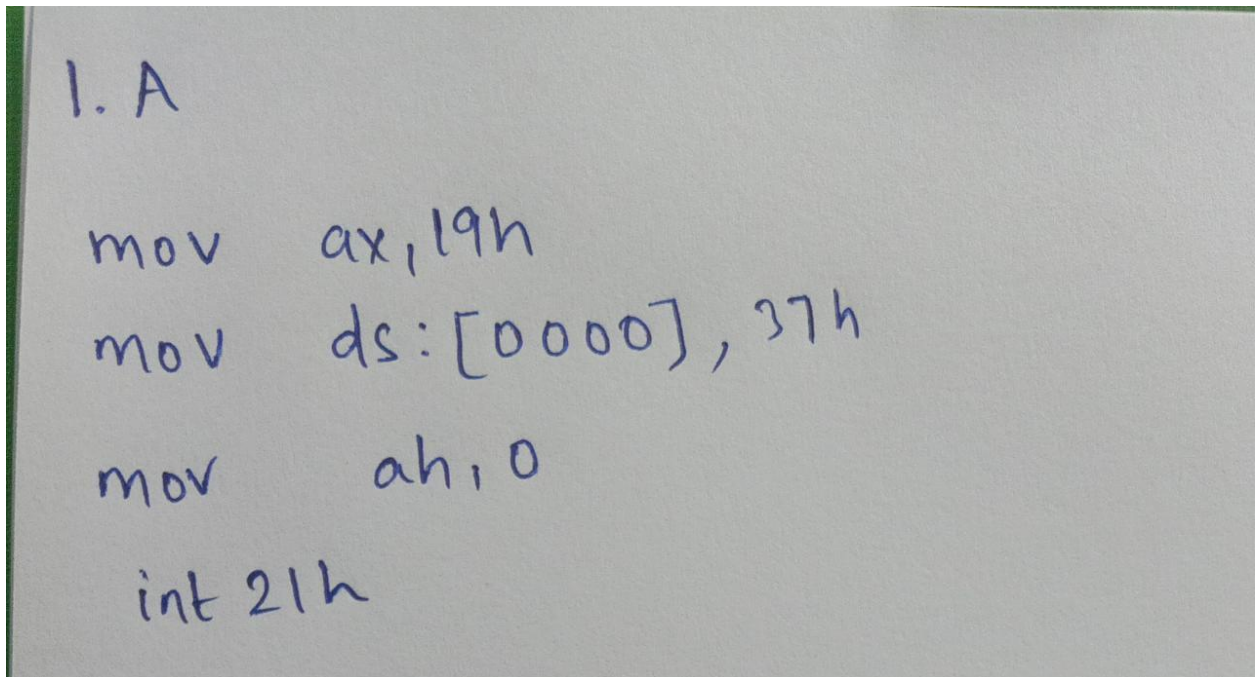
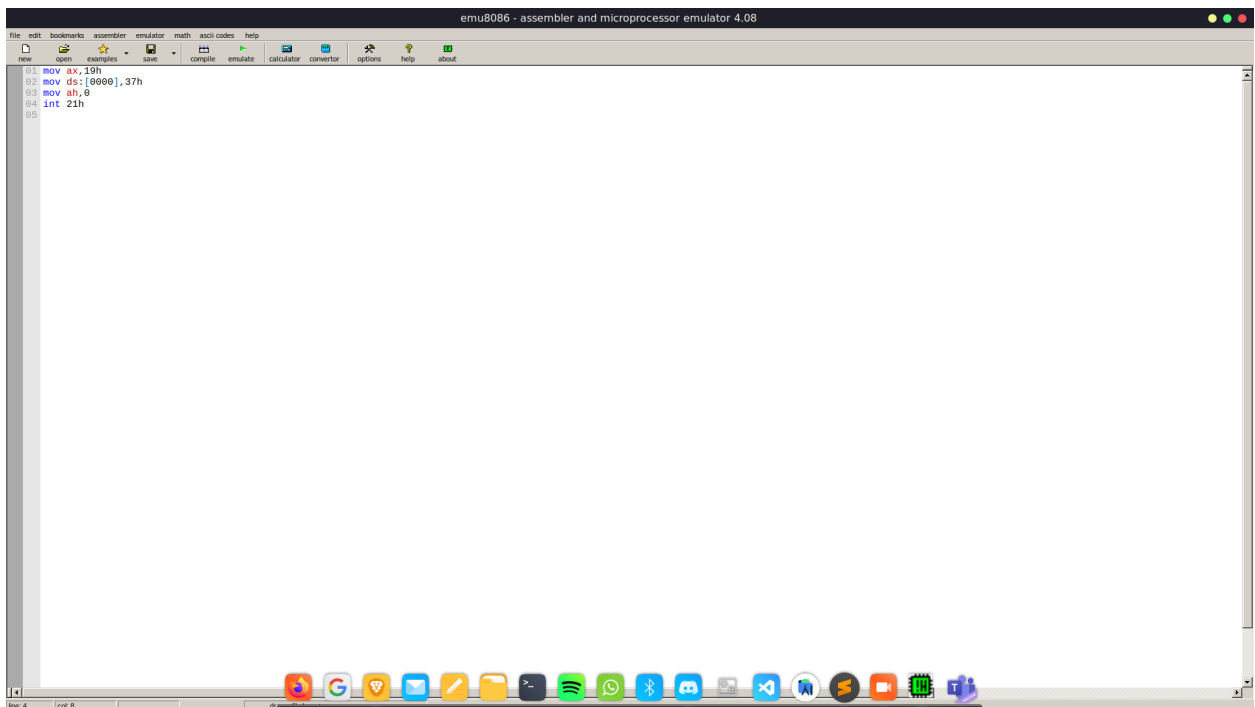


1.A

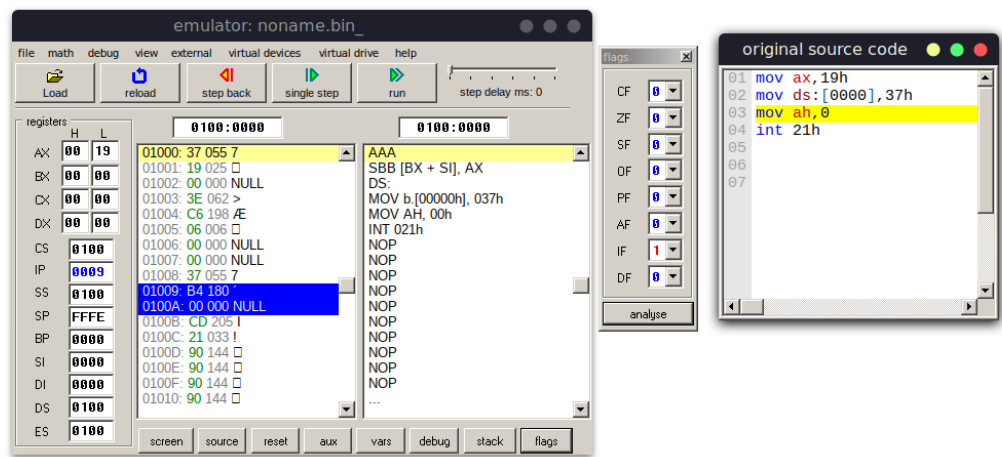
Handwritten Code:



Code:



Output:



1.B

Addition

Handwritten Code:

1. B

Addition

data_new segment

data1 dw 19h

data2 dw 37h

data3 dw 1 dup <?>

data_new ends

code segment

assume cs: code, ds: data_new

start:

mov ax, data_new

mov ds, ax

mov ax, data1

mov bx, data2

add ax, bx

mov data3, ax

code ends

end start

end

Code:

```

01 data_new segment
02     data1 dw 19h
03     data2 dw 37h
04     data3 dw 1 dup<?>
05 data_new ends
06
07 code segment
08     assume cs:code,ds:data_new
09     start:
10     mov ax,data_new
11     mov ds,ax
12
13     mov ax,data1
14     mov bx,data2
15     add ax,bx
16     mov data3,ax
17
18
19 code ends
20 end start
21 end
22

```

Output:

Subtraction

Handwritten Code: Code:

```

emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options help about
01 data_new segment
02 data1 dw 19h
03 data2 dw 37h
04 data3 dw 1 dup<?>
05 data_new ends
06
07 code segment
08 assume cs:code,ds:data_new
09 start:
10 mov ax,data_new
11 mov ds,ax
12
13 mov ax,data1
14 mov bx,data2
15 sub ax,bx
16 mov data3,ax
17
18
19 code ends
20 end start
21 end
22

```

Output:

```

emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options help about
01 data_new segment
02 data1 dw 19h
03 data2 dw 37h
04 data3 dw 1 dup<?>
05 data_new ends
06
07 code segment
08 assume cs:code,ds:data_new
09 start:
10 mov ax,data_new
11 mov ds,ax
12
13 mov ax,data1
14 mov bx,data2
15 sub ax,bx
16 mov data3,ax
17
18
19 code ends
20 end start
21 end
22

```

Registers window (Address: 0711:0025):

Register	Value	Comment
AX	FF E2	07132: 90 144 □
BX	00 37	07133: 90 144 □
CX	00 21	07134: 90 144 □
DX	00 00	07135: 00 000 NULL
SI	0711	07136: 00 000 NULL
SP	0025	07137: 00 000 NULL
BP	0000	07138: 00 000 NULL
IP	0710	07139: 00 000 NULL
ES	0710	0713A: 00 000 NULL
DS	0710	0713B: 00 000 NULL
FS	0710	0713C: 00 000 NULL
GS	0710	0713D: 00 000 NULL
SI	0000	0713E: 00 000 NULL
DI	0000	0713F: 00 000 NULL
DS	0000	07140: 00 000 NULL
ES	0710	07141: 00 000 NULL
DS	0710	07142: 00 000 NULL

Original source code window:

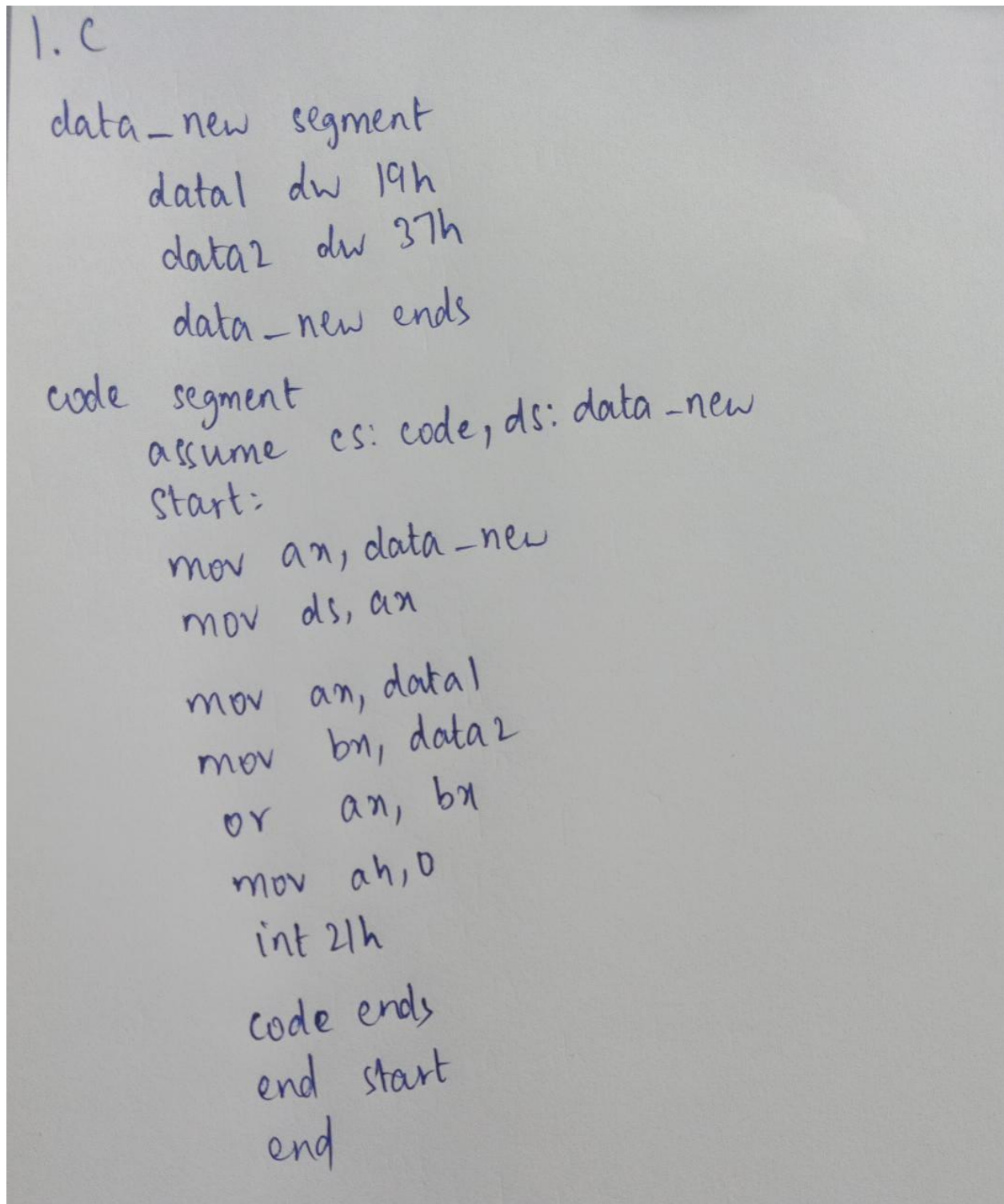
```

05 data3 dw 1 dup<?>
06 data_new ends
07
08 code segment
09 assume cs:code,ds:data_n
10 start:
11 mov ax,data_new
12 mov ds,ax
13
14 mov ax,data1
15 mov bx,data2
16 sub ax,bx
17 mov data3,ax
18
19
20 code ends
21 end start
22 end

```

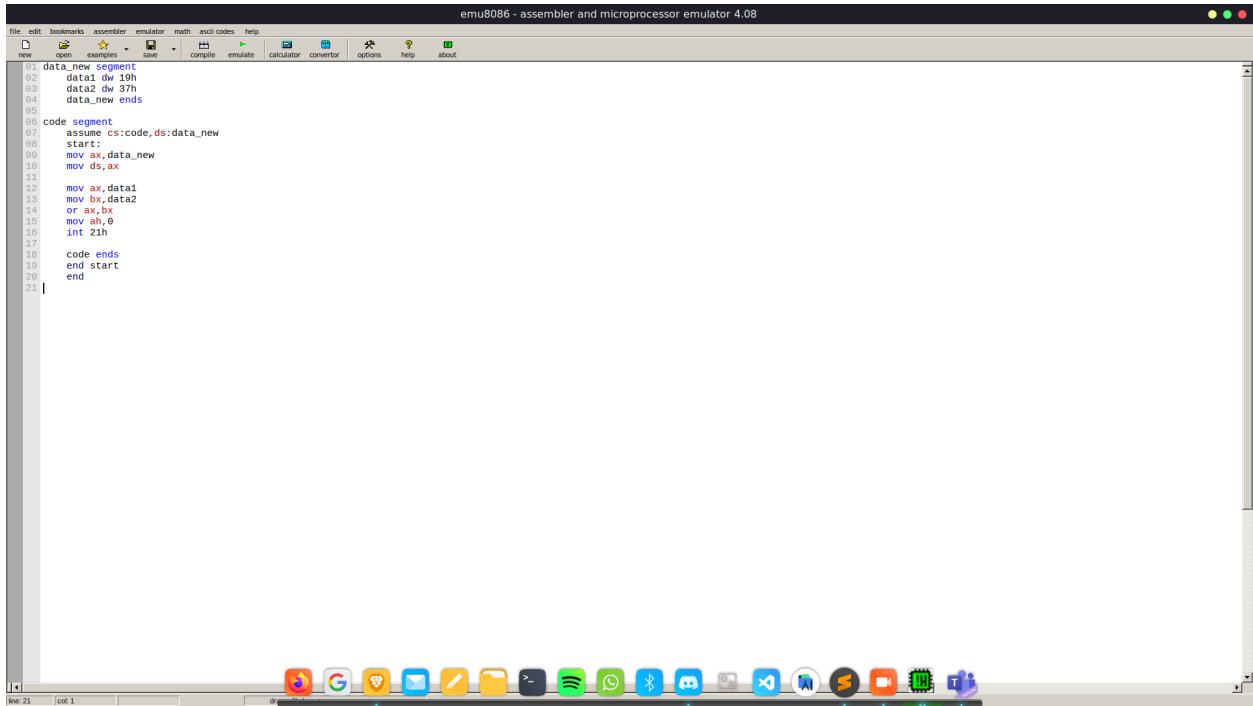
1.C

Handwritten Code:

A photograph of a piece of paper with handwritten assembly code in blue ink. The code is organized into two segments: 'data_new' and 'code'. The 'data_new' segment defines two data words, 'data1' (19h) and 'data2' (37h). The 'code' segment starts with an 'assume' statement for CS and DS, followed by a 'start' label. The code then moves the address of 'data_new' into AX, sets DS to AX, moves the values of 'data1' and 'data2' into AX and BX respectively, performs a bitwise OR of AX and BX, sets AH to 0, and triggers an interrupt 21h. The code concludes with 'code ends', 'end start', and 'end'.

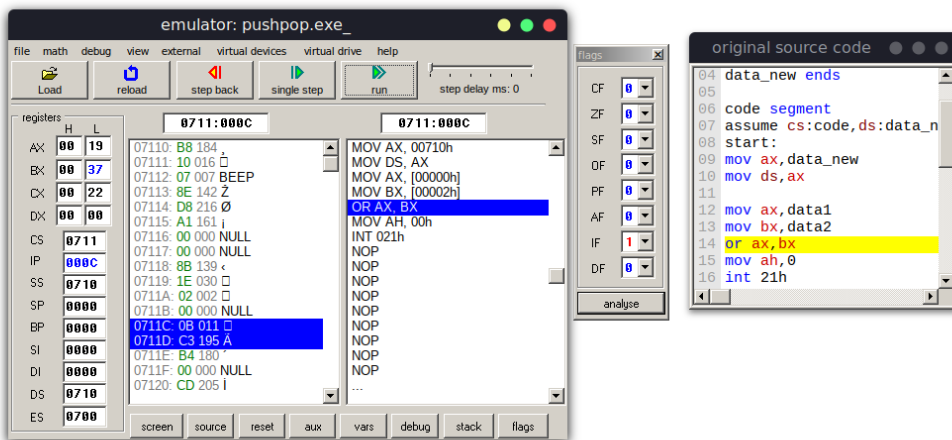
Code:

19BCE0637

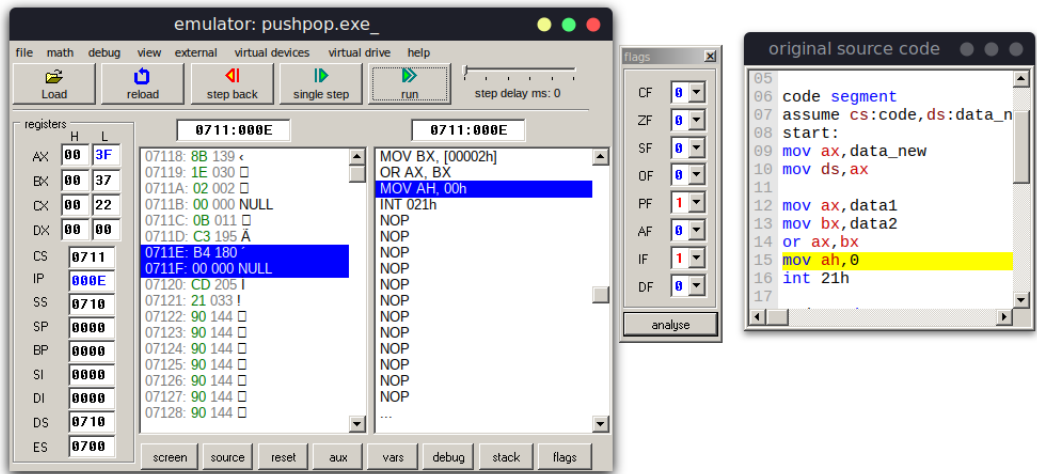


Output:

WEST VIEW



ta_new



XOR

Handwritten Code:

1. c

data_new segment

data1 dw 19h

data2 dw 37h

data_new ends

code segment

assume cs: code, ds: data_new

start:

mov ax, data_new

mov ds, ax

mov ax, data1

mov bx, data2

xor ax, bx

mov ah, 0

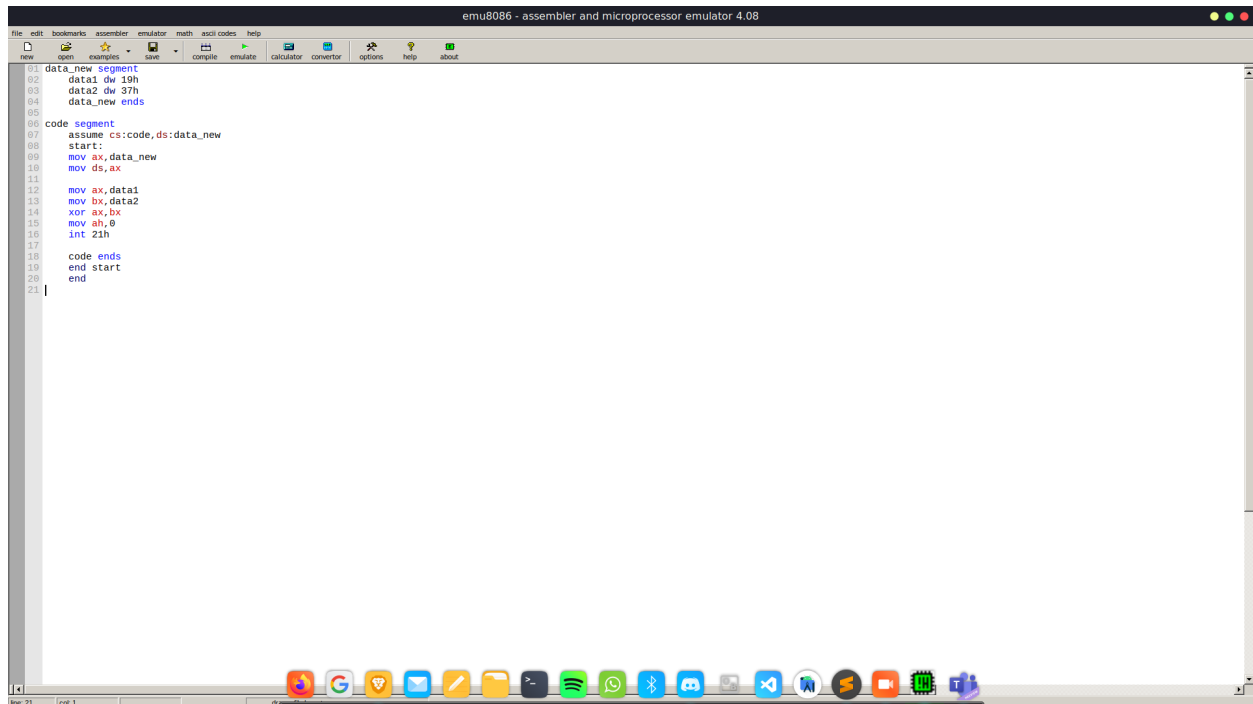
int 21h

code ends

end start

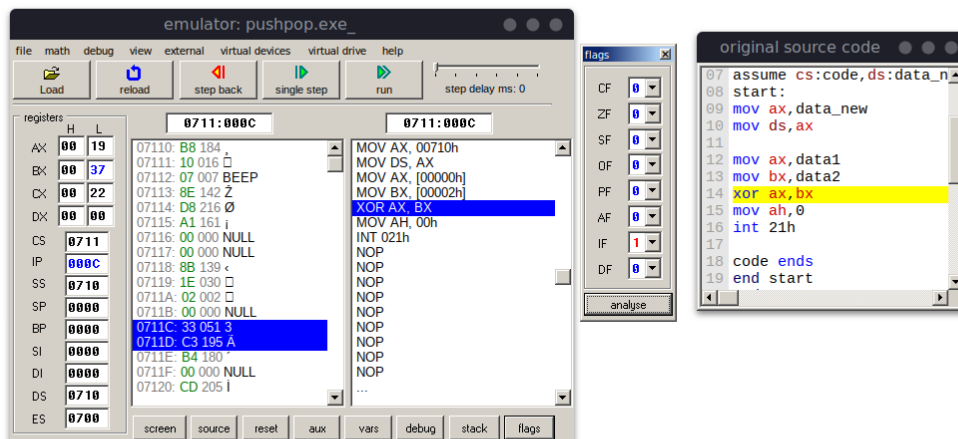
end

Code:

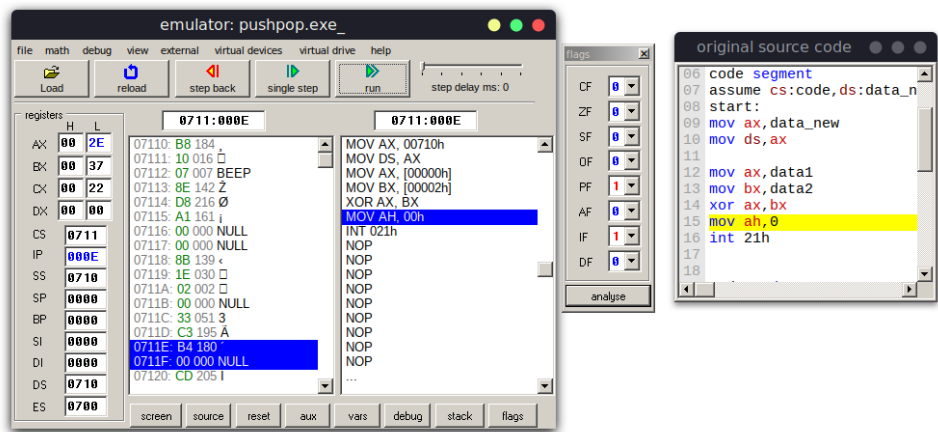


Output:

data_new



s:data_new



1.D

Handwritten Code:

1. D

```
data_new segment
    data1 dw 19h
    data2 dw 37h
    data3 dw 1 dup<?>
    data4 dw 50h
data_new ends
```

```
code segment
```

```
    assume cs:code, ds:data_new
```

```
start:
```

```
    mov ax, data_new
```

```
    mov ds, ax
```

```
    mov cx, data1
```

```
    mov bx, data2
```

```
    add cx, bx
```

```
    cmp cx, data4
```

```
    jle lesseq
```

```
    jg isgreater
```

```
isgreater: inc cx
```

```
lesseq: dec cx
```

```
    mov ah, 0
```

```
    int 21h
```

```
code ends
```

Code:

```

emu8086 - assembler and microprocessor emulator 4.08
File Edit Bookmarks Assembler Emulator Math ASCII Codes Help
new open examples save compile emulate calculator converter options help about
01 data_new segment
02 data1 dw 10h
03 data2 dw 37h
04 data3 dw 1 dup<?>
05 data4 dw 50h
06 data_new ends
07
08
09 code segment
10 assume cs:code,ds:data_new
11 start:
12 mov ax,data_new
13 mov ds,ax
14
15 mov ax,data1
16 mov bx,data2
17 add ax,bx
18
19 cmp ax,data4
20
21 jle lesseq
22 jg isgreater
23
24 isgreater:inc ax
25 lesseq:dec ax
26
27 mov ah,0
28 int 21h
29
30 code ends
31 end start
32 end

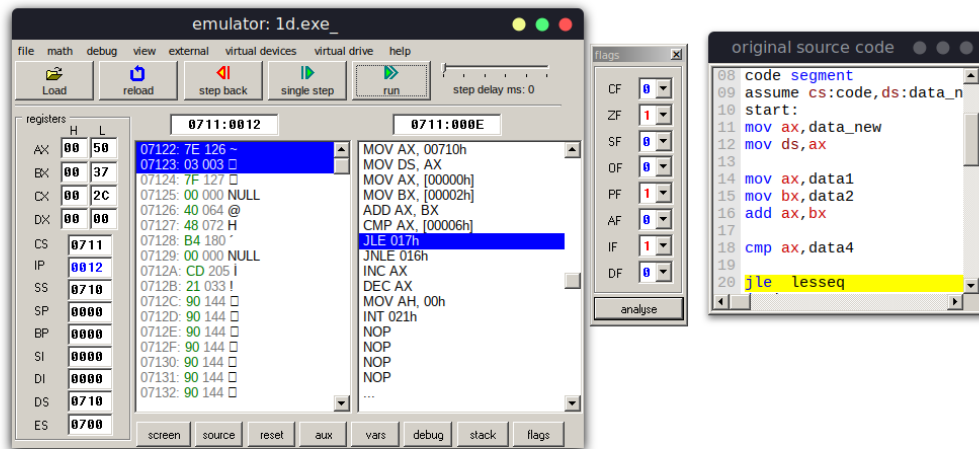
```

Output:

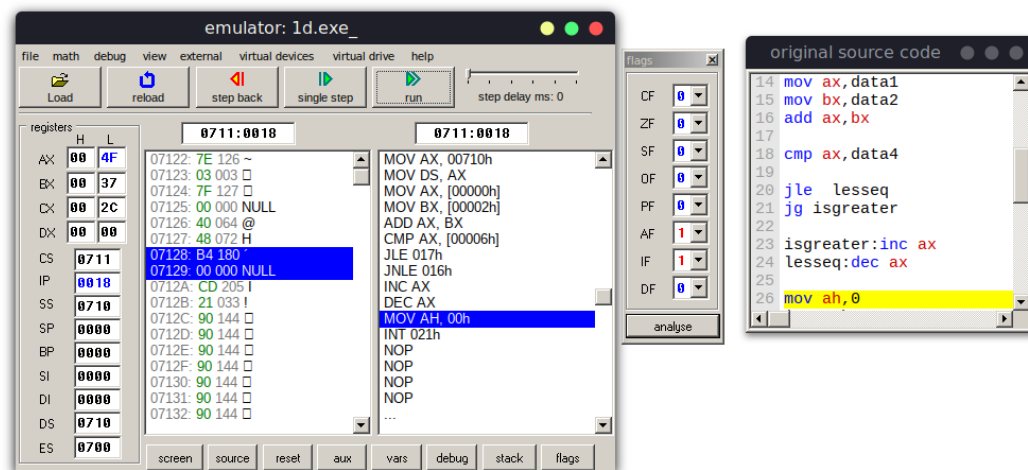
Is:data_new

The screenshot displays the emu8086 emulator interface. The main window shows the assembly code being executed, with the instruction `cmp ax,data4` highlighted in yellow. The registers window on the left shows the current state of the CPU registers, with the `AX` register containing `0050` and the `DS` register containing `0710`. The flags window on the right shows the status of various flags, including `CF`, `ZF`, `SF`, `OF`, `PF`, `AF`, `IF`, and `DF`. The assembly code window on the right shows the original source code, with the instruction `cmp ax,data4` highlighted in yellow.

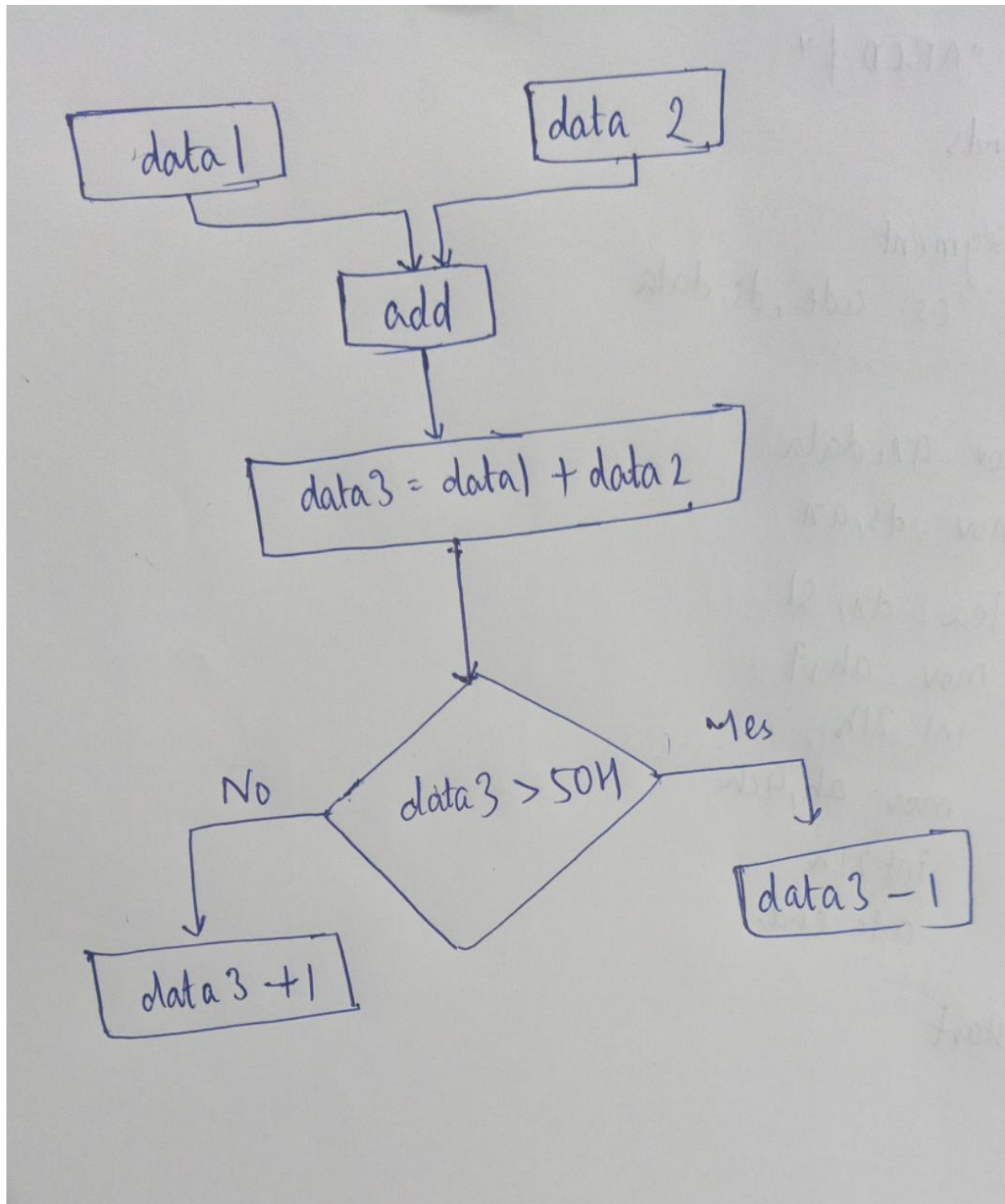
lata_new



lata_new



Flowchart:



1.E

Handwritten Code:

1. E

```
data_new segment  
    data1 dw 19h
```

```
code segment  
    assume cs:code, ds:data_new
```

```
start:
```

```
    mov ax, data_new  
    mov ds, ax
```

```
    mov ax, data1
```

```
    push ax
```

```
    pop ax
```

```
    mov ah, 0
```

```
    int 21h
```

```
code ends
```

```
end start
```

```
end
```

Code:

```

01 data_new segment
02     data1 dw 19h
03
04 code segment
05     assume cs:code,ds:data_new
06     start:
07     mov ax,data_new
08     mov ds,ax
09
10     mov ax,data1
11     push ax
12     pop ax
13
14     mov ah,0
15     int 21h
16
17 code ends
18 end start
19 end
20
21

```

Output:

emulator: pushpop.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers H L

AX	00	19
BX	00	00
CX	00	1A
DX	00	00
CS	0711	
IP	0009	
SS	0710	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0710	
ES	0700	

07110: B8 184 MOV AX, 00710h
 07111: 10 016 MOV DS, AX
 07112: 07 007 BEEP MOV AX, [000000h]
 07113: 8E 142 Z PUSH AX
 07114: D8 216 Ø POP AX
 07115: A1 161 j NOP
 07116: 00 000 NULL NOP
 07117: 00 000 NULL NOP
 07118: 50 080 P NOP
 07119: 53 088 X NOP
 0711A: 90 144 □ NOP
 0711B: 90 144 □ NOP
 0711C: 90 144 □ NOP
 0711D: 90 144 □ NOP
 0711E: 90 144 □ NOP
 0711F: 90 144 □ NOP
 07120: 90 144 □ ...

flags

CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

analyse

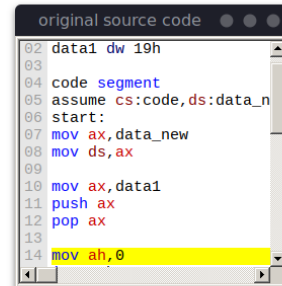
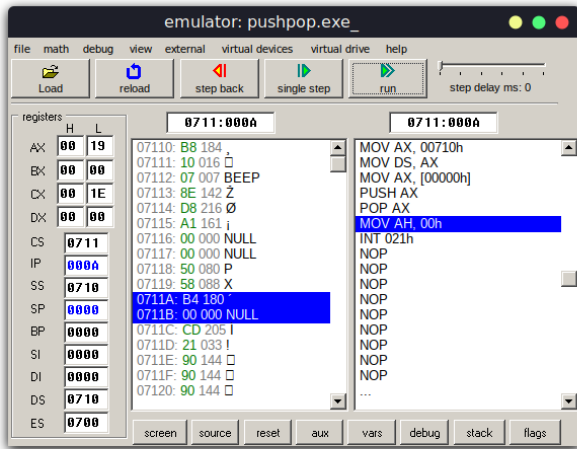
original source code

```

01 data_new segment
02     data1 dw 19h
03
04 code segment
05     assume cs:code,ds:data_n
06     start:
07     mov ax,data_new
08     mov ds,ax
09
10     mov ax,data1
11     push ax
12     pop ax
13

```

screen source reset aux vars debug stack flags



1.F

Handwritten Code:

1. F

data segment

```

s1 db "ABCD$"
l1 db $-s1
s2 db "abcd$"
l2 db $-s2
streq db "equal$"
struneq db "unequal$"

```

data ends

code segment

assume cs:code, ds:data

begin:

```

mov ax, data
mov ds, ax
mov es, ax
lea si, s1
lea di, s2
mov cx, 6
mov al, l1
mov bl, l2
cmp al, bl
jne ne

```

```

repe cmpsb
jne ne
jmp e

```

ne:

```

mov ah, 09h
lea dx, struneq
int 21h
jmp exit

```

e:

```

mov ah, 09h
lea dx, streq
int 21h

```

exit:

```

mov ax, 4c00h
int 21h

```

code ends

end begin

Code:

```

11 data segment
12     si db "ABCD$"
13     li db $-si
14     sz db "abcd$"
15     lz db $-sz
16     streq db "equals"
17     struneq db "unequal$"
18 data ends
19
20 code segment
21     Assume cs:code, ds:data
22     begin:
23         mov ax, data
24         mov ds, ax
25         mov es, ax
26         lea si, si
27         lea di, sz
28         mov cx, 0
29         mov al, li
30         mov bl, lz
31         cmp al, bl
32         jne ne
33
34         repe cmpsb
35         jne ne
36         jmp e
37
38     ne:
39         mov ah, 09h
40         lea dx, struneq
41         int 21h
42         jmp exit
43
44     e:
45         mov ah, 09h
46         lea dx, streq
47         int 21h
48     exit:
49         mov ax, 4c00h
50         int 21h
51 code ends
52 end begin

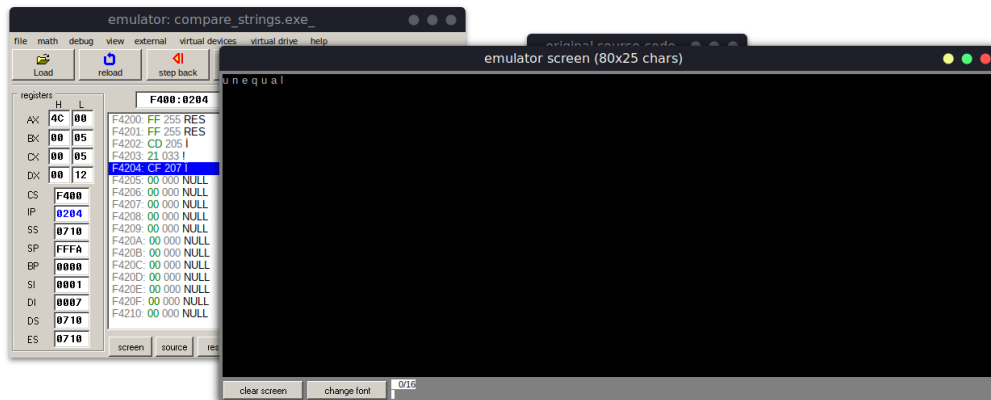
```

Output:

\$"

data

q

**Handwritten Code:**

1. F

data segment

si db "ABCD\$"

data ends

code segment

assume cs: code, ds: data

start:

mov ax, data

mov ds, ax

lea dx, si

mov ah, 9

int 21h

mov ah, 4ch

int 21h

code ends

end start

Code:

```

01  data segment
02
03  s1 db "ABCD$"
04  data ends
05
06  code segment
07  assume cs:code,ds:data
08  start:
09  mov ax,data
10  mov ds,ax
11
12  lea dx,s1
13  mov ah,9
14  int 21h
15
16  mov ah,4ch
17  int 21h
18  code ends
19
20 end start

```

Output: