CMSC B240 Computer Organization - Spring 2025
Lab Activity #8 – Subroutines

Here is an algorithm for multiplication (using repeated addition):

```
Given A, B
result ← 0

while B > 0 do
      result ← result + A
      B ← B – 1
```

We will use some additional registers (R0 and R2) in the LC-3 algorithm:

```
R0 ← R3 (save value of A)
R3 ← 0 (initialize result)
R2 ← R1 (save value of B) while R2 != 0 do
R3 ← R3 + R0
R2 ← R2 -1
```

Since the algorithm uses R0, and R2, we will need to save and restore those registers in the subroutine.

Here is the LC-3 code for MULT:

```
            .ORIG x3020
MULT        ST R0, SaveR0 ; Save R0 and R2
            ST R2, SaveR2
            ADD R0, R3, #0 ; R0 ← R3
            AND R3, R3, #0 ; R3 ← 0 (result)
            ADD R2, R1, #0 ; R2 ← R1
; while R2 != 0
LOOP        BRz MULDone
; do
            ADD R3, R3, R0 ; R3 ← R3 + R1
            ADD R2, R2, #-1 ; R1 ← R1 - 1
            BR LOOP
MULDone     LD R0, SaveR0 ; Restore R0 and R2
            LD R2, SaveR2
            RET
SaveR0 .BLKW 1
SaveR2 .BLKW 1
            .END
```

Implement a complete program in the LC-3 simulator and test to make sure the subroutine works correctly. Try different values of A (>0) and B (>=0).

CMSC B240 Computer Organization - Spring 2025
Lab Activity #8 – Subroutines

Nested Subroutines

Recall from class that when a subroutine calls another, we must save (and restore) the return address(es) that are in R7. To try this, let us write a subroutine FACT, that computes the factorial of a given number (>=0). Here is an algorithm that uses an iterative solution:

```
Fact(n):
      result ← 1
      i ← n
      while i != 0 do
            result ← result * i
            i ← i - 1
```

Assume that n will be supplied in the register R0, and the subroutine returns the result also in R0. That is,

```
R0 ← FACT(R0)
```

Here is the LC-3 algorithm. We are using R3 to store the result:

```
R3 ← 1
R1 ← R0
while R1 !- 0 do
      R3 ← R3 * R1
      R1 ← R1 - 1
      R0 ← R3 (put result back in R0)
```

Since we are using R1 and R3, we will need to save R1 and R3 in FACT. We will make use of MULT as we wrote above to do the multiplication. That also means we will need to save and restore R7 before calling MULT in FACT. Here is the complete algorithm:

```
FACT(R0)
      Save R1 and R2
      R3 ← 1
      R1 ← R0
      while R1 != 0 do
            R3 ← R3 * R1 (use MULT)
            R1 ← R1 - 1
      R0 ← R3 (put result back in R0)
      Restore R1 and R2
return
```

Exercise 2: Implement the `FACT` subroutine as described above and use it to compute the factorial of 4. Try a couple of other values.

Exercise 2 is your **Assignment 5** and it will be due on **Tuesday, April 16**. See the class web page for a complete description of what to hand in.

LC-3 Assembly Cheat Sheet

Instruction	Action	Addressing Mode
ADD R2, R2, R3	R2 = R2 + R3	Register
ADD R2, R2, #1	R2 = R2 + 1	Register, Immediate
AND R2, R2, R3	R2 = R2 AND R3	Register
AND R2, R2, #1	R2 = R2 AND 1	Register, Immediate
$BR_{[n][z][p]}$ LABEL	If [n][z][p] Go to LABEL	CC, PC-Relative
HALT	Stop program execution	
IN	R0 = Input char from keyboard	None
JMP R1	PC = R1	Register
JSR		
JSRR		
LD R2, LABEL	R2 = m[LABEL]	Register, PC-Relative
LDI R2, LABEL	R2 = m[m[LABEL]]	Register, Indirect
LDR R2, R0, $\#n_6$	M[R0 + n]	Base Register
LEA R2, LABEL	R2 = LABEL	Register, PC-Relative
NOT R2, R1	R2 = NOT(R1)	Register
OUT	Output R0 to Console	None
PUTS	Output String at M[R0] to console	None
RET		
RTI		
ST R2, LABEL	M[LABEL] = R2	Register, PC-Relative
STI R2, LABEL	M[M[LABEL]] = R2	Register, Indirect
STR R2, R0, $\#n_6$	M[R0 + n] = R2	Register, Base Register
TRAP $trapvect_8$	Execute Service # $trapvect_8$	Immediate