The LC-3 Instruction Set Architecture

The LC-3 ISA has three kinds of opcodes:

1. operations (with the ALU)
2. data movement (registers to/from memory)
3. control flow (conditionally change the PC)

Let’s look at each kind in turn.

The LC-3 Supports Three Operate Instructions

The LC-3 ALU is capable of three operations. The ISA includes one opcode for each operation: ADD, AND, and NOT.

Each operation uses
- one source register and
- one destination register.

The second operand (for ADD and AND only) allows a choice of addressing modes:
- register (another register) or
- immediate (a number stored in the instruction).

ADD and AND Have Two Addressing Modes

IR[5] is the mode bit for the second operand.

<table>
<thead>
<tr>
<th>opcode</th>
<th>DR</th>
<th>SR1</th>
<th>15 12 11 9 8 6 5 4 0</th>
<th>mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>0001</td>
<td>1</td>
<td>0 0 0 SR2</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>0101</td>
<td>0</td>
<td>1 imm5</td>
<td></td>
</tr>
</tbody>
</table>
What Can We Do with Immediate Mode?

Add small numbers
- +1
- -1
- (anything from -16 to +15).

Mask out high bits
- AND 1
- AND 3

Mask out low bits
- AND -2 (xFFFE)
- AND -4 (xFFFC)

Put 0 in a register! (AND 0)

Second Operand Bits are All 1 for the NOT Instruction

For NOT, IR[5:0] must be 1.

Execution of Operate Instructions in the Datapath

What if We Expanded the Register File to 16 Registers?

How would operate instructions be affected?
- Four bits for opcode
- Four bits for DR
- Four bits for SR1
- Four bits left … for SR2?

What about the mode bit?
Separate opcodes, ADDR and ADDI?
Or just remove immediate mode?
In that case, we have only a 4-bit immediate, too.
Loads and Stores Have Four Addressing Modes

Data movement instructions are of two types:
- loads (from memory to register)
- stores (from register to memory)

The LC-3 ISA supports four addressing modes for data movement instructions.

PC-Relative Addressing Adds an Offset to the PC

The first addressing mode is PC-relative.

Example of PC-Relative Addressing

What does this instruction (at memory address x1480) do?

\[ x1480 \ LD \ R3, x09 \]

In RTL: \( R3 \leftarrow M[PC + \text{SEXT16}(x09)] \)

What is PC?

Again, since execution occurs after FETCH, PC in this case is x1481. So...

\( R3 \leftarrow M[x1481] \) (NOT x1489!)

Time for a Quiz!

What’s stored at a memory address?

Bits. (16 of them, yes.)

How do we name a memory address?

Bits. (Again 16 of them.)

So we can use the bits at a memory address to name another memory address!
Indirect Addressing Accesses Memory Twice

The second addressing mode is **indirect**.

- **Indirect mode reads the address for the load/store from a PC-relative address.**

<table>
<thead>
<tr>
<th>opcode</th>
<th>SR/DR</th>
<th>imm9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1010 $\text{LDI DR} \leftarrow M[\text{PC + SEXT16}_{16}(\text{imm9})]$  
1011 $\text{STI M[PC + SEXT16}_{16}(\text{imm9})] \leftarrow SR$

What Purpose Does Indirect Mode Serve?

- Primarily to make you realize that
  - the bits at a memory address
  - can point to a memory address.

- The concept of a **pointer**
  - (just another word for a memory address)
  - is critical for understanding more complex representations in many programming languages (such as C).

Base + Offset Mode Uses Another Register

The third addressing mode is **base + offset**.

<table>
<thead>
<tr>
<th>opcode</th>
<th>SR/DR</th>
<th>BaseR</th>
<th>off6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0110 $\text{LDR DR} \leftarrow M[\text{BaseR + SEXT16}_{6}(\text{off6})]$  
0111 $\text{STR M[BaseR + SEXT16}_{6}(\text{off6})] \leftarrow SR$

Base + Offset Mode Enables Wider Memory Access

- **PC-relative and indirect** addressing
  - can only generate
  - addresses within a 9-bit offset
  - of the instruction address (-256 to +255)
  - (Indirect addressing can access any location, but the address has to be stored near the LDI/STI.)

**Base+Offset enables access to any address by using another register.**

But how do we get an address into a register?
Immediate Addressing Loads a Nearby Address

The fourth addressing mode is **immediate**.

**LEA** stands for “load effective address”:
- address generation is PC-relative, but
- memory is not accessed
  (so not really data movement).

Datapath Generation of PC-Relative Addressing

Handling the Data for Stores and Loads

For **LD**, MDR is copied over bus to DR.

For **ST**, ALU can pass input A (SR1) to bus to move to MDR.

LDI and STI Move MDR into MAR

For **LDI/STI**,  
- the PC-relative address read from memory into the **MDR**  
- can be copied across the bus into the **MAR**,  
- then used as the second access’ address.
Datapath Generation of Base+Offset Addressing

Control Flow Instructions Conditionally Change PC

After executing an instruction at address \( A \), the LC-3 next executes the instruction at address \( A + 1 \), then \( A + 2 \), and so forth.

So far, we have seen **operations** and **data movement** instructions.

But how can we do things like **if** statements and loops?

We need another kind of instruction to manage **control flow**.

**Control flow** instructions **conditionally change the PC**.

LC-3 ISA Provides Three Condition Codes: N, Z, and P

The **LC-3** maintains **three 1-bit registers** called **condition codes**.

These are based on the **last value written to the register file** (by operations or by loads).

- **N**: the last value was **negative**
- **Z**: the last value was **zero**
- **P**: the last value was **positive**

Obviously, **exactly one of these three bits is 1** in any cycle.

Conditional Branch \( BR^* \) Conditionally Changes PC

Let's start with conditional branch, **BR**.

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & n & z & p \\
15 & 12 & 11 & 9 & 8 & 1 & 0 \\
\end{array}
\]

**BEN**: \( PC \leftarrow PC + \text{SEXT}_{16}(\text{PCoffset9}) \)

The calculation of **BEN**, the **branch enable condition**, is specified in the opcode's name.

For example, **BRnp** has the \( n \) and \( p \) bits set in the instruction, while the \( z \) bit is zero.
Calculation of the Branch Enable Condition

BEN: \[ PC \leftarrow PC + \text{SEXT16(PCoffset9)} \]

The BEN condition is calculated:
- in the DECODE state
- based on instruction bits \( n \), \( z \), \( p \) and
- condition codes \( N \), \( Z \), and \( P \):

\[ \text{BEN} \leftarrow nN + zZ + pP \]

Examples of Branch Condition Names

Let’s consider a few examples...

- BRnz  branch if not positive
- BRnzp branch always
- BRnp  branch if not zero

Note: by convention, BR means BRnzp (unconditional branch), not “do nothing.”

Datapath Generation of Branch Targets (New PC Values)

new PC loaded based on BEN

new PC loaded based on BEN

JMP Instruction Copies Any Register to the PC

Branch target addresses are limited.
The BR instruction only has a 9-bit offset:
- PC is the BR address + 1, and
- add another -256 to +255.

What if we want to change PC to something farther from the current instruction?

Use a JMP (jump) instruction.* The RTL is:

\[ PC \leftarrow \text{BaseR} \]

*Look up the encoding if you need it.
Use TRAP to Invoke Operating System Services

One more instruction: TRAP.

The TRAP instruction invokes operating system services.*

The specific service depends on vector8.

*For a detailed explanation of the mechanism, read Ch. 8 & 9 of Patt and Patel, or wait for ECE220.

TRAP Functions Available in the LC-3 OS

<table>
<thead>
<tr>
<th>vec #</th>
<th>mnemonic</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>x20</td>
<td>GETC</td>
<td>read one ASCII character from keyboard into R0</td>
</tr>
<tr>
<td>x21</td>
<td>OUT</td>
<td>write one ASCII character from R0 to display</td>
</tr>
<tr>
<td>x25</td>
<td>HALT</td>
<td>end program (return control to the &quot;operating system&quot;)</td>
</tr>
</tbody>
</table>

For more detail, see p. 543 of Patt and Patel.

Do NOT Use R7 in Your ECE120 LC-3 Programs

Obviously,
- if you invoke the GETC trap,
- whatever bits were in R0 are lost.

Not so obviously,
- any TRAP will change R7.
- Do NOT use R7 for our class.

(Again, see Ch. 8 and 9 of Patt and Patel if you want to know why before ECE220.)