

## Κ22 - Θέμα 3

### Συστήματα Μικροπολογιστών - Ελένη Νασοπούλου

(Β) Στο μΕ 8085 όταν εκτελείται η εντολή **CALL 0610H**, ο μετρητής προγράμματος είναι (PC)=0520H και ο σεικτικής σωρού (SP)=0900H. Στο μέσον της εκτέλεσης της εντολής συμβαίνει διακοπή RST 5.5. Δώστε τις νέες τιμές των PC, SP, το περιεχόμενο του σωρού καθώς και τις λειτουργίες που συμβαίνουν στην αρχή και στην επιστροφή από την ρουτίνα εξυπηρέτησης. (0.5 ΜΟΝΑΔΕΣ)

Οι λειτουργίες που εκτελούνται ως ανάρτηση στη software διακοπή RST 5.5 είναι οι εξής:

1. Ολοκλήρωση τρέχουσας εντολής:

πριν:  $\left\{ \begin{array}{l} (PC) = 0520H \\ (SP) = 0900H \end{array} \right\}$     μετά:  $\left\{ \begin{array}{l} (PC) = 0610H \\ (SP) = 08FEH \end{array} \right\}$      $(08FEH) = 20H$   
 $(08FFH) = 05H$

2. σωζεται ο PC και η κατάσταση του μΕ και το PC μεταβαίνει στη ρουτίνα εξυπηρέτησης

$\left\{ \begin{array}{l} (PC) = 002CH \\ (SP) = 08FCH \end{array} \right\}$      $(08FC H) = 10H$   
 $(08FD H) = 06H$   
 $(08FE H) = 20H$   
 $(08FF H) = 05H$

|        |      |
|--------|------|
| RESET  | 0000 |
| ...    | ...  |
| RST1   | 0008 |
| ...    | ...  |
| RST2   | 0010 |
| ...    | ...  |
| RST3   | 0018 |
| RST4   | 0020 |
| ...    | ...  |
| TRAP   | 0024 |
| ...    | ...  |
| RST5   | 0028 |
| ...    | ...  |
| RST5,5 | 002C |
| ...    | ...  |
| RST6   | 0030 |
| ...    | ...  |
| RST6,5 | 0034 |
| ...    | ...  |
| RST7   | 0038 |
| ...    | ...  |
| RST7,5 | 003C |
| ...    | ...  |

3. μετά την ολοκλήρωση της ρουτίνας διακοπής ανακατάει η κατάσταση του μΕ και εκτελεί την επόμενη εντολή

$\left\{ \begin{array}{l} (PC) = 0610H \\ (SP) = 08FEH \end{array} \right\}$      $(08FE H) = 20H$   
 $(08FF H) = 05H$

## Κ22 - Θέμα 4

**ΘΕΜΑ 4<sup>ο</sup>:** (1.5 ΜΟΝΑΔΕΣ): Απαντήστε στα παρακάτω ερωτήματα (σύντομα και αιτιολογημένα):

(α) Ποιος είναι ο ρόλος της στοίβας σε έναν μικροεπεξεργαστή; Αναφέρετε δυο παραδείγματα χρήσης της. (0.3 ΜΟΝΑΔΕΣ)

(β) Ποια η διαφορά ρουτίνας και μακροεντολής; Σε ποιες περιπτώσεις θα προτιμούσατε την χρήση της μιας μεθόδου και σε ποιες της άλλης; (0.3 ΜΟΝΑΔΕΣ)

(γ) Για ποιο λόγο η μνήμη των μικροελεγκτών είναι χωρισμένη σε τμήματα με διαφορετική διευθυνσιοδότηση; (0.3 ΜΟΝΑΔΕΣ)

(δ) Οι μεταβλητές και οι σταθερές ενός προγράμματος σε ένα Μικροελεγκτή AVR σε ποιες μνήμες αποθηκεύονται και γιατί; Δώστε σε γλώσσα C (για AVR ενσωματωμένα συστήματα) από ένα παράδειγμα δήλωσης μίας συμβολοσειράς που δεσμεύει χώρο μνήμης στην SRAM, Flash και EEPROM; (0.3 ΜΟΝΑΔΕΣ)

## Κ22 - Θέμα 4

(ε) Οι μεταβλητές και οι σταθερές ενός προγράμματος σε ένα Μικροελεγκτή AVR σε ποιες μνήμες αποθηκεύονται και γιατί; Δώστε σε γλώσσα C (για AVR ενσωματωμένα συστήματα) από ένα παράδειγμα δήλωσης μιάς συμβολοσειράς που δεσμεύει χώρο μνήμης στην SRAM, Flash και EEPROM; (0,3 ΜΟΝΑΔΕΣ)

1. **SRAM** (Static RAM): Αυτή η μνήμη χρησιμοποιείται για τις μεταβλητές κατά την εκτέλεση του προγράμματος. Όλες οι μεταβλητές που μπορούν να αλλάξουν κατά τη διάρκεια της εκτέλεσης αποθηκεύονται στη SRAM. Αυτή η μνήμη είναι ταχύτερη, αλλά είναι πτητική, πράγμα που σημαίνει ότι χάνει τα δεδομένα της όταν το μικροελεγκτής απενεργοποιηθεί.
2. **Flash Memory**: Αυτή η μνήμη είναι μη πτητική και χρησιμοποιείται για την αποθήκευση του εκτελέσιμου κώδικα και των σταθερών δεδομένων. Οι σταθερές που δεν αλλάζουν κατά την εκτέλεση του προγράμματος αποθηκεύονται στη Flash Memory. Όταν ο μικροελεγκτής ενεργοποιηθεί, τα δεδομένα αυτά παραμένουν σταθερά.
3. **EEPROM** (Electrically Erasable Programmable Read-Only Memory): Αυτή η μνήμη είναι επίσης μη πτητική και χρησιμοποιείται για την αποθήκευση δεδομένων που πρέπει να παραμείνουν μετά από επανεκκίνηση του μικροελεγκτή. Είναι πιο αργή σε σχέση με τη SRAM και τη Flash, αλλά επιτρέπει τη διατήρηση των δεδομένων μεταξύ των κύκλων ενέργειας.

```
#include <avr/io.h>
```

```
#include <avr/pgmspace.h>
```

```
#include <avr/eeprom.h>
```

```
char sram_string[] = "Hello, SRAM!";
```

```
const char flash_string[] PROGMEM = "Hello, Flash!";
```

```
char eeprom_string[] EEMEM = "Hello, EEPROM!";
```

```
int main(void) {
```

```
    // Ανάγνωση από τη Flash
```

```
    char buffer[20];
```

```
    strcpy_P(buffer, flash_string);
```

```
    // Ανάγνωση από την EEPROM
```

```
    char eeprom_buffer[20];
```

```
    eeprom_read_block((void*)eeprom_buffer, (const void*)eeprom_string, sizeof(eeprom_string));
```

```
    return 0;
```

```
}
```

1. **\*\*SRAM\*\***: Η μεταβλητή `sram\_string` αποθηκεύεται στη SRAM και μπορεί να αλλάξει κατά την εκτέλεση του προγράμματος.
2. **\*\*Flash\*\***: Η σταθερά `flash\_string` αποθηκεύεται στη Flash Memory και δεν αλλάζει κατά την εκτέλεση του προγράμματος. Η μακροεντολή `PGMEM` χρησιμοποιείται για να δηλώσει ότι τα δεδομένα πρέπει να αποθηκευτούν στη Flash.
3. **\*\*EEPROM\*\***: Η συμβολοσειρά `eeprom\_string` αποθηκεύεται στην EEPROM χρησιμοποιώντας τη μακροεντολή `EEMEM`.

(β) Στο μΕ 8085 όταν εκτελείται η εντολή CALL 0530H, ο μετρητής προγράμματος είναι (PC)=0630H και ο δείκτης σωρού (SP)=0740H. Στο μέσον της εκτέλεσης της εντολής συμβαίνει διακοπή RST 5.5. Δώστε τις νέες τιμές των PC, SP, το περιεχόμενο του σωρού καθώς και τις λειτουργίες που συμβαίνουν στην αρχή και στην επιστροφή από την ρουτίνα εξυπηρέτησης.

(0.5 ΜΟΝΑΔΕΣ)

Οι λειτουργίες που ενδεχόμενα ως αναπάντη στη software διακοπή RST 5.5 είναι οι εξής.

1. Ολοκλήρωση τρέχουσας εντολής:

πριν: { (PC)=0630H }    μετὰ { (PC)=0530H }    (073FH)=30H  
           { (SP)=0740H }            { (SP)=073FH }    (073FH)=06H

2. Σωζέται ο PC και η κατάσταση του μΕ και το PC μεταβαίνει στη ρουτίνα εξυπηρέτησης

{ (PC)=002CH }    (073CH)=30H  
                       (0730H)=05H  
                       (073FH)=30H  
                       (073FH)=06H

3. Μετά την ολοκλήρωση της ρουτίνας διακοπής ανακατάει η κατάσταση του μΕ και εκτελεί την επόμενη εντολή

{ (PC)=0530H }    (073FH)=30H  
                       { (SP)=073FH }    (073FH)=06H

(β) Στο μΕ 8085 όταν εκτελείται η εντολή JMP 0820H, ο μετρητής προγράμματος είναι (PC)=0780H και ο δείκτης στοίβας (SP)=900H. Στο μέσον της εκτέλεσης της εντολής συμβαίνει διακοπή RST 6.5. Δώστε τις νέες τιμές των PC, SP, το περιεχόμενο της στοίβας καθώς και τις λειτουργίες που συμβαίνουν στην αρχή και στην επιστροφή από την ρουτίνα εξυπηρέτησης.

(0.5 ΜΟΝΑΔΕΣ)

Οι λειτουργίες που ενδεχόμενα ως αναπάντη στη software διακοπή RST 6.5 είναι οι εξής.

1. Ολοκλήρωση τρέχουσας εντολής:

πριν: { (PC)=0780H }    μετὰ { (PC)=0820H }  
           { (SP)=0900H }            { (SP)=0900H }

2. Σωζέται ο PC και η κατάσταση του μΕ και το PC μεταβαίνει στη ρουτίνα εξυπηρέτησης

{ (PC)=0034H }    (08FEH)=20H  
                       { (SP)=08FEH }    (08FEH)=08H

3. Μετά την ολοκλήρωση της ρουτίνας διακοπής ανακατάει η κατάσταση του μΕ και εκτελεί την επόμενη εντολή

{ (PC)=0820H }  
                       { (SP)=0900H }

|        |      |
|--------|------|
| RESET  | 0000 |
| ...    | ...  |
| RST1   | 0008 |
| ...    | ...  |
| RST2   | 0010 |
| ...    | ...  |
| RST3   | 0018 |
| RST4   | 0020 |
| ...    | ...  |
| TRAP   | 0024 |
| ...    | ...  |
| RST5   | 0028 |
| ...    | ...  |
| RST5.5 | 002C |
| ...    | ...  |
| RST6   | 0030 |
| ...    | ...  |
| RST6.5 | 0034 |
| ...    | ...  |
| RST7   | 0038 |
| RST7.5 | 003C |
| ...    | ...  |

(β) Έστω ο παρακάτω κώδικας προγράμματος σε assembly μΥ-Σ 8085. Να εξηγήσετε και να υπολογίσετε το άνω όριο του αριθμού ενεργοποιήσεων της διακοπής RST 5.5, ώστε να έχουμε ασφαλή λειτουργία του προγράμματος. Να προτείνετε βελτιστοποιήσεις για την ασφαλή λειτουργία του κώδικα assembly μΥ-Σ 8085, ανεξαρτήτως του αριθμού ενεργοποιήσεων των διακοπών RST 5.5.

-Interrupts

Slide 18

RST5.5: JMP LOOP1  
 ...  
 START: LXI SP, LOOP2+9  
 MVI B, FFH  
 LOOP1: INR B  
 MVI A, 0EH  
 SIM  
 EI  
 LOOP2: JMP LOOP2

|        |           |           |
|--------|-----------|-----------|
| επανάφ | JMP       | LOOP2     |
|        | (10 BYTE) | LOOP2 + 1 |
|        | (11 BYTE) | LOOP2 + 2 |
| επανάφ | (10 BYTE) | LOOP2 + 3 |
|        | (11 BYTE) | LOOP2 + 4 |
| επανάφ | (10 BYTE) | LOOP2 + 5 |
|        | (11 BYTE) | LOOP2 + 6 |
| επανάφ | PCL       | LOOP2 + 7 |
|        | PCW       | LOOP2 + 8 |
| SP →   |           | LOOP2 + 9 |

0EH → 0000 1110 → bit 3=1

από ενεργεία διακοπής RST 5.5

στην 5<sup>η</sup> επανάληψη ο σωρός θα επικαλύπτεται τον κώδικα του προγράμματος

Οπότε μπορεί:

RST5.5: JMP LOOP1  
 ...  
 START:  
 MVI B, 0H  
 MVI A, 0EH  
 SIM  
 EI  
 LOOP2: JMP LOOP2

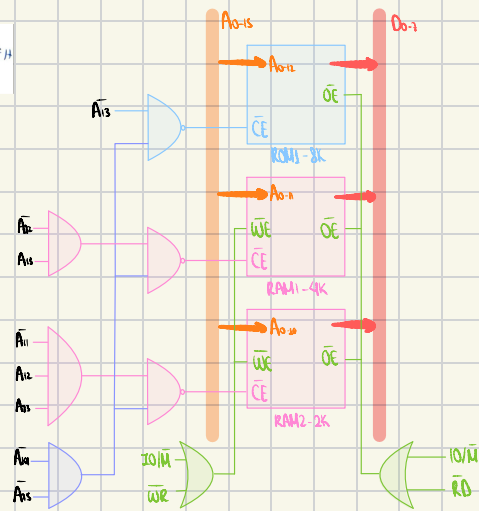
LOOP1:  
 INR B  
 EI  
 RET

RST5.5: JMP LOOP1  
 ...  
 START:  
 LXI SP, LOOP2+9  
 MVI B, FFH  
 LOOP1:  
 INR B  
 MVI A, 0EH  
 SIM  
 EI  
 LOOP2: JMP LOOP2

(6) Έστω ένα  $\mu\text{Υ-2}$  με τα παρακάτω χαρακτηριστικά: 1) Μονάδα μνήμης ROM 8K (Διευθύνσεις 0000H – 1FFFH); 2) Μονάδα μνήμης RAM 4K (Διευθύνσεις 2000H – 2FFFH); 3) Μονάδα μνήμης RAM 2K (Διευθύνσεις 3000H – 3FFFH). Να σχεδιάσετε το λογικό κύκλωμα που επιλέγει την κάθε μονάδα μνήμης ανάλογα με την εκάστοτε ζητούμενη διεύθυνση. Η υλοποίηση να γίνει με τη χρήση λογικών πυλών (0,6 ΜΟΝΑΔΕΣ).

| $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | addr  | mem  |
|----------|----------|----------|----------|----------|----------|-------|------|
| 0000     | 0000     | 0000     | 0000     | 0000     | 0000     | 0000H | ROM  |
| 0001     | 1111     | 1111     | 1111     | 1111     | 1111     | 1FFFH |      |
| 0010     | 0000     | 0000     | 0000     | 0000     | 0000     | 2000H | RAM1 |
| 0010     | 1111     | 1111     | 1111     | 1111     | 1111     | 2FFFH |      |
| 0011     | 0000     | 0000     | 0000     | 0000     | 0000     | 3000H | RAM2 |
| 0011     | 0111     | 1111     | 1111     | 1111     | 1111     | 3FFFH |      |

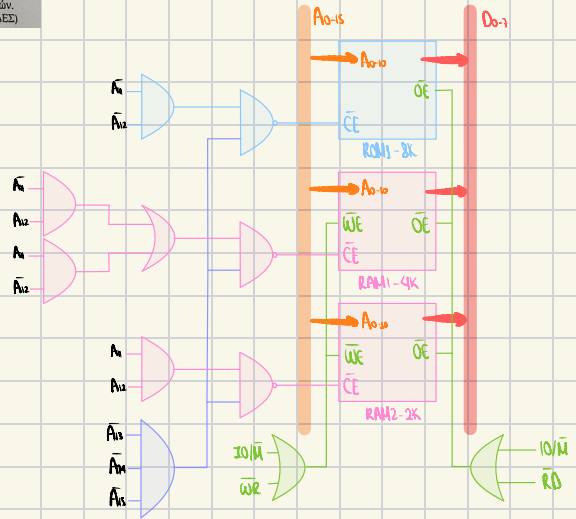
$A_{15}, A_{14}$ : επιλεγείν  
 $A_{15}, A_{14}$ : bit διαίρεσης  
 ROM:  $A_{15}=0$  και addr:  $A_{12}=0$   
 RAM1:  $(A_{15}, A_{14})=(1,0)$  και addr:  $A_{10}=0$   
 RAM2:  $(A_{15}, A_{14})=(1,1,0)$  και addr:  $A_{10}=0$



(7) Έστω ένα  $\mu\text{Υ-2}$  με τα εξής χαρακτηριστικά: 1) Μονάδα μνήμης ROM 2K (Διευθύνσεις 0000H – 07FFFH); 2) Μονάδα μνήμης RAM 4K (Διευθύνσεις 0800H – 17FFFH) και 3) Μονάδα μνήμης RAM 2K (Διευθύνσεις 1800H – 1FFFH). Με βάση τον τρόπο διεύθυνσής, να σχεδιάσετε το λογικό κύκλωμα που επιλέγει την κάθε μονάδα μνήμης ανάλογα με την εκάστοτε ζητούμενη διεύθυνση. Η υλοποίηση να γίνει αποκλειστικά με τη χρήση λογικών πυλών (0,6 ΜΟΝΑΔΕΣ).

| $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | addr   | mem  |
|----------|----------|----------|----------|----------|----------|--------|------|
| 0000     | 0000     | 0000     | 0000     | 0000     | 0000     | 0000H  | ROM  |
| 0000     | 0111     | 1111     | 1111     | 1111     | 1111     | 07FFFH |      |
| 0000     | 1000     | 0000     | 0000     | 0000     | 0000     | 0800H  | RAM1 |
| 0000     | 1111     | 1111     | 1111     | 1111     | 1111     | 0FFFH  |      |
| 0001     | 0000     | 0000     | 0000     | 0000     | 0000     | 1000H  |      |
| 0001     | 0111     | 1111     | 1111     | 1111     | 1111     | 17FFFH |      |
| 0001     | 1000     | 0000     | 0000     | 0000     | 0000     | 1800H  | RAM2 |
| 0001     | 1111     | 1111     | 1111     | 1111     | 1111     | 1FFFH  |      |

ROM:  $(A_{12}, A_{11})=(0,0)$  και addr:  $A_{10}=0$   
 RAM1:  $(A_{12}, A_{11})=(0,1)$  ή  $(A_{12}, A_{11})=(1,0)$  και addr:  $A_{10}$   
 RAM2:  $(A_{12}, A_{11})=(1,1)$  και addr:  $A_{10}=0$



## CALL 0950H

Εξέω ότι το η εντολή CALL βρίσκεται στη θέση 0630H και ο SP συν 0740H

πρι:  $\left\{ \begin{array}{l} (PC)=0630H \\ (SP)=0740H \end{array} \right\}$  μετα  $\left\{ \begin{array}{l} (PC)=0950H \\ (SP)=073EH \end{array} \right\}$  (073EH)=30H  
 (073FH)=06H

Σημειών θα μεταβαίμε στη διεύθυνση 0950H της CALL και η τρέχουσα διεύ θα πια στο stack

Θα ολοκληρωθεί η CALL και θα επανέλθουμε στην απεισ επόμενη εντολή και θα επανέλθουμε το stack  $\rightarrow (SP)=0740H$