ΟΜΑΔΑ 45

ΚΑΛΙΩΡΑΚΗΣ ΓΕΩΡΓΙΟΣ ΑΜ:57069

ΜΕΡΤΖΑΝΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ ΑΜ:57131

5ο Έτος

2η Εργασία στον Σχεδιασμό Ενσωματωμένων Συστημάτων

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Η εργασία που ακολουθεί έχει ως στόχο την παρουσίαση και την ανάλυση διαφορετικών τεχνικών ιεραρχίας μνήμης αλλά και την επιλογή – πρόταση μια εξ'αυτών. Βασιζόμενοι στο γεγονός ότι επιδιώκουμε την συνεχή βελτιστοποίηση ως προς την αποδοτικότερη λειτουργία του επεξεργαστή και του υπολογιστικό συστήματος ευρύτερα, επιλέξαμε την υλοποίηση των τεχνικών ιεραρχίας μνήμης στον κώδικα της 1ης εργασίας με το βέλτιστο αποτέλεσμα(βασιζόμενο στους κύκλους του επεξεργαστή), δηλαδή στο UNROLL.c. Στην προκειμένη περίπτωση, γνωρίζοντας ότι ο επεξεργαστής επεξεργάζεται τα δεδομένα με πολύ μεγαλύτερη ταχύτητα από την οποία η μνήμη μπορεί να τον τροφοδοτεί με δεδομένα, αναζητήθηκαν τρόποι έτσι ώστε η μνήμη να μην αποτελεί παράγοντα επιβράδυνσης του συστήματος, αλλά και ταυτόχρονα η υλοποίηση να είναι ρεαλιστική και ως προς .. την τσέπη μας. Αυτό σημαίνει ότι θα προσπαθήσουμε να ισοσταθμίσουμε τους παράγοντες της απόδοσης και του κόστους ώστε το αποτέλεσμά μας να είναι όσο το δυνατόν πραγματικό. Η διαδικασία που περιγράφεται, θα πραγματοποιηθεί αφού εξάγουμε τα αποτελέσματα μετά τις προτεινόμενες τροποποιήσεις, μέσω του προσομοιωτή ARMulator.

Θεωρητικό υπόβαθρο

Σε έναν υπολογιστή, υπάρχουν διάφορα μέρη που μπορούμε να αποθηκεύσουμε δεδομένα. Η αποθήκευση και η επεξεργασία δεδομένων, αποτελούν έναν από τους θεμελιώδεις λόγους της ευρείας διάδοσης της χρήσης των Η/Υ. Η επιστήμη τω υπολογιστών λοιπόν, δικαίως έχει αφιερώσει χρόνο και χρήμα στην βελτιστοποίηση των μεθόδων αποθήκευσης, προσπέλασης και επεξεργασίας δεδομένων, με τρόπο τέτοιο ώστε και η επικοινωνία μεταξύ των εσωτερικών συστημάτων- συσκευών να είναι πιο αποδοτική, όσο και στην δημιουργία μεθόδων – ρουτινών για την διασφάλιση της ομαλής και ασφαλής προσπέλασης δεδομένων από τον χρήστη. Σήμερα λοιπόν συναντάμε σωρεία αποθηκευτικών συσκευών - μεθόδων στο εσωτερικό ενός υπολογιστή, τα οποία παρά τη αλληλεξαρτώμενη λειτουργία τους, επιτελούν διαφορετικούς σκοπούς. Έχουμε λοιπόν το σκληρό δίσκο, που είναι η κύρια συσκευή αποθήκευσης που χρησιμοποιείται για την αποθήκευση όλων των απαραίτητων δεδομένων όπως αρχεία OS, εφαρμογές, μουσική, βίντεο κ.λ.π. Εκτός από τον σκληρό δίσκο, έχουμε επίσης τη μνήμη RAM, τη ROM, την κρυφή μνήμη(cache), την προσωρινή μνήμη (buffer - τμήμα του σκληρού δίσκου ή της RAM) κ.α.

Σε αυτήν την εργασία, δεν μας ενδιαφέρει η μελέτη των συσκευών-μνημών μόνιμης αποθήκευσης(HDD-SSD), αλλά το πως επηρεάζεται η λειτουργία του επεξεργαστή όταν τρέχουμε τον βελτιστοποιημένο κώδικα (βελτίωσης ακμών), ανάλογα με τα επίπεδα προσωρινής μνήμης που επιλέγουμε να γίνει η καταχώρηση των δεδομένων αλλά και το πως και το γιατί επηρεάζει η κάθε ιεραρχία αυτή τη λειτουργία. Ήδη από την πρώτη εργασία, έχουμε κάνει κάποιες αναφορές στο πως επηρεάζονται οι συνολικοί κύκλοι του επεξεργαστή ανάλογα με την μέθοδο αποθήκευσης αλλά αυτή την φορά θα το μελετήσουμε - διαπιστώσουμε σε ακόμα μεγαλύτερο βάθος. Ας πάρουμε όμως τα πράγματα από την αρχή.

Η μνήμη RAM είναι το μεγαλύτερο αποθηκευτικό μέσο που υπάρχει στο hardware του υπολογιστή και χρησιμοποιείται για την αποθήκευση των προγραμμάτων και των δεδομένων που χρησιμοποιεί η CPU σε πραγματικό χρόνο. Υπάρχουν δύο βασικοί τύποι RAM : η δυναμική RAM (DRAM) και η στατική RAM (SRAM). Η DRAM είναι η πιο κοινή μορφή αλλά πρέπει να «ανανεώνεται» (refresh) χιλιάδες φορές ανά δευτερόλεπτο για την διατήρηση της πληροφορίας, ενώ η SRAM δεν χρειάζεται κάτι τέτοιο, για αυτό καθίσταται και ως ταχύτερη. Η SRAM, ως διάταξη, είναι πιο δαπανηρή στην κατασκευή της - και επομένως στην αγορά της - σε σχέση με την DRAM. Με τον όρο RAM αναφερόμαστε στην κύρια ή κεντρική μνήμη ενός υπολογιστικού συστήματος, δηλαδή τη μνήμη στην οποία αποθηκεύονται προγράμματα και δεδομένα,

προκειμένου είτε να εκτελεστούν είτε να υποστούν επεξεργασία αντίστοιχα. Τμήμα, επίσης, της κεντρικής μνήμης είναι και η μνήμη μόνο ανάγνωσης (ROM), η οποία επίσης επιτρέπει την τυχαία προσπέλαση.

Μια μικρή σύγκριση RAM-ROM

Η βασική διαφορά των δύο τύπων μνήμης είναι ότι η μεν RAM διατηρεί τα περιεχόμενά της μόνο όσο της επιτρέπει ο χρήστης ή το λογισμικό που εκτελείται και μόνο εφόσον το υπολογιστικό σύστημα τροφοδοτείται με ηλεκτρική ενέργεια. Σε αντίθετη περίπτωση, τα περιεχόμενά της είτε αντικαθίστανται από άλλα είτε χάνονται ολοσχερώς, ενώ η ROM χρησιμοποιείται για μόνιμη αποθήκευση. Επίσης η RAM επιτρέπει τόσο την ανάγνωση όσο και την εγγραφή δεδομένων σε αυτήν, ενώ η ROM μόνο την ανάγνωση. Στον τομέα της ταχύτητας, η RAM υπερνικάει με διαφορά την ROM καθώς αποτελεί μια high-speed μνήμη. Επίσης η RAM έχει άμεση διάδραση με τον επεξεργαστή, ο οποίος μπορεί να έχει πρόσβαση στα δεδομένα που είναι αποθηκευμένα σε αυτήν, σε αντίθεση με τη ROM, που για να συμβεί αυτό πρέπει πρώτα να μεσολαβήσει η μεταφορά-αντιγραφή των δεδομένων στη RAM. Τελικά, όλα τα παραπάνω, που διαφοροποιούν τη λειτουργία των δύο μνημών σε συνδυασμό με την πολύ μεγαλύτερη χωρητικότητα της RAM, την καθιστούν μια αρκετά πιο κοστοβόρα μνήμη σε σχέση με τη ROM.

Ο λόγος που η ROM αποτελεί χρήσιμο εργαλείο στα ενσωματωμένα συστήματα, είναι ότι μεγάλο μέρος του κώδικα αλλά και των δεδομένων διατηρούνται σταθερά. Στη ROM φορτώνουμε όλα τα δεδομένα που είναι απαραίτητα για τη σωστή εκκίνηση του προγράμματος.

SRAM (Static RAM)

Είναι ένα τσιπ μνήμης που είναι ταχύτερο και χρησιμοποιεί λιγότερη ενέργεια από DRAM.

Κάθε κύτταρο SRAM αποθηκεύει ένα bit χρησιμοποιώντας ένα κύκλωμα έξι τρανζίστορ και ένα μανδαλωτη (Η DRAM χρησιμοποιεί τρανζίστορ και πυκνωτές). Η SRAM είναι πτητική, αλλά αν το σύστημα είναι τροφοδοτούμενο, διατηρεί τιμές δεδομένων χωρίς να επαναφορτίζει(refresh) τα δεδομένα στα κελιά. Είναι αρκετά ευαίσθητη στον ηλεκτρικό θόρυβο, το οποίο είναι ανεπιθύμητο ηλεκτρικό σήμα που παρεμβαίνει με ένα επιθυμητό σήμα. Λόγω του ότι δεν χρειάζεται refresh η SRAM είναι σημαντικά ταχύτερη από τη DRAM και κατά συνέπεια η κατασκευή της είναι πιο περίπλοκη ανεβάζοντας το κόστος της, ενώ κανονικά λειτουργεί ως κρυφή μνήμη CPU(cache).

DRAM (Dynamic RAM)

Είναι ένα τσιπ μνήμης που μπορεί να κρατήσει περισσότερα δεδομένα από ένα τσιπ SRAM, αλλά απαιτεί περισσότερη ενέργεια.

Κάθε κύτταρο DRAM αποθηκεύει ένα bit χρησιμοποιώντας ένα και μόνο ζευγάρι πυκνωτή και τρανζίστορ. Εφόσον τα ζεύγη εξαρτημάτων μπορούν να δημιουργήσουν ένα κύτταρο και τα δισεκατομμύρια από αυτά μπορούν να χωρέσουν σε ένα μόνο τσιπ, η DRAM μπορεί να έχει πολύ υψηλές πυκνότητες. Όπως η SRAM, η DRAM είναι πτητική. Το πρόβλημα είναι πως κάθε transistor έχει ηλεκτρικές απώλειες, και οι πυκνωτές σταδιακά αποφορτίζονται. Χρειάζεται λοιπόν να γίνεται refresh στην πληροφορία για να μην χαθεί. Ουσιαστικά η μνήμη διαβάζεται και ξαναγράφεται στο ίδιο σημείο. Οι συνεχείς αναγνώσεις και εγγραφές για το refresh, επιβραδύνουν τη συνολική λειτουργία της DRAM, ενώ οι ταχύτητες της κυμαίνονται συνήθως μεταξύ 60ns και 100ns -πιο αργή από την SRAM. Είναι ευαίσθητη στον ηλεκτρικό θόρυβο.

Η σημασία της κρυφής μνήμης (cache) στη συνύπαρξη κύριας μνήμης – CPU

Η μνήμη cache αποτελεί έναν μικρό αποθηκευτικό δίαυλο μεταξύ RAM – CPU, είναι απαραίτητη σε οποιοδήποτε σύγχρονο επεξεργαστή και ο λόγος είναι η διαφορά ανάμεσα στην ταχύτητα του επεξεργαστή και της μνήμης RAM. Η κρυφή μνήμη επεξεργαστή χρησιμοποιείται από αυτόν για να επιτύχει ταχύτερη πρόσβαση στην κύρια μνήμη. Ουσιαστικά αποθηκεύει δεδομένα , τα οποία αποτελούν αντίγραφα τιμών που βρίσκονται στη RAM. Αυτό συμβαίνει για τον εξής λόγο: Η μνήμη cache(τύπου SRAM) κατά πολύ ταχύτερη από την μνήμη DRAM, περιέχει τις πληροφορίες που μόλις χρησιμοποίησε ο επεξεργαστής, καθώς και αυτές που θα χρειαστεί στο μέλλον, ώστε να έχει όσο γίνεται πιο γρήγορη πρόσβαση σε αυτές. Προφανώς η κρυφή μνήμη, λόγω της κατασκευής της αλλά και της ταχύτητας της, δεν θα μπορούσε να προσφέρει τον αποθηκευτικό χώρο (σε μέγεθος) που προσφέρει η RAM, αλλά αντ'αυτού προσφέρει μια μικρότερη χωρητικότητα δεδομένων(σε πολλαπλές βαθμίδες) λόγω κόστους. Λόγω του γεγονότος ότι ο επεξεργαστής κατά την εκτέλεση της εφαρμογής τείνει για ένα σημαντικό ποσοστό της συνολικής χρονικής διάρκειας εκτέλεσης του προγράμματος να περιορίζεται σε μικρά υποσύνολα εντολών του προγράμματος (κάτι αναμενόμενο αφού τα προγράμματα περιέχουν βρόχους επανάληψης), η κρυφή μνήμη προσφέρει άμεση πρόσβαση (πολύ γρηγορότερη από τη RAM) στα αντίγραφα των τιμών αυτών των δεδομένων που επαναχρησιμοποιούνται, βελτιώνοντας κατά πολύ την απόδοση του υπολογιστικού συστήματος καθώς γεφυρώνει επιτυχώς το χρονικό χάσμα της (πιο αργής) απόθεσης δεδομένων από τη RAM σε σχέση με την (ταχύτερη) επεξεργασία τους από τον επεξεργαστή. Το να βρει ο επεξεργαστής την πληροφορία που θέλει στην Cache ονομάζεται Cache Hit, αλλιώς είναι Cache Miss. Με τη χρήση περίπλοκων αλγορίθμων, σε συνδυασμό με τον κώδικα του προγράμματος, η επόμενη πληροφορία που θα χρειαστεί ο επεξεργαστής προβλέπεται σωστά σε ποσοστό μέχρι και 95%.

Το φαινόμενο της συνεχούς επαναχρησιμοποίησης (αντιγράφων) μεταβλητών κατά το οποίο εντολές και λέξεις της κύριας μνήμης που χρησιμοποιούνται συχνά ή χρησιμοποιήθηκαν πρόσφατα, μεταφέρονται στην κρυφή μνήμη για να γλιτώσει κύκλους στον επεξεργαστή μας, ονομάζεται αρχή της τοπικότητας.

Η σημασία της προσωρινής μνήμης (buffer) στη συνύπαρξη κύριας μνήμης – CPU

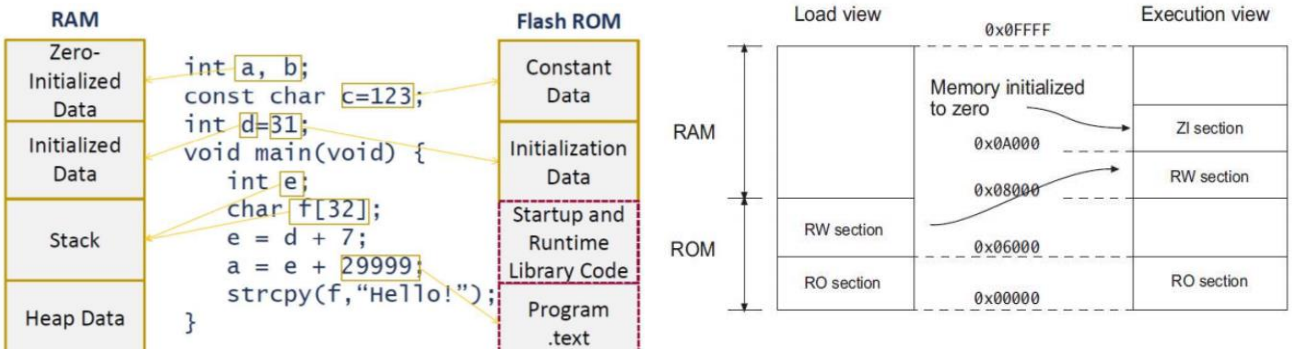
Με τον όρο προσωρινή μνήμη ή ενδιάμεση μνήμη αναφερόμαστε στη χρήση που κάνει ένα πρόγραμμα υπολογιστή (π.χ. το λειτουργικό σύστημα) μιας περιοχής της μνήμης για προσωρινή αποθήκευση δεδομένων που μετακινούνται συνεχόμενα από και προς αυτήν. Το buffer μπαίνει στο παιχνίδι σε περίπτωση που δύο στοιχεία hardware λειτουργούν σε διαφορετικές ταχύτητες (ο επεξεργαστής και οποιαδήποτε περιφερειακή συσκευή, για παράδειγμα) ή όταν δύο λογισμικά με διαφορετικές προτεραιότητες (μία διεργασία του λειτουργικού συστήματος και μία διεργασία ενός «δευτερεύοντος» προγράμματος, για παράδειγμα) πρέπει να επικοινωνήσουν το ένα με το άλλο. Σε περιπτώσεις όπως αυτές, το buffering (δηλαδή η ενέργεια της φόρτωσης δεδομένων στην εσωτερική μνήμη) επιτρέπει τη μείωση του χρόνου της εκτέλεσης της ενέργειας για να αποφευχθεί η απώλεια πολύτιμου χρόνου. Ακριβώς για το λόγο αυτό, συνήθως το buffer φιλοξενείται από τη μνήμη RAM, που εγγυάται γρήγορους χρόνους απόκρισης και ταχύτερη πρόσβαση από εκείνες ενός μαγνητικού σκληρού δίσκου.

Σε αντίθεση με την κρυφή μνήμη που έχει προκαθορισμένο το υλικό της μέρος και αποτελεί φυσικό και ανεξάρτητο τμήμα σε ένα υπολογιστικό σύστημα, το buffer ορίζεται από το λειτουργικό σύστημα ή ένα πρόγραμμα και μπορεί να αποτελεί τμήμα της RAM ή του σκληρού δίσκου. Η υλοποίηση της προσωρινής μνήμης συνήθως είναι ευθύνη του εκάστοτε σχεδιαστή λογισμικού.

Στα ενσωματωμένα συστήματα, οι προσωρινές μνήμες είναι απαραίτητες για την καταχώρηση μέρους του μεγάλου μεγέθους δεδομένων που συναντάται σε πίνακες δεδομένων απαραίτητους

για πολύπλοκες διεργασίες. Επειδή ορισμένα υπολογιστικά συστήματα προσφέρουν κρυφές μνήμες χαμηλής χωρητικότητας σε σχέση με τον όγκο των δεδομένων που πρέπει να επεξεργαστούμε (και γνωρίζοντας πως η προσωρινή αποθήκευση δεδομένων που συνεχώς επαναχρησιμοποιούνται πρέπει να γίνεται στην cache για βέλτιστη απόδοση επεξεργαστή), χρησιμοποιούμε τους buffers στους οποίους αποθηκεύουμε μέρος των δεδομένων, μεγέθους συμβατού με το μέγεθος της κρυφής μνήμης.

Περιοχές Μνήμης



Κατά την φόρτωση στη μνήμη ROM υπάρχουν τόσο τα RO όσο και τα RW Data, ενώ η RAM είναι κενή περιεχομένου. Κατά την εκτέλεση, στη ROM εξακολουθούν να βρίσκονται τα Read Only Data(Constant Data, Initialization Data) δηλαδή οι σταθερές, αλλά και οι τιμές μεταβλητών που ΔΕΝ αρχικοποιούνται στο εσωτερικό συναρτήσεων(global). Επίσης όλα τα υπόλοιπα δεδομένα που δεν μεταβάλλονται κατά την εκτέλεση του κώδικα, εξακολουθούν να είναι καταχωρημένα στη ROM, όπως οι απαραίτητες βιβλιοθήκες για την σωστή λειτουργία του κώδικα ή οι δηλώσεις σταθερών τιμών για χρήση στον κώδικα μέσω των εντολών define κτλ.

Αντίστοιχα στη RAM κατά τη διάρκεια της εκτέλεσης, φορτώνονται τα ZI Data και τα RW Data. Τα ZI Data αποτελούν τις global μεταβλητές, τα δεδομένα τα οποία είναι έξω από τη main και με το που «κατέβουν» στον μικροεπεξεργαστή θα πάρουν τιμή μηδέν. Τα RW είναι τα Δεδομένα τα οποία μπορώ να διαβάσω και να τους εκχωρήσω τιμή. Αποτελούν τις global μεταβλητές που αρχικοποιούνται από τον προγραμματιστή. Όπως φανερώνει και η πάνω αριστερά εικόνα, στη RAM αποθηκεύεται μόνο η αρχικοποιημένη μεταβλητή, ενώ η σταθερή τιμή της μένει στην ROM.Τέλος στη RAM βρίσκονται και δύο στοίβες η stack στην οποία αποθηκεύονται οι local μεταβλητές και η heap η οποία χρησιμοποιείται για τη δυναμική δεύσμευση μνήμης.

Όσον αφορά το Memory Map, χρησιμοποιήσαμε τις δοθέντες από το εργαστήριο τιμές ανάγνωσης/εγγραφής για τις ROM, DRAM, SRAM.

Ένα παράδειγμα του αρχείου memory.map

```
00000000 00008000 ROM 4 R 1/1 1/1
00008000 00008000 SRAM 4 RW 1/1 1/1
00180000 08000000 DRAM 4 RW 250/50 250/50
```

Επίσης στην αρχική υλοποίηση χρησιμοποιήσαμε την δοθέντα από το βοηθητικό υλικό δέσμευση μνήμης. Το εύρος διαύλου ορίστηκε στα 4 Bytes.

ΠΡΑΚΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

Όπως προείπαμε και στην εισαγωγή, σκοπός της εργασίας είναι η συνεχής βελτίωση της κατανομής της μνήμης προς όφελος της λειτουργίας του επεξεργαστή μας. Έτσι παραθέτουμε τα αποτελέσματα του βέλτιστου κώδικα της προηγούμενης εργασίας (UNROLL) για εξέταση και σύγκριση με τις ιεραρχίες μνήμης που θα προταθούν.

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1172         60          0    1226792       4756   Object Totals
     10196        314          0        300       4548   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11368        374          0    1227092       9304   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                11742 (  11.47kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11742 (  11.47kB)
================================================================================
```

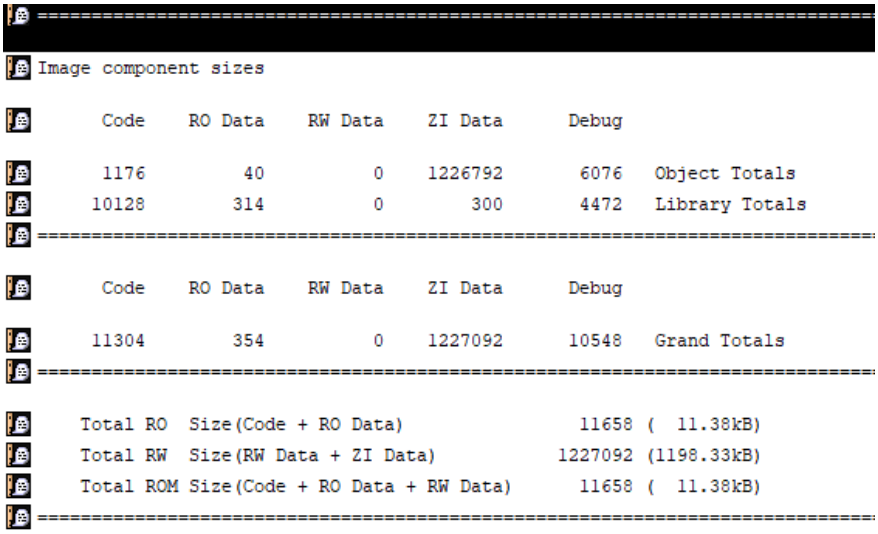
Internal Variables | Statistics

Reference_Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	9368079	17980587	8698324	7143987	2365485	0	18207796

1^{Ο} ΣΚΕΛΟΣ - ΑΛΛΑΓΕΣ ΣΤΑ ΜΕΓΕΘΗ ΤΩΝ ΜΝΗΜΩΝ ROM ΚΑΙ RAM

Αρκετά χρήσιμο θα ήταν να παρουσιάσουμε με συνεχείς δοκιμές, το κατά πόσο επηρεάζει η αλλαγή στο μέγεθος των μνημών ROM και RAM ξεκινώντας από ένα αρκετά μεγαλύτερο μέγεθος από ότι χρειαζόμαστε και για τις δύο μνήμες. Η διαδικασία που θα ακολουθηθεί σε πρώτο στάδιο περιλαμβάνει τη διαρκή μείωση του μεγέθους των μνημών για να κατανοήσουμε το πότε και με τι μέγεθος ο επεξεργαστής θα λειτουργεί πιο σωστά. Τελευταία προσέγγιση θα είναι με μέγεθος μνημών πολύ κοντά ή ακόμα και ακριβώς στα πλαίσια που χρειαζόμαστε.

Σημασία έχει σε αυτό το σημείο να παρατεθεί το αποτέλεσμα του make από το Armulator. Αξίζει να αναφερθεί πως σε όλες τις περιπτώσεις που υλοποιήσαμε μια ιεραρχία μνήμης μόνο με ROM και DRAM, για τον έλεγχο του κατά πόσο τα διαφορετικά μεγέθη μνήμης επηρεάζουν τους συνολικούς κύκλους επεξεργαστή είχαμε τα ίδια αποτελέσματα όσον αφορά τα image component sizes.



```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1176         40          0    1226792       6076   Object Totals
     10128        314          0        300       4472   Library Totals

      Code    RO Data    RW Data    ZI Data      Debug

     11304        354          0    1227092      10548   Grand Totals

    Total RO  Size(Code + RO Data)                11658 (  11.38kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11658 (  11.38kB)
```

1) ROM 0X80000 RAM 0x8000000

Memory map

memory.map

```
00000000 00080000 ROM 4 R 1/1 1/1
00080000 08000000 DRAM 4 RW 250/50 250/50
```

Scatter file

```
LOAD_ROM 0x0 0x08000000
{
EXEC_ROM 0x0 0x80000
{
*.o ( +RO )
}
DRAM 0x80000 0x8000000
{
* (ram)
* ( +ZI )
}
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007095	17448375	8603135	6782998	2289451	0	45278024	62953608	23781

2)ROM 0X20000 RAM 0x2000000

Memory map

memory.map

```
1   00000000 00020000 ROM 4 R 1/1 1/1
2   00020000 02000000 DRAM 4 RW 250/50 250/50
```

Scatter file

```
LOAD_ROM 0x0 0x02000000
{
EXEC_ROM 0x0 0x20000
{
*.o  (+RO)
}
DRAM 0x20000 0x2000000
{
* (ram)
* (+ZI, +RW)
}
}
```

Debug

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007095	17448375	8603135	6782998	2289451	0	45278024	62953608	23781

3)ROM 0X5000 RAM 0x500000

Memory map

memory.map

```
1   00000000 00005000 ROM 4 R 1/1 1/1
2   00005000 00500000 DRAM 4 RW 250/50 250/50
```

Scatter file

```
LOAD_ROM 0x0 0x500000
{
EXEC_ROM 0x0 0x5000
{
*.o  (+RO)
}
DRAM 0x5000 0x500000
{
* (ram)
* (+ZI, +RW)
}
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007095	17448375	8603135	6782998	2289451	0	45223866	62899450	23781

4)ROM 0x00002D8C RAM 0x0012B954

Memory map

memory.map

```
1   00000000 00002D8C ROM 4 R 1/1 1/1
2   00002D8C 0012B954 DRAM 4 RW 250/50 250/50
```

Scatter file

```
LOAD_ROM 0x0 0x00002D8C
{
EXEC_ROM 0x0 0x2D8C
{
*.o ( +RO )
}
DRAM 0x2D8C 0x0012B954
{
* (ram)
* ( +ZI )
* ( +RW )
}
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007083	17448363	8603123	6782998	2289451	0	32961678	50637250	23781

Συμπεράσματα

Αυτό που παρατηρούμε από τα Debugger Internal Statistics είναι ότι τόσο τα wait states όσο και τα total cycles στην αρχή παραμένουν σταθερά και όσο μειώνουμε τα μεγέθη των μνημών παρατηρούμε πτώση τους. Στις πρώτες δύο περιπτώσεις μένουν ακριβώς τα ίδια και στις δύο επόμενες μειώνονται. Αυτό έχει να κάνει με το κατά πόσο είναι αποδοτικότερο το να μειώνουμε τη μνήμη προσεγγίζοντας κάθε φορά στενά τα όρια που χρειαζόμαστε. Αυτό συμβαίνει διότι τα δεδομένα δεν αποθηκεύονται σειριακά στις διαθέσιμες διευθύνσεις μνήμης. Συνεπώς το να έχουμε μία αρκετά μεγάλη μνήμη(αρκετά μεγαλύτερη των δεδομένων που θέλουμε να «στοιβάξουμε») δεν μας διευκολύνει στην αποδοτικότητα λόγω των μεγάλων αποστάσεων και των περισσότερων προσπελάσεων (που ενδέχεται να γίνουν) μεταξύ των αποθηκευμένων δεδομένων. Έτσι με το να προσεγγίζουμε όσο το δυνατόν περισσότερο τα όρια που θέτουν τα δεδομένα για το μέγεθος της μνήμης μας βοηθάει στην πιο σωστή χρήση της. Όταν οι μνήμες που διαθέτουμε προσαρμόζονται ακριβώς στο μέγεθος των δεδομένων μας, τότε έχουμε το βέλτιστο αποτέλεσμα.

Επίσης αυτό που παρατειρείται είναι ότι παρόλο που το Total RO data είναι 11658 Bytes η ROM θα πρέπει να έχει ελάχιστη χωρητικότητα 11660 Bytes. Αυτό συμβαίνει επειδή η διεύθυνση που χρησιμοποιείται για λειτουργία μνήμης 32-bit (4 byte) πρέπει να ευθυγραμμιστεί με το εύρος διάυλου των 4 byte. Αυτό σημαίνει ότι η διεύθυνση πρέπει να είναι πολλαπλάσια του 4

2^{Ο} ΣΚΕΛΟΣ ΕΙΣΑΓΩΓΗ ΧΡΗΣΗΣ SRAM

Σε αυτό το σημείο θα προσπαθήσουμε να εισάγουμε και τη μνήμη SRAM και να αλλάζουμε τα μεγέθη και των τριών για να επεξεργαστούμε τα αποτελέσματα που θα προκύψουν

1)

A.ROM 0x00002D8C DRAM 0x00000134 SRAM 0x0012B820

Το λογικό εξαγόμενο που προέκυψε από τους προηγούμενους πειραματισμούς, είναι πως η χωρητικότητα μνήμης, πρεπει να καθορίζεται από την ποσότητα των δεδομένων που θα τοποθετηθούν σε αυτήν για να έχουμε την βέλτιστη λειτουργία επεξεργαστή. Επομένως, κατά την περεταίρω βελτιστοποίηση που περιμένουμε να αντικρύσουμε λόγω της χρήσης και μιας SRAM (όπου θα αποθηκεύσουμε τα επαναχρησιμοποιούμενα δεδομένα), θα προσαρμόσουμε και το συμπέρασμα των προηγούμενων ενδείξεων που λάβαμε, δηλαδή μια χρήση μνήμης «τόσο – όσο» για την ακριβή αποθήκευση των δεδομένων.

Για ανα βρούμε τα μεγέθη των SRAM και DRAM αρκεί να κάνουμε τις κατάλληλες πράξεις.

Μέσα στην SRAM θα βρίσκονται και οι 3 πίνακες current_y[N][M], A[N+2][M+2] ,newA[N+2][M+2].

```
/* code for armulator*/
#pragma arm section zidata="sram"
int current_y[N][M];
int A[N+2][M+2];
int newA[N+2][M+2];
#pragma arm section
```

Άρα το μέγεθος της SRAM θα είναι (288*352+290*354*2)*4= 1226784 = $(12B820)_{16}$ Bytes.Το μέγεθος της DRAM θα είναι Total RW size – SRAM= 308 = $(134)_{16}$ Bytes

Memory map

memory.map

```
1   00000000 00002D8C ROM 4 R 1/1 1/1
2   00002D8C 00000134 DRAM 4 RW 250/50 250/50
3   00002EC0 0012B820 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002D8C
{
EXEC_ROM 0x0 0x00002D8C
{
*.o  (+RO)
}
DRAM 0x00002D8C 0x00000134
{
* (+ZI, +RW)
}
SRAM 0x00002EC0 0x0012B820
{
*(sram)
}
}
```

Παρατηρούμε ότι παραθέτοντας τα παραπάνω Memory map και Scatter file η διαδικασία του make στο Armulator βγάζει τα ακόλουθα errors.

```
Error   : L6220E: Load region LOAD_ROM size (11680 bytes) exceeds limit (11660 bytes).
Error   : L6220E: Execution region EXEC_ROM size (11680 bytes) exceeds limit (11660 bytes).
Error   : L6221E: Execution region EXEC_ROM overlaps with Execution region DRAM.
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1176         60          0    1226792       6076   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11304        374          0    1227092      10548   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11678 (  11.40kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
```

Τα errors αυτά μας ενημερώνουν ότι το μέγεθος των LOAD_ROM και EXEC_ROM δεν είναι σωστά και μας ενημερώνει για το πόσο θα πρέπει να αλλάξουμε το μέγεθος των μνημών.Αυτό συμβαίνει καθώς όπως παρατηρούμε το Total RO Size μεγαλώνει κατά 20 Bytes. Παρόλα αυτά, λόγω του ότι 11678 είναι μη ακέραιο πολλαπλάσιο του 4(εύρος διάυλου), χρειαζόμαστε ROM 11680 Bytes. Στην προκειμένη θα πρέπει να αυξήσουμε το μέγεθος των LOAD_ROM και

EXEC_ROM κατά 20 bytes. Κατά συνέπεια προκύπτουν τα αμέσως επόμενα Memory map και Scatter file.

B.ROM 0x00002DA0 DRAM 0x00000134 SRAM 0x0012B820

Memory map

memory.map

```
1    00000000 00002DA0 ROM 4 R 1/1 1/1
2    00002DA0 00000134 DRAM 4 RW 250/50 250/50
3    00002ED4 0012B820 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002DA0
{
EXEC_ROM 0x0 0x00002DA0
{
*.o  (+RO)
}
DRAM 0x00002DA0 0x00000134
{
* (+ZI, +RW)
}
SRAM 0x00002ED4 0x0012B820
{
*(sram)
}
}
```

Σωστή εκτέλεση του make

```
==============================================================================
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1176         60          0    1226792       6076   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11304        374          0    1227092      10548   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                11678 (  11.40kB)
    Total RW  Size(RW Data + ZI Data)           1227092 (1198.33kB)
    Total ROM Size(Code + RO Data + RW Data)      11678 (  11.40kB)
==============================================================================
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007110	17448399	8603154	6783001	2289453	0	12843006	30518614	23781

2)ROM 0x00002DA0 DRAM 0x00064544 SRAM 0x000C7410

Με την ίδια λογική υπολογίζουμε το μέγεθος των DRAM και SRAM. Σε αυτή την περίπτωση, τοποθετούμε μέσα στην SRAM μόνο τους 2 πίνακες current_y[N][M], A[N+2][M+2].

```
/* code for armulator*/
#pragma arm section zidata="sram"
int current_y[N][M];
int A[N+2][M+2];
#pragma arm section
```

Προφανώς αναμένουμε αυτό να στερήσει απόδοσης, στον επεξεργαστή μας, αλλά ίσως να προσεγγίζει πιο ρεαλιστικά μεγέθη μνήμης cache για ένα κοινό υπολογιστικό σύστημα.

Άρα το μέγεθος της SRAM θα είναι (288*352+290*354)*4= 816144 $=(C7410)_{16}$ Bytes

Το μέγεθος της DRAM θα είναι Total RW size – SRAM= 410948 $=(64544)_{16}$ Bytes

Memory map

memory.map

```
1   00000000 00002DA0 ROM 4 R 1/1 1/1
2   00002DA0 00064544 DRAM 4 RW 250/50 250/50
3   000672E4 000C7410 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002DA0
{
EXEC_ROM 0x0 0x00002DA0
{
*.o  (+RO)
}
DRAM 0x2DA0 0x00064544
{
* (+ZI, +RW)
}
SRAM 0x000672E4 0x000C7410
{
*(sram)
}
}
```

Debug

Debugger Internals

Internal Variables | Statistics

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007110	17448399	8603154	6783001	2289453	0	16507950	34183558	23781

3)ROM 0x00002DA0 DRAM 0x000C8954 SRAM 0x00063000

Σε αυτή την περίπτωση μέσα στην SRAM θα έχουμε μόνο τον πίνακα current_y[N][M] συνεπώς θα αλλάξει το μέγεθός της και θα γίνει (288*352) * 4Bytes = 405504 = $(63000)_{16}$ Bytes.

```
/* code for armulator*/
#pragma arm section zidata="sram"
int current_y[N][M];
#pragma arm section
```

Σε αυτήν την περίπτωση αναμένουμε ακόμα μεγαλύτερη πτώση της επίδοσης του επεξεργαστή.

Το μέγεθος της DRAM θα γίνει Total RW size – SRAM = 821588 = $(C8954)_{16}$ Bytes

Memory map

memory.map

```
1   00000000 00002DA0 ROM 4 R 1/1 1/1
2   00002DA0 000C8954 DRAM 4 RW 250/50 250/50
3   000CB6F4 00063000 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002DA0
{
EXEC_ROM 0x0 0x00002DA0
{
*.o  (+RO)
}
DRAM 0x00002DA0 0x000C8954
{
* (+ZI, +RW)
}
SRAM 0x000CB6F4 0x00063000
{
*(sram)
}
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	9007463	17448399	8602801	6783354	2289453	0	24446094	42121702	23781

Συμπεράσματα

Όπως προείπαμε τα RO data έχουν αυξηθεί σε σχέση με τις προηγούμενες δοκιμές(χωρίς SRAM).

Αυτό που αναμέναμε και από το θεωρητικό υπόβαθρο, αποδείχθηκε και κατά την πρακτική υλοποίηση. Ότι δηλαδή όσο περισσότερη SRAM δεσμεύουμε για την «στοίβαξη» των επαναχρησιμοποιούμενων δεδομένων των πινάκων, τόσο βελτιώνεται η απόδοση του επεξεργαστή. Αυτό προφανώς μεταφράζεται σε μείωση των wait states και των total cycles και αυτό συμβαίνει γιατι η επεξεργασία των πραγματοποιείται γρηγορότερα. Παρόλα αυτά, η αρχική θεωρητική προσέγγιση της εργασίας, κατέστησε σαφές πως πρέπει η ιεραρχία μνημών που θα προτείνουμε, να αγγίζει τα όρια του ρεαλισμού για ένα μέσο υπολογιστικό σύστημα. Αυτό σημαίνει πως θα πρέπει να «στερηθούμε» μέχρι ενός σημείου την ταχύτητα απόδοσης δεδομένων στο επεξεργαστή από την cache, έτσι ώστε το κόστος να αγγίζει ένα μέσο Η/Υ.

3ο ΣΚΕΛΟΣ ΕΙΣΑΓΩΓΗ ΧΡΗΣΗΣ BUFFER

Λόγω του μεγάλου αριθμού προσπελάσεων που συμβαίνουν στους ογκώδεις πίνακες δεδομένων των οποίων τα στοιχεία επαναρησιμοποιούνται, δημιουργήσαμε τρεις μονοδιάστατους πίνακες – buffers στους οποίους θα αποθηκεύονται κατά το φιλτράρισμα ,με βηματική διαδικασία(δηλ. αρχικά buffer1→1η γραμμή πίνακα, buffer2→2η γραμμή πίνακα, buffer3→3η γραμμή πίνακα. Στη δεύτερη επανάληψη buffer1 → 2η γραμμή πίνακα, buffer2→ 3η γραμμή πίνακα , buffer3→4η γραμμή πίνακα κτλ.), τα δεδομένα του πίνακα Α. Η χρησιμότητα της διαδικασίας αυτής έγγυται στην αδυναμία υπολογιστικών συστημάτων να προσφέρουν μνήμες cache αρκετά μεγάλες, για να «χωρέσουν» το σύνολο των δεδομένων προς επεξεργασία. Παρόλα αυτά, στην δικιά μας περίπτωση λόγω απουσίας ενός τόσο μεγάλου όγκου δεδομένων, αναμένουμε να μην εμφανίσει κάποια βελτίωση-ίσως και επιβάρυνση- στον επεξεργαστή.

A.ROM 0x00002DA0 DRAM 0x0012A8BC SRAM 0x00001098

Όπως και στις προηγούμενες περιπτώσεις είμαστε σε θέση να υπολογίσουμε το μέγεθος της μνήμης SRAM άρα κατά συνέπεια και της μνήμης DRAM. Το μέγεθος της μνήμης SRAM θα υπολογιστεί ως εξής:

Εφόσον έχουμε 3 buffers οι οποίοι έχουν από 354 στοιχεία ο καθένας και είναι αποθηκευμένοι στην SRAM τότε το μέγεθός της θα υπολογιστεί ως εξής: $3*354*4= 4248 = (1098)_{16}$ Bytes

```
/* code for armulator*/
#pragma arm section zidata="sram"
int buffer1[M+2];
int buffer2[M+2];
int buffer3[M+2];
#pragma arm section
```

Άρα το μέγεθος της DRAM θα είναι Total RW size – SRAM= 1222844 = $(12A8BC)_{16}$ Bytes

Συνεπώς το Memory map και το Scatter file θα είναι τα παρακάτω

Memory map

memory.map

```
1   00000000 00002DA0 ROM 4 R 1/1 1/1
2   00002DA0 0012A8BC DRAM 4 RW 250/50 250/50
3   0012D65C 00001098 SRAM 4 RW 1/1 1/1
```

Scatter file

```
ROM 0x0 0x00002DA0
{
ROM 0x0 0x00002DA0
{
*.o  (+RO)
}
DRAM 0x00002DA0 0x0012A8BC
{
* (+ZI, +RW)
}
SRAM 0x0012D65C 0x00001098
{
*(sram)
}
}
```

Όπως παρατηρούμε παρακάτω όμως από το make που προέκυψε από το Armulator το μέγεθος τόσο της ROM όσο και της DRAM θα πρέπει να αλλάξουν.

```
Error   : L6220E: Load region ROM size (11884 bytes) exceeds limit (11680 bytes).
Error   : L6220E: Execution region ROM size (11884 bytes) exceeds limit (11680 bytes).
Error   : L6220E: Execution region DRAM size (1227096 bytes) exceeds limit (1222844 bytes).
Error   : L6221E: Execution region ROM overlaps with Execution region DRAM.
Error   : L6221E: Execution region DRAM overlaps with Execution region SRAM.
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1380         60          0    1231044       6328   Object Totals
     10128        314          0        300       4472   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11508        374          0    1231344      10800   Grand Totals
==============================================================================
```

Σύμφωνα με τα όρια που μας βγάζουν τα errors, και με τον τρόπο που συνέβη και στην σελίδα 11, πραγματοποιούμε τις κατάλληλες αλλαγές και τα Memory map και Scatter file που προκύπτουν είναι τα παρακάτω.

B. ROM 0x00002E6C DRAM 0x0012B958 SRAM 0x00001098

Memory map

memory.map

```
1   00000000 00002E6C ROM 4 R 1/1 1/1
2   00002E6C 0012B958 DRAM 4 RW 250/50 250/50
3   0012E7C4 00001098 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002E6C
{
EXEC_ROM 0x0 0x00002E6C
{
*.o  (+RO)
}
DRAM 0x00002E6C 0x0012B958
{
* (+ZI, +RW)
}
SRAM 0x0012E7C4 0x00001098
{
*(sram)
}
```

Σωστή εκτέλεση του make

```
========================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      1380         60          0    1231044       6328   Object Totals
     10128        314          0        300       4472   Library Totals
========================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     11508        374          0    1231344      10800   Grand Totals
========================================================================

    Total RO  Size(Code + RO Data)                11882 (  11.60kB)
    Total RW  Size(RW Data + ZI Data)           1231344 (1202.48kB)
    Total ROM Size(Code + RO Data + RW Data)      11882 (  11.60kB)
========================================================================
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	10541528	20516726	9831428	8163694	2749133	0	32386122	53130377	23814

2) ROM 0x00002E6C DRAM 0x000C8958 SRAM 0x00064098

Σε αυτή την περίπτωση θα ελέγξουμε τα αποτελέσματα σε περίπτωση που έχουμε μέσα στην SRAM τους 3 buffers μαζί με τον πίνακα current_y.

Έτσι τα μεγέθη των μνημών θα καθοριστούν από τις παρακάτω πράξεις

SRAM= (3*354+288*352)*4=(409752)=(64098) $_{16}$ Bytes

```
#pragma arm section zidata="sram"
int current_y[N][M];
int buffer1[M+2];
int buffer2[M+2];
int buffer3[M+2];
#pragma arm section
```

DRAM=Total RW size – SRAM=1231344-409752=(821592)=(C8958) $_{16}$ Bytes

ROM=(11884)=(2E6C) $_{16}$ Bytes

Το scatter file και το memory map και debug φαίνονται παρακάτω

Memory map

memory.map

```
1   00000000 00002E6C ROM 4 R 1/1 1/1
2   00002E6C 000C8958 DRAM 4 RW 250/50 250/50
3   000CB7C4 00064098 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002E6C
{
EXEC_ROM 0x0 0x00002E6C
{
*.o  (+RO)
}
DRAM 0x00002E6C 0x000C8958
{
* (+ZI, +RW)
}
SRAM 0x000CB7C4 0x00064098
{
*(sram)
}
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
;statistics	10541528	20516726	9831428	8163694	2749133	0	23870538	44614793	23814

3) ROM 0x00002E6C DRAM 0x0012BEE0 SRAM 0x00000B10

Σε αυτή την περίπτωση θα εξετάσουμε το ενδεχόμενο να έχουμε περισσότερους πίνακες στην DRAM και λιγότερους στην SRAM. Προφανώς περιμένουμε τα αποτελέσματα να είναι χειρότερα από την προηγούμενη δοκιμή καθώς όπως έχουμε πει επανειλημμένα η SRAM είναι πολύ γρηγορότερη της DRAM. Πιο συγκεκριμένα σε αυτή τη περίπτωση θα έχουμε μόνο τους 2 buffers στην SRAM και όλους τους υπόλοιπους στην DRAM.

Συνεπώς τα μεγέθη για τις μνήμες δίνονται από τις παρακάτω συνήθεις πράξεις.

SRAM=2*354*4=(2832)=(B10) $_{16}$ Bytes

```
/* code for armulator*/
#pragma arm section zidata="sram"
int buffer1[M+2];
int buffer2[M+2];
#pragma arm section
```

DRAM=Total RW size – SRAM=(1228512)=(12BEE0) $_{16}$ Bytes

ROM=(11884)=(2E6C) $_{16}$ Bytes

Το scatter file, memory map και το debug φαίνονται παρακάτω.

Memory map

memory.map

```
1   00000000 00002E6C ROM 4 R 1/1 1/1
2   00002E6C 0012BEE0 DRAM 4 RW 250/50 250/50
3   0012ED4C 00000B10 SRAM 4 RW 1/1 1/1
```

Scatter file

```
LOAD_ROM 0x0 0x00002E6C
{
EXEC_ROM 0x0 0x00002E6C
{
*.o  (+RO)
}
DRAM 0x00002E6C 0x0012BEE0
{
* (+ZI, +RW)
}
SRAM 0x0012ED4C 0x00000B10
{
*(sram)
}
}
```

Debug

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycle
$statistics	10541528	20516726	9831428	8163694	2749133	0	36049482	56793737	23814

Συμπεράσματα

Αρχικά παρατηρούμε ότι εμφανώς τα ZI data έχουν αυξηθεί το ίδιο και ο κώδικας καθώς έχουμε τρείς καινούριους μονοδιάστατους πίνακες οι οποίοι αρχικοποιούνται.

Όπως και πριν παρόλο που το Total RO size είναι 11882 Bytes θα πρέπει να το αυξήσουμε στα 11884 Bytes καθώς αυτό είναι το όριο για την μνήμη ROM.

Παρατηρούμε πως η χρήση πινάκων προσωρινής αποθήκευσης (buffers) κάθε φορά τμημάτων των συνολικών δεδομένων που υπάρχουν στους μεγάλους πίνακες όχι μόνο δεν μπόρεσε να μειώσει τον συνολικό αριθμό κύκλων- και τα wait states σε σχέση με το πέρας της προηγούμενης διαδικασίας αλλά αντίθετα τον αυξήσε. Παρόλα αυτά αυτό δεν καθιστά τη συγκεκριμένη διαδικασία αναξιόπιστη ή ασήμαντη καθώς αν επιθυμούμε να χρησιμοποιούμε την γρηγορότερη από την DRAM, SRAM, συχνά θα πρέπει να αποθηκεύουμε σε αυτή μικρότερα τμήματα των πινάκων τα οποία να είναι συχνά προσπελάσιμα καθώς αυτό προϋποθέτει η ίδια της η χρήση. Συνεπώς όχι μόνο δεν την απορρίπτουμε αλλά πολλές φορές δεν μπορούμε να χρησιμοποιήσουμε την SRAM χωρίς την χρήση buffers, λόγω του μεγάλου μη συμβατού με το μέγεθος της cache όγκου δεδομένων που επαναχρησιμοποιούνται στο πρόγραμμα.

Άρα καταλήγουμε στο συμπέρασμα ότι για να βρισκόμαστε εντός ρεαλιστικών πλαισίων σε ότι αφορά τη χρήση πινάκων προσωρινής αποθήκευσης(buffers) από το τρίτο σκέλος, παρόλο που το πιο αποδοτικό σε ότι έχει να κάνει με τα wait states και total cycles είναι η δεύτερη υλοποίηση θα πρέπει να επιλέξουμε την πρώτη. Αυτό γιατί τα φυσιολογικά όρια μίας μνήμης SRAM(επιπέδου 1) με χρόνο προσπέλασης 1 ns είναι μέχρι 64Kbytes. Συνεπώς επιλέγουμε την πρώτη περίπτωση, που αποθέτουμε 4Kbytes δεδομένων στην SRAM αντί της δεύτερης που αποθέτουμε 409Kbytes στην SRAM. Έτσι χάνουμε λίγο σε απόδοση, κερδίζοντας λίγο σε κόστος.

ΠΗΓΕΣ

ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ ΣΧΕΔΙΑΣΜΟΥ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

https://stackoverflow.com/questions/20926386/what-is-non-aligned-access-arm-keil

http://architecture.di.uoa.gr/k5en12.html

https://en.wikipedia.org/wiki/Wait_state

https://www.secnews.gr/164971/%CE%B4%CE%B9%CE%B1%CF%86%CE%BF%CF%81%CE%AC-ram-rom/

https://el.wikipedia.org/wiki/%CE%9A%CF%81%CF%85%CF%86%CE%AE_%CE%BC%CE%BD%CE%AE%CE%BC%CE%B7

https://www.pcsteps.gr/57215-%CE%BC%CE%BD%CE%AE%CE%BC%CE%B7-cache-%CE%B5%CF%80%CE%B5%CE%BE%CE%B5%CF%81%CE%B3%CE%B1%CF%83%CF%84%CE%AE%CF%82-%CE%B5%CF%80%CE%B9%CE%B4%CF%8C%CF%83%CE%B5%CE%B9%CF%82/

https://el.wikipedia.org/wiki/%CE%A0%CF%81%CE%BF%CF%83%CF%89%CF%81%CE%B9%CE%BD%CE%AE_%CE%BC%CE%BD%CE%AE%CE%BC%CE%B7_(%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%AD%CF%82)

https://www.digispot.gr/diadiktyo/1473/ti-einai-kai-pos-leitourgei-to-buffering/

https://www.enterprisestorageforum.com/storage-hardware/sram-vs-dram.html

https://el.wikipedia.org/wiki/%CE%9C%CE%BD%CE%AE%CE%BC%CE%B7_%CF%84%CF%85%CF%87%CE%B1%CE%AF%CE%B1%CF%82_%CF%80%CF%81%CE%BF%CF%83%CF%80%CE%AD%CE%BB%CE%B1%CF%83%CE%B7%CF%82