Addressing Modes for TI TMS320C55x

Version 1.4: Sep 3, 2007 2:10 pm GMT-5

Thomas Shen
Douglas L. Jones

Based on Addressing Modes for TI TMS320C54x* by
Douglas L. Jones
Swaroop Appadwedula
Matthew Berry
Mark Haun
Dima Moussa
Daniel Sachs

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

The TI TMS320C55x microprocessor provides a number of ways to specify the location of data to be used in calculations. Immediate addressing, direct addressing, and indirect addressing are the three main types. Knowing the basic addressing modes of a microprocessor is important because they map directly into assembly language syntax and because the need to use a particular addressing mode often dictates which instruction one picks for a given task.

Microprocessors provide a number of ways to specify the location of data to be used in calculations. For example, one of the data values to be used in an add instruction may be encoded as part of that instruction’s opcode, the raw machine language produced by the assembler as it parses your assembly language program. This is known as immediate addressing. Alternatively, perhaps the opcode will instead contain a memory address which holds the data (direct addressing). More commonly, the instruction will specify that an auxiliary register holds the memory address which in turn holds the data (indirect addressing). The processor knows which addressing mode is being used by examining special bit fields in the instruction opcode.

Knowing the basic addressing modes of your microprocessor is important because they map directly into assembly language syntax. Many annoying and sometimes hard-to-find bugs are caused by inadvertently using the wrong addressing mode in an instruction. Also, in any assembly language, the need to use a particular addressing mode often dictates which instruction one picks for a given task.

Chapter six, Addressing modes, in the CPU Reference Guide contains extended descriptions of the addressing modes described below.

1 Accumulators: ACw, ACx, ACy, ACz, xsrc, xdst, src, dst

Whenever the abbreviations ACw, ACx, ACy, ACz, xsrc or xdst are used in the assembly language syntax description for an instruction, it means that only the accumulators AC0, AC1, AC2 and AC3 may be used for

*http://cnx.org/content/m10806/2.7/
†http://creativecommons.org/licenses/by/2.0/
that particular operand. At times, src and dst may also be referring to the accumulators.

Examples:

\[
\text{MOV } *\text{AR3}, *\text{AR4}, \text{AC0} ; \text{AC0}(15-0) = \text{contents of memory location pointed to by AR3} \\
; \text{AC0}(39-16) = \text{contents of memory location pointed to by AR4} \\
; \text{sign extended to 24 bits}
\]

\[
\text{MOV } \text{AC0}, \text{AC1} ; \text{AC1} = \text{AC0}
\]

\[
\text{MOV } \text{AC0}, *\text{AR3} ; \text{sets content of memory location pointed to by AR3} = \text{AC0}(15-0)
\]

\[
\text{MOV } \text{HI(AC0)}, *\text{AR7}+ ; \text{sets (contents of memory location pointed to by AR7)} = \text{AC0}(31-16), \\
; \text{and then increments AR7 by one}
\]

2 Memory-mapped Registers: MMR, MMRx, MMRy

Many of the TMS320C55x registers are memory-mapped, meaning that they occupy real addresses at the low end of data memory space. The most commonly used of these are the accumulators AC0 through AC3, auxiliary registers AR0 through AR7. The temporary registers T0 - T3, BK, and various BSA (buffer start address) are also memory mapped. Memory mapped registers are stored from 00 0000h to 00 005Fh.

An mmr prefix can be used for indirect memory operands to assert that the memory access is to a memory-mapped register. This prefix can be used on Xmem, Ymem, indirect Smem, indirect Lmem, and Cmem operands. Refer to 1-19 of the SPRU374 for more information if necessary.

The mmap qualifier can be used with numerous instructions to force an access to a memory-mapped register. It can be used with Smem or Lmem direct memory access to prevent the dma access from being relative to the SP or DP. Instead, it will be relative to the memory-mapped register data page start address, which is 00 0000h.

Examples:

\[
\text{MOV } \text{mmap(AR6)}, \text{BSA67} ; \text{sets BSA67} = \text{value in AR6} \\
; \text{mmap must be used}
\]

\[
\text{MOV } \text{AR1, AR5} ; \text{sets AR5} = \text{AR1}
\]

3 Immediate Addressing: #k3, #k5, K, #k9, #lk

Immediate addressing means that the numerical value of the data is itself provided within the assembly instruction. Various TMS320C55x instructions allow immediate data of 3, 4, 5, 7, 8, 9, 12, 16, 23 bits in length, which are signified in the assembly language syntax descriptions with one of the above symbols. The 16-bit form is the most common and is signified by #k16.

An immediate data operand is almost always specified in assembler syntax by prepending a pound sign (#) to the data. Depending on the context, the assembler may assume that you meant immediate addressing anyway.

Examples:
ADD #FFFFh, *AR3 ; The content addressed by AR3 is added to a signed 16-bit value and the result is stored back into the location addressed by AR3
AMOV #7FFFFFh, XAR0 ; The 23-bit value is loaded into XAR0

Labels make this more complicated. Recall that a label in your assembly code is nothing more than shorthand for the memory address where the labeled code or data is stored. So does an instruction like

MOV      #coef, AR1 ; sets AR1 = memory address of label coef

mean to store the contents of memory location coef in AR1, or does it mean to store the memory address coef itself in AR1? The second interpretation is correct.

Many instructions have several versions allowing the use of different addressing modes (see mov for a good example of this). With these instructions, including the pound sign is not optional when specifying immediate addressing. The only safe rule, then, is always to prefix the label with a pound sign if you wish to specify the memory address of the label and not the contents of that address.

4 Direct Addressing: Smem and others

In the modes called direct addressing by TI, the memory offset is combined with values in the DPH (the high part of the extended data page register) and DP (data page register) or the SPH (high part of the extended stack pointer) and SP (data stack pointer) to obtain a complete 23-bit data-memory address. The DSP uses SPH/SP or DPH/DP depending on the value of the CPL bit in status register ST1_55.

For our purposes, the CPL bit is set and so we will use SPH/SP for direct addressing. SP is initialized for you in the core file and should not need to be modified. SP-referenced direct addressing is used by the psh and pop instructions for stack manipulation, as well as by all subroutine calls and returns, which save program addresses on the stack.

Examples:

; Assuming SPH = 0, SP = FF00h,
MOV *SP(5) , T2 ; T2 = value at location 00FF05h

; Assuming DPH = 3, DP = 0
MOV 00005h, T2 ; T2 = value at location 030005h
ADD *SP(6), AC0 ; AC0 = AC0 + (contents of memory location SPH:SP+6)

5 Absolute Addressing: *abs16(#k16), *(#k23)

This seems to be TI’s term for all the forms of direct addressing which it does not call direct addressing! There are three types of absolute addressing: k16, k23, and I/O. We will only be using the first two. It is represented in assembly-instruction syntax-definitions using one of the above abbreviations (* (lk) addressing is available when the syntax definition says Smem or Lmem).

http://cnx.org/content/m14262/1.4/
5.1 k16

k16 absolute addressing uses the operand *abs16(#k16) along with the 7-bit DPH to form a 23-bit address.

Example:

; Assuming DPH = 3
MOV *abs16(#2002h), T2 ; T2 = value at address 032002h

5.2 k23

k23 absolute addressing uses the operand *(#k23) as a 23-bit address.

Example:

MOV *(#032002h), T2 ; T2 = value at location 032002h
MOV AR1, *(#hold) ; sets (storage location at hold) = AR1

6 Indirect Addressing: Smem, Lmem, Xmem, Ymem, Cmem

Indirect addressing on the TMS320C55x uses the auxiliary registers AR0 through AR7 and the CDP. They can be used in place of Smem/Lmem or Xmem/Ymem.

6.1 AR Indirect: Smem/Lmem

In Smem/Lmem indirect addressing, only one indirect address is used in the instruction and a number of variations is possible (see the table on page 6-39 of the CPU Reference guide). An asterisk is always used, which usually signifies indirect addressing. Any of the registers AR0-AR7 may be used, with optional modifications: automatic post-decrement by one, pre- and post-increment by one, post-increment and post-decrement by n (n being stored in T0, T1, or AR0), and more, including many options for circular addressing (which automatically implements circular buffers) and bit-reversed addressing (which is useful for FFTs).

6.2 Dual AR Indirect: Xmem/Ymem

Xmem/Ymem indirect addressing is generally used in instructions that need two different indirect addresses, although there are a few instances where an Xmem by itself is specified in order to save bits in the opcode for other options. In Xmem/Ymem indirect addressing, fewer bits are used to encode the option modifiers in the opcode; hence, fewer options are available: post-increment by one, post-decrement by one, and post-increment by AR0, T0, or T1 with circular addressing.

ADD Xmem, Ymem, ACx

ADD *AR1+, *AR2+, AC0 ; Add values stored in memory locations referenced by
; AR1 and AR2 and store result in AC0.
; Increment AR1 and AR2 by 1 with or without circular
; addressing depending on circular addressing bit
; for the respective auxiliary registers
6.3 CDP Indirect : Cmem

CDP indirect addressing uses the coefficient data pointer (CDP) to point to data. For accessing data memory/registers, the 16-bit CDP is combined with the 7-bit CDPH to generate a 23-bit address. When concatenated, they are called the XCDP. CDP indirect addressing can also be used to address a specific bit in the accumulators, auxiliary registers, and the temporary registers. Pre- and post-increment and decrement as well as an offset can be used with CDP. CDP can be used as an operand in place of Smem, Lmem, and Cmem.

```assembly
MOV dbl(Lmem), Cmem
MOV dbl(*AR7), *CDP+ ; Values at XAR7 and XAR7 + 1 are read and stored at
                     ; XCDP and XCDP +/- 1 depending on if XCDP was even or odd.
                     ; CDP is incremented by 2 at the end.
```

6.4 Coefficient Indirect

Coefficient indirect addressing uses the same address generation process as CDP indirect addressing for data-space accesses. It is useful for instructions that need three memory operands per cycle. It can be used for finite impulse response filters, multiply [accumulate/subtract], and dual multiply [accumulate/subtract].

```assembly
MPY Xmem, Cmem, ACx
 :: MPY Ymem, Cmem, ACy ; Cmem must be in a different memory bank from Xmem/Ymem
                     ; for this to work in a single cycle
MPY *AR1+, *CDP+, AC0
 :: MPY *AR2+, *CDP+, AC1 ; The value at address XAR1 is multiplied by value at
                     ; address XCDP and stored in AC0. At the same time,
                     ; value at XAR2 is multiplied by value at address XCDP
                     ; and stored in AC1. Then CDP is incremented.
```

6.5 Other Examples

```assembly
AMAR *AR3+ ; increments AR3 by 1
```

**Note:** The `amar` (modify auxiliary register) instruction is unusual in the sense that it takes an Smem operand but does nothing with the data pointed to by the ARx register. Its purpose is to perform any of the allowed register modifications discussed above without having to do anything else. This is often handy when you are using an Xmem/Ymem-type instruction but need to do an ARx modification that is only allowed with an Smem-type operand.
7 Circular Addressing

Circular addressing is useful when implementing circular buffers. Circular addressing needs to be enabled for the specific register that is being used to point to memory. This is done by setting the corresponding ARnLC register using BSET ARnLC. When circular addressing is not needed, BCLR ARnLC on the corresponding ARn will disable circular addressing. The circular addressing length will depend on the BK03, BK47, and BKC register values. If you are using AR0 through AR3 for the addressing, then BK03 will be used. BKC is the buffer length for CDP. One thing to watch out for is the buffer start address registers (BSA01, BSA23, BSA45, BSA67, BSAC) are added to the auxiliary register or CDP register value whenever circular addressing is used. Be sure to re-initialize the BSA register when implementing multiple filters.

```
BSET AR1LC ; sets circular addressing for AR1
BCLR AR2LC ; normal addressing for AR2
MOV #13, BK03 ; set buffer length for AR0 through AR3
MACM *AR1+, *AR2+, AC0 ; AC0 = AC0 +
; (value at memory location AR1 + BSA01) x (value at memory location AR2)
; AR1 is incremented with circular addressing, length 13.
; AR2 is simply incremented by 1
```

8 Bit-Reversed Addressing

Bit-reversed addressing is often needed with the fast-fourier transform (FFT). This helps to set up a pointer for the next iteration. Enable bit-reversing on an operand by adding a B after the increment value. When a bit-reverse operand is used, the auxiliary register can not be used for circular addressing. If circular address is enabled, the corresponding buffer start address value (BSAxx) will be added to ARn, but ARn is not modified to stay within the circular buffer.

```
MOV Smem, dst
MOV *(AR4 + T0B), T2 ; T2 = value at memory location XAR4
; AR4 is incremented with T0, using reverse carry propagation.
```