Compilación de programas en lenguaje C para el microcontrolador AVR

1. Comenzando

Las herramientas de desarrollo de aplicaciones provienen de GNU y funcionan en Linux. Para arrancar este sistema operativo en los ordenadores del laboratorio hay que esperar que arranquen de la red (pueden tardar unos 20 seg.) y luego, en el menú que se muestra seleccionar "3) Linux".

Tras el arranque de Linux hay que entrar en la cuenta de prácticas:

Login: **********

Password: *******

Dentro de la cuenta de prácticas cada grupo tendrá su propio subdirectorio.

Este es un listado de las herramientas de desarrollo y otras aplicaciones útiles:

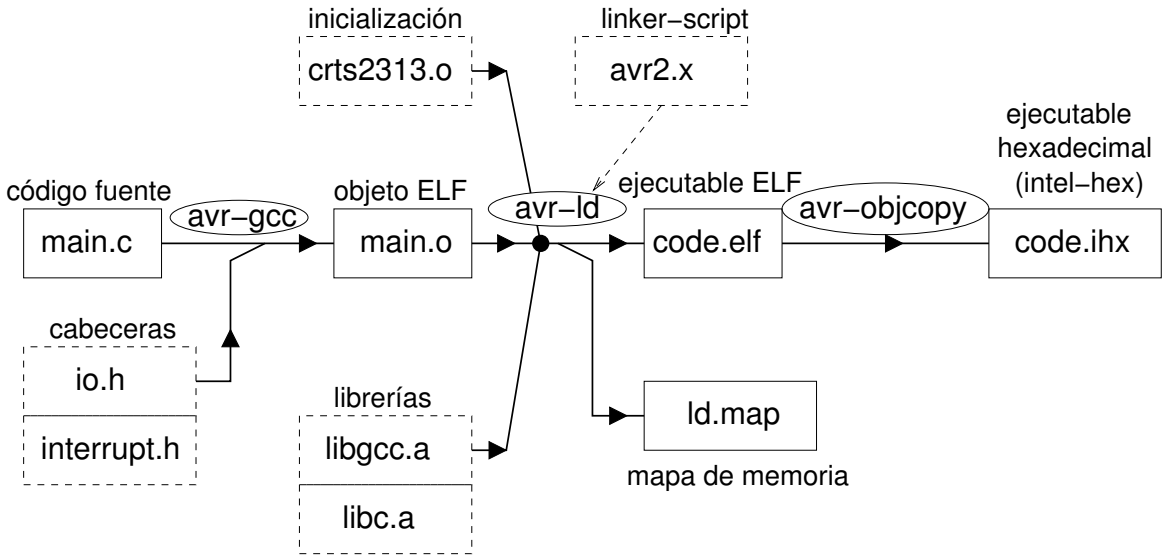
- avr-gcc: compilador de C para AVR.
- avr-as: ensamblador para AVR.
- avr-ld: linker.
- avr-objcopy: conversor de formatos de ficheros ejecutables.
- avr-objdump: desensamblador para AVR.
- make: gestor de proyectos.
- avrprg: programador para el prototipo del microcontrolador (defecto: COM1)
- terminal: terminal para el puerto serie (defecto: COM2, 9600 baudios)
- nedit: editor de texto para programas en C.

2. Programa de ejemplo

En el directorio “ejemplo” hay un conjunto de ficheros que mustran de forma básica como generar imágenes de memoria para el microcontrolador AVR a partir de código fuente en C y en ensamblador. Para compilar el programa de ejemplo en primer lugar copiaremos los ficheros a otro directorio y luego siemplemente ejecutaremos el comando “make”. Los ficheros del ejemplo son:

- main.c: programa en C que incluye la función “main”.
- Makefile: dependencias y control de la compilación.

El proceso de la compilación queda representado de forma gráfica en la siguiente figura:



En la compilación intervienen tanto nuestro código fuente como otros ficheros que se encuentran convenientemente instalados en el servidor. Estos últimos son los ficheros de cabecera, las librerías del gcc (libgcc.a) y estándar (libc.a), el código de inicialización del programa (crts2313.o) y el “linker-script” (avr2.x) que controla el proceso de enlazado. Las dependencias entre ficheros y los comandos que se han de ejecutar para generarlos están detallados en el fichero “Makefile”, que contiene líneas como las siguientes:

```
ACFLAGS = -Os -mmcu=at90s2313

OBJS = main.o
```

En estas líneas se definen algunas variables que se utilizarán a continuación. En particular la variable ACFLAGS contiene las opciones que se pasarán al compilador “avr-gcc”, tales como el nivel de optimización (-Os) y el modelo del microcontrolador para el que se va a generar el código (-mmcu=at90s2313). La variable OBJS contiene una lista de los ficheros objeto que intervienen en la compilación. En este caso tan sólo hay un fichero objeto (main.o).

```
code.ihx:(tab) code.elf
(tab)       avr-objcopy -O ihex $< $@
```

En estas líneas se indica que “code.ihx” se obtiene a partir de “code.elf” y que el comando que hay que ejecutar para generar “code.ihx” es “avr-objcopy ...”. Las variables “$<” y “$@” son comodines que hacen referencia al fichero de entrada “$<” y al de salida “$@” respectivamente.

```
code.elf: (tab) $(OBJS)
(tab)     avr-gcc $(ACFLAGS) -Wl,-Map,ld.map -o $@ $(OBJS)
```

Aquí indicamos que “code.elf”, el ejecutable ELF, depende de “main.o” a través de la variable “$(OBJS)” e indicamos el comando para generar el ejecutable en formato elf. El programa avr-gcc llamará al linker (avr-ld), pasándole entre los parámetros la expresión “-Map ld.map”, lo que dará lugar a que el linker genere un listado del mapa de memoria del ejecutable.

```
%.o :  (tab) %.c Makefile
(tab)         avr-gcc $(ACFLAGS) -c $<
```

Estas líneas definen una regla genérica para la compilación de los programas en C: Todo fichero objeto “ %.o” que dependa de un fichero C “ %.c” se compila con “avr-gcc $(ACFLAGS) -c $<”. La dependencia con el fichero Makefile fuerza por lo tanto la compilación de todos los fuentes en C cada vez que se cambia el propio fichero Makefile.

3. Opciones de los comandos

Como hemos visto en el ejemplo cada comando (ensamblador, compilador, linker,...) tienen un gran número de opciones que se pueden seleccionar en la línea de comandos. Destacaremos a continuación las más relevantes:

avr-as

- -o <fichero de salida>: Indica cual va a ser el fichero de salida

avr-gcc

- -S : genera un listado en ensamblador en lugar del fichero objeto.
- -c : genera sólo el fichero objeto. (No se intenta generar un ejecutable)
- -O<n>: nivel de optimización:

- -O0 : sin optimización.
- -O1 : optimización de transferencias entre registros.
- -O2 : más optimizaciones.
- -O3 : además se incluyen funciones como “inline”
- -Os : optimización para reducir el espacio de memoria ocupado por el código.

- -mmcu=<microcontrolador>: Selecciona el conjunto de instrucciones adecuado para el tipo de microcontrolador especificado.
- -Wl,<opción>,<opción>,... : Permite pasar opciones al linker.

avr-ld

- -T <fichero linker-script>: Indica el fichero que se usará para controlar el proceso de enlazado. En este fichero se detalla la organización de la memoria.
- -Map <fichero listado>: genera un listado del mapa de memoria en el fichero indicado.

avr-objcopy

- -O <formato>: indica el formato del fichero de salida (el fichero de entrada está en formato ELF). Los formatos que nos pueden interesar son:
  - srec : Formato hexadecimal de Motorola.
  - ihex : Formato hexadecimal de Intel.
  - binary : Imagen de la memoria en binario.

avr-objdump

- -d : desensamblar secciones de código del fichero objeto o ejecutable ELF que se indique.
- -D: desensamblar todas las secciones del fichero.

4. Particularidades del compilador GCC para AVR

Tipos de datos:

Las longitudes de los tipos de datos para avr-gcc son:

Tipo de dato	Longitud
char	8 bits
short	16 bits
int	16 bits
long	32 bits
float	32 bits
double	32 bits

Las variables de coma flotante aunque estén soportadas por el compilador no son nada recomendables, ya que su uso requiere incluir en el programa las rutinas de emulación de coma flotante de “libgcc.a”, que ocupan mucha memoria.

Modificadores

volatile

Este modificador indica que la variable a la que hace referencia puede modificarse por causas ajenas al programa, y por lo tanto el compilador no debe optimizar sus accesos a dichas variables. Este modificador debe usarse, por ejemplo, cuando se accede a los registros de los periféricos, o cuando una variable puede modificarse desde una rutina de interrupción.

Ejemplo:

```
dato=*(volatile unsigned char *)0x30; // lectura de PIND
```

En la dirección 0x30 está el puerto de E/S PIND y su valor depende de las tensiones aplicadas a los pines del microcontrolador. Por lo tanto el puntero a dicha dirección de memoria se debe declarar como “volatile”.

inline, extern inline

Si una función se declara como “inline” el compilador no genera llamadas a la subrutina de dicha función. En su lugar el código de la función se inserta en cada ocasión en la que se llama a la función en el programa. Esto puede hacer los programas más largos, pero más rápidos al evitar los tiempos de llamada y retorno de subrutinas. Este modificador es particularmente útil con funciones muy cortas. Cuando la función en cuestión se declara como “inline” todavía se genera una subrutina por si acaso es llamada desde otros ficheros objeto que se puedan enlazar con el programa. Esto se evita definiendo la función como “extern inline”. En este caso no se genera dicha subrutina.

Código en ensamblador

En los programas en C se puede insertar directamente código en ensamblador mediante la palabra clave "asm" o "asm volatile". Ejemplo:

```
asm volatile ("sei");  // Habilitamos interrupciones
```

El modificador "volatile" fuerza al compilador a insertar nuestro código exactamente en la parte del programa en la que se encuentra. El comando "asm" es mucho más sofisticado y permite el paso de parámetros entre el código en C y en ensamblador, como ocurre en el ejemplo siguiente:

```
a=100; b=20;
asm volatile ("swap %0\n\tadd %0, %1":"+r"(b):"r"(a));
```

que el compilador convierte en:

```
ldi r24,lo8(100)
ldi r25,lo8(20)
swap r25
add r25,r24
```

donde vemos que las variables "a" y "b" han sido almacenadas en los registros "r24" y "r25". La sintaxis de la línea "asm" es como sigue:

asm volatile ("*código ensamblador*" : *parámetros de salida* : *parámetros de entrada*);

Donde, en el código en ensamblador hacemos referencia a los parámetros como "%0" para el primero de las listas, "%1" para el siguiente y así sucesivamente. Las listas de parámetros de entrada y/o salida se definen como:

"modificadores" (variable), "modificadores" (variable)....

Los modificadores indican:

- "=" : se sobreescribe la variable.
- "+" : se lee y modifica.
- "r" : la variable reside en un registro.
- "m" : la variable es una posición de memoria.
- "i" : la variable es un operando inmediato (constante).

Funciones de interrupción

Las funciones de interrupción son especiales ya que están concebidas como manejadores de interrupciones. Esto significa que difieren de las funciones normales en los siguientes aspectos:

- Guardan todos los registros que modifican en la pila, incluyendo SREG, al comienzo de la función.
- Recuperan los valores originales de los registros de la pila al finalizar la función.
- Retornan con la instrucción "RETI".

Para que una función actúe como rutina de atención a una interrupción se debe declarar mediante la macro ISR(<vector>) tal y como se muestra en el ejemplo:

```
ISR(TIMER0_OVF0_vect)
{
    nirqs++;
}
```

La macro ISR del ejemplo proporciona un manejador de interrupciones para el desbordamiento del temporizador "TIMER0". Este manejador lo único que hace es incrementar una variable global del programa. Las funciones de interrupción no tienen parámetros ni devuelven resultados. Desde estas funciones no se debe llamar a ninguna otra función del programa pues se podría alterar el valor de algún registro que no ha sido previamente guardado en la pila.

Los nombres de los manejadores de interrupción dependen del microcontrolador. Para el at90s2313 son los siguientes:

Nombre del Manejador	Nº del vector
INT0_vect	1
INT1_vect	2
TIMER1_CAPT1_vect	3
TIMER1_COMP1_vect	4
TIMER1_OVF1_vect	5
TIMER0_OVF0_vect	6
UART_RX_vect	7
UART_UDRE_vect	8
UART_TX_vect	9
ANA_COMP_vect	10

Como recomendaciones generales para la escritura del código de rutinas de interrupción podemos destacar:

- No llamar a otras funciones desde una rutina de interrupción.
- Recordar declarar como "volatile" todas las variables globales que modifique la rutina de interrupción.
- No hacer retardos dentro de una rutina de interrupción. Evitar los bucles.
- Cuando una rutina de interrupción modifique una variable de 16 o 32 bits hay que garantizar un acceso atómico a dicha variable en el programa principal, lo que implica:
  - Apagar las interrupciones: cli();
  - Leer la variable: valor=var_global;
  - Activar las interrupciones: sei();
- Si la rutina de interrupción accede a algún registro de E/S de 16 bits del temporizador también hay que garantizar un acceso atómico desde el programa principal a TODOS los registros de 16 bits de temporizador. Ello se debe a que la interrupción puede alterar el valor de un registro temporal de 8 bits que se usa para sincronizar el acceso a la parte alta del dato de 16 bits.

Acceso a registros de E/S

Los registros de entrada y salida del AVR 90S2313 están definidos en el fichero de cabecera "<avr/io.h>". El contenido de este fichero varía dependiendo del modelo de microcontrolador que se especifice en la compilación (avr-gcc -mmcu=....). Los registros están defindos como posiciones fijas de la memoria de datos, tal y como se muestra en el ejemplo siguiente:

```
#define UBRR (*(volatile unsigned char *)0x29)
```

Por lo tanto, en nuestros programas, los nombres de los registros de E/S de microcontrolador representan a variables a las que se pueden asignar valores y que pueden intervenir en expresiones aritméticas y logicas. El compilador es suficientemente inteligente para convertir las referencias a dichas variables en instrucciones "IN" y "OUT", como en el siguiente ejemplo:

```
UBRR=64;
```

que se compila como:

```
ldi r24,0x40
out 0x09,r24
```

También el compilador puede detectar un cambio de un único bit en estas variables y utilizar las instrucciones de manejo de bit SBI/CBI, como en este ejemplo:

```
PORTD|=(1<<3);
```

que se compila como:

```
sbi 0x12, 3
```

Acceso a la memoria de programa

Desgraciadamente no es posible acceder directamente a la memoria de programa del AVR en los programas compilados con gcc, pero podemos usar soluciones alternativas como la del ejemplo:

```
#include <avr/pgmspace.h>
prog_uchar tabla[10]={1,2,3,4,5,6,7,8,9,10};
prog_uchar texto[]="En un lugar de la Mancha";
...
data=__LPM(&tabla[8]);
data=pgm_read_byte(&tabla[8]);
```

__LPM o su sinónimo "pgm_read_byte" es una macro que devuelve el valor de la posición de la memoria de programa cuya dirección se pasa como argumento. En el fichero de cabecera <avr/pgmspace.h>se definen varios tipos de datos que van a ser almacenados en la memoria de programa. Esto tipos son:

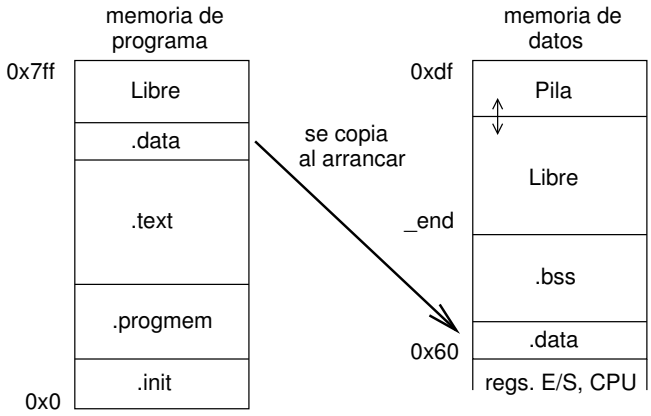
- prog_void * : para punteros constantes de 16 bits
- prog_char / prog_int8_t : constantes de 8 bits con signo
- prog_uchar / prog_uint8_t : constantes de 8 bits sin signo
- prog_int16_t : constantes de 16 bits con signo
- prog_uint16_t: constantes de 16 sin signo
- prog_int32_t: constantes de 32 bits con signo
- prog_uint32_t: constantes de 32 sin signo

Las variables de 16 y 32 bits se deben leer mediante variantes de 2 y 4 bytes de la macro __LPM. Ejemplo:

```
prog_int16_t def_val[3]={1234,4237,25315};
prog_int32_t def_val32[4]={1234567,10000000,99999999,87654321};
...
int val;
long v32;
val=__LPM_word(&def_val[1]);    // val=4237
val=pgm_read_word(&def_al[1]);  // sinónimo
v32=__LPM_dword(&def_val32[2]); // v32=99999999
v32=pgm_read_dword(&def_val32[2]);  //sinónimo
```

Organización de la memoria

La memoria en los programas compilados con gcc queda organizada en secciones, como se muestra en la siguiente figura:



Cada sección contiene un tipo distinto de datos. Estos son:

- .init : tabla de saltos de interrupciones y código de inicialización.
- .progmem : Constantes en la memoria de programa.
- .text : Código del programa.
- .data : Valores iniciales de variables estáticas inicializadas. Estos valores son copiados a la RAM antes de llamar a la función “main”.
- .bss : variables estáticas sin inicializar. Su valor inicial es 0.

El símbolo “_end” nos indica la primera posición libre de la memoria RAM.

Uso de los registros

Paso de parámetros en funciones del lenguaje C.

```
f() : función que se llama desde el programa
p8  : parámetro de 8 bits (char), (short)
p16 : parámetro de 16 bits (int), punteros
p32 : parámetro de 32 bits (long)
```

Ejemplos de asignación de registros a parámetros (orden de los parámetros: f(pr1,pr2,pr3,...);):

	Param. 1	Param. 2	Param. 3	Param. 4
f(p8)	R24			
f(p8,p8,p8,p8)	R24	R22	R20	R18
f(p16,p16)	R25:R24°	R23:R22		
f(p16,p16,p16)	R25:R24	R23:R22	R21:R20	
f(p32,p32)	R25:R24:R23:R22°	R21:R20:R19:R18		
f(p32,p32,p32)	R25:R24:R23:R22	R21:R20:R19:R18	R17:R16:R15:R14	
f(p8,p16,p32)	R24	R23:R22	R21:R20:R19:R18	
f(p32,p16,p8)	R25:R24:R23:R22	R21:R20	R18	

```
° MSB izq, LSB der.
```

Como podemos ver los parámetros se asocian a registros de modo que su byte menos significativo queda siempre en un registro par.

Valor de retorno (si lo hubiese):

```
p8  <-  R25:R24°°
p16 <-  R25:R24
p32 <-  R25:R24:R23:R22
°° R25 se pone en 0
```

Uso de los registros en funciones escritas en ensamblador que se llaman desde código en C:

Registro	Uso (código C)	Acción
R0	temporal	Se puede cambiar
R1	valor 0	Debe valer 0 al retornar
R2-R17	variables	Se deben PRESERVAR
R18-R25	variables/parámetros	Se pueden cambiar
R26:R27	Puntero X	Se pueden cambiar
R28:R29	Puntero Y	Se deben PRESERVAR
R30:R31	Puntero Z	Se pueden cambiar

Los registros marcados como "se puede cambiar" pueden usarse en subrutinas escritas en lenguaje ensamblador como varables temporales y su valor inicial no necesita ser recuperado antes de retornar.

Al compilador avr-gcc se le puede instruir para que no utilice determinados registros a la hora de generar el código. Esto se hace mediante opciones del tipo “gcc -avr -mmcu=at90s2313 -ffixed-r3 -ffixed-r4 ...”. En este ejemplo el compilador nos garantiza que no utilizará los registros R3 y R4. Eso no significa que no se utilicen en algunas funciones de librería, así que esta opción debe usarse con cuidado.