ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Ενσωματωμένα Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:01

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

- Cross-compile για ενσωματωμένα συστήματα. Χρήση ARM toolchain για δημιουργία εκτελέσιμων ενσωματωμένων προγραμμάτων για επεξεργαστές ARM.

(Α) 36 ερωτήσεις

(Β) 4 ασκήσεις/προγράμματα

Η διαδικασία ανάπτυξης προγράμματος για ένα ενσωματωμένο σύστημα είναι να βρούμε τα εργαλεία ανάπτυξης για αυτό το σύστημα να τα εκτελέσουμε στον υπολογιστή μας (ονομάζεται υπολογιστής HOST) και στη συνέχεια να μεταφέρουμε το πρόγραμμα στο ενσωματωμένο σύστημα. Η διαδικασία μετάφρασης και σύνδεσης στο δικό μας υπολογιστή ενός προγράμματος για μια διαφορετική αρχιτεκτονική, ονομάζεται cross-compile. Η ιδιομορφία είναι ότι το εκτελέσιμο που παράγεται δε μπορεί να εκτελεστεί στο δικό μας σύστημα, εκτός αν χρησιμοποιήσουμε κάποιον προσομοιωτή.

Σε αυτό το εργαστήριο θα χρησιμοποιήσουμε τα δωρεάν εργαλεία cross-compile GNU για την οικογένεια των ενσωματωμένων επεξεργαστών ARM.

Αν δεν έχετε ήδη συνδεθεί στο ΛΣ, συνδεθείτε στο ΛΣ σύμφωνα με τις οδηγίες που σας έχουν δοθεί.

Αν στο μηχάνημα που χρησιμοποιείται δε βρίσκεται ο κατάλογος **c:\devkitpro** τότε θα πρέπει να εγκαταστήσετε ως διαχειριστές τα αναπτυξιακά προγράμματα για τον επεξεργαστή ARM.

2. Παραδοτέα

- **Παραδοτέο C1:** Το πρόγραμμα my1st.c
- **Παραδοτέο C2:** Screenshot του compile με όνομα: exec01.jpg.
- **Παραδοτέο C3:** Το Makefile.
- **Παραδοτέο C4:** Screenshot του compile για psp με όνομα pspcompile.jpg

3. Εγκατάσταση αναπτυξιακών προγραμμάτων για τον επεξεργαστή ARM (μόνο αν δεν υπάρχουν)

Η διαδικασία αυτή θα γίνει μόνο αν δεν υπάρχουν εγκατεστημένα τα αναπτυξιακά εργαλεία για τον επεξεργαστή ARM στον υπολογιστή που εργάζεστε.

Αφού συνδεθείτε ως διαχειριστές, πηγαίνετε στη διαδρομή που είναι αποθηκευμένα τα αρχεία εγκατάστασης από την προτροπή της εκτέλεσης (συντόμευση πληκτρολογίου WinKey+R):

\\83.212.19.250

κατάλογος **lab\Χρήστες\Ενσωματωμένα Συστήματα**

και αντιγράψτε το φάκελο **Devkit-ARM.1.5.0** στην επιφάνεια εργασίας[1]

Αφού τον αντιγράψετε εκτελέστε το **"devkitProUpdater-1.5.0"**.

Μόλις ξεκινήσει το πρόγραμμα επιλέξτε την προτροπή:

"Install from downloaded files".

Επιλέξτε την πλήρη εγκατάσταση (**Full**) και μην τροποποιήσετε την τοποθεσία εγκατάστασης (**c:\devkitpro**).

Μόλις ολοκληρωθεί η εγκατάσταση μπορείτε να σβήσετε το φάκελο από την επιφάνεια εργασίας.

4. Cross Compile για ARM

Ανοίξτε ένα παράθυρο φλοιού CMD (από **Accessories >"Command Promt"**).

Σημειώστε τη διαδρομή που αναφέρεται δίπλα στην προτροπή (π.χ. c:\users\user15)

Δώστε

mkdir lab-arm

για να δημιουργήσετε ένα κατάλογο με το όνομά lab-arm και μπείτε μέσα σε αυτόν με την εντολή:

cd lab-arm

Προκειμένου να δημιουργήσουμε μια εφαρμογή την οποία θα την κάνουμε μετάφραση για ενσωματωμένο σύστημα. Εκτελέστε από τα προγράμματα του devkitpro το **Programmers Notepad**.

Στο περιβάλλον του editor κατασκευάστε ένα πρόγραμμα το οποίο θα έχει μια συνάρτηση **main()** η οποία θα κάνει κλήση σε μια δεύτερη συνάρτηση **printmsg()**. Η δεύτερη συνάρτηση θα εκτυπώνει το μήνυμα: **"Hello world!"** και θα επιστρέφει με return στη main.

Μπορείτε να χρησιμοποιήσετε το Internet για βοήθεια.

Όταν το ολοκληρώσετε αποθηκεύστε το στο φάκελο εργασίας που έχετε δημιουργήσει προηγουμένως με το όνομα my1st.c

Παραδοτέο C1: Το πρόγραμμα my1st.c

Δοκιμάστε να κάνετε compile το παραπάνω πρόγραμμα (my1st.c) με το πρόγραμμα μετάφρασης **arm-eabi-gcc**, το οποίο είναι το πρόγραμμα μετάφραση gcc ειδικά τροποποιημένο για τον επεξεργαστή ARM. Τα αρχεία που δημιουργούνται για τον ARM είναι της μορφής EABI (embedded application binary interface)[2]

1 Εναλλακτικά μπορείτε να κατεβάσετε τα αναπτυξιακά εργαλεία από εδώ.

2 Για περισσότερες πληροφορίες μπορείτε να ανατρέξετε εδώ

C:\devkitPro\devkitARM\bin\arm-eabi-gcc my1st.c -o my1st

Δώστε **dir** και δείτε τι αρχεία έχουν δημιουργηθεί. **Σημείωση**, το πρόγραμμα my1st που έχει δημιουργηθεί είναι πρόγραμμα για επεξεργαστές ARM, οπότε δε μπορεί να εκτελεστεί στο περιβάλλον που είστε.

Μπορείτε να το προσομοιώσετε όμως με τον αποσφαλματωτή GNU Debugger (gdb).

Δώστε:

C:\devkitPro\devkitARM\bin\arm-eabi-gdb my1st

Για φορτώσετε στο debugger το εκτελέσιμο πρόγραμμα.

Στην προτροπή (gdb) Δώστε **`help target`** και βρείτε με ποιόν τρόπο θα το εκτελέσετε. Επειδή δεν έχουμε κάποιον ενσωματωμένο επεξεργαστή, θα χρησιμοποιήσουμε έναν προσομοιωτή. Δώστε την αντίστοιχη εντολή για προσομοίωση στο (gdb).

Εντολή που δώσατε _____**(A1)**

Το επόμενο βήμα είναι να φορτώσετε το πρόγραμμα στη μνήμη ή να το μεταφέρετε στην αρχιτεκτονική στόχος. Στη δικιά μας περίπτωση η αρχιτεκτονική στόχος είναι η προσομοίωση. Η εντολή που πρέπει να δώσετε είναι η **`load`**

Καθώς γίνεται η **`load`** θα δείτε ότι φορτώνονται τα διάφορα τμήματα του κώδικα στη μνήμη. Όταν ολοκληρωθεί θα δείτε «Start address…».

Δώστε **`run`** και δείτε το αποτέλεσμα στην οθόνη ύστερα από αρκετά λεπτά, λόγω της προσομοίωσης. Μπορείτε να ανοίξετε ένα δεύτερο παράθυρο για να μην περιμένετε την ολοκλήρωση της εκτέλεσης. Πηγαίνετε στο νέο παράθυρο στην ίδια διαδρομή.

Προκειμένου να μπορέσετε να κάνετε καλύτερη αποσφαλμάτωση θα πρέπει στο compiler να δώσετε την παράμετρο **`-g`** . Αν θέλετε να εγκαταλείψετε το πρόγραμμα gdb θα δώσετε την εντολή quit.

Η μετάφραση με υποστήριξη αποσφαλμάτωσης γίνεται με την εντολή:

```
C:\devkitPro\devkitARM\bin\arm-eabi-gcc -g my1st.c -o my1stG
```

και επαναλάβετε το gdb μέχρι και την εντολή **load**

Δώστε:

```
break main
run
```

και στη συνέχεια πατήστε:

```
step
next
step
```

για να δείτε την εκτέλεση.

Στο arm-eabi-gcc μπορείτε να επιλέξετε και έναν οποιοδήποτε επεξεργαστή της οικογένειας ARM, προκειμένου να δημιουργηθεί κώδικας εξειδικευμένος για τη συγκεκριμένη αρχιτεκτονική.

Προκειμένου να δείτε πως λειτουργεί αυτό δώστε:

```
C:\devkitPro\devkitARM\bin\arm-eabi-gcc --target-help
```

Δοκιμάστε και κάντε compile το πρόγραμμα για τον επεξεργαστή ARM Cortex-A8.

Ποια εντολή έχετε δώσει; _____**(A2)**

Οι περισσότεροι επεξεργαστές ARM υποστηρίζουν 2 σετ εντολών. Το κανονικό (προεπιλογή) και το Thumb.

Ψάξτε στο Internet για το σετ εντολών Arm Thumb και γράψτε ένα πλεονέκτημα και ένα μειονέκτημα: _____**(A3)**

Δώστε την εντολή που θα μεταφράσει το πρόγραμμά σας για ένα επεξεργαστή ARM 7 με σετ εντολών Thumb._____**(A4)**

5. Προχωρημένα θέματα σε cross-compile με τα εργαλεία arm-gnu.

- **Δομή ενός προγράμματος ενσωματωμένου.**
- **Χρήση Makefile και arm-gnu.**
- **Memory Maps.**
- **Χρήση του objdump για εξαγωγή πληροφοριών από το εκτελέσιμο αρχείο.**

Κατασκευάστε το παρακάτω πρόγραμμα μέσα στον κατάλογο εργασίας και ονομάστε το **main.c** . Εξετάστε τη λειτουργία του.

```
int main (void)
{
while (1)
{
function();
}
return 0; /*Will never reach it */
}
```

Σε ένα τυπικό ενσωματωμένο σύστημα υπάρχει ένα `main()` το οποίο εκτελεί διαρκώς μια ή περισσότερες συναρτήσεις χωρίς ποτέ να τερματίζει. Για αυτό το λόγο χρησιμοποιείται ένας αενάως βρόχος (while(1)) χωρίς ποτέ να τερματίζεται η `main()`. Όπως είναι ευκολονόητο, αν η `main()` τερματιστεί, τότε δε θα υπήρχε τίποτα περαιτέρω να εκτελέσει ο επεξεργαστής.

Κατασκευάστε το παρακάτω πρόγραμμα και ονομάστε το **function.c**. Εξετάστε τη λειτουργία του.

```
int function()
{
input();
processA();
output();
return 1;
}
```

Σε ένα τυπικό ενσωματωμένο σύστημα επαναλαμβάνεται συνεχώς η είσοδος στοιχείων προς επεξεργασία (π.χ. από αισθητήρια-sensors), η επεξεργασία των στοιχείων και η έξοδος σε κάποιο περιφερειακό.

Κατασκευάστε το παρακάτω πρόγραμμα και ονομάστε το **library.c**. Σε ένα πραγματικό σύστημα οι παρακάτω συναρτήσεις που τώρα έχουν μόνο ένα printf θα είχαν κώδικα C.

```
#include <stdio.h>
void input() { printf("Input…\n");}
void processA() { printf("Processing A…\n"); }
void processB() { printf("Processing B…\n"); }
void output() { printf("Output…\n"); }
```

Προσοχή: Αν κάνετε copy paste τότε ενδέχεται κάποιοι χαρακτήρες να μη μεταφερθούν σωστά

Κάντε compile με την εντολή για να δείτε ότι μπορεί να δημιουργηθεί ένα εκτελέσιμο πρόγραμμα.

Παραδοτέο C2: Screenshot του compile με όνομα: exec01.jpg

Δώστε την εντολή για να γίνει compile το συγκεκριμένο project που αποτελείται από τα παραπάνω εκτελέσιμα με την παράμετρο προσθήκης κώδικα αποσφαλμάτωσης. Το όνομα του αρχείου που θα δημιουργηθεί να είναι το **exec01**:

_____**(A5)**

Αν ολοκληρωθεί με επιτυχία η προηγούμενη εντολή πηγαίνετε παρακάτω, διαφορετικά διορθώστε το κάθε σφάλμα.

6. Δημιουργία Project με χρήση Makefiles

Αν έχουμε ένα αρχείο C και θέλουμε να το μεταφράσουμε σε εκτελέσιμο, μπορούμε πολύ απλά να δώσουμε μια εντολή gcc και να το κάνουμε. Στον προγραμματισμό ενσωματωμένων συστημάτων όμως δεν υπάρχει μόνο ένα αρχείο source file. Σε ένα τυπικό Project θα συναντήσετε εκατοντάδες αρχεία. Το να κάθεται κάποιος προγραμματιστής και να εκτελεί μια-μια τις εντολές μετάφρασης και σύνδεσης ασφαλώς είναι χάσιμο χρόνου.

Για να βελτιστοποιηθεί αυτή η διαδικασία, ανακαλύφθηκε το 'Makefile'. Το Makefile είναι μια ανακάλυψη χρήσιμη σε όλους τους προγραμματιστές ή διαχειριστές συστημάτων. Αρκεί ο προγραμματιστής να κατασκευάσει ένα αρχείο με το όνομα 'Makefile', και να τοποθετήσει όλες τις συνδέσεις των αρχείων και το πώς πρέπει να γίνουν compile, κάτι που είναι μερικές φορές χρονοβόρο. Δύο είναι τα πλεονεκτήματα: Η χρήση του Makefile μας επιτρέπει να εξηγήσουμε στον υπολογιστή τις συνδέσεις μεταξύ των αρχείων. Για παράδειγμα, να του πούμε ότι πρέπει να γίνει πρώτα compile το αρχείο xxx1.c μετά το αρχείο xxx2.c και ούτω καθεξής. Επίσης σε περίπτωση που κάνουμε αλλαγή στο αρχείο xxx10.c δε θα χρειαστεί να γίνει πάλι compile τα προηγούμενα αρχεία αφού έχουμε καθορίσει μέσα στο Makefile τη σειρά compile των αρχείων. Τέλος, για να γίνει compile ένα project με εκατοντάδες αρχεία (ή ακόμη και με ένα) αρκεί να γράψουμε μια εντολή μόνο (τη make) και όχι να γράφουμε το gcc κάθε φορά.

Για περισσότερες πληροφορίες δείτε εδώ

Δημιουργήστε έναν καινούργιο κατάλογο (π.χ. maketest) και εισέλθετε:

```
mkdir maketest
cd maketest
```

Στον κατάλογο που βρισκόμαστε θα πρέπει να δημιουργήσουμε ένα αρχείο με το όνομα **Makefile** στο οποίο θα τοποθετήσουμε κάποιες εντολές.

Αρχικά κάντε download τα παρακάτω αρχεία:

bimod.c
examplegdb.c
examplegdb.h
transformMatrix.c
Τα αρχεία βρίσκονται στο eclass.

Δοκιμάστε να κάνετε compile τα αρχεία. Δώστε:

```
C:\devkitPro\devkitARM\arm-eabi\bin\gcc examplegdb.c
```

Έγινε με επιτυχία; _____**(A6)**

Ομοίως για το αρχείο bimod.c

Έγινε με επιτυχία; _____**(Α7)**

Κάντε ταυτόχρονα compile τα bimod.c transformMatrix.c examplegdb.h examplegdb.c

Στο αρχείο Makefile τοποθετούμε μια ετικέτα για compile ενός ή περισσοτέρων αρχείων ακολουθούμενη από τις εξαρτήσεις. Χρησιμοποιώντας το Pico και διαβάζοντας τις σημειώσεις που ακολουθούν κατασκευάστε ένα αρχείο με το όνομα Makefile με τις παρακάτω γραμμές.

Τροποποιήστε το gcc με το compiler που έχετε εγκαταστήσει και θέλετε να το χρησιμοποιήσετε!

```
#Linking source files
all: bimod.o transformMatrix.o examplegdb.o
gcc examplegdb.o transformMatrix.o bimod.o -o examplegdb
# Compiling source files
bimod.o: bimod.c
gcc -c bimod.c

transformMatrix.o: transformMatrix.c
gcc -c transformMatrix.c
examplegdb.o: examplegdb.c examplegdb.h
gcc -c examplegdb.c
```

Οι γραμμές που αρχίζουν από το **#** είναι σχόλια. Όταν υπάρχει μια λέξη και μετά ακολουθεί το **:** τότε αυτή η λέξη είναι ετικέτα. Η κάθε ετικέτα ακολουθείται από 0 ή περισσότερα ονόματα αρχείων από τα οποία εξαρτάται. Για παράδειγμα το bimod.o εξαρτάται από το bimod.c. Το bimod.o θα γίνει compile ΑΝ και ΜΟΝΟ ΑΝ τροποποιηθεί το αρχείο bimod.c . Κάτω από κάθε ετικέτα έχουμε πατήσει 2 φορές το πλήκτρο TAB (ΟΧΙ το spacebar). Η παράμετρος –c στο gcc σημαίνει ότι να γίνει compile το αρχείο αλλά όχι Link, δηλαδή να δημιουργηθεί μόνο το Object file με κατάληξη .o

Επιστρέψτε στη γραμμή εντολών και δώστε **make all**. Αυτή η εντολή έχει ως συνέπεια να εκτελεστεί η ετικέτα all: και να δει τι προγράμματα χρειάζονται για να γίνει compile το πρόγραμμα. Αν πάνε όλα καλά ΔΕ θα δείτε μηνύματα σφάλματος και θα δημιουργηθεί το αρχείο.

ΠΡΟΣΟΧΗ 1: Σε περίπτωση που κατά την εκτέλεση του make all, σας εμφανιστεί το μήνυμα λάθους **"make: *** No rule to make target 'all'. Stop"**. τότε αυτό σημαίνει, είτε ότι δεν υπάρχει αρχείο Makefile στο τρέχων κατάλογο (δώστε dir να επιβεβαιώσετε ότι υπάρχει αυτό το αρχείο), είτε ότι το Makefile δεν έχει μια καταχώρηση all: η οποία ξεκινάει από την αρχή της γραμμής (ανοίξτε το Makefile με τον επεξεργαστή της προτίμησή σας).

ΠΡΟΣΟΧΗ 2: Σε περίπτωση που κατά την εκτέλεση του make all, σας εμφανιστεί το μήνυμα **λάθους "make: cc: Command not found"**, τότε αυτό σημαίνει ότι δε βρίσκεται στη διαδρομή αναζήτησης (path) ο compiler. Επιβεβαιώστε ότι (α) χρησιμοποιείτε το compiler που βρίσκεται στο σύστημά σας (μπορεί να είναι gcc ή κάτι άλλο) και (β) μπορείτε να εκτελέσετε το compiler στον κατάλογο που βρίσκεστε (προσπαθήστε να εκτελέσετε το compiler στον τρέχων κατάλογο). Τροποποιήστε τη διαδρομή αναζήτησης και θα λυθεί το πρόβλημα.

Εκτός από εντολές για compile το Makefile μπορεί να εκτελέσει και άλλες εντολές του Linux. Για παράδειγμα προσθέστε τις παρακάτω γραμμές στο Makefile και δώστε **make clean** ώστε να εκτελεστούν αυτές μόνο:

```
clean:
 rm examplegdb.o
 rm transformMatrix.o
 rm bimod.o
 rm examplegdb
```

Τι κάνουν αυτές οι εντολές; _____**(A8)**

Τέλος, μπορούμε να τοποθετούμε και παραμέτρους στο Makefile προκειμένου να απλοποιήσουμε κάποια στοιχεία που επαναλαμβάνονται αρκετά συχνά. Για παράδειγμα μπορούμε να τοποθετήσουμε ως πρώτη γραμμή την εντολή η οποία δημιουργεί μια μεταβλητή με το όνομα CC και τις δίνει την τιμή gcc

CC=gcc

και να αντικαταστήσουμε στο Makefile την εντολή εκτέλεσης του compiler με την αναφορά **$(CC)**

Για παράδειγμα αντί για **gcc -c bimod.c** θα εμφανιστεί **$(CC) -c bimod.c**

Επιβεβαιώστε με το να δώσετε **make all**. Πρέπει να μην υπάρχει κανένα πρόβλημα.

7. Makefile

Δημιουργήστε το κατάλληλο Makefile μέσα στον κατάλογο εργασίας για το παραπάνω project. Χρησιμοποιήστε τις μεταβλητές

```
CC=arm-eabi-gcc
CFLAGS=-g -Wl,-Map=memory.map
```

Στο Makefile τοποθετείστε και συμπληρώστε σωστά τα παρακάτω labels

```
all:
clean:
function.o:
library.o:
main.o:
```

Παραδοτέο C3: Το Makefile

Με το παραπάνω CFLAGS ενεργοποιούμε την παραγωγή εκτελέσιμου για αποσφαλμάτωση και επίσης κατά τη διαδικασία της σύνδεσης να δημιουργηθεί ένα αρχείο με το όνομα memory.map, το οποίο περιέχει την χαρτογράφηση της μνήμης (που τοποθετείται η κάθε συνάρτηση). Το αρχείο αυτό θα δημιουργείται αυτόματα κάθε φορά.

Όταν το ολοκληρώσετε πηγαίνετε στον κατάλογο εργασίας και δώστε make για να γίνει compile αν παρουσιαστεί κάποιο πρόβλημα ξανα-κάντε edit το Makefile.

Βρείτε από την εντολή **arm-eabi-gcc --help** τι σημαίνει η παράμετρος -Wl :

_____**(A9)**

Ο linker που χρησιμοποιούμε είναι ο **`arm-eabi-ld`**.

Βρείτε από την εντολή **`arm-eabi-ld --help`** τι σημαίνει η παράμετρος –Map :

_____**(A10)**

Αφού ολοκληρωθεί το compile (χωρίς κανένα warning), θα δημιουργηθεί ένα αρχείο **memory.map**

Ανοίξτε το αρχείο Memory.map με το programmers notepad ή με όποιον άλλο editor θέλετε.

Χρησιμοποιώντας το Search βρείτε σε ποια διεύθυνση έχει τοποθετηθεί η συνάρτηση processA
_____**(A11)**

Επίσης βρείτε σε πιο τμήμα του εκτελέσιμου αρχείου βρίσκεται αυτή η συνάρτηση
_____**(A12)**

Βρείτε το τμήμα .init σε ποια διεύθυνση αρχίζει.

_____**(A13)**

Το τμήμα .init περιέχει κώδικα που θα εκτελεστεί πριν αρχίσει η εκτέλεση του προγράμματος.

Σημειώστε τα ονόματα των **.o** αρχείων που έχουν συνδεθεί στο τμήμα .init :
_____**(A14)**

Προκειμένου να αρχίσει να λειτουργεί ο επεξεργαστής προτού εκτελέσει το πρόγραμμά μας πρέπει να εκτελέσει ρουτίνες αρχικοποίησης που βρίσκονται σε κάποια αρχεία .o . Αυτά τα αρχεία χρησιμοποιούνται σε κάθε .init τμήμα κάθε προγράμματος που αναπτύσσουμε. Τοποθετούνται αυτόματα από το συνδέτη (linker). Ο συνδέτης GCC τοποθετεί στο τμήμα .init 2 συγκεκριμένα object files. Το ένα στην αρχή του .init που είναι ο “πρόλογος (prologue)” του τμήματος και το άλλο στο τέλος του τμήματος που είναι ο “επίλογος (epilogue)”.

Ο σωρός (.stack) σε τι διεύθυνση έχει τοποθετηθεί:

_____**(A15)**

Ένα βοηθητικό πρόγραμμα από τη σειρά προγραμμάτων για τον επεξεργαστή ARM είναι το:

`arm-eabi-objdump`

Δώστε στο command prompts το **`arm-elf-objdump`** και βρείτε τις παραμέτρους για Εμφάνιση των εντολών assembly κάθε τμήματος (Display Assembler Contents of all sections)

_____**(A16)**

Εμφάνιση της ετικέτας του εκτελέσιμου αρχείου. Η ετικέτα είναι μια σειρά από χαρακτηριστικά για το εκτελέσιμο αρχείο ELF (Display overall file header):

_____**(A17)**

Εμφάνιση πηγαίου κώδικα και κώδικα Assembly._____**(A18)**

Δώστε:

`arm-eabi-objdump απάντηση_A16 όνομα_εκτελέσιμου_αρχείου | more`

και σημειώστε την πρώτη εντολή assembly της συνάρτησης main

_____**(A19)**

Σε ποια διεύθυνση βρίσκεται αυτή (αριστερή στήλη);

_____**(A20)**

Πόσο μέγεθος έχει η πρώτη εντολή σε bytes *(δείτε τη διεύθυνση της επόμενης γραμμής και αφαιρέστε)*

_____**(A21)**

Δώστε:

```
arm-eabi-objdump απάντηση_A17 όνομα_εκτελέσιμου_αρχείου | more
```

και σημειώστε τη διεύθυνση που ξεκινάει το πρόγραμμα:

_____**(A22)**

και για ποια αρχιτεκτονική είναι:

_____**(A23)**

Δώστε

```
arm-eabi-objdump   απάντηση_A18 όνομα_εκτελέσιμου_αρχείου | more
```

και σημειώστε πόσες γραμμές assembly έχει η συνάρτηση input()

_____**(A24)**

η εντολή while(1) σε ποιες γραμμές assembly μεταφράζεται:

_____**(A25)**

8. Προχωρημένα θέματα στη δημιουργία Project ενσωματωμένων συστημάτων.

- **Κατασκευή και χρήση διαφορετικών Makefile για ποικίλες αρχιτεκτονικές.**
- **Χρήση προγραμμάτων ανάπτυξης GNU για την Ενσωματωμένη Αρχιτεκτονική Intel-psp.**
- **Χρήση του insight-debugger.**

Επιβεβαιώστε την καλή λειτουργία του Makefile που έχετε δημιουργήσει προηγουμένως σας δίνοντας στη γραμμή εντολών:

```
cd <κατάλογος εργασιας>
make clean
make all
```

{ ΑΝ ΥΠΑΡΧΕΙ ΠΡΟΒΛΗΜΑ ΜΗ ΣΥΝΕΧΙΣΕΤΕ ΠΑΡΑΚΑΤΩ }

Μερικές φορές θέλουμε τη δυνατότητα να κάνουμε compile τον κώδικά μας για ποικίλους ενσωματωμένους επεξεργαστές. Προκειμένου να το επιτύχουμε αυτό θα πρέπει να κατασκευάσουμε για **ΚΑΘΕ** επεξεργαστή ένα MAKEFILE.

Μια καλή πρακτική είναι να τοποθετούμε μια κατάληξη στο Makefile που υποδηλώνει την αρχιτεκτονική.

Μετονομάστε το **Makefile** σε **Makefile.arm**, επειδή το παραπάνω Makefile προορίζεται για αρχιτεκτονική ARM.

Αντιγράψτε το Makefile.arm σε Makefile.psp επειδή πρόκειται να δημιουργήσουμε ένα Makefile για την παραγωγή αρχείων για μια άλλη ενσωματωμένη αρχιτεκτονική. Μπορούμε για παράδειγμα να επιλέξουμε την Intel psp®.

Αφού μεταβείτε στον κατάλογο εργασίας, δώστε στη γραμμή εντολών

```
dir
```

Θα πρέπει να διακρίνετε ότι υπάρχουν τα αρχεία Makefile.arm και Makefile.psp

Τροποποιήστε το Makefile.arm ώστε να δημιουργεί εκτελέσιμο αρχείο με το όνομα executable-arm.

Τροποποιήστε το Makefile.psp ώστε να δημιουργεί εκτελέσιμο αρχείο με το όνομα executable-psp για την αρχιτεκτονική playstation

Σε ποιο σημείο τροποποιήσατε τα Makefile;_____**(A26)**

Τροποποιήστε το Makefile.psp ώστε να χρησιμοποιεί το μεταφραστή **CC=psp-gcc**

Σε αυτό το σημείο έχετε κατασκευάσει 2 αρχεία Makefile ένα για την αρχιτεκτονική psp και ένα για την αρχιτεκτονική ARM.

Προκειμένου να τα χρησιμοποιήσουμε θα πρέπει να χρησιμοποιήσουμε κατάλληλες παραμέτρους στο make.

Δώστε **`make --help`**

Από τη λίστα με τις διαθέσιμες παραμέτρους στο make βρείτε αυτήν που του λέει να διαβάσει το αρχείο που ακολουθεί την παράμετρο (Read File as a Makefile). Ποια είναι αυτή η παράμετρος;

_____**(A27)**

Δώστε την εντολή make για να κάνει compile για τον επεξεργαστή ARM.

Ποια εντολή δώσατε;

_____**(A28)**

Δώστε την εντολή make για να κάνει clean για τον επεξεργαστή ARM.

Ποια εντολή δώσατε;

_____**(A29)**

Δώστε την εντολή make για να κάνει compile για τον επεξεργαστή psp.

***Σε αυτό το σημείο **αν εμφανιστούν προβλήματα** κατά το compile *(όπως ότι δεν υπάρχει η συνάρτηση puts())*, επιδιορθώστε τα με το να συμπεριλάβετε στην εντολή compile τις βιβλιοθήκες του PSP που έχουν αυτή τη συνάρτηση. Για να συμπεριληφθούν οι βιβλιοθήκες χρησιμοποιήστε την παράμετρο -l στο gcc, για παράδειγμα **`-lpspkernel`** για να χρησιμοποιηθεί η βιβλιοθήκη pspkernel. Επίσης, μπορεί να χρειαστεί να δώσετε τη διαδρομή που βρίσκονται οι βιβλιοθήκες με την παράμετρο -L, π.χ. **`-LC:\devkitPro\devkitPSP\lib`** . Δεν υπάρχει όριο στο πόσες φορές μπορείτε να χρησιμοποιήσετε τις παραμέτρους -l και -L στην ίδια εντολή compile. ***

TIP για compile:

Αρχειο main.c (PSP) πριν από την while χρειάζονται τα

```
pspDebugScreenInit();
SetupCallbacks();
```

και μετά εφόσον δεν ήταν endless loop

```
sceKernelSleepThread();
```

με τις αντίστοιχες δηλώσεις στο αρχείο **library.c**

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>

#define printf pspDebugScreenPrintf
// Exit callback
int ExitCallback(int Arg1, int Arg2, void *Common)
{ sceKernelExitGame();
  return 0;}

// Callback thread
int CallbackThread(SceSize Args, void *Argp)
{  int CallbackId;
CallbackId=sceKernelCreateCallback("Exit Callback",ExitCallback,NULL);
  sceKernelRegisterExitCallback(CallbackId);
  sceKernelSleepThreadCB();
  return 0;}
 // Sets up the callback thread and returns its thread id
int SetupCallbacks(void)
{ int ThreadId = 0;
ThreadId=sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
 if (ThreadId >= 0)
 { sceKernelStartThread(ThreadId, 0, 0); }
  return ThreadId;}
#endif
```

Παραδοτέο C4: Screenshot του compile για psp με όνομα pspcompile.jpg

Ποια εντολή δώσατε; _____**(A30)**

Δώστε την εντολή make για να κάνει clean για τον επεξεργαστή psp.

Ποια εντολή δώσατε; _____**(A31)**

Βρείτε τη διεύθυνση που έχει τοποθετηθεί η συνάρτηση process για την αρχιτεκτονική ARM, όπως το έχετε κάνει σε προηγούμενο εργαστήριο.

_____**(A32)**

Τι μέγεθος έχει το εκτελέσιμο αρχείο για την αρχιτεκτονική ARM;

_____**(A33)**

Κάντε clean για την αρχιτεκτονική ARM **(απάντηση_c4)**

Βρείτε τη διεύθυνση που έχει τοποθετηθεί η συνάρτηση process για την αρχιτεκτονική psp, όπως το έχετε κάνει σε προηγούμενο εργαστήριο.

_____**(A34)**

Τι μέγεθος έχει το εκτελέσιμο αρχείο για την αρχιτεκτονική psp;

_____**(A35)**

Χρησιμοποιήστε τον αποσφαλματωτή **psp-gdb** σύμφωνα με προηγούμενο εργαστήριο για το εκτελέσιμο αρχείο **executable-psp**

Δώστε την εντολή να χρησιμοποιηθεί ο προσομοιωτής.

Δώστε την εντολή να φορτωθεί το πρόγραμμα στον προσομοιωτή.

Πατήστε **run** για να δείτε ότι εκτελείτε.

Πατήστε **CTRL+C** για να διακόψετε την εκτέλεση.

Δώστε **quit** για να επιστρέψετε στη γραμμή εντολών.

Ο αποσφαλματωτής μπορεί να λειτουργήσει και σε κατάσταση γραφικών. Για να χρησιμοποιήσετε τη γραφική μορφή πηγαίνετε στη διαδρομή **C:\devkitPro\insight\bin** και εκτελέστε το

psp-insight executable-psp

Ρυθμίστε τη χρησιμοποίηση του προσομοιωτή από το FILE Target Settings

Προκειμένου να θέσετε ένα breakpoint *(παύση εκτέλεσης)* σε μια συνάρτηση, επιλέξετε από τη λίστα των αρχείων που αρχικά λέει main.c το αρχείο library.c.