ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Ενσωματωμένα Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No 2

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Περιεχόμενα

1.Σκοπός της άσκησης.....4
2.Παραδοτέα 4
3.Χρήση του αναπτυξιακού περιβάλλοντος ARMulator.....4
4.Χρήση των εργαλείων γραμμής εντολών του ADS 1.2.....6
5.Χρήση Memory Maps για προσομοίωση χρήσης διαφορετικών μνημών σε ενσωματωμένο σύστημα. 11

1. Σκοπός της άσκησης

Εκτός από την αλυσίδα εργαλείων GNU για τον επεξεργαστή ARM υπάρχει και ένα δημοφιλές αναπτυξιακό περιβάλλον για τον ίδιο επεξεργαστή με το όνομα **ARM Developer Suite**. Το περιβάλλον αυτό το εμπορεύεται η εταιρία ARM *(η οποία είναι ιδιοκτήτρια των επεξεργαστών ARM)*. Σε αντίθετη με τα προηγούμενα εργαλεία που ήταν δωρεάν αυτό το πρόγραμμα απαιτεί την αγορά μιας άδειας.

Σε αυτό το εργαστήριο θα γνωρίσουμε αυτό το περιβάλλον.

(A) 39 ερωτήσεις

(C) 5 ασκήσεις

2. Παραδοτέα

- **Παραδοτέο C1:** Screenshot με όνομα firstcompile.jpg
- **Παραδοτέο C2:** Screenshot με όνομα hello2_compile.jpg
- **Παραδοτέο C3:** Screenshot με όνομα debugmonth.jpg
- **Παραδοτέο C4:** Screenshot με την επιλογή Scatter Description File
- **Παραδοτέο C5:** Screenshot

3. Χρήση του αναπτυξιακού περιβάλλοντος ARMulator

- Χρήση του αναπτυξιακού περιβάλλοντος ARMulator για την συγγραφή και προσομοίωση προγραμμάτων για τον επεξεργαστή ARM.
- Δημιουργία Προγράμματος και προσομοίωση. Χρήση του Memory Window.

Δημιουργήστε έναν κατάλογο εργασίας σε μια γνωστή τοποθεσία (π.χ. \\83.212.19.250\public\234). Κάντε απεικόνιση αυτής της διαδρομής σε ένα γράμμα δίσκου *(map network drive)* ώστε να έχετε άμεση πρόσβαση στα αρχεία σας.

Βρείτε στα προγράμματα και εκτελέστε το ARM IDE. Θα μας βγει το κεντρικό παράθυρο Codewarrior για ανάπτυξη προγραμμάτων στον ARM. Αρχικά θα δημιουργήσουμε ένα καινούργιο project.

File→New→Project→ARM Executable Image *(συμπληρώστε ένα δικό σας όνομα στο Project Name)* και επιλέξτε να αποθηκευτεί στον κατάλογό σας. Πατήστε OK . Θα δημιουργηθεί ένα παράθυρο σχετικά με το Project που μόλις δημιουργήσατε. Στο παράθυρο αυτό θα δείτε ότι δεν υπάρχει ακόμη κανένα αρχείο. Θα πρέπει να προστεθούν αρχεία που θα μεταφραστούν και θα συνδεθούν για να δημιουργηθεί το project.

Για αρχή θα φτιάξουμε ένα πρόγραμμα που εμφανίζει το μήνυμα **hello world**.

Για να δημιουργήσετε το αρχείο πηγαίνετε **FILE→New→File→Text File** στο όνομα δώστε **hello.c** δημιουργήστε τον αντίστοιχο C κώδικα και αποθηκεύστε το στον κατάλογο εργασίας. Επιλέξτε **Add to Project** για να προστεθεί στα αρχεία του project. Θα πρέπει επίσης να επιλέξουμε σε ποια targets θα πρέπει να συμμετέχει αυτό το αρχείο. Μερικές φορές κάποια αρχεία είναι μόνο για δοκιμές, οπότε θα έπρεπε να είναι μόνο στο **Debug**. Μερικά αρχεία χρησιμοποιούνται και σε δοκιμαστικές εφαρμογές οπότε συμμετέχουν και στο DebugRel. Αν το αρχείο αυτό χρειάζεται και για το τελικό προϊόν θα πρέπει να συμμετέχει και στο Release *(=τελική έκδοση)*. Επειδή αυτό το αρχείο θα συμμετέχει σε όλα θα πρέπει να επιλέξετε ΟΛΑ τα κουτάκια στο Targets. Πατήστε OK

Στο παράθυρο που θα σας βγει γράψτε το πρόγραμμα που να εκτυπώνει το string `"hello world"` στη κονσόλα.

Όταν ολοκληρώσετε το πρόγραμμα, αποθηκεύστε το, κλείστε το παράθυρο και πατήστε από το **Project→Make** για να γίνει compile. Αν σας βγουν λάθη ή warnings διαβάστε προσεκτικά τι έχει πάει στραβά και διορθώστε το ώστε να μην έχει το παραμικρό πρόβλημα.

Παραδοτέο C1: Screenshot με όνομα firstcompile.jpg

Όταν το καταφέρετε θα σας βγει στο παράθυρο *Errors & Warnings* κάποιες πληροφορίες σχετικά με το πρόγραμμά σας.

Τα πιο σημαντικά στοιχεία είναι το μέγεθος της ROM και της RAM μνήμης που απαιτεί το πρόγραμμά σας για να εκτελεστεί σε έναν επεξεργαστή ARM.

3.1 Ερωτήσεις A1-A13

Σημειώστε το μέγεθος της ROM που απαιτεί το πρόγραμμά σας καθώς και σε ποια γραμμή το βρήκατε *(ετικέτα κειμένου)*:_____**(A1)**

Σημειώστε το μέγεθος της RAM *(ReadWrite/RW)* που απαιτεί το πρόγραμμά σας καθώς και την ετικέτα του κειμένου που το βρήκατε:_____**(A2)**

Πόσα bytes είναι από τον κώδικα των βιβλιοθηκών που συνδέονται στο εκτελέσιμο σας *(καθώς και την ετικέτα του κειμένου που το βρήκατε)*_____**(A3)**

Πόσα bytes είναι από τον κώδικα του εκτελέσιμου σας *(μόνο)*, καθώς και την ετικέτα του κειμένου που το βρήκατε; _____**(A4)**

Πόσα bytes είναι τα στοιχεία που χρειάζονται μόνο να αναγνωστούν *(RO=read only)* καθώς και την ετικέτα του κειμένου που το βρήκατε _____**(A5)**

Πόσα bytes στοιχείων πρέπει να αρχικοποιηθούν στο 0 *(ZI – Zero Initialized)* καθώς και την ετικέτα του κειμένου που το βρήκατε; _____**(A6)**

Προκειμένου να χρησιμοποιήσουμε την κατάσταση αποσφαλμάτωσης πόσα bytes έχουν προστεθεί ως debug κώδικας, καθώς και την ετικέτα του κειμένου που το βρήκατε;_____ **(A7)** από αυτά πόσα είναι από τις βιβλιοθήκες _____**(A8)** & πόσα από το πρόγραμμα;_____**(A9)**

Επειδή δεν έχουμε κάποια πλακέτα hardware με τον επεξεργαστή ARM σε αυτό το εργαστήριο, θα πρέπει να δοκιμάσουμε το πρόγραμμά μας χρησιμοποιώντας τον προσομοιωτή. Αν δεν έχει φορτωθεί το παράθυρο AXD Debuger, δώστε **Project→Debug** για να φορτωθεί το περιβάλλον εκτέλεσης.

Θέλουμε να ρυθμίσουμε τις παραμέτρους προσομοίωσης.

Δώστε **Options→configure Target** στο παράθυρο AXD που σας έχει δημιουργηθεί.

Επιλέξετε τον προσομοιωτή ARMUL *(Arm Emulator)* και πατήστε configure.

Οι ρυθμίσεις πρέπει να είναι για επεξεργαστή ARM7, Emulated Clock με 200.000.000 *(=200Mhz)*, αρχιτεκτονική Little Endian, No Map File, χωρίς συνεπεξεργαστή πραγματικών αριθμών *(NO FPU)* και με σελιδοποίηση μνήμης *(default pagetables)*.

Πατήστε OK, OK και θα δείτε στο παράθυρο System Output Monitor ότι οι ρυθμίσεις σας έχουν τοποθετηθεί. Αν σας εμφανιστεί η προτροπή (Reload the last image?) πατήστε Yes.

Πατήστε **File→Load Image** και επιλέξτε το αρχείο που έχετε δημιουργήσει *(αν δεν έχει γίνει reload από το προηγούμενο βήμα)*. Πατήστε Open.

Εκτελέστε το με **Execute→GO** Θα σταματήσει στο πρώτο breakpoint που έχει ρυθμιστεί αυτόματα κατά την εισαγωγή στο κώδικα. Πατήστε πάλι Execute→GO και θα σας εμφανιστεί το μήνυμα `hello world` στο παράθυρο ARM7 Console.

Πατήστε **File→Reload** για να φορτωθεί πάλι το πρόγραμμα και να αρχικοποιηθεί ο Program Counter.

Πατήστε **Execute→go** για να φτάσετε πάλι στο πρώτο breakpoint.

Ενεργοποιήστε το παράθυρο των registers από Processor **Views→Registers**

Στα αριστερά του AXD θα σας ανοίξει ένα παράθυρο με το όνομα ARM7–Registers. Τροποποιήστε τα άλλα παράθυρα ώστε να μπορείτε να βλέπετε τα περιεχόμενα του παραθύρου.

Μέσα στην ομάδα των καταχωρητών Current. Υπάρχει ο καταχωρητής Program Counter (**PC**). Που χρησιμοποιείται και τι τιμή έχει;_____**(A10)**

Επίσης ανοίξτε από το Processor Views το παράθυρο Memory για να δείτε τα περιεχόμενα της μνήμης.

Θέλετε να δείτε τα περιεχόμενα της μνήμης στη θέση που δείχνει ο PC Οπότε, δώστε στο Start Address στο παράθυρο της μνήμης την τιμή **απάντηση_A10**.

Σε αυτόν τον επεξεργαστή ARM όλες οι εντολές αποτελούνται από 4 bytes.

Σημειώστε τα 4 bytes της εντολής που βρίσκεται στη διεύθυνση του PC

_____**(A11)**

Τα 4 bytes αποτελούν τον **opcode** της πρώτης εντολής.

Επειδή η αρχιτεκτονική είναι η Little Endian σημαίνει ότι πρώτα αναγράφεται *(ή τοποθετείται στη μνήμη)* το μικρότερο ψηφίο στη συνέχεια το αμέσως μεγαλύτερο και ούτω καθεξής. Δηλαδή αν τα byte είναι AA BB CC τότε η σωστή εντολή είναι η CC BB AA. Επειδή χρησιμοποιούμε το δεκαεξαδικό σύστημα κάθε ψηφίο του δεκαεξαδικού χρειάζεται 4 bit . Για αυτό το λόγο τα ψηφία του δεκαεξαδικού τα χρησιμοποιούμε 2 μαζί κάθε φορά για να συμπληρώνεται BYTE

Σημειώστε τα 4 bytes της εντολής που βρίσκεται στο PC με την κατάλληλη σειρά *(γνωρίζοντας ότι είναι Little Endian)*_____ **(A12)**

Πατήστε μερικές φορές **Execute→Step** ή το κουμπί **F10** και παρατηρείστε πως εκτελείται κάθε εντολή ASSEMBY του επεξεργαστή σε συνδυασμό με το PC.

Ο Program counter αυξάνεται ή μειώνεται ή και τα δύο στο πρόγραμμα σας; Υπάρχει περίπτωση σε ένα πρόγραμμα ο PC άλλες φορές να αυξάνεται και άλλες φορές να μειώνεται και γιατί;

_____**(A13)**

4. Χρήση των εργαλείων γραμμής εντολών του ADS 1.2.

- Χρήση των εργαλείων γραμμής εντολών του ADS 1.2.
- armcc, armsd, armlink, armelf.
- Disassembly & Εξαγωγή μετρήσεων επιδόσεων.

Το αναπτυξιακό περιβάλλον ARM Developer Suite (ADS), εκτός από το γραφικό περιβάλλον που έχει, παρέχει και εργαλεία της γραμμής εντολών για την ανάπτυξη πλήθους εφαρμογών για τους επεξεργαστές ARM.

Ανοίξτε το τερματικό της γραμμής εντολών.

Πηγαίνετε στο φάκελο εργασίας σας.

Χρησιμοποιώντας το notepad το παρακάτω πρόγραμμα και αποθηκεύστε το στον κατάλογο εργασίας με όνομα hello2.c. ***ΠΡΟΣΟΧΗ:*** Κατά την αποθήκευση να τοποθετήσετε το όνομα του αρχείου σε " (διπλά εισαγωγικά) για να μη μπει αυτόματα η κατάληξη .txt

```
/* hello.c */
#include <stdio.h>
#include <stdlib.h>

void subroutine(const char *message)
{
printf(message);
}
```

```
int main(void)
{
const char *greeting = "Hello from subroutine\n";
printf("Hello World from main\n");
subroutine(greeting);
printf("And Goodbye from main\n\n");
return 0;
}
```

Χρησιμοποιώντας το μεταφραστή armcc από το αναπτυξιακό περιβάλλον ADS 1.2 κάντε compile το παρακάτω πρόγραμμα: **`armcc –g hello2.c`**

Επιδιορθώστε ότι λάθος εμφανιστεί και κάντε πάλι compile.

Παραδοτέο C2: Screenshot με όνομα hello2_compile.jpg

4.1 Ερωτήσεις A14-A27

Για ποιο λόγο χρησιμοποιήσαμε την παράμετρο **–g** *(θυμηθείτε τα προηγούμενα εργαστήρια)*_____**(A14)**

Δώστε την εντολή εμφάνισης των αρχείων του καταλόγου και σημειώστε το αρχείο που δημιουργήθηκε: _____**(A15)**

Μπορείτε να εκτελέσετε το πρόγραμμα αυτό χρησιμοποιώντας τον προσομοιωτή αποσφαλμάτωσης του περιβάλλοντος ARM **`armsd`**

Εκτελέστε με προσομοίωση το αρχείο που δημιουργήθηκε χρησιμοποιώντας την εντολή: **`armsd –exec απάντηση_A15`**

Όταν θα ολοκληρωθεί η εκτέλεση θα εμφανιστεί ένα μήνυμα «Program Terminated normally». Σημειώστε την τιμή του PC (program counter) σε αυτό το σημείο: _____**(A16)**

Ποια είναι η λειτουργία του PC; _____**(A17)**

Χρησιμοποιήστε την παράμετρο –c στο μεταφραστή: **`armcc –c –g hello2.c`**

Η παράμετρος **–c** πως επηρεάζει τη διαδικασία της μετάφρασης *(θυμηθείτε προηγούμενο εργαστήριο)*; _____**(A18)**

Ποιο είναι το όνομα του αρχείου που δημιουργήθηκε;
_____**(A19)**

Προκειμένου να γίνει η σύνδεση (Link) σε περίπτωση που χρησιμοποιήσατε το διακόπτη **–c** πρέπει να εκτελέσετε το συνδέτη ως εξής

`armlink απάντηση_A19 –o myexecutable.axf`

Η παράμετρος **–o** πως επηρεάζει τη διαδικασία της σύνδεσης (θυμηθείτε από προηγούμενο εργαστήριο); _____**(A20)**

Γιατί κάποιες φορές είναι απαραίτητο να κάνουμε τη μετάφραση και τη σύνδεση σε 2 βήματα *(χρησιμοποιώντας την παράμετρο* ***–c****)* και όχι σε 1;

Γιατί μερικές φορές:

(α) είναι σημαντική η σειρά σύνδεσης των αρχείων **.o** και

(β) μπορεί μερικά αρχεία **.c** να χρειάζονται διαφορετικές παραμέτρους μετάφρασης.

Ένα ακόμη εργαλείο που διαβάζει τα αρχεία που έχουμε κατασκευάσει και μας παρέχει πληροφορίες σχετικά με αυτά είναι το **`fromelf`**

Εκτελέστε: **`fromelf –text/c hello2.o`**

Αν θέλετε να αποθηκεύστε την έξοδο της παραπάνω εντολής:

```
fromelf –text/c hello2.o > hello.dec
```

Δώστε **fromelf** και σημειώστε τι κάνουν οι παράμετροι –text/c
_____**(A21)**

Ανοίξτε το ARM IDE από τα “All programs”

Δημιουργήστε ένα αρχείο με το όνομα datatype.h και αποθηκεύστε το στον κατάλογο εργασίας με τα περιεχόμενα

```
struct DateType
{
int day;
int month;
int year;
};
```

Κατεβάστε το αρχείο **month.c** και τοποθετήστε το στο φάκελο εργασίας.

Δημιουργήστε ένα project με το όνομα calendar μέσα στο φάκελο εργασίας.

Πατήστε Project→Add Files . Προσθέστε τα 2 αρχεία **datatype.h** και **month.c** .

***ΠΡΟΣΟΧΗ: Θα πρέπει τα αρχεία και το project να τα τοποθετήσετε σε φάκελο που δεν έχει κενά και κάθε τμήμα του αποτελείται από 8 χαρακτήρες, π.χ.** **`c:\temp\dasygen\`** ***

Πατήστε **Project→Make** και διορθώστε το λάθος

Το δεύτερο λάθος οφείλεται στο γεγονός ότι δε γίνεται Include το αρχείο που έχουμε κατασκευάσει **`datatype.h`**

Η εντολή **#ifdef DATETYPE** σημαίνει ότι ΑΝ οριστεί το DATETYPE τότε και μόνο τότε φόρτωσε το αρχείο.

Εμείς όμως δεν έχουνε ορίσει *(define)* πουθενά αυτό. Για να το ορίσουμε πατήστε κλικ στο παράθυρο του project.

Μετά Project→DebugRel Settings →Language Settings→Target Settings→ARM C Compiler→Preprocessor

Στο παράθυρο βλέπουμε κάποια **#Define** που έχουν οριστεί. Γράψτε και εσείς DATETYPE και πατήστε ADD. Θα δείτε ότι έχει τοποθετηθεί στη λίστα. Πατήστε OK. Κλείστε τα παράθυρα εκτός από το παράθυρο του Project. Κάντε Make!

Προκειμένου να δούμε τις εντολές assembly του αρχείου μας **month.c** πατήστε δεξί κλικ στο αρχείο αυτό και στη συνέχεια πατήστε disassembly. Σημειώστε την τρέχουσα εντολή assembly *(δηλαδή την εντολή που θα εκτελεστεί στην επόμενη βηματική εκτέλεση ή με άλλα λόγια, την εντολή που βρίσκεται στη διεύθυνση μνήμης εντολών που δείχνει ο καταχωρητής επόμενης εντολής)*, καθώς και τη διεύθυνση μνήμης που βρίσκεται:_____**(A22)**

Πατήστε **Project→Debug** προκειμένου να το φορτώσουμε στον προσομοιωτή για να κάνουμε debug.

Παραδοτέο C3: Screenshot με όνομα debugmonth.jpg

Ένα πολύ σημαντικό στοιχείο για αυτόν που αναπτύσσει προγράμματα σε ενσωματωμένα είναι να μπορεί να βρει πληροφορίες **μέτρησης της απόδοσης** του προγράμματος του. Πρέπει να μπορεί να βρίσκει αν μια τροποποίηση που θα κάνει στο πρόγραμμά του θα βελτιώσει την επίδοση ή θα την χειροτερέψει. Το μέτρο σύγκρισης που χρησιμοποιούμε είναι οι κύκλοι εκτέλεσης. Όσους περισσότεροι κύκλοι απαιτούνται για την εκτέλεση τόσο πιο αργά θα εκτελείται η εφαρμογή (ή αντιστοίχως τόσο πιο δυνατό επεξεργαστή θα απαιτεί). Πατήστε System Views→Debugger Internals και τοποθετήστε το παράθυρο που έχει δημιουργηθεί σε ένα σημείο που να μπορείτε να το δείτε.

Πατήστε **Execute→go** . Το πρόγραμμα θα σταματήσει στο πρώτο breakpoint που μπαίνει αυτόματα κατά την εισαγωγή στη συνάρτηση main. Πατήστε δεξί κλικ πάνω στο παράθυρο του κώδικα και επιλέξετε Interleave Disassembly και θα δείτε ότι κάτω από κάθε εντολή C έχουν τοποθετηθεί κάποιες γραμμές assembly. Έτσι μπορείτε να δείτε ποιες εντολές μεταφράζονται σε πολλές εντολές assembly *(και άρα απαιτούν περισσότερους κύκλους για να εκτελεστούν)*.

4.2 Ερωτήσεις A23-A27

Ποια εντολή απαιτεί τις περισσότερες εντολές assembly από το πρόγραμμά σας; _____**(A23)**

Δεδομένου ότι κάθε εντολή είναι 32 bytes πόσα bytes Κώδικα έχει μεταφραστεί η παραπάνω εντολή; _____ **(A24)**

Δώστε τις πρώτες 5 εντολές assembly της παραπάνω εντολής:

_____**(A25)**

Πατήστε Execute→Go . Δώστε την τιμή **2025 11 26** και όταν ολοκληρωθεί δείτε το παράθυρο Statistics. Πόσες εντολές εκτελέστηκαν συνολικά *(Instructions);*

_____**(A26)**

Ψάξτε στη βιβλιογραφία *(help→online books)* ή στη σελίδα της ARM ή στο google για το τι είναι οι: **Core_Cycles**, **S_Cycles**, **N_Cycles**, **I_Cycles** και **C_Cycles** και γράψτε τι τιμές έχουν.

_____**(A27)**

5. Χρήση Memory Maps για προσομοίωση χρήσης διαφορετικών μνημών σε ενσωματωμένο σύστημα.

- Χρήση Memory Maps για προσομοίωση χρήσης διαφορετικών μνημών σε ενσωματωμένο σύστημα.
- Εξαγωγή μετρήσεων επιδόσεων.

Για να μπορέσουμε να έχουμε ακριβείς μετρήσεις σχετικά με την εκτέλεση του προγράμματός μας πρέπει να μοντελοποιήσουμε τη μνήμη που θα χρησιμοποιήσουμε όσον αφορά τους χρόνους εγγραφής και ανάγνωσης. Αν δεν το κάνουμε αυτό τότε ο ARMulator θεωρεί ότι όλοι οι χρόνοι πρόσβασης στη μνήμη είναι «zero-wait state» δηλαδή μηδενικοί *(κάθε εγγραφή/ανάγνωση γίνονται στιγμιαία)*. Για να εξαγάγουμε ρεαλιστικές εκτιμήσεις επιδόσεων θα πρέπει να περιγράψουμε κατάλληλα τη μνήμη χρησιμοποιώντας αρχεία προσδιορισμού που ονομάζονται **"memory map files"**. Σε αυτά τα αρχεία ο χρήστης μπορεί να περιγράψει το μέγεθος, τα bit καναλιού δεδομένων, τον τύπο πρόσβασης, και την ταχύτητα πρόσβασης για κάθε τύπου μνήμης που μπορεί να χρησιμοποιηθεί στο σύστημα. Η πιο απλή ρύθμιση είναι να τοποθετηθεί μια μνήμη ROM *(μόνο για ανάγνωση)* για να τοποθετηθεί το πρόγραμμα και μια μνήμη RAM για τα δεδομένα.

Πρέπει να σημειωθεί ότι όλες οι μνήμες βρίσκονται σε ένα ενιαίο χώρο διευθύνσεων *(π.χ. από 0-100 είναι η πρώτη μνήμη από 101-200 είναι η δεύτερη κτλ)*.

Η μορφή του αρχείου mapfile *(που συνήθως έχει όνομα **memory.map**)* είναι γραμμές *(κάθε γραμμή για κάθε ξεχωριστή μνήμη)* με:

αρχή μέγεθος όνομα πλάτος πρόσβαση χρόνοι-ανάγνωσης χρόνοι-εγγραφής

όπου

- ✓ **αρχή** η πρώτη διεύθυνση της μνήμης π.χ. 8000
- ✓ **μέγεθος** είναι το μέγεθος της μνήμης σε δεκαεξαδικό σύστημα π.χ.4000 *(=16384 στο δεκαδικό ή 16KB)*
- ✓ **όνομα** είναι μια απλή λέξη ορισμένη από το χρήστη που ονοματίζει τη μνήμη π.χ. SDRAM
- ✓ **πλάτος** είναι το πλάτος του διαύλου σε bytes δεδομένων της μνήμης π.χ.4 για δίαυλο 32-bit (4*8bit)

- ✓ **πρόσβαση** είναι ένας χαρακτήρας από τους **r** *(read only)* , **w** *(write only)*, **rw** *(read write),* **–** *(no access)*
- ✓ **χρόνοι-ανάγνωσης** είναι οι μη-διαδοχικοί και διαδοχικοί χρόνοι ανάγνωσης σε ns π.χ.135/85
- ✓ **χρόνοι εγγραφής** είναι οι μη-διαδοχικοί και διαδοχικοί χρόνοι εγγραφής σε ns π.χ. 135/85

Ειδικά για τους χρόνους ανάγνωσης και εγγραφής θα πρέπει να λάβετε υπόψη τους χρόνους αποκωδικοποίησης *(decoding)* και μετάδοσης *(propagation delay)*.

Παράδειγμα:

```
0       80000000       RAM       4       rw       135/85       135/85
```

(μια συνεχόμενη μνήμη 2GB RAM)

```
00000000      8000    SRAM   4 rw    1/1      1/1
00008000      8000    ROM    2 r    100/100  100/100
00010000      8000    DRAM   2 rw   150/100  150/100
7FFF8000      8000    Stack  2 rw   150/100  150/100
```

Μια πολύ γρήγορη μνήμη SRAM 16KB 32bit (=4*8) με 1 ns πρόσβαση (πιθανότατα εντός ολοκληρωμένου κυκλώματος), μια ROM 16KB και 16bit (=2*8), μια DRAM 16KB και μια μνήμη stack 16KB.

Τις περισσότερες φορές προσθέτουμε και μια «ψεύτικη» γραμμή **στο τέλος του αρχείου mapfile**, της μορφής

```
00000000 80000000 Dummy 4 - 1/1 1/1
```

ώστε σε περίπτωση που η μνήμη Dummy έχει προσβάσεις τότε να μας το αναφέρει ο armulator *(γιατί αυτό σημαίνει ότι γίνονται προσβάσεις σε διευθύνσεις μνήμης μη πραγματικές).*

Αφού έχουμε καθορίσει το memory map file θα πρέπει να «πούμε» στον **compiler/linker** πώς να το χρησιμοποιήσει και ιδιαίτερα που να τοποθετήσει τι. Αυτό ονομάζεται τοποθέτηση πριν την εκτέλεση *(pre-execution placement)*.

Η τοποθέτηση πριν την εκτέλεση γίνεται χρησιμοποιώντας ένα αρχείο προσδιορισμού για τον **compiler/linker** που ονομάζεται ***linker scatter file***. Στο αρχείο αυτό ορίζεται που θα τοποθετήσει ο Linker κάθε τμήμα του προγράμματός μας. Όπως είδαμε υπάρχουν σημαντικά τμήματα του προγράμματός μας. Το **RO** τμήμα *(readonly=Κώδικας)*, το **RW** *(readwrite=μεταβλητές που δεν έχουν αρχική τιμή 0)* και το **ZI** *(zero initialized=μεταβλητές που έχουν αρχική τιμή 0)*. Για παράδειγμα έχουμε τις εξής τοποθετήσεις σε αυτές τις γραμμές κώδικα:

```
int a ; // τοποθετείται στο ZI section
char a[10]=″minas″ // τοποθετείται στο RW section
int f (int a) { return a+2; } // τοποθετείται στο RO section
```

Για να προσδιορίσουμε τις τοποθετήσεις χρησιμοποιούμε ένα αρχείο **scatter file** της παρακάτω μορφής:

Κώδικας άσκησης	Επεξήγηση/σχολιασμός κώδικα

`INIT 0x0`
`{`
`ROM 0x0 0x8000`
`{`
`* ( +RO)`
`}`
`SRAM 0x8000 0x8000`
`{`
`* (sram)`
`stdio.o (+ZI, +RW)`
`libspace.o (+ZI, +RW)`
`}`
`DRAM 0x180000`
`{`
`* (dram)`
`* (+ZI)`
`* (+RW)`
`}`
`}`
	Ξεκινάει το scatter file στη διεύθυνση 0x0
	
	Ορίζεται ένα τμήμα ROM στη διεύθυνση 0x0 μεγέθους 0x8000 (32KB).
	Εκεί τοποθετούνται όλα (*) τα +RO δεδομένα
	
	Ορίζεται ένα τμήμα SRAM στη διεύθυνση 0x8000 και είναι μεγέθους 0x8000
	Εκεί τοποθετούνται όλα (*) τα δεδομένα που έχουν χαρακτηριστεί ως τμήμα sram, ενώ τοποθετούνται επίσης και τα ZI,RW τμήματα των αρχείων των βιβλιοθηκών stdio και libspace
	
	Τέλος ορίζεται στη διεύθυνση 0x180000 ένα τμήμα DRAM στο οποίο τοποθετούνται ότι έχει χαρακτηριστεί ως τμήμα (dram) και όλα τα υπόλοιπα ZI και RW δεδομένα.
	
	ΠΡΟΣΟΧΗ:
	Το scatter file θα πρέπει να ταιριάζει στις διευθύνσεις με το memory map file που θα χρησιμοποιήσετε στην προσομοίωση.

Προκειμένου να προσδιορίσουμε σαφώς κάθε τμήμα του προγράμματός μας που αντιστοιχεί, θα πρέπει να τοποθετούμε το αντίστοιχο τμήμα του προγράμματος ανάμεσα σε:

#pragma arm section zidata="?????" και

#pragma arm section,

όπου ????? είναι η λέξη που χαρακτηρίζει τα δεδομένα *(για το παραπάνω scatter file είναι sram είτε dram)*. Επίσης αν θέλουμε να προσδιορίσουμε τα δεδομένα **rwdata** θα πρέπει να τα τοποθετήσουμε αντιστοίχως μέσα σε:

#pragma arm section rwdata="?????"

#pragma arm section

Προκειμένου να χρησιμοποιήσουμε το scatter file θα πρέπει να πάμε στο **Edit→DebugRel** *(ή Debug)* **Settings→Linker→Arm Linker→Output→Scattered Link Type→**και να επιλέξουμε το *Scatter Description File* το αρχείο που έχουμε δημιουργήσει.

Παραδοτέο C4: Screenshot με την επιλογή Scatter Description File

Προκειμένου να δούμε που τοποθετείται το κάθε στοιχείο θα πρέπει να πάμε στο **Edit→DebugRel** *(ή Debug)* Settings→Linker→Arm Linker→Output→Listings→check το *Image Maps* και *Symbols*.

Κατεβάστε το zip αρχείο **fs-labfiles.zip**. Κάντε *extract* τα περιεχόμενά του στον κατάλογο εργασίας.

Δημιουργήστε ένα καινούργιο Project με την ονομασία **FullSearch** και αποθηκεύστε το στο φάκελο εργασίας.

Προσθέστε στο Project σας τα 2 αρχεία **lib.c** και **fullsearch.c**.

Πατήστε F7 για να γίνει **compile/Link** το project σας.

Αντιγράψτε τα 2 αρχεία **akiyo0.y** και **akiyo1.y** στον κατάλογο **\DebugRel**

Πατήστε F5 για να μεταβείτε στην προσομοίωση *(armulator)*. Προβείτε στις «γνωστές» ρυθμίσεις από προηγούμενο εργαστήριο.

Πατήστε SystemViews→Debugger Internals για να ενεργοποιηθεί το παράθυρο με τις μετρήσεις. Κάντε το resize.

Πατήστε F5 *(πάει στο πρώτο breakpoint αμέσως μετά τη main)* και πάλι F5 για να εκτελεστεί το πρόγραμμά σας.

Όταν εμφανιστεί το παράθυρο Program Terminated Normally τότε δείτε το παράθυρο με τις μετρήσεις του debugger.

5.1 Ερωτήσεις A28-A39

<u>Σημειώστε:</u>

Εντολές που εκτελέστηκαν *(Instructions)*:_____**(A28)**

Συνολικοί Κύκλοι *(Total Cycles)*:_____**(A29)**

Σημειώστε ότι οι υπόλοιπες κατηγορίες κύκλων **S_** (*sequential*) **N_** (*non-sequential*) **I_(***internal*) **C_(***coprocessor*) δε μας ενδιαφέρουν αλλά μόνο το total που είναι το σύνολο αυτών των κύκλων.

Κλείστε το παράθυρο του AXD.

Στο **ARM IDE** ορίστε στο Linker να χρησιμοποιήσει το scatter file που σας έχει δοθεί.

Επίσης ρυθμίστε να εμφανίζει το listing για το που τοποθετείται το κάθε στοιχείο.

Πατήστε F7 για να γίνει compile το πρόγραμμα. Σημειώστε τη διεύθυνση του πίνακα cblock _____**(A29)**.

Σε ποιο τμήμα έχει τοποθετηθεί σύμφωνα με το scatter file _____ *(sram or dram)* **(A30)**.

Ομοίως για τον πίνακα current _____**(A31)** καθώς και το τμήμα που έχει τοποθετηθεί _____ *(sram or dram)* **(A32)**.

Ανοίξτε το **fullsearch.c** και δείτε τις εντολές **#pragma** που έχουν τοποθετηθεί και προσδιορίζουν την τοποθέτηση. Συμφωνούν οι εντολές pragma με τις διευθύνσεις στις οποίες έχουν τοποθετηθεί τα στοιχεία; Αν υπάρχει απόκλιση διορθώστε τη.

Πατήστε F5 για να μεταβείτε στην κατάσταση debug. Ορίστε τη χρήση του mapfile κατά το Configure Target.

Διαπιστώστε ότι στο System Output Monitor σας εμφανίζει πληροφορίες σχετικά με το memory map.

Πατήστε F5 (πάει στο πρώτο breakpoint αμέσως μετά τη main) και πάλι F5 για να εκτελεστεί το πρόγραμμά σας.

Σημειώστε:

Εντολές που εκτελέστηκαν (*Instructions*):_____ **(A33)**

Συνολικοί Κύκλοι *(Total Cycles)*:_____ **(A34)**

Προκειμένου να δούμε συγκεκριμένο αριθμό εγγραφών σε ένα τμήμα μνήμης πηγαίνετε στο παράθυρο Debugger Internals→Internal Variables→$memstats και ανοίξτε μια από τις υποκατηγορίες. Το γράμμα S προσδιορίζει τις διαδοχικές προσβάσεις (*sequential*) το N τις μη διαδοχικές (*non-sequential*), το ns προσδιορίζει χρόνο σε ns. Σημειώστε πόσες προσβάσεις συνολικά γίνονται στην εξωτερική μνήμη DRAM: ανάγνωσης *(reads)*:_____**(A35)** και εγγραφής _____**(A36)**, και πόσες ομοίως στη SRAM **(A37)**, καθώς και που βρήκατε αυτά τα στοιχεία.

Παραδοτέο C5: Screenshot

Ποια είναι η συνολική τιμή χρόνου προσβάσεων:

_____**(A38)**

Δημιουργήστε ένα καινούργιο **memory map** και το αντίστοιχο **linker scatter file**, ώστε να υπάρχει επιπρόσθετα μια ακόμη SRAM με διπλάσιο χρόνο πρόσβασης από ότι η προηγούμενη SRAM, διπλάσιο μέγεθος από τη προηγούμενη και στην οποία έχει τοποθετηθεί ο πίνακας previous. Επαναλάβετε τις μετρήσεις προσβάσεων σε όλα τα επίπεδα της ιεραρχίας μνήμης. Σχολιάστε τη διαφορά ταχύτητας *(συνολικό αριθμό κύκλων)*:_____**(A39)**