



# 计算机组织与体系结构

## Computer Architectures

陆俊林

北京大学本科生主干基础课

2018.3.21

## 第四讲 RISC和MIPS指令（1）

### 本讲要点

首先简介RISC兴起的历程，其次分析MIPS指令的设计原则和主要特点，然后按照指令格式分类讲解主要的MIPS指令，本讲主要分析R型和I型的典型指令，其他指令类型将在下一讲讲解。

阅读教材“COD”：第2章，附录E

# 主要内容

通过学习本课程  
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法



**I RISC的发展变迁**

**II MIPS指令的主要特点**

**III MIPS指令分类说明：R型**

**IV MIPS指令分类说明：I型**





# RISC的先驱，两位传奇人物



戴维·帕特森  
David Patterson  
1947年出生



约翰·亨尼西  
John Hennessy  
1953年出生

# John Hennessy



- 1977年，进入斯坦福大学任职
- 1981年，领导RISC微处理器的研究小组
- 1984年，共同创立MIPS计算机系统公司
- 1989年~1999年，先后担任斯坦福大学计算机系统实验室主任、计算机系主任和工程学院院长等
- 2000年至2016年，任斯坦福大学校长
- 2018年，出任Alphabet董事长



约翰·亨尼斯  
John Hennessy  
1953年出生



IEEE Medal of Honor “for pioneering the RISC processor architecture and for leadership in computer engineering and higher education”

# MIPS公司的商业兴衰

- ④ 1984年，MIPS计算机系统公司成立
  - ④ 1988年，SGI公司在其计算机产品中采用MIPS处理器
  - ④ 1989年，MIPS第一次上市
  - ④ 1992年，SGI收购MIPS，更名为MIPS技术公司
  - ④ 1998年，MIPS再次上市
  - ④ 2012年，Imagination Technologies收购MIPS
- 
- ④ MIPS处理器广泛应用的领域：
    - 数字电视、机顶盒、蓝光播放器、游戏机、网络设备等

# MIPS指令的发展

1985年, R2000

1990年, R3000

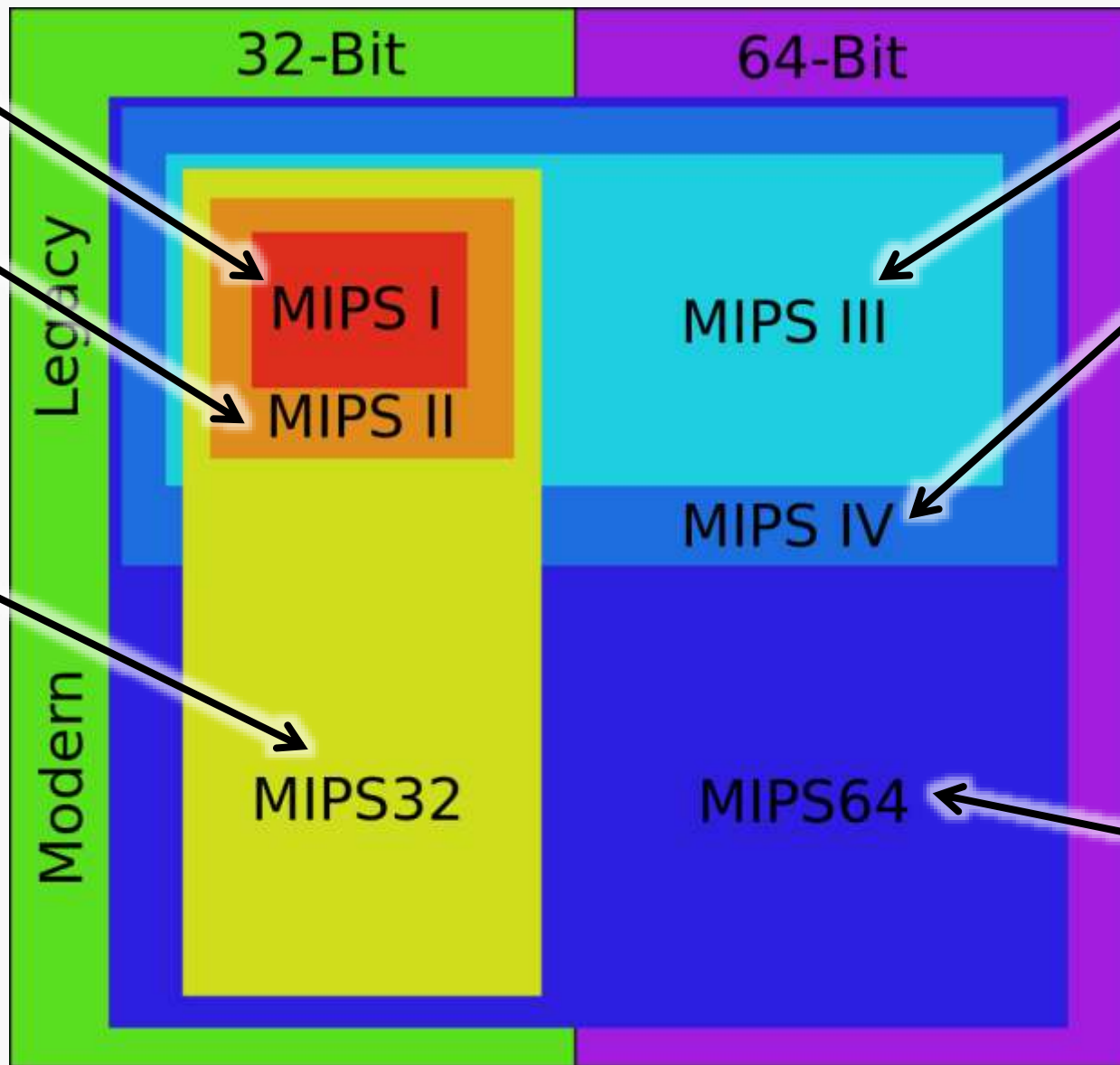
1999年  
以MIPS II为基础,  
增加了MIPS  
III/IV/V的部分特性

1992年, R4000  
扩展到64位

1994年, R8000

MIPS V  
1996年

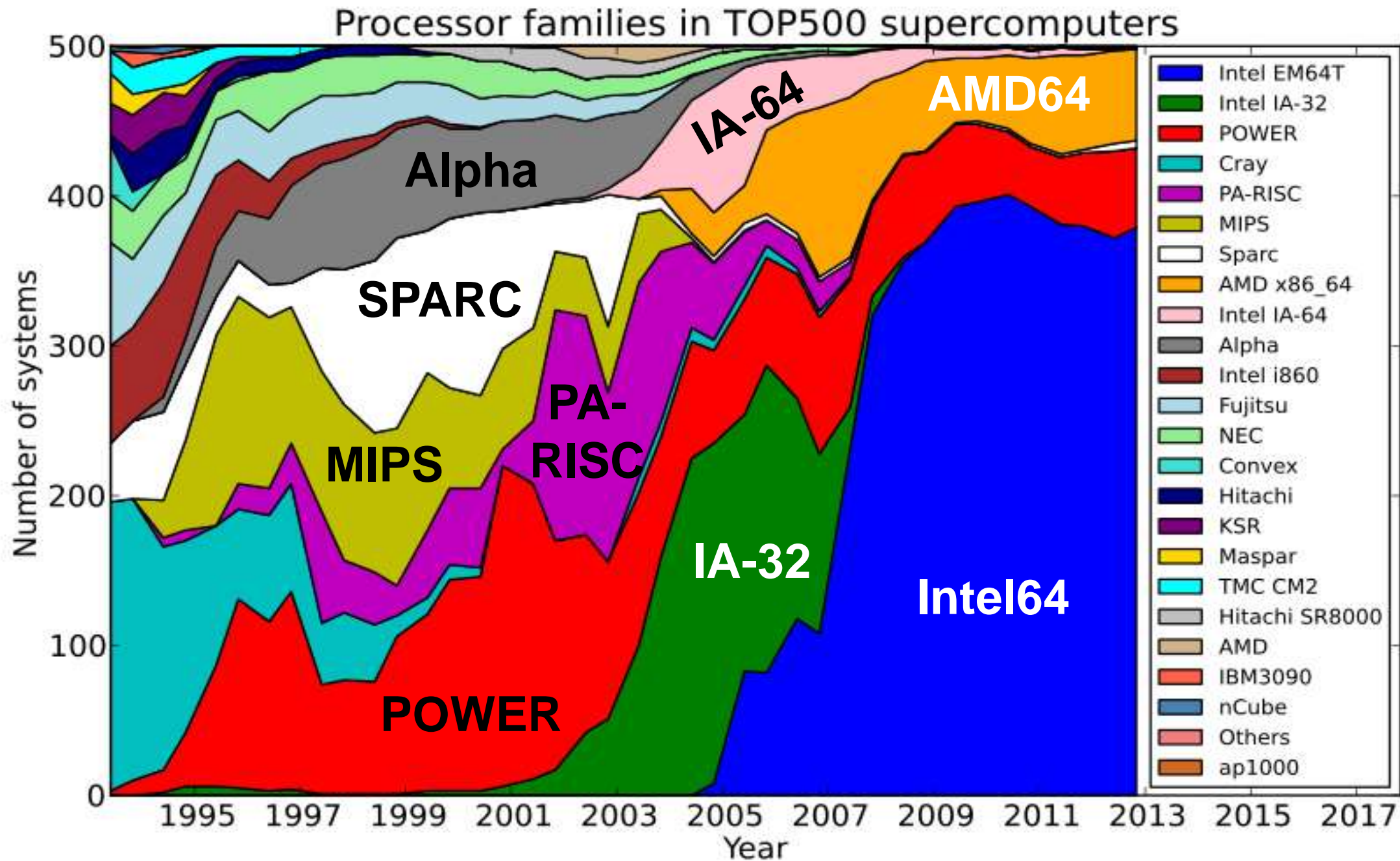
1999年  
以MIPS V为基础





超级计算领域

CISC vs. RISC

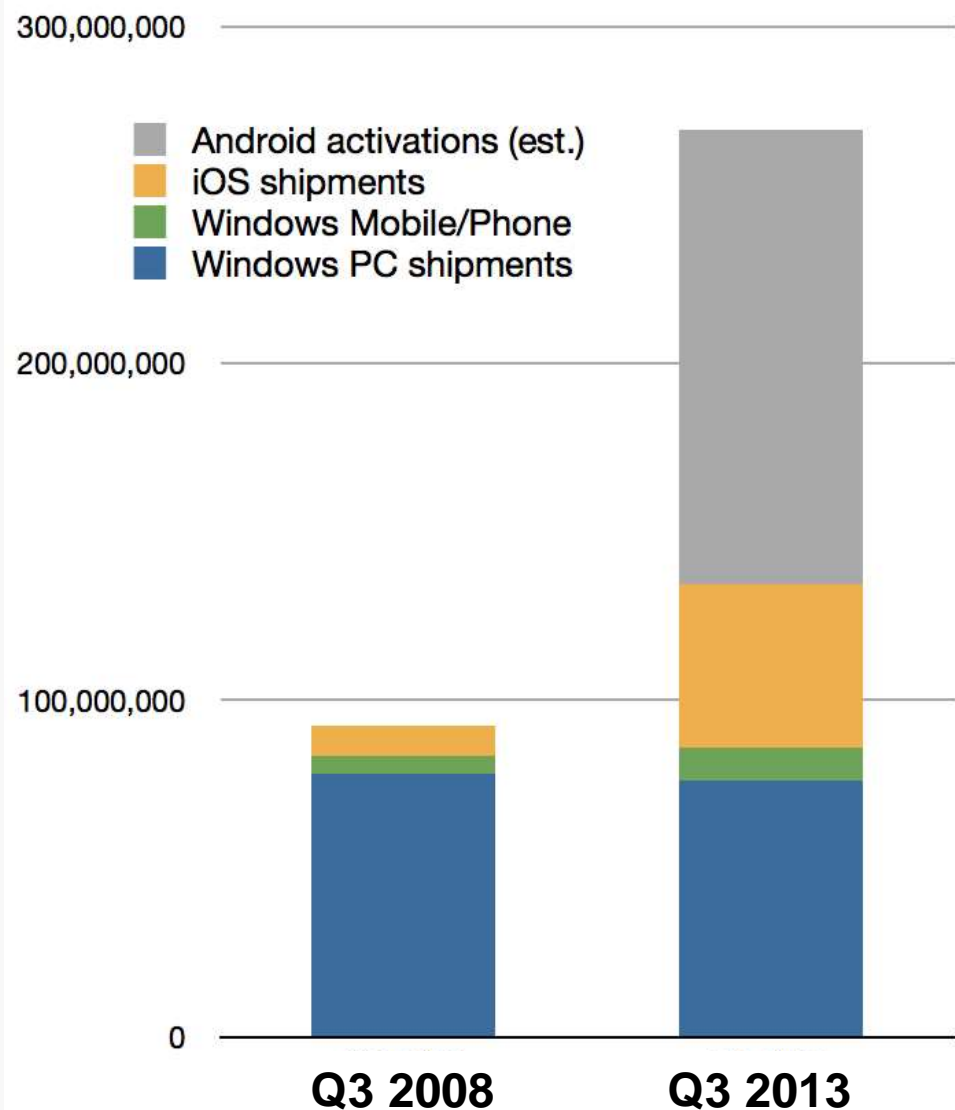




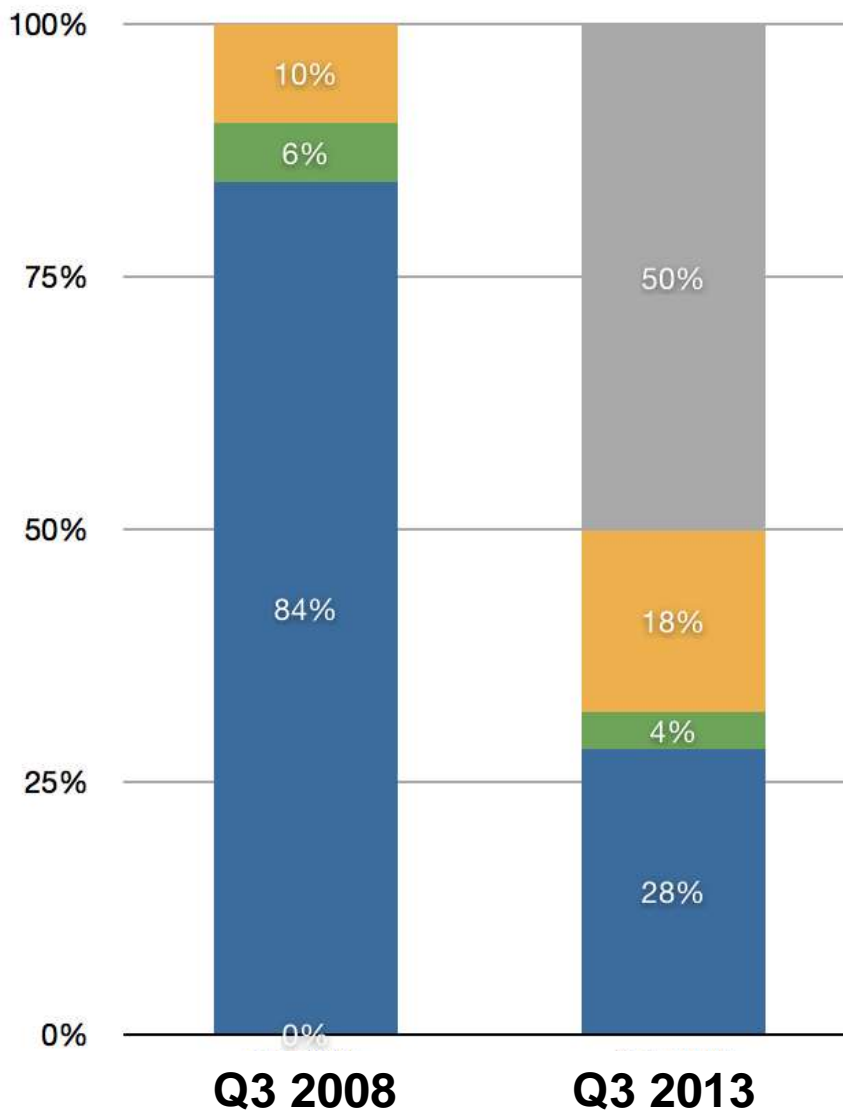
# 个人计算领域 CISC vs. RISC



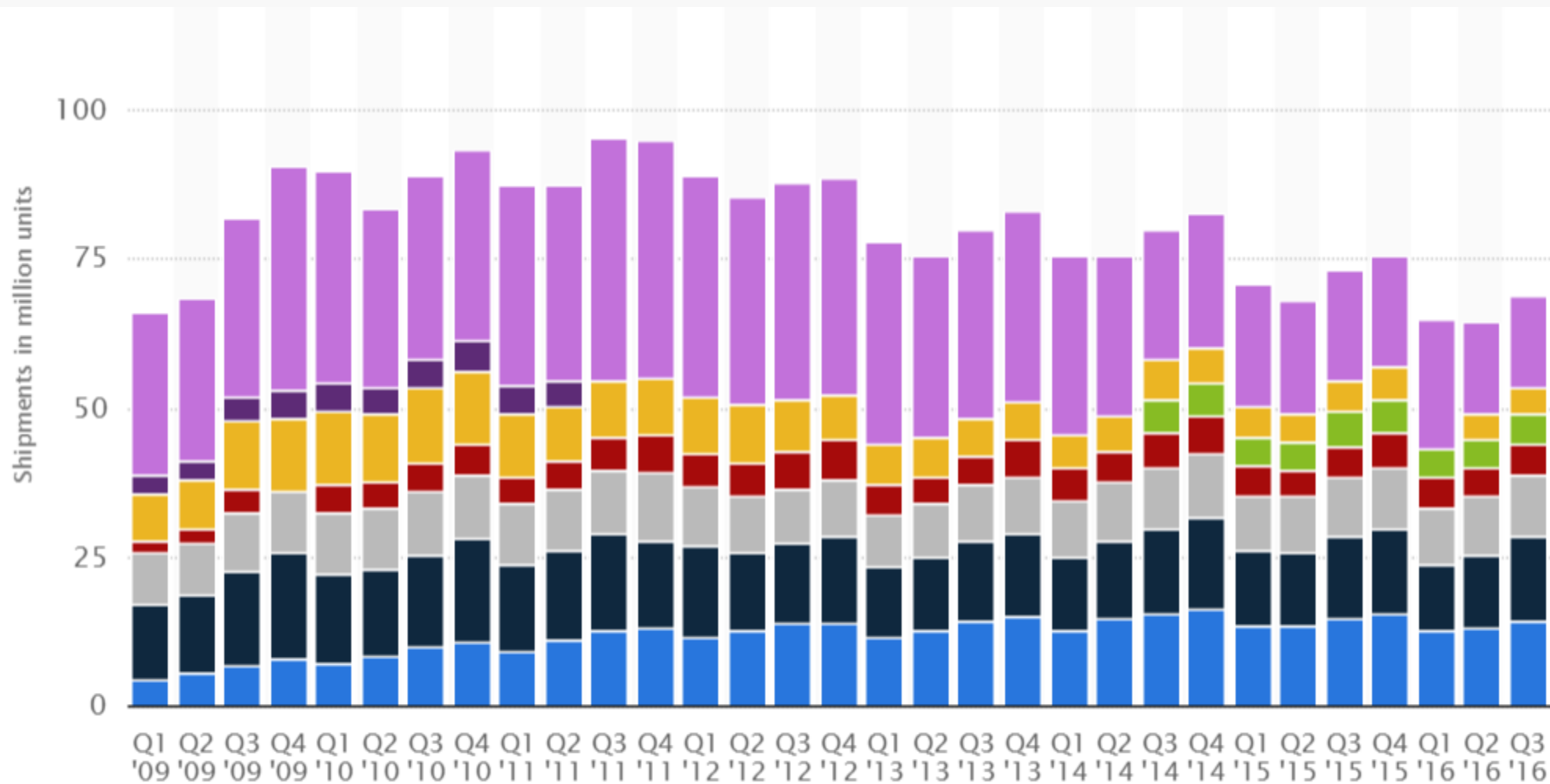
Change in Platforms (Q3 data, 2008, 2013)



Percent of Total



# 个人计算机的市场出货量（2009-2016）

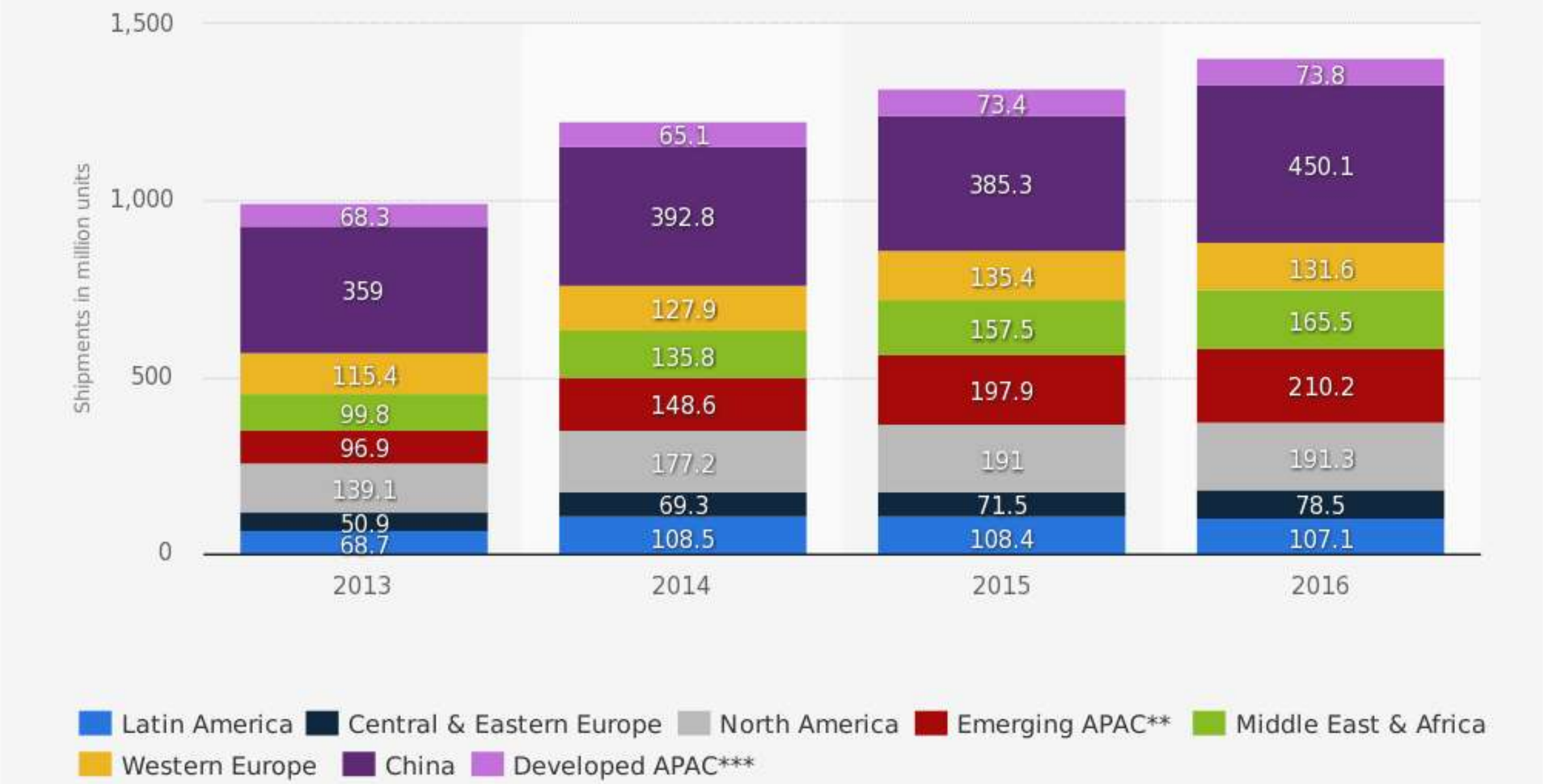


Q3 '16	
• Lenovo	14.43
• HP Inc**	14.06
• Dell	10.11
• Asus*	5.4
• Apple	4.95
• Acer	4.61
• Others	15.39

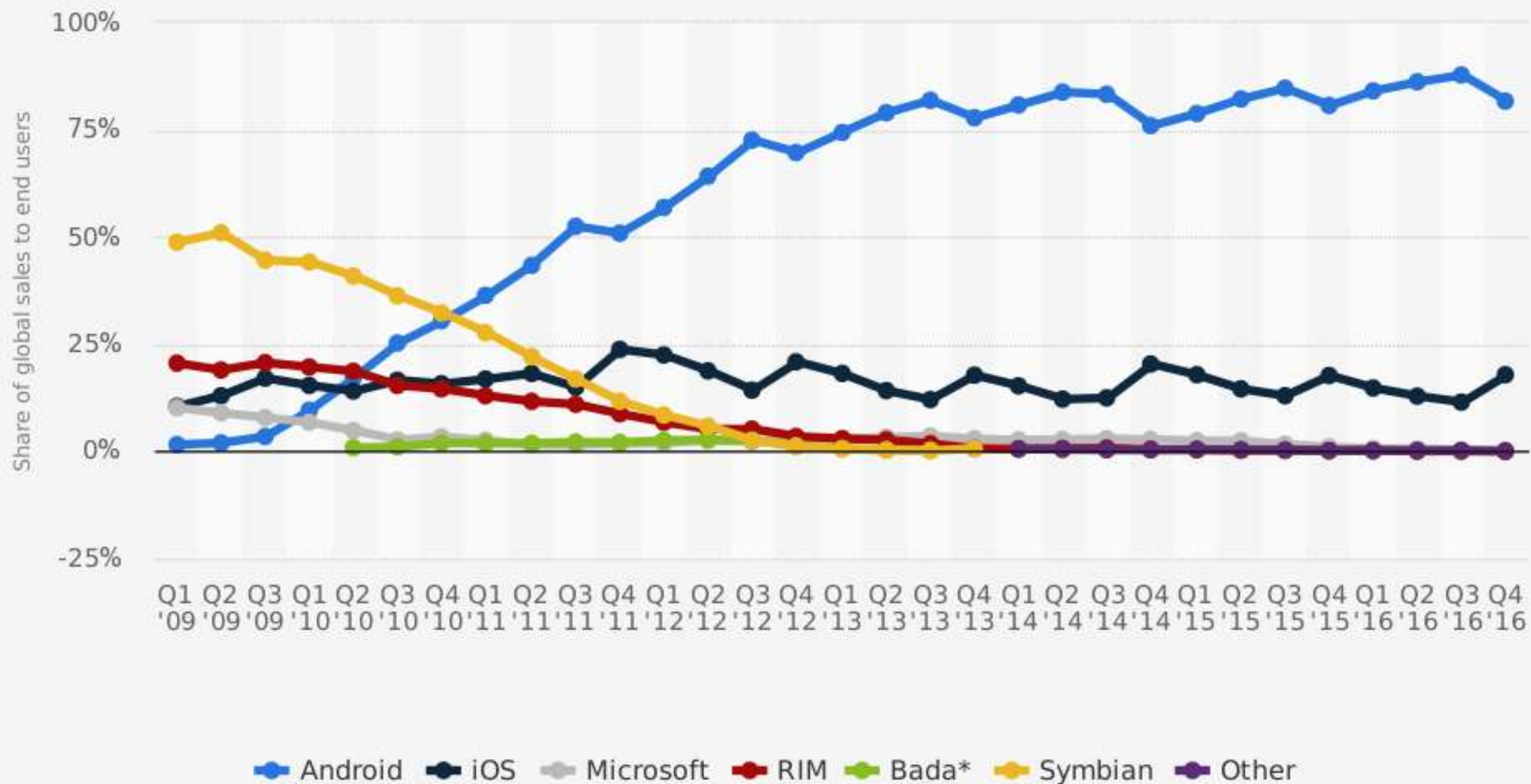
Lenovo HP Inc\*\* Dell Asus\* Apple Acer Toshiba\* Others



# 智能手机的市场出货量（2013-2016）

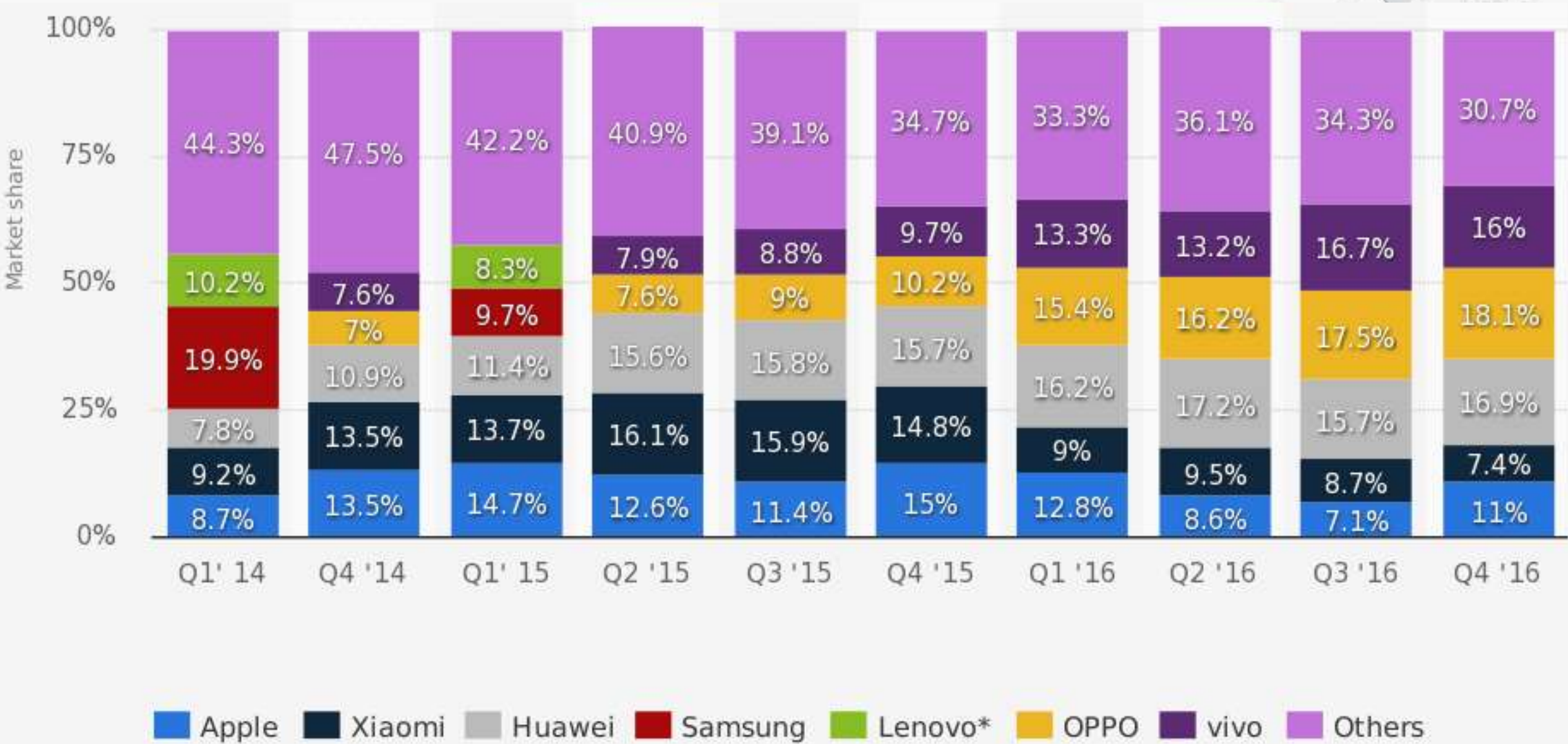


# 智能手机操作系统的市场份额（2009-2016）





# 中国的智能手机市场份额（2014-2016）



# 主要内容

通过学习本课程  
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I RISC的发展变迁



II MIPS指令的主要特点

III MIPS指令分类说明：R型

IV MIPS指令分类说明：I型



# MIPS的设计指导思想



## 🔍 MIPS的全称

- Microprocessor without Interlocked Piped Stages

## 🔍 主要关注点

- 减少指令的类型
- 降低指令复杂度

## 🔍 基本原则

- A simpler CPU is a faster CPU.

# MIPS指令示例



## 🔍 装载

- 格式: **lw \$8, (\$19)**
- 操作: 以19号寄存器的内容为地址, 取出存储器中的32位数据, 存入8号寄存器

## 🔍 加法

- 格式: **add \$10, \$9, \$8**
- 操作: 将8号和9号寄存器的内容相加, 结果存入10号寄存器中

## 🔍 存储

- 格式: **sw \$10, 32(\$19)**
- 操作: 将10号寄存器的内容存入存储器, 地址为19号寄存器的内容加32



# MIPS的通用寄存器（32个，每个都是32位宽）



编号	名称	用途	编号	名称	用途
<b>0</b>	\$zero	The Constant Value 0	<b>24-25</b>	\$t8-\$t9	Temporaries
<b>1</b>	\$at	Assembler Temporary	<b>26-27</b>	\$k0-\$k1	Reserved for OS Kernel
<b>2-3</b>	\$v0-\$v1	Values for Function Results and Expression Evaluation	<b>28*</b>	\$gp	Global Pointer
<b>4-7</b>	\$a0-a3	Arguments	<b>29*</b>	\$sp	Stack Pointer
<b>8-15</b>	\$t0-\$t7	Temporaries	<b>30*</b>	\$fp	Frame Pointer
<b>16-23*</b>	\$s0-\$s7	Saved Temporaries	<b>31*</b>	\$ra	Return Address

\* Preserved across a call

# 通用寄存器使用示例



🔍 以下指令与对应注释中的指令相同

```
lw    $t0, 12($s3)
```

```
# lw    $8, 12($19)
```

```
add   $t0, $s2, $t0
```

```
# add   $8, $18, $8
```

```
sw    $t0, 40($s3)
```

```
# sw    $8, 40($19)
```

编号	名称	用途
<b>8-15</b>	\$t0-\$t7	Temporaries
<b>16-23</b>	\$s0-\$s7	Saved Temporaries

# MIPS指令示例

🔍 假设变量和寄存器的对应关系如下

$f \rightarrow \$s0$	$g \rightarrow \$s1$	$h \rightarrow \$s2$
$i \rightarrow \$s3$	$j \rightarrow \$s4$	

$$f = (g + h) - (i + j)$$

```
add $t1, $s3, $s4
```

```
add $t2, $s1, $s2
```

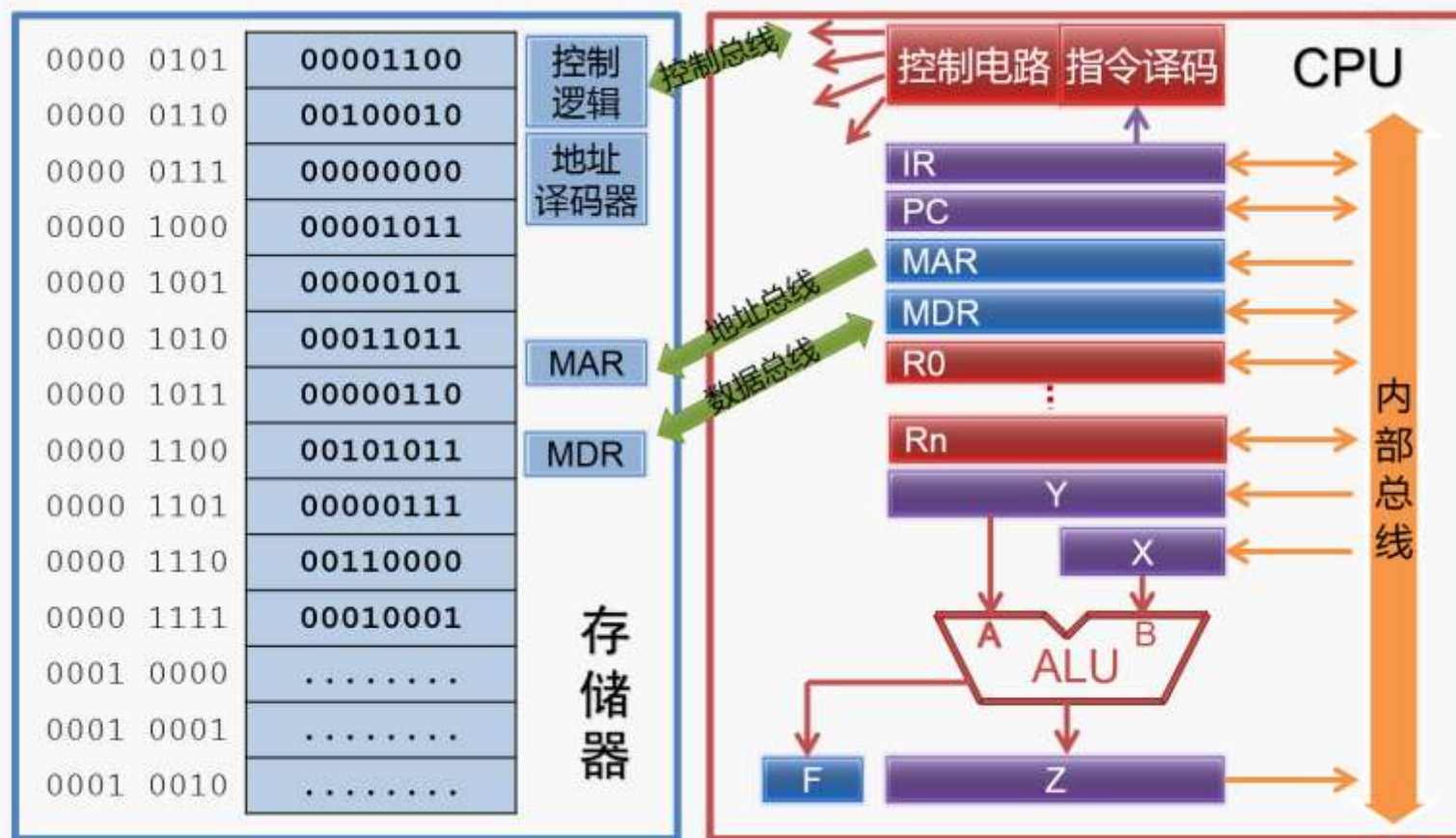
```
sub $s0, $t2, $t1
```

# MIPS指令的主要特点（1）

- 固定的指令长度（32-bit，即1 word）
  - 简化了从存储器取指令

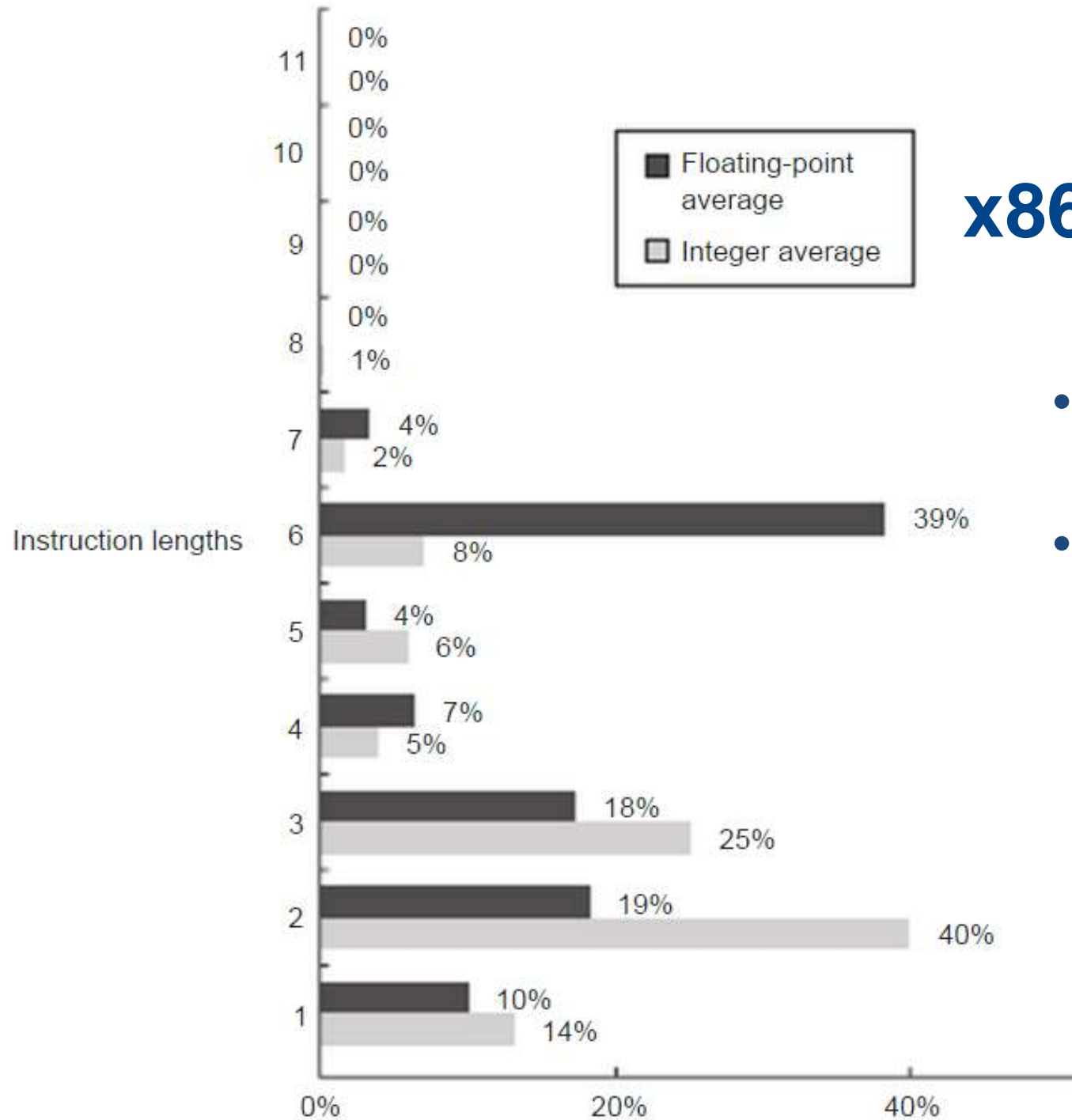
x86指令

- 长度不确定
- 最短1个字节
- 长可达15个字节





## x86指令不同长度的使用比例



- 整数指令平均长度2.8个字节
- 浮点指令平均长度4.1个字节

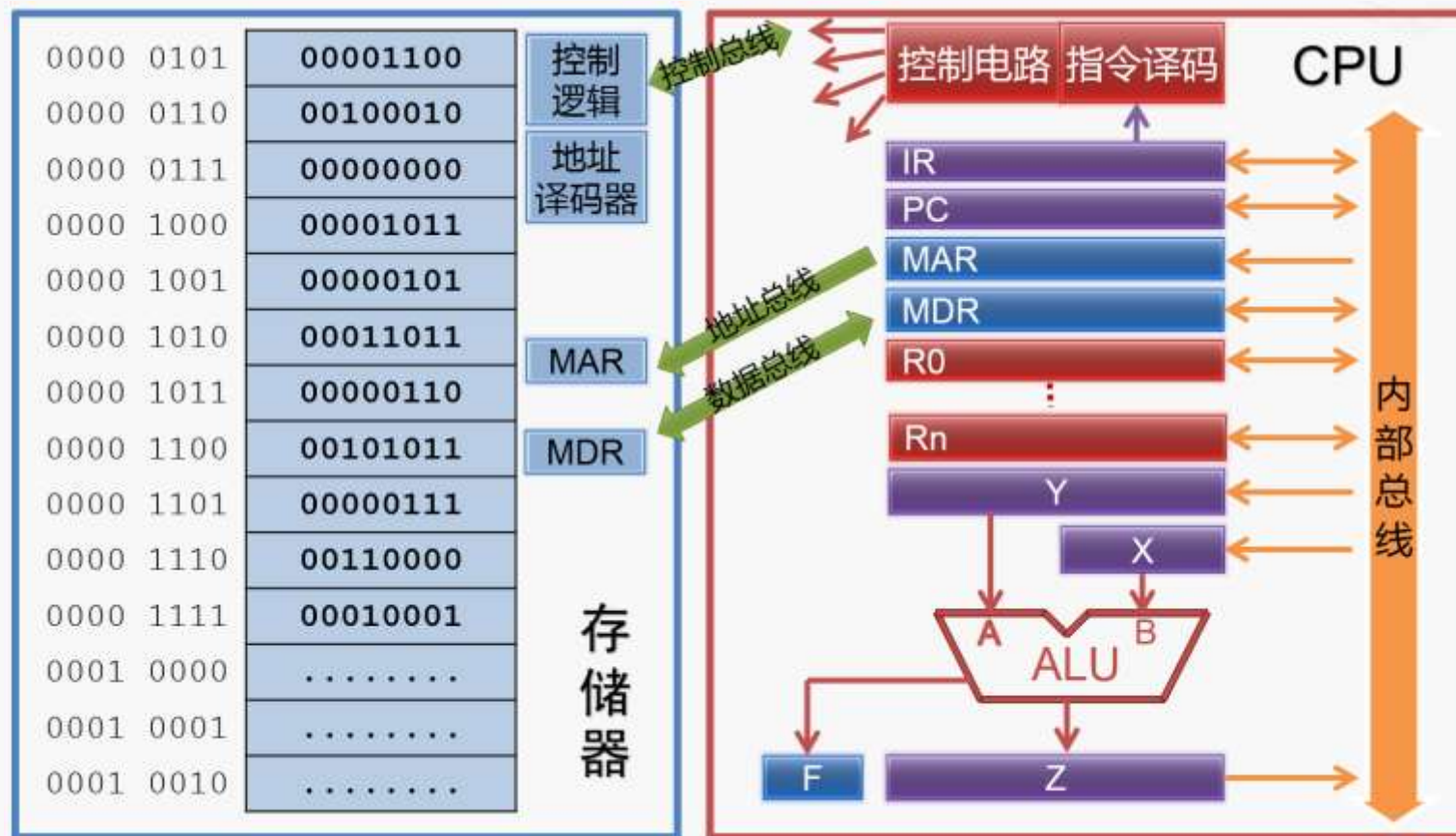
## MIPS指令的主要特点（2）

- 只有Load和Store指令可以访问存储器

例如，不支持x86

指令的这种操作：

ADD AX, [3000H]



# MIPS指令的主要特点 (3)

## 简单的寻址模式

- 简化了从存储器取操作数

**lw** \$8, (\$19)

**sw** \$10, 32(\$19)

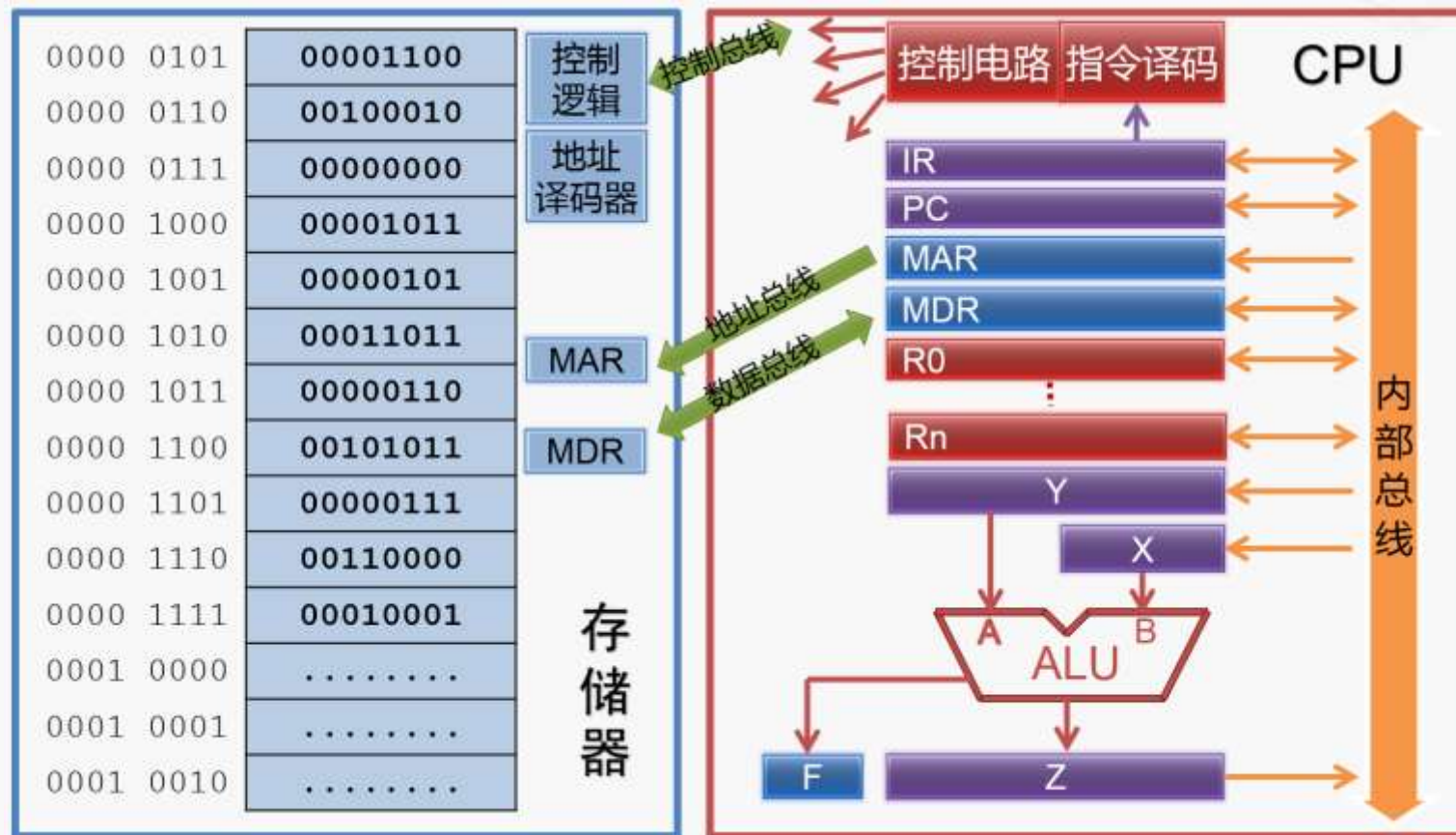
MOV EBX, 40

MOV AL, BL

MOV ECX, [1000H]

MOV [DI], AX

MOV DX, [BX+SI\*2+200H]



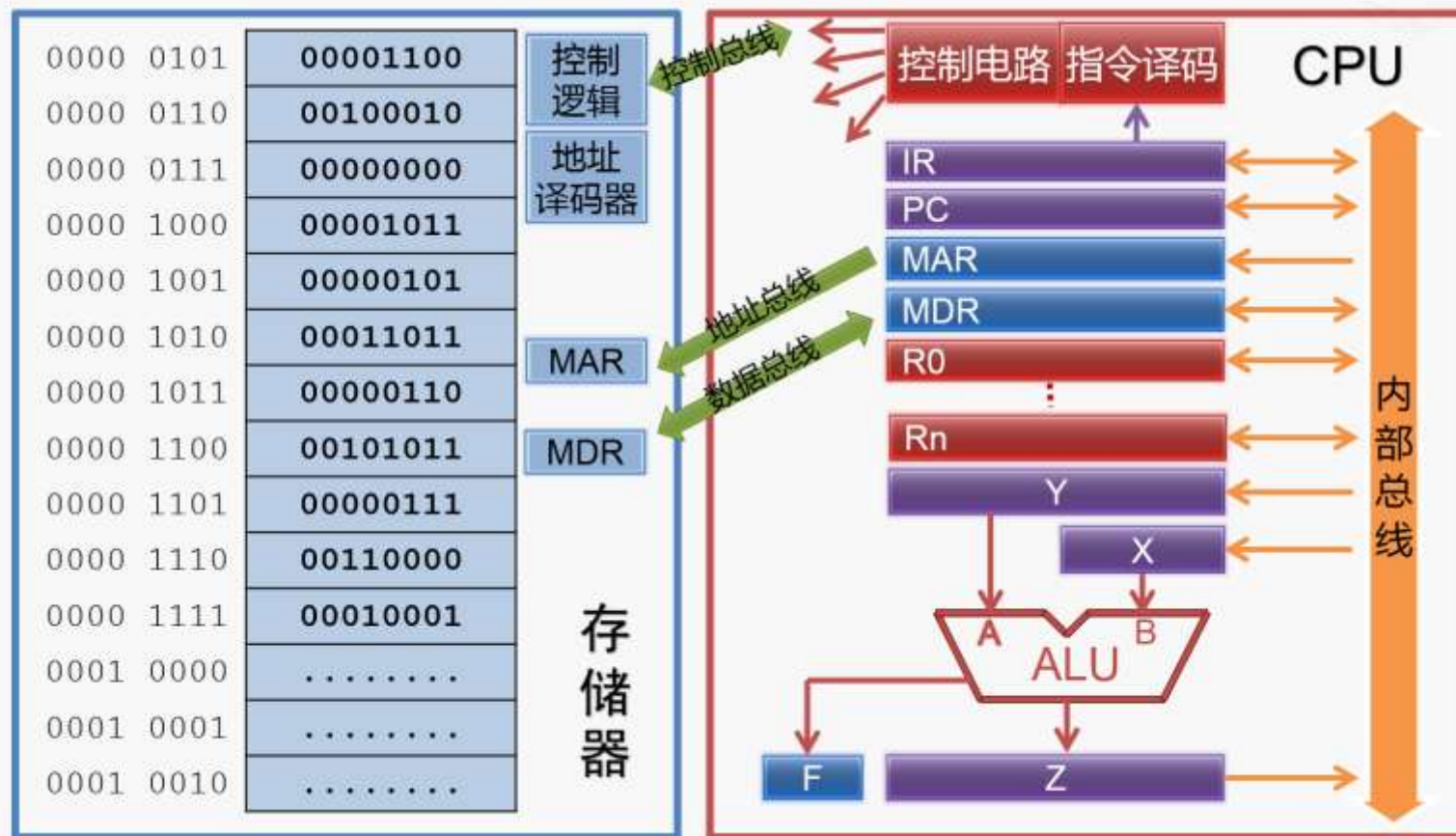
# MIPS指令的主要特点（4）

## 指令数量少，指令功能简单

。一条指令只完成一个操作，简化指令的执行过程

影响

- 处理器设计简单
- 处理器运行速度快
- 编程复杂
- 程序代码量大
- 需要优秀的编译器





# MIPS指令

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card. 2. Fold bottom side (columns 3 and 4) together.

## MIPS Reference Data

### CORE INSTRUCTION SET

NAME, Mnemonic	FOR	OPERATION (in Verilog)	OPCODE / FUNCTION
add	R	$R[d] \leftarrow R[s] + R[t]$	(1) 6/7/26
add immediate	imm	$R[d] \leftarrow R[s] + \text{immediate}$	(1,2) 6/26
add imm. Unmasked	imm	$R[d] \leftarrow R[s] + \text{immediate}$	(1,2) 6/26
add Unmasked	imm	$R[d] \leftarrow R[s] + R[t]$	6/7/26
and	R	$R[d] \leftarrow R[s] \& R[t]$	6/7/26
and immediate	imm	$R[d] \leftarrow R[s] \& \text{immediate}$	(3) 6/26
branch on equal	imm	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$	(4) 6/26
branch on not equal	imm	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$	(4) 6/26
jump	imm	$\text{PC} \leftarrow \text{jump\_addr}$	(5) 6/26
jump and link	imm	$\text{R[t]} \leftarrow \text{PC}; \text{PC} \leftarrow \text{jump\_addr}$	(5) 6/26
jump register	imm	$\text{PC} \leftarrow \text{R[s]}$	6/7/26
load byte Unmasked	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2) 26
load halfword Unmasked	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2) 26
load word	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2,7) 26
load upper half	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2) 26
load word	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2) 26
nor	R	$R[d] \leftarrow \sim (R[s] \& R[t])$	6/7/26
or	R	$R[d] \leftarrow R[s] \mid R[t]$	6/7/26
or immediate	imm	$R[d] \leftarrow R[s] \mid \text{immediate}$	(3) 6/26
set less than	imm	$R[d] \leftarrow (R[s] < \text{immediate}) ? 1 : 0$	6/7/26
set less than imm. Unmasked	imm	$R[d] \leftarrow (R[s] < \text{immediate}) ? 1 : 0$	(3,4) 6/26
set less than Unmasked	imm	$R[d] \leftarrow (R[s] < R[t]) ? 1 : 0$	(3,4) 6/26
shift left logical	imm	$R[d] \leftarrow R[s] \ll \text{immediate}$	6/7/26
shift right logical	imm	$R[d] \leftarrow R[s] \gg \text{immediate}$	6/7/26
store byte	imm	$\text{M}[R[s] + \text{immediate}] \leftarrow R[t] \ll 24$	(2) 26
store Conditional	imm	$\text{M}[R[s] + \text{immediate}] \leftarrow R[t] \ll 24$	(2,7) 26
store halfword	imm	$\text{M}[R[s] + \text{immediate}] \leftarrow R[t] \ll 16$	(2) 26
store word	imm	$\text{M}[R[s] + \text{immediate}] \leftarrow R[t]$	(2) 26
subtract	R	$R[d] \leftarrow R[s] - R[t]$	(3) 6/7/26
subtract Unmasked	imm	$R[d] \leftarrow R[s] - R[t]$	6/7/26

- May cause overflow exception
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)
- Signatures: (1) to (4) immediate (16), immediate (16), immediate (16), immediate (16)

### BASIC INSTRUCTION FORMATS



Copyright 2004 by Altera, Inc. All rights reserved. From Patterns and Formulas, Computer Organization and Design, 4th ed.

### ARITHMETIC CORE INSTRUCTION SET

NAME, Mnemonic	FOR	OPERATION	OPCODE / FUNCTION
branch on FF false	imm	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$	(4) 6/26
branch on FF true	imm	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$	(4) 6/26
double Unmasked	imm	$R[d] \leftarrow R[s] \times R[t]$	(8) 6/7/26
FF add single	imm	$R[d] \leftarrow R[s] + R[t]$	6/7/26
FF add	imm	$R[d] \leftarrow R[s] + R[t]$	6/7/26
FF compare single	imm	$\text{FF} \leftarrow (R[s] < R[t]) ? 1 : 0$	6/7/26
FF compare	imm	$\text{FF} \leftarrow (R[s] < R[t]) ? 1 : 0$	6/7/26
FF divide single	imm	$R[d] \leftarrow R[s] / R[t]$	6/7/26
FF divide	imm	$R[d] \leftarrow R[s] / R[t]$	6/7/26
FF multiply single	imm	$R[d] \leftarrow R[s] \times R[t]$	6/7/26
FF multiply	imm	$R[d] \leftarrow R[s] \times R[t]$	6/7/26
FF subtract single	imm	$R[d] \leftarrow R[s] - R[t]$	6/7/26
FF subtract	imm	$R[d] \leftarrow R[s] - R[t]$	6/7/26
Load FF Single	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2) 26
Load FF	imm	$R[d] \leftarrow \text{M}[R[s] + \text{immediate}]$	(2) 26
Move from Lo	imm	$R[d] \leftarrow R[Lo]$	6/7/26
Move from Hi	imm	$R[d] \leftarrow R[Hi]$	6/7/26
Move from Control	imm	$R[d] \leftarrow R[Control]$	6/7/26
Multiply	imm	$R[d] \leftarrow R[s] \times R[t]$	6/7/26
Multiply Unmasked	imm	$R[d] \leftarrow R[s] \times R[t]$	6/7/26
Shift Right Arith.	imm	$R[d] \leftarrow R[s] \ggg \text{immediate}$	6/7/26
Store FF single	imm	$\text{M}[R[s] + \text{immediate}] \leftarrow \text{FF}$	6/7/26
Store FF	imm	$\text{M}[R[s] + \text{immediate}] \leftarrow \text{FF}$	6/7/26

### FLOATING-POINT INSTRUCTION FORMATS



### PSEUDOINSTRUCTION SET

NAME	Mnemonic	OPERATION
branch less than	b.lt	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal	b.le	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal Unmasked	b.le.un	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal Unmasked	b.le.un	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal Unmasked	b.le.un	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal Unmasked	b.le.un	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal Unmasked	b.le.un	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$
branch less than or equal Unmasked	b.le.un	$\text{PC} \leftarrow \text{PC} + \text{branch\_addr}$

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

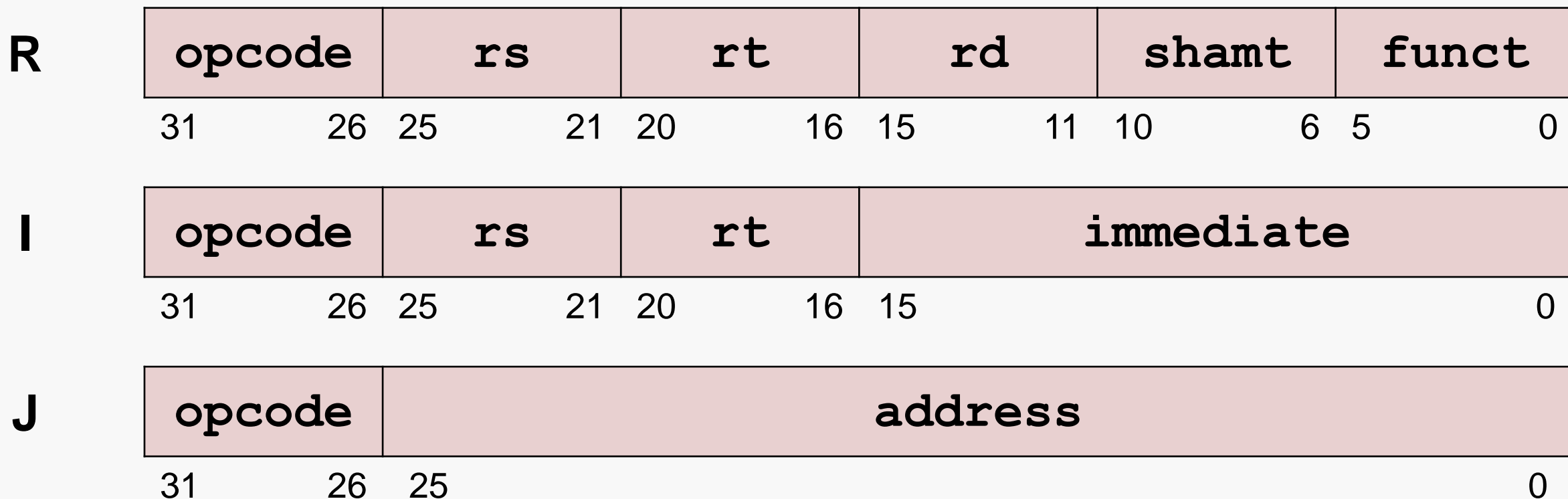
NAME	NUMBER	USE	PRESERVED ACROSS
\$zero	0	The constant value 0	Always
\$ra	31	Return address	Always
\$sp	29	Stack pointer	Always
\$fp	30	Frame pointer	Always
\$gp	31	Global pointer	Always
\$t0	2	Temporary	Not
\$t1	3	Temporary	Not
\$t2	4	Temporary	Not
\$t3	5	Temporary	Not
\$t4	6	Temporary	Not
\$t5	7	Temporary	Not
\$t6	8	Temporary	Not
\$t7	9	Temporary	Not
\$t8	10	Temporary	Not
\$t9	11	Temporary	Not
\$t10	12	Temporary	Not
\$t11	13	Temporary	Not
\$t12	14	Temporary	Not
\$t13	15	Temporary	Not
\$t14	16	Temporary	Not
\$t15	17	Temporary	Not
\$t16	18	Temporary	Not
\$t17	19	Temporary	Not
\$t18	20	Temporary	Not
\$t19	21	Temporary	Not
\$t20	22	Temporary	Not
\$t21	23	Temporary	Not
\$t22	24	Temporary	Not
\$t23	25	Temporary	Not
\$t24	26	Temporary	Not
\$t25	27	Temporary	Not
\$t26	28	Temporary	Not
\$t27	29	Temporary	Not
\$t28	30	Temporary	Not
\$t29	31	Temporary	Not

### OPCODES, BASE CONVERSION, ASCH SYMBOLS

OPCODE	BASE	CONVERSION	ASCH SYMBOLS
00000000	0	0	00000000
00000001	1	1	00000001
00000002	2	2	00000002
00000003	3	3	00000003
00000004	4	4	00000004
00000005	5	5	00000005
00000006	6	6	00000006
00000007	7	7	00000007
00000008	8	8	00000008
00000009	9	9	00000009
0000000A	A	A	0000000A
0000000B	B	B	0000000B
0000000C	C	C	0000000C
0000000D	D	D	0000000D
0000000E	E	E	0000000E
0000000F	F	F	0000000F
00000010	10	10	00000010
00000011	11	11	00000011
00000012	12	12	00000012
00000013	13	13	00000013
00000014	14	14	00000014
00000015	15	15	00000015
00000016	16	16	00000016
00000017	17	17	00000017
00000018	18	18	00000018
00000019	19	19	00000019
0000001A	1A	1A	0000001A
0000001B	1B	1B	0000001B
0000001C	1C	1C	0000001C
0000001D	1D	1D	0000001D
0000001E	1E	1E	0000001E
0000001F	1F	1F	0000001F
00000020	20	20	00000020
00000021	21	21	00000021
00000022	22	22	00000022
00000023	23	23	00000023
00000024	24	24	00000024
00000025	25	25	00000025
00000026	26	26	00000026
00000027	27	27	00000027
00000028	28	28	00000028
00000029	29	29	00000029
0000002A	2A	2A	0000002A
0000002B	2B	2B	0000002B
0000002C	2C	2C	0000002C
0000002D	2D	2D	0000002D
0000002E	2E	2E	0000002E
0000002F	2F	2F	0000002F
00000030	30	30	00000030
00000031	31	31	00000031
00000032	32	32	00000032
00000033	33	33	00000033
00000034	34	34	00000034
00000035	35	35	00000035
00000036	36	36	00000036
00000037	37	37	00000037
00000038	38	38	00000038
00000039	39	39	00000039
0000003A	3A	3A	0000003A
0000003B	3B	3B	0000003B
0000003C	3C	3C	0000003C
0000003D	3D	3D	0000003D
0000003E	3E	3E	0000003E
0000003F	3F	3F	0000003F
00000040	40	40	00000040
00000041	41	41	00000041
00000042	42	42	00000042
00000043	43	43	00000043
00000044	44	44	00000044
00000045	45	45	00000045
00000046	46	46	00000046
00000047	47	47	00000047
00000048	48	48	00000048
00000049	49	49	00000049
0000004A	4A	4A	0000004A
0000004B	4B	4B	0000004B
0000004C	4C	4C	0000004C
0000004D	4D	4D	0000004D
0000004E	4E	4E	0000004E
0000004F	4F	4F	0000004F
00000050	50	50	00000050
00000051	51	51	00000051
00000052	52	52	00000052
00000053	53	53	00000053
00000054	54	54	00000054
00000055	55	55	00000055
00000056	56	56	00000056
00000057	57	57	00000057
00000058	58	58	00000058
00000059	59	59	00000059
0000005A	5A	5A	0000005A
0000005B	5B	5B	0000005B
0000005C	5C	5C	0000005C
0000005D	5D	5D	0000005D
0000005E	5E	5E	0000005E
0000005F	5F	5F	0000005F
00000060	60	60	00000060
00000061	61	61	00000061
00000062	62	62	00000062
00000063	63	63	00000063
00000064	64	64	00000064
00000065	65	65	00000065
00000066	66	66	00000066
00000067	67	67	00000067
00000068	68	68	00000068
00000069	69	69	00000069
0000006A	6A	6A	0000006A
0000006B	6B	6B	0000006B
0000006C	6C	6C	0000006C
0000006D	6D	6D	0000006D
0000006E	6E	6E	0000006E
0000006F	6F	6F	0000006F
00000070	70	70	00000070
00000071	71	71	00000071
00000072	72	72	00000072
00000073	73	73	00000073
00000074	74	74	00000074
00000075	75	75	00000075
00000076	76	76	00000076
00000077	77	77	00000077
00000078	78	78	00000078
00000079	79	79	00000079
0000007A	7A	7A	0000007A
0000007B	7B	7B	0000007B
0000007C	7C	7C	0000007C
0000007D	7D	7D	0000007D
0000007E	7E	7E	0000007E
0000007F	7F	7F	0000007F
00000080	80	80	00000080
00000081	81	81	00000081
00000082	82	82	00000082
00000083	83	83	00000083
00000084	84	84	00000084
00000085	85	85	00000085
00000086	86	86	00000086
00000087	87	87	00000087
00000088	88	88	00000088
00000089	89	89	00000089
0000008A	8A	8A	0000008A
0000008B	8B	8B	0000008B
0000008C	8C	8C	0000008C
0000008D	8D	8D	0000008D

# MIPS指令的基本格式

- ▶ R: Register, 寄存器
- ▶ I: Immediate, 立即数
- ▶ J: Jump, 无条件转移





# 不同维度的指令分类（示例）

运算 指令	<code>add rd,rs,rt</code> <code>sll rd,rt,shamt</code>	<code>addi rt,rs,imm</code> <code>slti rt,rs,imm</code>	/
访存 指令	/	<code>lw rt,imm(rs)</code> <code>sw rt,imm(rs)</code>	/
分支 指令	<code>jr rs</code>	<code>beq rs,rt,imm</code>	<code>j addr</code>
	R型指令	I型指令	J型指令

# 主要内容

通过学习本课程  
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I RISC的发展变迁

II MIPS指令的主要特点

III MIPS指令分类说明：R型

IV MIPS指令分类说明：I型



# 不同维度的指令分类（示例）

运算 指令	<div>add rd,rs,rt sll rd,rt,shamt</div>	<div>addi rt,rs,imm slti rt,rs,imm</div>	/
访存 指令	/	<div>lw rt,imm(rs) sw rt,imm(rs)</div>	/
分支 指令	jr rs	beq rs,rt,imm	j addr
	R型指令	I型指令	J型指令





# R型指令的格式（1）

- ▶ R型指令格式包含6个域
  - 2个6-bit域，可表示0~63的数
  - 4个5-bit域，可表示0~31的数

用于指定指令的类型。对于所有R型指令，该域的值均为0

**opcode**

6-bit

5-bit

5-bit

与opcode域组合，精确地指定指令的类型

**funct**

5-bit

5-bit

6-bit

**R**

**opcode**

**rs**

**rt**

**rd**

**shamt**

**funct**

31

26

25

21

20

16

15

11

10

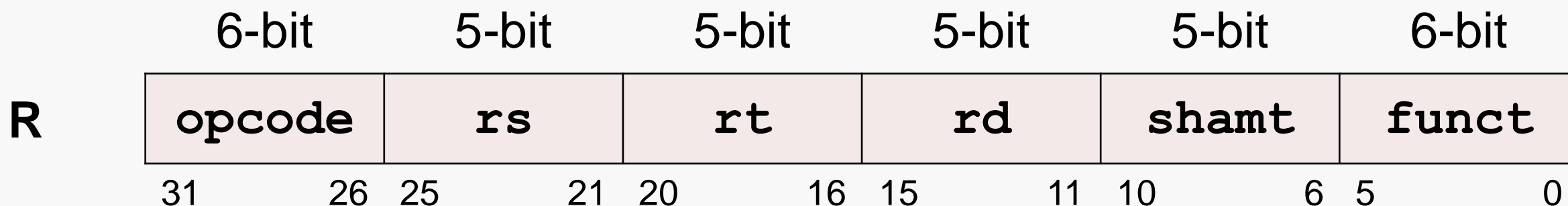
6

5

0

## R型指令的格式（2）

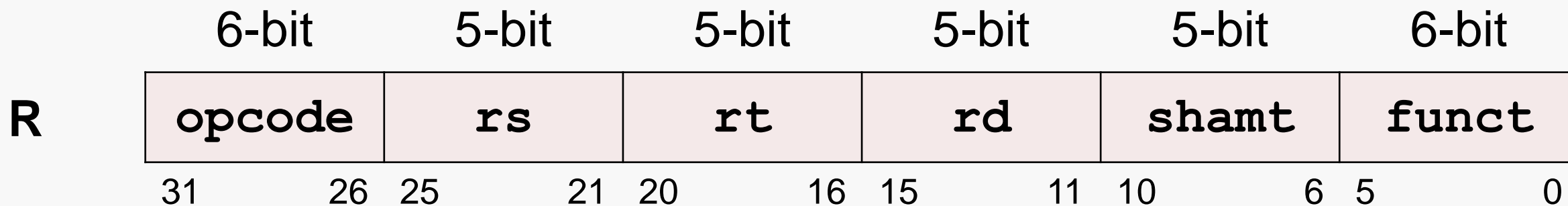
- 🕒 **rs Source Register**
  - 通常用于指定第一个源操作数所在的寄存器编号
- 🕒 **rt Target Register**
  - 通常用于指定第二个源操作数所在的寄存器编号
- 🕒 **rd Destination Register**
  - 通常用于指定目的操作数（保存运算结果）的寄存器编号
- 🕒 **5-bit的域可表示0~31，对应32个通用寄存器**



# R型指令的格式 (3)

shamt **shift amount**

- 用于指定移位指令进行移位操作的位数
- 5-bit的域可表示0~31，对于32-bit数，更多移位没有实际意义
- 对于非移位指令，该域设为0





# R型指令的编码示例（1）

▶ **add \$8,\$9,\$10**    #  $R[rd] = R[rs] + R[rt]$

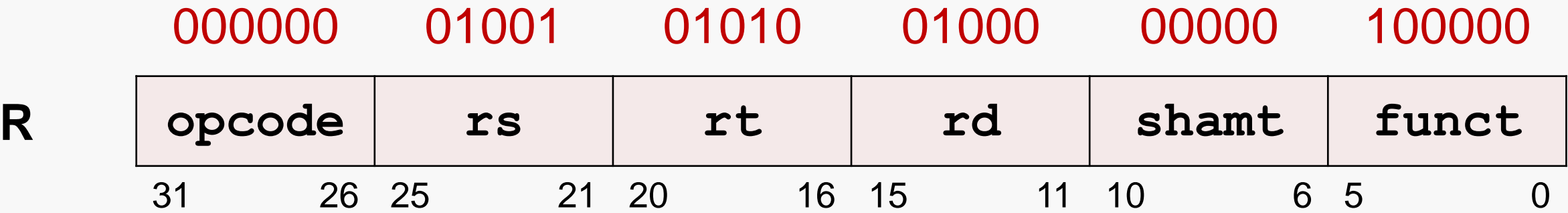
◦ 查指令编码表得到：

opcode = 0 , funct = 32, shamt = 0 （非移位指令）

◦ 根据指令操作数得到：

rd = 8 （目的操作数） , rs = 9 （第一个源操作数）

rt = 10 （第二个源操作数）





# R型指令的编码示例（2）

🎯 **sll \$8,\$9,10** # R[rd]=R[rt]<<shamt

◦ 查指令编码表得到：

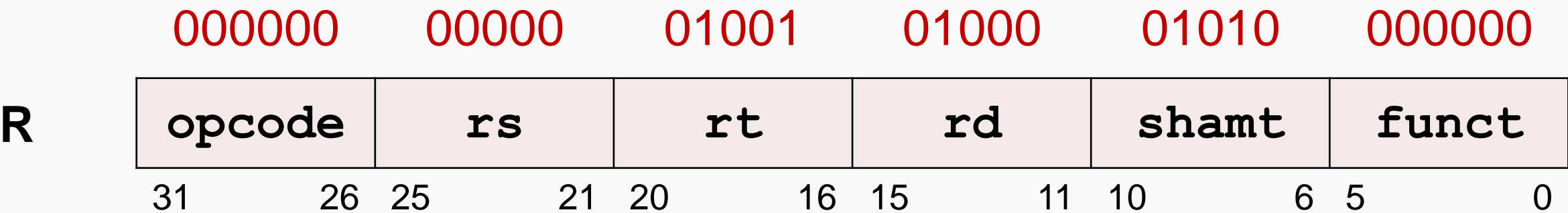
opcode = 0 , funct = 0, rs = 0 （未使用的寄存器）

◦ 根据指令操作数得到：

rd = 8 （目的操作数）

rt = 9 （源操作数）

shamt = 10 （移位数）





# 主要内容

通过学习本课程  
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

I RISC的发展变迁

II MIPS指令的主要特点

III MIPS指令分类说明：R型



IV MIPS指令分类说明：I型

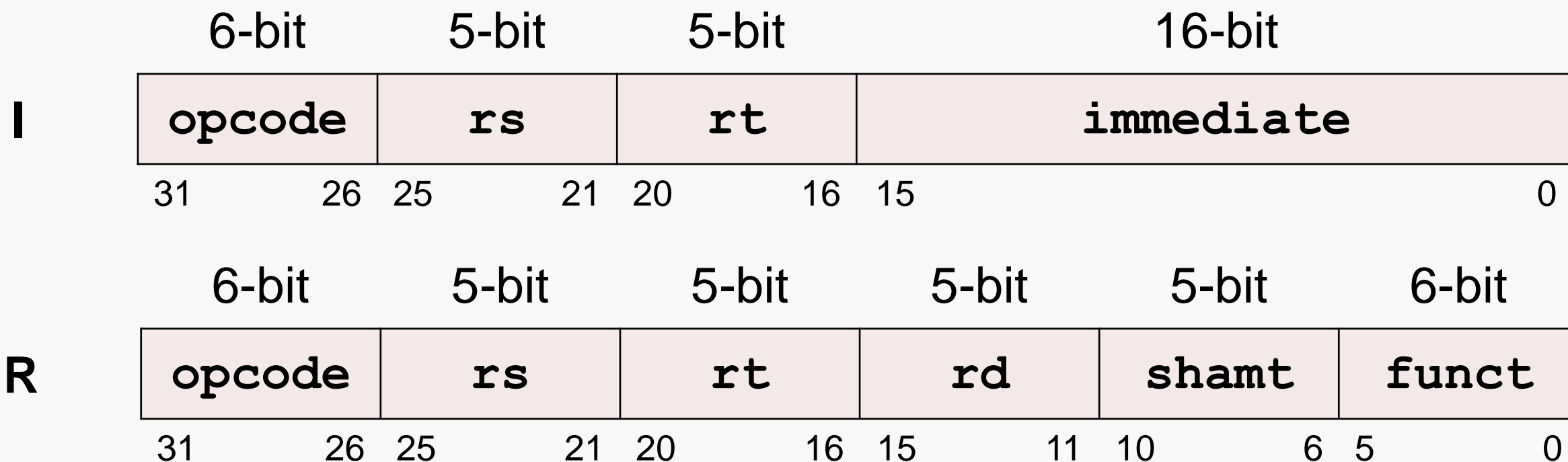


# 不同维度的指令分类（示例）

运算 指令	add rd,rs,rt sll rd,rt,shamt	addi rt,rs,imm slti rt,rs,imm	/
访存 指令	/	lw rt,imm(rs) sw rt,imm(rs)	/
分支 指令	jr rs	beq rs,rt,imm	j addr
	R型指令	I型指令	J型指令

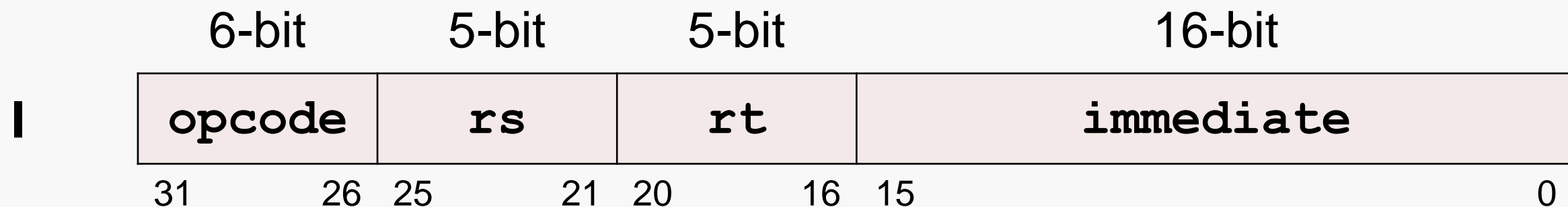
# I型指令的格式 (1)

- ▶ R型指令只有一个5-bit域表示立即数，范围为0~31
- ▶ 常用的立即数远大于这个范围，因此需要新的指令格式
- ▶ I型指令的大部分域与R型指令相同



# I型指令的格式（2）

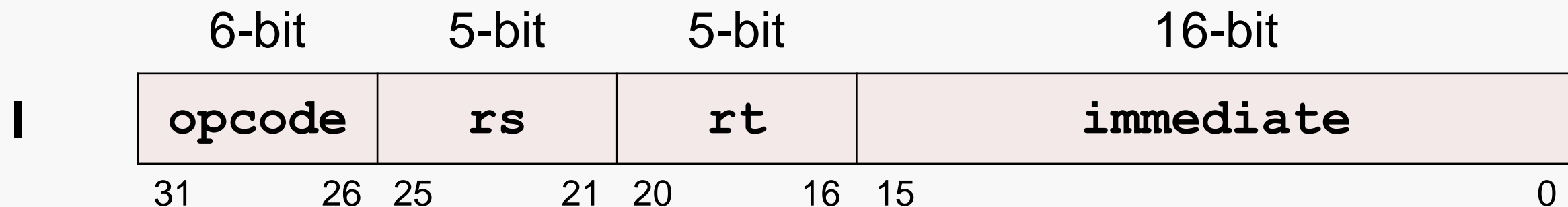
- ▶ opcode
  - 用于指定指令的操作类型（但没有`funct`域）
- ▶ `rs` Source Register
  - 指定第一个源操作数所在的寄存器编号
- ▶ `rt` Target Register
  - 指定用于目的操作数（保存运算结果）的寄存器编号
  - 对于某些指令，指定第二个源操作数所在的寄存器编号



# I型指令的格式 (3)

## ④ immediate

- 16-bit的立即数，可以表示 $2^{16}$ 个不同数值
- 对于访存指令，如`lw rt, imm(rs)`  
通常可以满足访存地址偏移量的需求 (-32768~+32767)
- 对于运算指令，如`addi rt, rs, imm`  
无法满足全部需求，但大多数时候可以满足需求







# I型指令的编码示例（1）

▶ **addi \$21,\$22,-50** #  $\$21 = \$22 + (-50)$

◦ 查指令编码表得到：

opcode = 8

◦ 分析指令得到：

rs = 22 （源操作数寄存器编号）

rt = 21 （目的操作数寄存器编号）

immediate = -50 （立即数）

001000      10110      10101      1111 1111 1100 1110

I

opcode		rs		rt		immediate					
31	26	25	21	20	16	15					0

## I型指令的编码示例 (2)

🔍 **lw \$21, -50(\$22)**    # \$21 = Mem[ \$22 + (-50) ]

◦ 查指令编码表得到:

opcode = 35

◦ 分析指令得到:

rs = 22 (源操作数寄存器编号)

rt = 21 (目的操作数寄存器编号)

immediate = -50 (立即数)

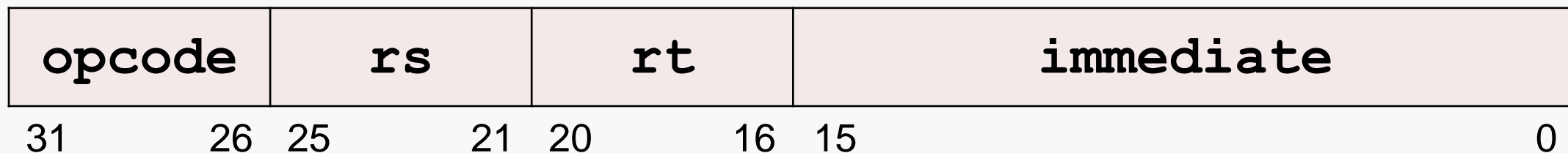
100011

10110

10101

1111 1111 1100 1110

I





# I型指令的编码示例（3）

🔍 **slti \$21,\$22,-50** #  $\$21 = (\$22 < (-50)) ? 1 : 0$

◦ 查指令编码表得到：

opcode = 10

◦ 分析指令得到：

rs = 22 （源操作数寄存器编号）

rt = 21 （目的操作数寄存器编号）

immediate = -50 （立即数）

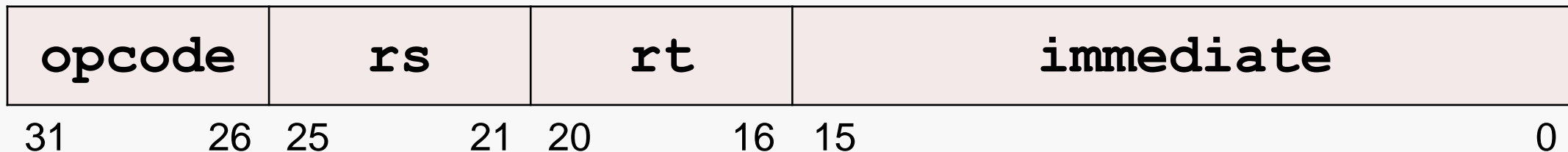
001010

10110

10101

1111 1111 1100 1110

I





# 本讲到此结束，谢谢 欢迎继续学习本课程

计算机组织与体系结构 Computer Architectures  
主讲：陆俊林