```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                                              @
@ boot.s                                                                       @
@                                                                              @
@ Bootloader for the EE52 ARM VoIP phone project.  It sets up any registers    @
@ needed to copy the main program from the external parallel ROM to the        @
@ external SRAM.  This includes the following peripherals:                     @
@                                                                              @
@   - Clock                                                                    @
@   - ROM and SRAM chips selects                                               @
@                                                                              @
@ It then copies the main program from the ROM to the SRAM and jumps to the    @
@ beginning of code in the SRAM.                                               @
@                                                                              @
@ Revision History:                                                            @
@                                                                              @
@   2010/02/02  Joseph Schmitz  Modified code from Arthur Chang to make it     @
@                               available to the students.                     @
@                                                                              @
@   2011/01/27  Joseph Schmitz  Split from crt0.s to boot.s                    @
@                                                                              @
@   2011/01/31  Joseph Schmitz  Updated exception vectors to include special   @
@                               word value at ARM vector 6. (see 13.3.2)       @
@                                                                              @
@   2011/02/06  Joseph Schmitz  Added documentation on exception vectors.      @
@                                                                              @
@   2017/05/21  William Werst   Modified to work for this project              @
@                                                                              @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

.include    "at91rm9200.inc"
.include    "system.inc"
.include    "macro.inc"
.include    "boot.inc"


.text
.arm


@@@ Exception Vectors @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
@ Start of the IRQ vector table.  This defines the interrupt handler for each
@ type of interrupt.  Must be at the memory address 0x0 (remapped at boot).
@
@ Exception                  Description
@
@ Reset                      Occurs when the processor reset pin is asserted.
@                            This exception is only expected to occur for
@                            signalling power-up, or for resetting as if the
@                            processor has just powered up. A soft reset can be
@                            done by branching to the reset vector (0x0000).
@
@ Undefined Instruction      Occurs if neither the processor, or any attached
@                            coprocessor, recognizes the currently executing
@                            instruction.  Software Interrupt (SWI) This is a
@                            user-defined synchronous interrupt instruction. It
@                            allows a program running in User mode, for example,
@                            to request privileged operations that run in
@                            Supervisor mode, such as an RTOS function.
@
@ Sofwarte Interrupt         Occurs when the processor generates a software
@                            interrupt.
@
@ Prefetch Abort             Occurs when the processor attempts to execute an
@                            instruction that has prefetched from an illegal
```

```
@                           address.
@
@ Data Abort                Occurs when a data transfer instruction attempts to
@                           load or store data at an illegal address.
@
@ Vector 6                  Used to specify the number of bytes to download from
@                           and external boot memory into internal SRAM on
@                           reset.
@
@ IRQ                       Occurs when the processor external interrupt request
@                           pin is asserted (LOW) and the I bit in the CPSR is
@                           clear.
@
@ FIQ                       Occurs when the processor external fast interrupt
@                           request pin is asserted (LOW) and the F bit in the
@                           CPSR is clear.
@
@
@ The format is as follows:
@
@   reset:    Reset vector
@   undef:    Undefine instruction
@   swi:      Software interrupt
@   abort:    Prefect abort
@   data:     Data abort
@   btldr:    Defines how much data to download from the boot memory
@   irq:      Normal interrupt (low priority).  Actual ISR address loaded from AIC
@   fiq:      Fast interrupt (high priority)

IRQTable:
.org 0x0

    reset:
        B   _start
    undef:
        B   undef
    swi:
        B   swi
    prefetch:
        B   prefetch
    data:
        B   data
    btldr:
        .word 0x00000008
    irq:
        LDR PC, [PC, #-0xF20]
    fiq:
        B   fiq



@@@ Startup Code (system reset restarts here) @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
.org 0x20

.global _start
_start:

@ In this file you must do the following (at least for now):
@
@   - Switch to the master clock
@
@   - Wait for it to stabilize.  The datasheet tells you how long this will
@     take.  You will need to force your CPU to do no external memory
@     operations during this time.  There is an easy way to do this, but
@     think about it and only ask me if you still can't come up with anything.
@
```

```
@    - Set up the chip selects for SRAM/ROM
@


@@@ Clock Initialization @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @Configure PLLA for DIVA = 5, MULA = 22

    mSET_HREG PMC_PLLAR, PMC_PLLAR_VAL

    @Configure MCK @MDIV = 00, PRES = 001, CSS = 10
    @First, need to change one value at a time, so change CSS first
    LDR r0,         =PMC_MCKR           @Load PMC_MCKR address
    LDR r1,         [r0]                @Load current PMC_MCKR value
    LDR r0,         =PMC_MCKR_VAL       @Check current value to see if any changes
    CMP r1,         r0                  @Because clock ready bit will not go
    BEQ DoneMCK                         @high if the same value is re-written.

    AND r1,     r1, #0xFFFFFFFC         @Mask out CSS bits
    LDR r0, =(PMC_MCKR_VAL & 0x03)      @Load the CSS component of final MCKR value
    ORR r1,         r0                  @Merge current PRES value with new CSS
    LDR r0,         =PMC_MCKR           @Load PMC_MCKR address
    STR r1, [r0]                        @Store intermediate value in MCKR
    LDR r2, =2000                       @Clock initialization timeout
WaitMCKRDY:
    SUBS r2, r2, #1                     @Timeout
    BLO DoneMCKRDY
    LDR r0, =PMC_SR
    LDR r1, [r0]
    TST r1, #0x8
    BEQ WaitMCKRDY
    @B DoneMCKRDY
DoneMCKRDY:


    mSET_HREG PMC_MCKR, PMC_MCKR_VAL
    mSET_HREG PMC_MCKR, (PMC_MCKR_VAL | (PMC_MCKR_MDIV << 8))
    @B DoneMCK
DoneMCK:
    @Configure Peripheral clock
    mSET_HREG PMC_PCER, PMC_PCER_VAL

    @Configure PCK0
    mSET_HREG PMC_PCK0, PMC_PCK0_VAL

    @Configure PCK1
    @mSET_HREG PMC_PCK1, PMC_PCK1_VAL

    @Configure PIOA
    @mSET_HREG PIOA_BSR, PIOA_BSR_VAL
    @mSET_HREG PIOA_OER, PIOA_OER_VAL
    @mSET_HREG PIOA_PDR, PIOA_PDR_VAL

    @Configure PIOB
    mSET_HREG PIOB_ASR, PIOB_ASR_VAL
    mSET_HREG PIOB_OER, PIOB_OER_VAL
    mSET_HREG PIOB_PDR, PIOB_PDR_VAL



    @Enable PCK0, PCK1 output
    mSET_HREG 0xFFFFFC00, 0x00000300


    @@@ CS Initialization @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

    @Setup SRAM
```

```
    mSET_HREG SMC_CSR1, SMC_CSR1_VAL

    @Setup DRAM
    mSET_HREG SMC_CSR2, SMC_CSR2_VAL

    @Setup ROM
    mSET_HREG SMC_CSR7, SMC_CSR7_VAL

    @@@ Verify DRAM and SRAM are functioning
    @@Check SRAM valid

    @Test SRAM
    LDR r0, =SRAM_START
    LDR r1, =SRAM_SIZE
    BL mem_test
    CMP     r0,     #TRUE
    BNE     memtestfail

    @Test DRAM
    LDR r0, =DRAM_START
    LDR r1, =DRAM_SIZE
    BL mem_test
    CMP     r0,     #TRUE
    BNE     memtestfail

    @@@ Copy Code from External ROM -> External SRAM @@@@@@@@@@@@
    LDR     r0,     =SRAM_SIZE - 4
    LDR     r1,     =ROM_START
    LDR     r2,     =SRAM_START
CopyROMToSRAM:
    LDR     r3,     [r1, r0]
    STR     r3,     [r2, r0]
    SUBS    r0,     #4
    BHS     CopyROMToSRAM

    @Check that code loaded into SRAM matches code in ROM
    LDR     r0,     =SRAM_SIZE - 4
    LDR     r1,     =ROM_START
    LDR     r2,     =SRAM_START
CheckCopyToSRAM:
    LDR     r3,     [r1, r0]
    LDR     r4,     [r2, r0]
    CMP     r3,     r4
    BNE     LoadToSRAMFailed
    SUBS    r0,     #4
    BHS     CheckCopyToSRAM
    @@@ Branch to the Main Body of Code Now Located in the External SRAM @@@@@@@@@@@@

    @Uncomment this to branch to the copied code
    BL      SRAM_START

@If don't want to branch to low_level_init, fall through to BootEndLoop
BootEndLoop:
    B BootEndLoop

memtestfail:
    B memtestfail

LoadToSRAMFailed:
    B LoadToSRAMFailed

.end
```