APPLICATION NOTE
PACE1750AE
BUILT-IN FUNCTIONS

About This Note

This application brief is intended to explain and demonstrate the use of the special functions that are built into the PACE1750AE processor. These powerful Built-In Functions (BIFs) are above and beyond the capabilities of the MIL-STD-1750A instruction set, but are allowed by MIL-STD-1750A as options to enhance the performance of 1750A processors in specialized applications.

The design of the PACE1750AE is based on its successful high-performance predecessor, the PACE1750A, and is in fact pin and software compatible with that part. However, certain architectural enhancements allow the PACE1750AE to gain a 40 percent throughput increase over the PACE1750A at any given clock frequency, up to 40 MHz, as measured on the DAIS instruction mix. The most significant of these enhancements is the addition of a multiply accumulate array capable of completing a single precision integer (16-bit x 16-bit) or single precision floating point (24-bit x 24-bit) multiply accumulate operation in just two clocks.

Besides improving the performance of the integer and floating point arithmetic by as much as 700 percent over the PACE1750A, the multiply accumulate array gave the designers of the PACE1750AE an important building block from which to create the very powerful BIFs that are the subject of this note. In addition to the functions which utilize the multiply accumulate array, there are functions which improve the usefulness of the A and B timers, and a block move in the I/O address space which greatly improves the speed at which the memory map can be changed. These functions combined can offer a significant improvement in the system performance of the PACE1750AE processor family in many typical embedded applications.

DSP Functions

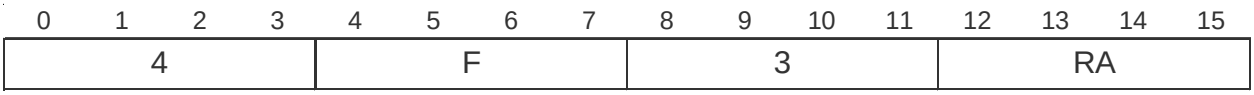
The P1750AE device incorporates a series of functions, utilizing the multiply accumulate array, intended to vastly improve performance on the operations most used in digital signal processing applications. With the addition of these functions, the P1750AE is able to perform some of the basic DSP functions with speeds rivaling those of many of the dedicated DSP processors currently available. All of these functions utilize input data in single or double precision integer format and provide results in a double precision integer format or in a special 48-bit integer format for improved accuracy.

These instructions may be used in a wide variety of applications, including FIR filters, convolution, correlation and matrix multiplication. A FIR filter implemented using the VDPS will execute in 8 clocks per tap. For the commonly used 3 x 3 matrix operations (graphics, coordinate transformation, Euler rotations, etc.) a special register based BIF is provided. It will compute a 3 x 3 column/row multiplication in only 6 clocks. Descriptions of each of these functions are provided in the following paragraphs. In all cases, for calculating the effects of wait states, WD is the number of data wait states used and WI is the number of instruction wait states used.

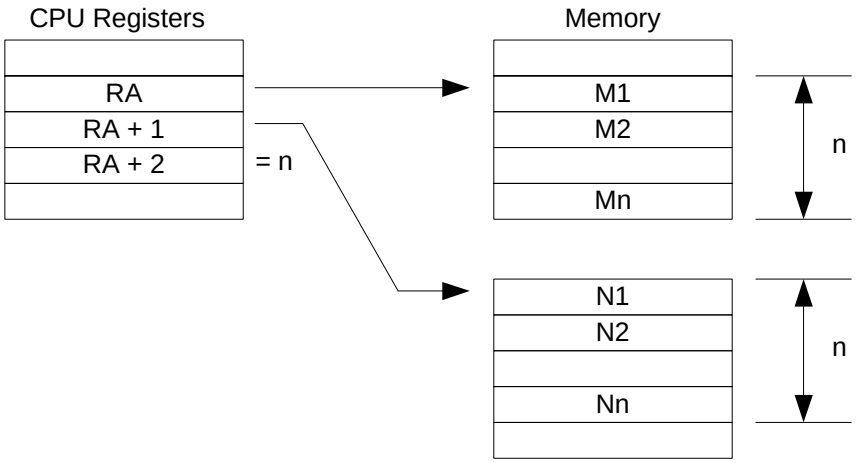
Parametric Memory to Memory Vector Dot Product
(16-bit data, 32-bit result)

Mnemonic: VDPS

Op code:



Function: $$Acc(0:31) \leftarrow Acc(0:31) + \sum_{i=1}^{n} M_i(0:15) \bullet N_i(0:15)$$



Description:

The VDPS instruction computes the vector dot product of two 16-bit memory arrays. The address of the first element of the first vector must be placed in register RA, the address of the first element of the second vector must be placed in register RA+1 and the number of elements in each vector (n) must be placed in register RA+2. When execution begins, each element of the first vector is multiplied with the corresponding element of the second vector and the 32-bit result added to the accumulator. When execution is complete, the value in RA+2 will have changed to zero and the result will be stored in the accumulator, which can be accessed via the STAC command, to be discussed later. Overflowing the accumulator will result in the Fixed-Point Overflow interrupt.

Execution Time: 10 + (8 • n) clocks (0 wait states)

Effect of Wait States: (WI-6) + (WD • 2n) additional clocks

Flags Affected: P, Z, N

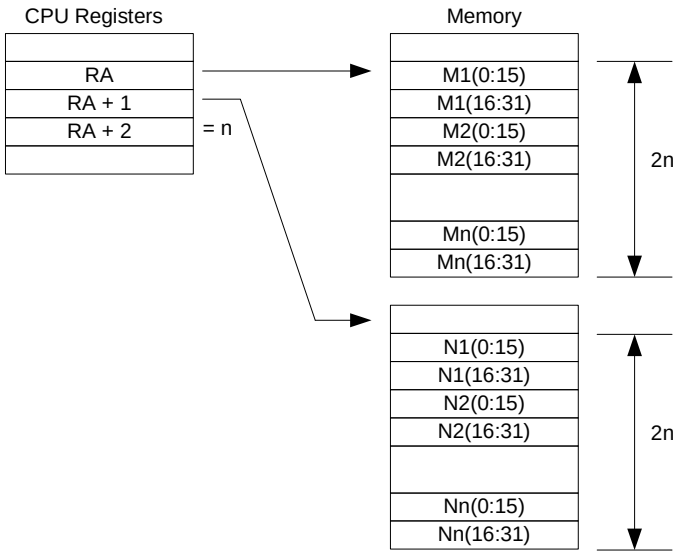
Parametric Memory to Memory Dot Product
(32-bit data, 48-bit result)

Mnemonic: VDPD

Op code:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4				F				1				RA			

Function: $$Acc(0:47) \leftarrow Acc(0:47) + \sum_{i=1}^{n} M_i(0:31) \bullet N_i(0:31)$$



Description:

The VDPD instruction computes the vector dot product of two 32 bit memory arrays. The address of the first element of the first vector must be placed in register RA, the address of the first element of the second in RA+1 and the number of elements in each vector in RA+2. The 32 bit elements must be placed in two adjacent memory locations with the most significant 16 bits residing in the memory location with the lower address. Both vectors must reside within the same address state (if an MMU is used). When execution is complete, the contents of register RA+2 will be zero and the result will be in the 48-bit accumulator. There will be no interrupt generated on an overflow.

Execution Time: 10 + 16n clocks

Effect of Wait States: (WI-6) + (WD • 4n) additional clocks

Flags Affected: None

3 X 3 Register Dot Product
(16-bit data, 32-bit result)

Mnemonic: R3DP

Op code:

Function: $Acc(0:31) \leftarrow Acc(0:31) + (R0 \bullet R3) + (R1 \bullet R4) + (R2 \bullet R5)$

CPU Registers

R0		R3
R1	•	R4
R2		R5

Description:

The R3DP instruction computes the dot product of the two 16-bit 3-element vectors formed by registers R0-R2 and R3-R5. R0-R2 contain the elements of the first vector and R3-R5 contain the elements of the second vector. The 32-bit result is placed in the accumulator. An overflow of the accumulator will cause a Fixed-point Overflow Interrupt to occur.

Execution Time: 6 clocks

Effect of Wait States: (WI-2) additional clocks

Flogs Affected: None

Double Precision Multiply/Accumulate
(32-bit data, 48-bit result)

Mnemonic: MACD

Op code:

Function: $Acc(0:47) \leftarrow Acc(0:47) + (R0, R1) \bullet (R2, R3)$

CPU Registers

Description:

The MACD instruction is the basic double precision multiply and accumulate function. It performs a 32-bit multiply of the data in two register pairs, R0-R1 and R2-R3, and adds the 48-bit result to the value in the accumulator. The most significant 16 bits of the data to be multiplied must be placed in R0 for the first operand and R2 for the second, and the least significant 16 bits must be placed in R1 and R3. An overflow of the accumulator will not cause an interrupt to the processor.

Execution Time: 8 clocks

Effect of Wait States: (WI-4) additional clocks

Flags Affected: None

Transcendental Functions

The P1750AE implements a polynomial evaluation BIF which may be used to evaluate any function which may be expressed as a series expansion. This includes functions such as sine, cosine, natural log, and many others. The use of the POLY BIF results in a significant performance improvement over the calculation of these functions with individual multiply and add instructions.

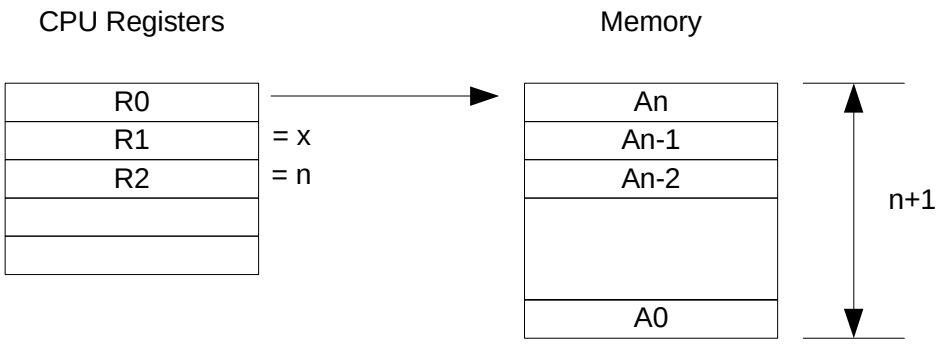
Polynomial

Mnemonic: POLY

Op code:

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
4	F	0	6

Function: $Acc(0:31) \leftarrow \sum_{i=0}^{n} A_n X^n$



Description:

The POLY instruction evaluates the polynomial of degree n defined by the registers and memory as depicted above. The values of X and the constants A_0-A_n are represented as un-normalized 16-bit two's complement fractions providing a range in value of -1<X<1. The most significant bit is a sign bit, and the remaining 15 bits represent the fixed point fraction. The result is represented as an un-normalized 24-bit two's complement fraction, providing increased accuracy. The address of A_n must be placed in register R0, the value of X must be placed in R1 and the value of n, the degree of the polynomial, must be placed in R2. The coefficients of the polynomial must be placed in memory with A_n in the first address and A_{n-1} through A_0 following in ascending addresses. An overflow of the accumulator will not cause an interrupt to the processor.

Execution Time: 6(n+1)+8 clocks

Effect of Wait States: (WD(n+1))+WI additional clocks

Flags Affected: None

Example: Using the POLY to evaluate SIN X

The POLY instruction lends itself well to the evaluation of functions which may be expanded into series representations. One such function is sin x, which may be expanded into the Taylor Series represented below with x equal to the value of an angle in radians:

$$(1) \qquad \sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

This may be reorganized into the following equation:

$$(2) \qquad \sin x = A_0 + A_1 x + A_2 x^2 + A_3 x^3 + A_4 x^4 + A_5 x^5 + A_6 x^6 + A_7 x^7 + \cdots$$

which is the format accepted by the POLY instruction. Since the series is infinite, with increasing accuracy for each additional term, the particular number of terms required for an acceptable error must be determined. It can be shown that for -1<x<1, an error of less than 0.0024 percent may be achieved by using just four of the terms of equation (1), which is acceptable in most applications. It is actually not possible to use any more than four terms for the sine calculation using the POLY instruction since the next constant required (1/9!) is too small in magnitude to represent as a 16-bit fixed point fraction. Thus, the following equation will be used to approximate the sine function:

$$(3) \qquad \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

$$= A_0 + A_1 x + A_2 x^2 + A_3 x^3 + A_4 x^4 + A_5 x^5 + A_6 x^6 + A_7 x^7$$

where:

$A_0 = 0$	$A_4 = 0$
$A_1 = 1 = 7FFFF$ (hex)	$A_5 = 1/5! = 0111$ (hex)
$A_2 = 0$	$A_6 = 0$
$A_3 = -1/3! = EAAA$ (hex)	$A_7 = -1/7! = FFFA$ (hex)

Since the above constants may only be entered as 16-bit fractions, some additional error is introduced, but the accuracy of the result is still very reasonable, as this example will show.

To calculate the value of sin x with x = 0.75 radians (42.97°) the following must be performed:

1. Place the constants in the memory. (In this example, address 00400 (hex) is chosen as the starting address)

Address	**Data**	**Address**	**Data**
00400	FFFA	00404	EAAA
00401	0000	00405	0000
00402	0111	00406	7FFF
00403	0000	00407	0000

2. Execute the following program:

```
LIM     R0, 0400        ; Load starting address of constant table
LIM     R1, 6000        ; Load the value of x (0.75)
LIM     R2, 0007        ; Load the value of n
POLY                    ; Execute the POLY instruction
```

3. The result is in the accumulator and may be retrieved as described later in this note. For this example, the result is 573EF3 (hex) which translates to 0.6816086 decimal. The actual value of sin 0.75 is 0.6816388, resulting in an error of 0.0044 percent.

In the example above, the POLY instruction executes in 56 clocks. If the same function were performed using individual multiply and add instructions with greater than 16 bits of precision, the calculation would require 4 double precision integer register multiplies (36 clocks), 1 single precision integer memory multiply with 32-bit product (13 clocks) and 3 double precision integer memory additions (48 clocks) for a total of 97 clocks. Thus, the POLY instruction calculates the sine function 42 percent faster than the discrete instructions. If floating point accuracy is required, then floating point adds and multiplies must be used.

The limitations on the value of x require that some preprocessing of the data be performed before the POLY is executed. For values of |x| greater than 1, the relationships shown below can be of help:

sin x = cos (x-π/2)
cos x = -sin (x-π/2)
sin x = sin (x-2π)
cos x = cos (x-2π)

Functions other than sine may be evaluated by the POLY as well. Any function which may be expanded into a series may be evaluated, however, for suitable accuracy the series must converge rather quickly. The series' approximating some important function are shown below:

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Accumulator Manipulation

The PACE1750AE provides a group of instructions to allow complete access to the 48-bit accumulator used in the BIFs described above. This provides the user the flexibility to use the multiply/accumulate array to supplement the performance of many custom applications. These instructions are described below:

Clear Accumulator

Mnemonic: CLAC

Op code:

Function: $Acc(0:47) \leftarrow 0$

Description: The 48-bit accumulator is loaded with 0. No registers are affected.

Execution Time: 4 clocks

Effect of Wait States: WI additional clocks

Flags affected: None

Store Accumulator into R0, R1

Mnemonic: STA

Op code:

Function: $R0, R1 \leftarrow Acc(0:31)$

Description:

The most significant 32 bits of the accumulator are placed in the R0,R1 register pair. The most significant 16 bits are placed in R0 and the least significant 16 bits are placed in R1.

Execution Time: 7 clocks

Effect of Wait States: (WI – 3) additional clocks

Flags Affected: None

Store Accumulator Long into R0,R1,R2

Mnemonic: STAL

Op Code:

Function: $R0, R1, R2 \leftarrow Acc(0:47)$

Description:

The 48-bit accumulator is placed in the three register group, R0,R1,R2. The most significant 16 bits are placed in R0, and the least significant 16 bits are placed in R2.

Execution Time: 10 clocks

Effect of Wait States: (WI – 6) additional clocks

Flags Affected: None

Load Accumulator

Mnemonic: LAC

Op code:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4				F				0				5			

Function:

Description:

The most significant 32 bits of the accumulator are loaded with the data contained in the register pair R0 and R1. The most significant 16 bits are loaded from R0 and the least significant from R1.

Execution Time: 10 clocks

Effect of Wait States: (WI – 6) additional clocks

Flags Affected: None

Load Accumulator Long

Mnemonic: LACL

Op code:

Function: $Acc(0:47) \leftarrow R0, R1, R2$

Description:

The 48-bit accumulator is loaded with the contents of the register group R0,R1,R2. The most significant 16 bits are loaded from R0 and the least significant 16 bits are loaded from R2.

Execution Time: 9 clocks

Effect of Wait States: (WI – 5) additional clocks

Flags Affected: None

Timers

The A and B timers are specified by MIL-STD-1750A as options which are implemented by the PACE1750AE. These timers are intended to provide periodic interrupts to the application software for real time operation. The can be loaded (via XIO instructions) with a value from which to count upward until reaching the value FFFF (hex), at which time they roll over to 0000 and generate an interrupt to the processor. From this point they will continue counting up toward FFFF (hex) again, unless loaded with a different value. Thus, if a periodic interrupt is desired with a period of less than 65,536 counts, the timer must be reloaded after each interrupt with a count value other than zero.

The PACE1750AE contains two additional registers, called reset registers, which may contain a value different than zero for the timers to count up from after reaching their terminal counts. This provides a very simple method of creating periodic real time interrupts with any period. The user must simply program each register once to set the interrupt period, after which the interrupts will occur regularly until a different value is written into one of the registers.

These two registers default to zero upon power-up, meaning that if they are never changed, full compatibility with the software developed for the PACE1750AE and other machines is maintained. The registers are loaded by the BIFs described below, but cannot be read.

Load Timer A Reset Register

Mnemonic: LTAR

Op code:

Function: $TAR \leftarrow R0$

Description:

The Timer A Reset Register is loaded with the value contained in R0. Timer A will count up until reaching FFFF (hex). Then, on the next count, the timer will be loaded with the value in the reset register where it will begin counting up. Timer A will continue to begin with the reset value after each terminal count until a new value is loaded into the reset register. To return to the normal MIL-STD-1750A operation mode, the reset register must be loaded with zero. No other timer functions are changed. The default value at power on and after each hardware reset is zero.

Load Timer B Reset Register

Mnemonic: LTBR

Op code:

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
4	F	0	E

Function:

Description:

The Timer B Reset Register is loaded with the value contained in R0. Timer B will count up until reaching FFFF (hex). Then, on the next count, the timer will be loaded with the value in the reset register where it will begin counting up. Timer B will continue to begin with the reset value after each terminal count until a new value is loaded into the reset register. To return to the normal MIL-STD-1750A operation mode, the reset register must be loaded with zero. No other timer functions are changed. The default value at power on and after each hardware reset is zero.

Execution Time: 4 clocks

Effect of Wait States: WI additional clocks

Flags Affected: None

Input/Output Speed Improvement

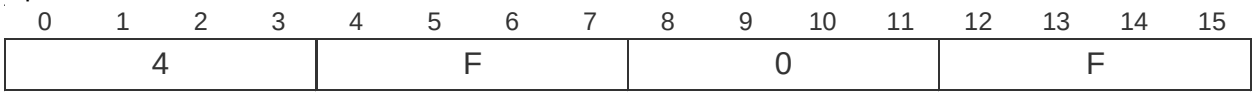
One factor which limits the performance of 1750A systems is the large overhead associated with the XIO instruction. Since this instruction is defined as privileged by MIL-STD-1750A, there are special steps which must be taken by the microcode each time an XIO or VIO instruction is encountered. This causes the length of the XIO instruction to be 24 to 34 clocks, depending on whether an output or an input is to be performed, even though the actual I/O bus cycle is only 4 clocks. This can restrict performance when large blocks of data must be transferred from memory to I/O devices, such as during loads of the MMU page registers.

The PACE1750AE provides a new instruction which allows a quick write of large blocks of data from memory to the I/O space. The privileged instruction overhead must be performed only once in this instruction. Thus, the overhead penalty is paid up front, and the incremental execution time is only 8 clocks per data transfer. In the case of the MMU page register load, if the complete set of instruction and operand page registers is loaded using XIO output instructions, a total of 512 XIO instructions must be executed at 24 clocks each. In addition to the XIOs, 512 Load Register from Memory instructions must be performed to place the data to be transferred into registers where it may be output by the XIO instructions. These instructions require 12 clocks each. Thus, the total execution time is 18,432 clocks. The same task performed by the IOMV BIF would execute in 4,118 clocks – a 78 percent improvement. The IOMV instruction is described below.

Block Move Memory to I/O

Mnemonic: IOMV

Op code:



Function: See diagram, next page.

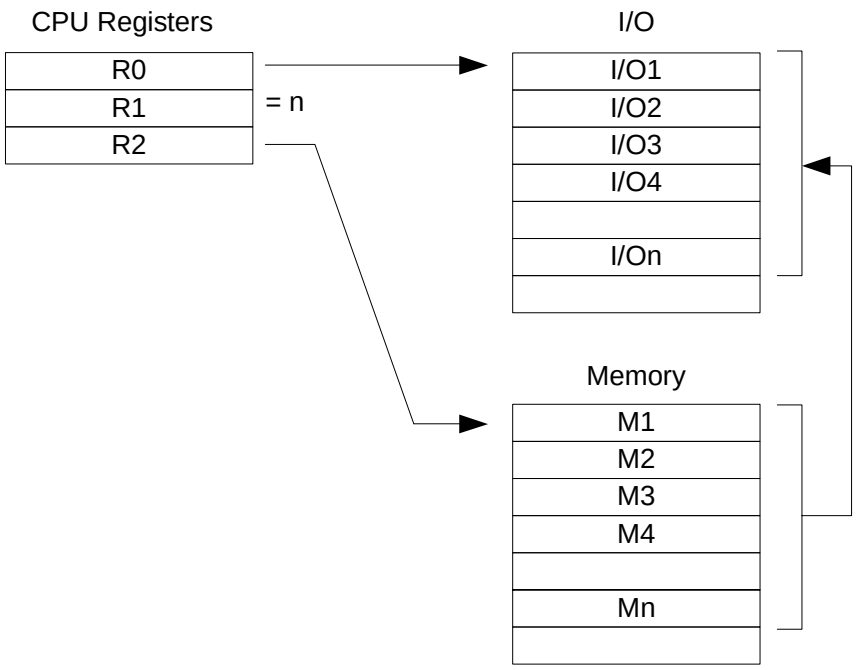
Description:

The memory block beginning at the address stored in R2 is moved to the I/O block beginning at the address stored in R0. The number of data words to be transferred is stored in R1. The maximum transfer size is 32,768, which is the number of legal I/O output addresses. All memory data must reside within the same address state if MMU is used.

Execution Time: (22 + 8n) clocks

Effect of Wait States: WI + 2n(WD) Additional clocks

Flags Affected: None



Conclusion

Though the PACE1750AE provides a significant improvement over the PACE1750A on existing software, the fullest potential of the processor may be obtained only by making use of the BIF instructions. Depending on the tasks performed by the application software, substantial improvement in the system performance of the 1750AE may be achieved through the use of the BIFs. System performance in embedded control applications was our goal when designing the PACE1750AE, and we believe that the BIFs are just one of the enhancements which allowed us, and you, to achieve that goal.

REVISIONS

DOCUMENT NUMBER: ANP16-003 **DOCUMENT TITLE**: APPLICATION NOTE - P1750AE BUILT-IN FUNCTIONS			
REV.	ISSUE DATE	ORIG. OF CHANGE	DESCRIPTION OF CHANGE
ORIG	May-89	RKK	New Data Sheet
A	Oct-05	JDB	Added Pyramid logo