Application Note 108
Cyrix Extension to the Multimedia Instruction Set

Cyrix® A National Semiconductor Company

Applies to the Cyrix MII™ CPU

APPLICATION NOTE 108

Cyrix Extensions to the Multimedia Instruction Set

Introduction

The Cyrix Extended Multimedia Instruction (EMMI) set add new Multimedia instructions to the existing Cyrix Multimedia Instruction (MMI) set. This application notes applies to the MII™ CPU. The Extended Multimedia Instruction (EMMI) facilitates the writing of multimedia applications. In general, these instructions allow more efficient implementation of multimedia algorithms, or more precision in computation than can be achieved using the basic set of MMI instructions. All of the added instructions follow the SIMD (single instruction, multiple data) format. Many of the instructions add flexibility to the MMI architecture by allowing both source operands of an instruction to be preserved, while the result goes to a separate register that is derived from the input.

Note: The Multimedia Instruction (MMI) set is compatible with Intel's MMX™ Instruction set.

Enabling Cyrix Extended Multimedia Instruction Instructions

The Cyrix Extended Multimedia Instruction Instructions are enabled by setting EMMI (CCR7 bit 0). This bit is set to 0 by default. The CCR7 register is accessed through index “EBh”. To access CCR7 requires privilege level 0 access.

Implied Registers

In the IDCT (Inverse Discrete Cosine Transform) algorithm, which is necessary for an MPEG video application, there are several places where two vector inputs are used in two separate calculations. In one calculation, the two vectors may be added, and in the second one of the vectors is subtracted from the other. In order to accomplish this algorithm using the basic MMX™ instructions from Intel (or the basic MMI instructions from Cyrix), one of the vectors must be copied in order to preserve its original value before the first computation. This is because the basic MMX instructions all destroy the contents of one of the source registers by using the same register as the destination. Several of the Cyrix-added instructions get around this problem by having an *implied destination* register, which is derived from the first source register. This way, the contents of both source vectors is preserved without having to make a copy of either one. A few of the instructions use an implied register as another source, so that the first register in the instruction is still the destination.

The implied register is calculated from the first source, according to the following table:

IMPLIED REGISTER PAIRS

First Source Register	Implied Register
mm0	mm1
mm1	mm0
mm2	mm3
mm3	mm2
mm4	mm5
mm5	mm4
mm6	mm7
mm7	mm6

From the table, it should be apparent that the source and destination registers are in pairs, where the pairs are determined by changing the least significant bit of the binary representation of the register number.

Example

The PADDSIW instruction performs the same function as Intel's MMI PADDSW instruction, except that it preserves the contents of both input vectors. If one of the vectors of interest is in register mm1 and the other is in register mm2, the instruction would look like this:

PADDSIW mm1, mm2

and the result would end up in register mm0. The instruction could also be written as

PADDSIW mm2, mm1

and the result would end up in register mm3. In this particular instruction, the second input can also be a memory operand, but the implied register stays the same, so

PADDSIW mm1, [si]

puts its result in register mm0.

Caution is required for programming with these instructions in order for them to have the desired effect. For example,

PADDSIW mm1, mm0

will put its result in register mm0, thus losing the original input value. The instruction written this way is exactly equivalent to

PADDSW mm0, mm1.

A few of the instructions that use an implied register still use the first register in the instruction as the destination. These instructions are the packed conditional move commands PMVZB, PMVZNB, PMVLZB, and PMVGEZB. Note that the mnemonics for these instructions do not have the "I" for "implied destination" in them, so there should be no ambiguity about where the result goes. In the case of the packed conditional move instructions, the packed values from the source are moved as packed values to the destination register, depending upon the packed values in the implied register. They are three-input instructions.

PADDSIW -- Packed Add with Saturation, using Implied Destination

Opcode	Instruction	Description
0F 51 /r	PADDSIW mm,mm/m64	Add signed packed byte in MMI register/memory to signed packed byte in MMI register, saturate, and write result to implied register.

Operation

mmi(15..0) <- SaturateToSignedWord(mm(15..0) + m64(15..0));

mmi(31..16) <- SaturateToSignedWord(mm(31..16) + m64(31..16));

mmi(47..32) <- SaturateToSignedWord(mm(47..32) + m64(47..32));

mmi(63..48) <- SaturateToSignedWord(mm(63..48) + m64(63..48));

Description

The PADDSIW instruction adds the signed words of the source operand to the signed words of the destination operand and writes the results to the implied MMI register. The purpose of this instruction is the same as the PADDSW instruction, except that it preserves both source operands.

The first source must be an MMI register. The second source can be either an MMI register or a 64-bit memory operand. The destination is an MMI register which depends on the first source.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

PAVEB -- Packed Average

Opcode	Instruction	Description
0F 50 /r	PAVEB mm,mm/m64	Average packed byte from MMI register/memory with packed byte in MMI register.

Operation

mm(7..0) <- (mm(7..0) + mm/m64(7..0)) >> 1;

mm(15..8) <- (mm(15..8) + mm/m64(15..8)) >> 1;

mm(23..16) <- (mm(23..16) + mm/m64(23..16)) >> 1;

mm(31..24) <- (mm(31..24) + mm/m64(31..24)) >> 1;

mm(39..32) <- (mm(39..32) + mm/m64(39..32)) >> 1;

mm(47..40) <- (mm(47..40) + mm/m64(47..40)) >> 1;

mm(55..48) <- (mm(55..48) + mm/m64(55..48)) >> 1;

mm(63..56) <- (mm(63..56) + mm/m64(63..56)) >> 1;

Description

The PAVEB instruction calculates the average of the unsigned bytes of the source operand and the unsigned bytes of the destination operand and writes the result to the MMI register. The PAVEB instruction cannot overflow.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

Caveat

M2 hardware versions before v1.3 interpret values as signed bytes on this instruction.

PDISTIB -- Packed Distance and Accumulate with Implied Register

Opcode	Instruction	Description
0F 54 /r	PDISTIB mm,m64	Find absolute value of difference between packed byte in memory and packed byte in MMI register and accumulate with value in implied destination register, using unsigned saturation.

Operation

mmi(7..0) <- SaturateToUnsignedByte(mmi(7..0) + abs(mm(7..0) - m64(7..0)));

mmi(15..8) <- SaturateToUnsignedByte(mmi(15..8) + abs(mm(15..8) - m64(15..8)));

mmi(23..16) <- SaturateToUnsignedByte(mmi(23..16) + abs(mm(23..16) - m64(23..16)));

mmi(31..24) <- SaturateToUnsignedByte(mmi(31..24) + abs(mm(31..24) - m64(31..24)));

mmi(39..32) <- SaturateToUnsignedByte(mmi(39..32) + abs(mm(39..32) - m64(39..32)));

mmi(47..40) <- SaturateToUnsignedByte(mmi(47..40) + abs(mm(47..40) - m64(47..40)));

mmi(55..48) <- SaturateToUnsignedByte(mmi(55..48) + abs(mm(55..48) - m64(55..48)));

mmi(63..56) <- SaturateToUnsignedByte(mmi(63..56) + abs(mm(63..56) - m64(63..56)));

Description

The PDISTIB instruction calculates the distance between the unsigned bytes of the two source operands, adds the result to the unsigned byte in the implied destination operand, and saturates the result. The result is written to the implied MMI register, which is calculated using the method discussed earlier.

The first source must be an MMI register. The second source must be a 64-bit memory operand. The accumulator and destination is an MMI register which depends on the first source.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

PMACHRIW -- Packed Multiply and Accumulate with Rounding

Opcode	Instruction	Description
0F 5E /r	PMACHRIW mm,m64	Perform the PMULHRW function on the two source operands, and accumulate the result with the packed signed word in the implied destination register.

Operation

mmi(15..0) <- mmi(15..0) + (mm(15..0) * mm/m64(15..0) + 0x00004000)(30..15) ;

mmi(31..16) <- mmi(31..16) + (mm(31..16) * mm/m64(31..16) + 0x00004000)(30..15) ;

mmi(47..32) <- mmi(47..32) + (mm(47..32) * mm/m64(47..32) + 0x00004000)(30..15) ;

mmi(63..48) <- mmi(63..48) + (mm(63..48) * mm/m64(63..48) + 0x00004000)(30..15) ;

Description

The PMACHRIW multiplies the two source operands using the method described for PMULHRW, and then accumulates the result with the value in the implied destination register using wrap-around arithmetic. The final result is placed in the implied destination register.

The first source is an MMI register. The second source must be a 64-bit memory operand. The destination operand is an implied MMI register that depends on the first source, as discussed earlier.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

PMAGW -- Packed Magnitude

Opcode	Instruction	Description
0F 52 /r	PMAGW mm,mm/m64	Set the destination equal to the packed word with the largest magnitude, between the packed word in MMI register/memory and MMI register.

Operation

IF abs(mm/m64(15..0)) > abs(mm(15..0)) THEN mm(15..0) <- mm/m64(15..0);

IF abs(mm/m64(31..16)) > abs(mm(31..16)) THEN mm(31..16) <- mm/m64(31..16);

IF abs(mm/m64(47..32)) > abs(mm(47..32)) THEN mm(47..32) <- mm/m64(47..32);

IF abs(mm/m64(63..56)) > abs(mm(63..56)) THEN mm(63..56) <- mm/m64(63..56);

Description

The PMAGW instruction compares the absolute value of the packed words in the source operand to the absolute value of the packed words in the destination operand and sets the destination words to the value that has the larger magnitude. The PMAGW instruction does not change the sign of the value with the larger magnitude and it does not saturate.

The source can be either an MMI register or a 64-bit memory operand. The destination operand is an MMI register.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

PMULHRW/PMULHRIW -- Packed Multiply High with Rounding

Opcode	Instruction	Description
0F 59 /r	PMULHRW mm,mm/m64	Multiply the signed packed word in the MMI register/memory with the signed packed word in MMI register, round with 1/2 bit 15, and store bits 30..15 of the result in MMI register.
0F 5D /r	PMULHRIW mm,mm/m64	Multiply the signed packed word in the MMI register/memory with the signed packed word in MMI register, round with 1/2 bit 15, and store bits 30..15 of the result in implied MMI register.

Operation

IF instruction is PMULHRW

THEN {

mm(15..0) <- (mm(15..0) * mm/m64(15..0) + 0x00004000)(30..15) ;

mm(31..16) <- (mm(31..16) * mm/m64(31..16) + 0x00004000)(30..15) ;

mm(47..32) <- (mm(47..32) * mm/m64(47..32) + 0x00004000)(30..15) ;

mm(63..48) <- (mm(63..48) * mm/m64(63..48) + 0x00004000)(30..15) ;

}

ELSE { (* instruction is PMULHRIW *)

mmi(15..0) <- (mm(15..0) * mm/m64(15..0) + 0x00004000)(30..15) ;

mmi(31..16) <- (mm(31..16) * mm/m64(31..16) + 0x00004000)(30..15) ;

mmi(47..32) <- (mm(47..32) * mm/m64(47..32) + 0x00004000)(30..15) ;

mmi(63..48) <- (mm(63..48) * mm/m64(63..48) + 0x00004000)(30..15) ;

}

Description

The PMULHRW instructions are intended to give a result of the form s.15 from a 16 x 16 bit multiply with the LSB rounded before truncating to 16 bits. This is in contrast to the PMULHW instruction which gives a result of the form ss.14 with no rounding.

The source can be either an MMI register or a 64-bit memory operand. The destination operand is an MMI register -- in the case of the PMULHRIW instruction, the destination register depends on the first source register, as described earlier. The intent of the PMULHRIW instruction is the same as the PMULHRW instruction except that both sources are preserved.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

PMVZB/PMVNZB/PMVLZB/PMVGEZB -- Packed Conditional Move

Opcode	Instruction	Description
0F 58 /r	PMVZB mm,m64	Conditionally move packed byte from memory to packed byte in MMI register if packed byte in implied MMI register is zero.
0F 5A /r	PMVNZB mm,m64	Conditionally move packed byte from memory to packed byte in MMI register if packed byte in implied MMI register is not zero.
0F 5B /r	PMVLZB mm,m64	Conditionally move packed byte from memory to packed byte in MMI register if packed byte in implied MMI register is less than zero.
0F 5C /r	PMVGEZB mm,m64	Conditionally move packed byte from memory to packed byte in MMI register if packed byte in implied MMI register is greater than or equal to zero.

Operation

```
IF instruction is PMVZB
THEN {
IF mmi(7..0) == 0 THEN mm(7..0) <- m64(7..0);
IF mmi(15..8) == 0 THEN mm(15..8) <- m64(15..8);
...
IF mmi(63..56) == 0 THEN mm(63..56) <- m64(63..56);
}
ELSE IF instruction is PMVNZB
THEN {
IF mmi(7..0) != 0 THEN mm(7..0) <- m64(7..0);
IF mmi(15..8) != 0 THEN mm(15..8) <- m64(15..8);
...
```

```
IF mmi(63..56) != 0 THEN mm(63..56) <- m64(63..56);
}
ELSE IF instruction is PMVLZB
THEN {
IF mmi(7..0) < 0 THEN mm(7..0) <- m64(7..0);
IF mmi(15..8) < 0 THEN mm(15..8) <- m64(15..8);
...
IF mmi(63..56) < 0 THEN mm(63..56) <- m64(63..56);
}
ELSE (* instruction is PMVGEZB *)
{
IF mmi(7..0) >= 0 THEN mm(7..0) <- m64(7..0);
IF mmi(15..8) >= 0 THEN mm(15..8) <- m64(15..8);
...
IF mmi(63..56) >= 0 THEN mm(63..56) <- m64(63..56);
}
```

Description

The PMV instructions conditionally move packed bytes from the source operand to packed bytes of the destination operand, depending on the value of packed bytes in the implied register.

The PMVZB instruction moves bytes from the source to the destination if the corresponding byte in the implied register is equal to zero.

The PMVNZB instruction moves bytes from the source to the destination if the corresponding byte in the implied register is not equal to zero.

The PMVLZB instruction moves bytes from the source to the destination if the corresponding byte in the implied register is less than zero.

The PMVGEZB instruction moves bytes from the source to the destination if the corresponding byte in the implied register is greater than or equal to zero.

The source must be a 64-bit memory operand. The condition comes from an implied MMI register which depends on the destination. The destination operand is an MMI register.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

PSUBSIW -- Packed Subtract with Saturation, using Implied Destination

Opcode	Instruction	Description
0F 55 /r	PSUBSIW mm,mm/m64	Subtract signed packed byte in MMI register/memory from signed packed byte in MMI register, saturate, and write result to implied register.

Operation

mmi(15..0) <- SaturateToSignedWord(mm(15..0) - mm/m64(15..0));

mmi(31..16) <- SaturateToSignedWord(mm(31..16) - mm/m64(31..16));

mmi(47..32) <- SaturateToSignedWord(mm(47..32) - mm/m64(47..32));

mmi(63..48) <- SaturateToSignedWord(mm(63..48) - mm/m64(63..48));

Description

The PSUBSIW instruction subtracts the signed words of the source operand from the signed words of the destination operand and writes the results to the implied MMI register. The purpose of this instruction is the same as the PSUBSW instruction, except that it preserves both source operands.

The first source must be an MMI register. The second source can be either an MMI register or a 64-bit memory operand. The destination is an MMI register which depends on the first source.

Flags & Exceptions

No flags are affected. Exceptions are the same as other MMI arithmetic instructions.

Printed in the United States of America

Cyrix Corporation

2703 North Central Expressway

Richardson, Texas 75080-2010

United States of America

September 9, 1998 11:12 am
C:\!!!devices\appnotes\108ap.fm5

Rev 0.93 Stated that this document applies to the MII CPU
Rev 0.92 First Draft