

MIPS汇编与PCSpim模拟器简单说明

先通过简单的MIPS汇编语言程序例子，初步来认识MIPS汇编语言的程序结构，以及它的编程风格，然后说明PCSpim模拟器界面结构及使用方法，再用例子来说明字符串和数据在存储器中的存储情况。

一、MIPS汇编语言程序简单结构说明

一个简单MIPS汇编程序。以下程序输出字符串“hello world”，从中认识一下MIPS汇编语言程序结构，看看编程风格。文件名：helloworld.asm。

```
.text                # 代码段 声明

.globl main          # globl指明程序的入口地址main，为全局名

main:               # 主程序名： main

    la $a0, str      # 取字符串地址
    li $v0, 4         # 4号功能调用，输出字符串
    syscall          # 系统调用，输出字符串

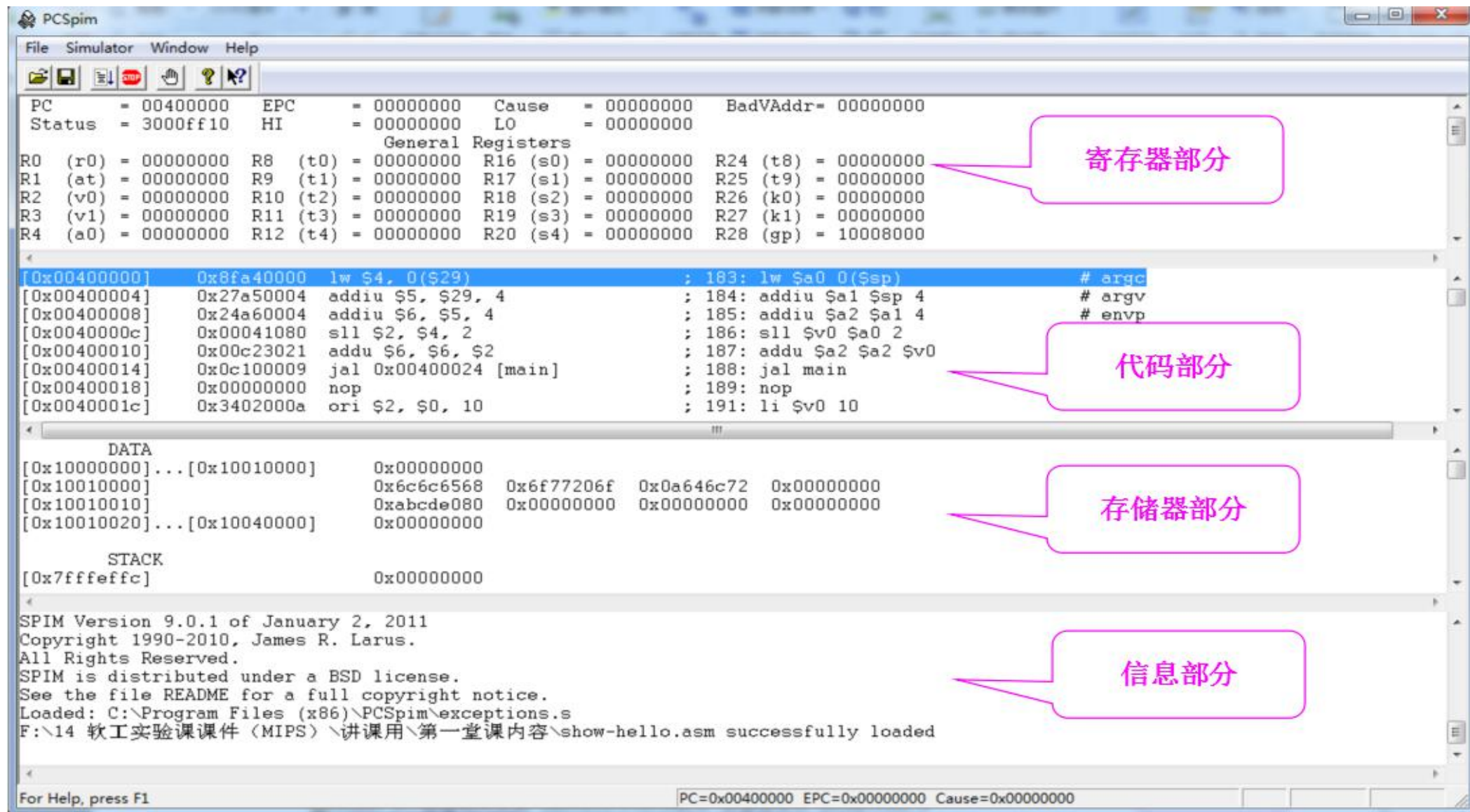
    li $v0, 10        # 退出
    syscall          # 系统调用，

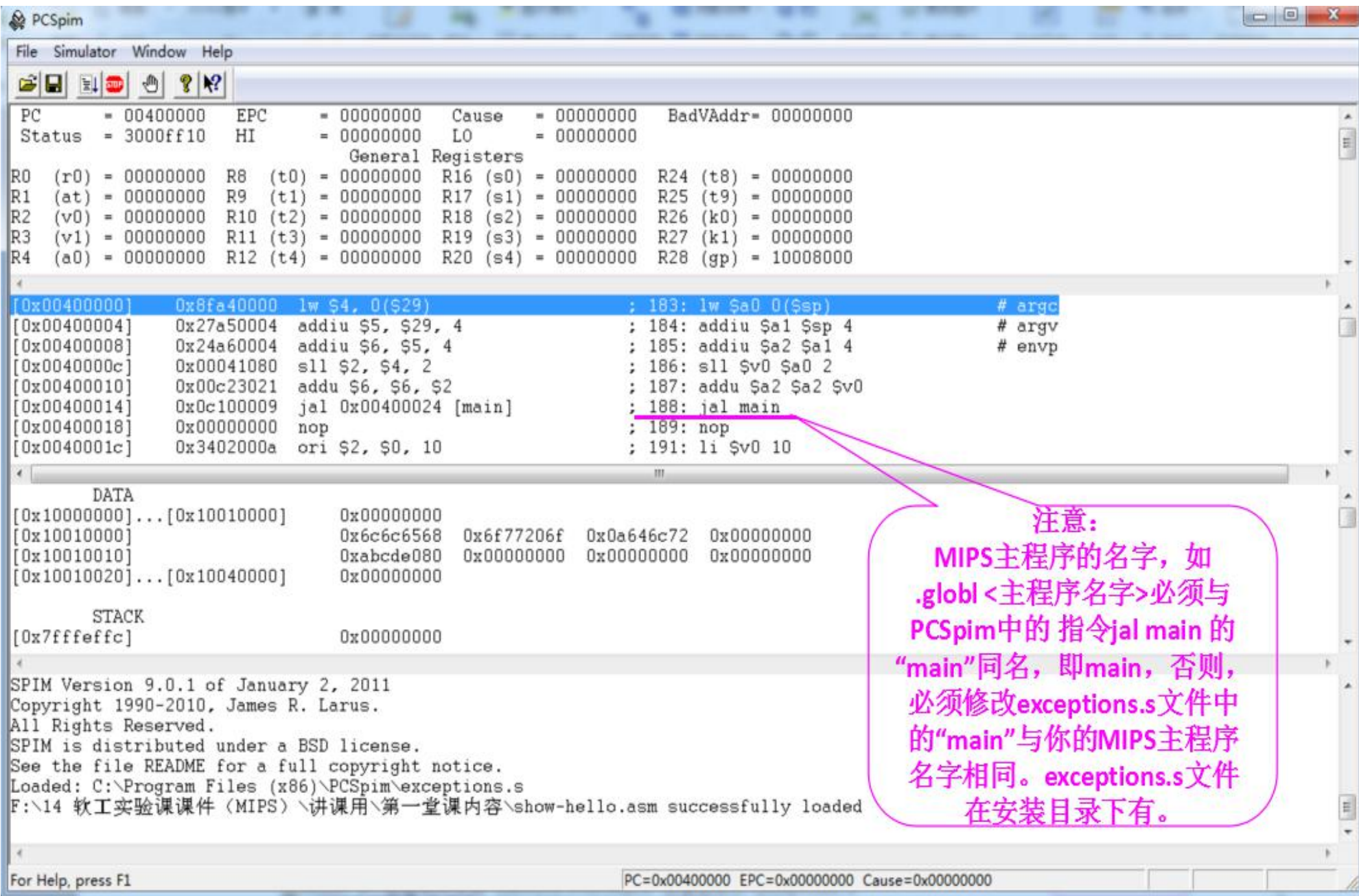
.data               # 数据段，变量定义部分

str:               # 字符串变量名称

    .ascii "hello world\n" # 该字符串为 str 的初值，以“00”为终止符结束。
```

二、PCSpim 模拟器界面简单说明





PCSpim

File Simulator Window Help

PC = 00400000 EPC = 00000000 Cause = 00000000 BadVAddr = 00000000
Status = 3000ff10 HI = 00000000 LO = 00000000

General Registers

R0 (r0) = 00000000	R8 (t0) = 00000000	R16 (s0) = 00000000	R24 (t8) = 00000000
R1 (at) = 00000000	R9 (t1) = 00000000	R17 (s1) = 00000000	R25 (t9) = 00000000
R2 (v0) = 00000000	R10 (t2) = 00000000	R18 (s2) = 00000000	R26 (k0) = 00000000
R3 (v1) = 00000000	R11 (t3) = 00000000	R19 (s3) = 00000000	R27 (k1) = 00000000
R4 (a0) = 00000000	R12 (t4) = 00000000	R20 (s4) = 00000000	R28 (gp) = 10008000

[0x00400000] 0x8fa40000 lw \$4, 0(\$29) ; 183: lw \$a0 0(\$sp) # argc
[0x00400004] 0x27a50004 addiu \$5, \$29, 4 ; 184: addiu \$a1 \$sp 4 # argv
[0x00400008] 0x24a60004 addiu \$6, \$5, 4 ; 185: addiu \$a2 \$a1 4 # envp
[0x0040000c] 0x00041080 sll \$2, \$4, 2 ; 186: sll \$v0 \$a0 2
[0x00400010] 0x00c23021 addu \$6, \$6, \$2 ; 187: addu \$a2 \$a2 \$v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 188: jal main
[0x00400018] 0x00000000 nop ; 189: nop
[0x0040001c] 0x3402000a ori \$2, \$0, 10 ; 191: li \$v0 10

DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x6c6c6568 0x6f77206f 0x0a646c72 0x00000000
[0x10010010] 0xabcde080 0x00000000 0x00000000 0x00000000
[0x10010020]...[0x10040000] 0x00000000

STACK
[0x7ffffeffc] 0x00000000

SPIM Version 9.0.1 of January 2, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
Loaded: C:\Program Files (x86)\PCSpim\exceptions.s
F:\14 软工实验课课件 (MIPS)\讲课用\第一堂课内容\show-hello.asm successfully loaded

For Help, press F1

PC=0x00400000 EPC=0x00000000 Cause=0x00000000

从这条指令开始执行

执行到本条指令将转到你的主程序执行，
地址为：0x00400024

内存地址

指令代码

MIPS 汇编程序（指令）

MIPS 汇编程序（伪指令）



PCSpim的使用是比较简单的，在这里就不一一写了。在以下对例子的操作过程中，给以说明。

三、一个简单例子，用于说明相关内容

以下程序输出字符串“hello world”，从中认识一下字符串和数据在内存中的存储情况。

```
.text          # 代码段 声明
.globl main    # globl指明程序的入口地址main，为全局名
main:          # 主程序名： main
    la $a0, str    # 取字符串地址
    li $v0, 4      # 4号功能调用，输出字符串
    syscall        # 系统调用，输出字符串

    li $v0, 10     # 退出
    syscall        # 系统调用，

.data          # 数据段，变量定义部分
str:           # 字符串变量名称
    .ascii "hello world\n" # 该字符串为 str 的初值，以“00”为终止符结束。
memdata:      # 数据变量名称，说明：这个数据变量定义在本程序中无意义，只是借用说明一下数据在内存中存储结构！
    .word 0xabcd080 # 数据定义，字长，32位
```

数据段部分用以分析字符串和数据的存储结构。以上的PCSpim界面简介也是用这个例子来说明的。

1、数据段：字符串的存储

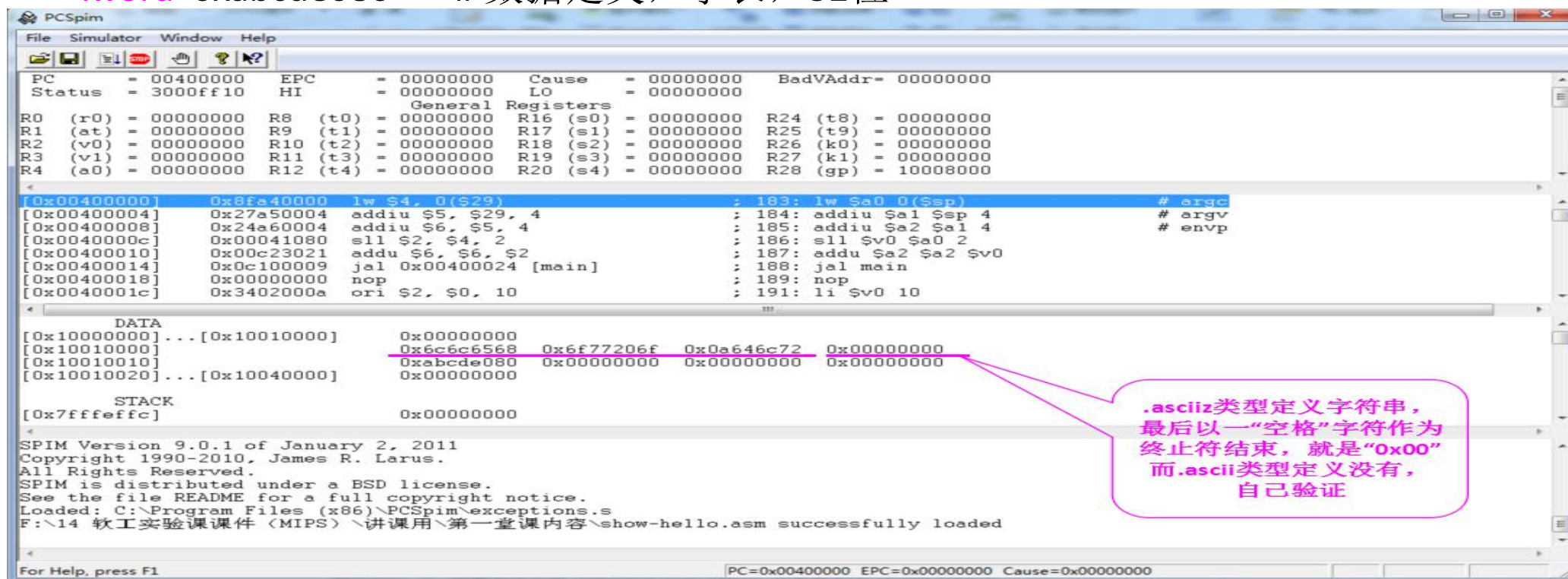
.data # 数据段

str: # 变量名称

.asciiz "hello world\n" # 字符串定义，以“00”为终止符结束。

memdata: # 变量名称，说明：这个数据定义在本程序中无意义，只是借用说明一下数据存储结构！

.word 0xabcde080 # 数据定义，字长，32位



The screenshot shows the PCSpim simulator interface. The assembly code window displays the following code:

```
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 188: jal main
[0x00400018] 0x00000000 nop ; 189: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 191: li $v0 10
```

The DATA window shows the following memory layout:

Address	Value
[0x10000000] ... [0x10010000]	0x00000000
[0x10010000]	0x6c6c6568
[0x10010010]	0xabcde080
[0x10010020] ... [0x10040000]	0x00000000

The STACK window shows the address [0x7ffffffc] with value 0x00000000.

A callout box points to the data segment entry for 'str' (0xabcde080) and contains the following text:

.asciiz类型定义字符串，最后以一“空格”字符作为终止符结束，就是“0x00”而**.ascii**类型定义没有，自己验证

字符串：模拟器中是以8位长度的十六进制数为一个显示表示单位，但存储器中存储是以字节为单位，即一个字符为存储单位。字符串存储按字符串顺序存放在内存中（字符从左到右，地址由低到高），当然，保存在内存中是它们的ASCII码。

存储结构分析：关于字符串"hello world\n"在内存中的存储情况，[0x10010000]表示内存地址为0x10010000的内存单元内容。如，[0x10010000]=0x68（‘h’），[0x10010001]=0x65（‘e’）；十六进制ASCII码：0x20 (表示空格)， 0x0a (表示换行符\n)。

分析：

十六进制 ASCII 码：	<u>0x6c 6c 65 68</u>	<u>0x6f 77 20 6f</u>	<u>0x0a 64 6c 72</u>
对应字符：	l l e h	o w o	\n d l r
十进制 ASCII 码：	<u>108 108 101 104</u>	<u>111 119 32 111</u>	<u>10 100 108 114</u>
对应字符：	l l e h	o w 空格 o	\n d l r

提示：显示字符简易操作：按[ALT]键 + 小键盘输入‘104’ => 显示‘h’

2、数据段：数据的存储

.data

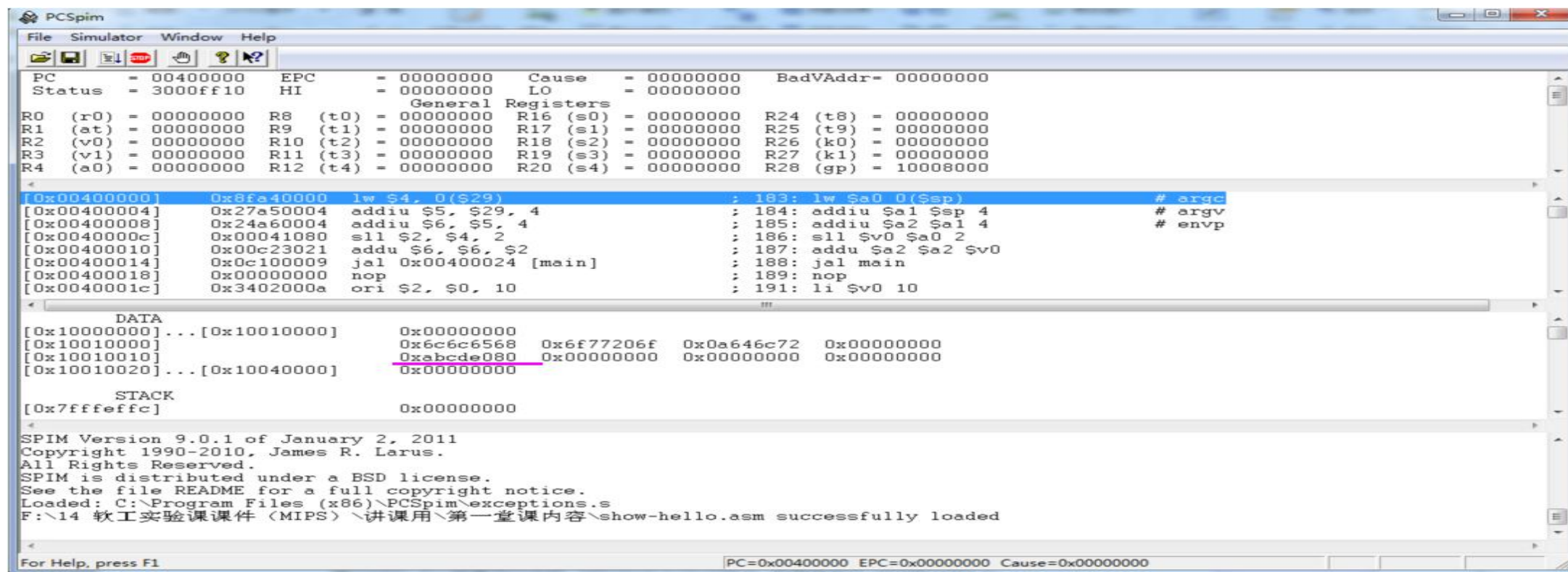
数据段

str: # 变量名称

.ascii "hello world\n" # 字符串定义，以“00”为终止符结束。

memdata: # 变量名称，说明：这个数据定义在本程序中无意义，只是借用说明一下数据存储结构！

.word 0xabcde080 # 数据定义，字长，32位



The screenshot shows the PCSpim MIPS simulator interface. The top panel displays assembly code with addresses, hex values, and instructions. The bottom panel shows the memory layout, including the DATA segment and the STACK segment. The DATA segment contains a word value 0xabcde080 at address 0x10010010. The STACK segment is located at address 0x7ffefffc.

```
PCSpim
File Simulator Window Help
PC = 00400000 EPC = 00000000 Cause = 00000000 BadVAddr = 00000000
Status = 3000ff10 HI = 00000000 LO = 00000000
General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000

[0x00400000] 0x8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 188: jal main
[0x00400018] 0x00000000 nop ; 189: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 191: li $v0 10

DATA
[0x10000000] ... [0x10010000] 0x00000000
[0x10010000] 0x6c6c6568 0x6f77206f 0x0a646c72 0x00000000
[0x10010010] 0xabcde080 0x00000000 0x00000000 0x00000000
[0x10010020] ... [0x10040000] 0x00000000

STACK
[0x7ffefffc] 0x00000000

SPIM Version 9.0.1 of January 2, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
Loaded: C:\Program Files (x86)\PCSpim\exceptions.s
F:\14 软工实验课课件 (MIPS)\讲课用\第一堂课内容\show-hello.asm successfully loaded

For Help, press F1 [PC=0x00400000 EPC=0x00000000 Cause=0x00000000]
```

数据：模拟器中是以8位长度的十六进制数作为一个显示表示单位，但存储器中存储是以字节为单位，低位数据存储在低位地址单元中，数据由低位到高位顺序存储，采用**小端存储模式**。“0xabcde080”在以上图中只是显示的表示形式，并非存储结构。

而“0xabcde080”在存储器中是这样存储的：（0x10010010等为内存地址，本例上图）

[0x10010010]=10000000，即0x80；[0x10010011]=11100000，即0xe0；

[0x10010012]=11001101，即0xcd；[0x10010013]=10101011，即0xab。

可以通过以下程序来认识存储器中数据存储情况：

```
.text          # 代码段
```

```
.globl main    # 程序从此开始
```

```
main:         # 主程序
```

```
    lw $t0,memdata  # 从存储器中读取一个字的数据到寄存器中，整32位， WORD
```

```
    lh $t1,memdata  # 从存储器中读取半个字的数据到寄存器中，半字符符号扩展， HALFWORD
```

```
    lb $t2,memdata  # 从存储器中读取一个字节的数据到寄存器中，字节符号扩展， BYTE
```

```
    lhu $t3,memdata # 从存储器中读取半个字的数据到寄存器中，无符号半字扩展， HALFWORD
```

```
    lbu $t4,memdata # 从存储器中读取一个字节的数据到寄存器中，无符号字节扩展， BYTE
```

```
    lb $s4,memdata+1 #（取memdata第二个单元数据）从存储器中读取一  
                    # 个字节的数据到寄存器中，字节符号扩展 BYTE
```

```
    li $v0, 10      # 退出
```

```
    syscall         # 系统调用
```


.data

数据段

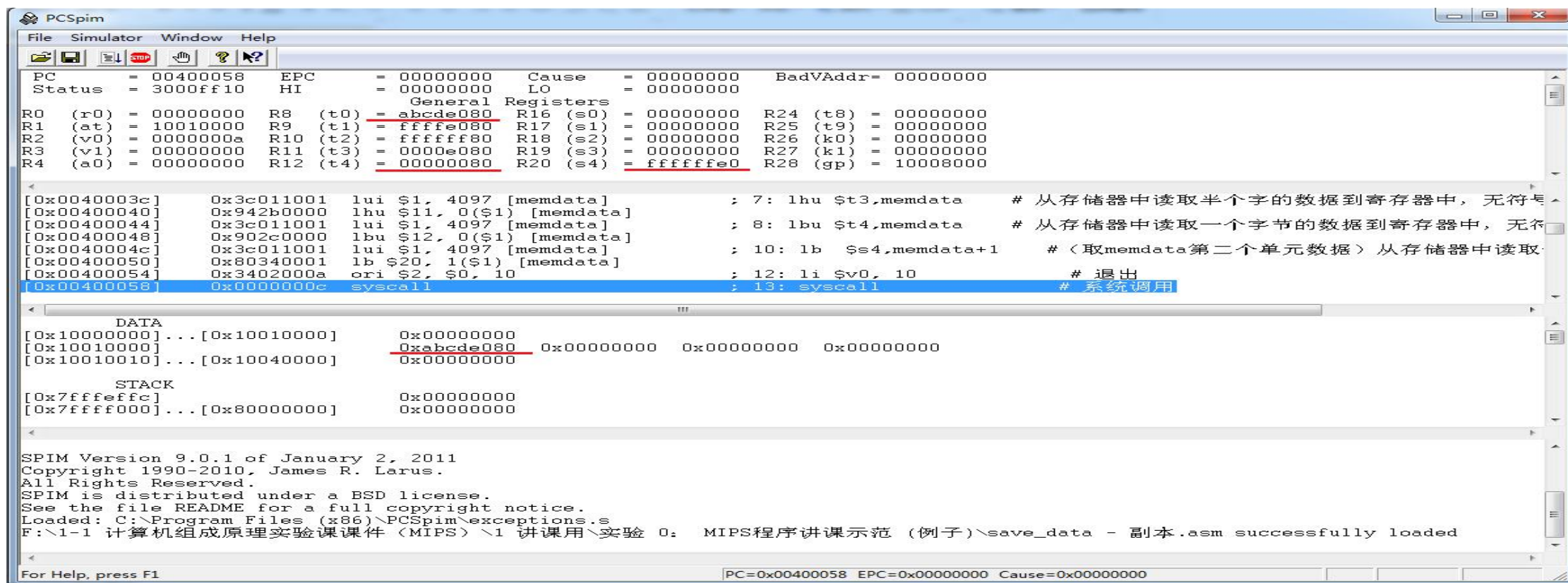
memdata: # 变量名称

.word 0xabcd080 # 数据定义-字（32位）

以上程序执行结果，相应寄存器中的内容为：

(\$t0) = abdce080 , (\$t1) = ffffe080 , (\$t2) = fffff80 , (\$t3) = 0000e080,

(\$t4) = 00000080 , (\$s4) = fffffe0, 看下图：



The screenshot shows the PCSpim MIPS simulator interface. The top window displays the state of the processor, including the Program Counter (PC) at 00400058, the EPC at 00000000, and the Cause register at 00000000. The Status register is 3000ff10, and the HI register is 00000000. The General Registers are listed, with R0 (r0) at 00000000, R1 (at) at 10010000, R2 (v0) at 0000000a, R3 (v1) at 00000000, R4 (a0) at 00000000, R8 (t0) at abcd080, R9 (t1) at fffe080, R10 (t2) at fffff80, R11 (t3) at 0000e080, R12 (t4) at 00000080, R16 (s0) at 00000000, R17 (s1) at 00000000, R18 (s2) at 00000000, R19 (s3) at 00000000, R20 (s4) at fffffe0, R24 (t8) at 00000000, R25 (t9) at 00000000, R26 (k0) at 00000000, R27 (k1) at 00000000, and R28 (gp) at 10008000.

The assembly code window shows the following instructions:

```
[0x0040003c] 0x3c011001 lui $1, 4097 [memdata] ; 7: lhu $t3,memdata # 从存储器中读取半个字的数据到寄存器中，无符号
[0x00400040] 0x942b0000 lhu $11, 0($1) [memdata] ; 8: lbu $t4,memdata # 从存储器中读取一个字的数据到寄存器中，无符号
[0x00400044] 0x3c011001 lui $1, 4097 [memdata] ; 10: lb $s4,memdata+1 # <取memdata第二个单元数据>从存储器中读取
[0x00400048] 0x902c0000 lbu $12, 0($1) [memdata] ; 12: li $v0, 10 # 退出
[0x00400050] 0x80340001 lb $20, 1($1) [memdata] ; 13: syscall # 系统调用
[0x00400054] 0x3402000a ori $2, $0, 10
[0x00400058] 0x0000000c syscall
```

The DATA section shows the memory layout:

```
[0x10000000]...[0x10010000] 0x00000000
[0x10010000]...[0x10020000] 0xabcd080 0x00000000 0x00000000 0x00000000
[0x10020000]...[0x10030000] 0x00000000
```

The STACK section shows the stack layout:

```
[0x7ffffeffc] 0x00000000
[0x7fffff000]...[0x800000000] 0x00000000
```

The bottom window shows the SPIM Version 9.0.1 of January 2, 2011, Copyright 1990-2010, James R. Larus. All Rights Reserved. SPIM is distributed under a BSD license. See the file README for a full copyright notice. Loaded: C:\Program Files (x86)\PCSpim\exceptions.s F:\1-1 计算机组成原理实验课课件 (MIPS) \1 讲课用\实验 0: MIPS程序讲课示范 (例子)\save_data - 副本.asm successfully loaded

For Help, press F1

PC=0x00400058 EPC=0x00000000 Cause=0x00000000

四、syscall系统调用功能表

Service	Trap code	Input	Output	Notes
print_int	$\$v0 = 1$	$\$a0$ = integer to print	prints $\$a0$ to standard output	
print_float	$\$v0 = 2$	$\$f12$ = float to print	prints $\$f12$ to standard output	
print_double	$\$v0 = 3$	$\$f12$ = double to print	prints $\$f12$ to standard output	
print_string	$\$v0 = 4$	$\$a0$ = address of first character		prints a character string to standard output
read_int	$\$v0 = 5$		integer read from standard input placed in $\$v0$	
read_float	$\$v0 = 6$		float read from standard input placed in $\$f0$	
read_double	$\$v0 = 7$		double read from standard input placed in $\$f0$	

read_string	\$v0 = 8	\$a0 = address to place string, \$a1 = max string length	reads standard input into address in \$a0	
<u>sbrk</u>	\$v0 = 9	\$a0 = number of bytes required	\$v0 = address of allocated memory	Allocates memory from the heap
	<p>heap: 是由 <u>malloc</u> 之类函数分配的空间所在地。地址是由低向高增长的。</p> <p>stack:: 是自动分配变量, 以及函数调用时所用的一些空间, 地址是由高向低减少的。</p>			
exit	\$v0 = 10			退出
print_char	\$v0 = 11	\$a0 = character (low 8 bits)		
read_char	\$v0 = 12		\$v0 = character (no line feed) echoed	
file_open	\$v0 = 13	\$a0 = full path (zero terminated string with no line feed), \$a1 = flags, \$a2 =	\$v0 = file descriptor	

		UNIX octal file mode (0644 for <u>rw-r--r--</u>)		
file_read	\$v0 = 14	\$a0 = file descriptor, \$a1 = buffer address, \$a2 = amount to read in bytes	\$v0 = amount of data in buffer from file (-1 = error, 0 = end of file)	
file_write	\$v0 = 15	\$a0 = file descriptor, \$a1 = buffer address, \$a2 = amount to write in bytes	\$v0 = amount of data in buffer to file (-1 = error, 0 = end of file)	
file_close	\$v0 = 16	\$a0 = file descriptor		

五、ASCII码对照表

ASCII表																										
(American Standard Code for Information Interchange 美国标准信息交换代码)																										
高四位 低四位		ASCII控制字符											ASCII打印字符													
		0000					0001						0010		0011		0100		0101		0110		0111			
		0					1						2		3		4		5		6		7			
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☹	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX		正文结束	19	!!	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK		肯定应答	22	—	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	\a	响铃	23	↕	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	▣	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◐	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[ESC	\e	溢出	43	+	59	;	75	K	91	[107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	└	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124		
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分分隔符	45	-	61	=	77	M	93]	109	m	125	}	
1110	E	14	🎵	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	🎵	^O	SI		移入	31	▼	^_	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	△	^Backspace 代码: DEL