数字逻辑与计算机组成

实验 6: 单周期 CPU 设计与测试

王卫东　221900332

2023 年 6 月 6 日

一　实验目的

1. 掌握 RV32I 控制器的设计方法。
2. 掌握单周期 CPU 中的时序设计。
3. 掌握 RISC-V 汇编语言程序的基本设计方法。
4. 理解汇编语言程序与机器语言代码之间的对应关系。

二　实验环境

Logisim：https://github.com/Logisim-Ita/Logisim
RISC-V 模拟器工具 RARS：https://github.com/thethirdone/rars

三　实验内容

1　控制器设计实验

1.1　整体方案设计

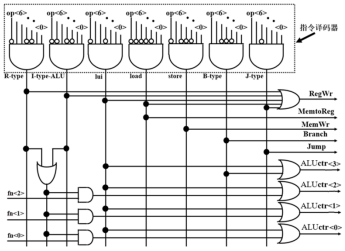

图 1: 控制器整体方案设计

1.2　引脚作用

opcode	指令代码中的操作码
funct3	指令代码中的功能码
funct7	指令代码中的功能码
ExtOp	指令代码中的扩展操作码
RegWr	寄存器写使能
ALUASrc	ALU 第一个操作数选择信号
ALUBSrc	ALU 第二个操作数选择信号
ALUctr	ALU 控制信号
MemWr	存储器写使能
MentoReg	存储器读数据到寄存器使能
Branch	分支使能
MemOp	存储器读写选择信号
halt	停机信号
U-lui 等	7 位指令 code 用一位标志位表示

表 1: 控制器引脚作用

RV32I 指令控制信号列表见数逻实验报告 6 指南, 此处只截取了一部分。

表 6.1 RV32I 指令控制信号列表

指令	类型	op[6:0]	func3	func7[5]	ExtOp	RegWr	ALUASrc	ALUBSrc	ALUctr
lui	U	0110111	×	×	001	1	×	10	1111
auipc	U	0010111	×	×	001	1	1	10	0000
addi	I	0010011	000	×	000	1	0	10	0000
slti	I	0010011	010	×	000	1	0	10	0010
sltiu	I	0010011	011	×	000	1	0	10	0011
xori	I	0010011	100	×	000	1	0	10	0100
ori	I	0010011	110	×	000	1	0	10	0110
andi	I	0010011	111	×	000	1	0	10	0111
slli	I	0010011	001	0	000	1	0	10	0001
srli	I	0010011	101	0	000	1	0	10	0101
srai	I	0010011	101	1	000	1	0	10	1101
add	R	0110011	000	0	×	1	0	00	0000
sub	R	0110011	000	1	×	1	0	00	1000
sll	R	0110011	001	0	×	1	0	00	0001
slt	R	0110011	010	0	×	1	0	00	0010
sltu	R	0110011	011	0	×	1	0	00	0011
xor	R	0110011	100	0	×	1	0	00	0100

图 2: RV32I 指令控制信号

1.3 原理图和电路图

原理图同整体方案设计图。

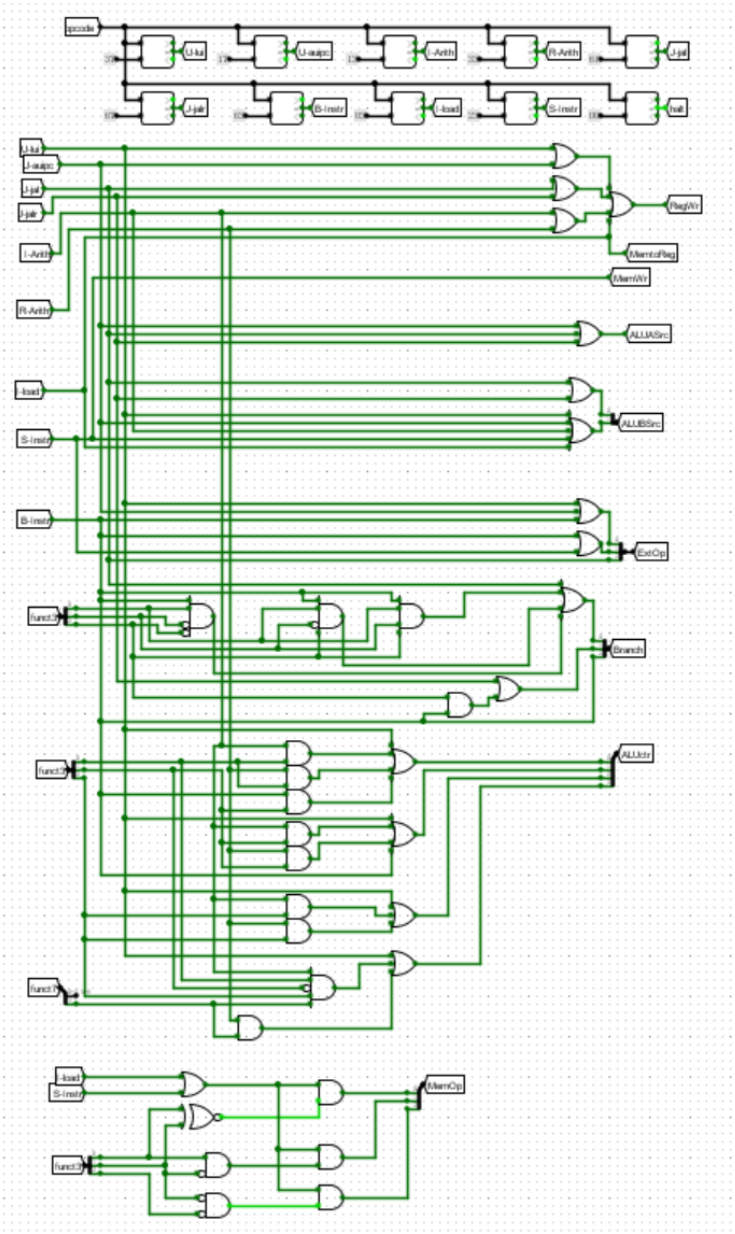
opcode
U-lui
U-auipc
I-Arith
R-Arith
J-jal
J-jalr
B-Instr
I-load
S-Instr
halt
RegWr
MemtoReg
MemWr
ALUASrc
ALUBSrc
ExtOp
Branch
ALUctr
MemOp

图 3: 控制器电路图

1.4 仿真测试图

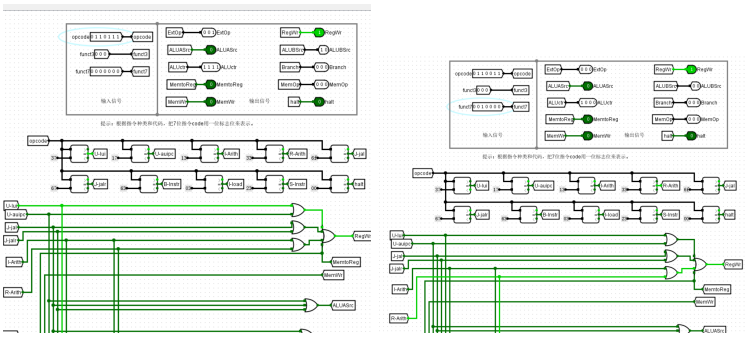

图 4: 控制器仿真测试图

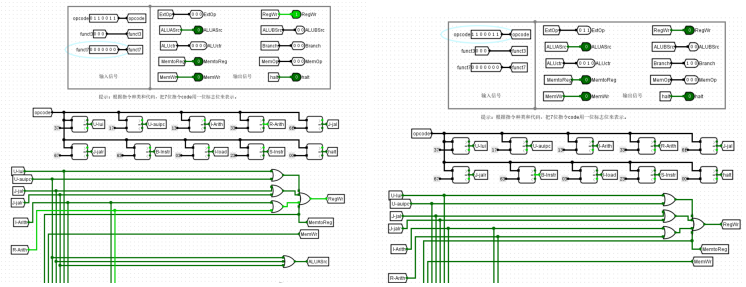

图 5: 控制器仿真测试图

- 图 1 为 lui 指令,opcode=0110111,funct3=000,funct7=0000000,ExtOp=001,RegWr=1，ALUASrc=0,ALUBSrc=10,ALUctr=1111,MemWr=0,MentoReg=0,Branch=0，MemOp=0,halt=0, 符合预期。
- 图 2 为 sub 指令,opcode=0110011,funct3=000,funct7=0010000,ExtOp=000,RegWr=1，ALUASrc=0,ALUBSrc=00,ALUctr=1000,MemWr=0,MentoReg=0,Branch=0，MemOp=0,halt=0, 符合预期。
- 图 3 为 add 指令,opcode=0110011,funct3=000,funct7=0000000,ExtOp=000,RegWr=1，ALUASrc=0,ALUBSrc=00,ALUctr=0000,MemWr=0,MentoReg=0,Branch=0，MemOp=0,halt=0, 符合预期。
- 图 4 为 beq 指令,opcode=1100011,funct3=000,funct7=0000000,ExtOp=011,RegWr=0，

ALUASrc=0,ALUBSrc=00,ALUctr=0010,MemWr=0,MentoReg=0,Branch=100，MemOp=0,halt=0, 符合预期。

1.5 错误现象及分析

在完成实验的过程中, 没有遇到任何错误。

2 单周期 CPU 设计实验

2.1 整体方案设计

在单周期 CPU 中, 每条的指令都需要在一个时钟周期内完成。本次实验中, 以时钟下降沿为每个时钟周期的开始, 写入操作在时钟下降沿时同步实现, 而读取操作, 则是异步实现, 只要输入有效地址后, 立即输出对应数据。同时需要考虑的时序信号, 有复位信号 Reset、片选信号 Sel、中止信号 halt 等。

整体上只需要加入控制器, 对实验 5 中的 DataPath 的引脚略作修改即可。

2.2 顶层模块设计

在前期的实验中已经完成全部模块的设计, 在实验 3 中实现了寄存器堆模块, 实验 4 中实现了 ALU 模块, 实验 5 中实现了数据存储器、取指令部件和数据通路模块, 本次实验完成了控制器模块。

2.3 引脚作用

clk	时钟信号
Init Addr	初始化地址
Reset	复位信号
Halt	中止信号
Halt0	判断程序执行周期终止信号

表 2: 单周期 CPU 引脚作用

2.4 原理图和电路图

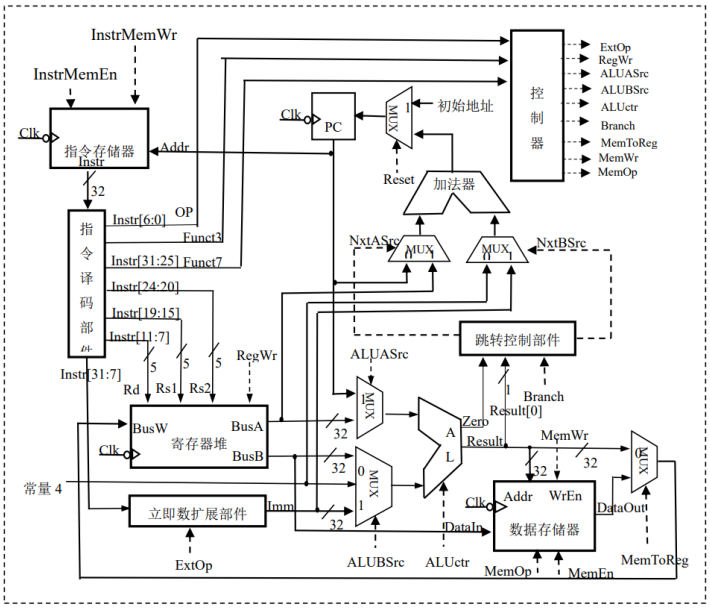

图 6: 单周期 CPU 原理图

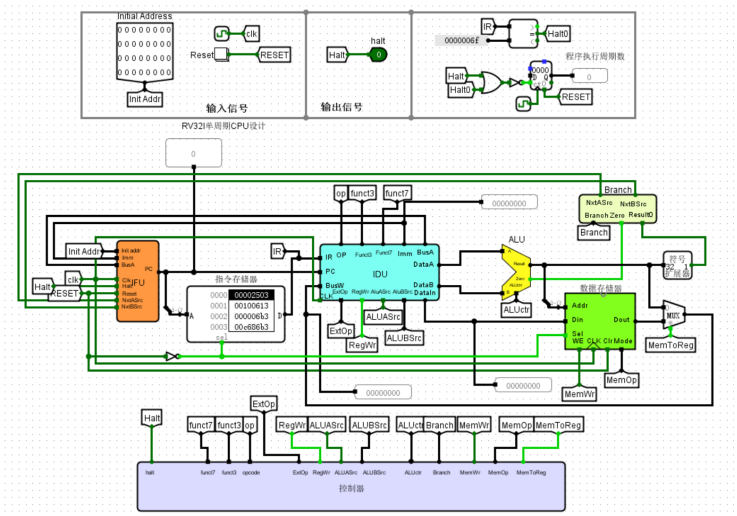

图 7: 单周期 CPU 电路图

2.5 仿真测试图

仿真测试见下三个验收实验。

2.6 错误现象及分析

在完成实验的过程中, 经常遇到 Regfile 红线的情况, 经检查是 Regfile 中线路重叠的问题, 将其解决后, 仿真测试通过。

3 用累加和程序验证 CPU 设计

首先, 将计算累加和的 RV32I 汇编语言程序在 RARS 中调试通过, 汇编成机器代码并导出。然后将机器代码载入到 CPU 的指令存储器部件中。设置参数 n, 启动时钟信号, 执行代码, 程序执行结束后, 观察寄存器和存储器指定存储单元中的结果, 验证指令执行的正确性。(实验文件已经给出机器代码镜像文件, 故直接展示仿真实验结果)

3.1 仿真测试图

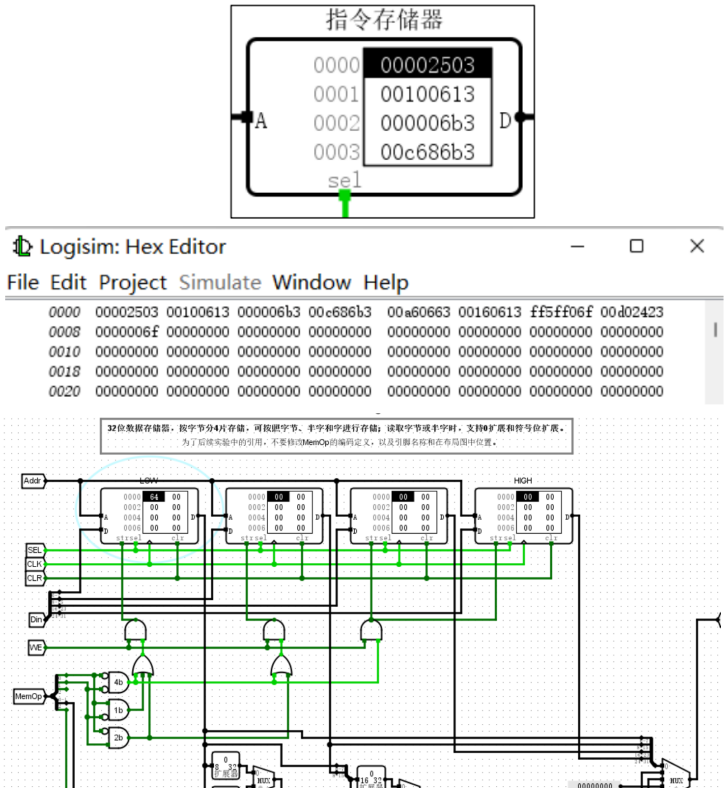

图 8: 累加和程序数据导入图

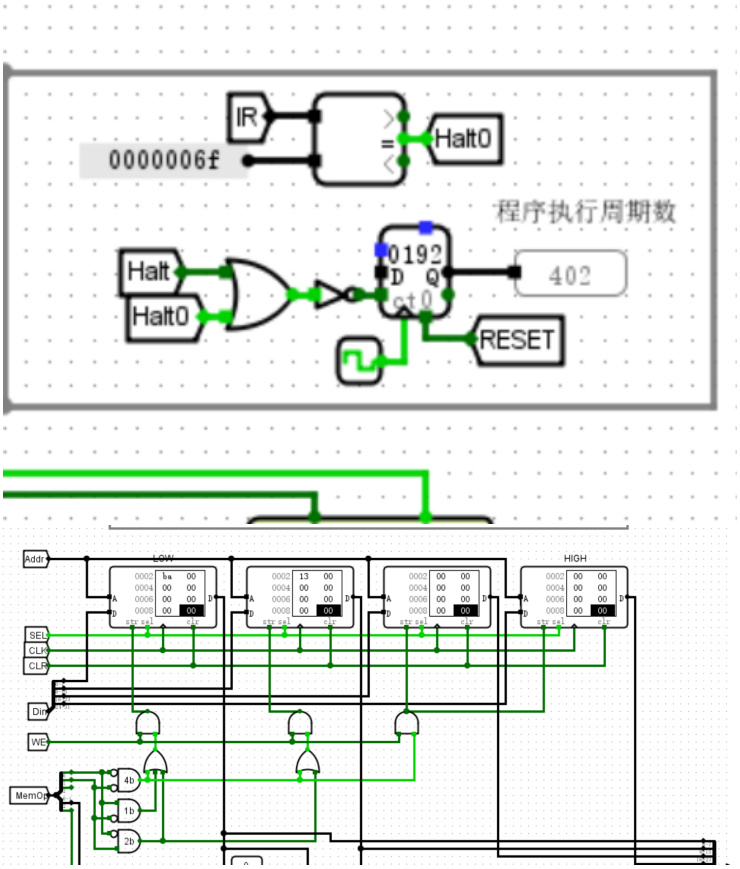

图 9: 累加和程序数据验证图

可以看到运行了 402 个周期, 最终累加和的计算机结果为 0x13ba（十进制数 5050）, 与预期结果一致。

4 用冒泡排序程序进行 CPU 设计验证

汇编代码中将数据个数 n 读到寄存器 $a0$, 常量 1 保存在寄存器 $a1$ 中, 外循环变量 i 保存在 $a2$, 内循环变量 j 保存在 $a3$, 第 j 个元素 $a[j]$ 的地址存放 $a4$。第 j 个元素 $a[j]$ 读入 $a6$, 第 $j+1$ 个元素 $a[j+1]$ 读入 $a7$。

在指令存储器中加载冒泡程序可执行机器代码数据镜像文件 bubble.hex, 打开数据存储器子电路, 在最低字节 RAM 中加载待排序数据镜像文件 bubble.dat, 观察运行情况。

4.1 仿真测试图

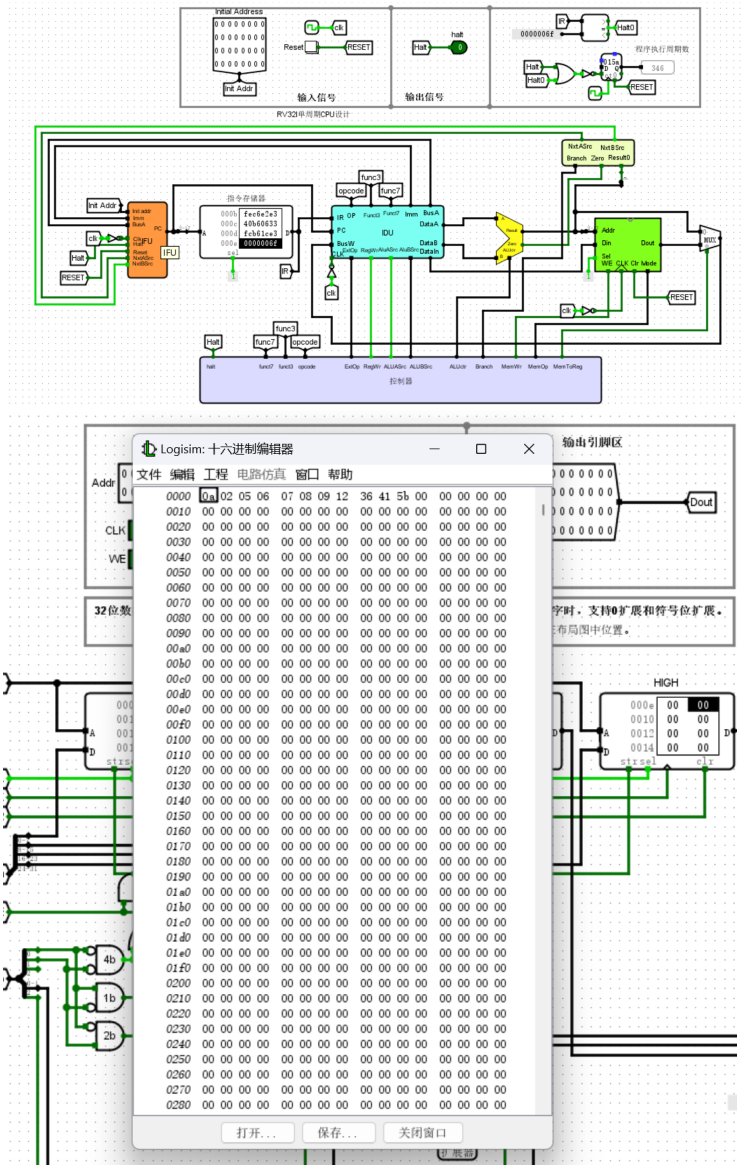

图 10: 累加和程序数据验证图

打开最低字节 RAM, 可以看到数据已经被排序：a 2 5 6 7 8 9 12 36 41 5b

5 C 程序汇编测试

将 b-sort-riscv32-npc.bin-logisim-inst.txt 文件加载到指令存储器中, 将 b-sort-riscv32-npc.bin-logisim-data0.txt 加载到最低字节的数据存储器中, 将 b-sort-riscv32-npc.bin-logisim-data1.txt 加载到次低字节的数据存储器, 以此类推, 加载 b-sort-riscv32-npc.bin-logisim-data2.txt、b-sort-riscv32-npc.bin-logisim-data3.txt 到指定的数据存储器中。选择连续时钟信号, 启动程序, 分析程序执行的结果。

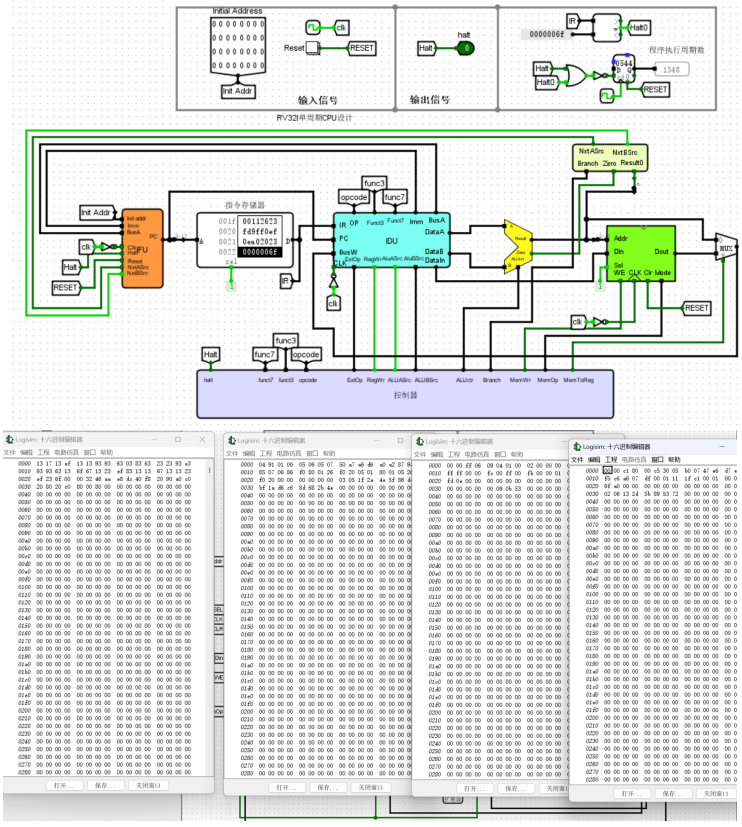

图 11: C 程序验证图

程序运行了 1348 个周期, 最终排序结果经整理如下, 符合预期。、

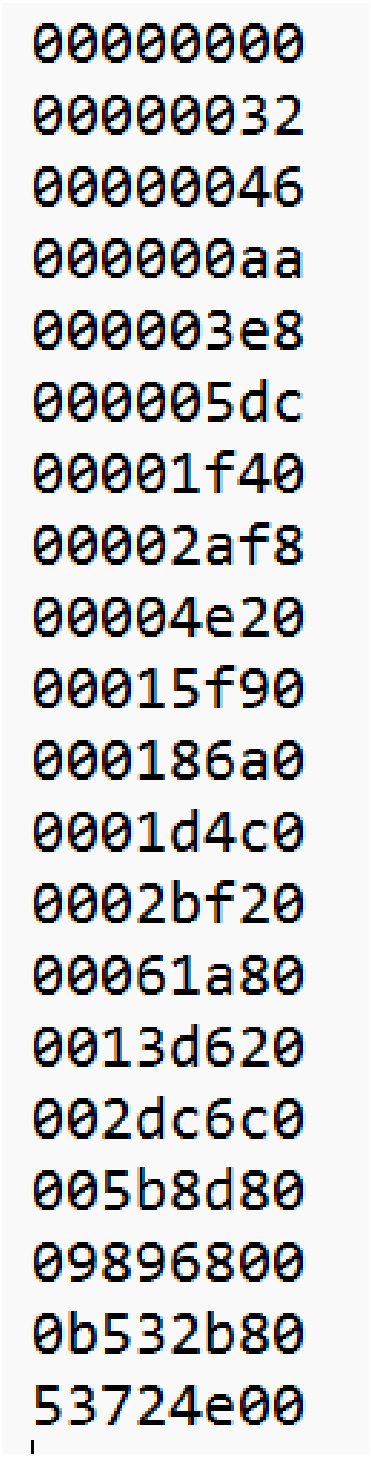
00000000
00000032
00000046
000000aa
000003e8
000005dc
00001f40
00002af8
00004e20
00015f90
000186a0
0001d4c0
0002bf20
00061a80
0013d620
002dc6c0
005b8d80
09896800
0b532b80
53724e00

图 12: C 程序验证结果

四　思考题

1　在累加和计算程序中, 添加溢出判读语句, 并把最大的不溢出累计和以及累加序数保存到数据存储器中输出。

在累加和计算程序中，添加溢出判断语句可以在每次累加操作后进行。可以通过比较累加结果和数据存储器中的最大值来判断是否发生了溢出。如果发生了溢出，需要将累加序数和最大不溢出累计和保存到数据存储器中输出。

可以在累加指令"add a3, a3, a2" 之后添加如下代码：

```
addiw a4, a3, 0 # 将累加和 a3 的低 32 位复制到 a4 中
blt a4, a3, overflow # 若 a4 < a3，则说明溢出，跳转到 overflow 处
j continue # 若未溢出，则跳转到 continue 处
```

在溢出处理代码"overflow" 处，可以将最大不溢出的累计和以及累加序数保存到数据存储器中输出。具体来说，可以添加如下代码：

```
sub a5, a5, a0 # 减去最后一个元素的值
sw a5, 4(x0) # 将最大不溢出的累计和保存到数据存储器 0x0004 单元
addi a6, a2, -1 # 将累加序数设为 i-1
sw a6, 12(x0) # 将累加序数保存到数据存储器 0x000C 单元
j end # 跳转到程序结束处
```

完整的代码如下：

```
main:
lw a0,0(x0) # 从数据存储器地址 0x0000 单元中读取参数 n 到寄存器 a0;
addi a2, x0,1 # 循环变量 i，存放在 a2，初值为 1
add a3,x0,x0 # 累计和存放在 a3，初值为 0
loop:
add a3, a3, a2 # 将 a3=a3+i
addiw a4, a3, 0 # 将累加和 a3 的低 32 位复制到 a4 中
blt a4, a3, overflow # 若 a4 < a3，则说明溢出，跳转到 overflow 处
j continue # 若未溢出，则跳转到 continue 处
overflow:
sub a5, a5, a0 # 减去最后一个元素的值
```

```
sw a5, 4(x0) # 将最大不溢出的累计和保存到数据存储器 0x0004 单元
addi a6, a2, -1 # 将累加序数设为 i-1
sw a6, 12(x0) # 将累加序数保存到数据存储器 0x000C 单元
j end # 跳转到程序结束处
continue:
beq a2, a0, finish # 若 i=n，则跳出循环
addi a2, a2, 1 # i++
jal x0, loop # 无条件跳转到 loop 执行
finish:
sw a3, 8(x0) # 将累加结果保存到数据存储器 0x0008 单元
end:
jal x0, end # 无条件跳转到 end
```

2 如果分支跳转指令不在 ALU 内部使用减法运算来实现, 而是在 ALU 外使用独立比较器来实现, 说说单周期 CPU 的电路原理图中需要做哪些修改?

1. 修改控制逻辑：将分支跳转指令的识别和控制信号生成从 ALU 控制部分移动到比较器模块控制部分。
2. 修改指令解码器: 将分支跳转指令的识别从 ALU 操作码识别部分移动到比较器操作码识别部分。
3. 修改 DataPath: 将比较器的输出和控制信号传递到 PC 寄存器和数据存储器，以实现分支跳转指令的功能。

3 实现单周期 CPU 后, 如何实现键盘输入、TTY 输出部件等输入输出设备的数据访问, 构建完整的计算机系统。

1. 键盘输入：可以通过添加一个键盘控制器，将键盘输入数据传输到 CPU 中。键盘控制器可以使用 PS/2 或 USB 接口与键盘连接，并将输入数据传输到 CPU 的 I/O 地址空间。CPU 可以通过读取 I/O 地址空间来获取键盘输入数据。
2. TTY 输出：可以通过添加一个串行通信控制器，将 CPU 输出数据传输到 TTY 设备中。串行通信控制器可以使用 RS-232 或 USB 接口与

TTY 设备连接，并将输出数据转换为串行数据流进行传输。CPU 可以通过写入 I/O 地址空间来将输出数据传输到串行通信控制器中。

4 如果需要实现 5 级流水线 RV32I CPU, 则如何在单周期 CPU 基础上进行修改？

1. 增加流水线寄存器：在单周期 CPU 的基础上, 需要增加 5 个流水线寄存器, 用于保存不同阶段的指令执行结果。这些寄存器可以用 D 触发器实现。
2. 增加控制逻辑：需要增加控制逻辑, 用于控制流水线的各个阶段。
3. 增加数据冲突检测。
4. 增加分支预测。