Capítol 9

ASSEMBLANT I SIMULANT AMB EDUP12

En aquest capítol s'introdueix un assemblador bàsic que permet escriure de forma més fàcil els programes que corren en EduP12.

L'assemblador que es presenta és simple i, tot i que realitza un petit tractament d'errors de sintaxi, en cap cas evita que la introducció d'errors de programació per part de l'usuari sigui corregit. Per això el capítol introdueix un conjunt de normes de *bona pràctica* en l'escriptura d'un programa en assemblador per evitar la pèrdua de temps degut a errades de sintaxi i de programació.

9.1 L'assemblador

L'assemblador és un petit llenguatge de baix nivell, molt proper a la màquina, que simplifica enormement la tasca de programació del processador. La seva missió principal és la de permetre al programador escriure el codi de la màquina emprant un conjunt de mnemotècnics que representen al codi màquina de la instrucció.

UEn programa escrit en llenguatge assemblador, un cop assemblat, es converteix en codi màquina. El llenguatge assemblador és, per tant, específic del processador. Un programa escrit en assemblador no es compila, sinó que s'assembla.

El programa que realitza l'assemblatge també s'anomena assemblador. El programa escrit en assemblador és el codi font, i el programa assemblat és el codi objecte. El codi objecte és el que conté el codi màquina

9.2 Assemblador asmEduP12.

L'assemblador del processador EduP12 consta de directives i de mnemotècnics. Les directives són comandes que simplifiquen la comprensió del programa per mitjà d'otorgar noms específics a les dades i als registres que empra el programa. Els mnemotècnics són els noms donats a les intruccions, que ja s'han introduït en capítols anteriors.

9.2.1 Directives.

Les directives emprades en el processador són:

.ORG <pos>

Indica la posició on es posa l'origen de les instruccions. <pos> és una constant numèrica (decimal o hexadecimal) indica la posició de memòria on es comencen a posar les instruccions que succeixen a la directiva.

.CON <nom> = <constant>

Dóna nom a valors constants que s'empren en el programa. <nom> és el nom que rep la constant. <constant> és una constant numèrica (decimal o hexadecimal.

.DEF <nom> = <Rx>

Dóna nom als regsitres que s'utilitzen en el programa. <nom> és el nom que rep el registre. <Rx> és el registre que s'anomena. Tot i que s'assigni un nom a un registre, no és requisit haver d'emprar el nom donat per referenciar al registre.

.DW <valor 1> , <valor 2> , ... , <valor n>

Permet guardar un conjunt de valors en memòria de programa. És útil, per exemple, per guardar taules de valors que han de ser utilitzats pel programa.

9.2.2 Instruccions

Els mnemotècnics de les instruccions que s'empren en el llenguatge assemblador ja s'han introduït en capítols anteriors i es troben especificades en l'apèndix A1. La primera columna de la taula mostra el format que cal seguir en cada instrucció.

9.3 Guia de bones pràctiques

Per a programar un processador en llenguatge assemblador es recomanable seguir els següents consells:

- Primer de tot cal fer un diagrama de flux del procés.
- Si es fa un diagrama de flux d'alt nivell, basat en instruccions de llenguatge de programació d'alt nivell, és útil convertir-lo a un diagrama de flux basat en instruccions del tipus RTL (llenguatge de transferència de registres).
- El diagrama de flux s'ha de basar en el format de les instruccions assemblador amb les que es treballa. Els típics bucles i salts condicionals de llenguatges de programació d'alt nivell aquí convé especificar-los en funció de les instrucció de salt de què es disposa, basades en el registre d'estat.

En el que respecte al format que s'ha de seguir en l'escriptura del programa en assemblador convé seguir les següents normes:

- Format general:
    - Totes les instruccions acaben en *;*. Quan no hi ha *;* en una línia es considera comentari i l'assemblador no la tracta.
    - Quan una instrucció s'identifica per etiqueta, aquesta ocupa el primer camp i acaba amb *:*.
- Respecte a directives:
    - És util redefinir constants i registres a valors pràctics del programa

  - Però s'han de designar als registres x, y i z com a tals sempre que s'usin com a índex.
  - La directiva .ORG és útil sempre que es vulgui posar un determinat programa en una posició determinada de la memòria. Cal anar en compta, però, en no reescriure una part del programa per un ús excessiu d'aquesta directiva.

- En números amb signe, no hi ha d'haver espai entre signe i número
- Quan s'empra pre-increment o pre-decrement en registre índex, el signe ha d'anar enganxat al registre i/o a la constant. El registre índex ha de ser X, Y o Z. Exemples:

```
LD R16, +X;
STD Z+40, R0;
```

- El format general és :

```
.ORG 0;              - Tot programa comença en la posició 0
RJMP main;           - Salt al programa principal
... RJMP isr1;       - Salts a rutines de servei d'interrupció (quan s'empren interrupcions)
...
main: ...    - Rutina principal del programa
....
Isr1: ...            - Inicia primera rutina de servei d'interrupció
...
RETI
...
```

9.4 Exemples.

Exemple 1.

En aquest primer exemple es mostra en els leds l'estat dels interruptors de la placa Spartan3. Quan s'apreta el botó 0 s'agafa la dada de l'interruptor i s'envia als leds. Mentre no s'apreti, no s'actualitzen els leds.

El programa presenta una nova versió delprograma ja introduït en apartats anteriors, però es fa pera què e puguin comparar la versió que no empra interrupcions amb la que sí n'empra.

El plantejament del programa és el següent:

- S'empren els següents elements de la placa:
  - Els interruptors. Es troben connectats en el PORTA d'entrada.
  - Els leds. Es troben connectats al port de sortida PORTB.
  - S'empra el botó 0. Es troba connectat en el pin 1 del port d'interrupcions. Com que no s'empren interrupcions, s'empra el port com a entrada sota el nom PORTE.
- El funcionament del programa és simple. S'està en un bucle d'espera fins que s'apreta el botó. Aleshores s'agafa l'estat dels interruptors i s'envia als leds.
- En el diagrama de flux (figura 9.1) s'observa que per conèixer si s'ha apretat el botó 0 (connectat al pin 1 del PORTA) es fa la AND amb 0x002. Com que el pin està en repòs a 0-lògic, mentre es compleixi la condició implica que no s'ha apretat el botó. En apretar el botó es llegeix el nou valor en el port d'interruptors (PORTA) i s'envia als leds (PORTB).
- En el programa s'ha previst una rutina d'espera d'uns 2 ms (treballant a 20MHz) per limitar l'efecte rebot del botó.

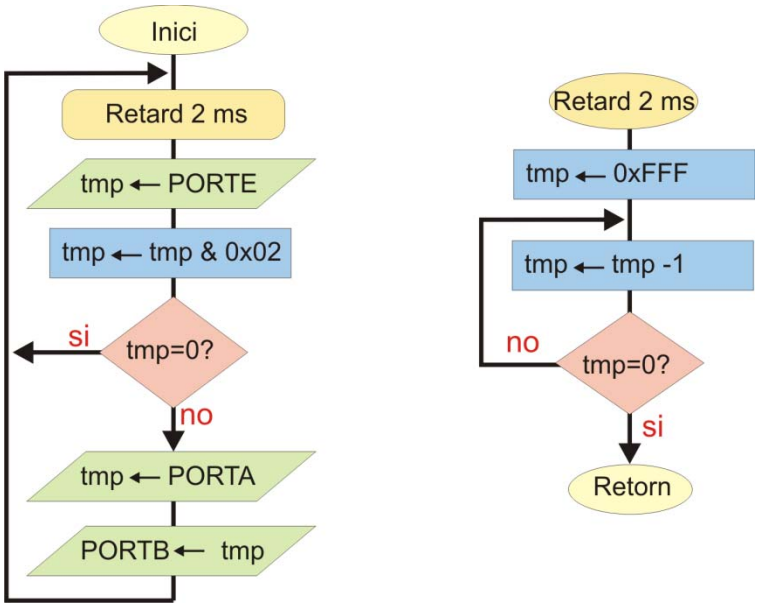

Figura 9.1. Diagrama de flux de l'exemple 1.

La figura 9.2 mostra el codi assemblador del programa.

-- INTERRUTPROS EN LEDS	
.DEF tmp = R16;	
inici: RCALL espera;	(0, "0101000000000111"), -- 0x5007 - rcall 7
IN tmp, PORTE;	(1, "0111000100000010"), -- 0x7102 - in r16 000010
ANDI tmp, 0x002;	(2, "0011000100000010"), -- 0x3102 - andi r16 0x002
	(3, "0000000000000010"), -- 0x2
BRZE inici;	(4, "1110111111011001"), -- 0xEFD9 - brze -5
IN tmp, PORTA;	(5, "0111000100000000"), -- 0x7100 - in r16 000000
OUT PORTB, tmp;	(6, "0111100100000001"), -- 0x7901 - out 000001 r16
RJMP inici;	(7, "0100111111111000"), -- 0x4FF8 - rjmp -8
--Bucle d'espera	
espera: LDI tmp, 0xFFF;	(8, "0011000100000000"), -- 0x3100 - ldi r16 0xfff
	(9, "0000111111111111"), -- 0xFFF
bucle: DEC tmp;	(10, "0011000100001001"), -- 0x3109 - dec r16
BRNZ bucle;	(11, "1111111111110001"), -- 0xFFF1 - brnz -2
RET;	(12, "0110000000000100"), -- 0x6004 - ret
a) Codi assemblador	**b) Codi màquina**

Figura 9.2. Codi assemblador i codi màquina de l'exemple 1.

Introducció d'interrupcions en l'exemple 1.

En un processador amb interrupcions, el plantejament fet d'aquest exercici és ineficient, ja que el programa sempre queda esperant que s'apreti el botó sense poder fer cap altra acció.

La interrupció permet millorar el funcionament del processador en tant que l'allibera de la tasca d'espera. Programant una interrupció per flanc (de pujada, per exemple) en el polsador 1, el processador pot estar realitzant altres tasques mentre no s'apreti el polsador. En apretar el polsador (produeix un flanc de pujada) s'entra en la rutina de servei de la interrupció i aquesta és la responsable de realitzar la corresponent actuació.

El plantejament del problema ara és molt més simple:

- El programa principal inicialitza el processador per interrupció per flanc de pujada en el polsador 1. Donat que el flanc de pujada del polsador 1 habilita la interrupció externa 4 s'ha d'incialitzar el registre de control d'interrupcions externes a 0x008.
- El programa principal és el responsable del procés del sistema. Pot estar fent altres accions mentre no s'apreti el botó. El fet que es deixi en un bucle infinit es deu al fet que aquest programa és molt simple i no se li requereixen més accions al processador.
- En activar el polsador el processador salta a tractar la rutina de servei de la interrupció, que agafa la dada dels interruptors (PORTA) i l'envia als leds (PORTB). La interrupció elimina la rutina d'espera.

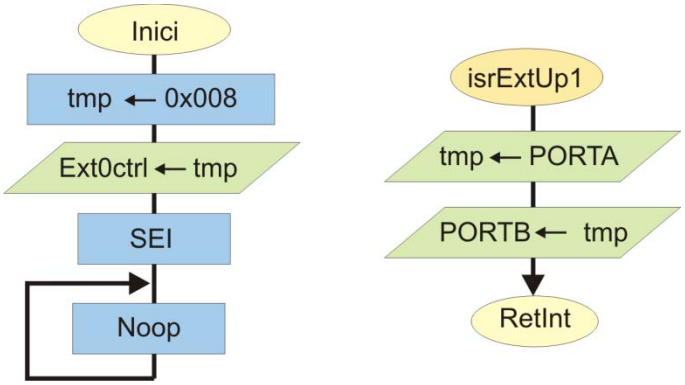

Figura 9.3. Diagrama de flux de l'exemple 1 amb interrupció.

La figura 9.4 mostra el codi assemblador del programa.

-- INTERRUPTOR EN LEDS	
--TREBALLANT AMB INTERRUPCIONS	
.DEF tmp = R16;	
.org 0;	
RJMP inici;	(0, "0100000000011111"), -- 0x401F - rjmp 31
.org 4;	
RJMP isrExtUp1;	(4, "0100000000101101"), -- 0x402D - rjmp 45
NOP;	(5, "0000000000000000"), -- 0x0 - nop
.org 32;	
inici: LDI tmp, 0x008;	(32, "0011000100000000"), -- 0x3100 - ldi r16 0x008
OUT EXT0ctrl, tmp;	(33, "0000000000001000"), -- 0x8
SEI;	(34, "0111100100000011"), -- 0x7903 - out 000011 r16
bucle: NOP;	(35, "0000000100000111"), -- 0x107 - sei
RJMP bucle;	(36, "0000000000000000"), -- 0x0 - nop
NOP;	(37, "0100111111111110"), -- 0x4FFE - rjmp -2
--Rutina de servei d'interrupció	(38, "0000000000000000"), -- 0x0 - nop
.org 50;	
isrExtUp1: IN tmp, PORTA;	(50, "0111000100000000"), -- 0x7100 - in r16 000000
OUT PORTB, tmp;	(51, "0111100100000001"), -- 0x7901 - out 000001 r16
RETI;	(52, "0110000000000101"), -- 0x6005 - reti
NOP;	(53, "0000000000000000"), -- 0x0 - nop
a) Codi assemblador	**b) Codi màquina**

Figura 9.4. Codi assemblador i codi màquina de l'exemple 1 amb interrupcions.

Exemple 2. Treballant amb els 7-segments.

Realitzar un programa que faci rotar un 1 entre 0's en els 7-segments.

El plantejament fet en l'exercici és el següent:

- S'inicialitzen dos registres amb els valors que mostren en els 7-segments un zero i un 1.
- En un bucle es fa rotar l'1 entre zeros en els 7-segments.
- S'introdueix una rutina d'espera de 200ms per veure rotar el 1.

La figura 9.5 mostra el diagrama de flux, mentre que en la figura 9.6 hi ha els codi assemblador i màquina.

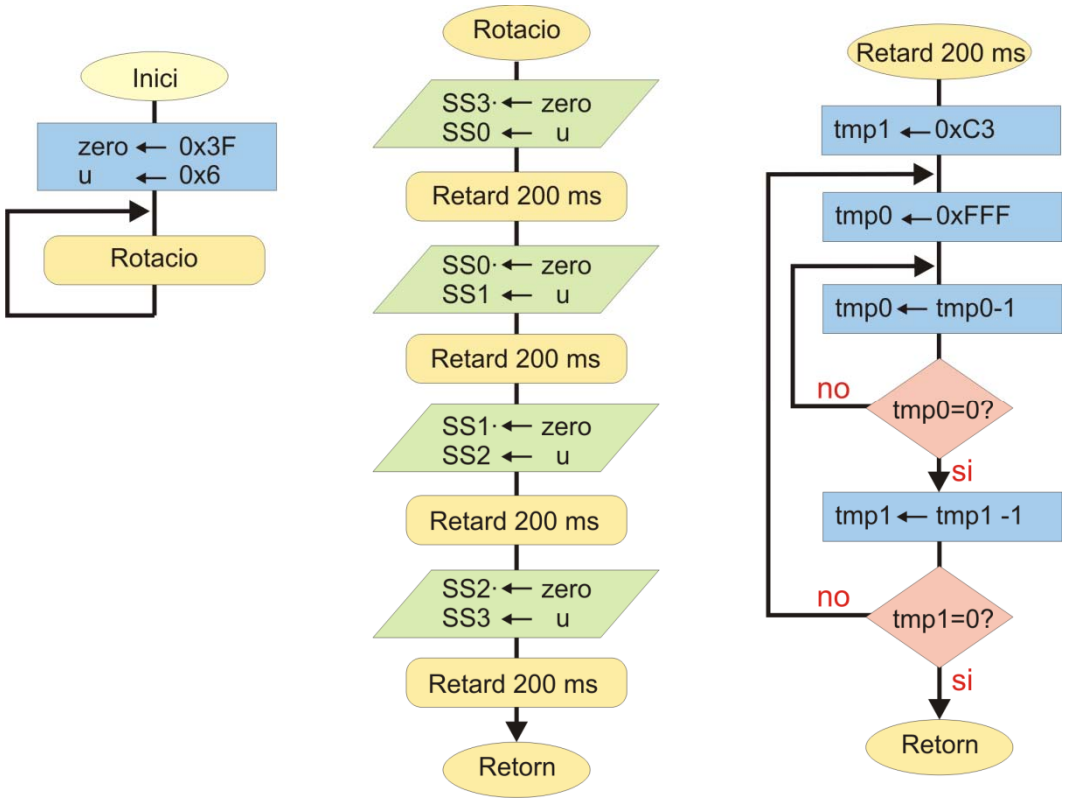

Figura 9.5. Diagrama de flux de l'exemple 2.

-- UN 1 ENTRE 0'S .DEF zero = R0; .DEF u = R1; .DEF tmp0 = R16; .DEF tmp1 = R17; .ORG 0; --Inici programa RJMP main; .ORG 32; --Programa principal main: LDI zero, 0x3F; LDI u, 0x6;	 (0, "0100000000011111"), -- 0x401F - rjmp 31 (32, "0011000000000000"), -- 0x3000 - ldi r0 0x3f (33, "0000000000111111"), -- 0x3F (34, "0011000000010000"), -- 0x3010 - ldi r1 0x6

	(35, "0000000000000110"), -- 0x6
torni: RCALL ss;	(36, "0101000000001101"), -- 0x500D - rcall 13
RJMP torni;	(37, "0100111111111110"), -- 0x4FFE - rjmp -2
.ORG 50; -- Rutina de rotació	
ss: OUT SS3, zero;	(50, "0111100000000111"), -- 0x7807 - out 000111 r0
OUT SS0, u;	(51, "0111100000010100"), -- 0x7814 - out 000100 r1
RCALL ESPERA200;	(52, "0101000000101111"), -- 0x502F - rcall 47
OUT SS0, zero;	(53, "0111100000000100"), -- 0x7804 - out 000100 r0
OUT SS1, u;	(54, "0111100000010101"), -- 0x7815 - out 000101 r1
RCALL ESPERA200;	(55, "0101000000101100"), -- 0x502C - rcall 44
OUT SS1, zero;	(56, "0111100000000101"), -- 0x7805 - out 000101 r0
OUT SS2, u;	(57, "0111100000010110"), -- 0x7816 - out 000110 r1
RCALL ESPERA200;	(58, "0101000000101001"), -- 0x5029 - rcall 41
OUT SS2, zero;	(59, "0111100000000110"), -- 0x7806 - out 000110 r0
OUT SS3, u;	(60, "0111100000010111"), -- 0x7817 - out 000111 r1
RCALL ESPERA200;	(61, "0101000000100110"), -- 0x5026 - rcall 38
RET;	(62, "0110000000000100"), -- 0x6004 - ret
NOP;	(63, "0000000000000000"), -- 0x0 - nop
.ORG 100; --Espera 200 ms	
espera200: LDI tmp1, 0xD0;	(100, "0011000100010000"), -- 0x3110 - ldi r17 0xc3
	(101, "0000000011000011"), -- 0xC3
bucle2: LDI tmp0, 0xfff;	(102, "0011000100000000"), -- 0x3100 - ldi r16 0xfff
	(103, "0000111111111111"), -- 0xFFF
bucle1: DEC tmp0;	(104, "0011000100001001"), -- 0x3109 - dec r16
BRNZ bucle1;	(105, "1111111111110001"), -- 0xFFF1 - brnz -2
DEC tmp1;	(106, "0011000100011001"), -- 0x3119 - dec r17
BRNZ bucle2;	(107, "1111111111010001"), -- 0xFFD1 - brnz -6
RET;	(108, "0110000000000100"), -- 0x6004 - ret
NOP;	(109, "0000000000000000"), -- 0x0 - nop

Figura 9.6. Codi assemblador i codi màquina de l'exemple 2.

Exemple 3. Comunicació amb ordinador emprant el port sèrie

Quan es produeix una interrupció externa sobre polsador 1 en flanc de baixada, el programa envia la dada posada en els interruptors (PortA) cap a l'ordinador.

Per altra part, tota dada rebuda de l'ordinador en el port RxD s'envia als leds (PortB).

El plantejament fet en l'exercici és el següent:

- Prèvia: En la comunicació amb l'hyperterminal de l'ordinador cal recordar que cal enviar caràcters ASCII per un fàcil reconeixement dels mateixos. Per tant, convé enviar els 8 bits com a caràcter ASCII (dos codis hexadecimals). Per exemple, el codi 0x30 representa el zero i vindrà donat pel codi b00110000.
- Els 7-segments s'inicialitzen amb un punt decimal.
- La transmissió s'inicia quan es polsa el botó 1. La rutina de servei d'interrupció agafa la dada dels interruptors i l'envia als leds, als 7-segments i al buffer de sortida de transmissió.

- La recepció es fa per interrupció. Cada cop que es rep una dada per RxD es genera interrupció de recepció de dada. En la rutina de servei de la interrupció s'envia la dada al PORTB (leds) i als 7-segments.
- A tot això el programa principal inicialitza ports i registres i queda en espera d'events.

La figura 9.7 mostra el diagrama de flux, mentre que en la figura 9.8 hi ha els codi assemblador i el codi màquina.

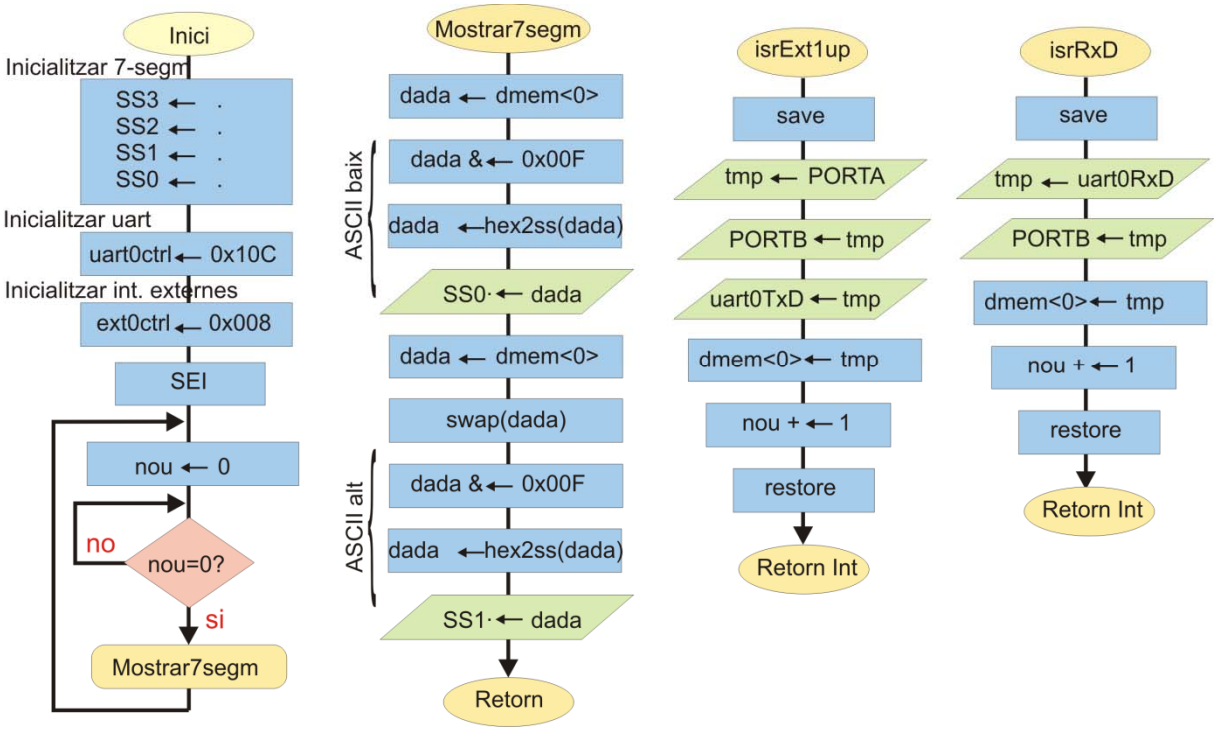

Figura 9.7. Diagrama de flux de l'exemple 3.

-- COMUNICACIONS	
.CON posDada = 0; --Posició de memòria	
.DEF nou = R0;	
.DEF dada = R1;	
.DEF tmp = R16;	
.ORG 0;	
RJMP inici;	
.ORG 4;	(0, "0100000000011111"), -- 0x401F - rjmp 31
RJMP isrExt1up;	--
.ORG 15;	(4, "0100001111100011"), -- 0x43E3 - rjmp 995
RJMP isrRxD;	--
	(15, "0100010000111100"), -- 0x443C - rjmp 1084
.ORG 32; --Programa principal	--
inici: LDI X, hex2ss16;	(32, "0011000111010000"), -- 0x31D0 - ldi r29 1616
	(33, "0000011001010000"), -- 0x650
LPM tmp, X;	(34, "1100001100001100"), -- 0xC30C - lpm r16 r29
OUT SS3, tmp;	(35, "0111100100000111"), -- 0x7907 - out 000111 r16

OUT SS2, tmp;	(36, "0111100100000110"), -- 0x7906 - out 000110 r16
OUT SS1, tmp;	(37, "0111100100000101"), -- 0x7905 - out 000101 r16
OUT SS0, tmp;	(38, "0111100100000100"), -- 0x7904 - out 000100 r16
LDI tmp, 0x10C; --115200baud	(39, "0011000100000000"), -- 0x3100 - ldi r16 0x10c
	(40, "0000000100001100"), -- 0x10C
OUT UART0CTRL, tmp;	(41, "0111110100000000"), -- 0x7D00 - out 100000 r16
LDI tmp, 0x008; --Ext1up	(42, "0011000100000000"), -- 0x3100 - ldi r16 0x008
	(43, "0000000000001000"), -- 0x8
OUT EXT0CTRL, tmp;	(44, "0111100100000011"), -- 0x7903 - out 000011 r16
SEI;	(45, "0000000100000111"), -- 0x107 - sei
bucle: CLR nou;	(46, "0010100000000000"), -- 0x2800 - clr r0
espera: CPI nou, 0;	(47, "0011000000000110"), -- 0x3006 - cpi r0 0
	(48, "0000000000000000"), -- 0x0
BRZE espera;	(49, "1110111111101001"), -- 0xEFE9 - brze -3
RCALL mostrarEn7S;	(50, "0101000000110001"), -- 0x5031 - rcall 49
RJMP bucle;	(51, "0100111111111010"), -- 0x4FFA - rjmp -6
NOP;	(52, "0000000000000000"), -- 0x0 - nop
.ORG 100; --Rutina tractament 7-segm	
mostrarEn7S: LDS dada, posDada;	(100, "1100000000010100"), -- 0xC014 - lds r1 0
	(101, "0000000000000000"), -- 0x0
ANDI dada, 0x00F;	(102, "0011000000010010"), -- 0x3012 - andi r1 0x00f
	(103, "0000000000001111"), -- 0xF
LDI X, hex2ss0;	(104, "0011000111010000"), -- 0x31D0 - ldi r29 1600
	(105, "0000011001000000"), -- 0x640
ADD X, dada;	(106, "0000100111010001"), -- 0x9D1 - add r29 r1
LPM dada, X;	(107, "1100001000011100"), -- 0xC21C - lpm r1 r29
OUT SS0, dada;	(108, "0111100000010100"), -- 0x7814 - out 000100 r1
LDS dada, posDada;	(109, "1100000000010100"), -- 0xC014 - lds r1 0
	(110, "0000000000000000"), -- 0x0
SWAP dada;	(111, "0011001000010000"), -- 0x3210 - swap r1
ANDI dada, 0x00F;	(112, "0011000000010010"), -- 0x3012 - andi r1 0x00f
	(113, "0000000000001111"), -- 0xF
LDI X, hex2ss0;	(114, "0011000111010000"), -- 0x31D0 - ldi r29 1600
	(115, "0000011001000000"), -- 0x640
ADD X, dada;	(116, "0000100111010001"), -- 0x9D1 - add r29 r1
LPM dada, X;	(117, "1100001000011100"), -- 0xC21C - lpm r1 r29
OUT SS1, dada;	(118, "0111100000010101"), -- 0x7815 - out 000101 r1
RET;	(119, "0110000000000100"), -- 0x6004 - ret
NOP;	(120, "0000000000000000"), -- 0x0 - nop
.ORG 1000; --Interrupció TxD	
isrExt1up: SAVE;	(1000, "0110100000000111"), -- 0x6807 - save
IN tmp, PORTA;	(1001, "0111000100000000"), -- 0x7100 - in r16 000000
OUT PORTB, tmp;	(1002, "0111100100000001"), -- 0x7901 - out 000001 r16
OUT UART0TXD, tmp;	(1003, "0111110100000010"), -- 0x7D02 - out 100010 r16
STS posDada, tmp;	(1004, "1101000100000100"), -- 0xD104 - sts 0 r16
	(1005, "0000000000000000"), -- 0x0
INC nou;	(1006, "0011000000001000"), -- 0x3008 - inc r0

RESTORE;	(1007, "0110100000000110"), -- 0x6806 - restore
RETI;	(1008, "0110000000000101"), -- 0x6005 - reti
NOP;	(1009, "0000000000000000"), -- 0x0 - nop
.ORG 1100;	
isrRxD: SAVE; --Interrupció RxD	(1100, "0110100000000111"), -- 0x6807 - save
IN tmp, UART0RXD;	(1101, "0111010100000001"), -- 0x7501 - in r16 100001
OUT PORTB, tmp;	(1102, "0111100100000001"), -- 0x7901 - out 000001 r16
STS posDada, tmp;	(1103, "1101000100000100"), -- 0xD104 - sts 0 r16
	(1104, "0000000000000000"), -- 0x0
INC nou;	(1105, "0011000000001000"), -- 0x3008 - inc r0
RESTORE;	(1106, "0110100000000110"), -- 0x6806 - restore
RETI;	(1107, "0110000000000101"), -- 0x6005 - reti
NOP;	(1108, "0000000000000000"), -- 0x0 - nop
.ORG 1600; --Codis ascii2ss	
hex2ss0: .dw 0x3F;	(1600, "0000000000111111"), -- 0x3F - .dw 0x3f
hex2ss1: .dw 0x6;	(1601, "0000000000000110"), -- 0x6 - .dw 0x6
hex2ss2: .dw 0x5B;	(1602, "0000000001011011"), -- 0x5B - .dw 0x5b
hex2ss3: .dw 0x4F;	(1603, "0000000001001111"), -- 0x4F - .dw 0x4f
hex2ss4: .dw 0x66;	(1604, "0000000001100110"), -- 0x66 - .dw 0x66
hex2ss5: .dw 0x6D;	(1605, "0000000001101101"), -- 0x6D - .dw 0x6d
hex2ss6: .dw 0x7D;	(1606, "0000000001111101"), -- 0x7D - .dw 0x7d
hex2ss7: .dw 0x7;	(1607, "0000000000000111"), -- 0x7 - .dw 0x7
hex2ss8: .dw 0x7F;	(1608, "0000000001111111"), -- 0x7F - .dw 0x7f
hex2ss9: .dw 0x67;	(1609, "0000000001100111"), -- 0x67 - .dw 0x67
hex2ss10: .dw 0x5F;	(1610, "0000000001011111"), -- 0x5F - .dw 0x5f
hex2ss11: .dw 0x7C;	(1611, "0000000001111100"), -- 0x7C - .dw 0x7c
hex2ss12: .dw 0x58;	(1612, "0000000001011000"), -- 0x58 - .dw 0x58
hex2ss13: .dw 0x5E;	(1613, "0000000001011110"), -- 0x5E - .dw 0x5e
hex2ss14: .dw 0x79;	(1614, "0000000001111001"), -- 0x79 - .dw 0x79
hex2ss15: .dw 0x71;	(1615, "0000000001110001"), -- 0x71 - .dw 0x71
hex2ss16: .dw 0x80;	(1616, "0000000010000000"), -- 0x80 - .dw 0x80

Figura 9.8. Codi assemblador i codi màquina de l'exemple 3.

9.4 Resum del capítol.

El capítol ha introduït l'assembladoir com al llenguatge de més baix nivell que simplifica la tasca de programació del processador.

La missió de l'assemblador és la de traduir un programa escrit en les instrucions assemblador del processador a codi màquina. Tot i que el llenguatge és molt simple, convé seguir les normes i consells donades en l'apartat 9.3

Els exemples presentats en l'apartat 9.4 serveixen per veure programes complerts realitzats amb l'assemblador del processador. Convé agafar-los com a guia.