RISC-V Assembly Project for the Computer Architecture Course – A.A. 2019/2020 –

Encrypted Messages

3rd version of the document, updated on 8/4/2020. Changes from previous versions

- Sharing on MOODLE
- Clarifications (expected input ciphers, exam mode, plaintext size, occurrence cipher, newline)
- Change cipher text to work only with characters c such that 32 ≤ cod (c) ≤ 127

Cryptography and Encryption

Codes are a way to alter a text message to hide its original meaning. This allows you to encrypt strings and generic data, to hide their meaning from third parties who might intercept the encrypted message, but do not have the key to decode it. One of the most famous examples is the Enigma machine (I suggest the vision of "The Imitation Game", for those who have not seen it!), an electromechanical device able to encrypt and decipher messages that was widely used by the service of the German armed forces during the Nazi period and the Second World War. In many cases and usually they require one **keyword** to be interpreted. Ciphers are algorithms applied to a message that hide or encrypt the information transmitted. These ciphers are then **reversed** in order to translate or decipher the message.

Enigma Machine
G version

For example, the plain text AMO AssEMbLY could be modified through an encryption function that replaces each letter with the next in the alphabet, obtaining the encrypted word (cyphertext) BNP BttFNcMZ. Only those who know a specific decryption function can interpret the text. In this case, the de-encryption function replaces each letter with the one preceding it in the alphabet, obtaining again AMO ASSEMbLY.

More strictly, given a plaintext pt, an encryption function fc, a decryption function fd, we have

$$\text{cyphertext ct} = fc(pt), \qquad pt = fd(ct)$$

In general, using a set of n encryption functions FC = {fc_1, ..., fc_n} and decryption functions FD = {fd_1, ..., fd_n}, *where each fd_i decrypts the message encrypted by the function fc_i*, we have

$$pt = fd_n(fd_{n-1}(\ldots fd_1(fc_1(fc_2(\ldots fc_n(pt))))))$$

In other words, by **sequentially applying the encryption functions** fc_n, fc_{n-1}, ... fc_1, you get the plaintext back by applying the **decryption functions in the reverse order** fd_1, fd_2, ... fd_n

Some Encryption Codes

Substitution Cipher / Caesar Cipher

(From Wikipedia) Caesar cipher is one of the oldest cryptographic algorithms historically traced. It is a monoalphabetic substitution cipher in which **each letter of the plaintext is replaced in the ciphertext by the letter which is found a certain number of places later in the alphabet.** These types of ciphers are also called substitution ciphers or shift ciphers because of the way they operate: the substitution takes place letter by letter, scrolling the text from beginning to end. A possible translation could be the following: the standard ASCII code on 8 bits of each character of the text message is modified by adding an integer constant K.

For the purposes of our project, the substitution is performed only for uppercase and lowercase letters (A-Z and a-z), which are transformed only in other letters. The other symbols (numbers, spaces, special characters) are left unchanged. Uppercase and lowercase are preserved.

Example

See the previous page with the plaintextAMO AssEMbLY (K=1)
In addition, assuming K = -2, we obtain cod(A) = Y, cod(d) = b, cod(@) = @, cod(z) = x, cod(1) = 1

Block Cipher

(From Wikipedia) A block cipher algorithm is made up of two parts, one that encrypts, using m characters for the block to be encrypted and k characters for the key to be used during encryption, returning m exit characters for the cyphertext. A simple version for block cipher can be understood as follows: **the word is partitioned into nb blocks**, obtained as nb = m/k rounded to the whole integer. Each block in B = {b_1, b_2, ... b_{nb}} contains at most k consecutive elements of the string to be encrypted. **Each element of each block is encrypted by adding the ASCII encoding of a character of the key to the ASCII encoding of the block of characters** as follows

$$\text{For each } b_i \text{ in B } (1 \le i \le nb),\ cb_i = cod(b_{ij}) + cod(key_j),\ 1 \le j \le k,$$

getting the cyphertext ct = {cb_1, cb_2, ... cb_{nb}}, composed of nb encrypted blocks. *All characters in the starting string are enrypted with this encoding.* Given the restriction for the input alphabet of characters c with $32 \le cod(c) \le 127$ for both plaintext and key, the encryption becomes:

$$\text{For each } b_i \text{ in B } (1 \le i \le nb),\ cb_i = \{[(cod(b_{ij}) - 32) + (cod(key_j) - 32)]\ \%\ 96\} + 32,\ 1 \le j \le k,$$

Example

pt = LAUREATO_1, key = OLE

Calculate Cod(O) = 79, Cod(L) = 76, Cod(E) = 69 by consulting the ASCII table and then from each Cod(ct) you will find the corresponding character in the ASCII table to obtain the cyptertext ct.

Pt	L	A	U	R	E	A	T	O	_	1
Cod(pt)	76	65	85	82	69	65	84	79	95	49
Key	O	L	E	O	L	E	O	L	E	O
Cod(key)	79	76	69	79	76	69	79	76	69	79
Cod(ct)	**123**	**109**	**122**	**33**	**113**	**102**	**35**	**123**	**36**	**96**
ct	**{**	**m**	**z**	**!**	**q**	**f**	**#**	**{**	**$**	**'**

Occurrence Encryption

Starting with the first character of the plaintext (at position 1), **the message is encrypted as a sequence of strings separated by exactly 1 space** (ASCII 32) in which each string has the form “*x-p1-...-pk*”, where x is the first occurrence of each character in the message, $p_1 ... p_k$ are the k positions where the character *x* appears in the message (with $p_1 < ... < p_k$), and where in each position is preceded by the separator character '-' (to distinguish the elements of the sequence of positions).

Note:

- there is no predetermined order for the letters once the string has been encoded
- the encoding uses two separators: the space (ASCII 32) and the hyphen (ASCII 45). What is between two dashes must always be a number indicating the occurrence of a letter in the base string, while what follows the space is always the character of reference, except in the case where the character of reference is the space itself (see example below).
- the cypertext obtained with this encoding generally has a greater length than the starting plaintext

Example

Pt = “sempio di messaggio criptato -1”

The encryption with this algorithm will produce a cyphertext ct = “e-2-12 s-1-13-14 m-3-11 p-4-24 i-5-9-18-23 o-6-19-28 -7-10-20-29 d-8 a-15-26 g-16-17 c-21 r-22 t-25-27 --30 1-31”.

- In the string “ -7-10-20-29” the encoding character is the space (‘ ‘, ASCII decimal 32), which appears in positions 7, 10, 20 e 29 of the message.
- In the string “--30” the encoding character is ‘-‘ (the second character ‘-‘ is the separator character between the elements of the sequence), which appears in position 30 of the messagge.
- In the string “1-31” the encoding character is ‘1’, which appears in position 31 of the messagge..
- In the memory, the first part of the cyphertext “e-2-12 ” must appear on 7 bytes, one per character, such as cod(e), cod(-), cod(2), cod(-), cod(1), cod(2), cod( ) -> 101, 45, 50, 45, 49, 50, 32

Dictionary

Each possible ASCII symbol is mapped with another ASCII symbol according to a certain function, which we report below defined by case.

- If the character *ci* is a lowercase letter (min), it is replaced with the uppercase equivalent of the alphabet in reverse order es. Z = ct(a), A = ct(z).
- If the character *ci* is an uppercase letter (mai), it is replaced with the lowercase equivalent of the alphabet in reverse order es. z = ct(A), y = ct(B), a =ct(Z).
- If the character *ci* is a number (num), ct(ci) = ASCII(cod(9)-num)
- In all other cases (sym), *ci* remains unchanged, that is ct(ci) = ci

Example

Pt = myStr0ng P4ssW_

Pt	m	y	S	t	r	0	n	g		P	4	s	s	W	_
Type ci	min	min	mai	min	min	num	min	min	sym	mai	num	min	min	mai	sym
ct	**N**	**B**	**h**	**G**	**I**	**9**	**M**	**T**		**k**	**5**	**H**	**H**	**d**	**_**

Note: Transformation of 0: ASCII(cod(9)-0) = ASCII(57-0) = ASCII(57) = 9,

Other Encryptions

2 other possible ciphers, based on modifications of the character string, are the following.

- **Inversion**: the cyphertext is represented by the inverted string e.g. pt = BUONANOTTE, ct = ETTONANOUB. Note how in case of palindrome words, pt = ct.
- **Substitution cipher differentiating vowels-consonants-symbols**: an extension of the substitution where instead of having a single value of k there are 4 values: kv (for vowels), kc (for consonants), kn (for numbers), and ks (for symbols, that is spaces, dashes etc).
  You can identify the following circular alphabets (declarable as strings in the .data field, to make it easier to scroll):
    - Dizv - Vowels: A E I O U Y A E ...
    - Dizc - Consonants: B C D F G H J K L M N P Q R S T V W X Z B C D ...
    - Dizn - Numbers: 0 1 2 3 4 5 6 7 8 9 0 1 2 ...
    - Dizs - Symbols in the table below.

ASCII Code	32	33	35	36	37	38	39	42	43	44	45	46	47	58	59	60	61	62	63	64	32
Symbol		!	#	$	%	&	‘	*	+	,	-	.	/	:	;	<	=	>	?	@	

The encryption works like the substitution, but **using the appropriate k є {kv, kc, kn, ks} according to the class** {vowel, consonant, number, symbol} of each character of the plaintext, **using the correct circular alphabet, and inverting uppercase and lowercase for vowels and consonants**. For example, the encoding of a vowel can only be a vowel. *So if I have the vowel A, and kc = -2, I will get cyphertext u (lowercase, uppercase and lowercase are inverted each time)*. Symbols that do not appear in any of the 4 categories above remain unchanged (see _ in the example below).

Pt = myStr0ng P4ssW_

Kv = -3, Kc = 2, Kn = 12, ks = 10

Pt	**m**	**y**	**S**	**t**	**r**	**0**	**n**	**g**		**P**	**4**	**s**	**s**	**W**	**_**
Type (c, v, n, s)	c	v	c	c	c	n	c	c	s	c	n	c	c	c	same
ct	**P**	**I**	**v**	**W**	**T**	**2**	**Q**	**J**	**-**	**r**	**6**	**V**	**V**	**z**	**_**

Assembly Project

Building on top of what said above, the AE 19/20 project is about designing and writing a RISC-V assembly code that simulates certain functions of encryption and decryption of a text message, interpreted as an ASCII character sequence.

In particular, the program must allow you to **encrypt** and **decrypt** a text message (plaintext) provided by the user as a string type variable *myplaintext* (.string in RIPES). Different encryption functions, and the consequent decryption functions, will have to be implemented as follows.

A. Substitution Cipher, configurable via sostK variable, which indicates the alphabetical shift.
B. Block Cipher, with the key being a string to be considered as a blocKey variable
C. Occurrence Cipher
D. Dictionary

if the group consists of 2 or 3 people, the group will also have to implement some ***addictional*** *encryption and decryption functions*, as described below. For groups of **two people**, you can choose **one of the two**, while the groups of **three people** will have to make **both**.

E. Inversion
F. Differentiating Substitution, configurable through variables kv, kc, kn, ks, which indicate the shifts for vowels, consonants, numbers, symbols.

In addition to the myplaintext variable (**maximum size 100 characters, characters c that can only be such that 32 ≤ cod(c) ≤ 127 to avoid special ASCII characters**), the program requires additional input that specifies *how to apply ciphers.* That variable *mycypher* is a string S = "$S_1 \ldots S_n$" consisting of a maximum of 5 characters (therefore with $1 \leq n \leq 5$), in which each character S_i (with $1 \leq i \leq n$) corresponds to one of the characters 'A', 'B', 'C', 'D', 'E', 'F', and identifies the *i*-th cipher to apply to the message. **The order of the ciphers is therefore estabilished by the order in which the characters appear in the string. Furthermore, each cipher returns a cyphertext which is a sequence of characters c such that 32 ≤ cod(c) ≤ 127**
As an example, some possible keywords are reported: "C", or "AEC", or "DEDD", ... For example, the encryption of the text message with the keyword "AEC" will determine the application of the algorithm A, then of the algorithm E (on the message already encrypted with A) and finally of the algorithm C (on the message already encrypted first with A and then with E).

The program will have to consider the environment variables myplaintext e mycypher, in addition to the various message parameters to be encrypted, and produce in output on video, the various cypthertexts obtained after the application of each single encryption step, separated by a newline. Similarly, decryption functions will have to be applied *starting from the previously encrypted message* in reverse order from the encryption. **The last message printed on screen must correspond to the starting plaintext**.

General Notes of the Project

- "Newline" has ASCII code 10, and can be used to differentiate the 3 outputs of the program
- RIPES does not handle well input strings with size divisible by 4 (does not put an end to the string). So, use an input that is always a string long n characters, where *n is not divisible by 4*.