```
'*****************************************************************
'                        Written by David Moran
' This is a genereric bootloader program
'*****************************************************************
' This program is after modification by Lester Johnson
' The echo fuction is actiated for implementing TTY protocol
' and they are changes in error messages to feed in
' Visual Basic Loader program.
' Baudrate is set for 19200 bps
' The PinC.1 is for selecting mode Boot or Run
'Program Name: BootM8.Bas
'
'Include at end of regular program
'
'Bootloader Code:
'The fusebits need to be set for 128 bytes for the boot code,
'starting at $F80
'
'Fusebits:        Boot  Application    Bootloader     End of         Boot Reset
'BOOTSZ1 BOOTSZ0  Size  Flash Section  Flash Section  Application    Address
'  1        1     128    $000 - $F7F    $F80 - $FFF    $F7F           $F80
'
'Standard Intel hex file can be sent: (spaces added for readability)
'All record sizes must be even, AVR uses WORDs, not bytes
'
' :Size Address  Type               Data Bytes                          Checksum
' :10    00 00    00    26 C0 B3 C0 B3 C0 B3 C0 B3 C0 C5 C0 C5 C0 D0 C0    A4
' :10    00 10    00    DB C0 E4 C0 ED C0 30 31 32 33 34 35 36 37 38 39    E7
'  -
'  -
' :10    05 30    00    55 DF 08 95 57 E5 57 DF 52 E5 55 DF 54 E5 53 DF    A2
' :10    05 40    00    5C E2 51 DF 53 2F 52 95 49 DF 53 2F 47 DF 01 D0    33
' :0B    05 50    00    08 95 5D E0 48 DF 5A E0 46 DF 08 95 A2 DC
' :00    00 00    01                                                       FF
'
'*****************************************************************************
$regfile = "M8def.dat"                                    'Set the chip type to ATmega8

Const Ramend_lo = $5f                                     'RAM ending location to set up stack pointer
Const Ramend_hi = $04

Const Pagesize = 32                                       'Flash Memory Page Size, in WORDS

'Variable Definitions:
!.def Tmp_reg = R16                                       'Temporary register for calculations etc.
!.def Hex_reg = R17                                       'Hex calculation register
```

```
!.def Ser_reg = R18                                     'Serial character buffer register
!.def SPM_reg = R19                                     'Temporary register for SPM register settings
!.def Rec_size = R20                                    'Number of data bytes in this Hex file line
!.def Chk_sum = R21                                     'Checksum storage

'Dummy Start code for Simulator

   jmp   $f80
   End                                                  'end program

'*****************************************************************************

'Start of Bootloader Code Area

'$boot = $f80                           'Set boot vector to F80 - CORRECT VALUE
$boot = $f7f                                            'Set boot vector to F7F and add a NOP, for BASCO
   nop

Disable Interrupts                                      'no interrupts allowed during bootloader program

_chk_for_bootload:                                      'Check for bootload, this one uses hardware, Por
   cbi   DDRC,1                                         'Clear the data direction bit for input
   sbi   PORTC,1                                        'Set the pull-up
   sbic  PINC,1                                         'Skip next instruction if pin is clear
   jmp   $0000                                          'Pin must be high, run normal code

_bootloader_start:                                      'Otherwise, run the bootloader
   ldi   tmp_reg,Ramend_Hi                              'Load temp reg with the top of SRAM value
  !out   SPH,tmp_reg                                    'Move out to stack pointer low byte
   ldi   tmp_reg,Ramend_Lo                              'Load temp reg with the top of SRAM value
  !out   SPL,tmp_reg                                    'Move out to stack pointer low byte
   ldi   tmp_reg,$00                                    'Load the temp register with USART settings
  !out   UCSRA,tmp_reg                                  'Set up the USART
   ldi   tmp_reg,$18                                    'Load the temp register with USART settings
  !out   UCSRB,tmp_reg                                  'Set up the USART
   ldi   tmp_reg,$86                                    'Load the temp register with USART settings
  !out   UCSRC,tmp_reg                                  'Set up the USART
   ldi   tmp_reg,$00                                    'Load the temp register with USART baud rate hig
  !out   UBRRH,tmp_reg                                  'Set up the USART
  '19.2 kbs
   ldi   tmp_reg,25                                     'Load the temp register with USART baud rate low
  '4800
  'ldi   tmp_reg,103                                     'Load the temp register with USART baud rate lo
  !out   UBRRL,tmp_reg                                  'Set up the USART
   clt                                                  'Clear the T flag, used to indicate end of file
```

```
_send_boot_msg:                         'Send a bootloader started message
   ldi   ser_reg, asc("B")              'Load "B" to show bootloader enabled
   rcall _send_ser                      'Call routine to send a character

_read_lines:                            'Read in the lines from serial port to SRAM
   rcall _receive_hex_line              'Receive a single line from the UART into SRAM

_parse_line:                            'Decode the current hex line
   ldi   XH,$01                         'Point to start of line, high byte, uses $0100
   ldi   XL,$00                         'Point to start of line, low byte
   clr   chk_sum                        'Clear the checksum register for this line

_read_header:
   ld    tmp_reg,x+                     'Get first character, should be ":"
   cpi   tmp_reg, asc(":")              'Compare with ":" to send as error flag
   breq  _header_ok                     'Fine, read the next record
 _header_err:                           'Not ":", send error character
   ldi   ser_reg, asc("!")              'Header error
   rcall _send_ser                      'Call routine to send a character
 _header_ok:                            'Fine, read the next record

_read_record_size:                      'Read the data byte count for this line
   rcall _char_to_byte                  'Call routine to convert two characters to byte
   mov   rec_size,hex_reg               'Save number of bytes in this line
   tst   rec_size                       'Test if record size is zero for this line
   brne _read_address                   'Not the final line, continue
   !set                                 'Set the T flag indicating write last page
   rjmp _write_current_page             'Final line, write current page exit to main pro

_read_address:                          'Read the address high byte and low bytes into Z
   rcall _char_to_byte                  'Call routine to convert two characters to byte
   mov   ZH,hex_reg                     'Load ZH with page address high byte
   rcall _char_to_byte                  'Call routine to convert two characters to byte
   mov   ZL,hex_reg                     'Load ZL with page address low byte

_read_record_type:                      'Read the record type for this line
   rcall _char_to_byte                  'Call routine to convert two characters to byte

_read_data_pairs:                       'Read the rest of the data bytes
   rcall _char_to_byte                  'Call routine to convert two characters to byte
   mov   r0,hex_reg                     'Save in R0, LS Byte of page write buffer
   rcall _char_to_byte                  'Call routine to convert two characters to byte
   mov   r1,hex_reg                     'Save in R1, MS Byte of page write buffer

_store_word:                            'Store current R1/R0 Word in page buffer at prop
   ldi   spm_reg,1                      'Load SPM Enable (SPMEN)
```

```
   rcall _exec_spm                                    'Execute current SPM, return is in exec_spm

_check_byte_count:                                    'Check if this was the last word of the line
   subi  rec_size,2                                   'Decrement the Record Size by two bytes
   breq _read_checksum                                'Done with this data, read checksum
   subi  ZL,$FE                                       'Not done, increment the low address byte by two
   rjmp  _read_data_pairs                             'Go back and read the next two characters

_read_checksum:                                       'Byte count = record size, next is checksum for
   rcall _char_to_byte                                'Call routine to convert two characters to byte
   breq  _checksum_ok                                 'Must have added to zero, checksum is okay
  _checksum_err:                                      'Checksum or other decoding error detected
   ldi   ser_reg, asc("#")                            '! zamiast #                              'Load
   rcall _send_ser                                    'Call routine to send a character
  _checksum_ok:

'Done decoding and storing one complete input line, so check if this page is full

_chk_page_full:                                       'Check if this page is full
   mov   tmp_reg, ZL                                  'Load the address low byte into the temp reg
   andi  tmp_reg,((Pagesize - 1 )*2)                  'AND with page size for this device, mask bits
   cpi   tmp_reg,((Pagesize - 1 )*2)                  'Compare with page size for this device

   brne  _read_lines                                  'Page buffer is not full, read another line

_write_current_page:                                  'Write current page if the page buffer is full

 _erase_page:                                         'Erase Page, page address is in Z12-Z6
   ldi   spm_reg,3                                    'Load Page Erase (PGERS) and SPM Enable (SPMEN)
   rcall _exec_spm                                    'Execute current SPM

 _write_page:                                         'Page address range is Z12-Z6 for Mega8
   andi  ZL,$C0                                       'Ensure that Z5 - Z0 are 0
   ldi   spm_reg,5                                    'Load page write (PGWRT) and SPM Enable (SPMEN)
   rcall _exec_spm                                    'Execute current SPM

 _enable_page:                                        'Re-enable the Read-While-Write section
   rcall _wait_spm                                    'Check if current write is complete
   ldi   spm_reg,11                                   'Set RWWSRE and SPMEN only
   rcall _exec_spm                                    'Execute current SPM

_check_end_of_file:                                   'Check if this is the end of the file
   brtc  _read_lines                                  'Not the last page, continue to read in data lin

_exit_bootloader:                                     'Done, exit the bootloader code
   'ldi   ser_reg, asc("D")                           'Load "D" to send as done flag
```

```
   'rcall _send_ser                                          'Call routine to send a character


   jmp $0000                                                'Jump to main program reset vector

'*****************************************************************************
'Send a serial character
_send_ser:                                                  'Send a serial characater
   sbis  UCSRA,UDRE                                         'Check if USART data register is empty
   rjmp  _send_ser                                          'Not ready yet, wait
  !out   UDR,ser_reg                                        'Send serial register
 ret
'*****************************************************************************
'Get one line from the serial port and store at start of SRAM
_receive_hex_line:
   ldi   XH,$01                                             'Set pointers to SRAM location $0100
   ldi   XL,$00                                             'Above all registers

   ldi   ser_reg, asc("?")                                  'No data now, so load "?" to request next charac
   rcall _send_ser                                          'Call routine to send a character

 _receive_hex_line_char:                                    'Get a character from UART and add to buffer
   sbis  UCSRA,RXC                                          'Check UART for a serial character received
   rjmp  _receive_hex_line_char                             'No, check again...
   in    tmp_reg,UDR                                        'Store input character in temp register
'Diag  Comment2 Lines
   'ldi   ser_reg, asc("*")
   mov   ser_reg, tmp_reg                                   'Echo this character for troubleshooting
   rcall _send_ser
                                                            'Call routine to send a character
'Diag

   cpi   tmp_reg,13                                         'Compare with <H0D>, input line terminator
   breq  _receive_hex_line_char


   cpi   tmp_reg,10                                         'Compare with <H0A>, input line terminator
   breq  _receive_hex_line_end                              'Yes, line is finished, branch to end
   st    x+,tmp_reg                                         'Otherwise store value then increment buffer and
   rjmp  _receive_hex_line_char                             'Go back and get next character

 _receive_hex_line_end:                                     'This input line is finished, so
 ret                                                        'Done with this line, so return
'*****************************************************************************
'Get two characters from buffer, add to checksum and return with result in hex_reg
_char_to_byte:
```

```
   ld    hex_reg,x+                                    'Load character into hex_reg, increment X
   subi  hex_reg,$41                                   'ASCII Value minus $41, "A"
   brcc  _char_to_byte1                                'Branch if value was greater than $41
   subi  hex_reg,$F9                                   'Not greater, subtract $F9
 _char_to_byte1:
   subi  hex_reg,$F6                                   'Subtract $F6
   lsl   hex_reg                                       'Shift this data
   lsl   hex_reg                                       'Left for four bits
   lsl   hex_reg                                       'To move it into the
   lsl   hex_reg                                       'High nibble

   ld    tmp_reg,x+                                    'Get next character,
   subi  tmp_reg,$41                                   'ASCII Value minus $41, "A"
   brcc  _char_to_byte2                                'Branch if value was greater than $41
   subi  tmp_reg,$F9                                   'Not greater, subtract $F9
 _char_to_byte2:
   subi  tmp_reg,$F6                                   'Subtract $F6

   add   hex_reg,tmp_reg                               'Add into hex register
   add   chk_sum,hex_reg                               'Add it into the checksum for this line
  ret
'*****************************************************************************
_exec_spm:                                             'Execute the current SPM instruction
  !out   spmcr,spm_reg                                 'Send to SPM Control Register
   spm                                                 'Do SPM instruction

 _wait_spm:                                            'Check if current flash write is complete
   in    spm_reg,spmcr                                 'Get the SPM control Register
   sbrc  spm_reg,spmen                                 'Check if SPM Enable flag is clear
   rjmp  _wait_spm                                     'No, go back and wait for SPMEN flag cleared
  ret                                                  'Flag cleared, Return
'*****************************************************************************
```