

Ενσωματωμένα Συστήματα Μικροεπεξεργαστών – ΗΡΥ411

Αναφορά Εργαστηρίου 6 – Warm Start, Cold Start και ο Watchdog Timer

Ομάδα: LAB41145877

Κιούλος Ευάγγελος 2016030056

Εισαγωγή:

Σκοπός της έκτης εργαστηριακής άσκησης ήταν η μελέτη της λειτουργίας και της χρησιμότητας του Watchdog Timer στον μικροελεγκτή AVR. Για να γίνει αυτό μελετήθηκε η υλοποίηση των λειτουργιών cold start και warm start με τη χρήση του watchdog timer. Στο συγκεκριμένο εργαστήριο υλοποιήθηκε πάνω στον κώδικα του προηγούμενου εργαστηρίου μόνο η λειτουργία του cold start.

Watchdog Timer:

Για την ενεργοποίηση του watchdog timer, πρέπει να θέσουμε το bit “WDE” του καταχωρητή “WDTCSR” στην τιμή ‘1’. Για να αρχικοποιήσουμε τον prescaler του watchdog timer, χρησιμοποιούμε τα bits “WDP2, WDP1, WDP0” του καταχωρητή “WDTCSR”. Αν δεν αρχικοποιήσουμε αυτά τα bits σε κάποια τιμή, ο watchdog timer θα κάνει time-out μετά από 16 χιλιάδες κύκλους. Για το συγκεκριμένο εργαστήριο τα bits “WDP2, WDP1, WDP0” τέθηκαν στην τιμή “001”. Σύμφωνα με τον πίνακα 17 στην σελίδα 43 του manual για την τιμή “001”, ο watchdog timer θα κάνει time-out μετά από 32 χιλιάδες κύκλους. Για την αρχικοποίηση του watchdog timer, δημιουργήθηκε η συνάρτηση “WDT_init()”, η οποία καλείται στην συνάρτηση main μαζί με τις υπόλοιπες αρχικοποιήσεις και φαίνεται παρακάτω.

```
/*  
 * Initialize watchdog timer.  
 */  
void WDT_init(void){  
  
    /*  
     * Bit WDE enables watchdog timer.  
     * Bits WDP0, WDP1, WDP2 set the prescaler for watchdog timer.  
     * Here, bit WDT0 is '1' and bits WDT1,WDT2 are '0'.  
     * So the watchdog timer runs for 32k cycles.  
     */  
    WDTCSR |= (1 << WDE) | (1 << WDP0);  
  
}
```

Όταν ο watchdog timer κάνει time-out, το πρόγραμμα μεταβαίνει στην διεύθυνση “\$000” και από εκεί στην διεύθυνση “\$02A” όπου βρίσκεται ο reset handler. Ο κώδικας του reset handler φαίνεται παρακάτω.

```
--- .././.././../crt1/gcrt1.S -----  
0000002A CLR R1          Clear Register  
--- .././.././../crt1/gcrt1.S -----  
0000002B OUT 0x3F,R1     Out to I/O location  
0000002C LDI R28,0x5F    Load immediate  
0000002D LDI R29,0x04    Load immediate  
0000002E OUT 0x3E,R29    Out to I/O location  
0000002F OUT 0x3D,R28    Out to I/O location  
00000030 CALL 0x00000086  Call subroutine  
00000032 JMP 0x00000154   Jump  
00000034 JMP 0x00000000   Jump
```

Στον παραπάνω κώδικα, παρατηρούμε ότι το πρόγραμμα αδειάζει την τιμή του καταχωρητή “R1” και έπειτα περνά στον καταχωρητή “0x3F” την τιμή του “R1”, δηλαδή την τιμή “0x00”. Ο καταχωρητής “0x3F” αντιστοιχεί στον καταχωρητή status register, επομένως το πρόγραμμα αρχικά σβήνει όλα τα flags από τον καταχωρητή status register. Στην συνέχεια φορτώνει την τιμή “0x5F” και “0x04” στους καταχωρητές “R28” και “R29”(οι οποίοι αντιστοιχούν στους καταχωρητές YL και YH). Επομένως, ο καταχωρητής Y παίρνει την τιμή “0x045F” που αντιστοιχεί στο τέλος της SRAM. Στην συνέχεια το πρόγραμμα φορτώνει την τιμή του καταχωρητή Y στους καταχωρητές “0x3D” και “0x3E” (που αντιστοιχούν τους SPL και SPH), επομένως, αρχικοποιεί την τον stack pointer στο τέλος της μνήμης. Τέλος το πρόγραμμα καλεί την συνάρτηση που βρίσκεται στην διεύθυνση “0x086”, δηλαδή την συνάρτηση main().

Κάθε φορά που λαμβάνουμε κάποιο χαρακτήρα στο USART, θέλουμε ο watchdog timer να γίνεται reset. Επομένως, η εντολή “WDR” καλείται στην αρχή του USART interrupt handler.

```
ISR(USART_RXC_vect){

    // Initialize watchdog timer
    //WDT_init();
    asm("WDR");
    // first address of bytes we want to display
    unsigned char *store_to_ram = (unsigned char *)0x60;
    // address where state is stored
    unsigned char *curr_state = (unsigned char *)0x6A;
    // used for reading input
    unsigned char readChar;
```

ΣΗΜΕΙΩΣΗ: Η συνάρτηση “_WDR()” που κάνει reset τον watchdog timer και αναφέρεται στην σελίδα 44 του manual δεν αναγνωρίζεται από τον compiler, επομένως καλούμε την εντολή “WDR” με inline assembly.

Cold Start:

Η λειτουργία του cold start αντιστοιχεί στην επανεκκίνηση του υπολογιστή χωρίς να διατηρούμε τα δεδομένα στην μνήμη. Επομένως, για την υλοποίηση του cold start με τον watchdog timer, αρκεί να μεταβούμε στις αρχικοποιήσεις της συνάρτησης main() μετά το time-out του watchdog timer. Αρχικά, στην συνάρτηση main() καλούμε την συνάρτηση “WDT_init()” για να αρχικοποιήσουμε τον watchdog timer.

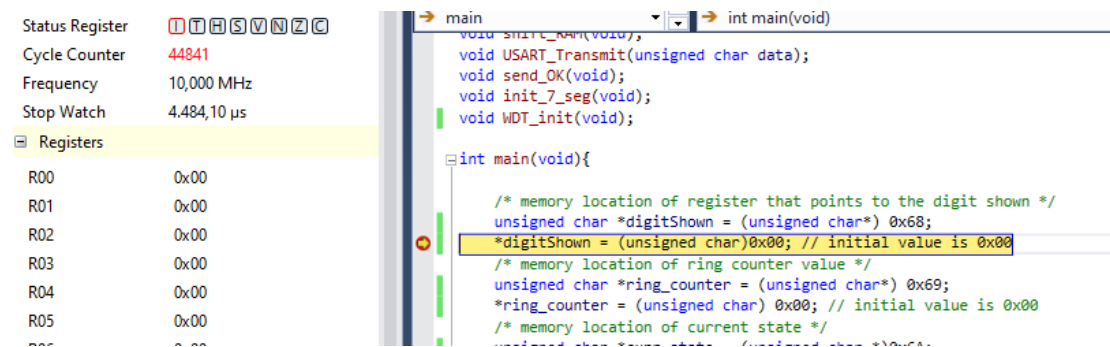
+ Watchdog Timer (WDT)				
Name	Address	Value	Bits	
+ WDTCSR	0x41	0x09	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Στην συνέχεια, τρέχουμε το πρόγραμμα μας με το αρχείο “.stim” για την μετάδοση της εντολής “N123<CR><LF>”. Αρχικά παρατηρούμε ότι τα ψηφία ‘1’, ‘2’ και ‘3’ έχουν αποθηκευτεί στην μνήμη.

Memory 4	
Memory:	data IRAM
Address:	0x0060,data
data 0x0060	03 02 01 0a 0a 0a 0a 0a 00 00 04 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00

Στην συνέχεια, μόλις τελειώσουμε την εκτέλεση του αρχείου “.stim”, περίπου στους 12 χιλιάδες κύκλους, μεταβαίνουμε στον handler της οθόνης. Τρέχοντας το πρόγραμμα,

παρατηρούμε ότι, μόλις το πρόγραμμα φτάσει στους 44 χιλιάδες κύκλους, δηλαδή 32 χιλιάδες κύκλους μετά την τελευταία φορά που έγινε reset ο watchdog timer, το πρόγραμμα μεταβαίνει στην αρχή της συνάρτησης main().



Τα στοιχεία της μνήμης πριν το time-out του watchdog timer φαίνονται παρακάτω.

Memory 4	
Memory: data IRAM	Address: 0x0060,data
data 0x0060	03 02 01 0a 0a 0a 0a 0a 04 08 00 00 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Να υπενθυμίσουμε ότι η πρώτη τιμή με κόκκινο χρώμα(δηλ. 04) δείχνει ποιο στοιχείο εμφανίζεται στην οθόνη, η δεύτερη τιμή με κόκκινο χρώμα(δηλ. 08) δείχνει την τιμή του ring counter και η τρίτη τιμή με κόκκινο χρώμα(δηλ. 00) δείχνει την κατάσταση του USART(η κατάσταση 0 είναι η αρχική κατάσταση του USART, μόλις στείλουμε "OK" το USART μεταβαίνει σε αυτή την κατάσταση). Συνεχίζοντας την εκτέλεση του προγράμματος, παρατηρούμε ότι οι αρχικοποιήσεις γίνονται από την αρχή, επομένως, οι τιμές που υπήρχαν στην μνήμη πριν το time-out του watchdog timer έχουν χαθεί, όπως φαίνεται και παρακάτω.

Memory 4	
Memory: data IRAM	Address: 0x0060,data
data 0x0060	0a 0a 0a 0a 0a 0a 0a 0a 00 00 00 00 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x008A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Δοκιμή Cold Start μετά από λάθος εντολή

Θα τρέχουμε το πρόγραμμα μας με το αρχείο ".stim" για την μετάδοση της εντολής "N123A<CR><LF>" (error.stim). Στον κώδικα του USART έχουμε προσθέσει έναν ατέρμονο βρόγχο σε κάθε κατάσταση για την περίπτωση που διαβάσουμε λανθασμένη εντολή, όπως φαίνεται παρακάτω.

```

else{
    *curr_state = (unsigned char) STATE_IDLE;
    while(1){
        asm("nop"); /* No Operation */
    }
}
break;

```

Παρατηρούμε ότι αφού το USART διαβάσει και αποθηκεύσει στην μνήμη τους χαρακτήρες “1,2,3”, διαβάζει τον χαρακτήρα “A” και μεταβαίνει σε έναν ατέρμονο βρόγχο όταν το πρόγραμμα έχει φτάσει στους περίπου 10 χιλιάδες κύκλους.

Cycle Counter: 10078
Frequency: 10,000 MHz
Stop Watch: 1.007,80 μs

Registers:

R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00

```

shift_RAM(); // shift bytes in RAM
store_to_ram[0] = readChar^0x30; // Mask the first
// Example, if input is 0x35, then in RAM we store 0x3
}
else if(readChar == 0x0D){
    *curr_state = (unsigned char) STATE_END;
}
else{
    *curr_state = (unsigned char) STATE_STORE;
    while(1){
        asm("nop"); /* No Operation */
    }
    break;
}

```

Τα δεδομένα στην μνήμη εκείνη την στιγμή είναι:

Memory 4																							
Memory:		data IRAM												Address:		0x0060,data							
data	0x0060	03	02	01	0a	0a	0a	0a	0a	00	00	04	00	00	00	00	00	c0	f9	a4	b0	99	
data	0x0075	92	82	f8	80	90	ff	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
data	0x008A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Στην συνέχεια, στους 42 χιλιάδες κύκλους, δηλαδή μετά από 32 χιλιάδες κύκλους, ο watchdog timer θα κάνει time-out. Επομένως θα μεταβούμε στην αρχή της main όπως φαίνεται στην παρακάτω εικόνα.

Cycle Counter: 42841
Frequency: 10,000 MHz
Stop Watch: 4.284,10 μs

Registers:

R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00

```

void shift_RAM(void);
void USART_Transmit(unsigned char data);
void send_OK(void);
void init_7_seg(void);
void WDT_init(void);

int main(void){
    /* memory location of register that points to the digit shown */
    unsigned char *digitShown = (unsigned char*) 0x68;
    *digitShown = (unsigned char)0x00; // initial value is 0x00
    /* memory location of ring counter value */
    unsigned char *ring_counter = (unsigned char*) 0x69;
    *ring_counter = (unsigned char) 0x00; // initial value is 0x00
    /* memory location of current state */
    unsigned char *curr_state = (unsigned char *)0x6A;
}

```

Συνεχίζοντας την εκτέλεση του προγράμματος, παρατηρούμε ότι τα δεδομένα της μνήμης παίρνουν τις αρχικές τους τιμές.

90 %

Memory 4

Memory: data IRAM Address: 0x0060,data

data 0x0060	0a	0a	0a	0a	0a	0a	0a	0a	00	00	00	00	00	00	00	c0 f9 a4 b0 99
data 0x0075	92	82	f8	80	90	ff	00	00	00	00	00	00	00	00	00	00 00 00 00
data 0x008A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00 00 00 00

Warm Start:

Η λειτουργία του warm start, αντιστοιχεί στην επανεκκίνηση του υπολογιστή χωρίς να αλλάξουν τα δεδομένα στην μνήμη. Επομένως, για την υλοποίηση του warm start με τον

watchdog timer, αρκεί να μεταβούμε στο σημείο της main μετά τις αρχικοποιήσεις της μνήμης. Για να δοκιμάσουμε την λειτουργία του warm start, θα αφαιρέσουμε από την main την κλήση της συνάρτησης “clear_RAM()” (που θέτει την τιμή “0x0A” στις διευθύνσεις 0x0060 – 0x0067) και την αρχικοποίηση των θέσεων 0x0068 που δείχνει ποιο ψηφίο προβάλλεται στην οθόνη, 0x0069 που δείχνει την τιμή του ring counter και 0x006A που δείχνει την κατάσταση της FSM του USART. Οι αρχικοποιήσεις αυτών των θέσεων, θα γίνουν στις ρουτίνες εξυπηρέτησης των interrupt. Επομένως η συνάρτηση main θα γίνει:

```
/* Initialize USART */
USART_init(MYUBRR);

/* Insert char 0x0A in allocated Memory (0x0060 - 0x0067) */
//clear_RAM();

/* Insert 7-segment digits in allocated Memory (0x0070 - 0x007A) */
init_7_seg();

// Set Port A and Port C as outputs
DDRA = 0xFF;
DDRC = 0xFF;

/* Initialize Timer/Counter0 */
TIMER0_init();

WDT_init();

sei(); // Global Interrupt Enable

while(1){
    asm("nop"); /* No Operation */
}
```

Στην συνέχεια, τρέχουμε ξανά το πρόγραμμα μας με το αρχείο “.stim” για την μετάδοση της εντολής “N123<CR><LF>”. Αρχικά, μόλις λάβουμε τον χαρακτήρα “N” θα σβήσουμε τα δεδομένα της οθόνης αποθηκεύοντας στις θέσεις 0x0060 με 0x0067 το χαρακτήρα “0x0A”, όπως φαίνεται παρακάτω.

Memory 4	
Memory:	data IRAM
Address:	0x0060,data
data 0x0060	0a 0a 0a 0a 0a 0a 0a 0a 00 00 04 00 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x008A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Στην συνέχεια, στους 12 χιλιάδες κύκλους, τελειώνει η εκτέλεση του αρχείου “.stim” επομένως, η λήψη χαρακτήρων από το USART τελειώνει και τα δεδομένα που έχουμε στην μνήμη είναι τα παρακάτω:

Memory 4	
Memory:	data IRAM
Address:	0x0060,data
data 0x0060	03 02 01 0a 0a 0a 0a 0a 01 01 00 00 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Η εκτέλεση του προγράμματος, συνεχίζει κανονικά μέχρι τους περίπου 44 χιλιάδες κύκλους όπου ο watchdog timer κάνει time out. Τα δεδομένα της μνήμης λίγο πριν το time out είναι τα παρακάτω:

Memory 4	
Memory: data IRAM	Address: 0x0060,data
data 0x0060	03 02 01 0a 0a 0a 0a 0a 04 08 00 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00

Μόλις γίνει το time out, το πρόγραμμα μεταβαίνει στην αρχή της συνάρτησης main όπως βλέπουμε και στην παρακάτω εικόνα.

The screenshot shows the AVR Studio IDE with the following components:

- Processor Status:**

Name	Value
Program Counter	0x00000086
Stack Pointer	0x045D
X Register	0x0000
Y Register	0x045F
Z Register	0x0000
Status Register	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Cycle Counter	44842
Frequency	10,000 MHz
Stop Watch	4.484,20 μs
- Registers:**

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
- Disassembly:**

```

main
/* Initialize USART */
USART_init(MYUBRR);

/* Insert char 0x0A in allocated Memory (0x0060 - 0x0067) */
//clear_RAM();

/* Insert 7-segment digits in allocated Memory (0x0070 - 0x007A) */
init_7_seg();

// Set Port A and Port C as outputs
DDRA = 0xFF;
DDRC = 0xFF;

/* Initialize Timer/Counter0 */
TIMER0_init();

WDT_init();

sei(); // Global Interrupt Enable

while(1){
    // ...
}

```
- Memory 4:**

Memory	Address	Data
data IRAM	0x0060	03 02 01 0a 0a 0a 0a 0a 04 08 00 00 00 00 00 c0 f9 a4 b0 99
data IRAM	0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00

Παρατηρούμε ότι ο program counter έχει την τιμή “0x086” που είναι η διεύθυνση της συνάρτησης main, όπως είδαμε από τον disassembler. Επιπλέον, τα περιεχόμενα της μνήμης δεν έχουν αλλάξει μετά το time-out. Συνεχίζοντας την εκτέλεση του προγράμματος, παρατηρούμε ότι τα δεδομένα έχουν παραμείνει ίδια.

The screenshot shows the AVR Studio IDE with the following components:

- Processor Status:**

Name	Value
Cycle Counter	44928
Frequency	10,000 MHz
Stop Watch	4.492,80 μs
- Registers:**

Register	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
- Disassembly:**

```

/* Initialize Timer/Counter0 */
TIMER0_init();

WDT_init();

sei(); // Global Interrupt Enable

while(1){
    asm("nop"); /* No Operation */
}

```
- Memory 4:**

Memory	Address	Data
data IRAM	0x0060	03 02 01 0a 0a 0a 0a 0a 04 08 00 00 00 00 00 c0 f9 a4 b0 99
data IRAM	0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00

Και στις δύο λειτουργίες παρατηρούμε ότι τα δεδομένα που υπάρχουν στους I/O καταχωρητές του μικροελεγκτή, έχουν χαθεί.

Δοκιμή Warm Start μετά από λάθος εντολή

Θα τρέχουμε το πρόγραμμα μας με το αρχείο “.stim” για την μετάδοση της εντολής “N123A<CR><LF>” (error.stim). Στον κώδικα του USART έχουμε προσθέσει έναν ατέρμονο βρόγχο σε κάθε κατάσταση για την περίπτωση που διαβάσουμε λανθασμένη εντολή, όπως φαίνεται παρακάτω.

```

else{
    *curr_state = (unsigned char) STATE_IDLE;
    while(1){
        asm("nop"); /* No Operation */
    }
}
break;

```

Παρατηρούμε ότι αφού το USART διαβάσει και αποθηκεύσει στην μνήμη τους χαρακτήρες “1,2,3”, διαβάζει τον χαρακτήρα “A” και μεταβαίνει σε έναν ατέρμονο βρόγχο όταν το πρόγραμμα έχει φτάσει στους περίπου 10 χιλιάδες κύκλους.

The screenshot shows the state of the microcontroller during execution. On the left, the 'Registers' window displays R00 through R04, all containing the value 0x00. The 'Cycle Counter' is at 10076, 'Frequency' is 10,000 MHz, and 'Stop Watch' is 1.007,60 μs. On the right, the code editor shows a C program snippet. A yellow highlight is placed on the line `asm("nop"); /* No Operation */` inside a `while(1){}` loop, which is part of the `STATE_IDLE` state handling.

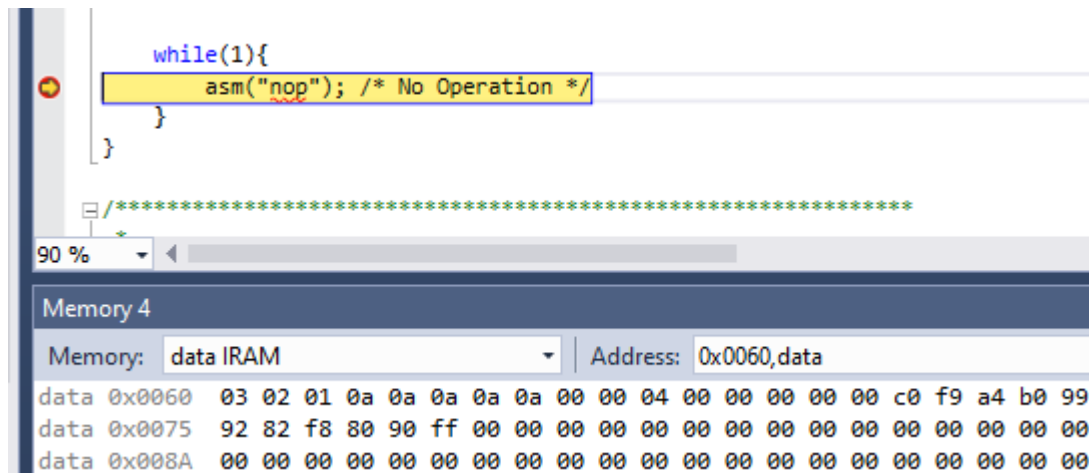
Τα δεδομένα στην μνήμη εκείνη την στιγμή είναι:

Memory 4	
Memory:	data IRAM
Address:	0x0060,data
data 0x0060	03 02 01 0a 0a 0a 0a 0a 00 00 04 00 00 00 00 c0 f9 a4 b0 99
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x008A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Στην συνέχεια, στους 42 χιλιάδες κύκλους, δηλαδή μετά από 32 χιλιάδες κύκλους, ο watchdog timer θα κάνει time-out. Επομένως θα μεταβούμε στην αρχή της main όπως φαίνεται στην παρακάτω εικόνα.

The screenshot shows the microcontroller's state after a reset. The 'Registers' window on the left shows the Program Counter (PC) at 0x00000086 and the Stack Pointer (SP) at 0x045D. Other registers (R00-R04) are 0x00. The 'Disassembly' window on the right shows the assembly code for the `main` function. The first instruction is `main: int main(void)`, followed by `/* Insert char 0x0A in allocated Memory (0x0060 - 0x0067) //clear_RAM();`, then `/* Initialize USART */ USART_init(MYUBRR);`, and finally `/* Insert 7-segment digits in allocated Memory (0x0070 - 0x0077) //init_7_seg();`. The code also shows `// Set Port A and Port C as outputs DDRA = 0xFF; DDRC = 0xFF;`.

Συνεχίζοντας την εκτέλεση του προγράμματος, παρατηρούμε ότι τα δεδομένα της μνήμης κρατούν τις προηγούμενες τιμές τους.



The screenshot shows an IDE with a C code snippet and a memory dump window. The code is a while loop containing an assembly instruction to perform a no-operation. The memory dump window, titled 'Memory 4', shows three lines of memory data from 0x0060 to 0x008A, with the first line containing non-zero values.

```
while(1){  
    asm("nop"); /* No Operation */  
}
```

90 %

Memory 4

Memory:	data IRAM	Address:	0x0060,data
data 0x0060	03 02 01 0a 0a 0a 0a 0a 00 00 04 00 00 00 00 00 c0 f9 a4 b0 99		
data 0x0075	92 82 f8 80 90 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
data 0x008A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

Πιθανή υλοποίηση του Warm Start

Με βάση τις παραπάνω παρατηρήσεις, για να υλοποιήσουμε την λειτουργία του warm start, μπορούμε να δεσμεύσουμε μία θέση στην μνήμη, στην οποία θα βλέπουμε σε τι κατάσταση είναι το πρόγραμμα μας. Στον κώδικα του reset handler, θα ελέγχεται η τιμή της θέσης αυτής. Αν η θέση αυτή έχει την τιμή '0', τότε σημαίνει ότι πρέπει να κάνουμε cold start, επομένως το πρόγραμμα θα μεταβαίνει στην αρχή της συνάρτησης main όπου υπάρχουν οι αρχικοποιήσεις τις μνήμης. Αν έχει τιμή διάφορη του '0', τότε θα κάνουμε warm start, μεταβαίνοντας στο σημείο της main μετά τις αρχικοποιήσεις της μνήμης. Αρχικά η τιμή αυτής της διεύθυνσης θα είναι '0'. Όταν τρέξει το πρόγραμμα μας, η τιμή αυτής της διεύθυνσης θα παίρνει κάποια τιμή διάφορη του '0'. Σε περίπτωση σφάλματος, όπου θα χρειάζεται cold start, η τιμή αυτή θα τίθεται πάλι στο '0' πριν το time-out του watchdog timer.