```
*****************************************************************************
*
*  MCF5206 COLDFIRE Assembly Header, Sample Initialization File             *
*  and Test Code                                                            *
*                                                                           *
*  Developed by:      John Cunningham                                       *
*                     Motorola                                              *
*                     Imaging and Storage Systems Division                  *
*                     Austin, TX                                            *
*****************************************************************************
*
* REV 1 beta (6/27/97)
*
*****************************************************************************
*
* This file contains the following information to aid designers            *
* using the MCF5206:                                                        *
*
*
*       1)Assembly header listing (i.e. EQUATES memory map)                 *
*       2)Module initialization code (SIM,UART, TIMER etc.)                 *
*       3)Sample test code for TIMER and UART                               *
*       4)Batch file for assembling with DIAB DATA assembler                *
*       5)Initialization file for use on the MCF5206 development system     *
*       6)DIAB DATA linker file
*
*****************************************************************************
*

****************
* Introduction *
*****************************************************************************
*
* This file provides examples on initializing and                          *
* configuring the various modules on the COLDFIRE  5206 processor.         *
* An explanation of how the module is configured is shown prior to         *
* the actual code.  It is best viewed using WORDPAD or NOTEPAD by          *
* Microsoft.                                                               *
*
*
* This code is written as if it were for a specific                        *
* design.  The specifications for that design are:                         *
*
**
*       1)Code space less than 16 meg (only address lines 0:23 used)       *
*       2)Dram size is 4 Meg, one bank utilized                             *
*       3)Upper addresses used for chip selects                             *
*       4)Both UARTS are used and use no interrupts                         *
*       5)MBUS port (I2C) is used, (interrupts disabled)                    *
*       5)Parallel port is used                                             *
*       6)Timers are used       (interrupts disabled)                       *
*       8)SRAM is used.  Stack pointer is set to the top of the SRAM        *
*       9)Cache disabled
*
```

```
*****************************************************************************
*

*****************************************************************************
*
* IMPORTANT:
        *
*
        *
* The SBC5206 development system uses a different memory mapping when       *
* compared to this initialization example code.  The evaluation board*
* boots off of FLASH and sets its code space to run out of DRAM.  All       *
* lines that have been commented out are to enable this code to work*
* on the SBC5206 development system
        *
*
        *
* ALSO: Make sure to use at least a 1 amp power supply with the             *
*      development system.  A 800ma wall-pack power supply has been         *
*         known not to have enough current when running UARTs.              *
*
        *

*****************************************************************************
*
* The 5206 development system sets these registers to the following *
* values:
        *
*       RAMBAR      $20000000
        *
*       MBAR        $10000000
        *
*       CACR        $00000000
        *
*       ACR0        $FFFF60E4
        *
*       ACR1        $FFFF60E4
        *
*       VBAR        $00000000
        *
*
        *
*****************************************************************************
*
* This code has been tested on the SBC5206 eval system with the             *
* DRAM, CHIP SELECTS, and the PAR register commented out ";".               *
*
        *
* This code gives the designer an idea on how to       *
* initialize and test the many modules and registers on the 5206. This      *
* code can be used, as is, in an embedded application, where the            *
* code for the DRAM, CHIP SELECTS, and the PAR are configured as shown      *
* in this example.  The user just needs to uncomment the lines.             *
```

```
*
        *
* Unfortunately, this code has only been tested with the 5206          *
* development system as it is written now. It is the responsibility    *
* of the end user to verify that this example code works in their      *
* design (i.e. with specific lines uncommented such as CACHE or DRAM   *
* initialization code).                                                          *
*********************************************************************************
*
*********************************************************************************
*

**************
* Memory map *
**************
*********************************************************************************
*
* The memory map for this example is set up as follows.  Note:  This          *
* example assumed a 29F040 flash memory size which is 4 Mbit or               *
* 524,288 bytes but the entire code space can be used. Remember, this         *
* map differs if using the SBC5206 development system.                        *
*
        *
*
        *
*       $00000000    -------------------------                              *
*                    -  usable code space   -                                       *
*                    -                              -
        *
*                   -                              -
        *
*                   -                              -
        *
*       $000F0000    -------------------------                              *
*                   -                              -
        *
*                   -                              -
        *
*                   -------------------------                              *
*       $00700000  ------------------------- VBR table                      *
*                   -  DRAM space (4 meg)          -
        *
*                   -                              -
        *
*       $00AFFFFF   -------------------------                              *
*       $00B00000   -------------------------                              *
*                   -         unused           -                                   *
*                   -                              -
        *
*                   -                              -
        *
*                   -------------------------                              *
*       $20000000   -------------------------                              *
```

```
*                       -       SRAM space          -
        *
*                       -   (512 bytes)             -
        *
*                       -                           -
        *
*       $200001FF       ------------------------                        *
*       $20000200       ------------------------                        *
*                       -       unused          -                               *
*                       -                           -
        *
*                       -                           -
        *
*       $FFEFFFFF       ------------------------                        *
*       $FFF003FD       ------------------------                        *
*                       -       unused          -                               *
*                       -                           -
        *
*                       ------------------------                        *
*       $FFFF0200       ------------------------                        *
*                       -       unused          -                               *
*                       -                           -
        *
*                       ------------------------                        *
*       $FFFFF000       ------------------------                        *
*                       -       Module base         -
        *
*                       -       (496 bytes)         -
        *
*                       -                           -
        *
*       $FFFF01F0       ------------------------                        *
*
        *
*******************************************************************************
*

************************
* DIAB DATA batch file *
*******************************************************************************
*
* The following lines are the lines used to compile this code. This     *
* batch file assumes the following:                                             *
*       1) The DIAB DATA compiler is located on the C drive                     *
*       2) The assembly file is named 5206jc.s and is located in the    *
*         directory c:\5206\test                                                *
*
        *
* Create a directory "C:\5206\test. Create a file called 5206.bat and    *
* with the 3 command lines listed below and copy it into the            *
* C:\5206\test directory.(Make sure to remove the "*" comments.)                *
*
        *
* c:\diab\3.7a\win32\bin\das -l -o c:\5206\test\5206jc.o                        *
```

```
*         -WDDTARGET=MCF5200-WDDOBJECT=F c:\5206\test\5206jc.s
          *
* c:\diab\3.7a\win32\bin\dldnew -YS,. -YP,c:\diab\3.7a\ace0fs -o          *
*         c:\5206\test\5206jc.elf c:\5206\test\5206jc.o -lc ace_diab.lnk *
* c:\diab\3.7a\win32\bin\ddump -R -o c:\5206\test\5206jc.hex              *
*         c:\5206\test\5206jc.elf                                                 *
*************************************************************************
*

***********************
* DIAB DATA link file *
*************************************************************************
*
* In addition to the batch file and the assembly file, DIAB DATA uses     *
* a link file.  For this design, copy the lines listed below into the     *
* C:\5206\test directory and name the file "Ace_diab.lnk".                *
* More information about the DIAB DATA linker can be found on the         *
* COLDFIRE CD that came with the SBC5206 development system promo *
* boards.  This CD contains the manual in PDF format.  The manual         *
* provides examples of how to define and change the attributes            *
* associated with the linker.                                             *
*************************************************************************
*
*
* MEMORY
* {
*  ram : org = 0x10000,len = 0x100000
* }
*
* SECTIONS
* {
*    .text: {} >ram
*    .data: {} >ram
*    .bss:  {} >ram
* }
*
* __HEAP_START   = ADDR(.bss)+SIZEOF(.bss);
* __SP_INIT      = ADDR(ram)+SIZEOF(ram);
* STKTOP         = __SP_INIT;
* __HEAP_END     = __SP_INIT-0x800;
* __SP_END       = __HEAP_END;
* __DATA_ROM     = ADDR(.text)+SIZEOF(.text);
* __DATA_RAM     = ADDR(.data);
* __DATA_END     = ADDR(.data)+SIZEOF(.data);
* __BSS_START    = ADDR(.bss);
* __BSS_END      = ADDR(.bss)+SIZEOF(.bss);

* ___HEAP_START   = __HEAP_START;
* ___SP_INIT      = __SP_INIT;
* ___HEAP_END     = __HEAP_END;
* ___SP_END       = __SP_END;
* ___DATA_ROM     = __DATA_ROM;
* ___DATA_RAM     = __DATA_RAM;
* ___DATA_END     = __DATA_END;
```

```
* ___BSS_START    = __BSS_START;
* ___BSS_END      = __BSS_END;
*
*****************************************************************************
*


*****************************************************************************
*
*                                EQUATES SECTION
        *
*****************************************************************************
*

*****************************************************************************
*
* The MBAR (Module Base Address Register)defines the base address       *
* of all internal peripheral registers.  It is accessed in the CPU      *
* space $C0F via the MOVEC instruction.  See COLDFIRE 5206 User manual, *
* pg 7-1.  An example on setting the MBAR to location $10000000         *
* would be:
        *
*                       move.l #$10000000,D0                            *
*                       movec  D0,MBAR                                  *
*
        *
* The assembler/compiler knows to store it in the system control      *
* register MBAR and "MBAR" does not have to be defined in an EQU              *
* statement  Make sure you type "MBAR" (Not case sensitive).          *
*
        *
* All the other registers are offsets of the MBAR.  I created a variable*
* called MBARx and used an "EQU" statement to define its value.  Make   *
* sure to give the MBARx value the same value you stored in the MBAR.   *
* Example
        *
*               MBARx           EQU    $10000000
        *
*
        *
* The MBARx value was defined to provide valid addresses for the        *
* rest of the memory map.                                                     *
*****************************************************************************
*

*****************************************************************************
*
* The MBARx that is commented out below is for this design example.     *
* It is commented out to enable this code to run on the SBC5206         *
* development system.  If you are using this code example without       *
* the development system, comment out the line that sets MBARx to       *
* $10000000 and uncomment the line that sets it to $FFFFF000.           *
*
        *
```

```
* The VBR is also commented out for use on the SBC5206 development      *
* system.  To properly set the VBR, it should be stored in an external  *
* SRAM/RAM location.  I suggest uing the first $3FF of your RAM for the *
* vector tables.  Therefore, to use this code for this design example   *
* comment the line that sets VBRx to $00000000 and use the line that    *
* sets it to $00700000 which is the beginning of DRAM.                  *
                                                                        *
*
        *
* RAMBARx is set the same as the SBC5206 development system.            *
*************************************************************************
*

*************************************************************************
*
* CPU Registers Memory Map
        *
* NOTE:  CPU registers must be accessed through a MOVEC instruction     *
* Registers that use the "MOVEC" command are:                                *
*       1)MBR 2)VBR 3)CACR 4)ACR0 5)ACR1 6)RAMBAR
        *
*
        *
* The assembler/compiler recognizes the above listed names and          *
* do not have to be defined in an EQU statement.  The example           *
* below will store the correct value in MBAR.  This is also true        *
* for VBR, CACR, ACR0, ACR1, RAMBAR
        *
*************************************************************************
*
* Example:
        *
*       move.l  #value,D0   ;Load data register with value          *
*       movec   D0,MBAR     ;Place value into CPU register          *
*************************************************************************
*
* Register   Location internal                                          *
* CACR      $00000002   ; Cache Control Register, 32-bit, W             *
* ACR0      $00000000   ; Access Control Register 0, 32-bit, W          *
* ACR1      $00000000   ; Access Control Register 1, 32-bit, W          *
* VBR       $00000801   ; Vector Base Register, 32-bit, W               *
* RAMBAR    $00000C04   ; SRAM Base Address Register, 32-bit, W              *
*************************************************************************
*
;MBARx      EQU   $FFFFF000     ;Module Base Address value
;                                       ;($FFFFF000 - $FFFFF1F0)(496 bytes)
MBARx       EQU   $10000000     ;Module Base Address value
                                        ;($10000000 - $100001F0)(496 bytes)

*************************************************************************
*
* The VBR should be set somewhere in RAM or SRAM.  The 5206                  *
* development system comes with 1 Meg of RAM where it executes its      *
* program code.  The 5206 development system begins executing code at   *
```

```
* address $00010000 and sets the VBR at location $00000000. The entire  *          *
* vector table uses 1024 bytes (Interrupts 0-255, 4 bytes each).                *
                                                                                *
*
          *
* This example initialization sets the DRAM1 bank at $00700000.                 *
* Therefore, if you are not using the 5206 development system,                  *
* the user should change the VBRx EQU statement below from $00000000 *
* to $00700000 to set the VBR at the beginning of RAM.                          *
*********************************************************************************
*
VBRx            EQU    $00000000     ;Vector Base Address ($000-3FC)
                                     ;1020 bytes.  Use register "A7"
                                     ;to load it's value

RAMBARx     EQU   $20000000      ;SRAM Base Address (512 bytes)
                                     ;and address masks (pg 5-2,3)
DRAM0x        EQU    $0070                 ;base address of DRAM0, $00700000

*********************************************************************
* System Integration Module                                                     *
*********************************************************************
SIMR       EQU      MBARx+$0003  ;SIM Configuration Register, 8-bit, R/W

*****************
* PAR register *
*****************
PAR        EQU      MBARx+$00CB  ;Pin Assignment Register, 8-bit, R/W

*********************************
* Interrupt EQUs                *
*********************************
ICR1       EQU      MBARx+$0014  ;Interrupt Control Register Ext1, 8-bit, R/W
ICR2       EQU      MBARx+$0015  ;Interrupt Control Register Ext2, 8-bit, R/W
ICR3       EQU      MBARx+$0016  ;Interrupt Control Register Ext3, 8-bit, R/W
ICR4       EQU      MBARx+$0017  ;Interrupt Control Register Ext4, 8-bit, R/W
ICR5       EQU      MBARx+$0018  ;Interrupt Control Register Ext5, 8-bit, R/W
ICR6       EQU      MBARx+$0019  ;Interrupt Control Register Ext6, 8-bit, R/W
ICR7       EQU      MBARx+$001A  ;Interrupt Control Register Ext7, 8-bit, R/W
ICR8       EQU      MBARx+$001B  ;Interrupt Control Register SWT, 8-bit, R/W
ICR9       EQU      MBARx+$001C  ;Interrupt Control Register TIMER1, 8-bit, R/W
ICR10      EQU      MBARx+$001D  ;Interrupt Control Register TIMER2, 8-bit, R/W
ICR11      EQU      MBARx+$001E  ;Interrupt Control Register MBUS, 8-bit, R/W
ICR12      EQU      MBARx+$001F  ;Interrupt Control Register UART1, 8-bit, R/W
ICR13      EQU      MBARx+$0020  ;Interrupt Control Register UART2, 8-bit, R/W
IMR        EQU      MBARx+$0036  ;Interrupt Mask Register, 32-bit, R/W
IPR        EQU      MBARx+$003A  ;Interrupt Pending Register, 32-bit, R
RSR        EQU      MBARx+$0040  ;Reset Status Register, 8-bit, R/W
*********************************
* Software Watchdog EQUs   *
*********************************
SYPCR      EQU      MBARx+$0041  ;System Protection Control Register, 8-bit, R/W
SWIVR      EQU      MBARx+$0042  ;Software Watchdog Interrupt Vector Register, 8-bit,
W
```

```
SWSR    EQU    MBARx+$0043  ;Software Watchdog Service Register,
                            ;8-bit, W

*****************************************************************
* DRAM Controller Registers                                     *
*****************************************************************
* This section defines the DRAM controller,   See pg 10-3 of the *
* of the 5206 users manual                                      *
*****************************************************************

DCRR    EQU    MBARx+$0046  ;DRAM Refresh Register, 16-bit, R/W
DCTR    EQU    MBARx+$004A  ;DRAM Timing Register, 16-bit, R/W
DCAR0   EQU    MBARx+$004C  ;DRAM Bank 0 Address Register, 16-bit, R/W
DCMR0   EQU    MBARx+$0050  ;DRAM Bank 0 Mask Register, 32-bit, R/W
DCCR0   EQU    MBARx+$0057  ;DRAM Bank 0 Control Register, 8-bit, R/W
DCAR1   EQU    MBARx+$0058  ;DRAM Bank 1 Address Register, 16-bit, R/W
DCMR1   EQU    MBARx+$005C  ;DRAM Bank 1 Mask Register, 32-bit, R/W
DCCR1   EQU    MBARx+$0063  ;DRAM Bank 1 Control Register, 8-bit, R/W

*****************************************************************
* Chip Select Registers                                         *
*****************************************************************
* This section defines Chip Selects for the 5206 Processor      *
* The information for this section can be found on pg 8-1 of the *
* Users manual.                                                 *
*****************************************************************

CSAR0   EQU    MBARx+$0064  ;Chip-Select 0 Base Address Register, 16-bit, R/W
CSMR0   EQU    MBARx+$0068  ;Chip-Select 0 Address Mask Register, 32-bit, R/W
CSCR0   EQU    MBARx+$006E  ;Chip-Select 0 Control Register, 16-bit, R/W
CSAR1   EQU    MBARx+$0070  ;Chip-Select 1 Base Address Register, 16-bit, R/W
CSMR1   EQU    MBARx+$0074  ;Chip-Select 1 Address Mask Register, 32-bit, R/W
CSCR1   EQU    MBARx+$007A  ;Chip-Select 1 Control Register, 16-bit, R/W
CSAR2   EQU    MBARx+$007C  ;Chip-Select 2 Base Address Register, 16-bit, R/W
CSMR2   EQU    MBARx+$0080  ;Chip-Select 2 Address Mask Register, 32-bit, R/W
CSCR2   EQU    MBARx+$0086  ;Chip-Select 2 Control Register, 16-bit, R/W
CSAR3   EQU    MBARx+$0088  ;Chip-Select 3 Base Address Register, 16-bit, R/W
CSMR3   EQU    MBARx+$008C  ;Chip-Select 3 Address Mask Register, 32-bit, R/W
CSCR3   EQU    MBARx+$0092  ;Chip-Select 3 Control Register, 16-bit, R/W
CSAR4   EQU    MBARx+$0094  ;Chip-Select 4 Base Address Register, 16-bit, R/W
CSMR4   EQU    MBARx+$0098  ;Chip-Select 4 Address Mask Register, 32-bit, R/W
CSCR4   EQU    MBARx+$009E  ;Chip-Select 4 Control Register, 16-bit, R/W
CSAR5   EQU    MBARx+$00A0  ;Chip-Select 5 Base Address Register, 16-bit, R/W
CSMR5   EQU    MBARx+$00A4  ;Chip-Select 5 Address Mask Register, 32-bit, R/W
CSCR5   EQU    MBARx+$00AA  ;Chip-Select 5 Control Register, 16-bit, R/W
CSAR6   EQU    MBARx+$00AC  ;Chip-Select 6 Base Address Register, 16-bit, R/W
CSMR6   EQU    MBARx+$00B0  ;Chip-Select 6 Address Mask Register, 32-bit, R/W
CSCR6   EQU    MBARx+$00B6  ;Chip-Select 6 Control Register, 16-bit, R/W
CSAR7   EQU    MBARx+$00B8  ;Chip-Select 7 Base Address Register, 16-bit, R/W
CSMR7   EQU    MBARx+$00BC  ;Chip-Select 7 Address Mask Register, 32-bit, R/W
CSCR7   EQU    MBARx+$00C2  ;Chip-Select 7 Control Register, 16-bit, R/W
DMCR    EQU    MBARx+$00C6  ;Default Memory Control Register, 16-bit, R/W

*****************************************************************
```

```
* Timer Registers                                                    *
**********************************************************************
* There are two timers on the MCF5206. With a 33 MHZ clock, the      *
* maximum period is 8 seconds and a resolution of 30ns.  See         *
* pg 13-1 of the users manual                                        *
**********************************************************************

***********
* Timer 1 *
***********
TMR1     EQU    MBARx+$0100  ;TIMER1 Mode Register, 16-bit, R/W
TRR1     EQU    MBARx+$0104  ;TIMER1 TIMER1 Mode Register, 16-bit, R/W
TCR1     EQU    MBARx+$0108  ;TIMER1 Capture Register, 16-bit, R
TCN1     EQU    MBARx+$010C  ;TIMER1 Counter, 16-bit, R/W
TER1     EQU    MBARx+$0111  ;TIMER1 Event Register, 8-bit, R/W

***********
* Timer 2 *
***********

TMR2     EQU    MBARx+$0120  ;TIMER2 Mode Register, 16-bit, R/W
TRR2     EQU    MBARx+$0124  ;TIMER2 Mode Register, 16-bit, R/W
TCR2     EQU    MBARx+$0128  ;TIMER2 Capture Register, 16-bit, R
TCN2     EQU    MBARx+$012C  ;TIMER2 Counter, 16-bit, R/W
TER2     EQU    MBARx+$0131  ;TIMER2 Event Register, 8-bit, R/W

**********************************************************************
* UART Module Registers                                              *
**********************************************************************
* The MCF5206 contains 2 universal asynchronous/synchronous     *
* receiver/transmitters (UARTS) that act independently.  See *
* pg 11-1 of the users manual. Note that some registers are   *
* defined with the same address location, such as USR & USCR.    *
* because one register is valid if READ and the other is         *
* valid when WRITE. See pg 11-17, table 11-1 for details in the  *
* 5206 user's manual.                                                *
**********************************************************************

*********
* UART1 *
*********

UMR11    EQU    MBARx+$0140  ;UART1 UMR1 Mode Register, 8-bit, R/W
UMR21    EQU    MBARx+$0140  ;UART1 UMR2 Mode Register, 8-bit, R/W
USR1     EQU    MBARx+$0144  ;UART1 Status Register, 8-bit, R
UCSR1    EQU    MBARx+$0144  ;UART1 Clock Select Register, 8-bit, W
UCR1     EQU    MBARx+$0148  ;UART1 Command Register, 8-bit, W
URB1     EQU    MBARx+$014C  ;UART1 Receiver Buffer, 8-bit, R
UTB1     EQU    MBARx+$014C  ;UART1 Transmitter Buffer, 8-bit, W
UIPCR1   EQU    MBARx+$0150  ;UART1 Input Port Change Register, 8-bit, R
UACR1    EQU    MBARx+$0150  ;UART1 Auxiliary Control Register, 8-bit, W
UISR1    EQU    MBARx+$0154  ;UART1 Interrupt Status Register, 8-bit, R
UIMR1    EQU    MBARx+$0154  ;UART1 Interrupt Mask Register, 8-bit, W
```

```
UBG11    EQU    MBARx+$0158  ;UART1 Baud Rate Generator PreScale MSB, 8-bit,
W
UBG21    EQU    MBARx+$015C  ;UART1 Baud Rate Generator PreScale LSB, 8-bit,
W
UIVR1    EQU    MBARx+$0170  ;UART1 Interrupt Vector Register, 8-bit, R/W
UIP1     EQU    MBARx+$0174  ;UART1 Input Port Register, 8-bit, R
UOP11    EQU    MBARx+$0178  ;UART1 Output Port Bit Set Command, 8-bit, W
UOP01    EQU    MBARx+$017C  ;UART1 Output Port Bit Reset Command, 8-bit, W

*********
* UART2 *
*********

UMR12    EQU    MBARx+$0180  ;UART2 UMR1 Mode Register, 8-bit, R/W
UMR22    EQU    MBARx+$0180  ;UART2 UMR1 Mode Register, 8-bit, R/W
USR2     EQU    MBARx+$0184  ;UART2 Status Register, 8-bit, R
UCSR2    EQU    MBARx+$0184  ;UART2 Clock Select Register, 8-bit, W
UCR2     EQU    MBARx+$0188  ;UART2 Command Register, 8-bit, W
URB2     EQU    MBARx+$018C  ;UART2 Receiver Buffer, 8-bit, R
UTB2     EQU    MBARx+$018C  ;UART2 Transmitter Buffer, 8-bit, W
UIPCR2   EQU    MBARx+$0190  ;UART2 Input Port Change Register, 8-bit, R
UACR2    EQU    MBARx+$0190  ;UART2 Auxiliary Control Register, 8-bit, W
UISR2    EQU    MBARx+$0194  ;UART2 Interrupt Status Register, 8-bit, R
UIMR2    EQU    MBARx+$0194  ;UART2 Interrupt Mask Register, 8-bit, W
UBG12    EQU    MBARx+$0198  ;UART1 Baud Rate Generator PreScale MSB, 8-bit,
W
UBG22    EQU    MBARx+$019C  ;UART2 Baud Rate Generator PreScale LSB, 8-bit,
W
UIVR2    EQU    MBARx+$01B0  ;UART2 Interrupt Vector Register, 8-bit, R/W
UIP2     EQU    MBARx+$01B4  ;UART2 Input Port Register, 8-bit, R
UOP12    EQU    MBARx+$01B8  ;UART2 Output Port Bit Set Command, 8-bit, W
UOP02    EQU    MBARx+$01BC  ;UART2 Output Port Bit Reset Command, 8-bit, W

*****************************************************************
* General Purpose I/O                                           *
*****************************************************************
* These registers define the 8-bit port.  The 8-bit port is muxed *
* with the internal processor status pins "PST[3..0]" and the     *
* "DDATA[3..0]" debug pins.  The PAR defines what function these  *
* pins have (i.e Debug or parallel port). See pg 7-16/17 to set   *
* up the PAR and pg 9-2 to set up the port direction and logic    *
* level.                                                          *
*****************************************************************

PADDR    EQU    MBARx+$01C5  ;Port A Data Direction Register,
                              ;8-bit, R/W
PADAT    EQU    MBARx+$01C9  ;Port A Data Register, 8-bit, R/W


*****************************************************************
* M-BUS Registers                                               *
*****************************************************************
* The M-bus is a high speed, 2 wire, serial bus that provides   *
* bidirectional serial transmission between on-board devices    *
```

```
* The bus is compatible with the Phillips I2C standard.  See pg        *
* 12-1 of the 5206 user's manual                                       *
************************************************************************

MADR     EQU     MBARx+$01E0  ;M-BUS Address Register, 8-bit, R/W
MFDR     EQU     MBARx+$01E4  ;M-BUS Frequency Divider Register, 8-bit, R/W
MBCR     EQU     MBARx+$01E8  ;M-BUS Control Register, 8-bit, R/W
MBSR     EQU     MBARx+$01EC  ;M-BUS Status Register, 8-bit, R/W
MBDR     EQU     MBARx+$01F0  ;M-BUS Data I/O Register, 8-bit, R/W

**********************
* Finish EQU section *
**********************

************************************************************************
************************************************************************


************************************************************************
************************************************************************

*******************************
* Interrupt vector table*
************************************************************************
* This table defines where the program will go when an interrupt      *
* occurs.  Since these values are loaded as an offset of the VBR,     *
* the label header "VBARx" is used to define the location.  This      *
* was created in the EQU section.  If you change the value of the *
* VBAR make sure it is the same value as the VBARx in the EQU    *
* section                                                             *
*                                                                     *
* For example, in the UART1 configuration I defined the UIVR          *
* interrupt register to have an interrupt vector of 64 ($100 hex)     *
* If the interrupt for UART1 was enabled and an interrupt occured,*
* the processor would load the program counter with the value         *
* found at interrupt vector number 64.  Since I loaded that place     *
* with "UART1_INT", the program will jump to the label UART1_INT.     *
* The simple example code below shows how to create an interrupt  *
* handling subroutine that just returns from the exception            *
*                                                                     *
* UART1_INT
        *
*                 rte                                                 *
*                                                                     *
* Although there are 64-255 possible "user defined interrupt *
* vectors", I only listed 64-100 below. Also, for those I have        *
* not set up, I left them set to $FFFFFFFF.  The end user of this *
* code should insert the proper value for these interrupt vectors     *
*                                                                     *
************************************************************************
V_TABLE
        DC.L   $FFFFFFFF   ;0 (Initial stack Pointer)
        DC.L   $FFFFFFFF   ;1 (Initial program counter)
        DC.L   $FFFFFFFF   ;2 (Access Error)
```

```
DC.L   $FFFFFFFF   ;3 (Address error)
DC.L   $FFFFFFFF   ;4 (Illegal Instruction)
DC.L   $FFFFFFFF   ;5 (reserved)
DC.L   $FFFFFFFF   ;6 (reserved)
DC.L   $FFFFFFFF   ;7 (reserved)
DC.L   $FFFFFFFF   ;8 (Privilege violation)
DC.L   $FFFFFFFF   ;9 (Trace)

DC.L   $FFFFFFFF   ;10(Unimplemented line-a code)
DC.L   $FFFFFFFF   ;11 (Unimplemented line-f code)
DC.L   $FFFFFFFF   ;12 (Debug Interrupt)
DC.L   $FFFFFFFF   ;13 (reserved)
DC.L   $FFFFFFFF   ;14 (Format error)
DC.L   $FFFFFFFF   ;15 (Uninitialized interrupt)
DC.L   $FFFFFFFF   ;16 (reserved)
DC.L   $FFFFFFFF   ;17 (reserved)
DC.L   $FFFFFFFF   ;18 (reserved)
DC.L   $FFFFFFFF   ;19 (reserved)

DC.L   $FFFFFFFF   ;20 (reserved)
DC.L   $FFFFFFFF   ;21 (reserved)
DC.L   $FFFFFFFF   ;22 (reserved)
DC.L   $FFFFFFFF   ;23 (reserved)
DC.L   $FFFFFFFF   ;24 (Spurious interrupt)
DC.L   TIMER1_INT  ;25 (Autovector 1)
DC.L   TIMER2_INT  ;26 (Autovector 2)
DC.L   MBUS_INT    ;27 (Autovector 3)
DC.L   $FFFFFFFF   ;28 (Autovector 4)
DC.L   $FFFFFFFF   ;29 (Autovector 5)

DC.L   $FFFFFFFF   ;30 (Autovector 6)
DC.L   $FFFFFFFF   ;31 (Autovector 7)
DC.L   $FFFFFFFF   ;32 (Trap 0)
DC.L   $FFFFFFFF   ;33 (Trap 1)
DC.L   $FFFFFFFF   ;34 (Trap 2)
DC.L   $FFFFFFFF   ;35 (Trap 3)
DC.L   $FFFFFFFF   ;36 (Trap 4)
DC.L   $FFFFFFFF   ;37 (Trap 5)
DC.L   $FFFFFFFF   ;38 (Trap 6)
DC.L   $FFFFFFFF   ;39 (Trap 7)

DC.L   $FFFFFFFF   ;40 (Trap 8)
DC.L   $FFFFFFFF   ;41 (Trap 9)
DC.L   $FFFFFFFF   ;42 (Trap 10)
DC.L   $FFFFFFFF   ;43 (Trap 11)
DC.L   $FFFFFFFF   ;44 (Trap 12)
DC.L   $FFFFFFFF   ;45 (Trap 13)
DC.L   $FFFFFFFF   ;46 (Trap 14)
DC.L   $FFFFFFFF   ;47 (Trap 15)
DC.L   $FFFFFFFF   ;48 (reserved)
DC.L   $FFFFFFFF   ;49 (reserved)

DC.L   $FFFFFFFF   ;50 (reserved)
DC.L   $FFFFFFFF   ;51 (reserved)
```

```
DC.L   $FFFFFFFF   ;52 (reserved)
DC.L   $FFFFFFFF   ;53 (reserved)
DC.L   $FFFFFFFF   ;54 (reserved)
DC.L   $FFFFFFFF   ;55 (reserved)
DC.L   $FFFFFFFF   ;56 (reserved)
DC.L   $FFFFFFFF   ;57 (reserved)
DC.L   $FFFFFFFF   ;58 (reserved)
DC.L   $FFFFFFFF   ;59 (reserved)

DC.L   $FFFFFFFF   ;60 (reserved)
DC.L   $FFFFFFFF   ;61 (reserved)
DC.L   $FFFFFFFF   ;62 (reserved)
DC.L   $FFFFFFFF   ;63 (reserved)
DC.L   UART1_INT   ;64 (user defined interrupt)
DC.L   UART2_INT   ;65 (user defined interrupt)
DC.L   $FFFFFFFF   ;66 (user defined interrupt)
DC.L   $FFFFFFFF   ;67 (user defined interrupt)
DC.L   $FFFFFFFF   ;68 (user defined interrupt)
DC.L   $FFFFFFFF   ;69 (user defined interrupt)

DC.L   WDOG_INT    ;70 (user defined interrupt)
DC.L   $FFFFFFFF   ;71 (user defined interrupt)
DC.L   $FFFFFFFF   ;72 (user defined interrupt)
DC.L   $FFFFFFFF   ;73 (user defined interrupt)
DC.L   $FFFFFFFF   ;74 (user defined interrupt)
DC.L   $FFFFFFFF   ;75 (user defined interrupt)
DC.L   $FFFFFFFF   ;76 (user defined interrupt)
DC.L   $FFFFFFFF   ;77 (user defined interrupt)
DC.L   $FFFFFFFF   ;78 (user defined interrupt)
DC.L   $FFFFFFFF   ;79 (user defined interrupt)

DC.L   $FFFFFFFF   ;80 (user defined interrupt)
DC.L   $FFFFFFFF   ;81 (user defined interrupt)
DC.L   $FFFFFFFF   ;82 (user defined interrupt)
DC.L   $FFFFFFFF   ;83 (user defined interrupt)
DC.L   $FFFFFFFF   ;84 (user defined interrupt)
DC.L   $FFFFFFFF   ;85 (user defined interrupt)
DC.L   $FFFFFFFF   ;86 (user defined interrupt)
DC.L   $FFFFFFFF   ;87 (user defined interrupt)
DC.L   $FFFFFFFF   ;88 (user defined interrupt)
DC.L   $FFFFFFFF   ;89 (user defined interrupt)

DC.L   $FFFFFFFF   ;90 (user defined interrupt)
DC.L   $FFFFFFFF   ;91 (user defined interrupt)
DC.L   $FFFFFFFF   ;92 (user defined interrupt)
DC.L   $FFFFFFFF   ;93 (user defined interrupt)
DC.L   $FFFFFFFF   ;94 (user defined interrupt)
DC.L   $FFFFFFFF   ;95 (user defined interrupt)
DC.L   $FFFFFFFF   ;96 (user defined interrupt)
DC.L   $FFFFFFFF   ;97 (user defined interrupt)
DC.L   $FFFFFFFF   ;98 (user defined interrupt)
DC.L   $FFFFFFFF   ;99 (user defined interrupt)
DC.L   $FFFFFFFF   ;100 (user defined interrupt)
```

```
*****************************************************************
* The XDEF command is for the DIAB DATA compiler and it lets the        *
* SDS source level debugger where to begin runing the code.             *
*****************************************************************
        XDEF            _main
_main

        move.w          #$2000,D0               ;Reset SR. Interrupts
        move.w          D0,SR                   ;possible at levels 0-7
*****************************************************************
* The lines below are commented out for use on the SBC5206              *
* development system.  If you are using this example without        *
* the use of the development system, uncomment them. (i.e remove *
* the ";" at the begnning of each line.                             *
*****************************************************************


*****************************************************************
* IMPORTANT: Changes if using the 5206 evaluation board.                *
*                                                                       *
* If you are using this code on the 5206 development system,        *
* uncomment the code listed below.  It will overwrite the           *
* configurations to the MBR, VBR, DCAR0, & RAMBARx registers.       *
* See pg 3-4 of the eval board SBC5206 User's manual, Rev 1             *
*                                                                       *
* The Diab Data compiler/assembler has a configuration file that    *
* sets the below listed registers.  This is mainly                  *
* for the SBC5206 evaluation board. See pg 3-4 of the SBC5206    *
* development system users manual                                       *
*****************************************************************
*   Base Register     From      To                                  *
*       VBR             =       $00000000 - $000003FF                       *
*       MBR             =       $10000000 - $100003FF                       *
*       RAMBAR          =       $20000000 - $200001FF                       *
*       DCAR0                   =       $00000000 - $01FFFFFF
        *
*****************************************************************

*************************************************
* Define base addresses of MBAR, & VBR              *
*****************************************************************
* Initialize the base modules. The values were defined earlier in *
* the EQU section. The code below defines where each MBAR & VBR   *
* are located in the memory map.                                        *
*****************************************************************
;       move.l #MBARx,D0        ;define the init location of the
;       movec  D0,MBAR          ;Vector Base register.  See pg.
                                ;7-7,8 of the users manual
*****************************************************************
* The A7 register needs to be used to load the correct value in     *
* the VBR register.  Do not use any other register, only A7.  *
*                                                                       *
* A vector table called V_TABLE is used to store the correct *
```

```
* values for the vector interrupts, see pg 3-7 of the 5206 user's       *
* manual                                                                 *
**************************************************************************
        move.l A7,A6                ;save stack pointer
        move.l #VBRx,A7             ;define the init location of the
        movec  A7,VBR               ;Vector Base register
                                            ;defined earlier in the EQU statements
                                            ;as $2000000
        move.l A6,A7                ;restore stack

        clr.l           D0
        clr.l           D1
        lea.l           VBRx,A0             ;point A0 to the VBRx location
        lea.l           V_TABLE,A1
VT_INIT
        move.l (A1)+,(A0)+          ;point A1 to the vector table
        addi.l  #1,D0
        cmp.l           #100,D0             ;D0 is just a counter to set up
        bne             VT_INIT             ;100 of 256 possible interrupts

********
* CH 4 *
**************************************************************************
* CACHE Initialization          Section 4 (Users manual)                  *
**************************************************************************
*                                                                        *
* The cache uses only 3 registers for configuration: CACR, ACR0,  *
* & ACR1.  The CACR register controls the operation of the cache. *
* The ACR0 & ACR1 registers define what memory space is cacheable.*
* This is accomplished by setting the base and mask bits (pg 4-9      *
* of the 5206 users manual).                                             *
* The registers can be set up to define what memory space is *
* is or is not cacheable (pg 4-10, bit 6 "CM").                          *
**************************************************************************

******************
* ACR registers *
******************
**************************************************************************
* The ACR registers define what areas of memory are cacheable.      *
* Bits 31:24 define the base address and bits 23:16 mask off the     *
* address comparison bits. For example, to define a space in         *
* memory, $01FFFFFF to $03FFFFFF, you would do the following:        *
*                                                                        *
*                       31  30  29  28  27  26  25  24                   *
*                 -------------------------------                        *
*        Bits 31:24  0   0   0   0   0   0   0   1   (base address)  *
*    Bits 23:16  0   0   0   0   0   0   1   1   (mask)              *
*                                                                        *
* The above diagram show that the base address starts at $01xxxxxx*
* and the mask ignores the lower 2 bits and only compares the        *
* upper bits (31:26).  Therefore, the cacheable memory goes up       *
* to $03FFFFFF.                                                          *
* It is also possible to define the same memory space listed above*
```

```
* as non-cacheable.  This is accomplished by setting bit 6, "CM"     *
* in the ACR registers.  The users has a maximum of 2 ACR                *
* registers to use.                                                      *
*************************************************************************
* The cache is set up as follows (see pg 4-6,7,8):                      *
*       1) Cache is enabled (bit 31)                                     *
*       2) CPUSHL is enabled (bit 28)                                                *
*       3) Cache freeze is in normal operation (bit 27)                  *
*       4) Cache is not invalidated (bit 24)                             *
*       5) Burst fetches on non cacheable disabled (bit 10)              *
*       6) Default cache mode enabled (bit 9)                            *
*       7) Buffered writes enabled (bit 8)                               *
*       8) Both read and write capability (bit 5)                        *
*       9) line miss fetches (bits 1,0)                                  *
*************************************************************************

;       move.l #91000323,D0         ;disable cache but set up other
;       movec  D0,CACR              ;functions for demonstration
                                            ;see pg 4-6,7,8 of users manual
                                            ;CACRx defined in EQU section
                                            ;which is $80000321

;       move.l #$0103C020,D0        ;cached memory from $01000000
;       movec  D0,ACR0              ;to $03FFFFFF.  ignore modes,enable
                                            ;ACR0, buffered writes and enable
                                            ;both read and write. see pg 4-9.10
********
* CH 5 *
*************************************************************************
* SRAM Initialization   Section 5 (Users manual)                         *
*************************************************************************
* The SRAM is set up easily since it only has one register to *
* initialize, RAMBAR.  The SRAM base register can be set up on    *
* any 512 byte boundary (pg 5-2).  The lower 8 bits define what   *
* type of data can be used or accessed and bits 31..9             *
* define the location of the base address.  The value of RAMBARx  *
* was defined earlier in the EQU statements section.              *
*                                                                              *
* The SRAM is configured, pg 5-2 of users manual, as follows:     *
*       1)Base address is $20000000 (bits 31:9)                          *
*       2)Allow read and write capabilities (bit 8)                      *
*       3)CPU space, User, and Supervisor code capable (bits 5:1)        *
*       4)Validate the SRAM space.  After loading the SRAM with          *
*         valid data, enable this bit. This is a good design             *
*         practice to do when powering up or resetting the chip          *
*         (bit 0)                                                                *
*************************************************************************
        move.l #RAMBARx+1,D0    ;define the init location of the
        movec  D0,RAMBAR        ;SRAM base register (pg 5-2,3)
                                        ;RAMBARx value was defined in the EQU
                                        ;section as $20000000
******
*CH 7*
*************************************************************************
```

```
* SIM Initialization      Section 7 (Users manual)                    *
*****************************************************************
* The MCF5206 SIM provides a centralized interrupt controller for  *
* the following modules                                                *
*       1)External interrupts                                          *
*       2)Software Watchdog Timer                                      *
*       3)Timer Modules                                                *
*       4)MBUS (I2C) module                                            *
*       5)UART modules                                                 *
*****************************************************************
* Written below is the code that will set up the interrupts        *
* for the 5206.  The interrupt levels and priorities were chosen   *
* by random for demonstrative purposes.  The end user should       *
* define the interrupt level and priorities for their own          *
* applications                                                         *
* The lines that are commented for the SIM are commented out to    *
* enable this code to function on the SBC5206 development system   *
*****************************************************************

*****************
* SIMR register *
*****************************************************************
* The SIMR, pg 7-9, is set up as follows:                          *
*       1)Disable watchdog when FREEZE is asserted (bit 7)         *
*       2)Disable bus monitor when FREEZE is asserted (bit 6)      *
*       3)5206 will negate the /BD signal                          *
*****************************************************************
;       move.b      #%11000000,D0     ;set up the SIMR (pg 7-9)
;       move.b      D0,SIMR           ;

*****************************************************************
* NOTE: The timer & MBUS peripherals cannot provide interrupt      *
* vectors.(see pg 7-5, 2nd paragraph).  Timer & MBUS peripherals   *
* are only autovectored. Interrupt values were chosen              *
* randomly for demonstrative purposes.  The end user should        *
* change these for their own application needs.                    *
*****************************************************************
        move.b      #%10000100,D0 ;set up Timer 1 Interrupt
        move.b      D0,ICR9       ;Level 2 interrupt, Priority 0,
                                  ;Autovector=ON, see pg 7-10
                                  ;avect 24+1=25 see bottom of pg 7-10

        move.b      #%10001001,D0 ;set up Timer 2 Interrupt
        move.b      D0,ICR10      ;Level 2 interrupt, Priority 1,
                                  ;Autovector=ON,avect 24+2=26,
                                  ;see bottom of pg 7-10

        move.b      #%10001010,D0 ;set up MBUS Interrupt
        move.b      D0,ICR11      ;Level 2 interrupt, Priority 2,
                                  ;Autovector=On,AVECT 24+3 = 27
                                  ;see pg 7-10

        move.b      #%00011011,D0 ;set up UART1 Interrupt
        move.b      D0,ICR12      ;Level 6 interrupt, Priority 3,
```

```
                                        ;Autovector=Off

        move.b      #%00001001,D0 ;set up UART2 Interrupt
        move.b      D0,ICR13      ;Level 1 interrupt, Priority 1,
                                  ;Autovector=Off

************************
* IMR register, pg 7-12 *
************************
*****************************************************************************
* No external interrupts enabled.  UART1, UART2 interrupts are     *
* disabled. MBUS, TIMER2 and TIMER2 have interrupts disabled       *
* (i.e masked. See pg 7-12 of the users manual). The SIM           *
* initialized the UARTs, M-BUS for interrupt vectors and the       *
* TIMERS for autovector.  Enable the interrupts for each           *
* individualmodule in the IMR register if you want them enabled.   *
*                                                                           *
* A logic "0" enables the corresponding interrupt and a logic "1"  *
* masks the interrupt. Bits 13:1 are used.                         *
*                                                                           *
* Note: How each module interrupts the processor is defined in     *
*       each module's configuration.                               *
*****************************************************************************

        move.w      #$3FFE,D0       ;set up the IM R (interrupt mask
                                    ;register) for no interrupts
        move.w      D0,IMR          ;see pg 7-12 of the users manual

*******************
* Watchdog timer *
*******************
        move.b      #$00,D0         ;Disable "Watchdog" and bus monitor
        move.b      D0,SYPCR        ;see pg 7-14 of the users manual

        move.b      #70,D0          ;Watchdog interrupt vector # 70
        move.b      D0,SWIVR        ;the processor knows vector 70 is
                                    ;vector offset $118
                                    ;see pg 7-16 of the users manual
                                    ;$70 is a random choice

******
*CH 8*
*****************************************************************************
* CHIP SELECT Initialization          Section 8 (Users manual)              *
*****************************************************************************
* The chip select is set up to control the external memory and     *
* the DRAM. In this example, the chip select is set up for a       *
* single external FLASH device, 29F040.  It has 20 address lines   *
* and an 8 bit data output.  If code size becomes larger than the  *
* FLASH device, a larger device can be used.  Remember that                 *
* the MCF5206 address lines A28:24 are muxed with the CS/WE.       *
* If your code exceeds 16 meg (i.e. 24 bits wide), you will need   *
* to alter you PAR register for correct CS/WE/Address logic. This  *
* is shown on pg 7-17 in the PAR pin assignments table             *
```

```
**********************************************************************
* Chip select 0 (CS0) will be set up as the /CS to the 29F040.       *
* The entire address space (20 address lines) will be dedicated      *
* to the FLASH.  Any additional peripherals that are added should    *
* use the other chip selects and be defined somewhere above the      *
* memory map of the FLASH user code (i.e. address lines 20 and up)*
**********************************************************************
* Chip select 2 is set up for addresses $001F0000 to $001FFFFF       *
* (65K space).  This can be used for an external peripheral.  It     *
* is configured for user data (UD).                                  *
**********************************************************************

**********************************************************************
*NOTE: These lines have been commented out for use on the SBC5206    *
*       development board.                                           *
**********************************************************************

*****************
*Chip select 0 *
*****************
;       move.w      #$0000,D0    ;set base address for Chip Select 0
;       move.w      D0,CSAR0     ;to $00000000.
                                 ;see pg 8-29,30 of the users manual

******************
*CSMR0 register *
**********************************************************************
* This register defines the address mask and what types of           *
* accesses to the defined memory space are enabled.  The CSAR        *
* register defined the base address.  In this example that was       *
* $00000000.  To define a space from $00000000 to $000FFFFF, the     *
* CSMR register needs to mask off the lower 20 bits.  A logic "1"    *
* means: do not use this bit in the comparison.  The table below     *
* shows that once an address call to bit 20 of higher occurs, it     *
* will fall outside the range of Chips Select 0.                     *
*                                                                    *
*       Bits     31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16    *
*        ----------------------------------------------      *
* (CSARx)   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0          *
* (CSMR0 reg)0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1         *
*                                                                    *
* The above table shows that an access to bit 20 or higher is  *
* not defined for chip select 0. See pg 8-30,31 for details.    *
**********************************************************************


;       move.l #$000F0000,D0 ;set the mask register for CS0
;       move.l D0,CSMR0      ;0-19 (i.e. $00000000 to $000FFFFF).
                             ;All access types (SC,SD,UC,UD
                             ;C/I) valid.  see pg 8-30,31
                             ;of the users manual

******************
*CSCR0 register *
```

```
*****************************************************************
* CSCR0 chip select control register is defined as follows (see *
* pg 8-(33-38) in the users manual):                            *
*       1)longest # of wait states until /TA is asserted        *
*         (bits 13-10)                                          *
*       2)bursting enabled (bit 9)                              *
*       3)port size for data is 8 bits (bits 7,6)               *
*       4)Chip select asserted with address (bit 4) pg 8-35     *
*       5)Release address when /CS or /WE negate on writes      *
*         (bit 3) pg 8-36                                       *
*       6)Release address when /CS negates on reads (bit 2) pg 8-37 *
*       7)No writes to this address space (user code) bit 1 pg 8-38 *
*       8)Allow reads to this address space (bit 0) pg 8-38     *
*****************************************************************


;       move.w      #$3F41,D0           ;set up the chip select
;       move.w      D0,CSCR0            ;control register


***************
*Chip select 1*
***************
;       move.w      #$0010,D0           ;set base address for Chip Select 1
;       move.w      D0,CSAR1            ;to $00100000.
                                        ;see pg 8-29,30 of the users manual

;       move.l #$000F001C,D0    ;set the mask register for addresses
;       move.l D0,CSMR1         ;$00100000 to $001FFFFF (65K space)
                                        ;Only valid access is UD (user data)
                                        ;See pg 8-30,31 of the users manual

*****************
*DMCR register *
*****************************************************************
* All other memory spaces not associated with the 8 chip selects *
* or DRAM banks are considered "default memory".  This register  *
* configures the memory space.  This is similar to the CACR      *
* registers.                                                     *
*****************************************************************
;       move.w      #$0000,D0           ;define the control register for
;       move.w      D0,DMCR             ;memory not defined by the chip select
                                        ;select registers.
                                        ;see pg 8-(38-43)of the users manual

*****************************************************************
* The following 2 lines set up the PAR register. See pg 7-16,17, *
* and table 7-9. In this example the register has been set up    *
* enable the pins as follows:                                    *
*               1)output reset on UART2 RTS pin(PAR_7)           *
*               2)set up IRQ7,4,1 inputs      (PAR_6)            *
*               3)set up general purpose I/O  (PAR_5,4)          *
*               4)WE[3:0] are all activated   (PAR_3,2,1,0)      *
*****************************************************************
```

```
;       move.b      #%00000000,D0     ;set up PAR (Pin Assignment Register.)
;       move.b      D0,PAR            ;see pg 7-16,17 of the users manual


*************************************************************************
* INDIVIDUAL MODULE INITIALIZATION        (Sections 9 - 13)
        *
*************************************************************************

***
*9*
*************************************************************************
* Parallel Port Initialization      Section 9 (Users manual)      *
*************************************************************************
* The 5206 has one 8 bit parallel port for general I/O.  The pins       *
* are muxed with the DEBUG pins. This example initialization            *
* code sets up the parallel port.  The port is set up as follows:       *
*                                                                       *
*       1) Bit 7 is configured as an input                              *
*       2) Bits 6:0 are configured as outputs set. All outputs are      *
*         set to logic "1"                                              *
* For proper operation, the PAR, bits 5&4 (pg 7-16,17) needs to         *
* initialized to logic "0"                                              *
*************************************************************************

        move.b      #%01111111,D0     ;Bit 7 is an input, bits 6:0 outputs
        move.b      D0,PADDR          ;see pg 9-2 of the users manual

        move.b      #$7F,D0           ;Initialize the Parallel port Data reg.
        move.b      D0,PADAT          ;outputs to logic "1".
                                      ;see pg 9-2,3 of the users manual

****
*10*
*************************************************************************
* DRAM Initialization  Section 10 (Users manual)                        *
*************************************************************************
* The DRAM controller is glueless and can be set up as:                 *
*       1)2 banks, 128K to 256M byte sizes                              *
*       2)Normal, fast page, and burst mode.  Also supports EDO         *
*       3)Programmable refresh rates                                    *
*                                                                       *
* Each bank is individually programmable except for the refresh         *
* registers (DCCR & DCTR).  These 2 registers apply to both             *
* banks.  The example below sets up a 4meg, normal mode, 512 page,*
* memory bank 0.  The refresh rates are set up for the fastest*
* refresh rates.  Bank 1 is not set up, therefore assumed not to        *
* be used in this example.  The registers initialized are:              *
*       1)DCTR - sets refresh rate                                      *
*       2)DCAR - sets up the base address of the memory bank            *
*       3)DCMR - defines the size of the DRAM memory.  Also defines *
*                what kinds of transfers can occur such as user,        *
*                supervisor, etc.                                       *
```

```
*       4)DCCR - defines port size, page size, page mode, and R/W *
*************************************************************************
*************************************************************************
* DCAR0 register                                                        *
*                                                                       *
* This example initializes the DRAM at address $00700000.  This        *
* was defined earlier in the EQU section with the variable              *
* DRAM0x.  The size of the DRAM is initialized by the DCMR              *
* register.                                                             *
*************************************************************************

*******************
* DCMR Register     *
*************************************************************************
*       The following table defines typical values for common           *
*       DRAM configurations.  Load the values into the DCMR             *
*       register for each size. The following table assumes             *
*       the transfer types (user, supervisor), bits 4:1 are             *
*       left at zero logic "0". Bit 16 of this register is not used,*
*       therefore, a 1 meg or larger DRAM will always have a hex        *
*       "E" in the fifth position.  For example, a 1 meg DRAM will      *
*       have a base address and a size from $0 to $FFFFF. A 4 meg *
*       will have a size from $0 to 3FFFFF.  See pg 10-59 of the        *
*       5206 users manual.                                              *
*                                                                       *
*                                                                       *
*       DRAM SIZE               DCMR value                              *
*       128K                    $00000000                               *
*       256K                    $00020000                               *
*       512K                    $00060000                               *
*       1 MEG                   $000E0000                               *
*       2 MEG                   $001E0000                               *
*       4 MEG                   $003E0000                               *
*       8 MEG                   $007E0000                               *
*       16 MEG                  $00FE0000                               *
*                                                                       *
*       Example code for setting up 8 meg DRAM for bank 0:              *
*               move.l #$007E0000,D0                                    *
*               move.l D0,DCMR0                                         *
*                                                                       *
* NOTE: Base address can be located anywhere as long as it does         *
*               not conflict with other memory spaces such as code      *
*               memory in EPROM.  It is suggested that DRAM should be *
*               located above the 128K memory space (i.e. Bit 17).      *
*************************************************************************

*****************
* DCTR register *
*************************************************************************
*                                                                       *
* The DCTR register is set up for the fastest rates.  Make sure         *
* the DRAM you use matches the timing requirements or change            *
* some of the values in the DCTR register accordingly. No specific*
* DRAM was chosen for this set up.  The designer should set up          *
```

```
* this section to match the specifications of their own DRAM*
*                                                                    *
* The DCTR register is initialized as follows:                       *
*                                                                    *
*       1)Don't drive address signals during alt. master (bit 15)    *
*       2)Standard DRAM, no EDO used (bit 14)                        *
*       3)/RAS asserts 1 sys_clk before /CAS (bit 12)                *
*       4)/RAS negates 1.5 sys_clks after /CAS asserts (bits 10,9)   *
*       5)/RAS precharges for 1.5 sys_clks   (bits 6,5)              *
*       6)/CAS asserts for 1.5 sys_clks (EDO=0) (bit 3)              *
*       7)/CAS negates for 0.5 sys)clks (EDO=0) (bit 1)              *
*       8)/CAS asserts 1 sys_clk before asserting /RAS (bit 0)       *
**********************************************************************

;       move.w      #$0000,D0     ;set up the DCTR1.  Set for fastest
;       move.w      D0,DCTR       ;cycle timing.  see pg 10-(52-58) of
                                  ;the users manual

******************
* DCCR0 register *
**********************************************************************
* This register defines the port width, page size, transfer mode     *
* and R/W enables.  See pg 10-60 of the users manual.                *
*                                                                    *
* This example is configured as follows:                             *
*                                                                    *
*       1)32 bit data port size (bits 7,6)                           *
*       2)512 byte page size (bits 5,4)                              *
*       3)Normal mode (bits 3,2)                                     *
*       4)Read and Write capabilities are enabled (bits 1,0)         *
**********************************************************************

;       move.l   #$03,D0              ;set up the DCCR0 for 32 bit port size,
;       move.l   D0,DCCR0  ;512 byte page, and Normal mode,
                                  ;with both read & write capability
                                  ;see pg 10-60,61 of the users manual

;       move.w      #DRAM0x,D0 ;set DRAM0 base address (bits 31-17);
;       move.w      D0,DCAR0      ;Lower 17 bits are default zeros.
                                  ;See pg 10-58 of the users manual
                                  ;DRAM0x was defined in the EQU section
                                  ;as $00700000.

;       move.w      #!32,D0       ;set up the DCRR0. set to refresh
;       move.w      D0,DCRR0      ;every 512 clocks.  32*16*(1/clk freq)
                                  ;see pg 10-51,52 of the users manual
                                  ;!32 is decimal 32


;       move.l   #$003E0000,D0     ;set up the DCMR0 (DRAM0 mask register)
;       move.l   D0,DCMR0  ;4Meg ram size
                                  ;See pg 10-59 of the users manual

****
```

```
*11*
************************************************************************
* UART Initialization   Section 11 (Users manual)                      *
************************************************************************
* The dual UARTs on the 5206 are initialized for baud rate,  *
* bits/character, parity, etc.  Some important points are:            *
*                                                                      *
*       1)To initialize the UARTs you MUST reset each UART in the      *
*         UCR register.                                                *
*       2)Setting up each UARTs UMR1&2 registers can be confusing.     *
*         For proper initialization, reset the mode register pointer*
*         in the UCR register (MISCX=001, pg 11-25). This sets the     *
*         pointer at UMR1. Now, configure the UMR1 register.  The      *
*         pointer will automatically increment to UMR2.       Now write *
*         to UMR2 (which is the same address as UMR1. The pointer      *
*         is internal to the processor and cannot be seen by the       *
*         programmer.                                                  *
*                                                                      *
*         This is also explained in the paragraph describing UMR1      *
*         on pg 11-17.                                                 *
*                                                                      *
*                                                                      *
* Also, see pg 11-34 for initialization procedures                     *
************************************************************************

;NOTE: UART Interrupts were disabled earlier in the IMR initialization
;       in the SIM section

*********
* UART1 *
*********
************************************************************************
* The code below initializes the UART1 as described on pg 11-25    *
************************************************************************
        move.b      #%00100000,D0     ;UART1 receiver reset
        move.b      D0,UCR1           ;see pg 11-24,25 of the users manual

        move.b      #%00110000,D0     ;UART1 transmitter reset
        move.b      D0,UCR1           ;see pg 11-24,25 of the users manual

        move.b      #%00010000,D0     ;UART1 Mode register reset
        move.b      D0,UCR1           ;see pg 11-24,25 of the users manual
                                      ;this sets the pointer to UMR1

********
* UMR1 *
************************************************************************
*                                                                      *
* UMR1 is set up as described below:                                   *
*       1) Receiver request to send is automatically negated           *
*               when FIFO has an empty position (BIT 7,UMR1). When     *
*               the UART is acting as a receiver, the /RTS acts as     *
*               a /CTS to the external UART sending the data.          *
*       2) Receiver interrupts when FIFO is full, not a single         *
```

```
*                received byte (BIT 6, UMR1)                              *
*        3) Error occurs in BLOCK mode (bit 5)                            *
*        3) No parity (BIT 4,3,2, UMR1)                                   *
*        4) 8 bits per character (BIT 1,0 UMR1)                           *
*                                                                              *
* Note that UMR11 and UMR21 both point to the same address in the       *
* EQU section.  Resetting the Mode register pointer (UCR1          *
* register) sets the pointer to UMR1.  After writing to UMR1       *
* the pointer points to UMR2.  You should initialize UMR2          *
* immediately after initializing UMR1.                                 *
*                                                                              *
*                                                                              *
* See pgs 11-(17-21)of the users manual.                                *
*******************************************************************

********
*(UMR2)*
*******************************************************************
* UMR2 for UART1 is set up as follows:                                  *
*        1) Normal Mode (NO echo or loopback (BIT 7,6,UMR2)             *
*        2) Transmitter ready-to send /RTS has no effect,          *
*                (BIT 5, UMR2)                                                 *
*        3) Transmitter /CTS assert has no effect                       *
*                (BIT 4, UMR2)                                                 *
*        4)1 stop but is sent (BIT 3,2,1,0 UMR2)                        *
*                                                                              *
* See pg 11-(19-21) of the users manual                                 *

*******************************************************************

        move.b      #$F3,D0     ;set up the UMR1 (UART1 mode register)
        move.b      D0,UMR11    ;see pg 11-17,18,19 of the users manual
                                ;After writing to this register, the mode
                                ;pointer points to UMR2 which I have defined
                                ;as UMR21, in the EQU section.  Note that
                                ;this has the same address as UMR11. See next
                                ;command below

        move.b      #$07,D0     ;set up the UMR2 (UART1 mode register)
        move.b      D0,UMR21    ;see pg 11-19,20 of the users manual

        move.b      #$DD,D0     ;select timer mode as the Tx/Rx clocking
        move.b      D0,UCSR1    ;see pg 11-24 of the users manual

        move.b      #$00,D0     ;Disable interrupts for the COS&/CTS
        move.b      D0,UACR1    ;see pg 11-29 of the users manual

        move.b      #$00,D0     ;No interrupts enabled
        move.b      D0,UIMR1    ;see pg 11-31 of the users manual

        move.b      #64,D0      ;Interrupt vector set to 64 ($100)
        move.b      D0,UIVR1    ;see pg 11-31 of the users manual
                                ;vector 64 was a random choice.
                                ;64 decimal is converted by the processor
```

```
                                    ; to the vector offset hex value of $100
*************************************************************************
* UART1 (UBG11 & UBG21)         pg 11-31
        *
*************************************************************************
* Baud rate is set by taking the system clock, dividing by              *
* 32, then divide that value by the concatenation of the decimal        *
* value in UBG1 & UBG2.                                                 *
* This example assumes that the clock select is in Timer mode           *
* as described on pg 11-24 in the USCR register                         *
* Below is an example and some approximate common baud rates:           *
*                                                                       *
* Example: 9600 baud rate (25mhz system clock)                          *
*           9600 = 25mhz * (1/32) * (1/81) where     UBG1=0             *
*                                                    & UBG2=81 decimal  *
*                                                        =51 hex)       *
*                                                                       *
*       baud rate               UBG1(hex)   UBG2(hex) (25MHZ sys clock) *
*       300                     0A          2C                          *
*       1200                    02          8B                          *
*       2400                    01          45                          *
*       4800                    00          A2                          *
*       9600                    00          51                          *
*       19200                   00          28                          *
*       28800                   00          1B                          *
*       33600                   00          17                          *
*       56000                   00          0E                          *
*       128000                  00          06                          *
*       390625                  00          02 (maximum)                *
*************************************************************************
*       baud rate               UBG1(hex)   UBG2(hex) (33MHZ sys clock) *
*       300                     0D          6E                          *
*       1200                    03          5B                          *
*       2400                    01          AE                          *
*       4800                    00          D7                          *
*       9600                    00          6B                          *
*       19200                   00          36                          *
*       28800                   00          24                          *
*       33600                   00          1F                          *
*       56000                   00          12                          *
*       128000                  00          08                          *
*       515625                  00          02 (maximum)                *
*************************************************************************
        move.b      #$00,D0             ;Set baud rate to 9,600 (25 mhz clk)
        move.b      D0,UBG11            ;see pg 11-31 of the users manual
        move.b      #$51,D0             ;concatenated HEX value $0051
        move.b      D0,UBG21            ;



*********
* UART2 *
*********
*************************************************************************
```

```
* The code below initializes the UART2 as described on pg 11-25    *
*********************************************************************
        move.b      #%00100000,D0       ;UART2 receiver reset
        move.b      D0,UCR2             ;see pg 11-24,25 of the users manual

        move.b      #%00110000,D0       ;UART2 transmitter reset
        move.b      D0,UCR2             ;see pg 11-24,25 of the users manual

        move.b      #%00010000,D0       ;UART2 Mode register reset
        move.b      D0,UCR2             ;see pg 11-24,25 of the users manual
                                        ;this sets the pointer to UMR1

********
* UMR1 *
*********************************************************************
*                                                                   *
* UMR1 for UART 2 is set up as described below:                     *
*       1) Receiver request to send is automatically negated        *
*               when FIFO has an empty position (BIT 7,UMR1). When  *
*               the UART is acting as a receiver, the /RTS acts as  *
*               a /CTS to the external UART sending the data.       *
*       2) Receiver interrupts when FIFO is full, not a single      *
*               received byte (BIT 6, UMR1)                         *
*       3) Error occurs in BLOCK mode (bit 5)                       *
*       3) No parity (BIT 4,3,2, UMR1)                              *
*       4) 8 bits per character (BIT 1,0 UMR1)                      *
*                                                                   *
* Note that UMR1 and UMR2 both point to the same address in the     *
* EQU section.  Resetting the Mode register pointer (UCR1           *
* (register) sets the pointer to UMR1.                              *
*                                                                   *
* See pgs 11-(17-21)of the users manual.                            *
*********************************************************************

********
*(UMR2)*
*********************************************************************
* UMR2 for UART2 is set up as follows:                              *
*       1) Normal Mode (NO echo or loopback (BIT 7,6,UMR2)          *
*       2) Transmitter ready-to send /RTS has no effect,            *
*               (BIT 5, UMR2)                                       *
*       3) Transmitter /CTS assert has no effect                    *
*               (BIT 4, UMR2)                                       *
*       4)1 stop but is sent (BIT 3,2,1,0 UMR2)                     *
*                                                                   *
* See pg 11-(19-21) of the users manual                             *

*********************************************************************

        move.b      #$F3,D0     ;set up the UMR1 (UART1 mode register)
        move.b      D0,UMR12    ;see pg 11-17,18,19 of the users manual

        move.b      #$07,D0     ;set up the UMR2 (UART1 mode register)
        move.b      D0,UMR22    ;see pg 11-19,20 of the users manual
```

```
        move.b      #$DD,D0     ;select timer mode as the Tx/Rx clocking
        move.b      D0,UCSR2    ;see pg 11-24 of the users manual

        move.b      #$00,D0     ;Disable interrupts for the COS&/CTS
        move.b      D0,UACR2    ;see pg 11-29 of the users manual

        move.b      #$02,D0     ;Interrupt only when FIFO is full
        move.b      D0,UIMR2    ;see pg 11-31 of the users manual

        move.b      #65,D0      ;Interrupt vector set to 65
        move.b      D0,UIVR2    ;see pg 11-31 of the users manual
                                ;65 was a random choice.
                                ;65 decimal is converted by the processor
                                ; to the vector offset hex value of $104

*************************************************************************
* UART2 (UBG12 & UBG22)       pg 11-31
        *
*************************************************************************
* Baud rate is set by taking the system clock, dividing by          *
* 32, then divide that value by the concatenation of the decimal    *
* value in UBG1 & UBG2.                                                 *
* This example assumes that the clock select is in Timer mode       *
* as described on pg 11-24 in the USCR register                         *
* Below is an example and some approximate common baud rates:       *
*                                                                       *
* Example: 9600 baud rate (25mhz system clock)                          *
*           9600 = 25mhz * (1/32) * (1/81) where    UBG1=0              *
*                                                   & UBG2=81 decimal   *
*                                                       =51 hex)        *
*                                                                       *
*       baud rate           UBG1(hex)   UBG2(hex) (25MHZ sys clock)     *
*       300                 0A          2C                              *
*       1200                02          8B                              *
*       2400                01          45                              *
*       4800                00          A2                              *
*       9600                00          51                              *
*       19200               00          28                              *
*       28800               00          1B                              *
*       33600               00          17                              *
*       56000               00          0E                              *
*       128000              00          06                              *
*       390625              00          02 (maximum)                    *
*************************************************************************
*       baud rate           UBG1(hex)   UBG2(hex) (33MHZ sys clock)     *
*       300                 0D          6E                              *
*       1200                03          5B                              *
*       2400                01          AE                              *
*       4800                00          D7                              *
*       9600                00          6B                              *
*       19200               00          36                              *
*       28800               00          24                              *
*       33600               00          1F                              *
```

```
*       56000           00          12                                  *
*       128000          00          08                                  *
*       515625          00          02 (maximum)                *
*************************************************************************
        move.b      #$00,D0         ;Set baud rate to 33,600 (25 mhz clk)
        move.b      D0,UBG12        ;see pg 11-31 of the users manual
        move.b      #$17,D0         ;HEX value $0017
        move.b      D0,UBG22        ;

****
*12*
*************************************************************************
* M-BUS Registers                                                       *
*************************************************************************
* The M-bus is a high speed, 2 wire, serial bus that provides*
* bidirectional serial transmission between on-board devices  *
* The bus is compatible with the Phillips I2C standard.  Maximum   *
* operating speed is 100K bps.                                    *
*                                                                       *
* See pg 12-1 of the users manual                                 *
*************************************************************************
* In this example the M_BUS will be initialized with the          *
* following specifications:                                       *
*       1) Address $1A (hex) wakes up the receiver (MADR).  The   *
*         value "$1A" was chosen randomly.                        *
*       2) Transmit speed is set to 56K baud (MFDR).  See pg 12-7 *
*       3) MBUS is set as master, transmitting, no interrupts,    *
*               no transmission of acknowledge bit, no repeat     *
*               start (MBCR).                                     *
*************************************************************************

*****************
* MFDR Register *
*************************************************************************
* M-BUS baud rate     pg 12-6,7                                         *
*                                                                       *
* Baud rate is set by taking the system clock and dividing by *
* the prescaler values as defined on pg 12-7 of the users manual.     *
* This value is created in the MFDR register.                     *
* Below is an example and a table of possible baud rates that *
* have been approximated.                                         *
*                                                                       *
* Example: 128K baud rate (25mhz system clock)                    *
*           128K = 25mhz * 1/192 (Hex 31)value stored in MFDR   *
*                                       points to the value of 192 as *
*                                       shown on pg 12-7          *
*************************************************************************
*       baud rate           MFDR(hex)   dec divide value        (25mhz clk)   *
*       12200               3F          2048                                  *
*       24400               3B          1024                                  *
*       32500               39          768                                   *
*       48800               37          512                                   *
*       56000               36          448                                   *
*       78000               34          320                                   *
```

```
*       97600                   33              256                                     *
*****************************************************************************
*****************************************************************************
*       baud rate               MFDR(hex)       dec divide value        (33mhz clk)     *
*       8600                    1F              3840                                    *
*       25780                   3C              1280                                    *
*       36830                   3A              896                                     *
*       51560                   38              640                                     *
*       64450                   37              512                                     *
*       73660                   36              448                                     *
*       85950                   35              384                                     *
*****************************************************************************

******************
* MADR register *
******************
*****************************************************************************
* The MADR register, pg 12-6, sets up the address that "wakes-up" *
* the M-Bus.  If a valid 8 bit value is sent, the M-bus will      *
* respond when enabled as a slave                                         *
*****************************************************************************
******************
* MBCR register *
******************
*****************************************************************************
* The MBCR register, pg 12-8,is set up as follows:                *
*       1)M-bus is enabled (bit 7)                                        *
*       2)Interrupts form the M-bus are disabled (bit 6)          *
*       3)M-bus is set up as a master (bit 5)                     *
*       4)Set to transmit mode (bit 4)                            *
*       5)No acknowledge signal is generated (bit 3)              *
*       6)No repeat start generated (bit 2)                               *
*****************************************************************************

        move.b          #$1A,D0         ;$1A is the address that wakes up the M-BUS
        move.b          D0,MADR         ;see pg 12-3 & 12-6 of the users manual

        move.b          #$36,D0         ;Set baud rate to 56,000 (25 mhz)
        move.b          D0,MFDR         ;see pg 12-6,7  of the users manual

        move.b          #%10111100,D0   ;Setup the M-BUS control register
        move.b          D0,MBCR         ;see pg 12-8 of the users manual

*****************************************************************************
* Other M-BUS registers used such as status and data register     *
*                                                                           *
* MBSR          ;M-BUS Status Register, 8-bit, R/W                        *
* MBDR      ;M-BUS Data I/O Register, 8-bit, R/W                  *
*****************************************************************************


****
*13*
*****************************************************************************
```

```
* Timer  Initialization          Section 13 of the users manual          *
*****************************************************************
* The 5206 has two timers.  They can be free running or count to  *
* a value and reset.  The following examples set up the timers:         *
*                                                                       *
* Timer 1 will count to $AFAF, toggle its output, and reset back        *
* to $0000.  This will continue infinitely until the timer is     *
* disabled or a reset occurs.  No interrupts are set.     Prescale      *
* is set at 256 and the system clock is divided by 16.                  *
* Therefore resolution is (16*(256))/25mhz = 163.84us.  Time out  *
* period is (16*256*44976)/25mhz = 7.369s. ($0 - $AFAF = 44976          *
* decimal)                                                              *
*                                                                       *
* Timer 2 will be free-running and send out a logic pulse               *
* every time it compares the count value in the TRR register.           *
* value, which for now, is randomly chosen as $1234.                    *
* Prescale is set at 127 with the sys_clock initially             *
* divided by 16 (by setting bits 2&1 of the TMR register to 10          *
* therefore, resolution is (16*(127))/25mhz = 81.28us.                  *
* Interrupts are NOT enabled.                                           *
*****************************************************************
* NOTE: The timers were initialized in the SIM to have interrupt        *
* values.  The examples below have the interrupts disabled              *
* The initialization in the SIM configuration was for reference.        *
* See the SIM initialization to see how the autovectors have been       *
* set up. The Timers CANNOT provide interrupt vectors, only             *
* autovectors.                                                          *
*                                                                       *
* See the SIM initialization code and also pg. 7-10 for setting         *
* up the ICRs                                                           *
*****************************************************************

***********
* Timer 1 *
***********

*****************
* TMR register *
*****************************************************************
* Bits 15:8       sets the prescale          to 256 ($FF)               *
* Bits 7:6 set for no interrupt ("00")                                  *
* Bits 5:4 sets output mode for "toggle".  No interrupts("10")          *
* Bits 3 set for "restart" ("1")                                        *
* Bits 2:1 set the clocking source to system clock/16 ("10")   *
* Bits 0 enables/disables the timer       ("0")                         *
*****************************************************************

        move.w      #$FF2C,D0   ;Setup the Timer mode register (TMR1)
        move.w      D0,TMR1     ;see pg 13-4 of the users manual
                                ;Bit 1 is set to 0 to disable the timer
                                ;See the Timer test code at the end of
                                ;this file.
        move.w      #$0000,D0   ;writing to the timer counter with
        move.w      D0,TCN1     ;any value resets it to zero
```

```
*****************
* TRR1 register *
*****************************************************************
* The TRR register is set to $AFAF.  The timer will count up to      *
* this value (TCN = TRR), toggle the "TOUT" pin, and reset the       *
* TCN to $0000. See pg 13-4,5                                                *
*****************************************************************

        move.w      #$AFAF,D0   ;Setup the Timer reference register (TRR1)
        move.w      D0,TRR1     ;see pg 13-5 of the users manual

*****************************************************************
* Other registers used for TIMER 1                                   *
*                                                                            *
* TCR1      ;TIMER1 Capture Register, 16-bit, R                  *
* TER1      ;TIMER1 Event Register, 8-bit, R/W                   *
*****************************************************************

***********
* Timer 2 *
***********

*****************
* TMR2 register *
*****************
*****************************************************************
* Bits 15:8      set the prescale to 127 ($7F)                       *
* Bits 7:6 set the capture mode and interrupt   ("00")          *
* Bits 5:4 set the output mode for "pulse" and no interrupt ("00")*
* Bits 3 set for free-running    ("0")                               *
* Bits 2:1 set the clocking source to clk/16 ("10")                  *
* Bits 1 enables the timer       ("0")                               *
*****************************************************************
        move.w      #$7F04,D0   ;Setup the Timer mode register (TMR2)
        move.w      D0,TMR2     ;see pg 13-4 of the users manual

        move.w      #$1234,D0   ;Set the Timer reference to $1234
        move.w      D0,TRR2     ;see pg 13-5 of the users manual

        move.w      #$0000,D0   ;writing to the timer counter with
        move.w      D0,TCN2     ;any value resets it to zero
*****************************************************************
* 0ther registers used                                               *
*                                                                            *
* TCR2      ;TIMER2 Capture Register, 16-bit, R                  *
* TER2      ;TIMER2 Event Register, 8-bit, R/W                   *
*****************************************************************

*****************************************************************
*                 END MODULE INITIALIZATION                          *
*****************************************************************

*****************************************************************
```

```
*  MAIN code.                                                                  *
*******************************************************************************
*                                                                             *
* Written below are some code segments to show and explain how   *
* some of the modules of the 5206 are used and configured.  A    *
* brief explanation of each is as follows:                       *
*                                                                             *
*1) TIMER1 code segment                                                       *
*       This piece of code counts to $AFAF 5 times.  Everytime it   *
*       reaches $AFAF is sets the output reference bit (and pin).    *
*       The counter resets after reaching $AFAF and begins counting        *
*       again.  Once the counter reaches 2,the code segment finishes*
*       and runs the UART TRANSMIT code segment.                             *
*2) UART TRANSMIT        code segment                                        *
        *
*       This piece of code transmits a message to a terminal such    *
*       as WIN95 Hyperterm.  It sends the message "Welcome to              *
*       Motorola's COLDFIRE!!!"     After completing this section, the     *
*       Program moves on to the UART RECEIVE code segment.                 *
*3) UART RECEIVE code segment                                                *
*       This piece of code recieves characters from a terminal such  *
*       as WIN95 Hyperterm.  It polls the RxRDY bit for a character        *
*       and stores it into memory.  After receiving 26 characters    *
*       it moves on the the UART INTERRUPT WITH INTERRUPTS code             *
*       segment. You must send 26 characters to exit this part of    *
*       the program otherwise it will continue to loop.                    *
*4) UART RECEIVE WITH INTERRUPTS code segment                                *
                                        *
*       This piece of code recieves characters from a terminal such  *
*       as WIN95 Hyperterm.  It uses interrupts to service the       *
*       storing of the character into memory.  It counts up to 10    *
*       characters and stores it into memory.  After receiving 10    *
*       characters the program halts by executing the HALT command.        *
*       You must send 10 characters to exit this part of the         *
*       program.                                                            *
*******************************************************************************
        XDEF  _test_code
_test_code

        lea.l           RAMBARx+$200,A7 ;Setup the stack pointer (A7)
                                        ;to point to the top of the
                                        ;SRAM.  SRAM will be used as
                                        ;part of the stack.
********************
* TIMER1 test code *
*******************************************************************************
* This simple piece of code enables TIMER1 and increments a      *
* a counter each time it toggles its output.  (See the TIMER1    *
* configuration code).  The code monitors bit 1 of the TER       *
* register (pg 13-6 of the users manual).  When the "ref" bit of *
* the TER register is set the counter increment until it reaches *
* 2 decimal.  Then the code will jump out of the loop, disable   *
* TIMER1 and go on to the next piece of code.  This creates a    *
* delay of approximatly 14.74 seconds.                                *
```

```
*                                                                       *
* INFO: You must write a "1" to the REF bit, bit 1, of the TER        *
*       register to clear it.                                          *
************************************************************************
T1_TST
        clr.l       D0          ;clear D3-D0 data registers
        clr.l       D1
        clr.l       D2
        clr.l       D3

        move.w      #$0000,D0   ;A write of any value to the TCN resets
        move.w      D0,TCN1     ;it to all zeros.  See pg 13-5 of users
                                ;manual

        move.b      #$03,D1     ;reset the REF & CAP bit in the TER
        move.b      D1,TER1     ;by writing a "1" see pg 13-6

        move.w      TMR1,D0     ;these 3 lines enable the timer
        bset        #0,D0       ;by setting bit 1 of the TMR1
        move.w      D0,TMR1     ;see pg 13-4 of the users manual

T1_LP
        move.b      TER1,D1     ;load the Timer Event register into D1
        btst        #1,D1       ;Does TRR1=TCN1.  (i.e. 'ref", bit 1
                                ;is set?  TRR1 was defined as $AFAF
                                ;in the Timer1 initialization
        beq         T1_LP       ;If("0")then continue looping. Go to next
                                ;command below when = "1"

        addi.l  #1,D2           ;count from 0 to 5
        cmp.l       #2,D2       ;does D2 = 2 yet
        beq         T1_FIN      ;Finish if equal to 5

        move.b      #$02,D1     ;reset the ref bit in the TER
        move.b      D1,TER1     ;by writing a "1" see pg 13-6

        jmp         T1_LP       ;loop until D2 = 5

T1_FIN
        move.w      TMR1,D0     ;these 3 lines disable TIMER1
        bclr        #0,D0       ;by clearing bit 0 of the TMR1
        move.w      D0,TMR1     ;pg 13-4

        move.b      #$02,D1     ;reset the ref bit in the TER
        move.b      D1,TER1     ;pg 13-6

        move.w      #$0000,D3   ;reset the counter to zero's
        move.w      D3,TCN1     ;pg 13-5

****************************
* UART1 transmit test code *
************************************************************************
* The code below prints out a message out of UART 1.  The message *
* is "Welcome to Motorola COLDFIRE". The code keeps sending out        *
```

```
* the UART until the number "0" is recognized at the end of the       *
* character string.                                                    *
*                                                                      *
* If you are using the SDS debugger, you can use one computer to      *
* check out this code.  Connect a 9 pin serial cable form the  *
* 5206 board (connector J6).  Use Win95's Hyperterminal with the *
* following settings:                                                  *
*       9600 baud, 8 data bits,1 stop bit, no parity,                  *
*       Hardware flow control=none                                     *
*                                                                      *
* Then run this program on the 5206 board.  The message               *
* "Welcome to Motorola's COLDFIRE" should appear on the Hyper-         *
* terminal screen.                                                     *
* Make sure to turn of 16550 UARTs are disabled.  This can be         *
* done in the WIN95 Hyperterminal under advanced settings.             *
************************************************************************

U1_T
        lea.l       UART_T,A1       ;set the address pointer to the
                                    ;beginning of the string

        clr.l       D0              ;clear the registers
        clr.l       D1

        move.b      #%00000100,D1   ;enable the transmitter (pg 11-25,26)
        move.b      D1,UCR1         ;and disable receiver
LOOP1
        move.b      (A1)+,D0        ;load the character into D0
                                    ;from the "MOT" ascii line

        cmp.l       #$30,D0         ;is the character an acsii "0"
        beq         URT_FIN         ;if so, jump out of loop

        move.b      D0,UTB1         ;load the value in the transmit
                                    ;buffer.  Loading the UTB1
                                    ;begins the transmission

WAIT    move.b      USR1,D1
        btst        #$03,D1         ;test bit 3 (transmitter empty)
        beq         WAIT

        jmp         LOOP1

URT_FIN

************************************
* End UART1 transmit test code      *
************************************

*******************************
*  UART1 receive test code    *
************************************************************************
* The code below receives text data from a terminal.  The best way*
* to run this code is to create a text file of the alphabet and    *
```

```
* send it to the 5206 board                                              *
* This code pools the RxRDY bit to see if a character has been           *
* received.  If one has, it gets stored into memory by the A1 *
* address pointer and the pointer is incremented                         *
* A counter counts for 26 charcaters.  When the counter, D2,             *
* reaches 25 (i.e.0-25=26)the routine leaves the loop and ends           *
* this part of the UART receive test code.                               *
**************************************************************************
U1_R
        lea.l           UART_R,A1       ;set the address pointer to the
                                        ;beginning of the string

        clr.l           D0              ;clear the registers
        clr.l           D1
        clr.l           D2

        move.b          #%00001001,D1   ;enable the receiver(pg 11-25,26)
        move.b          D1,UCR1         ;and disable transmitter

RWAIT
        move.b          USR1,D1

        btst            #$0,D1          ;has a character been received?
        nop
        beq             RWAIT                   ;if "0", then loop to RWAIT

        move.b          URB1,D0

        move.b          D0,(A1)+        ;load the character into D0
                                        ;and store in the storage at
                                        ;UART_R
        addi.l  #1,D2               ;count up to 26 (0-25)
        cmp.l           #25,D2
        beq             UR_FIN
        jmp             RWAIT

UR_FIN
        nop

************************************************
*  UART1 receive test code (Interrupts Enabled)       *
**************************************************************************
* This code does exactly what the UART1 receive code does except *
* the use of interrupts is implemented.  The processor interrupts        *
* when the URT1 receives a character.  The interrupt routine             *
* called "URT1_INT" simply stores the value into a memory block          *
* called "URT_R_INT".  After 10 characters are received, the code        *
* disables interrupts and stops with a HALT command.                     *
*                                                                        *
* The interrupt vectors were defined in the SIM section and URT1         *
* was assigned vector #64.  In the vector table, vector number           *
* 64 points to URT1_INT                                                  *
*                                                                        *
**************************************************************************
```

```
        lea.l       U1_RINT,A0      ;set the address pointer to the
                                    ;beginning of the string

        clr.l       D0              ;clear the registers
        clr.l       D1              ;
        clr.l       D2              ;


        move.b      #%00100000,D0   ;UART1 receiver reset
        move.b      D0,UCR1         ;see pg 11-24,25 of the users manual

        move.b      #%00001001,D0   ;enable the receiver(pg 11-25,26)
        move.b      D0,UCR1         ;and disable transmitter


        move.b      #%00010000,D0   ;UART1 Mode register reset
        move.b      D0,UCR1         ;see pg 11-24,25 of the users manual
                                    ;this sets the pointer to UMR1
**********************************
* Set up UMR1 for interrupts on *
* RxRDY (bit 6)                 *
**********************************
        move.b      #$B3,D0         ;set up the UMR1 (UART1 mode register)
        move.b      D0,UMR11        ;see pg 11-17,18,19 of the users manual
                                    ;After writing to this register, the mode
                                    ;pointer points to UMR2 which I have
defined

                                    ;as UMR21, in the EQU section.  See next
                                    ;command below

        move.b      #$07,D0         ;set up the UMR2 (UART1 mode register)
        move.b      D0,UMR21        ;see pg 11-19,20 of the users manual

        move.w      IMR,D0          ;enable interrupts for UART1 in the SIM
        bclr        #12,D0          ;see pg 7-12 of users manual
        move.w      D0,IMR          ;I used 3 lines of code to retain
                                    ;original settings in the IMR
                                    ;pg 7-12

        move.b      #$02,D0         ;set the receiver to interrupt on
        move.b      D0,UIMR1        ;RxRDY.  This is bit 2 of the UIMR1


**********************************************************************
* Below is a loop that will continue until D2=10.  The when a        *
* character is received, an interrupt will occur and load the    *
* character into the storage bytes located at "URT1_R_INT".          *
* The interrupt will return to the loop below.  When D2=10 the       *
* processor will leave the loop, disable UART1 interrupts and halt*
**********************************************************************
        clr.l       D1
        clr.l       D0
UNTIL_10
```

```
        move.b      USR1,D1
        move.w      IPR,D1

        cmp.l       #10,D2
        bne         UNTIL_10            ;loop until D2=10

        move.w      IMR,D0              ;disable interrupts for UART1
        bset        #12,D0              ;see pg 7-12 of users manual
        move.w      D0,IMR

        halt                            ;stop all processing
*******************************
* Interrupt handler routines *
*******************************
UART1_INT
        move.b      URB1,D0             ;load the received character
                                        ;into the D0 register
        move.b      D0,(A0)+            ;load the character into D0
                                        ;and store in the storage at
                                        ;URT1_RINT
        addi.l  #1,D2                   ;count up to 10 (0-9)
        rte

UART2_INT
        rte

TIMER1_INT
        rte

TIMER2_INT
        rte

MBUS_INT
        rte

WDOG_INT
        move.b      $55,D0
        move.b      D0,SWSR

        move.b      $AA,D0
        move.b      D0,SWSR
        rte


****************************************
* Definitions and reserved memory bytes *
****************************************
UART_T
        .ascii   "Welcome to Motorola's COLDFIRE!!0"

UART_R
        DS.B        26      ;Storage for 26 characters (alphabet)
                            ;from a terminal
U1_RINT
```

```
DS.B    10      ;Storage for 10 characters (alphabet)
                ;from a terminal
```