MIPS IDE 实验报告

郑天杨 3160102142

05/02/2019

基本功能

汇编

1. 汇编支持教材附录 A10 所有整数指令和伪指令。
2. 汇编支持数据声明和初始化伪指令。
3. 读取源.asm 文件，生成目标.bin 或.coe 格式文件。

反汇编

1. 反汇编支持附录 A10 所有整数指令。
2. 读取源 bin 或 coe 文件，生成目标 asm 或 bin 或 coe 格式文件。

文件 IO

1. 新建文件（N）（Ctrl+N）
2. 打开文件（O）→（3 个子项：asm、bin 和 coe）
3. 保存文件（S）（Ctrl+S）
4. 保存为（A）→（3 个子项：asm、bin 和 coe）

其他功能

编辑：可以编辑文本。包括复制、剪切和粘贴。

注意事项

1. 修改编辑区的内容后，必须保存，汇编/反汇编的内容才会更新。
2. 如果想要修改编辑区的文件类型，必须新建一个文件或者打开一个相同后缀的文件。

文件格式说明

asm 文件格式

1. 汇编指令以;为结束符（也可以没有），//和#为注释符。
2. 寄存器符号支持 R0 ~ R31（大小写均可）和 MIPS 使用约定。
3. 关于代码和数据区块的说明。代码区块和数据区块开始前，均需用地址伪指令指明起始地址和接下来的指令类型（代码/数据）。没有指明时，默认起始地址为 0，类型是代码指令。地址伪指令 BaseAddre 和代码伪指令 DataAddre 后都只能接一个地址。
例如

```
baseAddre: 00000de0
```

coe 文件格式

1. 只支持 16 进制表示。
2. 语句以;结束。;后为注释。
3. 不同语句以 newline 分开。
4. 每个文件中，先声明 16 进制，

```
memory_initialization_radix=16;
```

再定义初始化向量

```
memory_initialization_vector=
```

如

```
8c650000, 00a85824, 0800003e;
```

向量之间以逗号和（可选的）空白分隔。

bin 文件格式

1. bin 文件以字符的形式存储机器码（注意，这个汇编器支持的并不是真正的二进制文件）。每条机器码为 32 位。
2. 不同机器码以 newline 分开。其他的空白符被忽略。

汇编器实现

MIPS 指令

MIPS 指令符合教材的 MIPS 指令格式标准。不同指令以 newline 分隔。每行末尾的;是可选的。

MIPS 伪指令

同上。

地址伪指令

BaseAddre 和 DataAddre 说明接下来的指令是代码/数据指令，并指定起始地址。每次制定一个地址。两个地址申明之间的空余处填“00000000”，后继申明地址必须大于“前次申明地址+已使用空间”，否则结出错误指示。代码空间超出范围也结出错误指示。地址必须是 4 的倍数。

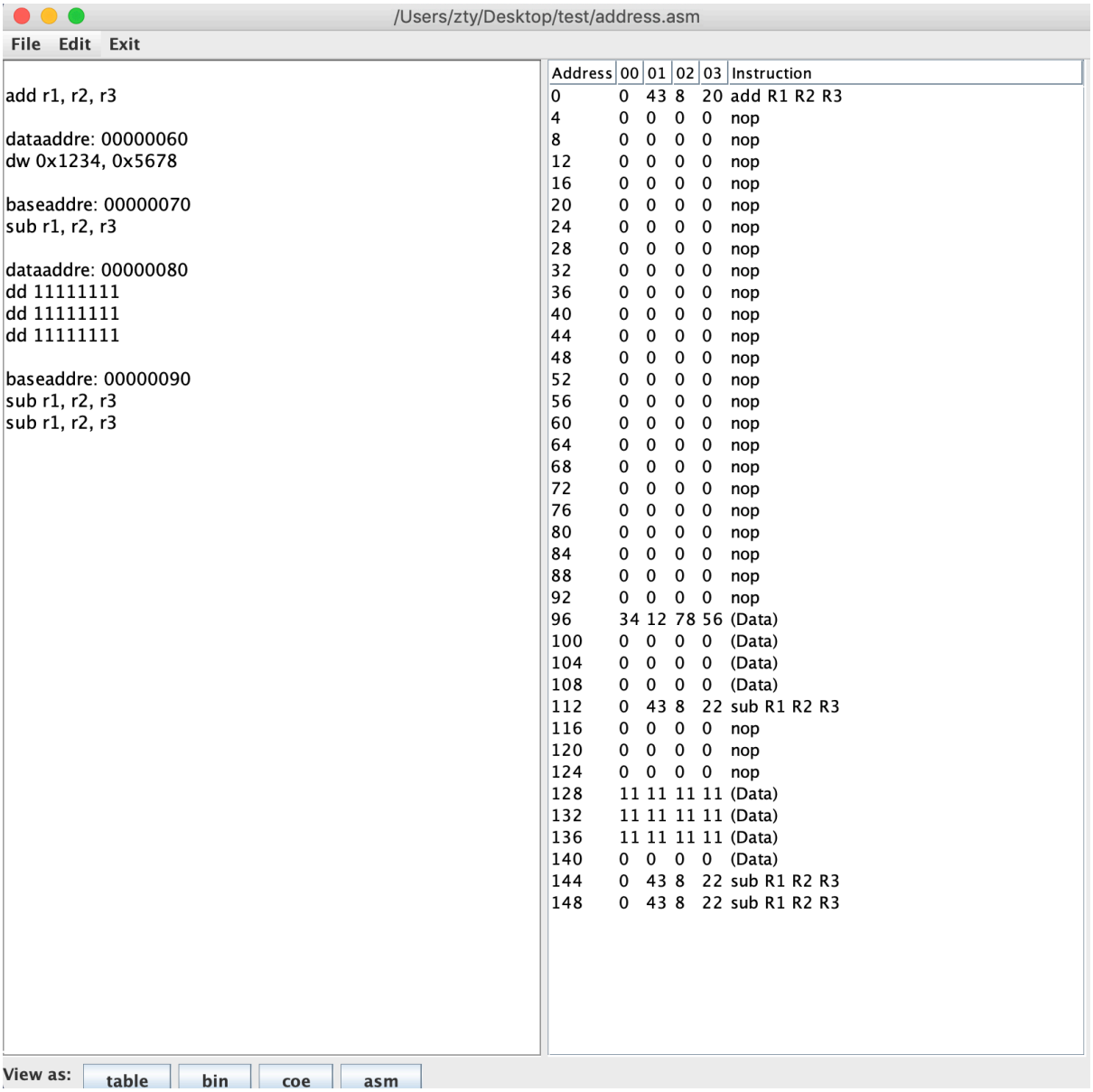


图 测试地址指令：默认起始地址为 0

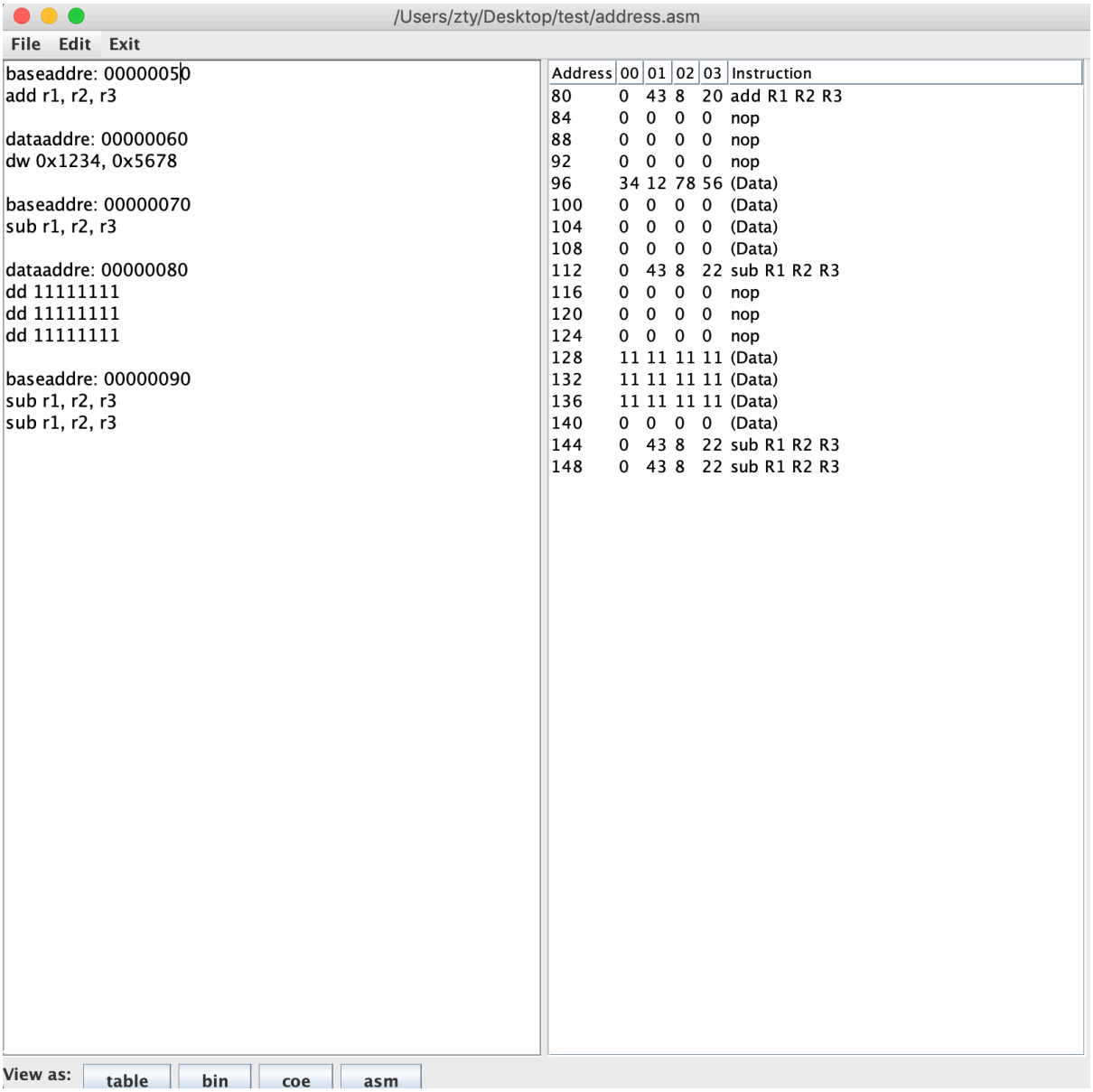


图 测试地址指令：在数据区域，指令显示为（Data）

数据伪指令

支持数据伪指令 DB, DW, DD, EQU。
DB, DW, DD 后接十六进制数，EQU 后接 10 进制数。

异常处理

遇到错误时终止汇编。

数据格式

MIPS 指令存储在 csv 文件（mips_int.csv）中。例如

```
jalr,0,ARG1,0,ARG2,0,9
```

表示 jalr 指令第一个域为 0，第二个域是 argument 1 的二进制码，以此类推。

MIPS 伪指令存储在 csv 文件（mips_pseudo.csv）中。例如

```
sgt,slt,$at,ARG2,ARG1,NEWLINE,bne,$at,$zero,LABEL+3,NEWLINE,ori,ARG1,$zero,
0,NEWLINE,beq,$zero,$zero,LABEL+2,NEWLINE,ori,ARG1,$zero,,,,,,
```

中，NEWLINE 分隔对应的 MIPS 指令，ARG1 表示应填入伪指令的第一个 argument。LABEL+3 表示在三条指令后插入一个 LABEL。

反汇编器实现

机器码

机器码为 32 位比特串（字符串）。两条机器码以 newline 分隔。

异常处理

遇到错误时打印 error。

数据格式

机器码与 MIPS 指令的匹配以正则表达式的形式存储在 csv 文件（mips_regex.csv）中。例如，

```
jalr,000000[01][01][01][01][01]00000[01][01][01][01][01]00000001001
```

表示匹配正则表达式“000000[01][01][01][01][01]00000[01][01][01][01][01]00000001001”的机器码对应的 MIPS 指令为 jalr。

GUI

界面

如图，窗口左边文字区域是编辑器，右边文字区域是汇编/反汇编结果。窗口可以拖拽缩放，文字区域的大小会相应改变。左下角的按钮可用于选择将汇编/反汇编结果以表格/bin/coe/asm 格式展现。

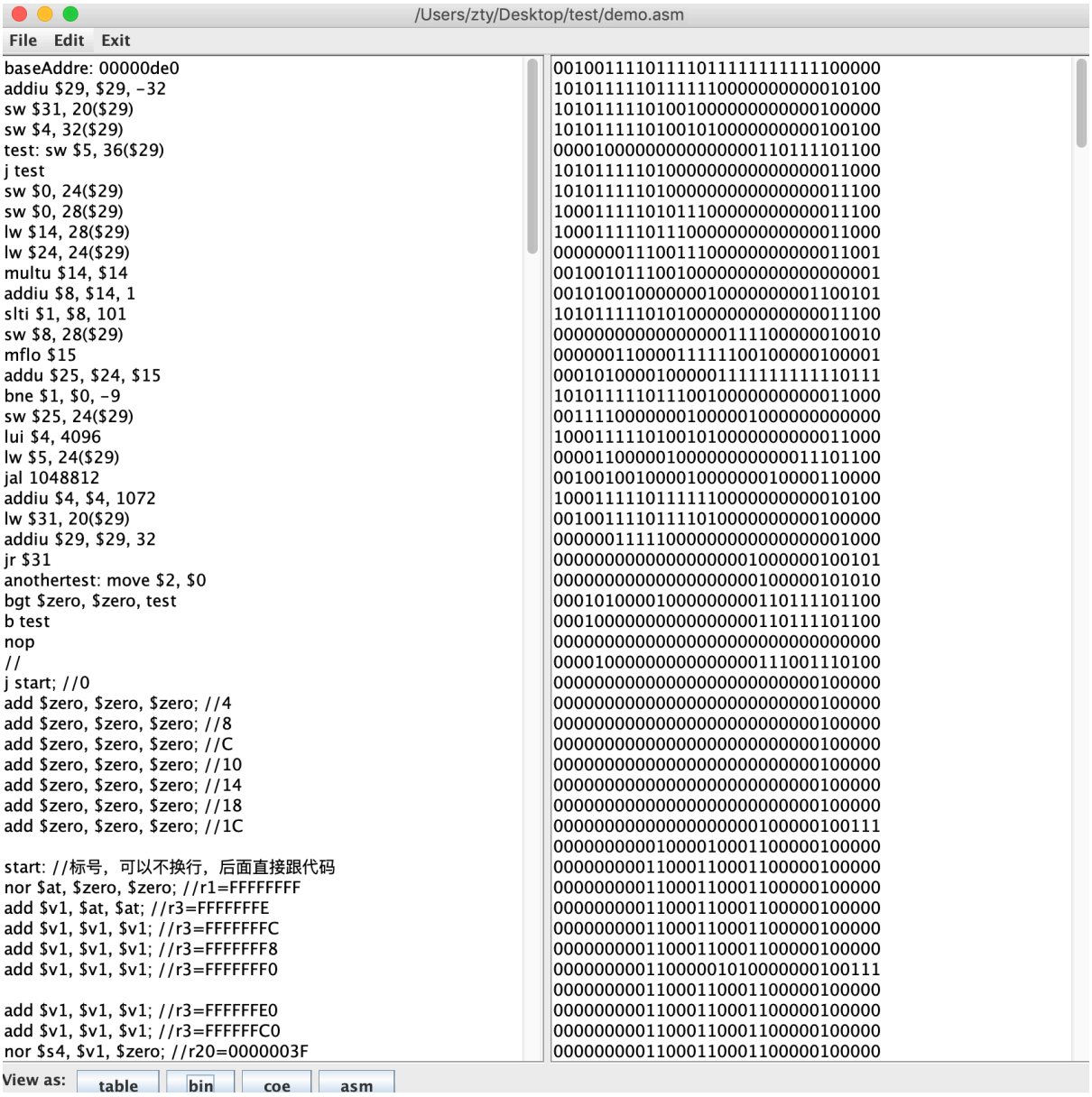


图 界面

文件 IO 功能

实现了打开文件（按后缀过滤）、保存编辑器（ctrl-S 保存编辑区的文件）和将另存为汇编/反汇编结果另存为的功能。

编辑器功能

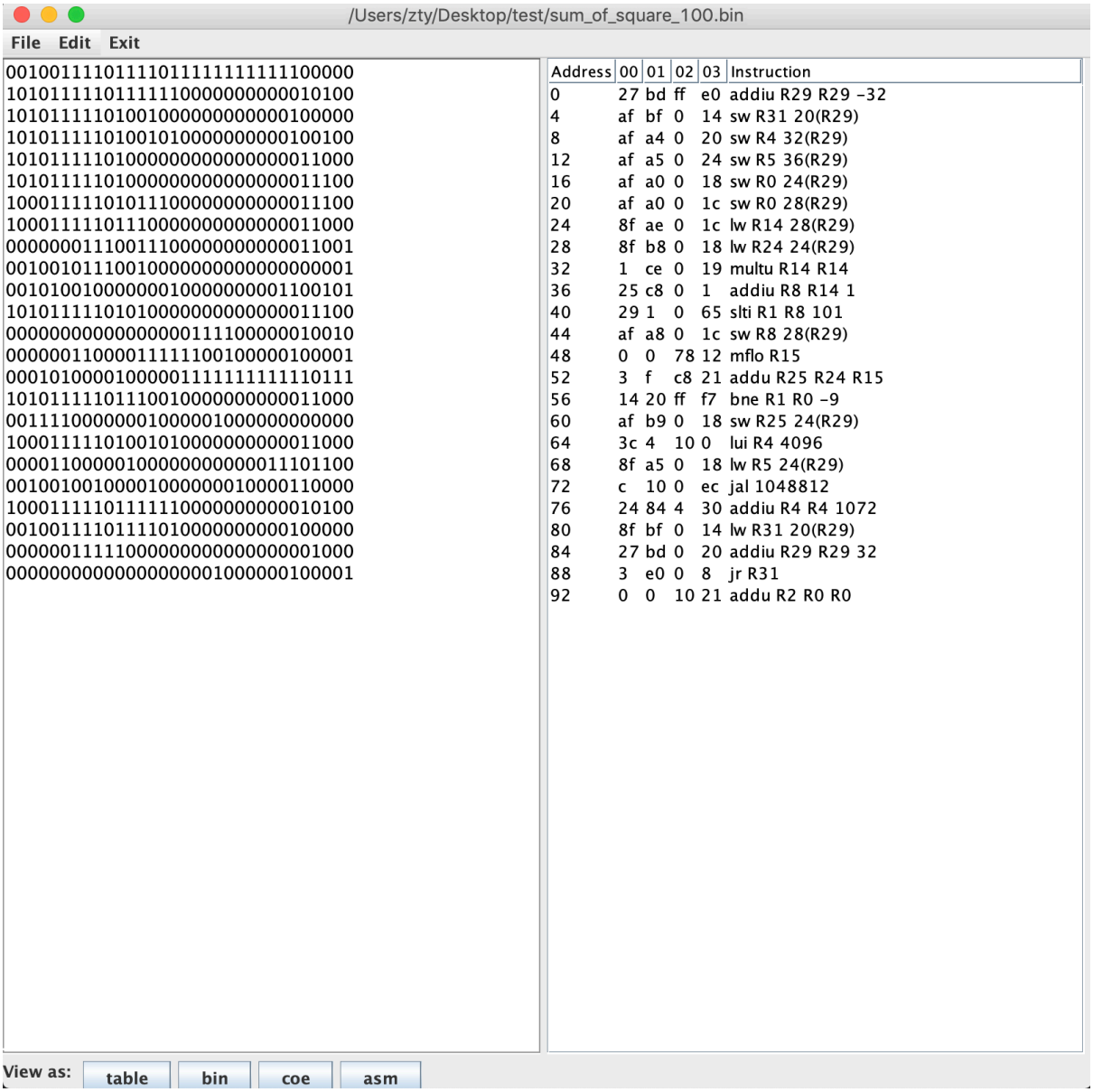


图 测试编辑-保存（复制最后一行机器码，然后 c-S 保存）

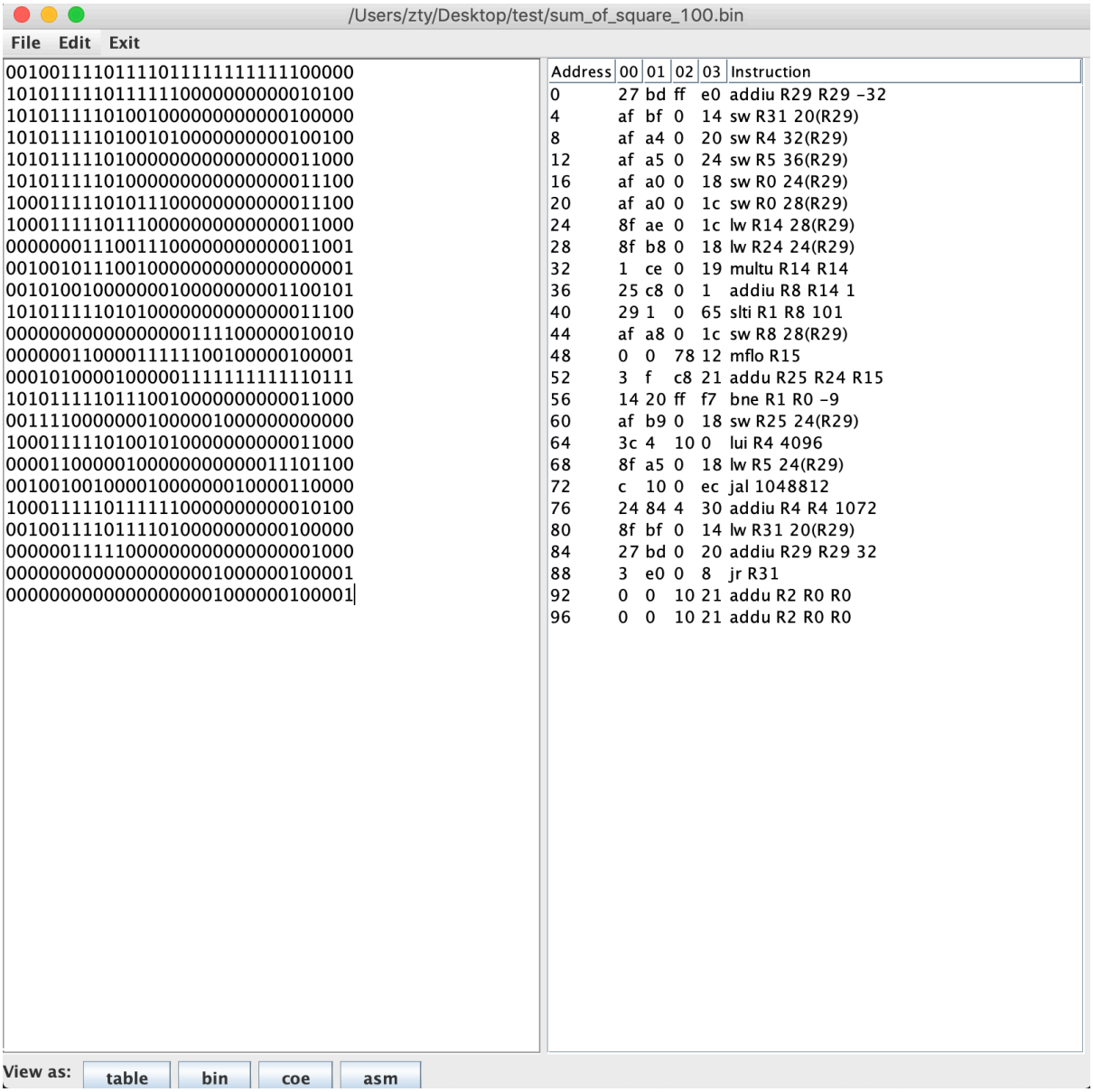


图 测试编辑-保存（复制最后一行机器码，然后 c-S 保存）

附录：测试样例

demo.asm

```
////////////////////////////////////////////////////////
//下面这部分是老师提供的样例
////////////////////////////////////////////////////////
baseAddre: 00000de0
addiu $29, $29, -32
sw $31, 20($29)
sw $4, 32($29)
test: sw $5, 36($29)
j test
sw $0, 24($29)
sw $0, 28($29)
lw $14, 28($29)
lw $24, 24($29)
multu $14, $14
addiu $8, $14, 1
slti $1, $8, 101
sw $8, 28($29)
mflo $15
addu $25, $24, $15
bne $1, $0, -9
sw $25, 24($29)
lui $4, 4096
lw $5, 24($29)
jal 1048812
addiu $4, $4, 1072
lw $31, 20($29)
addiu $29, $29, 32
jr $31
anothertest: move $2, $0
bgt $zero, $zero, test
b test
nop
//
j start; //0
add $zero, $zero, $zero; //4
add $zero, $zero, $zero; //8
add $zero, $zero, $zero; //C
add $zero, $zero, $zero; //10
add $zero, $zero, $zero; //14
add $zero, $zero, $zero; //18
add $zero, $zero, $zero; //1C

start: //标号，可以不换行，后面直接跟代码
nor $at, $zero, $zero; //r1=FFFFFFFF
add $v1, $at, $at; //r3=FFFFFFFE
add $v1, $v1, $v1; //r3=FFFFFFFC
add $v1, $v1, $v1; //r3=FFFFFFF8
add $v1, $v1, $v1; //r3=FFFFFFF0
```

```
add $v1, $v1, $v1; //r3=FFFFFFE0
add $v1, $v1, $v1; //r3=FFFFFFC0
nor $s4, $v1, $zero; //r20=0000003F
add $v1, $v1, $v1; //r3=FFFFFF80
add $v1, $v1, $v1; //r3=FFFFFF00

add $v1, $v1, $v1; //r3=FFFFFE00
add $v1, $v1, $v1; //r3=FFFFFC00
add $v1, $v1, $v1; //r3=FFFFF800
add $v1, $v1, $v1; //r3=FFFFF000

add $v1, $v1, $v1; //r3=FFFFE000
add $v1, $v1, $v1; //r3=FFFFC000
add $v1, $v1, $v1; //r3=FFFF8000
add $v1, $v1, $v1; //r3=FFFF0000

add $v1, $v1, $v1; //r3=FFFE0000
add $v1, $v1, $v1; //r3=FFFC0000
add $v1, $v1, $v1; //r3=FFF80000
add $v1, $v1, $v1; //r3=FFF00000

add $v1, $v1, $v1; //r3=FFE00000
add $v1, $v1, $v1; //r3=FFC00000
add $v1, $v1, $v1; //r3=FF800000
add $v1, $v1, $v1; //r3=FF000000

add $v1, $v1, $v1; //r3=FE000000
add $v1, $v1, $v1; //r3=FC000000
add $a2, $v1, $v1; //r6=F8000000
add $v1, $a2, $a2; //r3=F0000000

add $a0, $v1, $v1; //r4=E0000000

add $t5, $a0, $a0; //r13=C0000000
add $t0, $t5, $t5; //r8=80000000

loop: //标号，可以不换行，后面直接跟代码
slt $v0, $zero,$at; //r2=00000001
add $t6, $v0, $v0;
add $t6, $t6, $t6; //r14=4
nor $t2, $zero, $zero; //r10=FFFFFFFF
add $t2, $t2, $t2; //r10=FFFFFFFE

loop1:
sw $a2, 4($v1); //计数器端口:F0000004，送计数常数 r6=F8000000
lw $a1, 0($v1);
add $a1, $a1, $a1; //左移
add $a1, $a1, $a1;
sw $a1, 0($v1);

add $t1, $t1, $v0; //r9=r9+1
sw $t1, 0($a0); //r9 送 r4=E0000000 七段码端口
lw $t5, 14($zero); //取存储器 20 单元预存数据至 r13,程序计数延时常数

loop2: //标号，可以不换行，后面直接跟代码
lw $a1, 0($v1); //读 GPIO 端口 F0000000 状态
add $a1, $a1, $a1;
add $a1, $a1, $a1; //左移 2 位将 SW 与 LED 对齐，同时 D1D0 置 00，选择计数器 通道
0
sw $a1, 0($v1); //r5 输 出 到 GPIO 端 口 F0000000 ， 计 数 器 通 道 counter_set=00
```

```
lw $a1, 0($v1); //再读 GPIO 端口 F0000000 状态
and $t3,$a1,$t0; //取最高位=out0，屏蔽其余位送 r11
// beq $t3,$t0,C_init; //out0=0,Counter 通道 0 溢出,转计数器初始化,修改 7 段码显示:C_init
add $t5, $t5, $v0; //程序计数延时
beq $t5, $zero,C_init; //程序计数 r13=0,转计数器初始化,修改 7 段码显 示:C_init

l_next: // 判断 7 段码显示模式：SW[4:3]控制
lw $a1, 0($v1); //再读 GPIO 端口 F0000000 开关 SW 状态
add $s2, $t6, $t6; //r14=4,r18=00000008
add $s6, $s2, $s2; //r22=00000010
add $s2, $s2, $s6; //r18=00000018(00011000)
and $t3, $a1, $s2; //取 SW[4:3]
beq $t3, $zero, L20; //SW[4:3]=00,7 段显示"点"循环移位：L20，SW0=0
beq $t3, $s2, L21; //SW[4:3]=11，显示七段图形，L21，SW0=0
add $s2, $t6, $t6; //r18=8
beq $t3, $s2, L22; //SW[4:3]=01,七段显示预置数字，L22，SW0=1
sw $t1, 0($a0); //SW[4:3]=10，显示 r9，SW0=1
j loop2;

L20:
beq $t2, $at, L4; //r10=ffffffff,转移 L4
j L3;

L4:
nor $t2, $zero, $zero; //r10=ffffffff
add $t2, $t2, $t2; //r10=fffffffe

L3:
sw $t2, 0($a0); //SW[4:3]=00,7 段显示点移位后显示
j loop2;

L21:
lw $t1, 60($s1); //SW[4:3]=11，从内存取预存七段图形
sw $t1, 0($a0); //SW[4:3]=11，显示七段图形
j loop2;

L22:
lw $t1, 20($s1); //SW[4:3]=01，从内存取预存数字
sw $t1, 0($a0); //SW[4:3]=01,七段显示预置数字
j loop2;

C_init:
lw $t5, 14($zero); //取程序计数延时初始化常数
add $t2, $t2, $t2; //r10=fffffffc，7 段图形点左移
or $t2, $t2, $v0; //r10 末位置 1，对应右上角不显示
add $s1, $s1, $t6; //r17=00000004，LED 图形访存地址+4
and $s1, $s1, $s4; //r17=000000XX，屏蔽地址高位，只取 6 位
add $t1, $t1, $v0; //r9+1
beq $t1, $at, L6; //若 r9=ffffffff,重置 r9=5
j L7;

L6:
add $t1, $zero, $t6; //r9=4
add $t1, $t1, $v0; //重置 r9=5
```

```
L7:
lw $a1, 0($v1); //读 GPIO 端口 F0000000 状态
add $t3, $a1, $a1;
add $t3, $t3, $t3; //左移 2 位将 SW 与 LED 对齐，同时 D1D0 置 00，选择计数器 通道
0
sw $t3, 0($v1); //r5 输 出 到 GPIO 端 口 F0000000 ， 计 数 器 通 道 counter_set=00
sw $a2,4($v1); //计数器端口:F0000004，送计数常数 r6=F8000000

j l_next;

////////////////////////////////////////////////////////////
//从此处-00000FFC 填“00000000”。
////////////////////////////////////////////////////////////

DataAddre: 00001000; //数据地址，此处 00001000H 开始定义数据
Data1: //数据区 1，标号
dd FFFFFF00, 000002AB, 80000000, 0000003F, 00000001, FFFF0000, 0000FFFF,
80000000, 00000000, 11111111, 22222222, 33333333, 44444444, 55555555,
66666666, 77777777, 88888888, 99999999, AAAAAAAA, BBBBBBBB, CCCCCCCC,
DDDDDDDD, EEEEEEEE, FFFFFFFF;
db 0x55,0x56,0x57,0x58; //db 伪指令,定义数据 0x55…在 00001060
dw 'ab',0x1234; //dw 伪指令,定义数据 0x41,0x42,0x1234 在 000010604
dd 0x12345678; //dd 伪指令，定义数据 0x12345678 在 00001068
data2:RESB 16; //data2 是标号，从 0000106C 开始定义 16 个字节空 间

////////////////////////////////////////////////////////////
//此处 0000107C-000010FC 填“00000000”。
////////////////////////////////////////////////////////////

DataAddre: 00001100; //数据地址，此处 00001100H 开始定义数据
Data2: //数据区 2，标号
dd 557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D,
FFFF9DB9, FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9,
FFFF07E0, 007E0FFF, 03bdf020, 03def820, 08002300, 00000001;

////////////////////////////////////////////////////////////
//此处从 0000114C-000011FC 填“00000000”。
////////////////////////////////////////////////////////////


DataAddre: 00001200; //数据地址，此处 00001200H 开始定义数据
Data3:

buffer:RESD 32; //数据区 3，buffer 标号=00001200，定义 32 个 32 位字空间
 //到 00001280 结束

////////////////////////////////////////////////////////////
// 以上是老师提供的样例
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//下面这部分是额外的数据指令测试
////////////////////////////////////////////////////////////
//数据定义例子，逗号隔离(默认为 16 进制)：
db 0x55 ; //ust the byte 0x55
db 0x55,0x56,0x57; // three bytes in succession
db 'a',0x55; //character constants are OK
db 'hello',13,10,'$' ; // so are string constants
```

```
dw 0x1234 ; // 0x34 0x12
dw 'a' ; // 0x41 0x00 (it's just a number)
dw 'ab' ; //0x41 0x42 (character constant)
dw 'abc' ; //0x41 0x42 0x43 0x00 (string)
dd 0x12345678 ; //0x78 0x56 0x34 0x12
//定义空间的例子：
buffer: resb 64; // reserve 64 bytes，buffer 是标号
wordvar: resw 1; // reserve a word，wordvar 是标号

////////////////////////////////////////////////////////////
//下面这部分是额外的伪指令测试
////////////////////////////////////////////////////////////
baseAddre: 1300;
bgt $zero, $zero, test
abs $s1, $s2;
rol $s1, $s2, 13
move $t1, $t1;
```

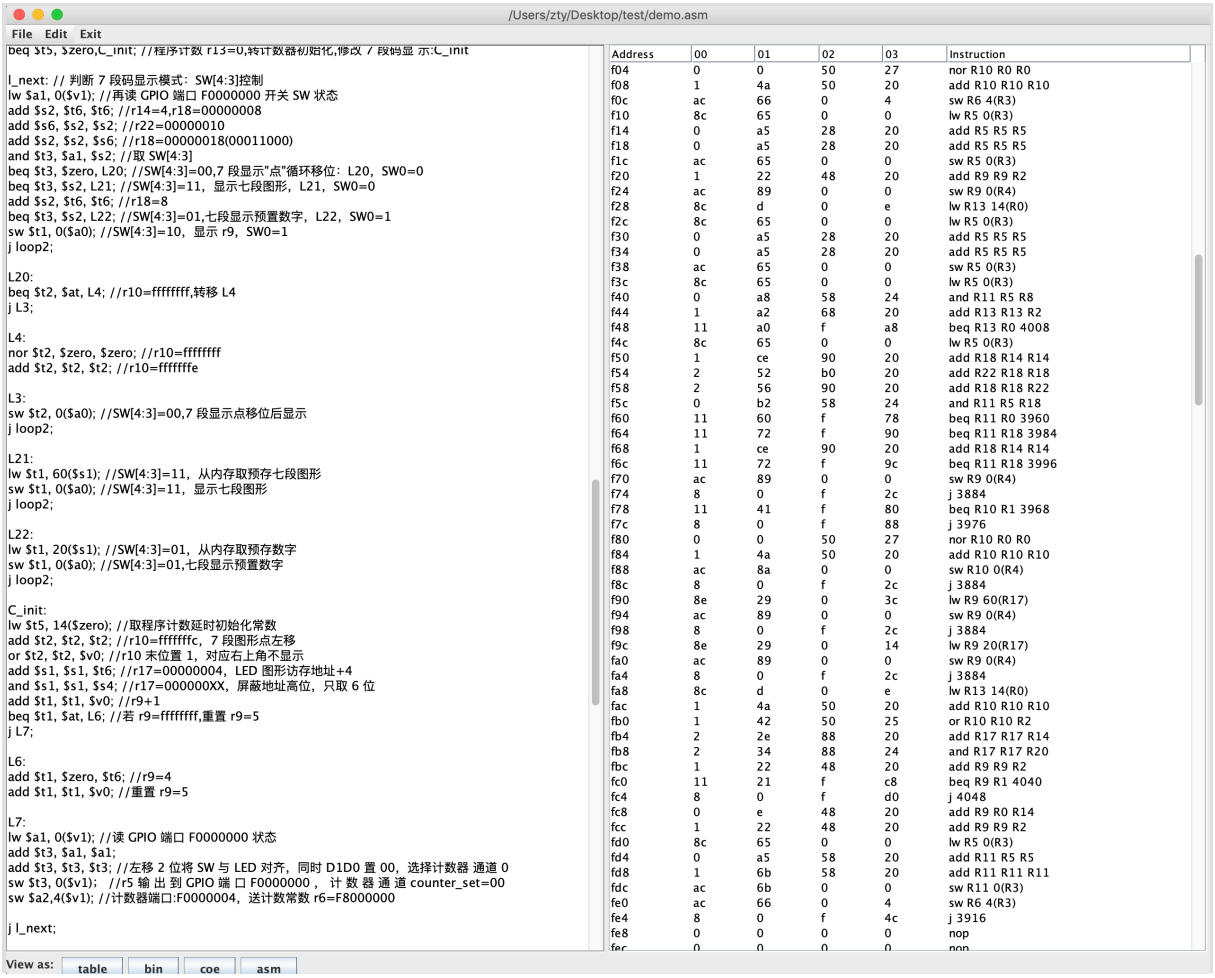
/Users/zty/Desktop/test/demo.asm

File Edit Exit

```
////////////////////////////////////////////////////////////
//下面这部分是老师提供的样例
////////////////////////////////////////////////////////////
baseAddre: 00000de0
addiu $29, $29, -32
sw $31, 20($29)
sw $4, 32($29)
test: sw $5, 36($29)
j test
sw $0, 24($29)
sw $0, 28($29)
lw $14, 28($29)
lw $24, 24($29)
multu $14, $14
addiu $8, $14, 1
slti $1, $8, 101
sw $8, 28($29)
mflo $15
addu $25, $24, $15
bne $1, $0, -9
sw $25, 24($29)
lui $4, 4096
lw $5, 24($29)
jal 1048812
addiu $4, $4, 1072
lw $31, 20($29)
addiu $29, $29, 32
jr $31
anothertest: move $2, $0
bgt $zero, $zero, test
b test
nop
//
j start; //0
add $zero, $zero, $zero; //4
add $zero, $zero, $zero; //8
add $zero, $zero, $zero; //C
add $zero, $zero, $zero; //10
add $zero, $zero, $zero; //14
add $zero, $zero, $zero; //18
add $zero, $zero, $zero; //1C

start: //标号，可以不换行，后面直接跟代码
nor $at, $zero, $zero; //r1=FFFFFFFF
add $v1, $at, $at; //r3=FFFFFFFE
add $v1, $v1, $v1; //r3=FFFFFFFC
add $v1, $v1, $v1; //r3=FFFFFFF8
add $v1, $v1, $v1; //r3=FFFFFFF0

add $v1, $v1, $v1; //r3=FFFFFFE0
add $v1, $v1, $v1; //r3=FFFFFFC0
nor $s4, $v1, $zero; //r20=0000003F
add $v1, $v1, $v1; //r3=FFFFFF80
add $v1, $v1, $v1; //r3=FFFFFF00

add $v1, $v1, $v1; //r3=FFFFFE00
add $v1, $v1, $v1; //r3=FFFFFC00
add $v1, $v1, $v1; //r3=FFFFF800
add $v1, $v1, $v1; //r3=FFFFF000
```

Address	00	01	02	03	Instruction
f04	0	0	50	27	nor R10 R0 R0
f08	1	4a	50	20	add R10 R10 R10
f0c	ac	66	0	4	sw R6 4(R3)
f10	8c	65	0	0	lw R5 0(R3)
f14	0	a5	28	20	add R5 R5 R5
f18	0	a5	28	20	add R5 R5 R5
f1c	ac	65	0	0	sw R5 0(R3)
f20	1	22	48	20	add R9 R9 R2
f24	ac	89	0	0	sw R9 0(R4)
f28	8c	d	0	e	lw R13 14(R0)
f2c	8c	65	0	0	lw R5 0(R3)
f30	0	a5	28	20	add R5 R5 R5
f34	0	a5	28	20	add R5 R5 R5
f38	ac	65	0	0	sw R5 0(R3)
f3c	8c	65	0	0	lw R5 0(R3)
f40	0	a8	58	24	and R11 R5 R8
f44	1	a2	68	20	add R13 R13 R2
f48	11	a0	f	a8	beq R13 R0 4008
f4c	8c	65	0	0	lw R5 0(R3)
f50	1	ce	90	20	add R18 R14 R14
f54	2	52	b0	20	add R22 R18 R18
f58	2	56	90	20	add R18 R18 R22
f5c	0	b2	58	24	and R11 R5 R18
f60	11	60	f	78	beq R11 R0 3960
f64	11	72	f	90	beq R11 R18 3984
f68	1	ce	90	20	add R18 R14 R14
f6c	11	72	f	9c	beq R11 R18 3996
f70	ac	89	0	0	sw R9 0(R4)
f74	8	0	f	2c	j 3884
f78	11	41	f	80	beq R10 R1 3968
f7c	8	0	f	88	j 3976
f80	0	0	50	27	nor R10 R0 R0
f84	1	4a	50	20	add R10 R10 R10
f88	ac	8a	0	0	sw R10 0(R4)
f8c	8	0	f	2c	j 3884
f90	8e	29	0	3c	lw R9 60(R17)
f94	ac	89	0	0	sw R9 0(R4)
f98	8	0	f	2c	j 3884
f9c	8e	29	0	14	lw R9 20(R17)
fa0	ac	89	0	0	sw R9 0(R4)
fa4	8	0	f	2c	j 3884
fa8	8c	d	0	e	lw R13 14(R0)
fac	1	4a	50	20	add R10 R10 R10
fb0	1	42	50	25	or R10 R10 R2
fb4	2	2e	88	20	add R17 R17 R14
fb8	2	34	88	24	and R17 R17 R20
fbc	1	22	48	20	add R9 R9 R2
fc0	11	21	f	c8	beq R9 R1 4040
fc4	8	0	f	d0	j 4048
fc8	0	e	48	20	add R9 R0 R14
fcc	1	22	48	20	add R9 R9 R2
fd0	8c	65	0	0	lw R5 0(R3)
fd4	0	a5	58	20	add R11 R5 R5
fd8	1	6b	58	20	add R11 R11 R11
fdc	ac	6b	0	0	sw R11 0(R3)
fe0	ac	66	0	4	sw R6 4(R3)
fe4	8	0	f	4c	j 3916
fe8	0	0	0	0	nop
fec	0	0	0	0	nop

View as: table bin coe asm


/Users/zty/Desktop/test/demo.asm
File Edit Exit
beq $t5, $zero,C_init; //程序计数 r13=0,转计数器初始化,修改 7 段码显示:C_init
l_next: // 判断 7 段码显示模式：SW[4:3]控制
lw $a1, 0($v1); //再读 GPIO 端口 F0000000 开关 SW 状态
add $s2, $t6, $t6; //r14=4,r18=00000008
add $s6, $s2, $s2; //r22=00000010
add $s2, $s2, $s6; //r18=00000018(00011000)
and $t3, $a1, $s2; //取 SW[4:3]
beq $t3, $zero, L20; //SW[4:3]=00,7 段显示"点"循环移位：L20，SW0=0
beq $t3, $s2, L21; //SW[4:3]=11，显示七段图形，L21，SW0=0
add $s2, $t6, $t6; //r18=8
beq $t3, $s2, L22; //SW[4:3]=01,七段显示预置数字，L22，SW0=1
sw $t1, 0($a0); //SW[4:3]=10，显示 r9，SW0=1
j loop2;
L20:
beq $t2, $at, L4; //r10=ffffffff,转移 L4
j L3;
L4:
nor $t2, $zero, $zero; //r10=ffffffff
add $t2, $t2, $t2; //r10=fffffffe
L3:
sw $t2, 0($a0); //SW[4:3]=00,7 段显示点移位后显示
j loop2;
L21:
lw $t1, 60($s1); //SW[4:3]=11，从内存取预存七段图形
sw $t1, 0($a0); //SW[4:3]=11，显示七段图形
j loop2;
L22:
lw $t1, 20($s1); //SW[4:3]=01，从内存取预存数字
sw $t1, 0($a0); //SW[4:3]=01,七段显示预置数字
j loop2;
C_init:
lw $t5, 14($zero); //取程序计数延时初始化常数
add $t2, $t2, $t2; //r10=fffffffc，7 段图形点左移
or $t2, $t2, $v0; //r10 末位置 1，对应右上角不显示
add $s1, $s1, $t6; //r17=00000004，LED 图形访存地址+4
and $s1, $s1, $s4; //r17=000000XX，屏蔽地址高位，只取 6 位
add $t1, $t1, $v0; //r9+1
beq $t1, $at, L6; //若 r9=ffffffff,重置 r9=5
j L7;
L6:
add $t1, $zero, $t6; //r9=4
add $t1, $t1, $v0; //重置 r9=5
L7:
lw $a1, 0($v1); //读 GPIO 端口 F0000000 状态
add $t3, $a1, $a1;
add $t3, $t3, $t3; //左移 2 位将 SW 与 LED 对齐，同时 D1D0 置 00，选择计数器 通道 0
sw $t3, 0($v1)； //r5 输 出 到 GPIO 端 口 F0000000 ， 计 数 器 通 道 counter_set=00
sw $a2,4($v1); //计数器端口:F0000004，送计数常数 r6=F8000000
j l_next;
View as: table bin coe asm
Address 00 01 02 03 Instruction
f04 0 0 50 27 nor R10 R0 R0
f08 1 4a 50 20 add R10 R10 R10
f0c ac 66 0 4 sw R6 4(R3)
f10 8c 65 0 0 lw R5 0(R3)
f14 0 a5 28 20 add R5 R5 R5
f18 0 a5 28 20 add R5 R5 R5
f1c ac 65 0 0 sw R5 0(R3)
f20 1 22 48 20 add R9 R9 R2
f24 ac 89 0 0 sw R9 0(R4)
f28 8c d 0 e lw R13 14(R0)
f2c 8c 65 0 0 lw R5 0(R3)
f30 0 a5 28 20 add R5 R5 R5
f34 0 a5 28 20 add R5 R5 R5
f38 ac 65 0 0 sw R5 0(R3)
f3c 8c 65 0 0 lw R5 0(R3)
f40 0 a8 58 24 and R11 R5 R8
f44 1 a2 68 20 add R13 R13 R2
f48 11 a0 f a8 beq R13 R0 4008
f4c 8c 65 0 0 lw R5 0(R3)
f50 1 ce 90 20 add R18 R14 R14
f54 2 52 b0 20 add R22 R18 R18
f58 2 56 90 20 add R18 R18 R22
f5c 0 b2 58 24 and R11 R5 R18
f60 11 60 f 78 beq R11 R0 3960
f64 11 72 f 90 beq R11 R18 3984
f68 1 ce 90 20 add R18 R14 R14
f6c 11 72 f 9c beq R11 R18 3996
f70 ac 89 0 0 sw R9 0(R4)
f74 8 0 f 2c j 3884
f78 11 41 f 80 beq R10 R1 3968
f7c 8 0 f 88 j 3976
f80 0 0 50 27 nor R10 R0 R0
f84 1 4a 50 20 add R10 R10 R10
f88 ac 8a 0 0 sw R10 0(R4)
f8c 8 0 f 2c j 3884
f90 8e 29 0 3c lw R9 60(R17)
f94 ac 89 0 0 sw R9 0(R4)
f98 8 0 f 2c j 3884
f9c 8e 29 0 14 lw R9 20(R17)
fa0 ac 89 0 0 sw R9 0(R4)
fa4 8 0 f 2c j 3884
fa8 8c d 0 e lw R13 14(R0)
fac 1 4a 50 20 add R10 R10 R10
fb0 1 42 50 25 or R10 R10 R2
fb4 2 2e 88 20 add R17 R17 R14
fb8 2 34 88 24 and R17 R17 R20
fbc 1 22 48 20 add R9 R9 R2
fc0 11 21 f c8 beq R9 R1 4040
fc4 8 0 f d0 j 4048
fc8 0 e 48 20 add R9 R0 R14
fcc 1 22 48 20 add R9 R9 R2
fd0 8c 65 0 0 lw R5 0(R3)
fd4 0 a5 58 20 add R11 R5 R5
fd8 1 6b 58 20 add R11 R11 R11
fdc ac 6b 0 0 sw R11 0(R3)
fe0 ac 66 0 4 sw R6 4(R3)
fe4 8 0 f 4c j 3916
fe8 0 0 0 0 nop
fec 0 0 0 0 nop

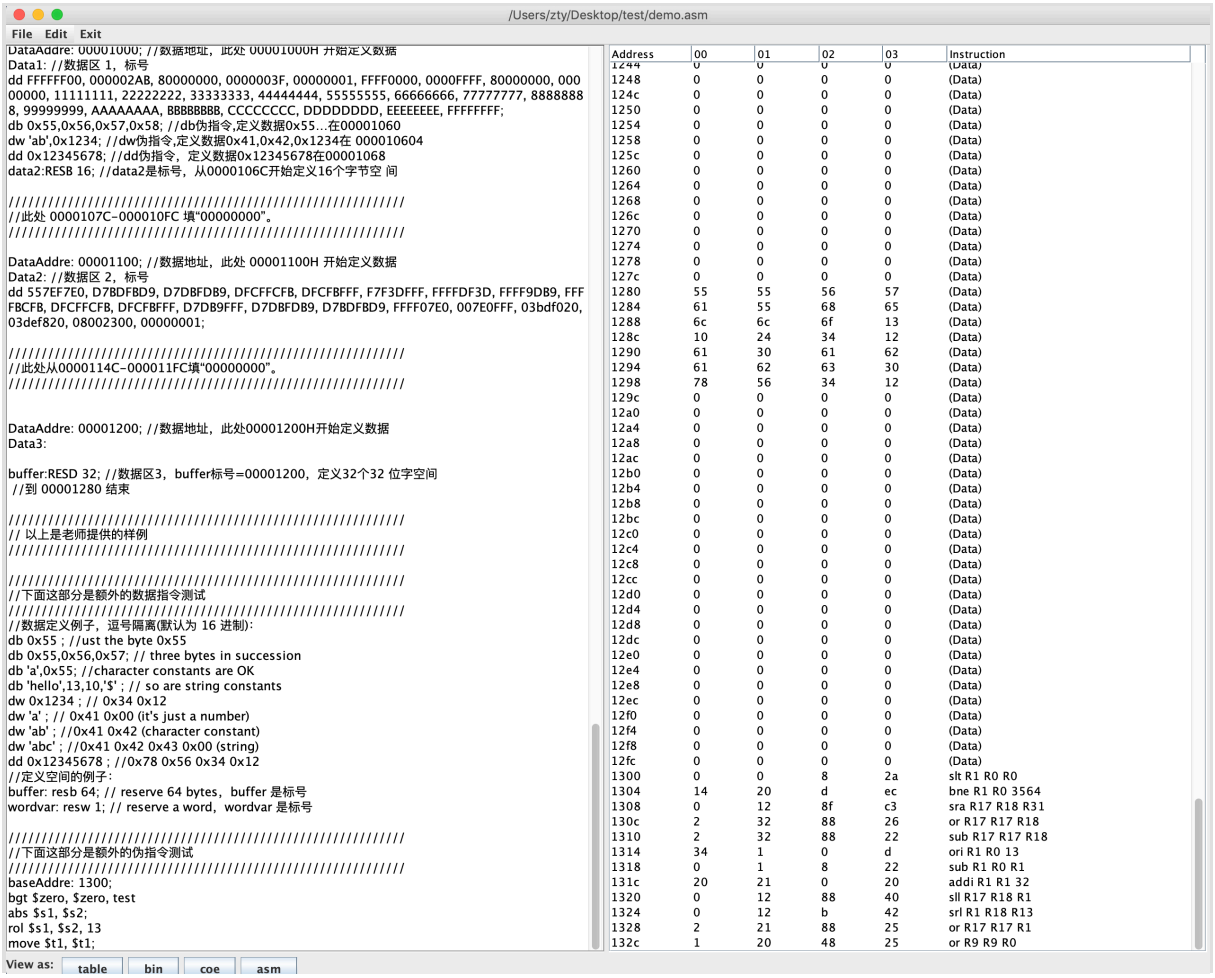
/Users/zty/Desktop/test/demo.asm

File Edit Exit

```
j i_next;

////////////////////////////////////////////////////////////////
//从此处-00000FFC 填"00000000"。
////////////////////////////////////////////////////////////////

DataAddre: 00001000; //数据地址，此处 00001000H 开始定义数据
Data1: //数据区 1，标号
dd FFFFFF00, 000002AB, 80000000, 0000003F, 00000001, FFFF0000, 0000FFFF, 80000000, 000
00000, 11111111, 22222222, 33333333, 44444444, 55555555, 66666666, 77777777, 8888888
8, 99999999, AAAAAAAA, BBBBBBBB, CCCCCCCC, DDDDDDDD, EEEEEEEE, FFFFFFFF;
db 0x55,0x56,0x57,0x58; //db伪指令,定义数据0x55…在00001060
dw 'ab',0x1234; //dw伪指令,定义数据0x41,0x42,0x1234在 000010604
dd 0x12345678; //dd伪指令，定义数据0x12345678在00001068
data2:RESB 16; //data2是标号，从0000106C开始定义16个字节空 间

////////////////////////////////////////////////////////////////
//此处 0000107C-000010FC 填"00000000"。
////////////////////////////////////////////////////////////////

DataAddre: 00001100; //数据地址，此处 00001100H 开始定义数据
Data2: //数据区 2，标号
dd 557EF7E0, D7BDFBD9, D7DBFBD9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FFF
FBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020,
03def820, 08002300, 00000001;

////////////////////////////////////////////////////////////////
//此处从0000114C-000011FC填"00000000"。
////////////////////////////////////////////////////////////////


DataAddre: 00001200; //数据地址，此处00001200H开始定义数据
Data3:

buffer:RESD 32; //数据区3，buffer标号=00001200，定义32个32 位字空间
 //到 00001280 结束

////////////////////////////////////////////////////////////////
// 以上是老师提供的样例
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
//下面这部分是额外的数据指令测试
////////////////////////////////////////////////////////////////
//数据定义例子，逗号隔离(默认为 16 进制):
db 0x55 ; //ust the byte 0x55
db 0x55,0x56,0x57; // three bytes in succession
db 'a',0x55; //character constants are OK
db 'hello',13,10,'$' ; // so are string constants
dw 0x1234 ; // 0x34 0x12
dw 'a' ; // 0x41 0x00 (it's just a number)
dw 'ab' ; //0x41 0x42 (character constant)
dw 'abc' ; //0x41 0x42 0x43 0x00 (string)
dd 0x12345678 ; //0x78 0x56 0x34 0x12
//定义空间的例子：
buffer: resb 64; // reserve 64 bytes，buffer 是标号
wordvar: resw 1; // reserve a word，wordvar 是标号

////////////////////////////////////////////////////////////////
//下面这部分是额外的伪指令测试
```

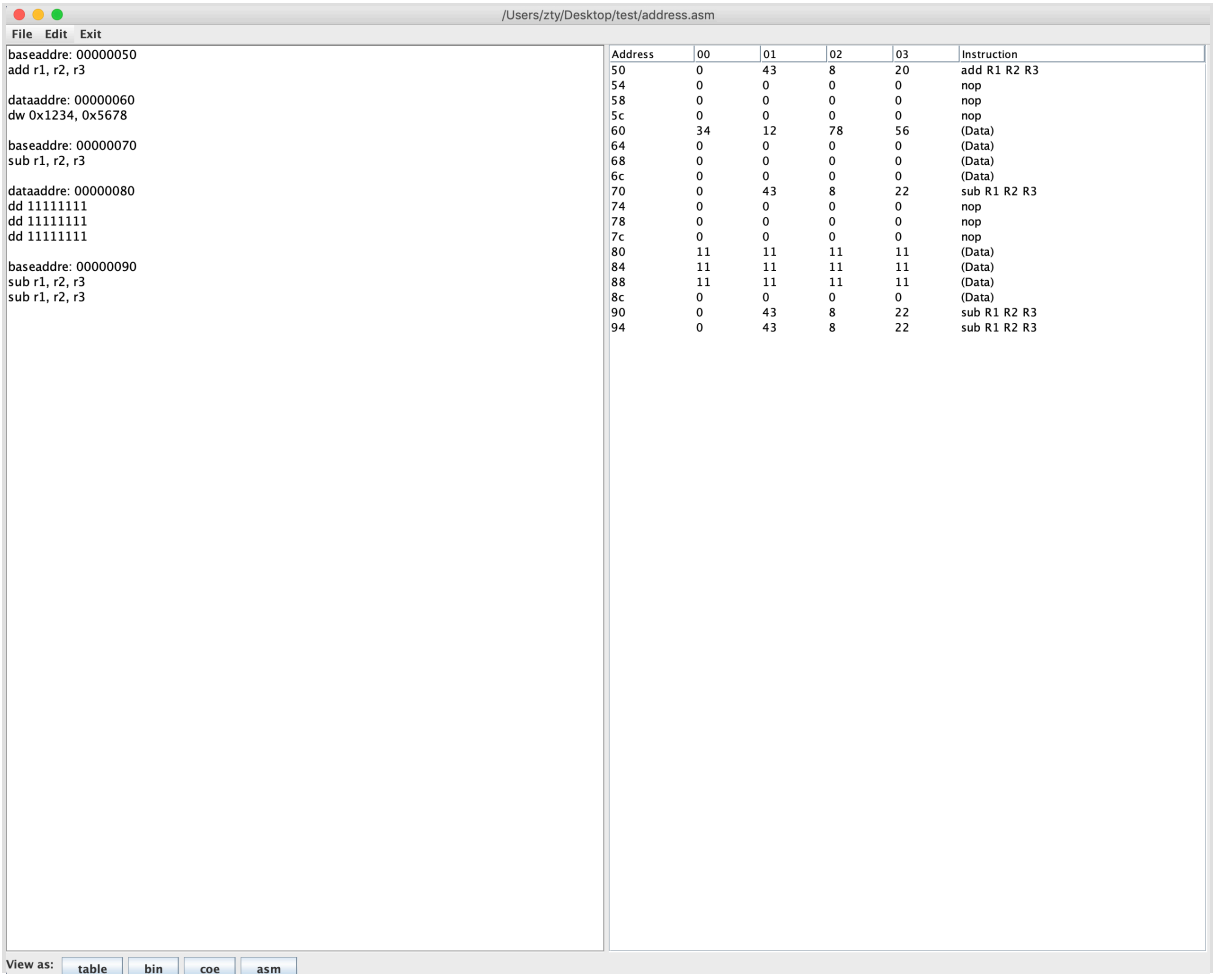
Address	00	01	02	03	Instruction
fe4	8	0	f	4c	j 3916
fe8	0	0	0	0	nop
fec	0	0	0	0	nop
ff0	0	0	0	0	nop
ff4	0	0	0	0	nop
ff8	0	0	0	0	nop
ffc	0	0	0	0	nop
1000	0	ff	ff	ff	(Data)
1004	ab	2	0	0	(Data)
1008	0	0	0	80	(Data)
100c	3f	0	0	0	(Data)
1010	1	0	0	0	(Data)
1014	0	0	ff	ff	(Data)
1018	ff	ff	0	0	(Data)
101c	0	0	0	80	(Data)
1020	0	0	0	0	(Data)
1024	11	11	11	11	(Data)
1028	22	22	22	22	(Data)
102c	33	33	33	33	(Data)
1030	44	44	44	44	(Data)
1034	55	55	55	55	(Data)
1038	66	66	66	66	(Data)
103c	77	77	77	77	(Data)
1040	88	88	88	88	(Data)
1044	99	99	99	99	(Data)
1048	aa	aa	aa	aa	(Data)
104c	bb	bb	bb	bb	(Data)
1050	cc	cc	cc	cc	(Data)
1054	dd	dd	dd	dd	(Data)
1058	ee	ee	ee	ee	(Data)
105c	ff	ff	ff	ff	(Data)
1060	55	56	57	58	(Data)
1064	61	62	34	12	(Data)
1068	78	56	34	12	(Data)
106c	0	0	0	0	(Data)
1070	0	0	0	0	(Data)
1074	0	0	0	0	(Data)
1078	0	0	0	0	(Data)
107c	0	0	0	0	(Data)
1080	0	0	0	0	(Data)
1084	0	0	0	0	(Data)
1088	0	0	0	0	(Data)
108c	0	0	0	0	(Data)
1090	0	0	0	0	(Data)
1094	0	0	0	0	(Data)
1098	0	0	0	0	(Data)
109c	0	0	0	0	(Data)
10a0	0	0	0	0	(Data)
10a4	0	0	0	0	(Data)
10a8	0	0	0	0	(Data)
10ac	0	0	0	0	(Data)
10b0	0	0	0	0	(Data)
10b4	0	0	0	0	(Data)
10b8	0	0	0	0	(Data)
10bc	0	0	0	0	(Data)
10c0	0	0	0	0	(Data)
10c4	0	0	0	0	(Data)
10c8	0	0	0	0	(Data)
10cc	0	0	0	0	(Data)

View as: table bin coe asm



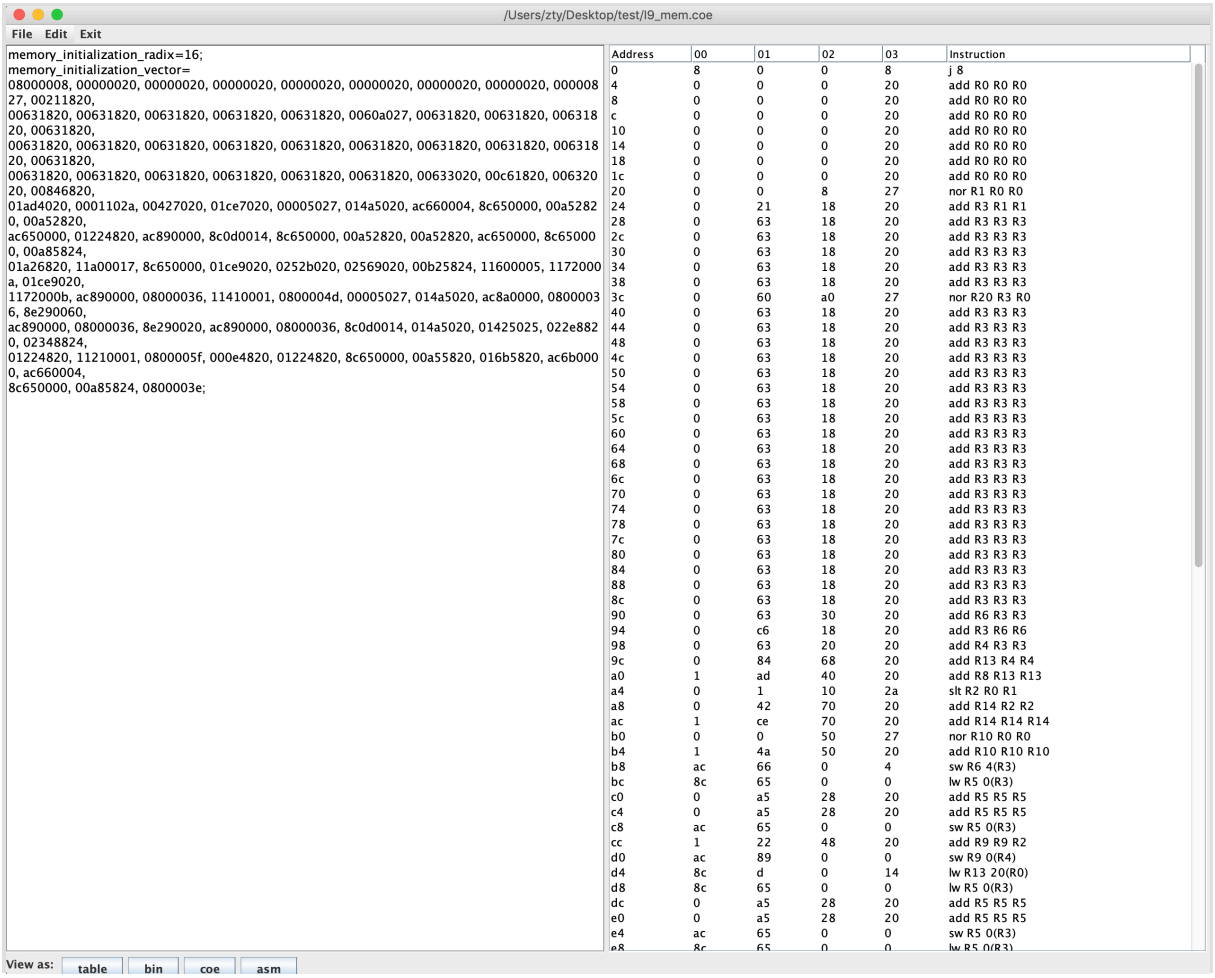
address.asm

```
baseaddre: 00000050
add r1, r2, r3

dataaddre: 00000060
dw 0x1234, 0x5678

baseaddre: 00000070
sub r1, r2, r3

dataaddre: 00000080
dd 11111111
dd 11111111
dd 11111111

baseaddre: 00000090
sub r1, r2, r3
sub r1, r2, r3
```



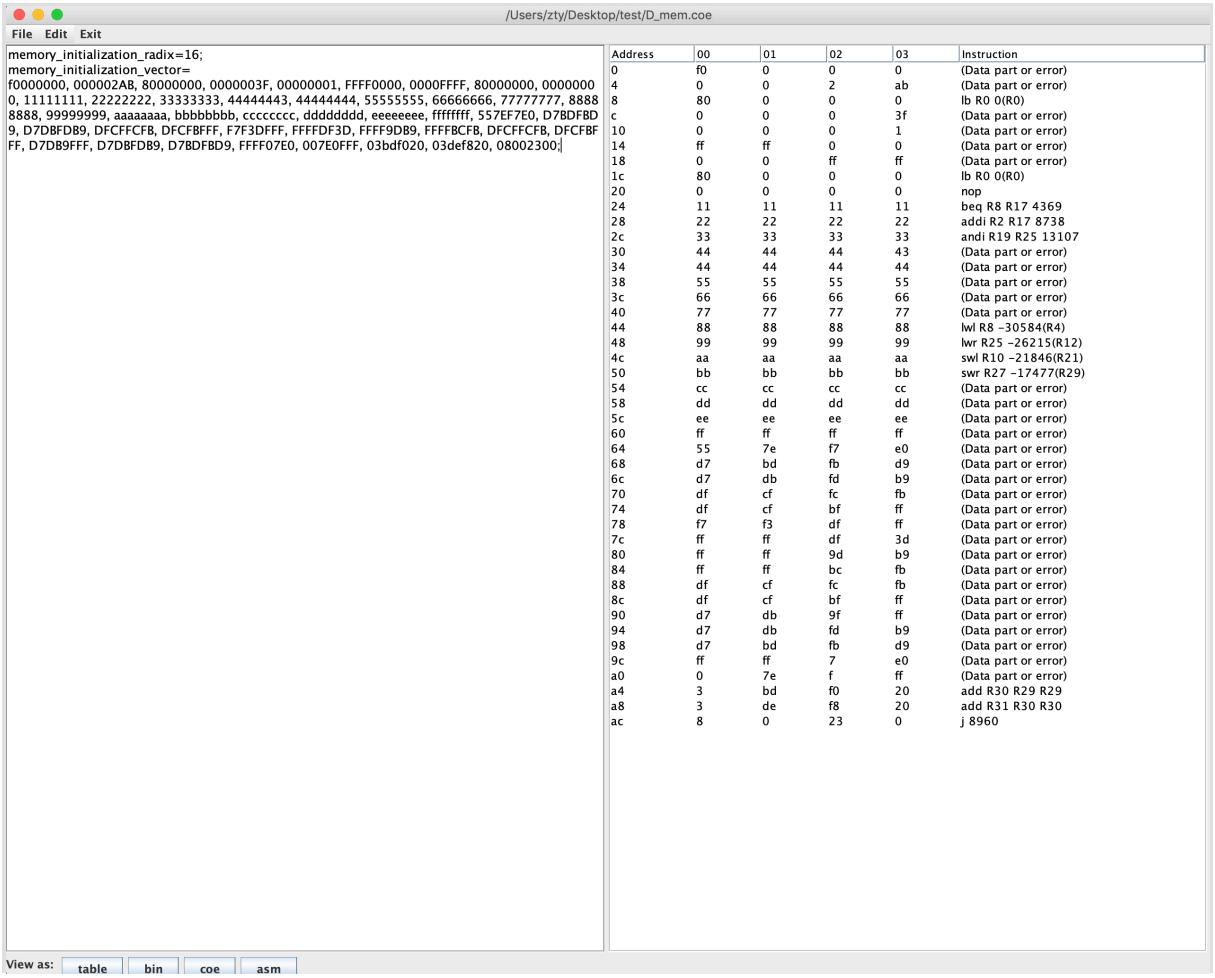
I9_mem.coe

```
memory_initialization_radix=16;
memory_initialization_vector=
08000008,  00000020,  00000020,  00000020,  00000020,  00000020,  00000020,
00000020, 00000827, 00211820,
00631820,  00631820,  00631820,  00631820,  00631820,  0060a027,  00631820,
00631820, 00631820, 00631820,
00631820,  00631820,  00631820,  00631820,  00631820,  00631820,  00631820,
00631820, 00631820, 00631820,
00631820,  00631820,  00631820,  00631820,  00631820,  00631820,  00633020,
00c61820, 00632020, 00846820,
01ad4020,  0001102a,  00427020,  01ce7020,  00005027,  014a5020,  ac660004,
8c650000, 00a52820, 00a52820,
ac650000,  01224820,  ac890000,  8c0d0014,  8c650000,  00a52820,  00a52820,
ac650000, 8c650000, 00a85824,
01a26820,  11a00017,  8c650000,  01ce9020,  0252b020,  02569020,  00b25824,
11600005, 1172000a, 01ce9020,
1172000b,  ac890000,  08000036,  11410001,  0800004d,  00005027,  014a5020,
ac8a0000, 08000036, 8e290060,
ac890000,  08000036,  8e290020,  ac890000,  08000036,  8c0d0014,  014a5020,
01425025, 022e8820, 02348824,
01224820,  11210001,  0800005f,  000e4820,  01224820,  8c650000,  00a55820,
016b5820, ac6b0000, ac660004,
8c650000, 00a85824, 0800003e;
```



D_mem.coe

```
memory_initialization_radix=16;
memory_initialization_vector=
f0000000,  000002AB,  80000000,  0000003F,  00000001,  FFFF0000,  0000FFFF,
80000000,  00000000,  11111111,  22222222,  33333333,  44444444,  55555555,
66666666,  77777777,  88888888,  99999999,  aaaaaaaa,  bbbbbbbb,  cccccccc,
dddddddd,  eeeeeeee,  ffffffff,  557EF7E0,  D7BDFBD9,  D7DBFDB9,  DFCFFCFB,
DFCFBFFF,  F7F3DFFF,  FFFFDF3D,  FFFF9DB9,  FFFFBCFB,  DFCFFCFB,  DFCFBFFF,
D7DB9FFF,  D7DBFDB9,  D7BDFBD9,  FFFF07E0,  007E0FFF,  03bdf020,  03def820,
08002300;
```



sum_of_quare_100.bin

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111110010000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

00000000000000000001000000100001