LC-3

• (Little computer version 3)
• 16-bit address width, 16-bit addressability. (Word size is 16 bits)
• 65K possible memory addresses
• Signed integers are 2’s complement
• 8 General purpose registers (GPRs), R0-R7.
  • This is our temp storage, access is one machine cycle
  • Memory access is > 1 cycle
• 3 bits for condition codes (more later)
• Fixed-length instructions, with 4-bit opcodes
Instructions

- **Data movement**: Load value into register, Store value from register into memory
- **Calculations/Ops on Data**: ADD, AND, NOT
- **Control flow**: Branch, Jump (various types)
Addressing modes

• “Where do we find our operands?”
• **Immediate:** we find it right here (in the instruction itself)
• **Register:** its sitting in one of the registers
• **Memory:** its at some memory location. How do we address it though? 3 ways:
  • **PC-relative:** effective addr = PC + offset
  • **Indirect:** effective addr = Mem[PC + offset]
  • **Base + offset:** effective addr = Reg[N] + offset
Data Movement with PC-Relative Addressing

• 3 instructions use PC-relative addressing. They all have:
  • 4 bits of opcode
  • 3 bits for register number
  • 9 bits for signed PC-relative offset: \((-256 \leq \text{offset} \leq +256\)"

• Effective address is \(\text{PC} + \text{SEXT16(Offset)}\)

• Note that PC is incremented in FETCH stage of instr cycle. When we get to EVALUATE ADDRESS stage, PC points to the next instruction
• PC offset of 0 means “address of the next instruction”
• PC offset of 1 means “the instruction after that”
• PC offset of -1 means “this instruction”
• PC offset of -2 means “the previous instruction”
• Etc.
LD (Load)

- “Load a register with the contents of the memory location at the specified address”
- Has a *destination register* (“where to put the result”)
- Reg[Dst] ← Mem[PC + offset]
Figure 5.6  Data path relevant to execution of LD R2, x1AF
LD example

• Suppose we have an instruction at address 0x2FFE
• The contents of address 0x2FF0 are 0010 011 111111000
• The first 0010 (opcode) means “this is LD”
• 011 is the dest register (R3)
• 111111000 is the PC-relative offset (it’s signed, so this is -8)
• So this means R3 ← Mem[0x3000 - 8] (Mem[0x2FF8])
ST example

- Opposite of load: “Store the value in the source register into memory”
- \( \text{Mem}[\text{PC} + \text{offset}] \leftarrow \text{Reg}[\text{Src}] \)
• For example, suppose we have an instruction at 0x2FFE:
  • 0011 011 111111000
  • ST  R3  -8
  • Mem[0x3000 - 8] ← R3
Load Effective Address (LEA)

• Similar to load, but we load the **address**, not the contents at that address!
• This instruction **does not involve a memory access**
• Reg[Dst] ← PC + offset (as opposed to Mem[PC + offset])

```
LEA 1 1 1 0 Dst PCoffset9
```
Figure 5.9  Data path relevant to the execution of LEA R5, #—3
• Suppose we have this at 0x2FFE
• 1110 011 111111000
• LEA R3 -8
• R3 ← 0x3000 - 8
Note on Assembler formats

• We get tired of writing these instructions in binary (machine code)
• We’ll use mnemonics for the opcodes, operands etc. This is called **assembly language**
• The simplest way to write PC-offset instructions uses the opcode mnemonic (LD, ST, or LEA), the reg number (R0-7), a comma, and the offset as a signed int
• Our LD example would be written: LD R3, -8
• Our ST example would be written: ST R3, -8
• Our LEA example would be written: LEA R3, -8
Data movement with Base-Offset addressing

• PC-relative addressing only has an effective range of +256 or -256 the current PC value
• This limits the addressable memory for an instruction
• Use a register to contain 16-bit base address instead
• Add 6-bits of sign extended offset to whatever address is in base reg
• Effective address = base reg + offset
LDR

• “Load using base register”
• Reg[Dest] ← Mem[Reg[Base] + offset]
Example

• 0110 011 111 111000
• LDR R3 R7 -8
• R3 ← Mem[R7 – 8]

• In assembler format this would be written: LDR R3, R7, -8
Figure 5.8  Data path relevant to the execution of LDR R1, R2, x1D
STR

• “Store using base reg”
• Mem[Base reg + offset] = Src Reg
• 0111 011 111 111000
• STR R3 R7 -8
• Mem[R7 - 8] ← R3

• In assembler format: STR R3, R7, -8
Indirect Addressing

• This addressing mode lets us use “pointers.” It gives us a way to perform single indirection via some memory address
• Use the address *that’s stored in memory*
• Here, effective addr = Mem[PC + offset]
• Rather than just PC + offset!
LDI

- "Load Indirect"
- Reg[Dest] ← Mem[Mem[PC + Offset]]
Example

• At address 0x2FFE we have the following instruction:
  • 0010 011 111111000
  • LDI R3 -8
• This means R3 ← Mem[Mem[0x3000 – 8]]
• Suppose 0x2FF8 has 0x4A30 stored in it
• Then this means R3 ← Mem[0x4A30]
• In assembler format: LDI R3, -8
Figure 5.7  Data path relevant to the execution of LDI R3, x1CC
STI

- “Store indirect”
- Mem[Mem[PC + offset]] ← Reg[Src]
Example

• At 0x2FFE we have:
• 0011 011 111111000
• STI R3 -8

• Again suppose we have the contents at 0x2FF8 (0x3000 – 8) be 0x4A30,
• Then this means Mem[0x4A30] ← R3