# Lab Assignment 0

This assignment is to refresh material from ECE 2035 and ECE 2020. You may collaborate with other students but **must work on and turn in individual solutions**.

## Part I:

**Purpose**: To reacquaint you with the datapath you created in ECE 2020. The important point is to recollect the control of basic register-register and register-memory operations by setting the signals that control the datapath. This sets up the translation of assembly instructions to control signals – an activity you will address later in this course.

For the datapath shown overleaf, write the microcode that implements the following expression.

R1 = 3R2 + 4M[100]

## Part II:

**Purpose**: This part will reacquaint you with the MIPS ISA. Write a MIPS program as described below – but first a few words about simulators. You may work with each other on this part of the assignment.

We will be using the QtSpim simulator in this class as a source of examples. You will not have assembly programming assignments in this class. The simulator will only be used to study program encodings. You could also use any available MIPS assembler, but ensure that the output is MIPS compliant.

The following warm-up will help you get familiar with QtSpim and should not take more than 20-30 minutes if you have no problems. *If you take longer, talk to the TA or me*!.

1. Download and install QtSpim:
   - https://sourceforge.net/projects/spimsimulator/files/
   - The following steps are simply to familiarize you with the SPIM simulator.
2. Select Simulator → Settings (QtSPIM->Preferences on a MAC) and the MIPS tab. Make sure *Accept Pseudoinstructions* is selected. *Bare Machine* should not be selected.
3. Select Window→ Tile. This will create tabs for multiple windows in SPIM.
4. Set starting address of the code to 0x00400000 by going to Simulator → Run Parameters
5. Set register display to hexadecimal by Registers→Hex
6. Select all windows under the Text Segment
7. Make sure all three structures are selected under the Data Segment and the Hex display is selected.
8. Make sure all elements are selected under Windows.

9. First we will study the example SPIM program provided *constant.example.s*
10. Load program: File → Load File. If there are any errors you should see this in the bottom windowpane.
11. Select the Int Regs tab. This should display all of the registers. Note the value of register $t4. It is 0x00000000. From the SPIM program you know that this is the register used to construct a 32-bit constant.
12. Execute this program by Simulator → Run/Continue
13. Now look at $t4. It will contain the constant value as specified in the original program. Study the program to refresh your memory about MIPS instructions.
14. Reload the program by File → Reinitialize and Load File
15. Select the Int Regs tab. This should display all of the registers. Note the value of register $t4. It is 0x00000000. Now single step the program with Simulator → Single Step. At step 2 you will see the upper bits set. At step 3 you will see the lower bits set. At the top of the window you will see the Program Counter advance by 4 with the execution of each instruction step.
16. Note the last two instructions: This is the proper way to terminate a SPIM program. Every program should end this way.
17. Load and execute the other example program *io.example.s*. This shows you how to read integers from the console and how to print characters to the console window.

## Main Assignment:

Your assignment is to write a program in MIPS assembly to perform the transpose of a matrix. You will recall that the transpose B[][] of a matrix A[][] is one where B[i][j] = A[j][i].

1. Write a SPIM program that will compute the transpose of the matrix stored in row major order starting at the location labeled Original in the starter template below. The transposed matrix should be stored starting at the location labeled Second. The arrays we use will be 4x4 elements. Here is the program template to help you get started.

```
.data
# This is the start of the original array.
Original:   .word  200,  270,  250,  100
            .word  205,  230,  105,  235
            .word  190,   95,   90,  205
            .word   80,  205,  110,  215

# The next statement allocates room for the other array.
# The array takes up 4*16=64 bytes.
#
Second:     .space 64
```

```
.align 2
.globl main
.text
main:    # Your fully commented program starts here.
Exit:    li $v0, 10      #terminate program
         syscall
```

2. The TA will just execute the program and look at the data segment for the transposed matrix.

**Suggestions:**

The following steps are recommended to complete the assignment.

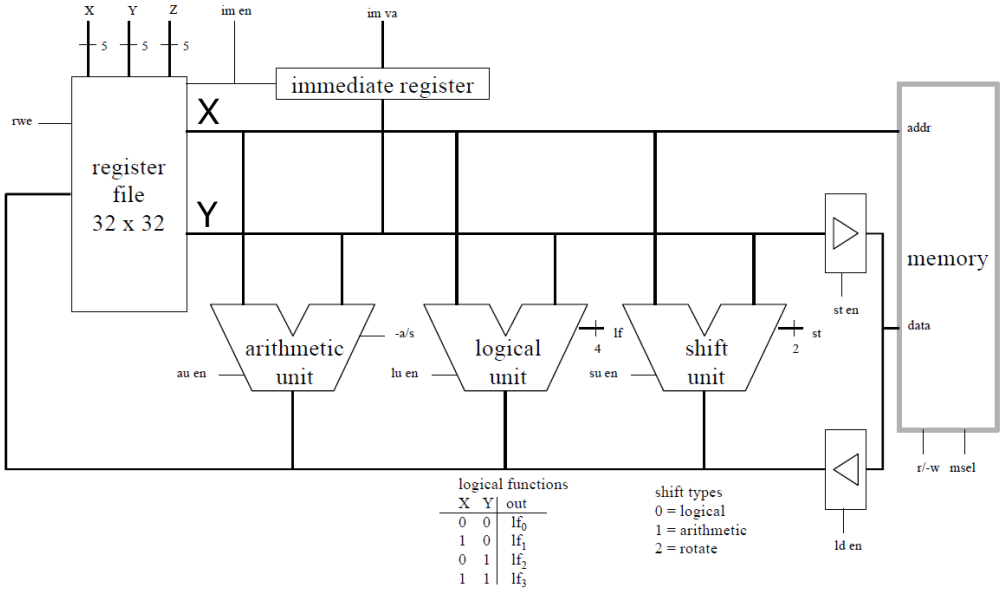
- First understand the layout of the matrix in memory. How do addresses differ between elements of a row or column?
- Perform register allocation: decide which registers you will use and for what purpose, e.g., loop count. Look at the examples.
- Refine the algorithm description in terms of registers and memory addresses
- Write and test blocks of code first. For example, test a loop for copying a row from the matrix A to a column of matrix B
- Use breakpoints to help in debugging.
- Use single step mode as the final resort in debugging. It is time consuming but gives you the most information.

## Grading Guidelines

This assignment is worth 10 points and counts towards 1% of the total grade. You just need to turn in a correct solution and working code. Minor errors will not be penalized. The precise definition of minor is at the discretion of the TAs. The goal is to refresh your memory from ECE 2035. Incorrect submissions will be penalized 5 pts each for Parts I and II. Note that the remaining assignments in the course will be worth 390 pts.

## Submission Guidelines:

All program submissions should be electronic via Canvas. Create a single PDF file with Part I and Part II.


X
Y
Z
im en
im va
5
5
5
immediate register
rwe
register file 32 x 32
X
Y
addr
memory
data
st en
arithmetic unit
-a/s
au en
logical unit
lu en
lf
4
shift unit
su en
st
2
r/-w
msel
ld en
logical functions
X Y out
0 0 lf0
1 0 lf1
0 1 lf2
1 1 lf3
shift types
0 = logical
1 = arithmetic
2 = rotate