# 第二章　指令系统原理与实例

- 2.1 简介
- 2.2 指令集系统结构的分类
- 2.3 存储器寻址
- 2.4 操作数的类型
- 2.5 指令系统的操作
- 2.6 控制流指令
- 2.7 指令系统的编码
- 2.8 编译器的角色
- 2.9 MIPS系统结构
- 2.10 谬误和易犯的错误
- 2.11 结论

# 2.5指令系统的操作

- 指令的功能设计：一种指令集结构中的指令到底要支持哪些类型的操作呢？

# 2.5指令系统的操作

- 大多数指令集系统结构支持的操作：

| 操作类型 | 举例 |
| --- | --- |
| 算术和逻辑运算 | 定点算术和逻辑操作：加、减、与、或、乘、除 |
| 数据传输 | Load-store指令（在REG-MEN结构计算机上是传送指令） |
| 控制 | 条件转移、跳转、过程调用和返回、陷阱 |
| 系统 | 操作系统调用、虚拟存储器管理指令 |
| 浮点 | 浮点操作：加、减、乘、除、比较 |
| 十进制 | 十进制加、十进制乘、十进制到字符的转换 |
| 字符串 | 字符串传送、字符串比较、字符串匹配 |
| 图像 | 像素、顶点操作、压缩/解压缩操作 |

Figure A.12

# 2.5指令系统的操作

| 操作类型 | 举例 |
| --- | --- |
| 算术和逻辑运算 | 定点算术和逻辑操作：加、减、与、或、乘、除 |
| 数据传输 | Load-store指令（在REG-MEN结构计算机上是传送指令） |
| 控制 | 条件转移、跳转、过程调用和返回、陷阱 |
| 系统 | 操作系统调用、虚拟存储器管理指令 |
| 浮点 | 浮点操作：加、减、乘、除、比较 |
| 十进制 | 十进制加、十进制乘、十进制到字符的转换 |
| 字符串 | 字符串传送、字符串比较、字符串匹配 |
| 图像 | 像素、顶点操作、压缩/解压缩操作 |

**指令系统有一条共同的规律：**

**使用最多的是一些简单指令。一般所有的计算机都提供前三类指令。指令集对后四类指令的支持数量可能为0，也可能包含大量特殊指令。**

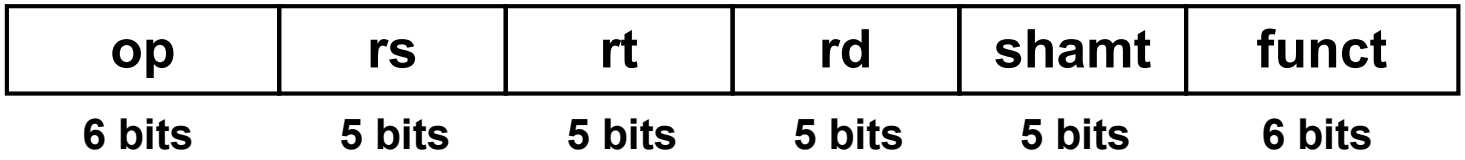
# MIPS Logical Operations

- **Instructions for bitwise manipulation**

| Operation | C | Java | MIPS |
|---|---|---|---|
| Shift left | << | << | `sll` |
| Shift right | >> | >>> | `srl` |
| Bitwise AND | & | & | `and, andi` |
| Bitwise OR | \| | \| | `or, ori` |
| Bitwise NOT | ~ | ~ | `nor` |

- **Useful for extracting and inserting groups of bits in a word**

# Shift Operations




- **shamt: how many positions to shift**
- **Shift left logical**
  - Shift left and fill with 0 bits
  - `sll` by *i* bits multiplies by $2^i$
- **Shift right logical**
  - Shift right and fill with 0 bits
  - `srl` by *i* bits divides by $2^i$ (unsigned only)

# Logic Shifting sll_1.s

- **Shift Left: `sll $s1,$s2,2` #s1=s2<<2**
  - Store in $s1 the value from $s2 shifted 2 bits to the left (they fall off end), inserting 0's on right; << in C.

    Before: $0000\ 0002_{hex}$

    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$

    After: $0000\ 0008_{hex}$

    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000_{two}$

- **Shift Right: `srl` is opposite shift; >>**

# COD 5e Exercise 2.3

- For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.
- 下面的C语言表达式对应的MIPS汇编代码是什么？假设变量f、g、h、i和j分别赋值给寄存器$s0、 $s1、$s2、 $s3和$s4。假设数组A和B的基地址分别在寄存器$s6和$s7中。
- B[8] = A[i−j];

# Answer

- For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.
- B[8] = A[i−j];
- sub $t0 , $s3 , $s4
- sll $t0 , $t0 , 2
- 
- add $t0 , $t0 , $s6
- 
- lw $t1 , 0($t0)
- sw $t1 , 32($s7)

# Arithmetic Right Shifting

sra_1.s

- **Shift right arithmetic moves *n* bits to the right (insert high order sign bit into empty bits)**
- **For example, if register $s0 contained**

  $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0111_{two} = -25_{ten}$

- **If executed sra $s2, $s0, 4, result is:**

  **$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$**

  - Unfortunately, this is NOT same as dividing by $2^n$
  - Fails for odd negative numbers
  - C arithmetic semantics is that division should round towards 0

# Shift Left Logical sll_2.s

```
sll  $t2,$s0,4      # reg $t2 = reg $s0 << 4 bits
```

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

| 0 | 0 | 16 | 10 | 4 | 0 |
|---|---|---|---|---|---|
| 000000 | 00000 | 10000 | 01010 | 00100 | 000000 |
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |
| 000000 | 00000 | 10000 | 01010 | 00100 | 000000 |

decimal

0 0 1 0 5 1 0 0

# AND Operations

and_1.s

- **Useful to mask bits in a word**
  - Select some bits, clear others to 0

```
and $t0, $t1, $t2
```

# OR Operations

or_1.s

- **Useful to include bits in a word**
  - Set some bits to 1, leave others unchanged

```
or $t0, $t1, $t2
```

# NOT Operations

not_1.s

- **Useful to invert bits in a word**
  - Change 0 to 1, and 1 to 0
- **MIPS has NOR 3-operand instruction**
  - a NOR b == NOT ( a OR b )

```
nor $t0, $t1, $zero
```

Register 0: always read as zero

| | |
|---|---|
| $t1 | 0000 0000 0000 0000 0011 1100 0000 0000 |
| $t0 | 1111 1111 1111 1111 1100 0011 1111 1111 |

# COD 5e Exercise 2.19.1

- Assume the following register contents:
- 假设如下寄存器内容：
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [1] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- 对于以 上的寄存器内容， 执行下面的指令序列后$t2的值是多少？
- sll $t2, $t0, 04
- or $t2, $t2, $t1

# COD 5e Exercise 2.19.1

- Assume the following register contents:
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [1] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- sll $t2, $t0, 04
- or $t2, $t2, $t1

# COD 5e Exercise 2.19.1

- Assume the following register contents:
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [1] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- sll $t2, $t0, 04
- or $t2, $t2, $t1

**0xBABEFEF8**

# 2.19.2

- Assume the following register contents:
- 假设如下寄存器内容：
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [2] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- 对于以 上的寄存器内容， 执行下面的指令序列后$t2的值是多少？
- sll $t2, $t0, 4
- andi $t2, $t2,-1

# 2.19.2

- Assume the following register contents:
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [2] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- sll $t2, $t0, 4
- andi $t2, $t2,-1

# 2.19 (2)

- Assume the following register contents:
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [2] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- sll $t2, $t0, 4
- andi $t2, $t2, –1

**0xAAAAAAA0**

```
# the parser will report error : -1 is out of range (0..65535)
# andi $t2 , $t2 , -1

#dengjian : method 2 ,     the result is 0xaaaaaaa0
li $t3 , -1
and $t2 , $t2 , $t3

#dengjian : method 3 , but the result is 0x0000aaa0
# andi $t2 , $t2 , 0xffff
```

# 2.19.3

- Assume the following register contents:
- 假设如下寄存器内容：
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [3] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- 对于以 上的寄存器内容， 执行下面的指令序列后$t2的值是多少？
- srl $t2, $t0, 3
- andi $t2, $t2, 0xFFEF

# 2.19 (3)

- Assume the following register contents:
- $t0 = 0xAAAAAAAA, $t1 = 0x12345678
- [3] For the register values shown above, what is the value of $t2 for the following sequence of instructions?
- srl $t2, $t0, 3
- andi $t2, $t2, 0xFFEF

0x00005545

# Example COD 5e Exercise 2.20

- Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register $t0 and uses the value of this field to replace bits 31 down to 26 in register $t1 without changing the other 26 bits of register $t1.
- 找出完成如下功能的最短的MIPS指令序列：从寄存器$t0中提取第16位到第11位，然后使用这些位替换寄存器$t1的第31位到第26位， 保持其他位不变。

# COD 5e Exercise 2.20

- **Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register $t0 and uses the value of this field to replace bits 31 down to 26 in register $t1 without changing the other 26 bits of register $t1.**

# 2.20

- **Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register $t0 and uses the value of this field to replace bits 31 down to 26 in register $t1 without changing the other 26 bits of register $t1.**

```
srl $t2 , $t0 , 11
sll $t2 , $t2 , 26

sll $t3 , $t1 , 6
srl $t3 , $t3 , 6

or  $t4 , $t2 , $t3
```

# 2.5指令系统的操作

- 80X86中执行最多的前10类指令：

| 排名 | 80x86指令 | 定点平均值（占百分比） |
|---|---|---|
| 1 | 载入（MOV） | 22% |
| 2 | 条件转移 | 20% |
| 3 | 比较 | 16% |
| 4 | 存储（MOV） | 12% |
| 5 | 加 | 8% |
| 6 | 与 | 6% |
| 7 | 减 | 5% |
| 8 | Reg-Reg传输 | 4% |
| 9 | 调用 | 1% |
| 10 | 返回 | 1% |
| 总计 | | 95% |

80X86上运行的定点程序中，10类占96%。

因此，它们是最常用的指令，执行起来应该尽量快。

Figure A.13