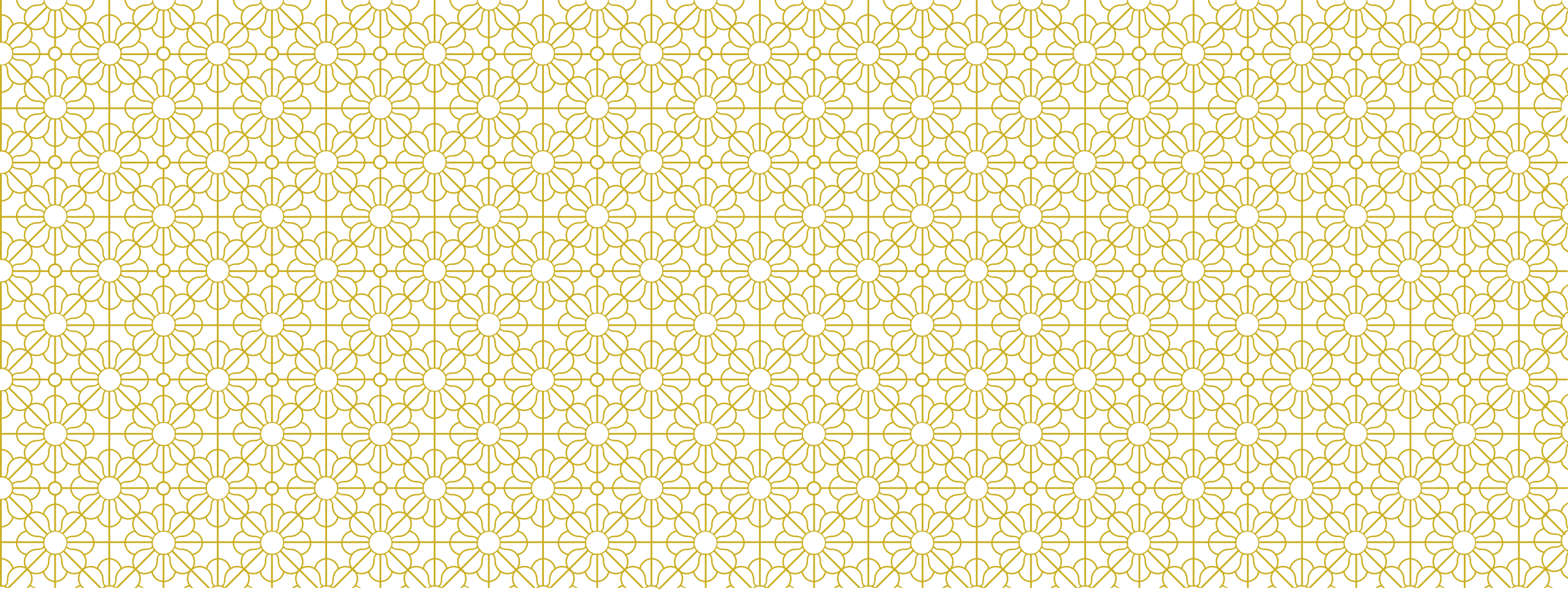


LOOP,LABEL,COUNTER REGISTER,INC

Loop, Label, Counter Register, Inc. (1)

Loop "Series of conditions that is repeated until a terminating condition is reached."

```
Mov dx, 'a'
Mov ah, 2
int 21h
```

Basic instructions for prnig single character. To repeat it again and again?

. Give name of series of instuction, In beginning, called it label.

LabelName: → Give name Series of instruction In beginning, we called it label

Mov dx, 'a'
Mov ah, 2
int 21h

} Instruction.

Loop LabelName → After series of instruction where it is called give its name after write Loop.

- LabelName should be any name.
- To call it how many times, it will be decided by Counter Register.

Counter Register.
- General Purpose Register.
- Count program
- Main purpose is to be used for a loop.

- The value placed in counter register, (a constant value) it will run according to it.

- [redacted]

- Before writing the loop we send value to Counter Register

Mov cx, 10 (runs 10 times)

- Value reduced from $Cx = 10$ to $Cx = 0$ loop stop. (Work on decremant by 1 till 0).

$Cx = 10$
$Cx = 9$
⋮
$Cx = 0$.

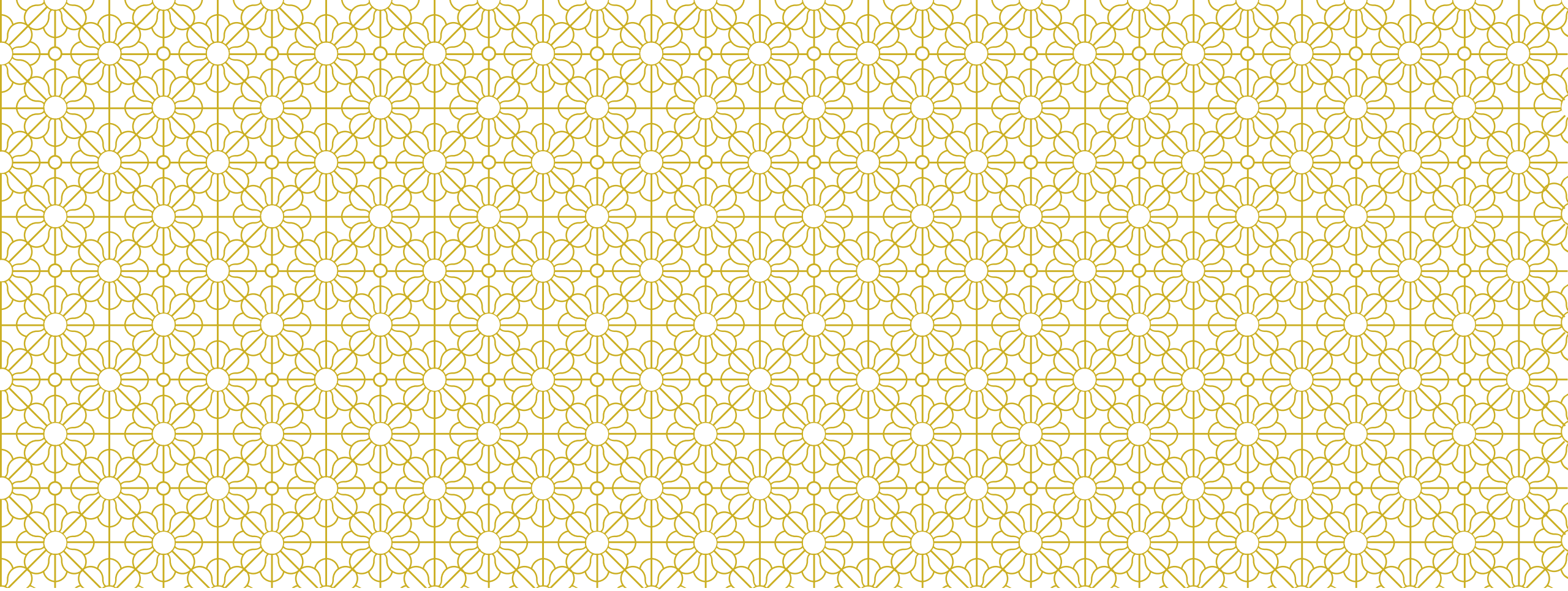
- **Label Syntax**
  - ✓ Test:
  - ✓ Test1:
  - ✓ T1:
  - ✗ 1Test:
  - ✗ Reserved word:

Label Rules

1) A label can be placed at the beginning of a statement, because the label is assigned the current value of line.

2) Label Name must not be a reserved word.
   e.g. Mov, Add, DB, DW, etc.

3) Colon : must be used with label while initializing, but not while calling.



LAB

Program to print 0 to 9.

```
.dosseg
.model small
.stack 100h.
.data
.code.
main proc
```

Mov cx, 10 ⟶ first Counter is given for 10 times (0 to 9).

(Then give label name),

```
L1:
    Mov dx,48
    Mov ah,2
    int 21h

Loop L1
```

In this case 0 print again and again. dx again & again get 48. So we placed it before loop Value go only one time.

So we change it.

```
Mov cx,10
mov dx,48
L1:
   mov ah,2
   int 21h
     Add dx,1
Loop L1
```

In this also zero print 1 time, In loop after print write [Add dx,1], 48 add 1, so on loop runs.

- We used,

```
Add dx, 1
```

OR

```
Inc dx
```

→ Increment by 1

```
.dosseg
.model small
.stack 100h
.data
.code
main proc
mov cx, 10
mov dx, 48
L1:
mov ah, 2
int 21h
Add dx, 1
Loop L1
mov ah, 4ch
int 21h
main endp
end main
```

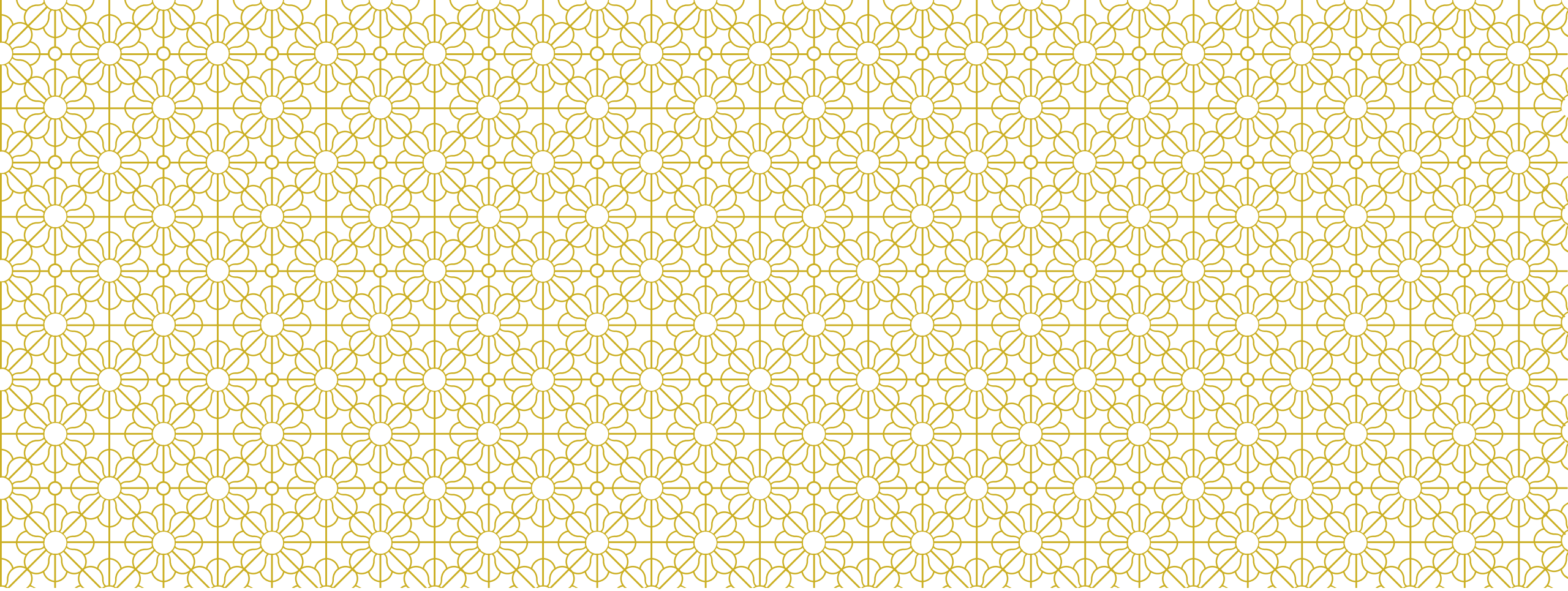
(2) Program to print Capital letters from A to Z using loop.

- tell counter register how many times the loop run, (A to Z) 26 characters,
  Mov cx, 26.
- Give loop name.

```
Main proc
Mov cx, 26      ⟶ loop run 26 times.
mov dx, 65      ⟶ 65 ASCII for A.
L1:
  mov ah, 2
  int 21h
```

```
inc dx      or   add dx, 1
Loop L1
mov ah, 4ch
int 21h
main endp
end main
```

(3) Program to print small letters from a to z using loop?



FLAG REGISTER

Flag Register

- It's a flag register, It has several bits, every bit has different function and names.
- Basically a flag register, which has different bit names

Why flag Register :-

1) We know about what controls the operations of CPU ? like int 21h ⟶ how CPU handle this
2). What handles the status of operations.

```
  10101010
+ 01010101
----------
```

Bit move to register & add performed, what about carry while addition, where carry is moved.

How flag Reg do this work in add, Sub, div.

- What bit handles the status of operation.

3) Conditional jump.

```
mov dl, 12
mov al, 10
```

If i want comparison b/w dl and al (If dl is greater then al) then it go up.

```
mov dx, 'a'
mov ah, 2
int 21h
```

- Use of conditional jump with the help of flag register.

4) Which number is lessee or greater ?

- Flag Register is a register that contain the current state of the processor.

In CPU flag register 16 bits (0_15).

Useful bits is 9.

Any bit of register like 64 or 128 useful is 9.

1) **Carry flag :-** (CF) : Addition of 2 bits, last final carry out is Handled by this Register.

last carry out → 1 1
1011
+1011
1010

Carry flag : CF
1 : When thee is last carry out
0 : When there is not last carry out.

2) **Parity flag :-** Important flag. (PF)

- Tells about integrity of data.
- Result of Add, Sub any operation, when this result is stored in register, go through some medium (wire/bus). When it reaches, the data is correct?, who tells about validity & verified it.

An additional extra bit is add in result to tell about its validity.

In 8086 arch Parity Flag : PF

1 : When there is even no. of ones.
0 : When there is not even no. of ones.

0 (00000010) → odd. (no. of bits).

3). Auxillary Flag (AF) In addition, the carry (not last carry) every 3rd bit carry.

Every 3rd bit carry

```
 1 0 1 0 1 0 1 0
+1 1 0 1 0 1 0 0
```

Every 3rd bit carry.

Every 3rd bit carry controlled by Auxillary flag.

1: When 3rd bit carry exists
0: When 3rd bit carry doesn't exist.

4) Zero flag: (ZF)

Perform subtraction of two numbers like.

```
  00000001  <-- 1
- 00000001  <-- 1
----------
  00000000  <-- 0
```

Result of operation is zero. handled by ZF

1: When result is zero.
0: When result is not zero.

5) Sign Flag (SF)

Subtract of two numbers like result is in minus,

```
  00000011   3
- 00000111   7
----------
  00000110  -4
```

Result is in minus, but not visible in binary, this minus is handled by Sign flag.

SF : 1 ⟶ When result is negative
0 ⟶ When result is positive. (handles sign in result).

⑥ Trap flag. Error trap. / Bugs trap.

When prog run it show errors, how it is show, How CPU show you the error, Shown by TF.

. System used this for Debugging.

1 : When single step mode (debugging) is required/needed
0 : When single step mode (debugging) is not needed.

Used by system by default at Backend.

⑦ Interupt Flag (IF) Interupt handled by IF
tells CPU that interupt is called.

IF : 1 ⟶ when interupt is called
0 ⟶ when interupt is not called.

⑧ Direction flag (DF): 'hello$' Any string, string print from h to 0, normally, If you want to print in reverse order, direction handled by DF.

DF : 1 : String automatically decrements the address.
0 : string does not automatically decrement the address.

'hello$'
⟵

⑨ Overflow flag(OF) If size is register is less then result, handled by OF.

Add dx, ax.

ax |||||||||||||||| max range 65536
dx |||||||||||||||| size 65536

Addition of ax and dx, become out of range.

1: When result is too big to fit in the Destination
0: When ~~result~~ there is not too big to fit in the destination.

Status flag: To handle the result of an operation.
CF, PF, AF, ZF, SF

Control flag: To control operation of CPU
TF, DF, IF