Titulación: GRADO DE INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Trabajo de fin de grado

CONTROL DE PERIFERICOS A TRAVÉS DE MICROCONTROLADOR ATMEL

Autor: Dorian Sebastien Bourguignon

Tutor: Evelio J. González González

Septiembre 2017

INDICE

CAPITULO 1: INTRODUCCIÓN

1.1 ABSTRACT

En este proyecto se explicará los pasos que se han dado para conseguir funcionar distintos periféricos de la placa EASYV7, así como los fallos cometidos por el camino y las soluciones para los problemas encontrados.

El procedimiento para conseguir el resultado ha sido simplificar los objetivos al principio, una vez se vaya consiguiendo resultados se han ido complicando los códigos.

Por orden tanto de dificultad como de interactuación, los elementos de la placa programados han sido los Leds, los displays de 7 segmentos, pantalla LCD y pantalla GLCD. Cada uno de ellos habrá tenido sus propias complicaciones y se han tenido que solucionarlos de maneras distintas.

El lenguaje de programación usado será el de ensamblador y se usará el microcontrolador ATmega32 de la familia AVR, fabricante estadounidense Atmel. La manera escribir los códigos será a partir del programa AVR STUDIO 4. Esta aplicación nos permitirá compilar los códigos y obtener los ficheros adecuados para programar el microcontrolador.

Los conocimientos adquiridos serán a partir de información encontrada en internet, datasheets, apuntes de clase y a base de experimentar.

In this Project, the steps taken to achieve the different peripherals of the EASYAVRV7 board will be explained, taking into consideration mistakes made and solutions found for problems occurring during the process.

The procedure for achieving the result was to simplify the initial goals and as results were attained, to complicate the codes.

Experimentation proceeded in ascending order of difficulty; the programmed elements of the board have been the leds, the 7 segments displays, and the LCD and GLCD screens. Each had its own individual complications for which different solutions had to be found.

The microcontroller ATmega32 of the AVR family, American manufacturer Atmel, is programmed using assembling language. The method for writing codes uses the AVR STUDIO4 program. This application allows us to compile codes and obtain the appropriate files for programming the microcontroller.

The knowledge acquired is based on information found on the internet, data sheets, class notes and experiment.

1.2 Objetivos

El principal objetivo será conseguir dos pequeños juegos en la pantalla GLCD, consiguiendo por el camino habilidades en el entorno del lenguaje del ensamblador.

Para conseguir este objetivo final se propuso antes otros objetivos intermedios, el primero de ellos será familiarizarse con la placa EASYV7. Esta placa contiene numerosos elementos como switchs o jumpers, donde será necesario saber configurarlos para poder interactuar de manera correcta con los periféricos de la placa.

Otro objetivo intermedio será aprender a usar las distintas herramientas de software requeridos para la escritura del lenguaje de ensamblador. La principal y más compleja aplicación será el AVR STUDIO 4, será donde se escriban los códigos en ensamblador. Existirán otras aplicaciones, más sencillas, que serán necesarias para el presente proyecto, como el AVRFLASH o el LCDAssistant.

A partir de aquí todos los recursos fueron a mejorar en las habilidades del lenguaje de ensamblador, a medida que vamos programando distintas partes de la placa. Se empezó por los elementos más fáciles, como los leds o los displays 7 segmentos hasta llegar a las pantallas LCD y GLCD.

CAPITULO 2: DESARROLLO DEL SOFTWARE

En este capítulo se enumerará los programas utilizados y su funcionamiento.

2.1 LCDAssistant

Programa gratuito conseguido en internet para transformar imágenes monocromáticas con formato BMP a arrays. A partir de estos arrays se podrán construir cualquier imagen en la pantalla GLCD.

El funcionamiento de esta aplicación es muy sencilla, el primer paso es seleccionar la opción “Load image” en la primera pestaña arriba a la izquierda “File”

Seleccionada la imagen a cargar, en la misma pestaña seleccionamos “Save output”, donde deberemos elegir en que carpeta queremos guardar el array.

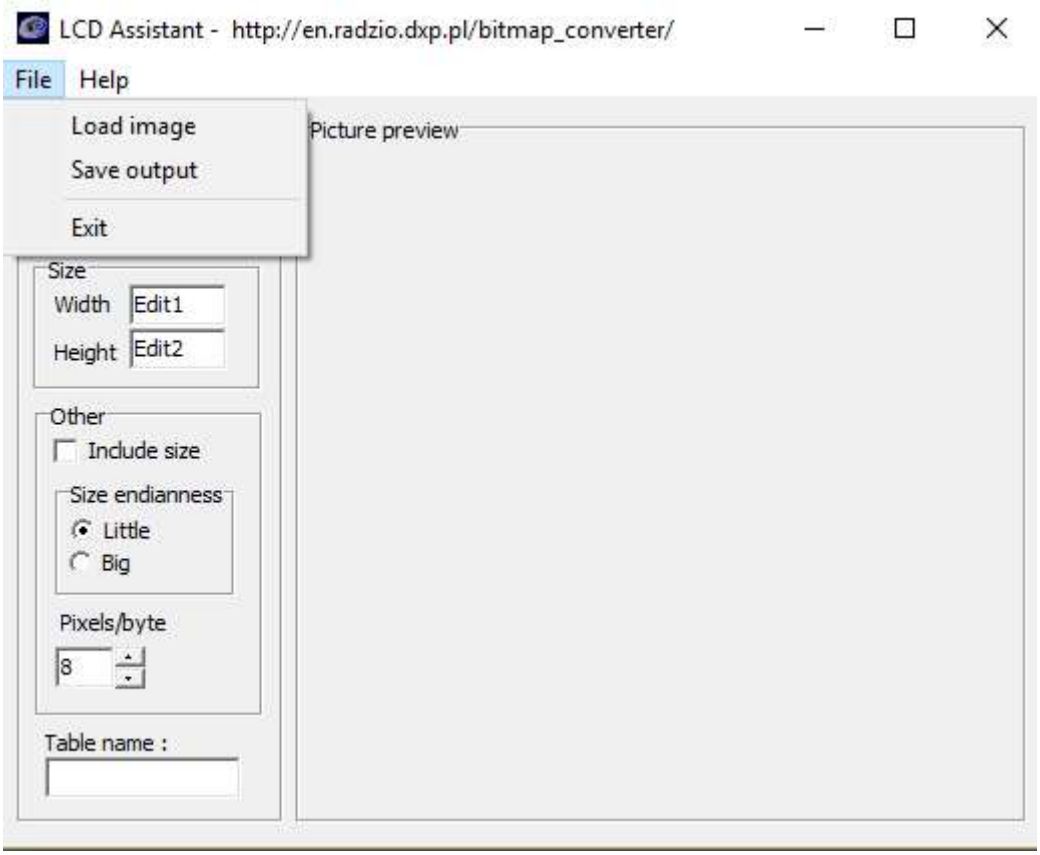


Figura 2.1 Programa LCD Assistant

El fichero de salida no tendrá ningún formato y se podrá abrir a partir del bloc de notas.

2.2 Paint

Se usará el Paint para dibujar los menús y distintos datos. Para el dibujo en sí no habrá ningún aspecto especial a tener en cuenta, sin embargo antes de empezar se deberá

ajustar la resolución al mismo de la pantalla y cuando queramos guardarlo habrá que especificar que es en formato BMP.

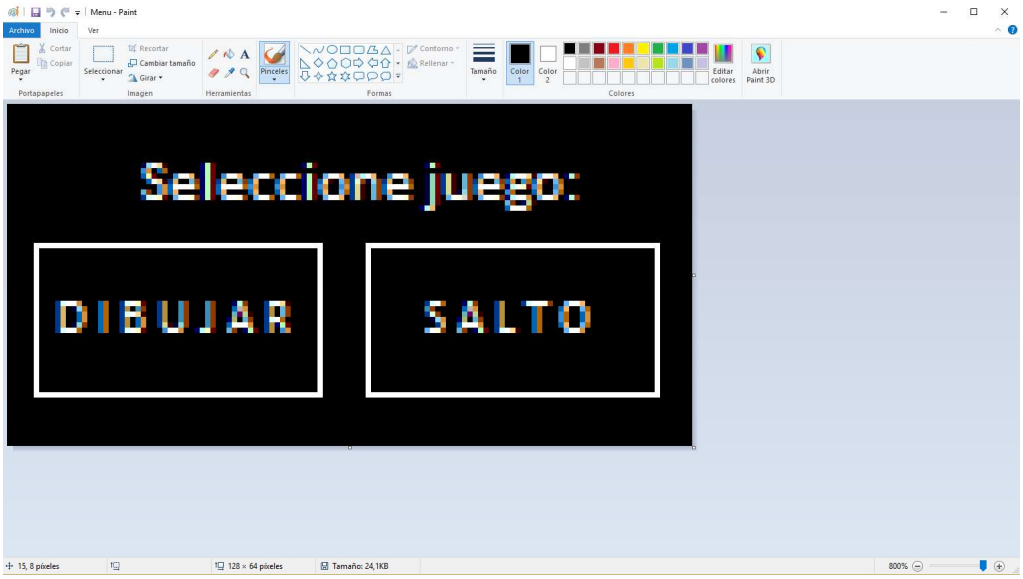


Figura 2.2 Programa Paint

2.3 AVR FLASH

Aplicación que permitirá comunicar el microcontrolador con el ordenador, siempre que esté debidamente conectado y encendido la placa. Se podrá leer, escribir, verificar o borrar lo que haya en el microcontrolador, pero para este proyecto solo se necesitará la opción de escribir, que enviará el código seleccionado, en formato HEX, al microcontrolador.

Para seleccionar el código se pulsará la pestaña de arriba a la izquierda “File” y vamos a la opción “Load HEX”. Una vez seleccionado el código se pulsará en “Write”.

Se deberá especificar a la aplicación el microcontrolador usado y la frecuencia de reloj.

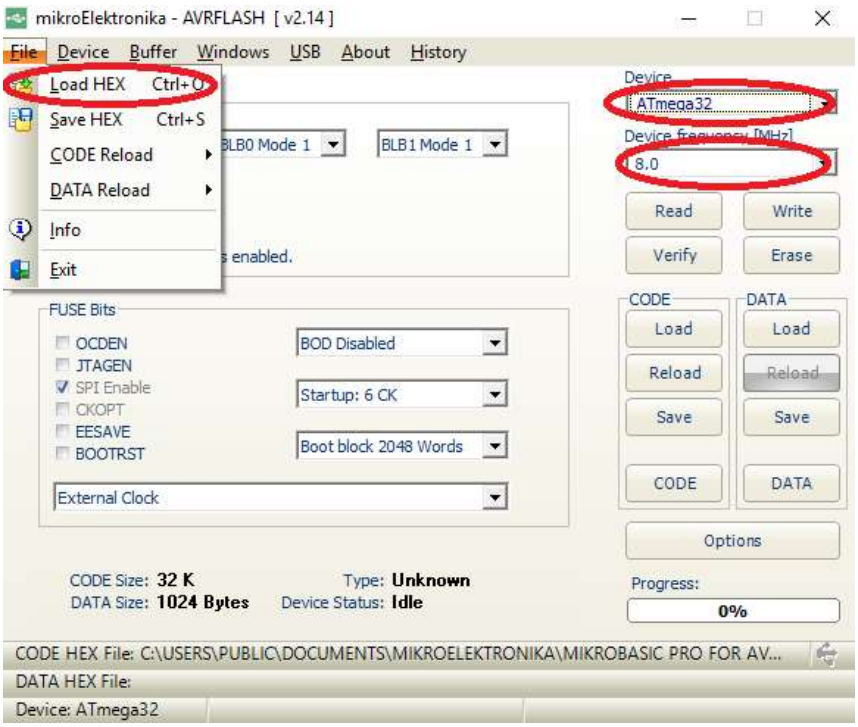


Figura 2.3 Programa AVRFLASH

2.4 AVR STUDIO 4

Será la herramienta más usada en el presente proyecto puesto es la aplicación con el que se escribirán los códigos. Tiene opciones interesantes como poder ejecutar los códigos, de forma que se pueden controlar los valores de distintos registros a medida que se van ejecutando.

Antes de empezar a programar se deberá crear un proyecto, donde se guardaran todos los archivos relacionados a nuestro código en una misma carpeta

Nada más abrir la aplicación podremos darle a la pestaña New Project:

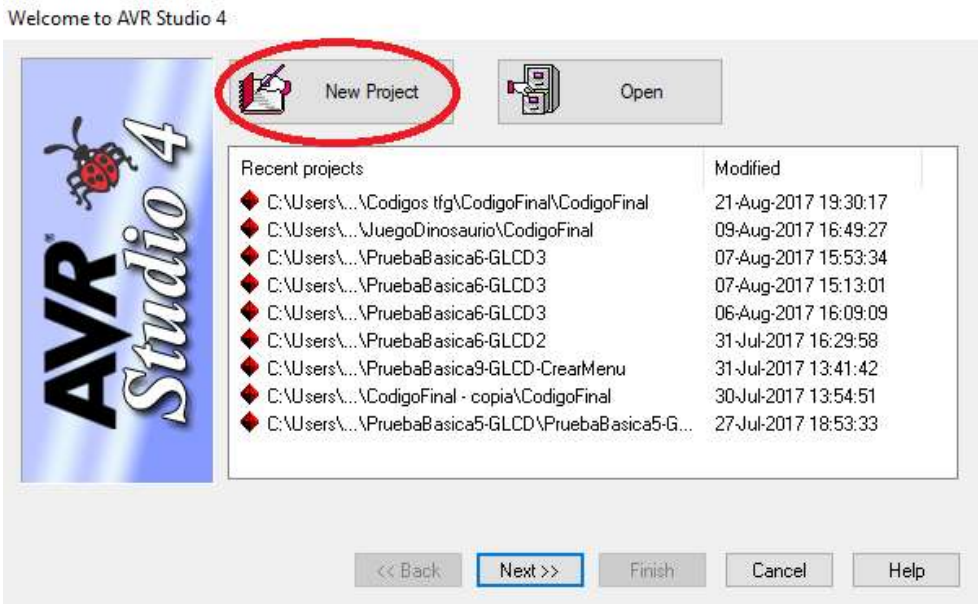


Figura 2.4 Localización New Project

Se abrirán otras opciones, en el tipo de proyecto se deberá indicar que es de Atmel AVR Assembler. También habrá que especificar el nombre del proyecto:

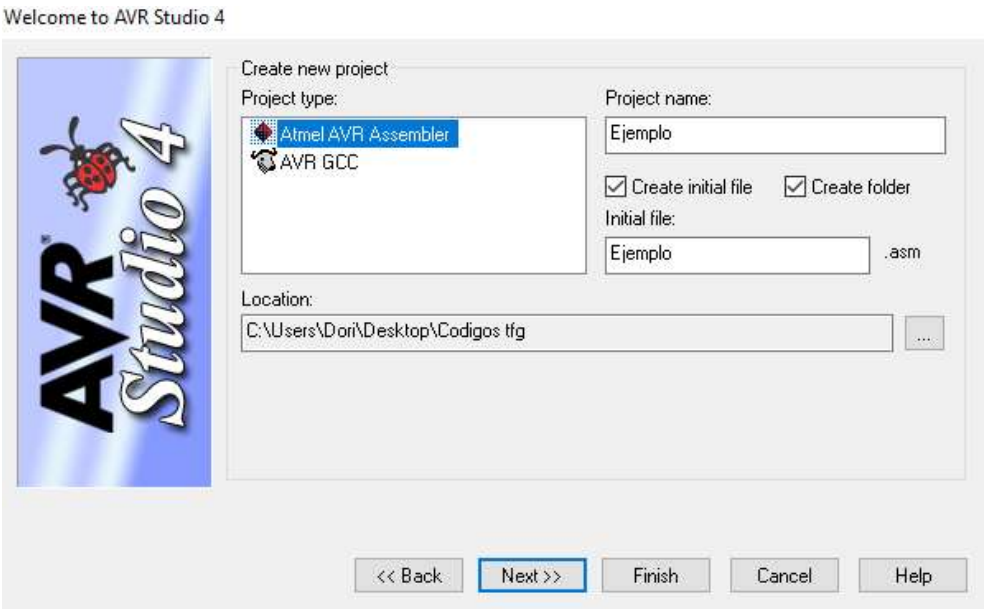


Figura 2.5 Tipo de proyecto y nombre

El último paso será especificar el microcontrolador usado. Saldrá un menú donde en la columna de la izquierda se elegirá "AVR Simulator 2". En la columna de la derecha saldrán distintos microcontroladores posibles, se elegirá el ATmega32.

Ya configurado el programa se podrá programar libremente en él. Una vez se haya terminado con el código se compilará a partir de la opción "Assemble" en la parte superior. Creará el archivo HEX, que se enviará al microcontrolador posteriormente a partir del AVR FLASH. Si existen fallos de sintaxis o de programación el programa nos avisará de ello.

Como se escribió al principio, ejecutar tus propios códigos será una herramienta potente. Existe la opción "Assemble and Run", que aparte de compilar, pondrá en marcha de manera automática el código. A partir de aquí, una flecha amarilla irá recorriendo cada línea de instrucción y lo ejecutará.

Los valores de los registros irán variando y se podrá tanto ver su contenido como modificarlo artificialmente. Se pasará una línea a otra a partir del botón F11 o F10. La diferencia es que con F10 las subrutinas se ejecutarán de forma instantánea y pasará a la siguiente línea, con F11 se introduciría en ella y ejecutará de forma individual cada una de sus instrucciones.

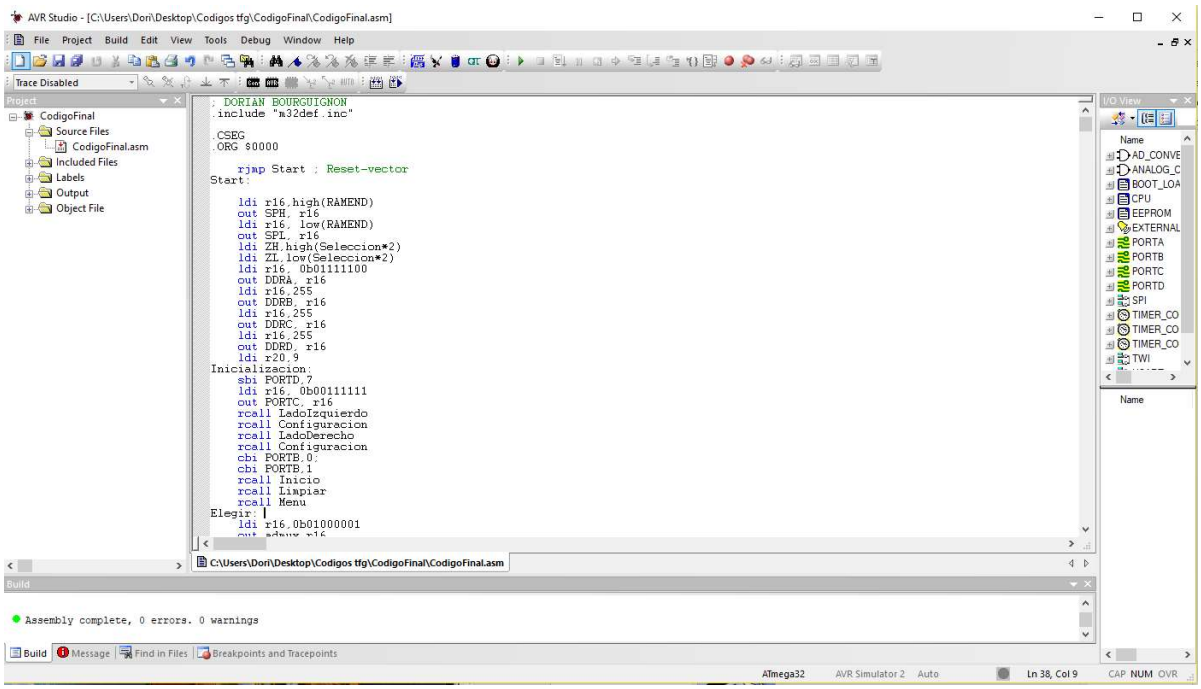


Figura 2.6 Programa AVR STUDIO 4

CAPITULO 3: DESARROLLO DEL HARDWARE

En este apartado se verá todos los elementos de la placa EasyAVR V7 usados en el proyecto.

EasyAVR V7 es la séptima generación de la placa de desarrollo AVR pensado tanto para los experimentados en el mundillo como para los recién llegados. Contiene el mikroProg, un programador y depurador, que funciona por USB y que soporta 65 microcontroladores Atmel distintos.

Figura 3.1 MikroProg

Estos tres indicarán el estado de las operaciones de programación. El led "LINK" se encenderá cuando exista una conexión por USB entre el ordenador y la placa. "ACTIVE" estará encendido cuando el programador está activado y "DATA" cuando se esté enviando información entre el programador y el ordenador.

Una de las bondades de esta placa es su posibilidad de expansión a partir de pequeñas placas de fácil conexión.

3.1 Switchs y jumpers

En la placa habrá interruptores que permitirán la configuración o la activación/desactivación de distintos periféricos. A lo largo de este capítulo se especificará en cada instrumento los valores del switch asociado.

Los jumpers tendrán el mismo objetivo, sin embargo para la realización del proyecto no se han tenido en cuenta, solamente a la hora de especificar la alimentación de la placa.

3.2 LCD de 4 bits

También llamado pantalla de cristal líquido, se usará para mostrar mensajes. Tiene dos líneas con la posibilidad de mostrar 16 caracteres en cada una de ellas. Tendrá las siguientes conexiones:

- **GND y VCC:** Necesario para alimentar al LCD. La tensión será de 5V.
- **V_0:** Tensión que regula el nivel de contraste. Será controlado por un potenciómetro.

- **RS:** Cuando está activo, la pantalla estará preparada para mostrar caracteres. Desactivado estará en modo configuración. Se controlará a partir del bit 2 de PORTA.
- **E:** Enable. Interactuando con este elemento se conseguirá enviar información al LCD. Se controlará a partir del bit 6 de PORTD.
- **R/W:** Establecerá si la pantalla está en modo escritura o de lectura. Como está conectada directamente a tierra siempre estará en modo escritura.
- **D0-D7:** Entradas principales del LCD para recibir la información a mostrar en pantalla. Este LCD al ser de 4 bits solo usará los 4 bits más significativos, de D4 a D7, que estarán conectadas a PORTC del bit 4 al 7. Los demás estarán conectadas a tierra.
- **LED+:** Conexión ánodo del LED.
- **LED-:** Conexión cátodo del LED.

Configuración switch 3:

Con SW3.1 activado se usará el brillo al máximo. Se podrá ajustar el brillo a partir de software si se activa SW3.2.

3.3 Display 7 segmentos

Elemento que consta de 8 leds para mostrar números de 0 a 9 y alguna letra. Un led será exclusivo para señalar el símbolo de punto decimal. Habrá 4 displays en la placa, quienes cada una de ellas mostrarán al mismo tiempo el mismo número. A partir de los primeros 4 bits de PORTA se controlará cuál de ellos estará encendido, y el número mostrado dependerá del valor de los bits de PORTC. Se puede ver en la figura de abajo las combinaciones de PORTC para mostrar los números de 0 a 9:

Salida por PORTC	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	0	0	0	0

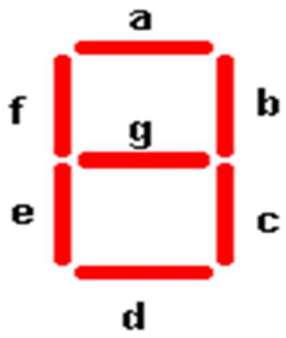


Figura 3.2 Combinaciones para formar los números y su leyenda

Configuración switch 8:

Activar de SW8.1 a SW8.4

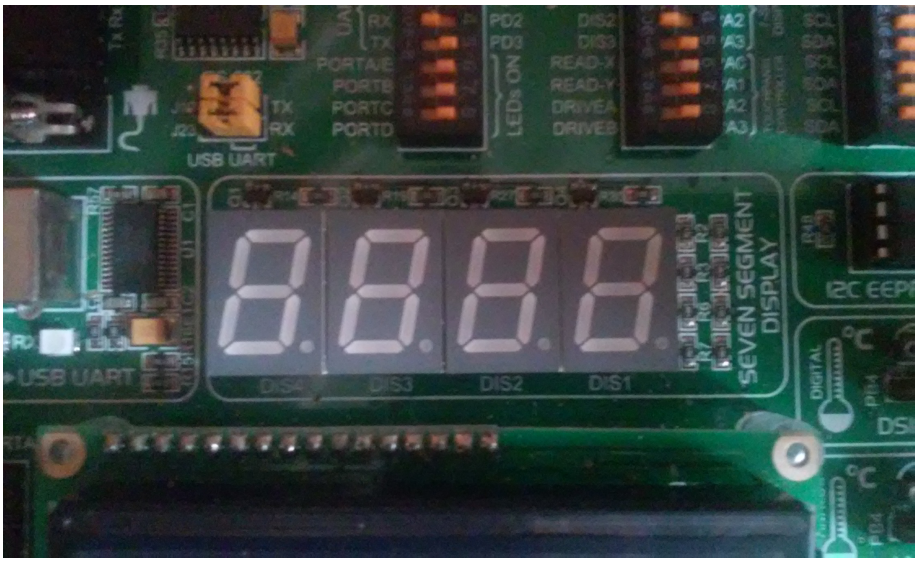


Figura 3.3 Displays 7 segmentos

3.4 LEDS

Hay 8 leds para cada PORT, representando cada una de ellas el estado de un bit. Muy usado para observar, en medio de la ejecución del programa, el estado de los bits de los PORT.

3.5 GLCD

También llamado pantalla gráfica de cristal líquido. Con una resolución de 128x64, será usado para mostrar contenido gráfico monocromático como texto, imágenes y otros contenidos.

La pantalla estará dividido en dos controladores, cada una de ellas se encargará de una parte de la pantalla. Se tendrá que especificar en el código si queremos escribir en el de la izquierda o en la derecha, aunque se podrán activar ambos si se quiere interactuar en ambas partes.

Para poder situar los pixeles se tiene dos coordenadas, X e Y. Las X también se denominan páginas y habrá 8, teniendo cada una de ellas 1 byte de información, 64 pixeles en total. La dirección Y sitúa los pixeles en la parte horizontal, cada controlador tendrá 64 pixeles.

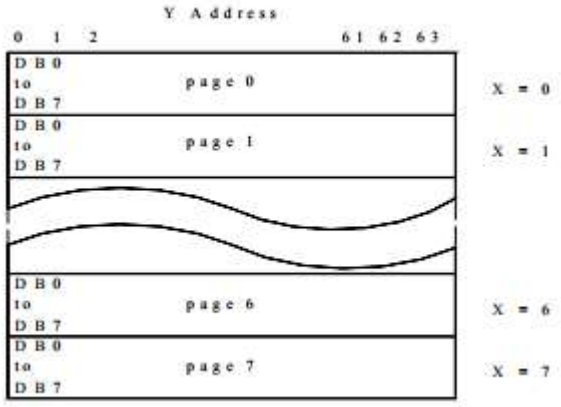


Figura 3.4 Coordenadas X e Y

A la hora de manipular los pixeles, se controla siempre una página entera a la vez de una dirección Y determinada.

Estas pantallas contienen una memoria RAM que tendrá como tamaño el mismo que la resolución de la pantalla, 128x64. En él se guardarán el estado de todos los pixeles de la pantalla.

Tendrá las siguientes conexiones:

- **VCC y GND:** Necesario para alimentar al LCD. La tensión será de 5V.
- **CS1 y CS2:** Los controladores de la pantalla. Si solo se activa CS1 la pantalla derecha estará operativa, si se activa CS2 la izquierda. CS1 estará controlado por el bit 0 de PORTB y CS2 por el bit 1.
- **V_0:** Nivel de contraste controlado por potenciómetro (P4).
- **RS:** Activado pondrá la pantalla en modo datos y desactivado en modo instrucción. Estará controlado por el bit 2 de PORTA.
- **R/W:** Determina si la pantalla está en modo escritura (0) o en modo lectura (1). Estará controlado por el bit 2 de PORTA.
- **E:** Enable. Interactuando con este elemento se conseguirá enviar información al GLCD. Se controlará a partir del bit 6 de PORTD.
- **D0 a D7:** Líneas de datos principales del GLCD para recibir o enviar información. Controlados por PORTC.
- **RST:** Señal de reset. Cuando está activado no se podrá interactuar con el GLCD. Controlado por el bit 7 de PORTD.
- **Vee:** Voltaje de referencia para el contraste del GLCD.
- **LED+:** Conexión ánodo del LED.
- **LED-:** Conexión cátodo del LED.

Configuración switch 3:

Con SW3.1 activado se usará el brillo al máximo. Se podrá ajustar el brillo a partir de software si se activa SW3.2.

3.6 Controlador de panel táctil

Panel de vidrio cuya superficie está cubierta con dos capas de material resistivo. Será de 4 hilos y registrará de manera precisa presiones, representándola en forma de voltajes analógicos.

Se deberá especificar si se quiere leer la parte vertical u horizontal de la presión a partir de los bits 2 y 3 de PORTA, dependiendo de cuál se elija enviará la tensión analógica por el bit 0 o 1 del mismo PORT.

Configuración swtich 8:

Activar de SW8.5 a SW8.8.

3.7 Microcontrolador ATmega32

Microcontrolador RISC (Reduced Instruction Set Computer) de la familia AVR del fabricante estadounidense Atmel.

El ATmega 32 es un microcontrolador de alto rendimiento y bajo consumo de 8 bits. Al ejecutar instrucciones potentes en un solo ciclo de reloj, permite optimizar el consumo y conseguir velocidad de procesamiento.

Alcanza los 16 MIPS (Millones de instrucciones por segundo), tiene 32k bytes de memoria flash, 2k bytes de memoria interna SRAM y 1k byte de memoria EEPROM.

Físicamente este microcontrolador abarca 40 pines, donde 32 de ellos tendrán propósitos generales de I/O divididos en 4 PORT. Cada uno de estos 32 pines se podrá configurar como entrada o salida, dependiendo del uso que se le dé en el código. Como entrada recibirá información y como salida la transmitirá.

Se usarán los dos bits menos significativos de PORTA para leer las tensiones del GLCD y transformarlo a señales digitales.

Este microcontrolador consta de 32 registros, etiquetados desde R0 hasta R31, con 8 bits de capacidad y que se usarán para almacenar números o para realizar operaciones simples. Los primeros 16 registros no podrán cargar un valor de forma inmediata, sino a base de pasos intermedios, por lo que no serán muy usadas en el presente proyecto. Existen parejas de registros que pueden desempeñar el papel de punteros si se les da la dirección de memoria. Los punteros serán explicados más detenidamente en el capítulo del desarrollo del software.

Existe en el microcontrolador la RAM Estática (SRAM), un tipo de memoria que no es accesible directamente por la CPU. Será usado como pila, torre de bloques con estructura LIFO (last in, first out) que servirá para ir almacenando datos en la pila y recuperarlo más tarde. Esto será posible a partir de la definición del puntero de pila de 16 bits, donde SPH es la parte más significativa y SPL la menos. El puntero se cargará con el valor de la dirección más alta de la SRAM.

```
ldi r16,high(RAMEND)
out SPH, r16
ldi r16, low(RAMEND)
out SPL, r16
```

Figura 3.5 Inicialización de la pila

Gracias a la pila se podrá usar las subrutinas. Las subrutinas son un conjunto de instrucciones que serán accesibles cuando sean llamadas desde el bloque principal. Servirán para mantener limpio el código principal, sobre todo si son una secuencia de instrucciones que se repiten con regularidad. La utilidad de la pila en este caso es que a la hora de llamar a la subrutina y acceder a la nueva dirección de memoria, se guarda en la pila la dirección desde donde se llama, de forma que cuando esta subrutina acabe se puede continuar desde ahí.

En las mismas subrutinas son bastante común a partir de los comandos push y pop quienes tienen como objetivo guardar valores de registros en la pila de forma que sus valores se mantengan en el bloque principal del programa.

Las interrupciones tienen un concepto muy parecido a las subrutinas, serán conjunto de instrucciones agrupadas en distintas direcciones de memoria. La principal diferencia será que no son llamadas de forma manual por el bloque principal, si no que se activarán al cumplirse alguna condición. Esto será realmente útil para situaciones donde se debe actuar rápidamente ante algún estímulo ya que el acceso será inmediato y no habrá espera a finalización de alguna instrucción. Existen 21 tipos de interrupciones, donde las diferencias entre ellas será el tipo de estímulo que lo activa. Está por ejemplo el que se activa al contar un determinado tiempo o la activación de un pin. Como en el caso de la subrutina, se guardará en la pila la dirección de la instrucción a ejecutar del bloque principal. No se usarán las interrupciones en los códigos del proyecto, sin contar con RESET, interrupción provocada al encender el microcontrolador y que servirá para empezar el funcionamiento del código.

Los retardos son elementos muy usados en la programación que tienen como función mantener un estado de inactividad, donde el microcontrolador no hace ninguna función aparente. En los códigos del proyecto será muy utilizado para las comunicaciones con las pantallas GLCD y LCD, y para las conversiones ADC.

Cuando se necesita un retardo se llamarán a subrutinas que desempeñan ese papel. En estas subrutinas se aprovecharán el hecho de que cada instrucción tarda un determinado tiempo en realizarse para crear numerosos bucles. Se usarán 3 registros que dependerán entre sí de manera jerárquica, el registro de menor nivel irá reduciendo su valor hasta llegar a 0, cuando ocurra, el registro de nivel intermedio bajará de valor y restablecerá el valor del registro de menor nivel. Cuando el registro de nivel intermedio sea 0, será el momento de reducir el de mayor nivel y restablecerá los otros dos registros. Cuando este registro alcance las 0 unidades se dará por finalizada el retardo. El tiempo de retardo dependerá de los valores de estos registros, por lo que se tendrá varias subrutinas a utilizar según el tiempo requerido.

```
Retardo100us:
    ldi r25, 2
bucle1100us:
    dec r25
    ldi r26, 10
    subi r25, 0
    brne bucle2100us
    ret
bucle2100us:
    ldi r27, 1
    subi r26, 0
    breq bucle1100us
    dec r26
bucle3100us:
    dec r27
    subi r27,0
    breq bucle2100us
    rjmp bucle3100us
```

Figura 3.6 Estructura de los retardos

A continuación se enumerará las instrucciones utilizadas en el código:

- **ADD:** Suma de dos registros.
- **SUB:** Resta de dos registros.
- **SUBI:** Resta una constante a un registro.
- **AND:** And lógica entre dos registros.
- **ANDI:** And lógica entre un registro y una constante.
- **OR:** Or lógico entre dos registros.
- **ORI:** Or lógico entre un registro y una constante.
- **SBR:** Activar bit determinado del registro.
- **CBR:** Desactivar bit determinado del registro.
- **INC:** Incremento.
- **DEC:** Decremento.
- **CLR:** Pone a 0 todos los bits del registro.
- **SER:** Pone a 1 todos los bits del registro.
- **RJMP:** Salto hacia otra parte del código.
- **RCALL:** Llamada a la subrutina.
- **RET:** Finalización subrutina.
- **CP:** Compara dos registros.
- **SBRC:** Salta la siguiente instrucción si el bit estudiado del registro es 0.
- **SBRS:** Salta la siguiente instrucción si el bit estudiado del registro es 1.
- **SBIC:** Salta la siguiente instrucción si el bit estudiado del PORT es 0.
- **SBIS:** Salta la siguiente instrucción si el bit estudiado del PORT es 1.
- **BREQ:** Salto hacia otra parte del código si el resultado de la última operación es 0.
- **BRNE:** Salto hacia otra parte del código si el resultado de la última operación es distinto de 0.
- **BRSH:** Salto hacia otra parte del código si el resultado de la última operación es positivo.
- **BRLO:** Salto hacia otra parte del código si el resultado de la última operación es negativo.
- **MOV:** Traspasa el valor de un registro a otro.

- **PUSH:** Guarda un registro en la pila.
- **POP:** Recupera un registro de la pila.
- **SBI:** Setea el valor de un bit de PORT.
- **CBI:** Pone a 0 el valor de un bit de PORT.
- **LSL:** Desplazamiento de bits a la izquierda.
- **LDI:** Carga al registro un valor inmediato.
- **LPM:** Carga a R0 el valor de la dirección apuntada por el puntero.
- **IN:** Carga al registro el valor de un PORT

CAPITULO 4:
Desarrollo del código

4.1 LEDS

Antes de empezar con el dispositivo táctil se decidió interactuar con otros elementos más fáciles de la placa para ir cogiendo soltura a la hora de programar en ensamblador en ATmega32 y para familiarizarse con la placa EasyAVR v7.

El primer elemento y el más básico serán los leds. El funcionamiento en sí es bastante simple, los leds asociados al PORT se encenderán cuando los pines se enciendan. Se probó el funcionamiento de los leds a partir de un simple código, se puede ver el resultado en la imagen.

```
.include "m32def.inc"

.CSEG
.ORG $0000
    rjmp Start

Start:
    ldi r16,high(RAMEND)
    out SPH, r16
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, 255
    out DDRA, r16
    out DDRC, r16
    ldi r16,0b00001011
    out PORTA, r16
    ldi r16,0b00000011
    out PORTC, r16


bucle:
    rjmp bucle
```

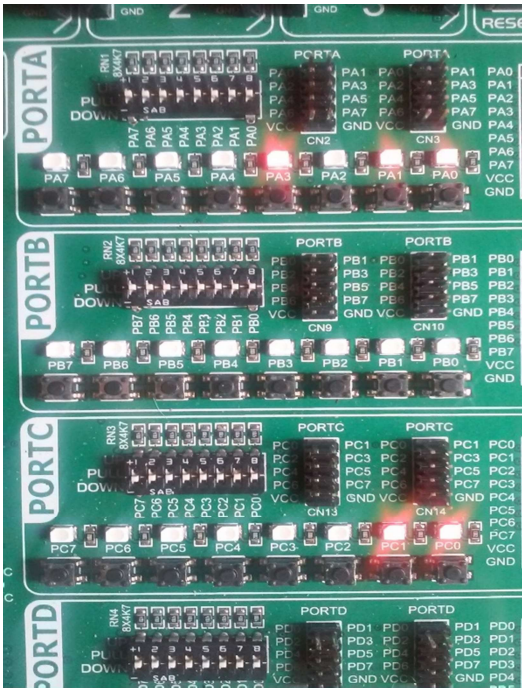


Figura 4.1 Código para encender leds y su resultado

Se observa en el código que todo PORTA y PORTC se comportará como salida, y tendrán como salida 00001011 y 00000011 respectivamente. El resultado es que los leds se encienden con el mismo orden que los bits de los puertos de salida.

Los problemas encontrados para este código serán los asociados a la falta de experiencia en el uso de la placa y del modelo del microcontrolador. Es necesario incluir al inicio del código un include para el microcontrolador equivalente, y fue necesario buscar en la red el código a incluir, en este caso “m32def.inc”. Se tuvo también que configurar la aplicación AVRFLASH para que esta se pueda comunicar con el microcontrolador, para ello se tuvo que especificar que nuestro microcontrolador es ATmega32 y la frecuencia a la que está funcionando (8MHz).

4.2 DISPLAYS 7 SEGMENTOS

El siguiente paso a seguir fue la programación de los displays 7 segmentos. Se propuso hacer un código basado en el ejercicio de clase de la asignatura, donde contábamos la cantidad de veces que se ha pulsado un pulsador, mostrando esta cantidad de dos dígitos en dos displays. Una vez alcanzado el número 15, los displays irán reduciendo el valor hasta llegar a 0.

La principal diferencia con respecto a la placa usada en clase es la ausencia del decodificador BCD a 7 segmentos, quien se encargaba de transformar el número que sale de PORTC para encender los segmentos del display pertinentes. Por ejemplo, si queremos mostrar el número 0, el decodificador cambiaba 00000000 por 00111111.

Para solucionar esta ausencia se introdujo en el código una especie de "traductor", quien dependiendo del número leído en el registro r16 sacará por PORTC el número modificado según la tabla.

```
Traductor:
    cpi r16, 1
    breq NumeroUno
    cpi r16, 2
    breq NumeroDos
    cpi r16, 3
    breq NumeroTres
    cpi r16, 4
    breq NumeroCuatro
    cpi r16, 5
    breq NumeroCinco
    cpi r16, 6
    breq NumeroSeis
    cpi r16, 7
    breq NumeroSiete
    cpi r16, 8
    breq NumeroOcho
    cpi r16, 9
    breq NumeroNueve
```

```
NumeroCero:
    ldi r16,0b00111111
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroUno:
    ldi r16,0b00000110
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroDos:
    ldi r16,0b01011011
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroTres:
    ldi r16,0b01001111
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroCuatro:
    ldi r16,0b01100110
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroCinco:
    ldi r16,0b01101101
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroSeis:
    ldi r16,0b01111101
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroSiete:
    ldi r16,0b00000111
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroOcho:
    ldi r16,0b01111111
    sbrs r17,0
    rjmp Siguiente
    rjmp SegundoNumero
NumeroNueve:
    ldi r16,0b01101111
    sbrc r17,0
    rjmp SegundoNumero
```

Figura 4.2 Solución a la falta de decodificador

Como tenemos dos displays se deberá repetir el proceso otra vez donde el valor de r17 nos indicará si ya hemos leído el segundo número o no.

En el proceso de este código sucedió un error inesperado, en un mismo display se mostraba los dos números a la vez. Los displays encendidos muestran el número que sale por portc, para conseguir parecer que cada display muestra un número distinto el truco estaba en ir apagando uno de ellos, y mostrar en el otro el número correspondiente y viceversa con una frecuencia muy alta, de forma que engañamos a nuestra vista. A pesar de estar basado en el código usado en clase, de alguna manera los números se seguían filtrando por lo que para solucionarlo se muestra el número solo en un instante de tiempo y luego se apaga, así para los dos displays. El resultado es que se consigue solucionar la filtración, pero los números de los displays se ven con menos fuerza, al estar gran parte del tiempo apagado.

4.3 Pantalla LCD

El último paso antes de empezar con la pantalla GLCD fue la programación de la pantalla LCD. A través del datasheet incluido en el anexo, el primer paso fue inicializar la pantalla, para ello se envían por PORTC distintas combinaciones de bits para ir configurándolo. Para hacer llegar la información a la pantalla se debe primero activar el bit 6 de PORTD, correspondiente al enable, sacar por PORTC la configuración deseada y después desactivar el enable, como vemos en la figura de abajo:

```
Envio:
    push r16
    sbi PORTD, 6
    out PORTC, r16
    cbi PORTD, 6
    pop r16
    ret
```

Figura 4.3 Subrutina para enviar información al LCD

Este paso se hará dos veces para configuración, puesto que la pantalla es de 4 bits y no se podrá enviar toda la información a partir de un solo envío, es necesario enviar primero los 4 bits más significativos y después los menos significativos. Además previamente debemos asegurarnos de que en todo el proceso de configuración, estemos en modo instrucción, manteniendo en bajo nivel el bit 2 de PORTA. Entre cada envío de configuración se necesita entremedio un retardo.

```
ConfiguracionLCD:
    rcall Retardo100ms
    cbi PORTA,2
    ldi r16, 0b00110000
    rcall Envio
    rcall Retardo4ms
    ldi r16, 0b00110000
    rcall Envio
    rcall Retardo100us
    ldi r16, 0b00110000
    rcall Envio
    rcall Retardo100us

    ldi r16, 0b00100000
    rcall Envio
    rcall Retardo100us

    ldi r16, 0b00100000
    rcall Envio
    ldi r16, 0b10000000
    rcall Envio
    rcall Retardo100us

    ldi r16, 0b00000000
    rcall Envio
    ldi r16, 0b10000000
    rcall Envio
    rcall Retardo100us

    ldi r16, 0b00000000
    rcall Envio
    ldi r16, 0b00010000
    rcall Envio
    rcall Retardo4ms

    ldi r16, 0b00000000
    rcall Envio
    ldi r16, 0b01100000
    rcall Envio
    rcall Retardo100us

    ldi r16, 0b00000000
    rcall Envio
    ldi r16, 0b11000000
    rcall Envio
    rcall Retardo100us

    sbi PORTA,2
```

La configuración usada en el código será la siguiente:

- Modo 4 bits.
- Carácter de 5x7 puntos.
- Dos líneas.
- El cursor no será visible.
- Se colocará el cursor en la primera posición.
- El cursor se moverá de izquierda a derecha.
- No se desplaza la visualización cada vez que se escriba un dato.

Figura 4.4 Configuración LCD

Una vez configurada la pantalla, esta está preparada para mostrar mensajes. Primero se ha de activar el bit 2 de PORTA, para hacerle saber a la pantalla que va a recibir caracteres.

El LCD es capaz de mostrar todos las letras del abecedario (tanto en mayúsculas como en minúsculas), cualquier número y distintos símbolos como %,< o &. Para cada uno de ellos estará asociado un número en hexadecimal, para mostrarlo el procedimiento será idéntico a enviar una configuración, se activa el enable, se envía por PORTC el carácter y se desactiva el enable (Se recuerda que al ser una pantalla de 4 bits se deben enviar en dos tandas).

Por ejemplo, la letra H:

```
ldi r16,0b01000000
rcall Envio
ldi r16,0b10000000
rcall Envio
rcall Retardo100us
```

Figura 4.5 Extracto del código donde se envía el mensaje y resultado final

4.4 GLCD

El último periférico a programar será el GLCD, cuyo objetivo final será diseñar dos juegos sencillos.

4.4.1 Configuración

La pantalla GLCD tendrá, como en la pantalla LCD, que ser configurado al principio y el procedimiento será parecido. Tendremos, primero, que desactivar la señal de reset con un 1, que se encuentra en el bit 7, a partir de este momento se podrá interactuar con la pantalla. A continuación se debe encender la pantalla, simplemente introduciendo a PORTC el número binario: 00111111.

Como se explicó anteriormente en el presente proyecto, esta pantalla consta de dos controladores, quienes manejaran cada una de ellas una mitad, por lo que se debe asegurar que cada una de ellas esté activada. Se ha diseñado una subrutina para cada una de las partes, en cada una de ellas se interactuará con los bits 0 y 1 de PORTB, como se ve en el extracto del código de abajo:

```
LadoIzquierdo:
    cbi PORTB,0
    sbi PORTB,1
    ret

LadoDerecho:
    sbi PORTB,0
    cbi PORTB,1
    ret
```

Figura 4.6 Subrutinas para activar una parte de la pantalla

Después de la llamada de cada una de estas subrutinas, se llamará a otra subrutina donde se activará y desactivará el enable (bit 6 de PORTD) con dos retardos de 100 microsegundos. A diferencia del código de la pantalla LCD, en esta ocasión se diferenciará este proceso en dos subrutinas distintas, uno para la configuración y otro para

el encendido/apagado de pixeles, ya que el bit 2 de PORTA, asociado al registro RS, estará desactivada y activada respectivamente.

```
Configuracion:
    push r16
    cbi PORTA,2
    cbi PORTA,3
    sbi PORTD,6
    rcall Retardo100us
    cbi PORTD,6
    rcall Retardo100us
    pop r16
    ret
```

```
Envio:
    push r16
    sbi PORTA,2
    cbi PORTA,3
    sbi PORTD,6
    rcall Retardo100us
    cbi PORTD,6
    rcall Retardo100us
    pop r16
    ret
```

Figura 4.7 Subrutinas para enviar información a la pantalla. El de la izquierda para configurar y el de la derecha para mostrarlo por pantalla.

La primera parte de la configuración quedaría de la siguiente manera:

```
Programa:

    sbi PORTD,7
    ldi r16, 0b00111111
    out PORTC, r16

    rcall LadoIzquierdo
    rcall Configuracion
    rcall LadoDerecho
    rcall Configuracion
```

Figura 4.8 Inicialización de la pantalla

Una vez encendido la pantalla y activado los dos controladores, se propuso proyectar alguna palabra o frase y para ello era necesario especificar al GLCD varias cosas previas. El primero de ellas será elegir en cuál de las pantallas queremos encender o apagar pixeles, llamando a la misma subrutina usada antes, LadoIzquierdo o LadoDerecho, no teniendo en esta ocasión la necesidad de llamar a Configuración. En el proceso del código se descubre la posibilidad de activar los dos controladores a la vez, el resultado es que en las dos partes de la pantalla se proyecta la misma información a la vez. Se debe especificar también las direcciones X(página) e Y, donde se empezará la manipulación de los pixeles, a partir de ahí la dirección Y irá avanzando automáticamente. El proceso será el visto anteriormente, se enviará un número binario por PORTC, y se llamará a la subrutina Configuracion para activar y desactivar los enable. El número binario a enviar para el caso de las paginas 10111XXX y para el caso de Y 11XXXXXX, donde las X representan la dirección deseada. Por ejemplo:

```
rcall LadoIzquierdo

ldi r16,0b01000111
out PORTC, r16
rcall Configuracion

ldi r16,0b10111001
out PORTC, r16
rcall Configuracion
```

Figura 4.9 Configuración de la dirección deseada

Para este caso estaríamos activando el lado izquierdo e iniciando el mensaje en la página 1 y la dirección 7.

4.4.2 Mensajes

Para cada dirección de Y podemos interactuar con todos los pixeles de una página (1 byte), especificando cual se enciende (1) y cual se apaga (0). Uno de los fallos cometidos en el desarrollo del código fue no darse cuenta que a la hora de activar la pantalla, por defecto, los pixeles ya están encendidos, y a la hora de intentar encender algunos al azar no mostraba diferencia alguna o se mostraba incorrectamente, por lo que se intenta apagar los pixeles necesarios para mostrar un mensaje.

Controlando los valores de las paginas en una cantidad suficiente de espacios de Y fue la mantera de mostrar el mensaje “Hola”, enviando para cada valor de Y por PORTC el numero binario asociado a los pixeles encendidos. Para cada uno de los valores se llamaba en esta ocasión la subrutina Envio, para poder ser entregado al LCD.

```
ldi r16,0b00000001      ;Empieza H
out PORTC, r16
rcall Envio

ldi r16,0b11101111
out PORTC, r16
rcall Envio

ldi r16,0b11101111
out PORTC, r16
rcall Envio

ldi r16,0b00000001      ;Termina H
out PORTC, r16
rcall Envio

ldi r16,0b11111111      ;ESPACIO
out PORTC, r16
rcall Envio

ldi r16,0b11111111      ;ESPACIO
out PORTC, r16
rcall Envio

ldi r16,0b00000001      ;Empieza O
out PORTC, r16
rcall Envio

ldi r16,0b01111101
out PORTC, r16
rcall Envio

ldi r16,0b01111101
out PORTC, r16
rcall Envio

ldi r16,0b00000001      ;Termina O
out PORTC, r16
rcall Envio
```

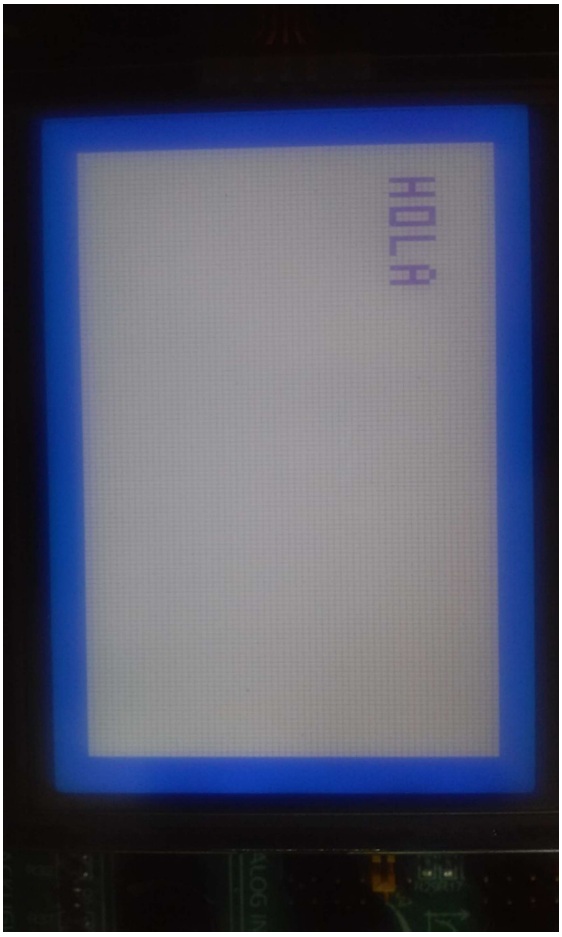


Figura 4.10 Extracto del código donde se envía el mensaje y resultado final

Este método de escritura es bastante poco práctico y muy sensible a cambios de última hora, ya que ir recorriendo todos los pixeles de la pantalla para mostrar, por ejemplo, una imagen es un trabajo arduo y lento. Es por estos motivos por el que se intentó buscar otro método más práctico.

4.4.3 TABLAS

La técnica usada para resolver estos inconvenientes es a partir del uso de tablas que contengan la información que se va a presentar en pantalla. La idea será ir recorriendo por toda la tabla a medida que se va mostrando su contenido. Se consigue el contenido de la tabla a partir del programa LCDAssistant, quien lo obtiene a partir de una imagen en formato BMP y se deberá asegurar que la imagen a transformar tenga las medidas 128x64, que corresponden con la de la pantalla.

El siguiente problema a resolver fue como recorrer la tabla, que finalmente fue resuelta a partir del uso de punteros. Existe la posibilidad de crear tres pareja de registros; r16:r27, r28:r29 o r30:r31, quienes tendrían cada uno una dirección de memoria de 8 bits. Los registros r27, r29 y r31 tienen la parte alta de la dirección y los registros r16, r28, r30 la parte baja.

En la práctica, solo se ha usado una pareja, r30 y r31 donde el registro alto se denomina ZH (high) y el bajo ZL (low). La inicialización del puntero se podrá observar en la figura X de abajo:

```
ldi ZH,high(Tabla*2)
ldi ZL,low(Tabla*2)
```

Figura 4.11 Inicialización del puntero

ZH y ZL obtendrán la dirección alta y baja respectivamente multiplicado por dos, esto es debido a que se debe ajustar la dirección hacia el formato de los 16 bits.

Como la pantalla se divide en dos partes, se debe ajustar en qué momento se activa una u otra, a medida que vamos dibujando la pantalla. Para ello tenemos un registro que utilizaremos como contador, una vez que sobrepase los 64 pixeles de la pantalla actual, se activa la otra y vuelve a contar desde el principio. Se utilizará otro registro contador para saber en todo momento qué página de la pantalla está activada, una vez sobrepasado las 8 páginas de la pantalla se finalizará el proceso.

Se usan varias subrutinas para realizar el proceso:

- IZQ activará la pantalla izquierda y se asegurará que empiece la interactuación con los pixeles desde la izquierda en la página adecuada.
- DER activará la pantalla derecha y se asegurará que empiece la interactuación con los pixeles desde la izquierda en la página adecuada.
- Dibujar manejará el puntero para leer el contenido de la tabla e ir mostrándolo a la pantalla hasta llegar al límite de pixeles de la pantalla.

El manejo del puntero se consigue de la siguiente manera, primero se lee el contenido del puntero y se guarda en r0. Este contenido se vuelca a la pantalla y la

dirección apuntada por el puntero se aumenta en 1. El registro que cuenta la cantidad de pixeles en Y disminuye, si llega a 0 se acaba la subrutina, sino se vuelve a empezar.

El resultado es que una vez completado el código, se podía escribir cualquier cosa o dibujar con una facilidad no posible de otra forma:

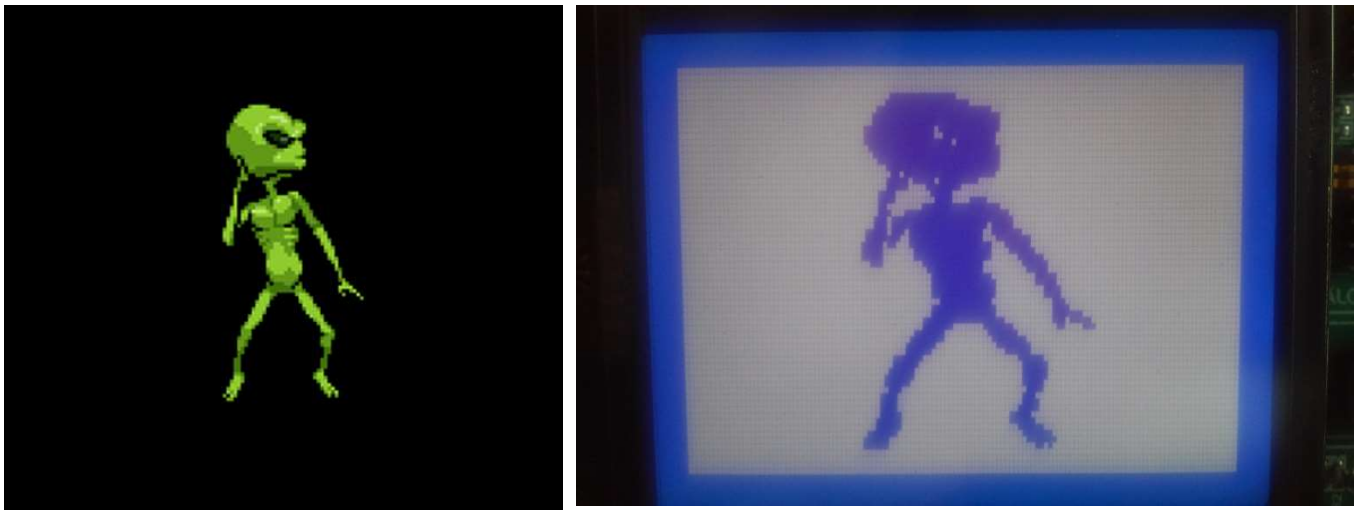


Figura 4.12 Ejemplo del funcionamiento para mostrar imágenes

Antes de entrar en el código para programar la parte táctil, se diseñó primero una subrutina que apague todos los pixeles, ya que como se comentó más arriba, de forma predeterminada la pantalla tendrá todos los pixeles encendidos.

El procedimiento será parecido al visto en el apartado de tablas, sin embargo en esta ocasión en vez de ir obteniendo valores de la tabla siempre se introducirá por PORTC el byte 00000000, el equivalente a apagar los pixeles. Además, se mantendrá en todo momento los dos controladores activos a la vez, se recorrerá los 64 pixeles de las pantallas y se irá pasando de página en página hasta recorrer toda la pantalla.

4.4.4 Pantalla táctil y dibujo

En este apartado se explicará el desarrollo del código para hacer funcionar la pantalla táctil, la parte más compleja del presente proyecto.

Si el bit 3 de PORTA está activada y el bit 2 desactivada, la pantalla táctil sacará un mayor valor de tensión a mayor altura se pulse, de la manera contraria, sacará mayor tensión cuando se pulse a la derecha de la pantalla. A partir de un tester, se pudo comprobar cuáles son los valores mínimos y máximos de tensión en cada caso:

- Vertical → 0.7v a 4.02v

- Horizontal→0.32v a 4.07v

Se necesitará un conversor que transforme esos valores a digitales para que pueda ser usado por el microcontrolador. Para ello existen varios registros en el microcontrolador Atmega32 que serán muy usados para este apartado, ADMUX y ADCSRA, registros encargados de la conversión de señales analógicas a digitales.

ADMUX será el registro encargado de configurar esta conversión, se podrá elegir el voltaje de referencia, elegir la presentación del resultado y elegir por cual pin será leído la señal analógica. Para nuestro código se elegirá como referencia 5v de alimentación (AVCC), el pin usado dependerá si queremos medir en vertical u horizontal, ya que el pin leído será distinto.

ADCSRA se usará para iniciar la conversión, se deberá activar de nuevo cada vez que esta conversión termine. También avisará cuando la conversión se haya acabado.

El valor convertido se guardará en un registro especial de 10 bits llamado ADC, dividido en dos partes, ADCH y ADCL. ADCH contendrá los 2 bits más significativos y en ADCL los restantes. Un error cometido a la hora de tratar con este registro es que no se consideró que su valor nunca se actualiza al siguiente resultado de conversión hasta que cada una de las partes no se haya guardado en un registro, será importante repetir esta práctica para cada conversión.

Una vez empezado la conversión, se revisará continuamente en bucle el valor del bit 4 de ADCSRA, que se activará cuando haya finalizado. Inicialmente se intentaba obtener el valor de la conversión una vez finalizada esta, sin embargo no daba el resultado esperado porque no daba tiempo a guardar el valor, por lo que se decidió poner un retardo.

Se intentó primero un código sencillo donde solamente se probase la parte vertical de la pantalla táctil. El objetivo de esta pequeña prueba era conseguir que justo donde pulsemos se encienda la página entera correspondiente. Para ello primero se configura ADMUX de forma que lea el pin 1 de PORTA, y se activa y desactiva el bit 3 y 2 respectivamente de PORTA. Una vez medido y convertido el valor, el siguiente problema fue como interpretarlo. Se sabe que el valor máximo y mínimo será de 4.02 y 0.7 voltios respectivamente, y sabiendo que 5v tendrá como resultado en ADC en binario 11(ADCH) 11111111(ADCL) (1023) debido a que es el mismo al de referencia, con una simple regla de tres se llega a la conclusión que el valor obtenido para 3.9v es de 823 y el de 0.7v es de 143.

Como el objetivo es encender la página adecuada, se debe establecer cuáles son los límites de cada una de ellas. Se sabe que hay 8 páginas en total, dividiéndolo entre el rango de valores de mínimo a máximo obtenemos estos límites:

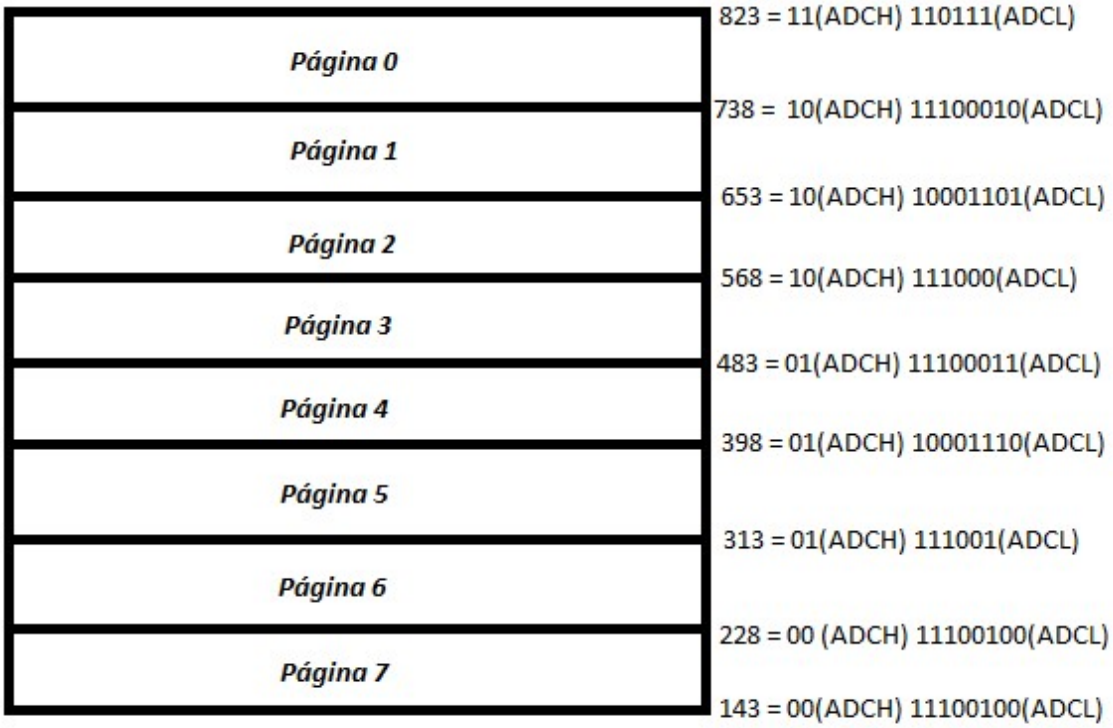


Figura 4.13 Valores límites que separan una página de otra

Así pues, se leerá primero ADCH y según su valor se comparará el valor ADCL con los límites para establecer cual página es el que debe encenderse:

```
in r16, adcl
in r17, adch
cpi r17,3
breq PrimerPaso
cpi r17,2

breq Duda
cpi r17,1
breq Duda2
rjmp Duda3
```

```
Duda3:
    cpi r16, 143
    brlo Rebucle
    cpi r16,228
    brsh Pagina6
    rjmp Pagina7

Duda2:
    cpi r16,227
    brsh Pagina3
    cpi r16,142
    brsh Pagina4
    cpi r16,57
    brsh Pagina5
    rjmp Pagina6

Duda:
    cpi r16, 226
    brsh Pagina0
    cpi r16,141
    brsh Pagina1
    cpi r16,56
    brsh Pagina2
    rjmp Pagina3
```

Figura 4.14 Extracto del código donde diferencia una página u otra según los valores leidos en la conversión

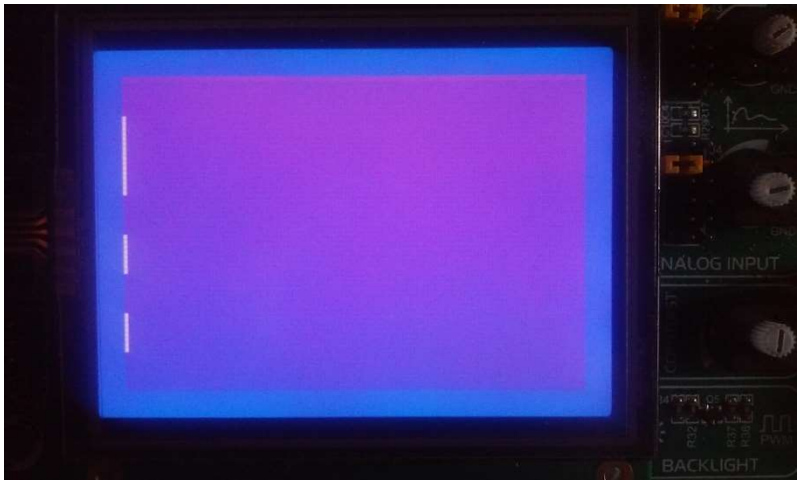


Figura 4.15 Primer resultado, se encienden páginas enteras

El siguiente paso será conseguir encender el pixel exacto, y no toda la página entera como hasta ahora. Una vez seleccionada la página se observará otra vez los valores límites de ella, pero esta vez se comparará solamente los valores de ADCL.

La idea es a menor sea el valor de ADCL más veces se desplazará a la izquierda el byte 00000001, quien será enviado por PORTC, de forma que el pixel encendido represente exactamente donde se haya pulsado en la pantalla. El bit menos significativo representará el pixel superior de la página, por lo que cuanto más alejado esté ADCL del valor máximo más abajo estará el pixel a encender.

Sabiendo que la diferencia entre el máximo y mínimo valor de la conversión ADC en cada página es de 85, y que se podrá desplazar hasta 7 veces el bit; se compara el máximo valor con el valor actual medido de ADCL, por cada 12 unidades de diferencia desplazará una vez, hasta superar el valor máximo.

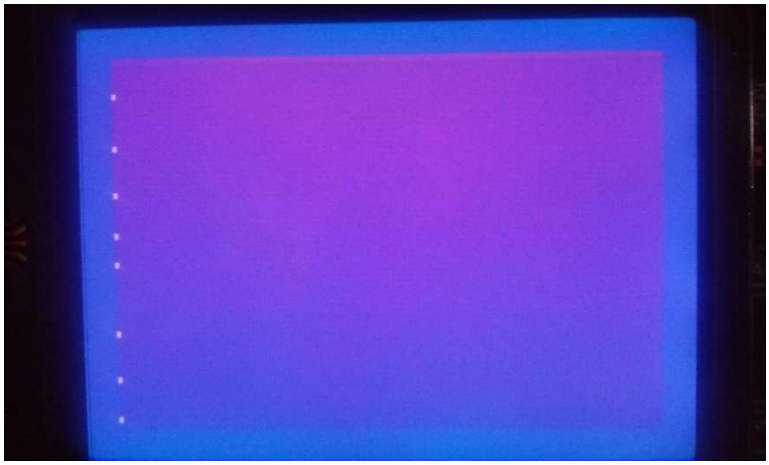


Figura 4.16 Segundo resultado, se enciende el pixel correspondiente

Se encontraron varios problemas a la hora de programar esta parte del código, el primero de ellos es que como solo se observa el valor de ADCL, se puede llegar el caso de que el valor máximo de la página es menor que el mínimo, dando a lugar a que el bit

no se desplace ninguna vez cuando si debería. Para solucionar esto se decidió para estos casos aumentar el valor hasta conseguir que ADCL mínimo alcance el valor 0, este mismo valor se le sumará al ADCL máximo y al medido en la conversión. De esta forma el ADCL máximo siempre será mayor o igual al medido. Por ejemplo para la página 0 tenemos los valores 55 y 226, se tuvo que sumar 30 para que el ADCL medido sea como mínimo 0 y como máximo 85, y mantener las 85 unidades de diferencia.

A pesar de ajustar estos valores no se conseguía encender siempre los pixeles apropiados, por lo que se tuvo que ir modificando valores de ADCL hasta que se consiguió optimizar la pantalla táctil.

Justo en el momento de pulsar la pantalla suelen encenderse pixeles erróneos, una vez que se mantiene pulsado se estabiliza y el resultado es el esperado. Esto puede ser debido a que no se le está dando a la pantalla táctil el tiempo suficiente para que suministre la tensión necesaria al microcontrolador, por lo que se decidió añadir un retardo después de cada conversión. Además, para asegurar que el valor convertido es el que deseamos se añadió otra conversión, donde a la finalización de la segunda se comparará los resultados de las dos. Si no son iguales, querrá decir que todavía no se ha conseguido la tensión deseada, por lo que se volverá a empezar el proceso. No se consigue solucionar al completo este problema sin embargo no será tan presente. Un efecto secundario es que al añadir retardos y otra conversión, se ralentizará un poco el proceso, por lo que dibujar en la pantalla no será tan fluido.

A estas alturas del código se decidió ir mejorándolo hasta conseguir un programa en el que se nos permita dibujar a placer y tener el primer minijuego de los dos planeados.

Un aspecto a mejorar era el hecho de que, como se ve en la figura 4.16, solo se dibuja un pixel en cada página. Esto es debido a que por PORTC solo se va activando un pixel y apagando los demás en cada página que se accede. La pantalla GLCD tiene una memoria RAM que guardará en todo momento el estado de todos los pixeles (si están apagadas o encendidas), recurso interesante que si se aprovecha bien se conseguirá mantener los pixeles encendidos a medida que se va recorriendo la pantalla.

Se creó una subrutina donde se pueda aprovechar esta memoria, que será muy parecida, por el uso del enable, a la de “Configuracion” y “Envio” ya mencionados en el presente proyecto. La primera diferencia es que en esta ocasión se mantendrá activa el bit 3 de PORTA, asociado a R/W, que activado indicará que vamos a leer de la memoria del GLCD. Como lo que se quiere es leer información, se deberá tener a PORTC como entrada.

Esta vez el enable se deberá activar y desactivar dos veces, la primera no hará ningún resultado aparente, de hecho en el datasheet lo nombran como “dummy read”, pero será totalmente necesario para completar el proceso. En el segundo encendido del enable se leerá el contenido de la memoria y se volverá a pagar. Antes de salir de la subrutina se establecerá de nuevo a PORTC como salida.

```
Leer:
    push r16
    push r18
    push r17

    ldi r16,0b00000000
    out DDRC, r16
    sbi PORTA,2
    sbi PORTA,3


    sbi PORTD,6
    rcall Retardo100us

    cbi PORTD,6
    rcall Retardo100us
    sbi PORTD,6
    rcall Retardo100us

    in r28, PINC
    cbi PORTD,6
    rcall Retardo100us
    ldi r16,255
    out DDRC, r16

    pop r17
    pop r18
    pop r16
    ret
```

Figura 4.17 Subrutina donde lee el estado de una página de una dirección Y específica

Antes de acceder a esta subrutina se debe especificar en qué página y dirección de Y queremos leer el valor del byte. Una vez leído, la idea es ejecutar un or del pixel a encender con los pixeles ya previamente encendidos en la misma página, de forma que se mantengan siempre los pixeles encendidos.

Un detalle que no se tuvo en cuenta y provocó numerosos problemas es que después de leer la memoria del GLCD, aumenta la dirección de Y en una unidad de manera automática, provocando que a la hora de intentar un or, se hacía con la información leida en la dirección X y se escribía en X + 1. Esto provocaba que la dirección X siempre estuviese apagada por lo que se arrastraba el mismo error donde solo se puede dibujar un pixel por página.

Una vez descubierto el error, la solución fue simplemente especificar de nuevo la dirección Y una vez leído la memoria GLCD.

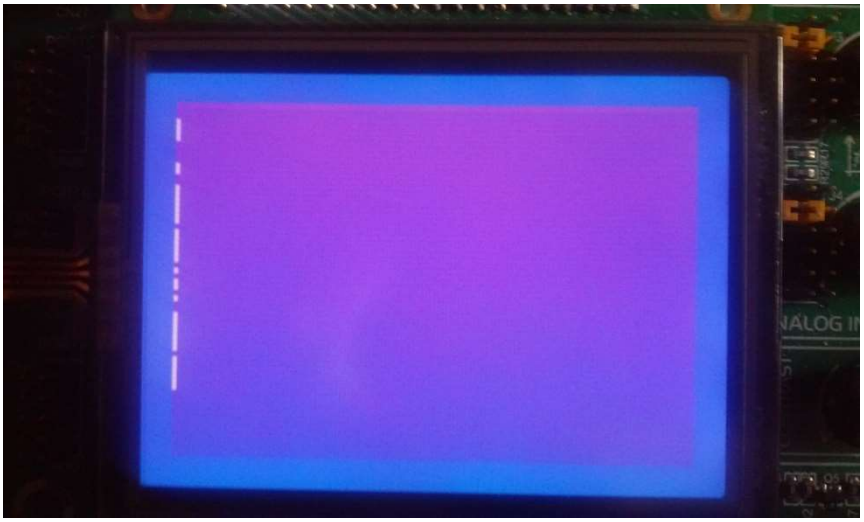


Figura 4.18 Tercer resultado, no se borra los pixeles

Hasta ahora la dirección Y no variaba, sino que se mantenía un valor fijo para controlar solamente la componente vertical de la pantalla. Para empezar a considerar la parte horizontal también, primero se activa el bit 2 de PORTA y se desactiva el bit 3 PORTA, y se configura el registro ADMUX para que el valor leído sea por el pin 0 de PORTA.

Se tiene que los valores de tensión serán de 0.32 a 4.07 cuyos resultados de conversión ADC, sabiendo que los 5V de referencia será el máximo, son 65 y 832 respectivamente.

Si se divide 832 unidades por la cantidad de pixeles que hay en la pantalla en una línea horizontal (128 pixeles) tenemos que por cada pixel equivaldrá aproximadamente a unas 6 unidades.

Sabiendo todo esto, la lógica del código fue, una vez se tiene la conversión ADC, aumentar el valor de un registro por cada 6 unidades de ADCL. Para cada valor de ADCH se sumarán al mismo registro unos 39, ya que cada unidad de ADCH equivale a 255 de ADCL. El valor de este registro será finalmente la dirección Y a la que se debe encender el pixel.

Se debe tener en cuenta determinados aspectos relacionados con el funcionamiento de la pantalla para que esta idea pueda funcionar. El primero es que si no se tiene en cuenta de que la conversión de la pantalla empieza a partir de los 0.32 voltios, supondrá un desfase de aproximadamente 11 pixeles a la derecha a la hora de introducir la dirección Y deseada. Se intentó primero descontar siempre justo antes de encender el pixel estas 11 unidades de dirección, resolviendo así en parte el problema. Sin embargo no se tuvo en cuenta que cuando se pulsaba por la parte izquierda de la pantalla, donde no están situados los pixeles, se sigue enviando un nivel de tensión y se produce igualmente la conversión. Al ser un valor pequeño y restar las 11 unidades daban números en negativo, que introduciéndolo como dirección la pantalla dejaba de funcionar correctamente. Esto obligó a crear un atajo para estos casos, donde se comprueba el valor de ADCL y ADCH. Si se confirma que se ha pulsado por la izquierda de la pantalla, se ahorra el proceso donde se va aumentando el valor del registro por cada 6 unidades y la reducción de 11 unidades. El registro no se verá aumentado por lo que finalmente se encenderá el primer pixel. Para el caso del borde de la derecha se usó el mismo procedimiento.

Otro aspecto importante es la división de la pantalla en dos controladores. Si no se tiene en cuenta, el contador que registra la dirección Y alcanzará valores mayores que 63, que son los pixeles máximos en cada pantalla. Para solucionarlo, antes de configurar la dirección se comprueba si no ha superado las 63 unidades, si es el caso se reducirá esta misma cantidad y se activará el lado derecho.

Con todo esto ya se tiene el minijuego de dibujo, el siguiente juego que se propuso es de saltar obstáculos.

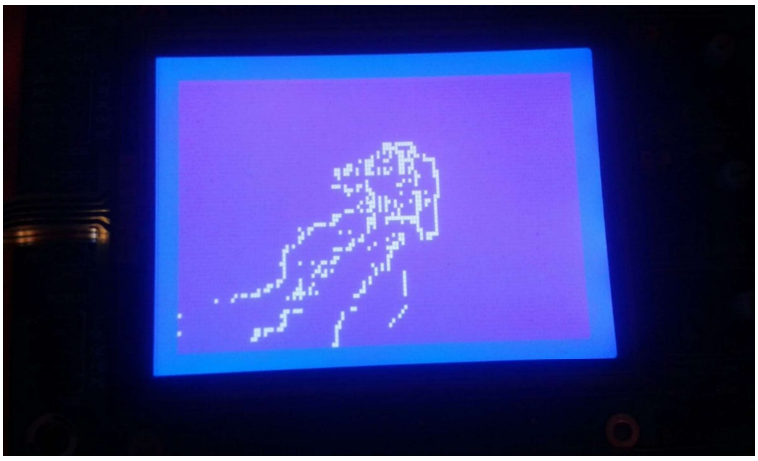


Figura 4.19 Con la parte horizontal y vertical programada se puede dibujar con libertad

4.4.5 JUEGO SALTOS

La idea de este juego es, a partir de una especie de bloque que está en movimiento, evadir obstáculos a partir de saltos. Estos saltos se activarán a través de un botón de la placa EASYV7.

Lo primero que se planteó es la puesta en escena, es decir, como sería el escenario, los obstáculos y el objeto en movimiento. Para el escenario simplemente se activaron todos los pixeles de la página 7, simulando el suelo. Los obstáculos serán finas barras, y el objeto móvil un simple cuadrado.

Para empezar se diseñó un código donde un obstáculo se moviese de izquierda a derecha, sin la posibilidad de interactuar con el objeto y que no haya ningún sistema de choque.

Para simular el movimiento del obstáculo su posición Y dependerá de un contador que se irá aumentando a medida que se recorre el bucle. El funcionamiento es el siguiente, una vez dibujado cada elemento (suelo, obstáculo y objeto) se mantendrá un retardo de poca duración y se apagarán todos los pixeles a partir de la subrutina ya comentada anteriormente. Una vez apagados, se volverán a dibujar los elementos, solo que al aumentar el contador el obstáculo estará dibujada en una posición superior en Y. Se tuvo cuidado en encender la pantalla adecuada para dibujar el obstáculo, si el contador superaba los 64 pixeles se tenía que comenzar de nuevo a 0 y usar otro registro para saber en todo momento en que pantalla se debe activar.

El siguiente paso fue añadir una mecánica de salto para el objeto. Al final del bucle se añadió una instrucción que lee el estado del BIT 7 de PINA, si está activa al registro R20 se le asignará el valor 10. En la subrutina asociada al dibujo del objeto, antes de dibujarlo se preguntará si este registro es mayor que 0, si es el caso se dibujará en una página inferior y se verá reducida en un valor, de forma que no pueda ser alcanzado con el obstáculo. De esta forma el objeto se mantendrá en alto durante 10 bucles para que pueda ser apreciado por el jugador, una vez que se haya reducido el registro R20 a 0 se dibujará en el suelo.

Se dibuja, cuando está en el suelo, en la página 6 encendiendo los 5 pixeles inferiores. Para que no haya contacto cuando salte, sabiendo que el obstáculo ocupa toda la página 6, se encenderán los mismos pixeles pero de la página 5.

Conseguida la mecánica de salto, el siguiente paso fue crear una interacción entre el obstáculo y el objeto. La idea principal es que cuando haya contacto entre estos el juego se pare automáticamente.

Se sabe que cuando R20 tenga como valor 0 el objeto estará en el suelo, además el objeto ocupará 6 pixeles, de la posición 7 a 12 de la pantalla de la derecha. En el momento de dibujar el obstáculo, si se cumple que R20 vale 0 y la dirección a dibujar este obstáculo sea en el intervalo de donde se encuentra el objeto, se considerará que existe contacto. Se comprobarán estas condiciones en la propia subrutina donde se dibuja el obstáculo, antes de dibujar la misma. Se podrá ver en la figura de abajo:

```
Impacto:
    sbrs r19,0
    rjmp FinObstaculo
    andi r16,0b00111111
    cpi r16,0b00000111
    brlo FinObstaculo
    cpi r16,0b00001101
    brsh FinObstaculo
    breq GameOver
```

Figura 4.20 Extracto del código donde se comprueba si hay impacto

Tras conseguir una forma de poder perder, se intentó complicar el juego añadiendo un segundo obstáculo, ya que era demasiado sencillo.

El segundo obstáculo se encontrará a 15 unidades situado a la izquierda del primero. Se deberá observar el valor del contador ya mencionado, quien tendrá como valor la dirección Y del obstáculo, este nuevo obstáculo solo podrá ser dibujado si el primero ya ha avanzado anteriormente 15 unidades y la manera de comprobarlo será midiendo el valor del contador. Sin embargo, a pesar de que este valor no supere las 15 unidades, puede existir la posibilidad de que sea porque está empezando por la segunda pantalla, por lo que el registro que comunica en que pantalla estará el primer obstáculo será importante en la decisión sobre si introducir el segundo o no. Si se cumple la condición, se volverá a llamar a la subrutina asociada al obstáculo.

Figura 4.21 Juego de saltos

Se añadió también una cantidad limitada de vidas y un contador de puntuación. Esta información estará visible en todo momento en la parte superior de la pantalla; se dibujó dos cuadrados en el Paint y a través del programa LCDAssistant y la subrutina que recorre tablas se consiguió fácilmente dibujarlo en la pantalla. Estos dos cuadrados serán dibujados antes de empezar con los otros elementos del juego, el de la izquierda señala cuántas vidas tiene el jugador y en la derecha la puntuación.

Para poder mostrar los números, se tuvo que diseñar cada uno de ellos a mano, señalando que pixeles mostrar para dar forma al número. Cada uno será llamado por una subrutina propia.

La cantidad de vidas estará registrado por un contador, que inicialmente tendrá 2. Antes de empezar el bucle, se observa su valor. Según el valor se llamará a la subrutina correspondiente al número y se introducirá dentro del cuadrado. Para introducirlo se tuvo que recurrir al ensayo y error, coger los valores exactos de la página y de la dirección Y para ajustarlo exactamente en el centro.

Para el caso de la puntuación será parecido, sin embargo para esta ocasión se determinará el valor a partir de dos registros. Esto es debido a que solo se ha diseñado números del 0 a 9, si el contador tuviera un número mayor no funcionaría correctamente. Un registro contará las unidades y el otro las decenas, se irá aumentando el valor de las unidades hasta que pase de 9 a 10, en ese momento se reseteará las unidades y las decenas se verá aumentada. Este método hará que se cuente la puntuación de 0 a 99, no se espera que se supere esa puntuación.

Mientras se probaba el minijuego, se observó que si se mantenía pulsado el botón de salto el objeto nunca llegaba a bajar. Esto ocurría porque justo al acabar el proceso de salto, al detectar de nuevo el botón saltaba de forma automática y da la sensación de que no ha tocado el suelo en ningún momento. Se añadió otro registro para obligar soltar el botón si se quiere volver a saltar. Una nueva condición para saltar será que este registro sea 0, una vez esté el objeto en el aire a este registro se le cambiará el valor. La única manera de que vuelva a ser 0 será dejar de pulsar el botón.

Figura 4.22 Estado final del juego

Terminado el código del juego, se diseñó un menú donde el jugador pudiese elegir cuál de ellos jugar. El primer paso fue dibujarlo en Paint y traspasarlo a la pantalla a partir del LCDAsistant. Una vez visualizado en la pantalla, se podía detectar cuales eran los niveles de tensión, tanto vertical como horizontal, que abarcaban los recuadros de las opciones a elegir.

Sabiendo los valores, se usó el mismo procedimiento que en la parte del dibujo en vertical. En vez de tener los límites de la página como referencia se usa estos márgenes para detectar cual ha sido la opción pulsada.

Capítulo 5: Presupuesto

Presupuesto

Para este capítulo se calculará los gastos totales que supone completar el proyecto en su totalidad. A parte de la compra de los componentes, se considerará al programador como mano de obra con un coste de 25€ la hora. A falta de conocer las horas exactas en las que se invertido en el proyecto, se estimará que serán de unos 180 horas aproximadamente, sin contar con los seminarios y tutorías.

Los precios de los componentes usados como referencia serán los encontrados en distintas páginas de internet, se harán una media para fijar un valor. Se considerará además de que la placa EasyAVR V7 llegará sin las pantallas necesarias para el proyecto, tanto el GLCD como LCD.

Elemento	Cantidad	Precio
Placa Easy AVR v7	1	150€
Módulo LCD 2x16	1	24€
Módulo GLCD 128x64	1	10€
Programador	1	4500€
Total		4684€

Capítulo 6: Conclusiones

Conclusiones

Los resultados del proyecto han sido los esperados, se ha conseguido hacer funcionar la pantalla GLCD de manera satisfactoria y se ha cumplido el objetivo de hacer funcionar varios juegos en él.

Este proyecto ha sido un constante aprendizaje tanto en el ámbito de la programación, como en el de la organización. Ha resultado vital saber distribuir la carga de trabajo de las asignaturas del cuarto curso con el trabajo fin de grado.

A pesar de que en la asignatura Informática Industrial se aprendió conceptos básicos de ensamblador, se ha tenido que adquirir nuevos conocimientos para conseguir llegar a los resultados.

Se ha descubierto la placa EasyAVR V7 como una herramienta altamente útil y accesible que facilitó el trabajo. Todas las conexiones y las configuraciones necesarias para los componentes de la placa estaban todo documentado en la guía de usuario.

Expected results were obtained; the GLCD screen operated satisfactorily and the goal of performing various games on the screen was achieved.

This Project has been a learning process both in the field of programming and in that of organization. Knowing how to manage and combine the fourth year class work load with the TFG has been vital.

Although in the subject industrial computing basic programming concepts were studied, it was necessary to acquire new knowledge in order to achieve the results.

The EasyAVR V7 was found to be an extremely useful and accessible tool which facilitated the work. All the connexions and necessary configurations for the components of the board where all well documented in the user’s guide.

Bibliografía

[1] Métodos de configuración – LCD
http://web.alfredstate.edu/weimandn/lcd/lcd_initialization/lcd_initialization_index.html

[2] Tutoriales AVR - http://www.avrbeginners.net/

[3] Tutoriales AVR - http://www.avr-asm-tutorial.net/

[4] Guía rápida del ensamblador de los microprocesadores ATMEL-AVR. Evelio J. González González, Grupo de Computadores y Control. Universidad de la Laguna

ANEXOS - Datasheets

Se mostrarán los datasheets de tres elementos, de la pantalla GLCD, LCD y del microcontrolador Atmega32. Al ser documentos de gran contenido, solo se aportarán al anexo los de gran importancia para el proyecto.

Features

- High-performance, Low-power Atmel®AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 32Kbytes of In-System Self-programmable Flash program memory
  - 1024Bytes EEPROM
  - 2Kbytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7V - 5.5V for ATmega32L
  - 4.5V - 5.5V for ATmega32
- Speed Grades
  - 0 - 8MHz for ATmega32L
  - 0 - 16MHz for ATmega32
- Power Consumption at 1MHz, 3V, 25°C
  - Active: 1.1mA
  - Idle Mode: 0.35mA
  - Power-down Mode: < 1µA

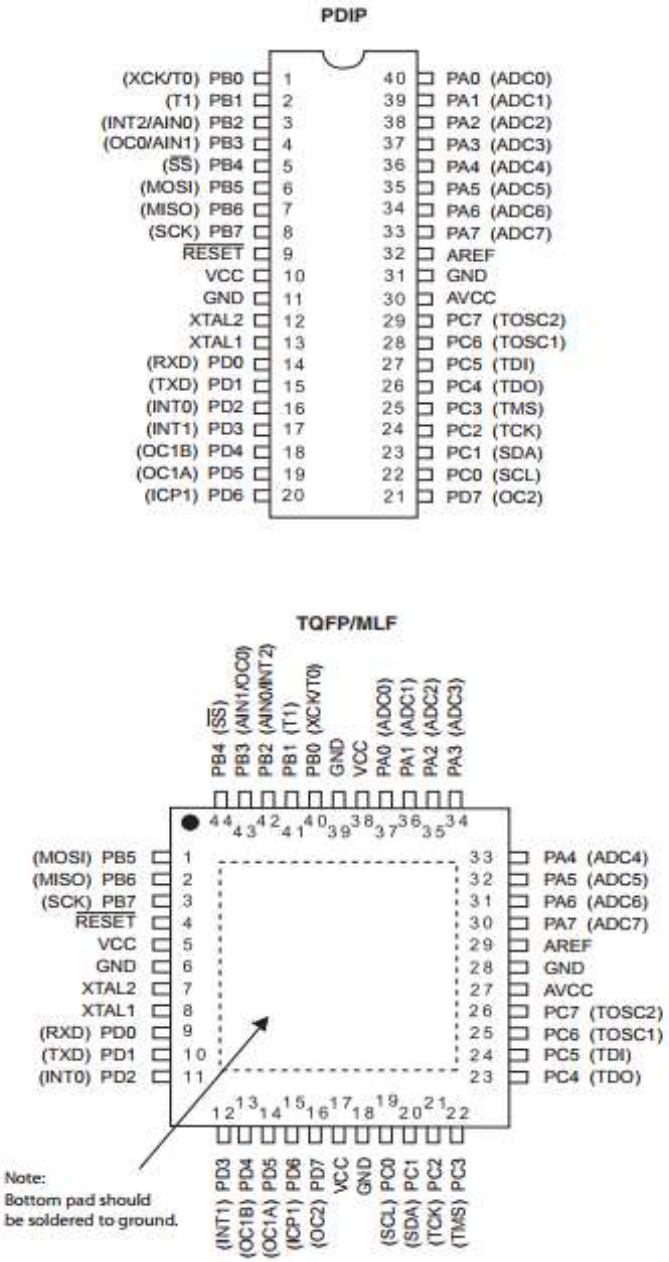
8-bit AVR® Microcontroller with 32KBytes In-System Programmable Flash

ATmega32
ATmega32L

2503Q–AVR–02/11

Pin Configurations

Figure 1. Pinout ATmega32



The Atmel®AVR®AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega32 provides the following features: 32Kbytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024bytes EEPROM, 2Kbyte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega32 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The Atmel AVR ATmega32 is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Pin Descriptions

VCC Digital supply voltage.

GND Ground.

Port A (PA7..PA0) Port A serves as the analog inputs to the A/D Converter.

Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the ATmega32 as listed on page 57.

Port C (PC7..PC0)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

The TD0 pin is tri-stated unless TAP states that shift out data are entered.

Port C also serves the functions of the JTAG interface and other special features of the ATmega32 as listed on page 60.

Port D (PD7..PD0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega32 as listed on page 62.

$\overline{\text{RESET}}$

Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 37. Shorter pulses are not guaranteed to generate a reset.

XTAL1

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting Oscillator amplifier.

AVCC

AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter.

AREF

AREF is the analog reference pin for the A/D Converter.

Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a half carry in some arithmetic operations. Half Carry is useful in BCD arithmetic. See the "Instruction Set Description" for detailed information.

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the "Instruction Set Description" for detailed information.

- **Bit 3 – V: Two's Complement Overflow Flag**

The Two's Complement Overflow Flag V supports two's complement arithmetics. See the "Instruction Set Description" for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

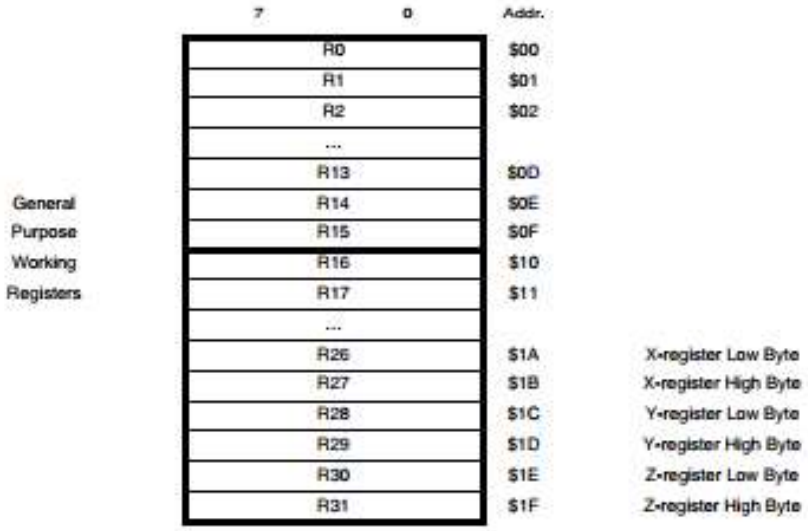
General Purpose Register File

The Register File is optimized for the Atmel®AVR® Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 4 shows the structure of the 32 general purpose working registers in the CPU.

Figure 4. AVR CPU General Purpose Working Registers



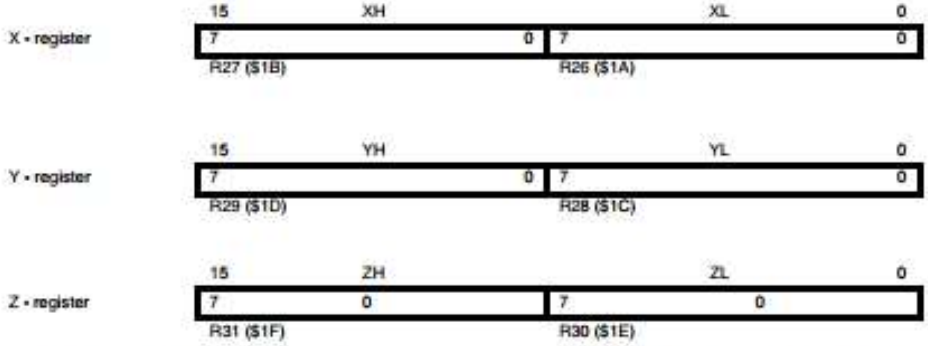
Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 4, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer Registers can be set to index any register in the file.

The X-register, Y-register and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space. The three indirect address registers X, Y, and Z are defined as described in Figure 5.

Figure 5. The X-, Y-, and Z-registers



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the Instruction Set Reference for details).

Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above $60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

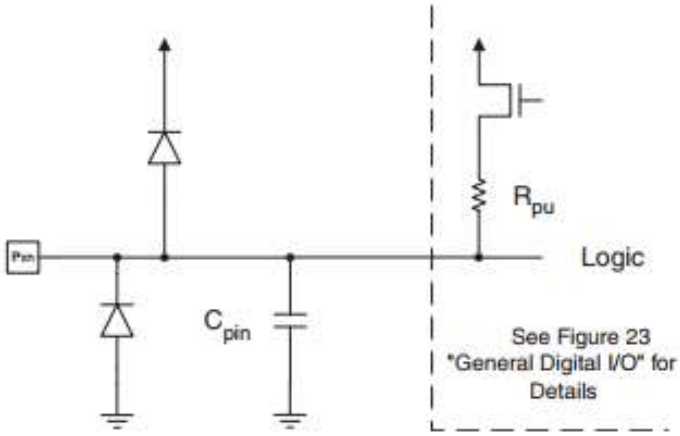
Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

I/O Ports

Introduction

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and Ground as indicated in Figure 22. Refer to "Electrical Characteristics" on page 287 for a complete list of parameters.

Figure 22. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case "x" represents the numbering letter for the port, and a lower case "n" represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. that is, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in "Register Description for I/O Ports" on page 64.

Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. In addition, the Pull-up Disable – PUD bit in SFIOR disables the pull-up function for all pins in all ports when set.

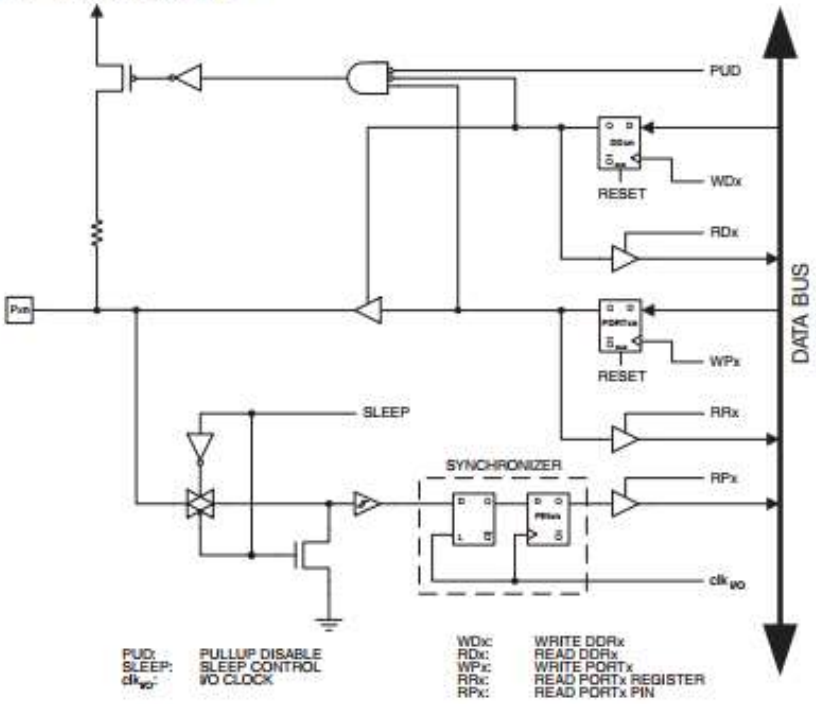
Using the I/O port as General Digital I/O is described in "Ports as General Digital I/O" on page 50. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in "Alternate Port Functions" on page 54. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 23 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 23. General Digital I/O[1]



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description for I/O Ports" on page 64, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when a reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled ({DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver

Table 33. Overriding Signals for Alternate Functions in PD3..PD0

Signal Name	PD3/INT1	PD2/INT0	PD1/TXD	PD0/RXD
PUOE	0	0	TXEN	RXEN
PUOV	0	0	0	PORTD0 • $\overline{PUD}$
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	0	0	TXEN	0
PVOV	0	0	TXD	0
DIEOE	INT1 ENABLE	INT0 ENABLE	0	0
DIEOV	1	1	0	0
DI	INT1 INPUT	INT0 INPUT	–	RXD
AIO	–	–	–	–

Register Description for I/O Ports

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port C Data Register – PORTC

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Data Direction Register – DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Input Pins Address – PINC

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Analog to Digital Converter

Features

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13 µs - 260 µs Conversion Time
- Up to 15 kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- 7 Differential Input Channels
- 2 Differential Input Channels with Optional Gain of 10x and 200x
- Optional Left adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 2.56V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

The ATmega32 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

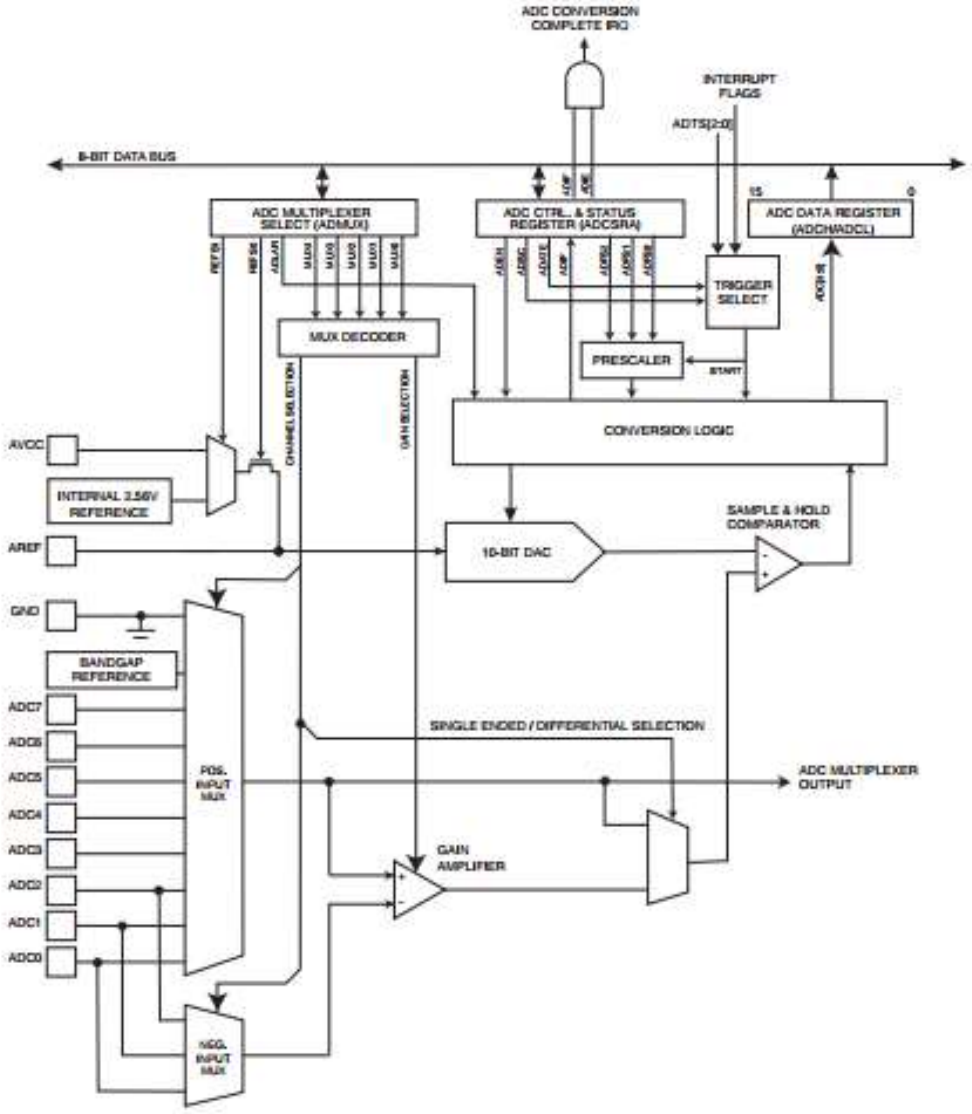
The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected.

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 98.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3V from V_{CC}. See the paragraph "ADC Noise Canceler" on page 208 on how to connect this pin.

Internal reference voltages of nominally 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

Figure 98. Analog to Digital Converter Block Schematic



Operation

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier.

If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor. This amplified value then

becomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

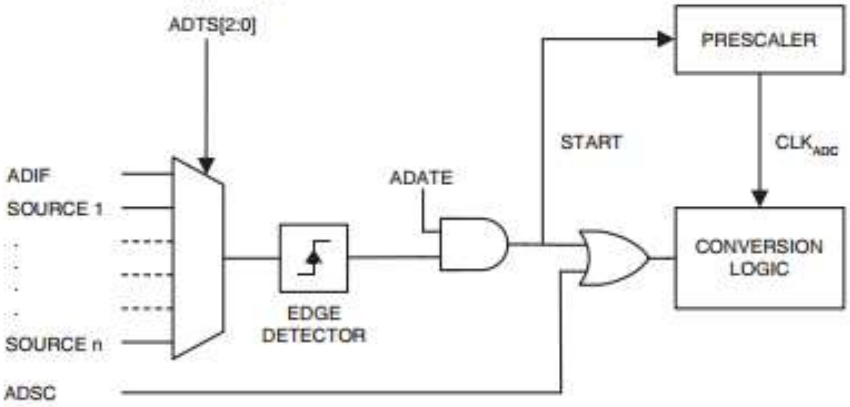
The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

Starting a Conversion

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR (see description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an Interrupt Flag will be set even if the specific interrupt is disabled or the global interrupt enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the Interrupt Flag must be cleared in order to trigger a new conversion at the next interrupt event.

Figure 99. ADC Auto Trigger Logic

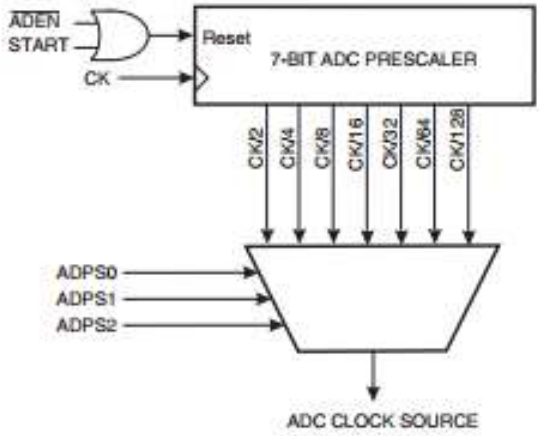


Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

Prescaling and Conversion Timing

Figure 100. ADC Prescaler



By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle. See "Differential Gain Channels" on page 206 for details on differential conversion timing.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of a first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In single conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place 2 ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

When using Differential mode, along with Auto Trigging from a source other than the ADC Conversion Complete, each conversion will require 25 ADC clocks. This is because the ADC must be disabled and re-enabled after every conversion.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 81.

Figure 101. ADC Timing Diagram, First Conversion (Single Conversion Mode)

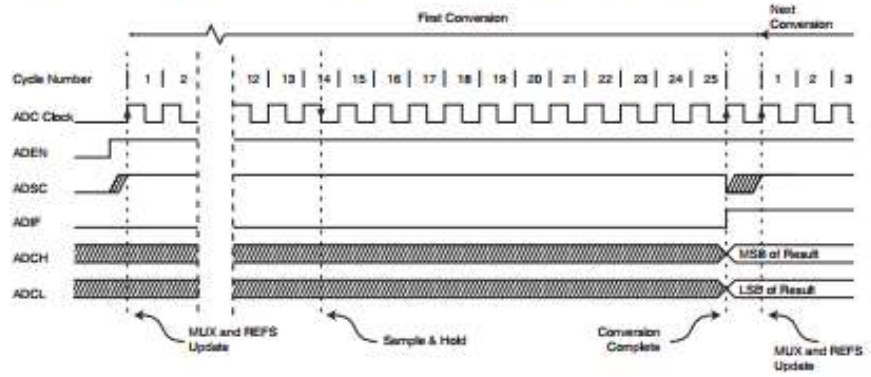


Figure 102. ADC Timing Diagram, Single Conversion

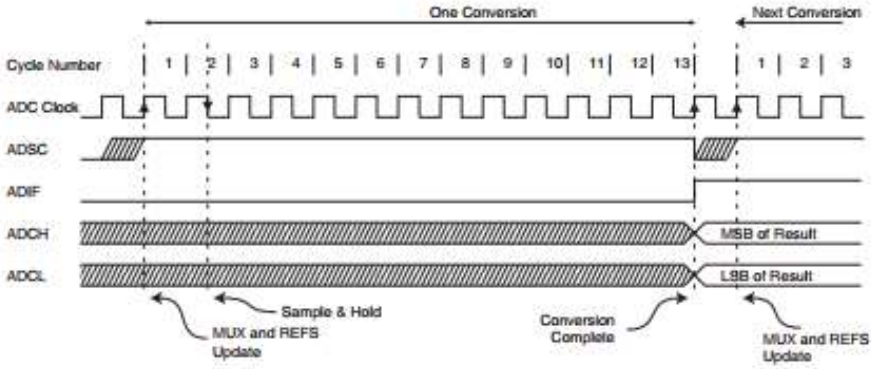


Figure 103. ADC Timing Diagram, Auto Triggered Conversion

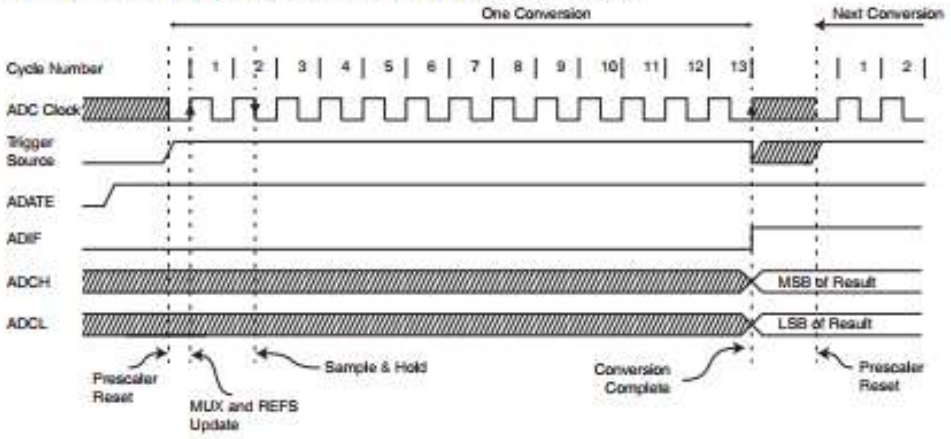


Figure 104. ADC Timing Diagram, Free Running Conversion

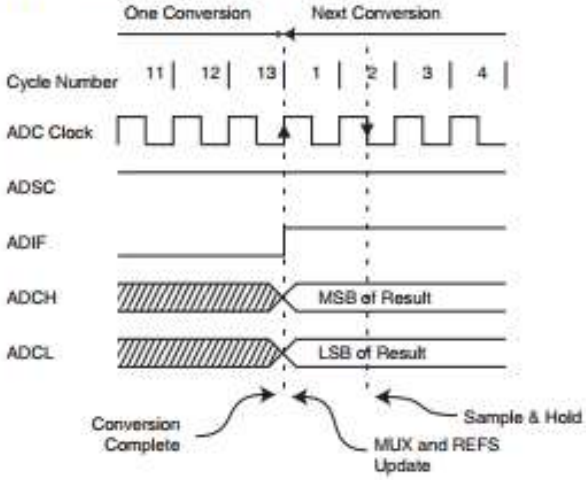


Table 81. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

Differential Gain Channels

When using differential gain channels, certain aspects of the conversion need to be taken into consideration.

Differential conversions are synchronized to the internal clock CK_{ADC2} equal to half the ADC clock. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of CK_{ADC2}. A conversion initiated by the user (that is,

all single conversions, and the first free running conversion) when CK_{ADC2} is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when CK_{ADC2} is high will take 14 ADC clock cycles due to the synchronization mechanism. In Free Running mode, a new conversion is initiated immediately after the previous conversion completes, and since CK_{ADC2} is high at this time, all automatically started (that is, all but the first) free running conversions will take 14 ADC clock cycles.

The gain stage is optimized for a bandwidth of 4kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. For example, the ADC clock period may be 6 µs, allowing a channel to be sampled at 12 kSPS, regardless of the bandwidth of this channel.

If differential gain channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing ADEN in ADCSRA to "0" then to "1"), only extended conversions are performed. The result from the extended conversions will be valid. See "Prescaling and Conversion Timing" on page 204 for timing details.

Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. After a conversion, before the Interrupt Flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the gain stage may take as much as 125 µs to stabilize to the new value. Thus conversions should not be started within the first 125 µs after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).

ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

ADC Voltage Reference

The reference voltage for the ADC (V_{REF}) indicates the conversion range for the ADC. Single ended channels that exceed V_{REF} will result in codes close to 0x3FF. V_{REF} can be selected as either AVCC, internal 2.56V reference, or external AREF pin.

AVCC is connected to the ADC through a passive switch. The internal 2.56V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. V_{REF} can also be measured at the AREF pin with a high impedant voltmeter. Note that V_{REF} is a high impedant source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between AVCC and 2.56V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

If differential channels are used, the selected reference should not be closer to AVCC than indicated in Table 121 on page 293.

ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

1. Make sure that the ADC is enabled and is not busy converting. Single Conversion Mode must be selected and the ADC conversion complete interrupt must be enabled.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption. If the ADC is enabled in such

sleep modes and the user wants to perform differential conversions, the user is advised to switch the ADC off and on after waking up from sleep to prompt an extended conversion to get a valid result.

Analog Input Circuitry

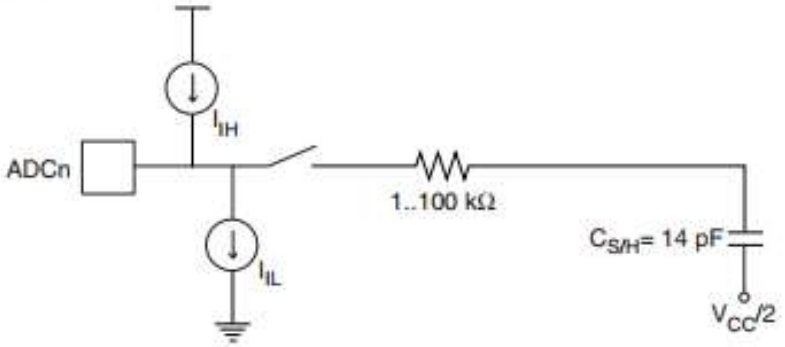
The Analog Input Circuitry for single ended channels is illustrated in Figure 105. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred kΩ or less is recommended.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 105. Analog Input Circuitry

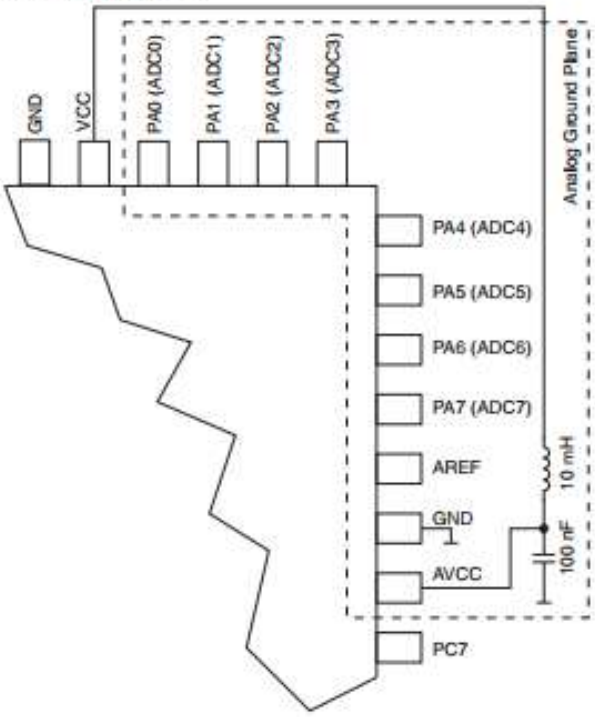


Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
2. The AVCC pin on the device should be connected to the digital V_{CC} supply voltage via an LC network as shown in Figure 106.
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

Figure 106. ADC Power Connections



Offset Compensation Schemes

The gain stage has a built-in offset cancellation circuitry that nulls the offset of differential measurements as much as possible. The remaining offset in the analog path can be measured directly by selecting the same channel for both differential inputs. This offset residue can be then subtracted in software from the measurement results. Using this kind of software based offset correction, offset on any channel can be reduced below one LSB.

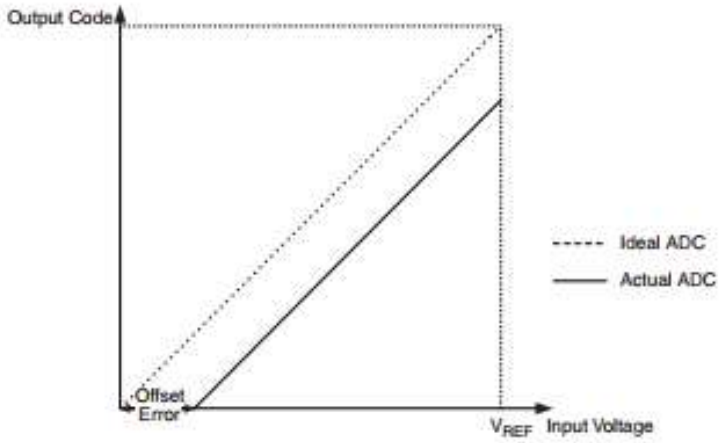
ADC Accuracy Definitions

An n-bit single-ended ADC converts a voltage linearly between GND and V_{REF} in 2^n steps (LSBs). The lowest code is read as 0, and the highest code is read as 2^n-1.

Several parameters describe the deviation from the ideal behavior:

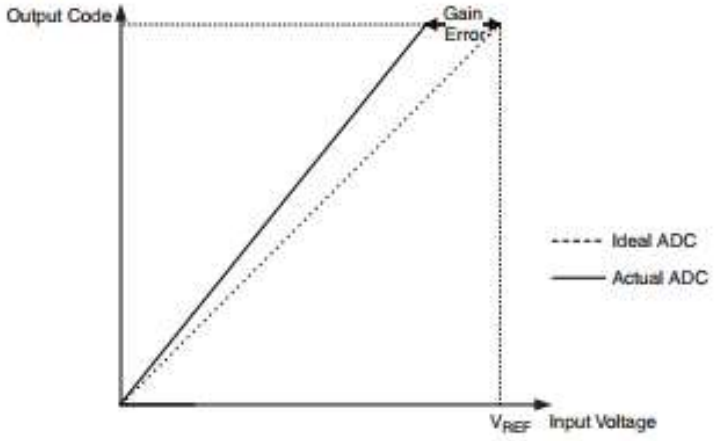
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

Figure 107. Offset Error



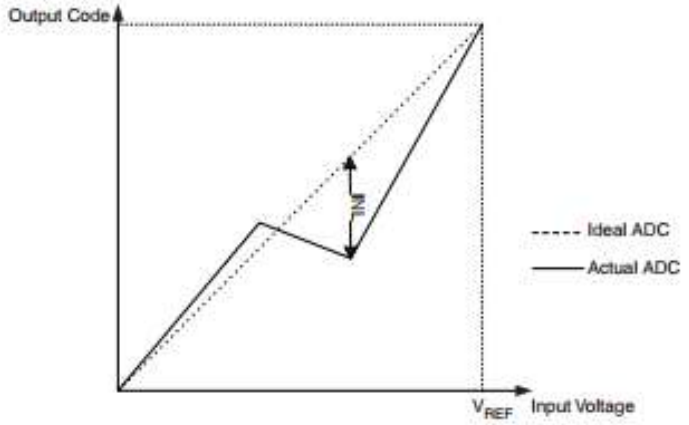
- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

Figure 108. Gain Error



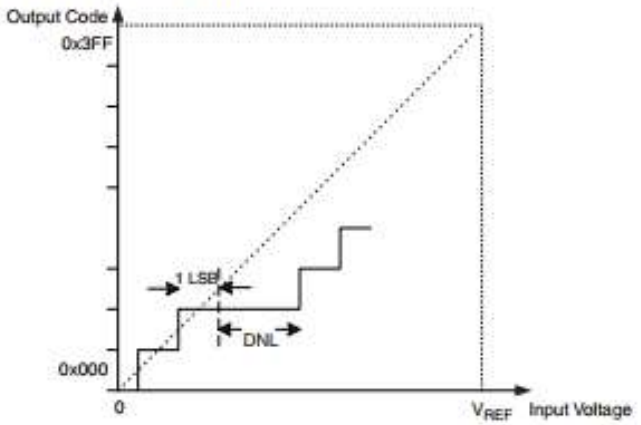
- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 109. Integral Non-linearity (INL)



- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

Figure 110. Differential Non-linearity (DNL)



- Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ±0.5 LSB.
- Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of Offset, Gain Error, Differential Error, Non-linearity, and Quantization Error. Ideal value: ±0.5 LSB.

ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference (see Table 83 on page 214 and Table 84 on page 215). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

If differential channels are used, the result is

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

where V_{POS} is the voltage on the positive input pin, V_{NEG} the voltage on the negative input pin, GAIN the selected gain factor, and V_{REF} the selected voltage reference. The result is presented in two's complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the results, it is sufficient to read the MSB of the result (ADC9 in ADCH). If this bit is one, the result is negative, and if this bit is zero, the result is positive. Figure 111 shows the decoding of the differential input range.

Table 82 shows the resulting output codes if the differential input channel pair (ADCn - ADCm) is selected with a gain of GAIN and a reference voltage of V_{REF}.

Figure 111. Differential Measurement Range

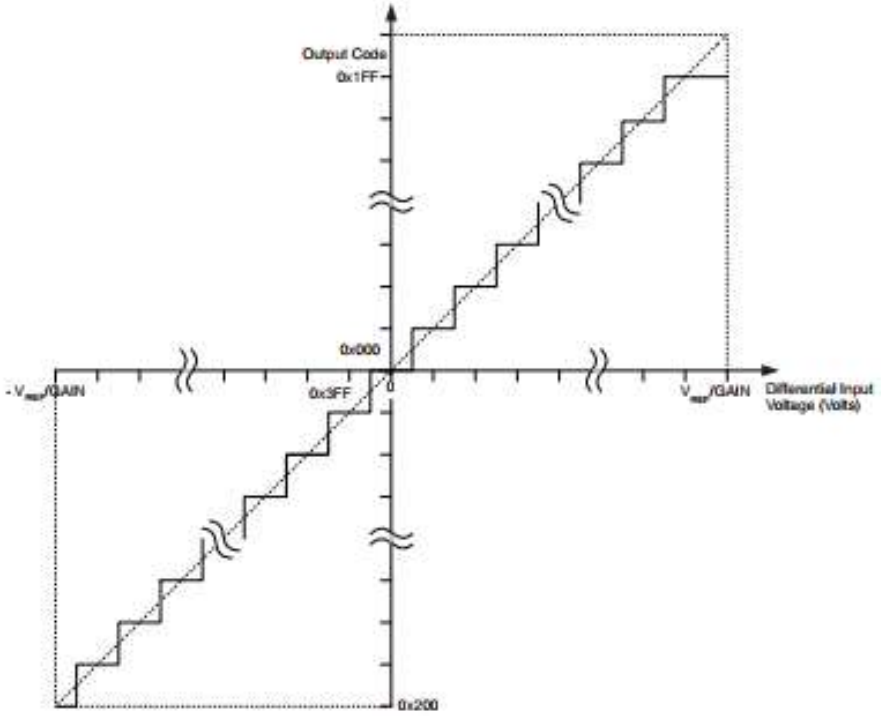


Table 82. Correlation between Input Voltage and Output Codes

V_{ADCn}	Read code	Corresponding Decimal Value
V_{ADCm} + V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 511/512 V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 510/512 V_{REF}/GAIN	0x1FE	510
...	...	...
V_{ADCm} + 1/512 V_{REF}/GAIN	0x001	1
V_{ADCm}	0x000	0
V_{ADCm} - 1/512 V_{REF}/GAIN	0x3FF	-1
...	...	...
V_{ADCm} - 511/512 V_{REF}/GAIN	0x201	-511
V_{ADCm} - V_{REF}/GAIN	0x200	-512

Example:

ADMUX = 0xED (ADC3 - ADC2, 10x gain, 2.56V reference, left adjusted result)

Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.

ADCR = 512 × 10 × (300 - 500) / 2560 = -400 = 0x270

ADCL will thus read 0x00, and ADCH will read 0x9C. Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 83. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 83. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conver-

sions. For a complete description of this bit, see "The ADC Data Register – ADCL and ADCH" on page 217.

- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 84 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 84. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	
10010		ADC2	ADC1	
10011		ADC3	ADC1	
10100		ADC4	ADC1	
10101		ADC5	ADC1	
10110		ADC6	ADC1	
10111		ADC7	ADC1	
11000		ADC0	ADC2	
11001		ADC1	ADC2	
11010		ADC2	ADC2	
11011		ADC3	ADC2	
11100		ADC4	ADC2	

Table 84. Input Channel and Gain Selections (Continued)

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
11101		ADC5	ADC2	1x
11110	1.22V (V_{BG})	N/A		
11111	0V (GND)			

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running Mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

Table 85. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The ADC Data Register – ADCL and ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two's complement form.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in "ADC Conversion Result" on page 213.

Special FunctionIO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:5 – ADTS2:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

Table 86. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

- **Bit 4 – Reserved Bit**

This bit is reserved for future use in the ATmega32. For ensuring compability with future devices, this bit must be written zero when SFIOR is written.

Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	10
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	12
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								82
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	47, 67
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	68
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	82, 112, 130
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	83, 112, 130
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	248
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	177
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	32, 66
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	40, 67, 228
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	80
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								82
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								30
	OCDR	On-Chip Debug Register								224
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	56,85,131,198,218
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	107
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	110
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								111
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								111
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								111
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								111
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								111
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								111
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								111
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								111
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	125
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								127
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								127
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	128
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	42
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				164
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	162
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	19
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								19
$1D ($3D)	EEDR	EEPROM Data Register								19
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	64
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	64
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	64
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	64
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	64
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	65
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	65
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	65
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	65
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	65
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	65
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	65
$0F ($2F)	SPDR	SPI Data Register								138
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	138
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	136
$0C ($2C)	UDR	USART I/O Data Register								159
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	160
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	161
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								164
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	199
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	214
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	216
$05 ($25)	ADCH	ADC Data Register High Byte								217
$04 ($24)	ADCL	ADC Data Register Low Byte								217
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								179
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	179

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	178
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								177

Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← Stack	None	4
RETI		Interrupt Return	PC ← Stack	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	Stack ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← Stack	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-Chip Debug Only	None	N/A

HD44780U (LCD-II)

(Dot Matrix Liquid Crystal Display Controller/Driver)

HITACHI

Description

The HD44780U dot-matrix liquid crystal display controller and driver LSI displays alphanumerics, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver.

A single HD44780U can display up to one 8-character line or two 8-character lines.

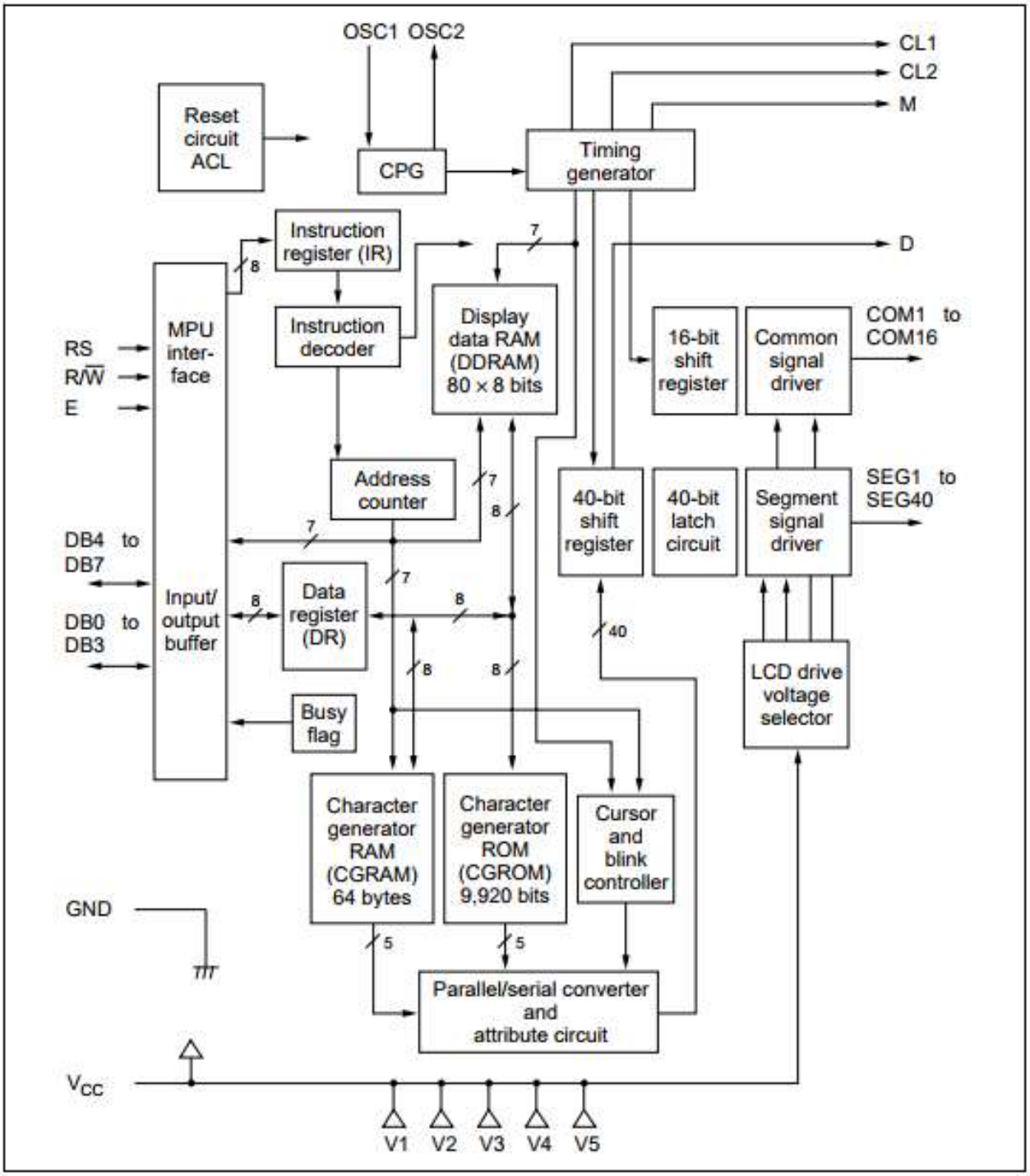
The HD44780U has pin function compatibility with the HD44780S which allows the user to easily replace an LCD-II with an HD44780U. The HD44780U character generator ROM is extended to generate 208 5 × 8 dot character fonts and 32 5 × 10 dot character fonts for a total of 240 different character fonts.

The low power supply (2.7V to 5.5V) of the HD44780U is suitable for any portable battery-driven product requiring low power dissipation.

Features

- 5 × 8 and 5 × 10 dot matrix possible
- Low power operation support:
  - 2.7 to 5.5V
- Wide range of liquid crystal display driver power
  - 3.0 to 11V
- Liquid crystal drive waveform
  - A (One line frequency AC waveform)
- Correspond to high speed MPU bus interface
  - 2 MHz (when V_{CC} = 5V)
- 4-bit or 8-bit MPU interface enabled
- 80 × 8-bit display RAM (80 characters max.)
- 9,920-bit character generator ROM for a total of 240 character fonts
  - 208 character fonts (5 × 8 dot)
  - 32 character fonts (5 × 10 dot)

HD44780U Block Diagram



Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.
CL1	1	O	Extension driver	Clock to latch serial data D sent to the extension driver
CL2	1	O	Extension driver	Clock to shift serial data D
M	1	O	Extension driver	Switch signal for converting the liquid crystal drive waveform to AC
D	1	O	Extension driver	Character pattern data corresponding to each segment signal
COM1 to COM16	16	O	LCD	Common signals that are not used are changed to non-selection waveforms. COM9 to COM16 are non-selection waveforms at 1/8 duty factor and COM12 to COM16 are non-selection waveforms at 1/11 duty factor.
SEG1 to SEG40	40	O	LCD	Segment signals
V1 to V5	5	—	Power supply	Power supply for LCD drive V_{cc} –V5 = 11 V (max)
V_{cc}, GND	2	—	Power supply	V_{cc}: 2.7V to 5.5V, GND: 0V
OSC1, OSC2	2	—	Oscillation resistor clock	When crystal oscillation is performed, a resistor must be connected externally. When the pin input is an external clock, it must be input to OSC1.

Function Description

Registers

The HD44780U has two 8-bit registers, an instruction register (IR) and a data register (DR).

The IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM) and character generator RAM (CGRAM). The IR can only be written from the MPU.

The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. Data written into the DR from the MPU is automatically written into DDRAM or CGRAM by an internal operation. The DR is also used for data storage when reading data from DDRAM or CGRAM. When address information is written into the IR, data is read and then stored into the DR from DDRAM or CGRAM by an internal operation. Data transfer between the MPU is then completed when the MPU reads the DR. After the read, data in DDRAM or CGRAM at the next address is sent to the DR for the next read from the MPU. By the register selector (RS) signal, these two registers can be selected (Table 1).

Busy Flag (BF)

When the busy flag is 1, the HD44780U is in the internal operation mode, and the next instruction will not be accepted. When RS = 0 and R/$\overline{W}$ = 1 (Table 1), the busy flag is output to DB7. The next instruction must be written after ensuring that the busy flag is 0.

Address Counter (AC)

The address counter (AC) assigns addresses to both DDRAM and CGRAM. When an address of an instruction is written into the IR, the address information is sent from the IR to the AC. Selection of either DDRAM or CGRAM is also determined concurrently by the instruction.

After writing into (reading from) DDRAM or CGRAM, the AC is automatically incremented by 1 (decremented by 1). The AC contents are then output to DB0 to DB6 when RS = 0 and R/$\overline{W}$ = 1 (Table 1).

Table 1 Register Selection

RS	R/$\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

Note: The user can specify any pattern for character-generator RAM.

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A02)

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

Reset Function

Initializing by Internal Reset Circuit

An internal reset circuit automatically initializes the HD44780U when the power is turned on. The following instructions are executed during the initialization. The busy flag (BF) is kept in the busy state until the initialization ends (BF = 1). The busy state lasts for 10 ms after V_{CC} rises to 4.5 V.

1. Display clear
2. Function set:
   DL = 1; 8-bit interface data
   N = 0; 1-line display
   F = 0; 5 × 8 dot character font
3. Display on/off control:
   D = 0; Display off
   C = 0; Cursor off
   B = 0; Blinking off
4. Entry mode set:
   I/D = 1; Increment by 1
   S = 0; No shift

Note: If the electrical characteristics conditions listed under the table Power Supply Conditions Using Internal Reset Circuit are not met, the internal reset circuit will not operate normally and will fail to initialize the HD44780U. For such a case, initial-ization must be performed by the MPU as explained in the section, Initializing by Instruction.

Instructions

Outline

Only the instruction register (IR) and the data register (DR) of the HD44780U can be controlled by the MPU. Before starting the internal operation of the HD44780U, control information is temporarily stored into these registers to allow interfacing with various MPUs, which operate at different speeds, or various peripheral control devices. The internal operation of the HD44780U is determined by signals sent from the MPU. These signals, which include register selection signal (RS), read/

write signal (R/$\overline{W}$), and the data bus (DB0 to DB7), make up the HD44780U instructions (Table 6). There are four categories of instructions that:

- Designate HD44780U functions, such as display format, data length, etc.
- Set internal RAM addresses
- Perform data transfer with internal RAM
- Perform miscellaneous functions

Normally, instructions that perform data transfer with internal RAM are used the most. However, auto-incrementation by 1 (or auto-decrementation by 1) of internal HD44780U RAM addresses after each data write can lighten the program load of the MPU. Since the display shift instruction (Table 11) can perform concurrently with display data write, the user can minimize system development time with maximum programming efficiency.

When an instruction is being executed for internal operation, no instruction other than the busy flag/address read instruction can be executed.

Because the busy flag is set to 1 while an instruction is being executed, check it to make sure it is 0 before sending another instruction from the MPU.

Note: Be sure the HD44780U is not in the busy state (BF = 0) before sending an instruction from the MPU to the HD44780U. If an instruction is sent without checking the busy flag, the time between the first instruction and next instruction will take much longer than the instruction time itself. Refer to Table 6 for the list of each instruc-tion execution time.

Table 6 Instructions

	Code											Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Instruction	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Table 6 Instructions (cont)

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

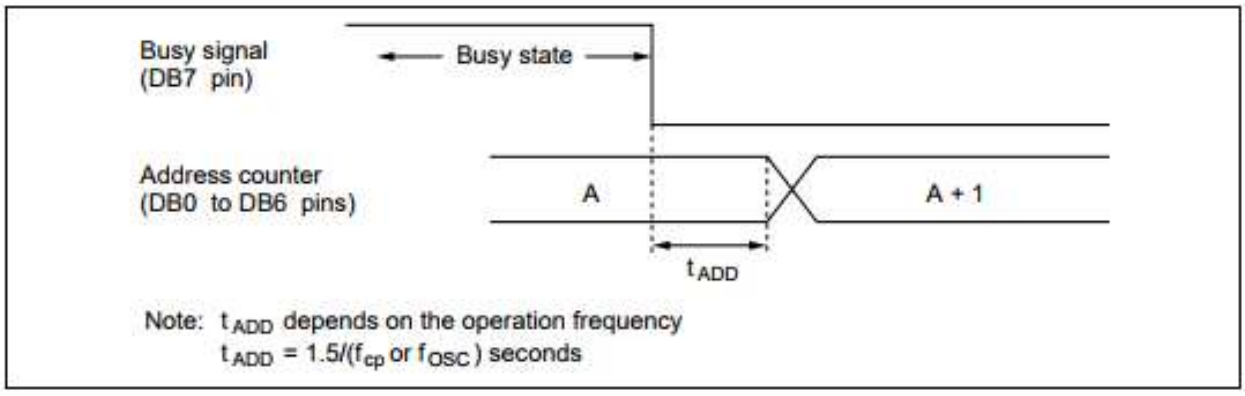


Figure 10 Address Counter Update

Instruction Description

Clear Display

Clear display writes space code 20H (character pattern for character code 20H must be a blank pattern) into all DDRAM addresses. It then sets DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. In other words, the display disappears and the cursor or blinking goes to the left edge of the display (in the first line if 2 lines are displayed). It also sets I/D to 1 (increment mode) in entry mode. S of entry mode does not change.

Return Home

Return home sets DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. The DDRAM contents do not change.

The cursor or blinking go to the left edge of the display (in the first line if 2 lines are displayed).

Entry Mode Set

I/D: Increments (I/D = 1) or decrements (I/D = 0) the DDRAM address by 1 when a character code is written into or read from DDRAM.

The cursor or blinking moves to the right when incremented by 1 and to the left when decremented by 1. The same applies to writing and reading of CGRAM.

S: Shifts the entire display either to the right (I/D = 0) or to the left (I/D = 1) when S is 1. The display does not shift if S is 0.

If S is 1, it will seem as if the cursor does not move but the display does. The display does not shift when reading from DDRAM. Also, writing into or reading out from CGRAM does not shift the display.

Display On/Off Control

D: The display is on when D is 1 and off when D is 0. When off, the display data remains in DDRAM, but can be displayed instantly by setting D to 1.

C: The cursor is displayed when C is 1 and not displayed when C is 0. Even if the cursor disappears, the function of I/D or other specifications will not change during display data write. The cursor is displayed using 5 dots in the 8th line for 5 × 8 dot character font selection and in the 11th line for the 5 × 10 dot character font selection (Figure 13).

B: The character indicated by the cursor blinks when B is 1 (Figure 13). The blinking is displayed as switching between all blank dots and displayed characters at a speed of 409.6-ms intervals when f_{cp} or f_{OSC} is 250 kHz. The cursor and blinking can be set to display simultaneously. (The blinking frequency changes according to f_{OSC} or the reciprocal of f_{cp}. For example, when f_{cp} is 270 kHz, 409.6 × 250/270 = 379.2 ms.)

Cursor or Display Shift

Cursor or display shift shifts the cursor position or display to the right or left without writing or reading display data (Table 7). This function is used to correct or search the display. In a 2-line display, the cursor moves to the second line when it passes the 40th digit of the first line. Note that the first and second line displays will shift at the same time.

When the displayed data is shifted repeatedly each line moves only horizontally. The second line display does not shift into the first line position.

The address counter (AC) contents will not change if the only action performed is a display shift.

Function Set

DL: Sets the interface data length. Data is sent or received in 8-bit lengths (DB7 to DB0) when DL is 1, and in 4-bit lengths (DB7 to DB4) when DL is 0.When 4-bit length is selected, data must be sent or received twice.

N: Sets the number of display lines.

F: Sets the character font.

Note: Perform the function at the head of the program before executing any instructions (except for the read busy flag and address instruction). From this point, the function set instruction cannot be executed unless the interface data length is changed.

Set CGRAM Address

Set CGRAM address sets the CGRAM address binary AAAAAA into the address counter.

Data is then written to or read from the MPU for CGRAM.

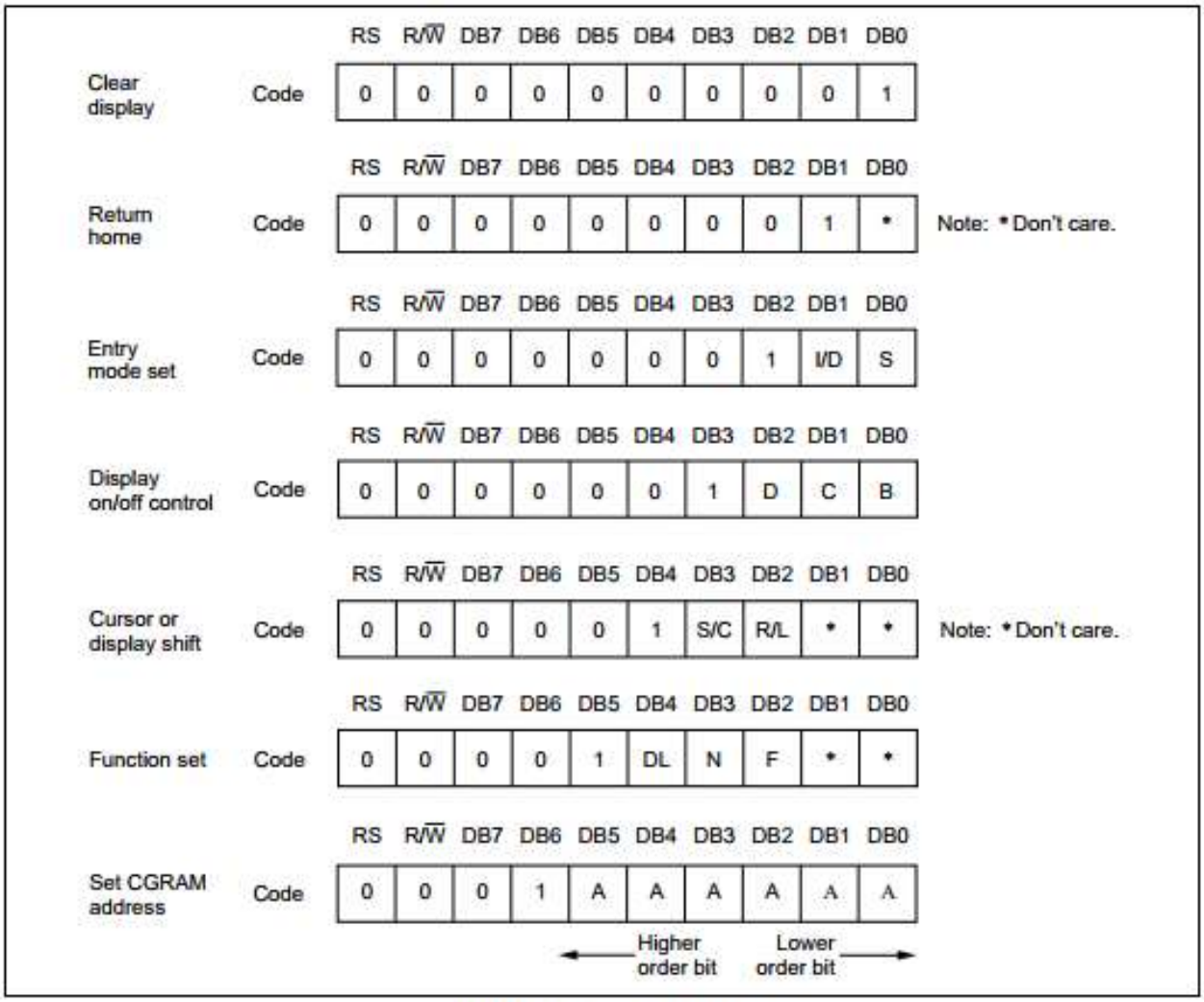


Figure 11 Instruction (1)

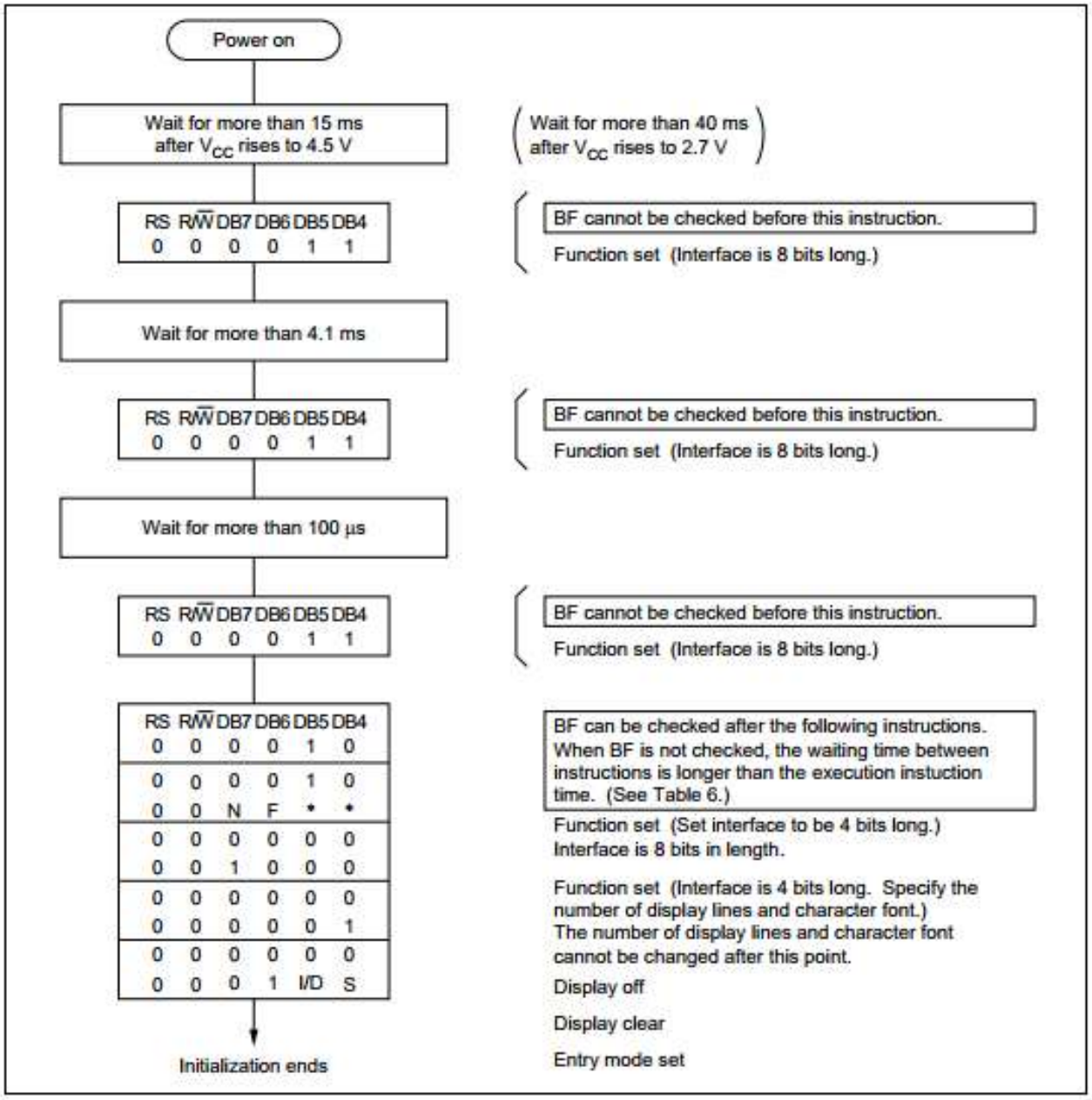


Figure 24 4-Bit Interface

WiRobot™

The Heart of Robot

MGL5128 128x64 Graphic LCD Module

User Manual

DrRobot®

Version: 1.0.0

January 2005

VI. Interface Description

Pin No.	Symbol	Level	Description
1	VDD	5.0V	Supply voltage for logic
2	V_{ss}	0V	Ground
3	V_o	(Variable)	Operating voltage for LCD
4	DB0	H/L	Data bit 0
5	DB1	H/L	Data bit 1
6	DB2	H/L	Data bit 2
7	DB3	H/L	Data bit 3
8	DB4	H/L	Data bit 4
9	DB5	H/L	Data bit 5
10	DB6	H/L	Data bit 6
11	DB7	H/L	Data bit 7
12	CS1	L	Select Column 1~ Column 64
13	CS2	L	Select Column 65~ Column 128
14	RST	L	Reset signal
15	R/W	H/L	H: Read (MPU•Module) , L: Write (MPU•Module)
16	D/I	H/L	H: Data , L : Instruction
17	E	H	Enable signal
18	Vee	—	Negative Voltage output
19	A	—	Power Supply for LED backlight (+)
20	K	—	Power Supply for LED backlight (-)

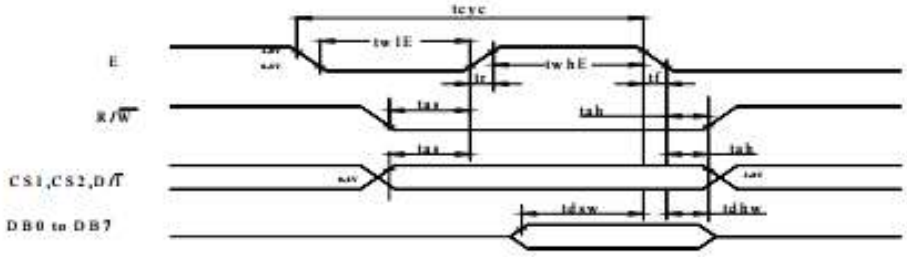
Dr Robot®
25 Valleywood Dr., Unit 20, Markham, ON, L3R 5L9 Canada Tel: (905) 943-9573 Fax: (905) 943-9187

VIII. Timing Characteristics

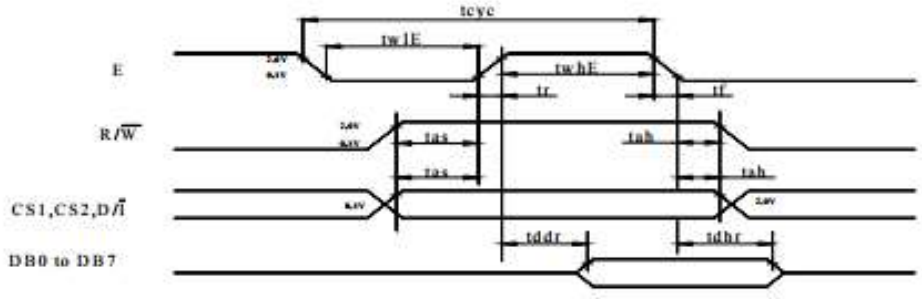
MPU Interface (T=25°, VDD=+5.0V±10.5)

Characteristic	Symbol	Min	Typ	Max	Unit
E cycle	tcyc	1000	—	—	ns
E high level width	twhE	450	—	—	ns
E low level width	twlE	450	—	—	ns
E rise time	tr	—	—	25	ns
E tall time	tf	—	—	25	ns
Address set-up time	tas	140	—	—	ns
Address hold time	tah	10	—	—	ns
Data set-up time	tdsw	200	—	—	ns
Data delay time	tddr	—	—	320	ns
Data hold time (write)	tdhw	10	—	—	ns
Data hold time (read)	tdhr	20	—	—	ns

MPU Read Timing



MPU Write Timing



Dr.Robot® www.DrRobot.com
25 Valleywood Dr., Unit 20, Markham, ON, L3R 5L9 Canada Tel: (905) 943-9572 Fax: (905) 943-9197

IX. Display Control Instruction

The display control instructions control the internal state of the KS0108B. Instruction is received from MPU to KS0108B for the display control. The following table shows various instructions

Instruction	D/I	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display ON/OFF	0	0	0	0	1	1	1	1	1	0/1	Controls the display on or off. Internal status and display RAM data are not affected. 0:OFF, 1:ON
Set Address	0	0	0	1	Y address (0~63)						Sets the Y address in the Y address counter.
Set Page (X address)	0	0	1	0	1	1	1	Page (0 ~7)			Sets the X address at the X address register.
Display Start Line	0	0	1	1	Display start line(0~63)						Indicates the display data RAM displayed at the top of the screen.
Status Read	0	1	BUSY	0	ON/OFF	RESET	0	0	0	0	Read status. BUSY 0:Ready 1:In operation ON/OFF 0:Display ON 1:Display OFF RESET 0:Normal 1:Reset
Write Display Data	1	0	Display Data								Writes data (DB0:7)into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read Display Data	1	1	Display Data								Reads data (DB0:7) from display data RAM to the data bus.

Dr Robot®
25 Valleywood Dr., Unit 20, Markham, ON, L3R 5L9 Canada Tel: (905) 943-9572 Fax: (905) 943-9197

X. Detailed Explanation

Display On/Off

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	1	1	1	D

The display data appears when D is and disappears when D is 0. Though the data is not on the screen with D = 0, it remains in the display data RAM. Therefore, you can make it appear by changing D = 0 into D = 1.

Display Start Line

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	1	A	A	A	A	A	A

Z address AAAAAA (binary) of the display data RAM is set in the display start line register and displayed at the top of the screen. Figure 2. shows examples of display (1/64 duty cycle) when the start line = 0-3. When the display duty cycle is 1/64 or more (ex. 1/32, 1/24 etc.), the data of total line number of LCD screen, from the line specified by display start line instruction, is displayed

Set Page (X Address)

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	1	1	1	A	A	A

X address AAA (binary) of the display data RAM is set in the X address register. After that, writing or reading to or from MPU is executed in this specified page until the next page is set. See Figure 1.

Set Y Address

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	A	A	A	A	A	A

Y address AAAAAA (binary) of the display data RAM is set in the Y address counter. After that, Y address counter is increased by 1 every time the data is written or read to or from MPU.

Status Read

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	Busy	0	On/Off	RESET	0	0	0	0

Dr.Robot www.DrRobot.com
33 Valleywood Dr., Unit 20, Markham, ON, L3R 5L9 Canada Tel: (905) 943-9573 Fax: (905) 943-9197

Busy

When busy is 1, the LSI is executing internal operations. No instruction are accepted while busy is 1, so you should make sure that busy is 0 before writing the next instruction.

ON/OFF

Shows the liquid crystal display condition: on condition or off condition.
When on/off is 1, the display is in off condition.

When on/off is 0, the display is in on condition

RESET

RESET = 1 shows that the system is being initialized. In this condition, no instructions except status read can be accepted.

RESET = 0 shows that initializing has system is in the usual operation condition.

Write Display Data

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D	D	D	D	D	D	D	D

Writes 8-bit data DDDDDDDD (binary) into the display data RAM. The Y address is increased by 1 automatically.

Read Display Data

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D	D	D	D	1	D	D	D

Reads out 8-bit data DDDDDDDD (binary) from the display data RAM. Then Y address is increased by 1 automatically.

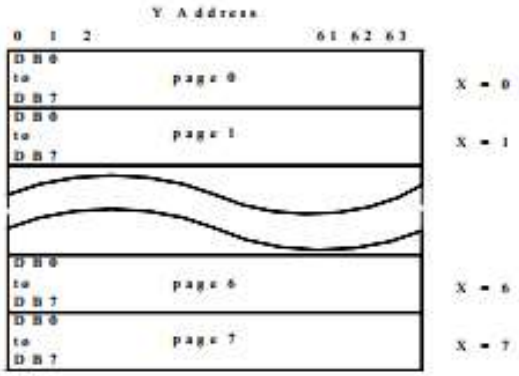
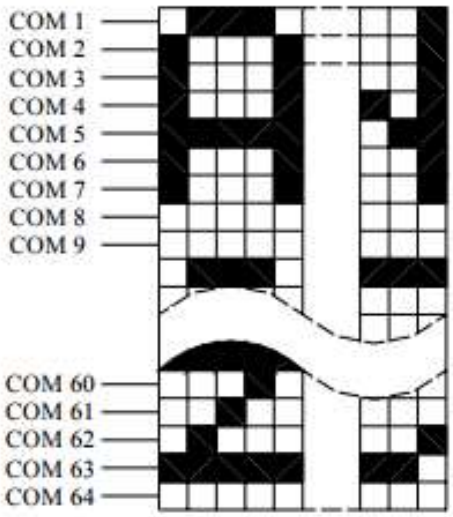


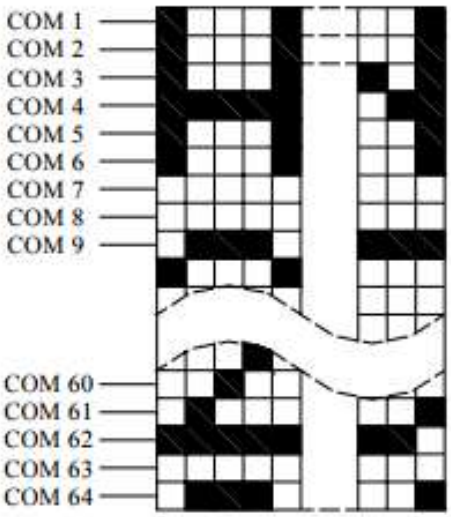
Figure 1.

Dr Robot®
www.DrRobot.com
25 Valleywood Dr., Unit 20, Markham, ON, L3R 5L9 Canada Tel: (905) 943-9572 Fax: (905) 943-9197

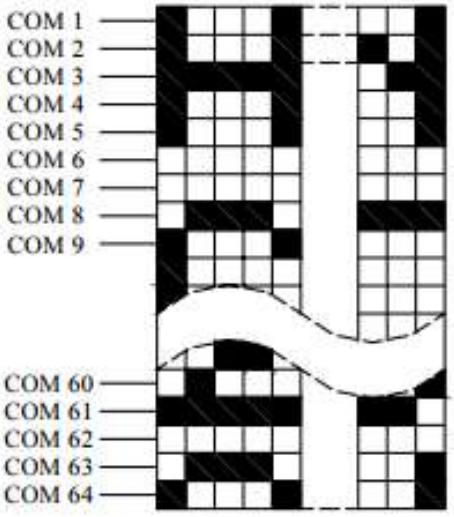
One dummy read is necessary right after the address setting. For details, refer to the explanation of output register in "Function of Each Block".



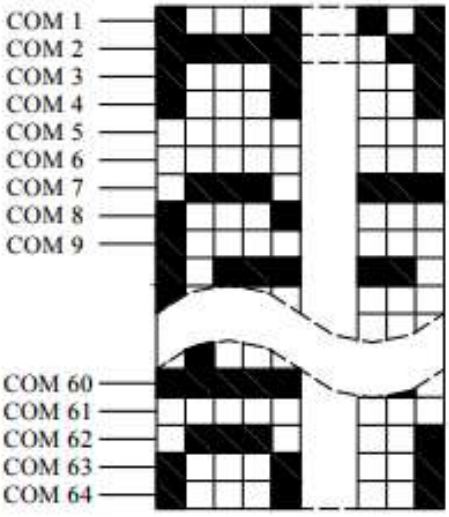
Start line = 0



Start line = 1



Start line = 3



Start line = 4

Dr Robot®
25 Valleywood Dr., Unit 20, Markham, ON, L3R 5L9 Canada Tel: (905) 943-8572 Fax: (905) 943-9197

ANEXOS - Códigos

LEDS

```
.include "m32def.inc"

.CSEG

.ORG $0000

        rjmp Start

Start:

        ldi r16,high(RAMEND)        ;Creación de la pila
        out SPH, r16
        ldi r16, low(RAMEND)
        out SPL, r16

        ldi  r16, 255               ;Se establece los PORT A y C como salida
        out DDRA, r16
        out DDRC, r16
        ldi r16,0b00001011          ;Activamos distintos bits para cada PORT
        out PORTA, r16
        ldi r16,0b00000011
        out PORTC, r16

bucle:

        rjmp bucle                  ;Bucle infinito
```

LCD

Código Principal:

```
.include "m32def.inc"


.CSEG

.ORG $0000
```

```
        rjmp Start


Start:

        ldi r16,high(RAMEND)            ;Creación de la pila
        out SPH, r16
        ldi r16, low(RAMEND)
        out SPL, r16

        ldi r16,0b11110000              ;Los cuatros últimos pines de PORTC serán de salida
        out DDRC, r16
        ldi r16, 0b01000000             ;El penúltimo pin de PORTD será de salida
        out DDRD, r16
        ldi r16, 0b00000100             ;El tercer pin de PORTA será de salida
        out DDRA, r16

        clr r17


ConfiguracionLCD:

        rcall Retardo100ms              ;Retardo de 100ms para que la pantalla pueda estabilizarse
        cbi PORTA,2                     ;Desactivando este bit entramos en modo configuración
        ldi r16, 0b00110000
        rcall Envio             ;Se tendrá que enviar el byte 00110000 varias veces con retardos
        rcall Retardo4ms                ;entre ellos para especificar que la pantalla es de 4 bits
        ldi r16, 0b00110000
        rcall Envio
        rcall Retardo100us
        ldi r16, 0b00110000
        rcall Envio
        rcall Retardo100us

        ldi r16, 0b00100000             ;A partir de aquí se enviarán distintas combinaciones de bytes
        rcall Envio                     ;al lcd para configurar a la pantalla LCD, en tandas de dos
        rcall Retardo100us

        ldi r16, 0b00100000
        rcall Envio
        ldi r16, 0b10000000
        rcall Envio
        rcall Retardo100us

        ldi r16, 0b00000000
        rcall Envio
        ldi r16, 0b10000000
        rcall Envio

        rcall Retardo100us

        ldi r16, 0b00000000
        rcall Envio
        ldi r16, 0b00010000
        rcall Envio
        rcall Retardo4ms

        ldi r16, 0b00000000
        rcall Envio
```

```
        ldi r16, 0b01100000
        rcall Envio
        rcall Retardo100us

        ldi r16, 0b00000000
        rcall Envio
        ldi r16, 0b11000000
        rcall Envio
        rcall Retardo100us

        sbi PORTA,2             ;Una vez configurado, activando este bit pondrá a la pantalla en
                                ;modo mensaje.
Mensaje:

        ldi r16,0b01000000      ;Se repetirá el procedimiento anterior, se enviarán bytes al
        rcall Envio             ;LCD en tanda de dos. En esta ocasión estos bytes contendrán
        ldi r16,0b10000000      ;la información a mostrar.
        rcall Envio
        rcall Retardo100us      ;Al final de esta parte la pantalla deberá mostrar “HOLA
                                ;MUNDO”.

        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b11110000
        rcall Envio
        rcall Retardo100us

        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b11000000
        rcall Envio
        rcall Retardo100us

        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b00010000
        rcall Envio
        rcall Retardo100us

        ldi r16,0b00100000
        rcall Envio
        ldi r16,0b00000000
        rcall Envio
        rcall Retardo100us

        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b11010000
        rcall Envio
        rcall Retardo100us

        ldi r16,0b01010000
        rcall Envio
        ldi r16,0b01010000

        rcall Envio
        rcall Retardo100us

        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b11100000
        rcall Envio
        rcall Retardo100us
```

```
        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b01000000
        rcall Envio
        rcall Retardo100us

        ldi r16,0b01000000
        rcall Envio
        ldi r16,0b11110000
        rcall Envio
        rcall Retardo100us

BucleInfinito:

        rjmp BucleInfinito                              ;Bucle infinito
```

Subrutinas:

```
Envio:    ;Subrutina donde se consigue enviar información al LCD

        push r16                ;Se guarda el valor del registro en la pila
        sbi PORTD, 6            ;Enable Activado
        out PORTC, r16          ;El valor del registro se envía a PORTC
        cbi PORTD, 6            ;Con Enable desactivado, se envía la información a la pantalla
        pop r16                 ;El registro recupera su valor original
        ret                     ;Fin de la subrutina
```

```
Retardo100ms:   ;Esta subrutina proporciona un retardo de 100 ms a partir de 3 bucles

        ldi r18, 10

bucle1:                 ;Primer bucle, a medida que se vaya reduciendo r18 se restablecerá el valor de r17

        dec r18
        ldi r17, 70
        brne bucle2 ;Si r18 llega a 0 se acaba el retardo, si no continua en bucle2.
        ret

bucle2:         ;Segundo bucle, en esta ocasión el registro a reducir será r17, restableciendo r16 cada vez

        ldi r16, 255
        subi r17, 0
        breq bucle1         ;Si r17 es 0 volvemos a bucle1 para reducir r18 y restablecer r17
        dec r17

bucle3:             ;Ultimo bucle, aquí se va reduciendo r16 hasta llegar a 0

        dec r16
        breq bucle2         ;Cuando llegue a 0 volvemos a bucle2 para reducir r17 y restablecer r16
        rjmp bucle3
```

```
;-----------------------------------------------------------------------------------------------------

Retardo4ms:         ;Esta subrutina proporciona un retardo de 4 ms a partir de 3 bucles

        ldi r18, 10

bucle14ms:      ;Primer bucle, a medida que se vaya reduciendo r18 se restablecerá el valor de r17

        dec r18
        ldi r17, 3
        brne bucle24ms      ;Si r18 llega a 0 se acaba el retardo, si no continua en segundo bucle.
        ret

bucle24ms:      ;Segundo bucle, en esta ocasión el registro a reducir será r17, restableciendo r16 cada vez

        ldi r16, 255
        subi r17, 0
        breq bucle14ms    ;Si r17 es 0 volvemos al primer bucle para reducir r18 y restablecer r17
        dec r17

bucle34ms:              ;Ultimo bucle, aquí se va reduciendo r16 hasta llegar a 0

        dec r16
        breq bucle24ms    ;Cuando llegue a 0 volvemos al segundo bucle para reducir r17 y restablecer r16
        rjmp bucle34ms

;-----------------------------------------------------------------------------------------------------

Retardo100us:       ;Esta subrutina proporciona un retardo de 4 ms a partir de 3 bucles

        ldi r18, 2

bucle1100us:    ;Primer bucle, a medida que se vaya reduciendo r18 se restablecerá el valor de r17

        dec r18
        ldi r17, 10
        brne bucle2100us    ;Si r18 llega a 0 se acaba el retardo, si no continua en segundo bucle.
        ret

bucle2100us:    ;Segundo bucle, en esta ocasión el registro a reducir será r17, restableciendo r16 cada vez

        ldi r16, 16
        subi r17, 0
        breq bucle1100us     ;Si r17 es 0 volvemos al primer bucle para reducir r18 y restablecer r17
        dec r17

bucle3100us:            ;Ultimo bucle, aquí se va reduciendo r16 hasta llegar a 0

        dec r16
        breq bucle2100us ;Cuando llegue a 0 volvemos al segundo bucle para reducir r17 y restablecer r16
        rjmp bucle3100us

;-----------------------------------------------------------------------------------------------------
```

GLCD

Código Principal:

```
.include "m32def.inc"


.CSEG

.ORG $0000


        rjmp Start

Start:

        ldi r16,high(RAMEND)                    ;Creación de la pila
        out SPH, r16
        ldi r16, low(RAMEND)
        out SPL, r16

        ldi     ZH,high(Seleccion*2)    ;El puntero tendrá la dirección de la tabla para el menú
        ldi     ZL,low(Seleccion*2)

        ldi r16, 0b01111100             ;Los 6 últimos pines de PORTA estarán configuradas
        out DDRA, r16                   ;como salida
        ldi r16,255
        out DDRB, r16                   ;Los PORT BCD estarán configuradas como salida
        out DDRC, r16
        out DDRD, r16


Inicializacion:

        sbi PORTD,7             ;Encendiendo este bit desactivamos el modo reset de la pantalla
        ldi r16, 0b00111111
        out PORTC, r16          ;Activamos la pantalla enviando a PORTC el byte 00111111
        rcall LadoIzquierdo     ;Se enciende el controlador izquierdo de la pantalla
        rcall Configuracion
        rcall LadoDerecho       ;Se enciende el controlador derecho de la pantalla
        rcall Configuracion

        rcall LadoIzquierdo     ;El controlador queda activado
        rcall Inicio ;Llamando a esta subrutina se consigue apuntar a la primera posición de la pantalla
        rcall Limpiar       ;A partir de esta posición, se va apagando todos los pixeles hasta el final
        rcall Menu              ;El menú es mostrado en la pantalla

ElegirJuego:

        ldi r16,0b01000001      ;El registro ADMUX quedará configurado de forma que la tensión de
        out admux,r16           ;referencia sea de 5V y donde la tensión a leer es en el pin 1 de PORTA
        cbi PORTA,2             ;Desactivando el bit 2 y activando el 3 la pantalla proporcionará
        sbi PORTA,3             ;tensiones según la parte vertical pulsada
        ldi r16, 0b11000000
        out adcsra, r16         ;Empieza la conversión
```

```
BucleLecturaVer:

        sbis adcsra,4                ;Cuando se active el bit la conversión habrá finalizado
        rjmp BucleLecturaVer

        rcall Retardo4ms

        in r16, adcl                 ;Se guarda el resultado de la conversión en dos registros
        in r17, adch

        mov r18,r16
        sbi adcsra,6

BucleLecturaVer2:                    ;Para evitar fallos se volverá ejecutar la conversión

        sbis adcsra,4
        rjmp BucleInfinito2

        rcall Retardo4ms

        in r16, adcl
        in r17, adch

        cp r16,r18                   ;Una vez leído los nuevos valores, se compararán con los anteriores
        brne ElegirJuego  ;Si los valores son iguales, se considerará que la conversión ha sido un éxito
        cpi r17,3      ;Se estudiará primero el valor ADCH, los dos más significativos de la conversión
        breq ElegirJuego  ;Mirando su valor, se podrá ver si se ha pulsado dentro de los limites verticales
        cpi r17,1             ;donde se encuentran las opciones del menú
        breq ElegirJuego2
        cpi r17,2
        breq DeteccionPulsacion

DeteccionPulsacion 2:;Si entramos en esta parte del código el valor de ADCH no será suficiente, se
                     ;estudiará ADCL
        cpi r16,235
        brsh ElegirJuego 2
        rjmp ElegirJuego

DeteccionPulsacion:      ;En esta ocasión ADCH tampoco será suficiente información

        cpi r16, 40
        brsh ElegirJuego

ElegirJuego2:        ;Si la pulsación ha sido dentro de los limites verticales, se estudiará también la
                     ;componente horizontal
        ldi r16,0b01000000    ;En esta ocasión se leerán los valores del pin 0 de PORTA
        out admux,r16
        sbi PORTA,2      ;Desactivando el bit 2 y activando el 3 la pantalla proporcionará
        cbi PORTA,3      ;tensiones según la parte vertical pulsada
        rcall Retardo4ms
        sbi adcsra,6     ;Empieza la conversión

BucleLecturaHor:

        sbis adcsra,4    ;Cuando se active el bit la conversión habrá finalizado
        rjmp BucleLecturaHor
        rcall LadoIzquierdo

        in r16, adcl     ;Se guarda el resultado de la conversión en dos registros
        in r17, adch
```

```
        cpi r17,3                   ;Como antes, el valor de ADCH no será suficiente información
        breq DetectorJuego          ;Dependiendo de su valor, se comparará ADCL con diferentes valores
        cpi r17,1           ;que nos permitirá saber en cual juego se ha pulsado o si se ha pulsado por fuera
        breq DetectorJuego2         ;de las dos opciones
        cpi r17,0
        breq DetectorJuego3
        rjmp Juego2

DetectorJuego3:

        cpi r16,92
        brsh Juego1
        rjmp ElegirJuego

DetectorJuego2:

        cpi r16, 180
        brlo Juego1
        cpi r16,215
        brsh Juego2
        rjmp ElegirJuego

DetectorJuego:

        cpi r16,41
        brsh ElegirJuego
        rjmp Juego2

Juego1:         ;Se accederá a esta parte del código si se pulsa la primera opción del menú, la del dibujo

        cbi PORTB,0     ;Se activa los dos controladores
        cbi PORTB,1
        rcall Inicio        ;Se restablecerá la pantalla antes de entrar en el juego
        rcall Limpiar
        ldi r20,9
        rjmp JuegoDibujo

Juego2:         ;Se accederá a esta parte del código si se pulsa la primera opción del menú, la de saltos

        cbi PORTB,0;    ;Se activa los dos controladores
        cbi PORTB,1

        rcall Inicio
        rcall Limpiar       ;Se restablecerá la pantalla antes de entrar en el juego

        clr r17
        clr r18
        clr r19
        clr r20
        clr r21
        clr r23
        ldi r22,49
        ldi r24,2
        ldi r29,2
        ldi r28,0
        clr r25
        ldi r26,31          ;Se inicializarán registros para el buen funcionamiento del juego
```

```
DatosSalto:
        ldi ZH,high(MenuSalto*2)    ;El puntero tendrá la dirección de la tabla con la información
        ldi ZL,low(MenuSalto*2)     ;para mostrar los recuadros donde se insertarán el recuento de
                                    ;vidas y saltos
        rcall Menu                  ;Se dibujaran estos elementos a partir de la subrutina
        rcall LadoIzquierdo         ;Se activa el controlador izquierdo

        ldi r16,0b10111011          ;Se apuntará dentro del recuadro del recuento de vidas especificando
        out PORTC, r16              ;las coordenadas X e Y
        rcall Configuracion
        ldi r16,0b01011011
        out PORTC, r16
        rcall Configuracion

        rcall Vidas                 ;Se llama la subrutina para dibujar la cantidad de vidas disponibles
        rcall LadoDerecho           ;Se activa el controlador izquierdo
        ldi r16,0b10111011
        out PORTC, r16              ;Se apuntará dentro del recuadro del recuento de saltos especificando
        rcall Configuracion         ;las coordenadas X e Y
        ldi r16,0b01010000
        out PORTC, r16
        rcall Configuracion

        rcall Saltos        ;Se llama la subrutina para dibujar las decenas de la cantidad de saltos realizados
        rcall Espacio
        rcall Saltos2       ;Se dibuja las unidades, con un espacio previo

        cbi PORTB,0         ;Se activa los dos controladores
        cbi PORTB,1
        ldi r16,0b10111111          ;Se apunta a la última página de la pantalla
        out PORTC, r16
        rcall Configuracion
        ldi r16,0b11111111
DibujandoSuelo:                     ;En este bucle se encenderán todos los pixeles de la última página

        out PORTC,r16
        rcall Envio
        inc r17
        sbrs r17,7                  ;Cuando se enciendan los pixeles saldrá del bucle
        rjmp DibujandoSuelo
        mov r21,r18 ;Se guardará el valor de la posición a dibujar los obstáculos en otro registro
        rcall Obstaculo             ;Se dibuja el primer obstáculo
        cpi r18,15
        brlo Probable2Obstaculo
        subi r18,15
        rcall Obstaculo             ;Si han pasado 15 unidades desde el primer obstáculo se dibujará el
                                    ;segundo obstáculo
DibujarCuadrado:

        rcall LadoDerecho           ;Concluido con los obstáculos se procede a dibujar el objeto
        rcall Cuadrado
        mov r18,r21                 ;Se recupera el valor del contador
        rjmp FinalizaciónSaltos
```

```
Probable2Obstaculo:

        sbrs r19,0          ;Si no han pasado 15 unidades lo siguiente a dibujar es el objeto.
        rjmp DibujarCuadrado
        add r18,r22
        clr r19
        rcall Obstaculo     ;En el caso de que no hayan pasado 15 unidades pero porque esté en la pantalla
        ser r19             ;de la derecha se dibujará el segundo obstáculo
        rjmp DibujarCuadrado

FinalizacionSaltos:         ;Si se activa el pin 7 de PORTA y los registros 20 y 23 están desactivados se
        cpi r20,0           ;le otorgará el valor 10 al registro 20. Darle este valor conseguirá hacer
        brne FinalizacionSaltos2    ;funcionar la mecánica de salto.
        sbis PINA,7
        rjmp FinalizacionSaltos2
        sbrc r23,0  ;El R23 nos obligará a soltar el botón de salto si queremos volver a saltar en el futuro
        rjmp FinalizacionSaltos2
        ser r23
        ldi r20,10

FinalizacionSaltos2:

        rcall RetardoVariable  ;Se usará un retardo para que el movimiento no sea excesivamente rápido
        sbis PINA,7                 ;Si soltamos el botón de salto R23 se desactiva
        clr r23
        clr r17
        inc r18             ;Se incrementa la dirección Y donde se dibujará el primer obstáculo
        sbrs r18,6          ;Si esta dirección llega a 64 habrá que cambiar el controlador de la pantalla
        rjmp DatosSalto
        clr r18                     ;Se resetea el controlador
        cpi r19,0                   ;El valor R19 estará relacionada con el controlador activado
        breq CambioPantalla         ;Si es 0 estará encendido el controlador izquierdo, y con 1 el derecho
        ldi r19,0
        inc r28    ;R28 estará asociada a las unidades del contador de salto, irá incrementándose
        cpi r26,1           ;R26 estará asociado con la velocidad del juego, a menor valor mayor velocidad
        breq FinalizacionSaltos3
        subi r26,2          ;Se irá reduciendo su valor hasta que tenga 1

FinalizacionSaltos3:

        sbrs r28,3          ;Si R28 llega al valor 10, se reseteará su valor y se aumentará el valor del registro
        rjmp DatosSalto  ;R25, asociado al valor de decenas del contador de saltos
        sbrs r28,1
        rjmp DatosSalto
        clr r28
        inc r25
        rjmp DatosSalto

CambioPantalla:             ;Si el contador indica que estamos en la pantalla de la derecha r19 se activa

        ldi r19,1
        rjmp DatosSalto
```

```
JuegoDibujo:

    ldi r16,0b01000001      ;El registro ADMUX quedará configurado de forma que la tensión
    out admux,r16           ;referencia sea de 5V y donde la tensión a leer es en el pin 1 de PORTA
    cbi PORTA,2             ;Desactivando el bit 2 y activando el 3 la pantalla proporcionará
    sbi PORTA,3             ;tensiones según la parte vertical pulsada
    ldi r16, 0b11000000
    out adcsra, r16         ;Empieza la conversión

LecturaVer:

    sbis adcsra,4               ;Cuando se active el bit la conversión habrá finalizado
    rjmp LecturaVer
    rcall Retardo4ms

    in r16, adcl                ;Se guarda el resultado de la conversión en dos registros
    in r17, adch

    mov r18,r16
    sbi adcsra,6                ;Para evitar fallos se volverá ejecutar la conversión
    rcall Retardo4ms            ;El retardo evitará fallos en la conversión


LecturaVer2:

    sbis adcsra,4               ;Cuando se active el bit la conversión habrá finalizado
    rjmp LecturaVer2

    in r16, adcl                ;Se guarda el resultado de la conversión en dos registros
    in r17, adch

    cp r16,r18          ;Una vez leído los nuevos valores, se compararán con los anteriores
    brne JuegoDibujo ;Si los valores son iguales, se considerará que la conversión ha sido un éxito
    ldi r18,0b00000001
    cpi r17,3       ;Si r17(ADCH) alcanza su máximo valor, es porque se ha pulsado la página 0
    breq Pagina0
    cpi r17,2           ;Para los otros 3 posibles valores, la página pulsada dependerá de
    breq DetectorPagina     ;distintos R16(ADCL)
    cpi r17,1
    breq DetectorPagina2

DetectorPagina 3:                       ;Si ADCH es 0 y…

    cpi r16, 143        ; ADCL es menor que 143 no se habrá pulsado la pantalla
    brlo JuegoDibujo
    cpi r16,228
    brsh Pagina6        ;ADCL es mayor que 228 se habrá pulsado la página 6
    rjmp Pagina7            ;ADCL es menor que 228 se habrá pulsado la página 7

DetectorPagina 2:                               ;Si ADCH es 1 y….

    cpi r16,227
    brsh Pagina3            ;ADCL es mayor que 228 se habrá pulsado la página 3
    cpi r16,142
    brsh Pagina4        ;ADCL es mayor que 142 se habrá pulsado la página 4
    cpi r16,57
```

```
        brsh Pagina5            ;ADCL es mayor que 57 se habrá pulsado la página 5
        rjmp Pagina6    ;ADCL es menor que 57 se habrá pulsado la página 6

DetectorPagina:                                 ;Si ADCH es 2 y….

        cpi r16, 226
        brsh Pagina0            ;ADCL es mayor que 226 se habrá pulsado la página 0
        cpi r16,141
        brsh Pagina1    ;ADCL es mayor que 141 se habrá pulsado la página 1
        cpi r16,56
        brsh Pagina2            ;ADCL es mayor que 56 se habrá pulsado la página 2
        rjmp Pagina3    ;ADCL es menor que 56 se habrá pulsado la página 3

DetectorDePixel:  ;En este bucle se irá comparando R16 y R25, cuando el primero sea mayor se saldrá de
                  ;ella. A medida que R16 se incrementa se desplaza un bit de R18 a la izquierda
        cp r16,r25
        brsh AjusteHor
        lsl r18     ;Desplazamiento a la izquierda de R18, originalmente tendrá como valor 00000001
        add r16,r20
        rjmp DetectorDePixel

Pagina0:                        ;Una vez se sabe la página pulsada se enciende el pixel exacto

        ldi r17,0b00000000      ;R17 tendrá el valor de la dirección X (página)
        ldi r19,30
        add r16,r19         ;Se aumenta R16 para que llegue como mínimo al valor 0 y sea menor que r25
        ldi r25, 85             ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel

Pagina1:

        ldi r17,0b00000001      ;R17 tendrá el valor de la dirección X (página)
        ldi r25, 220            ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel

Pagina2:

        ldi r17,0b00000010      ;R17 tendrá el valor de la dirección X (página)
        ldi r25, 125            ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel

Pagina3:

        ldi r17,0b00000011      ;R17 tendrá el valor de la dirección X (página)
        ldi r19,30
        add r16,r19         ;Se aumenta R16 para que llegue como mínimo al valor 0 y sea menor que r25

        ldi r25, 76             ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel

Pagina4:

        ldi r17,0b00000100      ;R17 tendrá el valor de la dirección X (página)
        ldi r25, 217            ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel

Pagina5:

        ldi r17,0b00000101      ;R17 tendrá el valor de la dirección X (página)
        ldi r25, 132            ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel
```

```
Pagina6:
        ldi r17,0b00000110      ;R17 tendrá el valor de la dirección X (página)
        ldi r19,25              ;R25 será el máximo valor R16 posible
        add r16,r19         ;Se aumenta R16 para que llegue como mínimo al valor 0 y sea menor que r25
        ldi r25, 81
        rjmp DetectorDePixel

Pagina7:
        ldi r17,0b00000111
        ldi r25, 218            ;R25 será el máximo valor R16 posible
        rjmp DetectorDePixel


AjusteHor:      ;Una vez se sabe el pixel a encender de la componente vertical habrá que estudiar la
                ;componente horizontal
        mov r30,r18
        ldi r16,0b01000000      ;La conversión se hará según el valor del pin 0 de PORTA
        out admux,r16
        clr r19
        sbi PORTA,2             ;Desactivando el bit 2 y activando el 3 la pantalla proporcionará
        cbi PORTA,3             ;tensiones según la parte vertical pulsada
        rcall Retardo4ms
        sbi adcsra,6            ;Empieza la conversión

LecturaHor:

        sbis adcsra,4           ;Cuando se active el bit la conversión habrá finalizado
        rjmp LecturaHor
        ldi r23,37
        rcall LadoIzquierdo     ;Hasta que no se demuestre lo contrario, el pixel a encender estará en la
                                ;pantalla de la izquierda
        in r16, adcl
        in r24, adch            ;Se guarda el resultado de la conversión en dos registros

        cpi r16,63
        brlo FueraDPantallaIz
        cpi r16,120
        brsh FueraDPantallaDer
        rjmp ContadorDireccionY

FueraDPantallaIz:

        cpi r24,0               ;Si el valor de la componente horizontal es demasiado pequeña
        brne ContadorDireccionY ;será porque se ha pulsado por fuera de la pantalla, se encenderá el
        rjmp DibujoDePixeles2           ;primer pixel

FueraDPantallaDer:

        cpi r24,3                       ;Si el valor de la componente horizontal es demasiado grande
        brne ContadorDireccionY         ;será porque se ha pulsado por fuera de la pantalla, se
        rcall LadoDerecho               ;encenderá el último pixel
        ldi r19,63
        rjmp DibujoDePixeles2
```

```
ContadorDireccionY:

        inc r19             ;R19 tendrá como valor la dirección Y del pixel a encender, según el valor de R16
        cpi r16,6                       ;R19 aumentará mas o menos veces
        brlo ContadorDireccionY2        ;Se irá reduciendo R16 a medida que vamos recorriendo el bucle,
        subi r16,6                      ;cuando sea así pasaremos a estudiar el valor de R24(ADCH)
        rjmp ContadorDireccionY

ContadorDireccionY2:

        cpi r24,0           ;A medida que se va reduciendo ADCH se incrementará R19 la misma cantidad
        breq DibujoDePixeles            ;de veces que si ADCL tuviera el máximo valor (255)
        dec r24
        add r19,r23
        rjmp ContadorDireccionY2        ;Una vez ajustado la dirección se saldrá del bucle

CambioDePantalla:

        subi r19,63
        rcall LadoDerecho       ;A parte de cambiar de pantalla habrá que resetar la dirección
        rjmp DibujoDePixeles2

DibujoDePixeles:

        subi r19,10         ;Habrá que reducir la dirección X por el desfase que hay en la señal de tensión
        cpi r19,64          ;Si la dirección es mayor que 63 se deberá activar la pantalla de la derecha
        brsh CambioDePantalla


DibujoDePixeles2:

        ori r19,0b01000000
        out PORTC, r19          ;Se configura la dirección Y
        rcall Configuracion
        ori r17,0b10111000
        out PORTC, r17          ;Se configura la dirección X
        rcall Configuracion
        rcall Leer      ;Se llama a la subrutina para conocer el estado de la página actual, de forma que no se
        or r18,r28      ;borren pixeles previamente encendidos
        out PORTC, r19
        rcall Configuracion     ;Después de llamar a la subrutina se debe reconfigurar la dirección Y
        out PORTC, r18
        rcall Envio             ;Encendido de pixeles
        rcall Retardo4ms
        rjmp JuegoDibujo        ;Recomienzo del bucle
```

Subrutinas:

```
Cuadrado:       ;Subrutina que dibuja el objeto cuadrado del juego de saltos

        push r16                ;Guardamos los valores de los registros en la pila
        push r17
        ldi r17,6
        cpi r20,0           ;Si R20 es distinto de 0 se deberá dibujar el objeto más arriba para simular salto
        brne Salto
        ldi r16,0b10111110      ;Página adecuada cuando el objeto está en el suelo
        rjmp Cuadrado2

Salto:

        ldi r16,0b10111101      ;En medio del salto estará dibujada una página por encima de lo normal
        dec r20

Cuadrado2:

        out PORTC, r16
        rcall Configuracion     ;Se configura la dirección Y
        ldi r16,0b01000111
        out PORTC, r16
        rcall Configuracion     ;Se configura la página
        ldi r16,0b11111000      ;El objeto tendrá 5 pixeles de altura

BucleCuadrado:                  ;Se repetirá este bucle para ajustar el grosor

        out PORTC, r16
        rcall Envio
        dec r17                 ;Como R17 tiene 6 unidades inicialmente tendrá 6 pixeles de grosor
        brne BucleCuadrado
        pop r16
        pop r17                 ;Recuperamos los valores originales de los registros
        ret
```

```
Obstaculo:      ;Subrutina donde se dibuja el obstáculo y se determina si existe impacto con esta

        push r16            ;Guardamos valor del registro en la pila
        sbrs r19,0          ;R19 determina en que pantalla se dibujara el obstáculo
        rjmp Izquierda
        rjmp Derecha

Izquierda:

        rcall LadoIzquierdo
        rjmp Obstaculo2

Derecha:

        rcall LadoDerecho

Obstaculo2:

        ldi r16,0b10111110      ;El valor de la página será siempre la misma, justo encima del suelo
        out PORTC, r16
```

```
        rcall Configuracion
        ldi r16,0b01000000
        or r16,r18
        out PORTC, r16      ;El valor de Y dependerá del contador R18, que irá aumentando en el bucle
        rcall Configuracion ;principal
        ldi r17,0b11111111
        out PORTC, r17
        rcall Envio             ;El obstáculo tendrá una página entera de altura y un pixel de grosor
        cpi r20,0           ;Si R20 es 0 querrá decir que el objeto cuadrado está en el suelo
        breq Impacto        ;y será posible el impacto
        rjmp FinObstaculo

Impacto:                ;En esta parte del código analizaremos si existe impacto

        sbrs r19,0              ;Si el obstáculo se dibujó en la pantalla de la derecha (donde está el objeto)
        rjmp FinObstaculo       ;las posibilidades se mantienen
        andi r16,0b00111111
        cpi r16,0b00000111
        brlo FinObstaculo       ;Si el obstáculo a dibujar está entre las posiciones 7 y 13 existe impacto
        cpi r16,0b00001101
        brsh FinObstaculo
        dec r29                 ;R29 será reducida, ya que cuenta el número de vidas del jugador
        breq GameOver           ;Si este registro llega a 0 la partida se termina
        clr r19
        clr r18
        clr r21

FinObstaculo:

        pop r16                 ;Se recupera el valor original de R16
        ret

GameOver:

        ldi ZH,high(ImagenGameOver*2)  ;El puntero tendrá la dirección de la tabla con la imagen de
        ldi ZL,low(ImagenGameOver*2)   ;GameOver
        rcall Menu                     ;Se dibuja la imagen

Aqui:

        rjmp Aqui                      ;La imagen se mostrará todo el rato


-------------------------------------------------------------------------------------------------------

Menu:                   ;Subrutina para dibujar imágenes a la pantalla

        push r16        ;Se guarda los valores de los registros en la pila
        push r20
        push r21
        ldi r21, 64     ;R21 tendrá el valor de la dirección Y
        ldi r20, 0      ;R20 tendrá el valor de la dirección X(página)

Return:

        rcall IZQ       ;Se activa la pantalla izquierda y se ajusta la página a dibujar
        rcall Dibujar   ;Se dibuja la página de todas las direcciones Y en la pantalla izquierda
        rcall DER       ;Se activa la pantalla derecha y se ajusta la página a dibujar
        rcall Dibujar   ;Se dibuja la página de todas las direcciones Y en la pantalla derecho
        inc r20         ;La dirección X se verá aumentada para poder recorrer toda la pantalla
```

```
        cpi r20,8               ;Si sobrepasamos la cantidad de paginas existentes se saldrá del bucle
        brne Return
        pop r21                 ;Se recupera los valores de los registros
        pop r20
        pop r16
        ret
```

```
IZQ:            ;Subrutina de apoyo para dibujar imagenes, aqui se ajustará la página
                ;donde se empezará a dibujar en la pantalla izquierda
        push r16                ;Se guarda los valores de los registros en la pila
        push r20
        rcall LadoIzquierdo
        ldi r16,0b01000000
        out PORTC, r16
        rcall Configuracion     ;Se empezará siempre en la dirección 0 de Y
        ldi r16,0b10111000
        or r16,r20
        out PORTC, r16          ;R20 indicará la página a dibujar
        rcall Configuracion
        pop r20                 ;Se recupera los valores de los registros
        pop r16
        ret
```

```
DER:            ;Subrutina de apoyo para dibujar imagenes, aqui se ajustará la página
                ;donde se empezará a dibujar en la pantalla derecha
        push r16                ;Se guarda los valores de los registros en la pila
        push r20
        rcall LadoDerecho
        ldi r16,0b01000000
        out PORTC, r16
        rcall Configuracion     ;Se empezará siempre en la dirección 0 de Y
        ldi r16,0b10111000
        or r16,r20
        out PORTC, r16          ;R20 indicará la página a dibujar
        rcall Configuracion
        pop r20                 ;Se recupera los valores de los registros
        pop r16
        ret
```

```
Dibujar:        ;Subrutina de apoyo para dibujar imagenes, una vez ajustado las direcciones y la
                ;pantalla activada se procede a rellenar la página de los 64 pixeles.
        push r0                 ;Se guarda los valores de los registros en la pila
        push r21

ReDibujar:
```

```
        lpm                 ;Carga a R0 el valor de la dirección apuntada por el puntero.
        out PORTC, r0
        rcall Envio         ;Y se muestra por pantalla
        adiw ZL,1           ;Se aumenta la dirección donde apunta el puntero
        dec r21
        brne ReDibujar      ;Cuando se haya dibujado 64 direcciones Y se terminará el bucle
        pop r21             ;Se recupera los valores de los registros
        pop r0
        ret
```

```
Limpiar:        ;Subrutina para apagar todos los pixeles de la pantalla

        push r18            ;Se guarda los valores de los registros en la pila
        push r17
        push r16
        cbi PORTB,0         ;Se activan los dos controladores
        cbi PORTB,1
        ldi r17,0           ;R17 tendrá el valor de la dirección X(página)
        ldi r18,64          ;R18 tendrá el valor de la dirección Y

Continuar:

        ldi r16,0b00000000
        out PORTC, r16
        rcall Envio         ;Se apagan los pixeles
        dec r18         ;Cuando hayamos recorrido los 64 pixeles de las dos pantallas pasamos a la
        breq SaltoLinea ;siguiente página
        rjmp Continuar

SaltoLinea:

        inc r17         ;Incrementamos la dirección X(página)
        mov r16,r17
        cpi r17,8       ;Si sobrepasamos la cantidad de paginas existentes se saldrá del bucle
        breq Final
        ori r16,0b10111000
        out PORTC, r16
        rcall Configuracion
        ldi r18,64
        rjmp Continuar

Final:

        pop r16         ;Se recupera los valores de los registros
        pop r17
        pop r18
        ret
```

```
Configuracion:      ;Subrutina para configurar la pantalla GLCD

        push r16        ;Se guarda los valores de los registros en la pila
        push r25
        push r26
        push r27
        cbi PORTA,2     ;Modo configuración
        cbi PORTA,3     ;Modo Escritura
        sbi PORTD,6     ;Activación Enable
```

```
        rcall Retardo100us
        cbi PORTD,6      ;Desactvación Enable
        rcall Retardo100us
        pop r27
        pop r26
        pop r25          ;Se recupera los valores de los registros
        pop r16
        ret
```

```
Leer:                    ;Subrutina para leer de la memoria RAM de la pantalla GLCD

        push r16         ;Se guarda los valores de los registros en la pila
        push r18
        push r17
        push r25
        push r26
        push r27
        ldi r16,0
        out DDRC, r16            ;Para poder leer de la pantalla PORTC deberá estar como entrada
        sbi PORTA,2              ;Modo datos
        sbi PORTA,3              ;Modo lectura
        sbi PORTD,6              ;Activación Enable
        rcall Retardo100us
        cbi PORTD,6              ;Desactvación Enable
        rcall Retardo100us
        sbi PORTD,6              ;Activación Enable
        rcall Retardo100us
        in r28, PINC             ;Se lee el contenido del byte de la memoria y se guarda en PORTC
        cbi PORTD,6              ;Desactvación Enable
        rcall Retardo100us
        ldi r16,255              ;Para el buen funcionamiento del resto del código se deberá volver
        out DDRC, r16            ;a establecer PORTC como salida
        pop r27
        pop r26                  ;Se recupera los valores de los registros
        pop r25
        pop r17
        pop r18
        pop r16
        ret
```

```
Envio:

        push r16                 ;Se guarda los valores de los registros en la pila
        push r18
        push r17
        push r25
        push r26
        push r27
        sbi PORTA,2              ;Modo datos
        cbi PORTA,3              ;Modo Escritura
```

```
        sbi PORTD,6             ;Activación Enable
        rcall Retardo100us
        cbi PORTD,6             ;Desactvación Enable
        rcall Retardo100us
        pop r27                 ;Se recupera los valores de los registros
        pop r26
        pop r25
        pop r17
        pop r18
        pop r16
        ret

-------------------------------------------------------------------------------------------------------

LadoIzquierdo:          ;Subrutina para activar la pantalla izquierda

        cbi PORTB,0;
        sbi PORTB,1
        ret

-------------------------------------------------------------------------------------------------------

LadoDerecho:            ;Subrutina para activar la pantalla derecha

        sbi PORTB,0;
        cbi     PORTB,1
        ret

-------------------------------------------------------------------------------------------------------

Inicio:                 ;Subrutina para inicializar las direcciones X e Y a 0

        push r16
        ldi r16,0b01000000
        out PORTC, r16
        rcall Configuracion
        ldi r16,0b10111000
        out PORTC, r16
        rcall Configuracion
        pop r16
        ret

-------------------------------------------------------------------------------------------------------

Retardo100us:           ;Esta subrutina proporciona un retardo de 100 us a partir de 3 bucles

        ldi r25, 2

bucle1100us:        ;Primer bucle, a medida que se vaya reduciendo R25 se restablecerá el valor de R26

        dec r25
        ldi r26, 10
        subi r25, 0
        brne bucle2100us        ;Si R25 llega a 0 se acaba el retardo, si no continua en el segundo bucle
        ret

bucle2100us: ;Segundo bucle, en esta ocasion el registro a reducir será R26, restableciendo r27 cada vez

        ldi r27, 1
        subi r26, 0
```

```
        breq bucle1100us          ;Si R26 es 0 volvemos al primer bucle para reducir R25 y restablecer R26
        dec r26

bucle3100us:                      ;Último bucle, aqui se va reduciendo R27 hasta llegar a 0

        dec r27
        subi r27,0
        breq bucle2100us   ;Cuando llegue a 0 volvemos al segundo bucle para reducir R26 y restablecer
        rjmp bucle3100us   ;R27


-----------------------------------------------------------------------------------------------------

Retardo4ms:                       ;Esta subrutina proporciona un retardo de 4ms a partir de 3 bucles

        ldi r25, 2

bucle14ms:        ;Primer bucle, a medida que se vaya reduciendo R25 se restablecerá el valor de R26

        dec r25
        ldi r26, 3
        subi r25, 0
        brne bucle24ms    ;Si R25 llega a 0 se acaba el retardo, si no continua en el segundo bucle
        ret

bucle24ms:  ;Segundo bucle, en esta ocasion el registro a reducir será R26, restableciendo r27 cada vez

        ldi r27, 255
        subi r26, 0
        breq bucle14ms    ;Si R26 es 0 volvemos al primer bucle para reducir R25 y restablecer R26
        dec r26

bucle34ms:                        ;Último bucle, aqui se va reduciendo R27 hasta llegar a 0

        dec r27
        subi r27,0
        breq bucle24ms   ;Cuando llegue a 0 volvemos al segundo bucle para reducir R26 y restablecer
        rjmp bucle34ms   ;R27

-----------------------------------------------------------------------------------------------------

RetardoVariable:                  ;Esta subrutina proporciona un retardo de variable a partir de 3 bucles

        push r25             ;Se guarda los valores de los registros en la pila
        push r26
        push r27
        mov r25, r26

bucle1Variable:  ;Primer bucle, a medida que se vaya reduciendo R25 se restablecerá el valor de R26

        dec r25
        ldi r26, 10
        subi r25, 0
        brne bucle2100ms
        rjmp FinRetardo        ;Si R25 llega a 0 se acaba el retardo, si no continua en el segundo bucle
```

```
bucle2Variable: ;Segundo bucle, en esta ocasion el registro a reducir será R26, restableciendo r27 cada
                ;vez

        ldi r27, 255
        subi r26, 0
        breq bucle1100ms        ;Si R26 es 0 volvemos al primer bucle para reducir R25 y restablecer R26
        dec r26

bucle3Variable:                 ;Último bucle, aqui se va reduciendo R27 hasta llegar a 0

        dec r27
        subi r27,0
        breq bucle2100ms    ;Cuando llegue a 0 volvemos al segundo bucle para reducir R26 y restablecer
        rjmp bucle3100ms    ;R27

FinRetardo:

        pop r27             ;Se recupera los valores de los registros
        pop r26
        pop r25
        ret

-------------------------------------------------------------------------------------------------------

Vidas:                  ;Subrutina que elige el número a dibujar según la cantidad de vidas

        push r16        ;Se guarda el valor del registro en la pila
        sbrc r29,0
        rcall Uno       ;Si el jugador tiene una vida se dibujará el número uno
        sbrc r29,1
        rcall Dos       ;Si el jugador tiene una vida se dibujará el número dos
        pop r16         ;Se recupera el valor del registro
        ret

-------------------------------------------------------------------------------------------------------

Saltos2:                ;Subrutina para elegir las unidades del número a dibujar según los saltos

        cpi r28,9       ;Se estudiará el valor de R28, asociado a las unidades en la cantidad de saltos
        breq EsNueve    ;Según su valor se llamará a una subrutina u otra para dibujar el dibujo
        cpi r28,8       ;correspondiente
        breq EsOcho
        cpi r28,7
        breq EsSiete
        cpi r28,6
        breq EsSeis
        cpi r28,5
        breq EsCinco
        cpi r28,4
        breq EsCuatro
        cpi r28,3
        breq EsTres
        cpi r28,2
        breq EsDos
        cpi r28,1
        breq EsUno

        rcall Cero
        ret
```

```
EsNueve:
        rcall Nueve
        ret
EsOcho:
        rcall Ocho
        ret
EsSiete:
        rcall Siete
        ret

EsSeis:
        rcall Seis
        ret
EsCinco:
        rcall Cinco
        ret
EsCuatro:
        rcall Cuatro
        ret
EsTres:
        rcall Tres
        ret
EsDos:
        rcall Dos
        ret
EsUno:
        rcall Uno
        ret
```

```
Saltos:                 ;Subrutina para elegir las decenas del número a dibujar según los saltos

        cpi r25,9       ;Se estudiará el valor de R25, asociado a las decenas en la cantidad de saltos
        breq EsNueve    ;Según su valor se llamará a una subrutina u otra para dibujar el dibujo
        cpi r25,8       ;correspondiente
        breq EsOcho
        cpi r25,7
        breq EsSiete
        cpi r25,6
        breq EsSeis
        cpi r25,5
        breq EsCinco
        cpi r25,4
        breq EsCuatro
```

```
        cpi r25,3
        breq estrés
        cpi r25,2
        breq EsDos
        cpi r25,1
        breq EsUno
        rcall Cero
        ret
```

;Conjunto de subrutinas para dibujar números. En todos se encenderán o no pixeles de forma que el
;número a representar sea identificable

Cero:

```
        push r16
        ldi r16,0b11111111
        out PORTC,r16
        rcall Envio
        ldi r16,0b10000001
        out PORTC,r16
        rcall Envio
        ldi r16,0b10000001
        out PORTC,r16
        rcall Envio
        ldi r16,0b10000001
        out PORTC,r16
        rcall Envio
        ldi r16,0b10000001
        out PORTC,r16
        rcall Envio
        ldi r16,0b11111111
        out PORTC,r16
        rcall Envio
        pop r16
        ret
```

Uno:

```
        push r16
        ldi r16,0b00000000
        out PORTC,r16
        rcall Envio
        ldi r16,0b00000000
        out PORTC,r16
        rcall Envio
        ldi r16,0b00000000
        out PORTC,r16
        rcall Envio
        ldi r16,0b11111111
        out PORTC,r16
        rcall Envio
```

```
ldi r16,0b00000000
out PORTC,r16
rcall Envio
ldi r16,0b00000000
out PORTC,r16
rcall Envio
pop r16
ret
```

Dos:

```
push r16
ldi r16,0b11110001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10011111
out PORTC,r16
rcall Envio
pop r16
ret
```

Tres:

```
push r16
ldi r16,0b10000001
out PORTC,r16
rcall Envio
ldi r16,0b10000001
out PORTC,r16
rcall Envio
ldi r16,0b10000001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b11111111
out PORTC,r16
rcall Envio
pop r16
ret
```

Cuatro:

```
push r16
ldi r16,0b00011111
out PORTC,r16
rcall Envio
ldi r16,0b00010000
out PORTC,r16
rcall Envio

ldi r16,0b00010000
out PORTC,r16
rcall Envio
ldi r16,0b00010000
out PORTC,r16
rcall Envio
ldi r16,0b00010000
out PORTC,r16
rcall Envio
ldi r16,0b11111111
out PORTC,r16
rcall Envio
pop r16
ret
```

Cinco:

```
push r16
ldi r16,0b10011111
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b11110001
out PORTC,r16
rcall Envio
pop r16
ret
```

Seis:

```
push r16
ldi r16,0b11111111
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
```

```
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio

ldi r16,0b11110001
out PORTC,r16
rcall Envio
pop r16
ret
```

Siete:

```
push r16
ldi r16,0b00000001
out PORTC,r16
rcall Envio
ldi r16,0b00000001
out PORTC,r16
rcall Envio
ldi r16,0b00000001
out PORTC,r16
rcall Envio
ldi r16,0b00010001
out PORTC,r16
rcall Envio
ldi r16,0b11111111
out PORTC,r16
rcall Envio
ldi r16,0b00010000
out PORTC,r16
rcall Envio
pop r16
ret
```

Ocho:

```
push r16
ldi r16,0b11111111
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
ldi r16,0b10010001
out PORTC,r16
rcall Envio
```

```
ldi r16,0b11111111
out PORTC,r16
rcall Envio
pop r16
ret
```

Nueve:

```
push r16
ldi r16,0b00011111
out PORTC,r16
rcall Envio
ldi r16,0b00010001
out PORTC,r16
rcall Envio
ldi r16,0b00010001
out PORTC,r16
rcall Envio
ldi r16,0b00010001
out PORTC,r16
rcall Envio
ldi r16,0b00010001
out PORTC,r16
rcall Envio
ldi r16,0b11111111
out PORTC,r16
rcall Envio
pop r16
ret
```

Espacio:

```
push r16
ldi r16,0b00000000
out PORTC,r16
rcall Envio
ldi r16,0b00000000
out PORTC,r16
rcall Envio
ldi r16,0b00000000
out PORTC,r16
rcall Envio
pop r16
ret
```

Tablas:

;Aqui se encuentran las tablas con la información para mostrar distintos imágenes

Seleccion:

```
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
```

```
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xC7, 0x87, 0x97, 0x27, 0x27, 0x3F, 0x1F
.db 0x5F, 0x5F, 0x1F, 0x3F, 0x07, 0x07, 0x07, 0x3F, 0x1F, 0x5F, 0x5F, 0x1F, 0x3F, 0x3F, 0x1F, 0xDF
.db 0xDF, 0xDF, 0x3F, 0x1F, 0xDF, 0xDF, 0xDF, 0x17, 0x17, 0x17, 0x3F, 0x1F, 0x9F, 0xDF, 0x1F, 0x3F
.db 0x1F, 0x1F, 0x1F, 0x9F, 0x1F, 0x3F, 0x3F, 0x1F, 0x5F, 0x5F, 0x1F, 0x3F, 0xFF, 0xFF, 0x17, 0x17
.db 0x17, 0x1F, 0x1F, 0x1F, 0xFF, 0x1F, 0x1F, 0x3F, 0x1F, 0x5F, 0x5F, 0x1F, 0x3F, 0x1F, 0x1F, 0x5F
.db 0x1F, 0x1F, 0x1F, 0x1F, 0x9F, 0xDF, 0x1F, 0x3F, 0xDF, 0xDF, 0xDF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC, 0xFC, 0xFD, 0xFC, 0xFC, 0xFE, 0xFC
.db 0xFD, 0xFD, 0xFD, 0xFD, 0xFC, 0xFC, 0xFC, 0xFE, 0xFC, 0xFD, 0xFD, 0xFD, 0xFD, 0xFE, 0xFC, 0xFD
.db 0xFD, 0xFD, 0xFE, 0xFC, 0xFD, 0xFD, 0xFD, 0xFC, 0xFC, 0xFC, 0xFE, 0xFC, 0xFD, 0xFD, 0xFC, 0xFE
.db 0xFC, 0xFC, 0xFC, 0xFF, 0xFC, 0xFC, 0xFE, 0xFC, 0xFD, 0xFD, 0xFD, 0xFD, 0xFF, 0xF7, 0xF0, 0xF0
.db 0xF8, 0xFE, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFE, 0xFC, 0xFD, 0xFD, 0xFD, 0xFD, 0xF0, 0xF0, 0xF5
.db 0xF1, 0xF1, 0xFA, 0xFC, 0xFD, 0xFD, 0xFC, 0xFE, 0xFD, 0xFD, 0xFD, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x03, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB
.db 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB
.db 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB
.db 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0x03, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0x03, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB
.db 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB
.db 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB
.db 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0xFB, 0x03, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0xDF, 0x9F, 0x1F, 0x3F, 0xFF
.db 0xFF, 0x1F, 0x1F, 0xFF, 0xFF, 0x1F, 0x1F, 0x5F, 0x1F, 0x1F, 0xFF, 0xFF, 0x1F, 0x1F, 0xFF, 0xFF
.db 0x1F, 0x1F, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0x1F, 0x7F, 0xFF, 0xFF
.db 0x1F, 0x1F, 0xDF, 0x1F, 0x1F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F
```

```
.db 0x5F, 0xDF, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0x1F, 0x7F, 0xFF, 0xFF, 0x1F, 0x1F, 0xFF, 0xFF, 0xFF
.db 0xDF, 0xDF, 0x1F, 0x1F, 0xDF, 0xDF, 0xFF, 0x3F, 0x1F, 0xDF, 0xDF, 0x1F, 0x3F, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xF8, 0xF8, 0xFB, 0xF9, 0xF8, 0xFC, 0xFF
.db 0xFF, 0xF8, 0xF8, 0xFF, 0xFF, 0xF8, 0xF8, 0xFB, 0xF8, 0xF8, 0xFF, 0xFF, 0xFC, 0xF8, 0xF9, 0xF9
.db 0xF8, 0xFC, 0xFF, 0xFB, 0xFB, 0xF8, 0xF8, 0xFF, 0xFF, 0xF8, 0xF8, 0xFD, 0xF8, 0xF8, 0xFB, 0xFF
.db 0xF8, 0xF8, 0xFC, 0xF8, 0xF9, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFB, 0xFA
.db 0xF8, 0xF8, 0xFF, 0xFF, 0xF8, 0xF8, 0xFD, 0xF8, 0xF8, 0xFB, 0xFF, 0xF8, 0xF8, 0xFB, 0xFB, 0xFB
.db 0xFF, 0xFF, 0xF8, 0xF8, 0xFF, 0xFF, 0xFF, 0xFC, 0xF8, 0xFB, 0xFB, 0xF8, 0xFC, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x80, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF
.db 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF
.db 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF
.db 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0x80, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0x80, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF
.db 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF
.db 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF
.db 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0xBF, 0x80, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF

MenuSalto:
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF
.db 0xE1, 0x81, 0x8F, 0x81, 0xE1, 0xFD, 0xFF, 0x81, 0x81, 0xFF, 0xFF, 0x81, 0x81, 0xBD, 0x99, 0x81
.db 0xC3, 0xFF, 0xFF, 0x8F, 0x81, 0xD1, 0x81, 0x87, 0xBF, 0xFF, 0xB1, 0xA1, 0x85, 0x8D, 0xFF, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xB1, 0xA1, 0x85, 0x8D, 0xFF, 0xFF, 0x8F, 0x81, 0xD1, 0x81
.db 0x87, 0xBF, 0xFF, 0x81, 0x81, 0xBF, 0xBF, 0xBF, 0xFD, 0xFD, 0x81, 0x81, 0xFD, 0xFD, 0xFF, 0xC3
.db 0x81, 0xBD, 0xBD, 0x81, 0xC3, 0xFF, 0xFF, 0xB1, 0xA1, 0x85, 0x8D, 0xFF, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F
.db 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20
.db 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x3F, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20
.db 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20
.db 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x3F, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

ImagenGameOver:
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
```

```
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0x87, 0x03, 0x13, 0x5B, 0x13, 0x13, 0xFF, 0xFF, 0x1F, 0x03, 0xA3, 0x03, 0x0F, 0x7F, 0xFF, 0x03
.db 0x03, 0x83, 0x0F, 0x0F, 0x83, 0x03, 0x03, 0xFF, 0xFF, 0x03, 0x03, 0x6B, 0x6B, 0x7B, 0xFF, 0x87
.db 0x03, 0x7B, 0x7B, 0x03, 0x87, 0xFF, 0xFB, 0xC3, 0x03, 0x1F, 0x03, 0xC3, 0xFB, 0xFF, 0x03, 0x03
.db 0x6B, 0x6B, 0x7B, 0xFF, 0x03, 0x03, 0x9B, 0x03, 0x23, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
```

```
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
.db 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
```