ARM Assembler

Strings

Characters or Strings

- A *string* is a sequence of characters
- ASCII — 7-bit (American) character codes
  One byte per character (`char`)
- Unicode — International character code
  Two bytes per (wide) character (`wchar`)
- Unicode32 — Extended version of Unicode
  Four bytes per (wide) character (`wchar`)
- You need to know which type you are using
  `strlen()` — Length of ASCII string
  `wcslen()` — Length of Unicode string

Fixed Length / Terminated Strings

- Fixed Length
  String is always n characters
  String is truncated if too long
  String is padded out if too short

- Terminated
  Special character to mark end of string
  Normally a NUL (0x00) is used

- `char name[10]`
  Fixed Length or Terminated ?

  Terminated if string is short (under 10 chars)
  Truncated if string is long (over 10 chars)
  $\Rightarrow$ Be careful of buffer overrun

Counted Strings

- First byte (or halfword) is a count which indicates number of characters in string

```
struct counted_string_s {
    int  count;
    char string[];
};
```

- Used by Pascal, ADA, Basic, Fortran, . . .

Program: strlencr.s

```
 1  ; Find the length of a CR terminated string
 2
 5  CR    EQU     0x0D
 6
10  Main
11          LDR     R0, =Data1  ; Load the address of the lookup table
12          EOR     R1, R1, R1  ; Clear R1 to store count
13  Loop
14          LDRB    R2, [R0], #1 ; Load the first byte into R2
15          CMP     R2, #CR     ; Is it the terminator ?
16          BEQ     Done        ; Yes => Stop loop
17          ADD     R1, R1, #1  ; No => Increment count
18          BAL     Loop        ; Read next char
19
24          AREA    Data1, DATA
27          DCB     "Hello, World", CR
```

Program: strlencr.s

```
 1  ; Find the length of a CR terminated string
 2
 5  CR    EQU     0x0D
 6
10  Main
11          LDR     R0, =Data1  ; Load the address of the lookup table
12          EOR     R1, R1, R1  ; Clear R1 to store count
13  Loop
14          LDRB    R2, [R0], #1 ; Load the first byte into R2
15          CMP     R2, #CR     ; Is it the terminator ?
16          BEQ     Done        ; Yes => Stop loop
17          ADD     R1, R1, #1  ; No => Increment count
18          BAL     Loop        ; Read next char
19
24          AREA    Data1, DATA
27          DCB     "Hello, World", CR
```

EQU Equate a label CR to 0x0D (13), the string terminator

Program: strlencr.s

```
 1  ; Find the length of a CR terminated string
 2
 5  CR    EQU     0x0D
 6
10  Main
11          LDR     R0, =Data1  ; Load the address of the lookup table
12          EOR     R1, R1, R1  ; Clear R1 to store count
13  Loop
14          LDRB    R2, [R0], #1 ; Load the first byte into R2
15          CMP     R2, #CR      ; Is it the terminator ?
16          BEQ     Done         ; Yes => Stop loop
17          ADD     R1, R1, #1   ; No => Increment count
18          BAL     Loop         ; Read next char
19
24          AREA    Data1, DATA
27          DCB     "Hello, World", CR
```

CMP Compare the character to the terminator (`CR`)

Easier to read than: `CMP R2, 0x0D`

Program: strlencr.s

```
 1  ; Find the length of a CR terminated string
 2
 5  CR    EQU     0x0D
 6
10  Main
11          LDR     R0, =Data1  ; Load the address of the lookup table
12          EOR     R1, R1, R1  ; Clear R1 to store count
13  Loop
14          LDRB    R2, [R0], #1 ; Load the first byte into R2
15          CMP     R2, #CR     ; Is it the terminator ?
16          BEQ     Done        ; Yes => Stop loop
17          ADD     R1, R1, #1  ; No => Increment count
18          BAL     Loop        ; Read next char
19
24          AREA    Data1, DATA
27          DCB     "Hello, World", CR
```

DCB Define a string of characters

Program: strlencr.s

```
 1  ; Find the length of a CR terminated string
 2
 5  CR    EQU     0x0D
 6
10  Main
11          LDR     R0, =Data1  ; Load the address of the lookup table
12          EOR     R1, R1, R1  ; Clear R1 to store count
13  Loop
14          LDRB    R2, [R0], #1 ; Load the first byte into R2
15          CMP     R2, #CR     ; Is it the terminator ?
16          BEQ     Done        ; Yes => Stop loop
17          ADD     R1, R1, #1  ; No => Increment count
18          BAL     Loop        ; Read next char
19
24          AREA    Data1, DATA
27          DCB     "Hello, World", CR
```

Using conditional execution we could write:

```
ADDNE R1, R1, #1
BNE   Loop
```

Program: strlen.s

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #-1         ; Start count at -1
10  Loop
11          ADD     R1, R1, #1      ; Increment count
12          LDRB    R2, [R0], #1    ; Load the first byte into R2
13          CMP     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

Program: strlen.s

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #-1         ; Start count at -1
10  Loop
11          ADD     R1, R1, #1      ; Increment count
12          LDRB    R2, [R0], #1    ; Load the first byte into R2
13          CMP     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

MOV Initialise R1 to -1 to counter the first increment

Program: strlen.s

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #-1         ; Start count at -1
10  Loop
11          ADD     R1, R1, #1      ; Increment count
12          LDRB    R2, [R0], #1    ; Load the first byte into R2
13          CMP     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

LDRB Increment pointer (R0)

Program: strlen.s

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #-1         ; Start count at -1
10  Loop
11          ADD     R1, R1, #1      ; Increment count
12          LDRB    R2, [R0], #1    ; Load the first byte into R2
13          CMP     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

CMP We must check for zero. as LDRB can not set the flags

Program: strlen.s (Revised)

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #0          ; Set strlen to zero
10  Loop
11          LDRB    R2, [R0, R1]    ; Read n-th byte
12          ADD     R1, R1, #1      ; Increment strlen count
13          TEQ     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

Program: strlen.s (Revised)

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #0          ; Set strlen to zero
10  Loop
11          LDRB    R2, [R0, R1]    ; Read n-th byte
12          ADD     R1, R1, #1      ; Increment strlen count
13          TEQ     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

MOV Initialise R1 to zero length

Program: strlen.s (Revised)

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #0          ; Set strlen to zero
10  Loop
11          LDRB    R2, [R0, R1]    ; Read n-th byte
12          ADD     R1, R1, #1      ; Increment strlen count
13          TEQ     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

LDRB

Read start of string + length byte (R0 + R1)
Does not change string pointer (R0)

Program: strlen.s (Revised)

```
 1  ; Find the length of a null terminated string
 2
 7  Main
 8          LDR     R0, =Data1      ; Load the address of the lookup table
 9          MOV     R1, #0          ; Set strlen to zero
10  Loop
11          LDRB    R2, [R0, R1]    ; Read n-th byte
12          ADD     R1, R1, #1      ; Increment strlen count
13          TEQ     R2, #0          ; Is it the terminator ?
14          BNE     Loop            ; No => Next char
15
16          STR     R1, CharCount ; Store result
17          SWI     &11
18
19          AREA    Data1, DATA
22          DCB     "Hello, World", 0
```

TEQ — Test Equality rather than full compare

Program: skipblanks.s

```
 5  Blank EQU    "  "
 8
 9  Main
10          LDR     R0, =Data1   ; load the address of the lookup table
11          MOV     R1, #Blank   ; store the blank char in R1
12  Loop
13          LDRB    R2, [R0], #1 ; load the first byte into R2
14          CMP     R2, R1       ; is it a blank
15          BEQ     Loop         ; if so loop
16
17          SUB     R0, R0, #1   ; otherwise done - adjust pointer
18          STR     R0, Pointer  ; and store it
19          SWI     &11
20
21          AREA    Data1, DATA
24          DCB     "       7    "
```

Program: skipblanks.s

```
 5  Blank EQU    "  "
 8
 9  Main
10          LDR     R0, =Data1   ; load the address of the lookup table
11          MOV     R1, #Blank   ; store the blank char in R1
12  Loop
13          LDRB    R2, [R0], #1 ; load the first byte into R2
14          CMP     R2, R1       ; is it a blank
15          BEQ     Loop         ; if so loop
16
17          SUB     R0, R0, #1   ; otherwise done - adjust pointer
18          STR     R0, Pointer  ; and store it
19          SWI     &11
20
21          AREA    Data1, DATA
24          DCB     "       7    "
```

EQU Define `Blank` to be a space character

Program: skipblanks.s

```
 5  Blank EQU    "  "
 8
 9  Main
10          LDR     R0, =Data1   ; load the address of the lookup table
11          MOV     R1, #Blank   ; store the blank char in R1
12  Loop
13          LDRB    R2, [R0], #1 ; load the first byte into R2
14          CMP     R2, R1       ; is it a blank
15          BEQ     Loop         ; if so loop
16
17          SUB     R0, R0, #1   ; otherwise done - adjust pointer
18          STR     R0, Pointer  ; and store it
19          SWI     &11
20
21          AREA    Data1, DATA
24          DCB     "       7    "
```

MOV R1 is the character to ignore

Program: skipblanks.s

```
 5  Blank EQU    "  "
 8
 9  Main
10          LDR     R0, =Data1   ; load the address of the lookup table
11          MOV     R1, #Blank   ; store the blank char in R1
12  Loop
13          LDRB    R2, [R0], #1 ; load the first byte into R2
14          CMP     R2, R1       ; is it a blank
15          BEQ     Loop         ; if so loop
16
17          SUB     R0, R0, #1   ; otherwise done - adjust pointer
18          STR     R0, Pointer  ; and store it
19          SWI     &11
20
21          AREA    Data1, DATA
24          DCB     "       7    "
```

Eat leading blank's

Program: skipblanks.s

```
 5  Blank EQU    "  "
 8
 9  Main
10          LDR     R0, =Data1   ; load the address of the lookup table
11          MOV     R1, #Blank   ; store the blank char in R1
12  Loop
13          LDRB    R2, [R0], #1 ; load the first byte into R2
14          CMP     R2, R1       ; is it a blank
15          BEQ     Loop         ; if so loop
16
17          SUB     R0, R0, #1   ; otherwise done - adjust pointer
18          STR     R0, Pointer  ; and store it
19          SWI     &11
20
21          AREA    Data1, DATA
24          DCB     "       7    "
```

SUB

Loop always read one character ahead
to check if the character is to be ignored
So back up pointer by one character

Program: padzero.s

```
 5   Blank EQU   ’  ’
 6   Zero  EQU   ’0’
10   Main
11         LDR   R0, =Data1   ; load the address of the lookup table
12         MOV   R1, #Zero    ; store the zero char in R1
13         MOV   R3, #Blank   ; and the blank char in R3
14   Loop
15         LDRB  R2, [R0], #1 ; load the first byte into R2
16         CMP   R2, R1       ; is it a zero
17         BNE   Done         ; if not, done
19         SUB   R0, R0, #1   ; otherwise adjust the pointer
20         STRB  R3, [R0]     ; and store it blank char there
21         ADD   R0, R0, #1   ; otherwise adjust the pointer
22         BAL   Loop         ; and loop
23
29         DCB   "000007000"
```

Program: padzero.s

```
 5  Blank EQU   ’ ’
 6  Zero  EQU   ’0’
10  Main
11        LDR   R0, =Data1   ; load the address of the lookup table
12        MOV   R1, #Zero    ; store the zero char in R1
13        MOV   R3, #Blank   ; and the blank char in R3
14  Loop
15        LDRB  R2, [R0], #1 ; load the first byte into R2
16        CMP   R2, R1       ; is it a zero
17        BNE   Done         ; if not, done
19        SUB   R0, R0, #1   ; otherwise adjust the pointer
20        STRB  R3, [R0]     ; and store it blank char there
21        ADD   R0, R0, #1   ; otherwise adjust the pointer
22        BAL   Loop         ; and loop
23
29        DCB   "000007000"
```

EQU Define and use `Blank`

Program: padzero.s

```
 5  Blank EQU  ’ ’
 6  Zero  EQU  ’0’
10  Main
11          LDR   R0, =Data1   ; load the address of the lookup table
12          MOV   R1, #Zero    ; store the zero char in R1
13          MOV   R3, #Blank   ; and the blank char in R3
14  Loop
15          LDRB  R2, [R0], #1 ; load the first byte into R2
16          CMP   R2, R1       ; is it a zero
17          BNE   Done         ; if not, done
19          SUB   R0, R0, #1   ; otherwise adjust the pointer
20          STRB  R3, [R0]     ; and store it blank char there
21          ADD   R0, R0, #1   ; otherwise adjust the pointer
22          BAL   Loop         ; and loop
23
29          DCB   "000007000"
```

EQU Define and use `Zero`

Program: padzero.s

```
 5  Blank EQU   ’  ’
 6  Zero  EQU   ’0’
10  Main
11        LDR   R0, =Data1   ; load the address of the lookup table
12        MOV   R1, #Zero    ; store the zero char in R1
13        MOV   R3, #Blank   ; and the blank char in R3
14  Loop
15        LDRB  R2, [R0], #1 ; load the first byte into R2
16        CMP   R2, R1       ; is it a zero
17        BNE   Done         ; if not, done
19        SUB   R0, R0, #1   ; otherwise adjust the pointer
20        STRB  R3, [R0]     ; and store it blank char there
21        ADD   R0, R0, #1   ; otherwise adjust the pointer
22        BAL   Loop         ; and loop
23
29        DCB   "000007000"
```

ADD/SUB Adjust pointer

Program: padzero.s

```
 5  Blank EQU   ' '
 6  Zero  EQU   '0'
10  Main
11        LDR   R0, =Data1   ; load the address of the lookup table
12        MOV   R1, #Zero    ; store the zero char in R1
13        MOV   R3, #Blank   ; and the blank char in R3
14  Loop
15        LDRB  R2, [R0], #1 ; load the first byte into R2
16        CMP   R2, R1       ; is it a zero
17        BNE   Done         ; if not, done
19        SUB   R0, R0, #1   ; otherwise adjust the pointer
20        STRB  R3, [R0]     ; and store it blank char there
21        ADD   R0, R0, #1   ; otherwise adjust the pointer
22        BAL   Loop         ; and loop
23
29        DCB   "000007000"
```

STRB

```
STRB R3, [R0, #-1]
```

We don't need to adjust the pointer

Program: padzero.s (written in C)

- Current Version

```
while (1) {
    R2 = *(R0++);
    if ( R2 != R1 ) {
      break;
    }
    R0--;
    *R0 = R3;
    R0++;
}
```

Program: padzero.s (written in C)

- Current Version

```
while (1) {
    R2 = *(R0++);
    if ( R2 != R1 ) {
      break;
    }
    R0--;
    *R0 = R3;
    R0++;
}
```

- Revised Version

```
do {
    R2 = *R0;
    if ( R2 == R1 ) {
      *(R0++) = R3;
    }
} while ( R2 == R1 );
```

Program: padzero.s (Revised)

```
 5  Blank EQU     ’  ’
 6  Zero  EQU     ’0’
10  Main
11        LDR     R0, =Data1    ; load the address of the lookup table
12        MOV     R1, #Zero     ; store the zero char in R1
13        MOV     R3, #Blank    ; and the blank char in R3
14  Loop
15        LDRB    R2, [R0]      ; read byte at pointer
16        TEQ     R1, R2        ; is it a zero (what we are looking for)
17        STREQB  R3, [R0], #1  ; yes => replace with blank/inc pointer
19        BEQ     Loop          ; Yes => Repeat until not target
20
21
22
23
29        DCB     "000007000"
```

Program: setparity.s (Outer loop)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load char into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24 ;shift so that we are dealing with msb
                ⋮
28          TST     R3, #1              ;is the parity even
29          BEQ     Even                ;if so branch
30          ORR     R6, R6, #0x80       ;otherwise set the parity bit
31          STRB    R6, [R5], #1        ;and store the amended char
32          BAL     Check
33  Even    STRB    R6, [R5], #1        ;store unamended char if even
34  Check   SUBS    R1, R1, #1          ;decrement the character count
35          BNE     MainLoop
```

Program: setparity.s (Outer loop)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load char into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift so that we are dealing with msb
                    ⋮
28          TST     R3, #1              ;is the parity even
29          BEQ     Even                ;if so branch
30          ORR     R6, R6, #0x80       ;otherwise set the parity bit
31          STRB    R6, [R5], #1        ;and store the amended char
32          BAL     Check
33  Even    STRB    R6, [R5], #1        ;store unamended char if even
34  Check   SUBS    R1, R1, #1          ;decrement the character count
35          BNE     MainLoop
```

SUBS/BNE Repeat . . . Until R1 is zero

Program: setparity.s (Outer loop)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load char into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24 ;shift so that we are dealing with msb
                ⋮
28          TST     R3, #1              ;is the parity even
29          BEQ     Even                ;if so branch
30          ORR     R6, R6, #0x80       ;otherwise set the parity bit
31          STRB    R6, [R5], #1        ;and store the amended char
32          BAL     Check
33  Even    STRB    R6, [R5], #1        ;store unamended char if even
34  Check   SUBS    R1, R1, #1          ;decrement the character count
35          BNE     MainLoop
```

LSL Shift LSByte to MSByte (24 bits)

Program: setparity.s (Outer loop)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load char into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift so that we are dealing with msb
                    ⋮
28          TST     R3, #1              ;is the parity even
29          BEQ     Even                ;if so branch
30          ORR     R6, R6, #0x80       ;otherwise set the parity bit
31          STRB    R6, [R5], #1        ;and store the amended char
32          BAL     Check
33  Even    STRB    R6, [R5], #1        ;store unamended char if even
34  Check   SUBS    R1, R1, #1          ;decrement the character count
35          BNE     MainLoop
```

TST Is the parity even – test LSBit

Program: setparity.s (Outer loop)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load char into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24 ;shift so that we are dealing with msb
                ⋮
28          TST     R3, #1              ;is the parity even
29          BEQ     Even                ;if so branch
30          ORR     R6, R6, #0x80       ;otherwise set the parity bit
31          STRB    R6, [R5], #1        ;and store the amended char
32          BAL     Check
33  Even    STRB    R6, [R5], #1        ;store unamended char if even
34  Check   SUBS    R1, R1, #1          ;decrement the character count
35          BNE     MainLoop
```

BEQ/ORR — Using conditional execution we can write:

```
ORRNE   R6, R6, #0x80   ;add parity if odd
STRB    R6, [R5], #1    ;store char
```

Program: setparity.s (Inner loop)

```
18              MOV     R3, #0              ;zero the bit counter
19              MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22              MOVS    R2, R2, LSL #1      ;left shift
23              BPL     DontAdd             ;if msb is not a one bit, branch
24              ADD     R3, R3, #1          ;otherwise add to bit count
25  DontAdd
26              SUBS    R4, R4, #1          ;update shift count
27              BNE     ParLoop             ;loop if still bits to check
```

Program: setparity.s (Inner loop)

```
18                  MOV     R3, #0          ;zero the bit counter
19                  MOV     R4, #7          ;init the shift counter
20
21          ParLoop
22                  MOVS    R2, R2, LSL #1  ;left shift
23                  BPL     DontAdd         ;if msb is not a one bit, branch
24                  ADD     R3, R3, #1      ;otherwise add to bit count
25          DontAdd
26                  SUBS    R4, R4, #1      ;update shift count
27                  BNE     ParLoop         ;loop if still bits to check
```

SUBS/BNE Repeat . . . Until R4 is zero

Program: setparity.s (Inner loop)

```
18                  MOV     R3, #0          ;zero the bit counter
19                  MOV     R4, #7          ;init the shift counter
20
21          ParLoop
22                  MOVS    R2, R2, LSL #1  ;left shift
23                  BPL     DontAdd         ;if msb is not a one bit, branch
24                  ADD     R3, R3, #1      ;otherwise add to bit count
25          DontAdd
26                  SUBS    R4, R4, #1      ;update shift count
27                  BNE     ParLoop         ;loop if still bits to check
```

MOVS/LSL Move bits up one, MSB is echoed in Negative flag

Program: setparity.s (Inner loop)

```
18                  MOV     R3, #0          ;zero the bit counter
19                  MOV     R4, #7          ;init the shift counter
20
21          ParLoop
22                  MOVS    R2, R2, LSL #1  ;left shift
23                  BPL     DontAdd         ;if msb is not a one bit, branch
24                  ADD     R3, R3, #1      ;otherwise add to bit count
25          DontAdd
26                  SUBS    R4, R4, #1      ;update shift count
27                  BNE     ParLoop         ;loop if still bits to check
```

BPL Skip add if positive (MSB is zero)

Program: setparity.s (Inner loop)

```
18                  MOV     R3, #0              ;zero the bit counter
19                  MOV     R4, #7              ;init the shift counter
20
21          ParLoop
22                  MOVS    R2, R2, LSL #1      ;left shift
23                  BPL     DontAdd             ;if msb is not a one bit, branch
24                  ADD     R3, R3, #1          ;otherwise add to bit count
25          DontAdd
26                  SUBS    R4, R4, #1          ;update shift count
27                  BNE     ParLoop             ;loop if still bits to check
```

BPL/ADD Using conditional execution we can avoid the Brach:

```
ADDMI  R3, R3, #1  ;inc count if MSB set
```

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

SUBS/BNE Repeat . . . Until R1 is zero (outer loop)

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

LSL Shift LSByte to MSByte (24 bits)

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

SUBS/BNE Repeat . . . Until R4 is zero (inner loop)

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

MOVS/LSL Move bits up one, set Negative flag to MSB

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

ADDMI Increment parity counter if MSB is hi

Program: setparity.s (Revised)

```
14  MainLoop
15          LDRB    R2, [R0], #1        ;load the first byte into R2
16          MOV     R6, R2              ;keep a copy of the original char
17          MOV     R2, R2, LSL #24     ;shift to deal with msb
18          MOV     R3, #0              ;zero the bit counter
19          MOV     R4, #7              ;init the shift counter
20
21  ParLoop
22          MOVS    R2, R2, LSL #1      ;shift MSB into Neg flag
23          ADDMI   R3, R3, #1          ;inc bit count if MSB is set
24          SUBS    R4, R4, #1          ;update shift count
25          BNE     ParLoop             ;loop if still bits to check
26          TST     R3, #1              ;is the parity even
27          ORRNE   R6, R6, #0x80       ;set the parity bit if odd
28          STRB    R6, [R5], #1        ;and store the amended char
29          SUBS    R1, R1, #1          ;decrement the character count
20          BNE     MainLoop
```

TST Is the parity even – test LSBit

Program: cstrcmp.s

```
11          LDR     R2, Match    ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4 ;store the first string length in R3
13          LDR     R4, [R1], #4 ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done         ;if they are different lengths,
17          CMP     R3, #0       ;test for zero length if both are
18          BEQ     Same         ;zero length, nothing else to do
21  Loop
22          LDRB    R5, [R0], #1 ;character of first string
23          LDRB    R6, [R1], #1 ;character of second string
24          CMP     R5, R6       ;are they the same
25          BNE     Done         ;if not the strings are different
26          SUBS    R3, R3, #1   ;use the string length as a counter
27          BEQ     Same         ;if we got to the end of the count
28                               ;the strings are the same
29          BAL     Loop         ;not done, loop
31  Same    MOV     R2, #0       ;clear the -1 from match (0 = match)
32  Done    STR     R2, Match    ;store the result
```

Program: cstrcmp.s

```
11          LDR     R2, Match    ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4 ;store the first string length in R3
13          LDR     R4, [R1], #4 ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done         ;if they are different lengths,
17          CMP     R3, #0       ;test for zero length if both are
18          BEQ     Same         ;zero length, nothing else to do
21    Loop
22          LDRB    R5, [R0], #1 ;character of first string
23          LDRB    R6, [R1], #1 ;character of second string
24          CMP     R5, R6       ;are they the same
25          BNE     Done         ;if not the strings are different
26          SUBS    R3, R3, #1   ;use the string length as a counter
27          BEQ     Same         ;if we got to the end of the count
28                               ;the strings are the same
29          BAL     Loop         ;not done, loop
31    Same  MOV     R2, #0       ;clear the -1 from match (0 = match)
32    Done  STR     R2, Match    ;store the result
```

LDR Assume result is *false*

Program: cstrcmp.s

```
11          LDR     R2, Match    ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4 ;store the first string length in R3
13          LDR     R4, [R1], #4 ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done         ;if they are different lengths,
17          CMP     R3, #0       ;test for zero length if both are
18          BEQ     Same         ;zero length, nothing else to do
21    Loop
22          LDRB    R5, [R0], #1 ;character of first string
23          LDRB    R6, [R1], #1 ;character of second string
24          CMP     R5, R6       ;are they the same
25          BNE     Done         ;if not the strings are different
26          SUBS    R3, R3, #1   ;use the string length as a counter
27          BEQ     Same         ;if we got to the end of the count
28                               ;the strings are the same
29          BAL     Loop         ;not done, loop
31    Same  MOV     R2, #0       ;clear the -1 from match (0 = match)
32    Done  STR     R2, Match    ;store the result
```

#4 We are using counted strings with a 4-byte count

Program: cstrcmp.s

```
11          LDR     R2, Match    ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4 ;store the first string length in R3
13          LDR     R4, [R1], #4 ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done         ;if they are different lengths,
17          CMP     R3, #0       ;test for zero length if both are
18          BEQ     Same         ;zero length, nothing else to do
21  Loop
22          LDRB    R5, [R0], #1 ;character of first string
23          LDRB    R6, [R1], #1 ;character of second string
24          CMP     R5, R6       ;are they the same
25          BNE     Done         ;if not the strings are different
26          SUBS    R3, R3, #1   ;use the string length as a counter
27          BEQ     Same         ;if we got to the end of the count
28                               ;the strings are the same
29          BAL     Loop         ;not done, loop
31  Same    MOV     R2, #0       ;clear the -1 from match (0 = match)
32  Done    STR     R2, Match    ;store the result
```

BNE Abort if string lengths differ

Program: cstrcmp.s

```
11          LDR     R2, Match     ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4  ;store the first string length in R3
13          LDR     R4, [R1], #4  ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done          ;if they are different lengths,
17          CMP     R3, #0        ;test for zero length if both are
18          BEQ     Same          ;zero length, nothing else to do
21    Loop
22          LDRB    R5, [R0], #1  ;character of first string
23          LDRB    R6, [R1], #1  ;character of second string
24          CMP     R5, R6        ;are they the same
25          BNE     Done          ;if not the strings are different
26          SUBS    R3, R3, #1    ;use the string length as a counter
27          BEQ     Same          ;if we got to the end of the count
28                                ;the strings are the same
29          BAL     Loop          ;not done, loop
31    Same  MOV     R2, #0        ;clear the -1 from match (0 = match)
32    Done  STR     R2, Match     ;store the result
```

BEQ Abort if no string to compare

Program: cstrcmp.s

```
11          LDR     R2, Match    ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4 ;store the first string length in R3
13          LDR     R4, [R1], #4 ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done         ;if they are different lengths,
17          CMP     R3, #0       ;test for zero length if both are
18          BEQ     Same         ;zero length, nothing else to do
21  Loop
22          LDRB    R5, [R0], #1 ;character of first string
23          LDRB    R6, [R1], #1 ;character of second string
24          CMP     R5, R6       ;are they the same
25          BNE     Done         ;if not the strings are different
26          SUBS    R3, R3, #1   ;use the string length as a counter
27          BEQ     Same         ;if we got to the end of the count
28                               ;the strings are the same
29          BAL     Loop         ;not done, loop
31  Same    MOV     R2, #0       ;clear the -1 from match (0 = match)
32  Done    STR     R2, Match    ;store the result
```

BNE Exit on first difference found

Program: cstrcmp.s

```
11          LDR     R2, Match    ;assume strings not equal – set to -1
12          LDR     R3, [R0], #4 ;store the first string length in R3
13          LDR     R4, [R1], #4 ;store the second string length in R4
14          CMP     R3, R4
15          BNE     Done         ;if they are different lengths,
17          CMP     R3, #0       ;test for zero length if both are
18          BEQ     Same         ;zero length, nothing else to do
21    Loop
22          LDRB    R5, [R0], #1 ;character of first string
23          LDRB    R6, [R1], #1 ;character of second string
24          CMP     R5, R6       ;are they the same
25          BNE     Done         ;if not the strings are different
26          SUBS    R3, R3, #1   ;use the string length as a counter
27          BEQ     Same         ;if we got to the end of the count
28                               ;the strings are the same
29          BAL     Loop         ;not done, loop
31    Same  MOV     R2, #0       ;clear the -1 from match (0 = match)
32    Done  STR     R2, Match    ;store the result
```

Number of registers used: R0, R1, R2, R3, R4, R5, R6

Program: strcmp.s

```
 9            LDR     R0, =Data1  ;load the address of the lookup table
10            LDR     R1, =Data2
11            LDR     R2, Match   ;assume strings not equal, set to -1
12            MOV     R3, #0      ;init register
13            MOV     R4, #0
14  Count1
15            LDRB    R5, [R0], #1 ;load the first byte into R5
16            CMP     R5, #0      ;is it the terminator
17            BEQ     Count2      ;if not, Loop
18            ADD     R3, R3, #1  ;increment count
19            BAL     Count1
20  Count2
21            LDRB    R5, [R1], #1 ;load the first byte into R5
22            CMP     R5, #0      ;is it the terminator
23            BEQ     Next        ;if not, Loop
24            ADD     R4, R4, #1  ;increment count
25            BAL     Count2
```

Program: strcmp.s

```
 9            LDR     R0, =Data1  ;load the address of the lookup table
10            LDR     R1, =Data2
11            LDR     R2, Match   ;assume strings not equal, set to -1
12            MOV     R3, #0      ;init register
13            MOV     R4, #0
14  Count1
15            LDRB    R5, [R0], #1 ;load the first byte into R5
16            CMP     R5, #0      ;is it the terminator
17            BEQ     Count2      ;if not, Loop
18            ADD     R3, R3, #1  ;increment count
19            BAL     Count1
20  Count2
21            LDRB    R5, [R1], #1 ;load the first byte into R5
22            CMP     R5, #0      ;is it the terminator
23            BEQ     Next        ;if not, Loop
24            ADD     R4, R4, #1  ;increment count
25            BAL     Count2
```

LDR Assume result is *false*

Program: strcmp.s

```
 9          LDR     R0, =Data1  ;load the address of the lookup table
10          LDR     R1, =Data2
11          LDR     R2, Match   ;assume strings not equal, set to -1
12          MOV     R3, #0      ;init register
13          MOV     R4, #0
14  Count1
15          LDRB    R5, [R0], #1 ;load the first byte into R5
16          CMP     R5, #0      ;is it the terminator
17          BEQ     Count2      ;if not, Loop
18          ADD     R3, R3, #1  ;increment count
19          BAL     Count1
20  Count2
21          LDRB    R5, [R1], #1 ;load the first byte into R5
22          CMP     R5, #0      ;is it the terminator
23          BEQ     Next        ;if not, Loop
24          ADD     R4, R4, #1  ;increment count
25          BAL     Count2
```

strlen(table1)
Calculate length of first string

Program: strcmp.s

```
 9          LDR     R0, =Data1  ;load the address of the lookup table
10          LDR     R1, =Data2
11          LDR     R2, Match   ;assume strings not equal, set to -1
12          MOV     R3, #0      ;init register
13          MOV     R4, #0
14  Count1
15          LDRB    R5, [R0], #1 ;load the first byte into R5
16          CMP     R5, #0      ;is it the terminator
17          BEQ     Count2      ;if not, Loop
18          ADD     R3, R3, #1  ;increment count
19          BAL     Count1
20  Count2
21          LDRB    R5, [R1], #1 ;load the first byte into R5
22          CMP     R5, #0      ;is it the terminator
23          BEQ     Next        ;if not, Loop
24          ADD     R4, R4, #1  ;increment count
25          BAL     Count2
```

strlen(table2)
Calculate length of second string

Program: strcmp.s

```
 9          LDR     R0, =Data1  ;load the address of the lookup table
10          LDR     R1, =Data2
11          LDR     R2, Match   ;assume strings not equal, set to -1
12          MOV     R3, #0      ;init register
13          MOV     R4, #0
14  Count1
15          LDRB    R5, [R0], #1 ;load the first byte into R5
16          CMP     R5, #0      ;is it the terminator
17          BEQ     Count2      ;if not, Loop
18          ADD     R3, R3, #1  ;increment count
19          BAL     Count1
20  Count2
21          LDRB    R5, [R1], #1 ;load the first byte into R5
22          CMP     R5, #0      ;is it the terminator
23          BEQ     Next        ;if not, Loop
24          ADD     R4, R4, #1  ;increment count
25          BAL     Count2
```

BEQ/ADD Conditional Execution:

```
ADDNE  Rx, Rx, #1  ; inc string length
```

Program: strcmp.s

```
 9            LDR     R0, =Data1  ;load the address of the lookup table
10            LDR     R1, =Data2
11            LDR     R2, Match   ;assume strings not equal, set to -1
12            MOV     R3, #0      ;init register
13            MOV     R4, #0
14  Count1
15            LDRB    R5, [R0], #1 ;load the first byte into R5
16            CMP     R5, #0      ;is it the terminator
17            BEQ     Count2      ;if not, Loop
18            ADD     R3, R3, #1  ;increment count
19            BAL     Count1
20  Count2
21            LDRB    R5, [R1], #1 ;load the first byte into R5
22            CMP     R5, #0      ;is it the terminator
23            BEQ     Next        ;if not, Loop
24            ADD     R4, R4, #1  ;increment count
25            BAL     Count2
```

Compared to two `LDR` instructions in `cstrcmp.s`

Program: strcmp.s

```
 9            LDR     R0, =Data1  ;load the address of the lookup table
10            LDR     R1, =Data2
11            LDR     R2, Match   ;assume strings not equal, set to -1
12            MOV     R3, #0      ;init register
13            MOV     R4, #0
14  Count1
15            LDRB    R5, [R0], #1 ;load the first byte into R5
16            CMP     R5, #0      ;is it the terminator
17            BEQ     Count2      ;if not, Loop
18            ADD     R3, R3, #1  ;increment count
19            BAL     Count1
20  Count2
21            LDRB    R5, [R1], #1 ;load the first byte into R5
22            CMP     R5, #0      ;is it the terminator
23            BEQ     Next        ;if not, Loop
24            ADD     R4, R4, #1  ;increment count
25            BAL     Count2
```

! Counted strings are easier to work with !