

数字逻辑与处理器基础

——汇编大作业

班级：无 26

学号：2022010645

姓名：李卓航

日期：2024 年 5 月 4 日

一、作业概述

在 MARS 模拟器上,将指定的 C/C++代码手动编译成 MIPS 汇编指令,然后汇编,运行,调试,完成能够解决给定问题的汇编程序设计。本次作业的目的在于理解汇编语言如何完成高级语言描述的算法,了解 MIPS 处理器的硬件结构如何实现指令的需求,同时学会如何编写调试汇编程序。

二、基础练习部分

(1) exp1_1

1.程序功能:

- (1)申请一个 8 byte 整数的内存空间。
- (2)从“a.in”读取两个整数到 buffer 数组。
- (3)向”a.out”写入 buffer 的两个整数。
- (4)输入一个整数 i,执行 $i = i + 10$, 并打印结果。

2.结果: 如图, 成功从 a.in 读入了两个整数到 buffer

Address	Value (+0)	Value (+4)
0x10010000	0x00000001	0x0000000a

如下图, 输入 5 之后生成了 15 并打印出来

```
5
15
— program is finished running (dropped off bottom) —
```

(2) exp1_2

1.程序功能:

- (1)将输入值 i 取相反数, 将输入值 j 取绝对值。(j 为非 0 整数)
 - (2)从变量 i 开始, 循环 j 轮, 每轮 $i = i + 1$
- 即计算 $-i + |j|$ 的值

2.结果: 如图, 输入 3、7 和 3、-7 进行测试, 均得到了 $7-3=4$ 的正确结果, 储存在了 v0 中

```
3
-7
4
— program is finished running (dropped off bottom) —

3
7
4
— program is finished running (dropped off bottom) —
```

zero	0	0x00000000
at	1	0x00000000
v0	2	0x00000004
v1	3	0x00000000
a0	4	0x00000004
a1	5	0x00000000
...

(3) exp1_3

1.程序功能:

- (1) 输入数组 a 的长度 n (保证 n 为偶数) 和任意 n 个整数
(2) 将数组 a 中的元素逐个加 1, 并且逆序存储在 a 中, 最后打印数组 a 的值 (中间不空)

2.结果: 如图, 输入 6、1、9、4、5、2、3 进行测试, 得到了 4365102 的正确结果

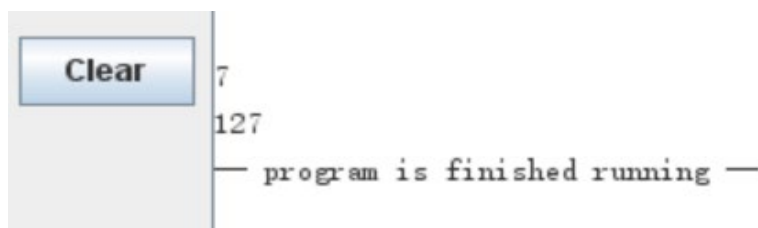


(4) exp1_4

1.程序功能:

输入一个整数 n , 通过函数递归调用计算 $Hanoi(n)$ 的值并打印输出

2.结果: 如图, 输入 7 进行测试, 得到了 127 ($2^7 - 1$) 的正确结果



三、实战应用部分

(1) insert_sort

1. 寄存器和变量之间的分配关系：

\$a0、\$v0 负责存储系统调用中的信息与\$a1、\$a2 完成文件打开关闭与读写；

\$s0 存储 buffer 数组首地址，\$s1 存储 N，\$s2 存储总比较次数，\$s3 存储 search 函数返回值，\$s4 存储 search 函数 temp 值，其在 insert 函数里作用类似；

\$t0 一般用于存储 insertion_sort 函数 for 循环的循环变量 i，\$t2 存储 search 函数返回值 place 在 insertion_sort 函数中调用，\$t4 存储 search 函数循环变量 i，\$t5 存储 search 函数 v[i]值，其在 insert 函数里作用类似，\$t6 负责在 insert 函数里存储 v[i+1]地址用于 sw 操作。

其余还涉及到一些\$sp 栈操作或是小循环处\$t 的调用，因为细节太多不具体一一罗列，详情见代码中注释。

2. 过程调用操作：

首先通过 main 函数进行文件读入存储到 buffer 数组中，接着加载 N 到\$s1，以 buffer[1] 作为数组首地址传参到 insert_sort 中，并在 insert_sort 中直接修改 buffer 元素的值，最终将比较次数存入 buffer[0]，写入到 out 文件里并关闭文件。

在 insert_sort 函数中，进入 for 循环，先调用 search 函数找到 i 前面所有元素中第一个比 v[i]小的位置，接着调用 insert 函数插入到该位置并将其后所有元素后移一位，完成排序；

在 search 中使用 for 循环从后往前查找第一个比 v[n]小的元素，查到则跳出循环，记录此时的位置，返回其一位并返回 insert_sort 函数作为 place；在 insert 中先将 v[i]值存储在 temp 中，接着依次该位置前面的值，将前面的值写入到后一位的地址中，直到 place 指定位置，完成元素后移，再将 place 位置的值加载为 temp，完成 i 在 place 位置的插入，返回 insert_sort 进行下一个 i 的操作。

3. 运行结果：

使用给定的 a.in 文件，加载其中 N 和 N 个待排序的数到 buffer，结果如下：

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000014	0x000041a8	0x00003af2	0x0000acda	0x0000e2b	0x0000b783	0x00004ac9	0x00008e49
0x10010004	0x000000ff	0x00002f44	0x0000044e	0x00009899	0x00003e56	0x00001284	0x00004b43	0x000044b4
0x10010008	0x00003748	0x00003918	0x00004112	0x0000c399	0x00004955	0x00000000	0x00000000	0x00000000
0x1001000c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001001c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001002c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001003c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

排序完以后，将比较次数和排好后数的写入 a.out 文件，a.out 用 mars 打开的结果如下：

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000067	0x0000044e	0x000009ff	0x00000c2b	0x00001284	0x00002f44	0x00003748	0x00003918
0x10010020	0x0000af2	0x0000c56	0x00004112	0x000041a0	0x00004955	0x00008ad9	0x00009899	0x0000aceda
0x10010040	0x0000b783	0x0000c399	0x0000d4b4	0x0000dac9	0x0000dbe3	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

第 1 位 0x67=103，为排序次数，与对应 cpp 文件运行结果一致，而且发现后面的数均呈正序排列，可见完成了排序，且总体流程执行过程与 cpp 一致

(2) binary_insert_sort

1. 寄存器和变量之间的分配关系：

\$a0、\$v0 负责存储系统调用中的信息与\$a1、\$a2 完成文件打开关闭与读写；

\$s0 存储 buffer 数组首地址，\$s1 存储 N，\$s2 存储总比较次数，\$s3 存储 binary_search 函数返回值，\$s4 存储 binary_search 函数 temp 值，其在 insert 函数里作用类似，\$s5、\$s6、\$s7 分别存储 binary_search 中 left、right 和 mid 指针；

\$t0 一般用于存储 binary_insertion_sort 函数 for 循环的循环变量 i，\$t2 存储 binary_search 函数返回值 place 在 binary_insertion_sort 函数中调用，\$t4 存储 binary_search 和 insert 函数里形参 n，\$t5 存储 binary_search 函数 v[i]值，其在 insert 函数里作用类似，\$t6 负责在 insert 函数里存储 v[i+1]地址用于 sw 操作。

其余还涉及到一些\$sp 栈操作或是小循环处\$t 的调用，因为细节太多不具体一一罗列，详情见代码中注释。

2. 过程调用操作：

首先通过 main 函数进行文件读入存储到 buffer 数组中，接着加载 N 到\$s1，以 buffer[1] 作为数组首地址传参到 binary_insert_sort 中，并在 binary_insert_sort 中直接修改 buffer 元素的值，最终将比较次数存入 buffer[0]，写入到 out 文件里并关闭文件。

在 binary_insert_sort 函数中，进入 for 循环，先调用 binary_search 函数找到 i 前面所有元素中第一个比 v[i]小的位置，接着调用 insert 函数插入到该位置并将其后所有元素后移一位，完成排序；

在 binary_search 函数中，先判断左右指针大小，如果左指针大则处于递归边界跳出循环进入 binary_search_end 返回左指针到\$s3 中，否则取左右指针中间值作为 mid，比较 v[mid] 和当前位置 v[i]大小，如果大于则二分查找左边，进入 binary_left，左指针不变，mid-1 作为新的右指针再递归调用 binary_search，否则二分查找右边，进入 binary_right，右指针不变，mid+1 作为新的左指针再递归调用 binary_search，最终找到指定位置返回 binary_insert_sort。

在 insert 中先将 v[i]值存储在 temp 中，接着依次该位置前面的值，将前面的值写入到后一位的地址中，直到 place 指定位置，完成元素后移，再将 place 位置的值加载为 temp，完成 i 在 place 位置的插入，返回 binary_insert_sort 进行下一个 i 的操作。

3. 运行结果:

使用给定的 a.in 文件, 加载其中 N 和 N 个待排序的数到 buffer, 结果如下:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000014	0x000041a8	0x00003af2	0x0000aada	0x0000c2b	0x0000b783	0x00004ae9	0x0000e4b
0x10010020	0x000099ff	0x00002f44	0x0000044a	0x00009999	0x00003c56	0x00001284	0x00004b43	0x000044b4
0x10010040	0x00003748	0x00003918	0x00004112	0x0000c399	0x00004953	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

排序完以后, 将比较次数和排好后数的写入 a.out 文件, a.out 用 mars 打开的结果如下:

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000003e	0x0000044a	0x000009ff	0x00000c2b	0x00001284	0x00002f44	0x00003748	0x00003918
0x10010020	0x00003af2	0x00003c56	0x00004112	0x000041a8	0x00004953	0x0000e4b	0x00009999	0x0000acda
0x10010040	0x0000b783	0x0000c399	0x0000d4b4	0x0000dae9	0x0000db43	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

第 1 位 0x3e=62, 为排序次数, 与对应 cpp 文件运行结果一致, 而且发现后面的数均呈正序排列, 可见完成了排序, 且总体流程执行过程与 cpp 一致

(3) merge_sort

1. 寄存器和变量之间的分配关系:

\$a0、\$v0 负责存储系统调用中的信息与\$a1、\$a2 完成文件打开关闭与读写;

\$s0 储存 buffer[1]首地址, \$s1 储存 N, \$s2 存储总比较次数, \$s3 存储 pointer 指针, \$s4 存储 head 头结点, \$s5 存储 msort 函数中 head 头结点;

\$t0 存储 main 函数创建链表 for 循环的循环变量 idx, \$t1 存储 new 后分配的结点地址, \$t3 存储创建链表中 buffer[idx]值, \$t6 存储排序后的链表的 head, \$t9 存储最后写入文件过程中 pointer[1];

在 msort 函数里, \$t2 存储快指针 stride_2_pointer, \$t1 存储慢指针 stride_1_pointer, \$t4 存储 stride_1_pointer[1], \$t3 存储 stride_2_pointer[1], \$t6 储存 msort 返回结点, \$s6 存放 l_head, \$s7 存放 r_head 用于后续 merge 函数;

在 merge 函数里, \$s6 存放虚拟头结点 head, \$t1 存放左链表指针 p_left, \$t2 存放右链表指针 p_right, 循环过程中\$t7 存储 p_right_temp, \$t8 存储 p_left[1], \$t9 存储 p_right[1], 循环结束后\$t7 储存最终返回结点 rv 用于 msort 调用;

其余还涉及到一些\$sp 栈操作或是小循环处\$t 的调用, 因为细节太多不具体一一罗列, 详情见代码中注释。

2. 过程调用操作:

main 同其他两个排序类似, 加载完 buffer 后进入 loop 创建链表, 如果计数器大于 N, 则结束循环进入 loop_exit, 先加载 s5 作为 head 跳转到 msort 函数中对链表进行排序, 最终排序后加载排序后链表头结点通过 pointer 将比较次数和排序后的值写入到 out 文件并关闭;

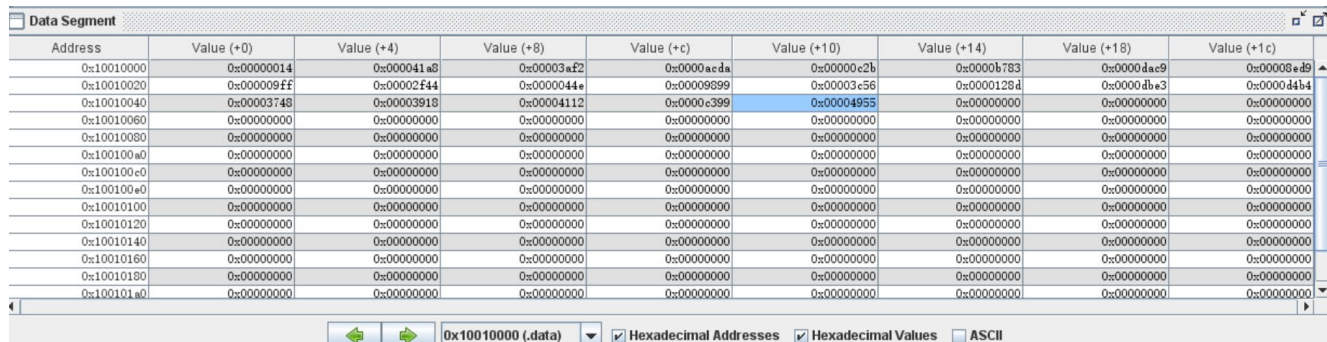
在 msort 函数里若 head[1] 为空则不在分割, 进入 msort_exit 返回 head, 否则进入到循环 msort_loop 里移动快慢指针找到中点, 若快指针到终点, 则退出进入 msort_loop_exit 跳转回 msort 里面, 将慢指针后一个位置置为新头结点, 和原来的头结点分别再调用 msort 函数进行递归, 划分, 子链表排序后返回头结点至父过程, 最终将通过 merge 来将两个子链表按照正序合并;

在 merge 函数里先构建虚拟头结点, 将左链表头结点作为新链表头结点, 进入 merge_outer_loop 外循环, 其中先进入内循环 merge_inner_loop 寻找左链中的插入位置, 然后判断 p_left[1] 是否为空, 若为空则进入 merge_outer_loop_exit1 跳出循环, 将后续右链表全部接入该位置后方, 否则进入内循环 merge_inner_loop1 寻找右链插入位置, 最终合并左右链表, 判断 p_right[1] 是否为空, 若为空则合并完成进入 merge_outer_loop_exit 跳出循环, 返回到父过程 msort 中作为新的子链表再进行合并。

在 merge_inner_loop 内循环中, 先判断 p_left[1] 是否为空, 为空则进入 merge_inner_loop_exit 跳转回外循环 merge_outer_loop; 否则判断两个指针此时的对应值, 若第一个对应值大于第二个则进入 merge_inner_loop_exit 跳转回外循环 merge_outer_loop, 否则将第一个指针移动到下一个位置, 往复循环。

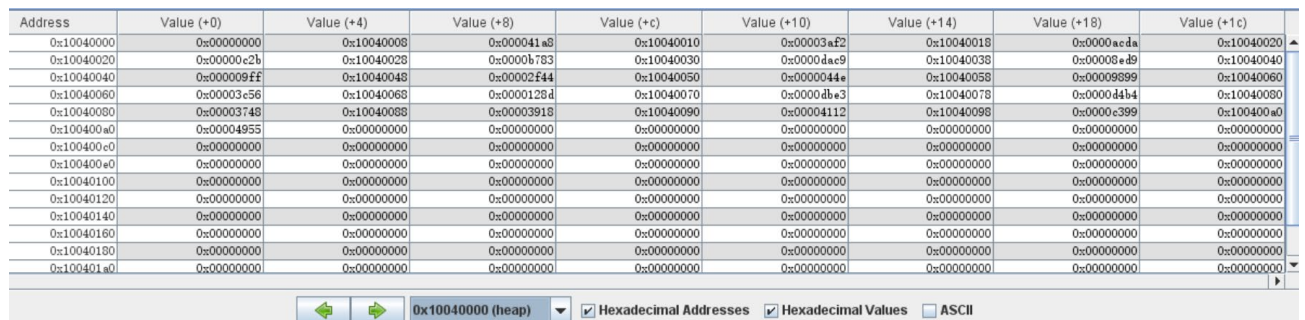
3. 运行结果:

使用给定的 a.in 文件, 加载其中 N 和 N 个待排序的数到 buffer, 结果如下:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000014	0x000041a8	0x00003ef2	0x0000acd8	0x00000e2b	0x0000b783	0x0000dac9	0x00008e89
0x10010020	0x000009ff	0x00002f44	0x0000044e	0x00009999	0x00003c56	0x0000128d	0x0000db3	0x0000d4b4
0x10010040	0x00003748	0x00003918	0x00004112	0x0000c399	0x00004955	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

归并排序先构造好链表, 如下图所示, 在链表中每个结点均占据 8byte, 2word 的空间, 除去 head 以外第一个字储存着待排序的值, 第二个字储存着下一个结点的首地址 (即 next 指针):



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10040000	0x00000000	0x10040008	0x000041a8	0x10040010	0x00003ef2	0x10040018	0x0000acd8	0x10040020
0x10040020	0x00000e2b	0x10040028	0x0000b783	0x10040030	0x0000dac9	0x10040038	0x00008e89	0x10040040
0x10040040	0x000009ff	0x10040048	0x00002f44	0x10040050	0x0000044e	0x10040058	0x00009999	0x10040060
0x10040060	0x00003c56	0x10040068	0x0000128d	0x10040070	0x0000db3	0x10040078	0x0000d4b4	0x10040080
0x10040080	0x00003748	0x10040088	0x00003918	0x10040090	0x00004112	0x10040098	0x0000c399	0x100400a0
0x100400a0	0x00004955	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100400c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100400e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100401a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

排完序后的链表如下图所示，可知此时链状结构已经按照值的大小呈正序排好（0x100400a8 以后的为调用过程新开辟的虚拟头结点，不算在链表里）：

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10040000	0x00000000	0x10040050	0x000041a8	0x100400a0	0x00003af2	0x10040060	0x0000acda	0x10040028
0x10040020	0x00000e2b	0x10040068	0x0000b783	0x10040098	0x0000dae9	0x10040070	0x00008ed9	0x10040058
0x10040040	0x000009ff	0x10040020	0x00002f44	0x10040080	0x0000044e	0x10040040	0x00009899	0x10040018
0x10040060	0x00003c56	0x10040090	0x0000128d	0x10040048	0x0000dbae3	0x00000000	0x000044b4	0x10040030
0x10040080	0x00003748	0x10040088	0x00003918	0x10040010	0x00004112	0x10040008	0x0000c399	0x10040078
0x100400a0	0x00004955	0x10040038	0x00000000	0x10040010	0x00000000	0x10040010	0x00000000	0x10040020
0x100400c0	0x00000000	0x10040020	0x00000000	0x10040038	0x00000000	0x10040040	0x00000000	0x10040050
0x100400e0	0x00000000	0x10040050	0x00000000	0x10040050	0x00000000	0x10040060	0x00000000	0x10040068
0x10040100	0x00000000	0x10040078	0x00000000	0x10040068	0x00000000	0x10040080	0x00000000	0x10040080
0x10040120	0x00000000	0x100400a0	0x00000000	0x10040080	0x00000000	0x10040068	0x00000000	0x10040050
0x10040140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10040180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100401a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10040000 (heap) Hexadecimal Addresses Hexadecimal Values ASCII

排序完以后，将比较次数和排好后数的写入 a.out 文件，a.out 用 mars 打开的结果如下：

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000004c	0x0000044e	0x000009ff	0x00000e2b	0x0000128d	0x00002f44	0x00003748	0x00003918
0x10010020	0x00003af2	0x00003c56	0x00004112	0x000041a8	0x00004955	0x00008ed9	0x00009899	0x0000acda
0x10010040	0x0000b783	0x0000c399	0x0000dbae3	0x0000dae9	0x0000dbae3	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

第 1 位 0x4c=76，为排序次数，与对应 cpp 文件运行结果一致，而且发现后面的数均呈正序排列，可见完成了排序，且总体流程执行过程与 cpp 一致