MODULE VI

TMS320 DSP PROCESSOR

The TMS320C24x is a member of the TMS320 family of digital signal processors (DSPs). The ’C24x is designed to meet a wide range of digital motor control (DMC) and embedded control applications.

TMS320 Family Overview

The TMS320 family consists of fixed-point, floating-point, multiprocessor digital signal processors (DSPs), and fixed-point DSP controllers. TMS320 DSPs have an architecture designed specifically for real-time signal processing. The ’C24x series of DSP controllers combines this real-time processing capability with controller peripherals to create an ideal solution for control system applications.

The characteristics of TMS320 family are

- Very flexible instruction set
- Inherent operational flexibility
- High-speed performance
- Innovative parallel architecture
- Cost effectiveness

In 1982, Texas Instruments introduced the TMS32010, the first fixed-point DSP in the TMS320 family. Before the end of the year, Electronic Products magazine awarded the TMS32010 the title “Product of the Year”.

Today, the TMS320 family consists of the following generations

- ’C1x, ’C2x, ’C24x, ’C5x, ’C54x, and ’C6x fixed-point DSPs;
- ’C3x and ’C4x floating-point DSPs; and
- ’C8x multiprocessor DSPs.

The ’C24x is considered part of the ’C24x family of fixed-point DSPs, and a member of the ’C2000 platform. Devices within a generation of the TMS320 family have the same CPU structure but different on-chip memory and peripheral configurations. By integrating memory and peripherals onto a single chip, TMS320 devices reduce system costs and save circuit board space.

- The ’C24x DSP controllers are designed to meet the needs of control-based applications.
- By integrating the high performance of a DSP core and the on-chip peripherals of a microcontroller into a single-chip solution, the ’C24x series yields a device that is an affordable alternative to traditional microcontroller units (MCUs) and expensive multichip designs.
- At 20 million instructions per second (MIPS), the ’C24x DSP controllers offer significant performance over traditional 16-bit microcontrollers and microprocessors.
- The 16-bit, fixed-point DSP core of the ’C24x device provides analog designers a digital solution that does not sacrifice the precision and performance of their systems. The ’C24x DSP controllers offer reliability and programmability. Analog control systems, on the other hand, are hardwired solutions and can experience performance degradation due to aging, component tolerance, and drift.
- The high-speed central processing unit (CPU) allows the digital designer to process algorithms in real time rather than approximate results with look-up tables
- The ’C24x architecture is also well-suited for processing control signals.
- It uses a 16-bit word length along with 32-bit registers for storing intermediate results, and has two hardware shifters available to scale numbers independently of the CPU. This combination minimizes quantization and truncation errors, and increases processing power for additional functions. Two examples of these additional functions are: a notch filter that cancels mechanical resonances in a system, and an estimation technique that eliminates state sensors in a system

TMS320C24x Nomenclature

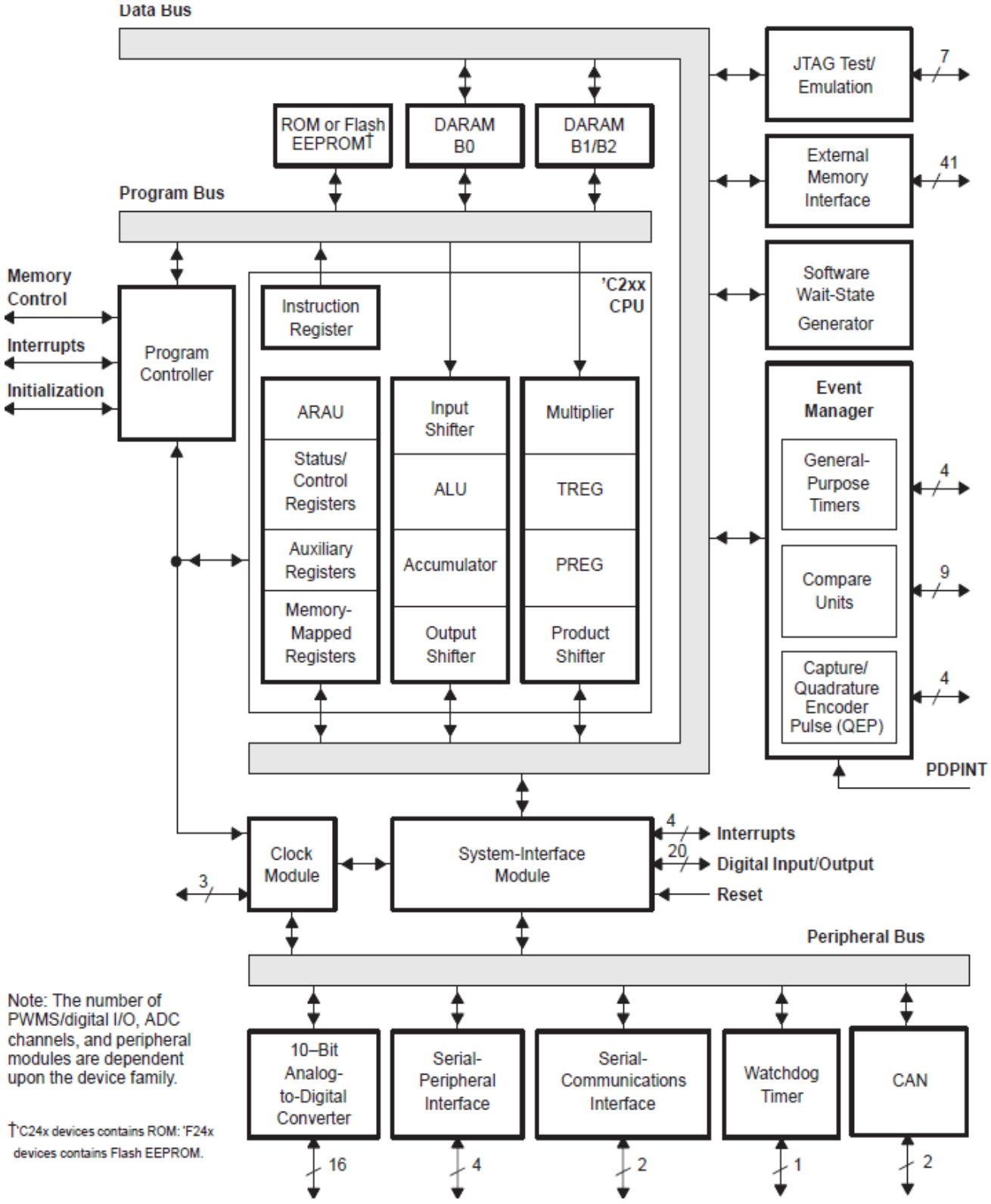
TMS – stands for qualified device

320 - TMS320 Family

C – CMOS technology

24x - device

TMS320C24x DSP Controller Functional Block Diagram



Note: The number of PWMS/digital I/O, ADC channels, and peripheral modules are dependent upon the device family.

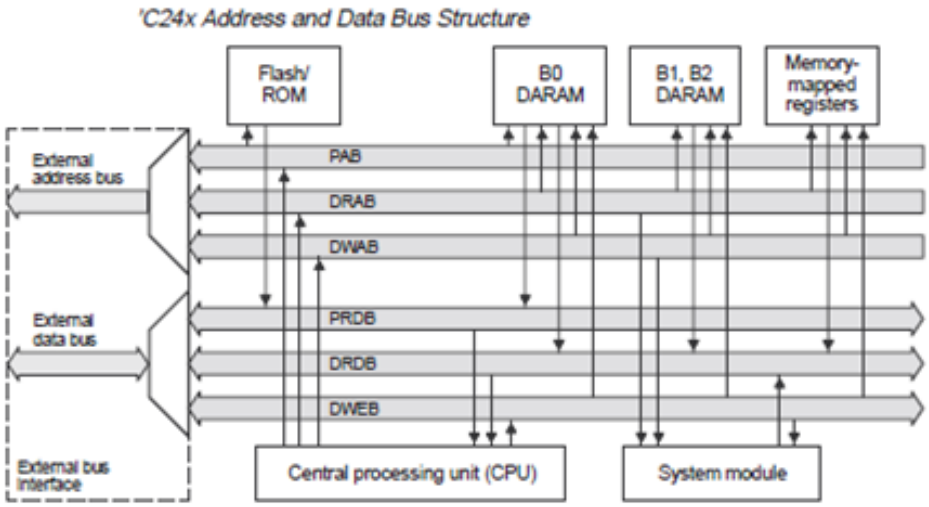
†'C24x devices contains ROM: 'F24x devices contains Flash EEPROM.

C24x CPU Internal Bus Structure

The ’C24x DSP, a member of the TMS320 family of DSPs, includes a ’C2xx DSP core designed using the ’2xLP ASIC core. The ’C2xx DSP core has an internal data and program bus structure that is divided into six 16-bit buses. The six buses are:

- **PAB.** The program address bus provides addresses for both reads from and writes to program memory.
- **DRAB.** The data-read address bus provides addresses for reads from data memory.
- **DWAB.** The data-write address bus provides addresses for writes to data memory.
- **PRDB.** The program read bus carries instruction code and immediate operands, as well as table information, from program memory to the CPU.
- **DRDB.** The data-read bus carries data from data memory to the central arithmetic logic unit (CALU) and the auxiliary register arithmetic unit (ARAU).
- **DWEB.** The data-write bus carries data to both program memory and data memory.

Having separate address buses for data reads (DRAB) and data writes (DWAB) allows the CPU to read and write in the same machine cycle.



Memory

The ’C24x contains the following types of on-chip memory:

- Dual-access RAM (DARAM)
- Flash EEPROM or ROM (masked)

The ’C24x memory is organized into four individually-selectable spaces:

- Program (64K words)
- Local data (64K words)
- Global data (32K words)
- Input/Output (64K words)

These spaces form an address range of 224K words.

On-Chip Dual-Access RAM (DARAM)

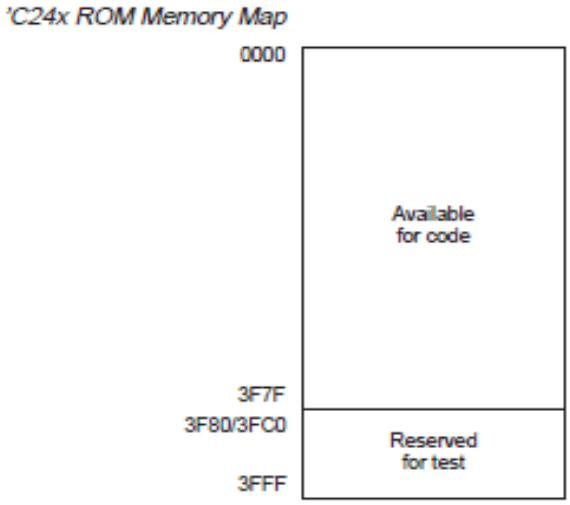
- The ’C24x has 544 words of on-chip DARAM, which can be accessed twice per machine cycle. This memory is primarily intended to hold data, but when needed, can also be used to hold programs.
- The memory can be configured in one of two ways, depending on the state of the CNF bit in status register ST1.
  - → When CNF = 0, all 544 words are configured as data memory.
  - → When CNF = 1, 288 words are configured as data memory and 256 words are configured as program memory.
- Because DARAM can be accessed twice per cycle, it improves the speed of the CPU.
- The CPU operates within a 4-cycle pipeline. In this pipeline, the CPU reads data on the third cycle and writes data on the fourth cycle. However, DARAM allows the CPU to write and read in one cycle; the CPU writes to DARAM on the master phase of the cycle and reads from DARAM on the slave phase. For example, suppose two instructions, A and B, store the accumulator value to DARAM and load the accumulator with a new value from DARAM. Instruction A stores the accumulator value during the master phase of the CPU cycle, and instruction B loads the new value in the accumulator during the slave phase. Because part of the dual-access operation is a write, it only applies to RAM.

Flash EEPROM

- Flash EEPROM provides an attractive alternative to masked program ROM.
- Like ROM, flash is a nonvolatile memory type; however, it has the advantage of in-target reprogrammability.
- The ’F24x incorporates one 16K/8K × 16-bit flash EEPROM module in program space.
- This type of memory expands the capabilities of the ’F24x in the areas of prototyping, early field testing, and single-chip applications.
- Unlike most discrete flash memory, the ’F24x flash does not require a dedicated state machine because the algorithms for programming and erasing the flash are executed by the DSP core. This enables several advantages, including reduced chip size and sophisticated adaptive algorithms..
- Other key features of the flash include zero-wait-state access rate and single 5-V power supply.

The following four algorithms are required for flash operations:

- **clear, erase, flash-write, and program.**



External Memory Interface Module

- In addition to full, on-chip memory support, some of the ’C24x devices provide access to external memory by way of the External Memory Interface Module.
- This interface provides 16 external address lines, 16 external data lines, and relevant control signals to select data, program, and I/O spaces. An on-chip wait-state generator allows interfacing with slower off-chip memory and peripherals.

Central Processing Unit

The ’C24x is based on TI’s ’C2xx CPU. It contains:

- A 32-bit central arithmetic logic unit (CALU)
- A 32-bit accumulator
- Input and output data-scaling shifters for the CALU
- A 16-bit × 16-bit multiplier
- A product-scaling shifter
- Data-address generation logic, which includes eight auxiliary registers and an auxiliary register arithmetic unit (ARAU)
- Program-address generation logic

Central Arithmetic Logic Unit (CALU) and Accumulator

The ’C24x performs 2s-complement arithmetic using the 32-bit CALU. The CALU uses 16-bit words taken from data memory, derived from an immediate instruction, or from the 32-bit multiplier result. In addition to arithmetic operations, the CALU can perform Boolean operations. The accumulator stores the output from the CALU; it can also provide a second input to the CALU. The accumulator is 32 bits wide and is divided into a highorder word (bits 31 through 16) and a low-order word (bits 15 through 0). Assembly language instructions are provided for storing the high- and loworder accumulator words to data memory.

Scaling Shifters

The ’C24x has three 32-bit shifters that allow for scaling, bit extraction, extended arithmetic, and overflow-prevention operations:

 Input data-scaling shifter (input shifter). This shifter left-shifts 16-bit input data by 0 to 16 bits to align the data to the 32-bit input of the CALU.

 Output data-scaling shifter (output shifter). This shifter left-shift output from the accumulator by 0 to 7 bits before the output is stored to data memory. The content of the accumulator remains unchanged.

 Product-scaling shifter (product shifter). The product register (PREG) receives the output of the multiplier. The product shifter shifts the output of the PREG before that output is sent to the input of the CALU. The product shifter has four product shift modes (no shift, left shift by one bit, left shift by four bits, and right shift by six bits), which are useful for performing multiply/accumulate operations, performing fractional arithmetic, or justifying fractional products.

Multiplier

The on-chip multiplier performs 16-bit × 16-bit 2s-complement multiplication with a 32-bit result. In conjunction with the multiplier, the ’C24x uses the 16-bit temporary register (TREG) and the

32-bit product register (PREG); TREG always supplies one of the values to be multiplied, and PREG receives the result of each multiplication. Using the multiplier, TREG, and PREG, the ’C24x efficiently performs fundamental DSP operations such as convolution, correlation, and filtering. The effective execution time of each multiplication instruction can be as short as one CPU cycle.

Auxiliary Register Arithmetic Unit (ARAU) and Auxiliary Registers

The ARAU generates data memory addresses when an instruction uses indirect addressing (see Chapter 6, Addressing Modes) to access data memory. The ARAU is supported by eight auxiliary registers (AR0 through AR7), each of which can be loaded with a 16-bit value from data memory or directly from an instruction word. Each auxiliary register value can also be stored in data memory. The auxiliary registers are referenced by a 3-bit auxiliary register pointer (ARP) embedded in status register ST0.

Program Control

Several hardware and software mechanisms provide program control:

 Program control logic decodes instructions, manages the 4-level pipeline, stores the status of operations, and decodes conditional operations. Hardware elements included in the program control logic are the program counter, the status registers, the stack, and the address-generation logic.

 Software mechanisms used for program control include branches, calls, conditional instructions, a repeat instruction, reset, interrupts, and powerdown modes.

Serial-Scan Emulation

The ’C24x has seven pins dedicated to the serial scan emulation port (JTAG port). This port allows for non-intrusive emulation of ’C24x devices, and is supported by Texas Instruments emulation tools and by many third party debugger tools.

Memory and I/O Spaces

The ’C24x has a 16-bit address line that accesses four individually selectable spaces (224K words total):

 A 64K-word program space

 A 64K-word local data space

 A 32K-word global data space

 A 64K-word I/O space

The ’C24x has multiple memory spaces accessible on three parallel buses: a program address bus (PAB), a data-read address bus (DRAB), and a data-write address bus (DWAB). Each of the three buses access different memory spaces for different phases of the device’s operation. Because the bus operations are independent, it is possible to access both the program and data spaces simultaneously. Within a given machine cycle, the CALU can execute as many as three concurrent memory operations.

The ’C24x address map is organized into four individually selectable spaces:

 Program memory (64K words) contains the instructions to be executed, as well as data used during program execution.

 Data memory (64K words) holds data used by the instructions.

 Global data memory (32K words) shares data with other devices or serves as additional data space.

 Input/output (I/O) space (64K words) interfaces to external peripherals and may contain on-chip registers.

These spaces provide a total address space of 224K words. The ’C24x includes on-chip memory to aid in system performance and integration, and numerous addresses that can be used for external memory and I/O devices.

The advantages of operating from on-chip memory are:

 Higher performance than external memory (because the wait states required for slower external memories are avoided)

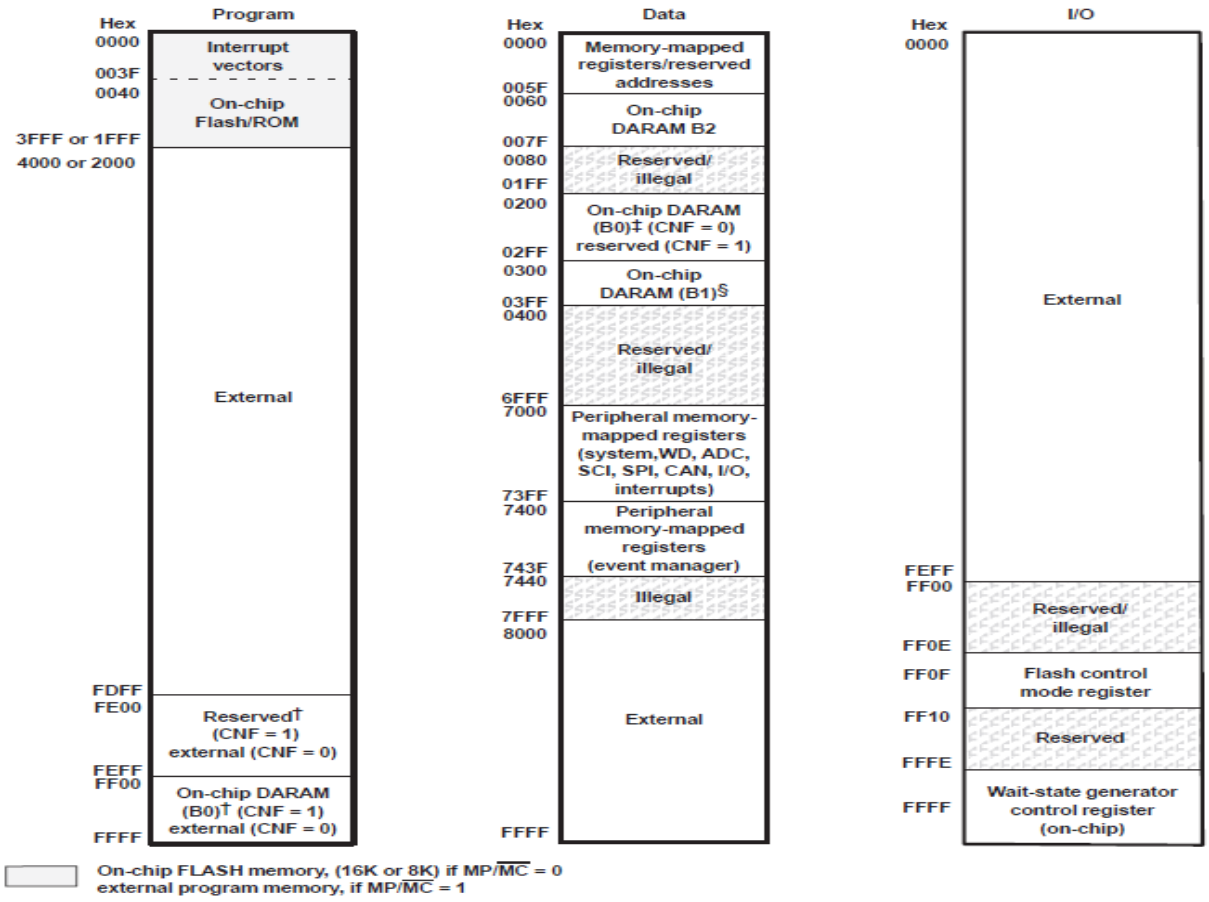
 Lower cost than external memory

 Lower power consumption than external memory

The advantage of operating from external memory is the ability to access a larger address space.

The memory maps shown in Figure is generic for all ’C24x devices; however, each device has its own set of memory maps. ’C24x devices are available with different combinations of on-chip memory and peripherals.

Figure . Generic Memory Maps for ’C24x DSP Controllers



† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

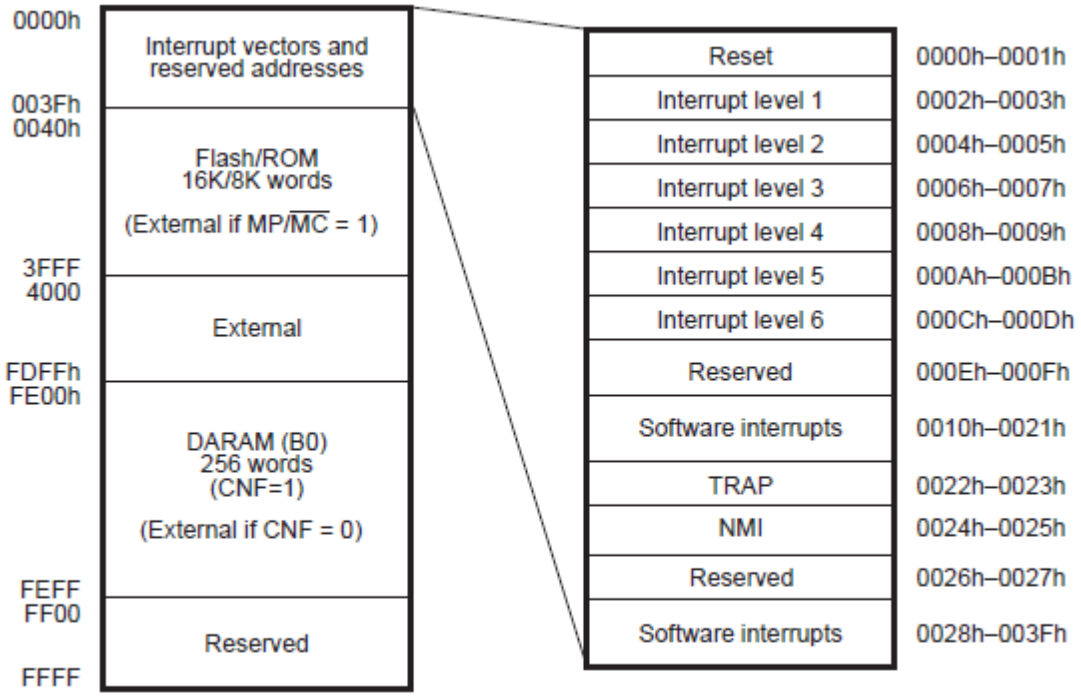
‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

Program Memory

The program-memory space is where the application program code resides; it can also hold table information and immediate operands. The program memory space addresses up to 64K 16-bit words. On the ’C24x device, these words include on-chip DARAM and on-chip ROM/flash EEPROM. When the ’C24x generates an address outside the set of addresses configured to onchip program memory, the device automatically generates an external access, asserting the appropriate

control signals (if an external memory interface is present). Figure 3–2 shows the ’C24x program memory map.



Note: Flash/ROM memory includes the address range 0000h–003Fh.

Figure. Program Memory Map for ’C24x

Program Memory Configuration

Depending on which types of memory are included in a particular ’C24x device, two factors contribute to the configuration of program memory:

 CNF bit. The CNF bit (bit 12) of status register ST1 determines whether the addresses for DARAM B0 are available for program space:

_ **CNF = 0.** There is no addressable on-chip program DARAM.

_ **CNF = 1.** The 256 words of DARAM B0 are configured for program use. At reset, any words of program/data DARAM are mapped into local data space (CNF = 0).

 MP/MC pin. The level on the MP/MC pin determines whether program instructions are read from on-chip ROM or flash EEPROM (if available) after reset:

_ **MP/MC = 0.** The device is configured as a microcomputer. The onchip ROM/flash EEPROM is accessible. The device fetches the reset vector from on-chip memory.

_ **MP/MC = 1.** The device is configured as a microprocessor. The device fetches the reset vector from external memory. Regardless of the value of MP/MC, the ’C24x fetches its reset vector at location 0000h of program memory.

Data Memory

Data-memory space addresses up to 64K 16-bit words. Each ’C24x device has three on-chip DARAM blocks: B0, B1, and B2. Block B0 is configurable as either data memory or program memory. Blocks B1 and B2 are available for data memory only. Data memory can be addressed with either of two addressing modes: direct addressing or indirect-addressing.

When direct addressing is used, data memory is addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 data pages labeled 0 through 511. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register ST0. Each of the 128 words on the current page is referenced by a 7-bit offset, which is taken from the instruction that is using direct addressing. Therefore, when an instruction uses direct addressing, you must specify both the data page (with a preceding instruction) and the offset (in the instruction that accesses data memory).

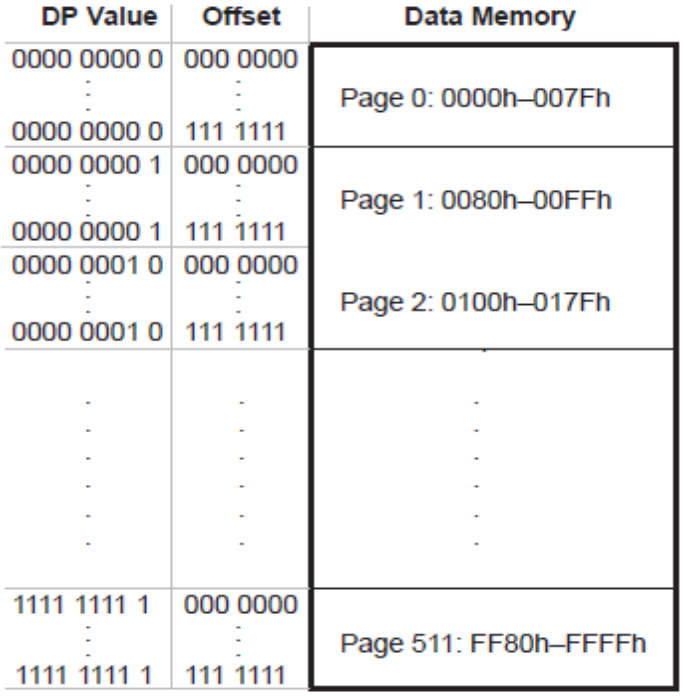


Figure. Pages of Data Memory

Data Page 0 Address Map

The data memory also includes the device’s memory-mapped registers (MMR), which reside at the top of data page 0 (addresses 0000h–007Fh).

Address	Name	Description
0000h–0003h	–	Reserved
0004h	IMR	Interrupt mask register
0005h	GREG	Global memory allocation register
0006h	IFR	Interrupt flag register
0023h–0027h	–	Reserved
002Bh–002Fh	–	Reserved for test/emulation
0060h–007Fh	B2	Scratch-pad RAM (DARAM B2)

Data Memory Configuration

The following contributes to the configuration of data memory:

 CNF bit. The CNF bit (bit 12) of status register ST1 determines whether the on-chip DARAM B0 is mapped into local data space or into program space.

_ **CNF = 1.** DARAM B0 is used for program space.

_ **CNF = 0.** B0 is used for data space. At reset, B0 is mapped into local data space (CNF = 0).

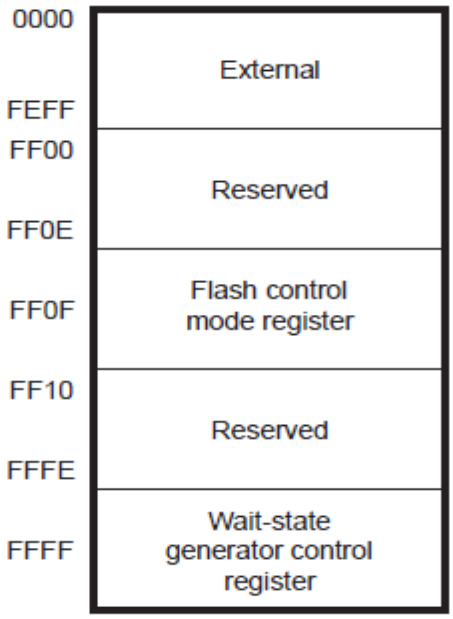
Global Data Memory

Addresses in the upper 32K words (8000h–FFFFh) of local data memory can be used for global data memory. The global memory allocation register (GREG) determines the size of the global data-memory space, which is between 256 and 32K words. The GREG is connected to the eight LSBs of the internal data bus and is memory-mapped to data-memory location 0005h. Any remaining addresses within 8000h–FFFFh are available for local data memory.

I/O Space

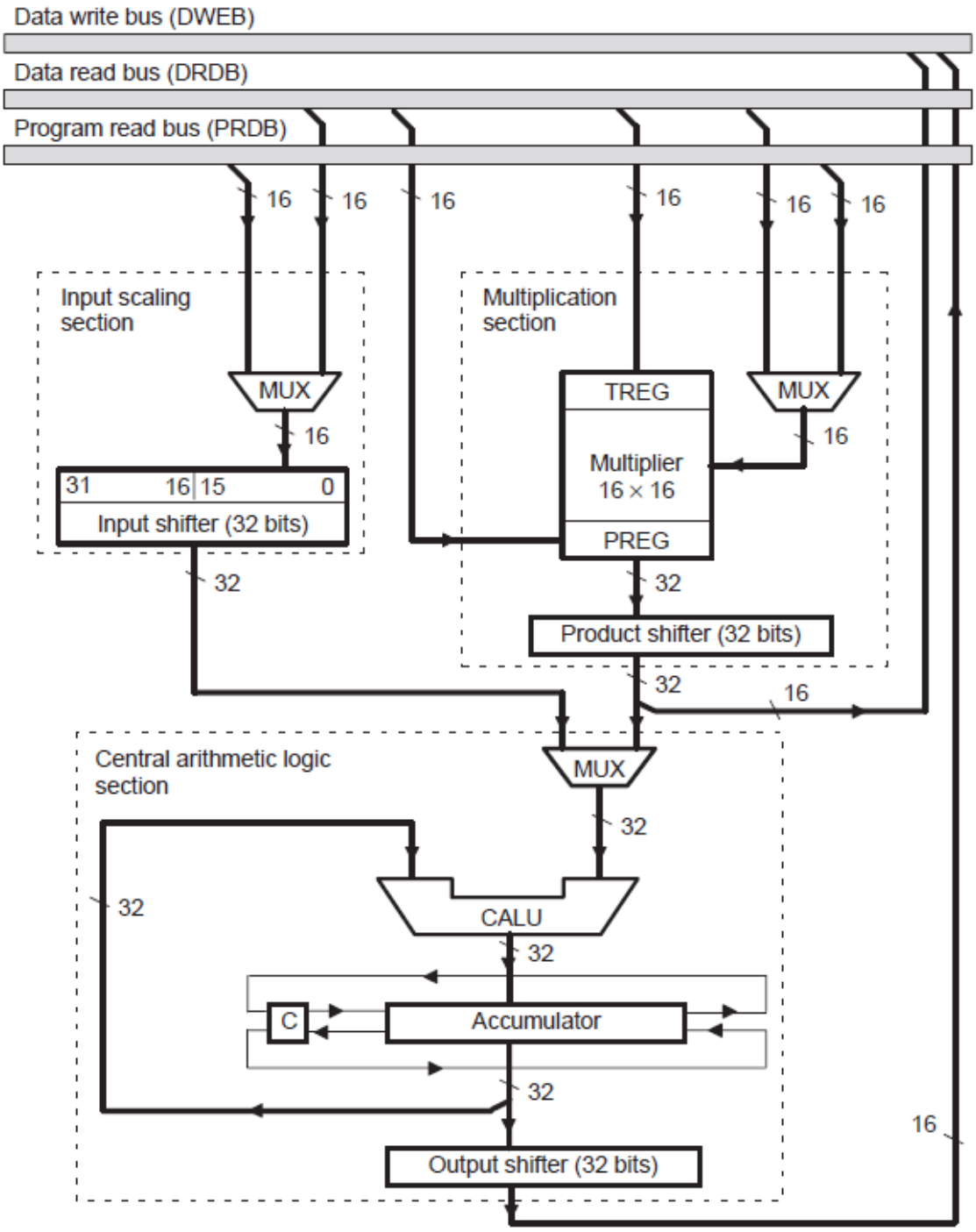
The I/O space memory addresses up to 64K 16-bit words. Figure 3–6 shows the I/O-space address map for the ’C24x.

I/O-Space Address Map for ’C24x



The I/O space is useful for mapping external peripherals and flash control registers. This I/O space is a generic space available for the ’C24x core. Depending on the specific device within the ’C24x family, the I/O space is partially available or disabled. You should refer to the specific data sheet for exact details. External I/O space is available only in ’24x devices that have an external memory interface; otherwise, this space is reserved.

CPU



Input Scaling Section

A 32-bit input data-scaling shifter (input shifter) aligns the 16-bit value from memory to the 32-bit central arithmetic logic unit (CALU). This data alignment is necessary for data-scaling arithmetic, as well as aligning masks for logical operations. The input shifter operates as part of the data path between program or data space and the CALU; and therefore, requires no cycle overhead.

Input. Bits 15 through 0 of the input shifter accept a 16-bit input from either of two

 The data read bus (DRDB). This input is a value from a data memory location referenced in an instruction operand.

 The program read bus (PRDB). This input is a constant value given as an instruction operand.

Output. After a value has been accepted into bits 15 through 0, the input shifter aligns the16-bit value to the 32-bit bus of the CALU

The shifter shifts the value left 0 to 16 bits and then sends the 32-bit result to the CALU.

Multiplication Section

The ’C24x uses a 16-bit × 16-bit hardware multiplier that can produce a signed or unsigned 32-bit product in a single machine cycle.

The multiplication section consists of:

 The 16-bit temporary register (TREG), which holds one of the multiplicands The multiplier, which multiplies the TREG value by a second value from data memory or program memory

 The 32-bit product register (PREG), which receives the result of the multiplication

 The product shifter, which scales the PREG value before passing it to the CALU

Multiplier

The 16-bit × 16-bit hardware multiplier can produce a signed or unsigned 32-bit product in a single machine cycle. The two numbers being multiplied are treated as 2s-complement numbers, except during unsigned multiplication (MPYU instruction). Descriptions of the inputs to, and output of, the multiplier

Inputs. The multiplier accepts two 16-bit inputs:

 One input is always from the 16-bit temporary register (TREG). The TREG is loaded before the multiplication with a data-value from the data read bus (DRDB).

 The other input is one of the following:

_ A data-memory value from the data read bus (DRDB)

_ A program memory value from the program read bus (PRDB)

Output. After the two 16-bit inputs are multiplied, the 32-bit result is stored in the product register (PREG). The output of the PREG is connected to the 32-bit product-scaling shifter. Through this shifter, the product is transferred from the PREG to the CALU or to data memory (by the SPH and SPL instructions).

Product-Scaling Shifter

The product-scaling shifter (product shifter) facilitates scaling of the product register (PREG) value. The shifter has a 32-bit input connected to the output of the PREG and a 32-bit output connected to the input of the CALU.

Input. The shifter has a 32-bit input connected to the output of the PREG.

Output. After the shifter completes the shift, all 32 bits of the result can be passed to the CALU, or 16 bits of the result can be stored to data memory.

Shift Modes. This shifter uses one of four product shift modes , these modes are determined by the product shift mode (PM) bits of status register ST1.

In the first shift mode (PM = 00), the shifter does not shift the product at all before giving it to the CALU or to data memory. The next two modes cause left shifts (of one or four), which are useful for implementing fractional arithmetic or justifying products. The right-shift mode shifts the product by six bits, enabling the execution of up to 128 consecutive multiply-and-accumulate operations without causing the accumulator to overflow. Note that the content of the PREG remains unchanged; the value is copied to the product shifter and shifted there.

Central Arithmetic Logic Section

The main components of the central arithmetic logic section are

 The central arithmetic logic unit (CALU), which implements a wide range of arithmetic and logic functions

 The 32-bit accumulator (ACC), which receives the output of the CALU and is capable of performing bit shifts on its contents with the help of the carry bit (C). Figure 4–6 shows the accumulator's high word (ACCH) and low word (ACCL).

 The output shifter, which can shift a copy of either the high word or low word of the accumulator before sending it to data memory for storage

Central Arithmetic Logic Unit (CALU)

The CALU implements a wide range of arithmetic and logic functions, most of which execute in a single clock cycle. These functions can be grouped into four categories:

 16-bit addition

 16-bit subtraction

 Boolean logic operations

 Bit testing, shifting, and rotating

Because the CALU can perform Boolean operations, you can perform bit manipulation. For bit shifting and rotating, the CALU uses the accumulator. The CALU is referred to as central because there is an independent arithmetic unit, the auxiliary register arithmetic unit (ARAU),

Inputs. The CALU has two inputs

 One input is always provided by the 32-bit accumulator.

 The other input is provided by one of the following:

_ The product-scaling shifter (see section 4.2.2)

_ The input data-scaling shifter (see Section 4.1)

Output. Once the CALU performs an operation, it transfers the result to the 32-bit accumulator, which is capable of performing bit shifts of its contents. The output of the accumulator is connected to the 32-bit output data-scaling shifter. Through the output shifter, the accumulator's upper and lower 16-bit words can be individually shifted and stored to data memory.

Sign-extension mode bit. For many but not all instructions, the sign-extension mode bit (SXM), bit 10 of status register ST1, determines whether the CALU uses sign extension during its calculations. If SXM = 0, sign extension is suppressed. If SXM = 1, sign extension is enabled.

Accumulator

Once the CALU performs an operation, it transfers the result to the 32-bit accumulator, which can then perform single-bit shifts or rotations on its contents. Each of the accumulator's upper and lower 16-bit words can be passed to the output data-scaling shifter, where it can be shifted and then stored in data memory. The following describes the status bits and branch instructions associated with the accumulator

Carry bit (C). C (bit 9 of status register ST1) is affected during:_ Additions to and subtractions from the accumulator:

C = 0 When the result of a subtraction generates a borrow

C = 1 When the result of an addition generates a carry

 Overflow mode bit (OVM). OVM (bit 11 of status register ST0) determines how the accumulator reflects arithmetic overflows.

 Overflow flag bit (OV). OV is bit 12 of status register ST0. When no accumulator overflow is detected, OV is latched at 0

Test/control flag bit (TC). TC (bit 11 of status register ST1) is set to 0 or 1 depending on the value of a tested bit.

Output Data-Scaling Shifter

The output data-scaling shifter (output shifter) has a 32-bit input connected to the 32-bit output of the accumulator and a 16-bit output connected to the data bus. The shifter copies all 32 bits of the accumulator and then performs a left shift on its content; it can be shifted from zero to seven bits, as specified in the corresponding store instruction. The upper word (SACH instruction) or lower word (SACL instruction) of the shifter is then stored to data memory. The content of the accumulator remains unchanged.

Auxiliary Register Arithmetic Unit (ARAU)

The CPU also contains the ARAU, an arithmetic unit independent of the CALU. The main function of the ARAU is to perform arithmetic operations on eight auxiliary registers (AR7 through AR0) in parallel with operations occurring in the CALU. Figure shows the ARAU and related logic.

ARAU Functions

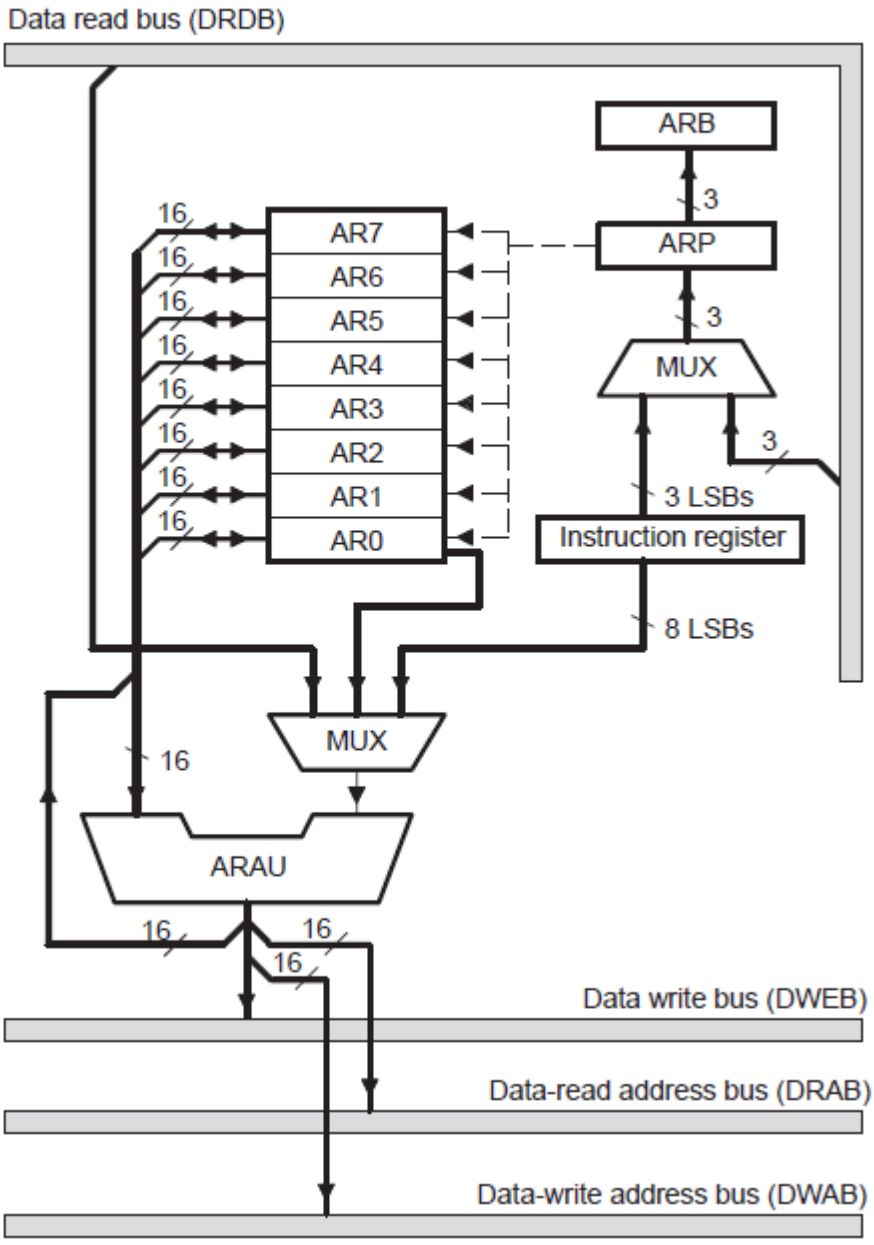
The ARAU performs the following operations:

 Increments or decrements an auxiliary register value by 1 or by an index amount (by way of any instruction that supports indirect addressing)

 Adds a constant value to an auxiliary register value (ADRK instruction) or subtracts a constant value from an auxiliary register value (SBRK instruction). The constant is an 8-bit value taken from the eight LSBs of the instruction word.

 Compares the content of AR0 with the content of the current AR and puts the result in the test/control flag bit (TC) of status register ST1 (CMPR instruction). The result is passed to TC by way of the data write bus (DWEB).

ARAU and Related Logic



Auxiliary Register Functions

In addition to using the auxiliary registers to reference data-memory addresses,

you can use them for other purposes.

 Use the auxiliary registers to support conditional branches, calls, and returns by using the CMPR instruction. This instruction compares the content of AR0 with the content of the current AR and puts the result in the test/control flag bit (TC) of status register ST1.

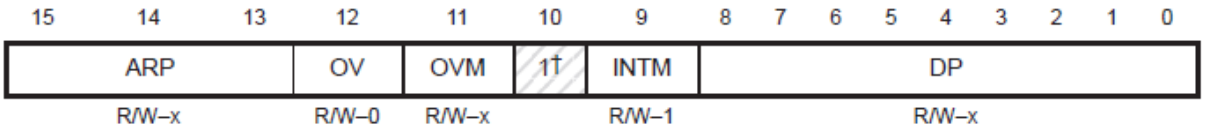
 Use the auxiliary registers for temporary storage by using the LAR instruction to load values into the registers and the SAR instruction to store AR values to data memory

 Use the auxiliary registers as software counters, incrementing or decrementing them as necessary

Status Registers ST0 and ST1

The ’C24x has two status registers, ST0 and ST1, which contain status and control bits. These registers can be stored to, and loaded from, data memory. This allows the status of the machine to be saved and restored for subroutines. The LST (load status register) instruction writes to ST0 and ST1, and the SST (store status register) instruction reads from ST0 and ST1 (with the exception of the INTM bit, which is not affected by the LST instruction). Many of the individual
bits of these registers can be set and cleared using the SETC and CLRC instructions. For example, the sign-extension mode is set with SETC SXM and cleared with CLRC SXM.

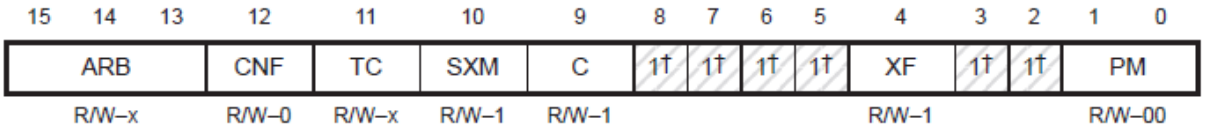
Figure Status Register ST0



Note: R = Read access; W = Write access; value following dash (–) is value after reset (x means value not affected by reset).

† This reserved bit is always read as 1. Writes have no effect on it.

Figure Status Register ST1



Note: R = Read access; W = Write access; value following dash (–) is value after reset (x means value not affected by reset).

† These reserved bits are always read as 1s. Writes have no effect on them.

External Memory Interface Operation

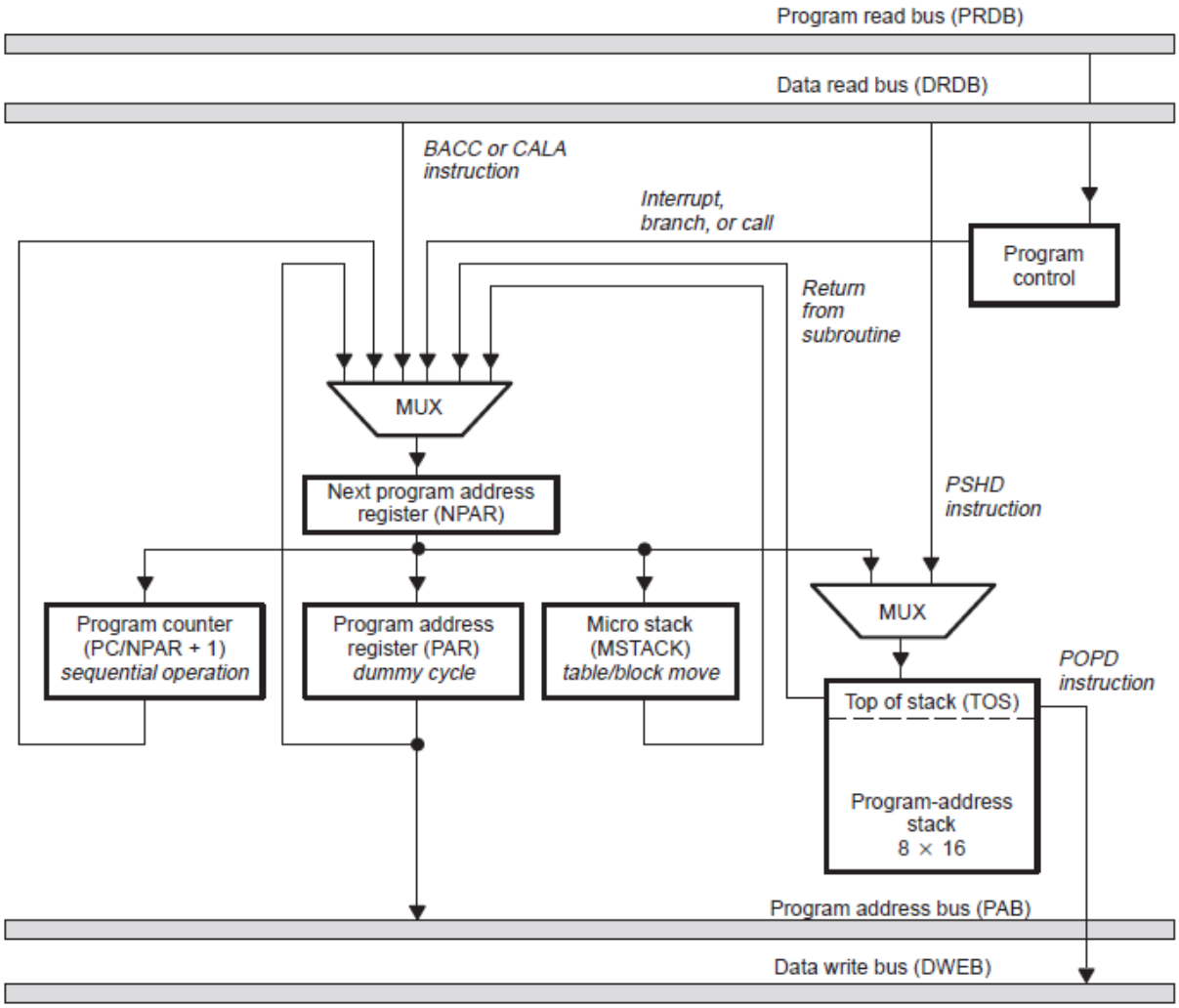
Some of the ’24x DSP controller devices have an external memory interface. This section explains the behavior of the ’24x external memory timings based on the PS, DS, IS, BR, STRB, RD, LR, and R/W signals. All bus cycles comprise integral numbers of CLKOUT cycles. One CLKOUT cycle is defined to be from one falling edge of CLKOUT to the next falling edge of CLKOUT. For full-speed, 0-wait-state operation, reads require one cycle. A write immediately preceded by a read or immediately followed by a read requires three cycles

Program Control

Program control involves controlling the order in which one or more blocks of instructions are executed. Normally, the flow of a program is sequential; the ’C24x executes instructions at consecutive program-memory addresses.

Program-Address Generation

Program flow requires the processor to generate the next program address (sequential or nonsequential) while executing the current instruction. Program- address generation is illustrated in Figure



The ’C24x program-address generation logic uses the following hardware:

 Program counter (PC). The ’C24x has a 16-bit program counter (PC) that addresses internal and external program memory when fetching instructions.

 Program address register (PAR). The PAR drives the program address bus (PAB). The PAB is a 16-bit bus that provides program addresses for both reads and writes.

 Stack. The program-address generation logic includes a 16-bit-wide, 8-level hardware stack for storing up to eight return addresses. In addition, you can use the stack for temporary storage.

 Microstack (MSTACK). Occasionally, the program-address generation logic uses the 16-bit-wide, 1-level MSTACK to store one return address.

 Repeat counter (RPTC). The 16-bit RPTC is used with the repeat (RPT) instruction to determine how many times the instruction following RPT is repeated.

Program Counter (PC)

The program-address generation logic uses the 16-bit program counter (PC) to address internal and external program memory. The PC holds the address of the next instruction to be executed. Through the program address bus (PAB), an instruction is fetched from that address in program memory and loaded into the instruction register. When the instruction register is loaded, the PC holds the next address.

Stack

The ’C24x has a 16-bit-wide, 8-level-deep hardware stack. The program-address generation logic uses the stack for storing return addresses when a subroutine call or interrupt occurs. When an instruction forces the CPU into a subroutine or an interrupt forces the CPU into an interrupt service routine, the return address is loaded to the top of the stack automatically, without the need for additional cycles. When the subroutine or interrupt service routine is com plete, a return instruction transfers the return address from the top of the stack to the program counter. When the eight levels are not used for return addresses, the stack may be used for saving context data during a subroutine or interrupt service routine or for other storage purposes You can access the stack with two sets of instructions:

 PUSH and POP. The PUSH instruction copies the 16 LSBs of the accumulator to the top of the stack. The POP instruction copies the value on the top of the stack to the 16 LSBs of the accumulator.

 PSHD and POPD. These instructions allow you to build a stack in data memory for the nesting of subroutines or interrupts beyond eight levels. The PSHD instruction pushes a data-memory value onto the top of the stack. The POPD instruction pops a value from the top of the stack to data memory

Pipeline Operation

Instruction pipelining consists of a sequence of bus operations that occur during the execution of an instruction. The ’C24x pipeline has four independent stages: instruction-fetch, instruction-

decode, operand-fetch, and instruction execute.Because the four stages are independent, these operations can overlap. During any given cycle, one to four different instructions can be active, each at a different stage of completion.

The pipeline is essentially invisible to you, except in the following cases:

 A single-word, single-cycle instruction immediately following a modification of the global-memory allocation register (GREG) uses the previous global map.

 The NORM instruction modifies the auxiliary register pointer (ARP) and uses the current auxiliary register (the one pointed to by the ARP) during the execute phase of the pipeline. If the next two instruction words change the values in the current auxiliary register or the ARP, they will do so during the instruction decode phase of the pipeline (before the execution of NORM). This would cause NORM to use the wrong auxiliary register value and the following instructions to use the wrong ARP value.

Branches, Calls, and Returns

The ’C24x has two types of branches, calls, and returns:

 Unconditional. An unconditional branch, call, or return is always Executed

Conditional. A conditional branch, call, or return is executed only if certain specified conditions are met

Interrupts

The ’C24x DSP supports both hardware and software interrupts. The hardware interrupts INT1 – INT6, along with NMI, TRAP, and RS, provide a flexible interrrupt scheme. The software interrupts offer flexibility to access interrupt vectors using software instructions. Since most of the ’C24x DSPs come with multiple peripherals, the core interrupts (INT1–IN6) are expanded using additional system or peripheral interrupt logic. Although the core interrupts are the same, the peripheral interrupt structure varies slightly among ’C240 and ’C24x class of DSP controllers.

CPU Interrupt Registers

There are two CPU registers for controlling interrupts:

 The interrupt flag register (IFR) contains flag bits that indicate when maskable interrupt requests have reached the CPU on levels INT1 through INT6.

 The interrupt mask register (IMR) contains mask bits that enable or disable each of the interrupt levels (INT1 through INT6).

Addressing Modes

The three modes are:

 Immediate addressing mode

 Direct addressing mode

 Indirect addressing mode

Immediate Addressing Mode

In the immediate addressing mode, the instruction word contains a constant to be manipulated by the instruction. The two types of immediate addressing modes are:

 Short-immediate addressing Instructions that use short-immediate addressing have an 8-bit, 9-bit, or 13-bit constant as an operand

Eg: RPT #99 ;Execute the instruction that follows RPT ;100 times.

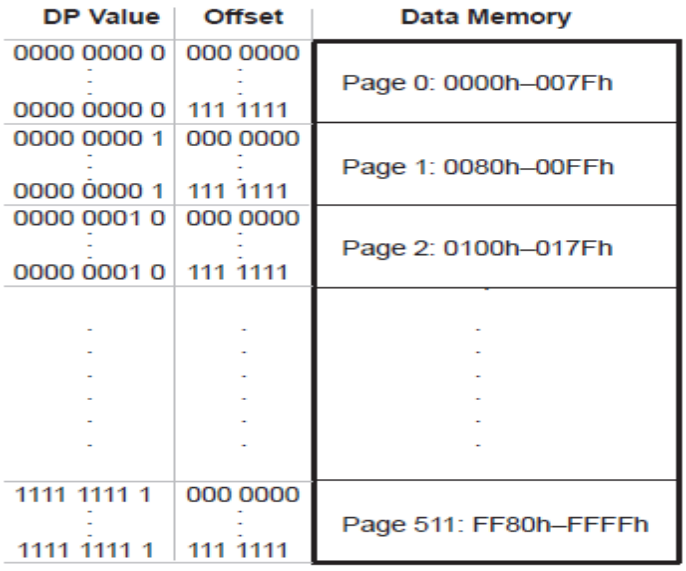
Long-immediate addressing. Instructions that use long-immediate addressing have a 16-bit constant as an operand and require two instruction words

ADD #16384,2 ;Shift the value 16384 left by two bits ;and add the result to the accumulator.

Direct Addressing Mode

In the direct addressing mode, data memory is addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 data pages labeled 0 through 511. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register

ST0. For example, if the DP value is 0 0000 00002, the current data page is If the DP value is 0 0000 00102, the current data page is 2.



Indirect Addressing Mode

Eight auxiliary registers (AR0–AR7) provide flexible and powerful indirect addressing. Any location in the 64K data memory space can be accessed using a 16-bit address contained in an auxiliary register.

Indirect Addressing Options

The ’C24x provides four types of indirect addressing options:

 No increment or decrement. The instruction uses the content of the current auxiliary register as the data memory address but neither increments nor decrements the content of the current auxiliary register.

 Increment or decrement by 1. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by one.

 Increment or decrement by an index amount. The value in AR0 is the index amount. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by the index amount.

. The addition and subtraction process is accomplished with the carry propagation reversed for fast Fourier transforms (FFTs).

Assembly Language Instructions

Instruction Set Summary

This section provides six tables (Table 7–1 to Table 7–6) that summarize the instruction set according to the following functional headings:

 Accumulator, arithmetic, and logic instructions

 Auxiliary register and data page pointer instructions (see Table 7–2 on page 7-7)

 TREG, PREG, and multiply instructions (see Table 7–3 on page 7-8)

 Branch instructions (see Table 7–4 on page 7-9)

 Control instructions (see Table 7–5 on page 7-10)

 I/O and memory operations (see Table 7–6 on page 7-11)

definitions of the symbols used in the six summary tables:

ACC The accumulator

AR The auxiliary register

ARX A 3-bit value used in the LAR and SAR instructions to designate

which auxiliary register will be loaded (LAR) or have its contents stored (SAR)

BITX A 4-bit value (called the bit code) that determines which bit of a designated data memory value will be tested by the BIT instruction

CM A 2-bit value. The CMPR instruction performs a comparison specified by the value of CM:

If CM = 00, test whether current AR = AR0

If CM = 01, test whether current AR < AR0

If CM = 10, test whether current AR > AR0

If CM = 11, test whether current AR p AR0

IAAA AAAA (One I followed by seven As) The I at the left represents a bit that reflects whether direct addressing (I = 0) or indirect addressing (I = 1) is being used. When direct addressing is used, the seven As are the seven least significant bits (LSBs) of a data memory address. For indirect addressing, the seven As are bits that control auxiliary register manipulation (see Section 6.3, Indirect Addressing Mode, on page 6-9).

IIII IIII (Eight Is) An 8-bit constant used in short immediate addressing

I IIII IIII (Nine Is) A 9-bit constant used in short immediate addressing for the LDP instruction

I IIII IIII IIII (Thirteen Is) A 13-bit constant used in short immediate addressing for the MPY instruction

I NTR# A 5-bit value representing a number from 0 to 31. The INTR instruction uses this number to change program control to one of the 32 interrupt vector addresses.

PM A 2-bit value copied into the PM bits of status register ST1 by the SPM instruction

SHF A 3-bit left-shift value

SHFT A 4-bit left-shift value

TP A 2-bit value used by the conditional execution instructions to represent four conditions:

BIO pin low TP = 00

TC bit =1 TP = 01

TC bit = 0 TP = 10

No condition TP = 11

Table 7–1. Accumulator, Arithmetic, and Logic Instructions

Mnemonic	Description	Words	Cycles	Opcode
ABS	Absolute value of ACC	1	1	1011 1110 0000 0000
ADD	Add to ACC with shift of 0 to 15, direct or indirect	1	1	0010 SHFT IAAA AAAA
	Add to ACC with shift 0 to 15, long immediate	2	2	1011 1111 1001 SHFT + 1 word
	Add to ACC with shift of 16, direct or indirect	1	1	0110 0001 IAAA AAAA
	Add to ACC, short immediate	1	1	1011 1000 IIII IIII
ADDC	Add to ACC with carry, direct or indirect	1	1	0110 0000 IAAA AAAA
ADDS	Add to low ACC with sign-extension suppressed, direct or indirect	1	1	0110 0010 IAAA AAAA
ADDT	Add to ACC with shift (0 to 15) specified by TREG, direct or indirect	1	1	0110 0011 IAAA AAAA
AND	AND ACC with data value, direct or indirect	1	1	0110 1110 IAAA AAAA
	AND with ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1011 SHFT + 1 word
	AND with ACC with shift of 16, long immediate	2	2	1011 1110 1000 0001 + 1 word
CMPL	Complement ACC	1	1	1011 1110 0000 0001
LACC	Load ACC with shift of 0 to 15, direct or indirect	1	1	0001 SHFT IAAA AAAA
	Load ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1000 SHFT + 1 word
	Load ACC with shift of 16, direct or indirect	1	1	0110 1010 IAAA AAAA
LACL	Load low word of ACC, direct or indirect	1	1	0110 1001 IAAA AAAA
	Load low word of ACC, short immediate	1	1	1011 1001 IIII IIII
LACT	Load ACC with shift (0 to 15) specified by TREG, direct or indirect	1	1	0110 1011 IAAA AAAA
NEG	Negate ACC	1	1	1011 1110 0000 0010
NORM	Normalize the contents of ACC, indirect	1	1	1010 0000 IAAA AAAA

Table 7–1. Accumulator, Arithmetic, and Logic Instructions (Continued)

Mnemonic	Description	Words	Cycles	Opcode
OR	OR ACC with data value, direct or indirect	1	1	0110 1101 IAAA AAAA
	OR with ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1100 SHFT + 1 word
	OR with ACC with shift of 16, long immediate	2	2	1011 1110 1000 0010 + 1 word
ROL	Rotate ACC left	1	1	1011 1110 0000 1100
ROR	Rotate ACC right	1	1	1011 1110 0000 1101
SACH	Store high ACC with shift of 0 to 7, direct or indirect	1	1	1001 1SHF IAAA AAAA
SACL	Store low ACC with shift of 0 to 7, direct or indirect	1	1	1001 0SHF IAAA AAAA
SFL	Shift ACC left	1	1	1011 1110 0000 1001
SFR	Shift ACC right	1	1	1011 1110 0000 1010
SUB	Subtract from ACC with shift of 0 to 15, direct or indirect	1	1	0011 SHFT IAAA AAAA
	Subtract from ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1010 SHFT + 1 word
	Subtract from ACC with shift of 16, direct or indirect	1	1	0110 0101 IAAA AAAA
	Subtract from ACC, short immediate	1	1	1011 1010 IIII IIII
SUBB	Subtract from ACC with borrow, direct or indirect	1	1	0110 0100 IAAA AAAA
SUBC	Conditional subtract, direct or indirect	1	1	0000 1010 IAAA AAAA
SUBS	Subtract from ACC with sign-extension suppressed, direct or indirect	1	1	0110 0110 IAAA AAAA
SUBT	Subtract from ACC with shift (0 to 15) specified by TREG, direct or indirect	1	1	0110 0111 IAAA AAAA

Table 7–1. Accumulator, Arithmetic, and Logic Instructions (Continued)

Mnemonic	Description	Words	Cycles	Opcode
XOR	Exclusive OR ACC with data value, direct or indirect	1	1	0110 1100 IAAA AAAA
	Exclusive OR with ACC with shift of 0 to 15, long immediate	2	2	1011 1111 1101 SHFT + 1 word
	Exclusive OR with ACC with shift of 16, long immediate	2	2	1011 1110 1000 0011 + 1 word
ZALR	Zero low ACC and load high ACC with rounding, direct or indirect	1	1	0110 1000 IAAA AAAA

Table 7–2. Auxiliary Register Instructions

Mnemonic	Description	Words	Cycles	Opcode
ADRK	Add constant to current AR, short immediate	1	1	0111 1000 IIII IIII
BANZ	Branch on current AR not 0, indirect	2	4 (condition true) 2 (condition false)	0111 1011 1AAA AAAA + 1 word
CMPR	Compare current AR with AR0	1	1	1011 1111 0100 01CM
LAR	Load specified AR from specified data location, direct or indirect	1	2	0000 0ARX IAAA AAAA
	Load specified AR with constant, short immediate	1	2	1011 0ARX IIII IIII
	Load specified AR with constant, long immediate	2	2	1011 1111 0000 1ARX + 1 word
MAR	Modify current AR and/or ARP, indirect (performs no operation when direct)	1	1	1000 1011 IAAA AAAA
SAR	Store specified AR to specified data location, direct or indirect	1	1	1000 0ARX IAAA AAAA
SBRK	Subtract constant from current AR, short immediate	1	1	0111 1100 IIII IIII

Table 7–3. TREG, PREG, and Multiply Instructions

Mnemonic	Description	Words	Cycles	Opcode
APAC	Add PREG to ACC	1	1	1011 1110 0000 0100
LPH	Load high PREG, direct or indirect	1	1	0111 0101 IAAA AAAA
LT	Load TREG, direct or indirect	1	1	0111 0011 IAAA AAAA
LTA	Load TREG and accumulate previous product, direct or indirect	1	1	0111 0000 IAAA AAAA
LTD	Load TREG, accumulate previous product, and move data, direct or indirect	1	1	0111 0010 IAAA AAAA
LTP	Load TREG and store PREG in accumulator, direct or indirect	1	1	0111 0001 IAAA AAAA
LTS	Load TREG and subtract previous product, direct or indirect	1	1	0111 0100 IAAA AAAA
MAC	Multiply and accumulate, direct or indirect	2	3	1010 0010 IAAA AAAA + 1 word
MACD	Multiply and accumulate with data move, direct or indirect	2	3	1010 0011 IAAA AAAA + 1 word
MPY	Multiply TREG by data value, direct or indirect	1	1	0101 0100 IAAA AAAA
	Multiply TREG by 13-bit constant, short immediate	1	1	110I IIII IIII IIII
MPYA	Multiply and accumulate previous product, direct or indirect	1	1	0101 0000 IAAA AAAA
MPYS	Multiply and subtract previous product, direct or indirect	1	1	0101 0001 IAAA AAAA
MPYU	Multiply unsigned, direct or indirect	1	1	0101 0101 IAAA AAAA
PAC	Load ACC with PREG	1	1	1011 1110 0000 0011
SPAC	Subtract PREG from ACC	1	1	1011 1110 0000 0101
SPH	Store high PREG, direct or indirect	1	1	1000 1101 IAAA AAAA
SPL	Store low PREG, direct or indirect	1	1	1000 1100 IAAA AAAA
SPM	Set product shift mode	1	1	1011 1111 0000 00PM
SQRA	Square and accumulate previous product, direct or indirect	1	1	0101 0010 IAAA AAAA
SQRS	Square and subtract previous product, direct or indirect	1	1	0101 0011 IAAA AAAA

Table 7–4. Branch Instructions

Mnemonic	Description	Words	Cycles	Opcode
B	Branch unconditionally, indirect	2	4	0111 1001 1AAA AAAA + 1 word
BACC	Branch to address specified by ACC	1	4	1011 1110 0010 0000
BANZ	Branch on current AR not 0, indirect	2	4 (condition true) 2 (condition false)	0111 1011 1AAA AAAA + 1 word
BCND	Branch conditionally	2	4 (conditions true) 2 (any condition false)	1110 00TP ZLVC ZLVC + 1 word
CALA	Call subroutine at location specified by ACC	1	4	1011 1110 0011 0000
CALL	Call subroutine, indirect	2	4	0111 1010 1AAA AAAA + 1 word
CC	Call conditionally	2	4 (conditions true) 2 (any condition false)	1110 10TP ZLVC ZLVC + 1 word
INTR	Soft interrupt	1	4	1011 1110 011I NTR#
NMI	Nonmaskable interrupt	1	4	1011 1110 0101 0010
RET	Return from subroutine	1	4	1110 1111 0000 0000
RETC	Return conditionally	1	4 (conditions true) 2 (any condition false)	1110 11TP ZLVC ZLVC
TRAP	Software interrupt	1	4	1011 1110 0101 0001

Table 7–6. I/O and Memory Instructions

Mnemonic	Description	Words	Cycles	Opcode
BLDD	Block move from data memory to data memory, direct/indirect with long immediate source	2	3	1010 1000 IAAA AAAA + 1 word
	Block move from data memory to data memory, direct/indirect with long immediate destination	2	3	1010 1001 IAAA AAAA + 1 word
BLPD	Block move from program memory to data memory, direct/indirect with long immediate source	2	3	1010 0101 IAAA AAAA + 1 word
DMOV	Data move in data memory, direct or indirect	1	1	0111 0111 IAAA AAAA
IN	Input data from I/O location, direct or indirect	2	2	1010 1111 IAAA AAAA + 1 word
OUT	Output data to port, direct or indirect	2	3	0000 1100 IAAA AAAA + 1 word
SPLK	Store long immediate to data memory location, direct or indirect	2	2	1010 1110 IAAA AAAA + 1 word
TBLR	Table read, direct or indirect	1	3	1010 0110 IAAA AAAA
TBLW	Table write, direct or indirect	1	3	1010 0111 IAAA AAAA

Table 7–5. Control Instructions

Mnemonic	Description	Words	Cycles	Opcode
BIT	Test bit, direct or indirect	1	1	0100 BITX IAAA AAAA
BITT	Test bit specified by TREG, direct or indirect	1	1	0110 1111 IAAA AAAA
CLRC	Clear C bit	1	1	1011 1110 0100 1110
	Clear CNF bit	1	1	1011 1110 0100 0100
	Clear INTM bit	1	1	1011 1110 0100 0000
	Clear OVM bit	1	1	1011 1110 0100 0010
	Clear SXM bit	1	1	1011 1110 0100 0110
	Clear TC bit	1	1	1011 1110 0100 1010
	Clear XF bit	1	1	1011 1110 0100 1100
IDLE	Idle until interrupt	1	1	1011 1110 0010 0010
LDP	Load data page pointer, direct or indirect	1	2	0000 1101 IAAA AAAA
	Load data page pointer, short immediate	1	2	1011 110I IIII IIII
LST	Load status register ST0, direct or indirect	1	2	0000 1110 IAAA AAAA
	Load status register ST1, direct or indirect	1	2	0000 1111 IAAA AAAA
NOP	No operation	1	1	1000 1011 0000 0000
POP	Pop top of stack to low ACC	1	1	1011 1110 0011 0010
POPD	Pop top of stack to data memory, direct or indirect	1	1	1000 1010 IAAA AAAA
PSHD	Push data memory value on stack, direct or indirect	1	1	0111 0110 IAAA AAAA
PUSH	Push low ACC onto stack	1	1	1011 1110 0011 1100
RPT	Repeat next instruction, direct or indirect	1	1	0000 1011 IAAA AAAA
	Repeat next instruction, short immediate	1	1	1011 1011 IIII IIII
SETC	Set C bit	1	1	1011 1110 0100 1111
	Set CNF bit	1	1	1011 1110 0100 0101
	Set INTM bit	1	1	1011 1110 0100 0001
	Set OVM bit	1	1	1011 1110 0100 0011
	Set SXM bit	1	1	1011 1110 0100 0111
	Set TC bit	1	1	1011 1110 0100 1011
	Set XF bit	1	1	1011 1110 0100 1101
SPM	Set product shift mode	1	1	1011 1111 0000 00PM
SST	Store status register ST0, direct or indirect	1	1	1000 1110 IAAA AAAA
	Store status register ST1, direct or indirect	1	1	1000 1111 IAAA AAAA