

## 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, 0x08
.equ NULL, 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, NULL @ 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 @ counter
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
24 main:
25 sub sp, sp, 8 @ space for fp, lr
26 str fp, [sp, 0] @ save fp
27 str lr, [sp, 4] @ and lr
28 add fp, sp, 4 @ set our frame pointer
29 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

```

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 inString1,-24 @ for 1st input string
8  .equ inString2,-36 @ for 2nd input string
9  .equ outString,-52 @ for output string
10 .equ locals,48 @ space for local vars
11
12 @ Constant program data
13 .section .rodata
14 .align 2
15 prompt:
16 .ascii "Enter up to 32-bit hex numbers: "
17 display:
18 .ascii "Their sum is: "
19
20 @ The program
21 .text
22 .align 2
23 .global main
24 .type main, %function
25
26 main:
27 sub sp, sp, 16 @ space for saving regs
28 str r4, [sp, 8] @ save r4
29 str r5, [sp, 4] @ r5

```

```

30 str r5, [sp, 4] @ r5
31 str fp, [sp, 8] @ fp
32 str lr, [sp, 12] @ lr
33 add fp, sp, 12 @ set our frame pointer
34 sub sp, sp, locals @ for local vars
35
36 ldr r0, promptAddr @ prompt user
37 bl writeStr
38
39 mov r1, maxChars @ limit input size
40 bl readLn
41 add r0, fp, inString1 @ user input
42 bl uDecToInt @ convert it
43 mov r4, r0 @ 1st int
44
45 ldr r0, promptAddr @ prompt for 2nd
46 bl writeStr
47
48 add r0, fp, inString2 @ 2nd input
49 mov r1, maxChars @ limit input size
50 bl readLn
51 add r0, fp, inString2 @ user input
52 bl hDecToInt @ convert it
53 mov r5, r0 @ 2nd int
54
55 add r1, r4, r5 @ add the two ints
56 add r0, fp, outString @ place for result
57 bl intToHex @ convert to hex string

```

```

57
58 ldr r0, displayAddr @ show user result
59 bl writeStr
60
61 add r0, fp, outString
62 bl writeStr
63
64 bl newLine @ looks nicer
65
66 mov r0, 0 @ return 0;
67 add sp, sp, locals @ deallocate local var
68 ldr r4, [sp, 0] @ restore r4
69 ldr r5, [sp, 4] @ r5
70 ldr fp, [sp, 8] @ fp
71 ldr lr, [sp, 12] @ lr
72 add sp, sp, 16 @ restore sp
73 bx lr @ return
74
75 promptAddr: .word prompt
76 displayAddr: .word display

```

## 14.7.1

```

1  .cpu cortex-a53
2  .fpu neon-fp-armv8
3  .syntax unified @ modern syntax
4
5  @ Constants for assembler
6  .equ maxChars,11 @ max input chars
7  .equ inputString,-16 @ for input string
8  .equ outputString,-28 @ for output string
9  .equ locals,32 @ space for local vars
10
11 @ Constant program data
12 .section .rodata
13 .align 2
14 prompt:
15 .ascii "Enter an unsigned number up to 4294967294: "
16
17 @ The program
18 .text
19 .align 2
20 .global main
21 .type main, %function
22
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

```

```

29
30 ldr r0, promptAddr @ prompt user
31 bl writeStr
32
33 add r0, fp, inputString @ place for user input
34 mov r1, maxChars @ limit input size
35 bl readLn
36
37 add r0, fp, inputString @ user input
38 bl uDecToInt @ convert it
39
40 add r1, r0, 1 @ increment user's number
41 add r0, fp, outputString
42 bl uIntToDec
43
44 add r0, fp, outputString
45 bl writeStr
46 bl newLine
47
48 mov r0, 0 @ return 0;
49 add sp, sp, locals @ deallocate local var
50 ldr fp, [sp, 0] @ restore caller fp
51 ldr lr, [sp, 4] @ lr
52 add sp, sp, 8 @ and sp
53 bx lr @ return
54
55 promptAddr: .word prompt

```

## 14.7.2

```

1  .cpu cortex-a53
2  .fpu neon-fp-armv8
3  .syntax unified @ modern syntax
4
5  @ Constant for assembler
6  .equ tempString,-40 @ for temp string
7  .equ locals,16 @ space for local vars
8  .equ zero,0x30 @ ascii 0
9  .equ NUL,0
10
11 @ The program
12 .text
13 .align 2
14 .global uIntToDec
15 .type uIntToDec, %function
16
17 uIntToDec:
18 sub sp, sp, 24 @ space for saving regs
19 str r4, [sp, 0] @ save r4
20 str r5, [sp, 4] @ r5
21 str r6, [sp, 8] @ r6
22 str r7, [sp, 12] @ r7
23 str fp, [sp, 16] @ fp
24 str lr, [sp, 20] @ lr
25 add fp, sp, 20 @ set our frame pointer
26 sub sp, sp, locals @ for local vars
27
28 mov r4, r0 @ caller's string pointer
29 add r5, fp, tempString @ temp string
30 mov r7, 10 @ decimal constant

```

```

31 mov r7, 10 @ decimal constant
32
33 mov r0, NUL @ end of C string
34 strb r0, [r5]
35 add r5, r5, 1 @ move to char storage
36
37 mov r0, zero @ assume the int is 0
38 strb r0, [r5]
39 movs r6, r1
40 beq copyLoop @ zero is special case
41
42 convertLoop:
43 cmp r6, 0 @ end of int?
44 beq copy @ yes, copy for caller
45 udiv r0, r6, r7 @ no, div to get quotient
46 mls r2, r0, r7, r6 @ the mod (remainder)
47 mov r6, r0 @ the quotient
48 orr r2, r2, zero @ convert to numeral
49 strb r2, [r5]
50 add r5, r5, 1 @ next char position
51 b convertLoop
52
53 copy:
54 sub r5, r5, 1 @ last char stored locally
55
56 copyLoop:
57 ldrb r0, [r5] @ get local char
58 strb r0, [r4] @ store the char for caller
59 cmp r0, NUL @ end of local string?
60 beq allDone @ yes, we're done
61 add r4, r4, 1 @ no, next caller location
62 strb r5, [r5, 1] @ next local char
63 b copyLoop

```

```

64
65 b copyLoop
66
67 allDone:
68 strb r0, [r4] @ end C string
69 add sp, sp, locals @ deallocate local var
70 ldr r4, [sp, 0] @ restore r4
71 ldr r5, [sp, 4] @ r5
72 ldr r6, [sp, 8] @ r6
73 ldr r7, [sp, 12] @ r7
74 ldr fp, [sp, 16] @ fp
75 ldr lr, [sp, 20] @ lr
76 add sp, sp, 24 @ sp
77 bx lr @ return

```

## 14.7.3

→ next page.

## incrementSigned.S

```

1      .cpu    cortex-a53
2      .fpu    neon-fp-armv8
3      .syntax unified      @ modern syntax
4
5      @ Constant program data
6      .section .rodata
7      .align 2
8
9      prompt:
10     .ascii "Enter a signed integer between -2147483648 and +2147483646: "
11
12 @ The program
13 .text
14 .align 2
15 .global main
16 .type main, %function
17
18 main:
19     sub    sp, sp, 8      @ space for fp, lr
20     str    fp, [sp, 0]    @ save fp
21     str    lr, [sp, 4]    @ and lr
22     add    fp, sp, 4      @ set our frame pointer
23
24     ldr    r0, promptAddr @ prompt user
25     bl     writeStr
26
27     bl     getDecInt      @ convert it
28
29     add    r0, r0, 1      @ increment user's number
30     bl     putDecInt      @ print result
31     bl     newline
32
33     mov    r0, 0          @ return 0
34     ldr    fp, [sp, 0]    @ restore caller fp
35     ldr    lr, [sp, 4]    @ lr
36     add    sp, sp, 8      @ and sp
37     bx     lr             @ return
38
39 promptAddr:
40     .word  prompt

```

## getDecInt.S

```

1      .cpu    cortex-a53
2      .fpu    neon-fp-armv8
3      .syntax unified      @ modern syntax
4
5      @ Constants for assembler
6      .equ    maxChars, 12 @ max input chars
7      .equ    inputString, -20 @ for input string
8      .equ    locals, 8    @ space for local vars
9
10     @ Useful source code constants
11     .equ    POS, 0
12     .equ    NEG, 1
13
14 @ The program
15 .text
16 .align 2
17 .global getDecInt
18 .type getDecInt, %function
19
20 getDecInt:
21     sub    sp, sp, 16     @ space for saving regs
22     @ (keeping 8-byte sp align)
23     str    r4, [sp, 4]    @ save r4
24     str    fp, [sp, 8]    @ fp
25     str    lr, [sp, 12]   @ lr
26     add    fp, sp, 12     @ set our frame pointer
27     sub    sp, sp, locals @ for the string
28
29     add    r0, fp, inputString @ place to store input
30     mov    r1, maxChars    @ limit input length
31     bl     readLn
32
33     add    r0, fp, inputString @ input string
34     mov    r4, POS         @ assume positive int

```

## getDecInt.S

```

35     ldrb   r1, [r0]       @ get char
36     cmp    r1, '-'        @ minus sign?
37     bne    checkPlus      @ no, check for plus sign
38     mov    r4, NEG        @ yes, flag as neg
39     add    r0, r0, 1      @ go to the number
40     b      convert        @ and convert
41
42 checkPlus:
43     cmp    r1, '+'        @ plus sign?
44     bne    convert        @ no, we're at the number
45     add    r0, r0, 1      @ go to the number
46
47 convert:
48     bl     uDecToInt
49
50     cmp    r4, POS        @ positive int?
51     beq    allDone        @ yes, we're done
52     mvn    r0, r0         @ no, complement
53     add    r0, r0, 1      @ and finish negate
54
55 allDone:
56     add    sp, sp, locals @ deallocate local var
57     ldr    r4, [sp, 4]    @ restore r4
58     ldr    fp, [sp, 8]    @ fp
59     ldr    lr, [sp, 12]   @ lr
60     add    sp, sp, 16     @ and sp
61     bx     lr             @ return

```

## putDecInt.S

```

1      .cpu    cortex-a53
2      .fpu    neon-fp-armv8
3      .syntax unified      @ modern syntax
4
5      @ Constants for assembler
6      .equ    decString, -20 @ for string
7      .equ    locals, 8    @ space for local varsbr
8
9      @ Useful source code constants
10     .equ    POS, 0
11     .equ    NEGBIT, 0x80000000
12
13 @ The program
14 .text
15 .align 2
16 .global putDecInt
17 .type putDecInt, %function
18
19 putDecInt:
20     sub    sp, sp, 16     @ space for saving regs
21     @ (keeping 8-byte sp align)
22     str    r4, [sp, 4]    @ save r4
23     str    fp, [sp, 8]    @ fp
24     str    lr, [sp, 12]   @ lr
25     add    fp, sp, 12     @ set our frame pointer
26     sub    sp, sp, locals @ space for the string
27
28     add    r4, fp, decString @ place to store string
29     mov    r1, '+'        @ assume positive

```

## putDecInt.S

```

26     add    r4, fp, decString @ place to store string
27     mov    r1, '+'        @ assume positive
28     strb   r1, [r4]
29     tst    r0, NEGBIT     @ negative int?
30     beq    positive       @ no, go on
31     mov    r1, '-'        @ yes, need to negate
32     strb   r1, [r4]
33     mvn    r0, r0         @ complement
34     add    r0, r0, 1      @ two's complement
35
36 positive:
37     mov    r1, r0         @ int to convert
38     add    r0, r4, 1      @ skip over sign char
39     bl     uIntToDec
40
41     add    r0, fp, decString @ string to write
42     bl     writeStr
43
44     add    sp, sp, locals @ deallocate local var
45     ldr    r4, [sp, 4]    @ restore r4
46     ldr    fp, [sp, 8]    @ fp
47     ldr    lr, [sp, 12]   @ lr
48     add    sp, sp, 16     @ and sp
49     bx     lr             @ return

```