Instruction Set Reference 15

15.1 QUICK LIST OF INSTRUCTIONS

This chapter is a complete reference for the instruction set of the ADSP-2100 family. The instruction set is organized by instruction group and, within each group, by individual instruction. The list below shows all of the instructions and the reference page for each.

15 Instruction Set Reference

15.2 OVERVIEW

This chapter provides an overview and detailed reference for the instruction set of the ADSP-2100 family of DSP microprocessors.

For information regarding the ADSP-2100 Family Development Software, refer to the *ADSP-2100 Family Assembler Tools & Simulator Manual*, *ADSP-2100 Family C Tools Manual*, and *ADSP-2100 Family C Runtime Library Manual*. These manuals provide a complete guide to the development software. The handbooks *Digital Signal Processing Applications Using The ADSP-2100 Family, Volume 1* and *Volume 2* present DSP applications programs with source code and discussion.

The instruction set is tailored to the computation-intensive algorithms common in DSP applications. For example, sustained single-cycle multiplication/accumulation operations are possible. The instruction set provides full control of the processors' three computational units: the ALU, MAC and Shifter. Arithmetic instructions can process single-precision 16-bit operands directly; provisions for multiprecision operations are available.

The high-level syntax of ADSP-2100 family source code is both readable and efficient. Unlike many assembly languages, the ADSP-2100 family instruction set uses an algebraic notation for arithmetic operations and for data moves, resulting in highly readable source code. There is no performance penalty for this; each program statement assembles into one 24-bit instruction which executes in a single cycle. There are no multicycle instructions in the instruction set. (If memory access times require, or contention for off-chip memory occurs, overhead cycles will be required, but all instructions can otherwise execute in a single cycle.)

In addition to JUMP and CALL, the instruction set's control instructions support conditional execution of most calculations and a DO UNTIL looping instruction. Return from interrupt (RTI) and return from subroutine (RTS) are also provided.

The IDLE instruction is provided for idling the processor until an interrupt occurs. IDLE puts the processor into a low-power state while waiting for interrupts.

Two addressing modes are supported for memory fetches. Direct addressing uses immediate address values; indirect addressing uses the I registers of the two data address generators (DAGs).

Instruction Set Reference 15

The 24-bit instruction word allows a high degree of parallelism in performing operations. The instruction set allows for single-cycle execution of any of the following combinations:

- any ALU, MAC or Shifter operation (conditional or non-conditional)
- any register-to-register move
- any data memory read or write
- a computation with any data register to data register move
- a computation with any memory read or write
- a computation with a read from two memories.

The instruction set allows maximum flexibility. It provides moves from any register to any other register, and from most registers to/from memory. In addition, almost any ALU, MAC or Shifter operation may be combined with any register-to-register move or with a register move to or from either internal or external memory.

15.3 INSTRUCTION TYPES & NOTATION CONVENTIONS

The ADSP-2100 family instruction set is grouped into the following categories:

- Computational: ALU, MAC, Shifter
- Move
- Program Flow
- Multifunction
- Miscellaneous

Because the multifunction instructions best illustrate the power of the processors' architecture, in the next section we begin with a discussion of this group of instructions.

Throughout this chapter you will find tables summarizing the syntax of the instruction groups. The following notation conventions are used in these tables and in the reference page for each instruction.

Square Brackets [ ]	Anything within square brackets is an optional part of the instruction statement.
Parallel Lines | |	Lists of operands are enclosed by vertical parallel bars. One of the operands listed must be chosen. If the parallel bars are within square brackets, then the operand is optional for that instruction.

15 Instruction Set Reference

CAPITAL LETTERS	Capital letters denote a literal in the instruction. Literals are the instruction name (e.g. ADD), register names, or operand selections. Literals must be typed exactly as shown.
operands	Some instruction operands are shown in lowercase letters. These operands may take different values in assembly code. For example, the operand *yop* may be one of several registers: AY0, AY1, or AF.
<exp>	Denotes exponent (shift value) in Shift Immediate instructions; must be an 8-bit signed integer constant.
<data>	Denotes an immediate data value. Can also be a symbol (address label or variable/buffer name) dereferenced by the '%' or '^' operators.
<addr>	Denotes an immediate address value to be encoded in the instruction. The <addr> may be either an immediate value (a constant) or a program label.
<reg>	Refers to any accessible register; see Table 15.7.
<dreg>	Refers to any data register; see Table 15.7.

Immediate values, <exp>, <data>, or <addr>, may be a constant in decimal, hexadecimal, octal or binary format. Default is to decimal.

15.4 MULTIFUNCTION INSTRUCTIONS

Multifunction operations take advantage of the inherent parallelism of the ADSP-2100 family architecture by providing combinations of data moves, memory reads/memory writes, and computation, all in a single cycle.

15.4.1 ALU/MAC With Data & Program Memory Read

Perhaps the single most common operation in DSP algorithms is the sum of products, performed as follows:

- Fetch two operands (such as a coefficient and data point)
- Multiply the operands and sum the result with previous products

Instruction Set Reference 15

The ADSP-2100 family processors can execute both data fetches and the multiplication/accumulation in a single-cycle. Typically, a loop of multiply/accumulates can be expressed in ADSP-21xx source code in just two program lines. Since the on-chip program memory of the ADSP-21xx processors is fast enough to provide an operand and the next instruction in a single cycle, loops of this type can execute with sustained single-cycle throughput. An example of such an instruction is:

```
MR=MR+MX0*MY0(SS), MX0=DM(I0,M0), MY0=PM(I4,M5);
```

The first clause of this instruction (up to the first comma) says that MR, the MAC result register, gets the sum of its previous value plus the product of the (current) X and Y input registers of the MAC (MX0 and MY0) both treated as signed (SS).

In the second and third clauses of this multifunction instruction two new operands are fetched. One is fetched from the data memory (DM) pointed to by index register zero (I0, post modified by the value in M0) and the other is fetched from the program memory location (PM) pointed to by I4 (post-modified by M5 in this instance). Note that indirect memory addressing uses a syntax similar to array indexing, with DAG registers providing the index values. Any I register may be paired with any M register within the same DAG.

As discussed in Chapter 2, “Computational Units,” registers are read at the beginning of the cycle and written at the end of the cycle. The operands present in the MX0 and MY0 registers at the beginning of the instruction cycle are multiplied and added to the MAC result register, MR. The new operands fetched at the end of this same instruction overwrite the old operands after the multiplication has taken place and are available for computation on the following cycle. You may, of course, load any data registers in conjunction with the computation, not just MAC registers with a MAC operation as in our example.

The computational part of this multifunction instruction may be any unconditional ALU instruction except division or any MAC instruction except saturation. Certain other restrictions apply: the next X operand must be loaded into MX0 from data memory and the new Y operand must be loaded into MY0 from program memory (internal and external memory are identical at the level of the instruction set). The result of the computation must go to the result register (MR or AR) not to the feedback register (MF or AF).

15.4.2 Data & Program Memory Read

This variation of a multifunction instruction is a special case of the multifunction instruction described above in which the computation is omitted. It executes only the dual operand fetch, as shown below:

```
AX0=DM(I2,M0), AY0=PM(I4,M6);
```

In this example we have used the ALU input registers as the destination. As with the previous multifunction instruction, X operands must come from data memory and Y operands from program memory (internal or external memory in either case, for the processors with on-chip memory).

15.4.3 Computation With Memory Read

If a single memory read is performed instead of the dual memory read of the previous two multifunction instructions, a wider range of computations can be executed. The legal computations include all ALU operations except division, all MAC operations and all Shifter operations except SHIFT IMMEDIATE. Computation must be unconditional. An example of this kind of multifunction instruction is:

```
AR=AX0+AY0, AX0=DM(I0,M3);
```

Here an addition is performed in the ALU while a single operand is fetched from data memory. The restrictions are similar to those for previous multifunction instructions. The value of AX0, used as a source for the computation, is the value at the beginning of the cycle. The data read operation loads a new value into AX0 by the end of the cycle. For this same reason, the destination register (AR in the example above) cannot be the destination for the memory read.

15.4.4 Computation With Memory Write

The computation with memory write instruction is similar in structure to the computation with memory read: the order of the clauses in the instruction line, however, is reversed. First the memory write is performed, then the computation, as shown below:

```
DM(I0,M0)=AR, AR=AX0+AY0;
```

Again the value of the source register for the memory write (AR in this example) is the value at the beginning of the instruction. The computation loads a new value into the same register; this is the value in AR at the end of this instruction. Reversing the order of the clauses of the instruction is illegal and causes the assembler to generate a warning; it would imply

that the result of the computation is written to memory when, in fact, the previous value of the register is what is written. There is no requirement that the same register be used in this way although this will usually be the case in order to pipeline operands to the computation.

The restrictions on computation operations are identical to those given above. All ALU operations except division, all MAC operations, and all Shifter operations except SHIFT IMMEDIATE are legal. Computations must be unconditional.

15.4.5 Computation With Data Register Move

This final type of multifunction instruction performs a data register to data register move in parallel with a computation. Most of the restrictions applying to the previous two instructions also apply to this instruction.

```
AR=AX0+AY0, AX0=MR2;
```

Here an ALU addition operation occurs while a new value is loaded into AX0 from MR2. As before, the value of AX0 at the beginning of the instruction is the value used in the computation. The move may be from or to all ALU, MAC and Shifter input and output registers except the feedback registers (AF and MF) and SB.

In the example, the data register move loads the AX0 register with the new value at the end of the cycle. All ALU operations except division, all MAC operations and all Shifter operations except SHIFT IMMEDIATE are legal. Computation must be unconditional.

A complete list of data registers is given in Table 15.7. A complete list of the permissible *xops* and *yops* for computational operations is given in the reference page for each instruction. Table 15.1 shows the legal combinations for multifunction instructions: you may combine operations on the same row with each other.

Unconditional Computations	*Data Move (DM=DAG1)*	*Data Move (PM=DAG2)*
None or any ALU (except Division) or MAC	DM read	PM read
Any ALU except Division Any MAC Any Shift except Immediate	DM read	—
	—	PM read
	DM write	—
	—	PM write
	Register-To-Register	

Table 15.1 Summary Of Valid Combinations For Multifunction Instructions