# Final Review

## Chapter 1

1. Convert from 1 base to any other base
2. What is used to represent everything inside the computer?
3. Why do we like using hex numbers to represent bits?
4. How space is made for multidimensional arrays
   1. Array of arrays
   2. One big chunk
   3. How does C decide which to use?
   4. Be able to write expressions that access elements of an array without using [ ] operator
      1. If ar was declared as int ar[3][4][7] how would you access element ar[i][j][k]
      2. Repeat the above but for ar declared as int\*\*\*. (Assume sufficient space has been already allocated for it)
      3. Be able to write the assembly code for these accesses as well
5. What does it mean if a machine is little endian?
   1. Big endian?
   2. Which is Intel?
   3. If the word size of the machine and the addressability of a machine are the same are there any endinaness issues? For example the size of a word is a byte and memory is byte addressable
   4. What would the value in eax be after the performing movl 100, %eax
      1. Memory looks like

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Addr | 100 | 101 | 102 | 103 | 104 | 105 |
| Value | 0x11 | 0x22 | 0x33 | 0x44 | 0x55 | 0x66 |

* + 1. If the machine is little Endian?
    2. Big Endinan?

1. What does 111 1010 represent?
2. Be able to find the unsigned, 2’s complement, sign magnitude, and floating point representation of any number
   1. Be able to find the decimal value of any bit string when interpreted as one of the above
3. Be able to write **valid** C code that uses bitwise operators to extract fields, set bits, clear bits or examine bits of a bit string
4. Are there types at the machine level?
5. If you have B bits how many unique things can you represent?
6. If you have S unique states, what is the minimum number of bits you need to represent them all?

## Chapter 2

1. What are the two major components of the computer?
2. Give at least 2 reasons why code compiled on one computer might not run on another computer
3. What is the difference between RAM and ROM?
   1. Why is it that turning off your computer and then turning it back on is likely to fix a problem you have with your computer?
4. What is the bus?
   1. Control Bus?
   2. Address Bus?
   3. Data Bus?
5. What do the following components inside the CPU do
   1. PC?
   2. IR?
   3. Data registers?
   4. MAR?
   5. MDR?
   6. ALU?
   7. ESP?
   8. FLAGS?
6. What are the steps in the CPU cycle?

## Chapter 3 & 4

1. Be able to convert any C code into assembly code.
2. What is the difference between jae and jge?
   1. How do you determine which you should use?
      1. For example if we had if(i > 7), which would you use?
      2. Do you need more information? If so what?
3. What determines which suffix (b, w, or l) you should use for an instruction when translating from C to assembly?
4. Be able to use the mul, imul, and div instructions
5. Be able to use the advanced indexing mode
   1. What is the difference between
      1. Movl 5(%eax, %ebx, 2),%ecx
      2. Leal 5(%eax, %ebx, 2), %ecx
6. Know which bytes of memory are accessed by an instruction. For example if eax = 10 and ebx = 5 which bytes of memory are accessed by
   1. Movl (%eax), %ecx
   2. Movw (%eax), %cx
   3. Movb (%eax), %cl
   4. Movl (%eax,%ebx, 4), %ecx
   5. Movw (%eax, %ebx, 2), %cx
   6. Movb (%eax, %ebx, 1), %cl
7. Know what each of the following preprocessor directives do
   1. .global
   2. .byte
   3. .word
   4. .long
   5. .space
   6. .rept
   7. .equ

## Chapter 7

1. What is the stack?
2. What is the current stack frame?
   1. What does it mean that the stack frames are chained?
      1. How is chaining accomplished?
3. What is the purpose of the prologue?
   1. The epilogue?
4. What are the gcc C calling conventions?
5. Where is space for variables made?
   1. Global?
   2. Static?
   3. Local?

## Chapter 8

1. How does I/O mapped I/O work?
   1. Memory Mapped?
2. How does polling (wait loop) I/O work?
   1. What are its advantages? Disadvantages?
3. How does interrupt driven I/O work?
   1. What are its advantages? Disadvantages?
   2. Explain in detail how an interrupt is serviced once the interrupt assert line is set to 1.

## Chapter 9

1. What are the 4 major services provided by the operating system?
2. What does it mean for a program to be running?
3. Assuming only a single CPU with a single core, if your program is running is the Operating System as well?
4. Can the OS stop your program from running if your program is currently running?
5. Explain how the program a.out gets loaded after your type ./a.out on the terminal
6. If the OS is the program that loads programs, how does the OS get loaded?
7. What is time sharing?
   1. On what hardware components does it depend on?
8. What is a context switch?
   1. Explain the steps taken in performing a context switch
9. How can control of the CPU switch between programs?
10. What are system calls?

## Chapter 11

1. How are system calls implemented?
   1. If we had a system call named bob that had call number 30 and we wanted to call bob as bob(20,8) how would we do it in assembly.

## Inline Assembly

1. Be able to translate C code into inline assembly
   1. What do each of the following constraint modifiers mean? When should you use them?
      1. +
      2. =
      3. &

## Debugging

1. Be able to do the following in GDB
   1. Set a breakpoint on a line number
   2. Print the value of a variable
   3. Print the value of a register
   4. Print the value of a value on the stack assuming the prologue has already been completed
   5. Print all the arguments on the stack as integers
   6. Print all the local variables for a function.
   7. Print the contents of an array
2. Print out the elements in an array if the pointer to an array is in the register
   1. If EAX contains the pointer to an array of integers named ar, print out the first 10 elements of ar.
3. Print out the elements in an array if the pointer to the array is on the stack and the prologue has been run
   1. Print the first 5 elements of ar in: foo(int a, int b, int\* ar)
   2. Print the first 5 elements of ar in: foo(int a, int b, char\* ar)
   3. Print the first len elements of ar in: foo(int a, int\* ar, int len)
4. Print out a row of elements of a 2D array
   1. int\*\* ar is stored in esi. Print out the first 4 elements of row at index 7
   2. foo(int\*\* ar, int num\_rows, int num\_cols). Print out the first 10 elements of ar at row index 2
   3. foo(int\*\* ar, int num\_rows, int num\_cols). Print out num\_cols elements of ar at row index 5