GETTING STARTED WITH THE TMS320F24x PROCESSOR

Tutorial 8: Multiplication

New Instructions Introduced

```
LT
MPY   MPYU
PAC
SPL   SPH
```

Introduction

The architecture of the TMS320F24x is well suited to multiplication. It can perform a 16-bit by 16-bit multiply in a single clock cycle. Multiplications are widely used in DSP applications. In this tutorial we shall use multiplication to scale the inputs of the analog-to-digital converter (ADC).

Registers Used for Multiplication

The range of multiply instructions on the TMS320F24x do not use the accumulator. Instead they use two special registers - the T register (temporary register, 16 bits) and the P register (product register, 32 bits).

Loading the T Register

In order to carry out a multiplication of two values, one value must be placed in the T (temporary) register. For this we use the instruction `LT` (load T register). There is a restriction that the value loaded into the T register must be from a data memory (RAM) address. It is illegal to load the T register with a constant.

Example 8-1.

	`LT 75h`	`; Direct addressing. Load the T`
		`; register with the contents of`
		`; the data memory address derived`
		`; from the data page pointer (DP)`
		`; and the operand 75h.`

	`LT *`	`; Indirect addressing. Load the T`

		; register with the contents of ; the data memory address pointed ; to by the current auxiliary ; register.
	LT #33h	; Invalid addressing mode. ; Immediate addressing not ; supported.

Multiplying by an Immediate Value

All we have done so far is to load the T (temporary) register. In order to carry out the multiplication itself, we must use one of the several multiplication instructions at our disposal.

To multiply the T (temporary) register by an immediate value, we use the instruction `MPY` (multiply). Let us load the T with the value 10 decimal and then carry out three successive multiplications. Each time we will multiply by a different number.

Example 8-2.

	.setsect ".text",	8800h
start:	SETC SXM	; Turn on sign-extension mode.
	CLRC CNF	; CNF = 0. Map block B0 into ; data memory.
	LAR AR0, #290h	; AR0 points to address 290h.
	MAR *, AR0	; ARP = 0. Use AR0 for indirect ; addressing.
	SPLK #10, *	; Store the value 10 decimal at ; the data memory address 290h.
	LT *	; Load the T register with the ; contents of data memory ; address 290h.
	MPY #10	; Multiply the contents of the T ; register by the constant 10 ; decimal. The P register now ; contains 100 decimal ; (00000064h).
	MPY #0FFF6h	; Multiply the contents of the T ; register by the constant -10 ; decimal. The P register now ; contains -100 decimal ; (FFFFFF9Ch).
	MPY #0	; Multiply the contents of the T ; register by 0. The P register ; now contains 00000000h.
	B *start*	; Return to the beginning.

When using the instruction `MPY` (multiply) with an immediate value as the operand, two points need to be taken into consideration. First, the constant is *always* signed,

that is, it can be positive or negative, regardless of the state of SXM. Second, the constant has only 13 bits, which means it must lie in the range -4096 to +4095 (1000h to 0FFFh).

Example 8-3.

```
        MPY #7FFFh      ; Incorrect. Operand out of
                        ; range. The maximum allowed
                        ; value is +4095 (0FFFh). The
                        ; operand will be truncated to
                        ; 1FFFh (-1 decimal).
        MPY #8000h      ; Incorrect. Operand out of
                        ; range. The minimum allowed
                        ; value is -4096 (1000h). The
                        ; operand will be truncated to
                        ; zero.
        MPY #0FFFFh     ; Unsuccessful attempt to
                        ; multiply the T register by
                        ; 65635. The operand will be
                        ; truncated to 1FFFh (-1 decimal).
```

When using the instruction `MPY` (multiply), the value stored in the T register is taken to be a signed; that is, it lies in the range 8000h to 7FFFh (-32768 to 32767 decimal). This means that the largest positive numbers that can be multiplied are 32767 by 4095.

Multiplying Two Variables Stored in Data Memory

The instruction `MPY` (multiply) can also be used with the contents of a data memory address. In this case, the operand can take a value between 8000h and 7FFFh, representing –32768 to +32767 decimal.

Let us multiply the 16-bit variable at data memory address 220h by the 16-bit variable at data memory address 221h.

Using direct addressing we would write:

Example 8-4.

```
        .setsect ".text",  8800h

start:  CLRC CNF        ; CNF = 0. Map block B0 into
                        ; data memory.
        LDP #4          ; Page 4. Gain access to data
                        ; memory addresses 200h to 27Fh.
        SPLK #7FFFh, 20h ; Test purposes. The maximum
                        ; positive operand that can be
                        ; loaded into the T register for
                        ; use with the instruction MPY.
        SPLK #7FFFh, 21h ; Test purposes. The maximum
```

		; positive operand that can be ; used with the instruction MPY.
	LT 20h	; Load the T register by the ; contents of the data memory ; address 200h + 20h = 220h.
	MPY 21h	; Multiply the contents of data ; memory address 220h by the ; contents of data memory address ; 200h + 21h = 221h. The product ; is stored in the P register.
	B *start*	; Go round again.

The product will be 3FFF0001h. We shall now repeat the same operation using indirect addressing.

Example 8-5.

	.setsect ".text",	8800h
start:	CLRC CNF	; CNF = 0. Map blocks B0 into ; data memory.
	LAR AR2, #220h	; Use AR2 as a pointer to data ; memory address 220h.
	MAR *, AR2	; Make ARP = 2. The instructions ; following will use AR2 for ; indirect addressing.
	SPLK #7FFFh, *+	; Test purposes. The maximum ; positive value that can be ; loaded into the T register for ; use with the instruction MPY. ; Increment AR2 to point to data ; memory address 220h + 1h = 221h
	SPLK #7FFFh, *-	; Test purposes. The maximum ; positive value that can be used ; as an operand with the ; instruction MPY. Decrement AR2 ; to point to data memory address ; 221h – 1h = 220h.
	LT *+	; Load the T register with the ; contents of the data memory ; address 220h. Increment AR2 to ; point to data memory address ; 200h + 20h = 221h.
	MPY *	; Multiply contents of the T ; register by the contents of ; data memory address 221h. The ; product is stored in the P ; register.
	B *start*	; Go round again.

Again, the product contained in the P register will be 3FFF0001h. In this particular case, using direct addressing uses less code than does direct addressing.

Unsigned Multiplication

In order to multiply by a value greater than +32767 (7FFFh), the instruction `MPYU` (multiply unsigned) must be used.

The instruction `MPYU` always uses the contents of a data memory address as the operand. It cannot use immediate data.

Example 8-6.

```
        MPYU 14h        ; Direct addressing. Multiply the
                        ; unsigned contents of the T
                        ; register by the unsigned
                        ; contents of the data memory
                        ; address derived from DP and
                        ; the operand 14h.
        MPYU *          ; Indirect addressing. Multiply
                        ; the unsigned contents of the T
                        ; register by the unsigned
                        ; contents of the data memory
                        ; address pointed to by the
                        ; auxiliary register, as
                        ; specified by ARP.
        MPYU #9932h     ; Illegal operand. Immediate
                        ; addressing not supported.
```

When using the instruction `MPYU`, both the value in the T register and the data memory address are treated as *unsigned*.

Let us take an example. In order to multiply the unsigned contents of data memory address 240h by the positive value FFC0h stored at data memory address 241h, we would write:

Example 8-7.

```
        .setsect ".text", 8800h

start:  CLRC CNF        ; Map block B0 into data memory.
        LAR AR6, #240h  ; Use AR6 as a pointer to data
                        ; memory address 240h.
        MAR *, AR6      ; Make ARP = 6. The instructions
                        ; following will use AR6 for
                        ; indirect addressing.
        LT *+           ; Indirect addressing. Load the
                        ; T register with the contents of
                        ; the data memory address 240h.
                        ; Increment AR6 to point to data
                        ; memory address 241h.
        SPLK #0FFC0h, * ; Save value FFC0h at data memory
```

		; address pointed to by AR6.
	MPYU *	; Multiply the unsigned contents ; of data memory address 241h by ; the unsigned contents of the T ; register. The product is stored ; in the P register.
	B *start*	; Return to beginning.

In this case we have used the operator *+ to increment the address pointed to by AR6.

Copying from the P Register to the Accumulator

When a multiplication is performed, the resulting value is put in the P (product) register. The P register is best viewed as temporary storage because it will be overwritten by the next multiplication.

We can copy the contents of the P register to the accumulator using the instruction PAC (load accumulator with P register). No operand is required.

Example 8-8.

	PAC	; Copy the 32-bit contents of the ; P register to the accumulator. ; The contents of the P register ; remain unchanged.

Saving the Result of a Multiplication

We can also save the product of a multiplication in data memory.

To copy the 32-bit result in the P register to data memory, we use the instructions SPL (store low P register) and SPL (store high P register).

Let us assume that the P register already contains 12345678h. In order to copy the low word (5678h) to data memory address 314h, and the high word (1234h) to data memory address 315h, then using direct addressing we could write:

Example 8-9.

	LDP #6	; Page 6. Gain access to data ; memory addresses 300h to 37Fh.
	SPL 14h	; Store low word of P register ; (5678h) at data memory address ; 300h + 14h = 314h.
	SPH 15h	; Store high word of P register ; (1234h) at data memory address ; 300h + 15h = 315h.

The instruction SPL stores the low (right-most 16 bits) of the P register in data memory. In a similar way, the instruction SPH stores the high (left-most 16 bits) of the P register in data memory.

Using indirect addressing to save the contents of the P register at a data memory addresses 314h and 315h we could write:

Example 8-10.

```
        LAR AR7, #314h      ; Auxiliary register AR7 points
                            ; to address 314h.
        MAR *, AR7          ; ARP = 7. For indirect
                            ; addressing use auxiliary
                            ; register AR7.
        SPL *+              ; Store low word of P register
                            ; (5678h) at data memory address
                            ; 314h. Increment AR7 to point to
                            ; the next data memory address
                            ; (315h).
        SPH *               ; Store high word of P register
                            ; (1234h) at data memory address
                            ; 315h. No need to increment AR7.
```

Modifying the Current Auxiliary Register

Each of the instructions LT (load T register), MPY (multiply) and MPYU (multiply unsigned) allow us to change the value of ARP as part of the instruction.

For this we use a second operand, which is one of the auxiliary registers AR0 to AR7.

Some typical ways in which the current auxiliary register can be modified are shown in Example 8-11.

Example 8-11.

```
        MAR *, AR1          ; Make auxiliary register AR1 the
                            ; current auxiliary register.
        LT *, AR4           ; Load the T register with the
                            ; contents of the data memory
                            ; address pointed to by AR1.
                            ; Make ARP = 4.
                            ; For successive operations using
                            ; indirect addressing use
                            ; auxiliary register AR4.
        MPY *+, AR0         ; Multiply the contents of the T
                            ; register by the contents of
                            ; the data memory address pointed
                            ; to by AR4. Increment AR4.
                            ; Make ARP = 0.
                            ; For successive operations using
```

```
                        ; indirect addressing use
                        ; auxiliary register AR0.
        MPYU *-, AR7    ; Multiply the unsigned contents
                        ; of the T register by the
                        ; contents of the data memory
                        ; address pointed to by AR0.
                        ; Decrement AR0.
                        ; Make ARP = 7
                        ; For successive operations using
                        ; indirect addressing use
                        ; auxiliary register AR7.
```

Being able to change the current auxiliary register as part of the instruction is very useful when working on values in different data blocks.

Suppose we wish to multiply a value at data memory address 200h by a value at data memory address 270h. The 32-bit product is to be stored at data memory addresses 300h and 301h.

We shall use three different auxiliary registers as pointers to different data memory addresses: AR0 to point to 200h, AR1 to point to 270h and AR3 to point to 300h.

Example 8-12.

```
                        
        .setsect ".text",   8800h
                        
start:  CLRC CNF        ; CNF = 0. Map block B0 into
                        ; data memory.
        LAR AR0, #200h  ; AR0 points to data memory
                        ; address 200h.
        LAR AR1, #270h  ; AR1 points to data memory
                        ; address 270h.
        LAR AR2, #300h  ; AR2 points to data memory
                        ; address 300h.
        MAR *, AR0      ; ARP = 0. Make AR0 the current
                        ; auxiliary register.
        LT *, AR1       ; Load the T register with
                        ; the contents of the data memory
                        ; address pointed to by AR0
                        ; (200h).
                        ; ARP = 1. Use AR1 for indirect
                        ; addressing for following
                        ; instructions.
        MPY *, AR2      ; Multiply the T register by the
                        ; contents of the data memory
                        ; address pointed to by AR1
                        ; (270h).
                        ; ARP = 2. Use AR2 for indirect
                        ; addressing for following
                        ; instructions.
        SPL *+          ; Store the low word of the P
                        ; register at the data memory
```

		; address pointed to by AR2 ; (300h). Increment AR2 (301h).
	SPH *	; Save the high word of the P ; register at the data memory ; address pointed to by AR2 ; (301h).
	B *start*	; Return to beginning.

In this example, it should be remembered that the operand specifying a change of auxiliary register does not take effect until the following instruction. Therefore the instruction `LT *, AR1` does *not* use AR1 when loading the T register.

Scaling the Result of an Analog-to-Digital Conversion

We have seen how to use multiply instructions. We shall now use a multiplication to scale the result of an analog-to-digital conversion.

The result of an analog-to-digital conversion is stored in the following format:

Figure 8-1. Format of Result of Analog-to-Digital Conversion

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0

The value is left-justified. This means the value will lie between a minimum of 0000h and a maximum of FFC0h. This could represent a whole range of inputs e.g. 0 - 5V, 0 - 1A, 4 - 20 mA or 0 to 100 percent.

In order to convert the result of the analog-to-digital conversion to the units we wish to work in, we can multiply the result by an appropriate *scaling factor*.

Calculating the Scaling Factor

Suppose we wish to use the result of the analog-to-digital conversion to represent 0 to 100 per cent. If we were to multiply the maximum result (FFC0h) by 100 decimal the contents of the P register will be:

`64h x FFC0h = 63E700h`

If we round down the result to 630000h, we are left with the value 63h (99 decimal) and also four zeroes, which we can ignore.

Alternately, if we round this number up we have `640000h`. This is 100 decimal (64h) shifted 16 places to the right. To extract the scaled value from the 32 bit product, we simply ignore the low (right-most) 16 bits and use the high (left-most) 16 bits.

Scaling the ADC Value

The result of the analog-to-digital conversion can be multiplied by 100 using the instruction `MPYU` (multiply unsigned). This will scale the ADC input 0 to 100 percent and save it in data memory. Example 8-13 shows how we would carry out scaling using the TMS320F243 DSK.

Example 8-13.

```
                .setsect ".text",       8800h

DP_ADC          .set    224             ; Access 7000h to 707Fh.
ADCTRL1         .set    32h             ; ADC Control Register 1
ADCTRL2         .set    34h             ; ADC Control Register 2
ADCFIFO1        .set    36h             ; Result of conversion
DP_GP           .set    6               ; Data Page 6. Gain access
                                        ; to 300h to 37Fh.
factor          .set    16h             ; Storage for scaling
                                        ; factor.
scaled          .set    17h             ; Storage for scaled ADC
                                        ; value.

start:          LDP #DP_ADC             ; Page 224.
                SPLK #0C01h, ADCTRL1    ; Continuous conversion and
                                        ; turn on ADC1.
                SPLK #0007h, ADCTRL2    ; Set prescaler to maximum.

loop:           LDP #DP_ADC             ; Page 224
                LT ADCFIFO1             ; Load the T register with
                                        ; the unsigned value in
                                        ; ADCFIFO1.
                LDP #DP_GP              ; Select data page general
                                        ; purpose data.
                SPLK #100, factor       ; Temporarily save scaling
                                        ; factor (100) at data
                                        ; memory address 300h + 16h
                                        ; = 316h.
                MPYU factor             ; Unsigned multiply of
                                        ; ADCFIFO1 by scaling
                                        ; factor 100.
                SPH scaled              ; Store the high word of
                                        ; the product in the P
                                        ; register at data memory
                                        ; address 300 + 17h = 317h.
                B loop                  ; Loop round again.
```

Note that the ADC of the TMS320F243 works differently to that of the TMS320C240 and TMS320LZ2407. However, the scaling would be done in an identical way.

Here the instruction `MPY` is not used because the maximum value of an analog-to-

digital conversion is FFC0h, which the instruction `MPY` will take as a *negative* number.

What has made the scaling easy is the fact that he result of an analog-to-digital conversion is left justified. This puts the product of the multiplication into the high word (left-most part) of the P register. In order to obtain a 16-bit result, we simply ignore the low 16-bits of the P register.

Scaling the ADC Value with Rounding

When the maximum ADC input is multiplied by the scaling factor, the P register will contain:

`64h x FFC0h = 63E700h`

The value 63E700h can be viewed as the number 63 plus a fractional part, which is E700h divided by 10000h.

Rounding up is very easy. We simply add 8000h to the product, which is the equivalent of 0.5 decimal.

	63 E700h			
+	8000h			
	64 6700h			

For the TMS320F243 DSK, the code to scale the ADC input with rounding is given in Example 8-14.

Example 8-14.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Access 7000h to 707Fh.
ADCTRL1	.set 32h	; ADC Control Register 1
ADCTRL2	.set 34h	; ADC Control Register 2
ADCFIFO1	.set 36h	; Result of conversion
DP_GP	.set 6	; Data Page 6. Gain access ; to 300h to 37Fh.
factor	.set 16h	; Storage for scaling ; factor.
scaled	.set 17h	; Storage for scaled ADC ; value.
start:	LDP #DP_ADC	; Page 224.
	SPLK #0C01h, ADCTRL1	; Continuous conversion and ; turn on ADC1.
	SPLK #0007h, ADCTRL2	; Set pre-scaler to ; maximum.
loop:	LDP #DP_ADC	; Page 224

	LT ADCFIFO1	; Load the T register with ; the unsigned value in ; ADCFIFO1.
	LDP #DP_GP	; Select data page general ; purpose data.
	SPLK #100, factor	; Temporarily save scaling ; factor (100) at data ; memory address 300h + 16h ; = 316h.
	MPYU factor	; Unsigned multiply of ; ADCFIFO1 by scaling ; factor 100.
	PAC	; Copy product in P ; register to accumulator.
	ADD #8000h	; Round up by adding ; fractional equivalent of ; 0.5 decimal.
	SACH	; Save rounded value at ; data memory address ; 300h + 17h = 317h.
	B *loop*	; Loop round again.

TMS320F243 DSK Experiments

Equipment Required

TMS320F243 DSK
10k-50k Linear Potentiometer

Experiment 8-1.

Objective: To show the effect of `SXM` upon the `MPY` instruction

Enter the code in Example 8-2 into a text editor. Assemble and load into the debugger. Step through the code and make note of the values in the P register. When a positive value is multiplied by a positive value, the product should also be positive.

Change the instruction `SETC SXM` to `CLRC SXM` to turn off sign-extension mode. Step through the code. The products should be exactly the same as when sign-extension mode was turned on.

Experiment 8-2.

Objective: To Multiply by Negative Numbers

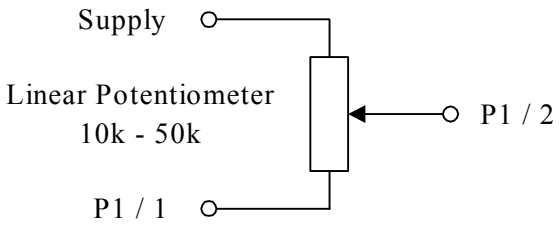
Taking the code from Example 8-2, change the line `SPLK #10, *` to `SPLK #0FFF6h, *`. Assemble and step through the code using the debugger. Make a note of the values in the P register. When a negative value is multiplied by a negative value, the product will be positive.

Change the instruction `SETC SXM` to `CLRC SXM` to turn off sign-extension mode. Step through the code. The products should be exactly the same as when sign-extension mode was turned on.

Experiment 8-3.

Objective: To scale the input from the ADC

Figure 8-2. Connection of Potentiometer



Connect the potentiometer as shown in Figure 8-2. For the TMS320F243 DSK, the supply can be up to 5V. For TMS320LZ2407 DSK the maximum allowed supply is 3.3V.

Enter the code in Example 8-13 into a text editor. Assemble it and run it on the debugger. Stop and start the program for various settings of the potentiometer. Observe that the high word (right-most part) of the P register is always in the range 0h to 63h, whatever the setting of the potentiometer.

Change the scaling factor in Example 8-13 from 64h (100 decimal) to 3E8h (1000 decimal). Assemble and run on the debugger. Observe that the high-word (right-most part) of the P register is always in the range 0 to 3E7h, whatever the setting of the potentiometer.

Experiment 8-4.

Objective: To scale the input from the ADC with rounding

Connect the potentiometer as shown in Figure 8-2.

Enter the code in Example 8-14 into a text editor. Assemble it and run load it into the debugger. Set a breakpoint at the instruction `B loop` and open the CPU window. Run the program by clicking on Animate. Observe that the high word (right-most part) of the P register is always in the range 0h to 63h, as the potentiometer is turned from one extreme to the other. The high word of the accumulator should remain in the range 0h to 64h.

Change the scaling factor in Example 8-14 from 64h (100 decimal) to 3E8h (1000 decimal). Assemble and run on the debugger. Observe that the high-word (right-most part) of the P register is always in the range 0 to 3E8h, whatever the setting of the potentiometer.

Design Problem 8-1.

Using the code in Example 8-13 as the starting point, scale the value in ADCFIFO1 in the range 0 to 100, using the instruction `MPY`. Note that the maximum value in ADCFIFO1 is FFC0h, which will be taken as a *negative* number, so some form of work-around is required.

Self-Test Questions..... *Click Here To View* Answers!

1.	What does the letter T in T register stand for? a) temporary b) tertiary c) timing d) trigger
2.	What does the letter P in P register stand for? a) permanent b) primary c) process d) product

3.	Which two of the following instructions are legal?
	`a) LT #22h`
	`b) LT AR4`
	`c) LT 58h`
	`d) LT *`

4.	How do we make the value in the P register zero?
5.	What effect does sign-extension mode have upon the instruction `MPY`?
6.	What is the most negative number we can use as intermediate data with the instruction multiply (`MPY`)?
7.	What are the most positive number we can use as immediate data with the instruction multiply (`MPY`)?
8.	When using the instruction `MPY` is value in T register taken to be signed or unsigned?
9.	Which TMS320F243 multiplication instructions use the accumulator? a) Neither `MPY` nor `MPYU` b) `MPY` c) `MPYU` d) Both `MPY` and `MPYU`.

10.	Which two of the following instructions are legal?
	`a) MPYU #8888h`
	`b) MPYU 22h`
	`c) MPYU AR6`
	`d) MPYU *`

11.	What are the differences between the instructions `MPY` and `MPYU`?
12.	How do we copy the value from the P register to the accumulator?
13.	How do we save the result of a multiplication at a data memory address?
14.	When scaling an ADC measurement, why do we use the instruction `MPYU` rather than the instruction `MPY`?

References

TMS320F/C24x DSP Controllers. CPU and Instruction Set. Reference number SPRU160

TMS320F/C240 DSP Controllers. Peripheral Library and Specific Devices. Reference Number SPRU161.

TMS320F243, TMS320F241 DSP Controllers. Reference Number SPRS064.

CLICK TO VIEW
Tutorials
1 2 3 4 5 6 7 8 9 10

Click here to view..... Route Map