Codi pràctica 6

mis_includes.h

```
#ifndef  MIS_INCLUDES
#define  MIS_INCLUDES

// Definició molt útil per a assignar bits
#define BIT0  0x01
#define BIT1  0x02
#define BIT2  0x04
#define BIT3  0x08
#define BIT4  0x10
#define BIT5  0x20
#define BIT6  0x40
#define BIT7  0x80

// Funció que posa un motor en un estat
void motor_estat (BYTE num_motor, BYTE codi_estat);

// Macros per a la utilització senzilla de la funció anterior
#define pinza_abrir motor_estat(4,0x2)
#define pinza_cerrar motor_estat(4,0x1)
#define pinza_parar motor_estat(4,0x3)
#define rot_antihorario motor_estat(1,0x2)
#define rot_horario motor_estat(1,0x1)
#define rot_parar motor_estat(1,0x3)
#define movh_atras motor_estat(2,0x2)
#define movh_delante motor_estat(2,0x1)
#define movh_parar motor_estat(2,0x3)
#define movv_arriba motor_estat(3,0x2)
#define movv_abajo motor_estat(3,0x1)
#define movv_parar motor_estat(3,0x3)
// Ho para tot
#define parar_todo pinza_parar;rot_parar;movh_parar;movv_parar

Codis de teclat rebuts girats (primer bit rebut és el de menor pes)
#define ABRIR_PINZA       0x00
#define CERRAR_PINZA      0x04
#define DELANTE           0x06
#define ATRAS             0x02
#define ARRIBA            0x01
#define ABAJO             0x05
#define ANTIHORARIO       0x03
#define HORARIO           0x07
#define INICIO            0x0E
#define ALMACENAR         0x09
#define REPRODUCIR        0x0D
#define BORRAR            0x0A

// Estructura que defineix una posició del robot
struct posicio {
   BYTE pos_pinca;
   BYTE pos_horit;
   BYTE pos_vert;
   BYTE rotacio;
};

#define MAX_ALMACENAR 32   // Nombre màxims de posicions que es poden guardar
#define MARGE_LENT  5      // Nombre de passos per a entrar en mode lent
#define MARGE_ERROR 3      // Marge que deixem del final per errors
#define TEMPS_MISSATGE  800  // Temps de mostra dels missatges
```

```
void escriu_frase(BYTE * frase);            // Escriu una frase per l'LCD
void init_lcd();                        // Inicialitza l'LCD

#endif
```

lcd.c

```
#include <embedded.h>
#include "register.h"
#include "typedefs.h"
#include "ports.h"
#include "mis_includes.h"

BYTE valor_P1 = 0x00;      // Valor del port P1
BYTE valor_P0 = 0x00;      // Valor del port P0
DWORD wait_ = 0;
BYTE cont;

// Definim algunes macros d'espera (molt aprox.)
#define ESPERA_40US for(wait_ = 0; wait_ < 20; wait_++){}
#define ESPERA_100US for(wait_ = 0; wait_ < 60; wait_++){}
#define ESPERA_15MS for(wait_ = 0; wait_ < 200; wait_++){}

// Marquem el fi de dada abaixant la senyal Enable i Read/Write
void fi_inst_dada(void) {
    valor_P1 = BIT5;
    SetP0(valor_P0);
}

// Escriu una instrucció al LCD
void escriu_inst_lcd(BYTE inst) {
    // RS = 0, R/W* = 0, E = 1
    valor_P1 = BIT6;
    valor_P0 = inst;

    SetP0(valor_P0);
    SetP1(valor_P1);

    ESPERA_40US    // Fem un pols amb E de uns 40US

    valor_P1 = 0;  // Netegem E però mantenim WRITE
    SetP1(valor_P1);
}

// Escriu una dada al LCD
void escriu_dada_lcd(BYTE dada) {
    // RS = 1, R/W* = 0, E = 1
    valor_P1 = BIT6|BIT4;
    valor_P0 = dada;

    SetP0(valor_P0);
    SetP1(valor_P1);

    ESPERA_40US    // Fem un pols amb E de uns 40US

    valor_P1 = 0;  // Netegem E però mantenim WRITE
    SetP1(valor_P1);
}

// Inicialitzar el LCD, extret del manual
void init_lcd() {
    ESPERA_15MS   // Esperem per garantir l'alimentació Vcc
    escriu_inst_lcd(BIT6|BIT5);   // Enviem la seqüència d'inici 3 cops
    ESPERA_15MS
```

```
    fi_inst_dada();

    escriu_inst_lcd(BIT6|BIT5);
    ESPERA_100US
    fi_inst_dada();

    escriu_inst_lcd(BIT6|BIT5);
    ESPERA_100US
    fi_inst_dada();

    // Activem el mode de dues línies i caràcters grans
    escriu_inst_lcd(BIT6|BIT5|BIT4|BIT3);
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT3);  // Apagar!
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT3|BIT2);  // Encendre!
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT4|BIT2);  // El cursor es desplaça a la dreta
    ESPERA_40US
    fi_inst_dada();

    escriu_inst_lcd(BIT1);  // Situem el cursor a l'inici
    ESPERA_40US
    fi_inst_dada();
}

// Escriu una frase sencera al LCD substituint l'anterior
// Escriurem espais en blanc si la frase no ocupa tot l'LCD
void escriu_frase(BYTE * frase) {
    BYTE maux;
    escriu_inst_lcd(BIT1);      // Cursor a l'inici
    ESPERA_40US
    fi_inst_dada();
    maux = 0;
    for (cont = 0; cont < 16; cont++) {
        if (frase[cont] == 0x00 || maux==1) {  // Fi de la frase
              escriu_dada_lcd(' ');
              maux = 1;
        }else{
              escriu_dada_lcd(frase[cont]);
        }
        ESPERA_40US
        fi_inst_dada();
    }
}

// Conversió de integer a character
void itoa(BYTE * cad, BYTE num, BYTE digit) {
   BYTE cont=digit-1;
   do {
         cad[cont] = (num%10)+'0';
         num = num /10;
         cont--;
   } while (cont != 0xFF) ;
}

// Funció que escriu a la pantalla les diferents posicions del robot de la
// forma R000H000V000P000
BYTE buffer[17];
void escriu_valors(struct posicio * pos) {
```

```
   buffer[0] = 'R';
   buffer[0+4] = 'H';
   buffer[0+8] = 'V';
   buffer[0+12] = 'P';
   itoa(&buffer[1],pos->rotacio,3);
   itoa(&buffer[1+4],pos->pos_horit,3);
   itoa(&buffer[1+8],pos->pos_vert,3);
   itoa(&buffer[1+12],pos->pos_pinca,3);
   escriu_frase(buffer);
}

// Escriu una frase seguida d'un enter, similar a fer
// printf ("Frase %d",enter);
void escriu_frase_integer (BYTE * frase, BYTE enter) {
   cont=0;
   while (frase[cont] != 0) {
         buffer[cont] = frase[cont];
         cont++;
   }
   itoa(&buffer[cont],enter,16-cont);
   escriu_frase(buffer);
}
```

ports.c

```
#include <embedded.h>
#include "register.h"
#include "typedefs.h"
#include "ports.h"
#include "mis_includes.h"

// Inicialització dels ports P0,P1,P2,PT
void IniPorts(void) {
   BYTE far *pbyReg; //punter a un registre
   // Port P2
   pbyReg = (BYTE far *)MK_FP(RSEG, PMC2);
   *pbyReg = 0x00;   //PMC2 = '00' -> mode port I/O
   --pbyReg;         //apuntar a PM2
   *pbyReg = 0x00;   //PM2 = '00' -> port 2 de sortida
   --pbyReg;         //apuntar a P2
   *pbyReg = 0x00;   //P2 = '00' -> port 2 valor inicial

   // Port 1
   pbyReg = (BYTE far *)MK_FP(RSEG, PMC1);
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;

   // Port 0
   pbyReg = (BYTE far *)MK_FP(RSEG, PMC0);
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;
   --pbyReg;
   *pbyReg = 0x00;

   //config PT
   pbyReg = (BYTE far *)MK_FP(RSEG, PMT);
   *pbyReg = 0x08;   //llindar entrades PT a 1.25V
}

// Funcions per llegir/escriure els diferents ports
void SetP2(BYTE byVal) {
```

```
   BYTE far *pbyP2;
   pbyP2 = (BYTE far *)MK_FP(RSEG, P2);
   *pbyP2 = byVal;
}

void SetP1(BYTE byVal) {
   BYTE far *pbyP1;
   pbyP1 = (BYTE far *)MK_FP(RSEG, P1);
   *pbyP1 = byVal;
}

void SetP0(BYTE byVal) {
   BYTE far *pbyP0;
   pbyP0 = (BYTE far *)MK_FP(RSEG, P0);
   *pbyP0 = byVal;
}

BYTE GetPT(void) {
   BYTE far *pbyPT;
   pbyPT = (BYTE far *)MK_FP(RSEG, PT);
   return(*pbyPT);
}
```

robot.c

```
#include <embedded.h>
#include "typedefs.h"
#include "register.h"
#include "timers.h"
#include "ports.h"
#include "mis_includes.h"

BYTE estat_P2 = 0x00;
BYTE reset = 1;
BYTE no_show = 0;
extern BYTE cicle_treball[4];
extern int contador_lcd;

BYTE reproduir = 0;  // Si es 0, mode normal, 1 mode reproduccio!

// S'actualitza sola indicant la posició del robot!
// Cal posarla a 0 al fer un reset!
struct posicio posicio_actual;
// Llista de posicions a guardades
struct posicio vector_posicions[MAX_ALMACENAR];
// Indica la pròxima posició que es guardara al enviar la tecla
BYTE proxima_posicio_gravar = 0;
// Per a esborrar les dades només cal posar-la a 0.
// Si és 0 també sabem que no hi ha res guardat.

BYTE paraula_moviment;
// Indica l'estat dels motors a cada instant al reproduir
BYTE cont_pinca = 0, cont_posh = 0, cont_posv = 0, cont_rot = 0;
// Valors dels pins compta-passos, necessaris per detectar flancs
BYTE moviment_1 = 0, moviment_2 = 0, moviment_3 = 0, moviment_4 = 0;
// Serà 1 o -1 segons el sentit del moviment

// Funció que calcula la paraula d'estat dels motors
BYTE calc_paraula_estat (BYTE num_motor, BYTE codi_estat, BYTE paraula) {
   BYTE mascara = (0x3 << 2*(num_motor-1));
   paraula &= ~(mascara);  // Netejem l'estat! El posem a 00
   // Enmascarem el codi per si de cas for incorrecte!
   paraula |= ( (codi_estat & 0x3) << 2*(num_motor-1) );
   return paraula;
```

```
}

// Actualitza l'estat d'un motor
void motor_estat (BYTE num_motor, BYTE codi_estat) {
   estat_P2=calc_paraula_estat(num_motor,codi_estat,estat_P2);
   SetP2(estat_P2);
}

// Funció que acaba quan el robot es troba a l'origen
void robot_inici() {
   BYTE status;
   pinza_abrir; rot_antihorario; movh_atras; movv_arriba;
   status = GetPT();
   // Esperem mentre no estigui a l'origen
   while ( (status & 0x01) != 0 ) {status = GetPT();}
   parar_todo;

   posicio_actual.pos_pinca = 0;
   posicio_actual.pos_horit = 0;
   posicio_actual.pos_vert = 0;
   posicio_actual.rotacio = 0;
}

void main() {
   BYTE i,temp;

   // Important iniciar els ports de I/O correctament
   IniPorts();
   // Inicialitza el LCD
   init_lcd();
   // Inicialitzem el port serie
   InicialitzaSerie();
   // Parem el robot
   parar_todo;
   // Activem els timers
   IniTimers();
   enable(); // Permetem interrupcions!

   while (1) {

   // Escribim els valors actuals si no_show==0
         if (no_show == 0) escriu_valors(&posicio_actual);

   // Es vol fer un reset!
         if (reset == 1) {
               escriu_frase("Robot: reset");
         disable();  // Parem interrupcions per a fer el reset, important!
         robot_inici();
         reset = 0;
         enable();   // Tornem al mode habitual
   }

   if (reproduir == 1) {  // Mode reproducció!
         escriu_frase("Robot: reproduir");
         // Comprovar que hi hagi alguna cosa a la llista de reproduccions
         if (proxima_posicio_gravar != 0) {
               parar_todo;  // Parem, resetegem i comencem a reproduir
               disable(); robot_inici(); enable();
               posicio_actual.pos_pinca = 0;
               posicio_actual.pos_horit = 0;
               posicio_actual.pos_vert = 0;
               posicio_actual.rotacio = 0;

               for (i = 0; i < proxima_posicio_gravar; i++) {
                     // Anem escrivint el moviment que estem fent
                     escriu_frase_integer("Moviment: ",i+1);
```

```
                        temp = 0x00;  // Variable que indica el moviment
                        while (!(temp == 0xFF)) {
                              // Ens quedem recalculant la paraula d'estat
                              // mentre no arribem  la següent posició
                              temp = 0xFF;
                              // Calcular com han d'estar els motors i revisar
                              // si s'ha assolit la posició!

                              if (vector_posicions[i].pos_pinca >
posicio_actual.pos_pinca) {  // Tancar la pinça
                                    moviment_4 = 1;
                                    temp=calc_paraula_estat(4,0x1,temp);
                              }else if (vector_posicions[i].pos_pinca <
posicio_actual.pos_pinca) {
                                    moviment_4 = -1;
                                    temp=calc_paraula_estat(4,0x2,temp);
                              }

                              if (vector_posicions[i].rotacio >
posicio_actual.rotacio) {  // Moviment horari
                                    moviment_1 = 1;
                                    temp=calc_paraula_estat(1,0x1,temp);
                              }else if (vector_posicions[i].rotacio <
posicio_actual.rotacio) {
                                    moviment_1 = -1;
                                    temp=calc_paraula_estat(1,0x2,temp);
                              }

                              if (vector_posicions[i].pos_horit >
posicio_actual.pos_horit) {  // Endevant
                                    moviment_2 = 1;
                                    temp=calc_paraula_estat(2,0x1,temp);
                              }else if (vector_posicions[i].pos_horit <
posicio_actual.pos_horit) {
                                    moviment_2 = -1;
                                    temp=calc_paraula_estat(2,0x2,temp);
                              }

                              if (vector_posicions[i].pos_vert >
posicio_actual.pos_vert) {  // Avall
                                    moviment_3 = 1;
                                    temp=calc_paraula_estat(3,0x1,temp);
                              }else if (vector_posicions[i].pos_vert <
posicio_actual.pos_vert) {
                                    moviment_3 = -1;
                                    temp=calc_paraula_estat(3,0x2,temp);
                              }

                              // Mirem a quina velocitat hem d'anar
                              if (abs(vector_posicions[i].rotacio -
posicio_actual.rotacio) < MARGE_LENT) cicle_treball[0] = 4;
                              else cicle_treball[0] = 1;
                              if (abs(vector_posicions[i].pos_horit -
posicio_actual.pos_horit) < MARGE_LENT) cicle_treball[1] = 4;
                              else cicle_treball[1] = 1;
                              if (abs(vector_posicions[i].pos_vert -
posicio_actual.pos_vert) < MARGE_LENT) cicle_treball[2] = 4;
                              else cicle_treball[2] = 1;
                              if (abs(vector_posicions[i].pos_pinca -
posicio_actual.pos_pinca) < MARGE_LENT) cicle_treball[3] = 4;
                              else cicle_treball[3] = 1;

                              paraula_moviment = temp;
                        }
                  }
            }else{
```

```
                // Cas de reproducció però no hem guardat res
                contador_lcd=0;
                no_show = 1;
                escriu_frase("Res a reproduir!");
          }
   reproduir = 0;
   }

   }  // while (1)

}
```

timers.c

```
#include <embedded.h>
#include "typedefs.h"
#include "int.h"
#include "register.h"
#include "ports.h"
#include "timers.h"
#include "mis_includes.h"

// Variables globals
int n_codivalid = 0;
int contador = 0;
int encendido = 0;
int contador_lcd = 0;
BYTE posicion = 1;
extern BYTE no_show;

void interrupt RSITimer0(void);
void interrupt RSITimer1(void);
void interrupt RSIRecepcio0 (void);

BYTE far *pbyReg;
BYTE cicle_treball[4];
BYTE cont_treball[4];

extern BYTE moviment;
extern BYTE reset;
extern BYTE reproduir;
extern struct posicio posicio_actual;
extern struct posicio vector_posicions[32];
extern BYTE proxima_posicio_gravar;
extern BYTE paraula_moviment;
extern BYTE cont_pinca, cont_posh , cont_posv , cont_rot ;
extern BYTE moviment_1, moviment_2, moviment_3, moviment_4;
BYTE almacenado = 0;


// Inicialització dels temporitzadors
void IniTimers(void) {
   BYTE far *pbyReg; //punter a un registre
   WORD far *pwoReg; //punter a un registre

   disable();        // no permetre cap interrupció

   setvect(INTBASETEMPS, RSIBaseTemps);
   pbyReg = (BYTE far *)MK_FP(RSEG, PRC);
   *pbyReg = 0x04;         //Fclk de 4 MHz
   pbyReg = (BYTE far *)MK_FP(RSEG, TBIC);
   *pbyReg &= 0xBF;  //activar interrupcions BT

   // Iniciarel timer 0
```

```
      pwoReg = (WORD far *)MK_FP(RSEG, MD1);
      *pwoReg = 512;

      pbyReg = (BYTE far *)MK_FP(RSEG, TMC0);
      *pbyReg = 0x80 | 0x40;

      setvect(INTCOUNTER0, RSITimer0);

      pbyReg = (BYTE far *)MK_FP(RSEG, TMIC0);
      *pbyReg &= 0xBF;  //activar interrupcions timer0

      // Iniciarel timer 1
      pwoReg = (WORD far *)MK_FP(RSEG, MD1);
      *pwoReg = 128;

      pbyReg = (BYTE far *)MK_FP(RSEG, TMC1);
      *pbyReg = 0x80 | 0x40;

      setvect(INTCOUNTER2, RSITimer1);

      pbyReg = (BYTE far *)MK_FP(RSEG, TMIC2);
      *pbyReg &= 0xBF;  //activar interrupcions timer1

      enable();
   }

   void InicialitzaSerie() {
      BYTE far *pbyReg;

      disable();

      pbyReg = (BYTE far *)MK_FP(RSEG, SCM0);   // Registre de descripció del
senyal
      *pbyReg = BIT3 | BIT6 | BIT0;
      // 8 bits de dades + recepció enabled  (no paritat , 1 bit stop)
      // MOLT IMPORTANT el BIT0 per activar mode ASINCRON

      pbyReg = (BYTE far *)MK_FP(RSEG, SCE0);   // Registre de control de errors
      *pbyReg = 0;  // Tots els errors capats + NO transmissió

      // Registre de control de d'interrupció d'errors
      pbyReg = (BYTE far *)MK_FP(RSEG, SEIC0);
      *pbyReg = 0;  // No volem petició d'error

      // Registre de control de d'interrupció de transmissió completada
      pbyReg = (BYTE far *)MK_FP(RSEG, SRIC0);
      *pbyReg = 0;  // No volem petició d'error

      // Registre de control de d'interrupció de recepció completada.
      pbyReg = (BYTE far *)MK_FP(RSEG, SRIC0);
      *pbyReg = BIT7;  // Volem petició

      pbyReg = (BYTE far *)MK_FP(RSEG, BRG0);
      *pbyReg = 15;  // 512bauds

      pbyReg = (BYTE far *)MK_FP(RSEG, SCC0);
      *pbyReg = 0x08;  // 512bauds

      setvect(INTSRX0, RSIRecepcio0);
      setvect(INTSERR0, RSIRecepcio0);
      setvect(INTSTX0, RSIRecepcio0);
      enable();
   }

   BYTE decide_signo(BYTE valor, BYTE medio) {
      if(valor > medio) return -1;
```

```
      return 1;
   }

   void interrupt RSIRecepcio0 (void) {
      BYTE j,dada;
      disable();

      if (reproduir == 0) {

            // Tractar recepció de dades! La dada esta a RxB0
            pbyReg = (BYTE far*)MK_FP(RSEG,  RxB0);   // Dada!
            dada = *pbyReg;
            if ( (dada & 0xF0) == 0xF0) {  // Els 4 primers bits d'stop
                  dada = dada & 0x0F;
                  moviment_1 = 0;moviment_2 = 0; moviment_3 = 0; moviment_4 = 0;
                  switch (dada) {
                  case ABRIR_PINZA:
                        moviment_4 = -1; almacenado=0;
                        parar_todo; pinza_abrir; break;
                  case CERRAR_PINZA:
                        moviment_4 = 1; almacenado=0;
                        parar_todo; pinza_cerrar; break;
                  case DELANTE:
                        moviment_2 = 1; almacenado=0;
                        parar_todo; movh_delante; break;
                  case ATRAS:
                        moviment_2 = -1; almacenado=0;
                        parar_todo; movh_atras; break;
                  case ARRIBA:
                        moviment_3 = -1; almacenado=0;
                        parar_todo; movv_arriba; break;
                  case ABAJO:
                        moviment_3 = 1; almacenado=0;
                        parar_todo; movv_abajo; break;
                  case ANTIHORARIO:
                        moviment_1 = -1; almacenado=0;
                        parar_todo; rot_antihorario; break;
                  case HORARIO:
                        moviment_1 = 1; almacenado=0;
                        parar_todo; rot_horario; break;
                  case INICIO:
                        almacenado=0; reset = 1; parar_todo; break;
                  case REPRODUCIR:
                        almacenado=0; paraula_moviment=0xFF; reproduir = 1;
                        parar_todo; break;
                  case ALMACENAR:
                        if (almacenado==1) break;
                        almacenado=1;
                        if (proxima_posicio_gravar >= MAX_ALMACENAR) {
                              contador_lcd=0;
                              no_show = 1;
                              escriu_frase("Cua plena!");
                              break;  // No n'hi caben més!
                        }
                        vector_posicions[proxima_posicio_gravar] =
posicio_actual;   // Guardar la posició
                        vector_posicions[proxima_posicio_gravar].rotacio +=
decide_signo(vector_posicions[proxima_posicio_gravar].rotacio,115)*MARGE_ERROR;
                        vector_posicions[proxima_posicio_gravar].pos_pinca +=
decide_signo(vector_posicions[proxima_posicio_gravar].pos_pinca,15)*MARGE_ERROR;
                        vector_posicions[proxima_posicio_gravar].pos_horit +=
decide_signo(vector_posicions[proxima_posicio_gravar].pos_horit,90)*MARGE_ERROR;
                        vector_posicions[proxima_posicio_gravar].pos_vert +=
decide_signo(vector_posicions[proxima_posicio_gravar].pos_vert,55)*MARGE_ERROR;
                        proxima_posicio_gravar++;
                        contador_lcd=0;
```

```
                        no_show = 1;
                        escriu_frase("Moviment guardat");
                        parar_todo; break;
                  case BORRAR:
                        almacenado=0;
                        proxima_posicio_gravar = 0;
                        contador_lcd=0;
                        no_show = 1;
                        escriu_frase("Tot borrat");
                        parar_todo; break;
                  default:
                        goto CODI_NO_VALID;
                        break;
                  }
                  // Codi valid! Seguir el moviment!
                  n_codivalid = 0;
            }
      }
   CODI_NO_VALID:
      enable();
      FINT;
   }

   // Servei d'interrupció de la base de temps
   void interrupt RSIBaseTemps(void) {
      disable();

      contador_lcd++;
      if (contador_lcd > TEMPS_MISSATGE) no_show=0;
      enable();

      FINT;
   }

   // Cicle de treball variable segons l'array cicle_treball
   void interrupt RSITimer1(void) {
      BYTE temp2,j;
      disable();  //permetre altres INT.

      if (reproduir == 1) {
            // Modular la sortida amb tren de polsos
            temp2 = paraula_moviment;
            for (j = 0; j < 4; j++) {
               if (cont_treball[j] != 0) {
                  temp2 |= (0x3 << (2*j));
               }
            }
            SetP2(temp2);
            for (j = 0; j < 4; j++) cont_treball[j] = (cont_treball[j]+1) %
cicle_treball[j];
      }
      enable();
      FINT;
   }

   void interrupt RSITimer0(void) {
      BYTE dada;
      disable();

      /// Comptapassos mostrejat a la freq del timer
      dada = GetPT();

      if ( ((dada & BIT1) == 0) && cont_rot == 1) { cont_rot=0;
posicio_actual.rotacio+=moviment_1; }
      if ( ((dada & BIT1) != 0) && cont_rot == 0) { cont_rot=1;
posicio_actual.rotacio+=moviment_1; }
```

```
      if ( ((dada & BIT2) == 0) && cont_posh == 1) { cont_posh=0;
posicio_actual.pos_horit+=moviment_2; }
      if ( ((dada & BIT2) != 0) && cont_posh == 0) { cont_posh=1;
posicio_actual.pos_horit+=moviment_2; }

      if ( ((dada & BIT3) == 0) && cont_posv == 1) { cont_posv=0;
posicio_actual.pos_vert+=moviment_3; }
      if ( ((dada & BIT3) != 0) && cont_posv == 0) { cont_posv=1;
posicio_actual.pos_vert+=moviment_3; }

      if ( ((dada & BIT4) == 0) && cont_pinca == 1) { cont_pinca=0;
posicio_actual.pos_pinca+=moviment_4; }
      if ( ((dada & BIT4) != 0) && cont_pinca == 0) { cont_pinca=1;
posicio_actual.pos_pinca+=moviment_4; }

      if (reproduir == 0) {  // Només si estem en mode normal
            // Aquesta rutina s'encarrega de parar el
            // motor si no s'ha apretat cap tecla!
            if (n_codivalid >= 2) {
                  parar_todo;  // to's quietos
            }else{
                  n_codivalid++;
            }
      }

      enable();
      FINT;
   }
```