计算机组织与体系结构

Computer Architectures

陆俊林

2024年春季学期

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。

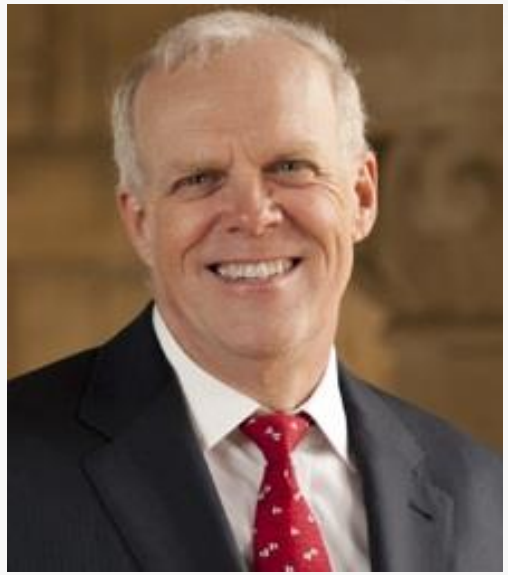
阅读教材“COD”：第2章，附录E

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

2017年图灵奖获得者

戴维·帕特森
David Patterson
1947年出生

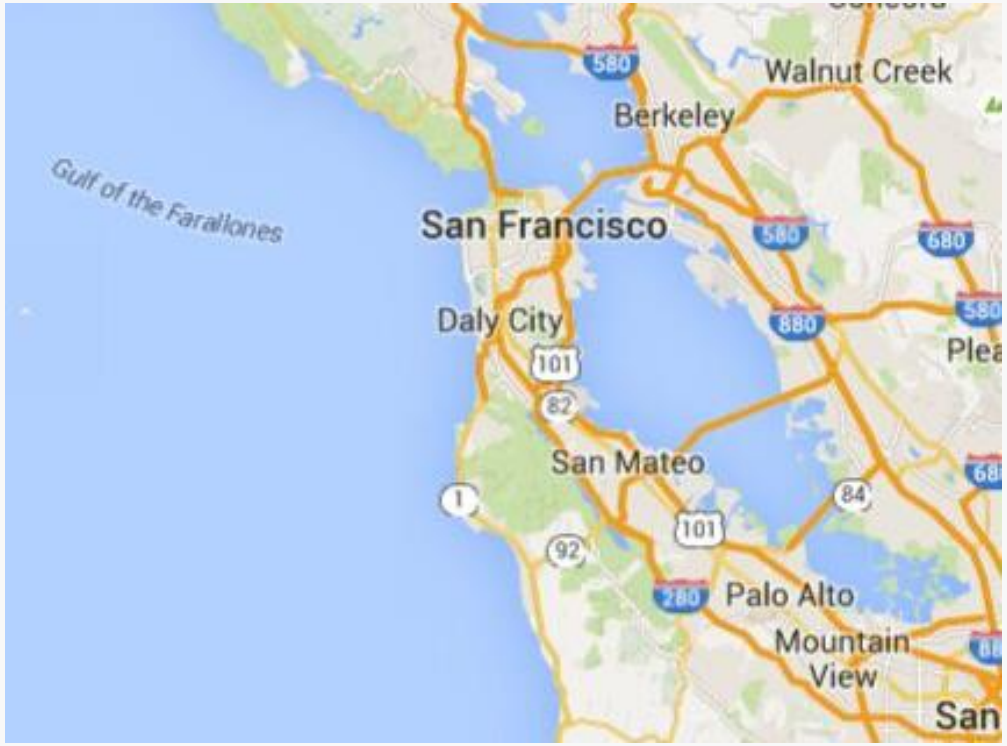


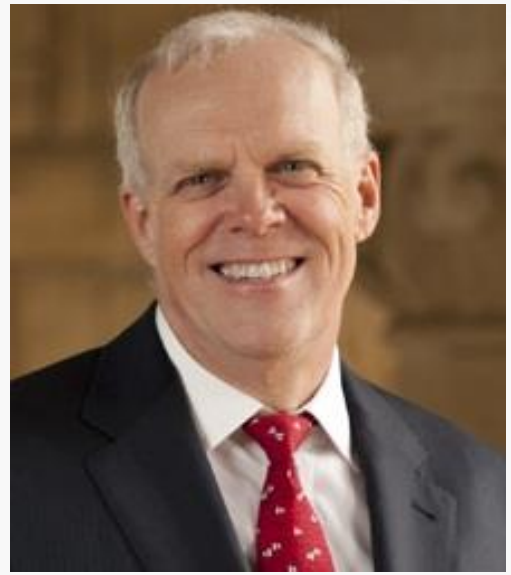
约翰·亨尼西
John Hennessy
1953年出生

For pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry.

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

MIPS公司的商业兴衰

- 1984年，MIPS计算机系统公司成立
- 1988年，SGI公司在其计算机产品中采用MIPS处理器
- 1989年，MIPS第一次上市
- 1992年，SGI收购MIPS，更名为MIPS技术公司
- 1998年，MIPS再次上市
- 2012年，Imagination Technologies收购MIPS

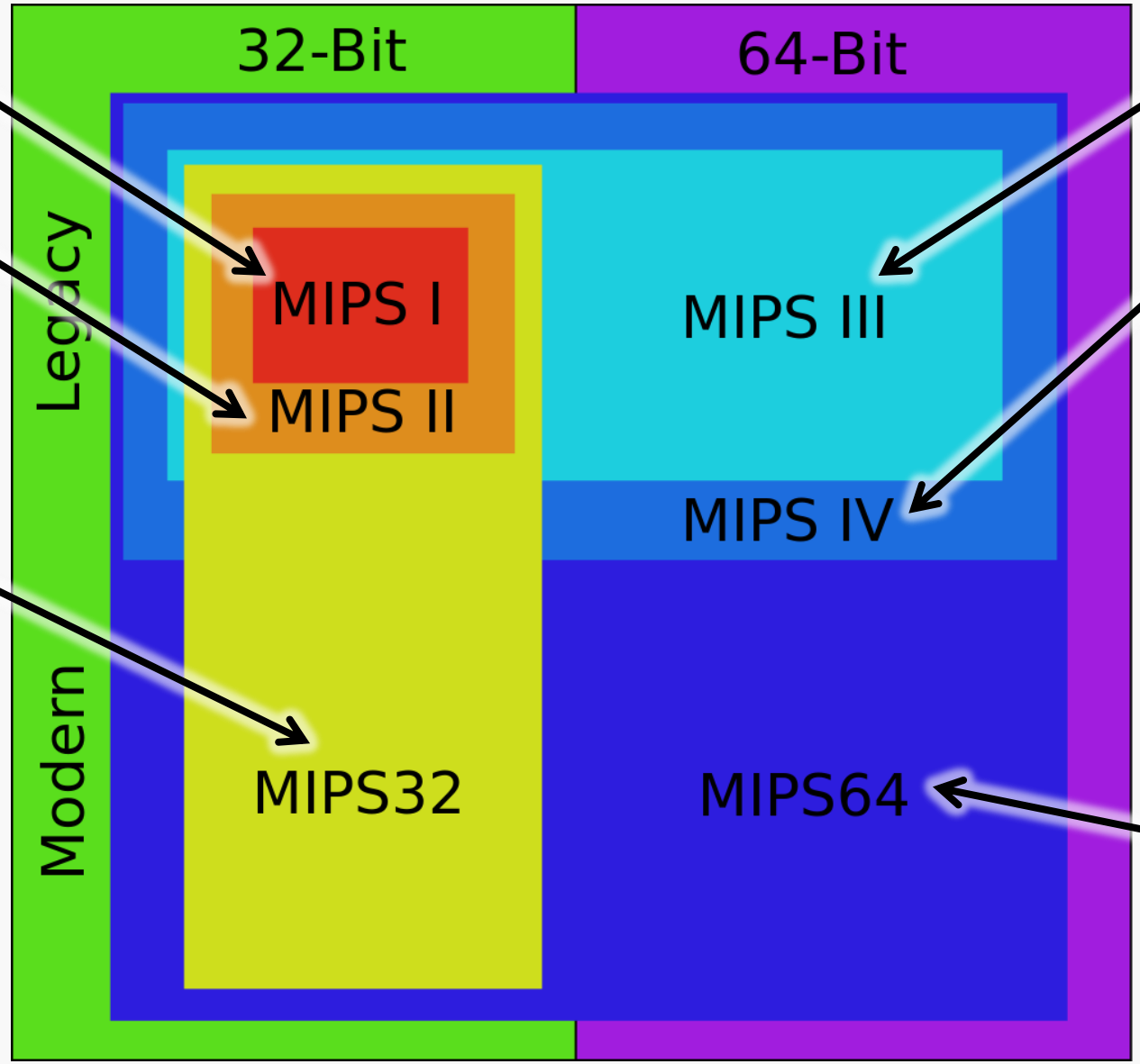
- MIPS处理器广泛应用的领域：
  - 数字电视、机顶盒、蓝光播放器、游戏机、网络设备等

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

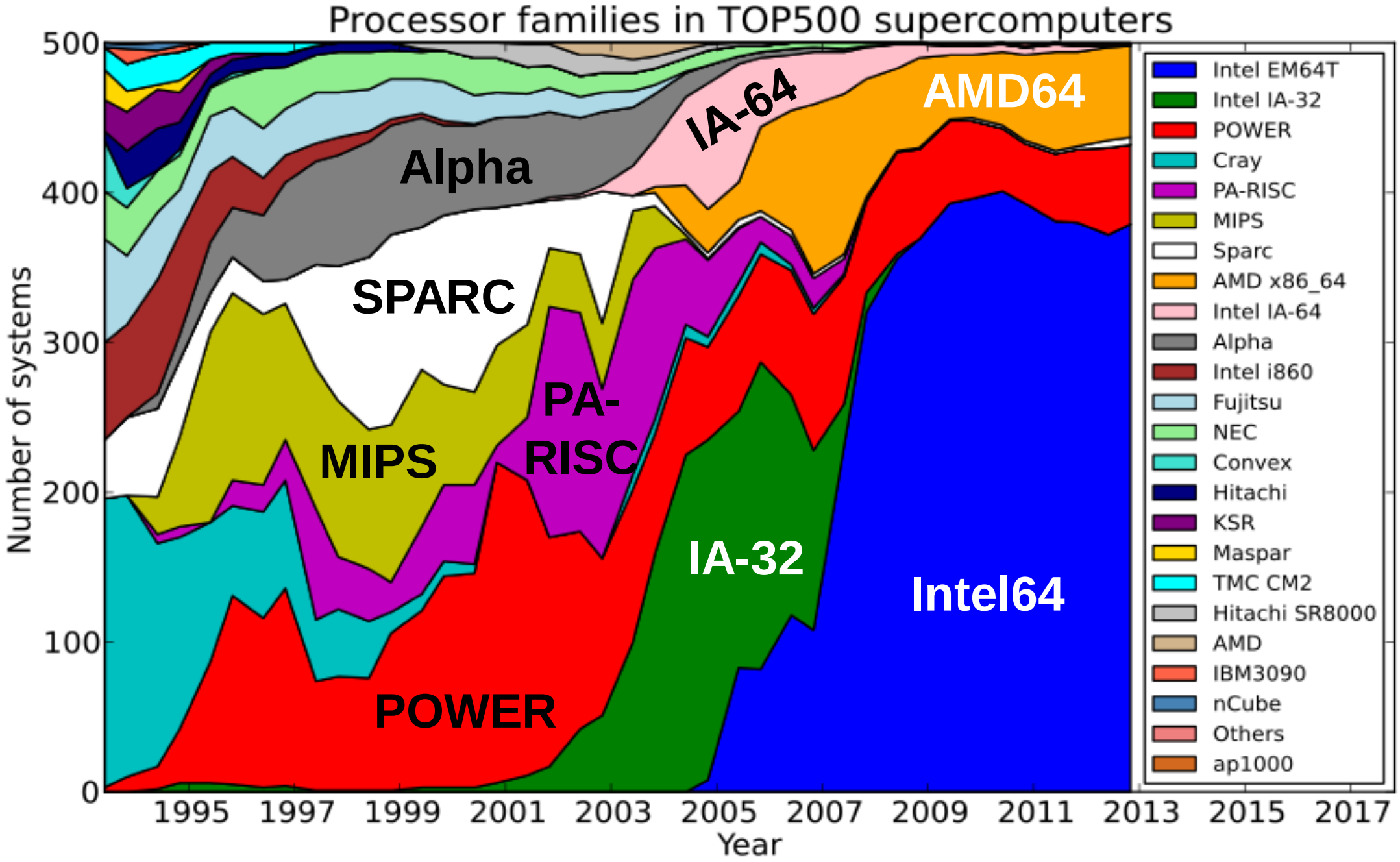
1994年，R8000

MIPS V
1996年

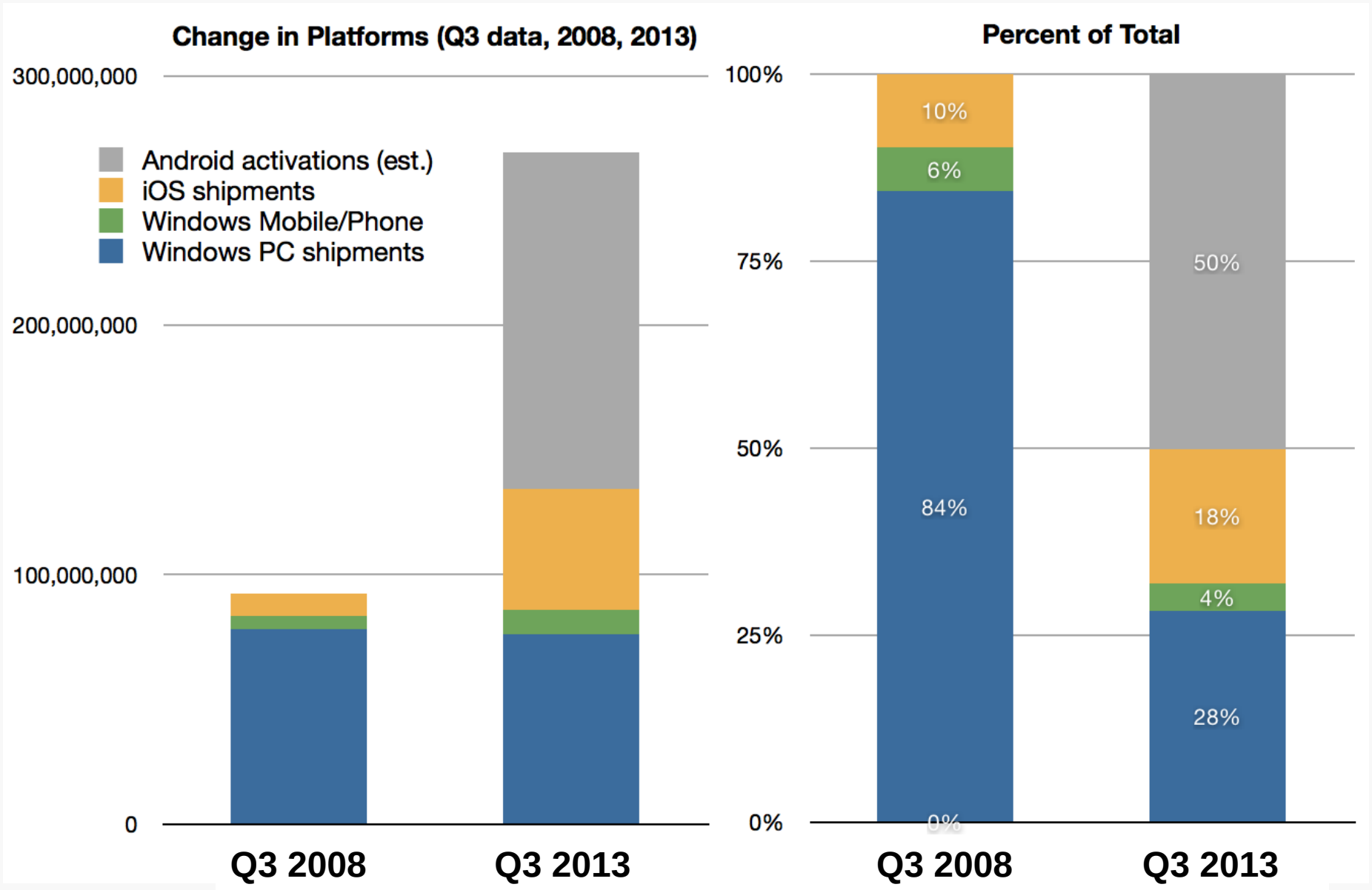
1999年
以MIPS V为基础

超级计算领域

CISC vs. RISC



个人计算领域 CISC vs. RISC



Android activations (est.)

iOS shipments

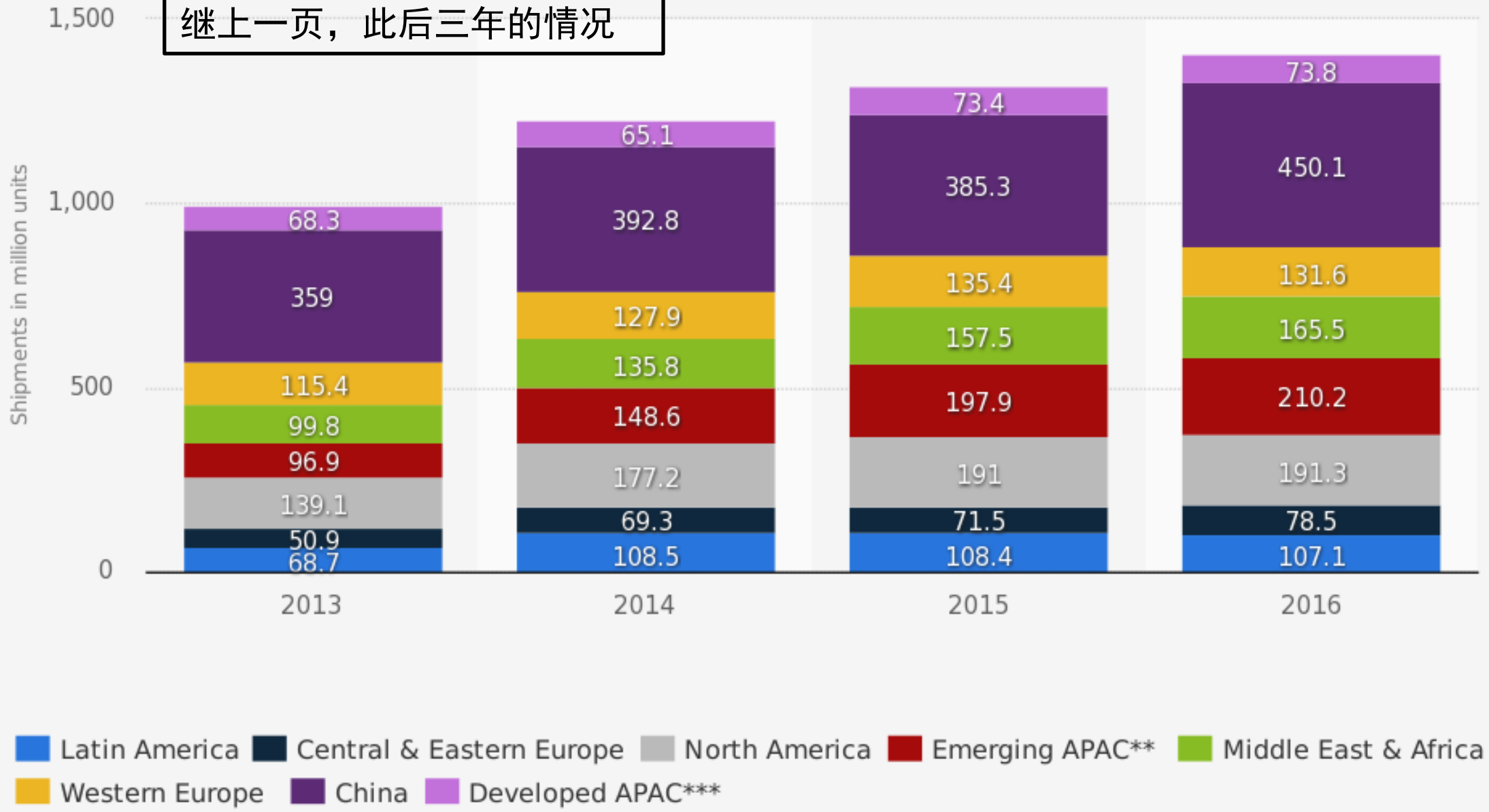
Windows Mobile/Phone

Windows PC shipments

2007 年， iPhone 推出。此后一年到五年的变化情况

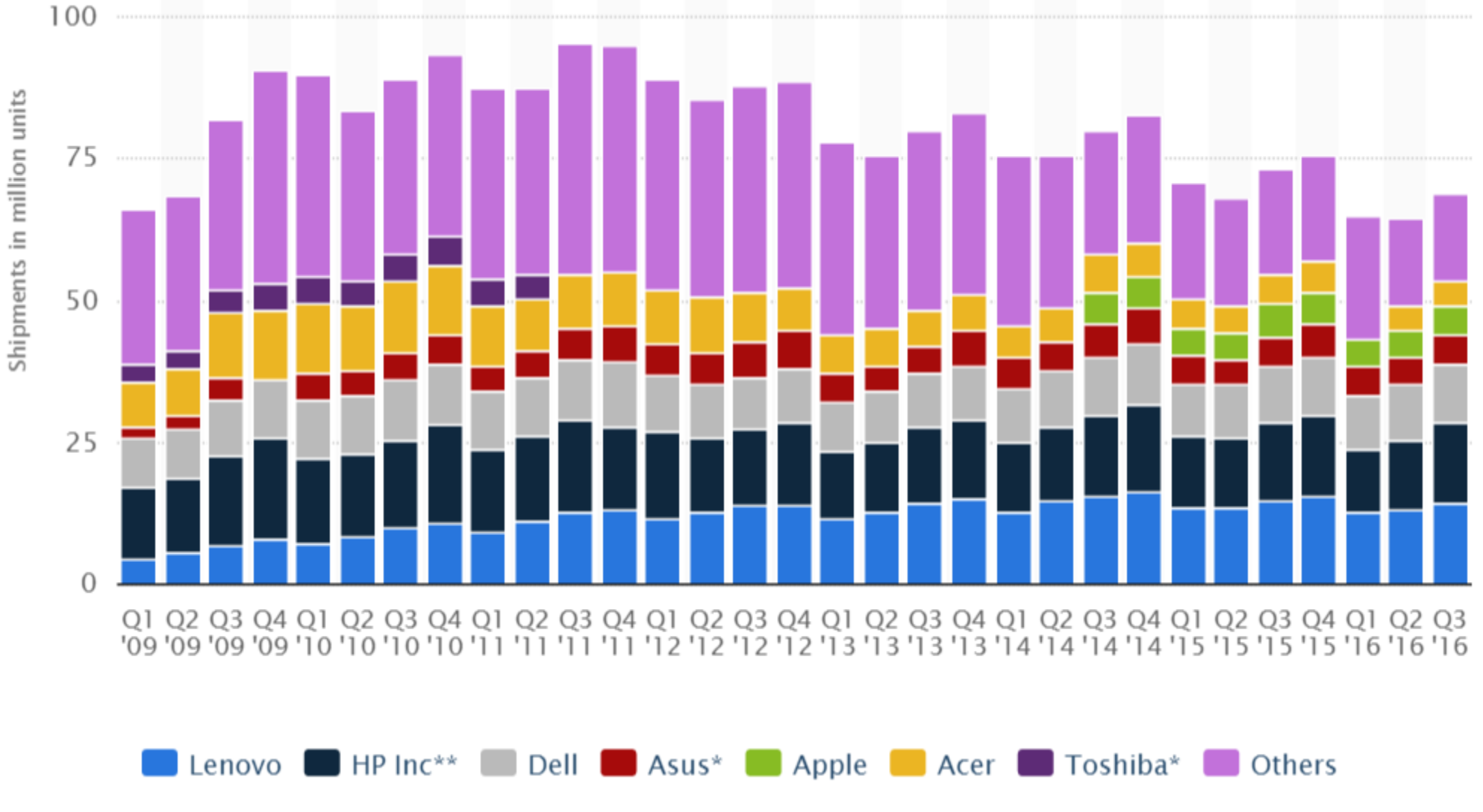
智能手机的市场出货量（2013-2016）

继上一页，此后三年的情况



个人计算机的市场出货量（2009-2016）

与上一页同时段的情况



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

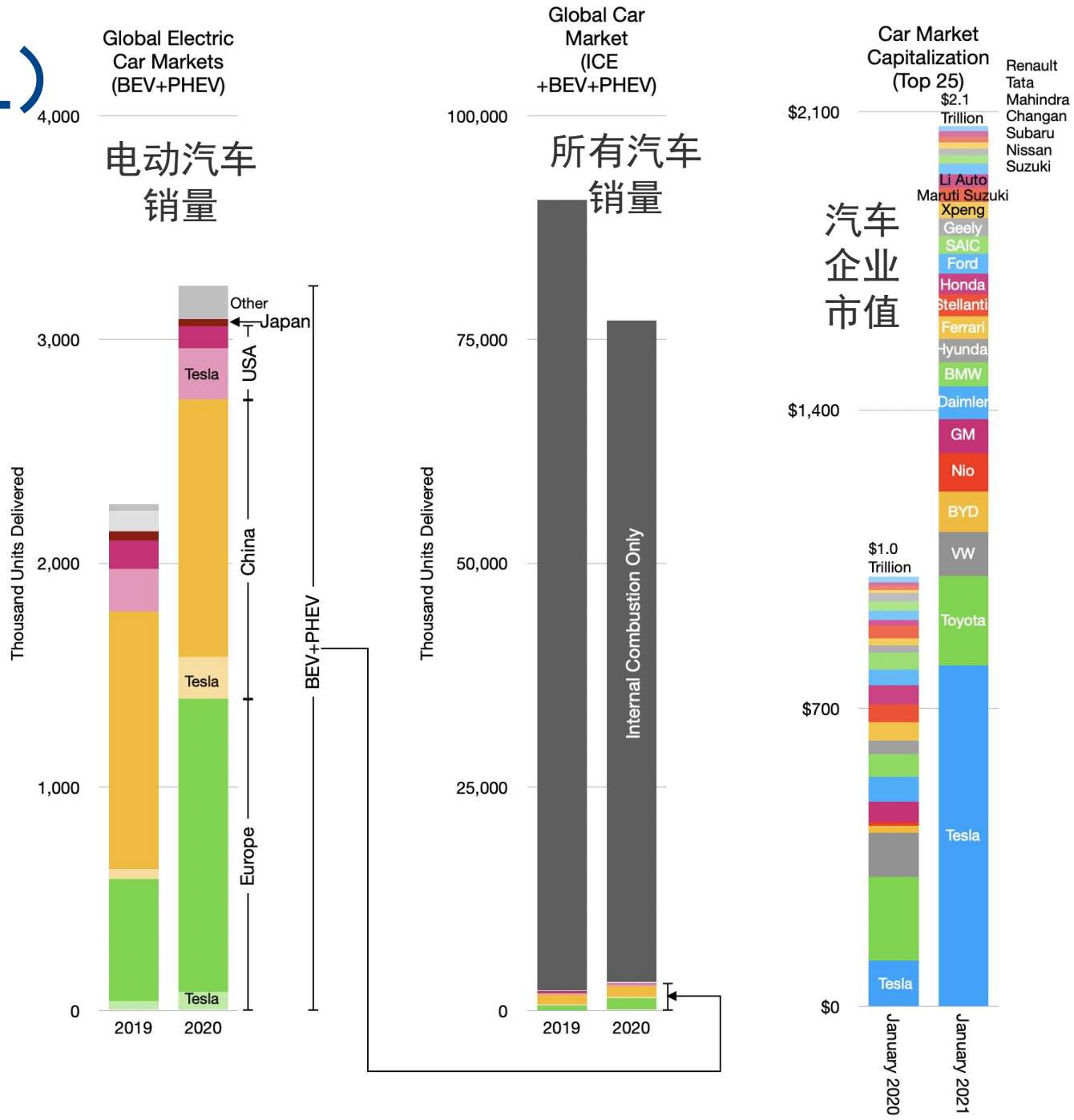
个人计算机的市场出货量（2017~2018）

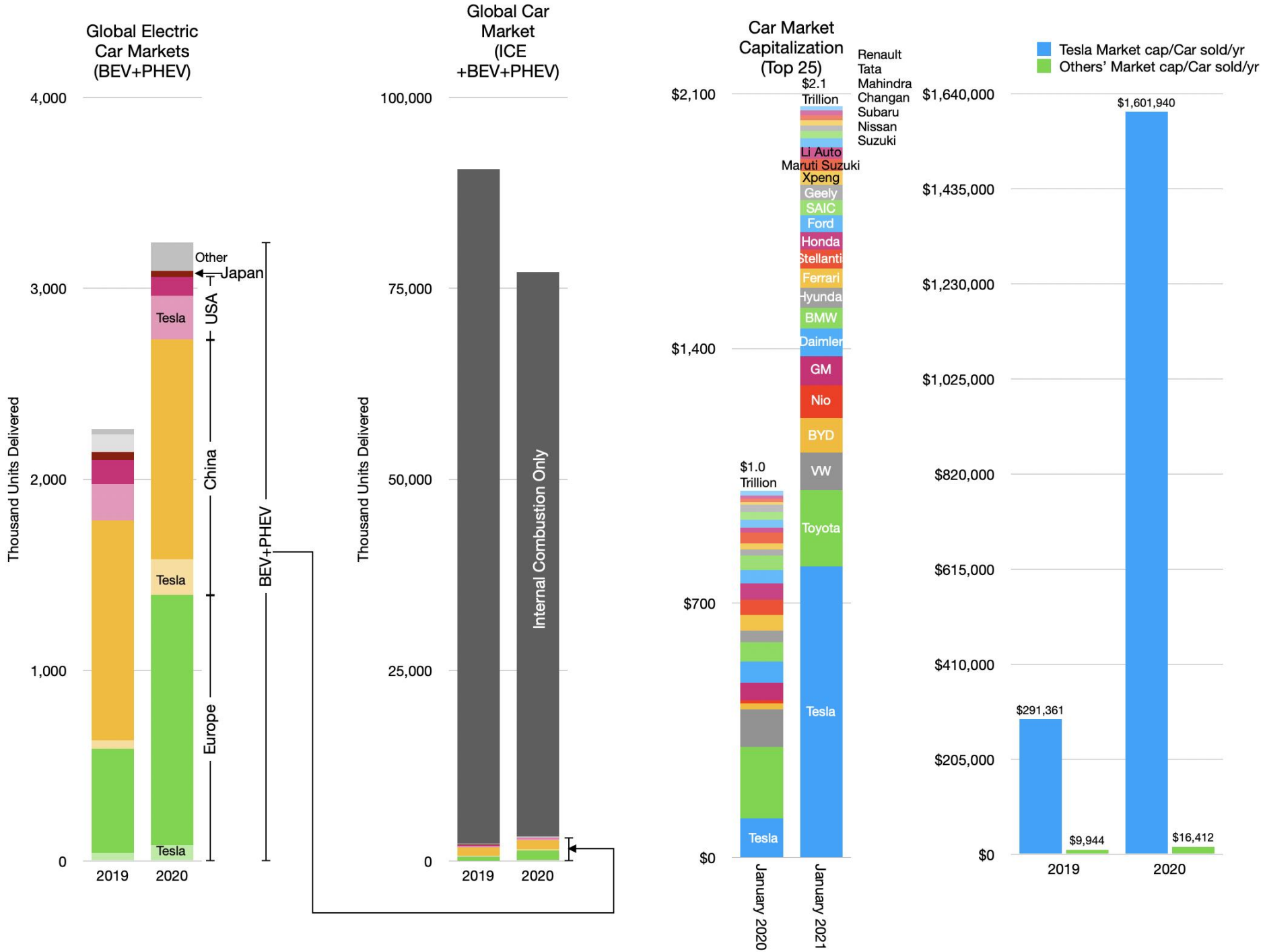
继上一页，之后两年的发展变化

Company	4Q18 Shipments	4Q18 Market Share (%)	4Q17 Shipments	4Q17 Market Share (%)	4Q18-4Q17 Growth (%)
Lenovo	16,628	24.2	15,697	21.9	5.9
HP Inc.	15,380	22.4	16,092	22.4	-4.4
Dell	10,915	15.9	10,763	15	1.4
Apple	4,920	7.2	5,112	7.1	-3.8
ASUS	4,211	6.1	4,716	6.6	-10.7
Acer Group	3,861	5.6	4,726	6.6	-18.3
Others	12,710	18.5	14,590	20.3	-12.9
Total	68,626	100.0	71,696	100.0	-4.3

新能源汽车（2019-2021）

- BEV，纯电动
  - BaiBattery Electrical Vehicle
- HEV，混合动力
  - Hybrid Electric Vehicle
- PHEV，插电式混合动力
  - Plug in Hybrid Electric Vehicle
- ICE，内燃机
  - Internal Combustion Engine




Global Electric Car Markets (BEV+PHEV)
Thousand Units Delivered
4,000
3,000
2,000
1,000
0
2019
2020
Other
Japan
USA
China
Europe
Tesla
Tesla
Tesla
BEV+PHEV
Global Car Market (ICE +BEV+PHEV)
Thousand Units Delivered
100,000
75,000
50,000
25,000
0
2019
2020
Internal Combustion Only
Car Market Capitalization (Top 25)
$2,100
$1,400
$700
$0
$1.0 Trillion
$2.1 Trillion
Renault
Tata
Mahindra
Changan
Subaru
Nissan
Suzuki
Li Auto
Maruti Suzuki
Xpeng
Geely
SAIC
Ford
Honda
Stellanti
Ferrari
Hyunda
BMW
Daimler
GM
Nio
BYD
VW
Toyota
Tesla
Tesla
January 2020
January 2021
Tesla Market cap/Car sold/yr
Others' Market cap/Car sold/yr
$1,640,000
$1,435,000
$1,230,000
$1,025,000
$820,000
$615,000
$410,000
$205,000
$0
$1,601,940
$291,361
$9,944
$16,412
2019
2020

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

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

MIPS指令示例

- 假设变量和寄存器的对应关系如下

```
f → $s0        g → $s1    h → $s2
i → $s3        j → $s4
```

f = (g + h) – (i + j)

```
add $t1, $s3, $s4
add $t2, $s1, $s2
sub $s0, $t2, $t1
```

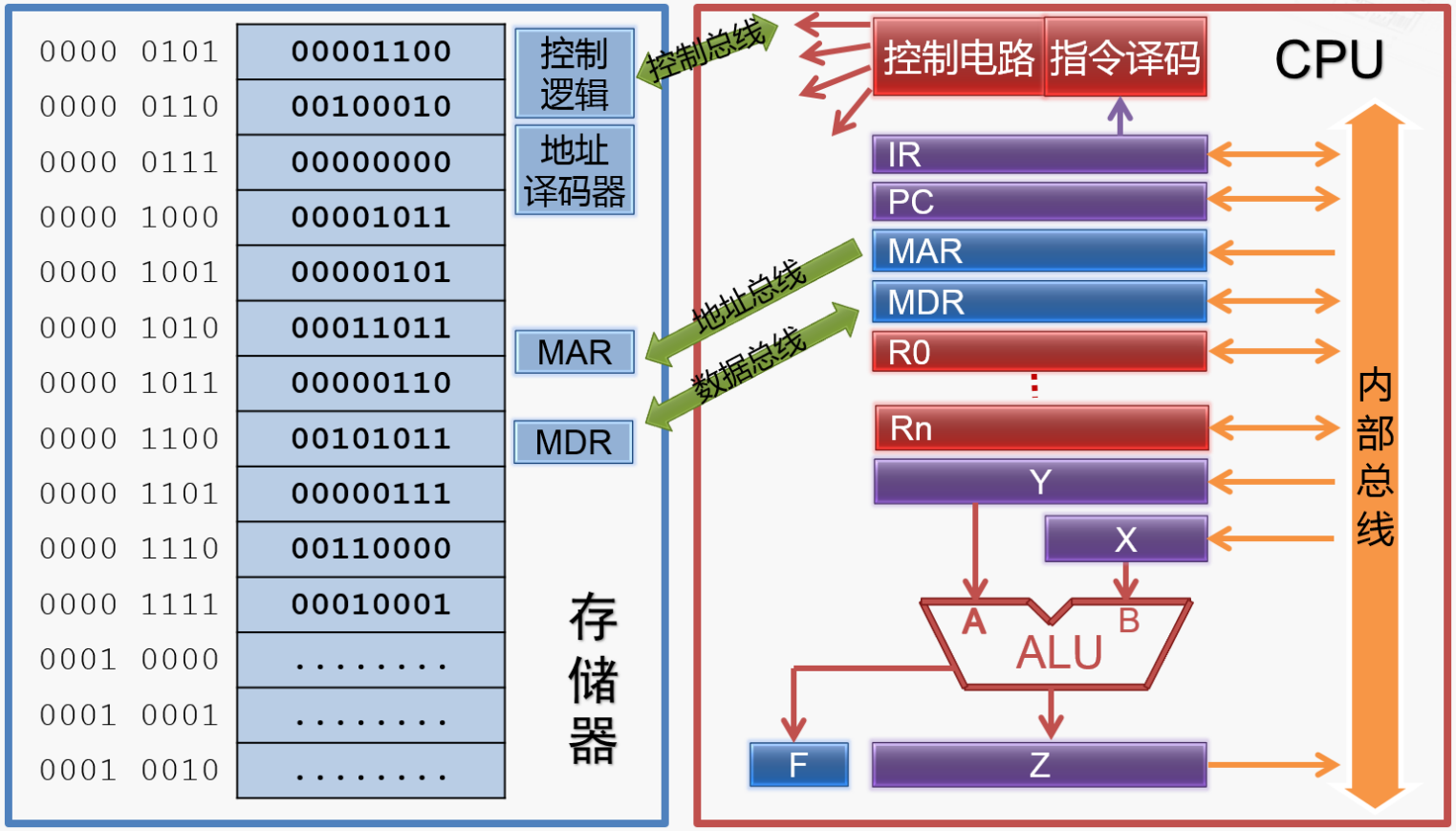
MIPS指令的主要特点（1）

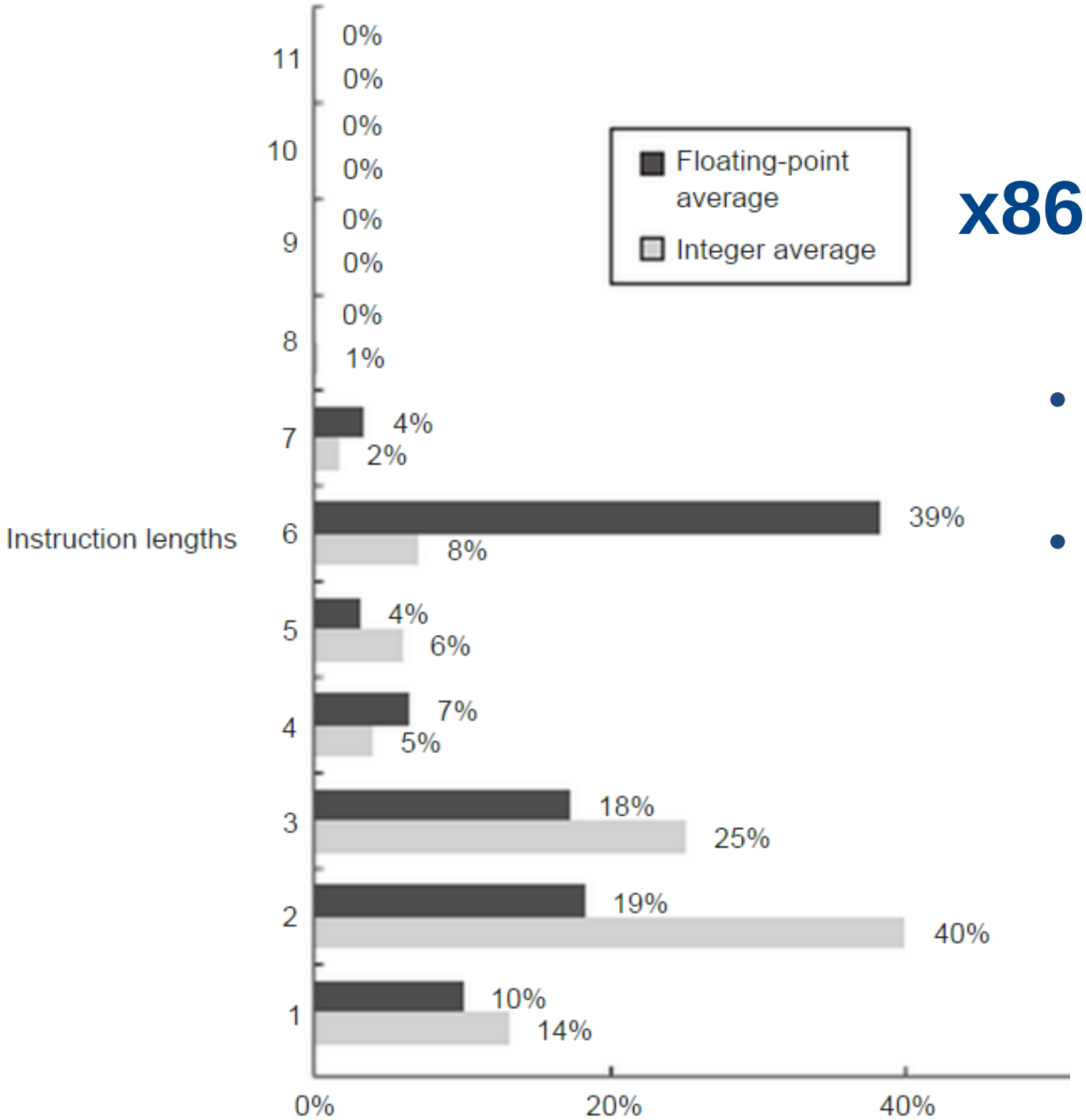
- 固定的指令长度（32-bit，即1 word）
  - 简化了从存储器取指令

x86指令

- 长度不确定
- 最短1个字节
- 长可达15个字节

模型机的CPU和存储器





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

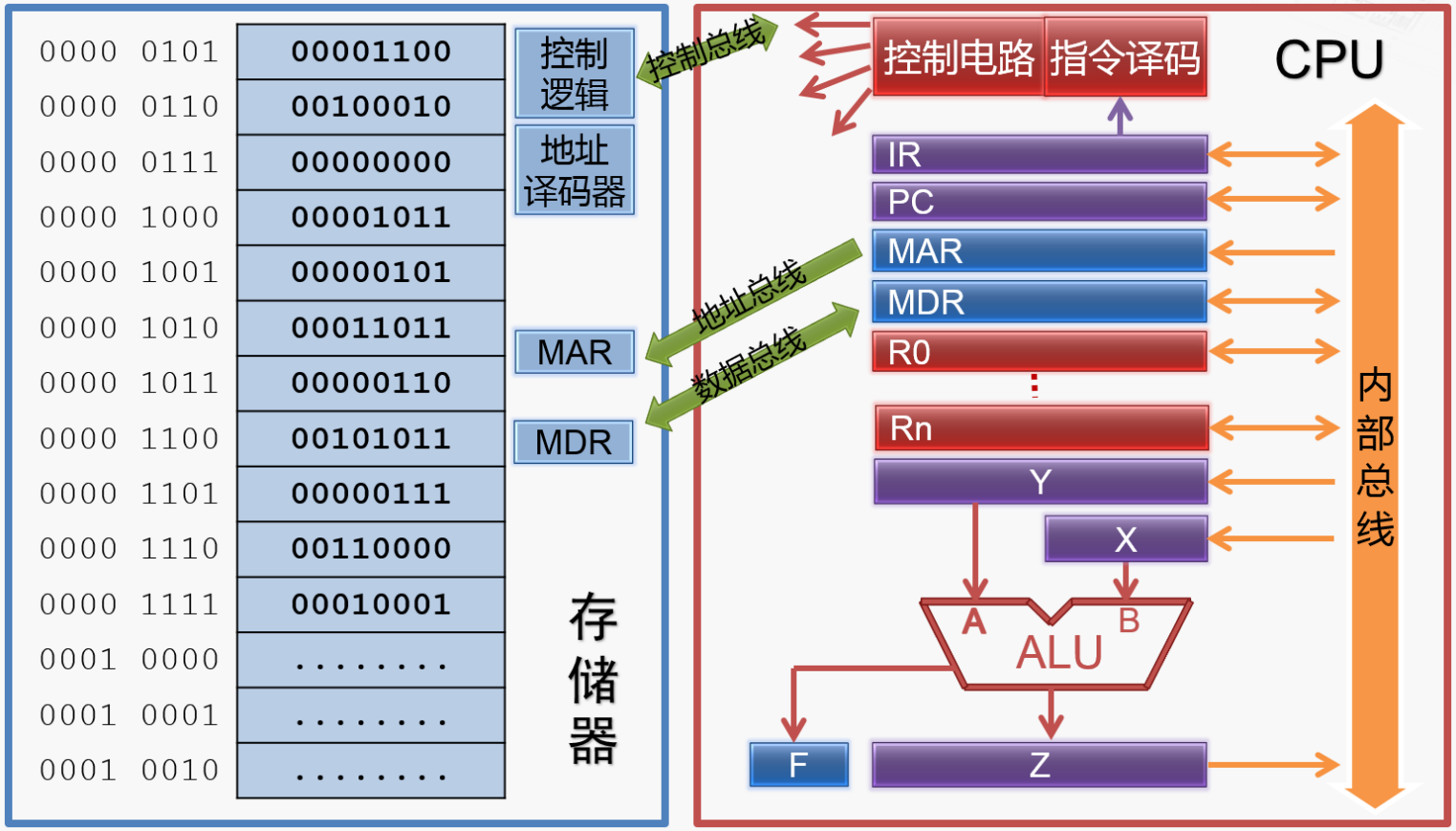
MIPS指令的主要特点（2）

- 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]
```

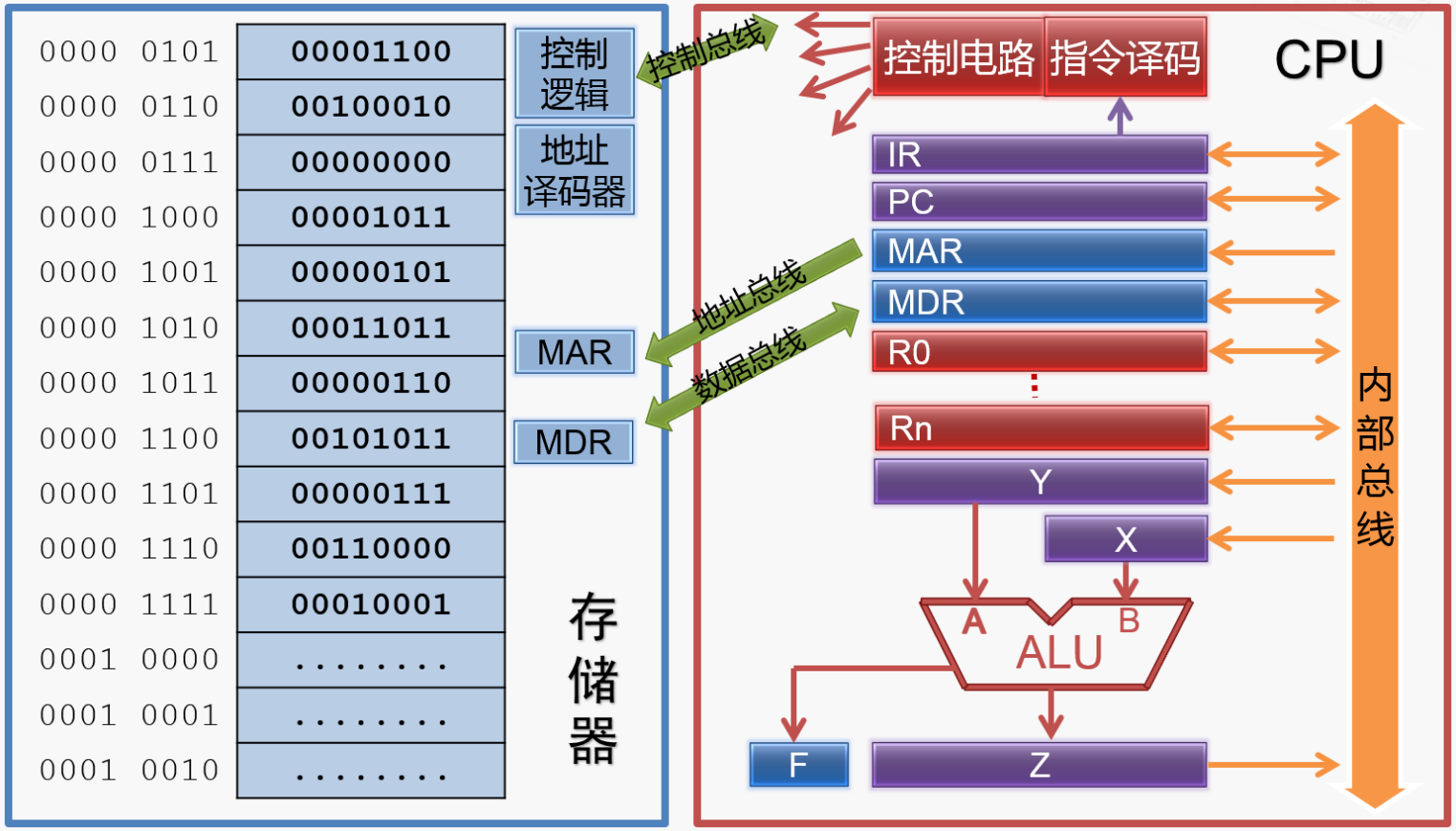


MIPS指令的主要特点（3）

- 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]
```

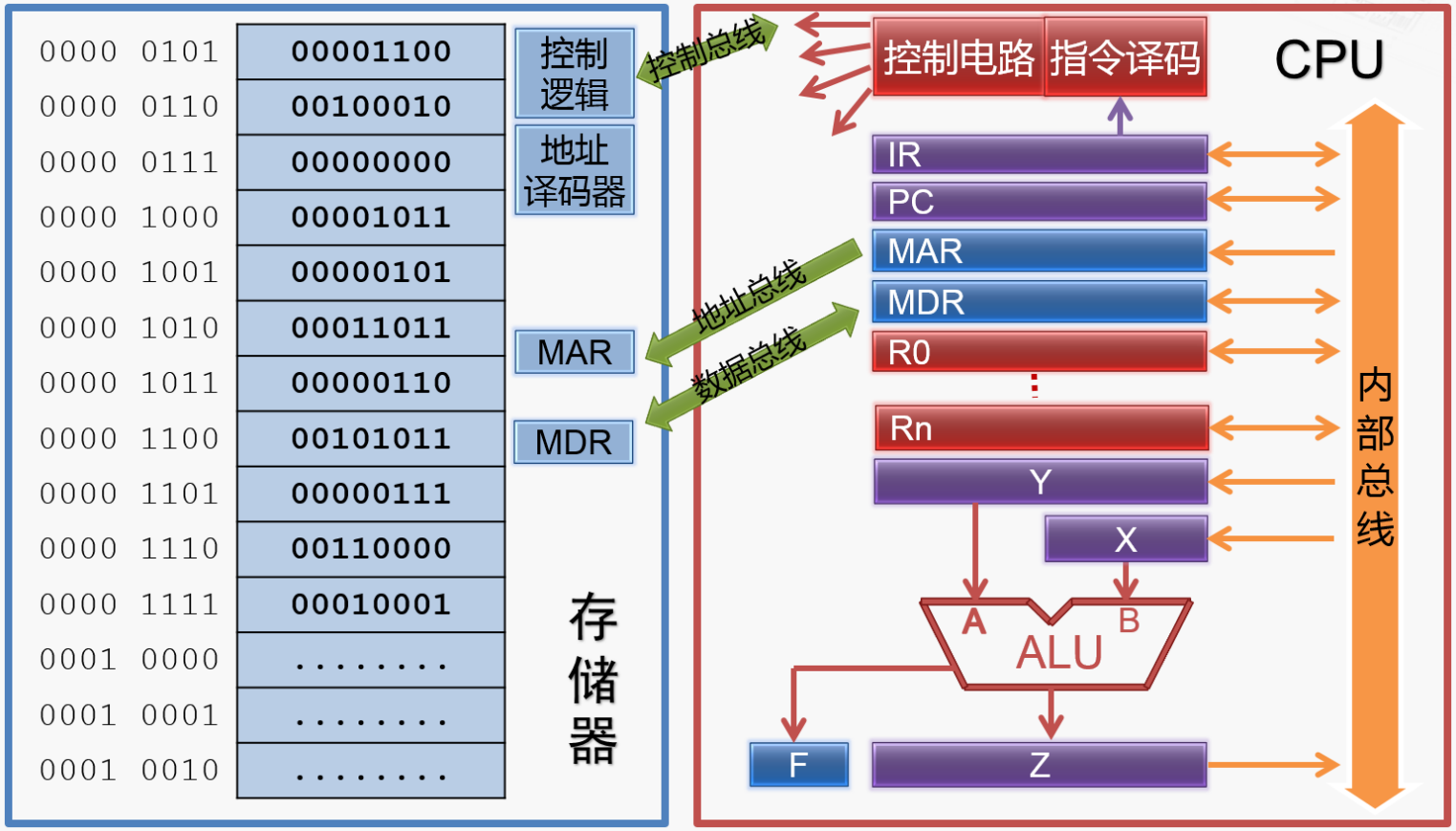


MIPS指令的主要特点（4）

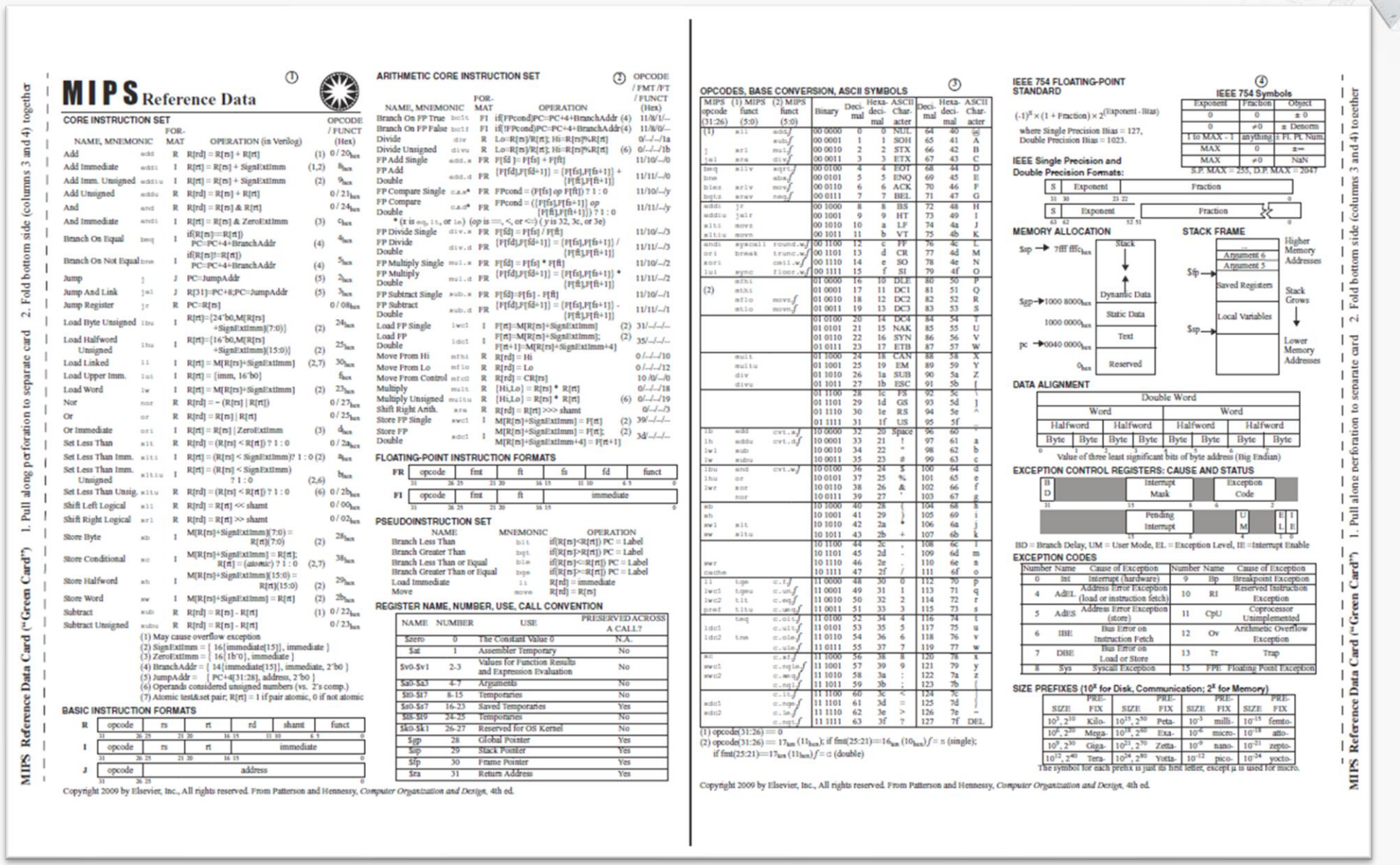
- 指令数量少，指令功能简单
  - 一条指令只完成一个操作，简化指令的执行过程

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-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] op F[ft]) ? 1 : 0		11/10/--/y
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0/--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0/--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS (1) opcode (31:26)	MIPS (1) funct (5:0)	MIPS (2) funct (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
(1)	sll	add.f	00 0000	0	0	NUL	64	40	@
		sub.f	00 0001	1	1	SOH	65	41	A
j	srl	mul.f	00 0010	2	2	STX	66	42	B
jal	sra	div.f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.f	00 0100	4	4	EOT	68	44	D
bne		abs.f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.f	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.f	00 1101	13	d	CR	77	4d	M
xori		ceil.w.f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.f	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movs.f	01 0010	18	12	DC2	82	52	R
	mtlo	movn.f	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.f	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.f	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.f	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.f	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.f	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.f	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.f	11 0011	51	33	3	115	73	s
	teq	c.olt.f	11 0100	52	34	4	116	74	t
ldc1		c.ult.f	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.f	11 0110	54	36	6	118	76	v
		c.ule.f	11 0111	55	37	7	119	77	w
sc		c.sf.f	11 1000	56	38	8	120	78	x
swc1		c.ngle.f	11 1001	57	39	9	121	79	y
swc2		c.seq.f	11 1010	58	3a	:	122	7a	z
		c.ngl.f	11 1011	59	3b	;	123	7b	{
		c.lt.f	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.f	11 1101	61	3d	=	125	7d	}
sdc2		c.le.f	11 1110	62	3e	>	126	7e	~
		c.ngt.f	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0
(2) opcode(31:26) == 17_{hex} (11_{hex}); if fmt(25:21)==16_{hex} (10_{hex}) f = s (single); if fmt(25:21)==17_{hex} (11_{hex}) f = d (double)

IEEE 754 FLOATING-POINT STANDARD

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE Single Precision and Double Precision Formats:

S.P. MAX = 255, D.P. MAX = 2047

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

MEMORY ALLOCATION

STACK FRAME

DATA ALIGNMENT

Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

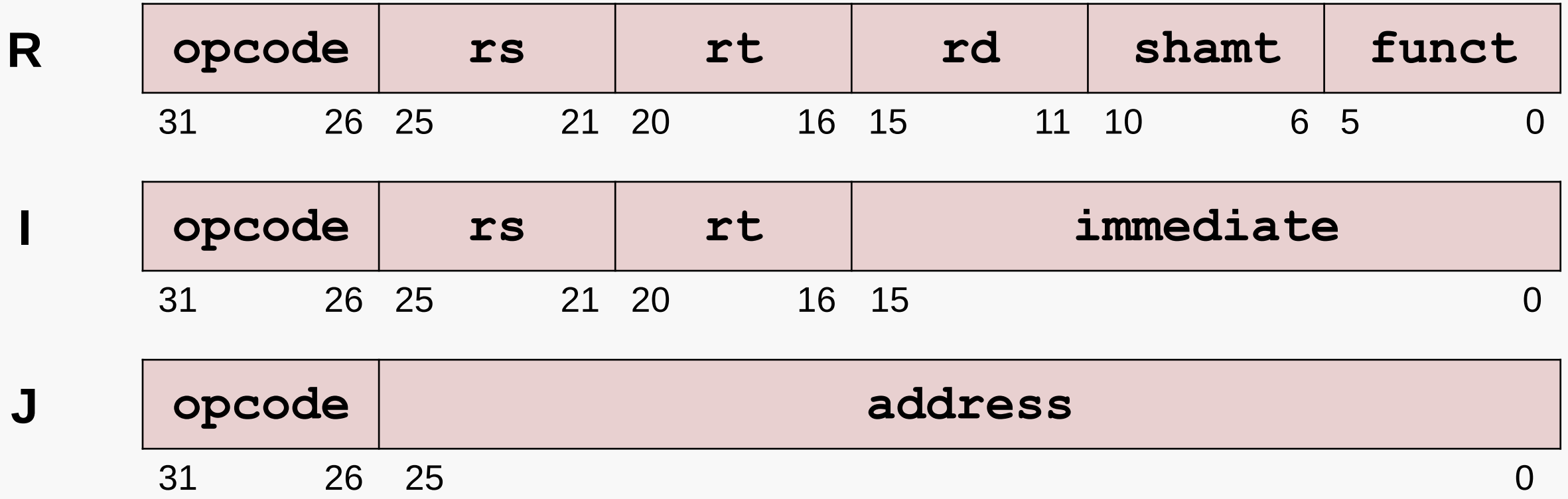
SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.

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

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	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

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

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

R型指令的格式（1）

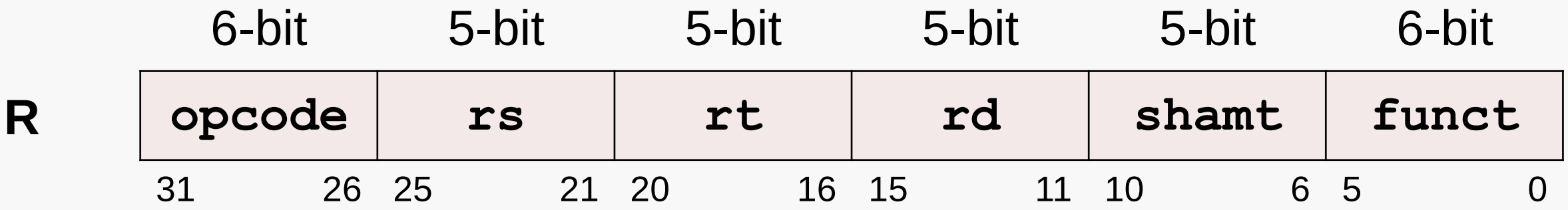
- R型指令格式包含6个域
  - 2个6-bit域，可表示0~63的数
  - 4个5-bit域，可表示0~31的数

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

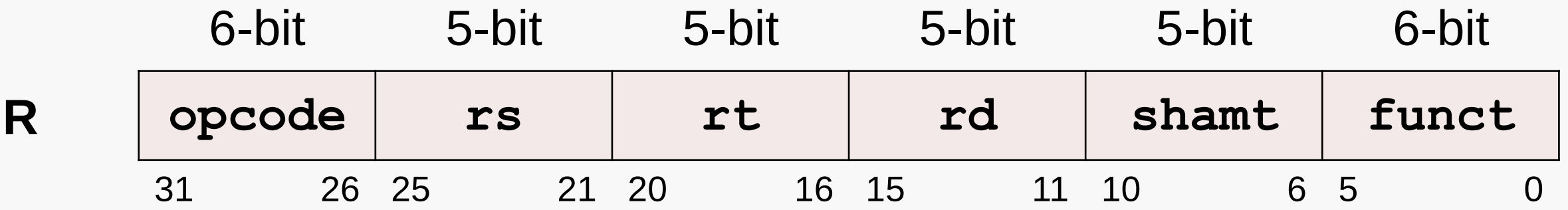
与opcode域组合，精确地指定指令的类型

funct



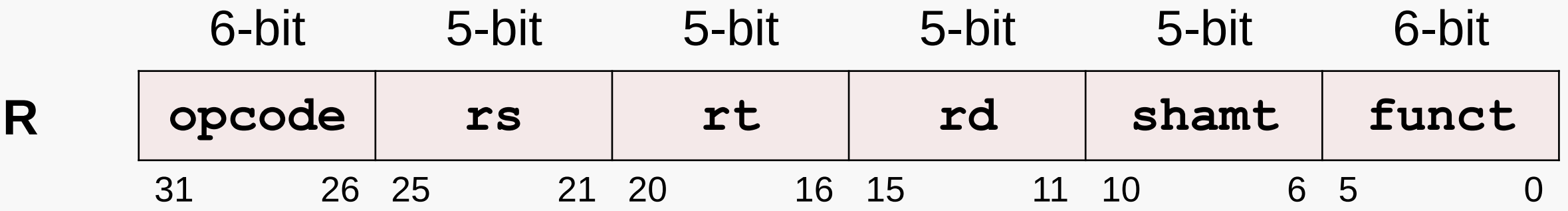
R型指令的格式（2）

- `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）

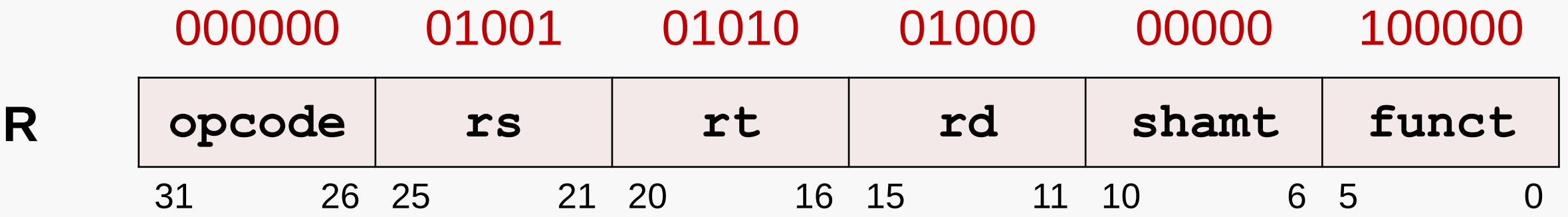
- **add $8,$9,$10**　# R[rd]=R[rs]+R[rt]
  - 查指令编码表得到：

  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：

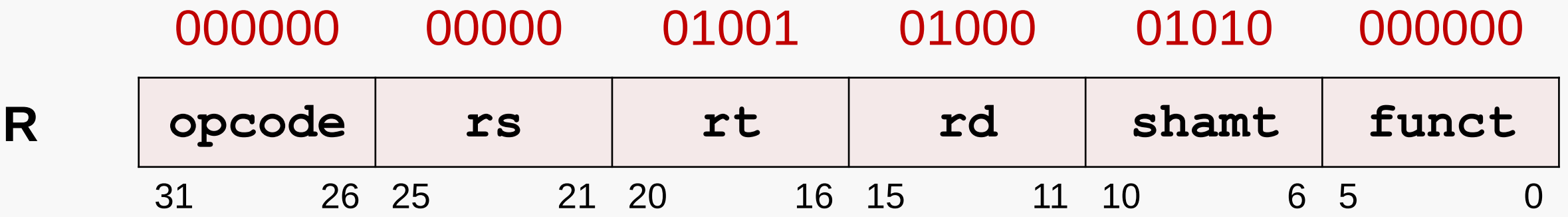
  opcode = 0， funct = 0， rs = 0 （未使用的寄存器）

  - 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

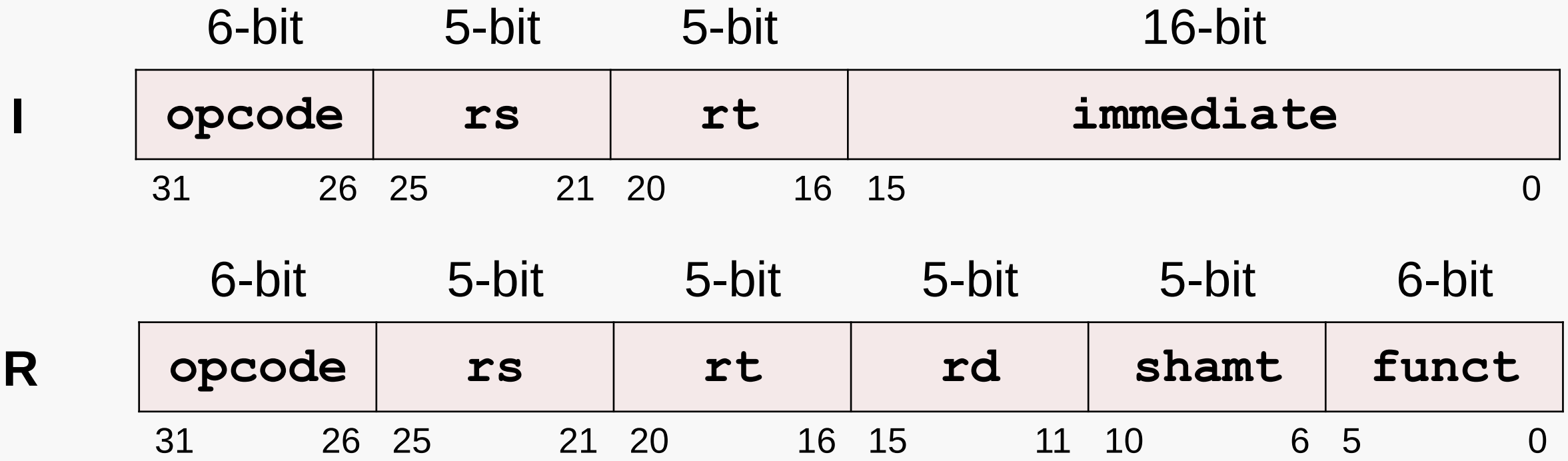
通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

不同维度的指令分类（示例）

	R型指令	I型指令	J型指令
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*

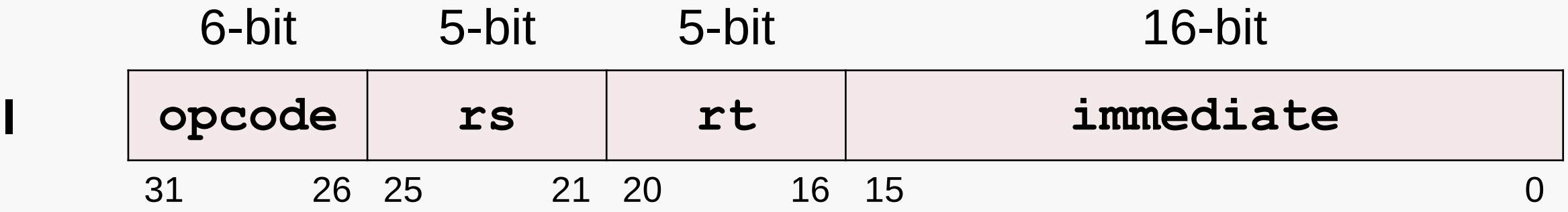
I型指令的格式（1）

- 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

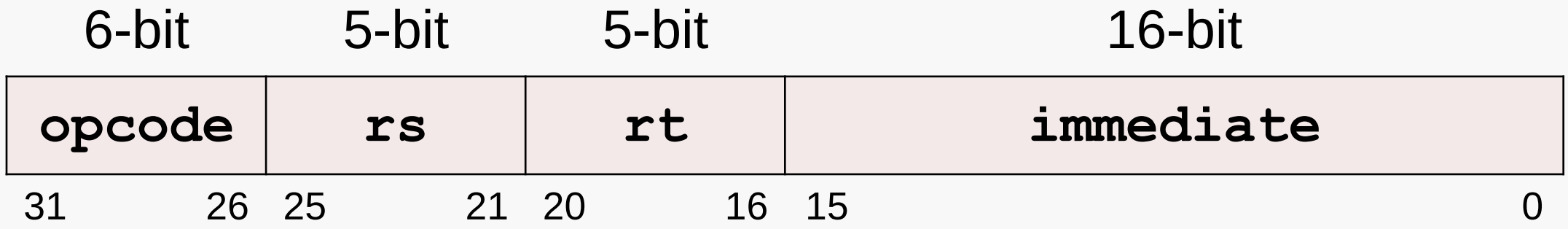


I型指令的格式（3）

- 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



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：

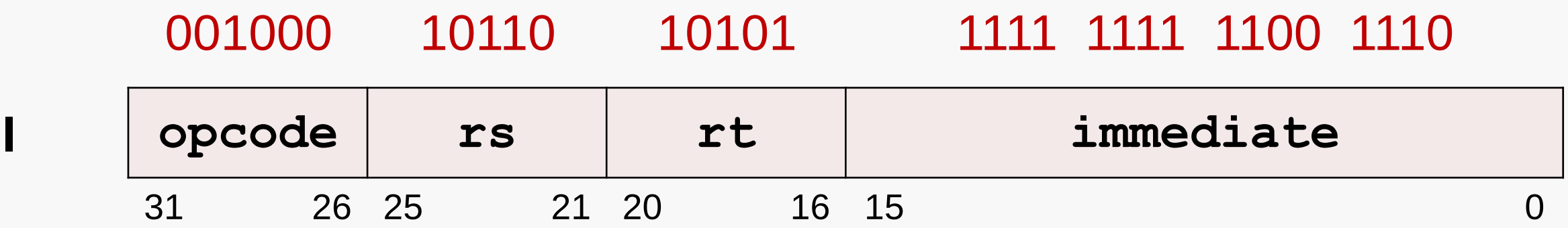
  opcode = 8

  - 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）



I型指令的编码示例（2）

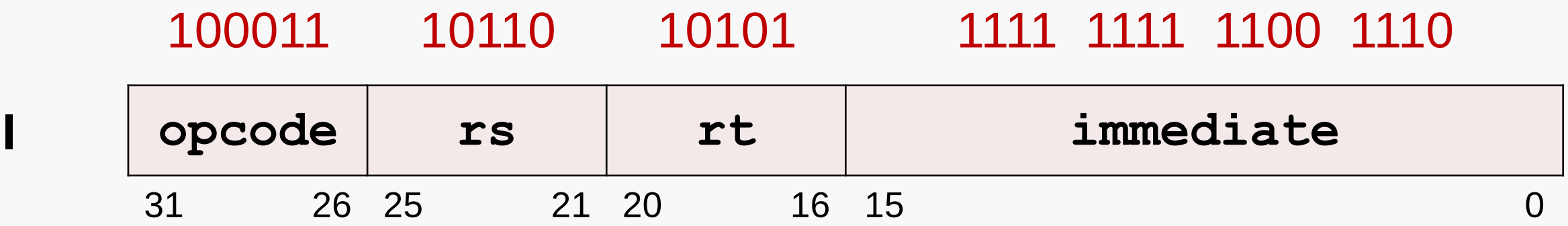
- **lw $21,-50($22)** # $21=Mem[$22+(-50)]
  - 查指令编码表得到：

  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）



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：

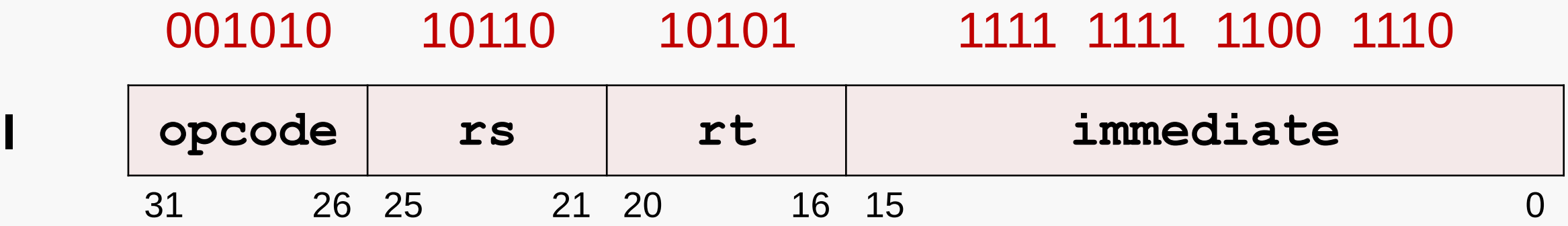
  opcode = 10

  - 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）



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

北京大学 Peking University