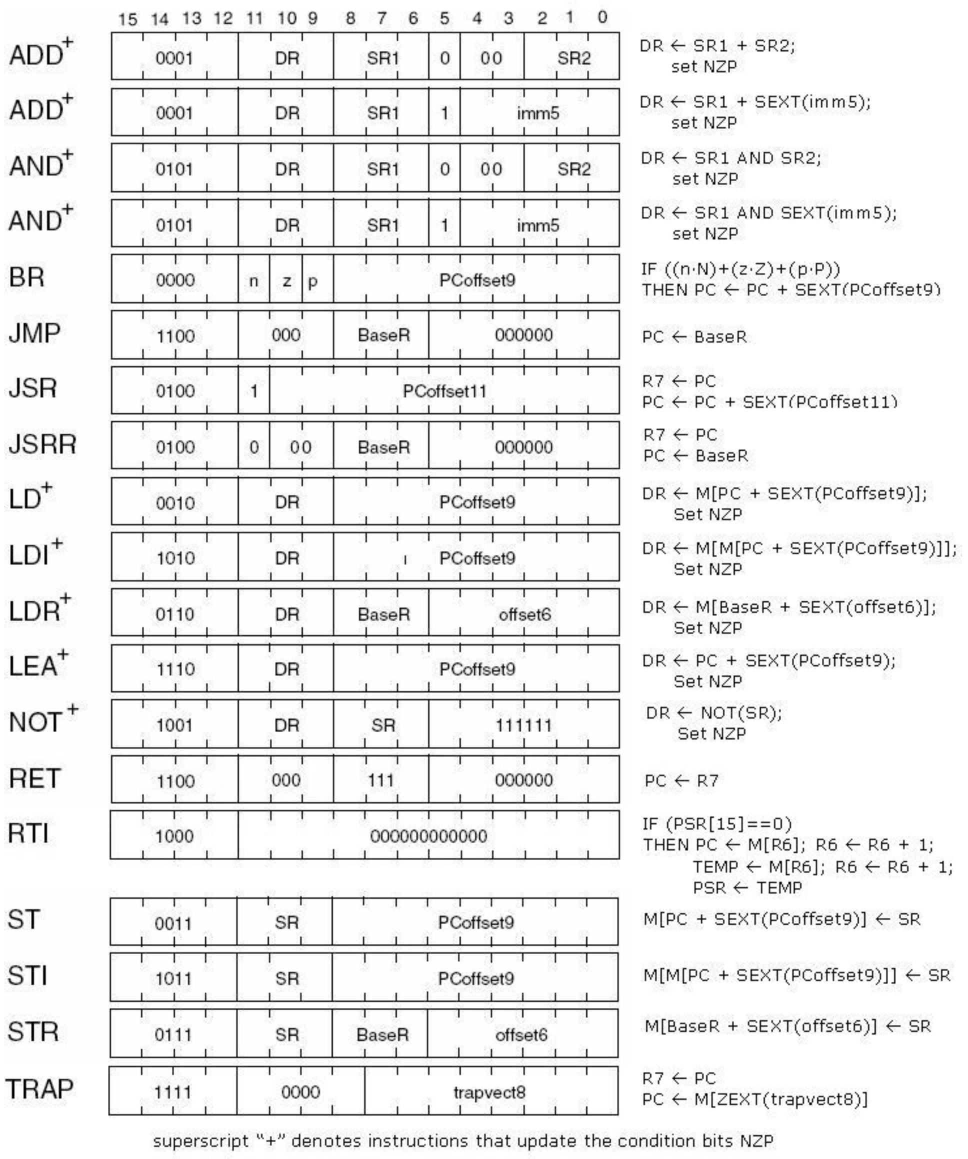
HCMUT ECE 120 Discussion Section 10: LC-3 Instruction Execution and Programming

In this discussion you will be given a sequence of binary words and you will be asked to convert each binary word to a corresponding LC-3 instruction. You will then trace the execution of the program and explain the function performed by it.

You should read Chapter 5 from the Textbook before attending this discussion.

LC-3 Instruction Set



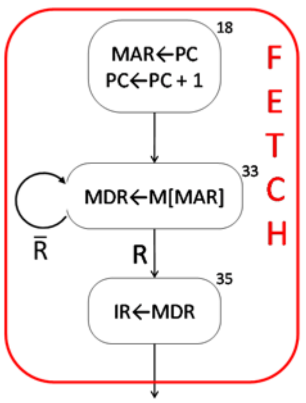
	15 14 13 12	11 10 9	8 7 6	5	4 3	2 1 0	
ADD$^+$	0001	DR	SR1	0	00	SR2	DR ← SR1 + SR2; set NZP
ADD$^+$	0001	DR	SR1	1	imm5		DR ← SR1 + SEXT(imm5); set NZP
AND$^+$	0101	DR	SR1	0	00	SR2	DR ← SR1 AND SR2; set NZP
AND$^+$	0101	DR	SR1	1	imm5		DR ← SR1 AND SEXT(imm5); set NZP
BR	0000	n z p	PCoffset9				IF ((n·N)+(z·Z)+(p·P)) THEN PC ← PC + SEXT(PCoffset9)
JMP	1100	000	BaseR	000000			PC ← BaseR
JSR	0100	1	PCoffset11				R7 ← PC PC ← PC + SEXT(PCoffset11)
JSRR	0100	0 00	BaseR	000000			R7 ← PC PC ← BaseR
LD$^+$	0010	DR	PCoffset9				DR ← M[PC + SEXT(PCoffset9)]; Set NZP
LDI$^+$	1010	DR	PCoffset9				DR ← M[M[PC + SEXT(PCoffset9)]]; Set NZP
LDR$^+$	0110	DR	BaseR	offset6			DR ← M[BaseR + SEXT(offset6)]; Set NZP
LEA$^+$	1110	DR	PCoffset9				DR ← PC + SEXT(PCoffset9); Set NZP
NOT$^+$	1001	DR	SR	111111			DR ← NOT(SR); Set NZP
RET	1100	000	111	000000			PC ← R7

RTI	1000	000000000000					IF (PSR[15]==0)
							THEN PC ← M[R6]; R6 ← R6 + 1;
							TEMP ← M[R6]; R6 ← R6 + 1;
							PSR ← TEMP

ST	0011	SR	PCoffset9				M[PC + SEXT(PCoffset9)] ← SR
STI	1011	SR	PCoffset9				M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111	SR	BaseR	offset6			M[BaseR + SEXT(offset6)] ← SR
TRAP	1111	0000	trapvect8				R7 ← PC PC ← M[ZEXT(trapvect8)]

superscript "+" denotes instructions that update the condition bits NZP

The FETCH instruction phase



1. LC-3 Instructions

Shown on the right is a snapshot of a portion of the contents of the LC-3 memory for addresses x3000-x3005 and x4000-x4003. Both addresses and data are shown in hexadecimal. The x3000-x3005 addresses contain a short program. The x4000-x4003 addresses contain data that will be processed by the program. In this worksheet, you will interpret the contents of memory, trace the program, and determine its functionality.

address	data
x3000	x14A1
x3001	x6680
x3002	x923F
x3003	x1261
x3004	x1243
x3005	x0BFA
x3FFF	36
x4000	x8438
x4001	xA164
x4002	x2319
x4003	xC8FB

1. Rewrite the contents of memory at addresses x3000-x3005 in binary, then interpret them as LC-3 instructions. For each instruction, write the name of the opcode as well as the effect of the instruction using RTL notation. Address x3000 has been done for you as an example.

Address	Binary Contents	Opcode	RTL (Be specific to this instruction)
x3000	0001 0100 1010 0001 (x14A1)	ADD	R2 ← R2 + 1, set CC
x3001			
x3002			
x3003			
x3004			
x3005			

2. Trace the entire instruction cycle, from the instruction FETCH phase through the instruction EXECUTE phase, for the instruction stored in memory at address x3000. Fill in the following table for the instruction. Values for PC, IR, MAR, MDA, and R0 should be written in hexadecimal notation; values for N, Z, and P should be written in binary notation. Fill in a column only in those phases of the instruction cycle in which a register is updated. Refer to the instruction FETCH FSM on page 1.

Instruction cycle phase	PC	MAR	MDR	IR	R2	N	Z	P
Prior to FETCH	x3000	-----------	-----------	-----------	x3FFF	-	-	-
FETCH Step 1								
FETCH Step 2								
FETCH Step 3								
DECODE	-----------	-----------	-----------	-----------	-----------	-	-	-
EXECUTE								

3. Explain the function performed by the series of instructions at x3002-x3004.

2. LC-3 Program Trace

1. Trace the execution of the given program segment until it fetches the instruction at x3006 by filling in the table below. Write down the values (in hexadecimal) stored in PC, IR, MAR, MDR, R0, R1, R2, and R3 at the **end of each instruction cycle**. Values for N, Z, and P should be written in binary. For example, the second row of the table shows the state of the processor after executing the instruction stored at address x3000, based on the values stored in the registers before the instruction was fetched. Some of the rows are already filled in to help you to stay on track.

PC	IR	MAR	MDR	R0	R1	R2	R3	N	Z	P
x3000	--------	--------	--------	x2319	x1168	x3FFF	x1658	-	-	-
x3001	x14A1	x3000	x14A1	x2319	x1168	x4000	x1658	0	0	1
x3002										
x3001	x14A1	x3000	x14A1	x2319	xA751	x4001	x8438			1
x3001	x14A1	x3000	x14A1	x2319	x7E4B	x4002	xA164			1

2. Examine your program trace. What function does the program execute? Your description should explain the high level behavior of the program in a single sentence and should not be a step-by-step description of what the program did. For example, "First the program adds R1 to R2 and stores it into R3…" would be unacceptable.

3. Systematic Decomposition and LC-3 Assembly

In this problem, you will design an algorithm to multiply two 2's complement numbers using LC-3 assembly. The two numbers are provided to you in registers R3 and R4. Your algorithm must calculate the product of the two registers and place it in R5. You may ignore overflow, but you may not make assumptions about the sign of either operand.

1. Let's start by developing an algorithm to multiply one non-negative integer by another (possibly negative) integer. This part of your algorithm should make use of the following algebraic equivalence:

$P = AB = A + A + \ldots + A$ (B times)

Assuming that the value A is stored in R3, and that the value B is stored in R4, and that $B > 0$, systematically decompose the task of calculating P and storing it into R5. Draw a flow chart of your solution. Be sure to break the tasks down to the point that each box in your flow chart can be implemented with a small number (one or two) LC-3 instructions.

2. Write out your program from **part 1** in LC-3 assembly. Ideally, you should test it and make sure that it works using the LC-3 simulator.

3. Now solve the more general problem. You should now make use of the following algebraic equivalence:

$P = AB = -(A(-B)) = -(A + A + \ldots + A)$ ($-B$ times)

Make use of your solution to **part 1** as a single box in the flow chart for this second problem.

Hint: Use another register to store the sign of B. Calculate the sign first (and take the absolute value of B if necessary), then calculate the product, then adjust the sign of the result.

4. Write out your program from **part 3** in LC-3 assembly. Ideally, you should test it and make sure that it works using the LC-3 simulator.

5. In fact, your program from **part 1** should produce the correct results even when $B < 0$, although somewhat more slowly than does your program from **part 3**. If you want to do so, you can verify this claim using the LC-3 simulator. Explain why the program produces the correct answers.