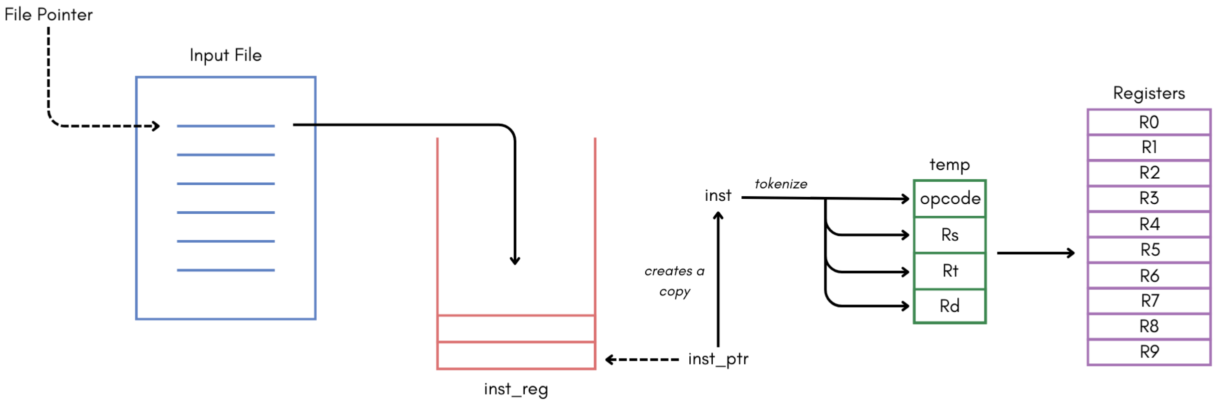
Simple Calculator Project (Computer Architecture)

Nathaniel Vallen
32221415
Global SW Convergence
Free Days: 5

Introduction:

The following project is a simple calculator designed to receive instructions from a separate text file in a format similar to basic MIPS commands. The program's capability revolves around performing essential arithmetic operations as well as some additional operations which will be specified in a later part of the document. It adopts the simple architecture of Von Neumann model, utilizing a memory, an instruction register, and an instruction pointer in executing its tasks. This project was done with the purpose of understanding and implementing the basics of creating our own instruction set architecture (ISA). Additionally, this was also done as the first of the four projects set for the computer architecture lecture.

The general steps taken to execute the program can be described in the following manner:



Initially, the file pointer reads the instructions written in the input file and copies each line to the stack "inst_reg" (instruction register). A program counter "inst_ptr" is set to the bottom of the stack, indicating the first instruction to execute. Next, a copy of the instruction pointed out by the "inst_ptr" is made to be tokenized and processed further before executing. These tokens will temporarily be stored in the array "temp". After that, these instructions will be executed, and depending on the instruction, it may or may not read/write the values in the specified register(s) and the program will print out the contents of each register. Once the execution is done, the program counter "inst_ptr" moves up by one in the stack to copy the next instruction to copy and tokenize. These steps will be repeated until the program encounters an error, a halt instruction (H), or reaches the end of the file.

Important Concepts & Considerations:

I. The submitted compressed file consists of two C files (reader.c and functions,.c), two header files (functions.h and variables.h), and an object (executable) file. In the case of the executable file not working, write the following line in order to compile the code:

gcc -o <object_file_name> reader.c functions.c

As a side note, the default object file name I created is “reader”.

II. Be sure to keep all the header files and C files in the same directory (one folder).

III. The proper usage format of the simple calculator is as follows:

<object_file_name> <input_file_name>

As a side note, the default (and attached) input file name I created is “input.txt”. Please make sure that the **input file is a text file** (.txt).

IV. The instructions written in the input file must satisfy the following format:

Instruction	**Format**	**Notes**
+	+ Rs Rt	Add *Rt* to *Rs* and store the result in R[0].
	+ Rs Imm or + Imm Rt	Add *Rs* to *Imm* or *Imm* to *Rt* and store the result in R[0].
	+ Imm_1 Imm_2	Add *Imm_1* to *Imm_2* and store the result in R[0].
-	- Rs Rt	Subtract *Rt* from *Rs* and locate the result in R[0].
	- Rs Imm or - Imm Rt	Subtract *Imm* from *Rs* or *Rt* from *Imm* and store the result in R[0].
	- Imm_1 Imm_2	Subtract *Imm_2* from *Imm_1* and store the result in R[0].
*	* Rs Rt	Multiply *Rt* by *Rs* and locate the result in R[0].
	* Rs Imm or * Imm Rt	Multiply *Rs* by *Imm* or *Imm* by *Rt* and store the result in R[0].
	* Imm_1 Imm_2	Multiply *Imm_1* by *Imm_2* and store the result in R[0].

/	/ Rs Rt	Divide *Rs* by *Rt* and locate the result in R[0].
	/ Rs Imm or / Imm Rt	Divide *Rs* by *Imm* or *Imm* by *Rt* and store the result in R[0].
	/ Imm_1 Imm_2	Divide *Imm_1* by *Imm_2* and store the result in R[0].
M	M Rs Rt	Copy the value of *Rs* to *Rt.*
	M Imm Rt	Copy *Imm* to *Rt.*
C	C Rs Rt	Compare the values of two registers *Rs* and *Rt*. If the values are equal, store 0 in R[0]. If *Rs* is less than *Rt*, store -1 in R[0]. If *Rs* is greater than *Rt*, store 1 in R[0].
	C Rs Imm or C Imm Rt	Compare the values of register *Rs* and an immediate value *Imm*. If the values are equal, store 0 in R[0]. If the first value is less than the second, store -1 in R[0]. If the first value is greater than the second, store 1 in R[0].
	C Imm_1 Imm_2	Compare the values of two immediate values *Imm_1* and *Imm_2*. If the values are equal, store 0 in R[0]. If the first value is less than the second, store -1 in R[0]. If the first value is greater than the second, store 1 in R[0].
J	J n	Jump to line number *n* where *n* is a round number from 1 to the number of lines in the input file.
BEQ	BEQ n	Compare the value of R[0] with 0. If R[0] = 0, jump to line number *n* where *n* is a round number from 1 to the number of lines in the input file.
BNE	BNE n	Compare the value of R[0] with 0. If R[0] ≠ 0, jump to line number *n* where *n* is a round number from 1 to the number of lines in the input file.
GCD	GCD Rs Rt	Calculate the GCD of two register values *Rs* and *Rt*, and store the result in R[0].
	GCD Rs Imm or GCD Imm Rt	Calculate the GCD of *Rs* and *Imm* or *Imm* and *Rt*, and store the result in R[0].

	GCD Imm_1 Imm_2	Calculate the GCD of two immediate values *Imm_1* and *Imm_2*, and store the result in R[0].
H	H <space>	Halt and terminate the program.
		Please make sure to **include a <space> character** after "H" (because the tokenize function uses the space character as a delimiter).

V. The system has 10 registers with the specifications as follows:

Register	**Content**
R[0]	Calculation result
R[1]	Free registers
R[2]	
R[3]	
R[4]	
R[5]	
R[6]	
R[7]	
R[8]	
R[9]	

VI. The number of instructions (lines) in the input file should be at most 100 lines, with each line consisting of only one instruction.

VII. If the instruction contains more operands than what was set in the format, the excessive operands will be ignored unless the instruction length exceeds the limit.

VIII. The program takes in strictly **one** input file.

IX. You can freely modify the instructions in the input file. The current version of instructions are designed to run all listed instruction types while also covering different formats used in varying situations to show that the program works perfectly fine.

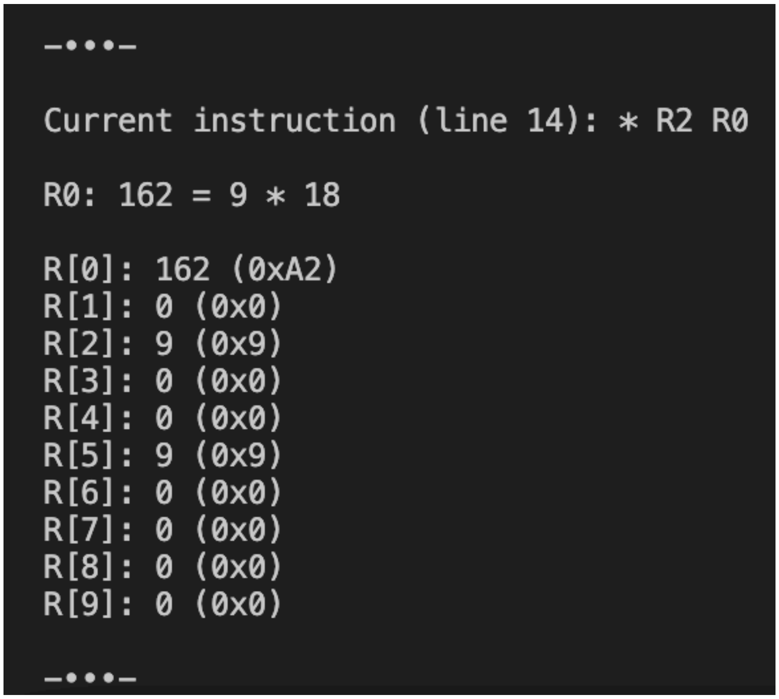
Unique Features:

- BNE instruction is added as a complement to BEQ instruction.
- Dividers between instruction printing support more easy-to-read output.

Build Configuration & Environment:

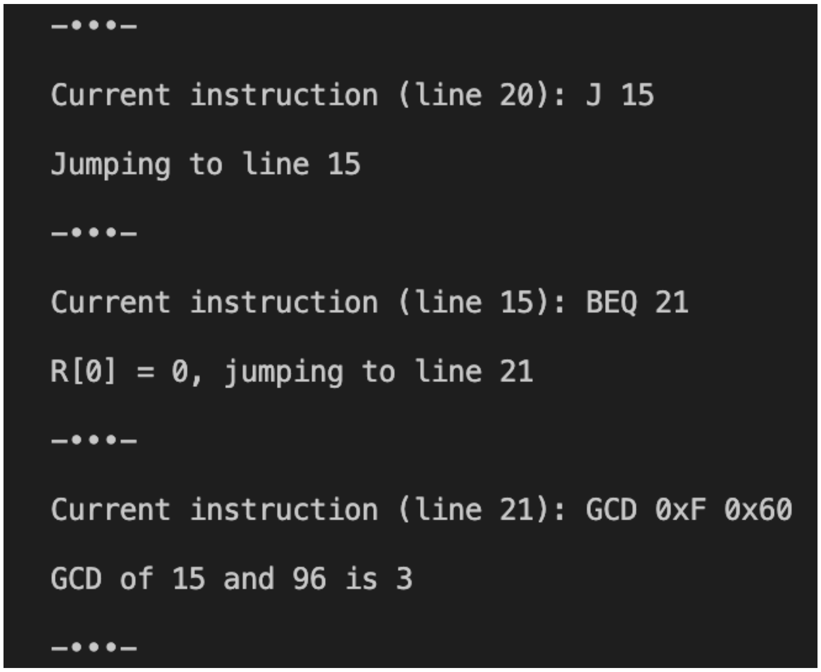
The code in this project was written in C programming language using Visual Studio Code IDE. Meanwhile, the input file “input.txt” was created through the Mac terminal using vi editor.

Screen Captures:



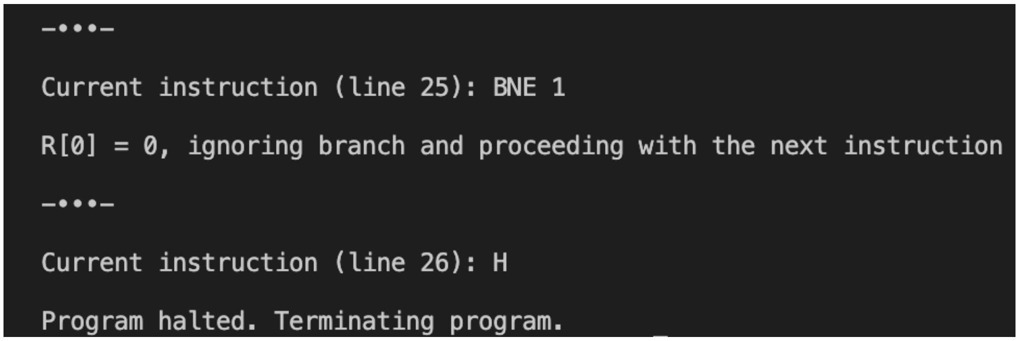
```
-•••-

Current instruction (line 14): * R2 R0

R0: 162 = 9 * 18

R[0]: 162 (0xA2)
R[1]: 0 (0x0)
R[2]: 9 (0x9)
R[3]: 0 (0x0)
R[4]: 0 (0x0)
R[5]: 9 (0x9)
R[6]: 0 (0x0)
R[7]: 0 (0x0)
R[8]: 0 (0x0)
R[9]: 0 (0x0)

-•••-
```

Figure 1. Snippet of the result (operation)*



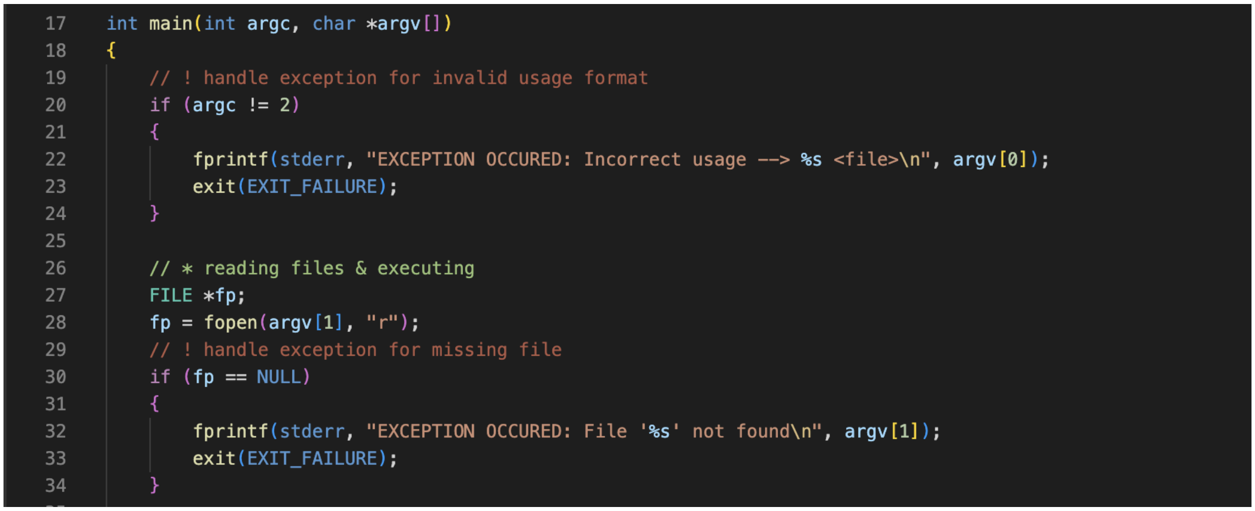
```
—•••—

Current instruction (line 20): J 15

Jumping to line 15

—•••—

Current instruction (line 15): BEQ 21

R[0] = 0, jumping to line 21

—•••—

Current instruction (line 21): GCD 0xF 0x60

GCD of 15 and 96 is 3

—•••—
```

Figure 2. Snippet of the result (J, BEQ, and GCD operation)



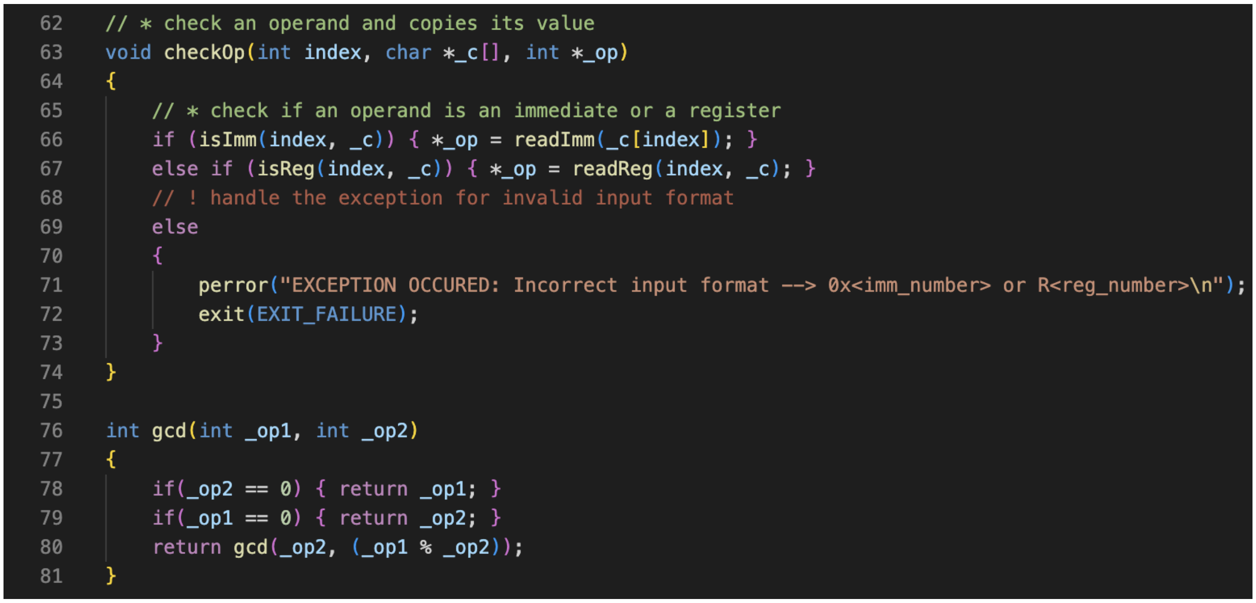
```
—•••—

Current instruction (line 25): BNE 1

R[0] = 0, ignoring branch and proceeding with the next instruction

—•••—

Current instruction (line 26): H

Program halted. Terminating program.
```

Figure 3. Snippet of the result (BNE and H operation)



```c
int main(int argc, char *argv[])
{
    // ! handle exception for invalid usage format
    if (argc != 2)
    {
        fprintf(stderr, "EXCEPTION OCCURED: Incorrect usage --> %s <file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // * reading files & executing
    FILE *fp;
    fp = fopen(argv[1], "r");
    // ! handle exception for missing file
    if (fp == NULL)
    {
        fprintf(stderr, "EXCEPTION OCCURED: File '%s' not found\n", argv[1]);
        exit(EXIT_FAILURE);
    }
```

Figure 4. Code snippet of initialization



```c
// * check an operand and copies its value
void checkOp(int index, char *_c[], int *_op)
{
    // * check if an operand is an immediate or a register
    if (isImm(index, _c)) { *_op = readImm(_c[index]); }
    else if (isReg(index, _c)) { *_op = readReg(index, _c); }
    // ! handle the exception for invalid input format
    else
    {
        perror("EXCEPTION OCCURED: Incorrect input format --> 0x<imm_number> or R<reg_number>\n");
        exit(EXIT_FAILURE);
    }
}

int gcd(int _op1, int _op2)
{
    if(_op2 == 0) { return _op1; }
    if(_op1 == 0) { return _op2; }
    return gcd(_op2, (_op1 % _op2));
}
```

Figure 5. Code snippet of some functions used