CSCI 2500 — Computer Organization
Group Project (document version 1.2) — Due December 8, 2023
How do computers actually compute?

- This project is due by the midnight EST on the above date via a Submitty gradeable.
- This project is a *group assignment*. You can have groups of up to five people.
- This project is out of 100 points. All students in a group receive the same grade.
- Start the project early. You can ask questions during office hours, in the Submitty forum, and during your lab session.

1 Project Overview

For the group project, you will design a computer architecture, test it using a simple benchmark consisting of two programs, and compare to another architecture.

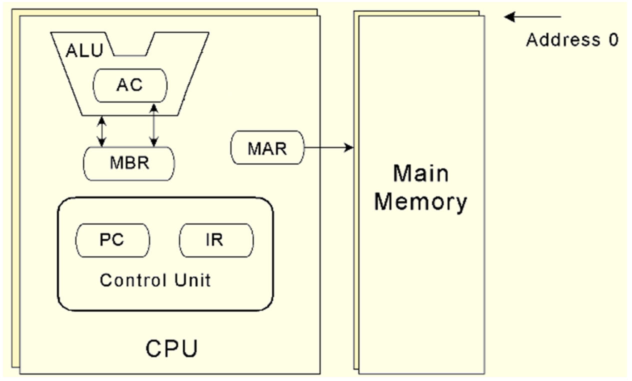
Note that this project is meant to be exploratory. Your team will have a fair amount of freedom in making certain choices but it also means that there is a number of decisions to make. Make sure you document your entire thought process and justifications for any decisions that you will be making.

When designing your computer architecture, please remember the Seven Great Ideas in Computer Architecture from our textbook. In particular, make sure that your design is minimal, i.e., only contains features that are necessary, and high-performance.

2 Design 1

First, you will need to design a simple computer with accumulator architecture similar to the one we reviewed in class. We will call this Design 1. The main components of the architecture are shown in Figure 1.

Figure 1: Main components of the accumulator architecture



Here is a brief description of the components:

- AC or Accumulator: intermediate data is stored within the AC

- PC or Program Counter: as the name suggests it counts the current position of the code, each line has its own address; PC needs to be incremented after each instruction
- MAR or Memory Access Register, stores or fetches the 'data' at the given address
- MBR or Memory Buffer Register, stores the data when being transferred
- IR or Instruction Register, stores the instruction word of the currently executing instruction
- ALU or Arithmetic and Logic Unit, performs arithmetic and Boolean operations
- Main memory, stores data and instructions

To simplify your design, you may assume the following:

- No need to support floating-point operations, only integer operations are supported.
- All integers are assumed to be in 2's complement notation.
- All memory is byte-addressable.
- No need to support any I/O, i.e., assume there are no peripheral devices (keyboard, screen, etc.) or file storage.
- No need to support dynamic memory allocation.
- Instructions and data are loaded into instruction and data memory before program is executed.
- There is only one program that runs until it encounters a `Halt` instruction. Once program halts, the CPU starts idling by fetching the same "Halt" instruction without advancing the `PC`.
- No need to support exceptions or interrupts.

Your ISA needs to support at least the operations (additional operations can be implemented for extra credit) given in Table 1. Note that all instructions in our Design 1 have at most one operand. While designing your first computer you will need to explicitly document each design decision that you make. For example, you will need to specify which endianness your computer is using and justify this decision.

Operation	**Microoperations**	**Description**
add X	MAR ← X # load X into MAR MBR ← M[MAR] # load value stored at address X into MBR AC ← AC + MBR # add value in AC with MBR value and store it back into AC	Adds value in AC with the value at address X and stores the result into AC
halt		Ends the program

load X	MAR ← X # load X (address) into MAR
	MBR ← M[MAR] # load value stored at address into MBR
	AC ← MBR # load value in MBR into AC
		Loads the value from address X into the AC

store X	MAR ← X # load address into MAR
	MBR ← AC # load AC value into MBR
	M[MAR] ← MBR # writes MBR value into the Memory of address indicated by the MAR
		Stores the current value from the AC into address X

Operation	Microoperations	Description
clear	AC ← 0	Writes 0 into AC
skip C	If (C), PC ← (PC) + 1	Skips the next instruction based on C. if (C) is: 0: Skips if AC < 0 2: Skips if AC = 0 4: Skips if AC > 0
jump X	PC ← X	Jumps to address X

Table 1: Operations that need to be supported by Design 1.

You can implement additional operations for extra credit (5 points extra credit for each additional instruction).

Operation	Microoperations	Description

subtract X	MAR ← X
	MBR ← M[MAR]
	AC ← AC - MBR
		Subtracts the value at address X from the value in AC and stores the result into AC

and X	MAR ← X
	MBR ← M[MAR]
	AC ← AC and MBR
		Performs Boolean ”and” on the value in AC and the value at address X and stores the result into AC

or X	MAR ← X
	MBR ← M[MAR]
	AC ← AC or MBR
		Performs Boolean ”or” on the value in AC and the value at address X and stores the result into AC

not	AC ← not AC	Performs Boolean ”not” on the value in AC and stores the result into AC

jump with linking X	MAR ← X # loads value X into MAR
	MBR ← PC + 1 # loads value of PC into MBR
	M[MAR] ← MBR # stores value in MBR into address of MAR
	AC ← X + 1 # increments X by 1 and stores it into AC
	PC ← AC # jumps program counter to address indicated by AC
		Stores PC at address X and jumps to X+1

return X	MAR ← X # loads value X into MAR MBR ← M[MAR] # loads value stored at address X into MBR MAR ← MBR # loads value back into MAR MBR ← M[MAR] # fetches the value at the address into MBR PC ← MBR # loads the value into PC	Uses the value at X as the address to jump to

Table 2: Operations that can be supported by Design 1 for extra credit.

While designing your first computer, you will need to determine the clock rate at which the CPU will run. You may assume that your main components have latency specified in Table 3. For all other components that you need to design, estimate their latency based on the information given in Table 3.

Component	**Latency**
DRAM	20 ns
SRAM	2 ns
Registers or register file	1 ns
Basic logic gate	0.25 ns

Table 3: Memory and basic gate latency.

Your Design 1 must comply with specifications given in Appendix A for your "Team ID' but:

- Have no cache
- Use a non-pipelined datapath
- Implement only one addressing mode: direct addressing

3 Improving Design 1

Once you are done with your initial design, you need to make certain improvements so that you can then compare the performance of different versions of your architecture. Note that each subsequent version builds on the previous one.

3.1 Design 1a

Add an additional addressing mode to your Design 1 according to specifications given in Appendix A for your "Team ID'. Please note that it is crucial that you select the correct row from the specifications given in Appendix A. `Team␣ID␣mod␣32` means that you need to take your team ID, divide it by 32 and take the remainder. E.g., if your Team ID is `00141_kuzmik2`, then you compute 141 mod 32 which is 13 and take the corresponding row `13|32|1Gi|256Mi␣x␣8|␣1,200,000|Indirect` from the table. Teams that do not implement their assigned version will incur a 25-point penalty.

3.2 Design 1b

Add data cache to your Design 1a according to specifications given in Appendix A for your "Team ID'. Choose organization and policies of cache as you see fit but justify all decisions made. Remember that your cache should fit within the max number of bits (not bytes!) specified in Appendix A.

3.3 Extra credit 5 points Design 1c

Replace instruction and data memory from your Design 1b with separate memory for instructions and data. Add instruction cache and choose its the size and organization as you see fit but justify your decisions.

3.4 Extra credit 20 points Design 1d

Change your Design 1b or Design 1c to implement pipelined datapath. You are free to design your pipelined datapath (e.g., determine the number of stages, deciding what additional hardware would be necessary) as you see fit but you should not alter the basic ISA of Design 1.

3.5 Extra credit 40 points Design 2

Design another computer which we will call Design 2 that is built according to specifications given in Appendix B for your "Team ID'. Feel free to make your own design decisions but justify and document all decisions.

3.6 Extra credit 30 points Vector Instructions

Implement vector extensions (at least one SIMD instruction) for your Design 1 or Design 2.

4 Implementation of your designs

You need to implement all your designs using Verilog. You are welcome to supply additional design documents like tables, figures, diagrams, etc. but providing Verilog files and waveforms (see below on testing your design) is required.

5 Simulation and testing

To test your designs you will write two programs using the assembly language that you designed for your ISA. These will be our benchmarks:

- Compute Fibonacci number F_{11} by using a loop.
- **Extra credit 10 points.** Matrix multiplication of two 8 by 8 matrices.
- **Extra credit 10 points.** Recursive implementation of factorial function. If recursive implementation is impossible, explain why and switch to the iterative implementation. Factorial function would still need to be invoked as a subroutine.

5.1 Benchmarking

Once you write your programs in assembly language, translate them into machine code and store those codes in instruction memory. Similarly, store all data that a program will be working with in data memory. We recommend manually translating these two programs because writing an assembler might be extremely time consuming. Note that since we assume that CPU only runs one program and then halts, you can treat each program as a separate test.

5.2 Verilog

Now, describe how your Verilog model simulates execution of your benchmark programs. You will need to generate waveforms and refer to them when explaining how your programs run. We also recommend that you add printing statements to your Verilog code to help you with debugging.

Please feel free to explore various Verilog tools that are available on the Web. There is a number of IDEs, online simulators, etc. that you can use. In your report, you will need to specify which tool or tools you are using. To help you get started with designing a testbench, you might want to check out the corresponding lab in zyBooks ("10.18 For instructors: creating new Verilog zyLabs" https://learn.zybooks.com/zybook/RPICSCI2500KuzminFall2023/chapter/10/section/18), although you won't be able to create an actual zyLab for your project on zyBooks. A good starting point in exploring different Verilog tools might be EDA Playground IDE (https://www.edaplayground.com/). You can also explore other resources available on the Web (like https://hardwarebee.com/ultimate-guide-verilog-test-bench/ but there are many others, of course.

Please note that your simulation has to be detailed enough to show the state of inputs and outputs of each key component of the system during each clock cycle. Feel free to use any means of representing the state that you find appropriate, e.g., tables, timing diagrams, etc. Your goal here is to demonstrate your understanding of the process of executing memory stored instructions by your Verilog model of the processor.

6 References

Your project report must include the "References" section that lists all resources that you used when working on this team project. You don't have to list every single resource that you accessed (e.g., if you opened a Web page just to see that it is not what you want) but every resource that you actually used (read, found useful, etc.) must be listed. Resources include not just our zyBook but also any other books, articles, blogs, forum posts, YouTube videos, tutorials, etc. For each reference you need to provide as much information as possible (title, author, publisher, publication date, URL, date accessed, page numbers or chapters, if it is a longer paper or a book, etc.)

7 Submission and Grading Criteria

7.1 Submission Requirements

- You MUST type up your report. Handwritten solutions will not be accepted or graded, even if they are scanned into a PDF file.
- We recommend using LaTeX (https://www.latex-project.org/). If you have never used LaTeX, you might want follow some tutorial, like this one: https://www.latex-tutorial.com/tutorials/. There is a convenient online LaTeX editor called Overleaf (https://www.overleaf.com/) that has a free plan for students.
- Submit your report on Submitty as a single PDF file named `project.pdf`.
- Submit all your Verilog files (your Verilog models, testbench files, your IDE project, etc.) and any additional files. All files need to be well organized into directories and packed into a single archived ZIP file.

Since final submission is due on the last day of the semester, it will not be possible to use any late days for the team project.

7.2 Grading Criteria

You will be graded based on the quality of work presented in the project report and supported by supplementary files (Verilog code, waveforms, diagrams, etc.) Your project report needs to:

- Fully explain and justify all design choices made for each of the design versions.
- Include all Verilog code (including testbenches) necessary to simulate your designs.
- List benchmark programs in assembly language that corresponds to the ISA that you designed.
- Describe the process of executing benchmark programs.
- Compare performance of different design versions for the given benchmark programs and explain the observed results. Specifically, for each design version and each program, you need to provide the execution time in some units of time as well as in the number of instructions and discuss how design decisions that you made affected performance.
- Explain contributions of each group member.
- Provide a "References" section that lists all resources that were used while working on this project.

Appendix A Specifications for Design 1

Specifications for the first architecture must be selected according to Table 4 based on your “Team ID”.

Team ID mod 32	Word size, bits	Main memory size, bytes	Main memory organization	Max number of bits to be used by L1 cache	Additional addressing mode
0	8	256	64 x 8	350	Indirect
1	16	64Ki	16Ki x 8	75,000	Immediate
2	16	64Ki	8Ki x 8	40,000	Indirect
3	32	1Gi	256Mi x 8	1,200,000	Immediate
4	32	1Gi	256Mi x 16	600,000	Indirect
5	8	256	64 x 8	350	Indirect
6	16	64Ki	16Ki x 8	75,000	Indirect
7	16	64Ki	8Ki x 8	40,000	Immediate
8	32	1Gi	256Mi x 8	1,200,000	Indirect
9	32	1Gi	256Mi x 16	600,000	Immediate
10	8	256	64 x 8	350	Indirect
11	16	64Ki	16Ki x 8	75,000	Indirect
12	16	64Ki	8Ki x 8	40,000	Immediate
13	32	1Gi	256Mi x 8	1,200,000	Indirect
14	32	1Gi	256Mi x 16	600,000	Indirect
15	8	256	64 x 8	350	Indirect
16	16	64Ki	16Ki x 8	75,000	Immediate
17	16	64Ki	8Ki x 8	40,000	Indirect
18	32	1Gi	256Mi x 8	1,200,000	Indirect
19	32	1Gi	256Mi x 16	600,000	Immediate
20	8	256	64 x 8	350	Indirect
21	16	64Ki	16Ki x 8	75,000	Immediate
22	16	64Ki	8Ki x 8	40,000	Indirect
23	32	1Gi	256Mi x 8	1,200,000	Immediate
24	32	1Gi	256Mi x 16	600,000	Indirect
25	8	256	64 x 8	350	Indirect
26	16	64Ki	16Ki x 8	75,000	Indirect
27	16	64Ki	8Ki x 8	40,000	Immediate
28	32	1Gi	256Mi x 8	1,200,000	Immediate
29	32	1Gi	256Mi x 16	600,000	Immediate
30	16	64Ki	8Ki x 8	40,000	Indirect
31	32	1Gi	256Mi x 8	1,200,000	Indirect

Table 4: Specifications for Design 1.

Appendix B Specifications for Design 2

Specifications for the first architecture must be selected according to Table 5 based on your "Team ID".

Team ID mod 4	**Architecture**	**Addressing modes**
0	General purpose register	Register addressing, immediate addressing, register indirect addressing
1	General purpose register	Register addressing, indexed addressing, register indirect addressing
2	General purpose register	Register addressing, based addressing, register indirect addressing
3	Stack architecture	Stack addressing

Table 5: Specifications for Design 2.