Práctica 1

Ejercicios de Repaso

e) Explicación de las siguientes sentencias en C:

- DDRB = 0x0F: establece el comportamiento de entrada/salida de cada uno de los bits del Puerto B. En este caso, 0x0F = 0000 1111, es decir:

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
IN	IN	IN	IN	OUT	OUT	OUT	OUT

- PORTB = 0x0E: indica los datos a establecer en los bits del Puerto B.

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	1	1	1	0

f) Explicación de **PORTC |= (1<<PC0) | (1<<PC2) | (1<<PC3)**

La sentencia anterior realiza una operación OR entre el valor almacenado como datos del Puerto C y el número binario 0b00001101. El resultado es asignado a la variable PORTC. Básicamente es una máscara, que establece en “1” (alto) los bits de los puertos PC0, PC2 y PC3, en otras palabras, los bits 0, 2 y 3 de PORTC.

La diferencia con el caso anterior es que, al tratarse de una máscara, **no se sobrescriben** los bits que se marcaron como 0 (en OR) o como 1 (en AND).

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
x	x	x	x	x	x	x	x
0	0	0	0	1	1	0	1
x	x	x	x	1	1	x	1

PC2 representa el número “2”, y la operación “1<<PC2” devuelve 0b00000100, es decir, un número binario con el bit 2 en alto y el resto en bajo.[1]

Cito textual de la Clase 4 (pág. 11): “La asociación entre las variables y los registros se realiza en el archivo de cabecera **io.h”**

[1] En estos casos, no parece gran cosa, pero es de gran utilidad cuando los nombres de los registros no siguen un orden particular, como puede ser el registro de estado de la CPU

Ejercicio de Simulación

```
/* Inclusión de cabeceras de bibliotecas de código */
#include <avr/io.h>                 // Definición de Registros del microcontrolador
#define F_CPU 8000000UL             // Especifico la frecuencia de reloj del MCU en 8MHz
#include <util/delay.h>             // Retardos por software – Macros: depende de F_CPU

/* Función main */
int main (void)
{
        /* Setup */
        DDRC &=~(1<<PC0);   // Comentarios a completar…
        PORTC |= (1<<PC0);  // …
        DDRB = 0xFF;         // …

        /* Loop */
        while(1)
        {
                if (PINC & (1<<PINC0))
                {
                        PORTB = 0b10101010; //
                        _delay_ms(100);     //
                        PORTB = 0x00;       //
                        _delay_ms(100);     //
                }
                else
                {
                        PORTB = 0b01010101; //
                        _delay_ms(100);     //
                        PORTB = 0x00;       //
                        _delay_ms(100);     //              }
        }
        /* Punto de finalización del programa (NO se debe llegar a este lugar) */
        return 0;
}
```

- DDRC &= ~(1<<PC0): es lo mismo que DDRC &= 0xFE, es decir, establece el puerto PORTC0 como entrada (0) y deja los 7 restantes sin cambios.
- PORTC |= (1<<PC0): es lo mismo que PORTC |= 0x01, es decir, manda un bit "1" como salida al PORTC0, y deja los 7 restantes sin cambios. Sin embargo, PC0 es entrada, así que no se modifica.
- DDRB = 0xFF: establece todo los bits del Puerto B como salida.
- PINC & (1<<PINC0): vale verdadero si el bit leído de PC0 es "1", sino es falso.
- PORTB = 0b10101010: escribe un "0" en los bits pares y "1" en los impares del Puerto B.
- PORTB = 0b01010101: escribe un "1" en los bits pares y "0" en los impares del Puerto B.
- PORTB = 0x00: escribe un "0" en todos los bits del Puerto B.
- _delay_ms(100): el programa se demora unos 100 ms antes de pasar a la siguiente instrucción.

Básicamente, este programa realiza un "intermitente" de los bits impares del PORTB si PC0 es "1", caso contrario lo hace con los bits pares. Cada "estado" dura 100 ms, chequeando la condición cada 200 ms.

Repaso del Lenguaje C — Ej. N°1

a1) Tipos de variables (datos):

Tipo		#bits	Rango de representación	bytes
signed char		8bits	-128 a+127	1
unsigned char		8bits	0 a 255	1
short int		16bits	-32768 a 32767	2
unsigned short		16bits	0 a 65535	2
int		???	(depende del compilador)	
long		32bits	- 2147483648 a 2147483647	4
unsigned long		32bits	0 a 4294967295	4
float	(IEEE32)	32bits	1.17549x10-38 a 3.40282x10+38	4
double	(IEEE64)	64bits	2.22507x10-308 a 1.79769x10+308	8

a2) Modificadores:

- **Static**: tenemos 3 casos aplicables
  - Variable local: tiene dirección fija en memoria, se conserva durante toda la ejecución del programa, para toda invocación de la función.
  - Variable global: su alcance está limitado al ".c" donde está definida.
  - Función: ídem anterior.
- **Volatile:** advierte al compilador que la variable puede cambiar por alguien externo, y no debe aplicar optimizaciones sobre la misma. Ej: registros MCU.
- **Register:** sugiere al compilador que coloque la variable en un registro CPU.
- **Const:** indica al compilador que la variable es sólo lectura, puede ir en ROM.

a3) Tipos de datos que utilizaremos (definidas en **stdint.h**):

- typedef signed char int8_t;
- typedef unsigned char uint8_t;
- typedef short int int16_t;
- typedef unsigned short int uint16_t;
- typedef long int int32_t;
- typedef unsigned long int uint32_t;
- typedef long long int int64_t;
- typedef unsigned long long int uint64_t;

b) Sentencias (directivas) del preprocesador:

- **#include:** reemplaza la sentencia por el contenido del archivo citado.
- **#define NAME (expresión):** define una constante a reemplazar en código.
- **#ifdef:** permite realizar compilación condicional según constantes definidas.
- **typedef:** permite definir tipos de variable a partir de otras ya existentes.

c) Una constante de tipo carácter es aquella que representa a un valor (dato) perteneciente al conjunto de caracteres que puede representar el ordenador. Las siguientes constantes de tipo carácter están expresadas por su valor: 'a', 'T', '5', '+'.

d) Diferencia entre variable local y global:

- **Local:** vive dentro del ámbito de una función mientras se esté ejecutando. Se alojan en los registros CPU o pila. Son descartadas al terminar la función.
- **Global:** accesible desde cualquier parte del programa, declaradas fuera de toda función y alojadas en la RAM. Modificables por cualquier función.

e) Operadores lógicos:

- A && B: devuelve verdadero si ambas condiciones A y B son verdaderas.
- A || B: devuelve verdadero si alguna o ambas condiciones son verdaderas.
- ! A: devuelve verdadero si la condición A es falsa, y viceversa.
- A < B: devuelve verdadero si el número A es menor estricto que B.
- A <= B: devuelve verdadero si el número A es menor o igual que B
- A > B: devuelve verdadero si el número A es mayor estricto que B
- A >= B: devuelve verdadero si el número A es mayor o igual que B
- A != B: devuelve verdadero si el valor A es distinto que B
- A == B: devuelve verdadero si el valor A es el mismo que B
- A & B: devuelve el resultado binario de aplicar AND entre A y B
- A | B: devuelve el resultado binario de aplicar OR entre A y B.
- A ^ B: devuelve el resultado binario de aplicar XOR entre A y B
- ~ A: devuelve el complemento a uno del número A (inversión de bits).
- A << N: devuelve el binario A desplazado N lugares a la izquierda.
- A >> N: devuelve el binario A desplazado N lugares a la derecha.
- A &= B, A |= B, A ^= B, A <<= B, A >>= B guardan el resultado en A.

La diferencia entre “&&” y “&”, es que el primero sólo devuelve 0 o 1 (booleano), mientras que el segundo devuelve un número binario de N bits (según operandos).

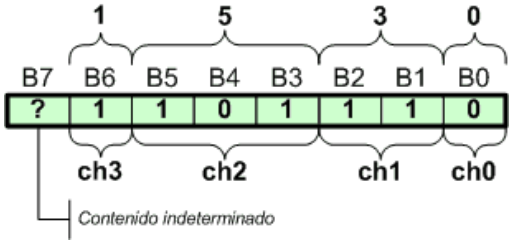
f) El prototipo de una función es la declaración de su tipo de retorno, nombre e información de los parámetros que recibe (tipos, número y orden). Se escribe antes de sus invocaciones, generalmente antes del main(). Los parámetros se pueden pasar por valor (copia) o por referencia (dirección). Se retorna con **return.**

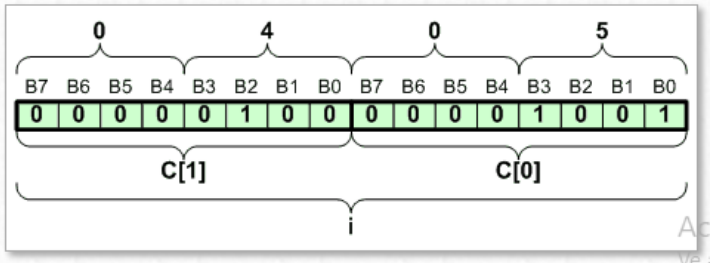
g) Los punteros son variables que almacenan la dirección de memoria de otra variable. Permiten pasar parámetros a una función por referencia. Al incrementar o decrementar su valor permite desplazarse N bytes en memoria, donde N es el tamaño que ocupa el tipo de variable a la que hacen referencia (ej: 2 bytes si es short, o 4 en el caso de Long). De esta manera, en un arreglo, para hacer referencia a una posición específica existen dos formas: **arreglo[i]**, y también ***(arreglo + i).**

h) Las estructuras son variables de tamaño fijo en memoria, que están compuestas por campos de uno o más tipos de datos. Estos mismos son accesibles mediante el operador punto “.” sucedido del nombre del campo. Para declarar campos de bits se especifica el número de bits que ocupará en memoria dicho campo:

Declaración del tipo de dato estructura de bits

```
// Declaración de una estructura de bits
struct Puertos{
        unsigned Ch0: 1;        // Rango 0,1
        unsigned Ch1: 2;        // Rango 0 a 3
        unsigned Ch2: 3;        // Rango 0 a 7
        unsigned Ch3: 1;        // Rango 0,1
};
```





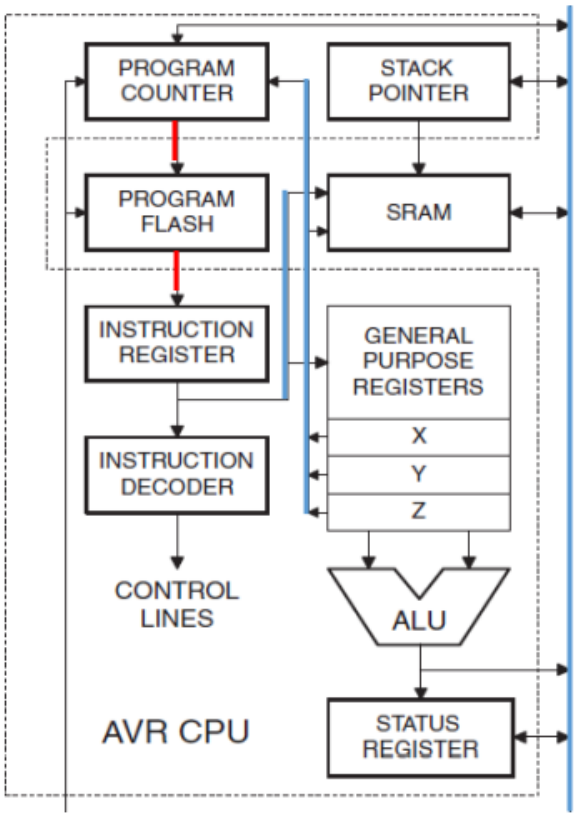
Contenido en memoria de la unión

Familia de Controladores AVR Ej. N°2

Modelo	Puertos	RAM	FLASH
ATmega 328P	3x (B, C, D)	2 KB	32 KB
ATmega 2560	11x (A-H, J-L)	8 KB	256 KB

AVR- CPU

- RISC: 131 instrucciones ejecutadas en su mayoría en 1 ciclo de clock
- Hardvard: Memoria de programa y memoria de datos con buses independientes
- Reloj hasta 20MHz (20 MIPS ideal)
- Arquitecturas basadas en registros:
  - 32 registros CPU de 8 bits
  - Operaciones sobre registros CPU minimizando el acceso a memoria
- En "Instruction Register" permite seleccionar los operandos a usar en la ejecución.
- A su vez el OPcode es decodificado en el "decoder" para generar las señales de control para ejecutar la instrucción.



Modos de direccionamiento:

- Inmediato: dato en la instrucción para asignarlo en el registro de una.
- Directo a registros CPU: se indica el registro que contiene el dato.
- Directo a registros I/O: la dirección del operando es a memoria mapeada.
- Directo a memoria: la instrucción tiene la dirección donde está el dato.
- Indirecto a memoria: se indica el registro que tiene la dirección del dato.
- Indirecto con desplazamiento: ídem pero además se suma algo al puntero.
- Indirecto con pre-decremento o pos-incremento.