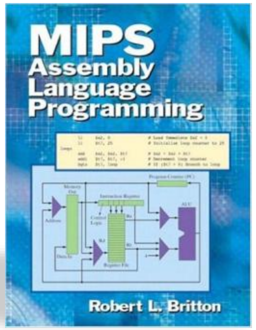
计算机组成和体系结构**实验**

- 体系结构（MIPS汇编语言）
- 微体系结构（单周期处理器）



PDF
Adobe

xgsun@fudan.edu.cn

孙晓光

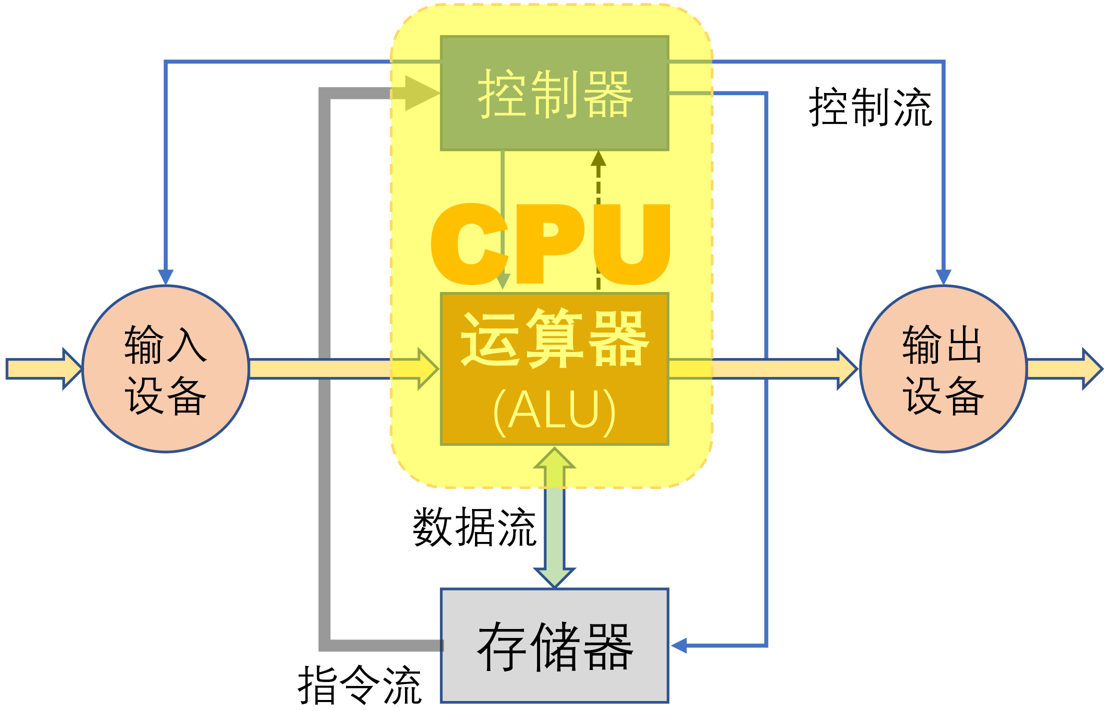
2024-12-1

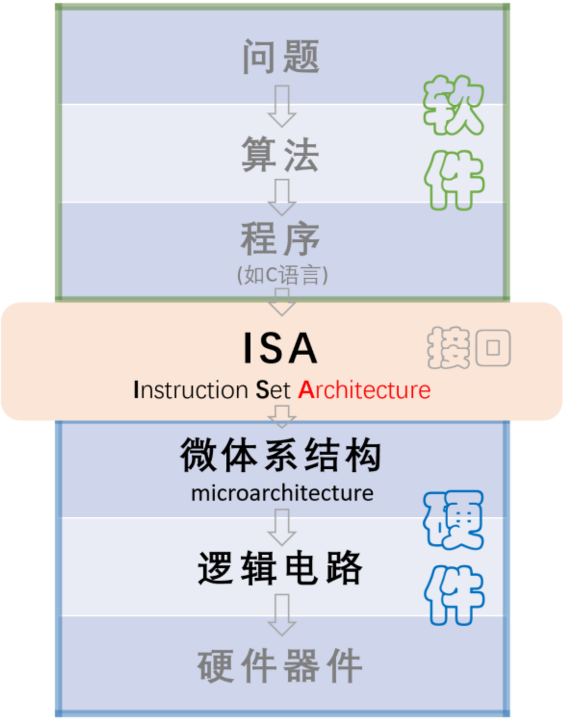
概 述

32位 MIPS指令集 + **CPU结构**

MIPS汇编语言 （QtSpim）	Verilog / SystemVerilog （**Vivado**）

NEXYS4 DDR开发板



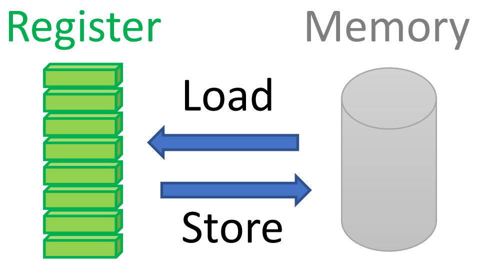


复杂指令集计算机 CISC vs RISC 精简指令集计算机

Complex Instruction Set Computer　　Reduced Instruction Set Computer

- 采用**复杂的指令系统**
- 指令数量多，功能复杂
- 指令长度可变，指令格式多样
- 寻址方式多

- 采用**简化的指令系统**
- 指令集只包含常用的指令
- 提供大量<u>通用寄存器</u>，少访问内存
- 只有**Load**和**Store**指令才能访问内存



VAX

ADDL (R9), (R10), (R11)

; mem[R9] ⟵ mem[R10] + mem[R11]

=

MIPS

lw R1, (R10)	# R1 ⟵ mem[R10]
lw R2, (R11)	# R2 ⟵ mem[R11]
add R3, R1, R2	# R3 ⟵ R1+R2
sw R3, (R9)	# mem[R9] ⟵ R3

MIPS 体系结构设计的4个**准则**

John Hennessy

① **简单设计**有助于规整化。如，指令长度相同，格式固定

② 加快**常见功能**。 如，指令集中只包含最常用的指令

③ **越小的设计**越快。 如，较少的硬件将有更少的延迟

④ 好的设计需要**好的折中方法**。如，指令有3种格式，采用扩展码技术

A simpler CPU is a faster CPU.

“要使某些事情变得非常复杂是非常简单的；但要使它变得简单将非常复杂。”

苏联 著名枪械(波波沙冲锋枪)设计师 **沙普金**

Opcode	Name	Description	Operation
000000 (0)	R-type	all R-type instructions	see Table B.2
000001 (1) (rt = 0/1)	bltz rs, label / bgez rs, label	branch less than zero/branch greater than or equal to zero	if ([rs] < 0) PC = BTA/ if ([rs] ≥ 0) PC = BTA
000010 (2)	j label	jump	PC = JTA
000011 (3)	jal label	jump and link	$ra = PC + 4, PC = JTA
000100 (4)	beq rs, rt, label	branch if equal	if ([rs] == [rt]) PC = BTA
000101 (5)	bne rs, rt, label	branch if not equal	if ([rs] != [rt]) PC = BTA
000110 (6)	blez rs, label	branch if less than or equal to zero	if ([rs] ≤ 0) PC = BTA
000111 (7)	bgtz rs, label	branch if greater than zero	if ([rs] > 0) PC = BTA
001000 (8)	addi rt, rs, imm	add immediate	[rt] = [rs] + SignImm
001001 (9)	addiu rt, rs, imm	add immediate unsigned	[rt] = [rs] + SignImm
001010 (10)	slti rt, rs, imm	set less than immediate	[rs] < SignImm ? [rt] = 1 : [rt] = 0
001011 (11)	sltiu rt, rs, imm	set less than immediate unsigned	[rs] < SignImm ? [rt] = 1 : [rt] = 0
001100 (12)	andi rt, rs, imm	and immediate	[rt] = [rs] & ZeroImm
001101 (13)	ori rt, rs, imm	or immediate	[rt] = [rs] | ZeroImm
001110 (14)	xori rt, rs, imm	xor immediate	[rt] = [rs] ^ ZeroImm
001111 (15)	lui rt, imm	load upper immediate	[rt] = {imm, 16'b0}
010000 (16) (rs = 0/4)	mfc0 rt, rd / mtc0 rt, rd	move from/to coprocessor 0	[rt] = [rd]/[rd] = [rt] (rd is in coprocessor 0)
010001 (17)	F-type	fop = 16/17: F-type instructions	see Table B.3
010001 (17) (rt = 0/1)	bc1f label/ bc1t label	fop = 8: branch if fpcond is FALSE/TRUE	if (fpcond == 0) PC = BTA/ if (fpcond == 1) PC = BTA
011100 (28) (func = 2)	mul rd, rs, rt	multiply (32-bit result)	[rd] = [rs] x [rt]
100000 (32)	lb rt, imm(rs)	load byte	[rt] = SignExt ([Address]$_{7:0}$)
100001 (33)	lh rt, imm(rs)	load halfword	[rt] = SignExt ([Address]$_{15:0}$)
100011 (35)	lw rt, imm(rs)	load word	[rt] = [Address]
100100 (36)	lbu rt, imm(rs)	load byte unsigned	[rt] = ZeroExt ([Address]$_{7:0}$)
100101 (37)	lhu rt, imm(rs)	load halfword unsigned	[rt] = ZeroExt ([Address]$_{15:0}$)
101000 (40)	sb rt, imm(rs)	store byte	[Address]$_{7:0}$ = [rt]$_{7:0}$
101001 (41)	sh rt, imm(rs)	store halfword	[Address]$_{15:0}$ = [rt]$_{15:0}$
101011 (43)	sw rt, imm(rs)	store word	[Address] = [rt]
110001 (49)	lwc1 ft, imm(rs)	load word to FP coprocessor 1	[ft] = [Address]
111001 (56)	swc1 ft, imm(rs)	store word to FP coprocessor 1	[Address] = [ft]

R-type

Funct	Name	Description	Operation
000000 (0)	sll rd, rt, shamt	shift left logical	[rd] = [rt] << shamt
000010 (2)	srl rd, rt, shamt	shift right logical	[rd] = [rt] >> shamt
000011 (3)	sra rd, rt, shamt	shift right arithmetic	[rd] = [rt] >>> shamt
000100 (4)	sllv rd, rt, rs	shift left logical variable	[rd] = [rt] << [rs]$_{4:0}$
000110 (6)	srlv rd, rt, rs	shift right logical variable	[rd] = [rt] >> [rs]$_{4:0}$
000111 (7)	srav rd, rt, rs	shift right arithmetic variable	[rd] = [rt] >>> [rs]$_{4:0}$
001000 (8)	jr rs	jump register	PC = [rs]
001001 (9)	jalr rs	jump and link register	$ra = PC + 4, PC = [rs]
001100 (12)	syscall	system call	system call exception
001101 (13)	break	break	break exception
[illegible]	[illegible]	move from hi	[rd] = [hi]
[illegible]	[illegible]	move to hi	[hi] = [rs]
[illegible]	[illegible]	move from lo	[rd] = [lo]
[illegible]	[illegible]	move to lo	[lo] = [rs]
[illegible]	t	multiply	{[hi], [lo]} = [rs] × [rt]
[illegible]	rt	multiply unsigned	{[hi], [lo]} = [rs] × [rt]
[illegible]	[illegible]	divide	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
[illegible]	t	divide unsigned	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
100000 (32)	add rd, rs, rt	add	[rd] = [rs] + [rt]
100001 (33)	addu rd, rs, rt	add unsigned	[rd] = [rs] + [rt]
100010 (34)	sub rd, rs, rt	subtract	[rd] = [rs] - [rt]
100011 (35)	subu rd, rs, rt	subtract unsigned	[rd] = [rs] - [rt]
100100 (36)	and rd, rs, rt	and	[rd] = [rs] & [rt]
100101 (37)	or rd, rs, rt	or	[rd] = [rs] | [rt]
100110 (38)	xor rd, rs, rt	xor	[rd] = [rs] ^ [rt]
100111 (39)	nor rd, rs, rt	nor	[rd] = ~([rs] | [rt])
101010 (42)	slt rd, rs, rt	set less than	[rs] < [rt] ? [rd] = 1 : [rd] = 0
101011 (43)	sltu rd, rs, rt	set less than unsigned	[rs] < [rt] ? [rd] = 1 : [rd] = 0

F-type

Funct	Name	Description	Operation
000000 (0)	add.s fd, fs, ft / add.d fd, fs, ft	FP add	[fd] = [fs] + [ft]
000001 (1)	sub.s fd, fs, ft / sub.d fd, fs, ft	FP subtract	[fd] = [fs] - [ft]
000010 (2)	mul.s fd, fs, ft / mul.d fd, fs, ft	FP multiply	[fd] = [fs] × [ft]
000011 (3)	div.s fd, fs, ft / div.d fd, fs, ft	FP divide	[fd] = [fs]/[ft]
000101 (5)	abs.s fd, fs / abs.d fd, fs	FP absolute value	[fd] = ([fs] < 0) ? [-fs] : [fs]
000111 (7)	neg.s fd, fs / neg.d fd, fs	FP negation	[fd] = [-fs]
111010 (58)	c.seq.s fs, ft / c.seq.d fs, ft	FP equality comparison	fpcond = ([fs] == [ft])
111100 (60)	c.lt.s fs, ft / c.lt.d fs, ft	FP less than comparison	fpcond = ([fs] < [ft])
111110 (62)	c.le.s fs, ft / c.le.d fs, ft	FP less than or equal comparison	fpcond = ([fs] ≤ [ft])

MIPS 指令集

理论上，Load/Store/Inc/Branch四种指令，足够编制任何可计算程序，但程序会很长。

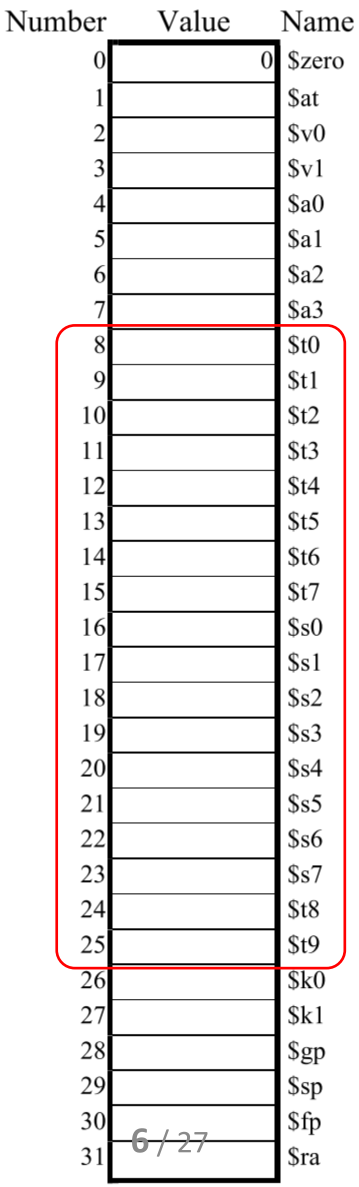
操作数：寄存器、存储器、常数

指令 instruction	=	操作 operation	+	操作数 operand		
		add		t ,	b ,	c
		add		`$t0,`	`$s1,`	`$s2`
		sub		a ,	t ,	d
		sub		`$s0,`	`$t0,`	`$s3`

a = b + c - d;

t = b + c;

a = t - d;



名称	编号	用途
$0	0	**常数0**
$at	1	汇编器临时变量
$v0 ~ $v1	2~3	函数返回值
$a0 ~ $a3	4~7	函数参数
$t0 ~ $t7	8~15	**临时变量**
$s0 ~ $s7	16~23	**保存变量**

名称	编号	用途
$t8 ~ $t9	24~25	**临时变量**
$k0 ~$k1	26~27	操作系统临时变量
$gp	28	全局指针
$sp	29	栈指针
$fp	30	帧指针
$ra	31	函数返回地址

操作数：寄存器、存储器、常数

32位地址，4GB空间；32位数据字长。字节(8位)寻址，每1个字节都有1个单独地址。

字 寻址

```
lw  $s3, 1($0)
sw  $s7, 5($0)
```

Word Address	Data	
⋮	⋮	⋮
00000003	4 0 F 3 0 7 8 8	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

字节 寻址

```
lw  $s3, 4($0)
sw  $s7, 0x10($0)
```

Word Address	Data				
⋮	⋮				⋮
0000000C	4 0	F 3	0 7	8 8	Word 3
00000008	0 1	E E	2 8	4 2	Word 2
00000004	F 2	F 1	A C	0 7	Word 1
00000000	A B	C D	E F	7 8	Word 0

每个字地址都是4的倍数

width = 4 bytes

操作数：寄存器、存储器、常数

因常数的值可以立即访问，故又称为**立即数**(immediate)。

```
# $s0=a, $s1=b
addi $s0, $s1, 4    # a=b+4
addi $s1, $s0,-2    # b=a-2
```

- 立即数采用**16位补码**表示，[-32768, 32767]
- 减法相当于加上一个负数，故没有 subi 指令

设 $s0 最初包含 0x23456789。运行下面代码后 $s0 = ?

```
sw $s0, 0($0)
lb $s0, 1($0) # 将字节地址(1+$0)=1中的数据装入$s0的最低有效字节中
```

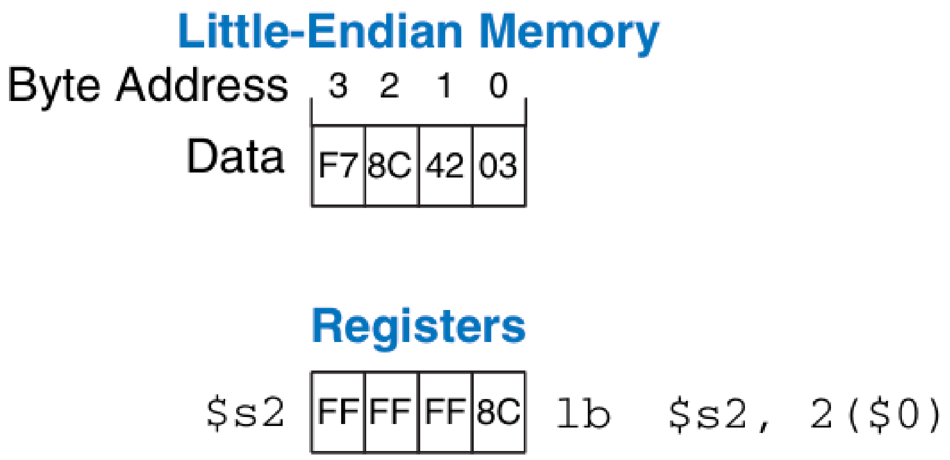
解:

大端

字节地址	0	1	2	3	**字地址**
数据值	23	45	67	89	**0**
	MSB			LSB	

小端

字节地址	3	2	1	0	**字地址**
数据值	23	45	67	89	**0**
	MSB 最高有效位			LSB 最低有效位	



MIPS指令集3种指令格式

32 bits

① **Register 型**

3寄存器

```
add $s0, $s1, $s2
sll $s0, $s1,  2
 jr $s0
```

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

② **Immediate型**

2寄存器+**16位立即数**

```
addi $s0, $s1, -4
lw   $s0, 8($0)
```

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

③ **Jump 型**

26位立即数

```
j   label
jal label
```

op	addr
6 bits	26 bits

① Register 型

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
操作码=0	2个源寄存器		1个目的寄存器	移位操作	函数：决定何种R操作

汇编代码: **add** $s0, $s1, $s2

rd rs rt

字段值:

0	17	18	16	0	32
op	rs	rt	rd	shamt	funct

机器代码:

000000	10001	10010	10000	00000	100000
6位	5位	5位	5位	5位	6位

机器指令: 0x02328020

②Immediate 型

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
操作码	寄存器		立即数

汇编代码: `addi $s0, $s1, -4`

rt: $s0　rs: $s1

字段值:

8	17	16	-4
op	rs	rt	imm

机器代码:

0010 00	10001	10010	1111 1111 1111 1100
6位	5位	5位	16位

机器指令: 0x2232FFFC

对正立即数，高16位都补0
对负立即数，高16位都补1

③ Jump 型

op	addr
6 bits	26 bits
操作码	26位地址操作数

Assembly Code	Field Values		Machine Code		
	op	addr	op	addr	
jal sum	3	0x0100028	000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
	6 bits	26 bits	6 bits	26 bits	

32-bit sum地址 JTA 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

转换为 26-bit addr 0000 $\underbrace{00}_{0}\underbrace{00\ 01}_{1}\underbrace{00\ 00}_{0}\underbrace{00\ 00}_{0}\underbrace{00\ 00}_{0}\underbrace{00\ 10}_{2}\underbrace{10\ 00}_{8}$00 (0x0100028)

去掉前4位　　**保留中间26位**　　去掉后2位(/4)

$$\boxed{\mathrm{PC}' = \{(\mathrm{PC}+4)[31:28],\ \boldsymbol{addr},\ 2'b0\}}$$

① 逻辑指令

(R型) **and rd、rs、rt**

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

没有NOT，可用下面代替

A NOR $0 = NOT A

Assembly Code	Result								
`and $s3, $s1, $s2`	$s3	0100	0110	1010	0001	0000	0000	0000	0000
`or  $s4, $s1, $s2`	$s4	1111	1111	1111	1111	1111	0000	1011	0111
`xor $s5, $s1, $s2`	$s5	1011	1001	0101	1110	1111	0000	1011	0111
`nor $s6, $s1, $s2`	$s6	0000	0000	0000	0000	0000	1111	0100	1000

(I型) **andi rt、rs、imm**

Source Values

$s1	0000	0000	0000	0000	0000	0000	1111	1111
imm	0000	0000	0000	0000	1111	1010	0011	0100

zero-extended

Assembly Code	Result								
`andi $s2, $s1, 0xFA34`	$s2	0000	0000	0000	0000	0000	0000	0011	0100
`ori  $s3, $s1, 0xFA34`	$s3	0000	0000	0000	0000	1111	1010	1111	1111
`xori $s4, $s1, 0xFA34`	$s4	0000	0000	0000	0000	1111	1010	1100	1011

② 移位指令

sll rd、rt、shamt

Assembly Code / Field Values

	Assembly Code	op	rs	rt	rd	shamt	funct	
逻辑左移:	`sll $t0, $s1, 2`	0	0	17	8	2	0	\$t0 <= \$s1 << 2
逻辑右移:	`srl $s2, $s1, 2`	0	0	17	18	2	2	\$s2 <= \$s1 >> 2
算数右移:	`sra $s3, $s1, 2`	0	0	17	19	2	3	\$s3 <= \$s1 >>> 2
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Machine Code

op	rs	rt	rd	shamt	funct	
000000	00000	10001	01000	00010	000000	(0x00114080)
000000	00000	10001	10010	00010	000010	(0x00119082)
000000	00000	10001	10011	00010	000011	(0x00119883)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

- 逻辑左移低位补0
- 逻辑右移高位补0
- 算术右移**高位补符号位**

② 移位指令

sllv rd、rt、rs

Assembly Code		Field Values						Machine Code						
		op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct	
可变逻辑左移	sllv $s3, $s1, $s2	0	18	17	19	0	4	000000	10010	10001	10011	00000	000100	(0x02519804)
可变逻辑右移	srlv $s4, $s1, $s2	0	18	17	20	0	6	000000	10010	10001	10100	00000	000110	(0x0251A006)
可变算术右移	srav $s5, $s1, $s2	0	18	17	21	0	7	000000	10010	10001	10101	00000	000111	(0x0251A807)
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

$s2低5位给出移位值

Source Values

$s1	1111	0011	0000	0100	0000	0010	1010	1000
$s2	0000	0000	0000	0000	0000	0000	000**0**	**1000**

Assembly Code	Result								
sllv $s3, $s1, $s2	$s3	0000	0100	0000	0010	1010	1000	0000	0000
srlv $s4, $s1, $s2	$s4	0000	0000	1111	0011	0000	0100	0000	0010
srav $s5, $s1, $s2	$s5	1111	1111	1111	0011	0000	0100	0000	0010

③ 生成常数指令

- **16-bit 常量 用 `addi`:**

C Code

```
int a = 0x4f3c;
```

MIPS assembly code

```
# $s0 = a
addi $s0, $0, 0x4f3c
```

- **32-bit 常量 用 `lui` (load upper immediate) 和 `ori`:**

C Code

```
int a = 0xFEDC_8765;
```

MIPS assembly code

```
# $s0 = a
lui $s0, 0xFEDC
ori $s0, $s0, 0x8765
```

lui指令：将一个16位立即数装入到寄存器的高16位，并将低16位都置0.

ori指令：将一个16位立即数合并到寄存器的低16位。

④ 乘法指令、除法指令

```
mult rs, rt
```

{[hi] , [lo]} = [rs] x [rt]

```
mult $s0, $s1
```

```
div rs, rt
```

[lo] = [rs] / [rt]　　商

[hi] = [rs] % [rt]　　余数

```
div $s0, $s1
```

⑤ 条件分支指令

beq rs, rt, label

branch if equal

bne rs, rt, label

branch if not equal

```
0x40           addi $s0, $0, 4        # $s0 = 0 + 4 = 4
0x44           addi $s1, $0, 1        # $s1 = 0 + 1 = 1
0x48           sll  $s1, $s1, 2       # $s1 = 1 << 2 = 4
0x4C           beq  $s0, $s1, target  # branch is taken
0x50           addi $s1, $s1, 1       # not executed
0x54           sub  $s1, $s1, $s0     # not executed
0x58   target: add  $s1, $s1, $s0     # $s1 = 4 + 4 = 8
```

PC

PC + 4

2

PC′

$$PC' = PC + 4 + (SignImm \ll 2)$$

beq $s0, $s1, target

op	rs	rt	imm
5	16	17	2
000101	10000	10001	0000 0000 0000 0010
6位	5位	5位	16位

⑥无条件分支指令

`j label`	Jump 跳转
`jr rs`	跳转到寄存器所保存的地址

```
 addi    $s0, $0, 4        # $s0 = 4
 addi    $s1, $0, 1        # $s1 = 1
 j       target            # jump to target
 sra     $s1, $s1, 2       # not executed
 addi    $s1, $s1, 1       # not executed
 sub     $s1, $s1, $s0     # not executed

target:
 add     $s1, $s1, $s0     # $s1 = 1 + 4 =5
```

⑦ 设置小于指令 set less than

`slt rd, rs, rt`

[rs] < [rt] ? [rd]=1 : [rd]=0

C Code

```c
// add the powers of 2
// from 1 to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2)
{
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
    addi $s0, $0, 1       # i=1
    addi $s1, $0, 0       # sum=0
    addi $t0, $0, 101     # $t0=101
loop:
    slt  $t1, $s0, $t0    # if(i<101) $t1=1,
                          # else      $t1=0
    beq  $t1, $0, done    # if $t1==0(i>=101)
                          # branch to done.
    add  $s1, $s1, $s0    # sum=sum+i
    sll  $s0, $s0, 1      # i=i*2
    j    loop
done:
```

if 语句

C Code

```c
if (i == j)
  f = g + h;
else
  f = f - i;
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j          (反着写)
   bne $s3, $s4, else   #if i!=j, branch to else
   add $s0, $s1, $s2    #if block: f = g + h
   j   done             #skip past the else block
else:
   sub $s0, $s0, $s3    #else block: f = f - i
done:
```

while 语句

C Code

```c
// determines the power
// of x such that 2x = 128
int pow = 1;
int x   = 0;

while (pow != 128)
{
  pow = pow * 2;
  x = x + 1;
}
```

MIPS assembly code

```
# $s0 = pow, $s1 = x

       addi $s0, $0, 1      # pow=1
       add  $s1, $0, $0     # x=0
       addi $t0, $0, 128    # t0=128
while: beq  $s0, $t0, done  # if pow==128,
                            # exit while loop
       sll  $s0, $s0, 1     # pow=pow*2
       addi $s1, $s1, 1     # x=x+1
       j    while
done:
```

for 语句

C Code

```c
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1)
{
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
    addi $s1, $0, 0       # sum=0
    add  $s0, $0, $0      # i=0
    addi $t0, $0, 10      # $t0=10
for:
    beq  $s0, $t0, done   # if i==10,
                          # branch to done
    add  $s1, $s1, $s0    # sum=sum+i
    addi $s0, $s0, 1      # increment i
    j    for
done:
```

MIPS 寻址方式

① **寄存器寻址**：寄存器（源操作数、目的操作数）
所有R指令。如，add rd, rs, rt

② **立即数寻址**：16位立即数
有些I指令，如，addi rt, rs, imm

③ **基地址寻址**：存储器地址 = 基地址 + 立即数扩展后
存储器访问指令，如，lw rt, imm(rs)

④ **PC相对寻址**：$\mathrm{PC}' = (\mathrm{PC} + 4) +$ 立即数符号扩展$\times 4$
条件分支指令，如，beq rs, rt, label

⑤ **伪直接寻址**：$\mathrm{PC}' = \{(\mathrm{PC} + 4)[31{:}28],\ \boldsymbol{addr},\ 2'b0\}$
跳转指令，如，j label

高级语言 → 汇编代码 → 机器代码

C 代码

```c
int f, g, y; //global variables

int main(void)
{
  f = 2;
  g = 3;
  y = sum(f, g);

  return y;
}

int sum(int a, int b)
{
  return (a + b);
}
```

编译 ⇒

MIPS汇编代码

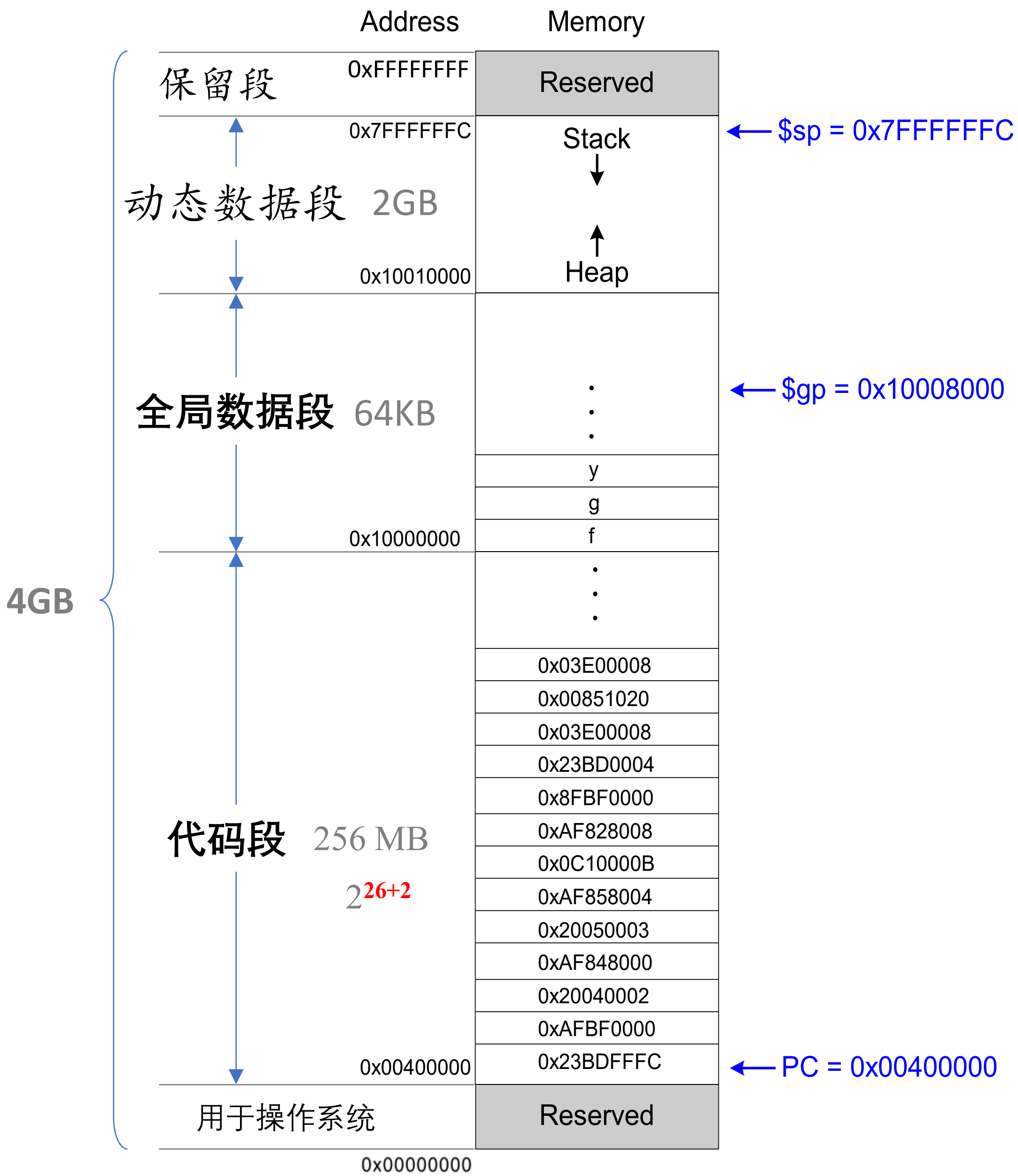
```
.data
f:
g:
y:
.text
main: addi $sp, $sp, -4   # stack frame
      sw   $ra, 0($sp)    # store $ra
      addi $a0, $0, 2     # $a0 = 2
      sw   $a0, f         # f = 2
      addi $a1, $0, 3     # $a1 = 3
      sw   $a1, g         # g = 3
      jal  sum            # call sum
      sw   $v0, y         # y = sum()
      lw   $ra, 0($sp)    # restore $ra
      addi $sp, $sp, 4    # restore $sp
      jr   $ra            # return to OS
sum:  add  $v0, $a0, $a1 # $v0 = a + b
      jr   $ra            # return
```

汇编 ⇒

符号	地址
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

代码段地址	MIPS机器代码
0x00400000	0x23BDFFFC
0x00400004	0xAFBF0000
0x00400008	0x20040002
0x0040000C	0xAF848000
0x00400010	0x20050003
0x00400014	0xAF858004
0x00400018	0x0C10000B
0x0040001C	0xAF828008
0x00400020	0x8FBF0000
0x00400024	0x23BD0004
0x00400028	0x03E00008
0x0040002C	0x00851020
0x00400030	0x03E00008

OS将可执行文件从硬盘装入内存



Executable file header	Text Size	Data Size	
统计信息	0x34 (52 bytes)	0xC (12 bytes)	
Text segment	**Address**	**Instruction**	
代码段	0x00400000	0x23BDFFFC	addi $sp, $sp, -4
	0x00400004	0xAFBF0000	sw $ra, 0 ($sp)
	0x00400008	0x20040002	addi $a0, $0, 2
	0x0040000C	0xAF848000	sw $a0, 0x8000 ($gp)
	0x00400010	0x20050003	addi $a1, $0, 3
	0x00400014	0xAF858004	sw $a1, 0x8004 ($gp)
	0x00400018	0x0C10000B	jal 0x0040002C
	0x0040001C	0xAF828008	sw $v0, 0x8008 ($gp)
	0x00400020	0x8FBF0000	lw $ra, 0 ($sp)
	0x00400024	0x23BD0004	addi $sp, $sp, -4
	0x00400028	0x03E00008	jr $ra
	0x0040002C	0x00851020	add $v0, $a0, $a1
	0x00400030	0x03E00008	jr $ra
Data segment	**Address**	**Data**	
全局数据段	0x10000000	f	
	0x10000004	g	
	0x10000008	y	