

ECE 212 – Digital Circuits II

MIPS Assembly Programming

Otieno Maurice (Scribe)

Alex Villalba

November 1, 2023

https://github.com/otienomaurice1/ece212_alex_maurice.git

1. Introduction

MIPS assembly language refers to the assembly level language for Microprocessors without interlocked pipelined stages (MIPS) based processors such as PIC32MX250F128B that we used to drive the lafbot. It allows us to write code that communicates directly with the microprocessor in the form of acronyms that represent the instruction to be executed. The instructions themselves are binary i.e., 1's and 0's compounded into 32 bits that can be executed by the machine.

In this lab we wrote MIPS assembly code to compute the Caesar Cyphered encoding of a particular string in Memory. We first wrote C code containing the instructions in a text editor. We then used this high-level code as our guide to write MIPS assembly code for those instructions. We then used Microchip's MPLAB X IDE simulation environment to step through each and every instruction in Debug Mode. Every time we stepped through an instruction, we kept track of the CPU registers and watched how they changed. We also checked stack and memory changes every time we computed a store or load instruction in memory.

2. Design

8. A "Caesar Cipher" is an ancient encryption scheme attributed to Julius Caesar in which letters in a "plaintext" are replaced by alternative characters in the encrypted text (or "ciphertext"). Specifically, the Caesar Cipher *shifts* each character by some offset k . For example, the shift $k=4$ **Plaintext:**

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ciphertext (Shifted to the right by 4)

W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Thus, the plaintext message "HELLO ECE" would be encoded as the ciphertext "DAHHK AYA."
(Homework4: question 8 – Lauren Biernaki).

65	A	81	Q
66	B	82	R
67	C	83	S
68	D	84	T
69	E	85	U
70	F	86	V
71	G	87	W
72	H	88	X
73	I	89	Y
74	J	90	Z
75	K	91	[
76	L	92	\
77	M	93]
78	N	94	^
79	O	95	_

Figure 1 ASCII Table for uppercase letters of the English alphabet

For purposes of explaining the design, we included the C code for the instructions but the MIPS code is available at the repository accessed by the linked at the top of this document.

```
int main () {
    int k = 4;

    char [] s; //declare s

s = "S1"

    encode_string(&s,k); // call encode string an pass in the offset k and the location of the string s
s = "WELCOME BACK MY FRIENDS 2 THE show THAT NEVER ENDS"

    encode_string(&s,k); // second call of encode string with the updated value of k


    return 0; // return back to caller if successful
}

void encode_string (char *s, int k) {
    char c; //declare c

    while ( (*s != '\0') ) { // while not end of file

        c = *s; // dereference s, update c to value held in s
```

```

    *s = encode_char (c, k); //dereference s, pass c and k to encode_char, update s to value returned
    by encode_char
    s++;
}
}

char encode_char (char c, int k) {
    if (c < 'A' || c > 'Z') //check that c is an uppercase character
        return c;
    else {
        int offset = c - 'A'; // Shift to the right k, considering wraparound and // negative values
        offset = ((offset - k) + 26) % 26; // Map value back into ASCII range and update msg
        return (offset + 'A');
    }
}
}

```

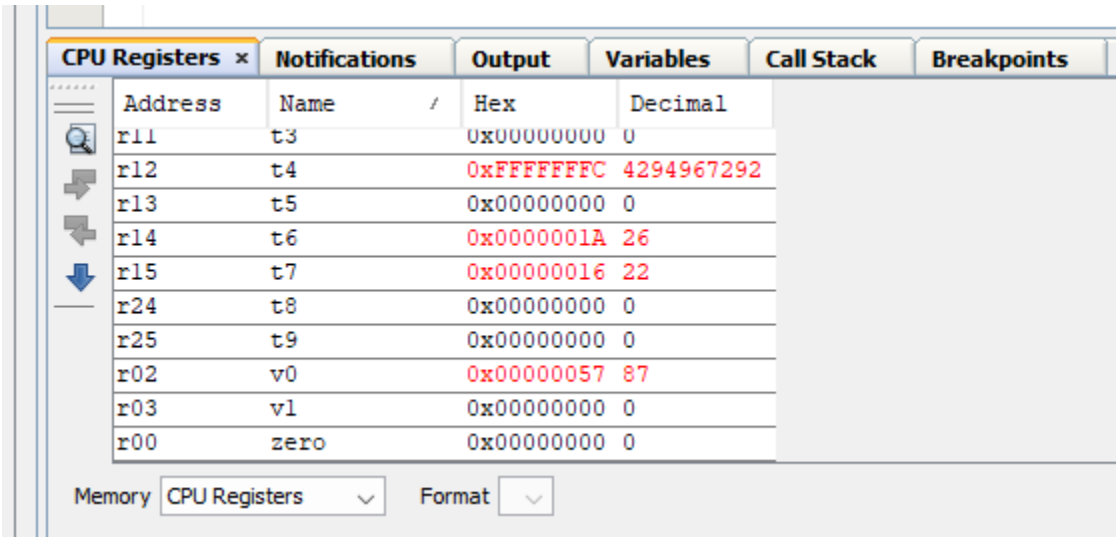
The Caesar cypher encoding utilizes three functions. The main function called by the Operating System, the encode string function called by the main function and the encode char function called by the encode string function. The encode string function has two parameters with which it receives two arguments passed down from the main method. These arguments include a pointer to the string we wish to access and the offset k with which we use to encode. The encode string function loops through each character in the string and checks if the value is the null character \0. As long as the null character is not found i.e., we haven't reached the end of the string, it passes the value of c and the offset k to encode character function.

The encode character function receives the character that is to be encoded as an argument. It encodes it and returns the encoded character back to encode string function. The encode string function then pushes the encoded string back to memory.

In this lab we first began by testing the string length function MIPS assembly. We later tested for the encode character function separately with characters A and Z. Finally, we tested the entire program with strings stored in Memory.

3. Results

encode char register screenshots

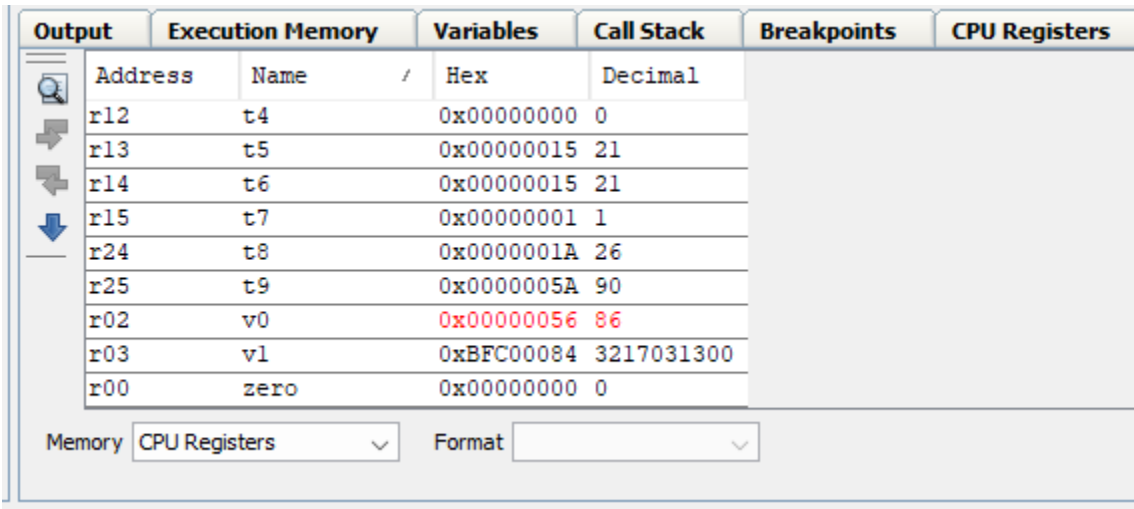


Address	Name	Hex	Decimal
r11	t3	0x00000000	0
r12	t4	0xFFFFFFFF	4294967292
r13	t5	0x00000000	0
r14	t6	0x0000001A	26
r15	t7	0x00000016	22
r24	t8	0x00000000	0
r25	t9	0x00000000	0
r02	v0	0x00000057	87
r03	v1	0x00000000	0
r00	zero	0x00000000	0

Memory CPU Registers Format

Figure 2 encoded form of uppercase character A in register \$v0

As expected, when we encoded A the return register \$v0 got the ASCII value of w = 87.



Address	Name	Hex	Decimal
r12	t4	0x00000000	0
r13	t5	0x00000015	21
r14	t6	0x00000015	21
r15	t7	0x00000001	1
r24	t8	0x0000001A	26
r25	t9	0x0000005A	90
r02	v0	0x00000056	86
r03	v1	0xBFC00084	3217031300
r00	zero	0x00000000	0

Memory CPU Registers Format

Figure 3 encode char register view with \$v0 holding the encoded value of k

As expected, when we encoded Z, register \$v0 got the ASCII value of v = 86.

Memory and stack screenshots

0000_01D0	00000000	00000000	00000000	00000000
0000_01E0	00000000	00000000	00000000	00000000
0000_01F0	00000000	00000000	00000000	00000000
0000_0200	434C4557	20454D4F	4B434142	20594D20	WELCOME BACK MY
0000_0210	45495246	2053444E	48542032	68732045	FRIENDS 2 THE sh
0000_0220	5420776F	20544148	4556454E	4E452052	ow THAT NEVER EN
0000_0230	00005344	00000000	00000000	00000000	DS.....
0000_0240	00000000	00000000	00000000	00000000
0000_0250	00000000	00000000	00000000	00000000

Figure 4 Memory address 0000_2000 before program execution

0000_7ED0	00000000	00000000	00000000	00000000
0000_7EE0	00000000	00000000	00000000	00000000
0000_7EF0	00000000	00000000	00000000	00000000
0000_7F00	00000000	00000000	00000000	00000000
0000_7F10	00000000	00000000	00000000	00000000
0000_7F20	00000000	00000000	00000000	00000000
0000_7F30	00000000	00000000	00000000	00000000
0000_7F40	00000000	00000000	00000000	00000000
0000_7F50	00000000	00000000	00000000	00000000

Figure 5 stack before program execution

Output	Variables	Call Stack	Breakpoints	CPU Registers	Data Memory x	Data Memory
	Address	00	04	08	0C	ASCII
	0000_7F70	00000000	00000000	00000000	00000000
	0000_7F80	00000000	00000000	00000000	00000000
	0000_7F90	00000000	00000000	00000000	00000000
	0000_7FA0	00000000	00000000	00000000	00000000
	0000_7FB0	00000000	00000000	00000000	00000000
	0000_7FC0	00000000	00000000	00000000	00000000
	0000_7FD0	A0000200	9D000014	00000000	00000000
	0000_7FE0	00000000	00000000	00000000	BFC0014CL...
	0000_7FF0	00000000	00000000	00000000	00000000

Memory Data Memory Format Data

Figure 6 stack at beginning of encode char

The address 0000_FD00 corresponds to the value held by the argument register \$a0 i.e., value passed from main. It is saved before the register is overridden with the value; we are passing to encode string.

The address 0000_FD04 holds the value of the return address of the encode string function. This value is stored before we jump to the function, we called in this case encode char and the register overridden with the return address of encode char.

Output	Variables	Call Stack	Breakpoints	CPU Registers	Data Memory	Data Memory x
	Address	00	04	08	0C	ASCII
	0000_01C0	00000000	00000000	00000000	00000000
	0000_01D0	00000000	00000000	00000000	00000000
	0000_01E0	00000000	00000000	00000000	00000000
	0000_01F0	00000000	00000000	00000000	00000000
	0000_0200	5700314F	4F434C45	4220454D	204B4341	01.WELCO ME BACK
	0000_0210	4620594D	4E454952	32205344	45485420	MY FRIEN DS 2 THE
	0000_0220	6F687320	48542077	4E205441	52455645	show TH AT NEVER
	0000_0230	444E4520	00003F53	00000000	00000000	ENDS?..
	0000_0240	00000000	00000000	00000000	00000000

Memory Data Memory Format Data

Figure 7 memory at the end of the first call of encode string

In the first call of encode string function, we encode the string "S1". Since 1 can't be encoded with our encode char function, its original value is returned. As you can see from the screen shot S has been encoded to O as per the table shown in the design section. Note that the encoded value is stored in the same location the original string was received.

Output	Variables	Call Stack	Breakpoints	CPU Registers	Data Memory	Data Memory x
	Address	00	04	08	0C	ASCII
	0000_01C0	00000000	00000000	00000000	00000000
	0000_01D0	00000000	00000000	00000000	00000000
	0000_01E0	00000000	00000000	00000000	00000000
	0000_01F0	00000000	00000000	00000000	00000000
	0000_0200	4100314F	53475049	46204951	204F4745	01.AIPGS QI FEGO
	0000_0210	4A204351	52494D56	32205748	494C5820	QC JVMIR HW 2 XLI
	0000_0220	6F687320	4C582077	52205845	56495A49	show XL EX RIZIV
	0000_0230	48524920	00003F57	00000000	00000000	IRHW?..
	0000_0240	00000000	00000000	00000000	00000000

Memory Data Memory Format Data

Figure 8 memory at the end of the second encode string call

In the second call of encode string we encode the string " WELCOME BACK MY FRIENDS 2 THE show THAT NEVER ENDS?". All the characters that can't be encoded by encode char are returned as they were while the uppercase letters are encoded and stored in memory.

9. Conclusion

The main challenge faced was coming up with working MIPS code for encode String and encode char functions. We initially forgot to include instructions like sb \$v0,0(\$a0) to store encoded values back to memory. We also confused sw and sb and its position relative to sequence of instructions. We were initially computing sw instead of sb and before restoring the argument register \$a0's value from the stack. The effect was that only the first character within the first word of memory was encoded and the remaining 3 bytes within that word in memory were filled with the null character.

Consequently, in the next iteration of the while loop, the encode string function found the null character in memory and jumped back to main failing to encode other characters.

10. Time spent on the lab

4 hrs.