Θεόδωρος Μπάρκας 2016030050 LAB3

Σκοπός άσκησης

Στη συγκεκριμένη άσκηση σκοπός ήταν η εξοικείωση με τη θυρα USART του ενσωματωμένου συστήματος ATMEGA16 και γενικότερα των AVR microcontroller.Πιο συγκεκριμένα να υλοποιήσουμε μια λειτουργία παρόμοια με την εγγραφή ψηφίων στο κομπιουτεράκι.Όπου ο χρήστης στέλνει δεδομένα μέσω της θύρας USART με Baud Rate = 9600Hz και εμείς του απαντάμε με OK.

Σκοπός άσκησης

Αρχικά αρχικοποιήσαμε τη θύρα USART ως εξης:

```
;USART FUNCTIONS
USART_Init:
; Set baud rate
ldi r16,0x00
out UBRRH, r17
ldi r16,0x40
out UBRRL, r16
ldi r16, 0x00
out UCSRA,r16
; Enable receiver and transmitter AND enaBLE interrupts for them
ldi r16, (1<<RXCIE)|(0<<UDRIE)|(1<<RXEN)|(1<<TXEN)
out UCSRB,r16
; Set frame format: 8data, 1stop bit,8 char
ldi r16, (0<<URSEL)|(0<<USBS)|(1<<UCSZ1)|(1<<UCSZ0)
out UCSRC,r16
ret
```

Το UBRR προέκυψε από τον τύπο του MANUAL του ATMEGA16(σελ.147)

Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$

UBRR=64.104666..

Το οποιο αν στρογγυλοποιιθει στον κοντινοτερο ακεραιο $UBRR_{Closest\ Match}$=64=0x0040 δινει ενα ERROR%=-0.1625% (σελ.168) , το οποιο ειναι αμελητεο καθως η USART επιτρεπει ακομα και μεγαλυτερα ERRORS χωρις κανενα προβλημα.

$$Error[\%] = \left(\frac{BaudRate_{Closest\ Match}}{BaudRate} - 1\right) \bullet 100\%$$

Επίσης αξιοσημείωτο είναι οτι απενεργοποιήσαμε τα flags UDRIE καθώς θα κρέμαγε η συνάρτηση στο TESTING (Λόγω του οτι αν κάνουμε OUT δεν θα λάβει κανείς το μήνυμα και δεν θα ολοκληρωθεί το Transittion), καθώς και για το λόγω ότι χωρίς μεγάλη χρονική – επεξεργαστική σπατάλη μπορούμε να διαχειριστούμε το Transmition του OK με Polling.

USART RECEIVE και μηχανή πεπερασμένων καταστάσεων

Αρχικά ορίζουμε μια θέση μνήμης στην οποία θα αποθηκεύεται το State της μηχανής καταστάσεων καθώς και τιμές με .equ για ευκολία στην ανάγνωση και στο Debugging.

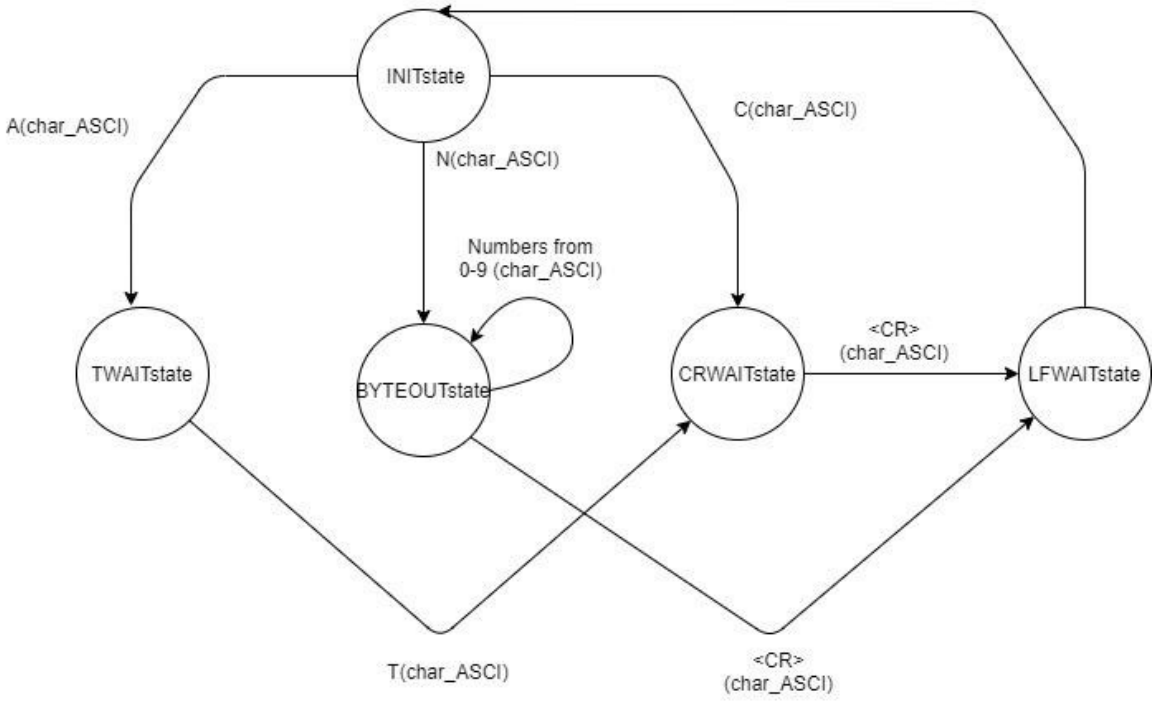
```
.equ    INITstate       = 0x00
.equ    TWAITstate      = 0x01
.equ    BYTEOUTstate    = 0x02
.equ    CRWAITstate     = 0x03
.equ    LFWAITstate     = 0x04
```

```
.equ ControlUsart =0x0086
```

Memory location Of Current State

STATES

Ύστερα υλοποιούμε τον κώδικα ο οποίος προκύπτει από το εξής διάγραμμα της μηχανής πεπερασμένων καταστάσεων:



Κώδικας Σχόλια

- Αρχικά ο κώδικας έχει ένα σύνολο από branches που στέλνει ανάλογα με το τωρινό State στο κατάλληλο Handler-Label τη συνάρτηση.
-
```
;STATE MANAGMENT
;Jump To The Correct State Handler--
cpi r20,INITstate ;INIT
breq INIT
cpi r20,TWAITstate ;T   Just waiting for T in this state
breq TWAIT
cpi r20,BYTEOUTstate ;ByteOut
breq BYTEOUT
cpi r20,CRWAITstate ;<CR>
breq CRWAIT
cpi r20,LFWAITstate ;<CR> when you are at <CR> you wait for <LF> print Ok and then make the state==INIT
breq LFWAIT
;end-------------------------------
```
  πεπερασμένων καταστάσεων τα STATES στους Handlers που προαναφέρθηκαν φαίνονται στο κώδικα παρακάτω.
- Αξίζει να σημειωθεί οτι στο Return1: label κάνουμε εγγραφή στην θέση μνήμης X=ControlUsart τον r20 που είναι το nextState που μας δόθηκε από την επεξεργασία του τωρινού State και της εισόδου από την πύλη USART. Ύστερα αρχικοποιούμε τον καταχωρητή X(δηλαδή τον αφήνουμε στη τιμή που είχε πριν γίνει το Interrupt) και κάνουμε Return.

```
return1:
ldi XL,ControlUsart
st X,r20 ;Store the new State to Memory
ldi XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
reti
```

- Η `USART_Transmit_OK` Χρησιμοποιώντας την τεχνική Polling (μπλοκάροντας δηλαδή τον επεξεργαστή ,δηλαδή αφήνοντας τον να τρέχει σε loop και περιμένοντας το Interrupt Flag).Στέλνει Ok στον χρήστη. Σημαντικό σε αυτήν την υλοποίηση είναι η αντικατάσταση του `out UDR, r18` με `out TCNT2, r16 ; replaced` για λόγους Simulation.

Πιο συγκεκριμένα (και έχει αναφερθεί και παραπάνω)δεν μπορούμε να στείλουμε πληροφορία σε Simulation Environment επομένως βγάζουμε την έξοδο στον TCNT2 απλά για να μπορούμε να την παρατηρήσουμε πιο εύκολα στο Simulation.

```
USART_Transmit_OK:
; Wait for data to be transmitted
sbis UCSRA, UDRE
rjmp USART_Transmit_OK
; Transmit data
ldi r16,0x4F ;r16=O (from OK)
;out UDR, r18
out TCNT2, r16 ; replaced
USART_Transmit_K:
sbis UCSRA, UDRE
rjmp USART_Transmit_K
ldi r16,0x4B ;r16=K (from OK)
;out UDR, r18
out TCNT2, r16 ; replaced
ret
```

Επίσης αξίζει να σημειωθούν και οι δυο συναρτήσεις :

- `clearAllBytes` Όπου χρησιμοποιούμε για τον καθαρισμό της οθόνης που αντιστοιχεί σε είσοδο από τη θύρα USART του χαρακτήρα ASCII: C, N.
- `ShiftAllBytesRight` Όπου χρησιμοποιείτε στο State: BYTESOUTstate αν η είσοδος από την πύλη USART είναι αριθμός και όχι το <CR>.

```
;Before you use this function keep your X Register In Memory
;r18 The Inserted Byte -> From USART for this exercise
shiftAllBytesRight:
    ldi XL,LOW(SRAM_START)      ; initialize X pointer
    ldi XH,HIGH(SRAM_START)
    ldi r20,0x08
    add r20,XL ;R20=XL+8
    ;whileLoop With init start
    ;ldi r18,0x01
    ;ld r18,x ;Init -> The first time in while Loop r16 get the right number Inserted Number
    whileLoop2:
    cp XL,r20 ;gives Zflag = 1 when r20-0x08=0 ;
    breq if2 ;escape while loop
        mov r16,r18     ;r16=r18=last Byte not Yet Shifted
        inc xl          ;We save next Byte
        ld r18,-x       ;we save to r18 for next loop and then go back in ram to
        st x+,r16       ;store the new value also increrment x for next loop x=x+1
        rjmp whileLoop2
    if2:
    ;whileLoop End
ret


;Before you use this function keep your X Register In Memory
clearAllBytes:
    ldi XL,LOW(SRAM_START)      ; initialize X pointer
    ldi XH,HIGH(SRAM_START)
    ldi r18,0x08
    add r18,XL ;R18=XL+8
    ;whileLoop With init start
    whileLoop3:
    cp XL,r18 ;gives Zflag = 1 when r18-0x08=0 ;
    breq if3                ;escape while loop
        ldi r16,0x0A        ;r16=0x0A=CLOSED SEGMENTS /NO LIGHT
        st x+,r16           ;store the new value also increment x for next loop x=x+1
        rjmp whileLoop3
    if3:
    ;whileLoop End
    ret
```

Κώδικας

Παραθέτετε για λόγους διευκόλυνσης καθώς το Project έχει πολλές γραμμές κώδικα γραμμένες από προηγούμενα εργαστήρια.

Κωδικας USART_RECEIVE

Εδώ έχουμε όλων τον Interrupt Handler κώδικα στον οποίο βλέπουμε λεπτομερείς για τον τρόπο λειτουργίας της μηχανής πεπερασμένων καταστάσεων.

```
;USART RECEIVE
;------------------------
;Receive in r18 the value Of UDR
USART_Receive:
;STATE MACHINE
.equ   INITstate            = 0x00
.equ   TWAITstate           = 0x01
.equ   BYTEOUTstate  = 0x02
.equ   CRWAITstate          = 0x03
.equ   LFWAITstate          = 0x04
;FinishOperation;0x01->CR;0x02->LFBeforeOk
       ldi XL,ControlUsart
       ;REMEMBER that r17,r19 In Use BURNED
       in r18, UDR ;->REAL CODE
       in r18, UDR ;->REAL CODE
       mov r18,r15    ;->`TESTING CODE
       ld r20,X ;r20 Is the register with the state

       ;STATE MANAGMENT
       ;Jump To The Correct State Handler--
       cpi r20,INITstate ;INIT
       breq INIT
       cpi r20,TWAITstate ;T      Just waiting for T in this state
       breq TWAIT
       cpi r20,BYTEOUTstate ;ByteOut
       breq BYTEOUT
       cpi r20,CRWAITstate ;<CR>
       breq CRWAIT
       cpi r20,LFWAITstate ;<CR> when you are at <CR> you wait for <LF> print Ok and then make
                           the state==INIT
       breq LFWAIT
       ;end------------------------------

       ;Handlers=function(r18(char se ascii),r20(currentState))
       INIT:
       cpi r18,0x41;r18==A
       breq controlif1
       cpi r18,0x4E;r18==N
       breq controlif2
       cpi r18,0x43;r18==C
       breq controlif3
       controlif1:;r18==A
              ldi r20,TWAITstate ;nextsState=TWAITstate
              rjmp return1
       controlif2:;r18==N
              rcall clearAllBytes ;uses r18 but we dont use it from now on
              ldi r20,BYTEOUTstate ;nextsState=BYTEOUTstate
              rjmp return1
       controlif3:;r18==C
```

```
        rcall clearAllBytes ;uses r18 but we dont use it from now on
        ldi r20,CRWAITstate ;CRWAITstate
        rjmp return1
;Handler(r18(char se ascii),r20(currentState))
rjmp return1
TWAIT:
        ldi r20,CRWAITstate ;<CR>
;Handler(r18(char se ascii),r20(currentState))
rjmp return1
BYTEOUT:
cpi r18,0x0D;If inserted character = <CR>
breq controlif4
        andi r18,0b00001111 ;Setting Mask to make the number from ASCII to BSD
                                        ;r18 is the BSD number
        rcall shiftAllBytesRight    ;This Function Changes r20 and r18 /r18 doesn't matter
                                    to us
        ldi r20,BYTEOUTstate        ;But We Want to keep r20(STATE) until we save it to
                                    RAM
rjmp return1
controlif4:;IF CR is imported
        ldi r20,LFWAITstate
rjmp return1
CRWAIT:
        ldi r20,LFWAITstate ;<CR>
rjmp return1
LFWAIT:
        ldi r21,0xFF
        ldi r20,INITstate ;<CR>
        ;MUST PRINT OK
        rcall USART_Transmit_OK
rjmp return1

return1:
ldi XL,ControlUsart
st X,r20 ;Store the new State to Memory
ldi    XL, LOW(SRAM_START) ;getting the X pointer where it was before decoder block
add XL,r17
reti
```

USART TRANSMITTER και μηχανή πεπερασμένων καταστάσεων

```
USART_Transmit_OK:
; Wait for data to be transmitted
sbis UCSRA, UDRE
rjmp USART_Transmit_OK
; Transmit data
ldi r16,0x4F ;r16=O (from OK)
;out UDR, r18
out TCNT2, r16 ; replaced
USART_Transmit_K:
sbis UCSRA, UDRE
rjmp USART_Transmit_K
ldi r16,0x4B ;r16=K (from OK)
;out UDR, r18
out TCNT2, r16 ; replaced

USART_Transmit_CR:
sbis UCSRA, UDRE
rjmp USART_Transmit_CR
ldi r16,0x0D ;r16=<CR>
```

```
	;out UDR, r18
	out TCNT2, r16 ; replaced
	USART_Transmit_LF:
	sbis UCSRA, UDRE
	rjmp USART_Transmit_LF
	ldi r16,0x0A ;r16=<LF>
	;out UDR, r18
	out TCNT2, r16 ; replaced
	ret
```

FUNCTIONS

```
;FUNCTIONS
;-----------------------------------------------------------------
	;Before you use this function keep your X Register In Memory
	;r18 The Inserted Byte -> From USART for this exercise
	shiftAllBytesRight:
		ldi	XL,LOW(SRAM_START)		; initialize X pointer
		ldi	XH,HIGH(SRAM_START)
		ldi r20,0x08
		add r20,XL ;R20=XL+8
		;whileLoop With init start
		;ldi r18,0x01
		;ld r18,x ;Init -> The first time in while Loop r16 get the right number Inserted
		Number
		whileLoop2:
		cp XL,r20 ;gives Zflag = 1 when r20-0x08=0 ;
		breq if2 ;escape while loop
			mov r16,r18		;r16=r18=last Byte not Yet Shifted
			inc xl			;We save next Byte
			ld r18,-x		;we save to r18 for next loop and then go back in ram
			to
			st x+,r16		;store the new value also increrment x for next loop
			x=x+1
			rjmp whileLoop2
		if2:
		;whileLoop End
	ret

	;Before you use this function keep your X Register In Memory
	clearAllBytes:
		ldi	XL,LOW(SRAM_START)		; initialize X pointer
		ldi	XH,HIGH(SRAM_START)
		ldi r18,0x08
		add r18,XL ;R18=XL+8
		;whileLoop With init start
		whileLoop3:
		cp XL,r18 ;gives Zflag = 1 when r18-0x08=0 ;
		breq if3			;escape while loop
			ldi r16,0x0A	;r16=0x0A=CLOSED SEGMENTS /NO LIGHT
			st x+,r16		;store the new value also increment x for next
			loop x=x+1
			rjmp whileLoop3
		if3:
		;whileLoop End
		ret
```

```
;USART FUNCTIONS
USART_Init:
; Set baud rate
ldi r16,0x00
out UBRRH, r17
ldi r16,0x40
out UBRRL, r16
ldi r16, 0x00
out UCSRA,r16
; Enable receiver and transmitter AND enaBLE interrupts for them
ldi r16, (1<<RXCIE)|(0<<UDRIE)|(1<<RXEN)|(1<<TXEN)
out UCSRB,r16
; Set frame format: 8data, 1stop bit,8 char
ldi r16, (0<<URSEL)|(0<<USBS)|(1<<UCSZ1)|(1<<UCSZ0)
out UCSRC,r16
ret
```

Testing

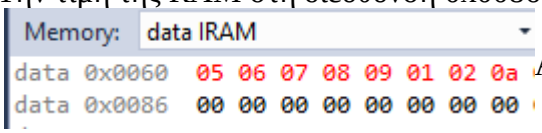
Βάλαμε ένα Break Point στον Counter και στο πρώτο interrupt που δόθηκε ενεργοποιήσαμε τον κώδικα Stimulifile.

Παρακατω θα παρατηρήσουμε τη συμπεριφορα του Microcontroller μετα απο την ενεργοποίηση του Stimulifile (Δηλαδη στα Interrupts οπου ομως το UDR προσομοιαζεται με το r15,παρολα αυτα πρεπει να γινει ανάλωση του χαρακτηρα UDR ωστε να κλεισει το Flag RXC (Ακριβως οπως φαίνεται στον κώδικα παραπανω)).

Συγκεκριμενα για να εξετάσουμε ολες τις διαφορετικες περιπτωσεις είχαμε 3 διαφορετικά Stimulifiles.

Στα παρακάτω τεστ αξίζει να προσέξουμε:

- την τιμή r15 η οποία προσομοιάζει το UDR και δείχνει την είσοδο που μόλις λάβαμε.
- Τις τιμες της RAM και πιο συγκεκριμένα τις τιμές σε διευθύνσεις από 0x0060 έως την τιμη 0x0067 που δείχνουν από σειρά χαμηλότερης προς υψηλότερης τα MOST significant Bytes.
- Την τιμή της RAM στη διεύθυνση 0x0086 όπου αποθηκεύεται το STATE της μηχανής.



Αρχικοποιημένες θέσεις

- Καθώς και τον Counter που μας λέει αν στάλθηκε ή όχι το OK στο State <LF>(Για το λόγο αυτό βάζουμε 4 Break points και στη συνάρτηση Polling : `USART_Transmit_OK` στα σημεία οπού αλλάζει ο Counter (Ο οποιος Counter προσομοιωνει το transmitted Value))
- Εκτός αυτού στα σημεία που μας έρχεται το <LF> ενδεικτικά θέτουμε και τον καταχωρητή r21 στην τιμή 0xFF : ldi r21,0xFF
- Επίσης προφανώς και θέτουμε ένα Break Point όταν γίνεται Return από το Interrupt έτσι ώστε σε κάθε βήμα να βλέπουμε τι Process έχει γίνει από τν FSM .
- **Επίσης θυμίζω τα States για Reference**

```
.equ    INITstate       = 0x00
.equ    TWAITstate      = 0x01
.equ    BYTEOUTstate    = 0x02
.equ    CRWAITstate     = 0x03
.equ    LFWAITstate     = 0x04
```

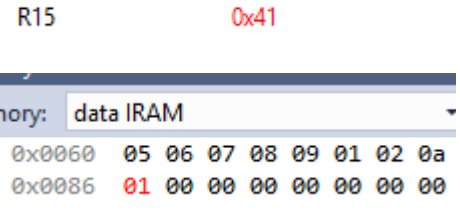
Περίπτωση 1

Είσοδος από USART:AT<CR><LF>

STIMULIFILE:

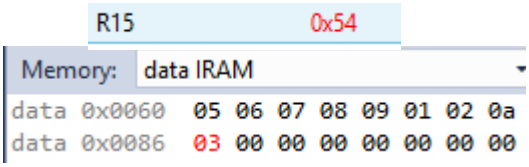
```
1  R15 = 0x41
2  UCSRA = 0b10000000
3  #4000
4  R15 = 0x54
5  UCSRA = 0b10000000
6  #4000
7  R15 = 0x0D
8  UCSRA = 0b10000000
9  #4000
10 UCSRA = 0b10000000
11 R15 = 0x0A
12
13 $log TCNT2
14 $startlog lab3.log
15 #10000
16 $stoplog
```

A



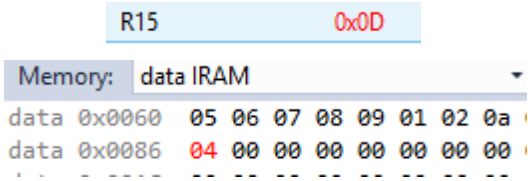
nextState=TWAITstate

T



nextState=CRWAITstate

<CR>

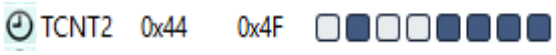


nextState=LSWAITstate

<LS>

1ο BREAK:transmit Digit

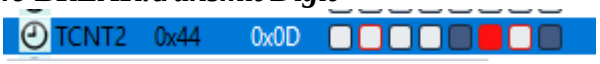
Output=O(ASCII)



3ο BREAK:transmit Digit

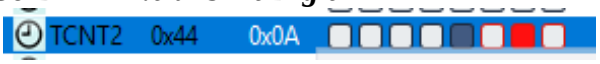
Output=K(ASCII)

4ο BREAK:transmit Digit



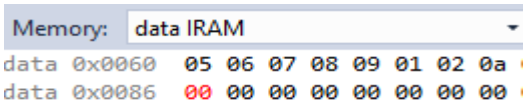
Output=<CR>(ASCII)

5ο BREAK:transmit Digit



Output=<LF>(ASCII)

6ο BREAK: reti



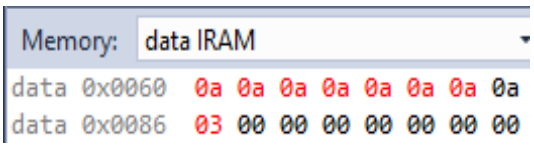
<u>Περίπτωση 2</u>

Είσοδος από USART:C<CR><LF>

<u>STIMULIFILE:</u>

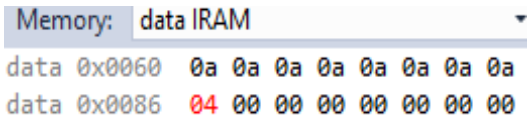
```
1  R15 = 0x41
2  UCSRA = 0b10000000
3  #4000
4  R15 = 0x54
5  UCSRA = 0b10000000
6  #4000
7  R15 = 0x0D
8  UCSRA = 0b10000000
9  #4000
10 UCSRA = 0b10000000
11 R15 = 0x0A
12
13 $log TCNT2
14 $startlog lab3.log
15 #10000
16 $stoplog
```

C



nextState=CRWAITstate

<CR>



nextState=LSWAITstate

<LS>

Το LS Simulation ειναι ακριβως ιδιο με προηγουμενος

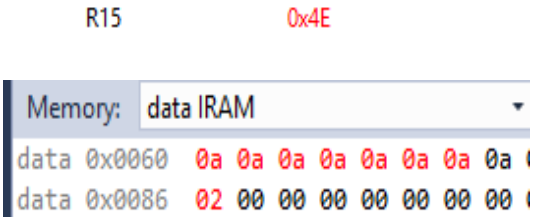
Περίπτωση 3

Είσοδος από USART:N012345<CR><LF>

STIMULIFILE:

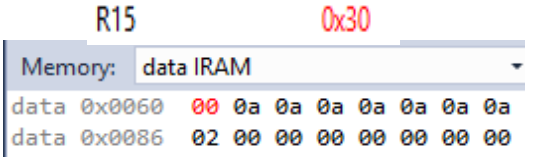
```
R15 = 0x4E
UCSRA = 0b10000000
#3000
R15 = 0x30
UCSRA = 0b10000000
#3000
R15 = 0x31
UCSRA = 0b10000000
#3000
R15 = 0x32
UCSRA = 0b10000000
#3000
R15 = 0x33
UCSRA = 0b10000000
#3000
R15 = 0x34
UCSRA = 0b10000000
#3000
R15 = 0x35
UCSRA = 0b10000000
#3000
R15 = 0x0D
UCSRA = 0b10000000
#3000
UCSRA = 0b10000000
R15 = 0x0A

$log TCNT2
$startlog lab3.log
#10000
$stoplog
```

N



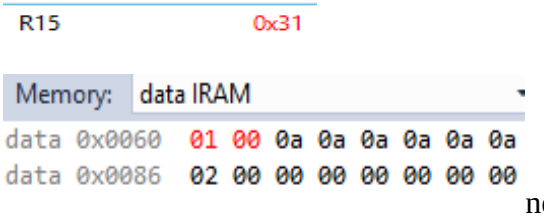
nextState=BYTEOUTstate

0



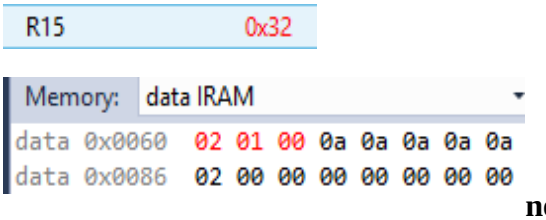
nextState=BYTEOUTstate

1



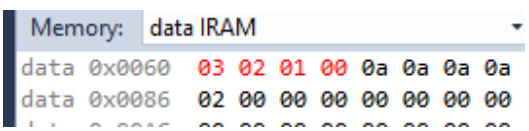
nextState=BYTEOUTstate

2



nextState=BYTEOUTstate

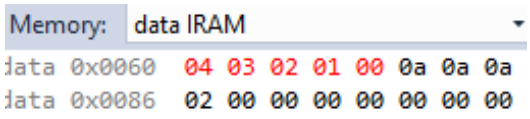
3



nextState=BYTEOUTstate

4

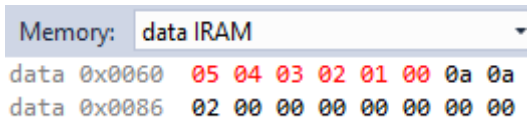
R15 0x34



nextState=BYTEOUTstate

5

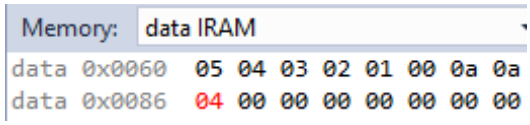
R15 0x35



nextState=LSWAITstate

<CR>

R15 0x0D



nextState=LSWAITstate

<LS>

Το LS Simulation είναι ακριβώς ίδιο με προηγουμένως