

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

1Η ΑΣΚΗΣΗ

2 /4 /2021

Αντώνης Παρλαπάνης ΑΜ: 1059709

Κωνσταντίνα Υφαντή ΑΜ: 1054972

# Πείραμα 1

Ο κώδικας της άσκησης είναι ο ακόλουθος:

|  |
| --- |
| .file "HelloWorld.S"  .global \_start  .text  .code 32  @ Register definition, for ARM VersatilePB with PL011 UART  .equ UART0\_BASE, 0x101f1000 //διεύθυνση στη μνήμη για αποθήκευση  .equ UARTDR, 0x0 //βήμα  \_start:  @ Display 'str' to the UART:  ldr r0, =(str-1) //αποθηκεύει στον καταχωρητή r0 το string  ldr r1, =UART0\_BASE //αποθηκεύει στον καταχωρητή r1 την διεύθυνση  1: ldrb r2, [r0, #1]! //φορτώνει στον r2 με βήμα 1 κάθε byte του string που περιέχεται στον r0  cmp r2, #0 //ελέγχει αν ο r2 είναι 0  beq . //αν είναι 0 μεταβαίνει στο .end  str r2, [r1, #UARTDR] //αποθηκεύει το περιεχόμενο του r2 στη θέση μνήμης του r1 αφού το #UARTDR = 0  b 1b //θα ψάξει προς τα πίσω το 1 label για να συνεχίσει η επαναληπτική διαδικασία  str: .asciz "Hello world!\n"  .end |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνονται οι λειτουργίες που πραγματοποιούνται σε κάθε εντολή.

Στον παραπάνω κώδικα ...

# Αποτελέσματα

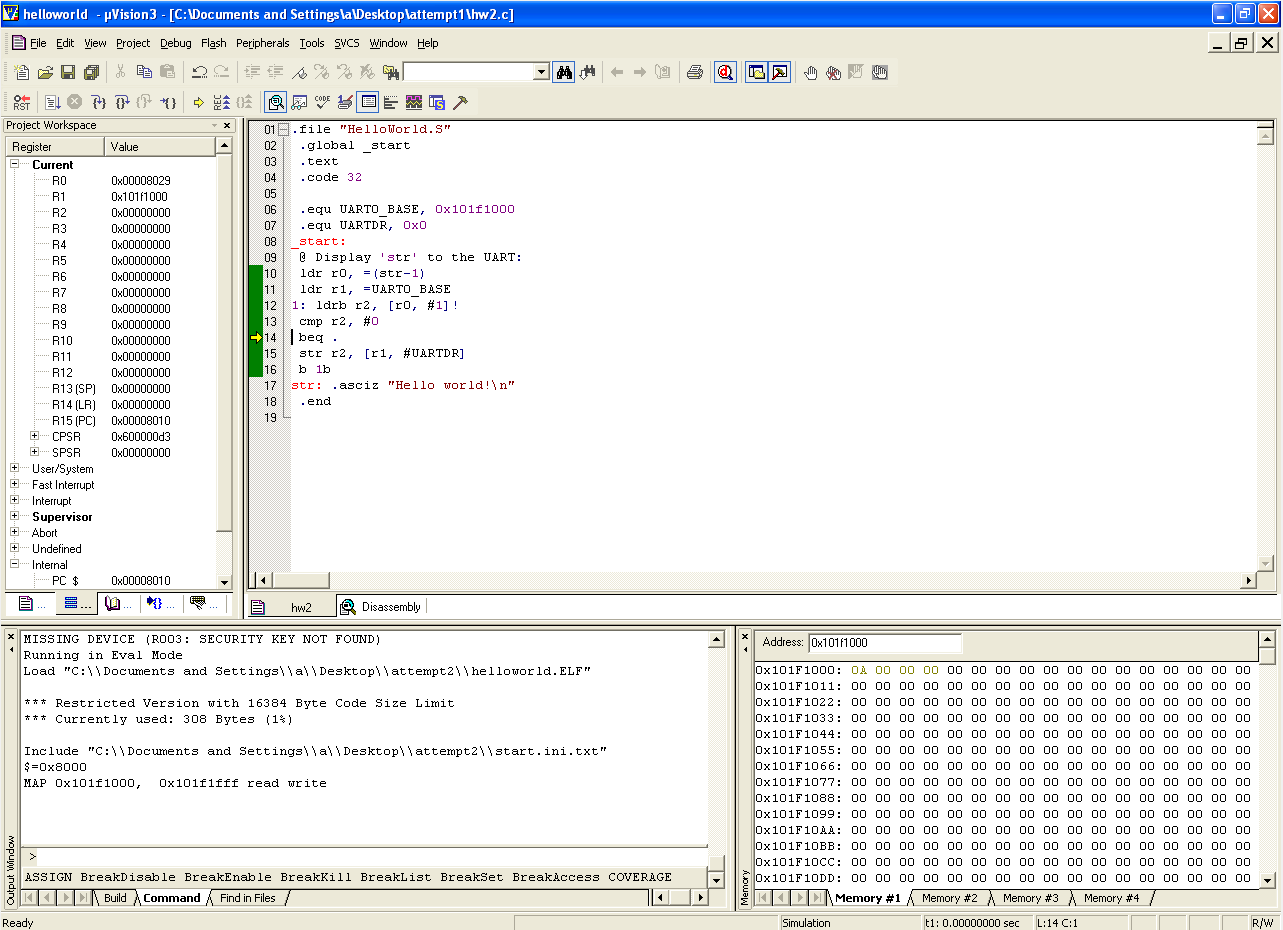
Στον προσομοιωτή παρατηρήσαμε ότι …

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Συμπεράσματα

Ο παραπάνω κώδικας διαβάζει το string "Hello world!\n" και αποθηκεύει τους χαρακτήρες έναν έναν στην ίδια θέση μνήμης κάνοντας rewrite.Στο τέλος του προγράμματος βλέπουμε ότι θέση μνήμης 0x101F1000 υπάρχει η τιμή 0Α , όπου αντιπροσωπεύει το line separator στο ascii.



# Πείραμα 2

Ο κώδικας της άσκησης είναι ο ακόλουθος:

|  |
| --- |
| //pointer στη θέση μνήμης που θα γίνει η αποθήκευση  volatile unsigned int \* const Uart\_DR = (unsigned int \*)0x101f1000;  void \_start() {  char \*s = "Hello world!\n";  while (\*s != '\0') //επαναλητπική διαδικασία μέχρι τον τελευταίο χαρακτήρα  \*Uart\_DR = (unsigned int)(\*s++); //στη θέση μνήμης που δείχνει ο pointer αποθηκεύονται οι χαρακτήρες  while (1) ;  } |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνονται οι λειτουργίες που πραγματοποιούνται σε κάθε εντολή.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε ότι …

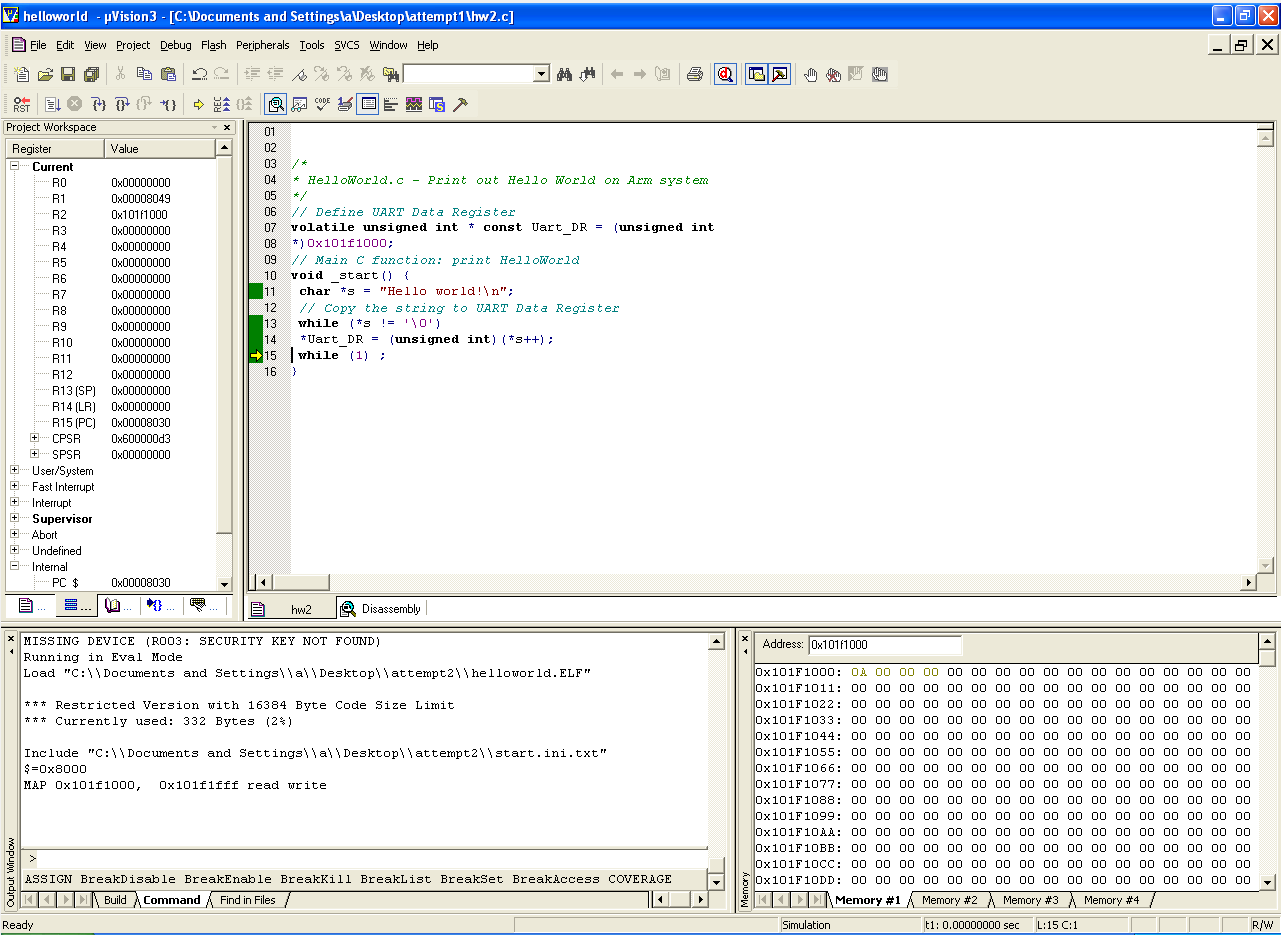
Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Συμπεράσματα

1. Όπως και στο Πείραμα 1, ο παραπάνω κώδικας διαβάζει το string "Hello world!\n" και αποθηκεύει τους χαρακτήρες έναν έναν στην ίδια θέση μνήμης κάνοντας rewrite.
2. Η assembly που παράγεται από το πρόγραμμα σε c είναι κατά 5 γραμμές μεγαλύτερη από την assembly που τρέξαμε αρχικά. Επομένως μπορούμε να συμπεράνουμε ότι ο χρόνος εκτέλεσης είναι μικρότερος στην περίπτωση της assembly. Όσο αναφορά την ευκολία προγραμματισμού, στην C , έχουμε μεγαλύτερη ευκολία λόγο εξοικείωσης με τη προγραμματιστική λογική.

Ομοίως με το πείραμα ένα στο τέλος του προγράμματος υπάρχει η τιμή ascii του line separator στη μνήμη.



Σημείωση: Εκ παραδρομής υλοποιήσαμε τα ερωτήματα 3 και 4 σε c αλλά και assembly, οπότε παρακάτω παραθέτουμε και τις 2 υλοποιήσεις.

# Πείραμα 3

Ο κώδικας της άσκησης υλοποιημένος σε c είναι ο ακόλουθος, με χρώμα μπλε φαίνονται οι αλλαγές που κάναμε σε σχέση με το κώδικα του Πειράματος 2:

|  |
| --- |
| volatile unsigned int \* const Uart\_DR = (unsigned int \*)0x101f1000;  void \_start() {  char \*s = "Hello world!\n";  while (\*s != '\0')  \*Uart\_DR++ = (unsigned int)(\*s++); //αυξάνουμε τον pointer που δείχνει τις θέσεις μνήμης που θα αποθηκευτεί το string 1 θέση  while (1) ;  } |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνεται η λειτουργία της εντολής που αλλάξαμε καθώς τα υπόλοιπα είναι ίδια με το Πείραμα 2.

Όπως φαίνεται, χρειάστηκε απλώς να προχωράμε το δείκτη που έχει τη διεύθυνση μνήμης μια θέση μπροστά σε κάθε επανάληψη.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Συμπεράσματα

Ο παραπάνω κώδικας διαβάζει το string "Hello world!\n" και αποθηκεύει τους χαρακτήρες έναν έναν σε διαδοχικές θέσεις μνήμης.

Σε αντίθεση με την υλοποίηση σε assembly που φαίνεται παρακάτω, οι τροποποιήσεις που χρειάστηκαν στην c ήταν πολύ πιο απλές.

Ο κώδικας της άσκησης υλοποιημένος σε assembly είναι ο ακόλουθος, με χρώμα μπλε φαίνονται οι αλλαγές που κάναμε σε σχέση με το κώδικα του πειράματος 1:

|  |
| --- |
| .file "HelloWorld.S"  .global \_start  .text  .code 32  @ Register definition, for ARM VersatilePB with PL011 UART  .equ UART0\_BASE, 0x101f1000  .equ UARTDR, 0x4 //το βήμα γίνεται 4 για σωστή στοίχιση της μνήμης  \_start:  @ Display 'str' to the UART:  ldr r0, =(str-1)  ldr r1, =UART0\_BASE  1: ldrb r2, [r0, #1]!  cmp r2, #0  beq .  str r2, [r1], #UARTDR //πρώτα αποθηκεύουμε το περιεχόμενο του r2 στην θέση μνήμης που δείχνει ο r1 και μετά τον αυξάνουμε κατά #UARTDR=4  b 1b  str: .asciz "Hello world!\n"  .end |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνεται η λειτουργία των εντολών που αλλάξαμε καθώς τα υπόλοιπα είναι ίδια με το Πείραμα 1.

Αρχικά, χρειάστηκε να αλλάξουμε το βήμα κατά 4, ώστε να γίνεται σωστή στοίχιση της μνήμης, καθώς η ldr που βρίσκεται παρακάτω στον κώδικα μεταφέρει δεδομένα 32-bits από την εξωτερική μνήμη στους καταχωρητές. Στη συνέχεια, φροντίσαμε πρώτα να γίνεται η προσπέλαση στην μνήμη με την str και μετά να μετατοπίζεται ο καταχωρητής.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text, application

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Συμπεράσματα

Ο παραπάνω κώδικας διαβάζει το string "Hello world!\n" και αποθηκεύει τους χαρακτήρες έναν έναν σε διαδοχικές θέσεις μνήμης.

# Πείραμα 4

Ο κώδικας της άσκησης υλοποιημένος σε c είναι ο ακόλουθος, με χρώμα μπλε φαίνονται οι αλλαγές που κάναμε σε σχέση με το κώδικα του πειράματος 2:

|  |
| --- |
| //δήλωση της διεύθυνσης σε pointer τύπου char  volatile unsigned char \* const Uart\_DR = (unsigned int \*)0x101f1000;  void \_start() {  char \*s = "Hello world!\n";  while (\*s != '\0')  \*Uart\_DR++ = (unsigned int)(\*s++); //αυξάνουμε τον pointer που δείχνει τις θέσεις μνήμης που θα αποθηκευτεί το string 1 θέση  while (1) ;  } |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνεται η λειτουργία των εντολών που αλλάξαμε καθώς τα υπόλοιπα είναι ίδια με το Πείραμα 3α.

Αρχικά, χρειάστηκε να αλλάξουμε τον pointer που δείχνει την διεύθυνση στη μνήμη που θα αποθηκευτούν οι χαρακτήρες σε char ώστε να γίνεται αποθήκευση ανά byte. Όπως και στο Πείραμα 3α , προχωράμε το δείκτη που έχει τη διεύθυνση μνήμης μια θέση μπροστά σε κάθε επανάληψη.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Διορθώσεις

…

# Συμπεράσματα

Ο παραπάνω κώδικας διαβάζει το string "Hello world!\n" και αποθηκεύει τους χαρακτήρες σε διαδοχικές θέσεις byte.

Ο κώδικας της άσκησης υλοποιημένος σε assembly είναι ο ακόλουθος, με χρώμα μπλε φαίνονται οι αλλαγές που κάναμε σε σχέση με το κώδικα του πειράματος 1:

|  |
| --- |
| .file "HelloWorld.S"  .global \_start  .text  .code 32  @ Register definition, for ARM VersatilePB with PL011 UART  .equ UART0\_BASE, 0x101f1000  .equ UARTDR, 0x1 //κάνουμε το βήμα 1  \_start:  @ Display 'str' to the UART:  ldr r0, =(str-1)  ldr r1, =UART0\_BASE  1: ldrb r2, [r0, #1]!  cmp r2, #0  beq .  strb r2, [r1], #UARTDR //αποθήκευση στη μνήμη σε μορφή byte  b 1b  str: .asciz "Hello world!\n"  .end |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνεται η λειτουργία των εντολών που αλλάξαμε, καθώς τα υπόλοιπα είναι ίδια με το Πείραμα 3β.

Αρχικά, χρειάστηκε να κάνουμε το βήμα 1 καθώς θέλουμε διαδοχικά bytes, όχι θέσεις μνήμης. Στη συνέχεια, χρησιμοποιήσαμε την εντολή strb που μεταφέρει δεδομένο 8-bits από τον καταχωρητή στην εξωτερική μνήμη. Η προσπέλαση και μετατόπιση της διεύθυνσης έγινε με τον ίδιο τρόπο του Πειράματος 3β.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text, application

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Διορθώσεις

…

# Συμπεράσματα

Ο παραπάνω κώδικας διαβάζει το string "Hello world!\n" και αποθηκεύει τους χαρακτήρες σε διαδοχικά bytes στην εξωτερική μνήμη.

# Πείραμα 5

Ο κώδικας της άσκησης, υλοποιημένος σε c είναι ο ακόλουθος:

|  |
| --- |
| //ορισμός διαδοχικών διευθύνσεων μνήμης  volatile unsigned int \* const A = (unsigned int \*)0x101f1000;  volatile unsigned int \* const B = (unsigned int \*)0x101f1004;  volatile unsigned int \* const C = (unsigned int \*)0x101f1008;  void \_start() {  //αρχικοποίηση  \*B = 2;  \*C = 3;  //πράξη και αποθήκευση  \*A = \*B + \*C;  } |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνονται οι λειτουργίες που πραγματοποιούνται σε κάθε εντολή.

Αρχικά, χρειάστηκε να δηλώσουμε τις διευθύνσεις που θα αποθηκευθούν οι τιμές στη μνήμη. Επιλέξαμε να είναι διαδοχικές, επομένως είναι σε πολλαπλάσια του 4. Όπως φαίνεται, αρχικοποιήσαμε με τυχαίες τιμές τις θέσεις μνήμης που δείχνουν οι pointers B και C· και μετά αποθηκεύσαμε το άθροισμα τους στη θέση μνήμης που δείχνει ο pointer A.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Διορθώσεις

…

# Συμπεράσματα

Ο παραπάνω κώδικας εκτελεί το άθροισμα A = B + C και αποθηκεύει τις τιμές σε διαδοχικές θέσεις μνήμης.

# Πείραμα 6

Ο κώδικας της άσκησης είναι ο ακόλουθος:

|  |
| --- |
| //ορισμός διαδοχικών διευθύνσεων μνήμης  volatile unsigned int \* Y = (unsigned int \*)0x101f1000;  volatile unsigned int \* K = (unsigned int \*)0x101f1004;  void \_start() {  int i;  for(i=0; i<10; i++) {  \*K = 10; //για ευκολία ελέγχου του αποτελέσματος  \*K++; //διαδοχικές θέσεις μνήμης  }  int z;  for(z=0; z<=10; z++){  \*Y = \*Y + \*K; //πράξη και αποθήκευση  \*K--; //από την τελική στην αρχική θέση μνήμης  }  } |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνονται οι λειτουργίες που πραγματοποιούνται σε κάθε εντολή.

Αρχικά, χρειάστηκε να δηλώσουμε τις διευθύνσεις που θα αποθηκευθούν οι τιμές στη μνήμη. Επιλέξαμε να είναι διαδοχικές, επομένως είναι σε πολλαπλάσια του 4. Στην πρώτη for, αρχικοποιήσαμε με τη τιμή 10 τις θέσεις μνήμης που αντιστοιχούν στα k σε 10 διαδοχικές θέσεις. Στη συνέχεια, με μια δεύτερη for εκτελέσαμε την ζητούμενη πράξη και αποθηκεύσαμε το αποτέλεσμα στη θέση μνήμης που δείχνει ο pointer Y. Καθώς στο τέλος της πρώτης for ο Κ έχει φτάσει στην 11η θέση μνήμης η δεύτερη for είναι για z<=10 και ο K μετακινείται σε κάθε βήμα μια θέση προς τα πίσω.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Διορθώσεις

…

# Συμπεράσματα

Ο παραπάνω κώδικας εκτελεί την ζητούμενη πράξη αποθηκεύοντας στην πρώτη θέση μνήμης που δηλώνεται το αποτέλεσμα και σε επόμενες διαδοχικές τις τιμές του k.

# Πείραμα 7

Ο κώδικας της άσκησης είναι ο ακόλουθος:

|  |
| --- |
| //ορισμός διαδοχικών διευθύνσεων μνήμης  volatile unsigned int \* Y = (unsigned int \*)0x101f1000;  volatile unsigned int \* Α = (unsigned int \*)0x101f1004;  volatile unsigned int \* Β = (unsigned int \*)0x101f1008;  void \_start() {  int i;  for(i=0; i<10; i++) {  //για ευκολία ελέγχου του αποτελέσματος  \*Α = 1;  \*Β = 2;  //εκτέλεση πράξης και εκχώρηση στην μνήμη  \*Y = \*Y + (\*A \* \*B);  //αύξηση των θέσεων μνήμης και διατήρηση της διαδοχικότητας  A = A + 2;  B = B + 2;  }  } |

Παρατηρήσεις σχετικά με τον κώδικα:

Στα σχόλια πάνω φαίνονται οι λειτουργίες που πραγματοποιούνται σε κάθε εντολή.

Αρχικά, χρειάστηκε να δηλώσουμε τις διευθύνσεις που θα αποθηκευθούν οι τιμές στη μνήμη. Επιλέξαμε να είναι διαδοχικές, επομένως είναι σε πολλαπλάσια του 4. Όλες οι ζητούμενες διαδικασίες υλοποιούνται σε μια for. Αρχικοποιήσαμε για ευκολία στον έλεγχο με τις τιμές 1 και 2 τις θέσεις μνήμης που αντιστοιχούν στα Α και Β αντίστοιχα. Στη συνέχεια, εκτελέσαμε την ζητούμενη πράξη και αποθηκεύσαμε το αποτέλεσμα στη θέση μνήμης που δείχνει ο pointer Y. Καθώς τα A και Β πρέπει να αποθηκεύονται με τη συγκεκριμένη σειρά σε διαδοχικές θέσεις μετακινούνται σε κάθε βήμα δυο θέση μετά.

# Αποτελέσματα

Στον προσομοιωτή παρατηρήσαμε τα ακόλουθα αποτελέσματα:

Graphical user interface, text, application

Description automatically generated

Στο εργαστήριο θα χρειαζόταν να τρέξουμε τις παρακάτω εντολές:

1. mw 0 0xFFFFF400 0x02
2. …
3. …
4. …
5. …
6. …
7. …
8. …

# Διορθώσεις

…

# Συμπεράσματα

Ο παραπάνω κώδικας εκτελεί την ζητούμενη πράξη αποθηκεύοντας στην πρώτη θέση μνήμης που δηλώνεται το αποτέλεσμα και σε επόμενες διαδοχικές τις τιμές των a και b.