TUTORÍA #1

BY BRANDO MIGUEL PALACIOS MOGOLLON

By Brando miguel palacios mogollon

PALABRAS RESERVADAS

- MOV A,#3Bh
- SETB P1.1 El bit se pone a 1
- MUL AB
- DIV AB
- INC R1 R1← R1+1
- DEC R3 R3←R3-1
- SUBB A,R2 // A ← A-C-R2 C:carry A<R2+C → C=1 si A>=R2+C → C=0
- ADD A,R3 // A←A+R3 A+R3>#FFh → C=1

Instrucción: **CJNE**
Función: Compara y salta si los dos operandos no son iguales
Sintaxis: CJNE operando1,operando2,offset

Instrucción: DJNZ
Función: Decrementa y salta si el operando no es 0
Sintaxis: DJNZ operando,offset

Instrucción: JNB
Función: Salta si el bit implicado no vale 1
Sintaxis: JNB bit,offset

Instrucción: JB
Función: Salta si el bit implicado vale 1
Sintaxis: JB bit,offset

Instrucción: RR
Función: Rota el acumulador hacia la derecha
Sintaxis: RR A

Instrucción: CPL
Función: Complementa los ocho bits del acumulador
Sintaxis: CPL A

Instrucción: CLR
Función: Pone a cero los ocho bits del acumulador
Sintaxis: CLR A

Instrucción: RET
Función: Retorno desde subrutina
Sintaxis: RET

Instrucción: RETI
Función: Retorno desde interrupción
Sintaxis: RETI

Instrucción: RL
Función: Rota el acumulador hacia la izquierda
Sintaxis: RL A

Instrucción: LJMP
Función: Salto incondicional
Sintaxis: LJMP *dir_16*

Instrucción: AJMP
Función: Salto absoluto dentro de un bloque de 2K
Sintaxis: AJMP *dir_11*

Instrucción: SJMP
Función: Salto corto
Sintaxis: SJMP offset

By Brando Miguel Palacios Mogollon

Instrucción: SJMP
Función: Salto corto
Sintaxis: SJMP offset

Instrucción	Código de Operación	2° Byte	Bytes	Ciclos	Flags
SJMP offset	0x80	offset	2	2	-

***Operación**: SJMP offset*

(PC)<= (PC) + 2
(PC)<= (PC) + offset

***Descripción**:* El control del programa salta incondicionalmente a la dirección indicada. La dirección a donde saltar se obtiene sumando el *offset* (último byte de la instrucción), al PC (*Program Counter*) después de que éste se haya incrementado hasta el comienzo de la siguiente instrucción. El offset representa una cantidad entera con signo, y permite saltos de hasta 127 posiciones hacia adelante, y hasta 128 posiciones hacia atrás, medidos desde la dirección de comienzo de la siguiente instrucción.

By Brando Miguel Palacios Mogollon

Instrucción: LJMP
Función: Salto incondicional
Sintaxis: LJMP *dir_16*

Instrucción	Código de Operación	Byte 2°	Byte 3°	Bytes	Ciclos	Flags
LJMP *dir_16*	0x02	dir 15-8	dir 7-0	3	2	-

Operación: LJMP

(PC)<= dir_16

Descripción: *LJMP* realiza una salto incondicional a la dirección de 16 bits indicada en los dos últimos bytes de la instrucción. El destino puede ser cualquier posición de los 64 Kbytes de memoria de código.

Instrucción: AJMP
Función: Salto absoluto dentro de un bloque de 2K
Sintaxis: AJMP *dir_11*

Instrucción	Código de Operación	2° Byte	Bytes	Ciclos	Flags
AJMP *dir_11*	a10 a9 a8 0 0 0 0 1	dir 7-0	2	2	-

Operación: AJMP

(PC)<= (PC) + 2
(PC10-0)<= dir_11

Descripción: *AJMP realiza una salto a la dirección indicada de la memoria de código.* La dirección de salto, o nuevo valor para el PC (Program Counter) se obtiene uniendo a los 5 bits de mayor peso del PC (incrementado dos veces), los bits 7-5 del código de operación y el segundo byte de la instrucción.

Como la instrucción AJMP sólo afecta a los 11 bits de menor peso del PC, la llamada siempre se produce a una dirección de memoria de código situada dentro del bloque de 2K al que pertenece el primer byte de la instrucción que sigue al AJMP.

Instrucción: RET
Función: Retorno desde subrutina
Sintaxis: RET

Instrucción	Código de Operación	Bytes	Ciclos	Flags
RET	0x22	1	2	-

***Operación**: RET*

(PC15-8)<= ((SP))
(SP)<= (SP) - 1
(PC7-0)<= ((SP))
(SP)<= (SP) - 1

***Descripción**:* RET se utiliza para retornar desde una subrutina llamada previamente con LCALL o ACALL. La ejecución del programa continúa desde la dirección formada al extraer 2 bytes de la pila. En primer lugar de la pila se saca el byte más significativo.

Instrucción: RETI
Función: Retorno desde interrupción
Sintaxis: RETI

Instrucción	Código de Operación	Bytes	Ciclos	Flags
RETI	0x32	1	2	-

***Operación**: RETI*

(PC15-8)<= ((SP))
(SP)<= (SP) - 1
(PC7-0)<= ((SP))
(SP)<= (SP) - 1

***Descripción**:* RETI se utiliza para retornar desde una rutina de atención a una interrupción. La ejecución del programa continúa desde la dirección formada al extraer 2 bytes de la pila. En primer lugar de la pila se saca el byte más significativo. Antes de que el programa salte a la dirección extraída de la pila, RETI repone el sistema de interrupciones para que sean aceptadas las interrupciones con menor o igual prioridad a la que se acaba de atender.

Instrucción: CPL
Función: Complementa los ocho bits del acumulador
Sintaxis: CPL A

Instrucción	Código de Operación	Bytes	Ciclos	Flags
CPL A	0xF4	1	1	-

***Operación**: CPL A*

(A) <= NOT (A)

***Descripción**: CPL* A complementa el contenido del acumulador. Cada bit del acumulador que esté a "1" se pondrá a "0" y al revés.

Instrucción: CLR
Función: Pone a cero los ocho bits del acumulador
Sintaxis: CLR A

Instrucción	Código de Operación	Bytes	Ciclos	Flags
CLR A	0xE4	1	1	-

***Operación**: CLR A*

(A) <= 0

***Descripción**: CLR* A pone a cero el acumulador.

By Brando Miguel Palacios Mogollon

Instrucción: CJNE

Función: Compara y salta si los dos operandos no son iguales

Sintaxis: CJNE operando1,operando2,offset

Instrucción	Código de Operación	Byte 2°	Byte 3°	Bytes	Ciclos	Flags
CJNE A,direcc,offset	0xB5	direcc	offset	3	2	C
CJNE A,#dato,offset	0xB4	dato	offset	3	2	C
CJNE Rn,#dato,offset	1 0 1 1 1 r r r	dato	offset	3	2	C
CJNE @Ri,#dato,offset	1 0 1 1 0 1 1 i	dato	offset	3	2	C

Operación: *CJNE* operando1,operando2,offset

(PC)<= (PC) + 3
IF operando1 < > operando2
THEN
(PC)<= (PC) + offset
IF operando1 < operando2
THEN
(C)<= 1
ELSE
(C)<= 0

By Brando Miguel Palacios Mogollon

Instrucción: DJNZ
Función: Decrementa y salta si el operando no es 0
Sintaxis: DJNZ operando,offset

Instrucción	Código de Operación	Byte 2°	Byte 3°	Bytes	Ciclos	Flags
DJNZ Rn,offset	1 1 0 1 1 r r r	offset	-	2	2	-
DJNZ direcc,offset	0xD5	direcc	offset	3	2	-

Operación: *DJNZ* operando,offset

(PC)<= (PC) + 2 (ó 3)
operando <= operando -1
IF operando < > 0
THEN
(PC)<= (PC) + offset

Descripción: *DJNZ* decrementa el *operando* y si el nuevo valor es distinto de cero, se produce el salto. Si el valor del operando es cero, el programa continúa con la siguiente instrucción a DJNZ. La dirección a donde saltar se obtiene sumando el *offset* (último byte de la instrucción), al PC (*Program Counter*) después de que éste se haya incrementado hasta el comienzo de la siguiente instrucción. El offset representa una cantidad entera con signo, y permite saltos de hasta 127 posiciones hacia adelante, y hasta 128 posiciones hacia atrás, sobre la dirección de comienzo de la siguiente instrucción.

Si el valor inicial del *operando* es cero, al ser decrementado pasa a valer 0xFF, sin que el bit C se vea afectado por ello. Se puede utilizar para repetir un conjunto de instrucciones desde 1 hasta 256 veces.

By Brando Miguel Palacios Mogollon

Instrucción: JNB
Función: Salta si el bit implicado no vale 1
Sintaxis: JNB bit,offset

Instrucción	Código de Operación	Byte 2°	Byte 3°	Bytes	Ciclos	Flags
JNB bit,offset	0x30	bit	offset	3	2	-

***Operación**: JNB bit,offset*

(PC)<= (PC) + 3
IF (bit) = 0
THEN
(PC)<= (PC) + offset

***Descripción**:* Si el bit implicado es igual a 0 se salta a la dirección indicada. En caso contrario se procesa la siguiente instrucción. La dirección a donde saltar se obtiene sumando el *offset* (último byte de la instrucción), al PC (*Program Counter*) después de que éste se haya incrementado hasta el comienzo de la siguiente instrucción. El offset representa una cantidad entera con signo, y permite saltos de hasta 127 posiciones hacia adelante, y hasta 128 posiciones hacia atrás, medidos desde la dirección de comienzo de la siguiente instrucción.

By Brando Miguel Palacios Mogollon

Instrucción: RR
Función: Rota el acumulador hacia la derecha
Sintaxis: RR A

Instrucción	Código de Operación	Bytes	Ciclos	Flags
RR A	0x03	1	1	-

***Operación**: RR A*

(An)<= (An + 1) Para n = 0-6
(A7)<= (A0)

***Descripción**: RR* A rota los ocho bits del acumulador un lugar hacia la derecha. El bit 0 se lleva a la posición del bit 7.

By Brando Miguel Palacios Mogollon

SUBRUTINAS

- Simplificación de código
- Incompleto aprendizaje

By Brando Miguel Palacios Mogollon

SUBRUTINAS DE ENTRADA

GETCHR

Esta rutina lee un caracter del puerto serie y lo guarda en el acumulador.

GETBYT

Esta rutina lee un número ascii hexadecimal de 2 dígitos desde el puerto serie. El resultado es retornado en el acumulador.

By Brando Miguel Palacios Mogollon

SUBRUTINAS DE CONVERCION

BINASC

Toma el contenido del acumulador y lo convierte en dos números ascii hexadecimales. El resultado es retornado en el acumulador y R2.

ASCBIN

Esta rutina toma el caracter ascii pasado en el acumulador y lo convierte a un número binario de 4 bits que es retornado en el acumulador.

By Brando Miguel Palacios Mogollon

SUBRUTINAS DE SALIDA

PRTHEX

Esta rutina toma el contenido del acumulador y lo envía vía serie como 2 dígitos ascii hexadecimal.

PRINT

Toma la cadena inmediatamente que sigue a "lcall“ y lo envia por el puerto serie. La cadena debe terminar con un nulo(0). Esta subrutina retornará a la instrucción que sigue inmediatamente a la cadena.

SNDCHR

Que envia un carácter que está contenido en el acumulador A.

By Brando Miguel Palacios Mogollon

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

ollon

TIMERS:

Para el **Timer 0** y el **Timer 1** es:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Para el **Timer 2** es:

T2CON

0	0	0	0	0	0	0	0	Value after reset
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2	Bit name
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

By Brando Miguel Palacios Mogollon

(TIMER 0)

TF0: Flag del TIMER 0 es un valor booleano de 0 y 1, es el indicador del TIMER cuento es desborda

TR0: Bit de control del TIMER sirva para activar el uso del TIMER 0

#Spoilers:

IE0: Bandera de transición de la interrupción externa 0

IT0: BIT de control de interrupción 0

TH0: Posición de memoria donde se almacena los valores de partida del desvorde primera prioridad (TIMER HIGH)

TL0: Posición de memoria donde se almacena los valores de partida del desvorde segunda prioridad (TIMER LOW) //Tanto TH0 como TL0 dependerá del modo y uso.

By Brando Miguel Palacios Mogollon

OPCIONES DE TMOD

TMOD: Moderador de timers, registro que posee opciones de como trabajar con un timer como temporizadores (timers) o como contadores (counters).

M1	M0	MODO	DESCRIPCION
0	0	0	Timer/Contador de 13 bits.
0	1	1	Timer/Contador de 16 bits.
1	0	2	Timer/Contador de 8 bits recargables.
1	1	3	Timer 0, TL0 Timer/Contador de 8 bits, controlado por los bits de control del Timer 0, TH0 Timer de 8 bits controlado por los bits de control del timer1. El Timer 1 no se utiliza.

By Brando Miguel Palacios Mogollon

TMOD 0 Y 1

MOD 0 : Sea el timer que sea, configuran como registros de 13 bits, que consisten en los 8 bits del registro de TH y los 5 bits menos significativos del registro TL. Los 3 bits más significativos de TL no son utilizados en este modo.

MOD 1 : Este modo será con registros de 16 bits, es decir de 8 bits cada uno respectivamente

Nota: Esta forma de almacenar memoria se hace para aumentar la capacidad de los FFh.

By Brando Miguel Palacios Mogollon

TMOD 2

MOD 2: Este es un conteo de solo 8 bits con recarga automática. Al ser sobrepasada la capacidad de TL, éste es recargado automáticamente, con el contenido de TH y a su vez es activada (TF=1) la bandera de sobreflujo.

By Brando Miguel Palacios Mogollon

TIMER VS COUNTER

TIMER: mide frecuencias físicas mediante cálculos matemáticos y desbordamientos. Se busca llenar el flag hasta desbordando empezando desde un punto inicial.

Contador: Como se se tiende en la misma palabra esta forma de uso se emplea para contar los pulsos hasta un punto final.

By Brando Miguel Palacios Mogollon

CÁLCULOS

- Modo 0 : 2^13 =8192
- Modo 1: 2^16 = 65536
- Modo 2: 2^8 = 256

Caso 1:

Kx2^nx(12/11.0592x10^6)s=1/f s donde f: frecuencia

Cuando el K es menor a 1 siendo un valor poco acotado

Caso 2:

(65536-j)x(12/11.0592x10^6)s=0.033s

Nota: el resultado de K y j se almacenan en la posición TH0 como prioridad, se sobrar se lo coloca en TL0 como un numero mas.

By Brando Miguel Palacios Mogollon

Frecuencia de 10Hz con un reloj de 11.0592Mhz

Modo 0:

Modo 1:

Modo 2:

By Brando Miguel Palacios Mogollon

```
        org 8000h
        mov TMOD , # 00000010b
        mov TH0 , # 31
        setb TR0 ; INICIA el timer 0
espera_desborde:
        jnb TF0 , $ ; salta así mismo mientras TF0 es 0 (sin desbordamiento)
        cpl P1. 4 ; complementario el pin 5 (bits 0 -> bits 1 y bits 1 -> bits 0)
        clr TF0 ; TF0 pasa de ser 1 a 0
        sjmp espera_desborde
        end
```

By Brando Miguel Palacios Mogollon

CONTROL DEL PUERTO SERIE (SCON)

- Utilidades:
- Ingresar datos del terminal a a la fuente mediante las subrutinas necesarias

EJM:

EJEMPLO 1: Enviar a la PC vía comunicación serie los caracteres 1, 2, 3, 4 y 5 luego de recibir el carácter B

By Brando Miguel Palacios Mogollon

```
init    equ 127h
getchr  equ 121h
sndchr  equ 148h
 org 8000h
            lcall init; inicializa el puerto serie y configura el baud rate
repite:     lcall getchr; espera que se le envíe un caracter vía puerto serie
            cjne A, #42h, repite; si A no es "B" salta a repite, caso contrario
continua
  mov 30h, A
```

```
repite2:
  mov A, #31h; se mueve "1" al
acumulador
  lcall sndchr
  mov A, #32h
  lcall sndchr
  mov A, #33h
  lcall sndchr
  mov A, #34h
  lcall sndchr
  mov A, #35h
  lcall sndchr
  mov A, #0Dh; retorno de carro
  lcall sndchr
  mov A, #0Ah; salto de línea
  lcall sndchr
  sjmp repite2
  end
```

By Brando Miguel Palacios Mogollon