本节主题

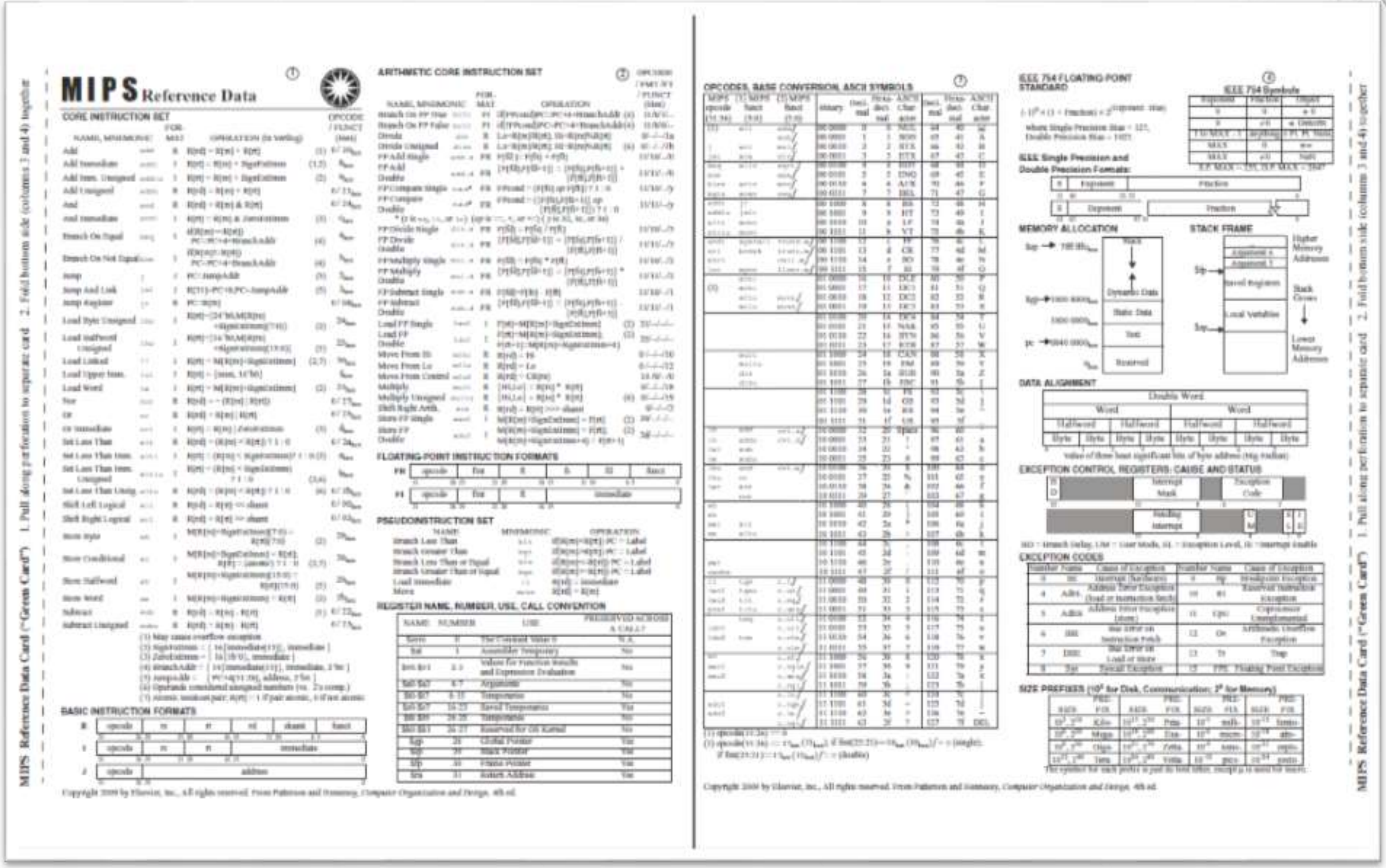
MIPS指令简介

北京大学 · 慕课

计算机组成

制作人：陆俊林

MIPS指令



MIPS Reference Data

CORE INSTRUCTION SET

ARITHMETIC CORE INSTRUCTION SET

BASIC INSTRUCTION FORMATS

FLOATING-POINT INSTRUCTION FORMATS

PSEUDOINSTRUCTION SET

REGISTER NAME, NUMBER, USE, CALL CONVENTION

OPCODES, BASE CONVERSION, ASCII SYMBOLS

IEEE 754 FLOATING-POINT STANDARD

IEEE 754 Symbols

IEEE Single Precision and Double Precision Formats:

MEMORY ALLOCATION

STACK FRAME

DATA ALIGNMENT

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

EXCEPTION CODES

SIZE PREFIXES

[illegible]

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

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0

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

J	opcode	address
	31 26	25 0

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

思考：
　　为什么不将opcode域和funct域合并成一个12-bit的域？

opcode：用于指定指令的类型。对于所有R型指令，该域的值均为0

funct：与opcode域组合，精确地指定指令的类型

	6-bit	5-bit	5-bit	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

	6-bit	5-bit	5-bit	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（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

	6-bit	5-bit	5-bit	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型指令的编码示例

- **add $8,$9,$10**
  - 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）

	000000	01001	01010	01000	00000	100000
R	**opcode**	**rs**	**rt**	**rd**	**shamt**	**funct**
	31 26	25 21	20 16	15 11	10 6	5 0

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

	6-bit	5-bit	5-bit	16-bit
I	opcode	rs	rt	immediate
	31 26	25 21	20 16	15 0

	6-bit	5-bit	5-bit	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

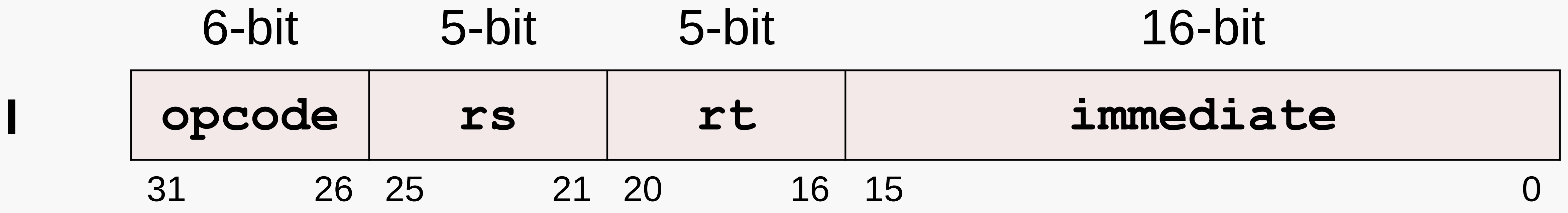
I型指令的格式（2）

- 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

	6-bit	5-bit	5-bit	16-bit
I	opcode	rs	rt	immediate
	31 26	25 21	20 16	15 0

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型指令的编码示例

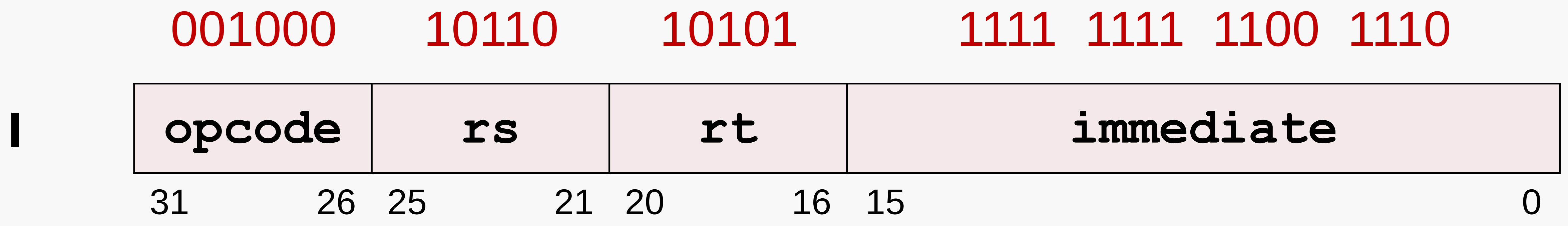
- **addi $21,$22,-50** # $21=$22+(-50)
  - 查指令编码表得到：

    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）



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

分支指令的分类

- **Branch**
  - 分支：改变控制流

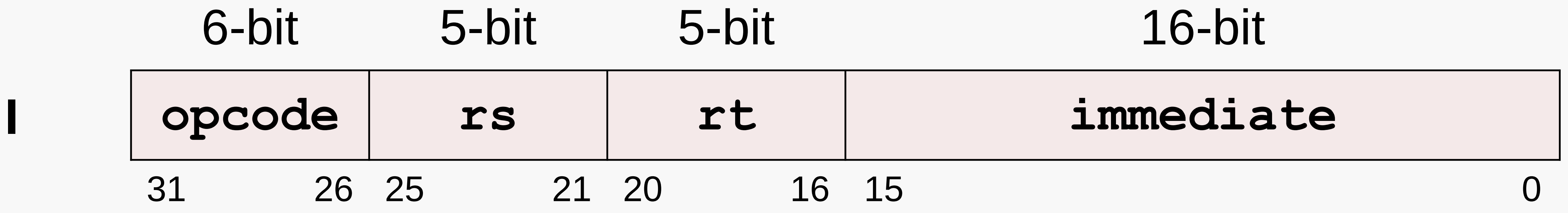
- Conditional Branch
  - 条件分支：根据比较的结果改变控制流
  - 两条指令：branch *if* equal (`beq`) ；branch *if not* equal (`bne`)
- Unconditional Branch
  - 非条件分支：无条件地改变控制流
  - 一条指令： jump (`j`)

条件分支指令（I型）

- 条件分支
  - `beq rs,rt,imm          # opcode=4`
  - `bne rs,rt,imm          # opcode=5`

- 格式：**`beq reg1,reg2,L1`**

```
if (value in reg1)==(value in reg2)
    goto L1
```



条件分支指令的示例

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

C语言代码

```
 beq $s3,$s4,True       # branch i==j
 sub $s0,$s1,$s2        # f=g-h(false)
 j Fin                  # goto Fin

True: add $s0,$s1,$s2 # f=g+h (true)
Fin: …
```

MIPS汇编语言代码

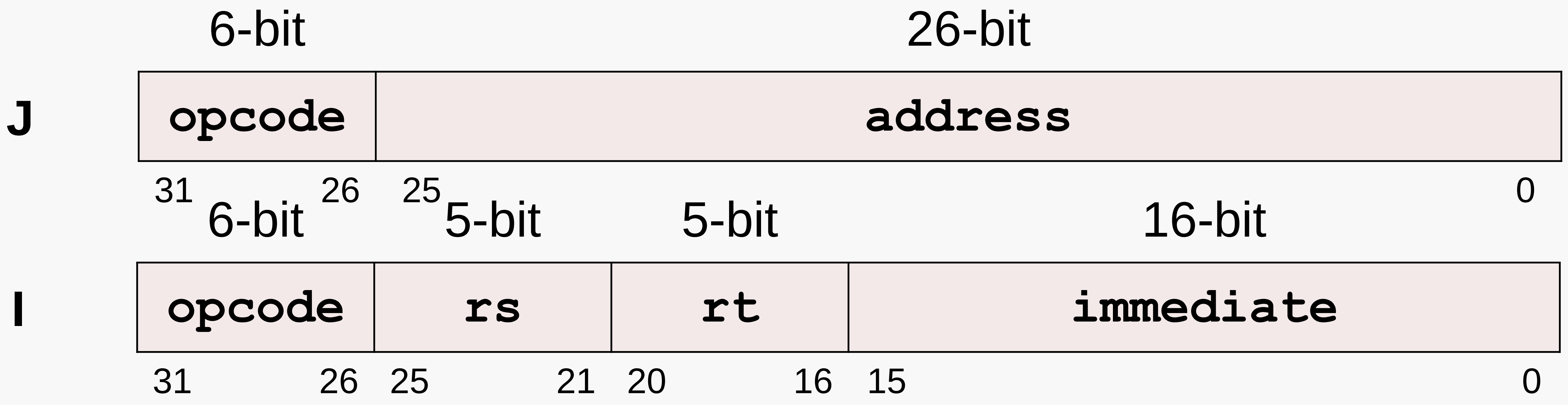
条件分支指令的目标地址范围

- 如何充分发挥16-bit的作用？
  - 以当前PC为基准，16-bit位移量可以表示$\pm 2^{15}$ bytes
  - MIPS的指令长度固定为32-bit（word）
  - 16-bit位移量可以表示 $\pm 2^{15}$ words = $\pm 2^{17}$ bytes（±128KB）
- 目标地址计算方法：
  - 分支条件不成立，`PC = PC + 4 =` next instruction
  - 分支条件成立，`PC = (PC+4) + (immediate*4)`

	6-bit	5-bit	5-bit	16-bit
I	opcode	rs	rt	immediate
	31 26	25 21	20 16	15 0

非条件分支指令（J型）

- 在不需要条件判断的情况下，如何扩大目标地址范围
  - 理想情况，直接使用32-bit地址
  - 冲突：MIPS的指令长度固定为32-bit，`opcode`占用了6-bit
- 目标地址计算方法：
  - `New PC ={(PC+4)[31..28], address, 00}`



两种分支指令示例

- 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	

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

```
       bne $s3,$s4,Else
       add $s0,$s1,$s2
       j Exit
Else:  sub $s0,$s1,$s2
Exit:
```

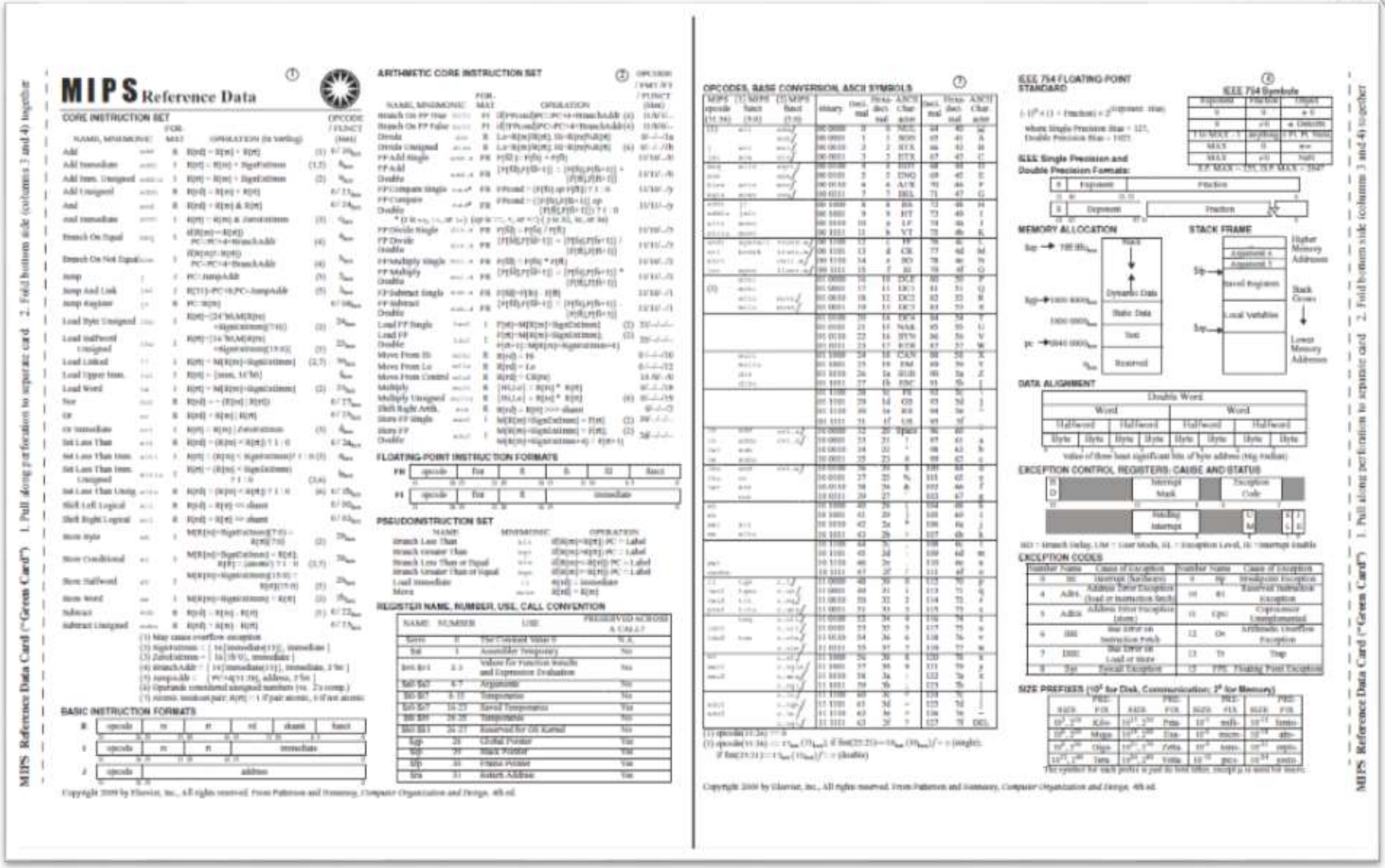
非条件分支指令（R型）

- J型指令的目标地址范围：±2^{28}bytes（±256MB）
- 如何到达更远的目标地址
  - 2次调用`j`指令
  - 使用`jr`指令：`jr rs`

	6-bit	5-bit	5-bit	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

	6-bit	26-bit
J	`opcode`	`address`
	31 26	25 0

MIPS指令



MIPS Reference Data

CORE INSTRUCTION SET

[illegible]

ARITHMETIC CORE INSTRUCTION SET

[illegible]

FLOATING-POINT INSTRUCTION FORMATS

[illegible]

PSEUDOINSTRUCTION SET

[illegible]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

[illegible]

BASIC INSTRUCTION FORMATS

[illegible]

OPCODES, BASE CONVERSION, ASCII SYMBOLS

[illegible]

IEEE 754 FLOATING-POINT STANDARD

IEEE 754 Symbols

[illegible]

IEEE Single Precision and Double Precision Formats:

[illegible]

MEMORY ALLOCATION

STACK FRAME

[illegible]

DATA ALIGNMENT

[illegible]

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

[illegible]

EXCEPTION CODES

[illegible]

SIZE PREFIXES

[illegible]

本节小结

MIPS指令简介

北京大学 · 慕课

计算机组成

制作人：陆俊林