; Program to read nchar ASCII characters and sum up any HEX digits (0-F)
; HEX digits can also be lower case
; This is the MAS version
; Just uses a pre-defined string of characters starting at msg. Sum is in D2 at end.
; Output result to screen when done

xref   strout, decout, newline, stop

start: lea     msg, a1   ; put address of msg in a1
move.w   nchar, d1  ; number of bytes to read
clr.w    d2      ; clear the register for the sum (specify .w)
jmp      enter

loop:    move.b (a1)+, d0  ; move the character from msg array to d0
jsr      cnvhx   ; subroutine to find valid HEX digits
add.w    d0, d2  ; number or 0 returned in d0
enter:   dbra    d1, loop  ; subtract 1 from d1 and see if done
                ; all data processed, sum in d2
                ; now output the information
jsr      newline
lea      msg5, a0  ; output sum message
move.w   #31, d0
jsr      strout
move.w   d2, d0  ; output the sum
jsr      decout
jsr      newline
jsr      stop     ; end of program
                ; Subroutine cnvhx tests ASCII characters to see if they represent HEX digits
                ; and if so, returns value (.w) in d0. If not valid digit, returns 0

cnvhx:   and.b    #$7F, d0 ; mask off parity bit of character
cmp.b    #$30, d0  ; see if it is less than $30
blt      exit     ; if so, exit subroutine with zero result
cmp.b    #$39, d0  ; see if it is greater than $39
bgt      skip     ; if so, check for a-f
and.w    #$000F, d0 ; return 16 bit pos. number in d0
rts

exit:    clr.w    d0  ; not a hex digit - return 0 as 16 bit word
          rts

          ; Look for hex digits ($41-$46 for A-F or $61-$66 for a-f)
          ; Set bit 5 (lsb=0) so we only have to check lower case characters
          ; When get valid digit, subtract $57 to convert to HEX value
skip:    or.b     #$20, d0  ; set bit to assure lower case
          cmp.b    #$61, d0  ; see if less than $61 (a)
          blt      exit     ; if so exit with 0 result
          cmp.b    #$66, d0  ; see if greater than $66 (f)
          bgt      exit     ; exit with 0 result
          sub.b    #$57, d0  ; found valid HEX digit offset by $57 from value
          and.w    #$000F, d0 ; clear high order bits (if any) for 16 bit word
          rts
data

msg5:    dc.b     'The sum of numbers entered is: '
msg:     dc.b     'abC89HiJK185973nb642aBc89HiJK185973nb642'  ; define character string
nchar:   dc.w     40       ; number of characters to read
even
end
Program to read nchar ASCII characters and sum up any HEX digits (0-F)
HEX digits can also be lower case
Modified for Easy68K simulator
Just uses a pre-defined string of characters starting at msg. Sum is in D2 at end.

start: org $1000
lea msg,a1 ; put address of msg in a1
move.w nchar,d1 ; number of bytes to read
clr.w d2 ; clear the register for the sum (specify .w)
jmp enter ; enter loop at end
loop: move.b (a1)+,d0 ; move the character from msg array to d0
jsr cnvhx ; subroutine to find valid HEX digits
add.w d0,d2 ; number or 0 returned in d0
enter: dbra d1,loop ; subtract 1 from d1 and see if done
; all data processed, sum in d2
STOP #$2000 ; end of program

Subroutine cnvhx tests ASCII characters to see if they represent HEX digits
; and if so, returns value (.w) in d0. If not valid digit, returns 0

cnvhx: and.b #$7F,d0 ; mask off parity bit of character
cmp.b #$30,d0 ; see if it is less than $30
blt exit ; if so, exit subroutine with zero result
ncmp.b #$39,d0 ; see if it is greater than $39
bgt skip ; if so, check for a-f
and.w #$000F,d0 ; return 16 bit pos. number in d0
rts ; all done if 0-9
exit: clr.w d0 ; not a hex digit - return 0 as 16 bit word
rts

Look for hex digits ($41-$46 for A-F or $61-$66 for a-f)
Set bit 5 (lsb=0) so we only have to check lower case characters
When get valid digit, subtract $57 to convert to HEX value

skip: or.b #$20,d0 ; set bit to assure lower case
ncmp.b #$61,d0 ; see if less than $61 (a)
blt exit ; if so exit with 0 result
ncmp.b #$66,d0 ; see if greater than $66 (f)
bgt exit ; exit with 0 result
sub.b #$57,d0 ; found valid HEX digit offset by $57 from value
and.w #$000F,d0 ; clear high order bits (if any) for 16 bit word
rts

org $2000 ; data storage area
msg: dc.b 'aBc89HiJKl85973nb642aBc89HiJKl85973nb642' ; define character string
nchar: dc.w 40 ; number of characters to read
END START
Prob. 2 solution:

(a) No. It is a directive for the assembler indicating that these subroutines are external (not part of the current code segment). The "linker" program will need to provide access to these routines when the program is loaded into the computer memory to run.

(b) This is address register indirect with post-increment mode.
   1) The initial address of the msg string is placed in A1. The initial result is placed in that address, then the postincrement mode increases the contents of A1 by 1 (due to .b operation) so the next character goes into the next byte, etc.

(c) The MAS system uses A5 to store the address of the beginning of the data area. To use A5 for our own array would interfere with the proper operation of the program since references to the data area use the address in A5, for example the instruction on line 37 "lea msg4, so" puts the address of msg4 into A0. To do this it finds msg4 with an indirect addressing mode relative to A5 (address register indirect with displacement mode - see Notes on N68000 Programming).

(c) This is immediate addressing. The data (37) is in the instruction itself (in the program area).

(d) The instruction decrements a loop count in AL and branches to loop: unless the result of subtracting 1 (16 bits) is -1. If the result is -1, the next instruction is executed thereby exiting the loop.

(e) bte yes, b1t no, bgt yes.

(f) Yes. It is used for subroutine calls to store the address of the next instruction to be performed on return from the subroutine. The last occupied address on the stack is contained in the stack pointer (USP in the case of a MAS program).
Problem 2 (continued):

(9) To convert to uppercase, must look for characters between $61$ and $7A$ and clear bit 5 (where bit 0 = 56).

\[
\begin{align*}
&\text{bit } 0 = 01100001 \\
&\text{bit } 5 = 01111010 \\
&\text{Since bit 5 is set for all such characters, we can clear it safely by subtracting $20$.}
\end{align*}
\]

So change \(70\) to skip: \(\text{cmp.b } #61, \text{d0}\)

\[
\begin{align*}
&\text{cmp.b } #7A, \text{d0} \\
&\text{sub.w} \ #20, \text{d0}
\end{align*}
\]

(11) Between 83 and 84, say, insert:

- \(\text{movw} \ #0, \text{lo} \)
- \(\text{movw} \ #0, \text{hi} \)
- \(\text{addw} \ #0, \text{lo} \)
- \(\text{addw} \ #1, \text{hi} \)
- \(\text{movb} \ #0, \text{lo} \)
- \(\text{movb} \ #1, \text{hi} \)
- \(\text{movb} \ #2, \text{lo} \)
- \(\text{movb} \ #3, \text{hi} \)
- \(\text{movb} \ #4, \text{lo} \)
- \(\text{movb} \ #5, \text{hi} \)
- \(\text{movb} \ #6, \text{lo} \)
- \(\text{movb} \ #7, \text{hi} \)
- \(\text{movb} \ #8, \text{lo} \)
- \(\text{movb} \ #9, \text{hi} \)
- \(\text{movb} \ #A, \text{lo} \)
- \(\text{movb} \ #B, \text{hi} \)
- \(\text{movb} \ #C, \text{lo} \)
- \(\text{movb} \ #D, \text{hi} \)
- \(\text{movb} \ #E, \text{lo} \)
- \(\text{movb} \ #F, \text{hi} \)

Now you might want to output newmsg at the end, but this was not specified.

Problem 3

Note correction - this is a read cycle. See next page for figure.

\(\overline{AD} = \text{correct address on } AD - MS (\text{active low})\)

\(\overline{OE} = \text{correct address on } OE - MS (\text{active low})\)

\(\overline{OE} \text{ for input register is } \overline{AD} \cdot \overline{E} \cdot \overline{DS} = AD \cdot E \cdot DS\)

When \(OE\) is low, data are presented from input register on DO-07. The input register must have tri-state outputs so they can go into high-impedance mode when the register is not being addressed to read.

In the figure, I assumed the data bits were already present in the input register at the beginning of the memory cycle. Figure 11.4 in Thrope and Hill shows a delay before valid DO-07 during the read cycle to make the point that the data need not be valid until shortly before DS goes high. Thus data could be latched when DS goes low.
3. Diagrams below from Horowitz and Hill show connections of an input register and an output register to an M68008 microprocessor. Also shown is a read cycle timing diagram with $R/W$, VALID ADDRESS (lumped together as usual), etc. Specifications for the 74HCT574 octal D latch are attached.

(a) On the timing diagram, show the address decoder output, $\overline{AD}$, and the $\overline{OE}$ signal on the input register octal D latch when it is accessed with its valid address during a read cycle. Also show the data bus signals from the register (D0-7 lumped together like A0-19).

(b) Explain why the input register must have tri-state output lines.
9. (c) First binary number = 100
   If comparator output is low, the DAC output is too high, clear msb and try next bit; send 010.
   Now comparator output is high, so keep 2nd bit and try 3rd; send 011. Now comparator is low,
   so final output is 010. This is as close as we get with only 3 bits.

(b) This is in principle faster since fewer steps are required to produce the answer.

(c) The flash ADC determines the answer in one step, but more circuitry is required (8 bits requires 256 comparators).