Computer Systems Coursework

Index:  
Part 1 – [Report](#Report)

Part 2 – [Report](#Report2)

Part 3 – [Report](#Report3)

Part 1 – [Code (with comments)](#Code)

Part 1 – [Code (without comments)](#CodeNo)

Part 2 – [Code (with comments)](#Code2)

Part 2 – [Code (without comments)](#CodeNo2)

Part 3 – [Pseudocode](#Pseudo)

Part 3 – [Code (with comments)](#Code3)

Part 3 – [Code (without comments)](#CodeNo3)

Part 1 – Report

This assembly program *checks if the second number that we input into the machine is greater, lower or equal to the first number that we input.* This code is fully working.  
I decided to approach the solution like this: I take two inputs (The first one goes into register 0 (R0) and the second one goes into register 1 (R1)) using the *“INP Rd,2”* command. After that, I compare those using the command *“CMP”* which compares the value stored into R1 with the value stored into R0 *(CMP R1,R0)*. Then we use the branches as if statements to jump to the labels depending on the result of the comparison *(BGT, BLT, BEQ)*. Each label contains a *“MOV”* command that stores into R2 the sentence in ASCII characters that we are going to print out and then a branch to the *output label*. In the end, the *“out”* label contains the *“OUT”* commands. We have two different types of *OUT* commands in this code. The *OUT Rd, 4* is used for numbers and the *OUT* *Rd, 8* for printing ASCII characters as string. that print the second value, the sentence and finally the first value. After these, we *HALT* the program.

Part 2 – Report

This assembly program *checks if the second number that we input into the machine is greater lower or equal to the first number that we input only if it’s between 1 and 13*. This code is fully working.  
This code inherits the one from Part 1. The only difference is that when we input the first number into R0 using the *INP R0,2* command we use the *CMP* command in order to *check if the number is lower than 1* (using the syntax *#1* because we are comparing with decimal 1). In order to do this, we use *BLT* to branch to a certain label if the number is lower than 1. Then we compare the same number with the same command (*CMP*) but this time we compare it with *#13* and *we branch if the number is greater than 13*. These two branches are going to a label called *“error”* which contains an error message that gets printed on screen using the *OUT Rd,8* command. After that, the program *HALTS*.

Part 3 – Report

This assembly program *is a card game that tells us if the second card that we input is greater, lower or equal to the first one.* This code is fully working.  
This part inherits everything from Part 2. The difference here is the fact that we are converting the *numbers* into *strings* (we are not just making number 11 *“J”*, we are outputting number 2 as a *string* and not as a *number*). We are using an *array* of strings that we are going to use in order to [*convert*](#convert) the numbers into strings. This is mentioned in the pseudocode before the if statements for the comparison between the values. This is different from the actual assembly code since we are doing the conversion after the comparison and just before the output. The first value in the array is *0x00* which *will never be referenced*. This will allow us to use the card numbers as *offsets* for the array. So, we are getting A out of 1 since it’s the first card that we can get and it’s the second value in the array. We are storing into *register 3* a *space character* so we are outputting *the second number as a string*, *space character*, *“is greater than”/”is lower than”/”is equal to”*, *space character and then the first number (as you can see in the pseudocode as well in the if statements) as a string* using *mode 8* of the *OUT* command. After this we HALT the program.

Part 1 – Code (with comments)

//Part 1

//Get the first number

INP R0,2

//Get the second number

INP R1,2

//Compare the second number with the first number

CMP R1,R0

//Branches

BGT greater

BLT lower

BEQ equal

greater:

MOV R2,#greaterstring //Puts into R2 "is greater than"

B out

lower:

MOV R2,#lowerstring //Puts into R2 "is lower than"

B out

equal:

MOV R2,#equalstring //Puts into R2 "is equal to"

B out

out:

//output branch

OUT R1,4

OUT R2,8

OUT R0,4

HALT //end of the program

//DATA

greaterstring:

// define the greater than output string here

dat 0x67207369

dat 0x74616572

dat 0x74207265

dat 0x006e6168

lowerstring:

// define the less than output string here

dat 0x6c207369

dat 0x7265776f

dat 0x61687420

dat 0x0000006e

equalstring:

// define the equals than output string here

dat 0x65207369

dat 0x6c617571

dat 0x006f7420

//Registers--------------

//R0 = First number

//R1 = Second number

//R2 = Text output

[Back to index](#Index)

Part 1 – Code (without comments)

INP R0,2

INP R1,2

CMP R1,R0

BGT greater

BLT lower

BEQ equal

greater:

MOV R2,#greaterstring

B out

lower:

MOV R2,#lowerstring

B out

equal:

MOV R2,#equalstring

B out

out:

OUT R1,4

OUT R2,8

OUT R0,4

HALT

greaterstring:

dat 0x67207369

dat 0x74616572

dat 0x74207265

dat 0x006e6168

lowerstring:

dat 0x6c207369

dat 0x7265776f

dat 0x61687420

dat 0x0000006e

equalstring:

dat 0x65207369

dat 0x6c617571

dat 0x006f7420

[Back to index](#Index)

Part 2 – Code (with comments)

//Part 2

//Get first number

INP R0,2

//Check if it is lower than 1

CMP R0,#1

BLT error //Branch if it is lower than 1

//Check if it is higher than 13

CMP R0,#13

BGT error //Branch if it is higher than 13

//Get second number

INP R1,2

//Check if it is lower than 1

CMP R1,#1

BEQ error //Branch if it is lower than 1

//Check if it is higher than 13

CMP R1,#13

BEQ error //Branch if it is higher than 13

//Compare the second number with the first number

CMP R1,R0

//Branches

BGT greater

BLT lower

BEQ equal

greater:

MOV R2,#greaterstring //Puts into R2 "is greater than"

B out

lower:

MOV R2,#lowerstring //Puts into R2 "is lower than"

B out

equal:

MOV R2,#equalstring //Puts into R2 "is equal to"

B out

error:

//error messages when the numbers are not valid

MOV R2,#errorstring

OUT R2,8

HALT

out:

//output branch

OUT R1,4

OUT R2,8

OUT R0,4

HALT //end of the program

//DATA

greaterstring:

// define the greater than output string here

dat 0x67207369

dat 0x74616572

dat 0x74207265

dat 0x006e6168

lowerstring:

// define the less than output string here

dat 0x6c207369

dat 0x7265776f

dat 0x61687420

dat 0x0000006e

equalstring:

// define the equals than output string here

dat 0x65207369

dat 0x6c617571

dat 0x006f7420

errorstring:

//error string

dat 0x61766e49

dat 0x2064696c

dat 0x626d756e

dat 0x00007265

//Registers--------------

//R0 = First number

//R1 = Second number

//R2 = Text output

[Back to index](#Index)

Part 2 – Code (without comments)

INP R0,2

CMP R0,#1

BLT error

CMP R0,#13

BGT error

INP R1,2

CMP R1,#1

BEQ error

CMP R1,#13

BEQ error

CMP R1,R0

BGT greater

BLT lower

BEQ equal

greater:

MOV R2,#greaterstring

B out

lower:

MOV R2,#lowerstring

B out

equal:

MOV R2,#equalstring //Puts into R2 "is equal to"

B out

error:

MOV R2,#errorstring

OUT R2,8

HALT

out:

OUT R1,4

OUT R2,8

OUT R0,4

HALT

greaterstring:

dat 0x67207369

dat 0x74616572

dat 0x74207265

dat 0x006e6168

lowerstring:

dat 0x6c207369

dat 0x7265776f

dat 0x61687420

dat 0x0000006e

equalstring:

dat 0x65207369

dat 0x6c617571

dat 0x006f7420

errorstring:

dat 0x61766e49

dat 0x2064696c

dat 0x626d756e

dat 0x00007265

[Back to index](#Index)

Part 3 – Pseudocode

input firstNumber

if (firstNumber < 1 || firstNumber > 13)

print an error message

halt

input secondNumber

if (secondNumber < 1 || secondNumber > 13)

print error message

halt

convert firstNumber to card

convert secondNumber to card

if (secondNumber > firstNumber)

print secondNumber

print spaceChar

print greaterString

print spaceChar

print firstNumber

else if (secondNumber < firstNumber)

print secondNumber

print spaceChar

print lowerString

print spaceChar

print firstNumber

else if (secondNumber == firstNumber)

print secondNumber

print spaceChar

print equalString

print spaceChar

print firstNumber

halt

[Back to index](#Index)

Part 3 – Code (with comments)

// Part 3

// get first number

INP R0,2

//check if it is a valid number between 1 and 13

CMP R0,#1

BLT error

CMP R0,#13

BGT error

// get second number

INP R1,2

//check if it is a valid number between 1 and 13

CMP R1,#1

BLT error

CMP R1,#13

BGT error

// do comparison and branch (jump) to handler

CMP R1,R0

BGT greater

BLT lower

BEQ equal

// Point to the correct output string below

greater:

// handle greater than case

MOV R2,#greaterstring

B out

lower:

// handle lower than case

MOV R2,#lowerstring

B out

equal:

// handle when equal

MOV R2,#equalstring

B out

error:

//error messages when the numbers are not valid

MOV R2,#errorstring

OUT R2,8

HALT

out:

// print to the output device

MOV R3,#space //takes the space character

MOV R4,#cardletters

ADD R1,R1,R4

ADD R0,R0,R4

//Text output

OUT R1,8

OUT R3,8

OUT R2,8

OUT R3,8

OUT R0,8

HALT

halt //terminate program

// DATA

// Make sure strings terminate with 0x00

greaterstring:

// define the greater than output string here

dat 0x67207369

dat 0x74616572

dat 0x74207265

dat 0x006e6168

lowerstring:

// define the less than output string here

dat 0x6c207369

dat 0x7265776f

dat 0x61687420

dat 0x0000006e

equalstring:

// define the equals than output string here

dat 0x65207369

dat 0x6c617571

dat 0x006f7420

errorstring:

//error string

dat 0x61766e49

dat 0x2064696c

dat 0x626d756e

dat 0x00007265

cardletters:

// this is an array of all of the letters corresponding to the card values

dat 0x00 // this should never be referenced

dat 0x41 // A

dat 0x32 // 2

dat 0x33 // 3

dat 0x34 // 4

dat 0x35 // 5

dat 0x36 // 6

dat 0x37 // 7

dat 0x38 // 8

dat 0x39 // 9

dat 0x3031 // 10

dat 0x4A // J

dat 0x51 // Q

dat 0x4B // K

space:

dat 0x20 // space char

// minimum and maximum card values

min: dat 1 // min value of card, A

max: dat 13 // max value of card, K

//Registers---------------

//R0=First number

//R1=Second number

//R2=Text output

//R3=space character

//R4=array of cards

[Back to index](#Index)

Part 3 – Code (without comments)

INP R0,2

CMP R0,#1

BLT error

CMP R0,#13

BGT error

INP R1,2

CMP R1,#1

BLT error

CMP R1,#13

BGT error

CMP R1,R0

BGT greater

BLT lower

BEQ equal

greater:

MOV R2,#greaterstring

B out

lower:

MOV R2,#lowerstring

B out

equal:

MOV R2,#equalstring

B out

error:

MOV R2,#errorstring

OUT R2,8

HALT

out:

MOV R3,#space

MOV R4,#cardletters

ADD R1,R1,R4

ADD R0,R0,R4

OUT R1,8

OUT R3,8

OUT R2,8

OUT R3,8

OUT R0,8

HALT

greaterstring:

dat 0x67207369

dat 0x74616572

dat 0x74207265

dat 0x006e6168

lowerstring:

dat 0x6c207369

dat 0x7265776f

dat 0x61687420

dat 0x0000006e

equalstring:

dat 0x65207369

dat 0x6c617571

dat 0x006f7420

errorstring:

dat 0x61766e49

dat 0x2064696c

dat 0x626d756e

dat 0x00007265

cardletters:

dat 0x00 // this should never be referenced

dat 0x41 // A

dat 0x32 // 2

dat 0x33 // 3

dat 0x34 // 4

dat 0x35 // 5

dat 0x36 // 6

dat 0x37 // 7

dat 0x38 // 8

dat 0x39 // 9

dat 0x3031 // 10

dat 0x4A // J

dat 0x51 // Q

dat 0x4B // K

space:

dat 0x20 // space char

min: dat 1 // min value of card, A

max: dat 13 // max value of card, K

[Back to index](#Index)

Gianmarco Natuzzi

BSc Games Development