-write a main program and subroutine equivalent to:

```
int FINDSum (int N){
    int sum=0;
    while (N!=0){
        sum += N;
        --N;
    }
    return (sum);
}
```

ARM:

```
.text
.global _start

_start: mov   R4, #N   // R4 points to N
        LDR   R0, [R4] // R0=N
        BL    FINDSUM
        STR   R0, [R4, #4]
END:    B     END

N:   .word 10
Sum: .word 0
// N is in R0; sum is returned in R0
FINDSUM: mov R2, #0
WHILE:   CMP R0, #0
         BEQ  ENDLOOP
         ADD  R2, R0
         SUB  R0, #1
         B    WHILE.
ENDLOOP: mov  R0, R2
         mov  PC, LR
```

ref:

Enhanced Processor:

```
MAIN:   mvi   r4, #N   // r4 ← addr of N
        ld    r0, [r4] // r0 = N
        mv    r5, r7   // first set return addr.
        mvi   r7, FINDSUM  // call subroutine.
        mvi   r1, #1
        add   r4, r1   // every address is a word (not byte addressable)
                       //   so add "1" not "4"   r4 has addr. of sum.
        st.   r0, [r4]
END:    mvi   r7, #END
N:      .word 10
Sum:    .word 0
// N is in R0, sum returned in R0
FINDSum: mvi  r2, #0
WHILE:  mvi   r1, #L1
        mv    r0, r0
        mvnz  r7, r1
        mvi   r7, #ENDLOOP
L1:     add   r2, r0
        mvi   r1, #1
        sub   r0, r1
        mvi   r7, #WHILE.
ENDLOOP: mv   r0, r2
        mvi   r1, #1
        add   r5, r1
        add   r5, r1
```

```
.text
.global _start
_start: mov   R4, #N   // R4 points to N
        LDR   R0, [R4] // R0 = N
        BL    FINDSUM
        STR   R0, [R4, #4]
END:    B     END

N:    .word 10
Sum:  .word 0
```

ref.

```
FINDSUM: mov  R2, #0
WHILE:  CMP   R0, #0
        BEQ   ENDLOOP
        ADD   R2, R0
        SUB   R0, #1
        B     WHILE.
ENDLOOP: mov  R0, R2
        mov   PC, LR
```

mv r7, r5 //return.

ARM vs. Enhanced processor (difference)

ARM

32 bits

Byte-addressable.

Register R0-R15

4-flags (N, C, V, Z)

load/store

mov

LDR, LDRB, STR, STRB, ...

-unconditional branching

B Label

- branching using Z flag: BEQ, BNE

```
LABEL: SUBS R0,R1
       BNE  L1
       ...
```

```
L1:    SUBS R0,R1
       BEQ  L2
       ...
L2:    ...
```

(Z: BEQ → L2; $\bar{Z}$: BNE → ...)

Enhanced. Processor.

16-bits.

word-addressable.

registers r0-r7

2-flags (Z, C)

load/ store.

mv, mvi (use mvi to move immediate #)

ld, st.

-unconditional branching

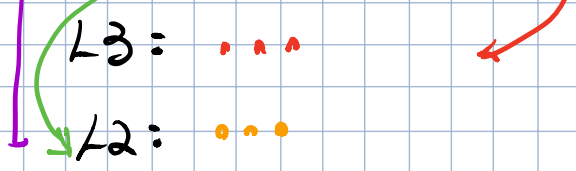
mvi r7 #Label

- branching using Z flag: mvnz

```
LABEL: mvi  r4, #L1
       sub  r0, r1
       mvnz r7.
       ...
```

```
L1:    mvi  r4, #L3
       sub  r1, r0
       mvnz r7, r4
       mvi  r7. #L2
       ...
L2:    ...
```

($\bar{Z}$: mvnz → L3; Z: mvi → L2)


L3:
L2: