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.

2. NEAR JUMP, which is the default, has the format "JMP label". This is a near jump (within the current code segment) and has the opcode E9. The target address can be any of the addressing modes of direct, register, register indirect, or memory indirect:

(a) Direct JUMP is exactly like the short jump explained earlier, except that the target address can be anywhere in the segment within the range +32767 to -32768 of the current IP.

(b) Register indirect JUMP; the target address is in a register. For example, in "JMP BX", IP takes the value BX.

(c) Memory indirect JMP; the target address is the contents of two memory locations pointed at by the register. Example: "JMP [DI]" will replace the IP with the contents of memory locations pointed at by DI and DI+ 1.

3. FAR JUMP which has the format "JMP FAR PTR label". This is a jump out of the current code segment, meaning that not only the IP but also the CS is replaced with new values.

CALL statements

Another control transfer instruction is the CALL instruction, which is used to call a procedure. CALLs to procedures are used to perform tasks that need to be performed frequently. This makes a program more structured. The target address could be in the current segment, in which case it will be a NEAR call or outside the current CS segment, which is a FAR call. To make sure that after execution of the called subroutine the microprocessor knows where to come back, the microprocessor automatically saves the address of the instruction following the call on the stack. It must be noted that in the NEAR call only the IP is saved on the stack, and in a FAR call both CS and IP are saved. When a subroutine is called, control is transferred to that subroutine and the processor saves the IP (and CS in the case of a FAR call) and begins to fetch instructions from the new location. After finishing execution of the subroutine, for control to be transferred back to the caller, the last instruction in the called subroutine must be RET (return). In the same way that the assembler generates different opcode for FAR and NEAR calls, the opcode for the RET instruction in the case of NEAR and FAR is different, as well. For NEAR calls, the IP is restored; for FAR calls, both CS and IP are restored. This will ensure that control is given back to the caller. As an example, assume that SP = FFFEH and the following code is a portion of the program unassembled in DEBUG:

```
12B0:0200 BB1295  MOV BX,9512
12B0:0203 E8FA00  CALL 0300
12B0:0206 B82F14  MOV AX,142F
```