# 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

# 2.4操作数的类型

- 操作数的类型如何指定：

  第一种，操作数类型可以通过操作码的编码来指定，是最常用的方法；（MIPS：add，add.s，add.d）

  第二种，操作数中用硬件解释的字段表示数据类型。

- 常见的操作数类型：

| 数据类型 | 位数 | 数值范围 | C语言中的对应 |
|---|---|---|---|
| 字节 | 8 | -128 — +127 | signed char |
| 无符号字节 | 8 | 0 — 255 | unsigned char |
| 半字 | 16 | -32768 — +32767 | short int |
| 无符号半字 | 16 | 0 — 65535 | unsigned short int |
| 字 | 32 | -2147483648 — +2147483647 | int |
| 无符号字 | 32 | 0 — 4294967295 | signed int |
| 单精度浮点数 | 32 | | float |
| 双精度浮点数 | 64 | | double |

定点通常用二进制补码表示，字符通常是ASCII编码格式。

# 2.4操作数的大小与类型

- 64位处理器中基准测试程序中数据访问的大小分布

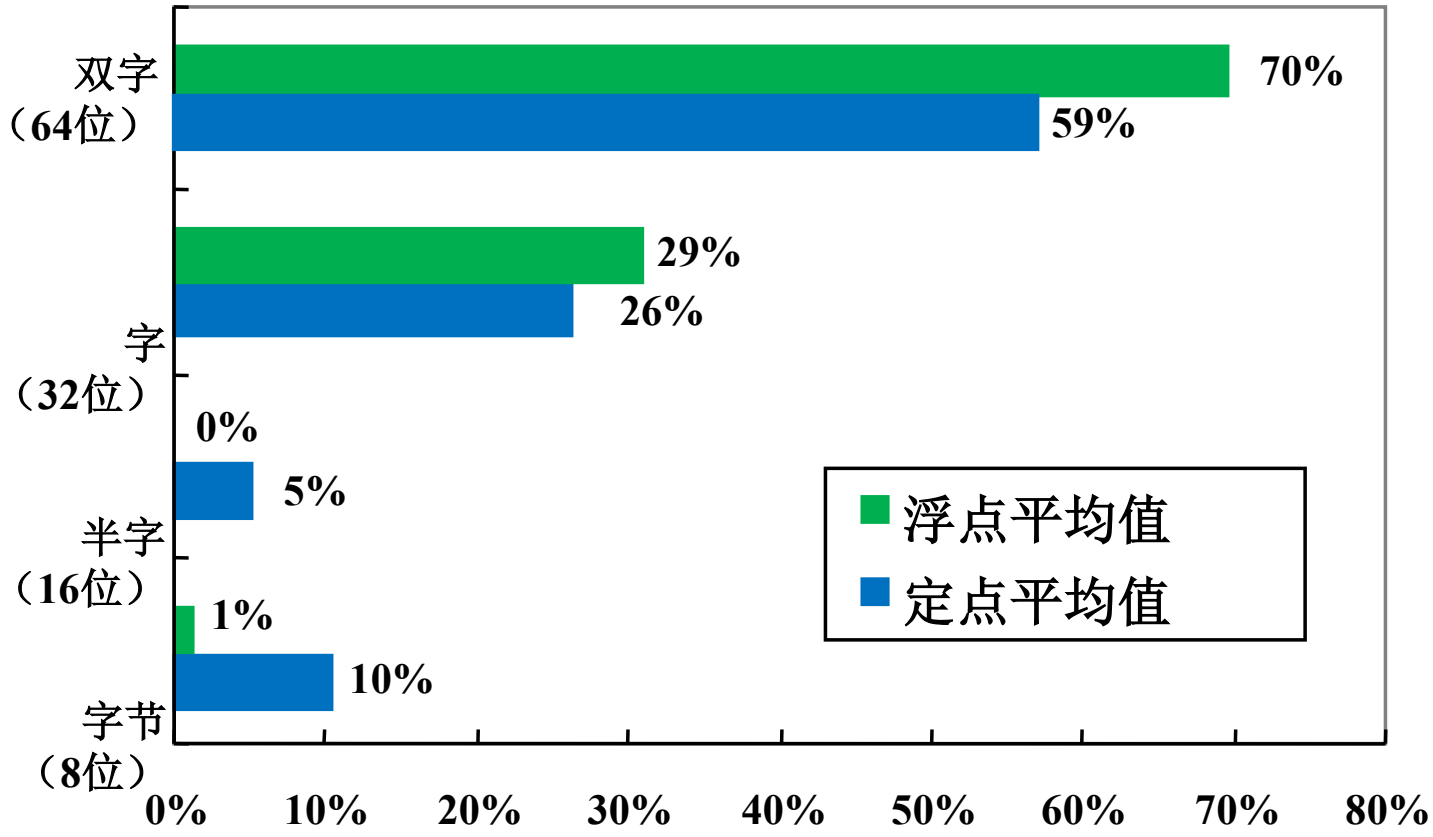



Figure A.11

# 2.4操作数的大小与类型

- 基准测试程序中数据访问的大小分布

由于系统为64位地址，双字数据类型可用于浮点程序中的双精度和存储器地址。在32位地址的计算机中，64位地址将被32位地址取代，定点程序中几乎所有的双字访问都转换成单字访问。

在一些系统结构中，寄存器的数据可能以字节或者半字来访问（这种情况很少发生）。在VAX计算机中，统计数据表明这种情况不超过所有寄存器的12%，大约占这些程序中所有操作数访问的6%。

# IBM 370

- 信息有字节、半字（双字节）、单字（4字节）和双字（8字节）等宽度。
- 主存宽度为8个字节。采用按字节编址，各类信息都是用该信息的首字节地址来寻址。

# MIPS:I-Format Instructions

## The Immediate Field:

- `addi, slti, sltiu, lw, sw`, the immediate is sign-extended to 32 bits. Thus, it's treated as a signed integer.
- `andi, ori`, the immediate is zero-extended to 32 bits. Thus, it's treated as a unsigned integer.

# Two's-Complement Examples

- Assume for simplicity 4 bit width, -8 to +7 represented

```
 3  0011        3  0011        -3  1101
+2  0010   + (-2) 1110   + (-2)  1110
 5  0101        1 10001        -5 11011
```

*Overflow when magnitude of result too big to fit into result representation*

```
 7  0111        -8  1000
+1  0001   + (-1)  1111
-8  1000       +7 10111
```

*Overflow!* *Overflow!*

Carry into MSB = Carry Out MSB

Carry into MSB ≠ Carry Out MSB

*Carry in = carry from less significant bits*
*Carry out = carry to more significant bits*

# Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

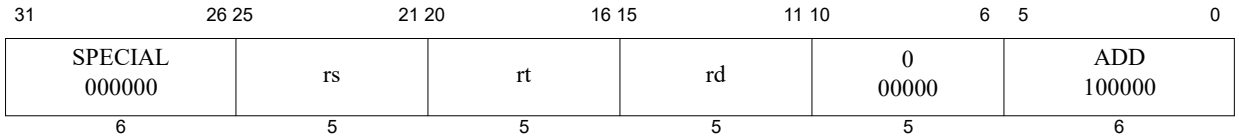
| | |
|---|---|
| +7: | 0000 0000 … 0000 0111 |
| –6: | 1111 1111 … 1111 1010 |
| +1: | 0000 0000 … 0000 0001 |

- Overflow if result out of range
  - Subtracting two +ve or two –ve operands, no overflow
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

# Overflow handling in MIPS

- **Some languages detect overflow (Ada), some don't (most C implementations)**
- **MIPS solution is 2 kinds of arithmetic instructions:**
  - These cause overflow to be detected
    - add (add)
    - add immediate (addi)
    - subtract (sub)
  - These do not cause overflow detection
    - add signed (addu)
    - add immediate signed (addiu)
    - subtract signed (subu)
- **Compiler selects appropriate arithmetic**
  - MIPS C compilers produce addu, addiu, subu

# add



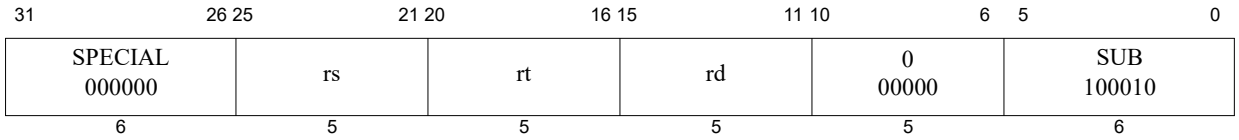

**Description:** GPR[rd] ← GPR[rs] + GPR[rt]

**Operation:**

```
temp ← (GPR[rs]31||GPR[rs]31..0) + (GPR[rt]31||GPR[rt]31..0)

if temp32 ≠ temp31 then
    SignalException(IntegerOverflow)
else
    GPR[rd] ← temp
endif
```

# sub




**Description:** GPR[rd] ← GPR[rs] - GPR[rt]

**Operation:**

```
temp ← (GPR[rs]31||GPR[rs]31..0) - (GPR[rt]31||GPR[rt]31..0)

if temp32 ≠ temp31 then
    SignalException(IntegerOverflow)
else
    GPR[rd] ← temp31..0
endif
```

# Classroom Test

- **COD 5E Exercise 2.12(1)**
- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- What is the value of $t0 for the following assembly code?
- add $t0, $s0, $s1

- **COD 5E Exercise 2.12(2)**
- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- add $t0, $s0, $s1
- Has the result in $t0 been overflow?

# Classroom Test 1

- **COD 5E Exercise 2.12(1)**
- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- What is the value of $t0 for the following assembly code?
- add $t0, $s0, $s1

# Classroom Test 1 Answer

- **COD 5E Exercise 2.12(1)**
- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- What is the value of $t0 for the following assembly code?
- add $t0, $s0, $s1
- `0x50000000`

# Classroom Test 2

- **COD 5E Exercise 2.12(2)**
- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- add $t0, $s0, $s1
- Has the result in $t0 been overflow?

# Classroom Test 2 Answer

- **COD 5E Exercise 2.12(2)**
- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- add $t0, $s0, $s1
- Has the result in $t0 been overflow?
- overflow