CST8216 Lab Week Twelve – Social Insurance Number (SIN) Validation

Human Resources and Skills Development Canada

Congratulations! You have been contracted by Human Resources and Skills Development Canada (HRDC) to write the back end of a software application in Assembly Language that automates the current manual process used to validate a Social Insurance Number (SIN). Additionally, as a cost savings measure, HRDC has now mandated that SINs are now only 7 digits in length.

The Current Manual Process of SIN Validation

Once the HRDC clerk has received the SIN from the client, the HRDC clerk currently uses a pencil and paper to manually perform all of the calculations required to validate a SIN, by using the Canadian Government SIN Validation Algorithm (CGSINVA). Here is an example of the current manual process that is used by the HRDC clerk to validate a SIN from a client.

Canadian Government SIN Validation Algorithm (CGSINVA)

The Social Insurance Number (SIN) is written down by the HRDC clerk.

3	3	9	2	2	3	6

Next, the validity of the SIN is checked using the following procedure:

Double the values of the alternating digits of the SIN,

3	**3**	**9**	**2**	**2**	**3**	**6**
x2		**x2**		**x2**		**x2**
6		18		4		12

Cross-add the separate digits of the products found above.

(note that 18 yields the separate digits 1 and 8)

Thus the sum of the products is:								
6	**+**	9	**+**	4	**+**	3	=	**22**
The sum of the unaffected digits is								
	3	**+**	2	**+**	3		=	**8**
						Total		**30**

The total found by following the preceding steps
must be evenly divisible by 10 (must end in 0) for it to be a valid SIN

Analysis of Current SIN **Valid SIN**

Human Resources and Skills Development Canada

Week Twelve Lab Exercise – SIN Validation

This lab forms assignment four and is worth 10% of your course marks and will be evaluated as follows. Deadlines are in accordance with the instructions in the following table.

Item	Percentage of Assignment Four	Due Date in your Portfolio Folder
Lab Week Twelve	100%	• Monday's Lab Group Tasks demonstrated and submitted to your portfolio folder by the end of lab period Monday, December 6, 2010 • Thursday's Lab Group Tasks demonstrated and submitted to your portfolio folder by the end of lab period Thursday, December 9, 2010 • Friday's Lab Group Tasks demonstrated and submitted to your portfolio folder by the end of lab period Friday, December 10, 2010
Total	100%	*To assist you in your end of semester workload, I will accept late demonstrations and submissions without penalty in my office on Friday 10 Dec 10 from 1:30 – 3:00 p.m. After that time, your lab work will not be accepted and you will receive a mark of zero for Assignment Four.*

Demonstrations may also be arranged during any of my office hours as long as all students in the group are present – missing group members will receive a mark of zero for the demonstration portion of the lab exercise. If necessary, I can provide a computer to run your demonstration on – just let me know in advance.

Assessment

Assignment Four will consist of Lab Exercise Week 12 results submitted to your portfolio folder.
This assignment may be (optionally) completed in groups of up to three students. Where the group consists of students from two or more different lab sections, the due date will be the day of the earliest member's lab period – e.g. Student A – Thursday's Lab Group, Student B and C Monday's Lab Group – due date is at the end of the Monday's Lab Group (**this is different than Assignment 3's submission guidelines**). All members of the group must be present during the demonstration; otherwise, the missing student(s) will not receive credit for the demonstration.

PURPOSE OF LAB:

The purpose of this lab exercise is to provide you with the opportunity to solidify your knowledge of HCS12 Assembly Language programming.

Lab Prerequisites

You should have completed Lab Week Ten before attempting the Task.
You should have attended all of the in-class lectures, which supports the Task.

Directory Structures

Create a folder on your N: drive called **ASM**. Then create a subfolder called **SIN**. Place your assembly language program (**Sin_Validation.asm**) in that folder.

Instructions

To gain credit for this lab exercise, complete the task outlined on the first page of this lab exercise, paying particular attention to the functionality. Demonstrate the completed task and submit the required documents into your portfolio folder held by the course professor by the due date and time.

While your portfolio work will not be returned before the final exam, your work will be marked and results posted on BB before the final exam. My solution will also be posted, which you should review before the final exam as it may form part of the final exam questions. You may pick up your portfolios after the exam is over.

PLACE THIS COMPLETED SHEET INTO YOUR PORTFOLIO.

Student Name _____ ***Lab Day*** _____

Student Name _____ ***Lab Day*** _____

Professor's Initials – Demonstration

Demo this task as soon as you have it completed – maximum demo time is 5 minutes, so plan ahead – have everything ready to go!

Assessment:

This Task must be demonstrated. **Your fully documented source code must be submitted**. Grading criteria is as follows:

Functionality (60 marks – 10 marks for each correctly validated SIN in a single program run)
Well Organized Program Layout (20 marks)
Program Documentation (20 marks)
Bonus Marks (as applicable)

Use the skeleton code provided in SIN_Validation.asm as the starting point of your Assembly Language solution. Uncomment the appropriate line of code that loads the six SINs to validate. Do NOT change any of the equate statements or Memory Locations where data is stored as I will look for the results of your solution at the addresses denoted by **InvalidResult** and **ValidResult** – see skeleton code for more information.

In your solution, I highly recommend that you consider a "main-type" program that calls subroutines via "bsr <name>" – e.g. **bsr** ToDigit. Here is a list of subroutines that you may wish to implement in your solution:

; ToDigit ; Converts from ASCII to Integer	; AddOdd ; Add up the odd numbered digits	; AddEven ; Add up the even numbered digits
; Adjust ; Add higher and lower nibbles	; Validate ; Add Even and Odd values of SIN ; to see if sum is evenly divisible- ; by 10.	

Note that the use of subroutines is not a mandatory requirement; however, without such a structure it might be difficult to assess your solution as a well organized one. It will likely also be harder for you to debug your program if you chose to not use subroutines.

Functionality CheckList – you are to check your functionality with the following test plan ***before*** demonstrating your implementation. If the functionality is not implemented or doesn't work, then note that fact.

Item	Results	As Tested by Student(s)	As Tested by Professor
Functionality	All SINs correctly validated in a single program run (60 marks)		
Well Organized Program Layout	Efficiently implemented using supplied Constants and meaningful Labels (20%)		
Program Documentation	Meaningful documentation that is not "register-specific" (20%) – e.g. No ***;load A with 6***-type comments Rather use ***;Number of SINs to validate***-*type comments*		

