**Final Project: Jurassic Park**

**Parth Patel**

**Spring 2020**

Contents

[List of Figures and Tables 4](#_Toc39682948)

[Introduction 5](#_Toc39682949)

[Purpose 5](#_Toc39682950)

[Assumptions 5](#_Toc39682951)

[Design 5](#_Toc39682952)

[Peripherals 5](#_Toc39682953)

[ Stepper Motor 5](#_Toc39682954)

[ DC Motor 6](#_Toc39682955)

[ Keypad 6](#_Toc39682956)

[ Potentiometer: 6](#_Toc39682957)

[ LCD: 6](#_Toc39682958)

[ IRQ: 6](#_Toc39682959)

[ Speaker 6](#_Toc39682960)

[ Switches 6](#_Toc39682961)

[ LEDs 6](#_Toc39682962)

[ RTI 6](#_Toc39682963)

[Software Implementation 6](#_Toc39682964)

[Design Changes 9](#_Toc39682965)

[Project Additions 9](#_Toc39682966)

[Division of Work 9](#_Toc39682967)

[Functionality of Project 10](#_Toc39682968)

[Working 10](#_Toc39682969)

[Not working 10](#_Toc39682970)

[Conclusion 11](#_Toc39682971)

[Future Improvements 11](#_Toc39682972)

[Appendix I: User Manual 12](#_Toc39682973)

[Common Instructions and Notes 12](#_Toc39682974)

[Contents of each Menu Descriptions of major files 12](#_Toc39682975)

[Main Menu 12](#_Toc39682976)

[Traveling 12](#_Toc39682977)

[Pen 1-3 13](#_Toc39682978)

[Feed Menu 13](#_Toc39682979)

[Emergency Menu 14](#_Toc39682980)

[RTI 14](#_Toc39682981)

[Appendix II: Code 15](#_Toc39682982)

[Main.asm 15](#_Toc39682983)

[Keypad.asm 23](#_Toc39682984)

[Traveling.asm 97](#_Toc39682985)

[Pen3.asm 102](#_Toc39682986)

[Pen2.asm 107](#_Toc39682987)

[Pen1.asm 112](#_Toc39682988)

[Feed1.asm 117](#_Toc39682989)

[Feed2.asm 122](#_Toc39682990)

[Feed3.asm 127](#_Toc39682991)

[Delay2.asm 132](#_Toc39682992)

[Roar1.asm 134](#_Toc39682993)

[Roar2.asm 134](#_Toc39682994)

[Roar3.asm 135](#_Toc39682995)

[Emergency\_menu.asm 135](#_Toc39682996)

[RTI\_ISR.asm 142](#_Toc39682997)

[Gameover.asm 147](#_Toc39682998)

[IRQ\_ISR.asm 149](#_Toc39682999)

[Appendix III: Project and Report Requirements 149](#_Toc39683000)

[Project Requirements: 150](#_Toc39683001)

[Report Requirements: 152](#_Toc39683002)

# List of Figures and Tables

**Figure 1: General Menu Flowchart 6**

**Figure 2: Pen Flowchart 7**

**Figure 3: Emergency Flowchart 7**

# Introduction

Purpose**:**

The purpose of this project is to combine all the ideas from the previous labs into one final project. This project combines all the peripherals used throughout the semester to create a Jurassic park simulation. The park has multiple pens that each house a dinosaur, each pen can be visited by using a jeep to travel to each pen, the park is also equipped with a security system incase a dinosaur manages to escape. All the meus display using the lcd and the keypad and potentiometer are used to interact with the menus. The speaker is used to create the dinosaur and security sound. The motors are used to function as the park rangers watch and the motor of the jeep. The dinosaurs have various stats that are tracked with variables and displayed using the leds and lcd sceen.

Assumptions**:**

Some of the assumptions made for this project were the exact timing for the RTI since not all time was specified so some assumptions were made to make sure the park ran smoothly. All of these values were stored as separate variables and used and reset by the RTI whenever the park resets. Another assumption that was made was that the jeep speed was the exact same as the potentiometer value and did not need to be changed. Another assumption that was made was how often the different dinosaurs would require food and how long it would take before they destroyed the park.

# Design

Peripherals

The microcontroller used for this project was the Motorola 68HC12 and this is the only piece of equipment required to run the park. This microcontroller has 2 8-bit accumulators A & B that combine into a 16-bit accumulator D. There are also 2 more 16-bit registers X & Y that are used to move and store data. The microcontroller also has various other peripherals that will be listed below.

* Stepper Motor**:** The motor should spin clockwise every second if the park is stable, and as soon as a dinosaur escapes and the park starts lockdown the motor will spin faster.
* DC Motor**:** This motor serves as a representation of the jeep motor. The motor should only move as the jeep travels between pens. The speed depends on the speed of the jeep.
* Keypad**:** The keypad is used to enter in the password to the system and make all the menu selections.
* Potentiometer: The potentiometer was used to navigate the menus and control the speed of the jeep while traveling.
* LCD: The LCD screen is used to display the various menus and any emergency alerts.
* IRQ: The IRQ was used to call for security after a dinosaur has broken out so that the park can return to normal
* Speaker**:** The speaker was used to play the sounds of the dinosaurs and sound an alarm if a dinosaur broke out
* Switches**:** The switches serve as a parking brake for the jeep and must be in a certain configuration before the jeep can move. Switches 0-6 must be high for the jeep to move. The switches also serve as an enable for the alarm. The only way the alarm can sound is if switch 7 is high.
* LEDs**:** The LEDs display the current hunger level of a dinosaur if that pen is visited.
* RTI**:** The RTI controls all the timing for the park including dinosaur hunger, travel time, stepper motor time, and the time required for the dinosaur to destroy the park.

## Software Implementation

The software implementation is showed through the various flowcharts below. The main file initializes all the variables need for the park and asks the user for their credentials before allowing them to choose a pen to visit. After a selection is made the user is taken to the traveling section that relies on the travel flag to set an RTI to keep the user in the travel menu for a set amount of time before they arrive at a pen. After the amount of time has passed the travel flag is reset and the user arrives at a pen that allows the user to look at that dinosaurs’ stats and either feed or hear it roar. Depending on the choice that was made the user will be taken to a separate file to do that function before returning to the pen. The feed function will allow the user to keep feeding the dinosaur before returning to the pen. If the dinosaur is given the correct food the hunger level will go up and if they are given the incorrect food the anger level goes up. The roar section just plays the sound before returning to the pen. In the background during this whole process at different time intervals each dinosaurs’ hunger level is decreasing. If a dinosaur becomes hungry enough or angry enough, they will break out causing the emergency flag to turn on and that causes the rti to call the emergency menu subroutine. If this subroutine is called, then the user has a set amount of time to enter their credentials again and call security. Depending on what happens during this time a different message will play on the LCD and the park will be reset.

A screenshot of a cell phone

Description automatically generated

Figure 1 General Flow Chart

A picture containing computer

Description automatically generated

Figure 2 Pen Flow Chart

A screenshot of a cell phone

Description automatically generated

Figure 3 Emergency Flow Chart

**Error Handling and Failsafe Techniques**

Failsafe techniques for this project include:

* The jeep will not start unless parking brake is off
* The system requires a password before starting the park
* The alarm must be enabled

Design Changes

The only major change that was made for this project was to change the push button requirement to the IRQ instead. The project does exactly as instructed otherwise.

Project Additions

There were very additions to this project other then the hidden anger stat amongst dinosaurs. The dinosaurs have a hidden anger stat that goes up if they are fed the incorrect food for their diet and the only way to reset anger is to give them cake no matter the dinosaur.

# Division of Work

Since this was a solo project I, Parth Patel, did all the required work for this project. This includes all the assembly coding and debugging along with the report.

Functionality of Project

Working**:**

* **Stepper Motor**
* **DC motor (partially)**
* **Keypad**
* **Potentiometer**
* **LCD**
* **IRQ**
* **Speaker (partially)**
* **Switches**
* **LEDS**
* **RTI**

Everything not marked as partially works as instructed through the use of the RTI and various subroutines in this project.

Not working**:**

* **DC motor only turns on and off as the jeep travels. There is no speed control and the dc motor only turns at one speed while the jeep is in motion. This is mostly since I was not able to practice using this component on a real board so testing and debugging was much harder to do.**
* **Speaker currently plays a random sound from memory every time that it is called. This was due to the fact that the instructions from lab were different from this project and I was not able to test this on an actual board so I could not accurately test to see how the sound changed with different values.**

# Conclusion

In conclusion, this project required all the of the knowledge I had gained throughout the semester to make sure that the project worked as intended. I was constantly looking back at previous labs to see how something functioned so that I could successfully manipulate all the peripherals.

# Future Improvements

Some components of this project that could be improved upon in the future include setting up the username and password inside the program instead of having it hard coded. Another addition could be various other stats that the dinosaurs have like anger and happiness that require different things to make sure that the dinosaur is happy and healthy.

# Appendix I: User Manual

Common Instructions and Notes**:**

* **To visit a pen from main menu select a pen using 1,2, or 3.**
  + **Pen 1 is the T-Rex**
  + **Pen 2 is the Raptor**
  + **Pen 3 is the triceratops**
* **To call security press the IRQ when prompted**
* **To control the speed of the jeep adjust the potentiometer when the jeep is in motion**
* **To view any menu choices again press F or any key not specified during the menu screen**
* **After scrolling through all menu choices the only way to make a choice is to max the potentiometer.**
  + **This tells the system that you are done viewing the menu and are ready to make a choice**

Contents of each Menu Descriptions of major files**:**

### Main Menu

* + **On startup menu will ask for the passcode (1234)**
  + **After password is entered the user is allowed to scroll through the menu**
  + **Using the potentiometer to scroll the user can scroll freely until they max out the potentiometer.**
  + **Once the potentiometer is maxed out the program will read from the keypad and wait for an input.**
    - **1. Visit Pen1**
    - **2. Visit Pen2**
    - **3. Visit Pen3**
  + **If a different value is entered, then the main menu will read the potentiometer again before checking the menu.**
  + **This was designed so that if the user accidently maxed the potentiometer, they could go back review the choices before choosing something.**
  + **Entering 1-3 will change the travel flag and cause the program to jump to the traveling subroutine**

### Traveling

* + **This subroutine first checks to see if the switches corresponding to the parking brake are in the correct order.**
  + **The switches 0-6 must all be high before the jeep moves**
  + **As soon as the brake is off the motor will start and wait for the travel flag to be reset by the RTI**
  + **The RTI will see that main changed the travel flag to greater then 0 and will start a timer for 20 seconds that will force the traveling subroutine to loop until the timer is up and the travel flag is reset.**
  + **Once the flag is reset depending on the value entered in the main menu stored in a variable the pen corresponding to that input will be jumped to.**

Pen 1-3 **(all are slight variations of each other and have the same basic functions)**

* + **Gets that dinosaurs hunger value and displays it to the leds lighting up the number of leds corresponding to how full the dinosaur is.**
  + **Then the menu will be displayed**
    - **Name of dinosaur**
    - **Type of dinosaur**
    - **Diet**
    - **1. Feed**
    - **2. Hear Roar**
    - **3. Return to main gate**
  + **Similar to main menu this will require the user to use the potentiometer to scroll through the menu before reading a keypad value after the potentiometer is maxed out.**
  + **If the user selects hear roar, then the roar subroutine for that dinosaur is called playing the sound then returning to this menu.**
  + **If they select feed, then the feed subroutine for that dinosaur is called before returning to this menu.**
  + **If they select return, then the travel function is called before returning to main.**
  + **Any other key will cause the menu to repeat so the user can view the options again.**

### Feed Menu

* + **This menu is the same for all dinosaurs but the results of the food change depending on the dinosaur.**
    - **1. Lion (meat)**
    - **2. Mega-Salad (Vegetarian)**
    - **3. Cake**
  + **Carnivores like lion meat and their hunger level will go up when eating to indicate they enjoy the food.**
  + **Herbivores enjoy mega-salads and their hunger level will go up with this choice.**
  + **If the wrong food is chosen, then the hunger level stays the same and anger goes up**
  + **If cake is chosen then the dinosaur hunger level increases and anger level decreases no matter the dinosaur.**

### Emergency Menu

* + **This will only appear if a dinosaur breaks out due to hunger or anger**
  + **That dinosaur will roar right before this menu appears**
  + **First the lcd will say a dinosaur has escaped**
  + **Then the user has to reenter the passcode (1234)**
  + **After that the user is prompted to call security and this will wait for the user to push the IRQ**
  + **If IRQ pushed outside of emergency menu then it will not do anything.**

### RTI

* + **Counters**
    - **Travel counter that will count for 20 seconds in traveling menu then reset**
    - **If the emergency flag is on then emergency counter will count to 35 seconds before park is destroyed**
    - **Dino count that will count to 30 seconds**
      * **T-rex gets hungry every 20 seconds**
      * **Raptor gets hungry every 25 seconds**
      * **Triceratops gets hungry every 30 seconds**
  + **Flags**
    - **Travel flag gets turned on in main or pen when leaving main or leaving pen**
      * **If flag is 0 then not traveling**
      * **1 means going to pen 1**
      * **2 means going to pen 2**
      * **3 means going to pen 3**
      * **4 means returning to main**
    - **Emergency Flag**
      * **This gets turned on if one of the dinos gets hungry or angry**
      * **If emergency flag is 1 then the stopwatch will also tick at every half second as apposed to every second**
      * **This flag also changes the IRQ from doing nothing to changing the button val variable to 1 telling the emergency menu the button was pressed only when the emergency menu was on**

# Appendix II: Code

**(**This includes the c files provided in lab and they will be marked with a \* in the title)

(The description of each file is summarized above and heavily commented below)

## Main.asm

**XDEF Entry, \_Startup, main, keyval,travel,disp,emergency\_flag,RTIFLG,IRQCR**

**; we use export 'Entry' as symbol. This allows us to**

**; reference 'Entry' either in the linker .prm file**

**; or from C/C++ later on**

**XREF \_\_SEG\_END\_SSTACK,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,traveling,second,seq\_count,travel\_count,dino\_count,hunger1,hunger2,hunger3,anger1,anger2,anger3,lockdown,RTI\_ISR ; symbol defined by the linker for the end of the stack**

**; variable/data section**

**MY\_EXTENDED\_RAM: SECTION**

**; Insert here your data definition.**

**keyval: ds.b 1**

**disp: ds.b 33**

**travel: ds.b 1**

**emergency\_flag ds.b 1**

**MY\_CONTSANT: SECTION**

**RTIENA EQU $0038 ;location to initialize the interrupt enable regiter**

**RTICNT EQU $003B ;RTI interval control register location**

**RTIFLG EQU $0037 ;RTI flag register**

**IRQCR EQU $001E ;IRQ**

**; code section**

**MyCode: SECTION**

**main:**

**\_Startup:**

**Entry:**

**LDS #\_\_SEG\_END\_SSTACK ; initialize the stack pointer**

**MOVB #$80, RTIENA ; turn on bit 7 of RTIENA to enable RTI**

**MOVB #$40, RTICNT ; set the RTI interval to 1ms**

**movb #$C0, IRQCR**

**movb $0,emergency\_flag**

**movb $0,travel**

**movw #$0,second**

**movw #$0,seq\_count**

**ldab #0**

**MOVW #$0, travel\_count ;initialize count to 0**

**MOVW #$0, dino\_count ;initialize count to 0\**

**movb #$8,hunger1**

**movb #$8,hunger2 ;each dino loses hunger value at differnt intervals**

**movb #$8,hunger3**

**movb #$0,anger1**

**movb #$0,anger2**

**movb #$0,anger3**

**movw #$0,lockdown**

**CLI ; enable interrupts**

**jsr init\_LCD**

**movb #0, keyval**

**movb #' ',disp**

**movb #'E',disp+1 ;asks for password**

**movb #'n',disp+2**

**movb #'t',disp+3**

**movb #'e',disp+4**

**movb #'R',disp+5**

**movb #' ',disp+6**

**movb #'P',disp+7**

**movb #'a',disp+8**

**movb #'s',disp+9**

**movb #'s',disp+10**

**movb #'c',disp+11**

**movb #'o',disp+12**

**movb #'d',disp+13**

**movb #'e',disp+14**

**movb #':',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #':',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**pass1: jsr display\_string**

**jsr KeyPad ;gets a keypad value and if it matches checks next key**

**ldaa keyval**

**cmpa #1 ;if the correct value is entered then displays a X in the first slot and gets the second value**

**bne pass1**

**movb #'X',disp+17**

**ldd #disp**

**jsr display\_string**

**pass2: jsr KeyPad**

**jsr display\_string**

**ldaa keyval ;gets second key and will return here until second key is pressed**

**cmpa #2**

**bne pass2**

**movb #'X',disp+18**

**ldd #disp**

**jsr display\_string**

**pass3: jsr KeyPad**

**jsr display\_string**

**ldaa keyval**

**cmpa #3**

**bne pass3**

**movb #'X',disp+19**

**ldd #disp**

**jsr display\_string**

**pass4: jsr KeyPad**

**jsr display\_string**

**ldaa keyval**

**cmpa #4**

**bne pass4**

**movb #' ',disp**

**movb #'W',disp+1**

**movb #'e',disp+2 ;Sucessful login screen**

**movb #'l',disp+3**

**movb #'c',disp+4**

**movb #'o',disp+5**

**movb #'m',disp+6**

**movb #'e',disp+7**

**movb #' ',disp+8**

**movb #'b',disp+9**

**movb #'a',disp+10**

**movb #'c',disp+11**

**movb #'k',disp+12**

**movb #' ',disp+13**

**movb #'D',disp+14**

**movb #'r',disp+15**

**movb #'.',disp+16**

**movb #' ',disp+17**

**movb #'M',disp+18**

**movb #'a',disp+19**

**movb #'l',disp+20**

**movb #'c',disp+21**

**movb #'o',disp+22**

**movb #'l',disp+23**

**movb #'m',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**MainMenu: jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input**

**ldab pot\_value ;if lower third then display first option**

**cmpb #$33**

**lbge option2**

**movb #'V',disp**

**movb #'i',disp+1**

**movb #'s',disp+2**

**movb #'i',disp+3**

**movb #'t',disp+4**

**movb #' ',disp+5**

**movb #'P',disp+6**

**movb #'e',disp+7**

**movb #'n',disp+8**

**movb #'1',disp+9**

**movb #' ',disp+10**

**movb #' ',disp+11**

**movb #' ',disp+12**

**movb #' ',disp+13**

**movb #' ',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra MainMenu**

**option2: cmpb #$66 ;if middle third display second option**

**bge option3**

**movb #'2',disp+9**

**ldd #disp**

**jsr display\_string**

**lbra MainMenu**

**option3: cmpb #$99 ;if higher third then display last option**

**bge option4 ;if the highest value then and only then get a keypad value**

**;this makes sure the strings are all printed and that they are printed for enough time**

**movb #'3',disp+9**

**ldd #disp**

**jsr display\_string**

**lbra MainMenu**

**option4: jsr KeyPad ;gets a keypad value and makes sure it is a valid entry**

**ldaa keyval ;if valid sets that flag and if not valid rechecks the menu**

**cmpa #1**

**bne check2**

**movb #1,travel**

**jsr traveling**

**check2: cmpa #2**

**bne check3**

**movb #2,travel**

**jsr traveling**

**check3: cmpa #3**

**bne check4**

**movb #3,travel**

**jsr traveling**

**check4: lbra MainMenu**

## Keypad.asm

**XDEF KeyPad**

**XREF Delay2, keyval**

**MY\_VAR: SECTION**

**value ds.b 1**

**MY\_CONSTANT: SECTION**

**port\_u equ $268**

**ddr\_u equ $26A**

**psr\_u equ $26D**

**per\_u equ $26C**

**lut: dc.b $eb,$77,$7b,$7d,$b7,$bb,$bd,$d7,$db,$dd,$e7,$ed,$7e,$be,$de,$ee**

**seq: dc.b $70,$b0,$d0,$e0**

**KeyPad: pshx**

**pshy**

**pshd**

**movb #$F0, ddr\_u ;set ddr for u**

**movb #$F0, psr\_u ;set polarity select reg**

**movb #$0F, per\_u ;set polarity enable reg**

**st: ldx #seq**

**Scan: cpx #seq+4**

**beq st**

**ldaa 1,x+**

**staa port\_u**

**jsr Delay2**

**ldaa port\_u**

**staa value**

**anda #$0F**

**cmpa #$0F**

**beq Scan ;if not presssed go back to main routine**

**ldab #0**

**ldy #lut**

**lutloop: ldaa 1,y+**

**cmpa value**

**beq store ;if something is pressed store value of b on stack and then return to main**

**incb**

**bra lutloop**

**store: stab keyval**

**pwm: puld**

**puly**

**pulx**

**rts**

**Datapage.c \***

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**FILE : datapage.c**

**PURPOSE : paged data access runtime routines**

**MACHINE : Freescale 68HC12 (Target)**

**LANGUAGE : ANSI-C**

**HISTORY : 21.7.96 first version created**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include "hidef.h"**

**#include "non\_bank.sgm"**

**#include "runtime.sgm"**

**#ifndef \_\_HCS12X\_\_ /\* it's different for the HCS12X. See the text below at the #else // \_\_HCS12X\_\_ \*/**

**/\***

**According to the -Cp option of the compiler the**

**\_\_DPAGE\_\_, \_\_PPAGE\_\_ and \_\_EPAGE\_\_ macros are defined.**

**If none of them is given as argument, then no page accesses should occur and**

**this runtime routine should not be used !**

**To be on the save side, the runtime routines are created anyway.**

**If some of the -Cp options are given an adapted versions which only covers the**

**needed cases is produced.**

**\*/**

**/\* if no compiler option -Cp is given, it is assumed that all possible are given : \*/**

**/\* Compile with option -DHCS12 to activate this code \*/**

**#if defined(HCS12) || defined(\_HCS12) || defined(\_\_HCS12\_\_) /\* HCS12 family has PPAGE register only at 0x30 \*/**

**#define PPAGE\_ADDR (0x30+REGISTER\_BASE)**

**#ifndef \_\_PPAGE\_\_ /\* may be set already by option -CPPPAGE \*/**

**#define \_\_PPAGE\_\_**

**#endif**

**/\* Compile with option -DDG128 to activate this code \*/**

**#elif defined DG128 /\* HC912DG128 derivative has PPAGE register only at 0xFF \*/**

**#define PPAGE\_ADDR (0xFF+REGISTER\_BASE)**

**#ifndef \_\_PPAGE\_\_ /\* may be set already by option -CPPPAGE \*/**

**#define \_\_PPAGE\_\_**

**#endif**

**#elif defined(HC812A4)**

**/\* all setting default to A4 already \*/**

**#endif**

**#if !defined(\_\_EPAGE\_\_) && !defined(\_\_PPAGE\_\_) && !defined(\_\_DPAGE\_\_)**

**/\* as default use all page registers \*/**

**#define \_\_DPAGE\_\_**

**#define \_\_EPAGE\_\_**

**#define \_\_PPAGE\_\_**

**#endif**

**/\* modify the following defines to your memory configuration \*/**

**#define EPAGE\_LOW\_BOUND 0x400u**

**#define EPAGE\_HIGH\_BOUND 0x7ffu**

**#define DPAGE\_LOW\_BOUND 0x7000u**

**#define DPAGE\_HIGH\_BOUND 0x7fffu**

**#define PPAGE\_LOW\_BOUND (DPAGE\_HIGH\_BOUND+1)**

**#define PPAGE\_HIGH\_BOUND 0xBFFFu**

**#define REGISTER\_BASE 0x0u**

**#ifndef DPAGE\_ADDR**

**#define DPAGE\_ADDR (0x34u+REGISTER\_BASE)**

**#endif**

**#ifndef EPAGE\_ADDR**

**#define EPAGE\_ADDR (0x36u+REGISTER\_BASE)**

**#endif**

**#ifndef PPAGE\_ADDR**

**#define PPAGE\_ADDR (0x35u+REGISTER\_BASE)**

**#endif**

**/\***

**The following parts about the defines are assumed in the code of \_GET\_PAGE\_REG :**

**- the memory region controlled by DPAGE is above the area controlled by the EPAGE and**

**below the area controlled by the PPAGE.**

**- the lower bound of the PPAGE area is equal to be the higher bound of the DPAGE area + 1**

**\*/**

**#if EPAGE\_LOW\_BOUND >= EPAGE\_HIGH\_BOUND || EPAGE\_HIGH\_BOUND >= DPAGE\_LOW\_BOUND || DPAGE\_LOW\_BOUND >= DPAGE\_HIGH\_BOUND || DPAGE\_HIGH\_BOUND >= PPAGE\_LOW\_BOUND || PPAGE\_LOW\_BOUND >= PPAGE\_HIGH\_BOUND**

**#error /\* please adapt \_GET\_PAGE\_REG for this non default page configuration \*/**

**#endif**

**#if DPAGE\_HIGH\_BOUND+1 != PPAGE\_LOW\_BOUND**

**#error /\* please adapt \_GET\_PAGE\_REG for this non default page configuration \*/**

**#endif**

**/\* this module does either control if any access is in the bounds of the specified page or \*/**

**/\* ,if only one page is specified, just use this page. \*/**

**/\* This behavior is controlled by the define USE\_SEVERAL\_PAGES. \*/**

**/\* If !USE\_SEVERAL\_PAGES does increase the performance significantly \*/**

**/\* NOTE : When !USE\_SEVERAL\_PAGES, the page is also set for accesses outside of the area controlled \*/**

**/\* by this single page. But this is should not cause problems because the page is restored to the old value before any other access could occur \*/**

**#if !defined(\_\_DPAGE\_\_) && !defined(\_\_EPAGE\_\_) && !defined(\_\_PPAGE\_\_)**

**/\* no page at all is specified \*/**

**/\* only specifying the right pages will speed up these functions a lot \*/**

**#define USE\_SEVERAL\_PAGES 1**

**#elif defined(\_\_DPAGE\_\_) && defined(\_\_EPAGE\_\_) || defined(\_\_DPAGE\_\_) && defined(\_\_PPAGE\_\_) || defined(\_\_EPAGE\_\_) && defined(\_\_PPAGE\_\_)**

**/\* more than one page register is used \*/**

**#define USE\_SEVERAL\_PAGES 1**

**#else**

**#define USE\_SEVERAL\_PAGES 0**

**#if defined(\_\_DPAGE\_\_) /\* check which pages are used \*/**

**#define PAGE\_ADDR PPAGE\_ADDR**

**#elif defined(\_\_EPAGE\_\_)**

**#define PAGE\_ADDR EPAGE\_ADDR**

**#elif defined(\_\_PPAGE\_\_)**

**#define PAGE\_ADDR PPAGE\_ADDR**

**#else /\* we do not know which page, decide it at runtime \*/**

**#error /\* must not happen \*/**

**#endif**

**#endif**

**#if USE\_SEVERAL\_PAGES /\* only needed for several pages support \*/**

**/\*--------------------------- \_GET\_PAGE\_REG --------------------------------**

**Runtime routine to detect the right register depending on the 16 bit offset part**

**of an address.**

**This function is only used by the functions below.**

**Depending on the compiler options -Cp different versions of \_GET\_PAGE\_REG are produced.**

**Arguments :**

**- Y : offset part of an address**

**Result :**

**if address Y is controlled by a page register :**

**- X : address of page register if Y is controlled by an page register**

**- Zero flag cleared**

**- all other registers remain unchanged**

**if address Y is not controlled by a page register :**

**- Zero flag is set**

**- all registers remain unchanged**

**--------------------------- \_GET\_PAGE\_REG ----------------------------------\*/**

**#if defined(\_\_DPAGE\_\_)**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**static void NEAR \_GET\_PAGE\_REG(void) { /\*lint -esym(528, \_GET\_PAGE\_REG) used in asm code \*/**

**\_\_asm {**

**L\_DPAGE:**

**CPY #DPAGE\_LOW\_BOUND ;// test of lower bound of DPAGE**

**#if defined(\_\_EPAGE\_\_)**

**BLO L\_EPAGE ;// EPAGE accesses are possible**

**#else**

**BLO L\_NOPAGE ;// no paged memory below accesses**

**#endif**

**CPY #DPAGE\_HIGH\_BOUND ;// test of higher bound DPAGE/lower bound PPAGE**

**#if defined(\_\_PPAGE\_\_)**

**BHI L\_PPAGE ;// EPAGE accesses are possible**

**#else**

**BHI L\_NOPAGE ;// no paged memory above accesses**

**#endif**

**FOUND\_DPAGE:**

**LDX #DPAGE\_ADDR ;// load page register address and clear zero flag**

**RTS**

**#if defined(\_\_PPAGE\_\_)**

**L\_PPAGE:**

**CPY #PPAGE\_HIGH\_BOUND ;// test of higher bound of PPAGE**

**BHI L\_NOPAGE**

**FOUND\_PPAGE:**

**LDX #PPAGE\_ADDR ;// load page register address and clear zero flag**

**RTS**

**#endif**

**#if defined(\_\_EPAGE\_\_)**

**L\_EPAGE:**

**CPY #EPAGE\_LOW\_BOUND ;// test of lower bound of EPAGE**

**BLO L\_NOPAGE**

**CPY #EPAGE\_HIGH\_BOUND ;// test of higher bound of EPAGE**

**BHI L\_NOPAGE**

**FOUND\_EPAGE:**

**LDX #EPAGE\_ADDR ;// load page register address and clear zero flag**

**RTS**

**#endif**

**L\_NOPAGE:**

**ORCC #0x04 ;// sets zero flag**

**RTS**

**}**

**}**

**#else /\* !defined(\_\_DPAGE\_\_) \*/**

**#if defined( \_\_PPAGE\_\_ )**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**static void NEAR \_GET\_PAGE\_REG(void) { /\*lint -esym(528, \_GET\_PAGE\_REG) used in asm code \*/**

**\_\_asm {**

**L\_PPAGE:**

**CPY #PPAGE\_LOW\_BOUND ;// test of lower bound of PPAGE**

**#if defined( \_\_EPAGE\_\_ )**

**BLO L\_EPAGE**

**#else**

**BLO L\_NOPAGE ;// no paged memory below**

**#endif**

**CPY #PPAGE\_HIGH\_BOUND ;// test of higher bound PPAGE**

**BHI L\_NOPAGE**

**FOUND\_PPAGE:**

**LDX #PPAGE\_ADDR ;// load page register address and clear zero flag**

**RTS**

**#if defined( \_\_EPAGE\_\_ )**

**L\_EPAGE:**

**CPY #EPAGE\_LOW\_BOUND ;// test of lower bound of EPAGE**

**BLO L\_NOPAGE**

**CPY #EPAGE\_HIGH\_BOUND ;// test of higher bound of EPAGE**

**BHI L\_NOPAGE**

**FOUND\_EPAGE:**

**LDX #EPAGE\_ADDR ;// load page register address and clear zero flag**

**RTS**

**#endif**

**L\_NOPAGE: ;// not in any allowed page area**

**;// its a far access to a non paged variable**

**ORCC #0x04 ;// sets zero flag**

**RTS**

**}**

**}**

**#else /\* !defined(\_\_DPAGE\_\_ ) && !defined( \_\_PPAGE\_\_) \*/**

**#if defined(\_\_EPAGE\_\_)**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**static void NEAR \_GET\_PAGE\_REG(void) { /\*lint -esym(528, \_GET\_PAGE\_REG) used in asm code \*/**

**\_\_asm {**

**L\_EPAGE:**

**CPY #EPAGE\_LOW\_BOUND ;// test of lower bound of EPAGE**

**BLO L\_NOPAGE**

**CPY #EPAGE\_HIGH\_BOUND ;// test of higher bound of EPAGE**

**BHI L\_NOPAGE**

**FOUND\_EPAGE:**

**LDX #EPAGE\_ADDR ;// load page register address and clear zero flag**

**RTS**

**L\_NOPAGE: ;// not in any allowed page area**

**;// its a far access to a non paged variable**

**ORCC #0x04 ;// sets zero flag**

**RTS**

**}**

**}**

**#endif /\* defined(\_\_EPAGE\_\_) \*/**

**#endif /\* defined(\_\_PPAGE\_\_) \*/**

**#endif /\* defined(\_\_DPAGE\_\_) \*/**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**/\*--------------------------- \_SET\_PAGE --------------------------------**

**Runtime routine to set the right page register. This routine is used if the compiler**

**does not know the right page register, i.e. if the option -Cp is used for more than**

**one page register or if the runtime option is used for one of the -Cp options.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**Result :**

**- page part written into the correct page register.**

**- the old page register content is destroyed**

**- all processor registers remains unchanged**

**--------------------------- \_SET\_PAGE ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_SET\_PAGE(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**STAB 0,X ;// set page register**

**L\_NOPAGE:**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**STAB PAGE\_ADDR ;// set page register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_LOAD\_FAR\_8 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**Result :**

**- value to be read in the B register**

**- all other registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_LOAD\_FAR\_8 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_LOAD\_FAR\_8(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHA ;// save A register**

**LDAA 0,X ;// save page register**

**STAB 0,X ;// set page register**

**LDAB 0,Y ;// actual load, overwrites page**

**STAA 0,X ;// restore page register**

**PULA ;// restore A register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**LDAB 0,Y ;// actual load, overwrites page**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHA ;// save A register**

**LDAA PAGE\_ADDR ;// save page register**

**STAB PAGE\_ADDR ;// set page register**

**LDAB 0,Y ;// actual load, overwrites page**

**STAA PAGE\_ADDR ;// restore page register**

**PULA ;// restore A register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_LOAD\_FAR\_16 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**Result :**

**- value to be read in the Y register**

**- all other registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_LOAD\_FAR\_16 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_LOAD\_FAR\_16(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHA ;// save A register**

**LDAA 0,X ;// save page register**

**STAB 0,X ;// set page register**

**LDY 0,Y ;// actual load, overwrites address**

**STAA 0,X ;// restore page register**

**PULA ;// restore A register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**LDY 0,Y ;// actual load, overwrites address**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHA ;// save A register**

**LDAA PAGE\_ADDR ;// save page register**

**STAB PAGE\_ADDR ;// set page register**

**LDY 0,Y ;// actual load, overwrites address**

**STAA PAGE\_ADDR ;// restore page register**

**PULA ;// restore A register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_LOAD\_FAR\_24 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**Result :**

**- value to be read in the Y:B registers**

**- all other registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_LOAD\_FAR\_24 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_LOAD\_FAR\_24(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHA ;// save A register**

**LDAA 0,X ;// save page register**

**STAB 0,X ;// set page register**

**LDAB 0,Y ;// actual load, overwrites page of address**

**LDY 1,Y ;// actual load, overwrites offset of address**

**STAA 0,X ;// restore page register**

**PULA ;// restore A register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**LDAB 0,Y ;// actual load, overwrites page of address**

**LDY 1,Y ;// actual load, overwrites offset of address**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHA ;// save A register**

**LDAA PAGE\_ADDR ;// save page register**

**STAB PAGE\_ADDR ;// set page register**

**LDAB 0,Y ;// actual load, overwrites page of address**

**LDY 1,Y ;// actual load, overwrites offset of address**

**STAA PAGE\_ADDR ;// restore page register**

**PULA ;// restore A register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_LOAD\_FAR\_32 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**Result :**

**- low 16 bit of value to be read in the D registers**

**- high 16 bit of value to be read in the Y registers**

**- all other registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_LOAD\_FAR\_32 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_LOAD\_FAR\_32(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**LDAA 0,X ;// save page register**

**PSHA ;// put it onto the stack**

**STAB 0,X ;// set page register**

**LDD 2,Y ;// actual load, low word**

**LDY 0,Y ;// actual load, high word**

**MOVB 1,SP+,0,X ;// restore page register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**LDD 2,Y ;// actual load, low word**

**LDY 0,Y ;// actual load, high word**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**LDAA PAGE\_ADDR ;// save page register**

**PSHA ;// put it onto the stack**

**STAB PAGE\_ADDR ;// set page register**

**LDD 2,Y ;// actual load, low word**

**LDY 0,Y ;// actual load, high word**

**MOVB 1,SP+,PAGE\_ADDR ;// restore page register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_STORE\_FAR\_8 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**- value to be stored in the B register**

**Result :**

**- value stored at the address**

**- all registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_STORE\_FAR\_8 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_STORE\_FAR\_8(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHB ;// save B register**

**LDAB 0,X ;// save page register**

**MOVB 0,SP, 0,X ;// set page register**

**STAA 0,Y ;// store the value passed in A**

**STAB 0,X ;// restore page register**

**PULB ;// restore B register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**STAA 0,Y ;// store the value passed in A**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHB ;// save A register**

**LDAB PAGE\_ADDR ;// save page register**

**MOVB 0,SP,PAGE\_ADDR ;// set page register**

**STAA 0,Y ;// store the value passed in A**

**STAB PAGE\_ADDR ;// restore page register**

**PULB ;// restore B register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_STORE\_FAR\_16 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**- value to be stored in the X register**

**Result :**

**- value stored at the address**

**- all registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_STORE\_FAR\_16 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_STORE\_FAR\_16(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHA**

**LDAA 0,X ;// save page register**

**STAB 0,X ;// set page register**

**MOVW 1,SP,0,Y ;// store the value passed in X**

**STAA 0,X ;// restore page register**

**PULA ;// restore A register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**STX 0,Y ;// store the value passed in X**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHA ;// save A register**

**LDAA PAGE\_ADDR ;// save page register**

**STAB PAGE\_ADDR ;// set page register**

**STX 0,Y ;// store the value passed in X**

**STAA PAGE\_ADDR ;// restore page register**

**PULA ;// restore A register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_STORE\_FAR\_24 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address in the B register**

**- value to be stored in the X:A registers (X : low 16 bit, A : high 8 bit)**

**Result :**

**- value stored at the address**

**- all registers remains unchanged**

**- all page register still contain the same value**

**--------------------------- \_STORE\_FAR\_24 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_STORE\_FAR\_24(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHA**

**LDAA 0,X ;// save page register**

**STAB 0,X ;// set page register**

**MOVW 1,SP, 1,Y ;// store the value passed in X**

**MOVB 0,SP, 0,Y ;// store the value passed in A**

**STAA 0,X ;// restore page register**

**PULA ;// restore A register**

**PULX ;// restore X register**

**RTS**

**L\_NOPAGE:**

**STX 1,Y ;// store the value passed in X**

**STAA 0,Y ;// store the value passed in X**

**PULX ;// restore X register**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHA ;// save A register**

**LDAA PAGE\_ADDR ;// save page register**

**STAB PAGE\_ADDR ;// set page register**

**MOVB 0,SP, 0,Y ;// store the value passed in A**

**STX 1,Y ;// store the value passed in X**

**STAA PAGE\_ADDR ;// restore page register**

**PULA ;// restore A register**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_STORE\_FAR\_32 --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of an address in the Y register**

**- page part of an address is on the stack at 3,SP (just below the return address)**

**- value to be stored in the X:D registers (D : low 16 bit, X : high 16 bit)**

**Result :**

**- value stored at the address**

**- all registers remains unchanged**

**- the page part is removed from the stack**

**- all page register still contain the same value**

**--------------------------- \_STORE\_FAR\_32 ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_STORE\_FAR\_32(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**PSHX ;// save X register**

**\_\_PIC\_JSR(\_GET\_PAGE\_REG)**

**BEQ L\_NOPAGE**

**PSHD**

**LDAA 0,X ;// save page register**

**MOVB 6,SP, 0,X ;// set page register**

**MOVW 2,SP, 0,Y ;// store the value passed in X (high word)**

**MOVW 0,SP, 2,Y ;// store the value passed in D (low word)**

**STAA 0,X ;// restore page register**

**PULD ;// restore A register**

**BRA done**

**L\_NOPAGE:**

**MOVW 0,SP, 0,Y ;// store the value passed in X (high word)**

**STD 2,Y ;// store the value passed in D (low word)**

**done:**

**PULX ;// restore X register**

**MOVW 0,SP, 1,+SP ;// move return address**

**RTS**

**}**

**#else /\* USE\_SEVERAL\_PAGES \*/**

**\_\_asm {**

**PSHD ;// save D register**

**LDAA PAGE\_ADDR ;// save page register**

**LDAB 4,SP ;// load page part of address**

**STAB PAGE\_ADDR ;// set page register**

**STX 0,Y ;// store the value passed in X**

**MOVW 0,SP, 2,Y ;// store the value passed in D (low word)**

**STAA PAGE\_ADDR ;// restore page register**

**PULD ;// restore D register**

**MOVW 0,SP, 1,+SP ;// move return address**

**RTS**

**}**

**#endif /\* USE\_SEVERAL\_PAGES \*/**

**}**

**/\*--------------------------- \_FAR\_COPY\_RC --------------------------------**

**This runtime routine is used to access paged memory via a runtime function.**

**It may also be used if the compiler option -Cp is not used with the runtime argument.**

**Arguments :**

**- offset part of the source int the X register**

**- page part of the source in the A register**

**- offset part of the dest int the Y register**

**- page part of the dest in the B register**

**- number of bytes to be copied is defined by the next 2 bytes after the return address.**

**Result :**

**- memory area copied**

**- no registers are saved, i.e. all registers may be destroyed**

**- all page register still contain the same value as before the call**

**- the function returns after the constant defining the number of bytes to be copied**

**stack-structure at the loop-label:**

**0,SP : destination offset**

**2,SP : source page**

**3,SP : destination page**

**4,SP : source offset**

**6,SP : points to length to be copied. This function returns after the size**

**A usual call to this function looks like:**

**struct Huge src, dest;**

**; ...**

**LDX #src**

**LDAA #PAGE(src)**

**LDY #dest**

**LDAB #PAGE(dest)**

**JSR \_FAR\_COPY\_RC**

**DC.W sizeof(struct Huge)**

**; ...**

**--------------------------- \_FAR\_COPY\_RC ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_RC(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**DEX ;// source addr-=1, because loop counter ends at 1**

**PSHX ;// save source offset**

**PSHD ;// save both pages**

**DEY ;// destination addr-=1, because loop counter ends at 1**

**PSHY ;// save destination offset**

**LDY 6,SP ;// Load Return address**

**LDX 2,Y+ ;// Load Size to copy**

**STY 6,SP ;// Store adjusted return address**

**loop:**

**LDD 4,SP ;// load source offset**

**LEAY D,X ;// calculate actual source address**

**LDAB 2,SP ;// load source page**

**\_\_PIC\_JSR(\_LOAD\_FAR\_8) ;// load 1 source byte**

**PSHB ;// save value**

**LDD 0+1,SP ;// load destination offset**

**LEAY D,X ;// calculate actual destination address**

**PULA ;// restore value**

**LDAB 3,SP ;// load destination page**

**\_\_PIC\_JSR(\_STORE\_FAR\_8) ;// store one byte**

**DEX**

**BNE loop**

**LEAS 6,SP ;// release stack**

**\_SRET ;// debug info only: This is the last instr of a function with a special return**

**RTS ;// return**

**}**

**#else**

**\_\_asm {**

**PSHD ;// store page registers**

**TFR X,D**

**PSHY ;// temporary space**

**LDY 4,SP ;// load return address**

**ADDD 2,Y+ ;// calculate source end address. Increment return address**

**STY 4,SP**

**PULY**

**PSHD ;// store src end address**

**LDAB 2,SP ;// reload source page**

**LDAA PAGE\_ADDR ;// save page register**

**PSHA**

**loop:**

**STAB PAGE\_ADDR ;// set source page**

**LDAA 1,X+ ;// load value**

**MOVB 4,SP, PAGE\_ADDR ;// set destination page**

**STAA 1,Y+**

**CPX 1,SP**

**BNE loop**

**LDAA 5,SP+ ;// restore old page value and release stack**

**STAA PAGE\_ADDR ;// store it into page register**

**\_SRET ;// debug info only: This is the last instr of a function with a special return**

**RTS**

**}**

**#endif**

**}**

**/\*--------------------------- \_FAR\_COPY --------------------------------**

**The \_FAR\_COPY runtime routine was used to copied large memory blocks in previous compiler releases.**

**However this release now does use \_FAR\_COPY\_RC instead. The only difference is how the size of**

**the area to be copied is passed into the function. For \_FAR\_COPY the size is passed on the stack just**

**above the return address. \_FAR\_COPY\_RC does expect the return address just after the JSR \_FAR\_COPY\_RC call**

**in the code of the caller. This allows for denser code calling \_FAR\_COPY\_RC but does also need a slightly**

**larger runtime routine and it is slightly slower.**

**The \_FAR\_COPY routine is here now mainly for compatibility with previous releases.**

**The current compiler does not use it.**

**--------------------------- \_FAR\_COPY ----------------------------------\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY(void) {**

**#if USE\_SEVERAL\_PAGES**

**\_\_asm {**

**DEX ;// source addr-=1, because loop counter ends at 1**

**PSHX ;// save source offset**

**PSHD ;// save both pages**

**DEY ;// destination addr-=1, because loop counter ends at 1**

**PSHY ;// save destination offset**

**LDX 8,SP ;// load counter, assuming counter > 0**

**loop:**

**LDD 4,SP ;// load source offset**

**LEAY D,X ;// calculate actual source address**

**LDAB 2,SP ;// load source page**

**\_\_PIC\_JSR(\_LOAD\_FAR\_8) ;// load 1 source byte**

**PSHB ;// save value**

**LDD 0+1,SP ;// load destination offset**

**LEAY D,X ;// calculate actual destination address**

**PULA ;// restore value**

**LDAB 3,SP ;// load destination page**

**\_\_PIC\_JSR(\_STORE\_FAR\_8) ;// store one byte**

**DEX**

**BNE loop**

**LDX 6,SP ;// load return address**

**LEAS 10,SP ;// release stack**

**JMP 0,X ;// return**

**}**

**#else**

**\_\_asm {**

**PSHD ;// store page registers**

**TFR X,D**

**ADDD 4,SP ;// calculate source end address**

**STD 4,SP**

**PULB ;// reload source page**

**LDAA PAGE\_ADDR ;// save page register**

**PSHA**

**loop:**

**STAB PAGE\_ADDR ;// set source page**

**LDAA 1,X+ ;// load value**

**MOVB 1,SP, PAGE\_ADDR ;// set destination page**

**STAA 1,Y+**

**CPX 4,SP**

**BNE loop**

**LDAA 2,SP+ ;// restore old page value and release stack**

**STAA PAGE\_ADDR ;// store it into page register**

**LDX 4,SP+ ;// release stack and load return address**

**JMP 0,X ;// return**

**}**

**#endif**

**}**

**#else /\* \_\_HCS12X\_\_ \*/**

**/\***

**The HCS12X knows two different kind of addresses:**

**- Logical addresses. E.g.**

**MOVB #page(var),RPAGE**

**INC var**

**- Global addresses E.g.**

**MOVB #page(var),GPAGE**

**GLDAA var**

**INCA**

**GSTAA var**

**Global addresses are used with G-Load's and G-Store's, logical addresses are used for all the other instructions**

**and occasions. As HC12's or HCS12's do not have the G-Load and G-Store instructions,**

**global addresses are not used with these processor families.**

**They are only used with HCS12X chips (and maybe future ones deriving from a HCS12X).**

**Logical and Global addresses can point to the same object, however the global and logical address of an object**

**are different for most objects (actually for all except the registers from 0 to 0x7FF).**

**Therefore the compiler needs to transform in between them.**

**HCS12X Pointer types:**

**The following are logical addresses:**

**- all 16 bit pointers**

**- "char\* \_\_near": always.**

**- "char \*" in the small and banked memory model**

**- 24 bit dpage, epage, ppage or rpage pointers (\*1) (note: the first HCS12X compilers may not support these pointer types)**

**- "char \*\_\_dpage": Note this type only exists for**

**orthogonality with the HC12 A4 chip which has a DPAGE reg.**

**It does not apply to the HCS12X.**

**- "char \*\_\_epage": 24 bit pointer using the EPAGE register**

**- "char \*\_\_ppage": 24 bit pointer using the PPAGE register.**

**As the PPAGE is also used for BANKED code,**

**using this pointer type is only legal from non banked code.**

**- "char \*\_\_rpage": 24 bit pointer using the RPAGE register**

**The following are global addresses:**

**"char\*": in the large memory model (only HCS12X)**

**"char\* \_\_far": always for HCS12X.**

**(\*1): For the HC12 and HCS12 "char\* \_\_far" and "char\*" in the large memory model are also logical.**

**Some notes for the HC12/HCS12 programmers.**

**The address of a far object for a HC12 and for a HCS12X is different, even if they are at the same place in the memory map.**

**For the HC12, a far address is using the logical addresses, for the HCS12X however, far addresses are using global addresses.**

**This does cause troubles for the unaware!**

**The conversion routines implemented in this file support the special HCS12XE RAM mapping (when RAMHM is set).**

**To enable this mapping compile this file with the "-MapRAM" compiler option.**

**HCS12X Logical Memory map**

**Logical Addresses Used for shadowed at page register Global Address**

**0x000000 .. 0x0007FF Peripheral Registers Not Paged 0x000000**

**0x??0800 .. 0x??0BFF Paged EEPROM EPAGE (@0x17) 0x100000+EPAGE\*0x0400**

**0x000C00 .. 0x000FFF Non Paged EEPROM 0xFF0800..0xFF0FFF Not Paged 0x13FC00**

**0x??1000 .. 0x??1FFF Paged RAM RPAGE (@0x16) 0x000000+RPAGE\*0x1000**

**0x002000 .. 0x003FFF Non Paged RAM 0xFE1000..0xFF1FFF Not Paged 0x0FE000**

**0x004000 .. 0x007FFF Non Paged FLASH 0xFC8000..0xFCBFFF Not Paged 0x7F4000**

**0x??8000 .. 0x00BFFF Paged FLASH PPAGE (@0x30) 0x400000+PPAGE\*0x4000**

**0x00C000 .. 0x00FFFF Non Paged FLASH 0xFF8000..0xFFBFFF Not Paged 0x7FC000**

**NA: Not Applicable**

**HCS12X Global Memory map**

**Global Addresses Used for Logical mapped at**

**0x000000 .. 0x0007FF Peripheral Registers 0x000000 .. 0x0007FF**

**0x000800 .. 0x000FFF DMA registers Not mapped**

**0x001000 .. 0x0FFFFF RAM 0x??1000 .. 0x??1FFF**

**0x0FE000 .. 0x0FFFFF RAM, Log non paged 0x002000 .. 0x003FFF**

**0x100000 .. 0x13FFFF EEPROM 0x??0800 .. 0x??0BFF**

**0x13FC00 .. 0x13FFFF EEPROM non paged 0x000C00 .. 0x000FFF**

**0x140000 .. 0x3FFFFF External Space Not mapped**

**0x400000 .. 0x7FFFFF FLASH 0x??8000 .. 0x??BFFF**

**0x7F4000 .. 0x7F7FFF FLASH, Log non paged 0x004000 .. 0x007FFF**

**0x7FC000 .. 0x7FFFFF FLASH, Log non paged 0x00C000 .. 0x00FFFF**

**HCS12XE Logical Memory map (with RAMHM set)**

**Logical Addresses Used for shadowed at page register Global Address**

**0x000000 .. 0x0007FF Peripheral Registers Not Paged 0x000000**

**0x??0800 .. 0x??0BFF Paged EEPROM EPAGE 0x100000+EPAGE\*0x0400**

**0x000C00 .. 0x000FFF Non Paged EEPROM 0xFF0800..0xFF0FFF Not Paged 0x13FC00**

**0x??1000 .. 0x??1FFF Paged RAM RPAGE 0x000000+RPAGE\*0x1000**

**0x002000 .. 0x003FFF Non Paged RAM 0xFA1000..0xFB1FFF Not Paged 0x0FA000**

**0x004000 .. 0x007FFF Non Paged RAM 0xFC1000..0xFF1FFF Not Paged 0x0FC000**

**0x??8000 .. 0x00BFFF Paged FLASH PPAGE 0x400000+PPAGE\*0x4000**

**0x00C000 .. 0x00FFFF Non Paged FLASH 0xFF8000..0xFFBFFF Not Paged 0x7FC000**

**NA: Not Applicable**

**HCS12X Global Memory map (with RAMHM set)**

**Global Addresses Used for Logical mapped at**

**0x000000 .. 0x0007FF Peripheral Registers 0x000000 .. 0x0007FF**

**0x000800 .. 0x000FFF DMA registers Not mapped**

**0x001000 .. 0x0FFFFF RAM 0x??1000 .. 0x??1FFF**

**0x0FA000 .. 0x0FFFFF RAM, Log non paged 0x002000 .. 0x007FFF**

**0x100000 .. 0x13FFFF EEPROM 0x??0800 .. 0x??0BFF**

**0x13FC00 .. 0x13FFFF EEPROM non paged 0x000C00 .. 0x000FFF**

**0x140000 .. 0x3FFFFF External Space Not mapped**

**0x400000 .. 0x7FFFFF FLASH 0x??8000 .. 0x??BFFF**

**0x7F4000 .. 0x7F7FFF FLASH, Log non paged Not mapped**

**0x7FC000 .. 0x7FFFFF FLASH, Log non paged 0x00C000 .. 0x00FFFF**

**How to read this table:**

**For logical addresses, the lower 16 bits of the address do determine in which area the address is,**

**if this address is paged, then this entry also controls and which of the EPAGE, PPAGE or RPAGE**

**page register is controlling the bits 16 to 23 of the address.**

**For global addresses, the bits 16 to 23 have to be in the GPAGE register and the lower 16 bits**

**have to be used with the special G load or store instructions (e.g. GLDAA).**

**As example the logical address 0x123456 is invalid. Because its lower bits 0x3456 are in a**

**non paged area, so the page 0x12 does not exist.**

**The address 0xFE1020 however does exist. To access it, the RPAGE has to contain 0xFE and the**

**offset 0x1020 has to be used.**

**ORG $7000**

**MOVB #0xFE, 0x16 ; RPAGE**

**LDAA 0x1020 ; reads at the logical address 0xFE1020**

**Because the last two RAM pages are also accessible directly from 0x2000 to 0x3FFF, the**

**following shorter code does read the same memory location:**

**ORG $7000**

**LDAA 0x2020 ; reads at the logical address 0x2020**

**; which maps to the same memory as 0xFE1020**

**This memory location also has a global address. For logical 0xFE1020 the global address is 0x0FE020.**

**So the following code does once more access the same memory location:**

**ORG $7000**

**MOVB #0x0F, 0x10 ; GPAGE**

**GLDAA 0xE020 ; reads at the global address 0x0FE020**

**; which maps to the same memory as the logical addr. 0xFE1020**

**Therefore every memory location for the HCS12X has up to 3 different addresses.**

**Up to two logical and one global.**

**Notes.**

**- Not every address has a logical equivalent. The external space is only available in the global address space.**

**- The PPAGE must only be set if the code is outside of the 0x8000 to 0xBFFF range.**

**If not, the next code fetch will be from the new wrong PPAGE value.**

**- Inside of the paged area, the highest pages are allocated first. So all HCS12X's do have the FF pages**

**(if they have this memory type at all).**

**- For RPAGE, the value 0 is illegal. Otherwise the global addresses would overlap with the registers.**

**\*/**

**#if \_\_OPTION\_ACTIVE\_\_("-MapRAM")**

**#define \_\_HCS12XE\_RAMHM\_SET\_\_**

**#endif**

**/\*--------------------------- pointer conversion operations -------------------------------\*/**

**/\*--------------------------- \_CONV\_GLOBAL\_TO\_LOGICAL --------------------------------**

**Convert 24 bit logical to 24 bit global pointer**

**("char\*\_\_far" to "char\*\_\_gpage")**

**Arguments :**

**- B : page part of global address**

**- X : 16 offset part of global address**

**Postcondition :**

**- B == page of returned logical address**

**- X == offset of returned logical address**

**- Y remains unchanged**

**- A remains unchanged**

**\*/**

**/\*--------------------------- Convert 24 bit global to 24 bit logical pointer ----------------------------------\*/**

**/\* B:X = Logical(B:X) \*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_FRAME**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**void NEAR \_CONV\_GLOBAL\_TO\_LOGICAL(void) {**

**\_\_asm {**

**CMPB #0x40 ;// flash (0x400000..0x7FFFFF) or not?**

**BLO Below400000**

**// from 0x400000 to 0x7FFFFF**

**CMPB #0x7F ;// check for Unpaged areas 0x7FC000..0x7FFFFF and 0x7F4000..0x7F7FFF**

**BNE PAGED\_FLASH\_AREA**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**BITX #0x4000**

**BEQ PAGED\_FLASH\_AREA**

**#else**

**CPX #0xC000**

**BLO PAGED\_FLASH\_AREA**

**#endif**

**// from 0x7F4000 to 0x7F7FFF or 0x7FC000 to 0x7FFFFF**

**;// Note: offset in X is already OK.**

**CLRB ;// logical page == 0**

**RTS**

**PAGED\_FLASH\_AREA: ;// paged flash. Map to 0x8000..0xBFFF**

**// from 0x400000 to 0x7F3FFF or 0x7F8000 to 0x7FBFFF**

**LSLX ; // shift 24 bit address 2 bits to the left to get correct page in B**

**ROLB**

**LSLX**

**ROLB**

**LSRX ; // shift back to get offset from 0x8000 to 0xBFFF**

**SEC**

**RORX**

**RTS ;// done**

**Below400000:**

**// from 0x000000 to 0x3FFFFF**

**#if 0 /\* How should we handle mapping to External Space. There is no logical equivalent. This is an error case! \*/**

**CMPB #0x14 ;// check if above 0x140000. If so, its in the external space**

**BLO Below140000**

**ERROR !!!! ;// this mapping is not possible! What should we do?**

**RTS**

**Below140000:**

**// from 0x000000 to 0x13FFFF**

**#endif**

**CMPB #0x10 ;// if >= 0x100000 it's EEPROM**

**BLO Below100000**

**// from 0x100000 to 0x13FFFF (or 0x3FFFFF)**

**CMPB #0x13 ;// check if its is in the non paged EEPROM area at 0x13FC00..0x13FFFF**

**BLO Below13FC00**

**CPX #0xFC00**

**BLO Below13FC00**

**// from 0x13FC00 to 0x13FFFF (or 0x3FFFFF)**

**LEAX 0x1000,X ;// same as SUBX #0xF000 // map from 0xFC00 to 0x0C00**

**CLRB**

**RTS**

**Below13FC00:**

**// from 0x100000 to 0x13FBFF**

**PSHA**

**TFR XH,A ;// calculate logical page**

**EXG A,B**

**LSRD**

**LSRD**

**PULA**

**ANDX #0x03FF**

**LEAX 0x0800,X ;// same as ORX #0x0800**

**RTS**

**Below100000:**

**// from 0x000000 to 0x0FFFFF**

**TSTB**

**BNE RAM\_AREA**

**CPX #0x1000**

**BLO Below001000**

**RAM\_AREA:**

**// from 0x001000 to 0x0FFFFF**

**CMPB #0x0F**

**BNE PagedRAM\_AREA**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**CPX #0xE000**

**BLO PagedRAM\_AREA**

**// from 0x0FE000 to 0x0FFFFF**

**SUBX #(0xE000-0x2000) ;// map 0xE000 to 0x2000**

**#else**

**CPX #0xA000**

**BLO PagedRAM\_AREA**

**// from 0x0FA000 to 0x0FFFFF**

**SUBX #(0xA000-0x2000) ;// map 0xA000 to 0x2000**

**#endif**

**CLRB ;// Page is 0**

**RTS**

**PagedRAM\_AREA:**

**// from 0x001000 to 0x0FDFFF**

**PSHA**

**TFR XH, A ;// calculate logical page**

**EXG A,B**

**LSRD**

**LSRD**

**LSRD**

**LSRD**

**PULA**

**ANDX #0x0FFF**

**LEAX 0x1000,X ;// same as ORX #0x1000**

**RTS**

**Below001000:**

**// from 0x000000 to 0x000FFF**

**#if 0**

**CMPA #0x08**

**BLO Below000800**

**// from 0x000800 to 0x000FFF**

**// ??? DMA Regs?**

**RTS**

**Below000800:**

**// from 0x000000 to 0x0007FF**

**#endif**

**CLRB**

**RTS**

**}**

**}**

**/\*--------------------------- \_CONV\_GLOBAL\_TO\_NEAR --------------------------------**

**Convert 24 bit global to 16 bit logical pointer**

**("char\*\_\_far" to "char\*")**

**Arguments :**

**- B : page part of global address**

**- X : 16 offset part of global address**

**Postcondition :**

**- B is undefined**

**- A remains unchanged**

**- X == offset of returned logical address**

**- Y remains unchanged**

**\*/**

**/\*--------------------------- Convert 24 bit global to 16 bit logical pointer ----------------------------------\*/**

**/\* X = Logical(B:X) \*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#define \_REUSE\_CONV\_GLOBAL\_TO\_LOGICAL 1**

**#pragma NO\_FRAME**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**void NEAR \_CONV\_GLOBAL\_TO\_NEAR(void){**

**#if \_REUSE\_CONV\_GLOBAL\_TO\_LOGICAL /\* do we want an optimized version? \*/**

**\_\_asm JMP \_CONV\_GLOBAL\_TO\_LOGICAL; /\* offset for NEAR is same as for LOGICAL. \*/**

**#else**

**\_\_asm {**

**CMPB #0x40 ;// flash (0x400000..0x7FFFFF) or not?**

**BLO Below400000**

**// from 0x400000 to 0x7FFFFF**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**CMPB #0x7F ;// check for Unpaged areas 0x7FC000..0x7FFFFF and 0x7F4000..0x7F7FFF**

**BNE PAGED\_FLASH\_AREA**

**CPX #0x4000**

**BLO PAGED\_FLASH\_AREA**

**// from 0x7F4000 to 0x7FFFFF**

**#else**

**CMPB #0x7F ;// check for Unpaged area 0x7FC000..0x7FFFFF**

**BNE PAGED\_FLASH\_AREA**

**CPX #0xC000**

**BLO PAGED\_FLASH\_AREA**

**// from 0x7FC000 to 0x7FFFFF**

**#endif**

**;// note non PAGED flash areas or paged area 0x7F8000..0x7FBFFF which are mapping all correctly**

**RTS**

**PAGED\_FLASH\_AREA: ;// paged flash. Map to 0x8000..0xBFFF**

**// from 0x400000 to 0x7F3FFF**

**ANDX #0x3F00 ;// cut to 0.. 0x3FFF**

**LEAX 0x8000,X ;// same as ORX #0x8000 ;// move to 0x8000..0xBFFF**

**RTS ;// done**

**Below400000:**

**// from 0x000000 to 0x3FFFFF**

**#if 0 /\* How should we handle mapping to External Space. There is no logical equivalent. This is an error case! \*/**

**CMPB #0x14 ;// check if above 0x140000. If so, its in the external space**

**BLO Below140000**

**ERROR !!!! ;// this mapping is not possible! What should we do?**

**RTS**

**Below140000:**

**// from 0x000000 to 0x13FFFF**

**#endif**

**CMPB #0x10 ;// if >= 0x100000 it's EEPROM**

**BLO Below100000**

**// from 0x100000 to 0x13FFFF (or 0x3FFFFF)**

**CMPB #0x13 ;// check if its is in the non paged EEPROM area at 0x13FC00..0x13FFFF**

**BNE Below13FC00**

**CPX #0xFC00**

**BLO Below13FC00**

**// from 0x13FC00 to 0x13FFFF (or 0x3FFFFF)**

**SUBX #0xF000 ;// map from 0xFC00 to 0x0C00**

**RTS**

**Below13FC00:**

**// from 0x100000 to 0x13FBFF**

**ANDX #0x03FF**

**LEAX 0x800,X ;// same as ORX #0x0800**

**RTS**

**Below100000:**

**// from 0x000000 to 0x0FFFFF**

**TBNE B,RAM\_AREA**

**CPX #0x1000**

**BLO Below001000**

**RAM\_AREA:**

**// from 0x001000 to 0x0FFFFF**

**CMPB #0x0F**

**BNE PagedRAM\_AREA**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**CPX #0xE000**

**BLO PagedRAM\_AREA**

**// from 0x0FE000 to 0x0FFFFF**

**SUBX #(0xE000-0x2000) ;// map 0xE000 to 0x2000**

**#else**

**CPX #0xA000**

**BLO PagedRAM\_AREA**

**// from 0x0FA000 to 0x0FFFFF**

**SUBX #(0xA000-0x2000) ;// map 0xA000 to 0x2000**

**#endif**

**RTS**

**PagedRAM\_AREA:**

**// from 0x001000 to 0x0FDFFF (0x001000 to 0x0F9FFF if HCS12XE RAM mapping is enabled)**

**ANDX #0x0FFF**

**LEAX 0x1000,X ;// same as ORX #0x1000**

**RTS**

**Below001000:**

**// from 0x000000 to 0x000FFF**

**RTS**

**}**

**#endif**

**}**

**/\*--------------------------- \_CONV\_NEAR\_TO\_GLOBAL --------------------------------**

**Convert 16 bit logical to 24 bit global pointer**

**("char\*\_\_near" to "char\*\_\_far")**

**Arguments :**

**- X : 16 bit near pointer**

**Postcondition :**

**- B == page of returned global address**

**- X == offset of returned global address**

**- Y remains unchanged**

**- A is unspecified**

**\*/**

**/\*--------------------------- Convert 16 bit logical to 24 bit global pointer ----------------------------------\*/**

**/\* B:X = Global(X) \*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_FRAME**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**void NEAR \_CONV\_NEAR\_TO\_GLOBAL(void){**

**\_\_asm {**

**// syntax:**

**// input 16 bit offset is bit15..bit0**

**// ppage values: ppage7..ppage0**

**// epage values: epage7..epage0**

**// dpage values: dpage7..dpage0**

**// rpage values: rpage7..rpage0**

**PSHX ;// D contains bit15..bit0**

**TFR X,D ;// D is cheaper to shift**

**LSLD ;// D contains 0 bit14..bit0, C contains bit15**

**BCC Below8000 ;// bit15 == 0?**

**// from 0x8000 to 0xFFFF**

**LSLD ;// D contains 00 bit13..bit0, C contains bit14**

**BCC BelowC000**

**LDAB #0x7F**

**PULX**

**RTS ;// returns 0b0111 1111 11 bit13...bit0**

**BelowC000: ;// from 0x8000 to 0xBFFF**

**TFR D,X**

**LDAB \_\_PPAGE\_ADR\_\_**

**SEC**

**RORB**

**RORX**

**LSRB**

**RORX**

**LEAS 2,SP**

**RTS ;// returns 0b01 ppage7..ppage0 bit13...bit0**

**Below8000:**

**LSLD ;// D contains 00 bit13..bit0, C contains bit14**

**BCC Below4000**

**// from 0x4000 to 0x7FFF**

**PULX**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**LDAB #0x7F**

**#else**

**LEAX (0xC000-0x4000),X**

**LDAB #0x0F**

**#endif**

**RTS ;// returns 0b0111 1111 01 bit13...bit0**

**Below4000:**

**LSLD ;// D contains 000 bit12..bit0, C contains bit13**

**BCC Below2000**

**// from 0x2000 to 0x3FFF**

**PULX**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**LEAX (0xE000-0x2000),X**

**#else**

**LEAX (0xA000-0x2000),X**

**#endif**

**LDAB #0x0F**

**RTS ;// returns 0b0000 1111 111 bit12...bit0**

**Below2000:**

**LSLD ;// D contains 0000 bit11..bit0, C contains bit12**

**BCC Below1000**

**// from 0x1000 to 0x1FFF**

**LDAB \_\_RPAGE\_ADR\_\_**

**LDAA #0x10**

**MUL**

**EORB 0,SP**

**EORB #0x10 ;// clear 1 bit**

**STAB 0,SP**

**TFR A,B**

**PULX**

**RTS**

**Below1000:**

**LSLD ;// D contains 0000 0 bit10..bit0, C contains bit11**

**BCC Below0800**

**// from 0x0800 to 0x0FFF**

**LSLD ;// D contains 0000 00 bit9..bit0, C contains bit10**

**BCC Below0C00**

**// from 0x0C00 to 0x0FFF**

**LDAB #0x13**

**PULX**

**LEAX 0xF000,X**

**RTS ;// returns 0b0001 0011 1111 11 bit9...bit0**

**Below0C00:**

**// from 0x0800 to 0x0BFF**

**LDAB \_\_EPAGE\_ADR\_\_**

**LDAA #0x04**

**MUL**

**EORB 0,SP**

**EORB #0x08**

**STAB 0,SP**

**TFR A,B**

**ORAB #0b00010000**

**PULX**

**RTS**

**Below0800:**

**PULX**

**CLRB**

**RTS**

**}**

**}**

**/\*--------------------------- \_CONV\_STACK\_NEAR\_TO\_GLOBAL --------------------------------**

**Convert 16 bit logical of address on the stack 24 bit global pointer**

**("char\*\_\_near" to "char\*\_\_far")**

**Arguments :**

**- X : 16 bit near pointer**

**Postcondition :**

**- B == page of returned global address**

**- X == offset of returned global address**

**- Y remains unchanged**

**- A is unspecified**

**\*/**

**/\*--------------------------- Convert 16 bit logical stack address to 24 bit global pointer ----------------------------------\*/**

**/\* B:X = Global(D) \*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_FRAME**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**void NEAR \_CONV\_STACK\_NEAR\_TO\_GLOBAL(void){**

**\_\_asm {**

**// syntax:**

**// input 16 bit offset is bit15..bit0**

**// ppage values: ppage7..ppage0**

**// epage values: epage7..epage0**

**// dpage values: dpage7..dpage0**

**// rpage values: rpage7..rpage0**

**// stack must be between $1000 and $3FFF.**

**// actually placing the stack at $1000 implies that the RPAGE register is not set (and correctly initialized)**

**CPX #0x2000**

**BLO PAGED\_RAM**

**// Map 0x2000 to 0x0FE000 (0x0FA000 for HCS12XE RAM mapping is enabled)**

**LDAB #0x0F**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**LEAX (0xE000-0x2000),X ;// LEAX is one cycle faster than ADDX #**

**#else**

**LEAX (0xA000-0x2000),X ;// LEAX is one cycle faster than ADDX #**

**#endif**

**RTS**

**PAGED\_RAM:**

**PSHX**

**LDAB \_\_RPAGE\_ADR\_\_**

**LDAA #0x20**

**MUL**

**EORB 0,SP**

**EORB #0x10 ;// clear 1 bit**

**STAB 0,SP**

**TFR A,B**

**PULX**

**RTS**

**}**

**}**

**/\*--------------------------- \_CONV\_LOGICAL\_TO\_GLOBAL --------------------------------**

**Convert 24 bit global to 24 bit logical pointer**

**("char\*\_\_far" to "char\*\_\_gpage")**

**Arguments :**

**- B : page part of logical address**

**- X : 16 offset part of logical address**

**Postcondition :**

**- B == page of returned global address**

**- X == offset of returned global address**

**- Y remains unchanged**

**- A remains unchanged**

**\*/**

**/\*--------------------------- Convert 24 bit logical to 24 bit global pointer ----------------------------------\*/**

**/\* B:X = Logical(B:X) \*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_FRAME**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**void NEAR \_CONV\_LOGICAL\_TO\_GLOBAL(void) {**

**\_\_asm {**

**// syntax:**

**// input 16 bit offset is bit15..bit0**

**// ppage values: ppage7..ppage0**

**// epage values: epage7..epage0**

**// dpage values: dpage7..dpage0**

**// rpage values: rpage7..rpage0**

**PSHA ;// save A across this routine.**

**PSHX ;// D contains bit15..bit0**

**PSHB ;// store page**

**TFR X,D ;// D is cheaper to shift**

**LSLD ;// D contains 0 bit14..bit0, C contains bit15**

**BCC Below8000 ;// bit15 == 0?**

**// from 0x8000 to 0xFFFF**

**LSLD ;// D contains 00 bit13..bit0, C contains bit14**

**BCC BelowC000**

**PULB ;// cleanup stack**

**LDAB #0x7F**

**PULX**

**PULA**

**RTS ;// returns 0b0111 1111 11 bit13...bit0**

**BelowC000: ;// from 0x8000 to 0xBFFF**

**TFR D,X**

**PULB ;// cleanup stack**

**SEC**

**RORB**

**RORX**

**LSRB**

**RORX**

**LEAS 2,SP**

**PULA**

**RTS ;// returns 0b01 ppage7..ppage0 bit13...bit0**

**Below8000:**

**LSLD ;// D contains 00 bit13..bit0, C contains bit14**

**BCC Below4000**

**;// from 0x4000 to 0x7FFF**

**PULB ;// cleanup stack**

**PULX**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**LDAB #0x7F**

**#else**

**LEAX (0xC000-0x4000),X**

**LDAB #0x0F**

**#endif**

**PULA**

**RTS ;// returns 0b0111 1111 01 bit13...bit0**

**Below4000:**

**LSLD ;// D contains 000 bit12..bit0, C contains bit13**

**BCC Below2000**

**// from 0x2000 to 0x3FFF**

**PULB ;// cleanup stack**

**PULX**

**#ifndef \_\_HCS12XE\_RAMHM\_SET\_\_**

**LEAX (0xE000-0x2000),X**

**#else**

**LEAX (0xA000-0x2000),X**

**#endif**

**LDAB #0x0F**

**PULA**

**RTS ;// returns 0b0000 1111 111 bit12...bit0**

**Below2000:**

**LSLD ;// D contains 0000 bit11..bit0, C contains bit12**

**BCC Below1000**

**// from 0x1000 to 0x1FFF**

**PULB**

**LDAA #0x10**

**MUL**

**EORB 0,SP**

**EORB #0x10 ;// clear 1 bit**

**STAB 0,SP**

**TFR A,B**

**PULX**

**PULA**

**RTS**

**Below1000:**

**LSLD ;// D contains 0000 0 bit10..bit0, C contains bit11**

**BCC Below0800**

**// from 0x0800 to 0x0FFF**

**LSLD ;// D contains 0000 00 bit9..bit0, C contains bit10**

**BCC Below0C00**

**// from 0x0C00 to 0x0FFF**

**PULB ;// cleanup stack**

**LDAB #0x13**

**PULX**

**LEAX 0xF000,X**

**PULA**

**RTS ;// returns 0b0001 0011 1111 11 bit9...bit0**

**Below0C00:**

**// from 0x0800 to 0x0BFF**

**PULB**

**LDAA #0x04**

**MUL**

**EORB 0,SP**

**EORB #0x08**

**STAB 0,SP**

**TFR A,B**

**ORAB #0b00010000**

**PULX**

**PULA**

**RTS**

**Below0800:**

**PULB**

**PULX**

**PULA**

**CLRB**

**RTS**

**}**

**}**

**/\*--------------------------- \_FAR\_COPY\_RC HCS12X Routines --------------------------------**

**copy larger far memory blocks**

**There are the following memory block copy routines:**

**\_COPY : 16 bit logical copies.**

**Src and dest are both near. Note: implemented in rtshc12.c and not here.**

**\_FAR\_COPY\_RC HC12/HCS12 struct copy routine.**

**Expects HC12/HCS12 logical 24 bit address.**

**Note: Does not exist for the HCS12X.**

**The HC12/HCS12 implementation is implemented above.**

**\_FAR\_COPY\_GLOBAL\_GLOBAL\_RC:**

**\_FAR\_COPY\_GLOBAL\_LOGICAL\_RC:**

**\_FAR\_COPY\_LOGICAL\_GLOBAL\_RC:**

**\_FAR\_COPY\_LOGICAL\_LOGICAL\_RC:**

**\_FAR\_COPY\_NEAR\_GLOBAL\_RC:**

**\_FAR\_COPY\_NEAR\_LOGICAL\_RC:**

**\_FAR\_COPY\_GLOBAL\_NEAR\_RC:**

**\_FAR\_COPY\_LOGICAL\_NEAR\_RC: HCS12X specific far copy routine. The name describes what the src/dest address format are.**

**All near src arguments are passed in X, all 24 bit src in X/B.**

**All near dest arguments are passed in Y, all 24 bit src in Y/A.**

**(Note: HC12 \_FAR\_COPY\_RC is using X/A as src and Y/B as dest, so the register usage is not the same!)**

**Arguments :**

**- B:X : src address (for NEAR/\_COPY: only X)**

**- A:Y : dest address (for NEAR/\_COPY: only Y)**

**- number of bytes to be copied behind return address (for \_COPY: in D register). The number of bytes is always > 0**

**Result :**

**- memory area copied**

**- no registers are saved, i.e. all registers may be destroyed**

**- for \_COPY: D contains 0.**

**- for HCS12X \_FAR\_COPY\_... routines: GPAGE state is unknown**

**\*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_GLOBAL\_GLOBAL\_RC(void) {**

**\_\_asm {**

**PSHD**

**PSHY**

**LDY 4,SP ;// load return address**

**LDD 2,Y+ ;// load size**

**STY 4,SP ;// store return address**

**PULY**

**PSHD**

**LDAB 3,SP**

**Loop:**

**STAB \_\_GPAGE\_ADR\_\_**

**GLDAA 1,X+**

**MOVB 2,SP,\_\_GPAGE\_ADR\_\_**

**GSTAA 1,Y+**

**DECW 0,SP**

**BNE Loop**

**LEAS 4,SP**

**\_SRET ;// debug info only: This is the last instr of a function with a special return**

**RTS**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_SET\_PAGE\_REG\_HCS12X(void) {**

**// Sets the page contained in A to the register controlling the logical addr contained in X.**

**// saves the old page before and returns it in A together with the page address just below the return address.**

**// X/Y both remain valid.**

**\_\_asm {**

**PSHX**

**// 0000..FFFF**

**CPX #0x8000**

**BLO \_LO8000**

**LDX #\_\_PPAGE\_ADR\_\_**

**BRA Handle**

**\_LO8000:**

**// 0000..7FFF**

**CPX #0x1000**

**BLO \_LO1000**

**LDX #\_\_RPAGE\_ADR\_\_**

**BRA Handle**

**\_LO1000:**

**LDX #\_\_EPAGE\_ADR\_\_**

**Handle:**

**LDAA 0,X ;// load old page register content**

**STAB 0,X ;// set new page register**

**STX 4,SP**

**PULX**

**RTS**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_GLOBAL\_LOGICAL\_RC(void) {**

**\_\_asm {**

**STAB \_\_GPAGE\_ADR\_\_**

**EXG X,Y**

**TFR A,B**

**PSHY ;// space to store size**

**PSHX ;// allocate some space where \_SET\_PAGE\_REG\_HCS12X can return the page**

**LDY 4,SP ;// load return address**

**LDX 2,Y+ ;// load size**

**STY 4,SP**

**LDY 2,SP ;// restore dest pointer**

**STX 2,SP ;// store size**

**LDX 0,SP ;// reload src pointer**

**\_\_PIC\_JSR(\_SET\_PAGE\_REG\_HCS12X)**

**Loop: GLDAB 1,Y+**

**STAB 1,X+**

**DECW 2,SP**

**BNE Loop**

**PULX ;// reload page register address**

**STAA 0,X ;// restore old page content (necessary if it was PPAGE)**

**PULX ;// clean up stack**

**\_SRET ;// debug info only: This is the last instr of a function with a special return**

**RTS**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_LOGICAL\_GLOBAL\_RC(void) {**

**\_\_asm {**

**STAA \_\_GPAGE\_ADR\_\_**

**PSHY ;// space to store size**

**PSHX ;// allocate some space where \_SET\_PAGE\_REG\_HCS12X can return the page**

**LDY 4,SP ;// load return address**

**LDX 2,Y+ ;// load size**

**STY 4,SP**

**LDY 2,SP ;// restore dest pointer**

**STX 2,SP ;// store size**

**LDX 0,SP ;// reload src pointer**

**\_\_PIC\_JSR(\_SET\_PAGE\_REG\_HCS12X)**

**Loop: LDAB 1,X+**

**GSTAB 1,Y+**

**DECW 2,SP**

**BNE Loop**

**PULX**

**STAA 0,X ;// restore old page content (necessary if it was PPAGE)**

**PULX ;// clean up stack**

**\_SRET ;// debug info only: This is the last instr of a function with a special return**

**RTS**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_LOGICAL\_LOGICAL\_RC(void) {**

**\_\_asm {**

**PSHA**

**\_\_PIC\_JSR(\_CONV\_LOGICAL\_TO\_GLOBAL);**

**PULA**

**\_\_PIC\_JMP(\_FAR\_COPY\_GLOBAL\_LOGICAL\_RC);**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_NEAR\_GLOBAL\_RC(void) {**

**\_\_asm {**

**CLRB**

**\_\_PIC\_JMP(\_FAR\_COPY\_LOGICAL\_GLOBAL\_RC);**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_NEAR\_LOGICAL\_RC(void) {**

**\_\_asm {**

**PSHA**

**\_\_PIC\_JSR(\_CONV\_NEAR\_TO\_GLOBAL);**

**PULA**

**\_\_PIC\_JMP(\_FAR\_COPY\_GLOBAL\_LOGICAL\_RC);**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_GLOBAL\_NEAR\_RC(void) {**

**\_\_asm {**

**CLRA /\* near to logical (we may have to use another runtime if this gets non trivial as well :-( \*/**

**\_\_PIC\_JMP(\_FAR\_COPY\_GLOBAL\_LOGICAL\_RC);**

**}**

**}**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_LOGICAL\_NEAR\_RC(void) {**

**\_\_asm {**

**EXG A,B**

**EXG X,Y**

**PSHA**

**\_\_PIC\_JSR(\_CONV\_NEAR\_TO\_GLOBAL);**

**PULA**

**EXG A,B**

**EXG X,Y**

**\_\_PIC\_JMP(\_FAR\_COPY\_LOGICAL\_GLOBAL\_RC);**

**}**

**}**

**/\* \_FAR\_COPY\_LOGICAL\_GLOBAL: is used by some old wizard generated projects. Not used by current setup anymore \*/**

**#ifdef \_\_cplusplus**

**extern "C"**

**#endif**

**#pragma NO\_ENTRY**

**#pragma NO\_EXIT**

**#pragma NO\_FRAME**

**void NEAR \_FAR\_COPY\_LOGICAL\_GLOBAL(void) {**

**\_\_asm {**

**STAA \_\_GPAGE\_ADR\_\_**

**PSHX ;// allocate some space where \_SET\_PAGE\_REG\_HCS12X can return the page**

**\_\_PIC\_JSR(\_SET\_PAGE\_REG\_HCS12X)**

**Loop: LDAB 1,X+**

**GSTAB 1,Y+**

**DECW 4,SP**

**BNE Loop**

**PULX**

**STAA 0,X ;// restore old page content (necessary if it was PPAGE)**

**LDX 4,SP+ ;// load return address and clean stack**

**JMP 0,X**

**}**

**}**

**#endif /\* \_\_HCS12X\_\_ \*/**

**/\*----------------- end of code ------------------------------------------------\*/**

**Lcddisp.c \***

**#include "derivative.h"**

**#include "funct.h"**

**int data = 0; //used to determine if a command or data is being sent, sets RS**

**#define LCDDATA PTADHi**

**#define LCDDATADDR DDRADHi**

**#define LCDCONTROLDDR DDRM**

**#define ECLK PTM\_PTM4**

**#define RW PTM\_PTM5**

**#define RS PTM\_PTM6**

**#define LINE1 0x80**

**#define LINE2 0xc0**

**void delay\_ms(int delay) //Not a precise delay**

**{**

**int x;**

**while (delay > 0)**

**{**

**for (x = 700; x > 0; x--);**

**delay--;**

**}**

**}**

**void init\_LCD()**

**{**

**LCDDATADDR = 0xFF; //Set PORT AD 8-15 as outputs,**

**LCDDATA = 0x00; // Clear Port AD 8-15 I/O register**

**LCDCONTROLDDR = LCDCONTROLDDR | 0x70; //Set bits 4, 5, and 6 of Port M as outputs (ECLK, RW, RS respectively)**

**delay\_ms(5);**

**ECLK = 1;**

**delay\_ms(50); //Small Start Delay**

**//LCD Initialization**

**data = 0;**

**LCDDATA = 0x03;**

**epulse();**

**delay\_ms(5);**

**epulse();**

**delay\_ms(5);**

**epulse();**

**delay\_ms(5);**

**//Setup LCD Screen**

**sendCommand(0x3C); //Set Interface Length, 8-data lines, 2 display lines**

**sendCommand(0x10); //Turn off Display**

**sendCommand(0x01); //Clear Display**

**sendCommand(0x06); //Set Cursor Move direction, Auto-Increment cursor after each byte written, dont shift display**

**sendCommand(0x0c); //Display on, cursor off**

**}**

**void display\_string(char\* SendStr)**

**{**

**unsigned char count=0;**

**//return home**

**sendCommand(LINE1); //Start from the beginning of the LCD (Line 1)**

**while(\*SendStr!=0)**

**{**

**if(count == 16)**

**{**

**sendCommand(LINE2); //Switch to Line 2**

**}**

**++count;**

**data = 1; //Send Character**

**LCDDATA = \*SendStr;**

**epulse();**

**++SendStr;**

**}**

**}**

**void Delay(unsigned int DelayCount)**

**{**

**unsigned int i, j;**

**for(i = 0; i < DelayCount; i++)**

**for(j = 0; j < 2000; j++)**

**;**

**}**

**void epulse(void) //Pulse E Clock**

**{**

**if (data) RS = 1;**

**else RS = 0;**

**delay\_ms(1);**

**ECLK = 0;**

**delay\_ms(1);**

**ECLK = 1;**

**}**

**void sendCommand( int byte )//Send a byte command to LCD**

**{**

**data = 0; //Send Command**

**LCDDATA = byte;**

**epulse();**

**}**

**Funct.h \***

**#ifndef FUNCT\_H**

**#define FUNCT\_H**

**#define LCDDATA PTADHi**

**#define LCDDATADDR DDRADHi**

**#define LCDCONTROLDDR DDRM**

**#define ECLK PTM\_PTM4**

**#define RW PTM\_PTM5**

**#define RS PTM\_PTM6**

**#define LINE1 0x80**

**#define LINE2 0xc0**

**void init\_LCD(void);**

**void display\_string(char \*);**

**void read\_pot(void);**

**void Delay(unsigned int);**

**void delay\_ms(int);**

**void epulse(void);**

**void sendCommand(int);**

**#endif**

**Derivative.h \***

**/\***

**\* Note: This file is recreated by the project wizard whenever the MCU is**

**\* changed and should not be edited by hand**

**\*/**

**/\* Include the derivative-specific header file \*/**

**#include <mc9s12e128.h>**

**#pragma LINK\_INFO DERIVATIVE "mc9s12e128"**

**Main\_asm.h \***

**#ifndef \_MAIN\_ASM\_H**

**#define \_MAIN\_ASM\_H**

**#ifdef \_\_cplusplus**

**extern "C" { /\* our assembly functions have C calling convention \*/**

**#endif**

**void asm\_main(void);**

**/\* interface to my assembly main function \*/**

**#ifdef \_\_cplusplus**

**}**

**#endif**

**#endif /\* \_MAIN\_ASM\_H \*/**

**Potentiometer.c \***

**#include "derivative.h"**

**#include "funct.h"**

**unsigned int pot\_value;**

**void read\_pot(void)**

**{**

**//turn on A/D**

**ATDCTL2 = 0x80;**

**//wait for power-up**

**Delay(2);**

**//start conversion channel 0**

**ATDCTL5 = 0;**

**//wait for conversion complete**

**while(!(ATDSTAT0 & 0x80))**

**;**

**//get data**

**pot\_value = ATDDR0H;**

**}**

## Traveling.asm

**XDEF traveling,travel\_flag**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad, Pen1, Pen2, Pen3,disp**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**bitmask1: equ $7F ;01111111 all switches other then emergency need to be on**

**ddr\_t EQU $242 ;data direction register of DC motor**

**MY\_EXTENDED\_RAM: SECTION**

**; Insert here your data definition.**

**travel\_flag: ds.b 1**

**MY\_CODE: SECTION**

**traveling:movb #$1E,p\_ddr**

**MOVB #08,ddr\_t ; set ddr t to output**

**movb #'T',disp**

**movb #'u',disp+1**

**movb #'r',disp+2**

**movb #'n',disp+3**

**movb #' ',disp+4**

**movb #'O',disp+5**

**movb #'f',disp+6**

**movb #'f',disp+7**

**movb #' ',disp+8**

**movb #'P',disp+9**

**movb #'a',disp+10**

**movb #'r',disp+11**

**movb #'k',disp+12**

**movb #'i',disp+13**

**movb #'n',disp+14**

**movb #'g',disp+15**

**movb #' ',disp+16**

**movb #'B',disp+17**

**movb #'r',disp+18**

**movb #'e',disp+19**

**movb #'a',disp+20**

**movb #'k',disp+21**

**movb #'!',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**start: ldaa port\_t**

**bita #bitmask1 ;checks switches aginst brake value**

**bne start ;will only break out once the sequence is matched**

**movb #1,travel\_flag ;starts the travel flag for the interrupt**

**movb #'T',disp**

**movb #'h',disp+1**

**movb #'e',disp+2**

**movb #' ',disp+3**

**movb #'c',disp+4**

**movb #'u',disp+5**

**movb #'r',disp+6**

**movb #'r',disp+7**

**movb #'e',disp+8**

**movb #'n',disp+9**

**movb #'t',disp+10**

**movb #' ',disp+11**

**movb #'s',disp+12**

**movb #'p',disp+13**

**movb #'e',disp+14**

**movb #'e',disp+15**

**movb #'d',disp+16**

**movb #' ',disp+17**

**movb #'i',disp+18**

**movb #'s',disp+19**

**movb #':',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #'m',disp+25**

**movb #'p',disp+26**

**movb #'h',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32**

**jsr read\_pot**

**ldd pot\_value**

**MOVB #$08, port\_t ;set bit 3 of DC motor register to rotate**

**ldab #$30**

**ldx #100**

**idiv ;D/X q in x remainder in d**

**ldab #$30**

**abx**

**tfr x,a**

**staa disp+21**

**ldx #10**

**idiv ; q in x reaminder in d**

**ldab #$30**

**abx**

**tfr x,a**

**staa disp+22**

**addd #$30**

**std disp+23**

**ldd #disp**

**jsr display\_string ;displays pot value into lcd one digit at a time**

**ldaa travel\_flag ;travel flag stays on while traveling and the interrupt will turn it off after a set time**

**cmpa #0**

**beq pen\_select**

**lbra traveling**

**pen\_select: movb #$00,port\_t ;stops the dc motor**

**ldab travel ;after travel time is done will get the travel value from main menu and jump to that pen**

**cmpb #1**

**beq pen1**

**cmpb #2**

**beq pen2**

**cmpb #3**

**beq pen3 ;travel will be 4 when returning so that it will just leave the subroutine instead of visiting another pen**

**bra end**

**pen1: jsr Pen1**

**pen2: jsr Pen2**

**pen3: jsr Pen3**

**end: rts**

## Pen3.asm

**XDEF Pen3**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,feed3,roar3,hunger3,keyval,disp,traveling**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**port\_s: equ $248**

**ddr\_s: equ $24A**

**MY\_EXTENDED\_RAM: SECTION**

**seq1 dc.b $0,$1,$3,$7,$F,$1F,$3F,$7F,$FF**

**MY\_CODE: SECTION**

**Pen3: ldab hunger3 ;gets hunger val and an array with the leds that match index of hunger val**

**ldx #seq1**

**ldaa b,x ;loads a with the one led value that matches hunger that displays it**

**staa port\_s**

**jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input same as main menu**

**ldab pot\_value**

**cmpb #$33**

**lbge option2**

**movb #'N',disp**

**movb #'a',disp+1**

**movb #'m',disp+2**

**movb #'e',disp+3**

**movb #':',disp+4**

**movb #' ',disp+5**

**movb #'C',disp+6**

**movb #'e',disp+7**

**movb #'r',disp+8**

**movb #'a',disp+9**

**movb #' ',disp+10**

**movb #' ',disp+11**

**movb #'T',disp+12**

**movb #'Y',disp+13**

**movb #'p',disp+14**

**movb #'e',disp+15**

**movb #':',disp+16**

**movb #' ',disp+17**

**movb #'T',disp+18**

**movb #'r',disp+19**

**movb #'i',disp+20**

**movb #'c',disp+21**

**movb #'e',disp+22**

**movb #'r',disp+23**

**movb #'a',disp+24**

**movb #'t',disp+25**

**movb #'o',disp+26**

**movb #'p',disp+27**

**movb #'s',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen3**

**option2: cmpb #$66**

**lbge option3**

**movb #'D',disp**

**movb #'i',disp+1**

**movb #'e',disp+2**

**movb #'t',disp+3**

**movb #':',disp+4**

**movb #' ',disp+5**

**movb #'H',disp+6**

**movb #'e',disp+7**

**movb #'r',disp+8**

**movb #'b',disp+9**

**movb #'i',disp+10**

**movb #'v',disp+11**

**movb #'o',disp+12**

**movb #'r',disp+13**

**movb #'e',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #'1',disp+18**

**movb #'.',disp+19**

**movb #'F',disp+20**

**movb #'e',disp+21**

**movb #'e',disp+22**

**movb #'d',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen3**

**option3: cmpb #$99**

**lbge option4**

**movb #'2',disp**

**movb #'.',disp+1**

**movb #'H',disp+2**

**movb #'e',disp+3**

**movb #'a',disp+4**

**movb #'r',disp+5**

**movb #' ',disp+6**

**movb #'r',disp+7**

**movb #'o',disp+8**

**movb #'a',disp+9**

**movb #'r',disp+10**

**movb #' ',disp+11**

**movb #' ',disp+12**

**movb #'3',disp+13**

**movb #'.',disp+14**

**movb #'R',disp+15**

**movb #'e',disp+16**

**movb #'t',disp+17**

**movb #'u',disp+18**

**movb #'r',disp+19**

**movb #'n',disp+20**

**movb #' ',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #' ',disp+24**

**movb #'g',disp+25**

**movb #'a',disp+26**

**movb #'t',disp+27**

**movb #'e',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen3**

**option4: jsr KeyPad ;after maxed**

**ldaa keyval**

**cmpa #1**

**bne check2**

**jsr feed3**

**check2: cmpa #2**

**bne check3**

**jsr roar3**

**check3: cmpa #3**

**bne check4**

**movb #4,travel**

**jsr traveling**

**rts**

**check4: lbra Pen3**

## Pen2.asm

**XDEF Pen2**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,feed2,roar2,hunger2,keyval,disp,traveling**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**port\_s: equ $248**

**ddr\_s: equ $24A**

**MY\_EXTENDED\_RAM: SECTION**

**seq1 dc.b $0,$1,$3,$7,$F,$1F,$3F,$7F,$FF**

**MY\_CODE: SECTION**

**Pen2: ldab hunger2 ;gets hunger val and an array with the leds that match index of hunger val**

**ldx #seq1**

**ldaa b,x ;loads a with the one led value that matches hunger that displays it**

**staa port\_s**

**jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input**

**ldab pot\_value**

**cmpb #$33**

**lbge option2**

**movb #'N',disp**

**movb #'a',disp+1**

**movb #'m',disp+2**

**movb #'e',disp+3**

**movb #':',disp+4**

**movb #' ',disp+5**

**movb #'D',disp+6**

**movb #'e',disp+7**

**movb #'l',disp+8**

**movb #'t',disp+9**

**movb #'a',disp+10**

**movb #' ',disp+11**

**movb #'T',disp+12**

**movb #'Y',disp+13**

**movb #'p',disp+14**

**movb #'e',disp+15**

**movb #':',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #'R',disp+19**

**movb #'a',disp+20**

**movb #'p',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #'r',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen2**

**option2: cmpb #$66**

**lbge option3**

**movb #'D',disp**

**movb #'i',disp+1**

**movb #'e',disp+2**

**movb #'t',disp+3**

**movb #':',disp+4**

**movb #' ',disp+5**

**movb #'C',disp+6**

**movb #'a',disp+7**

**movb #'r',disp+8**

**movb #'n',disp+9**

**movb #'i',disp+10**

**movb #'v',disp+11**

**movb #'o',disp+12**

**movb #'r',disp+13**

**movb #'e',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #'1',disp+18**

**movb #'.',disp+19**

**movb #'F',disp+20**

**movb #'e',disp+21**

**movb #'e',disp+22**

**movb #'d',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen2**

**option3: cmpb #$99**

**lbge option4**

**movb #'2',disp**

**movb #'.',disp+1**

**movb #'H',disp+2**

**movb #'e',disp+3**

**movb #'a',disp+4**

**movb #'r',disp+5**

**movb #' ',disp+6**

**movb #'r',disp+7**

**movb #'o',disp+8**

**movb #'a',disp+9**

**movb #'r',disp+10**

**movb #' ',disp+11**

**movb #' ',disp+12**

**movb #'3',disp+13**

**movb #'.',disp+14**

**movb #'R',disp+15**

**movb #'e',disp+16**

**movb #'t',disp+17**

**movb #'u',disp+18**

**movb #'r',disp+19**

**movb #'n',disp+20**

**movb #' ',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #' ',disp+24**

**movb #'g',disp+25**

**movb #'a',disp+26**

**movb #'t',disp+27**

**movb #'e',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen2**

**option4: jsr KeyPad**

**ldaa keyval**

**cmpa #1**

**bne check2**

**jsr feed2**

**check2: cmpa #2**

**bne check3**

**jsr roar2**

**check3: cmpa #3**

**bne check4**

**movb #4,travel**

**jsr traveling**

**rts**

**check4: lbra Pen2**

## Pen1.asm

**XDEF Pen1**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,feed1,roar1,hunger1,keyval,disp,traveling**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**port\_s: equ $248**

**ddr\_s: equ $24A**

**MY\_EXTENDED\_RAM: SECTION**

**seq1 dc.b $0,$1,$3,$7,$F,$1F,$3F,$7F,$FF**

**MY\_CODE: SECTION**

**Pen1: ldab hunger1 ;gets hunger val and an array with the leds that match index of hunger val**

**ldx #seq1**

**ldaa b,x ;loads a with the one led value that matches hunger that displays it**

**staa port\_s**

**jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input**

**ldab pot\_value**

**cmpb #$33**

**lbge option2**

**movb #'N',disp**

**movb #'a',disp+1**

**movb #'m',disp+2**

**movb #'e',disp+3**

**movb #':',disp+4**

**movb #' ',disp+5**

**movb #'T',disp+6**

**movb #'i',disp+7**

**movb #'n',disp+8**

**movb #'y',disp+9**

**movb #' ',disp+10**

**movb #'T',disp+11**

**movb #'y',disp+12**

**movb #'p',disp+13**

**movb #'e',disp+14**

**movb #':',disp+15**

**movb #' ',disp+16**

**movb #'T',disp+17**

**movb #'-',disp+18**

**movb #'R',disp+19**

**movb #'e',disp+20**

**movb #'x',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen1**

**option2: cmpb #$66**

**lbge option3**

**movb #'D',disp**

**movb #'i',disp+1**

**movb #'e',disp+2**

**movb #'t',disp+3**

**movb #':',disp+4**

**movb #' ',disp+5**

**movb #'C',disp+6**

**movb #'a',disp+7**

**movb #'r',disp+8**

**movb #'n',disp+9**

**movb #'i',disp+10**

**movb #'v',disp+11**

**movb #'o',disp+12**

**movb #'r',disp+13**

**movb #'e',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #'1',disp+18**

**movb #'.',disp+19**

**movb #'F',disp+20**

**movb #'e',disp+21**

**movb #'e',disp+22**

**movb #'d',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen1**

**option3: cmpb #$99**

**lbge option4**

**movb #'2',disp**

**movb #'.',disp+1**

**movb #'H',disp+2**

**movb #'e',disp+3**

**movb #'a',disp+4**

**movb #'r',disp+5**

**movb #' ',disp+6**

**movb #'r',disp+7**

**movb #'o',disp+8**

**movb #'a',disp+9**

**movb #'r',disp+10**

**movb #' ',disp+11**

**movb #' ',disp+12**

**movb #'3',disp+13**

**movb #'.',disp+14**

**movb #'R',disp+15**

**movb #'e',disp+16**

**movb #'t',disp+17**

**movb #'u',disp+18**

**movb #'r',disp+19**

**movb #'n',disp+20**

**movb #' ',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #' ',disp+24**

**movb #'g',disp+25**

**movb #'a',disp+26**

**movb #'t',disp+27**

**movb #'e',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra Pen1**

**option4: jsr KeyPad**

**ldaa keyval**

**cmpa #1**

**bne check2**

**jsr feed1**

**check2: cmpa #2**

**bne check3**

**jsr roar1**

**check3: cmpa #3**

**bne check4**

**movb #4,travel**

**jsr traveling**

**rts**

**check4: lbra Pen1**

## Feed1.asm

**XDEF feed1**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,Pen1,hunger1,anger1,keyval,disp**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**MY\_CODE: SECTION**

**feed1: jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input**

**ldab pot\_value**

**cmpb #$33 ;lion meat only good for carnivores**

**lbge option2 ;mega-salad only good of herbicores**

**movb #'1',disp ;cake is magical and makes dino happier and restores food units**

**movb #'.',disp+1**

**movb #'L',disp+2**

**movb #'i',disp+3**

**movb #'o',disp+4**

**movb #'n',disp+5**

**movb #' ',disp+6**

**movb #'(',disp+7**

**movb #'M',disp+8**

**movb #'e',disp+9**

**movb #'a',disp+10**

**movb #'t',disp+11**

**movb #')',disp+12**

**movb #' ',disp+13**

**movb #' ',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed1**

**option2: cmpb #$66**

**lbge option3**

**movb #'2',disp**

**movb #'.',disp+1**

**movb #'M',disp+2**

**movb #'e',disp+3**

**movb #'g',disp+4**

**movb #'a',disp+5**

**movb #'-',disp+6**

**movb #'S',disp+7**

**movb #'a',disp+8**

**movb #'l',disp+9**

**movb #'a',disp+10**

**movb #'d',disp+11**

**movb #' ',disp+12**

**movb #'(',disp+13**

**movb #'V',disp+14**

**movb #'e',disp+15**

**movb #'g',disp+16**

**movb #'a',disp+17**

**movb #'t',disp+18**

**movb #'a',disp+19**

**movb #'r',disp+20**

**movb #'i',disp+21**

**movb #'a',disp+22**

**movb #'n',disp+23**

**movb #')',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed1**

**option3: cmpb #$99**

**lbge option4**

**movb #'3',disp**

**movb #'.',disp+1**

**movb #'C',disp+2**

**movb #'a',disp+3**

**movb #'k',disp+4**

**movb #'e',disp+5**

**movb #' ',disp+6**

**movb #'D',disp+7**

**movb #'e',disp+8**

**movb #'s',disp+9**

**movb #'e',disp+10**

**movb #'r',disp+11**

**movb #'t',disp+12**

**movb #' ',disp+13**

**movb #'.',disp+14**

**movb #'4',disp+15**

**movb #'.',disp+16**

**movb #'R',disp+17**

**movb #'e',disp+18**

**movb #'r',disp+19**

**movb #'n',disp+20**

**movb #' ',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #' ',disp+24**

**movb #'p',disp+25**

**movb #'e',disp+26**

**movb #'n',disp+27**

**movb #'1',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed1**

**option4: jsr KeyPad**

**ldaa keyval**

**cmpa #1**

**bne check2**

**inc hunger1**

**lbra feed1**

**check2: cmpa #2**

**bne check3**

**inc anger1**

**lbra feed1**

**check3: cmpa #3**

**bne check4**

**dec anger1**

**inc hunger1**

**lbra feed1**

**check4: rts**

## Feed2.asm

**XDEF feed2**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,Pen2,hunger2,anger2,disp,keyval**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**MY\_CODE: SECTION**

**feed2: jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input**

**ldab pot\_value**

**cmpb #$33**

**lbge option2**

**movb #'1',disp**

**movb #'.',disp+1**

**movb #'L',disp+2**

**movb #'i',disp+3**

**movb #'o',disp+4**

**movb #'n',disp+5**

**movb #' ',disp+6**

**movb #'(',disp+7**

**movb #'M',disp+8**

**movb #'e',disp+9**

**movb #'a',disp+10**

**movb #'t',disp+11**

**movb #')',disp+12**

**movb #' ',disp+13**

**movb #' ',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed2**

**option2: cmpb #$66**

**lbge option3**

**movb #'2',disp**

**movb #'.',disp+1**

**movb #'M',disp+2**

**movb #'e',disp+3**

**movb #'g',disp+4**

**movb #'a',disp+5**

**movb #'-',disp+6**

**movb #'S',disp+7**

**movb #'a',disp+8**

**movb #'l',disp+9**

**movb #'a',disp+10**

**movb #'d',disp+11**

**movb #' ',disp+12**

**movb #'(',disp+13**

**movb #'V',disp+14**

**movb #'e',disp+15**

**movb #'g',disp+16**

**movb #'a',disp+17**

**movb #'t',disp+18**

**movb #'a',disp+19**

**movb #'r',disp+20**

**movb #'i',disp+21**

**movb #'a',disp+22**

**movb #'n',disp+23**

**movb #')',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed2**

**option3: cmpb #$99**

**lbge option4**

**movb #'3',disp**

**movb #'.',disp+1**

**movb #'C',disp+2**

**movb #'a',disp+3**

**movb #'k',disp+4**

**movb #'e',disp+5**

**movb #' ',disp+6**

**movb #'D',disp+7**

**movb #'e',disp+8**

**movb #'s',disp+9**

**movb #'e',disp+10**

**movb #'r',disp+11**

**movb #'t',disp+12**

**movb #' ',disp+13**

**movb #'.',disp+14**

**movb #'4',disp+15**

**movb #'.',disp+16**

**movb #'R',disp+17**

**movb #'e',disp+18**

**movb #'r',disp+19**

**movb #'n',disp+20**

**movb #' ',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #' ',disp+24**

**movb #'p',disp+25**

**movb #'e',disp+26**

**movb #'n',disp+27**

**movb #'2',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed2**

**option4: jsr KeyPad**

**ldaa keyval**

**cmpa #1**

**bne check2**

**inc hunger2**

**lbra feed2**

**check2: cmpa #2**

**bne check3**

**inc anger2**

**lbra feed2**

**check3: cmpa #3**

**bne check4**

**dec anger2**

**inc hunger2**

**lbra feed2**

**check4: rts**

## Feed3.asm

**XDEF feed3**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,Pen3,hunger3,anger3,keyval,disp**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**MY\_CODE: SECTION**

**feed3: jsr read\_pot ;uses pot to scroll menu and after it is maxed will read an input**

**ldab pot\_value**

**cmpb #$33**

**lbge option2**

**movb #'1',disp**

**movb #'.',disp+1**

**movb #'L',disp+2**

**movb #'i',disp+3**

**movb #'o',disp+4**

**movb #'n',disp+5**

**movb #' ',disp+6**

**movb #'(',disp+7**

**movb #'M',disp+8**

**movb #'e',disp+9**

**movb #'a',disp+10**

**movb #'t',disp+11**

**movb #')',disp+12**

**movb #' ',disp+13**

**movb #' ',disp+14**

**movb #' ',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed3**

**option2: cmpb #$66**

**lbge option3**

**movb #'2',disp**

**movb #'.',disp+1**

**movb #'M',disp+2**

**movb #'e',disp+3**

**movb #'g',disp+4**

**movb #'a',disp+5**

**movb #'-',disp+6**

**movb #'S',disp+7**

**movb #'a',disp+8**

**movb #'l',disp+9**

**movb #'a',disp+10**

**movb #'d',disp+11**

**movb #' ',disp+12**

**movb #'(',disp+13**

**movb #'V',disp+14**

**movb #'e',disp+15**

**movb #'g',disp+16**

**movb #'a',disp+17**

**movb #'t',disp+18**

**movb #'a',disp+19**

**movb #'r',disp+20**

**movb #'i',disp+21**

**movb #'a',disp+22**

**movb #'n',disp+23**

**movb #')',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed3**

**option3: cmpb #$99**

**lbge option4**

**movb #'3',disp**

**movb #'.',disp+1**

**movb #'C',disp+2**

**movb #'a',disp+3**

**movb #'k',disp+4**

**movb #'e',disp+5**

**movb #' ',disp+6**

**movb #'D',disp+7**

**movb #'e',disp+8**

**movb #'s',disp+9**

**movb #'e',disp+10**

**movb #'r',disp+11**

**movb #'t',disp+12**

**movb #' ',disp+13**

**movb #'.',disp+14**

**movb #'4',disp+15**

**movb #'.',disp+16**

**movb #'R',disp+17**

**movb #'e',disp+18**

**movb #'r',disp+19**

**movb #'n',disp+20**

**movb #' ',disp+21**

**movb #'t',disp+22**

**movb #'o',disp+23**

**movb #' ',disp+24**

**movb #'p',disp+25**

**movb #'e',disp+26**

**movb #'n',disp+27**

**movb #'3',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**lbra feed3**

**option4: jsr KeyPad**

**ldaa keyval**

**cmpa #2**

**bne check2**

**inc hunger3**

**lbra feed3**

**check2: cmpa #1**

**bne check3**

**inc anger3**

**lbra feed3**

**check3: cmpa #3**

**bne check4**

**dec anger3**

**inc hunger3**

**lbra feed3**

**check4: rts**

## Delay2.asm

**XDEF Delay2**

**Delay2: pshy**

**ldy #1000**

**loop: dey**

**bne loop**

**puly**

**rts**

**SendsChr.c \***

**#include <hidef.h> /\* common defines and macros \*/**

**#include "derivative.h" /\* derivative-specific definitions \*/**

**#include "funct.h"**

**int tone=0; //value to hold the tone to be generated**

**int tone\_count=0; //counter to test the tone to be generated**

**void SendsChr(char NewTone, int dummy)**

**{**

**dummy++; //increment the dummy value to supress the warning**

**tone=NewTone; //set the tone passed on the stack**

**}**

**void PlayTone(void)**

**{**

**tone\_count++; //increment tone counter**

**if(tone\_count>=tone) //if the tone count is equal to or higher than the tone**

**{**

**tone\_count=0; //reset the counter**

**PTT = PTT ^ 0x20; //toggle the speaker bit.**

**}**

**}**

## Roar1.asm

**XDEF roar1**

**XREF SendsChr,PlayTone**

**MY\_CODE: SECTION**

**roar1:**

**ldx 5,sp ;puts data in X**

**ldy 3,sp ;puts the counter in Y**

**nextval: ldaa 1,x+; gets a value from the array one at a time**

**psha ;pushes the current value from the array to stack**

**call SendsChr**

**call PlayTone**

**leas -1,sp**

**dey**

**bne nextval**

**rts**

## Roar2.asm

**XDEF roar2**

**XREF SendsChr,PlayTone**

**MY\_CODE: SECTION**

**roar2:**

**ldx 5,sp ;puts data in X**

**ldy 3,sp ;puts the counter in Y**

**nextval: ldaa 1,x+; gets a value from the array one at a time**

**psha ;pushes the current value from the array to stack**

**call SendsChr**

**call PlayTone**

**leas -1,sp**

**dey**

**bne nextval**

**rts**

## Roar3.asm

**XDEF roar3**

**XREF SendsChr,PlayTone**

**MY\_CODE: SECTION**

**roar3:**

**ldx 5,sp ;puts data in X**

**ldy 3,sp ;puts the counter in Y**

**nextval: ldaa 1,x+; gets a value from the array one at a time**

**psha ;pushes the current value from the array to stack**

**call SendsChr**

**call PlayTone**

**leas -1,sp**

**dey**

**bne nextval**

**rts**

## Emergency\_menu.asm

**XDEF emergency\_menu**

**XREF travel,init\_LCD,read\_pot,display\_string,pot\_value,KeyPad,keyval,disp,emergency\_flag,hunger1,hunger2,hunger3,anger1,anger2,anger3,button\_val**

**My\_Constant: section**

**port\_p equ $258**

**p\_ddr equ $25A**

**port\_t: equ $240 ;input ports**

**t\_ddr: equ $242**

**port\_s: equ $248**

**ddr\_s: equ $24A**

**My\_Variable: section**

**var\_1 dc.b $0**

**MY\_CODE: SECTION**

**emergency\_menu: ldaa port\_t ;makes sure emergency switch is on before printing menu**

**bita #%10000000 ;if switch is off then ends the menu**

**lbne end**

**ldaa var\_1**

**orab #%11111111 ;toggle value for all leds**

**stab port\_s**

**movb #'D',disp**

**movb #'i',disp+1**

**movb #'n',disp+2**

**movb #'o',disp+3**

**movb #' ',disp+4**

**movb #' ',disp+5**

**movb #'l',disp+6**

**movb #'o',disp+7**

**movb #'o',disp+8**

**movb #'s',disp+9**

**movb #'e',disp+10**

**movb #'!',disp+11**

**movb #'!',disp+12**

**movb #' ',disp+13**

**movb #'C',disp+14**

**movb #'a',disp+15**

**movb #'l',disp+16**

**movb #'l',disp+17**

**movb #' ',disp+18**

**movb #'S',disp+19**

**movb #'e',disp+20**

**movb #'c',disp+21**

**movb #'u',disp+22**

**movb #'r',disp+23**

**movb #'i',disp+24**

**movb #'t',disp+25**

**movb #'y',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**button\_loop: ldaa button\_val ;button val should be 1 after IRQ is pressed during an emergency**

**cmpa $1**

**bne button\_loop**

**Password:**

**movb #' ',disp**

**movb #'E',disp+1 ;asks for password**

**movb #'n',disp+2**

**movb #'t',disp+3**

**movb #'e',disp+4**

**movb #'R',disp+5**

**movb #' ',disp+6**

**movb #'P',disp+7**

**movb #'a',disp+8**

**movb #'s',disp+9**

**movb #'s',disp+10**

**movb #'c',disp+11**

**movb #'o',disp+12**

**movb #'d',disp+13**

**movb #'e',disp+14**

**movb #':',disp+15**

**movb #' ',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #':',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**pass1: jsr display\_string**

**jsr KeyPad ;gets a keypad value and if it matches checks next key**

**ldaa keyval**

**cmpa #1 ;if the correct value is entered then displays a X in the first slot and gets the second value**

**bne pass1**

**movb #'X',disp+17**

**ldd #disp**

**jsr display\_string**

**pass2: jsr KeyPad**

**jsr display\_string**

**ldaa keyval ;gets second key and will return here until second key is pressed**

**cmpa #2**

**bne pass2**

**movb #'X',disp+18**

**ldd #disp**

**jsr display\_string**

**pass3: jsr KeyPad**

**jsr display\_string**

**ldaa keyval**

**cmpa #3**

**bne pass3**

**movb #'X',disp+19**

**ldd #disp**

**jsr display\_string**

**pass4: jsr KeyPad**

**jsr display\_string**

**ldaa keyval**

**cmpa #4**

**bne pass4**

**movb #' ',disp**

**movb #'C',disp+1**

**movb #'a',disp+2 ;Sucessful login screen**

**movb #'l',disp+3**

**movb #'l',disp+4**

**movb #'i',disp+5**

**movb #'n',disp+6**

**movb #'g',disp+7**

**movb #' ',disp+8**

**movb #'S',disp+9**

**movb #'e',disp+10**

**movb #'c',disp+11**

**movb #'u',disp+12**

**movb #'r',disp+13**

**movb #'i',disp+14**

**movb #'t',disp+15**

**movb #'y',disp+16**

**movb #' ',disp+17**

**movb #' ',disp+18**

**movb #' ',disp+19**

**movb #' ',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**end: movb $0,emergency\_flag**

**movb #$8,hunger1**

**movb #$8,hunger2**

**movb #$8,hunger3**

**movb #$0,anger1**

**movb #$0,anger2**

**movb #$0,anger3**

**rts**

## RTI\_ISR.asm

**XDEF RTI\_ISR,hunger1,hunger2,hunger3,anger1,anger2,anger3,dino\_count,lockdown,travel\_count,second,seq\_count**

**XREF emergency\_flag,travel\_flag,emergency\_menu,roar1,roar2,roar3,display\_string,disp,Gameover,RTIFLG**

**MY\_Constant: section**

**val1 dc.b $0a,$12,$14,$0C**

**p\_ddr equ $25A**

**port\_p equ $258**

**MY\_VARIABLE:SECTION**

**second: ds.w 1**

**seq\_count: ds.w 1**

**travel\_count: ds.w 1**

**dino\_count: ds.w 1**

**hunger1: ds.b 1**

**hunger2: ds.b 1**

**hunger3: ds.b 1**

**anger1: ds.b 1**

**anger2: ds.b 1**

**anger3: ds.b 1**

**lockdown: ds.w 1**

**My\_code: Section**

**RTI\_ISR: ldaa emergency\_flag**

**cmpa $0**

**bne danger ;checks emergency flag and decides speed depending**

**inc second**

**ldy second**

**cpy #1000 ;full speed every second ;if emergency is off then ticks every second**

**bne travel\_ISR**

**ldy #0**

**sty second**

**ldx #val1**

**ldaa b,x**

**staa port\_p**

**inc seq\_count**

**ldab seq\_count**

**cmpb #4**

**bne travel\_ISR**

**ldab #0**

**stab seq\_count**

**lbra travel\_ISR**

**danger: inc second**

**ldy second**

**cpy #500 ;double speed every half second**

**lbne emergency\_ISR**

**ldy #0**

**sty second**

**ldx #val1 ;if emergency on then ticks every half second**

**ldaa b,x**

**staa port\_p**

**incb**

**cmpb #4**

**lbne emergency\_ISR**

**ldab #0**

**stab seq\_count**

**travel\_ISR: ldaa travel\_flag**

**cmpa #0**

**beq dino\_ISR**

**INC travel\_count ;increment count**

**LDX travel\_count ;count till 1000 for 1 millisecond**

**CPX #20000 ;20 seconds**

**BNE dino\_ISR ;branch if equal to 1 millisecond, exit**

**LDX #0 ;reset to 0 if = 1s**

**STX travel\_count**

**movb #0,travel\_flag**

**;travel for 20 seconds then reset travel flag which is set and reset from main menu**

**dino\_ISR: ldaa hunger1 ;check hunger before decrement and hunger should not go below 0 but just incase**

**cmpa $00**

**lble set\_emergency1**

**ldaa hunger2**

**cmpa $00**

**lble set\_emergency2**

**ldaa hunger3**

**cmpa $00**

**lble set\_emergency3**

**ldaa anger1 ;check hunger before decrement and hunger should not go below 0 but just incase**

**cmpa $03**

**lble set\_emergency1**

**ldaa anger2**

**cmpa $03**

**lble set\_emergency2**

**ldaa anger2**

**cmpa $03**

**lble set\_emergency3**

**INC dino\_count ;increment count**

**LDX dino\_count ;count till 1000 for 1 millisecond**

**CPX #20000 ;20 seconds**

**BNE check2 ;branch if equal to 1 millisecond, exit**

**dec hunger2**

**check2: cpx #25000 ;25 seconds**

**bne check3**

**dec hunger3**

**check3: cpx #30000 ;30 seconds**

**lbne EXIT**

**LDX #0 ;reset to 0 if = 1s**

**STX dino\_count**

**dec hunger1**

**lbra emergency\_ISR**

**set\_emergency1: movb #$1, emergency\_flag**

**jsr roar1**

**jsr emergency\_menu**

**lbra emergency\_ISR**

**set\_emergency2: movb #$1, emergency\_flag**

**jsr roar2**

**jsr emergency\_menu**

**lbra emergency\_ISR**

**set\_emergency3: movb #$1, emergency\_flag**

**jsr roar3**

**jsr emergency\_menu**

**emergency\_ISR:ldaa emergency\_flag**

**cmpa $1**

**lbne EXIT ;checks emergency flag and decides speed depending**

**inc lockdown**

**ldy lockdown**

**cpy #35000 ;35 seconds**

**lbne EXIT**

**ldy #0**

**sty lockdown**

**jsr Gameover**

**EXIT: BSET RTIFLG, $80 ;reset RTI flag**

**RTI**

## Gameover.asm

**XDEF Gameover**

**XREF display\_string,disp,roar1,roar2,roar3,emergency\_flag,hunger1,hunger2,hunger3,anger1,anger2,anger3**

**MY\_CODE:SECTION**

**Gameover: movb #'G',disp**

**movb #'a',disp+1**

**movb #'m',disp+2**

**movb #'e',disp+3**

**movb #' ',disp+4**

**movb #'O',disp+5**

**movb #'v',disp+6**

**movb #'e',disp+7**

**movb #'r',disp+8**

**movb #' ',disp+9**

**movb #'D',disp+10**

**movb #'i',disp+11**

**movb #'n',disp+12**

**movb #'o',disp+13**

**movb #'s',disp+14**

**movb #' ',disp+15**

**movb #'W',disp+16**

**movb #'i',disp+17**

**movb #'n',disp+18**

**movb #'!',disp+19**

**movb #'!',disp+20**

**movb #' ',disp+21**

**movb #' ',disp+22**

**movb #' ',disp+23**

**movb #' ',disp+24**

**movb #' ',disp+25**

**movb #' ',disp+26**

**movb #' ',disp+27**

**movb #' ',disp+28**

**movb #' ',disp+29**

**movb #' ',disp+30**

**movb #' ',disp+31**

**movb #0,disp+32 ;string terminator, acts like '\0'**

**ldd #disp**

**jsr display\_string**

**jsr roar1**

**jsr roar2**

**jsr roar3**

**movb $0,emergency\_flag**

**movb #$8,hunger1**

**movb #$8,hunger2**

**movb #$8,hunger3**

**movb #$0,anger1**

**movb #$0,anger2**

**movb #$0,anger3**

**rts**

## IRQ\_ISR.asm

**XDEF IRQ\_ISR,button\_val**

**XREF IRQCR,emergency\_flag**

**MY\_VARIABLE:SECTION**

**button\_val: ds.w 1**

**MY\_CODE:SECTION**

**IRQ\_ISR: ldaa emergency\_flag**

**cmpa #0**

**lbge END**

**movb #1,button\_val ;flags button as pressed only after emergency menu is on**

**END: RTI**

# Appendix III: Project and Report Requirements

## Project Requirements:

Jurassic Park

Objective:

The objective of this project is to write a program that simulates Jurassic Park. The user will act as the park ranger and tend to the needs of the dinosaurs. They will also be responsible for some of the logistics of the park including lockdown and security procedures in case of an emergency. Dinosaurs will get mad and break out if they aren’t fed, so the ranger will have to be sure to feed them periodically or the park might be destroyed!

Requirements:

1. The system must have at least 3 types of dinosaurs, each with a unique name, hunger capacity, type (herbivore, carnivore) and sound.

2. The system must have at least 3 different kinds of food (sheep, watermelon), each with a certain amount of food units.

3. The user will have to first login with a set username and password (this can be preset), then there will be a menu that allows them to choose which dinosaur paddock to visit. At each paddock, the user can view the statistics of the dinosaur (name, hunger level, type), listen to their sound, and feed them.

4. The user will have to drive a Jeep between paddocks since they’re not within walking distance.

5. Each dinosaur should have a unique hunger capacity (in food units) which goes down over time. In order to replenish it, the user will have to feed them the appropriate food to restore that food’s amount of units. If a dinosaur’s hunger reaches 0, then it should get mad and break out of containment. This should cause the system to alert the user that a dinosaur broke out and play that dinosaurs sound. This should also start a new countdown. If the user doesn’t call security before time is up, the dinosaur will destroy the park.

6. If the user tries to feed the dinosaur the wrong kind of food, the dinosaur will just get mad and make its sound instead of eating.

7. If the park is destroyed there should be a failure message on the screen and the program should reset.

Peripherals:

1. Stepper Motor: The stepper motor should act as the ranger’s wristwatch. It should tick clockwise once every second. When the dinosaur breaks out, it should start to tick faster to add to the panic!

2. DC Motor: The DC motor will simulate the Jeep that the park ranger uses to drive between paddocks. Each time the user chooses a new paddock, there must be a short drive simulation with a rotating dc motor while the user “drives” there. The user should be able to adjust the speed of the Jeep using the potentiometer as the gas pedal. This should make the motor spin faster but magically will not affect trip time.

3. Keypad: The keypad will be used to enter in credentials when the ranger clocks into work. This will be required to enter the main system menu and start the simulation. It will also be used to make menu choices in each of the system menus.

4. Potentiometer: The potentiometer should be used to scroll between menu options in the main setup menu. For example, if there are three dinosaurs to choose from, the LCD screen should display one at a time for choosing and the potentiometer will be used to cycle through them. It should also be used to control the speed of the Jeep (DC motor) while driving between paddocks.

5. LCD: The LCD will be the main display for the menu and alert system. It will display the menus and various messages such as dinosaur hunger level and emergency alerts when dinosaurs break out.

6. Push Button: The push button should be used to initiate a lockdown and call security when a dinosaur breaks out. It will first ask the user for their password, then sound the system alarm and broadcast a message over the LCD to call security. If this isn’t done in time after a dinosaur breaks out then the park will be destroyed.

7. Speaker: The speaker should be used to play dinosaur sounds and the alarm when security is called.

8. Switches: The switches should be used as a security handbrake for the Jeep. It shouldn’t move until the user has entered the correct configuration (can be preset). One switch should also be used separately to control the alarm system. If this switch is set, the alarm will sound. Otherwise it will not and security will not receive an alert even when the push button system is used.

9. LEDs: The LEDs should show the dinosaurs hunger when the ranger views their statistics (as a rough percentage out of 8). They should also flash on and off when the alarm is sounding.

10. RTI: A real-time interrupt should be used to control all the system timing. No delay subroutines are allowed except for the keypad debouncing.

Note:

You are encouraged to be creative and make this project your own. You can make reasonable assumptions in the development of this project, but keep in mind that the assumptions must make sense to the user (and to the Lab TAs).

If you have any questions pertaining to this project, please discuss it with your Lab TAs as early as possible. You may use any C code provided through the lab, but all other codes must be written in assembly.

## Report Requirements:

Final Project Report Requirement

The final project should include the following sections:

1. Cover letter:

a. The project title

b. The name of the team members

c. Project date (Fall 2013)

d. Should include a picture of the equipment and/or a picture relevant to the project

2. Table of contents

3. List of figures and tables

4. Introduction:

a. Describe the goal and the purpose of the project.

b. Assumptions made

5. Design:

a. Show the peripherals that are used in the project and what they are used for

b. Software implementation of the project:

i. Give a high level description and discussion

ii. General system flow chart

iii. Flow charts for each module

iv. Error handling and fail safe techniques

c. Changes made in the design

d. Additions to the project

6. Description of the division of work between team members

7. Description of which parts of the proposed project is working and which part is not working

8. Conclusion

9. Discussion and suggestions for future improvements on your project

10. Appendixes:

a. User Manual

b. Code:

i. The code should be commented (useful and meaningful comments)

ii. Description of each file, subroutine and procedure:

• Name

• Inputs/Outputs and method of passing the parameters

• General Description

11. References

Note:

• Use headings and subheadings throughout the report

• Pay attention to grammatical and spelling errors

• All figures and flow charts should be done using software (paint, Visio etc.)

• The report should be consistent in style

• CODE SHOULD BE SUBMITTED AS ENTIRE PROJECT IN ZIPPED FORMAT ALONG WITH REPORT.

• Fonts and sizes:

o Single spaced

o Use the “Times New Roman” font or any similar font

o Use font size 14 bold for headings

o Use font size 12 for subheadings

o Use font size 12 for text

o Use the “Courier New” font for the code and the size should be 10