```
.ORIG x3000

;; Main program

; Set R6 as our stack pointer. R6 is the one register
; that cannot be set to anything else unless the stack is
; empty, otherwise data will be lost. If absolutely necessary, there is one
; label - SAVEDR6 - that is used in one subroutine.
AND R6, R6, #0
LEA R6, HEAD

;; Print input prompt for A
LEA R1, APROMPT
LDR R0, R1, #-0
TRAP x21
ADD R1, R1, #1
LDR R0, R1, #-0
BRnp -5

;;Input for A
LEA R1, ASTRING   ; Load Message address in R1
JSR INPUT

;Print newline
LEA R1, LINE
LDR R0, R1, #-0
TRAP x21

;;Print input prompt for B
LEA R1, BPROMPT
LDR R0, R1, #-0
TRAP x21
ADD R1, R1, #1
LDR R0, R1, #-0
BRnp -5
;;Input for B
LEA R1, BSTRING   ; Load Message address in R1
JSR INPUT
;Print newline
LEA R1, LINE
LDR R0, R1, #-0
TRAP x21


;; Input processing

;1. Sign extending A and B
LEA R1, ASTRING
JSR SIGNEXT
LEA R1, BSTRING
JSR SIGNEXT

;; NAND output
LEA R1, NANDPROMPT
LDR R0, R1, #-0
TRAP x21
ADD R1, R1, #1
LDR R0, R1, #-0
BRnp -5
LEA R1, ASTRING
```

```
LEA R2, BSTRING
JSR NAND
;Print newline
LEA R1, LINE
LDR R0, R1, #-0
TRAP x21

;; NOR output
LEA R1, NORPROMPT
LDR R0, R1, #-0
TRAP x21
ADD R1, R1, #1
LDR R0, R1, #-0
BRnp -5
LEA R1, ASTRING
LEA R2, BSTRING
JSR NOR
;Print newline
LEA R1, LINE
LDR R0, R1, #-0
TRAP x21

;; XOR output
LEA R1, XORPROMPT
LDR R0, R1, #-0
TRAP x21
ADD R1, R1, #1
LDR R0, R1, #-0
BRnp -5
LEA R1, ASTRING
LEA R2, BSTRING
JSR XOR
;Print newline
LEA R1, LINE
LDR R0, R1, #-0
TRAP x21


HALT


;;Labels
AHEX  .FILL x0000
BHEX  .FILL x0000
CHEX  .FILL x0000
APROMPT .STRINGZ "Enter A: "
BPROMPT .STRINGZ "Enter B: "
NANDPROMPT .STRINGZ "A NAND B: "
NORPROMPT .STRINGZ "A NOR B: "
XORPROMPT .STRINGZ "A XOR B: "

;;Run Time Stack, has to go here due to 256 limit from start of app
MAX .BLKW 30  ; Run Time stack limited to 30...
HEAD  .BLKW 1   ; + 1

;Data objects
TWOSCP  .FILL xFFF6     ;  Two's comp of ascii newline, 0A = F6
LINE    .FILL x000A ;  Printable newline
BEEP    .FILL x0007
ASTRING .BLKW 17  ;  Input limited to 16 chars + 1 term char x0000
```

```
BSTRING .BLKW 17  ;  "                        "
CSTRING .BLKW 17  ;  "                        "
OSTRING .BLKW 30  ; The output string
ZEROASCII .FILL x0030
ONEASCII .FILL x0031
XORSUM .FILL xFF9F  ; '1' + '0' = x61, 0110 0001 -> 1001 1111
SAVEDR6 .FILL x0000

;; More labels
LOWASCII .FILL xFFD0  ;  x0030 = xFFD0
HIGHASCII .FILL xFFCF ;  x0031 = xFFCF

;;Subroutines
INPUT
;; Input characters
;;R1 = string addess where input will be stored
;; Ex. on how to call
;; LEA R1, ASTRING    ; Load Message address in R1
;; JSR INPUT

;Push all registers onto the stack
ADD R6, R6, #-1   ; PUSH R0
STR R0, R6, #0
ADD R6, R6, #-1   ; PUSH R1
STR R1, R6, #0
ADD R6, R6, #-1   ; PUSH R2
STR R2, R6, #0
ADD R6, R6, #-1   ; PUSH R3
STR R3, R6, #0
ADD R6, R6, #-1   ; PUSH R4
STR R4, R6, #0
ADD R6, R6, #-1   ; PUSH R5
STR R5, R6, #0
ADD R6, R6, #-1   ; PUSH R7
STR R7, R6, #0

LEA R7, LOWASCII
LDR R4, R7, #-0
LEA R7, HIGHASCII
LDR R5, R7, #-0
LEA R7, TWOSCP          ;
LDR R2, R7, #-0         ; Load R2 with ascii test val

LEA R7, SAVEDR6
STR R6, R7, #0

AND R6, R6, #0    ; R6 will now keep track of how many chars
ADD R6, R6, #15
ADD R6, R6, #1    ; 16 max

TRAP x20                ; input
ADD R3, R0, R2          ; test to see if input was a newline
BRz 14
ADD R3, R0, R4
BRn 3                   ; Test char too low in val
ADD R3, R0, R5
BRp 1                   ; Test if char too high
BRnzp 4
LEA R0, BEEP
LDR R0, R0, #-0
```

```
  TRAP x21
  BRnzp -12
  TRAP x21                ; echo input
  STR R0, R1, #0          ; store input in add from R1
  ADD R1, R1, #1          ; increment R1
  ADD R6, R6, #-1   ; test value of R7
  BRnp -17               ; loop back

  LEA R7, SAVEDR6
  AND R6, R6, #0
  LDR R6, R7, #0

  ;Restore the registers
  LDR R7, R6, #0
  ADD R6, R6, #1    ; POP into R7
  LDR R5, R6, #0
  ADD R6, R6, #1    ; POP into R5
  LDR R4, R6, #0
  ADD R6, R6, #1    ; POP into R4
  LDR R3, R6, #0
  ADD R6, R6, #1    ; POP into R3
  LDR R2, R6, #0
  ADD R6, R6, #1    ; POP into R2
  LDR R1, R6, #0
  ADD R6, R6, #1    ; POP into R1
  LDR R0, R6, #0
  ADD R6, R6, #1    ; POP into R0

  RET

SIGNEXT

  ;Push all registers onto the stack
  ADD R6, R6, #-1   ; PUSH R0
  STR R0, R6, #0
  ADD R6, R6, #-1   ; PUSH R1
  STR R1, R6, #0
  ADD R6, R6, #-1   ; PUSH R2
  STR R2, R6, #0
  ADD R6, R6, #-1   ; PUSH R3
  STR R3, R6, #0
  ADD R6, R6, #-1   ; PUSH R4
  STR R4, R6, #0
  ADD R6, R6, #-1   ; PUSH R5
  STR R5, R6, #0
  ADD R6, R6, #-1   ; PUSH R7
  STR R7, R6, #0

  ;Count how many chars are in the string
  AND R3, R3, #0    ; R3 = the count/length
  ADD R2, R1, #0    ; Set R2 to beginning of string
  LDR R0, R2, #0
  BRz 3
  ADD R2, R2, #1
  ADD R3, R3, #1
  BRnzp -5

  ;Put string in CSTRING, and then put it back in original string,
  ;starting at the incremented top considering R3, the length
  ADD R2, R1, #0    ; set R2 to the beginning of string
```

```
  LEA R4, CSTRING
  LDR R0, R2, #0
  BRz 4
  STR R0, R4, #0
  ADD R2, R2, #1
  ADD R4, R4, #1
  BRnzp -6
  ;..go down the whole original string, setting all chars to the MSB
  ADD R2, R1, #0    ; Set R2 to beginning of the string
  AND R5, R5, #0
  ADD R5, R5, #15   ; Set R5 to string len
  ADD R5, R5, #1
  LEA R0, CSTRING   ; Load MSB of the string in R0
  LDR R0, R0, #0    ; and sign ext every time there is an iter
  STR R0, R2, #0
  ADD R2, R2, #1    ; Move R2 down the string block R5 times
  ADD R5, R5, #-1
  BRnp -4
  ;now move R2 up to the location that the string should be shifted
  AND R5, R5, #0
  ADD R5, R3, #0    ; Set R5 to string len
  ADD R2, R2, #-1
  ADD R5, R5, #-1
  BRnp -3
  ;and set the chars in R2 on to the chars in CSTRING
  AND R5, R5, #0    ; set R5 to count/len
  ADD R5, R3, #0
  LEA R4, CSTRING   ; set R4 to CSTRING beginning
  LDR R0, R4, #0    ; copy R4 to R2
  STR R0, R2, #0
  ADD R2, R2, #1    ; increment R2 and R4...
  ADD R4, R4, #1
  ADD R5, R5, #-1
  BRnp -6      ; iterate R5 (len) times

  ;Restore the registers
  LDR R7, R6, #0
  ADD R6, R6, #1    ; POP into R7
  LDR R5, R6, #0
  ADD R6, R6, #1    ; POP into R5
  LDR R4, R6, #0
  ADD R6, R6, #1    ; POP into R4
  LDR R3, R6, #0
  ADD R6, R6, #1    ; POP into R3
  LDR R2, R6, #0
  ADD R6, R6, #1    ; POP into R2
  LDR R1, R6, #0
  ADD R6, R6, #1    ; POP into R1
  LDR R0, R6, #0
  ADD R6, R6, #1    ; POP into R0

  RET

NAND
  ;; R1 arg is set to the first string, R2 arg set to the second
  ;Push all registers onto the stack
  ADD R6, R6, #-1   ; PUSH R0
  STR R0, R6, #0
  ADD R6, R6, #-1   ; PUSH R1
  STR R1, R6, #0
```

```
  ADD R6, R6, #-1   ; PUSH R2
  STR R2, R6, #0
  ADD R6, R6, #-1   ; PUSH R3
  STR R3, R6, #0
  ADD R6, R6, #-1   ; PUSH R4
  STR R4, R6, #0
  ADD R6, R6, #-1   ; PUSH R5
  STR R5, R6, #0
  ADD R6, R6, #-1   ; PUSH R7
  STR R7, R6, #0

  LEA R3, LOWASCII
  LDR R3, R3, #0    ; R3 = 2's comp of ascii '1'
  LEA R4, ZEROASCII
  LDR R4, R4, #0    ; R4 = '0'
  LEA R5, ONEASCII
  LDR R5, R5, #0    ; R5 = '1'

  LDR R0, R1, #0
  ADD R0, R3, R0
  BRz 6
  LDR R0, R2, #0    ; if R1 = 0, print 1
  ADD R0, R3, R0
  BRz 3     ; if R2 = 0, print 1
  ADD R0, R4, #0    ; else, print 0
  TRAP x21
  BRnzp 2
  ADD R0, R5, #0
  TRAP x21
  ADD R1, R1, #1    ; increment R1
  ADD R2, R2, #1    ; increment R2
  LDR R0, R1, #0
  BRnp -15

  ;Restore the registers
  LDR R7, R6, #0
  ADD R6, R6, #1    ; POP into R7
  LDR R5, R6, #0
  ADD R6, R6, #1    ; POP into R5
  LDR R4, R6, #0
  ADD R6, R6, #1    ; POP into R4
  LDR R3, R6, #0
  ADD R6, R6, #1    ; POP into R3
  LDR R2, R6, #0
  ADD R6, R6, #1    ; POP into R2
  LDR R1, R6, #0
  ADD R6, R6, #1    ; POP into R1
  LDR R0, R6, #0
  ADD R6, R6, #1    ; POP into R0

  RET

NOR
  ;; R1 arg is set to the first string, R2 arg set to the second
  ;Push all registers onto the stack
  ADD R6, R6, #-1   ; PUSH R0
  STR R0, R6, #0
  ADD R6, R6, #-1   ; PUSH R1
  STR R1, R6, #0
  ADD R6, R6, #-1   ; PUSH R2
```

```
  STR R2, R6, #0
  ADD R6, R6, #-1   ; PUSH R3
  STR R3, R6, #0
  ADD R6, R6, #-1   ; PUSH R4
  STR R4, R6, #0
  ADD R6, R6, #-1   ; PUSH R5
  STR R5, R6, #0
  ADD R6, R6, #-1   ; PUSH R7
  STR R7, R6, #0

  LEA R3, LOWASCII
  LDR R3, R3, #0    ; R3 = 2's comp of ascii '1'
  LEA R4, ZEROASCII
  LDR R4, R4, #0    ; R4 = '0'
  LEA R5, ONEASCII
  LDR R5, R5, #0    ; R5 = '1'

  LDR R0, R1, #0
  ADD R0, R3, R0
  BRnp 6
  LDR R0, R2, #0    ; if R1 = 1, print 0
  ADD R0, R3, R0
  BRnp 3       ; if R2 = 1, print 0
  ADD R0, R5, #0    ; print 1
  TRAP x21
  BRnzp 2
  ADD R0, R4, #0    ; print 0
  TRAP x21
  ADD R1, R1, #1    ; increment R1
  ADD R2, R2, #1    ; increment R2
  LDR R0, R1, #0
  BRnp -15

  ;Restore the registers
  LDR R7, R6, #0
  ADD R6, R6, #1    ; POP into R7
  LDR R5, R6, #0
  ADD R6, R6, #1    ; POP into R5
  LDR R4, R6, #0
  ADD R6, R6, #1    ; POP into R4
  LDR R3, R6, #0
  ADD R6, R6, #1    ; POP into R3
  LDR R2, R6, #0
  ADD R6, R6, #1    ; POP into R2
  LDR R1, R6, #0
  ADD R6, R6, #1    ; POP into R1
  LDR R0, R6, #0
  ADD R6, R6, #1    ; POP into R0

  RET

XOR
  ;; R1 arg is set to the first string, R2 arg set to the second
  ;Push all registers onto the stack
  ADD R6, R6, #-1   ; PUSH R0
  STR R0, R6, #0
  ADD R6, R6, #-1   ; PUSH R1
  STR R1, R6, #0
  ADD R6, R6, #-1   ; PUSH R2
  STR R2, R6, #0
```

```
  ADD R6, R6, #-1   ; PUSH R3
  STR R3, R6, #0
  ADD R6, R6, #-1   ; PUSH R4
  STR R4, R6, #0
  ADD R6, R6, #-1   ; PUSH R5
  STR R5, R6, #0
  ADD R6, R6, #-1   ; PUSH R7
  STR R7, R6, #0

  LEA R3, XORSUM
  LDR R3, R3, #0    ; R3 = 2's comp of ascii '1'
  LEA R4, ZEROASCII
  LDR R4, R4, #0    ; R4 = '0'
  LEA R5, ONEASCII
  LDR R5, R5, #0    ; R5 = '1'

  LDR R0, R1, #0
  LDR R7, R2, #0
  ADD R0, R0, R7    ; R0 = 61d if we want to print 1
  ADD R0, R3, R0
  BRz 3     ; print 1 if zero sum
  ADD R0, R4, #0    ; print 0
  TRAP x21
  BRnzp 2
  ADD R0, R5, #0    ; print 1
  TRAP x21
  ADD R1, R1, #1    ; increment R1
  ADD R2, R2, #1    ; increment R2
  LDR R0, R1, #0
  BRnp -14

  ;Restore the registers
  LDR R7, R6, #0
  ADD R6, R6, #1    ; POP into R7
  LDR R5, R6, #0
  ADD R6, R6, #1    ; POP into R5
  LDR R4, R6, #0
  ADD R6, R6, #1    ; POP into R4
  LDR R3, R6, #0
  ADD R6, R6, #1    ; POP into R3
  LDR R2, R6, #0
  ADD R6, R6, #1    ; POP into R2
  LDR R1, R6, #0
  ADD R6, R6, #1    ; POP into R1
  LDR R0, R6, #0
  ADD R6, R6, #1    ; POP into R0

  RET

.END
```