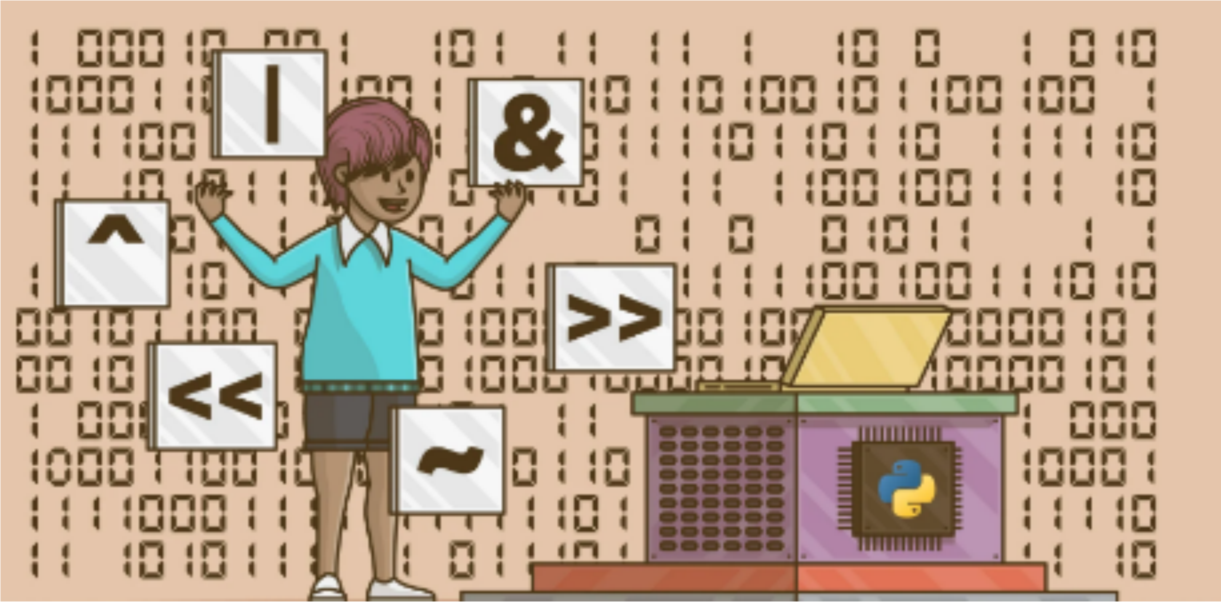
Extracting File Date Fields

Shifting and masking: The two most important operations used to extract bit strings are shifting and masking.

Shifting allows you to move the bit string to the desired position within a register, while **masking** allows you to clear any unwanted bits.

Using the AX register: The AX register is a convenient register to use for extracting bit strings, as it is 16 bits wide. This means that it can hold two 8-bit byte values.



This can be useful for extracting bit strings that are spread across two bytes, such as the month and day fields of a date stamp.

Storing the extracted bit strings: Once you have extracted the bit strings, you need to store them somewhere.

This can be done by copying them to other registers or to memory.

The following code snippet shows how to extract the day, month, and year fields of a date stamp integer stored in the DX register:

```
;Make a copy of DL and mask off bits not belonging to the day field.
mov     al, dl
and     al, 00011111b
mov     day, al

;Shift bits 5 through 8 into the low part of AL before masking off all other bits.
mov     ax, dx
shr     ax, 5
and     al, 00001111b
mov     month, al

;Copy the year field from DH to AL and shift right by 1 bit to clear AH.
mov     al, dh
shr     al, 1
mov     ah, 0
add     ax, 1980
mov     year, ax
```

This code snippet first makes a copy of the DL register to the AL register. Then, it masks off all bits except for the day field (bits

0 through 4). Finally, it copies the masked value to the day variable.

Next, the code snippet shifts bits 5 through 8 of the DX register into the low part of the AX register. Then, it masks off all bits except for the month field (bits 5 through 8). Finally, it copies the masked value to the month variable.

Finally, the code snippet copies the year field (bits 9 through 15) from the DH register to the AL register. Then, it shifts the value right by 1 bit to clear the AH register.

Finally, it adds 1980 to the value to account for the fact that the year field is relative to 1980. The code snippet then copies the final value to the year variable.

Once the day, month, and year fields have been extracted, they can be used for any purpose, such as displaying the date or calculating the number of days since the file was last modified.

----

1. Write assembly language instructions that calculate EAX * 24 using binary multiplication.

Here's how you can calculate **EAX * 24** in assembly language using binary multiplication:

```
mov ecx, 4      ; Initialize a counter for the number of bits to shift
mov ebx, eax    ; Make a copy of the original value in EAX
shl eax, 3      ; Multiply EAX by 2^3 (which is 8)
add eax, ebx    ; Multiply the result by 3 (24 = 8 * 3)
```

2. Write assembly language instructions that calculate EAX * 21 using binary multiplication.

Hint: 21 = 24 - 22 - 20.

To calculate EAX * 21, you can use binary multiplication based on the hint provided:

```
mov ebx, eax     ; Copy the original value to EBX
shl eax, 3       ; Multiply EAX by 8 (2^3)
sub ebx, eax     ; Subtract the original value by the result (EBX - EAX)
shl eax, 1       ; Multiply EAX by 2 (2^1)
add eax, ebx     ; Add the result to the previous result (EAX + EBX)
```

3. What change would you make to the BinToAsc procedure in Section 7.2.3 in order to display the binary bits in reverse order?

To display the binary bits in reverse order in the BinToAsc procedure, you can modify the loop that processes the bits. Instead of starting from the most significant bit (bit 31) and moving towards the least significant bit (bit 0), you can reverse the loop to start from the least significant bit and move towards the most significant bit. Here's a modified version of the BinToAsc procedure:

```
BinToAsc PROC
    pushad                       ; Preserve registers
    mov     edi, 31              ; Start from the least significant bit
    mov     ecx, 32              ; Loop through all 32 bits
    mov     esi, OFFSET outputStr ; Address of the output buffer

ConvertLoop:
    mov     al, [ebx + edi/8]    ; Load a byte from the binary data
    shl     al, cl               ; Shift the bit of interest to the lowest position
    and     al, 1                ; Mask all bits except the lowest one
    add     al, '0'              ; Convert the bit to its ASCII representation
    stosb                        ; Store the character in the output buffer
    loop    ConvertLoop
    mov     byte ptr [esi], 0    ; Null-terminate the output string
    popad                        ; Restore registers
    ret
BinToAsc ENDP
```

In this modified version, we start with the least significant bit (bit 0) and iterate through the bits in reverse order, which will display the binary bits in reverse.

4. The time stamp field of a file directory entry uses bits 0 through 4 for the seconds, bits 5 through 10 for the minutes, and bits 11 through 15 for the hours. Write instructions that extract the minutes and copy the value to a byte variable named bMinutes.

Here are the assembly instructions to extract the minutes from the time stamp and store the value in a byte variable named bMinutes:

```
mov     edx, [DirectoryEntryTime] ; Load the directory entry time stamp (assuming it's in edx)
and     edx, 0x07E0               ; Mask out the bits for minutes (5 through 10)
shr     edx, 5                    ; Shift the extracted minutes to the least significant bits
mov     byte ptr [bMinutes], dl   ; Store the extracted minutes in bMinutes
```

In this code, we use the and and shr instructions to isolate and shift the bits representing the minutes in the directory entry time stamp.

Finally, we store the extracted minutes in the bMinutes byte variable. Please replace [DirectoryEntryTime] with the actual address of the time stamp in your program.