MODULE-3

PROGRAMMABLE DIGITAL SIGNAL PROCESSORS

❖ LEARNING OBJECTIVES

- ➢ Introduction,
- ➢ Commercial digital Signal-processing Devices,
- ➢ Data Addressing Modes of TMS32OC54xx.,
- ➢ Memory Space of TMS32OC54xx
- ➢ Processors,
- ➢ Program Control.

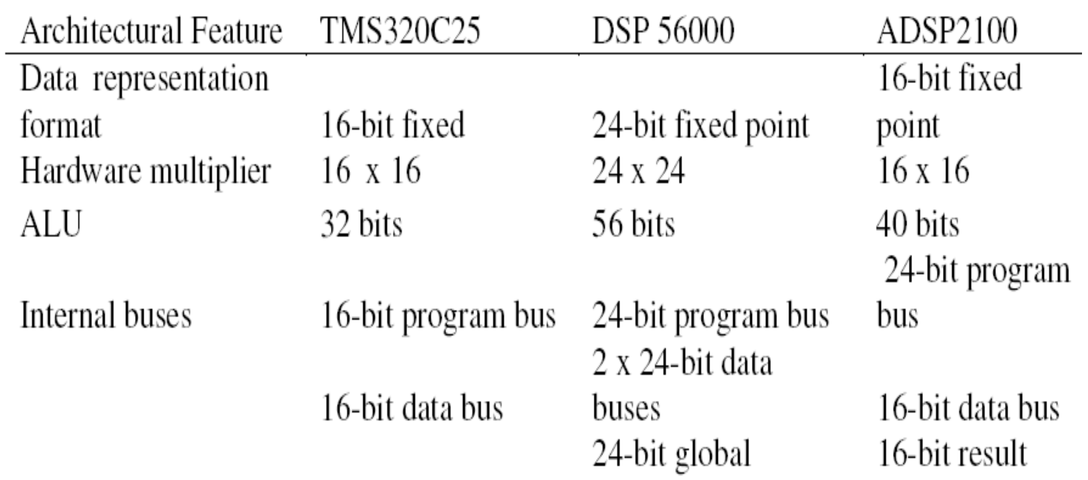
Introduction:

Leading manufacturers of integrated circuits such as Texas Instruments (TI), Analog devices & Motorola manufacture the digital signal processor (DSP) chips. These manufacturers have developed a range of DSP chips with varied complexity.

The TMS320 family consists of two types of single chips DSPs: 16-bit fixed point &32-bit floating- point. These DSPs possess the operational flexibility of high-speed controllers and the numerical capability of array processors

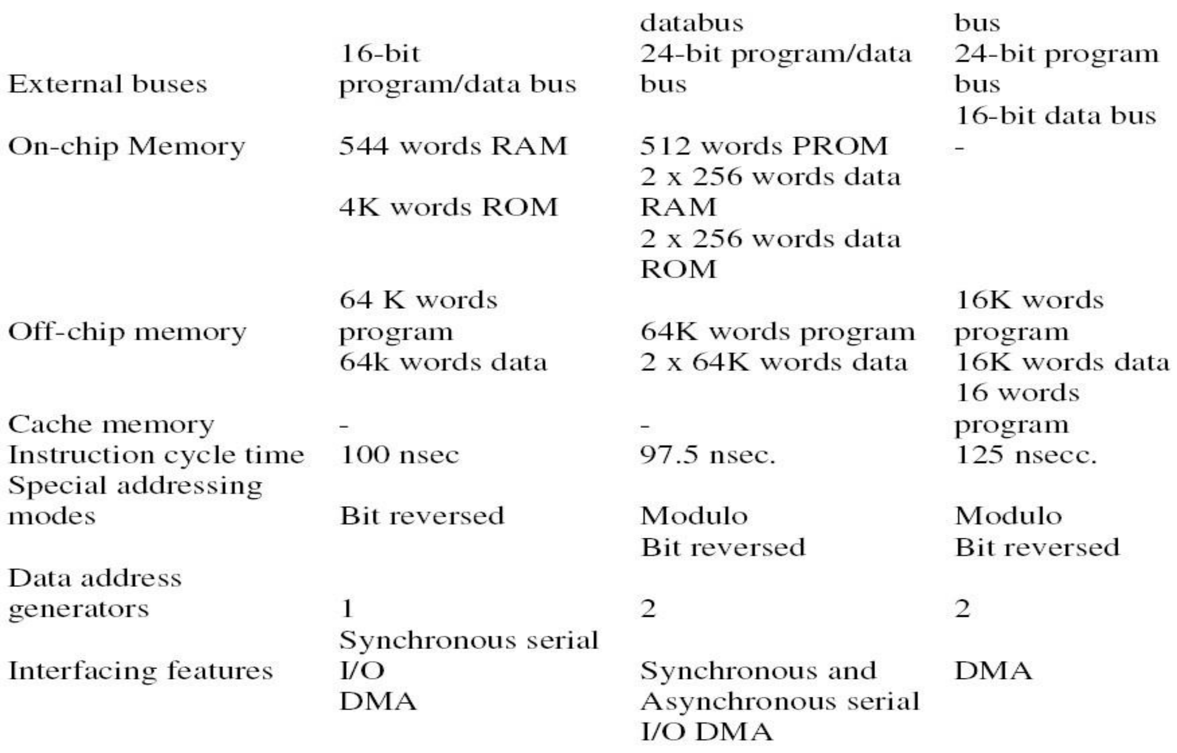
Commercial Digital Signal-Processing Devices:

Summary of the Architectural Features of three fixed-Points DSPs



Architectural Feature	TMS320C25	DSP 56000	ADSP2100
Data representation format	16-bit fixed	24-bit fixed point	16-bit fixed point
Hardware multiplier	16 x 16	24 x 24	16 x 16
ALU	32 bits	56 bits	40 bits
Internal buses	16-bit program bus	24-bit program bus	24-bit program bus
	16-bit data bus	2 x 24-bit data buses	16-bit data bus
		24-bit global	16-bit result

There are several families of commercial DSP devices. Right from the early eighties, when these devices began to appear in the market, they have been used in numerous applications, such as communication, control, computers, Instrumentation, and consumer electronics. The architectural features and the processing power of these devices have been constantly upgraded based on the advances in technology and the application needs. However, their basic versions, most of them have Harvard architecture, a single-cycle hardware multiplier, an address generation unit with dedicated address registers, special addressing modes, on-chip peripherals interfaces. Of the various families of programmable DSP devices that are commercially available, the three most popular ones are those from Texas Instruments, Motorola, and Analog Devices. Texas Instruments was one of the first to come out with a commercial programmable DSP with the introduction of its TMS32010 in 1982.



		databus	bus
External buses	16-bit program/data bus	24-bit program/data bus	24-bit program bus 16-bit data bus
On-chip Memory	544 words RAM 4K words ROM	512 words PROM 2 x 256 words data RAM 2 x 256 words data ROM	-
Off-chip memory	64 K words program 64k words data	64K words program 2 x 64K words data	16K words program 16K words data
Cache memory	-	-	16 words program
Instruction cycle time	100 nsec	97.5 nsec.	125 nsecc.
Special addressing modes	Bit reversed	Modulo Bit reversed	Modulo Bit reversed
Data address generators	1	2	2
Interfacing features	Synchronous serial I/O DMA	Synchronous and Asynchronous serial I/O DMA	DMA

The architecture of TMS320C54xx digital signal processors:

TMS320C54xx processors retain in the basic Harvard architecture of their predecessor, TMS320C25, but have several additional features, which improve their performance over it. Figure 3.1 shows a functional block diagram of TMS320C54xx processors. They have one program and three data memory spaces with separate buses, which provide simultaneous accesses to program instruction and two data operands and enables writing of result at the same time. Part of the memory is implemented on-chip and consists of combinations of ROM, dual-access RAM, and single-access RAM. Transfers between the memory spaces are also possible.

The central processing unit (CPU) of TMS320C54xx processors consists of a 40- bit arithmetic logic unit (ALU), two 40-bit accumulators, a barrel shifter, a 17x17 multiplier, a 40-bit adder, data address generation logic (DAGEN) with its own arithmetic unit, and program address generation logic (PAGEN). These major functional units are supported by a number of registers and logic in the architecture. A powerful instruction set with a hardware-supported, single-instruction repeat and block repeat operations, block memory move instructions, instructions that pack two or three simultaneous reads, and arithmetic instructions with parallel store and load make these devices very efficient for running high-speed DSP algorithms.

Several peripherals, such as a clock generator, a hardware timer, a wait state generator, parallel I/O ports, and serial I/O ports, are also provided on-chip. These peripherals make it convenient to interface the signal processors to the outside world. In these following sections, we examine in detail the various architectural features of the TMS320C54xx family of processors.

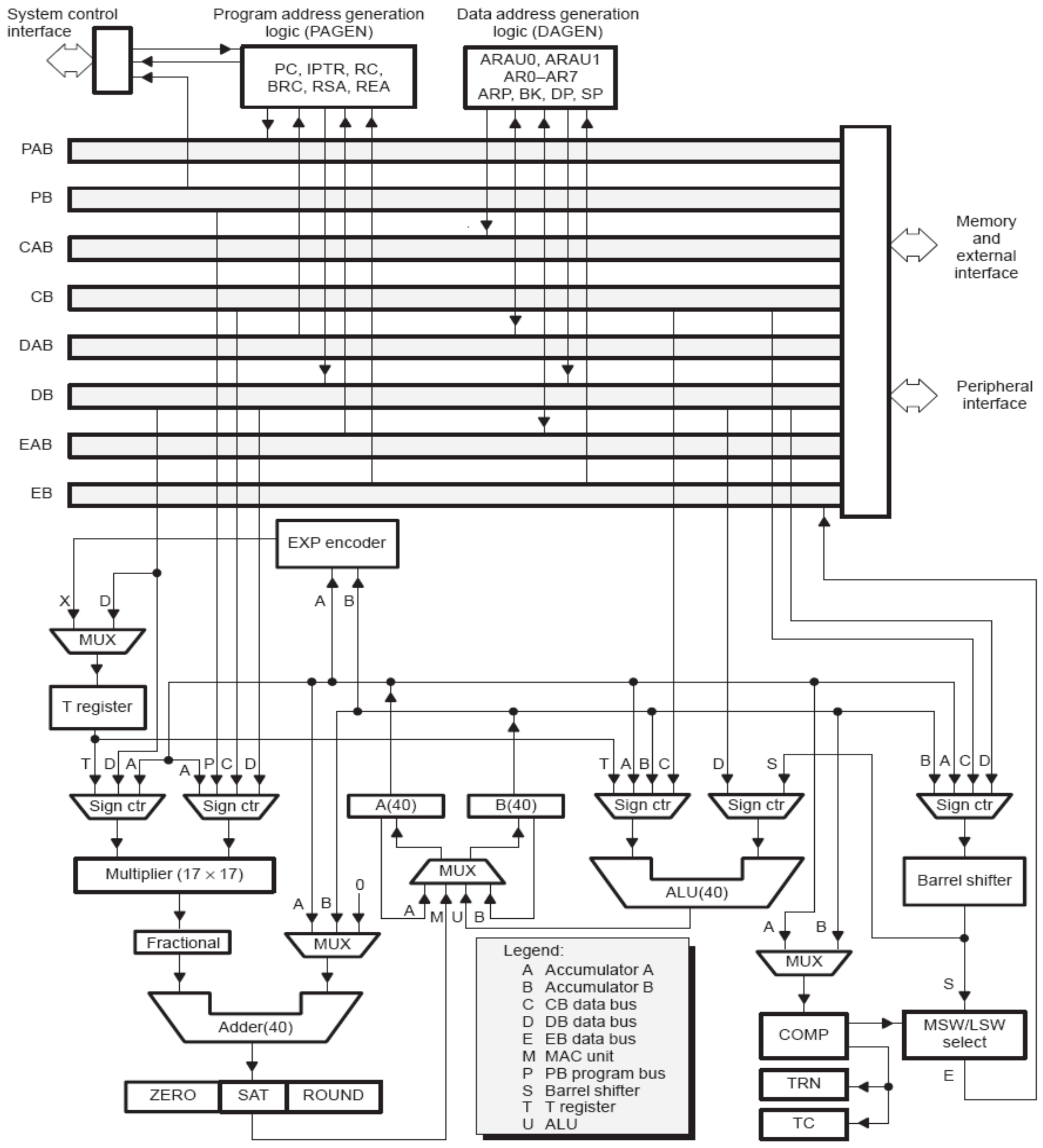


Figure 3.1.Functional architecture for TMS320C54xx processors.

Bus Structure:

The performance of a processor gets enhanced with the provision of multiple buses to provide simultaneous access to various parts of memory or peripherals. The 54xx architecture is built around four pairs of 16-bit buses with each pair consisting of an address bus and a data bus. As shown in Figure 3.1, these are The program bus pair (**PAB, PB**); which carries the instruction code from the program memory. Three data bus pairs (**CAB, CB; DAB, DB**; and **EAB, EB**); which interconnected the various units within the CPU. In Addition the pair CAB, CB and DAB, DB are used to read from the data memory, while The pair **EAB, EB**; carries the data to be written to the memory. The ‘54xx can generate up to two data-memory addresses per cycle using the two auxiliary register arithmetic unit (ARAU0 and ARAU1) in the DAGEN block. This enables accessing two operands simultaneously.

Central Processing Unit (CPU):

The ‘54xx CPU is common to all the ‘54xx devices. The ’54xx CPU contains a 40-bit arithmetic logic unit (**ALU**); two 40-bit accumulators (**A** and **B**); a barrel shifter; a 17 x 17-bit multiplier; a 40-bit adder; a compare, select and store unit (**CSSU**); an exponent encoder(**EXP**); a data address generation unit (**DAGEN**); and a program address generation unit (**PAGEN**).

The ALU performs 2’s complement arithmetic operations and bit-level Boolean operations on 16, 32, and 40-bit words. It can also function as two separate 16-bit ALUs and perform two 16-bit operations simultaneously. Figure 3.2 show the functional diagram of the ALU of the TMS320C54xx family of devices.

Accumulators A and B store the output from the ALU or the multiplier/adder block and provide a second input to the ALU. Each accumulators is divided into three parts: guards bits (bits 39-32), high- order word (bits-31-16), and low-order word (bits 15- 0), which can be stored and retrieved individually. Each accumulator is memory-mapped and partitioned. It can be configured as the destination registers. The guard bits are used as a head margin for computations.

AG(39-32)	AH(31-16)	AL(15-0)

BG(39-32)	BH(31-16)	BL(15-0)

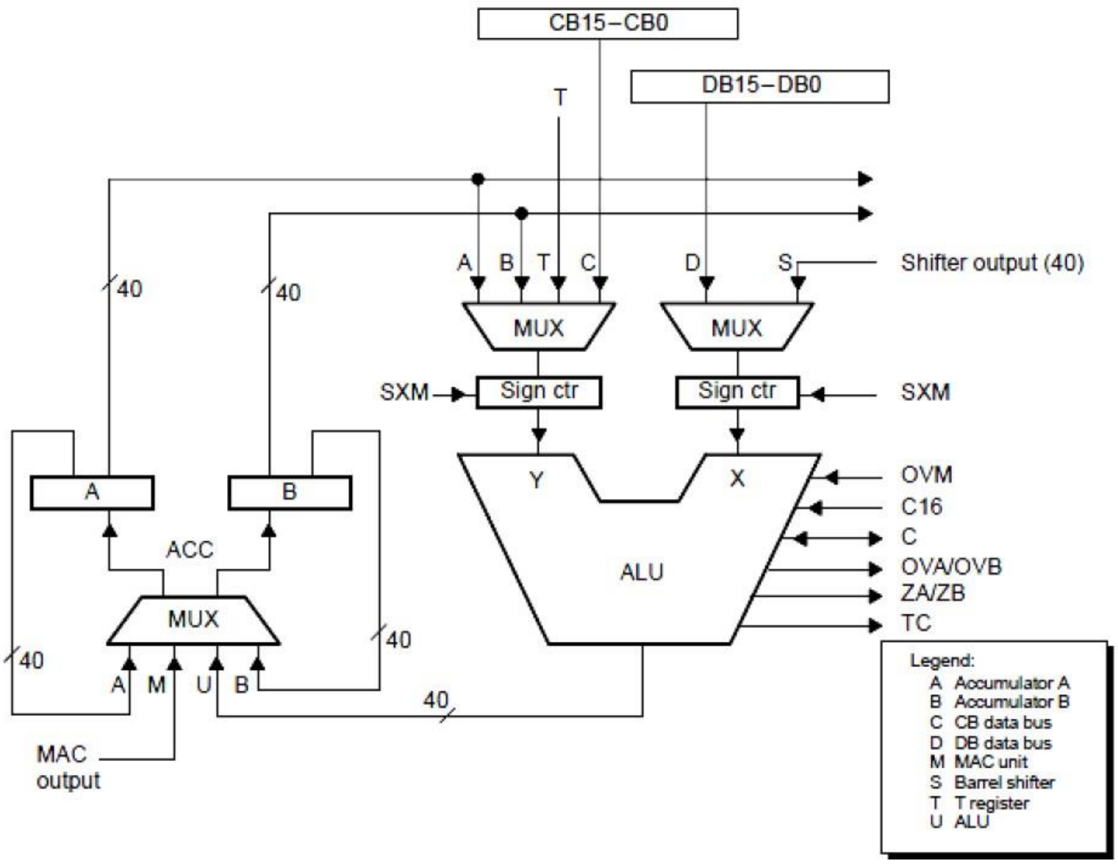


Figure 3.2.Functional diagram of the central processing unit of the TMS320C54xx processors.

Barrel shifter: provides the capability to scale the data during an operand read or write. No overhead is required to implement the shift needed for the scaling operations. The'54xx barrel shifter can produce a left shift of 0 to 31 bits or a right shift of 0 to 16 bits on the input data. The shift count field of status registers ST1, or in the temporary register T. Figure 3.3 shows the functional diagram of the barrel shifter of TMS320C54xx processors. The barrel shifter and the exponent encoder normalize the values in an accumulator in a single cycle. The LSBs of the output are filled with0s, and the MSBs can be either zero filled or sign extended, depending on the state of the sign-extension mode bit in the status register ST1. An additional shift capability enables the processor to perform numerical scaling, bit extraction, extended arithmetic, and overflow prevention operations.

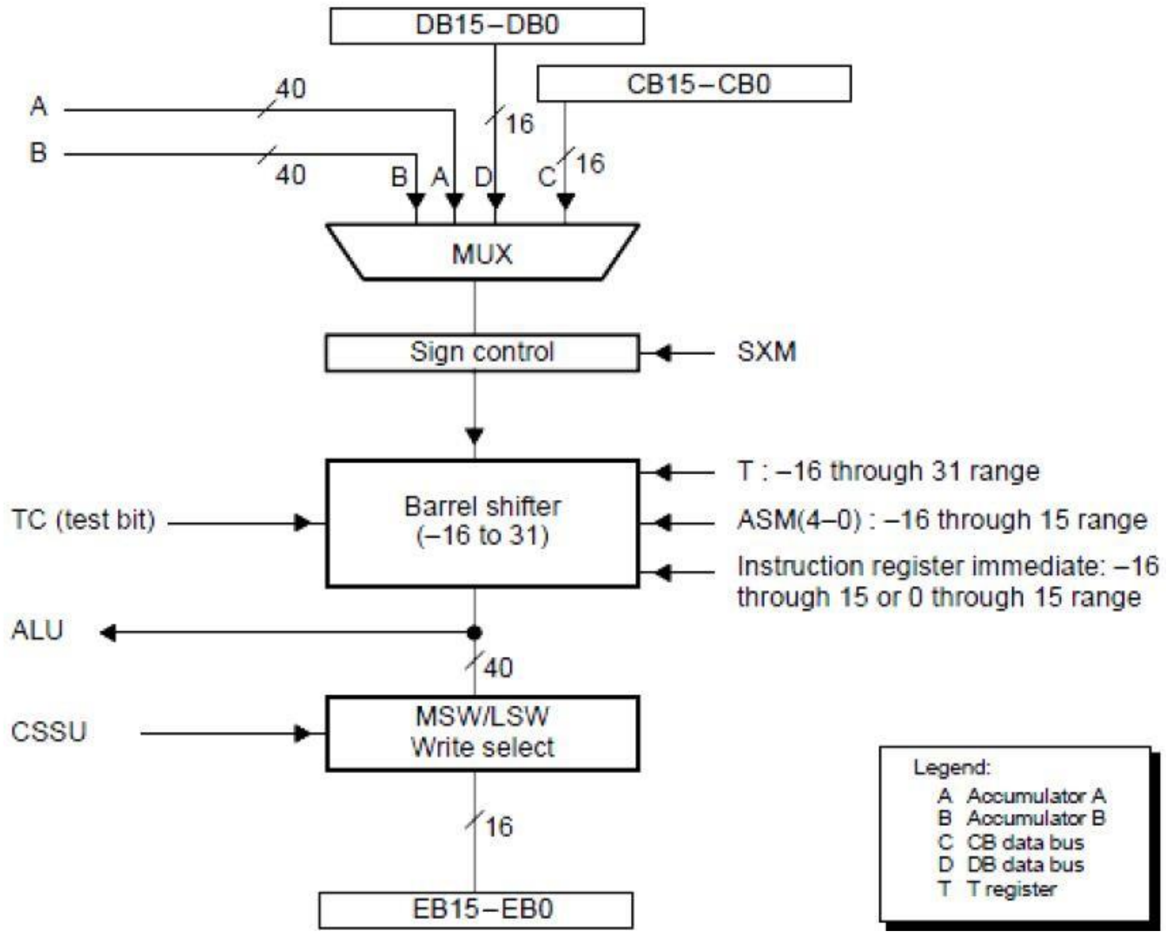


Figure 3.3.Functional diagram of the barrel shifter

Multiplier/adder unit: The kernel of the DSP device architecture is multiplier/adder unit. The multiplier/adder unit of TMS320C54xx devices performs 17 x 17 2's complement multiplication with a 40-bit addition effectively in a single instruction cycle.

In addition to the multiplier and adder, the unit consists of control logic for integer and fractional computations and a 16-bit temporary storage register, T. Figure 3.4 show the functional diagram of the multiplier/adder unit of TMS320C54xx processors. The compare, select, and store unit (CSSU) is a hardware unit specifically incorporated to accelerate the add/compare/select operation. This operation is essential to implement the *Viterbi* algorithm used in many signal-processing applications. The exponent encoder unit supports the EXP instructions, which stores in the T register the number of leading redundant bits of the accumulator content. This information is useful while shifting the accumulator content for the purpose of scaling.

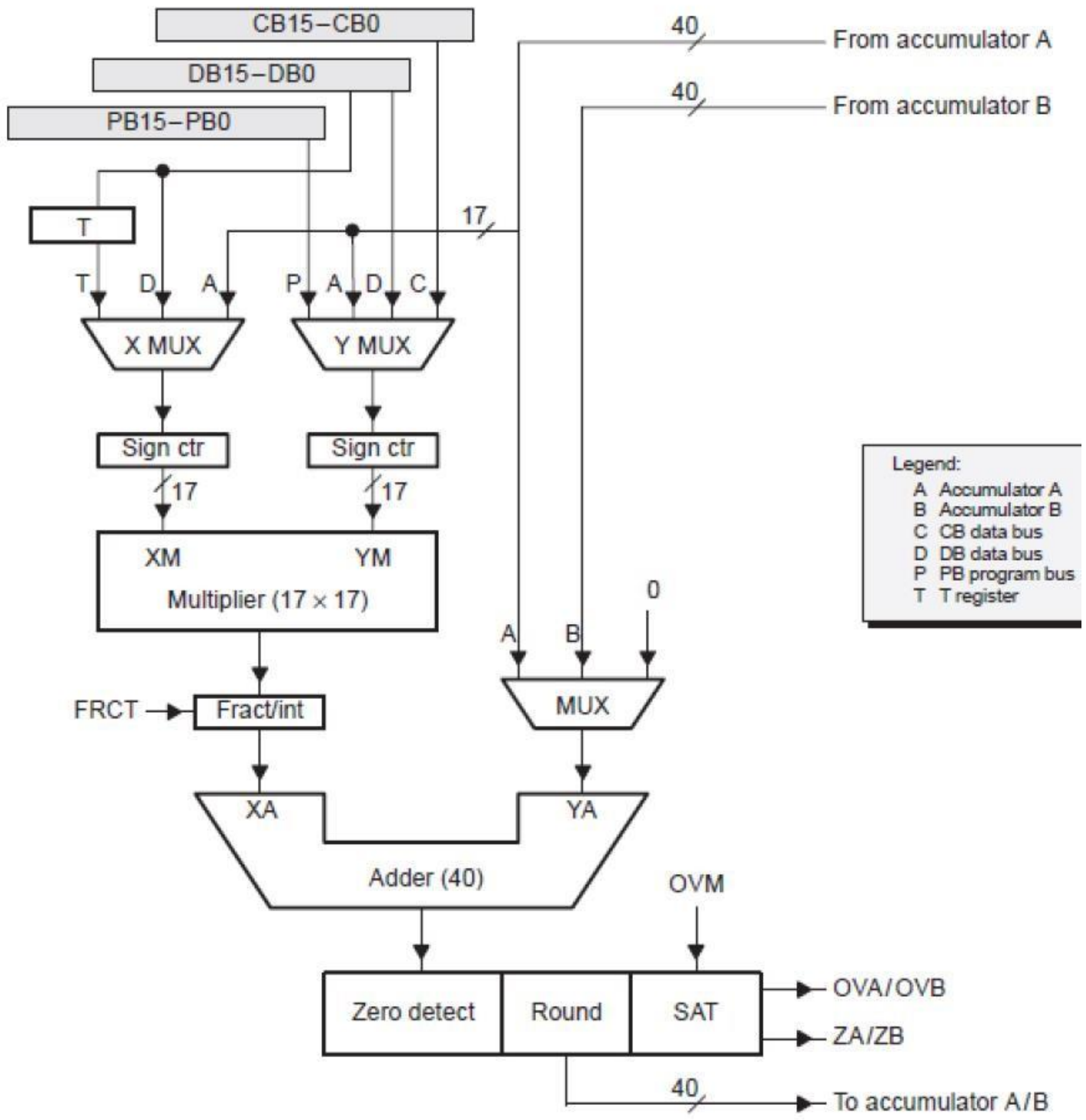
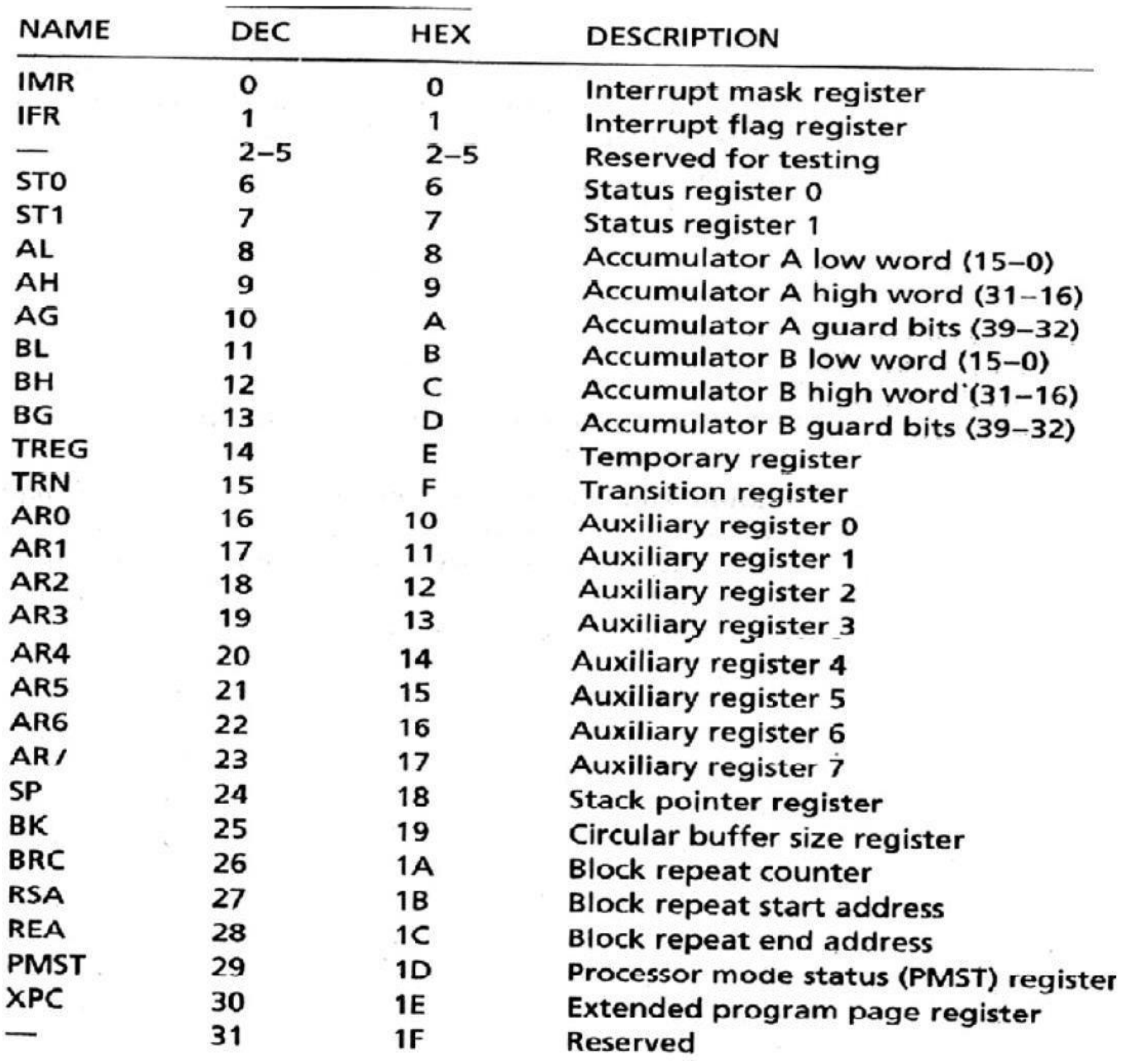


Figure 3.4. Functional diagram of the multiplier/adder unit of TMS320C54xx processors.

Internal Memory and Memory-Mapped Registers:

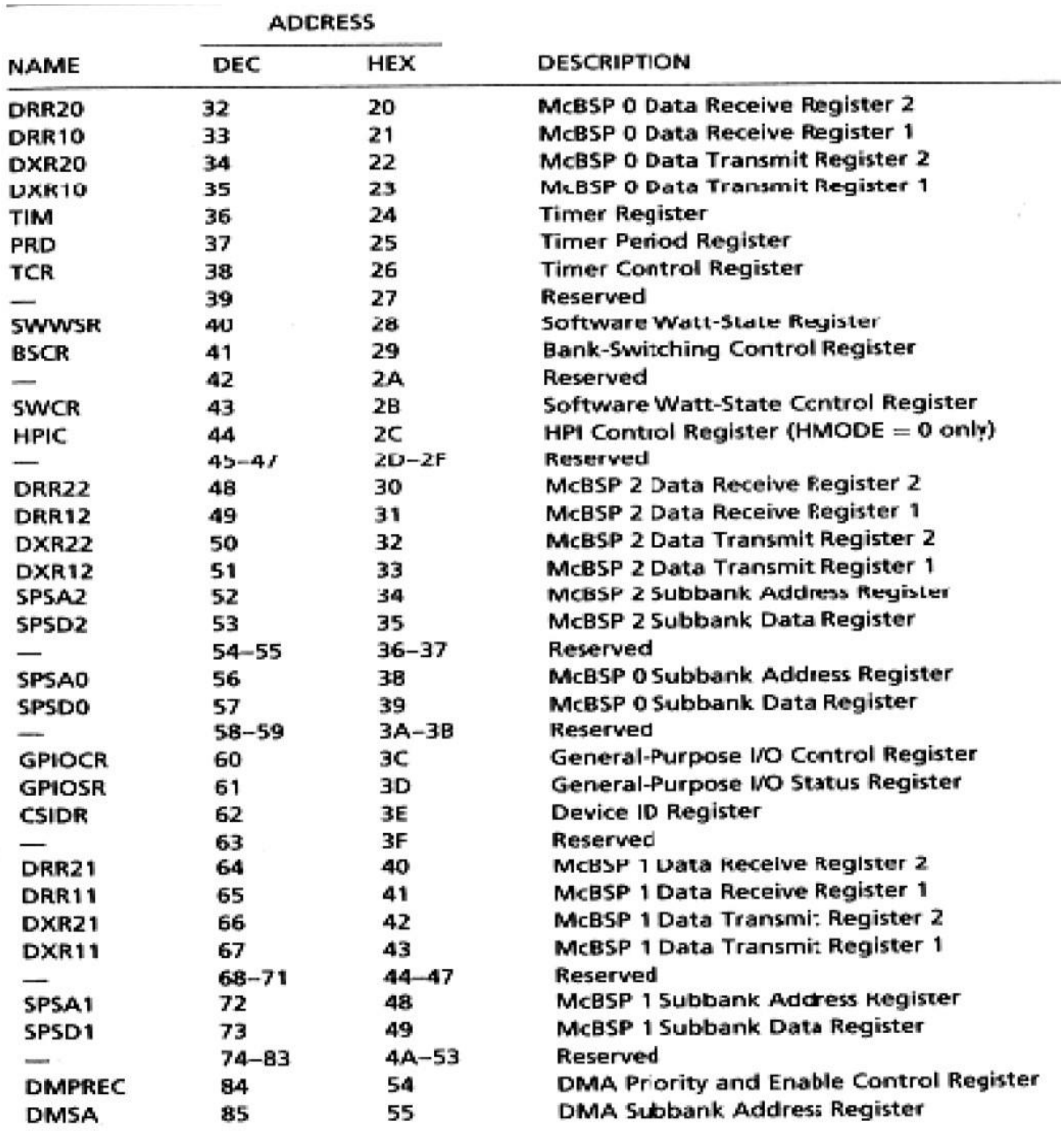
The amount and the types of memory of a processor have direct relevance to the efficiency and performance obtainable in implementations with the processors. The ‘54xx memory is organized into three individually selectable spaces: program, data, and I/O spaces. All ‘54xx devices contain both RAM and ROM. RAM can be either dual-access type (DARAM) or single-access type (SARAM). The on-chip RAM for these processors is organized in pages having 128 word locations on each page.

The ‘54xx processors have a number of CPU registers to support operand addressing and computations. The CPU registers and peripherals registers are all located on page 0 of data memory. Figure 3.5(a) and (b) shows the internal CPU registers and peripheral registers with their addresses. The processors mode status (PMST) registers that is used to configure the processor. It is a memory-mapped register located at address 1Dh on page 0 of the RAM. A part of on-chip ROM may contain a boot loader and look-up tables for function such as sine, cosine, μ*- law, and A*- law.



NAME	DEC	HEX	DESCRIPTION
IMR	0	0	Interrupt mask register
IFR	1	1	Interrupt flag register
—	2–5	2–5	Reserved for testing
ST0	6	6	Status register 0
ST1	7	7	Status register 1
AL	8	8	Accumulator A low word (15–0)
AH	9	9	Accumulator A high word (31–16)
AG	10	A	Accumulator A guard bits (39–32)
BL	11	B	Accumulator B low word (15–0)
BH	12	C	Accumulator B high word (31–16)
BG	13	D	Accumulator B guard bits (39–32)
TREG	14	E	Temporary register
TRN	15	F	Transition register
AR0	16	10	Auxiliary register 0
AR1	17	11	Auxiliary register 1
AR2	18	12	Auxiliary register 2
AR3	19	13	Auxiliary register 3
AR4	20	14	Auxiliary register 4
AR5	21	15	Auxiliary register 5
AR6	22	16	Auxiliary register 6
AR7	23	17	Auxiliary register 7
SP	24	18	Stack pointer register
BK	25	19	Circular buffer size register
BRC	26	1A	Block repeat counter
RSA	27	1B	Block repeat start address
REA	28	1C	Block repeat end address
PMST	29	1D	Processor mode status (PMST) register
XPC	30	1E	Extended program page register
—	31	1F	Reserved

Figure 3.5(a) Internal memory-mapped registers of TMS320C54xx processors.



	ADDRESS		
NAME	DEC	HEX	DESCRIPTION
DRR20	32	20	McBSP 0 Data Receive Register 2
DRR10	33	21	McBSP 0 Data Receive Register 1
DXR20	34	22	McBSP 0 Data Transmit Register 2
DXR10	35	23	McBSP 0 Data Transmit Register 1
TIM	36	24	Timer Register
PRD	37	25	Timer Period Register
TCR	38	26	Timer Control Register
—	39	27	Reserved
SWWSR	40	28	Software Wait-State Register
BSCR	41	29	Bank-Switching Control Register
—	42	2A	Reserved
SWCR	43	2B	Software Wait-State Control Register
HPIC	44	2C	HPI Control Register (HMODE = 0 only)
—	45–47	2D–2F	Reserved
DRR22	48	30	McBSP 2 Data Receive Register 2
DRR12	49	31	McBSP 2 Data Receive Register 1
DXR22	50	32	McBSP 2 Data Transmit Register 2
DXR12	51	33	McBSP 2 Data Transmit Register 1
SPSA2	52	34	McBSP 2 Subbank Address Register
SPSD2	53	35	McBSP 2 Subbank Data Register
—	54–55	36–37	Reserved
SPSA0	56	38	McBSP 0 Subbank Address Register
SPSD0	57	39	McBSP 0 Subbank Data Register
—	58–59	3A–3B	Reserved
GPIOCR	60	3C	General-Purpose I/O Control Register
GPIOSR	61	3D	General-Purpose I/O Status Register
CSIDR	62	3E	Device ID Register
—	63	3F	Reserved
DRR21	64	40	McBSP 1 Data Receive Register 2
DRR11	65	41	McBSP 1 Data Receive Register 1
DXR21	66	42	McBSP 1 Data Transmit Register 2
DXR11	67	43	McBSP 1 Data Transmit Register 1
—	68–71	44–47	Reserved
SPSA1	72	48	McBSP 1 Subbank Address Register
SPSD1	73	49	McBSP 1 Subbank Data Register
—	74–83	4A–53	Reserved
DMPREC	84	54	DMA Priority and Enable Control Register
DMSA	85	55	DMA Subbank Address Register

Figure 3.5(b).peripheral registers for the TMS320C54xx processors

Status registers (ST0, ST1):
ST0: Contains the status of flags (OVA, OVB, C, TC) produced by arithmetic operations & bit manipulations.
ST1: Contain the status of various conditions & modes. Bits of ST0&ST1registers can be set or clear with the SSBX & RSBX instructions.
PMST: Contains memory-setup status & control information.

Status register0 diagram:

ARP (15-13)	TC (12)	C (11)	OVA (10)	OVB (9)	DP (8-0)

Figure 3.6(a). ST0 diagram

ARP: Auxiliary register pointer. TC: Test/control flag.
C: Carry bit.
OVA: Overflow flag for accumulator A.
OVB: Overflow flag for accumulator B.
DP: Data-memory page pointer.

Status register1 diagram:

BRAF(15)	CPL (14)	XF (13)	HM (12)	INTM (11)	0 (10)	OVM (9)	SXM (8)	C16 (7)	FRCT(6)	CMPT(5)	ASM (4-0)

Figure 3.6(b). ST1 diagram

BRAF: **Block repeat active flag**
BRAF=0, the block repeat is deactivated.
BRAF=1, the block repeat is activated.

CPL: Compiler mode
CPL=0, the relative direct addressing mode using data page pointer is selected.
CPL=1, the relative direct addressing mode using stack pointer is selected.

HM: Hold mode, indicates whether the processor continues internal execution or acknowledge for external interface.

INTM: Interrupt mode, it globally masks or enables all interrupts.
INTM=0_all unmasked interrupts are enabled. INTM=1_all masked interrupts are disabled. 0: Always read as 0

OVM: Overflow mode.
OVM=1_the destination accumulator is set either the most positive value or the most negative value. OVM=0_the overflowed result is in destination accumulator.

SXM: Sign extension mode.
SXM=1_Data is sign extended

C16: Dual 16 bit/double-Precision arithmetic mode. C16=0_ALU operates in double-

Precision arithmetic mode. C16=1_ALU operates in dual 16-bit arithmetic mode.

FRCT: Fractional mode.
FRCT=1_the multiplier output is left-shifted by 1bit to compensate an extra sign bit.

CMPT: Compatibility mode.
CMPT=0_ ARP is not updated in the indirect addressing mode. CMPT=1_ARP is updated in the indirect addressing mode.

ASM: Accumulator Shift Mode.
5 bit field, & specifies the Shift value within -16 to 15 range.

<u>Processor Mode Status Register (PMST):</u>

IPTR(15-7)	MP/MC(6)	OVLY(5)	AVIS(4)	DROM(3)	CLKOFF(2)	SMUL(1)	SST(0)

Figure 3.6(c).PMST register diagram

INTR: Interrupt vector pointer, point to the 128-word program page where the interrupt vectors reside.
MP/MC: Microprocessor/Microcomputer mode, MP/MC=0, the on chip ROM is enabled. MP/MC=1, the on chip ROM is enabled.
OVLY: RAM OVERLAY, OVLY enables on chip dual access data RAM blocks to be mapped into program space.
AVIS: It enables/disables the internal program address to be visible at the address pins.
DROM: Data ROM, DROM enables on-chip ROM to be mapped into data space.
CLKOFF: CLOCKOUT off.
SMUL: Saturation on multiplication.
SST: Saturation on store.

3.4 Data Addressing Modes of TMS320C54X Processors:

Data addressing modes provide various ways to access operands to execute instructions and place results in the memory or the registers. The 54XX devices offer seven basic addressing modes

1. Immediate addressing.
2. Absolute addressing.
3. Accumulator addressing.
4. Direct addressing.
5. Indirect addressing.
6. Memory mapped addressing
7. Stack addressing.

Immediate addressing:

The instruction contains the specific value of the operand. The operand can be short (3,5,8 or 9 bit in length) or long (16 bits in length). The instruction syntax for short operands occupies one memory location,

Example: LD #20, DP.

RPT #0FFFFh.

Absolute Addressing:

The instruction contains a specified address in the operand.

1. Dmad addressing. MVDK Smem,dmad, MVDM dmad,MMR
2. Pmad addressing. MVDP Smem,pmad, MVPD pmem,Smad
3. PA addressing.

PORTR PA, Smem,

4.*(lk) addressing .

Accumulator Addressing:

Accumulator content is used as address to transfer data between Program and Data memory.

Ex: READA *AR2

Direct Addressing:

Base address + 7 bits of value contained in instruction = 16 bit address. A page of 128 locations can be accessed without change in DP or SP.Compiler mode bit (CPL) in ST1 register is used.

If CPL =0 selects DP CPL = 1 selects SP,

It should be remembered that when SP is used instead of DP, the effective address is computed by adding the 7-bit offset to SP.

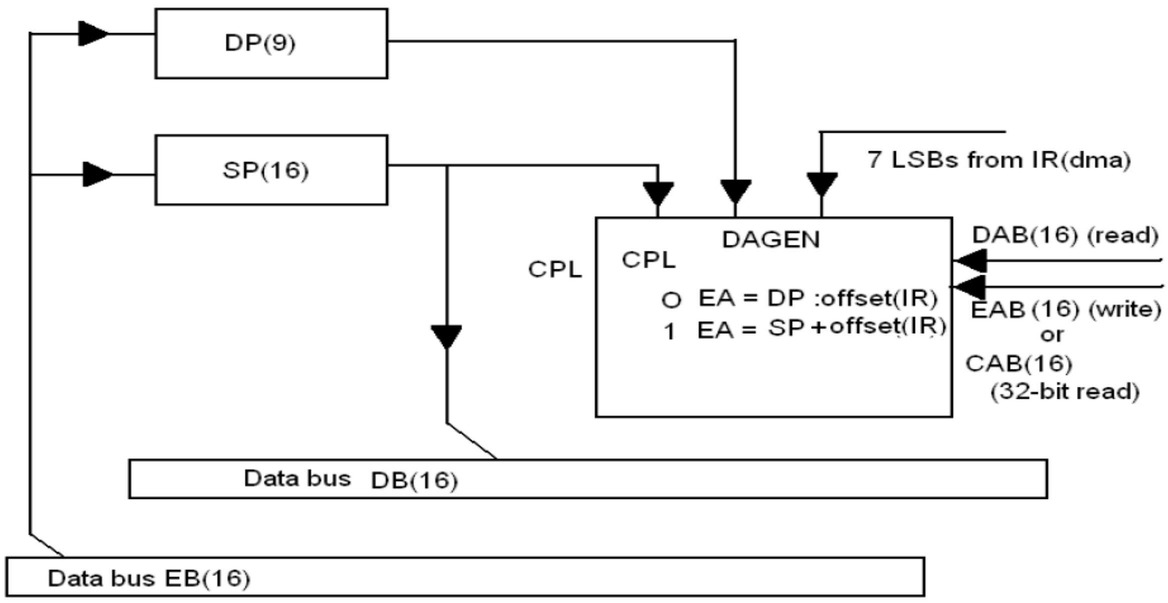


Figure 3.7 Block diagram of the direct addressing mode for TMS320C54xx Processors.

Indirect Addressing:

Data space is accessed by address present in an auxiliary register.

TMS320C54xx have 8, 16 bit auxiliary register (AR0 – AR 7). Two auxiliary register arithmetic units (ARAU0 & ARAU1) Used to access memory location in fixed step size. AR0 register is used for indexed and bit reverse addressing modes.

– operand addressing MOD _ type of indirect addressing ARF _ AR used for addressing ARP depends on (CMPT) bit in ST1

CMPT = 0, Standard mode, ARP set to zero

CMPT = 1, Compatibility mode, Particularly AR selected by ARP

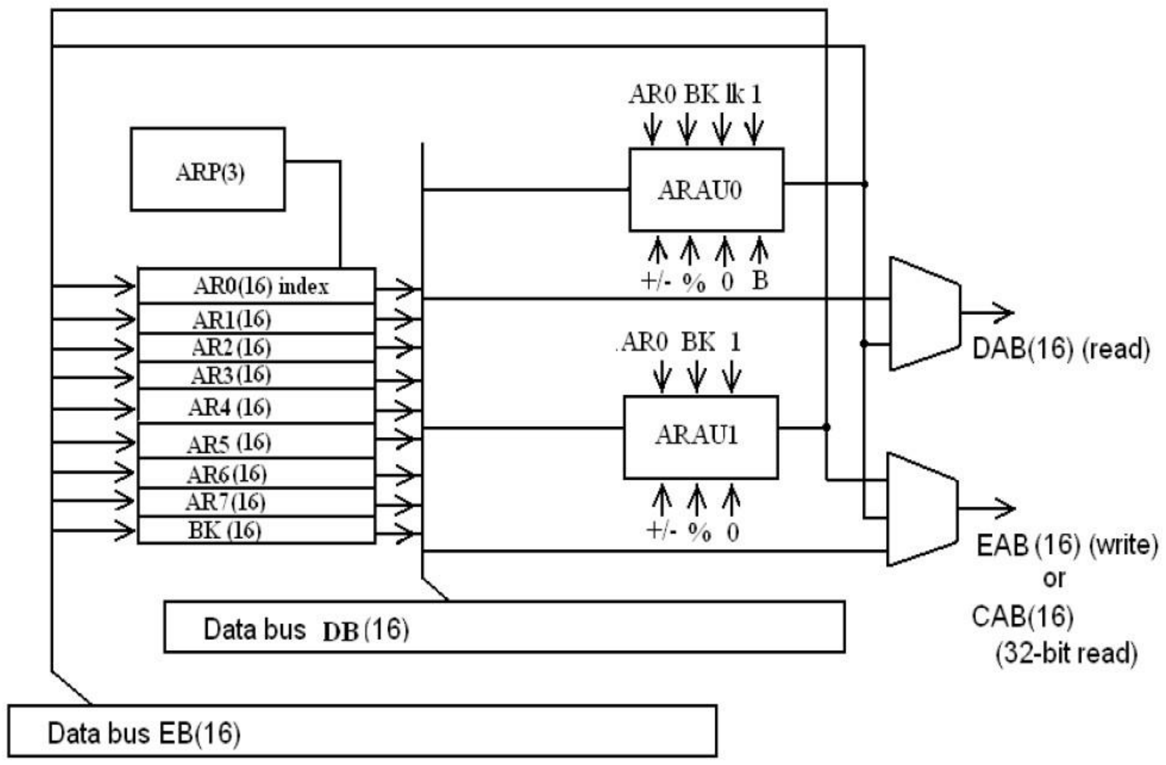


Figure 3.8 Block diagram of the indirect addressing mode for TMS320C54xx Processors.

Operand syntax	Function
*ARx	Addr = ARx;
*ARx -	Addr = ARx ; ARx = ARx -1
*ARx +	Addr = ARx; ARx = ARx +1
*+ARx	Addr = ARx+1; ARx = ARx +1
*ARx - 0B	Addr = ARx ; ARx = B(ARx – AR0)
*ARx - 0	Addr = Arx ; ARx = ARx – AR0
*ARx + 0	Addr = Arx ; ARx = ARx +AR0
*ARx + 0B	Addr = ARx ; ARx = B(ARx + AR0)
*ARx - %	Addr = ARx ; ARx = circ(ARx – 1)

Table 3.2 Indirect addressing options with a single data –memory operand.

Circular Addressing;

- Used in convolution, correlation and FIR filters.
- A circular buffer is a sliding window contains most recent data. Circular buffer

of size R must start on a N-bit boundary, where 2N > R .

- The circular buffer size register (BK): specifies the size of circular buffer.
- Effective base address (EFB): By zeroing the N LSBs of a user selected AR (ARx).
- End of buffer address (EOB) : By repalcing the N LSBs of ARx with the N LSBs of BK.
  If 0 _ index + step < BK ; index = index +step;
  else if index + step _ BK ; index = index
  + step - BK; else if index + step < 0;
  index + step + BK

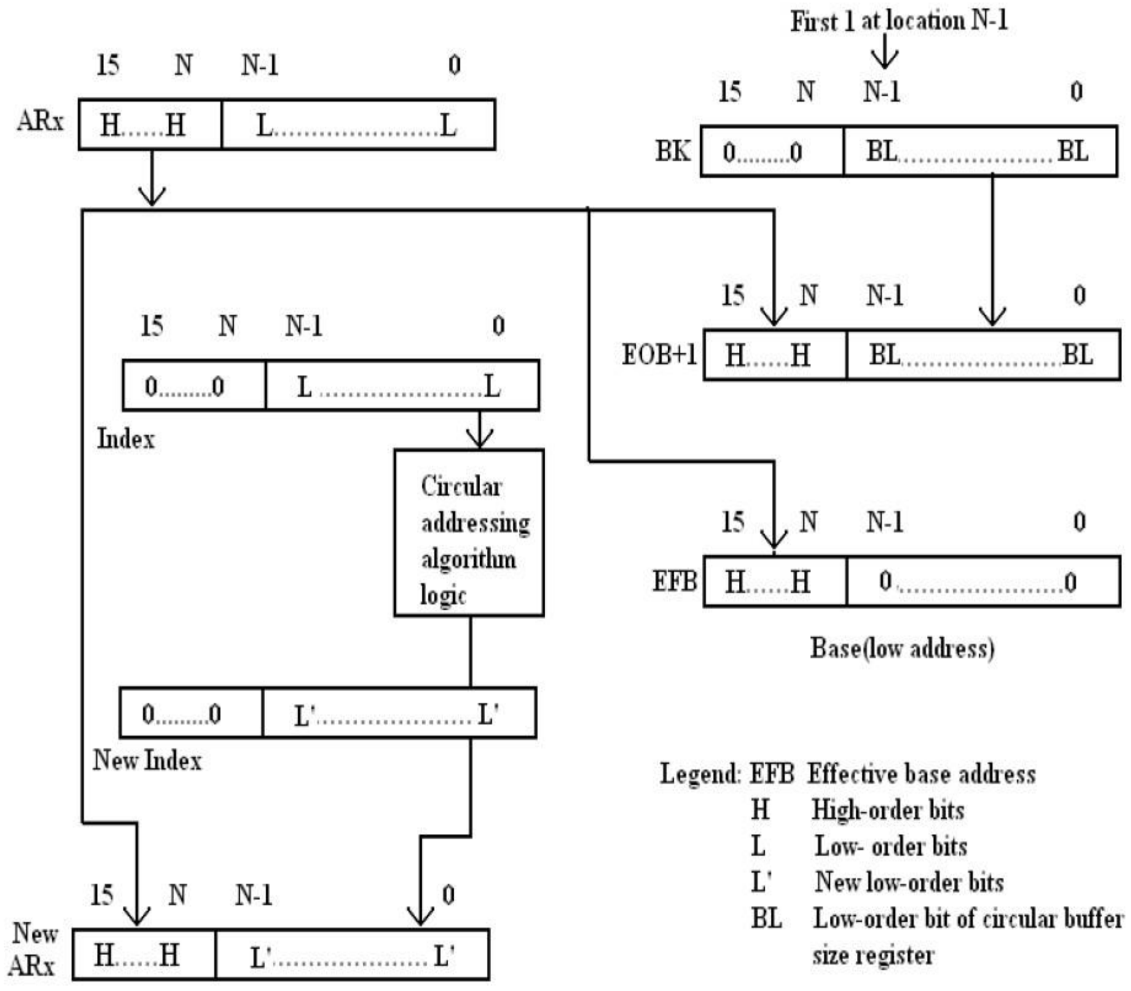


Figure 3.9 Block diagram of the circular addressing mode for TMS320C54xx Processors.

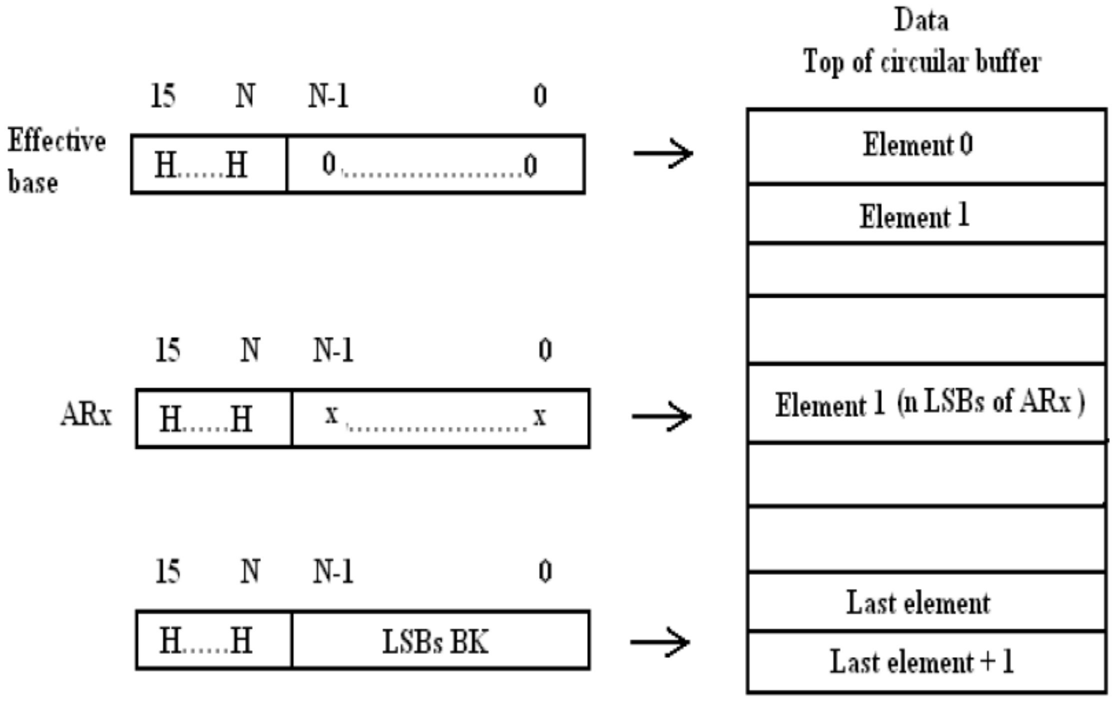


Figure 3.10 circular addressing mode implementation for TMS320C54xx Processors.

Bit-Reversed Addressing:

- Used for FFT algorithms.
- AR0 specifies one half of the size of the FFT.
- The value of AR0 = 2N-1: N = integer FFT size = 2N
- AR0 + AR (selected register) = bit reverse addressing.
- The carry bit propagating from left to right.

Dual-Operand Addressing:

Dual data-memory operand addressing is used for instruction that simultaneously perform two reads (32-bit read) or a single read (16-bit read) and a parallel store (16-bit store) indicated by two vertical bars, II. These instructions access operands using indirect addressing mode.

If in an instruction with a parallel store the source operand the destination operand point to the same location, the source is read before writing to the destination. Only 2 bits are available in the instruction code for selecting each auxiliary register in this mode. Thus, just four of the auxiliary registers, AR2-AR5, can be used, The ARAUs together with these registers, provide capability to access two operands in a single cycle. Figure 3.11 shows how an address is generated using dual data- memory operand addressing.

15 - 8	7 - 6	5 - 4	3 - 2	1 - 0
Opcode	Xmod	Xar	Ymod	Yar

Name	Function
Opcode	This field contains the operation code for the instruction
Xmod	Defined the type of indirect addressing mode used for accessing the Xmem operand
XAR	Xmem AR selection field defines the AR that contains the address of Xmem
Ymod	Defies the type of inderect addressing mode used for accessing the Ymem operand
Yar	Ymem AR selection field defines the AR that contains the address of Ymem

Table 3.3.Function of the different field in dual data memory operand addressing

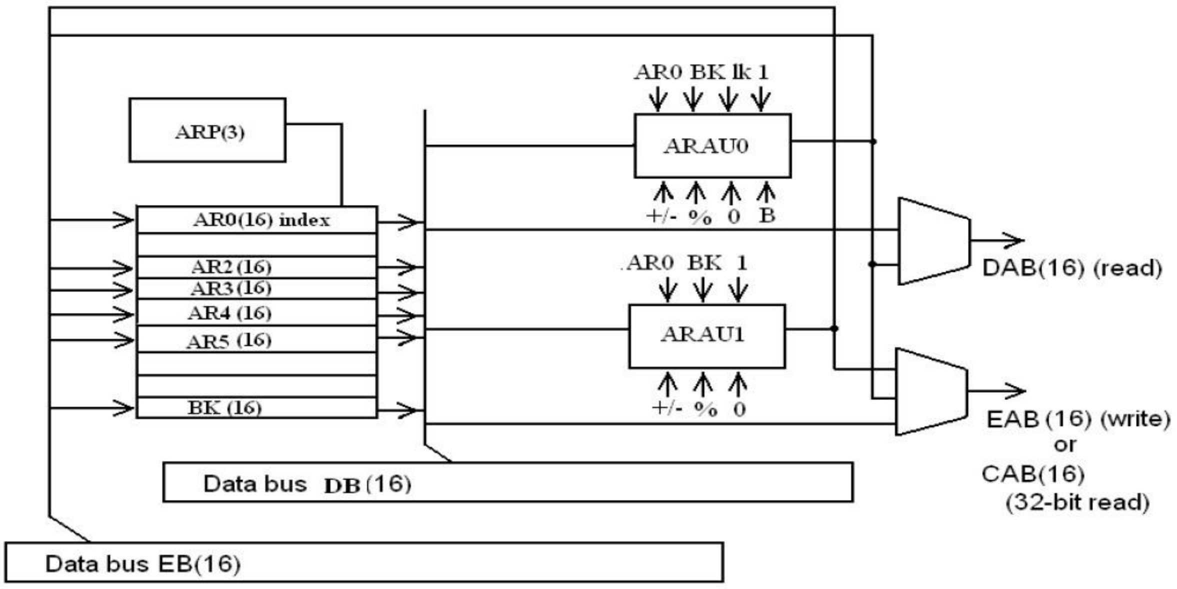


Figure 3.11 Block diagram of the Indirect addressing options with a dual data –memory operand.

Memory-Mapped Register Addressing:

- Used to modify the memory-mapped registers without affecting the current data page
- pointer (DP) or stack-pointer (SP)
    - Overhead for writing to a register is minimal
    - Works for direct and indirect addressing
    - Scratch –pad RAM located on data PAGE0 can be modified

- ➢ STM #x, DIRECT
- ➢ STM #tbl, AR1

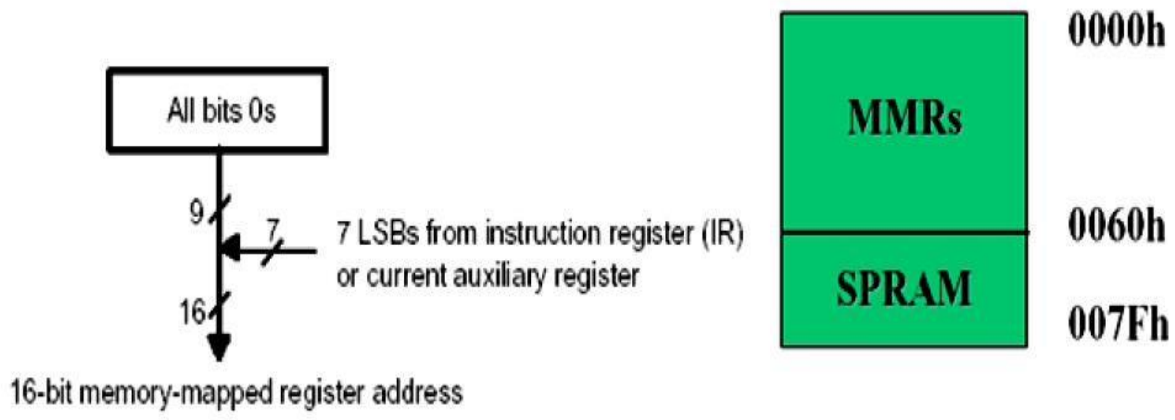

All bits 0s
9
7
7 LSBs from instruction register (IR) or current auxiliary register
16
16-bit memory-mapped register address
MMRs
SPRAM
0000h
0060h
007Fh

Figure 3.12.16 bit memory mapped register address generation.

3.4.7 Stack Addressing:

- Used to automatically store the program counter during interrupts and subroutines.
- Can be used to store additional items of context or to pass data values.
- Uses a 16-bit memory-mapped register, the stack pointer (SP).
- PSHD X2

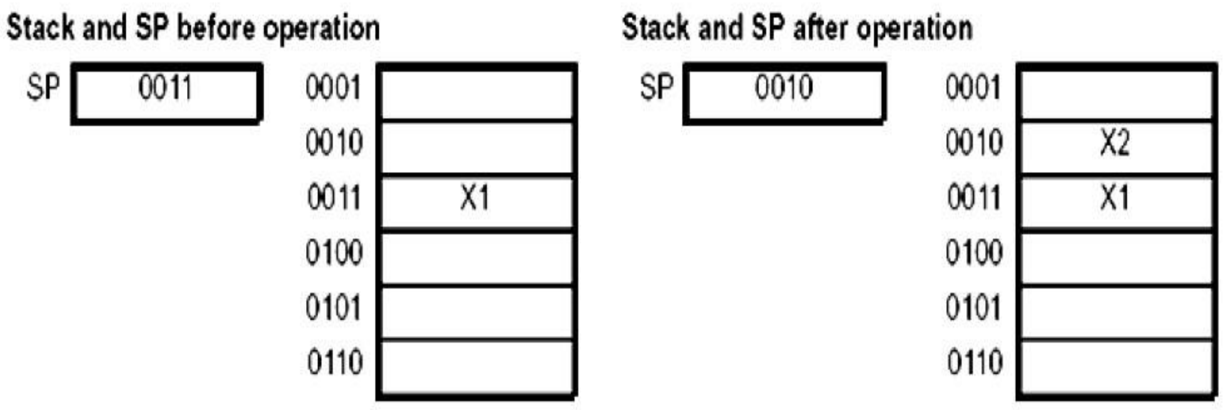


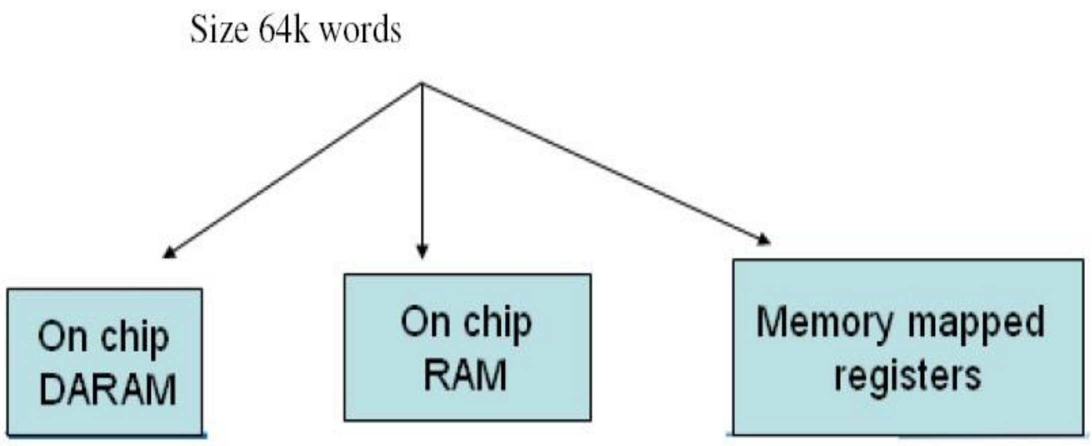
Figure 3.13. Values of stack &SP before and after operation.

Memory Space of TMS320C54xx Processors

- ➢ A total of 128k words extendable up to 8192k words.
- ➢ Total memory includes RAM, ROM, EPROM, EEPROM or Memory mapped peripherals.

Data memory: To store data required to run program and for external memory mapped resisters.

.



Program memory: To store program instructions &tables used in the execution of programs.

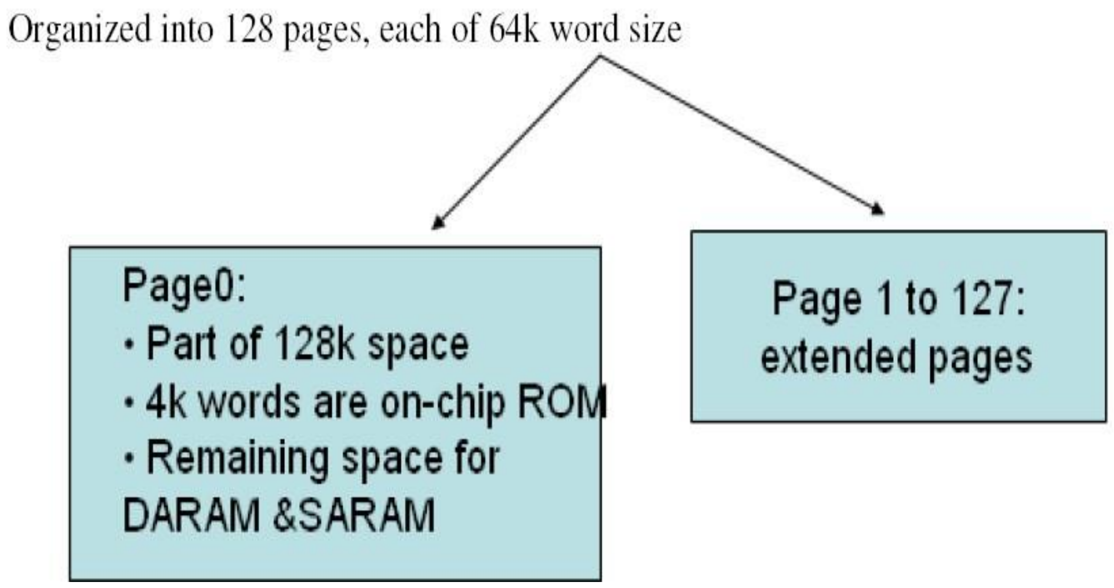

Organized into 128 pages, each of 64k word size
Page0:
• Part of 128k space
• 4k words are on-chip ROM
• Remaining space for DARAM &SARAM
Page 1 to 127: extended pages

Table 3.4.Function of different pin PMST register

PMST bit	Logic	On-chip memory configuration
MP/MC	0 1	ROM enabled ROM not available
OVLY	0 1	RAM in data space RAM in program space
DROM	0 1	ROM not in data space ROM in data space

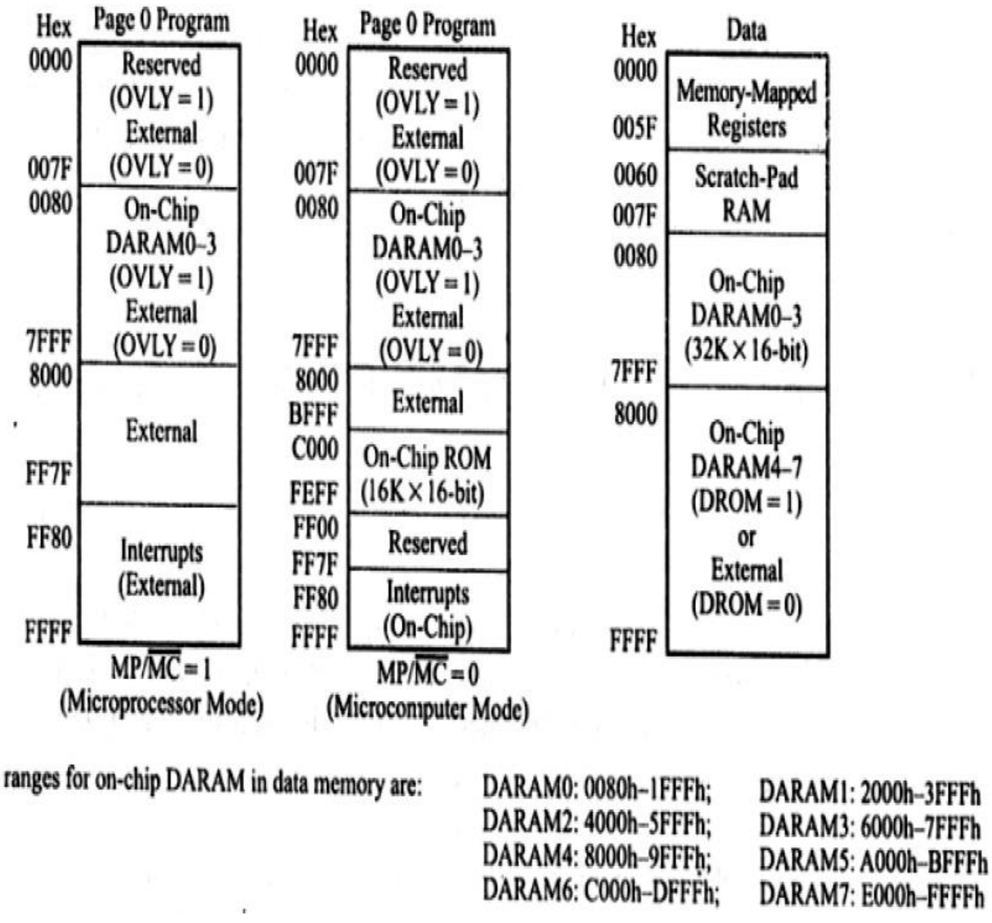

Hex
Page 0 Program
0000
Reserved (OVLY = 1) External (OVLY = 0)
007F
0080
On-Chip DARAM0–3 (OVLY = 1) External (OVLY = 0)
7FFF
8000
External
FF7F
FF80
Interrupts (External)
FFFF
MP/$\overline{MC}$ = 1 (Microprocessor Mode)
Hex
Page 0 Program
0000
Reserved (OVLY = 1) External (OVLY = 0)
007F
0080
On-Chip DARAM0–3 (OVLY = 1) External (OVLY = 0)
7FFF
8000
External
BFFF
C000
On-Chip ROM (16K × 16-bit)
FEFF
FF00
Reserved
FF7F
FF80
Interrupts (On-Chip)
FFFF
MP/$\overline{MC}$ = 0 (Microcomputer Mode)
Hex
Data
0000
Memory-Mapped Registers
005F
0060
Scratch-Pad RAM
007F
0080
On-Chip DARAM0–3 (32K × 16-bit)
7FFF
8000
On-Chip DARAM4–7 (DROM = 1) or External (DROM = 0)
FFFF
Address ranges for on-chip DARAM in data memory are:
DARAM0: 0080h–1FFFh;
DARAM1: 2000h–3FFFh
DARAM2: 4000h–5FFFh;
DARAM3: 6000h–7FFFh
DARAM4: 8000h–9FFFh;
DARAM5: A000h–BFFFh
DARAM6: C000h–DFFFh;
DARAM7: E000h–FFFFh

Figure 3.14 Memory map for the TMS320C5416 Processor.

Program Control

- It contains program counter (PC), the program counter related H/W, hard stack, repeat counters &status registers.
- PC addresses memory in several ways namely:
- Branch: The PC is loaded with the immediate value following the branch instruction
- Subroutine call: The PC is loaded with the immediate value following the call instruction
- Interrupt: The PC is loaded with the address of the appropriate interrupt vector.
- Instructions such as BACC, CALA, etc ;The PC is loaded with the contents of the accumulator low word
- End of a block repeat loop: The PC is loaded with the contents of the block repeat program address start register.
- Return: The PC is loaded from the top of the stack.

Problems:1

Assuming the current content of AR3 to be 200h, what will be its contents after each of the following TMS320C54xx addressing modes is used? Assume that the contents of AR0 are 20h.

a. *AR3+0
b. *AR3-0
c. *AR3+

d. *AR3
e. *AR3
f. *+AR3(40h) g. *+AR3 (-40h)

Solution:

a. AR3 ← AR3 + AR0; AR3 = 200h + 20h = 220h
b. AR3← AR3 - AR0; AR3 = 200h - 20h = 1E0h
c. AR3 ← AR3 + 1; AR3 = 200h + 1 = 201h
d. AR3 ← AR3 - 1; AR3 = 200h - 1 = 1FFh
e. AR3 is not modified.AR3 = 200h
f. AR3 ← AR3 + 40h; AR3 = 200 + 40h = 240h
g. AR3 ← AR3 - 40h; AR3 = 200 - 40h = 1C0

Problems:2

Assuming the current contents of AR3 to be 200h, what will be its contents after each of the following TMS320C54xx addressing modes is used? Assume that the contents of AR0 are 20h

a. *AR3 + 0B
b. *AR3 – 0B

Solution:

a. AR3 ← AR3 + AR0 with reverse carry propagation; AR3 = 200h + 20h (with reverse carry propagation) = 220h.
b.AR3 ← AR3 - AR0 with reverse carry propagation; AR3 = 200h - 20h (with reverse carry propagation) = 23Fh.

ASSIGNMENT NO:3(Recommended Questions)

1. Compare architectural features of TMS320C25 and DSP6000 fixed point digital signal processors.**(Dec.09-Jan.10, 6m)**

2. Write an explanatory note on direct addressing mode of TMS320C54XX processors. Give example.**(Dec.09-Jan.10, 6m)**

3. Describe the operation of the following instructions of TMS320C54XX processors.
   - **i)** MPY *AR2-,*AR4+0B
   - **ii)** MAC *ar5+,#1234h,A
   - **iii)** STH A,1,*AR2
   - **iv)** SSBX SXM **(Dec.09-Jan.10, 8m)**

4. With a block diagram explain the indirect addressing mode of TMS320C54XX processor using dual data memory operand. (**June.12, 6m)**

5. What is the function of an address generation unit explain with the help of block diagram (Dec.12, 6m)

6. Why circular buffers are required in DSP processor? How they are implemented? **(Dec.12, 2m)**

7. Explain the direct addressing mode of the TMS320C54XX processor with the help of a block diagram. **(Dec.12, 2m)**

8. Describe the multiplier/adder unit of TMS320c54xx processor with a neat block diagram. (May/June2010, 6m)

9. Describe any four data addressing modes of TMS320c54xx processor **(May/June2010, 8m)**

10. Assume that the current content of AR3 is 400h, what will be its contents after each of the following. Assume that the content of AR0 is 40h. **(May/June2010, 8m)**

11. Explain PMST register. **(May/June2011, 8m)**

12. With an example each, explain immediate, absolute, and direct addressing mode. **(May/June2011,**12**m)**

13. Explain the functioning of barrel shifter in TMS320C54XX processor. (**June.12, 6m)**

14. Explain sequential and other types of program control(**June.11, 7m)**

15. With an example each, explain immediate, absolute, and direct addressing mode.

16. Explain the functioning of barrel shifter in TMS320C54XX processor.

17. Explain sequential and other types of program control

18. Assume that the current content of AR3 is 400h, what will be its contents after each of the following. Assume that the content of AR0 is 40h.

19. Explain PMST register.

20. Compare architectural features of TMS320C25 and DSP6000 fixed point digital signal processor

INSTRUCTION AND PROGRAMMING

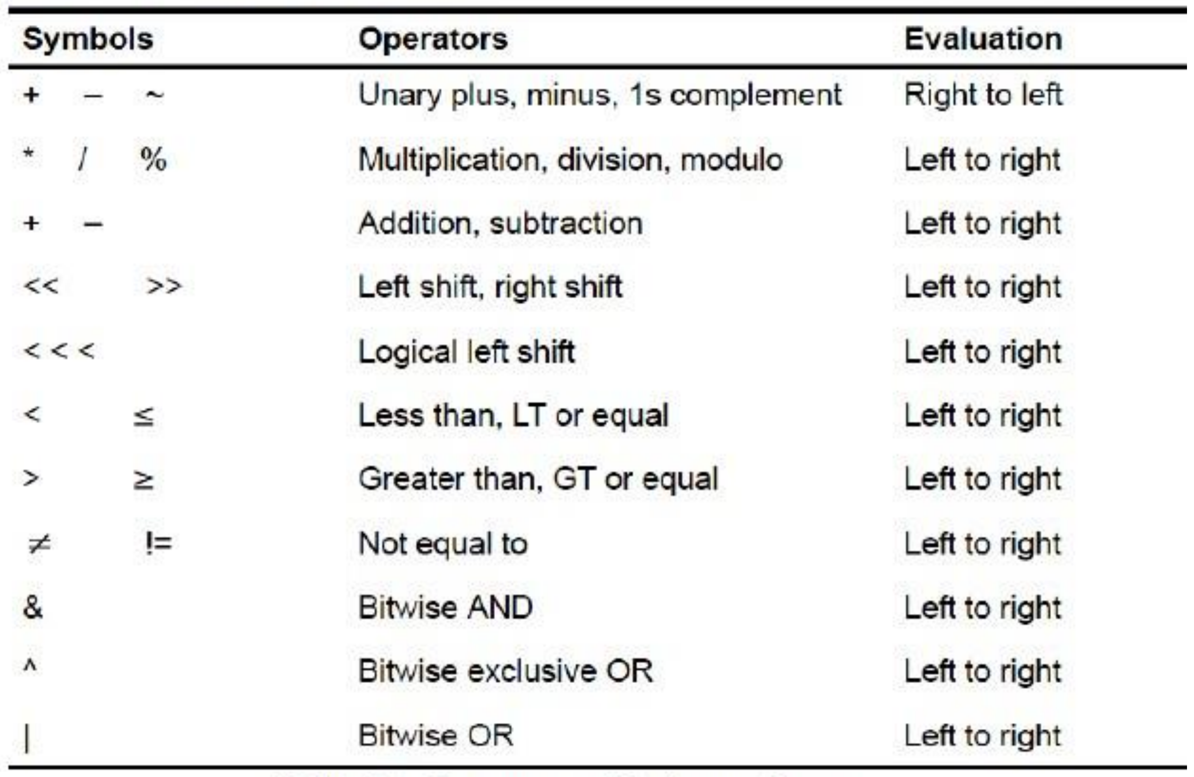
❖ LEARNING OBJECTIVES

- Detail Study of TMS320C54X & 54xx
- Instructions and Programming,
- On-Chip peripherals,
- Interrupts of TMS32OC54XX Processors,
- Pipeline Operation of TMS32OC54xx Processor

4.1 Assembly language instructions can be classified as:

- Arithmetic operations
- Load and store instructions.
- Logical operations
- Program-control operations

Operators Used in Instruction Set:



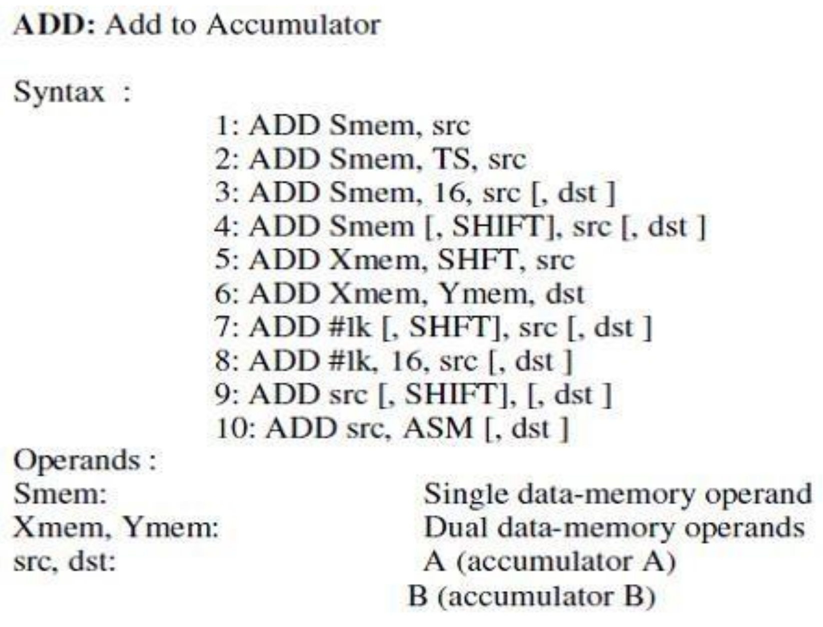
Symbols	Operators	Evaluation
+ – ~	Unary plus, minus, 1s complement	Right to left
* / %	Multiplication, division, modulo	Left to right
+ –	Addition, subtraction	Left to right
<< >>	Left shift, right shift	Left to right
< < <	Logical left shift	Left to right
< ≤	Less than, LT or equal	Left to right
> ≥	Greater than, GT or equal	Left to right
≠ !=	Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right

Table 4.1. Operator used in instruction set

4.1.1 Arithmetic Instructions:

Add Instructions:

Syntax	Expression
ADD Smem, src	src src = src + Smem
ADD Smem, TS, src	src = src + Smem << TS
ADD Smem, 16, src [, dst]	dst = src + Smem << 16
ADD Smem [, SHIFT], src [, dst]	dst = src + Smem << SHIFT
ADD Xmem, SHFT, src	src = src + Xmem <<□SHFT
ADD Xmem, Ymem, dst	dst = Xmem << 16 + Ymem << 16
ADD #lk [, SHFT], src [, dst]	dst = src + #lk << SHFT
ADD #lk, 16, src [, dst]	dst = src + #lk << 16
ADD src [, SHIFT] [, dst]	dst = dst + src << SHIFT
ADD src, ASM [, dst]	dst = dst + src << ASM
ADDC Smem, src	src = src + Smem + C
ADDM #lk, Smem	Smem = Smem + #lk



ADD: Add to Accumulator

Syntax :

1: ADD Smem, src
2: ADD Smem, TS, src
3: ADD Smem, 16, src [, dst]
4: ADD Smem [, SHIFT], src [, dst]
5: ADD Xmem, SHFT, src
6: ADD Xmem, Ymem, dst
7: ADD #lk [, SHFT], src [, dst]
8: ADD #lk, 16, src [, dst]
9: ADD src [, SHIFT], [, dst]
10: ADD src, ASM [, dst]

Operands :

Smem:	Single data-memory operand
Xmem, Ymem:	Dual data-memory operands
src, dst:	A (accumulator A)
	B (accumulator B)

$-32\,768 \le lk \le 32\,767$
$-16 \le SHIFT \le 15$
$0 \le SHFT \le 15$

Execution :

1: (Smem) + (src) → src
2: (Smem) << (TS) + (src)→ src
3: (Smem) << 16 + (src) → dst
4: (Smem) [<< SHIFT] + (src) → dst

SUB: Subtract From Accumulator

Syntax

1: **SUB** *Smem*, *src*
2: **SUB** *Smem*, **TS**, *src*
3: **SUB** *Smem*, **16**, *src* [, *dst*]
4: **SUB** *Smem* [, *SHIFT*], *src* [, *dst*]
5: **SUB** *Xmem*, *SHFT*, *src*
6: **SUB** *Xmem*, *Ymem*, *dst*
7: **SUB** *#lk* [, *SHFT*], *src* [, *dst*]
8: **SUB** *#lk*, **16**, *src* [, *dst*]
9: **SUB** *src* [, *SHIFT*], [, *dst*]
10: **SUB** *src*, **ASM** [, *dst*]

Operands

src, dst:	A (accumulator A)
	B (accumulator B)
Smem:	Single data-memory operand
Xmem, Ymem:	Dual data-memory operan

$-32\,768 \le lk \le 32\,767$
$0 \le SHFT \le 15$
$-16 \le SHIFT \le 15$

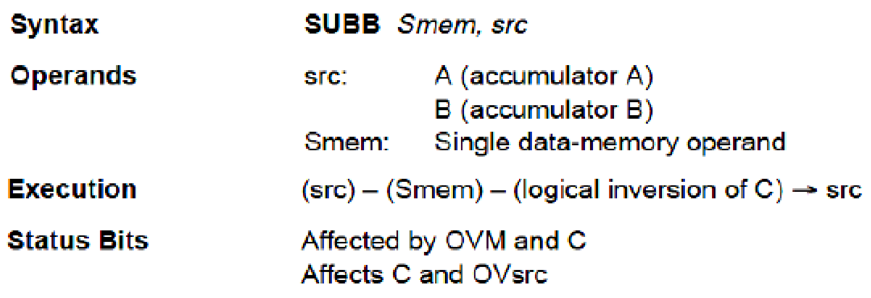
Execution

1: (src) – (Smem) → src
2: (src) – (Smem) << TS → src
3: (src) – (Smem) << 16 → dst
4: (src) – (Smem) << SHIFT → dst
5: (src) – (Xmem) << SHFT → src
6: (Xmem) << 16 – (Ymem) << 16 → dst
7: (src) – lk << SHFT → dst
8: (src) – lk << 16 → dst
9: (dst) – (src) << SHIFT → dst
10: (dst) – (src) << ASM → dst

Status Bits

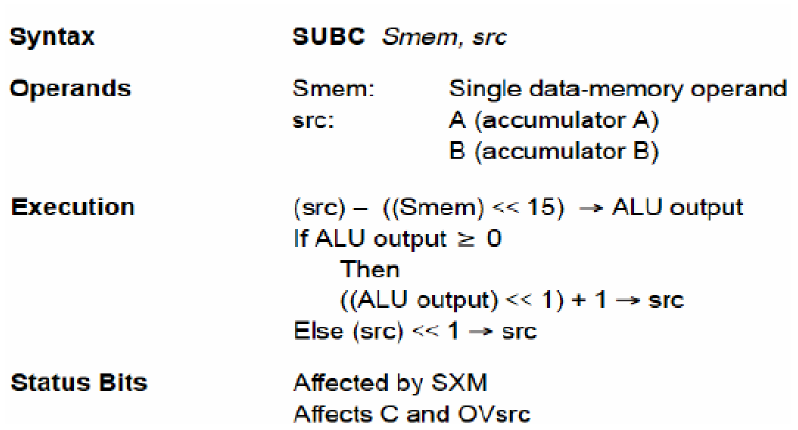
Affected by SXM and OVM
Affects C and OVdst (or OVsrc, if dst = src)

SUBB: Subtract From Accumulator with Borrow



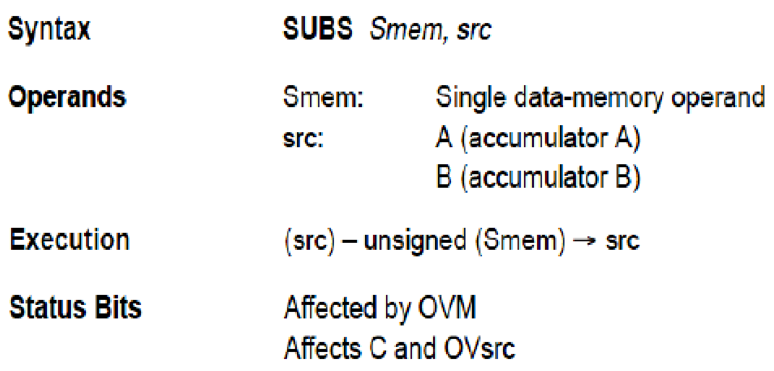
Syntax	**SUBB** *Smem, src*	
Operands	src:	A (accumulator A) B (accumulator B)
	Smem:	Single data-memory operand
Execution	(src) – (Smem) – (logical inversion of C) → src	
Status Bits	Affected by OVM and C Affects C and OVsrc	

SUBC: Subtract Conditionally



Syntax	**SUBC** *Smem, src*	
Operands	Smem: src:	Single data-memory operand A (accumulator A) B (accumulator B)
Execution	(src) – ((Smem) << 15) → ALU output If ALU output ≥ 0 Then ((ALU output) << 1) + 1 → src Else (src) << 1 → src	
Status Bits	Affected by SXM Affects C and OVsrc	

SUBS: Subtract with accumulator with sign extension suppressed



Syntax	**SUBS** *Smem, src*	
Operands	Smem: src:	Single data-memory operand A (accumulator A) B (accumulator B)
Execution	(src) – unsigned (Smem) → src	
Status Bits	Affected by OVM Affects C and OVsrc	

MPY: Multiply With/Without Rounding

Syntax	1: **MPY[R]** *Smem, dst* 2: **MPY** *Xmem, Ymem, dst* 3: **MPY** *Smem, #lk, dst* 4: **MPY** *#lk, dst*
Operands	Smem: Single data-memory operand Xmem, Ymem: Dual data-memory operands dst: A (accumulator A) B (accumulator B) $-32\,768 \le lk \le 32\,767$
Execution	1: $(T) \times (Smem) \rightarrow dst$ 2: $(Xmem) \times (Ymem) \rightarrow dst$ $(Xmem) \rightarrow T$ 3: $(Smem) \times lk \rightarrow dst$ $(Smem) \rightarrow T$ 4: $(T) \times lk \rightarrow dst$
Status Bits	Affected by FRCT and OVM Affects OVdst

MPYA: Multiply by Accumulator A

Syntax	1: **MPYA** *Smem* 2: **MPYA** *dst*
Operands	Smem: Single data-memory operand dst: A (accumulator A) B (accumulator B)
Execution	1: $(Smem) \times (A(32\text{–}16)) \rightarrow B$ $(Smem) \rightarrow T$ 2: $(T) \times (A(32\text{–}16)) \rightarrow dst$
Status Bits	Affected by FRCT and OVM Affects OVdst (OVB in syntax 1)

MPYU: Multiply Unsigned

Syntax	**MPYU** *Smem, dst*
Operands	Smem: Single data-memory operand dst: A (accumulator A) B (accumulator B)
Execution	unsigned(T) × unsigned(Smem) → dst
Status Bits	Affected by FRCT and OVM Affects OVdst

SQUR: Square

Syntax	1: **SQUR** *Smem, dst* 2: **SQUR A,** *dst*
Operands	Smem: Single data-memory operand dst: A (accumulator A) B (accumulator B)
Execution	1: (Smem) → T (Smem) × (Smem) → dst 2: (A(32–16)) × (A(32–16)) → dst
Status Bits	Affected by OVM and FRCT Affects OVsrc

SQURA: Square and Accumulate

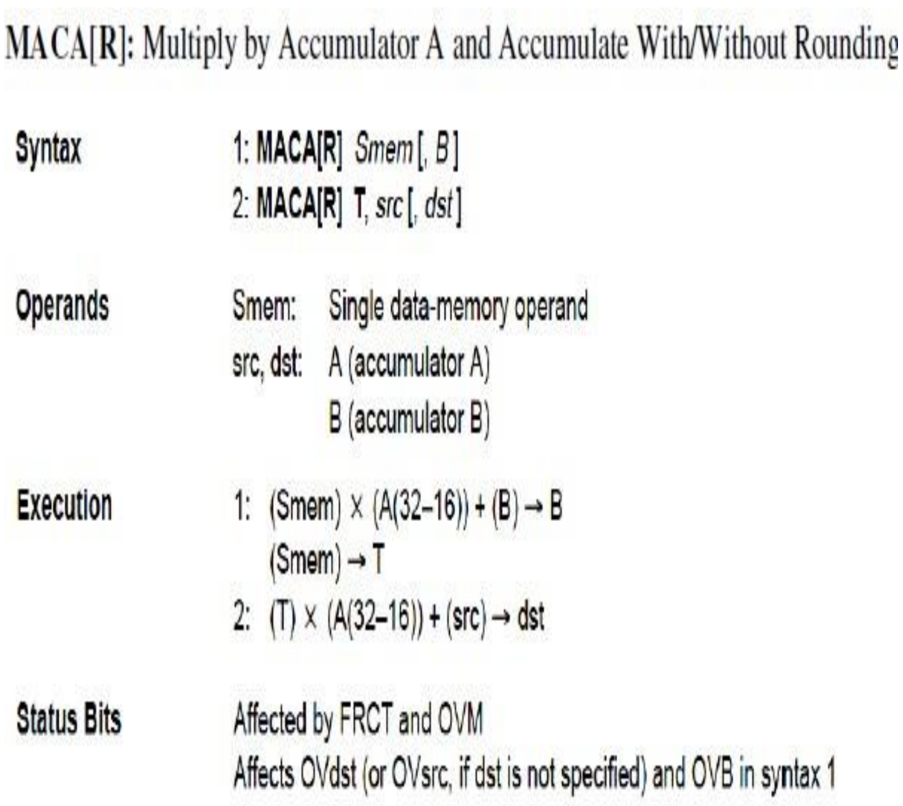
Syntax	**SQURA** *Smem, src*
Operands	Smem: Single data-memory operand src: A (accumulator A) B (accumulator B)
Execution	(Smem) → T (Smem) × (Smem) + (src) → src
Status Bits	Affected by OVM and FRCT Affects OVsrc

SQURS: Square and Subtract

Syntax	**SQURS** *Smem, src*
Operands	Smem: Single data-memory operand src: A (accumulator A) B (accumulator B)
Execution	(Smem) → T (src) – (Smem) × (Smem) → src
Status Bits	Affected by OVM and FRCT Affects OVsrc

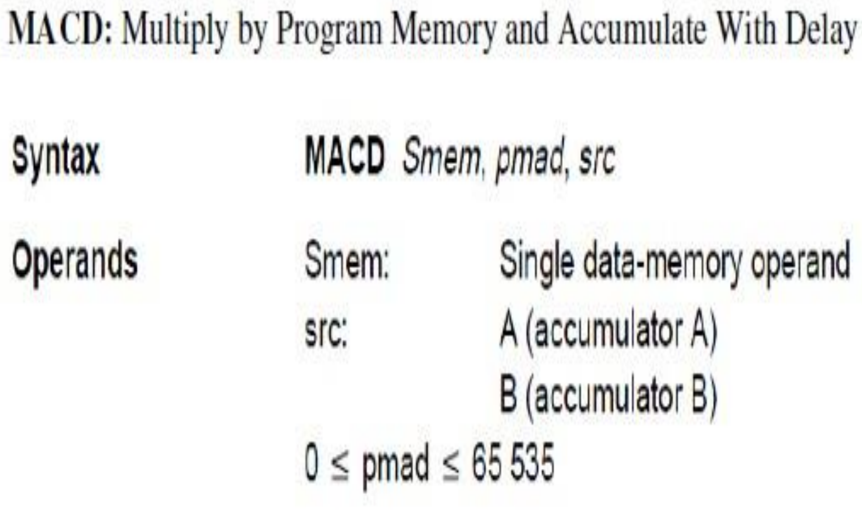
MAC[R]: Multiply Accumulate With/Without Rounding

Syntax	1: **MAC[R]** *Smem, src* 2: **MAC[R]** *Xmem, Ymem, src* [, *dst*] 3: **MAC** *#lk, src* [, *dst*] 4: **MAC** *Smem, #lk, src* [, *dst*]
Operands	Smem: Single data-memory operands Xmem, Ymem: Dual data-memory operands src, dst: A (accumulator A) B (accumulator B) $-32\,768 \leq lk \leq 32\,767$
Execution	1: (Smem) × (T) + (src) → src 2: (Xmem) × (Ymem) + (src) → dst (Xmem) → T 3: (T) × lk + (src) → dst 4: (Smem) × lk + (src) → dst (Smem) → T
Status Bits	Affected by FRCT and OVM Affects OVdst (or OVsrc, if dst is not specified)



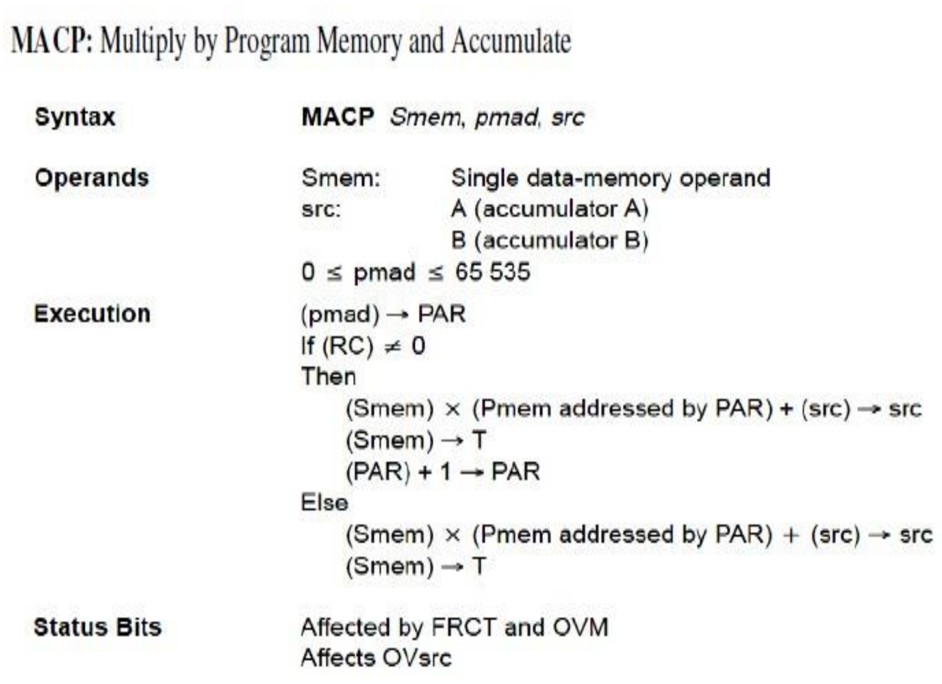
MACA[R]: Multiply by Accumulator A and Accumulate With/Without Rounding

Syntax	1: **MACA[R]** *Smem* [, *B*] 2: **MACA[R]** **T**, *src* [, *dst*]
Operands	Smem: Single data-memory operand src, dst: A (accumulator A) B (accumulator B)
Execution	1: (Smem) × (A(32–16)) + (B) → B (Smem) → T 2: (T) × (A(32–16)) + (src) → dst
Status Bits	Affected by FRCT and OVM Affects OVdst (or OVsrc, if dst is not specified) and OVB in syntax 1



MACD: Multiply by Program Memory and Accumulate With Delay

Syntax	**MACD** *Smem, pmad, src*
Operands	Smem: Single data-memory operand src: A (accumulator A) B (accumulator B) 0 ≤ pmad ≤ 65 535



MACP: Multiply by Program Memory and Accumulate

Syntax **MACP** *Smem, pmad, src*

Operands

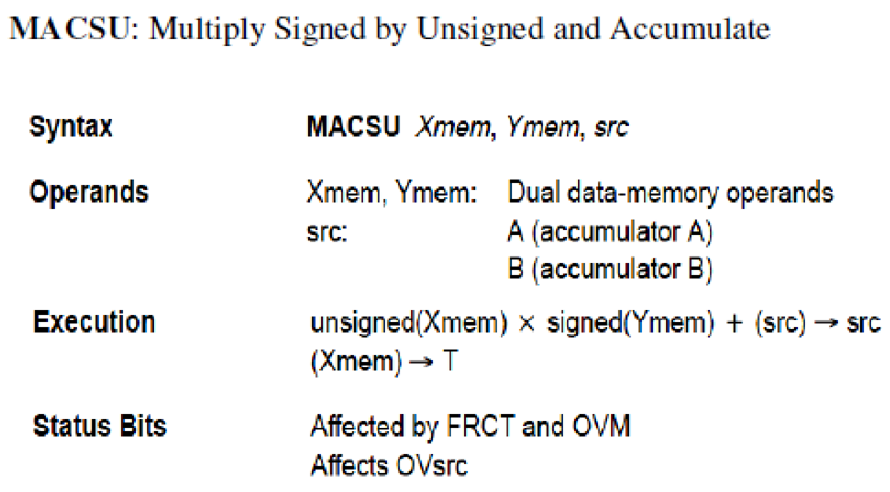
Smem:	Single data-memory operand
src:	A (accumulator A)
	B (accumulator B)

$0 \leq \text{pmad} \leq 65\,535$

Execution

```
(pmad) → PAR
If (RC) ≠ 0
Then
    (Smem) × (Pmem addressed by PAR) + (src) → src
    (Smem) → T
    (PAR) + 1 → PAR
Else
    (Smem) × (Pmem addressed by PAR) + (src) → src
    (Smem) → T
```

Status Bits Affected by FRCT and OVM
Affects OVsrc



MACSU: Multiply Signed by Unsigned and Accumulate

Syntax **MACSU** *Xmem, Ymem, src*

Operands

Xmem, Ymem:	Dual data-memory operands
src:	A (accumulator A)
	B (accumulator B)

Execution

```
unsigned(Xmem) × signed(Ymem) + (src) → src
(Xmem) → T
```

Status Bits Affected by FRCT and OVM
Affects OVsrc

MACSU: Multiply Signed by Unsigned and Accumulate

Syntax	**MACSU** *Xmem*, *Ymem*, *src*
Operands	Xmem, Ymem: Dual data-memory operands src: A (accumulator A) B (accumulator B)
Execution	unsigned(Xmem) × signed(Ymem) + (src) → src (Xmem) → T
Status Bits	Affected by FRCT and OVM Affects OVsrc

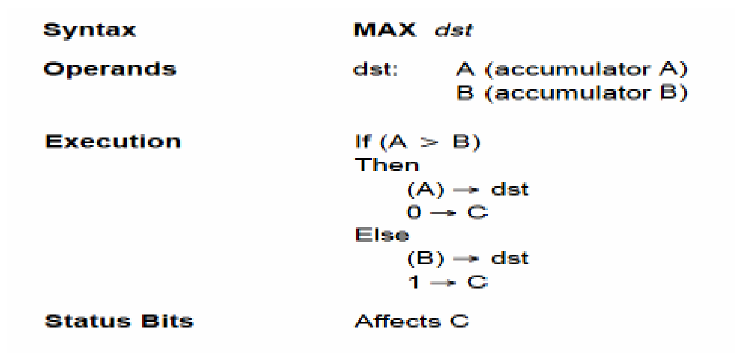
MAS[R] : Multiply and Subtract With/Without Rounding

Syntax	1: **MAS[R]** *Smem*, *src* 2: **MAS[R]** *Xmem*, *Ymem*, *src* [, *dst*]
Operands	Smem: Single data-memory operand Xmem, Ymem: Dual data-memory operands src, dst: A (accumulator A) B (accumulator B)

MASA[R] :Multiply by Accumulator A and Subtract With/Without Rounding

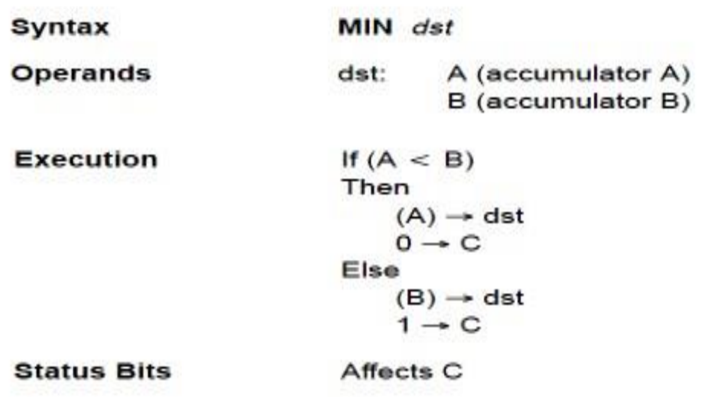
Syntax	1: **MASA** *Smem* [, *B*] 2: **MASA[R]** T, *src* [, *dst*]
Operands	Smem: Single data-memory operand src, dst: A (accumulator A) B (accumulator B)
Execution	1: (B) − (Smem) × (A(32–16)) → B (Smem) → T 2: (src) − (T) × (A(32–16)) → dst
Status Bits	Affected by FRCT and OVM Affects OVdst (or OVsrc, if dst is not specified) and OVB in syntax 1

MAX :Accumulator Maximum



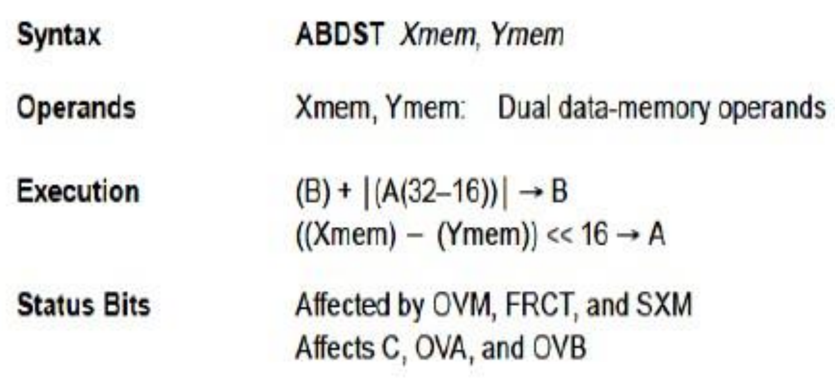
Syntax	**MAX** *dst*
Operands	dst: A (accumulator A) B (accumulator B)
Execution	If (A > B) Then (A) → dst 0 → C Else (B) → dst 1 → C
Status Bits	Affects C

MIN : Accumulator Minimum



Syntax	**MIN** *dst*
Operands	dst: A (accumulator A) B (accumulator B)
Execution	If (A < B) Then (A) → dst 0 → C Else (B) → dst 1 → C
Status Bits	Affects C

ABDST: Absolute Distance



Syntax	**ABDST** *Xmem, Ymem*
Operands	Xmem, Ymem: Dual data-memory operands
Execution	$(B) + \|(A(32–16))\| \rightarrow B$ $((Xmem) - (Ymem)) << 16 \rightarrow A$
Status Bits	Affected by OVM, FRCT, and SXM Affects C, OVA, and OVB

ABS: Absolute Value of Accumulator

```
ABS A
```

	Before Instruction		After Instruction
A	03 1234 5678	A	00 7FFF FFFF
OVM	1	OVM	1

CMPL :Complement Accumulator

Syntax	**CMPL** *src* [, *dst*]
Operands	src, dst: A (accumulator A) B (accumulator B)
Execution	($\overline{src}$) → dst
Status Bits	None

CMPM :Compare Memory With Long Immediate

Syntax	**CMPM** *Smem, #lk*
Operands	Smem: Single data-memory operan –32 768 ≤ lk ≤ 32 767
Execution	If (Smem) = lk Then 1 → TC Else 0 → TC
Status Bits	Affects TC

CMPS :Compare, Select and Store Maximum

Syntax **CMPS** *src, Smem*

Operands

src: A (accumulator A)
B (accumulator B)
Smem: Single data-memory operand

Execution

```
If ((src(31–16)) > (src(15–0)))
Then
      (src(31–16)) → Smem
      (TRN) << 1 → TRN
      0 → TRN(0)
      0 → TC
Else
      (src(15–0)) → Smem
      (TRN) << 1 → TRN
      1 → TRN(0)
      1 → TC
```

Status Bits Affects TC

EXP: Accumulator Exponent

Syntax	**EXP** *src*
Operands	src: A (accumulator A) B (accumulator B)
Execution	If (src) = 0 Then $0 \rightarrow T$ Else (Number of leading bits of src) $- 8 \rightarrow T$
Status Bits	None

SAT :Saturate Accumulator

Operands	src: A (accumulator A) B (accumulator B)
Execution	Saturate (src) $\rightarrow$ src
Status Bits	Affects OVsrc

NORM: Normalization

Syntax	**NORM** *src* [, *dst*]
Operands	src, dst : A (accumulator A) B (accumulator B)
Execution	(src) << TS $\rightarrow$ dst
Status Bits	Affected by SXM and OVM Affects OVdst (or OVsrc, when dst = src)

4.1.2 Logical Operations:

AND: AND With Accumulator

Syntax	1: **AND** *Smem*, *src* 2: **AND** #*lk* [, *SHFT*], *src* [, *dst*] 3: **AND** #*lk*, **16**, *src* [, *dst*] 4: **AND** *src* [, *SHIFT*], [, *dst*]
Operands	Smem: Single data-memory operand src: A (accumulator A) B (accumulator B) $-16 \leq$ SHIFT ≤ 15 $0 \leq$ SHFT ≤ 15 $0 \leq$ lk $\leq 65\,535$
Execution	1: (Smem) AND (src) $\rightarrow$ src 2: lk << SHFT AND (src)$\rightarrow$ dst 3: lk << 16 AND (src)$\rightarrow$ dst 4: (dst) AND (src) << SHIFT $\rightarrow$ dst
Status Bits	None

ANDM:AND Memory With Long Immediate

Syntax	**ANDM** #*lk*, *Smem*
Operands	Smem: Single data-memory operand $0 \leq$ lk $\leq 65\,535$
Execution	lk AND (Smem) $\rightarrow$ Smem
Status Bits	None

OR: OR with Accumulator

Syntax	1: **OR** *Smem, src* 2: **OR** #*lk* [, *SHFT*], *src* [, *dst*] 3: **OR** #*lk*, **16**, *src* [, *dst*] 4: **OR** *src* [, *SHIFT*], [, *dst*]
Operands	src, dst : A (accumulator A) B (accumulator B) Smem : Single data-memory operand $0 \le$ SHFT ≤ 15 $-16 \le$ SHIFT ≤ 15 $0 \le$ lk $\le 65\,535$
Execution	1: (Smem) OR (src(15–0)) → src src(39–16) unchanged 2: lk << SHFT OR (src) → dst 3: lk << 16 OR (src) → dst 4: (src or [dst]) OR (src) << SHIFT → dst
Status Bits	None

ORM: OR Memory With Constant

Syntax	**ORM** #*lk, Smem*
Operands	Smem: Single data-memory operand $0 \le$ lk $\le 65\,535$
Execution	lk OR (Smem) → Smem
Status Bits	None

XOR: Exclusive OR With Accumulator

Syntax	1: **XOR** *Smem, src* 2: **XOR** #*lk* [, *SHFT*], *src* [, *dst*] 3: **XOR** #*lk*, **16**, *src* [, *dst*] 4: **XOR** *src* [, *SHIFT*] [, *dst*]
Operands	src, dst: A (accumulator A) B (accumulator B) Smem: Single data-memory operand $0 \le$ SHFT ≤ 15 $-16 \le$ SHIFT ≤ 15 $0 \le$ lk $\le 65\,535$

Execution	1: (Smem) XOR (src) → src 2: lk << SHFT XOR (src) → dst 3: lk << 16 XOR (src) → dst 4: (src) << SHIFT XOR (dst) → dst
Status Bits	None

XORM: Exclusive OR Memory with Constant

Syntax	**XORM** #*lk*, *Smem*
Operands	Smem: Single data-memory operand $0 \le lk \le 65\,535$
Execution	lk XOR (Smem) → Smem
Status Bits	None

ROL: Rotate Accumulator Left

Syntax	**ROL** *src*
Operands	src : A (accumulator A) B (accumulator B)
Execution	(C) → src(0) (src(30–0)) → src(31–1) (src(31)) → C 0 → src(39–32)
Status Bits	Affected by C Affects C

ROLTC: Rotate Accumulator Left Using TC

Syntax	**ROLTC** *src*
Operands	src: A (accumulator A) B (accumulator B)
Execution	(TC) → src(0) (src(30–0)) → src(31–1) (src(31)) → C 0 → src(39–32)
Status Bits	Affects C Affected by TC

ROR: Rotate Accumulator Right

Syntax ROR *src*

Operands src: A (accumulator A)
B (accumulator B)

Execution

```
(C) → src(31)
(src(31–1)) → src(30–0)
(src(0)) → C
0 → src(39–32)
```

Status Bits Affects C
Affected by C

SFTA: Shift Accumulator Arithmetically

Syntax SFTA *src*, *SHIFT* [, *dst*]

Operands src, dst A (accumulator A)
B (accumulator B)
$-16 \le \text{SHIFT} \le 15$

Execution

```
If SHIFT < 0
Then
    (src((–SHIFT) – 1)) → C
    (src(39–0)) << SHIFT → dst
    If SXM = 1
    Then
        (src(39)) → dst(39–(39 + (SHIFT + 1))) [or src(39–(39 + (SHIFT + 1))),
        if dst is not specified]
    Else
        0 → dst(39–(39 + (SHIFT + 1))) [or src(39–(39 + (SHIFT + 1))),
        if dst is not specified]
Else
    (src(39 – SHIFT)) → C
    (src) << SHIFT → dst
    0 → dst((SHIFT – 1)–0) [or src((SHIFT – 1)–0), if dst is not specified]
```

Status Bits Affected by SXM and OVM
Affects C and OVdst (or OVsrc, if dst = src)

Syntax SFTC *src*

Operands src: A (accumulator A)
B (accumulator B)

Execution

```
If (src) = 0
Then
    1 → TC
Else
    If (src(31)) XOR (src(30)) = 0
    Then (two significant sign bits)
        0 → TC
        (src) << 1 → src
    Else (only one sign bit)
        1 → TC
```

Status Bits Affects TC

SFTC: Shift Accumulator Conditionally

Syntax **SFTL** *src*, *SHIFT* [, *dst*]

Operands src, dst: A (accumulator A)
B (accumulator B)
$-16 \le \text{SHIFT} \le 15$

Execution

```
If SHIFT < 0
Then
    src((−SHIFT) − 1) → C
    src(31−0) << SHIFT → dst
    0 → dst(39−(31 + (SHIFT + 1)))
If SHIFT = 0
Then
    0 → C
Else
    src(31 − (SHIFT − 1)) → C
    src((31 − SHIFT)−0) << SHIFT → dst
    0 → dst((SHIFT − 1)−0) [or src((SHIFT − 1)−0), if dst is not specified]
    0 → dst(39−32) [or src(39−32), if dst is not specified]
```

Status Bits Affects C

SFTL: Shift Accumulator Logically

BIT : Test Bit

Syntax	**BIT** *Xmem, BITC*
Operands	Xmem: Dual data-memory operand $0 \leq$ BITC ≤ 15
Execution	(Xmem(15 – BITC)) → TC
Status Bits	Affects TC

BITF: Test Bit Field Specified by Immediate Value

Syntax	**BITF** *Smem, #lk*
Operands	Smem: Single data-memory operand $0 \leq$ lk $\leq 65\,535$
Execution	If ((Smem) AND lk) = 0 Then 0 → TC Else 1 → TC
Status Bits	Affects TC

BITT : Test Bit Specified by T

Example `BITT *AR7+0`

	Before Instruction		After Instruction
T	C	T	C
TC	0	TC	1
AR0	0008	AR0	0008
AR7	0100	AR7	0108
Data Memory			
0100h	0008	0100h	0008

4.1.3.Load and Store operations:

LD: Load Accumulator with Shift

Syntax

1: **LD** *Smem, dst*
2: **LD** *Smem*, **TS**, *dst*
3: **LD** *Smem*, **16**, *dst*
4: **LD** *Smem* [, *SHIFT*], *dst*
5: **LD** *Xmem, SHFT, dst*
6: **LD** *#K, dst*
7: **LD** *#lk* [, *SHFT*], *dst*
8: **LD** *#lk*, **16**, *dst*
9: **LD** *src*, **ASM** [, *dst*]
10: **LD** *src* [, *SHIFT*], *dst*

For additional load instructions, see *Load T/DP/ASM/ARP* on page 4-70.

Operands

Smem:	Single data-memory operand
Xmem:	Dual data-memory operand
src, dst:	A (accumulator A)
	B (accumulator B)

$0 \leq K \leq 255$
$-32\,768 \leq lk \leq 32\,767$
$-16 \leq SHIFT \leq 15$
$0 \leq SHFT \leq 15$

Execution

1: (Smem) $\rightarrow$ dst
2: (Smem) << TS $\rightarrow$ dst
3: (Smem) << 16 $\rightarrow$ dst
4: (Smem) << SHIFT $\rightarrow$ dst
5: (Xmem) << SHFT $\rightarrow$ dst
6: K $\rightarrow$ dst
7: lk << SHFT $\rightarrow$ dst
8: lk << 16 $\rightarrow$ dst
9: (src) << ASM $\rightarrow$ dst
10: (src) << SHIFT $\rightarrow$ dst

Status Bits

Affected by SXM in all accumulator loads
Affected by OVM in loads with SHIFT or ASM shift
Affects OVdst (or OVsrc, when dst = src) in loads with SHIFT or ASM shift

LD :Load T/DP/ASM/ARP

Syntax	1: **LD** *Smem*, **T**
	2: **LD** *Smem*, **DP**
	3: **LD** *#k9*, **DP**
	4: **LD** *#k5*, **ASM**
	5: **LD** *#k3*, **ARP**
	6: **LD** *Smem*, **ASM**
Operands	Smem: Single data-memory operand
	$0 \le k9 \le 511$
	$-16 \le k5 \le 15$
	$0 \le k3 \le 7$
Execution	1: (Smem) $\rightarrow$ T
	2: (Smem(8–0)) $\rightarrow$ DP
	3: k9 $\rightarrow$ DP
	4: k5 $\rightarrow$ ASM
	5: k3 $\rightarrow$ ARP
	6: (Smem(4–0)) $\rightarrow$ ASM
Status Bits	None

LDM: Load Memory-Mapped Register

Syntax	**LDM** *MMR, dst*
Operands	MMR: Memory-mapped register
	dst: A (accumulator)
	B (accumulator)
Execution	(MMR) $\rightarrow$ dst(15–0)
	00 0000h $\rightarrow$ dst(39–16)
Status Bits	None

LD||MAC[R] : Load Accumulator With Parallel Multiply Accumulate With/Without Rounding

Syntax

LD *Xmem, dst*
|| **MAC[R]** *Ymem* [, *dst_*]

Operands

dst:	A (accumulator A)
	B (accumulator B)
dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A
Xmem, Ymem:	Dual data-memory operands

Execution

```
(Xmem) << 16 → dst (31–16)
If (Rounding)
     Round (((Ymem) × (T)) + (dst_)) → dst_
Else
     ((Ymem) × (T)) + (dst_) → dst_
```

Status Bits

Affected by SXM, FRCT, and OVM
Affects OVdst_

LD||MAS[R]: Load Accumulator With Parallel Multiply Subtract With/Without Rounding

Syntax

LD *Xmem, dst*
|| **MAS[R]** *Ymem* [, *dst_*]

Operands

Xmem, Ymem:	Dual data-memory operands
dst:	A (accumulator A)
	B (accumulator B)
dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A

Execution

```
(Xmem) << 16 → dst (31–16)
If (Rounding)
     Round ((dst_) – ((T) × (Ymem))) → dst_
Else
     (dst_) – ((T) × (Ymem)) → dst_
```

Status Bits

Affected by SXM, FRCT, and OVM
Affects OVdst_

LDR: Load Memory Value in Accumulator High With Rounding

Syntax	**LDR** *Smem, dst*	
Operands	Smem:	Single data-memory operand
	dst:	A (accumulator A)
		B (accumulator B)
Execution	$(Smem) << 16 + 1 << 15 \rightarrow dst(31-16)$	
Status Bits	Affected by SXM	

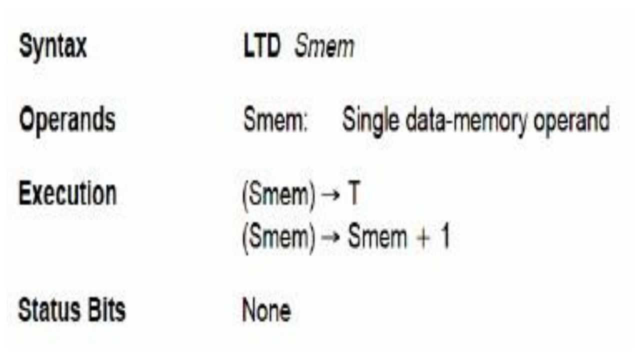
LDU :Load Unsigned Memory Value

Syntax	**LDU** *Smem, dst*	
Operands	Smem:	Single data-memory operand
	dst:	A (accumulator A)
		B (accumulator B)
Execution	$(Smem) \rightarrow dst(15-0)$	
	$00\ 0000h \rightarrow dst(39-16)$	

LMS: Least Mean Square

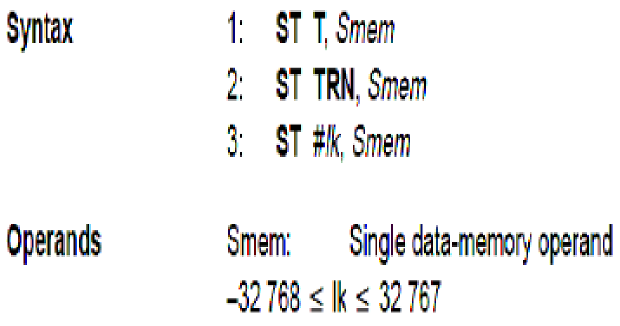
Syntax	**LMS** *Xmem, Ymem*	
Operands	Xmem, Ymem:	Dual data-memory operands
Execution	$(A) + (Xmem) << 16 + 2^{15} \rightarrow A$	
	$(B) + (Xmem) \times (Ymem) \rightarrow B$	
Status Bits	Affected by SXM, FRCT, and OVM	
	Affects C, OVA, and OVB	

LTD :Load T and Insert Delay

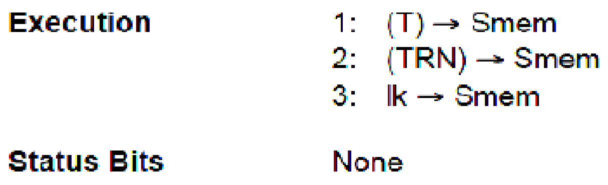


Syntax	**LTD** *Smem*
Operands	Smem: Single data-memory operand
Execution	(Smem) → T (Smem) → Smem + 1
Status Bits	None

ST : Store T, TRN, or Immediate Value Into Memory



Syntax	1:	**ST T**, *Smem*
	2:	**ST TRN**, *Smem*
	3:	**ST** #*lk*, *Smem*
Operands	Smem:	Single data-memory operand
	$-32\,768 \le lk \le 32\,767$	



Execution	1:	(T) → Smem
	2:	(TRN) → Smem
	3:	lk → Smem
Status Bits	None	

STH : Store Accumulator High Into Memory

Syntax	1: **STH** *src*, *Smem* 2: **STH** *src*, **ASM**, *Smem* 3: **STH** *src*, *SHFT*, *Xmem* 4: **STH** *src* [, *SHIFT*], *Smem*
Operands	src: A (accumulator A) B (accumulator B) Smem: Single data-memory operand Xmem: Dual data-memory operand $0 \leq \text{SHFT} \leq 15$ $-16 \leq \text{SHIFT} \leq 15$
Execution	1: $(src) << (-16) \rightarrow Smem$ 2: $(src) << (ASM - 16) \rightarrow Smem$ 3: $(src) << (SHFT - 16) \rightarrow Xmem$ 4: $(src) << (SHIFT - 16) \rightarrow Smem$
Status Bits	Affected by SXM

STL: Store Accumulator Low Into Memory

Syntax	1: **STL** *src*, *Smem* 2: **STL** *src*, **ASM**, *Smem* 3: **STL** *src*, *SHFT*, *Xmem* 4: **STL** *src* [, *SHIFT*], *Smem*
Operands	src: A (accumulator A) B (accumulator B) Smem: Single data-memory operand Xmem: Dual data-memory operand $0 \leq \text{SHFT} \leq 15$ $-16 \leq \text{SHIFT} \leq 15$
Execution	1: $(src) \rightarrow Smem$ 2: $(src) << ASM \rightarrow Smem$ 3: $(src) << SHFT \rightarrow Xmem$ 4: $(src) << SHIFT \rightarrow Smem$
Status Bits	Affected by SXM

ST||ADD : Store Accumulator With Parallel Add

Syntax	**ST** *src, Ymem*	
	|| **ADD** *Xmem, dst*	
Operands	src, dst:	A (accumulator A)
		B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands
	dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A
Execution	(src) << (ASM − 16) → Ymem	
	(dst_) + (Xmem) << 16 → dst	
Status Bits	Affected by OVM, SXM, and ASM	
	Affects C and OVdst	

ST||LD: Store Accumulator with Parallel Load

Syntax	1:	**ST** *src, Ymem*
		|| **LD** *Xmem, dst*
	2:	**ST** *src, Ymem*
		|| **LD** *Xmem*, **T**
Operands	src, dst:	A (accumulator A)
		B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands
Execution	1.	(src) << (ASM − 16) → Ymem
		(Xmem) << 16 → dst
	2.	(src) << (ASM − 16) → Ymem
		(Xmem) → T
Status Bits	Affected by OVM and ASM	
	Affects C	

ST||MAC[R]: Store Accumulator With Parallel Multiply Accumulate With/Without Rounding

Syntax

ST *src, Ymem*
|| MAC[R] *Xmem, dst*

Operands

	src, dst:	A (accumulator A)
		B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands

Execution

```
(src << (ASM – 16)) → Ymem
If (Rounding)
    Then
    Round ((Xmem) × (T) + (dst)) → dst
Else
    (Xmem) × (T) + (dst) → dst
```

Status Bits

Affected by OVM, SXM, ASM, and FRCT
Affects C and OVdst

ST||MAS[R]: Store Accumulator With Parallel Multiply Subtract With/Without Rounding

Syntax

ST *src, Ymem*
|| MAS[R] *Xmem, dst*

Operands

	src, dst:	A (accumulator A)
		B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands

Execution

```
(src << (ASM – 16)) → Ymem
If (Rounding)
    Then
    Round ((dst) – (Xmem) × (T))→ dst
Else
    (dst) – (Xmem) × (T) → dst
```

Status Bits

Affected by OVM, SXM, ASM, and FRCT
Affects C and OVdst

ST||MPY: Store Accumulator With Parallel Multiply

Syntax	**ST** *src, Ymem* || **MPY** *Xmem, dst*	
Operands	src, dst:	A (accumulator A) B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands
Execution	(src << (ASM − 16)) → Ymem (T) × (Xmem) → dst	
Status Bits	Affected by OVM, SXM, ASM, and FRCT Affects C and OVdst	

ST||SUB: Store Accumulator With Parallel Subtract

Syntax	**ST** *src, Ymem* || **SUB** *Xmem, dst*	
Operands	src, dst:	A (accumulator A) B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands
	dst_:	If *dst* = A, then *dst_* = B; if *dst* = B, then *dst_* = A.
Execution	(src << (ASM − 16)) → Ymem (Xmem) << 16 – (dst_) → dst	
Status Bits	Affected by OVM, SXM, and ASM Affects C and OVdst	

STRCD: Store T Conditionally

Syntax **STRCD** *Xmem, cond*

Operands Xmem: Dual data-memory operand

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) ≠ 0	0100	BNEQ	(B) ≠ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) ≥ 0	0010	BGEQ	(B) ≥ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) ≤ 0	0111	BLEQ	(B) ≤ 0	1111

Execution

```
If (cond)
    (T) → Xmem
Else
    (Xmem) → Xmem
```

Status Bits None

4.1.4. Miscellaneous Load-Type and Store-Type Instructions

MVDD: Move Data From Data Memory to Data Memory With X, Y addressing

Syntax **MVDD** *Xmem, Ymem*

Operands Xmem, Ymem: Dual data-memory operands

Execution (Xmem) → Ymem

Status Bits None

MVDK: Move Data From Data Memory to Data Memory With Destination Addressing

Syntax **MVDK** *Smem, dmad*

Operands Smem: Single data-memory operand
$0 \le \text{dmad} \le 65\,535$

Execution

```
(dmad) → EAR
If (RC) ≠ 0
Then
    (Smem) → Dmem addressed by EAR
    (EAR) + 1 → EAR
Else
    (Smem) → Dmem addressed by EAR
```

Status Bits None

MVDM: Move Data From Data Memory to Memory-Mapped Register

Syntax **MVDM** *dmad, MMR*

Operands MMR: Memory-mapped register
$0 \leq \text{dmad} \leq 65\,535$

Execution

```
dmad → DAR
If (RC) ≠ 0
Then
    (Dmem addressed by DAR) → MMR
    (DAR) + 1 → DAR
Else
    (Dmem addressed by DAR) → MMR
```

Status Bits None

MVDP: Move Data from Data Memory to Program Memory

Syntax **MVDP** *Smem, pmad*

Operands Smem: Single data-memory operand
$0 \leq \text{pmad} \leq 65\,535$

Execution

```
pmad → PAR
If (RC) ≠ 0
Then
    (Smem) → Pmem addressed by PAR
    (PAR) + 1 → PAR
Else
    (Smem) → Pmem addressed by PAR
```

Status Bits None

MVKD: Move Data From Data Memory to Data Memory With Source Addressing

Syntax **MVKD** *dmad, Smem*

Operands Smem: Single data-memory operand
0 ≤ dmad ≤ 65 535

Execution

```
dmad → DAR
If (RC) ≠ 0
Then
    (Dmem addressed by DAR) → Smem
    (DAR) + 1 → DAR
Else
    (Dmem addressed by DAR) → Smem
```

Status Bits None

Example 1 `MVKD 300h, 0`

	Before Instruction		After Instruction
DP	004	DP	004
Data Memory			
0200h	ABCD	0200h	1234
0300h	1234	0300h	1234

MVMD: Move Data From Memory-Mapped Register to Data Memory

Syntax **MVMD** *MMR, dmad*

Operands MMR: Memory-mapped register
0 ≤ dmad ≤ 65 535

Execution

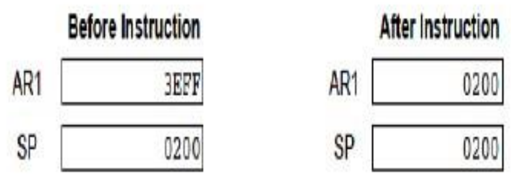
```
dmad → EAR
If (RC) ≠ 0
Then
    (MMR) → Dmem addressed by EAR
    (EAR) + 1 → EAR
Else
    (MMR) → Dmem addressed by EAR
```

Status Bits None

MVMM: Move Data From Memory-Mapped Register to Memory-Mapped Register

Syntax	**MVMM** *MMRx, MMRy*
Operands	MMRx: AR0–AR7, SP MMRy: AR0–AR7, SP
Execution	(MMRx) → MMRy
Status Bits	None

Example MVMM SP, AR1



MVPD: Move Data From Program Memory to Data Memory

Syntax	**MVPD** *pmad, Smem*
Operands	Smem: Single data-memory operand 0 ≤ pmad ≤ 65 535
Execution	pmad → PAR If (RC) ≠ 0 Then (Pmem addressed by PAR) → Smem (PAR) + 1 → PAR Else (Pmem addressed by PAR) → Smem
Status Bits	None

PORTR: Read Data from Port

PORTW: Write Data to Port

Syntax	**PORTW** *Smem, PA*
Operands	Smem: Single data-memory operand 0 ≤ PA ≤ 65 535
Execution	(Smem) → PA
Status Bits	None

READA: Read Program Memory addressed by Accumulator A and Store in Data Memory

Syntax	**READA** *Smem*
Operands	Smem: Single data-memory operand
Execution	A → PAR If ((RC)≠0) (Pmem (addressed by PAR)) → Smem (PAR) + 1 → PAR (RC) – 1 → RC Else (Pmem (addressed by PAR)) → Smem
Status Bits	None

WRITA: Write Data to Program Memory Addressed by Accumulator A

Syntax	**WRITA** *Smem*
Operands	Smem: Single data-memory operand
Execution	A → PAR If (RC) ≠ 0 Then (Smem) → (Pmem addressed by PAR) (PAR) + 1 → PAR (RC) – 1 → RC Else (Smem) → (Pmem addressed by PAR)
Status Bits	None

Branch Instructions

B[D]: Branch Unconditionally

Syntax	**B[D]** *pmad*
Operands	$0 \leq$ pmad $\leq$ 65 535
Execution	pmad → PC
Status Bits	None

BACC[D]: Branch to Location Specified by Accumulator

Syntax	**BACC[D]** *src*
Operands	src: A (accumulator A) B (accumulator B)
Execution	(src(15–0)) → PC
Status Bits	None

BANZ[D]: Branch on Auxiliary Register Not Zero

Syntax	**BANZ[D]** *pmad, Sind*
Operands	Sind: Single indirect addressing operand 0 ≤ pmad ≤ 65 535
Execution	If ((ARx) ≠ 0) Then pmad → PC Else (PC) + 2 → PC
Status Bits	None

BC [D]: Branch Conditionally

Syntax	**BC[D]** *pmad, cond* [, *cond* [, *cond*]]
Execution	If (cond(s)) Then pmad → PC Else (PC) + 2 → PC
Status Bits	Affects OVA or OVB if OV or NOV is chosen

FB [D]: Far Branch Unconditionally

Syntax	**FB[D]** *extpmad*
Operands	0 ≤ extpmad ≤ 7F FFFF
Execution	(pmad(15–0)) → PC (pmad(22–16)) → XPC
Status Bits	None

FBACC [D]: Far Branch to Location Specified by Accumulator

Syntax	**FBACC[D]** *src*
Operands	src: A (accumulator A) B (accumulator B)
Execution	(src(15–0)) → PC (src(22–16)) → XPC
Status Bits	None

CALA [D]: Call Subroutine at Location Specified by Accumulator

Syntax	**CALA[D]** *src*
Operands	src: A (accumulator A) B (accumulator B)
Execution	**Nondelayed** (SP) – 1 → SP (PC) + 1 → TOS (src(15–0)) → PC **Delayed** (SP) – 1 → SP (PC) + 3 → TOS (src(15–0)) → PC
Status Bits	None

CALL[D]: Call Unconditionally

Syntax	**CALL[D]** *pmad*
Operands	$0 \leq$ pmad $\leq 65\,535$
Execution	**Nondelayed** (SP) – 1 → SP (PC) + 2 → TOS pmad → PC **Delayed** (SP) – 1 → SP (PC) + 4 → TOS pmad → PC
Status Bits	None

CC [D]: Call Conditionally

Syntax **CC[D]** *pmad*, *cond* [, *cond* [, *cond*]]

Operands $0 \le \text{pmad} \le 65\,535$

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{\text{BIO}}$ low	0000 0011	NBIO	$\overline{\text{BIO}}$ high	0000 0010
C	C = 1	0000 1100	NC	C = 0	0000 1000
TC	TC = 1	0011 0000	NTC	TC = 0	0010 0000
AEQ	(A) = 0	0100 0101	BEQ	(B) = 0	0100 1101
ANEQ	(A) ≠ 0	0100 0100	BNEQ	(B) ≠ 0	0100 1100
AGT	(A) > 0	0100 0110	BGT	(B) > 0	0100 1110
AGEQ	(A) ≥ 0	0100 0010	BGEQ	(B) ≥ 0	0100 1010
ALT	(A) < 0	0100 0011	BLT	(B) < 0	0100 1011
ALEQ	(A) ≤ 0	0100 0111	BLEQ	(B) ≤ 0	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Execution

```
Nondelayed
If (cond(s))
Then
    (SP) − 1 → SP
    (PC) + 2 → TOS
    pmad → PC
Else
    (PC) + 2 → PC

Delayed
If (cond(s))
Then
    (SP) − 1 → SP
    (PC) + 4 → TOS
    pmad → PC
Else
    (PC) + 2 → PC
```

Status Bits Affects OVA or OVB (if OV or NOV is chosen)

FCALA [D]: Far Call Subroutine at Location Specified by Accumulator

Syntax	FCALA[D] *src*
Operands	src: A (accumulator A) B (accumulator B)

Execution

```
Nondelayed
(SP) – 1 → SP
(PC) + 1 → TOS
(SP) – 1 → SP
(XPC) → TOS
(src(15–0)) → PC
(src(22–16)) → XPC

Delayed
(SP) – 1 → SP
(PC) + 3 → TOS
(SP) – 1 → SP
(XPC) → TOS
(src(15–0)) → PC
(src(22–16)) → XPC
```

Status Bits None

FCALL[D]: Far Call Unconditionally

Syntax	**FCALL[D]** *extpmad*
Operands	0 ≤ extpmad ≤ 7F FFFF
Execution	**Nondelayed** (SP) − 1 → SP (PC) + 2 → TOS (SP) − 1 → SP (XPC) → TOS (pmad(15–0)) → PC (pmad(22–16)) → XPC
	Delayed (SP) − 1 → SP (PC) + 4 → TOS (SP) − 1 → SP (XPC) → TOS (pmad(15–0)) → PC (pmad(22–16)) → XPC
Status Bits	None

Interrupt Instructions : INTR: Software Interrupt

Syntax	**INTR** *K*
Operands	0 ≤ K ≤ 31
Execution	(SP) − 1 → SP (PC) + 1 → TOS interrupt vector specified by K → PC 1 → INTM
Status Bits	Affects INTM and IFR

TRAP: Software Interrupt

Syntax	**TRAP** *K*
Operands	$0 \leq K \leq 31$
Execution	$(SP) - 1 \rightarrow SP$ $(PC) + 1 \rightarrow TOS$ Interrupt vector specified by $K \rightarrow PC$
Status Bits	None

Return Instructions

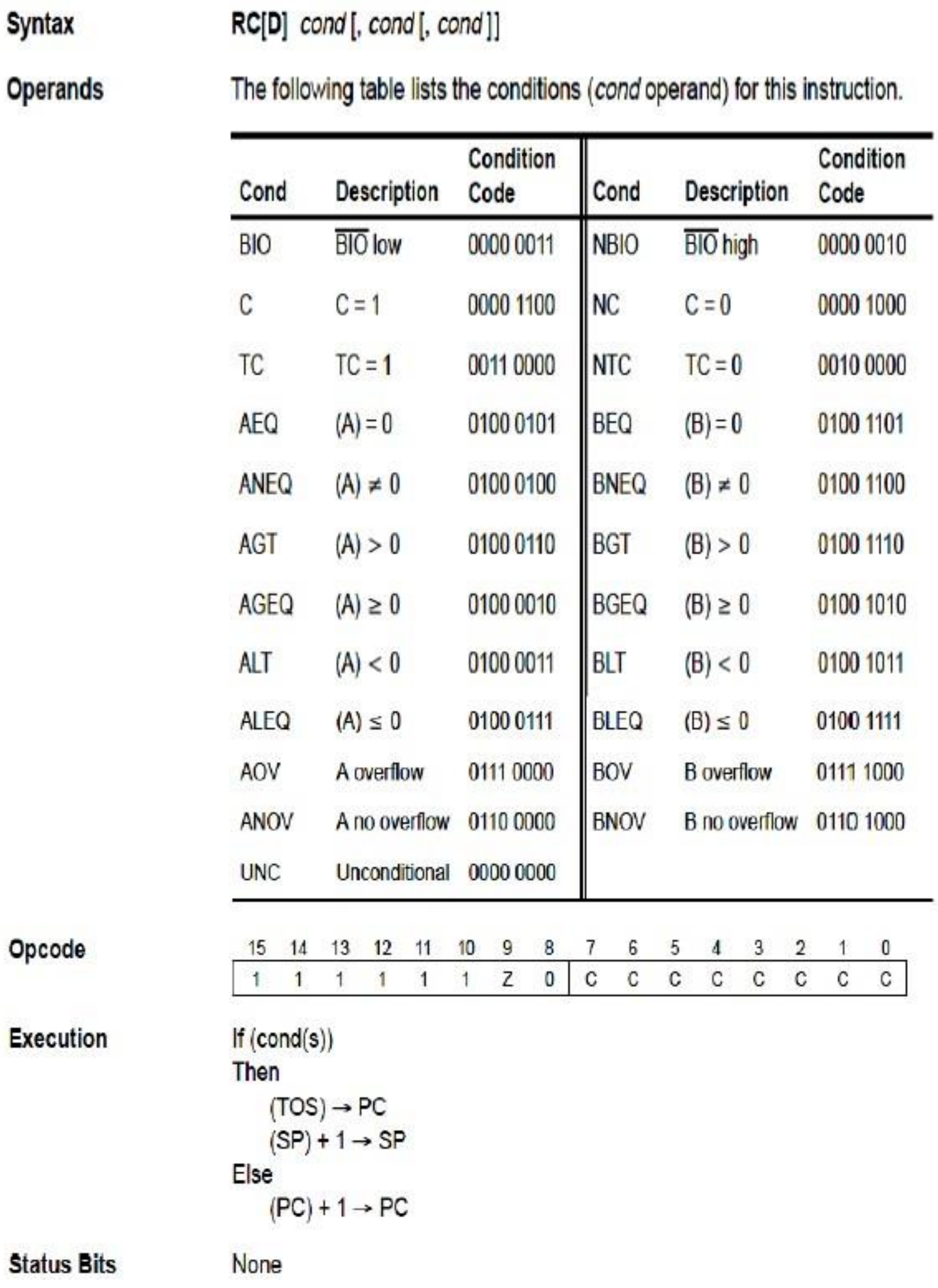
FRET [D]: Far Return

Syntax	**FRET[D]**
Operands	None
Execution	$(TOS) \rightarrow XPC$ $(SP) + 1 \rightarrow SP$ $(TOS) \rightarrow PC$ $(SP) + 1 \rightarrow SP$
Status Bits	None

FRETE [D]: Enable Interrupts and Far Return From Interrupt

Syntax	**FRETE[D]**
Operands	None
Execution	$(TOS) \rightarrow XPC$ $(SP) + 1 \rightarrow SP$ $(TOS) \rightarrow PC$ $(SP) + 1 \rightarrow SP$ $0 \rightarrow INTM$
Status Bits	Affects INTM

RC [D]: Return Conditionally



Syntax RC[D] *cond* [, *cond* [, *cond*]]

Operands The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{\text{BIO}}$ low	0000 0011	NBIO	$\overline{\text{BIO}}$ high	0000 0010
C	C = 1	0000 1100	NC	C = 0	0000 1000
TC	TC = 1	0011 0000	NTC	TC = 0	0010 0000
AEQ	(A) = 0	0100 0101	BEQ	(B) = 0	0100 1101
ANEQ	(A) ≠ 0	0100 0100	BNEQ	(B) ≠ 0	0100 1100
AGT	(A) > 0	0100 0110	BGT	(B) > 0	0100 1110
AGEQ	(A) ≥ 0	0100 0010	BGEQ	(B) ≥ 0	0100 1010
ALT	(A) < 0	0100 0011	BLT	(B) < 0	0100 1011
ALEQ	(A) ≤ 0	0100 0111	BLEQ	(B) ≤ 0	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Opcode

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

Execution

```
If (cond(s))
Then
    (TOS) → PC
    (SP) + 1 → SP
Else
    (PC) + 1 → PC
```

Status Bits None

RET [D]: Return

Syntax	**RET[D]**
Operands	None
Execution	(TOS) → PC (SP) + 1 → SP
Status Bits	None

RETF [D]: Enable Interrupts and Fast Return From Interrupt

Syntax	**RETF[D]**
Operands	None
Execution	(RTN) → PC (SP) + 1 → SP 0 → INTM
Status Bits	Affects INTM

Repeat Instructions

RPT: Repeat Next Instruction

Syntax	1: **RPT** *Smem* 2: **RPT** #*K* 3: **RPT** #*lk*
Operands	Smem: Single data-memory operand $0 \leq K \leq 255$ $0 \leq lk \leq 65\,535$
Execution	1: (Smem) → RC 2: K → RC 3: lk → RC
Status Bits	None

RPTB [D]: Block Repeat

Syntax	**RPTB[D]** *pmad*
Operands	0 ≤ pmad ≤ 65 535
Execution	1 → BRAF If (delayed) then (PC) + 4 → RSA Else (PC) + 2 → RSA pmad → REA
Status Bits	Affects BRAF

RPTZ: Repeat Next Instruction and Clear Accumulator

Syntax	**RPTZ** *dst, #lk*
Operands	dst: A (accumulator A) B (accumulator B) 0 ≤ lk ≤ 65 535
Execution	0 → dst lk → RC
Status Bits	None

Stack-Manipulating Instructions

FRAME: Stack Pointer Immediate Offset

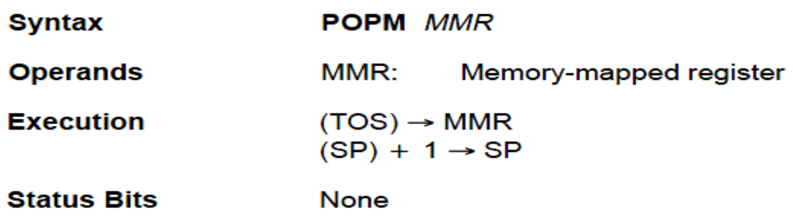
Operands	–128 ≤ K ≤ 127
Execution	(SP) + K → SP
Status Bits	None
Example	`FRAME 10h`

	Before Instruction		After Instruction
SP	1000	SP	1010

POPD: Pop Top of Stack to Data Memory

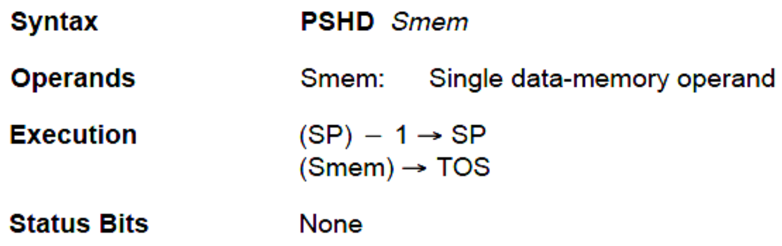
Syntax	**POPD** *Smem*
Operands	Smem: Single data-memory operand
Execution	(TOS) → Smem (SP) + 1 → SP
Status Bits	None

POPM: Pop Top of Stack to Memory-Mapped Register



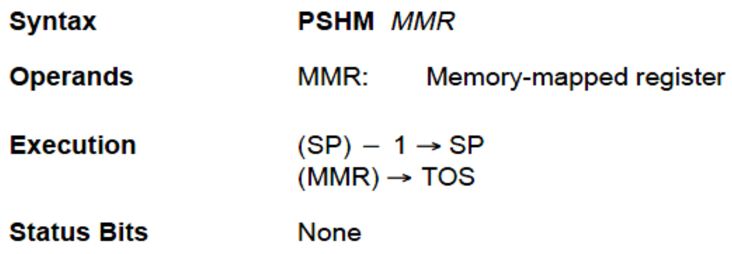
Syntax	**POPM** *MMR*
Operands	MMR: Memory-mapped register
Execution	(TOS) → MMR (SP) + 1 → SP
Status Bits	None

PSHD: Push Data-Memory Value onto Stack



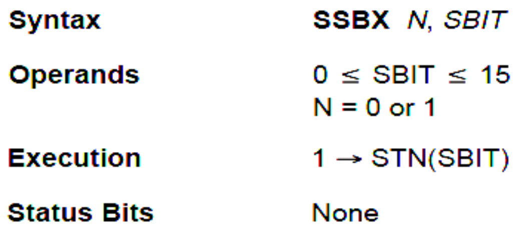
Syntax	**PSHD** *Smem*
Operands	Smem: Single data-memory operand
Execution	(SP) − 1 → SP (Smem) → TOS
Status Bits	None

PSHM: Push Memory-Mapped Register onto Stack



Syntax	**PSHM** *MMR*
Operands	MMR: Memory-mapped register
Execution	(SP) − 1 → SP (MMR) → TOS
Status Bits	None

Miscellaneous Program-Control
Instructions SSBX: Set Status
Register Bit



Syntax	**SSBX** *N*, *SBIT*
Operands	0 ≤ SBIT ≤ 15 N = 0 or 1
Execution	1 → STN(SBIT)
Status Bits	None

RSBX: Reset Status Register Bit

Syntax	**RSBX** *N, SBIT*
Operands	0 ≤ SBIT ≤ 15 N = 0 or 1
Execution	0 → STN(SBIT)
Status Bits	None

Example 1 `RSBX SXM ; SXM means: n=1 and SBIT=8`

	Before Instruction		After Instruction
ST1	35CD	ST1	34CD

NOP: No Operation

Syntax	**NOP**
Operands	None
Execution	None
Status Bits	None

RESET: Software Reset

Syntax	**RESET**
Operands	None
Execution	These fields of PMST, ST0, and ST1 are loaded with the values shown:

(IPTR) << 7 → PC	0 → OVA	0 → OVB
1 → C	1 → TC	0 → ARP
0 → DP	1 → SXM	0 → ASM
0 → BRAF	0 → HM	1 → XF
0 → C16	0 → FRCT	0 → CMPT
0 → CPL	1 → INTM	0 → IFR
0 → OVM		

Status Bits	The status bits affected are listed in the execution section.

On chip peripherals:

It facilitates interfacing with external devices. The peripherals are:

- General purpose I/O pins
- A software programmable wait state generator.
- Hardware timer
- Host port interface (HPI)
- Clock generator
- Serial port

It has two general purpose I/O pins:

➢ BIO-input pin used to monitor the status of external devices.
➢ XF- output pin, software controlled used to signal external devices

Software programmable wait state generator:
➢ Extends external bus cycles up to seven machine cycles.

Hardware Timer
 An on chip down counter
 Used to generate signal to initiate any interrupt or any other process

Consists of 3 memory mapped registers:
➢ The timer register (TIM)
➢ Timer period register (PRD)
➢ Timer controls register (TCR)
- Pre scaler block (PSC).
- TDDR (Time Divide Down ratio)
- TIN &TOUT

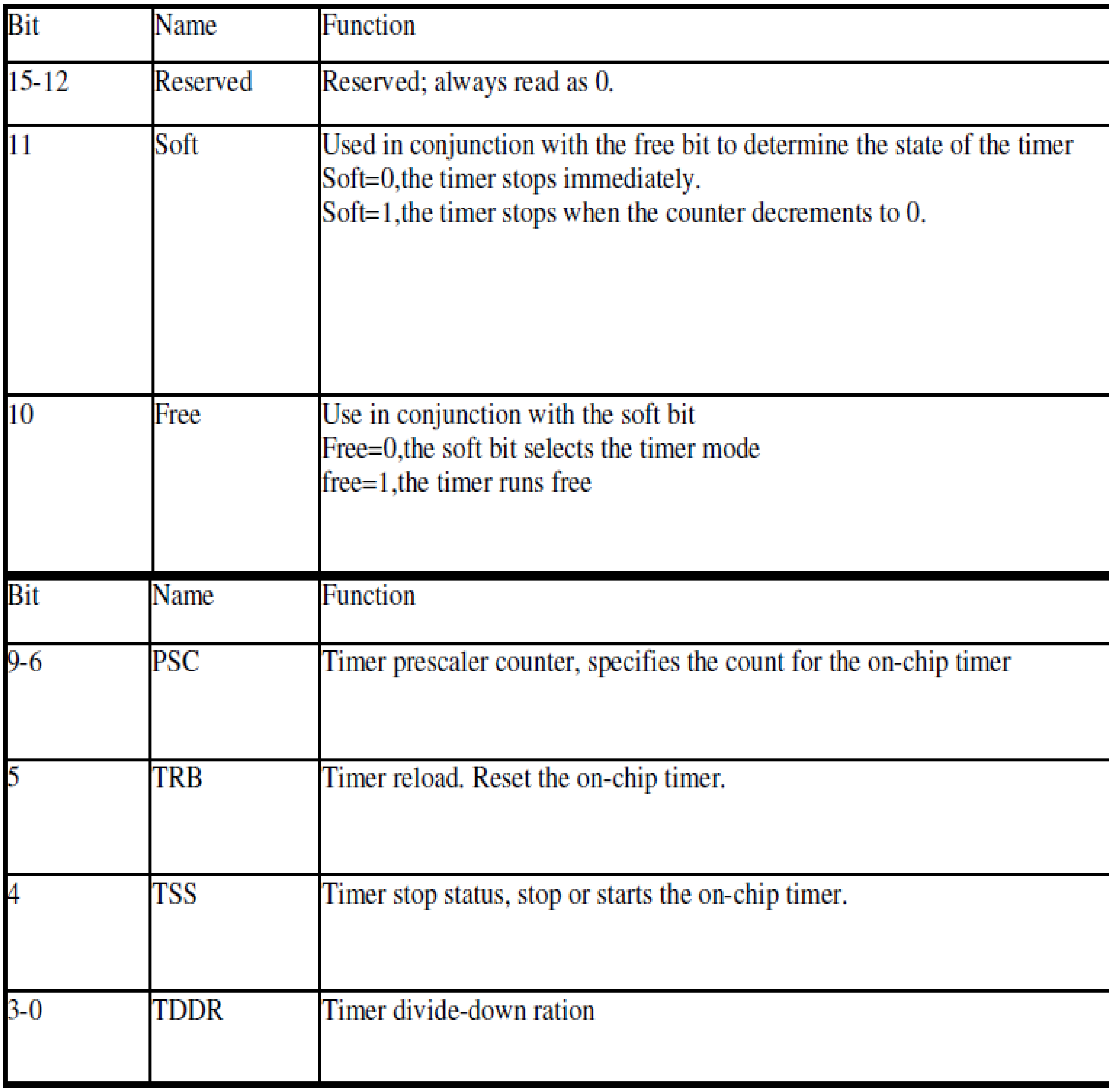
The timer register (TIM) is a 16-bit memory-mapped register that decrements at every pulse from the prescaler block (PSC).
The timer period register (PRD) is a 16-bit memory-mapped register whose contents are loaded onto the TIM whenever the TIM decrements to zero or the device is reset (SRESET).

The timer can also be independently reset using the TRB signal. The timer control register (TCR) is a 16-bit memory-mapped register that contains status and control bits. Table shows the functions of the various bits in the TCR.

The prescaler block is also an on-chip counter. Whenever the prescaler bits count down to 0, a clock pulse is given to the TIM register that decrements the TIM register by 1. The TDDR bits contain the divide-down ratio, which is loaded onto the prescaler block after each time the prescaler bits count down to 0.

That is to say that the 4-bit value of TDDR determines the divide-by ratio of the timer clock with respect to the system clock. In other words, the TIM decrements either at the rate of the system clock or at a rate slower than that as decided by the value of the TDDR bits. TOUT and TINT are the output signal generated as the TIM register decrements to 0. TOUT can trigger the start of the conversion signal in an ADC interfaced to the DSP.

The sampling frequency of the ADC determines how frequently it receives the TOUT signal. TINT is used to generate interrupts, which are required to service a peripheral such as a DRAM controller periodically. The timer can also be stopped, restarted, reset, or disabled by specific status bits.



Bit	Name	Function
15-12	Reserved	Reserved; always read as 0.
11	Soft	Used in conjunction with the free bit to determine the state of the timer Soft=0,the timer stops immediately. Soft=1,the timer stops when the counter decrements to 0.
10	Free	Use in conjunction with the soft bit Free=0,the soft bit selects the timer mode free=1,the timer runs free
9-6	PSC	Timer prescaler counter, specifies the count for the on-chip timer
5	TRB	Timer reload. Reset the on-chip timer.
4	TSS	Timer stop status, stop or starts the on-chip timer.
3-0	TDDR	Timer divide-down ration

Table 4.6. Pin details of software wait state generator

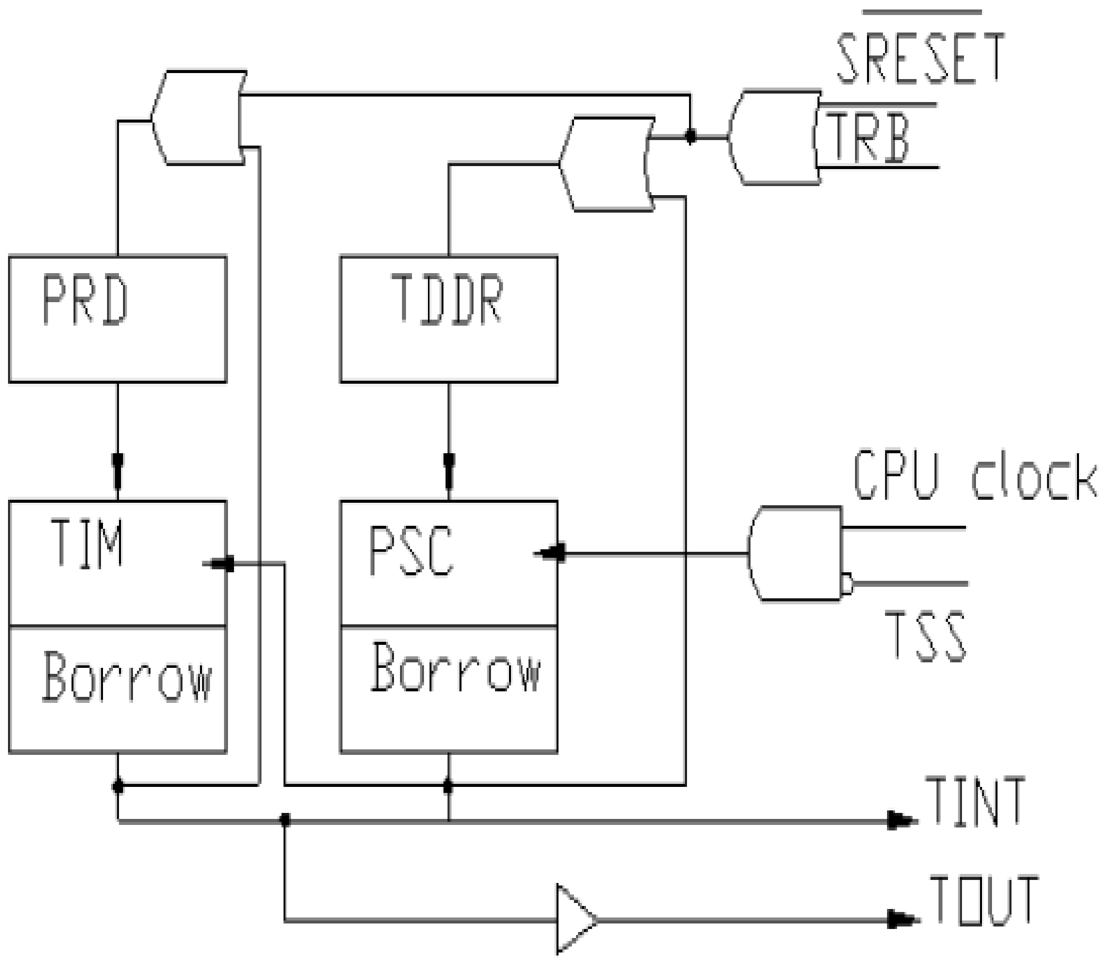
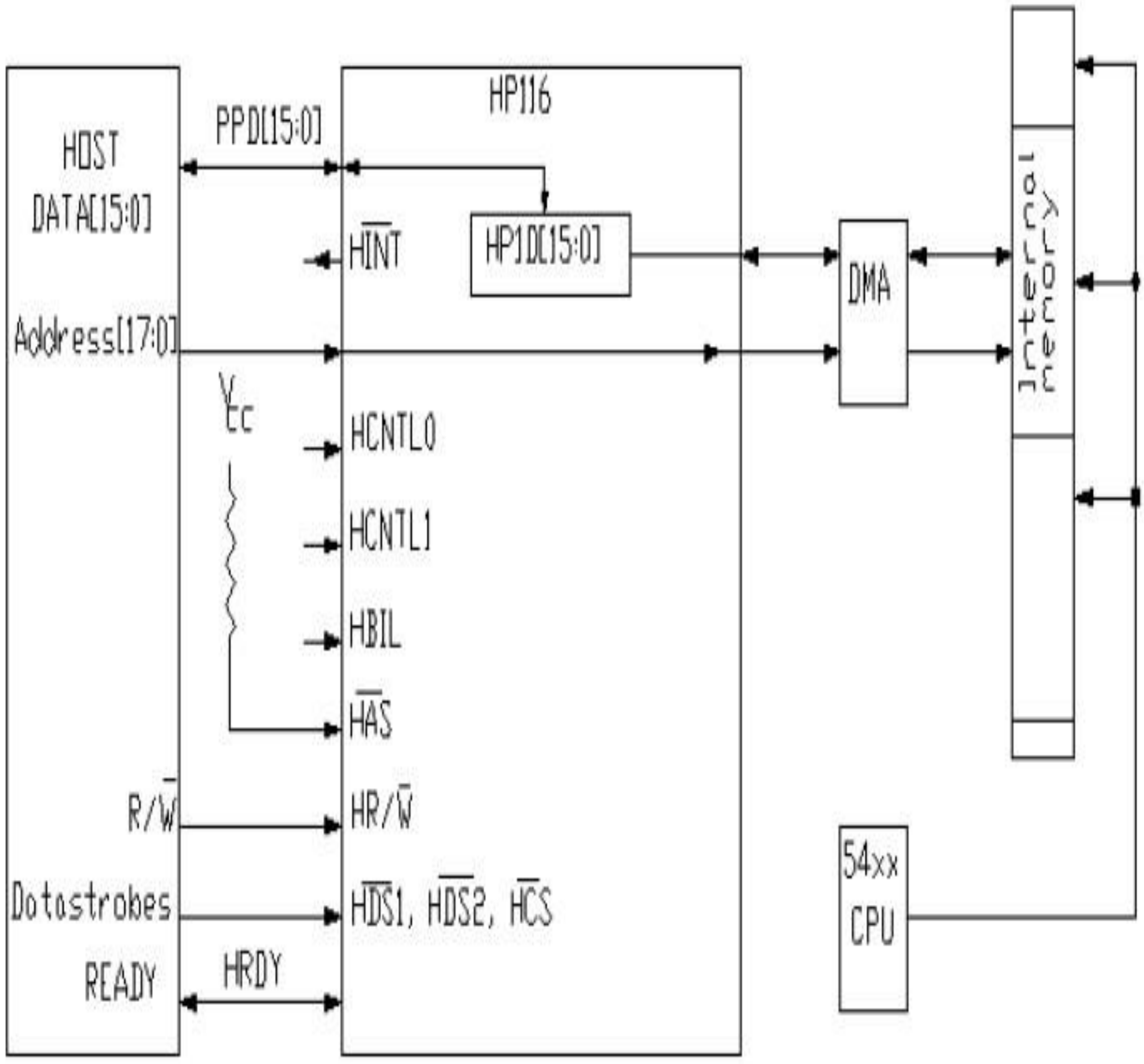


Figure 4.2.Logical block diagram of timer circuit.

Host port interface (HPI):

- Allows to interface to an 8bit or 16bit host devices or a host processor
- Signals in HPI are:
- Host interrupt (HINT)
- HRDY
- HCNTL0 &HCNTL1
- HBIL
- HR/w



4.3. A generic diagram of the host port interface (HPI)

Important signals in the HPI are as follows:

- The 16-bit data bus and the 18-bit address bus.
- The host interrupt, Hint, for the DSP to signal the host when it attention is required.
- HRDY, a DSP output indicating that the DSP is ready for transfer.
- HCNTL0 and HCNTL1, control signal that indicate the type of transfer to carry out.
- The transfer types are data, address, etc.
- HBIL. If this is low it indicates that the current byte is the first byte; if it is high, it indicates that it is second byte.
- HR/W indicates if the host is carrying out a read operation or a write operation

Clock Generator:
The clock generator on TMS320C54xx devices has two options-an external clock and the internal clock. In the case of the external clock option, a clock source is directly connected to the device. The internal clock source option, on the other hand, uses an internal clock generator and a phase locked loop (PLL) circuit. The PLL, in turn, can be hardware configured or software programmed. Not all devices of the TMS320C54xx family have all these clock options; they vary from device to device.

Serial I/O Ports:
Three types of serial ports are available:

- Synchronous ports.
- Buffered ports.
- Time-division multiplexed ports.

The synchronous serial ports are high-speed, full-duplex ports and that provide direct communications with serial devices, such as codec, and analog-to-digital (A/D) converters. A buffered serial port (BSP) is synchronous serial port that is provided with an auto buffering unit and is clocked at the full clock rate. The head of servicing interrupts. A time- division multiplexed (TDM) serial port is a synchronous serial port that is provided to allow time- division multiplexing of the data. The functioning of each of these on-chip peripherals is controlled by memory-mapped registers assigned to the respective peripheral.

Interrupts of TMS320C54xx Processors:
Many times, when CPU is in the midst of executing a program, a peripheral device may require a service from the CPU. In such a situation, the main program may be interrupted by a signal generated by the peripheral devices. This results in the processor suspending the main program in order to execute another program, called interrupt service routine, to service the peripheral device. On completion of the interrupt service routine, the processor returns to the main program to continue from where it left.
Interrupt may be generated either by an internal or an external device. It may also be generated by software. Not all interrupts are serviced when they occur. Only those interrupts that are called *nonmaskable* are serviced whenever they occur. Other interrupts, which are called *maskable* interrupts, are serviced only if they are enabled. There is also a priority to determine which interrupt gets serviced first if more than one interrupts occur simultaneously.
Almost all the devices of TMS320C54xx family have 32 interrupts. However, the types and the number under each type vary from device to device. Some of these interrupts are reserved for use by the CPU.

Pipeline operation of TMS320C54xx Processors:
The CPU of ‘54xx devices have a six-level-deep instruction pipeline. The six stages of the pipeline are independent of each other. This allows overlapping execution of instructions. During any given cycle, up to six different instructions can be active, each at a different stage of processing. The six levels of the pipeline structure are program prefetch, program

fetch, decode, access, read and execute.

1 During program prefetch, the program address bus, PAB, is loaded with the address of the next instruction to be fetched.

2 In the fetch phase, an instruction word is fetched from the program bus, PB, and loaded into the instruction register, IR. These two phases from the instruction fetch sequence.

3 During the decode stage, the contents of the instruction register, IR are decoded to determine the type of memory access operation and the control signals required for the data-address generation unit and the CPU.

4 The access phase outputs the read operand's on the data address bus, DAB. If a second operand is required, the other data address bus, CAB, also loaded with an appropriate address. Auxiliary registers in indirect addressing mode and the stack pointer (SP) are also updated. In the read phase the data operand(s), if any, are read from the data buses, DB and CB. This phase completes the two-phase read process and starts the two phase write processes. The data address of the write operand, if any, is loaded into the data write address bus, EAB.

5 The execute phase writes the data using the data write bus, EB, and completes the operand write sequence. The instruction is executed in this phase.

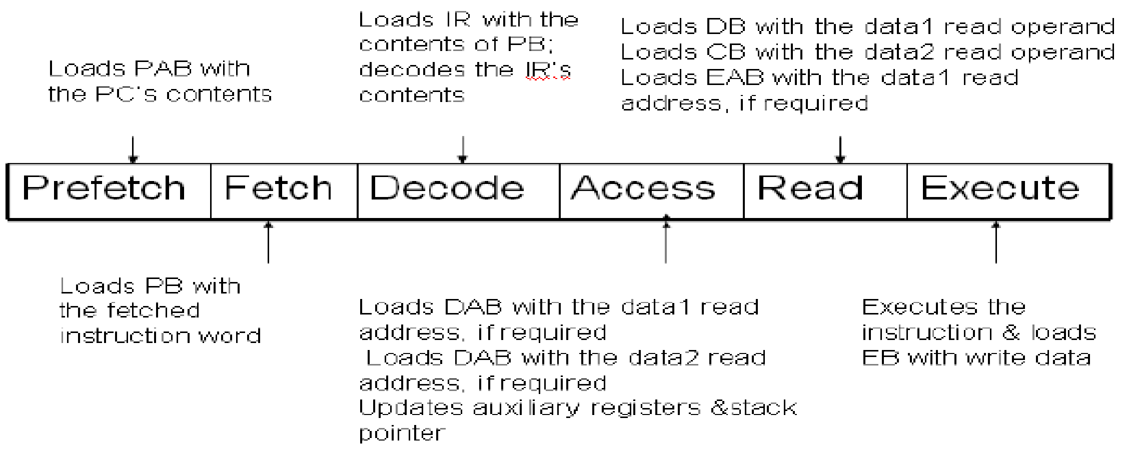

Loads PAB with the PC's contents
Loads IR with the contents of PB; decodes the IR's contents
Loads DB with the data1 read operand Loads CB with the data2 read operand Loads EAB with the data1 read address, if required
Prefetch
Fetch
Decode
Access
Read
Execute
Loads PB with the fetched instruction word
Loads DAB with the data1 read address, if required Loads DAB with the data2 read address, if required Updates auxiliary registers &stack pointer
Executes the instruction & loads EB with write data

Figure 4.4. Pipeline operation of TMS320C54xx Processors

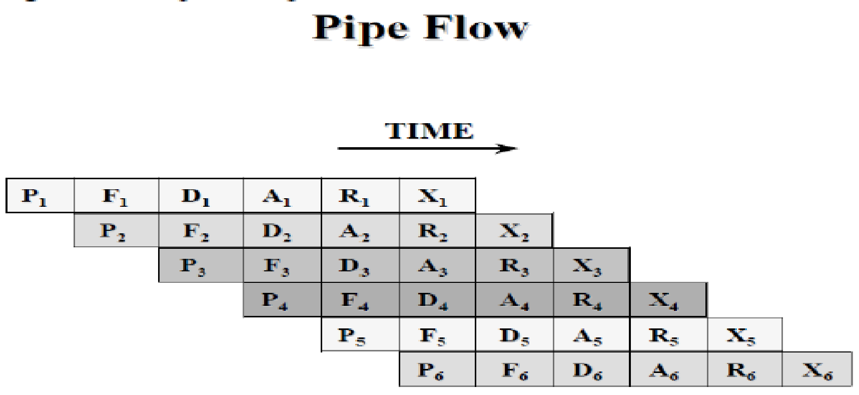


Figure 4.5.Pipe flow diagram

ASSIGNMENT NO: 5 (Recommended Questions)

1. Describe Host Port Interface and explain its signals.
2. writes an assembly language program of TMS320C54XX processors to compute the sum of three product terms given by the equation y(n)=h(0)x(n)+h(1)x(n-1)+h(2)x(n-2) with usual notations. Find y (n) for signed 16 bit data samples and 16 bit constants.
3. Describe the pipelining operation of TMS320C54XX processors.
4. Explain the operation of serial I/O ports and hardware timer of TMS320C54XXon chip peripherals.
5. Explain the different types of interrupts in TMS320C54xx Processor**s.**
6. Describe the operation of the following instructions of TMS 320c54xx processor, with example Describe the operation of hardware timer with neat diagram.
7. By means of a figure explain the pipeline operation of the following sequence of instruction if the initial values of AR1,AR3,A are 104,101,2 and the values stored in the memory locations 101,102,103,104 are 4,6,8,12. Also provide the values of registers AR3, AR1,T & A**.**
8. Describe the operation of the following instructions of TMS320C54XX processors.
9. Describe the operation of the following instructions of TMS320C54XX processors. **(July 12, 8m)**
10. Explain the following assembler directives of TMS320C54XX processors (i) .mmregs (ii) global (iii) .include ‘xx’ (iv) .data (v) .end (vi) .bss **(Dec09/Jan 10 6marks)**
11. Describe Host Port Interface and explain its signals. **(Dec 09/Jan 10 6marks)**
12. writes an assembly language program of TMS320C54XX processors to compute the sum of three product terms given by the equation y(n)=h(0)x(n)+h(1)x(n-1)+h(2)x(n-2) with usual notations. Find y (n) for signed 16 bit data samples and 16 bit constants. **(May/June 2011, 6m)**
13. Describe the pipelining operation of TMS320C54XX processors.**(Dec.11, 8m)**
14. Explain the operation of serial I/O ports and hardware timer of TMS320C54Xon chip peripherals. **(Dec.11, 8m)**
15. Explain the different types of interrupts in TMS320C54xx Processors**.(May/June 2009, 6m)**