University of New York in Tirana
Fall Semester 2020-2021

Delivered to: Dr. Ervin Ramollari

Delivered by: Ledjo Pilua

Computer Organization & System Architecture COURSE PROJECT

Notes and Documentation

Introduction

This is a document report for the course project of Computer Organization & System Architecture class. It includes detailed description from the beginning of the stages of the project (logic, algorithm, functions used) to then visually representing it through chart flows for the design and also the code of C++ programs and ARMv8 (AArch64) is pasted. Screenshots and references are at the end of this document.

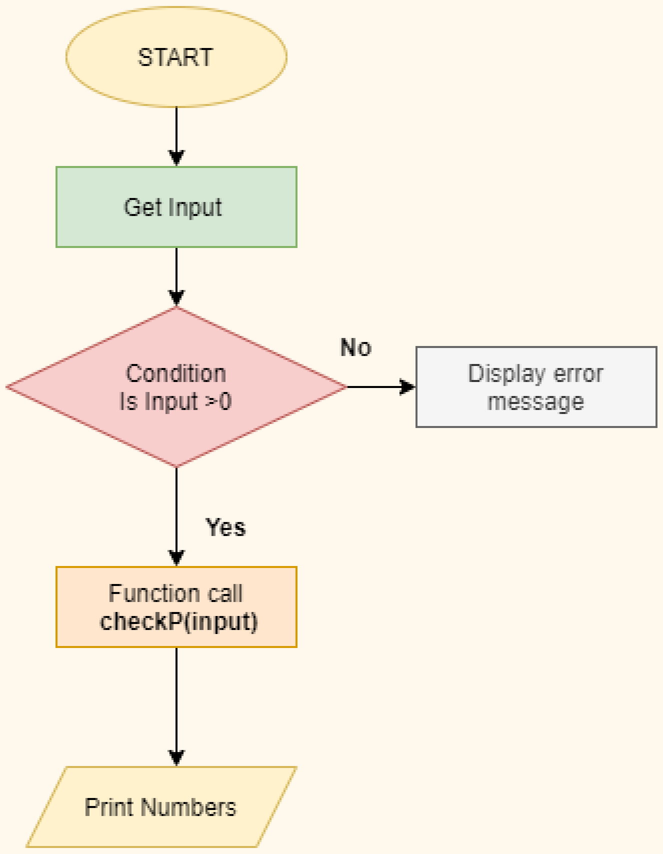
TASK I ‘’PrimeLess’’

The first task of the project consists on writing a program, first in C++ and in ARMv8 which will read a number (integer) and if that number is not a positive integer it will display error and if it is indeed a positive integer it is going to output all the prime numbers which are less than the inputted number.

First step, I created a logical design for the following example to then implement it to a C++ code. Essentially, I want the program to read a number which is inputted by the user and then check if that number is positive. If it is not → it will display an error message to the user. If it is, it will continue with another function which will generate all the prime numbers that are less than the inputted positive integer. The way to do it is simple, after declaring an integer type, the user is prompted to insert a number and then that number is compared using If-else-if statement which will only execute blocks of code if criteria is met and it's useful for checking multiple conditions.

Having inputted a positive integer, that is if(input>0), the block code will contain a call to a function which takes a parameter int, in this case input number, and two for loops will iterate until all the numbers are printed. The number "1" is not a prime number therefore it won't be printed. Prime numbers are those positive integers which can be divided only by 1 and themselves.

The first for loop checks for all numbers starting from two up to the inputted number, without including it since we only want to print those prime numbers that are less than. The second for loop will check that if the number is not prime, that is (j % i == 0), it will change the value of y to zero and will continue to check for other numbers. If the (j % i == 0) is false and the initialized y is equal to 1, it will print the number and continue for others until it reaches the upper bound which in this case is the inputted number.



C++ Code is below:

```
#include <iostream>
using namespace std;


void testP(int x)
{
    for (int j=2;j<x;j++){
                  int y=1;
                        for (int i=2;i<j;i++){
                        if (j%i==0){

                       y=0;
                  }
            }
            if (y==1){
                  cout<<j;
                  cout<<" ";
            }
      }
}
int main (){
      int primelessnumb;
      cout<<"Please, enter the number you want to find prime less ";
      cin>> primelessnumb;


      if (primelessnumb>0) {
          cout<<"Prime numbers smaller than your number are:";
      testP(primelessnumb);}

      else if (primelessnumb == 0){
      cout<<"No prime number < 0";}

      else {
          cout<<"error:";}


}
```

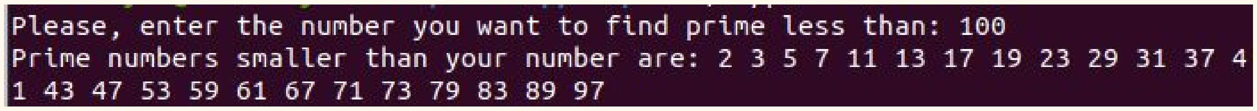
Firstly, to write the program in ARMv8 I converted for loops to while loops but I also need to take the consideration that I need to convert the input from the user from string to number. The address of the input buffer is passed as argument and loaded into X0. After that the conversion is done reusing the function "conversionAtoD '' from rosettacode and branching with Link branches, and keep it in register X0. X0-X7 are used to return values and pass parameters as well.
After that, we keep the number which is now converted in X21 using MOV which copies the value to a destination register. Then the number is compared with 0 and if it fulfills the condition from B.cond (EQ=equal) then we go to else which will print a message which was declared under ".data".
The same logic is used for other conditions, and if the condition for input>0 is fulfilled then it's going to call checkP function which will take input as parameter and its going to iterate through two while loops.
For our program, it is very important to check modulus but ARMv8 64 bit does not feature a modulus instruction. Therefore, we use UDIV (Unsigned divide) to divide between two numbers without remainder, and then MUL(multiply) divisor and quotient and SUB(subtract) to find the

modulus.
Then a conversion to decimal is performed reusing the function "conversion10" from rosetta code. Depending on the input, that is a correct input, a system call 'exit' is performed which exits the program otherwise it will tell the user to insert another input and it will do that until the input is >0.

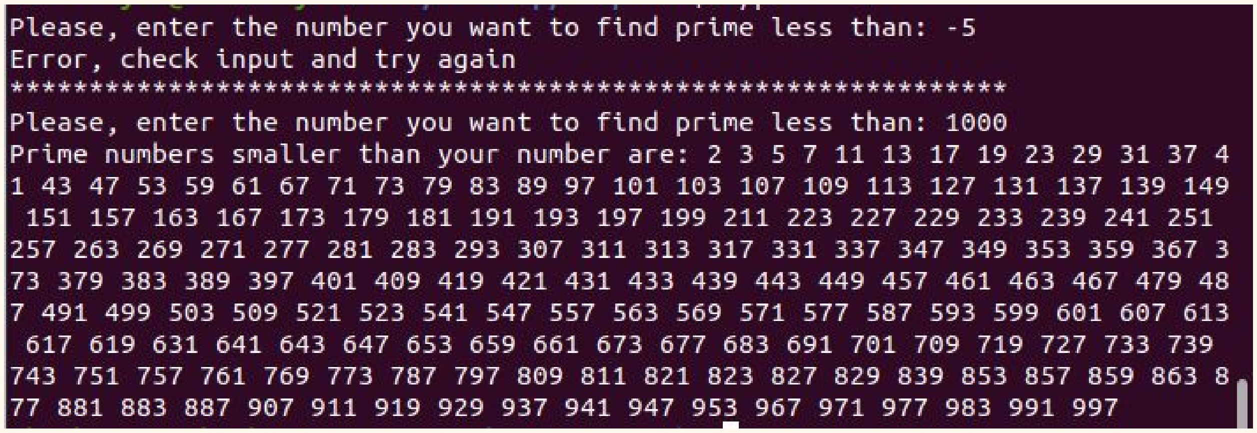
```
.include "includeConstantsARM64.inc"
.data
first:        .asciz "Please, enter the number you want to find prime less
than: "
output1:       .asciz "Prime numbers smaller than your number are: "
output2:      .asciz "Error, check input and try again"
newline:       .asciz " \n"
format:       .asciz " "
aster:        .asciz
"****************************************************************\n"

.equ BUFFERSIZE, 210

.bss
conversionBuffer: .skip 17
inputBuffer: .skip BUFFERSIZE

.text
.global main
main:

    LDR x0,=first
    BL printMess
    MOV x0, STDIN           // standard input stream
    LDR x1, =inputBuffer    // buffer address
    MOV X2, BUFFERSIZE      // the size of the  buffer
    MOV X8, READ            // request to read the data
    SVC #0                  // call system
    LDR X1, =inputBuffer    // buffer address
    MOV X2, #0              //
    STRB w1, [X1,X0]        // for input store the null byte in the end

      /* Get Input */

    LDR X0, =inputBuffer    // address of input buffer, passed as argument
    BL conversionAtoD       // convert to int using the function
conversionAtoD
```

```
    MOV X21, X0             // after convert, keep the number  in X21

      /* Check Input */

         CMP X21,0                  // will compare input with 0
         B.LE Error                // if it is less or equal to 0, then
goto Error
         LDR x0,=output1          // otherwise print the text of output1
       BL printMess            // call printMess
         BL testP               // call testP(input)
       LDR X0, =newline      //for printing a new line after the result
from testP
       BL printMess
       B Exit               //exit

     /* Error Message and call to main to re-ask for input */

Error:
      LDR x0,=output2         // print the text output2
      BL printMess           //call printMess function from files
      LDR X0, =newline      //to print a new line
      BL printMess         //call printMess function to print the new line
      LDR X0, =aster      //print the text in  asterisks
      BL printMess       // call printMess function to print asterisks
      B main            //go to main

    /* EXIT */

Exit:

      MOV X0, #0                // return 0 from main()
      MOV X8, EXIT             // exit system call
      SVC #0                  // do the system call

    /* checks for prime and prints numbers */

testP:
      MOV X28, X30
      MOV x18, #2              // assinging 2 to j

    /* first while loop */
firstwhile:
      CMP x18, X21             // While(j < input)
      B.GE exitfirstwhile     // go to exitfirstwhile
      MOV X2, 2              // assign 2 to i
      MOV X3, 1             // assign 1 to y

      /* second while loop */
secondwhile:
      CMP X2, x18              // While( i< j)
      B.GE exitsecondwhile
      UDIV X4, x18, X2            // j % i
      MUL  X4, X4, X2            //
      SUB  X5, x18, X4          // X5 = j % i
      CMP  X5, 0               // compare if remainder is zero
      B.NE iplusplus          //Branch if not equal to iplusplus
      MOV  X3, #0            // the value of y becomes 0\

        /* iplusplus */
iplusplus:
      ADD X2, X2, 1             // i=i+1
      B secondwhile            // go to secondwhile

        /* exit secondwhile */
exitsecondwhile:
      CMP X3, 1                  // compare y with 1
      B.NE jplusplus              //Branch if not equal to jplusplus
      MOV X0, x18                // first argument to conversion10
      LDR X1, =conversionBuffer // second argument by reference to
conversion10
      BL conversion10          // make the procedure call to convert to
decimal

      LDR X0, =conversionBuffer
      BL printMess            // print j
      LDR x0,=format         //format it with spaces
      BL printMess

    /* jplusplus */
jplusplus:
      ADD x18, x18, 1            // j= j + 1
      B firstwhile               // go to the firstwhile
   /* exitfirstwhile */
exitfirstwhile:
      MOV X30, X28
      BR LR   //branch and register

.include "includeARM64.inc"
```

[SCREENSHOTS]



```
Please, enter the number you want to find prime less than: 100
Prime numbers smaller than your number are: 2 3 5 7 11 13 17 19 23 29 31 37 4
1 43 47 53 59 61 67 71 73 79 83 89 97
```

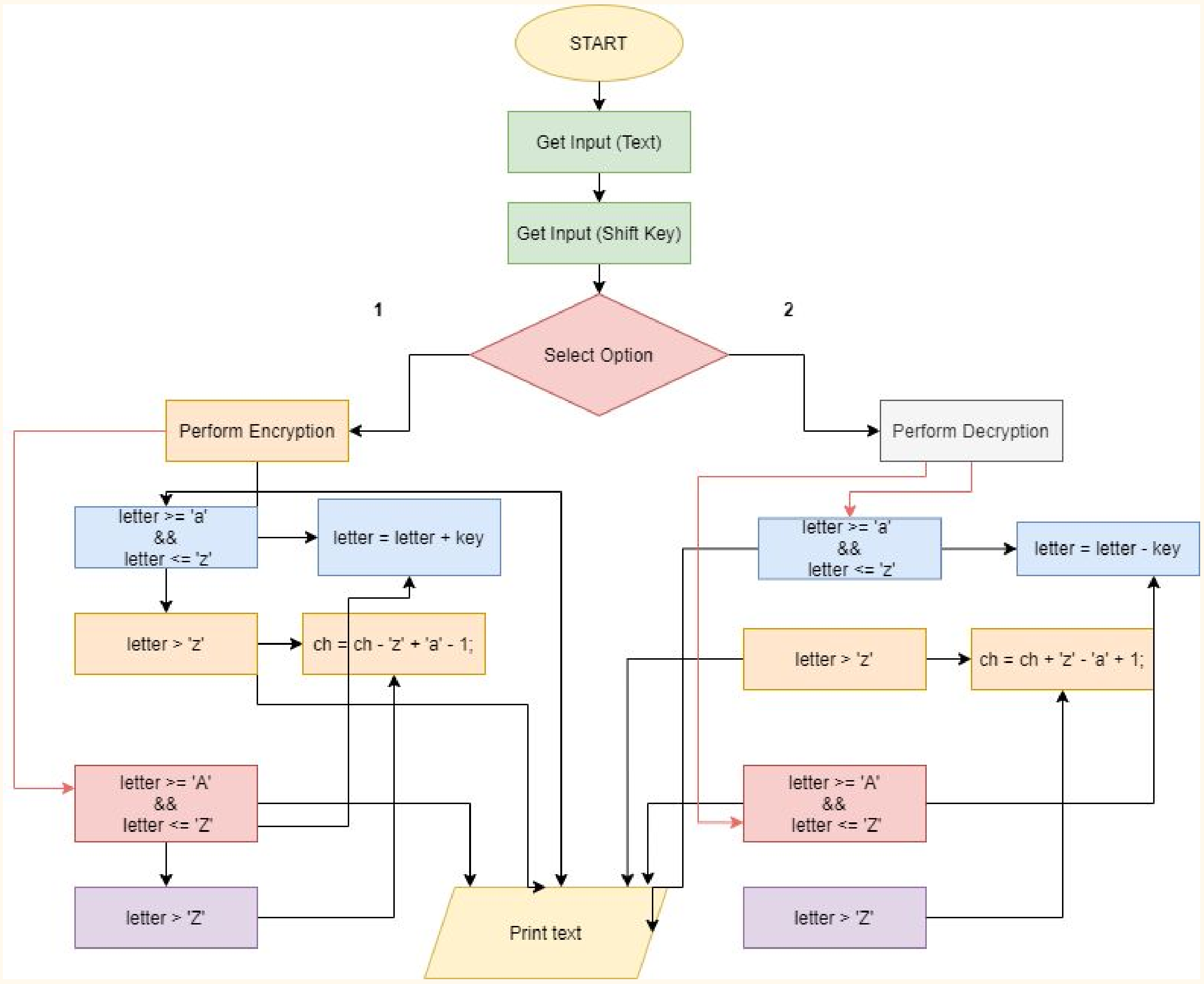


```
Please, enter the number you want to find prime less than: -5
Error, check input and try again
***************************************************************
Please, enter the number you want to find prime less than: 1000
Prime numbers smaller than your number are: 2 3 5 7 11 13 17 19 23 29 31 37 4
1 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149
 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251
257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 3
73 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 48
7 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613
 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739
743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 8
77 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

TASK II “Mystery”

The second task of the project is to write a program which will perform encryption and decryption (Primitive/Ceaser Cipher) of an inputted string from the user to shift the letters according to the English alphabet and the number of shifting is also provided by the user(input).

The Chart Flow for the algorithm is designed:



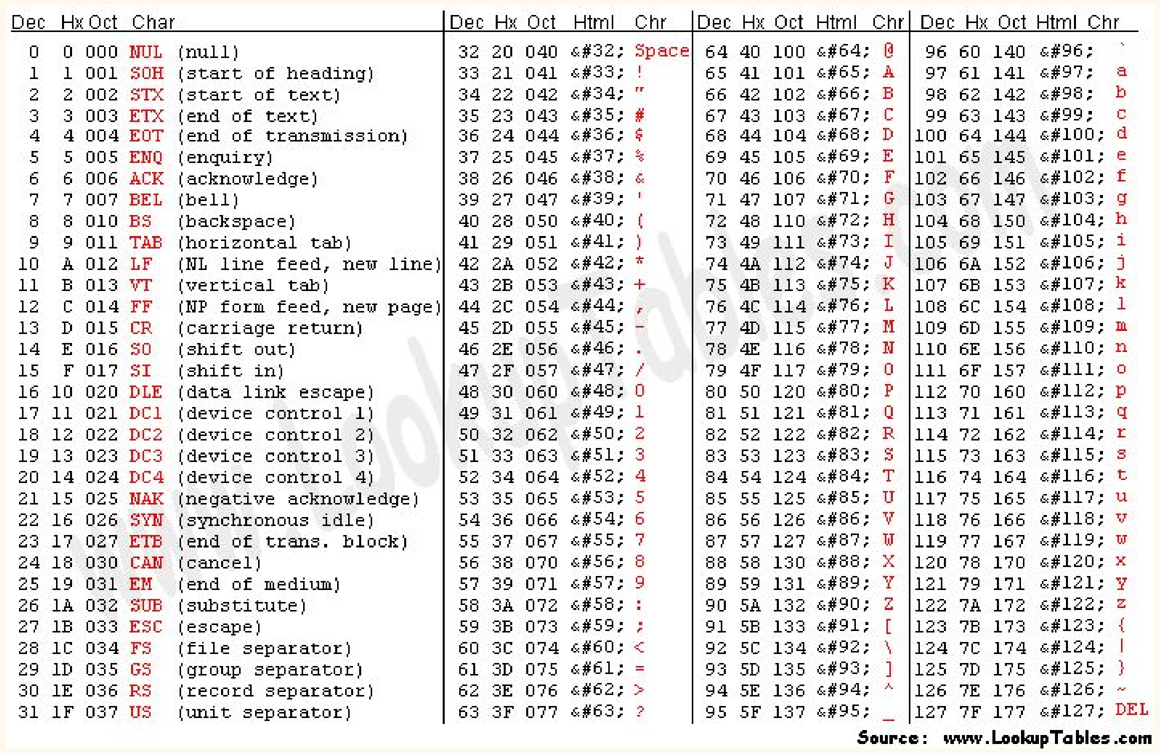
To implement this program in C++, I am going to follow the following logic. First, the user is asked by the program(cout) to enter a message which they want to encrypt/decrypt. After that, the user is then asked by the program(cout) to enter the shifting key (cin) and then it will provide a menu(cout) to enter 1 for encryption or 2 for decryption.

For the encryption implementation part:

We use an array to store the inputted text from the user.Using the '\0' terminating character of a string, we use the null character. We store the first value of input in another char type when iterating through a for loop. Comparing the letters then in using if statements, I added the key to the single letter if the letter was between a-z(inclusive). We know that a character is in fact just a number.

For example, the letter 'a' has the value x and 'z' the value y. Between x-y are the other letters. And by adding the key to the letter, that is char, we get another letter shifted by the key. However, if the letter is greater than 'z', that is after adding the key, we subtract the letter to 'z' add 'a' and subtract 1.

Same logic is also for Uppercase letters, which have different values than lowercase. Therefore, comparing letters between 'A' and 'Z' inclusive, we add the key (number) to the letter (char value). If the letter is greater than 'Z' we subtract 'Z' add 'A' and subtract 1. For numbers and other symbols, we don't perform anything because we want them to remain the same.



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Retrieved from: http://www.asciitable.com/ February 2021

However, char type can hold a maxim of 127. And when dealing with larger shift keys, according to our algorithm, we have to add it first but the maximum value a (signed char) can store is
127 [0111 1111] and since the key will always be positive, we can use unsigned char, which will store a maximum of **255 [1111 1111]**. This is done for encryption only, because the decryption implementation will work also with a signed char.

For the decryption implementation part:

If users input the number 2, they are going to perform decryption of their inputted text.

The same logic is used for separating the text to letters and comparing conditions. However, the thing that changes is the performance of addition with key from the encryption program becomes in fact subtraction. (letter - key).And if the letter is greater than 'z', that is after adding the key, we add the letter to 'z' subtract 'a' and add 1. Same thing for Uppercase letters. When the for loop has finished, the decrypted message is going to be printed/displayed in the screen for the user to see.

Implementation for C ++

```
#include<iostream>

using namespace std;

int main()
{
      char mystery[200];
      unsigned char firstL;
      int j;
      int shift;
      int option;

      cout << "Input the message you want to encrypt or decrypt ";
      cin.getline(mystery, 200);
      cout << "Enter shift key: ";
      cin >> shift;


      if (shift > 0 and shift < 26) {
      cout << "Enter 1 for encryption and 2 for decryption: ";
      cin >>option;
      if(option == 1){

      for(j = 0; mystery[j] != '\0'; ++j){

            firstL = mystery[j];

            if(firstL >= 'a' && firstL <= 'z'){
                  firstL = firstL + shift;

                  if(firstL > 'z'){
                        firstL = firstL - 'z' + 'a' - 1;
                  }

                  mystery[j] = firstL;
            }
            else if(firstL >= 'A' && firstL <= 'Z'){
                  firstL = firstL + shift;

                  if(firstL > 'Z'){
```

```
                  firstL = firstL - 'Z' + 'A' - 1;
            }

            mystery[j] = firstL;
      }

}
      cout << "Encrypted Message: " << mystery <<"\n";

}

else if(option == 2) {

for(j = 0; mystery[j] != '\0'; ++j){

      firstL = mystery[j];

      if(firstL >= 'a' && firstL <= 'z'){
            firstL = firstL - shift;

            if(firstL < 'a'){
                  firstL = firstL + 'z' - 'a' + 1;
            }

            mystery[j] = firstL;
      }
      else if(firstL >= 'A' && firstL <= 'Z'){
            firstL = firstL - shift;

            if(firstL <'A'){
                  firstL = firstL + 'Z' - 'A' + 1;
            }

            mystery[j] = firstL;
      }
}

cout << "Decrypted Message: " << mystery <<"\n";;}

}
else {
      cout << "ERROR, please insert a number from 1-25 ";
}
```

```
}
```

Writing the program in ARMv8 (AArch64):

Under .data I have declared all the messages that will be displayed to the user and under .bss (uninitialized data) I have included keys for decryption and choice when user inputs a number to select, as sort of an “option”, 1 for Encryption and 2 for decryption. We load the text to the register X0 using LDRX0,=menu and using the BL(branch with link) we go to the function printMess: (included file from rossetacode) which displays the string with size compute.

Since I am going to execute my program using the Linux terminal, I will use the Linux input console STDIN(Standard input). Then, I request to read the message using MOV X8, READ. and SVC #0 for the call system. Same thing with the key (again inputted from the user, hence the Linux Input console (STDIN)), load the key to register X1 and perform the request to read key and call system. Then, the option menu will be displayed on the screen where the user can choose 1 or 2. This is done by loading the message to register X0 and calling the function **printMess** with a Branch with link to the included file from rosettacode.

Same logic is used to get the user's choice. Then we load a byte from X1(choice of user), compare the input (register X0) to '1' using CMP and if it is indeed equal to 1 (B.EQ), it will call Encrypt procedure and if '2', it will call decrypt procedure. It will check whether choice and option are correct, if not then it will go to main and re ask for the input from the beginning.

Encryption and decryption procedures are quite similar with only minor changes, the logic is the same.

Then the program will iterate through loop1. I want the message[1] to be different from 0 so I use CMP w2, 0 to compare and if it is equal (B.EQ) the function exitloop is called. Same thing for other conditions, if the letter is between 'a' and 'z' then the key will be added to the ascii value and if it is not between 'a' and 'z', the uppercase function is called to check for uppercase letters 'A' to 'Z'. Same thing, if it is between 'A' and 'Z', numerically valued speaking, it will add the key to the letter value. The calculation (letter=letter-'z'+'a'-1) is going to be performed after checking conditions for (letter>'Z' and letter>'z'(different cases))using SUB, ADD AND ADD immediate which is not necessary to specify in AArch64. After that, the skip function is called using Branch B (unconditional) to save the value, increment and branch back to the loop1. This is going to be performed for each letter--which is indeed a value, inputted from the user.

The decryption procedure is called when the user inputs '2'. Likewise, the same logic is followed respectively. The only change is that for decryption, the key is subtracted from the letter rather than adding it. And the calculation performed for certain conditions(letter < 'a', letter <'A') becomes letter=letter+'z'-'a'+1.

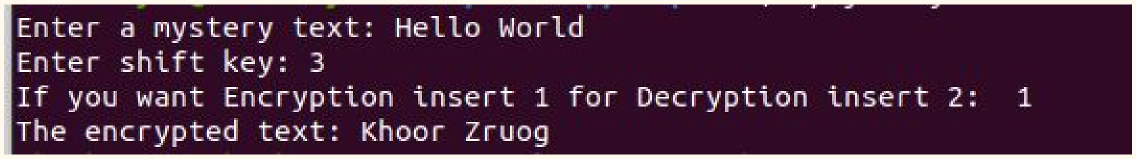
And then it prints the output text and the message (encrypted or decrypted, depending on user choice) using the BL printMess.

Finally, the exit is performed: returning 0 from main(), selecting system call 'exit' and then performing the system call using SVC #0.

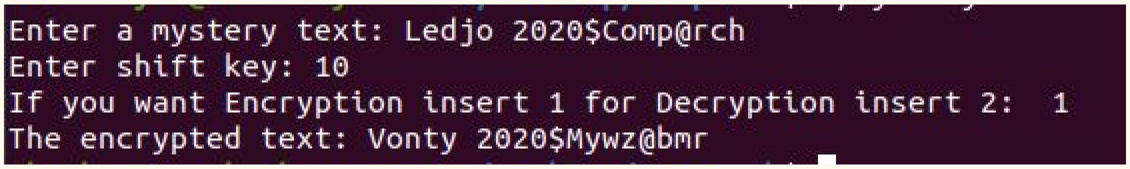
ARMv8 Code:

```
.include "includeConstantsARM64.inc"

.data
menu:   .asciz "If you want Encryption insert 1 for Decryption insert 2:  "
screen:   .asciz "Enter a mystery text: "
inputKey: .asciz "Enter shift key: "
firstoutput:   .asciz "The encrypted text: "
secondoutput:  .asciz "The decrypted text: "
error:  .asciz "* ERROR * Please choose from 1-25. \n"
error2:  .asciz "* ERROR * Please choose either 1 or 2 from menus \n"
.equ BUFFERSIZE, 210
.bss
messageBuffer: .skip BUFFERSIZE
firstkey:           .skip BUFFERSIZE
choice:        .skip 2

.text
.global main
main:
    LDR x0,=screen                           // print screen
    BL printMess

    MOV x0, STDIN                            // Linux input console
    LDR x1, =messageBuffer                   // buffer address
    MOV X2, BUFFERSIZE                       // buffer size
    MOV X8, READ                         // request to read message
    SVC #0                               // call system
    LDR X1, =messageBuffer
    MOV X2, #0
    STRB W2, [X1,X0]                     // store null byte at the end of
input string

    LDR x0, =inputKey
    BL  printMess

    MOV x0, STDIN                          // Linux input console
    LDR x1, =firstkey                    // buffer address
    MOV X2, BUFFERSIZE                   // buffer size
    MOV X8, READ                        // request to read key
    SVC #0                             // call system

    /* Checking the first key */
    LDR X0,=firstkey                 //load to register X0
    BL conversionAtoD               //convert to decimal using the function
from rosettacode
    MOV X21, X0                    //move to register X21
    CMP X21, 26                   // Compare the first key to 26
    B.GE errorkey                // if it is greater than or equal to 26 go
to errorkey
    CMP X21, 0                  // compare to 0
    B.LE errorkey              //if it is less than 0 or equal to 0 go to
errorkey

    /* after validating input, prints to the user the menu */
    LDR x0,=menu                       // print menu
    BL printMess

    MOV x0, STDIN                       // Linux input console
    LDR x1, =choice                     // buffer address
    MOV X2, 2                           // buffer size
    MOV X8, READ                        // request to read user choice
    SVC #0                              // call system

    /* after the user inputs a number (choice) we check if it is correct*/

        LDR X0,=choice                       //load the choice input to
register X0
        BL conversionAtoD                    //convert to decimal
        MOV X22, X0                         //move to register X22
        CMP X22, 3                       //compare the number(choice) with
3
        B.GE errorchoice                // if it is greater than or equal
to 3 go to errorchoice
        CMP X22, 0                     //compare the number(choice) to 0
        B.LE errorchoice              //if it is less than or equal to 0 go
to errorchoice
        CMP X22, 1                   //compare the number to 1
        B.EQ Encryption             //if the number(choice) is equal to 1
go to encryption
        CMP X22, 2                 //compare to 2
        B.EQ Decryption           //if number(choice) is equal to 2 go to
Decryption
        B Exit                   //exit


      /* Performs the encryption of a text given the shift key */
Encryption:


         LDR X0,=firstkey                     // store the first key to
register X0
         BL conversionAtoD                   //convert to decimal using the
function
         MOV X24, X0                        //copy register X0 (firstkey)
to register X24
         LDR x1, =messageBuffer            // get starting address of
message in X1

 firstloop:
         LDRB w2, [X1]                         // gets the first char
         CMP  w2,0                            // compare to 0
         B.EQ exitloop                       // if it is zero go to
exitloop
       CMP  w2, 'a'                         // compare the first letter to
'a'
       B.LT secondloop                      //if it is larger than a go to
secondloop
       CMP  w2, 'z'                       // compare to 'z'
       B.GT secondloop                    // if it is greater than go to
secondloop


         ADD  X2,X2,X24                     // add the key to the letter
       CMP  X2, 'z'                      // compare the result with z
       B.LE firstiterate                 // if it is less than or equal to
z go to firstiterate
       SUB  X2,X2,'z'                   //  letter = letter - 'z'
       ADD  X2,X2,'a'                  //letter= letter + 'a'
       SUB  X2,X2, 1                  // letter = letter - 1
       B firstiterate                        // go to firstiterate

 secondloop:
        CMP  w2, 'A'                        // compare letter(char value)
to 'A'
       B.LT firstiterate                  //if larger than or equal to 'A'
value go to firstiterate
       CMP  w2, 'Z'                      // compare it to Z value
       B.GT firstiterate                // goes to firstiterate if it is
greater than Z value

         ADD  X2,X2,X24                        // letter = letter +
shiftingkey
       CMP  X2, 'Z'                         // compare the result with Z
       B.LE firstiterate                  // if it is less than or equal
to Z go to firstiterate
       SUB  X2,X2,'Z'                   //  letter = letter - 'Z'
       ADD  X2,X2,'A'                  //   letter = letter + 'A'
       SUB  X2,X2, 1                 //    letter = letter - 1
       B firstiterate            //    go to firstiterate

 firstiterate:
        STRB w2,[x1]                      // store
        ADD  x1,x1,1                     // j = j+1
        B firstloop                     // branch to firstloop

 exitloop:
       LDR x0, =firstoutput                     // X0=firstoutput text
       BL  printMess                           // print the text using
printMess function
       LDR x0, =messageBuffer                 // get the encrypted message
       BL  printMess                         // print the encrypted
message
       B Exit                               // go to exit

/* Performs the decryption of a text given the shift key */

Decryption:
      LDR x0,=firstkey                        // get key value in x0
      BL conversionAtoD
      MOV X24, X0
      LDR x1, =messageBuffer             // get starting address of message
in x1


 thirdloop:
      LDRB w2, [x1]                          // get first letter
      CMP  w2,0                             // compare its value to 0
      B.EQ exitthirdloop                   // if it is equal to zero then
go to exitthirdloop
      CMP  w2, 'a'                          // compare letter with 'a'
      B.LT fourthloop                      // if it is less than 'a' go to
fourthloop
      CMP  w2, 'z'                        // compare to 'z'
      B.GT fourthloop                    // if it is greater than go to
fourthloop
      SUB  X2,X2,X24                    // letter = letter - shiftingkey
      CMP  X2, 'a'                     //compare to 'a'
      B.GE seconditerate              // if it is greater than or equal go
to seconditerate
      ADD  X2,X2,'z'                 // then letter = letter + 'z'
      SUB  X2,X2,'a'                // letter = letter - 'a'
      ADD  X2,X2, 1                // letter = letter + 1
      B seconditerate                     // go to seconditerate

 fourthloop:
      CMP  w2, 'A'                          // compare letter with 'A' char
value
      B.LT seconditerate                   // if it is less than A go to
seconditerate
      CMP  w2, 'Z'                         //  compare to Z
      B.GT seconditerate                  // if it is greater than go to
seconditerate

      SUB  X2,X2,X24                       // letter = letter - shiftkey
      CMP  X2, 'A'                        // compare letter to 'A'
      B.GE seconditerate                 // go to skip 1 if greater than or
equal to A
      ADD  X2,X2,'Z'                    // then letter = letter + 'Z'
      SUB  X2,X2,'A'                   // letter = letter - 'A'
      ADD  X2,X2, 1                   // letter = letter + 1
      B seconditerate                              // branch to seconditerate

 seconditerate:
     STRB w2,[x1]                                // store register byte
     ADD  x1,x1,1                               // j = j+1
     B thirdloop                               // branch to thirdloop

 exitthirdloop:
    LDR x0, =secondoutput                      // X0 = secondoutput text
    BL  printMess                             // print using the function
    LDR x0, =messageBuffer                   // print the message
    BL  printMess                           // using the function printMess



Exit:

    MOV X0, #0                                 // return 0 from main()
    MOV X8, EXIT                               // X8= exit system call
    SVC #0                                     // do the system call

errorkey:
    LDR x0, =error                             // print output text
    BL  printMess
    B main                                    // repeat
errorchoice:
    LDR x0, =error2                             // print output text
    BL  printMess
    B main                                     // repeat

.include "includeARM64.inc"
```

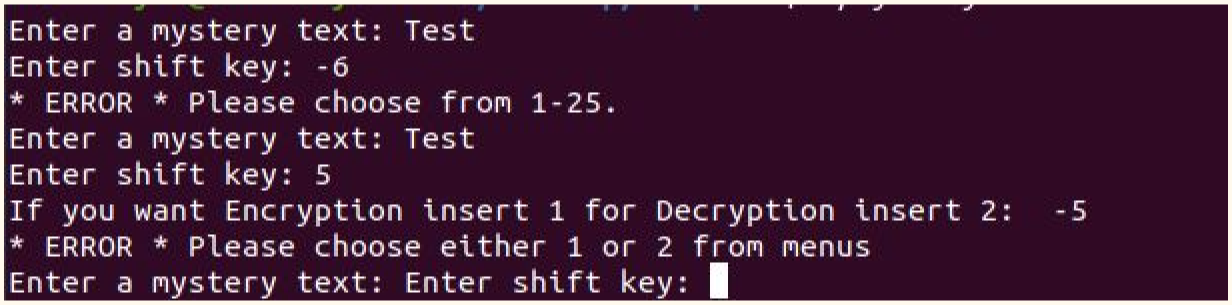
[SCREENSHOTS]



```
Enter a mystery text: Hello World
Enter shift key: 3
If you want Encryption insert 1 for Decryption insert 2:  1
The encrypted text: Khoor Zruog
```



```
Enter a mystery text: Ledjo 2020$Comp@rch
Enter shift key: 10
If you want Encryption insert 1 for Decryption insert 2:  1
The encrypted text: Vonty 2020$Mywz@bmr
```



```
Enter a mystery text: Test
Enter shift key: -6
* ERROR * Please choose from 1-25.
Enter a mystery text: Test
Enter shift key: 5
If you want Encryption insert 1 for Decryption insert 2:  -5
* ERROR * Please choose either 1 or 2 from menus
Enter a mystery text: Enter shift key:
```

CONCLUSION

Finally, researching about AArch64 was not easy but it was immensely interesting and it brought life to the course content and knowledge I acquired during the semester. For the most part, some parts of the code are inspired and taken from the tutorial given in class and rosetta code examples.

SOURCES

Pyeatt, L., & Ughetta, W. *ARM 64-Bit Assembly Language*

- Course Website Comp Arch UNYT 2020-2021

Category:AArch64 Assembly - Rosetta Code. (2021). Retrieved February 2021, from http://rosettacode.org/wiki/Category:AArch64_Assembly

A Guide to ARM64 / AArch64 Assembly on Linux with Shellcodes and Cryptography. (2021). Retrieved February 2021, from https://modexp.wordpress.com/2018/10/30/arm64-assembly/#set