Mapping to LC3 instructions

- Conditional branching

Logical expression	Numeric expression	Branch instruction	negated branch instruction
if (a < b)	if ((a - b) < 0)	BRn	BRzp
if (a <= b)	if ((a - b) <= 0)	BRnz	BRp
if (a == b)	if ((a - b) == 0)	BRz	BRnp
if (a >= b)	if ((a - b) >= 0)	BRzp	BRn
if (a > b)	if ((a - b) > 0)	BRp	BRnz
if (a != b)	if ((a - b) != 0)	BRnp	BRz

- Simple IF

High level code	LC3 Code

if (a < b) {
  // do something
}
	LD R0, a ; load a
	LD R1, b ; load b
	NOT R1, R1 ; begin 2'scomplement of b
	ADD R1, R1, #1 ; R1 now has -b
	ADD R0, R0, R1 ; R0 = a + (-b)
	  ; condition code now set
	BRzp END_IF ; if false, skip over code
	
	; code to do something (the then clause)
	
	END_IF: ; remainder of code after if

- Simple IF-Else Statement

High level code	LC3 Code

if (a < b) {
  // do something
}
else {
  // do something else
}
	LD R0, a ; load a
	LD R1, b ; load b
	NOT R1, R1 ; begin 2's complement of b
	ADD R1, R1, #1 ; R1 now has -b
	ADD R0, R0, R1 ; R0 = a + (-b)
	  ; condition code now set
	BRzp ELSE ; if false, skip over code
	
	ELSE: ; code to do something (the then clause)

	BR END_ELSE ; don't execute else code ELSE: ; code for else clause here END_ELSE: ; remainder of code after else

- Loops

High level code	LC3 Code	
i = 0;	01: AND R0, R0, #0 ; AND with 0 yields zero 02: ST R0, i ; store 0 in i	01: AND R0, R0, #0 ; AND with 0 yields zero 02: TOP: ST R0, i ; store i (0 1st time)
i < limit;	03: TEST: LD R0, i ; get i 04: LD R1, limit ; get limit 05: NOT R1, R1 ; two's complement negation 06: ADD R1, R1, #1 07: ADD R0, R0, R1 ; compute i - limit 08: BRzp END ; done when (i - limit) >= 0	; R0 contains i 03: LD R1, limit ; get limit 04: NOT R1, R1 ; two's complement 05: ADD R1, R1, #1 06: ADD R0, R0, R1 ; compute i - limit 07: BRzp END ; done when (i - limit) >= 0
i++	09: LD R0, i ; get i 10: ADD R0, R0, #1 ; increment it 11: ST RO, i ; save the new value	08: LD R0, i ; get i 09: ADD R0, R0, #1 ; increment it
	12: BR TEST ; go back and test again 13: END: ; code after loop completes	10: BR TOP ; store and test again 11: END: ; code after loop completes

- Stack Example

Want to compute E = (A + B).(C + D). Let say A = 25, B = 17, C = 3 and D = 2

Stack Implementation	Register Implementation

Push A ; push the value of variable A from memory to stack
Push B
Add ; 2 POP's occur so that operands can be given to ALU, and the result is pushed back to the stack
Push C
Push D
Add
Mul
POP E ; pop it off stack and store it in the address of variable E represents
	LD R0,A
	LD R1,B
	ADD R0,R0,R1
	LD R2,C
	LD R3,D
	ADD R2,R2,R3
	MUL R0,R0,R2
	ST R0,E