



Universidad autónoma de baja California

Ingeniería en computación

Microcontroladores

Practica 3 delay por software

Erik garcia Chávez 01275973

Jesús Adán Garcia López

28 de febrero del 2025

Ecuaciones para los retardos de 103uS, 1mS y 1S:

103uS:

Ecuación:

$$14 + 4x + 3xyz + xy(2z-1) + 4xy + x(2y-1) + 2x + (2x-1) \rightarrow 7x + 5xy + 5xyz + 13 = 1648$$

$$\rightarrow x(7 + 5y + 5yz) + 13 = 1648, \text{ donde } X = 5, Y = 8 \text{ y } Z = 7$$

$$7(5) + 5(5 \cdot 8) + 5(5 \cdot 8 \cdot 7) = 1648 //$$

Formulación de la ecuación:

retardo_103uS:

;4 ciclos rcall

clr r20 ;1

clr r21 ; 1

clr r22 ;1

ldi r22,5 ;-> 1 -> x

ldi r21, 7

;-----

;5 ciclos total

nxt0:

nop ; ->1x

nop; 1x

nop ;1x

ldi r20, 8 ;1x

;-----

;4x

nxt1:

dec r21 ;1zyx

nop ;1zyx

```

nop ;1xyz
brne nxt1 ;  $xy(2z-1)$ 
;-----
;  $3xyz + xy(2z-1)$ 

```

nxt2:

```

ldi r21,7 ;-> 1xy
nop ; 1xy
nop ;1xy
dec r20      ;-> 1xy
brne nxt1 ;  $x(2y-1)$ 
;-----
;  $4xy + x(2y-1)$ 
dec r22 ;x -> 1x
nop ; 1x
brne nxt0 ;->  $(2x-1)$ 
ret ;5 ciclos
;-----
;  $2x+(2x-1)+5$ 
;-----
;  $7x + 5xy + 5xyz + 13$ 

```

Retardo 1mS:

$$4 + 5 + 2 + 2x + 1xy + 1xyz + xy(2z-1) + 1xy + x(2y-1) + 1x + (2x-1) \rightarrow$$

$$4x + 3xy + 3xyz + 10 = 16000 \rightarrow x(4 + 3y + 3yz) + 10 = 16,000$$

$$x \rightarrow 15, y \rightarrow 59, z \rightarrow 5$$

$$4(15) + 3(15 \cdot 59) + 3(15 \cdot 59 \cdot 5) + 10 = 16,000$$

Formulación de la ecuación:

retardo_1mS:

; rcall \rightarrow 4 ciclos

; para un retardo de 1mS es necesario 16,000 ticks

; esto porque $(1 \times 10^{-3}) \cdot (16 \times 10^6) \rightarrow 16,000$

ldi r20, 15 ; 1

nop ; 1

nxt_mS:

; este es el ciclo superior a todos

ldi r21, 59 ; 1x

nop ; 1x

nxt1_mS

ldi r22, 5 ; 1x

nxt2_mS:

; este es el ciclo mas pequeno. en el ciclo

dec r22 ; 1xyz

brne nxt2_mS ; $xy(2z-1)$

dec r21 ; 1xy

brne nxt1_mS ; $x(2y-1)$

```

dec r20 ; 1x
brne nxt_mS ; 2x-1
ret ; final del delay

```

retardo de 1S:

$$1 + 4 + 5 + 4x + 3xy + 1xyz + xy(2z-1) + xy + x(2y-1) + x + (2x-1)$$

$$6x + 5xy + 3xyz + 9 = 16,000,000$$

$$X = 241, Y = 144, Z = 15$$

$$6(241) = 5(241 * 144) + x(241 * 144 * 152) + 9 = 16,000,000$$

Formulación de ecuación:

retardo_1S:

```

ldi r20, 241;           /1
    nxt_S: ldi r21, 144;   /x
    nop;                  /x
    nop;                  /x
    nop;                  /x
        nxt2_S: ldi r22, 152; /xy
        nop;              /xy
        nop;              /xy
            nxt3_S: dec r22; /xyz
            brne nxt3_S;    /xy(2z-1)
        dec r21;           /xy
        brne nxt2_S;       /x(2y-1)
    dec r20;               /x
    brne nxt_S;            /2x-1

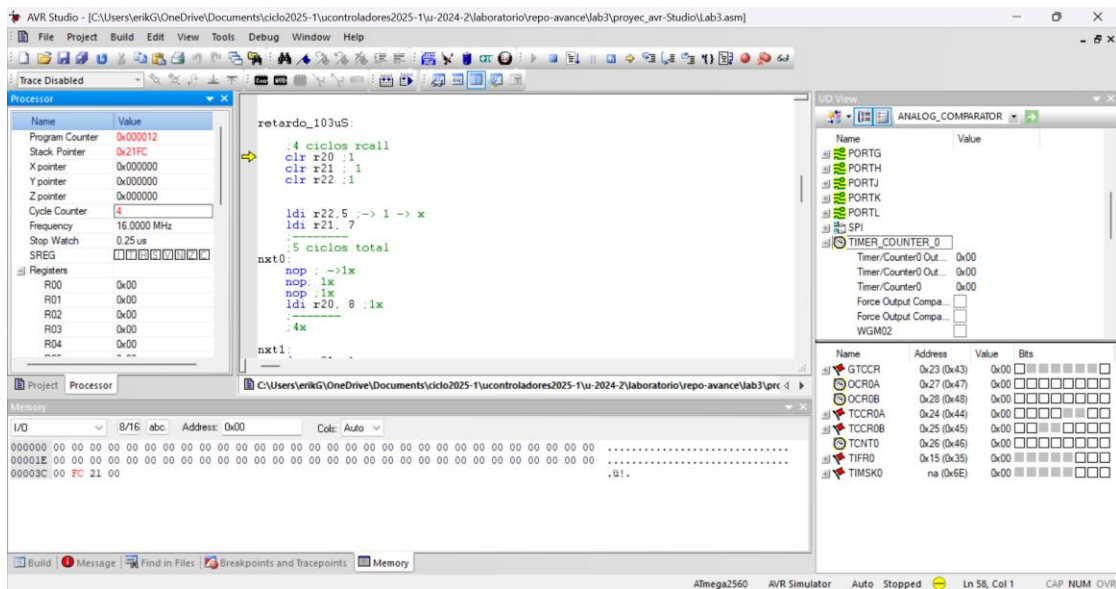
```

ret

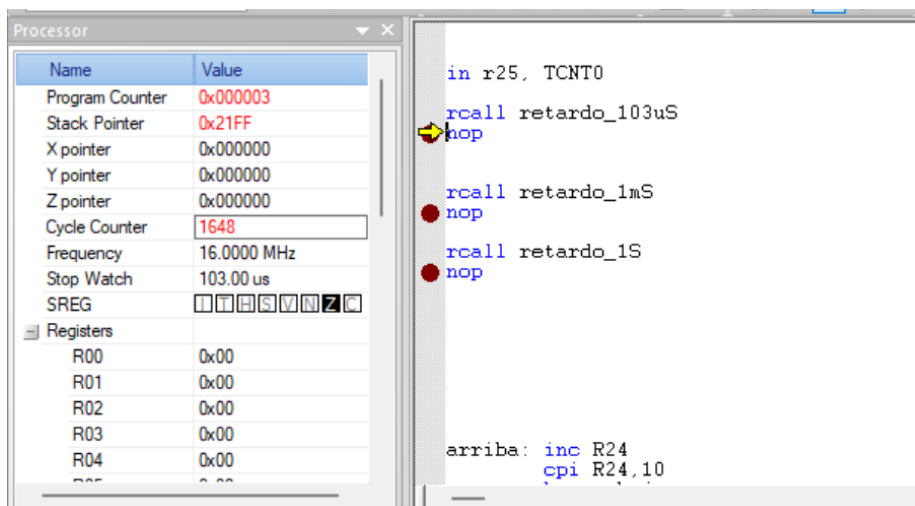
implementacion:

103uS:

Inicio, entrando a la subrutina:

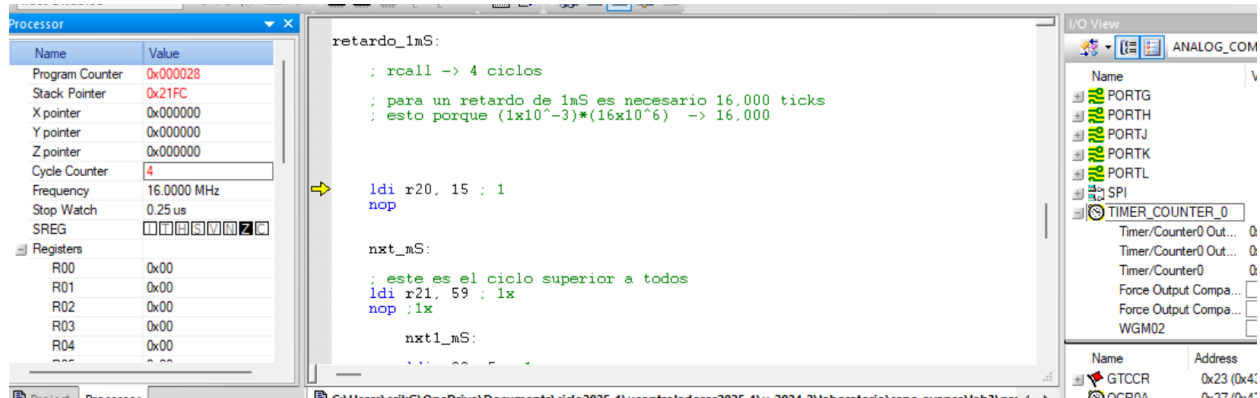


Finalización:

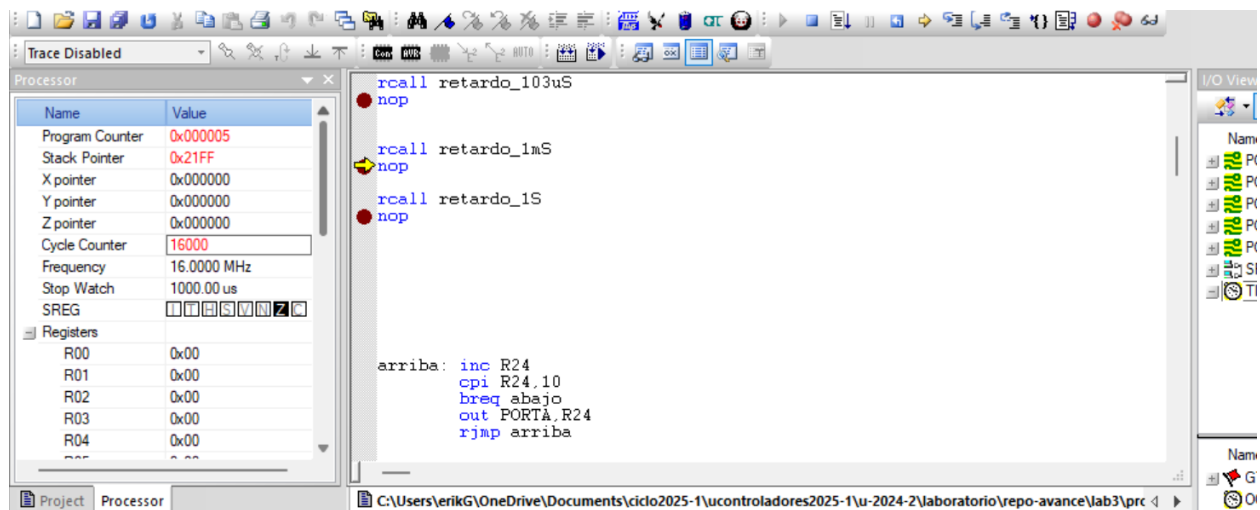


Implementación 1mS:

Inicio:



Finalizacion del delay:



Delay de 1S:

inicializacion

The screenshot shows the AVR Studio interface with the 'Processor' window on the left and the assembly code editor on the right. The 'Processor' window displays the following values:

Name	Value
Program Counter	0x000034
Stack Pointer	0x21FC
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	4
Frequency	16.0000 MHz
Stop Watch	0.25 us
SREG	0x00
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00

The assembly code in the editor is as follows:

```
retardo_1S:
    ldi r20, 241;          /1
    nxt_S: ldi r21, 144;   /x
    nop;                  /x
    nop;                  /x
    nop;                  /x
    nxt2_S: ldi r22, 152;  /xy
    nop;                  /xy
    nop;                  /xy
    nxt3_S: dec r22;       /xyz
    brne nxt3_S;          /xy(2z-1)
    dec r21;              /xy
    brne nxt2_S;          /x(2y-1)
    dec r20;              /x
    brne nxt_S;           /2x-1

    ret
```

Finalización:

The screenshot shows the AVR Studio interface with the 'Processor' window on the left and the assembly code editor on the right. The 'Processor' window displays the following values:

Name	Value
Program Counter	0x000007
Stack Pointer	0x21FF
X pointer	0x000000
Y pointer	0x000000
Z pointer	0x000000
Cycle Counter	16000000
Frequency	16.0000 MHz
Stop Watch	1000000.00 us
SREG	0x00
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00

The assembly code in the editor is as follows:

```
rcall retardo_103uS
nop

rcall retardo_1mS
nop

rcall retardo_1S
nop

arriba: inc R24
        cpi R24, 10
        breq abajo
        out PORTA, R24
        rjmp arriba
```


Generador del número aleatorio: generación congruencial lineal simple

Este genreador responde a la siguiente propiedad:

Generador congruencial lineal simple [\[editar \]](#)

Dada una semilla x_0 , un multiplicador a , una constante c llamada incremento y un módulo m , se define

$$x_i = (ax_{i-1} + c) \bmod m$$

Donde:

X_n = el valor actual de la secuencia

a = el multiplicador

c = es el incremento

m = es el modulo

X_{n+1} = el siguiente valor de la secuencia:

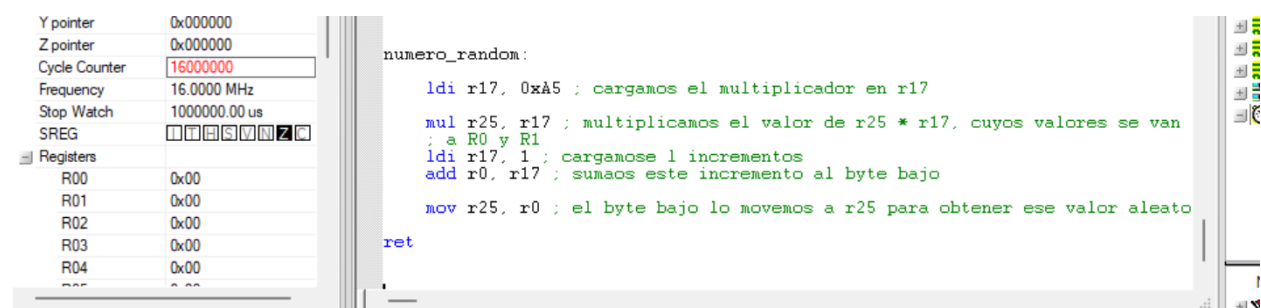
Para implementarlo en ASM para un numero de 8 bits debemos establecer los siguiente parámetros

$M= 256$ dado que se estará trabajando con números de 8 bits,

A y C , se establecen en 165 y 1 respectivamente dado que estos son valores que se dan para tener la mayor distribución de numero aleatorios el siguiente que se puede usar es demasiado grande para ser almacenado en un registro de 8 bits.

X_0 , este valor es la semilla, el cual puede ser obtenido de varias formas, en el caso que se implemento utilizamos el valor que hay en el timer0, cuando se simula este código es necesario introducir el valor manualmente al timer para que este pueda generar los valores aleatorios dado que en la simulación todos empiezan en 0, en físico esto no será así porque este iniciara en un valor X .

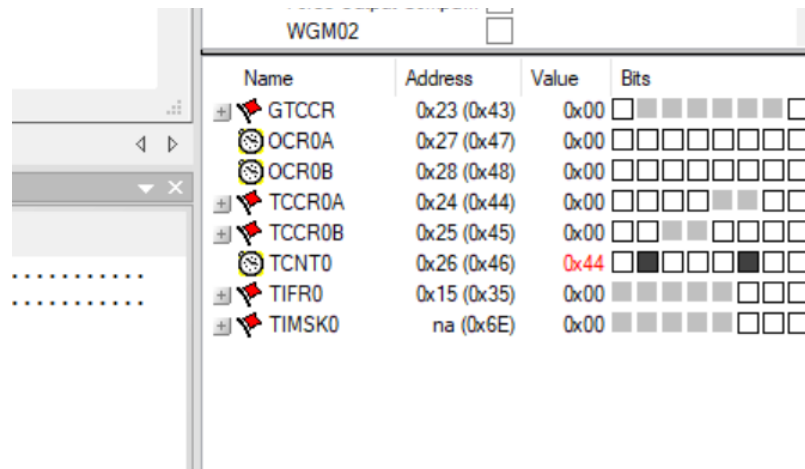
Implementación:



```
numero_random:
    ldi r17, 0xA5 ; cargamos el multiplicador en r17
    mul r25, r17 ; multiplicamos el valor de r25 * r17, cuyos valores se van
                  ; a R0 y R1
    ldi r17, 1 ; cargamos 1 incrementos
    add r0, r17 ; sumamos este incremento al byte bajo
    mov r25, r0 ; el byte bajo lo movemos a r25 para obtener ese valor aleato
    ret
```

Ejemplo:

Ingresamos un valor X al timer0:

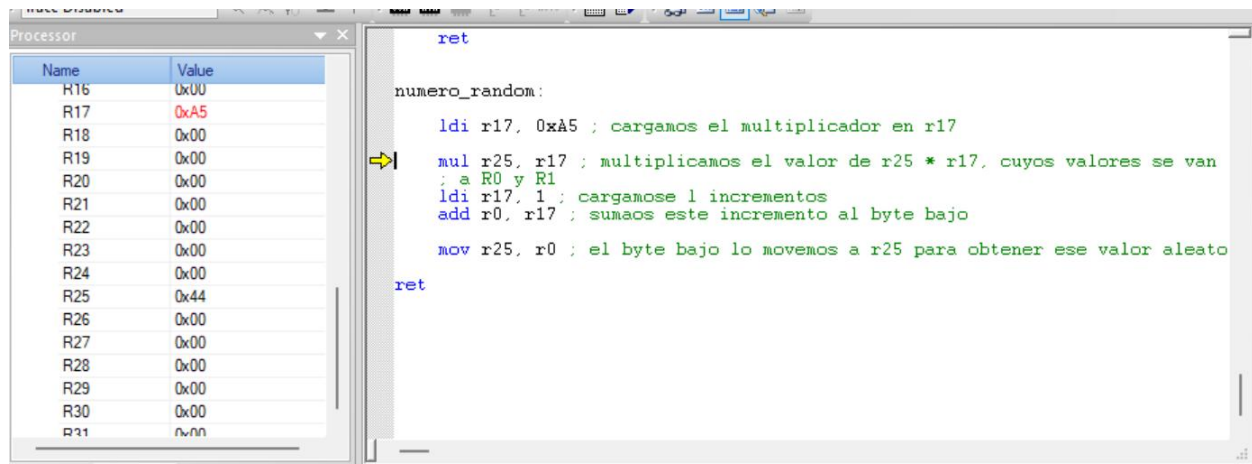


Name	Address	Value	Bits
GTCCR	0x23 (0x43)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
OCR0A	0x27 (0x47)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
OCR0B	0x28 (0x48)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
TCCR0A	0x24 (0x44)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
TCCR0B	0x25 (0x45)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
TCNT0	0x26 (0x46)	0x44	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
TIFR0	0x15 (0x35)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
TIMSK0	na (0x6E)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

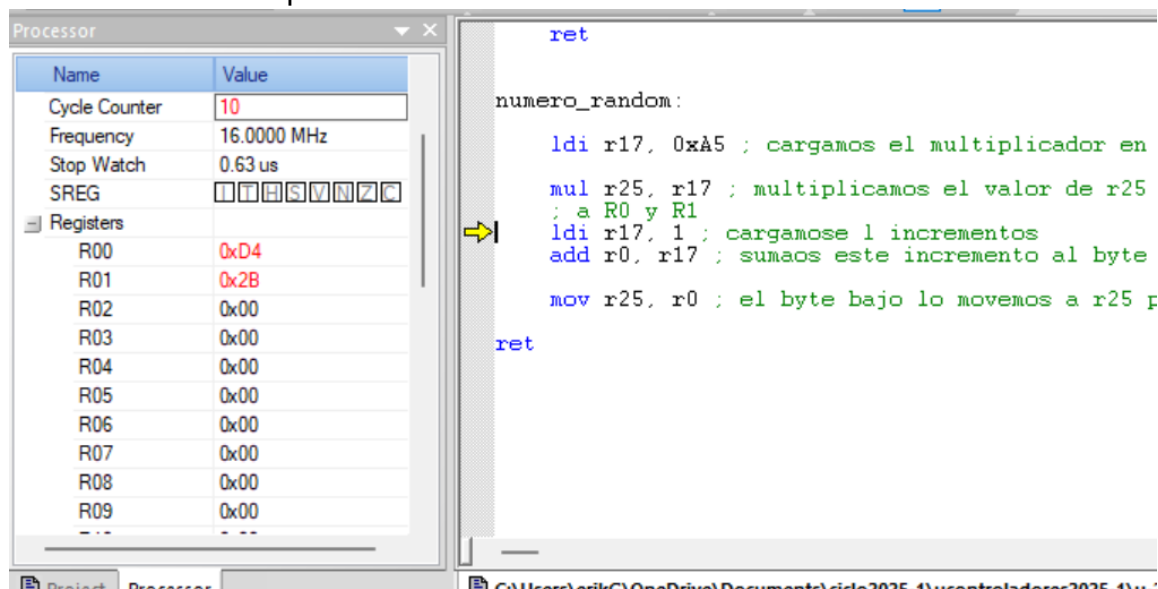
La semilla se cargo a R25:

Name	Value
R15	0x00
R16	0x00
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x44
R26	0x00
R27	0x00
R28	0x00
R29	0x00
R30	0x00

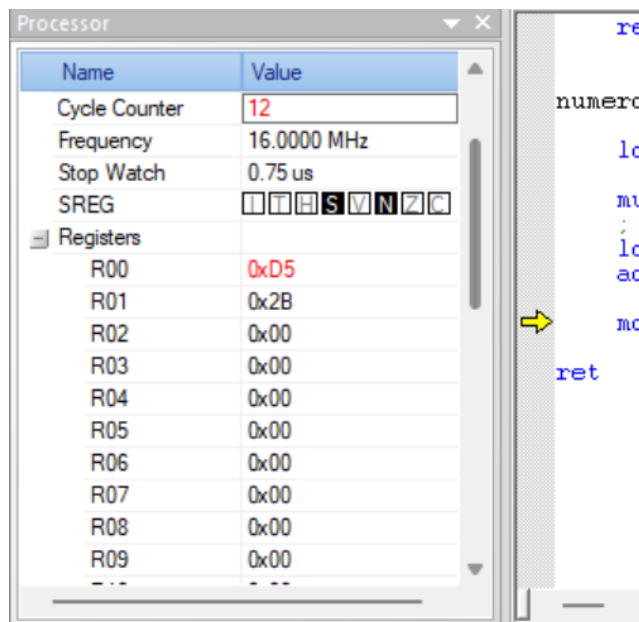
Se carga el 165(0xA5) a R17 que es nuestro multiplicador



Realizamos la multiplicación:



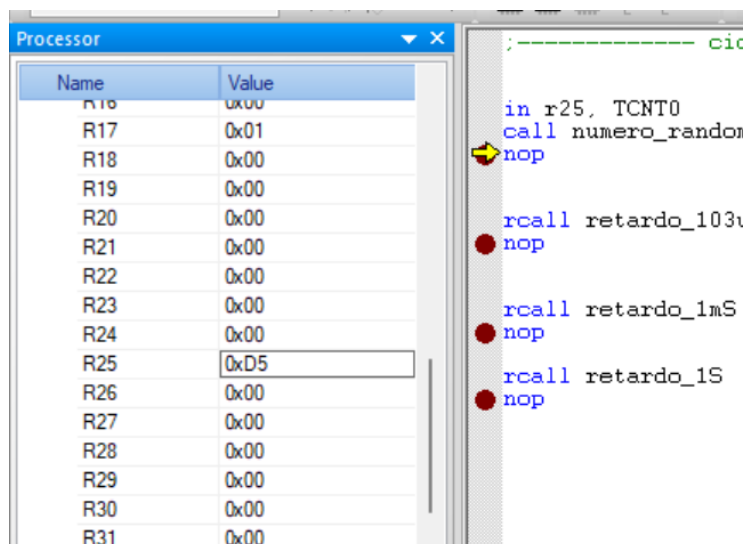
Le sumamos el incremento a R0 que seria la parte baja de la multiplicación:



The screenshot shows the AVR Processor window. On the left, the 'Registers' section lists R00 through R09. R00 has a value of 0xD5, which is highlighted in red. To the right, the assembly code is visible, showing instructions like 're', 'numero', 'ld', 'mu', 'ld', 'ad', 'mo', and 'ret'. A yellow arrow points to the 'mo' instruction.

Name	Value
Cycle Counter	12
Frequency	16.0000 MHz
Stop Watch	0.75 us
SREG	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>
Registers	
R00	0xD5
R01	0x2B
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00

R25 ahora tiene el numero aleatorio que se generó:



The screenshot shows the AVR Processor window. On the left, the 'Registers' section lists R16 through R31. R25 has a value of 0xD5, which is highlighted in red. To the right, the assembly code is visible, showing instructions like 'in r25, TCNT0', 'call numero_random', 'nop', 'rcall retardo_103', 'nop', 'rcall retardo_1mS', 'nop', 'rcall retardo_1S', and 'nop'. A yellow arrow points to the first 'nop' instruction.

Name	Value
R16	0x00
R17	0x01
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0xD5
R26	0x00
R27	0x00
R28	0x00
R29	0x00
R30	0x00
R31	0x00

Conclusiones:

En la practica pudimos poner en practica los retardos por software, todo lo que conlleva el realizar uno de cierta cantidad de tiempo, lo cual es un ciclo que lo que hace es quemar tiempo. Ticks, para cumplir con su objetivo, asi como poder crear una subrutina que crea un pseudo numero aleatorio, hay muchas formas de hacerlo, en mi caso use la formula el cual lleva el nombre de generación congruencial lineal simple, usando como semilla el registro del timer0. Este fue el que mas se me complico dado que no sabia como representar la formula en ASM.