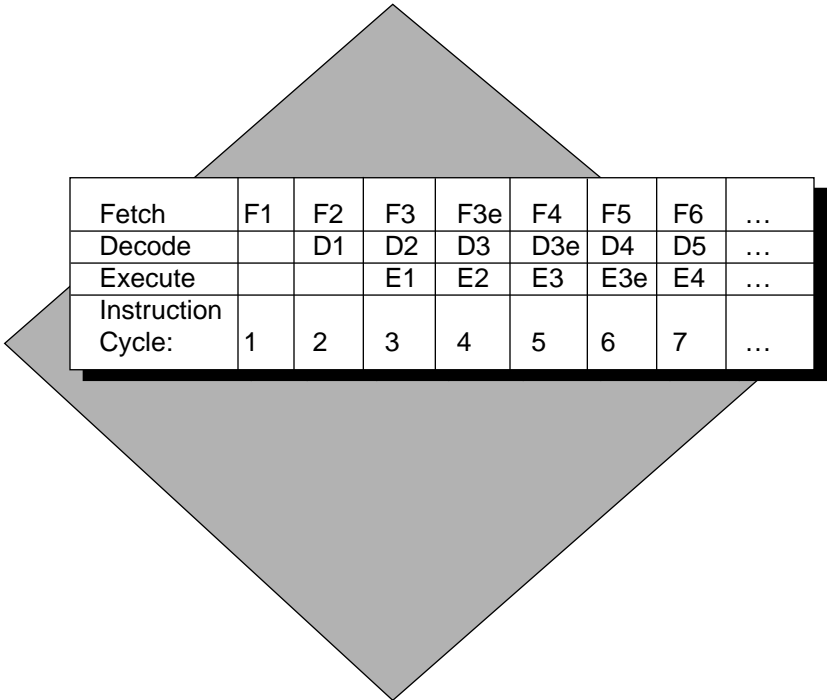


SECTION 6

INSTRUCTION SET AND EXECUTION



Fetch	F1	F2	F3	F3e	F4	F5	F6	...
Decode		D1	D2	D3	D3e	D4	D5	...
Execute			E1	E2	E3	E3e	E4	...
Instruction Cycle:	1	2	3	4	5	6	7	...

SECTION CONTENTS

6.1	INTRODUCTION	6-3
6.2	INSTRUCTION GROUPS	6-3
6.2.1	Arithmetic Instructions	6-3
6.2.2	Logical Instructions	6-4
6.2.3	Bit Field Manipulation Instructions	6-5
6.2.4	Loop Instructions	6-5
6.2.5	Move Instructions	6-6
6.2.6	Program Control Instructions	6-6
6.3	INSTRUCTION FORMATS	6-6
6.4	INSTRUCTION EXECUTION	6-7
6.4.1	Instruction Processing	6-7
6.4.2	Memory Access Processing	6-8

6.1 INTRODUCTION

As indicated by the programming model in Chapter 5, the DSP architecture can be viewed as three functional units operating in parallel (Data ALU, AGU and PCU). The goal of the instruction set is to keep each of these units busy each instruction cycle. This achieves maximum speed and minimum use of program memory.

This section introduces the DSP instruction set and instruction format. The complete range of instruction capabilities combined with the flexible addressing modes provide a very powerful assembly language for digital signal processing algorithms. The instruction set has also been designed to allow efficient coding for future high-level DSP language compilers. Execution time is enhanced by the hardware looping capabilities.

6.2 INSTRUCTION GROUPS

The instruction set is divided into the following groups:

- Arithmetic
- Logical
- Bit Field Manipulation
- Loop
- Move
- Program Control

Each instruction group is described in the following sections. Detailed information on each instruction is given in Appendix A.

6.2.1 Arithmetic Instructions

The arithmetic instructions perform all of the arithmetic operations within the Data ALU. They may affect all of the condition code register bits. Arithmetic instructions are register-based (register direct addressing modes used for operands) so that the Data ALU operation indicated by the instruction does not use the XDB or the GDB. Optional data transfers may be specified with most arithmetic instructions. This allows for parallel data movement over the XDB and over the GDB during a Data ALU operation. This allows new data to be prefetched for use in following instructions and results calculated by previous instructions to be stored. These instructions execute in one instruction cycle. The following are the arithmetic instructions.

ABS	Absolute Value
ADC	Add Long with Carry
ADD	Add
ASL	Arithmetic Shift Left
ASL4	4 Bit Arithmetic Shift Left*
ASR	Arithmetic Shift Right

ASR4	4 Bit Arithmetic Shift Right*
ASR16	16 Bit Arithmetic Shift Right*
CLR	Clear an Accumulator
CLR24	Clear 24 MSBs of an Accumulator
CMP	Compare
CMPM	Compare Magnitude
DEC	Decrement Accumulator
DEC24	Decrement upper word of Accumulator
DIV	Divide Iteration*
DMAC	Double (Multi) precision oriented MAC*
EXT	Sign Extend Accumulator from bit 31*
IMAC	Integer Multiply-Accumulate*
IMPY	Integer Multiply*
INC	Increment Accumulator
INC24	Increment 24 MSBs of Accumulator
MAC	Signed Multiply-Accumulate
MACR	Signed Multiply-Accumulate and Round
MPY	Signed Multiply
MPYR	Signed Multiply and Round
MPY(su,uu)	Mixed mode Multiply*
MAC(su,uu)	Mixed mode Multiply-Accumulate*
NEG	Negate
NEGC	Negate with Borrow*
NORM	Normalize*
RND	Round
SBC	Subtract Long with Carry
SUB	Subtract
SUBL	Shift Left and Subtract
SWAP	Swap MSP and LSP of an Accumulator*
Tcc	Transfer Conditionally*
TFR	Transfer Data ALU Register (Accumulator as destination)
TFR2	Transfer Accumulator (32 bit Data Alu register as destination)*
TST	Test an accumulator
TST2	Test an ALU data register*
ZERO	Zero Extend Accumulator from bit 31*

*These instructions do not allow parallel data moves.

6.2.2 Logical Instructions

The logical instructions perform all of the logical operations within the Data ALU. They may affect all of the condition code register bits. Logical instructions are register-based as are the arithmetic instructions above. Optional data transfers may be specified with most logical instructions. This allows for parallel data movement over the XDB and over the GDB during a Data ALU operation. This allows new data to be prefetched for use in following instructions and results calculated in previous instructions to be stored. With the exceptions of ANDI or ORI the destination of all logical instructions is A1 or B1.

These instructions execute in one instruction cycle. The following are the logical instructions.

AND	Logical AND
ANDI	AND Immediate Program Controller Register*
EOR	Logical Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	OR Immediate Program Controller Register*
ROL	Rotate Left
ROR	Rotate Right

*These instructions do not allow parallel data moves.

6.2.3 Bit Field Manipulation Instructions

This group tests the state of any set of bits within a byte in a memory location or a register and then sets, clears, or inverts bits in this byte. Bit fields which can be tested include the upper byte and the lower byte in a 16 bit value. The carry bit of the condition code register will contain the result of the bit test for each instruction. These instructions are read-modify-write type operations and require two instruction cycles. The following are the bit field manipulation instructions.

BFTSTL	Bit Field Test Low
BFTSTH	Bit Field Test High
BFCLR	Bit Field Test and Clear
BFSET	Bit Field Test and Set
BFCHG	Bit Field Test and Change

6.2.4 Loop Instructions

The loop instructions control hardware looping by initiating a program loop and setting up looping parameters, or by “cleaning” up the system stack when terminating a loop. Initialization includes saving registers used by a program loop (LA and LC) on the system stack so that program loops can be nested. The address of the first instruction in a program loop is also saved to allow no-overhead looping. The end address of the DO loop is specified as PC relative. The following are the loop instructions.

DO	Start Hardware Loop
DO FOREVER	Hardware Loop for ever
ENDDO	Disable Current Loop and Unstack Parameters
BRKcc	Conditional Exit from Hardware Loop

6.2.5 Move Instructions

The move instructions perform data movement over the XDB and over the GDB. Move instructions do not affect the condition code register except the limit bit L if limiting is performed when reading a Data ALU accumulator register. AGU instructions are also included among the following move instructions. These instructions do not allow optional data transfers. In addition to the following move instructions, there are parallel moves which can be used simultaneously with many of the other instructions.

LEA	Load Effective Address
MOVE	Move Data with or without register transfer – TFR(3)
MOVE(C)	Move Control Register
MOVE(I)	Move Immediate Short
MOVE(M)	Move Program Memory
MOVE(P)	Move Peripheral Data
MOVE(S)	Move Absolute Short

6.2.6 Program Control Instructions

The program control instructions include branches, jumps, conditional branches and jumps and other instructions which affect the PC and system stack. Program control instructions may affect the condition code register bits as specified in the instruction. The following are the program control instructions.

Bcc	Branch Conditionally
BSR	Branch to Subroutine (PC relative)
BRA	Branch
BScc	Branch to Subroutine Conditionally
DEBUG	Enter Debug Mode
DEBUGcc	Enter Debug Mode Conditionally
Jcc	Jump Conditionally
JMP	Jump
JSR	Jump to Subroutine
JScc	Jump to Subroutine Conditionally
NOP	No Operation
REP	Repeat Next Instruction
REPcc	Repeat Next Instruction Conditionally
RESET	Reset Peripheral Devices
RTI	Return from Interrupt
RTS	Return from Subroutine
STOP	Stop Processing (low power stand-by)
SWI	Software Interrupt
WAIT	Wait for Interrupt (low power stand-by)

6.3 INSTRUCTION FORMATS

Instructions are one or two words in length. The instruction and its length are specified by the first word of the instruction. The next word may contain information about the instruction itself or about an operand for the instruction. The assembly language source code

for a typical one word instruction is shown below. The source code is organized into four columns.

Opcode	Operands	X Bus Data	G Bus Data
MAC	X0,Y0,A	X:(R0)+,X0	X:(R3)+,Y0

The Opcode column indicates the Data ALU, AGU, or PCU operation to be performed. The Operands column specifies the operands to be used by the opcode. The X Bus Data and G Bus Data columns specify optional data transfers over the X Bus and the addressing modes to be used. The Opcode column must always be included in the source code.

The DSP offers parallel processing using the Data ALU, AGU and PCU. For the instruction word above, the DSP will perform the designated ALU operation (Data ALU), up to two data transfers specified with address register updates (AGU), and will also decode the next instruction and fetch an instruction from program memory (PCU) all in one instruction cycle. When an instruction is more than one word in length, an additional instruction execution cycle is required. Most instructions involving the Data ALU are register-based (all operands are in Data ALU registers) and allow the programmer to keep each parallel processing unit busy. An instruction which is memory-oriented (such as a bit field manipulation instruction) or that causes a control flow change (such as a branch/jump) prevents the use of parallel processing resources during its execution.

6.4 INSTRUCTION EXECUTION

Instruction execution is pipelined to allow most instructions to execute at a rate of one instruction every clock cycle. However, certain instructions will require additional time to execute. These include instructions which are longer than one word, instructions which use an addressing mode that requires more than one cycle, instructions which make use of the global data bus more than once, and instructions which cause a control flow change. In the latter case a cycle is needed to clear the pipeline.

6.4.1 Instruction Processing

Pipelining allows the fetch-decode-execute operations of an instruction to occur during the fetch-decode-execute operations of other instructions. While an instruction is executed, the next instruction to be executed is decoded, and the instruction to follow the instruction being decoded is fetched from program memory. If an instruction is two words in length, the additional word will be fetched before the next instruction is fetched. The illustration below demonstrates pipelining; F1, D1 and E1 refer to the fetch, decode and execute operations, respectively, of the first instruction. Note, the third instruction contains an instruction extension word and takes two cycles to execute.

	F1	F2	F3	F3e	F4	F5	F6	...
		D1	D2	D3	D3e	D4	D5	...
			E1	E2	E3	E3e	E4	...
Instruction Cycle:	1	2	3	4	5	6	7	...

Figure 6-1 Instruction Pipelining

Each instruction requires a minimum of 12 clock phases to be fetched, decoded, and executed. A new instruction may be started after four phases. Two word instructions require a minimum of 16 phases to execute and a new instruction may start after eight phases.

6.4.2 Memory Access Processing

One or more of the DSP memory sources (X data memory and program memory) may be accessed during the execution of an instruction. Each of these memory sources may be internal or external to the DSP. These address buses (XA1, XA2, and PAB) and three data buses (XD, program data, and Global Data) are available for internal memory accesses during one instruction cycle but only one address bus and one data bus are available for external memory accesses (when an external bus is available). If all memory sources are internal to the DSP, one or more of the two memory sources may be accessed in one instruction cycle (i.e., program memory access or program memory access plus an X memory reference). However, when one or more of the memories are external to the DSP, memory references may require additional instruction cycles. With internal program memory and one internal data memory, memory references will not require any additional instruction cycles (i.e. X memory references will take one instruction cycle). When program memory is external and the data memory is internal, no additional instruction cycles are required for all types of operand references. If the data memory is also external, an additional cycle is necessary when the external data memory is accessed (i.e., when X memory references are specified). If each memory source is external to the DSP, one additional cycle is required when one data memory is accessed i.e., when a X memory reference is specified).