Chapter 8

Structure of Assembly Code

An assembly language program must be an ASCII text file. Any line of assembly code can include up to seven items:

- ❑ Label
- ❑ Parallel bars
- ❑ Conditions
- ❑ Instruction
- ❑ Functional unit
- ❑ Operands
- ❑ Comment

Topic **Page**

8.1 Labels

A label identifies a line of code or a variable and represents a memory address that contains either an instruction or data.

Figure 8−1 shows the position of the label in a line of assembly code. The colon following the label is optional.

Figure 8−1. Labels in Assembly Code

label:	parallel bars	[condition]	instruction	unit	operands	; comments

Labels must meet the following conditions:

- The first character of a label must be a letter or an underscore (_) followed by a letter.
- The first character of the label must be in the first column of the text file.
- Labels can include up to 32 alphanumeric characters.

8.2 Parallel Bars

An instruction that executes in parallel with the previous instruction signifies this with parallel bars (||). This field is left blank for an instruction that does not execute in parallel with the previous instruction.

Figure 8−2. Parallel Bars in Assembly Code

label:	**parallel bars**	[condition]	instruction	unit	operands	; comments

8.3 Conditions

Five registers on the C62x/C67x are available for conditions: A1, A2, B0, B1, and B2. Six registers on the C64x/C64x+ are available for conditions: A0, A1, A2, B0, B1, and B2. Figure 8−3 shows the position of a condition in a line of assembly code.

Figure 8−3. Conditions in Assembly Code

label:	parallel bars	**[condition]**	instruction	unit	operands	; comments

All C6000 instructions are conditional:

- ❑ If no condition is specified, the instruction is always performed.
- ❑ If a condition is specified and that condition is true, the instruction executes. For example:

With this condition ...	The instruction executes if ...
`[A1]`	A1 ! = 0
`[!A1]`	A1 = 0

- ❑ If a condition is specified and that condition is false, the instruction does not execute.

With this condition ...	The instruction does not execute if ...
`[A1]`	A1 = 0
`[!A1]`	A1! = 0

8.4 Instructions

Assembly code instructions are either directives or mnemonics:

- ❑ *Assembler directives* are commands for the assembler that control the assembly process or define the data structures (constants and variables) in the assembly language program. All assembler directives begin with a period, as shown in the partial list in Table 8−1. See the *TMS320C6000 Assembly Language Tools User's Guide* for a complete list of directives.
- ❑ *Processor mnemonics* are the actual microprocessor instructions that execute at run time and perform the operations in the program. Processor mnemonics must begin in column 2 or greater. For more information about processor mnemonics, see the *TMS320C6000 CPU and Instruction Set User's Guide*.

Figure 8−4 shows the position of the instruction in a line of assembly code.

Figure 8−4. Instructions in Assembly Code

label:	parallel bars	[condition]	**instruction**	unit	operands	; comments

Table 8−1. Selected TMS320C6x Directives

Directives	Description
.sect "*name*"	Create section of information (data or code)
.double *value*	Reserve two consecutive 32 bits (64 bits) in memory and fill with double-precision (64-bit) IEEE floating-point representation of specified value
.float *value*	Reserve 32 bits in memory and fill with single-precision (32-bit) IEEE floating-point representation of specified value
.int *value* .long *value* .word *value*	Reserve 32 bits in memory and fill with specified value
.short *value* .half *value*	Reserve 16 bits in memory and fill with specified value
.byte *value*	Reserve 8 bits in memory and fill with specified value

See the *TMS320C6000 Assembly Language Tools User's Guide* for a complete list of directives.

8.5 Functional Units

The C6000 CPU contains eight functional units, which are shown in Figure 8−5 and described in Table 8−2.

Figure 8−5. TMS320C6x Functional Units

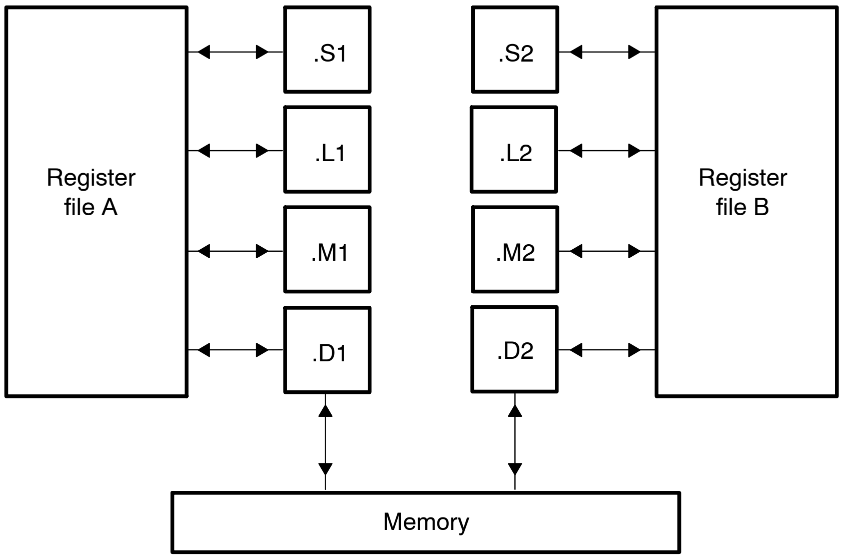

Table 8–2. Functional Units and Operations Performed

Functional Unit	Fixed-Point Operations	Floating-Point Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits **Byte shifts** **Data packing/unpacking** **5-bit constant generation** **Dual 16-bit arithmetic operations** **Quad 8-bit arithmetic operations** **Dual 16-bit min/max operations** **Quad 8-bit min/max operations** ***Dual data packing operations*** ***Simultaneous add and sub operations***	Arithmetic operations DP → SP, INT → DP, INT → SP conversion operations
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) **Byte shifts** **Data packing/unpacking** **Dual 16-bit compare operations** **Quad 8-bit compare operations** **Dual 16-bit shift operations** **Dual 16-bit saturated arithmetic operations** **Quad 8-bit saturated arithmetic operations** ***Dual 16-bit min/max operations*** ***Dual register move operation***	Compare Reciprocal and reciprocal square-root operations Absolute value operations SP → DP conversion operations

Note: Fixed-point operations are available on all three devices. Floating-point operations and 32 x 32-bit fixed-point multiply are available only on the C67x. Additional C64x/C64x+ functions are shown in bold. C64x+ only functions are shown in bold and intalics.

Table 8−2. Functional Units and Operations Performed (Continued)

Functional Unit	Fixed-Point Operations	Floating-Point Operations
.M unit (.M1, .M2)	16 x 16 multiply operations	32 X 32-bit fixed-point multiply operations
		Floating-point multiply operations
	16 x 32 multiply operations	
	Quad 8 x 8 multiply operations	
	Dual 16 x 16 multiply operations	
	Dual 16 x 16 multiply with add/subtract operations	
	Quad 8 x 8 multiply with add operation	
	Bit expansion	
	Bit interleaving/de-interleaving	
	Variable shift operations	
	Rotation	
	Galois Field Multiply	
	32x32 multiply operations	
	Complex mulitply operations	
	Quad 16x16 multiply with add operations	
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation	Load doubleword with 5-bit constant offset
	Loads and stores with 5-bit constant offset	
	Loads and stores with 15-bit constant offset (.D2 only)	
	Dual 16-bit arithmetic operations	
	Load and store double words with 5-bit constant	
	Load and store non-aligned words and double words	
	5-bit constant generation	
	32-bit logical operations	

Note: Fixed-point operations are available on all three devices. Floating-point operations and 32 x 32-bit fixed-point multiply are available only on the C67x. Additional C64x/C64x+ functions are shown in bold. C64x+ only functions are shown in bold and intalics.

Figure 8−6 shows the position of the unit in a line of assembly code.

Figure 8−6. Units in the Assembly Code

label: parallel bars [condition] instruction **unit** operands ; comments

Specifying the functional unit in the assembly code is optional. The functional unit can be used to document which resource(s) each instruction uses.

8.6 Operands

The C6000 architecture requires that memory reads and writes move data between memory and a register. Figure 8−7 shows the position of the operands in a line of assembly code.

Figure 8−7. Operands in the Assembly Code

label:	parallel bars	[condition]	instruction	unit	**operands**	; comments

Instructions have the following requirements for operands in the assembly code:

- ❑ All instructions require a destination operand.
- ❑ Most instructions require one or two source operands.
- ❑ The destination operand must be in the same register file as one source operand.
- ❑ One source operand from each register file per execute packet can come from the register file opposite that of the other source operand.

  When an operand comes from the other register file, the unit includes an X, as shown in Figure 8−8, indicating that the instruction is using one of the cross paths. (See the *TMS320C6000 CPU and Instruction Set Reference Guide* for more information on cross paths.)

Figure 8−8. Operands in Instructions

```
ADD     .L1     A0,A1,A3
ADD     .L1X    A0,B1,A3
```

All registers except B1 are on the same side of the CPU.

The C6000 instructions use three types of operands to access data:

- ❑ Register operands indicate a register that contains the data.
- ❑ Constant operands specify the data within the assembly code.
- ❑ Pointer operands contain addresses of data values.

  Only the load and store instructions require and use pointer operands to move data values between memory and a register.

8.7 Comments

As with all programming languages, comments provide code documentation. Figure 8−9 shows the position of the comment in a line of assembly code.

Figure 8−9. Comments in Assembly Code

label:	parallel bars	[condition]	instruction	unit	operands	**; comments**

The following are guidelines for using comments in assembly code:

- ❑ A comment may begin in any column when preceded by a semicolon (;).
- ❑ A comment must begin in first column when preceded by an asterisk (*).
- ❑ Comments are not required but are recommended.