CSS3
ISA Project
Due: Finals Week

Program Description

You are assigned the task of designing a new ISA. Your ISA should be general enough to run all the provided benchmark programs, while being simple enough to allow you to implement a reasonably fast processor in the allotted time frame. Then, you will implement two essential components for architectural validation and verification: an assembler and simulator. An assembler translates your program written in your assembly language into binary code; a simulator performs a simulation upon a provided binary code. Collectively, they form essential infrastructure for your processor design. For the implementation of the assembler and the simulator, you need to use either C++, Java, or Python.

Requirements

- You must design an ISA that fulfills the following requirements.
  - Has a fixed instruction width.
  - Fixed width for data, memory addresses, and/or registers.
  - Opcode of fixed, or varying, width.
  - General purpose enough to be able to run provided benchmarks and additional general programs that you will asked to perform on the fly during the final.
  - Will likely include most of the following instructions
    - *in* [dest] : read data from the keyboard and save at [dest].
    - *out* [src] : writes data from [src] to the screen.
    - *halt* Stop execution and return control to the simulator's command prompt.
    - Math operations – Addition? Subtraction? Multiplication? etc.
    - Instruction access – Jumps? Loops?
    - Data access – immediate, direct, indirect, offset? Don't need each one.
  - You should leave opcode space for expansion in case you leave out an instruction you need in the future.
- You must develop an assembler to take your mnemonic assembly code and assemble it to machine code. It should stop assembly if your code has a syntax error. Primarily, you should look for instructions and registers that don't exist, or unlabeled memory, to stop assembly.
- You must develop a simulator to validate the correctness of your ISA by running your assembled code.
- You *must* work in teams. If you can't find a team, you'll be assigned one.

ISA

- **Design Guideline -** Ultimately your ISA and resulting implementation will be evaluated according to the following criteria:

  - **Correctness** - Is it able to execute the provided benchmark programs, and other general purpose programs from the binary code your assembler creates?
  - **Novelty** - How much is it different from the MARIE ISA? I like creativity!
  - **Design -** Are you implementing an ISA which follows computer architectural design principles that we've covered in class, using registers and memory, specific addressing, etc?

Assembler

With the assembler, you'll be taking your assembly code and converting it into machine code. This should generate a file with zeroes and ones. The easiest way to do this may be to read each line as a string, and then parse it into its appropriate binary representation.

Simulator

With a simulator, you can easily verify your ISA without actually implementing hardware, debug your application without having actual hardware, and improve your ISA by spotting performance bottlenecks in benchmark programs. Your simulator operates instruction by instruction; you must be able to execute instructions one by one and update states (eg. register, memory, ...) at a certain time when the execution of an instruction is completed. Note, this doesn't mean that you have to output the value of each register, like MARIE does, but that your simulator has variables for registers, array for memory, that if you step through the program, will contain the correct values.
Again, you'll likely be reading each instruction in as a string of zeroes and ones, an instruction width at a time, and then identify which part is opcode, which is operand, and then actually perform what the code specifies. For example, if your word length is 18 bits, you will read in 18 characters of zeroes and ones, and perhaps your ISA says that the first 6 bits are the opcode, and the next 4 are for register 1, then 4 for register 2, and then the last 4 for register 3.

Benchmarks

Two benchmarks are provided below to help guide your ISA development. You need to write assembly programs based on your ISA for both benchmarks.

Benchmark 1:

- Clear all registers
- Allow the user to input a number of values to store, and store that amount
- Use a loop to read the needed values, and then compute the sum
  - Note, the loop could be done on the C++ simulator end, within an ISA instruction.
- Display the sum

Benchmark 2:

- Create an array of 5 integers
- Clear all registers
- Populate the array with values (either randomly or via user input)
- Ask the user for a number to find in the array
- Show a message as to whether or not the number was found
  - This can be a message that is hardcoded in the C++ simulator

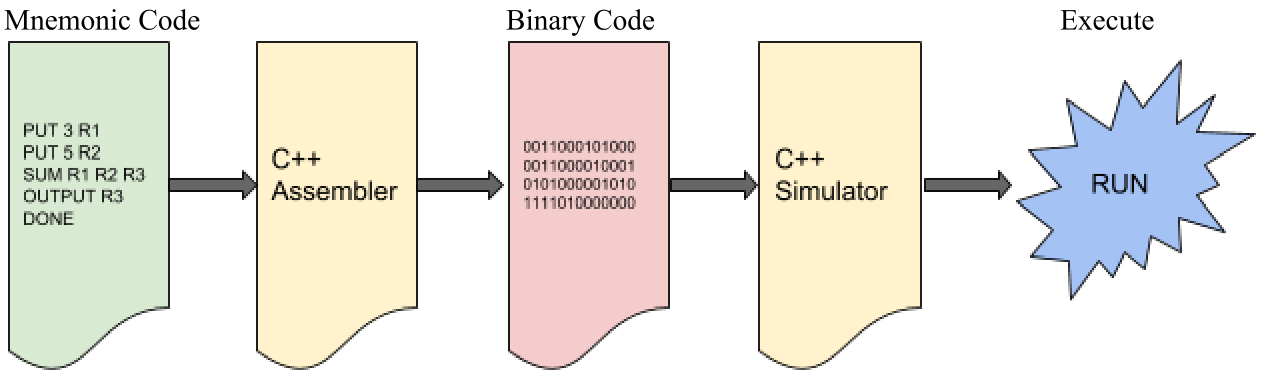
Recommended Steps

- Familiarize yourself with the project specifications.
- Decide which instructions (opcodes) you need to meet the specifications.
- Decide on the addressing modes to be used.
  - Immediate? Direct? Indirect? Multiple modes?
- Decide what type(s) of operands will be used (variables? registers? addresses? A mix?)
- Design your ISA
- How many bits for opcode
  - Depends on number of opcodes
- How many bits for operands
  - May be different for each instruction since opcode may use different args.

Example project start

```
My group’s architecture
    3 GPRs: R1, R2, R3  OR accumulator
    MAR - what it is used for
    etc.
************************************************
My group’s ISA

    MOD reg1, reg2

    Performs a modulus operation on the values
    in the 2 register operands. The value in
    reg1 is divided by the value in reg2, and
    the remained of the division is stored in
    reg1.

    Example:
       If R1 contains a 7 and R2 contains a 4,
            MOD R1, R2
       would result in a 3 being stored in R1
*************************************************
// data file to load into c++ assembler:

   PUT 3 R1
   PUT 5 R2
   SUM R1 R2 R3
   OUTPUT R3
   DONE
*************************************************
I don’t include a binary file because the length of those instructions
will depend on what you decide is needed for opcode size, register and
memory size, etc. But, it’d be zeroes and ones.

*************************************************
// simulator that runs the binary file generated by the assembler

void put(unsigned int, const int);
void show(unsigned int);
void mod(int& r1, int r2)
{
   r1 %= r2;
}

int main()
{
   //  declare needed vars
   //  call function to read data file
   //  execute instructions in data file,
   //  as if it were the assembler running the program
```

Diagram showing flow



Deliverables

- Documentation that shows specifications for your ISA. This should include how your instruction word is divided into different parts, what the opcodes are and how they work, as well as specifications about the architecture needed for your ISA.
- C++, Java or Python programs that compile your ISA and then run your compiled program.
- 2 files with the benchmark programs coded using your ISA.
- Source file that tests both benchmark programs.
- Ensure that all team members turn in all deliverables and that the names of all team members are on all work.
- A short presentation that details your ISA, code written using your ISA, something in particular you wanted to point out regarding your IDA, and a demo of both benchmark programs. These will take place during our scheduled finals time.

How to submit?

Turn in your program deliverables on **Canvas**.