CLASE 5

MEJORANDO EL REPERTORIO DE INSTRUCCIONES (ISA)
Parte II
La pila, entrada/salida e interrupciones

MEJORANDO EL REPERTORIO DE INSTRUCCIONES (ISA)

Algunas mejoras que suelen incorporarse a los procesadores. Requieren nuevas instrucciones y registros. Requieren menos instrucciones para realizar la misma tarea.

a) Aumento de la cantidad de memoria
b) Aumento del número de registros de propósitos generales: disminución de los accesos a memoria.
c) Mejora de la ALU: operaciones y tipos de datos
d) Ampliación del repertorio de saltos condicionales
e) Loops: mejoras en las repeticiones
f) Nuevos modos de direccionamiento y registros de uso específico: mejora en el acceso a los datos
g) La pila: mejora en la implementación de subrutinas
h) El sistema de entrada/salida, interrupciones

Subrutinas en MARIE

Instrucciones **JnS** y **JumpI**

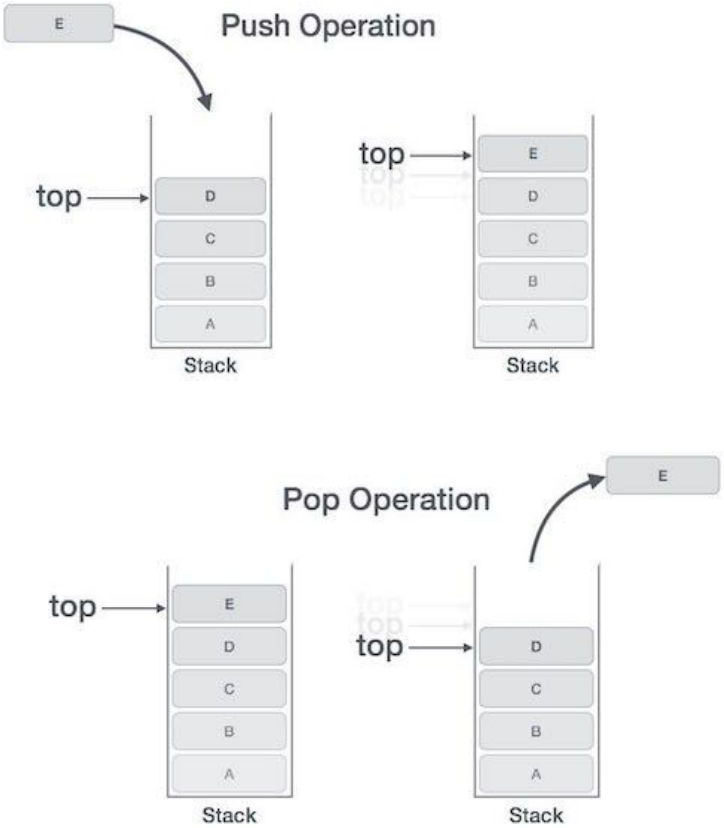
Opcode	Instruction	RTN
0000	JnS *X*	MBR ⟵ PC MAR ⟵ *X* M[MAR] ⟵ MBR MBR ⟵ *X* AC ⟵ 1 AC ⟵ AC + MBR PC ⟵ AC
1100	JumpI *X*	MAR ⟵ *X* MBR ⟵ M[MAR] PC ⟵ MBR

¿Cómo se le pasa el argumento a la subrutina? ¿Por qué no se utiliza el acumulador?

¿Cómo retorna el resultado? ¿Por qué ahora sí puede utilizarse el acumulador?

```
/ Duplicar con subrutina

      Load  X          / Primer número a doblar
      Store Tmp        / Uso Tmp para pasar el número
      JnS Dupl         / Salto a subrutina
      Store X          / Almaceno el primer número, duplicado

      Load  Y          / Segundo número y repito
      Store Tmp
      JnS Dupl
      Store Y
      Halt

X,    DEC   20
Y,    DEC   48
Tmp,  DEC   0          / Argumento (global, implicito)

Dupl, HEX   0          / JnS guarda aquí la dirección de retorno
      Load  Temp       / Subrutina
      Add   Temp       / Duplico
      JumpI Dupl       / Retorno
```

Una posible implementación de pila (stack) en MARIE



https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

Stack data structure: LIFO abstract data type
Implementación:

a) Nuevo registro Stack Pointer (SP, puntero de pila), contiene la dirección del top (init final).
b) Una zona de memoria RAM (final)
c) Dos instrucciones nuevas (operaciones primitivas).

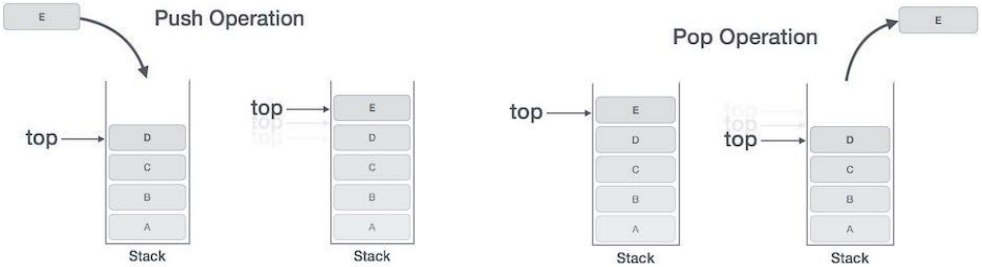
PUSH (apilar): mover el contenido del acumulador al stack

```
MBR <- ACC
SP <- SP - 1
MAR <- SP
M[MAR] <- MBR
```

POP (desapilar): mover el primer elemento (top) del stack al acumulador

```
MAR <- SP
MBR <- M[MAR]
ACC <- MBR
SP <- SP + 1
```

Una posible implementación de pila (stack) en MARIE



SUBRUTINAS - Solución

```
CALL Subr
       ACC <- PC           ; Dirección de retorno a la pila
       PUSH                ; (Trace)
       PC <- Subr          ; Salto

RET
       POP                 ; Dirección de retorno de la pila
       PC <- ACC
```

Si se desea pasar argumentos a la subrutina se puede hacer también por la pila, previo al salto. Se debe asegurar que se "consumen" en el orden correcto. Idem para retornar. Hay que manipular el SP.
STACK FRAME

Stack data structure: LIFO
Implementación:

a) Nuevo registro Stack Pointer (SP, puntero de pila), contiene la dirección del top (init final).
b) Una zona de memoria RAM (final)
c) Dos instrucciones nuevas (operaciones primitivas).

PUSH (apilar): mover el contenido del acumulador al stack

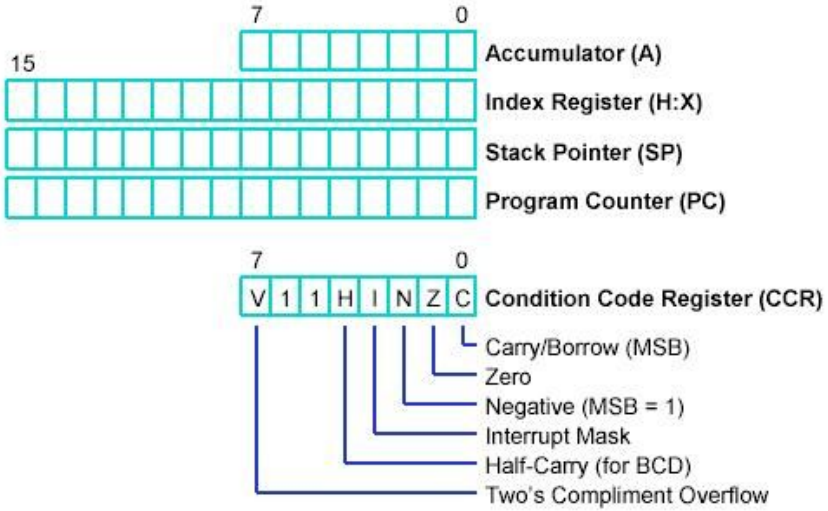
```
MBR <- ACC
SP <- SP - 1
MAR <- SP
M[MAR] <- MBR
```

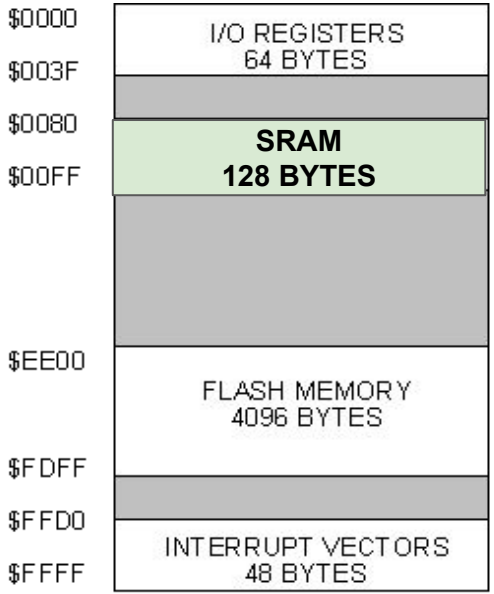
POP (desapilar): mover el primer elemento (top) del stack al acumulador

```
MAR <- SP
MBR <- M[MAR]
ACC <- MBR
SP <- SP + 1
```

CPU08 Stack Pointer

Init $00FF y crece hacia abajo (direcciones menores, en la figura hacia arriba)





68HC908QT4 and 68HC908QY4 Memory Map

[Manual 68HC08]

PSHA Push Accumulator onto Stack (PUSH)
PULA Pull Accumulator from Stack (POP)
JSR Jump to Subroutine (CALL)
RTS Return from Subroutine (RET)

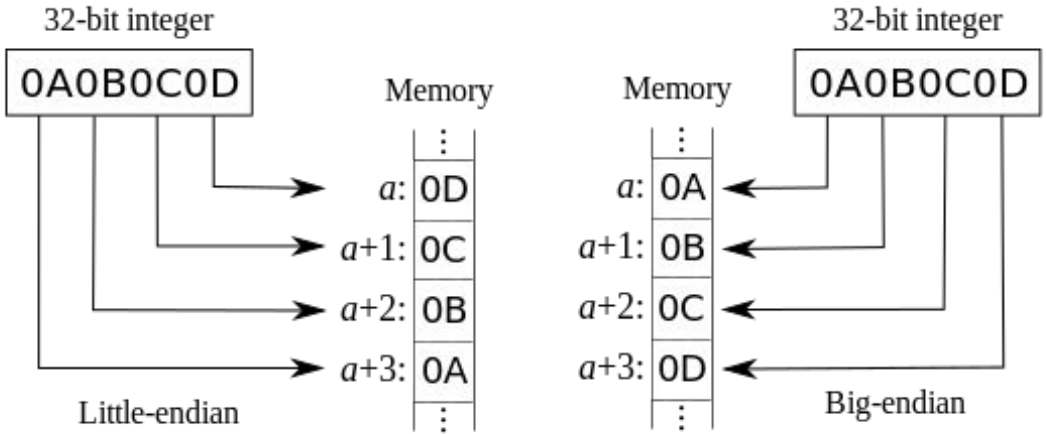
Ejemplo

```
0      ---
1      ---
2      ---
3      ---
4      CALL SUB1           ; PUSH 5, JMP 100, SP-1=FFE
5      ---
6      ---
7      CALL SUB2           ; PUSH 8, JMP 200, SP-1=FFE
8      ---
9      ---


100    SUB1:  ---
101           ---
102           ---
103           ---
104           RET          ; Primera vez: POP 5, JMP 5, SP+1=FFF   ; Segunda vez: POP 203, JMP 203, SP+1=FFE



200    SUB2:  ---
201           ---
202           CALL SUB1    ; PUSH 203, JMP 100, SP-1=FFD
203           ---
204           RET          ; POP 8, JMP 8, SP=FFF


FFD
FFE          203
FFF    5     8            ; la pila arranca en la dirección FFF y crece hacia “arriba” (PUSH disminuye SP)
```

Endianness

Formato en el que se almacenan los datos de más de un byte
big-endian Motorola, little-endian Intel

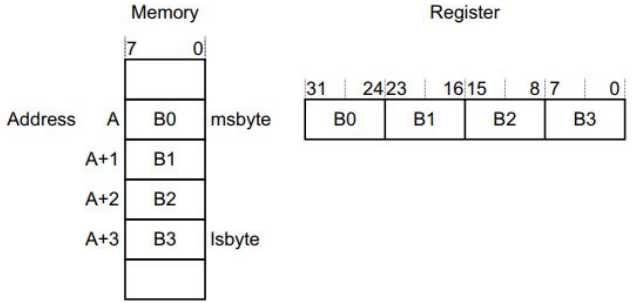
```c
#include <stdio.h>
#include <stdint.h>

int main(void)
{
   int16_t i = 1;
   int8_t *p = (int8_t *) &i;

   if (p[0] == 1) printf("Little Endian\n");
   else           printf("Big Endian\n");

   return 0;
}
```



Endianness

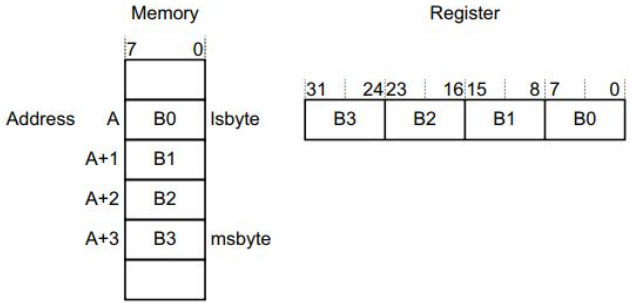
Byte-invariant big-endian format

In byte-invariant big-endian format, the processor stores the *most significant byte* (msbyte) of a word at the lowest-numbered byte, and the *least significant byte* (lsbyte) at the highest-numbered byte. For example:



Little-endian format

In little-endian format, the processor stores the lsbyte of a word at the lowest-numbered byte, and the msbyte at the highest-numbered byte. For example:



Tarea

MARIE no tiene pila ¿Se podría implementar una pila por software?
Pista: Ver las últimas instrucciones del repertorio:

```
StoreI (opcode D)
LoadI  (opcode E)
```

¿Cómo funcionan la pila y las llamadas a subrutina en AVR?
https://microchipdeveloper.com/8avr:stack

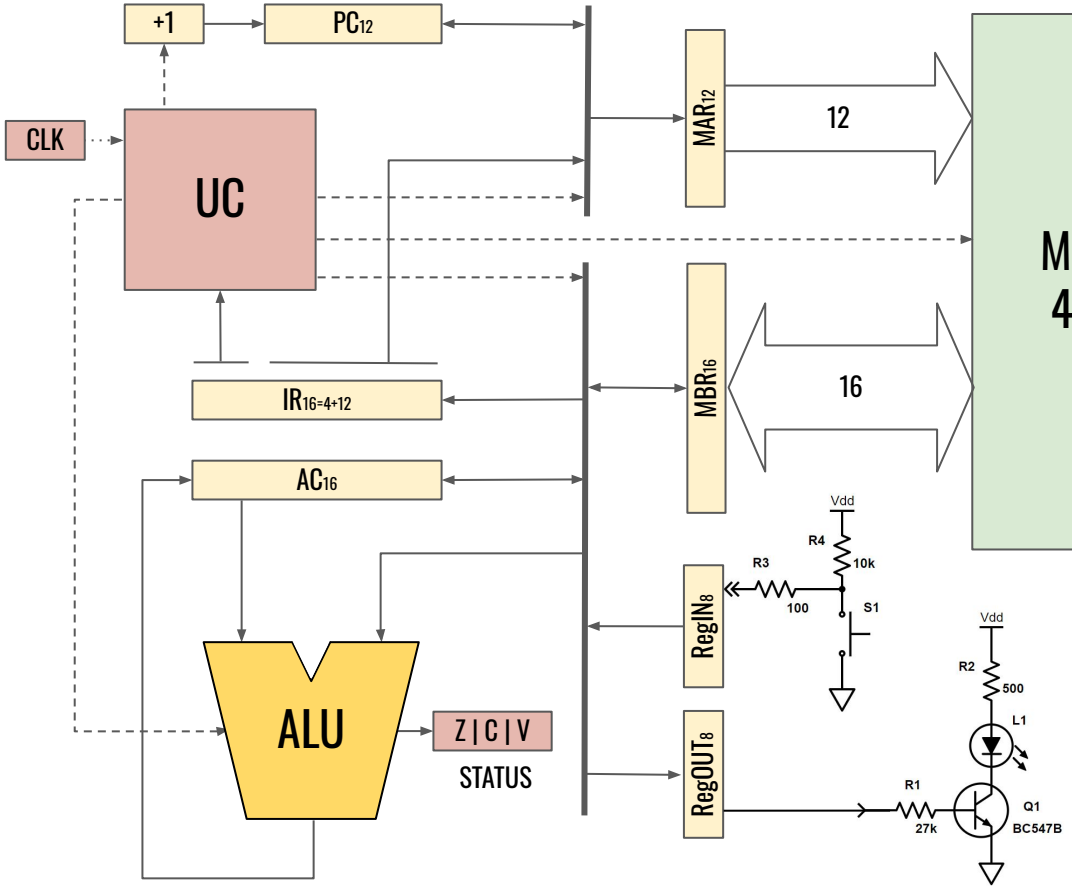
Intervalo 15’

MEJORANDO EL REPERTORIO DE INSTRUCCIONES (ISA)

Algunas mejoras que suelen incorporarse a los procesadores. Requieren nuevas instrucciones y registros. Requieren menos instrucciones para realizar la misma tarea.

a) Aumento de la cantidad de memoria
b) Aumento del número de registros de propósitos generales: disminución de los accesos a memoria.
c) Mejora de la ALU: operaciones y tipos de datos
d) Ampliación del repertorio de saltos condicionales
e) Loops: mejoras en las repeticiones
f) Nuevos modos de direccionamiento y registros de uso específico: mejora en el acceso a los datos
g) La pila: mejora en la implementación de subrutinas
h) El sistema de entrada/salida, interrupciones

Entrada/salida en MARIE




```
0001) LOAD X
        MBR <- M[MAR]
        AC  <- MBR
0010) STORE X
        MBR <- AC
        M[MAR] <- MBR
0101) INPUT
        AC <- RegIN
0110) OUTPUT
        RegOUT <- AC
```

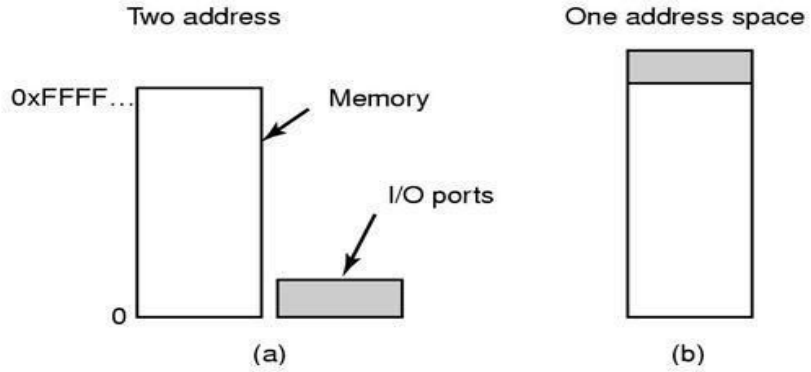
Entrada de 1 bit: botón (teclado, mouse)... Sensor digital, fin de carrera, ADC y sensor analógico Contador/Temporizador

Salida de 1 bit: LED (display, monitor)... Actuador electromecánico, electromagnético, relé, etc. DAC

Se pueden agregar registros de i/o?

Los registros de i/o se podrían ubicar en el mapa de memoria?

Port mapped I/O	Memory mapped I/O
Un espacio de direcciones de memoria y otro separado para entrada/salida	Un espacio de direccionamiento único para memoria y dispositivos de entrada/salida
El espacio de memoria está completamente disponible	Los dispositivos de entrada/salida ocupan una zona del espacio de memoria
Instrucciones diferentes para memoria (load/store) y para entrada/salida (input/output). Pueden utilizarse diferentes modos de direccionamiento (bit-addressable)	Las mismas instrucciones para acceder a memoria y a entrada/salida (load/store)



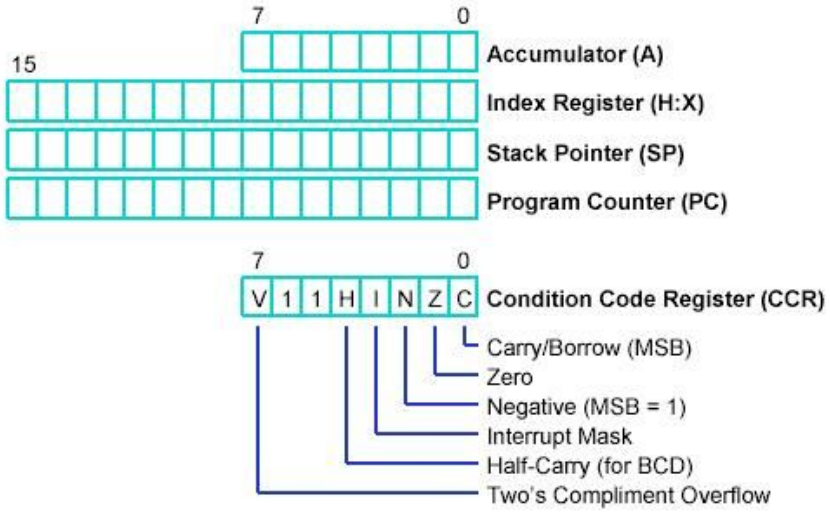
MARIE **Port-mapped I/O**
- Un registro de 8 bits de entrada **RegIN**
- Un registro de 8 bits de salida **RegOUT**
- Una instrucción para leer el registro de entrada **INPUT** (ACC <- RegIN)
- Una instrucción para escribir el registro de salida **OUTPUT** (RegOUT <- ACC)

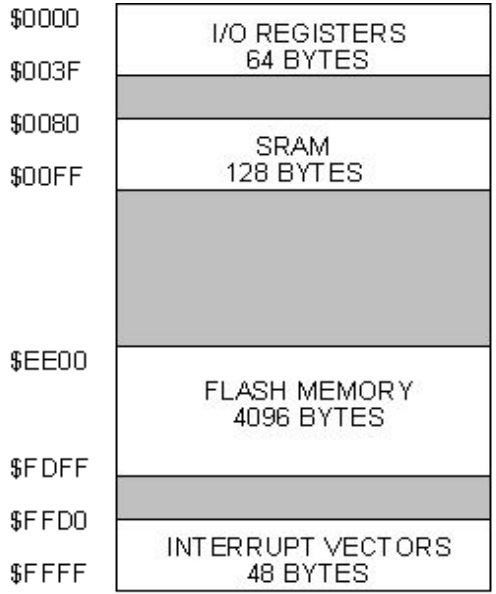
CPU08 **Memory-mapped I/O**
- **64 bytes** del mapa de memoria
- **ld / st** y sus modos de direccionamiento

AVR **Ambos,** si se usa port-mapped es más rápido y además bit-addressable

CPU08 memory map

Memory mapped io, 64 bytes del mapa de memoria, load/store y sus modos de direccionamiento

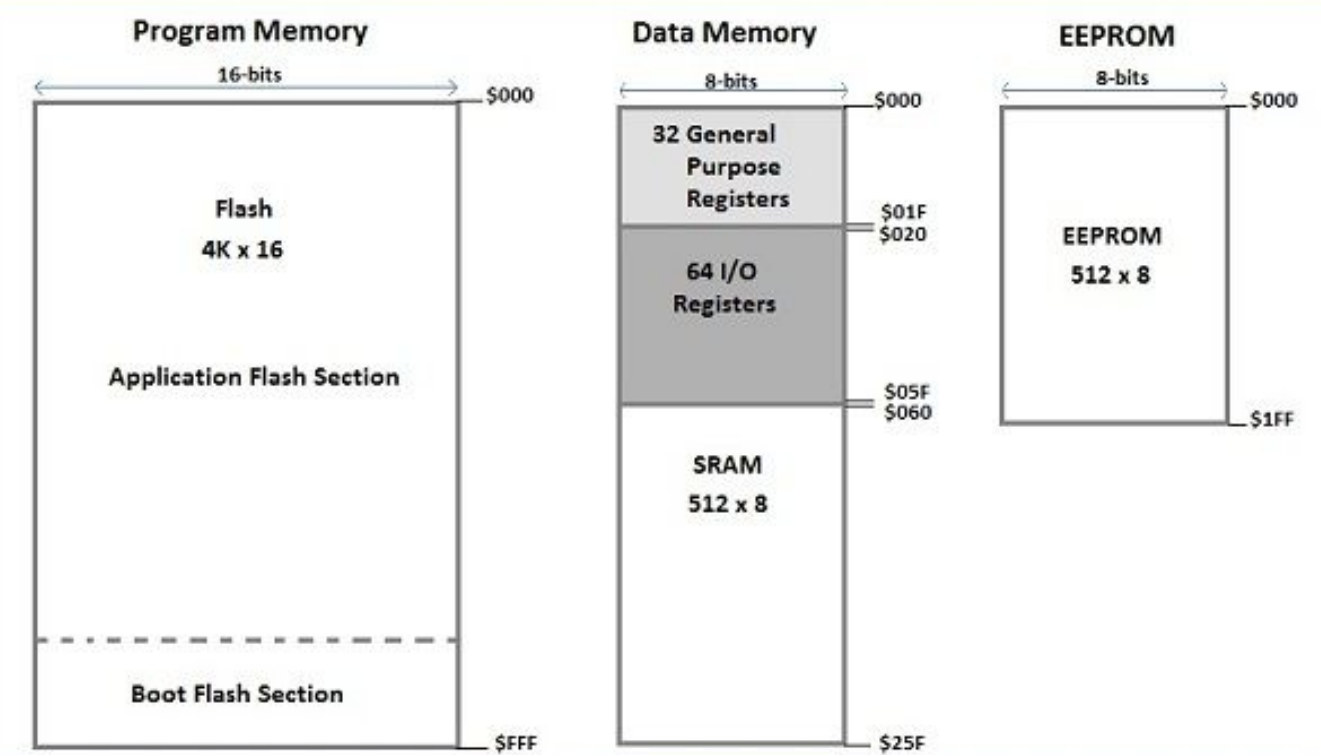




68HC908QT4 and 68HC908QY4 Memory Map

AVR memory map

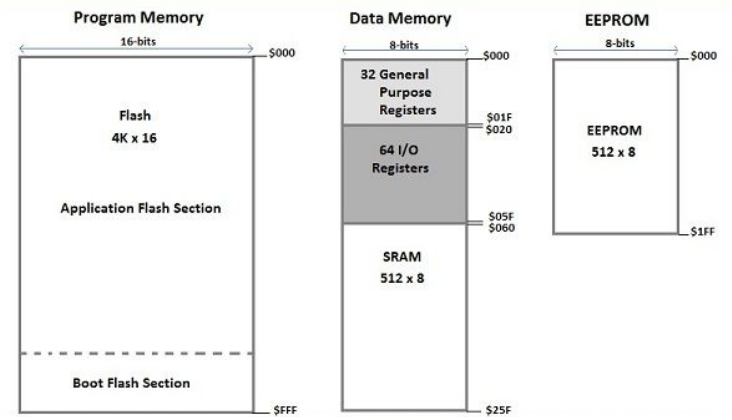
Memory-mapped I/O (0x020 a 0x05F), 64 registros de 8 bits



AVR memory map

Memory-mapped I/O (0x020 a 0x05F), 64 (0x40) registros de 8 bits

http://www.rjhcoding.com/avr-asm-io.php




```
; Ejemplo: Leer el Port B (registro PINB)

in r25,$03          ; 1 word - 1 cycle

lds r25,$0023       ; 2 words - 2 cycles

clr r27
ldi r26,$23
ld r25,X+           ; 1 word - 2 cycles
```

Memory Mapped I/O:
LOAD/STORE, direcciones 0x0020-0x005F

```
LDS     Words 2 (4 bytes)     Cycles 2          Directo
LD      Words 1 (2 bytes)     Cycles 2          Indirecto
```

Codificación: Instrucción larga de 2 palabras (o corta si uso direccionamiento indirecto). 5 bits de registro y 16 bits de posición de memoria (o implícito si uso puntero).
Siempre se ejecuta en **dos ciclos**.
Ventaja: Acepta modos de direccionamiento directo (LDS) e indirecto (LD) usando X, Y y Z, con post incremento/decremento.

Port Mapped I/O:
IN/OUT, direcciones 0x00-0x3F

```
IN      Words 1 (2 bytes)    Cycles 1
```

Codificación: Instrucción corta: 5 bits de registro y 6 bits de port. Entra todo en una palabra.
Siempre ejecuta en **un ciclo.**
Ventaja: Acepta direccionamiento por bits sólo para los primeros 32 (CBI/SBI). También skip condicional (SBIC/SBIS).

Each port consists of three registers, where x = Port Name (A, B, C or D)

DDRx – Data Direction Register

The DDxn bits in the DDRx Register select the direction of this pin. If DDxn is written to '1', Pxn is configured as an output pin. If DDxn is written to '0', Pxn is configured as an input pin.

PORTx – Pin Output Register

As an Output: If a '1' is written to the bit when the pin is configured as an output pin, the port pin is driven high. If a '0' is written to the bit when the pin is configured as an output pin, the port pin is driven low.
As an Input: If a '1' is written to the bit when the pin is configured as an input pin, the pull-up resistor is activated. If a '0' is written to the bit when the pin is configured as an input pin, the port pin is tri-stated.

PINx – Pin Input Register

The PINxn bits in the PINx register are used to read data from port pin. When the pin is configured as a digital input (in the DDRx register), and the pull-up is enabled (in the PORTx register) the bit will indicate the state of the signal at the pin (high or low).
Note: If a port is made an output, then reading the PINx register will give you data that has been written to the port pins.
As a Tri-State Input: When the PORTx register disables the pull-up resistor the input will be tri-stated, leaving the pin left floating. When left in this state, even a small static charge present on surrounding objects can change the logic state of the pin. If you try to read the corresponding bit in the pin register, its state cannot be predicted.

Name: DDRB
Offset: 0x24
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x04

Bit	7	6	5	4	3	2	1	0
	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – DDRBn: Port B Data Direction [n = 7:0]

Name: PORTB
Offset: 0x25
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x05

Bit	7	6	5	4	3	2	1	0
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – PORTBn: Port B Data [n = 0:7]

Name: PINB
Offset: 0x23
Reset: N/A
Property: When addressing as I/O Register: address offset is 0x03

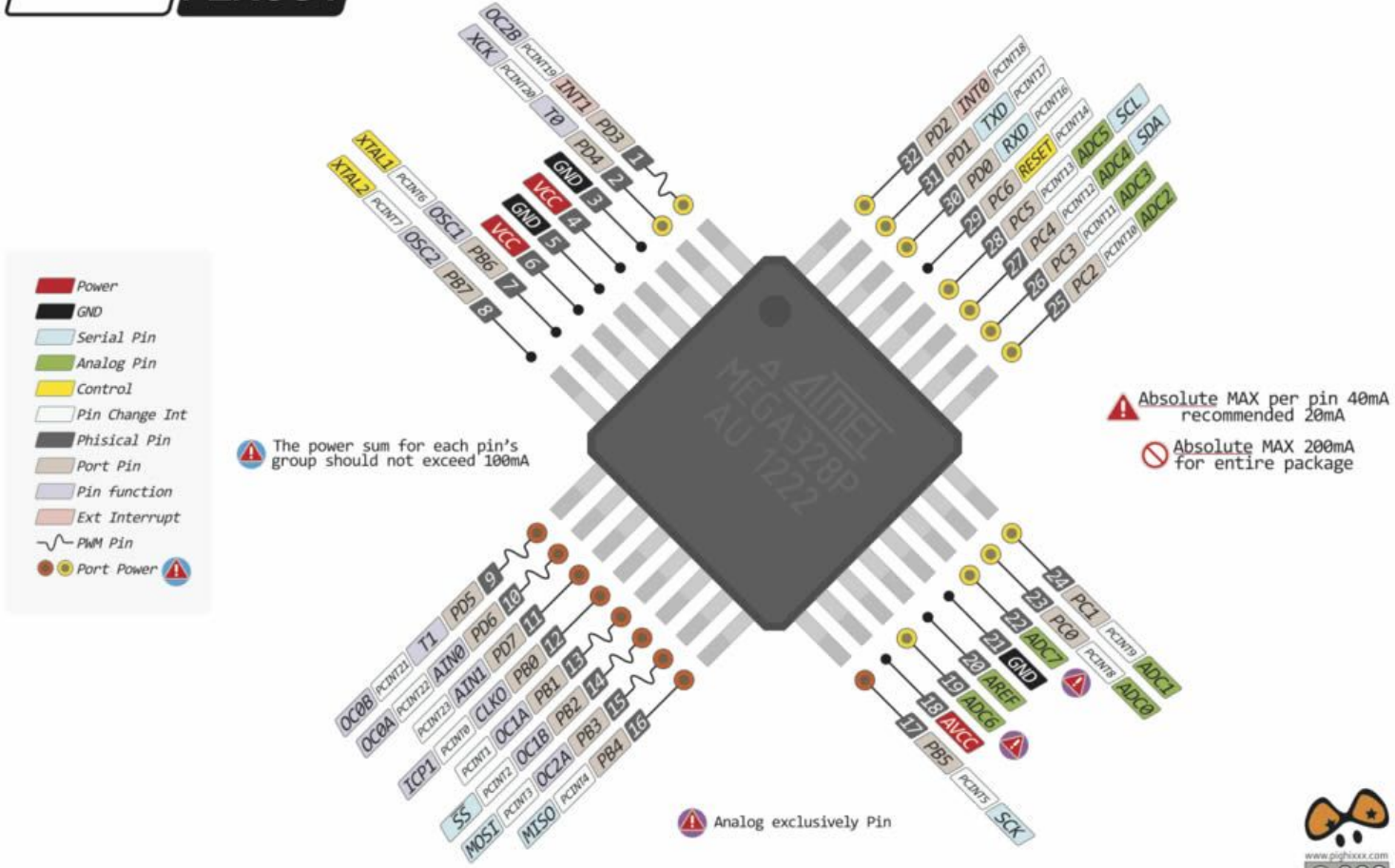
Bit	7	6	5	4	3	2	1	0
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Bits 7:0 – PINBn: Port B Input Pins Address [n = 7:0]

Offset	Name	Bit Pos.								
0x23	PINB	7:0	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x24	DDRB	7:0	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
0x25	PORTB	7:0	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x26	PINC	7:0		PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x27	DDRC	7:0		DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
0x28	PORTC	7:0		PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x29	PIND	7:0	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x2A	DDRD	7:0	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
0x2B	PORTD	7:0	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x2C ... 0x34	Reserved									
0x35	TIFR0	7:0						OCFB	OCFA	TOV
0x36	TIFR1	7:0			ICF			OCFB	OCFA	TOV
0x37	TIFR2	7:0						OCFB	OCFA	TOV
0x38										

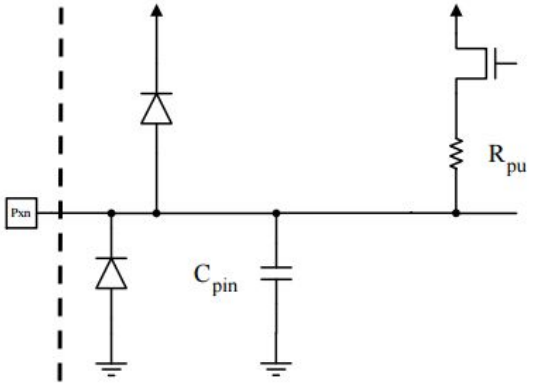
ATMEGA328 PINOUT

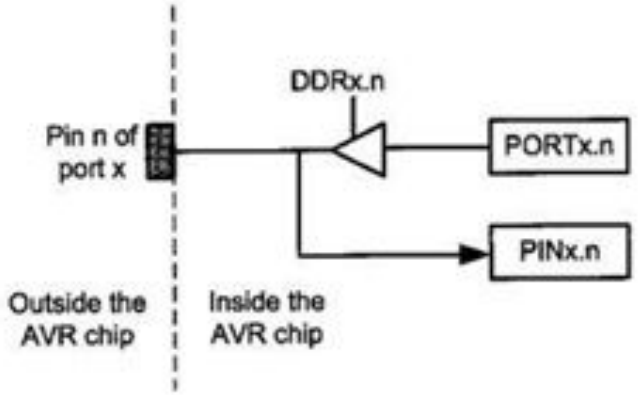
Pinout ATmega328 SMD



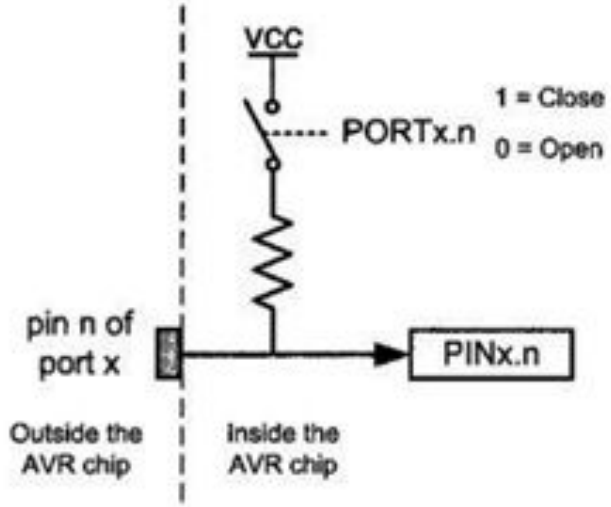
16 JUL 2014

ver 3 rev 0

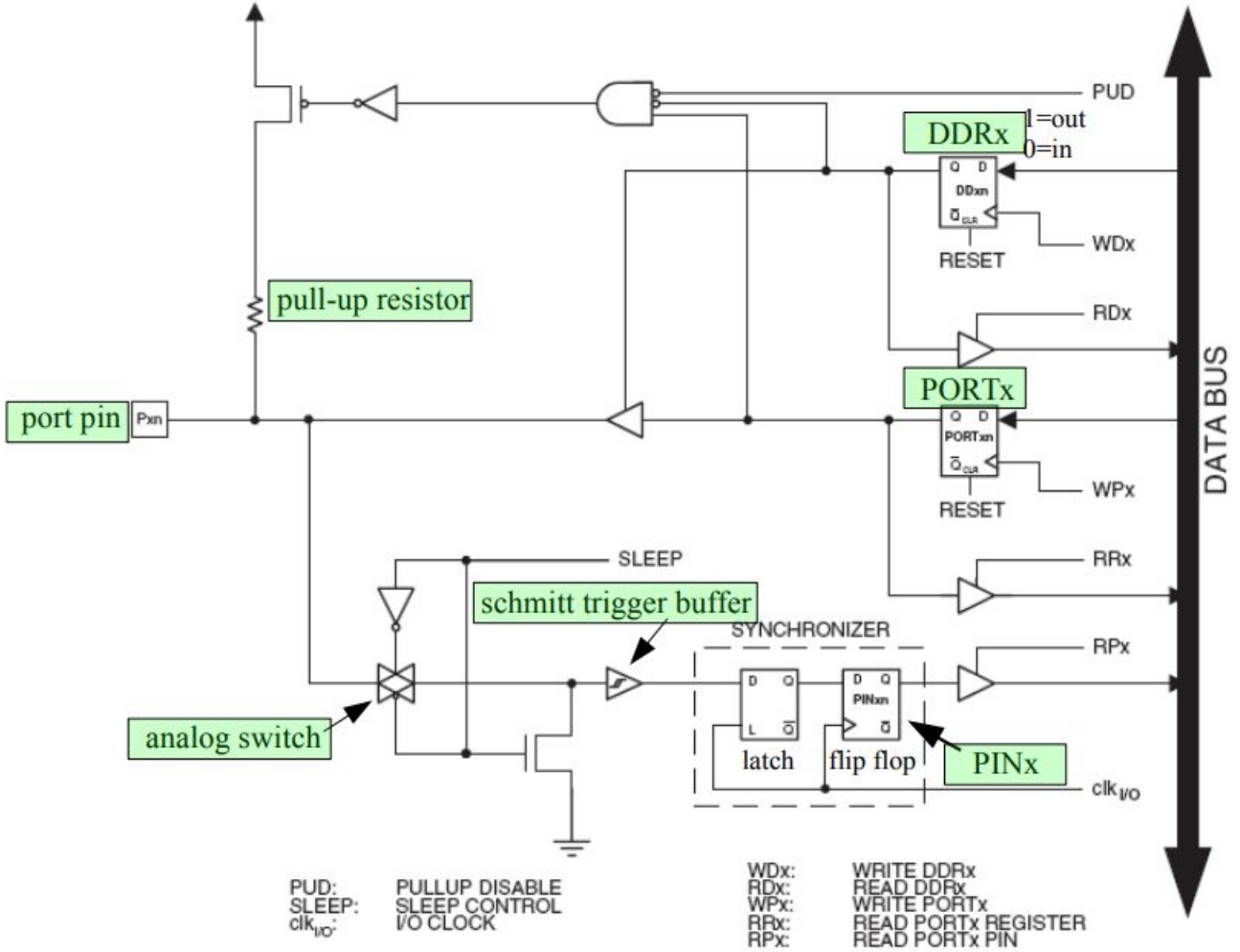




I/OPortin AVR microcontrollers



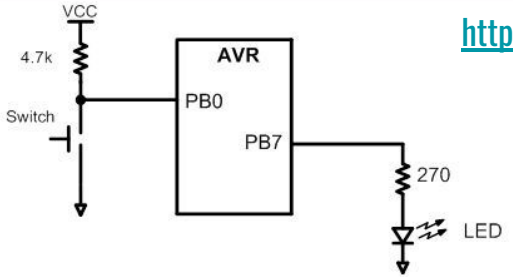
Pull-up Resistor


PUD
1=out
0=in
DDRx
DDxn
WDx
RESET
RDx
pull-up resistor
PORTx
port pin
Pxn
PORTxn
WPx
RESET
DATA BUS
SLEEP
RRx
schmitt trigger buffer
SYNCHRONIZER
RPx
PINxn
analog switch
latch
flip flop
PINx
clk I/O
PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clk I/O: I/O CLOCK
WDx: WRITE DDRx
RDx: READ DDRx
WPx: WRITE PORTx
RRx: READ PORTx REGISTER
RPx: READ PORTx PIN

AVR Direccionamiento I/O por bits

http://www.rjhcoding.com/avr-asm-io.php

Mnemonic	Description
in	in from I/O location
out	out from I/O location
cbi	clear bit in I/O register
sbi	set bit in I/O register
sbic	skip if bit in I/O register cleared
sbis	skip if bit in I/O register set



SBI: Set bit in IO register
CBI: Clear bit in IO register

SBIC: Skip if bit in IO register cleared

if

```
; Switch conectado al pin PB0 y LED al pin PB7
; Leer el estado del switch y enviarlo al LED.

       CBI DDRB,0    ; PB0 entrada
       SBI DDRB,7    ; PB7 salida
AGAIN: SBIC PINB,0   ; skip next if switch clear
       JMP ON
OFF:   CBI  PORTB,7  ; apagar LED
       JMP AGAIN
ON:    SBI PORTB,7   ; prender LED
       JMP AGAIN
```

while

```
wait:  sbic PINB,PINB0        ; skip if PINB0 is low
       rjmp wait              ; repeat loop
```

AVR Assembly Tutorials

http://www.rjhcoding.com/avr-asm-tutorials.php

AVR oficial

https://microchipdeveloper.com/8avr:start

Interrupciones

Dentro del ciclo de instrucción hay que verificar

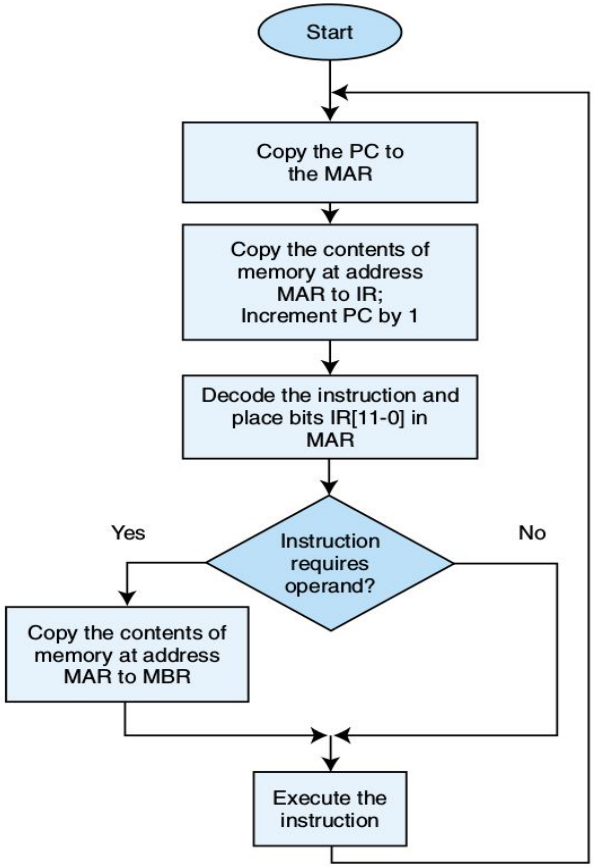


FIGURE 4.11 **The Fetch-Decode-Execute Cycle**

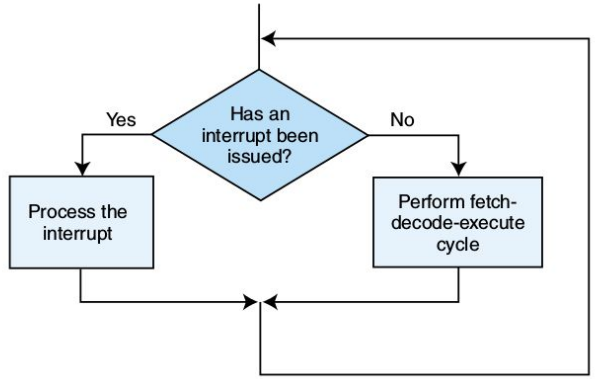
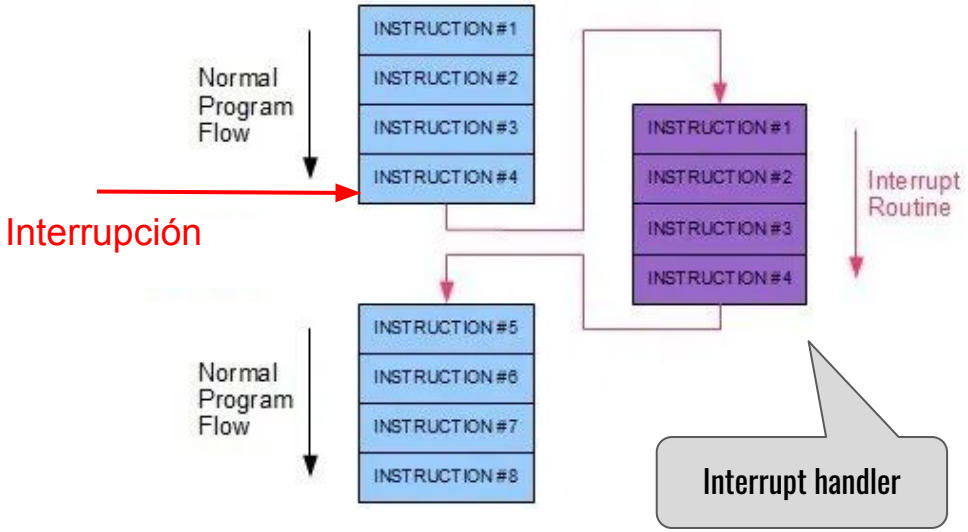


FIGURE 4.12 **Modified Instruction Cycle to Check for Interrupt**

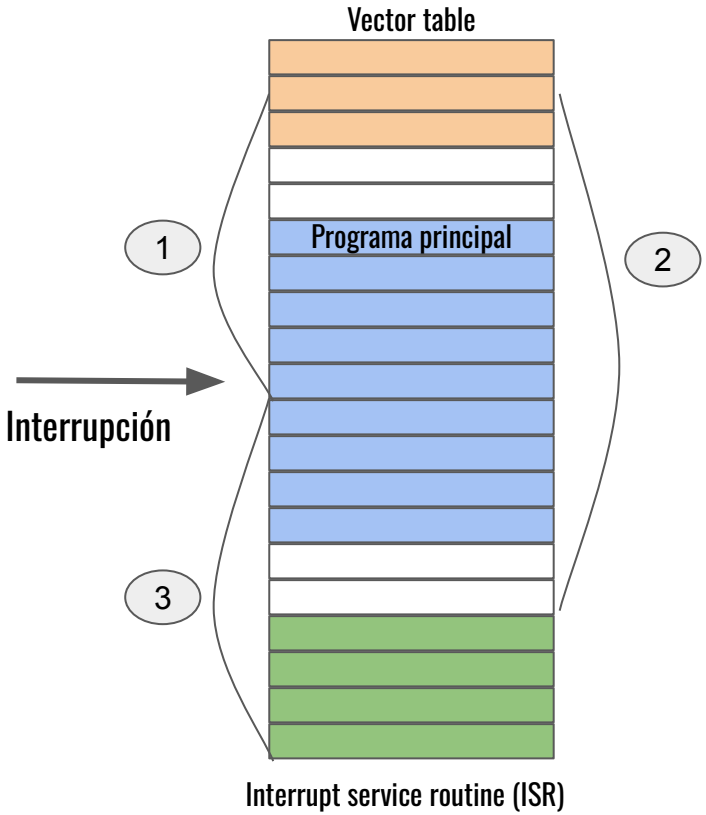
Interrupciones



Fuentes de interrupciones
Entrada/salida, ejemplo teclado
Excepciones (div por cero, overflow)
Timer interno

Fuentes de RESET
External reset (RESET pin)
Power-on reset (POR) circuit
COP watchdog
Illegal opcode reset
Illegal address reset
Low voltage inhibit (LVI) reset

Interrupciones



Vector table - RTI (return from interrupt)
Prioridades - Enmascarado - No anidado (nesting)
Detalles de implementación dentro del ciclo de instrucción
Cómo volver al estado original?
Context switch - Stack

Interrupciones

1. Detección de la interrupción
2. Arbitraje (evaluación de la prioridad)
3. Stacking del estado de la máquina. Los registros internos son salvados en la pila en un determinado orden y recuperados en el orden inverso. VER
4. Captación del vector correspondiente

Reset

1. Detección del reset (sincrónico o asincrónico)
2. Captación del vector de reset

RTI

1. Unstacking, incluye CCR, enable
2. Último el PC, retorno

AVR interrupt vectors

https://microchipdeveloper.com/8avr:int

16.1. Interrupt Vectors in ATmega328/P

Table 16-1. Reset and Interrupt Vectors in ATmega328/P

Vector No	Program Address(2)	Source	Interrupts definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event

SREG – AVR Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	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	

INTERRUPT ENABLE
STATUS: Status Register I : Global Interrupt Enable

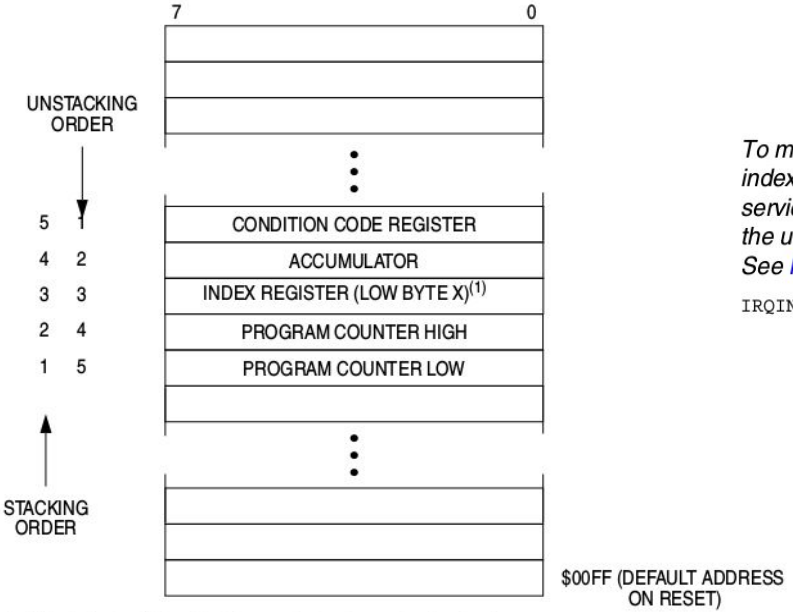
SECUENCIA AVR

- When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled.
- The interrupt vector directs program control to the proper ISR or execution.
- That ISR can write logic one to the I-bit to enable nested interrupts.
- All enabled interrupts can then interrupt the current interrupt routine.
- When the ISR is completed and the return (RETI) command is executed from the ISR, the Global I-bit is automatically set to 'ON' and the program execution returns to the main program at the instruction that was interrupted.

TIEMPO DE RESPUESTA AVR
Ver manual

CPU08 interrupt stack

Orden de apilado: PC, X, A y CCR



1. High byte (H) of index register is not stacked.

Figure 3-1. Interrupt Stack Frame

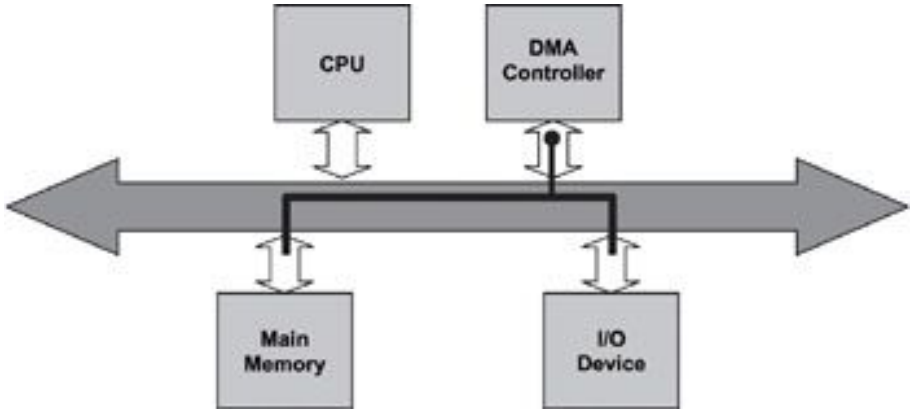
NOTE

To maintain compatibility with the M6805 Family, H (the high byte of the index register) is not stacked during interrupt processing. If the interrupt service routine modifies H or uses the indexed addressing mode, it is the user's responsibility to save and restore it prior to returning.
See Figure 3-2.

```
IRQINT      PSHH
            |
            |Interrupt service routine
            |
            |
            PULH
            RTI
```

Figure 3-2. H Register Storage

DMA (direct memory access)



El controlador de acceso directo a memoria provee un camino alternativo (datapath) entre los dispositivos de entrada/salida y la memoria. El procesador configura la operación de transferencia de un bloque, indicando la dirección fuente, la dirección destino y el número de bytes.

Una vez iniciada, la transferencia no ocupa ciclos de CPU pero sí del bus.

Ver detalles de la implementación en cada caso particular.