CS 135: Computer Architecture I

Instructor: Prof. Bhagi Narahari
Dept. of Computer Science
Course URL: www.seas.gwu.edu/~bhagiweb/cs135/

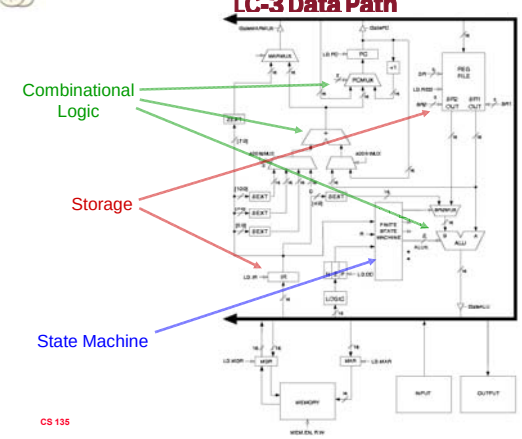
Summary: Digital Logic Circuits

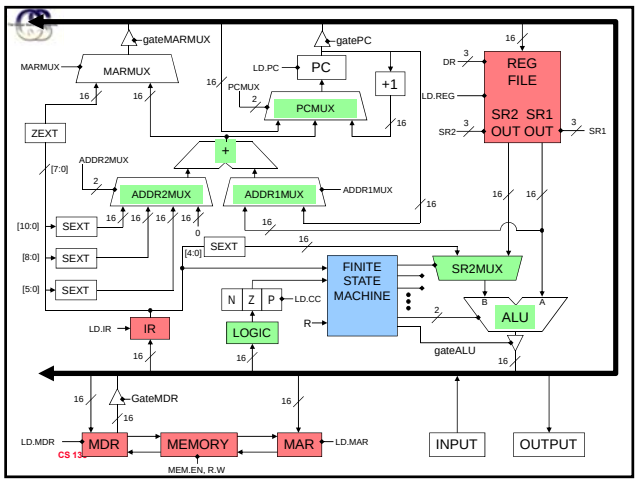
- Combinational logic
  - Basic gates, complex devices (Multiplexer, decoder, memory…)
  - Output is function of input
- Sequential logic
  - Clock
  - Flip-flops (latches): store "state" – current value
  - Output is function of input and stored values
  - Finite state diagram describes how machine functions
  - Finite state diagram to circuit design

From Logic to Processor Data Path

- The data path of a computer is all the logic used to process information.
  - Eg. data path of the LC-3.
- Combinational Logic
  - Decoders -- convert instructions into control signals
  - Multiplexers -- select inputs and outputs
  - ALU (Arithmetic and Logic Unit) -- operations on data
- Sequential Logic
  - State machine -- coordinate control signals and data movement
  - Registers and latches -- storage elements

LC-3 Data Path





What Next ?

- Next topic: The von Neumann model of computer architecture
  - Basic components
  - How instructions are processed
- The LC3 computer and instruction set
  - The ISA of LC3
  - Programming the LC3
  - Assembly Language programming
- Chapters 4,5,6,7

CS 135

Recall: what are Computers meant to do ?

- We will be solving problems that are describable in English (or Greek or French or Hindi or Chinese or ...) and using a box filled with electrons and magnetism to accomplish the task.
  - This is accomplished using a system of well defined (sometimes) transformations that have been developed over the last 50+ years.

CS 135

Problem Transformation - levels of abstraction

The desired behavior: the application

- Natural Language
- Algorithm
- Program
- Machine Architecture
- Micro-architecture
- Logic Circuits
- Devices

The building blocks: electronic devices

CS 135

Putting it all together

- The goal:
  - Turn a theoretical device - Turing's Universal Computational Machine - into an actual computer ...
  - … interacting with data and instructions from the outside world, and producing output data.
- Smart building blocks:
  - We have at our disposal a powerful collection of combinational and sequential logic devices.
- Now we need a master plan ...

The Stored Program Computer

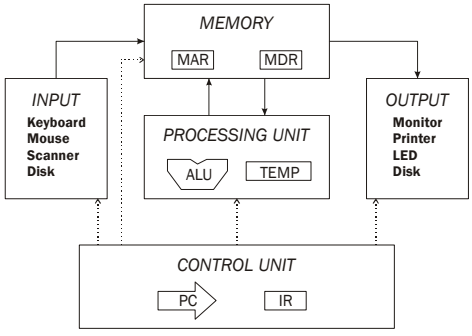
- 1943: ENIAC
  - Presper Eckert and John Mauchly -- first general electronic computer. (or was it John V. Atanasoff in 1939?)
  - Hard-wired program -- settings of dials and switches.
- 1944: Beginnings of EDVAC
  - among other improvements, includes program stored in memory
- 1945: John von Neumann
  - wrote a report on the stored program concept, known as the *First Draft of a Report on EDVAC*
- The basic structure proposed in the draft became known as the "von Neumann machine" (or model).
  - a ***memory***, containing instructions and data
  - a ***processing unit***, for performing arithmetic and logical operations
  - a ***control unit***, for interpreting instructions

For more history, see http://www.maxmon.com/history.htm

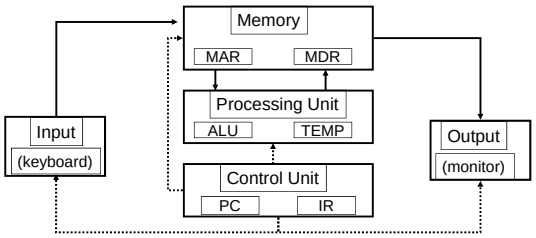
Von Neumann Model

- The central idea in the von Neumann model of computer processing is that
  - the *program* and *data* are both *stored* as sequences of bits in the *computer's memory*, and
  - the program is executed, one instruction at a time, under the direction of the control unit.

Von Neumann Model



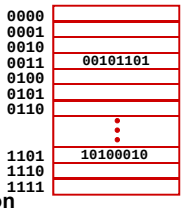
The von Neumann Model



- Memory: holds both data and instructions
- Processing Unit: carries out the instructions
- Control Unit: sequences and interprets instructions
- Input: external information into the memory
- Output: produces results for the user

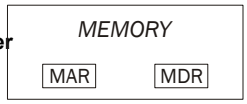
Von Neuman Model: Memory

- 2^k x m array of stored bits
- Address
  - unique (k-bit) identifier of location
- Contents/Addressability
  - m-bit value stored in location
- Basic Operations:
- LOAD (READ)
  - read a value from a memory location
- STORE (WRITE)
  - write a value to a memory location



Interface to Memory

- How does processing unit get data to/from memory?
- MAR: Memory Address Register
- MDR: Memory Data Register
  - Also called MBR: mem. Buffer reg.
- To LOAD a location (A):
  1. Write the address (A) into the MAR.
  2. Send a "read" signal to the memory.
  3. Read the data from MDR.
- To STORE a value (X) to a location (A):
  1. Write the data (X) to the MDR.
  2. Write the address (A) into the MAR.
  3. Send a "write" signal to the memory.



Von Neumann Model: Processing Unit

- Processing Unit- does the actual work!
  - Can consist of many units, each specializing in one complex function.
  - At a minimum, has Arithmetic & Logic Unit (ALU) and General Purpose Registers (GPRs).
  - The number of bits a basic Processing Unit operation can handle is called the WORD SIZE of the machine.
- ALU
  - Performs basic operations: add, subtract, and, not, etc.
  - Generally operates on whole words of data.
  - Some can also operate on subsets of words (eg. single bits or bytes)
  - LC3 does ADD, AND, NOT
- Registers:
  - Fast "on-board" storage for a small number of words.
  - Invaluable for intermediate data storage while processing
  - Close to the ALU (much faster access than RAM)
  - LC3 has 8 general purpose registers R_0,R_1,…,R_7.

Von Neumann Model: Input and Output

- Devices for getting data into and out of computer memory - peripherals

INPUT	OUTPUT
Keyboard	Monitor
Mouse	Printer
Scanner	LED
Disk	Disk

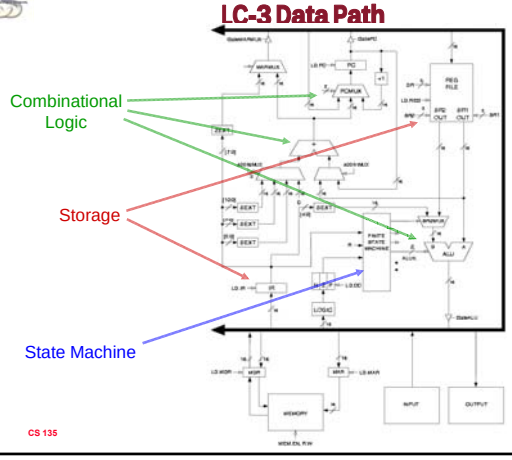
- Each device has its own interface, usually a set of registers like the memory's MAR and MDR
  - LC-3 supports keyboard (input) and monitor (output)
  - keyboard: data register (KBDR) and status register (KBSR)
  - monitor: data register (DDR) and status register (DSR)
- Some devices provide both input and output
  - disk, network
- Program that controls access to a device is usually called a *device driver*.

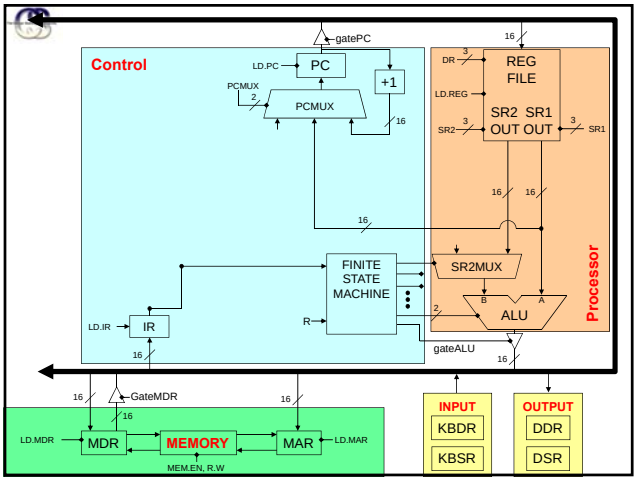
Von Neumann Model: Control Unit

- Orchestrates execution of the program

- **Instruction Register** (IR) contains the ***current instruction***.
- **Program Counter** (PC) contains the ***address*** of the next instruction to be executed.
- **Control unit**:
  - reads an instruction from memory
    - the instruction's address is in the PC
  - interprets the instruction, generating signals that tell the other components what to do
    - an instruction may take many *machine cycles* to complete

LC-3 Data Path





What is an Instruction

- The instruction is the fundamental unit of work.
- Specifies two things:
  - *opcode*: operation to be performed
  - *operands*: data/locations to be used for operation
- An instruction is encoded as a sequence of bits. *(Just like data!)*
  - Often, but not always, instructions have a fixed length (16,32,..),
  - Control unit interprets instruction:
    - generates sequence of control signals to carry out operation.
  - Operation is either executed completely, or not at all.
- A computer's instructions and their formats is known as its *Instruction Set Architecture (ISA)*.

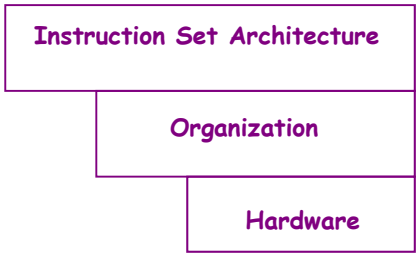
ISA

- The ISA specifies all the information about the computer that the software needs to be aware of.
- Who uses an ISA?
- What is specified?
- How big an ISA
  - Reduced Instruction set (RISC)
  - Complex Instruction set (CISC)

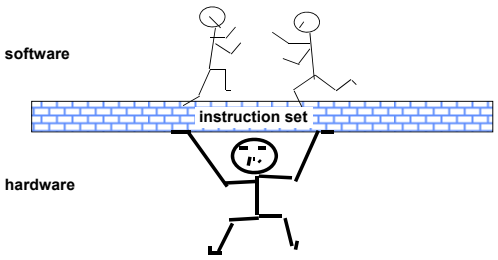
Instruction Set Architecture

- ISA = All of the *programmer-visible* components and operations of the computer
  - memory organization
    - address space -- how may locations can be addressed?
    - addressibility -- 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).

Computer Architecture is ...

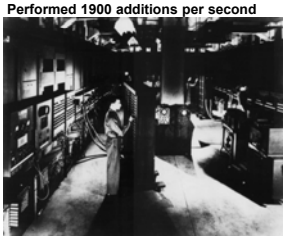


What is the Hardware/Software Interface ?



Historical Perspective

- ENIAC built in World War II was the first general purpose computer
  - Used for computing artillery firing tables
  - 80 feet long by 8.5 feet high and several feet wide
  - Each of the twenty 10 digit registers was 2 feet long
  - Used 18,000 vacuum tubes
  - Performed 1900 additions per second

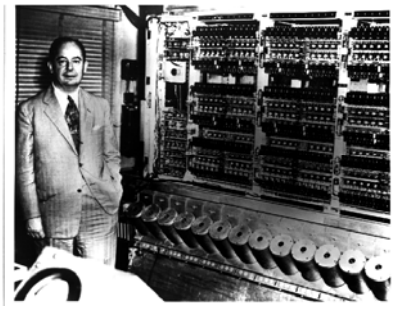


–Since then:

Moore's Law:

transistor capacity doubles every 18-24 months

Harvard Mark I



John von Neumann & EDVAC

What is an Instruction

- The instruction is the fundamental unit of work.
- Specifies two things:
  - *opcode*: operation to be performed
  - *operands*: data/locations to be used for operation
- An instruction is encoded as a sequence of bits. *(Just like data!)*
  - Often, but not always, instructions have a fixed length (16,32,..),
  - Control unit interprets instruction:
    - generates sequence of control signals to carry out operation.
  - Operation is either executed completely, or not at all.
- A computer's instructions and their formats is known as its *Instruction Set Architecture (ISA)*.

ISA

- The ISA specifies all the information about the computer that the software needs to be aware of.
- Who uses an ISA?
- What is specified?
- How big an ISA
  - Reduced Instruction set (RISC)
  - Complex Instruction set (CISC)

Instruction Set Architecture

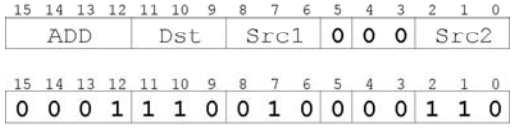
- ISA = All of the *programmer-visible* components and operations of the computer
  - memory organization
    - address space -- how may locations can be addressed?
    - addressibility -- how many bits per location?
  - register set
    - how many? what size? how are they used?
  - instruction set
    - 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).

ISA: Types of Instruction

- *1. Operate* Instructions
  - process data (addition, logical operations, etc.)
- *2. Data Movement* Instructions …
  - move data between memory locations and registers.
- *3. Control* Instructions …
  - change the sequence of execution of instructions in the stored program.
    - The default is sequential execution: the PC is incremented by 1 at the start of every Fetch, in preparation for the next one.
    - Control instructions set the PC to a new value during the Execute phase, so the next instruction comes from a different place in the program.
    - This allows us to build control structures such as loops and branches.

Example: LC-3 ADD Instruction

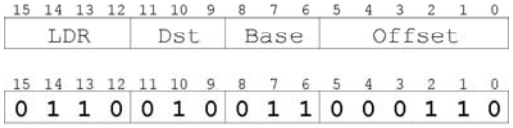
- LC-3 has 16-bit instructions.
  - Each instruction has a four-bit opcode, bits [15:12].
- LC-3 has eight *registers* (R0-R7) for temporary storage.
  - Sources and destination of ADD are registers.



"Add the contents of R2 to the contents of R6, and store the result in R6."

Example: LC-3 LDR Instruction

- Load instruction -- reads data from memory
- Base + offset mode:
  - add offset to base register -- result is memory address
  - load from memory address into destination register



"Add the value 6 to the contents of R3 to form a memory address. Load the contents of that memory location to R2."

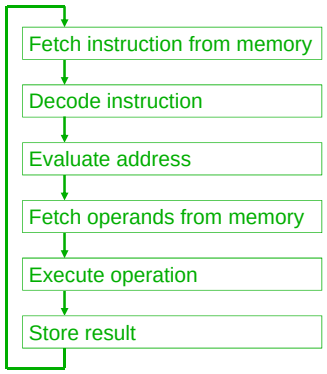
How do instructions get executed ? Instruction Cycle - overview

- The Control Unit orchestrates the complete execution of each instruction:
  - At its heart is a Finite State Machine that sets up the state of the logic circuits according to each instruction.
  - This process is governed by the system clock - the FSM goes through one transition ("machine cycle") for each tick of the clock.
  - 1 Ghz (10^9) clock frequency = 1 nanosecond clock cycle

Instruction Cycle - overview

- Six phases of the complete Instruction Cycle
  - **Fetch**: load IR with instruction from memory
  - **Decode**: determine action to take (set up inputs for ALU, RAM, etc.)
  - **Evaluate address**: compute memory address of operands, if any
  - **Fetch operands**: read operands from memory or registers
  - **Execute**: carry out instruction
  - **Store results**: write result to destination (register or memory)

Instruction Processing



CS 135

Instruction Processing Step 1: FETCH

- Load next instruction (at address stored in PC) from memory into Instruction Register (IR).
  - 1.Copy contents of PC into MAR: MAR ← (PC)
  - 2.Send "read" signal to mem and read: MDR ← (MAR)
  - 3.Copy contents of MDR into IR: IR ← MDR
  - 4. increment PC, so that it points to next inst in sequence: PC = PC+1
- FETCH takes at least 3 steps/cycles
  - 1,3,4 take one cycle, but 2 can take more
  - 1,4 can be done in same cycle

CS 135

Instruction Processing Step 2: DECODE

- First identify the opcode.
  - In LC-3, this is always the first four bits of instruction.
    - A 4-to-16 decoder asserts a control line corresponding to the desired opcode.
- Depending on opcode, identify other operands from the remaining bits.
  - Example:
    - for LDR, last six bits is offset
    - for ADD, last three bits is source operand #2

CS 135

Instruction Processing Step 3: EVALUATE ADDRESS

- For instructions that require memory access, compute address used for access.
  - Called Effective Address (EA)
- Examples:
  - add offset to base register (as in LDR)
  - add offset to PC
  - add offset to zero

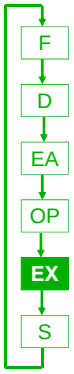
CS 135

Instruction Processing Step 4: FETCH OPERANDS

- Obtain source operands needed to perform operation.
  - Effective address computed in previous step used to fetch operands
- Examples:
  - load data from memory (LDR)
  - read data from register file (ADD)

Instruction Processing Step 5: EXECUTE

- Perform the operation, using the source operands.
- Examples:
  - send operands to ALU and assert ADD signal
  - do nothing (e.g., for loads and stores)



Instruction Processing Step 6: STORE RESULT

- Write results to destination. (register or memory)
- Examples:
  - result of ADD is placed in destination register
  - result of memory load is placed in destination register
  - for store instruction, data is stored to memory
    - write address to MAR, data to MDR
    - assert WRITE signal to memory

Instruction Processing Cycle - step 7

- Start over …
  - The control unit just keeps repeating this whole process: so it now Fetches a new instruction from the address currently stored in the PC.
    - Recall that the PC was incremented in the first step (FETCH), so the instruction retrieved will be the next in the program as stored in memory - unless the instruction just executed changed the contents of the PC.
- Note: Some instructions don't need all 6 phases
  - If only using registers, skip Evaluate Address
  - If only moving data, skip Execute

Flow Control

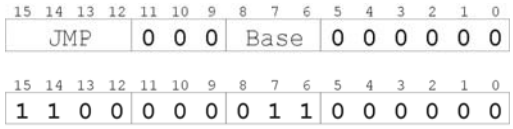
- Normally we execute instructions one after another
- When might we not want to do this?

Changing the Sequence of Instructions

- In the FETCH phase, we increment the Program Counter by 1.
- What if we don't want to always execute the instruction that follows this one?
  - examples: loop, if-then, function call
- Need special instructions that change the contents of the PC.
- These are called *control instructions*.
  - jumps are unconditional -- always change the PC
  - branches are conditional -- change the PC only if some condition is true (e.g., the result of an ADD is zero)

Example: LC-3 JMP Instruction

- Set the PC to the value contained in a register. This becomes the address of the next instruction to fetch.



"Load the contents of R3 into the PC."

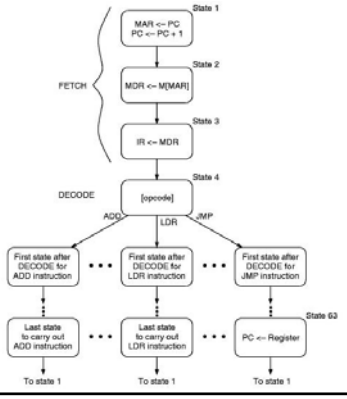
Instruction Processing Summary

- Instructions look just like data -- it's all interpretation.
- Three basic kinds of instructions:
  - Compute/operate instructions (ADD, AND, …)
  - data movement instructions (LD, ST, …)
  - control instructions (JMP, BRnz, …)
- Six basic phases of instruction processing:
- $F \rightarrow D \rightarrow EA \rightarrow OP \rightarrow EX \rightarrow S$
  - not all phases are needed by every instruction
  - phases may take variable number of machine cycles

Control Unit State Diagram

- The control unit is a state machine
  - Transition from state to state based on the steps in the instruction cycle, the opcode, and outcome (for branches)
  - simplified state diagram for the LC-3
  - Appendix C has complete state diagram

The Instruction Cycle as FSM

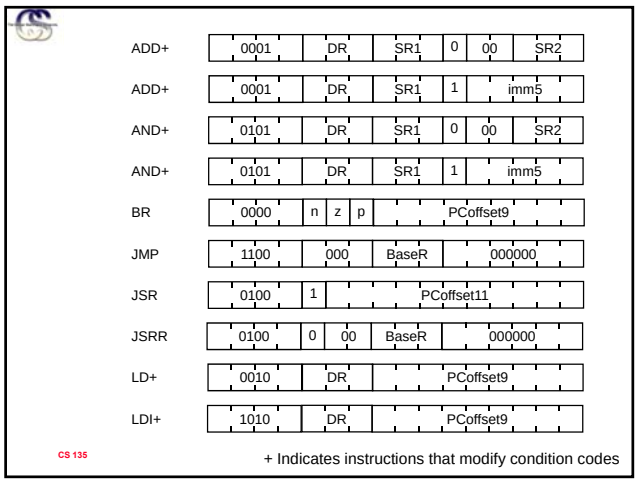


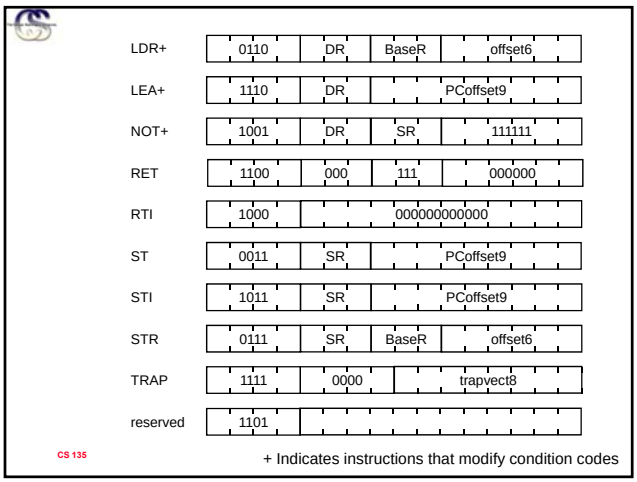
Next..

- The Instruction set architecture (ISA) of the LC3
  - How is each instruction implemented by the control and data paths in the LC3
  - Programming in machine code
  - How are programs executed
    - Memory layout, programs in machine code
- Assembly programming
  - Assembly and compiler process
  - Assembly programming with simple programs

LC 3 Instruction Set

- The Instruction set architecture (ISA) of the LC3
  - How is each instruction implemented by the control and data paths in the LC3
  - Programming in machine code
  - How are programs executed
    - Memory layout, programs in machine code
- Assembly programming
  - Assembly and compiler process
  - Assembly programming with simple programs





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

CS 135

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*

CS 135

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

- GPR ↔ Memory
- GPR ↔ I/O Devices
- GPR ← Memory ???
- Memory ← GPR ???

Addressing Modes

- Where can operands be found?

1

2

3

Data Movement Instructions

- Load -- read data from memory to register
  - LD: PC-relative mode
  - LDR: base+offset mode
  - LDI: indirect mode
- Store -- write data from register to memory
  - ST: PC-relative mode
  - STR: base+offset mode
  - STI: indirect mode
- 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.

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)

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

Branch Instruction

- Branch specifies one or more condition codes.
- If the set bit is specified, 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.