# Assembly: Chapters 3, 4, 7

1. Be able to translate C code into assembly that is callable from C.
   1. Be able to translate if, for, while, and function calls
   2. Be able to translate expressions involving multidimensional arrays declared either statically or dynamically
   3. Be able to use the advanced indexing mode to access arrays
      1. 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
            1. **Bytes 10 - 13**
         2. Movw (%eax), %cx
            1. **Bytes 10 - 11**
         3. Movb (%eax), %cl
            1. **Byte 10**
         4. Movl (%eax,%ebx, 4), %ecx
            1. **Bytes 30 - 33**
         5. Movw (%eax, %ebx, 2), %cx
            1. **Bytes 20 - 21**
         6. Movb (%eax, %ebx, 1), %cl
            1. **Byte 15**
   4. How does the typing of variables affect the assembly code you write? For example translate the following lines of C
      1. **It affects the suffix of the instructions, the registers, and the scale in the advanced indexing mode that you use**
      2. Int A = 7; //assume A is in EAX
         1. **Movl $7, %eax**
      3. Short B = 7; //assume B is in EBX
         1. **Movw $7, %bx**
   5. Be able to use the leal instruction.
   6. Be able to translate recursive functions
2. What are the gcc C calling conventions? What happens if you break them?
   1. **Arguments are passed through the stack. They are pushed in reverse order of declaration**
   2. **EAX, ECX, and EDX will not have live values when function is called**
   3. **The return value, if there is one, will be placed in EAX**
3. What is defined to be the stack?
   1. **Wherever ESP points and all addresses higher than that**
4. What is defined to be the current stack frame?
   1. **All values in memory between ESP and EBP**
5. What does it mean for the stack frames to be chained? How do they become chained?
   1. **That from the current stack frame you can reach the previous stack frames. They become chained from the prolog. Namely the push %ebp. Movl, %esp, %ebp part**
6. What is the code for the prologue? What is its purpose?
   1. Push %ebp
   2. Movl %esp, %ebp
   3. Subl space for locals
   4. Save callee saved regs we plan on using
   5. Purpose is to establish this functions stack frame
7. What is the code for the epilogue? What is its purpose?
   1. Pop saved regs
   2. Movl %ebp, %esp
   3. Pop %ebp
   4. Ret
   5. To undo the prologue
8. Know what push, pop, call, and ret do.
   1. **Push adds a value to the top of the stack**
   2. **Pop remove the top element from the stack and place in destination**
   3. **Call: save return address on stack and then execute function**
   4. **Ret: pop the top of the stack into eip**
9. Where is space for local variables made?
   1. **On the stack**
   2. Global?
      1. **In the .data section**
   3. Static?
      1. **In the .data section**
      2. What prevents you from accessing a static variable outside of the function it is declared in?
         1. **The compiler**

# Debugging

Be able to do following

1. Print out a variable
   1. **P varName**
2. Print out a register
   1. **P $regName**
3. Put out the value of an argument on the stack after the prologue has run
   1. **P ((int\*)$ebp)[2 + argLocation]**
4. Print out the value of a local variable after the prologue has run
   1. **P ((int\*)$ebp)[local location]**
5. Print out the elements in an array if the pointer to an array is in the register
   1. If EAX contains the pointer to an array of integers named ar, print out the first 10 elements of ar.
      1. **P ((int\*)$eax)[0]@10**
6. 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)
      1. **P ((int\*\*)$ebp)[4][0]@5**
   2. Print the first 5 elements of ar in: foo(int a, int b, char\* ar)
      1. **P ((char\*\*)$ebp)[4][0]@5**
   3. Print the first len elements of ar in: foo(int a, int\* ar, int len)
      1. **P ((int\*\*)$ebp)[3][0]@((int\*\*)$ebp)[4]**
7. 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
      1. **P ((int\*\*)$esi)[7][0]@4**
   2. foo(int\*\* ar, int num\_rows, int num\_cols). Print out the first 10 elements of ar at row index 2
      1. **P ((int\*\*\*)$ebp)[2][2][0]@10**
   3. foo(int\*\* ar, int num\_rows, int num\_cols). Print out num\_cols elements of ar at row index 5
      1. **P ((int\*\*\*)$ebp)[2][5][0]@P ((int\*)$ebp)[4]**
8. Set a break point on a line
   1. **B lineNumber**
9. Go to the next break point
   1. **continue**
10. Go to the next line of code skipping over function calls
    1. **next**
11. Go to the next line of code going inside of function calls
    1. **step**