EEL3801: Computer Organization and Design
Project 3: Design Option

1.0 Submission Requirements

1.1 Preface

When responding to a professional work solicitation/submission, we gain practice with the protocol that Submission Requirements must be met to be considered for credit. Most companies set this standard, and most government agencies uphold such requirements per their legal contracting and acquisition processes. So we learn these conventions in our course projects and gain practice with formal "Project Solicitation"-style Task Definitions.

1.2 Submission Mechanism

Please upload the submissions identified below to webcourses using the Assignments tab. No other form of submission can be accepted.

1.3 Program Code file

For your program code, submit a *single* file with **`.asm`** extension that provides all parts of the project as a single program. The required file naming convention is **`Project-3-Code-<LastName>-<FirstName>.asm`** are replaced with your last name and first name.

1.4 Project Report file

For your project report, submit a *single* file in .pdf, .doc, or .docx format with the filename **`Project-3-Report-<LastName>-<FirstName>.doc`** where the fields <LastName>-<FirstName> are replaced with your last name and first name, or else .pdf or .docx extensions as appropriate. This submission will be a single file containing your entire report for the project. If your file is not already a single **`.doc`**, **`.docx`** , or **`.pdf`** file, please convert it using free online tools such as http://www.convert-jpg-to-pdf.net/ or merge it using http://www.pdfmerge.com/

1.5 Incremental Submission

Upload your submissions as you progress in creating results, which will overwrite previous partial submissions. Only the last submission received before the due date/time will be graded. Upload what you've completed as you go along to avoid lack of submission in case of system issues prior to the due time or any other reason.

1.6 Submission Deadline

The deadline for submission of materials for consideration is **Monday 8 April 2019 during lab time for Monday sessions** and **Wednesday 10 April 2019 during lab time for Wednesday sessions** as timestamped in Webcourses. Please upload your submission with adequate time before the deadline, in case any contingencies may occur. Latest possible deadline to submit Project3 Code and Report for late submission, which incurs 30% deduction on the overall Project 3 grade, is **Monday 15 April 2019 at 11:59PM for Monday sessions** and **Wednesday 17 April 2019 at 11:59PM for Wednesday sessions**. For fairness, submissions after the deadline cannot be accepted and will not be graded. Only submissions **already in webcourses at the due time** can be graded for credit. Following the syllabus professionalism policies, please do not request late submission: see syllabus for clarification.

2.0 Project Functionality

2.1 Overview

A key topic today is *data encryption*. One approach to data encryption computes *ciphers* using repeated calculations with data values. We examine efficient mathematical realizations for a small case study for producing a cipher using two unsigned integer elements represented as MIPS words. Through this project you will increase your understanding of instruction execution for procedures, stacks, and arrays in memory. You will apply selected instructions and procedures sufficient enough to handle a few cipher-related tasks while also determining cache size required for a certain hit rate.

2.2 Part A: `QuadMinMixer(x,y)`

You are tasked to write a callable/returning MIPS subroutine called **`QuadMinMixer(x,y)`**

Specifically, **QuadMinMixer(x,y)** accepts two unsigned integers **x** and **y** as input parameters. The value returned is the least of each corresponding 4-bit field comprising **x** and **y**.
For example, **QuadMinMixer(1792801454$_{ten}$, 2016082984$_{ten}$) = 682afa28$_{Hex}$** since:

x = 1792801454$_{ten}$= 0110 1010 1101 1011 1111 1010 1010 1110 as 32 bits

y = 2016082984$_{ten}$= 0111 1000 0010 1010 1111 1100 0010 1000 as 32 bits

Minimum per 4-bit field: 0110 1000 0010 1010 1111 1010 0010 1000 as 32 bits

where the lesser 4-bit field that appears in the result is underlined to highlight the term selected. The result equals **682afa28$_{Hex}$** for the example inputs.
In your design to receive credit:

- **x** and **y** shall be passed to **QuadMinMixer(x,y)** as arguments in **$a0** and **$a1**, respectively
- a single unsigned integer value shall be returned in **$v0**
- to gain experience with stack access, the **$ra** register shall be saved on the stack upon entry to the **QuadMinMixer(x,y)** function and restored prior to exit of the **QuadMinMixer(x,y)** function, leaving the stack pointer unchanged. This is required for credit as the customer wants **QuadMinMixer(x,y)** to have recursive capability for future applications.
- if any "save" (**$s0, $s1, ..., $s7**) registers are modified inside of the **QuadMinMixer(x,y)** function, then they shall be saved on the stack upon entry to the **QuadMinMixer(x,y)** function and restored prior to exit of the **QuadMinMixer(x,y)** function
- don't use any memory access inside of **QuadMinMixer(x,y)** besides to/from the stack. This means that any {**lw,lb,sw,sb**} instructions in **QuadMinMixer(x,y)** are only allowed to reference **$sp**. No other use of memory access instructions shall occur in **QuadMinMixer(x,y)**. The purpose of this restriction is so that results get passed back properly as a value, rather than making updates to memory within this subroutine function. Meanwhile, stack use inside of **QuadMinMixer(x,y)** is not restricted.
- the code for the **QuadMinMixer(x,y)** function shall have label **QuadMinMixer:** and terminate with a **jr** instruction

FYI (informational only not part of the project), **QuadMinMixer(x,y)** can be used in signal processing to "clip" a packed input signal **x** at a maximum level specified by **y**, or extended for encryption by choosing alternate computing functions and specifying **y** as an encryption key for the data value **x**.

2.3 Part B: QuadBitCipher(x,y)

Given two unsigned integers **x** and **y**, compute the **QuadBitCipher(x,y)** which has the following pseudocode representing the main code in combined Part A with Part B:

```
QuadBitCipher(x,y) // this will be main() where x and y are input from keyboard
{
      z = QuadMinMixer(x,y);
      For each 4-bit field of z do {
            Convert the 4-bit field to a decimal value;
            Increment the corresponding count seen for that decimal value;
      }
      Display the counts in descending order;
}
```

To receive credit, the above procedure shall use the *radix bin method* to count the frequency of occurrence for each value. Radix binning uses an integer array in memory. Binning with a 4-bit field uses an array containing 16 elements. The i^{th} element of the array contains the count seen so far for the i^{th} value.
More formally, to perform radix binning to count values, initialize to zero an array of 16 integer elements in memory numbered from 0_{ten} to 15_{ten} inclusive. Each element contains the count encountered so far for that hexadecimal value. Scan the values being binned to increment the value at the index in the array as each value is considered in succession. After all counts are tabulated, the output is expressed as a sequence of decimal counts. The counts are packed as a stream of decimal digits.
For example, suppose we are given as input to Part B the value **682afa28$_{Hex}$**. Then radix binning is the process of counting how many of each hex digit exist in that hexadecimal number. Therefore, we can see that:

- 6 occurs one time
- 8 occurs two times
- 2 occurs two times
- a occurs two times
- f occurs one time

The above process is called *binning*, which can be used to find the frequency for components of an input signal. Next, we need to output the above mentioned information in descending order which is listed below:

0001	0000	0000	0000	0000	0010	0000	0010	0000	0001	0000	0000	0000	0010	0000	0000	← *count in binary*
'f's	'e's	'd's	'c's	'b's	'a's	'9's	'8's	'7's	'6's	'5's	'4's	'3's	'2's	'1's	'0's	← *digits*

So that the final output is displayed:
1000020201000200 ← *one F, two A's, two 8's, one 6, two 2's*

2.3.1 Sample C-Prototype for Radix Binning

```
int main(){
    int i;                                      // initialize loop counter
    int arr[8]={9,8,2,10,15,10,2,8};            // output of PART A as an input to PARTB
    int count[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};  //initializing an array for counts

    //Data Binning
    for (i=0;i<=7;i++) count[arr[i]]++;         //for loop to count the frequency of each digit

    //Display counts
    for (i=0;i<=15;i++)                         //for loop to display the frequency of each digit
        printf("%d occurs %d times.\n",i,count[i]);

    //Display the output of PART B in one line
    printf("\nOutput of Part B: ");
    for (i=0;i<=15;i++) printf("%d",count[15-i]); // print the digits in the correct order
    return 0;
}
```

2.3.2 Output from Sample C-Prototype for Radix Binning

```
0 occurs 0 times.
1 occurs 0 times.
2 occurs 2 times.
3 occurs 0 times.
4 occurs 0 times.
5 occurs 0 times.
6 occurs 1 times.
7 occurs 0 times.
8 occurs 2 times.
9 occurs 0 times.
10 occurs 2 times.
11 occurs 0 times.
12 occurs 0 times.
13 occurs 0 times.
14 occurs 0 times.
15 occurs 1 times.

Output of Part B: 1000020201000200
```

2.4 Additional Requirements and Error Handling

2.4.1 Well-formed Input Values

The values of **x** and **y** will each be non-negative integers which each individually do not exceed a value $2{,}100{,}000{,}000_{ten}$.

2.4.2 Error Handling

No error handling checking code is required for this project.

2.5 Single Program Submission
For full credit, implement Parts A and B in a single MIPS program that can execute in MARS. Operationally, the user will enter **`x`** from the keyboard, followed by **`y`**. The program should produce output in **exactly the sequence shown** along with the declarative labels listed in Section 3.6.

2.6 Sample Program Output
Using the values of **`x=1792801454`** and **`y`**=**`2016082984`** in Section 3.2 above, then the program shall output:

```
QuadMinMixer = 0x682afa28
QuadBitCipher = 1000020201000200
```

Using **`x=1277564965`** and **`y=735994003`**, then the program shall output:

```
QuadMinMixer = 0x2b261023
QuadBitCipher = 0000100001001311
```

3.0 Program Evaluation
3.1 Evaluation Method
Project Code scores will be evaluated by entering stress test values to evaluate functionality. Any missing, incorrect, or extra unspecified functionality from these stress test values will result in a proportional or complete deduction of credit.

3.2 Operational Code Required
Any assembler errors (e.g. MARS is unable to assemble the program code), as well as program runtime failures or infinite loops will result in no credit for affected parts of the Program Code for this Project. Please verify that your submission is operational prior to final submission.

3.3 Individual Work
- The entire submission of Program Code and Project Report shall be your own individual original work. Any unauthorized use of shared, jointly written, prototyped, template, example, or re-used code results in no credit for the entire Project.
- Code must be developed manually without use of any automated translator/compiler support.
- Submitter must understand code adequately to explain in-person each line of code submitted, else credit for the entire submission cannot be provided.
- Peer and For-Hire Assistance prohibited from tracked sites including: **`chegg.com`**, **`freelancer.com`**, etc. via automated Google alerts, Digital Millennium Copyright Act bots, and reciprocal agreements: http://www.phoenix.edu/news/releases/2013/07/university-of-phoenix-wins-settlement-on-academic-integrity-continues-fight-against-external-market-for-academic-cheating.html
- **NOTE: to be fair to all students, plagiarism or inadequate use of citations will result in a zero score. Namely, the source for any text and/or figures which you have not originated yourself must be formally cited as a reference.**

4.0 Grading Rubric

4.1 Project Code submission: 70 points total as follows:

- **Proper functionality for Part A:** [20 points for passing all applied stress tests]
  - or else Partial Credit for correct output except for 1 stress test: 15 of 20 pts earned
- **Proper functionality for Part B:** [20 points for passing all applied stress tests]
  - or else Partial Credit for correct output except for 1 stress test: 15 of 20 pts earned
- **Comments:** up to 10 points awarded based on completeness and quality of comments provided compared to the Positive Examples in webcourses as follows [10 points]:
  - program header containing all components of Positive Examples in webcourses: 4 pts
  - description for each code block as shown in Positive Examples in webcourses: 3 pts
  - one comment for each line of code in the program: 3 pts
- **Dynamic Instruction Efficiency:** [10 points]
  Dynamic Instruction Count will be evaluated by the **MARS4.4→Tools→Instruction Counter** for the keyboard-entered values **`x=1792801454`** and **`y=2016082984`** resulting in:
  - Dynamic Instruction Count of 315 instructions or less in total for both parts A and B executed as a single MIPS program: 10 pts (in actuality fewer instructions are achievable, although 315 or less is acceptable for full credit)
  - If above value is not met, but no more than 350 instructions executed then 5 pts (half partial credit) is provided
  - Dynamic Instruction Count is calculated from the first instruction execution until completion of a proper exit using syscall, as calculated in the "**Instructions so far:**" dialog box.
- **Cache Efficiency:** [10 points]
  Since we are covering Cache Memory in Lecture, thus *Cache Hit Rate* will be evaluated by the **MARS4.4→Tools→Data Cache Simulator** for the keyboard-entered values **`x=1792801454`** and **`y=2016082984`** under the parameters of:

  Replacement Policy = Direct Mapping
  Block Replacement Policy = LRU
  Set Size = 1

  You are tasked to report to the memory system design team:
  "What is the minimum cache size in Bytes sufficient to achieve Cache Hit Rate >= 90% ?"
  - Cache Hit Rate shall be calculated from the first instruction execution until completion of a proper exit using syscall in MARS 4.4 for your complete program.
  - You will need to try different combinations of **Number of Blocks** and **Cache block size**, to determine the necessary cache size.
  - To obtain the required Cache Hit Rate, if needed you can also optimize the sequence of "data transfer category" (memory access) instructions in your program based on your understanding of how cache works to increase hit rate mentioned during Lecture.
  - Identify the minimum required **Cache size (Bytes)** indicated in MARS to achieve a 90% hit rate for the inputs above, as the last sentence in *Section 2.0 Technical Design* of your report, in order to receive credit for this task.

4.2 Project Report submission: 30 points total as follows:

- **Professional preparation:** [3 points total] as follows:
  i.e. Provide a full page for coverpage which lists project title, student name, email address, course name, and due date. Submit a typed document with text of the paragraphs in Times New Roman 11 pt font, clear and grammatically well-formed explanations, page numbering, and document heading numbering (1.0, 2.0, 3.0, etc to identify the required sections listed below).

- **Report Content:** [27 points total] as follows having the following numbered section headings:

  1.0 Project Description: project name, narrative description of at least 4 sentences, including identification of program inputs and outputs. [1 point]

  2.0 Technical Design: narrative description of how your code operates, and a flowchart with sufficient explanation about the program design for someone else familiar with MIPS to be able replicate your

design [4 points for detailed narrative and 4 points for adequately-detailed flowchart]. State as a final sentence in this section, the minimum required Cache Size in Bytes to achieve >=90% hit rate.

3.0 Symbol Table: a 2-column Table describing all Registers used and their specific Purpose in the code, where each register is listed on a separate row and identified by register name **`$t0, $s0,`** etc., as well as any Labels used and their purpose on separate rows. [2 points for register table and 2 points for label table]

4.0 Learning Coverage: provide a meaningful list of at least 5 technical topics learned from this project that you could mention in a job interview. To receive credit, topics mentioned must be completely specific and exclusive to this particular project assignment, versus other projects in this class. [5 points]

5.0 Prototype in C-language: You can use your own C-compiler or free C-compiler available at http://codepad.org Paste the C-code in the report as you do not need to submit a .c file and do not need to provide a screenshot of the C-language output for this project. However, the .c code shall be a viable fully working prototype for all parts to receive credit. [4 points]

6.0 Test Plan: provide details in sentences identifying the inputs chosen to test the program and justification why they provide adequate test coverage. [2 points]

7.0 Test Results: provide screen shot(s) of at least 3 proper MIPS code executions in MARS corresponding to your Test Plan inputs. [2 points]

8.0 References: provide a list of all reference materials you used in the project. [1 point]

5.0 Useful Considerations, Techniques, and Hints

- Prototype all parts in C-language to have them fully operational *before* starting any MIPS code. Bring your prototyped C-code to office hours and any help sessions so we may better assist to understand your code by seeing a working design in C-language.
- Use of a loop is required to receive credit.
- Use of an array is required to receive credit.
- Below is the syntax to output 8-digit hexadecimal value in C:

```
int i = 7;
printf("0x%.8x\n", i);  // outputs 0x00000007
```

- The **`.word`** directive stores the specified values as 32-bit words within the **`.data`** segment.
- The **`sllv`** and **`srlv`** instructions can be used to shift left or right a variable number of bit positions.
- This program does not require a huge number of MIPS instructions, so if your code is long or complicated then go back to C-language to make your flow more efficient.
- **Please do not email your code for any reason: instead please attend recitation or visit in office hours.**
- You must bring your completed prototyped C-code when seeking assistance so we can understand your design to be able to ask you questions in real-time, and get you back on your way quickly.

