Generals:
16-bit addresses
64k memory locations (16-bit word @ each loc.)
2'sC integers, 8 data reisters (3bits to name reg)
3 condition code bits, 4bit opcodes(16 intsructs)
Words and addresses are totally unrelated

Instruc. Cycle for all LC3 instructs: (not orthog)
Fetch instruction → Decode Instruction →
Evaluate instruction → Fetch op from Mem →
Execute operation → Store result.

5 ways to specify operand (addressing modes):
Immediate: contained in instruction
Register: number -> 000, 001, …, 110, 111
3ways to specify mem locs: Base-offset, PC-offset, and Indirect.

3kinds of instructions:
Data movement: Load (value into register)
Store (value from register)
Calculation: ADD
Control: Branch, Jump – modify the PC during execute instruction phase of cycle (otherwise PC is incremented during Fetch instuc., after reading instruct. From memory) – (IR – decodes instruction)

Differences from SDC:
Address size, Word size, radix, #of Register, CondCode, #of opcodes, SDC uses Absolute addr. (while addr is part of instruct.)
LC3 has 16bit addr and 16bit instruc

```
; This is an LC3 program that left-shifts
;     a value from one word to another.
; Variables and their meanings:
; R0 - R
; R1 - L
; R2 - I

      .ORIG      x3000             ; Start program at line x3000
      LD    R0, X                  ; R0 is R
      LD    R1, L                  ; R1 is L (0)
      LD    R2, N                  ;Store I into R2 to keep track
                                   ;     of shifts left to do
      LD    R5, P                  ;Store 1 into R5
      NOT   R5, R5                 ;Store the NOT of 1 into R5

  Loop      BRZ   Done             ;While I>0
      ADD   R1, R1, R1             ;Left-shift L by multiplying
      ADD   R0, R0, 0              ;inspect the left-most bit of R
      BRZP  NOTneg                 ;If leftt-most bit of R is 1
      NOT   R3, R1                 ;Take the not of L to begin OR
      AND   R4, R3, R5       ;AND L with the not of 1 and store it
      NOT   R1, R4           ;Finish OR by using Demorgan setting
                             ;     right most bit of L to 1

NOTneg      ADD   R0, R0, R0       ;Left-shift R one bit
      ADD   R2, R2, -1             ;decrement the count
      BR    Loop                   ;end while loop


  Done      ST    R1, L            ;store L into memory
      ST    R0, R                  ;store R into memory
      HALT                         ;end program

  X     .Fill       xFFFF     ;This is the number to be shifted
  N     .Fill       5         ;number of shifts to be done
  P     .Fill       x0001     ;setting P = 1 for compare purposes
  R     .Blkw       1         ;initializing R to 0
  L     .Blkw       1         ;initilizing L to 0

      .END                     ;Tell assembler this ends file.
```

Instruction goes with →	*Action*
13. ADD R_1 R_2 1 00000	f. $R_1 \leftarrow R_2$
14. AND R_1 R_2 1 00000	d. $R_1 \leftarrow 0$
15. ADD R_1 R_2 1 00001	j. $R_1 \leftarrow R_2 + 1$
16. AND R_1 R_2 1 00001	a. $R_1 \leftarrow R_2[0]$
17. ADD R_1 R_2 1 11111	i. $R_1 \leftarrow R_2 - 1$
18. AND R_1 R_2 1 11111	f. $R_1 \leftarrow R_2$
19. ADD R_1 R_2 000 R_2	e. $R_1 \leftarrow 2 * R_2$
20. AND R_1 R_2 000 R_2	f. $R_1 \leftarrow R_2$
21. ADD R_1 R_2 000 000	g. $R_1 \leftarrow R_2 + R0$
22. AND R_1 R_2 000 111	h. $R_1 \leftarrow R_2$ AND R7
23. NOT R_1 R_2 11111	c. $R_1 \leftarrow -R_2 - 1$

2. (Implement `if R0 < 0 then R0 ← 0 else M[x30AC] ← R0`)

Addr	*Value*	*Asm*	*Action/Comment*
x4000	0001 000 000 1 00000	ADD R0,R0,0	Test value of R0
x4001	0000 011 000000010	BRZP 2	If R0 ≥ 0, go to false arm
x4002	0101 000 000 1 00000	AND R0,R0,0	. R0 ← 0
x4003	0000 111 000000001	BR 1	Skip over false arm
x4004	0011 000 0 1010 0111	ST R0,xA7	. M[x40AC] ← R0
x4005	...		(code after if-else)

3. Implement "`if R7 = 1 then go to x5000`". (R1 is a temporary register.)

Addr	*Value*	*Asm*	*Action/Comment*
x8000	0001 001 111 1 11111	ADD R1,R7,-1	R1 ← R7 – 1
x8001	0000 101 000000011	BRNP 3	If R7 ≠ 1, go to end if
x8002	0010 001 000000001	LD R1,1	. R1 ← Target location
x8003	1100 000 001 00000	JMP R1	. Jump to target
x8004	x5000		Location of target
x8005	...		(code after if-then)

```
; Register usage: R1 = k, R2 = X, R3 = product
;
        .ORIG    x3050
         LD      R2, X          ; R2 = X
         AND     R3, R3, 0      ; R3 = X * (Y-k)
         LD      R1, Y          ; k = Y
Loop     BRZ     Done           ; until k = 0
         ADD     R3, R3, R2     ;    R3 = R3 + X
         ADD     R1, R1, -1     ;    k--
         BR      Loop
Done     ST      R3, product    ; product = X*Y
         HALT

X        .FILL   16
Y        .FILL   6
product  .BLKW   1              ; Holds X*Y at end
```

```
; The main program exercises the readline subroutine.
;
          .ORIG     x3000
          LEA       R0, string1   ; read first message
          JSR       readstring

          LEA       R0, string2   ; read second message
          JSR       readstring

          HALT

string1   .BLKW     100
string2   .BLKW     100

   ; Restore registers and return
                LD      R7, RSsave7   ; Restore R7
                LD      R3, RSsave3   ; Restore R3
                LD      R2, RSsave2   ; Restore R2
                LD      R1, RSsave1   ; Restore R1
                LD      R0, RSsave0   ; Restore R0
                JMP     R7
```

```
        .ORIG    x3000
         LEA     R1, buffer     ; buffer_posn = &buffer
         LEA     R0, msg
         PUTS                   ; prompt for input
         GETC                   ; get char into R0
         LD      R2, retChar    ; R2 = return char
         NOT     R2, R2         ; R2 = -(return char) - 1
         ADD     R2, R2, 1      ; R2 = -(return char)
         ADD     R3, R0, R2     ; calculate R0 - return char
Loop     BRZ     Done           ; until r0 = return char
         OUT                    ; print char in R0
         STR     R0, R1, 0      ; *buffer_posn = char read in
         ADD     R1, R1, 1      ; buffer_posn++
         GETC                   ; get char into R0
         ADD     R3, R0, R2     ; calc char - return char
         BR      Loop           ; continue loop
Done     OUT                    ; print return char in R0
         AND     R3, R3, 0      ; R3 = null char ('\0')
         STR     R3, R1, 0      ; terminate string in buffer
         LEA     R0, buffer
         PUTS                   ; print the string we read in
         HALT

retChar  .FILL   x0A            ; Return character (\n)
msg      .STRINGZ "Enter chars (return to halt): "
buffer   .BLKW   100            ; buffer space for string
         .END
```

Addr	Value	Asm	Action/Comment
x3010	0010 010 000011111	LD R2,x1F	R2 ← N
x3011	0000 110 000001111	BRNZ 15	Top: if R2 ≤ 0, exit loop
x3012	...		(Loop body)
...	...		
x301F	0001 010 010 1 11111	ADD R2,R2,-1	R2--
x3020	0000 111 111110000	BR -16	(bottom of loop) go to Top
x3030	...		(Code after loop)
			Value of N

Addr	Instruction	Asm	Action
x30F6	1110 001 111111101	LEA R1,-3	R1 ← PC-3 = x30F7-3 = x30F4
x30F7	0001 010 001 1 01110	ADD R2,R1,14	R2 ← R1+14 = x30F4+14 = x3102
x30F8	0011 010 111111011	ST R2,-5	M[PC-5] ← R2 M[x30F4] ← x3102
x30F9	0101 010 010 1 00000	AND R2,R2,0	R2 ← R2 AND 0 (= 0)
x30FA	0001 010 010 1 00101	ADD R2,R2,5	R2 ← R2+5 = 0+5 = 5
x30FB	0111 010 001 001110	STR R2,R1,14	M[R1+14] ← R2 M[x3102] ← 5
x30FC	1010 011 111110111	LDI R3,-9	R3 ← M[M[PC-9]] = M[M[x30FD-9]] = M[M[x30F4]] = M[x3102] = 5

Addr	Value	Asm	Action/Comment
x3000	1110 000 000000100	LEA R0,4	Pt R0 to prompt string
x3001	1111 0000 0010 0010	TRAP x22	PUTS (print prompt)
x3002	1111 0000 0010 0000	TRAP x20	GETC (read char into R0)
x3003	1111 0000 0010 0001	TRAP x21	OUT (print char in R0)
x3004	1111 0000 0010 0101	TRAP x25	HALT
x3005	0000 0000 0011 1110		Prompt: '>' = x3E
x3006	0000 0000 0010 0000		' '
x3007	0000 0000 0000 0000		end of prompt

LC-3 Reference -- CS 350: Computer Org & Assembler Lang Pgmg (v. 2015-03-26)
(You can bring this sheet to Quiz 2 & the Final Exam)

OPCODES SORTED BY MNEMONIC

Op	Hex	Bin	Arguments
ADD	1	0001	Dst Src1 0 00 Src2
ADD	1	0001	Dst Src1 1 Immed5
AND	5	0101	Dst Src1 0 00 Src2
AND	5	0101	Dst Src1 1 Immed5
BR	0	0000	NZP PCoffset9
err	D	1101	(unused opcode)
JMP	C	1100	000 Base 000000
JSR	4	0100	1 PCoffset11
JSRR	4	0100	000 Base 000000
LD	2	0010	Dst PCoffset9
LDI	A	1010	Dst PCoffset9
LDR	6	0110	Dst Base Offset6
LEA	E	1110	Dst PCoffset9
NOP	0	1110	000 0..0 (BR w/000 mask)
NOT	9	1001	Dst Src1 111111
RET	C	1100	000 111 000000 (JMP R7)
RTI	8	1000	0000 0000 0000
ST	3	0011	Src PCoffset9
STI	B	1011	Src PCoffset9
STR	7	0111	Src Base Offset6
TRAP	F	1111	0000 TrapVec8

OPCODES SORTED BY OPCODE NBR

Hex	Bin	Op	Arguments
0	0000	BR	NZP PCoffset9
0	0000	NOP	000 0..0 (BR w/000 mask)
1	0001	ADD	Dst Src1 0 00 Src2
1	0001	ADD	Dst Src1 1 Immed5
2	0010	LD	Dst PCoffset9
3	0011	ST	Src PCoffset9
4	0100	JSR	1 PCoffset11
4	0100	JSRR	000 Base 000000
5	0101	AND	Dst Src1 0 00 Src2
5	0101	AND	Dst Src1 1 Immed5
6	0110	LDR	Dst Base Offset6
7	0111	STR	Src Base Offset6
8	1000	RTI	0000 0000 0000
9	1001	NOT	Dst Src1 111111
A	1010	LDI	Dst PCoffset9
B	1011	STI	Src PCoffset9
C	1100	JMP	000 Base 00000
C	1100	RET	000 111 000000 (JMP R7)
D	1101	err	(unused opcode)
E	1110	LEA	Dst PCoffset9
F	1111	TRAP	0000 TrapVec8

Trap Vectors (Note: TRAP, JSR, JSRR modify R7)

```
x20 - GETC   Read character from keyboard into R0[7..0]; clear R0[15..8].
x21 - OUT    Print character in R0[7..0].
x22 - PUTS   Print string of ASCII chars starting at location pointed to
             by R0 (one char per location; stop at word = x0000).
x23 - IN     Like x20 but prints a prompt on the screen first.
x24 - PUTSP  Like x22 but each location contains two characters; the one
             at 7..0 is printed first then the one at 15..8.  Stop at x0000.
x25 - HALT   Halt execution.
```

Assembler Directives (below, n can be in decimal or hex)

```
.ORIG  n         Load program starting at address n (typically hex constant)
.FILL  n         Allocate 1 word of memory initialized to n
.FILL  label     Allocate 1 word of memory initialized to address of label
.BLKW  n         Allocate n words of memory initialized to 0. (Like n .FILL 0's)
.STRINGZ "str"   Allocate M+1 words for M characters and terminal null
.END             Last line of assembler program
```

ASCII: Space = 32 = x20; Newline = 10 = xA; '0'= 48 = x30; 'A'= 65 = x41; 'a'= 97 = x61
Multiples of 16: 32 48 64 80 96 112 128 144 160 176 192 208 224 240 256

PCoffset9 range: -256 ≤ PC ≤ 255
ADD & AND Immed5 range -16 ≤ val ≤ 15
CC: N100 or Z010 or P001
LD: R# <- M[PC+offset]
ST: M[PC+offset] <- R#
LEA: R# <- PC+offset
LDR: R# <- M[RB#+offset]
STR: M[RB#+offset] <- R#
LDI: R# <- M[M[PC+offset]]
STI: M[M[PC+offset]] <- R#
JMP: PC <- RB#
BR: if (CC&Mask≠000) then PC <- PC+offset
TRAP: R7 <- PC <- M[TrapVec8]
JSR: R7 <- PC; PC <- PC+Sext(PCoffset11)
JSRR: target(goto)<- RB#; R7 <- PC; PC <- target
RET = JMP R7

Unsupported Operations:
Subtraction: for X-Y → use ADD like X+NOT Y+1
OR: for X OR Y → NOT(NOT X AND NOT Y)
Setting R to 0: AND with 0 → AND R1, R1, 0
Copy R to R: 3 ways: ADD R0, R1, 0;
AND R0, R1, R1; AND R0, R1, xFFFF

Directives begin with period
Only Labels are case sensitive

If P calls Q, the activation record for the call will contain

(a) The return address to the code for P.
(c) A link to the activation record for P.
(f) Space for the local variables of Q.
(g) Space for the arguments being passed to Q.
(i) Space for the value that Q will return to P.

To implement `k = q+r;` (using `R0` and `R1` as temporary registers)

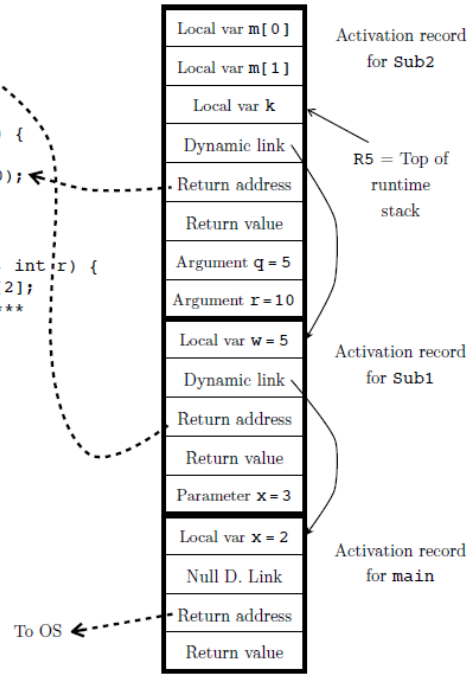
```
LDR   R0,R5,4    ; R0 = q
LDR   R1,R5,5    ; R1 = r
ADD   R0,R1,R0   ; R0 = q+r
STR   R0,R5,0    ; k = q+r
```

- At the point marked "Here" in Sub2, the activation stack looks like this:

```
main() {
  int x = 2;
  Sub1(x+1);
}

int Sub1(int x) {
  int w = x+2;
  w = Sub2(w,10);
  return w;
}

int Sub2(int q, int r) {
  int k; int m[2];
  // *** here ***
  return k;
}
```




```
main() {
  int x = 2;
  Sub1(x+1) /* Location 1 */;
}

int Sub1(int x) {
  int w = x+2;
  call Sub2(w,10)
  /* Location 2 */
  set w to result of Sub2
  set RV = w
  return;
}

int Sub2(int q, int r) {
  int k; int m[2];
  /* Location 3 */
  set RV = k;
  return;
}
```

```
Frame for Sub2:
    m[0] = 0
    m[1] = 0
    Location 4: k = 0
    DL = Location 5
    RA = Location 2
    RV = 0
    q = 5
    r = 10
For Sub1:
    Location 5: w = 5
    DL = Location 6
    RA = Location 1
    RV = 0
    x = 3
For main:
    Location 6: x = 2
    DL = null
    RA to OS
    RV = 0
```

```
======== Problem 1 ========
At location 1:
For f:
        Location 9: r = 0
        DL = Location 8
        RA = Location 2
        RV = 1
        n = 1
For f:
        Location 8: r = 0
        DL = Location 7
        RA = Location 2
        RV = 0
        n = 2
For f:
        Location 7: r = 0
        DL = Location 6
        RA = Location 4
        RV = 0
        n = 3
For main:
        Location 6: m = 0;
        DL = null
        RA to OS
        RV = 0
--------------------
At location 3:
For f:
        Location 8: r = 1
        DL = Location 7
        RA = Location 2
        RV = 2
        n = 2
For f:
        Location 7: r = 0
        DL = Location 6
        RA = Location 4
        RV = 0
        n = 3
For main:
        Location 6: m = 0;
        DL = null
        RA to OS
        RV = 0
--------------------
At location 3:
For f:
        Location 7: r = 2
        DL = Location 6
        RA = Location 4
        RV = 6
        n = 3
For main:
        Location 6: m = 0;
        DL = null
        RA to OS
        RV = 0
--------------------
At location 5:
For main:
        Location 6: m = 6;
        DL = null
        RA to OS
        RV = 0
```

```
int f(int n) {
    int r = 1;
    if (n <= 1) {
        RV = 1 /* Location 1 */;
        return;
    }
    else {
        call f(n-1) /* Location 2 */;
        set r to result of f
        RV = r*n; /* Location 3 */;
        return;
    }
}
int main() {
    int m = 0;
    call f(3) /* Location 4 */
    set m to result of f; /* Location 5 */
    return 0;
}
```

```
int g(int n, int r) {
    if (n <= 1)
        RV = r;
        /* Location 1: */ return;
    else {
        call g(n-1, r*n) /* Location 2 */
        set RV = result of g
        /* Location 3 */
        return;
    }
}

int main() {
    int m = 0;
    call g(3, 1) /* Location 4 */;
    set m = result of g
    /* Location 5 */
    return;
}
```

```
;////~~~  Location 1  ~~~////
For g:
    DL = Location 20
    RA = Location 4
    RV = 6
    n = 1
    r = 6
For main:
    Location 20: m = 0
    DL = null
    RA to OS
    RV = 0
;////~~~  Location 3  ~~~////
; -- not accessed because of
tail recursion --
; ignores Location 3 instruction

;////~~~  Location 5  ~~~////

For main:
    Location 20: m = 6
    DL = null
    RA to OS
    RV = 0
```

```
======== Problem 2 ========
[Note: this is the non-tail-
recursive solution]
At Location 1:
For g:
        Location 9: DL = Location 8
        RA = Location 2
        RV = 6
        n = 1
        r = 6
For g:
        Location 8: DL = Location 7
        RA = Location 2
        RV = 0
        n = 2
        r = 3
For g:
        Location 7: DL = Location 6
        RA = Location 4
        RV = 0
        n = 3
        r = 1
For main:
        Location 6: m = 0;
        DL = null
        RA to OS
        RV = 0
--------------------
At Location 3:
For g:
        Location 8: DL = Location 7
        RA = Location 2
        RV = 6
        n = 2
        r = 3
For g:
        Location 7: DL = Location 6
        RA = Location 4
        RV = 0
        n = 3
        r = 1
For main:
        Location 6: m = 0;
        DL = null
        RA to OS
        RV = 0
--------------------
At Location 3:
For g:
        Location 7: DL = Location 6
        RA = Location 4
        RV = 6
        n = 3
        r = 1
For main:
        Location 6: m = 0;
        DL = null
        RA to OS
        RV = 0
--------------------
At Location 5:
For main:
        Location 6: m = 6;
        DL = null
        RA to OS
        RV = 0
```

Bits	U/S	S/M	1C	2C
1111	15	-7	**-0**	-1
1110	14	-6	-1	-2
1101	13	-5	-2	-3
1100	12	-4	-3	-4
1011	11	-3	-4	-5
1010	10	-2	-5	-6
1001	9	-1	-6	-7
1000	8	**-0**	-7	-8

Conversions To:

S/M: Leftmost used for sign. Has (±0).

1'sC: **Flip** 7(0111) to get -7(1000). Has (±0).

2'sC: **Flip+1** 7(0111) to (1000+1)= -7(1001). Has more neg. than pos.

Conversions From:

S/M: Replace leftmost with a sign.

1'sC: **Flip** -7(1000) to get 7(0111).

2'sC: **Flip except last 1 or 0**

-3(1101) to 3(0011).