

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。

/Users/zty/Desktop/test/address.asm

File Edit Exit

```

add r1, r2, r3
dataaddr: 00000060
dw 0x1234, 0x5678

baseaddr: 00000070
sub r1, r2, r3

dataaddr: 00000080
dd 11111111
dd 11111111
dd 11111111

baseaddr: 00000090
sub r1, r2, r3
sub r1, r2, r3

```

Address	00	01	02	03	Instruction
0	0	43	8	20	add R1 R2 R3
4	0	0	0	0	nop
8	0	0	0	0	nop
12	0	0	0	0	nop
16	0	0	0	0	nop
20	0	0	0	0	nop
24	0	0	0	0	nop
28	0	0	0	0	nop
32	0	0	0	0	nop
36	0	0	0	0	nop
40	0	0	0	0	nop
44	0	0	0	0	nop
48	0	0	0	0	nop
52	0	0	0	0	nop
56	0	0	0	0	nop
60	0	0	0	0	nop
64	0	0	0	0	nop
68	0	0	0	0	nop
72	0	0	0	0	nop
76	0	0	0	0	nop
80	0	0	0	0	nop
84	0	0	0	0	nop
88	0	0	0	0	nop
92	0	0	0	0	nop
96	34	12	78	56	(Data)
100	0	0	0	0	(Data)
104	0	0	0	0	(Data)
108	0	0	0	0	(Data)
112	0	43	8	22	sub R1 R2 R3
116	0	0	0	0	nop
120	0	0	0	0	nop
124	0	0	0	0	nop
128	11	11	11	11	(Data)
132	11	11	11	11	(Data)
136	11	11	11	11	(Data)
140	0	0	0	0	(Data)
144	0	43	8	22	sub R1 R2 R3
148	0	43	8	22	sub R1 R2 R3

View as: [table](#) [bin](#) [coe](#) [asm](#)

图 测试地址指令：默认起始地址为 0

/Users/zty/Desktop/test/address.asm

Address	00	01	02	03	Instruction
80	0	43	8	20	add R1 R2 R3
84	0	0	0	0	nop
88	0	0	0	0	nop
92	0	0	0	0	nop
96	34	12	78	56	(Data)
100	0	0	0	0	(Data)
104	0	0	0	0	(Data)
108	0	0	0	0	(Data)
112	0	43	8	22	sub R1 R2 R3
116	0	0	0	0	nop
120	0	0	0	0	nop
124	0	0	0	0	nop
128	11	11	11	11	(Data)
132	11	11	11	11	(Data)
136	11	11	11	11	(Data)
140	0	0	0	0	(Data)
144	0	43	8	22	sub R1 R2 R3
148	0	43	8	22	sub R1 R2 R3

View as: [table](#) [bin](#) [coe](#) [asm](#)

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]00000[01][01][01][01]00000001001
```

表示匹配正则表达式

“000000[01][01][01][01]00000[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

The screenshot shows a memory editor window with the following details:

- Title Bar:** /Users/zty/Desktop/test/sum_of_square_100.bin
- Menu Bar:** File Edit Exit
- Table View:** A large table showing memory dump data. The columns are Address, 00, 01, 02, 03, and Instruction.
- Binary Data:** The left side of the table displays binary values in groups of four bytes (e.g., 00100111101111101111111111111111000000).
- Assembly Instructions:** The right side of the table lists assembly instructions corresponding to the binary data. For example:
 - Address 0: 27 bd ff e0 addiu R29 R29 -32
 - Address 4: af bf 0 14 sw R31 20(R29)
 - Address 8: af a4 0 20 sw R4 32(R29)
 - Address 12: af a5 0 24 sw R5 36(R29)
 - Address 16: af a0 0 18 sw R0 24(R29)
 - Address 20: af a0 0 1c sw R0 28(R29)
 - Address 24: 8f ae 0 1c lw R14 28(R29)
 - Address 28: 8f b8 0 18 lw R24 24(R29)
 - Address 32: 1 ce 0 19 multu R14 R14
 - Address 36: 25 c8 0 1 addiu R8 R14 1
 - Address 40: 29 1 0 65 slti R1 R8 101
 - Address 44: af a8 0 1c sw R8 28(R29)
 - Address 48: 0 0 78 12 mflo R15
 - Address 52: 3 f c8 21 addu R25 R24 R15
 - Address 56: 14 20 ff f7 bne R1 R0 -9
 - Address 60: af b9 0 18 sw R25 24(R29)
 - Address 64: 3c 4 10 0 lui R4 4096
 - Address 68: 8f a5 0 18 lw R5 24(R29)
 - Address 72: c 10 0 ec jal 1048812
 - Address 76: 24 84 4 30 addiu R4 R4 1072
 - Address 80: 8f bf 0 14 lw R31 20(R29)
 - Address 84: 27 bd 0 20 addiu R29 R29 32
 - Address 88: 3 e0 0 8 jr R31
 - Address 92: 0 0 10 21 addu R2 R0 R0
- View Options:** At the bottom, there are buttons for View as: table, bin, coe, and asm. The 'asm' button is highlighted.

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=FFFFFFF  
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=FFFFFFF80
add $v1, $v1, $v1; //r3=FFFFFFF00

add $v1, $v1, $v1; //r3=FFFFFE000
add $v1, $v1, $v1; //r3=FFFFFC000
add $v1, $v1, $v1; //r3=FFFFF8000
add $v1, $v1, $v1; //r3=FFFFF0000

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=FFFFFF
add $t2, $t2, $t2; //r10=FFFFFFE

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=0

```

```

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=ffffffffff

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=ffffffffc, 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-0000FFC 填“0000000”。
///////////////////////////////

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, BBBB BBBB, 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“0000000”。
///////////////////////////////

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“0000000”。
///////////////////////////////

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

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
fbcc	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	nnn

View as: table | bin | coe | asm

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

File	Edit	Exit
beq \$t3, \$zero, C_init; //程序计数 r15=0,转计数器初始化,修改 / 段码显示:C_init		
<pre> 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=fffffe 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=fffffc, 7 段图形点左移 or \$t2, \$t2, \$v0; //r10 未位置 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=0000000X, 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 f14 0 a5 28 20 f18 0 a5 28 20 f1c ac 65 0 0 f20 1 22 48 20 f24 ac 89 0 0 f28 8c d 0 e f2c 8c 65 0 0 f30 0 a5 28 20 f34 0 a5 28 20 f38 ac 65 0 0 f3c 8c 65 0 0 f40 0 a8 58 24 f44 1 a2 68 20 f48 11 a0 f a8 f4c 8c 65 0 0 f50 1 ce 90 20 f54 2 52 b0 20 f58 2 56 90 20 f5c 0 b2 58 24 f60 11 60 f 78 f64 11 72 f 90 f68 1 ce 90 20 f6c 11 72 f 9c f70 ac 89 0 0 f74 8 0 f 2c f78 11 41 f 80 f7c 8 0 f 88 f80 0 0 50 27 f84 1 4a 50 20 f88 ac 8a 0 0 f8c 8 0 f 2c f90 8e 29 0 3c f94 ac 89 0 0 f98 8 0 f 2c f9c 8e 29 0 14 fa0 ac 89 0 0 fa4 8 0 f 2c fa8 8c d 0 e fac 1 4a 50 20 fb0 1 42 50 25 fb4 2 2e 88 20 fb8 2 34 88 24 fbfc 1 22 48 20 fc0 11 21 f c8 fc4 8 0 f d0 fc8 0 e 48 20 fcc 1 22 48 20 fd0 8c 65 0 0 fd4 0 a5 58 20 fd8 1 6b 58 20 fdc ac 6b 0 0 fe0 ac 66 0 4 fe4 8 0 f 4c fe8 0 0 0 0 fer 0 0 0 0 </pre>		

View as: [table](#) | [bin](#) | [coe](#) | [asm](#)

File Edit Exit /Users/zty/Desktop/test/demo.asm

```

; 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 FFFF00_000002AB, 80000000, 0000003F, 00000001, FFFF0000, 0000FFFF, 80000000, 000
0000, 11111111, 22222222, 33333333, 44444444, 55555555, 66666666, 77777777, 8888888
8, 99999999, AAAAAA, BBBB8888, 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
dataZ: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 57EEF7E0, D7DBFD89, D7DBFD89, DFCFFCF8, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FF
FBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFD89, D7DBFD89, 0FFF07E0, 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; //use 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

File Edit Exit

```
DataAddre: UUUU1UUU; //数据地址, 此处 UUUU1UUUH 为暂定义数据
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, 88888888
8, 99999999, AAAAAA, BBBB8888, CCCCCCCC, DDDDDDD, EEEEEEE, FFFFFFF;
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为标号, 从00000106C开始定义16个字节空间
DataAddre: 00001100; //数据地址, 此处 00001100H 开始定义数据
Data2: //数据区 2, 标号
dd 557EF7E0, D7DBFDB9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FFF
FBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF, D7DBFDB9, D7DBFDB9, 007E0FF, 03bd0f020,
03def820, 08002300, 00000001;
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, 逗号隔开(默认认为 16 进制):
db 0x55; //us 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
baseAddre: 1300;
bgt $zero, $zero, test
abs $s1, $s2;
rol $s1, $s2, 13
move $t1, $t1;
```

Address	00	01	02	03	Instruction
1244	0	0	0	0	(Data)
1248	0	0	0	0	(Data)
124c	0	0	0	0	(Data)
1250	0	0	0	0	(Data)
1254	0	0	0	0	(Data)
1258	0	0	0	0	(Data)
125c	0	0	0	0	(Data)
1260	0	0	0	0	(Data)
1264	0	0	0	0	(Data)
1268	0	0	0	0	(Data)
126c	0	0	0	0	(Data)
1270	0	0	0	0	(Data)
1274	0	0	0	0	(Data)
1278	0	0	0	0	(Data)
127c	0	0	0	0	(Data)
1280	55	55	56	57	(Data)
1284	61	55	68	65	(Data)
1288	6c	6c	6f	13	(Data)
128c	10	24	34	12	(Data)
1290	61	30	61	62	(Data)
1294	61	62	63	30	(Data)
1298	78	56	34	12	(Data)
129c	0	0	0	0	(Data)
12a0	0	0	0	0	(Data)
12a4	0	0	0	0	(Data)
12a8	0	0	0	0	(Data)
12ac	0	0	0	0	(Data)
12b0	0	0	0	0	(Data)
12b4	0	0	0	0	(Data)
12b8	0	0	0	0	(Data)
12bc	0	0	0	0	(Data)
12c0	0	0	0	0	(Data)
12c4	0	0	0	0	(Data)
12c8	0	0	0	0	(Data)
12cc	0	0	0	0	(Data)
12d0	0	0	0	0	(Data)
12d4	0	0	0	0	(Data)
12d8	0	0	0	0	(Data)
12dc	0	0	0	0	(Data)
12e0	0	0	0	0	(Data)
12e4	0	0	0	0	(Data)
12e8	0	0	0	0	(Data)
12ec	0	0	0	0	(Data)
12f0	0	0	0	0	(Data)
12f4	0	0	0	0	(Data)
12f8	0	0	0	0	(Data)
12fc	0	0	0	0	(Data)
1300	0	0	8	2a	slt R1 R0 R0
1304	14	20	d	ec	bne R1 R0 3564
1308	0	12	8f	c3	sra R17 R18 R31
130c	2	32	88	26	or R17 R17 R18
1310	2	32	88	22	sub R17 R17 R18
1314	34	1	0	d	ori R1 R0 13
1318	0	1	8	22	sub R1 R0 R1
131c	20	21	0	20	addi R1 R1 32
1320	0	12	88	40	sll R17 R18 R1
1324	0	12	b	42	srl R1 R18 R13
1328	2	21	88	25	or R17 R17 R1
132c	1	20	48	25	or R9 R9 R0

View as: [table](#) [bin](#) [coe](#) [asm](#)

address.asm

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

dataaddr: 00000060
dw 0x1234, 0x5678

baseaddr: 00000070
sub r1, r2, r3

dataaddr: 00000080
dd 11111111
dd 11111111
dd 11111111

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

File Edit Exit /Users/zty/Desktop/test/address.asm

	Address	00	01	02	03	Instruction
baseaddr: 00000050 add r1, r2, r3	50	0	43	8	20	add R1 R2 R3
dataaddr: 00000060 dw 0x1234, 0x5678	54	0	0	0	0	nop
baseaddr: 00000070 sub r1, r2, r3	58	0	0	0	0	nop
dataaddr: 00000080 dd 11111111 dd 11111111 dd 11111111	5c	0	0	0	0	nop
baseaddr: 00000090 sub r1, r2, r3 sub r1, r2, r3	60	34	12	78	56	(Data)
	64	0	0	0	0	(Data)
	68	0	0	0	0	(Data)
	6c	0	0	0	0	(Data)
	70	0	43	8	22	sub R1 R2 R3
	74	0	0	0	0	nop
	78	0	0	0	0	nop
	7c	0	0	0	0	nop
	80	11	11	11	11	(Data)
	84	11	11	11	11	(Data)
	88	11	11	11	11	(Data)
	8c	0	0	0	0	(Data)
	90	0	43	8	22	sub R1 R2 R3
	94	0	43	8	22	sub R1 R2 R3

View as: [table](#) [bin](#) [coe](#) [asm](#)

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, 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;
```

/Users/zty/Desktop/test/l9_mem.coe

Address	00	01	02	03	Instruction
0	8	0	0	8	j 8
4	0	0	0	20	add R0 R0 R0
8	0	0	0	20	add R0 R0 R0
c	0	0	0	20	add R0 R0 R0
10	0	0	0	20	add R0 R0 R0
14	0	0	0	20	add R0 R0 R0
18	0	0	0	20	add R0 R0 R0
1c	0	0	0	20	add R0 R0 R0
20	0	0	8	27	nor R1 R0 R0
24	0	21	18	20	add R3 R1 R1
28	0	63	18	20	add R3 R3 R3
2c	0	63	18	20	add R3 R3 R3
30	0	63	18	20	add R3 R3 R3
34	0	63	18	20	add R3 R3 R3
38	0	63	18	20	add R3 R3 R3
3c	0	60	a0	27	nor R20 R3 R0
40	0	63	18	20	add R3 R3 R3
44	0	63	18	20	add R3 R3 R3
48	0	63	18	20	add R3 R3 R3
50	0	63	18	20	add R3 R3 R3
54	0	63	18	20	add R3 R3 R3
58	0	63	18	20	add R3 R3 R3
5c	0	63	18	20	add R3 R3 R3
60	0	63	18	20	add R3 R3 R3
64	0	63	18	20	add R3 R3 R3
68	0	63	18	20	add R3 R3 R3
6c	0	63	18	20	add R3 R3 R3
70	0	63	18	20	add R3 R3 R3
74	0	63	18	20	add R3 R3 R3
78	0	63	18	20	add R3 R3 R3
7c	0	63	18	20	add R3 R3 R3
80	0	63	18	20	add R3 R3 R3
84	0	63	18	20	add R3 R3 R3
88	0	63	18	20	add R3 R3 R3
8c	0	63	18	20	add R3 R3 R3
90	0	63	30	20	add R6 R3 R3
94	0	c6	18	20	add R3 R6 R6
98	0	63	20	20	add R4 R3 R3
9c	0	84	68	20	add R13 R4 R4
a0	1	ad	40	20	add R8 R13 R13
a4	0	1	10	2a	slt R2 R0 R1
a8	0	42	70	20	add R14 R2 R2
ac	1	ce	70	20	add R14 R14 R14
b0	0	0	50	27	nor R10 R0 R0
b4	1	4a	50	20	add R10 R10 R10
b8	ac	66	0	4	sw R6 4(R3)
bc	8c	65	0	0	lw R5 0(R3)
c0	0	a5	28	20	add R5 R5 R5
c4	0	a5	28	20	add R5 R5 R5
c8	ac	65	0	0	sw R5 0(R3)
cc	1	22	48	20	add R9 R9 R2
d0	ac	89	0	0	sw R9 0(R4)
d4	8c	d	0	14	lw R13 20(R0)
d8	8c	65	0	0	lw R5 0(R3)
dc	0	a5	28	20	add R5 R5 R5
e0	0	a5	28	20	add R5 R5 R5
e4	ac	65	0	0	sw R5 0(R3)
ea	8c	65	0	0	lw R5 0(R3)

View as: [table](#) [bin](#) [coe](#) [asm](#)

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, aaaaaaaaaa, bbbbbbbb, cccccccc,
dddddddd, eeeeeeee, ffffffff, 557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFC9B,
DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFC9B, DFCFBFFF,
D7DB9FFF, D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020, 03def820,
08002300;
```

File Edit Exit

memory_initialization_radix=16;
memory_initialization_vector=
f0000000, 000002AB, 80000000, 0000003F, 00000001, FFFF0000, 0000FFFF, 80000000, 00000000
0, 11111111, 22222222, 33333333, 44444443, 44444444, 55555555, 66666666, 77777777, 8888
8888, 99999999, aaaaaaaaaa, bbbbbbbb, cccccccc, dddddd, eeeeeeee, ffffffff, 557EF7E0, D7BDFBD
FF, D7DBFB89, DFCFFCFB, DFCFBFFF, F7F3DFFF, FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFCFB, DFCFBF
FF, D7DB9FFF, D7DBFB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020, 03def820, 08002300;]

Address	00	01	02	03	Instruction
0	f0	0	0	0	(Data part or error)
4	0	0	2	ab	(Data part or error)
8	80	0	0	0	lb R0 0(R0)
c	0	0	0	0	3f (Data part or error)
10	0	0	0	1	(Data part or error)
14	ff	ff	0	0	(Data part or error)
18	0	0	ff	ff	(Data part or error)
1c	80	0	0	0	lb R0 0(R0)
20	0	0	0	0	nop
24	11	11	11	11	beq R8 R17 4369
28	22	22	22	22	addi R2 R17 8738
2c	33	33	33	33	andi R19 R25 13107
30	44	44	44	43	(Data part or error)
34	44	44	44	44	(Data part or error)
38	55	55	55	55	(Data part or error)
3c	66	66	66	66	(Data part or error)
40	77	77	77	77	(Data part or error)
44	88	88	88	88	lw R8 -30584(R4)
48	99	99	99	99	lw R25 -26215(R12)
4c	aa	aa	aa	aa	swl R10 -21846(R21)
50	bb	bb	bb	bb	swr R27 -17477(R29)
54	cc	cc	cc	cc	(Data part or error)
58	dd	dd	dd	dd	(Data part or error)
5c	ee	ee	ee	ee	(Data part or error)
60	ff	ff	ff	ff	(Data part or error)
64	55	7e	f7	e0	(Data part or error)
68	d7	bd	fb	d9	(Data part or error)
6c	d7	db	fd	b9	(Data part or error)
70	df	cf	fc	fb	(Data part or error)
74	df	cf	bf	ff	(Data part or error)
78	f7	f3	df	ff	(Data part or error)
7c	ff	ff	df	3d	(Data part or error)
80	ff	ff	9d	b9	(Data part or error)
84	ff	ff	bc	fb	(Data part or error)
88	df	cf	fc	fb	(Data part or error)
8c	df	cf	bf	ff	(Data part or error)
90	d7	db	9f	ff	(Data part or error)
94	d7	db	fd	b9	(Data part or error)
98	d7	bd	fb	d9	(Data part or error)
9c	ff	ff	7	e0	(Data part or error)
a0	0	7e	f	ff	(Data part or error)
a4	3	bd	f0	20	add R30 R29 R29
a8	3	de	f8	20	add R31 R30 R30
ac	8	0	23	0	j 8960

sum of square 100.bin

