2.1 CONTROL TRANSFER INSTRUCTIONS

In the sequence of instructions to be executed, it is often necessary to transfer program control to a different location. There are many instructions in the 80x86 to achieve this. This section covers the control transfer instructions available in the 8086 Assembly language. Before that, however, it is necessary to explain the concept of FAR and NEAR as it applies to jump and call instructions.

FAR and NEAR

If control is transferred to a memory location within the current code segment, it is NEAR. This is sometimes called *intrasegment* (within segment). If control is transferred outside the current code segment, it is a FAR or intersegment (between segments) jump. Since the CS:IP registers always point to the address of the next instruction to be executed, they must be updated when a control transfer instruction is executed. In a NEAR jump, the IP is updated and CS remains the same, since control is still inside the current code segment. In a FAR jump, because control is passing outside the current code segment, both CS and IP have to be updated to the new values. In other words, in any control transfer instruction such as jump or call, the IP must be changed, but only in the FAR case is the CS changed, too.

Conditional jumps

Conditional jumps, summarized in Table 2-1, have mnemonics such as JNZ (Jump not zero) and JC (Jump if carry). In the conditional jump, control is transferred to a new location if a certain condition is met. The flag register is the one that indicates the current condition. For example, with "JNZ label", the processor looks at the zero flag to see if it is raised. If not, the CPU starts to fetch and execute instructions from the address of the label. If ZF = 1, it will not jump but will execute the next instruction below the JNZ.

Short jumps

All conditional jumps are short jumps. In a short jump, the address of the target must be within -128 to + 127 bytes of the IP. In other words, the conditional jump is a two-byte instruction: one byte is the opcode of the J condition and the second byte is a value between 00 and FF. An offset range of 00 to FF gives 256 possible addresses; these are split between backward jumps (to -128) and forward jumps (to + 127).
In a jump backward, the second byte is the 2's complement of the displacement value. To calculate the target address, the second byte is added to the IP of the instruction after the jump. To understand this, look at the unassembled code below.

```
1067:0000 B86610      MOV   AX,1066
1067:0003 8ED8        MOV   DS,AX
1067:0005 B90500      MOV   CX,0005
1067:0008 BB0000      MOV   BX,0000
1067:000D 0207        ADD   AL,[BX]
1067:000F 43          INC   BX
1067:0010 49          DEC   CX
1067:0011 75FA        JNZ   000D
1067:0013 A20500      MOV   [0005],AL
1067:0016 B44C        MOV   AH,4C
1067:0018 CD21        INT   21
```

Table 2-1: 8086 Conditional Jump Instructions

Mnemonic	Condition Tested	"Jump IF ..."
JA/JNBE	(CF = 0) and (ZF = 0)	above/not below nor zero
JAE/JNB	CF = 0	above or equal/not below
JB/JNAE	CF = 1	below/not above nor equal
JBE/JNA	(CF or ZF) = 1	below or equal/not above
JC	CF = 1	carry
JE/JZ	ZF = 1	equal/zero
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	(SF xor OF) = 0	greater or equal/not less
JL/JNGE	(SF xor OR) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNC	CF = 0	not carry
JNE/JNZ	ZF = 0	not equal/not zero
JNO	OF = 0	not overflow
JNP/JPO	PF = 0	not parity/parity odd
JNS	SF = 0	not sign
JO	OF = 1	overflow
JP/JPE	PF = 1	parity/parity equal
JS	SF = 1	sign

The instruction "JNZ AGAIN" was assembled as "JNZ 000D", and 000D is the address of the instruction with the label AGAIN. The instruction "JNZ 000D" has the opcode 75 and the target address FA, which is located at offset addresses 0011 and 0012. This is followed by "MOV SUM,AL", which is located beginning at offset address 0013. The IP value of MOV, 0013, is added to FA to calculate the address of label AGAIN (0013 + FA = 000D) and the carry is dropped. In reality, FA is the 2's complement of -6, meaning that the address of the target is -6 bytes from the lP of the next instruction.

Similarly, the target address for a forward jump is calculated by adding the IP of the following instruction to the operand. In that case the displacement value is

positive, as shown next. Below is a portion of a list file showing the opcodes for several conditional jumps.

```
0005  8A 47 02 AGAIN:    MOV   AL,[BX]+2
0008  3C 61              CMP   AL,61H
000A  72 06              JB    NEXT
000C  3C 7A              CMP   AL,7AH
000E  77 02              JA    NEXT
0010  24 DF              AND   AL,ODFH
0012  88 04  NEXT:       MOV   [SI],AL
```

In the program above, "JB NEXT" has the opcode 72 and the target address 06 and is located at IP = 000A and 000B. The jump will be 6 bytes from the next instruction, which is IP = OOOC. Adding gives us OOOCH + 0006H = 0012H, which is the exact address of the NEXT label. Look also at "JA NEXT", which has 77 and 02 for the opcode and displacement, respectively. The IP of the following instruction, 0010, is added to 02 to get 0012, the address of the target location.

It must be emphasized that regardless of whether the jump is forward or backward, for conditional jumps the address of the target address can never be more than -128 to + 127 bytes away from the IP associated with the instruction following the jump (- for the backward jump and + for the forward jump). If any attempt is made to violate this rule, the assembler will generate a "relative jump out of range" message. These conditional jumps are sometimes referred to as SHORT jumps.

Unconditional jumps

"JMP label" is an unconditional jump in which control is transferred unconditionally to the target location label. The unconditional jump can take the following forms:

1. SHORT JUMP, which is specified by the format "JMP SHORT label". This is a jump in which the address of the target location is within -128 to + 127 bytes of memory relative to the address of the current IP. In this case, the opcode is EB and the operand is 1 byte in the range 00 to FF. The operand byte is added to the current IP to calculate the target address. If the jump is backward, the operand is in 2's complement. This is exactly like the J condition case. Coding the directive "short" makes the jump more efficient in that it will be assembled into a 2-byte instruction instead of a 3-byte instruction.