Ceng 111 – Fall 2015
Week 5a

Computer Organization

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.
Binary Representation of Textual Information

- Characters are mapped onto binary numbers
  - ASCII (American Standard Code for Information Interchange) code set
    - Originally: 7 bits per character; 128 character codes
  - Unicode code set
    - 16 bits per character
  - UTF-8 (Universal Character Set Transformation Format) code set.
    - Variable number of 8-bits.
### Binary Representation of Textual Information (cont’d)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Val.</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>00110000</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>00110001</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>00110010</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>00110011</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>00110100</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>00110101</td>
<td>5</td>
</tr>
<tr>
<td>54</td>
<td>00110110</td>
<td>6</td>
</tr>
<tr>
<td>55</td>
<td>00110111</td>
<td>7</td>
</tr>
<tr>
<td>56</td>
<td>00111000</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>00111001</td>
<td>9</td>
</tr>
<tr>
<td>58</td>
<td>00111010</td>
<td>:</td>
</tr>
<tr>
<td>59</td>
<td>00111011</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>00111100</td>
<td>&lt;</td>
</tr>
<tr>
<td>61</td>
<td>00111101</td>
<td>=</td>
</tr>
<tr>
<td>62</td>
<td>00111110</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>00111111</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>01000000</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>01000001</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>01000010</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dec.</th>
<th>Unicode</th>
<th>Charac.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30</td>
<td>0x0030</td>
<td>0</td>
</tr>
<tr>
<td>0x31</td>
<td>0x0031</td>
<td>1</td>
</tr>
<tr>
<td>0x32</td>
<td>0x0032</td>
<td>2</td>
</tr>
<tr>
<td>0x33</td>
<td>0x0033</td>
<td>3</td>
</tr>
<tr>
<td>0x34</td>
<td>0x0034</td>
<td>4</td>
</tr>
<tr>
<td>0x35</td>
<td>0x0035</td>
<td>5</td>
</tr>
<tr>
<td>0x36</td>
<td>0x0036</td>
<td>6</td>
</tr>
<tr>
<td>0x37</td>
<td>0x0037</td>
<td>7</td>
</tr>
<tr>
<td>0x38</td>
<td>0x0038</td>
<td>8</td>
</tr>
<tr>
<td>0x39</td>
<td>0x0039</td>
<td>9</td>
</tr>
<tr>
<td>0x3A</td>
<td>0x003A</td>
<td>:</td>
</tr>
<tr>
<td>0x3B</td>
<td>0x003B</td>
<td>;</td>
</tr>
<tr>
<td>0x3C</td>
<td>0x003C</td>
<td>&lt;</td>
</tr>
<tr>
<td>0x3D</td>
<td>0x003D</td>
<td>=</td>
</tr>
<tr>
<td>0x3E</td>
<td>0x003E</td>
<td>&gt;</td>
</tr>
<tr>
<td>0x3F</td>
<td>0x003F</td>
<td>?</td>
</tr>
<tr>
<td>0x40</td>
<td>0x0040</td>
<td>@</td>
</tr>
<tr>
<td>0x41</td>
<td>0x0041</td>
<td>A</td>
</tr>
<tr>
<td>0x42</td>
<td>0x0042</td>
<td>B</td>
</tr>
</tbody>
</table>

**ASCII**: 7 bits long

**Unicode**: 16 bits long

Partial listings only!
## UTF-8 Illustrated

<table>
<thead>
<tr>
<th>Bits</th>
<th>Last code point</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>U+007F</td>
<td>0xxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+07FF</td>
<td>110xxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>U+FFFF</td>
<td>1110xxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>U+F1FFFF</td>
<td>11110xxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>U+F3FFFFF</td>
<td>111110xx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>U+F7FFFFF</td>
<td>1111110x</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character</th>
<th>Binary code</th>
<th>Binary UTF-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>§</td>
<td>U+0024</td>
<td>0100100</td>
</tr>
<tr>
<td>¶</td>
<td>U+00A2</td>
<td>00010100010</td>
</tr>
<tr>
<td>€</td>
<td>U+20AC</td>
<td>0010000010101100</td>
</tr>
<tr>
<td>髋</td>
<td>U+24B62</td>
<td>000100100101101100010</td>
</tr>
</tbody>
</table>
How about a text?

Text in a computer has two alternative representations:

1. A fixed-length number representing the length of the text followed by the binary values of the characters in the text.
   - Ex: “ABC” =>
     00000011 01000001 01000001 01000001 (3 ‘A’ ‘B’ ‘C’)

2. Binary values of the characters in the text ended with a unique marker, like “00000000” which has no value in the ASCII table.
   - Ex: “ABC” =>
     01000001 01000001 01000001 00000000 (‘A’ ‘B’ ‘C’ END)
Digitization of an Analog Signal

(a) Sampling the Original Signal

(b) Recreating the Signal from the Sampled Values
Binary Representation of Sound and Images (continued)

Representing image data

- Images are sampled by reading color and intensity values at even intervals across the image
- Each sampled point is a pixel
- Image quality depends on number of bits at each pixel

More image information:
http://cat.xula.edu/tutorials/imaging/grayscale.php

http://photo.net/equipment/digital/basics/
The Reliability of Binary Representation

- Electronic devices are most reliable in a bistable environment

- Bistable environment
  - Distinguishing only two electronic states
    - Current flowing or not, or
    - Direction of flow

- Computers are bistable: hence binary representations
Von Neumann Architecture & Its Implementation

- How are instructions coded?
- How are instructions executed?
- How do the different subcomponents interact?
  - Memory
  - ALU
  - The Bus System
  - Registers
The Components of a Computer System

- Von Neumann architecture has four functional units:
  - Memory
  - Input/Output
  - Arithmetic/Logic unit
  - Control unit

- Sequential execution of instructions
- Stored program concept
Instruction Execution
Memory and Cache

- Information is stored and fetched from memory subsystem
- Memory maps addresses to memory locations
- Cache memory keeps values currently in use in faster memory to speed access times
Memory and Cache (continued)

- RAM (Random Access Memory)
  Often called memory, primary memory
  - Memory made of addressable “cells”
  - Cell size is 8 bits
    - Nowadays, it is 32 or 64 bits.
  - All memory cells accessed in equal time
  - Memory address
    - Unsigned binary number $N$ long
    - Address space is then $2^N$ cells
Memory and Cache (continued)

Parts of the memory subsystem

- Fetch/store (or Read/Write) controller
  - **Fetch**: retrieve a value from memory
  - **Store**: store a value into memory
- Memory address register (MAR)
- Memory data register (MDR)
Memory and Cache (continued)

- Fetch operation
  - The address of the desired memory cell is moved into the MAR
  - Fetch/store controller signals a “fetch,” accessing the memory cell
  - The value at the MAR’s location flows into the MDR
Memory and Cache (continued)

- Store operation
  - The address of the cell where the value should go is placed in the MAR
  - The new value is placed in the MDR
  - Fetch/store controller signals a “store,” copying the MDR’s value into the desired cell
Overall RAM Organization

$2^N$ memory cells organized into a $2^{N/2} \times 2^{N/2}$ square as shown in Figure 5.6
Cache Memory

- Memory access is much slower than processing time
- Faster memory is too expensive to use for all memory cells
- Locality principle
  - Once a value is used, it is likely to be used again
- Small size, fast memory just for values currently in use speeds computing time
Previously on CENG 111!

From Computer Desktop Encyclopedia
© 1999 The Computer Language Co. Inc.
The Arithmetic/Logic Unit

- **Arithmetic and Logic Unit**
  - “Manufacturing” section of computer
  - Contains decision mechanisms and can make calculations+comparisons
  - Actual computations are performed

- **Primitive operation circuits**
  - Arithmetic [+,-, *, /]
  - Comparison [equality or CE, GT, LT, NEQ]
  - Logic [AND, OR, NOT, XOR]

- Data inputs and results stored in registers
- Multiplexer selects desired output
Using a Multiplexor Circuit to Select the Proper ALU Result
(Not totally correct)
The Arithmetic/Logic Unit (continued)

**ALU process**

- Values for operations copied into ALU’s input register locations
- All circuits compute results for those inputs
- Multiplexor selects the one desired result from all values *(Not totally correct)*
- Result value copied to desired result register
The Control Unit

- Manages stored program execution

Task

1. **Fetch** from memory the next instruction to be executed
2. **Decode it**: determine what is to be done
3. **Execute it**: issue appropriate command to ALU, memory, and I/O controllers
Machine Language Instructions

- Can be decoded and executed by control unit

- Parts of instructions
  - Operation code (op code)
    - Unique unsigned-integer code assigned to each machine language operation
  - Address field(s)
    - Memory addresses of the values on which operation will work
Typical Machine Language Instruction Format
More on Instructions

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Address field 1</th>
<th>Address field 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE X, Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD X, Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPARE X, Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUMP X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUMPGT X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **LOAD X** -> Load register R with the contents of memory cell X
- **STORE X** -> Store register R into memory cell X
- **MOVE X, Y** -> Copy the contents of X into Y
- **ADD X** -> Add contents of X to the contents of R
- **ADD X, Y** -> Add contents of X to the contents of Y, and put the result in register R
- **COMPARE X, Y** -> Set GT (greater than), EQ (equal) and LT (less than) condition codes
- **JUMP X** -> Jump unconditionally to the instruction in cell X
- **JUMPGT X** -> Jump, if GT=1, to the instruction in cell X
- **HALT**
Now

- Continue on how the computer works
Machine Language Instructions (continued)

- Types of machine instructions:
  - Data transfer
    - Move values to and from memory and registers
  - Arithmetic/logic
    - Perform ALU operations that produce numeric values
  - Compares
    - Set bits of compare register to hold result
  - Branches
    - Jump to a new memory address to continue processing
<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Value of a</td>
</tr>
<tr>
<td>101</td>
<td>Value of b</td>
</tr>
<tr>
<td>102</td>
<td>Value of c</td>
</tr>
</tbody>
</table>

**Algorithmic notation**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>(Commentary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>LOAD 101</td>
<td>Put the value of b into register R.</td>
</tr>
<tr>
<td>51</td>
<td>ADD 102</td>
<td>Add c to register R. It now holds b + c.</td>
</tr>
<tr>
<td>52</td>
<td>STORE 100</td>
<td>Store the contents of register R into a.</td>
</tr>
</tbody>
</table>

1. Set a to the value \( b + c \)

2. If \( a > b \) then
   - set c to the value \( a \)
   - else
     - set c to the value \( b \)

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>(Commentary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>COMPARE 100, 101</td>
<td>Compare ( a ) and ( b ) and set condition codes. Go to location 54 if ( a &gt; b ).</td>
</tr>
<tr>
<td>51</td>
<td>JUMPGT 54</td>
<td>Get here if ( a \leq b ), so move ( b ) into ( c ) and skip the next instruction.</td>
</tr>
<tr>
<td>52</td>
<td>MOVE 101, 102</td>
<td>Move ( a ) into ( c ).</td>
</tr>
<tr>
<td>53</td>
<td>JUMP 55</td>
<td>Next statement begins here.</td>
</tr>
<tr>
<td>54</td>
<td>MOVE 100, 102</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>⋯</td>
<td></td>
</tr>
</tbody>
</table>
Control Unit Registers and Circuits

- Parts of control unit
  - Linked to other subsystems
  - Instruction decoder circuit (main responsibility)
- Two special registers:
  - Program Counter (PC)
    - Stores the memory address of the next instruction to be executed
  - Instruction Register (IR)
    - Stores the code for the current instruction
Organization of the Control Unit Registers and Circuits
List of BC Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Size (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>16</td>
<td>Data Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds memory operand</td>
</tr>
<tr>
<td>AR</td>
<td>12</td>
<td>Address Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds address for memory</td>
</tr>
<tr>
<td>AC</td>
<td>16</td>
<td>Accumulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Processor register</td>
</tr>
<tr>
<td>IR</td>
<td>16</td>
<td>Instruction Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds instruction code</td>
</tr>
<tr>
<td>PC</td>
<td>12</td>
<td>Program Counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds address of instruction</td>
</tr>
<tr>
<td>TR</td>
<td>16</td>
<td>Temporary Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds temporary data</td>
</tr>
<tr>
<td>INPR</td>
<td>8</td>
<td>Input Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds input character</td>
</tr>
<tr>
<td>OUTR</td>
<td>8</td>
<td>Output Register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Holds output character</td>
</tr>
</tbody>
</table>
Main Types of Registers

- **User-accessible Registers:**
  - The most common division of user-accessible registers is into data registers and address registers.

- **Data registers:** hold numeric values such as integer and floating-point values.
  - In some older and low end CPUs, a special data register, known as the accumulator, is used implicitly for many operations.

- **Address registers:** hold addresses and are used by instructions that indirectly access memory.

- **Conditional registers:** hold truth values often used to determine whether some instruction should or should not be executed.

- **General purpose registers (GPRs)** can store both data and addresses, i.e., they are combined Data/Address registers.

http://en.wikipedia.org/wiki/Processor_register
Main Types of Registers (cont’d)

- **Floating point registers** (FPRs) store floating point numbers in many architectures.
- **Constant registers** hold read-only values such as zero, one, or pi.
- **Special purpose registers** (SPR) hold program state; they usually include the program counter (aka instruction pointer), stack pointer, and status register (aka processor status word). In embedded microprocessors, they can also correspond to specialized hardware elements.
- **Instruction registers** store the instruction currently being executed.

http://en.wikipedia.org/wiki/Processor_register
Some Architectures &
The number of registers on them

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Integer registers</th>
<th>FP registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>x86-64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>IBM/360</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Z/Architecture</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Itanium</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>UltraSPARC</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>IBM POWER</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Alpha</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>6502</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>PIC microcontroller</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AVR microcontroller</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>ARM</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

http://en.wikipedia.org/wiki/Processor_register
Input/Output and Mass Storage

- Communication with outside world and external data storage
  - **Input unit**
    - “Receiving” section of computer
    - Obtains data from input devices
      - Usually a keyboard, mouse, disk or scanner
    - Places data at disposal of other units
  - **Output unit**
    - “Shipping” section of computer
    - Puts processed info on various output devices
      - Screens, paper printouts, speakers
    - Makes info available outside the computer
Input/Output and Mass Storage

- **Human interfaces**: monitor, keyboard, mouse
- **Archival storage**: not dependent on constant power
- External devices vary tremendously from each other
Input/Output and Mass Storage (continued)

- **Volatile storage**
  - Information disappears when the power is turned off
  - **Example**: RAM

- **Nonvolatile storage**
  - Information does not disappear when the power is turned off
  - **Example**: mass storage devices such as disks and tapes
Input/Output and Mass Storage (continued)

- **Mass (secondary) storage devices**
  - Direct access storage device
    - Hard drive, CD-ROM, DVD, etc.
    - Uses its own addressing scheme to access data
  - Sequential access storage device
    - Tape drive, etc.
    - Stores data sequentially
    - Used for backup storage *these days*
Mass or Secondary storage unit

- Long-term, high-capacity “warehouse”
- Stores programs or data not currently being used by other units on secondary storage devices (like discs)
- Takes longer to access than primary memory
- Data location
Direct access storage devices
- Data stored on a spinning disk
- Disk divided into concentric rings (tracks)
- Read/write head moves from one ring to another while disk spins
- Access time depends on:
  - Time to move head to correct sector
  - Time for sector to spin to data location
- We will come back to rotational storage media!
The IBM 350

• 1956—the first hard disk drive
• 4.4 MB of storage
• Weighed over a ton
• About 1,000 systems built

Image from www.ichigo.se/noterat/wp-content;
information from http://en.wikipedia.org/wiki/RAMAC

I/O controller

- Intermediary between central processor and I/O devices
- Processor sends request and data, then goes on with its work
- I/O controller interrupts processor when request is complete
Organization of an I/O Controller
Putting All the Pieces Together—the Von Neumann Architecture

- Subsystems connected by a bus
  - **Bus**: wires that permit data transfer among them

- At this level, ignore the details of circuits that perform these tasks: Abstraction!

- Computer repeats fetch-decode-execute cycle indefinitely
The Organization of a Von Neumann Computer

Wrong! These are on the CPU
Von Neumann Architecture

**Pros:**
- Simplifies hardware (both circuit-design and layout) - routing multiple data-buses can be awkward on compact layouts.
- Easier to generate re-locatable code, which makes multi-tasking easier to implement. Perhaps not an issue here, but consider a complicated product such as a digital set-top-box...

**Cons:**
- Instructions must be multiples of the data bus-width - can be inefficient.
- Variable number of cycles required for instructions. For example, an instruction that requires data from memory must wait at least another cycle before it can complete, whereas some instructions execute much faster. This can be a problem for time-critical applications.

http://www.mhennessy.f9.co.uk/pic/architecture.htm

---

S. Kalkan & G. Ucoluk & A. Cosar - CEng 111

METU Computer Engineering

METU Computer Engineering
Harvard Architecture

**Pros:**
- Data and address busses can be of different widths. This means the program memory word can be wide enough to incorporate an instruction and a literal (fixed data) in a single instruction.
- A built-in two-stage pipeline overlaps fetch and execution of instructions, meaning most instructions execute in a single clock cycle.

**Cons:**
- Slightly more confusing at first.
- Hardware is more complicated.

http://www.mhennessy.f9.co.uk/pic/architecture.htm