Chapter 6

SYMBOLIC INSTRUCTIONS AND ADDRESSING

Objective: To provide the basics of the assembly language instruction set and the requirements for addressing data.

INTRODUCTION

This chapter introduces the categories of the processor's instruction set. The instructions formally this chapter are MOV, MOVSX, MOVZX, XCHG, LEA, INC, DEC, ADD, SUB, and INT, as well of constants in instruction operands as immediate values. Finally, the chapter describes the basic formats that are used throughout the rest of the book, and then explains address alignment and t override prefix.

THE SYMBOLIC INSTRUCTION SET—AN OVERVIEW

The following is a list of the symbolic instructions for the Intel processor family, arranged by Although the list seems formidable, many of the instructions are rarely needed.

ARITHMETIC

ADC: Add with Carry
ADD: Add Binary Numbers
DEC: Decrement by 1
DIV: Unsigned Divide
IDIV: Signed (Integer) Divide
IMUL: Signed (Integer) Multiply
INC: Increment by 1
MUL: Unsigned Multiply
NEG: Negate
SBB: Subtract with Borrow
SUB: Subtract Binary Values
XADD: Exchange and Add

ASCII-BCD CONVERSION

AAA: ASCII Adjust After Addition
AAD: ASCII Adjust Before Division
AAM: ASCII Adjust After Multiplication
AAS: ASCII Adjust After Subtraction
DAA: Decimal Adjust After Addition
DAS: Decimal Adjust After Subtraction

BIT SHIFTING

RCL: Rotate Left Through Carry
RCR: Rotate Right Through Carry
ROL: Rotate Left
ROR: Rotate Right
SAL: Shift Algebraic Left
SAR: Shift Algebraic Right
SHL: Shift Logical Left
SHR: Shift Logical Right
SHLD: Shift Left Double (80386+)
SHRD: Shift Right Double (80386+)

COMPARISON

BSF/BSR: Bit Scan (80386+)
BT/BTC/BTR/BTS: Bit Test (80386+)
CMP: Compare
CMPSn: Compare String
CMPXCHG: Compare and Exchange (80486+)
CMPXCHG8B: Compare and Exchange (Pentium+)
TEST: Test Bits

DATA TRANSFER

LDS: Load Data Segment Register
LEA: Load Effective Address
LES: Load Extra Segment Register
LODS: Load String
LSS: Load Stack Segment Register
MOV: Move Data
CMOVcc : Conditional Move
MOVS: Move String
MOVSX: Move With Sign-Extend
MOVZX: Move With Zero-Extend
STOS: Store String
XCHG: Exchange
XLAT: Translate

FLAG OPERATIONS

CLC: Clear Carry Flag
CLD: Clear Direction Flag
CLI: Clear Interrupt Flag
CMC: Complement Carry Flag
LAHF: Load AH from Flags
POPF: Pop Flags off Stack
PUSHF: Push Flags onto Stack
SAHF: Store AH in Flags
STC: Set Carry Flag
STD: Set Direction Flag
STI: Set Interrupt Flag
SETcc: Set byte on Flag

INPUT/OUTPUT

IN: Input Byte or Word
INSn: Input String (80286+)
OUT: Output Byte or Word
OUTSn: Output String (80286+)

LOGICAL OPERATIONS

AND: Logical AND
NOT: Logical NOT
OR: Logical OR
XOR: Exclusive OR

LOOPING

LOOP: Loop until Complete
LOOPE: Loop While Equal
LOOPZ: Loop While Zero
LOOPNE: Loop While Not Equal
LOOPNZ: Loop While Not Zero
LOOPNEW: Loop While Not Equal (80386+)
LOOPNZW: Loop While Not Zero (80386+)

PROCESSOR CONTROL

HLT: Enter Halt State
LOCK: Lock Bus
NOP: No Operation
WAIT: Put Processor in Wait State

STACK OPERATIONS

ENTER: Make Stack Frame (80286+)
LEAVE: Terminate Stack Frame (80286+)
POP: Pop Word off Stack
POPF: Pop Flags off Stack
POPA: Pop All General Registers (80286+)
PUSH: Push Word onto Stack
PUSHA: Push All General Registers (80286+)
PUSHF: Push Flags off Stack

STRING OPERATIONS

CMPS: Compare String
LODS: Load String
MOVS: Move String
REP: Repeat String
REPE: Repeat While Equal
REPZ: Repeat While Zero
REPNE: Repeat While Not Equal
REPNZ: Repeat While Not Zero
SCAS: Scan String
STOS: Store String

TRANSFER (CONDITIONAL)

INTO: Interrupt on Overflow
JA: Jump If Above
JAE: Jump If Above/Equal
JB: Jump If Below
JBE: Jump If Below/Equal
JC: Jump If Carry
JCXZ: Jump If CX Is Zero
JE: Jump If Equal
JG: Jump If Greater
JGE: Jump If Greater/Equal
JL: Jump If Less
JLE: Jump If Less/Equal
JNA: Jump If Not Above
JNAE: Jump If Not Above/Equal
JNB: Jump If Not Below
JNBE: Jump If Not Below/Equal
JNC: Jump If No Carry
JNE: Jump If Not Equal
JNG: Jump If Not Greater
JNGE: Jump If Not Greater/Equal
JNL: Jump If Not Less
JNLE: Jump If Not Less/Equal
JNO: Jump If No Overflow
JNP: Jump If No Parity
JNS: Jump If No Sign
JNZ: Jump If Not Zero
JO: Jump If Overflow
JP: Jump If Parity
JPE: Jump If Parity Even
JPO: Jump If Parity Odd
JS: Jump If Sign
JZ: Jump If Zero

TRANSFER (UNCONDITIONAL)

CALL: Call a Procedure
INT: Interrupt
IRET: Interrupt Return
JMP: Unconditional Jump
RET: Return
RETN/RETF: Return Near/Return Far

TYPE CONVERSION

CBW: Convert Byte to Word
CDQ: Convert Doubleword to Quadword (80386+)
CWD: Convert Word to Doubleword
CWDE: Convert Word to Extended Doubleword (80386+)

FLOATING POINT INSTRUCTIONS

FLD: Load Floating-point Value
FST: Store Floating-point Value
FSTP: Store Floating-Point Value and Pop
FILD: Load Integer
FIST: Store Integer
FISTP: Store Integer and Pop
FBLD: Load BCD
FBSTP: Store BCD and Pop
FXCH: Exchange Registers
FCMOVE/FCMOVNE: Floating-point Conditional Move If Equal/Not Equal(Pentium)

FADD: Add Floating-point
FADDP: Add Floating-point and Pop
FIADD: Add Integer
FSUB: Subtract Floating-point
FSUBP: Subtract Floating-point and Pop
FISUB: Subtract Integer
FSUBR: Subtract Floating-point Reverse
FSUBRP: Subtract Floating-point Reverse and Pop
FISUBR: Subtract Integer Reverse
FMUL: Multiply Floating-point
FMULP: Multiply Floating-point and Pop
FIMUL: Multiply Integer
FDIV: Divide Floating-point
FDIVP: Divide Floating-point and Pop
FIDIV: Divide Integer
FDIVR: Divide Floating-point Reverse
FDIVRP: Divide Floating-point Reverse and Pop
FIDIVR: Divide Integer Reverse
FPREM: Partial Remainder
FPREM1: IEEE Partial Remainder
FABS: Absolute Value
FCHS: Change Sign
FRNDINT: Round to Integer
FSCALE: Scale by Power of Two
FSQRT: Square Root
FXTRACT: Extract Exponent and Significand

FCOM: Compare Floating-point
FCOMP: Compare Floating-point and Pop
FCOMPP: Compare Floating-point and Pop twice
FUCOM: Unordered Compare Floating-point
FUCOMP: Unordered Compare Floating-point and Pop (80387+)
FUCOMPP: Unordered Compare Floating-point and Pop twice (80387+)

FICOM: Compare Integer
FICOMP: Compare Integer and Pop
FCOMI: Compare Floating-point and Set EFLAGS
FUCOMI: Unordered Compare Floating-point and Set EFLAGS

FCOMIP: Compare Floating-point, Set EFLAGS, and Pop
FUCOMIP: Unordered Compare Floating-point, Set EFLAGS, and Pop

FTST: Test Floating-point (Compare with 0.0)
FXAM: Examine Floating-point

FSIN: Sine (80387+)
FSINCOS: Sine and Cosine (80387+)
FPATAN: Partial Arctangent
FYL2X: y*log2x

FLD1: Load + 1.0
FLDPI: Load pi
FLDLN2: Load loge2
FLDLG2: Load log102

FCOS: Cosine (80387+)
FPTAN: Partial Tangent
F2XM1: 2x – 1
FYL2XP1: y*log2(x+1)

FLDZ: Load +0.0
FLDL2E: Load log2e
FLDL2T: Load log210

FINCSTP/FDECSTP: Increment/Decrement FPU Register Stack Pointer
FFREE: Free Floating-point Register
FINIT: Initialize FPU after checking error conditions
FNINIT: Initialize FPU without checking error conditions
FCLEX: Clear Floating-point Exception Flags after checking for error conditions
FNCLEX: Clear Floating-point Exception Flags without checking for error conditions
FSTCW: Store FPU Control Word after checking error conditions
FNSTCW: Store FPU Control Word without checking error conditions
FLDCW: Load FPU Control Word
FSTENV: Store FPU Environment after checking error conditions
FNSTENV: Store FPU Environment without checking error conditions
FLDENV: Load FPU Environment
FSAVE: Save FPU State after checking error conditions
FNSAVE: Save FPU State without checking error conditions
FRSTOR: Restore FPU State
FSTSW: Store FPU Status Word after checking error conditions
FNSTSW: Store FPU Status Word without checking error conditions
WAIT/FWAIT: Wait for FPU
FNOP: No operation

Instructions that indicate a processor, such as (80386+), require the use of a processor directiv
ered in Chapter 3) to assemble properly.

DATA TRANSFER INSTRUCTIONS

This section describes some of the commonly-used instructions concerned with data transfer.

The MOV Instruction

MOV transfers (or copies) data referenced by the address of the second operand to the address of th
operand. The sending field is unchanged. The operands that reference memory or registers must agree
(both must be bytes, both words, or both double-words). The format for MOV is

[label:]	MOV	*register/memory,register/memory/immediate*