C2000 *Teaching Materials*

GETTING STARTED WITH THE TMS320F24x PROCESSOR

Tutorial 7: Indirect Addressing

New Instructions Introduced

```
LAR
MAR
```

New Operators Introduced

```
*
*-
*+
```

Introduction

In earlier tutorials we saw how to transfer data using *direct addressing*. This tutorial illustrates another method of moving data known as *indirect addressing.* Indirect addressing is an important mode of operation of the TMS320F24x family, but does tend to take a little time to master.

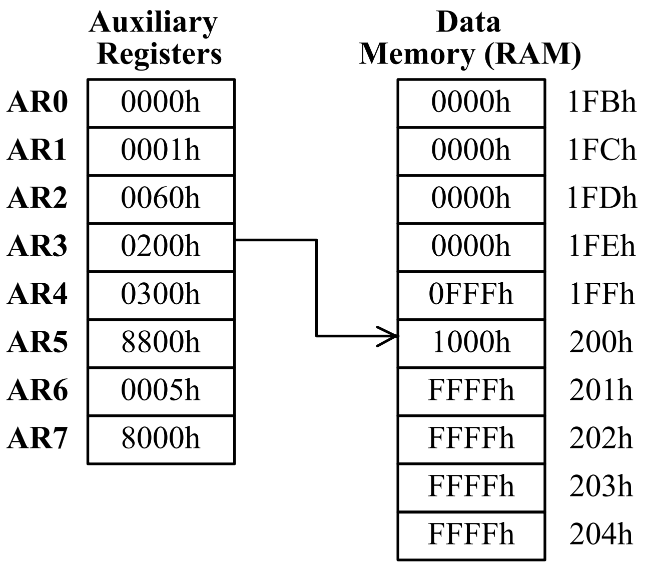
Differences between Direct Addressing and Indirect Addressing

When using *direct addressing*, the data memory address used for an operation is determined by adding the data page pointer (DP) to the operand provided by the instruction.

Indirect addressing differs in that the operand does not provide the data memory address; instead it tells us which auxiliary register contains this information. The TMS320F24x makes use of auxiliary registers AR0 to AR7 as *pointers* to data memory addresses.

Figure 7-1 shows the mechanism of indirect addressing for the TMS320F243 in simplified diagrammatic form:

Figure 7-1. A Simple Model of Indirect Addressing



In Figure 7-1, in order to work upon the contents of data memory address 200h, we have put the value 200h into auxiliary register AR3.

Note that auxiliary register AR3 does not contain the data to be worked upon, but instead contains the *address* in data memory where this data value is to be found.

A Simple Model of Indirect Addressing using the Analogy of C

We can model the behavior of indirect addressing, as shown in Figure 7-1 using the analogy of C code. Let us look at the case of where we wish to load a 32-bit variable (named `accumulator`) with the contents of memory address 200h.

Example 7-1.

`unsigned int *AR3;`	`// AR3 is a pointer to a`
	`// data memory location.`

`unsigned long accumulator;`	`// Our 32-bit variable.`

`AR3 = 0x200;`	`// Load AR3 with the address`
	`// of memory location 200h.`

`accumulator = *AR3;`	`// Load accumulator with`
	`// the contents of data`
	`// memory address`
	`// 200h.`

Please note that this is an analogy only, and we would never actually write code of this type. The important point to remember is that one of the auxiliary registers AR0 to AR7 contains the *address* of the data memory location we wish to work upon.

Loading an Auxiliary Register

Let us now implement part of the operation we saw in Example 7-1, but this time using assembly language.

In order to load auxiliary register AR3 with the address 200h, we use the instruction `LAR` (load auxiliary register):

Example 7-2.

`LAR AR3, #200h`	`; Load auxiliary register AR3 with the`
	`; immediate value 200h. AR3 now contains`
	`; 200h.`

The first operand used with the instruction `LAR` is the name of the auxiliary register and must lie in the range AR0 to AR7. The second operand is the address we wish to load. In this case the address is taken to be a constant and must be preceded by the symbol # to indicate an immediate value.

Because auxiliary registers AR0 to AR7 are 16-bit registers, the address is therefore 16-bit. This gives us access to any data memory address in the range 0 to 65535 (0 to FFFFh). We do not need to use the data page pointer (DP).

Forgetting to type in the symbol # when loading an auxiliary register with an address is likely to cause a wrong address to be loaded.

Example 7-3.

`LAR AR5, 10h`	`; Load auxiliary register AR5 with the`
	`; contents of the data memory address`
	`; derived from the data page pointer`
	`; (DP) and the operand 10h. This most`
	`; likely does not load AR5 with 10h.`

Loading the Accumulator

To load the accumulator with the contents of the data memory address pointed to by an auxiliary register, we use the instruction `LACC` (load accumulator). As an operand, we use the symbol `*` to indicate indirect addressing.

Example 7-4.

`LACC *`	`; Load the accumulator with the contents`
	`; of a data memory address whose value`
	`; is to be found in a particular`
	`; auxiliary register.`

Note that we do not use the actual name of the auxiliary register as the operand.

Example 7-5.

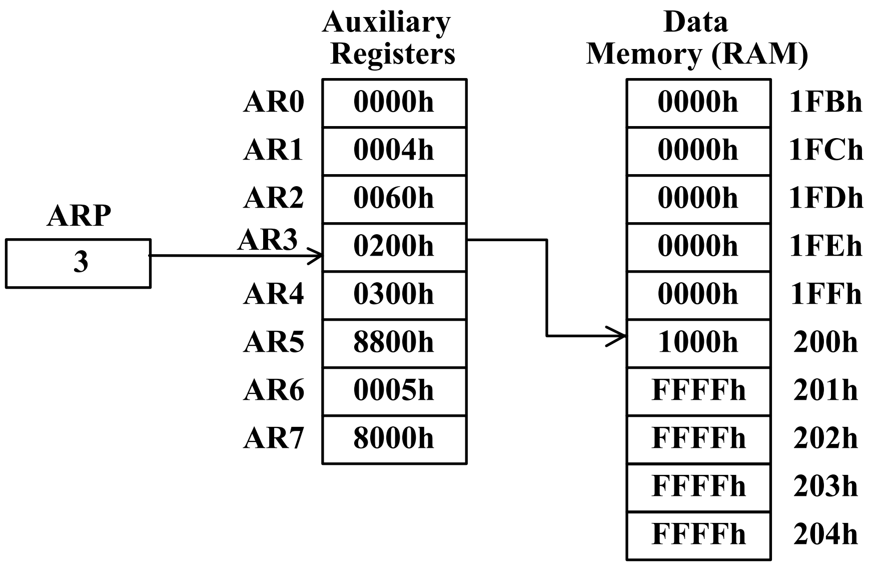
`LACC *AR4`	`; Incorrect syntax. Should be LACC *`
`LACC AR4`	`; Incorrect syntax. Should be LACC *`

The symbol `*` is used with the instruction `LACC` (load accumulator) to indicate that a *pointer* is being used to a data memory address. In this particular case, but *not* always, the symbol `*` works in the same way as the operator `*` does in C for pointers.

A Model of Indirect Addressing for the TMS320F24x

The simple model of indirect addressing we saw in Figure 7-1 is not quite how the indirect addressing works on the TMS320F24x. There is a further element, which must be taken into consideration known as the *auxiliary register pointer* (ARP). The full model of operation indirect addressing of the TMS320F24x family is shown diagrammatically in Figure 7-2:

Figure 7-2. TMS320F24x Model of Indirect Addressing



The auxiliary register pointer (ARP) holds a number between 0 and 7, corresponding to auxiliary registers AR0 to AR7 respectively. The auxiliary register selected by ARP is referred to as the *current auxiliary register.*

In order to select one of the auxiliary registers to be the one used for indirect addressing, we must first set the appropriate value of ARP. For example, to use AR3 for indirect addressing, we must make ARP = 3.

When ARP is set to a value, all following instructions using indirect addressing will use the corresponding auxiliary register until the value in ARP is changed again..

Modeling the Auxiliary Register Pointer in C

We can model the behavior of the auxiliary register pointer (ARP) using the analogy of C code, and is shown in Example 7-6:

Example 7-6

`unsigned int *AR[8];`	`// An array of 8 pointers.`
`unsigned int ARP;`	`// Index in range 0 to 7.`
`unsigned long accumulator;`	`// Accumulator. 32 bits.`
`ARP = 3;`	`// Set index to 3.`

`AR[ARP] = 0x200;`	`// Use AR[3] to point to`
	`// address 200h.`

`accumulator = *AR[ARP];`	`// Load accumulator with`
	`// the contents of the`
	`// memory address pointed to`
	`// by AR[3] = 200h.`

In this case we are treating auxiliary registers AR0 to AR7 as an array of eight pointers. To select AR[3] we make ARP =3. We then load the accumulator using the selected pointer.

Note again this is an analogy only, and in practice we would never write C code of this type. The main point to remember is that the ARP (auxiliary register pointer) is a *pointer* to a *pointer* to a data memory address.

Modifying an Auxiliary Register

Arguably the most difficult TMS320F24x instruction to master is the instruction `MAR` (modify auxiliary register).

The instruction `MAR` takes two operands, each of which is responsible for a completely different function. For the moment we shall consider the first operand only. This operand is used to change the value of an auxiliary register.

Example 7-7 gives the three operators that can be used as the first operand.

Example 7-7.

`MAR *`	`; Make no change to the auxiliary`
	`; register selected by ARP.`

`MAR *+`	`; Increment the auxiliary register`
	`; selected by ARP.`

`MAR *-`	`; Decrement the auxiliary register`
	`; selected by ARP.`

The symbol `*` must always be used, either on its own or combined with + or `-`. It is used to indicate that the auxiliary register contains the address of a data memory address. However, care needs to be taken not to read too much into the symbol `*`. It simply means that the operation is based upon the contents of the auxiliary register. *How* it is used is taken from the context of the instruction.

The second operand used with the instruction MAR changes the value of ARP. The operand itself must be one of the auxiliary registers AR0 to AR7. It selects the register for instructions using indirect addressing *following* the current instruction, not for the instruction itself.

Example 7-8.

	.setsect ".text",	8800h
start:	MAR *, AR2	; Make no change to the auxiliary ; register specified by ARP. Make ; ARP = 2. ; Auxiliary register AR2 is used ; for operations following using ; indirect addressing.
	MAR *+, AR3	; Increment AR2. Make ARP = 3. ; This instruction does *not* ; increment AR3.
	MAR *-, AR4	; Decrement AR3. Make ARP = 4. ; This instruction does *not* ; decrement AR4.
	B *start*	; Go round again.

Note that for the second operand, a number between 0 and 7 can be used instead of AR0 to AR7. The second operand may also be omitted altogether.

Example 7-9.

MAR *, 3	; ARP = 3. For instructions following ; use AR3 for indirect addressing.
MAR *	; If the second operand is omitted, ; auxiliary register AR0 will be used as ; the default.

Neither syntax in Example 7-9 is recommended. Although they will work on the TMS320F24x, they are not guaranteed to work on other Texas Instruments DSP devices i.e. they may not be compatible.

The most confusing facet of altering ARP is that is does not work on the instruction itself, but does so on the instructions following.

Differences between Instructions LAR and MAR

The instruction LAR (load auxiliary register) is used to load a value into an auxiliary register. This instruction does not increment or decrement the contents of the auxiliary register.

Example 7-10.

LAR AR2, #100h	; Load value 100h into auxiliary ; register AR2. AR2 now contains 100h.

On the other hand, the instruction `MAR` (modify auxiliary register) does not load a value into the auxiliary register. However, the instruction `MAR` can increment or decrement the contents of the current auxiliary register, as well as to modify the auxiliary register pointer (ARP). Consider the following case where the instruction `MAR` is used three times in succession.

Example 7-11.

MAR *+, AR4	; Increment the auxiliary register ; specified by ARP. Make ARP = 4. Use ; AR4 for future operations using ; indirect addressing. This does not ; increment AR4.
MAR *-, AR4	; Decrement auxiliary register specified ; by ARP (AR4). Leave ARP unchanged.
MAR *, AR4	; Does not alter auxiliary register or ; ARP. This instruction acts as a NOP.

It should be remembered that the auxiliary register to be modified is not the one specified by the operand used with the instruction `MAR`; it is the one specified by the previous instruction that put a value into ARP.

Differences between Direct Addressing and Indirect Addressing

We have already seen how to load the accumulator with the contents of a data memory address using the instruction `LACC` (load accumulator) using direct addressing. This is shown in Example 7-12. It is not necessary to change the data page pointer (DP) when working on different data pages.

Example 7-12.

CLRC CNF	; Put block B0 into data memory.
LDP #4	; Load the data page pointer (DP) with 4 ; to address page 4. Gain access to data ; memory addresses 200h to 27Fh.
LACC 3h	; Direct addressing. Load the accumulator ; with the contents of data memory ; address 200h + 3h = 203h.
LDP #6	; Load the data page pointer (DP) with 6 ; to address data page 6. Gain Access to ; data memory addresses 300h to 37Fh.

We shall now carry out exactly the same operation as shown in Example 7-12, but this time using indirect addressing. In this case the instruction `LACC` is used in a slightly different way, as shown in Example 7-13:

Example 7-13.

CLRC CNF	; Put block B0 into data memory.
MAR *, AR3	; Do not alter auxiliary register. ; Make ARP = 3. Use AR3 for future ; operations that use indirect ; addressing.
LAR AR3 #203h	; Load auxiliary register AR3 with the ; address 203h.
LACC *	; Indirect addressing. Load the ; accumulator with the contents of the ; data memory address pointed to by the ; current auxiliary register (AR3).
LAR AR3, #300h	; Gain access to data memory address ; 300h.

In Example 7-13, When used with the instruction `LACC` (load accumulator), the symbol `*` indicates indirect addressing. However, there are cases where the symbol `*` does *not* mean indirect addressing, such as when used with the instruction `MAR`.

Incrementing an Auxiliary Register as Part of an Instruction

In C code, when using pointers, the operator ++ is used to post-increment the pointer. An example of this is shown in Example 7-14.

Example 7-14.

unsigned long accumulator;	// Accumulator.
unsigned int *AR4 ;	// AR4 is a pointer to // memory.
AR4 = 0x350;	// AR4 points to address // 350h.

accumulator = *AR4++;	// Load accumulator with
	// the contents of data
	// memory address 350h, then
	// increment the pointer to
	// address 351h.

When we implement the C code in Example 7-14, the expression `*AR4++` is replaced by `*+` in assembly language.

Example 7-15.

MAR *, AR4	; Make AR4 the current auxiliary ; register. ARP = 4.
LAR AR4, #350h	; Store the address 350h in auxiliary ; register AR4.
LACC *+	; Load the accumulator with the contents ; of the data memory address pointed to ; by ARP (350h). Then increment AR4 to ; point to the next data memory address ; (351h).

Here we have used the operand `*+` to mean that the auxiliary register contains the value to be used as the basis of our operation and that after doing the operation, we increment the contents of the auxiliary register. Note that the operand `*+` increments the *address* of the data memory location, but does not increment the data contained at that data memory address.

Decrementing an Auxiliary Register as Part of an Instruction

When using C code, the operator `--` is used to post-decrement a pointer, as shown in Example 7-16.

Example 7-16.

`unsigned long accumulator;`	`// Accumulator.`

`unsigned int *AR3 ;`	`// AR3 is a pointer to a`
	`// word in memory.`

`AR3 = 0x300;`	`// AR3 points to address`
	`// 300h.`

`accumulator = *AR3--;`	`// Load the accumulator with`
	`// the contents of memory`
	`// address 300h, then`
	`// decrement the pointer to`
	`// point to the next`
	`// address (2FFh).`

We shall now implement Example 7-16 using assembly language. In order to decrement the contents of an auxiliary register when using indirect addressing we would use the operator `*-`:

Example 7-17.

`MAR *, AR3`	`; ARP = 3. Use AR3 for following`
	`; instructions using indirect`
	`; addressing.`

`LAR AR3, #300h`	`; Store address 300h in auxiliary`
	`; register AR3.`

`LACC *-`	`; Load the accumulator with the contents`
	`; of the data the memory address pointed`
	`; to by AR3 (300h). Decrement AR3 to`
	`; point to the next lower data memory`
	`; address (2FFh).`

The expression `*AR3--` in C is replaced by the operand `*-` in assembly language.

Filling Data Memory with Known Values

At the start of a program, it is often the case that we fill blocks of data memory with known values to act as defaults, rather than random values. This gives more predictable behavior.

To fill a single data memory address with 0000h we could write:

Example 7-18.

MAR *, AR5	; Make AR5 the current auxiliary ; register. ARP = 5.
LAR AR5, #320h	; Store the address 320h in auxiliary ; register AR5.
SPLK #0, *+	; Store the immediate value 0000h at the ; data memory address pointed to by ; the current auxiliary register ; (AR5 = 320h). ; Then increment the current auxiliary ; register (AR5) to point to the next ; data memory address (321h).

In order to fill a block of data memory with the same value we need to use a loop counter.

Suppose we wish to fill data memory addresses 300h to 3FFh with 0000h. For this we shall use auxiliary register AR4 as the pointer and the accumulator as the loop counter.

Example 7-19.

	.setsect “.text”,	8800h
start:	MAR *, AR4	; ARP = 4. Use AR4 for indirect ; addressing.
	LAR AR4, #300h	; AR4 points to address 300h.
	LACC #256	; Number of counts = 256.
loop:	SPLK #0, *+	; Store the immediate value 0 ; decimal at the data memory ; address pointed to by AR4. ; Increment AR4 to point to next ; data memory address.
	SUB #1	; Decrement counter.
	BCND *loop*, NEQ	; Test if counter has reached ; zero. If not, go round loop ; again.
	B *start*	; Return to beginning.

Here we have used the accumulator as a counter starting at 256, counting downwards to zero. To test for zero is straightforward because we can use the branch conditional instruction BCND with the operand NEQ. This example will loop round until the contents of the accumulator reaches zero.

There is no reason why we cannot carry out the same operation in reverse order, starting from the highest address and working down. This is shown in Example 7-20:

Example 7-20.

	.setsect ".text",	8800h
start:	MAR *, AR4	; ARP = 4. Use AR4 for indirect ; addressing.
	LAR AR4, #3FFh	; AR4 points to address 3FFh.
	LACC #100h	; Number of counts = 100h.
loop:	SPLK #55AA, *-	; Store the immediate value 55AAh ; at the data memory address ; pointed to by AR4. ; Decrement AR4 to point to next ; lower data memory address.
	SUB #1	; Decrement counter.
	BCND *loop*, NEQ	; Test if counter has reached ; zero. If not, go round loop ; again.
	B *start*	; Return to beginning.

Advantages of Indirect Addressing

Indirect addressing offers two advantages over direct addressing.

First, the auxiliary register used with indirect addressing contains the full 16-bit data memory address, whereas the operand used with direct addressing provides only 7 bits.

The second advantage of using indirect addressing is that it provides a mechanism to modify the contents of the auxiliary register as part of the instruction. This can be used with a loop counter to work across a block of data memory. With direct addressing it may be necessary to change the data page pointer (DP) between instructions.

TMS320F243 DSK Experiments

Equipment Required

TMS320F243 DSK

Experiment 7-1.

Objective: To show the effect of the instruction `MAR`

Enter the code in Example 7-8 into a text editor, save it, then assemble it and load it into the debugger. Make a note of the initial values of AR2, AR3 and AR4. Use the StepInto Button on the debugger to step through the code. Monitor Auxiliary Registers AR2, AR3 and AR4 using the CPU window. The value in AR2 should be incremented, the value in AR3 decremented and the value in AR4 unchanged.

Experiment 7-2.

Objective: To fill a block of memory with a particular value

Enter the code in Example 7-19 into a text editor and save it. Assemble the program and load into the debugger. Use the Edit-Fill Memory option to fill memory addresses with a value other than 0000h. Run the program. Check that each of the data memory addresses in the range 300h to 3FFh is filled with the value 0000h.

Experiment 7-3.

Objective: To fill a block of memory in reverse order

Enter the code in Example 7-20 into a text editor and save it. Assemble the program and load into the debugger. Run the program. Check using a Memory window that the data memory addresses 300h to 3FFh each contains 55AAh. Check that data memory addresses 2FFh and 400h have not been overwritten with 55AAh.

Design Problem 7-1.

Modify the code in Example 7-19 so that it fills data memory addresses 200h to 3FFh with the value AA55h. Note that there will be the need to set the `CNF` flag the appropriate value in order to do so.

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

1.	What are the differences between *direct addressing* and *indirect addressing*?
2.	Which two of the following instructions are correct? a) `LAR #400h, AR4` b) `LAR #20, AR5` c) `LAR AR1, #200h` d) `LAR AR7+, *` e) `LAR -AR5, *` f) `LAR AR6, 30h`
3.	What is meant by the term *auxiliary register pointer* (ARP)?
4.	What is meant by the term *current auxiliary register*?
5.	The instruction `MAR` means which of the following? a) Match all registers b) Modify all registers c) Modify auxiliary register d) Move all registers e) Multiply and raise
6.	Which three of the following correctly use the instruction `MAR`? a) `MAR *` b) `MAR -*` c) `MAR +*` d) `MAR *-` e) `MAR *+`
7.	Which three of the following `LACC` instructions are correct? a) `LACC *` b) `LACC -*` c) `LACC +*` d) `LACC *-` e) `LACC *+` f) `LACC *.*`
8.	What are the differences between the instructions `LAR` and `MAR`?
9.	How do we increment the contents of auxiliary register AR4 when using indirect addressing to load the accumulator with data from data memory address 200h?
10.	How do we decrement the contents of auxiliary register AR3 when using indirect addressing to load the accumulator with data from data memory address 180h?
11.	How do we make AR6 the current auxiliary register?
12.	Assuming that ARP =3, what effect does the instruction `LACC *+` have? a) Load accumulator with data then increment AR3 b) Load accumulator with data then increment accumulator A c) Load accumulator with data then increment indirect data.
13.	What are the advantages of *indirect addressing* over *direct addressing*?
14.	When loading the accumulator using the instruction `LACC`, we have the choice of using *direct addressing* and *indirect addressing*. Which form of addressing is more efficient in terms of clock cycles. Refer to SPRU160.

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