University of Western Ontario, Computer Science Department
CS3350B, Computer Organization

Assignment 3 **Due date 17, March,2024**

General Instructions: This assignment consists of 5 pages, 4 exercises, and is marked out of 60. For any question involving calculations you must provide your workings. You may collaborate with other students in the class in the sense of general strategies to solve the problems. But each assignment and the answers within are to be solely individual work and completed independently. Any plagiarism found will be taken seriously and may result in a mark of 0 on this assignment, removal from the course, or more serious consequences.

Submission Instructions: The answers to this assignment are to be submitted on OWL as a **single PDF file**. Ideally, the answers are to be typed. At the very least, clearly *scanned* copies (no photographs) of hand-written work. If the person correcting your assignment is unable to easily read or interpret your answer then it may be marked as incorrect without the possibility of remarking.

Useful Things:

A MIPS simulator, *spim*: http://pages.cs.wisc.edu/~larus/spim.html

List of MIPS instructions: https://inst.eecs.berkeley.edu/ cs61c/resources/MIPS_help.html

Example MIPS code which runs on spim is provided in OWL under resources.

Labels can be used in assembly in replace of calculating exact values for branch and jump instructions. The following is an example.

```
int isNeg(int a0) {
    if(a0 < 0) {
        return 1;
    } else {
        return 0;
    }
}
```

```
isNeg:
    slt  $t0 $a0 $0
    beq  $t0 $0  isPos
    addi $v0 $0  1
    jr   $ra
isPos:
    add  $v0 $0  $0
    jr   $ra
```

When translating code to and from assembly, one should usually consider local variables as being stored in registers and memory accesses as being done through pointers or arrays.

```
int loadEx(int* a0) {
    int t0 = a0[0];
    return t0;
}
```

```
loadEx:
    lw   $t0 0($a0)
    add  $v0 $t0 $0
    jr   $ra
```

Exercise 1. [10 marks] Consider the following MIPS function, `exer1`.

```
exer1:
    lw   $t0, 0($a0)
    lw   $t1, 4($a0)
    lw   $t2, 0($a1)
    lw   $t3, 4($a1)
    slt  $t4, $t0, $t2
    slt  $t5, $t1, $t3
    and  $t4, $t4, $t5
    beq  $t4, $0, foo
    sub  $t3, $t3, $t2
    j    bar
foo:
    sub  $t3, $t2, $t3
bar:
    sw   $t3, 0($a0)
    jr   $ra
```

Translate the MIPS function into C code. You may use any variable names you wish in place of register names. Ensure the function has the correct declaration. Assume the data type `int` in C is 4 bytes.

Exercise 2. [14 marks] Consider the following C functions, `exer2a` and `exer2b`.

```
int exer2a(int v) {
    int i = 0;
    while (v > 0) {
        v = exer2b(v);
        i++;
    }
    return i;
}

int exer2b(int v) {
        int t = v + 1;
        return t >> 2;
}
```

Translate the C functions into MIPS assembly code. Some important things to consider:

- Ensure that you adhere to the semantics of each register (e.g. saved registers vs temporary registers).
- You will need to use the stack. Review it's usage in the `sort` function of the lecture slides.
- You may find the instruction `blez` useful. `blez $reg, label` will cause the program to branch to `label` if the value stored in `$reg` is less than or equal to 0.

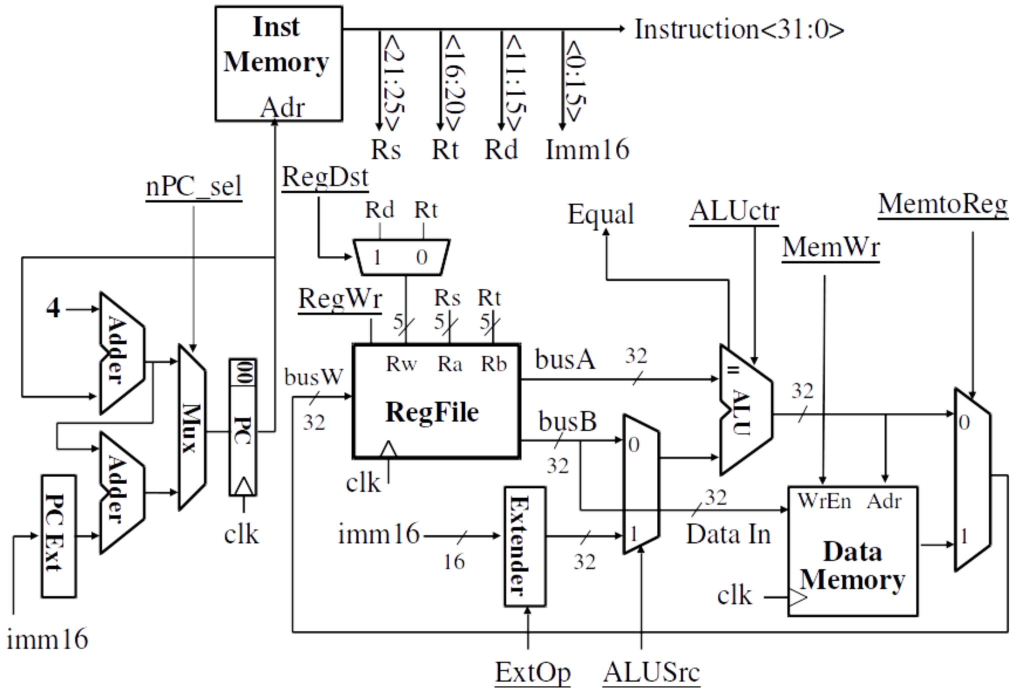

Figure 1: MIPS datapath with control signals

Exercise 3. [**16 marks**] Consider the MIPS datapath with control signals as presented in Figure 1. For each of the below instructions, give the values of the control signals required to execute that instruction. You may use "x" to denote a "don't care" value. You may use the semantic meaning for each control signal (e.g. ALUCtr = "add").

(a) `slt $s1 $t3 $t5`

(b) `bne $s2 $s3 $t1`

Exercise 4. [20 Marks] Consider again the single-cycle MIPS datapath with control signals as presented in Figure 1. We want to add a new instruction to the MIPS instruction set architecture: `foo`. Its specification is as follows:

MIPS assembly	RTL
`foo $rd, $rs, $rt`	Reg[\$rt] ← Mem[Reg[\$rd]] Mem[Reg[\$rd]] ← Reg[\$rs] + Reg[\$rt] PC ← PC + 4*

*Note that this is the normal "PC + 4" as for any non-branch, non-jump instruction.

Your task is to modify the MIPS datapath so that it can fulfill this new instruction `foo`. To do that, you should:

(i) add new wires, ports, circuitry, MUX, control signals, etc. to the datapath so that it can execute the new instruction `foo` (see, e.g., Slides 16-23 in L11-CPUControl);

(ii) ensure that any newly added circuitry and control signals do not hinder the execution of any existing operations in the MIPS ISA (i.e. your modified datapath should still be able to successfully execute all the preexisting instructions in the MIPS ISA).

Assume that memory is fast enough to read and write within one clock cycle, and that the read from memory occurs before the write to memory. To structure your solution to this exercise, perform the following:

(a) Modify (using Photoshop, Gimp, OneNote, etc.) the MIPS datapath (supplied as `MIPSDatapath.png`) with new circuits and control signals so that it can perform the `foo` instruction. That is, simply draw your new circuits and wires on top of the original image. Add your modifications in a colour other than black so they are clearly distinguishable from the original data path. Ensure you include bit-widths and labels.

(b) Give a brief English description of the modifications you have made. Explain how data flows through the modified datapath when the `foo` instruction is being executed. Explain how data flows through the modified datapath when any instruction besides `foo` is being executed.

(c) Give the values of the control signals required to execute your instruction. That is, when executing the `foo` instruction, give the values of the 8 preexisting control signals as well as the values of any new control signals you have added.

(d) If you have added any new control signals, give the values of these control signals when any instruction *besides* `foo` is being executed.