MXU Instruction Usage Guide

This article introduces usage and compilation's steps of embedded assembly of MXU multimedia's acceleration instruction.

1 Embed compilation usage of MXU instruction

Ingenic processor (e.g jz4740) has implemented 60 SIMD instructions for the multimedia codec optimization, such as MPEG4、H264、VC-1、RMVB and so on, which can be optimized by MXU instructions. Normally, we use embedded assembly in the C code to use MXU instructions.
Here we have to introduce its usage. Please refer to 《Ingenic Media Extension Instruction Set》to get more MXU instructions.

First, we have defined the MXU instructions by the way of the Marco, and is included in a header file. This file defines 17 MXU registers (xr0 ~ xr16) and all multimedia instructions, please refer to jz_mxh.h.

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

Above example which defines S32LCD instruction including three operations filed xra, rb and s12, complete to read a word data from memory [rb + s12] to xra (a = 1 ~ 16) registers.

To introduce how to use the MXU embedded assembly instructions for programming, we intercept a typical calculation in XVID: 8X8 block's residuals added the motion compensation of block, the corresponding C code and optimized assembly code with MXU instructions listed, easy to understand.

```
/*
 * 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)
 */
////////////////////////////////////////////////////
// original C code
```

```
//////////////////////////////////////////////////
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;
                }
        }
}

//////////////////////////////////////////////////
// C code which includes MXU instruction embedded assembly
//////////////////////////////////////////////////
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);
   }
}
```

Need to note the following points:

1) before using the MXU instruction, must open the MXU hardware units, and when not in use, MXU hardware unit should be closed in order to keep the low power consumption. Generally open it in the application initialization phase, close it in he application withdraw. Can use the following two simple functions:

```
void mxu_open (void) (
S32I2M (xr16, 3); / * NOTE: after this open instruction, at least three non-MXU instructions are required to implement to segment the follow-up effective MXU instruction * /
)
void mxu_close (void) (
S32I2M (xr16, 0); / * Note: This closure is effective immediately, followed by MXU computing the instruction will produce the illegal instruction exception * /
)
```

2) should contain a special header file jz_mxu.h in the MXU compiled .c / .h file

2 Compilation steps

Ingenic provides an awk script mxu_as to the user, the script is used to translate MXU macro into machine code. The script command format is as follows:

```
mxu_as src_file> target_file
```

src_file contains the source files including MXU macro, target_file is file which is translated into machine code. We give an example, if the source file has the following 1 MXU macro:

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

.By mxu_as processed, the instruction will be translated into machine code. As follows:

```
. word   0b01110011111000000000010010010000   # S32LDD XR2, $ 31,4
```

Specific compilation steps are as follows:

1) Complile C / C + + source files with the MXU embedded assembly (for example, mxu_test.c) to assembly format:

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

2) The MXU instructions of mxu_test.mid was translated into machine code using mxu_as:

```
mxu_as mxu_test.mid> mxu_test.s
```

3. Compile Mxu_test.s into the target file format:

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

Finally mxu_test.o can be linked with other object files to generate the executable file.

Note: mxu_as and jz_mxu.h have been included in the MIPS cross-tool chain directory.