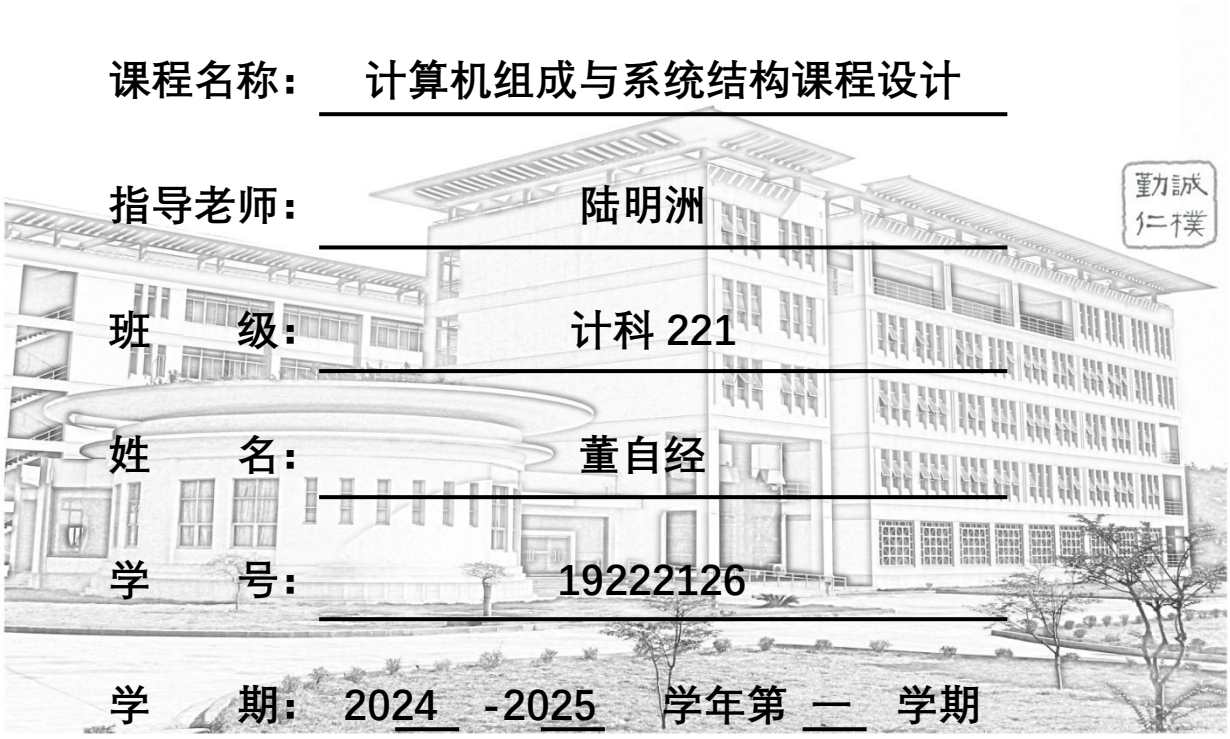
南京農業大學

NANJING AGRICULTURAL UNIVERSITY

实验报告册



课程名称：计算机组成与系统结构课程设计

指导老师：陆明洲

班　　级：计科 221

姓　　名：董自经

学　　号：19222126

学　　期：2024 -2025 学年第 一 学期

目录

计算机组成原理与系统结构课程设计

计科 221 董自经

一、简介及设计目标

1.1 简介

计算机组成原理是研究计算机系统的基本结构、工作原理和设计方法的一门学科。它涵盖了计算机硬件的各个组成部分，包括中央处理器（CPU）、存储器、输入输出系统和总线等，以及这些组件之间的相互关系和数据传输机制。计算机组成原理的重要性在于它为设计和优化计算机系统提供了理论基础和技术手段，有助于提高计算机的性能、可靠性和效率。通过理解计算机的内部工作原理，开发者可以更好地进行硬件设计、系统编程和性能调优，从而推动计算机技术的发展和应用的创新。

1.2 设计目标

自主完成基本的计算机硬件设计工作，并深入理解计算机系统的工作原理与设计方法。学习和掌握当代数字系统的基本设计方法，能够独立进行数字电路的设计、分析和测试。

此外，学生将了解计算机整机及各部件的基本工作原理，包括海明校验码（Hamming Code）、逻辑运算器（ALU）、存储器（RAM）中央处理器（CPU），并理解这些部件在系统中的作用及工作机制。通过观察软件在硬件平台上的执行过程，学生能够将软件活动与硬件特性联系起来，提升对系统整体的理解。

二、海明校验模块

海明校验码（Hamming Code）由理查德·海明（Richard Hamming）于 1950 年提出，是一种用于检测和纠正数据传输错误的编码方法。海明码通过在数据中加入冗余校验位，并将数据分为若干组，对每一组进行就检测，使得接收端可以检测并纠正单比特错误，从而提高数据传输的可靠性。

2.1 设计内容

在本次课程设计中要实现对 16 位数据的海明编码和解码并判断出错位置以及出错位数。

海明校验中最为重要的部分就是检验码的生成，这其实借鉴了奇偶校验码，将相应的数据位进行异或以得到校验位的数值，并组装到最终数据的相应位置。再设计过程中，最为困难的部分是码距的确定，以及码距值的相关作用，最终选择了袁春风教授编写的《计算机组与系统结构》中的定义：

(1)如果码距 d 为奇数，则能发现 d-1 位错，或者能纠正(d-1)/2 位错。

(2)如果码距 d 为偶数，则能发现 d/2 位错，并能纠正(d/2-1)位错。

2.1.1 检验位的位数的确定

假定被校验数据的位数为 n,校验位为 k 位，则故障字的位数也为 k 位。k 位的故障字所能表示的状态最多是 2^k 种，每种状态可用来说明一种出错情况。对于最多只有一位错的情况，其结果可能是无错或 n 位数据中某一位出错或 k 位校验码中某一位出错。因此，共有 1+n+k 种情况。综上可知，要能对一位错的所有结果进行正确表示，则 n 和 k 必须满足下列

关系：

$$2^k \geq 1+n+k, \text{即 } 2^k-1 \geq n+k$$

2.1.2 分组方式的确定

以十六位数据为例，由 2.1 可得检验位有五位，分别占据第 1、2、4、8、16 号位置，之后数据再从低位一次填入海明码的剩余位置，排列之后以新的位置序号代表原数据序号，如 D1 在海明码中的位置位 3，则 D1 的序号为 00011，即 D1 的故障字为 00011，不难看出，如果 D1 的数据发生变化，会影响第一二位的码字所以 D1 被分到第一、二组，决定 P1、P2 的值，由此可以得到全部数据位的分组情况，如下表所示：

	序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
位值	分组	P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14	D15	D16	P6
32	第六组	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
16	第五组																√	√	√	√	√	√	√
8	第四组								√	√	√	√	√	√	√	√							
4	第三组				√	√	√	√					√	√	√	√					√	√	√
2	第二组		√	√			√	√			√	√			√	√			√	√			√
1	第一组	√		√		√		√		√		√		√		√		√		√		√	

表 1：海明编码分组

将表中的分别按行异或就可以得到相应检错位的值，具体如下：

P1 = D1⊕D2⊕D4⊕D5⊕D7⊕D9⊕D11⊕D14⊕D16

P2 = D1⊕D3⊕D4⊕D6⊕D7⊕D10⊕D11⊕D13⊕D14

P3 = D2⊕D3⊕D4⊕D8⊕D9⊕D10⊕D11⊕D15⊕D16

P4 = D5⊕D6⊕D7⊕D8⊕D9⊕D10⊕D11

P5 = D12⊕D13⊕D14⊕D15⊕D16

2.1.3 检错、纠错

用接收到的数据再次产生新的检验位 P_i'，并将其与接收到的检错位进行异或得到检错位 G_i，用并线器将得到的 5 位 G_i 合并得到数据 G 再用一个解码器就可以得到接收到的数据出错的位置，将其取反即可获得正确数据，故障字及出错位情况如下表。

故障字	正确	出错位																					
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
G5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
G4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
G3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
G2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
G1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

表 2：故障字与出错位情况

至此完成“纠一”的功能，从表 1、表 2 可以看出，每一个数据位至少参与了两个检错位的生成，所以海明码的码距 d = 3，根据公式，码距为 3 可以发现 2 位错误，或者纠正 1 位错误，要使其具有“检二”的功能，需要扩大码距位 4，引入新的纠错位 P6，让每一个数据都参与 P6 的值，就可以保证每一个数据都至少参与了三个检验位的生成，即表 1 中的第六组，即 P6 = P1⊕P2⊕P3⊕P4⊕P5⊕D1⊕D2⊕D3⊕D4⊕D5⊕D6⊕D7⊕D8⊕D9⊕D10⊕D11⊕D12⊕D13⊕D14⊕D15⊕D16

当 G 不为 0 时，表示数据没有出错或者 P6 出错，当 P6 为 1 时表示有一位错，这一位也

有可能是 P6 出错，当 P6 为 0 且 G 不为 0 时表示有两位错，相应关系如下表：

G	P6	出错情况
0	0	无错
不为 0	1	一位错
0	1	P6 错，一位错
不为 0	0	两位错

表 3：纠错码、P6 以及出错位数情况

2.2 电路图

根据上述原理，可以画出相应的解码图以及编码图：

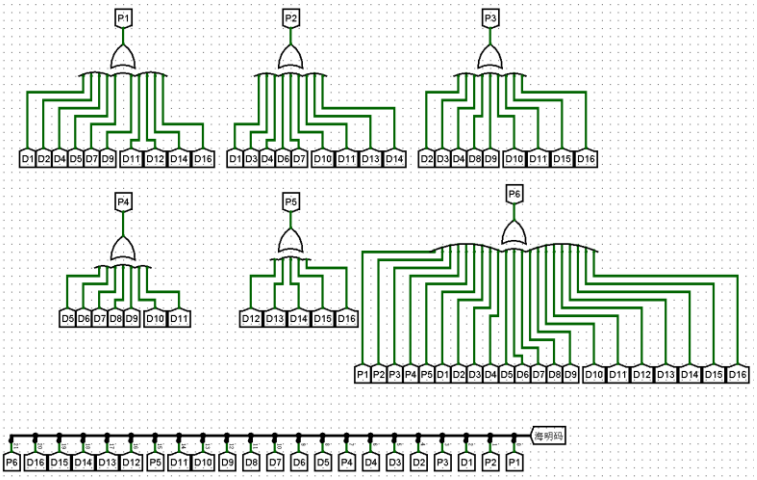


图 1：海明编码

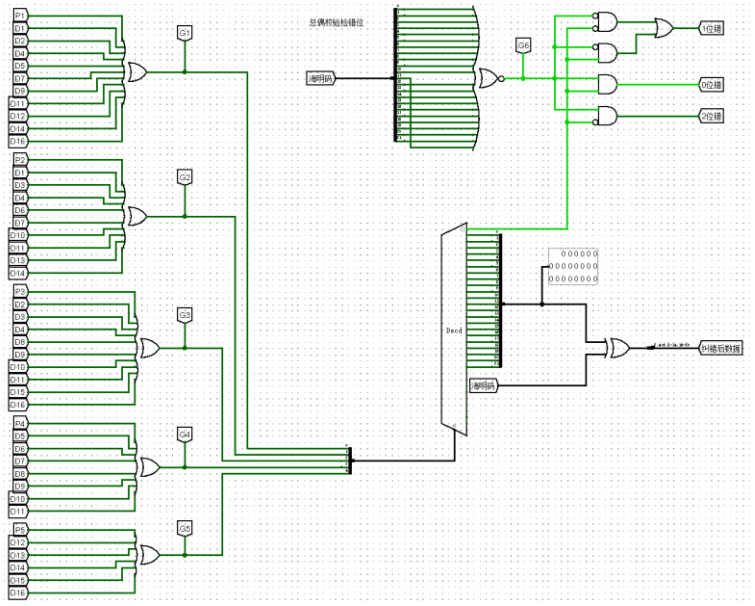


图 2：海明解码

2.3 测试分析

在给定模块的测试界面进行测试，在一开始时，解码结果（图中几位错误）全部错误，后经仔细观察线路图，发现是在对纠错码的 0 位和 P6 进行操作时有的与门引脚有错误的取反操作导致的，改正之后顺利通过该测试。

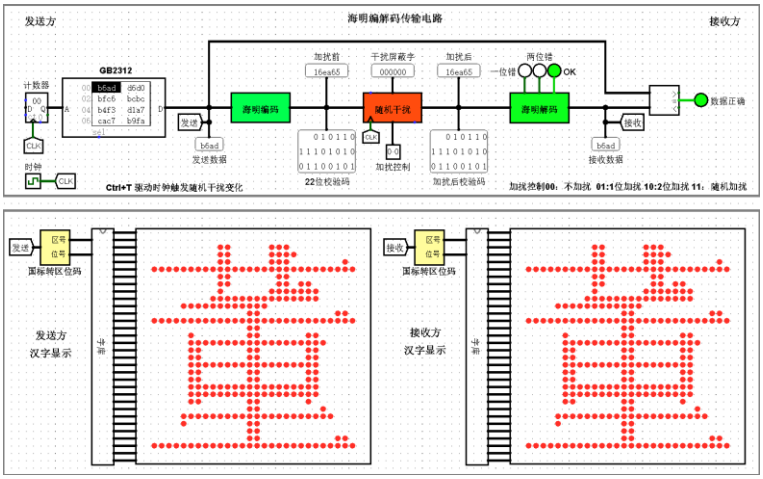


图 3：海明传输测试--无错

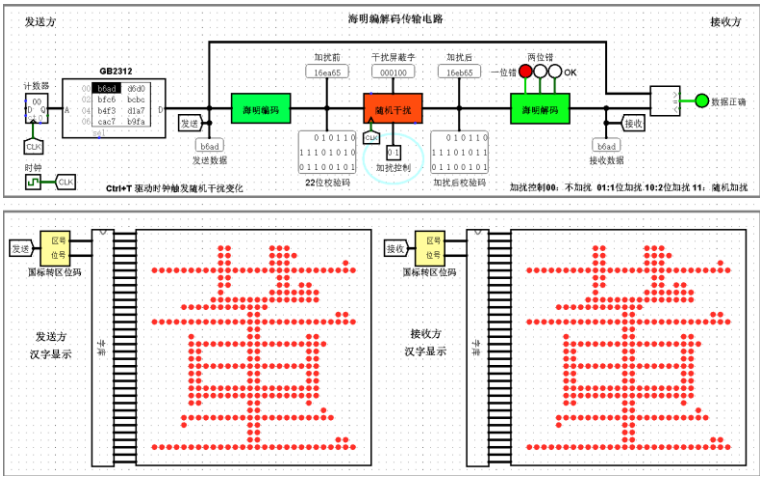


图 4：海明传输测试---一位错并纠正

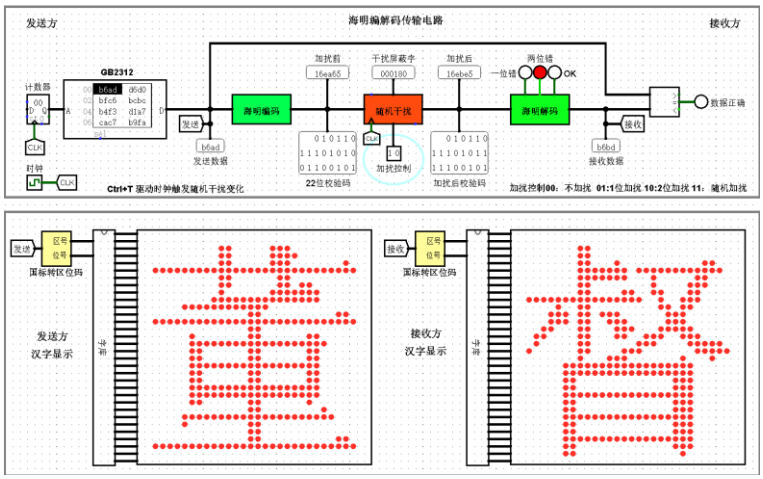


图 5：海明传输测试--两位错并检测

三、32 位 ALU

3.1 设计内容

在此次设计中，ALU 实现的功能有（按输出顺序）：逻辑左移、算数右移、逻辑右移、乘

法、除法、32 位加法、32 位减法、与、或、异或、或非、有符号数比较、无符号数比较。其中加法、减法是使用自己组装的 4 位先行进位加法器 CLA74182 经过并行、串行得到的 32 位快速加法器实现。其余算数、逻辑、位移等运算都是由系统自带的运算器实现。

3.1.1 32 位快速加法器的组装

串行进位加法原理的全加器虽然简单，且弥补了半加器无法实现多位相加的缺陷，但是每一位的运算都必须等待低位的进位信号，这导致了延迟的增加，所以在此使用不需要等待低位进位的加法—先行进位加法。

先行进位加法的原理是当机器数位数确定时，可以先计算出每一位的进位，而无需等待低位的进位信号，从而减少运算消耗的时间。在 4 位先行进位电路中，进位信号

$$C_i = G_i+P_iC_{i-1} = X_iY_i+(X_i \oplus Y_i)C_{i-1},$$

其中，$G_i = X_iY_i$ 是进位生成函数，$P_i = X_i \oplus Y_i$ 是进位传递函数，由此可以推出：

$$C_1 = P_1C_{in}+G_1$$

$$C_2 = P_2C_1+G_2 = P_2P_1C_{in}+P_2G_1+G_2$$

$$C_3 = P_3C_2+G_3 = P_3P_2P_1C_{in}+P_3P_2G_1+P_3G_2+G_3$$

$$C_4 = P_4G_3+G_4 = P_4P_3P_2P_1C_{in}+P_4P_3P_2G_1+P_4P_3G_2+P_4G_3+G_4$$

$$G^* = G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1 \quad \text{成组进位生成函数}$$

$$P^* = P_4P_3P_2P_1 \quad \text{成组进位传递函数}$$

根据公式可以画出四位先行进位电路 CLA74182：

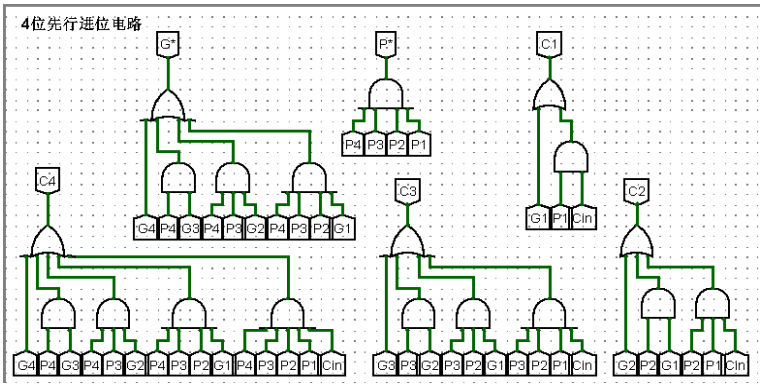


图 6：4 位先行进位电路 CLA74182

得到 CLA74182 后可以计算每一位的值 $S_i = X_i \oplus Y_i \oplus C_i = P_i \oplus C_i$，经过封装组合之后就可以画出相应的 4 位快速加法器（4 级门电路延迟）：

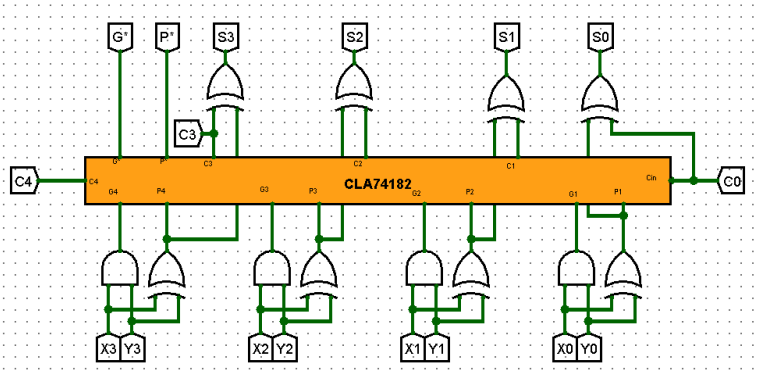


图 7：4 位快速加法器

要实现 32 位快速加法器，可以先实现 16 位快速加法器，之后将两片 16 位快速加法器组装到一起即可，而 16 位快速加法器可以由 4 位快速加法器组间并行组成，要实现成组并

行进位，就要用到上面的 G^*和 P^*，将 16 位数据中，每四位数据使用 4 位快速加法器得到的的 G^*和 P^*（三级延迟）分别作为该 4 位新的数据 G_i' 和 P_i' 输入到 CLA74182 中进行运算，得到相应的进位（两级延迟）后再传回到 4 位快速加法器中进行最终的运算得到 S_i（三级延迟），电路图如下，共八级延迟：

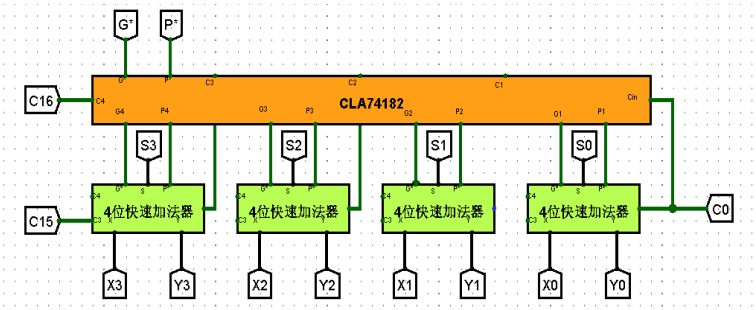


图 8：16 位快速加法器

将两片 16 位快速加法器串联，低位的进位信号作为高位的输入 C_0 就可以得到 32 位快速加法器，电路图如下：

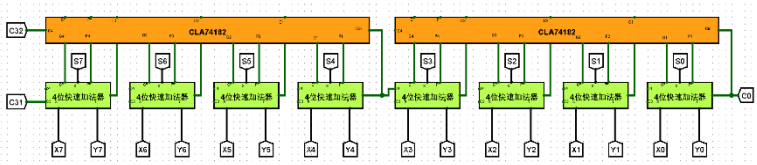


图 9：32 位快速加法器

3.1.2 ALU 的构建

ALU 一共 16 位，13 位有具体意义，后三位可以用 0 填补，每一位具体作用如下：

0-2	位移，使用相应的移位器实现（逻辑左移、算数右移、逻辑右移）
3-4	使用自带的乘法器、除法器实现
5	加法，用自制的 32 位快速加法器实现，OF 表示符号加法溢出，UOF 表示无符号加法溢出（上溢）
6	减法，用自制的 32 位快速加法器实现，Y 求补用一个补码器实现，OF 表示符号减法溢出，UOF 表示无符号减法溢出（下溢）
7-10	用系统系带的与门、非门、异或门、或非门实现
11	有符号比较器，0 扩展
12	无符号比较器，0 扩展，还输出 equal 信号
13-15	无意义，用 0 填充

表 4：ALU 相应位说明

溢出判断：对于无符号数加法，最高位进位就表示超过了当前位数所表示的最大值，即产生了溢出，对于无符号数减法，是将减法换成加法在进行计算的，具体的是将减数全部按位取反并加一之后与被减数进行相加，如果最高位进位为 1，表示没有溢出。以 8 位机器数为例，减数的补码是 2^8-减数，被减数加减数相当于 2^8+被减数-减数，如果被减数大于减数，则结果会超过 256，进位为 1，不会产生溢出，进位为 0 表示结果小于 256，进位为 0，表示被减数小于减数，产生溢出。对于有符号数加减法的溢出判断，使用最高位进位和符号位进位是否一致即可判断，如果一致则表示没有溢出，反之表示有溢出。相应的关系如下表：

最高位进位 \ 符号位进位	1	0
1	正常结果	正溢出
0	负溢出	正常结果

表 5：最高位进位、符号位进位与溢出关系

3.2 电路图

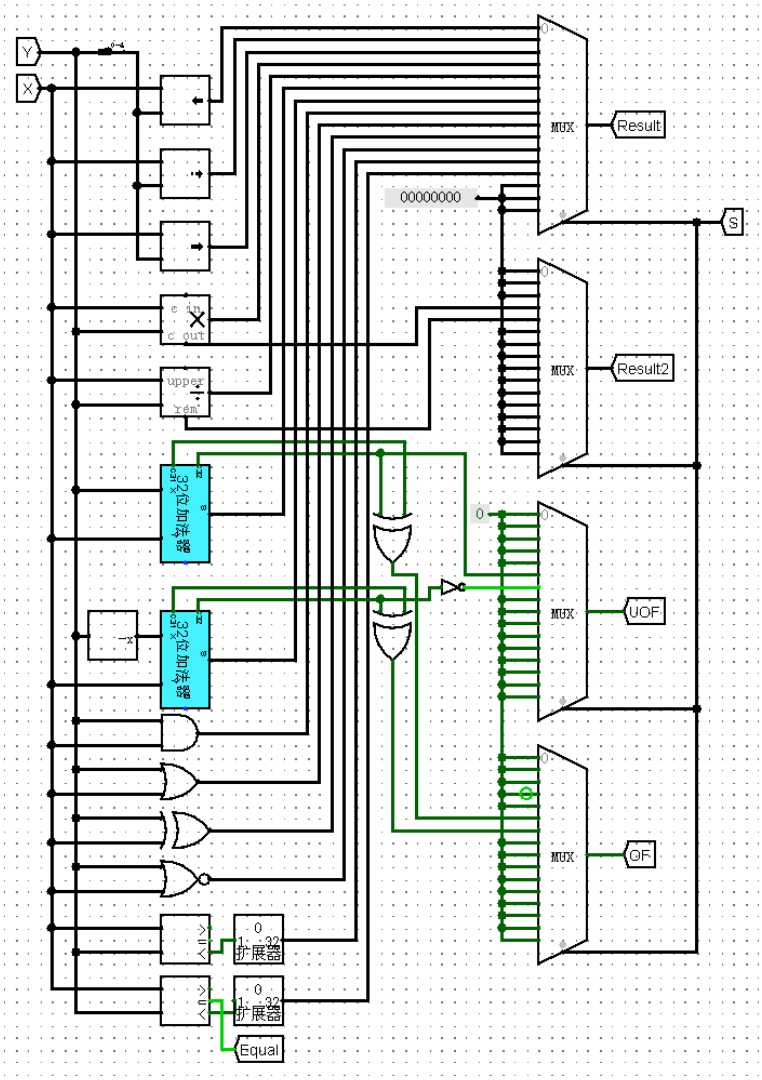


图 10：ALU 总电路图

3.3 测试分析

在初步进行调试时，在测试电路中无符号溢出标志以及减法标志报错，仔细检查相应的 32 位加法器后发现，Y 的求补不能简单使用取反进行，同时 C0 输入也不是 1，将其修改为 Y 求补使用补码器实现，C0 设置为 0，可以消除减法报错。将 C32 取反后再输入到无符号位溢出可以消除无符号位溢出报错的问题。

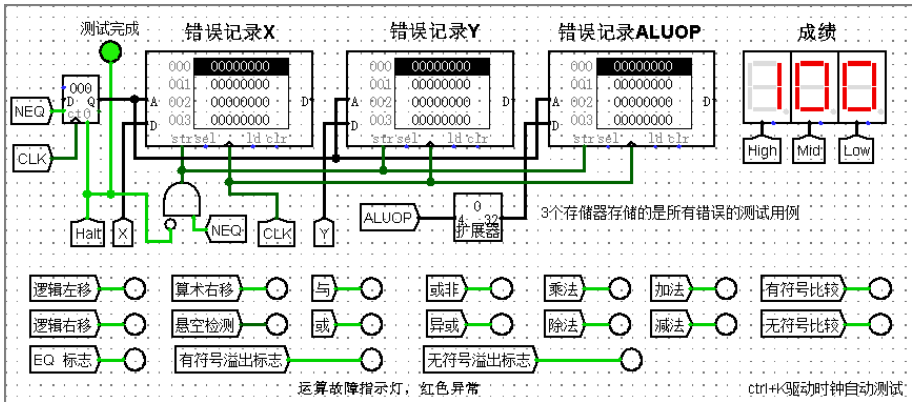


图 11：ALU 测试

四、MIPS RAM

4.1 设计内容

使用四片 1Kbx8 的 RAM 芯片在 MODE 和 12 位字节地址的控制下对 32 位输入信号进行操作，从而实现字（32 位）、半字（16 位）、字节（8 位）的输出。

4.1.1 输入位及使能信号

要使用四片 1Kbx8 的 RAM 芯片实现字（32 位）、半字（16 位）、字节（8 位）的输出，就要对齐进行相应的位扩展，由于 1Kb 的地址占用 10 位，故输入的字节地址的高十位用于片内寻址，而低两位（la）用于片间寻址。

在 Mode = 00 的情况下，需要实现 32 位的输出，此时字节地址的低两位信号被忽略，在此情况下所有的 RAM 都要进行输出，输入信息分别是输入信号的 0-7 位、8-15 位、16-23 位、24-31 位。

在 Mode = 01 的情况下，需要实现 8 位的输出，此时需要使用字节地址的低两位选择一片 Ram 进行工作，无论哪一片 RAM 被选中，其输入信息都是输入信号的 0-7 位。

在 Mode = 10 的情况下，需要实现 16 位的输出，输入信号中仅低 16 位有效，高 16 位中被忽略，使用字节地址的倒数第二位也就是 la_1 作为控制信号对 RAM 芯片进行使能控制，如果 la_1=0，则选择 R0、R1 工作，如果 la_1=1，则选择 R2、R3 进行工作，同时要使用 Mode 选择输入的数据，从低到高依次是字节地址的 0-7 位、8-15 位。

将上述内容整理成表 4、表 5：

RAM 序号	32 字	16 字	8 字
R0	Mode = 00	Mode = 10, la_{0-1} = 00	Mode = 01, la_{0-1} = 00
R1	Mode = 00	Mode = 10, la_{0-1} = 00	Mode = 01, la_{0-1} = 01
R2	Mode = 00	Mode = 10, la_{0-1} = 01	Mode = 01, la_{0-1} = 10
R3	Mode = 00	Mode = 10, la_{0-1} = 01	Mode = 01, la_{0-1} = 11

表 6：使能端有效时 Mode 及 la

Mode\RAM	0	1	2	3
00	0-7	8-15	16-23	24-31
10	0-7	8-15	0-7	8-15
01	0-7	0-7	0-7	0-7

表 7：不同字长下各 RAM 的输入位

据此可以画出各芯片的输入信号以及使能电路图：

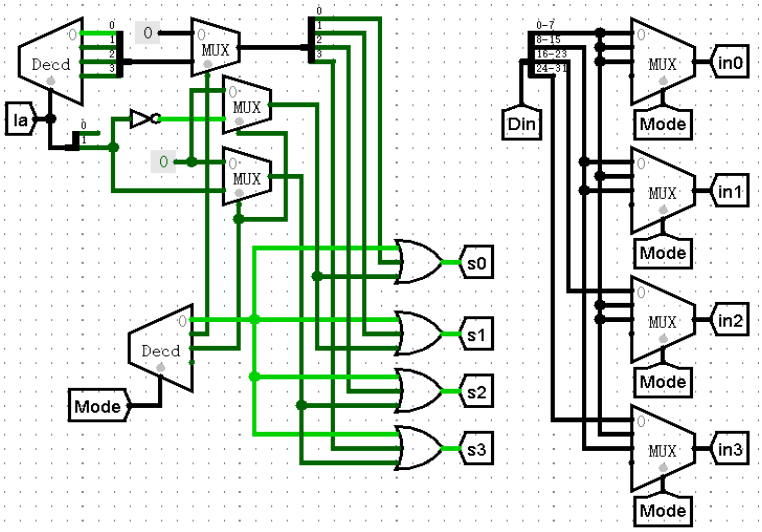


图 12：输入位及使能电路图

4.1.2 输出

输出位是要将 4 片 RAM 的输出合并到最终输出中。在 Mode = 00 时，需要将 4 片 RAM 的输出通过选择器合并成 32 位再到最终输出。在 Mode = 10 时，根据 la 选择一组 RAM 的输出作为输出的低位，经过位扩展输入到最终结果中。在 Mode = 01 时，需要根据 la 选择一片 RAM 的输出作为输出的低位，经过位扩展输入到最终结果,同时可以画出输出的电路图：

Mode \ Dout	0-7	8-15	16-23	24-31	la_{0-1}
00	R0	R1	R2	R3	xx
10	R0	R1			00
10	R2	R3			01
00	R0				00
00	R1				01
00	R2				10
00	R3				11

表 8：Dout 与 Mode、la 的关系

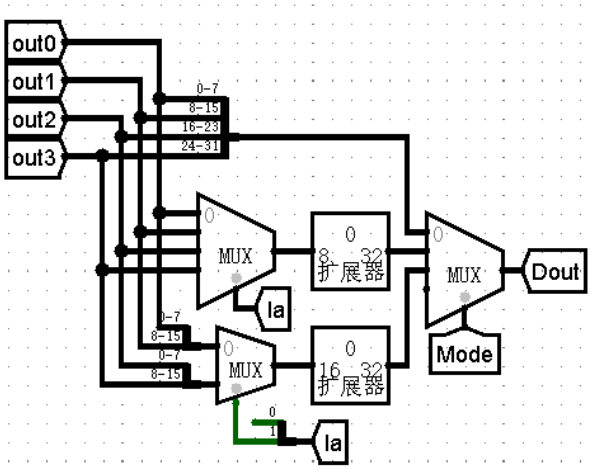


图 13：输出电路图

4.2 电路图

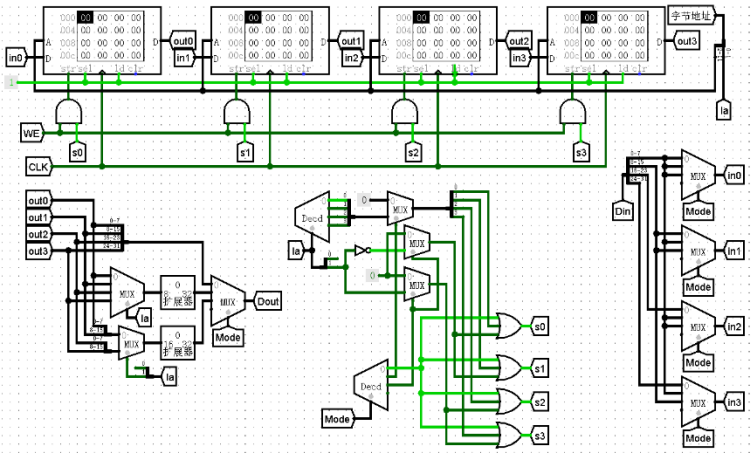


图 14：MIPS RAM 总电路图

4.3 测试分析

组装好 MIPS RAM 后进行 MIPS RAM 测试，显示 PASS，设计电路正确。

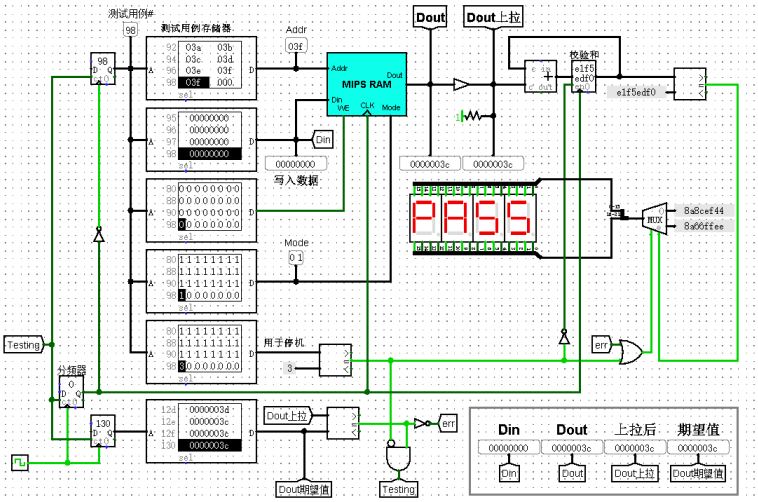


图 15：RAM 测试

五、单周期 8 指令 MIPS CPU

5.1 设计内容

本次设计中，要自行设计电路实现如下表所示的 8 条 MIPS 指令：

NO	MIPS 指令	功能描述
1	add $rd,$rs,$rt	R[$rd] ← R[$rs] + R[$rt] 溢出时产生异常且不修改 R[$rd]
2	slt $rd,$rs,$rt	R[$rd] ← R[$rs] < R[$rt] 小于置 1，有符号比较
3	addi $rt,$rs,imm	R[$rt] ← R[$rs] + $SignExt_{16b}$(imm) 溢出产生异常
4	lw $rt,imm($rs)	R[$rt] ← Mem_{4B}(R[$rs] + $SignExt_{16b}$(imm)) 加载字
5	sw $rt,imm($rs)	Mem_{4B}(R[$rs] + $SignExt_{16b}$(imm)) ← R[$rt] 保存字
6	beq $rs,$rt,imm	if(R[$rs] == R[$rt]) PC ← PC + $Signext_{18b}$({imm,00}) 相等
7	bne $rs,$rt,imm	if(R[$rs] != R[$rt]) PC ← PC + $SignExt_{18b}$({imm,00}) 不相等

8	syscall	系统调用，用于停机

表 9：8 条 MIPS 指令

8 条指令可以分为 R 型指令和 I 型指令两类指令，其指令结构如下表所示：

类型 \ 位数	6bits	5bits	5bits	5bits	5bits	6bits
R 型	OP	rs	rt	rd	shamt(位移量)	FUNC
I 型	OP	rs	rt	imm16		

表 10：指令类型及相应位含义

5.1.1 控制信号的产生

对于一次操作，先根据 PC 的值从内存中取出一条指令，经过分离器将高六位 OP、低六位 FUNC、三个操作数 rs、rt、rd 分离，将 OP 和 FUNC 送到单周期硬布线控制器中产生控制信号。本次设计的指令中更需要 9 中控制信号，对应的控制信号内容如下表所示：

	控制信号	信号说明	产生条件
1	MemToReg	寄存器写入数据来自存储器	lw 指令
2	MemWrite	写内存控制信号	sw 指令　未单独设置 MemRead 信号
3	Beq	Beq 指令译码信号	Beq 指令，用于有条件分支控制
4	Bne	Bne 指令译码信号	Bne 指令，用于有条件分支控制
5	ALU_OP	运算器操作控制符（4 位）	R 型指令根据 Func 选择
6	ALuSrc	运算器输入选择立即数还是取数	lw 指令，sw 指令，立即数运算类指令
7	RegWrite	寄存器写使能	寄存器写回信号
8	RegDst	写入寄存器编号 rt/rd 选择	R 型指令
9	Halt	Syscall 指令译码信号	根据$V0 寄存器的值，决定是停机还是输出

表 11：控制信号说明及产生条件

5.1.2 控制信号与指令的关系

根据指令的内容要在单周期硬布线控制器中设置相应的控制信号（下表中 OP、FUNC 不是此处所指的控制信号）的值，以 add 指令为例，实现 add 指令需要对寄存器进行修改，读出数据，并且 ALU 要进行工作，要对相应的 ALU_OP 赋值为 5，所以在 add 列下，ALU_OP、RegWrite、RegDst 为 1。其他指令类似（此处的比较是简单的进行比较，使用一个系统自带的比较器完成，不需要 ALU_OP 的控制）。最终关系如下表所示：

控制信号　指令	add	slt	addi	lw	sw	beq	bne	syscall
MemToReg	0	0	0	1	x	x	x	x
MemWrite	0	0	0	0	1	0	0	0
Beq	0	0	0	0	0	1	0	0
Bne	0	0	0	0	0	0	1	x
ALuSrc	0	0	1	1	1	0	0	x
RegWrite	1	1	1	1	0	0	0	0
RegDst	1	1	0	0	x	x	x	x
Halt	x	x	x	x	x	x	x	1
OP	00	00	08	23	2b	04	05	00
FUNC	20	2a	20	x	x	x	x	0c
ALU_OP	05	0b	05	0	0	0	0	0

表 12：不同指令对应的控制信号

将表中控制信号所在行为 1 的列的值相或就可以画出单周期硬布线控制器、ALU_OP 以及不同指令的电路图：

MemToReg = lw

MemWrite = sw

Beq = beq

Bne = bne

AluSrc = adii + lw + sw

RegWrite = R_TYPE + addi + lw

RegDst = R_Type

Halt = syscall

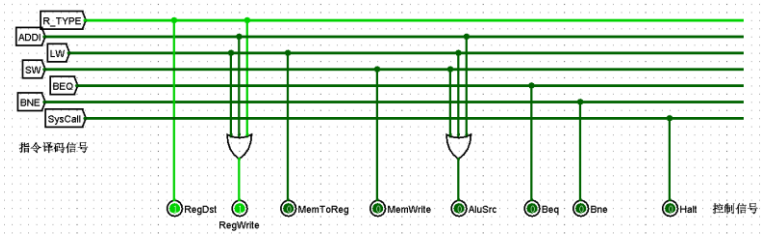


图 16：控制器输出信号逻辑

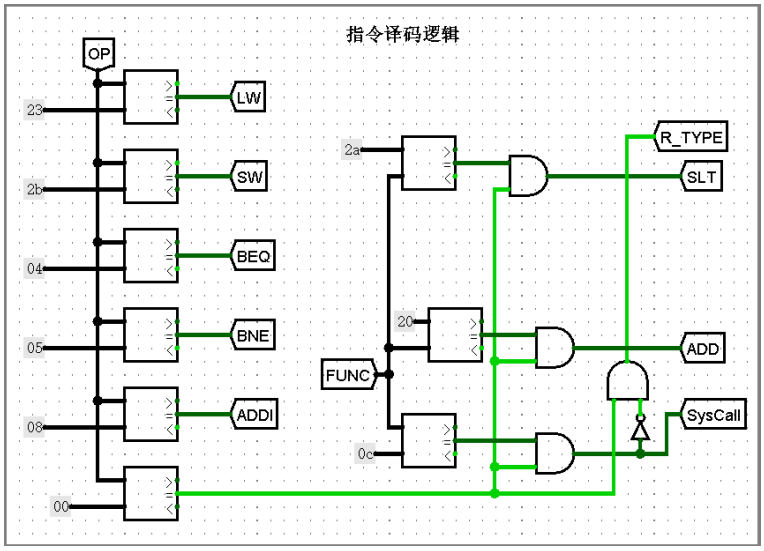


图 17：指令译码逻辑

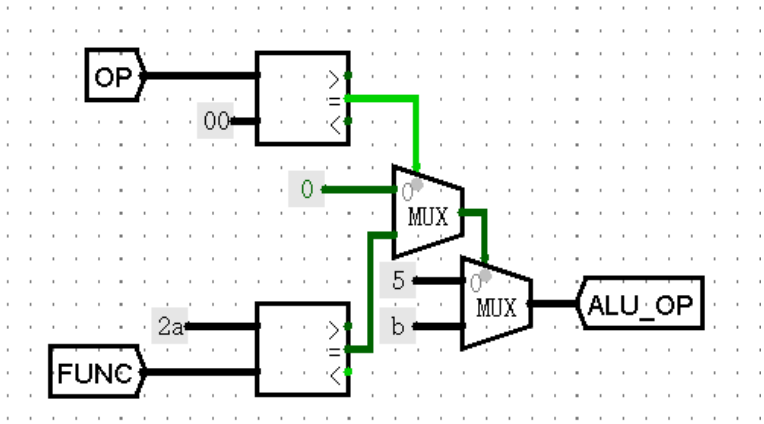


图 18：ALU 控制器逻辑

5.2 电路图

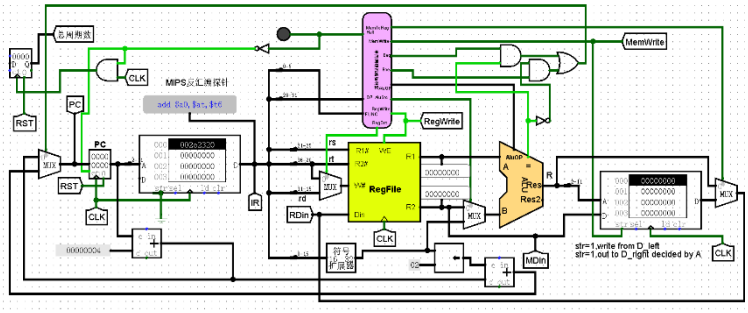


图 19：单周期 8 指令 CPU 总电路图

5.3 测试分析

在初步进行测试时无法从寄存器中读出相应的数据，原因是在对指令进行分离时，将 rs、rt 的值混淆，导致结果错误，交换后结果正确。

在进行相等判断时，一开始没有将 ALU 的运算结果输出，导致 BNE、BEQ 的结果都是错误的，将 ALU 的结果输出并与控制信号相与得到正确结果。

如图，当指令为 1000ffff 时，高六位 OP 得到指令为 beq，数据来源是寄存器中 rs 位置的数据和 rt 位置的数据，执行结束后，如果两处数据相等，则将 PC 指针的值加上 16 位立即数扩展后的数字。

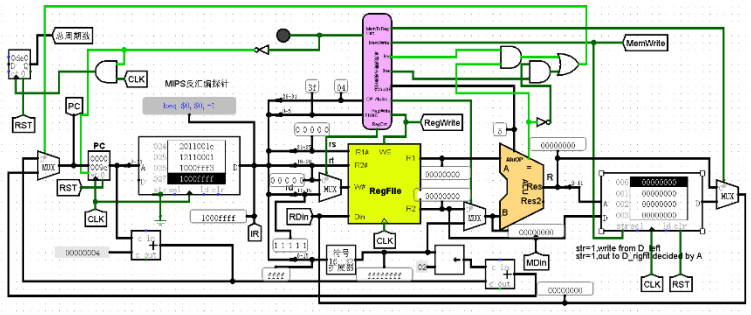


图 20：测试例子

```
020  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
030  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
040  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
050  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
070  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
080  00000006 00000005 00000004 00000003  00000002 00000001 00000000 ffffffff  000(
090  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
0a0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
0b0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
0c0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
0d0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
0e0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
0f0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
100  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  000(
```

图 21：测试后数据存储器

六、心得体会

此次课程设计是计算机组成原理与系统结构课程与数字电路两门课程相结合的一次实践活动，加深了我对计算机系统部分组件的理解，对计算机系统的层次结构有了新的认识，从 CPU 到内存再到外设，每一个组件都是不可或缺的一部分。时间过程让我体验到了设计计

算机 CPU 的复杂性，尤其是在选择合适的硬件框架方面，细节决策显得尤为重要。

在实践过程中，我遇到了很多问题，最大的问题是部分逻辑不清导致逻辑门的错误使用，进而得到了错误的实验结果，只有对其中的逻辑关系非常清晰，才有可能设计出有效的电路。其次就是对于门电路延迟的考虑，虽然在此次课程设计中，门电路的延迟并不是重点关注的问题，也没有对电路造成太大的影响，但在实际设计中，门电路延迟是不容忽略的问题，其直接影响了电路的效率以及正确性。

此外，在这次课程设计过程中，没有考虑到的问题是门电路的延迟，因为都是软件实现的硬件，所以软件会将部分门电路的延迟自动计算考虑在内，而无需用户去进行计算，但在实际应用中，门电路延迟是一个非常重要的问题，甚至可能会影响到整个系统的使用。

总结来说，此次课程设计不仅让我提高了对计算机组成与系统架构的理解，更让我认识到实践的重要性。未来，我希望能将这段经验运用到更复杂的项目中，继续探索计算机科学。