UNIVERSITY OF SOUTHAMPTON

ELEC2204 Computer Engineering Coursework: Computer Simulation

by

Yan Jie, Kee

Student ID: 29883407

Email address: yjk1g18@soton.ac.uk

Technical Report

Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

28 April 2020

Contents

List of Figures

Listings

Chapter 1

Design

Figure 1.1 shows the data-path of my design, all of my work is based on the MIPS architecture from the lecture slides (1) as well as the recommended textbook (2). Some notable changes are...

1. Shift-2 operators are removed from jump instructions.
2. Instruction J(Jump) no longer takes the first 4 digits of program counter.
3. A single multiplexer is used to decide the next program count. Inspired from (3).
4. Another layer is added between general register and $ra(Return Address) to allow the instruction JAL(Jump and Link) to function.
5. BYTE_ACCESS control signal is added to allow memory access in byte length instead of word.
6. Lastly, to comply with the Von Neumann architecture of a single memory object, a control signal PC_READ is used to differentiate the bus into 'Addr' input of the memory module from being from the program counter or ALU.

Most of the changes made are to simplify the design.

1.1 General Register

Table A.1 is the design for my general register. It is the same as from MIPS, expect the register $mp, which is a pointer used to ease the process of logging into memory (Highlighted on the table). Otherwise, all other register is used as intended for MIPS. Although all 32 register is created, not all are being used, for example: $at, $k and $gp.

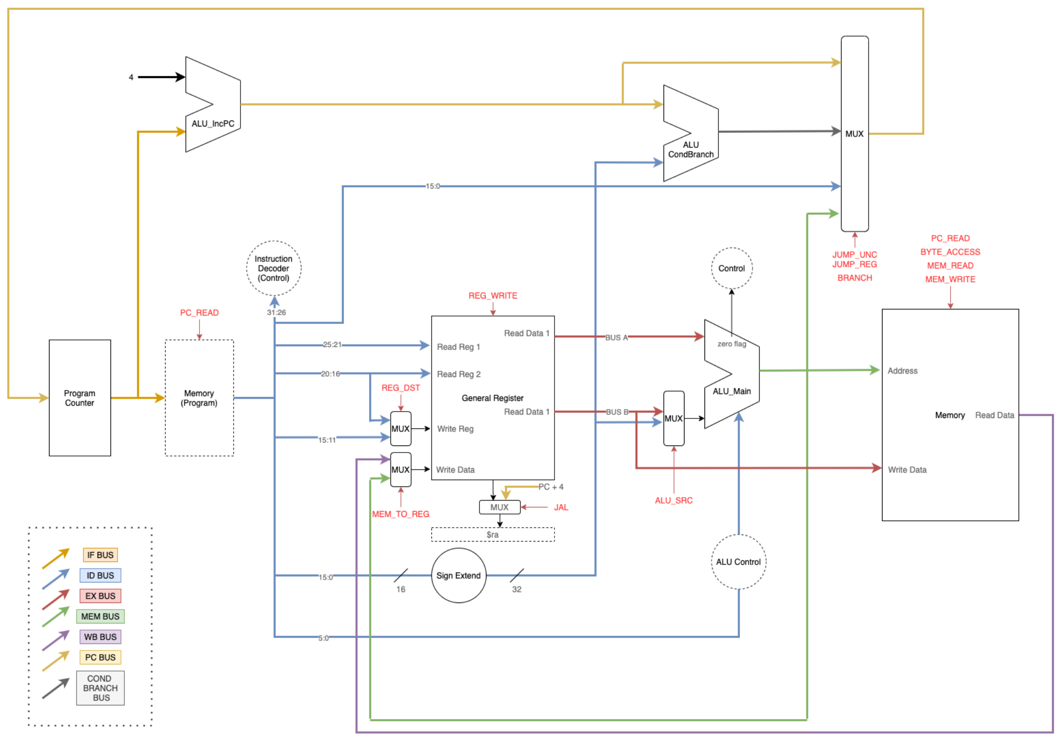


FIGURE 1.1: The architecture of the computer design.

1.2 Memory Structure

The memory structure of the design is the same as from the textbook (2) too. Though is it much smaller in size to compensate the simplification done on the jump instruction operation (removal of shift-2 operator and more). Other than that, a dedicated section of memory is used for logging the results of the showcase testing. The diagram of the memory structure can be seen on Appendix A, Figure A.1.

1.3 Instruction Set

The instructions are all from the MIPS architecture too. They were referred from (4) and (5). A slight changes is made on instruction JR, which is supposed to be an R-type, but the opcode was changed (Highlighted on table). The full table of instruction set is on Appendix A, Table A.2.

1.4 Remarks

The reason why all the designs are based on the MIPS architecture is because that the design of this computer is done top-down from a C function, to MIPS Assembly code and then decision of the design. Hence why the resemblance.

Chapter 2

Functionality

This computer consists of a total of 7 modules, namely:

1. Program Counter
2. Memory
3. General Register
4. ALU (Main)
5. ALU (PC Increment)
6. ALU (Conditional Branch)
7. Instruction Decoder/ Control

Not much explanation is needed for each modules, since their functionality is similarly to that of MIPS's modules. They are genuinely based from the lecture slides, (1).

2.1 Program Code

The similarity of the C++ object code is that, each object will have a pointer to two or more buses defined when its constructor is called. On the Figure 1.1, a different color is assigned to different buses, that is how each objects be connected in the code. Besides, every object will have a `run()` function that will read the input bus(es) and write onto the output bus, which functionality depends of the control signals. In reality, each object will be "running" all the time, which is not simulated in my program. The advantage of this program is that, each stages (IF, ID, EX, MEM, WB) can be clearly separated.

Therefore, in the `main()` function, buses (wires) are defined as `uint32_t` which makes it 32-bits-wide. Then, each objects is constructed with their assigned buses. In the case of a multiplexer that changes the bus an object connected to entirely, an additional function called `change_bus_in()` is given so some objects for this specific reason. All the C++ objects code are submitted in another folder containing all source code. Otherwise, an alternative is to pull from the GitHub. All accounts of the project are stored there.

Once all the objects and buses are initialised, the `main()` function goes into a `while(1)` loop. The cycle of IF, ID, EX, MEM and WB is repeated continuously until a special instruction: TERMINATE (opcode = 0b111111) is called. This will break the infinite loop by returning from `main()`.

2.2 Remarks

There is a fudge that is worth mentioning in this report. To comply with the Von Neumann design philosophy of a single memory module, an additional control signal is added and named PC_READ. This signal is not controlled by the op-code, instead it is manually set depending on the stage in a cycle (IF stage). Due to time constraint, no further study is done to find a better solution to this problem. The only solution that I come up with, is to make an instruction called 'fetch' that will be called in between every instructions. It is dedicated to do just an instruction fetch. This in turn, does not comply with the coursework guidance to make a single instruction cycle of IF, ID, EX, MEM and WB. Hence, my approach. Though, it can be seen as a control signal set by a clock that sync with the one connected to the PC. Note that both "clocks" are non-existence in my program code.

Chapter 3

Basic Testing

Basic testing plan and results.

3.1 C Function

Two simple C functions are created to complete the two tasks for showcase. The results are compared with a built-in square function for C, and online prime number list. The created functions are listed in Appendix B, B.1 and B.2. The algorithm of the primality test is called the Sieve of Eratosthenes (6).

3.2 Assembly Code, MARS

The C code is translated into assembly code manually, and tested on MARS. The results (from syscall) is again tested with the reference from before. Due to difficulty of the `square()` function (B.1), the algorithm is changed to simple iterative addition. The .asm file are attached in Appendix C for reference (C.1 and C.2)

3.3 C++ Objects

Once the assembly code is working, an instruction set is defined and the required objects for the computer is created. Each object is tested with a simple script once created to ease the process of integration. The testing functions are defined in the main source file. To test the functionality of each object, a debug function is added to display the state of object onto the terminal. The debug mode can be turned on and off by commenting out `ENABLE_DEBUG` macro in the `include/debug.h` file. The results of the testing is available on Appendix D.

3.4 C++ Integration

Integration is done by connecting every modules with their respective buses, and adding required multiplexers in between the objects. This stage is tested by running every instructions and reading the output of debug messages. The script of the test can be found in the folder `instruction_test_text/` submitted. Results of each instruction is kept in a file called `test/` but is not attached in this report due to the size.

Note: Alternative design is thought prior to the last version. A lot of reference is done on a lecture slide found online (3), which implemented the design with a signal ALU, general register, memory and program counter. The idea was abandoned once the complication is realised, and due to limited amount of time.

3.5 Machine Code

Lastly, the machine code for both functions(square and primality test) are generated manually and tested. The results are printed when the computer accesses to the ”log” memory section, which prove its functionality.

Chapter 4

Showcase Testing

The final version of the program code is design to output the logging into a text file in `log/main_compressed` or Appendix E. In it is the debug message that is triggered when a memory write is done within the log section (0x00000800 - 0x00000cff) of memory module. There is another log file generated by the final code, a detailed version. But due to the number of iterations done to complete the two tasks, that version is deemed unreadable, with a file size of 69.5 MB. It contains the state of every object in every cycle, the file is kept in the same folder (`log/main`).

4.1 Reproducing Results

In order to reproduce the results, a few command in the terminal is required:

1. Either download the submitted folder,
   or cloning from git using:
   `git clone https://github.com/yanjiekee/computer-simulation.git`.
2. `cd computer-simulation`
3. The log file is defined in `src/debug.cpp/` line 26, it can be changed to your liking.
4. `make`
5. `make run`

Chapter 5

Conclusions

The focus in this work is not on innovation but a deeper understanding in MIPS computer architecture, hence the similarity. It is almost a direct replication of the MIPS computer we learnt in the lecture, with a few tweaks.

The main improvement on my work on computer simulation is definitely the translation of assembly language to machine code, it is a draining job to write in binary for almost 90+ lines for the final version of my code. It will be very useful to create another simple decoder that can do the repetitive work for user. I've developed strong appreciation to GCC more than ever after this coursework. Lastly, contradicting to the first paragraph, I think once I have better understanding on the computer architecture, it would be more helpful to venture more creative approach, which is to design a new model than can do a specific job much more efficiently that a general purpose computer.

Appendix A

Computer Design

Name	Register Number	Description
$zero	0	Value zero, cannot be re-written
$at	1	Reserved for pseudo-instruction
$v0 - $v1	2 - 3	Results
$a0 - $a3	4 - 7	Arguments
$t0 - $t9	8 - 15	Temporary storage
$s0 - $s7	16 - 23	Saved content storage
$t8 - $t9	24 - 25	Temporary storage
$k0	26	Reserved for OS
$mp	27	”Memory” pointer: for logging
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

TABLE A.1: General Register

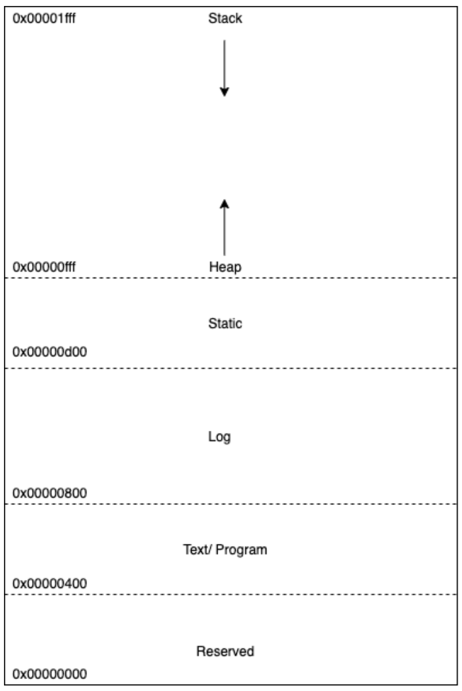


FIGURE A.1: Memory structure.

Instr	**Description**	**Machine Code**	**Type**
LW	Loading data from memory onto register of word length (\$t = \$s + offset)	op-s-t-offset	I
LB	Loading data from memory onto register of byte length (\$t = \$s + offset)	op-s-t-offset	I
SW	Storing data from register onto memory of word length (\$s + offset = \$t)	op-s-t-offset	I
SB	Storing data from register onto memory of byte length (\$s + offset = \$t)	op-s-t-offset	I
ADD	Adding two registers and store result onto another register (\$d = \$s + \$t)	op-s-t-d-shamt-func	R
ADDI	Adding a register with a constant and store result onto a register (\$t = \$s + const)	op-s-t-const	I
SUB	Subtracting one register from another and store result in another register (\$d = \$s - \$t)	op-s-t-d-shamt-func	R
SLT	Set a value of register to 1/0 depending on value of two registers. (Set \$d = 1 if \$s less than \$t)	op-s-t-d-shamt-func	R
BEQ	Branch if two registers value are equal (Next PC = PC + 4 + Offset if \$s = \$t)	op-s-t-offset	I
J	Unconditional jump (Next PC = Addr)	op-addr	J
JAL	Unconditional jump and store the value of next PC onto \$ra (return address) (Next PC = Addr, and \$ra = PC + 4)	op-addr	J
JR	Unconditional jump to the value stored in \$ra (return address) (Next PC = \$ra)	op-ra-0-0-shamt-func	I/R

Table A.2: Instruction Set

Appendix B

Task's Function Planning

B.1 Squaring an Integer

```
#include <stdint.h>
// uint16_t arr[NUM_ELEMENTS] = {...};

uint16_t square(uint16_t n);
uint16_t square_alt(uint16_t);

uint16_t square(uint16_t n) {
    if (n == 0)
        return 0;

    // x = floor(n/2)
    uint16_t x = (n >> 1);
    if (n & 1)
        return ((square(x) << 2) + (x << 2) + 1);
    else
        return ((square(x) << 2));
}

uint16_t square_alt(uint16_t n) {
    uint16_t k = 0;
    for (uint16_t i = 0; i < n; i++) {
        k += n;
    }
    return k;
}
```

Listing B.1: square.h

B.2 Primality Test using Sieve of Eratosthenes Algorithm

```
#include <stdint.h>
// Include the task square() too

void sieveOfEratosthenes(uint16_t upperLimit);
void sieveOfEratosthenes(uint16_t upperLimit) {
    // To have upperLimit as the last index, default boolean is true
    bool numberList[upperLimit + 1];

    for (uint16_t i = 0; i <= upperLimit; i++) {
        numberList[i] = true;
    }

    // Instead of i <= root of n, we can use condition squared i <= n
    for (uint16_t i = 2; (i * i) <= upperLimit; i++) {
        if (numberList[i]) {
            for (uint16_t j = (i * i); j <= upperLimit; j += i)
                numberList[j] = false;
        }
    }

    // Print out the remaining (prime) numbers after eliminating composite numbers
    // for (int i = 2; i <= upperLimit; i++) {
    //     if (numberList[i])
    //         printf("%d\n" ,i);
    // }
    return;
}
```

Listing B.2: sieveOfEratosthenes.h

Appendix C

MARS Code (.asm)

C.1 Square

```
.data

.text
main:   addi $s0 $zero 1
                addi $s1 $zero 99
L6:             slt $t0 $s0 $s1
                beq $t0 $zero exitL6
                addi $sp $sp -8
                sw $s0 0($sp)
                sw $s1 4($sp)
                add $a0 $s0 $zero
                jal square
                lw $s1 4($sp)
                lw $s0 0($sp)
                addi $sp $sp 8
                addi $s0 $s0 1
                j L6
exitL6: addi $v0 $zero 10
        syscall
square: addi $v0 $zero 0
                addi $t1 $zero 0
L5:             slt $t3 $t1 $a0
                beq $t3 $zero exitL5
                add $v0 $v0 $a0
                addi $t1 $t1 1
                j L5
exitL5:
addi $sp $sp -8
        sw $a0 0($sp)
        add $a0 $v0 $zero
        sw $v0 4($sp)
        li $v0 1

        syscall
        li $v0 11
        li $a0 10       # newline
        syscall
        lw $v0 4($sp)
        lw $a0 0($sp)
        addi $sp $sp 8
                jr $ra
```

Listing C.1: square_ori.asm

C.2 Primality Test

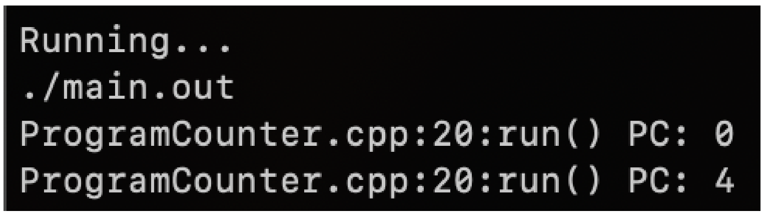
```
.data
range:  .word   100

.text
main:
        lw $t0 range
        add $a0 $zero $t0
        jal primalityTest

        addi $s0 $zero 1
        addi $s1 $zero 99
L6:             slt $t0 $s0 $s1
                beq $t0 $zero exitL6
                addi $sp $sp -8
                sw $s0 0($sp)
                sw $s1 4($sp)
                add $a0 $s0 $zero
                jal square
                addi $sp $sp -8
        sw $a0 0($sp)
        add $a0 $v0 $zero
        sw $v0 4($sp)
        li $v0 1

        syscall
        li $v0 11
        li $a0 10       # newline
        syscall
        lw $v0 4($sp)
        lw $a0 0($sp)
        addi $sp $sp 8
                lw $s1 4($sp)
                lw $s0 0($sp)
                addi $sp $sp 8
                addi $s0 $s0 1
                j L6

exitL6: addi $v0 $zero 10
        syscall

primalityTest:
        add $fp $sp $zero
        sub $sp $sp $t0
        addi $t0 $zero 0
        addi $t1 $a0 1

L1:
        slt $t2 $t0 $t1
        beq $t2 $zero exitL1
        sub $t2 $fp $t0
        addi $t3 $zero 1
        sb $t3 0($t2)
        addi $t0 $t0 1
        j L1

exitL1:
        addi $s0 $zero 2

L2:
        addi $sp $sp -12
        sw $a0 8($sp)
        sw $s0 4($sp)
        sw $ra 0($sp)
        add $a0 $s0 $zero
        jal square
        lw $ra 0($sp)
        lw $s0 4($sp)
        lw $a0 8($sp)
        addi $sp $sp 12
        addi $t0 $a0 1
        slt $t1 $v0 $t0
        beq $t1 $zero exitL2

C1:
        sub $t0 $fp $v0
        lb $t1 0($t0)
        beq $t1 $zero exitC1
        # else, initialise for-loop (first iterate and also tweek variable for condition)
        addi $sp $sp -12
        sw $a0 8($sp)
        sw $s0 4($sp)
        sw $ra 0($sp)
        add $a0 $s0 $zero
        jal square
        lw $ra 0($sp)
        lw $s0 4($sp)
        lw $a0 8($sp)
        addi $sp $sp 12
        add $t0 $v0 $zero       # j = i^2
        addi $t1 $a0 1

L3:
        slt $t3 $t0 $t1
        beq $t3 $zero exitL3
        sub $t4 $fp $t0
        addi $t2 $zero 0
        sb $t2 0($t4)
        add $t0 $t0 $s0
        j L3

exitL3:

exitC1:
        addi $s0 $s0 1
        j L2
```

```
exitL2:
        addi $t0 $zero 2
        addi $t1 $a0 1

L4:
        slt $t2 $t0 $t1
        beq $t2 $zero exitL4

C2:
        sub $t2 $fp $t0
        lb $t3 0($t2)
        beq $t3 $zero exitC2
        addi $sp $sp -4 # From here to bottom is printing result onto the screen, change to store into
        sw $a0 0($sp)
        li $v0 1
        add $a0 $t0 $zero
        syscall
        li $v0 11
        li $a0 10       # newline
        syscall
        lw $a0 0($sp)
        addi $sp $sp 4

exitC2:
        addi $t0 $t0 1
        j L4

exitL4:
        lw $t0 range
        add $sp $sp $t0
        jr $ra

square:
        addi $v0 $zero 0
        addi $t1 $zero 0

L5:
        slt $t3 $t1 $a0
        beq $t3 $zero exitL5
        add $v0 $v0 $a0
        addi $t1 $t1 1
        j L5

exitL5:
        jr $ra
```

Listing C.2: primalityTest_ori.asm

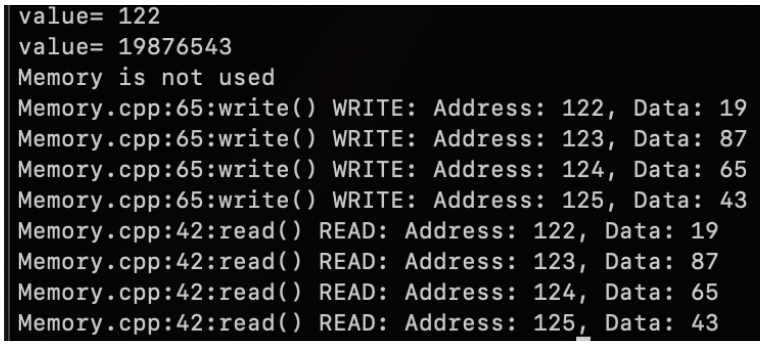
Appendix D

Basic Testing Results Evidence



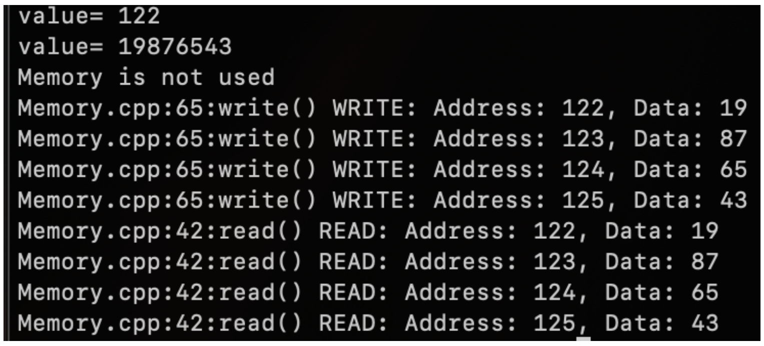
```
Running...
./main.out
ProgramCounter.cpp:20:run() PC: 0
ProgramCounter.cpp:20:run() PC: 4
```

FIGURE D.1: Test results for PC and PC Increment Adder.



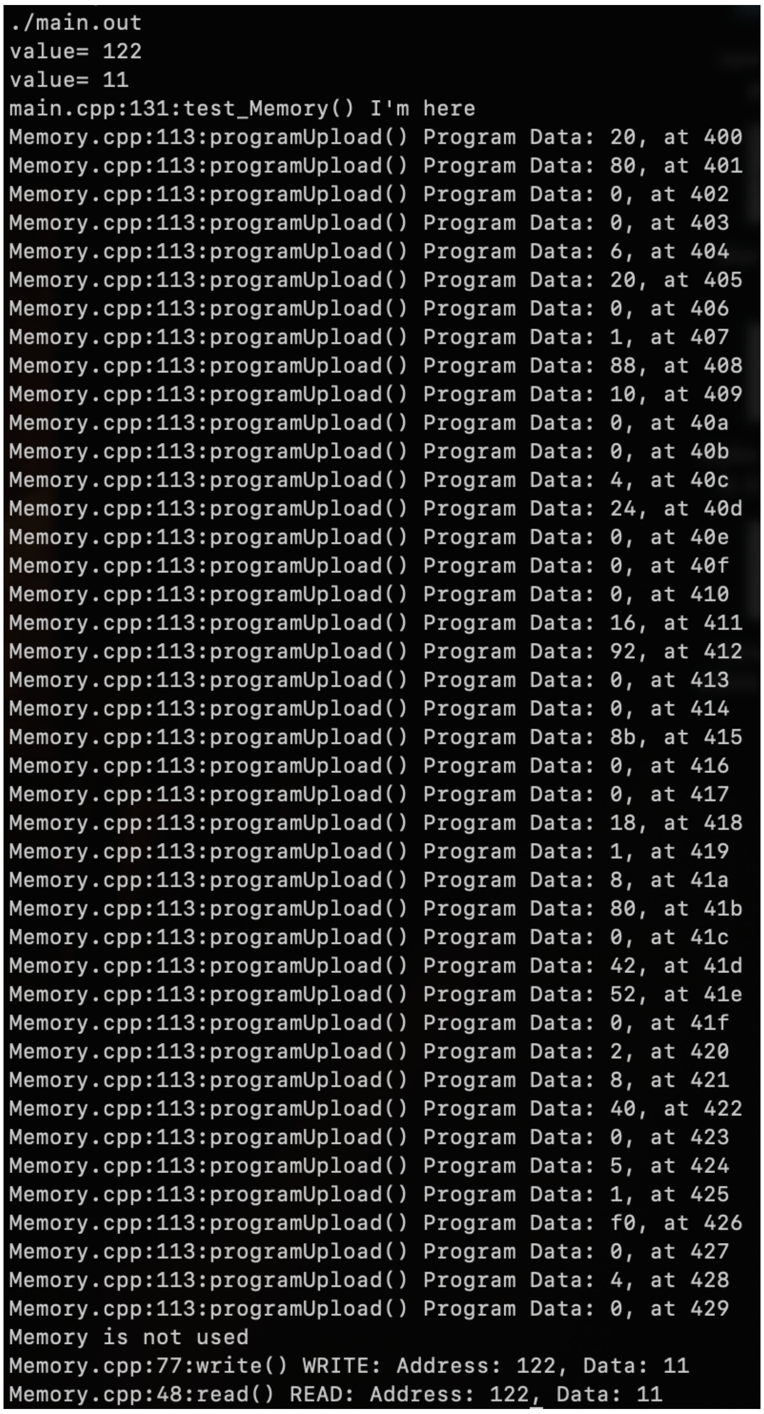
```
value= 122
value= 19876543
Memory is not used
Memory.cpp:65:write() WRITE: Address: 122, Data: 19
Memory.cpp:65:write() WRITE: Address: 123, Data: 87
Memory.cpp:65:write() WRITE: Address: 124, Data: 65
Memory.cpp:65:write() WRITE: Address: 125, Data: 43
Memory.cpp:42:read() READ: Address: 122, Data: 19
Memory.cpp:42:read() READ: Address: 123, Data: 87
Memory.cpp:42:read() READ: Address: 124, Data: 65
Memory.cpp:42:read() READ: Address: 125, Data: 43
```

FIGURE D.2: Test results for Memory module's private functions.



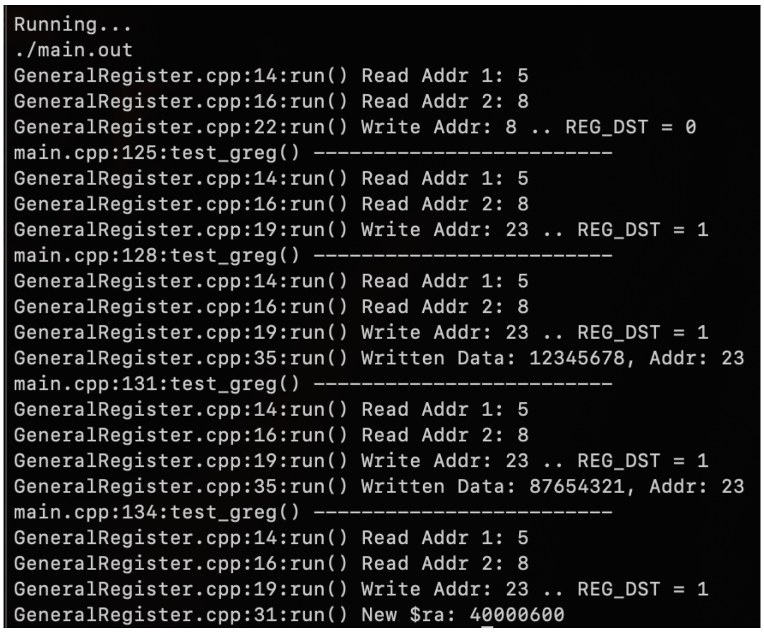
```
value= 122
value= 19876543
Memory is not used
Memory.cpp:65:write() WRITE: Address: 122, Data: 19
Memory.cpp:65:write() WRITE: Address: 123, Data: 87
Memory.cpp:65:write() WRITE: Address: 124, Data: 65
Memory.cpp:65:write() WRITE: Address: 125, Data: 43
Memory.cpp:42:read() READ: Address: 122, Data: 19
Memory.cpp:42:read() READ: Address: 123, Data: 87
Memory.cpp:42:read() READ: Address: 124, Data: 65
Memory.cpp:42:read() READ: Address: 125, Data: 43
```

FIGURE D.3: Test results for Memory module.



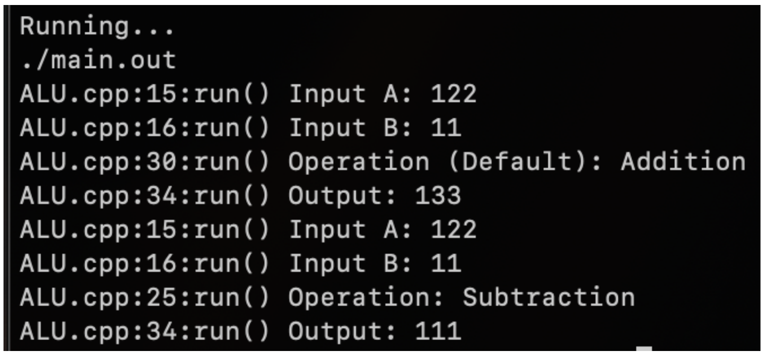
```
./main.out
value= 122
value= 11
main.cpp:131:test_Memory() I'm here
Memory.cpp:113:programUpload() Program Data: 20, at 400
Memory.cpp:113:programUpload() Program Data: 80, at 401
Memory.cpp:113:programUpload() Program Data: 0, at 402
Memory.cpp:113:programUpload() Program Data: 0, at 403
Memory.cpp:113:programUpload() Program Data: 6, at 404
Memory.cpp:113:programUpload() Program Data: 20, at 405
Memory.cpp:113:programUpload() Program Data: 0, at 406
Memory.cpp:113:programUpload() Program Data: 1, at 407
Memory.cpp:113:programUpload() Program Data: 88, at 408
Memory.cpp:113:programUpload() Program Data: 10, at 409
Memory.cpp:113:programUpload() Program Data: 0, at 40a
Memory.cpp:113:programUpload() Program Data: 0, at 40b
Memory.cpp:113:programUpload() Program Data: 4, at 40c
Memory.cpp:113:programUpload() Program Data: 24, at 40d
Memory.cpp:113:programUpload() Program Data: 0, at 40e
Memory.cpp:113:programUpload() Program Data: 0, at 40f
Memory.cpp:113:programUpload() Program Data: 0, at 410
Memory.cpp:113:programUpload() Program Data: 16, at 411
Memory.cpp:113:programUpload() Program Data: 92, at 412
Memory.cpp:113:programUpload() Program Data: 0, at 413
Memory.cpp:113:programUpload() Program Data: 0, at 414
Memory.cpp:113:programUpload() Program Data: 8b, at 415
Memory.cpp:113:programUpload() Program Data: 0, at 416
Memory.cpp:113:programUpload() Program Data: 0, at 417
Memory.cpp:113:programUpload() Program Data: 18, at 418
Memory.cpp:113:programUpload() Program Data: 1, at 419
Memory.cpp:113:programUpload() Program Data: 8, at 41a
Memory.cpp:113:programUpload() Program Data: 80, at 41b
Memory.cpp:113:programUpload() Program Data: 0, at 41c
Memory.cpp:113:programUpload() Program Data: 42, at 41d
Memory.cpp:113:programUpload() Program Data: 52, at 41e
Memory.cpp:113:programUpload() Program Data: 0, at 41f
Memory.cpp:113:programUpload() Program Data: 2, at 420
Memory.cpp:113:programUpload() Program Data: 8, at 421
Memory.cpp:113:programUpload() Program Data: 40, at 422
Memory.cpp:113:programUpload() Program Data: 0, at 423
Memory.cpp:113:programUpload() Program Data: 5, at 424
Memory.cpp:113:programUpload() Program Data: 1, at 425
Memory.cpp:113:programUpload() Program Data: f0, at 426
Memory.cpp:113:programUpload() Program Data: 0, at 427
Memory.cpp:113:programUpload() Program Data: 4, at 428
Memory.cpp:113:programUpload() Program Data: 0, at 429
Memory is not used
Memory.cpp:77:write() WRITE: Address: 122, Data: 11
Memory.cpp:48:read() READ: Address: 122, Data: 11
```

Figure D.4: Test results for Memory module's upload program function.



```
Running...
./main.out
GeneralRegister.cpp:14:run() Read Addr 1: 5
GeneralRegister.cpp:16:run() Read Addr 2: 8
GeneralRegister.cpp:22:run() Write Addr: 8 .. REG_DST = 0
main.cpp:125:test_greg() -------------------------
GeneralRegister.cpp:14:run() Read Addr 1: 5
GeneralRegister.cpp:16:run() Read Addr 2: 8
GeneralRegister.cpp:19:run() Write Addr: 23 .. REG_DST = 1
main.cpp:128:test_greg() -------------------------
GeneralRegister.cpp:14:run() Read Addr 1: 5
GeneralRegister.cpp:16:run() Read Addr 2: 8
GeneralRegister.cpp:19:run() Write Addr: 23 .. REG_DST = 1
GeneralRegister.cpp:35:run() Written Data: 12345678, Addr: 23
main.cpp:131:test_greg() -------------------------
GeneralRegister.cpp:14:run() Read Addr 1: 5
GeneralRegister.cpp:16:run() Read Addr 2: 8
GeneralRegister.cpp:19:run() Write Addr: 23 .. REG_DST = 1
GeneralRegister.cpp:35:run() Written Data: 87654321, Addr: 23
main.cpp:134:test_greg() -------------------------
GeneralRegister.cpp:14:run() Read Addr 1: 5
GeneralRegister.cpp:16:run() Read Addr 2: 8
GeneralRegister.cpp:19:run() Write Addr: 23 .. REG_DST = 1
GeneralRegister.cpp:31:run() New $ra: 40000600
```

FIGURE D.5: Test results General Register.



```
Running...
./main.out
ALU.cpp:15:run() Input A: 122
ALU.cpp:16:run() Input B: 11
ALU.cpp:30:run() Operation (Default): Addition
ALU.cpp:34:run() Output: 133
ALU.cpp:15:run() Input A: 122
ALU.cpp:16:run() Input B: 11
ALU.cpp:25:run() Operation: Subtraction
ALU.cpp:34:run() Output: 111
```

FIGURE D.6: Test results for main ALU.

Appendix E

Showcase Testing Results Evidence

```
Memory.cpp:87:write() LOG: Data: d'2, addr: x'800
Memory.cpp:87:write() LOG: Data: d'3, addr: x'804
Memory.cpp:87:write() LOG: Data: d'5, addr: x'808
Memory.cpp:87:write() LOG: Data: d'7, addr: x'80c
Memory.cpp:87:write() LOG: Data: d'11, addr: x'810
Memory.cpp:87:write() LOG: Data: d'13, addr: x'814
Memory.cpp:87:write() LOG: Data: d'17, addr: x'818
Memory.cpp:87:write() LOG: Data: d'19, addr: x'81c
Memory.cpp:87:write() LOG: Data: d'23, addr: x'820
Memory.cpp:87:write() LOG: Data: d'29, addr: x'824
Memory.cpp:87:write() LOG: Data: d'31, addr: x'828
Memory.cpp:87:write() LOG: Data: d'37, addr: x'82c
Memory.cpp:87:write() LOG: Data: d'41, addr: x'830
Memory.cpp:87:write() LOG: Data: d'43, addr: x'834
Memory.cpp:87:write() LOG: Data: d'47, addr: x'838
Memory.cpp:87:write() LOG: Data: d'53, addr: x'83c
Memory.cpp:87:write() LOG: Data: d'59, addr: x'840
Memory.cpp:87:write() LOG: Data: d'61, addr: x'844
Memory.cpp:87:write() LOG: Data: d'67, addr: x'848
Memory.cpp:87:write() LOG: Data: d'71, addr: x'84c
Memory.cpp:87:write() LOG: Data: d'73, addr: x'850
Memory.cpp:87:write() LOG: Data: d'79, addr: x'854
Memory.cpp:87:write() LOG: Data: d'83, addr: x'858
Memory.cpp:87:write() LOG: Data: d'89, addr: x'85c
Memory.cpp:87:write() LOG: Data: d'97, addr: x'860
Memory.cpp:87:write() LOG: Data: d'101, addr: x'864
Memory.cpp:87:write() LOG: Data: d'103, addr: x'868
Memory.cpp:87:write() LOG: Data: d'107, addr: x'86c
Memory.cpp:87:write() LOG: Data: d'109, addr: x'870
Memory.cpp:87:write() LOG: Data: d'113, addr: x'874
Memory.cpp:87:write() LOG: Data: d'127, addr: x'878
Memory.cpp:87:write() LOG: Data: d'131, addr: x'87c
Memory.cpp:87:write() LOG: Data: d'137, addr: x'880
Memory.cpp:87:write() LOG: Data: d'139, addr: x'884
Memory.cpp:87:write() LOG: Data: d'149, addr: x'888
Memory.cpp:87:write() LOG: Data: d'151, addr: x'88c
Memory.cpp:87:write() LOG: Data: d'157, addr: x'890
Memory.cpp:87:write() LOG: Data: d'163, addr: x'894
Memory.cpp:87:write() LOG: Data: d'167, addr: x'898
Memory.cpp:87:write() LOG: Data: d'173, addr: x'89c
Memory.cpp:87:write() LOG: Data: d'179, addr: x'8a0
Memory.cpp:87:write() LOG: Data: d'181, addr: x'8a4
Memory.cpp:87:write() LOG: Data: d'191, addr: x'8a8
Memory.cpp:87:write() LOG: Data: d'193, addr: x'8ac
Memory.cpp:87:write() LOG: Data: d'197, addr: x'8b0
Memory.cpp:87:write() LOG: Data: d'199, addr: x'8b4
Memory.cpp:87:write() LOG: Data: d'211, addr: x'8b8
Memory.cpp:87:write() LOG: Data: d'223, addr: x'8bc
Memory.cpp:87:write() LOG: Data: d'227, addr: x'8c0
Memory.cpp:87:write() LOG: Data: d'229, addr: x'8c4
Memory.cpp:87:write() LOG: Data: d'233, addr: x'8c8
Memory.cpp:87:write() LOG: Data: d'239, addr: x'8cc
Memory.cpp:87:write() LOG: Data: d'241, addr: x'8d0
Memory.cpp:87:write() LOG: Data: d'251, addr: x'8d4
Memory.cpp:87:write() LOG: Data: d'257, addr: x'8d8
Memory.cpp:87:write() LOG: Data: d'263, addr: x'8dc
Memory.cpp:87:write() LOG: Data: d'269, addr: x'8e0
Memory.cpp:87:write() LOG: Data: d'271, addr: x'8e4
Memory.cpp:87:write() LOG: Data: d'277, addr: x'8e8
Memory.cpp:87:write() LOG: Data: d'281, addr: x'8ec
Memory.cpp:87:write() LOG: Data: d'283, addr: x'8f0
Memory.cpp:87:write() LOG: Data: d'293, addr: x'8f4
Memory.cpp:87:write() LOG: Data: d'307, addr: x'8f8
Memory.cpp:87:write() LOG: Data: d'311, addr: x'8fc
Memory.cpp:87:write() LOG: Data: d'313, addr: x'900
Memory.cpp:87:write() LOG: Data: d'317, addr: x'904
Memory.cpp:87:write() LOG: Data: d'331, addr: x'908
Memory.cpp:87:write() LOG: Data: d'337, addr: x'90c
Memory.cpp:87:write() LOG: Data: d'347, addr: x'910
Memory.cpp:87:write() LOG: Data: d'349, addr: x'914
Memory.cpp:87:write() LOG: Data: d'353, addr: x'918
Memory.cpp:87:write() LOG: Data: d'359, addr: x'91c
Memory.cpp:87:write() LOG: Data: d'367, addr: x'920
Memory.cpp:87:write() LOG: Data: d'373, addr: x'924
Memory.cpp:87:write() LOG: Data: d'379, addr: x'928
Memory.cpp:87:write() LOG: Data: d'383, addr: x'92c
Memory.cpp:87:write() LOG: Data: d'389, addr: x'930
Memory.cpp:87:write() LOG: Data: d'397, addr: x'934
Memory.cpp:87:write() LOG: Data: d'401, addr: x'938
Memory.cpp:87:write() LOG: Data: d'409, addr: x'93c
Memory.cpp:87:write() LOG: Data: d'419, addr: x'940
Memory.cpp:87:write() LOG: Data: d'421, addr: x'944
Memory.cpp:87:write() LOG: Data: d'431, addr: x'948
Memory.cpp:87:write() LOG: Data: d'433, addr: x'94c
Memory.cpp:87:write() LOG: Data: d'439, addr: x'950
Memory.cpp:87:write() LOG: Data: d'443, addr: x'954
Memory.cpp:87:write() LOG: Data: d'449, addr: x'958
Memory.cpp:87:write() LOG: Data: d'457, addr: x'95c
Memory.cpp:87:write() LOG: Data: d'461, addr: x'960
Memory.cpp:87:write() LOG: Data: d'463, addr: x'964
Memory.cpp:87:write() LOG: Data: d'467, addr: x'968
Memory.cpp:87:write() LOG: Data: d'479, addr: x'96c
Memory.cpp:87:write() LOG: Data: d'487, addr: x'970
Memory.cpp:87:write() LOG: Data: d'491, addr: x'974
Memory.cpp:87:write() LOG: Data: d'499, addr: x'978
Memory.cpp:87:write() LOG: Data: d'503, addr: x'97c
Memory.cpp:87:write() LOG: Data: d'509, addr: x'980
Memory.cpp:87:write() LOG: Data: d'521, addr: x'984
Memory.cpp:87:write() LOG: Data: d'523, addr: x'988
Memory.cpp:87:write() LOG: Data: d'541, addr: x'98c
Memory.cpp:87:write() LOG: Data: d'547, addr: x'990
Memory.cpp:87:write() LOG: Data: d'557, addr: x'994
Memory.cpp:87:write() LOG: Data: d'563, addr: x'998
Memory.cpp:87:write() LOG: Data: d'569, addr: x'99c
Memory.cpp:87:write() LOG: Data: d'571, addr: x'9a0
Memory.cpp:87:write() LOG: Data: d'577, addr: x'9a4
Memory.cpp:87:write() LOG: Data: d'587, addr: x'9a8
Memory.cpp:87:write() LOG: Data: d'593, addr: x'9ac
Memory.cpp:87:write() LOG: Data: d'599, addr: x'9b0
Memory.cpp:87:write() LOG: Data: d'601, addr: x'9b4
Memory.cpp:87:write() LOG: Data: d'607, addr: x'9b8
Memory.cpp:87:write() LOG: Data: d'613, addr: x'9bc
Memory.cpp:87:write() LOG: Data: d'617, addr: x'9c0
Memory.cpp:87:write() LOG: Data: d'619, addr: x'9c4
Memory.cpp:87:write() LOG: Data: d'631, addr: x'9c8
Memory.cpp:87:write() LOG: Data: d'641, addr: x'9cc
Memory.cpp:87:write() LOG: Data: d'643, addr: x'9d0
Memory.cpp:87:write() LOG: Data: d'647, addr: x'9d4
Memory.cpp:87:write() LOG: Data: d'653, addr: x'9d8
Memory.cpp:87:write() LOG: Data: d'659, addr: x'9dc
Memory.cpp:87:write() LOG: Data: d'661, addr: x'9e0
Memory.cpp:87:write() LOG: Data: d'673, addr: x'9e4
Memory.cpp:87:write() LOG: Data: d'677, addr: x'9e8
Memory.cpp:87:write() LOG: Data: d'683, addr: x'9ec
Memory.cpp:87:write() LOG: Data: d'691, addr: x'9f0
Memory.cpp:87:write() LOG: Data: d'701, addr: x'9f4
Memory.cpp:87:write() LOG: Data: d'709, addr: x'9f8
Memory.cpp:87:write() LOG: Data: d'719, addr: x'9fc
Memory.cpp:87:write() LOG: Data: d'727, addr: x'a00
Memory.cpp:87:write() LOG: Data: d'733, addr: x'a04
Memory.cpp:87:write() LOG: Data: d'739, addr: x'a08
Memory.cpp:87:write() LOG: Data: d'743, addr: x'a0c
Memory.cpp:87:write() LOG: Data: d'751, addr: x'a10
Memory.cpp:87:write() LOG: Data: d'757, addr: x'a14
Memory.cpp:87:write() LOG: Data: d'761, addr: x'a18
Memory.cpp:87:write() LOG: Data: d'769, addr: x'a1c
Memory.cpp:87:write() LOG: Data: d'773, addr: x'a20
Memory.cpp:87:write() LOG: Data: d'787, addr: x'a24
Memory.cpp:87:write() LOG: Data: d'797, addr: x'a28
Memory.cpp:87:write() LOG: Data: d'809, addr: x'a2c
Memory.cpp:87:write() LOG: Data: d'811, addr: x'a30
Memory.cpp:87:write() LOG: Data: d'821, addr: x'a34
Memory.cpp:87:write() LOG: Data: d'823, addr: x'a38
Memory.cpp:87:write() LOG: Data: d'827, addr: x'a3c
Memory.cpp:87:write() LOG: Data: d'829, addr: x'a40
Memory.cpp:87:write() LOG: Data: d'839, addr: x'a44
Memory.cpp:87:write() LOG: Data: d'853, addr: x'a48
Memory.cpp:87:write() LOG: Data: d'857, addr: x'a4c
Memory.cpp:87:write() LOG: Data: d'859, addr: x'a50
Memory.cpp:87:write() LOG: Data: d'863, addr: x'a54
Memory.cpp:87:write() LOG: Data: d'877, addr: x'a58
Memory.cpp:87:write() LOG: Data: d'881, addr: x'a5c
Memory.cpp:87:write() LOG: Data: d'883, addr: x'a60
Memory.cpp:87:write() LOG: Data: d'887, addr: x'a64
Memory.cpp:87:write() LOG: Data: d'907, addr: x'a68
Memory.cpp:87:write() LOG: Data: d'911, addr: x'a6c
Memory.cpp:87:write() LOG: Data: d'919, addr: x'a70
Memory.cpp:87:write() LOG: Data: d'929, addr: x'a74
Memory.cpp:87:write() LOG: Data: d'937, addr: x'a78
Memory.cpp:87:write() LOG: Data: d'941, addr: x'a7c
Memory.cpp:87:write() LOG: Data: d'947, addr: x'a80
Memory.cpp:87:write() LOG: Data: d'953, addr: x'a84
Memory.cpp:87:write() LOG: Data: d'967, addr: x'a88
Memory.cpp:87:write() LOG: Data: d'971, addr: x'a8c
Memory.cpp:87:write() LOG: Data: d'977, addr: x'a90
Memory.cpp:87:write() LOG: Data: d'983, addr: x'a94
Memory.cpp:87:write() LOG: Data: d'991, addr: x'a98
Memory.cpp:87:write() LOG: Data: d'997, addr: x'a9c
Memory.cpp:87:write() LOG: Data: d'1, addr: x'aa0
Memory.cpp:87:write() LOG: Data: d'4, addr: x'aa4
Memory.cpp:87:write() LOG: Data: d'9, addr: x'aa8
Memory.cpp:87:write() LOG: Data: d'16, addr: x'aac
Memory.cpp:87:write() LOG: Data: d'25, addr: x'ab0
Memory.cpp:87:write() LOG: Data: d'36, addr: x'ab4
Memory.cpp:87:write() LOG: Data: d'49, addr: x'ab8
Memory.cpp:87:write() LOG: Data: d'64, addr: x'abc
Memory.cpp:87:write() LOG: Data: d'81, addr: x'ac0
Memory.cpp:87:write() LOG: Data: d'100, addr: x'ac4
Memory.cpp:87:write() LOG: Data: d'121, addr: x'ac8
Memory.cpp:87:write() LOG: Data: d'144, addr: x'acc
Memory.cpp:87:write() LOG: Data: d'169, addr: x'ad0
Memory.cpp:87:write() LOG: Data: d'196, addr: x'ad4
Memory.cpp:87:write() LOG: Data: d'225, addr: x'ad8
Memory.cpp:87:write() LOG: Data: d'256, addr: x'adc
Memory.cpp:87:write() LOG: Data: d'289, addr: x'ae0
Memory.cpp:87:write() LOG: Data: d'324, addr: x'ae4
Memory.cpp:87:write() LOG: Data: d'361, addr: x'ae8
Memory.cpp:87:write() LOG: Data: d'400, addr: x'aec
Memory.cpp:87:write() LOG: Data: d'441, addr: x'af0
Memory.cpp:87:write() LOG: Data: d'484, addr: x'af4
Memory.cpp:87:write() LOG: Data: d'529, addr: x'af8
Memory.cpp:87:write() LOG: Data: d'576, addr: x'afc
Memory.cpp:87:write() LOG: Data: d'625, addr: x'b00
Memory.cpp:87:write() LOG: Data: d'676, addr: x'b04
Memory.cpp:87:write() LOG: Data: d'729, addr: x'b08
Memory.cpp:87:write() LOG: Data: d'784, addr: x'b0c
Memory.cpp:87:write() LOG: Data: d'841, addr: x'b10
Memory.cpp:87:write() LOG: Data: d'900, addr: x'b14
Memory.cpp:87:write() LOG: Data: d'961, addr: x'b18
Memory.cpp:87:write() LOG: Data: d'1024, addr: x'b1c
Memory.cpp:87:write() LOG: Data: d'1089, addr: x'b20
Memory.cpp:87:write() LOG: Data: d'1156, addr: x'b24
Memory.cpp:87:write() LOG: Data: d'1225, addr: x'b28
Memory.cpp:87:write() LOG: Data: d'1296, addr: x'b2c
Memory.cpp:87:write() LOG: Data: d'1369, addr: x'b30
Memory.cpp:87:write() LOG: Data: d'1444, addr: x'b34
Memory.cpp:87:write() LOG: Data: d'1521, addr: x'b38
Memory.cpp:87:write() LOG: Data: d'1600, addr: x'b3c
Memory.cpp:87:write() LOG: Data: d'1681, addr: x'b40
Memory.cpp:87:write() LOG: Data: d'1764, addr: x'b44
Memory.cpp:87:write() LOG: Data: d'1849, addr: x'b48
Memory.cpp:87:write() LOG: Data: d'1936, addr: x'b4c
Memory.cpp:87:write() LOG: Data: d'2025, addr: x'b50
Memory.cpp:87:write() LOG: Data: d'2116, addr: x'b54
Memory.cpp:87:write() LOG: Data: d'2209, addr: x'b58
Memory.cpp:87:write() LOG: Data: d'2304, addr: x'b5c
Memory.cpp:87:write() LOG: Data: d'2401, addr: x'b60
Memory.cpp:87:write() LOG: Data: d'2500, addr: x'b64
Memory.cpp:87:write() LOG: Data: d'2601, addr: x'b68
Memory.cpp:87:write() LOG: Data: d'2704, addr: x'b6c
Memory.cpp:87:write() LOG: Data: d'2809, addr: x'b70
Memory.cpp:87:write() LOG: Data: d'2916, addr: x'b74
Memory.cpp:87:write() LOG: Data: d'3025, addr: x'b78
Memory.cpp:87:write() LOG: Data: d'3136, addr: x'b7c
Memory.cpp:87:write() LOG: Data: d'3249, addr: x'b80
Memory.cpp:87:write() LOG: Data: d'3364, addr: x'b84
Memory.cpp:87:write() LOG: Data: d'3481, addr: x'b88
Memory.cpp:87:write() LOG: Data: d'3600, addr: x'b8c
Memory.cpp:87:write() LOG: Data: d'3721, addr: x'b90
Memory.cpp:87:write() LOG: Data: d'3844, addr: x'b94
Memory.cpp:87:write() LOG: Data: d'3969, addr: x'b98
Memory.cpp:87:write() LOG: Data: d'4096, addr: x'b9c
Memory.cpp:87:write() LOG: Data: d'4225, addr: x'ba0
Memory.cpp:87:write() LOG: Data: d'4356, addr: x'ba4
Memory.cpp:87:write() LOG: Data: d'4489, addr: x'ba8
Memory.cpp:87:write() LOG: Data: d'4624, addr: x'bac
Memory.cpp:87:write() LOG: Data: d'4761, addr: x'bb0
Memory.cpp:87:write() LOG: Data: d'4900, addr: x'bb4
Memory.cpp:87:write() LOG: Data: d'5041, addr: x'bb8
Memory.cpp:87:write() LOG: Data: d'5184, addr: x'bbc
Memory.cpp:87:write() LOG: Data: d'5329, addr: x'bc0
Memory.cpp:87:write() LOG: Data: d'5476, addr: x'bc4
Memory.cpp:87:write() LOG: Data: d'5625, addr: x'bc8
Memory.cpp:87:write() LOG: Data: d'5776, addr: x'bcc
Memory.cpp:87:write() LOG: Data: d'5929, addr: x'bd0
Memory.cpp:87:write() LOG: Data: d'6084, addr: x'bd4
Memory.cpp:87:write() LOG: Data: d'6241, addr: x'bd8
Memory.cpp:87:write() LOG: Data: d'6400, addr: x'bdc
Memory.cpp:87:write() LOG: Data: d'6561, addr: x'be0
Memory.cpp:87:write() LOG: Data: d'6724, addr: x'be4
Memory.cpp:87:write() LOG: Data: d'6889, addr: x'be8
Memory.cpp:87:write() LOG: Data: d'7056, addr: x'bec
Memory.cpp:87:write() LOG: Data: d'7225, addr: x'bf0
Memory.cpp:87:write() LOG: Data: d'7396, addr: x'bf4
Memory.cpp:87:write() LOG: Data: d'7569, addr: x'bf8
Memory.cpp:87:write() LOG: Data: d'7744, addr: x'bfc
Memory.cpp:87:write() LOG: Data: d'7921, addr: x'c00
Memory.cpp:87:write() LOG: Data: d'8100, addr: x'c04
Memory.cpp:87:write() LOG: Data: d'8281, addr: x'c08
Memory.cpp:87:write() LOG: Data: d'8464, addr: x'c0c
Memory.cpp:87:write() LOG: Data: d'8649, addr: x'c10
Memory.cpp:87:write() LOG: Data: d'8836, addr: x'c14
Memory.cpp:87:write() LOG: Data: d'9025, addr: x'c18
Memory.cpp:87:write() LOG: Data: d'9216, addr: x'c1c
Memory.cpp:87:write() LOG: Data: d'9409, addr: x'c20
Memory.cpp:87:write() LOG: Data: d'9604, addr: x'c24
```

Listing E.1: log/main_compressed

Bibliography

[1] A. Weddell, “Elec2204 computer engineering: Datapaths,” 2020.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fourth Edition*, 4th ed. Elsevier Science, 2008.

[3] P. Gillard, “Cs3725: Computer architecture review,” 2015. [Online]. Available: http://web.cs.mun.ca/~paul/cs3725/material/review.pdf

[4] 2019. [Online]. Available: https://web.archive.org/web/20190222083348/http://logos.cs.uic.edu/366/notes/MIPS%20Quick%20Tutorial.htm

[5] J. Frenzel, “Mips instruction reference,” 1998. [Online]. Available: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

[6] [Online]. Available: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes