|  |
| --- |
| 台科大 |
| HW1 |
| EE3005301 計算機組織 |

|  |
| --- |
| B11032006周 柏宇  2024/3/13 |

內容

[Modules 2](#_Toc161370484)

[Register File 2](#_Toc161370485)

[Arithmetic\_Logic\_Unit 3](#_Toc161370486)

[Complete ALU 5](#_Toc161370487)

[Test Commands 6](#_Toc161370488)

[Helper Program 6](#_Toc161370489)

[Simple Compiler 6](#_Toc161370490)

[Verify Program 8](#_Toc161370491)

[Test Commands 10](#_Toc161370492)

[Test Results 12](#_Toc161370493)

[RF 12](#_Toc161370494)

[ALU 12](#_Toc161370495)

[CmpALU 13](#_Toc161370496)

[Tests for srl 13](#_Toc161370497)

[Tests for and 13](#_Toc161370498)

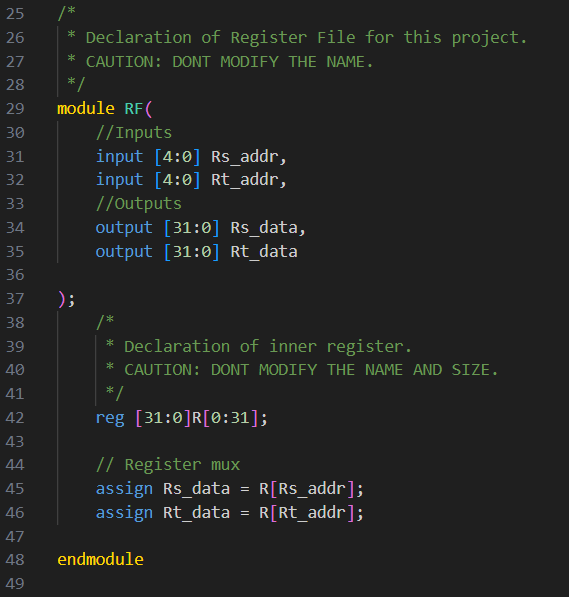
[Tests for subu 13](#_Toc161370499)

[Tests for addu 14](#_Toc161370500)

[Conclusion 15](#_Toc161370501)

# Modules

## Register File



Line 29~37

Declaration and I/O.

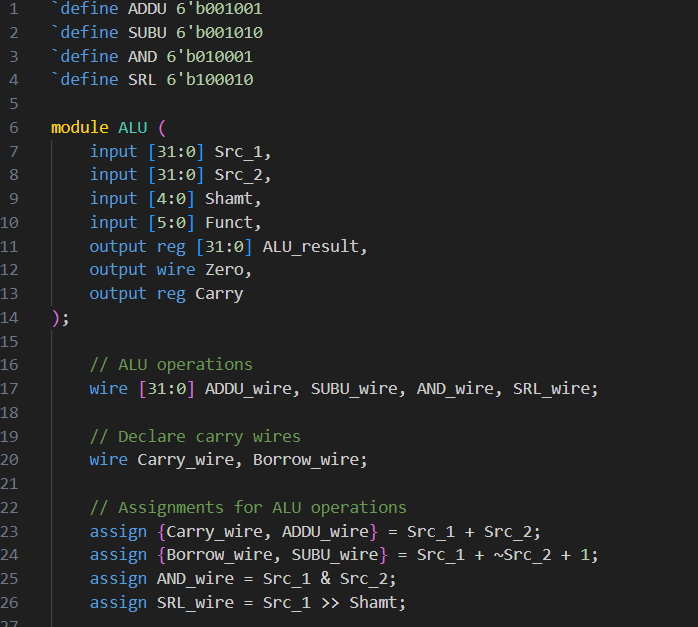
Line 42

Allocate 32 x 32-bits sram with named R.

Line 45~46

Rs\_addr, Rt\_addr are registers’ index (5 bits = [0, 31]). Use square brackets to fetch data (32 bits = [0, 2^31]).

## Arithmetic\_Logic\_Unit



Line 1~4

Define constants.

Line 17, 20

Use wires to store results.

Line 24

Use 2’s complement to implement subtraction in order to get borrow bit.

Line 23~26

Use “assign” to get all the corresponding results simultaneously.



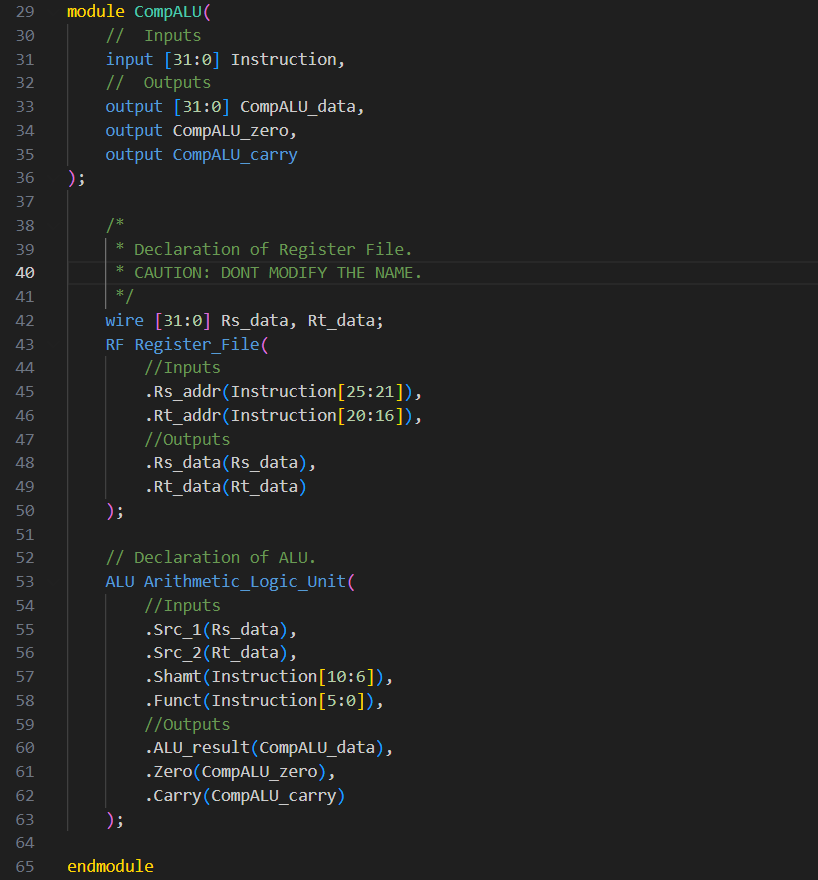
Line 30 ~ 51

Use conditional statement to decide which data should be output.

Line 54

Set Zero flag.

## Complete ALU



Line 42

Use wire to communicate between multiple components.

Line 43~50

Declare an RF instance with corresponding I/O.

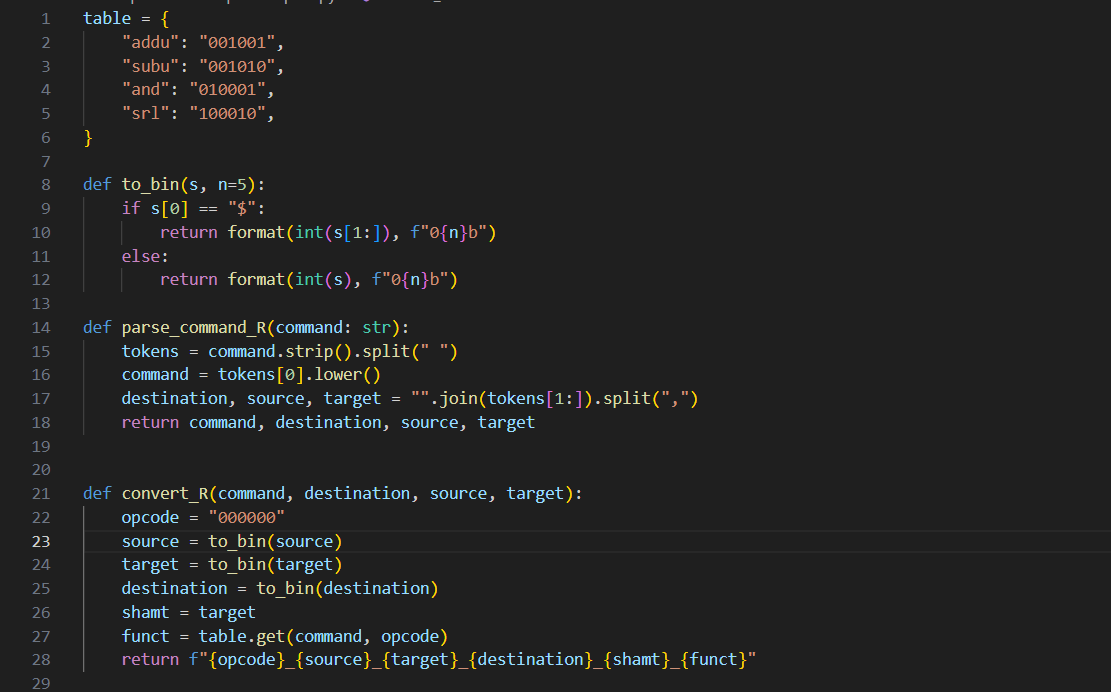
Line 53~63

Declare an ALU instance with corresponding I/O.

# Test Commands

## Helper Program

### Simple Compiler



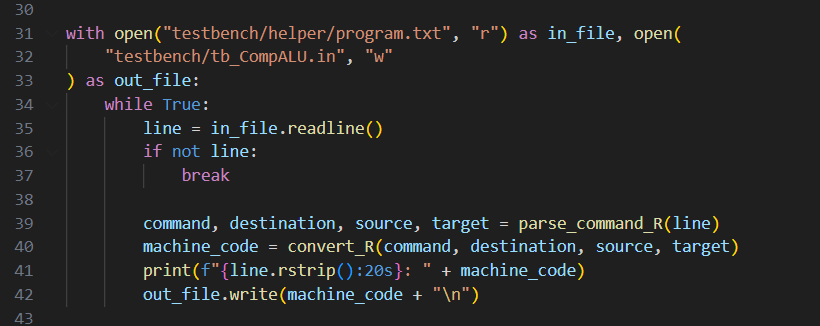
# This program convert assembly to machine code.

Line 1~6

Machine code look-up table.

Line 8~28

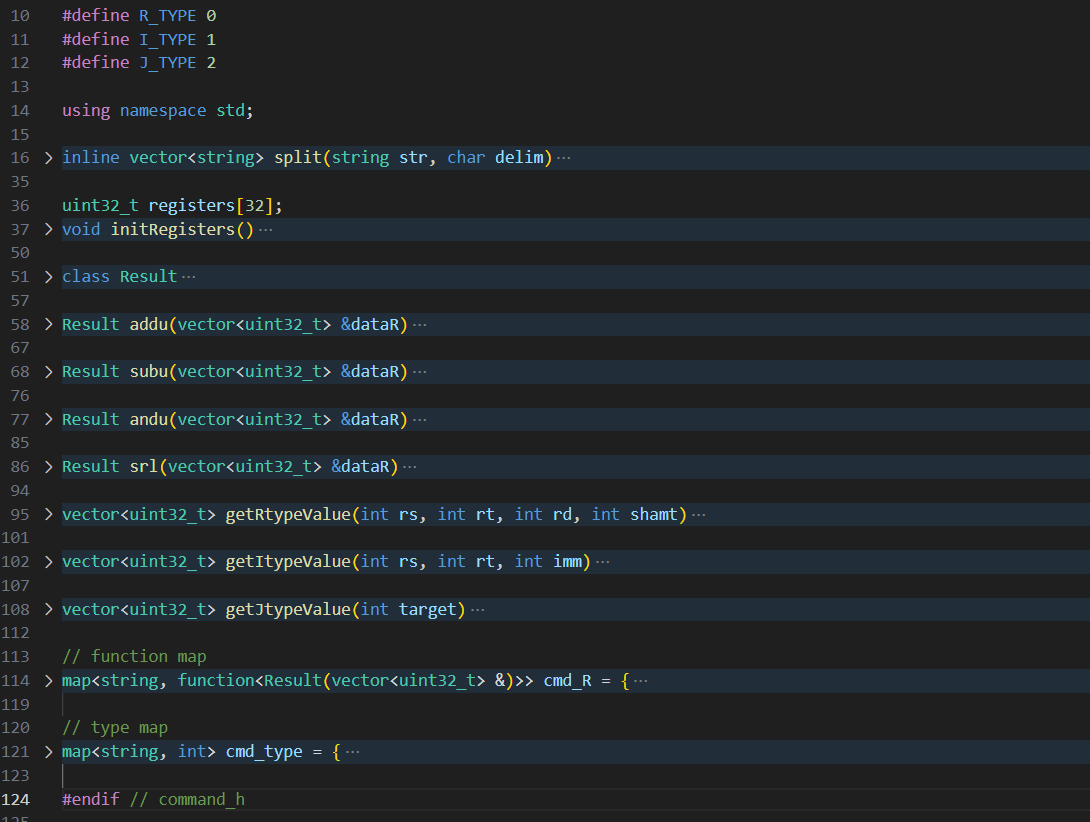
Simple parser. Since target and shamt will not be used at the same time, I set them to be the same value.



Line 31~42

Procedure.

### Verify Program



# This program get program from machine code and output from modelsim to verify ans.

Line 16~49

Utility function. There is no need for 2 functions to create a new file, so they’re here.

Line 51~56

Wrapper for value, zero, and carry.

Line 58~93

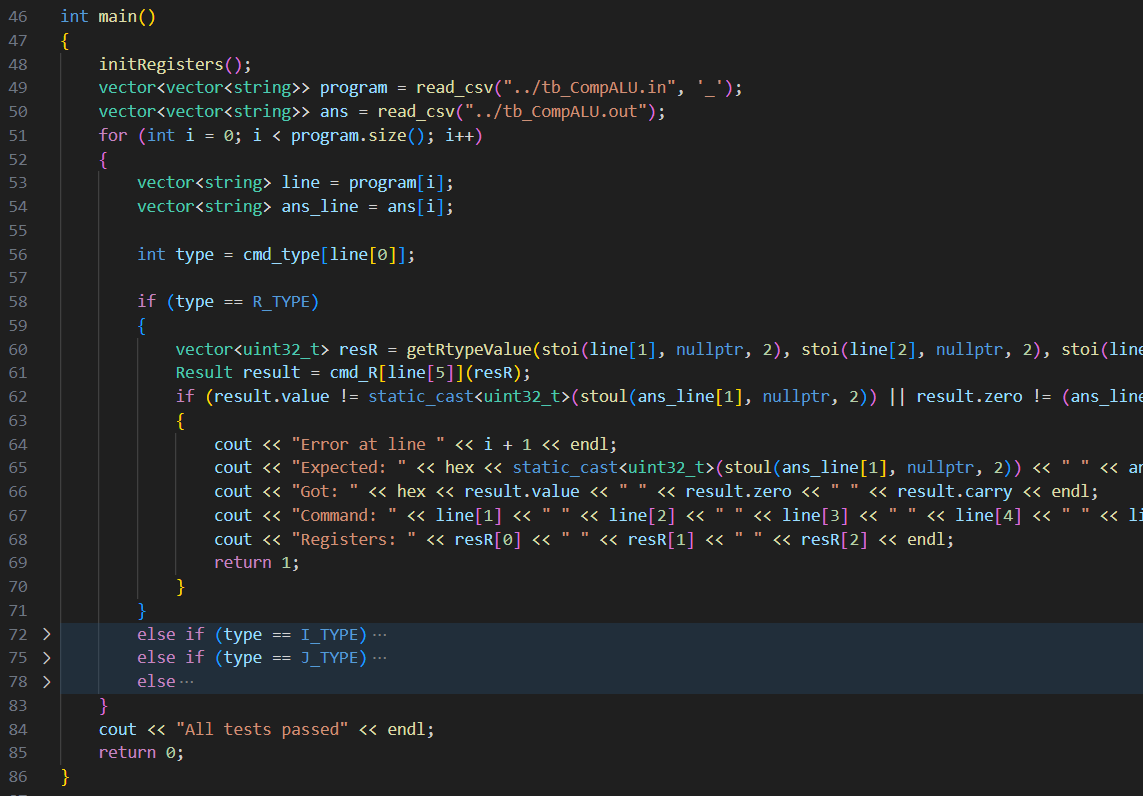
Functions implemented in this homework.

Line 95~111

Instruction’s type to value decomposer. For simplicity, all types return the same data type.

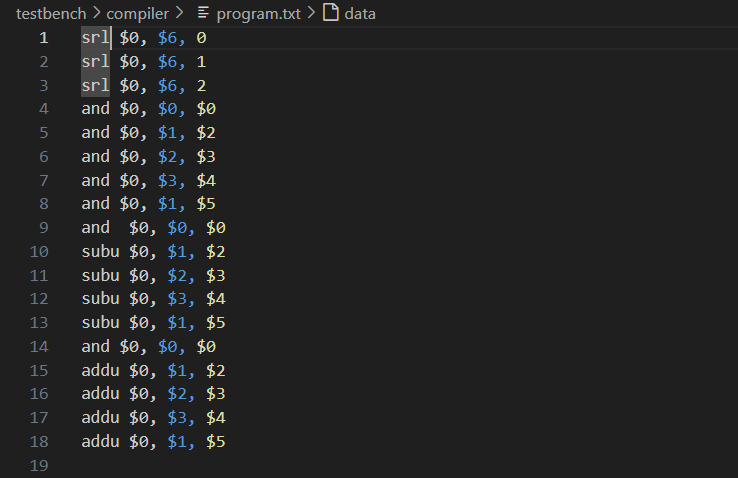
Line 114~122

Maps.

Line 48~84

Verify.

## Test Commands



#Note: it’s colored because I have csv extension.

Line 4, 9, 14

“and $0, $0, $0” is to zero the output acting as separator.

Line 1~3

Tests for srl.

Line 5~8

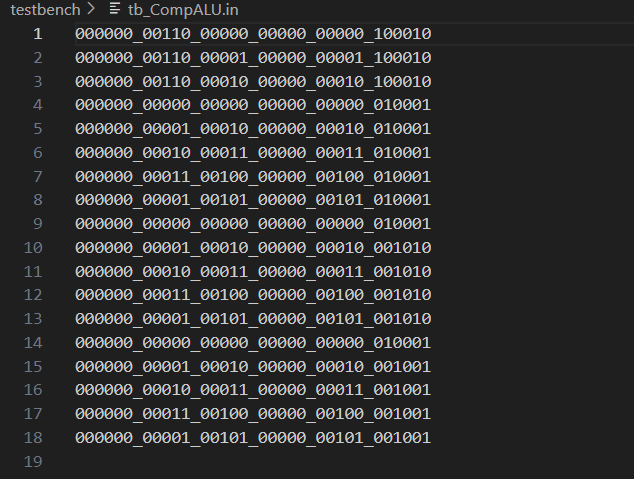
Tests for and.

Line 10~13

Tests for subu. Including both having borrow or not.

Line 15~18

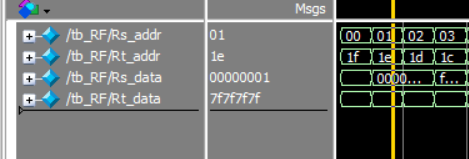
Tests for addu. Including both overflowing or not.



Corresponding machine code.

# Test Results

## RF

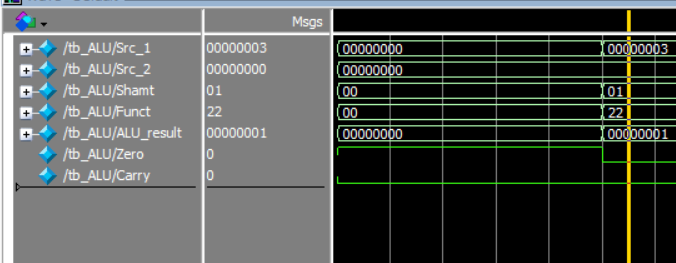


Rs\_addr: 0x01 = 110, Rs\_data: 0x000001

Rt\_addr: 0x1e = 3010, Rt\_data: 0x7f7f7f7f

Initial register data is in ./testbench/RF.dat

## ALU

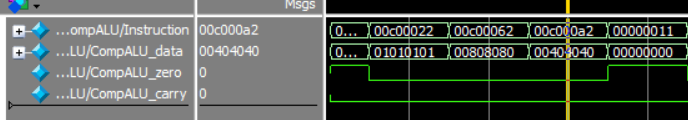


Funct = 100010 in this hw means srl

Shift right logically perform on Src\_1 by Shamt bits: 0b11 >> 1 = 0b01 = 110.

## CmpALU

### Tests for srl



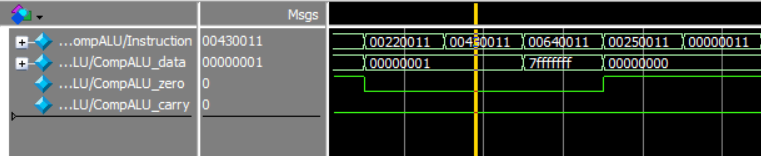
#Note and $0, $0, $0 in machine code is 0x00000011. It’s acted as sperator.

srl $0, $6, 0: 0x0101\_0101 >> 0x0 = 0x0101\_0101 (equivalent as $6 / 2^0)

srl $0, $6, 1: 0x0101\_0101 >> 0x1 = 0x0080\_8080 (equivalent as $6 / 2^1)

srl $0, $6, 2: 0x0101\_0101 >> 0x2 = 0x0040\_4040 (equivalent as $6 / 2^2)

### Tests for and



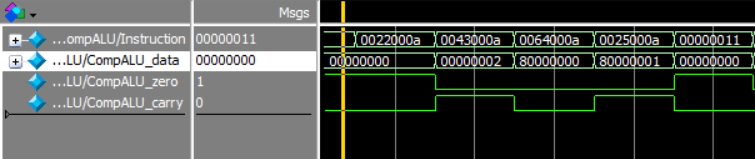
and $0, $1, $2: 0x0000\_0001 & 0x0000\_0001 = 0x0000\_0001

and $0, $2, $3: 0x0000\_0001 & 0xFFFF\_FFFF = 0x0000\_0001

and $0, $3, $4: 0xFFFF\_FFFF & 0x7FFF\_FFFF = 0x7FFF\_FFFF

and $0, $1, $5: 0x0000\_0001 & 0x8000\_0000 = 0x0000\_0000

### Tests for subu



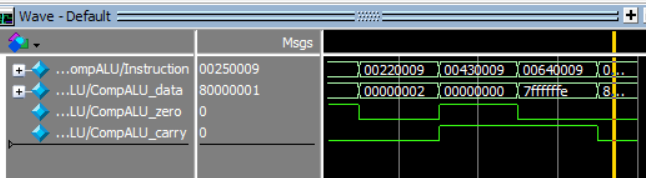
subu $0, $1, $2: 0x0000\_0001 - 0x0000\_0001 = 0x0000\_0000, Z = 1, C = 0

subu $0, $2, $3: 0x0000\_0001 - 0xFFFF\_FFFF = 0x0000\_0002, Z = 0, C = 1

subu $0, $3, $4: 0xFFFF\_FFFF - 0x7FFF\_FFFF = 0x8000\_0000, Z = 0, C = 0

subu $0, $1, $5: 0x0000\_0001 - 0x8000\_0000 = 0x8000\_0001, Z = 0, C = 1

### Tests for addu



addu $0, $1, $2: 0x0000\_0001 + 0x0000\_0001 = 0x0000\_0002

addu $0, $2, $3: 0x0000\_0001 + 0xFFFF\_FFFF = 0x0000\_0000, Z = 1, C = 1

addu $0, $3, $4: 0xFFFF\_FFFF + 0x7FFF\_FFFF = 0x7FFF\_FFFE, Z = 0, C = 1

addu $0, $1, $5: 0x8000\_0000 + 0x0000\_0001 = 0x8000\_0001

# Conclusion

This homework is to make students have more understand about the basic principles of MIPS.

By employing distinct modules to implement various functionalities, akin to the top-down design method prevalent in programming. Breaking down the complex architecture into smaller, manageable structures enable students to tackle problems step by step.

Through the process, students not only gain insight into MIPS but also develop essential skills in modular design and problem-solving.