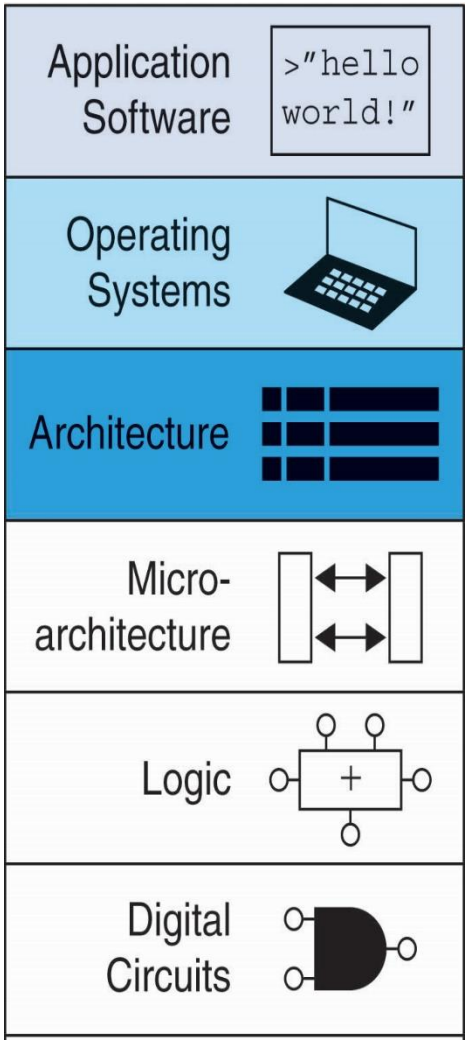
计算机体系结构实验

1. MIPS微处理器原理

- 体系结构（MIPS汇编语言）
- 微体系结构（单周期处理器）

结构层次



针对同一体系结构有不同的微体系结构设计

体系结构：**程序员所见到的计算机**。

MIPS、X86… 未定义底层的硬件实现。

指令集（汇编语言）

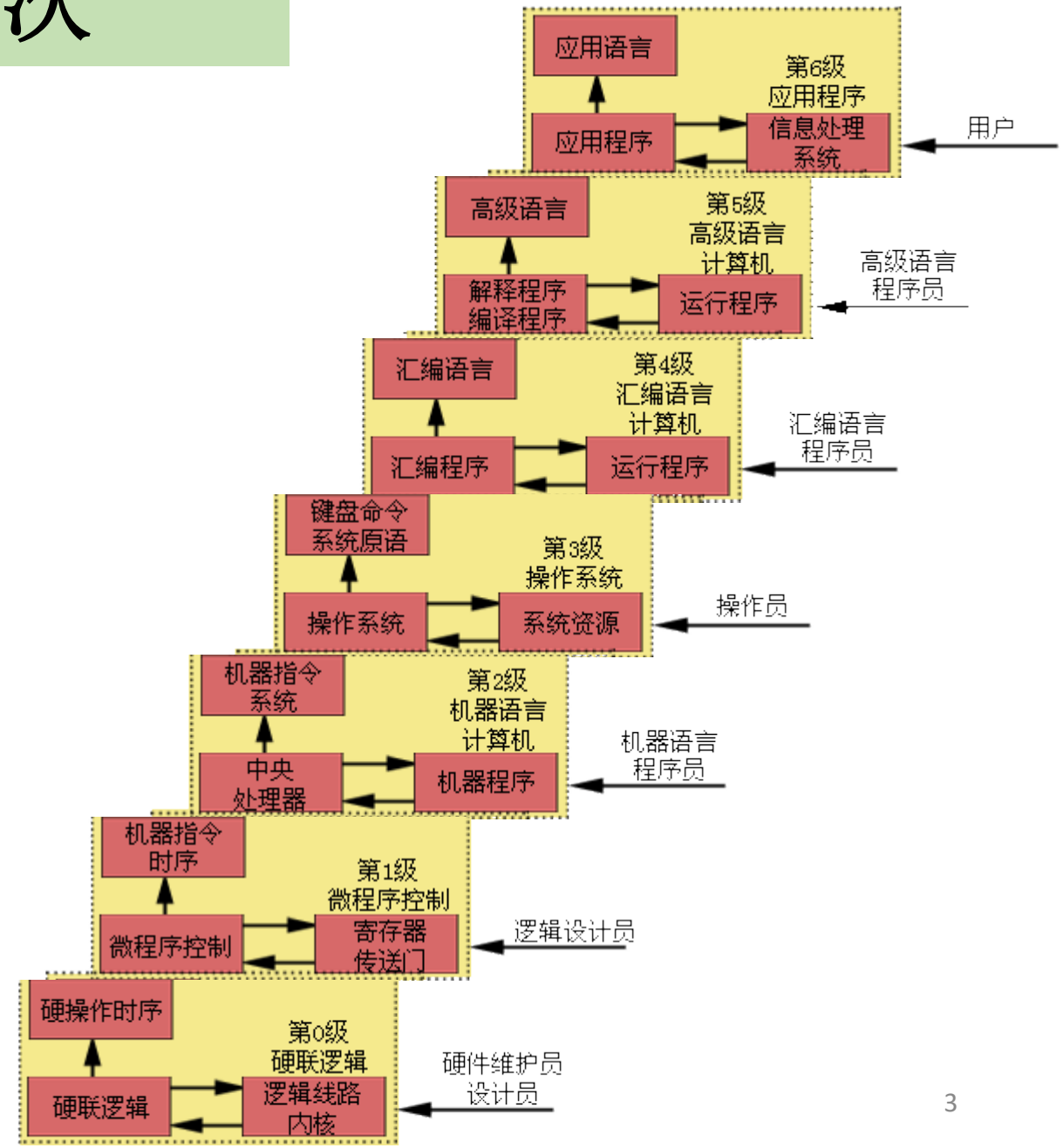
寄存器

微体系结构：由硬件实现一种**体系结构**。

（寄存器、存储器、ALU、有限状态机……）

数字逻辑

结构层次


应用语言
第6级
应用程序
应用程序
信息处理
系统
用户
高级语言
第5级
高级语言
计算机
解释程序
编译程序
运行程序
高级语言
程序员
汇编语言
第4级
汇编语言
计算机
汇编程序
运行程序
汇编语言
程序员
键盘命令
系统原语
第3级
操作系统
操作系统
系统资源
操作员
机器指令
系统
第2级
机器语言
计算机
中央
处理器
机器程序
机器语言
程序员
机器指令
时序
第1级
微程序控制
微程序控制
寄存器
传送门
逻辑设计员
硬操作时序
第0级
硬联逻辑
硬联逻辑
逻辑线路
内核
硬件维护员
设计员

MIPS体系结构

- Developed by John Hennessy and his colleagues at Stanford and in the 1980's.
- Used in many commercial systems, including Silicon Graphics, Nintendo, and Cisco.

4个准则：

① 简单设计有助于规整化；

② 加快常见功能；

③ 越小的设计越快；

④ 好的设计需要好的折中方法。

RISC 指令集的特点

Reduced **I**nstruction **S**et **C**omputer

Complex **I**nstruction **S**et **C**omputer

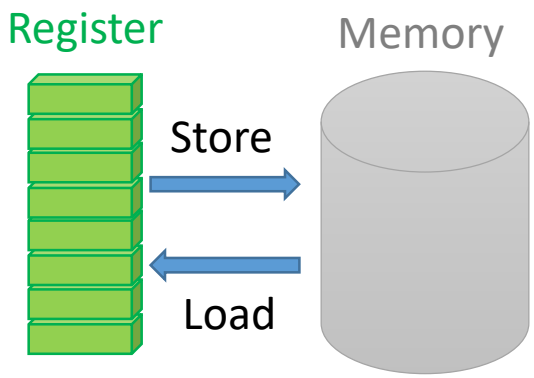
- 精简了指令系统，流水线以及常用指令均可用硬件执行；
- 采用大量的**寄存器**，使大部分指令操作都在寄存器之间进行，提高了处理速度；
- 每条指令的功能尽可能简单，并在一个机器周期内完成；
- 所有指令长度均相同；
- 只有Load和Store操作指令访问**存储器**，其他指令操作均在**寄存器**之间进行。

用来存放程序和数据的记忆设备

Memory

Register

中央处理器内的组成部分



MIPS Assembly Language 汇编语言

- 一种用于电子计算机、微处理器、微控制器或其他可编程器件的低级语言，亦称为**符号语言**。
- 在汇编语言中，用**助记符**代替**机器指令的操作码**，用**地址符号**(Symbol)或**标号**(Label)代替指令或**操作数的地址**。
- 在不同的设备中，汇编语言对应着不同的机器语言指令集，通过汇编过程转换成机器指令。
- 特定的汇编语言和特定的机器语言指令集是一一对应的，不同平台之间不可直接移植。

指令 Instruction

指令 instruction = 操作 operation + 操作数 operand

```
add   t , b ,  c
sub   a , t ,  d
```

a = b + c - d;

t = b + c;

a = t - d;

- **操作数**：可以存放在**寄存器**或**存储器**中，
  也可以作为常数存储在**指令**中。
- 访问存放在指令中的常数或寄存器中的操作数速度非常快；但其容量少；更多数据需要访问大容量存储器。

操作数：寄存器、存储器、常数

MIPS体系结构有32个32位寄存器。

a = b + c - d;

```
add  t , b ,  c
add $t0, $s1, $s2
sub  a , t ,  d
sub $s0, $t0, $s3
```

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	保存变量

MIPS Register Set

Name	Register Number	Usage
`$0`	0	the constant value 0
`$at`	1	assembler temporary
`$v0-$v1`	2-3	Function return values
`$a0-$a3`	4-7	Function arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved variables
`$t8-$t9`	24-25	more temporaries
`$k0-$k1`	26-27	OS temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	Function return address

操作数：寄存器、存储器、常数

- MIPS体系结构采用32位存储器地址，32位数据字长。
- MIPS采用字节(8位)寻址存储器，每1个字节都有1个单独地址。

装入字：

```
lw $s3, 1($0)
lw $s3, 4($0)
```

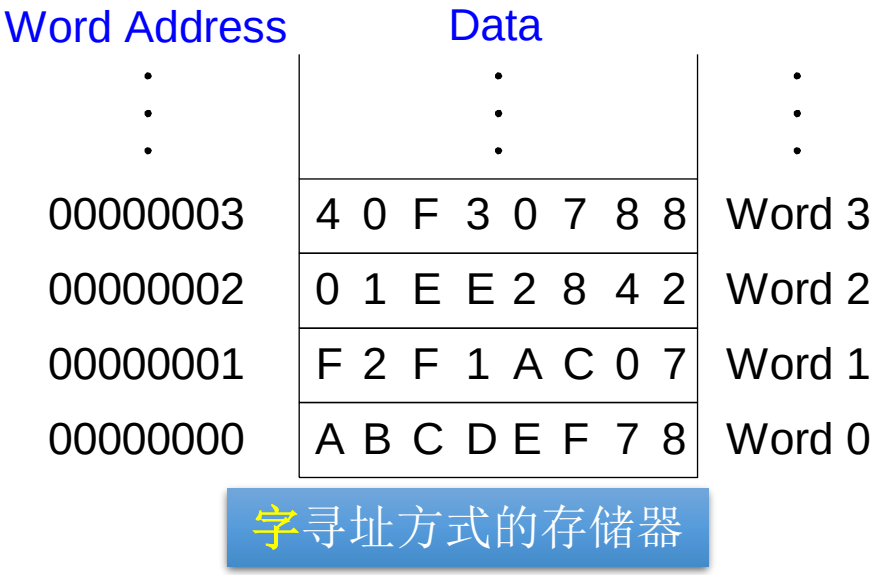
存储字：

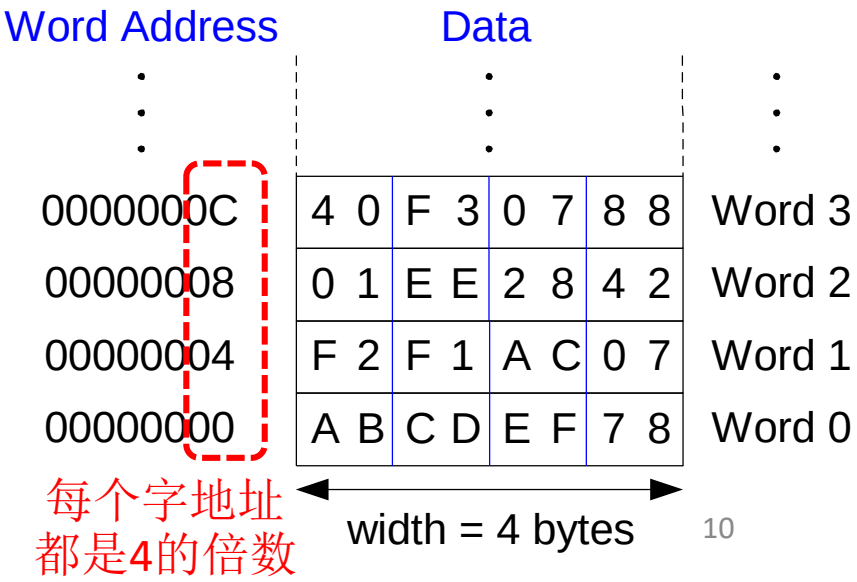
```
sw $s7, 5($0)
sw $s7, 10($0)
```

从存储器中读出数据放到$3中，
地址为1=$0+1，数据为 F2F1AC07

将$7中数据写入地址为5=$0+5
的存储器中

字节寻址存储器





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)。

加立即数指令 (**addi**)：

```
# $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。

MIPS指令集有3种指令格式

指令 instruction = 操作 operation + 操作数 operand

MIPS采用32位指令

```
add   a , b ,  c
```

a = b + c;

① Register类型
3寄存器

```
add $s0, $s1, $s2
```
保存寄存器

```
add $t0, $s1, $s2
```
临时寄存器

② Immediate类型
2寄存器+16位立即数

```
addi $s0, $s1, -4
```
16位 立即数

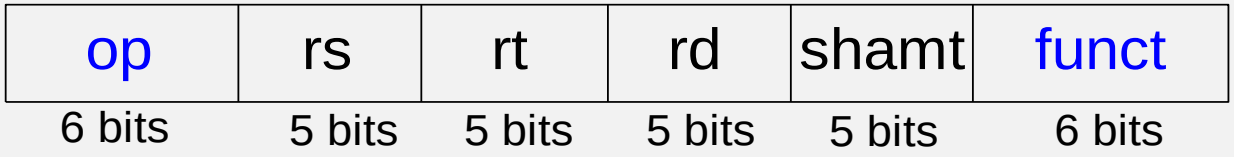
```
lw $s0, 8($0)
```
word2 →$s0

③ Jump类型
26位立即数

```
jr $s0
```
无条件跳转

① Register 类型机器指令格式 3寄存器

R-Type



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

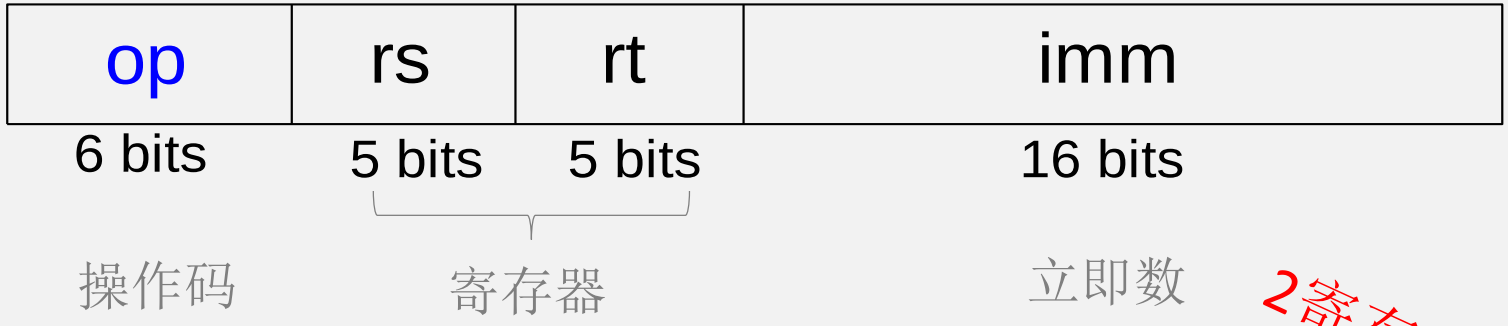
机器代码:

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

机器指令: 0x02328020

② Immediate 类型机器指令格式

I-Type



2寄存器+立即数

汇编代码: **addi $s0, $s1, -4**

rt rs

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

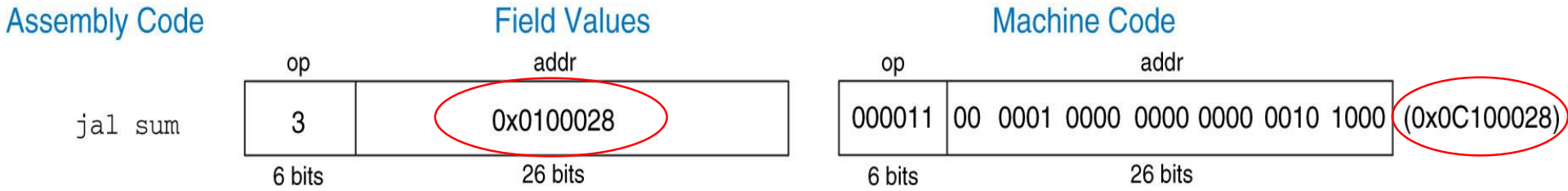
机器指令: **0x2232FFFC**

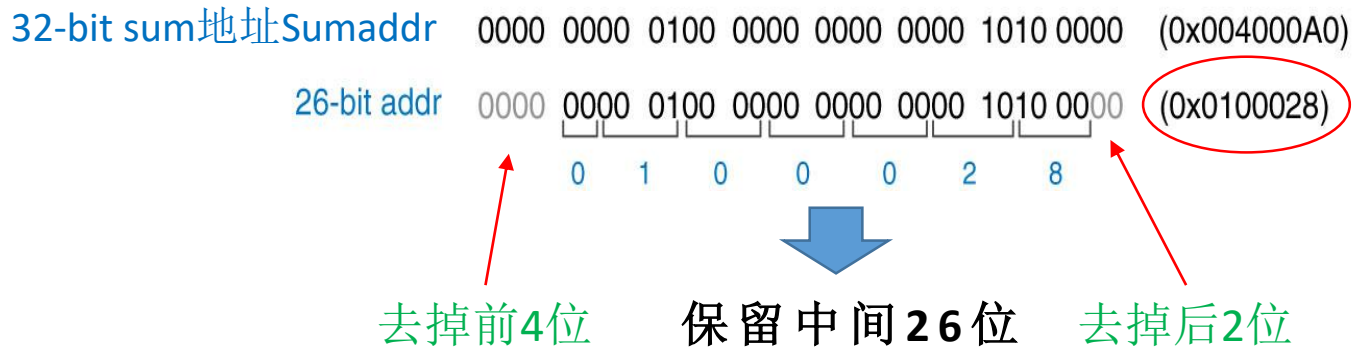
对正立即数，高16位都补0
对负立即数，高16位都补1

③ Jump 类型机器指令格式

26位立即数

J-Type



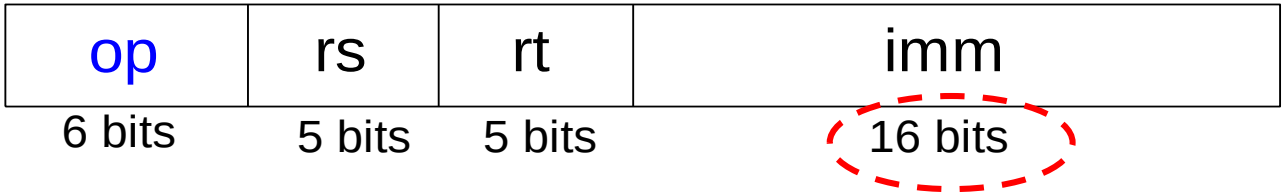


如果已知sum地址Sumaddr,则指令中的**addr=Sumaddr/4** (即去掉最后两位),再去掉最高4位

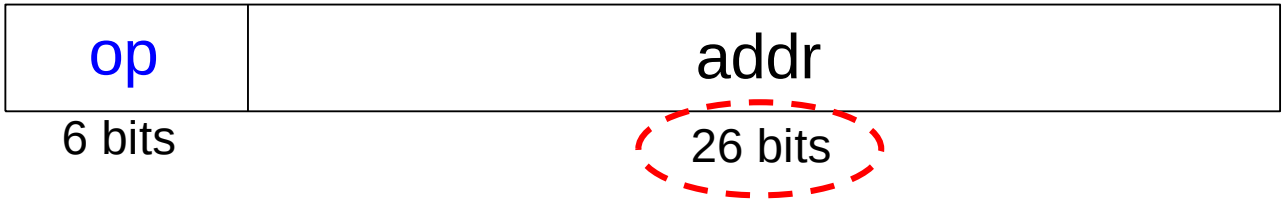
Review: Instruction Formats

R-Type

I-Type



J-Type



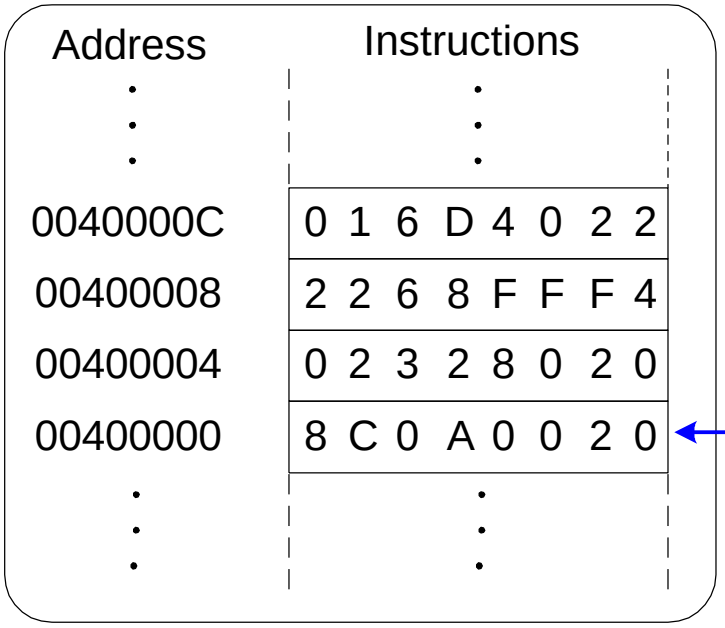
The Stored Program 存储程序

Assembly Code	Machine Code
`lw $t2, 32($0)`	`0x8C0A0020`
`add $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub $t0, $t3, $t5`	`0x016D4022`

用机器语言编写的程序：

一系列32位数表示的指令集.

Stored Program



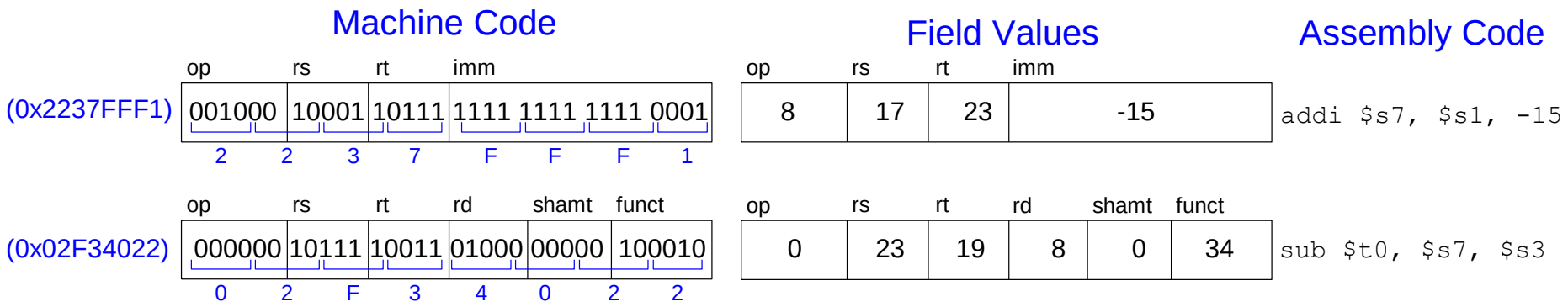
指令开始地址

Program **C**ounter

程序计数器

指示当前指令的地址

Interpreting Machine Code



程序运行步骤：

① 操作系统将PC值设为0x00400000

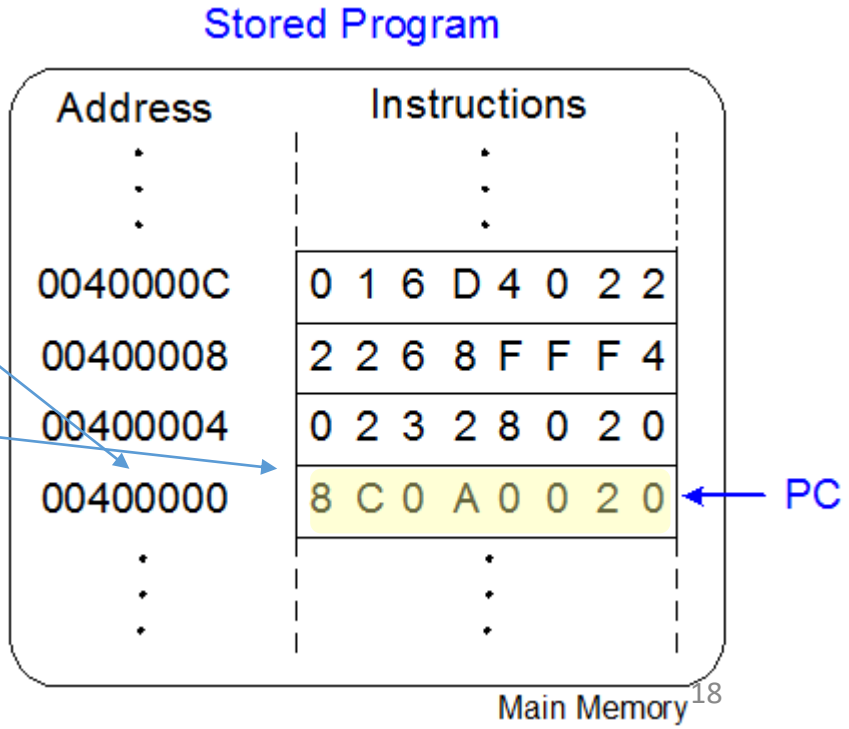
② 处理器读出这个地址的指令

③ 处理器执行0x8C0A0020指令

④ 处理器将**PC+4**

⑤ 取出并执行该地址指令

⑥ 重复上述过程.....



MIPS指令

① 逻辑指令

and rd、rs、rt

Assembly Code

```
and $s3, $s1, $s2
or  $s4, $s1, $s2
xor $s5, $s1, $s2
nor $s6, $s1, $s2
```

Source Registers

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

Result

$s3	0100	0110	1010	0001	0000	0000	0000	0000
$s4	1111	1111	1111	1111	1111	0000	1011	0111
$s5	1011	1001	0101	1110	1111	0000	1011	0111
$s6	0000	0000	0000	0000	0000	1111	0100	1000

没有NOT，可用下面代替

A NOR $0 = **NOT** A

andi rt、rs、imm

Assembly Code

```
andi $s2, $s1, 0xFA34
ori  $s3, $s1, 0xFA34
xori $s4, $s1, 0xFA34
```

Source Values

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

zero-extended

Result

$s2	0000	0000	0000	0000	0000	0000	0011	0100
$s3	0000	0000	0000	0000	1111	1010	1111	1111
$s4	0000	0000	0000	0000	1111	1010	1100	1011

MIPS指令

② 移位指令

Assembly Code

Field Values

		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	

sll rd、rt、shamt

- 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**。

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	

MIPS指令

② 移位指令

变量逻辑左移sllv、可变变量逻辑右移srlv、变量算术右移srav

Assembly Code	op	rs	rt	rd	shamt	funct	Machine Code	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits		
sllv $s3, $s1, $s2	0	18	17	19	0	4	000000 10010 10001 10011 00000 000100	(0x02519804)
srlv $s4, $s1, $s2	0	18	17	20	0	6	000000 10010 10001 10100 00000 000110	(0x0251A006)
srav $s5, $s1, $s2	0	18	17	21	0	7	000000 10010 10001 10101 00000 000111	(0x0251A807)

$s2低5位给出移位值

```
sllv rd、rt、rs
```

Source Values								
$s1	1111	0011	0000	0100	0000	0010	1010	1000
$s2	0000	0000	0000	0000	0000	0000	0000	**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

MIPS指令

③ 生成常数指令

- **16-bit** constants using `addi`:

C Code

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

MIPS assembly code

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

- **32-bit** constants using load upper immediate (`lui`) and `ori`:

C Code

```
int a = 0xFEDC8765;
```

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。

MIPS指令

④ 乘法指令、除法指令

```
mult rs, rt
```

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

```
div rs, rt
```

[lo] = [rs] / [rt], [hi] = [rs] % [rt]

- Special registers: `lo`, `hi`
- 32 × 32 multiplication, **64** bit result
  - **`mult`** `$s0, $s1` `# result in {hi, lo}`
- 32-bit division, 32-bit quotient, remainder
  - **`div`** `$s0, $s1`
  - 商 Quotient in `lo`, 余数 Remainder in `hi`
- Moves from `lo`/`hi` special registers
  - `mflo $s2`、`mfhi $s3`

MIPS指令

⑤ 条件分支指令

`beq rs, rt, label` branch if equal

`bne rs, rt, label` branch if not equal

```
 addi $s0, $0, 4          # $s0 = 0 + 4 = 4
 addi $s1, $0, 1          # $s1 = 0 + 1 = 1
 sll  $s1, $s1, 2         # $s1 = 1 << 2 = 4
 beq  $s0, $s1, target    # branch is taken
 addi $s1, $s1, 1         # not executed
 sub  $s1, $s1, $s0       # not executed

target:                   # label
 add  $s1, $s1, $s0       # $s1 = 4 + 4 = 8
```

MIPS指令

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
```

高级语言结构 → MIPS汇编代码

⑦ 设置小于指令

set less than

`slt rd, rs, rt`

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

C Code

```
// 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 $s1, $0, 0     # sum=0
   addi $s0, $0, 1     # i=1
   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:
```

高级语言结构 → MIPS汇编代码

if 语句

C Code

```
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:
```

高级语言结构 → MIPS汇编代码

while 语句

C Code

```
// determines the power
// of x such that 2^x = 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:
```

高级语言结构 → MIPS汇编代码

for 语句

C Code

```
// 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**：使用寄存器存储所有源操作数和目的操作数。

所有R指令。如， add rd, rs, rt

② **立即数寻址**：使用16位立即数和寄存器作为操作数。

有些I指令，如， addi rt, rs, imm

③ **基地址寻址**：操作数中的地址 = rs中基地址 + 立即数扩展后

存储器访问指令，如， lw rt, imm(rs)

④ **PC相对寻址**：$PC' = PC + 4 +$立即数符号扩展$\times 4$ (将字转化为字节)

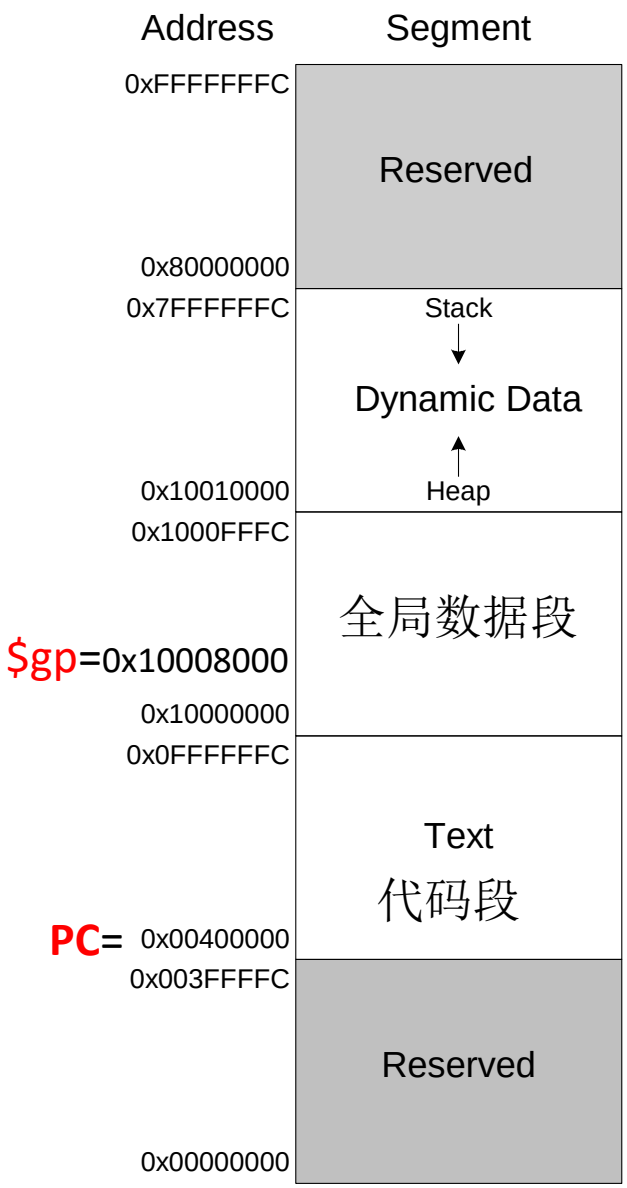
条件分支指令，如， beq rs, rt, label

⑤ **伪直接寻址**：指令中只有26位表达跳转目的地址，不是32位.

$PC' = \{(PC + 4)[31:28], addr, 2'b0\}$ 组合出32位

跳转指令，如， j label

MIPS Memory Map 内存映射



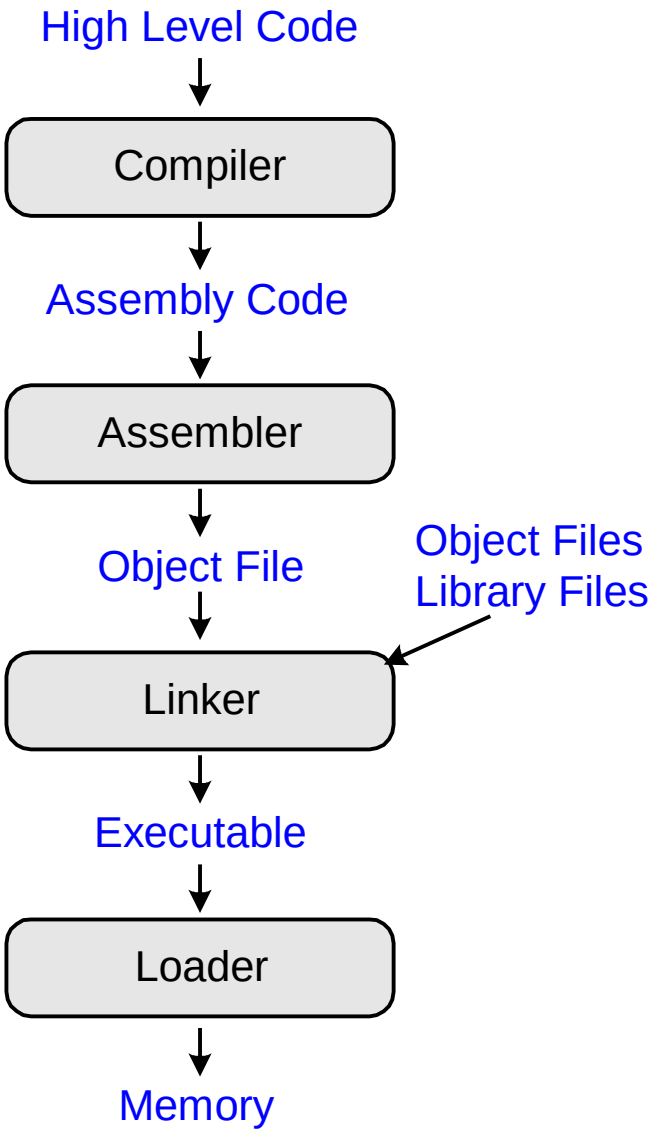
定义代码、数据和栈内存中的存储位置。

MPS地址的宽度为**32位**，**4GB**地址空间

2^{32} = 4 gigabytes (4 GB)

- **动态数据段**：保存栈、堆。2GB空间
  - 栈：保存和恢复函数使用的寄存器。
  - 堆：存储运行时程序分配的数据。
- **全局数据段**：存储全局变量。
  可容纳64KB全局变量
- **代码段**：存储机器语言程序。
  可容纳256MB代码
- **保留段**：用于操作系统。
  不能直接被程序使用

高级语言转换为机器代码并执行



C Code → 编译 → MIPS Assembly

```
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);
}
```

```
.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
```

MIPS Assembly 汇编 机器语言

机器语言代码

0x23BDFFFC	addi $sp, $sp, -4
0xAFBF0000	sw $ra, 0 ($sp)
0x20040002	addi $a0, $0, 2
0xAF848000	sw $a0, 0x8000 ($gp)
0x20050003	addi $a1, $0, 3
0xAF858004	sw $a1, 0x8004 ($gp)
0x0C10000B	jal 0x0040002C
0xAF828008	sw $v0, 0x8008 ($gp)
0x8FBF0000	lw $ra, 0 ($sp)
0x23BD0004	addi $sp, $sp, -4
0x03E00008	jr $ra
0x00851020	add $v0, $a0, $a1
0x03E00008	jr $ra

符 号 表

Symbol	Address
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

链接为可执行文件

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)

Text segment	Address	Instruction
	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

Data segment	Address	Data
	0x10000000	f
	0x10000004	g
	0x10000008	y

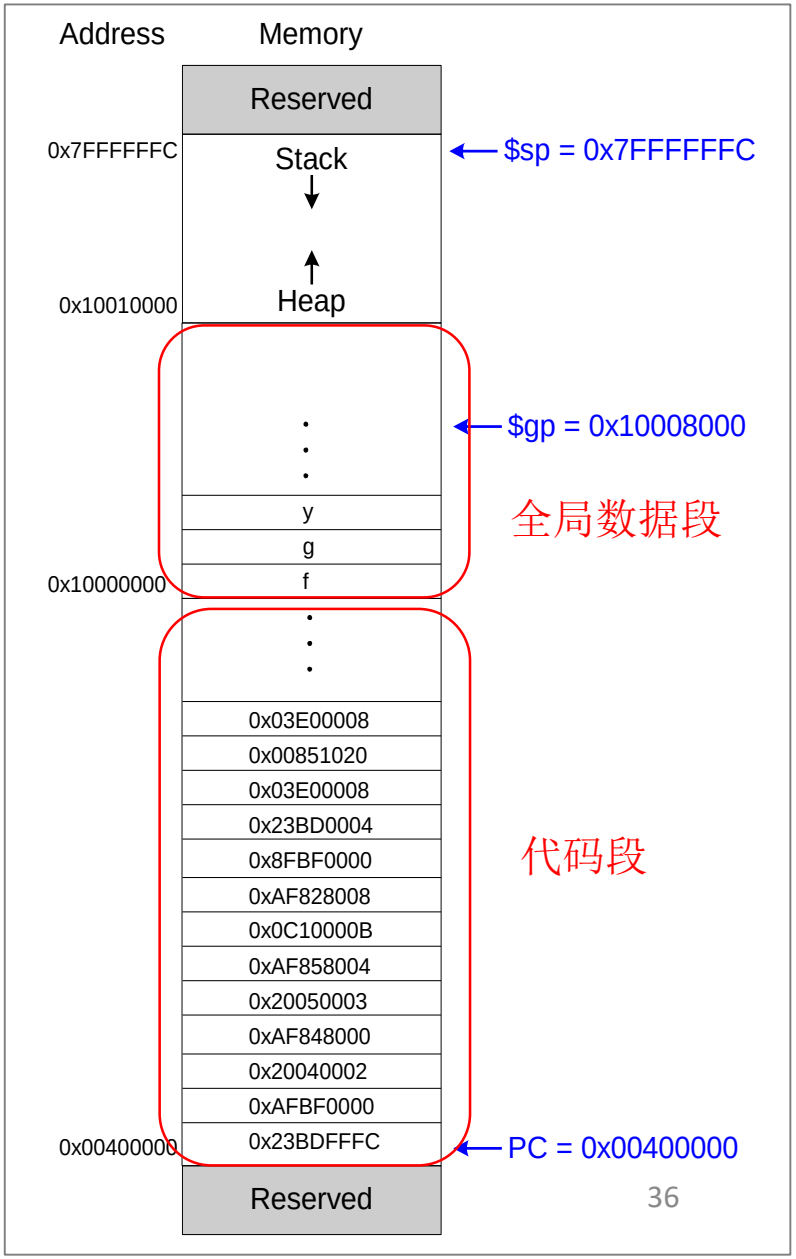
```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```

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
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E00008

Data segment	Address	Data
全局数据段	0x10000000	f
	0x10000004	g
	0x10000008	y



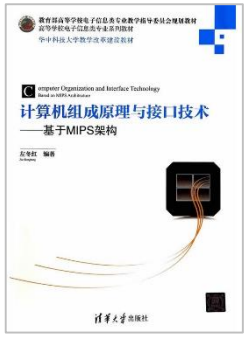
参考资料

数字设计和计算机体系结构

Digital Design and Computer Architecture 2nd

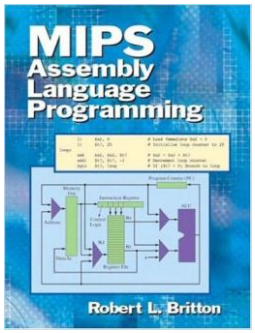
David Money Harris，陈俊颖 译

机械工业出版社，2016，**第6章 体系结构**



计算机组成原理与接口技术: 基于MIPS架构

左冬红，清华大学出版社，2014



MIPS Assembly Language Programming

Robert Britton

Publisher: Pearson, 2003