Άσκηση 5 - Απαντήσεις

Σύνθετοι έλεγχοι – Δομές επανάληψης

Α. ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Για το θεωρητικό μέρος, ανατρέξτε στην άσκηση 4 (εκεί έχουν παρουσιαστεί οι τεχνικές ελέγχου)

Β. ΠΕΙΡΑΜΑΤΙΚΟ ΜΕΡΟΣ

1. Το πρόγραμμα που ακολουθεί, ελέγχει αν ο αριθμός που είναι αποθηκευμένος στη θέση [0002] ανήκει στο διάστημα [1A, 40]. Αν ανήκει, τότε θα αποθηκεύεται ο αριθμός 5 στη θέση μνήμης [0005].

Πρόγραμμα	**Πρόγραμμα στον debugger**		**Ερωτήσεις**
mov AL,[0001]	0100	mov AL,[0002]	α) Ποιος είναι ακριβώς ο έλεγχος που υλοποιείται ;
CMP AL,1a	0103	CMP AL,1a	
JGE syn2	0105	JGE 0109	β) Σχεδιάστε το διάγραμμα ροής
jmp exit	0107	jmp 0114	
syn2:	0109		γ) Δοκιμάστε το πρόγραμμα για περιεχόμενο θέσης μνήμης [0002] τους αριθμούς 10, 20 και 41
CMP AL,40		CMP AL,40	
JLE syn3	010B	JLE 010F	
jmp exit	010D	jmp 0114	*το λογισμικό αντικαθιστά το JGE με το JNL (Non Less)
syn3:	010F		
mov [0005],5		mov [0005],5	
exit:	0114		

Απαντήσεις

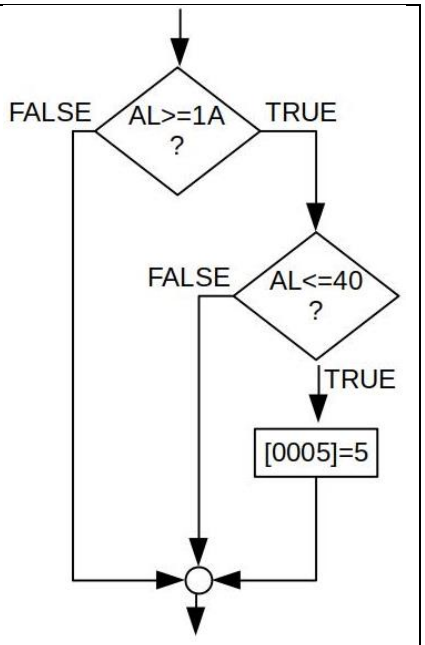
Το πρόγραμμα ελέγχει αν ένας αριθμός (που προέρχεται από θέση μνήμης), ανήκει σε ένα συγκεκριμένο κλειστό διάστημα. Εφόσον το διάστημα είναι κλειστό, οι οριακές τιμές συμπεριλαμβάνονται σε αυτό. Πιο συγκεκριμένα, ο έλεγχος που υλοποιείται είναι:

Αν ((AL>=1A) and (AL<=40)) τότε
  [0005]=5

Στη γλώσσα Assembly, ο παραπάνω έλεγχος υλοποιείται με την ακόλουθη λογική (τμηματικά):

Αν (AL>=1A), τότε
  Αν (AL<=40), τότε
    [0005]=5

Με απλά λόγια, Αν ο αριθμός είναι μεγαλύτερος ή ίσος από το αριστερό όριο και μικρότερος ή ίσος από το δεξί όριο του διαστήματος, τότε ανήκει σε αυτό. Αν η πρώτη συνθήκη είναι False, τότε ο αριθμός είναι εκτός διαστήματος από την αριστερή πλευρά. Αντίθετα, αν η δεύτερη συνθήκη είναι False, τότε ο αριθμός είναι εκτός διαστήματος από τη δεξιά πλευρά. Αν οποιαδήποτε από τις δύο συνθήκες είναι False, τότε ο αριθμός δεν ανήκει στο συγκεκριμένο διάστημα



2. Το πρόγραμμα που ακολουθεί, ελέγχει αν ο αριθμός που είναι αποθηκευμένος στη θέση [0002] ανήκει στο διάστημα [1A, 40]. Αν ανήκει, τότε θα αποθηκεύεται ο αριθμός 5 στη θέση μνήμης [0005].

Πρόγραμμα	**Πρόγραμμα στον debugger**		**Ερωτήσεις**
mov AL,[0001]	0100	mov AL,[0002]	α) Ποιος είναι ακριβώς ο έλεγχος που υλοποιείται ;
CMP AL,1a	0103	CMP AL,1a	
JL exit	0105	JL 0110	β) Σχεδιάστε το διάγραμμα ροής
CMP AL,40	0107	CMP AL,40	
JG exit	0109	JG 0110	γ) Δοκιμάστε το πρόγραμμα για περιεχόμενο θέσης μνήμης [0002] τους αριθμούς 10, 20 και 41
mov [0005],5	010B	mov [0005],5	
exit:	0110		δ) Να σχολιάσετε επαρκώς τις διαφορές των δύο υλοποιήσεων-προγραμμάτων (ερωτήματα 1 και 2)

Απαντήσεις

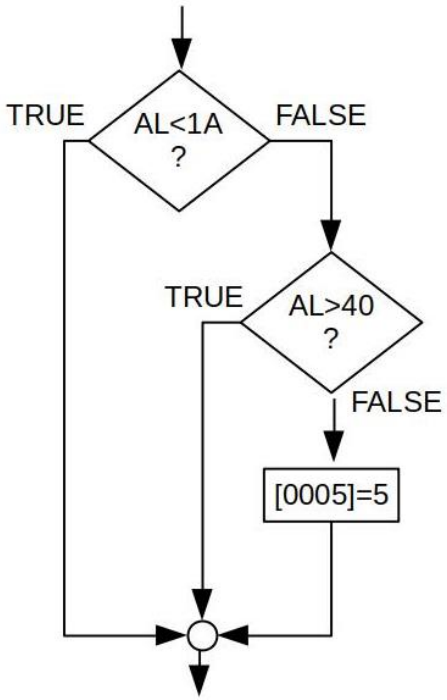
Παρά το γεγονός ότι και σε αυτή την περίπτωση ελέγχουμε αν ο αριθμός ανήκει στο συγκεκριμένο διάστημα, ο έλεγχος υλοποιείται με την ανάποδη λογική (σε σχέση με το προηγούμενο ερώτημα). Αν ο αριθμός είναι μικρότερος από το αριστερό όριο ή μεγαλύτερος από το δεξί όριο, τότε δεν ανήκει στο διάστημα, διαφορετικά ανήκει. Αρχικά λοιπόν, γίνεται έλεγχος αν ο αριθμός είναι μικρότερος από το 1A. Αν αυτό ισχύει, τότε ο αριθμός δεν ανήκει στο διάστημα και οδηγούμαστε στην έξοδο. Διαφορετικά (περίπτωση False της συνθήκης, άρα ο αριθμός είναι μεγαλύτερος ή ίσος από το 1A), ελέγχουμε και το δεξί όριο. Αν αυτή η συνθήκη είναι True (ο αριθμός είναι δηλαδή μεγαλύτερος από 40), τότε οδηγούμαστε στην έξοδο. Αν και οι δύο συνθήκες είναι False (ο αριθμός δεν είναι μικρότερος από το 1A, αλλά ούτε και μεγαλύτερος από το 40), τότε ο αριθμός ανήκει στο διάστημα.
Έτσι, η λογική ελέγχου είναι:

```
Αν ((AL<1A) ή (AL>40)), τότε
  Έξοδος
Διαφορετικά
  [0005]=5
```

Στην Assembly, η παραπάνω υλοποίηση, είναι στη μορφή:

```
Αν (AL<1A), τότε
  Έξοδος
Διαφορετικά
  Αν (AL>40), τότε
    Έξοδος
  Διαφορετικά
     [0005]=5
```

Στο ερώτημα 1 ξεκινάμε με τη λογική ελέγχου, αν ανήκει (λογική AND για τα όρια), ενώ σε αυτό το ερώτημα ξεκινάμε με τη λογική ελέγχου αν δεν ανήκει (λογική OR για τα όρια).



3. Δοκιμάστε το ακόλουθο πρόγραμμα:

Πρόγραμμα	**Πρόγραμμα στον debugger**		**Ερωτήσεις**
mov al,1	0100	mov al,1	α) Σχεδιάστε διάγραμμα ροής
mov bl,0	0102	mov bl,0	
start:	0104:		β) Τι υπολογίζει ;
add bl,al		add bl,al	
inc al	0106	inc al	γ) Ποια είναι τα βασικά συστατικά ενός βρόχου επανάληψης ; Πώς υλοποιούνται στη γλώσσα Assembly ;
cmp al,0a	0108	cmp al,0a	
JLE start	010A	JLE 0104	

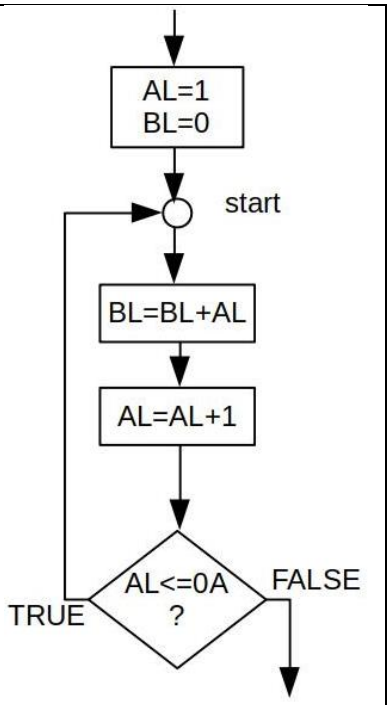
Απαντήσεις

Πρόκειται για την υλοποίηση ενός βρόχου επανάληψης τύπου do-while. Ο καταχωρητής al παίζει το ρόλο του μετρητή, με αρχική τιμή 1. Από την άλλη μεριά, ο καταχωρητής bl φιλοξενεί το άθροισμα το οποίο υπολογίζεται προσθέτοντας την τρέχουσα τιμή του μετρητή. Η επανάληψη συνεχίζεται όσο ο μετρητής (καταχωρητής al) παίρνει τιμές μικρότερες ή ίσες από το 0A (δηλαδή την τιμή 10 στο δεκαδικό). Άρα, θα γίνουν συνολικά 10 επαναλήψεις. Έτσι, το πρόγραμμα υπολογίζει το άθροισμα 1+2+…+10.

Στη δομή τύπου do-while, ο έλεγχος για τη συνέχιση ή όχι της επανάληψης, γίνεται στο τέλος της δομής. Αυτό σημαίνει ότι, οι εντολές εντός της επανάληψης θα εκτελεστούν τουλάχιστον μια φορά πριν τον έλεγχο.

Ένας βρόχος επανάληψης περιλαμβάνει υποχρεωτικά, τα ακόλουθα συστατικά:
α) μετρητή για το πλήθος των επαναλήψεων (εδώ το al)
β) ενημέρωση του μετρητή (π.χ. αύξηση ή μείωση) (inc)
γ) έλεγχο για τη συνέχιση ή όχι των επαναλήψεων (cmp, jle)

Εξαίρεση αποτελούν ειδικοί βρόχοι με τους οποίους θέλουμε να υλοποιήσουμε μια επαναληπτική δομή που δεν τερματίζεται ποτέ (άπειρη επανάληψη).



4. Δοκιμάστε το ακόλουθο πρόγραμμα:

Πρόγραμμα	**Πρόγραμμα στον debugger**		**Ερωτήσεις**
mov al,0a	0100	mov al,0a	α) Σχεδιάστε διάγραμμα ροής
mov bl,0	0102	mov bl,0	
start:	0104:		β) Τι υπολογίζει ;
add bl,al		add bl,al	
dec al	0106	dec al	γ) Γιατί δεν υπάρχει εντολή CMP, ενώ απαιτείται έλεγχος ;
JNZ start	0108	JNZ 0104	
	010A		δ) Ποια προγραμματιστική δομή υλοποιείται στα ερωτήματα 3 και 4;

*το λογισμικό αντικαθιστά το JNZ με το JNE (Not Equal)

Απαντήσεις

Το διάγραμμα ροής έχει την ίδια μορφή με το προηγούμενο ερώτημα, αφού γίνεται ξανά υλοποίηση δομής τύπου do-while. Αυτό το πρόγραμμα υπολογίζει το ίδιο άθροισμα με το προηγούμενο, αλλά με αρχική τιμή το 0A (10 στο δεκαδικό). Υπολογίζει δηλαδή το άθροισμα 10+9+8+…+1. Η μείωση του al (dec al), φέρνει κάποια στιγμή το αντίστοιχο περιεχόμενο στην τιμή μηδέν. Όταν συμβεί αυτό, ενεργοποιείται το bit Z του καταχωρητή κατάστασης. Οι εντολές JNZ και JNE ελέγχουν στην ουσία αυτό το bit (Z) και όσο αυτό δεν έχει ενεργοποιηθεί, γίνεται άλμα στο επιθυμητό σημείο. Έτσι, δεν χρειάζεται εντολή CMP, αφού η

εντολή dec είναι αυτή που ενεργοποιεί το bit το οποίο στη συνέχεια ελέγχει η εντολή άλματος. Θυμηθείτε ότι, όταν εκτελείται μια εντολή cmp, τα αντίστοιχα δυο ορίσματα αφαιρούνται και προκαλείται πάλι η ενεργοποίηση κάποιων bit του καταχωρητή κατάστασης. Αν όμως θα έπρεπε να κάνουμε τον έλεγχο αν AL<=10, τότε θα χρησιμοποιούσαμε εντολή cmp. Τέλος, όπως έχει ήδη αναφερθεί, υλοποιείται και στα δύο ερωτήματα η δομή επανάληψης τύπου do-while.

5. Δοκιμάστε το ακόλουθο πρόγραμμα:

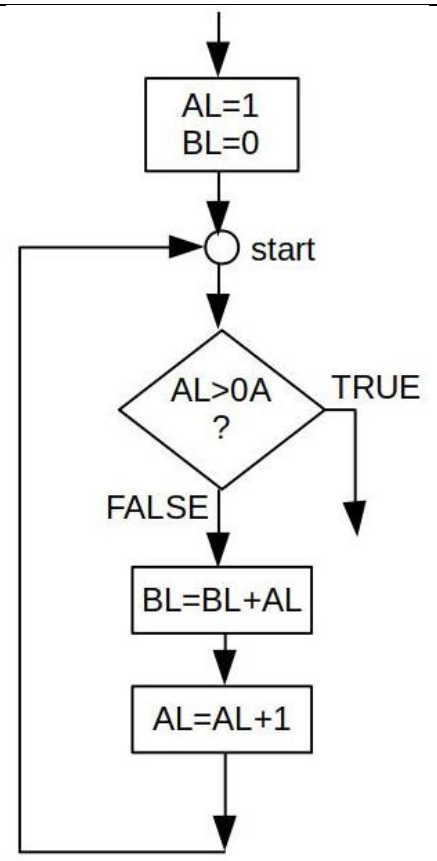
Πρόγραμμα	**Πρόγραμμα στον debugger**		**Ερωτήσεις**
mov al,1	0100	mov al,1	α) Σχεδιάστε διάγραμμα ροής
mov bl,0	0102	mov bl,0	
start:	0104		β) Τι υπολογίζει ;
cmp al,0a		cmp al,0a	
JG exit	0106	JG 010E	γ) Ποια προγραμματιστική δομή υλοποιείται;
add bl,al	0108	add bl,al	
inc al	010A	inc al	δ) Να συγκρίνετε τις υλοποιήσεις των ερωτημάτων 4 και 5
JMP start	010C	jmp 0104	
exit:	010E		ε) Ποιες αλλαγές θα έπρεπε να γίνουν στο πρόγραμμα, αν αντικαθιστούσαμε την εντολή JG με JLE ;

Απαντήσεις

Υπολογίζει το άθροισμα 1+2+3+…+10 με τη χρήση μιας δομής τύπου while-do. Σε αυτό τον τύπο επανάληψης, ο έλεγχος γίνεται στην αρχή, πράγμα που σημαίνει ότι δεν εκτελούνται οι εντολές εντός της επανάληψης καμία φορά, αν δεν έχει προηγηθεί ο έλεγχος. Αρχικά, γίνεται σύγκριση του al με το 0A (10 στο δεκαδικό). Αν το al είναι μεγαλύτερο του 0A, δεν έχει νόημα να γίνει η επανάληψη, αφού ξεπεράστηκε το όριο. Έτσι, αν η συνθήκη αυτή είναι True, η ροή εκτέλεσης οδηγείται στην έξοδο. Αν όμως η συνθήκη είναι False (άρα ισχύει ότι al<=0A), τότε δεν γίνεται άλμα στο σημείο exit και μοιραία εκτελείται η εντολή που ακολουθεί και έτσι βρισκόμαστε εντός βρόχου. Μετά την ενημέρωση του αθροίσματος (εντολή add) αλλά και του μετρητή (εντολή inc), πρέπει να γίνει ξανά έλεγχος αν ο μετρητής ξεπέρασε τη μέγιστη τιμή. Για αυτό το λόγο, χρησιμοποιείται η εντολή JMP start που οδηγεί ξανά τη ροή εκτέλεσης στο σημείο ελέγχου για τη συνέχιση ή όχι του βρόχου.
Οι διαφορές στις υλοποιήσεις (ερωτήματα 4 και 5) βασίζονται στις διαφορές των δομών επανάληψης do-while και while-do.

Αντικαθιστώντας την εντολή JG με JLE, το πρόγραμμα διαμορφώνεται ως εξής:

```
mov al,1
mov bl,0
start:
cmp al,0a
JLE syneheia
JMP exit
syneheia:
  add bl,al
  inc al
  JMP start
exit:
```



6. Υλοποιήστε μια δομή επανάληψης **do-while** σε Assembly (από τα προηγούμενα ερωτήματα) και **C**. Να σχολιάσετε ομοιότητες και διαφορές.

Απαντήσεις

Γλώσσα Assembly	**Γλώσσα C**
mov al,0A	i=10;
start:	do {
;Εντολές	/* Εντολές */
dec al	i=i-1;
JNZ start	} while (i>0);

Ο βρόχος do-while όπως φαίνεται στον διπλανό κώδικα, έχει ακριβώς την ίδια μορφή, είτε σε γλώσσα Assembly, είτε σε γλώσσα C. Αυτό συμβαίνει επειδή βασίζονται στον ίδιο αλγόριθμο. Στο παράδειγμα αυτό, φαίνονται και τα συστατικά αυτής της δομής.

Το συμπέρασμα που βγαίνει από αυτό το παράδειγμα είναι ότι, αν γνωρίζουμε τον αλγόριθμο, τότε μπορούμε να κάνουμε την υλοποίηση σε οποιαδήποτε γλώσσα προγραμματισμού.

Επομένως, αν δυσκολευόμαστε στην ανάπτυξη κώδικα Assembly, σημαίνει ότι πρέπει να μελετήσουμε τον αντίστοιχο αλγόριθμο. Η δυσκολία δηλαδή, δεν βρίσκεται στη ίδια τη γλώσσα Assembly.

7. Να υλοποιήσετε ένα πρόγραμμα για την ακόλουθη λειτουργία:

Φόρτωση του AL από τη θέση μνήμης [0005]
Αν AL<0, τότε [0002]=FF
Αν AL=0, τότε υπολογισμός του 1+2+…+10
Αν AL>0, τότε [0003]=1 και [0004]=2

Δοκιμάστε το πρόγραμμα για περιεχόμενο της θέσης [0005] τις τιμές 00, 01 και FF

Απάντηση

```
mov al, [0005]
cmp al,0
JL MIK
JE ISO
MEG:
  mov [0003],1
  mov [0004],2
  jmp EXIT
MIK:
  mov [0002],0ff
  jmp EXIT
ISO:
  mov al,1
  mov bl,0
  start:
    add bl,al
    inc al
    cmp al,0a
  JLE start
EXIT:
```