**ECE 212 – Digital Circuits II**

**MIPS Assembly Programming**

**Otieno Maurice (Scribe)**

**Alex Villalba**

**November 1, 2023**

[**https://github.com/otienomaurice1/ece212\_alex\_maurice.git**](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.

1. **Design**
2. 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).

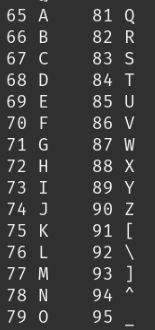


Figure ASCI Table for uppercase letters of the English alphabet

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 fist 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.

1. **Results  
   encode char register screenshots**

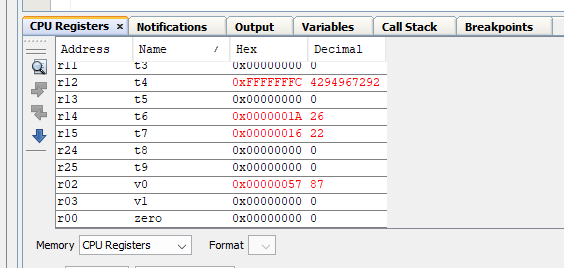


Figure encoded form of uppercase character A in register $v0

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

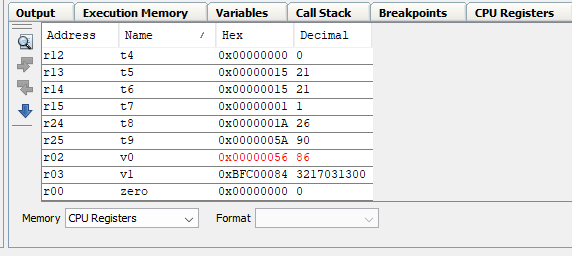


Figure encode char register view with $v0 holding the encoded value of k

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

**Memory and stack screenshots**

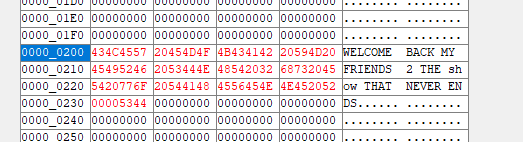


Figure 4 Memory address 0000\_2000 before program execution

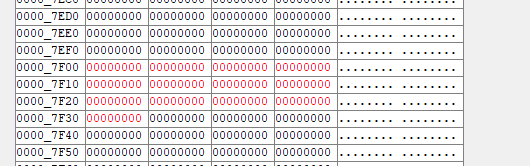


Figure 5 stack before program execution

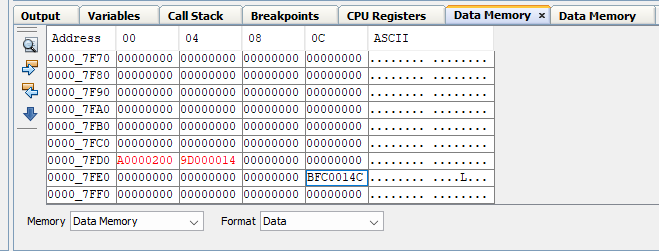


Figure 6 stack at beginning of encode char

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.

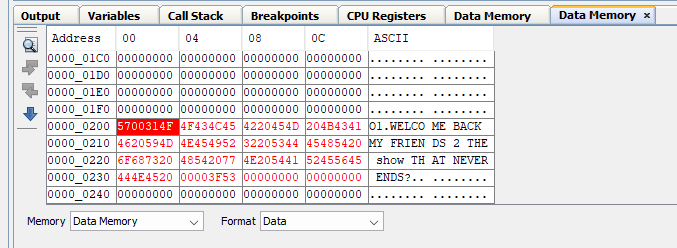


Figure memory at the end of the first call of encode string

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.

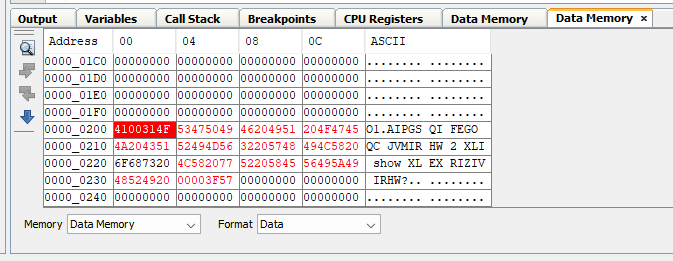


Figure memory at the end of the second encode string call

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.

1. **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.

1. **Time spent on the lab**

**4 hrs.**