沈阳航空航天大学

# 计算机组成原理与体系结构课程设计报告

| | | | |
|---|---|---|---|
| **院系** | **人工智能学院** | **专　　业** | 物联网工程 |
| **题目** | 定点补码加法器的设计与实现 | | |
| **班级** | 物联网 2202 | **姓　　名** | **陈梓欣** |
| **学号** | **223428010210** | **指导教师** | **王传云** |

以下内容由指导教师填写

| **序号** | **评价项目** | **评分** | |
|---|---|---|---|
| | | **满分** | **得分** |
| **1** | 查阅资料和自我学习的能力。 | 10 | |
| **2** | 课设题目的理解情况和完成情况，以及仿真的正确性与完善性。 | 40 | |
| **3** | 检查过程中问题回答的准确程度 | 20 | |
| **4** | 课程设计报告的格式和内容，侧重考虑内容充实度、图表齐全度、对设计和实现过程的描述详实度、仿真和测试的全面度等方面。 | 30 | |
| 累计得分 | | | |

指导教师评语：

**课设成绩：**

指导教师签名

日期　　年　　月　日

**注：成绩评定采用五级记分制。优秀(90～100 分)、良好(80～89 分)、中等(70～79 分)、及格(60～69 分)、不及格(60 分以下)**

# 沈阳航空航天大学

# 课程设计任务书

| | | | | | |
|---|---|---|---|---|---|
| 课程名称 | 计算机组成原理与体系结构课程设计 | | | 专业 | 物联网工程 |
| 学生姓名 | 陈梓欣 | 班级 | 物联网 2202 | 学号 | 223428010210 |
| 题目名称 | 定点补码加法器的设计与实现 | | | | |
| 起止日期 | 2024 年 12 月 16 日起至 2024 年 12 月 27 日止 | | | | |

课设内容和要求：

一、课程设计内容

采用伟福 COP2000 实验箱设计并实现定点补码加法器。

二、课程设计要求

1．分析题目要求，用汇编语言编程实现定点补码加法器；

2．理解算法思想，在掌握 COP2000 实验箱的体系结构后，充分利用实验箱提供的汇编语言，以及有限的硬件资源，完成程序的编写；

3．两相加数的位数为 7 位，其中包括一位符号位，采用原码输入；

4．对设计程序进行仿真并验证其正确性，仿真数据由指导教师给出；

5．实现编程下载和整体测试；

6．独立设计、调试、仿真、下载和硬件测试并通过指导教师现场验收；

7．撰写课程设计报告。

参考资料：

[1] 唐朔飞.计算机组成原理（第 2 版）[M].北京：高等教育出版社，2008

[2] 曹昕燕. EDA 技术实验与课程设计 [M].北京：清华大学出版社，2006

[3] 范延滨.微型计算机系统原理、接口与 EDA 设计技术[M].北京：北京邮电大学出版社，2006

[4] 王爱英.计算机组成与结构(第 4 版)[M].北京：清华大学出版社，2006

**题目难度：** **B** A 难度或工作量较大 B 难度或工作量一般 C 难度或工作量较小

**教研室审核意见：同意（√） 不同意（ ） 教研室主任签字：** 王静

**指导教师（签名）** 王静 **2024** 年 **12** 月 **12** 日

**学生签名** 陈梓欣 **2024** 年 **12** 月 **12** 日

# 课程设计总结

通过本次课程设计，我对计算机组成原理和汇编语言的理解有了更深入的掌握，特别是在使用 COP2000 软件模拟进行有符号二进制七位数的运算和补码运算方面积累了宝贵经验。在设计过程中，我完成了一个定点补码加法器的实现，深入研究了补码的转换过程，解决了多项实际操作中的挑战，并加深了对计算机体系结构和指令集的理解。

设计首先从数的表示和输入开始。根据要求，我使用原码表示输入数，并通过程序将其转换为补码。通过学习和编程，我理解了补码加法器的工作原理，特别是在负数加法时的补码转换。通过对数 A 和数 B 的补码加法，我解决了进位和溢出的处理问题。同时，程序设计中需要有效检测输入错误，确保数据合法，防止计算错误。

在编程过程中，我遇到了一些问题。最初在补码转换时，符号位判断不准确，导致补码转换错误。通过查阅资料并修正符号位判断条件，我确保了负数的补码加法能正确执行。溢出检测与处理也是一大挑战，尤其是在补码加法中可能出现的上溢和下溢情况。通过调试与验证，我完善了溢出检测条件，确保程序的稳定性和正确性。

此次课程设计不仅加深了我对汇编语言的理解，也提升了我的问题分析与解决能力。最重要的是，我学会了如何在硬件与软件之间架起桥梁，通过合理的程序设计优化硬件功能。设计过程中遇到的问题促使我更细致地思考，意识到编程中的细节至关重要，逐步调试确保程序的正确性。

总之，本次课程设计是对我计算机基础知识的全面实践，增强了我对计算机硬件和汇编语言的理解。在今后的学习和工作中，我将保持严谨的态度，继续探索并解决问题，提升自身能力和知识水平。

# 目　　录

**1　题目介绍 ..... 1**

1.1 题目内容 ..... 1
1.2 设计思路 ..... 1
1.3 设计环境 ..... 1

**2　详细设计方案 ..... 2**

2.1 整体设计方案 ..... 2
2.1.1 原、补码转换部分 ..... 2
2.1.2 加法运算执行 ..... 2
2.2　初始化功能模块 ..... 3
2.3 输入原码数据模块 ..... 4
2.4 原码到补码转换模块 ..... 5
2.5 补码加法运算模块 ..... 6
2.6 溢出检测模块 ..... 7
2.7 结果输出模块 ..... 9
2.8 错误处理模块 ..... 10

**3　测试与验证 ..... 11**

3.1　测试样例 ..... 11
3.2　测试过程 ..... 12
3.2.1 输入数据非法 ..... 12
3.2.2 输入合法但结果上溢 ..... 14
3.2.3 输入合法但结果下溢 ..... 15
3.2.4 输入与结果均合法 ..... 16
3.2.5 输入数据存在“0”项 ..... 18

**参考文献 ..... 20**

**附　　录 ..... 21**

# 1 题目介绍

## 1.1 题目内容

本次课程设计要求利用伟福 COP2000 实验系统实现定点补码加法器。程序需通过汇编语言实现对两个有符号二进制七位数的加法运算，支持补码表示，处理加法过程中的进位和溢出问题，并能够检测和转换输入的原码数据为补码格式。设计中还包括溢出检测功能，判断是否发生上溢或下溢，最终将结果存储在内存中并输出。

## 1.2 设计思路

程序首先通过原码输入数据，转换为补码格式，并存储在相应的寄存器和内存中。然后，通过执行补码加法运算，考虑进位和符号位，计算加法结果。在加法结果存储之前，程序需要检测是否发生了溢出。溢出检测包括上溢和下溢，通过分析加数的符号位和结果符号位来判断。在溢出发生时，程序会输出相应的溢出标志；若没有溢出，则直接输出加法结果。

设计中包括以下几个关键步骤：

a) 输入检查：检测输入的原码数据是否符合一位符号位，六位数据位格式。

b) 原码转换为补码：将输入的原码数据转换为补码，处理正数和负数的情况。

c) 补码加法运算：对两个补码数进行加法，并考虑进位的处理。

d) 溢出检测：检测加法过程中的上溢和下溢情况。

e) 结果输出：根据计算结果，判断并输出溢出标志，或输出加法结果。

## 1.3 设计环境

本次设计使用伟福 COP2000 计算机组成原理实验系统，该系统支持汇编语言编程和计算机组成原理实验。在该环境下，我们可直接操作内存和寄存器，通过汇编指令来实现各项功能。开发工具为 COP2000 的软件模拟，通过该工具，我们可以在未连接 COP2000 实验仪的情况下，也可仿真的进行程序的编写、调试和验证。

# 2 详细设计方案

## 2.1 整体设计方案

程序的设计目标是完成两个有符号数的原码补码转换以及对转换后的补码进行加法运算。在设计过程中，首先需要将输入的原码数转换为补码形式，然后使用补码进行加法运算，最终判断加法结果是否发生溢出。溢出检测的关键在于分析加法过程中符号位的变化，根据符号位的溢出特性来判定上溢或下溢的发生情况。根据这一过程，整个设计可以分为两个主要部分：原、补码转换部分和加法运算执行部分。

### 2.1.1 原、补码转换部分

程序首先通过输入的原码数据判断数值的有效性。如果输入数据在有效范围内，程序会继续执行补码转换。如果原码数是正数，则原码和补码相同，直接保存原码即可；如果原码数是负数，则程序需要将原码转换为补码。类似的，由于加法运算后的结果为补码格式，需要将其转换回原码格式。

### 2.1.2 加法运算执行

程序会对两个数的补码进行加法运算，并将加法结果存储在指定的内存地址。补码加法的过程通过 ADD 指令实现，运算结果将保存在内存中，并根据符号位进行溢出检测。

如果加法结果没有发生溢出，在补码格式正确转换回原码格式后，程序将输出原码格式的加法结果。

整体结构流程图如图 2.1 所示。

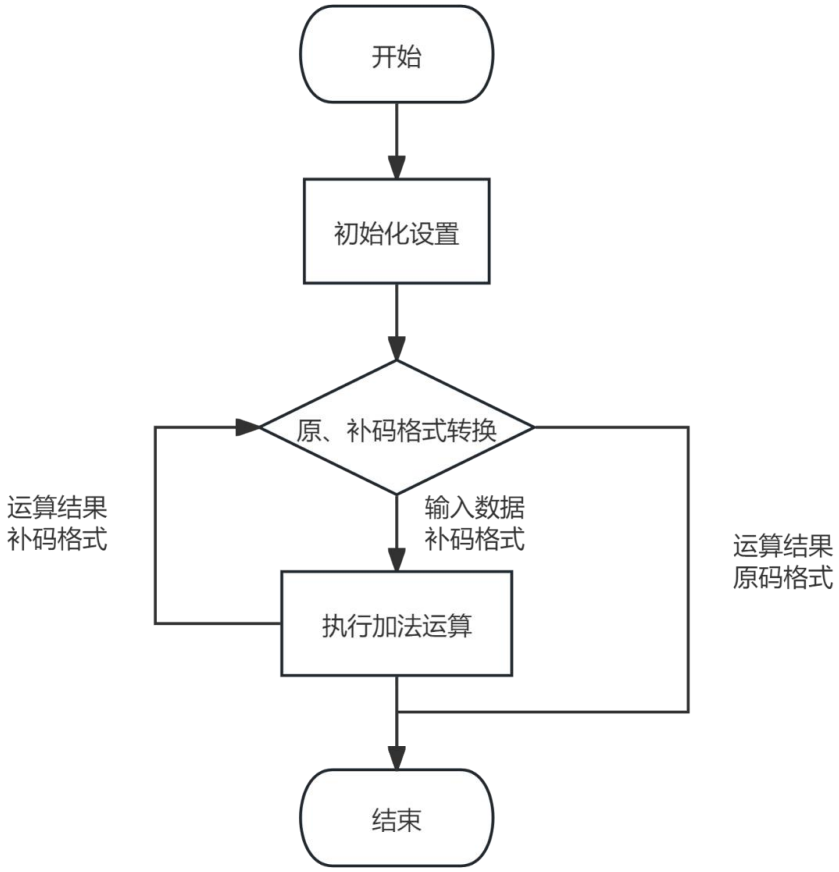


**图 2.1　整体结构流程图**

## 2.2　初始化功能模块

在程序开始时，首先需要对所有寄存器和内存单元进行初始化，以确保程序能够正确运行。初始化的目标是为后续的数值转换、加法运算和溢出检测提供合适的初始状态。具体来说，首先，需要初始化的是内存单元 0D0H 到 0D2H，来存储未进行数据合法性检查的数 A、数 B、加法运算结果的原始数据格式；其次，我们将 R0 寄存器初始化为 0，用于存储数 A 的原码；R1 寄存器初始化为 0，用于存储数 B 的原码；R2 和 R3 分别初始化为 0，用于存储数 A 和数 B 的补码。此外，程序还需要初始化内存单元 0E0H 到 0E5H，用于存储溢出检测时需要用到的符号位及其状态。通过初始化这些内存单元和寄存器，程序确保了数据的正确加载和后续操作的顺利进行。初始化模块的流程图如图 2.2 所示。

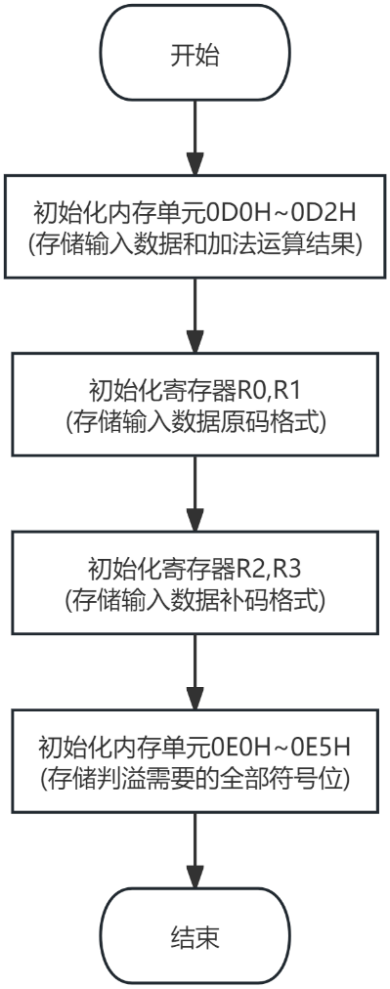


**图 2.2　初始化模块实现流程图**

## 2.3 输入原码数据模块

输入原码数据模块是程序的核心输入部分，负责从外部获取原码数据并将其存储到指定的内存单元中。在本模块中，程序会读取数 A 和数 B 的原码并验证其有效性。具体操作是将数 A 的原码存储到内存单元 0D0H 中，将数 B 的原码存储到内存单元 0D1H 中。通过对原码的检查，程序可以判断输入的数据是否符合规定的范围：00H~7FH 之间（7 位有效原码）。如果数据合法，则程序将跳转到下一个模块进行处理；否则，程序会跳转到错误处理模块进行处理。输入原码数据模块的流程图如图 2.3 所示。

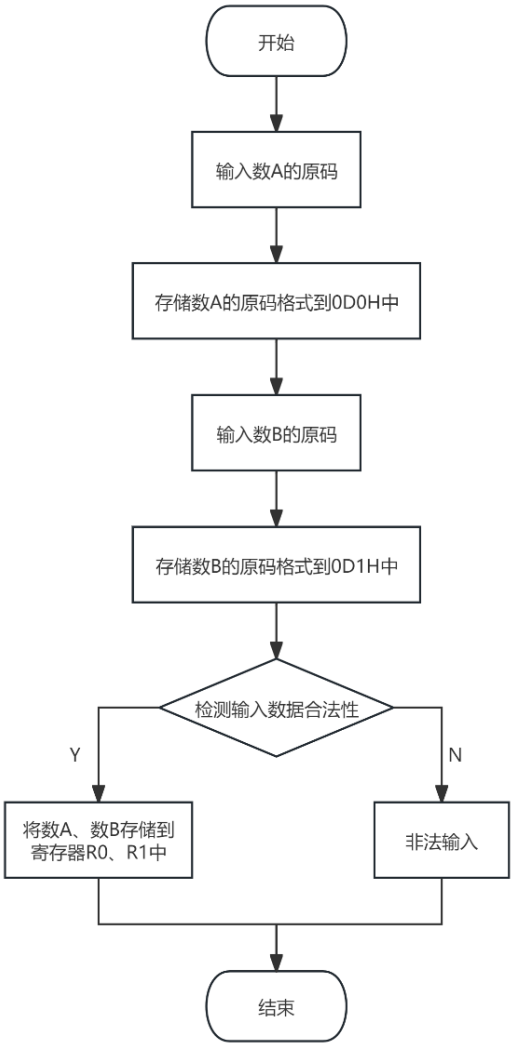


**图 2.3　输入原码数据模块实现流程图**

## 2.4 原码到补码转换模块

在进行加法运算之前，程序需要将输入的原码转换为补码。数值的补码表示对于负数而言是必要的，补码可以直接用于加法运算。在转换过程中，程序首先检查数 A 和数 B 的符号位（原码的次高位）。如果符号位为 0，则表示数为正数，原码和补码相同，直接将原码存储为补码；如果符号位为 1，则表示数为负数，程序会先对原码进行按位取反，再加 41H 得到补码，最后丢弃最高位，并将补码存储在 R2 和 R3 寄存器中。对于 0 值的数，直接赋值为补码 0。原码到补码转换模块的流程图如图 2.4 所示。

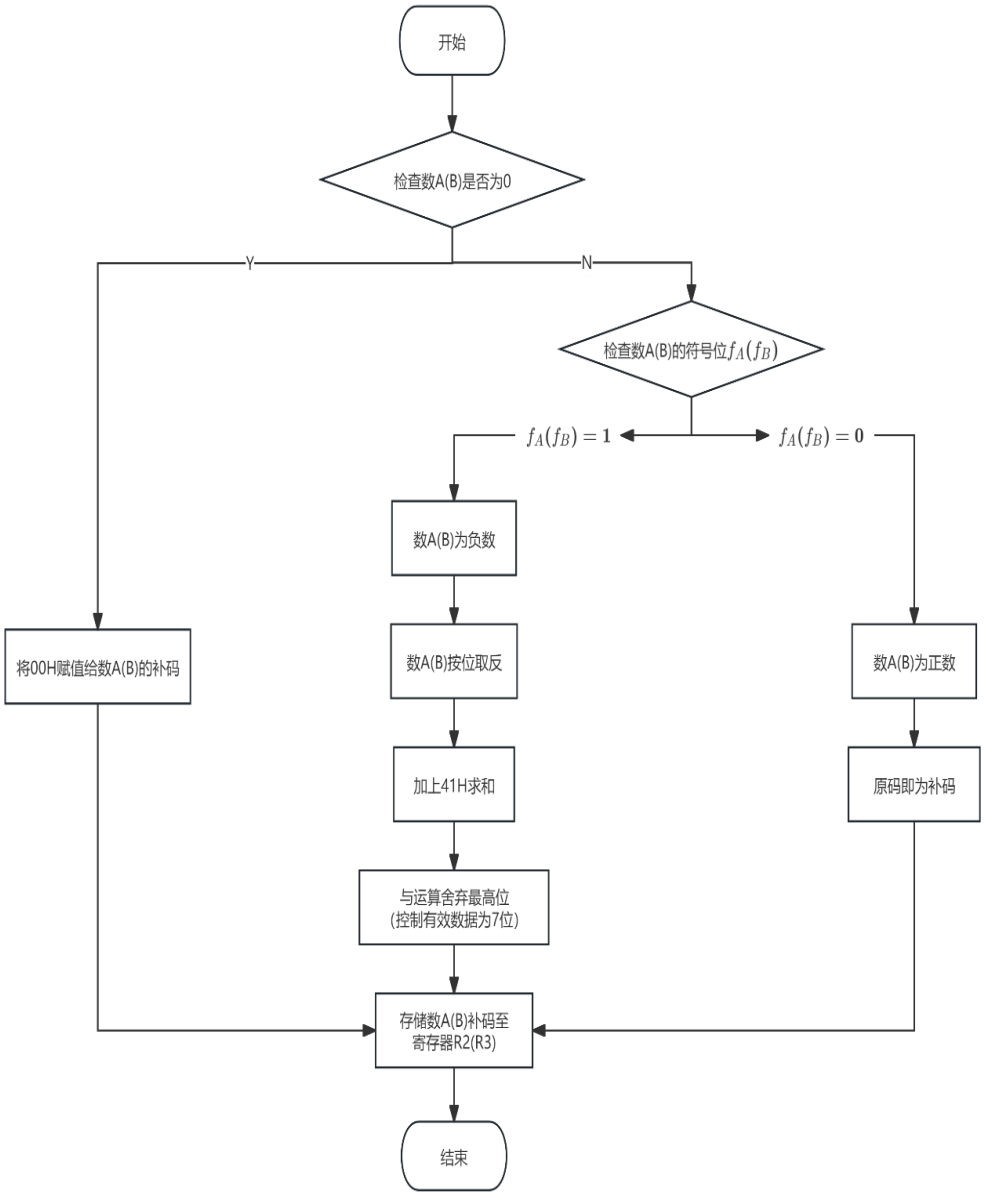


**图 2.4　原码到补码转换模块实现流程图**

## 2.5 补码加法运算模块

补码加法运算模块负责执行数 A 和数 B 的补码加法。加法过程使用了基本的 `ADD` 指令，它将数 A 和数 B 的补码加在一起，并将结果存储在内存单元 0D2H 中。加法过程中，程序需要考虑可能的进位，因此需要检查符号位的变化，确保加法结果正确。补码加法运算模块还包括对溢出检测的操作，通过分析符号位（即加法结果的最高位）是否发生了不符合预期的变化，来判断是否发生了上溢或下溢。

补码加法运算模块的流程图如图 2.5 所示。

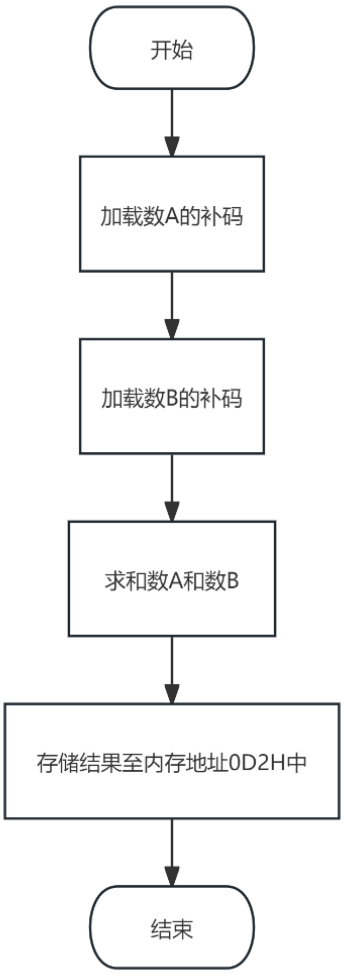


**图 2.5　补码加法运算模块实现流程图**

## 2.6 溢出检测模块

溢出检测模块负责检测加法运算结果是否发生溢出。由于加法操作涉及到符号位（有效数据最高位）的变化，程序需要对符号位进行详细的分析。通过提取数 A、数 B 及其加法结果的符号位，程序可以判断是否发生了溢出。

判断溢出公式为：$V = [(f_A f_B \overline{f_S}) + (\overline{f_A f_B} f_S)]$ 若求得 V= 1，则该运算结果溢出，反之 V＝0，则结果正常。其中，判断下溢公式为：$(f_A f_B \overline{f_S})$，若这三者与运算结果为 1，则该次补码加法运算结果下溢；判断上溢公式为：$(\overline{f_A f_B} f_S)$，若这三

者与运算结果为 1，则该次补码加法运算结果上溢。此外，“$f_A$”为补码格式下，数 A 的符号位；“$f_B$”为补码格式下，数 B 的符号位；“$f_S$”为补码格式下，加法运算结果的符号位。

具体而言，如果两个正数相加得到负数，则发生了上溢；如果两个负数相加得到正数，则发生了下溢。在此模块中，程序通过对符号位的运算（例如或操作、与操作、取反操作等）来判断是否发生溢出，进而执行相应的处理。如果检测到溢出，程序将根据溢出类型输出相应的溢出标志。溢出检测模块的流程图如图 2.6 所示。

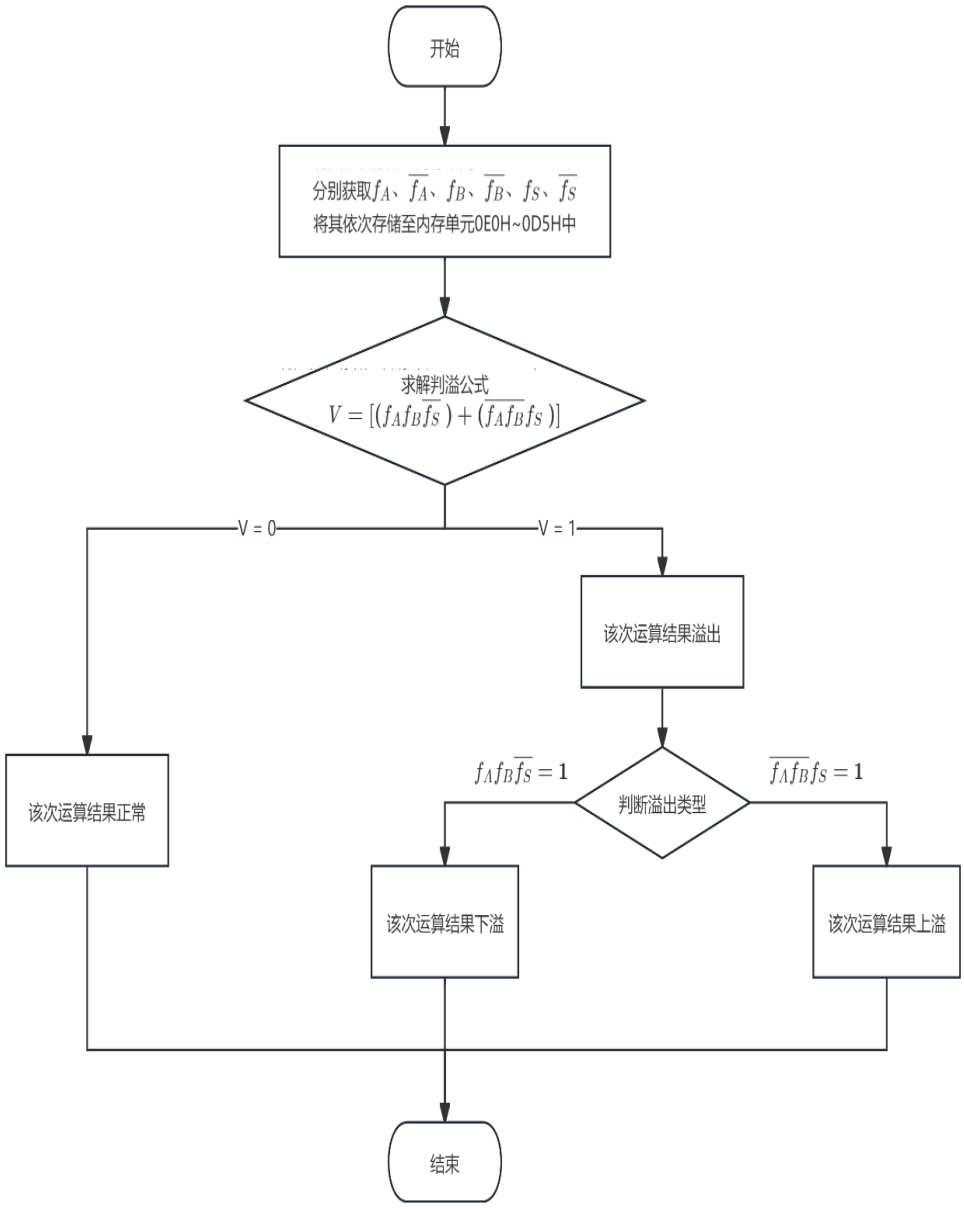


**图 2.6　溢出检测模块实现流程图**

## 2.7 结果输出模块

结果输出模块负责将加法结果或溢出标志输出到指定的内存单元或外部设备。输出的内容根据加法结果的符号位进行处理。如果加法结果是正数，直接输出补码形式的结果；如果加法结果是负数，程序将补码转换回原码（取反加 1）后输出。输出过程中，程序首先将补码转换为原码，然后将最终结果输出。对于溢出的情况，程序会根据溢出的类型输出相应的溢出标志（上溢或下溢）。结果输出模块的流程图如图 2.7 所示。

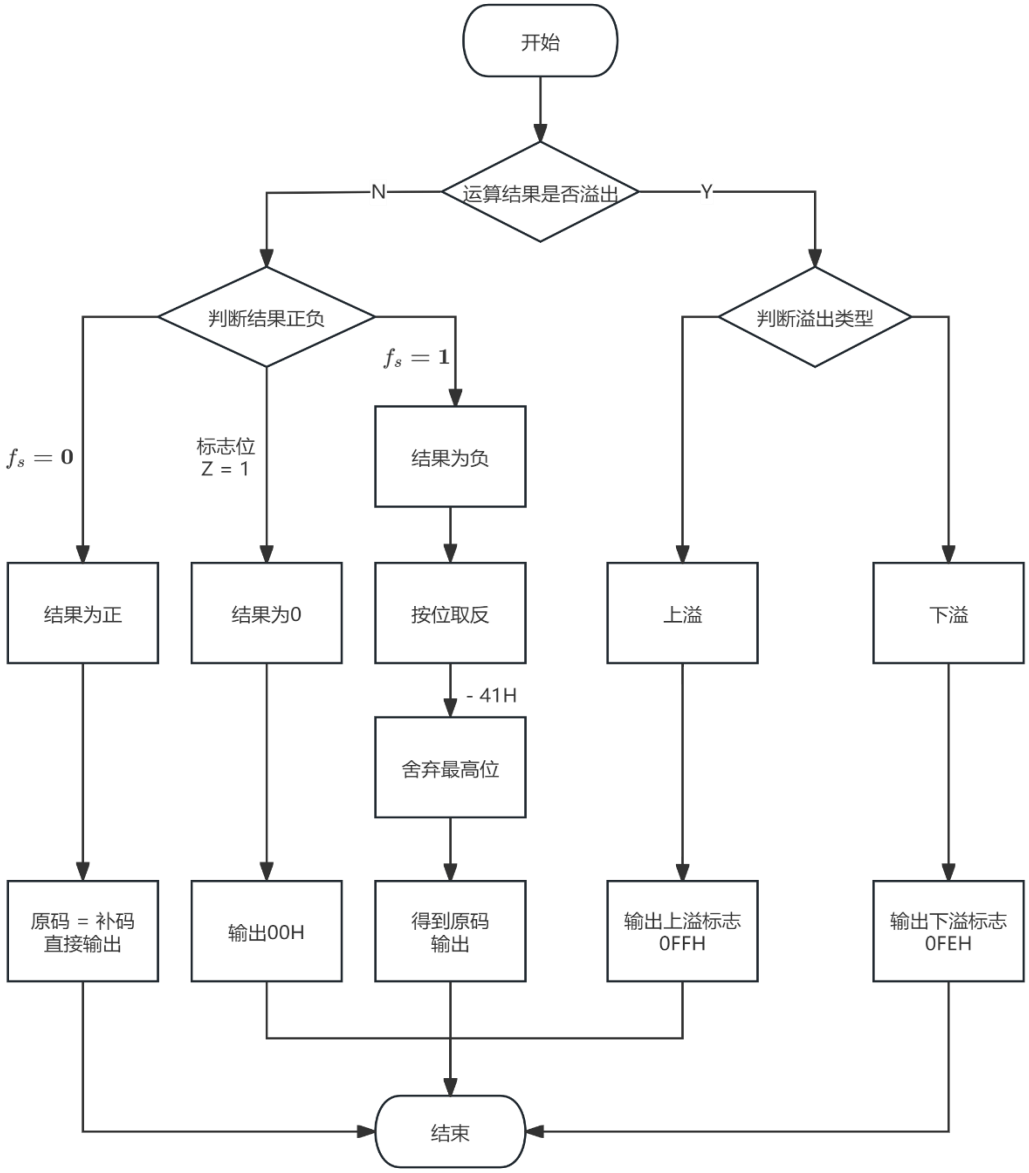


**图 2.7　结果输出模块实现流程图**

## 2.8 错误处理模块

错误处理模块负责处理程序在输入或执行过程中遇到的任何错误。如果输入数据无效，或者在运算过程中出现无法处理的异常，程序将跳转到错误处理模块。此模块会通过输出错误标志来提示用户，并使程序进入停止状态。错误处理模块确保了程序在遇到异常情况时能够安全终止，避免了不必要的错误继续执行。错误处理模块的流程图如图 2.8 所示。

通过这些模块的协同工作，程序能够实现对两个有符号数的补码加法运算，并正确地检测和处理溢出。每个模块都承担着特定的功能，通过模块化设计，程序结构清晰，功能明确，有助于后续的调试和维护。

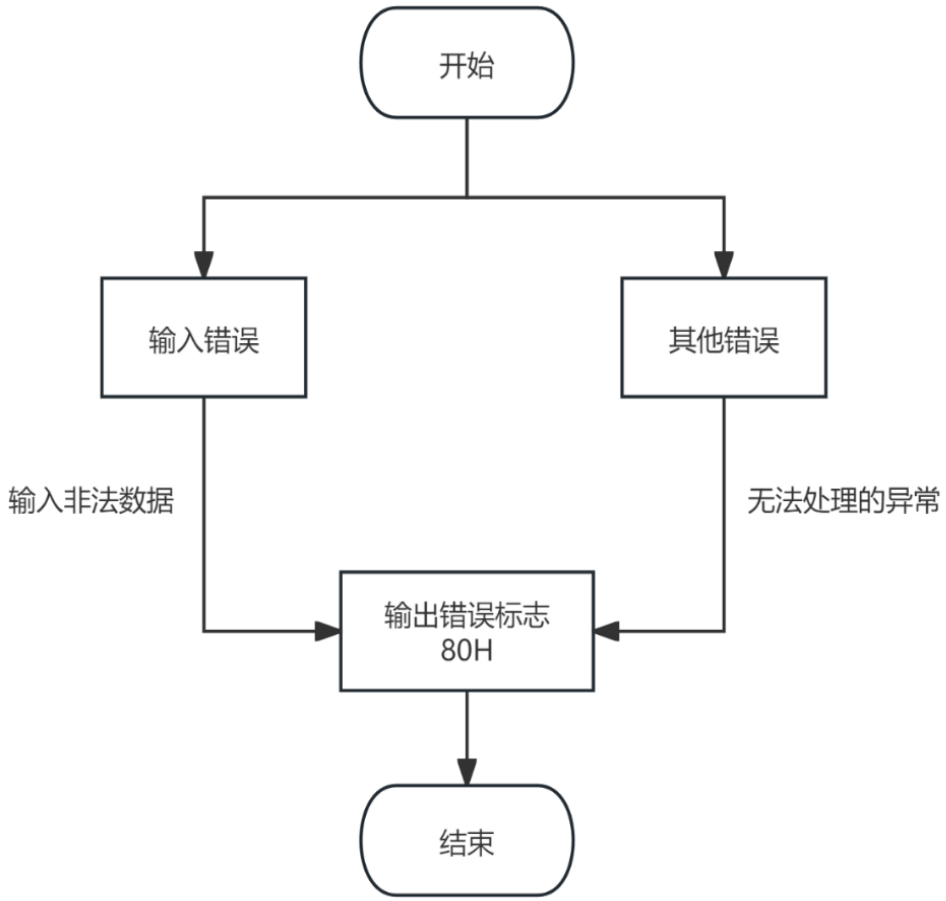


**图 2.8　错误处理模块实现流程图**

# 3 测试与验证

## 3.1 测试样例

本次课设测试样例为自定义输入测试数据，即需自行设置测试点，一组测试点需有两组输入测试数据：数 A、数 B，反复多次测试并包含边界，进行测试验证功能。

具体内容及手工计算实现码制转换和加法运算结果如表 3.1 所示。

**表 3.1 测试数据**

| 测试点 | | 数据合法 | | 原码→补码 | | 补码加法 | 结果溢出 | 补码→原码 | 输出 |
|---|---|---|---|---|---|---|---|---|---|
| 数A | 数B | 数A | 数B | 数A | 数B | | | | |
| 3CH | 8EH | ✓ | ✗ | | | | ✗ | ✗ | 80H |
| 94H | 0AH | ✗ | ✓ | | | | ✗ | ✗ | 80H |
| 0ACH | 80H | ✗ | ✗ | | | | ✗ | ✗ | 80H |
| 3CH | 0AH | ✓ | ✓ | | | | 上溢 | ✗ | 0FFH |
| 6EH | 59H | ✓ | ✓ | | | | 下溢 | ✗ | 0FEH |
| 1EH | 50H | ✓ | ✓ | 1EH | 70H | 8EH | | 0EH | 0EH |
| 6DH | 2BH | ✓ | ✓ | 53H | 2BH | 7EH | | 42H | 42H |
| 2CH | 0AH | ✓ | ✓ | 2CH | 0AH | 36H | | 36H | 36H |
| 4AH | 59H | ✓ | ✓ | 76H | 67H | 0DDH | | 63H | 63H |
| 00H | 00H | ✓ | ✓ | | | 00H | | | 00H |

其中输入非法测试为三组，输入合法测试且加法运算结果上溢为两组，输入合法测试且加法运算结果下溢为两组，输入合法测试且加法运算结果正常为五组。

## 3.2 测试过程

根据测试样例设计测试流程为通过执行输入测试数据与立即数 80H 与运算，检验输入数据合法性；通过执行输入测试数据与立即数 40H 与运算，判别数据的正负，从而针对性地实现原码到补码的转换；通过存储在内存地址 0E0H~0E5H 中的数据，辨别执行加法操作后结果是否溢出以及若溢出属于哪种溢出类型；通过执行运算结果与立即数 40H 与运算，判别数据的正负，为后续转换回原码操作奠定基础。

注意最高位的舍弃以及 0 的特殊性。

### 3.2.1 输入数据非法

首先，我们对程序在输入数据非法时的行为进行了验证。具体测试步骤为，手动输入超出可表示范围的数据到内存地址，并运行程序。预期结果是，程序能够正确识别非法输入并进行错误提示或处理，防止继续执行错误操作，确保系统的健壮性。

当输入数据为数 A 合法，数 B 非法时验证结果如图 3.1 所示；当输入数据为数 A 非法，数 B 合法时验证结果如图 3.2 所示；当输入数据为数 A，数 B 均非法时验证结果如图 3.3 所示。

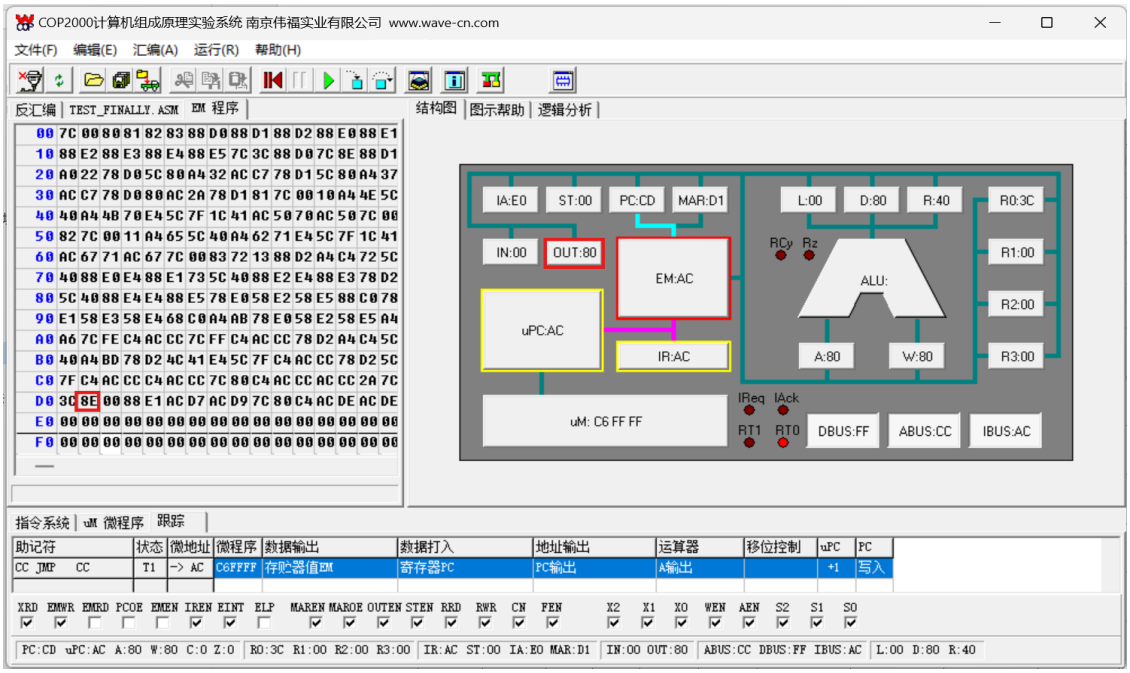


图 3.1 数 A 合法，数 B 非法验证

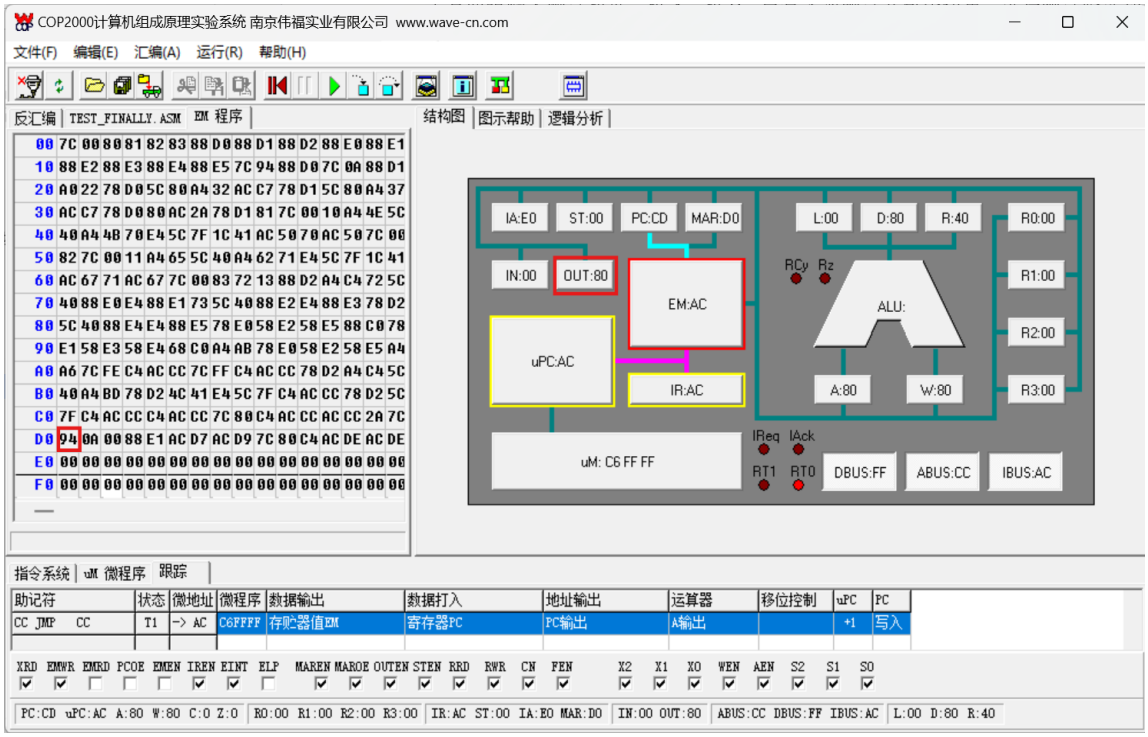


图 3.2　数 A 非法，数 B 合法验证

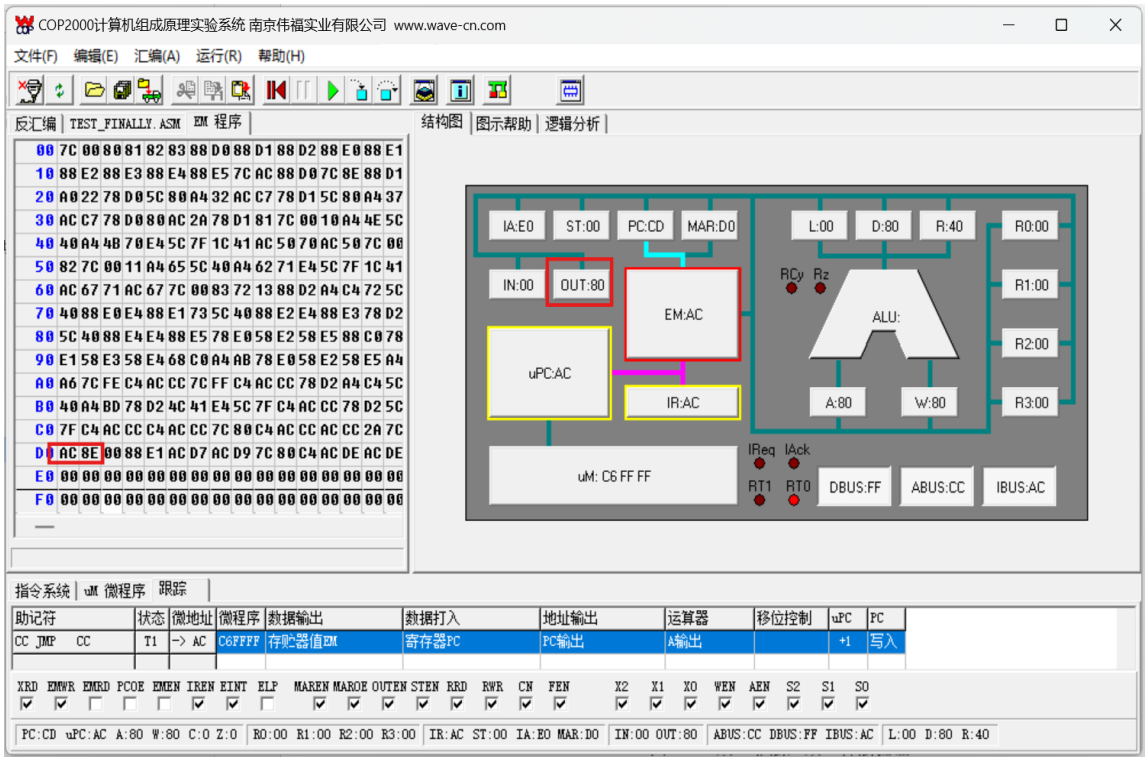


图 3.3　数 A，数 B 均非法验证

### 3.2.2 输入合法但结果上溢

为验证加法操作的溢出处理，我输入了两个正数 3CH（60）和 0AH（10）。这两个数相加后将超出 7 位的表示范围，结果为 70，即 46H。在执行加法时，程序应通过计算溢出条件：$\{V=[(f_Af_B\overline{f_S})+(\overline{f_A}\overline{f_B}f_S)]=1\}\cap\{(\overline{f_A}\overline{f_B}f_S)=1\}$，检测到溢出并判断溢出类型。$f_A$、$\overline{f_A}$、$f_B$、$\overline{f_B}$、$f_S$、$\overline{f_S}$（数 A 的符号位、数 A 的符号位的非、数 B 的符号位、数 B 的符号位的非、结果的符号位、结果的符号位的非）依次存储在内存单元 0E0H~0E5H 中。

验证结果如图 3.4 所示，0D0H~0D2H 分别存储数 A、数 B 的原码以及结果的补码；R0、R1 存储数 A、数 B 的原码；R2、R3 存储数 A、数 B 的补码；0E0H~0E5H 存储计算溢出条件涉及到的全部符号位及状态。

由此，程序能正确识别溢出并准确判断溢出类型，验证了系统在处理大于表示范围的加法时的准确性。

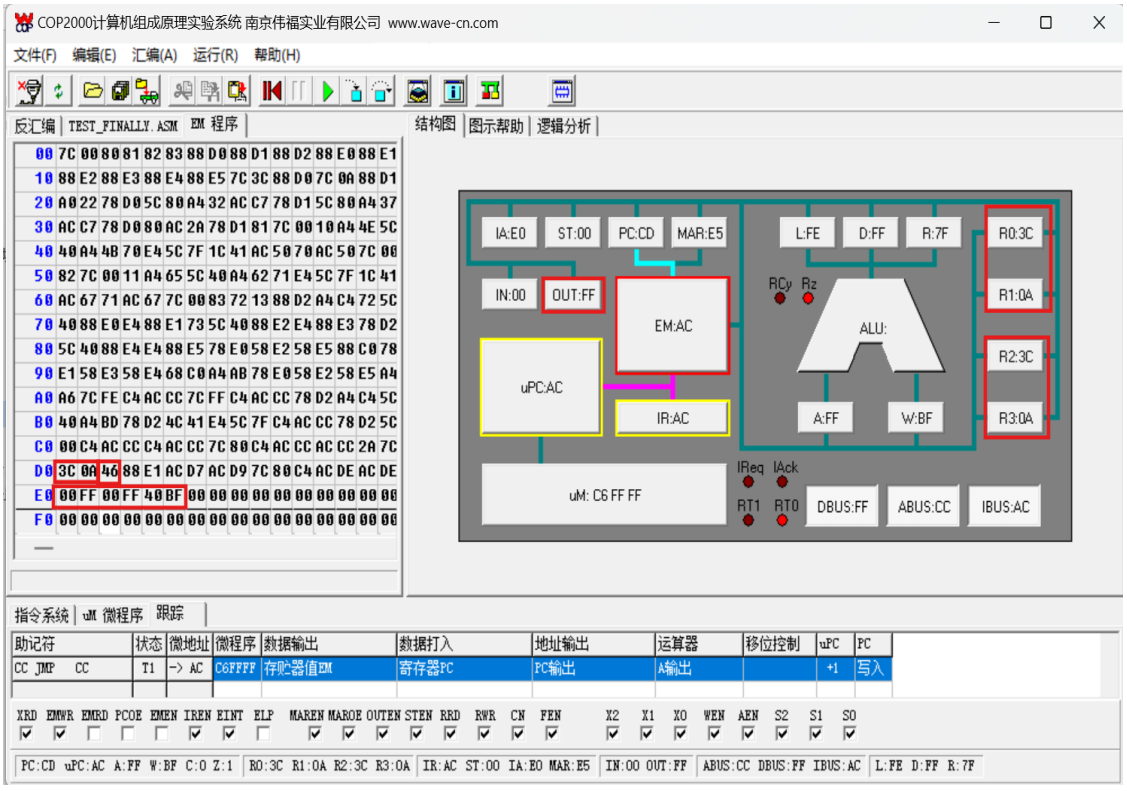


图 3.4　输入合法但结果上溢验证

### 3.2.3 输入合法但结果下溢

为验证加法操作的溢出处理，我输入了两个负数 6EH（-46）和 59H（-25）。这两个数相加后将超出 7 位的表示范围，结果为 -71，即 C7H。在执行加法时，程序应通过计算溢出条件：$\{V=[(f_Af_B\overline{f_S})+(\overline{f_A}\overline{f_B}f_S)]=1\}\cap\{(f_Af_B\overline{f_S})=1\}$，检测到溢出并判断溢出类型。$f_A$、$\overline{f_A}$、$f_B$、$\overline{f_B}$、$f_S$、$\overline{f_S}$（数 A 的符号位、数 A 的符号位的非、数 B 的符号位、数 B 的符号位的非、结果的符号位、结果的符号位的非）依次存储在内存单元 0E0H~0E5H 中。

验证结果如图 3.5 所示，0D0H~0D2H 分别存储数 A、数 B 的原码以及结果的补码；R0、R1 存储数 A、数 B 的原码；R2、R3 存储数 A、数 B 的补码；0E0H~0E5H 存储计算溢出条件涉及到的全部符号位及状态。

由此，程序能正确识别溢出并准确判断溢出类型，验证了系统在处理大于表示范围的加法时的准确性。

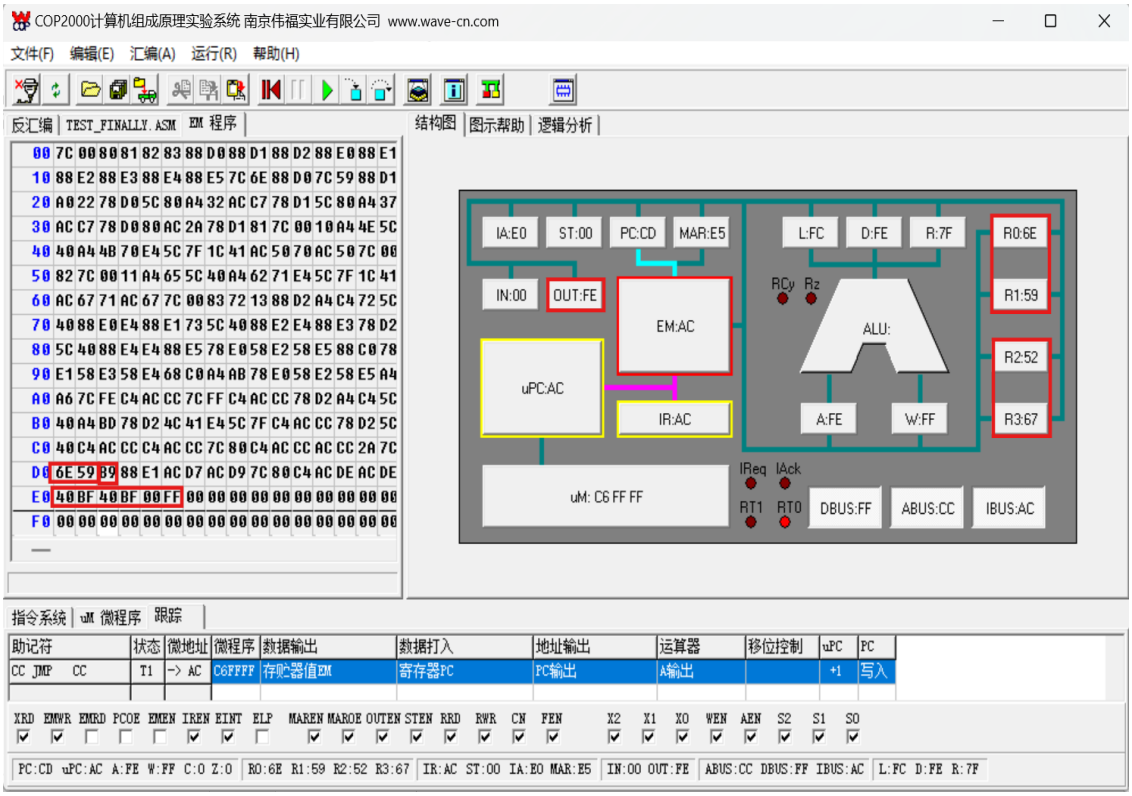


图 3.5　输入合法但结果下溢验证

### 3.2.4 输入与结果均合法

当验证输入与结果均合法情况时，我将其分为了四个状态，分别为：两正数相加、两负数相加、数 A 为正且数 B 为负、数 A 为负且数 B 为正，进行验证。同样的，在执行加法时，程序应通过计算溢出条件： $V=[(f_Af_B\overline{f_S})+(\overline{f_Af_B}f_S)]$，判断该次验证结果是否溢出。若求得 V=0，则该次验证结果有效，继续下一步骤实现：补码转换回原码。转换时，根据运算结果的符号位（次高位）认定结果的正负，进而跳转相应的转换程序，正确获得结果的原码格式。

两正数相加、两负数相加、数 A 为正且数 B 为负、数 A 为负且数 B 为正,四种状态的验证结果依次如图 3.6~图 3.9 所示。其中，0D0H~0D2H 分别存储数 A、数 B 的原码以及结果的补码；R0、R1 存储数 A、数 B 的原码；R2、R3 存储数 A、数 B 的补码；0E0H~0E5H 存储计算溢出条件涉及到的全部符号位及状态。

由此，程序在同正、同负及正负数混合加法中也能稳定运行，确保了加法运算的准确性和程序的稳定性。

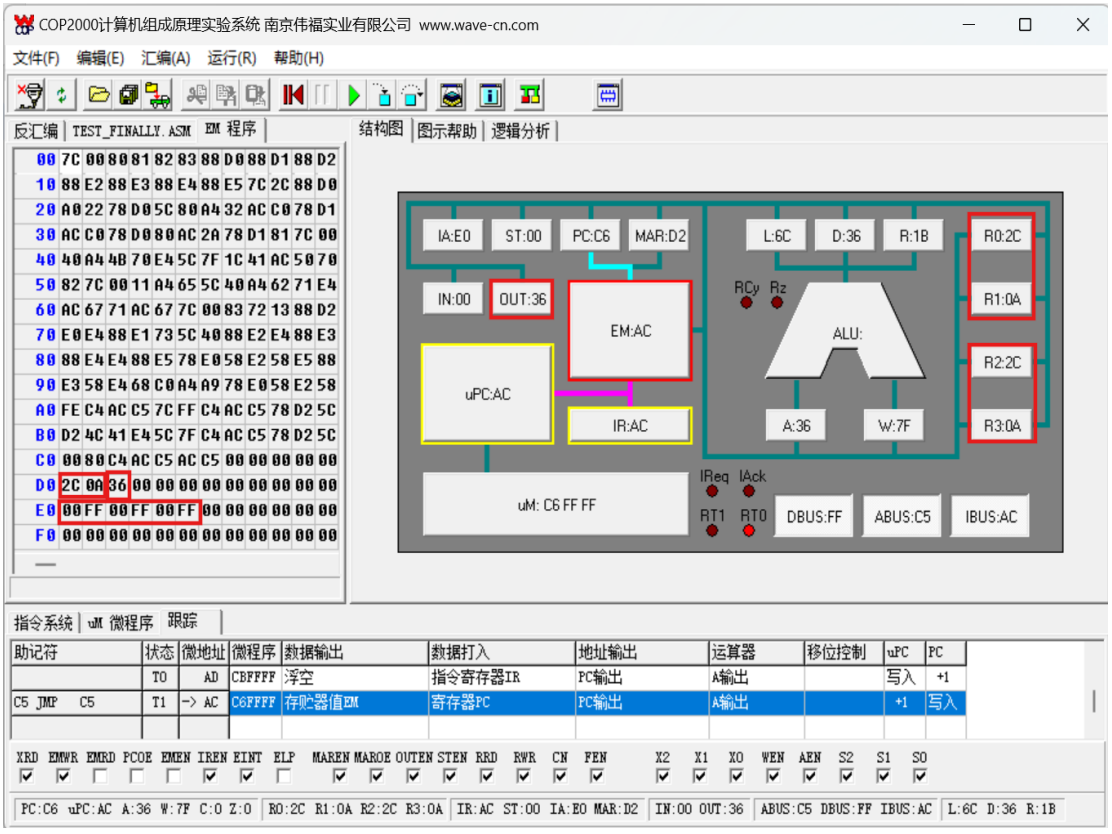


图 3.6　两正数相加验证

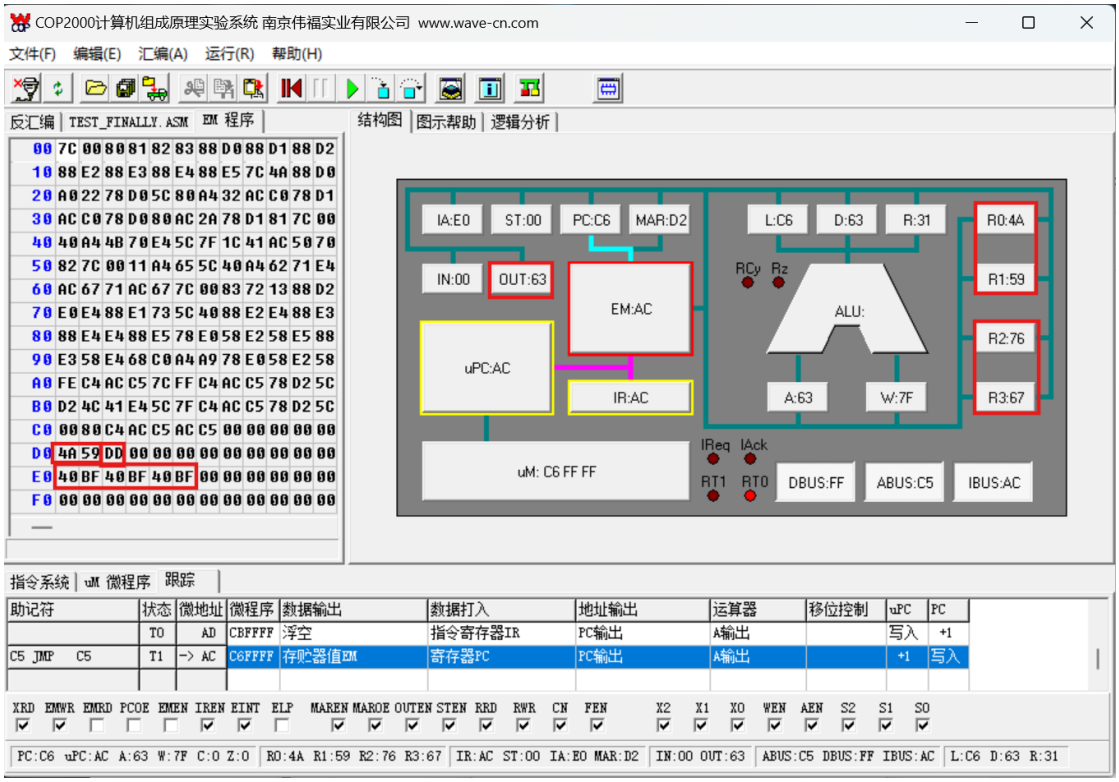


图 3.7　两负数相加验证

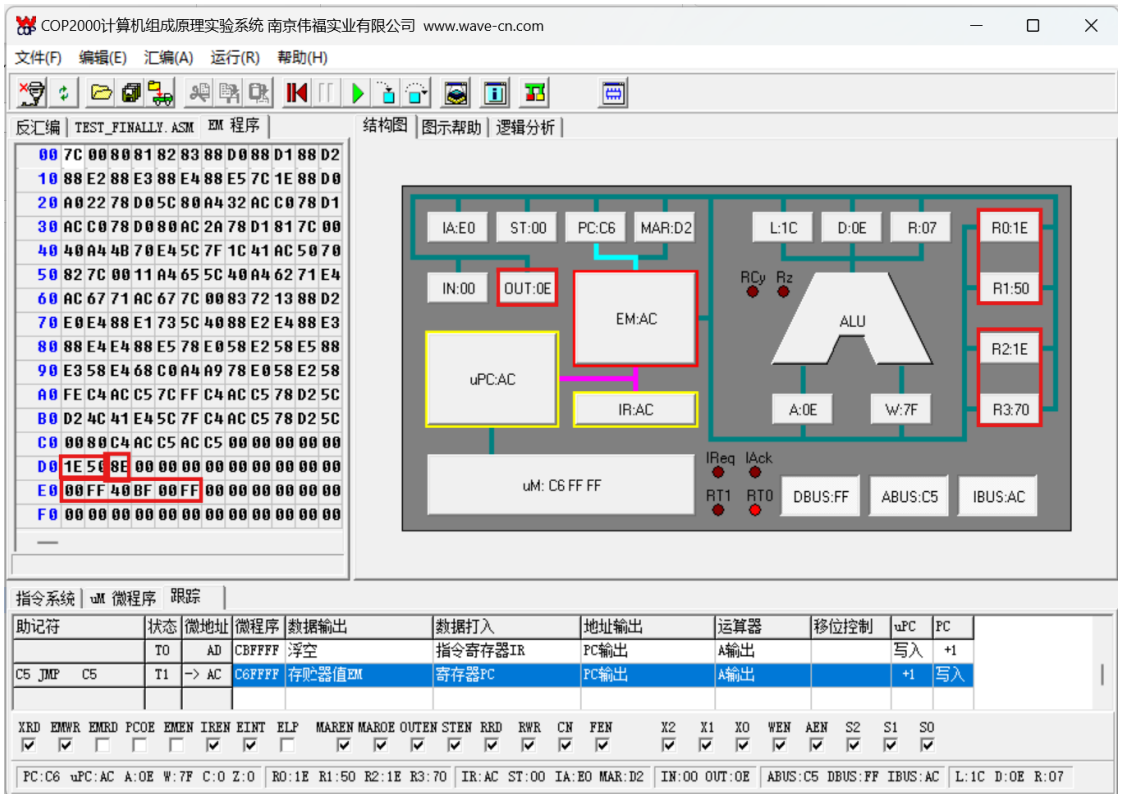


图 3.8　数 A 为正，数 B 为负求和验证

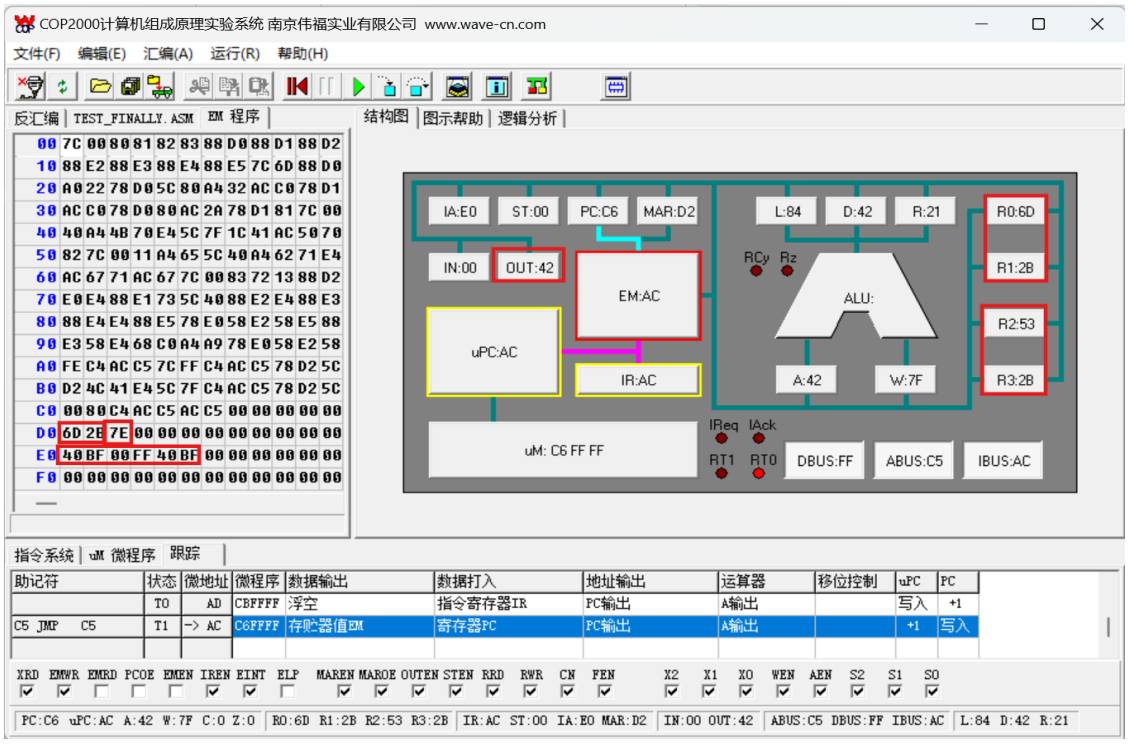


**图 3.9　数 A 为负，数 B 为正求和验证**

### 3.2.5 输入数据存在“0”项

当检测到输入测试数据数 A 和数 B 均为 0 时，不需要区分符号位是 0 还是 1，也不需要执行溢出条件的计算（即无需考虑 V 标志位）。直接将结果赋值为 00H，并按照加法逻辑进行计算即可。

当检测到输入数据数 A 或数 B 中某一项为“0”项，而另一项为非零项时，仅需对非零项需要区分符号位是 0 还是 1，在输入数据合法的前提下，无需执行判断溢出条件的计算（即无需考虑 V 标志位）。直接将结果赋值为非零项的符号位和绝对值部分。

输入测试数据数 A 和数 B 均为“0”时，验证结果如图 3.10 所示；输入测试数据数 A 为“0”、数 B 为非零项时，验证结果如图 3.11 所示。

由此，出于对“0”的特殊性的考虑，抽象了输入测试数据为“0”项的处理逻辑：由于检测零标志位 Z = 1，无需进行原，补码转换，直接对其赋值 00H，且由于检测零标志位 Z = 1，无需判溢，直接输出 00H，使得程序在输入数据存在“0”项时能稳定运行。

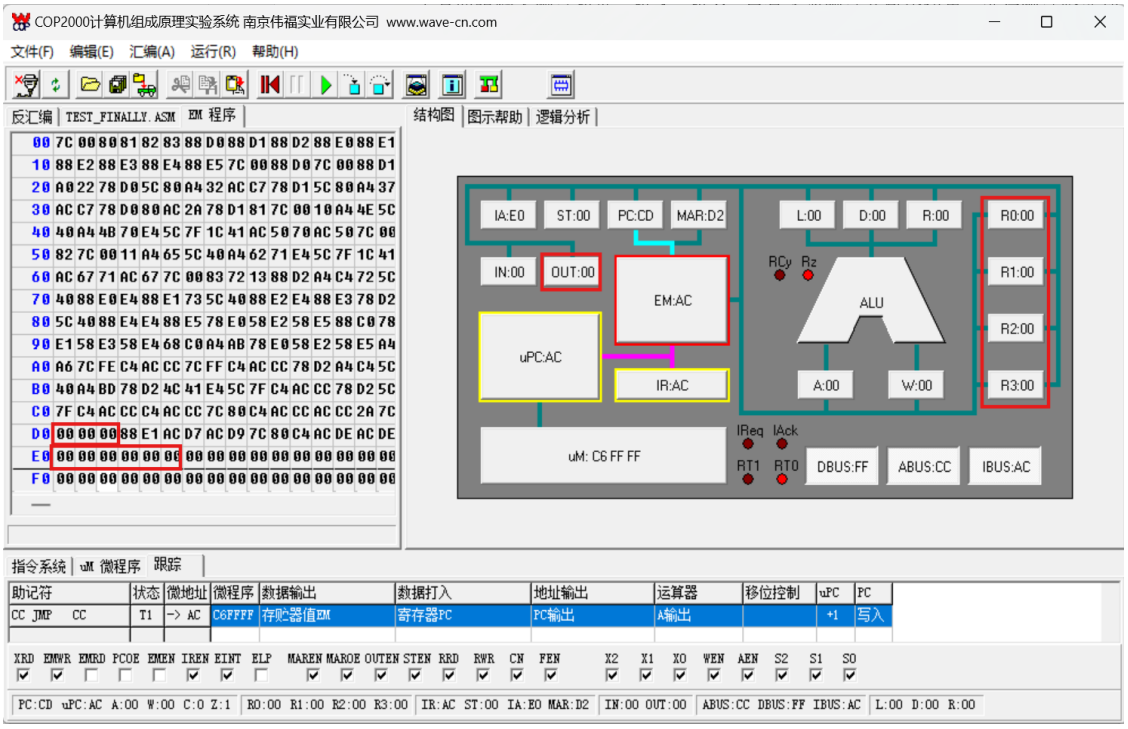


图 3.10　输入数据均为“0”验证

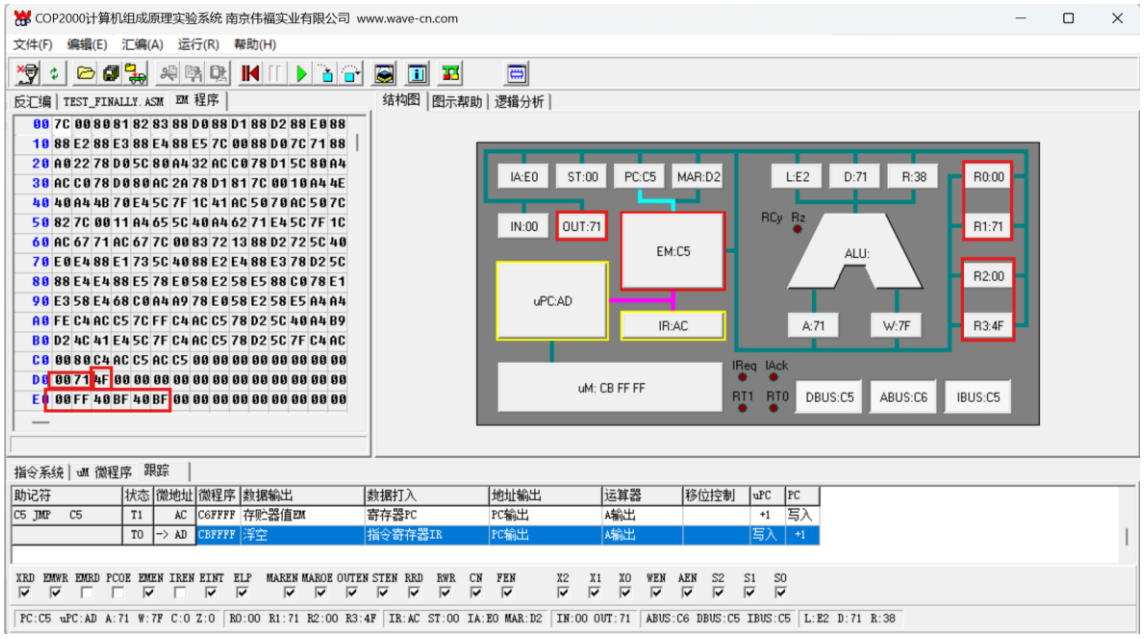


图 3.10　输入数据“0”项和非零项同时存在验证

# 参考文献

[1] 唐朔飞.计算机组成原理（第 2 版）[M]．北京：高等教育出版社，2008

[2] 范延滨.微型计算机系统原理、接口与 EDA 设计技术[M].北京：北京邮电大学出版社，2006

[3] 王爱英.计算机组成与结构(第 4 版)[M].北京：清华大学出版社，2006

[4] 董涛. 浅析计算机数值编码中的原码, 反码与补码[J]. 数字技术与应用, 2011 (1): 118-119.

[5] 孙静霞.浅谈计算机中数的原码, 补码, 反码 (以下所讨论的均为整数, 二进制整数均以 8 位为例)[J]. 科技资讯, 2009 (5): 34-34.

[6]魏益堂.负数补码的探究[J].电子测试,2013,(12):40-41.

[7]贾成伟,李海燕.定点补码加法运算公式的缜密证明[J].黑龙江农垦师专学报,2002,(02):77-78.

[8]王学峰.永恒的加法运算[J].赤峰教育学院学报,2000,(01):52-53.

# 附　录

| 程序地址 | 机器码 | 汇编指令 | 指令说明 |
|---|---|---|---|
| 00 | 7C00 | MOV　A, #00 | 初始化 |
| 02 | 80 | MOV　R0, A | 初始化 R0（数 A 的原码存储） |
| 03 | 81 | MOV　R1, A | 初始化 R1（数 B 的原码存储） |
| 04 | 82 | MOV　R2, A | 初始化 R2（数 A 的补码存储） |
| 05 | 83 | MOV　R3, A | 初始化 R3（数 B 的补码存储） |
| 06 | 88D0 | MOV　D0, A | 初始化内存存储地址 0D0H |
| 08 | 88D1 | MOV　D1, A | 初始化内存存储地址 0D1H |
| 0A | 88D2 | MOV　D2, A | 初始化内存存储地址 0D2H |
| 0C | 88E0 | MOV　E0, A | 数 A 次高位存储 |
| 0E | 88E1 | MOV　E1, A | 数 A 次高位取反存储 |
| 10 | 88E2 | MOV　E2, A | 数 B 次高位存储 |
| 12 | 88E3 | MOV　E3, A | 数 B 次高位取反存储 |
| 14 | 88E4 | MOV　E4, A | 运算结果次高位存储 |
| 16 | 88E5 | MOV　E5, A | 运算结果次高位取反存储 |
| 18 | 7C1E | MOV　A, #1E | 输入数 A 的原码 |
| 1A | 88D0 | MOV　D0, A | 存储到数 A 地址 |
| 1C | 7C50 | MOV　A, #50 | 输入数 B 的原码 |
| 1E | 88D1 | MOV　D1, A | 存储到数 B 地址 |
| 20 | A022 | JC　22 | 跳转数 A 合法性检查 |
| 22 | 78D0 | MOV　A, D0 | 加载数 A |
| 24 | 5C80 | AND　A, #80 | 确保合法性检查(7 位范围内) |
| 26 | A432 | JZ　32 | 输入正常跳转存储 |
| 28 | ACC0 | JMP　C0 | 输入错误跳转 |
| 2A | 78D1 | MOV　A, D1 | 加载数 B |
| 2C | 5C80 | AND　A, #80 | 确保合法性检查(7 位范围内) |
| 2E | A437 | JZ　37 | 输入正常跳转存储 |

| | | | |
|---|---|---|---|
| 30 | ACC0 | JMP C0 | 输入错误跳转 |
| 32 | 78D0 | MOV A, D0 | 数 A 为 7 位有效原码，读取数 A |
| 34 | 80 | MOV R0, A | 保存至 R0 |
| 35 | AC2A | JMP 2A | 跳转数 B 合法性检查 |
| 37 | 78D1 | MOV A, D1 | 数 B 为 7 位有效原码，读取数 B |
| 39 | 81 | MOV R1, A | 保存至 R1 |
| 3A | 7C00 | MOV A, #00 | 清零标志位 |
| 3C | 10 | ADD A, R0 | 加载数 A 原码 |
| 3D | A44E | JZ 4E | 如果数 A 为 0，跳过转换 |
| 3F | 5C40 | AND A, #40 | 检查符号位（次高位） |
| 41 | A44B | JZ 4B | 如果为正数，跳到正数处理 |
| 43 | 70 | MOV A, R0 | 加载数 A 原码 |
| 44 | E4 | CPL A | 如果为负数，取反 |
| 45 | 5C7F | AND A, #7F | 最高位丢弃 |
| 47 | 1C41 | ADD A, #41 | 获取补码 |
| 49 | AC50 | JMP 50 | 跳转数 A 补码保存 |
| 4B | 70 | MOV A, R0 | 直接将原码赋值为补码 |
| 4C | AC50 | JMP 50 | 跳转数 A 补码保存 |
| 4E | 7C00 | MOV A, #00 | 数 A 为 0 时，补码赋值为 0 |
| 50 | 82 | MOV R2, A | 将数 A 的补码存储到 R2 |
| 51 | 7C00 | MOV A, #00 | 清零标志位 |
| 53 | 11 | ADD A, R1 | 加载数 B 原码 |
| 54 | A465 | JZ 65 | 如果数 B 为 0，跳过转换 |
| 56 | 5C40 | AND A, #40 | 检查符号位（次高位） |
| 58 | A462 | JZ 62 | 如果为正数，跳到正数处理 |
| 5A | 71 | MOV A, R1 | 加载数 B 原码 |
| 5B | E4 | CPL A | 如果为负数，取反 |
| 5C | 5C7F | AND A, #7F | 最高位丢弃 |
| 5E | 1C41 | ADD A, #41 | 获取补码 |
| 60 | AC67 | JMP 67 | 跳转数 B 补码保存 |
| 62 | 71 | MOV A, R1 | 直接将原码赋值为补码 |
| 63 | AC67 | JMP 67 | 跳转数 B 补码保存 |

| | | | |
|---|---|---|---|
| 65 | 7C00 | MOV A, #00 | 数 B 为 0 时，补码赋值为 0 |
| 67 | 83 | MOV R3, A | 将数 B 的补码存储到 R3 |
| 68 | 72 | MOV A, R2 | 加载数 A 补码 |
| 69 | 13 | ADD A, R3 | 加数 B 补码求和 |
| 6A | 88D2 | MOV D2, A | 暂存结果 |
| 6C | 72 | MOV A, R2 | 获取数 A 补码的次高位 |
| 6D | 5C40 | AND A, #40 | 提取次高位 |
| 6F | 88E0 | MOV E0, A | 存储次高位 |
| 71 | E4 | CPL A | 取反 |
| 72 | 88E1 | MOV E1, A | 存储次高位取反 |
| 74 | 73 | MOV A, R3 | 获取数 A 补码的次高位 |
| 75 | 5C40 | AND A, #40 | 提取次高位 |
| 77 | 88E2 | MOV E2, A | 存储次高位 |
| 79 | E4 | CPL A | 取反 |
| 7A | 88E3 | MOV E3, A | 存储次高位取反 |
| 7C | 78D2 | MOV A, D2 | 获取运算结果的补码的次高位 |
| 7E | 5C40 | AND A, #40 | 提取次高位 |
| 80 | 88E4 | MOV E4, A | 存储次高位 |
| 82 | E4 | CPL A | 取反 |
| 83 | 88E5 | MOV E5, A | 存储次高位取反 |
| 85 | 78E0 | MOV A, E0 | 数 A 次高位 |
| 87 | 58E2 | AND A, E2 | 与数 B 次高位 |
| 89 | 58E5 | AND A, E5 | 与加法结果次高位取反 |
| 8B | 88C0 | MOV C0, A | 暂存中间结果 1 |
| 8D | 78E1 | MOV A, E1 | 数 A 次高位取反 |
| 8F | 58E3 | AND A, E3 | 与数 B 次高位取反 |
| 91 | 58E4 | AND A, E4 | 与加法结果次高位 |
| 93 | 68C0 | 0R A, C0 | 合并两个结果 |
| 95 | A4A9 | JZ A9 | 如果结果为 0，不溢出 |
| 97 | 78E0 | MOV A, E0 | 数 A 次高位 |
| 99 | 58E2 | AND A, E2 | 与数 B 次高位 |
| 9B | 58E5 | AND A, E5 | 与加法结果次高位取反 |

| | | | |
|---|---|---|---|
| 9D | A4A4 | JZ A4 | 如果为 0，跳到上溢处理 |
| 9F | 7CFE | MOV A, #FE | 下溢标志 |
| A1 | C4 | OUT | 输出 |
| A2 | ACC5 | JMP C5 | 程序结束，在此处原地循环 |
| A4 | 7CFF | MOV A, #FF | 上溢标志 |
| A6 | C4 | OUT | 输出 |
| A7 | ACC5 | JMP C5 | 程序结束，在此处原地循环 |
| A9 | 78D2 | MOV A, D2 | 获取结果 |
| AB | 5C40 | AND A, #40 | 检查符号位（次高位） |
| AD | A4B9 | JZ B9 | 如果为正数，跳到正数处理 |
| AF | 78D2 | MOV A, D2 | 获取结果 |
| B1 | 4C41 | SUBC A, #41 | 转换回原码 |
| B3 | E4 | CPL A | |
| B4 | 5C7F | AND A, #7F | 最高位丢弃 |
| B6 | C4 | OUT | 输出 |
| B7 | ACC5 | JMP C5 | 程序结束，在此处原地循环 |
| B9 | 78D0 | MOV A, D2 | 直接将原码赋值为补码 |
| BB | 5C7F | AND A, #7F | 最高位丢弃 |
| BD | C4 | OUT | 输出 |
| BE | ACC5 | JMP C5 | 程序结束，在此处原地循环 |