Keiran Berry

CENG320L

Lab 7 Report

7 November, 2022

1. Lab Overview

The purpose of this lab was to learn integer mathematics in assembly and implement them through two different programs, one which finds prime numbers and one which counts saved change. For the prime program, we were to have an isprime function which returns 0 if the number is not prime and 1 if it is, and we were to have a divide function as well. Since the algorithm we learned during the lab period did not require a division function, I wrote my own version of the division through shifting and subtracting in a separate file, just so I at least got practice with writing one even if my prime program worked without one. For the change counting function, we were to implement multiplication as shift and add operations, and division by a constant. In both functions, I got to practice more with the printf and scanf functions as well.

For the prime program, I began with prompting the user to input a natural number. This number was taken in and every number from 1 to the number was checked by the isprime function. The isprime function worked through a nested loop, checking each possible combination up to p/2 for both i and j, to see if any combinations of the two integers multiplied to p. If it made it all the way through the loop without kicking out, it was prime, and if it kicked out of the loop it meant that factors were found and the number was composite. This was just repeated for all of the numbers up until n (the number which was input) and if a prime was found then it would output it to the terminal. One interesting thing with this would be memoization/dynamic programming. The program already runs really fast but it would be interesting to see it when all the prime numbers are tabled and it wouldn't have to check for factors hardly at all.

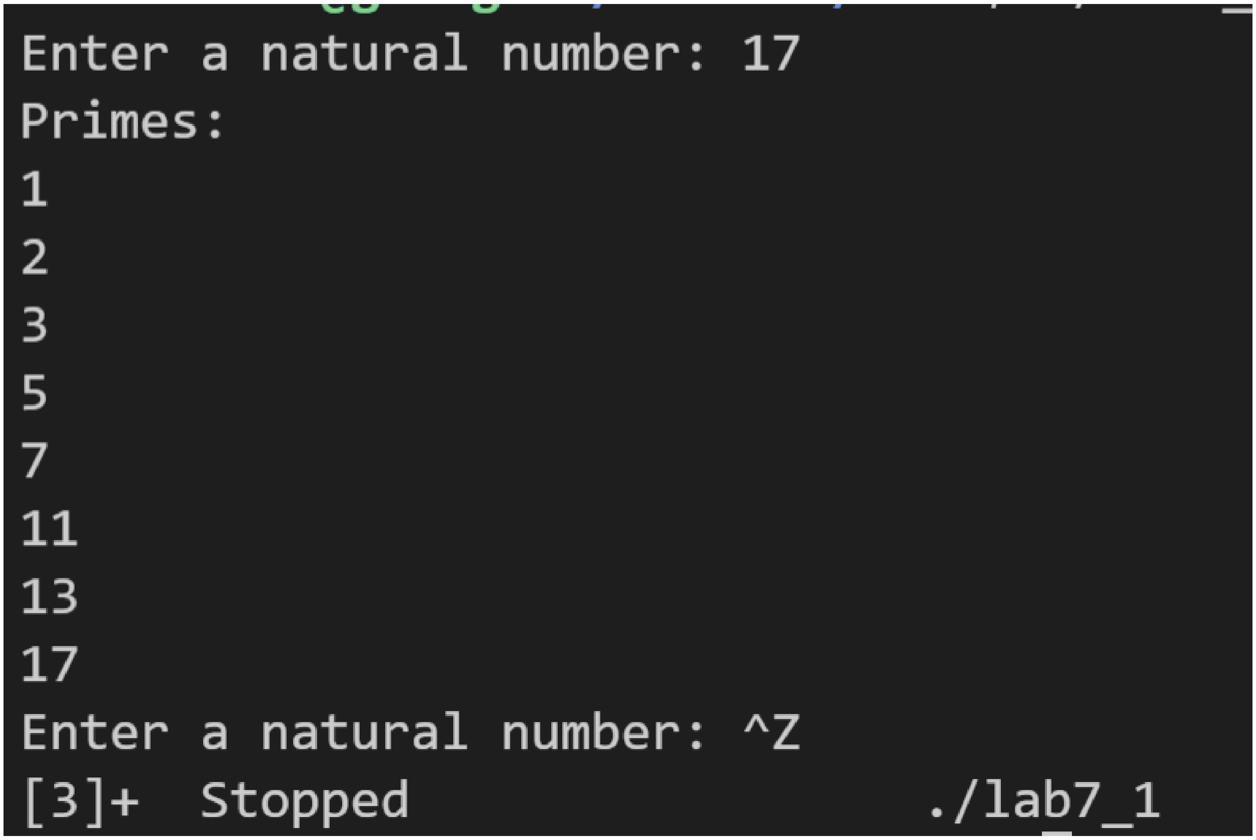
For the counting coins program, the user is prompted to input 4 weeks worth of pennies, nickels, dimes, and quarters. These come in one week at a time, and are read in with a format string of "%d %d %d %d" to get all 4 at once. This loops until all 4 weeks are read in, adding the coins to their own respective register each time. The coins are then summed up using multiplication (a nickel is 2^2 + 2^0, for example) so that each register contains the value of its coins rather than the number of coins. The division by a constant algorithm is then used in order to divide by 100 and get its remainder to get dollars and cents, output that as the total for the month, and the same algorithm is used with 400 to get the weekly average. Finally, the number of cents can be multiplied by 12 using shifts and adds to get the yearly average and then divided by 100 to get it in dollars and cents.

2. Bugs and Hurdles

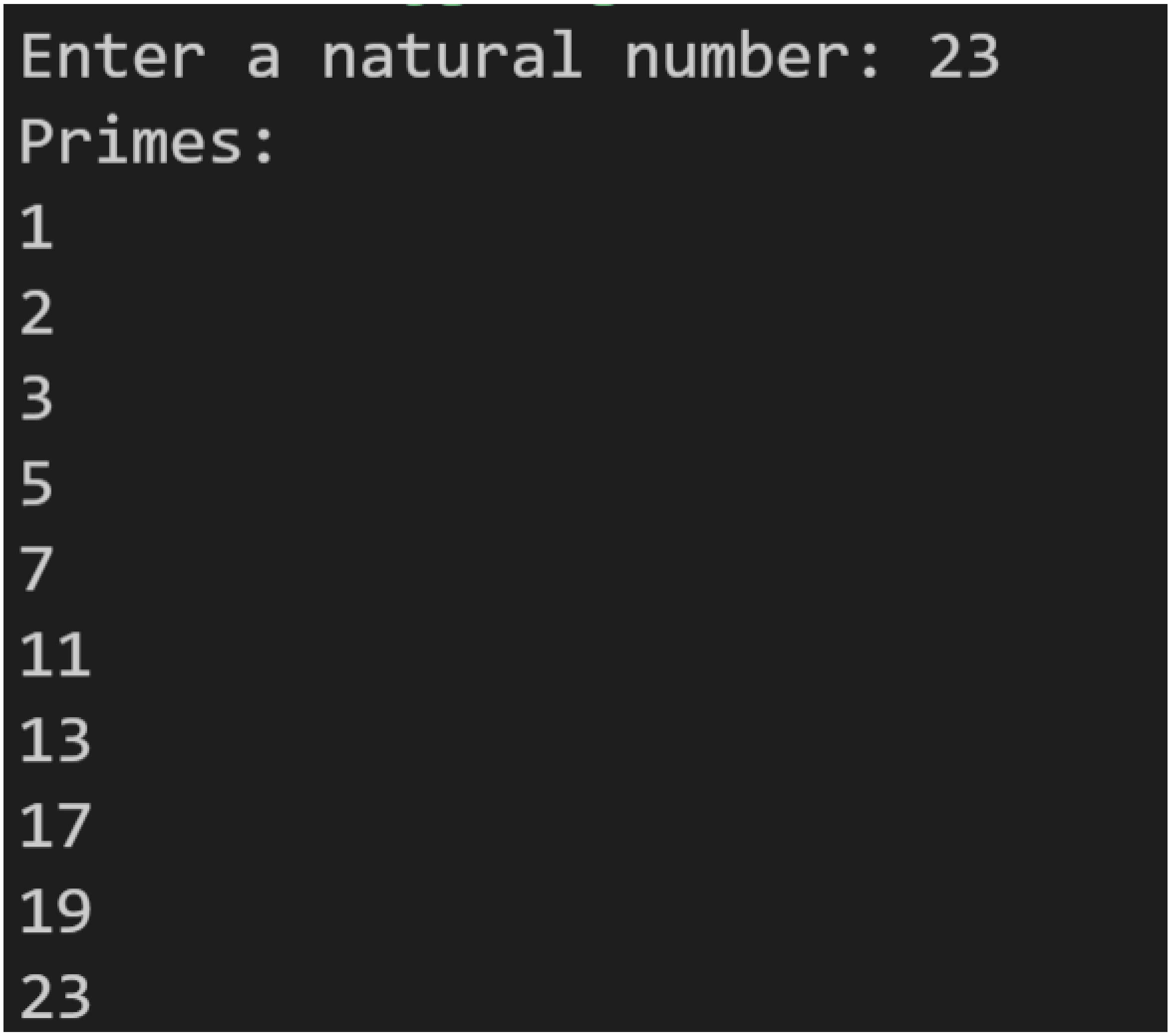
The first hurdle I encountered was some numbers not being considered composite when they should be. This was an easy fix, just changing the blt statements to ble in case the required i or j factors were exactly p/2. There was one core dump issue with my prime program, because I was storing and loading the registers within the isprime function. I ended up getting the complete correct output, then getting a core dump at the very end. I went on and finished the second part of the lab, and then came back to the prime function to try and fix the issue. I remedied the problem through storing/loading the registers in main so that they weren't constantly being messed with and their value could still be changed when needed in the function. This got the function to work fine once, then afterwards it asks for another input. This is odd to me because my printf for the prompt is right at the beginning of main, and it isn't in any sort of loop. It is easy, however, to just ctrl+z out of the program after getting your desired output once. If you try to use the secondary prompt, it will output the correct answer for you and then give the core dump that it gave before. The coin counting function went very smoothly in comparison to the former.

3. Results

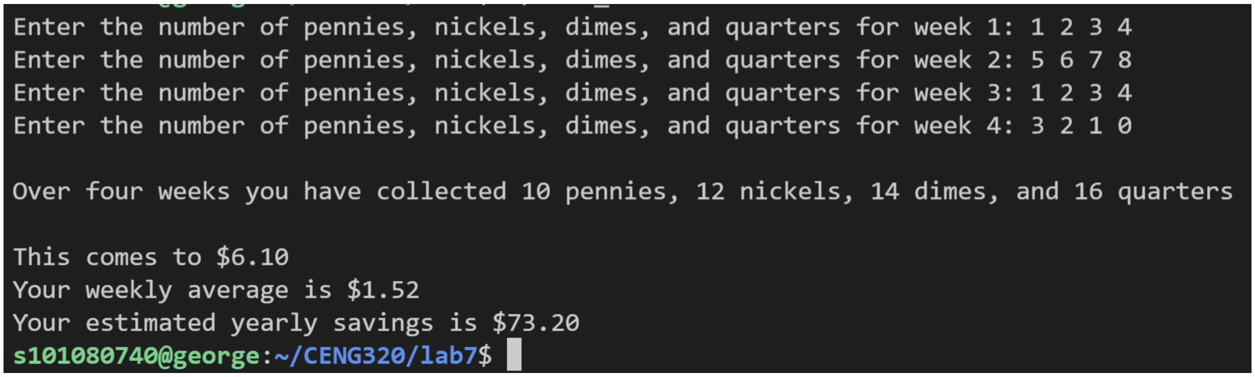
This lab taught me a lot more about how integer mathematics actually work in assembly rather than just doing them hypothetically on paper. I was also able to get more practice with printf and scanf, as well as gdb since nothing seemed to work on the first try. I got to learn more algorithms than the ones which were done on the homework, as well. In the end, I got through everything and both programs work, even if there is a bug in the first one. I was able to troubleshoot and fix it to the best of my ability, and the fact that I was able to implement the algorithm with the nested loops as quickly as I did proved to me how much the time I spent on Lab 5 helped me.



```
Enter a natural number: 17
Primes:
1
2
3
5
7
11
13
17
Enter a natural number: ^Z
[3]+  Stopped                 ./lab7_1
```

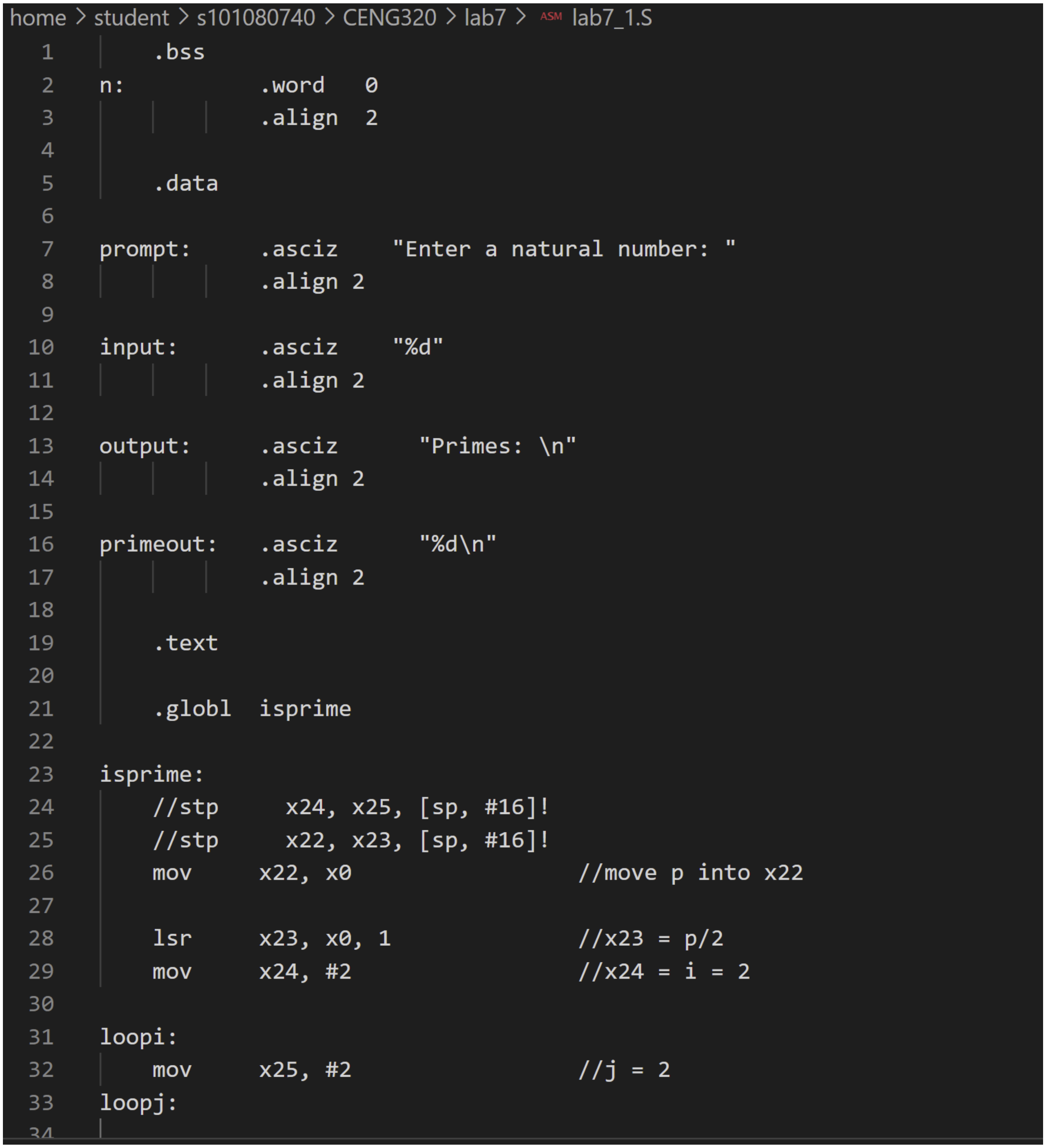


```
Enter a natural number: 23
Primes:
1
2
3
5
7
11
13
17
19
23
```



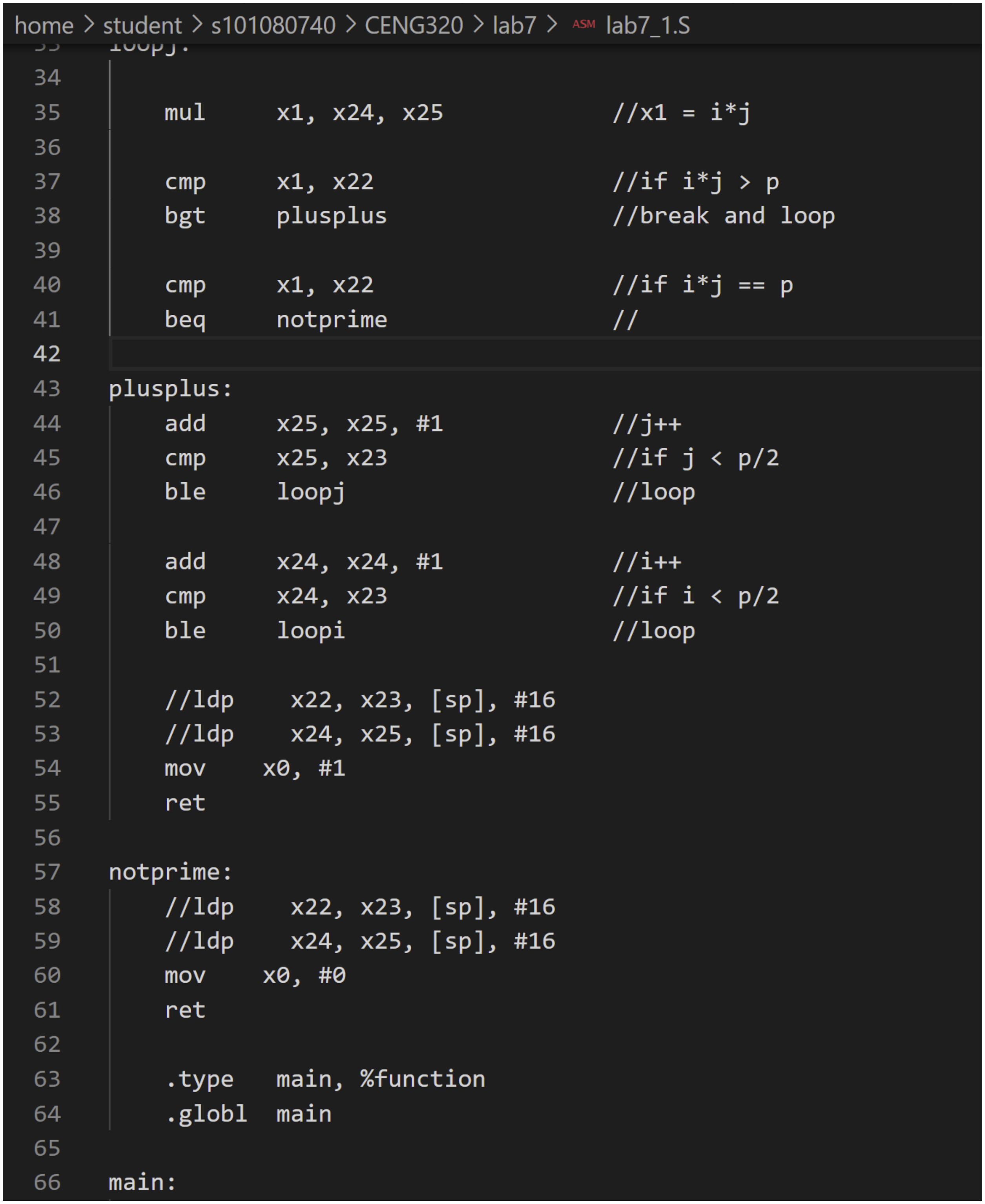
```
Enter the number of pennies, nickels, dimes, and quarters for week 1: 1 2 3 4
Enter the number of pennies, nickels, dimes, and quarters for week 2: 5 6 7 8
Enter the number of pennies, nickels, dimes, and quarters for week 3: 1 2 3 4
Enter the number of pennies, nickels, dimes, and quarters for week 4: 3 2 1 0

Over four weeks you have collected 10 pennies, 12 nickels, 14 dimes, and 16 quarters

This comes to $6.10
Your weekly average is $1.52
Your estimated yearly savings is $73.20
s101080740@george:~/CENG320/lab7$
```

Part 1

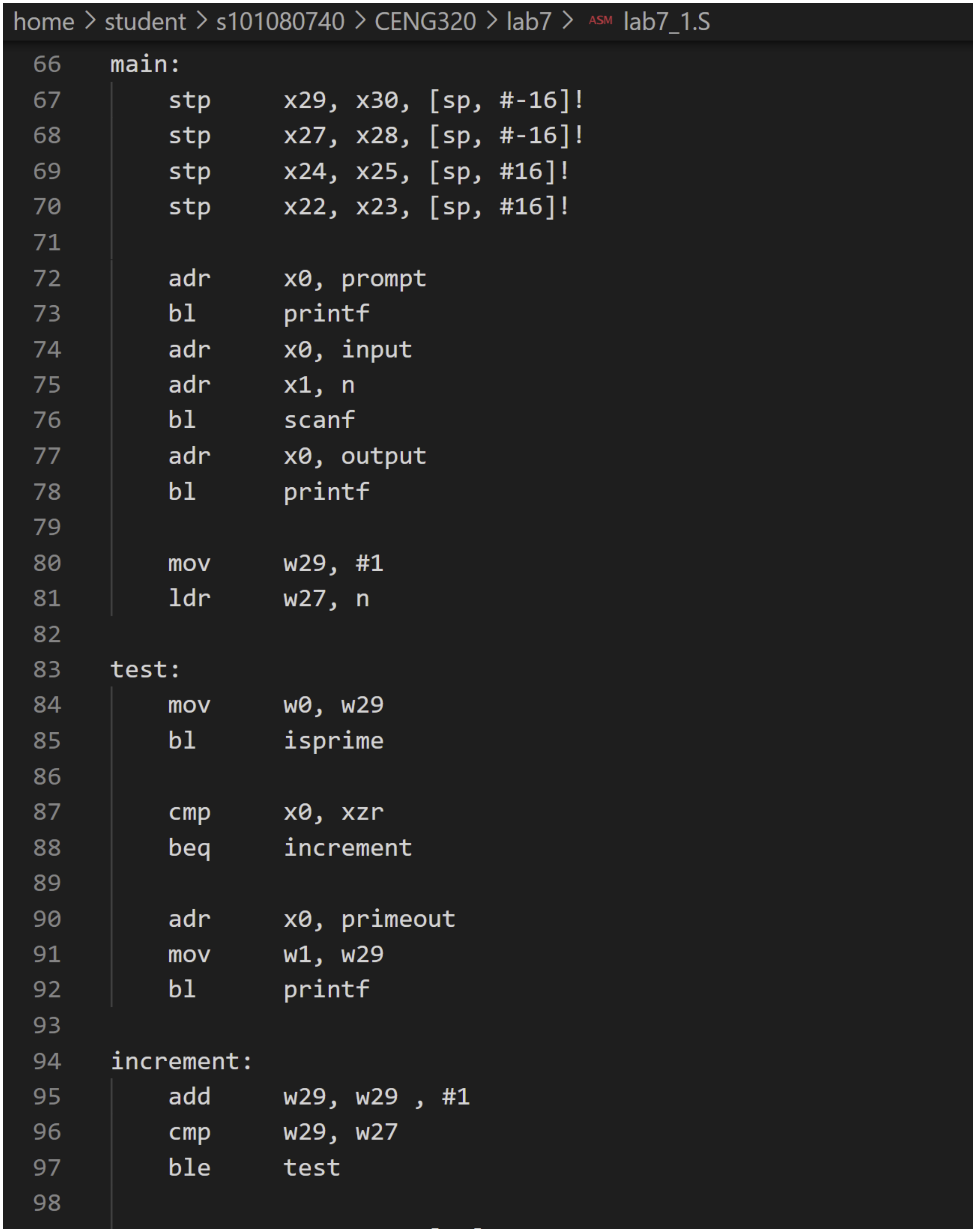


home > student > s101080740 > CENG320 > lab7 > lab7_1.S

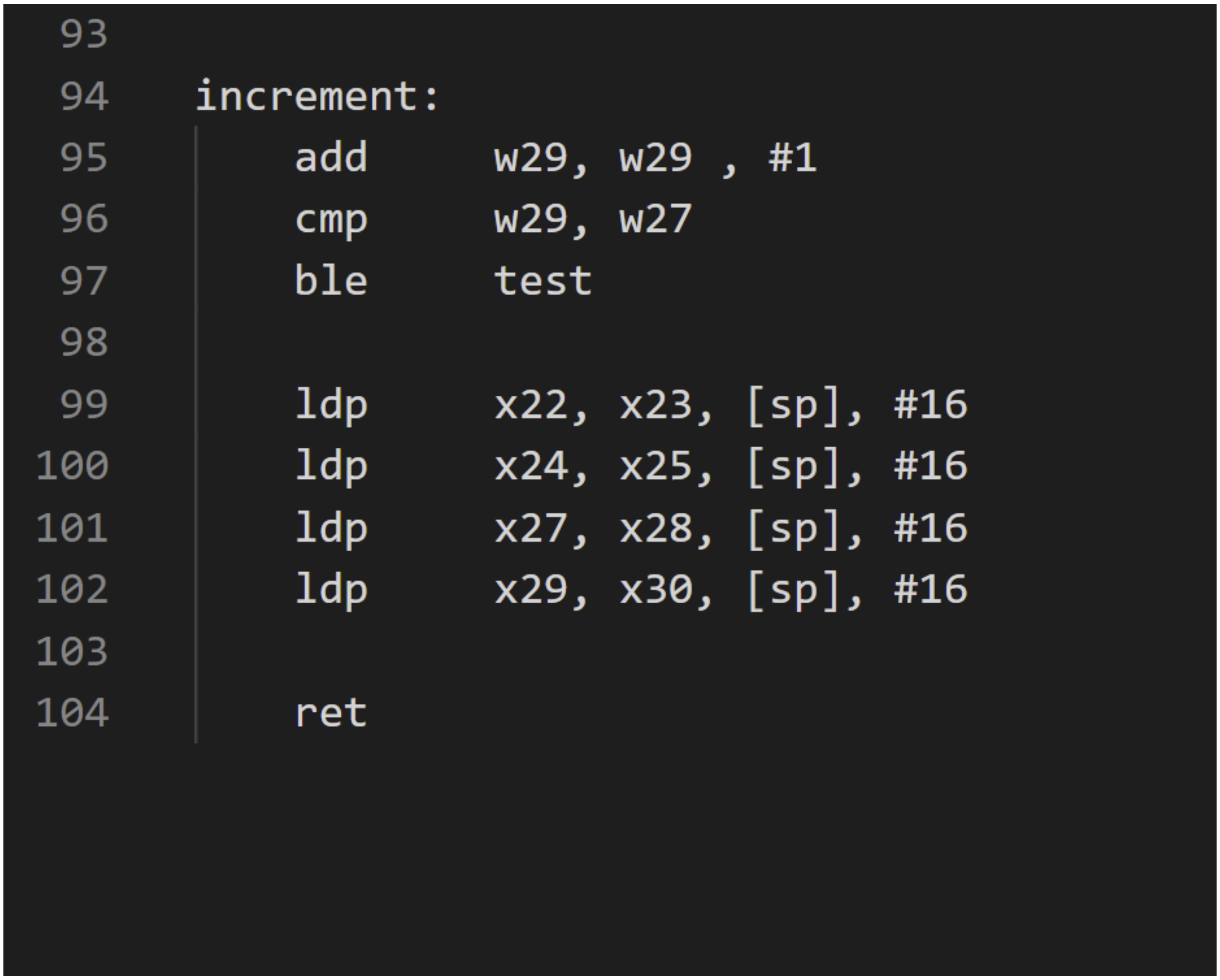
```
    .bss
n:          .word   0
            .align  2

    .data

prompt:     .asciz    "Enter a natural number: "
            .align 2

input:      .asciz    "%d"
            .align 2

output:     .asciz      "Primes: \n"
            .align 2

primeout:   .asciz      "%d\n"
            .align 2

    .text

    .globl  isprime

isprime:
    //stp     x24, x25, [sp, #16]!
    //stp     x22, x23, [sp, #16]!
    mov     x22, x0                 //move p into x22

    lsr     x23, x0, 1              //x23 = p/2
    mov     x24, #2                 //x24 = i = 2

loopi:
    mov     x25, #2                 //j = 2
loopj:
```



```
loopj:

    mul     x1, x24, x25            //x1 = i*j

    cmp     x1, x22                 //if i*j > p
    bgt     plusplus                //break and loop

    cmp     x1, x22                 //if i*j == p
    beq     notprime                //

plusplus:
    add     x25, x25, #1            //j++
    cmp     x25, x23                //if j < p/2
    ble     loopj                   //loop

    add     x24, x24, #1            //i++
    cmp     x24, x23                //if i < p/2
    ble     loopi                   //loop

    //ldp    x22, x23, [sp], #16
    //ldp    x24, x25, [sp], #16
    mov    x0, #1
    ret

notprime:
    //ldp    x22, x23, [sp], #16
    //ldp    x24, x25, [sp], #16
    mov    x0, #0
    ret

    .type   main, %function
    .globl  main

main:
```

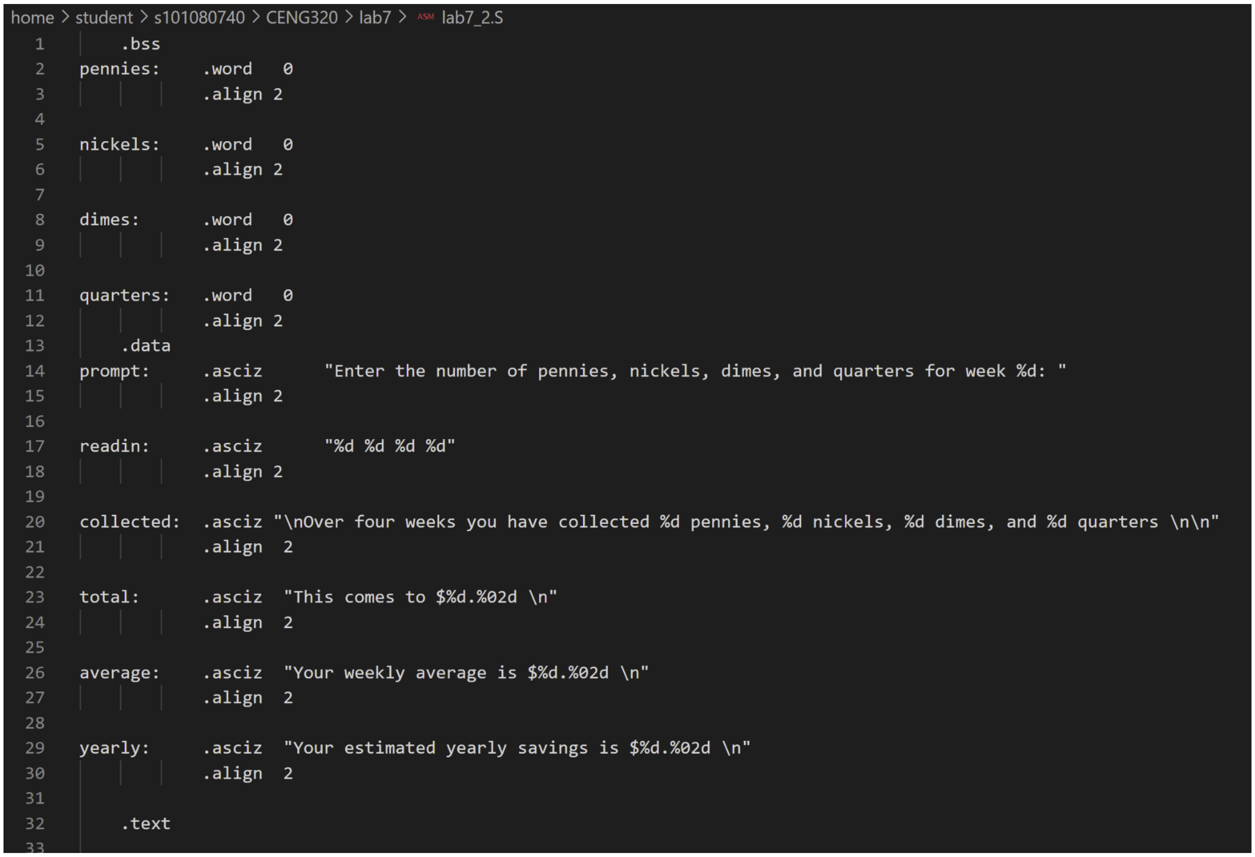


```
main:
    stp     x29, x30, [sp, #-16]!
    stp     x27, x28, [sp, #-16]!
    stp     x24, x25, [sp, #16]!
    stp     x22, x23, [sp, #16]!

    adr     x0, prompt
    bl      printf
    adr     x0, input
    adr     x1, n
    bl      scanf
    adr     x0, output
    bl      printf

    mov     w29, #1
    ldr     w27, n

test:
    mov     w0, w29
    bl      isprime

    cmp     x0, xzr
    beq     increment

    adr     x0, primeout
    mov     w1, w29
    bl      printf

increment:
    add     w29, w29 , #1
    cmp     w29, w27
    ble     test

```

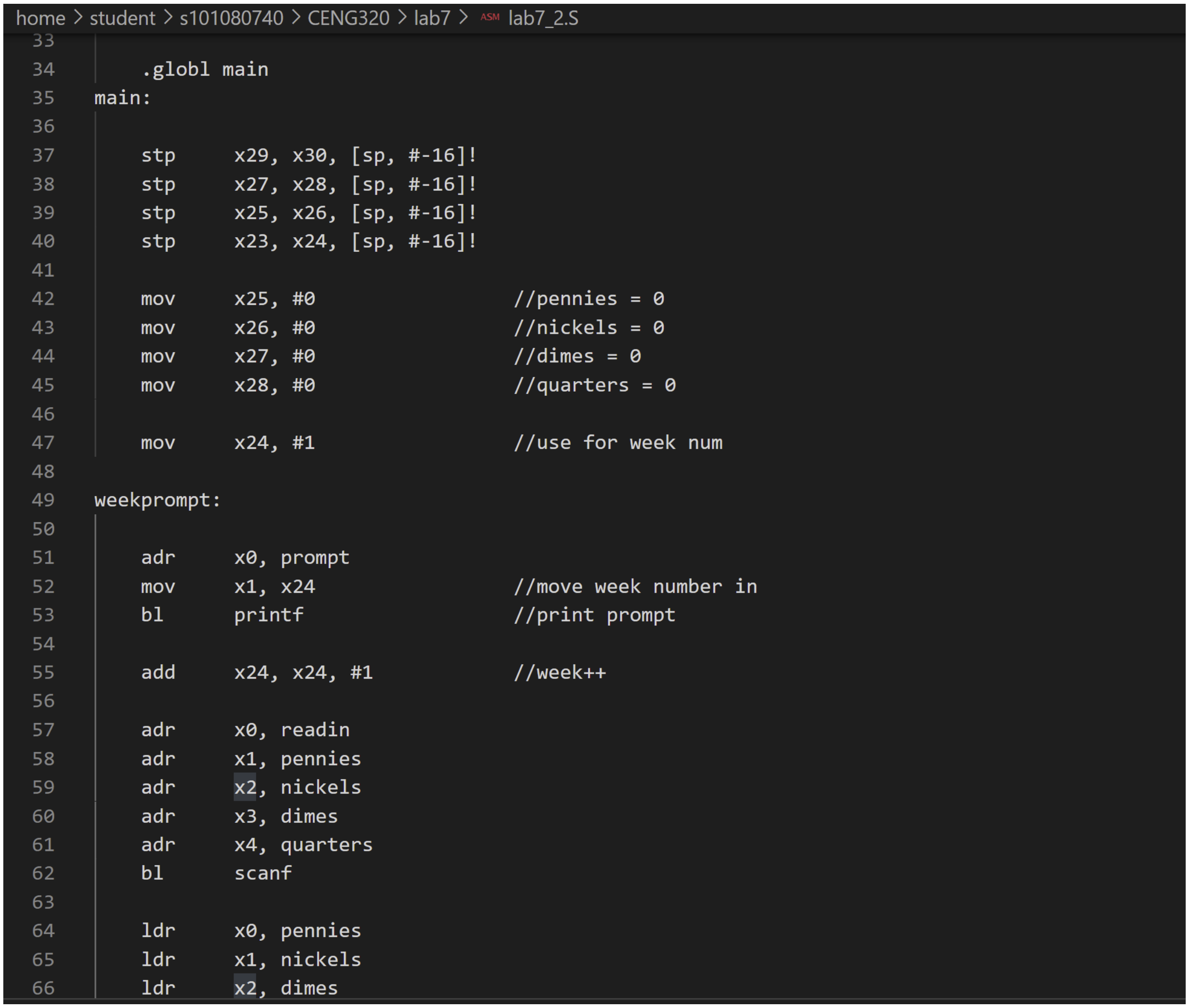


```
increment:
    add     w29, w29 , #1
    cmp     w29, w27
    ble     test

    ldp     x22, x23, [sp], #16
    ldp     x24, x25, [sp], #16
    ldp     x27, x28, [sp], #16
    ldp     x29, x30, [sp], #16

    ret
```

Part Two

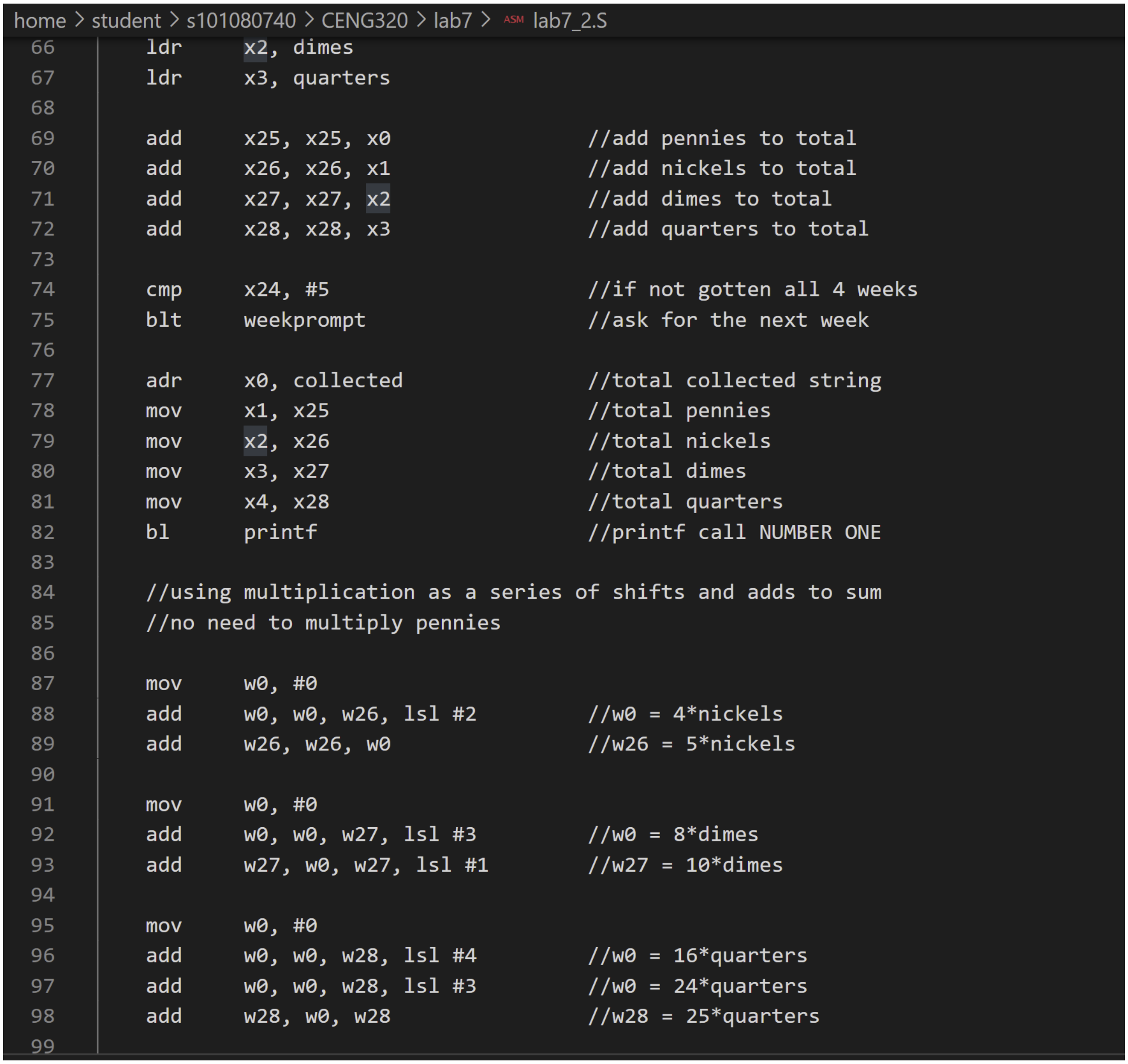


```
    .bss
pennies:    .word   0
            .align 2

nickels:    .word   0
            .align 2

dimes:      .word   0
            .align 2

quarters:   .word   0
            .align 2
    .data
prompt:     .asciz      "Enter the number of pennies, nickels, dimes, and quarters for week %d: "
            .align 2

readin:     .asciz      "%d %d %d %d"
            .align 2

collected:  .asciz "\nOver four weeks you have collected %d pennies, %d nickels, %d dimes, and %d quarters \n\n"
            .align  2

total:      .asciz  "This comes to $%d.%02d \n"
            .align  2

average:    .asciz  "Your weekly average is $%d.%02d \n"
            .align  2

yearly:     .asciz  "Your estimated yearly savings is $%d.%02d \n"
            .align  2

    .text
```



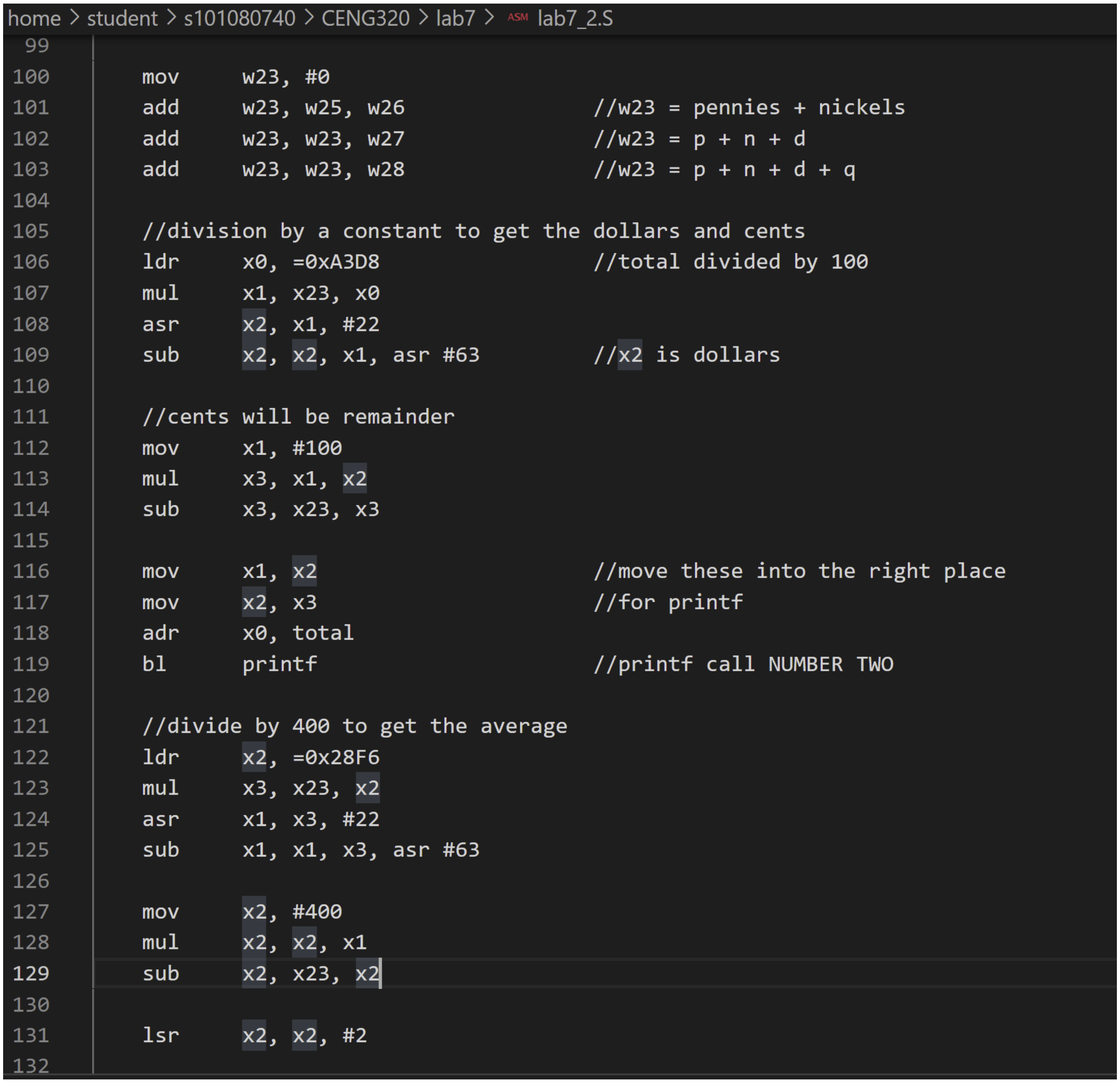
home > student > s101080740 > CENG320 > lab7 > lab7_2.S

```
    .globl main
main:

    stp     x29, x30, [sp, #-16]!
    stp     x27, x28, [sp, #-16]!
    stp     x25, x26, [sp, #-16]!
    stp     x23, x24, [sp, #-16]!

    mov     x25, #0                 //pennies = 0
    mov     x26, #0                 //nickels = 0
    mov     x27, #0                 //dimes = 0
    mov     x28, #0                 //quarters = 0

    mov     x24, #1                 //use for week num

weekprompt:

    adr     x0, prompt
    mov     x1, x24                 //move week number in
    bl      printf                  //print prompt

    add     x24, x24, #1            //week++

    adr     x0, readin
    adr     x1, pennies
    adr     x2, nickels
    adr     x3, dimes
    adr     x4, quarters
    bl      scanf

    ldr     x0, pennies
    ldr     x1, nickels
    ldr     x2, dimes
```

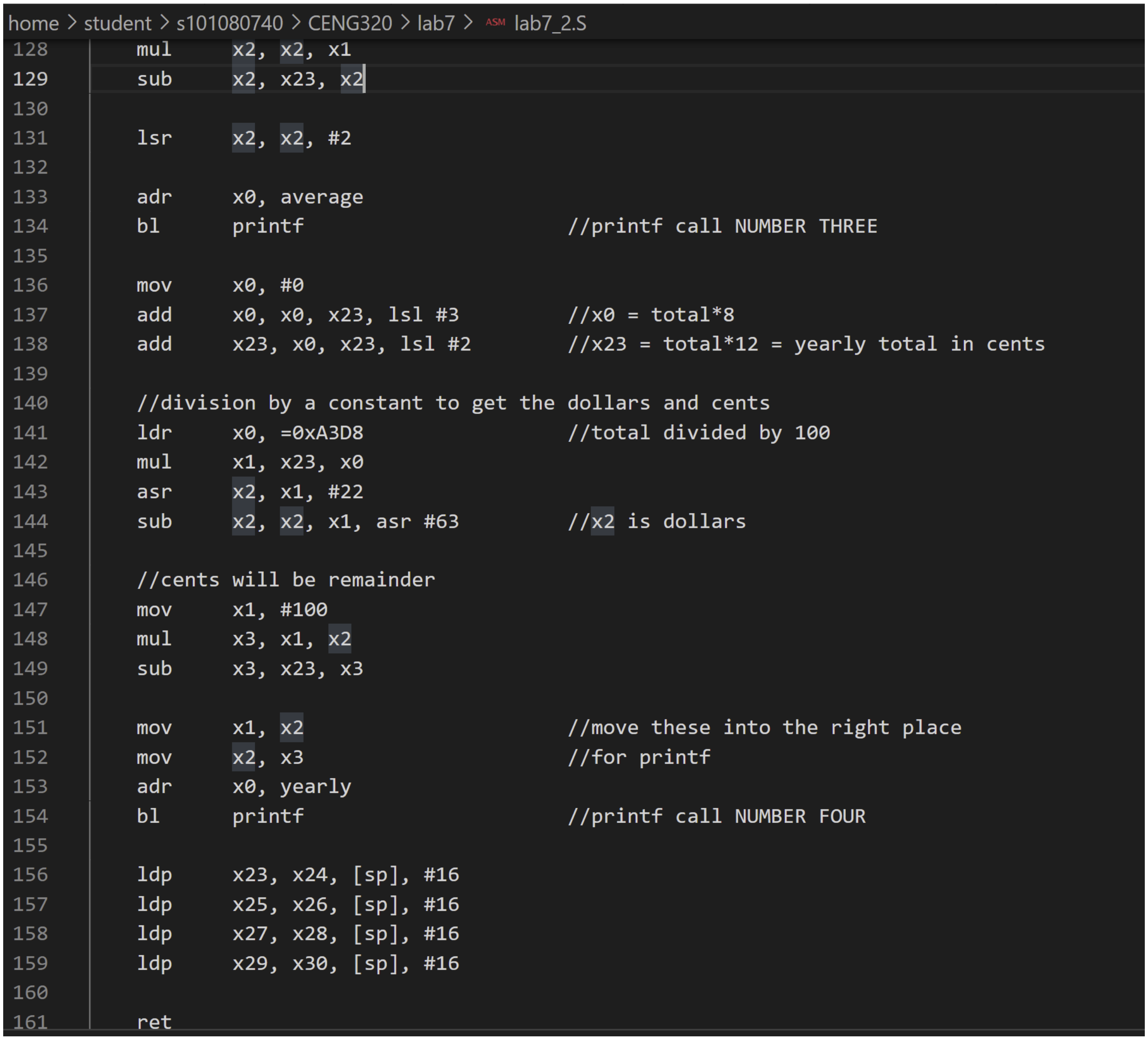


home > student > s101080740 > CENG320 > lab7 > lab7_2.S

```
    ldr     x2, dimes
    ldr     x3, quarters

    add     x25, x25, x0                //add pennies to total
    add     x26, x26, x1                //add nickels to total
    add     x27, x27, x2                //add dimes to total
    add     x28, x28, x3                //add quarters to total

    cmp     x24, #5                     //if not gotten all 4 weeks
    blt     weekprompt                  //ask for the next week

    adr     x0, collected               //total collected string
    mov     x1, x25                     //total pennies
    mov     x2, x26                     //total nickels
    mov     x3, x27                     //total dimes
    mov     x4, x28                     //total quarters
    bl      printf                      //printf call NUMBER ONE

    //using multiplication as a series of shifts and adds to sum
    //no need to multiply pennies

    mov     w0, #0
    add     w0, w0, w26, lsl #2         //w0 = 4*nickels
    add     w26, w26, w0                //w26 = 5*nickels

    mov     w0, #0
    add     w0, w0, w27, lsl #3         //w0 = 8*dimes
    add     w27, w0, w27, lsl #1        //w27 = 10*dimes

    mov     w0, #0
    add     w0, w0, w28, lsl #4         //w0 = 16*quarters
    add     w0, w0, w28, lsl #3         //w0 = 24*quarters
    add     w28, w0, w28                //w28 = 25*quarters

```

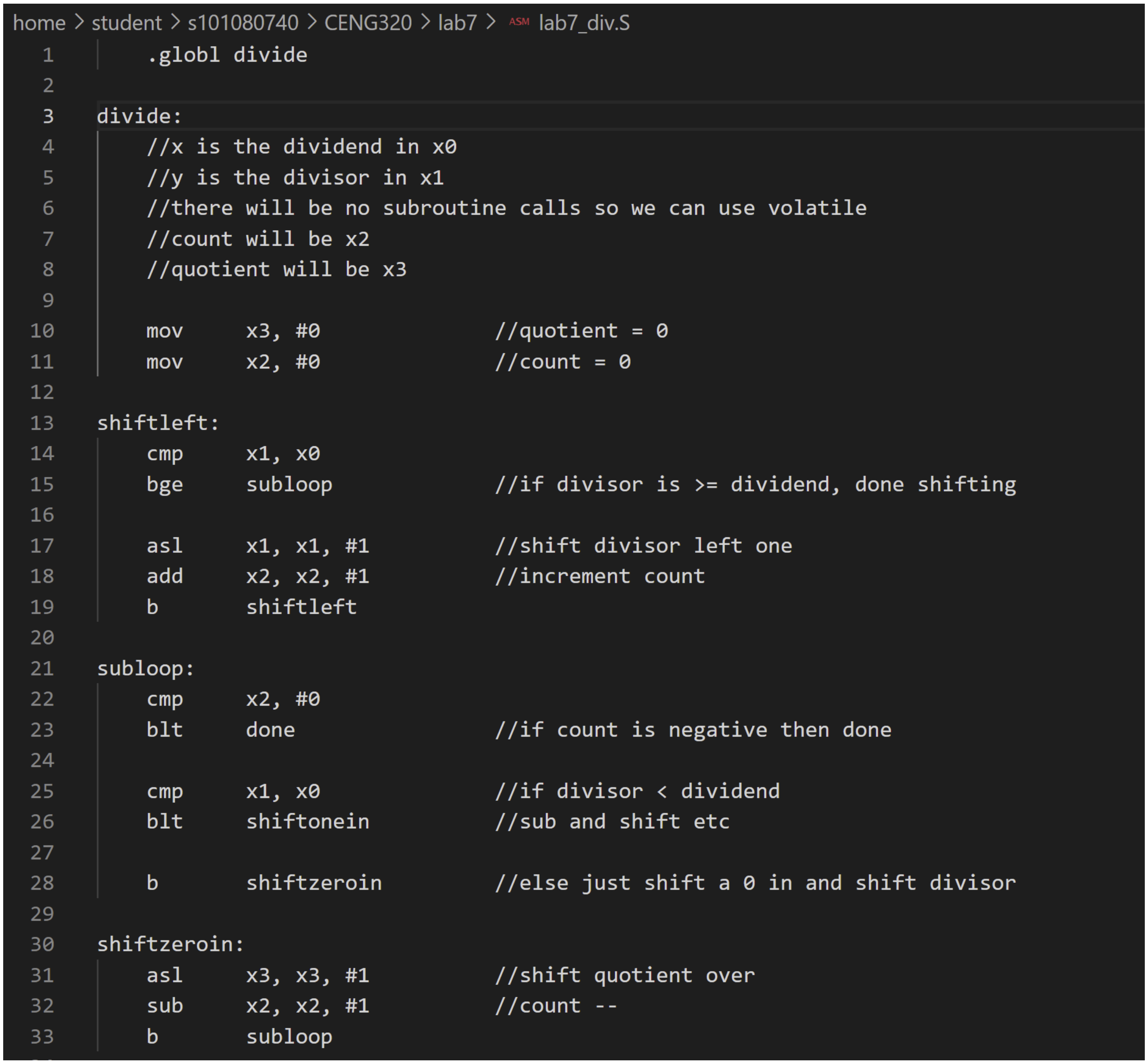


```

    mov     w23, #0
    add     w23, w25, w26               //w23 = pennies + nickels
    add     w23, w23, w27               //w23 = p + n + d
    add     w23, w23, w28               //w23 = p + n + d + q

    //division by a constant to get the dollars and cents
    ldr     x0, =0xA3D8                 //total divided by 100
    mul     x1, x23, x0
    asr     x2, x1, #22
    sub     x2, x2, x1, asr #63         //x2 is dollars

    //cents will be remainder
    mov     x1, #100
    mul     x3, x1, x2
    sub     x3, x23, x3

    mov     x1, x2                      //move these into the right place
    mov     x2, x3                      //for printf
    adr     x0, total
    bl      printf                      //printf call NUMBER TWO

    //divide by 400 to get the average
    ldr     x2, =0x28F6
    mul     x3, x23, x2
    asr     x1, x3, #22
    sub     x1, x1, x3, asr #63

    mov     x2, #400
    mul     x2, x2, x1
    sub     x2, x23, x2

    lsr     x2, x2, #2

```



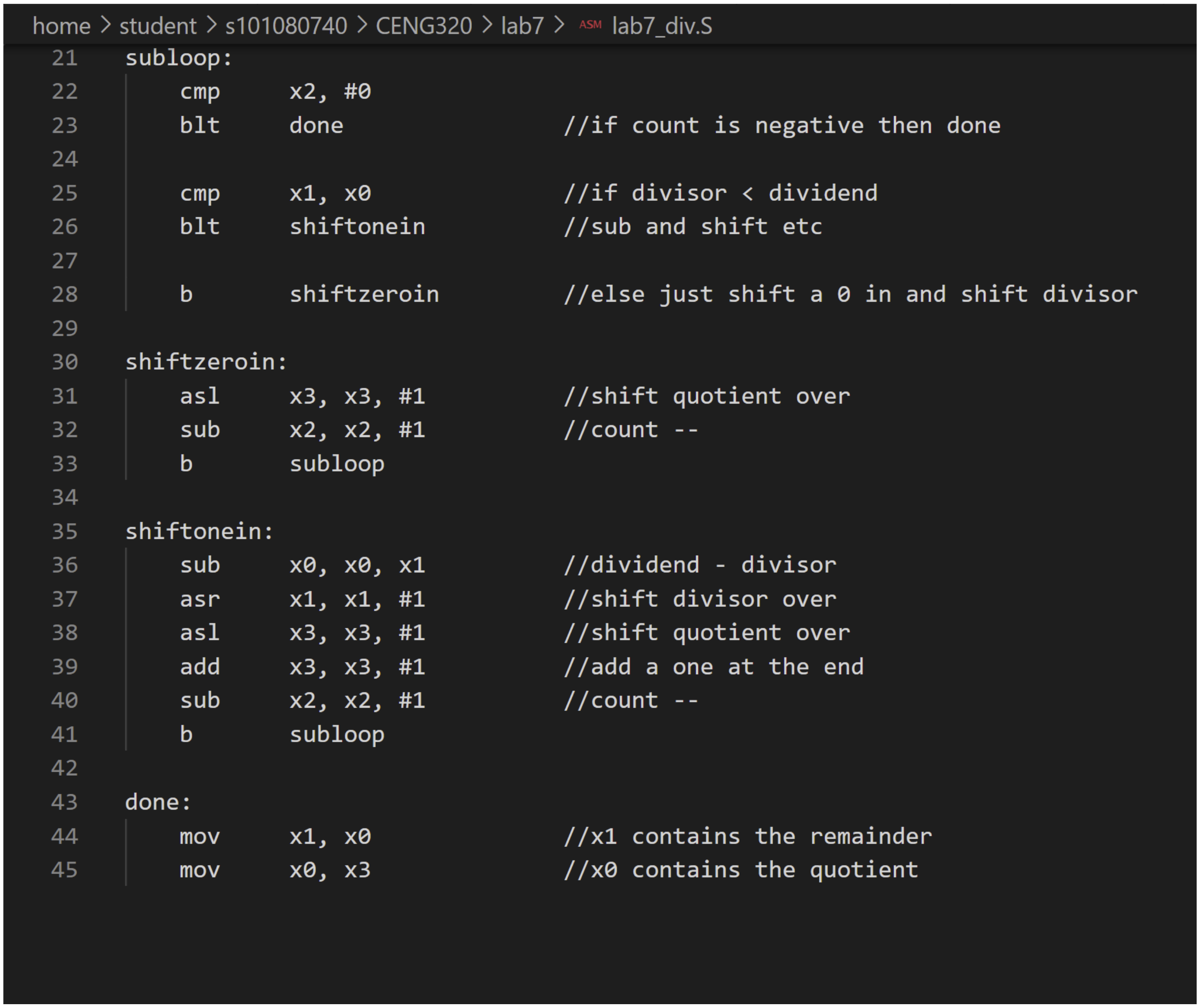
```
    mul     x2, x2, x1
    sub     x2, x23, x2

    lsr     x2, x2, #2

    adr     x0, average
    bl      printf                      //printf call NUMBER THREE

    mov     x0, #0
    add     x0, x0, x23, lsl #3         //x0 = total*8
    add     x23, x0, x23, lsl #2        //x23 = total*12 = yearly total in cents

    //division by a constant to get the dollars and cents
    ldr     x0, =0xA3D8                 //total divided by 100
    mul     x1, x23, x0
    asr     x2, x1, #22
    sub     x2, x2, x1, asr #63         //x2 is dollars

    //cents will be remainder
    mov     x1, #100
    mul     x3, x1, x2
    sub     x3, x23, x3

    mov     x1, x2                      //move these into the right place
    mov     x2, x3                      //for printf
    adr     x0, yearly
    bl      printf                      //printf call NUMBER FOUR

    ldp     x23, x24, [sp], #16
    ldp     x25, x26, [sp], #16
    ldp     x27, x28, [sp], #16
    ldp     x29, x30, [sp], #16

    ret
```

Division Algorithm



home > student > s101080740 > CENG320 > lab7 > ASM lab7_div.S

```
    .globl divide

divide:
    //x is the dividend in x0
    //y is the divisor in x1
    //there will be no subroutine calls so we can use volatile
    //count will be x2
    //quotient will be x3

    mov     x3, #0              //quotient = 0
    mov     x2, #0              //count = 0

shiftleft:
    cmp     x1, x0
    bge     subloop             //if divisor is >= dividend, done shifting

    asl     x1, x1, #1          //shift divisor left one
    add     x2, x2, #1          //increment count
    b       shiftleft

subloop:
    cmp     x2, #0
    blt     done                //if count is negative then done

    cmp     x1, x0              //if divisor < dividend
    blt     shiftonein          //sub and shift etc

    b       shiftzeroin         //else just shift a 0 in and shift divisor

shiftzeroin:
    asl     x3, x3, #1          //shift quotient over
    sub     x2, x2, #1          //count --
    b       subloop
```



home > student > s101080740 > CENG320 > lab7 > lab7_div.S

```
subloop:
    cmp     x2, #0
    blt     done                    //if count is negative then done

    cmp     x1, x0                  //if divisor < dividend
    blt     shiftonein              //sub and shift etc

    b       shiftzeroin             //else just shift a 0 in and shift divisor

shiftzeroin:
    asl     x3, x3, #1              //shift quotient over
    sub     x2, x2, #1              //count --
    b       subloop

shiftonein:
    sub     x0, x0, x1              //dividend - divisor
    asr     x1, x1, #1              //shift divisor over
    asl     x3, x3, #1              //shift quotient over
    add     x3, x3, #1              //add a one at the end
    sub     x2, x2, #1              //count --
    b       subloop

done:
    mov     x1, x0                  //x1 contains the remainder
    mov     x0, x3                  //x0 contains the quotient
```