第二章

23336003 陈政宇

1～5

```
01-05: DBADA
```

7～21

```
07-11: CADBD 12-16: CACBB 17-21: ACDBD
```

24～33

```
24-28: DACCA 29-33: BACAA
```

35～39

```
35-39: CACCD
```

41～47

```
41-45: BDBDA 46-47: AC
```

49～52

```
49-52: DBBB
```

54～58

```
54-58: CDDCA
```

6

(1)

整数乘法可以通过加法和位移操作来实现。具体来说，乘法可以看作是多个加法的累积。例如，x * y 可以表示为 x 加 x，共 y 次。

```
unsigned umul(unsigned x, unsigned y) {
    unsigned result = 0;
    for(; y; y >>= 1, x <<= 1) if(y & 1) result += x;
    return result;
}
```

(2)

控制循环次数、加法、位移等操作。

(3)

- 执行时间最长：a) 没有乘法指令
  理由：需要通过循环、加法和位移操作逐位计算乘积，执行时间与 y 的位数成正比。
- 执行时间最短：c) 有使用阵列乘法器实现的乘法指令
  理由：阵列乘法器通过并行计算实现乘法运算，可以在一个时钟周期内完成乘法运算，速度最快。

(4)

- $n = 31, x = 2^{32} - 1, y = 2$ 时，
  无符号：`0x00000001FFFFFFFE`
  有符号：`0x00000001FFFFFFFE`
- `umul` 溢出；`imul` 未溢出。
- 溢出判断方法：
  如果高 n 位不全为 0，则表示发生了溢出。
  如果高 n 位全为 0，则表示没有溢出。

22

(1)

- 由于 n 为无符号整数，当 `n=0` 时，`n-1` 会变为 `0xFFFFFFFF`，导致死循环。
- 不会。因为 `i=0` < `n-1=-1` 会直接退出循环。

(2)

相等。
`f1(23)` 机器数为 `0x00FFFFFF`；
`f2(23)` 机器数为 `0x4B7FFFFF`。

(3)

因为 `IEEE754` 标准下 `float` 尾数共23位，精度不够。

(4)

- **原因**：
  - `int` 类型能表示的最大正整数为 `2^31 - 1`。当计算 `2^32 - 1` 时，结果超出了 `int` 类型的表示范围，发生溢出，结果变为 `-1`。
- **最大的 `n`**：
  - 为了避免溢出，`f1(n)` 的返回值必须在 `int` 类型的表示范围内。
  - `f(n) = 2^(n+1) - 1`，最大值为 `2^31 - 1`。
  - 解方程 `2^(n+1) - 1 <= 2^31 - 1`，得到 `n <= 30`。
  - 因此，最大的 `n` 是 `30`。

(5)

- **对应的值**：
  - `7F80 0000H` 是 IEEE 754 单精度浮点数的表示，表示正无穷大（Infinity）。
- **最大的 `n`（不溢出）**：
  - `float` 类型的最大值是 `3.4028235 × 10^38`。
  - `f(n) = 2^(n+1) - 1`，最大值为 `2^128 - 1`。
  - `2^128` 超出了 `float` 类型的表示范围，因此 `f2(n)` 会溢出。
  - 最大的 `n` 是 `127`。

- **最大的 `n`（精确）**：
    - `float` 类型的尾数部分有 23 位有效数字。
    - `f(n) = 2^(n+1) - 1`，当 `n+1 <= 23` 时，结果可以精确表示。
    - 解方程 `n+1 <= 23`，得到 `n <= 22`。
    - 因此，最大的 `n` 是 `22`。

23

(1)

计算机 M 是 CISC。理由如下：

指令长度不固定，指令格式种类多，寻址方式种类多。例如，`cmp` 指令可以直接比较内存中的值和寄存器中的值。`cmp` 指令占用 3 个字节，而 `shl` 指令占用 2 个字节。

(2)

- `00401020 55 push ebp`: 1 byte
- `0040105E 39 4D F4 cmp dword ptr [ebp-0Ch],ecx`: 3 byte
- `00401066 D1 E2 shl edx,1`: 2 byte
- `0040107F C3 ret`: 1 byte

总字节数为，

$$1 + 3 + 2 + 1 = 7$$

(3)

假设 `i` 存储在 `[ebp-0Ch]`，`n` 存储在 `ecx` 中。

当 `n=0` 时，`n-1` 为 `-1`（即 `0xFFFFFFFF`）。

当 `i=0` 时，`cmp` 指令执行 `0 - (-1)`，即 `0-0xFFFFFFFF`

由于 `0` < `0xFFFFFFFF`，需要借位，因此 CF 置为 1.

(4)

不能。因为 `f2` 中的 `power` 数据类型是 `float`。

34

(1)

按字节编址。

(2)

4字节，32位。

(3)

- `0xFFFA`，值为 `-6`
- `OFFSET` = `PC` + `4` + `OFFSET` × `4`

(4)

由于数据相关而发生阻塞的指令为第2、3、4、6条，因为第2、3、4、6条指令都与各自的前一条指令发生数据相关。第6条指令会发生控制冒险。当前循环的第5条指令与下次循环的第1条指令虽有数据相关，但由于第6条指令后面有3个时钟周期的阻塞，因而消除了该数据相关。

40

(1)

按字节编址；最多能反向跳转 `127` 条指令。

(2)

- 若该指令执行时 CF=0，ZF=0，NF=1。由于`NF=1` 满足转移条件。则该指令执行后 PC 的值是 `1FD4H`。
- 若该指令执行时 CF=1，ZF=0，NF=0。由于 `ZF=0`,`NF=0` 不满足转移条件。则该指令执行后 PC 的值是 `200EH`。

(3)

`CF=1`,`ZF=1`,`NF=0`

(4)

①：指令寄存器；用于存放当前指令。

②：移位寄存器；左移一位，乘2。

③：加法器；偏移量做加法得到跳转地址。

48

(1)

- R1: `x=134`, `86H`
- R5: `z1=x-y=134-246=-112=256-112=144`, `90H`
- R6: `z2=x+y=134+246=380=380-256=124`, `7CH`

(2)

- `m=x=134=-122`
- `k1=m-n=134-246=-112`

(3)

是。理由如下：

- **加法器**：加法器本身不区分带符号和无符号数，它只执行二进制加法。
- **减法器**：减法可以通过加法器和补码来实现，即 a - b 可以表示为 a + (-b)，其中 -b 是 b 的补码。
- **带符号和无符号数**：带符号和无符号数的区别在于解释结果的方式，而不是计算的方式。加法器可以处理二进制数的加法和减法，结果的解释由上下文决定。

(4)

- 判断溢出：
    - 带符号整数加法溢出：如果两个正数相加得到负数，或两个负数相加得到正数，则发生溢出。
    - 带符号整数减法溢出：如果正数减去负数得到负数，或负数减去正数得到正数，则发生溢出。
- 程序段中的溢出：
    - `m = x`：不会溢出，因为 `x` 在带符号整数范围内。

    - `n = y`：会溢出，因为 `y = 246` 超出了 8 位带符号整数的范围。
    - `k1 = m - n`：会溢出，因为 `m` 和 `n` 的值超出了带符号整数的范围。
    - `k2 = m + n`：会溢出，因为 `m` 和 `n` 的值超出了带符号整数的范围。

53

(1)

- 最多可有 16 条指令。
- 最多有 8 个通用寄存器。
- MAR 至少需要 16 位。
- MDR 至少需要 16 位。

(2)

- 转移指令采用相对寻址方式，相对偏移量用补码表示。
- 16 位补码的取值范围为 -2^{15} 到 $2^{15}-1$。
- 因此，相对偏移量的范围是 -32768 到 32767。

(3)

- 机器码（十六进制）：`2315H`
- `R5` 和 `5678H` 的内容会改变；`R5: 5679H`，`(5678H): 68ACH`