Dalvik虚拟机操作码

作者：[Gabor Paller](http://pallergabor.uw.hu/" \t "https://blog.csdn.net/qq373432361/article/details/_blank)    翻译：[YULIANGMAX](http://weibo.com/yuliangmax" \t "https://blog.csdn.net/qq373432361/article/details/_blank)

v1.0

表中的vx、vy、vz表示某个Dalvik寄存器。根据不同指令可以访问16、256或64K寄存器。

表中lit4、lit8、lit16、lit32、lit64表示字面值（直接赋值），数字是值所占用位的长度。

long和double型的值占用两个寄存器，例：一个在v0寄存器的double值实际占用v0,v1两个寄存器。

boolean值的存储实际是1和0，1为真、0为假；boolean型的值实际是转成int型的值进行操作。

所有例子的字节序都采用高位存储格式，例：0F00 0A00的编译为0F, 00, 0A, 00 存储。

有一些指令没有说明和例子，因为我没有在正常使用中看到过这些指令，它们的存在是从这里知道的：[Android opcode constant list](http://code.google.com/android/reference/dalvik/bytecode/Opcodes.html" \t "https://blog.csdn.net/qq373432361/article/details/_blank)。

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode  操作码(hex) | Opcode name  操作码名称 | Explanation  说明 | Example  示例 |
| 00 | nop | 无操作 | 0000 – nop |
| 01 | move vx, vy | 移动vy的内容到vx。两个寄存器都必须在最初的256寄存器范围以内。 | 0110 – move v0, v1  移动v1寄存器中的内容到v0。 |
| 02 | move/from16 vx, vy | 移动vy的内容到vx。vy可能在64K寄存器范围以内，而vx则是在最初的256寄存器范围以内。 | 0200 1900 – move/from16 v0, v25  移动v25寄存器中的内容到v0。 |
| 03 | move/16 | 未知注4 |  |
| 04 | move-wide | 未知注4 |  |
| 05 | move-wide/from16 vx, vy | 移动一个long/double值，从vy到vx。vy可能在64K寄存器范围以内，而vx则是在最初的256寄存器范围以内。 | 0516 0000 – move-wide/from16 v22, v0  移动v0,v1寄存器中的内容到 v22,v23。 |
| 06 | move-wide/16 | 未知注4 |  |
| 07 | move-object vx, vy | 移动对象引用，从vy到vx。 | 0781 – move-object v1, v8  移动v8寄存器中的对象引用到v1。 |
| 08 | move-object/from16 vx, vy | 移动对象引用，从vy到vx。vy可以处理64K寄存器地址，vx可以处理256寄存器地址。 | 0801 1500 – move-object/from16 v1, v21  移动v21寄存器中的对象引用到v1。 |
| 09 | move-object/16 | 未知注4 |  |
| 0A | move-result vx | 移动上一次方法调用的返回值到vx。 | 0A00 – move-result v0  移动上一次方法调用的返回值到v0。 |
| 0B | move-result-wide vx | 移动上一次方法调用的long/double型返回值到vx,vx+1。 | 0B02 – move-result-wide v2  移动上一次方法调用的long/double型返回值到v2,v3。 |
| 0C | move-result-object vx | 移动上一次方法调用的对象引用返回值到vx。 | 0C00 – move-result-object v0  移动上一次方法调用的对象引用返回值到v0。 |
| 0D | move-exception vx | 当方法调用抛出异常时移动异常对象引用到vx。 | 0D19 – move-exception v25  当方法调用抛出异常时移动异常对象引用到v25。 |
| 0E | return-void | 返回空值。 | 0E00 – return-void  返回值为void，即无返回值，并非返回null。 |
| 0F | return vx | 返回在vx寄存器的值。 | 0F00 – return v0  返回v0寄存器中的值。 |
| 10 | return-wide vx | 返回在vx,vx+1寄存器的double/long值。 | 1000 – return-wide v0  返回v0,v1寄存器中的double/long值。 |
| 11 | return-object vx | 返回在vx寄存器的对象引用。 | 1100 – return-object v0  返回v0寄存器中的对象引用。 |
| 12 | const/4 vx, lit4 | 存入4位常量到vx。 | 1221 – const/4 v1, #int 2  存入int型常量2到v1。目的寄存器在第二个字节的低4位，常量2在更高的4位。 |
| 13 | const/16 vx, lit16 | 存入16位常量到vx。 | 1300 0A00 – const/16 v0, #int 10  存入int型常量10到v0。 |
| 14 | const vx, lit32 | 存入int 型常量到vx。 | 1400 4E61 BC00 – const v0, #12345678 // #00BC614E  存入常量12345678到v0。 |
| 15 | const/high16 v0, lit16 | 存入16位常量到最高位寄存器，用于初始化float值。 | 1500 2041 – const/high16 v0, #float 10.0 // #41200000  存入float常量10.0到v0。该指令最高支持16位浮点数。 |
| 16 | const-wide/16 vx, lit16 | 存入int常量到vx,vx+1寄存器，扩展int型常量为long常量。 | 1600 0A00 – const-wide/16 v0, #long 10  存入long常量10到v0,v1寄存器。 |
| 17 | const-wide/32 vx, lit32 | 存入32位常量到vx,vx+1寄存器，扩展int型常量到long常量。 | 1702 4e61 bc00 – const-wide/32 v2, #long 12345678 // #00bc614e  存入long常量12345678到v2,v3寄存器。 |
| 18 | const-wide vx, lit64 | 存入64位常量到vx,vx+1寄存器。 | 1802 874b 6b5d 54dc 2b00- const-wide v2, #long 12345678901234567 // #002bdc545d6b4b87  存入long常量12345678901234567到v2,v3寄存器。 |
| 19 | const-wide/high16 vx, lit16 | 存入16位常量到最高16位的vx,vx+1寄存器，用于初始化double 值。 | 1900 2440 – const-wide/high16 v0, #double 10.0 // #402400000  存入double常量10.0到v0,v1。 |
| 1A | const-string vx, 字符串ID | 存入字符串常量引用到vx，通过字符串ID或字符串。 | 1A08 0000 – const-string v8, “” // string@0000  存入string@0000（字符串表#0条目）的引用到v8。 |
| 1B | const-string-jumbo | 未知注4 |  |
| 1C | const-class vx, 类型ID | 存入类对象常量到vx，通过类型ID或类型（如Object.class）。 | 1C00 0100 – const-class v0, Test3 // type@0001  存入Test3.class（类型ID表#1条目）的引用到v0。 |
| 1D | monitor-enter vx | 获得vx寄存器中的对象引用的监视器。 | 1D03 – monitor-enter v3  获得v3寄存器中的对象引用的监视器。 |
| 1E | monitor-exit | 释放vx寄存器中的对象引用的监视器。 | 1E03 – monitor-exit v3  释放v3寄存器中的对象引用的监视器。 |
| 1F | check-cast vx, 类型ID | 检查vx寄存器中的对象引用是否可以转换成类型ID对应类型的实例。如不可转换，抛出ClassCastException异常，否则继续执行。 | 1F04 0100 – check-cast v4, Test3 // type@0001  检查v4寄存器中的对象引用是否可以转换成Test3（类型ID表#1条目）的实例。 |
| 20 | instance-of vx, vy, 类型ID | 检查vy寄存器中的对象引用是否是类型ID对应类型的实例，如果是，vx存入非0值，否则vx存入0。 | 2040 0100 – instance-of v0, v4, Test3 // type@0001  检查v4寄存器中的对象引用是否是Test3（类型ID表#1条目）的实例。如果是，v0存入非0值，否则v0存入0。 |
| 21 | array-length vx, vy | 计算vy寄存器中数组引用的元素长度并将长度存入vx。 | 2111 – array-length v0, v1  计算v1寄存器中数组引用的元素长度并将长度存入v0。 |
| 22 | new-instance vx, 类型ID | 根据类型ID或类型新建一个对象实例，并将新建的对象的引用存入vx。 | 2200 1500 – new-instance v0, java.io.FileInputStream // type@0015  实例化java.io.FileInputStream（类型ID表#15H条目）类型，并将其对象引用存入v0。 |
| 23 | new-array vx, vy, 类型ID | 根据类型ID或类型新建一个数组，vy存入数组的长度，vx存入数组的引用。 | 2312 2500 – new-array v2, v1, char[] // type@0025  新建一个char（类型ID表#25H条目）数组，v1存入数组的长度，v2存入数组的引用。 |
| 24 | filled-new-array {参数}, 类型ID | 根据类型ID或类型新建一个数组并通过参数填充注5。新的数组引用可以得到一个move-result-object指令，前提是执行过filled-new-array 指令。 | 2420 530D 0000 – filled-new-array {v0,v0},[I // type@0D53  新建一个int（类型ID表#D53H条目）数组，长度将为2并且2个元素将填充到v0寄存器。 |
| 25 | filled-new-array-range {vx..vy}, 类型ID | 根据类型ID或类型新建一个数组并以寄存器范围为参数填充。新的数组引用可以得到一个move-result-object指令，前提是执行过filled-new-array 指令。 | 2503 0600 1300 - filled-new-array/range {v19..v21}, [B // type@0006  新建一个byte（类型ID表#6条目）数组，长度将为3并且3个元素将填充到v19,v20,v21寄存器注4。 |
| 26 | fill-array-data vx, 偏移量 | 用vx的静态数据填充数组引用。静态数据的位址是当前指令位置加偏移量的和。 | 2606 2500 0000 - fill-array-data v6, 00e6 // +0025  用当前指令位置+25H的静态数据填充v6寄存器的数组引用。偏移量是32位的数字，静态数据的存储格式如下：  0003 // 表类型：静态数组数据  0400 // 每个元素的字节数（这个例子是4字节的int型）  0300 0000 // 元素个数  0100 0000 // 元素 #0：int 1  0200 0000 // 元素 #1：int 2  0300 0000 // 元素 #2：int 3 |
| 27 | throw vx | 抛出异常对象，异常对象的引用在vx寄存器。 | 2700 - throw v0  抛出异常对象，异常对象的引用在v0寄存器。 |
| 28 | goto 目标 | 通过短偏移量注2无条件跳转到目标。 | 28F0 - goto 0005 // -0010  跳转到当前位置-16（hex 10）的位置，0005是目标指令标签。 |
| 29 | goto/16目标 | 通过16位偏移量注2无条件跳转到目标。 | 2900 0FFE - goto/16 002f // -01f1  跳转到当前位置-1F1H的位置，002f是目标指令标签。 |
| 2A | goto/32目标 | 通过32位偏移量注2无条件跳转到目标。 |  |
| 2B | packed-switch vx, 索引表偏移量 | 实现一个switch 语句，case常量是连续的。这个指令使用索引表，vx是在表中找到具体case的指令偏移量的索引，如果无法在表中找到vx对应的索引将继续执行下一个指令（即default case）。 | 2B02 0C00 0000 - packed-switch v2, 000c // +000c  根据v2寄存器中的值执行packed switch，索引表的位置是当前指令位置+0CH，表如下所示：  0001 // 表类型：packed switch表  0300 // 元素个数  0000 0000 // 基础元素  0500 0000 0: 00000005 // case 0: +00000005  0700 0000 1: 00000007 // case 1: +00000007  0900 0000 2: 00000009 // case 2: +00000009 |
| 2C | sparse-switch vx, 查询表偏移量 | 实现一个switch 语句，case常量是非连续的。这个指令使用查询表，用于表示case常量和每个case常量的偏移量。如果vx无法在表中匹配将继续执行下一个指令（即default case）。 | 2C02 0c00 0000 - sparse-switch v2, 000c // +000c  根据v2寄存器中的值执行sparse switch ，查询表的位置是当前指令位置+0CH，表如下所示：  0002 // 表类型：sparse switch表  0300 // 元素个数  9cff ffff // 第一个case常量: -100  fa00 0000 // 第二个case常量: 250  e803 0000 // 第三个case常量: 1000  0500 0000 // 第一个case常量的偏移量: +5  0700 0000 // 第二个case常量的偏移量: +7  0900 0000 // 第三个case常量的偏移量: +9 |
| 2D | cmpl-float vx, vy, vz | 比较vy和vz的float值并在vx存入int型返回值注3。 | 2D00 0607 - cmpl-float v0, v6, v7  比较v6和v7的float值并在v0存入int型返回值。非数值默认为小于。如果参数为非数值将返回-1。 |
| 2E | cmpg-float vx, vy, vz | 比较vy和vz的float值并在vx存入int型返回值注3。 | 2E00 0607 - cmpg-float v0, v6, v7  比较v6和v7的float值并在v0存入int型返回值。非数值默认为大于。如果参数为非数值将返回1。 |
| 2F | cmpl-double vx, vy, vz | 比较vy和vz注2的double值并在vx存入int型返回值注3。 | 2F19 0608 - cmpl-double v25, v6, v8  比较v6,v7和v8,v9的double值并在v25存入int型返回值。非数值默认为小于。如果参数为非数值将返回-1。 |
| 30 | cmpg-double vx, vy, vz | 比较vy和vz注2的double值并在vx存入int型返回值注3。 | 3000 080A - cmpg-double v0, v8, v10  比较v8,v9和v10,v11的double值并在v0存入int型返回值。非数值默认为大于。如果参数为非数值将返回1。 |
| 31 | cmp-long vx, vy, vz | 比较vy和vz的long值并在vx存入int型返回值注3。 | 3100 0204 - cmp-long v0, v2, v4  比较v2和v4的long值并在v0存入int型返回值。 |
| 32 | if-eq vx,vy, 目标 | 如果vx == vy注2，跳转到目标。vx和vy是int型值。 | 32b3 6600 - if-eq v3, v11, 0080 // +0066  如果v3 == v11，跳转到当前位置+66H。0080是目标指令标签。 |
| 33 | if-ne vx,vy, 目标 | 如果vx != vy注2，跳转到目标。vx和vy是int型值。 | 33A3 1000 - if-ne v3, v10, 002c // +0010  如果v3 != v10，跳转到当前位置+10H。002c是目标指令标签。 |
| 34 | if-lt vx,vy, 目标 | 如果vx < vy注2，跳转到目标。vx和vy是int型值。 | 3432 CBFF - if-lt v2, v3, 0023 // -0035  如果v2 < v3，跳转到当前位置-35H。0023是目标指令标签。 |
| 35 | if-ge vx, vy, 目标 | 如果vx >= vy注2，跳转到目标。vx和vy是int型值。 | 3510 1B00 - if-ge v0, v1, 002b // +001b  如果v0 >= v1，跳转到当前位置+1BH。002b是目标指令标签。 |
| 36 | if-gt vx,vy, 目标 | 如果vx > vy注2，跳转到目标。vx和vy是int型值。 | 3610 1B00 - if-ge v0, v1, 002b // +001b  如果v0 > v1，跳转到当前位置+1BH。002b是目标指令标签。 |
| 37 | if-le vx,vy, 目标 | 如果vx <= vy注2，跳转到目标。vx和vy是int型值。 | 3756 0B00 - if-le v6, v5, 0144 // +000b  如果v6 <= v5，跳转到当前位置+0BH。0144是目标指令标签。 |
| 38 | if-eqz vx, 目标 | 如果vx == 0注2，跳转到目标。vx是int型值。 | 3802 1900 - if-eqz v2, 0038 // +0019  如果v2 == 0，跳转到当前位置+19H。0038是目标指令标签。 |
| 39 | if-nez vx, 目标 | 如果vx != 0注2，跳转到目标。 | 3902 1200 - if-nez v2, 0014 // +0012  如果v2 != 0，跳转到当前位置+18(hex 12)。0014是目标指令标签。 |
| 3A | if-ltz vx, 目标 | 如果vx < 0注2，跳转到目标。 | 3A00 1600 - if-ltz v0, 002d // +0016  如果v0 < 0，跳转到当前位置+16H。002d是目标指令标签。 |
| 3B | if-gez vx, 目标 | 如果vx >= 0注2，跳转到目标。 | 3B00 1600 - if-gez v0, 002d // +0016  如果v0 >= 0，跳转到当前位置+16H。002d是目标指令标签。 |
| 3C | if-gtz vx, 目标 | 如果vx > 0注2，跳转到目标。 | 3C00 1D00 - if-gtz v0, 004a // +001d  如果v0 > 0，跳转到当前位置+1DH。004a是目标指令标签。 |
| 3D | if-lez vx, 目标 | 如果vx <= 0注2，跳转到目标。 | 3D00 1D00 - if-lez v0, 004a // +001d  如果v0 <= 0，跳转到当前位置+1DH。004a是目标指令标签。 |
| 3E | unused\_3E | 未使用 |  |
| 3F | unused\_3F | 未使用 |  |
| 40 | unused\_40 | 未使用 |  |
| 41 | unused\_41 | 未使用 |  |
| 42 | unused\_42 | 未使用 |  |
| 43 | unused\_43 | 未使用 |  |
| 44 | aget vx, vy, vz | 从int数组获取一个int型值到vx，对象数组的引用位于vy，需获取的元素的索引位于vz。 | 4407 0306 - aget v7, v3, v6  从数组获取一个int型值到v7，对象数组的引用位于v3，需获取的元素的索引位于v6。 |
| 45 | aget-wide vx, vy, vz | 从long/double数组获取一个long/double值到vx,vx+1，数组的引用位于vy，需获取的元素的索引位于vz。 | 4505 0104 - aget-wide v5, v1, v4  从long/double数组获取一个long/double值到v5,vx6，数组的引用位于v1，需获取的元素的索引位于v4。 |
| 46 | aget-object vx, vy, vz | 从对象引用数组获取一个对象引用到vx，对象数组的引用位于vy，需获取的元素的索引位于vz。 | 4602 0200 - aget-object v2, v2, v0  从对象引用数组获取一个对象引用到v2，对象数组的引用位于v2，需获取的元素的索引位于v0。 |
| 47 | aget-boolean vx, vy, vz | 从boolean数组获取一个boolean值到vx，数组的引用位于vy，需获取的元素的索引位于vz。 | 4700 0001 - aget-boolean v0, v0, v1  从boolean数组获取一个boolean值到v0，数组的引用位于v0，需获取的元素的索引位于v1。 |
| 48 | aget-byte vx, vy, vz | 从byte数组获取一个byte值到vx，数组的引用位于vy，需获取的元素的索引位于vz。 | 4800 0001 - aget-byte v0, v0, v1  从byte数组获取一个byte值到v0，数组的引用位于v0，需获取的元素的索引位于v1。 |
| 49 | aget-char vx, vy, vz | 从char数组获取一个char值到vx，数组的引用位于vy，需获取的元素的索引位于vz。 | 4905 0003 - aget-char v5, v0, v3  从char数组获取一个char值到v5，数组的引用位于v0，需获取的元素的索引位于v3。 |
| 4A | aget-short vx, vy, vz | 从short数组获取一个short值到vx，数组的引用位于vy，需获取的元素的索引位于vz。 | 4A00 0001 - aget-short v0, v0, v1  从short数组获取一个short值到v0，数组的引用位于v0，需获取的元素的索引位于v1。 |
| 4B | aput vx, vy, vz | 将vx的int值作为元素存入int数组，数组的引用位于vy，元素的索引位于vz。 | 4B00 0305 - aput v0, v3, v5  将v0的int值作为元素存入int数组，数组的引用位于v3，元素的索引位于v5。 |
| 4C | aput-wide vx, vy, vz | 将vx,vx+1的double/long值作为元素存入double/long数组，数组的引用位于vy，元素的索引位于vz。 | 4C05 0104 - aput-wide v5, v1, v4  将v5,v6的double/long值作为元素存入double/long数组，数组的引用位于v1，元素的索引位于v4。 |
| 4D | aput-object vx, vy, vz | 将vx的对象引用作为元素存入对象引用数组，数组的引用位于vy，元素的索引位于vz。 | 4D02 0100 - aput-object v2, v1, v0  将v2的对象引用作为元素存入对象引用数组，数组的引用位于v1，元素的索引位于v0。 |
| 4E | aput-boolean vx, vy, vz | 将vx的boolean值作为元素存入boolean数组，数组的引用位于vy，元素的索引位于vz。 | 4E01 0002 - aput-boolean v1, v0, v2  将v1的boolean值作为元素存入boolean数组，数组的引用位于v0，元素的索引位于v2。 |
| 4F | aput-byte vx, vy, vz | 将vx的byte值作为元素存入byte数组，数组的引用位于vy，元素的索引位于vz。 | 4F02 0001 - aput-byte v2, v0, v1  将v2的byte值作为元素存入byte数组，数组的引用位于v0，元素的索引位于v1。 |
| 50 | aput-char vx, vy, vz | 将vx的char值作为元素存入char数组，数组的引用位于vy，元素的索引位于vz。 | 5003 0001 - aput-char v3, v0, v1  将v3的char值作为元素存入char数组，数组的引用位于v0，元素的索引位于v1。 |
| 51 | aput-short vx, vy, vz | 将vx的short值作为元素存入short数组，数组的引用位于vy，元素的索引位于vz。 | 5102 0001 - aput-short v2, v0, v1  将v2的short值作为元素存入short数组，数组的引用位于v0，元素的索引位于v1。 |
| 52 | iget vx, vy, 字段ID | 根据字段ID读取实例的int型字段到vx，vy寄存器中是该实例的引用。 | 5210 0300 - iget v0, v1, Test2.i6:I // field@0003  读取int型字段i6（字段表#3条目）到v0，v1寄存器中是Test2实例的引用。 |
| 53 | iget-wide vx, vy, 字段ID | 根据字段ID读取实例的double/long型字段到vx,vx+1注1，vy寄存器中是该实例的引用。 | 5320 0400 - iget-wide v0, v2, Test2.l0:J // field@0004  读取long型字段l0（字段表#4条目）到v0,v1，v2寄存器中是Test2实例的引用。 |
| 54 | iget-object vx, vy, 字段ID | 根据字段ID读取一个实例的对象引用字段到vx，vy寄存器中是该实例的引用。 | iget-object v1, v2, LineReader.fis:Ljava/io/FileInputStream; // field@0002  读取FileInputStream对象引用字段fis（字段表#2条目）到v1，v2寄存器中是LineReader实例的引用。 |
| 55 | iget-boolean vx, vy, 字段ID | 根据字段ID读取实例的boolean型字段到vx，vy寄存器中是该实例的引用。 | 55FC 0000 - iget-boolean v12, v15, Test2.b0:Z // field@0000  读取boolean型字段b0（字段表#0条目）到v12，v15寄存器中是Test2实例的引用。 |
| 56 | iget-byte vx, vy, 字段ID | 根据字段ID读取实例的byte型字段到vx，vy寄存器中是该实例的引用。 | 5632 0100 - iget-byte v2, v3, Test3.bi1:B // field@0001  读取byte型字段bi1（字段表#1条目）到v2，v3寄存器中是Test2实例的引用。 |
| 57 | iget-char vx, vy, 字段ID | 根据字段ID读取实例的char型字段到vx，vy寄存器中是该实例的引用。 | 5720 0300 - iget-char v0, v2, Test3.ci1:C // field@0003  读取char型字段bi1（字段表#3条目）到v0，v2寄存器中是Test2实例的引用。 |
| 58 | iget-short vx, vy, 字段ID | 根据字段ID读取实例的short型字段到vx，vy寄存器中是该实例的引用。 | 5830 0800 - iget-short v0, v3, Test3.si1:S // field@0008  读取short型字段si1（字段表#8条目）到v0，v3寄存器中是Test2实例的引用。 |
| 59 | iput vx, vy, 字段ID | 根据字段ID将vx寄存器的值存入实例的int型字段，vy寄存器中是该实例的引用。 | 5920 0200 - iput v0, v2, Test2.i6:I // field@0002  将v0寄存器的值存入实例的int型字段i6（字段表#2条目），v2寄存器中是Test2实例的引用。 |
| 5A | iput-wide vx, vy, 字段ID | 根据字段ID将vx,vx+1寄存器的值存入实例的double/long型字段，vy寄存器中是该实例的引用。 | 5A20 0000 - iput-wide v0, v2, Test2.d0:D // field@0000  将v0,v1寄存器的值存入实例的double型字段d0（字段表#0条目），v2寄存器中是Test2实例的引用。 |
| 5B | iput-object vx, vy, 字段ID | 根据字段ID将vx寄存器的值存入实例的对象引用字段，vy寄存器中是该实例的引用。 | 5B20 0000 - iput-object v0, v2, LineReader.bis:Ljava/io/BufferedInputStream; // field@0000  将v0寄存器的值存入实例的对象引用字段bis（字段表#0条目），v2寄存器中是BufferedInputStream实例的引用。 |
| 5C | iput-boolean vx, vy, 字段ID | 根据字段ID将vx寄存器的值存入实例的boolean型字段，vy寄存器中是该实例的引用。 | 5C30 0000 - iput-boolean v0, v3, Test2.b0:Z // field@0000  将v0寄存器的值存入实例的boolean型字段b0（字段表#0条目），v3寄存器中是Test2实例的引用。 |
| 5D | iput-byte vx, vy, 字段ID | 根据字段ID将vx寄存器的值存入实例的byte型字段，vy寄存器中是该实例的引用。 | 5D20 0100 - iput-byte v0, v2, Test3.bi1:B // field@0001  将v0寄存器的值存入实例的byte型字段bi1（字段表#1条目），v2寄存器中是Test2实例的引用。 |
| 5E | iput-char vx, vy, 字段ID | 根据字段ID将vx寄存器的值存入实例的char型字段，vy寄存器中是该实例的引用。 | 5E20 0300 - iput-char v0, v2, Test3.ci1:C // field@0003  将v0寄存器的值存入实例的char型字段ci1（字段表#3条目），v2寄存器中是Test2实例的引用。 |
| 5F | iput-short vx, vy, 字段ID | 根据字段ID将vx寄存器的值存入实例的short型字段，vy寄存器中是该实例的引用。 | 5F21 0800 - iput-short v1, v2, Test3.si1:S // field@0008  将v0寄存器的值存入实例的short型字段si1（字段表#8条目），v2寄存器中是Test2实例的引用。 |
| 60 | sget vx, 字段ID | 根据字段ID读取静态int型字段到vx。 | 6000 0700 - sget v0, Test3.is1:I // field@0007  读取Test3的静态int型字段is1（字段表#7条目）到v0。 |
| 61 | sget-wide vx, 字段ID | 根据字段ID读取静态double/long型字段到vx,vx+1。 | 6100 0500 - sget-wide v0, Test2.l1:J // field@0005  读取Test2的静态long型字段l1（字段表#5条目）到v0,v1。 |
| 62 | sget-object vx, 字段ID | 根据字段ID读取静态对象引用字段到vx。 | 6201 0C00 - sget-object v1, Test3.os1:Ljava/lang/Object; // field@000c  读取Object的静态对象引用字段os1（字段表#CH条目）到v1。 |
| 63 | sget-boolean vx, 字段ID | 根据字段ID读取静态boolean型字段到vx。 | 6300 0C00 - sget-boolean v0, Test2.sb:Z // field@000c  读取Test2的静态boolean型字段sb（字段表#CH条目）到v0。 |
| 64 | sget-byte vx, 字段ID | 根据字段ID读取静态byte型字段到vx。 | 6400 0200 - sget-byte v0, Test3.bs1:B // field@0002  读取Test3的静态byte型字段bs1（字段表#2条目）到v0。 |
| 65 | sget-char vx, 字段ID | 根据字段ID读取静态char型字段到vx。 | 6500 0700 - sget-char v0, Test3.cs1:C // field@0007  读取Test3的静态char型字段cs1（字段表#7条目）到v0。 |
| 66 | sget-short vx, 字段ID | 根据字段ID读取静态short型字段到vx。 | 6600 0B00 - sget-short v0, Test3.ss1:S // field@000b  读取Test3的静态short型字段ss1（字段表#CH条目）到v0。 |
| 67 | sput vx, 字段ID | 根据字段ID将vx寄存器中的值赋值到int型静态字段。 | 6700 0100 - sput v0, Test2.i5:I // field@0001  将v0寄存器中的值赋值到Test2的int型静态字段i5（字段表#1条目）。 |
| 68 | sput-wide vx, 字段ID | 根据字段ID将vx,vx+1寄存器中的值赋值到double/long型静态字段。 | 6800 0500 - sput-wide v0, Test2.l1:J // field@0005  将v0,v1寄存器中的值赋值到Test2的long型静态字段l1（字段表#5条目）。 |
| 69 | sput-object vx, 字段ID | 根据字段ID将vx寄存器中的对象引用赋值到对象引用静态字段。 | 6900 0c00 - sput-object v0, Test3.os1:Ljava/lang/Object; // field@000c  将v0寄存器中的对象引用赋值到Test3的对象引用静态字段os1（字段表#CH条目）。 |
| 6A | sput-boolean vx, 字段ID | 根据字段ID将vx寄存器中的值赋值到boolean型静态字段。 | 6A00 0300 - sput-boolean v0, Test3.bls1:Z // field@0003  将v0寄存器中的值赋值到Test3的boolean型静态字段bls1（字段表#3条目）。 |
| 6B | sput-byte vx, 字段ID | 根据字段ID将vx寄存器中的值赋值到byte型静态字段。 | 6B00 0200 - sput-byte v0, Test3.bs1:B // field@0002  将v0寄存器中的值赋值到Test3的byte型静态字段bs1（字段表#2条目）。 |
| 6C | sput-char vx, 字段ID | 根据字段ID将vx寄存器中的值赋值到char型静态字段。 | 6C01 0700 - sput-char v1, Test3.cs1:C // field@0007  将v1寄存器中的值赋值到Test3的char型静态字段cs1（字段表#7条目）。 |
| 6D | sput-short vx, 字段ID | 根据字段ID将vx寄存器中的值赋值到short型静态字段。 | 6D00 0B00 - sput-short v0, Test3.ss1:S // field@000b  将v0寄存器中的值赋值到Test3的short型静态字段ss1（字段表#BH条目）。 |
| 6E | invoke-virtual {参数}, 方法名 | 调用带参数的虚拟方法。 | 6E53 0600 0421 - invoke-virtual { v4, v0, v1, v2, v3}, Test2.method5:(IIII)V // method@0006  调用Test2的method5（方法表#6条目）方法，该指令共有5个参数（操作码第二个字节的4个最高有效位5）注5。参数v4是"this"实例，v0, v1, v2, v3是method5方法的参数，(IIII)V的4个I分表表示4个int型参数，V表示返回值为void。 |
| 6F | invoke-super {参数}, 方法名 | 调用带参数的直接父类的虚拟方法。 | 6F10 A601 0100 invoke-super {v1},java.io.FilterOutputStream.close:()V // method@01a6  调用java.io.FilterOutputStream的close（方法表#1A6条目）方法，参数v1是"this"实例。()V表示close方法没有参数，V表示返回值为void。 |
| 70 | invoke-direct {参数}, 方法名 | 不解析直接调用带参数的方法。 | 7010 0800 0100 - invoke-direct {v1}, java.lang.Object.<init>:()V // method@0008  调用java.lang.Object 的<init>（方法表#8条目）方法，参数v1是"this"实例注5。()V表示<init>方法没有参数，V表示返回值为void。 |
| 71 | invoke-static {参数}, 方法名 | 调用带参数的静态方法。 | 7110 3400 0400 - invoke-static {v4}, java.lang.Integer.parseInt:( Ljava/lang/String;)I // method@0034  调用java.lang.Integer 的parseInt（方法表#34条目）静态方法，该指令只有1个参数v4注5，(Ljava/lang/String;)I中的Ljava/lang/String;表示parseInt方法需要String类型的参数，I表示返回值为int型。 |
| 72 | invoke-interface {参数}, 方法名 | 调用带参数的接口方法。 | 7240 2102 3154 invoke-interface {v1, v3, v4, v5}, mwfw.IReceivingProtocolAdapter.receivePackage:(ILjava/lang/String;Ljava/io/InputStream;)Z // method@0221  调用mwfw.IReceivingProtocolAdapter 接口的receivePackage方法（方法表#221条目），该指令共有4个参数注5，参数v1是"this"实例，v3,v4,v5是receivePackage方法的参数，(ILjava/lang/String;Ljava/io/InputStream;)Z中的I表示int型参数，Ljava/lang/String;表示String类型参数，Ljava/io/InputStream;表示InputStream类型参数，Z表示返回值为boolean型。 |
| 73 | unused\_73 | 未使用 |  |
| 74 | invoke-virtual/range {vx..vy}, 方法名 | 调用以寄存器范围为参数的虚拟方法。该指令第一个寄存器和寄存器的数量将传递给方法。 | 7403 0600 1300 - invoke-virtual {v19..v21}, Test2.method5:(IIII)V // method@0006  调用Test2的method5（方法表#6条目）方法，该指令共有3个参数。参数v19是"this"实例，v20,v21是method5方法的参数，(IIII)V的4个I分表表示4个int型参数，V表示返回值为void。 |
| 75 | invoke-super/range {vx..vy}, 方法名 | 调用以寄存器范围为参数的直接父类的虚拟方法。该指令第一个寄存器和寄存器的数量将会传递给方法。 | 7501 A601 0100 invoke-super {v1},java.io.FilterOutputStream.close:()V // method@01a6  调用java.io.FilterOutputStream的close（方法表#1A6条目）方法，参数v1是"this"实例。()V表示close方法没有参数，V表示返回值为void。 |
| 76 | invoke-direct/range {vx..vy}, 方法名 | 不解析直接调用以寄存器范围为参数的方法。该指令第一个寄存器和寄存器的数量将会传递给方法。 | 7603 3A00 1300 - invoke-direct/range {v19..21},java.lang.Object.<init>:()V // method@003a  调用java.lang.Object 的<init>（方法表#3A条目）方法，参数v19是"this"实例（操作码第五、第六字节表示范围从v19开始，第二个字节为03表示传入了3个参数），()V表示<init>方法没有参数，V表示返回值为void。 |
| 77 | invoke-static/range {vx..vy}, 方法名 | 调用以寄存器范围为参数的静态方法。该指令第一个寄存器和寄存器的数量将会传递给方法。 | 7703 3A00 1300 - invoke-static/range {v19..21},java.lang.Integer.parseInt:(Ljava/lang/String;)I // method@0034  调用java.lang.Integer 的parseInt（方法表#34条目）静态方法，参数v19是"this"实例（操作码第五、第六字节表示范围从v19开始，第二个字节为03表示传入了3个参数），(Ljava/lang/String;)I中的Ljava/lang/String;表示parseInt方法需要String类型的参数，I表示返回值为int型。 |
| 78 | invoke-interface-range {vx..vy}, 方法名 | 调用以寄存器范围为参数的接口方法。该指令第一个寄存器和寄存器的数量将会传递给方法。 | 7840 2102 0100 invoke-interface {v1..v4}, mwfw.IReceivingProtocolAdapter.receivePackage:(ILjava/lang/String;Ljava/io/InputStream;)Z // method@0221  调用mwfw.IReceivingProtocolAdapter 接口的receivePackage方法（方法表#221条目），该指令共有4个参数注5，参数v1是"this"实例，v2,v3,v4是receivePackage方法的参数，(ILjava/lang/String;Ljava/io/InputStream;)Z中的I表示int型参数，Ljava/lang/String;表示String类型参数，Ljava/io/InputStream;表示InputStream类型参数，Z表示返回值为boolean型。 |
| 79 | unused\_79 | 未使用 |  |
| 7A | unused\_7A | 未使用 |  |
| 7B | neg-int vx, vy | 计算vx = -vy并将结果存入vx。 | 7B01 - neg-int v1,v0  计算-v0并将结果存入v1。 |
| 7C | not-int vx, vy | 未知注4 |  |
| 7D | neg-long vx, vy | 计算vx,vx+1 = -(vy,vy+1) 并将结果存入vx,vx+1。 | 7D02 - neg-long v2,v0  计算-(v0,v1) 并将结果存入(v2,v3)。 |
| 7E | not-long vx, vy | 未知注4 |  |
| 7F | neg-float vx, vy | 计算vx = -vy并将结果存入vx。 | 7F01 - neg-float v1,v0  计算-v0并将结果存入v1。 |
| 80 | neg-double vx, vy | 计算vx,vx+1=-(vy,vy+1) 并将结果存入vx,vx+1。 | 8002 - neg-double v2,v0  计算-(v0,v1) 并将结果存入(v2,v3)。 |
| 81 | int-to-long vx, vy | 转换vy寄存器中的int型值为long型值存入vx,vx+1。 | 8106 - int-to-long v6, v0  转换v0寄存器中的int型值为long型值存入v6,v7。 |
| 82 | int-to-float vx, vy | 转换vy寄存器中的int型值为float型值存入vx。 | 8206 - int-to-float v6, v0  转换v0寄存器中的int型值为float型值存入v6。 |
| 83 | int-to-double vx, vy | 转换vy寄存器中的int型值为double型值存入vx,vx+1。 | 8306 - int-to-double v6, v0  转换v0寄存器中的int型值为double型值存入v6,v7。 |
| 84 | long-to-int vx, vy | 转换vy,vy+1寄存器中的long型值为int型值存入vx。 | 8424 - long-to-int v4, v2  转换v2,v3寄存器中的long型值为int型值存入v4。 |
| 85 | long-to-float vx, vy | 转换vy,vy+1寄存器中的long型值为float型值存入vx。 | 8510 - long-to-float v0, v1  转换v1,v2寄存器中的long型值为float型值存入v0。 |
| 86 | long-to-double vx, vy | 转换vy,vy+1寄存器中的long型值为double型值存入vx,vx+1。 | 8610 - long-to-double v0, v1  转换v1,vy2寄存器中的long型值为double型值存入v0,v1。 |
| 87 | float-to-int vx, vy | 转换vy寄存器中的float型值为int型值存入vx。 | 8730 - float-to-int v0, v3  转换v3寄存器中的float型值为int型值存入v0。 |
| 88 | float-to-long vx, vy | 转换vy寄存器中的float型值为long型值存入vx,vx+1。 | 8830 - float-to-long v0, v3  转换v3寄存器中的float型值为long型值存入v0,v1。 |
| 89 | float-to-double vx, vy | 转换vy寄存器中的float型值为double型值存入vx,vx+1。 | 8930 - float-to-double v0, v3  转换v3寄存器中的float型值为double型值存入v0,v1。 |
| 8A | double-to-int vx, vy | 转换vy,vy+1寄存器中的double型值为int型值存入vx。 | 8A40 - double-to-int v0, v4  转换v4,v5寄存器中的double型值为int型值存入v0。 |
| 8B | double-to-long vx, vy | 转换vy,vy+1寄存器中的double型值为long型值存入vx,vx+1。 | 8B40 - double-to-long v0, v4  转换v4,v5寄存器中的double型值为long型值存入v0,v1。 |
| 8C | double-to-float vx, vy | 转换vy,vy+1寄存器中的double型值为float型值存入vx。 | 8C40 - double-to-float v0, v4  转换v4,v5寄存器中的double型值为float型值存入v0。 |
| 8D | int-to-byte vx, vy | 转换vy寄存器中的int型值为byte型值存入vx。 | 8D00 - int-to-byte v0, v0  转换v0寄存器中的int型值为byte型值存入v0。 |
| 8E | int-to-char vx, vy | 转换vy寄存器中的int型值为char型值存入vx。 | 8E33 - int-to-char v3, v3  转换v3寄存器中的int型值为char型值存入v3。 |
| 8F | int-to-short vx, vy | 转换vy寄存器中的int型值为short型值存入vx。 | 8F00 - int-to-short v3, v0  转换v0寄存器中的int型值为short型值存入v0。 |
| 90 | add-int vx, vy, vz | 计算vy + vz并将结果存入vx。 | 9000 0203 - add-int v0, v2, v3  计算v2 + v3并将结果存入v0注4。 |
| 91 | sub-int vx, vy, vz | 计算vy - vz并将结果存入vx。 | 9100 0203 - sub-int v0, v2, v3  计算v2 – v3并将结果存入v0。 |
| 92 | mul-int vx, vy, vz | 计算vy \* vz并将结果存入vx。 | 9200 0203 - mul-int v0,v2,v3  计算v2 \* w3并将结果存入v0。 |
| 93 | div-int vx, vy, vz | 计算vy / vz并将结果存入vx。 | 9303 0001 - div-int v3, v0, v1  计算v0 / v1并将结果存入v3。 |
| 94 | rem-int vx, vy, vz | 计算vy % vz并将结果存入vx。 | 9400 0203 - rem-int v0, v2, v3  计算v3 % v2并将结果存入v0。 |
| 95 | and-int vx, vy, vz | 计算vy 与 vz并将结果存入vx。 | 9503 0001 - and-int v3, v0, v1  计算v0 与 v1并将结果存入v3。 |
| 96 | or-int vx, vy, vz | 计算vy 或 vz并将结果存入vx。 | 9603 0001 - or-int v3, v0, v1  计算v0 或 v1并将结果存入v3。 |
| 97 | xor-int vx, vy, vz | 计算vy 异或 vz并将结果存入vx。 | 9703 0001 - xor-int v3, v0, v1  计算v0 异或 v1并将结果存入v3。 |
| 98 | shl-int vx, vy, vz | 左移vy，vz指定移动的位置，结果存入vx。 | 9802 0001 - shl-int v2, v0, v1  以v1指定的位置左移v0，结果存入v2。 |
| 99 | shr-int vx, vy, vz | 右移vy，vz指定移动的位置，结果存入vx。 | 9902 0001 - shr-int v2, v0, v1  以v1指定的位置右移v0，结果存入v2。 |
| 9A | ushr-int vx, vy, vz | 无符号右移vy，vz指定移动的位置，结果存入vx。 | 9A02 0001 - ushr-int v2, v0, v1  以v1指定的位置无符号右移v0，结果存入v2。 |
| 9B | add-long vx, vy, vz | 计算vy,vy+1 + vz,vz+1并将结果存入vx,vx+1注1。 | 9B00 0305 - add-long v0, v3, v5  计算v3,v4 + v5,v6并将结果存入v0,v1。 |
| 9C | sub-long vx, vy, vz | 计算vy,vy+1 - vz,vz+1并将结果存入vx,vx+1注1。 | 9C00 0305 - sub-long v0, v3, v5  计算v3,v4 - v5,v6并将结果存入v0,v1。 |
| 9D | mul-long vx, vy, vz | 计算vy,vy+1 \* vz,vz+1并将结果存入vx,vx+1注1。 | 9D00 0305 - mul-long v0, v3, v5  计算v3,v4 \* v5,v6并将结果存入v0,v1。 |
| 9E | div-long vx, vy, vz | 计算vy,vy+1 / vz,vz+1并将结果存入vx,vx+1注1。 | 9E06 0002 - div-long v6, v0, v2  计算v0,v1 / v2,v3并将结果存入v6,v7。 |
| 9F | rem-long vx, vy, vz | 计算vy,vy+1 % vz,vz+1并将结果存入vx,vx+1注1。 | 9F06 0002 - rem-long v6, v0, v2  计算v0,v1 % v2,v3并将结果存入v6,v7。 |
| A0 | and-long vx, vy, vz | 计算vy,vy+1 与 vz,vz+1并将结果存入vx,vx+1注1。 | A006 0002 - and-long v6, v0, v2  计算v0,v1 与 v2,v3并将结果存入v6,v7。 |
| A1 | or-long vx, vy, vz | 计算vy,vy+1 或 vz,vz+1并将结果存入vx,vx+1注1。 | A106 0002 - or-long v6, v0, v2  计算v0,v1 或 v2,v3并将结果存入v6,v7。 |
| A2 | xor-long vx, vy, vz | 计算vy,vy+1 异或 vz,vz+1并将结果存入vx,vx+1注1。 | A206 0002 - xor-long v6, v0, v2  计算v0,v1 异或 v2,v3并将结果存入v6,v7。 |
| A3 | shl-long vx, vy, vz | 左移vy,vy+1，vz指定移动的位置，结果存入vx,vx+1注1。 | A302 0004 - shl-long v2, v0, v4  以v4指定的位置左移v0,v1，结果存入v2,v3。 |
| A4 | shr-long vx, vy, vz | 右移vy,vy+1，vz指定移动的位置，结果存入vx,vx+1注1。 | A402 0004 - shr-long v2, v0, v4  以v4指定的位置右移v0,v1，结果存入v2,v3。 |
| A5 | ushr-long vx, vy, vz | 无符号右移vy,vy+1，vz指定移动的位置，结果存入vx,vx+1注1。 | A502 0004 - ushr-long v2, v0, v4  以v4指定的位置无符号右移v0,v1，结果存入v2,v3。 |
| A6 | add-float vx, vy, vz | 计算vy + vz并将结果存入vx。 | A600 0203 - add-float v0, v2, v3  计算v2 + v3并将结果存入v0。 |
| A7 | sub-float vx, vy, vz | 计算vy - vz并将结果存入vx。 | A700 0203 - sub-float v0, v2, v3  计算v2 - v3并将结果存入v0。 |
| A8 | mul-float vx, vy, vz | 计算vy \* vz并将结果存入vx。 | A803 0001 - mul-float v3, v0, v1  计算v0 \* v1并将结果存入v3。 |
| A9 | div-float vx, vy, vz | 计算vy / vz并将结果存入vx。 | A903 0001 - div-float v3, v0, v1  计算v0 / v1并将结果存入v3。 |
| AA | rem-float vx, vy, vz | 计算vy % vz并将结果存入vx。 | AA03 0001 - rem-float v3, v0, v1  计算v0 % v1并将结果存入v3。 |
| AB | add-double vx, vy, vz | 计算vy,vy+1 + vz,vz+1并将结果存入vx,vx+1注1。 | AB00 0305 - add-double v0, v3, v5  计算v3,v4 + v5,v6并将结果存入v0,v1。 |
| AC | sub-double vx, vy, vz | 计算vy,vy+1 - vz,vz+1并将结果存入vx,vx+1注1。 | AC00 0305 - sub-double v0, v3, v5  计算v3,v4 - v5,v6并将结果存入v0,v1。 |
| AD | mul-double vx, vy, vz | 计算vy,vy+1 \* vz,vz+1并将结果存入vx,vx+1注1。 | AD06 0002 - mul-double v6, v0, v2  计算v0,v1 \* v2,v3并将结果存入v6,v7。 |
| AE | div-double vx, vy, vz | 计算vy,vy+1 / vz,vz+1并将结果存入vx,vx+1注1。 | AE06 0002 - div-double v6, v0, v2  计算v0,v1 / v2,v3并将结果存入v6,v7。 |
| AF | rem-double vx, vy, vz | 计算vy,vy+1 % vz,vz+1并将结果存入vx,vx+1注1。 | AF06 0002 - rem-double v6, v0, v2  计算v0,v1 % v2,v3并将结果存入v6,v7。 |
| B0 | add-int/2addr vx, vy | 计算vx + vy并将结果存入vx。 | B010 - add-int/2addr v0,v1  计算v0 + v1并将结果存入v0。 |
| B1 | sub-int/2addr vx, vy | 计算vx - vy并将结果存入vx。 | B140 - sub-int/2addr v0, v4  计算v0 – v4并将结果存入v0。 |
| B2 | mul-int/2addr vx, vy | 计算vx \* vy并将结果存入vx。 | B210 - mul-int/2addr v0, v1  计算v0 \* v1并将结果存入v0。 |
| B3 | div-int/2addr vx, vy | 计算vx / vy并将结果存入vx。 | B310 - div-int/2addr v0, v1  计算v0 / v1并将结果存入v0。 |
| B4 | rem-int/2addr vx, vy | 计算vx % vy并将结果存入vx。 | B410 - rem-int/2addr v0, v1  计算v0 % v1并将结果存入v0。 |
| B5 | and-int/2addr vx, vy | 计算vx 与 vy并将结果存入vx。 | B510 - and-int/2addr v0, v1  计算v0 与 v1并将结果存入v0。 |
| B6 | or-int/2addr vx, vy | 计算vx 或 vy并将结果存入vx。 | B610 - or-int/2addr v0, v1  计算v0 或 v1并将结果存入v0。 |
| B7 | xor-int/2addr vx, vy | 计算vx 异或 vy并将结果存入vx。 | B710 - xor-int/2addr v0, v1  计算v0 异或 v1并将结果存入v0。 |
| B8 | shl-int/2addr vx, vy | 左移vx，vy指定移动的位置，并将结果存入vx。 | B810 - shl-int/2addr v0, v1  以v1指定的位置左移v0，结果存入v0。 |
| B9 | shr-int/2addr vx, vy | 右移vx，vy指定移动的位置，并将结果存入vx。 | B910 - shr-int/2addr v0, v1  以v1指定的位置右移v0，结果存入v0。 |
| BA | ushr-int/2addr vx, vy | 无符号右移vx，vy指定移动的位置，并将结果存入vx。 | BA10 - ushr-int/2addr v0, v1  以v1指定的位置无符号右移v0，结果存入v0。 |
| BB | add-long/2addr vx, vy | 计算vx,vx+1 + vy,vy+1并将结果存入vx,vx+1注1。 | BB20 - add-long/2addr v0, v2  计算v0,v1 + v2,v3并将结果存入v0,v1。 |
| BC | sub-long/2addr vx, vy | 计算vx,vx+1 - vy,vy+1并将结果存入vx,vx+1注1。 | BC70 - sub-long/2addr v0, v7  计算v0,v1 - v7,v8并将结果存入v0,v1。 |
| BD | mul-long/2addr vx, vy | 计算vx,vx+1 \* vy,vy+1并将结果存入vx,vx+1注1。 | BD70 - mul-long/2addr v0, v7  计算v0,v1 \* v7,v8并将结果存入v0,v1。 |
| BE | div-long/2addr vx, vy | 计算vx,vx+1 / vy,vy+1并将结果存入vx,vx+1注1。 | BE20 - div-long/2addr v0, v2  计算v0,v1 / v2,v3并将结果存入v0,v1。 |
| BF | rem-long/2addr vx, vy | 计算vx,vx+1 % vy,vy+1并将结果存入vx,vx+1注1。 | BF20 - rem-long/2addr v0, v2  计算v0,v1 % v2,v3并将结果存入v0,v1。 |
| C0 | and-long/2addr vx, vy | 计算vx,vx+1 与 vy,vy+1并将结果存入vx,vx+1注1。 | C020 - and-long/2addr v0, v2  计算v0,v1 与 v2,v3并将结果存入v0,v1。 |
| C1 | or-long/2addr vx, vy | 计算vx,vx+1 或 vy,vy+1并将结果存入vx,vx+1注1。 | C120 - or-long/2addr v0, v2  计算v0,v1 或 v2,v3并将结果存入v0,v1。 |
| C2 | xor-long/2addr vx, vy | 计算vx,vx+1 异或 vy,vy+1并将结果存入vx,vx+1注1。 | C220 - xor-long/2addr v0, v2  计算v0,v1 异或 v2,v3并将结果存入v0,v1。 |
| C3 | shl-long/2addr vx, vy | 左移vx,vx+1，vy指定移动的位置，并将结果存入vx,vx+1。 | C320 - shl-long/2addr v0, v2  以v2指定的位置左移v0,v1，结果存入v0,v1。 |
| C4 | shr-long/2addr vx, vy | 右移vx,vx+1，vy指定移动的位置，并将结果存入vx,vx+1。 | C420 - shr-long/2addr v0, v2  以v2指定的位置右移v0,v1，结果存入v0,v1。 |
| C5 | ushr-long/2addr vx, vy | 无符号右移vx,vx+1，vy指定移动的位置，并将结果存入vx,vx+1。 | C520 - ushr-long/2addr v0, v2  以v2指定的位置无符号右移v0,v1，结果存入v0,v1。 |
| C6 | add-float/2addr vx, vy | 计算vx + vy并将结果存入vx。 | C640 - add-float/2addr v0,v4  计算v0 + v4并将结果存入v0。 |
| C7 | sub-float/2addr vx, vy | 计算vx - vy并将结果存入vx。 | C740 - sub-float/2addr v0,v4  计算v0 - v4并将结果存入v0。 |
| C8 | mul-float/2addr vx, vy | 计算vx \* vy并将结果存入vx。 | C810 - mul-float/2addr v0, v1  计算v0 \* v1并将结果存入v0。 |
| C9 | div-float/2addr vx, vy | 计算vx / vy并将结果存入vx。 | C910 - div-float/2addr v0, v1  计算v0 / v1并将结果存入v0。 |
| CA | rem-float/2addr vx, vy | 计算vx % vy并将结果存入vx。 | CA10 - rem-float/2addr v0, v1  计算v0 % v1并将结果存入v0。 |
| CB | add-double/2addr vx, vy | 计算vx,vx+1 + vy,vy+1并将结果存入vx,vx+1注1。 | CB70 - add-double/2addr v0, v7  计算v0,v1 + v7,v8并将结果存入v0,v1。 |
| CC | sub-double/2addr vx, vy | 计算vx,vx+1 - vy,vy+1并将结果存入vx,vx+1注1。 | CC70 - sub-double/2addr v0, v7  计算v0,v1 - v7,v8并将结果存入v0,v1。 |
| CD | mul-double/2addr vx, vy | 计算vx,vx+1 \* vy,vy+1并将结果存入vx,vx+1注1。 | CD20 - mul-double/2addr v0, v2  计算v0,v1 \* v2,v3并将结果存入v0,v1。 |
| CE | div-double/2addr vx, vy | 计算vx,vx+1 / vy,vy+1并将结果存入vx,vx+1注1。 | CE20 - div-double/2addr v0, v2  计算v0,v1 / v2,v3并将结果存入v0,v1。 |
| CF | rem-double/2addr vx, vy | 计算vx,vx+1 % vy,vy+1并将结果存入vx,vx+1注1。 | CF20 - rem-double/2addr v0, v2  计算v0,v1 % v2,v3并将结果存入v0,v1。 |
| D0 | add-int/lit16 vx, vy, lit16 | 计算vy + lit16并将结果存入vx。 | D001 D204 - add-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 + 1234并将结果存入v1。 |
| D1 | sub-int/lit16 vx, vy, lit16 | 计算vy - lit16并将结果存入vx。 | D101 D204 - sub-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 - 1234并将结果存入v1。 |
| D2 | mul-int/lit16 vx, vy, lit16 | 计算vy \* lit16并将结果存入vx。 | D201 D204 - mul-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 \* 1234并将结果存入v1。 |
| D3 | div-int/lit16 vx, vy, lit16 | 计算vy / lit16并将结果存入vx。 | D301 D204 - div-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 / 1234并将结果存入v1。 |
| D4 | rem-int/lit16 vx, vy, lit16 | 计算vy % lit16并将结果存入vx。 | D401 D204 - rem-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 % 1234并将结果存入v1。 |
| D5 | and-int/lit16 vx, vy, lit16 | 计算vy 与 lit16并将结果存入vx。 | D501 D204 - and-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 与 1234并将结果存入v1。 |
| D6 | or-int/lit16 vx, vy, lit16 | 计算vy 或 lit16并将结果存入vx。 | D601 D204 - or-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 或 1234并将结果存入v1。 |
| D7 | xor-int/lit16 vx, vy, lit16 | 计算vy 异或 lit16并将结果存入vx。 | D701 D204 - xor-int/lit16 v1, v0, #int 1234 // #04d2  计算v0 异或 1234并将结果存入v1。 |
| D8 | add-int/lit8 vx, vy, lit8 | 计算vy + lit8并将结果存入vx。 | D800 0201 - add-int/lit8 v0,v2, #int1  计算v2 + 1并将结果存入v0。 |
| D9 | sub-int/lit8 vx, vy, lit8 | 计算vy - lit8并将结果存入vx。 | D900 0201 - sub-int/lit8 v0,v2, #int1  计算v2 - 1并将结果存入v0。 |
| DA | mul-int/lit8 vx, vy, lit8 | 计算vy \* lit8并将结果存入vx。 | DA00 0002 - mul-int/lit8 v0,v0, #int2  计算v0 \* 2并将结果存入v0。 |
| DB | div-int/lit8 vx, vy, lit8 | 计算vy / lit8并将结果存入vx。 | DB00 0203 - mul-int/lit8 v0,v2, #int3  计算v2 / 3并将结果存入v0。 |
| DC | rem-int/lit8 vx, vy, lit8 | 计算vy % lit8并将结果存入vx。 | DC00 0203 - rem-int/lit8 v0,v2, #int3  计算v2 % 3并将结果存入v0。 |
| DD | and-int/lit8 vx, vy, lit8 | 计算vy 与 lit8并将结果存入vx。 | DD00 0203 - and-int/lit8 v0,v2, #int3  计算v2 与 3并将结果存入v0。 |
| DE | or-int/lit8 vx, vy, lit8 | 计算vy 或 lit8并将结果存入vx。 | DE00 0203 - or-int/lit8 v0, v2, #int 3  计算v2 或 3并将结果存入v0。 |
| DF | xor-int/lit8 vx, vy, lit8 | 计算vy异或lit8并将结果存入vx。 | DF00 0203 | 0008: xor-int/lit8 v0, v2, #int 3  计算v2 异或 3并将结果存入v0。 |
| E0 | shl-int/lit8 vx, vy, lit8 | 左移vy，lit8指定移动的位置，并将结果存入vx。 | E001 0001 - shl-int/lit8 v1, v0, #int 1  将v0左移1位，结果存入v1。 |
| E1 | shr-int/lit8 vx, vy, lit8 | 右移vy，lit8指定移动的位置，并将结果存入vx。 | E101 0001 - shr-int/lit8 v1, v0, #int 1  将v0右移1位，结果存入v1。 |
| E2 | ushr-int/lit8 vx, vy, lit8 | 无符号右移vy，lit8指定移动的位置，并将结果存入vx。 | E201 0001 - ushr-int/lit8 v1, v0, #int 1  将v0无符号右移1位，结果存入v1。 |
| E3 | unused\_E3 | 未使用 |  |
| E4 | unused\_E4 | 未使用 |  |
| E5 | unused\_E5 | 未使用 |  |
| E6 | unused\_E6 | 未使用 |  |
| E7 | unused\_E7 | 未使用 |  |
| E8 | unused\_E8 | 未使用 |  |
| E9 | unused\_E9 | 未使用 |  |
| EA | unused\_EA | 未使用 |  |
| EB | unused\_EB | 未使用 |  |
| EC | unused\_EC | 未使用 |  |
| ED | unused\_ED | 未使用 |  |
| EE | execute-inline {参数}, 内联ID | 根据内联ID注6执行内联方法。 | EE20 0300 0100 - execute-inline {v1, v0}, inline #0003  执行内联方法#3，参数v1,v0，其中参数v1为"this"的实例，v0是方法的参数。 |
| EF | unused\_EF | 未使用 |  |
| F0 | invoke-direct-empty | 用于空方法的占位符，如Object.<init>。这相当于正常执行了nop指令注6。 | F010 F608 0000 - invoke-direct-empty {v0}, Ljava/lang/Object;.<init>:()V // method@08f6  替代空方法java/lang/Object;<init>。 |
| F1 | unused\_F1 | 未使用 |  |
| F2 | iget-quick vx, vy, 偏移量 | 获取vy寄存器中实例指向+偏移位置的数据区的值，存入vx注6。 | F221 1000 - iget-quick v1, v2, [obj+0010]  获取v2寄存器中的实例指向+10H位置的数据区的值，存入v1。 |
| F3 | iget-wide-quick vx, vy, 偏移量 | 获取vy寄存器中实例指向+偏移位置的数据区的值，存入vx,vx+1注6。 | F364 3001 – iget-wide-quick v4, v6, [obj+0130]  获取v6寄存器中的实例指向+130H位置的数据区的值，存入v4,v5。 |
| F4 | iget-object-quick vx, vy, 偏移量 | 获取vy寄存器中实例指向+偏移位置的数据区的对象引用，存入vx注6。 | F431 0C00 – iget-object-quick v1, v3, [obj+000c]  获取v3寄存器中的实例指向+0CH位置的数据区的对象引用，存入v1。 |
| F5 | iput-quick vx, vy, 偏移量 | 将vx寄存器中的值存入vy寄存器中的实例指向+偏移位置的数据区注6。 | F521 1000 – iput-quick v1, v2, [obj+0010]  将v1寄存器中的值存入v2寄存器中的实例指向+10H位置的数据区。 |
| F6 | iput-wide-quick vx, vy, 偏移量 | 将vx,vx+1寄存器中的值存入vy寄存器中的实例指向+偏移位置的数据区注6。 | F652 7001 – iput-wide-quick v2, v5, [obj+0170]  将v2,v3寄存器中的值存入v5寄存器中的实例指向+170H位置的数据区。 |
| F7 | iput-object-quick vx, vy, 偏移量 | 将vx寄存器中的对象引用存入vy寄存器中的实例指向+偏移位置的数据区注6。 | F701 4C00 – iput-object-quick v1, v0, [obj+004c]  将v1寄存器中的对象引用存入v0寄存器中的实例指向+4CH位置的数据区。 |
| F8 | invoke-virtual-quick {参数}, 虚拟表偏移量 | 调用虚拟方法，使用目标对象虚拟表注6。 | F820 B800 CF00 – invoke-virtual-quick {v15, v12}, vtable #00b8  调用虚拟方法，目标对象的实例指向位于v15寄存器，方法位于虚拟表#B8条目，方法所需的参数位于v12。 |
| F9 | invoke-virtual-quick/range {参数范围}, 虚拟表偏移量 | 调用虚拟方法，使用目标对象虚拟表注6。 | F906 1800 0000 – invoke-virtual-quick/range {v0..v5},vtable #0018  调用虚拟方法，目标对象的实例指向位于v0寄存器，方法位于虚拟表#18H条目，方法所需的参数位于v1..v5。 |
| FA | invoke-super-quick {参数}, 虚拟表偏移量 | 调用父类虚拟方法，使用目标对象的直接父类的虚拟表注6。 | FA40 8100 3254 – invoke-super-quick {v2, v3, v4, v5}, vtable #0081  调用父类虚拟方法，目标对象的实例指向位于v2寄存器，方法位于虚拟表#81H条目，方法所需的参数位于v3,v4,v5。 |
| FB | invoke-super-quick/range {参数范围}, 虚拟表偏移量 | 调用父类虚拟方法，使用目标对象的直接父类的虚拟表注6。 | F906 1B00 0000 – invoke-super-quick/range {v0..v5}, vtable #001b  调用父类虚拟方法，目标对象的实例指向位于v0寄存器，方法位于虚拟表#1B条目，方法所需的参数位于v1..v5。 |
| FC | unused\_FC | 未使用 |  |
| FD | unused\_FD | 未使用 |  |
| FE | unused\_FE | 未使用 |  |
| FF | unused\_FF | 未使用 |  |

注1：   Double和long值占用两个寄存器。（例：在vy地址上的值位于vy,vy+1寄存器）

注2：   偏移量可以是正或负，从指令起始字节起计算偏移量。偏移量在（2字节每1偏移量递增/递减）时解释执行。负偏移量用二进制补码格式存储。偏移量当前位置是指令起始字节。

注3：   比较操作，如果第一个操作数大于第二个操作数返回正值；如果两者相等，返回0；如果第一个操作数小于第二个操作数，返回负值。

注4：   正常使用没见到过的，从[Android opcode constant list](http://code.google.com/android/reference/dalvik/bytecode/Opcodes.html" \t "https://blog.csdn.net/qq373432361/article/details/_blank)引入。

注5：   调用参数表的编译比较诡异。如果参数的数量大于4并且%4=1，第5（第9或其他%4=1的）个参数将编译在指令字节的下一个字节的4个最低位。奇怪的是，有一种情况不使用这种编译：方法有4个参数但用于编译单一参数，指令字节的下一个字节的4个最低位空置，将会编译为40而不是04。

注6：   这是一个不安全的指令，仅适用于ODEX文件。