Project 1: RISC-V Assembly Language

UW-Madison ECE 552: Fall 2025

Project Introduction

In this project phase, you will refresh your memory on **assembly programming** and gain exposure to the **32-bit RISC-V (RV32I)** instruction set architecture (ISA) which we will use in this course. You are expected to have prior experience writing and debugging **C and assembly languages** at the level of ECE/CS 354 and to have familiarity with digital logic at the level of ECE/CS 252 for this assignment.

Solve each coding problem in the specified language and answer the free-response questions associated with each problem (provided near the end of this document). This assignment should be completed **by yourself**. You may discuss the problems with other students but must work on your own answers.

You are **permitted** to use generative artificial intelligence (e.g. ChatGPT); if you choose to do so, please indicate that in your submission. Please ensure your usage of LLMs adheres to the course policies. We have reviewed the answers given by AI by copying and pasting the questions into various LLMs.

You are **encouraged** to use the Internet to answer the questions provided, but please do not share answers online.

Problem 1: Unsigned Multiplication in RV32I Assembly

Using the RISC-V RV32I ISA reference sheet provided in class and online, write an assembly routine to perform **multiplication between two unsigned 32-bit numbers** and output the **lower 32 least significant bits** of the product as a **32-bit unsigned result**.

You are **not allowed to use any of the RISC-V multiplication instructions** (`mul[h][s][u]`), which are not present in RV32I and not described in the reference sheet. You may use any of the instructions that do appear in the reference sheet (e.g. `add`). **Do not use any pseudo-instructions** such as `mv` or `ret`. There is **no maximum number of instructions** you may use, but you are strongly encouraged to write the most compact (and fast) implementation you can.

Note: your implementation **should not be linear in complexity** with respect to the actual numbers provided. Therefore, an implementation that multiplies **a × b** by adding **a** to itself **b** times is not acceptable, as it is much too slow.

See the starter code provided under **`starter/project1/umul.s`** in the WISC25 public git repository. You can clone the repository and edit **`umul.s`** locally – the other files are for building and grading, and need not be modified. Note: you need to be on the UW network or connected over the VPN to clone from GitLab; if you don't want to do this, you can copy the files to a local directory instead.

You can debug your code using a simulator such as Venus. Use "step" to execute assembly code one line at a time, or use "run" to run all lines of code at once. Some simulators use breakpoints to allow for more flexibility in debugging.

To run the grader locally, a docker image has been provided. You can run the following on a Linux box such as the CAE Linux machines (macOS and Windows can run docker with more setup work, but it's more complicated and not recommended). Note: the docker image expects the provided **`Makefile`**, **`harness.c`**, *and* **`grader.py`** *to be present in the project code directory. If you chose to copy the code instead of cloning, make sure to copy the test files as well.*

```
$ cd path/to/project1/code
$ docker run --rm –v $(pwd):/project1 -it coderkalyan/ece552-
tools:latest bash
root@530481afd799:/# cd /project1
root@530481afd799:/project1 # make test-umul
```

Problem 2: Dot Product in C

Write C code for a function that takes in **two arrays as inputs** and **returns their dot product as an integer**. That is, given two input arrays **`A[N]`** and **`B[N]`**, the program should compute output = **sum**(**`A[i] * B[i]`**) for **`i`** in **`[0, N)`**. **Your code should utilize the multiply routine you developed in problem 1**; this can be achieved by calling the extern function provided in the starter code.

See the starter code provided under **`starter/project1/dot.c`** in the distribution git repository.

The compiler toolchain in the docker image provided is also capable of compiling and linking this C program with your previous assembly. Try building and running the tests:

```
root@530481afd799:/project1 # make test-dot
```

Problem 3: Mystery Assembly Program

Read the following mystery routine implemented in RISC-V assembly and answer the following questions about what it does.

Try stepping through this code using a simulator such as Venus and passing in function arguments using register **a0** (per the standard RISC-V calling convention) to store the first function argument and obtain the return value. The register **ra** is used for the return address.

The code is also available under **starter/project1/mystery.s** *– if you want, you can compile it and try running it on different values. While the process is the same, we haven't provided a* **make** *target for mystery – we encourage you to look at the file and run* **gcc** *appropriately inside the docker image!*

```
main:
    addi t5, zero, 8
    slli t5, t5, 2
    sub  t0, a0, t5
    slli t1, t0, 2
    add  t1, t1, t0
    addi t2, zero, 0
    addi t3, t1, 0
    addi t4, zero, 9
loop:
    blt  t3, t4, done
    sub  t3, t3, t4
    addi t2, t2, 1
    beq  zero, zero, loop
done:
    addi a0, t2, 0
    jalr zero, 0(ra)
```

Questions

Partial credit will be given for answers based on how close they are to the correct answer.

Problem 1

[Q0] Reflect: Is the result in a0 always the **correct and complete product** of the two 32-bit input numbers? Why or why not?

[Q1] Think: Can you think of any other arithmetic operations not listed on the reference sheet that could be implemented using RISC-V assembly code? **Multiplication and division are not allowed as answers.**

[Q2] Think: Multiplication can also be implemented in hardware by creating a `mul` instruction. What might be the benefits and costs of introducing a hardware multiplication instruction? Describe **at least one benefit and one cost/downside**.

[Q3] Research: Recall the IA-32 (32-bit x86) assembly language you learned in ECE 354 (**Wikipedia**). Describe **3 differences** between RV32I and IA-32. **Do not simply list terminology** (like CISC or RISC) without describing what it means or providing an example.

Problem 2

[Q4] Think: Provide a brief description of the **differences** between a high-level language such as **C** and a low-level language such as **RISC-V**. In what situations would you use one or the other?

[Q5] Research: What **software components** allow the C code you write to be realized as machine instructions that can be executed by a processor? How might their design impact overall performance?

[Q6] Research: Adding a hardware multiplication instruction would improve performance for certain programs that utilize this heavily, such as dot product. What would be the effects, if any, of adding this optimization to a CPU on other programs that **do not require multiplication as extensively?**

Problem 3

[Q7] Reflect: What is this program doing (be specific as to the equation this implements)?

[Q8] Think: Explain how you would modify this assembly code to produce the opposite effect.

[Q9] Research: A simple arithmetic program such as this can still run faster by means of hardware improvement. Name one hardware improvement you could make that would make this particular program run faster.

Submission

Submit the following files to the appropriate Canvas assignment:

Deliverable	Points	Notes
umul.s	10	See problem 1.
dot.c	10	See problem 2.
project1.txt	20	**Submission Template** *Author: your_name* *[Q0]: answers[0]* *[Q1]: answers[1]* *...* ai_statement

If you **have used LLMs**, include the following in project1.txt:
I certify that my usage of LLMs is consistent with UW-Madison's academic integrity policies.

Otherwise, include the following:
I certify that I have not used LLMs to complete this assignment in any shape or form.

a0 = Whether you used LLMs (1) or not (0)

```
beq a0, zero, no_llm
llm:
    mv ai_statement, used_llms
    beq t0, t0, add_to_project1
no_llm:
    mv ai_statement, did_not_use_llms
add_to_project1:
    addi project1.txt, project1.txt, ai_statement

.data
used_llms: .asciiz "I certify that my usage of LLMs is consistent with UW-Madison's academic integrity policies."
did_not_use_llms: .asciiz "I certify that I have not used LLMs to complete this assignment in any shape or form."
```

The results of your submission will be made available after the deadline.