ARM Assembler

Structure / Loops

Loops

- Four parts to any loop
  - i) *initialisation* ⟨*init*⟩
  - ii) *body* ⟨*body*⟩
  - iii) *increment* or *decrement* ⟨*inc*⟩ ⟨*dec*⟩
  - iv) *exit condition* ⟨*cond*⟩
- Three basic types of loop
  - i) Repeat . . . Until
  - ii) While
  - iii) Counted

The Repeat . . . Until Loop

- Exit condition tested at end of loop
- Loop is executed at least once

Pseudocode:

⟨*init*⟩
Repeat
    ⟨*body*⟩
    ⟨*inc*⟩/⟨*dec*⟩
Until ⟨*cond*⟩

The Repeat . . . Until Loop

- Exit condition tested at end of loop
- Loop is executed at least once

Pseudocode:

⟨*init*⟩
Repeat
  ⟨*body*⟩
  ⟨*inc*⟩/⟨*dec*⟩
Until ⟨*cond*⟩

C:

```
⟨init⟩;
do {
  ⟨body⟩;
  ⟨inc⟩/⟨dec⟩;
} while (⟨cond⟩);
```

The Repeat . . . Until Loop

- Exit condition tested at end of loop
- Loop is executed at least once

Pseudocode:

⟨*init*⟩
Repeat
  ⟨*body*⟩
  ⟨*inc*⟩/⟨*dec*⟩
Until ⟨*cond*⟩

C:

⟨*init*⟩;
`do {`
  ⟨*body*⟩;
  ⟨*inc*⟩/⟨*dec*⟩;
`} while (`⟨*cond*⟩`);`

Assembler:

  ⟨*init*⟩
label ⟨*body*⟩
  ⟨*inc*⟩/⟨*dec*⟩
  ⟨*cond*⟩
  B⟨*cc*⟩ label

The While Loop

- Exit condition tested at start of loop
- Loop may never be executed (zero or more times)

Pseudocode:

⟨*init*⟩
While ⟨*cond*⟩
    ⟨*body*⟩
    ⟨*inc*⟩/⟨*dec*⟩
End While

The While Loop

- Exit condition tested at start of loop
- Loop may never be executed (zero or more times)

Pseudocode:

⟨*init*⟩
While ⟨*cond*⟩
  ⟨*body*⟩
  ⟨*inc*⟩/⟨*dec*⟩
End While

C:

```
⟨init⟩;
while (⟨cond⟩) {
  ⟨body⟩;
  ⟨inc⟩/⟨dec⟩;
}
```

The While Loop

- Exit condition tested at start of loop
- Loop may never be executed (zero or more times)

Pseudocode:

⟨*init*⟩
While ⟨*cond*⟩
  ⟨*body*⟩
  ⟨*inc*⟩/⟨*dec*⟩
End While

C:

⟨*init*⟩;
`while (`⟨*cond*⟩`) {`
  ⟨*body*⟩;
  ⟨*inc*⟩/⟨*dec*⟩;
`}`

Assembler:

⟨*init*⟩
label1 ⟨*cond*⟩
B⟨*cc*⟩ label2
⟨*body*⟩
⟨*inc*⟩/⟨*dec*⟩
BAL label1
label2 . . .

The Counted Loop

Up Counting

- Pseudocode:

For ⟨*init*⟩ **up to** ⟨*cond*⟩
    ⟨*body*⟩
Next

Down Counting

For ⟨*init*⟩ **down to** ⟨*cond*⟩
    ⟨*body*⟩
Next

The Counted Loop

Up Counting

- Pseudocode:

For ⟨*init*⟩ **up to** ⟨*cond*⟩
  ⟨*body*⟩
Next

- C:

```
for (⟨init⟩; ⟨cond⟩; ⟨inc⟩) {
  ⟨body⟩;
}
```

Down Counting

For ⟨*init*⟩ **down to** ⟨*cond*⟩
  ⟨*body*⟩
Next

```
for (⟨init⟩; ⟨cond⟩; ⟨dec⟩) {
  ⟨body⟩;
}
```

The Counted Loop

Up Counting

- Pseudocode:

  For ⟨*init*⟩ **up to** ⟨*cond*⟩
  ⟨*body*⟩
  Next

- C:

```
for (⟨init⟩; ⟨cond⟩; ⟨inc⟩) {
  ⟨body⟩;
}
```

- Assembler:

```
        ⟨init⟩
label   ⟨body⟩
        ⟨inc⟩
        ⟨cond⟩
        B⟨cc⟩  label
```

Down Counting

For ⟨*init*⟩ **down to** ⟨*cond*⟩
⟨*body*⟩
Next

```
for (⟨init⟩; ⟨cond⟩; ⟨dec⟩) {
  ⟨body⟩;
}
```

```
        ⟨init⟩
label   ⟨body⟩
        ⟨dec⟩
        ⟨cond⟩
        B⟨cc⟩  label
```

Up or Down Counter

- Zero is easy to detect
- When counting down we can merge the *decrement* and ⟨*cond*⟩ code into a single `subs` instruction.

Up Counting

```
        LDR   R0, =Table
        MOV   R1, #0
        MOV   R2, #0
Loop    LDRB  R3, [R0]
        ADD   R1, R1, R3
        ADD   R0, R0, #1
        ADD   R2, R2, #1
        CMP   R2, #10
        BLT   Loop
```

Down Counting

```
        LDR   R0, =Table
        MOV   R1, #0
        MOV   R2, #10
Loop    LDRB  R3, [R0]
        ADD   R1, R1, R3
        ADD   R0, R0, #1
        SUBS  R2, R2, #1
        BNE   Loop
```

Up or Down Counter

- Zero is easy to detect
- When counting down we can merge the *decrement* and ⟨*cond*⟩ code into a single `subs` instruction.

Up Counting

```
        LDR   R0, =Table
        MOV   R1, #0
        MOV   R2, #0
Loop    LDRB  R3, [R0]
        ADD   R1, R1, R3
        ADD   R0, R0, #1
        ADD   R2, R2, #1
        CMP   R2, #10
        BLT   Loop
```

Down Counting

```
        LDR   R0, =Table
        MOV   R1, #0
        MOV   R2, #10
Loop    LDRB  R3, [R0]
        ADD   R1, R1, R3
        ADD   R0, R0, #1
        SUBS  R2, R2, #1
        BNE   Loop
```

Up or Down Counter

- Zero is easy to detect
- When counting down we can merge the *decrement* and ⟨*cond*⟩ code into a single `subs` instruction.

Up Counting

```
        LDR   R0, =Table
        MOV   R1, #0
        MOV   R2, #0
Loop    LDRB  R3, [R0]
        ADD   R1, R1, R3
        ADD   R0, R0, #1
        ADD   R2, R2, #1
        CMP   R2, #10
        BLT   Loop
```

Down Counting

```
        LDR   R0, =Table
        MOV   R1, #0
        MOV   R2, #10
Loop    LDRB  R3, [R0]
        ADD   R1, R1, R3
        ADD   R0, R0, #1
        SUBS  R2, R2, #1
        BNE   Loop
```

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

EOR Quick way of setting R1 to zero

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

Loop Label the next instruction

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

ADD Move pointer (R0) to next word

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

LDR/ADD Using Post-index addressing we can remove the ADD:

```
LDR R3, [R0], #4
```

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

SUBS Subtract and set flags
Decrement loop counter, R2

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

BNE *Branch* to *Loop* if counter is *n*ot *e*qual to zero

Program: sum16.s

```
 7  Main
 8          LDR     R0, =Data1  ;load the address of the lookup table
 9          EOR     R1, R1, R1  ;clear R1 to store sum
10          LDR     R2, Length  ;init element count
11  Loop
12          LDR     R3, [R0]    ;get the data
13          ADD     R1, R1, R3  ;add it to r1
14          ADD     R0, R0, #+4 ;increment pointer
15          SUBS    R2, R2, #1  ;decrement count with zero set
16          BNE     Loop        ;if zero flag is not set, loop
19
22  Table   DCW     &2040       ;table of values to be added
24          DCW     &1C22
28  TablEnd DCD     0
29
31  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

DCW Assembler will calculate the length of data table for me

Program: sum16b.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store sum
11          LDR     R2, Length      ;init element count
12          CMP     R2, #0          ;zero length table ?
13          BEQ     Done            ;yes => skip over sum loop
14  Loop
15          LDR     R3, [R0]        ;get the data that R0 points to
16          ADD     R1, R1, R3      ;add it to R1
17          ADD     R0, R0, #+4     ;increment pointer
18          SUBS    R2, R2, #0x1    ;decrement count with zero set
19          BNE     Loop            ;if zero flag is not set, loop
20  Done
21          STR     R1, Result      ;otherwise done - store result
22          SWI     &11
```

Program: sum16b.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store sum
11          LDR     R2, Length      ;init element count
12          CMP     R2, #0          ;zero length table ?
13          BEQ     Done            ;yes => skip over sum loop
14  Loop
15          LDR     R3, [R0]        ;get the data that R0 points to
16          ADD     R1, R1, R3      ;add it to R1
17          ADD     R0, R0, #+4     ;increment pointer
18          SUBS    R2, R2, #0x1    ;decrement count with zero set
19          BNE     Loop            ;if zero flag is not set, loop
20  Done
21          STR     R1, Result      ;otherwise done - store result
22          SWI     &11
```

EOR Quick way of setting R1 to zero

Program: sum16b.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store sum
11          LDR     R2, Length      ;init element count
12          CMP     R2, #0          ;zero length table ?
13          BEQ     Done            ;yes => skip over sum loop
14  Loop
15          LDR     R3, [R0]        ;get the data that R0 points to
16          ADD     R1, R1, R3      ;add it to R1
17          ADD     R0, R0, #+4     ;increment pointer
18          SUBS    R2, R2, #0x1    ;decrement count with zero set
19          BNE     Loop            ;if zero flag is not set, loop
20  Done
21          STR     R1, Result      ;otherwise done - store result
22          SWI     &11
```

CMP Is table length zero?

Program: sum16b.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store sum
11          LDR     R2, Length      ;init element count
12          CMP     R2, #0          ;zero length table ?
13          BEQ     Done            ;yes => skip over sum loop
14  Loop
15          LDR     R3, [R0]        ;get the data that R0 points to
16          ADD     R1, R1, R3      ;add it to R1
17          ADD     R0, R0, #+4     ;increment pointer
18          SUBS    R2, R2, #0x1    ;decrement count with zero set
19          BNE     Loop            ;if zero flag is not set, loop
20  Done
21          STR     R1, Result      ;otherwise done - store result
22          SWI     &11
```

BEQ Skip zero length tables

Protects from processing an empty list

Program: sum16b.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store sum
11          LDR     R2, Length      ;init element count
12          CMP     R2, #0          ;zero length table ?
13          BEQ     Done            ;yes => skip over sum loop
14  Loop
15          LDR     R3, [R0]        ;get the data that R0 points to
16          ADD     R1, R1, R3      ;add it to R1
17          ADD     R0, R0, #+4     ;increment pointer
18          SUBS    R2, R2, #0x1    ;decrement count with zero set
19          BNE     Loop            ;if zero flag is not set, loop
20  Done
21          STR     R1, Result      ;otherwise done - store result
22          SWI     &11
```

LDR/ADD Using Post-index addressing we can remove the ADD:

```
LDR R3, [R0], #4
```

Program: sum16b.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store sum
11          LDR     R2, Length      ;init element count
12          CMP     R2, #0          ;zero length table ?
13          BEQ     Done            ;yes => skip over sum loop
14  Loop
15          LDR     R3, [R0]        ;get the data that R0 points to
16          ADD     R1, R1, R3      ;add it to R1
17          ADD     R0, R0, #+4     ;increment pointer
18          SUBS    R2, R2, #0x1    ;decrement count with zero set
19          BNE     Loop            ;if zero flag is not set, loop
20  Done
21          STR     R1, Result      ;otherwise done - store result
22          SWI     &11
```

SUBS/BNE Decrement counter and branch to *Loop* if not zero

Program: countneg.s

```
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is table empty
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, #0          ;is it positive?
16          BPL     Looptest        ;yes => skip next line
17          ADD     R1, R1, #1      ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;until count is zero
27
28  Table   DCD     &F1522040       ;table of values to be added
31  TablEnd DCD     0
34  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

Program: countneg.s

```
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is table empty
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, #0          ;is it positive?
16          BPL     Looptest        ;yes => skip next line
17          ADD     R1, R1, #1      ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;until count is zero
27
28  Table   DCD     &F1522040       ;table of values to be added
31  TablEnd DCD     0
34  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

CMP/BEQ Skip zero length tables

Program: countneg.s

```
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is table empty
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, #0          ;is it positive?
16          BPL     Looptest        ;yes => skip next line
17          ADD     R1, R1, #1      ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;until count is zero
27
28  Table   DCD     &F1522040       ;table of values to be added
31  TablEnd DCD     0
34  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

BPL Brach if positive (plus)

Program: countneg.s

```
10              LDR     R2, Length      ;init element count
11              CMP     R2, #0          ;is table empty
12              BEQ     Done            ;yes => skip loop
13      Loop
14              LDR     R3, [R0]        ;get the data
15              CMP     R3, #0          ;is it positive?
16              BPL     Looptest        ;yes => skip next line
17              ADD     R1, R1, #1      ;increment -ve number count
18      Looptest
19              ADD     R0, R0, #+4     ;increment pointer
20              SUBS    R2, R2, #0x1    ;decrement count with zero set
21              BNE     Loop            ;until count is zero
27
28      Table   DCD     &F1522040       ;table of values to be added
31      TablEnd DCD     0
34      Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

BPL/ADD

Using Conditional Execution we can write:

```
ADDMI R1, R1, #1 ;inc -ve count if -ve
```

This would be faster, as it does not flush the pipeline

Program: countneg.s

```
10              LDR     R2, Length      ;init element count
11              CMP     R2, #0          ;is table empty
12              BEQ     Done            ;yes => skip loop
13      Loop
14              LDR     R3, [R0]        ;get the data
15              CMP     R3, #0          ;is it positive?
16              BPL     Looptest        ;yes => skip next line
17              ADD     R1, R1, #1      ;increment -ve number count
18      Looptest
19              ADD     R0, R0, #+4     ;increment pointer
20              SUBS    R2, R2, #0x1    ;decrement count with zero set
21              BNE     Loop            ;until count is zero
27
28      Table   DCD     &F1522040       ;table of values to be added
31      TablEnd DCD     0
34      Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

LDR/ADD Move to next word, can be merged into one:

```
LDR R3, [R0], #4 ; get next value
```

Program: countneg.s

```
10              LDR     R2, Length      ;init element count
11              CMP     R2, #0          ;is table empty
12              BEQ     Done            ;yes => skip loop
13      Loop
14              LDR     R3, [R0]        ;get the data
15              CMP     R3, #0          ;is it positive?
16              BPL     Looptest        ;yes => skip next line
17              ADD     R1, R1, #1      ;increment -ve number count
18      Looptest
19              ADD     R0, R0, #+4     ;increment pointer
20              SUBS    R2, R2, #0x1    ;decrement count with zero set
21              BNE     Loop            ;until count is zero
27
28      Table   DCD     &F1522040       ;table of values to be added
31      TablEnd DCD     0
34      Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

SUBS/BNE Decrement counter and branch to *Loop* if not zero

Program: countneg.s

```
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is table empty
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, #0          ;is it positive?
16          BPL     Looptest        ;yes => skip next line
17          ADD     R1, R1, #1      ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;until count is zero
27
28  Table   DCD     &F1522040       ;table of values to be added
31  TablEnd DCD     0
34  Length  DCW     (TablEnd - Table) / 4 ;because we're having to align
```

DCW Assembler will calculate the length of data table

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1          ;load the address of the lookup table
10          EOR     R1, R1, R1          ;clear R1 to store count
11          LDR     R2, Length          ;init element count
12          TEQ     R2, #0              ;is count zero?
13          BEQ     Done                ;yes => skip loop
14  Loop
15          LDR     R3, [R0]            ;get the data
16          AND     R3, R3, #0x8000     ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000         ;bit is 1
18          BNE     Looptest            ;skip next line if zero
19          ADD     R1, R1, #1          ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4         ;increment pointer
22          SUBS    R2, R2, #0x1        ;decrement count with zero set
23          BNE     Loop                ;if zero flag is not set, loop
```

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1          ;load the address of the lookup table
10          EOR     R1, R1, R1          ;clear R1 to store count
11          LDR     R2, Length          ;init element count
12          TEQ     R2, #0              ;is count zero?
13          BEQ     Done                ;yes => skip loop
14  Loop
15          LDR     R3, [R0]            ;get the data
16          AND     R3, R3, #0x8000     ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000         ;bit is 1
18          BNE     Looptest            ;skip next line if zero
19          ADD     R1, R1, #1          ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4         ;increment pointer
22          SUBS    R2, R2, #0x1        ;decrement count with zero set
23          BNE     Loop                ;if zero flag is not set, loop
```

TEQ Test R2 for 0

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

TEQ/BEQ Protect loop from zero length tables

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

AND Clear all but halfword sign
Reset lower 15 bits

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

CMP Is halfword sign bit (bit 15) set (negative)

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

BNE Skip to *Looptest* if value is positive

Program: countneg16.s

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

SUBS/BNE — Using subtract and set to automatically detect zero
Branch to *Loop* if counter is not zero

Program: countneg16.s (Revised)

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

LDR/ADD

By using Post-Index addressing we can write:

```
LDR R3, [R0], #4 ; read data and move on
```

Program: countneg16.s (Revised)

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

AND/CMP — Using the “Set” option we can combine these into:

```
ANDS R3, R3, #0x8000
```

Zero flag is set if R3 is positive

Program: countneg16.s (Revised)

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

BEQ/ADD — Using conditional execution we can avoid the branch:

```
ADDEQ R1, R1, #1 ; Inc counter if -ve
```

Program: countneg16.s (Revised)

```
 8  Main
 9          LDR     R0, =Data1      ;load the address of the lookup table
10          EOR     R1, R1, R1      ;clear R1 to store count
11          LDR     R2, Length      ;init element count
12          TEQ     R2, #0          ;is count zero?
13          BEQ     Done            ;yes => skip loop
14  Loop
15          LDR     R3, [R0]        ;get the data
16          AND     R3, R3, #0x8000 ;bit wise AND to see if the 16th
17          CMP     R3, #0x8000     ;bit is 1
18          BNE     Looptest        ;skip next line if zero
19          ADD     R1, R1, #1      ;increment -ve number count
20  Looptest
21          ADD     R0, R0, #+4     ;increment pointer
22          SUBS    R2, R2, #0x1    ;decrement count with zero set
23          BNE     Loop            ;if zero flag is not set, loop
```

LDRSH — If we could use LDRSH this would be as simple as countnet.s – but we can't

Program: countneg16.s (Revised)

```
 8  Main
 9          LDR     R0, =Data1          ;load the address of the lookup table
10          EOR     R1, R1, R1          ;clear R1 to store count
11          LDR     R2, Length          ;init element count
12          TEQ     R2, #0              ;is count zero?
13          BEQ     Done                ;yes => skip loop
14  Loop
15          LDR     R3, [R0], #4        ;get the data
16          ANDS    R3, R3, #0x8000     ;is halfword sign set
17
18
19          ADDNE   R1, R1, #1          ;increment -ve number count
20
21
22          SUBS    R2, R2, #0x1        ;decrement count with zero set
23          BNE     Loop                ;if zero flag is not set, loop
```

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

EOR Quick way of setting R1 to zero

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

CMP/BEQ Protect loop from empty list (zero length)

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

CMP Compare new value (R3) against current largest (R1)

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

BLS Branch if new is less than or same as current

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

SUBS/BNE Using subtract to automatically detect zero
Branch to *Loop* if counter is not zero

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

LDR/ADD

With Post-Index addressing we can write:

```
LDR R3, [R0], #4 ; read data and move on
```

Program: largest16.s

```
 7  Main
 8          LDR     R0, =Data1      ;load the address of the lookup table
 9          EOR     R1, R1, R1      ;clear R1 to store largest
10          LDR     R2, Length      ;init element count
11          CMP     R2, #0          ;is it an empty table
12          BEQ     Done            ;yes => skip loop
13  Loop
14          LDR     R3, [R0]        ;get the data
15          CMP     R3, R1          ;compare to largest
16          BLS     Looptest        ;skip next line if zero
17          MOV     R1, R3          ;increment -ve number count
18  Looptest
19          ADD     R0, R0, #+4     ;increment pointer
20          SUBS    R2, R2, #0x1    ;decrement count with zero set
21          BNE     Loop            ;if zero flag is not set, loop
22  Done
```

BLS/ADD — Using conditional execution we can avoid the branch:

```
MOVGT R1, R3 ; Save new current largest
```

Program: normalize.s

```
 1  ; normalize a binary number
 2
 7  Main
 8          LDR     R0, =Data1          ;load the address of the lookup table
 9          EOR     R1, R1, R1          ;clear R1 to store shift count
10          LDR     R3, [R0]            ;get the value to normalize
11          CMP     R3, R1              ;is it a non-zero value
12          BEQ     Done                ;yes => already normalised
13  Loop
14          ADD     R1, R1, #1          ;increment shift counter
15          MOVS    R3, R3, LSL #0x1    ;shift value by one bit
16          BPL     Loop                ;loop until upper bit (sign bit) set
17  Done
18          STR     R1, Shifted         ;otherwise done - store result
19          STR     R3, Normal
20          SWI     &11                 ;exit
```

Program: normalize.s

```
 1  ; normalize a binary number
 2
 7  Main
 8          LDR     R0, =Data1          ;load the address of the lookup table
 9          EOR     R1, R1, R1          ;clear R1 to store shift count
10          LDR     R3, [R0]            ;get the value to normalize
11          CMP     R3, R1              ;is it a non-zero value
12          BEQ     Done                ;yes => already normalised
13  Loop
14          ADD     R1, R1, #1          ;increment shift counter
15          MOVS    R3, R3, LSL #0x1    ;shift value by one bit
16          BPL     Loop                ;loop until upper bit (sign bit) set
17  Done
18          STR     R1, Shifted         ;otherwise done - store result
19          STR     R3, Normal
20          SWI     &11                 ;exit
```

EOR Quick way of setting R1 to zero

Program: normalize.s

```
 1  ; normalize a binary number
 2
 7  Main
 8          LDR     R0, =Data1          ;load the address of the lookup table
 9          EOR     R1, R1, R1          ;clear R1 to store shift count
10          LDR     R3, [R0]            ;get the value to normalize
11          CMP     R3, R1              ;is it a non-zero value
12          BEQ     Done                ;yes => already normalised
13  Loop
14          ADD     R1, R1, #1          ;increment shift counter
15          MOVS    R3, R3, LSL #0x1    ;shift value by one bit
16          BPL     Loop                ;loop until upper bit (sign bit) set
17  Done
18          STR     R1, Shifted         ;otherwise done - store result
19          STR     R3, Normal
20          SWI     &11                 ;exit
```

CMP/BEQ Protect from zero entry
Otherwise we will enter a never ending loop

Program: normalize.s

```
 1  ; normalize a binary number
 2
 7  Main
 8          LDR     R0, =Data1          ;load the address of the lookup table
 9          EOR     R1, R1, R1          ;clear R1 to store shift count
10          LDR     R3, [R0]            ;get the value to normalize
11          CMP     R3, R1              ;is it a non-zero value
12          BEQ     Done                ;yes => already normalised
13  Loop
14          ADD     R1, R1, #1          ;increment shift counter
15          MOVS    R3, R3, LSL #0x1    ;shift value by one bit
16          BPL     Loop                ;loop until upper bit (sign bit) set
17  Done
18          STR     R1, Shifted         ;otherwise done - store result
19          STR     R3, Normal
20          SWI     &11                 ;exit
```

ADD Increment shift counter

Program: normalize.s

```
 1  ; normalize a binary number
 2
 7  Main
 8          LDR     R0, =Data1          ;load the address of the lookup table
 9          EOR     R1, R1, R1          ;clear R1 to store shift count
10          LDR     R3, [R0]            ;get the value to normalize
11          CMP     R3, R1              ;is it a non-zero value
12          BEQ     Done                ;yes => already normalised
13  Loop
14          ADD     R1, R1, #1          ;increment shift counter
15          MOVS    R3, R3, LSL #0x1    ;shift value by one bit
16          BPL     Loop                ;loop until upper bit (sign bit) set
17  Done
18          STR     R1, Shifted         ;otherwise done - store result
19          STR     R3, Normal
20          SWI     &11                 ;exit
```

MOVS/LSL Shift value up by one bit and set flags

Program: normalize.s

```
 1  ; normalize a binary number
 2
 7  Main
 8          LDR     R0, =Data1          ;load the address of the lookup table
 9          EOR     R1, R1, R1          ;clear R1 to store shift count
10          LDR     R3, [R0]            ;get the value to normalize
11          CMP     R3, R1              ;is it a non-zero value
12          BEQ     Done                ;yes => already normalised
13  Loop
14          ADD     R1, R1, #1          ;increment shift counter
15          MOVS    R3, R3, LSL #0x1    ;shift value by one bit
16          BPL     Loop                ;loop until upper bit (sign bit) set
17  Done
18          STR     R1, Shifted         ;otherwise done - store result
19          STR     R3, Normal
20          SWI     &11                 ;exit
```

BPL Repeat until upper bit (sign bit) is positive