基于 E500 Core 的 PowerPC 指令集

@曾宇祥 2014.7.15~2014.8.3

1. 指令筛选说明及代码片段例证

1.1 lwarx/stwcx.

lwarx/stwcx.指令共同完成原子操作。
lwarx: Load Word And Reversed Indexed
stwcx.: Store Word Conditional Indexed
代码片段举例：

```
void inline compare_and_swap (volatile int * p, int oldval, int newval)
{
    int fail;
    __asm__ __volatile__ (
        "0: lwarx %0, 0, %1\n\t"
              "     xor. %0, %3, %0\n\t"
              " bne 1f\n\t"
               " stwcx. %2, 0, %1\n\t"
              " bne- 0b\n\t"
        " isync\n\t"
        "1: "
        : "=&r"(fail)
        : "r"(p), "r"(newval), "r"(oldval)
        : "cr0");
}
```

1.2 wrteei

wrteei: Write MSR External Enable Immediate
代码片段举例：

```
static inline void arch_local_irq_enable(void)
{
    #ifdef CONFIG_BOOKE
    asm volatile("wrteei 1" : : : "memory");
    65c:   7c 00 81 46   .long 0x7c008146
    local_irq_enable();

    /* Interrupts are enabled now so all GFP allocations are safe. */
    gfp_allowed_mask = __GFP_BITS_MASK;
    660:   3c 00 00 7f   lis     r0,127
    664:   3d 20 00 00   lis     r9,0
```

```
    668:   60 00 ff ff   ori     r0,r0,65535
    66c:   90 09 00 00   stw     r0,0(r9)
    ...
}
```

1.3 twi

twi: Trap Word Immediate

代码片段举例：

常用语 BUG()、WORN_TAINT()等宏汇编。

定义在..\linux-2.6.38\arch\powerpc\include\asm\bug.h 中

```
#define BUG() do {                      \
    __asm__ __volatile__(                   \
        "1: twi 31,0,0\n"           \
        _EMIT_BUG_ENTRY                 \
        : : "i" (__FILE__), "i" (__LINE__),     \
            "i" (0), "i"  (sizeof(struct bug_entry)));  \
    unreachable();                      \
} while (0)

#define __WARN_TAINT(taint) do {            \
    __asm__ __volatile__(                   \
        "1: twi 31,0,0\n"           \
        _EMIT_BUG_ENTRY                 \
        : : "i" (__FILE__), "i" (__LINE__),     \
          "i" (BUGFLAG_TAINT(taint)),       \
          "i" (sizeof(struct bug_entry)));      \
} while (0)
```

1.4 lwzx

lwzx: Load Word and Zero Indexed

代码片段举例：

使用-O1 以上优化时生成。

```
static void kbd_rawcode(unsigned char data)
{
    struct vc_data *vc = vc_cons[fg_console].d;
    1f38:  54 09 10 3a   rlwinm  r9,r0,2,0,29
    1f3c:  54 00 20 36   rlwinm  r0,r0,4,0,27
    1f40:  7c 09 02 14   add     r0,r9,r0
    1f44:  3d 20 00 00   lis     r9,0
    1f48:  39 29 00 00   addi    r9,r9,0
    1f4c:  7d 29 00 2e   lwzx    r9,r9,r0

    kbd = kbd_table + vc->vc_num;
    1f50:  a0 09 00 5c   lhz     r0,92(r9)
```

```
    1f54:  54 0b 10 3a   rlwinm  r11,r0,2,0,29
    1f58:  7c 0b 02 14   add     r0,r11,r0
    1f5c:  3d 60 00 00   lis     r11,0
    1f60:  39 6b 00 00   addi    r11,r11,0
    1f64:  39 6b 00 01   addi    r11,r11,1
    1f68:  7d 6b 02 14   add     r11,r11,r0
    if (kbd->kbdmode == VC_RAW)
    1f6c:  88 0b 00 03   lbz     r0,3(r11)
    ...
}
```

vc_data vc_cons 结构体见..\linux-2.6.38\include\linux\console_struct.h

1.5 addc/adde

addc: Add Carring

adde: Add Extended

通常在高精度（long long）加低精度时生成。

代码片段举例：

```
00000000 <main>:
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    long long a, b;

    a += b;
   c:   81 7f 00 08   lwz     r11,8(r31)
  10:   81 9f 00 0c   lwz     r12,12(r31)
  14:   81 3f 00 10   lwz     r9,16(r31)
  18:   81 5f 00 14   lwz     r10,20(r31)
  1c:   7d 4a 60 14   addc    r10,r10,r12
  20:   7d 29 59 14   adde    r9,r9,r11
  24:   91 3f 00 08   stw     r9,8(r31)
  28:   91 5f 00 0c   stw     r10,12(r31)

    return 0;
  2c:   38 00 00 00   li      r0,0
}
  30:   7c 03 03 78   mr      r3,r0
  34:   39 7f 00 20   addi    r11,r31,32
  38:   83 eb ff fc   lwz     r31,-4(r11)
  3c:   7d 61 5b 78   mr      r1,r11
  40:   4e 80 00 20   blr
```

1.6 addic/addze

addic: Add Immediate Carrying
addze: Add to Zero Extended
通常在高精度（longlong）加立即数时生成。
代码片段举例：

```
00000000 <main>:
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    long long a;
    int b;

    a += 2;
   c:   81 3f 00 08   lwz     r9,8(r31)
  10:   81 5f 00 0c   lwz     r10,12(r31)
  14:   31 4a 00 02   addic   r10,r10,2
  18:   7d 29 01 94   addze   r9,r9
  1c:   91 3f 00 08   stw     r9,8(r31)
  20:   91 5f 00 0c   stw     r10,12(r31)

    return 0;
  24:   38 00 00 00   li      r0,0
}
  28:   7c 03 03 78   mr      r3,r0
  2c:   39 7f 00 20   addi    r11,r31,32
  30:   83 eb ff fc   lwz     r31,-4(r11)
  34:   7d 61 5b 78   mr      r1,r11
  38:   4e 80 00 20   blr
```

1.7 lmw/stmw

lmw: Load Multiple Word
stmw: Store Multiple Word
用于函数首尾，快速保护 GPR 以及恢复 GPR。
代码片段举例：

```
// from linux-2.6.38/fs/ioctl.c
00000000 <fiemap_fill_next_extent>:
   0:   94 21 ff 90   stwu    r1,-112(r1)
   8:   be c1 00 48   stmw    r22,72(r1)       # Store Multiple word
   c:   7c 7f 1b 78   mr      r31,r3         # r22~r31 maybe used later
  10:   7c dd 33 78   mr      r29,r6
  14:   90 01 00 74   stw     r0,116(r1)
  18:   7c bc 2b 78   mr      r28,r5
```

```
 1c:  7d 1b 43 78   mr      r27,r8
 20:  7c fa 3b 78   mr      r26,r7
 24:  7d 59 53 78   mr      r25,r10
      ......        ......
164:  ba c1 00 48   lmw     r22,72(r1)       # Load Multiple word
168:  38 21 00 70   addi    r1,r1,112
16c:  7c 08 03 a6   mtlr    r0
170:  4e 80 00 20   blr
```

1.8 rlwinm

rlwinm: Rotate Left Word Immediate then And with MASK

常用于不同精度运算。

代码片段举例：

（1）

```
   for (i=0, j=0; i<16; ++i, j+=4)
 90:  38 00 00 00   li      r0,0
 94:  90 1f 00 08   stw     r0,8(r31)
 98:  38 00 00 00   li      r0,0
 9c:  90 1f 00 0c   stw     r0,12(r31)
 a0:  48 00 00 44   b       e4 <main+0x68>
      arr[i] = i*j;
 a4:  81 3f 00 08   lwz     r9,8(r31)
 a8:  80 1f 00 0c   lwz     r0,12(r31)
 ac:  7d 29 01 d6   mullw   r9,r9,r0
 b0:  3c 00 00 00   lis     r0,0
 b4:  31 60 00 00   addic   r11,r0,0
 b8:  80 1f 00 08   lwz     r0,8(r31)
 bc:  54 00 10 3a   rlwinm  r0,r0,2,0,29
 c0:  7c 0b 02 14   add     r0,r11,r0
 c4:  7c 0b 03 78   mr      r11,r0
 c8:  91 2b 00 00   stw     r9,0(r11)
```

（2）

```
00000000 <main>:
int main() {
   0:  94 21 ff e0   stwu    r1,-32(r1)
   4:  93 e1 00 1c   stw     r31,28(r1)
   8:  7c 3f 0b 78   mr      r31,r1
    int a;
    char b;

    a += b;
   c:  88 1f 00 08   lbz     r0,8(r31)
  10:  54 00 06 3e   clrlwi  r0,r0,24     # rlwinm
  14:  81 3f 00 0c   lwz     r9,12(r31)
```

```
 18:  7c 09 02 14   add    r0,r9,r0
 1c:  90 1f 00 0c   stw    r0,12(r31)

   return 0;
 20:  38 00 00 00   li     r0,0
}
```

1.9 rlwini

rlwini: Rotate Left Word Immediate then Mask Insert
常用语低精度向高精度转换。
代码片段举例：
（1）

```
// sb:super_block   s_bdev:block_device
// bd_inode:inode*  i_size:loff_t
// ../linux-2.6.38/fs/hfsplus/wrapper.c
static int hfsplus_get_last_session(struct super_block *sb,
                sector_t *start, sector_t *size)
{
 364:  94 21 ff a0   stwu   r1,-96(r1)
 368:  7c 08 02 a6   mflr   r0
 36c:  90 01 00 64   stw    r0,100(r1)
 370:  bf 01 00 40   stmw   r24,64(r1)
 374:  90 61 00 28   stw    r3,40(r1)
 378:  90 81 00 2c   stw    r4,44(r1)
 37c:  90 a1 00 30   stw    r5,48(r1)
   struct cdrom_multisession ms_info;
   struct cdrom_tocentry te;
   int res;

   /* default values */
   *start = 0;
 380:  80 01 00 2c   lwz    r0,44(r1)
 384:  39 20 00 00   li     r9,0
 388:  39 40 00 00   li     r10,0
 38c:  7c 0b 03 78   mr     r11,r0
 390:  91 2b 00 00   stw    r9,0(r11)
 394:  91 4b 00 04   stw    r10,4(r11)
   *size = sb->s_bdev->bd_inode->i_size >> 9;
 398:  80 01 00 28   lwz    r0,40(r1)
 39c:  7c 09 03 78   mr     r9,r0     #   sb
 3a0:  80 09 00 84   lwz    r0,132(r9)
 3a4:  7c 0b 03 78   mr     r11,r0    #   s_dev
 3a8:  80 0b 00 04   lwz    r0,4(r11)
 3ac:  7c 09 03 78   mr     r9,r0     #   bd_inode
```

```
 3b0:  81 69 00 70   lwz     r11,112(r9)  # i_size
 3b4:  81 89 00 74   lwz     r12,116(r9)  # i_size
 3b8:  55 8a ba 7e   rlwinm  r10,r12,23,9,31
 3bc:  51 6a b8 10   rlwimi  r10,r11,23,0,8
...
(2)
00000000 <main>:
int main() {
   0:  94 21 ff e0   stwu    r1,-32(r1)
   4:  93 e1 00 1c   stw     r31,28(r1)
   8:  7c 3f 0b 78   mr      r31,r1
    long long a;
    int b;

    b = a>>9;
   c:  81 3f 00 08   lwz     r9,8(r31)
  10:  81 5f 00 0c   lwz     r10,12(r31)
  14:  55 4c ba 7e   rlwinm  r12,r10,23,9,31
  18:  51 2c b8 10   rlwimi  r12,r9,23,0,8
  1c:  7d 2b 4e 70   srawi   r11,r9,9
  20:  91 9f 00 10   stw     r12,16(r31)

    return 0;
  24:  38 00 00 00   li      r0,0
}
```

1.10 bcctr/mtspr

bcctr:Branch Conditional to Count Register
mtspr:Move To Special Purpose Register
bcctr 常用于使用函数指针，mtspr 用于函数嵌套。

```
int f( int (*comp)(int, int) ) {
    int a = 10, b = 11;
    int c;

    c = comp(a, b);

    return c;
}

int min(int a, int b) {
    return a<b ? a:b;
}

int main() {
```

```
    int x;

    x = f(&min);

    return 0;
}

00000000 <f>:
int f( int (*comp)(int, int) ) {
   0:   94 21 ff d0   stwu    r1,-48(r1)
   4:   7c 08 02 a6   mflr    r0
   8:   90 01 00 34   stw     r0,52(r1)
   c:   93 e1 00 2c   stw     r31,44(r1)
  10:   7c 3f 0b 78   mr      r31,r1
  14:   90 7f 00 18   stw     r3,24(r31)
    int a = 10, b = 11;
  18:   38 00 00 0a   li      r0,10
  1c:   90 1f 00 08   stw     r0,8(r31)
  20:   38 00 00 0b   li      r0,11
  24:   90 1f 00 0c   stw     r0,12(r31)
    int c;

    c = comp(a, b);
  28:   80 1f 00 18   lwz     r0,24(r31)
  2c:   80 7f 00 08   lwz     r3,8(r31)
  30:   80 9f 00 0c   lwz     r4,12(r31)
  34:   7c 09 03 a6   mtctr   r0           # mtspr
  38:   4e 80 04 21   bctrl
  3c:   90 7f 00 10   stw     r3,16(r31)

    return c;
  40:   80 1f 00 10   lwz     r0,16(r31)
}
  44:   7c 03 03 78   mr      r3,r0
  48:   39 7f 00 30   addi    r11,r31,48
  4c:   80 0b 00 04   lwz     r0,4(r11)
  50:   7c 08 03 a6   mtlr    r0        # mtspr
  54:   83 eb ff fc   lwz     r31,-4(r11)
  58:   7d 61 5b 78   mr      r1,r11
  5c:   4e 80 00 20   blr
```

1.11 subfc/subfe/subfic

subfc:Subtract From Carring

subfe:Subtract From Extended

subfic:Subtract From Immedaite Carring
subfc/subfe 常用于 long long 的精度减法。
subfic 常用于 short 以下精度的立即数（作被减数）减法（立即数作减数使用 addic 指令）。
代码片段举例：

```
   long long lla; int ia;
   lla -= ia;
 14c:  81 9f 00 14   lwz    r12,20(r31)
 150:  80 1f 00 14   lwz    r0,20(r31)
 154:  7c 00 fe 70   srawi   r0,r0,31
 158:  7c 0b 03 78   mr     r11,r0
 15c:  80 ff 00 20   lwz    r7,32(r31)
 160:  81 1f 00 24   lwz    r8,36(r31)
 164:  7d 8c 40 10   subfc   r12,r12,r8
 168:  7d 6b 39 10   subfe   r11,r11,r7
 16c:  91 7f 00 20   stw    r11,32(r31)
 170:  91 9f 00 24   stw    r12,36(r31)
   short sa, sb;
   sa = 0x1234 - sb;
 250:  a0 1f 00 0a   lhz    r0,10(r31)
 254:  54 00 04 3e   clrlwi  r0,r0,16
 258:  20 00 12 34   subfic  r0,r0,4660
 25c:  54 00 04 3e   clrlwi  r0,r0,16
 260:  b0 1f 00 0c   sth    r0,12(r31)
```

1.12 mulli/mullw/mulhw/mulhwu

mulli:Multiply Low Immediate
mullw:Multiply Low Word
mulhw:Multiply High Word
mulhwu:Multiply High Word Unsigned
常用于各类乘法运算。其中 mullw+mulhw 可用于 long long 乘法，mullw+mulhwu 可用于 unsigned long long 乘法。
代码片段举例：

```
test:     file format elf32-powerpc


Disassembly of section .text:

00000000 <main>:
int main() {
   0:  94 21 ff a0   stwu   r1,-96(r1)
   4:  93 e1 00 5c   stw    r31,92(r1)
   8:  7c 3f 0b 78   mr     r31,r1
```

```
  unsigned long long llc, lld;
  unsigned int ic, id;
  unsigned short sc, sd;
  unsigned char cc, cd;

  lla = llb = ia = ib = sa = sb = ca = cb = 1;
 c:  38 00 00 01  li     r0,1
10:  98 1f 00 08  stb    r0,8(r31)
14:  88 1f 00 08  lbz    r0,8(r31)
18:  98 1f 00 09  stb    r0,9(r31)
1c:  88 1f 00 09  lbz    r0,9(r31)
20:  54 00 06 3e  clrlwi  r0,r0,24
24:  b0 1f 00 0a  sth    r0,10(r31)
28:  a0 1f 00 0a  lhz    r0,10(r31)
2c:  b0 1f 00 0c  sth    r0,12(r31)
30:  a0 1f 00 0c  lhz    r0,12(r31)
34:  7c 00 07 34  extsh   r0,r0
38:  90 1f 00 10  stw    r0,16(r31)
3c:  80 1f 00 10  lwz    r0,16(r31)
40:  90 1f 00 14  stw    r0,20(r31)
44:  80 1f 00 14  lwz    r0,20(r31)
48:  90 1f 00 1c  stw    r0,28(r31)
4c:  7c 00 fe 70  srawi   r0,r0,31
50:  90 1f 00 18  stw    r0,24(r31)
54:  81 3f 00 18  lwz    r9,24(r31)
58:  81 5f 00 1c  lwz    r10,28(r31)
5c:  91 3f 00 20  stw    r9,32(r31)
60:  91 5f 00 24  stw    r10,36(r31)
  llc = lld = ic = id = sc = cd = cc = cd = 2;
64:  38 00 00 02  li     r0,2
68:  98 1f 00 28  stb    r0,40(r31)
6c:  88 1f 00 28  lbz    r0,40(r31)
70:  98 1f 00 29  stb    r0,41(r31)
74:  88 1f 00 29  lbz    r0,41(r31)
78:  98 1f 00 28  stb    r0,40(r31)
7c:  88 1f 00 28  lbz    r0,40(r31)
80:  54 00 06 3e  clrlwi  r0,r0,24
84:  b0 1f 00 2a  sth    r0,42(r31)
88:  a0 1f 00 2a  lhz    r0,42(r31)
8c:  54 00 04 3e  clrlwi  r0,r0,16
90:  90 1f 00 2c  stw    r0,44(r31)
94:  80 1f 00 2c  lwz    r0,44(r31)
98:  90 1f 00 30  stw    r0,48(r31)
9c:  80 1f 00 30  lwz    r0,48(r31)
```

```
 a0:  90 1f 00 3c  stw    r0,60(r31)
 a4:  38 00 00 00  li     r0,0
 a8:  90 1f 00 38  stw    r0,56(r31)
 ac:  81 3f 00 38  lwz    r9,56(r31)
 b0:  81 5f 00 3c  lwz    r10,60(r31)
 b4:  91 3f 00 40  stw    r9,64(r31)
 b8:  91 5f 00 44  stw    r10,68(r31)

   lla *= llb;
 bc:  81 3f 00 20  lwz    r9,32(r31)
 c0:  80 1f 00 1c  lwz    r0,28(r31)
 c4:  7d 29 01 d6  mullw   r9,r9,r0
 c8:  81 5f 00 18  lwz    r10,24(r31)
 cc:  80 1f 00 24  lwz    r0,36(r31)
 d0:  7c 0a 01 d6  mullw   r0,r10,r0
 d4:  7c 09 02 14  add    r0,r9,r0
 d8:  80 7f 00 24  lwz    r3,36(r31)
 dc:  80 9f 00 1c  lwz    r4,28(r31)
 e0:  7d 23 20 16  mulhwu  r9,r3,r4
 e4:  7d 43 21 d6  mullw   r10,r3,r4
 e8:  7c 00 4a 14  add    r0,r0,r9
 ec:  7c 09 03 78  mr     r9,r0
 f0:  91 3f 00 20  stw    r9,32(r31)
 f4:  91 5f 00 24  stw    r10,36(r31)
 f8:  91 3f 00 20  stw    r9,32(r31)
 fc:  91 5f 00 24  stw    r10,36(r31)
   ia *= ib;
100:  81 3f 00 14  lwz    r9,20(r31)
104:  80 1f 00 10  lwz    r0,16(r31)
108:  7c 09 01 d6  mullw   r0,r9,r0
10c:  90 1f 00 14  stw    r0,20(r31)
   sa *= sb;
110:  a0 1f 00 0c  lhz    r0,12(r31)
114:  54 09 04 3e  clrlwi  r9,r0,16
118:  a0 1f 00 0a  lhz    r0,10(r31)
11c:  54 00 04 3e  clrlwi  r0,r0,16
120:  7c 09 01 d6  mullw   r0,r9,r0
124:  54 00 04 3e  clrlwi  r0,r0,16
128:  b0 1f 00 0c  sth    r0,12(r31)
   ca *= cb;
12c:  89 3f 00 09  lbz    r9,9(r31)
130:  88 1f 00 08  lbz    r0,8(r31)
134:  7c 09 01 d6  mullw   r0,r9,r0
138:  98 1f 00 09  stb    r0,9(r31)
```

```
  sa *= ca;
13c:  88 1f 00 09  lbz    r0,9(r31)
140:  54 00 06 3e  clrlwi  r0,r0,24
144:  54 09 04 3e  clrlwi  r9,r0,16
148:  a0 1f 00 0c  lhz    r0,12(r31)
14c:  54 00 04 3e  clrlwi  r0,r0,16
150:  7c 09 01 d6  mullw   r0,r9,r0
154:  54 00 04 3e  clrlwi  r0,r0,16
158:  b0 1f 00 0c  sth    r0,12(r31)
  ia *= sa;
15c:  a0 1f 00 0c  lhz    r0,12(r31)
160:  7c 00 07 34  extsh   r0,r0
164:  81 3f 00 14  lwz    r9,20(r31)
168:  7c 09 01 d6  mullw   r0,r9,r0
16c:  90 1f 00 14  stw    r0,20(r31)
  lla *= ia;
170:  80 df 00 14  lwz    r6,20(r31)
174:  80 1f 00 14  lwz    r0,20(r31)
178:  7c 00 fe 70  srawi   r0,r0,31
17c:  7c 05 03 78  mr     r5,r0
180:  80 1f 00 20  lwz    r0,32(r31)
184:  7d 20 31 d6  mullw   r9,r0,r6
188:  80 1f 00 24  lwz    r0,36(r31)
18c:  7c 00 29 d6  mullw   r0,r0,r5
190:  7c 09 02 14  add    r0,r9,r0
194:  80 9f 00 24  lwz    r4,36(r31)
198:  7d 24 30 16  mulhwu  r9,r4,r6
19c:  7d 44 31 d6  mullw   r10,r4,r6
1a0:  7c 00 4a 14  add    r0,r0,r9
1a4:  7c 09 03 78  mr     r9,r0
1a8:  91 3f 00 20  stw    r9,32(r31)
1ac:  91 5f 00 24  stw    r10,36(r31)
1b0:  91 3f 00 20  stw    r9,32(r31)
1b4:  91 5f 00 24  stw    r10,36(r31)

  llc *= lld;
1b8:  81 3f 00 40  lwz    r9,64(r31)
1bc:  80 1f 00 3c  lwz    r0,60(r31)
1c0:  7d 29 01 d6  mullw   r9,r9,r0
1c4:  81 5f 00 38  lwz    r10,56(r31)
1c8:  80 1f 00 44  lwz    r0,68(r31)
1cc:  7c 0a 01 d6  mullw   r0,r10,r0
1d0:  7c 09 02 14  add    r0,r9,r0
```

```
1d4:  80 bf 00 44  lwz    r5,68(r31)
1d8:  80 df 00 3c  lwz    r6,60(r31)
1dc:  7d 25 30 16  mulhwu  r9,r5,r6
1e0:  7d 45 31 d6  mullw   r10,r5,r6
1e4:  7c 00 4a 14  add    r0,r0,r9
1e8:  7c 09 03 78  mr     r9,r0
1ec:  91 3f 00 40  stw    r9,64(r31)
1f0:  91 5f 00 44  stw    r10,68(r31)
1f4:  91 3f 00 40  stw    r9,64(r31)
1f8:  91 5f 00 44  stw    r10,68(r31)
  ic *= id;
1fc:  81 3f 00 30  lwz    r9,48(r31)
200:  80 1f 00 2c  lwz    r0,44(r31)
204:  7c 09 01 d6  mullw   r0,r9,r0
208:  90 1f 00 30  stw    r0,48(r31)
  sc *= sd;
20c:  a1 3f 00 2a  lhz    r9,42(r31)
210:  a0 1f 00 48  lhz    r0,72(r31)
214:  7c 09 01 d6  mullw   r0,r9,r0
218:  b0 1f 00 2a  sth    r0,42(r31)
  cc *= sd;
21c:  a0 1f 00 48  lhz    r0,72(r31)
220:  54 09 06 3e  clrlwi  r9,r0,24
224:  88 1f 00 29  lbz    r0,41(r31)
228:  7c 09 01 d6  mullw   r0,r9,r0
22c:  98 1f 00 29  stb    r0,41(r31)

  sc *= cc;
230:  88 1f 00 29  lbz    r0,41(r31)
234:  54 00 06 3e  clrlwi  r0,r0,24
238:  54 09 04 3e  clrlwi  r9,r0,16
23c:  a0 1f 00 2a  lhz    r0,42(r31)
240:  7c 09 01 d6  mullw   r0,r9,r0
244:  b0 1f 00 2a  sth    r0,42(r31)
  ic *= sc;
248:  a0 1f 00 2a  lhz    r0,42(r31)
24c:  54 00 04 3e  clrlwi  r0,r0,16
250:  81 3f 00 30  lwz    r9,48(r31)
254:  7c 09 01 d6  mullw   r0,r9,r0
258:  90 1f 00 30  stw    r0,48(r31)
  llc *= ic;
25c:  81 1f 00 30  lwz    r8,48(r31)
260:  38 e0 00 00  li     r7,0
264:  80 1f 00 40  lwz    r0,64(r31)
```

```
268:  7d 20 41 d6  mullw   r9,r0,r8
26c:  80 1f 00 44  lwz     r0,68(r31)
270:  7c 00 39 d6  mullw   r0,r0,r7
274:  7c 09 02 14  add     r0,r9,r0
278:  80 df 00 44  lwz     r6,68(r31)
27c:  7d 26 40 16  mulhwu  r9,r6,r8
280:  7d 46 41 d6  mullw   r10,r6,r8
284:  7c 00 4a 14  add     r0,r0,r9
288:  7c 09 03 78  mr      r9,r0
28c:  91 3f 00 40  stw     r9,64(r31)
290:  91 5f 00 44  stw     r10,68(r31)
294:  91 3f 00 40  stw     r9,64(r31)
298:  91 5f 00 44  stw     r10,68(r31)

  lla = 0x700012345678 * ib;
29c:  81 9f 00 10  lwz     r12,16(r31)
2a0:  80 1f 00 10  lwz     r0,16(r31)
2a4:  7c 00 fe 70  srawi   r0,r0,31
2a8:  7c 0b 03 78  mr      r11,r0
2ac:  3c 00 12 34  lis     r0,4660
2b0:  60 00 56 78  ori     r0,r0,22136
2b4:  7d 2b 01 d6  mullw   r9,r11,r0
2b8:  1c 0c 70 00  mulli   r0,r12,28672
2bc:  7c 09 02 14  add     r0,r9,r0
2c0:  3d 20 12 34  lis     r9,4660
2c4:  61 28 56 78  ori     r8,r9,22136
2c8:  7d 2c 40 16  mulhwu  r9,r12,r8
2cc:  7d 4c 41 d6  mullw   r10,r12,r8
2d0:  7c 00 4a 14  add     r0,r0,r9
2d4:  7c 09 03 78  mr      r9,r0
2d8:  91 3f 00 20  stw     r9,32(r31)
2dc:  91 5f 00 24  stw     r10,36(r31)
2e0:  91 3f 00 20  stw     r9,32(r31)
2e4:  91 5f 00 24  stw     r10,36(r31)
  ia = 0x70001234 * sb;
2e8:  a0 1f 00 0a  lhz     r0,10(r31)
2ec:  7c 09 07 34  extsh   r9,r0
2f0:  3c 00 70 00  lis     r0,28672
2f4:  60 00 12 34  ori     r0,r0,4660
2f8:  7c 09 01 d6  mullw   r0,r9,r0
2fc:  90 1f 00 14  stw     r0,20(r31)
  ia = 0x1234 * sb;
300:  a0 1f 00 0a  lhz     r0,10(r31)
304:  7c 00 07 34  extsh   r0,r0
```

```
 308:  1c 00 12 34   mulli   r0,r0,4660
 30c:  90 1f 00 14   stw     r0,20(r31)
   sa = 0x1234 * sb;
 310:  a0 1f 00 0a   lhz     r0,10(r31)
 314:  54 00 04 3e   clrlwi  r0,r0,16
 318:  1c 00 12 34   mulli   r0,r0,4660
 31c:  54 00 04 3e   clrlwi  r0,r0,16
 320:  b0 1f 00 0c   sth     r0,12(r31)
   ca = 0x12 * cb;
 324:  88 1f 00 08   lbz     r0,8(r31)
 328:  54 00 08 3c   rlwinm  r0,r0,1,0,30
 32c:  54 09 18 38   rlwinm  r9,r0,3,0,28
 330:  7c 00 4a 14   add     r0,r0,r9
 334:  98 1f 00 09   stb     r0,9(r31)

   ic = 0x70001 * ic;
 338:  81 7f 00 30   lwz     r11,48(r31)
 33c:  7d 60 5b 78   mr      r0,r11
 340:  54 00 80 1e   rlwinm  r0,r0,16,0,15
 344:  54 09 18 38   rlwinm  r9,r0,3,0,28
 348:  7d 20 48 50   subf    r9,r0,r9
 34c:  7c 09 5a 14   add     r0,r9,r11
 350:  90 1f 00 30   stw     r0,48(r31)

   return 0;
 354:  38 00 00 00   li      r0,0
}
 358:  7c 03 03 78   mr      r3,r0
 35c:  39 7f 00 60   addi    r11,r31,96
 360:  83 eb ff fc   lwz     r31,-4(r11)
 364:  7d 61 5b 78   mr      r1,r11
 368:  4e 80 00 20   blr
```

1.13 oris

oris:OR Immediate Shift

常用于或 32 位数。

代码片段举例：

```
   b = a | 0xabcd1234;
  14:  80 1f 00 08   lwz     r0,8(r31)
  18:  64 00 ab cd   oris    r0,r0,43981 # 43981 = 0xabcd
  1c:  60 00 12 34   ori     r0,r0,4660     # 4660 = 0x1234
  20:  90 1f 00 0c   stw     r0,12(r31)
```

而与 32 位数不适用 andis

```
   b = a & 0xabcd1234;
```

```
  24:  81 3f 00 08   lwz     r9,8(r31)
  28:  3c 00 ab cd   lis     r0,-21555
  2c:  60 00 12 34   ori     r0,r0,4660
  30:  7d 20 00 38   and     r0,r9,r0
  34:  90 1f 00 0c   stw     r0,12(r31)
```

1.14 mfmsr/mtmsr

mfmsr:Move From Machine State Register

mtmsr: Move From Machine State Register

常用于 mfmsr（）、mtmsr（）函数，用于中断中。

代码片段举例：

```
// ..\linux-2.6.38\arch\powerpc\include\asm\reg.h
#define mfmsr()     ({unsigned long rval; \
          asm volatile("mfmsr %0" : "=r" (rval)); rval;})
```

1.15 andc

andc: AND with Complement

在 linux 中用于 clear_bit，其实可以使用 nor+or 指令代替。

代码片段举例：

```
..\linux-2.6.38\arch\powerpc\include\asm\bitops.h
/* Macro for generating the ***_bits() functions */
#define DEFINE_BITOP(fn, op, prefix, postfix)  \
static __inline__ void fn(unsigned long mask,  \
       volatile unsigned long *_p)   \
{                   \
    unsigned long old;          \
    unsigned long *p = (unsigned long *)_p; \
    __asm__ __volatile__ (         \
    prefix              \
"1:"   PPC_LLARX(%0,0,%3,0) "\n"      \
    stringify_in_c(op) "%0,%0,%2\n"       \
    PPC405_ERR77(0,%3)          \
    PPC_STLCX "%0,0,%3\n"          \
    "bne- 1b\n"             \
    postfix                 \
    : "=&r" (old), "+m" (*p)       \
    : "r" (mask), "r" (p)          \
    : "cc", "memory");          \
}

DEFINE_BITOP(set_bits, or, "", "")
DEFINE_BITOP(clear_bits, andc, "", "")
DEFINE_BITOP(clear_bits_unlock, andc, PPC_RELEASE_BARRIER, "")
DEFINE_BITOP(change_bits, xor, "", "")
```

1.16 rlwnm

rlwnm:Rotate Left Word the AND Mask
gcc 可通过如下形式的代码编译得到，不需要任何优化。（似乎是仅以下形式）
代码举例：

```
// ..\linux-2.6.38\include\linux\bitops.h
00000000 <rol32>:
#define __u32 unsigned int

static inline __u32 rol32(__u32 word, unsigned int shift)
{
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
   c:   90 7f 00 08   stw     r3,8(r31)
  10:   90 9f 00 0c   stw     r4,12(r31)
    return (word << shift) | (word >> (32 - shift));
  14:   81 3f 00 08   lwz     r9,8(r31)
  18:   80 1f 00 0c   lwz     r0,12(r31)
  1c:   5d 20 00 3e   rotlw   r0,r9,r0 #rlwnm r0,r9,r0,0,31
}
  20:   7c 03 03 78   mr      r3,r0
  24:   39 7f 00 20   addi    r11,r31,32
  28:   83 eb ff fc   lwz     r31,-4(r11)
  2c:   7d 61 5b 78   mr      r1,r11
  30:   4e 80 00 20   blr

00000034 <ror32>:

static inline __u32 ror32(__u32 word, unsigned int shift)
{
  34:   94 21 ff e0   stwu    r1,-32(r1)
  38:   93 e1 00 1c   stw     r31,28(r1)
  3c:   7c 3f 0b 78   mr      r31,r1
  40:   90 7f 00 08   stw     r3,8(r31)
  44:   90 9f 00 0c   stw     r4,12(r31)
    return (word >> shift) | (word << (32 - shift));
  48:   80 1f 00 0c   lwz     r0,12(r31)
  4c:   20 00 00 20   subfic  r0,r0,32
  50:   81 3f 00 08   lwz     r9,8(r31)
  54:   5d 20 00 3e   rotlw   r0,r9,r0 #rlwnm r0,r9,r0,0,31
}
  58:   7c 03 03 78   mr      r3,r0
  5c:   39 7f 00 20   addi    r11,r31,32
```

```
  60:   83 eb ff fc   lwz     r31,-4(r11)
  64:   7d 61 5b 78   mr      r1,r11
  68:   4e 80 00 20   blr
```

1.17 neg

neg:Negate

常用于相反数的运算。

代码片段举例：

```
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    int a = -1;
   c:   38 00 ff ff   li      r0,-1
  10:   90 1f 00 08   stw     r0,8(r31)
    a = -a;
  14:   80 1f 00 08   lwz     r0,8(r31)
  18:   7c 00 00 d0   neg     r0,r0
  1c:   90 1f 00 08   stw     r0,8(r31)
    return 0;
  20:   38 00 00 00   li      r0,0
}
```

1.18 crnor

crnor:Conditon Register NOR

GCC 中用于对小于比较表达式的赋值。

代码片段举例：

```
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    int a = 10;
   c:   38 00 00 0a   li      r0,10
  10:   90 1f 00 08   stw     r0,8(r31)

    a = a<15;
  14:   80 1f 00 08   lwz     r0,8(r31)
  18:   2f 80 00 0e   cmpwi   cr7,r0,14
  1c:   4f dd e8 42   crnot   4*cr7+eq,4*cr7+gt
  20:   7c 00 00 26   mfcr    r0
  24:   54 00 ff fe   rlwinm  r0,r0,31,31,31
  28:   90 1f 00 08   stw     r0,8(r31)

    return 0;
```

```
  2c:  38 00 00 00   li     r0,0
```

1.19 lwbrx/lhbrx/sthbrx/stwbrx

lwbrx:Load Word Byte Reversed Indexed
lhbrx:Load Half word Byte Reversed Indexed
stwbrx:Store Word Byte Reversed Indexed
sthbrx:Store Half word Byte Reversed Indexed
常用于获得相反尾端的指令或数据有关。（从 BE 获得 LE）
代码片段举例：

```
// ..\linux-2.6.38\arch\powerpc\include\asm\io.h
#define DEF_MMIO_IN_BE(name, size, insn)                \
static inline u##size name(const volatile u##size __iomem *addr)
    \
{                                   \
    u##size ret;                            \
    __asm__ __volatile__("sync;"#insn"%U1%X1 %0,%1;twi
0,%0,0;isync"\
        : "=r" (ret) : "m" (*addr) : "memory");         \
    return ret;                             \
}

#define DEF_MMIO_OUT_BE(name, size, insn)               \
static inline void name(volatile u##size __iomem *addr, u##size val)
    \
{                                   \
    __asm__ __volatile__("sync;"#insn"%U0%X0 %1,%0"             \
        : "=m" (*addr) : "r" (val) : "memory");         \
    IO_SET_SYNC_FLAG();                     \
}
DEF_MMIO_IN_LE(in_le16, 16, lhbrx);
DEF_MMIO_IN_LE(in_le32, 32, lwbrx);
DEF_MMIO_OUT_LE(out_le16, 16, sthbrx);
DEF_MMIO_OUT_LE(out_le32, 32, stwbrx);
```

1.20 lbzx/stbx

lbzx:Load Byte and Zero Indexed
stbx:Store Byte Indexed
lbzx 常用于对数组（仅 extern）的下标索引。
代码片段举例：

```
// ..\linux-2.6.38\include\linux\kernel.h
extern char hex_asc[];
#define hex_asc_lo(x)   hex_asc[((x) & 0x0f)]
#define hex_asc_hi(x)   hex_asc[((x) & 0xf0) >> 4]
```

```
int main() {
   0:   94 21 ff c0   stwu    r1,-64(r1)
   4:   93 e1 00 3c   stw     r31,60(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    int x = 8;
   c:   38 00 00 08   li      r0,8
  10:   90 1f 00 08   stw     r0,8(r31)
    char ch;
    char arr[30];
    ch = hex_asc_lo(x);
  14:   80 1f 00 08   lwz     r0,8(r31)
  18:   54 00 07 3e   clrlwi  r0,r0,28
  1c:   3d 20 00 00   lis     r9,0
  20:   39 29 00 00   addi    r9,r9,0
  24:   7c 09 00 ae   lbzx    r0,r9,r0
  28:   98 1f 00 0c   stb     r0,12(r31)
    hex_asc_lo(x) = ch;
  2c:   80 1f 00 08   lwz     r0,8(r31)
  30:   54 00 07 3e   clrlwi  r0,r0,28
  34:   3d 20 00 00   lis     r9,0
  38:   39 29 00 00   addi    r9,r9,0
  3c:   89 7f 00 0c   lbz     r11,12(r31)
  40:   7d 69 01 ae   stbx    r11,r9,r0

    return 0;
  44:   38 00 00 00   li      r0,0
}
```

1.21 addme

addme:Add to Minus one Extended

常用于 longlong 减小于 longlong 的运算。可使用 addic 或 addc+adde 精简。

```
00000000 <main>:
int main() {
   0:   94 21 ff d0   stwu    r1,-48(r1)
   4:   93 e1 00 2c   stw     r31,44(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    unsigned long long a = 1;
   c:   39 20 00 00   li      r9,0
  10:   39 40 00 01   li      r10,1
  14:   91 3f 00 08   stw     r9,8(r31)
  18:   91 5f 00 0c   stw     r10,12(r31)
    unsigned long port = 2;
  1c:   38 00 00 02   li      r0,2
  20:   90 1f 00 10   stw     r0,16(r31)
```

```
   unsigned long long end = 3;
  24:   39 20 00 00   li      r9,0
  28:   39 40 00 03   li      r10,3
  2c:   91 3f 00 18   stw     r9,24(r31)
  30:   91 5f 00 1c   stw     r10,28(r31)

   a = port + end - 10;
  34:   81 9f 00 10   lwz     r12,16(r31)  # r12 = port
  38:   39 60 00 00   li      r11,0
  3c:   81 3f 00 18   lwz     r9,24(r31)       # r9 = end_high
  40:   81 5f 00 1c   lwz     r10,28(r31)  # r10 = end_low
  44:   7d 4a 60 14   addc    r10,r10,r12  # r10 += r12
  48:   7d 29 59 14   adde    r9,r9,r11    # r9 += CA
  4c:   31 4a ff f6   addic   r10,r10,-10  # r10 -= 10
  50:   7d 29 01 d4   addme   r9,r9        # r9 += 0xff...ff
  54:   91 3f 00 08   stw     r9,8(r31)
  58:   91 5f 00 0c   stw     r10,12(r31)
```

1.22 xoris

xoris:XOR Immediate Shifted

常用于判断与大于 16 位立即数不相等的比较表达式赋值。

代码片段举例：

```
00000000 <main>:
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    int a = 0x12340001;
   c:   3c 00 12 34   lis     r0,4660
  10:   60 00 00 01   ori     r0,r0,1
  14:   90 1f 00 08   stw     r0,8(r31)

   a = a!=0x12340023;
  18:   80 1f 00 08   lwz     r0,8(r31)
  1c:   6c 00 12 34   xoris   r0,r0,4660   # 4660 = 0x1234
  20:   68 00 00 23   xori    r0,r0,35
  24:   2f 80 00 00   cmpwi   cr7,r0,0
  28:   7c 00 00 26   mfcr    r0
  2c:   54 00 ff fe   rlwinm  r0,r0,31,31,31
  30:   68 00 00 01   xori    r0,r0,1
  34:   90 1f 00 08   stw     r0,8(r31)
```

1.23 andis

andis.:AND Immediate Shifted

用于低 16 位全零立即数与
代码片段举例：

```
00000000 <main>:
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    long long a = 1;
   c:   39 20 00 00   li      r9,0
  10:   39 40 00 01   li      r10,1
  14:   91 3f 00 08   stw     r9,8(r31)
  18:   91 5f 00 0c   stw     r10,12(r31)
    a = 0x34000000 & a;
  1c:   80 1f 00 08   lwz     r0,8(r31)
  20:   70 00 00 00   andi.   r0,r0,0
  24:   90 1f 00 08   stw     r0,8(r31)
  28:   80 1f 00 0c   lwz     r0,12(r31)
  2c:   74 00 34 00   andis.  r0,r0,13312
  30:   90 1f 00 0c   stw     r0,12(r31)
```

1.24 subfze

subfze:Subtract From Zero Extended
常用于立即数（小于等于 16 位）减 long long 类型，做高位拓展。
代码片段举例：

```
00000000 <main>:
int main() {
   0:   94 21 ff e0   stwu    r1,-32(r1)
   4:   93 e1 00 1c   stw     r31,28(r1)
   8:   7c 3f 0b 78   mr      r31,r1
    long long a = 1;
   c:   39 20 00 00   li      r9,0
  10:   39 40 00 01   li      r10,1
  14:   91 3f 00 08   stw     r9,8(r31)
  18:   91 5f 00 0c   stw     r10,12(r31)
    a = 34-a;
  1c:   81 3f 00 08   lwz     r9,8(r31)
  20:   81 5f 00 0c   lwz     r10,12(r31)
  24:   21 4a 00 22   subfic  r10,r10,34
  28:   7d 29 01 90   subfze  r9,r9
  2c:   91 3f 00 08   stw     r9,8(r31)
  30:   91 5f 00 0c   stw     r10,12(r31)
```

1.25 lwzu/lwzux/lhzu/lhzux/lbzu/lbzux/lhau/lhaux/stbu/stbu

x/sthu/sthux/stwu/stwux

以上 Load/Store 类指令均将存储地址写回基址寄存器。该类指令均可使用普通的 Load/Store 指令+计算类指令替换。

1.26 lha/lhax

lha:Load Half Word Algebraic

lhax:Load Half Word Algebraic Indexed

以上指令几乎不出现，是因为 PPC 具有 extsb/extsh/extsw 类指令。或者优化 GCC 编译，或者精简指令。

1.27 bclrl

bclrl:Branch Conditional to Link Register

适用于..\linux-2.6.38\arch\powerpc\kernel\entry_32.S 以及..\linux-2.6.38\arch\powerpc\kernel\misc_32.S 中。

代码片段举例：

```
    /* To be certain of avoiding problems with self-modifying code
     * execute a serializing instruction here.
     */
    isync
    sync

    mfspr  r3, SPRN_PIR /* current core we are running on */
    mr  r4, r5 /* load physical address of chunk called */

    /* jump to the entry point, usually the setup routine */
    mtlr   r5
    blrl
```

1.28 crand/crxor

crand:Condition Register AND

crxor:Condition Register XOR

crand 出现于../linux-2.6.38/arch/powerpc/lib/copy_32.S 中。

crxor 出现于../linux-2.6.38/arch/powerpc/lib/copy_32.S、../linux-2.6.38/arch/powerpc/kernel/vdso32/cacheflush.S、../linux-2.6.38/arch/powerpc/kernel/vdso32/datapage.S、../linux-2.6.38/arch/powerpc/kernel/misc_32.S。但仅 misc_32.S 在 vmlinux 中使用一次。其实 CR 类的逻辑指令，均可以通过 mfcr/mtcrf 以及相应的逻辑运算指令实现。

2. 核心指令集

3. 核心指令集 RTL 描述

3.0 指令操作说明

助记符	说明
←	赋值
$<_u$, $>_u$	无符号比较
EXTS(X)	符号扩展 X
MASK(X, Y)	若 X <= Y，从 X 到 Y 位（且包括 Y）均为 1，其余位为 0； 若 X > Y，从 Y 到 X 位（且包括 Y）均为 0，其余位为 1；
MEM(X, Y)	去除存储器地址为 X 的数据。其中，Y=1，2，4： Y = 1，目标数据为字节； Y = 2，目标数据为半字； Y = 4，目标数据为字；
$ROTL_{32}$(X, Y)	数据 X 向左旋转 Y 位。（即 X||X 左移 Y 位后，取高 32 位）
TRAP	唤醒系统陷阱
PC	当前指令地址
NPC	下条指令地址
X:Y	从第 X 位到第 Y 位（包括第 Y 位）的数据
undefined	未定义值
CA	XER[CA]
OV	XER[OV]
SO	XER[SO]
CR0	CR[0]
m	模式选择。若为 32 位，m=32；若为 64 位，m=0。

3.1 add: 有符号加　　XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	100001010	Rc
	6	5	5	5	1	9	1
格式	add rD,rA,rB					(OE=0,Rc=0)	
	add. rD,rA,rB					(OE=0,Rc=1)	
	addo rD,rA,rB					(OE=1,Rc=0)	
	addo. rD,rA,rB					(OE=1,Rc=1)	

操作	$temp_{m:64}$ ← (GPR[rA]$_m$||GPR[rA]$_{m:63}$) + (GPR[rB]$_m$||GPR[rB]$_{m:63}$)
	GPR[rD] ← $temp_{m+1:64}$
	if OE==1 then
	OV ← $temp_{m+1}$ XOR $temp_m$
	SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	LT ← $temp_{m+1:64}$ < 0
	GT ← $temp_{m+1:64}$ > 0						

	EQ ← $temp_{m+1:64}$ == 0 CR0 ← LT || GT || EQ || SO
其它 寄存器	CR0 (if RC==1) SO OV (if OE==1)

3.2 addc: 有符号加并修改 CA XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	000001010	Rc
	6	5	5	5	1	9	1
格式	addc rD,rA,rB (OE=0,Rc=0) addc. rD,rA,rB (OE=0,Rc=1) addco rD,rA,rB (OE=1,Rc=0) addco. rD,rA,rB (OE=1,Rc=1)						

操作	$temp_{m:64}$ ← (GPR[rA]$_m$||GPR[rA]$_{m:63}$) + (GPR[rB]$_m$||GPR[rB]$_{m:63}$)
	GPR[rD] ← $temp_{m+1:64}$
	CA ← $temp_{m+1}$
	if OE==1 then
	OV ← $temp_{m+1}$ XOR $temp_m$
	SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	LT ← $temp_{m+1:64}$ < 0
	GT ← $temp_{m+1:64}$ > 0
	EQ ← $temp_{m+1:64}$ == 0
	CR0 ← LT || GT || EQ || SO						

其它 寄存器	CA CR0 (if RC==1) SO OV (if OE==1)						

3.3 adde: 有符号加并加 CA 同时修改 CA XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	010001010	Rc
	6	5	5	5	1	9	1
格式	adde rD,rA,rB (OE=0,Rc=0) adde. rD,rA,rB (OE=0,Rc=1) addeo rD,rA,rB (OE=1,Rc=0) addeo. rD,rA,rB (OE=1,Rc=1)						
操作	$temp_{m:64}$ ← (GPR[rA]$_m$||GPR[rA]$_{m:63}$) + (GPR[rB]$_m$||GPR[rB]$_{m:63}$)+CA GPR[rD] ← $temp_{m+1:64}$ CA ← $temp_{m+1}$ if OE==1 then OV ← $temp_{m+1}$ XOR $temp_m$ SO ← SO | $temp_{m+1}$ XOR $temp_m$						

	if Rc==1 then LT ← $temp_{m+1:64}$ < 0 GT ← $temp_{m+1:64}$ > 0 EQ ← $temp_{m+1:64}$ == 0 CR0 ← LT || GT || EQ || SO
其它 寄存器	CA CR0 (if RC==1) SO OV (if OE==1)

3.4 addi: 有符号加立即数 D-Form

编码	0 5	6 10	11 15	16 31
	001110	rD	rA	SIMM
	6	5	5	16
格式	addi rD,rA,SIMM			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] $temp_{m:63}$ ← $a_{m:63}$ + $EXTS(SIMM)_{m:63}$ GPR[rD] ← $temp_{m:63}$			
其它 寄存器				

3.5 addic: 有符号加立即数并修改 CA D-Form

编码	0 5	6 10	11 15	16 31
	001100	rD	rA	SIMM
	6	5	5	16
格式	addic rD,rA,SIMM			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] $temp_{m:63}$ ← $a_{m:63}$ + $EXTS(SIMM)_{m:63}$ GPR[rD] ← $temp_{m:63}$ CA ← $temp_{m}$			
其它 寄存器	CA			

3.6 addic.: 有符号加立即数并修改 CA 与 CR0 D-Form

编码	0 5	6 10	11 15	16 31
	001101	rD	rA	SIMM
	6	5	5	16
格式	addic. rD,rA,SIMM			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] $temp_{m:63}$ ← $a_{m:63}$ + $EXTS(SIMM)_{m:63}$ GPR[rD] ← $temp_{m:63}$			

	CA ← $temp_m$ LT ← $temp_{m:63}$ < 0 GT ← $temp_{m:63}$ > 0 EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO
其它寄存器	CA CR0

3.7 addis: 有符号加移位立即数 D-Form

编码	0 5	6 10	11 15	16 31
	001101	rD	rA	SIMM
	6	5	5	16
格式	addis rD,rA,SIMM			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] $temp_{m:63}$ ← $a_{m:63}$ + EXTS(SIMM)$_{m:63}$ GPR[rD] ← $temp_{m:63}$			
其它寄存器				

3.8 addme: 有符号加-1 及 CA 同时修改 CA XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	00000	OE	011101010	Rc
	6	5	5	5	1	9	1
格式	addme rD,rA					(OE=0,Rc=0)	
	addme. rD,rA					(OE=0,Rc=1)	
	addmeo rD,rA					(OE=1,Rc=0)	
	addmeo. rD,rA					(OE=1,Rc=1)	

操作	$temp_{m:64}$ ← (GPR[rA]$_m$||GPR[rA]$_{m:63}$) + (1||0xFFFF..FF) + CA
	GPR[rD] ← $temp_{m+1:64}$
	CA ← $temp_{m+1}$
	if OE==1 then
	  OV ← $temp_{m+1}$ XOR $temp_m$
	  SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	  LT ← $temp_{m+1:64}$ < 0
	  GT ← $temp_{m+1:64}$ > 0
	  EQ ← $temp_{m+1:64}$ == 0
	  CR0 ← LT || GT || EQ || SO						

其它寄存器	CA CR0 (if RC==1) SO OV (if OE==1)						

3.9 addze: 有符号加 0 及 CA 同时修改 CA XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	00000	OE	011001010	Rc
	6	5	5	5	1	9	1
格式	addze rD,rA (OE=0,Rc=0) addze. rD,rA (OE=0,Rc=1) addzeo rD,rA (OE=1,Rc=0) addzeo. rD,rA (OE=1,Rc=1)						

操作	$temp_{m:64}$ ← (GPR[rA]$_m$||GPR[rA]$_{m:63}$) + CA
	GPR[rD] ← $temp_{m+1:64}$
	CA ← $temp_{m+1}$
	if OE==1 then
	  OV ← $temp_{m+1}$ XOR $temp_m$
	  SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	  LT ← $temp_{m+1:64}$ < 0
	  GT ← $temp_{m+1:64}$ > 0
	  EQ ← $temp_{m+1:64}$ == 0
	  CR0 ← LT || GT || EQ || SO						

其它寄存器	CA CR0 (if RC==1) SO OV (if OE==1)						

3.10 and: 与 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0000011100	Rc
	6	5	5	5	10	1
格式	and rA,rS,rB (Rc=0) and. rA,rS,rB (Rc=1)					

操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ & GPR[rB]$_{m:63}$
	GPR[rA] ← $temp_{m:63}$
	if Rc==1 then
	  LT ← $temp_{m:63}$ < 0
	  GT ← $temp_{m:63}$ > 0
	  EQ ← $temp_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO					

其它寄存器	CR0 (if RC==1)					

3.11 andc: 与反码 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31

	011111	rS	rA	rB	0000111100	Rc
	6	5	5	5	10	1
格式	andc rA,rS,rB (Rc=0) andc. rA,rS,rB (Rc=1)					
操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ & ~(GPR[rB]$_{m:63}$) GPR[rA] ← $temp_{m:63}$ if Rc==1 then LT ← $temp_{m:63}$ < 0 GT ← $temp_{m:63}$ > 0 EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					
其它寄存器	CR0 (if RC==1)					

3.12 andi.: 与立即数 D-Form

编码	0 5	6 10	11 15	16 31
	011100	rS	rA	UIMM
	6	5	5	16
格式	andi. rS,rA,UIMM			
操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ & (0^{48-m}||UIMM) GPR[rA] ← $temp_{m:63}$ LT ← $temp_{m:63}$ < 0 GT ← $temp_{m:63}$ > 0 EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO			
其它寄存器	CR0			

3.13 andis.: 与移位立即数 D-Form

编码	0 5	6 10	11 15	16 31
	011101	rS	rA	UIMM
	6	5	5	16
格式	andis. rS,rA,UIMM			
操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ & (0^{32-m}||UIMM||0^{16}) GPR[rA] ← $temp_{m:63}$ LT ← $temp_{m:63}$ < 0 GT ← $temp_{m:63}$ > 0 EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO			
其它寄存器	CR0			

3.14 b: 绝对分支立即数 I-Form

	0 5	6 29	30	31
编码	010010	LI	AA	LK
	6	24	1	1

格式	b LI	(AA=0,LK=0)
	ba LI	(AA=1,LK=0)
	bl LI	(AA=0,LK=1)
	bla LI	(AA=1,LK=1)
操作	if AA==1 then a ← 0^{32} else a ← PC NPC ← a + EXTS(LI||00) if LK==1 then LR ← PC + 4	
其它寄存器	LR (if LK==1)	

3.15 bc: 条件分支立即数 B-Form

	0 5	6 10	11 15	16 29	30	31
编码	010000	BO	BI	BD	AA	LK
	6	5	5	14	1	1

格式	bc BO,BI,BD	(AA=0,LK=0)
	bca BO,BI,BD	(AA=1,LK=0)
	bcl BO,BI,BD	(AA=0,LK=1)
	bcla BO,BI,BD	(AA=1,LK=1)

操作	if BO_2==0 then CTR ← CTR − 1
	ctr_ok ← BO_2 | (($CTR_{m:63}$ ≠ 0) XOR BO_3)
	cond_ok ← BO_0 | (CR_{BI} == BO_1)
	if ctr_ok & cond_ok then
	  if AA==1 then a ← 0^{32} else a ← PC
	  NPC ← a + EXTS(BD||00)
	else
	  NPC ← PC + 4
	if LK==1 then
	  LR ← PC + 4	

其它寄存器	CTR (if BO_2==0) LR (if LK==1)	

3.16 bcctr: 分支 CTR XL-Form

	0 5	6 10	11 15	16 18	19 20	21 30	31
编码	010011	BO	BI	000	BH	1000010000	LK
	6	5	5	3	2	10	1

格式	bcctr BO,BI,BH	(LK=0)

	bcctr. BO,BI,BH (LK=1)

操作	cond_ok ← BO_0 | (CR_{BI} == BO_1)
	if cond_ok then
	  NPC ← $CTR_{m:61}$ || 00
	else
	  NPC ← PC + 4
	if LK==1 then
	  LR ← PC + 4

其它寄存器	LR (if LK==1)

3.17 bclr: 分支寄存器 XL-Form

编码	0 5	6 10	11 15	16 18	19 20	21 30	31
	010011	BO	BI	000	BH	0000010000	LK
	6	5	5	3	2	10	1

格式	bclr BO,BI,BH (LK=0) bclr. BO,BI,BH (LK=1)

操作	if BO_2==0 then CTR ← CTR − 1
	ctr_ok ← BO_2 | (($CTR_{m:63}$ ≠ 0) XOR BO_3)
	cond_ok ← BO_0 | (CR_{BI} == BO_1)
	if ctr_ok & cond_ok then
	  NPC ← $LR_{m:61}$ || 00
	else
	  NPC ← PC + 4
	if LK==1 then
	  LR ← PC + 4

其它寄存器	CTR (if BO_2==0) LR (if LK==1)

3.18 cmp: 有符号比较寄存器 X-Form

编码	0 5	6 8	9	10	11 15	16 20	21 30	31
	011111	BF	0	L	rA	rB	0000000000	0
	6	3	1	1	5	5	10	1

格式	cmp BF,L,rA,RB

操作	if L==0 then
	  a ← EXTS(GPR[rA]$_{32:63}$)
	  b ← EXTS(GPR[rB]$_{32:63}$)
	else
	  a ← GPR[rA]
	  b ← GPR[rB]
	if a < b then c ← 100
	if a > b then c ← 010

	if a == b then c ← 001 $CR_{4*BF:4*BF+3}$ ← c || SO
其它寄存器	CR

3.19 cmpi: 有符号比较立即数 D-Form

编码	0 5	6 8	9	10	11 15	16 31
	001011	BF	0	L	rA	SIMM
	6	3	1	1	5	16
格式	cmpi BF,L,rA,SIMM					

操作	if L==0 then
	a ← EXTS(GPR[rA]$_{32:63}$)
	else
	a ← GPR[rA]
	b ← EXTS(SIMM)
	if a < b then c ← 100
	if a > b then c ← 010
	if a == b then c ← 001
	$CR_{4*BF:4*BF+3}$ ← c || SO					

其它寄存器	CR					

3.20 cmpl: 无符号比较寄存器 X-Form

编码	0 5	6 8	9	10	11 15	16 20	21 30	31
	011111	BF	0	L	rA	rB	0000100000	0
	6	3	1	1	5	5	10	1
格式	cmpl BF,L,rA,RB							

操作	if L==0 then
	a ← 0^{32} || GPR[rA]$_{32:63}$
	b ← 0^{32} || GPR[rB]$_{32:63}$
	else
	a ← GPR[rA]
	b ← GPR[rB]
	if a $<_u$ b then c ← 100
	if a $>_u$ b then c ← 010
	if a == b then c ← 001
	$CR_{4*BF:4*BF+3}$ ← c || SO							

其它寄存器	CR							

3.21 cmpli: 无符号比较立即数 D-Form

编码	0 5	6 8	9	10	11 15	16 31
	001011	BF	0	L	rA	UIMM
	6	3	1	1	5	16
格式	cmpli BF,L,rA,UIMM					

操作	if L==0 then
	a ← 0^{32} || GPR[rA]$_{32:63}$
	else
	a ← GPR[rA]
	b ← 0^{48-m} || UIMM
	if a $<_u$ b then c ← 100
	if a $>_u$ b then c ← 010
	if a == b then c ← 001
	CR$_{4*BF:4*BF+3}$ ← c || SO					

其它寄存器	CR					

3.22 cntlzw: 计算字前导 0 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	00000	0000011010	Rc
	6	5	5	5	10	1
格式	cntlzw rA,rS (Rc=0) cntlzw. rA,rS (Rc=1)					

操作	n ← 32
	while n < 64 do
	if GPR[rS]$_n$ == 1 then break
	n ← n + 1
	temp ← n - 32
	GPR[rA] ← temp
	if Rc==1 then
	LT ← temp < 0
	GT ← temp > 0
	EQ ← temp == 0
	CR0 ← LT || GT || EQ || SO					

其它寄存器	CR0 (if RC==1)					

3.23 crand: CR 与 XL-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	010011	BT	BA	BB	0100000001	0
	6	5	5	5	10	1
格式	crand BT,BA,BB					
操作	CR$_{BT}$ ← CR$_{BA}$ & CR$_{BB}$					

其它 寄存器	CR

3.24 crnor: CR 或非 XL-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	010011	BT	BA	BB	0000100001	0
	6	5	5	5	10	1
格式	crnor BT,BA,BB					
操作	$CR_{BT} \leftarrow CR_{BA}$ NOR CR_{BB}					
其它 寄存器	CR					

3.25 crxor: CR 异或 XL-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	010011	BT	BA	BB	0011000001	0
	6	5	5	5	10	1
格式	crxor BT,BA,BB					
操作	$CR_{BT} \leftarrow CR_{BA}$ XOR CR_{BB}					
其它 寄存器	CR					

3.26 divw: 字有符号除 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	111101011	Rc
	6	5	5	5	1	9	1
格式	divw rD,rA,rB (OE=0,Rc=0) divw. rD,rA,rB (OE=0,Rc=1) divwo rD,rA,rB (OE=1,Rc=0) divwo. rD,rA,rB (OE=1,Rc=1)						

操作	dividend ← $GPR[rA]_{32:63}$
	divisor ← $GPR[rB]_{32:63}$
	quotient ← dividend / divisor
	if (quotient==-2^{31} & divisor==-1) | (divisor==0) then
	  quotient ← undefined
	  if OE==1 then
	  OV ← 1
	  SO ← SO | OV
	  if Rc==1 then
	  LT ← undefined
	  GT ← undefined
	  EQ ← undefined						

	CR0 ← LT || GT || EQ || SO
	else
	if OE==0 then
	OV ← 0
	SO ← SO | OV
	if Rc==1 then
	LT ← quotient < 0
	GT ← quotient > 0
	EQ ← quotient == 0
	CR0 ← LT || GT || EQ || SO
	GPR[rD] ← undefined || quotient

其它寄存器	CR0 (if RC==1) SO OV (if OE==1)

3.27 divwu: 字无符号除 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	111001011	Rc
	6	5	5	5	1	9	1

格式	divwu rD,rA,rB (OE=0,Rc=0) divwu. rD,rA,rB (OE=0,Rc=1) divwuo rD,rA,rB (OE=1,Rc=0) divwuo. rD,rA,rB (OE=1,Rc=1)

操作	dividend ← GPR[rA]$_{32:63}$
	divisor ← GPR[rB]$_{32:63}$
	quotient ← dividend / divisor
	if divisor==0 then
	quotient ← undefined
	if OE==1 then
	OV ← 1
	SO ← SO | OV
	if Rc==1 then
	LT ← undefined
	GT ← undefined
	EQ ← undefined
	CR0 ← LT || GT || EQ || SO
	else
	if OE==0 then
	OV ← 0
	SO ← SO | OV
	if Rc==1 then
	LT ← quotient < 0
	GT ← quotient > 0
	EQ ← quotient == 0

	CR0 ← LT || GT || EQ || SO GPR[rD] ← undefined || quotient
其它 寄存器	CR0 (if RC==1) SO OV (if OE==1)

3.28 extsb: 有符号拓展字节 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	00000	1110111010	Rc
	6	5	5	5	10	1
格式	extsb rA,rS (Rc=0) extsb. rA,rS (Rc=1)					
操作	s ← GPR[rS]$_{56}$ temp$_{m:63}$ ← s^{56-m} || GPR[rS]$_{56:63}$ GPR[rA] ← temp$_{m:63}$ if Rc==1 then LT ← temp$_{m:63}$ < 0 GT ← temp$_{m:63}$ > 0 EQ ← temp$_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					
其它 寄存器	CR0 (if RC==1)					

3.29 extsh: 有符号拓展半字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	00000	1110011010	Rc
	6	5	5	5	10	1
格式	extsh rA,rS (Rc=0) extsh. rA,rS (Rc=1)					
操作	s ← GPR[rS]$_{48}$ temp$_{m:63}$ ← s^{48-m} || GPR[rS]$_{48:63}$ GPR[rA] ← temp$_{m:63}$ if Rc==1 then LT ← temp$_{m:63}$ < 0 GT ← temp$_{m:63}$ > 0 EQ ← temp$_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					
其它 寄存器	CR0 (if RC==1)					

3.30 lbz: 无符号装载字节(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16	31

	100010	rD	rA	D
	6	5	5	16
格式	lbz rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← 0^{56-m} || MEM(EA, 1)			
其它寄存器				

3.31 lbzu: 无符号装载字节并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31
	100011	rD	rA	D
	6	5	5	16
格式	lbzu rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← 0^{56-m} || MEM(EA, 1) GPR[rA] ← EA			
其它寄存器				

3.32 lbzx: 无符号装载字节(寄存器作偏移量) X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	0001010111	0
	6	5	5	5	10	1
格式	lbzx rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ GPR[rD] ← 0^{56-m} || MEM(EA, 1)					
其它寄存器						

3.33 lha: 有符号装载半字(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16 31
	101010	rD	rA	D
	6	5	5	16
格式	lha rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← EXTS(MEM(EA, 2))$_{m:63}$			

其它 寄存器	

3.34 lhau: 有符号装载半字并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31
	101011	rD	rA	D
	6	5	5	16
格式	lhau rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← EXTS(MEM(EA, 2))$_{m:63}$ GPR[rA] ← EA			
其它 寄存器				

3.35 lhax: 有符号装载半字(寄存器作偏移量) X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	0101010111	0
	6	5	5	5	10	1
格式	lhax rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ GPR[rD] ← EXTS(MEM(EA, 2))$_{m:63}$					
其它 寄存器						

3.36 lhbrx: 无符号字节逆向装载半字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	1100010110	0
	6	5	5	5	10	1
格式	lhbrx rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ $temp_{0:15}$ = MEM(EA, 2) GPR[rD] ← 0^{48-m} || $temp_{8:15}$ || $temp_{0:7}$					
其它 寄存器						

3.37 lhz: 无符号装载半字(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16 31
	101000	rD	rA	D
	6	5	5	16
格式	lhz rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← 0^{48-m} || MEM(EA, 2)			
其它寄存器				

3.38 lhzu: 无符号装载半字并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31
	101001	rD	rA	D
	6	5	5	16
格式	lhzu rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← 0^{48-m} || MEM(EA, 2) GPR[rA] ← EA			
其它寄存器				

3.39 lhzx: 无符号装载半字(寄存器作偏移量) X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	0100010111	0
	6	5	5	5	10	1
格式	lhzx rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ GPR[rD] ← 0^{48-m} || MEM(EA, 2)					
其它寄存器						

3.40 lmw: 装载多字 D-Form

编码	0 5	6 10	11 15	16 31
	101110	rD	rA	D
	6	5	5	16
格式	lmw rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$			

	r ← rD do while r <= 31 GPR[r] ← 0^{32-m} || MEM(EA, 4) r ← r + 1 EA ← EA + 4
其它寄存器	

3.41 lwarx: 无符号原子装载字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	0000010100	EH
	6	5	5	5	10	1
格式	lwarx rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + $GPR[rB]_{m:63}$ RESERVE ← 1 RESERVE_LENGTH ← 1 RESERVE_ADDR ← real_addr(EA) GPR[rD] ← 0^{32-m} || MEM(EA, 4)					
其它寄存器						

3.42 lwbrx: 无符号字节逆向装载字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	1000010110	0
	6	5	5	5	10	1
格式	lwbrx rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + $GPR[rB]_{m:63}$ $temp_{0:31}$ = MEM(EA, 4) GPR[rD] ← 0^{32-m} || $temp_{24:31}$ || $temp_{16:23}$ || $temp_{8:15}$ || $temp_{0:7}$					
其它寄存器						

3.43 lwz: 无符号装载字(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16 31
	100000	rD	rA	D
	6	5	5	16
格式	lwz rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA]			

	EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← 0^{32-m} || MEM(EA, 4)
其它寄存器	

3.44 lwzu: 无符号装载字并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31
	100001	rD	rA	D
	6	5	5	16
格式	lwzu rD,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ GPR[rD] ← 0^{32-m} || MEM(EA, 4) GPR[rA] ← EA			
其它寄存器				

3.45 lwzx: 无符号装载字(寄存器作偏移量) X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rD	rA	rB	0000010111	0
	6	5	5	5	10	1
格式	lwzx rD,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ GPR[rD] ← 0^{32-m} || MEM(EA, 4)					
其它寄存器						

3.46 mcrf: CR 域移动 XL-Form

编码	0 5	6 8	9 10	11 13	14 15	16 20	21 30	31
	010011	BF	00	BFA	00	00000	0000000000	0
	6	3	2	3	2	5	10	1
格式	mcrf BF,BFA							
操作	$CR_{4*BF:4*BF+3}$ ← $CR_{4*BFA:4*BFA+3}$							
其它寄存器	CR							

3.47 mfcr: 从 CR 移动到 GPR XFX-Form

编码	0 5	6 10	11 20	21 30	31

	011111	rD	0000000000	0000010011	0
	6	5	10	10	1
格式	mfcr rD				
操作	GPR[rD] ← 0^{32-m} || CR				
其它寄存器					

3.48 mfmsr: 从 MSR 移动到 GPR X-Form

编码	0 5	6 10	11 20	21 30	31
	011111	rD	0000000000	0001010011	0
	6	5	10	10	1
格式	mfmsr rD				
操作	GPR[rD] ← 0^{32-m} || $MSR_{32:63}$				
其它寄存器					

3.49 mfspr: 从 SPR 移动到 GPR XFX-Form

编码	0 5	6 10	11 20	21 30	31
	011111	rD	SPRN	0101010011	0
	6	5	10	10	1
格式	mfspr rD,SPRN				

操作	sprn ← $SPRN_{5:9}$ || $SPRN_{0:4}$
	if SPR[sprn] use a 64-bit SPR then
	GPR[rD] ← SPR[sprn]
	else
	GPR[rD] ← 0^{32} || SPR[sprn]				

其它寄存器					

3.50 mtcrf: 从 GPR 移动到 CR XFX-Form

编码	0 5	6 10	11	12 19	20	21 30	31
	011111	rS	0	FXM	0	0010010000	0
	6	5	1	8	1	10	1
格式	mtcrf FXM,rS						
操作	i ← 0 do while i < 8 if FXM_i==1 then $CR_{4*i:4*i+3}$ ← $GPR[rS]_{4*i+32:4*i+35}$ i ← i + 1						
其它	CR						

寄存器	

3.51 mtmsr: 从 GPR 移动到 MSR X-Form

编码	0 5	6 10	11 20	21 30	31
	011111	rS	0000000000	0010010010	0
	6	5	10	10	1
格式	mtmsr rS				
操作	MSR ← GPR[rS]				
其它寄存器	MSR				

3.52 mtspr: 从 GPR 移动到 SPR XFX-Form

编码	0 5	6 10	11 20	21 30	31
	011111	rS	SPRN	0111010011	0
	6	5	10	10	1
格式	mtspr SPRN,rS				

操作	sprn ← $SPRN_{5:9}$ || $SPRN_{0:4}$
	if SPR[sprn] use a 64-bit SPR then
	SPR[sprn] ← GPR[rS]
	else
	SPR[sprn] ← $GPR[rS]_{32:63}$				

其它寄存器	SPR				

3.53 mulhw: 字有符号乘 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	0	001001011	Rc
	6	5	5	5	1	9	1
格式	mulhw rD,rA,rB (Rc=0) mulhw. rD,rA,rB (Rc=1)						

操作	$prod_{0:63}$ ← $GPR[rA]_{32:63}$ * $GPR[rB]_{32:63}$
	GPR[rD] ← $prod_{m:63}$
	if Rc==1 then
	if m==32 then
	LT ← $prod_{m:63}$ < 0
	GT ← $prod_{m:63}$ > 0
	EQ ← $prod_{m:63}$ == 0
	CR0 ← LT || GT || EQ || SO
	else
	CR0 ← $undefined^{3}$ || SO						

其它寄存器	CR0 (if RC==1)

3.54 mulhwu: 字无符号乘 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	0	000001011	Rc
	6	5	5	5	1	9	1
格式	mulhwu rD,rA,rB (Rc=0) mulhwu. rD,rA,rB (Rc=1)						

操作	$prod_{0:63} \leftarrow GPR[rA]_{32:63} *_u GPR[rB]_{32:63}$
	$GPR[rD] \leftarrow prod_{m:63}$
	if Rc==1 then
	  if m==32 then
	  LT ← $prod_{m:63}$ < 0
	  GT ← $prod_{m:63}$ > 0
	  EQ ← $prod_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO
	  else
	  CR0 ← undefined[3] || SO						

其它寄存器	CR0 (if RC==1)						

3.55 mulli: 立即数有符号乘 D-Form

编码	0 5	6 10	11 15	16 31
	000111	rD	rA	SIMM
	6	5	5	16
格式	mulli rD,rA,SIMM			
操作	$prod_{m+m:127} \leftarrow GPR[rA]_{m:63} * EXTS(SIMM)_{m:63}$ $GPR[rD] \leftarrow prod_{m+m:127}$			
其它寄存器				

3.56 mullw: 字有符号低位乘 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	011101011	Rc
	6	5	5	5	1	9	1
格式	mullw rD,rA,rB (OE=0,Rc=0) mullw. rD,rA,rB (OE=0,Rc=1) mullwo rD,rA,rB (OE=1,Rc=0) mullwo. rD,rA,rB (OE=1,Rc=1)						
操作	$prod_{0:63} \leftarrow GPR[rA]_{32:63} * GPR[rB]_{32:63}$						

	GPR[rD] ← $prod_{m:63}$
	if OE==1 then
	  OV ← ($prod_{0:31}$ ≠ 0^{32}) & ($prod_{0:31}$ ≠ 1^{32})
	  SO ← SO | ($prod_{0:31}$ ≠ 0^{32}) & ($prod_{0:31}$ ≠ 1^{32})
	if Rc==1 then
	  LT ← $prod_{m:63}$ < 0
	  GT ← $prod_{m:63}$ > 0
	  EQ ← $prod_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO

其它寄存器	CR0 (if RC==1) SO OV (if OE==1)

3.57 nand: 与非 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0111011100	Rc
	6	5	5	5	10	1
格式	nand rA,rS,rB (Rc=0) nand. rA,rS,rB (Rc=1)					

操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ NAND GPR[rB]$_{m:63}$
	GPR[rA] ← $temp_{m:63}$
	if Rc==1 then
	  LT ← $temp_{m:63}$ < 0
	  GT ← $temp_{m:63}$ > 0
	  EQ ← $temp_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO					

其它寄存器	CR0 (if RC==1)					

3.58 neg: 取补 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	00000	OE	001101000	Rc
	6	5	5	5	1	9	1
格式	neg rD,rA (OE=0,Rc=0) neg. rD,rA (OE=0,Rc=1) nego rD,rA (OE=1,Rc=0) nego. rD,rA (OE=1,Rc=1)						

操作	$temp_{m:63}$ ← ~GPR[rA]$_{m:63}$ + 1
	GPR[rA] ← $temp_{m:63}$
	if OE==1 then
	  if m==32 then
	  if GPR[rA]$_{32:63}$==0x8000_0000 then
	  OV ← 1						

	SO ← SO | 1
	if m==64 then
	if GPR[rA]$_{0:63}$==0x8000_0000_0000_0000 then
	OV ← 1
	SO ← SO | 1
	if Rc==1 then
	LT ← temp$_{m:63}$ < 0
	GT ← temp$_{m:63}$ > 0
	EQ ← temp$_{m:63}$ == 0
	CR0 ← LT || GT || EQ || SO

其它寄存器	CR0 (if RC==1) SO OV (if OE==1)

3.59 nor: 或非 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0001111100	Rc
	6	5	5	5	10	1
格式	nor rA,rS,rB (Rc=0) nor. rA,rS,rB (Rc=1)					
操作	temp$_{m:63}$ ← GPR[rS]$_{m:63}$ NOR GPR[rB]$_{m:63}$ GPR[rA] ← temp$_{m:63}$ if Rc==1 then LT ← temp$_{m:63}$ < 0 GT ← temp$_{m:63}$ > 0 EQ ← temp$_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					
其它寄存器	CR0 (if RC==1)					

3.60 or: 或 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0110111100	Rc
	6	5	5	5	10	1
格式	or rA,rS,rB (Rc=0) or. rA,rS,rB (Rc=1)					
操作	temp$_{m:63}$ ← GPR[rS]$_{m:63}$ | GPR[rB]$_{m:63}$ GPR[rA] ← temp$_{m:63}$ if Rc==1 then LT ← temp$_{m:63}$ < 0 GT ← temp$_{m:63}$ > 0 EQ ← temp$_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					

其它寄存器	CR0 (if RC==1)

3.61 orc: 或反 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0110011100	Rc
	6	5	5	5	10	1
格式	orc rA,rS,rB (Rc=0) orc. rA,rS,rB (Rc=1)					
操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ | ~GPR[rB]$_{m:63}$ GPR[rA] ← $temp_{m:63}$ if Rc==1 then LT ← $temp_{m:63}$ < 0 GT ← $temp_{m:63}$ > 0 EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					
其它寄存器	CR0 (if RC==1)					

3.62 ori: 或立即数 D-Form

编码	0 5	6 10	11 15	16 31
	011000	rS	rA	UIMM
	6	5	5	16
格式	ori rA,rS,UIMM			
操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ | (0^{48-m}||UIMM) GPR[rA] ← $temp_{m:63}$			
其它寄存器				

3.63 oris: 或移位立即数 D-Form

编码	0 5	6 10	11 15	16 31
	011001	rS	rA	UIMM
	6	5	5	16
格式	oris rA,rS,UIMM			
操作	$temp_{m:63}$ ← GPR[rS]$_{m:63}$ | (0^{32-m}||UIMM||0^{16}) GPR[rA] ← $temp_{m:63}$			
其它寄存器				

3.64 rfi: 从中断返回 XL-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	010011	00000	00000	00000	0000110010	0
	6	5	5	5	10	1
格式	rfi					
操作	if SRR1[CM]==0 then z ← 32 else z ← 0 NPC ← 0^{z} || $SRR0_{z:61}$ || 00 MSR ← SRR1					
其它寄存器	MSR					

3.65 rlwimi: 左旋立即数并插屏蔽 M-Form

编码	0 5	6 10	11 15	16 20	21 25	26 30	31
	010100	rS	rA	SH	MB	ME	Rc
	6	5	5	5	5	5	1
格式	rlwimi rA,rS,SH,MB,ME (Rc=0) rlwimi. rA,rS,SH,MB,ME (Rc=1)						

操作	n ← SH
	b ← MB + 32
	e ← ME + 32
	r ← $ROTL_{32}$(GPR[rS]$_{32:63}$, n)
	k ← MASK(b, e)
	$temp_{m:63}$ ← (r & k) | (GPR[rA]$_{m:63}$ & ~k)
	GPR[rA] ← $temp_{m:63}$
	if Rc==1 then
	  LT ← $temp_{m:63}$ < 0
	  GT ← $temp_{m:63}$ > 0
	  EQ ← $temp_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO						

其它寄存器	CR0 (if RC==1)						

3.66 rlwinm: 左旋立即数并与屏蔽 M-Form

编码	0 5	6 10	11 15	16 20	21 25	26 30	31
	010101	rS	rA	SH	MB	ME	Rc
	6	5	5	5	5	5	1
格式	rlwinm rA,rS,SH,MB,ME (Rc=0) rlwinm. rA,rS,SH,MB,ME (Rc=1)						
操作	n ← SH b ← MB + 32 e ← ME + 32 r ← $ROTL_{32}$(GPR[rS]$_{32:63}$, n) k ← MASK(b, e)						

	$temp_{m:63}$ ← r & k
	GPR[rA] ← $temp_{m:63}$
	if Rc==1 then
	  LT ← $temp_{m:63}$ < 0
	  GT ← $temp_{m:63}$ > 0
	  EQ ← $temp_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO

其它寄存器	CR0 (if RC==1)

3.67 rlwnm: 左旋寄存器并与屏蔽 M-Form

编码	0 5	6 10	11 15	16 20	21 25	26 30	31
	010101	rS	rA	rB	MB	ME	Rc
	6	5	5	5	5	5	1
格式	rlwnm rA,rS,rB,MB,ME (Rc=0) rlwnm. rA,rS,rB,MB,ME (Rc=1)						

操作	n ← GPR[rB]$_{59:63}$
	b ← MB + 32
	e ← ME + 32
	r ← ROTL$_{32}$(GPR[rS]$_{32:63}$, n)
	k ← MASK(b, e)
	$temp_{m:63}$ ← r & k
	GPR[rA] ← $temp_{m:63}$
	if Rc==1 then
	  LT ← $temp_{m:63}$ < 0
	  GT ← $temp_{m:63}$ > 0
	  EQ ← $temp_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO						

其它寄存器	CR0 (if RC==1)						

3.68 SC: 系统调用 SC-Form

编码	0 5	6 10	11 15	16 19	20 26	27 29	30	31
	01001	00000	00000	0000	LEV	000	1	0
	6	5	5	4	7	3	1	1
格式	SC SC LEV							
操作								
其它寄存器								

3.69 slw: 逻辑左移 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0000011000	Rc
	6	5	5	5	10	1
格式	slw rA,rS,rB (Rc=0) slw. rA,rS,rB (Rc=1)					

操作	n ← GPR[rB]$_{59:63}$
	r ← ROTL$_{32}$(GPR[rS]$_{32:63}$, n)
	if GPR[rB]$_{58}$==0 then
	  k ← MASK(32, 63-n)
	else
	  k ← 0^{64-m}
	temp$_{m:63}$ ← r & k
	GPR[rA] ← temp$_{m:63}$
	if Rc==1 then
	  LT ← temp$_{m:63}$ < 0
	  GT ← temp$_{m:63}$ > 0
	  EQ ← temp$_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO					

其它寄存器	CR0 (if RC==1)					

3.70 sraw: 算术右移 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	1100011000	Rc
	6	5	5	5	10	1
格式	sraw rA,rS,rB (Rc=0) sraw. rA,rS,rB (Rc=1)					

操作	n ← GPR[rB]$_{59:63}$
	r ← ROTL$_{32}$(GPR[rS]$_{32:63}$, 64-n)
	if GPR[rB]$_{58}$==0 then
	  k ← MASK(n+32, 63)
	else
	  k ← 0^{64-m}
	s ← GPR[rS]$_{32}$
	temp$_{m:63}$ ← (r & k) | (s^{64} & ~k)
	GPR[rA] ← temp$_{m:63}$
	CA ← s & ((r & ~k)$_{m:63}$ ≠ 0)
	if Rc==1 then
	  LT ← temp$_{m:63}$ < 0
	  GT ← temp$_{m:63}$ > 0
	  EQ ← temp$_{m:63}$ == 0					

	CR0 ← LT || GT || EQ || SO
其它 寄存器	CA CR0 (if RC==1)

3.71 srawi: 算术右移立即数 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	SH	1100111000	Rc
	6	5	5	5	10	1
格式	srawi rA,rS,SH (Rc=0) srawi. rA,rS,SH (Rc=1)					

操作	n ← SH
	r ← $ROTL_{32}$(GPR[rS]$_{32:63}$, 64-n)
	k ← MASK(n+32, 63)
	s ← GPR[rS]$_{32}$
	temp$_{m:63}$ ← (r & k) | (s^{64} & ~k)
	GPR[rA] ← temp$_{m:63}$
	CA ← s & ((r & ~k)$_{m:63}$ ≠ 0)
	if Rc==1 then
	  LT ← temp$_{m:63}$ < 0
	  GT ← temp$_{m:63}$ > 0
	  EQ ← temp$_{m:63}$ == 0
	  CR0 ← LT || GT || EQ || SO					

其它 寄存器	CA CR0 (if RC==1)					

3.72 srw: 逻辑右移 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	1000011000	Rc
	6	5	5	5	10	1
格式	srw rA,rS,rB (Rc=0) srw. rA,rS,rB (Rc=1)					

操作	n ← GPR[rB]$_{59:63}$
	r ← $ROTL_{32}$(GPR[rS]$_{32:63}$, 64-n)
	if GPR[rB]$_{58}$==0 then
	  k ← MASK(n+32, 63)
	else
	  k ← 0^{64-m}
	temp$_{m:63}$ ← r & k
	GPR[rA] ← temp$_{m:63}$
	if Rc==1 then
	  LT ← temp$_{m:63}$ < 0
	  GT ← temp$_{m:63}$ > 0					

	EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO
其它寄存器	CR0 (if RC==1)

3.73 stb: 存储字节(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16 31	
	100110	rS	rA	D	
	6	5	5	16	
格式	stb rS,D(rA)				
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ MEM(EA, 1) ← GPR[rS]$_{56:63}$				
其它寄存器					

3.74 stbu: 存储字节并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31	
	100111	rS	rA	D	
	6	5	5	16	
格式	stbu rS,D(rA)				
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ MEM(EA, 1) ← GPR[rS]$_{56:63}$ GPR[rA] ← EA				
其它寄存器					

3.75 stbx: 存储字节(寄存器作偏移量) X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0011010111	0
	6	5	5	5	10	1
格式	stbx rS,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ MEM(EA, 1) ← GPR[rS]$_{56:63}$					
其它寄存器						

3.76 sth: 存储半字(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16 31
	101100	rS	rA	D
	6	5	5	16
格式	sth rS,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ MEM(EA, 2) ← GPR[rS]$_{48:63}$			
其它寄存器				

3.77 sthbrx: 字节逆向存储半字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	1110010110	0
	6	5	5	5	10	1
格式	sthbrx rS,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ MEM(EA, 2) ← GPR[rS]$_{56:63}$ || GPR[rS]$_{48:55}$					
其它寄存器						

3.78 sthu: 存储字并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31
	101101	rS	rA	D
	6	5	5	16
格式	sthu rS,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ MEM(EA, 2) ← GPR[rS]$_{48:63}$ GPR[rA] ← EA			
其它寄存器				

3.79 sthx: 存储字(寄存器作偏移量) X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0110010111	0
	6	5	5	5	10	1
格式	sthx rS,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$					

	MEM(EA, 2) ← GPR[rS]$_{48:63}$
其它寄存器	

3.80 stmw: 存储多字 D-Form

编码	0 5	6 10	11 15	16 31
	101111	rS	rA	D
	6	5	5	16
格式	stmw rS,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ r ← rS do while r <= 31 MEM(EA, 4) ← GPR[rS]$_{32:63}$ r ← r + 1 EA ← EA + 4			
其它寄存器				

3.81 stw: 存储字(立即数作偏移量) D-Form

编码	0 5	6 10	11 15	16 31
	100100	rS	rA	D
	6	5	5	16
格式	stw rS,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + EXTS(D)$_{m:63}$ MEM(EA, 4) ← GPR[rS]$_{32:63}$			
其它寄存器				

3.82 stwbrx: 字节逆向存储半字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	1010010110	0
	6	5	5	5	10	1
格式	stwbrx rS,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + GPR[rB]$_{m:63}$ MEM(EA, 4) ← GPR[rS]$_{56:63}$ || GPR[rS]$_{48:55}$ || GPR[rS]$_{40:47}$ || GPR[rS]$_{32:39}$					
其它						

寄存器	

3.83 stwcx.：原子存储字 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0010010110	1
	6	5	5	5	10	0
格式	stwcx. rS,rA,rB					

操作	if rA==0 then a ← 0^{64} else a ← GPR[rA]
	EA ← $a_{m:63}$ + $GPR[rB]_{m:63}$
	if RESERVE then
	  if RESERVE_LENGTH==4 then
	  if RESERVE_ADDR=real_addr(EA) then
	  perform_store ← 1
	  undefined_case ← 0
	  else
	  undefined_case ← 1
	  else
	  undefined_case ← 1
	else
	  undefined_case ← 0
	  perform_store ← 0
	if undefined_case then
	  u ← undefined_case
	  if u then MEM(EA, 4) ← $GPR[rS]_{32:63}$
	  CR0 ← 00 || u || SO
	else
	  if perform_store then MEM(EA, 4) ← $GPR[rS]_{32:63}$
	  CR0 ← 00 || perform_store || SO
	RESERVE ← 0					

其它寄存器	CR0					

3.84 stwu: 存储字并更新寄存器 D-Form

编码	0 5	6 10	11 15	16 31
	100101	rS	rA	D
	6	5	5	16
格式	stwu rS,D(rA)			
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + $EXTS(D)_{m:63}$ MEM(EA, 4) ← $GPR[rS]_{32:63}$ GPR[rA] ← EA			
其它				

寄存器	

3.85 stwx: 存储字(寄存器作偏移量) X-Form

	0 5	6 10	11 15	16 20	21 30	31
编码	011111	rS	rA	rB	0010010111	0
	6	5	5	5	10	1
格式	stwx rS,rA,rB					
操作	if rA==0 then a ← 0^{64} else a ← GPR[rA] EA ← $a_{m:63}$ + $GPR[rB]_{m:63}$ MEM(EA, 4) ← $GPR[rS]_{32:63}$					
其它寄存器						

3.86 subf: 逆向减 XO-Form

	0 5	6 10	11 15	16 20	21	22 30	31
编码	011111	rD	rA	rB	OE	000101000	Rc
	6	5	5	5	1	9	1
格式	subf rD,rA,rB					(OE=0,Rc=0)	
	subf. rD,rA,rB					(OE=0,Rc=1)	
	subfo rD,rA,rB					(OE=1,Rc=0)	
	subfo. rD,rA,rB					(OE=1,Rc=1)	

操作	$temp_{m:64}$ ← ~($GPR[rA]_m$||$GPR[rA]_{m:63}$) + ($GPR[rB]_m$||$GPR[rB]_{m:63}$) +1
	GPR[rD] ← $temp_{m+1:64}$
	if OE==1 then
	  OV ← $temp_{m+1}$ XOR $temp_m$
	  SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	  LT ← $temp_{m+1:64}$ < 0
	  GT ← $temp_{m+1:64}$ > 0
	  EQ ← $temp_{m+1:64}$ == 0
	  CR0 ← LT || GT || EQ || SO						

其它寄存器	CR0 (if RC==1) SO OV (if OE==1)						

3.87 subfc: 逆向减并修改 CA XO-Form

	0 5	6 10	11 15	16 20	21	22 30	31
编码	011111	rD	rA	rB	OE	000001000	Rc
	6	5	5	5	1	9	1
格式	subfc rD,rA,rB					(OE=0,Rc=0)	
	subfc. rD,rA,rB					(OE=0,Rc=1)	
	subfco rD,rA,rB					(OE=1,Rc=0)	

	subfco. rD,rA,rB (OE=1,Rc=1)

操作	$temp_{m:64} \leftarrow \sim(GPR[rA]_m||GPR[rA]_{m:63}) + (GPR[rB]_m||GPR[rB]_{m:63}) +1$
	$GPR[rD] \leftarrow temp_{m+1:64}$
	$CA \leftarrow temp_{m+1}$
	if OE==1 then
	  $OV \leftarrow temp_{m+1}$ XOR $temp_m$
	  $SO \leftarrow SO \mid temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	  $LT \leftarrow temp_{m+1:64} < 0$
	  $GT \leftarrow temp_{m+1:64} > 0$
	  $EQ \leftarrow temp_{m+1:64} == 0$
	  CR0 ← LT || GT || EQ || SO

其它寄存器	CA CR0 (if RC==1) SO OV (if OE==1)

3.88 subfe: 拓展逆向减 XO-Form

编码	0 5	6 10	11 15	16 20	21	22 30	31
	011111	rD	rA	rB	OE	010001000	Rc
	6	5	5	5	1	9	1

格式	subfe rD,rA,rB (OE=0,Rc=0) subfe. rD,rA,rB (OE=0,Rc=1) subfeo rD,rA,rB (OE=1,Rc=0) subfeo. rD,rA,rB (OE=1,Rc=1)

操作	$temp_{m:64} \leftarrow \sim(GPR[rA]_m||GPR[rA]_{m:63})+(GPR[rB]_m||GPR[rB]_{m:63})+CA$
	$GPR[rD] \leftarrow temp_{m+1:64}$
	$CA \leftarrow temp_{m+1}$
	if OE==1 then
	  $OV \leftarrow temp_{m+1}$ XOR $temp_m$
	  $SO \leftarrow SO \mid temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	  $LT \leftarrow temp_{m+1:64} < 0$
	  $GT \leftarrow temp_{m+1:64} > 0$
	  $EQ \leftarrow temp_{m+1:64} == 0$
	  CR0 ← LT || GT || EQ || SO

其它寄存器	CA CR0 (if RC==1) SO OV (if OE==1)

3.89 subfic: 逆向减立即数并修改 CA D-Form

编码	0 5	6 10	11 15	16 31
	001000	rD	rA	SIMM

	6	5	5	16
格式	subfic rD,rA,SIMM			
操作	$temp_{m:63}$ ←- ~GPR[rA]$_{m:63}$ + EXTS(SIMM)$_{m:63}$ + 1 GPR[rD] ← $temp_{m:63}$ CA ← $temp_m$			
其它寄存器	CA			

3.90 subfme: 拓展逆向减-1 XO-Form

	0 5	6 10	11 15	16 20	21	22 30	31
编码	011111	rD	rA	00000	OE	011101000	Rc
	6	5	5	5	1	9	1
格式	subfme rD,rA (OE=0,Rc=0) subfme. rD,rA (OE=0,Rc=1) subfmeo rD,rA (OE=1,Rc=0) subfmeo. rD,rA (OE=1,Rc=1)						

操作	$temp_{m:64}$ ← ~(GPR[rA]$_m$||GPR[rA]$_{m:63}$) + (1||0xFFFF..FF) + CA
	GPR[rD] ← $temp_{m+1:64}$
	CA ← $temp_{m+1}$
	if OE==1 then
	  OV ← $temp_{m+1}$ XOR $temp_m$
	  SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	  LT ← $temp_{m+1:64}$ < 0
	  GT ← $temp_{m+1:64}$ > 0
	  EQ ← $temp_{m+1:64}$ == 0
	  CR0 ← LT || GT || EQ || SO						

其它寄存器	CA CR0 (if RC==1) SO OV (if OE==1)						

3.91 subfze: 拓展逆向减 0 XO-Form

	0 5	6 10	11 15	16 20	21	22 30	31
编码	011111	rD	rA	00000	OE	011001000	Rc
	6	5	5	5	1	9	1
格式	subfze rD,rA (OE=0,Rc=0) subfze. rD,rA (OE=0,Rc=1) subfzeo rD,rA (OE=1,Rc=0) subfzeo. rD,rA (OE=1,Rc=1)						
操作	$temp_{m:64}$ ← ~(GPR[rA]$_m$||GPR[rA]$_{m:63}$) + CA GPR[rD] ← $temp_{m+1:64}$ CA ← $temp_{m+1}$						

	if OE==1 then
	OV ← $temp_{m+1}$ XOR $temp_m$
	SO ← SO | $temp_{m+1}$ XOR $temp_m$
	if Rc==1 then
	LT ← $temp_{m+1:64}$ < 0
	GT ← $temp_{m+1:64}$ > 0
	EQ ← $temp_{m+1:64}$ == 0
	CR0 ← LT || GT || EQ || SO

其它寄存器	CA CR0 (if RC==1) SO OV (if OE==1)

3.92 sync: 同步 X-Form

编码	0 5	6 8	9 10	11 15	16 20	21 30	31
	011111	000	L	00000	00000	1001010110	0
	6	3	2	5	5	10	1
格式	sync L						
操作							
其它寄存器							

3.93 twi: 陷阱立即数 D-Form

编码	0 5	6 10	11 15	16 31
	000011	TO	rA	SIMM
	6	5	5	16
格式	twi TO,rA,SIMM			

操作	a ← EXTS(GPR[rA]$_{32:63}$)
	b ← EXTS(SIMM)
	if (a < b) & TO_0 then TRAP
	if (a > b) & TO_1 then TRAP
	if (a == b) & TO_2 then TRAP
	if (a $<_u$ b) & TO_3 then TRAP
	if (a $>_u$ b) & TO_4 then TRAP			

其它寄存器				

3.94 wrteei: 立即数修改 MSR.EE X-Form

编码	0 5	6 10	11 15	16	17 20	21 30	31
	011111	00000	00000	E	0000	0010010111	0
	6	5	5	1	4	10	1
格式	wrteei E						

操作	$MSR_{EE} \leftarrow E$
其它寄存器	

3.95 xor: 异或 X-Form

编码	0 5	6 10	11 15	16 20	21 30	31
	011111	rS	rA	rB	0100111100	Rc
	6	5	5	5	10	1
格式	xor rA,rS,rB (Rc=0) xor. rA,rS,rB (Rc=1)					
操作	$temp_{m:63} \leftarrow GPR[rS]_{m:63}$ XOR $GPR[rB]_{m:63}$ $GPR[rA] \leftarrow temp_{m:63}$ if Rc==1 then LT ← $temp_{m:63}$ < 0 GT ← $temp_{m:63}$ > 0 EQ ← $temp_{m:63}$ == 0 CR0 ← LT || GT || EQ || SO					
其它寄存器	CR0 (if RC==1)					

3.96 xori: 异或立即数 D-Form

编码	0 5	6 10	11 15	16 31
	011010	rS	rA	UIMM
	6	5	5	16
格式	xori rA,rS,UIMM			
操作	$temp_{m:63} \leftarrow GPR[rS]_{m:63}$ XOR $(0^{48-m}\|\|UIMM)$ $GPR[rA] \leftarrow temp_{m:63}$			
其它寄存器				

4. 单周期 RTL 描述规则

通过指令 RTL 描述详细地刻画部件间数据的流向，构造 EXCEL 表，即可以通过分析多路选择情况，构造相关的多路选择器以及控制器，从而构造完整的处理器顶层模块。现针对单周期的 RTL 描述自动化工具做如下规则限定：

（1）部件间的数据流向必须通过“MODULE_NAME.PORT_NAME”制定部件名称，如 GPR 的 addr 端口需制定为 GPR.addr，同时使用“->”作为流动标志，箭头左侧代表源，右侧代表目的；

（2）部件内部的数据流向可以写出，也可以省略；

（3）针对附加的 assign 语句，如 assign rA = INSTR[11:15]，以#作为 assign 语句的标记，描述为”#INSTR[11:15] -> rA”（此时 rA 不属于任何 MODULE，因此无需指定模块）；

（4）针对部件定义好的控制信号，如写信号（描述为 XXX.XXXWr 或者 XXX.Wr 均可），工具生成代码时会自动合成为 XXXWr 的形式（XXX 为部件名），除写信号外的控制信号，如 ALU 的操作码控制信号，以 Op 作为后缀，即写为（ALU.ALUOp 或 ALU.Op），工具生成代码时会统一合称为 ALU.ALUOp；
（5）对于一些宏定义，请注明宏定义符号`；
（6）对于复杂的数据传输（存在 if-else 条件），请使用三目操作符”()?:”完成，可支持嵌套（但未测试超过 2 级以上的嵌套），对于数据源（MUX），工具会将不同条件的可能值进行分析（而非表达式本身）；针对控制，系统将保留原条件，将数值替换为可能的片选编码；
（7）需制定部件的模块接口定义，便于生成顶层模块时实例化各部件，其中必须包括指令的操作码宏定义文件（instruction_def.v，此文件仅包括 XO 与 OPCD 定义）。

5.