**- 1st Day**

**PROGRAM 2: I - Table**

Write a program to produce a number table as described here. This table should be built from a single integer value provided by the user. The program will display a square 5X5 of various numbers. The entered number should show up in the table making a capital I shape. Every other spot inside the box should also be filled with a number. Those excess numbers should start with one bigger than the entered number and increment by one for every additional number used. For example, the following output should be produced when the user inputs the starting value 15:

Gimme a starting value: 15

15 15 15 15 15

16 17 15 18 19

20 21 15 22 23

24 25 15 26 27

15 15 15 15 15

And here is the output that would be produced when the user inputs the starting value 20:

Gimme a starting value: 20

20 20 20 20 20

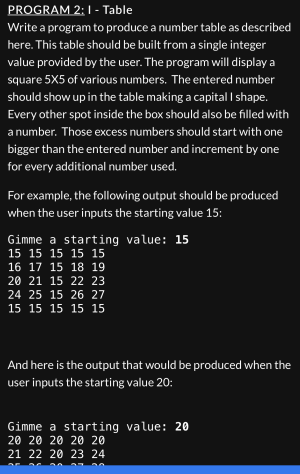
21 22 20 23 24

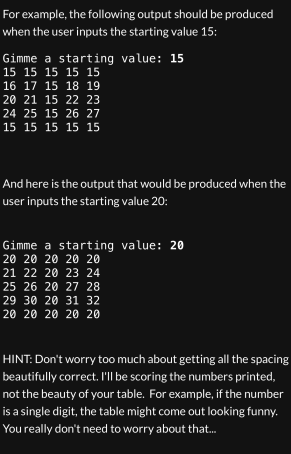
25 26 20 27 28

29 30 20 31 32

20 20 20 20 20

HINT: Don't worry too much about getting all the spacing beautifully correct. I'll be scoring the numbers printed, not the beauty of your table. For example, if the number is a single digit, the table might come out looking funny. You really don't need to worry about that…





**- 2nd Day**

**PROGRAM 4: Rolling Table**

Using the ROL and ROR instructions, write a program to produce a rolling table.

This table should be built from a single int8 value provided by the user and print 3 rows from the starting value, each offset by one from the starting value. In each individual row, the entered number should be ROL’ed, as shown below.

For example, the following output should be produced when the user inputs the starting value 4:

Gimme a starting value: 4

Rolling Table

4: 8 2 16 1

5: 10 2 20 1

6: 12 3 24 1

For example, the following output should be produced when the user inputs the starting value 12:

Gimme a starting value: 12

Rolling Table

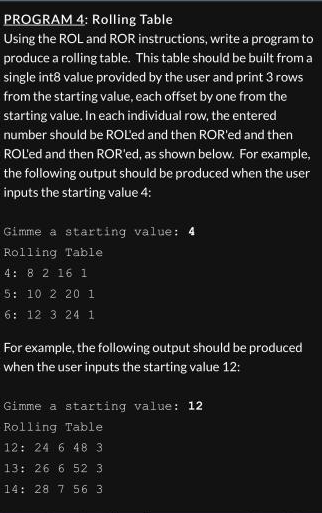
12: 24 6 48 3

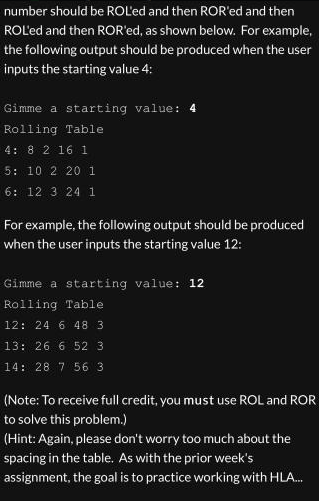
13:26 6 52 3

14: 28 7 56 3

(Note: To receive full credit, you must use ROL and ROR to solve this problem.)

(Hint: Again, Please don’t worry too much about the spacing in the table.As with the prior week’s assignment, the goal is to practice working with HLA…)

****

****

**- 3rd Day**

**PROGRAM 5: Big Numbers!**

Write an HLA Assembly language program that prompts for n, an int8 value, and then displays a repeated digit pattern starting with that number.Then repeated digit pattern should show one n, two n-1s, three n-2s,...,n-12s and n 1s, Shown below is a sample program dialogue.

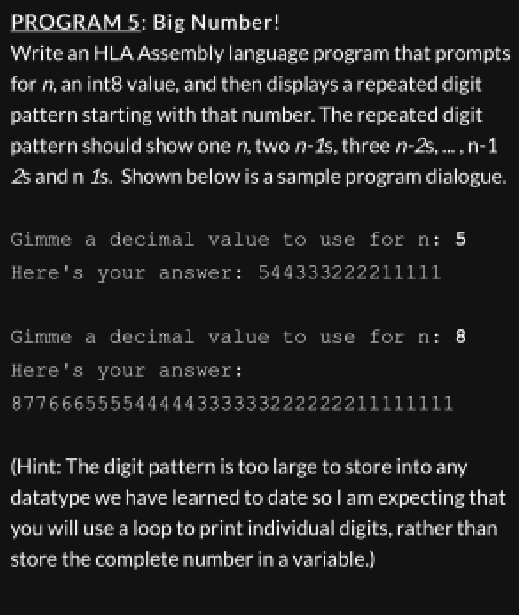
Gimme a decimal value to use for n:5

Here’s your answer:544333222211111

Gimme a decimal value to use for n:8

Here’s your answer:877666555544444333333222222211111111

(Hint: The digit pattern is too large to store into any data type we have learned to date so I am expecting that you will use a loop to print individual digits, rather than store the complete number in a variable.)



program BigNumbers;

#include ("stdlib.hhf") // include the HLA standard library

static

userInput: int8; // variable to hold user input

outerValue: int8; // variable to use for outer loop index

innerValue: int8; // variable to use for inner loop index

begin BigNumbers;

stdout.put("Gimme a decimal value to use for n: "); // prompt user for input

stdin.get(userInput); // read user input

stdout.put("Big Numbers", nl); // print a message to the console

mov( userInput, AL ); // move user input to AL register

mov( AL, outerValue ); // move value from AL register to outerValue variable

OuterLoop:

cmp( outerValue, 0 ); // compare outerValue to 0

je OuterLoopEnd; // if outerValue equals 0, jump to OuterLoopEnd label

mov(userInput, AL); // move user input to AL register

sub(outerValue, AL); // subtract outerValue from AL register

mov(AL, innerValue); // move value from AL register to innerValue variable

InnerLoop:

stdout.put(outerValue); // print the value of outerValue to the console

cmp( innerValue, 0 ); // compare innerValue to 0

je InnerLoopEnd; // if innerValue equals 0, jump to InnerLoopEnd label

dec( innerValue ); // decrement innerValue by 1

jmp InnerLoop; // jump to InnerLoop label

InnerLoopEnd:

dec( outerValue ); // decrement outerValue by 1

jmp OuterLoop; // jump to OuterLoop label

OuterLoopEnd: // label for OuterLoop end

end BigNumbers; // end of program

**PROGRAM 6: Crazy 8s Game**

Write a program that reads a set of three different numbers.

Then by subtracting off tens, determine if any of the values ends in an eight.

Continue looping as long as one of the numbers in the set ends in eight.

Three sets with a value ending in eight wins the game!

Shown below are sample program dialogues to help you build your program.

Gimme a number:20

Gimme a number:12

Gimme a number:44

Sorry Charlie! You lose the game!

Gimme a number:58

Gimme a number:23

Gimme a number:70

One of them ends in eight!

Gimme a number:1

Gimme a number:12

Gimme a number:28

One of them ends in eight!

Gimme a number:7

Gimme a number:8

Gimme a number:22

One of them ends in eight!

You Win the Game!

Gimme a number:51

Gimme a number:51

Gimme a number:51

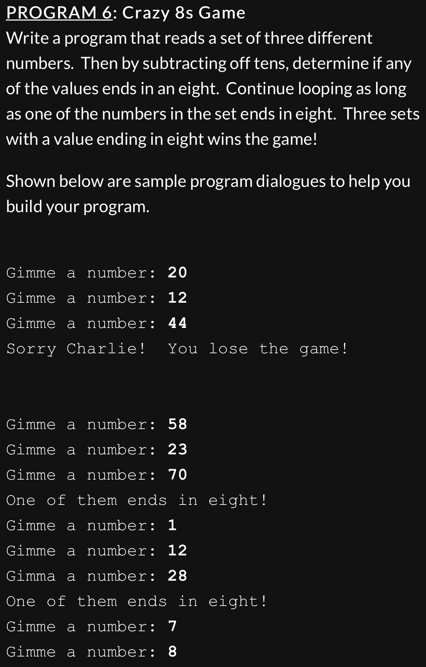
Sorry Charlie! You lose the game!

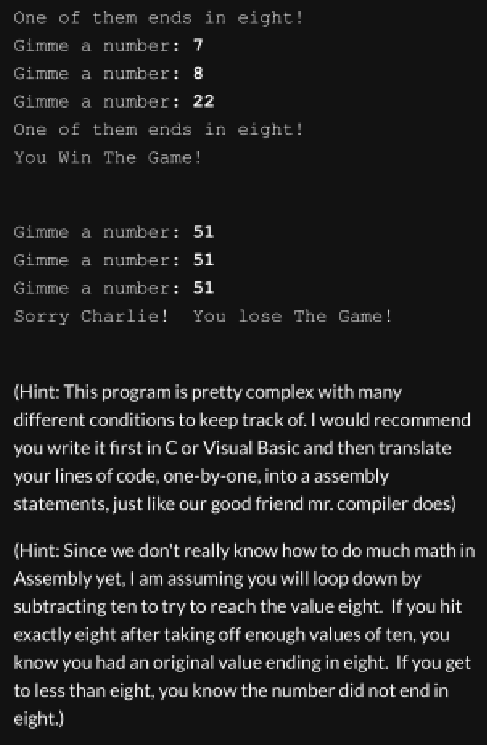
(Hint: This program is pretty complex with many different conditions to keep track of.  
I would recommend you write it first in C or Visual Basic and then translate your lines of code, one-by-one, into a assembly statements, just like our good friend mr, compiler does)

(Hint: Since we don’t really know how to do much mach in Assembly yet, I am assuming you will loop down by subtracting ten to try to reach the value eight.

If you hit exactly eight after taking off enough values of ten, you know you had an original value ending in eight .

If you get to less than eight, you know the number did not end in eight.)





**- 4th Day**

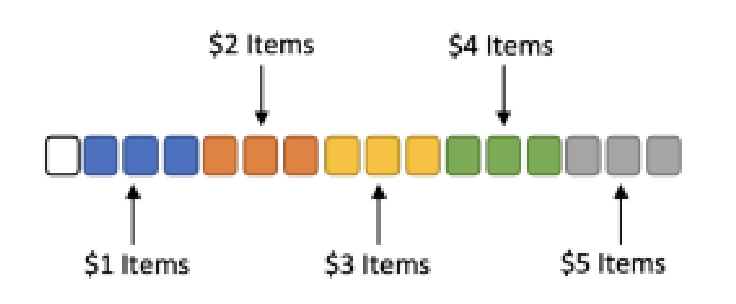
**PROGRAM 7: Dollar ValueMenu!**

Write an HLA Assembly language program that calculates the cost of an order at a logical fast food restaurant from its dollar value menu of items.

The cost will be based on a single 16-bit value entered by the value.

The value will be used to specify a number of $5 items, $4 items, $3 items, $2 items, $1 items.

The format of this bit field is diagrammed below:



Three bits are being used to specify the number of order items.  
Total the cost of the order and print this amount.

Since 16 bits are being entered here, your program should expect to read 4 hexadecimal digits.

Below are some sample program dialogues that demonstrate these ideas.

(Hint: Do this in small steps, bit-by-bit.There’s a lot to it…)

(Another Hint: HLA reads in hex format when you read directly into a register. So do that…)

(Further Hint: The most important part of this assignment is to worked with the packed data field entered by the user to extract the sub-parts out of it.The overlapping design of the intel registers helps you parse this kind of data field and you can shift the bits around to get the right part into BH or BL, fir example….)

(Final Hint: Since we haven’t learned how to do multiplication yet, although it’s kinda painful, I was expecting that you would perform the multiplication by a looping set of addition instructions.)

Feed me your order as 4 hex digits: 1000

1 $1 item

0 $2 item

0 $3 item

0 $4 item

0 $5 item

Total Order Costs: $1

Feed me your order as 4 hex digits: 1240

1 $1 item

1 $2 item

1 $3 item

0 $4 item

0 $5 item

Total Order Costs: $6

Feed me your order as 4 hex digits: 76C9

7 $1 item

3 $2 item

3 $3 item

1 $4 item

1 $5 item

Total Order Costs: $31

Feed me your order as 4 hex digits: 0009

0 $1 item

0 $2 item

0 $3 item

1 $4 item

1 $5 item

Total Order Costs: $9

