CDA 4205L Lab #2: Abstraction and Tradeoffs

University of South Florida

Lab Date: Jan. 22

Report due: One week after the lab section, see Canvas

Welcome to CDA 4205L Lab #2! The purpose of this lab is to help you better understand abstraction in computer architecture, and how different hardware designs can impact power, performance, and area.

Prelab

Consider the different layers of abstraction in a computer system. At the highest level, an application (e.g., sorting a list) can be implemented in a number of different ways. You can use bubble sort, heapsort, quicksort, and many other algorithms. Algorithms can be implemented in a programming language of your choice, and, as long as an appropriate compiler or interpreter exists for your target system, you can execute your program on that target.

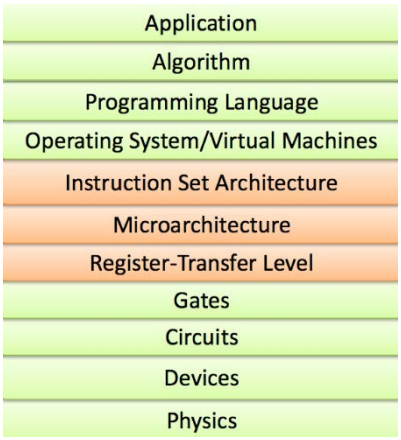


Figure 1: Layers of abstraction in a computer. The middle represents layers addressed by computer architecture.

Computer Architecture considers the levels of abstraction between the operating system and gate-level implementation of circuits. This comprises the instruction set architecture (ISA), the microarchitecture, and the register transfer level. The ISA defines the interface between the hardware and software; in other words, it defines how the processor is controlled by the software, and what the processor is capable of doing. The microarchitecture defines a *particular* implementation of an ISA. Just as there are many algorithms that can achieve the same goal (e.g., sorting a list), many different microarchitectures can implement the same ISA. The microarchitecture defines structures like the ALU, register file, and connections between them. In turn, each of these structures can be defined in many different ways – this is the *register transfer level*. This defines how data move between registers, and how the data is transformed in the process.

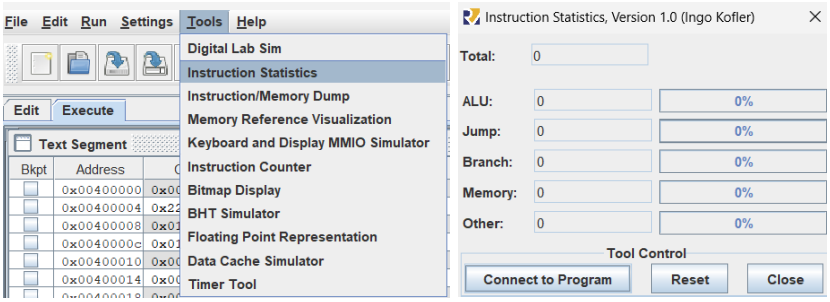
In this lab, we will consider an ISA that defines a "multiplication" operation. This operation takes two operands, multiplies them together, and generates a product. For now, we will assume that the inputs are 32-bit words, but the input values are only ever 16-bits (at most). Hence, the result will fit into one 32-bit product register. Although the ISA defines the "multiplication" operation, it does not specify how the multiplication happens, just that it does. Two different microarchitectures exist which implement the ISA. In one, there is a dedicated multiplication circuit, and in the other, it must perform repeated addition, i.e., when the inputs are 4 and 5, the hardware will add 4 to itself 5 times, taking 5 cycles to complete. While the multiplier is physically larger (it takes more logic gates) and more energy, the repeated addition is slower and takes less energy *per addition*. Hence, for certain applications, you may decide to use uArch_1 or uArch_2, depending on the power, performance, and area budget.

Lab.

1. Download the uArch_1.asm file from Canvas (Files > Labs > Lab 02). This contains the skeleton of the first assembly program you will write.
2. Implement the function labeled "_multiply". Recall that in uArch_1, there is a physical multiplier circuit. You can use this feature by writing the "mul" instruction.
3. Set a0 to 550 and a1 to 21, assemble, and run the code. Note: you may modify a0 and a1 values while debugging but your final results must use 550 and 21, respectively.

   T1: Take a screenshot of the RARS console output for both files. Include this in your report.

4. Open the Instruction Statistics tool and click on "Connect to Program"



5. Re-run your program. You should see the total number of instructions, along with a breakdown of different instruction types.

   T2: How many total instructions were there? How many of each type?

6. Repeat 1-5 using the file uArch_2.asm. Recall that this represents the second microarchitecture which *does not* have a dedicated multiplication circuit and instead uses repeated addition. Hence, your implementation in #2 is not allowed to use a "mul" instruction (including mulh, mulhu, etc.).

   T3/T4: Remember to repeat T1 and T2 for uArch_2 and include the results in your report.

7. Answer the following. Show your work and explain your result.

   T5: Assume the implementation of uArch_1 has an area of 8 um^2, and the implementation of uArch_2 has an area of 7 um^2. What is the area overhead for the multiplier? Report your result both as a raw value as well as a percent overhead.

   T6: Assume the multiplication itself consumes 500 pJ of energy, whereas other ALU instructions (those used for arithmetic such as add or addi excluding branch/comparison/etc.) consume 4 pJ each. If performance and area are not considered, at what point is it better to use uArch_1 rather than uArch_2 in terms of energy?

   T7: One important metric when evaluating different microarchitectures is *energy efficiency*, often reported as the Energy Delay Product, or EDP. This is the total delay multiplied by the total energy to perform the operation. Using the values provided in T6 and a 500ps CPU clock period, compute the EDP for uArch_1 and uArch_2 when performing $21*21=21^2$.

In-class work - Complete at least **T1-T4**, show them to the TA for review and submit them on Canvas before the lab session ends. Submit your in-class work as a Word document (.docx) format.

Final-Report - Submit your report pdf with answers to all questions and screenshots. Also, submit the final assembly file(s). Combine these in a .zip file. It is due in 1 week, before the next lab.