**Project 3 Report**

Christopher Badolato

[badolato.christopher@knights.ucf.edu](mailto:badolato.christopher@knights.ucf.edu)

EEL 3801-0011: Computer Organization

4/8/2019

1. Project Description

The purpose of this design project is to create a QuadBitCipher of two user input values. This cipher will, compare these two values, 4-bit by 4-bit (Hex by Hex) to create a new output z. We will “break” the value into sections of 4-bits to see which set of 4-bits (4-bits from input x, 4-bits from input y) is smaller. The smaller of the two will be stored onto our output z, creating a new 32-bit integer. Once we have this new 32-bit value we will break it up into 4-bit groups and count the frequency of each. We will then print out the frequency array which will represent our cipher.

1. Program Design

To begin, QuadBitCipher: will first ask the user for input integers representing 32 binary bits each. Inputs x and y will be stored into registers $t0 and $t1. We can then move these values into our registers $a1 and $a2. We do this to pass these values as parameters to the QuadMinMixer function which will be called immediately after with jal QuadMinMixer.

We are now in our QuadMinMixer function. We can first begin by creating space on the stack by adding a (-12) to the $sp register. We can then load our three values to save ($ra, $s0, $s1) to the stack.

Next, we need to create a mask value initially storing 0x0000000f. We can bitwise AND this mask with our input to create a 4-bit copy of each 4-bit group of the inputs. ANDing anything and 0 will result in 0 but ANDing anything with 1 will create a copy of the value (works like multiplication). Shifting this mask left each iteration by 4 will result in us creating a mask of each 4-bit sets of the 32-bit inputs. We also need to create a result variable ($t1), that initially will store 0, or 32-bits of 0. Since the OR operation will essentially add two binary bits together, we can continuously OR our result with the 4-bit values found in each iteration. We iterate until our 32-bit result is full or 8 times. Filling in the result 4-bit at a time from left to right (LSB to MSB).

So, for each iteration we first create temporary masks by ANDing each input ($a1, $a2) with our mask ($t0), which is shift left immediately after each iteration by 4. We store these temporary masks into (result of x&&mask ($s0) and result of y&&mask ($s1)). Essentially, ($s0 = $t0 && $a1). We can now compare these values because they represent the same group of 4-bit powers of 2. Using bge we compare ($s0 and s$1). If ($s0) is greater, we can take our branch xIsGreater(1-8), which will OR ($s1) to the result ($t1) since y is the minimum of the two values then continue to the next iteration. Otherwise, we can assume x is the minimum of the two values and OR ($s0) with result ($t1) then jump to yIsGreater(1-8):. Which skips over xIsGreater(1-8):. These branches will continue until the result ($t1) is full with 8 hex bits, or 8 groups of 4-bit binary.

After the function has finished, we can pop the values ($ra, $s0, $s1) off the stack and return the space by adding 12 to ($sp). We can as well return the result ($t0) to QuadBitCipher by storing it into $v0.

State as a final

sentence in this section, the minimum required Cache Size in Bytes to achie

ve >=90% hit rate.

1. Symbol and Label Tables

Symbol Table

|  |
| --- |
| **Registers used in main function (QuadBitCipher)** |

|  |  |
| --- | --- |
| Register | Assignment |
| count .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 | Stores and initializes the frequency array |
| $v0 | Stores our return value, from QuadMinMixer.  Also used throughout the program to store our syscall value. |
| Sa1 |  |
| $a2 |  |
| $s0 | When QuadMinMixer is finished we store the return value in our register $s0.  We will count the frequency of hex values from this. It is z. |
| $s1 | Stores base address of our frequency array “count”. |
| $t0 | Stores our input x.  Is overwritten to store:  Mask position value. Originally (0x0000000f) |
| $t1 | Stores our input y.  Is overwritten to store:  Result of ANDing mask with input z($s0) for each set of 4-bits. |
| $t2 | Value used to load the CURRENT frequency in the count array.  We will load the value from the count array.  Increment it, then store it back onto the array  lw $t2, 0($s1)  add $t2, $t2, 1  sw $t2, 0($s1) |
| $t3 | Counter for print loop. Incremented to 16 to print frequency array. |
| $t4 | Used to load value at current index of frequency array in our printLoop. |

|  |
| --- |
| **Registers used in quadMinMixer function.** |

|  |  |
| --- | --- |
| register | Assignment |
| $sp | “stack pointer”, used to keep track of the caller, as well as the passed parameters to QuadMinMixer. |
| $ra | Stores return address of the call of QuadMinMixer in QuadBitCipher |
| $a1 | Stores input x as a parameter to the function. |
| $a2 | Stores input y as a parameter to the function. |
| $s0 | Masked value. Stores each iteration’s group of four bits masked from the input x.  Is || (or’d) with result stored in $t1 each iteration to form output z.  Represents **tempx** in C prototype. |
| $s1 | Masked value. Stores each iteration’s group of four bits masked from the input y.  Is || (or’d) with result stored in $t1 each iteration to form output z.  Represents **tempy** in C prototype. |
| $t0 | Stores the mask value that we will AND && with each input value to create a temporary value that we can compare. Then we can decide the minimum.  Shifted left by 4 each iteration.  Represents **mask** in C prototype. |
| $t1 | Stores output result that is created 4-bits at a time starting from the right least significant 4-bits by ORing it with either $s0 or $s1  Represents **results** in C prototype |
| $v0 | After all 8 iterations of 4-bits has finished we will store $t1 result into $v0 to be returned to our main function. |

Label Table

|  |
| --- |
| **QuadBitCipher Labels** |

|  |  |
| --- | --- |
| Label | Use |
| QuadMinMixer | Calls function QuadMinMixer |
| printLoop: | Loops 16 times to print the frequency array after occupation. |

|  |
| --- |
| **QuadMinMixer Labels** |

|  |  |
| --- | --- |
| Label | Use |
| QuadMinMixer: | Represents the start of the function. |
| xIsGreater1-xIsGreater8 | Act as if else statements  We will compare the values stored in $s0 and $s1 (our masked 4 bits from inputs x and y)  If x is greater we will branch to xIsGreater1 which will add $s1 (y’s 4-bits) to the result. Otherwise, |
| yIsGreater1-yIsGreater8 | Otherwise we can assume y is greater. Which will add $s0 (x’s 4-bits) then JUMP us to yIsGreater1, which will continue us on to the next iteration.  Each iteration contains one of each of the two labels above to create this “if else” statement. |

1. Learning Coverage

* Using the stack to store previous iterations of values for potential to use program recursively.
* Indexing of an array with a previously unknown value by shifting loaded base address by memory offset.
* Creation of function in MIPS using parameters, up to 4 or as many as we’d like. As well as returning a value from a function.
* 4
* 5

5.0 Prototype in C

Code can be easily copy and pasted into IDE or preferred text editor.

|  |
| --- |
| /\*  Christopher Badolato  4/8/2019  EEL 3801 0011  Project 3  This program is a C prototype of the QuadMinMixer as well as the QuadBitCipher  Takes two 32 bit input values, grabs the minimum of each group of four bits between the two and creates  a new output Z. The Hexadecimal frequency is then taken of the output value Z.  \*/  #include <stdio.h>  //Function prototypes  unsigned QuadMinMixer(unsigned x, unsigned y);  int main (){  //Initialize frequency array, loop counter as well as temps and mask.  int count[16], i;  unsigned x, y, z, temp;  unsigned mask = 0x0000000F;  //get user inputs of x and y.  scanf("%u", &x);  scanf("%u", &y);  //Call QuadMinMixer with x and y and store the output in z.  //Print the output  z = QuadMinMixer(x,y);  printf("QuadMinMixer = ");  printf("0x%x\n", z);  //Initialize count array.  for(i = 0; i < 16; i++){  count[i] = 0;  }  //To start our cipher we must first mask our first set of 4 bits from Z.  //and also increment the frequency count at those 4-bits  temp = mask & z;  count[temp]++;  //After the first iteration we can loop through the rest.  //We shift the mask by 4 bits each loop creating a mask of z each iteration  //as well as taking a frequency count at the index of that temporary stored mask.  for(i = 0; i < 7; i++){  z = z >> 4;  temp = z & mask;  count[temp]++;  }  //Finally we can the array (backwards for our expected output.)  printf("QuadBitCipher = ");  for(i = 15; i >= 0; i--){  printf("%d", count[i]);  }  return 0;  }  unsigned QuadMinMixer(unsigned x, unsigned y){  unsigned tempx = 0, tempy = 0, result = 0;  int i;  unsigned mask = 0x0000000F;  //create the temps for our FIRST set of 4-bits to be compared.  tempx = x & mask;  tempy = y & mask;  //We went to "or" or add the minimum of the two temp values to the result.  //if x is larger take y.  //otherwise take x.  if(tempx >= tempy){  result = result | tempy;  }  else{  result = result | tempx;  }  //After our first iteration we can continue shifting the mask left until we've masked the other 7  //remaining groups of 4-bits or hex value.  for(i=0; i < 7; i++){  mask = mask << 4;  tempx = x & mask;  tempy = y & mask;  //if x is greater take y  //other wise take x.  if(tempx >= tempy){  result = result | tempy;  }  else{  result = result | tempx;  }  }  //Finally we return the results.  return result;  } |

6.0 Test Plan

to the output of our MIPS run version. We hardcoded the strings directly into the program.

|  |  |
| --- | --- |
| Inputs | Outputs |
| 1792801454  2016082984 | 1792801454  2016082984  QuadMinMixer = 0x682afa28  QuadBitChipher = 1000020201000200 |
|  |  |
|  |  |
|  |  |
|  |  |

1. Test Results

Following the input strings from above we receive these outputs below. We can see that each character is printed above with the frequency of each character. As well as our hashtag histogram.

|  |
| --- |
| Testcase 1 |
| Testcase 2 |
| Testcase 3 |
| Testcase 4 |
| Testcase 5 |

1. References

* EEL 3801 00219 DeMara Slides, Module 03-MIPS-ISA.pdf.
* Mars 4.5 Mips Simulator
* Recitation files
* Testing reference Sheet
* Based my MIPS code on this stack overflow code. Used to see how to use the stack and store to the stack frame <https://stackoverflow.com/questions/15100476/mips-relevant-use-for-a-stack-pointer-sp-and-the-stack>