MXU 指令使用指南

本文介绍君正 MXU 多媒体加速指令的嵌入汇编使用方法及编译步骤。

1、MXU 指令的嵌入汇编使用方法

君正处理器（如 Jz4740）实现了 60 条 SIMD 指令用来进行多媒体编解码的优化，象我们常用的 MPEG4、H264、VC-1、RMVB 等多媒体解码算法都可以使用 MXU 指令来进行优化。通常情况下我们需要在 C 代码中以嵌入汇编的方式来使用 MXU 指令，这里我们就来介绍其使用方法。详细 MXU 指令请参考《Ingenic Media Extension Instruction Set》规范文档。

首先，我们已经以宏的方式定义好了 MXU 指令，并包含在一个头文件中。该文件定义了 17 个 MXU 寄存器（xr0 ~ xr16）和所有多媒体指令，详细内容请参考头文件 jz_mxh.h。下面举一个例子：

```
#define S32LDD(xra,rb,s12)                                  \
  do {                                                      \
    __asm__ __volatile ("S32LDD xr%0,%z1,%2"                \
            :                                               \
            :"K"(xra),"d" (rb),"I"(s12));                   \
  } while (0)
```

上面的宏定义了 S32LDD 指令，该指令包含三个操作域 xra, rb 和 s12，完成从内存[rb + s12]读取一个 word 数据到 xra（a = 1 ~ 16）寄存器。

为了介绍如何使用 MXU 指令的嵌入汇编进行编程，我们截取 XVID 中一个典型计算： 8X8 block 的 IDCT 残差加上对该 block 的运动补偿，相应的 C 代码和用 MXU 指令嵌入汇编优化的代码都列出，便于对照理解。

```
/*
 * SRC - the source buffer (NOTE: offset 0 must be 4-byte aligned)
 * DST - the destination buffer (NOTE: offset 0 always is 4-byte aligned)
 *
 * Then the function does the 16->8 bit transfer and this serie of operations :
 *
 *      SRC (16bit) = SRC
 *      DST (8bit)    = max(min(DST+SRC, 255), 0)
 */
////////////////////////////////////////////////////////
//原始的 C 代码
////////////////////////////////////////////////////////
void
```

```
transfer_16to8add_c(uint8_t * const dst,
                    const int16_t * const src,
                    uint32_t stride)
{
      uint32_t i, j;

      for (j = 0; j < 8; j++) {
            for (i = 0; i < 8; i++) {
                  int16_t pixel = (int16_t) dst[j * stride + i] + src[j * 8 + i];

                  if (pixel < 0) {
                        pixel = 0;
                  } else if (pixel > 255) {
                        pixel = 255;
                  }
                  dst[j * stride + i] = (uint8_t) pixel;
            }
      }
}

//////////////////////////////////////////////////////
//带 MXU 指令嵌入汇编的 C 代码
//////////////////////////////////////////////////////
void
transfer_16to8add_mxu(uint8_t * dst,
                    const int16_t * const src,
                    uint32_t stride)
{
   int32_t *src_data;
   int32_t i;

   src_data = (int32_t *)src - 1;
   dst -= stride;

   for (i = 0; i < 8; i++) {
      S32LDIV(xr5, dst, stride, 0);
      S32LDD(xr6, dst, 4);
      S32LDI(xr1, src_data, 4);
      S32LDI(xr2, src_data, 4);
      S32LDI(xr3, src_data, 4);
      S32LDI(xr4, src_data, 4);

      Q8ACCE_AA(xr2, xr5, xr0, xr1);
      Q8ACCE_AA(xr4, xr6, xr0, xr3);
```

```
        Q16SAT(xr5, xr2, xr1);
        Q16SAT(xr6, xr4, xr3);

        S32STD(xr5, dst, 0);
        S32STD(xr6, dst, 4);
    }
}
```

需要注意以下几点：

1）在使用 MXU 指令之前一定要开启 MXU 硬件单元，而不用时为了低功耗应关闭 MXU 硬件单元。一般可以在应用程序初始化阶段开启，应用程序退出前关闭。可以套用下面两个简单函数：

```
void mxu_open(void) {
    S32I2M(xr16, 3);   /*注意：这条开启指令之后需要执行至少三条非 MXU 指令来区隔后续的有效 MXU 指令*/
}
void mxu_close(void) {
    S32I2M(xr16, 0);   /*注意：这里的关闭立即生效，紧接其后的 MXU 运算型指令的执行会产生指令非法异常*/
}
```

2) 在使用了 MXU 嵌入汇编的.c/.h 文件中应该包含专门的头文件 jz_mxu.h。

2、编译步骤

君正提供给用户一个 awk 脚本 mxu_as，该脚本用来将 MXU 宏指令翻译成机器码。该脚本命令格式如下：

```
mxu_as src_file > target_file
```

其中 src_file 是指包含有 MXU 宏指令的源文件，target_file 是指翻译成机器码后的文件。我们举一个例子，如果源文件中有下面一条 MXU 宏指令：

```
S32LDD XR2,$31,4
```

通过 mxu_as 处理后，该指令将被翻译成如下机器码：

```
.word    0b01110011111100000000000100100100000    #S32LDD XR2,$31,4
```

具体编译步骤如下：

1) 将带有 MXU 嵌入汇编的 C/C++源文件（例如 mxu_test.c）编译成汇编格式文件：

mipsel-linux-gcc -O2 -S -o mxu_test.mid mxu_test.c

2) 用 mxu_as 这个 awk 脚本将 mxu_test.mid 中的 MXU 指令翻译成机器码：

mxu_as mxu_test.mid > mxu_test.s

3. 将 mxu_test.s 这个汇编格式文件编译成目标文件：

mipsel-linux-gcc -c -o mxu_test.o mxu_test.s

最后 mxu_test.o 就可以与其他目标文件链接生成最终的可执行文件了。

注：mxu_as 和 jz_mxu.h 已经被包含在了 MIPS 交叉工具链目录下。