Instruction Set Architecture
ISA = All of the programmer-visible components and operations of the computer
- memory organization
  - address space — how many locations can be addressed?
  - addressability — how many bits per location?
- register set
  - how many? what size? how are they used?
- instruction set
  - opcodes
  - data types
  - addressing modes
ISA provides all information needed for someone that wants to write a program in machine language (or translate from a high-level language to machine language).

LC-3 Overview: Memory and Registers
Memory
- address space: $2^{16}$ locations (16-bit addresses)
- addressability: 16 bits

Registers
- temporary storage, accessed in a single machine cycle
  - accessing memory generally takes longer than a single cycle
- eight general-purpose registers: R0 - R7
  - each 16 bits wide
  - how many bits to uniquely identify a register?
- other registers
  - not directly addressable, but used by (and affected by) instructions
  - PC (program counter), condition codes

LC-3 Overview: Instruction Set
OpCodes
- 15 opcodes
- Operate Instructions: ADD, AND, NOT
- Data movement instructions: LD, LDI, LDR, LEA, ST, STR, STI
- Control instructions: BR, JSR/JSRR, JMP, RTI, TRAP
- some opcodes set/clear condition codes, based on result:
  - N = negative, Z = zero, P = positive (> 0)

Data Types
- 16-bit 2’s complement integer

Addressing Modes
- How is the location of an operand specified?
- non-memory addresses: immediate, register
- memory addresses: PC-relative, indirect, base+offset

Operate Instructions
Only three operations: ADD, AND, NOT

Source and destination operands are registers
- These instructions do not reference memory.
- ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.

Will show dataflow diagram with each instruction.
- Illustrates when and where data moves to accomplish the desired operation
Data Movement Instructions

Load -- read data from memory to register
- LD: PC-relative mode
- LDR: base+offset mode
- LDI: indirect mode (watch online lecture)

Store -- write data from register to memory
- ST: PC-relative mode
- STR: base+offset mode
- STI: indirect mode (watch online lecture)

Load effective address -- compute address, save in register
- LEA: immediate mode
  - does not access memory

PC-Relative Addressing Mode
Want to specify address directly in the instruction
- But an address is 16 bits, and so is an instruction!
  - After subtracting 4 bits for opcode and 3 bits for register, we have 9 bits available for address.

Solution:
- Use the 9 bits as a signed offset from the current PC.

9 bits: \(-256 \leq \text{offset} \leq 255\)
Can form any address \(X\), such that: \(PC - 256 \leq X \leq PC + 255\)

Remember that PC is incremented as part of the FETCH phase; This is done before the EVALUATE ADDRESS stage.
Base + Offset Addressing Mode

With PC-relative mode, can only address data within 256 words of the instruction.

- What about the rest of memory?

Solution #2:
- Use a register to generate a full 16-bit address.

4 bits for opcode, 3 for src/dst register, 3 bits for base register -- remaining 6 bits are used as a signed offset.

- Offset is sign-extended before adding to base register.

Load Effective Address

Computes address like PC-relative (PC plus signed offset) and stores the result into a register.

Note: The address is stored in the register, not the contents of the memory location.
Control Instructions

Used to alter the sequence of instructions (by changing the Program Counter)

Conditional Branch
- branch is taken if a specified condition is true
  > signed offset is added to PC to yield new PC
  > else, the branch is not taken
- PC is not changed, points to the next sequential instruction

Unconditional Branch (or Jump)
- always changes the PC

TRAP
- changes PC to the address of an OS “service routine”
- routine will return control to the next instruction (after TRAP)

Branch Instruction
Branch specifies one or more condition codes.
If the specified bit is set, the branch is taken.
- PC-relative addressing: target address is made by adding signed offset (IR[8:0]) to current PC.
  > Note: PC has already been incremented by FETCH stage.
  > Note: Target must be within 256 words of BR instruction.
If the branch is not taken, the next sequential instruction is executed.

Using Branch Instructions
Compute sum of 12 integers.
Numbers start at location x3100. Program starts at location x3000.

Condition Codes
LC-3 has three condition code registers:
N -- negative
Z -- zero
P -- positive (greater than zero)
Set by any instruction that writes a value to a register
(ADD, AND, NOT, LD, LDR, LDI, LEA)
Exactly one will be set at all times
- Based on the last instruction that altered a register

Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>x30F6</td>
<td>1 1 1 0 0 0 1 1 1 1 1 1 0 1</td>
<td>R1 = PC - 3 = x30F4</td>
</tr>
<tr>
<td>x30F7</td>
<td>0 0 0 1 0 1 0 0 1 0 1 1 1 0 0</td>
<td>R2 = R1 + 14 = x3102</td>
</tr>
<tr>
<td>x30F8</td>
<td>0 0 1 0 1 0 1 1 1 1 1 1 0 1 1</td>
<td>PC - 5 = R2</td>
</tr>
<tr>
<td>x30F9</td>
<td>0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0</td>
<td>R2 = 0</td>
</tr>
<tr>
<td>x30FA</td>
<td>0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0</td>
<td>R2 = R2 + 5 = 5</td>
</tr>
<tr>
<td>x30FB</td>
<td>0 1 1 0 1 0 0 1 0 1 1 1 0 1 1 1</td>
<td>M[R1+16] = R2</td>
</tr>
</tbody>
</table>

_opcode_

What happens if bits [11:0] are all zero? All ones?
### Sample Program

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3000</td>
<td>1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1</td>
<td>R1 ← x3100 (PC+0xFF)</td>
</tr>
<tr>
<td>x3001</td>
<td>0 1 0 1 0 1 1 0 1 1 0 0 0 0 0</td>
<td>R3 ← 0</td>
</tr>
<tr>
<td>x3002</td>
<td>0 1 0 1 0 1 0 1 0 1 0 0 0 0 0</td>
<td>R2 ← 0</td>
</tr>
<tr>
<td>x3003</td>
<td>0 0 0 1 0 1 0 1 0 1 0 1 1 0 0</td>
<td>R2 ← 12</td>
</tr>
<tr>
<td>x3004</td>
<td>0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0</td>
<td>If Z, goto x300A (PC-5)</td>
</tr>
<tr>
<td>x3005</td>
<td>0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0</td>
<td>Load next value to R4</td>
</tr>
<tr>
<td>x3006</td>
<td>0 0 0 1 0 1 1 1 1 0 0 0 1 0 0 0</td>
<td>Add R4 to R3</td>
</tr>
<tr>
<td>x3007</td>
<td>0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0</td>
<td>Increment R1 (pointer)</td>
</tr>
<tr>
<td>x3008</td>
<td>0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1</td>
<td>Decrement R2 (counter)</td>
</tr>
<tr>
<td>x3009</td>
<td>0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0</td>
<td>Goto x3004 (PC-6)</td>
</tr>
</tbody>
</table>

### Summary

- **LC-3 Memory and Registers**
- **Operate instructions:** ADD/AND/NOT
- **LC-3 Addressing modes:** PC-relative, Base+Offset
- **Data Movement Instructions:** LD/ST/LDR/STR/LEA
- **Control Instructions:** Branch
- **Online:** additional instructions, examples, LC-3 datapath