



Computer Architecture HW1

TA: 曾品珊

Due: Apr. 05, 2024 (11:59 p.m.)



Outline

- ◆ Jupiter: RISC-V Simulator
- ◆ HW1-1 Recursive Function
- ◆ HW1-2 Encryption
- ◆ Report
- ◆ Rules
- ◆ Submission



Jupiter: RISC-V Simulator

- ◆ An open source RISC-V assembler and runtime simulator
- ◆ Download here: <https://github.com/andrescv/Jupiter>

📖 Readme
📄 GPL-3.0 license
📄 Code of conduct
★ 376 stars
👁 11 watching
🍴 34 forks

Can only simulate 32-bit registers
Remember to use the right instruction!!
Ex. lw, sw...

Jupiter-3.1-linux.zip	40 MB
Jupiter-3.1-mac.zip	35.7 MB
Jupiter-3.1-win.zip	37.6 MB

Releases 9

📦 Jupiter v3.1 Latest
on Sep 4, 2019

+ 8 releases

Running Jupiter on Linux or macOS

```
./image/bin/jupiter # for GUI mode  
./image/bin/jupiter [options] <files> # for CLI mode
```

Running Jupiter on Windows

```
image\bin\jupiter # for GUI mode  
image\bin\jupiter [options] <files> # for CLI mode
```

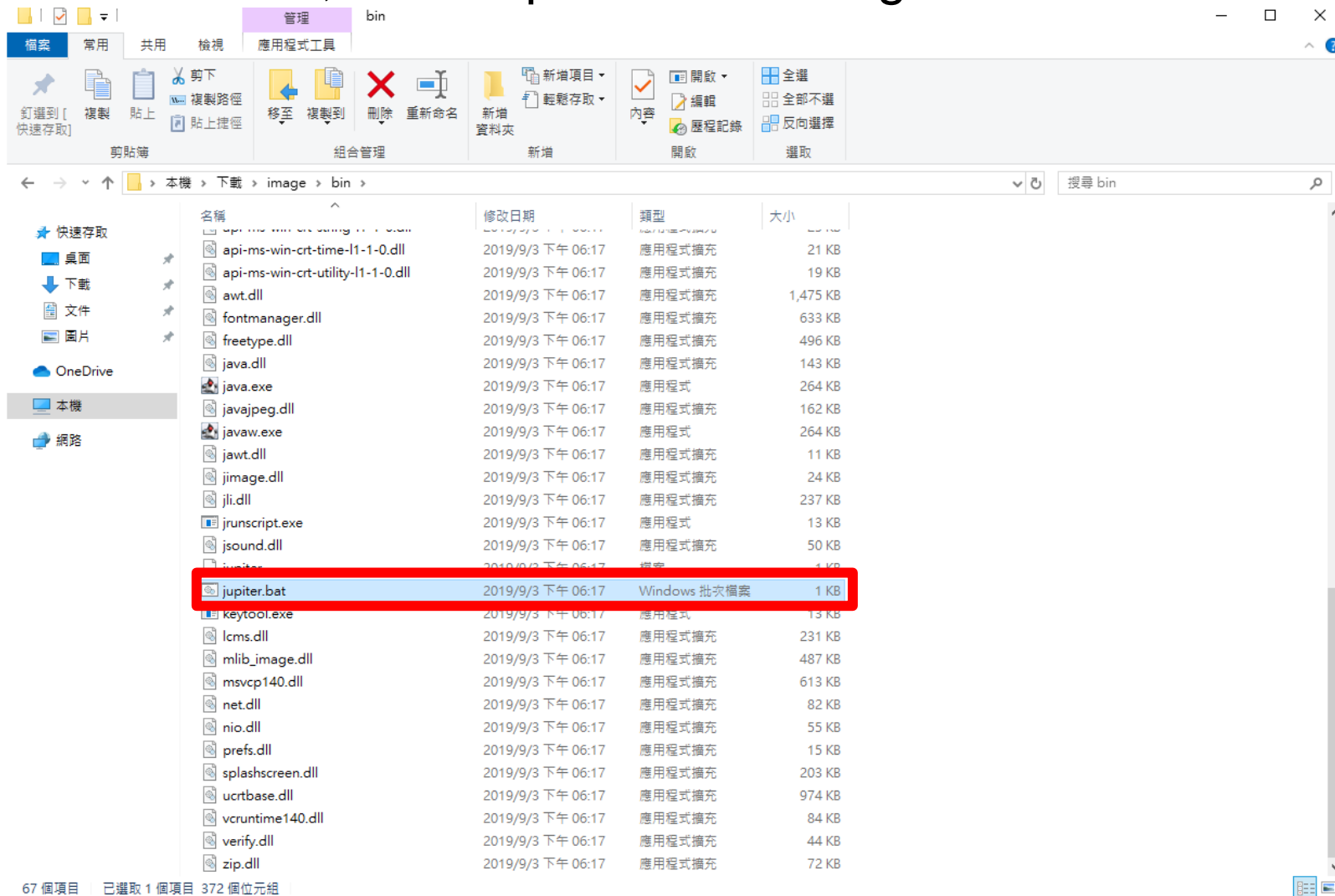
Packages

No packages published



Jupiter: RISC-V Simulator (Cont.)

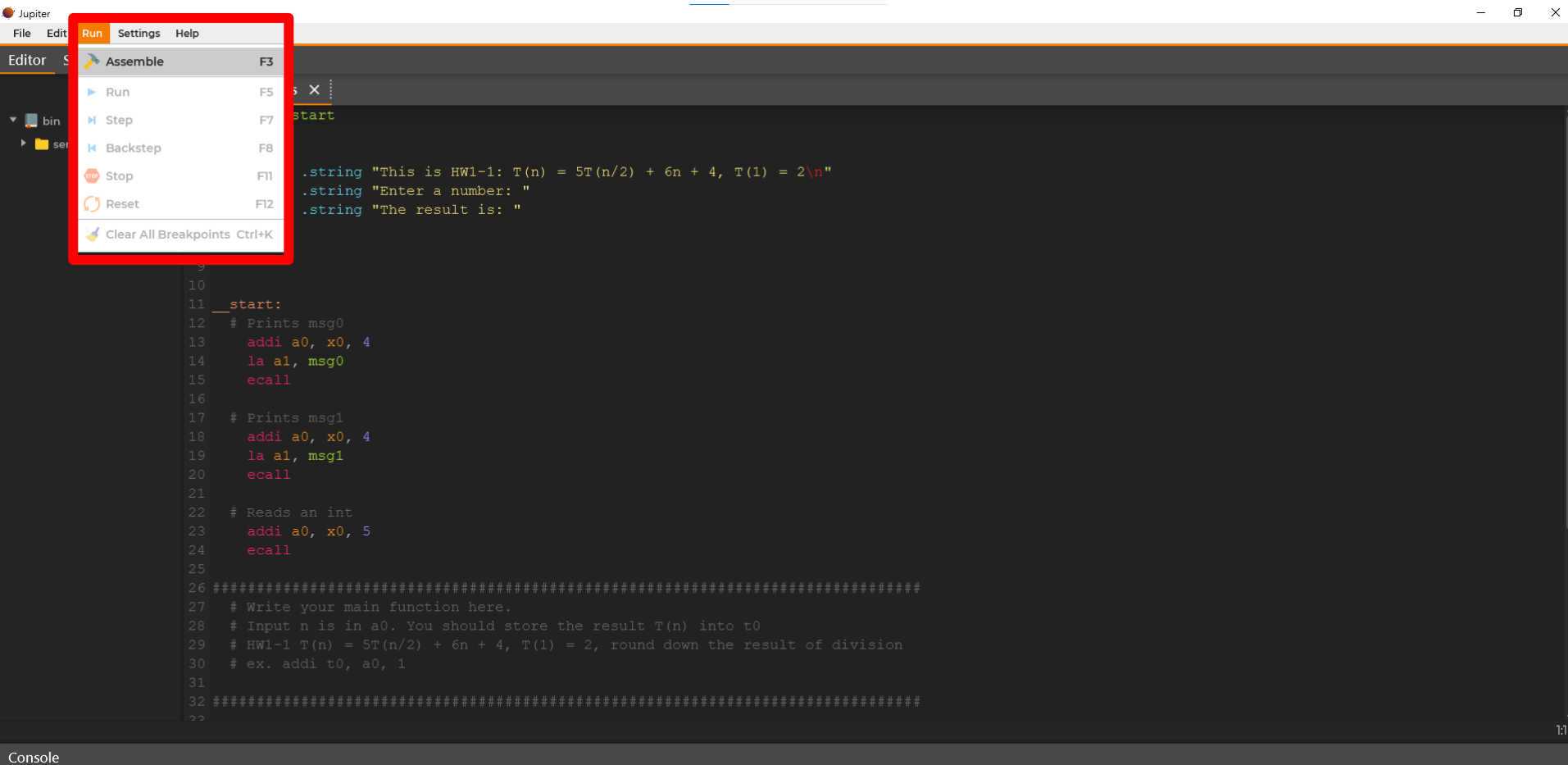
- ◆ To launch it, click Jupiter.bat in image/bin





GUI of Jupiter

- ◆ To run the code
 - ◆ Click Run -> Assemble





GUI of Jupiter (Cont.)

Monitor register and memory here

Run / Step / Backstep / Stop

Jupiter

File Edit Run Settings Help

Editor Simulator

▶ ⏸ ⏪ ⏹ ↺ 📄 🖨

Bkpt	Address	Machine Code	Basic Code	Source Code
<input type="checkbox"/>	0x00010000	0x00000317	auipc x6, 0	auipc x6, 0
<input type="checkbox"/>	0x00010004	0x00030067	jair x0, x6, 8	jair x0, x6, 8
<input type="checkbox"/>	0x00010008	0x00400513	addi x10, x0, 4	addi a0, x0, 4
<input type="checkbox"/>	0x0001000c	0x00000597	auipc x11, 0	la a1, msg0
<input type="checkbox"/>	0x00010010	0x04858593	addi x11, x11, 72	la a1, msg0
<input type="checkbox"/>	0x00010014	0x00000073	ecall	ecall
<input type="checkbox"/>	0x00010018	0x00400513	addi x10, x0, 4	addi a0, x0, 4
<input type="checkbox"/>	0x0001001c	0x00000597	auipc x11, 0	la a1, msg1
<input type="checkbox"/>	0x00010020	0x06b58593	addi x11, x11, 107	la a1, msg1
<input type="checkbox"/>	0x00010024	0x00000073	ecall	ecall
<input type="checkbox"/>	0x00010028	0x00500513	addi x10, x0, 5	addi a0, x0, 5
<input type="checkbox"/>	0x0001002c	0x00000073	ecall	ecall
<input type="checkbox"/>	0x00010030	0x00400513	addi x10, x0, 4	addi a0, x0, 4
<input type="checkbox"/>	0x00010034	0x00000597	auipc x11, 0	la a1, msg2
<input type="checkbox"/>	0x00010038	0x06458593	addi x11, x11, 100	la a1, msg2

Registers Memory Cache ST

Address	+3	+2	+1	+0
0x0001007c	29	31	28	54
0x00010078	20	2c	34	20
0x00010074	2b	20	6e	36
0x00010070	20	2b	20	29
0x0001006c	32	2f	6e	28
0x00010068	54	35	20	3d
0x00010064	20	29	6e	28
0x00010060	54	20	3a	31
0x0001005c	2d	31	57	48
0x00010058	20	73	69	20
0x00010054	73	69	68	54
0x00010050	00	00	00	73
0x0001004c	00	a0	05	13
0x00010048	00	00	00	73

text

Console

This is HW1-1: $T(n) = 5T(n/2) + 6n + 4, T(1) = 2$
Enter a number:

Type input here



GUI of Jupiter (Cont.)

◆ Memory type: text, data, stack, heap

Jupiter

File Edit Run Settings Help

Editor Simulator

▶ ⏮ ⏪ ⏩ ⏭ ⏮ ⏪ ⏩ ⏭

▶ ⏮ ⏪ ⏩ ⏭

Bkpt	Address	Machine Code	Basic Code	Source Code
<input type="checkbox"/>	0x00010000	0x00000317	auipc x6, 0	auipc x6, 0
<input type="checkbox"/>	0x00010004	0x00830067	jalr x0, x6, 8	jalr x0, x6, 8
<input type="checkbox"/>	0x00010008	0x00400513	addi x10, x0, 4	addi a0, x0, 4
<input type="checkbox"/>	0x0001000c	0x00000597	auipc x11, 0	la a1, msg0
<input type="checkbox"/>	0x00010010	0x04858593	addi x11, x11, 72	la a1, msg0
<input type="checkbox"/>	0x00010014	0x00000073	ecall	ecall
<input type="checkbox"/>	0x00010018	0x00400513	addi x10, x0, 4	addi a0, x0, 4
<input type="checkbox"/>	0x0001001c	0x00000597	auipc x11, 0	la a1, msg1
<input type="checkbox"/>	0x00010020	0x06b58593	addi x11, x11, 107	la a1, msg1
<input type="checkbox"/>	0x00010024	0x00000073	ecall	ecall
<input type="checkbox"/>	0x00010028	0x00500513	addi x10, x0, 5	addi a0, x0, 5
<input type="checkbox"/>	0x0001002c	0x00000073	ecall	ecall
<input type="checkbox"/>	0x00010030	0x00400513	addi x10, x0, 4	addi a0, x0, 4
<input type="checkbox"/>	0x00010034	0x00000597	auipc x11, 0	la a1, msg2
<input type="checkbox"/>	0x00010038	0x06458593	addi x11, x11, 100	la a1, msg2

Registers Memory Cache ST

Address	+3	+2	+1	+0
0x0001007c	29	31	28	54
0x00010078	20	2c	34	20
0x00010074	2b	20	6e	36
0x00010070	20	2b	20	29
0x0001006c	32	2f	6e	28
0x00010068	54	35	20	3d
0x00010064	20	29	6e	28
0x00010060	54	20	3a	31
0x0001005c	2d	31	57	48
0x00010058	20	73	69	20
0x00010054	73	69	68	54
0x00010050	00	00	00	73
0x0001004c	00	a0	05	13
0x00010048	00	00	00	73

Console

This is HW1-1: $T(n) = 5T(n/2) + 6n + 4$, $T(1) = 2$
Enter a number:

text

text

data

stack

heap



HW1-1 Recursive Function

- ◆ Input

- ◆ A positive integer n

- ◆ Output $T(n)$

- ◆
$$T(n) = \begin{cases} 3T\left(\lfloor \frac{n}{4} \rfloor\right) + 10n + 3, & \text{if } n \geq 4 \\ 3, & n = 1, 2, 3 \end{cases}$$

- ◆ e.g., $T(6) = 72$, $T(23) = 419$

- ◆ Round down the result of division to an integer

- ◆ e.g., $5/4 = 1$, $9/4 = 2$

- ◆ Implement with recursive function only



Template of Homework 1-1

- ◆ The input stores in a0 (i.e., x10)
- ◆ The output should be stored into t0 (i.e., x5)
- ◆ Write your code in the red frame
 - ◆ You may use a function and write a jump to execute it.

```

1  .globl __start
2
3  .rodata
4      msg0: .string "This is HW1-1: T(n) = 3T(n/4) + 10n + 3, T(1)=T(2)=T(3) = 3\n"
5      msg1: .string "Enter a number: "
6      msg2: .string "The result is: "
7
8  .text
9
10
11  __start:
12      # Prints msg0
13      addi a0, x0, 4
14      la a1, msg0
15      ecall
16
17      # Prints msg1
18      addi a0, x0, 4
19      la a1, msg1
20      ecall
21
22      # Reads an int
23      addi a0, x0, 5
24      ecall
25
26      #####
27      # Write your main function here.
28      # Input n is in a0. You should store the result T(n) into t0
29      # HW1-1 T(n) = 3T(n/4) + 10n + 3, T(1)=T(2)=T(3) = 3, round down the result of division
30      # ex. addi t0, a0, 1
31      #####
32
33
34  result:
35      # Prints msg2
36      addi a0, x0, 4
37      la a1, msg2
38      ecall
39
40      # Prints the result in t0
41      addi a0, x0, 1
42      add a1, x0, t0
43      ecall
44
45      # Ends the program with status code 0
46      addi a0, x0, 10
47      ecall

```



You May Ask

- ◆ What is a0 and t0
- ◆ It is just a mnemonic
- ◆ In this homework, you can use any registers you want

Mnemonic	Number	Value
gp	x3	0x10008000
tp	x4	0x00000000
t0	x5	0x00000000
t1	x6	0x00000000
t2	x7	0x00000000
s0	x8	0x00000000
s1	x9	0x00000000
a0	x10	0x00000000
a1	x11	0x00000000
a2	x12	0x00000000
a3	x13	0x00000000
a4	x14	0x00000000
a5	x15	0x00000000
a6	x16	0x00000000



HW1-2 Encryption

- ◆ a-z: use Caesar cipher
 - ◆ Case1: shift = 3
 - Plaintext: abcdefghijklmnopqrstuvwxyz
 - Ciphertext: defghijklmnopqrstuvwxyzabc
 - ◆ Case2: shift = -3
 - Plaintext: abcdefghijklmnopqrstuvwxyz
 - Ciphertext: xyzabcdefghijklmnopqrstuvw
- ◆ Space: encode to incremental integers starting from 0
 - ◆ Plaintext is “abc and cde”
 - ◆ Ciphertext is “def0dqq1fgh”



HW1-2 Encryption

◆ Input

- ◆ Inputs are only lower-case alphabets and spaces
- ◆ The count of spaces will not exceed ten
- ◆ Shift can be -12 ~ 13
- ◆ Plaintext will end with '\n' (decimal 10)

◆ Output

- ◆ You **must** store the ciphertext in memory address from **66048(0x10200)**
- ◆ Use "j print_char" when your code is finished



HW1-2 Encryption (Cont.)

- ◆ Character are stored as ASCII code
- ◆ A character is 8 bits

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	`
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	\$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(72	48	H	104	68	h
^I	9	09		HT	41	29)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[27	1B		ESC	59	3B	;	91	5B	[123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	
^]	29	1D		GS	61	3D	=	93	5D]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^~	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	Δ*



HW1-2 Encryption (Cont.)

- ◆ The function “print_char” have been provided in the sample
- ◆ Usage:
 - ◆ 1. Store the beginning address in x20
 - ◆ 2. Use "j print_char"
 - ◆ The function will print the string stores from x20
 - ◆ When finished, the whole program with return value 0



Template of Homework 1-2

- ◆ The plaintext stores in a0 (i.e., x10)
- ◆ Shift will be store in a6 (i.e., x16)
- ◆ Do store “66048(=0x10200)” in x20 before jump to print_char
- ◆ Write your code in the red frame

```

9 #####
10 # print_char function
11 # Usage:
12 #   1. Store the beginning address in x20
13 #   2. Use "j print_char"
14 #   The function will print the string stored from x20
15 #   When finish, the whole program with return value 0
16
17 print_char:
18     addi a0, x0, 4
19     la a1, msg2
20     ecall
21
22     add a1,x0,x20
23     ecall
24
25 # Ends the program with status code 0
26     addi a0,x0,10
27     ecall
28
29 #####

```

```

# Prints msg1
addi a0, x0, 4
la a1, msg1
ecall

# Reads an int
addi a0, x0, 5
ecall
add a6, a0, x0

# Prints msg2
addi a0, x0, 4
la a1, msg2
ecall

addi a0,x0,8
li a1, 0x10150
addi a2,x0,2047
ecall

# Load address of the input string into a0
add a0,x0,a1

```

```

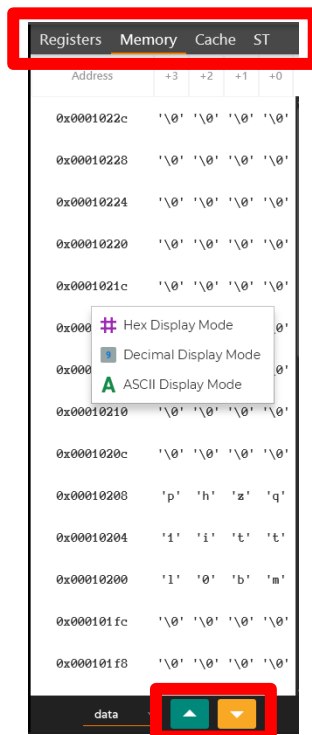
#####
# Write your main function here.
# a0 stores the beginning Plaintext
# x16 stores the offset
# Do store 66048(0x10200) into x20
# ex. j print_char
#####

```



HW1 Report

- ◆ HW1-1: snapshot the result with input n=35
- ◆ HW1-2: snapshot the result with following inputs and the value in memory 0x10200 as **ASCII code format**
 - ◆ shift = 5, plaintext = d gjqz xv nj hpzc
- ◆ Make this file into a **pdf** file (read the submission)



Right
click
and
choose
ASCII



Address	+3	+2	+1	+0
0x00010228	'\0'	'\0'	'\0'	'\0'
0x00010224	'\0'	'\0'	'\0'	'\0'
0x00010220	'\0'	'\0'	'\0'	'\0'
0x0001021c	'\0'	'\0'	'\0'	'\0'
0x00010218	'\0'	'\0'	'\0'	'\0'
0x00010214	'\0'	'\0'	'\0'	'\0'
0x00010210	'\0'	'\0'	'\0'	'h'
0x0001020c	'c'	'u'	'm'	'3'
0x00010208	'o'	's'	'2'	'a'
0x00010204	'c'	'1'	'e'	'v'
0x00010200	'o'	'l'	'0'	'i'
0x000101fc	'\0'	'\0'	'\0'	'\0'
0x000101f8	'\0'	'\0'	'\0'	'\0'

```

Console
This is HW1-2:
Enter shift: 5
Plaintext:  d gjqz xv nj hpzc
Ciphertext: i0love1ca2so3much
  
```




Rules

- ◆ For HW 1-1 and 1-2, brute-force is not allowed
 - ◆ Implement HW1-1 with recursive function
 - ◆ Implement HW1-2 with loop function only
- ◆ Please do write some comments in your codes
- ◆ Input will be changed while grading
- ◆ Do **NOT** modify the input, output, and any provided instructions



Submission

- ◆ Deadline: Apr. 05, 2024 (11:59 p.m.)
 - ◆ No late submission allowed
- ◆ Hand in two source codes and a pdf report on NTU cool
- ◆ Your homework should be copied into a folder and packed into a zip file with the following naming rules
 - ◆ hw1_<student_id>.zip
 - hw1_<student_id>
 - hw1-1_<student_id>.s
 - hw1-2_<student_id>.s
 - hw1_report _<student_id>.pdf
 - ◆ Use lowercase for student ID
 - ◆ Ex: hw1_r12943033.zip