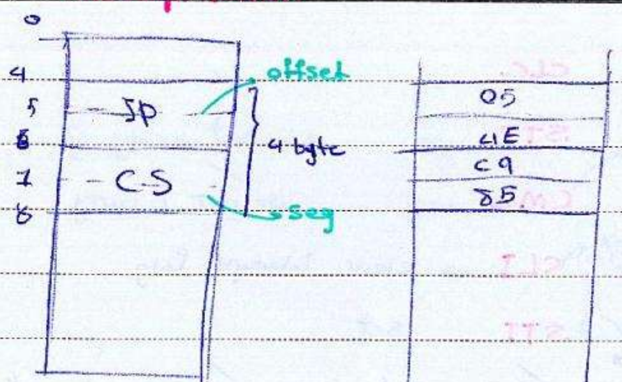
IP به دستوری که می‌خواهد اجرا شود.

Subject
Date

Interrupt vector

Interrupt ۲



CS:IP

برای اجرا شدن باید IP و CS به جایی که برنامه می‌خواهد اجرا شود point شود

IP 4E05

CS 85C9

۱۴ را اول

مرجعی است که Interrupt routin را بنویسیم یا نوشته شده است می‌رود و آن را اجرا می‌دهد (قبل از تغییر IP, CS آنها را در stack ، push می‌کند تا بتوانیم بعد از تمام شدن Interrupt ، به محل قطع شدن برنامه برگردد)

مراحل اجرای دستور INT
1. push کردن flag ها در stack
2. Flag مربوط به TRAPENABLE غیرفعال می‌کند
3. push کردن CS در stack
4. push کردن IP در stack
5. IP و CS را از آدرس برداری دارد

در Call کردن فقط دستورات 3 و 4 اجرای شود.

مراحل اجرای دستور IRET →
1. pop کردن IP از stack
2. pop کردن CS از stack
3. flag seg

INTO → → در میکروپروسسور overflow Interrupt هست آن conditional Interrupt است (معادل INT4) اجام شود
Interrupt if overflow
که مربوط به overflow است عملیات مربوط به overflow را اجرا می‌دهد

org → مشخص می‌کنیم که دستورهای بعدی که نوشته می‌شود (بعد از org) در چه آدرسی از حافظه قرار بگیرد)

2A → تاریخ
2B
2C
2D

PAPCO

RET سه فقط آدرس بازگشت را برمی‌دارد و flag ها داخل stack باقی ماند

IRET سه آدرس بازگشت و flag ها را از stack برمی‌دارد اگر بخواهیم یک procedure را

Interrupt routine باید از IRET استفاده کنیم

ROUTINE PROC

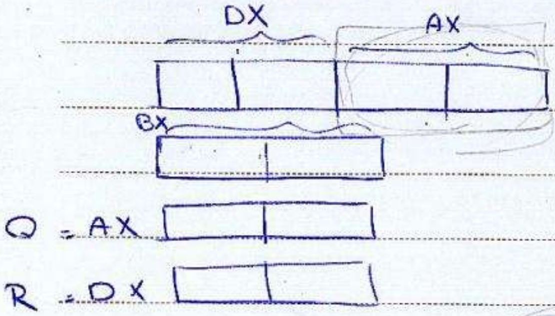
IRET

ROUTINE ENDS

اگر بخواهیم 240 routin را تغییر دهیم seg و offset آن را در آدرس 240*4 load می‌کنیم

NOP → no operation — یک byte size دارند که هیچ کاری انجام نمی‌دهند

تمرین ← تقسیم 32 بیتی به 16 بیتی ؟



$$\frac{R_1 \cdot 2^{16} + AX}{BX}$$

$$\frac{DX \cdot 2^{16} + AX}{BX} = \frac{DX \cdot 2^{16}}{BX} + \frac{AX}{BX}$$

Q_1, R_1 Q_2, R_2

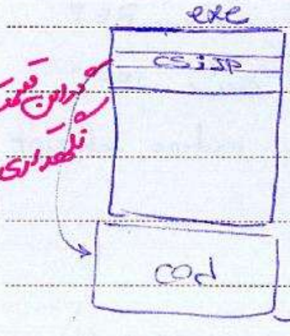
$\frac{21}{2}$ $\frac{2 \times 10 + 1}{2}$

Q , Q

10 + 1

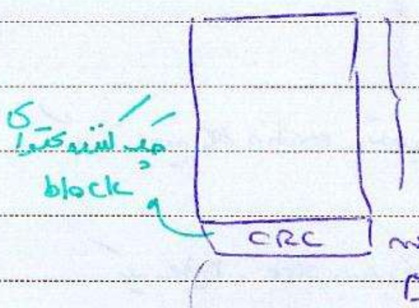
11

Subject
Date
Interrupt exe سیر برای کار با PC Intern



باید ابتدا که خود را اضافه کنیم به CS:IP
را به گونه‌ای که کد خود را نوشتیم تغییر دهیم. بعد از
آن باید به آخر code به CS:IP قبلی Jump
کنیم

اگر بخواهیم که در بالای header اضافه کنیم باید کل آدرس‌ها را
shift می‌دادیم که در این صورت باید حواسمان به Jump ها هم باشد
این در واقع یک نوع ویروس نویسی است.



برای اینکه برای هر کدام از این byte ها یک parity
اضافه کنیم در آخر یک خانه برای کل parity ها اضافه می‌کنیم

از XOR داریم
هم سرعت بیشتر است و هم آدرس‌دهی راحت‌تر
هم فضایی که اشغال می‌کند کمتر است

اسم MACRO (لیستی از دستورالعمل‌ها) Directive

ENDM

اگر مجموعه دستوراتی داریم که چند بار در برنامه اجرا می‌شود می‌توانیم داخل macro بنویسیم و هر دفعه
آن را call کنیم عیناً همه آنچه می‌شود دستورالعمل‌هایی که به جای آن call قرار
می‌دهد

تعریف داخل MACRO، valid نیست

```
XCHNGWRD  MACRO  VALUE1, VALUE2
          LOCAL  LABEL1
          MOV    AX, VALUE1
          MOV    BX, VALUE2
LABEL1:
          XCHG   AX, BX
          MOV    VALUE1, AX
          MOV    VALUE2, BX
          JZ     LABEL1
ENDM
```

به این معنا می‌باشد که هر وقت macro را call کنیم label آن یکی باشد

MACRO می‌توانند آرگومان داشته باشند مثل proc
می‌توانند

قبل از assemble کردن خط call کردن را
برمی‌دارد و به جای آن عیناً کپی می‌کند در واقع
extended code است نه call

LABEL ها را باید از نوع local تعریف کنیم به کمک Directive به نام
local

اگر دو بار از MACRO استفاده کنیم آنوقت دو تا LABEL1
داریم و در Jump کردن مشکل پیش می‌آید

Subject
Date

در MACRO سرعت بالاتر است چون نیازی به کار با stack ندارد. در proc در Runtime اجرا می‌شود ولی MACRO در هنگام assemble شدن کپی می‌شود.

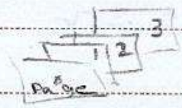
اگر قبل از وارد شدن MACRO با stack، AX و BX کار کرده باشیم بعد از اینکه وارد MACRO شد خراب می‌شود و مقدار جدیدی می‌شود. عاقلانه این است که قبل از MACRO در stack، push کنیم و بعد از آن pop کنیم.

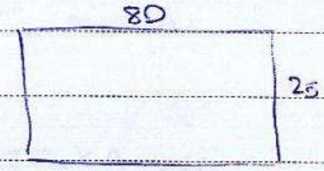
```
MOVECURSOR   MACRO   X,Y
      MOV    AH,02H
      MOV    BH,0
      MOV    DH,X
      MOV    DL,Y

      INT 10H
ENDM
```

(کنار MOV AH,02H: ۴ تا page از صفحه‌ها را می‌تواند نگه دارد)

(کنار MOV BH,0: مشخص می‌کند که مقصود ما کدام page است)

(کنار MOV DH,X و MOV DL,Y: سطرها و ستون‌ها را باید در این قرار دهیم)





$0 \leq x \leq 24$
$0 \leq y \leq 79$

```
WRTMSG  MACRO  XYZ
      MOV AH,09H
      LEA DX,XYZ

      INT 21H
ENDM
```

(کنار INT 21H: تا به $ برسد چاپ می‌کند)

```
X DB 'THIS IS'
MOVCURSOR  10,10
WRTMSG X
```

Subject

Date

برنامه جذر $y=\sqrt{x}$

```
STK SEGMENT PARA Stack 'stack'
    DB 1024 DUP(?)
STK ENDS

CONSEG SEGMENT
    ASSUM ____ , SS: STK
    ;
    MOV AX, STK
    MOV SS, AX
    MOV SP, 10234
```

Subject
Date

تمرین) ۲ تا برنامه بنویسید که عدد فیبوناچی را بدست آورید

الف) بوسیله proc ب) بوسیله macro ج) فرق PROC با macro در برنامه فوق چیست؟

یک عددی می‌دهیم باید عدد فیبوناچی متناظر با آن را بدهد

Directive هستند در Runtime عدد مشخص نمی‌شود

```
FACT MACRO N
    IFE N            ; اگر n=0 بود
        F=1          ; mov F,1
    ELSE
        FACT N-1     ; result به صورت default در F بوده
        MOV AX,F
        MUL N
        MOV F,AX
    ENDF
ENDM
```

به تعداد call های proc، push داریم و به تعداد ret ها pop داریم

کدهای macro عیناً copy، paste می‌شوند

تابع macro را باید تعریفی که در هنگام اجرا مشخص می‌شود

مقدارش چیست را نمی‌توان صدا زد

چون stack نداریم یک قسمت از حافظه [F] را مشخص کنیم

در برنامه جزء قبل استفاده کنید macro تعریف کنیم که فقط expression، $A(\frac{N}{20} + 20/2)$ را حساب کنید

EXITM دستوری است که از macro بیرون می‌برد

در مورد MACRO ها آرگومان هایی که می‌توانند optional باشند

```
IFB   ADDER MACRO res, exp0, exp1, exp2, ...
        MOV AX, exp0
        IFNB exp1
            ADD AX, exp1
        ENDIF
        IFNB exp2
            ADD AX, exp2
        ENDF
        MOV Res, AX
```

ADDER BX,1
ADDER BX,1,5
ADDER BX,1,4,3

هر سه valid است (macro با آرگومان های optional)

PAPCO

Subject
Date

تمرین ۲: تابع برنامه‌ای بنویسید که عدد فیبوناچی را بدست آورید

الف) بوسیله proc ب) بوسیله macro ج) فرق PROC با macro در برنامه فوق

چیست؟ که عددی را دهیم باید عدد فیبوناچی متناظر با آن را بدهد

Directive هستند در Runtime مشخص نمی‌شود

```
FACT MACRO N
IFE N
    F = 1 = mov F,1
ELSE
    FACT N-1
    MOV AX, F
    MUL N
    MOV F, AX
ENDF
ENDM
```

(اگر N=0 بود)

(result به صورت default در F وارد)

به تعداد call های proc، push داریم و به تعداد ret ها pop داریم

کدهایی که برایش عینا copy، paste می‌شوند

تابع macro را بیشتر تعریف کنید که در هنگام اجرا مشخص می‌شود

بیشترش چیست را می‌توان ساده

جای stack ندارم که یک مقدار از جای F

را استفاده بکنیم

در برنامه جمع که کدام است باید macro تعریف کنیم که فقط expression، $A \frac{N}{Y_0} + Y_0 / 2$ را محاسبه کند

EXIT M دستوری است که از macro بیرون می‌آییم

در مورد macro ها آرگومان های که می‌توانند optional باشند

```
IFB ADDER MACRO res, exp0, exp1, exp2, ...
    MOV AX, exp0
    IFNB exp1
        ADD AX, exp1
    ENDIF
    IFNB exp2
        ADD AX, exp2
    ENDF
    MOV Res, AX
```

ADDER BX, 1
ADDER BX, 1, 5
ADDER BX, 1, 4, 3

هر سه valid است macro با آرگومان های optional

P4PCO

Subject
Date

```
SWAP   MACRO   AR1, AR2
       IF NDF TMP  Tmp L
       Tmp DB ?
L: ENDF MOV   TMP, AR1
       MOV   AR1, AR2
       MOV   AR2, TMP
       ENDM
```

چون می‌خواهم تعریف
tmp اجرا شود در (runtime
هم tmp و هم L باید به صورت local تعریف شوند تا به هر دفعه استفاده از macro فرق کند

اگر متغیر temp بخواهیم استفاده کنیم
باید در شرط اجرای برنامه و شرط Define کنیم
چون اگر اجرا شود مقدار آن unknown
می‌شود برای ما مشکل‌ساز است
برای همین Tmp را این شرط تعریف کردیم
اینست که بخواهیم tmp را مستقل کنیم یعنی خارج از macro بتوان از tmp استفاده کرد یعنی tmp اگر بخواهیم در خارج از macro در دسترس نیست

در Assembly متغیر scope معنی ندارد.

تمام macro‌ها را می‌توانیم در یک فایل با پسوند .LIB save کنیم در هنگام نیاز INCLUDE کنیم
وقتی true است که در pass 1 اجرا شود

```
IF1
    INCLUDE my.LIB
ENDF
```

تکرار کردن کد

```
        REPT   exp
MACRO   IRP    exp, < آرگومان‌ها >
        IRPC   exp, string     < Ali >
```

از دستور بعدی تا دستور ENDM را تکرار می‌کند

```
MOV   X, 1
REPT  10        ← به تعداد ده بار تکرار می‌شود
ADD   X, 1
ENDM
```

P4PCO

Subject

Date

```
MOV   X,1
IRP   X,<1,2,3,5>
  ADD X,1
ENDM
```

به تعداد ۱ و به تعداد 2 و به تعداد 3 و به تعداد 5 تکرار می‌کند

۴ بار تکرار شد

```
ADDER   MACRO   res, NO1, NO2, NO3
   MOV   X,0
   IRP X,<NO1, NO2, NO3>
   ADD  X,1
   ENDM
```

در IRPC به تعداد دندانه‌های موجود در string تکرار می‌شد

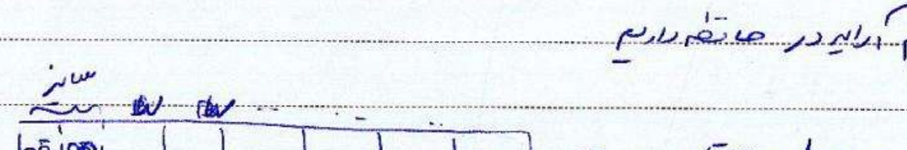
string , mn Xy ;

همان IRP هست فقط به جای اینکه مستقیم عدد بدهیم حرف می‌دهیم

که عددش متناظر با همان حرف است.

Data Structure *

فرض می‌کنیم آرایه در حافظه داریم



در extra seg است

DI ← آدرس شروع آرایه

می‌خواهیم مقدار AX را در صورتی که وجود ندارد به لیست اضافه کند

P4PCO

Subject:
Year. Month. Date. ( )

← مقداری روی مانیتور بنویسید مکرر را جابجا کند ...

INT 21H و INT 10H

کار با مانیتور و نمایش متن در مانیتور

نمایش pixel در مانیتور

کار با Text گرافیک کار می‌کند

INT 10H

AX

00 H → تعیین کردن مد صفحه نمایش

01 H ↗ تعیین اندازه مربع مکان نما)

02 H تغییر محل مکان نما

03 H پیدا کردن موقعیت با اندازه مکان نما

05 H انتخاب Active page

(برای نمایش به تعداد کاراکترهای صفحه نمایش وجود دارد که page است)

06 H ↗ پاک کردن صفحه و scroll ap

07H " " dawn

08 H خواندن یک حرف به همراه صفات آن در محل مکان نما (زمینه و پس زمینه)

09 H نوشتن تعدادی حرف همراه رنگ در محل مکان نما (بدون تغییر محل مکان نما)

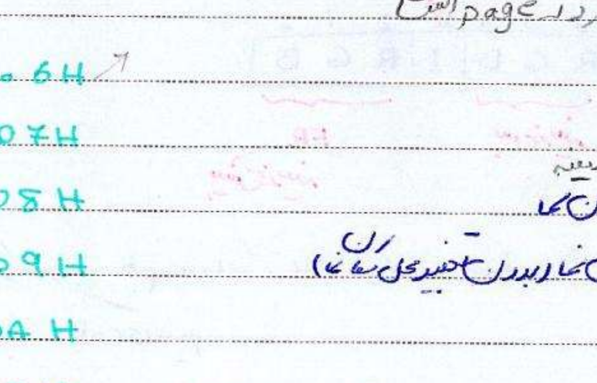
0A H " " بدون رنگ

0C H روشن کردن pixel

0D H خواندن مشخصات یک pixel

0E H نوشتن یک حرف روی مانیتور و تغییر مکان نما

0F H پیدا کردن مد جاری در صفحه نمایش



25 x 80 = 2000

بایت برای نگه داشتن اطلاعات صفحه نمایش

(هرکدام از این بایت‌ها، ۱ بایت attribute دارند رنگ و پس زمینه ...)

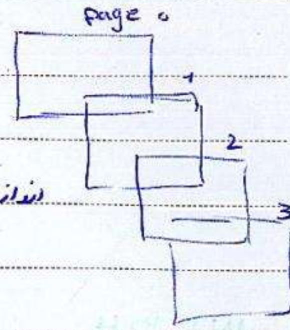
2000(B) (char)

2000(B) (attr)

4000 B

Subject:
Year. Month. Date. ( )

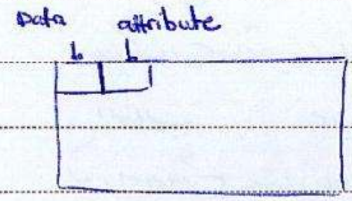
page 0: آدرس شروع page0 B800:0000



اندازه هر page ، 4K است

B900:0000 → B9000
BA00:0000 → BA000
BB00:0000 → BB000

} physicall address



اول Data بعدش attribute آن data
می‌آید

Intensity (شدت نور)

7	6	5	4	3	2	1	0
B	R	G	B	I	R	G	B

- bit 7: Blinking
- bits 6–4: پس زمینه
- bit 3: Intensity (شدت نور)
- bits 2–0: FR پیش زمینه

در هر بار interrupt ، AH نوع function را مشخص می‌کند به شماره function را در AH می‌گذاریم
بعد call می‌کنیم

سرویس 00H

MODE ها :

color	Resolution	تعداد حروف	تعداد pixel برای هر حرف	
mono	400 x 360	25x40	9x16	0
16	400 x 360	25x40		1
mono	400x 360	25x80		2
16	400 x 720	25x80		3
mono	400 x 720	25x80		7

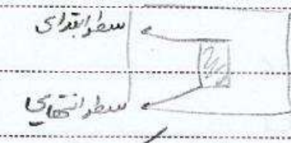
PAPCO

Subject:
Year. Month. Date. ( )

AH ← 00
AL ← (03 مثلاً) نوع متنی video mode (0-7)
INT 10H

نوع 01 H ـ زیادی الله

default — mono 11-12, color 6-7
AH ← 01H
range — mono 0-13, color 0-7
CH ← سطر ابتدای (بالای)
CL ← " انتهای (پایین)
IN 10H



محوری زیادی کنیم افقی نمی‌شود زیاد کرد
مثلاً می‌توانیم کمتر از اینجا تا اینجا باشد
محوری عمودی را می‌توانیم تغییر دهیم

function سر 02H

AH ← 02
DH ← سطر 0-24
DL ← ستون 0-79 or 0-39 (by default)
BH ← page (0, 1, 2, 3)
INT 10H

function سر 03H

AH ← 03
BH ← page
بعد از call ~ INT 10H
DH → سطر
DL → ستون
ending line CL → } اندازه شکل‌ها
starting line CH → }

ST
ACK
Busy
Out of P
Select in
Select
Auto feed
Error
ST

Subject:
Year. Month. Date. ( )

function

AH ← 05

AL ← Page

INT 10H

function 06H

یک تکه از صفحه را در نظر بگیریم می‌توانیم آن را
بالا ببریم و زیر آن خالی می‌شود در نقطه باید
رنگ آن بدهیم که با space که by default
رنگ مشکی است پر می‌شود

AH ← 06

CH ← c_1

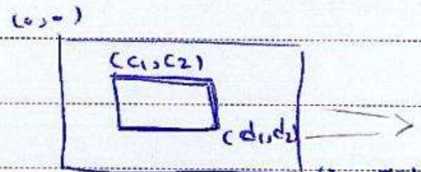
CL ← c_2

DH ← d_1

DL ← d_2

BH ← رنگ زمینه و حروف

AL ← تعداد خطوطی که باید scroll up شود



طول کادری که می‌خواهیم ... را مشخص می‌کنیم

AL=0 → کل آن بالا می‌رود و صفحه clear می‌شود

AL=1 → خط صفر پاک می‌شود و همه خطوط یکی بالا → shift می‌کند

```
CLRSCR  MACRO
    mov  AH, 06H
    mov  CX, 0
    mov  DL, 79
    mov  DH, 24
    mov  AL, 0      → کل صفحه
    mov  BH, 0H     → سیاه همه صفحه می‌شود
ENDM
```

function ... همین است فقط رو به پایین scroll می‌کند

PAPCO

Subject:

Year. Month. Date. ( )

برای گرافیک video mode کاربرد دارد

function 08H

AH ← 08H

BH ← Page

INT 10H

AL ← حرف

AH ← attribute

function 09H

AH ← 09H

BH ← page number

AL ← character (کد اسکی)

BL ← attr.

CX ← Counter → تعداد دفعاتی که می‌خواهیم حرف چاپ شود مثلاً اگر بخواهیم 5 تا حرف a را در کل کرسر جابجا شود

BL ← attribute color ← text mode ← grafic

function 0AH

AH ← 0AH

AL ← char

BH ← page

CX ← counter → برای تغییر بیش از یک کاراکتر

function 0EH

AH ← 0EH

AL ← char

BL ← attr (grafic mode only)

BH ← page

PAPCO

Subject:

Year. Month. Date. ( )

function 0EH

AH ← 0EH

INT 10H

AL ← شماره حالت

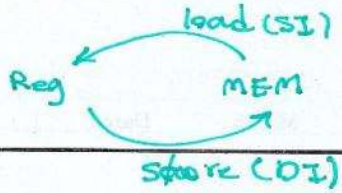
AH ← تعداد حروف در یک خط

BH ← شماره Page

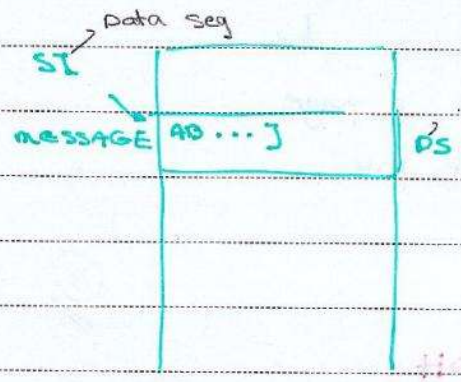
تمرین: برنامه ای بنویسید که نام و نام خانوادگی خودتان را به صورت افقی به سمت چپ scroll کند

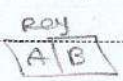
Subject:

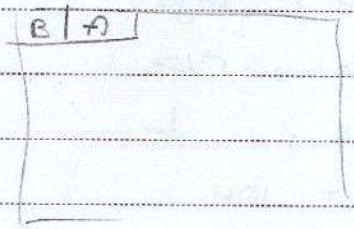
Year. Month. Date. ( )




```
MESSAGE DB 'ABCDEFGHIJ'
MOV AX,B800H
MOV ES,AX

CALL PRINTMSG

MOV AX,4C00H
INT 21H

PRINTMSG PROC Near
    MOV CX,10
    CLD
    MOV DI,1800
    LEA SI,MESSAGE
BACK: LODSB → byte string
    MOV AH,0EH ← يعني كذا
    STOSW
    LOOP BACK
    RET
PRINTMSG ENDP
```





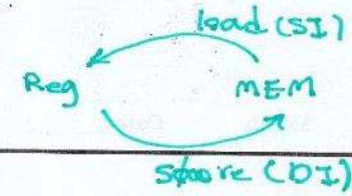


ستين

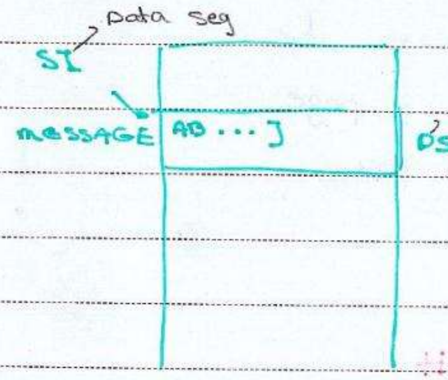
attribute = ? = $160i + 2j + 1$

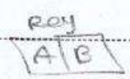
character = $160i + 2j$

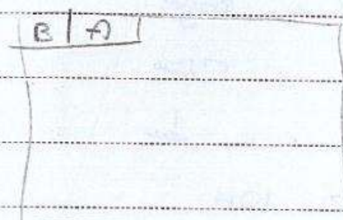
Subject:
Year. Month. Date. ( )




```
        MESSAGE DB 'ABCDEFGHIJ'
    MOV AX,B800H
    MOV ES,AX

    CALL PRINTMSG

    MOV AX,4C00H
    INT 21H

PRINTMSG PROC Near
    MOV CX,10
    CLD
    MOV DI,1800
    LEA SI,MESSAGE
BACK:  LODSB  →  byte string
       MOV AH,0EH  ←  تعیین کنه
       STOSW
    LOOP BACK
    RET
PRINTMSG ENDP
```







سطر ستون

attribute = ? = $160i + 2j + 1$

character = $160i + 2j$

P4PCO

Subject:

Year. Month. Date. ( )

0CH Pixel رنگ یک

AH → 0CH

AL → Attribute

CX → ستون

DX → سطر

BH → page

INT 10H

0DH pixel رنگ یک نقطه

AH ← 0DH

BH ← page

CX ← ستون

DX ← سطر

INT 10H

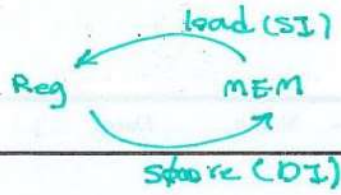
AL → color

مدهای گرافیک

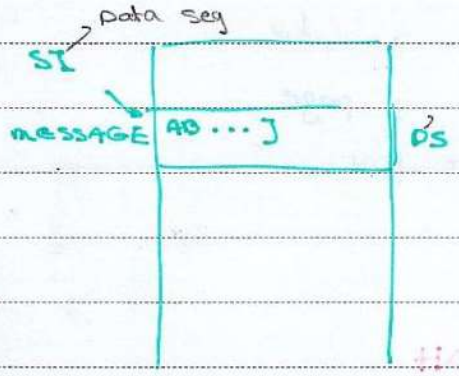
حالت	Resolution	color
04,05H	200x	4
06H	200x	mono
0DH	200x	16
0EH	200x640	16
0FH	350x640	mono
10H	480x640	16
11H	480x640	2
12H	480x320	16
13H	200x320	256

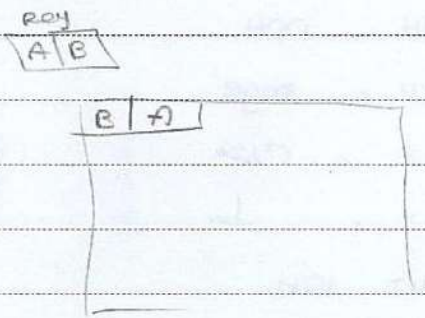
Subject:

Year. Month. Date. ( )




```
MESSAGE DB 'ABCDEFGHIJ'
MOV AX,B800H
MOV ES, AX

CALL PRINTMSG

MOV AX,4C00H
INT 21 H

PRINTMSG PROC Near
    MOV CX,10
    CLD
    MOV DI,1800
    LEA SI, MESSAGE
BACK: LODSB →  byte string
    MOV AH,0E H ←
    STOSW
    LOOP BACK
    RET
PRINTMSG ENDP
```





ستين

attribute = ? = 160 i + 2j + 1

character = 160 i + 2j