数电课设报告

刘正浩 2019270103005，李仁轩 2019270103011，唐晨烨 2019270103003

2021 年 7 月 11 日

Contents

1 概述

本次数电课程设计的要求是：利用 HDL 设计一个至少支持 8 条基本 MIPS 指令（add、sub、and、or、slt、lw、sw、beq）的 MCU，并利用 Vivado® 工具和 Digilent Basys 3 Artix-7 FPGA Board 进行 MCU 的上板验证，验证内容为 8 条基本指令，以及排序算法、DCT 算法的二选一实现。

2 背景及意义

在这学期数电课的学习过程中，可以说我们了解了从零开始构建一个完整的 CPU 的过程。课程的内容包括从晶体管到门电路，再到组合电路、时序电路、逻辑块、状态机、FPGA 和 PLC，以及不同的指令集、处理器架构、I/O 设备和存储器等一系列知识。有了这些知识，我们就可以以它们为基础来尝试制作一些基本的逻辑电路，最后可以实现一个简单的 MCU。在设计 MCU 的过程中也可以检验我们平时的学习情况以及对知识的掌握程度。

3 目标及完成情况

3.1 目标

1. 完成 MCU 中的关键部分之一——ALU 的设计与实现。
2. 选定要实现的 MCU 的类型，设计结构。
3. 完成指令集与机器码的设计与互相对应。
4. 完成 MCU 的设计。

3.2 完成情况

3.2.1 ALU 的设计与实现

根据 MCU 完成必要指令的需要，ALU 至少需要支持两个 32 位数加减、逻辑与或和判断大小的功能。具体的功能和控制信号列表如表1。

F(2:0)	Function
000	A AND B
001	A OR B
010	A + B
011	reserved
100	A AND B’
101	A OR B’
110	A - B
111	SLT（比较大小）

表 1: ALU 功能及对应控制信号

ALU 的结构如图1。经过测试，ALU 的所有功能工作正常。

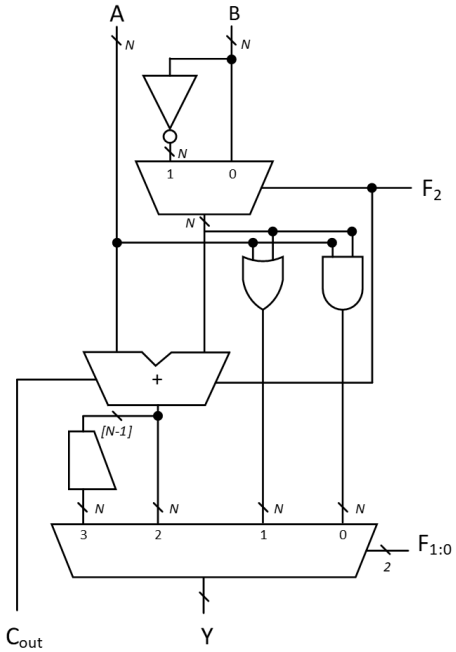

图 1: ALU 结构（图中 N=32）

3.2.2 指令集、机器码与 MCU 的设计与实现

由于时间比较紧迫，我们选择了比较简单的单周期 MCU 制作。具体的结构参考了《数字设计和计算机体系结构》书中所提到的单周期 MIPS 处理器结构。经过测验，我们的处理器实现了八条基本指令，并且可以运行排序算法。具体的设计请见详细设计报告。

4 关键创新点及效果

4.1 关于 ALU 的 SLT 功能的实现

我们规定，当 ALU 的控制信号为“111”时，ALU 进行 SLT 操作，即比较两个输入的数的大小，如果 A<B 则输出的结果 Y 为”0...01”, 否则输出的 32 位全为 0。一种常规的方法是利用 A-B 的值与 0 相比较，如果 A-B>0 则 A>B。但我们采用了不同于这种方法的另一种方法。我们分别考察最高位的三个输入，分别是 A(31)，B(31) 和 SUM(31)（SUM 为 A-B 的值）。构造这三个变量作为输入，Y 的最低位作为输出的卡诺图，进而得到一个简单的组合电路。画出的卡诺图如表2。

AB and SUM	00	01	11	10
0	0	0	0	1
1	1	0	1	1

表 2: 卡诺图

得到的化简后的表达式为

$$Y_0 = A_{31} \cdot B'_{31} + A_{31} \cdot SUM_{31} + B'_{31} \cdot SUM_{31} \tag{1}$$

根据化简后的式子，就可以搭建出相应的逻辑电路。

4.2 关于汇编程序

对于运算结果的保存，我们采用了无跳转的设计。如果在写入每一位时都判断是否为原最小值所在位置，那么会产生大量跳转指令，浪费很多周期。考虑到此排序问题的特殊性，我们实际上可以对所有位置的先写入最小值，而后直接在原最小值位置写入次小值，这样就大大降低了保存结果时的周期消耗（节约了近 30 个时钟周期）。

5 详细设计报告

5.1 硬件架构设计

5.1.1 ALU 架构设计

在前文中已经提到过 ALU 的功能列表以及结构设计，这里不在赘述。在 ALU 中，我们采用了门级描述，单独设计了 2:1 多路复用器、4:1 多路复用器和加法器。其中加法器为最简单的行波进位加法器。综合后的 ALU 结构图如图6。

5.1.2 MCU 架构设计

MCU 共分为四大部分：控制器、数据通路、RAM 和 ROM。控制器负责产生所有控制信号，数据通路负责数据和立即数的运算，RAM 中存放原式数据和运算后的数据，ROM 中存放汇编指令。MCU 的结构如图2。其中，Instruction Memory（ROM）和 Data Memory（RAM）通过调用

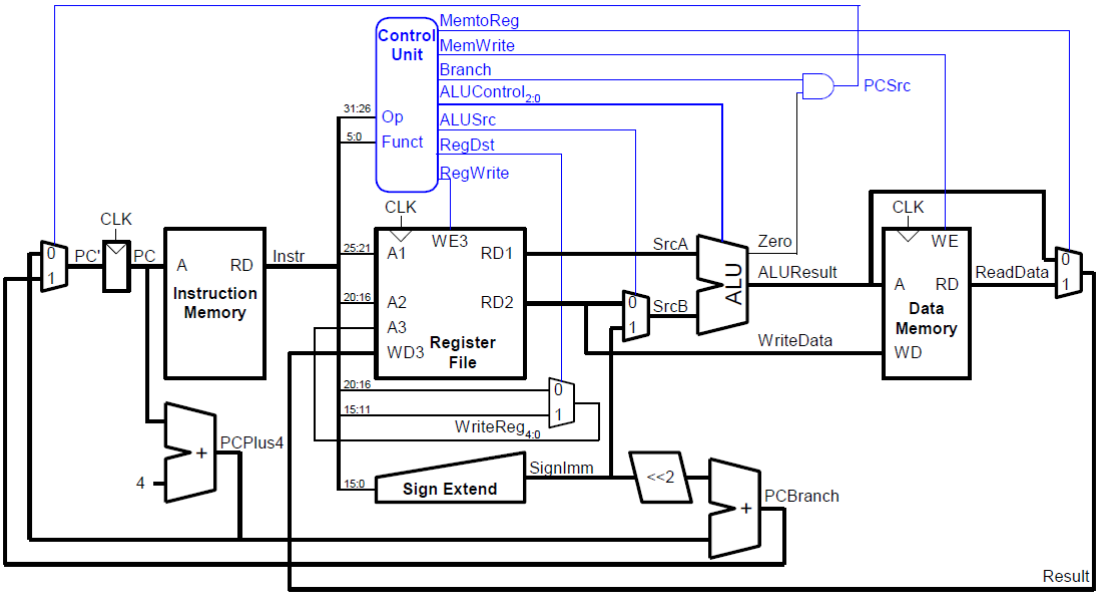

图 2: MCU 结构

IP 核“Distributed Memory Generator”来实现，其余部分通过编写 HDL 文件实现。

Datapath（数据路径）中主要包括寄存器文件、ALU、PC（program counter，程序计数器）以及数据选通器几个成分。

寄存器文件即在 MCU 中的寄存器。我们按照标准的 MIPS 寄存器集来设计我们的寄存器文件。寄存器集中共有 32 个寄存器，从 0 到 31 进行编号。寄存器文件共有七个端口，分别是 CLK（时钟）、A1 和 A2（两个 5 位读数据地址输入端）、A3（5 位写数据地址输入端）、RD1 和 RD2（两

个 32 位读数据数据端）、WD3（32 位写数据数据端）和 WE3（数据写使能控制端，高电平为写使能）。

ALU 即为上文中介绍的 ALU。

PC 是一个 32 位寄存器，通过一个加法器来实现 PC+4（指向下一条指令）的功能，还可以通过数据选通器以及符号扩展单元的加入来实现向指定的指令地址跳转的功能。

Controller（控制器）分为主译码器和 ALU 译码器两个部分。控制器负责将机器码中的 opcode 字段（$Instr_{31:26}$）和 funct（$Instr_{5:0}$）提取出来进行译码，得到用于控制 MCU 各个部分（寄存器文件、ALU、数据存储器以及各个选通器）的信号。主译码器和 ALU 译码器的真值表分别如图3和图4。

指令	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
R 类型	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01

图 3: 主译码器真值表

ALUOp	Funct	ALUControl
00	X	010(加)
X1	X	110(减)
1X	100000(add)	010(加)
1X	100010(sub)	110(减)
1X	100100(and)	000(与)
1X	100101(or)	001(或)
1X	101010(slt)	111(小于置位)

图 4: ALU 译码器真值表

综合后的 MCU 结构如图7。资源消耗情况和时序分析结果如图5。其中 ILA（Integrated Logic Analyzer，集成逻辑分析仪）是为了检测 MCU 内部信号情况而调用的 IP 核，不计算在 MCU 的消耗中。这样，MCU 就总共消耗了 836 个 LUT（Look up table，查找表）和 745 个 FF（Flip flop，触发器）。

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	WNS	TNS	WHS	THS	TPWS	LUT	FF	BRAM	URAM	DSP
synth_1 (active)	**constrs_1**						**343**	**44**	**0.0**	**0**	**0**
impl_1	constrs_1	5.677	0.000	0.044	0.000	0.000	1890	2513	25.0	0	0
Out-of-Context Module Runs											
clk_wiz_0_synth_1	clk_wiz_0						0	0	0.0	0	0
dmem_synth_1	dmem						32	32	0.0	0	0
ila_0_synth_1	ila_0						1054	1768	25.0	0	0
imem											

图 5: 资源消耗和时序分析结果

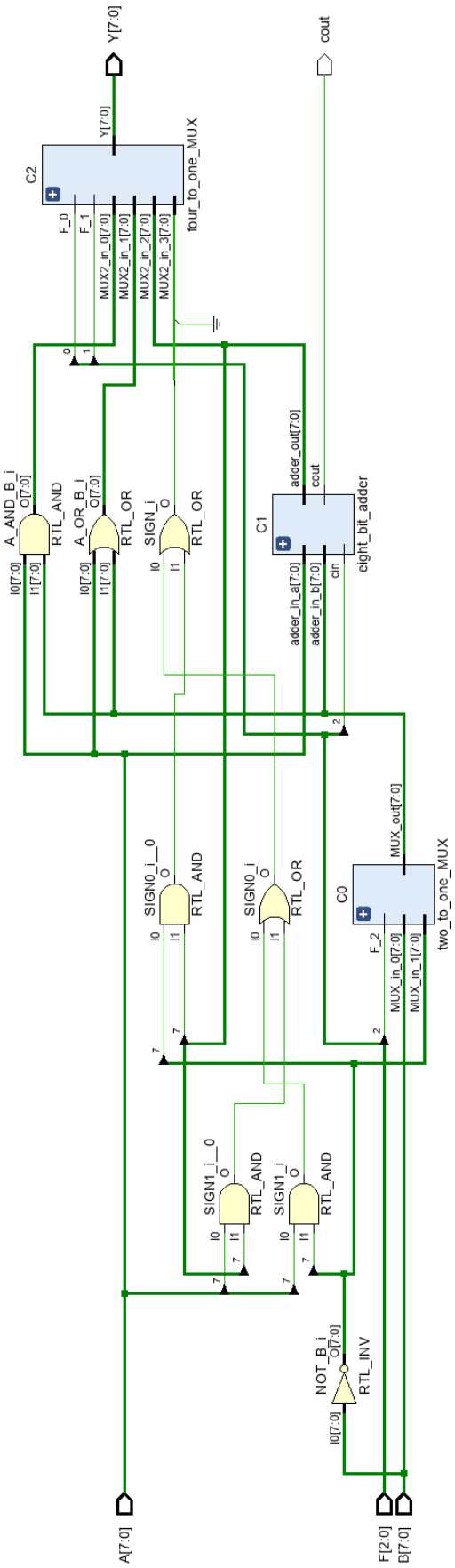
Y[7:0]
cout
C2
four_to_one_MUX
F_0
F_1
MUX2_in_0[7:0]
MUX2_in_1[7:0]
MUX2_in_2[7:0]
MUX2_in_3[7:0]
A_AND_B_i
O[7:0]
RTL_AND
A_OR_B_i
RTL_OR
SIGN_i
C1
adder_out[7:0]
eight_bit_adder
adder_in_a[7:0]
adder_in_b[7:0]
cin
SIGN0_i__0
SIGN0_i
C0
MUX_out[7:0]
two_to_one_MUX
F_2
MUX_in_0[7:0]
MUX_in_1[7:0]
SIGN1_i__0
SIGN1_i
NOT_B_i
RTL_INV
A[7:0]
F[2:0]
B[7:0]

图 6: 综合后的 ALU

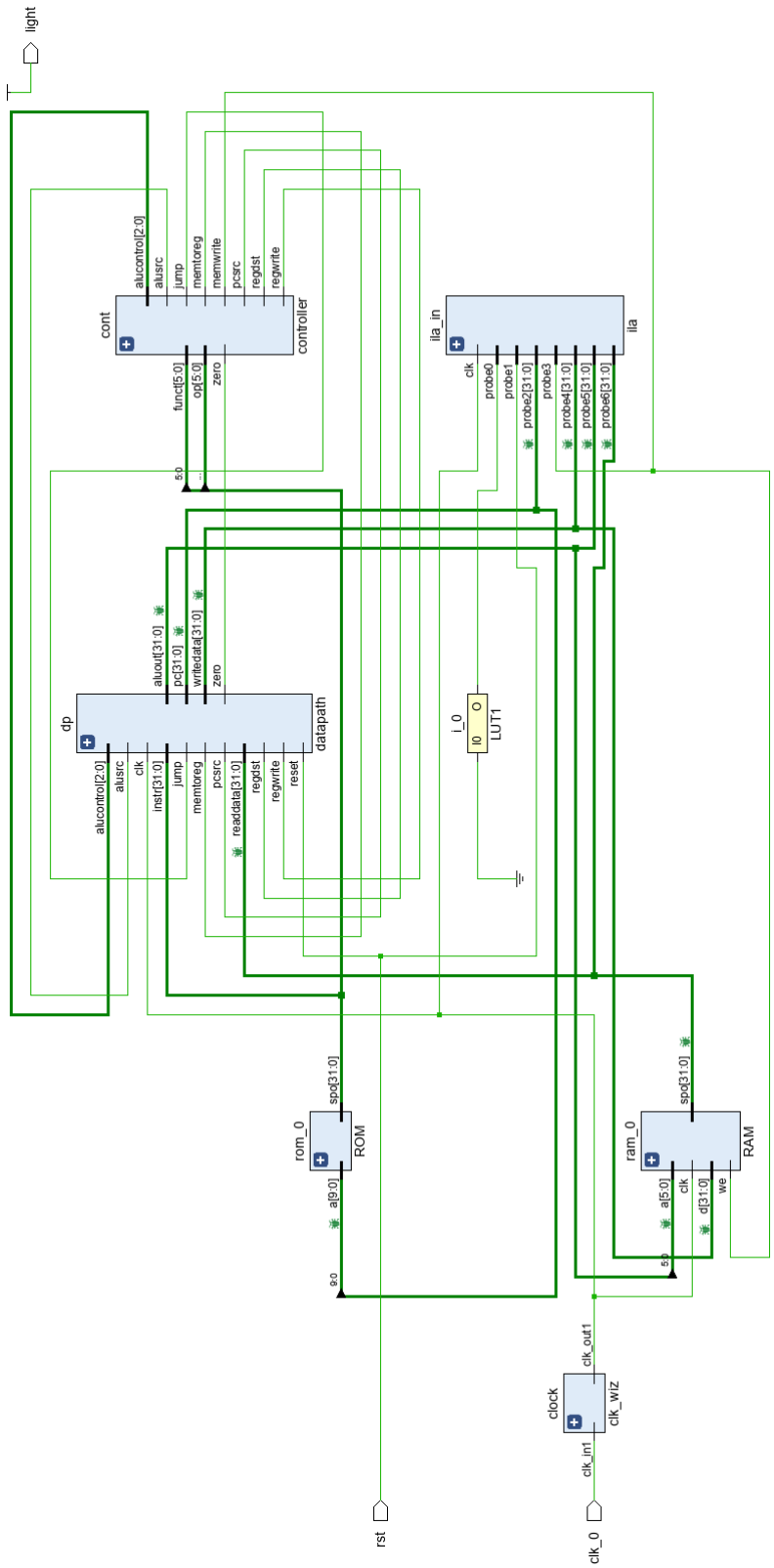
light
cont
controller
aluctrol[2:0]
alusrc
jump
memtoreg
memwrite
pcsrc
regdst
regwrite
funct[5:0]
op[5:0]
zero
ila_in
ila
clk
probe0
probe1
probe2[31:0]
probe3
probe4[31:0]
probe5[31:0]
probe6[31:0]
dp
datapath
aluctrol[2:0]
alusrc
clk
instr[31:0]
jump
memtoreg
pcsrc
readdata[31:0]
regdst
regwrite
reset
aluout[31:0]
pc[31:0]
writedata[31:0]
zero
i_0
LUT1
I0
O
rom_0
ROM
spo[31:0]
a[9:0]
ram_0
RAM
spo[31:0]
a[5:0]
clk
d[31:0]
we
clock
clk_wiz
clk_out1
clk_in1
rst
clk_0

图 7: 综合后的 MCU

5.2 软件设计

优化方法：由于寄存器有限，考虑到问题的特殊性，故分两次读入数据。采用“打擂台”方式取得最小值、次小值和最小值位置：即每次将输入值与最小值和次小值分别比较并更新，如果小于最小值则更新最小值、次小值、最小值位置，如果只小于次小值则只更新次小值。采用无跳转设计来减少写入时花费的周期数。下附完整的汇编代码和机器码。

5.2.1 汇编代码

```
        add $t0,$0,$0 #初始化
        add $t1,$0,$0
        add $t2,$0,$0
        add $t3,$0,$0
        lw $s0,0x0000($0) #读入第一批数据
        lw $s1,0x0001($0)
        lw $s2,0x0002($0)
        lw $s3,0x0003($0)
        lw $s4,0x0004($0)
        lw $s5,0x0005($0)
        lw $s6,0x0006($0)
        lw $s7,0x0007($0)
        slt $t2,$s0,$0 #取绝对值
        beq $t2,$0,a0
        sub $s0,$0,$s0
a0:     slt $t2,$s1,$0
        beq $t2,$0,a1
        sub $s1,$0,$s1
a1:     slt $t2,$s2,$0
        beq $t2,$0,a2
        sub $s2,$0,$s2
a2:     slt $t2,$s3,$0
        beq $t2,$0,a3
        sub $s3,$0,$s3
a3:     slt $t2,$s4,$0
        beq $t2,$0,a4
        sub $s4,$0,$s4
a4:     slt $t2,$s5,$0
        beq $t2,$0,a5
        sub $s5,$0,$s5
a5:     slt $t2,$s6,$0
        beq $t2,$0,a6
        sub $s6,$0,$s6
a6:     slt $t2,$s7,$0
        beq $t2,$0,a7
```

```
        sub $s7,$0,$s7
a7: add $t0,$s0,$0 #初始化最小值、次小值、最小值位置
        slt $t2,$s1,$t0
        beq $t2,$0,b1
        add $t1,$t0,$0
        add $t0,$s1,$0
        addi $t3,$0,0x0001
        beq $0,$0,c1
b1: add $t1,$s1,$0
c1: slt $t2,$s2,$t1 #打擂台
        beq $t2,$0,c2
        slt $t2,$s2,$t0
        beq $t2,$0,b2
        add $t1,$t0,$0
        add $t0,$s2,$0
        addi $t3,$0,0x0002
        beq $0,$0,c2
b2: add $t1,$s2,$0
c2: slt $t2,$s3,$t1
        beq $t2,$0,c3
        slt $t2,$s3,$t0
        beq $t2,$0,b3
        add $t1,$t0,$0
        add $t0,$s3,$0
        addi $t3,$0,0x0003
        beq $0,$0,c3
b3: add $t1,$s3,$0
c3: slt $t2,$s4,$t1
        beq $t2,$0,c4
        slt $t2,$s4,$t0
        beq $t2,$0,b4
        add $t1,$t0,$0
        add $t0,$s4,$0
        addi $t3,$0,0x0004
        beq $0,$0,c4
b4: add $t1,$s4,$0
c4: slt $t2,$s5,$t1
        beq $t2,$0,c5
        slt $t2,$s5,$t0
        beq $t2,$0,b5
        add $t1,$t0,$0
        add $t0,$s5,$0
```

```
        addi $t3,$0,0x0005
        beq $0,$0,c5
b5: add $t1,$s5,$0
c5: slt $t2,$s6,$t1
        beq $t2,$0,c6
        slt $t2,$s6,$t0
        beq $t2,$0,b6
        add $t1,$t0,$0
        add $t0,$s6,$0
        addi $t3,$0,0x0006
        beq $0,$0,c6
b6: add $t1,$s6,$0
c6: slt $t2,$s7,$t1
        beq $t2,$0,c7
        slt $t2,$s7,$t0
        beq $t2,$0,b7
        add $t1,$t0,$0
        add $t0,$s7,$0
        addi $t3,$0,0x0007
        beq $0,$0,c7
b7: add $t1,$s7,$0
c7: lw $s0,0x0008($0) #读入第二批数据
        lw $s1,0x0009($0)
        lw $s2,0x000A($0)
        lw $s3,0x000B($0)
        lw $s4,0x000C($0)
        lw $s5,0x000D($0)
        lw $s6,0x000E($0)
        lw $s7,0x000F($0)
        slt $t2,$s0,$0 #取绝对值
        beq $t2,$0,d0
        sub $s0,$0,$s0
d0: slt $t2,$s1,$0
        beq $t2,$0,d1
        sub $s1,$0,$s1
d1: slt $t2,$s2,$0
        beq $t2,$0,d2
        sub $s2,$0,$s2
d2: slt $t2,$s3,$0
        beq $t2,$0,d3
        sub $s3,$0,$s3
d3: slt $t2,$s4,$0
```

```
        beq $t2,$0,d4
        sub $s4,$0,$s4
d4:  slt $t2,$s5,$0
        beq $t2,$0,d5
        sub $s5,$0,$s5
d5:  slt $t2,$s6,$0
        beq $t2,$0,d6
        sub $s6,$0,$s6
d6:  slt $t2,$s7,$0
        beq $t2,$0,d7
        sub $s7,$0,$s7
d7:  slt $t2,$s0,$t1 #打擂台
        beq $t2,$0,f0
        slt $t2,$s0,$t0
        beq $t2,$0,e0
        add $t1,$t0,$0
        add $t0,$s0,$0
        addi $t3,$0,0x0008
        beq $0,$0,f0
e0:  add $t1,$s0,$0
f0:  slt $t2,$s1,$t1
        beq $t2,$0,f1
        slt $t2,$s1,$t0
        beq $t2,$0,e1
        add $t1,$t0,$0
        add $t0,$s1,$0
        addi $t3,$0,0x0009
        beq $0,$0,f1
e1:  add $t1,$s1,$0
f1:  slt $t2,$s2,$t1
        beq $t2,$0,f2
        slt $t2,$s2,$t0
        beq $t2,$0,e2
        add $t1,$t0,$0
        add $t0,$s2,$0
        addi $t3,$0,0x000A
        beq $0,$0,f2
e2:  add $t1,$s2,$0
f2:  slt $t2,$s3,$t1
        beq $t2,$0,f3
        slt $t2,$s3,$t0
        beq $t2,$0,e3
```

```
        add $t1,$t0,$0
        add $t0,$s3,$0
        addi $t3,$0,0x000B
        beq $0,$0,f3
e3: add $t1,$s3,$0
f3: slt $t2,$s4,$t1
        beq $t2,$0,f4
        slt $t2,$s4,$t0
        beq $t2,$0,e4
        add $t1,$t0,$0
        add $t0,$s4,$0
        addi $t3,$0,0x000C
        beq $0,$0,f4
e4: add $t1,$s4,$0
f4: slt $t2,$s5,$t1
        beq $t2,$0,f5
        slt $t2,$s5,$t0
        beq $t2,$0,e5
        add $t1,$t0,$0
        add $t0,$s5,$0
        addi $t3,$0,0x000D
        beq $0,$0,f5
e5: add $t1,$s5,$0
f5: slt $t2,$s6,$t1
        beq $t2,$0,f6
        slt $t2,$s6,$t0
        beq $t2,$0,e6
        add $t1,$t0,$0
        add $t0,$s6,$0
        addi $t3,$0,0x000E
        beq $0,$0,f6
e6: add $t1,$s6,$0
f6: slt $t2,$s7,$t1
        beq $t2,$0,f7
        slt $t2,$s7,$t0
        beq $t2,$0,e7
        add $t1,$t0,$0
        add $t0,$s7,$0
        addi $t3,$0,0x000F
        beq $0,$0,f7
e7: add $t1,$s7,$0
f7: sw $t0,0x0010($0) #写入最小值
```

```
sw $t0,0x0011($0)
sw $t0,0x0012($0)
sw $t0,0x0013($0)
sw $t0,0x0014($0)
sw $t0,0x0015($0)
sw $t0,0x0016($0)
sw $t0,0x0017($0)
sw $t0,0x0018($0)
sw $t0,0x0019($0)
sw $t0,0x001A($0)
sw $t0,0x001B($0)
sw $t0,0x001C($0)
sw $t0,0x001D($0)
sw $t0,0x001E($0)
sw $t0,0x001F($0)
sw $t1,0x0010($t3) #写入次小值
```

5.2.2 机器码

```
memory_initialization_radix = 16;
memory_initialization_vector =
                        00004020,
                        00004820,
                        00005020,
                        00005820,
                        8c100000,
                        8c110001,
                        8c120002,
                        8c130003,
                        8c140004,
                        8c150005,
                        8c160006,
                        8c170007,
                        0200502a,
                        11400001,
                        00108022,
                        0220502a,
                        11400001,
                        00118822,
                        0240502a,
                        11400001,
                        00129022,
                        0260502a,
```

11400001,
00139822,
0280502a,
11400001,
0014a022,
02a0502a,
11400001,
0015a822,
02c0502a,
11400001,
0016b022,
02e0502a,
11400001,
0017b822,
02004020,
0228502a,
11400004,
01004820,
02204020,
200b0001,
10000001,
02204820,
0249502a,
11400007,
0248502a,
11400004,
01004820,
02404020,
200b0002,
10000001,
02404820,
0269502a,
11400007,
0268502a,
11400004,
01004820,
02604020,
200b0003,
10000001,
02604820,
0289502a,
11400007,

0288502a ,
11400004 ,
01004820 ,
02804020 ,
200b0004 ,
10000001 ,
02804820 ,
02a9502a ,
11400007 ,
02a8502a ,
11400004 ,
01004820 ,
02a04020 ,
200b0005 ,
10000001 ,
02a04820 ,
02c9502a ,
11400007 ,
02c8502a ,
11400004 ,
01004820 ,
02c04020 ,
200b0006 ,
10000001 ,
02c04820 ,
02e9502a ,
11400007 ,
02e8502a ,
11400004 ,
01004820 ,
02e04020 ,
200b0007 ,
10000001 ,
02e04820 ,
8c100008 ,
8c110009 ,
8c12000a ,
8c13000b ,
8c14000c ,
8c15000d ,
8c16000e ,
8c17000f ,

0200502a,
11400001,
00108022,
0220502a,
11400001,
00118822,
0240502a,
11400001,
00129022,
0260502a,
11400001,
00139822,
0280502a,
11400001,
0014a022,
02a0502a,
11400001,
0015a822,
02c0502a,
11400001,
0016b022,
02e0502a,
11400001,
0017b822,
0209502a,
11400007,
0208502a,
11400004,
01004820,
02004020,
200b0008,
10000001,
02004820,
0229502a,
11400007,
0228502a,
11400004,
01004820,
02204020,
200b0009,
10000001,
02204820,

0249502a ,
11400007 ,
0248502a ,
11400004 ,
01004820 ,
02404020 ,
200b000a ,
10000001 ,
02404820 ,
0269502a ,
11400007 ,
0268502a ,
11400004 ,
01004820 ,
02604020 ,
200b000b ,
10000001 ,
02604820 ,
0289502a ,
11400007 ,
0288502a ,
11400004 ,
01004820 ,
02804020 ,
200b000c ,
10000001 ,
02804820 ,
02a9502a ,
11400007 ,
02a8502a ,
11400004 ,
01004820 ,
02a04020 ,
200b000d ,
10000001 ,
02a04820 ,
02c9502a ,
11400007 ,
02c8502a ,
11400004 ,
01004820 ,
02c04020 ,

200b000e ,
10000001 ,
02c04820 ,
02e9502a ,
11400007 ,
02e8502a ,
11400004 ,
01004820 ,
02e04020 ,
200b000f ,
10000001 ,
02e04820 ,
ac080010 ,
ac080011 ,
ac080012 ,
ac080013 ,
ac080014 ,
ac080015 ,
ac080016 ,
ac080017 ,
ac080018 ,
ac080019 ,
ac08001a ,
ac08001b ,
ac08001c ,
ac08001d ,
ac08001e ,
ac08001f ,
ad690010 ;

6 测试报告

经过测试，我们的 MCU 可以运行所有指令，并能够完成排序算法。由于时间紧迫，我们没有对最终的验收成果进行截图。图8为一次中途验收的效果截图。图中的存入顺序和数据都是正确的。

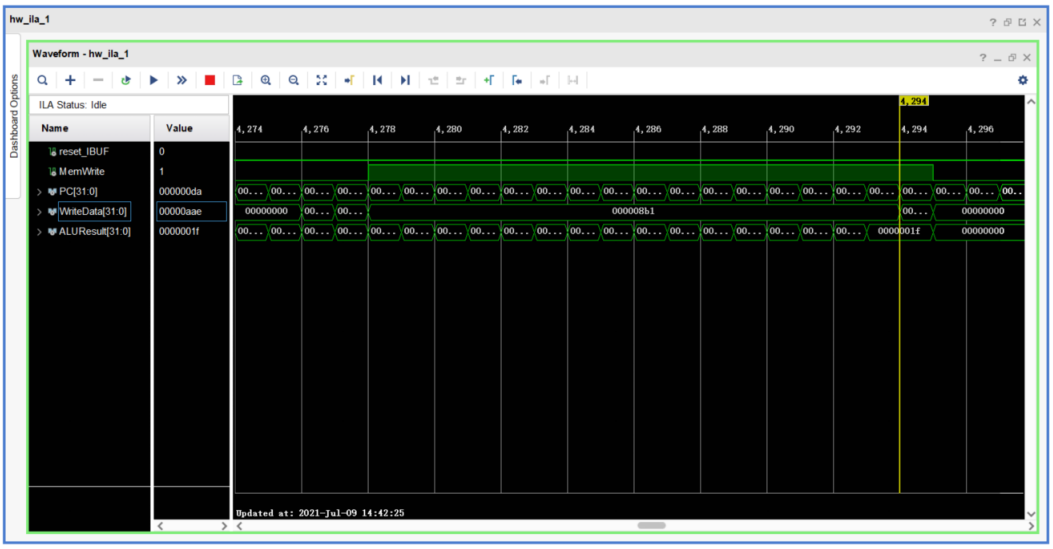

图 8: 中途验证的结果

7 总结和展望

我们这次的课程设计总的来说是成功的。我们从零开始实现了一个完整的 MCU，自己完成了算法的设计和汇编语言的书写，受益匪浅。

关于硬件的展望与思索：在设计过程中我们依然走了很多弯路，比如有些地方可以用行为级描述来书写结果却用了门级描述；没有在开始设计前构思好 MCU 的具体结构，导致中途返工多次，等等。这也提示我们今后在进行设计时要多动脑，同时做好规划和总结。

关于算法的展望和思索：如果测试数据的排列顺序为从大到小，则使用当前的算法进行排序时将会消耗较大的时间，浪费很多时钟周期。后续可以通过改进算法来提升运行效率。