Today….What Next?

• We are (almost) done with HW circuits and devices
  • What remains: how to put them together to implement a computer
• Next topic: The von Neumann model of computer architecture; Chapter 4,5
  • Basic components
  • How instructions are processed
• The LC3 computer and instruction set; chapters 6-10
  • The ISA of LC3
  • Programming the LC3
  • Assembly Language programming

Important Note: Building circuits using ‘standard’ devices

• now that we have a set of combinational devices, we can build/design circuits using these devices from a “library”
  • Adders, Decoders, Multiplexers, Flip Flops, Registers, Memory,…..
  • You do not have to keep going to the transistor or gate level when designing a ‘system’
• Analogous to using library functions (or functions you have implemented earlier) to write your program
  • Work on formulating a solution/design by using ‘high level’ abstractions/devices
  • Example: Need to store a value, think ‘register’ or ‘memory’ instead of ‘RS latch’

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.
  • We now know how to get electronics to run around!
Problem Transformation
- levels of abstraction

<table>
<thead>
<tr>
<th>The desired behavior: the application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural Language</td>
</tr>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>Program</td>
</tr>
<tr>
<td>Machine Architecture</td>
</tr>
<tr>
<td>Micro-architecture</td>
</tr>
<tr>
<td>Logic Circuits</td>
</tr>
<tr>
<td>Devices</td>
</tr>
</tbody>
</table>

| The building blocks: electronic devices |

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 ...
- Model of a computer: von Neumann architecture
  - Today we focus on the definitions of the different components

From Logic to Processor Design
- 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

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.maxmen.com/history.htm](http://www.maxmen.com/history.htm)
Historical Perspective
• ENIAC built during 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

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.

John von Neumann & EDVAC
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

- $2^k \times m$ array of stored bits
- Address
  - unique ($k$-bit) identifier of location
- Contents/Addressability
  - $m$-bit value stored in location
- Abstraction: Interacting with memory
  - (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, i.e., enable Write

Reality…Memory access
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 (e.g., single bits or bytes)
  - LC3 does ADD, AND, NOT
  - You have seen a design of a simple ALU (to Add/Compare or Add/Sub!)
- 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 R0, R1,…,R7.

Von Neumann Model: Input and Output

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

  ![CONTROL UNIT Diagram](image)

  - IR (Instruction Register) contains the **current instruction**
  - PC (Program Counter) contains the **address** of the next instruction to be executed

- **Control unit**
  - Reads an instruction from memory and stores it in IR (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)

---

**Instruction**

- An instruction if the fundamental unit of work
- Specifies two things:
  - **Opcode**: operation to be performed
  - **Operand**: data/locations to be used for the operation

- Instruction is encoded as a sequence of bits (just like data)
  - Often instructions have fixed length (16 bit, 32 bit, 64 bit, ...)
  - Control unit interprets instruction based on the encoding
    - 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 many 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 ...

![Diagram of Hardware/Software Interface]

**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.

**Encoding the operations/opcode**
- N-bit word used by processor (addressability)
- Use some of these bits to encode the different instructions
- Example: We have 32-bit processor
  - We have 50 instructions we need to encode
  - We need 6 bits to encode 50 different binary strings
    - Opcode is specified using these 6 bits
    - Remaining 26 bits available to specify operands
- In reality: could get more ‘creative’ than just sticking to these 6 bits…..
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.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Dst</td>
<td>Src1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Src2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Dst is destination/output register. Src1, Src2 are the source/input registers

Encoding using binary 3-bit

Semantics: “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

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR</td>
<td>Dst</td>
<td>Base</td>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

“Semantics: 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

CPU + memory

- Address and data exchange
- Instruction execution cycle
Instruction Cycle – Six phases

Six phases of the complete Instruction Cycle

1. Fetch: load IR with instruction from memory
2. Decode: determine action to take (set up inputs for ALU, RAM, etc.)
3. Evaluate address: compute memory address of operands, if any
4. Fetch operands: read operands from memory or registers
5. Execute: carry out instruction
6. Store results: write result to destination (register or memory)

The Von Neumann “Loop”

- A Von Neumann Processor essentially does this
  - Fetch instruction at PC
  - Decode instruction (i.e., convert to control signals)
  - Execute instruction (read inputs, operate, write output)
  - Update PC
  - Rinse
  - Repeat

Critical requirement
- Each iteration of this loop must appear atomic (all or nothing)
- Key word from programmer perspective? Atomic
  - Maintains sanity
- Key word from hardware perspective? Appear
  - Enables lot of cool performance tricks

Instruction Processing

1. Fetch instruction from memory
2. Decode instruction
3. Evaluate address
4. Fetch operands from memory
5. Execute operation
6. Store result

What actions take place in each step....

- Next, take a closer look at the “control” signals needed and the actions that take place at each step of the instruction cycle
  - At a later time, go into the actions/steps to implement each instruction

- Important: you will need these concepts when you implement a processor in Project 3!

- Next week we will go into detail on how the processor datapath and control is implemented, and some sequential logic devices.
**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: \( \text{MAR} \leftarrow (\text{PC}) \)
  - 2. Send “read” signal to mem and read: \( \text{MDR} \leftarrow (\text{MAR}) \)
  - 3. Copy contents of MDR into IR: \( \text{IR} \leftarrow \text{MDR} \)
  - 4. Increment PC, so that it points to next instruction sequence: \( \text{PC} = \text{PC} + 1 \)
- 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

**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

**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

**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.

```
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
JMP 0 0 0 Base 0 0 0 0 0 0 0 0 0
```

“Load the contents of R3 into the PC.”

Early programming languages had a “GOTO ..” statement

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 → D → EA → OP → EX → S
  • not all phases are needed by every instruction
  • phases may take variable number of machine cycles

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
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 FETCH phase

- 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
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
  - Reading: The textbook does a really good job covering the materials
    - Start getting familiar with the LC3 instruction set
      - Good news: only 15 instructions to remember!
      - Bad news: pain in the **** to program using only these 15!
- Assembly programming
  - Assembly and compiler process
  - Assembly programming with simple programs

### FSM diagram for FETCH phase

State 1
- MAR = PC
- PC = PC + 1

State 2
- MDR = (MAR)
- Read

State 3
- IR = MDR

Note: State 2 will have to wait for memory to finish Read operation...Loop until read

### The Instruction Cycle as FSM

### Instruction Set for the LC3

- **ADD+**: 0001
  - DR
  - SR1
  - 0
  - 0
  - SR2

- **ADD-**: 0001
  - DR
  - SR1
  - 1
  - ammb

- **AND+**: 0101
  - DR
  - SR1
  - 0
  - 0
  - SR2

- **AND-**: 0101
  - DR
  - SR1
  - 1
  - imm5

- **BR**: 0000
  - n
  - z
  - p
  - PCoffset9

- **JMP**: 1100
  - 000
  - BaseR
  - 00000

- **JSR**: 0100
  - 1
  - PCoffset11

- **JSRR**: 0100
  - 0
  - BaseR
  - 200000

- **LD+**: 0010
  - DR
  - PCoffset9

- **LDI+**: 1010
  - DR
  - PCoffset9

*Indicates instructions that modify condition codes*
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Format</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR+</td>
<td>0110 DR BaseR offset6</td>
<td>+ Indicates instructions that modify condition codes</td>
</tr>
<tr>
<td>LEA+</td>
<td>1110 DR PCoffset9</td>
<td></td>
</tr>
<tr>
<td>NOT+</td>
<td>1001 DR SR 111111</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>1100 000 111 000000</td>
<td></td>
</tr>
<tr>
<td>RTI</td>
<td>1000 0000000000000000</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>0011 SR PCoffset9</td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>1011 SR PCoffset9</td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>0111 SR BaseR offset6</td>
<td></td>
</tr>
<tr>
<td>TRAP</td>
<td>1111 0000 trapvech</td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>1101</td>
<td></td>
</tr>
</tbody>
</table>