# Question 1)

The code for this question is going to follow the pseudocode given as closely as possible. To achieve this, I’ve written subroutines for both the multiplication and exponents. Both functions are written generically enough to be reusable, except for negative exponents on the power function.

The following is the source code for this question:

////////////////////////////////////////////////////////////////////////////////

/////////////////// QUESTION 1 /////////////////////

////////////////////////////////////////////////////////////////////////////////

ORG 000

JnS main / Start main method

Store exitCode / Store output of main as exitCode

Jump exit / Jump to exit.

startVal, DEC 14

main, Clear

Load startVal

Store x / x = startVal

Subt N5

mainIfCond, Skipcond 000 / if x < 5

Jump mainGte5 / else

Jump mainLt5 / then

mainLt5, Load x

Subt N2

Store y / y = x-2

Load N20

Store x / x = 20

Jump mainEndIf

mainGte5, Load x

Store pBase / base = x

Load N3

Store pExp / exponent = 3

JnS pow

Store y / y = pow(x, 3)

Store p1 / p1 = y

Load Neg1

Store p2 / p2 = -1

JnS mult

Store y / y = mult(-1, pow(x, 3))

Load N8

Store x / x = 8

mainEndIf, Load x

Output / Print x

Load y

Output / Print y

mainEnd, Load N0 / \

Store exitCode / |>> return 0

JumpI main / /

/ Input:

/ p1 - The first integer

/ p2 - The second integer

/ Output: The result of the multiplication

mult, Clear

Load N0

Store multVal / multVal = 0

multP1Eq0, Load p1

Skipcond 400 / if p1 == 0

Jump multLoopP2Neq0 / else

Jump multEnd / then return 0

multLoopP2Neq0, Load p2

Skipcond 400 / while p2 != 0

Jump multLoopBody

Jump multEnd

multLoopBody, Load p2

Skipcond 000 / if p2 < 0

Jump multPos / else

Jump multNeg / then

multNeg, Load multVal / \

Subt p1 / |>> multVal -= p1

Store multVal / /

Load p2 / \

Add N1 / |>> p2++

Store p2 / /

Jump multLoopP2Neq0

multPos, Load multVal / \

Add p1 / |>> multVal += p1

Store multVal / /

Load p2 / \

Subt N1 / |>> p2--

Store p2 / /

Jump multLoopP2Neq0

multEnd, Load multVal

JumpI mult / return multVal

/ Input:

/ pBase - The base for the operation

/ pExp - The exponent for the operation

/ Output: Integer result of the exponent operation

/ NOTE: exits with -1 when a negative exponent is received

pow, Clear

Load N1

Store powVal / powVal = 1

Load pExp

Skipcond 000 / if exp < 0

Jump powLoopCond / else

Jump powErr / then throw error

powLoopCond, Load pExp

Skipcond 400 / while exp != 0

Jump powLoopBody / do

Jump powEnd / return powVal

powLoopBody, Load pBase

Store p1 / p1 = base

Load powVal

Store p2 / p2 = powVal

JnS mult

Store powVal / powVal = mult(base, powVal)

Load pExp / \

Subt N1 / |>> pExp--

Store pExp / /

Jump powLoopCond / Go back to loop condition

powErr, Load Neg1

Store exitCode / exit(-1)

Jump exit

powEnd, Load powVal

JumpI pow / return powVal

exit, Clear

Load exitCode

Output

Halt

////////////////////////////////////////////////////////////////////////////////

//////////////////// DECLARATIONS ////////////////////

////////////////////////////////////////////////////////////////////////////////

x, DEC 20

y, DEC 0

/ mult variables / mult(a, b)

p1, DEC 0 / Param: a

p2, DEC 0 / Param: b

multVal, DEC 0 / The working value returned

/ pow variables / pow(base, exp) -> base ^ exp

pBase, DEC 0 / Param: The base for the calc

pExp, DEC 0 / Param: The exponent to raise to

powVal, DEC 0 / The current working value returned

/ Constants

Neg1, DEC -1

N0, DEC 0

N1, DEC 1

N2, DEC 2

N3, DEC 3

N5, DEC 5

N8, DEC 8

N20, DEC 20

/ reserved variables

exitCode, DEC 0 / Param: 0 - success, else - fail

The following table is the corresponding outputs for different given starting values of x:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Starting X value** | | | |
| **-25** | **14** | **15** | **20** |
| **X** | 20 | 8 | 8 | 8 |
| **Y** | -27 | -2744 | -3375 | -8000 |
| **exitCode** | 0 | 0 | 0 | 0 |

This is what each of the labels stored contain when execution first begins.

|  |  |  |
| --- | --- | --- |
| **Label** | **Address** | **Value** |

|  |  |  |
| --- | --- | --- |
| startVal | 003 | 000E |
| main | 004 | 0001 |
| mainIfCond | 008 | 8000 |
| mainLt5 | 00B | 1062 |
| mainGte5 | 011 | 1062 |
| mainEndIf | 01E | 1062 |
| mainEnd | 022 | 106B |
| mult | 025 | 001B |
| multP1Eq0 | 028 | 1064 |
| multLoopP2Neq0 | 02C | 1065 |
| multLoopBody | 030 | 1065 |
| multNeg | 034 | 1066 |
| multPos | 03B | 1066 |
| multEnd | 042 | 1066 |
| pow | 044 | 0016 |
| powLoopCond | 04B | 1068 |
| powLoopBody | 04F | 1067 |
| powErr | 059 | 106A |
| powEnd | 05C | 1069 |
| exit | 05E | A000 |
| x | 062 | 0008 |
| y | 063 | F548 |
| p1 | 064 | 0AB8 |
| p2 | 065 | 0000 |
| multVal | 066 | F548 |
| pBase | 067 | 000E |
| pExp | 068 | 0000 |
| powVal | 069 | 0AB8 |
| Neg1 | 06A | FFFF |
| N0 | 06B | 0000 |
| N1 | 06C | 0001 |
| N2 | 06D | 0002 |
| N3 | 06E | 0003 |
| N5 | 06F | 0005 |
| N8 | 070 | 0008 |
| N20 | 071 | 0014 |
| exitCode | 072 | 0000 |

And this is what each memory address contains.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0004 | 2072 | 905E | 000E | A000 | 1003 | 2062 | 406F | 8000 | 9011 | 900B | 1062 | 406D | 2063 | 1071 | 2062 |
| **010** | 901E | 1062 | 2067 | 106E | 2068 | 0044 | 2063 | 2064 | 106A | 2065 | 0025 | 2063 | 1070 | 2062 | 1062 | 6000 |
| **020** | 1063 | 6000 | 106B | 2072 | C004 | A000 | 106B | 2066 | 1064 | 8400 | 902C | 9042 | 1065 | 8400 | 9030 | 9042 |
| **030** | 1065 | 8000 | 903B | 9034 | 1066 | 4064 | 2066 | 1065 | 306C | 2065 | 902C | 1066 | 3064 | 2066 | 1065 | 406C |
| **040** | 2065 | 902C | 1066 | C025 | A000 | 106C | 2069 | 1068 | 8000 | 904B | 9059 | 1068 | 8400 | 904F | 905C | 1067 |
| **050** | 2064 | 1069 | 2065 | 0025 | 2069 | 1068 | 406C | 2068 | 904B | 106A | 2072 | 905E | 1069 | C044 | A000 | 1072 |
| **060** | 6000 | 7000 | 0014 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | FFFF | 0000 | 0001 | 0002 | 0003 | 0005 |
| **070** | 0008 | 0014 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

With an input of 15, this is what the memory looks like right after the power function returns

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0004 | 2072 | 905E | 000F | 0001 | 1003 | 2062 | 406F | 8000 | 9011 | 900B | 1062 | 406D | 2063 | 1071 | 2062 |
| **010** | 901E | 1062 | 2067 | 106E | 2068 | 0044 | 2063 | 2064 | 106A | 2065 | 0025 | 2063 | 1070 | 2062 | 1062 | 6000 |
| **020** | 1063 | 6000 | 106B | 2072 | C004 | 001B | 106B | 2066 | 1064 | 8400 | 902C | 9042 | 1065 | 8400 | 9030 | 9042 |
| **030** | 1065 | 8000 | 903B | 9034 | 1066 | 4064 | 2066 | 1065 | 306C | 2065 | 902C | 1066 | 3064 | 2066 | 1065 | 406C |
| **040** | 2065 | 902C | 1066 | C025 | 0016 | 106C | 2069 | 1068 | 8000 | 904B | 9059 | 1068 | 8400 | 904F | 905C | 1067 |
| **050** | 2064 | 1069 | 2065 | 0025 | 2069 | 1068 | 406C | 2068 | 904B | 106A | 2072 | 905E | 1069 | C044 | A000 | 1072 |
| **060** | 6000 | 7000 | 000F | 0D2F | 0D2F | FFFF | 0000 | 000F | 0000 | 0D2F | FFFF | 0000 | 0001 | 0002 | 0003 | 0005 |
| **070** | 0008 | 0014 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

And the registers are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IR** | **MAR** | **MBR** | **PC** | **IN** | **OUT** |
| 2066 | 066 | 0000 | 028 | 0000 | 0000 |

At the end of execution for an input of 15, this is what the memory contains:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0004 | 2072 | 905E | 000F | 0001 | 1003 | 2062 | 406F | 8000 | 9011 | 900B | 1062 | 406D | 2063 | 1071 | 2062 |
| **010** | 901E | 1062 | 2067 | 106E | 2068 | 0044 | 2063 | 2064 | 106A | 2065 | 0025 | 2063 | 1070 | 2062 | 1062 | 6000 |
| **020** | 1063 | 6000 | 106B | 2072 | C004 | 001B | 106B | 2066 | 1064 | 8400 | 902C | 9042 | 1065 | 8400 | 9030 | 9042 |
| **030** | 1065 | 8000 | 903B | 9034 | 1066 | 4064 | 2066 | 1065 | 306C | 2065 | 902C | 1066 | 3064 | 2066 | 1065 | 406C |
| **040** | 2065 | 902C | 1066 | C025 | 0016 | 106C | 2069 | 1068 | 8000 | 904B | 9059 | 1068 | 8400 | 904F | 905C | 1067 |
| **050** | 2064 | 1069 | 2065 | 0025 | 2069 | 1068 | 406C | 2068 | 904B | 106A | 2072 | 905E | 1069 | C044 | A000 | 1072 |
| **060** | 6000 | 7000 | 0008 | F2D1 | 0D2F | 0000 | F2D1 | 000F | 0000 | 0D2F | FFFF | 0000 | 0001 | 0002 | 0003 | 0005 |
| **070** | 0008 | 0014 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

# Question 2)

The code for this question again tries to follow the example code as closely as possible. To replicate this, I wrote my own scanf, printf, and strlen functions for reusability instead of hardcoding them in where they’re used. For all the strings, instead of knowing the length of the string beforehand, I use the backslash character as a null terminator to tell the program when the string has ended.

The following is the code for this question:

////////////////////////////////////////////////////////////////////////////////

/////////////////// QUESTION 2 /////////////////////

////////////////////////////////////////////////////////////////////////////////

ORG 000

JnS main

Store exitCode

Jump exit / exit(main())

/ Input: none

/ Output: Return value that's sent as the exit code to the program

main, Clear

Load promptAddr

Store printfpNext

JnS printf / printf("Enter a passphrase: ")

Load N8

Store func1pX

JnS func1 / func1(8)

mainEnd, Load N0

JumpI main / return 0

/ Input:

/ - pX - The length passed from main

/ Output: void

func1, Clear

Load func1yAddr

Store scanfpNext

JnS scanf / scanf(y)

Load func1yAddr

Store strlenpNext

JnS strlen / strlen(y)

Subt func1pX

func1IfCond, Skipcond 400 / if (strlen(y) != x)

Jump func1End / else

Jump func1Err / then

func1Err, Load N1

Store exitCode

Jump exit / exit(1)

func1End, Clear

JumpI func1

/ Input:

/ pNext1 - The starting mem addr of the string

/ Output: void

printf, Clear

printfLoopCond, Clear

AddI printfpNext / Load next value into acc

Subt nullTerm / subtract the null terminator code

Skipcond 400 / Will only be 0 if the character is \

Jump printfLoopBody / If not \, continue to body of loop

JumpI printf / Otherwise, break from loop

printfLoopBody, Clear

AddI printfpNext / Load the memory address into acc

Output / Print the character

Load printfpNext / Load the next memory address

Add N1 / Increment the address

Store printfpNext / Store it in the next address pointer

Jump printfLoopCond / Go back to beginning of loop

/ Input:

/ pNext2 - the starting mem addr of the string

/ Output: the length of the string

/ NOTE:

/ The null terminator is not included as the length of the string.

/ If someone enters "password" then length will be 8

/ Even though the string is \*stored\* as "password\"

strlen, Clear

strlenLoopCond, Clear

AddI strlenpNext / Load the value at starting address

Subt nullTerm / Subtract null terminator

Skipcond 400 / Check if this was null terminator

Jump strlenLoopBody / If not, Jump to body

Jump strlenEnd

strlenLoopBody, Clear / Set acc to 0

Load strlenpNext / Set acc to the current mem addr

Add N1 / Move to the next mem addr

Store strlenpNext / Store it back

Load strlenLen / Load the output length

Add N1 / Add 1 to the length

Store strlenLen / Store the length

Jump strlenLoopCond / Continue looping over characters

strlenEnd, Load strlenLen

JumpI strlen

/ Input: The memory address of the variable to input into

/ Output: void

scanf, Clear

scanfDoWhile, Input / Prompt user for input.

Store scanfInput / Store the input

StoreI scanfpNext / Store the input at the next mem addr

Load scanfpNext / Load the mem addr for curr input

Add N1 / Add one for next input addr

Store scanfpNext / Store it as next mem addr

Load scanfInput / Load the user's input

Subt nullTerm / subtract null terminator's value

Skipcond 400 / Check if it's a null terminator

Jump scanfPrintIn / Continue to the body if it is

Jump scanfEnd / Exit subroutine

scanfPrintIn, Load scanfInput

Output

Jump scanfDoWhile

scanfEnd, JumpI scanf

/ Input:

/ exitCode - The exit code for the program. 0: pass, 1: fail (POSIX standard)

/ Output: N/A, kills the program

exit, Clear

Load exitMsgAddr

Store printfpNext

JnS printf / printf("Exit code: ")

Load exitCode

Skipcond 400 / if (exitCode == 0)

Jump exitSuccess / then

Jump exitFail / else

exitSuccess, Load Uni0

Output / Print 0

Halt

exitFail, Load Uni1

Output / Print 1

Halt

////////////////////////////////////////////////////////////////////////////////

//////////////////// DECLARATIONS ////////////////////

////////////////////////////////////////////////////////////////////////////////

/ Constants

Uni0, HEX 30

Uni1, HEX 31

N0, DEC 0

N1, DEC 1

N8, DEC 8

newline, HEX 0a

nullTerm, HEX 5C / Null termination: "\"

/ main's variables.

promptAddr, HEX 61 / Starting Address

prompt, HEX 45 / E

HEX 6E / n

HEX 74 / t

HEX 65 / e

HEX 72 / r

HEX 20

HEX 61 / a

HEX 20

HEX 70 / p

HEX 61 / a

HEX 73 / s

HEX 73 / s

HEX 70 / p

HEX 68 / h

HEX 72 / r

HEX 61 / a

HEX 73 / s

HEX 65 / e

HEX 3A / :

HEX 0A / New line

HEX 5C / NULL termination

/ func1 variables

func1pX, DEC 0 / Param: size of password allowed

func1yAddr, HEX 78 / starting mem addr of lY

func1y, DEC 0 / Local variable Y

DEC 0

DEC 0

DEC 0

DEC 0

DEC 0

DEC 0

DEC 0

DEC 0

/ printf's variables

printfpNext, DEC 0 / Param: first mem addr of string

/ scanf variables

scanfpNext, DEC 0 / Param: starting address to store into

scanfInput, DEC 0 / Local: The user's current input

/ strlen variables

strlenpNext, DEC 0 / The starting address of the string

strlenLen, DEC 0 / The outputed length

/ Exit

exitCode, DEC 0 / Param: 0 - success, else - fail

exitMsgAddr, HEX 88

exitMsg, HEX A / New line

HEX 45 / E

HEX 78 / x

HEX 69 / i

HEX 74 / t

HEX 20

HEX 63 / c

HEX 6f / o

HEX 64 / d

HEX 65 / e

HEX 3a / :

HEX 20

HEX 5C / \

The following table corresponds to the output for different inputted passwords:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Passphrase** | | | |
| **Hello** | **HelloWorld** | **Password** | **12345678** |
| **Prompt** | Enter a passphrase: | Enter a passphrase: | Enter a passphrase: | Enter a passphrase: |
| **Password** | Hello\ | HelloWorld\ | Password\ | 12345678\ |
| **Exit message** | Exit code: 1 | Exit code: 1 | Exit code: 0 | Exit code: 0 |

This is what each of the labels stored contain when execution first begins.

|  |  |  |
| --- | --- | --- |
| **Label** | **Address** | **Value** |

|  |  |  |
| --- | --- | --- |
| main | 003 | A000 |
| mainEnd | 00A | 105B |
| func1 | 00C | A000 |
| func1IfCond | 014 | 8400 |
| func1Err | 017 | 105C |
| func1End | 01A | A000 |
| printf | 01C | A000 |
| printfLoopCond | 01D | A000 |
| printfLoopBody | 023 | A000 |
| strlen | 02A | A000 |
| strlenLoopCond | 02B | A000 |
| strlenLoopBody | 031 | A000 |
| strlenEnd | 039 | 1085 |
| scanf | 03B | A000 |
| scanfDoWhile | 03C | 5000 |
| scanfPrintIn | 047 | 1083 |
| scanfEnd | 04A | C03B |
| exit | 04B | A000 |
| exitSuccess | 053 | 1059 |
| exitFail | 056 | 105A |
| Uni0 | 059 | 0030 |
| Uni1 | 05A | 0031 |
| N0 | 05B | 0000 |
| N1 | 05C | 0001 |
| N8 | 05D | 0008 |
| newline | 05E | 000A |
| nullTerm | 05F | 005C |
| promptAddr | 060 | 0061 |
| prompt | 061 | 0045 |
| func1pX | 076 | 0000 |
| func1yAddr | 077 | 0078 |
| func1y | 078 | 0000 |
| printfpNext | 081 | 0000 |
| scanfpNext | 082 | 0000 |
| scanfInput | 083 | 0000 |
| strlenpNext | 084 | 0000 |
| strlenLen | 085 | 0000 |
| exitCode | 086 | 0000 |
| exitMsgAddr | 087 | 0088 |
| exitMsg | 088 | 000A |

And this is what each memory address contains.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0003 | 2086 | 904B | A000 | 1060 | 2081 | 001C | 105D | 2076 | 000C | 105B | C003 | A000 | 1077 | 2082 | 003B |
| **010** | 1077 | 2084 | 002A | 4076 | 8400 | 901A | 9017 | 105C | 2086 | 904B | A000 | C00C | A000 | A000 | B081 | 405F |
| **020** | 8400 | 9023 | C01C | A000 | B081 | 6000 | 1081 | 305C | 2081 | 901D | A000 | A000 | B084 | 405F | 8400 | 9031 |
| **030** | 9039 | A000 | 1084 | 305C | 2084 | 1085 | 305C | 2085 | 902B | 1085 | C02A | A000 | 5000 | 2083 | E082 | 1082 |
| **040** | 305C | 2082 | 1083 | 405F | 8400 | 9047 | 904A | 1083 | 6000 | 903C | C03B | A000 | 1087 | 2081 | 001C | 1086 |
| **050** | 8400 | 9053 | 9056 | 1059 | 6000 | 7000 | 105A | 6000 | 7000 | 0030 | 0031 | 0000 | 0001 | 0008 | 000A | 005C |
| **060** | 0061 | 0045 | 006E | 0074 | 0065 | 0072 | 0020 | 0061 | 0020 | 0070 | 0061 | 0073 | 0073 | 0070 | 0068 | 0072 |
| **070** | 0061 | 0073 | 0065 | 003A | 000A | 005C | 0000 | 0078 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| **080** | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0088 | 000A | 0045 | 0078 | 0069 | 0074 | 0020 | 0063 | 006F |
| **090** | 0064 | 0065 | 003A | 0020 | 005C | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

With an input of “password\”, this is what memory contains right before calling the strlen function

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0003 | 2086 | 904B | 0001 | 1060 | 2081 | 001C | 105D | 2076 | 000C | 105B | C003 | 000A | 1077 | 2082 | 003B |
| **010** | 1077 | 2084 | 002A | 4076 | 8400 | 901A | 9017 | 105C | 2086 | 904B | A000 | C00C | 0007 | A000 | B081 | 405F |
| **020** | 8400 | 9023 | C01C | A000 | B081 | 6000 | 1081 | 305C | 2081 | 901D | 0013 | A000 | B084 | 405F | 8400 | 9031 |
| **030** | 9039 | A000 | 1084 | 305C | 2084 | 1085 | 305C | 2085 | 902B | 1085 | C02A | 0010 | 5000 | 2083 | E082 | 1082 |
| **040** | 305C | 2082 | 1083 | 405F | 8400 | 9047 | 904A | 1083 | 6000 | 903C | C03B | A000 | 1087 | 2081 | 001C | 1086 |
| **050** | 8400 | 9053 | 9056 | 1059 | 6000 | 7000 | 105A | 6000 | 7000 | 0030 | 0031 | 0000 | 0001 | 0008 | 000A | 005C |
| **060** | 0061 | 0045 | 006E | 0074 | 0065 | 0072 | 0020 | 0061 | 0020 | 0070 | 0061 | 0073 | 0073 | 0070 | 0068 | 0072 |
| **070** | 0061 | 0073 | 0065 | 003A | 000A | 005C | 0008 | 0078 | 0070 | 0061 | 0073 | 0073 | 0077 | 006F | 0072 | 0064 |
| **080** | 005C | 0075 | 0081 | 005C | 0078 | 0000 | 0000 | 0088 | 000A | 0045 | 0078 | 0069 | 0074 | 0020 | 0063 | 006F |
| **090** | 0064 | 0065 | 003A | 0020 | 005C | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

And the registers are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IR** | **MAR** | **MBR** | **PC** | **IN** | **OUT** |
| 1084 | 084 | 0078 | 033 | 005C | 0064 |

At the end of execution, with an input of “password\”, this is what the memory contains:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0003 | 2086 | 904B | 0001 | 1060 | 2081 | 001C | 105D | 2076 | 000C | 105B | C003 | 000A | 1077 | 2082 | 003B |
| **010** | 1077 | 2084 | 002A | 4076 | 8400 | 901A | 9017 | 105C | 2086 | 904B | A000 | C00C | 004F | A000 | B081 | 405F |
| **020** | 8400 | 9023 | C01C | A000 | B081 | 6000 | 1081 | 305C | 2081 | 901D | 0013 | A000 | B084 | 405F | 8400 | 9031 |
| **030** | 9039 | A000 | 1084 | 305C | 2084 | 1085 | 305C | 2085 | 902B | 1085 | C02A | 0010 | 5000 | 2083 | E082 | 1082 |
| **040** | 305C | 2082 | 1083 | 405F | 8400 | 9047 | 904A | 1083 | 6000 | 903C | C03B | A000 | 1087 | 2081 | 001C | 1086 |
| **050** | 8400 | 9053 | 9056 | 1059 | 6000 | 7000 | 105A | 6000 | 7000 | 0030 | 0031 | 0000 | 0001 | 0008 | 000A | 005C |
| **060** | 0061 | 0045 | 006E | 0074 | 0065 | 0072 | 0020 | 0061 | 0020 | 0070 | 0061 | 0073 | 0073 | 0070 | 0068 | 0072 |
| **070** | 0061 | 0073 | 0065 | 003A | 000A | 005C | 0008 | 0078 | 0070 | 0061 | 0073 | 0073 | 0077 | 006F | 0072 | 0064 |
| **080** | 005C | 0094 | 0081 | 005C | 0080 | 0008 | 0001 | 0088 | 000A | 0045 | 0078 | 0069 | 0074 | 0020 | 0063 | 006F |
| **090** | 0064 | 0065 | 003A | 0020 | 005C | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

# Question 3)

My code for question 3 is considerable simpler than that of question 1 and 2. There’s very little reusability to be done in this question, so there’s no extra methods to be writing and calling. I’m just emulating the for loop as much as I can, although under the hood as seen here, it’s just a while loop since for loops are just syntactic sugar.

The following is the code for this question:

////////////////////////////////////////////////////////////////////////////////

/////////////////// QUESTION 3 ////////////////////

////////////////////////////////////////////////////////////////////////////////

ORG 000

JnS main

Store exitCode

Jump exit / exit(main())

main, Clear

mainFor, Load N0

Store i / for i = 0

mainForCond, Subt N100

Skipcond 000 / ; i < 100

Jump mainForEnd

jump mainForBody

mainForBody, Load x

Add i

Store x / x += i

mainIfCond, Subt N102

Skipcond 000 / if (x > 102)

Jump mainForEnd / then

Load i / \

Add N1 / |>> i++

Store i / /

Jump mainForCond

mainForEnd, Load x

Output / print x

mainEnd, Load N0

JumpI main / return 0

exit, Clear

Load exitCode

Output

Halt

////////////////////////////////////////////////////////////////////////////////

//////////////////// DECLARATIONS ////////////////////

////////////////////////////////////////////////////////////////////////////////

N0, DEC 0

N1, DEC 1

N100, DEC 100

N102, DEC 102

i, DEC 0

x, DEC 10

exitCode, DEC 0 / Param: 0 - success, else – fail

There are no inputs to this program since the value of x is hardcoded to 10, however we can test what happens if we change that:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Starting X value** | | | |
| **-20** | **0** | **10** | **50** |
| **X** | 116 | 105 | 115 | 105 |
| **exitCode** | 0 | 0 | 0 | 0 |

This is what each of the labels stored contain when execution first begins.

|  |  |  |
| --- | --- | --- |
| **Label** | **Address** | **Value** |

|  |  |  |
| --- | --- | --- |
| main | 003 | A000 |
| mainFor | 004 | 101C |
| mainForCond | 006 | 401E |
| mainForBody | 00A | 1021 |
| mainIfCond | 00D | 401F |
| mainForEnd | 014 | 1021 |
| mainEnd | 016 | 101C |
| exit | 018 | A000 |
| N0 | 01C | 0000 |
| N1 | 01D | 0001 |
| N100 | 01E | 0064 |
| N102 | 01F | 0066 |
| i | 020 | 0000 |
| x | 021 | 000A |
| exitCode | 022 | 0000 |

And this is what each memory address contains.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0003 | 2022 | 9018 | A000 | 101C | 2020 | 401E | 8000 | 9014 | 900A | 1021 | 3020 | 2021 | 401F | 8000 | 9014 |
| **010** | 1020 | 301D | 2020 | 9006 | 1021 | 6000 | 101C | C003 | A000 | 1022 | 6000 | 7000 | 0000 | 0001 | 0064 | 0066 |
| **020** | 0000 | 000A | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

With a starting x value of 10, this is what memory looks like right after breaking from the loop:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0003 | 2022 | 9018 | 0001 | 101C | 2020 | 401E | 8000 | 9014 | 900A | 1021 | 3020 | 2021 | 401F | 8000 | 9014 |
| **010** | 1020 | 301D | 2020 | 9006 | 1021 | 6000 | 101C | C003 | A000 | 1022 | 6000 | 7000 | 0000 | 0001 | 0064 | 0066 |
| **020** | 000E | 0073 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

And the registers are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IR** | **MAR** | **MBR** | **PC** | **IN** | **OUT** |
| 9014 | 00F | 9014 | 014 | 0000 | 0000 |

At the end of execution, this is what the memory contains when using an x value of 10:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0** | **+1** | **+2** | **+3** | **+4** | **+5** | **+6** | **+7** | **+8** | **+9** | **+A** | **+B** | **+C** | **+D** | **+E** | **+F** |
| **000** | 0003 | 2022 | 9018 | 0001 | 101C | 2020 | 401E | 8000 | 9014 | 900A | 1021 | 3020 | 2021 | 401F | 8000 | 9014 |
| **010** | 1020 | 301D | 2020 | 9006 | 1021 | 6000 | 101C | C003 | A000 | 1022 | 6000 | 7000 | 0000 | 0001 | 0064 | 0066 |
| **020** | 000E | 0073 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |