ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

Ver. 1, Rev. 3

Εγχειρίδιο Ασκήσεων Εργαστηρίου Συμβολικής Γλώσσας (Assembly)

ΧΑΡΙΔΗΜΟΣ ΒΕΡΓΟΣ

Νοέμβριος 2008

Περιεχόμενα

Εργασία 1

Μεταφράστε το παρακάτω πρόγραμμα και εκτελέστε το **βηματικά**:

Μετακίνηση Δεδομένων

```
1   .arm
2   .text
3   .global main
4
5   main:
6   STMDB R13!, {R0-R12, R14}   @E0
7
8   MOV R0, #0x20               @E1
9   MOV R1, R0, LSL #2          @E2
10  MVN R2, R1, LSL #1          @E3
11
12  LDR R3, =Values             @E4
13  LDR R4, [R3], #4            @E5
14  LDRB R5, [R3], #2           @E6
15  LDRSH R6, [R3], #2          @E7
16  LDR R3, =Stack              @E8
17  STMIA R3!, {R0-R2, R4-R6}   @E9
18  LDMDB R3!, {R0-R2}          @E10
19  LDMDB R3!, {R4-R6}          @E11
20
21  LDMIA R13!, {R0-R12, PC}    @E12
22
23  .data
24  Values:
25  .word 0xCAFEBABA
26  .word 0x82345678
27  Stack:
28  .word 0,0,0,0
29  .word 0,0,0,0
```

Μετά από την εκτέλεση κάθε εντολής καταγράψτε το περιεχόμενο των καταχωρητών R0-R6, PC.

	R0	R1	R2	R3	R4	R5	R6	PC
E0								
E1								
E2								
E3								
E4								
E5								
E6								
E7								
E8								
E9								
E10								
E11								

Εργασία 2

i. Αθροιση bytes

Δίδονται οι δύο παρακάτω πίνακες Α και Β. Κάθε ένας τους περιέχει 16 στοιχεία, όπου κάθε στοιχείο είναι ένας δεκαδικός αριθμός που απαιτεί για την αναπαράστασή του στο δυαδικό σύστημα 8 δυαδικά ψηφία. Αναπτύξτε πρόγραμμα σε συμβολική γλώσσα που να προσθέτει τα αντίστοιχα στοιχεία των δύο αυτών πινάκων (το στοιχείο της 1ης γραμμής του πίνακα Α προστίθεται με αυτό της 1ης γραμμής του πίνακα Β, κοκ.) και να αποθηκεύει τα 16 αποτελέσματα σε ένα νέο πίνακα, έστω Γ, του οποίου τα στοιχεία έχουν εύρος 8 δυαδικά ψηφία το καθένα. Η μεθοδολογία που θα ακολουθήσετε για την ανάπτυξη του προγράμματός σας θα πρέπει να χρησιμοποιεί ετικέτες και άλματα υπό συνθήκη. Συμπληρώστε τα αθροίσματα που προέκυψαν στον παρακάτω πίνακα στο δεκαεξαδικό, και κάνοντας τη μετατροπή, και στο δεκαδικό. Στη τελευταία στήλη του πίνακα σημειώστε με «Χ» κάθε περίπτωση που το αποτέλεσμα διαφέρει από το αναμενόμενο. Προσπαθείστε να εξηγήσετε τα μη αναμενόμενα αποτελέσματα.

byte	Πίνακας Α	Πίνακας Β	Πίνακας Γ		Μη αναμενόμενο
			Δεκαεξαδικό	Δεκαδικό	
0	32	19			
1	127	1			
2	254	18			
3	57	89			
4	22	90			
5	111	112			
6	48	89			
7	11	32			
8	87	23			
9	45	98			
10	114	67			
11	45	83			
12	66	146			
13	23	140			
14	134	200			
15	168	67			

ii. Αθροιση halfwords

byte	Πίνακας Α	Πίνακας Β	Πίνακας Γ
0	32	19	
1	127	1	
2	254	18	
3	57	89	
4	22	90	
5	111	112	
6	48	89	
7	11	32	
8	87	23	
9	45	98	
10	114	67	
11	45	83	
12	66	146	
13	23	140	
14	134	200	
15	168	67	

Στο δεύτερο μέρος πρέπει να εκτελέσετε τις αντίστοιχες προσθέσεις, θεωρώντας πως οι πίνακές μας αποτελούνται από 8 στοιχεία - αριθμούς των 16 δυαδικών ψηφίων ο καθένας. Τα αποτελέσματα της κάθε πρόσθεσης θα πρέπει να είναι επίσης εύρους 16 δυαδικών ψηφίων. Για παράδειγμα, η πρώτη πρόσθεση θα γίνει ανάμεσα στους αριθμούς (127*256 + 32 = 32544) και (1*256 + 19 = 275). Συμπληρώστε τα αποτελέσματα που προκύπτουν στο δεκαδικό. Θυμηθείτε πως η αρχιτεκτονική μας είναι little endian, το οποίο σημαίνει ότι το λιγότερο σημαντικό byte ενός halfword είναι τοποθετημένο στη χαμηλότερη διεύθυνση μνήμης, ενώ το περισσότερο σημαντικό byte είναι τοποθετημένο στην υψηλότερη. Στην περίπτωσή μας, τα bytes στις θέσεις 1,3,5,. . . είναι τα περισσότερο σημαντικά, ενώ τα bytes στις θέσεις 0,2,4,. . . είναι τα λιγότερα σημαντικά.

iii. Αθροιση words

byte	Πίνακας Α	Πίνακας Β	Πίνακας Γ
0	32	19	
1	127	1	
2	254	18	
3	57	89	
4	22	90	
5	111	112	
6	48	89	
7	11	32	
8	87	23	
9	45	98	
10	114	67	
11	45	83	
12	66	146	
13	23	140	
14	134	200	
15	168	67	

Στο τρίτο μέρος πρέπει να γίνουν οι αθροίσεις θεωρώντας πως τα στοιχεία των πινάκων Α και Β έχουν εύρος 32 δυαδικά ψηφία, όπως επίσης και τα ζητούμενα αθροίσματα. Και εδώ θα πρέπει να λάβετε υπόψη σας την little endian αρχιτεκτονική και τη σειρά των bytes.

iv. Αθροιση longwords

byte	Πίνακας Α	Πίνακας Β		Πίνακας Γ
0	32	19		
1	127	1	⎫	
2	254	18	⎪	
3	57	89	⎪	
4	22	90	⎪	
5	111	112	⎪	
6	48	89	⎪	
7	11	32	⎪ 16	
8	87	23	⎭ bytes	
9	45	98	⎪	
10	114	67	⎪	
11	45	83	⎪	
12	66	146	⎪	
13	23	140	⎪	
14	134	200	⎭	
15	168	67		

Στο τελευταίο μέρος θα θεωρήσουμε πως οι πίνακές μας έχουν ένα μόνο στοιχείο εύρους 16 bytes. Το αποτέλεσμα της άθροισης αυτών των δύο στοιχείων (το στοιχείο δηλαδή του πίνακα Γ) θα πρέπει επίσης να έχει εύρος 16 bytes. Επειδή η αρχιτεκτονική της ΑΛΜ του επεξεργαστή μας δεν υποστηρίζει πράξεις επί ποσοτήτων μεγαλύτερων από 32 δυαδικά ψηφία θα πρέπει να κατασκευάσουμε ένα πρόγραμμα το οποίο να υλοποιεί αυτή

την εξειδικευμένη πρόσθεση. Ξεκινάμε με την πρόσθεση των λιγότερο σημαντικών words, αποθηκεύουμε το κρατούμενο της πράξης στη σημαία Carry του καταχωρητή κατάστασης και το χρησιμοποιούμε στην πρόσθεση των δύο αμέσως σημαντικότερων words. Η διαδικασία αυτή θα πρέπει να επαναληφθεί 4 φορές, μέχρι να προσθέσουμε και τις περισσότερο σημαντικές words των αριθμών μας. Πόσες προσθέσεις θα χρειαζόμασταν αν εκτελούσαμε προσθέσεις bytes και όχι words;

Εργασία 3

i. Μελέτη καταχωρητή κατάστασης

Αριθμητικές πράξεις

```
1   .arm
2   .text
3   .global main
4
5   main:
6   STMDB R13!, {R0-R12, R14}
7
8   MOV R0, #94
9   MOV R1, R0, LSR #1
10
11  ADDS R2, R0, R0          @Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης
12  ADDS R2, R1, R1          @Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης
13  ADDS R2, R0, R1          @Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης
14
15  MOV R0, #0x80000000
16  ADD R1, R0, #0x80
17  MOV R2, #1
18
19  SUBS R3, R0, R2          @Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης
20  SUBS R3, R0, R1          @Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης
21  RSBS R3, R0, R1          @Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης
22
23  LDMIA R13!, {R0-R12, PC}
```

Μεταγλωτίστε το παραπάνω πρόγραμμα και εκτελέστε το βηματικά. Αμέσως μετά από την εκτέλεση κάθε γραμμής που έχει το σχόλιο *@Σημειώστε το περιεχόμενο του καταχωρητή κατάστασης*, ελέγξτε τις σημαίες του καταχωρητή κατάστασης και καταγράψτε ποιες ενεργοποιούνται και ποιες απενεργοποιούνται. Στη στήλη «Σχόλια» προσπαθείστε να εξηγήσετε το γιατί. Τέλος, ελέγξτε τα αποτέλεσματα που αποθηκεύονται στους καταχωρητές R2 και R3 μετά από κάθε πρόσθεση και αφαίρεση αντίστοιχα. Αναφέρετε αν εμφανίζεται το φαινόμενο αποκοπής δυαδικών ψηφίων του αποτελέσματος λόγω περιορισμένου εύρους καταχωρητών και σε ποιες πράξεις.

	N	C	Z	V	Σχόλια
@11					
@12					
@13					

@19					
@20					
@21					

ii. Προσπέλαση διαδοχικών θέσεων μνήμης

Γράψτε ένα πρόγραμμα, το οποίο να αποθηκεύει τους αριθμούς από 0... 5 στις θέσεις μνήμης από [Stor]... [Stor+5]. Η ετικέτα Stor να οριστεί σαν ετικέτα στο κομμάτι κώδικα δεδομένων (data region) που ακολουθεί τον εκτελέσιμο κώδικα.

iii. Υπολογισμός αριθμών Fibonacci

Μετατρέψτε το πρόγραμμα που κατασκευάσατε στο (ii), ώστε να παράγει τους 6 πρώτους αριθμούς Fibonacci στις θέσεις μνήμης από [Stor] ... [Stor+5]. Οι αριθμοί Fibonacci παράγονται με βάση τον τύπο a_n = a_{n-1} + a_{n-2} όπου το $n > 2$ και $a_0 = 1, a_1 = 2$. Δηλαδή, θα χρειαστείτε 3 καταχωρητές, έναν για να κρατά το a_n, έναν για το a_{n-1} και έναν για το a_{n-2}. Σε κάθε επανάληψη θα πρέπει να μεταφέρετε τα περιεχόμενα του a_{n-1} στον a_{n-2} και του a_n στον a_{n-1} και να υπολογίζετε τον νέο a_n σαν το άθροισμα των άλλων 2.

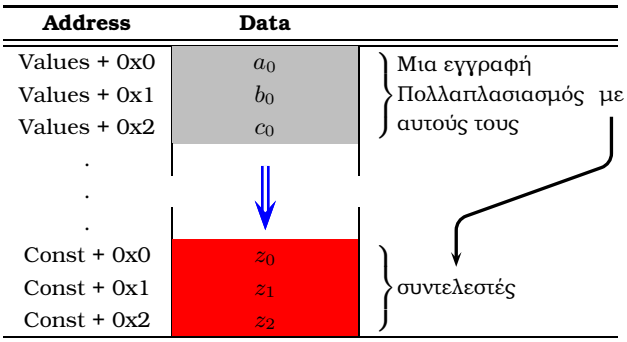
Εργασία 4

i. Υπολογισμός μαθηματικού τύπου

Δίδεται ο μαθηματικός τύπος $x = 5 * (a_i * z_0 + b_i * z_1 - c_i * z_2)/64$, όπου οι όροι a_i, b_i, c_i συμβολίζουν μεταβλητές, ενώ οι όροι z_i συμβολίζουν σταθερές. Οι μεταβλητές είναι ομαδοποιημένες στη μνήμη, με την ακόλουθη διάταξη:

Address	Variable
0x00	a_0
0x01	b_0
0x02	c_0
0x03	a_1
0x04	b_1
0x05	c_1
⋮	⋮

Παρατηρείστε πως οι μεταβλητές είναι τοποθετημένες διαδοχικά (a_i, b_i, c_i) και κάθε τριάδα ξεκινά ανά 3 θέσεις μνήμης (δηλαδή η μεταβλητή a_i θα βρίσκεται 3 θέσεις μετά από τη θέση του a_{i-1}). Επιπλέον, γνωρίζοντας τη θέση του a_i μπορεί να υπολογιστεί η θέση του b_i και c_i, διότι βρίσκονται 1 και 2 θέσεις μετά από τη θέση του a_i αντίστοιχα. Οι σταθερές z είναι τοποθετημένες με την ίδια διάταξη (δηλαδή z_0,$z_1 z_2$) σε διαφορετική θέση μνήμης.



Καλείστε να υλοποιήσετε μια υπορουτίνα υπολογισμού του παραπάνω μαθηματικού τύπου, η οποία θα λαμβάνει στον καταχωρητή R0 τη διεύθυνση της μεταβλητής a_i (για τον υπολογισμό του τύπου με δεδομένα εισόδου την i-οστή τριάδα a_i, b_i, c_i). Πριν ολοκληρωθεί η υπορουτίνα, το αποτέλεσμα από τον υπολογισμό της i-οστής τριάδας θα πρέπει να τοποθετηθεί στον καταχωρητή R0. Ενδεικτικά βήματα του αλγορίθμου, από τη στιγμή που αρχίζει η εκτέλεση της υπορουτίνας, είναι:

1. Αποθήκευση του περιεχομένου των καταχωρητών που θα χρησιμοποιήσουμε στο σωρό.
2. Μεταφορά των δεδομένων από τη μνήμη στους καταχωρητές.

3. Εκτέλεση των πράξεων και αποθήκευση του αποτελέσματος στον R0.
4. Επαναφορά των αρχικών τιμών στους καταχωρητές. (μετά από αυτό το βήμα, οι τιμές των R1-R12 πρέπει να είναι ίδιες με αυτές που είχαν πριν ξεκινήσει η υπορουτίνα)

Για παράδειγμα ο παρακάτω κώδικας παρουσιάζει τα περιγραφόμενα βήματα:

Υπορουτίνα

```
1   .arm
2   .text
3   .global main
4
5   main:
6   STMDB R13!, {R1, R2}
7   MOV R0, =Values          @Αποθηκεύουμε τη διεύθυνση των δεδομένων
8   BL Subrtn                @Καλούμε την υπορουτίνα
9   LDMIA R13!, {R1, R2}
10
11  Subrtn:
12  STMDB R13!, {R1, R2}     @Αποθηκεύουμε στο σωρό το περιεχόμενο των καταχωρητών
13  LDRB R1, [R0, #0]        @Μεταφέρουμε στον R1 το byte της διεύθυνσης μνήμης όπου
                             δείχνει ο R0
14  LDRB R2, [R0, #1]        @Μεταφέρουμε στον R2 το byte της επόμενης διεύθυνσης μ-
                             νήμης απ'όπου δείχνει ο R0
15  MUL R1, R2, R1           @Τα πολλαπλασιάζουμε και αποθηκεύουμε το αποτέλεσμα στον
                             R1
16  STRB R1, [R0, #2]        @Μεταφέρουμε το αποτέλεσμα στη θέση μνήμης που βρίσκεται
                             2 θέσεις μετά από αυτή που δείχνει ο R0
17  LDMIA R13!, {R1, R2}     @Επανακτούμε το περιεχόμενο των καταχωρητών που είχαμε
                             σώσει
18  MOV PC,LR                @Επιστρέφουμε από την υπορουτίνα στο σημείο όπου κλήθηκε
19
20  .data
21  Values:
22  .byte 0x02, 0x03, 0x00
23
```

Μόλις ετοιμάσετε την υπορουτίνα αναπτύξτε ένα πρόγραμμα, όπου η βασική συνάρτηση main καλεί την υπορουτίνα και της περνά τα εξής δεδομένα (τοποθετώντας στον R0 τη διεύθυνση της μεταβλητής a_i):

Δεδομένα

```
1   .data
2   Values:
3   .byte 0x02, 0x03, 0x04
4   .byte 0x10, 0x05, 0x06
5   .byte 0x0B, 0x02, 0x0D
6   .byte 0x01, 0x0C, 0x08
```

```
7
8  Const:
9  .byte 0x04, 0x07, 0x05
```

Καταγράψτε στον ακόλουθο πίνακα τα 4 αποτελέσματα που παράγει η κλήση της υπορουτίνας.

Επανάληψη	Αποτέλεσμα
1	
2	
3	
4	

Υπόδειξη

Για λόγους απλότητας μπορείτε να καλέσετε την υπορουτίνα 4 φορές. Στον καταχωρητή R0 την πρώτη φορά θα αποθηκεύσετε την τιμή **=Values**, την δεύτερη φορά την τιμή **=Values** και θα προσθέσετε και 3 κλπ.

ii. Εύρεση μέγιστης τιμής σε πίνακα αποτελεσμάτων

Σε αυτό το μέρος καλείστε να μετατρέψετε τον κώδικά σας, έτσι ώστε να εντοπίζει ποιο από τα αποτελέσματα της υπορουτίνας είναι το μεγαλύτερο και ποιο σύνολο δεδομένων το παρήγαγε. Αν δηλαδή το 3ο σετ παράγει το αποτέλεσμα 0x35 θέλουμε να αποθηκευτεί στο 4ο byte μετά από την ετικέτα Const η τιμή 0x35 και στο 5ο byte το νούμερο του συνόλου, ξεκινώντας την αρίθμηση από το 0 (δηλαδή αν το πρώτο σετ δώσει το μεγαλύτερο αποτέλεσμα, στο 5ο byte θα πρέπει να αποθηκευτεί το 0).

iii. Υπολογισμός πολυωνύμου

Σκοπός του τελευταίου μέρους της εργασίας είναι ο υπολογισμός της τιμής ενός πολυωνύμου 6ου βαθμού, με γενικό τύπο $\sum_{i=0}^{6} a_i * x^i$. Οι σταθερές a_i παραμένουν ίδιες κάθε φορά (όπως οι σταθερές z_i στα προηγούμενα υποερωτήματα), αλλά η παράμετρος που εισάγεται στην υπορουτίνα μέσω του R0 είναι η τιμή του x (και όχι η διεύθυνσή του). Οι σταθεροί όροι είναι αποθηκευμένοι από τον a_0 προς τον a_6, με τον πρώτο να βρίσκεται στη διεύθυνση Const, ενώ ο τελευταίος στη διεύθυνση Const+6. Θέλει προσοχή το γεγονός πως το x δεν είναι byte, αλλά word. Υλοποιήστε την υπορουτίνα και εκτελέστε τη στα παρακάτω δεδομένα:

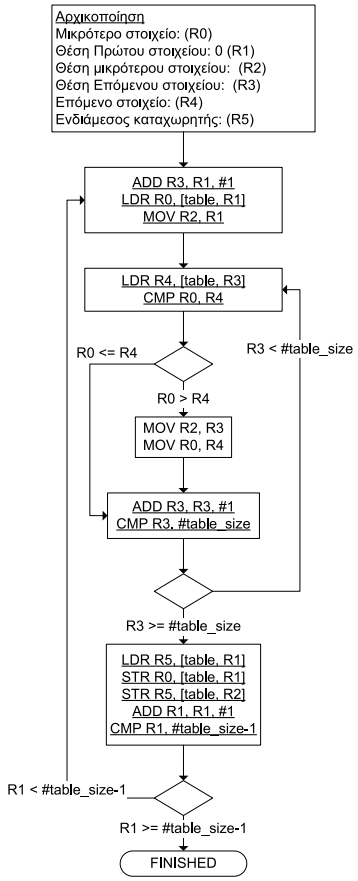
Δεδομένα	
1	**.data**
2	**Values:**
3	**.word 0x10**
4	**.word 0x50A**
5	**.word 0xCDCA**
6	**.word 0x80AB**
7	
8	**Const:**
9	**.byte 0x04, 0x07, 0x05**
10	**.byte 0x20, 0x1A, 0x12, 0x06**

Υπόδειξη

Ενώ φαίνεται απλό να πολλαπλασιαστεί το x με τον εαυτό του τόσες φορές όσες είναι η αντίστοιχη δύναμή του, και στη συνέχεια να το πολλαπλασιαστεί με τον σταθερό συντελεστή, κάτι τέτοιο δεν είναι πολύ αποδοτικό (για τον παραπάνω υπολογισμό θα χρειαστούμε 21 πολλαπλασιασμούς και 6 προσθέσεις). Μια εναλλακτική και πιο αποδοτική αντιμετώπιση είναι η επαναληπτική εκτέλεση των πράξεων : $b_{i-1} = b_i * x + a_{i-1}$ για i : 6...1, με $b_6 = a_6$. Ο όρος b_0 αποτελεί τη τιμή του πολυωνύμου (αν αναπτυχθεί ο επαναληπτικός τύπος μόνο σε όρους a και x, θα παρατηρείσετε ότι εμφανίζεται ο αρχικός τύπος του πολυωνύμου) και υπολογίζεται με μόνο 6 πολλαπλασιασμούς και 6 προσθέσεις.

Εργασία 5

Σε αυτή την εργασία θα κατασκευάσουμε ένα αλγόριθμο ταξινόμησης δεδομένων. Η λογική κάθε αλγορίθμου ταξινόμησης στηρίζεται σε μια σειρά από συγκρίσεις ανάμεσα στα δεδομένα και στην εναλλαγή της σειράς τους, έτσι ώστε στην αρχή του πίνακα να βρεθεί το μικρότερο στοιχείο και στο τέλος το μεγαλύτερο. Οι διάφοροι αλγόριθμοι ταξινόμησης κατατάσσονται ως προς την αποδοτικότητά τους ανάλογα με τον αριθμό των συγκρίσεων και το χώρο μνήμης που απαιτούν. Στην εργασία αυτή θα ασχοληθούμε με τον αλγόριθμο ταξινόμησης Insertion sort/in-place. Η ιδέα πίσω από αυτόν τον αλγόριθμο είναι η εξής: *Ξεκινώντας από το πρώτο στοιχείο πίνακα εισόδου, αναζητούμε το μικρότερο στοιχείο που υπάρχει στον πίνακα. Το τοποθετούμε στην αρχή, εναλλάσοντάς το με το στοιχείο που βρισκόταν στην αρχική θεση και επανεκτελούμε τη διαδικασία ξεκινώντας από την επόμενη θέση κάθε φορά, μέχρι να μείνει μόνο ένα στοιχείο.*
Η υλοποίηση του αλγορίθμου σε συμβολική γλώσσα μπορεί να ακολουθήσει το ακόλουθο διάγραμμα ροής :



Ας παρακολουθήσουμε μια επανάληψη του διαγράμματος ροής, για να καταλάβουμε καλύτερα πως λειτουργεί ο αλγόριθμος. Υποθέστε ότι ο πίνακάς μας αποτελείται από τα

εξής 6 στοιχεία : 0x45, 0x82, 0x34, 0xDA, 0x10, 0x28. Κατά τη πρώτη επανάληψη εκτελούνται τα εξής βήματα :

1. R0 = 0x45, R3 = 1, R2 = 0 (Θα συγκριθεί το 0x45 με το 0x82)
2. R0 = 0x45, R3 = 2, R2 = 0 (Θα συγκριθεί το 0x45 με το 0x34)
3. R0 = 0x34, R3 = 3, R2 = 2 (Θα συγκριθεί το 0x34 με το 0xDA)
4. R0 = 0x34, R3 = 4, R2 = 2 (Θα συγκριθεί το 0x34 με το 0x10)
5. R0 = 0x10, R3 = 5, R2 = 4 (Θα συγκριθεί το 0x10 με το 0x28)
6. R0 = 0x10, R3 = 6, R2 = 4

Μόλις τελειώσει η πρώτη επανάληψη του αλγορίθμου, στον R0 θα υπάρχει το μικρότερο στοιχείο του πίνακα και στον R2 η θέση του (με τον όρο θέση εννοείται η μετατόπιση από την αρχή του πίνακα). Ετσι, μετά το τέλος της πρώτης επανάληψης μεταφέρεται το πρώτο στοιχείο στη θέση [αρχή πίνακα + 4] του πίνακα και το μικρότερο (αυτό που υπάρχει στον R0) στην αρχή. Ο πίνακάς μας έχει τώρα τη μορφή : 0x10, 0x82, 0x34, 0xDA, 0x45, 0x28. Στην επόμενη επανάληψη θα αρχίσουμε τις συγκρίσεις μας από τα στοιχεία 0x82 & 0x34.

i. Υλοποίηση Insertion sort/in-place

Κατασκευάστε μια υπορουτίνα που να εκτελεί τη λειτουργία του Insertion sort/in-place. Η υπορουτίνα σας θα πρέπει να δέχεται σαν παραμέτρους τα εξής:

* Στον καταχωρητή R0 την αρχική διεύθυνση του πίνακα.
* Στον καταχωρητή R1 τον αριθμό των στοιχείων (σε bytes).

Η υπορουτίνα πρέπει να επενεργεί πάνω σε bytes και όχι σε words!

ii. Εκτέλεση αλγορίθμου και επιβεβαίωση ορθότητας αποτελεσμάτων

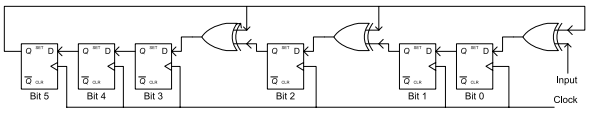
Για να διαπιστώσετε την ορθότητα της ρουτίνας χρησιμοποιήστε 20 τυχαίους αριθμούς και εφαρμόστε την υπορουτίνα ταξινόμησης πάνω σε αυτούς. Για να μπορέσετε να διαπιστώσετε αν όντως τα αποτελέσματα είναι σωστά, κατασκευάστε μια μικρή υπορουτίνα η οποία θα ξεκινά από το πρώτο στοιχείο, θα ελέγχει αν το επόμενο είναι μεγαλύτερο ή ίσο και θα μεταβαίνει στο επόμενο. Έτσι, αν το επόμενο στοιχείο του τρέχοντος είναι μικρότερο, θα σας ειδοποιήσει και θα γνωρίζετε αν ο αλγόριθμος ταξινόμησης είναι σωστός.

Εργασία 6

Σε αυτή την εργασία θα προσπαθήσουμε να παράγουμε ψευδοτυχαίους αριθμούς. Αυτό επιτυγχάνεται σε υλικό μέσω ενός κυκλώματος, γνωστού ως ολισθητής γραμμικής ανάδρασης Linear Feedback Shift Register (LFSR). Το κύκλωμα αυτό κατασκευάζεται από flip-flop (FF) και πύλες Exclusive OR (EOR) συνδεδεμένα με τη μορφή αλυσίδας με ανάδραση. Κάποια από τα FF απλά μεταφέρουν τα περιεχόμενά τους στο επόμενο της αλυσίδας, ενώ κάποια άλλα μέσω πυλών EOR στις οποίες συμμετέχει και η ανάδραση. Η τοποθέτηση των πυλών EOR δεν είναι τυχαία. Αποδεικνύεται ότι ανάλογα με τη θέση τους, το LFSR αντιστοιχεί σε ένα χαρακτηριστικό δυαδικό πολυώνυμο, οι μαθηματικές ιδιότητες του οποίου καθορίζουν την παραγωγή των τυχαίων αριθμών. Για παράδειγμα, για την υλοποποίηση του πολυωνύμου $x^6 + x^3 + x^2 + 1$, θα πρέπει να υπάρχουν :

- Λόγω του παράγοντα x^2 μία EOR που να οδηγεί την είσοδο του FF στη θέση 3 (Bit 2 της παρακάτω εικόνας)
- Λόγω του παράγοντα x^3 άλλη μία EOR που να οδηγεί την είσοδο του FF στη θέση 4 (Bit 3 της παρακάτω εικόνας).

Οι όροι 1 και x^6 αντιπροσωπεύονται με τη πράξη που οδηγεί το λιγότερο σημαντικό bitμέσω της ανάδρασης και των bits που εισάγονται στην είσοδο του κυκλώματος από εξωτερική πηγή.



Σκοπός μας είναι να περιγράψουμε με λογισμικό τη λειτουργία ενός LFSR. Το πολυώνυμο που θα χρησιμοποιήσουμε στην εργασία αυτή για την παραγωγή των τυχαίων αριθμών είναι το $x^{32} + x^{28} + x^{22} + x^{17} + x^{14} + x^9 + 1$, ή αλλιώς το 0x10424200 σε δεκαεξαδική αναπαράσταση (δε περιέχονται οι όροι x^{32} και 1). Το λογισμικό μας θα πρέπει σε κάθε επανάληψη να εκτελεί τα ακόλουθα βήματα :

- Αριστερή ολίσθηση κατά 1 θέση του καταχωρητή που έχουμε ορίσει σαν LFSR.
- Πρόσθεση του LFSR και του 0 με κρατούμενο ώστε να εισαχθεί στη λιγότερο σημαντική θέση το bit που είχε ολισθήσει εκτός.
- Πράξη EOR ανάμεσα στον LFSR και σε bit που ερχεται από εξωτερική πηγή. Προφανώς όταν η είσοδος από την εξωτερική πηγή είναι 0 δε χρειάζεται να γίνει αυτή η πράξη.
- Πράξη EOR ανάμεσα στον LFSR και στο χαρακτηριστικό πολυώνυμο.

Παράδειγμα

Έστω ότι στον καταχωρητή R1 υπάρχει ο αριθμός 0xCAFEBABA και το πολυώνυμο 0x10424200 είναι αποθηκευμένο στον καταχωρητή R0. Επίσης στον καταχωρητή R2 υπάρχει η τιμή της εξωτερικής εισόδου, την οποία υποθέτουμε 0. Με αριστερή ολίσθηση ο R1 γίνεται

0x95FD7574. Στο CPSR C flag υπάρχει το κρατούμενο εξόδου από την ολίσθηση και είναι 1. Προσθέτουμε τον R1 με το 0 και κρατούμενο και γίνεται 0x95FD7575. Θέλοντας να διατηρήσουμε μόνο το λιγότερο σημαντικό bit από τον R2 εκτελούμε την λογική πράξη AND ανάμεσα σε αυτόν και την τιμή 0x00000001 η οποία διατηρεί μόνο το λιγότερο σημαντικό bit και μηδενίζει όλα τα υπόλοιπα, οπότε έχουμε σαν αποτέλεσμα τον αριθμό 0. Εκτελούμε τη λογική πράξη EOR ανάμεσα στο 0 και το 0x95FD7575 και ο δεύτερος αριθμός παραμένει αμετάβλητος. Εκτελούμε τη λογική πράξη EOR ανάμεσα στον καταχωρητή R0 και τον R1(=0x95FD7575) και ο R1 γίνεται 0x85BF3775. Αν εφαρμόσουμε την ίδια διαδικασία στον R1 πάλι, θα πάρουμε ως αποτέλεσμα ένα διαφορετικό αριθμό, και μόνο μετά άπό $2^{32}-1$ επαναλήψεις θα καταλήξουμε στον ίδιο αριθμό που είχαμε βάλει αρχικά. Οποιαδήποτε επόμενη επανάληψη θα παράγει τους αριθμούς που παρήχθησαν και πρώτα με την ίδια ακριβώς σειρά (γι'αυτό και ονομάζονται ψευδοτυχαίοι).

i. Υλοποίηση LFSR

Κατασκευάστε μια υπορουτίνα που να εκτελεί τη λειτουργία του LFSR, και να δέχεται σαν παραμέτρους τα εξής:

* Στον καταχωρητή R0 το χαρακτηριστικό πολυώνυμο.
* Στον καταχωρητή R1 τον αρχικό αριθμό.

Μετά την ολοκλήρωση της εκτέλεσης της υπορουτίνας, ο R1 πρέπει να περιέχει τον ανανεωμένο αριθμό, ενώ ο R0 να παραμείνει αμετάβλητος. Παρατηρείστε πως δεν θα χρησιμοποιήσουμε αριθμούς από εξωτερική πηγή, οπότε στη θέση τους θα εισάγουμε το 0.

ii. Χρήση LFSR για παραγωγή τυχαίων αριθμών

Τώρα κατασκευάστε ένα πρόγραμμα που να καλεί την υπορουτίνα του προηγούμενου υποερωτήματος για ένα καθορισμένο αριθμό επαναλήψεων και να αποθηκεύει τους τυχαίους αριθμούς σε διαδοχικές θέσεις μνήμης. Οι παράμετροι είναι οι εξής:

* Στον καταχωρητή R3 θα πρέπει να αποθηκεύεται η αρχική διεύθυνση του χώρου αποθήκευσης.
* Στον καταχωρητή R2 θα πρέπει να αποθηκεύεται ο αριθμός επαναλήψεων.

Μπορείτε να ξεκινήσετε με όποιον αρχικό αριθμό επιθυμείτε, εκτός του 0. Αφού παράγετε 20 τυχαίους αριθμούς, καταγράψτε τους στον ακόλουθο πίνακα.

iii. Χρήση LFSR εύρους 4 bit για παραγωγή τυχαίων αριθμών

Τροποποιήστε τον κώδικα του πρώτου υποερωτήματος, ώστε να χρησιμοποιεί το πολυώνυμο x^4+x^3+1, (το οποίο σε δεκαεξαδική αναπαράσταση είναι το 0x8). Όλες οι πράξεις σας πρέπει να εκτελούνται στα 8 bits, ενώ τα τα αποτελέσματά σας είναι των 4 bits. Παράγετε 20 τυχαίους αριθμούς και καταγράψτε τους στον ακόλουθο πίνακα. Μετά από πόσους αριθμούς αρχίζει η ακολουθία να επαναλαμβάνεται;

Υπόδειξη

Ο αλγόριθμος του LFSR των 4 bits παρουσιάζει την ιδιομορφία πως το περισσότερο σημαντικό bit του LFSR δεν ολισθαίνει στο carry bit του καταχωρητή κατάστασης. Εξετάστε συνεπώς το πως μπορείτε να εξετάσετε τη κατάσταση αυτού του ψηφίου με την εντολή tst.