君正®

MXU User Guide

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

MXU User Guide

Release history

Date	Revision	Revision History
2017-06-06	1.0	First release

Disclaimer

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10, Xibeiwang East Road, Haidian Dist., Beijing, China, 100193

Tel: 86-10-56345000

Fax: 86-10-56345001

Http: //www.ingenic.com

目录

1. 简介

工具链中 MXU 指令以内嵌汇编的方式提供。MXU 有 16 个 32bit 通用寄存器(xr0~xr15)，一个 32 bit 控制寄存器(xr16)。使用工具链中的 MXU 宏定义时需要程序员手动选择要使用的 MXU 寄存器。如 Q16ADD_AA_WW(xr1,xr2,xr3,xr4),其中 Q16ADD_AA_WW 是内嵌汇编宏，xr1,xr2,xr3,xr4 是寄存器编号。指令的使用请详见**附录**，寄存器和内嵌汇编宏的定义请详见工具链中的 jzmedia.h (PATH_toolchain /mips-linux-gnu/libc/usr/include/jzmedia.h)。

- 使用 MXU 的内嵌汇编宏需要引用 jzmedia.h
- 使用 MXU 指令，在编译时需要添加“-mmxu” 编译选项。
- 使用 MXU 指令，在反汇编时需要添加“-Mmxu” 选项。

2. 指令使用举例

以 Q16ADD 指令为例

```
/* test.c */
#include <stdio.h>
#include <jzmedia.h>
void main()
{
        short src1 [2] = { 1 , 2 };
        short src2 [2] = { 10,20 };
        short dst1 [2] ;
        short dst2 [2] ;

        S32LDD(xr1, src1, 0);
        S32LDD(xr2, src2, 0);

      /*调用 Q16ADD 指令，其中
          EPTN2 :AA（00）,表示加加运算，
          OPTN2:WW（00），表示高位与高位，低位与低位操作
          Operation:
                xr4 = {xr1.l+xr2.l, xr1.h+xr2.h};
                xr3= {xr1.l+xr2.l, xr1.h+xr2.h};*/
    Q16ADD_AA_WW(xr4,xr2,xr1,xr3);

      S32STD(xr3, dst1, 0);
      S32STD(xr4, dst2, 0);
      printf(" dst1[0] is %d, dst1[1] is %d\n", dst1[0], dst2[1]);
      printf(" dst2[0] is %d, dst2[1] is %d\n", dst2[0], dst2[1]);
}
```

编译：

$ mips-linux-gnu-gcc test.c -o test -mmxu

反汇编：

$ mips-linux-gnu-objdump -D test -Mmxu

运行：

$./test

dst1[0] is 11, dst1[1] is 22

dst2[0] is 11, dst2[1] is 22

3. 附录

MXU 指令具体含义及使用方法请参考《XBurst_ISA_MXU_PM》,MXU 指令使用列表如下：

注：寄存器名称（xra,xrb,xrc,xrd...）为示例，不代表真实寄存器。

D16MUL_WW(xra,xrb,xrc,xrd);
D16MUL_HW(xra,xrb,xrc,xrd);
D16MUL_LW(xra,xrb,xrc,xrd);
D16MUL_XW(xra,xrb,xrc,xrd);

D16MULF_WW(xra,xrb,xrc);
D16MULF_LW(xra,xrb,xrc);
D16MULF_HW(xra,xrb,xrc);
D16MULF_XW(xra,xrb,xrc);

D16MULE_WW(xra,xrb,xrc,xrd);
D16MULE_LW(xra,xrb,xrc,xrd);
D16MULE_HW(xra,xrb,xrc,xrd);
D16MULE_XW(xra,xrb,xrc,xrd);

D16MAC_AA_WW(xra,xrb,xrc,xrd);
D16MAC_AA_LW(xra,xrb,xrc,xrd);
D16MAC_AA_HW(xra,xrb,xrc,xrd);
D16MAC_AA_XW(xra,xrb,xrc,xrd);

D16MAC_AS_WW(xra,xrb,xrc,xrd);
D16MAC_AS_LW(xra,xrb,xrc,xrd);
D16MAC_AS_HW(xra,xrb,xrc,xrd);
D16MAC_AS_XW(xra,xrb,xrc,xrd);

D16MAC_SA_WW(xra,xrb,xrc,xrd);
D16MAC_SA_LW(xra,xrb,xrc,xrd);
D16MAC_SA_HW(xra,xrb,xrc,xrd);

```
D16MAC_SA_XW(xra,xrb,xrc,xrd);

D16MAC_SS_WW(xra,xrb,xrc,xrd);
D16MAC_SS_LW(xra,xrb,xrc,xrd);
D16MAC_SS_HW(xra,xrb,xrc,xrd);
D16MAC_SS_XW(xra,xrb,xrc,xrd);

D16MACF_AA_WW(xra,xrb,xrc,xrd);
D16MACF_AA_LW(xra,xrb,xrc,xrd);
D16MACF_AA_HW(xra,xrb,xrc,xrd);
D16MACF_AA_XW(xra,xrb,xrc,xrd);

D16MACF_AS_WW(xra,xrb,xrc,xrd);
D16MACF_AS_LW(xra,xrb,xrc,xrd);
D16MACF_AS_HW(xra,xrb,xrc,xrd);
D16MACF_AS_XW(xra,xrb,xrc,xrd);

D16MACF_SA_WW(xra,xrb,xrc,xrd);
D16MACF_SA_LW(xra,xrb,xrc,xrd);
D16MACF_SA_HW(xra,xrb,xrc,xrd);
D16MACF_SA_XW(xra,xrb,xrc,xrd);

D16MACF_SS_WW(xra,xrb,xrc,xrd);
D16MACF_SS_LW(xra,xrb,xrc,xrd);
D16MACF_SS_HW(xra,xrb,xrc,xrd);
D16MACF_SS_XW(xra,xrb,xrc,xrd);

D16MADL_AA_WW(xra,xrb,xrc,xrd);
D16MADL_AA_LW(xra,xrb,xrc,xrd);
D16MADL_AA_HW(xra,xrb,xrc,xrd);
D16MADL_AA_XW(xra,xrb,xrc,xrd);

D16MADL_AS_WW(xra,xrb,xrc,xrd);
D16MADL_AS_LW(xra,xrb,xrc,xrd);
D16MADL_AS_HW(xra,xrb,xrc,xrd);
D16MADL_AS_XW(xra,xrb,xrc,xrd);

D16MADL_SA_WW(xra,xrb,xrc,xrd);
D16MADL_SA_LW(xra,xrb,xrc,xrd);
D16MADL_SA_HW(xra,xrb,xrc,xrd);
D16MADL_SA_XW(xra,xrb,xrc,xrd);

D16MADL_SS_WW(xra,xrb,xrc,xrd);
D16MADL_SS_LW(xra,xrb,xrc,xrd);
```

```
D16MADL_SS_HW(xra,xrb,xrc,xrd);
D16MADL_SS_XW(xra,xrb,xrc,xrd);

S16MAD_A_HH(xra,xrb,xrc,xrd);
S16MAD_A_LL(xra,xrb,xrc,xrd);
S16MAD_A_HL(xra,xrb,xrc,xrd);
S16MAD_A_LH(xra,xrb,xrc,xrd);

S16MAD_S_HH(xra,xrb,xrc,xrd);
S16MAD_S_LL(xra,xrb,xrc,xrd);
S16MAD_S_HL(xra,xrb,xrc,xrd);
S16MAD_S_LH(xra,xrb,xrc,xrd);

Q16ADD_AA_WW(xra,xrb,xrc,xrd);
Q16ADD_AA_LW(xra,xrb,xrc,xrd);
Q16ADD_AA_HW(xra,xrb,xrc,xrd);
Q16ADD_AA_XW(xra,xrb,xrc,xrd);

Q16ADD_AS_WW(xra,xrb,xrc,xrd);
Q16ADD_AS_LW(xra,xrb,xrc,xrd);
Q16ADD_AS_HW(xra,xrb,xrc,xrd);
Q16ADD_AS_XW(xra,xrb,xrc,xrd);

Q16ADD_SA_WW(xra,xrb,xrc,xrd);
Q16ADD_SA_LW(xra,xrb,xrc,xrd);
Q16ADD_SA_HW(xra,xrb,xrc,xrd);
Q16ADD_SA_XW(xra,xrb,xrc,xrd);

Q16ADD_SS_WW(xra,xrb,xrc,xrd);
Q16ADD_SS_LW(xra,xrb,xrc,xrd);
Q16ADD_SS_HW(xra,xrb,xrc,xrd);
Q16ADD_SS_XW(xra,xrb,xrc,xrd);

D16MACE_AA_WW(xra,xrb,xrc,xrd);
D16MACE_AA_LW(xra,xrb,xrc,xrd);
D16MACE_AA_HW(xra,xrb,xrc,xrd);
D16MACE_AA_XW(xra,xrb,xrc,xrd);

D16MACE_AS_WW(xra,xrb,xrc,xrd);
D16MACE_AS_LW(xra,xrb,xrc,xrd);
D16MACE_AS_HW(xra,xrb,xrc,xrd);
D16MACE_AS_XW(xra,xrb,xrc,xrd);

D16MACE_SA_WW(xra,xrb,xrc,xrd);
```

D16MACE_SA_LW(xra,xrb,xrc,xrd);
D16MACE_SA_HW(xra,xrb,xrc,xrd);
D16MACE_SA_XW(xra,xrb,xrc,xrd);

D16MACE_SS_WW(xra,xrb,xrc,xrd);
D16MACE_SS_LW(xra,xrb,xrc,xrd);
D16MACE_SS_HW(xra,xrb,xrc,xrd);
D16MACE_SS_XW(xra,xrb,xrc,xrd);

Q8MUL(xra,xrb,xrc,xrd);
Q8MULSU(xra,xrb,xrc,xrd);
Q8MOVZ(xra,xrb,xrc);
Q8MOVN(xra,xrb,xrc);
D16MOVZ(xra,xrb,xrc);
D16MOVN(xra,xrb,xrc);
S32MOVZ(xra,xrb,xrc);
S32MOVN(xra,xrb,xrc);

Q8MAC_AA(xra,xrb,xrc,xrd);
Q8MAC_AS(xra,xrb,xrc,xrd);
Q8MAC_SA(xra,xrb,xrc,xrd);
Q8MAC_SS(xra,xrb,xrc,xrd);

Q8MACSU_AA(xra,xrb,xrc,xrd);
Q8MACSU_AS(xra,xrb,xrc,xrd);
Q8MACSU_SA(xra,xrb,xrc,xrd);
Q8MACSU_SS(xra,xrb,xrc,xrd);

Q8MADL_AA(xra,xrb,xrc,xrd);
Q8MADL_AS(xra,xrb,xrc,xrd);
Q8MADL_SA(xra,xrb,xrc,xrd);
Q8MADL_SS(xra,xrb,xrc,xrd);

S32SFL(xra, xrb, xrc, xrd, optn2) //optn2: ptn0(00), ptn1(01), ptn2(10), ptn3(11).

Q16SCOP(xra,xrb,xrc,xrd);
Q8SAD(xra,xrb,xrc,xrd);

D32ADD_AA(xra,xrb,xrc,xrd);
D32ADD_AS(xra,xrb,xrc,xrd);
D32ADD_SA(xra,xrb,xrc,xrd);
D32ADD_SS(xra,xrb,xrc,xrd);

D32ACC_AA(xra,xrb,xrc,xrd);

```
D32ACC_AS(xra,xrb,xrc,xrd);
D32ACC_SA(xra,xrb,xrc,xrd);
D32ACC_SS(xra,xrb,xrc,xrd);

D32ACCM_AA(xra,xrb,xrc,xrd);
D32ACCM_AS(xra,xrb,xrc,xrd);
D32ACCM_SA(xra,xrb,xrc,xrd);
D32ACCM_SS(xra,xrb,xrc,xrd);

D32ASUM_AA(xra,xrb,xrc,xrd);
D32ASUM_AS(xra,xrb,xrc,xrd);
D32ASUM_SA(xra,xrb,xrc,xrd);
D32ASUM_SS(xra,xrb,xrc,xrd);

Q16ACC_AA(xra,xrb,xrc,xrd);
Q16ACC_AS(xra,xrb,xrc,xrd);
Q16ACC_SA(xra,xrb,xrc,xrd);
Q16ACC_SS(xra,xrb,xrc,xrd);

Q16ACCM_AA(xra,xrb,xrc,xrd);
Q16ACCM_AS(xra,xrb,xrc,xrd);
Q16ACCM_SA(xra,xrb,xrc,xrd);
Q16ACCM_SS(xra,xrb,xrc,xrd);

D16ASUM_AA(xra,xrb,xrc,xrd);
D16ASUM_AS(xra,xrb,xrc,xrd);
D16ASUM_SA(xra,xrb,xrc,xrd);
D16ASUM_SS(xra,xrb,xrc,xrd);

Q8ADDE_AA(xra,xrb,xrc,xrd);
Q8ADDE_AS(xra,xrb,xrc,xrd);
Q8ADDE_SA(xra,xrb,xrc,xrd);
Q8ADDE_SS(xra,xrb,xrc,xrd);

Q8ACCE_AA(xra,xrb,xrc,xrd);
Q8ACCE_AS(xra,xrb,xrc,xrd);
Q8ACCE_SA(xra,xrb,xrc,xrd);
Q8ACCE_SS(xra,xrb,xrc,xrd);

D32ADDC(xra,xrb,xrc,xrd);
D8SUM(xra,xrb,xrc);
D8SUMC(xra,xrb,xrc);

S32CPS(xra,xrb,xrc);
```

```
D16CPS(xra,xrb,xrc);
Q8ABD(xra,xrb,xrc);
Q16SAT(xra,xrb,xrc);

S32SLT(xra,xrb,xrc);
D16SLT(xra,xrb,xrc);
D16AVG(xra,xrb,xrc);
D16AVGR(xra,xrb,xrc);
Q8AVG(xra,xrb,xrc);
Q8AVGR(xra,xrb,xrc);
Q8ADD_AA(xra,xrb,xrc);
Q8ADD_AS(xra,xrb,xrc);
Q8ADD_SA(xra,xrb,xrc);
Q8ADD_SS(xra,xrb,xrc);

S32MAX(xra,xrb,xrc);
S32MIN(xra,xrb,xrc);
D16MAX(xra,xrb,xrc);
D16MIN(xra,xrb,xrc);
Q8MAX(xra,xrb,xrc);
Q8MIN(xra,xrb,xrc);
Q8SLT(xra,xrb,xrc);
Q8SLTU(xra,xrb,xrc);

D32SLL (xra,xrb,xrc,xrd,sft4);      //sft4: range 0 ~ 15.
D32SLR (xra,xrb,xrc,xrd,sft4);      //sft4: range 0 ~ 15.
D32SAR (xra,xrb,xrc,xrd,sft4);      //sft4: range 0 ~ 15.
D32SARL (xra,xrb,xrc,sft4);         //sft4: range 0 ~ 15.

Q16SLL(xra, xrb, xrc, xrd, sft4);   //sft4: range 0 ~ 15.
Q16SLR(xra, xrb, xrc, xrd, sft4);   //sft4: range 0 ~ 15.
Q16SAR(xra, xrb, xrc, xrd, sft4);   //sft4: range 0 ~ 15.


D32SLLV(xra,xrb,rb);
D32SLRV(xra,xrb,rb);
D32SARV(xra,xrb,rb);
Q16SLLV(xra,xrb,rb);
Q16SLRV(xra,xrb,rb);
Q16SARV(xra,xrb,rb);

S32MADD(xra,xrb,rs,rt);
S32MADDU(xra,xrb,rs,rt);
```

```
S32MSUB(xra,xrb,rs,rt);
S32MSUBU(xra,xrb,rs,rt);

S32MUL(xra,xrb,rs,rt);
S32MULU(xra,xrb,rs,rt);

S32EXTR(xra,xrd,rs,bits5)    //bits5: range 1~31

S32EXTRV(xra,xrd,rs,rt);

D32SARW(xra,xrb,xrc,rb);

S32ALN(xra,xrb,xrc,rb);
S32ALNI(xra,xrb,xrc,optn3); //optn3 : ptn0(000), ptn1(001), ptn2(010), ptn3(011), ptn4(100),

S32LUI(xra,s8,optn3);          //s8: range -128 ~ 127;
                               //optn3: ptn0(000), ptn1(001), ptn2(010), ptn3(011),
                                        ptn4(100), ptn5(101), ptn6(110), ptn7(111).
S32NOR(xra,xrb,xrc);
S32AND(xra,xrb,xrc);
S32OR(xra,xrb,xrc);
S32XOR(xra,xrb,xrc);

LXB (rb,rc,strd2);             //STRD2: 0~2
LXBU (rb,rc,strd2);            //STRD2: 0~2
LXH (rb,rc,strd2);             //STRD2: 0~2
LXHU (rb,rc,strd2);            //STRD2: 0~2
LXW (rb,rc,strd2);             //STRD2: 0~2

S16LDD(xra,rb,s10,optn2);       //s10: range  - 512 ~ 510;
                                //optn2: ptn0(000), ptn1(001), ptn2(010), ptn3(011).
S16STD(xra,rb,s10,optn2);    //s10: range  - 512 ~ 510;
                               //optn2: ptn0(000), ptn1(001), ptn2(010), ptn3(011).

S32STDR(xra,rb,s12) ;        //s12: range  - 2048 ~ 2044.
S32LDI(xra,rb,s12) ;         //s12: range  - 2048 ~ 2044.
S32LDIR(xra,rb,s12) ;        //s12: range  - 2048 ~ 2044.
S32SDI(xra,rb,s12) ;         //s12: range  - 2048 ~ 2044.
S32SDIR(xra,rb,s12) ;        //s12: range  - 2048 ~ 2044.

S8LDD(xra,rb,s8,optn3);    //s8: range  - 128 ~ 127;
                           //optn3: ptn0(000), ptn1(001), ptn2(010), ptn3(011),
                             ptn4(100), ptn5(101), ptn6(110), ptn7(111).
S8STD(xra,rb,s8,optn3);    //s8: range  - 128 ~ 127;
```

```
                                // optn3: ptn0(000), ptn1(001), ptn2(010), ptn3(011),
                                          ptn4(100), ptn5(101), ptn6(110), ptn7(111).
S8LDI(xra,rb,s8,optn3);    //s8: range －128 ~ 127;
                                optn3: ptn0(000), ptn1(001), ptn2(010), ptn3(011),
                                          ptn4(100), ptn5(101), ptn6(110), ptn7(111).
S8SDI(xra,rb,s8,optn3);    //s8: range －128 ~ 127;
                               //optn3: ptn0(000), ptn1(001), ptn2(010), ptn3(011),
                                          ptn4(100), ptn5(101), ptn6(110), ptn7(111).

S32SDIR(xra,rb,s12);       //s12: range －2048 ~ 2044.
S32LDDR(xra,rb,s12);       //s12: range －2048 ~ 2044.
```