14.2.1

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax
        .equ    nBytes,50  @ amount of memory for string
@ Constant program data
        .section .rodata
        .align  2
prompt:
        .asciz        "Enter some alphabetic characters: "
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 16      @ space for saving regs
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer
        mov     r0, nBytes      @ get memory from heap
        bl      malloc
        mov     r4, r0          @ pointer to new memory
        ldr     r0, promptAddr  @ prompt user
        bl      writeStr
        mov     r0, r4          @ get user input
        mov     r1, nBytes      @ limit input size
        bl      readLn

        mov r0, r4      @ convert to lowercase
        bl  toLower

        mov     r0, r4          @ echo user input
        bl      writeStr

        mov     r0, r4          @ free heap memory
        bl      free

        mov     r0, 0           @ return 0;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @ restore sp
        bx      lr              @ return
promptAddr:
        .word    prompt
```

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Useful source code constants
        .equ    lowerMask,0x20
        .equ    NUL,0

@ The code
        .text
        .align  2
        .global toLower
        .type   toLower, %function
toLower:
        sub     sp, sp, 16      @ space for saving regs
        str     r4, [sp, 0]     @ save r4
        str     r5, [sp, 4]     @      r5
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer

        mov     r4, r0          @ r4 = string pointer
        mov     r5, #0          @ r5 = count
whileLoop:
        ldrb    r3, [r4]        @ get a char
        cmp     r3, NUL         @ end of string?
        beq     allDone         @ yes, all done

        orr     r3, r3, lowerMask  @ convert to lowercase
        strb    r3, [r4]        @ update string

        add     r4, r4, 1       @ increment pointer var
        add     r5, r5, 1       @ count++
        b       whileLoop       @ back to top
allDone:
        mov     r0, r5          @ return count;
        ldr     r4, [sp, 0]     @ restore r4
        ldr     r5, [sp, 4]     @         r5
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @ restore sp
        bx      lr              @ return
```

14.4.1

gcc -g -o hexConvert1 hexToInt2.o writeStr.o readLn.o hexConvert1.c

14.4.2

It works for both cases because all alphabetic characters are numerically higher than the numeral characters in the ASCII Code.

14.4.3

```
1         .cpu    cortex-a53
2         .fpu    neon-fp-armv8
3         .syntax unified         @ modern syntax
4
5     @ Constant for assembler
6         .equ    maxChars,9      @ max input chars
7         .equ    theString,-16   @ for input string
8         .equ    locals,16       @ space for local vars
9
10    @ Constant program data
11        .section .rodata
12        .align  2
13    prompt:
14        .asciz        "Enter up to 32-bit hex number: "
15    display:
16        .asciz        "The integer is: %i\n"
17
18    @ The program
19        .text
20        .align  2
21        .global main
22        .type   main, %function
23    main:
24        sub     sp, sp, 8       @ space for fp, lr
25        str     fp, [sp, 0]     @ save fp
26        str     lr, [sp, 4]     @   and lr
27        add     fp, sp, 4       @ set our frame pointer
28        sub     sp, sp, locals  @ for local vars
```

```
28        sub     sp, sp, locals  @ for local vars
29
30        ldr     r0, promptAddr  @ prompt user
31        bl      writeStr
32
33        add     r0, fp, theString  @ place for user input
34        mov     r1, maxChars    @ limit input size
35        bl      readLn
36
37        add     r0, fp, theString  @ user input
38        bl      hexToInt        @ convert it
39
40        mov     r1, r0          @ result returned in r0
41        ldr     r0, displayAddr @ show user the result
42        bl      printf          @ from C Standard Lib.
43
44        mov     r0, 0           @ return 0;
45        add     sp, sp, locals  @ deallocate local var
46        ldr     fp, [sp, 0]     @ restore caller fp
47        ldr     lr, [sp, 4]     @       lr
48        add     sp, sp, 8       @   and sp
49        bx      lr              @ return
50
51    promptAddr:
52        .word    prompt
53    displayAddr:
54        .word    display
```

14.4.4

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constant for assembler
        .equ    maxChars,9      @ max input chars
        .equ    inString1,-24   @ for 1st input string
        .equ    inString2,-36   @ for 2nd input string
        .equ    outString,-52   @ for output string
        .equ    locals,40       @ space for local vars

@ Constant program data
        .section .rodata
        .align  2
prompt:
        .asciz      "Enter up to 32-bit hex number: "
display:
        .asciz      "Their sum is: "

@ The program
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 16      @ space for saving regs
        str     r4, [sp, 0]     @ save r4
        str     r5, [sp, 4]     @      r5
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer
        sub     sp, sp, locals  @ for local vars

        ldr     r0, promptAddr  @ prompt user
        bl      writeStr

        add     r0, fp, inString1  @ 1st input
        mov     r1, maxChars    @ limit input size
        bl      readLn
        add     r0, fp, inString1  @ user input
        bl      hexToInt        @ convert it
        mov     r4, r0          @ 1st int

        ldr     r0, promptAddr  @ prompt for 2nd
        bl      writeStr

        add     r0, fp, inString2  @ 2nd input
        mov     r1, maxChars    @ limit input size
        bl      readLn
        add     r0, fp, inString2  @ user input
        bl      hexToInt        @ convert it
        mov     r5, r0          @ 2nd int

        add     r1, r4, r5      @ add the two ints
        add     r0, fp, outString  @ place for result
        bl      intToHex        @ convert to hex string

        ldr     r0, displayAddr @ show user result
        bl      writeStr

        add     r0, fp, outString
        bl      writeStr

        bl      newLine         @ looks nicer

        mov     r0, 0           @ return 0;
        add     sp, sp, locals  @ deallocate local var
        ldr     r4, [sp, 0]     @ restore r4
        ldr     r5, [sp, 4]     @      r5
        ldr     fp, [sp, 8]     @      fp
        ldr     lr, [sp, 12]    @      lr
        add     sp, sp, 16      @ restore sp
        bx      lr              @ return

promptAddr:
        .word   prompt
displayAddr:
        .word   display
```

14.7.1

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    maxChars,11     @ max input chars
        .equ    inputString,-16 @ for input string
        .equ    outputString,-28 @ for output string
        .equ    locals,32       @ space for local vars

@ Constant program data
        .section .rodata
        .align  2
prompt:
        .asciz      "Enter an unsigned number up to 4294967294: "

@ The program
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @   and lr
        add     fp, sp, 4       @ set our frame pointer
        sub     sp, sp, locals  @ for local vars

        ldr     r0, promptAddr  @ prompt user
        bl      writeStr

        add     r0, fp, inputString  @ place for user input
        mov     r1, maxChars    @ limit input size
        bl      readLn

        add     r0, fp, inputString  @ user input
        bl      uDecToInt       @ convert it

        add     r1, r0, 1       @ increment user's number
        add     r0, fp, outputString
        bl      uIntToDec

        add     r0, fp, outputString
        bl      writeStr
        bl      newLine

        mov     r0, 0           @ return 0;
        add     sp, sp, locals  @ deallocate local var
        ldr     fp, [sp, 0]     @ restore caller fp
        ldr     lr, [sp, 4]     @       lr
        add     sp, sp, 8       @   and sp
        bx      lr              @ return

promptAddr:
        .word   prompt
```

14.7.2

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constant for assembler
        .equ    tempString,-40  @ for temp string
        .equ    locals,16       @ space for local vars
        .equ    zero,0x30       @ ascii 0
        .equ    NUL,0

@ The program
        .text
        .align  2
        .global uIntToDec
        .type   uIntToDec, %function
uIntToDec:
        sub     sp, sp, 24      @ space for saving regs
        str     r4, [sp, 0]     @ save r4
        str     r5, [sp, 4]     @      r5
        str     r6, [sp, 8]     @      r6
        str     r7, [sp, 12]    @      r7
        str     fp, [sp, 16]    @      fp
        str     lr, [sp, 20]    @      lr
        add     fp, sp, 20      @ set our frame pointer
        sub     sp, sp, locals @ for local vars

        mov     r4, r0          @ caller's string pointer
        add     r5, fp, tempString @ temp string
        mov     r7, 10          @ decimal constant

        mov     r7, 10          @ decimal constant

        mov     r0, NUL         @ end of C string
        strb    r0, [r5]
        add     r5, r5, 1       @ move to char storage

        mov     r0, zero        @ assume the int is 0
        strb    r0, [r5]
        movs    r6, r1          @ int to convert
        beq     copyLoop        @ zero is special case
convertLoop:
        cmp     r6, 0           @ end of int?
        beq     copy            @ yes, copy for caller
        udiv    r0, r6, r7      @ no, div to get quotient
        mls     r2, r0, r7, r6  @ the mod (remainder)
        mov     r6, r0          @ the quotient
        orr     r2, r2, zero    @ convert to numeral
        strb    r2, [r5]
        add     r5, r5, 1       @ next char position
        b       convertLoop
copy:
        sub     r5, r5, 1       @ last char stored locally
copyLoop:
        ldrb    r0, [r5]        @ get local char
        strb    r0, [r4]        @ store the char for caller
        cmp     r0, NUL         @ end of local string?
        beq     allDone         @ yes, we're done
        add     r4, r4, 1       @ no, next caller location
        sub     r5, r5, 1       @ next local char
        b       copyLoop

        b       copyLoop

allDone:
        strb    r0, [r4]        @ end C string
        add     sp, sp, locals  @ deallocate local var
        ldr     r4, [sp, 0]     @ restore r4
        ldr     r5, [sp, 4]     @      r5
        ldr     r6, [sp, 8]     @      r6
        ldr     r7, [sp, 12]    @      r7
        ldr     fp, [sp, 16]    @      fp
        ldr     lr, [sp, 20]    @      lr
        add     sp, sp, 24      @      sp
        bx      lr              @ return
```

14.7.3

next page.

incrementSigned.S

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constant program data
        .section .rodata
        .align  2
prompt:
        .asciz      "Enter a signed integer between -2147483648 and +2147483646: "

@ The program
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @   and lr
        add     fp, sp, 4       @ set our frame pointer

        ldr     r0, promptAddr  @ prompt user
        bl      writeStr

        bl      getDecInt       @ convert it

        add     r0, r0, 1       @ increment user's number
        bl      putDecInt       @ print result
        bl      newLine

        mov     r0, 0           @ return 0;
        ldr     fp, [sp, 0]     @ restore caller fp
        ldr     lr, [sp, 4]     @      lr
        add     sp, sp, 8       @   and sp
        bx      lr              @ return

promptAddr:
        .word   prompt
```

getDecInt.S

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    maxChars,12     @ max input chars
        .equ    inputString,-20 @ for input string
        .equ    locals,8        @ space for local vars

@ Useful source code constants
        .equ    POS,0
        .equ    NEG,1

@ The program
        .text
        .align  2
        .global getDecInt
        .type   getDecInt, %function
getDecInt:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer
        sub     sp, sp, locals  @ for the string

        add     r0, fp, inputString  @ place to store input
        mov     r1, maxChars    @ limit input length
        bl      readLn

        add     r0, fp, inputString  @ input string
        mov     r4, POS         @ assume postive int
```

getDecInt.S

```
        ldrb    r1, [r0]        @ get char
        cmp     r1, '-          @ minus sign?
        bne     checkPlus       @ no, check for plus sign
        mov     r4, NEG         @ yes, flag as neg
        add     r0, r0, 1       @ go to the number
        b       convert         @ and convert it
checkPlus:
        cmp     r1, '+          @ plus sign?
        bne     convert         @ no, we're at the number
        add     r0, r0, 1       @ go to the number
convert:
        bl      uDecToInt

        cmp     r4, POS         @ positive int?
        beq     allDone         @ yes, we're done
        mvn     r0, r0          @ no, complement it
        add     r0, r0, 1       @ and finish negate
allDone:
        add     sp, sp, locals  @ deallocate local var
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return
```

putDecInt.S

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    decString,-20   @ for  string
        .equ    locals,8        @ space for local varsbr

@ Useful source code constants
        .equ    POS,0
        .equ    NEGBIT,0x80000000

@ The program
        .text
        .align  2
        .global putDecInt
        .type   putDecInt, %function
putDecInt:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer
        sub     sp, sp, locals  @ space for the string

        add     r4, fp, decString  @ place to store string
        mov     r1, '+          @ assume positive
```

putDecInt.S

```
        add     r4, fp, decString  @ place to store string
        mov     r1, '+          @ assume positive
        strb    r1, [r4]
        tst     r0, NEGBIT      @ negative int?
        beq     positive        @ no, go on
        mov     r1, '-          @ yes, need to negate
        strb    r1, [r4]
        mvn     r0, r0          @ complement
        add     r0, r0, 1       @ two's complement
positive:
        mov     r1, r0          @ int to convert
        add     r0, r4, 1       @ skip over sign char
        bl      uIntToDec

        add     r0, fp, decString  @ string to write
        bl      writeStr

        add     sp, sp, locals  @ deallocate local var
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return
```