10/12/16 Assembly

Class notes

Coding division and multiplication

Register efficiency

* If you need to use a register that already holds a value, it is possible to create a local temp variable on the stack to copy values
  + C++ ex) int x,y,temp;

x = 5;

y = 6;

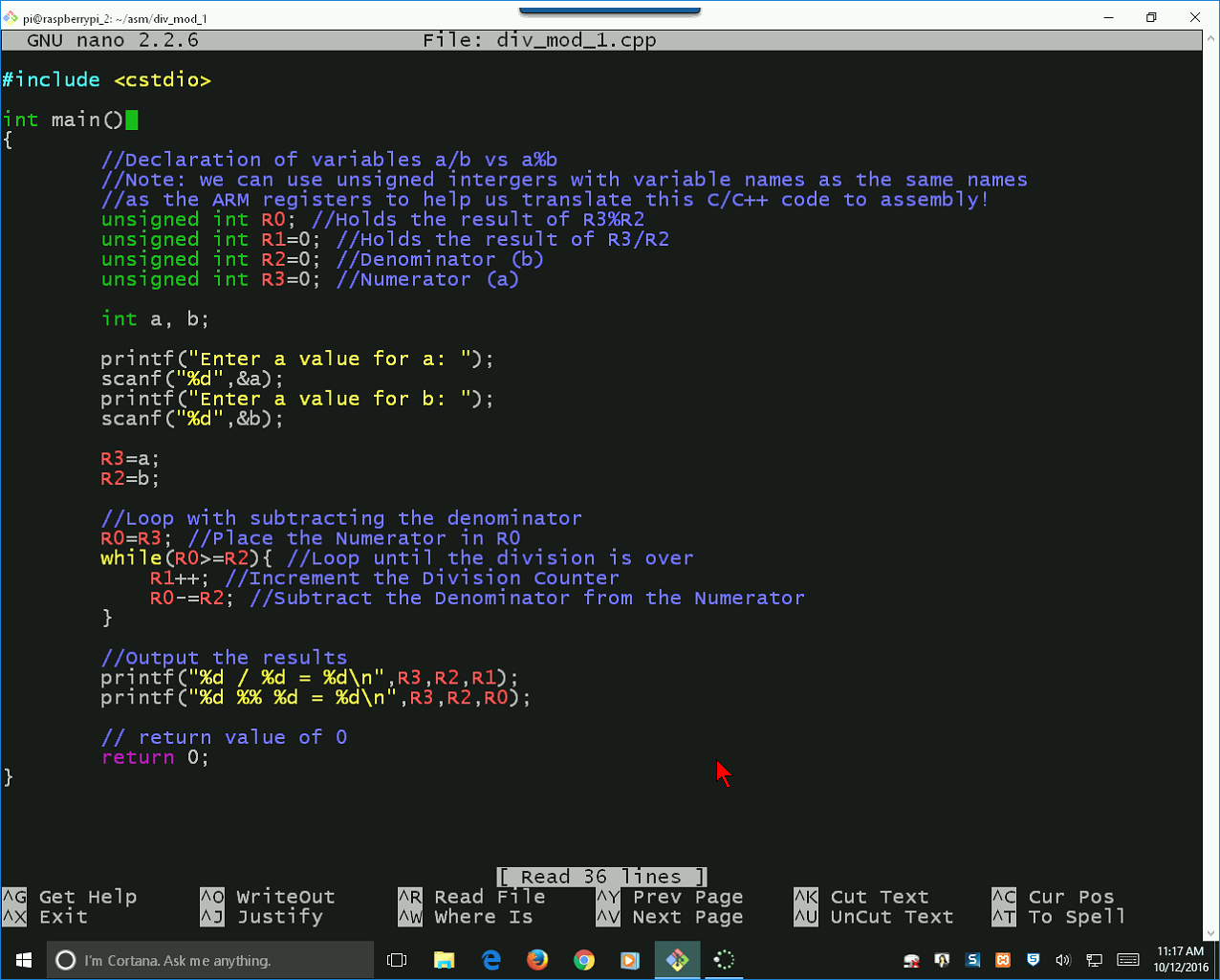
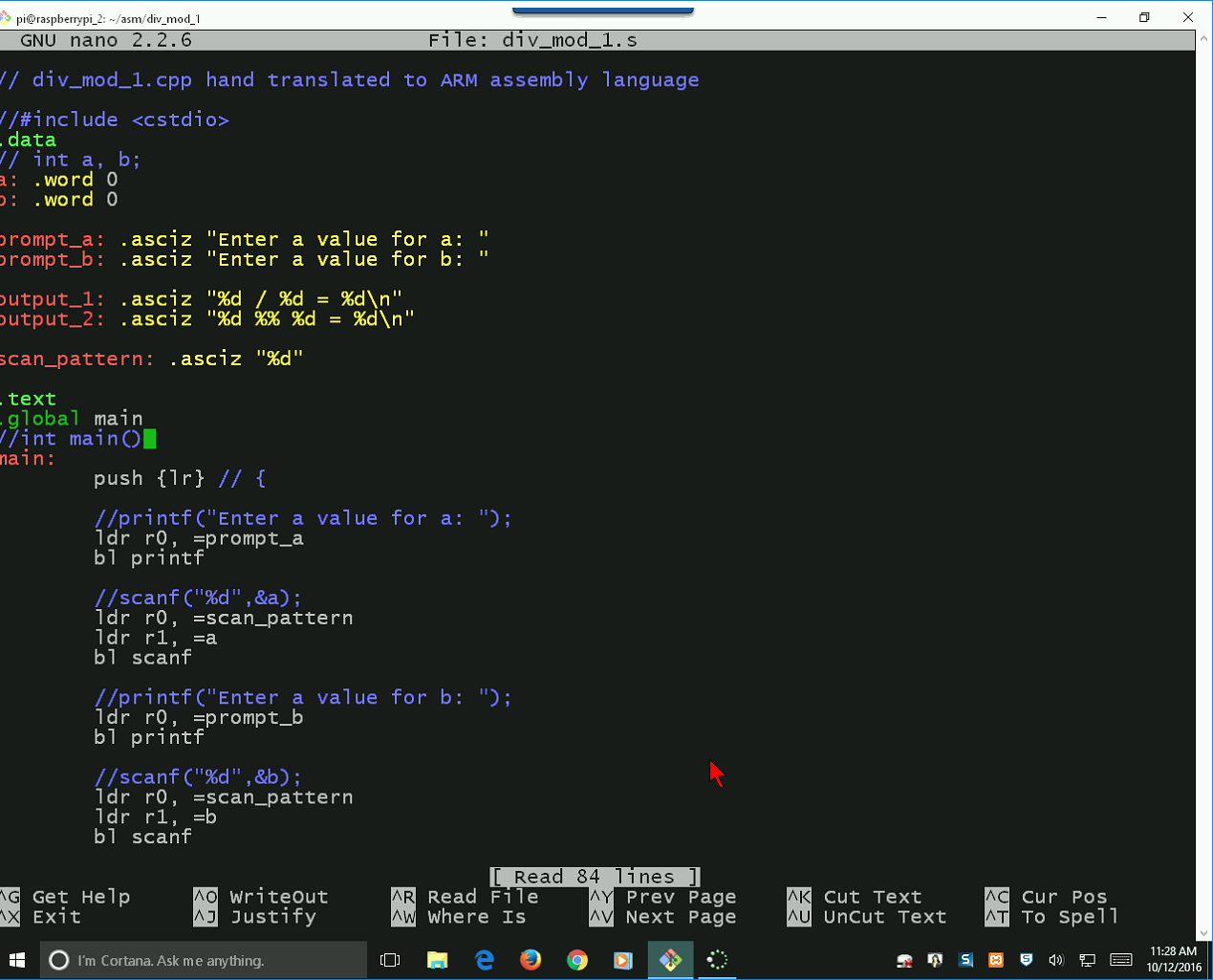
temp = x;

x = y;

y = temp;

Multiplication

Division

* Assembly does not have a division keyword
* To divide, call a loop to subtract the value and count how many times it subtracts
* C++ division example:
* Assembly ex part 1)

Printf

* Requires r0 to contain the format string
* If needed, r1-r3 contain the values for the format string
* Asm syntax) ldr r0, =string //Load string to r0

bl printf //Print string

Scanf

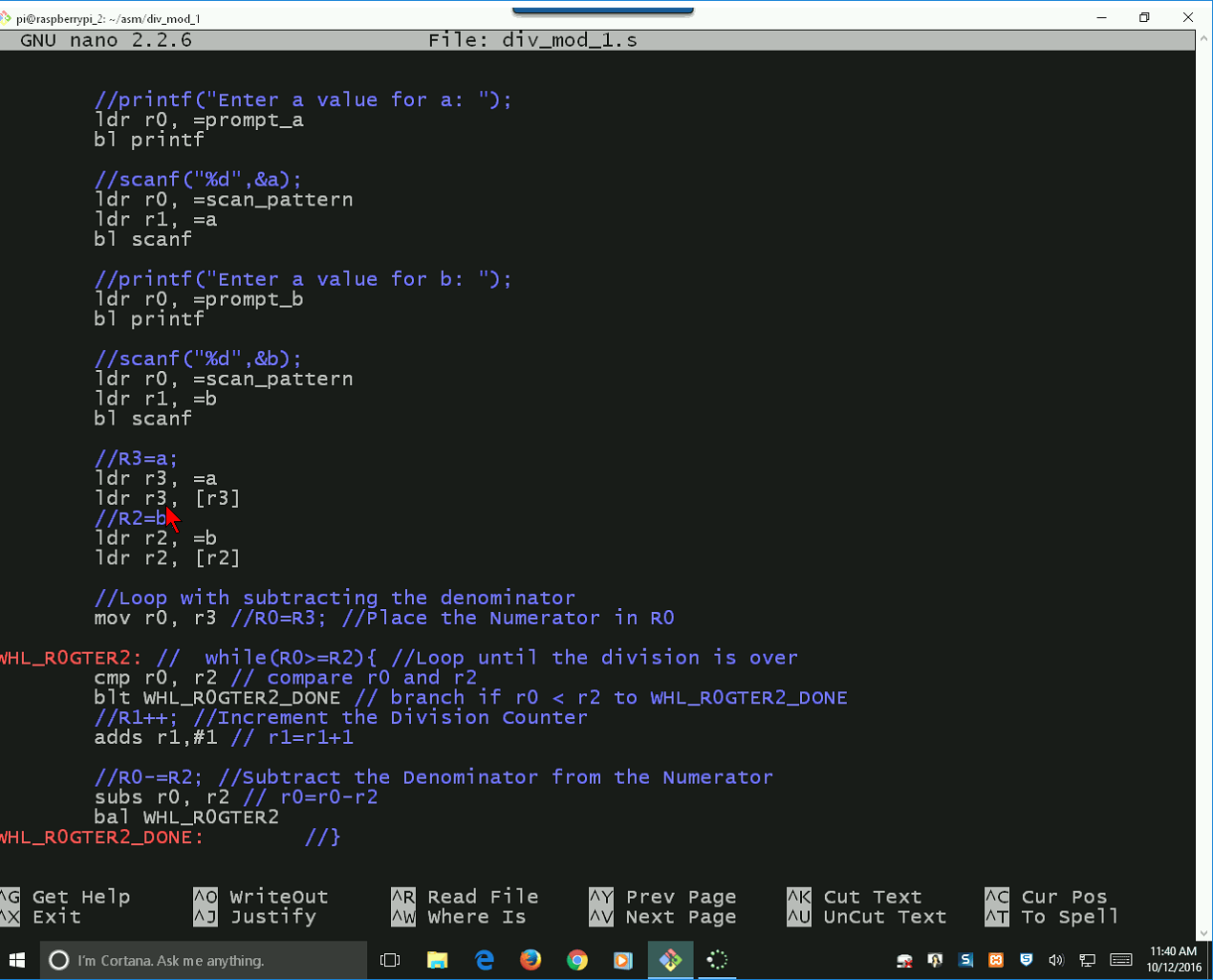
* C++ syntax) scanf(“%d”, value);
* Asm syntax) ldr r0, =scan\_pattern //Load %d format

ldr r1, =value //Load value

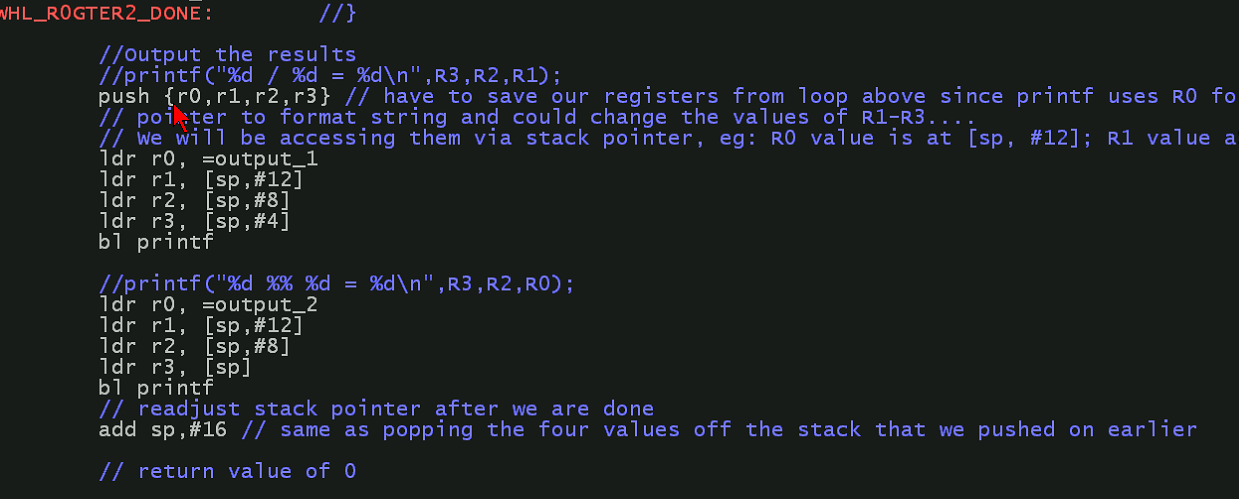
bl scanf //call for scanf

Functions

* R0-r4 contains the 1st,2nd,3rd, and 4th params for the function
* Anything onward gets pushed to the stack

Asm ex pt2)

* Green box
  + ldr r3, =a //Loads the address of a and b into the 2nd and 3rd registers
  + ldr r3, [r3] //Dereferences register to store the value into their respective registers
* Orange box
  + cmp r0, r2 //Compares the values in r0 and r2 and sets flags
  + blt <End\_labelname> //If r0 < r2, jumps to end label
    - def) Branch if less than
  + //Increment division counter
  + //Subtracts value from total
  + bal <Orangebox\_label> //Jumps to label start
    - def) Branch always

Asm ex part3)

* Push {value}
  + Def) Pushes value onto stack
  + Since we need to use the registers, we push the values contained in r0-r3 into the stack
  + Stack is used as the temporary value
* sp
  + Def) Stack pointer
  + Directs program to a value in memory
* Green box
  + Moves value from r1 to r3 and vice versa for utility
* Orange box
  + Moves the values around for output utility
* Each value is 4 bytes each
  + Remember: You must unload the stack from the top down
  + Stack:
    - sp – r0
    - +4 – r1
    - +8 – r2
    - +12 – r3
  + Values:
    - r3 – #12
    - r2 – #8
    - r1 – #4
    - r0 – sp
* add sp, #16
  + Adds 16 to returns the stack to its original pointer