计算机组成原理实验报告

MIPS 五级流水处理器开发

班级：1618001　　专业：计算机软件　　日期：2020-6-1

成员：161810129 董世晨　161810104 翁静

161810130 崔明暄　161810101 孙茜茹

一、 功能设计

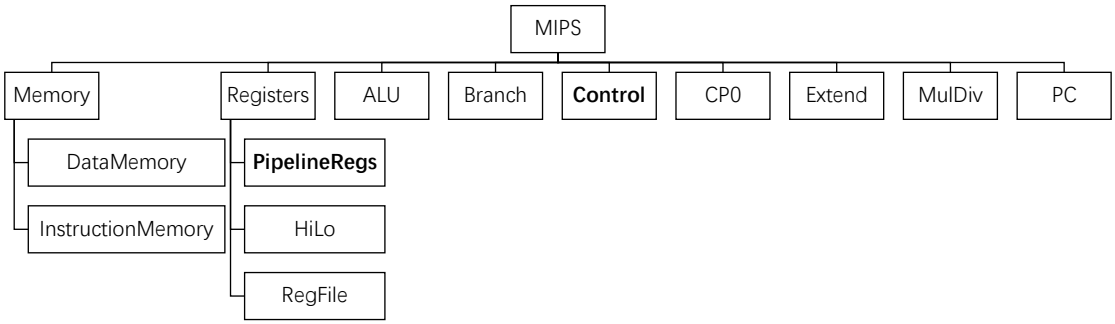
1. 完成以下 57 条 MIPS 指令
   1) 算术运算指令:add addi addu addiu sub subu slt slti sltu sltiu div divu mult multu
   2) 逻辑运算指令：and andi lui nor or ori xor xori
   3) 移位指令：sll sllv sra srav srl srlv
   4) 分支跳转指令：beq bne bgez bgtz bltz bltzal bgezal j jal jr jalr
   5) 数据移动指令：mfhi mflo mthi mtlo
   6) 自陷指令：break syscall
   7) 访存指令：lb lbu lh lhu lw sb sh sw
   8) 特权指令：eret mfc0 mtc0

2. 支持以下 7 种异常：
   1) 2 个软件中断
   2) 地址错例外——取指
   3) 地址错例外——数据访问
   4) 保留指令例外
   5) 整形溢出例外
   6) break 指令例外
   7) syscall 指令例外

3. 处理器为五级流水设计，具有以下特点：
   1) 实现流水线握手机制，支持任意阶段任意周期的 stall
   2) 在 ID 阶段处理所有条件/无条件跳转，实现包括 load-store 在内的所有数据转发，只有 load-use hazard 需要 stall
   3) 使用 Python 脚本根据 Excel 真值表自动生成流水寄存器模块和 Control 模块代码
   4) 使用 git 进行版本控制，记录设计的各个阶段
   5) 除了自动生成的代码，所有模块均有完整的测试文件，代码覆盖率 100%

二、 设计概览

1. 模块层次结构示意图
   自动生成的模块已用粗体显示



2. 流水阶段介绍
   1) IF 阶段：PC 模块计算下一条指令地址，InstructionMemory 取出指令
   2) ID 阶段：RegFile 模块读取寄存器，Control 模块分析指令得到控制信号，Extend 模块对立即数进行扩展，Branch 模块进行分支前移判断
   3) EX 阶段：ALU 模块和 MulDiv 模块进行计算
   4) MEM 阶段：DataMemory 读取或写入数据，CP0 模块和 HiLo 模块读取数据
   5) WB 阶段：RegFile 模块、CP0 模块和 HiLo 模块写入数据，CP0 模块处理异常

3. 数据转发处理方式

   为了不重不漏地解决所有的 Data hazard，我们从另一个角度看待 Data hazard：Data hazard 就是读取寄存器时寄存器的值还未写入，那么我们将所有可能发生 Data hazard 的寄存器封装一下，从产生数据的地方获取正确的数据来替换寄存器读取的值。将转发后的值作为寄存器模块输出的值保存到流水寄存器中，这样无论在哪里访问都能访问到转发后的值。

   经过谨慎的时序分析，由于我们的流水线设计使得 CP0 模块和 HiLo 模块在 MEM 阶段读取数据，所以这两个寄存器只需要内部进行从输入到输出的转发。唯一需要外部转发的是 RegFile 模块。

   我们将本应该在 WB 阶段考虑的 RegDinSrc 控制信号在 ID 阶段提前考虑，该信号指明了 RegFile 模块的输入来源，根据这个输入来源，我们将已经存在的数据进行转发，如果数据不存在，可以等到 EX 阶段再进行转发（这也就处理了 load-store hazard），或者 stall 一个周期。这样就可以考虑到所有的数据转发情况，这种设计方式具备很强的可扩展性。

   详细的代码见 MIPS.v 的 274 行~301 行，以及 379 行~392 行。

4. 异常及 Stall/Flush 处理方式

   对流水线的每一个模块，可以输出 requireStall 和 exception 信号：requireStall 信号为真表示该模块需要至少 stall 一个周期才能完成执行，exception 信号为真表示执行过程中出现了异常。

对每一个流水寄存器，接受 flush 和 stall 控制信号：flush 控制信号为真时要求流水寄存器将寄存器置零，stall 控制信号为真时要求流水寄存器不进行更新。

在发生异常时，需要保证异常发生之前的指令正确执行，因此通过流水寄存器将模块发出的异常信号带到 WB 阶段供 CP0 模块处理。CP0 模块接受每个模块的 requireStall 以及通过流水寄存器传到 WB 阶段的 exception 信号。一旦在 WB 阶段检测到异常，需要 flush 所有流水寄存器，保证异常之后的指令不进行寄存器或 DataMemory 写入。CP0 模块进行综合处理后输出每个流水寄存器的 flush 和 stall 控制信号，以此来控制流水线的执行或暂停/清空。

该设计方法也同样具备很强的可扩展性，详细的代码见 CP0.v 的 111 行~120 行。

5. 自动生成代码脚本介绍

我们首先在 Excel 中完成对 Control 模块真值表的设计，以及四个流水寄存器的每个域的名称和宽度,Python 脚本会通过 openpyxl 库读取 Excel 文件内容,自动生成 Control 模块和四个流水寄存器模块。这些自动生成的文件不需要任何手动操作，实现了真正的一键自动生成，做到了“面向 Excel 设计 CPU”。

另外，在设计顶层模块时需要实例化很多接口，同时保证代码对齐，这是比较枯燥的。这个脚本会自动将所有已实现的模块实例化，保存到剪切板中。

这个实验的设计过程高度依赖此脚本，极大地方便了后期调试阶段对流水寄存器和 Control 模块真值表的修改。此脚本共生成了正好 1000 行代码，占总行数的 59.3%。

详细的代码见 Script.py。

三、 控制信号

控制信号可以分为两类：一类是作为模块的输入，控制模块的行为，如 AluOp 等，这一类以“<模块缩写><控制信号描述>”来命名；另一类控制模块的输入来源,如 AluDin1Src 等,这一类以“<模块缩写><接口名>Src”命名。以下给出各个控制信号的描述。

流水阶段	信号名称	类型	取值	描述
IF	DelaySlot	第一类	0	该指令不处于延迟槽中
			1	该指令处于延迟槽中
ID	BraEnable	第一类	0	该指令不是条件分支指令
			1	该指令是条件分支指令
ID	BraOp	第一类	000	beq 指令，din1 == din2
			001	bne 指令，din1 != din2
			010	bgez/bgezal 指令，din1 >= 0
			011	bgtz 指令，din1 > 0
			100	blez 指令，din1 <= 0
			101	bltz/bltzal 指令，din1 < 0

ID	TakeJumpImm	第一类	0	该指令不是无条件分支指令
			1	该指令是无条件分支指令
ID	TakeJumpReg	第一类	0	该指令不是寄存器跳转指令
			1	该指令是寄存器跳转指令
ID	ExtSign	第一类	0	无符号扩展
			1	有符号扩展
EX	RegAddr3Src	第二类	00	指令的 rd 域
			01	指令的 rt 域
			10	固定值 31（jal/bgezal/bltzal）
EX	AluOp	第一类	0000	add/addi 指令（整形溢出例外）
			0001	add/addi/访存指令
			0010	sub 指令（整形溢出例外）
			0011	subu 指令
			0100	slt/slti 指令
			0101	sltu/sltiu 指令
			0110	and/andi 指令
			0111	lui 指令
			1000	nor 指令
			1001	or/ori 指令
			1010	xor/xori 指令
			1011	sllv/sll 指令
			1100	srav/sra 指令
			1101	srlv/srl 指令
			1110	不使用 ALU 模块
EX	AluDin1Src	第二类	0	RegFile 模块的第一个输出
			1	指令的 sa 域
EX	AluDin2rc	第二类	0	RegFile 模块的第二输出
			1	扩展后的立即数
EX	MdOp	第一类	00	div 指令
			01	divu 指令
			10	mult 指令
			11	multu 指令
MEM	MemWrite	第一类	0	DataMemory 不可写
			1	DataMemory 可写
MEM	MemRead	第一类	0	DataMemory 不可读
			1	DataMemory 可读
MEM	MemSize	第一类	00	读取/写入字节
			01	读取/写入半字
			10	读取/写入字
MEM	MemSign	第一类	0	读取有符号数
			1	读取无符号数
WB	RegWrite	第一类	0	RegFile 模块不可写
			1	RegFile 模块可写

WB	RegDinSrc	第二类	000	ALU 模块的输出
			001	PC + 8
			010	HiLo 模块的输出 Hi
			011	HiLo 模块的输出 Lo
			100	DataMemory 的输出
			101	CP0 模块的输出
WB	CP0Write	第一类	0	CP0 模块不可写
			1	CP0 模块可写
WB	HlWrite	第一类	00	Hi 寄存器不可写，Lo 寄存器不可写
			01	Hi 寄存器不可写，Lo 寄存器可写
			10	Hi 寄存器可写，Lo 寄存器不可写
			11	Hi 寄存器可写，Lo 寄存器可写
WB	HlDinHiSrc	第二类	0	MulDiv 模块的输出 Hi
			1	RegFile 模块的第一个输出
WB	HlDinLoSrc	第二类	0	MulDiv 模块的输出 Lo
			1	RegFile 模块的第一个输出
N/A	TakeEret	N/A	0	该指令不是 eret 指令
			1	该指令是 eret 指令
N/A	Exception	N/A	00	无异常
			01	break 例外
			10	syscall 例外
			11	未知指令例外

由于 Control 模块的真值表过于庞大，难以排版，具体内容见 Excel 文件的 Control 表。另外，可以在 Excel 文件的 PipelineRegs 表中找到四个流水寄存器中保存的内容。

四、 模块功能与接口

1. PC 模块
   a) 描述
      保存当前指令地址，并计算下一条指令地址
   b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
stall	input	1	Stall 信号
branchImmEx	input	32	符号扩展后的条件跳转指令立即数
jumpImm	input	26	无条件跳转指令立即数
jumpReg	input	32	寄存器跳转目标
epc	input	32	EPC 寄存器的值
takeException	input	1	是否检测到异常
takeEret	input	1	是否是 eret 指令
takeBranch	input	1	是否进行有条件跳转

takeJumpImm	input	1	控制信号（描述见上）
takeJumpReg	input	1	
pc	output	32	当前指令地址
pc4	output	32	下一条指令地址，即 pc+4

c) 功能

保存指令的地址

若复位信号为真，将 pc 重置为 0xBFC0_0000

若检测到异常，将 pc 设置为 0xBFC0_0380

若指令为 eret，将 pc 设置为 EPC 寄存器的值

若不需要 stall：

若需要进行有条件跳转，将 pc 设置为 pc+(branchImmEx<<2)

若需要进行无条件跳转，将 pc 设置为{pc[31:28],jumpImm,00}

若需要进行寄存器跳转，将 pc 设置为寄存器的值

否则将 pc 设置为 pc+4

2. 四个流水寄存器模块

a) 描述

IF/ID、ID/EX、EX/MEM、MEM/WB 流水寄存器（此模块由脚本自动生成）

b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
flush	input	1	Flush 信号
stall	input	1	Stall 信号
<上阶段名>_<信号名>	input	N/A	上阶段的信号
<下阶段名>_<信号名>	output	N/A	下阶段的信号

c) 功能

若复位信号或 Flush 信号为真，将寄存器所有的域都置零

若不需要 stall，将上阶段的信号保存到流水寄存器，并输出

3. InstructionMemory

a) 描述

封装外部 inst_ram 接口，提供取指功能

b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
addr	input	32	指令地址
dout	output	32	读取的指令
requireStall	output	1	请求 stall
exception	output	1	取指地址错例外
inst_ram_en	output	1	外部接口
inst_ram_wen	output	4	
inst_ram_addr	output	32	

inst_ram_wdata	output	32	
inst_ram_rdata	input	32	

c) 功能

由于 block_memory 需要一个周期进行读取，所以每隔一个周期此模块需要 stall 一次

检查地址错例外，将输入和输出映射到外部接口上

4. DataMemory

a) 描述

封装外部 data_ram 接口，提供数据读取和写入功能

b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
addr	input	32	数据地址
din	input	32	输入数据
memWrite	input	1	控制信号（描述见上）
memRead	input	1	
memSize	input	2	
memSign	input	1	
dout	output	32	读取的数据
requireStall	output	1	请求 stall
exception	output	2	00：无异常 01：读取时地址错例外 10：写入时地址错例外
data_ram_en	output	1	外部接口
data_ram_wen	output	4	
data_ram_addr	output	32	
data_ram_wdata	output	32	
data_ram_rdata	input	32	

c) 功能

由于 block_memory 需要一个周期进行读取，所以每当读取地址改变或 memRead 上升沿时，此模块需要 stall 一个周期

根据四个控制信号，读取或写入对应的数据，进行必要的符号扩展，检查地址错例外，将输入和输出映射到外部接口上

5. RegFile 模块

a) 描述

通用寄存器模块

b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
addr1	input	5	第一个读取地址

addr2	input	5	第二个读取地址
addr3	input	5	写入地址
din	input	32	输入数据
regWrite	input	1	控制信号（描述见上）
dout1	output	32	第一个输出数据
dout2	output	32	第二个输出数据

c) 功能

若复位信号为真，重置寄存器内容

若写的不是 0 号寄存器，则更新该寄存器内容

根据写地址是否和读地址相同，进行输入到输出的转发

6. HiLo 模块

a) 描述

Hi 和 Lo 寄存器

b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
dinHi	input	32	Hi 输入数据
dinLo	input	32	Lo 输入数据
hlWrite	input	2	控制信号（描述见上）
doutHi	output	32	Hi 输出数据
doutLo	output	32	Lo 输出数据

c) 功能

若复位信号为真，重置寄存器内容

若写控制信号为真，更新寄存器内容，并进行输入到输出的转发

7. Branch 模块

a) 描述

条件分支前移模块

b) 接口

名称	方向	位宽	描述
din1	input	32	第一个输入
din2	input	32	第二个输入
braEnable	input	1	控制信号（描述见上）
braOp	input	2	
takeBranch	output	1	是否进行条件跳转

c) 功能

根据 braOp 对 din1 和 din2 进行比较，再和 braEnable 相与，判断是否需要进行条件跳转

8. Extend 模块

a) 描述

有符号/无符号扩展模块

b) 接口

名称	方向	位宽	描述
din	input	16	输入
extSign	input	1	控制信号（描述见上）
dout	output	32	扩展后的输出

c) 功能

根据 extSign 对 din 进行有符号/无符号扩展，输出到 dout

9. ALU 模块

a) 描述

运算逻辑单元

b) 接口

名称	方向	位宽	描述
aluOp	input	4	控制信号（描述见上）
din1	input	32	第一个输入
din2	input	32	第二个输入
dout	output	32	输出
exception	output	1	整形溢出例外

c) 功能

根据 aluOp 对 din1 和 din2 进行对应的运算，输出到 dout

检测相应的整数溢出例外

10. MulDiv 模块

a) 描述

乘除运算单元

b) 接口

名称	方向	位宽	描述
mdOp	input	2	控制信号（描述见上）
din1	input	32	第一个输入
din2	input	32	第二个输入
doutHi	output	32	高位输出
doutLo	output	32	低位输出

c) 功能

根据 mdOp 对 din1 和 din2 进行对应的运算，高位输出到 doutHi，低位输出到 doutLo

11. Control 模块

a) 描述

控制单元（此模块由脚本自动生成）

b) 接口

名称	方向	位宽	描述
instr	input	32	指令
<控制信号名>	output	N/A	控制信号（描述见上）

c) 功能

根据指令和真值表输出相应的控制信号

12. CP0 模块

a) 描述

0 号协处理器，负责处理异常、例外和 Stall，保存 BadVAddr、Status、Cause、EPC 等系统控制寄存器

b) 接口

名称	方向	位宽	描述
clk	input	1	时钟信号
rst	input	1	复位信号
addrR	input	5	系统寄存器读 Address
selR	input	6	系统寄存器读 Select（恒为 0）
addrW	input	5	系统寄存器写 Address
selW	input	6	系统寄存器写 Select（恒为 0）
din	input	32	输入数据
cp0Write	input	1	控制信号（描述见上）
dout	output	32	输出数据
epc	output	32	EPC 寄存器内容
takeEret	input	1	控制信号（描述见上）
imExcept	input	1	InstructionMemory 异常
ctrlExcept	input	2	Control 模块异常（描述见上）
aluExcept	input	1	ALU 模块异常
dmExcept	input	1	DataMemory 异常
memWrite	input	1	控制信号（描述见上）
memRead	input	1	
delaySlot	input	1	
WB_pc4	input	32	WB 阶段的 pc4
WB_aluDout	input	32	WB 阶段的 ALU 模块输出
takeException	output	1	是否检测到异常
imRequireStall	input	1	InstructionMemory 请求 Stall
dmRequireStall	input	1	DataMemory 请求 Stall
fwdRequireStall	input	1	数据转发请求 Stall
PC_stall	output	1	PC 模块 stall 信号
IF_ID_stall	output	1	各流水寄存器 stall 信号
ID_EX_stall	output	1	
EX_MEM_stall	output	1	
MEM_WB_stall	output	1	
IF_ID_flush	output	1	各流水寄存器 flush 信号
ID_EX_flush	output	1	
EX_MEM_flush	output		
MEM_WB_flush	output	1	

c) 功能

若复位信号位真，重置系统控制寄存器内容

若写信号为真，更新寄存器内容，且进行输入到输出的转发

输出 EPC 寄存器内容

检测软件中断和各模块异常，控制流水寄存器的 Stall 和 Flush 信号

五、 测试代码

1. 模块测试代码

在实现各模块时，我们对每个模块进行了详细的测试，保证每个模块的行为符合预期，为之后整体调试减轻负担。我们共编写了 795 行模块测试代码，考虑到了所有可能发生的情况，测试代码覆盖率达到了 100%。

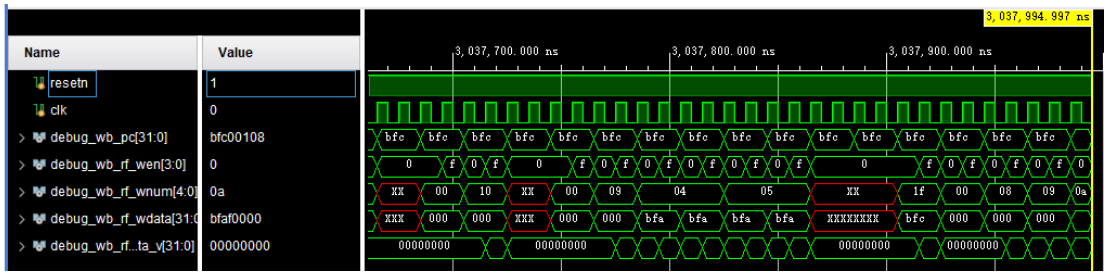
详细的模块测试代码见 ModuleTest 文件夹。

2. Trace 测试

此 CPU 通过了 Trace 的所有 89 个测试点，共用时 3037995ns。

```
——[2863345 ns] Number 8'd83 Functional Test Point PASS!!!
       [2872000 ns] Test is running, debug_wb_pc = 0xbfc004b0
       [2882000 ns] Test is running, debug_wb_pc = 0xbfc004dc
       [2892000 ns] Test is running, debug_wb_pc = 0xbfc00698
——[2892265 ns] Number 8'd84 Functional Test Point PASS!!!
       [2902000 ns] Test is running, debug_wb_pc = 0xbfc00498
       [2912000 ns] Test is running, debug_wb_pc = 0xbfc004bc
——[2921205 ns] Number 8'd85 Functional Test Point PASS!!!
       [2922000 ns] Test is running, debug_wb_pc = 0xbfc28820
       [2932000 ns] Test is running, debug_wb_pc = 0xbfc00478
       [2942000 ns] Test is running, debug_wb_pc = 0xbfc00394
——[2950125 ns] Number 8'd86 Functional Test Point PASS!!!
       [2952000 ns] Test is running, debug_wb_pc = 0xbfc46a4c
       [2962000 ns] Test is running, debug_wb_pc = 0xbfc0039c
       [2972000 ns] Test is running, debug_wb_pc = 0xbfc46c54
——[2979045 ns] Number 8'd87 Functional Test Point PASS!!!
       [2982000 ns] Test is running, debug_wb_pc = 0xbfc27db0
       [2992000 ns] Test is running, debug_wb_pc = 0x0000000c
       [3002000 ns] Test is running, debug_wb_pc = 0xbfc00694
——[3007965 ns] Number 8'd88 Functional Test Point PASS!!!
       [3012000 ns] Test is running, debug_wb_pc = 0xbfc006b8
       [3022000 ns] Test is running, debug_wb_pc = 0xbfc17488
       [3032000 ns] Test is running, debug_wb_pc = 0xbfc00528
——[3036885 ns] Number 8'd89 Functional Test Point PASS!!!
================================================================

Test end!
——PASS!!!
$finish called at time : 3037995 ns : File "F:/Trace/soc_sram_func/testbench/mycpu_tb.v" Line 261
run: Time (s): cpu = 00:00:27 ; elapsed = 00:00:21 . Memory (MB): peak = 1011.672 ; gain = 0.000
```



六、 实验时间安排

我们使用了 git 来进行版本控制和记录每个阶段的进度。可以在目录下使用 git log 命令查看详细进展。

时间	工作
5-24 2:44:40	开始设计 CPU
5-24 14:41:23	设计并测试 Extend 模块
5-24 14:49:33	设计并测试 RegFile 模块
5-24 19:43:59	设计并测试 ALU 模块
5-24 21:14:24	设计并测试 Branch 模块
5-24 23:27:34	设计并测试 PC 模块
5-25 1:46:33	添加异常处理
5-25 21:59:35	设计并测试 MulDiv 模块
5-26 22:17:33	设计真值表
5-27 12:48:31	设计并测试 HiLo 模块
5-27 23:44:08	调整文件接口，设计并测试 IF_ID 模块
5-28 5:37:37	修改 DataMemory 和 InstructionMemory
5-28 17:12:47	自动生成流水寄存器模块
5-28 19:12:45	自动生成 Control 模块
5-29 20:15:23	给 Memory 添加请求 Stall 信号
5-29 21:59:04	设计并测试 CP0 模块的基础功能
5-30 1:36:57	完成 MIPS 顶层模块，但不包含 Hazard 和异常
5-30 3:16:33	设计 Stall 和 Flush
5-30 8:15:23	设计数据转发
5-30 11:18:24	调试
5-31 12:52:40	通过 Trace 的所有测试点
6-5 15:43:25	添加 Load-store 转发，优化执行效率

七、 心得体会

大局观和注重细节是设计 CPU 的两个必不可少因素。这次试验没有采用上一次实验的代码，完全从头开始设计，这样子有助于在一开始的设计阶段考虑到所有的情况，避免了之后反复修改代码，提高了代码的可读性和编写效率。另一方面，我们在设计控制信号、数据转发时都采用了一种巧妙而优雅的方式做到不重不漏地设计：将控制信号分为两类、将数据转发视作对寄存器输出的一次封装。清晰的逻辑帮我们尽可能地缩短了调试的时间。

在进行 Trace 测试之前，我们对每个模块单独进行了测试，测试代码达到了 CPU 代码的一半。这使得我们在上 Trace 之后没有遇到任何模块执行出错的情况。

我们在 Excel 中设计完真值表后，编写了一个 Python 脚本自动生成模块代码，这帮助我们在最后调试时只需要修改 Excel，就可以一键生成对应的代码，提高了调试效率。

我们还使用了 git 来帮助我们进行版本控制和记录进度。有时候常常会遇到修改了代码导致莫名其妙的错误，就可以使用 git 命令回滚到之前的版本。