ΕΡΓΑΣΤΗΡΙΟ ΒΑΣΙΚΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Βασιλική Ευαγγελία Δούρου- Α.Μ.:1072633

Παύλος Πεσκελίδης- Α.Μ.:1072483

**ΕΡΓΑΣΙΑ 4:**

1. Ο κώδικας για την υπορουτίνα είναι ο ακόλουθος:

Subrtn:

STMDB R13!, {R2-R7, R9-R10}

LDRB R2, [R0, #0]

LDRB R3, [R0, #1]

LDRB R4, [R0, #2]

LDRB R5, [R1, #0]

LDRB R6, [R1, #1]

LDRB R7, [R1, #2]

MOV R9, R0

MOV R10, R1

MUL R0, R2, R5

MUL R2, R3, R6

MUL R3, R4, R7

ADD R0, R0, R2

SUB R0, R0, R3

MOV R2, #5

MUL R1, R0, R2

MOV R0, R1, LSR #6

STRB R0, [R8]

MOV R0, R9

MOV R1, R10

LDMIA R13!, {R2-R7, R9-R10}

MOV PC,LR

Αρχικά, δεσμεύουμε τους καταχωρητές που πρόκειται να χρησιμοποιήσουμε στη μνήμη, επειδή οι R0, R1, R8 χρησιμοποιούνται από το κύριο πρόγραμμα (που θα γραφεί παρακάτω) για την προσπέλαση των περιεχομένων των Values, Const και Result, δεν θα δεσμευτούν στην υπορουτίνα μας. Έπειτα, μεταφέρουμε στον R2 το περιεχόμενο της διεύθυνσης που δείχνει ο R0, στον R3 το περιεχόμενο που δείχνει ο R0 + 1 και στον R4 το περιεχόμενο που δείχνει ο R0 + 2, χωρίς να αλλάξουμε σε καμία από αυτές τις εντολές το R0. Στη συνέχεια κάνουμε το ίδιο και με τους R5, R6 και R7 μεταφέροντας τους το περιεχόμενο της θέσης μνήμης που δείχνει ο R1, R1 + 1 και R1 + 2 αντίστοιχα, χωρίς να αλλάξουμε την τιμή του R1. Επιπλέον, μεταφέρουμε στον R9 το περιεχόμενο του R0 και στον R10 το περιεχόμενο του R1, γιατί μετά από κάθε κλήση της υπορουτίνας θέλουμε τους R0, R1 να έχουν το αρχικό τους περιεχόμενο. Μετά πολλαπλασιάζουμε τον R2 με τον R5 και αποθηκεύουμε το αποτέλεσμα στον R0, τον R3 με τον R6 και αποθηκεύουμε το αποτέλεσμα στον R2 και τον R4 με τον R7 και αποθηκεύουμε το αποτέλεσμα στον R3. Με αυτόν τον τρόπο, έχουμε αποθηκευμένους τους όρους ai ∗ z0, bi ∗ z1, ci ∗ z2 του μαθηματικού μας τύπου στους καταχωρητές R0, R2, R3 αντίστοιχα. Έπειτα, προσθέτουμε τον R0 με τον R2 και αποθηκεύουμε το αποτέλεσμα στον R0, ο οποίος πλέον περιέχει τον όρο ai ∗ z0 + bi ∗ z1, και μετά αφαιρούμε από τον R0 τον R3 και αποθηκεύουμε και αυτό το αποτέλεσμα στον R0, ο οποίος περιέχει πλέον τον όρο ai ∗ z0 + bi ∗ z1 - ci ∗ z2. Ακολούθως, μεταφέρουμε στον R2 την τιμή 5, πολλαπλασιάζουμε τον R2 με τον R0 και αποθηκεύουμε το αποτέλεσμα στον R1, που περιέχει τώρα το 5\*(ai∗z0+bi∗z1-ci∗z2). Ακόμη, μεταφέρουμε στον R0 το περιεχόμενο του R1 ολισθημένο προς τα δεξιά 6 φορές, αφού η κάθε ολίσθηση προς τα δεξιά ισοδυναμεί με διαίρεση του αριθμού με το δύο και άρα μετά από 6 ολισθήσεις το περιεχόμενο του R1 θα έχει διαιρεθεί με το 64, και έτσι ο R0 θα περιέχει τον όρο 5\*(ai∗z0+bi∗z1-ci∗z2)/64 που είναι και το αποτέλεσμα μας. Τέλος, μεταφέρουμε στον R0 και στον R1 τις αρχικές τιμές τους από τους R9, R10 αντίστοιχα, αποδεσμεύουμε τους καταχωρητές που χρησιμοποιήσαμε στην υπορουτίνα και μεταφέρουμε τον PC στο σημείο όπου κλήθηκε η υπορουτίνα στη main.

Το ολοκληρωμένο πρόγραμμα θα είναι το ακόλουθο:

.arm

.text

.global main

main:

STMDB R13!, {R0-R12, R14}

LDR R0, =Values

LDR R1, =Const

LDR R8, =Result

Loop:

BL Subrtn

ADD R0, R0, #3

ADD R8, R8, #1

CMP R0, R1

BLO Loop

LDMIA R13!, {R0-R12, PC}

Subrtn:

STMDB R13!, {R2-R7, R9-R10}

LDRB R2, [R0, #0]

LDRB R3, [R0, #1]

LDRB R4, [R0, #2]

LDRB R5, [R1, #0]

LDRB R6, [R1, #1]

LDRB R7, [R1, #2]

MOV R9, R0

MOV R10, R1

MUL R0, R2, R5

MUL R2, R3, R6

MUL R3, R4, R7

ADD R0, R0, R2

SUB R0, R0, R3

MOV R2, #5

MUL R1, R0, R2

MOV R0, R1, LSR #6

STRB R0, [R8]

MOV R0, R9

MOV R1, R10

LDMIA R13!, {R2-R7, R9-R10}

MOV PC,LR

.data

Values:

.byte 0x02, 0x03, 0x04

.byte 0x10, 0x05, 0x06

.byte 0x0B, 0x02, 0x0D

.byte 0x01, 0x0C, 0x08

Const:

.byte 0x04, 0x07, 0x05

Result:

.byte 0,0,0,0

Στην main, αρχικά, δεσμεύουμε τους καταχωρητές που πρόκειται να χρησιμοποιήσουμε και φορτώνουμε στον καταχωρητή R0 τη διεύθυνση της ετικέτας Values, στον R1 τη διεύθυνση της ετικέτας Const και στον R8 τη διεύθυνση της ετικέτας Result. Έπειτα, ξεκινάμε ένα loop, το οποίο θα καλεί την υπορουτίνα και μετά θα αυξάνει το περιεχόμενο του R0 κατά 3 για να δείχνει στην επόμενη τριάδα ai, bi, ci και το περιεχόμενο του R8 κατά 1 για να εγγραφεί εκεί το αποτέλεσμα της κλήσης της επόμενης υπορουτίνας. Η δομή επανάληψης θα πραγματοποιείται όσο το R0 είναι μικρότερο του R1. Στο τέλος του προγράμματος αποδεσμεύονται όλοι οι καταχωρητές που χρησιμοποιήθηκαν από τη μνήμη.

Ο πίνακας συμπληρώνεται όπως ακολουθεί:

|  |  |
| --- | --- |
| Επανάληψη | Αποτέλεσμα |
| 1 | 00 |
| 2 | 05 |
| 3 | FF |
| 4 | 03 |

1. Ο νέος κώδικας είναι ο ακόλουθος:

.arm

.text

.global main

main:

STMDB R13!, {R0-R12, R14}

LDR R0, =Values

LDR R1, =Const

LDR R8, =Result

Loop:

BL Subrtn

ADD R0, R0, #3

ADD R8, R8, #1

CMP R0, R1

BLO Loop

LDR R8, =Result

LDR R2, =Result + 4

LDR R6, =Result + 1

MOV R5, #0

SecondLoop:

LDRSB R3, [R8]

LDRSB R4, [R6]

CMP R4, R3

MOVGT R3, R4

STRB R3, [R8]

ADDGT R5, R5, #1

ADD R6, R6, #1

CMP R6, R2

BLO SecondLoop

STRB R3, [R1, #3]

STRB R5, [R1, #4]

LDMIA R13!, {R0-R12, PC}

Subrtn:

STMDB R13!, {R2-R7, R9-R10}

LDRB R2, [R0, #0]

LDRB R3, [R0, #1]

LDRB R4, [R0, #2]

LDRB R5, [R1, #0]

LDRB R6, [R1, #1]

LDRB R7, [R1, #2]

MOV R9, R0

MOV R10, R1

MUL R0, R2, R5

MUL R2, R3, R6

MUL R3, R4, R7

ADD R0, R0, R2

SUB R0, R0, R3

MOV R2, #5

MUL R1, R0, R2

MOV R0, R1, LSR #6

STRB R0, [R8]

MOV R0, R9

MOV R1, R10

LDMIA R13!, {R2-R7, R9-R10}

MOV PC,LR

.data

Values:

.byte 0x02, 0x03, 0x04

.byte 0x10, 0x05, 0x06

.byte 0x0B, 0x02, 0x0D

.byte 0x01, 0x0C, 0x08

Const:

.byte 0x04, 0x07, 0x05

.byte 0,0

Result:

.byte 0,0,0,0

Σε σχέση με τον αρχικό κώδικα, μετά το τέλος του πρώτου Loop φορτώνουμε στον καταχωρητή R8 πάλι τη διεύθυνση της ετικέτας Result, στον R2 τη διεύθυνση της Result + 4, στον R6 τη διεύθυνση της Result + 1 και στον R5 μεταφέρουμε την τιμή 0. Έτσι, με τον R6 θα διατρέχουμε τον πίνακα των αποτελεσμάτων και με τον R5 θα μετράμε τη θέση που θα βρίσκεται το μεγαλύτερο αποτέλεσμα. Έπειτα ξεκινάει μία δεύτερη δομή επανάληψης που ονομάζεται SecondLoop, στην οποία φορτώνουμε στους καταχωρητές R3, R4 τα περιεχόμενα των θέσεων μνήμης που δείχνουν οι R8, R6 αντίστοιχα, χρησιμοποιώντας όμως την εντολή LDRSB αντί της LDRB, καθώς θέλουμε στη μεταφορά να έχει διατηρηθεί και το πρόσημο των αριθμών. Μετά συγκρίνουμε τα R3 και R4 και αν το περιεχόμενο του R4 είναι μεγαλύτερο από αυτό του R3, το μεταφέρουμε στον R3 και φορτώνουμε μετά την τιμή του R3 πάλι στο R8 ως byte. Έπειτα, αν είναι μεγαλύτερο το περιεχόμενο του R4, αυξάνουμε και το περιεχόμενο του R5 κατά ένα. Τέλος, είτε είναι μεγαλύτερο είτε μικρότερο το R4 από το R3, αυξάνουμε την τιμή του R6 κατά ένα ώστε να προχωρήσουμε στην επόμενη σύγκριση. Η δομή επανάληψης τελειώνει όταν το R6 γίνει ίσο μεγαλύτερο από το R2, δηλαδή όταν έχει τελειώσει ο πίνακας των αποτελεσμάτων μας. Μετά το τέλος της επανάληψης μεταφέρουμε το περιεχόμενο του R3, το οποίο θα είναι το μέγιστο αποτέλεσμα μας, αφού σε κάθε επανάληψη μεταφέρουμε τον μεγαλύτερο αριθμό στη θέση μνήμης που δείχνει ο R8 και μετά βάζουμε στον R3 το περιεχόμενο που δείχνει αυτός, 3 θέσεις μετά από εκείνη που δείχνει ο R1, δηλαδή στην πρώτη θέση μετά το τέλος των σταθερών όρων του τύπου μας, και το περιεχόμενο του R5 μία θέση μετά από αυτή. Επίσης, έχουμε κάνει και μία αλλαγή στα .data έχοντας δεσμεύσει δύο παραπάνω θέσεις μνήμης, μία για το μεγαλύτερο αποτέλεσμα και μία για τη θέση του.

1. Ο κώδικας του ερωτήματος είναι ο ακόλουθος:

.arm

.text

.global main

main:

STMDB R13!, {R0-R12, R14}

LDR R0, =Values

LDR R1, =Const

LDR R2, =Result

LDR R3, =Const + 6

LDR R8, =Values + 16

Loop:

BL Subroutine

LDR R3, =Const + 6

ADD R0, R0, #4

CMP R0, R8

BLO Loop

LDMIA R13!, {R0-R12, PC}

Subroutine:

STMDB R13!, {R4-R7}

LDR R4, [R0]

LDRSB R5, [R3]

SecondLoop:

SUB R3, R3, #1

LDRSB R6, [R3]

MLA R7, R5, R4, R6

MOV R5, R7

CMP R1, R3

BLO SecondLoop

STR R7, [R2, #4]!

LDMIA R13!, {R4-R7}

MOV PC,LR

.data

Values:

.word 0x10

.word 0x50A

.word 0xCDCA

.word 0x80AB

Const:

.byte 0x04, 0x07, 0x05

.byte 0x20, 0x1A, 0x12, 0x06

Result:

.word 0, 0, 0, 0

**Σχόλιο:** Σύμφωνα με την ανακοίνωση σχετικά με την άσκηση, τα αποτελέσματα του προγράμματος μας χρησιμοποιούν 32 bit για την αποθήκευση τους στο τρίτο ερώτημα.

Αρχικά, δεσμεύουμε τους καταχωρητές που πρόκειται να χρησιμοποιήσουμε στη μνήμη. Έπειτα, φορτώνουμε στον καταχωρητή R0 τη διεύθυνση που δείχνει η ετικέτα Values, και έτσι με τον R0 θα διατρέχουμε τα x που έχουμε, στον R1 τη διεύθυνση που έχει η ετικέτα Const ώστε με αυτόν να διατρέχουμε τους σταθερούς όρους, στον R2 τη διεύθυνση που έχει η ετικέτα Result, στον R3 τη διεύθυνση του Const + 6, καθώς στον τύπο που χρησιμοποιούμε ξεκινάμε από την σταθερά α6 και φτάνουμε στην α0 και ώστε να ξέρουμε πότε θα τελειώσει το SecondLoop για τον υπολογισμό του κάθε αποτελέσματος, και στον R8 αποθηκεύουμε τη διεύθυνση του Values + 16, καθώς το x μας είναι σε word και θέλουμε το πρώτο Loop να τελειώσει όταν τελειώσουν όλα τα x. Μετά ξεκινάει το πρώτο Loop που τελειώνει όταν το R0 γίνει ίσο με το R8, δηλαδή όταν έχουν διαβαστεί όλα τα x. Στη διάρκεια αυτού του πρώτου Loop εκτελείται η υπορουτίνα, επαναφέρεται η αρχική τιμή του R3 και αυξάνεται η τιμή του R0 κατά 4 ώστε να δείχνει στο επόμενο word προς ανάγνωση. Μετά το τέλος του Loop επαναφέρονται και οι αρχικές τιμές σε όλους τους καταχωρητές μας και τελειώνει το πρόγραμμα. Όσο αφορά την υπορουτίνα μας, στην αρχή δεσμεύουμε τους καταχωρητές που πρόκειται να χρησιμοποιήσουμε σε αυτή, φορτώνουμε τον καταχωρητή R4 με το περιεχόμενο της διεύθυνσης που δείχνει ο R0, δηλαδή με το x μας, και τον R5 με το περιεχόμενο που δείχνει ο R3, δηλαδή με το α6 μας που ισούται με το b6. Έπειτα, ξεκινάει ένα δεύτερο loop στη διάρκεια του οποίου, αφαιρούμε 1 από τον R3 και έτσι δείχνει στην αμέσως προηγούμενη θέση, φορτώνουμε το περιεχόμενο που δείχνει τώρα ο R3 στον R6, το οποίο έχει τώρα το αi-1, πολλαπλασιάζουμε τον R5 με τον R4, προσθέτουμε στο γινόμενο το περιεχόμενο του R6 και μεταφέρουμε το αποτέλεσμα στον R7. Μετά μεταφέρουμε το αποτέλεσμα στον R5, αφού αυτός θα έχει τα bi μας. Το δεύτερο loop τελειώνει όταν ο R3 γίνει ίσος με τον R1, δηλαδή όταν τελειώσουν οι σταθερές μας. Έπειτα, μεταφέρεται στη μνήμη το αποτέλεσμα, που θα το έχει ο R7 σε μορφή word και αποδεσμεύονται οι καταχωρητές που χρησιμοποιήθηκαν.