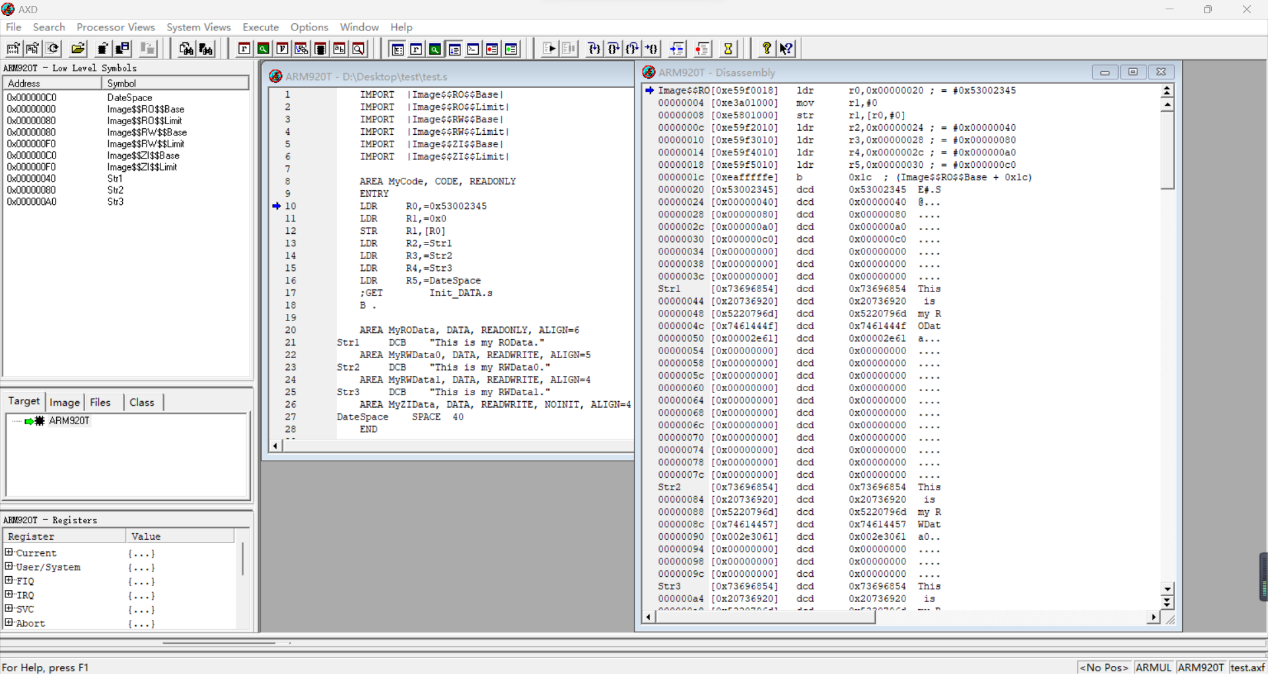
嵌入式作业1

一、观察映像文件的组织：

**（1）观察各段的安排次序。**



a）Image$$RO$$Base=0x00, Image $$RO$$Limit =0x80

源程序中共8条汇编指令，反汇编后可以看到6条伪指令（LDR）转为6条机器指令，所以共8条机器指令，占32字节（0x00到0x1F）。此外，5条LDR伪指令需要通过文字池实现，所以在代码段后面开辟5个字，但需要8字对齐，所以空3个字，总共开辟8个字的存储空间（0x20到0x3F）。只读数据段MyROData段对齐6，即地址后6位为0，共18个字符，需占用0x40到0x51共18字节。但由于64字节对齐，该数据段至少64字节（0x40到0x80）。

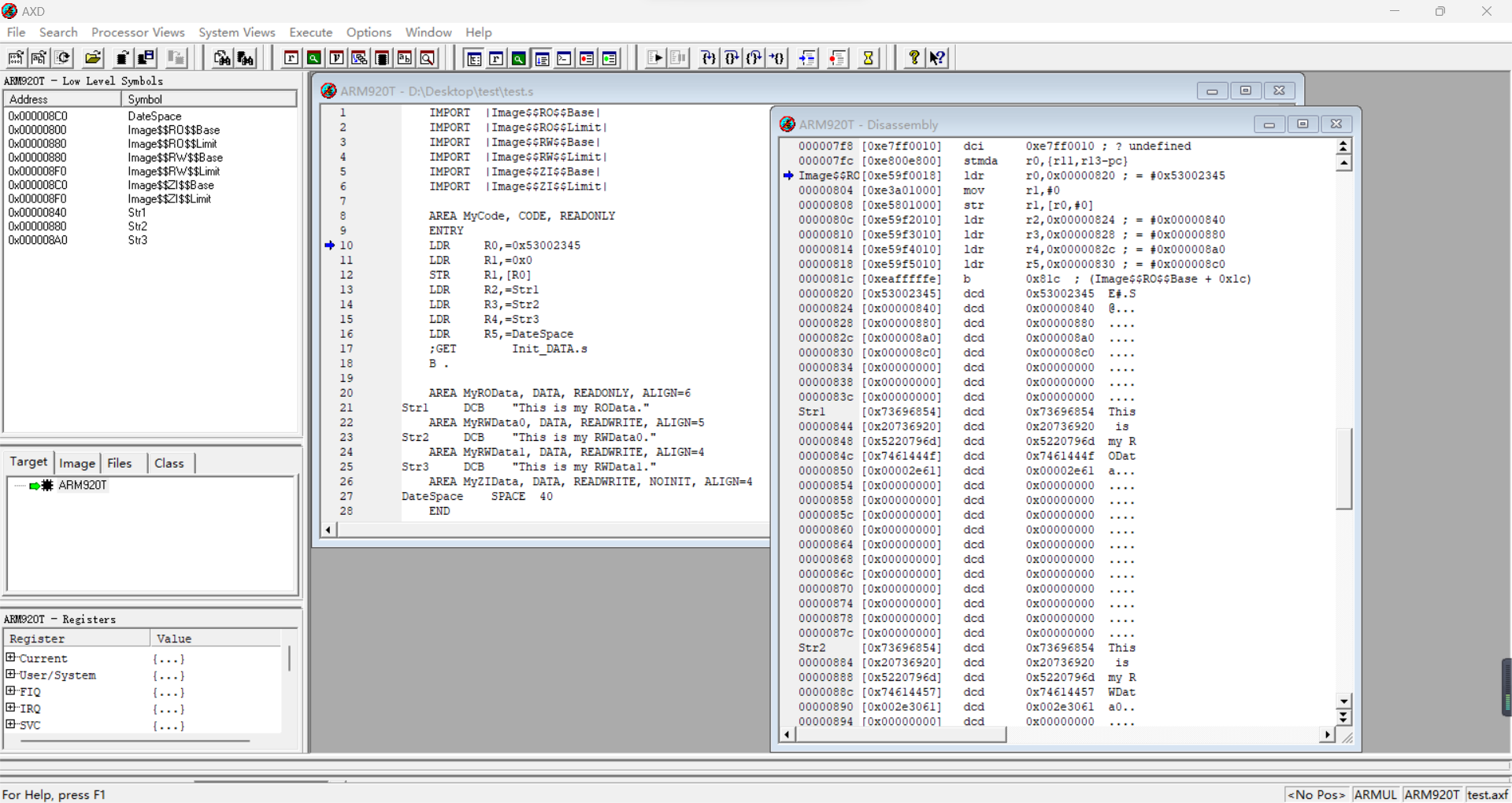
b）Image$$RW$$Base=0x80，Image$$RW$$Limit=0xF0

数据段MyRWData0段对齐5，该数据段至少32字节（0x80到0x9F）；数据段MyRWData1段对齐4，该数据段至少16字节，但由于字符超过16个，因此占32字节（0xA0到0xBF）。

c）Image$$ZI$$Base=0xC0，Image$$ZI$$Limit=0xF0

未初始化的数据段MyZIData段对齐4，该数据段至少16字节，但由于分配了40字节，因此实际占48字节（0xC0到0xF0）。此外，RW的结尾默认和ZI结尾一致.

**（2）修改RO Base为0x800，汇编后再观察各段的次序、地址。**



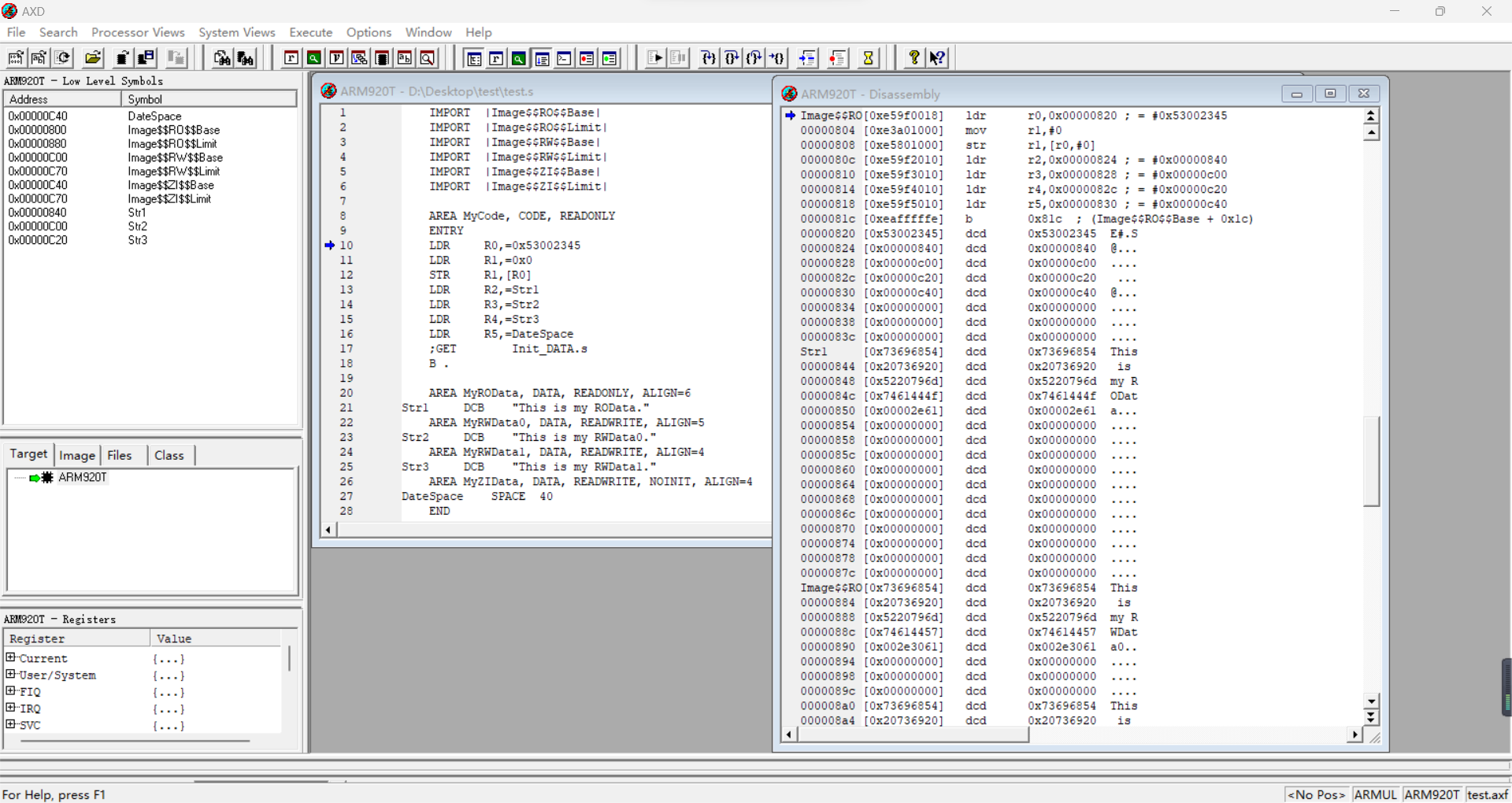
Image$$RO$$Base=0x800, Image $$RO$$Limit =0x880

Image$$RW$$Base=0x880，Image$$RW$$Limit=0x8F0

Image$$ZI$$Base=0x8C0，Image$$ZI$$Limit=0x8F0

各段长度及分布与（a）中分析相同。

**（3）修改RW Base为0xC00，汇编后再观察各段的次序、地址。**



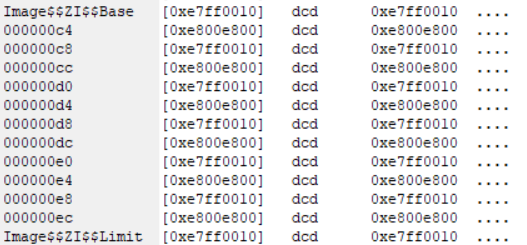
Image$$RO$$Base=0x800, Image $$RO$$Limit =0x880

Image$$RW$$Base=0xC00，Image$$RW$$Limit=0xC70

Image$$ZI$$Base=0xC40，Image$$ZI$$Limit=0xC70

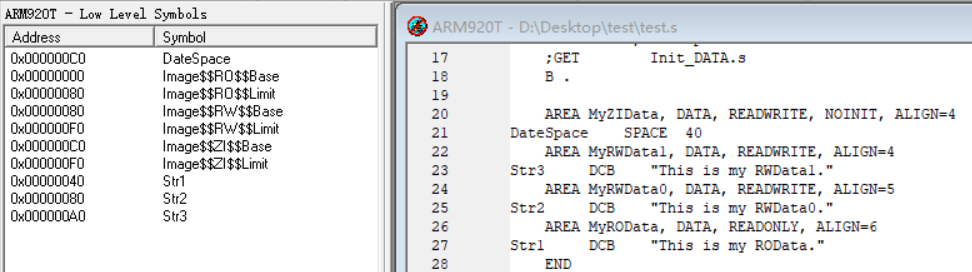
但实际上，反汇编后发现RW数据段紧跟着存储在RO段之后，其起始地址为0x880，但文字池中的RW数据段地址却是0xC00、0xC20、0xC40。原因是在AXD中观察存储映像时，看到的主存实际上是flash的情况。因此，需要有代码在开机后从flash中将RW段搬到主存0xC00处。

**（4）观察MyZIData数据段是否存在，为什么？**

****

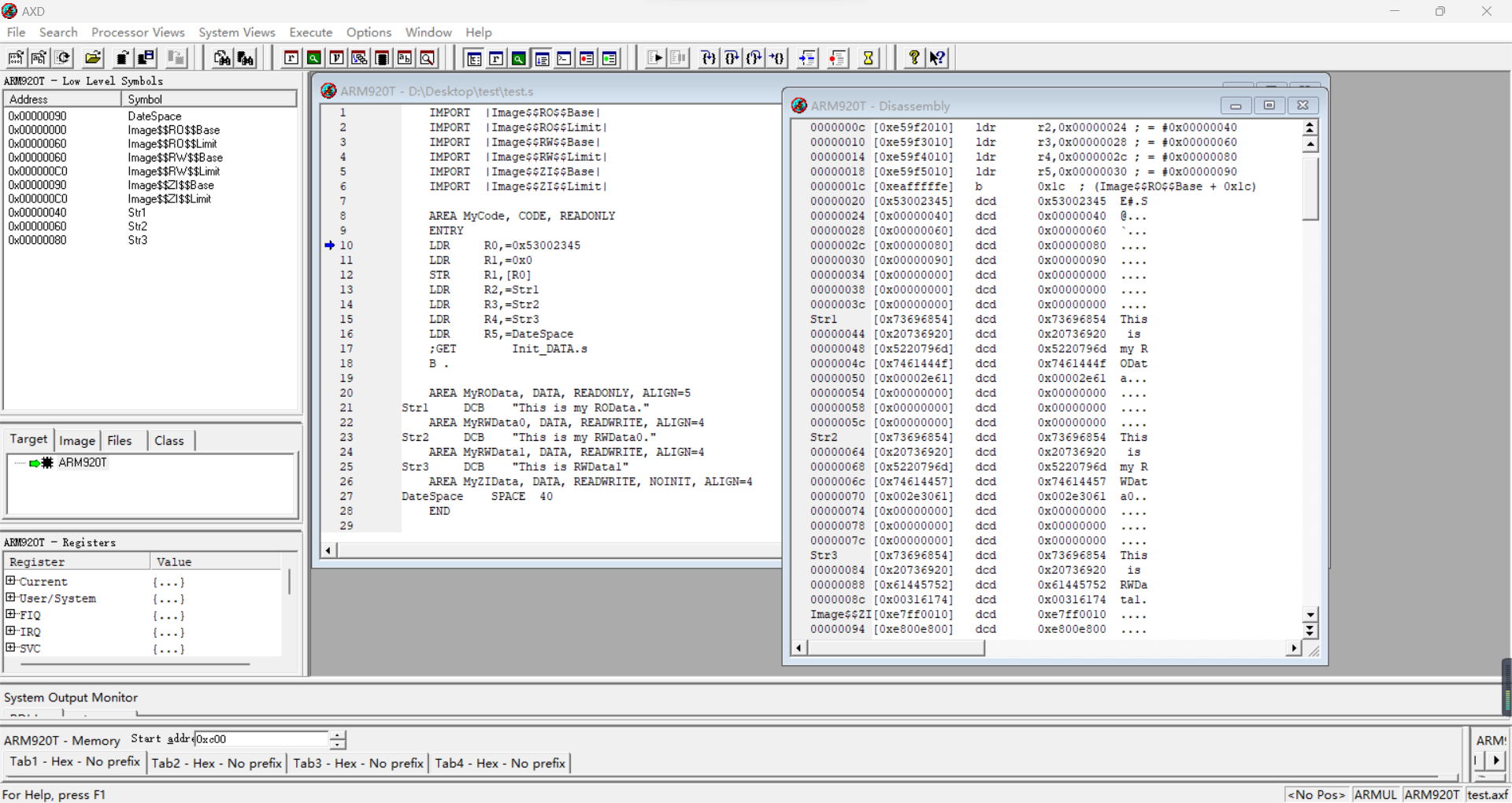
不存在，汇编指令给MyZIData数据段分配40个字节并初始化为0，但反汇编后发现其中存储的并不是0。只有在装入内存后，ZI段才存在并被赋0。

**（5）调整源代码中各段顺序，汇编后再观察各段的次序、地址。**

****

将源代码中各段顺序依次调整为MyZIData、MyRWData1、MyRWData0、MyROData，汇编后观察发现各段次序与之前的保持一致，即RO段在RW段前面，而ZI段位于RW段内的最后位置。对于同处于RW段的MyRWData0与MyRWData1段，MyRWData0段位于MyRWData1段之前，两者是依据段名进行排序分布的。

**（6）改变各段对齐方式，改变各数据段长度，再进行观察，总结对齐规律。**



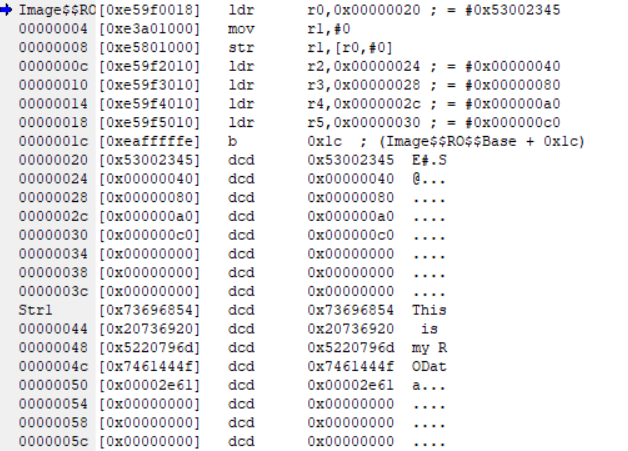
MyROData段对齐5，字符串长度为18字节，数据段总共占32字节；

MyRWData0段对齐4，字符串长度为19字节，数据段总共占32字节；

MyRWData1段对齐4，字符串长度为15字节，数据段总共占16字节。

对齐规律为：当数据段对齐N时，其起始地址的后N位为0，整个数据段的大小总共占k\*(2^N)字节，其中k为正整数，且k要尽可能小并使得数据段能容纳下所有定义的数据。

二、观察伪指令“LDR R0,=0x53002345”及其它指令



文字池

**（1）有没有文字池，若有，观察文字池的地址及内容**

文字池位于代码段之后，依次存储着0x53002345、0x00000040、0x00000080、0x000000a0、0x000000c0，分别表示伪指令“LDR R0, =0x53002345”中的立即数以及数据段Str1、Str2、Str3、DateSpace的起始地址。

**（2）观察这些指令对应的机器指令，说明对应的汇编指令及寻址方式**

伪指令LDR R0, =0x53002345中的立即数大于8位，故该伪指令被转换为LDR R0, [PC, #0x18]，执行该指令时，PC=0x08，该指令将会从地址0x20（文字池）中取出数据赋给寄存器R0，指令的寻址方式为相对寻址；

伪指令LDR R1,=0x0中的立即数小于8位，可以由8位立即数0x00循环右移0位得到，故该伪指令被转换为MOV R1, #0，指令的寻址方式为立即寻址；

汇编指令STR R1, [R0] 将R1寄存器的值存入以R0内容为起始地址的存储器的4字节中，指令的寻址方式为寄存器间接寻址；

伪指令LDR R2, =Str1将Str1所表示的地址的值存入寄存器R2中，由于地址为32位，故实际的值将被存储在文字池中，该伪指令被转换为LDR R2 [PC, #0x10]，指令的寻址方式为相对寻址；

后续的三条LDR伪指令分析同上；

汇编指令B .中的’.’表示当前地址，使得程序在此陷入死循环。

三、单步执行，观察寄存器情况。

①R0=0x53002345，PC=0x00000004

②R1=0x00000000，PC=0x00000008

③PC=0x0000000C

④R2=0x00000040，PC=0x00000010

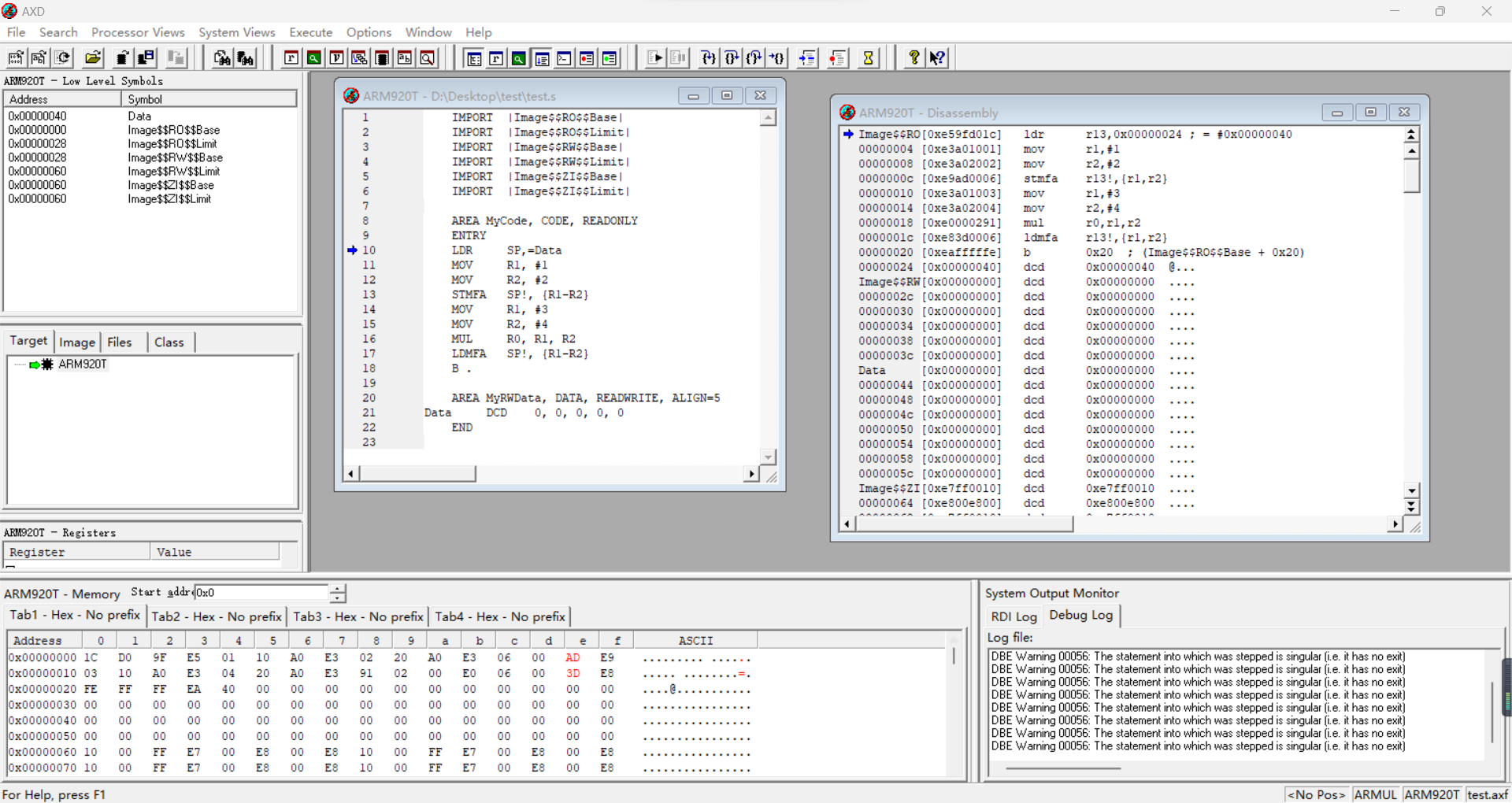
⑤R3=0x00000080，PC=0x00000014

⑥R4=0x000000A0，PC=0x00000018

⑦R5=0x000000C0，PC=0x0000001C

⑧PC=0x0000001C（在此陷入死循环）

四、其它指令



单步执行时，各寄存器值如下：

①LDR SP,=Data ;R13=0x00000040，PC=0x00000004

②MOV R1, #1 ;R1=0x00000001，PC=0x00000008

③MOV R2, #2 ;R2=0x00000002，PC=0x0000000C

④STMFA SP!, {R1-R2} ;R13=0x00000048，PC=0x00000010

⑤MOV R1, #3 ;R1=0x00000003，PC=0x00000014

⑥MOV R2, #4 ;R2=0x00000004，PC=0x00000018

⑦MUL R0, R1, R2 ;R0=0x0000000C，PC=0x0000001C

⑧LDMFA SP!, {R1-R2}

;R1=0x00000001，R2=0x00000002，R13=0x00000040，PC=0x00000020

⑨B . ;PC=0x00000020（在此陷入死循环）