Procedure to multiply its argument by 10

```
     .ORIG     x4000
;; Activation record for mult10
;;
;;  offsets to R5 during call
;;  -1 -- SAVED R1 (local local)     -- R6 here during call
;;   0 -- SAVED R0 (first local)     -- R5 here during call
;;   1 -- dynamic link
;;   2 -- return PC
;;   3 -- return value     (10*X)    -- R6 here on exit
;;   4 -- ARG 1            (X)       -- R6 here on entry
;;

;; CREATE THE ACTIVATION RECORD
mult10  STR     R7,R6,#-2       ;; Store caller PC
        STR     R5,R6,#-3       ;; Store caller dynamic link
        ADD     R5,R6,#-4       ;; Set R5 to first local
        ADD     R6,R5,#-1       ;; Set R6 to last local

;; SAVE REGS
        STR     R0,R5,#0
        STR     R1,R5,#-1

;; DO THE WORK
        LDR     R0,R5,#4        ;; load X into R0
        ADD     R1,R0,R0
        ADD     R1,R1,R1
        ADD     R0,R0,R1
        ADD     R0,R0,R0        ;; R0 contains 10*X

;; SAVE RETURN VALUE
        STR     R0,R5,#3        ;; Save return value

;; RESTORE REGS
        LDR     R0,R5,#0
        LDR     R1,R5,#-1

;; RETURN AND DELETE ACTIVATION RECORD
        LDR     R7,R5,#2
        ADD     R6,R5,#3
        LDR     R5,R5,#1
        RET

        .END
```

Procedure to compare to 16 bits numbers

Similar to the compare function needed by C's `qsort`

```
        .ORIG   x4100
;; Activation record for cmp16
;;
;;  offsets to R5 during call
;;  -2 -- SAVED R2 (last  local)    -- R6 here during call
;;  -1 -- SAVED R1
;;   0 -- SAVED R0 (first local)    -- R5 here during call
;;   1 -- dynamic link
;;   2 -- return PC
;;   3 -- return value              -- R6 here on exit
;;   4 -- ARG 1         (A)         -- R6 here on entry
;;   5 -- ARG 2         (B)
;;
;;   If A<B, return -1
;;   If A=B, return  0
;;   If A>B, return  1

;; CREATE THE ACTIVATION RECORD
cmp16   STR     R7,R6,#-2       ;; Store caller PC
        STR     R5,R6,#-3       ;; Store caller dynamic link
        ADD     R5,R6,#-4       ;; Set R5 to first local
        ADD     R6,R5,#-2       ;; Set R6 to last local

;; SAVE REGS
        STR     R0,R5,#0
        STR     R1,R5,#-1
        STR     R2,R5,#-2

;; DO THE WORK
        AND     R2,R2,#0        ;; R2 <- 0
        LDR     R1,R5,#5        ;; R1 <- B
        BRn     Bneg
        BRz     Bzero

;; 0 < B, if here
        LDR     R0,R5,#4        ;; R0 <- B
        BRnz    retm1           ;; A <= 0 < B, return -1

;; 0 < A && 0 < B, if here
        NOT     R1,R1
        ADD     R1,R1,#1        ;; R1 <- A-B
        BRnzp   atestem
```

```
Bzero
;; B == 0, if here
        LDR     R0,R5,#4        ;; R0 <- A = A-B
        BRnzp   testem

Bneg
;; B < 0, if here
        LDR     R0,r5,#4        ;; R0 <- A
        BRzp    retp1           ;; A >= 0 > B, return 1

;; A < 0 && B < 0, if here
        NOT     R1,R1
        ADD     R0,R0,#1        ;; This makes it work even if B=x8000
atestem ADD     R0,R0,R1        ;; R0 <- A-B

;; NZP set for result A-B, if here
testem  BRn     retm1
        BRz     retnow

retp1   ADD     R2,R2,#1        ;; return +1
        BRnzp   retnow

retm1   ADD     R2,R2,#-1       ;; return -1

retnow
;; SAVE RETURN VALUE
        STR     R2,R5,#3        ;; Save return value

;; RESTORE REGS
        LDR     R0,R5,#0
        LDR     R1,R5,#-1
        LDR     R2,R5,#-2

;; RETURN AND DELETE ACTIVATION RECORD
        LDR     R7,R5,#2
        ADD     R6,R5,#3
        LDR     R5,R5,#1
        RET

     .END
```

Procedure to compare to 16 bits numbers

Similar to C's < operator

```
        .ORIG   x4200
;; Activation record for lt16
;;
;;  offsets to R5 during call
;;  -2 -- SAVED R2 (last  local)    -- R6 here during call
;;  -1 -- SAVED R1
;;   0 -- SAVED R0 (first local)    -- R5 here during call
;;   1 -- dynamic link
;;   2 -- return PC
;;   3 -- return value  (A<B)       -- R6 here on exit
;;   4 -- ARG 1         (A)         -- R6 here on entry
;;   5 -- ARG 2         (B)
;;

;; CREATE THE ACTIVATION RECORD
lt16    STR     R7,R6,#-2       ;; Store caller PC
        STR     R5,R6,#-3       ;; Store caller dynamic link
        ADD     R5,R6,#-4       ;; Set R5 to first local
        ADD     R6,R5,#-2       ;; Set R6 to last local

;; SAVE REGS
        STR     R0,R5,#0
        STR     R1,R5,#-1
        STR     R2,R5,#-2

;; DO THE WORK
        LDR     R0,R5,#4        ;; R0 <- A
        LDR     R1,R5,#5        ;; R1 <- B
        LD      R2,acmp16       ;; R2 <- address of cmp16 routine

;; Push A and B on the call stack
        STR     R1,R6,#-1
        STR     R0,R6,#-2
        ADD     R6,R6,#-2

        JSRR    R2              ;; call cmp16

;; Load cmp16 return value and load lt16 return value in R0
        AND     R0,R0,#0        ;; R0 <- 0
        LDR     R1,R6,#0        ;; R1 <- cmp16(A,B)
        BRzp    Leave
        ADD     R0,R0,#1
```

```
Leave
;; POP RETURN VALUE AND ARGS
        ADD     R6,R6,#3

;; SAVE RETURN VALUE
        STR     R0,R5,#3        ;; Save return value

;; RESTORE REGS
        LDR     R0,R5,#0
        LDR     R1,R5,#-1
        LDR     R2,R5,#-2

;; RETURN AND DELETE ACTIVATION RECORD
        LDR     R7,R5,#2
        ADD     R6,R5,#3
        LDR     R5,R5,#1
        RET

acmp16  .FILL   x4100           ;; Address of CMP16 routine
        .END
```

Euclid's **slow** way to do GCD

```
int GCD(int N, int M) {
   if (N==M) {
      return N ;
   } else if (N<M) {
      return GCD(M,N) ;
   } else {
      return GCD(N-M, M) ;
   }
}
```

or

```
int GCD(int N, int M) {
   int newN, newM  ;

   if (N==M) {
      return N ;
   } else {
      if (N>M) {
         newN = N-M ;
         newM = M ;
      } else {
         newN = M ;
         newM = N ;
      }
      return GCD(newN, newM) ;
   }
}
```

Recursive GCD

```
        .ORIG   x4B00
;; Activation record for GCD
;;
;;  offsets to R5 during call
;;  -2 -- SAVED R2 (last local)     -- R6 here during call
;;  -1 -- SAVED R1
;;   0 -- SAVED R0 (first local)    -- R5 here during call
;;   1 -- dynamic link
;;   2 -- return PC
;;   3 -- return value  (GDC(N,M))  -- R6 here on exit
;;   4 -- ARG 1         (N)         -- R6 here on entry
;;   5 -- ARG 2         (M)
;;

;; CREATE THE ACTIVATION RECORD
GCD     STR     R7,R6,#-2       ;; Store caller PC
        STR     R5,R6,#-3       ;; Store caller dynamic link
        ADD     R5,R6,#-4       ;; Set R5 to first local
        ADD     R6,R5,#-2       ;; Set R6 to last local

;; SAVE REGS
        STR     R0,R5,#0
        STR     R1,R5,#-1
        STR     R2,R5,#-2

;; DO THE WORK
        LDR     R0,R5,#4        ;; R0 <- N
        LDR     R1,R5,#5        ;; R1 <- M
        NOT     R2,R1
        ADD     R2,R2,#1
        ADD     R2,R0,R2        ;; R2 <- N-M
        BRz     Leave           ;; if N=M, return N
        Brp     Recurse         ;; if N>M, recurse with N-M,M

;; N < M, if here
        ADD     R2,R1,#0        ;; R2 <- M
        ADD     R1,R0,#0        ;; R1 <- N
```

```
Recurse
;; Ready for a recursive call
;;   First argument is in R2
;;   Second argument is in R1
        STR     R2,R6,#-2       ;; Push N and M onto call stack
        STR     R1,R6,#-1
        ADD     R6,R6,#-2
        JSR     GCD             ;; Ring me again

;; Get the return value
        LDR     R0,R6,#0        ;; R0 <- return value
        ADD     R6,R6,#3        ;; pop return value and args

Leave
;; SAVE RETURN VALUE
        STR     R0,R5,#3        ;; Save return value

;; RESTORE REGS
        LDR     R0,R5,#0
        LDR     R1,R5,#-1
        LDR     R1,R5,#-2

;; RETURN AND DELETE ACTIVATION RECORD
        LDR     R7,R5,#2
        ADD     R6,R5,#3
        LDR     R5,R5,#1
        RET

        .END
```