Assignment 2 Report

1. Switch case and compress spaces in a character sequence program chosen.
   1. The pseudocode for the program. (JavaTPoint, 2018) (pattis, N/A)

// The program goes through the string of characters

// and checks the ascii of each character and changes

// it accordingly or not depending on what it is.

// This is then added to a new string and that string is outputted.

data = passed into program the thing getting changed

iteratorSpace = 0

iteratorNewData = 0

newData = output for program currently null

currentAsciiValue = 0

i = 0

currentAsciiValue = (int) data[i]

while currentAsciiValue != 0 do

if currentAsciiValue >= 65 and currentAsciiValue <= 90 then

currentAsciiValue = currentAsciiValue + 32

iteratorSpace = 0

newData[iteratorNewData] = (char) currentAsciiValue

iteratorNewData = iteratorNewData + 1

else if currentAsciiValue >= 97 and currentAsciiValue <= 122 then

currentAsciiValue = currentAsciiValue - 32

iteratorSpace = 0

newData[iteratorNewData] = (char) currentAsciiValue

iteratorNewData = iteratorNewData + 1

else if currentAsciiValue == 32 then

iteratorSpace = iteratorSpace + 1

if iteratorSpace == 1 then

newData[iteratorNewData] = (char) currentAsciiValue

iteratorNewData = iteratorNewData + 1

else

iteratorSpace = 0

newData[iteratorNewData] = (char) currentAsciiValue

iteratorNewData = iteratorNewData + 1

endif

i = i + 1

currentAsciiValue = (int) data[i]

endwhile

newData[iteratorNewData] = 0

* 1. The assembly language program. (pattis, N/A) Assembler Help page in Brookshear machine used.

//Switch Case and compress character sequence

//by 198735

mov 00 -> R0 //setup

mov [input] -> R1

mov [R1] -> R2

mov [output] -> R3

mov 1 -> R4

mov 0 -> R5

mov +32 -> R6

mov -32 -> R7

mov nocaps -> R8

mov space -> R9

mov else -> RA

mov newloop -> RB

startloop: jmpeq endloop, R2 //start of while loop

caps: mov +65 -> R0 //converting value to capital

jmplt R8, R2

mov +90 -> R0

jmpgt R8, R2 //jumps to nocaps

addi R2, R6 -> R2

mov 0 -> R5

mov R2 -> [R3]

addi R3, R4 -> R3

jmp newloop

nocaps: mov +97 -> R0 //converting value to lowercase

jmplt R9, R2

mov +122 -> R0

jmpgt R9, R2 //jumps to space

addi R2, R7 -> R2

mov 0 -> R5

mov R2 -> [R3]

addi R3, R4 -> R3

jmp newloop

space: mov +32 -> R0 //checks if spaces >1

jmpne RA, R2

addi R5, R4 -> R5

mov 1 -> R0

jmpne RB, R5 //if it is it doesn't add the space

mov R2 -> [R3]

addi R3, R4 -> R3

jmp newloop

else: mov 0 -> R5

mov R2 -> [R3]

addi R3, R4 -> R3

newloop: mov 00 -> R0 //resets R0 to 00 value

addi R1, R4 -> R1 //increments memory address from current

mov [R1] -> R2

jmp startloop //end of while loop

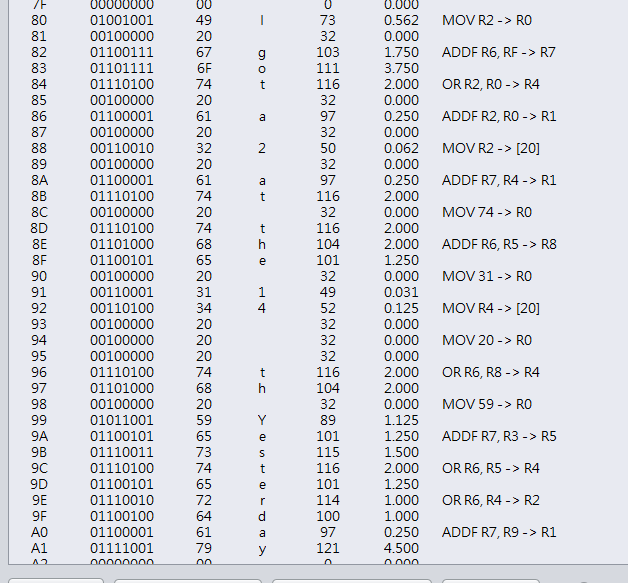
endloop: halt

input: data 10000000 //input address (80)

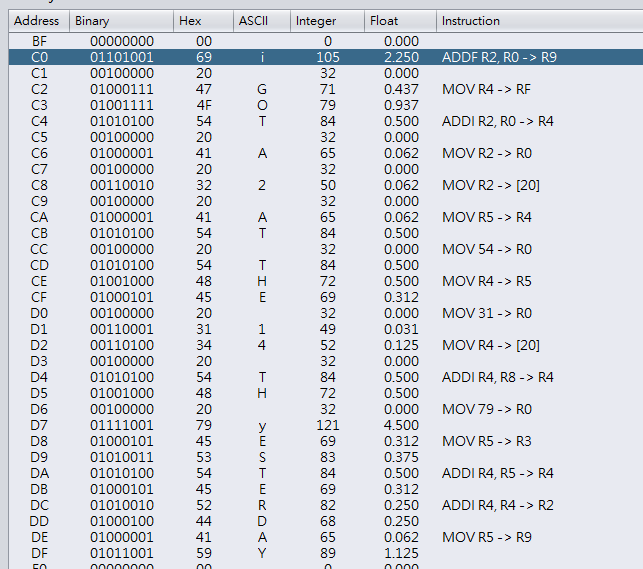
output: data 11000000 //output address (c0)

80: data "I got a 2 at the 14 th Yesterday" //input strings

* 1. The character input at the start of the program



The output at the end of the program



The program executes as specified in the brief and has 589 instructions with this input string above.

* 1. The use of the registers are as follows:
* R0 – stores the exit condition and the if conditions, as this is the default for all jump comparisons
* R1 – stores the current memory address for the input, i.e. the pointer for memory input
* R2 – stores the current input value, this is changed in the program and stored in the output string
* R3 – stores the current memory address for the output, i.e. the pointer for memory output
* R4 – stores a 1 used as an iterator for the memory addresses in R1 and R3 and used with R5
* R5 – stores the iterator for the spaces, used to decide whether there have been 2 or more spaces
* R6 – stores the add value, to convert from uppercase to lowercase i.e. +32.
* R7 – stores the minus value, to convert from lowercase to uppercase i.e. -32.
* R8 – stores the memory address of the nocaps label (seen in assembly code).
* R9 – stores the memory address of the space label (seen in assembly code).
* RA – stores the memory address of the else label (seen in assembly code).
* RB – stores the memory address of the newloop label (seen in assembly code).

The storage of memory address from R8 onwards is due to the assembly language not being able to take labels for certain JMP conditions, for example less than jump and the greater than jump.

The main jump is what forms the while loop as this keeps the program looping, this is the first jump you see in the program and is also the last instruction before halt, this jumps back to the top to check the condition again before making another iteration.

In the Pseudocode some of the IF statements have two conditions, hence the meaning of the less than and greater than jumps, as if it doesn’t meet the condition it shouldn’t execute the code below the jump. The jump statements also jump to the next if statement like the would in the Pseudocode above.

The not equals jump checks whether the character is a space and if it is then it can execute the code for space otherwise it can’t. The jump after this then checks if it’s the space is the first one or isn’t which eliminates any extra spaces.

1. 1. An interpreter translates high-level language into machine code line by line, the CPU then executes each instruction before moving on to the next instruction. (BBC Bitesize, 2019) It is used for quick de-bugging of code, as the interpreter stops when it hits an error and allows programmers to test a section of code or even just a few lines. This is instead of potentially waiting several minutes waiting for the program to compile only to find it can’t be built. Well-known languages that use interpreters include Python and JavaScript.

An interpreter maybe used in conjunction with a compiler in an IDE as both have benefits over the other in different areas. For example, run speed of a whole program is quicker with a compiler than an interpreter. Another example would be that when testing an interpreter uses less memory than a compiled program. (teach-ict, N/A) This allows the programmer to use the best tool for the job as they say, for a real world example, you can eat Baked beans with a fork, but its easier to eat them with a spoon. Therefore, the programmer may use the interpreter when developing their program, but when it comes to testing and finishing they may use the compiler to test the overall game. The compiler also has the edge here as it is harder for a user to get access to the source code of the program. And if one IDE doesn’t do this then its not as functional as the one that does have both an interpreter and a compiler.

* 1. A disassembler converts an executable file into assembly language. It is the exact opposite of an assembler. It allows users to view individual commands and see how a program works. (Tech-FAQ, 2019)

A programmer may use it to investigate a computer virus as this is no doubt some type of executable file. The disassembler can disassemble the program into assembly language and then the program can find a way to reverse whatever the virus does to the computer, a so called vaccine you could say. However, the disassembler doesn’t name the variables nicely at all they are just numbers and letters, (The Computer Language Co Inc., N/A) because it cannot work out what the original person named the variable, disassembled code is also difficult to maintain and requires you to rename it and store it on your computer. This is potentially deadly for your computer as you have essentially put a virus on your computer.

# References

BBC Bitesize, 2019. *Programming Software and the IDE.* [Online]   
Available at: https://www.bbc.com/bitesize/guides/zgmpr82/revision/2  
[Accessed 8 May 2019].

JavaTPoint, 2018. *Java Convert int to char.* [Online]   
Available at: https://www.javatpoint.com/java-int-to-char  
[Accessed 8 May 2019].

pattis, N/A. *Ascii Table.* [Online]   
Available at: https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html  
[Accessed 8 May 2019].

teach-ict, N/A. *Qualities of an interpreter.* [Online]   
Available at: https://www.teach-ict.com/as\_as\_computing/ocr/H447/F453/3\_3\_2/translators\_compilers/miniweb/pg12.htm  
[Accessed 8 May 2019].

Tech-FAQ, 2019. *Disassembler.* [Online]   
Available at: http://www.tech-faq.com/disassembler.html  
[Accessed 8 May 2019].

The Computer Language Co Inc., N/A. *Definition of Disassembler.* [Online]   
Available at: https://www.pcmag.com/encyclopedia/term/41465/disassembler  
[Accessed 8 May 2019].