CSE 332: Computer Organization and Architecture

Instructor: Dr. Mohammad Abdul Qayum (MAQm)

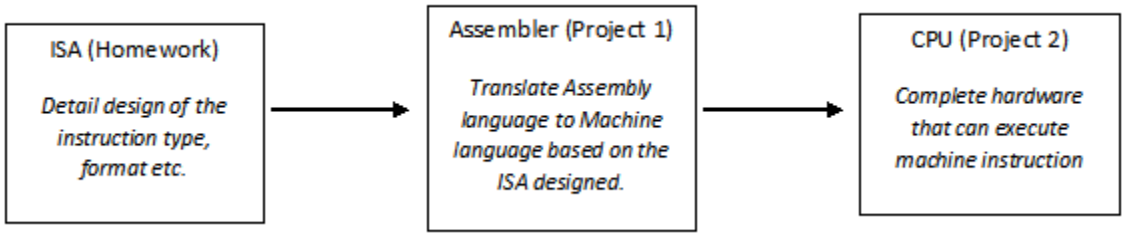
Office: SAC 1044 A

E-mail : mohammad.qayum@northsouth.edu

Phone: 02-55668200 Ext. 6186

Design Your Own CPU and Own OS

One of the course objectives is to be able to design a complete CPU system. It starts with learning about ISA, assembly language, advanced ALU, data path and control, pipelining in theory. In order to make this learning more effective a step by design of the complete CPU system is carried out through pen and paper, simulation tools, simulated hardware etc. Students work in a group and try to develop small pieces and finally connect them together to get their cpu with OS as explained in the following diagram.



Title :ISA study Type: Document and Program

Consider yourself as a computer architect and you are employed in a CPU design company like Apple, Intel or AMD. The company told you that they are going to design a new 32 bit single-cycle CPU that has separate Data and Instruction Memory. The ISA should be general purpose enough to be able to run provided general programs

Input/Output Operations

Since we cannot connect any external display units (e.g. seven segment display or LED) to display results or any data and a keyboard or something similar to get input from the user. you should make any necessary arrangements (*e.g. screenshot or register files, data memory*) to accommodate this external communication into your ISA design. For example, you might consider including dedicated instructions like LW and SW to perform the input/output operations to load to or store data to specific memory addresses in the Data memory unit.

Design requirements

As an ISA designer your job may have to propose a detailed design of the ISA. How many operands should each instruction support to maintain a balance between simplicity and functionality? Should the operands be register-based, memory-based, or perhaps a mix of both to allow flexibility in instruction execution? How many operations are necessary to cover basic computational, logical, memory, and control functionalities without making the instruction set overly complex? What types of operations should be included—arithmetic, logical, branch, memory—and how many from each category would be sufficient? Could a table be created listing each instruction along with its type, opcode, and functionality to better visualize the design? How many instruction formats would be appropriate for this architecture, and what should each format look like in terms of field names and bit allocation? Should we use R-type for register operations, I-type for immediate or memory-based operations, and J-type for control flow instructions? What should be the length of each instruction in bits, and how should the bits be distributed among opcode, registers, immediate values, or addresses? How many general-purpose registers are needed for basic tasks, and what should they be named—perhaps R0 through R7? Could a register table be constructed showing the register names and example values to illustrate how they are used in practice? **Discuss them in perspective of MIPS ISA.**

Benchmark Programs

You have to study your ISA focusing on the following three categories of programs: Simple arithmetic & logic operation, programs that require checking conditions and loop type of program. It means that you are writing an Operating system for your CPU just writing programs using ISA.

Guideline

Your task of the benchmark program implementation will be evaluated according to the following criteria:

1. Ability to execute the provided benchmark programs (test programs), and other general purpose programs?

2. Discuss how ISA is used to guide your writing the program.

You must answer those with your reasoning. While you are deciding on the above issues, you might consider some sample high level program (such as C) that can be run on this CPU using your ISA. Say, during the decisions about the types of operation to include, you can think about the type of high-level language program it will be able to execute. Analysis might vary from one group to another and there might be multiple possible solutions. You will be scored based on your clear reasoning.

Documentation : You must prepare your own documentation to include this discussion.

Title : Assembler Type : Software

It is difficult and error-prone to manually write machine code. The problem can be addressed by writing an assembler, which can automatically generate a machine code from an assembly file. In this project, you will be provided an assembler for your ISA. The assembler reads a program written using assembly language in a text file (.asm or .s), then translates it into binary code and generates an output file (.txt or .bin) containing machine code. The generated output files will later be useful to run a program when you will develop your actual CPU in verilog.

Language:

You can use any high-level language such as C++ or Java to develop assembler. Some demo codes are provided in the following https://github.com/RoySRC/UpgradedMIPS32Assembler.git (the model output is binary that includes address, you will have automatically generated binary without address called no_adress.text.bin and no_adress.data.bin). You are strongly advised to use the example (Inttest2.s) to test your CPU to save your time. You might need to modify the existing functions/classes to fit your needs.

Moreover, you can take advantage of AI such as ChatGPT or Gemini to develop your own assembler.

I/O fromat:

The input code will be written in a text file in assembly format following your ISA. There will be one instruction per line. The output will be generated in Binary format or hexadecimal format. This will be helpful for us to later transfer this code into the RAM block of the verilog model.

Documentation : You must prepare your own documentation to discuss how you build, compile and test your assembler

Title : Operating System based on MIPS assembly Type : Software

It is challenging to write a whole or even a small operating system for a CPU which does not have IO support. However, we can write a small program in assembly language using MIPS ISA to run our own built CPU to test its capabilities. In this case, you will write a program that has a main function and call other other three functions that calculate MAX, MIN and MEAN. Choose ten integers carefully so that you can track all the parameters. Since your CPU will not have any divide function, a MIPS divide subroutine will be provided. You have to use lots of JAL and JR instructions to implement functions.

Language:

You will use assembly language based on MIPS ISA. You can use your assignment codes as reference. You are strongly advised to use the example to save your time. You might need to use three functions such as MIN, MAX and MEAN to fit your needs. Moreover, you cannot use a data segment for your initial 10 numbers. You can use instructions like “li, load data in the register and then use “sw” to store data in data memory.

Also, you can take advantage of AI such as ChatGPT or Gemini to develop your own OS

I/O fromat:

The input code will be written in a text file in assembly format following your ISA. There will be one instruction per line. The output will be generated in Binary format or hexadecimal format. This will be helpful for us to later transfer this code into the RAM block of verilog model in model sim.

Documentation : You must prepare your own documentation to discuss your program written in Assembly language using MIPS ISA

Title : Full CPU Type : Verilog

In this part of the project you have to design a Datapath and Control path of your proposed 32-bit architecture. Datapath must have all the necessary components. The components must be adequately connected.

1. A complete hardware based on 32 bit ISA - *Students will be designing the hardware parts such as ALU, Register Files, Memory, Control Unit and connect them. There will be few simulation labs throughout the semester using LogiSim. The work in simulation labs will be helpful in understanding this part. Sample Verilog files of a completed CPU without JAL and JR instruction will be provided. You can use them and make modifications based on your own ISA.* ***However, you must add JAL and JR, SRL, SLL instructions to the model.***

2. Implement datapath and control – With this the hardware can execute instructions automatically.

2. Testing of the complete design-

This information explains to you how I am going to test your 32 bit processor once you have finished building it completely and submit. So before submission, you can test the design yourself using the same method shown in the class. Modify the $readmemb/$readmemh command to read binary or hex machine code files. Your machine files will be generated from your assembly program provided with this project

Step 1: Write a piece of simple assembly level code (representing the solution of a specific problem) that includes a calling and called procedure and translate it into MIPS assembly code manually.

Step 2: Compile the Assembler, run the executable of the Assembler to test the assembly file (e.g. instest.s) to generate the machine code output.txt. The machine code must be generated in Hex/binary format. This is basically another text file but instead of containing the instructions in binary or hex values. The model output is binary that includes address, you will have an automatically generated binary without address called no_adress.bin. It will contain the converted (bin or hex) machine code value. The values in the text file must be written in the text file in the following format:

```
a9b22714
29cb1011
2004001b
38850005
00853020
00a43822
00e6402a
00e04825
00e05024
00075840
00076042
```

```
00100000000001000000000000011011
00111000100001010000000000000101
00000000100001010011000000100000
00000000101001000011100000100010
00000000111001100100000000101010
00000000111000000100100000100101
00000000111000000101000000100100
00000000000001110101100001000000
00000000000001110110000001000010
```

Step 3. Open ModelSim, create a project, add all your verilog files and instruction files (binary or hex). Then compile them. If all files compile correctly, then go to the next step.

Step 4. Simulate the program so that instructions are started to read from instruction memory of your processor. The file containing instructions in hex or bin format will update the instruction memory of your own datapath.

Step 5. Run the program on your processor. And stop after some time.

Step 7. Check the result in register files, data memory file and instruction memory file. The result of the program will be written in the data memory and register. So check whether the data memory and register file of the processor was able to run the program correctly.

You must answer those with your reasoning. While you are implementing instructions such as JAL and JR, what challenges you faced. What are the changes in the control path and data path you have applied? Discuss all units in the complete CPU and link them to JAL and JR instructions.

Documentation : You must prepare your own documentation to show screenshot of data memory where results of MIN, MAX and MEAN are stored. Also, results of the initial assembly program test stored in the register file. You have to provide all verilog codes, assembly codes, and writing in the same documents. Moreover, you also have to submit each file in a folder and zip/rar it to upload to the canvas.

```
# Division Subroutine

divide: move $t0, $a0      # Load dividend into $t0
        li $t1, 2       # Load divisor into $t1
        li $t2, 0           # Initialize quotient to 0
       # Loop for division
```

```
dloop:

    sub $t0, $t0, $t1  # Subtract divisor from dividend
    bgez $t0, increment_quotient  # If dividend >= divisor, increment quotient
    j end_division



increment_quotient:
        addi $t2, $t2, 1   # Increment quotient
        j dloop

end_division:
            move $v0, $t2
            jr $ra
```