Overview

This Lab is a competition amongst the other groups in this course to see who can design and implement a computer. This lab is a continuation of the previous with many new levels of complexity added. Our goal for this assignment is to have a computer that is capable of doing 2x2 matrix multiplications. We will be using Logism to implement this and also code a program in "C" to simulate and compare performance with our digital circuit. From this code we will have the required assembly language. The combinational circuit design methodology is as follows:

Step 1: Define problem
Step 2: Encode variables
Step 3: Create truth table
Step 4: Find simplified function(s)
Step 5: Draw logic circuit
Step 6: Convert to NAND's
Step 7: Check assumptions
Step 8: Chip circuit diagram

A sample of the circuit assembly language taken from our lab document.

00h (OP-CODE = 00000000) Ri + counter#1 \rightarrow Rk
01h (OP-CODE = 00000001) Rk + counter#2 \rightarrow Rk
02h (OP-CODE = 00000010) Ri + Rj \rightarrow Rk
03h (OP-CODE = 00000011) counter#1 + counter#2 \rightarrow Rk

04h to 07h (OP-CODE = 000001XX) Reserved for subtraction instructions

The Completed Circuit

A Sample of the C input/math code

```
/*Have the user enter in the 4 entries in the first matrix
 *printl("Matrix 1\n");
 *printf("\n\n\n\n\n\n");
 */
int temp1;
int temp2;
//Find first entry
temp1 = matrix1[2] * matrix2[2];
temp2 = matrix1[3] * matrix2[3];
resultMat[2] = temp1 + temp2;

//Find second entry
temp1 = matrix1[3] * matrix2[1];
temp2 = matrix1[1] * matrix2[3];
resultMat[1] = temp1 + temp2;
```

![Control Logic](image)

The picture above shows the rom and how based on the clock generator it sequentially executes the desired operation. Although abstract to show how the rom works the picture below shows the various values within the rom.

To initiate the sequence in Logism the user can select the frequency at which the clock ticks. This makes the circuit quite fast after the user initiates the simulation. This also means that we can select a very slow frequency which helps people understand what operations are doing what when we present this design.

Some problems that we encountered span from decision making to implementation issues. Amongst the group members we had difficulty picking what we wanted our machine to do. Once we had a nice list of options we voted on what was best. After that was solved the other issue we had was how to handle the overflow with the circuit. The solution to that was a quick meeting with our professor explaining to him the situation. As stated earlier in this paper because we proved to be capable of handling a single case of multiplication overflow it was not necessary to implement that into this system.

To design our "C" programming portion of this lab we went onto a PC running a Linux distribution called Ubuntu to code, create the assembly language representation of our code, and test the operation of our code.

The code is too long to print but here is a sample of it:

```
.LC11:
.string "Entry 2x2: %dxn"
.text
.globl main
.type main, @function
main:
.LFB0:
   .cfi_startproc
   pushq %rbp
   .cfi_def_cfa_offset 16
   .cfi_offset 6, -16
   movq %rdx, %rsi
   movq $.LC0, %rdi
   movq $48, %r8p
   call $L1C1
   movq %rdx, %rax
   movq %rax, %rdi
   movq $L1C2, %rdx
   leaq -48(%rbp), %rdx
   movq %rdx, %rax
   movq %rax, %rax
   .cfi_endproc
```

LAB #9, 10, 11 “COMPUTER DESIGN COMPETITION”
Josiah Buxton
Andrew Khela
Brady Rudesill
EGR 433: Advanced Computer Architecture
Elizabethtown College, Elizabethtown, PA