TEMA 1
Programación de uC en C

Introducción

Sistema empotrado :

- Tiene hardware único, diseñado para la función a desarrollar.
- Tiene un software único, dirigido a realizar solo una tarea especifica.
- Tiene que ser efectivo en coste.

Historia :

- 4004 de INTEL en 1971, solución a problemas similares.
- Microprocesadores de 4 y 8 bits.
- Microcontroladores de 16 y 32 bits.

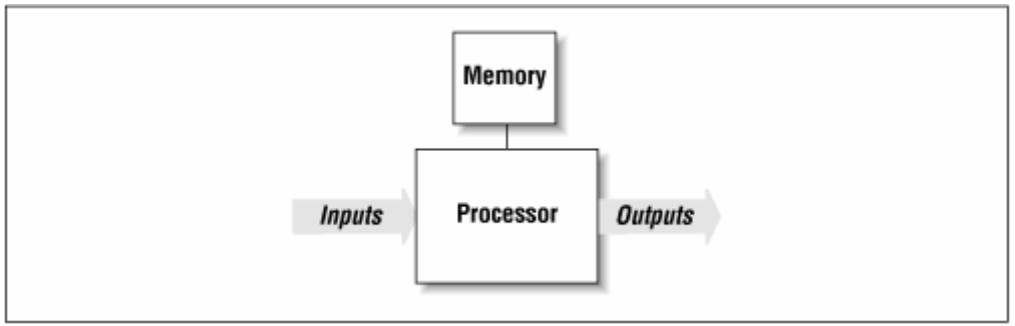
Sistemas en Tiempo Real:

- Integrados por microcontroladores.
- Deben responder en un tiempo preestablecido en cualquier caso.

Estructura básica

El sistema empotrado contiene el microcontrolador, que a su vez dispone de :

- Memoria.
- CPU.
- Entradas y Salidas (Periféricos). Existe especialización:
  - Puertos.
  - Temporizadores/Contadores.
  - Interrupciones.
  - Extensión del Bus Interno.



El resto del hardware es único para el problema que tengamos

Necesidades del diseño del sistema empotrado

1. Potencia de procesamiento.
2. Cantidad de memoria.
3. Costes de desarrollo.
4. Número de unidades que se van a comercializar.
5. Tiempo de vida esperado del producto.
6. Fiabilidad.

Criterio	Bajo	Medio	Alto
Procesamiento	4 u 8 bits	16 bits	32 o 64 bits
Memoria	< 16 KB	64 KB a 1MB	> 1MB
Costo desarrollo	< 100K eur	100K a 1M eur	> 1M eur
Costo de producción	< 10 eur	10 a 1K eur	> 1K eur
Núm de unidades	< 100	100 – 10.000	> 10.000
Tiempo de vida	Dias, semanas o meses	Años	Decadas
Fiabilidad	Puede fallar	Debe ser fiable	a prueba de fallos

VENTAJAS DEL EMPLEO DE C

1. Lenguaje más usado en sistemas.
2. Existencia de compiladores para todos uC.
3. Sencillo y compacto.
4. Independencia del procesador empleado.
5. Fácil de encontrar algoritmos ya desarrollados.
6. Posibilidad de actuar sobre controles de bajo nivel (orientado a operaciones de bajo nivel).

PRIMER PROGRAMA EN C

Al trabajar con un nuevo lenguaje de programación se realiza un pequeño programa sencillo para comenzar a programar. Este consiste en sacar una cadena de texto por consola. No es posible si el sistema empotrado no tiene consola.

En su lugar emplearemos un LED parpadeante:

```
/*****************************************************************
 *
 * Funcion:    main()
 *
 * Descripcion: Parpadea LED verde una vez por segundo.
 *
 * Notas:       El bucle no depende del hardware.  Sin embargo,
 *              depende de dos funciones dependientes del hardware.
 *
 * Devuelve:    Contiene un bucle infinito.
 *
 *****************************************************************/
void main(void)
{
    while (1) {
        toggleLed(LED_GREEN);       /* Cambia el estado del LED.  */
        delay(500);                 /* Pausa 500ms.     */
    }

}   /* main() */
```

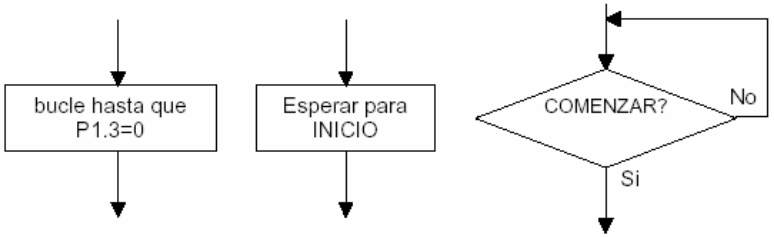
FUNCIONES PARA MEJORAR LA PORTABILIDAD DEL PROGRAMA

```
#include <reg51.h>      /* Incluye la direccion del registro PORT2. */

/*****************************************************************
 *
 * Funcion:    toggleLed()
 *
 * Descripcion: Cambia el estado de uno o varios LEDs del puerto P2.
 *
 * Notas:       Esta funcion es especifica a la placa 80C537.
 *
 * Devuelve:     Nada.
 *
 *****************************************************************/

void
toggleLed(unsigned char ledMask)
{
   P2=P2&ledMask;  /* Cambia los bits pedidos.       */
                   /* y escribe valor nuevo del registro.*/
}   /* toggleLed() */
```

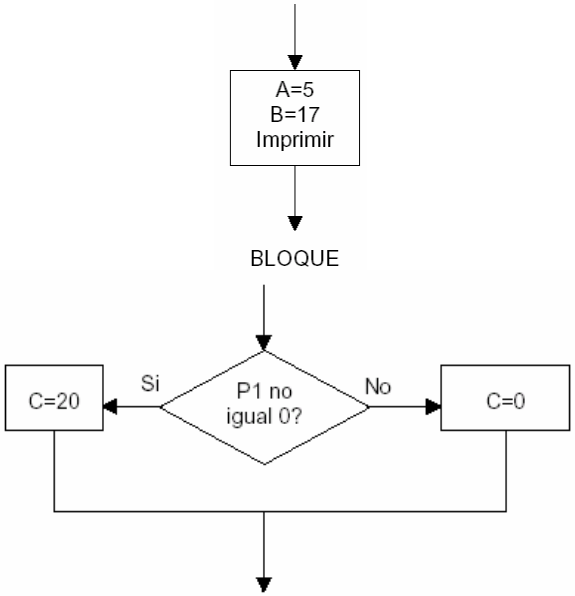
TEMPORIZACIÓN ACTIVA

```
/*********************************************************************
 *
 * Funcion:    delay()
 *
 * Descripcion: Espera activa durante los ms pedidos.
 *
 * Notas:       El numero de ciclos por milisegundos de decremento
 *              y test fue determinado mediante prueba y error.  Este
 *              valor depende del tipo de procesador y velocidad
 *
 * Devuelve:     Nada.
 *
 *********************************************************************/
void
delay(unsigned int nMilliseconds)
{
    #define CICLOS_POR_MS 260 /* Ciclos de decremento y test. */

    unsigned long nCycles = nMilliseconds * CICLOS_POR_MS;

    while (nCycles--);

}   /* delay() */
```

Diagramas de Flujo – Estructuras básicas

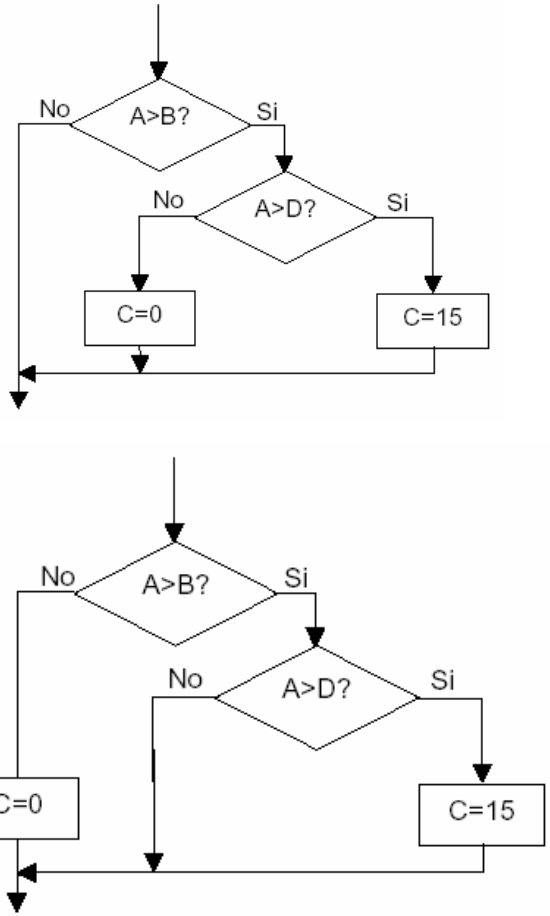



```
{ /* estructura básica de bloque */
a=5;
b=17;
print;
}
{ /* estilos alternativos de bloque */
A=5; B=17; print();
}
{A=5; B=17; print();}
```



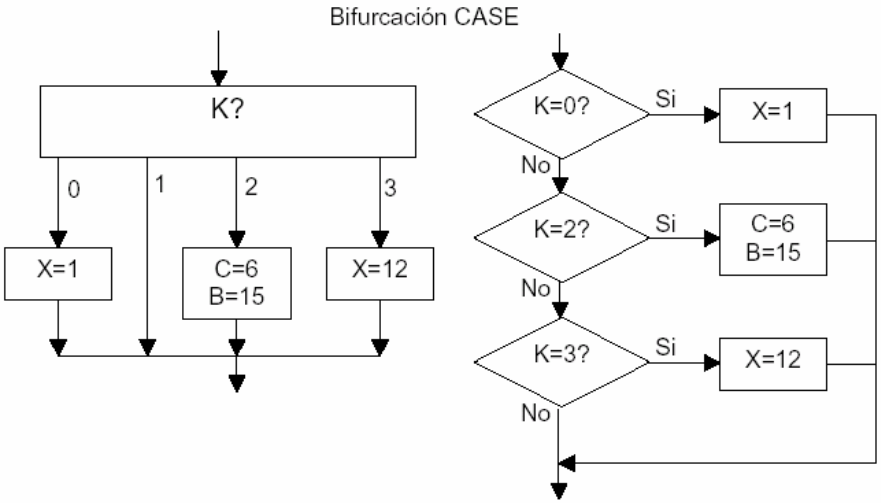

```
/* decisión if/else */
if (P1!=0) c=20;
else c=0;
```

Bifurcación



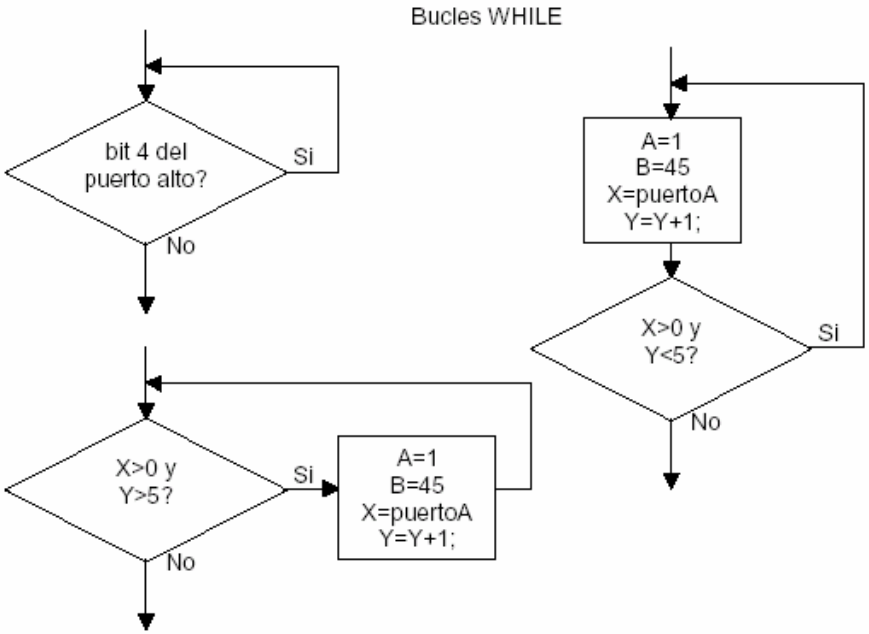

```
/* bloques if/else anidados */
/* versiones 1 y 2 */
if (a>b) {
if (a>d) c=15;
else c=0;
}
/* ver2 */
if (a>b)
if (a>d) c=15;
else c=0;
/* ver3 */
if (a>b) {
if (a>d) c=15;
}
else c=0;
```

Bifurcación Case



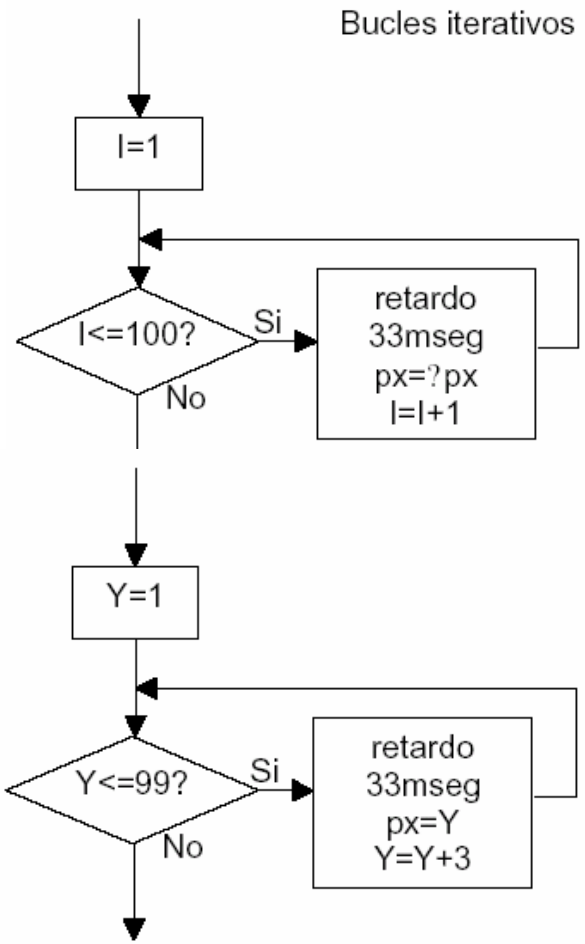

```
/* bifurcación case básica */
switch (k) {
case 0:
x=1;
break;
case 2:
c=6;
b=15;
break;
case 3: x=12;
break;
default: break;
}
```

Bucles WHILE



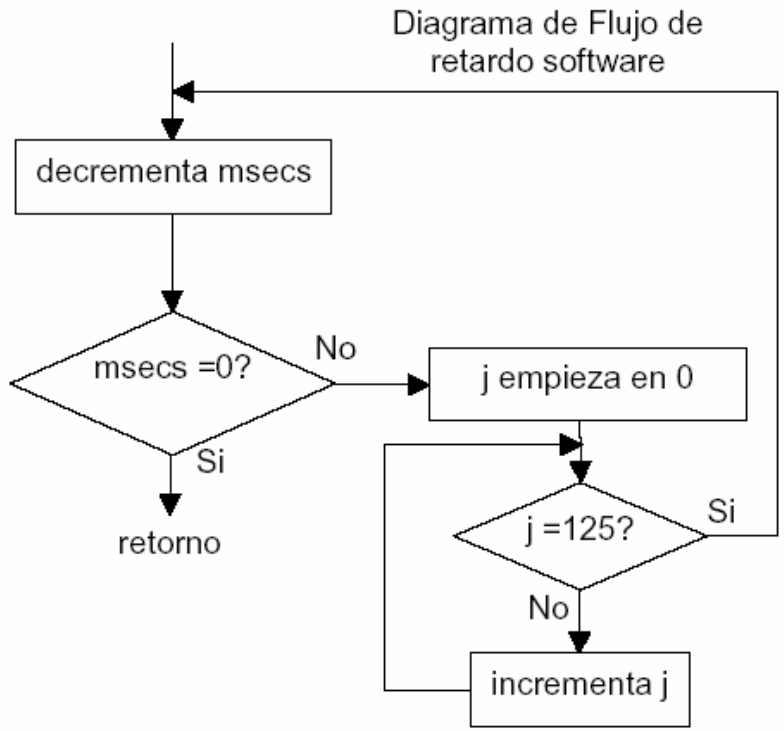

```
/* bucle while simple (vacio) */
while ((P1 & 0x10)==0);

/* bucle while normal */
while (x>0 && y++==5) {
a=1;
b=45;
x=P1;
}

/* bucle do while */
do {
a=1;
b=45;
x=P1;
}
while (x>0 && y++==5);
```

Bucle Iterativo




```
/* estructuras de bucles iterativos */
for (i=1;i<=100;i++) {
retardo(33);
px=?px;
}
/* **** */
for (y=0;y<=99;y=y+3) {
retardo(33);
px=y;
}
/* **** */
for (da=inicio;estado==ocupado;leds=?leds) {
retardo(33);
}
```

Ejemplo: Retardo de Tiempo




```
/* retardo mediante bucle software */
void msec(unsigned int x) {
unsigned char j;

while (x-- > 0) {
for (j=0;j<125;j++){;}
  }
}
```

Estructura Programa basado en pooling

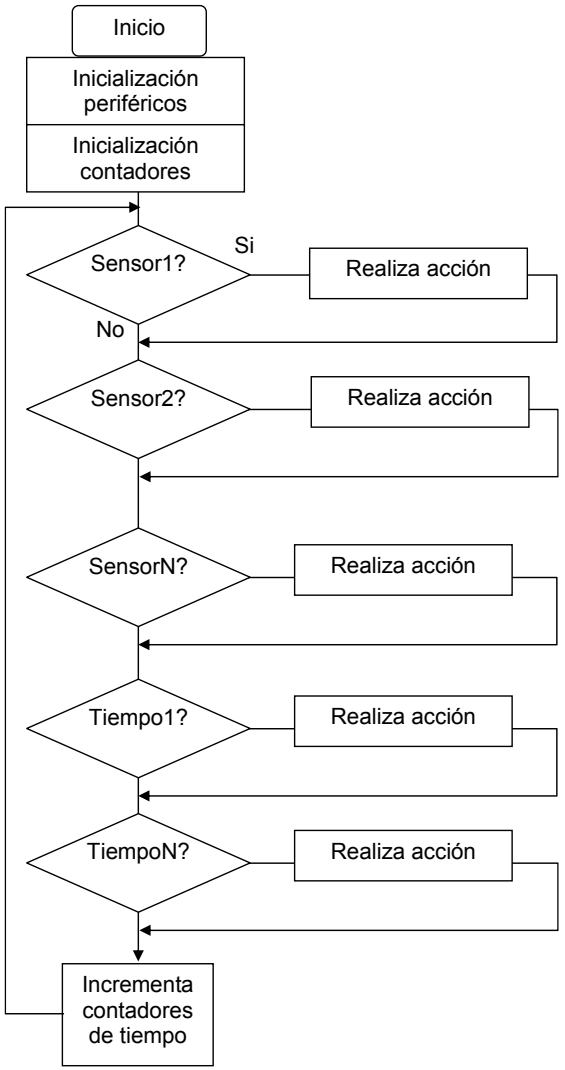


Diagrama de Flujo de programa tipo

- Inicialización de variables y sensores.
- Inicialización de contadores.

- Bucle para la encuesta de sensores.
- Realiza la acción de inmediato, la acción debe se rápida para no dejar de atender a los demás sensores.
- Si se programa una acción pasado un tiempo se lanza un temporizador (variable software).

- Testeo de temporizadores expirados. Si expira una variable de tiempo software se realiza la acción.

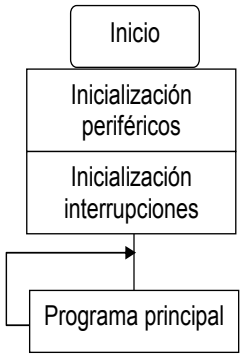
- Incremento de los temporizadores si hay alguno funcionando.

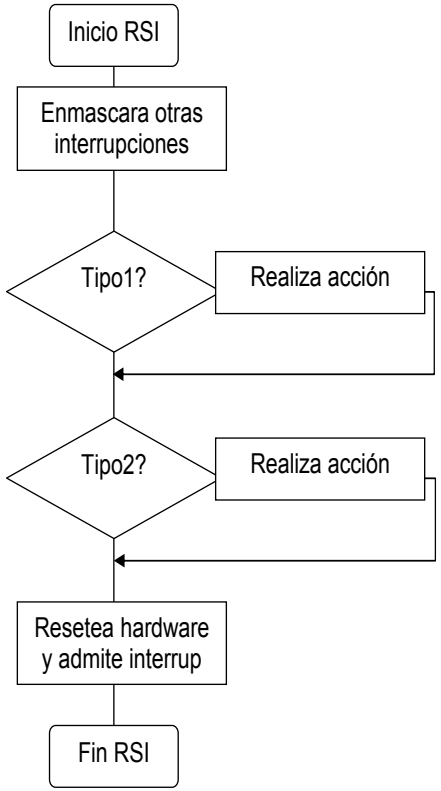
Programa basado en pooling

```
/* Reloj digital empleando temporización por bucle */

#include <16f876.h>

/* Función que introduce un retardo de un segundo aproximadamente */
void retardo(void) {
unsigned char i,j,k;

     for (i=0;i<100;i++)
          for (j=0;j<200;j++)
               for (k=0;k<200;k++);
}

void main(void) {
unsigned char segundos,minutos,horas;

     /* Inicializa reloj */
     segundos=minutos=horas=0;

     while (1) {
          segundos++;
          if (segundos >= 60) { segundos=0;minutos++; }
          if (minutos >= 60) { minutos=0;horas++; }
          if (horas >= 24) horas=0;
          retardo();
          muestra_display(horas,minutos,segundos);
     }
}
```

Estructura Programa basado en interrupción

Diagramas de Flujo
de programa tipo
usando interrupciones





- Inicialización de variables y sensores.
- Inicialización de interrupciones.

- Bucle para realizar el programa principal (puede ser cualquier cosa).

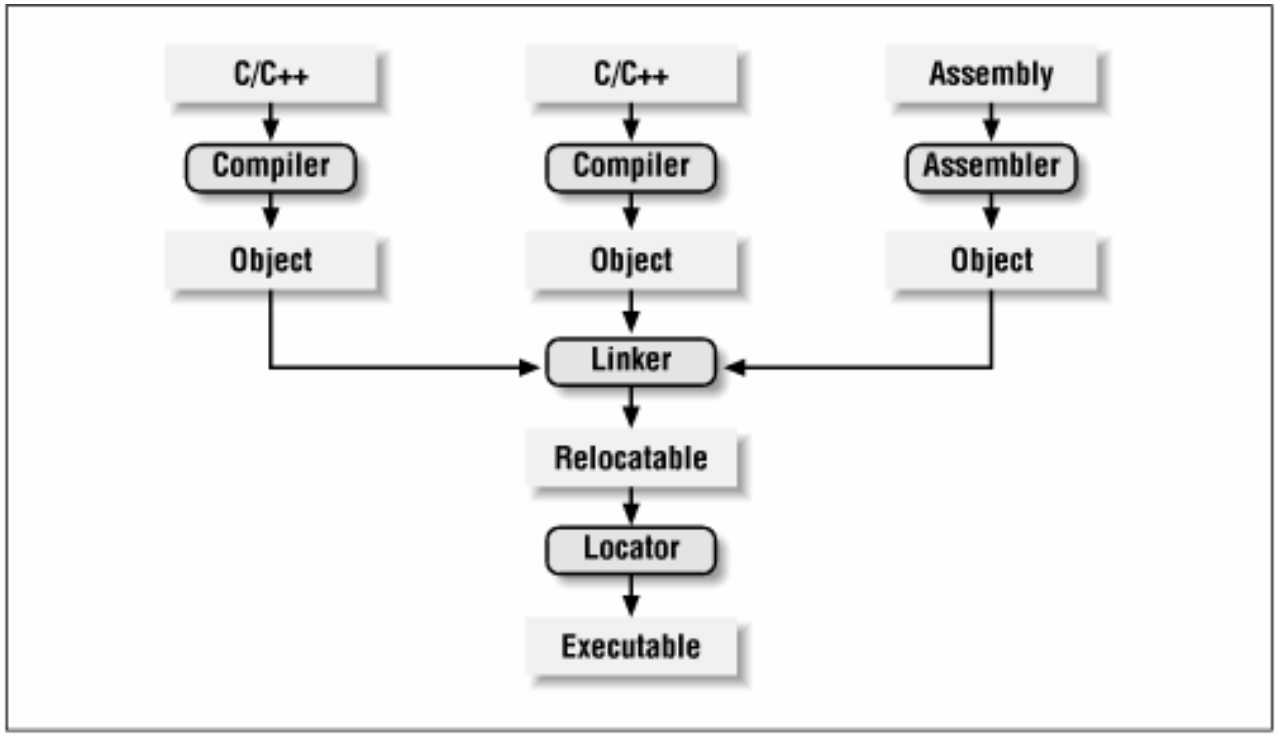
Cada RSI, se ejecuta independientemente de las demás:

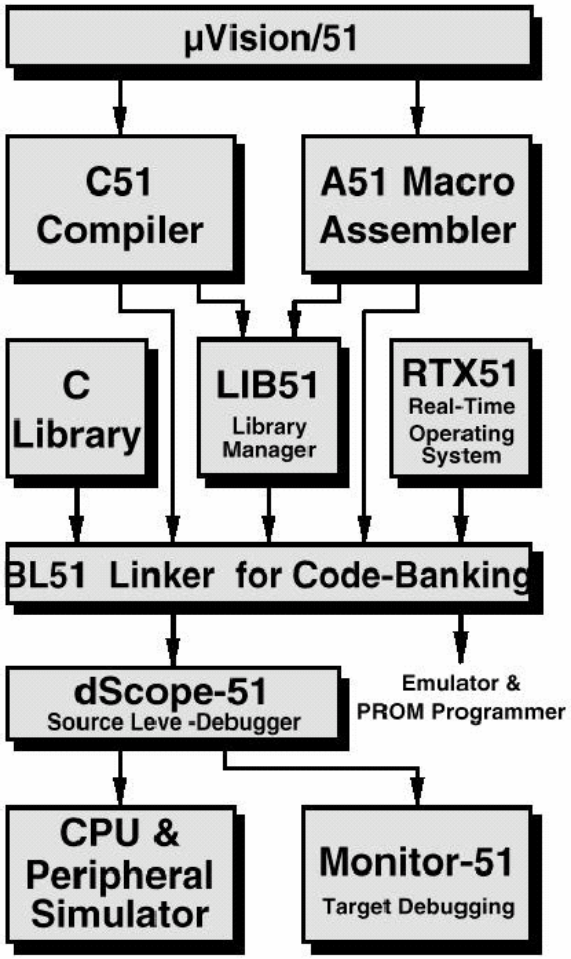
- Realiza la acción de inmediato, la acción debe se rápida para no dejar el sistema bloqueado.
- Si se programa una acción que requiera más tiempo se modifica una variable compartida con el programa principal (bandera).
- Una misma interrupción puede gestionar varios periféricos (PIC).
- La RSI termina reseteando el hardware y de forma especial (RETI).

Programa basado en interrupción

```
#include <reg51.h>
#include <stdio.h>
#define TIEMPO 1000

#define bajo(x)  ((unsigned char)((x)&0x00FF))
#define alto(x) ((unsigned char)((x)>>8))

unsigned char segundos,minutos,horas;
unsigned int msec;

/* Función de atención a la interrupción */
void timer0() interrupt 1 {

     msec++;
     if (msec>=1000) {msec=0;segundos++;}
     if (segundos >= 60) { segundos=0;minutos++; }
     if (minutos >= 60) { minutos=0;horas++; }
     if (horas >= 24) horas=0;
     /* Reprogramar el timer */
     TH0 = alto(65535-TIEMPO);
     TL0 = bajo(65535-TIEMPO);
}

void main(void) {
     /* Inicializa puerto serie */
     SCON = 0x52; /* Configura comunicación serie modo 1 */
     TCON = 0x40;
     TMOD = 0x20; /* Timer 1 en Modo 2, GATE=0 y C/T=0 */
     TH1 = 0x0FD; /* Valor para 9600 baudios */

     /* Inicializa reloj */
     msec=segundos=minutos=horas=0;

     /* Inicializa timer 0 */
     TMOD |= 0x02;
     TH0 = alto(65535-TIEMPO);
     TL0 = bajo(65535-TIEMPO);
     TR0 = 1;  /* Pone en march timer 0 */
     ET0 = 1; /* Habilita int timer 0 */
     EA = 1; /* Habilitacion general de int */

     while (1) {
          printf("%02u:%02u:%02u\r",(unsigned
          int)horas,(unsigned int)minutos,(unsigned
          int)segundos);
     }
}
```


El proceso de Compilar y Linkar



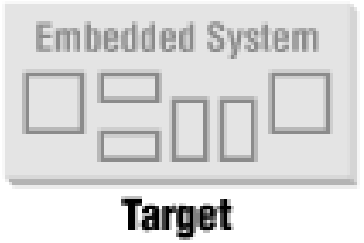


Resultado del proceso de linkado

```
LINK MAP OF MODULE:  pru (GENERADOR)


            TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
            -----------------------------------------------------

            * * * * * * *   D A T A   M E M O R Y   * * * * * * *
            REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
            IDATA   0008H     0001H     UNIT         ?STACK

            * * * * * * *   C O D E   M E M O R Y   * * * * * * *
            CODE    0000H     0003H     ABSOLUTE
            CODE    0003H     000CH     UNIT         ?C_C51STARTUP
            CODE    000FH     0009H     UNIT         ?PR?MAIN?GENERADOR
            CODE    0018H     0007H     UNIT         ?PR?_RETARDO?GENERADOR
```

El sistema de desarrollo

Las herramientas de desarrollo que generan el software empotrado se ejecutan en un ordenador de propósito general

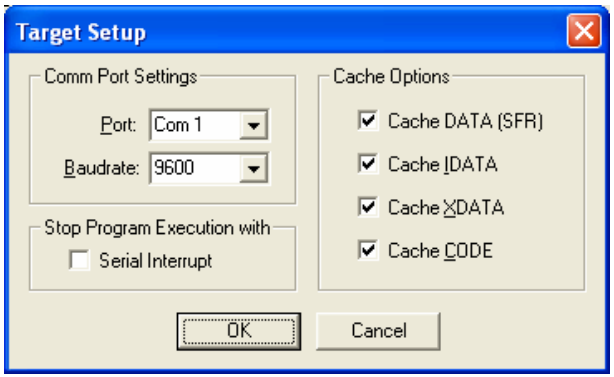


El software empotrado que se genera con estas herramientas se ejecuta en el sistema empotrado

El depurado básico de programas

- Programador de Dispositivos: Willeprom, NEEDHAM.

- Depuradores Remotos (monitores). Ejemplo: Paulmon, Keil.



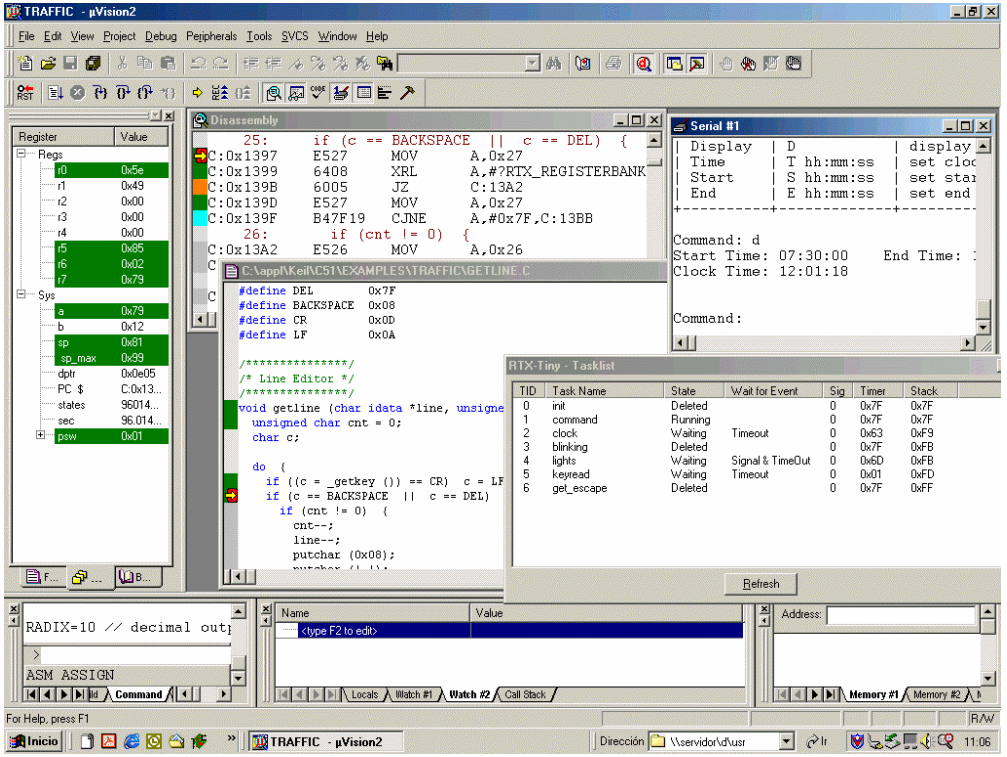
Depurado avanzado de programas

- Emuladores en Circuito: Costosos pero permiten depuración avanzada. Ejemplo: NOHAU.

- Emuladores de ROM: Sustituyen a la ROM de la placa.

Depurado de programas, otras herramientas

- Simuladores.



- Analizadores lógicos.
- Osciloscopios.

Metodologia para conocer el hardware

- Identificar componentes principales.
- Conseguir Hojas de características de los componentes periféricos.

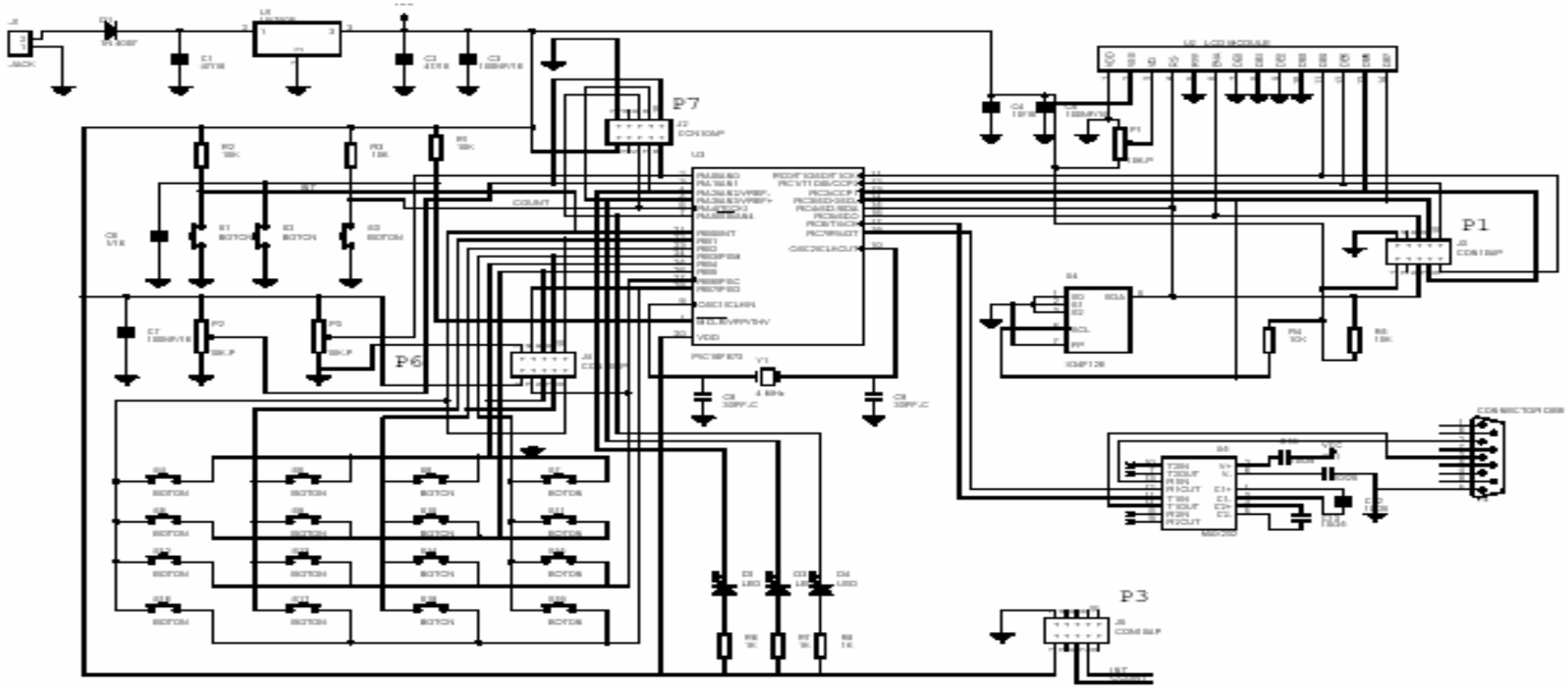
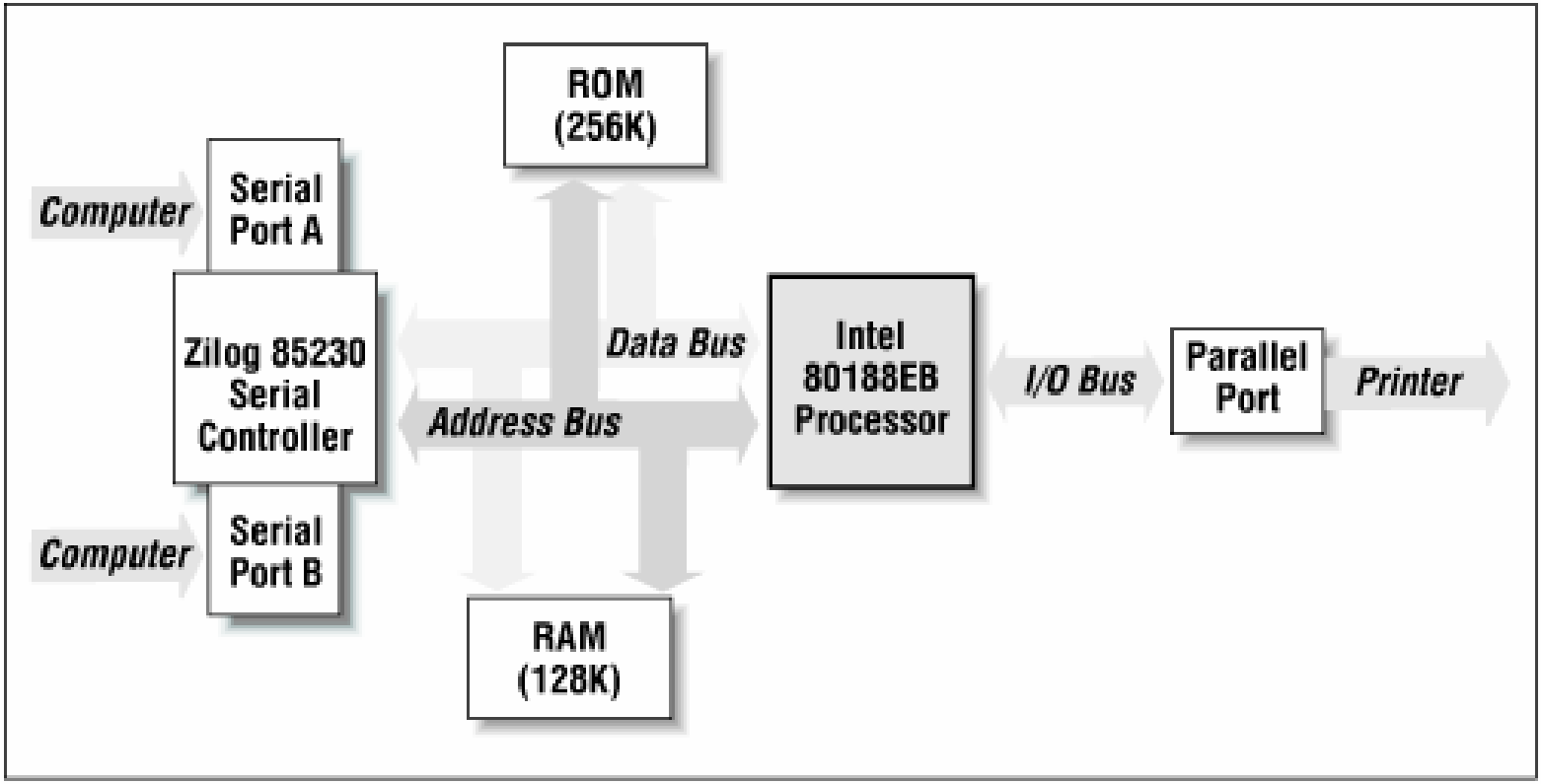


Diagrama de bloques del hardware



¿Cuál es la finalidad del sistema?

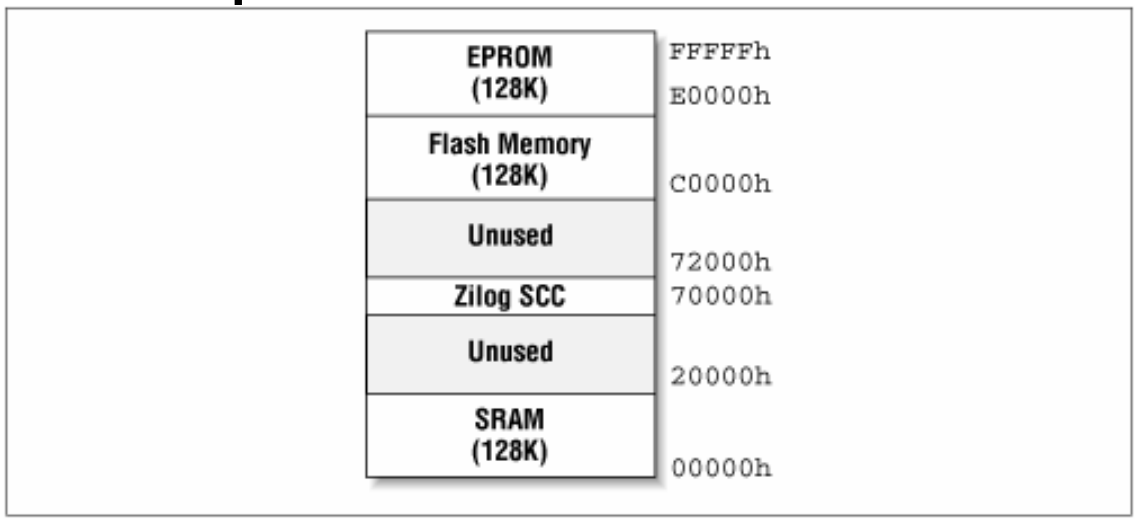
- Lista de funciones que realiza
- Ordenar las funciones
- Realizar diagrama de flujo

¿Cuál es el flujo de datos?

- Lista de datos que procesa
- Lista de variables (arrays)

Mapa de Memoria del Sistema

Distribución de memoria y periféricos en todo el rango direccionable mediante el bus de direcciones interno o externo del microprocesador.



- A partir del esquema eléctrico se deduce mapa de memoria del sistema.
- Es imprescindible para programar el sistema y realizar ampliaciones.
- Algunos sistemas que no disponen de memoria externa tienen mapa predeterminado por fabricante.

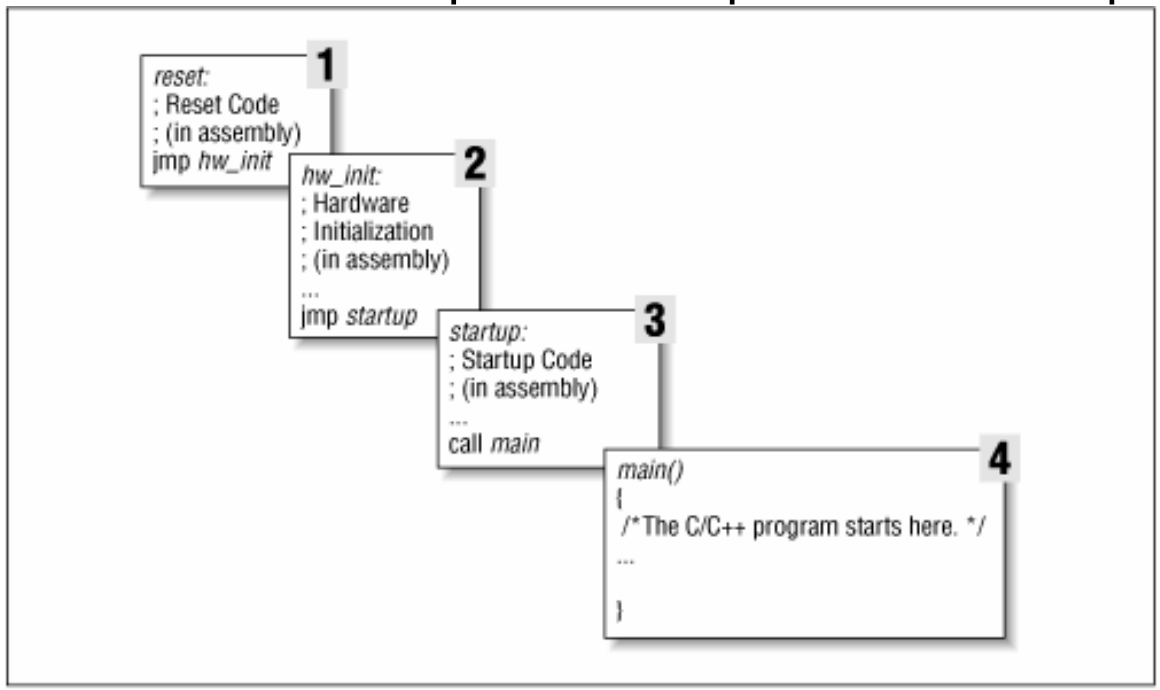
Mapa de Memoria del Sistema

Normalmente la dirección de memoria de los periféricos se coloca en un fichero de cabecera.

```
/*****************************************************************
 *
 *  I/O Map
 *
 *            Base Address    Description
 *            --------------- ---------------------------------------
 *            0000h           Unused
 *            FC00h           SourceVIEW Debugger Port (SVIEW)
 *            FD00h           Parallel I/O Port (PIO)
 *            FE00h           Unused
 *            FF00h           Peripheral Control Block (PCB)
 *
 *****************************************************************/

#define    SVIEW_BASE   0xFC00
#define    PIO_BASE     0xFD00
#define     PCB_BASE     0xFF00
```

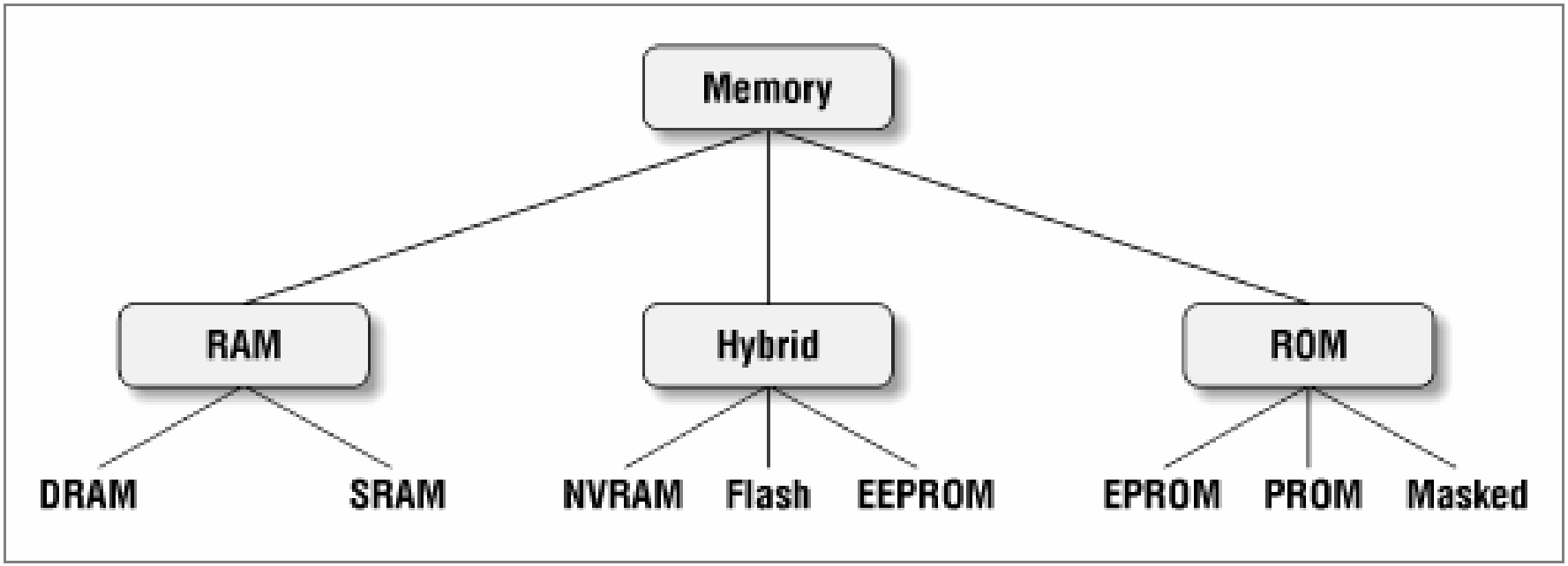
Inicialización de Software y Hardware

- La inicialización Hardware básica tiene lugar antes de ejecutar main().
- Se realiza en ensamblador en el fichero STARTUP.ASM o crt.s.
- A continuación se realiza la inicialización software.
- Luego se llama a main.
- Dentro de main se inicializan los periféricos que usa nuestro programa.



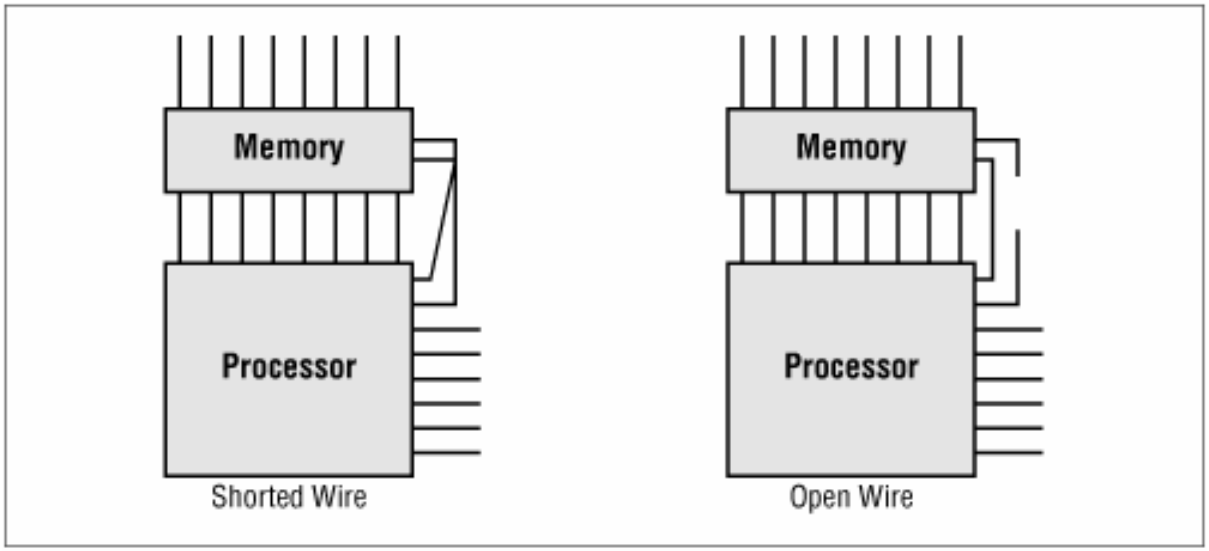
Tipos de memoria del sistema

- RAM: SRAM o DRAM.
- HIBRIDA: EEPROM, FLASH, NVRAM.
- ROM: Máscara, PROM o EPROM.



Técnicas de testeo de memoria

- Problemas con el cableado entre el procesador y la memoria.
- Chips de memoria no colocados
- Chips de memoria mal insertados



- Estrategias de Test:
  - Testeo del bus de datos con un ¨1¨ desplazándose entre los bits.
  - Testeo del bus de direcciones con un “1” desplazándose entre los bits.
  - Testeo de la memoria en sí usando un patrón incremental.

Periféricos

Cada periférico interno o externo se accede a través de uno o varios registros de control y registros de estado.

- **Registro de control**: registro de solo escritura donde se programan las funciones u órdenes en el periférico.
- **Registro de estado**: registro de solo lectura donde se lee el estado del periférico.

No hay forma estándar de acceder en C -> se accede en ensamblador o se accede a través de punteros.

- **Device Driver**: es un módulo que contiene todas las instrucciones en C o en ensamblador necesarias para programar un determinado dispositivo. Tiene la ventaja de que oculta al programador la complejidad de la programación de bajo nivel o básica.

Como se hace un device driver en C

1. Crear una estructura de datos o un puntero que mapee en memoria el conjunto de registros de lectura y escritura del periférico.

```
struct TimerCounter
{
    unsigned short  count;              // Current Count, offset 0x00
    unsigned short  maxCountA;          // Maximum Count, offset 0x02
    unsigned short  _reserved;          // Unused Space,  offset 0x04
    unsigned short  control;            // Control Bits,  offset 0x06
};

#define TIMER_ENABLE    0xC000          // Enable the timer.
#define TIMER_DISABLE   0x4000          // Disable the timer.
#define TIMER_INTERRUPT 0x2000          // Enable timer interrupts.
#define TIMER_MAXCOUNT  0x0020          // Timer complete?
#define TIMER_PERIODIC  0x0001          // Periodic timer?
```

2. Definir un juego de variables para mantener el estado del dispositivo.
3. Crear funciones para inicializar el periférico.
4. Crear funciones que conjuntamente proporcionen una interfaz a los usuarios del device driver.
5. Crear las necesarias RSI (rutinas de servicio de interrupción)