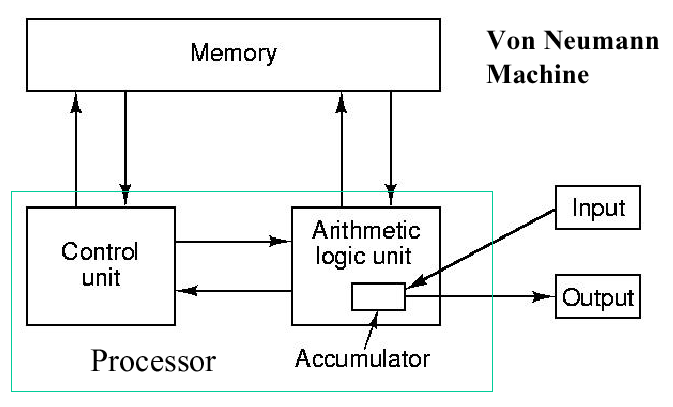
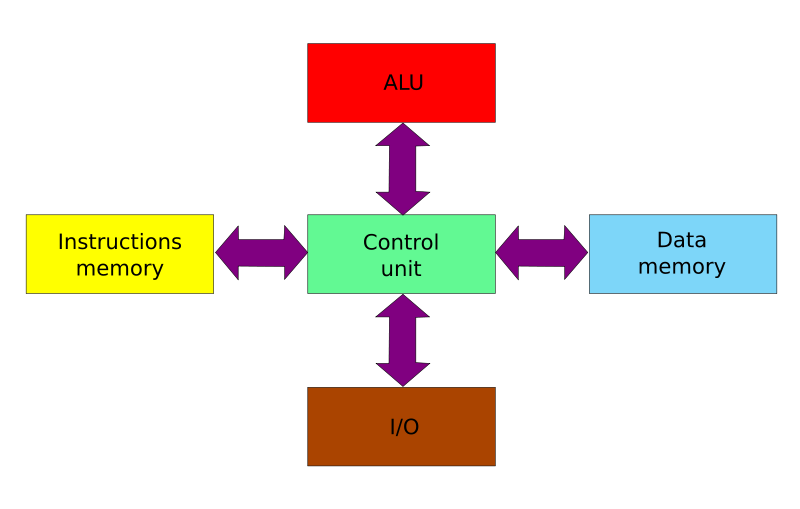
# Exam Review

## Computer Internals

* Major components of the internal structure of computers include:
  + CPU, Memory, BUS, I/O devices, storage, NIC (network interface cards)
* Programs are stored as data: written in high level languages, converted into assembler and then machine code that the PC can understand
  + Assemblers are machine specific, and compiler independent
* What makes up the CPU?
  + The CPU (Central Processing Unit) consists of registers, a program counter, ALU (Arithmetic Logic Unit), and a control unit.
* What’s the deal with memory?
  + Random Access Memory 🡪 any part can be accessed in constant time.
  + Addresses 🡪 stored in either hex, or octal, points to a specific piece of memory
  + Volatile memory goes away if the machine is turned off.
  + Memory speed is not up to par compared to the speed at which a CPU register can read/write, but it is less expensive.
* What does the BUS do?
  + It connects stuff: transfers information from one place to another.
* I/O Devices: what do they do?
  + Connect to the user in forms such as keyboards and mice
  + Non-volatile: the memory is recoverable (external harddrive, etc)
  + Graphics Cards
* What does the ALU do?
  + Gets instructions to do things on floating, bitwise, set point, or comparison operations, and gives the results in a condition code.
  + Controls flow instructions, character operations, interrupts handling
* Von Neumann Architecture
  + Von Neumann Bottleneck: only one operation can be done at time!



* Fetch and Execute Cycle
  + Step 1: Fetch 🡪 go to a place in memory, get the instruction there.
  + Step 2: Decode 🡪 what is the instruction telling me to do?
  + Step 3: Execute 🡪 do the instruction, put values where they belong.
  + Step 4: Repeat
* Operating Systems
  + The job of an OS is to ensure that every program runs on the hardware.
* Harvard Architecture
  + Better performance than Von Neumann, contains multiple BUSes, separate instruction and data memory

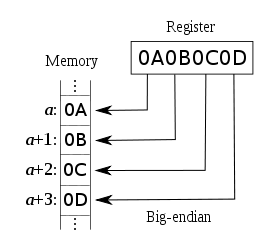


## Number Conversion w/Radix Points

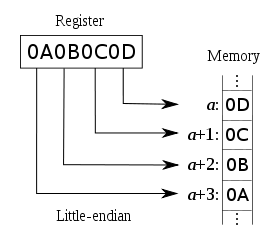
* There are several ways to convert one base to another base.
* 1st Method: Expansion
  + To convert a number, such as binary number, to decimal, just use the definition of a number representation as an abbreviated polynomial.
    - Example: 10101.1 = 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^-1
      * 16 + 0 + 4 + 0 + 1 + .5 = 21.5
  + This technique is easy to use when converting TO decimal, since our brains think in base 10.
* 2nd Method: Multiplication/Division Method
  + **Division method**
    - To convert a number from one base to another, take the number, divide it by the target base, keep the remainder and divide the quotient by the base until the quotient that remains is zero. Number generated from right🡪 left
    - Example: 1310 🡪11012
      * 13/2 = 6 remainder **1**
      * 6 / 2 = 3 remainder **0**
      * 3 / 2 = 1 remainder **1**
      * 1 / 2 = 0 remainder **1**
  + **Multiplication Method**
    - This method is generally used to convert fractions
    - Multiply the number by the base, and keep the digit that is generated, then multiply the thing to the right of the decimal by the base again, until you have nothing to the right of the decimal.
    - Example: .7812510 🡪 .110012
      * .78125 x 2 = 1.56250
      * .56250 x 2 = 1.1250
      * .1250 x 2 = 0.250
      * .250 x 2 = 0.50
      * .50 x 2 = 1.0

## Endian-ness

* There are generally two forms of Endian-ness, Big Endian and Little Endian.
  + Mixed-endian does exist, but we don’t deal with it so it doesn’t matter.
* Endian-ness refers to the ordering of bytes stored in memory, determined by the significance of the byte.
* What’s the difference between the two different types?
  + **Big Endian**
    - Big Endian machines store the MOST significant byte first, at the LOWEST byte address
    - Most machines that use Big Endian do not run an architecture that we are likely to come in contact with.
      * Some examples include Motorola 6800, IBM Power and SPARC



* + **Little Endian**
    - Little Endian machines store the LEAST significant byte first, at the LOWEST byte address
    - Most machines that we use in our day-to-day life is Little Endian.
      * Examples include x86, x86-64, Z80, etc.



## Helpful C Tips/Refreshers

* Pointers (PASS BY REFERENCE)are used to get the address of other variables in memory
  + & 🡪 Gives you the address of whatever follows it.
  + \* 🡪 de-references a pointer and gives you the value it points to.
    - Quick Exercise
      * int \* p; 🡪 pointer to an integer, p
      * int x; 🡪 an integer, x
      * p = &x; 🡪 p now points to x’s address
      * \*p = 3; 🡪 de-references p, and set’s x to 3
      * P = 0; 🡪 null pointer
  + Remember that pointers can be used to represent everything from arrays to strings, and can have a pointer of ANY type
    - Strings MUST end with a terminating character, such as \0
* , 🡪 separates multiple expressions: the value is the last thing entered.
* 🡪 (arrow) gets the value inside of a struct.

## Gray Code

* This stuff is actually super simple, although it may seem complicated at first.
* Remember to only change one digit at a time.
* Easiest way to explain how to do it: when given an n-bit gray code, and attempting to get n + m –bit code, you take the n-bit code, “flip” it over an imaginary horizontal line, and then add 0’s for a new column in the top half, and 1’s for the new column on the bottom half.
  + Example: want to go from 2-bit gray code to 3-bit gray code
  + 2-bit
    - 0 0  
      0 1  
      1 1  
      1 0
  + 3-bit
    - 0| 0 0  
      0| 0 1  
      0| 1 1  
      0| 1 0  
      ------  
      1| 1 0  
      1| 1 1  
      1| 0 1  
      1| 0 0