COMPUTER ORGANIZATION (FALL 2018)

LAB 1 REPORT

GROUP 4

1.0 INTRODUCTION

In the lab 1 assignment, we were given a MIPS simulator skeleton in C language, header file, makefile and input files. Our task was to implement handle_instruction and print_instruction function in the .c file. The handle_instruction function consists of the logic we use in implementing the given instructions (in our case the input 1, 2 and 3) whereas the print_instruction function contains the implementation of printing the instructions in MIPS assembly format.

2.0 IMPLEMENTATION

2.1 HANDLE INSTRUCTION

Based on the inputs we were given, we implemented the following instructions: LUI, ADDIU, ADD, SW, LW, C, OR, SLL, SUBU, SUB, XOR, XORI, SRL, SRA, AND, ANDI, J, BNE, BEQ, and ORI.

Steps:

1. Fetch instruction.
2. Decode the opcode by masking.
3. Decode the register numbers by masking and shifting right.
4. Decode the shift amount by masking and shifting right.
5. Compare the opcode to find the instruction format.
6. Find and execute the instruction.
7. Update the registers and/or the memory.

Implementation Decisions:

LUI: In the code we shifted the immediate value left by 16 bits and added the value to the RT register. We shifted by 16bit left to move the content in the lower 16 bits to the upper 16bit.

ADDIU: We added the content of RS and the unsigned immediate value into destination register RT. We left the immediate value unsigned because the instruction does not work with signed value.

ADD: We added the content of Rs and Rt register into Rd register.

SW: We sign extended the immediate value to get the offset and we added it to the value of the base register RS to get the address. Thereafter, since the data segment started from 0x10010000 and we have to store data in the data segment, we added our address to 0x10010000 to generate the appropriate address.

LW: We sign extended the immediate value to get the offset and we added it to the value of the base register RS to get the address. Since the data segment started from 0x10010000 and we have to Load word from the data segment, we added our address to 0x10010000 to generate the appropriate address.

SLL/ SRA: Since the instruction required shifting left, we shifted the content in Rs register by the shift amount value.

SUBU/SUB: We subtracted the content of Rt register from Rs register and added the result to the Rd register.

XOR: We exclusively OR the Rs and Rt register and added the value to the Rd register.

XORI: Since XORI is a logical operation, we left the immediate value unsigned. Just like with our XOR case, we exclusively OR the Rs and Rt and added the content to the Rd register.

SRL: Since the instruction is shift right logical, we shifted right the value in the Rt register by the shift amount value.

AND: We AND the content of the Rs and Rt register and stored it in the Rd register.

ANDI: We zero extended the immediate value since it is a logical operation and then, we added the immediate value to the content Rs register and stored the value in Rt register.

J: We masked the first 4 bit of the opcode and added the value to the 26 bit (shifted by two to the left) to get the jump address.

BNE: We first sign-extended the immediate value. If Rs equals Rt, we branch to the target address calculated by adding the PC value to the immediate value, else, we increment the program counter by 4.

BEQ: We first sign-extended the immediate value. If Rs does not equal to Rt, we branch to the target address calculated by adding the PC value to the immediate value, else, we increment the program counter by 4.

ORI: We zero extended the immediate value since it is a logical operation. Then, OR the immediate value with the content of Rs register and added the result to Rt register.

C: It is called system call. When the program encounters it, the run flag is set to false and the program terminate.

Note: We also implemented some other instructions we came accross in the LAB 1 manual but commented it out since they were not part of the input instructions.

2.2 PRINT INSTRUCTION:

Based on the inputs we were given, we printed the following instructions in MIPS assembly language format for: LUI, ADDIU, ADD, SW, LW, C, OR, SLL, SUBU, SUB, XOR, XORI, SRL, SRA, AND, ANDI, J, BNE, BEQ, and ORI.

Steps:

1. Fetch instruction
2. Decode the opcode by masking.
3. Decode the register numbers by masking and shifting right.
4. Decode the shift amount by masking and shifting right.
5. Compare the opcode to find the instruction format.
6. Print the instruction in MIPS assembly language such that the register values are part of the print**.**

3.0 SIMULATED RESULT

The result of the program we executed are as shown below:

3. 1 Input: test1

3.1.1 register dump result

```
File  Edit  View  Search  Terminal  Help
Opcode = 0x00000000
 This is a R-type instruction
 rs = 0
 rt = 0
 rd = 0
 sa = 0
 function = 0x0000000c
Simulation Finished.

MU-MIPS SIM:> rdump
-------------------------------------
Dumping Register Content
-------------------------------------
# Instructions Executed : 32
PC       : 0x00400080
-------------------------------------
[Register]       [Value]
-------------------------------------
[R0]     : 0x00000000
[R1]     : 0x00000000
[R2]     : 0x0000000a
[R3]     : 0x10000004
[R4]     : 0x00000000
[R5]     : 0x000000ff
[R6]     : 0x000001fe
[R7]     : 0x000003fc
[R8]     : 0x0000792c
[R9]     : 0x000000ff
[R10]    : 0x000001fe
[R11]    : 0x000003fc
[R12]    : 0x0000792c
[R13]    : 0x000000ff
[R14]    : 0x000000ff
[R15]    : 0x000001fe
[R16]    : 0x000003fc
[R17]    : 0x0000881d
[R18]    : 0x00000000
[R19]    : 0x00000000
[R20]    : 0x00000000
[R21]    : 0x00000000
[R22]    : 0x00000000
[R23]    : 0x00000000
[R24]    : 0x00000000
[R25]    : 0x00000000
[R26]    : 0x00000000
[R27]    : 0x00000000
[R28]    : 0x00000000
[R29]    : 0x00000000
[R30]    : 0x00000000
[R31]    : 0x00000000
-------------------------------------
[HI]     : 0x00000000
[LO]     : 0x00000000
-------------------------------------
MU-MIPS SIM:>
```

Fig 1: Screen shot of rdump (test1)

3.1.2 Memory dump result

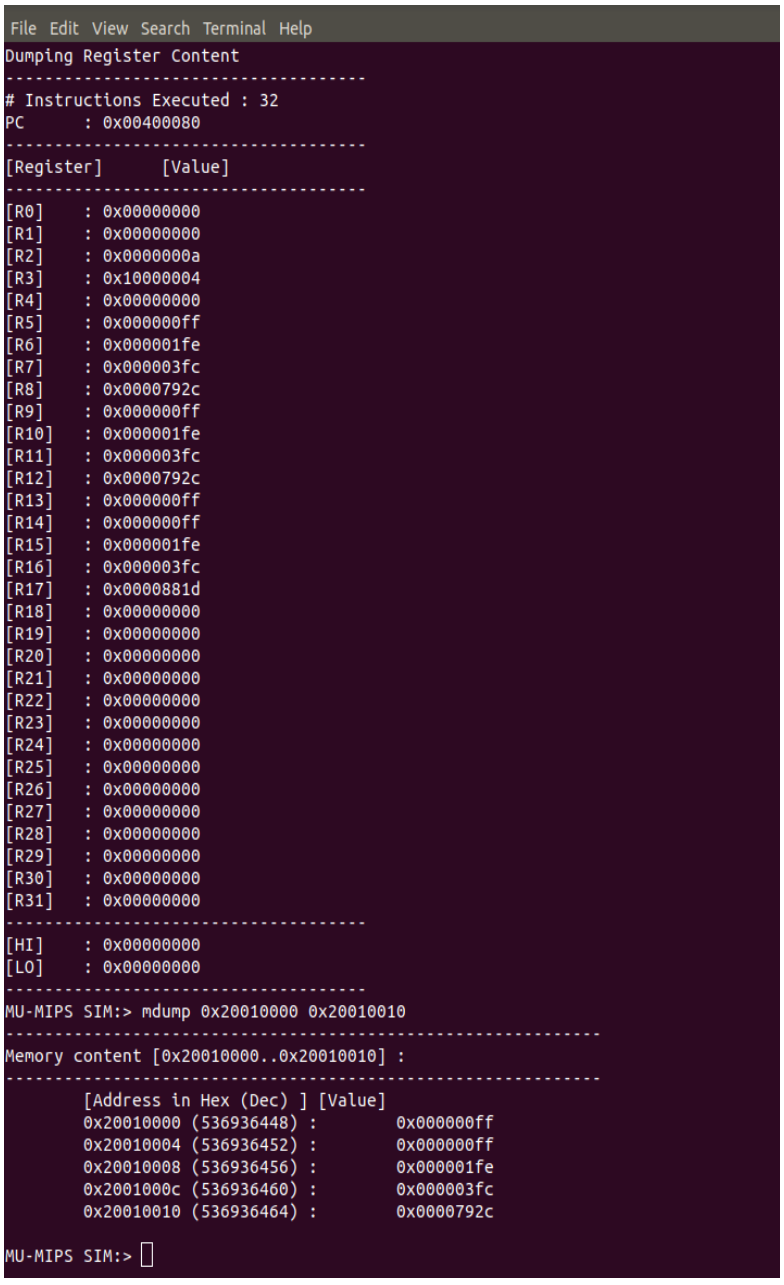

Fig 2: Screen shot of mdump (test1)

3.1.3 Print result

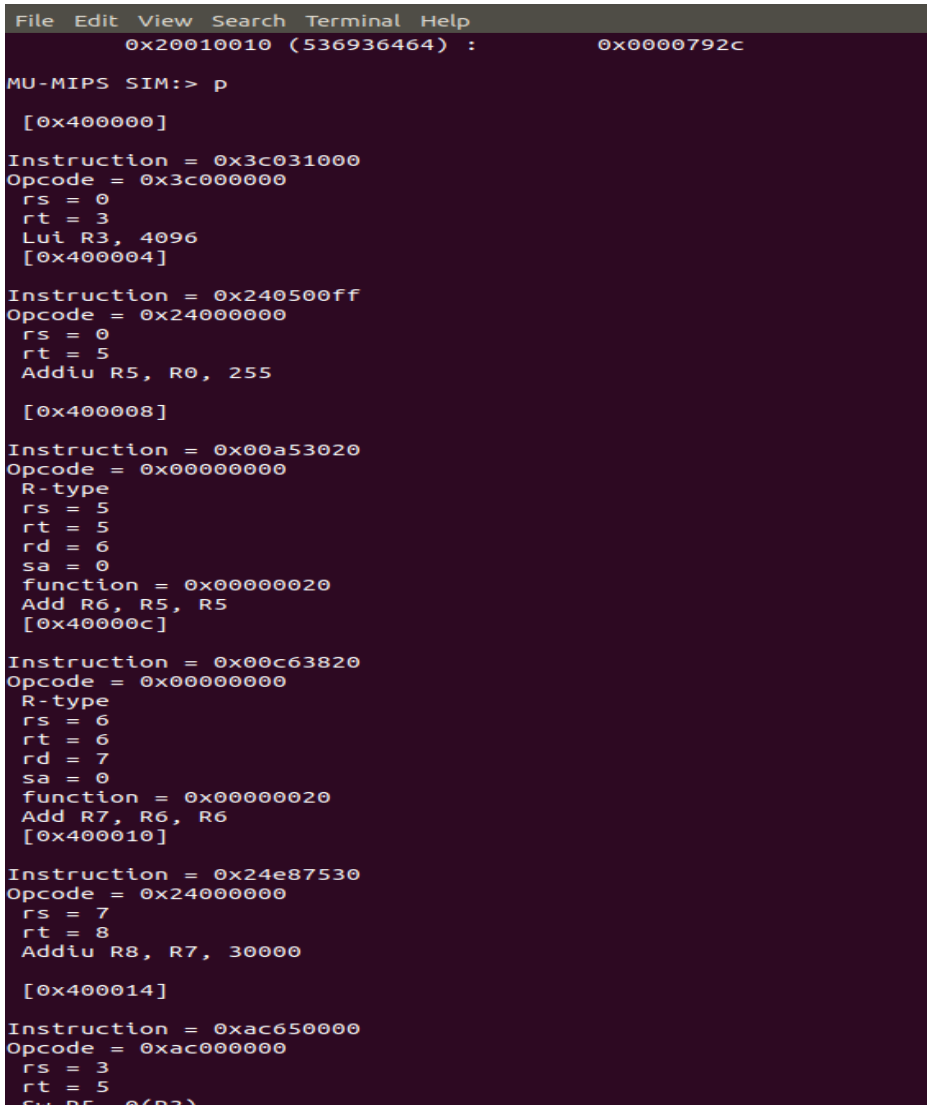

Fig 3: Screen shot of print result (test1)

Note: Only first few instructions of the print result are shown in the screen shot above.

3. 2 Input: test2

3.2.1 Register dump result

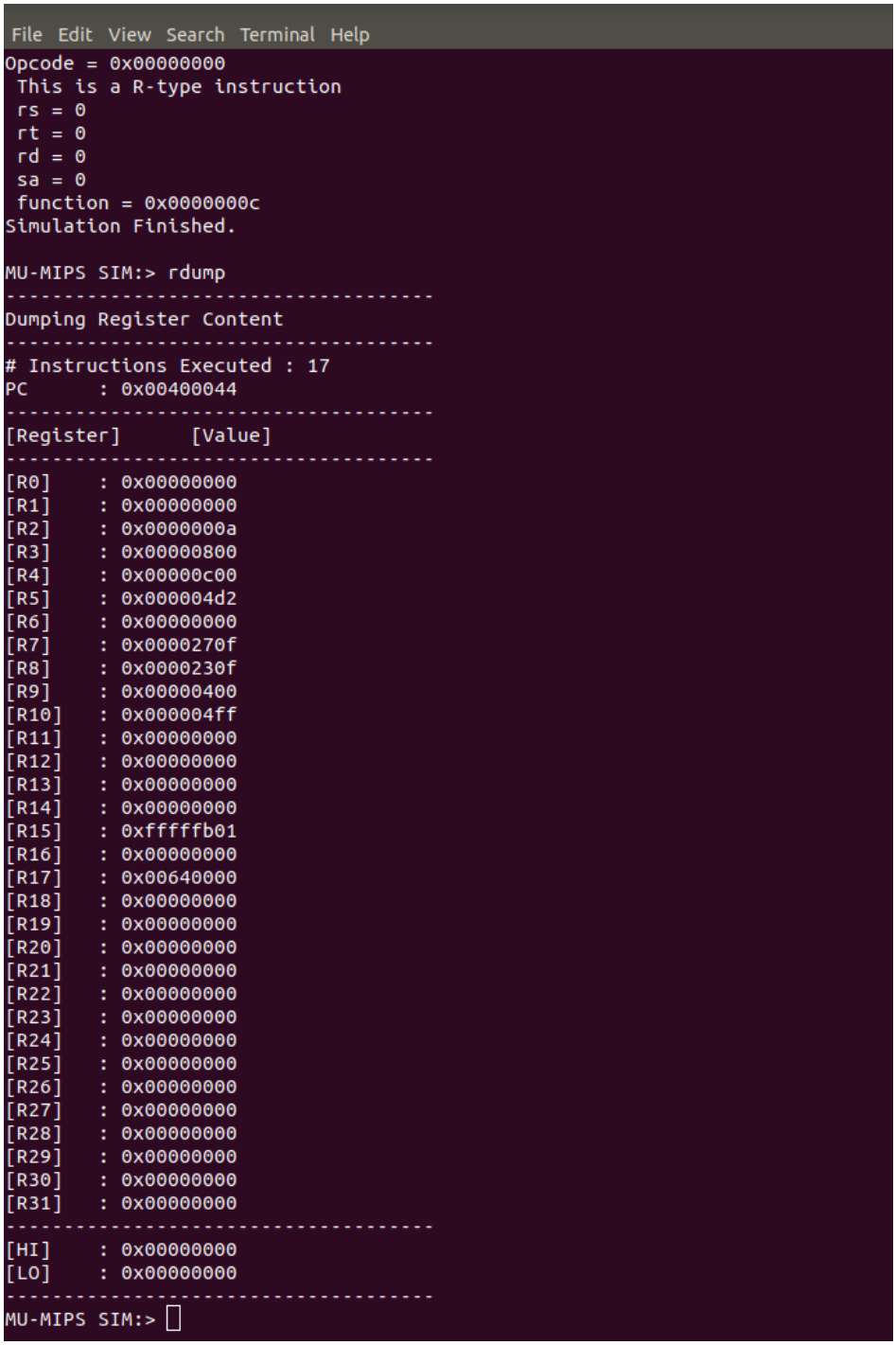

Fig 4: Screen shot of rdump (test2)

3.2.2 Print result

```
File  Edit  View  Search  Terminal  Help
-------------------------------------
[HI]    : 0x00000000
[LO]    : 0x00000000
-------------------------------------
MU-MIPS SIM:> p

 [0x400000]

Instruction = 0x24020400
Opcode = 0x24000000
 rs = 0
 rt = 2
 Addiu R2, R0, 1024

 [0x400004]

Instruction = 0x00421821
Opcode = 0x00000000
 R-type
 rs = 2
 rt = 2
 rd = 3
 sa = 0
 function = 0x00000021
 Addu R3, R2, R2
 [0x400008]

Instruction = 0x00622025
Opcode = 0x00000000
 R-type
 rs = 3
 rt = 2
 rd = 4
 sa = 0
 function = 0x00000025
 Or R4, R3, R2
 [0x40000c]

Instruction = 0x200504d2
Opcode = 0x20000000
 rs = 0
 rt = 5
 Addi R5, R0, 1234
 [0x400010]

Instruction = 0x00053400
Opcode = 0x00000000
 R-type
 rs = 0
 rt = 5
 rd = 6
 sa = 16
 function = 0x00000000
 Sll R6, R0, 16
 [0x400014]
```

Fig 5: Screen shot of print result (test2)

Note: Only first few instructions of the print result are shown in the screen shot above.

3.3 Input: test3

3.3.1 Register dump result

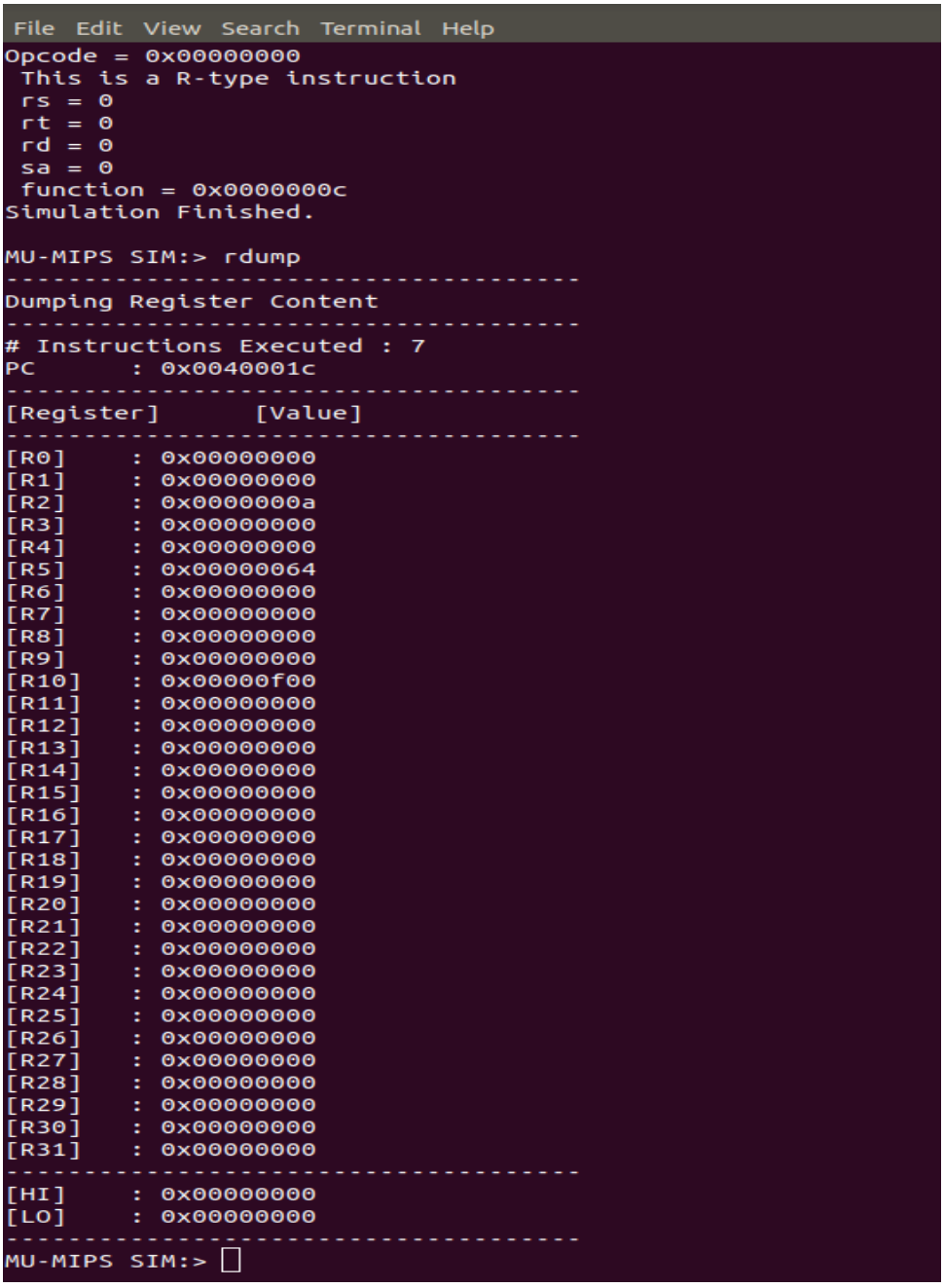

Fig 6: Screen shot of rdump (test3)

3.3.2 Print result

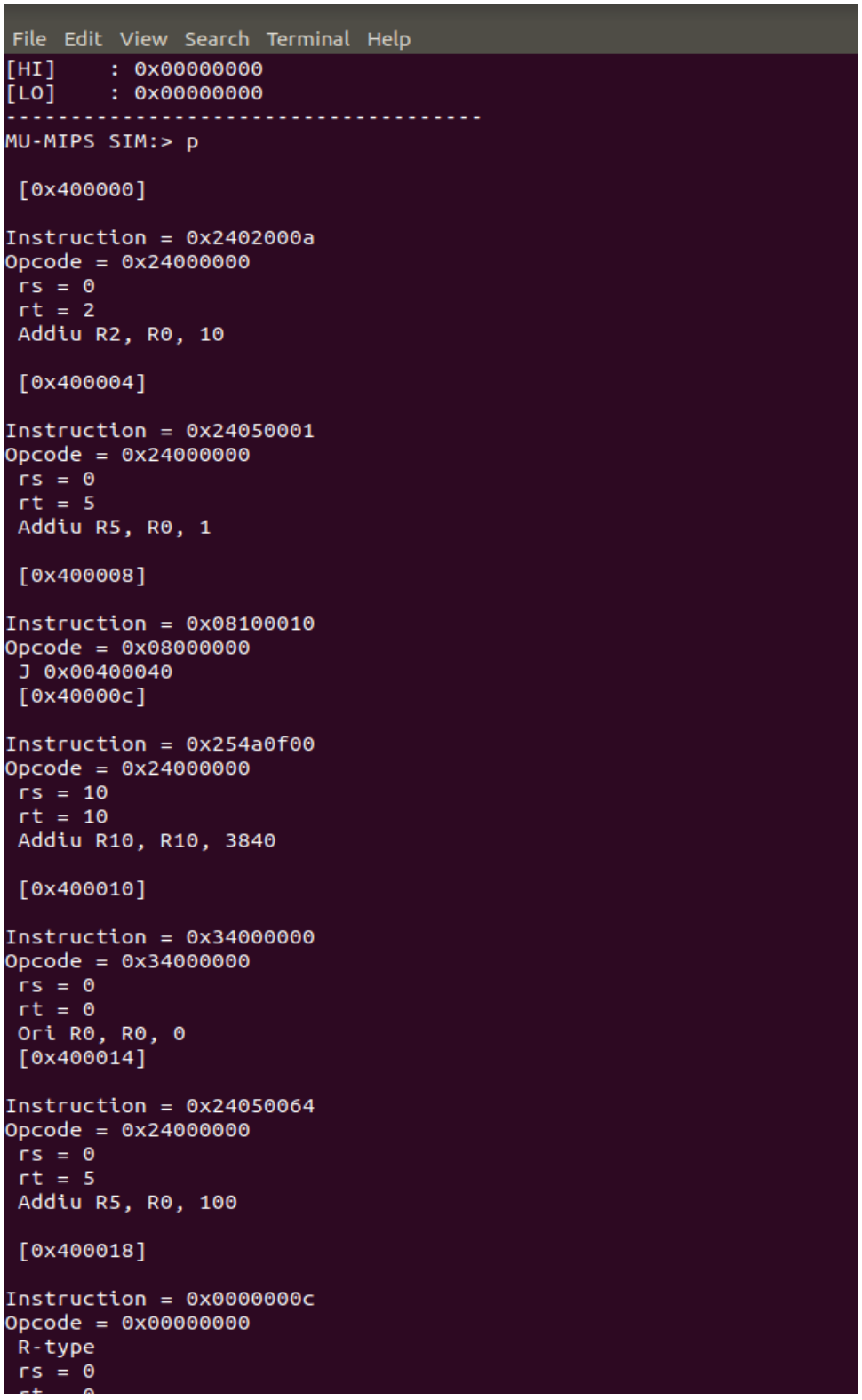

Fig 7: Screen shot of print result (test3)

4.0 Milestone

4. 1 First week: In the first week, we spent the first few days decoding the instruction manually and figuring out how to decode the instruction in c program language. The rest of the week was spent implementing the R type instruction under the handle instruction function.

4.2 Second week: We completed the implementation of I and j format instructions. We also implemented the print program function. Further, we did simulation of the code where we obtained the Rdump, Mdump and print result. Finally, we wrote the Lab report.

5.0 Work Distribution

Tochukwu Idika: She focused majorly in decoding the instructions manually and figuring out how the instructions can be implemented in C language. She also participated in writing the lab 1 report.

Shradha Shalini: She focused majorly in understanding and implementing the code in c language and she cleaned the code. She also participated in the writing of the lab 1 report.

Pooneh Safayenikoo: She focused more in debugging errors in the code. She also participated in writing the lab 1 report.

6.0 Conclusion

In summary, the lab was challenging yet fulfilling at the end. In course of doing the lab work, we learnt how to decode instructions manually as well as in c language. We equally learnt how to implement the instructions to execute the required operations. Furthermore, we learnt how to print the instructions in MIPS assembly format. Finally, we learnt how the memory works and how to successfully declare an address such that information can be read and writing to the memory.