Chapter 6

Display Memory

ASCII Codes

In computing, characters are translated into numbers. When you press a key on your keyboard, a specific number is sent to the computer, which interprets it as a character. This numeric representation is crucial for how computers understand and display characters.

- **What is ASCII?**: ASCII stands for American Standard Code for Information Interchange. It's a standardized system that assigns numbers to characters and symbols to ensure consistency in character representation across different computers and operating systems.
- **Universal Character Representation**: ASCII ensures that characters are universally understood. For example, **'A'** in ASCII is **'A'** on any computer or OS, making it a common language for character encoding.
- **Character Set**: Standard ASCII includes **128** characters, each with a unique number from **0** to **127**.
- **IBM PC Extension**: IBM extended ASCII by defining **128** additional characters for purposes like drawing lines and representing non-English characters.
- **Extended ASCII**: While not a formal standard like standard ASCII, the extended ASCII used in IBM PCs has become an industry practice. Peripherals and devices related to IBM PCs recognize these characters.
- **Character Arrangement**: The important thing to observe in the ASCII table is the contiguous arrangement of the uppercase alphabets **(41-5A)**, the lowercase alphabets **(61-7A)**, and the numbers **(30-39)**. This helps in certain operations with ASCII, for example converting the case of characters by adding or subtracting **0x20** from it. It also helps in converting a digit into its ASCII representation by adding **0x30** to it.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Display Memory Formation

Working with ASCII Codes:

- The VGA card in an 8088 processor-based computer is responsible for displaying visual information on the screen. It interprets commands and data sent to it to generate the display.
- To illustrate this, sending the value 0x40 to the VGA card might result in the display of the character **'A'** on the screen. This **'A'** is not a physical entity but an interpretation by the VGA card.

VGA Memory as a 2D Space:

- The VGA controller's memory is seen by the computer as a memory area containing ASCII codes currently displayed on the screen and a set of I/O ports for control.
- Importantly, the VGA memory is accessible to the processor like regular system memory. The computer doesn't differentiate between them, as they share the same memory bus.
- Consequently, changes made to the VGA memory directly affect what's displayed on the screen.

Mapping VGA Memory to 2D Screen:

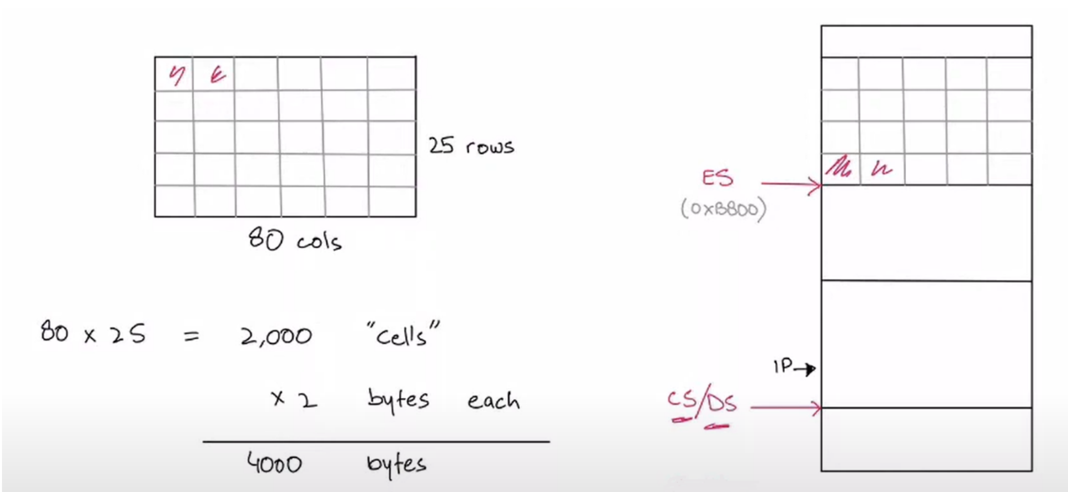
- While the memory is linear, the screen is a 2D space with rows and columns (e.g., **80 rows** and **25 columns**). VGA memory is mapped onto this 2D space linearly.
- Each character on the screen corresponds to one word in the video memory. The first 80 words in VGA memory correspond to the first row of the screen, the next 80 to the second row, and so on. There are

total 2000 words.

- For example, clearing a block of memory in the video controller will clear a corresponding portion of the screen.

Display Memory Base Address:

- To ensure compatibility across different video cards, the VGA memory base address is standardized. In this case, it's fixed at the physical memory location of **B8000**.
- The first byte at this location contains the ASCII code for the character displayed at the top left of the video screen.
- This address can be loaded into a segment register to access the video memory.

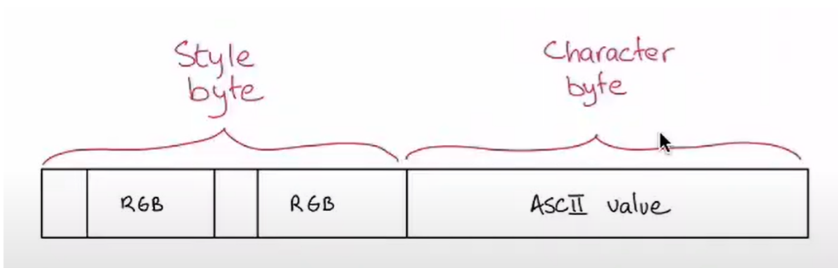


Attribute Byte:

- The second byte in the word designated for one screen location holds the character's video attribute, which includes foreground and background colors.
- It's composed of bits for specifying the colors and intensity of both foreground and background, along with options like blinking.
- Thus making **16** possible colors of the foreground and **8** possible colors for the background. When bit **7** is set the character keeps on blinking on the screen.

7	6	5	4	3	2	1	0

7 – Blinking of foreground character
6 – Red component of background color
5 – Green component of background color
4 – Blue component of background color
3 – Intensity component of foreground color
2 – Red component of foreground color
1 – Green component of foreground color
0 – Blue component of foreground color



Examples of Display Operations:

Both DS (Data Segment) and ES (Extra Segment) registers can be employed to access video memory in assembly language programming. However, it is a common practice to reserve DS for accessing general data segments, while ES is often loaded with the segment address of video memory. In the 8088 architecture, it's important to note that loading a segment register with an immediate operand, such as a constant, is not allowed. Therefore, segment registers are typically loaded via a general-purpose register or other methods.

For example, in the code below:

```
mov ax, 0xb800 ; Load the segment address of video memory into ax
mov es, ax     ; Set ES to point to the video memory
```

This sequence of instructions establishes a connection to the video memory. Subsequently, you can use instructions like the following to manipulate the display. In this case, the code prints the character **'A'** at the top-left corner of the screen in **white color** on a **black background**:

```
mov word [es:0], 0x0741
```

- The segment override `[es:]` is used because ES is pointing to the video memory.
- Since we're writing to the first word in memory, the character **'A'** will appear at the top-left of the screen.
- The value `0x0741` consists of two bytes: `0x41`, which is the ASCII code for **'A'**, and `0x07`, which represents the attribute. The attribute byte specifies the foreground color (white in low intensity) and the background

color (black with no blinking).

To illustrate further, consider the following instruction:

```
mov word [es:160], 0x1230
```

This command writes data 80 words (or characters) after the start. Since each row of the screen contains 80 characters, this data is displayed in the first column of the second row. In this case, the ASCII code used is `0x30`, representing **'0'**, while the attribute byte `0x12` signifies **green text** on a **blue background**.

Additionally, here's an example provided to clear the screen:

```
[org 0x0100]
mov ax, 0xb800  ; Load the segment address of video memory into ax
mov es, ax      ; Set ES to point to the video memory
mov di, 0       ; Initialize di to point to the top-left column

nextchar:
mov word [es:di], 0x0720 ; Clear the next character on the screen
add di, 2        ; Move to the next screen location (increment di by 2)
cmp di, 4000     ; Check if the entire screen has been cleared
jne nextchar     ; If not, clear the next position

mov ax, 0x4c00   ; Terminate the program
int 0x21
```

Here's a breakdown of this example:

- It clears the screen by setting every character to a blank character with the attribute for low-intensity white on a black background (ASCII code `0x20` and attribute `0x07`).
- The `DI` register is incremented by **2** after each character is cleared because each screen location corresponds to two bytes in video memory.
- The program checks if the entire screen has been cleared by comparing `DI` to `4000`, as there are 80 columns and 25 rows **(80 * 25 * 2 = 4000 bytes)** in the VGA screen memory.
- Finally, the program terminates with `0x4c00` (an exit code) using an interrupt (`int 0x21`).

When executed, this program clears the screen and returns to the command prompt.

“Hello World” in Assembly Language

In assembly language, declaring characters involves storing their ASCII codes in bytes. There are multiple syntax options provided by assemblers to simplify this process, including declaring consecutive characters as strings. The following three declarations are equivalent in meaning:

```
db 0x61, 0x62, 0x63
db 'a', 'b', 'c'
```

```
db 'abc'
```

In reality, whether characters are stored in high-level or low-level languages, the actual data stored in a byte is the ASCII code. The language syntax primarily simplifies the declaration process.

Traditionally, the first program in many high-level languages is the **"Hello, World!"** program, which prints this message on the screen. Due to the detailed nature of assembly language, we are now able to write a **"Hello, World!"** program in assembly. To achieve this, we create a generic routine that can print any string on the screen.

Here's an explanation of the key elements in the example program:

1. **String Declaration**:
   - The program starts by declaring the string to be displayed as `'hello world'`. It also calculates the length of this string, which is `11` characters.
2. **Subroutine to Clear the Screen**:
   - The `clrscr` subroutine is designed to clear the entire screen. It saves and restores registers (`es`, `ax`, and `di`) and performs screen clearing in a loop, ensuring that every character position is cleared.
3. **Subroutine to Print a String**:
   - The `printstr` subroutine is responsible for displaying a string at the top-left corner of the screen. It accepts two parameters: the address of the string and its length.
   - The subroutine uses pointer registers (`si` and `di`) to traverse the string and the screen memory. It also utilizes the `cx` register to control the loop and `ah` to set the attribute for the characters.
   - Characters from the string are loaded into `al`, and the pair of ASCII code and attribute is written to the video memory.
   - The loop instruction (`loop nextchar`) is used to repeat the process for each character in the string.
4. **Program Execution**:
   - The program starts by calling `clrscr` to clear the screen.
   - It then pushes the address of the message and its length onto the stack and calls `printstr` to display the message.
   - Finally, the program terminates with `int 0x21`.

This program demonstrates how to clear the screen and print a string on the screen using assembly language.

```
; Hello World in Assembly
[org 0x0100]
jmp start
message: db 'hello world' ; String to be printed
length: dw 11 ; Length of the string

; Subroutine to clear the screen
clrscr:
```

```
    push es
    push ax
    push di

    ; Remember that if this example is run in a DOS window on some newer
    ; operating systems, a full-screen DOS application must be run before this
    ; program so that screen access is enabled.

    mov ax, 0xb800
    mov es, ax ; Point es to video base
    mov di, 0 ; Point di to the top left column

nextloc:
    mov word [es:di], 0x0720 ; Clear the next character on the screen
    add di, 2 ; Move to the next screen location
    cmp di, 4000 ; Has the whole screen been cleared?
    jne nextloc ; If not, clear the next position

    pop di
    pop ax
    pop es
    ret

; Subroutine to print a string at the top left of the screen
; Takes the address of the string and its length as parameters
printstr:
    push bp
    mov bp, sp
    push es
    push ax
    push cx
    push si
    push di

    mov ax, 0xb800
    mov es, ax ; Point es to video base
    mov di, 0 ; Point di to the top left column
    mov si, [bp+6] ; Point si to the string
    mov cx, [bp+4] ; Load the length of the string into cx
    mov ah, 0x07 ; Normal attribute fixed in ah

nextchar:
    mov al, [si] ; Load the next character of the string
    mov [es:di], ax ; Show this character on the screen
    add di, 2 ; Move to the next screen location
```

```
    add si, 1 ; Move to the next character in the string
    loop nextchar ; Repeat the operation cx times

    pop di
    pop si
    pop cx
    pop ax
    pop es
    pop bp
    ret 4

start:
    call clrscr ; Call the clrscr subroutine
    mov ax, message
    push ax ; Push the address of the message
    push word [length] ; Push the message length
    call printstr ; Call the printstr subroutine
    mov ax, 0x4c00 ; Terminate the program
    int 0x21
```

Number Printing in Assembly

Number Printing Algorithm:

- The goal is to print a number in its ASCII representation. The algorithm begins by dividing the number by the specified base (e.g., 10 for decimal).
- The remainder from the division corresponds to the rightmost digit of the original number.
- The remainder is then converted into its ASCII representation (e.g., adding `0x30` to the remainder for decimal) and saved on the stack.
- The algorithm repeats the division process with the quotient as long as it's non-zero, saving each digit's ASCII representation on the stack.
- Finally, the digits are popped from the stack and printed on the screen from left to right.

DIV Instruction:

- The `DIV` instruction is used for integer division.
- There are two versions or forms of the DIV instruction:
  1. First Form: It divides a big **32-digit** number into two smaller parts called DX and AX. After dividing, it stores the answer (quotient) in AX and the leftover (remainder) in DX.
  2. Second Form: It takes a smaller **16-digit** number in AX and divides it by an even smaller 8-digit number. It stores the answer (quotient) in AL and the leftover (remainder) in AH.

Printnum Subroutine:

```
; Number Printing Algorithm
[org 0x0100]
jmp start

;;;;; COPY LINES 008-025 FROM EXAMPLE 6.2 (clrscr) ;;;;;
; Subroutine to print a number at the top left of the screen
; Input: The number to be printed as its parameter
printnum:
    push bp
    mov bp, sp
    push es
    push ax
    push bx
    push cx
    push dx
    push di

    ; Set ES to point to the video memory (0xb800)
    mov ax, 0xb800
    mov es, ax

    ; Load the number to be printed into AX
    mov ax, [bp+4]

    ; Use base 10 for division
    mov bx, 10

    ; Initialize the count of digits
    mov cx, 0

nextdigit:
    ; Clear the upper half of DX
    mov dx, 0

    ; Divide AX by 10
    div bx

    ; Convert the digit into its ASCII value
    add dl, 0x30

    ; Save the ASCII value on the stack
    push dx

    ; Increment the count of values
```

```
    inc cx

    ; Check if the quotient (AX) is zero
    cmp ax, 0
    jnz nextdigit ; If not, divide again

    ; Set DI to point to the top left column
    mov di, 0

nextpos:
    ; Remove a digit from the stack
    pop dx

    ; Use the normal attribute
    mov dh, 0x07

    ; Print the character on the screen
    mov [es:di], dx

    ; Move to the next screen location
    add di, 2

    ; Repeat for all digits on the stack
    loop nextpos

    ; Clean up the stack
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp

    ; Return from the subroutine
    ret 2

start:
    ; Call the clrscr subroutine to clear the screen
    call clrscr

    ; Load the number 4529 into AX
    mov ax, 4529

    ; Place the number on the stack
```

```
    push ax

    ; Call the printnum subroutine to print the number
    call printnum

    ; Terminate the program
    mov ax, 0x4c00
    int 0x21
```

This program provides a flexible number printing algorithm that can be used to display numbers in various bases. You can adjust the `bx` register to specify the desired base for conversion (e.g., 10 for decimal, 2 for binary, 8 for octal, or 16 for hexadecimal).

Screen Location Calculation

We want to be able to print text at any spot on the screen with any color. To do this, we need to figure out where that spot is in memory. Imagine the screen as a grid with rows and columns. The screen has 80 columns in each row. So, we need to convert the row and column coordinates into a memory location.

MUL Instruction:

MUL (multiply) performs an unsigned multiplication of the source operand and the accumulator. If the source operand is a byte, then it is multiplied by register **AL** and the double-length result is returned in **AH** and **AL**. If the source operand is a word, then it is multiplied by register **AX**, and the double-length result is returned in registers **DX** and **AX**.

Printing at Desired Location:

- We adjust our string printing program to accept the X-position (horizontal), Y-position (vertical), the desired text attribute (like color), the address of the string, and the length of the string as inputs.
- We use MUL to calculate the memory location where we want to put our text. This location is like a memory address where the screen starts.
- **location = (rows * 80 + columns) * 2**
- Then, we go through each character of the string and put it on the screen at that calculated location with the chosen attribute.
- This way, we can print text anywhere on the screen with any color.

Example:

```
; hello world at desired screen location
[org 0x0100]
jmp start
message: db 'hello world' ; string to be printed
length: dw 11 ; length of the string
;;;;; COPY LINES 008-025 FROM EXAMPLE 6.2 (clrscr) ;;;;;
```

```
; subroutine to print a string at top left of screen
; takes x position, y position, string attribute, address of string
; and its length as parameters
printstr:
    push bp
    mov bp, sp
    push es
    push ax
    push cx
    push si
    push di
    mov ax, 0xb800
    mov es, ax ; point es to video base
    mov al, 80 ; load al with columns per row
    mul byte [bp+10] ; multiply with y position
    add ax, [bp+12] ; add x position
    shl ax, 1 ; turn into byte offset
    mov di, ax ; point di to required location
    mov si, [bp+6] ; point si to string
    mov cx, [bp+4] ; load length of string in cx
    mov ah, [bp+8] ; load attribute in ah
nextchar:
    mov al, [si] ; load next char of string
    mov [es:di], ax ; show this char on screen
    add di, 2 ; move to next screen location
    add si, 1 ; move to next char in string
    loop nextchar ; repeat the operation cx times
    pop di
    pop si
    pop cx
    pop ax
    pop es
    pop bp
    ret 10

start:
    call clrscr ; call the clrscr subroutine
    mov ax, 30
    push ax ; push x position
    mov ax, 20
    push ax ; push y position
    mov ax, 1 ; blue on black attribute
    push ax ; push attribute
    mov ax, message
    push ax ; push address of message
```

```
    push word [length] ; push message length
    call printstr ; call the printstr subroutine
    mov ax, 0x4c00 ; terminate program
    int 0x21
```

Color Scheme

```
Hex   Binary       Meaning
01    00000001     Normal Intensity, No Blink, Blue Text on Black Background
02    00000010     Normal Intensity, No Blink, Green Text on Black Background
03    00000011     Normal Intensity, No Blink, Cyan Text on Black Background
04    00000100     Normal Intensity, No Blink, Red Text on Black Background
05    00000101     Normal Intensity, No Blink, Magenta Text on Black Background
06    00000110     Normal Intensity, No Blink, Yellow Text on Black Background
07    00000111     Normal Intensity, No Blink, White Text on Black Background
08    00001000     Normal Intensity, No Blink, Black Text on Black Background
09    00001001     High Intensity, No Blink, Blue Text on Black Background
0A    00001010     High Intensity, No Blink, Green Text on Black Background
0B    00001011     High Intensity, No Blink, Cyan Text on Black Background
0C    00001100     High Intensity, No Blink, Red Text on Black Background
0D    00001101     High Intensity, No Blink, Magenta Text on Black Background
0E    00001110     High Intensity, No Blink, Yellow Text on Black Background
0F    00001111     High Intensity, No Blink, White Text on Black Background
10    00010000     High Intensity, No Blink, Black Text on Blue Background
11    00010001     Normal Intensity, No Blink, Blue Text on Blue Background
12    00010010     Normal Intensity, No Blink, Green Text on Blue Background
13    00010011     Normal Intensity, No Blink, Cyan Text on Blue Background
14    00010100     Normal Intensity, No Blink, Red Text on Blue Background
15    00010101     Normal Intensity, No Blink, Magenta Text on Blue Background
16    00010110     Normal Intensity, No Blink, Yellow Text on Blue Background
17    00010111     Normal Intensity, No Blink, White Text on Blue Background
18    00011000     Normal Intensity, No Blink, Black Text on Blue Background
19    00011001     High Intensity, No Blink, Blue Text on Blue Background
1A    00011010     High Intensity, No Blink, Green Text on Blue Background
1B    00011011     High Intensity, No Blink, Cyan Text on Blue Background
1C    00011100     High Intensity, No Blink, Red Text on Blue Background
1D    00011101     High Intensity, No Blink, Magenta Text on Blue Background
1E    00011110     High Intensity, No Blink, Yellow Text on Blue Background
1F    00011111     High Intensity, No Blink, White Text on Blue Background
20    00100000     High Intensity, No Blink, Black Text on Green Background
21    00100001     Normal Intensity, No Blink, Blue Text on Green Background
22    00100010     Normal Intensity, No Blink, Green Text on Green Background
23    00100011     Normal Intensity, No Blink, Cyan Text on Green Background
24    00100100     Normal Intensity, No Blink, Red Text on Green Background
```

25	00100101	Normal Intensity, No Blink, Magenta Text on Green Background
26	00100110	Normal Intensity, No Blink, Yellow Text on Green Background
27	00100111	Normal Intensity, No Blink, White Text on Green Background
28	00101000	Normal Intensity, No Blink, Black Text on Green Background
29	00101001	High Intensity, No Blink, Blue Text on Green Background
2A	00101010	High Intensity, No Blink, Green Text on Green Background
2B	00101011	High Intensity, No Blink, Cyan Text on Green Background
2C	00101100	High Intensity, No Blink, Red Text on Green Background
2D	00101101	High Intensity, No Blink, Magenta Text on Green Background
2E	00101110	High Intensity, No Blink, Yellow Text on Green Background
2F	00101111	High Intensity, No Blink, White Text on Green Background
30	00110000	High Intensity, No Blink, Black Text on Cyan Background
31	00110001	Normal Intensity, No Blink, Blue Text on Cyan Background
32	00110010	Normal Intensity, No Blink, Green Text on Cyan Background
33	00110011	Normal Intensity, No Blink, Cyan Text on Cyan Background
34	00110100	Normal Intensity, No Blink, Red Text on Cyan Background
35	00110101	Normal Intensity, No Blink, Magenta Text on Cyan Background
36	00110110	Normal Intensity, No Blink, Yellow Text on Cyan Background
37	00110111	Normal Intensity, No Blink, White Text on Cyan Background
38	00111000	Normal Intensity, No Blink, Black Text on Cyan Background
39	00111001	High Intensity, No Blink, Blue Text on Cyan Background
3A	00111010	High Intensity, No Blink, Green Text on Cyan Background
3B	00111011	High Intensity, No Blink, Cyan Text on Cyan Background
3C	00111100	High Intensity, No Blink, Red Text on Cyan Background
3D	00111101	High Intensity, No Blink, Magenta Text on Cyan Background
3E	00111110	High Intensity, No Blink, Yellow Text on Cyan Background
3F	00111111	High Intensity, No Blink, White Text on Cyan Background
40	01000000	High Intensity, No Blink, Black Text on Red Background
41	01000001	Normal Intensity, No Blink, Blue Text on Red Background
42	01000010	Normal Intensity, No Blink, Green Text on Red Background
43	01000011	Normal Intensity, No Blink, Cyan Text on Red Background
44	01000100	Normal Intensity, No Blink, Red Text on Red Background
45	01000101	Normal Intensity, No Blink, Magenta Text on Red Background
46	01000110	Normal Intensity, No Blink, Yellow Text on Red Background
47	01000111	Normal Intensity, No Blink, White Text on Red Background
48	01001000	Normal Intensity, No Blink, Black Text on Red Background
49	01001001	High Intensity, No Blink, Blue Text on Red Background
4A	01001010	High Intensity, No Blink, Green Text on Red Background
4B	01001011	High Intensity, No Blink, Cyan Text on Red Background
4C	01001100	High Intensity, No Blink, Red Text on Red Background
4D	01001101	High Intensity, No Blink, Magenta Text on Red Background
4E	01001110	High Intensity, No Blink, Yellow Text on Red Background
4F	01001111	High Intensity, No Blink, White Text on Red Background
50	01010000	High Intensity, No Blink, Black Text on Magenta Background
51	01010001	Normal Intensity, No Blink, Blue Text on Magenta Background

```
52    01010010     Normal Intensity, No Blink, Green Text on Magenta Background
53    01010011     Normal Intensity, No Blink, Cyan Text on Magenta Background
54    01010100     Normal Intensity, No Blink, Red Text on Magenta Background
55    01010101     Normal Intensity, No Blink, Magenta Text on Magenta Backgrour
56    01010110     Normal Intensity, No Blink, Yellow Text on Magenta Backgrounc
57    01010111     Normal Intensity, No Blink, White Text on Magenta Background
58    01011000     Normal Intensity, No Blink, Black Text on Magenta Background
59    01011001     High Intensity, No Blink, Blue Text on Magenta Background
5A    01011010     High Intensity, No Blink, Green Text on Magenta Background
5B    01011011     High Intensity, No Blink, Cyan Text on Magenta Background
5C    01011100     High Intensity, No Blink, Red Text on Magenta Background
5D    01011101     High Intensity, No Blink, Magenta Text on Magenta Background
5E    01011110     High Intensity, No Blink, Yellow Text on Magenta Background
5F    01011111     High Intensity, No Blink, White Text on Magenta Background
60    01100000     High Intensity, No Blink, Black Text on Yellow Background
61    01100001     Normal Intensity, No Blink, Blue Text on Yellow Background
62    01100010     Normal Intensity, No Blink, Green Text on Yellow Background
63    01100011     Normal Intensity, No Blink, Cyan Text on Yellow Background
64    01100100     Normal Intensity, No Blink, Red Text on Yellow Background
65    01100101     Normal Intensity, No Blink, Magenta Text on Yellow Backgrounc
66    01100110     Normal Intensity, No Blink, Yellow Text on Yellow Background
67    01100111     Normal Intensity, No Blink, White Text on Yellow Background
68    01101000     Normal Intensity, No Blink, Black Text on Yellow Background
69    01101001     High Intensity, No Blink, Blue Text on Yellow Background
6A    01101010     High Intensity, No Blink, Green Text on Yellow Background
6B    01101011     High Intensity, No Blink, Cyan Text on Yellow Background
6C    01101100     High Intensity, No Blink, Red Text on Yellow Background
6D    01101101     High Intensity, No Blink, Magenta Text on Yellow Background
6E    01101110     High Intensity, No Blink, Yellow Text on Yellow Background
6F    01101111     High Intensity, No Blink, White Text on Yellow Background
70    01110000     High Intensity, No Blink, Black Text on White Background
71    01110001     Normal Intensity, No Blink, Blue Text on White Background
72    01110010     Normal Intensity, No Blink, Green Text on White Background
73    01110011     Normal Intensity, No Blink, Cyan Text on White Background
74    01110100     Normal Intensity, No Blink, Red Text on White Background
75    01110101     Normal Intensity, No Blink, Magenta Text on White Background
76    01110110     Normal Intensity, No Blink, Yellow Text on White Background
77    01110111     Normal Intensity, No Blink, White Text on White Background
78    01111000     Normal Intensity, No Blink, Black Text on White Background
79    01111001     High Intensity, No Blink, Blue Text on White Background
7A    01111010     High Intensity, No Blink, Green Text on White Background
7B    01111011     High Intensity, No Blink, Cyan Text on White Background
7C    01111100     High Intensity, No Blink, Red Text on White Background
7D    01111101     High Intensity, No Blink, Magenta Text on White Background
7E    01111110     High Intensity, No Blink, Yellow Text on White Background
```

```
7F    01111111     High Intensity, No Blink, White Text on White Background


In case of Blink, the 8th bit will be set to 1.
80    10000000     High Intensity, Blink, Black Text on Black Background
```