardar en:*” buscar en “c:” la carpeta “Mis Documentos”
- Crear la carpeta “*Laboratorio_EC_II*”
- Dentro de la anterior carpeta, crear la carpeta “*Grupo_x*” (1,2 o 3)
- Nombre del proyecto, p.ej., “*Ejercicio_I*” con la extensión “*uv2*”
- Validar

3. Selección de Microprocesador

- Al validar el nombre del proyecto, nos aparecerá una ventana de selección de microprocesador. Seleccionar “*+Philips*”
- Pinchar el símbolo “+” y de la lista de micros seleccionar “*80C552*”. Leer la leyenda que aparece a la derecha del microcontrolador escogido.
- Validar

4. Opciones del Proyecto

- Observamos que se ha creado una carpeta denominada “*Tarjet I*”
- Situamos el ratón sobre esta carpeta y pulsamos el botón derecho del ratón (también podemos ir al menú “*Project*” → “*Options*”)
- Se despliega un menú “*Options for Tarject ‘Tarject I’*”
- Tarjet: Dejar sin modificar
- Output: “*Select Folder for Objects*” : “*Grupo_x*”
- Listing: Idem
- Debug: Aparecen dos pantallas

Pantalla izquierda

- Use Simulator

Pantalla derecha

- Use Keil Monitor
botón “Settings”
- COM1
- 19200 baudios

5. Crear Programa Fuente

- Hay que crear y añadir un fichero al “*Tarjet I*”
- Nos situamos en el menú superior “*File*” → “*New*” → “*Text1*”: → opción “*Save As*” lo renombramos, p.ej. “*ejercicio_I.asm*” (se incorpora la extensión “*asm*” de ensamblador)
- Tarjet I*
|-- *Source Group I*
- Posicionamos el ratón sobre esta última carpeta y pulsamos el botón derecho
- Escogemos la opción “*Add Files to Group*”
- Aparece una pantalla en la que escogeremos las extensiones “*Asm Source Files (*.a*)*”, que son las de lenguaje ensamblador
- Con la opción “*Add*” añadimos “*ejercicio_I.asm*”

6. Escritura del Programa

- Se escribirá el programa en el ensamblador del 8051

b. Comienzo ORG 0x00

.....

.....

END

- Se escribirá tabulado

ETIQUETA: CÓDIGO OPERANDOS ;COMENTARIOS

7. Ensamblado/Compilación del Programa

- Nos situamos en el menú superior “Project” → “Build Tarject”
- Se realiza el ensamblado (compilación si fuera alto nivel) y nos fijaremos en la ventana inferior donde se obtiene un resumen de los errores y avisos del ensamblado
- Por rapidez este proceso puede lanzarse pinchando el icono relativo a “Build Tarject” ubicado en las líneas de iconos del Keil

8. Depuración del Programa

- Nos podemos situar sobre el menú “Debug” → “Start/Stop ...” pero lo realizaremos con los iconos
- Pinchamos el icono *lupa “d”* (símbolo rojo con una “d” en su interior)
- Aparece una pantalla con registros
 - r0 ... r7
 - a, b, ...
 - etc...

Estos son los registros del 80552

- Ejecutamos el programa pinchando el icono que tiene una flecha entre llaves “{↓}” su función es ejecutar el programa “paso a paso” (“step” o “step into”)
 - Vemos en el programa el desplazamiento de “barras de color” indicando la posición actual
 - Una flecha amarilla nos indica el comienzo del programa
 - Al ir ejecutándose las instrucciones, los registros afectados se resaltan en azul
 - Aparece una pantalla “Dissassembly” que es el programa desensamblado. Como trabajamos en Lenguaje Ensamblador, el código es el conocido (en C se correspondería con el desensamblado de las instrucciones de alto nivel)
 - Es interesante disponer del mapa de memoria RAM cuando le afectan determinadas operaciones. Nos situamos en el menú “View” → “Memory Window”, si la ventana aparece en blanco, introduciremos una dirección, p.ej. D: 0x00 <↓
- Hay que escribir “D:” para indicar que estamos en Memoria de Datos

9. Ejemplo Sencillo

Veamos el siguiente programa

```

ORG      0x00      ; Origen. Dirección en Hexadecimal
SJMP    INICIO    ; Salto corto
ORG      0x20      ; Dirección de comienzo del programa
                ; Las primeras líneas se dedican a “interrupciones”
INICIO:   MOV      A,#0x10  ; Carga en A el valor inmediato 10H
          MOV      0x60,A   ; Carga el contenido de A (10H) en la dir. 60H
          MOV      B,0x60   ; Carga en B el contenido de la dir. 60H (10H)
          END            ; Fin del programa para el Ensamblador

```

Sobre este ejemplo efectuaremos el Ensamblado y Depuración, observando registros y posiciones de memoria. Finalizado el ejercicio **salvaremos** a disquete el programa.

Notas del Alumno:

Ejercicio N° 1

1. Ejercicio: Llevar a cabo la creación del primer proyecto “Ejercicio_1” siguiendo todos los pasos dados en la introducción al entorno de desarrollo. Escribir y depurar el ejercicio propuesto como ejemplo.

Nota: Una vez finalizado el ejercicio, se salvará a disco el archivo generado. Este recordatorio es extensivo al resto de ejercicios y no volverá a ser repetido.

Notas del Alumno:

Ejercicio N° 2

2.1 Aspectos teóricos:

- a. Mostrar Mapas de memoria interna: Acceso Directo – Acceso Indirecto – SFR
- b. Acumulador: Actuación sobre A

2.2 Ejercicio:

- a. Almacenar dos valores (0FH y 11H) en las posiciones de RAM alta 80H y 81H.
- b. Sumar los dos valores y dejar el resultado en la posición de RAM baja 7FH
- c. Llevar el resultado de la posición 7FH a la FFH

Notas del Alumno:

Ejercicio N° 3

3.1 Aspectos teóricos:

- a. Mapas de memoria interna: Acceso Directo – Acceso Indirecto – SFR
- b. Registro PSW: Actuación sobre Bancos de trabajo – Visión en la ventana de depuración
- c. Actuación sobre bits: SETB – CLR. Mapa 20H .. 2FH
- d. Acumulador: Actuación sobre bits de A – Mapa SFR: 0E0H
- e. Acceso a puertos: P1[]: 090H, P4[]: 0C0H, P5[]: 0C4H
- f. Visión de P1 en depuración
- g. Registro DPTR

3.2 Ejercicio:

- a. Mover bits del área de memoria de bits:
Poner a 1 el bit 0 de la 20H, etc..
- b. Actuar sobre bits de la puerta P1 y leer su estado:
Mover 00H a P1, P1 a A, FFH a P1, P1 a A, Borrar P1, Mover P1 a A, Poner a 1 el bit 7 de P1, Leer la puerta P1, Borrar bit 7 de P1, Mover P1 a A.
- c. Trabajar con R0 del BK0, R0 del BK1, R0 del BK2, R0 del BK3:
Mover 55H a la dirección 80H por medio de R0 del BK0
Mover AAH a la dirección 81H por medio de R0 del BK1
Mover 1FH a la dirección 82H por medio de R0 del BK02
Mover F1H a la dirección 83H por medio de R0 del BK3
- d. Sumar el contenido de la dirección 80H (R0-BK0) con el contenido de 81H (R0-BK1),..., 83H (R0-BK3), dejando el resultado en la posición 30H.

Notas del Alumno:

Ejercicio N° 4

4.1 Aspectos teóricos:

- a. Mapas de memoria interna: Acceso Directo – Acceso Indirecto – SFR
- b. Registro PSW: Actuación sobre Bancos de trabajo – Visión en la ventana de depuración
- c. Acumulador: Actuación sobre el A

4.2 Ejercicio:

- a. Área A de n bytes y B de n bytes de Datos. Cargarlas por medio de R0 de BK0 y BK1 respectivamente
- b. Sumar A con B uno a uno todos los valores y dejar el resultado en el área C de n bytes. El área C se controla con R0 del BK2.
- c. Dividir los valores del área C por 2 y llevar el resultado al área D de n bytes. El área D se controla con R0 del BK3.

Nota 1: Las cuatro áreas están en la posición alta de memoria.

Nota 2: Los registros R0 y R1 se utilizarán como apuntadores.

Nota 3: El registro R1 guardará la dirección origen y el R0 la dirección incremental

Nota 4: Las áreas A, B, C y D tendrán un tamaño de 4 bytes. El valor del tamaño (4 bytes) se guardará en R2 del BK0.

Nota 5: Los valores a cargar en A serán 01H, 02H, 10H y 20H. Los valores de B serán 10H, 20H, 11H y 22H.

Nota 6: La carga de valores así como la realización de las distintas operaciones, se ejecutarán por medio de un bucle.

Notas del Alumno:

Ejercicio N° 5

5.1 Aspectos teóricos:

- 5.1 Mapas de memoria interna: Acceso Directo – Acceso Indirecto – SFR
- 5.2 Registro PSW: Actuación sobre Bancos de trabajo – Visión en la ventana de depuración
- 5.3 Acumulador: Actuación sobre el A
- 5.4 Tratamiento de subrutinas: LCALL
- 5.5 Instrucciones condicionales de salto orientadas a Bit y a Byte:
 - a. JB bit, etiqueta ;orientado a bit
 - b. JBC bit, etiqueta
 - c. JNB bit, etiqueta
 - d. DJNZ registro, etiqueta ;orientado a byte
 - e. DJNZ dirección, etiqueta
 - f. CJNE operando1,operando2,etiqueta

5.2 Ejercicio 5.1: Subrutina que devuelve el complemento de una variable

- 5.2.1 De la posición 30H se lee una variable y se calcula su complemento, dejando el valor en el registro A. El valor devuelto en A se lleva a la posición 31H.
- 5.2.2 En la posición 30H cargar el valor 55H.

5.3 Ejercicio 5.2: Regulación del nivel de agua en un depósito

- 5.3.1 Se efectuará una regulación del nivel por medio de dos Detectores de Nivel NB (Nivel Bajo) y NA (Nivel Alto) del depósito y dos Electroválvulas EV1 y EV2 la primera de admisión de agua al depósito y la segunda de distribución.
- 5.3.2 Los Detectores de Nivel NB y NA estarán asociados a las entradas P1.0 y P1.1 respectivamente. Las activaciones de las Electroválvulas EV1 y EV2 estarán asociadas a las salidas P1.4 y P1.5 respectivamente. La detección de las posiciones de las EV1 (PEV1) y EV2 (PEV2) estarán asociados a las entradas P1.2 y P1.3 respectivamente.
- 5.3.3 Estados: si NB entonces “1” si no “0”, si NA entonces “1” si no “0”, si PEV1 o PEV2 abierta “1” si no “0”.
- 5.3.4 Acciones: activar (abrir) EV1 o EV2 por “1”, desactivar (cerrar) por “0”.
- 5.3.5 Acciones:
 - a. Cuando NB entonces activar EV1 y desactivar EV2
 - b. Cuando NA entonces desactivar EV2 y activar EV1
 - c. Cuando no NB y no NA entonces activar EV1 y EV2
 - d. Estudiar casos de error

Nota: El programa se desarrollará con la premisa de que la detección de Nivel y la detección de Posición de electroválvulas sean efectuadas por medio de sendas **subrutinas**. No se admitirán otras soluciones.

Notas del Alumno:

Apéndice 1

Diagrama de Bloques de un μ C 8xC552

NOTA 1: Bloque extraído de la bibliografía recomendada.

NOTA 2: Puede consultarse la siguiente página de Philips donde se encuentra reseñado este diagrama de bloques <http://www.semiconductors.philips.com/pip/P80C552.html>

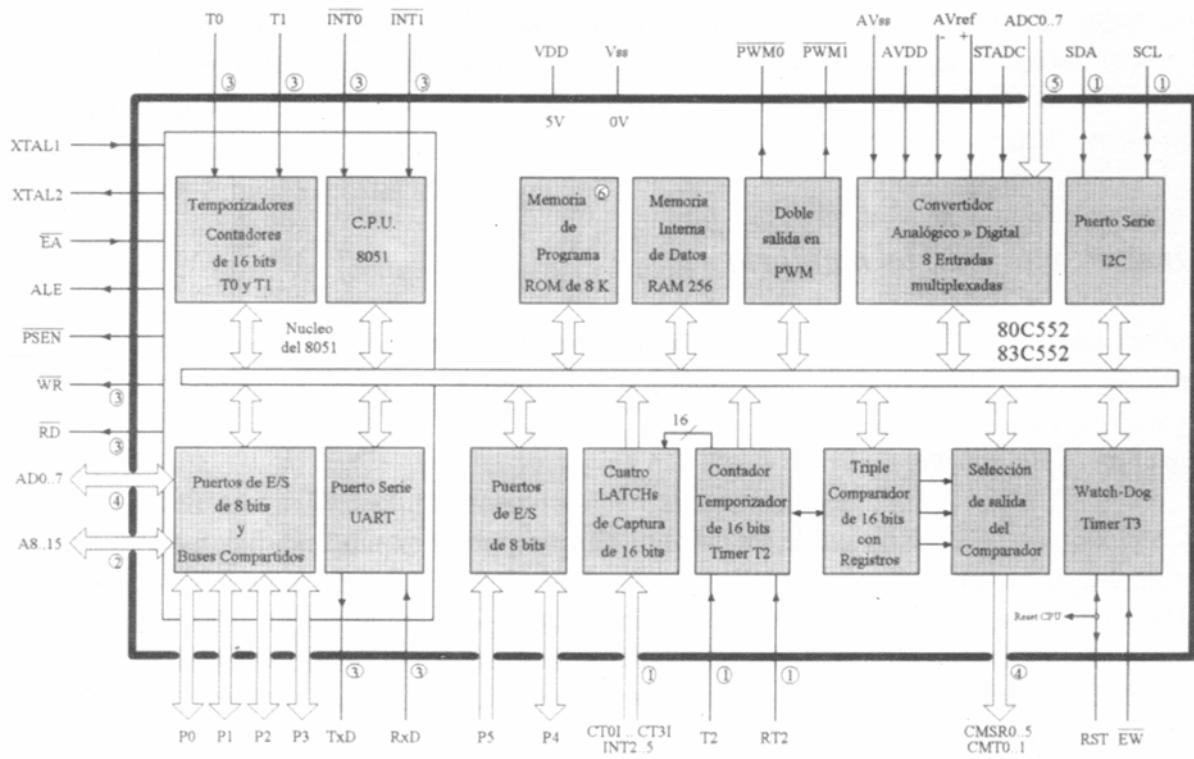


Figura 1. Diagrama de bloques de los μC 8XC552

Apéndice 2

Registros de Funciones Especiales SFR

NOTA 1: Bloque extraído de la bibliografía recomendada.

NOTA 2: Una documentación completa puede extraerse de la página de Philips siguiente

http://www.semiconductors.philips.com/acrobat_download/various/8XC552_562OVERVIEW_2.pdf

SFR. Acceso sólo DIRECTO (80H..FFH)

La tabla 1. contiene todos los registros de funciones especiales del 80C552, que sólo pueden ser leídos o escritos mediante acceso directo. Los SFR con dirección hexadecimal terminada en 0 o en 8, permiten el acceso a nivel bit.

Tabla 1. Registros de funciones especiales del 8XC552.

REGISTRO	DIRECCION DE BIT									RESET	DIRE
T3	Timer del Watchdog									0000.0000	FFH
PWMP	Predvisor del contador del PWM									0000.0000	FEH
PWM1	Registro de comparación del PWM1									0000.0000	FDH
PWM0	Registro de comparación del PWM0									0000.0000	FCH
	†										
IP1 Interrupt Priority 1	FF PT2	FE PCM2	FD PCM1	FC PCM0	FB PCT3	FA PCT2	F9 PCT1	F8 PCT0	0000.0000	F8H	
	†										
B	F7	F6	F5	F4	F3	F2	F1	F0	0000.0000	F0H	
RTE Reset/Toggle Enable	TP47 CM2 & T2	TP46 CM2 & T2	RP45 CM1 & T2	RP44 CM1 & T2	RP43 CM1 & T2	RP42 CM1 & T2	RP41 CM1 & T2	RP40 CM1 & T2	0000.0000	EFH	
STE Set Enable	TG47 -	TG46 -	SP45 CM0 & T2	SP44 CM0 & T2	SP43 CM0 & T2	SP42 CM0 & T2	SP41 CM0 & T2	SP40 CM0 & T2	1100.0000	EEH	
#TMH2	Parte Alta del Timer 2									0000.0000	EDH
#TML2	Parte Baja del Timer 2									0000.0000	ECH
CTCON Capture Control	CTN3 CT3↓	CTP3 CT3↑	CTN2 CT2↓	CTP2 CT2↑	CTN1 CT1↓	CTP1 CT1↑	CTN0 CT0↓	CTP0 CT0↑	0000.0000	EBH	
TM2CON Timer 2 Control	T2IS1 16 Bit Ov.	T2IS0 Byte Ov.	T2ER Ext Rst En.	T2B0 Ov. Int.	T2P1 Fc/1, Fc/2	T2P0 Fc/4, Fc/8	T2MS1 Halt, Osc/12	T2MS0 Pin T2	0000.0000	EAH	
	†										
IEN1 Interrupt Enable 1	ET2 Overflow	ECM2 Compara	ECM1 Compara	ECM0 Compara	ECT3 Captura	ECT2 Captura	ECT1 Captura	ECT0 Captura	0000.0000	E8H	
	†										
ACC	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	0000.0000	E0H	
	†										
SIADR	Registro de Dirección I2C (S1ADR.0=GC)									0000.0000	DBH
SIDAT	Registro de Datos I2C									0000.0000	DAH
#SISTA	Registro de Estado I2C (SC4..0)									1111.1000	D9H
S1CON Control I2C	CR2 Clk Rate	ENS1 Enable S1	STA Start	STO Stop	SI S1 Int.	AA Assert Ack.	CR1 Clk Rate	CR0 Clk Rate	0000.0000	D8H	
	†										
PSW	CY	AC	F0	RS1	RS0	OV	F1	P	0000.0000	D0H	
#CTH3	Parte Alta del Registro de Captura 3									xxxx.xxxx	CFH
#CTH2	Parte Alta del Registro de Captura 2									xxxx.xxxx	CEH
#CTH1	Parte Alta del Registro de Captura 1									xxxx.xxxx	CDH
#CTH0	Parte Alta del Registro de Captura 0									xxxx.xxxx	CCH
CMH2	Parte Alta del Registro de Comparación 2									0000.0000	CBH
CMH1	Parte Alta del Registro de Comparación 1									0000.0000	CAH
CMH0	Parte Alta del Registro de Comparación 0									0000.0000	C9H
TM2IR Timer 2 Interrupt	T2OV 16 Bits Ov.	CM12 CM2 Int.	CM11 CM1 Int.	CM10 CM0 Int.	CTI3 CT3 Int.	CTI2 CT2 Int.	CTI1 CT1 Int.	CTI0 CT0 Int.	0000.0000	C8H	

indica que los registros son únicamente de lectura

Tabla 1. Registros de funciones especiales del 8XC552. Continuación

#ADCH	Parte Alta del Convertidor AD ADC[9..2]								xxxx.xxxx	C6H
ADCON A/D Control	ADC.1 Bit 1	ADC.0 Bit 0	ADEX Enable Start	ADCI ADC Int.	ADCS ADC Start	AADR2 Dir 2	AADR1 Dir 1	AADR0 Dir 0	xx00.0000	C5H
#P5	Puerto con las 8 Entradas Analógicas multiplexadas								xxxx.xxxx	C4H
	†									
P4 Unido con STE y RTE	P4.7 CMT1	P4.6 CMT0	P4.5 CMSR5	P4.4 CMSR4	P4.3 CMSR3	P4.2 CMSR2	P4.1 CMSR1	P4.0 CMSR0	1111.1111	C0H
	†									
IP0 Interrupt Priority 0	-	PAD ADC Int.	PS1 I2C Int.	PS0 UART	PT1 Timer 1	PX1 Ext. 1	PT0 Timer 0	PX0 Ext. 0	x000.0000	B8H
	†									
P3 Multifunction Port	P3.7 RD	P3.6 WR	P3.5 T1	P3.4 T0	P3.3 INT1	P3.2 INT0	P3.1 TxD	P3.0 RxD	1111.1111	B0H
#CTL3	Parte Baja del Registro de Captura 3								xxxx.xxxx	AFH
#CTL2	Parte Baja del Registro de Captura 2								xxxx.xxxx	AEH
#CTL1	Parte Baja del Registro de Captura 1								xxxx.xxxx	ADH
#CTL0	Parte Baja del Registro de Captura 0								xxxx.xxxx	ACH
CML2	Parte Baja del Registro de Comparación 2								0000.0000	ABH
CML1	Parte Baja del Registro de Comparación 1								0000.0000	AAH
CML0	Parte Baja del Registro de Comparación 0								0000.0000	A9H
IEN0 Interrupt Enable 0	EA All	EAD ADC	ES1 I2C	ES0 UART	ET1 Timer 1	EX1 Ext. 0	ET0 Timer 0	EX0 Ext. 0	0000.0000	A8H
	†									
P2 High Address Bus	P2.7 A15	P2.6 A14	P2.5 A13	P2.4 A12	P2.3 A11	P2.2 A10	P2.1 A9	P2.0 A8	1111.1111	A0H
	†									
SOBUF	Serial Buffer								xxxx.xxxx	99H
S0CON Serial Control	SM0 Modo	SM1 Modo	SM2 Modo	REN RxD Enable	TB8 9º bit TxD	RB8 9º bit RxD	TI TxD Int.	RI RxD Int.	0000.0000	98H
	†									
P1 Multifunction Port	P1.7 SDA	P1.6 SCL	P1.5 RT2	P1.4 T2	P1.3 CT3I/INT5	P1.2 CT2I/INT4	P1.1 CT1I/INT3	P1.0 CT0I/INT2	1111.1111	90H
	†									
TH1	Parte Alta del Timer 1								0000.0000	8DH
TH0	Parte Alta del Timer 0								0000.0000	8CH
TL1	Parte Baja del Timer 1								0000.0000	8BH
TL0	Parte Baja del Timer 0								0000.0000	8AH
TMOD Timers Mode	GATE1 Pin Enable	C/T1 Cnt/Tim	M1.1 Modo	M1.0 Modo	GATE0 Pin Enable	C/T0 Cnt/Tim	M0.1 Modo	M0.0 Modo	0000.0000	89H
TCON Timers Control	TF1 Tim 1 Ov.	TR1 Tim 1 Run	TF0 Tim 0 Ov.	TR0 Tim 0 Run	IE1 Enable Ext1	IT1 Nivel / \	IE0 Enable Ext0	IT0 Nivel / \	0000.0000	88H
PCON Power Control	SMOD Doble Baud	-	-	WLE Watchdog	GF1 General 1	GF0 General 0	PD P. Down	IDL Idle	0xx0.0000	87H
	†									
DPH	Parte Alta del DPTR								0000.0000	83H
DPL	Parte Baja del DPTR								0000.0000	82H
SP	Stack Pointer								0000.0111	81H
P0 Data/Low Address Bus	P0.7 D7/A7	P0.6 D6/A6	P0.5 D5/A5	P0.4 D4/A4	P0.3 D3/A3	P0.2 D2/A2	P0.1 D1/A1	P0.0 D0/A0	1111.1111	80H

indica que los registros son únicamente de lectura.

Apéndice 3

Juego de Instrucciones del µC 8051

NOTA 1: Bloque extraído de la bibliografía recomendada.

NOTA 2: El documento original en inglés (p51_pg.pdf) puede ser obtenido de la página de Keil, el enlace es http://www.keil.com/dd/docs/datasheets/philips/p51_pg.pdf

Índice de instrucciones (Paginación respecto al Apéndice)

Tabla de instrucciones que modifican Flags	4
ACALL addr11	4
ADD A,<byte-fuente>	5
ADD A,Rn	5
ADD A,direct	5
ADD A,@Ri	5
ADD A,#data	5
ADDC A,<byte-fuente>	5
ADDC A,Rn	5
ADDC A,direct	5
ADDC A,@Ri	6
ADDC A,#data	6
AJMP addr11	6
ANL <byte-destino>,<byte-fuente>	6
ANL A,Rn	6
ANL A,direct	6
ANL A,@Ri	6
ANL A,#data	7
ANL direct,A	7
ANL direct,#data	7
ANL C,<bit-fuente>	7
ANL C,bit	7
ANL C,/bit	7
CJNE <operando1>,<operando2>,rel	7
CJNE A,direct,rel	8
CJNE A,#data,rel	8
CJNE Rn,#data,rel	8
CJNE @Ri,#data,rel	9
CLR A	9
CLR <bit>	9
CLR C	9
CLR bit	9
CPL A	9
CPL <bit>	10
CPL C	10
CPL bit	10
DA A	10

DEC byte	11
DEC A	11
DEC Rn	11
DEC direct	11
DEC @Ri	11
DIV AB	11
DJNZ <byte>,rel	12
DJNZ Rn,rel	12
DJNZ direct,rel	12
INC <byte>	12
INC A	13
INC Rn	13
INC direct	13
INC @Ri	13
INC DPTR	13
JB bit,rel	13
JBC bit,rel	14
JC rel	14
JMP @A+DPTR	14
JNB bit,rel	15
JNC rel	15
JNZ rel	15
JZ rel	15
LCALL addr16	16
LJMP addr16	16
MOV <byte-destino>,<byte-fuente>	16
MOV A,Rn	16
MOV A,direct	17
MOV A,@Ri	17
MOV A,#data	17
MOV Rn,A	17
MOV Rn,direct	17
MOV Rn,#data	17
MOV direct,A	17
MOV direct,Rn	17
MOV direct_destino,direct_fuente	17
MOV direct,@Ri	18
MOV direct,#data	18
MOV @Ri,A	18
MOV @Ri,direct	18
MOV @Ri,#data	18
MOV <bit-destino>,<bit-fuente>	18
MOV C,bit	18
MOV bit,C	18
MOV DPTR,#data16	19
MOVC A,@A+<base-reg>	19
MOVC A,@A+DPTR	19
MOVC A,@A+PC	19
MOVX <byte-destino>,<byte-fuente>	19

MOVX A,@Ri	20
MOVX A,@DPTR	20
MOVX @Ri,A	20
MOVX @DPTR,A	20
MUL AB	20
NOP	21
ORL <byte-destino>,<byte-fuente>	21
ORL A,Rn	21
ORL A,direct	22
ORL A,@Ri	22
ORL A,#data	22
ORL direct,A	22
ORL direct,#data	22
ORL C,<bit-fuente>	22
ORL C,bit	22
ORL C,/bit	22
POP direct	22
PUSH direct	23
RET	23
RETI	23
RL A	24
RLC A	24
RR A	24
RRC A	24
SETB <bit>	25
SETB C	25
SETB bit	25
SJMP rel	25
SUBB A,<byte-fuente>	25
SUBB A,Rn	26
SUBB A,direct	26
SUBB A,@Ri	26
SUBB A,#data	26
SWAP A	26
XCH A,<byte>	26
XCH A,Rn	27
XCH A,direct	27
XCH A,@Ri	27
XCHD A,@Ri	27
XRL <byte-destino>,<byte-fuente>	27
XRL A,Rn	27
XRL A,direct	28
XRL A,@Ri	28
XRL A,#data	28
XRL direct,A	28
XRL direct,#data	28

NOTA

Rn	Registro R7-R0 del banco actual.
direct	Dirección 0-127 de RAM Interna o un SFR (128-255).
@Ri	Dirección de RAM Interna apuntada por R1-R0.
#data	Dato inmediato de 8 bits.
#data16	Dato inmediato de 16 bits.
#addr16	Dirección de 16 bits (LCALL, LJMP).
#addr11	Dirección de 11 bits (ACALL, AJMP).
rel	Desplazamiento en saltos relativos (desde -128 hasta +127).
bit	Bit de RAM Interna o SFR (Direccionamiento directo).

Tabla de instrucciones que modifican Flags

Instrucción	Flags			Instrucción	Flags		
	C	OV	AC		C	OV	AC
ADD	X	X	X	SETB C	1		
ADDC	X	X	X	CLR C	0		
SUBB	X	X	X	CPL C	X		
MUL AB	0	X		ANL C,bit	X		
DIV AB	0	X		ANL C,/bit	X		
DA A	X			ORL C,bit	X		
RRC	X			ORL C,/bit	X		
RLC	X			MOV C,bit	X		
CJNE	X						

ACALL addr11

Llamada incondicional a un subprograma situado en la dirección indicada. El PC (Program Counter), o Contador de Programa, se incrementa hasta obtener la dirección de la siguiente instrucción a ejecutar, guarda dicha dirección en la PILA (el byte de menor peso primero), e incrementa dos veces el Stack Pointer (SP) o Apuntador de Pila. La dirección de llamada se obtiene encadenando a los 5 bits de mayor peso que ya tiene el PC con los bits 7-5 del código de operación y los 8 bits del segundo byte de la instrucción. De modo que el subprograma llamado debe comenzar dentro del mismo segmento de 2 Kbytes de memoria.

Bytes : 2
 Ciclos : 2
 Código : [A10 A9 A8 1] 0001 [A7 A6 A5 A4 A3 A2 A1 A0]
 Operación : (PC) \leftarrow (PC) + 2
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.7-0)
 (SP) \leftarrow (SP) + 1
 ((SP)) \leftarrow (PC.15-8)
 (PC.10-0) \leftarrow dirección de página

ADD A,<byte-fuente>

Suma el byte implicado al Acumulador, dejando el resultado en el Acumulador. El Carry y el Auxiliar-Carry se activan si se produce llevada en los bits 7 y 3 respectivamente. Cuando se suman enteros sin signo, el *flag* de carry indica que se ha producido un rebosamiento. El *flag* OV (Overflow) se activa si se produce llevada en el bit 7 pero no en el 6, o si ésta se produce en el 6 pero no en el 7. Cuando se suman números enteros con signo, el flag OV indica que se ha producido un resultado negativo al sumar dos operandos positivos, o un resultado positivo con dos sumandos negativos.

Se pueden emplear cuatro modos de direccionamiento: registro, directo, indirecto por registro, inmediato.

ADD A,Rn

Bytes : 1
 Ciclos : 1
 Código : **0010 1rrrr**
 Operación : $(A) \leftarrow (A) + (R_n)$

ADD A,direct

Bytes : 2
 Ciclos : 1
 Código : **0010 0101 direct**
 Operación : $(A) \leftarrow (A) + (\text{direct})$

ADD A,@Ri

Bytes : 1
 Ciclos : 1
 Código : **0010 011i**
 Operación : $(A) \leftarrow (A) + ((R_i))$

ADD A,#data

Bytes : 2
 Ciclos : 1
 Código : **0010 0100 data**
 Operación : $(A) \leftarrow (A) + \#data$

ADDC A,<byte-fuente>

Opera de modo similar a la instrucción ADD, salvo que en ésta además se le suma el valor del Carry.

ADDC A,Rn

Bytes : 1
 Ciclos : 1
 Código : **0011 1rrrr**
 Operación : $(A) \leftarrow (A) + (C) + (R_n)$

ADDC A,direct

Bytes : 2
 Ciclos : 1
 Código : **0011 0101 direct**
 Operación : $(A) \leftarrow (A) + (C) + (\text{direct})$

ADDC A,@Ri

Bytes : 1
 Ciclos : 1
 Código : **0011 011i**
 Operación : $(A) \leftarrow (A) + (C) + ((Ri))$

ADDC A,#data

Bytes : 2
 Ciclos : 1
 Código : **0011 0100** **[data]**
 Operación : $(A) \leftarrow (A) + (C) + \#data$

AJMP addr11

Realiza un salto incondicional a la dirección indicada. La forma en que obtiene la dirección es igual a la empleada por la instrucción ACALL, y del mismo modo, la dirección de salto debe estar dentro del mismo segmento de 2 Kbytes de memoria. Las instrucciones de salto únicamente modifican el contenido del PC (Program Counter).

Bytes : 2
 Ciclos : 2
 Código : **A10 A9 A8 0 | 0 0 0 1** **[A7 A6 A5 A4 A3 A2 A1 A0]**
 Operación : $(PC) \leftarrow (PC) + 2$
 $(PC.10-0) \leftarrow \text{dirección de página}$

ANL <byte-destino>,<byte-fuente>

Realiza la función lógica AND entre las variables indicadas, y almacena el resultado en la variable destino. Entre ambos operandos permiten seis modos de direccionamiento: Si el byte destino es el Acumulador, el byte fuente puede ser: registro, directo, indirecto por registro, o inmediato. Si el destino es una dirección, el byte fuente puede ser: Acumulador, o inmediato. Cuando esta instrucción se emplea para modificar el valor de un puerto, el valor del puerto no se lee de sus patillas, sino que lo hace del *latch* de salida.

ANL A,Rn

Bytes : 1
 Ciclos : 1
 Código : **0101 1rrr**
 Operación : $(A) \leftarrow (A) \text{ AND } (Rn)$

ANL A,direct

Bytes : 2
 Ciclos : 1
 Código : **0101 0101** **[direct]**
 Operación : $(A) \leftarrow (A) \text{ AND } (\text{direct})$

ANL A,@Ri

Bytes : 1
 Ciclos : 1
 Código : **0101 011i**
 Operación : $(A) \leftarrow (A) \text{ AND } ((Ri))$

ANL A,#data

Bytes : 2
 Ciclos : 1
 Código : **0101 0100** [data]
 Operación : (A) \leftarrow (A) AND #data

ANL direct,A

Bytes : 2
 Ciclos : 1
 Código : **0101 0010** [direct]
 Operación : (direct) \leftarrow (direct) AND (A)

ANL direct,#data

Bytes : 3
 Ciclos : 2
 Código : **0101 0011** [direct] [data]
 Operación : (direct) \leftarrow (direct) AND #data

ANL C,<bit-fuente>

Es la función lógica AND a nivel de bit, donde el Carry hace de Acumulador. La barra o slash (/) puede preceder al operador indicando el complemento lógico del bit-fuente, pero este no resulta afectado. Solamente se permiten bit-fuente con direccionamiento directo.

ANL C,bit

Bytes : 2
 Ciclos : 2
 Código : **1000 0010** [bit address]
 Operación : (C) \leftarrow (C) AND (bit)

ANL C,/bit

Bytes : 2
 Ciclos : 2
 Código : **1011 0000** [bit address]
 Operación : (C) \leftarrow (C) AND (NOT(bit))

CJNE <operando1>,<operando2>,rel

Esta instrucción compara la magnitud de ambos operandos y salta si sus valores no son iguales. La dirección de salto se obtiene sumando un desplazamiento (rel). El desplazamiento puede ser positivo o negativo, viene expresado en complemento a dos, y permite realizar saltos con respecto a la dirección de comienzo de la siguiente instrucción, de hasta 128 posiciones hacia atrás, y hasta 127 posiciones hacia adelante. El flag C se activa (desactiva) si el contenido del <byte-destino> es menor (mayor o igual) que el contenido del <byte-fuente>, considerando ambos contenidos como enteros sin signo.

Con los dos primeros operandos se dispone de cuatro modos de direccionamiento: el Acumulador puede ser comparado con el contenido de cualquier posición accesible con direccionamiento directo o con un dato inmediato, y el contenido de cualquier posición de memoria accesible mediante direccionamiento indirecto por registro o registro, puede ser comparada con un dato inmediato.

Ejemplo 1:

```

        CJNE R7, #60H, NO_IGUAL
;           ...      ; R7 = 60H
NO_IGUAL: JC    R7_MENOR          ; SALTA SI R7 < 60H
;           ...      ; R7 > 60H
R7_MENOR:

```

Ejemplo 2:

```
ESPERA: CJNE A, P1, ESPERA
```

Si el dato presente en el puerto 1 es igual al contenido del Acumulador, el programa continúa con la siguiente instrucción, en caso contrario el programa salta a la etiqueta ESPERA, ejecutando la misma instrucción de forma repetida, hasta que los contenidos de A y P1 sean diferentes

CJNE A,direct,rel

Bytes : 3
Ciclos : 2
Código : **1011 0101** [direct] [rel]
Operación : (PC) ← (PC) + 3
SI (A) <> (direct)
ENTONCES
(PC) ← (PC) + relative offset
SI (A) < (direct)
ENTONCES
(C) ← 1
SI (A) >= (direct)
(C) ← 0

CJNE A,#data,rel

Bytes : 3
Ciclos : 2
Código : **1011 0100** [data] [rel]
Operación : (PC) ← (PC) + 3
SI (A) <> #data
ENTONCES
(PC) ← (PC) + relative offset
SI (A) < #data
ENTONCES
(C) ← 1
SI (A) >= #data
(C) ← 0

CJNE Rn,#data,rel

Bytes : 3
Ciclos : 2
Código : **1011 1rrr** [data] [rel]
Operación : (PC) ← (PC) + 3
SI (Rn) <> #data
ENTONCES
(PC) ← (PC) + relative offset
SI (Rn) < #data
ENTONCES
(C) ← 1
SI (Rn) >= #data
(C) ← 0

CJNE @Ri,#data,rel

Bytes : 3
 Ciclos : 2
 Código : **1011 011i data rel**
 Operación : $(PC) \leftarrow (PC) + 3$
 SI $((R_i)) \neq \#data$
 ENTONCES
 $(PC) \leftarrow (PC) + \text{relative offset}$
 SI $((R_i)) < \#data$
 ENTONCES
 $(C) \leftarrow 1$
 SI $((R_i)) \geq \#data$
 $(C) \leftarrow 0$

CLR A

Todos los bits del Acumulador son puestos a cero sin afectar a ningún señalizador (*flag*).

Bytes : 1
 Ciclos : 1
 Código : **1110 0100**
 Operación : $(A) \leftarrow 0$

CLR <bit>

El bit indicado es puesto a cero. Ningún otro *flag* se ve afectado. CLR puede operar con el Carry y con cualquier bit direccionable.

CLR C

Bytes : 1
 Ciclos : 1
 Código : **1100 0011**
 Operación : $(C) \leftarrow 0$

CLR bit

Bytes : 2
 Ciclos : 1
 Código : **1100 0010 bit address**
 Operación : $(\text{bit}) \leftarrow 0$

CPL A

Complementa el contenido del acumulador. Cada bit del acumulador que esté a "1" se pondrá a "0" y al revés. Ningún *flag* queda afectado.

Bytes : 1
 Ciclos : 1
 Código : **1111 0100**
 Operación : $(A) \leftarrow \text{NOT}(A)$

CPL <bit>

Complementa el bit especificado. Puede operar sobre el Carry o sobre cualquier bit direccionable de forma directa. Cuando se usa esta instrucción para modificar una patilla de salida de un puerto, la lectura no se realiza de la patilla, se realiza del *latch* de salida de datos.

CPL C

Bytes : 1
 Ciclos : 1
 Código : **1011 0011**
 Operación : (C) \leftarrow NOT(C)

CPL bit

Bytes : 2
 Ciclos : 1
 Código : **1011 0010** [bit address]
 Operación : (bit) \leftarrow NOT(bit)

DA A

Ajuste a decimal del Acumulador para la suma. DA A ajusta los 8 bits del acumulador tras la suma producida (en formato BCD) al sumar 2 variables con la instrucción ADD o ADDC.

Si el nibble bajo del acumulador contiene un número mayor que 9, o si está a 1 el Auxiliar-Carry, se le suma 6 al acumulador con objeto de obtener el dígito BCD apropiado en el nibble bajo. Esta suma interna puede llegar a activar el Carry si la propagación que se produce hacia el nibble alto produce llevada, pero de ningún modo puede borrar el Carry.

Si el carry está ahora activado, o si el nibble alto supera el valor 9, este nibble alto se incrementa en 6 para obtener el dígito BCD apropiado en el nibble alto. De nuevo es posible que en esta operación interna de adición se active el Carry, indicando que el resultado obtenido es mayor que 100, permitiendo precisión múltiple en la suma decimal. El *flag* de *overflow* OV no se ve afectado.

Todo esto ocurre en un único ciclo de instrucción. En esencia, esta instrucción realiza la conversión decimal sumando 00H, 06H, o 66H al acumulador, dependiendo del estado inicial del acumulador y del PSW (Program Status Word) o Palabra de Estado de Programa.

DA A no sirve para convertir un número en hexadecimal que está en el acumulador a formato BCD. Además no debe emplearse DA A en la resta decimal.

Ejemplo: Supongamos A=56H, R3=67H y bit C=1. Para sumar las cantidades 56 y 67 en BCD con un bit de llevada previa, hay que ejecutar las siguientes instrucciones:

ADDC A, R3
 DA A

El resultado obtenido es 0BEH, y se borran tanto el Carry-Auxiliar como el Carry. El ajuste a decimal de dicho valor da como resultado 24H, que representa al nº 24 en BCD, y el Carry se pone a 1 indicando que el nº es mayor que 100. Efectivamente $56+67+1=124$.

Bytes : 1
 Ciclos : 1
 Código : **1101 0100**
 Operación : El nº en el Acumulador está en BCD.
 SI $[(A3-0) > 9 \text{ OR } [(AC) = 1]]$
 ENTONCES $(A3-0) \leftarrow (A3-0) + 6$
 Y SI $[(A7-4) > 9 \text{ OR } [(C) = 1]]$
 ENTONCES $(A7-4) \leftarrow (A7-4) + 6$

DEC byte

Decrementa en una unidad la variable indicada. Si el valor a decrementar es 00H, entonces el resultado será OFFH. Ningún *flag* se verá afectado. Permite cuatro formas de direccionamiento : acumulador, registro, directo o indirecto por registro. Cuando se utiliza esta instrucción para modificar la salida de un puerto, el valor utilizado para decrementar no se lee de las patillas del puerto, se lee del *latch* de salida.

DEC A

Bytes : 1
 Ciclos : 1
 Código : **0001 0100**
 Operación : $(A) \leftarrow (A) - 1$

DEC Rn

Bytes : 1
 Ciclos : 1
 Código : **0001 1rrr**
 Operación : $(Rn) \leftarrow (Rn) - 1$

DEC direct

Bytes : 2
 Ciclos : 1
 Código : **0001 0101** [direct]
 Operación : $(\text{direct}) \leftarrow (\text{direct}) - 1$

DEC @Ri

Bytes : 1
 Ciclos : 1
 Código : **0001 011i**
 Operación : $((Ri)) \leftarrow ((Ri)) - 1$

DIV AB

Divide un entero sin signo de 8 bits situado en el acumulador, entre un entero sin signo de 8 bits situado en el registro B. Tras efectuarse la división, el acumulador recibirá la parte entera de la división, y el registro B recibirá el resto resultante de la división. Los *flags* Carry y Overflow se borran. Si el registro B tuviera 00H, el resultado obtenido en el acumulador y el registro B sería indeterminado, y se activaría el *flag* OV. El Carry sería igualmente borrado. Ejemplo:

El acumulador tiene 251 (0FBH).

El registro B tiene 18 (12H).

Tras ejecutar DIV AB quedarían con:
 El acumulador : 13 (0DH).
 El registro B : 17 (11H).
 Dado que : $251 = (13 \times 18) + 17$. CY y OV borrados.

Bytes : 1
 Ciclos : 4
 Código : **1000 0100**
 Operación : (A) 15-8 ← Parte Entera de (A) / (B)
 (B) 7-0 ← Resto de (A) / (B)

DJNZ <byte>,rel

Decrement and Jump if Not Zero. Esta instrucción decrementa en una unidad la variable indicada y si el nuevo valor es distinto de cero, se produce un salto a la dirección obtenida al sumar un desplazamiento (rel) que puede ser positivo o negativo (indicado en complemento a dos), a la dirección de comienzo de la siguiente instrucción. En caso de que el nuevo valor de la variable sea cero, no se produce el salto, y se procesa la siguiente instrucción. La variable a decrementar puede ser un registro o una dirección de RAM interna. Si la variable vale cero, al decrementarla pasa a valer OFFH, sin que el flag C se vea afectado por ello. Cuando se utiliza esta instrucción para modificar la salida de un puerto, el valor utilizado para la modificación no se lee de las patillas, se lee del *latch* de salida. Ejemplo:

```
MOV R2,#8      ;Para generar 4 pulsos de reloj
CAMBIA: CPL P1.7 ;por la patilla P1.7
          DJNZ R2,CAMBIA
```

DJNZ Rn,rel

Bytes : 2
 Ciclos : 2
 Código : **1101 1rrr rel**
 Operación : (PC) ← (PC) + 2
 (Rn) ← (Rn) - 1
 SI (Rn) <> 0
 ENTONCES
 (PC) ← (PC) + rel

DJNZ direct,rel

Bytes : 3
 Ciclos : 2
 Código : **1101 0101 direct rel**
 Operación : (PC) ← (PC) + 2
 (direct) ← (direct) - 1
 SI (direct) <> 0
 ENTONCES
 (PC) ← (PC) + rel

INC <byte>

INC incrementa en una unidad la variable indicada. Si esta contiene OFFH, pasará a valer 00H. Ningún flag se verá afectado. Cuando se utiliza esta instrucción para modificar la salida de un puerto, el valor utilizado para incrementar no se lee de las patillas del puerto, se lee del

latch de salida. Los modos de direccionamiento son : registro, directo, o indirecto por registro.

INC A

Bytes : 1
 Ciclos : 1
 Código : **0000 0100**
 Operación : $(A) \leftarrow (A) + 1$

INC Rn

Bytes : 1
 Ciclos : 1
 Código : **0000 1rrr**
 Operación : $(Rn) \leftarrow (Rn) + 1$

INC direct

Bytes : 2
 Ciclos : 1
 Código : **0000 0101 direct**
 Operación : $(direct) \leftarrow (direct) + 1$

INC @Ri

Bytes : 1
 Ciclos : 1
 Código : **0000 011i**
 Operación : $((Ri)) \leftarrow ((Ri)) + 1$

INC DPTR

Incrementa en una unidad el apuntador de 16 bits DPTR. Un desbordamiento en la parte baja DPL (paso de OFFH a 0), hará que la parte alta DPH se incremente en una unidad. Si DPTR contiene 0FFFFH, después de la instrucción "INC DPTR" su contenido será 0. Ningún *flag* resulta afectado.

Bytes : 1
 Ciclos : 2
 Código : **1010 0011**
 Operación : $(DPTR) \leftarrow (DPTR) + 1$

JB bit,rel

Salta si el bit indicado está a 1, de lo contrario continuará la ejecución en la siguiente instrucción. La dirección de salto se obtiene sumando un desplazamiento (rel) que puede ser positivo o negativo (indicado en complemento a dos), a la dirección que tendrá el PC después de incrementar el PC hasta el comienzo de la siguiente instrucción. No se modifican ni los *flags*, ni el bit en cuestión.

Bytes : 3
 Ciclos : 2
 Código : **0010 0000 bit address rel**
 Operación : $(PC) \leftarrow (PC) + 3$
 SI (bit) = 1
 ENTONCES
 $(PC) \leftarrow (PC) + rel$

JBC bit,rel

Si el bit indicado está a 1, lo borra y salta, de lo contrario continuará la ejecución en la siguiente instrucción. La dirección de salto se obtiene sumando un desplazamiento (rel) que puede ser positivo o negativo (indicado en complemento a dos), a la dirección que tendrá el PC después de incrementar el PC hasta el comienzo de la siguiente instrucción. No se modifican los *flags*. Cuando se utiliza esta instrucción para modificar la salida de un puerto, la lectura del valor del bit no se hace de las patillas, sino del *latch* de salida.

Bytes : 3
 Ciclos : 2
 Código : 0001 0000 [bit address] [rel]
 Operación : (PC) \leftarrow (PC) + 3
 SI (bit) = 1
 ENTONCES
 (bit) \leftarrow 0
 (PC) \leftarrow (PC) + rel

JC rel

Salta si el *flag* Carry está activado, de lo contrario continuará la ejecución en la siguiente instrucción. La dirección de salto se obtiene sumando un desplazamiento (rel) que puede ser positivo o negativo (indicado en complemento a dos), a la dirección que tendrá el PC después de incrementar el PC hasta el comienzo de la siguiente instrucción. No se modifican los *flags*.

Bytes : 2
 Ciclos : 2
 Código : 0100 0000 [rel]
 Operación : (PC) \leftarrow (PC) + 3
 SI (C) = 1
 ENTONCES
 (PC) \leftarrow (PC) + rel

JMP @A+DPTR

Salto indirecto. Suma el entero sin signo de 8 bits contenido en el acumulador con los 16 bits del DPTR y coloca su resultado en el PC. Esta instrucción suele utilizarse para bifurcación múltiple. No se modifican ni los *flags*, ni el DPTR, ni el acumulador. Ejemplo : Debe cargarse un nº entre 0, 2, y 4 en el acumulador. La siguiente secuencia de instrucciones, desviará la ejecución del programa a una de las tres etiquetas de la tabla de saltos.

```
MOV DPTR, #TABLA_SALTOS
JMP @A+DPTR
TABLA_SALTOS: AJMP ETIQUETA_1
                AJMP ETIQUETA_2
                AJMP ETIQUETA_3
ETIQUETA_1: ...
ETIQUETA_2: ...
ETIQUETA_3: ...
```

Bytes : 1
 Ciclos : 2
 Código : 0111 0011
 Operación : (PC) \leftarrow (A) + (DPTR)

JNB bit,rel

Salto relativo si el bit no está activado, de lo contrario se continúa con la siguiente instrucción. Ni el bit comprobado, ni los *flags* resultan modificados.

Bytes : 3
 Ciclos : 2
 Código : **0011 0000** [bit address] [rel]
 Operación : $(PC) \leftarrow (PC) + 3$
 SI (bit) = 0
 ENTONCES
 $(PC) \leftarrow (PC) + rel$

JNC rel

Salto relativo si el Carry no está activado, de lo contrario se continúa con la siguiente instrucción. No modifica el bit de Carry.

Bytes : 2
 Ciclos : 2
 Código : **0101 0000** [rel]
 Operación : $(PC) \leftarrow (PC) + 2$
 SI (C) = 0
 ENTONCES
 $(PC) \leftarrow (PC) + rel$

JNZ rel

Salto relativo si alguno de los bits del Acumulador está a uno, de lo contrario se continúa con la siguiente instrucción. No modifica ni el acumulador ni los *flags*.

Bytes : 2
 Ciclos : 2
 Código : **0111 0000** [rel]
 Operación : $(PC) \leftarrow (PC) + 2$
 SI (A) $\neq 0$
 ENTONCES
 $(PC) \leftarrow (PC) + rel$

JZ rel

Salto relativo si todos los bits del Acumulador están a cero, de lo contrario se continúa con la siguiente instrucción. No modifica ni el acumulador ni los *flags*.

Bytes : 2
 Ciclos : 2
 Código : **0110 0000** [rel]
 Operación : $(PC) \leftarrow (PC) + 2$
 SI (A) = 0
 ENTONCES
 $(PC) \leftarrow (PC) + rel$

LCALL addr16

Llamada a un subprograma situado en la dirección indicada de 16 bits. Esta instrucción suma 3 al contenido del PC y lo guarda en la PILA (el byte de menor peso en primer lugar), a continuación se carga la nueva dirección en el PC para que el µControlador ejecute el subprograma hasta encontrar una instrucción de retorno que recupere la dirección guardada en la PILA de nuevo en el PC. La llamada puede realizarse a cualquier dirección de los 64K de memoria de programa. Ningún *flag* se verá afectado.

Bytes	:	3
Ciclos	:	2
Código	:	0001 0010 [addr 15-8] [addr 7-0]
Operación	:	(PC) \leftarrow (PC) + 3 (SP) \leftarrow (SP) + 1 $((SP)) \leftarrow (PC.7-0)$ (SP) \leftarrow (SP) + 1 $((SP)) \leftarrow (PC.15-8)$ (PC) \leftarrow addr16

LJMP addr16

Salto incondicional a la dirección de 16 bits indicada. El salto puede realizarse a cualquier dirección de los 64K de memoria de programa. Ningún *flag* se verá afectado.

Bytes	:	3
Ciclos	:	2
Código	:	0000 0010 [addr 15-8] [addr 7-0]
Operación	:	(PC) \leftarrow addr16

MOV <byte-destino>,<byte-fuente>

Mueve (copia) el contenido del byte fuente al byte destino. El byte fuente no se modifica, ni tampoco los *flags*. Esta es la operación más flexible del microcontrolador, ya que soporta 15 combinaciones diferentes de direccionamiento entre fuente y destino.

Ejemplo: La dirección 30H de RAM interna contiene el valor 40H. La dirección 40H de RAM interna contiene el valor 10H. El dato presente en el Puerto 1 es 0CAH.

```

MOV R0, #30H ;R0  $\leftarrow$  30H
MOV A, @R0 ;A  $\leftarrow$  40H
MOV R1, A ;R1  $\leftarrow$  40H
MOV B, @R1 ;B  $\leftarrow$  10H
MOV @R1, P1 ;RAM (40H)  $\leftarrow$  0CAH
MOV P2, P1 ;P2  $\leftarrow$  0CAH

```

MOV A,Rn

Bytes	:	1
Ciclos	:	1
Código	:	1110 1rrr
Operación	:	(A) \leftarrow (Rn)

MOV A,direct

MOV A,ACC no es una instrucción válida.

Bytes : 2
Ciclos : 1Código : **1110 0101** direct
Operación : (A) ← (direct)***MOV A,@Ri***Bytes : 1
Ciclos : 1Código : **1110 011i**
Operación : (A) ← ((Ri))***MOV A,#data***Bytes : 2
Ciclos : 1Código : **0111 0100** data
Operación : (A) ← #data***MOV Rn,A***Bytes : 1
Ciclos : 1Código : **1111 1rrr**
Operación : (Rn) ← (A)***MOV Rn,direct***Bytes : 2
Ciclos : 2Código : **1010 1rrr** direct
Operación : (Rn) ← (direct)***MOV Rn,#data***Bytes : 2
Ciclos : 1Código : **0111 1rrr** data
Operación : (Rn) ← #data***MOV direct,A***Bytes : 2
Ciclos : 1Código : **1111 0101** direct
Operación : (direct) ← (A)***MOV direct,Rn***Bytes : 2
Ciclos : 2Código : **1000 1rrr** direct
Operación : (direct) ← (Rn)***MOV direct_destino,direct_fuente***Bytes : 3
Ciclos : 2Código : **1000 0101** direcc. fuente direcc. destino
Operación : (direct) ← (direct)

MOV direct,@Ri

Bytes : 2
 Ciclos : 2
 Código : **1000 011i** **direct**
 Operación : **(direct)** \leftarrow **((Ri))**

MOV direct,#data

Bytes : 3
 Ciclos : 2
 Código : **0111 0101** **direct** **data**
 Operación : **(direct)** \leftarrow **#data**

MOV @Ri,A

Bytes : 1
 Ciclos : 1
 Código : **1111 011i**
 Operación : **((Ri))** \leftarrow **(A)**

MOV @Ri,direct

Bytes : 2
 Ciclos : 2
 Código : **1010 011i** **direct**
 Operación : **((Ri))** \leftarrow **(direct)**

MOV @Ri,#data

Bytes : 2
 Ciclos : 1
 Código : **0111 011i** **data**
 Operación : **((Ri))** \leftarrow **#data**

MOV <bit-destino>,<bit-fuente>

Mueve (copia) el contenido del bit fuente hacia el bit destino. El bit fuente no se modifica, ni tampoco los *flags*.

Ejemplo:

```
MOV P1.3,C      ;P1.3  $\leftarrow$  Carry
MOV C,P3.3      ;Carry  $\leftarrow$  P3.3
```

MOV C,bit

Bytes : 2
 Ciclos : 1
 Código : **1010 0010** **bit address**
 Operación : **(C)** \leftarrow **(bit)**

MOV bit,C

Bytes : 2
 Ciclos : 2
 Código : **1001 0010** **bit address**
 Operación : **(bit)** \leftarrow **(C)**

MOV DPTR,#data16

Carga el apuntador a la memoria de datos DPTR con un dato de 16 bits. La parte alta en DPH, y la baja en DPL.

Bytes : 3
 Ciclos : 2
 Código : 1001 0000 [data 15-8] [data 7-0]
 Operación : (DPTR) ← #data16
 ó DPH & DPL ← #data15-8 & #data7-0

MOVC A,@A+<base-reg>

Las instrucciones MOVC se emplean para transferir al Acumulador un byte de la memoria de código (indicado por la "C" de Code). La dirección del byte viene dada por la suma del contenido del Acumulador con el doble registro base-reg que puede ser el PC o el DPTR. En el primer caso, el PC es incrementado para apuntar a la siguiente instrucción antes de que se realice la suma con el contenido del Acumulador. En el segundo caso, el registro base no se modifica. Los flags no resultan afectados.

Ejemplo: Un valor comprendido entre 0 y 3 cargado en el acumulador permitirá realizar la lectura de uno de los cuatro valores definidos con la directiva de ensamblado DB.

```
REL_PC: INC A ;necesaria para salvar el byte de RET
        MOVC A,@A+PC
        RET
DB      66H,77H,88H,99H
```

MOVC A,@A+DPTR

Bytes : 1
 Ciclos : 2
 Código : 1001 0011
 Operación : (A) ← ((A)+(DPTR))

MOVC A,@A+PC

Bytes : 1
 Ciclos : 2
 Código : 1000 0011
 Operación : (PC) ← (PC) + 1
 (A) ← ((A)+(PC))

MOVX <byte-destino>,<byte-fuente>

Las instrucciones MOVX transfieren datos entre el Acumulador y un byte de la memoria externa de datos (indicado por la "X" de external). De los dos operandos, uno siempre es el acumulador, el otro operando es @DPTR o @Ri. Cuando un operando es @DPTR, los registros DPH y DPL proporcionan una dirección de 16 bits para acceder a cualquier posición de los 64 Kbytes de memoria externa. El puerto P2 saca la parte alta de la dirección (contenido de DPH), y P0 la baja (contenido de DPL). El SFR P2 retiene su anterior contenido mientras saca el valor de DPH. Este método es el más rápido y efectivo para el acceso a grandes arrays de hasta 64K, sin necesidad de instrucciones adicionales.

Cuando un operando es @Ri, el contenido de R0 o R1 del banco de registros actualmente seleccionado proporciona una dirección de 8 bits, multiplexada con datos en el puerto P0. Solo se necesitan 8 bits para expandir E/S decodificadas externamente o para pequeños arrays en RAM. Para manipular arrays más largos, puede emplearse algunas patillas de salida de un puerto para sacar los bits más altos de la dirección. Estas patillas se controlarán con una instrucción de salida que preceda a MOVX.

En algunas situaciones es posible la mezcla de los dos tipos de instrucciones MOVX. Un array muy largo en RAM con la parte alta de la dirección controlada por P2 puede ser direccionado mediante el DPTR o, escribiendo en P2 la parte alta de la dirección, y a continuación mediante instrucciones MOVX que utilicen R0 o R1.

Ejemplo: Una RAM externa de 256 bytes direccionada empleando las líneas de Direcciones/Datos multiplexados (por ejemplo un Intel 8155 RAM,I/O,Timer conectado a P0). El puerto P3 proporciona las líneas de control para memoria externa. P1 y P2 se utilizan como líneas de E/S. R0 y R1 contienen 12H y 34H respectivamente. La posición 34H de RAM externa contiene el dato 56H.

```
MOVX A,@R1
MOVX @R0,A
```

La ejecución de las instrucciones anteriores, habrá realizado la copia del dato 56H en el Acumulador y en la dirección 12H de RAM externa.

MOVX A,@Ri

Bytes	:	1
Ciclos	:	2
Código	:	1110 001i
Operación	:	(A) ← ((Ri))

MOVX A,@DPTR

Bytes	:	1
Ciclos	:	2
Código	:	1110 0000
Operación	:	(A) ← ((DPTR))

MOVX @Ri,A

Bytes	:	1
Ciclos	:	2
Código	:	1111 001i
Operación	:	((Ri)) ← (A)

MOVX @DPTR,A

Bytes	:	1
Ciclos	:	2
Código	:	1111 0000
Operación	:	((DPTR)) ← (A)

MUL AB

Multiplica un entero sin signo de 8 bits situado en el Acumulador por otro situado en el registro B. El byte bajo del resultado se deja en el Acumulador, y el byte alto en el registro

B. Si el producto supera el valor 255 (0FFH), el *flag* de Overflow se activa, de lo contrario se borra. El Carry siempre se borra.

Ejemplo: El Acumulador tiene 80 (50H). El Registro B tiene 160 (0A0H).

MUL AB

Se obtiene el producto 12.800 (3200H), con 32H cargado en el registro B, y 00H en el Acumulador. El *flag* Overflow queda activado, y el flag Carry borrado.

Bytes	:	1
Ciclos	:	4
Código	:	1010 0100
Operación	:	(A) 7-0 ← Parte Baja de (A) x (B) (B) 15-8 ← Parte Alta de (A) x (B)

NOP

No OPeración. El programa continúa en la siguiente instrucción, modificando tan solo el PC.

Ejemplo: Se desea generar un pulso de valor 0 en el bit 7 del puerto 2 durante 5 ciclos exactamente. Una única secuencia SETB/CLR generaría un pulso de solo un ciclo de duración. Para añadir los otros 4 ciclos adicionales hay que insertar 4 instrucciones NOP. Se supone que no están permitidas las interrupciones ya que en el caso de producirse alguna, el número de ciclos podría aumentar considerablemente mientras se atiende dicha interrupción.

```
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

Bytes	:	1
Ciclos	:	1
Código	:	0000 0000
Operación	:	(PC) ← (PC) + 1

ORL <byte-destino>,<byte-fuente>

La instrucción ORL proporciona la "operación lógica" OR entre las dos variables, dejando el resultado de la operación en byte-destino. Ningún *flag* se verá afectado. Los dos operandos pueden operar con seis modos diferentes de direccionamiento. Cuando el destino es el Acumulador, la fuente puede emplear direccionamiento directo, por registro, indirecto por registro o dato inmediato. Cuando el destino es una dirección la fuente puede ser un dato inmediato o el Acumulador.

Ejemplo: Supongamos que el Acumulador contiene 0C3H y el registro R0 contiene 55H. Después de ejecutar la instrucción: "ORL A,R0", El Acumulador tendrá 0D7H.

ORL A,Rn

Bytes	:	1
Ciclos	:	1
Código	:	0100 1rrr
Operación	:	(A) ← (A) OR (Rn)

ORL A,direct

Bytes : 2
 Ciclos : 1
 Código : **0100 0101** [direct]
 Operación : (A) \leftarrow (A) OR (direct)

ORL A,@Ri

Bytes : 1
 Ciclos : 1
 Código : **0100 011i**
 Operación : (A) \leftarrow (A) OR ((Ri))

ORL A,#data

Bytes : 2
 Ciclos : 1
 Código : **0100 0100** [data]
 Operación : (A) \leftarrow (A) OR #data

ORL direct,A

Bytes : 2
 Ciclos : 1
 Código : **0100 0010** [direct]
 Operación : (direct) \leftarrow (direct) OR (A)

ORL direct,#data

Bytes : 3
 Ciclos : 2
 Código : **0100 0011** [direct] [data]
 Operación : (direct) \leftarrow (direct) OR #data

ORL C,<bit-fuente>

Activa el Carry si el bit-fuente vale 1, de lo contrario lo deja como estaba. Si el bit-fuente va precedido del carácter "/", entonces se emplea el valor complementario del bit-fuente, pero este no se verá afectado. Ningún otro flag se verá afectado. Ejemplos:

```
MOV C,P1.0      ; Carry  $\leftarrow$  P1.0
ORL C,ACC.7     ; Carry  $\leftarrow$  Carry OR ACC.7
ORL C,/OV       ; Carry  $\leftarrow$  Carry OR NOT(OVerflow)
```

ORL C,bit

Bytes : 2
 Ciclos : 2
 Código : **0111 0010** [bit address]
 Operación : (C) \leftarrow (C) OR (bit)

ORL C,/bit

Bytes : 2
 Ciclos : 2
 Código : **1010 0000** [bit address]
 Operación : (C) \leftarrow (C) OR (bit)

POP direct

Se lee el contenido de una dirección de RAM interna apuntada por el Stack Pointer, y éste se decrementa en una posición. El valor leído se transfiere a la dirección indicada (direct).

Ningún *flag* se verá afectado.

Ejemplo: El SP tiene el valor 32H, y la posición de RAM interna 30H hasta la 32H, contienen los valores 20H,23H, y 01H respectivamente.

```
POP DPH ; DPH ← 01H y SP ← 31H
POP DPL ; DPL ← 23H y SP ← 30H
POP SP ; SP ← 20H
```

El último POP es un caso especial, el SP se decrementa a 2FH antes de cargarse con el valor 20H sacado de la pila.

Bytes : 2
Ciclos : 2

Código : **1101 0000** **direct**
Operación : $(\text{direct}) \leftarrow ((\text{SP}))$
 $(\text{SP}) \leftarrow (\text{SP}) - 1$

PUSH direct

El SP se incrementa en una unidad y el contenido de la dirección indicada (*direct*), se copia en la posición de RAM apuntada por el SP. Ningún *flag* se verá afectado.

Ejemplo: El SP contiene 09H. El DPTR tiene 0123H.

```
PUSH DPL ; SP ← 0AH y (0AH) ← 23H
PUSH DPH ; SP ← 0BH y (0BH) ← 01H
Bytes : 2
Ciclos : 2
Código : 1100 0000 direct  
Operación :  $(\text{SP}) \leftarrow (\text{SP}) + 1$   
 $((\text{SP})) \leftarrow (\text{direct})$ 
```

RET

Retorno de subprograma. RET saca de la PILA (POP) el byte alto y el byte bajo que colocará en el PC, decrementando el SP en 2 unidades. La ejecución del programa continúa en dicha dirección que por lo general es la correspondiente a la dirección siguiente a una instrucción ACALL o LCALL. Ningún *flag* queda afectado.

Bytes : 1
Ciclos : 2
Código : **0010 0010**
Operación : $(\text{PC.15-8}) \leftarrow ((\text{SP}))$
 $(\text{SP}) \leftarrow (\text{SP}) - 1$
 $(\text{PC.7-0}) \leftarrow ((\text{SP}))$
 $(\text{SP}) \leftarrow (\text{SP}) - 1$

RETI

Retorno desde Interrupción. RETI saca de la PILA (POP) el byte alto y el byte bajo que colocará en el PC, decrementando el SP en 2 unidades. Inicializa la lógica de atención a las interrupciones con objeto de que puedan ser atendidas otras interrupciones con el mismo nivel de prioridad que la que se ha terminado de procesar. La ejecución del programa continúa en la dirección obtenida de la PILA, que por lo general corresponde a la instrucción siguiente a aquella otra en que la petición de interrupción fue detectada y atendida.

Si una interrupción del mismo nivel o inferior está pendiente mientras la instrucción RETI se encuentra en proceso, se ejecutará otra instrucción mas, antes de que la interrupción pendiente sea aceptada.

Bytes : 1
 Ciclos : 2
 Código : **0011 0010**
 Operación : $(PC.15-8) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) - 1$
 $(PC.7-0) \leftarrow ((SP))$
 $(SP) \leftarrow (SP) - 1$

RL A

Los ocho bits del Acumulador serán rotados un bit a la izquierda (Left). El bit 7 será rotado sobre el bit 0. Ningún flag se verá afectado.

Bytes : 1
 Ciclos : 1
 Código : **0010 0011**
 Operación : $(A.n+1) \leftarrow (A.n)$ $n = 0..6$
 $(A.0) \leftarrow (A.7)$

RLC A

Los ocho bits del Acumulador serán rotados un bit a la izquierda (Left). El Carry será rotado sobre el bit 0, y el bit 7 sobre el Carry. Ningún flag se verá afectado.

Bytes : 1
 Ciclos : 1
 Código : **0011 0011**
 Operación : $(A.n+1) \leftarrow (A.n)$ $n = 0..6$
 $(A.0) \leftarrow (C)$
 $(C) \leftarrow (A.7)$

RR A

Los ocho bits del Acumulador serán rotados un bit a la derecha (Right). El bit 0 será rotado sobre el bit 7. Ningún flag se verá afectado.

Bytes : 1
 Ciclos : 1
 Código : **0000 0011**
 Operación : $(A.n) \leftarrow (A.n+1)$ $n = 0..6$
 $(A.7) \leftarrow (A.0)$

RRC A

Los ocho bits del Acumulador serán rotados un bit a la derecha (Right). El bit Carry será rotado sobre el bit 7, y el bit 0 sobre el Carry. Ningún flag se verá afectado.

Bytes : 1
 Ciclos : 1
 Código : **0001 0011**
 Operación : $(A.n) \leftarrow (A.n+1)$ $n = 0..6$
 $(A.7) \leftarrow (C)$
 $(C) \leftarrow (A.0)$

SETB <bit>

Pone a 1 el bit indicado. Este bit puede ser el Carry o cualquier bit direccionable.

SETB C

Bytes : 1
 Ciclos : 1
 Código :

1101	0011
------	------

 Operación : (C) \leftarrow 1

SETB bit

Bytes : 2
 Ciclos : 1
 Código :

1101	0010
------	------

 [bit address]
 Operación : (bit) \leftarrow 1

SJMP rel

Realiza un salto relativo incondicional. La dirección de salto se obtiene sumando un desplazamiento (rel) que puede ser positivo o negativo (indicado en complemento a dos), a la dirección que tendrá el PC cuando apunte al comienzo de la siguiente instrucción. De este modo el rango de destinos permitido va desde los 128 bytes precedentes al comienzo de la siguiente instrucción, hasta los 127 bytes siguientes.

Bytes : 2
 Ciclos : 2
 Código :

1000	0000
------	------

 [rel]
 Operación : (PC) \leftarrow (PC) + 2
 (PC) \leftarrow (PC) + rel

SUBB A,<byte-fuente>

Resta con Carry negativo (Borrow). SUBB resta la variable y el Carry al Acumulador, dejando el resultado en el Acumulador. SUBB activa el Carry (Borrow), si se necesita un Borrow para el bit 7, de lo contrario borrará el Carry. (Si el Carry estaba activado antes de ejecutar la instrucción SUBB, esto indica que se necesitó un Borrow para el paso previo en una resta de precisión múltiple, de forma que el Carry se restará del Acumulador junto con el byte-fuente) El Auxiliar-Carry se activará en el caso de necesitar un Borrow para el bit 3, de lo contrario se borrará. El flag OV se activará si se necesitara un Borrow para el bit 6 pero no para el 7, o para el 7 pero no para el 6.

Cuando se restan enteros con signo, el flag OV indica resultados negativos cuando se resta un número negativo de otro positivo, o un resultado positivo cuando se resta un número positivo de otro negativo.

Ejemplo: El Acumulador tiene 0C9H, R2 tiene 54H, y el Carry = 1.
 SUBB A,R2

Una vez ejecutada la instrucción, el Acumulador tendrá 74H, los flags C, y AC borrados, y el flag OV activado. Dado que 0C9H menos 54H es 75H, la diferencia entre este valor y el

resultado obtenido se debe al Carry (Borrow), activado antes de haberse ejecutado la instrucción SUBB. Si no se conoce el valor del Carry antes de comenzar una resta en simple o múltiple precisión, debe ponerse a cero con la instrucción CLR C.

SUBB A,Rn

Bytes : 1
 Ciclos : 1
 Código :

1001	1	rrrr
------	---	------

 Operación : $(A) \leftarrow (A) - (C) - (Rn)$

SUBB A,direct

Bytes : 2
 Ciclos : 1
 Código :

1001	0101	direct
------	------	--------

 Operación : $(A) \leftarrow (A) - (C) - (\text{direct})$

SUBB A,@Ri

Bytes : 1
 Ciclos : 1
 Código :

1001	011i
------	------

 Operación : $(A) \leftarrow (A) - (C) - ((Ri))$

SUBB A,#data

Bytes : 2
 Ciclos : 1
 Código :

1001	0100	data
------	------	------

 Operación : $(A) \leftarrow (A) - (C) - \#data$

SWAP A

Intercambio de los *nibbles* del Acumulador. Esta instrucción sustituye a cuatro instrucciones consecutivas de rotación. Ejemplo: Si el Acumulador contiene el valor 0C5H, después de la instrucción "SWAP A", el Acumulador tendrá el valor 5CH.

Bytes : 1
 Ciclos : 1
 Código :

1100	0100
------	------

 Operación : $(A.3-0) \leftrightarrow (A.7-4)$

XCH A,<byte>

XCH carga en el Acumulador el contenido de la variable indicada, y al mismo tiempo escribe el contenido original del Acumulador en la variable indicada. Los modos de direccionamiento son: registro, directo, e indirecto por registro. Ejemplo: Si el Acumulador tiene 3FH, el registro R0 contiene el valor 20H, y la dirección 20H de RAM interna contiene el valor 75H, después de ejecutar la instrucción "XCH A,@R0", quedará el Acumulador con el valor 75H, y la posición 20H de RAM interna con el valor 3FH.

XCH A,Rn

Bytes : 1
 Ciclos : 1

Código :

1100	1rrr
------	------

Operación : (A) ↔ (Rn)

XCH A,direct

Bytes : 2
 Ciclos : 1

Código :

1100	0101	direct
------	------	--------

Operación : (A) ↔ (direct)

XCH A,@Ri

Bytes : 1
 Ciclos : 1

Código :

1100	011i
------	------

Operación : (A) ↔ ((Ri))

XCHD A,@Ri

XCHD intercambia el *nibble* bajo del Acumulador (bits 3-0), con el *nibble* bajo de la posición de RAM interna apuntada por el registro Ri. Los *nibbles* altos (bits 7-4) de ambos registros no se ven afectados, ni tampoco los *flags*.

Bytes : 1
 Ciclos : 1

Código :

1101	011i
------	------

Operación : (A.3-0) ↔ ((Ri.3-0))

XRL <byte-destino>,<byte-fuente>

La instrucción XRL realiza la función OR exclusiva (XOR) entre las dos variables, dejando el resultado de la operación en byte-destino. Ningún *flag* se verá afectado.

Los dos operandos pueden operar con seis modos diferentes de direccionamiento. Cuando el destino es el Acumulador, la fuente puede emplear direccionamiento directo, por registro, indirecto por registro o dato inmediato. Cuando el destino es una dirección, la fuente puede ser un dato inmediato o el Acumulador.

XRL A,Rn

Bytes : 1
 Ciclos : 1

Código :

0110	1rrr
------	------

Operación : (A) ← (A) XOR (Rn)

XRL A,direct

Bytes : 2
 Ciclos : 1

Código :

0110	0101	direct
------	------	--------

Operación : (A) \leftarrow (A) XOR (direct)

XRL A,@Ri

Bytes : 1
 Ciclos : 1

Código :

0110	011i
------	------

Operación : (A) \leftarrow (A) XOR ((Ri))

XRL A,#data

Bytes : 2
 Ciclos : 1

Código :

0110	0100	data
------	------	------

Operación : (A) \leftarrow (A) XOR #data

XRL direct,A

Bytes : 2
 Ciclos : 1

Código :

0110	0010	direct
------	------	--------

Operación : (direct) \leftarrow (direct) XOR (A)

XRL direct,#data

Bytes : 3
 Ciclos : 2

Código :

0110	0011	direct	data
------	------	--------	------

Operación : (direct) \leftarrow (direct) XOR #data

Apéndice 4

Exámenes de Laboratorio

NOTA 1: Exámenes realizados hasta la fecha de publicación.

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIAL**INFORMÁTICA DE GESTIÓN 2º****EXÁMEN DE ESTRUCTURA DE COMPUTADORES II.**(Convocatoria Ordinaria de ENERO 2004)**LABORATORIO****1. EJERCICIO 1º :**

Tenemos una ruleta de tres posiciones, estas son la 1, 2 y 3; además tenemos un interruptor con el cual indicamos que comienza el juego.

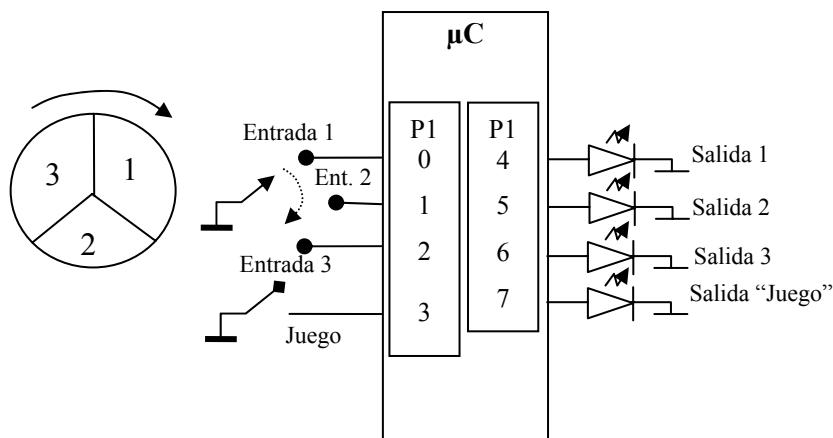
El programa está a la espera de que se active el interruptor para ver cual es el número que sale, esta señal está en la puerta “0” de P1. Cuando se active este interruptor activaremos la salida 7 de P1 y comenzaremos el juego.

- Cuando sea un “1” al contenido de la dirección 30H se le suma un “1” y se almacena el resultado en la dirección 80H y se activará la salida 4 de P1.
- Cuando sea un “2” al contenido de la dirección 30H se le suma un “2” y se almacena el resultado en la dirección 81H y se activará la salida 5 de P1.
- Cuando sea un “3” al contenido de la dirección 30H se le suma un “3” y se almacena el resultado en la dirección 82H y se activará la salida 6 de P1.

La señal de “1” se leerá de la puerta “0” de P1.

La señal de “2” se leerá de la puerta “1” de P1.

La señal de “3” se leerá de la puerta “2” de P1.



Las entradas son activas por nivel “0” y las salidas por nivel “1”.

2. EJERCICIO 2º :

Sumar los contenidos de las direcciones 80H, 81H, 82H y 83H, dejando el resultado en la posición 30H. Rotar a la izquierda el resultado y sacar el valor obtenido por el puerto P1.

NOTA : El ejercicio nº 1 se puntuará con 1.5 puntos y el nº 2 con 0.5 puntos

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIAL**INFORMÁTICA DE GESTIÓN 2º****EXÁMEN DE ESTRUCTURA DE COMPUTADORES II.****(Convocatoria Extraordinaria de JUNIO 2004)****LABORATORIO****1. EJERCICIO 1º :**

Disponemos de dos pulsadores, estos son el Pulsador 1 y el Pulsador 2; además disponemos de un interruptor “Juego” con el cual vamos a indicar al microcontrolador que comienza el juego.

El programa está a la espera de que se active el interruptor de “Juego”, una vez activo, el programa determinará cual es el pulsador que primero se ha activado o si se han activado los dos a la vez, ésta señal está en la puerta “2” de P1. Cuando se active este interruptor activaremos la salida 7 de P1, borraremos el contenido de las direcciones 30H, 31H y 32H y comenzaremos el juego.

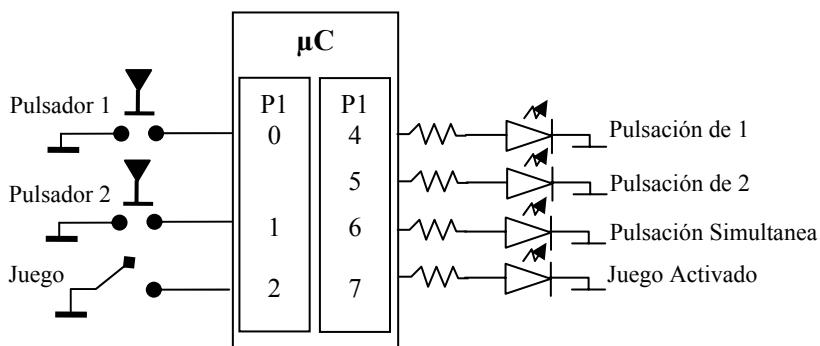
Las reglas son las siguientes :

- Cuando se pulse el “Pulsador 1” el contenido de la dirección 30H se incrementa en “1” y se almacena el resultado en la dirección 80H y se activa la salida 4 de P1.
- Cuando se pulse el “Pulsador 2” el contenido de la dirección 31H se incrementa en “1” y se almacena el resultado en la dirección 81H y se activa la salida 5 de P1.
- Cuando se pulsen simultáneamente los “Pulsadores 1 y 2”, el contenido de la dirección 32H se incrementa en “1” y se almacena el resultado en la dirección 82H y se activa la salida 6 de P1.
- Nuevas pulsaciones se tienen en cuenta cuando se han liberado ambos pulsadores.

La señal de “Pulsador 1” se leerá de la puerta “0” de P1.

La señal de “Pulsador 2” se leerá de la puerta “1” de P1.

La señal de “Juego” se leerá de la puerta “2” de P1.



Las entradas son activas por nivel “0” y las salidas por nivel “1”.

Las señales de salida permanecen hasta que se liberan ambos pulsadores o bien se desactiva la señal de juego.

2. EJERCICIO 2º :

Sumar los contenidos de las direcciones F0H y F1H, dejando el resultado en la posición 20H. Si hay bit de Carry, dejarlo en el bit “0” de 21H.

NOTA : El ejercicio nº 1 se puntuará con 1.5 puntos y el nº 2 con 0.5 puntos

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIALINFORMÁTICA DE GESTIÓN 2ºEXÁMEN DE ESTRUCTURA DE COMPUTADORES II.(Convocatoria Ordinaria de ENERO 2005)LABORATORIO**1. EJERCICIO 1º :**

Se trata de realizar un programa que efectúe una de dos operaciones (multiplicar o dividir) según el estado de un interruptor externo al procesador, la operación se habilita según el estado de otro interruptor. Las operaciones se realizarán en sendas subrutinas.

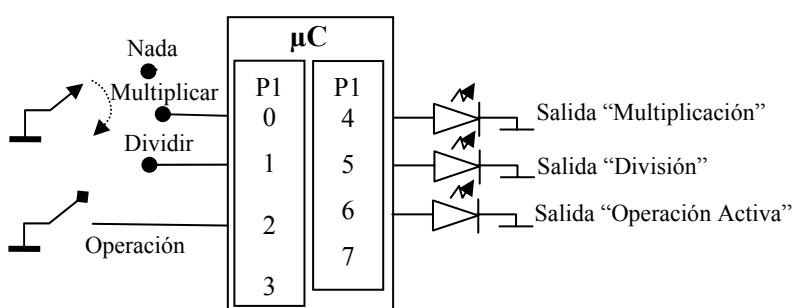
Disponemos de un interruptor de operación de tres posiciones:

- 1: No hacer nada. Posición al aire.
- 2: Multiplicar. Entrada asignada a la puerta “0” de P1
- 3: Dividir. Entrada asignada a la puerta “1” de P1

El interruptor con el cual habilitamos la operación está asignado a la puerta “2” de P1.

Si la entrada de “Operación” está activa, entonces se activará la salida “Operación Activa” puerta “6” de P1, en este momento se pasará a escrutar las entradas “Multiplicar” y “Dividir” para ver que operación se realiza.

- Si la entrada “Multiplicar” está activa, se activará la salida “Multiplicación” puerta 4 de P1 y se llamará a una **Subrutina** que multiplique por 2 el contenido de la dirección 40H. El valor obtenido en la subrutina se devolverá en un registro. El programa principal guardará este valor en la dirección 41H. El programa continuará revisando las nuevas condiciones.
- Si la entrada “Dividir” está activa, se activará la salida “División” puerta 5 de P1 y se llamará a una **Subrutina** que divida por 2 el contenido de la dirección 40H. El valor obtenido en la subrutina se devolverá en un registro. El programa principal guardará este valor en la dirección 42H. El programa continuará revisando las nuevas condiciones.



Las entradas son activas por nivel “0” y las salidas por nivel “1”.

2. EJERCICIO 2º :

Almacenar los valores 01H, 02H, 10H y 20H en las posiciones de memoria 90H, 91H, 92H y 93H respectivamente. Sumar los contenidos de estas posiciones de memoria dejando el resultado en la posición 30H y en la FFH.

NOTAS:

1. Ambos programas comenzarán en la dirección 20H.
2. Se comentarán ambos ejercicios (programas), no se admitirá un ejercicio con solo instrucciones.

El ejercicio nº 1 se puntuará con 1.5 puntos y el nº 2 con 0.5 puntos.

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIALINFORMÁTICA DE GESTIÓN 2ºEXÁMEN DE ESTRUCTURA DE COMPUTADORES II.

(Convocatoria Ordinaria de JUNIO 2005)

LABORATORIO**1. EJERCICIO 1º :**

Disponemos de tres entradas al microcontrolador, estas son E 1, E 2 y E 3; además disponemos de un interruptor “Proceso” con el cual vamos a indicar al microcontrolador que comienza el Proceso.

El programa está a la espera de que se active el interruptor de “Proceso”, ésta señal está en la puerta “0” de P1. Cuando se active este interruptor activaremos la salida 4 de P1, borraremos el contenido de las direcciones 20H, 21H y 22H y comenzaremos el Proceso. Una vez activo el Proceso, el programa determinará cual es la secuencia de Entradas (E) que se ha activado

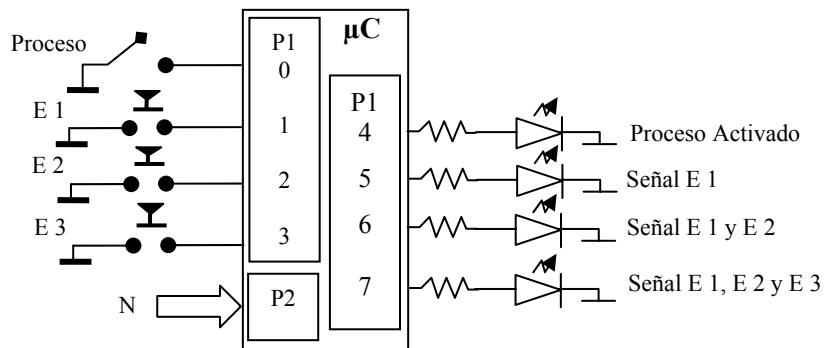
La reglas del Proceso son las siguientes :

- Cuando se active “E 1”, se activa la salida 5 de P1, se leerá el dato N de la puerta P2, se guardará este dato en la dirección 20H, se incrementa en “1” y se almacena el resultado en la dirección 80H.
- Cuando se activen simultáneamente “E 1 y E 2”, se activa la salida 6 de P1, se leerá el dato N de la puerta P2. Se guardará en la dirección 21H, se incrementa en “2” y se almacena el resultado en la dirección 81H.
- Cuando se activen simultáneamente “E 1, E 2 y E 3”, se activa la salida 7 de P1, se leerá el dato N de la puerta P2. Se guardará en la dirección 22H, se incrementa en “3” y se almacena el resultado en la dirección 82H.
- **Nuevas pulsaciones** se tienen en cuenta cuando se han liberado todos los pulsadores.

La señal “E 1” se leerá de la puerta “1” de P1.

La señal “E 2” se leerá de la puerta “2” de P1.

La señal “E 3” se leerá de la puerta “3” de P1.



Las entradas son activas por nivel “0” y las salidas por nivel “1”.

Las señales de salida permanecen hasta que se liberan ambos pulsadores o bien se desactiva la señal de proceso.

2. EJERCICIO 2º :

En la dirección 10H y 11H hay dos valores cualesquiera, si el contenido de la dirección 10H es mayor que el contenido de la 11H, entonces se decrementará el contenido de la 10H y se llevará el resultado a la dirección 90H. Si el contenido es menor entonces se incrementará y el resultado se llevará a la dirección 91H.

NOTAS:

1. Ambos programas comenzarán en la dirección 20H.
2. Se comentarán ambos ejercicios (programas), no se admitirá un ejercicio con solo instrucciones.

El ejercicio nº 1 se puntuará con 1.5 puntos y el nº 2 con 0.5 puntos.

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIALINFORMÁTICA DE GESTIÓN 2ºEXÁMEN DE ESTRUCTURA DE COMPUTADORES II.

(Convocatoria Ordinaria de ENERO 2006)

LABORATORIO**1. EJERCICIO 1º :**

Se trata de realizar un programa que efectúe una de tres operaciones (sumar, restar o no operar). El estado de un interruptor externo al procesador iniciará las operaciones. Las operaciones se realizarán en sendas subrutinas. Los datos a operar son N1 y N2 (enteros sin signo) ambos de cuatro bits, dispuestos sobre la puerta P1.

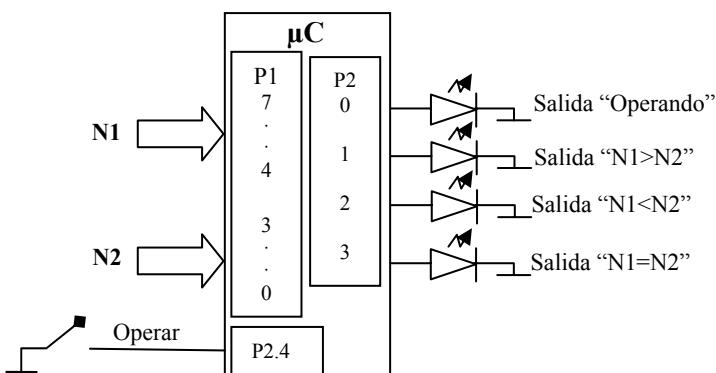
Disponemos de un interruptor “Operar” de dos posiciones, asignado a la puerta “4” de P2:

- 1: Interruptor abierto: No hacer nada.
- 2: Interruptor cerrado: Operar.

Si la entrada “Operar” está activa, entonces se activará la salida “Operando” puerta “0” de P2, en este momento se pasará a escrutar las entradas de datos “N1” (P1.7 .. P1.4) y “N2” (P1.3 .. P1.0) para ver qué operación se realiza, según las siguientes condiciones:

- Si N1 es mayor que N2, se activará la salida “N1>N2” puerta 1 de P2 y se llamará a una **Subrutina** que reste estos dos valores (N1-N2). El valor obtenido se guardará en la dirección 80H. El programa continuará revisando las nuevas condiciones.
- Si N1 es menor que N2, se activará la salida “N1<N2” puerta 2 de P2 y se llamará a una **Subrutina** que sume estos dos valores (N1+N2). El valor obtenido se guardará en la dirección 81H. El programa continuará revisando las nuevas condiciones.
- Si N1 es igual a N2, se activará la salida “N1=N2” puerta 3 de P2, no realizándose ninguna operación. El programa continuará revisando las nuevas condiciones.

Una vez terminada una operación, para volver a realizar nuevas operaciones hay que esperar a que la señal “Operar” se desactive.



La entrada “Operar” es activa por nivel “0” y las salidas por nivel “1”.

Los valores N1 y N2 se salvarán en posiciones de memoria baja o en registros, a voluntad del alumno.

2. EJERCICIO 2º :

Almacenar los valores 01H, 02H, 11H y 22H en las posiciones de memoria 80H, 81H, 82H y 83H respectivamente. Sumar los contenidos de estas posiciones de memoria dejando el resultado en la posición 84H y llevar el resultado a la puerta P1.

NOTAS:

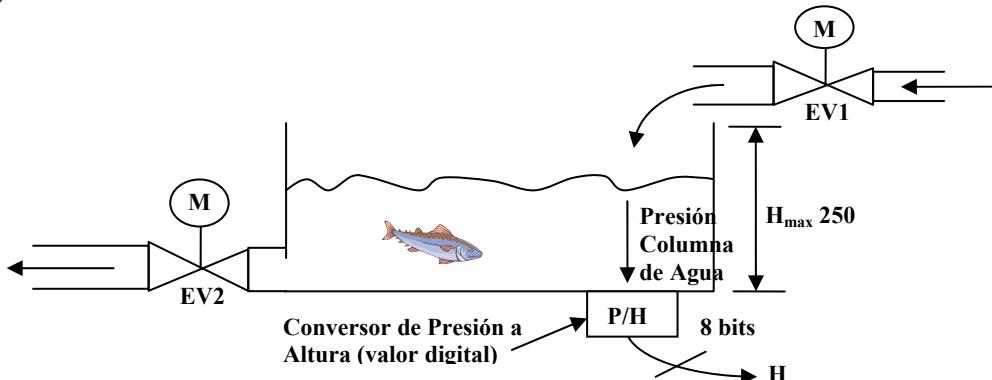
1. Ambos programas comenzarán en la dirección 20H.
2. Se comentarán ambos ejercicios (programas), no se admitirá un ejercicio con solo instrucciones.
3. El ejercicio nº 1 se puntuará con 1.5 puntos y el nº 2 con 0.5 puntos.

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIALINFORMÁTICA DE GESTIÓN 2ºEXÁMEN DE ESTRUCTURA DE COMPUTADORES II.

(Convocatoria Ordinaria de JUNIO 2006)

LABORATORIO**1. EJERCICIO 1º :**

Se trata de realizar un programa que regule el nivel de agua en un depósito. La altura (H) del nivel de agua se dispone sobre la puerta P2 de nuestro microControlador. H (altura) es un **valor digital** que se lee de un conversor directamente. Se Abren o Cierran dos electroválvulas EV1 (puerta 2 de P1) y EV2 (puerta 3 de P1). La posición de las electroválvulas nos la da los sensores PEV1 (puerta 0 de P1) y PEV2 (puerta 1 de P1).

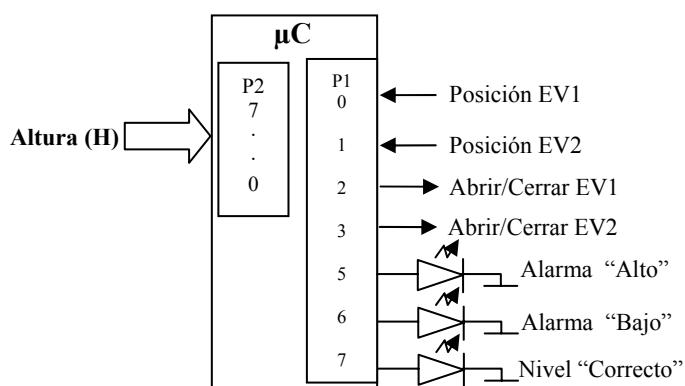


La funcionalidad es la siguiente:

- Si H es mayor que 220 cm., se activará la salida Alarma “Alto”, puerta 5 de P1, y se llamará a una **Subrutina** que Cierre EV1 y Abra EV2. El programa continuará revisando las nuevas condiciones.
- Si H es menor que 50 cm., se activará la salida Alarma “Bajo”, puerta 6 de P1, y se llamará a una **Subrutina** que Abra EV1 y Cierre EV2. El programa continuará revisando las nuevas condiciones.
- Si H es mayor que 50 cm. y menor que 220 cm., se activará la salida Nivel “Correcto”, puerta 7 de P1, y se llamará a una **Subrutina** que Abra EV1 y Abra EV2. El programa continuará revisando las nuevas condiciones.

Nota: Antes de realizar la acción de Abrir o Cerrar, se comprobará la posición para determinar si se lleva a cabo o no. **El programa no termina nunca.**

Para facilitar el ejercicio, vamos a suponer que con la orden de Abrir o Cerrar las electroválvulas, éstas Abren y Cierran siempre.

**2. EJERCICIO 2º :**

Sumar los contenidos de las direcciones F0H y F1H, dejando el resultado en la posición 20H. Si hay bit de Carry, dejarlo en el bit “0” de 21H.

NOTAS:

1. Ambos programas comenzarán en la dirección 20H.
2. Se comentarán (partes significativas) ambos ejercicios, no se admitirá un ejercicio con solo instrucciones.
3. El ejercicio nº 1 se puntuará con 1.5 puntos y el nº 2 con 0.5 puntos.