REPORT LAB02

PB19010450 和泳毅

Requirements

Write a program in LC-3 assembly language that is used to calculate the greatest common divisor **(GCD**) of two positive numbers.

- Two positive 16-bit signed integers will be given in R0 and R1 register (You can fill the values in the code or modify registers manually in LC3 simulator)；
- The output value should be put in R0；
- Your program should start at memory location **x3000** and end with **HALT**.

Example

	Before Execution	After Execution
R0	**x0009**	**x0003**
R1	**x000C**	Any value

Design

Calculate the GCD of two positive numbers with the **Euclidean Algorithm**：

1. The lemma：$If \quad a = b*q+r \quad (0 \le r < b), then \quad GCD(a,b) = GCD(b,r)$.

   Prove:

   $$GCD(a,b)|b => GCD(a,b)|(a-b*q) => GCD(a,b)|r => GCD(a,b)|GCD(b,r)$$

   $$GCD(b,r)|b => GCD(b,r)|(b*q+r) => GCD(b,r)|a => GCD(b,r)|GCD(a,b)$$

   $$So, GCD(a,b) = GCD(b,r) \quad \#$$

2. Euclidean Algorithm：　$a, b \in Z^+, a > b$

   $$a = b*q_0 + r_0, 0 \le r_0 < b$$

   $$b = r_0 * q_1 + r_1, 0 \le r_1 < r_0$$

   $$\dots\dots$$

   $$r_{i+1} = r_{i+2} * q_{i+3} + 0$$

   $$=> GCD(a,b) = GCD(b,r_0) = \dots = GCD(r_{i+1}, r_{i+2}) = r_{i+2}$$

3. Detailed design:

- Compare the size of the two number in R0 and R1,and set R0 to the larger one；
- The remainder operation is achieved by subtraction, where R0 is continuously subtracted from R1, and each time the result is stored in R0 for updating until the result is negative. And the remainder is obtained by adding this negative number to R1;
- Determine whether the remainder is 0. If it isn't zero, swap R0 and R1 so that the value of R0 is larger, then repeat the second step ; if it is zero, result is the value of R1;
- Swap R0 and R1 so that the result is stored in R0.

Code Writing

1. Instructions to be used

ADD	DR,SR1,imm5	DR=SR1+SEXT(imm5)
NOT	DR,SR	DR=NOT(SR)
BR	LABEL	PC=LABEL
BRz	LABEL	IF(z AND Z) PC=LABEL
BRnz	LABEL	IF((z AND Z)OR(n AND N)) PC=LABEL
JSR	LABEL	R7=PC+1,PC=LABEL
RET		PC=R7
HALT		HALT THE PROGRAM

2. Start at memory location **x3000**

```
START    .ORIG    X3000; Start at x3000
```

3. Compare the size and set R0 to the larger value

```
        NOT     R2,R0
        ADD     R2,R2,#1;Negative number
        ADD     R2,R2,R1;Compare the value in the two registers
        BRnz    GCD
        ;If the value in R0 is larger, the remainder operation is performed.
        JSR     SWAP
        ;If the value in R0 is smaller, the swap operation is performed.
```

4. Euclidean Algorithm

```
GCD     NOT     R2,R1
        ADD     R2,R2,#1;Negative number
REM     ADD     R0,R0,R2
        BRzp    REM;The remainder operation is implemented by subtraction.
        ADD     R0,R0,R1
        BRz     GCDEND;Ends if the remainder is 0
        JSR     SWAP
        BR      GCD
GCDEND  JSR     SWAP;Store the result in R0
        HALT
```

5. Swap function

```
SWAP    ADD     R2,R0,#0
        ADD     R0,R1,#0
        ADD     R1,R2,#0
        RET
```

6. Register Usage

R0	初始值，余数，结果
R1	初始值，余数
R2	交换中间值

Result Test

1. R1<R0， $GCD(9,12)=3$

执行前

R0	x0009	9
R1	x000C	12
R2	x0000	0

执行后

R0	x0003	3
R1	x0000	0
R2	x0000	0

2. R1>R0， $GCD(12,9)=3$

执行前

R0	x000C	12
R1	x0009	9
R2	x0000	0

执行后

R0	x0003	3
R1	x0000	0
R2	x0000	0

3. GCD is one of the initial values， $GCD(15,5)=5$

执行前

R0	x000F	15
R1	x0005	5
R2	x0000	0

执行后

R0	x0005	5
R1	x0000	0
R2	x0000	0

4. Coprime integers， $GCD(1,128)=1$

执行前

R0	x0001	1
R1	x0080	128
R2	x0000	0

执行后

R0	x0001	1
R1	x0000	0
R2	x0000	0

5. Increase the data， $GCD(4438,1218)=14$

执行前

R0	x1156	4438
R1	x04C2	1218
R2	x0000	0

执行后

R0	x000E	14
R1	x0000	0
R2	x0000	0

6. Another test found that if the initial value has a zero or negative number, the program will get stuck in a loop.

Thinking

1. HALT operation will make changes to R0 and R1, so test with breakpoints before HALT;
2. Functions that are used repeatedly can be written as sub-code for invocation;
3. Addition can achieve subtraction, multiplication, division, and even taking remainders.

Summary

Writing LC-3 programs in assembly language is much simpler than machine language. The functions of assembly language and the objects to be manipulated are clear at a glance. The use of LABLE also makes the program to be versatile, allowing jumps to be made at any location without having to calculate addresses when writing code.

Rewriting in RISC-V Assembly Language

The idea remains the same, using the Euclidean Algorithm. Use **Jupiter**, an open source and education-oriented RISC-V assembler and runtime simulator.

1. Instructions to be used
   - **MV RD RS**

     i.e.**ADD RD RS1 0**

     Achieve copy function by addition operation, x[RD] = x[RS1] + 0
   - **REM RD,RS1,RS2**

     Modulo operation, x[RD] = x[RS1] mod x[RS2]
   - **BGQ RS1,RS2,LABLE**

     Branch, if (RS1 = RS2) pc = LABEL
   - **J LABEL**

     Jump to the LABEL
2. Core Code

```
GCD:    REM     x1,x1,x2
        BEQ     x1,x0,NEXT
        MV      x3,x1
        MV      x1,x2
        MV      x2,x3
        J       GCD
NEXT:   MV      x3,x1
        MV      x1,x2
        MV      x2,x3
```

3. Register usage

ra(x1)	初始值，余数，结果
sp(x2)	初始值，余数
gp(x3)	交换中间值

4. Result test

Appendix

Complete code:

LC-3:

```
START   .ORIG   X3000
        NOT     R2,R0
        ADD     R2,R2,#1;取反加一变负数
        ADD     R2,R2,R1;比大小
        BRnz    GCD;如果R0较大,进行取余
        JSR     SWAP;如果R0较小,进行交换
GCD     NOT     R2,R1;
        ADD     R2,R2,#1;取反加一变负数
REM     ADD     R0,R0,R2;
        BRzp    REM;用减法取余
        ADD     R0,R0,R1;
        BRz     GCDEND;如果余数为0则结束
        JSR     SWAP;如果余数不为0，继续辗转相除
```

```
        BR      GCD;
GCDEND  JSR     SWAP;让R0存结果
        HALT
;
SWAP    ADD     R2,R0,#0;交换
        ADD     R0,R1,#0
        ADD     R1,R2,#0
        RET
;
        .END
```

RISC-V：

```
.globl main
.test
main:
    BGE     x1,x2,GCD
    MV      x3,x1
    MV      x1,x2
    MV      x2,x3
GCD:
    REM     x1,x1,x2
    BEQ     x1,x0,NEXT
    MV      x3,x1
    MV      x1,x2
    MV      x2,x3
    J       GCD
NEXT:
    MV      x3,x1
    MV      x1,x2
    MV      x2,x3
    LI      a0, 10
    ECALL
```