2ο μερος εργασίας ομάδας 34 :

Μαρουλάκος - Σεφεριάδης Πάρης : 57128
Μπαντήλας Παναγιώτης : 57138

Εισαγωγή

Αρχικά είναι απαραίτητη η επεξήγηση των επιμέρους στοιχείων που εξετάζουμε στο σε αυτό το σκέλος της εργασίας, γι'αυτό τον λόγο ακολουθεί μια σύντομη περιγραφή αυτών των στοιχείων.

ROM (Read Only Memory)

Οι μνήμες μόνο για ανάγνωση (Read-only memories – ROM) είναι προγραμματισμένες από πριν με σταθερά δεδομένα. Είναι ιδιαίτερα χρήσιμες στα ενσωματωμένα συστήματα δεδομένου ότι σημαντικό μέρος του κώδικα και ενδεχομένως ορισμένα δεδομένα δεν αλλάζουν στο χρόνο.Η μνήμη ROM είναι μη πτητική, δηλαδή δε χάνει τα δεδομένα της με τη διακοπή τροφοδοσίας του ρεύματος. Γι' αυτό σ' αυτήν φορτώνουμε όλα τα δεδομένα τα οποία είναι απαραίτητα έτσι ώστε να πρόγραμμα να εκκινεί πάντα σωστά (π.χ. μετά από κάποιο reset). Η μνήμη ROM είναι πολύ πιο φθηνή από τη RAM, αλλά συνήθως είναι πιο αργή και επίσης δεν μπορούν όλα τα δεδομένα της να προσπελαστούν απευθείας από την ΚΜΕ.
Οι μνήμες ROM είναι επίσης λιγότερο ευαίσθητες στα σφάλματα που προκαλούνται λόγω ακτινοβολίας. Υπάρχουν διαθέσιμες διάφορες ποικιλίες μνημών ROM.

RAM (Random Access Memory)

Οι μνήμες τυχαίας προσπέλασης μπορούν τόσο να διαβαστούν όσο και να γραφούν. Καλούνται τυχαίας προσπέλασης επειδή, αντίθετα από τους μαγνητικούς δίσκους, οι διευθύνσεις μπορούν να διαβαστούν με οποιαδήποτε σειρά.Υπάρχουν δύο βασικές κατηγορίες μνήμης τυχαίας προσπέλασης: η στατική RAM (static RAM – SRAM) κατασκευασμένη από transistor και latches και η δυναμική RAM (dynamic RAM – DRAM) κατασκευασμένη από transistor και πυκνωτές. Αυτοί οι δυο τύποι έχουν σημαντικές διαφορές στα χαρακτηριστικά τους.

- Η SRAM είναι γρηγορότερη από τη DRAM.
- Η SRAM καταναλώνει περισσότερη ενέργεια από τη DRAM.
- Μπορούμε να τοποθετήσουμε περισσότερη DRAM σε ένα απλό chip.
- Οι τιμές των δεδομένων στις DRAM πρέπει να ανανεώνονται (refreshed) περιοδικά.

Περιγραφή περιοχών μνήμης

Στη ROM φορτώνουμε τόσο τα RO data και τον κώδικα όσο και τα RW data παρόλο που κατά τη διάρκεια της εκτέλεσης τα RW data θα αντιγραφούν στη μνήμη RAM για προφανείς λόγους (η ROM είναι προορισμένη μόνο για ανάγνωση). Κατά τη διάρκεια της εκτέλεσης στη ROM βρίσκονται οι σταθερές (π.χ. *const char c=123;*) καθώς και οι τιμές στις οποίες αρχικοποιούμε global μεταβλητές(π.χ. το 31 όταν αρχικοποιούμε τη global μεταβλητή *int d=31;*).Επίσης, όπως προείπαμε, βρίσκεται ήδη φορτωμένα τα αρχεία του αντικειμενικού κώδικα και οι βιβλιοθήκες καθώς και κάποια περιεχόμενα του αρχικού κώδικα (π.χ. τιμές τις οποίες εκχωρούμε απευθείας σε μεταβλητές και τιμές οι οποίες ορίζονται με το define).
Στη RAM κατά τη διάρκεια της εκτέλεσης αποθηκεύονται τα zero-initialized data, δηλαδή global μεταβλητές οι οποίες όταν "κατεβαίνουν" στον επεξεργαστή αρχικοποιούνται στο μηδέν, αντίστοιχα εκεί αποθηκεύονται και οι global μεταβλητές οι οποίες αρχικοποιούνται από τον προγραμματιστή(όπως προείπαμε στη RAM αποθηκεύεται μόνο η global μεταβλητή, ενώ η τιμή στην οποία αρχικοποιείται αποθηκεύεται στη ROM). Επίσης στη RAM βρίσκονται και δύο στοίβες η stack στην οποία αποθηκεύονται οι local μεταβλητές και αυξάνεται (γίνεται pop) προς τα κάτω και η heap η οποία χρησιμοποιείται για τη δυναμική δεύσμευση μνήμης και η οποία αυξάνεται προς τα πάνω.

Memory Map

Λαμβάνοντας υπ' όψη την κατάταξη των μνημών -όπως παρουσιάστηκε παραπάνω- όσον αφορά την ταχύτητα και ανατρέχοντας σε ορισμένα datasheets, καταταλήξαμε στους παρακάτω χρόνους ανάγνωσης/ εγγραφής:

- Για την ROM 250 ns χρόνος ανάγνωσης.
- Για την DRAM 150 ns χρόνος ανάγνωσης και 250 ns χρόνος εγγραφής.

- Για την SRAM 55 ns χρόνος ανάγνωσης και χρόνος εγγραφής.

Στα memory maps όταν δηλώνουμε τα χαρακτηριστικά και οριοθετούμε το μέγεθος της ROM βάλαμε ως χρόνο εγγραφής σειριακών και μη-σειριακών δεδομένων ίσο με 1. Αυτός ο χρόνος είναι προφανώς φαινομενικός καθώς η ROM δεν επιτρέπει την εγγραφή (δηλώνεται και ως R και όχι RW). Ως εύρος διάυλου ορίσαμε τα 4 Bytes για όλες τις μνήμες.

Κώδικας

Παρακάτω βλέπουμε τα δεδομένα που μας έδοσε ο αλγόριθμος που υλοποιήσαμε στο 1^{ο} σκέλος της εργασίας :

Image Component Sizes:

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

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         60          8    4481116       6184   Object Totals
     20564        466          0        300       8484   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23820        526          8    4481416      14668   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24346 (  23.78kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
    Total ROM Size(Code + RO Data + RW Data)      24354 (  23.78kB)
==============================================================================
```

Debugger Internal Statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Total
$statistics	276397576	399658854	311181398	67600232	58255214	0	437036844

1. Μέγεθος μνημών ROM και RAM

Το πρώτο πράγμα το οποίο διερευνήσαμε είναι αν το μέγεθος των μνημών επηρεάζει τους συνολικούς κύκλους και με ποιον τρόπο.

Στις μετρήσεις που πραγματοποιήσαμε, επικεντρώνοντας σε αυτό το ζήτημα, λάβαμε τα ίδια αποτελέσματα όσον αφορά τα image component sizes εκτός δύο περιπτώσεων οπου το μέγεθος της RAM και της ROM ξεπερνούσε τα όρια.

1. ROM 0x80000 // RAM 0x8000000

Image Component Sizes:

```
Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         40          8    4481116       7608   Object Totals
     20496        466          0        300       8408   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        506          8    4481416      16016   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                24258 (  23.69kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
    Total ROM Size(Code + RO Data + RW Data)      24266 (  23.70kB)
================================================================================
```

Debugger Internal Statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	275890601	398542948	310674272	67295874	57950792	0	1073899762	1509820700	19950912

Memory Map:

```
00000000 00080000 ROM 4 R 250/1 250/1
00080000 08000000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x08000000
{
    ROM 0x0 0x80000
    {
        *.o  (+RO)
    }
    RAM 0x80000 0x8000000
    {
        * (+ZI, +RW)
    }
}
```

2. **ROM 0x40000 // RAM 0x4000000**

Image Component Sizes:

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

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         40          8    4481116       7548   Object Totals
     20496        466          0        300       8408   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        506          8    4481416      15956   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24258 (  23.69kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
    Total ROM Size(Code + RO Data + RW Data)      24266 (  23.70kB)
==============================================================================
```

Debugger Internal Statistics:

Reference Points	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True Idle Cycles
$statistics	2758890601	398542948	3106674272	67295874	57950792	0	1073899762	1509820700	19950912

Memory Map:

```
00000000 00040000 ROM 4 R 250/1 250/1
00040000 04000000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x04000000
{
    ROM 0x0 0x40000
    {
        *.o   (+RO)
    }
    RAM 0x40000 0x4000000
    {
        * (+ZI, +RW)
    }
}
```

3. ROM 0x8000 // RAM 0x800000

Image Component Sizes:

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

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         40          8    4481116       7548   Object Totals
     20496        466          0        300       8408   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        506          8    4481416      15956   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24258 (  23.69kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
    Total ROM Size(Code + RO Data + RW Data)      24266 (  23.70kB)
==============================================================================
```

Debugger Internal Statistics:

Reference Points	Instructions	Core Cycles	S Cycles	N Cycles	I Cycles	C Cycles	Wait States	Total	True Idle Cycles
$statistics	2758901601	3985342948	3106742272	67295874	57950792	0	10738999762	15098220700	19950912

Memory Map:

```
00000000 00008000 ROM 4 R 250/1 250/1
00008000 00800000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x0800000
{
    ROM 0x0 0x8000
    {
        *.o   (+RO)
    }
    RAM 0x8000 0x800000
    {
        * (+ZI, +RW)
    }
}
```

---

4. ROM 0x4000 // RAM 0x40000

Image Component Sizes:

```
Error   : L6220E: Execution region ROM size (24260 bytes) exceeds limit (16384 bytes).
Error   : L6220E: Execution region RAM size (4481424 bytes) exceeds limit (4194304 bytes).
Error   : L6221E: Execution region ROM overlaps with Execution region RAM.
================================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         40          8    4481116       7548   Object Totals
     20496        466          0        300       8408   Library Totals
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        506          8    4481416      15956   Grand Totals
================================================================================

    Total RO  Size(Code + RO Data)                24258 (  23.69kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
```

Memory Map:

```
00000000 00004000 ROM 4 R 250/1 250/1
00004000 00400000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x0400000
{
    ROM 0x0 0x4000
    {
        *.o   (+RO)
    }
    RAM 0x4000 0x400000
    {
        * (+ZI, +RW)
    }
}
```

5. ROM 0x3000 // RAM 0x30000

Image Component Sizes:

```
Error   : L6220E: Execution region ROM size (24260 bytes) exceeds limit (12288 bytes).
Error   : L6220E: Execution region RAM size (4481424 bytes) exceeds limit (3145728 bytes).
Error   : L6221E: Execution region ROM overlaps with Execution region RAM.
==============================================================================

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         40          8    4481116       7548   Object Totals
     20496        466          0        300       8408   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        506          8    4481416      15956   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24258 (  23.69kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
```

Memory Map:

```
00000000 00003000 ROM 4 R 250/1 250/1
00003000 00300000 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x0300000
{
    ROM 0x0 0x3000
    {
        *.o   (+RO)
    }
    RAM 0x3000 0x300000
    {
        * (+ZI, +RW)
    }
}
```

6. ROM_0x5EC4 RAM_0x446190

Image Component Sizes:

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

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         40          8    4481116       7548   Object Totals
     20496        466          0        300       8408   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        506          8    4481416      15956   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24258 (  23.69kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
    Total ROM Size(Code + RO Data + RW Data)      24266 (  23.70kB)
==============================================================================
```

Debugger Internal Statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	275890585	398542924	310674257	67295867	57950790	0	1073899664	1509820578	19950912

Memory Map:

```
00000000 00005EC4 ROM 4 R 250/1 250/1
00005EC4 00446190 RAM 4 RW 150/250 150/250
```

Scatter File:

```
ROM 0x0 0x00446190
{
    ROM 0x0 0x5EC4
    {
        *.o  (+RO)
    }
    RAM 0x5EC4 0x446190
    {
        * (+ZI, +RW)
    }
}
```

Συμπεράσματα

Όσο μειώνεται η χωρητικότητα των μνημών, τα wait states και τα total cycles μειώνονται ή παραμένουν σταθερά. Φυσικά, αυτό γίνεται μέχρι τα δεδομένα τα οποία προορίζονται για την εκάστοτε μνήμη να «χωράνε» ακριβώς, οπότε έχουμε και το βέλτιστο αποτέλεσμα. Αυτό συμβαίνει διότι εμείς μπορεί να έχουμε την εντύπωση ότι τα δεδομένα αποθηκεύονται σειριακά, αλλά αυτό δεν ισχύει. Έτσι όταν η μνήμη είναι μεγαλύτερη, έχουμε και μεγαλύτερα πήγαινε-έλα μέσα στην μνήμη για την προσπέλαση των δεδομένων. Αξίζει να τονιστεί ότι κατά αυτό τον τρόπο κάνουμε σωστή χρήση της μνήμης.

2. ROM, DRAM και SRAM

Σε αυτό το μέρος εξειδικεύσαμε περισσότερο την έννοια της RAM και την χωρίσαμε στα δύο. Έτσι βασιζόμενοι και στο θεωρητικό μέρος θεωρήσαμε πως πρέπει να έχουμε μία DRAM στην οποία θα αποθηκεύονται γενικά τα Zero-Initialized και Read/Write δεδομένα (όπως χρησιμοποιούσαμε προηγουμένως τη RAM) και μία SRAM στην οποία θα χρησιμοποιούμε ορισμένα δεδομένα τα οποία χρησιμοποιούμε συχνότερα και αξίζει να έχουμε ταχύτερη πρόσβαση σε αυτά. Με βάση τα προηγούμενα οριοθετήσαμε το μέγεθος των μνημών τόσο όσο απαιτείται έτσι ώστε τα δεδομένα που προορίζονται να αποθηκευτούν σε αυτές να «χωράνε» ακριβώς.

ROM 0x5EE0 // DRAM 0x 3E3190 // SRAM 0x63000

Pragma section:

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

Όπως φαίνεται και παραπάνω, για την SRAM προορίζεται αρχικά μόνο ο πίνακας (current_y).

N=288 και M=352,

οπότε το μέγεθος της SRAM προκύπτει ως εξής: $(288 * 352) * 4Bytes = 405504 = (63000)_{16}$ Bytes

Το μέγεθος της DRAM προκύπτει ως εξής: Total RW Size – SRAM Size = $3E3190_{16}$

Το μέγεθος της ROM προκύπτει όπως και προηγουμένως: Total ROM Size + 2 Bytes =$5EE0_{16}$

Scatter file:

```
ROM 0x0 0x446190
{
    ROM 0x0 0x5ee0
    {
        *.o  (+RO)
    }
    DRAM 0x5ee0 0x3e3190
    {
        * (+ZI, +RW)
    }
    SRAM 0x3e9070 0x63000
    {
        *(sram)
    }
}
```

Memory Map:

```
00000000 00005ee0 ROM 4 R 250/1 250/1
00005ee0 003e3190 DRAM 4 RW 150/250 150/250
003e9070 00063000 SRAM 4 RW 55/55 55/55
```

Debugger statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
$statistics	275890628	398542984	310674303	67295877	57950794	0	1122593916	1558514890	19950912

ROM 0x5EE0 // DRAM 0x37ED80 // SRAM 0x12B820

Pragma section:

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

Όπως φαίνεται και παραπάνω, για την SRAM προορίζεται ο πίνακας current_y και άλλοι δύο προσωρινοί πίνακες(φίλτρο gauss)

N=288 και M=352, οπότε το μέγεθος της SRAM προκύπτει ως εξής: $(288 * 352 + 290 * 354 + 290 * 354) * 4Bytes = 1226784 = (12B820)_{16}\ Bytes$

Το μέγεθος της DRAM προκύπτει ως εξής: Total RW Size – SRAM Size = $(37ED80)_{16}$Bytes

Το μέγεθος της ROM προκύπτει όπως και προηγουμένως: Total ROM Size + 2 Bytes $(5EE0)_{16}$Bytes

Scatter file:

```
ROM 0x0 0x446190
{
    ROM 0x0 0x5ee0
    {
        *.o  (+RO)
    }
    DRAM 0x5ee0 0x37ed80
    {
        * (+ZI, +RW)
    }
    SRAM 0x384c60 0x12b820
    {
        *(sram)
    }
}
```

Memory Map:

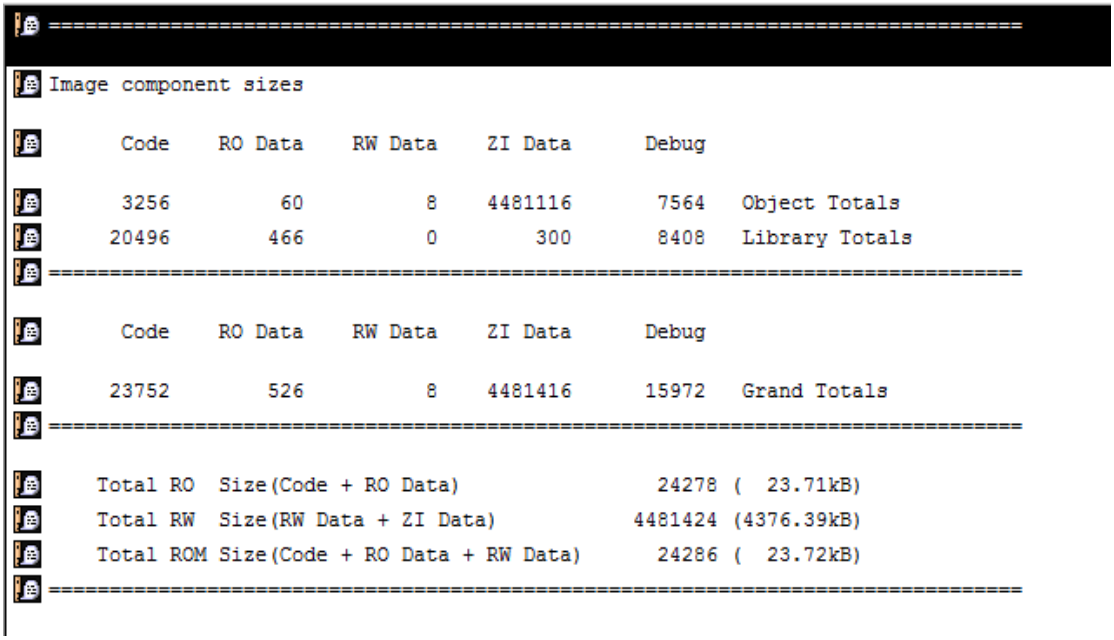
```
00000000 00005ee0 ROM 4 R 250/1 250/1
00005ee0 0037ed80 DRAM 4 RW 150/250 150/250
00320850 00384c60 SRAM 4 RW 55/55 55/55
```

Debugger statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
&statistics	275890628	398542984	310674303	67295877	57950794	0	1068033252	1503954226	19950912

Παρατηρήσεις & Συμπεράσματα

Αρχικά και στις δύο περιπτώσεις έχουμε ίδια Data



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

Image component sizes

      Code    RO Data    RW Data    ZI Data      Debug

      3256         60          8    4481116       7564   Object Totals
     20496        466          0        300       8408   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     23752        526          8    4481416      15972   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24278 (  23.71kB)
    Total RW  Size(RW Data + ZI Data)           4481424 (4376.39kB)
    Total ROM Size(Code + RO Data + RW Data)      24286 (  23.72kB)
==============================================================================
```

Όσον αφορά το μέγεθος των δεδομένων, παρατηρούμε ότι τα RO data έχουν αυξηθεί σε σχέση με τις μετρήσεις που κάναμε όταν χρησιμοποιούσαμε μόνο ROM και RAM

Έπειτα, όσον αφορά τους κύκλους, παρατηρούμε ότι τα wait states και τα total cycles μειώνονται όσο αυξάνουμε το πλήθος των δεδομένων που αποθηκεύονται στην SRAM κάτι το οποίο είναι αναμενόμενο. Ακόμη παρατηρούμε πώς οι χωρητικότητες των SRAM και DRAM πρέπει να αθροίζονται στο Total RW Size (RW Data + ZI Data). Και αυτό για να έχουμε τους λιγότερους δυνατούς κύκλους.(Προφανώς, έχουμε λιγότερα wait states και total cycles αν αποθηκεύσουμε όλα τα RW και ZI data στην SRAM παρόλα αυτά στα πραγματικά συστήματα προτιμούμε μεγαλύτερη DRAM από SRAM για λόγους κόστους)

3. Χρήση πινάκων προσωρινής αποθήκευσης

Σε αυτό το σημείο χρησιμοποιήσαμε πίνακες δεδομένων προσωρινής αποθήκευσης(συγκεκριμένα 3 πίνακες) τμήματος δεδομένων από μεγάλου μεγέθους πίνακα δεδομένων που εμφανίζουν μεγάλο αριθμό προσπελάσεων.

ROM 0x6140 // DRAM 0x446194 // SRAM 0x1098

Pragma section:

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

Τα μεγέθη των μνημών υπολογίστηκαν όπως προηγουμένως.

Scatter file:

```
ROM 0x0 0x44722c
{
    ROM 0x0 0x6140
    {
        *.o  (+RO)
    }
    DRAM 0x6140 0x446194
    {
        * (+ZI, +RW)
    }
    SRAM 0x44c2d4 0x1098
    {
        *(sram)
    }
}
```

Memory Map:

```
00000000 00006140 ROM 4 R 250/1 250/1
00006140 00446194 DRAM 4 RW 150/250 150/250
0044c2d4 00001098 SRAM 4 RW 55/55 55/55
```

Debugger statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
&statistics	114074168	183698614	129467738	43621276	12572406	0	554905123	740566543	699544

ROM 0x2EC0 // DRAM 0xC8958 // SRAM 0x64098

Pragma section:

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

int i,j,k;
```

Τα μεγέθη των μνημών υπολογίστηκαν όπως προηγουμένως.

Scatter file:

```
1  ROM 0x0 0x44722c
2  {
3      ROM 0x0 0x6140
4      {
5          *.o  (+RO)
6      }
7      DRAM 0x6140 0x3e3194
8      {
9          * (+ZI, +RW)
10     }
11     SRAM 0x3e92d4 0x64098
12     {
13         *(sram)
14     }
15 }
```

Memory Map:

```
1  00000000 00006140 ROM 4 R 250/1 250/1
2  00006140 003e3194 DRAM 4 RW 150/250 150/250
3  003e92d4 00064098 SRAM 4 RW 55/55 55/55
```

Debugger statistics:

Reference Points	Instructions	Core_Cycles	S_Cycles	N_Cycles	I_Cycles	C_Cycles	Wait_States	Total	True_Idle_Cycles
&statistics	114074168	183898614	129467738	43621276	12572406	0	548841753	734503173	699544

Παρατηρήσεις & Συμπεράσματα

Αρχικά προφανώς έχουμε τα ίδια Data και στις 2 περιπτώσεις

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

      Code    RO Data    RW Data    ZI Data      Debug

      3864         60          8    4485368       8056   Object Totals
     20496        466          0        300       8408   Library Totals
==============================================================================

      Code    RO Data    RW Data    ZI Data      Debug

     24360        526          8    4485668      16464   Grand Totals
==============================================================================

    Total RO  Size(Code + RO Data)                24886 (  24.30kB)
    Total RW  Size(RW Data + ZI Data)           4485676 (4380.54kB)
    Total ROM Size(Code + RO Data + RW Data)      24894 (  24.31kB)
==============================================================================
```

Όσον αφορά το μέγεθος των δεδομένων, παρατηρούμε ότι τα ZI data έχουν αυξηθεί σε σχέση με τις άλλες περιπτώσεις, αφού οι 3 καινούριοι μονοδιάστατοι πίνακες θα αρχικοποιηθούν με μηδενικά όταν «κατέβουν» στον επεξεργαστή. Και για προφανείς λόγους αυξήθηκε και ο κώδικας. **Έπειτα, όσον αφορά τους κύκλους, παρατηρούμε ότι τα wait states και τα total cycles μειώθηκαν(περίπου στα μισά) αφού εισάγαμε τους πίνακες δεδομένων προσωρινής αποθήκευσης**. Επίσης παρατηρούμε ξανά πως όσα περισσότερα δεδομένα αποθηκεύσω στην SRAM τόσο λιγότεροι είναι οι συνολικοί κύκλοι. Άρα η συγκεκριμένη τακτική έχει νόημα να χρησιμοποιηθεί στον κώδικα μας.