

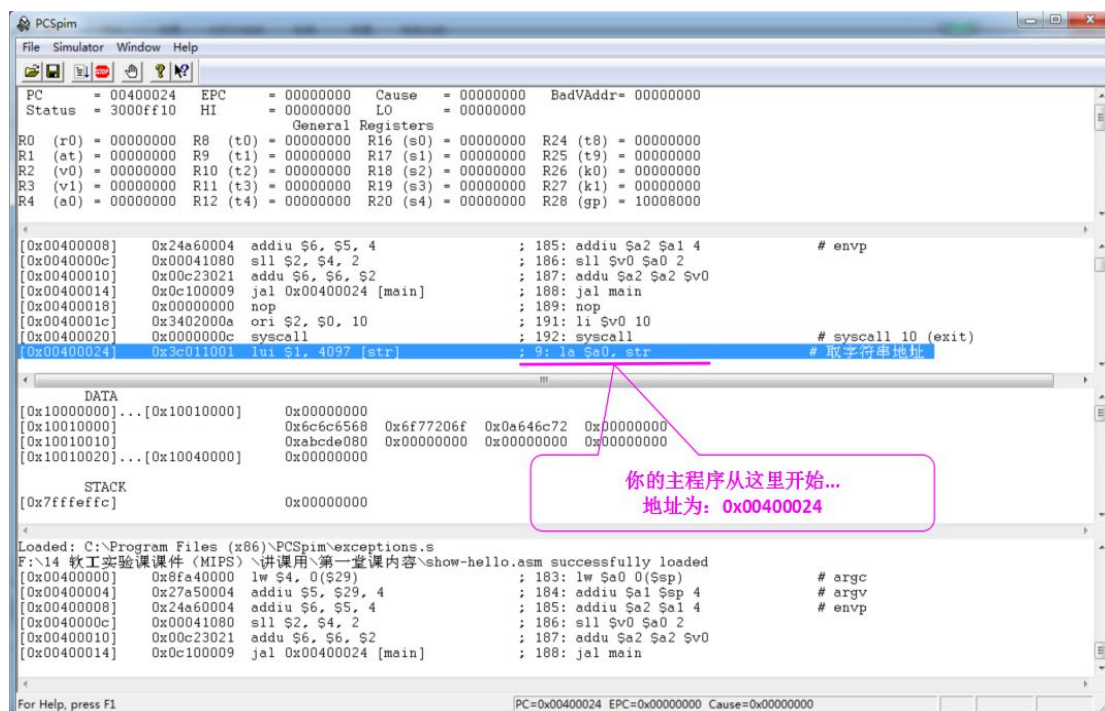
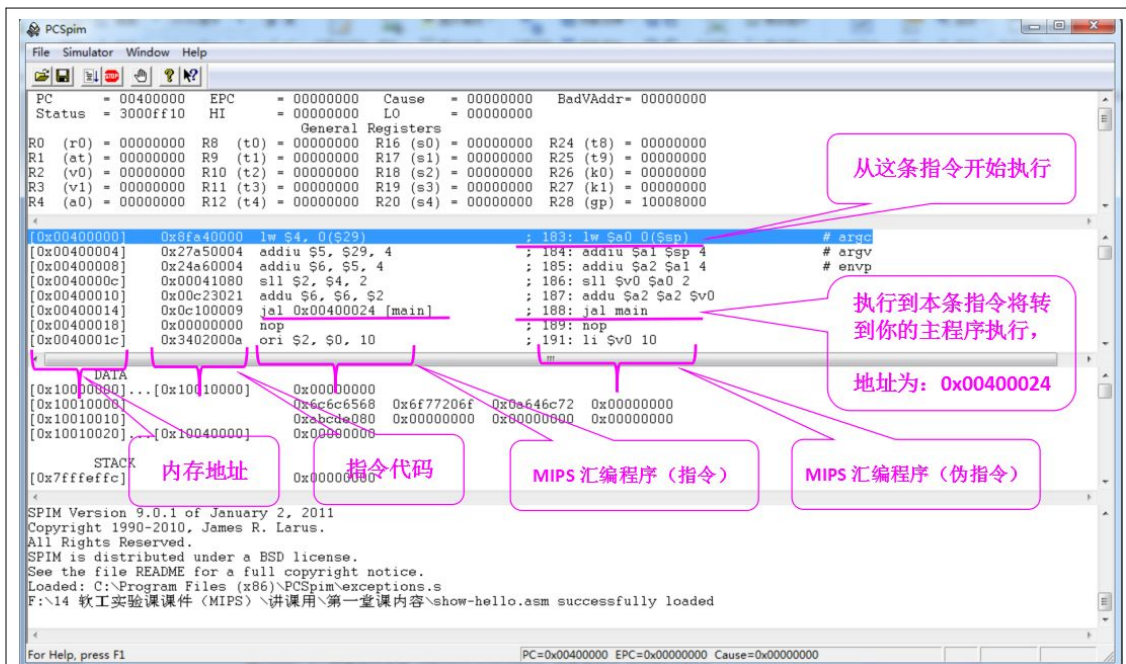
PCSpim 模拟器界面介绍

The screenshot shows the PCSpim simulator interface with the following sections:

- 寄存器部分 (Registers):** Displays the current state of the processor registers. PC is 00400000, Status is 3000fff10, EPC is 00000000, Cause is 00000000, and BadVAddr is 00000000. General registers R0 through R31 are listed with their values.
- 代码部分 (Code):** Shows the assembly code being executed. The current instruction is `lw $s4, 0($s29)` at address 0x00400000. Other instructions include `addiu $s5, $s29, 4`, `addiu $s6, $s5, 4`, `sll $s2, $s4, 2`, `addu $s6, $s6, $s2`, `jal 0x00400024 [main]`, `nop`, and `ori $s2, $s0, 10`.
- 存储器部分 (Memory):** Displays the memory layout. DATA segment starts at 0x10000000. STACK segment starts at 0x7fffffc.
- 信息部分 (Information):** Shows the SPIM version (9.0.1 of January 2, 2011) and copyright information (Copyright 1990-2010, James R. Larus). It also displays the loaded file path: `F:\14 软工实验课课件 (MIPS) \讲读用\第一堂课程内容\show-hello.asm`.

The screenshot shows the PCSpim simulator interface with a note about the main function name:

注意:
MIPS主程序的名字, 如 `.globl <主程序名字>` 必须与PCSpim中的指令 `jal main` 的“main”同名, 即main, 否则, 必须修改exceptions.s文件中的“main”与你的MIPS主程序名字相同。exceptions.s文件在安装目录下有。



（一）以下程序输出字符串“hello world”，以及分析字符串与数据的存储结构。以上的介绍也是用这个例子来说明的。

```
.text
.globl main
main:
    la $a0, str
    li $v0, 4
    syscall

    li $v0, 10
```

代码段 声明

globl 指明程序的入口地址 main

入口地址 main

取字符串地址

4 号功能调用，输出字符串

系统调用，输出字符串

退出

syscall # 系统调用

.data # 数据段 声明

str: # 变量名称

.asciiz "hello world\n" # 字符串定义

memory: # 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!

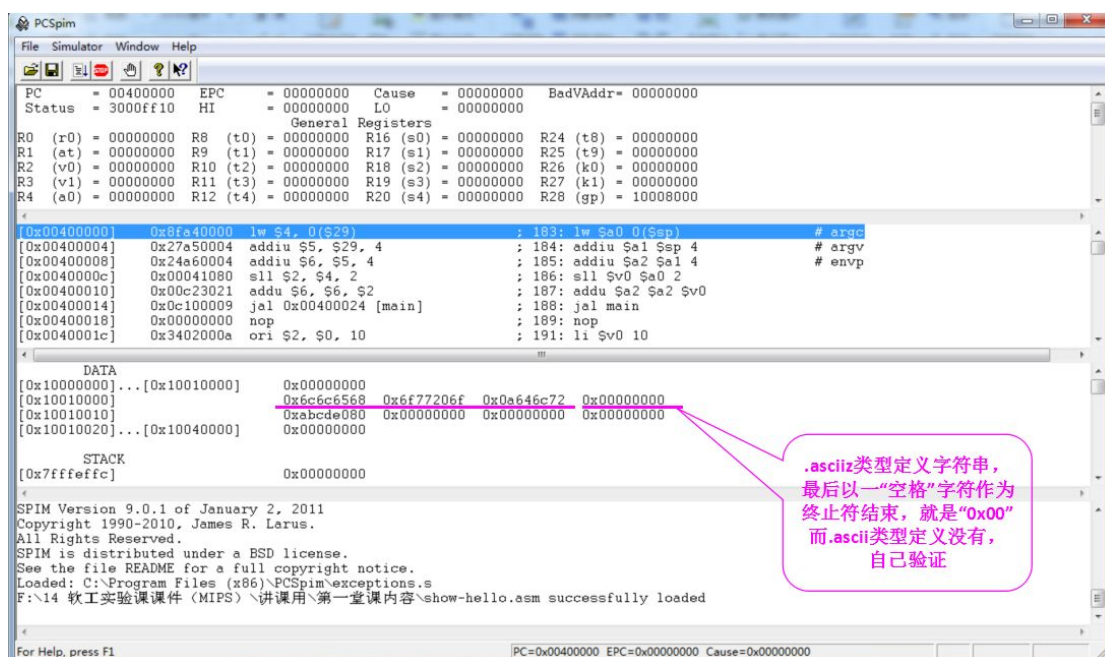
.word 0xabcd080 # 数据定义, 32 位长度

(1) 以下分析数据段部分 (变量定义与存储情况, 字符串):

.data # 数据段

str: # 变量名称

.asciiz "hello world\n" # 字符串定义



字符串: 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 码值。

存储结构分析: 关于 "hello world\n", 如, [0x10010000]=0x68 ('h'), [0x10010001]=0x65 ('e'), 十六进制 ASCII 码值: 20 (sp 空格), 0a (LF, 换行符\n)

分解:

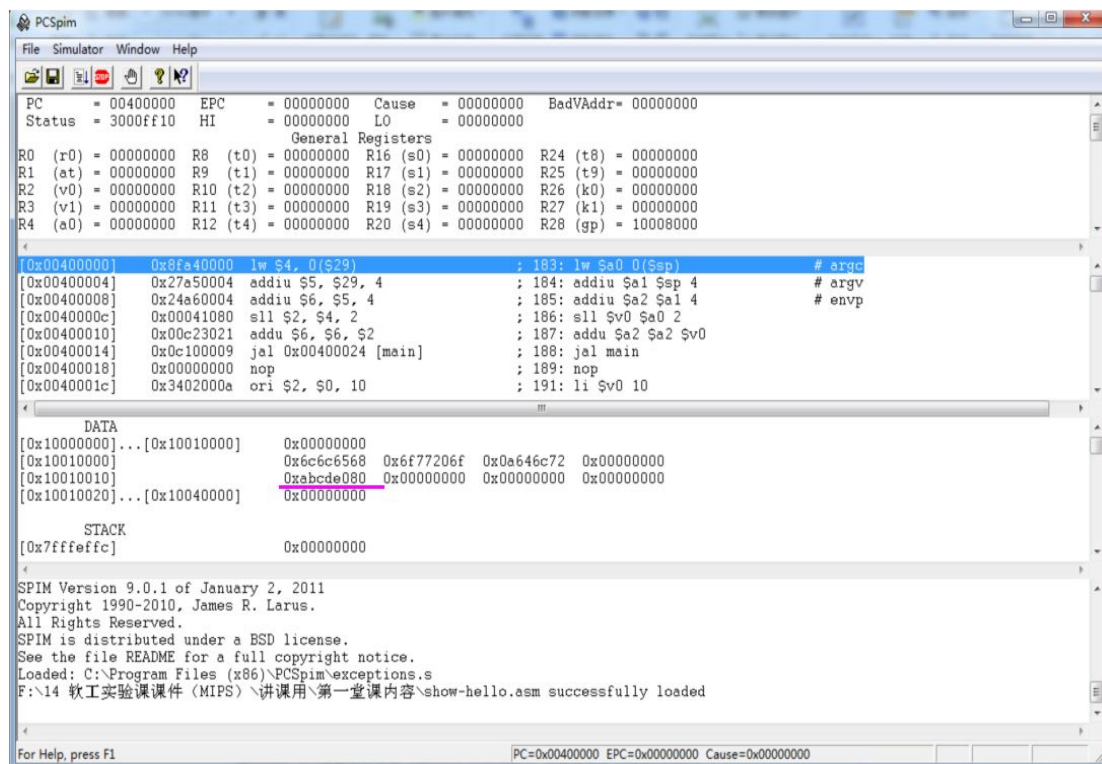
十六进制 ASCII 码值: 0x6c 6c 65 68 0x6f 77 20 6f 0x0a 64 6c 72
对应字符: l l e h o w o \n d l r
十进制 ASCII 码值: 108 108 101 104 111 119 32 111 10 100 108 114
对应字符: l l e h o w 空格 o \n d l r

显示字符简易操作: 按【ALT】键 + 小键盘输入 '104' => 显示 'h'

(2) 以下分析数据段部分 (变量定义与存储情况, 数据):

.data # 数据段声明


```
memory:          # 变量名称
.word 0xabcde080 # 数据定义, 32 位长度
```



数据：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，但存储是以字节为单位，低位数存储在低位地址单元中，数据由低位到高位顺序存储。“0xabcde080”在以上图中只是显示的形式，并非存储结构。

而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：

```
.text          # 代码段
.globl main    # 程序从此开始
main:          # 主程序
    lw $t0,memory    # 从存储器中读取一个字的数据到寄存器中，整 32 位 WORD
    lh $t1,memory    # 从存储器中读取半个字的数据到寄存器中，半符号扩展 HALFWORD
    lb $t2,memory    # 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,memory   # 从存储器中读取半个字的数据到寄存器中，无符号扩展 HALFWORD
    lbu $t4,memory   # 从存储器中读取一个字节的数据到寄存器中，无符号扩展 BYTE

    lb $s4,memory+1  # (取 memory 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
```

memory:

.word 0xabcde080

变量名称

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：

```
PCSpim
File Simulator Window Help
Status = 3000ff10 HI = 00000000 LO = 00000000
General Registers
R0 (r0) = 00000000 R8 (t0) = abdce080 R15 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 10010000 R9 (t1) = ffffe080 R16 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = fffff80 R17 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 0000e080 R18 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000080 R19 (s4) = fffffe0 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

DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000]...[0x10020000] 0xabcde080 0x00000000 0x00000000 0x00000000
[0x10020000]...[0x10030000] 0x00000000

STACK
[0x7fffffc] 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

For Help, press F1 PC=0x00400054 EPC=0x00000000 Cause=0x00000000
```

(二) 以下是系统调用相关内容：

System Call

Service	System Call Code	Arguments	Result
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	\$a0 = buffer, \$a1 = length	
sbrk	9	\$a0 = amount	address (in \$v0)
exit	10		

SYSCALL 系统功能调用表详细说明

Examples of system calls (used by SPIM)

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	
sbrk	\$v0 = 9	\$a0 = number of bytes required	\$v0 = address of allocated memory	Allocates memory from the heap
	<p>heap: 是由 malloc 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			
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 = UNIX octal file mode	\$v0 = file descriptor	

		(0644 for rw-r--r--)		
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		

ASCII 字符代码表

ASCII 字符代码表 一

高四位	ASCII非打印控制字符										ASCII 打印字符															
	0000					0001					0010		0011		0100		0101		0110		0111					
	0					1					2		3		4		5		6		7					
	+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl			
0000	0	0	BLANK NULL	^@ NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p				
0001	1	1	☺	^A SOH	头标开始	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 BOT	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	反确认	37	%	53	5	69	E	85	U	101	e	117	u				
0110	6	6	♠	^F ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v				
0111	7	7	●	^G BEL	震铃	23	↑	^W	ETB	传输块结束	39	'	55	7	71	G	87	w	103	g	119	w				
1000	8	8	◼	^H BS	退格	24	↑	^X	CAN	取消	40	(56	8	72	H	88	X	104	h	120	x				
1001	9	9	○	^I TAB	水平制表符	25	↓	^Y	EM	媒体结束	41)	57	9	73	I	89	Y	105	i	121	y				
1010	A	10	◻	^J LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z				
1011	B	11	♂	^K VT	垂直制表符	27	←	^[ESC	转意	43	+	59	;	75	K	91	[107	k	123	{				
1100	C	12	♀	^L FF	换页/新页	28	└	^\ FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124						
1101	D	13	♪	^M CR	回车	29	↔	^] GS	组分隔符	45	-	61	=	77	M	93]	109	m	125	}					
1110	E	14	🎵	^N SO	移出	30	▲	^6 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	Δ					

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键”输入