**Table of Contents Page**

1. **Introduction**…......................................................................................................ii
2. **Design**

Part A…......................................................................................................................1

Part B…......................................................................................................................1

1. **Testing**

Part A…......................................................................................................................2

Part B…......................................................................................................................3

1. **Questions**….............................................................................................................3
2. **Conclusion**…..........................................................................................................4
3. **Appendix**

Part A Flowchart…................................................................................................5

Part A Assembler Code…....................................................................................6

Part A Sample Calculations…...........................................................................8

Part B Flowchart…................................................................................................9

Part B Assembler Code….................................................................................10

Part B Sample Calculations….........................................................................11

1. **Marking Sheet**…....................................................................................................................12

**Introduction**

In this design laboratory project, we were formally introduced to assembly language programming. More precisely, we were introduced to some of the basic instruction set in Netburner Eclipse. In order to gain experience coding in assembly language, in this lab exercise we were asked to write two programs.

For the first program we had to implement a body of codes that would convert an ASCII character to its hexadecimal equivalent. Valid set of characters chosen for this program were 0-9, A-F and a-f. Anything other than these characters in the memory would result in a hex error code ‘FFFFFFFF’ (or ‘-1’ in decimal) in another memory location. Moreover, if the ASCII code for ‘Enter’ was encountered, the program was to immediately exit the program.

Next, as per specifications the second program we had to write would convert an ASCII letter to its corresponding upper or lower case equivalent. As expected, the valid characters were all upper or lower case letters in the English alphabet. Similar to the first program, this program would discard invalid characters by recording an error code in the memory and would stop and exit as soon as the ASCII equivalent of ‘Enter’ was encountered.

**Design**

Part A

Firstly, we chose address register a2 to point to the specified memory location 0x2300000. This is where our data was supposed to be stored. The ‘SetZeros.s’ and ‘DataStorage.s’ template files were used to initialize the memory contents. We then specified address register a3 to point to memory location 0x2310000 so that we could later start storing our converted values starting there.

As a next step, we coded a loop that would start by temporarily storing a value from the memory location to the data register d2. This was done so that we could do further comparisons on the character to check if was valid/invalid and if it was valid, what exactly it was. If the character was not stored in a data register, we would not be able to write valid codes to do comparisons, which require that the destination operands must be a data register in our assembler language program. Anyway, following this we noted down the case of ASCII code, 0x0D, for ‘Enter’ that would signal the end of the program by channeling our program to the ‘exit’ label. Then, we accounted for all the cases that would imply an invalid character and ultimately an error code. The most obvious cases for error code were any ASCII characters less than 30 and greater than 66. We also accounted for the cases of invalid characters from 9 (0x39) to A (0x41) and F (0x46) to a (0x61) exclusively. In cases of any of these invalid values, we would then divert our program to the label ‘error’ to store the error code ‘FFFFFFFF’ in memory. After this, we checked for the cases if the character in memory was a number between 0 – 9, a letter between A-F and a-f inclusively – all in a well-organized manner of increasing ASCII values.

After the phase of validating the characters, we would then send it to the specific labels for conversion to it’s respective hex. equivalent and store it in memory. The specifics of the conversions along with details of our code are in the attached printout of the program and the hand-written sample calculations in the Appendix (page ). Lastly, our program would iterate to the next memory location for finding data and subsequent storage after manipulation until the ASCII ‘Enter’ code is reached, in which case, the program would break out of the loop and exit signaling the end.

Part B

Most of the steps followed for part B in the lab were very similar to part A. Once again, we pointed to the memory location 0x2300000 with address register a2 and to 0x2320000 with a3. We also assigned d2 with the responsibility for temporarily storing the ASCII character for further operational purposes.

Along with the restrictions for invalid characters specified in part A, we also had to opt out the numbers from 0-9 for this part of the lab. However, we had to validate the otherwise invalid ASCII characters for part A – G-Z and g-z. Our program would start a loop that would check first check for the ASCII ‘Enter’ code to exit. Then it would look for the specified invalid characters and if found, would send them to the ‘error’ label for storing an error code ‘FFFFFFFF’ in memory. Our program would reject any character that is not in the English alphabet. Particular cases of invalid characters worth mentioning are those between Z (0x5A) and a (0x61). Next, our program would check for upper and lower case letters sequentially and would consequently send the character to the specific label for modification. The details of these conversions and our program are in the attached print-out and sample calculations in the Appendix (page ). After the conversion and storage for valid/invalid characters, our program would iterate to the next memory location and run through the loop until the ASCII equivalent of ‘Enter’ is found in the memory. At that point the program would exit and no further conversions would be implemented.

**Testing**

Part A

We initially tested our program with ‘Lab1Test.s’. For this, we first connected the MTTTY Serial Terminal to the board. Next, we compiled our program and downloaded it on the microcontroller board. From the MTTTY menu, we selected to run test our program from part A. the test program went through our code and displayed all the converted characters on MTTTY. Every valid ASCII character was properly converted to its respective hex. equivalent except for the upper case letters from A to E, for which, we were getting ‘-1’. This implied that the hex. Error code ‘FFFFFFFF’ was being recorded in the memory for the valid upper case letters. Hence, we had to go back to our code to correct the error. Subsequent examination of our program brought out the problem. We noticed that we were considering all the characters from 9 (0x39) to F (0x46) exclusively to be rejected and recorded as error. In fact, it should have been A (0x41) instead of F (0x46). Fortunately, this also exposed some other problems – we mistakenly considered f (0x66) as invalid and commented out a section where we were comparing the character from memory against invalid character ranges. After making required corrections, we ran our test program again and obtained the expected hex. equivalent for all the cases involving 0-9, A-F and a-f.

We further tested our code against the provided template files – ‘DataStorage.s’, ‘DataStorage1.s’, ‘DataStorage2.s’ and ‘DataStorage3.s’. Along with the expected results for the valid characters, our program displayed the decimal error code ‘-1’ (in contrast to ‘FFFFFFFF’ in hex.) for all the invalid ASCII characters. Finally, our program exited after all the conversions were done and the ASCII code for ‘Enter’ was perceived.

Part B

As for our program in B, we checked it with ‘Lab1Test.s’ as well as against all the other provided template files following the same procedure as for part A. Looking back at the mistakes we made on part A, we double-checked out program before testing and yet, when we tried to run our program, it did not even compile. So, after another round of check we realized that we were missing a hash tag at one particular line in our code, which somehow stopped our program from compiling. Anyway, after placing in the required hash tag our program made the expected conversion of upper case letters to lower case and vice versa. We also got the same result for the letters against the all template files. In contrast to part A, we did not get a ‘-1’ for the invalid characters for this program but instead, it was blank. Further investigation revealed that our code was actually outputting everything in their corresponding ASCII hex. equivalent format but since, ‘FFFFFFFF’ does not have any meaning either in hexadecimal base or in ASCII, the respective field for output was left blank even though the memory was actually holding on to this error code. Nevertheless, our program successfully made the required conversions on the valid and invalid characters and similar to part A, exited when the ASCII ‘0x0D’ code for ‘Enter’ was encountered.

**Questions**

1. *What happens when there is no exit code ‘0x0D’ provided in the initialization process? Would it cause a problem? Why or why not?*

Yes, the lack of an exit code would cause problems to our program. The program would continue looping through the conversion process over and over again. Consequently, because of the absence of an exit code the program would try to access unspecified memory locations and would ultimately crash.

1. *How can our code be modified to provide a variable address range? For example, what if I only wanted to convert the first 10 data entires?*

One possible way that our code can be adjusted to account for a variable address range, say the first 10 data entries (in this case), would be to use a data register as a counter. In this case, we would have to initialize a data register with 10 or any other positive integer that implies the number of data entries we want to modify. Then, at the end of each run through the loop we would subtract 1 from the value stored in the data register and compare it against 0 such that if the value stored in the register after the subtraction is 0, we would exit the program. Otherwise, we would keep looping through and continue the abovementioned process.

**Conclusion**

In this design project, we implemented two programs. For the first program, we wrote down codes that aimed at converting ASCII characters 0-9, A-F and a-f to their respective hex. equivalents. Our program rejected all other characters by storing an error code in memory. For the second program, the purpose was to execute codes that would convert an upper letter to its corresponding lower case and vice versa. Both the programs were tested against some template files. In each of the cases, after resolving some errors and mistakes, we got the correct output – the valid ASCII characters were converted according to the specified conditions, error code was recorded in memory for the invalid characters and at the end, the program exited and displayed the converted values. To summarize, the design met the required specifications. Nevertheless, this was a valuable introductory learning experience for us in assembly language programming.

Part A Assembler Code

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global AssemblyProgram

AssemblyProgram:

lea -40(%a7),%a7 /\*Backing up data and address registers \*/

movem.l %d2-%d7/%a2-%a5,(%a7)

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab1a.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Ishtiak Ahmed (1269389) and Mohammad \*\*/

/\* Sirajee (1255986) \*\*/

/\* Date: February 5, 2013 \*\*/

/\* General Description: Lab 1A Introduction to Assembly Language \*\*/

/\* A program that converts the ASCII characters 0 to 9, \*\*/

/\* A-F, a-f to their hexadecimal equivalents. \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

clr.l %d2 /\* Clear data register %d2 \*/

movea.l #0x2300000, %a2 /\* Pointer for memory location 0x2300000 \*/

movea.l #0x2310000, %a3 /\* Pointer for memory location 0x2310000 \*/

Repeat: move.l (%a2), %d2 /\* [%d2] <-- (%a2) i.e., move contents starting at memory location a2 to d2\*/

/\* Exit program when the memory location contains the ASCII 'Enter' Code 0x0D \*/

cmp.l #0x0D, %d2

beq exit

/\* Check if the ASCII character is less/greater than the allowed characters and if so, send it to 'error'\*/

cmp.l #0x30, %d2

ble error

cmp.l #0x66, %d2

bgt error

/\* Check whether the ASCII character is a number (arabic numeral) or not. If it is a number, send it to 'num' \*/

cmp.l #0x30, %d2

bge is\_it\_number

is\_it\_number: cmp.l #0x39, %d2

ble num

/\* Check if the memory content is an invalid character in the ASCII table between 9 (0x39) and A (0x41) and reject it with an error code \*/

cmp.l #0x3A, %d2

bge reject

reject: cmp.l #0x40, %d2

ble error

/\* Check if character is an allowed upper case letter. If so, send it to 'bigalpha' \*/

cmp.l #0x41, %d2

bge IS\_IT\_LETTER

IS\_IT\_LETTER: cmp.l #0x46, %d2

ble bigalpha

/\* Check if the character is between F (0x46) and a (0x61) in the ASCII table and reject it by sending it to 'error' \*/

cmp.l #0x47, %d2

bge REJECT\_AGAIN

REJECT\_AGAIN: cmp.l #0x60, %d2

ble error

/\* Check if character is an allowed lower case letter. If so, send it to 'alpha' \*/

cmp.l #0x61, %d2

bge is\_it\_letter

is\_it\_letter: cmp.l #0x66, %d2

ble alpha

/\* Record the error code 'FFFFFFFF' for an invalid character in memory \*/

error: move.l #0xFFFFFFFF, %d2

move.l %d2, (%a3)

bra loop

/\* Make conversions and store the hexadecimal equivalent of the numbers in memory \*/

num: sub.l #0x30, %d2

move.l %d2, (%a3)

bra loop

/\* Make conversions and store the hexadecimal equivalent of the valid upper case letters in memory \*/

bigalpha: sub.l #0x37, %d2

move.l %d2, (%a3)

bra loop

/\* Store the hexadecimal equivalent of the valid lower case letters in memory \*/

alpha: sub.l #0x57, %d2

move.l %d2, (%a3)

/\* Repeat the loop \*/

loop: adda.l #4, %a2 /\* Point to the next memory location \*/

adda.l #4, %a3 /\* Point to the next memory location \*/

bra Repeat

exit:

/\*End of program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DO NOT MODIFY THIS --------------------------------------------\*/

movem.l (%a7),%d2-%d7/%a2-%a5 /\*Restore data and address registers \*/

lea 40(%a7),%a7

rts

/\*----------------------------------------------------------------\*/

/\* DO NOT MODIFY THIS --------------------------------------------\*/

movem.l (%a7),%d2-%d7/%a2-%a5 /\*Restore data and address registers \*/

lea 40(%a7),%a7

rts

/\*----------------------------------------------------------------\*/

Part B Assembler Code

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global AssemblyProgram

AssemblyProgram:

lea -40(%a7),%a7 /\*Backing up data and address registers \*/

movem.l %d2-%d7/%a2-%a5,(%a7)

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab1b.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Ishtiak Ahmed (1269389) and Mohammad \*\*/

/\* Sirajee (1255986) \*\*/

/\* Date: February 5, 2013 \*\*/

/\* General Description: Lab 1B Introduction to Assembly Language \*\*/

/\* A program that converts an ASCII letter to its upper or lower \*\*/

/\* case equivalent. \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

clr.l %d2 /\*Clear %d2\*/

movea.l #0x2300000, %a2 /\* Pointer for memory location 0x2300000 \*/

movea.l #0x2320000, %a3 /\* Pointer for memory location 0x2320000 \*/

Repeat: move.l (%a2), %d2 /\* [%d2] <-- (%a2) i.e., move contents starting at memory location a2 to d2\*/

/\* Exit program when the memory location contains the ASCII 'Enter' Code 0x0D \*/

cmp.l #0x0D, %d2

beq exit

/\* Check if the ASCII character is less/greater than the allowed characters (i.e., < A (0x41) or > z (0x7A) and if so, send it to 'error'\*/

cmp.l #0x41, %d2

blt error

cmp.l #0x7A, %d2

bgt error

/\* Check if memory content is an invalid character between Z (0x5A) and a (0x61) in the ASCII table and reject it with an error code \*/

cmp.l #0x5B, %d2

bge reject

reject: cmp.l #0x60, %d2

ble error

/\* Check if character is an ASCII upper case letter. If so, send it to 'bigalpha' \*/

cmp.l #0x41, %d2

bge LETTER

LETTER: cmp.l #0x5A, %d2

ble bigalpha

/\* Check if character is an ASCII lower case letter. If so, send it to 'alpha' \*/

cmp.l #0x61, %d2

bge letter

letter: cmp.l #0x7A, %d2

ble alpha

/\* Record the error code 'FFFFFFFF' for an invalid character in memory \*/

error: move.l #0xFFFFFFFF, %d2

move.l %d2, (%a3)

bra loop

/\* Store the valid upper case letter's lower case equivalent in memory \*/

bigalpha: add.l #0x20, %d2

move.l %d2, (%a3)

bra loop

/\* Store the valid lower case letter's upper case equivalent in memory \*/

alpha: sub.l #0x20, %d2

move.l %d2, (%a3)

/\* Repeat the loop \*/

loop: adda.l #4, %a2 /\* Point to the next memory location \*/

adda.l #4, %a3 /\* Point to the next memory location \*/

bra Repeat

exit:

/\*End of program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DO NOT MODIFY THIS --------------------------------------------\*/

movem.l (%a7),%d2-%d7/%a2-%a5 /\*Restore data and address registers \*/

lea 40(%a7),%a7

rts

/\*----------------------------------------------------------------\*/