www.jozveha.ir

Subject:
Year. Month. Date.

زبان ماشین

هر دستور معادل یک بایت است.

ارتباط یک به یک

زبان ماشین: محتویات صفر و یک

زبان اسمبلی: text Base

MOV A,B

زبان پردازنده‌ها

زبان‌های سطح بالا:

Fortan

C , pascal

C++ , Java

Visual Basic , C , C++ , Java

.Net C#

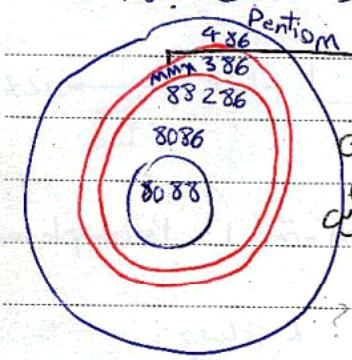
vb.net

تمام برنامه‌های با پسوند exe. شامل صفر و یک است و برنامه‌های OS هم از زبان‌های ماشین هستند

برنامه‌هایی که با سطح بالا نوشته می‌شوند در نهایت به زبان ماشین تبدیل می‌شوند

هر دستور در زبان سطح بالا معادل با چند دستور در زبان ماشین است. سرعت در زبان اسمبلی بالاتر است

یکی از کاربردهای آن CRACK است.



قسمتی سخت افزاری که عملیات گرافیکی را خود انجام می‌دهد (درون پردازنده) در نتیجه سرعت افزایش می‌یابد.

Direct x

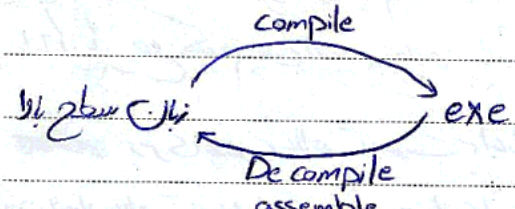
پروژه‌ها:

1) Crack : Find exe

2) exe تغییر فایل

3) FAT 12, 16, 32, NTFS } File System

4) Boot loader



پسوند فایل‌های مربوط به زبان ماشین asm. است و editor مورد استفاده notepad می‌باشد پس فایل را Assamble کرده به obj. تبدیل می‌کنیم که از به هم پیوستن چندین obj فایل exe بوجود می‌آید که این کار از طریق linker انجام می‌شود.

www.jozveha.ir

Data_sfe\Common\zarandi\Machine Language &--

Subject:
Year. Month. Date. ( )

notepad
mov A,B
code.asm

Assamble

code.obj

linker

code.exe

* assambler
* Linker
* Disassembler
* Simulator / Emulator
شبیه‌ساز — مقلد

همان simulator است ولی از سخت افزار هم استفاده می‌کند

Binary Numbering:

اعداد صحیح و اعشاری

- بدون علامت: 0 – 255
- با علامت
  - عدد علامت دار: 7 ... 0 — مثبت 0، منفی 1 — −127 → +127 — +0، −0
  - عدد (علامت به طور ضمنی)

2's complement (x) = $\bar{x}$ + 1 = 1's complement + 1

R's complement (x) = (R−1)'s complements + 1 ⟶ R درمبنای

(R−1)'s complement (x) = (R−2)'s complements + 1 ⟶ R−1 درمبنای

⋮

2's complement (x) = $\bar{x}$ + 1

R's complement (x) = $\bar{x}$ + (R−1)

نتیجه اشتباه است زیرا نمی‌توانیم مبناهای متفاوتی را با یکدیگر جمع کنیم پس نتیجه نهایی نیز اشتباه است.

برتری بدون علامت نسبت به اعداد علامت دار در دامنه اعدادی است که شامل می‌شود و ساده بودن آن و در اعداد علامت دار عدد با وجود اینکه به زبان انسان نزدیک است ولی صفر دو بار نگهداری می‌شود.

دامنه اعداد در روش 2's com از −128 تا 127 است.

01111111 10000000

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مکمل یک عدد یعنی وقتی با عدد مربوط جمع می‌شود تمام بیت های آن یا ارقام آن صفر شده و ۱ به بیرون بیاندازد

$$\begin{array}{r} 54 \\ -\,28 \\ \hline 26 \end{array} \rightsquigarrow \text{مکمل این عدد یعنی عددی که اگر با 28 جمع شود } 100 \text{ می‌شود } 100-28=72_2 \Longrightarrow \begin{array}{r} 54 \\ +\,72 \\ \hline \cancel{1}\,26 \end{array}$$

نمایش اعداد:

(binary coded decimal)

Decimal	binary	Hex	BCD	Askii	Gray
0	0000	0	0000	Null	0000
1	0001	1	0001	☺	0001
2	0010	2	0010	⋮	0011
3	0011	3	⋮		0010
⋮	⋮	⋮			0110
		9	1001	'a'	0110
15	1111	A	0001 0000	⋮	0111
16	10000	B	0001 0001	'z'	0101
17	⋮	C	⋮	'A'	0100
⋮		D		⋮	1100
		E		'Z'	1101
		F → 15		0	1111
		10		⋮	1110
n		11		9	1010
		⋮			1011
					1001
					1000

برای نمایش عدد n نیاز به $\log n$ بیت داریم تا بتوانیم آن را نمایش دهیم.

انسان: هر دو عدد متوالی حالت نمایشی برای ارتباط در یک بیت قرار دارند

کامپیوتر: رابط مهندسین برای اعداد با انسان

نالگوریتم

binary → BCD → Decimal

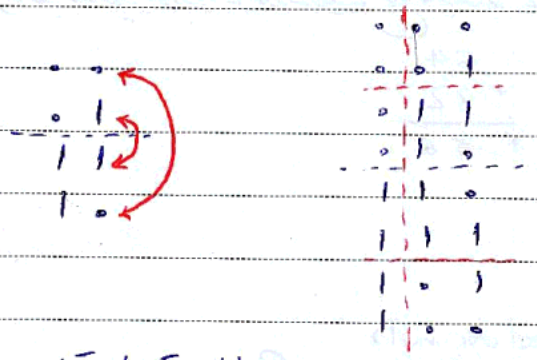
having Distance: کمترین تعداد بیتی که دو عدد متوالی با هم فرق می‌کنند را having Dis. گوییم که در Gray code این مقدار برابر یک است و در اعداد باینری بدون parity نیز یک می‌باشد ولی در حالتی که parity داشته باشد چون خود بیت parity یکی در میان عوض می‌شود having Distance آن برابر ۲ است.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

بدست آوردن Gray code :

۱- تقارن :

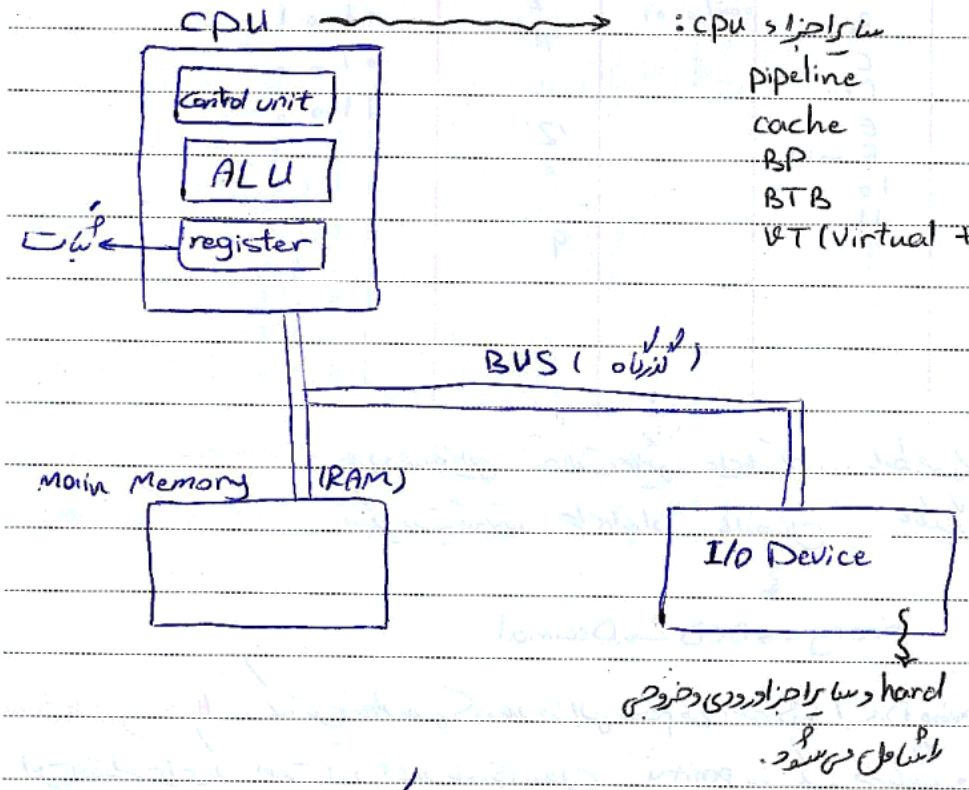


۲- از طریق باینری : برای بدست آوردن gray code از روی binary از سمت راست به سمت چپ عیناً کپی می‌کنیم به هر ۱ رسیده شد، عدد جلوی خود را Not می‌کنیم.

binary: 0101110101000

gray: 0111001111100

معماری کامپیوتر :



سایر اجزاء cpu :
pipeline
cache
BP
BTB
VT (Virtual translation)

BUS : هر گونه ارتباطی که هیچ الویتی نداشته باشد و n نقطه را به هم وصل می‌کند.

BUS
- Address n
- Data n
- Control 4_5

www.jozveha.ir

Subject:
Year. Month. Date. ( )

پردازنده n بیتی: تعداد بیت اختصاص داده شده به پردازنده از روی تعداد Bus Data (پهنای داده‌هایی که می‌تواند بگیرد یا بدهد) مشخص می‌شود.

4004 : پردازنده 4 بیتی
8085 : " 8 "
486، 386، 286، 8086 : " 16 "
Pentiom : " 32 "
Pentiom : " 64 "

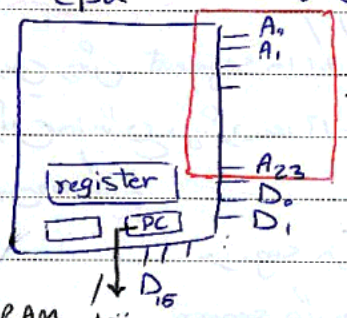
Pentium املای صحیح

باید توجه داشت که Bus ها نمی‌توانند داده‌ها را بر روی خود نگه دارند و فقط انتقال دهنده هستند و برای نگه داشتن داده‌ها تا زمان انتقال هر کدام از Device ها یک register دارند که اطلاعات را نگه می‌دارند. در نتیجه تعداد بیت‌های reg باید با پهنای باند Bus برابر باشد ولی نیازی به هم پهنا بودن register و Bus آدرس نیست.

CPU 80286:

در این cpu، Bus آدرس دارای ۲۴ پایه و Bus داده ۱۶ پایه می‌باشد یعنی یک پردازنده ۱۶ بیتی است که دارای ۱۴ رجیستر می‌باشد:

special purpos: 2 = فقط برای کار داخلی cpu به کار می‌رود و نمی‌توان به آن دسترسی داشت.
IR -2 PC -1
general purpos: 12 = قابل دسترسی و مشاهده کردن است.
} register 14



$0 \longrightarrow 24$

$2^{24} - 1$ ← بزرگترین عدد

2^{24} B = 16 MB ⟵ مقداری از حافظه که می‌توان آدرس‌دهی کرد.

نزدیک RAM است.

PC (program counter):

یا IP (instruction pointer) که مهم‌ترین ثبات است و دستور جاری را مشخص می‌کند و همیشه (در واقع آدرس خانه مربوط را مشخص می‌کند و می‌تواند به عنوان counter در loop ها نیز به کار رود.)

بعد از اجرای دستور:
PC = PC + 1
یا
PC ← n

www.jozveha.ir

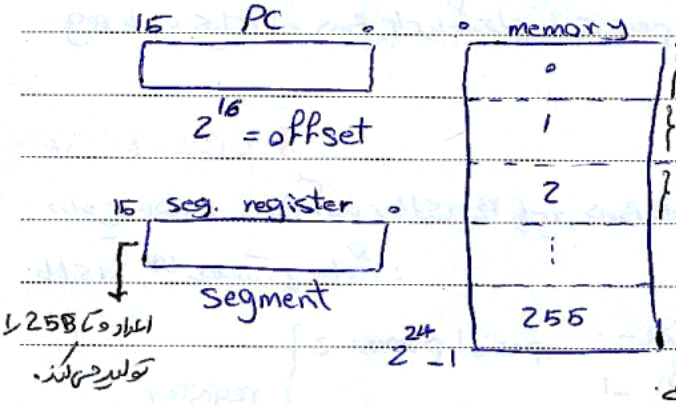
Subject:
Year. Month. Date. ( )

از معایب segmentation این است که قسمت های مختلف حافظه به یکدیگر تجاوز نمی کنند. در صورت اضافه بودن مقدار آن از یک seg. خطا می دهد ولی داده های دیگر را خراب نمی کند.

IR (instruction register): نیازی است که دستور جاری را نگه می دارد و اولین رجیستری است که در مقابل Bus قرار می گیرد.

1- segmentation:

حال می خواهیم از طریق PC که خود دارای 16 بیت می باشد 24 بیت را آدرس دهی کنیم. برای اینکه چنین عملی امکان پذیر باشد از روش segmentation استفاده می کنیم. در واقع بوسیله PC با 16 بیت می توانیم 2^{16} خانه را آدرس دهی کنیم سپس حافظه را به قسمت های $2^{16}B = 64k$ است تقسیم کرده و هر قسمت را بوسیله یک عدد که به آن segment می گوییم مشخص می کنیم که تولید segment از طریق یک رجیستر 16 بیتی انجام می شود.



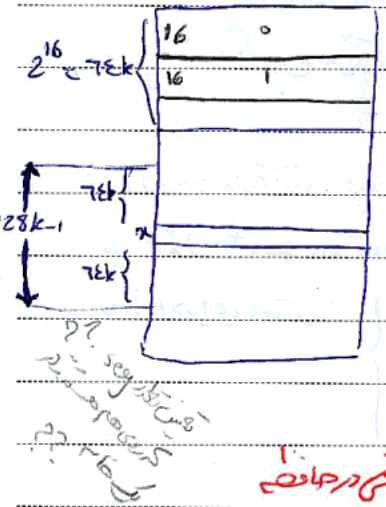
$\frac{2^{24}}{2^{16}} = 2^8 \rightsquigarrow$ segment 256 یعنی ایجاد می شود.

(seg , offset) seg : offset

0:10 ← segment صفرم و offset دهم از حافظه.

2- هم پوشانی:

در این روش segment ها با یکدیگر هم پوشانی دارند یعنی بعد از گذشتن از 16B که پایین می آید seg بعدی از آن جا شروع می شود. عدد 16 براساس تجربه بدست آمد. در این روش هر خانه را می توانیم به شیوه های مختلفی آدرس دهی کنیم.



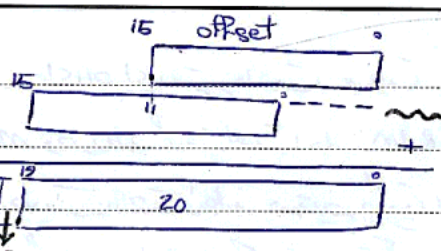
$\frac{2^{16}}{2^4} = 2^{12}$ ← هر seg. خود با این تکرار دیگر هم پوشانی می شود segment

$\frac{2^{24}}{2^4} = 2^{20}$ ← می توان 2^{20} seg. داشت اما فقط 2^{16} تا seg. reg. ولی از طریق یک از آن امکان دارد دسترسی را در نظر نمی گیریم.

آدرس واقعی در حافظه = S × 16 + O

S ↓ segment ، O ↓ offset

www.jozveha.ir



همانی که در ۱۶ ضرب می‌شود مثل این است که ۴ بیت به چپ shift داده می‌شود.

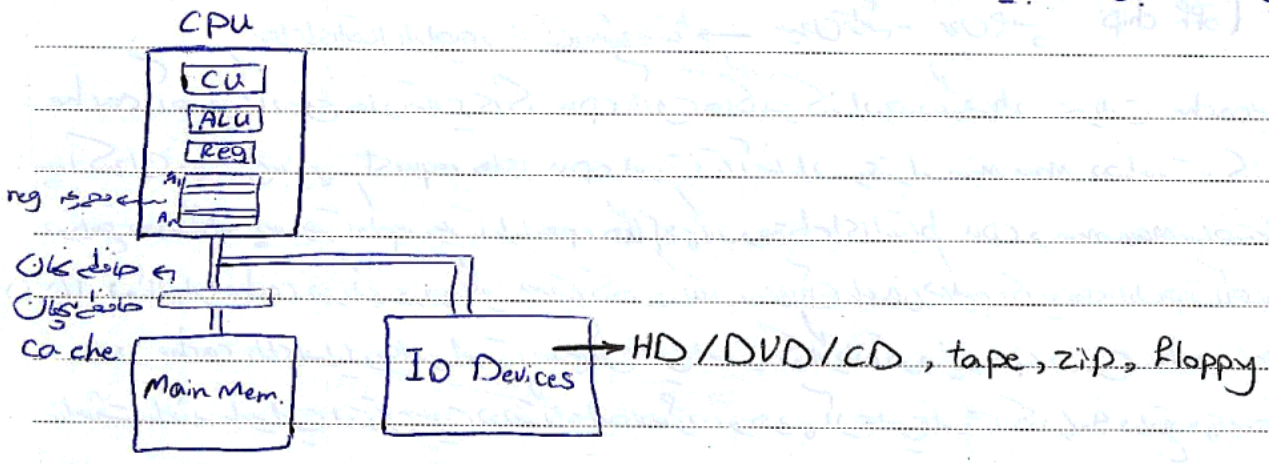
در نتیجه با این روش نهایتاً ۲۴ پایه یا ۲۴ خانه را آدرس‌دهی کردیم و دو بیت باقی‌مانده را به حساب نگذاریم.

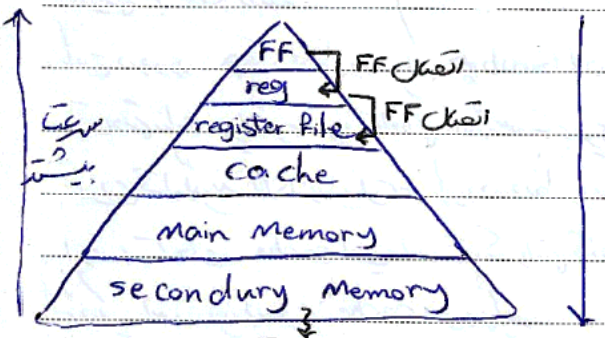
Real Address Mode

protected Mode ← از سه بیت هم استفاده می‌کند

دو Mode کار می‌کند

شرایط پیچیده‌تری است جدولی برای نگه داشتن اول و آخر seg داریم و همچنین برای نمایش آدرس از ۲۴ بیت استفاده می‌کنیم. حداقل حافظه‌ای که هر برنامه به خود اختصاص می‌دهد ۶۴k است و ممکن است یک برنامه چند seg و offset را به خود اختصاص دهد. در نتیجه segment ی که بعضی از آن استفاده شده است جای برنامه دیگری قابل استفاده نیست زیرا برای نگه داشتن آخرین offset اختصاص داده شده نیاز به چهار بایت داریم و این خود کار را دشوار می‌کند. در نتیجه در روش اول قسمت‌های زیادی از حافظه بیهوده هدر می‌رفت ولی در روش دوم نهایتاً 17B از بین می‌رود و این مقدار در مقابل 2^{16} بسیار اندک است.





در کامپیوتر ارتباط بین CPU و اطلاعاتی که در حافظه قرار دارند یک هرم ایجاد می‌کند که به صورت روبرو است.

DVD, HD, Floppy, CD, tape ← همان IO هاست

www.jozveha.ir

Subject:
Year. Month. Date. ( )

بیشتر زمانی که کامپیوتر اجرا می‌کند برای اجرای دستورهاست. سرعت cpu همیشه بیشتر سرعت Main Mem. زیرا تکنولوژی که در main mem وجود دارد ثابت است و از SRAM ها هستند که باید به برق وصل باشند تا داده‌ها از بین نروند و همه آنها از T جیبیست ساخته شده‌اند در نتیجه دیگر قابل توسعه نیست ولی در cpu چون کار پردازش را انجام می‌دهد می‌توان از روش هایی استفاده کرد که سرعت آن افزایش یابد زیرا نیازی به تکنولوژی ندارد. توجه داشته باشید که ارتباط بین cpu و main mem لازم و ملزوم است.

روش های پیشنهاد شده برای تسهیل ارتباط بین cpu و main mem :

1) استفاده از حافظه‌ی نهان (cache) :

سرعت قطعات خارج از تراشه کمتر از قطعات درون هستند لذا cache سرعت پایین‌تری نسبت به register File ولی از main mem بیشتر است.

on chip — برچسب — درون تراشه — هر از یک تا چند ثانیه هستند و چون ارتباط آنها سریعتر است

off chip — برون چسب — بیرون تراشه — زیرا برای اتصال از لحیم و استفاده شده و ارتباط

cache آخرین آدرس ها و مقادیری که cpu با آن ها کار می‌کند را در خود نگه می‌دارد. تحرکات cache به سمت پیدا کردن و پیش بینی request های cpu است تا آنها را سریعتر از Main Mem درخواست کند.

در واقع دو دستگاه پرسرعت و کم سرعت را با هم، هماهنگ می‌کند و فقط برای ارتباط cpu و main mem ساخته شده‌اند.

I/O D) مثالی از cache در زندگی روزمره: کیف دستی که همه روز کتاب های آن را عوض می‌کنیم، کاربرهای اداره جات آخرین پرونده‌ها را استفاده می‌کنند.

وجود cache دارای مزایا و معایبی است: بدی آن این است که اگر داده‌ی مورد نیاز cpu در درون cache وجود نداشته باشد باید این انتقال بین دو دستگاه ایجاد شود، و خوبی آن این است که در ۹۰٪ مواقع می‌تواند جواب معقولی به cpu بدهد.

نکته: در مورد حافظه‌ها اینکه هم داده‌ها را درون حافظه به صورت افقی ذخیره کنیم یا عمودی سرعت را کند می‌کند و تفاوتی ندارد زیرا اگر به صورت افقی باشد سرعت به نوع داده باید طول بیشتری را طی کند و اگر به صورت عمودی قرار گیرند برای رسیدن به رکورد مورد نظر باید خط هایی را طی کند. (تعداد دستگاه‌ها زیاد می‌شود).

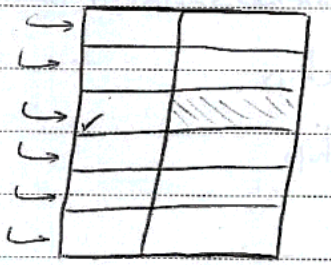
از قدرت های cache این است که زمانی که درخواستی داده می‌شود، چندین داده که از نظر زمانی یا مکانی به آن نزدیک است نیز درون cache قرار می‌گیرد.

cache با خاموش شدن کامپیوتر reset می‌شود.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

نکته: سرعت cache از این نظر سریعتر از main mem. است که پیدا کردن آدرس را به صورت موازی انجام می‌دهد.



آدرس مربوط به تمام خانه‌ها فرستاده می‌شود هر کدام که تطبیق داشت داده‌ی روی آن انتقال داده می‌شود.

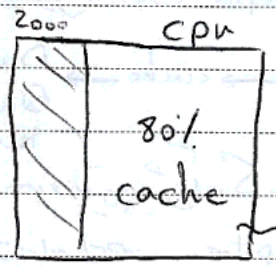
از این لحاظ به cache حافظه‌ی پنهان می‌گویند که نه cpu و نه main mem از وجود آن اطلاع ندارند و فکر می‌کنند هیچ کدام به آن دسترسی ندارند تمام عملیات درون آن سخت افزاری انجام می‌شوند (سرعت عملیات سخت افزاری بیشتر نرم افزار است ولی سخت افزار هزینه‌ی بیشتری را مصرف می‌کند.)

هسته‌ی اولین قسمت یا آخرین قسمت‌های main mem. مربوط به OS (سیستم عامل) است که در این مدل 80x86 که بررسی می‌کنیم در آخر حافظه قرار دارد در نتیجه با روشن شدن کامپیوتر حافظه شروع به خواندن از آخر می‌کند در ابتدا cache خالی است که با درخواست‌های cpu برای راه اندازی OS و غیره شروع به پر شدن می‌کند تا زمانی که cache کامل پر نشده می‌گویند cold است و متن محتوای داخل آن پر شود Hot می‌گویند که در این حالت به بهترین حالت رسیده و می‌تواند به درخواست‌ها بهترین جواب را بدهد و تا قبل از رسیدن به درجه‌ی hot به هیچ دردی نمی‌خورد.

بدترین حالت زمانی است که درخواست به صورت رندم باشند و cache مجبور باشد مرتب از main mem بخواند و بهترین حالت زمانی است که خطوط پشت سر هم درخواست شوند و ارتباط بیشتری با هم داشته باشند.

راه‌های افزایش سرعت cache:

۱- بزرگ کردن cache: با بزرگ کردن آن هزینه بالاتر می‌رفت تا جایی که در سال 2000، 80٪ هزینه خرید برای cache صرف می‌شد.



برای بالا بردن سرعت cache را در دو cpu قرار دادند.

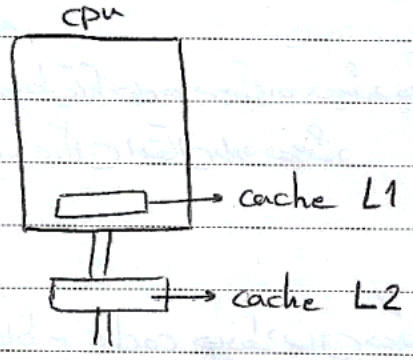
www.jozveha.ir

proj : تشخیص حافظه نهان با استفاده از برنامه.

Subject:
Year. Month. Date. ( )

cache { h = ? , size = ? }

2_ قرار دادن cache در بین cpu : حتی از دو cache استفاده کردند یکی درون cpu و یکی بیرون و ارتباط آن ها سه مرحله ای شد.



Level 1 : on chip

Level 2 : off chip

hit : موفقیت

Miss : شکست

اگر به درخواستی که از طرف cpu بتوانیم داخل cache به آن جواب دهیم

" " " " " نتوانیم " " " "

hit ratio = hit / (hit + miss) = تعداد موفقیت / کل درخواست ها = نسبت موفقیت

>95% ↓

اساس مقایسه cache ها

با وجود اینکه سیستم و مکانیزم های آن بر روی hit ratio تأثیر می گذارد ولی از طریق آزمایش می توان میزان آن را تعیین کرد.

replacement : وقتی مجبوری داده ای را انتخاب کنیم که با داده جدید جایگزین شود (مثلاً جایگزین با دورترین داده داده)

placement : شیوه قرار گرفتن آدرس ها درون آن را گویند (نگاشت) ← به ترتیب قرار دهیم

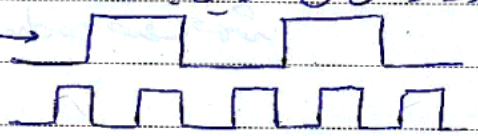
First in First out

1
4 ✓
100
400
6
20

با انتخاب placement و replacement و ... می توان بر روی میزان hit ratio تأثیر بگذارد.

برای بالا بردن سرعت cpu طول کلاک پردازنده را کم کردند و فرکانس آن افزایش یافت.

1980 →



→ pipeline → super scalare → cache → Dual Quad Hex

ولی محدودیتی پیش آمد که نتوانستند دیگر clock را کم کردند زیرا به noise تبدیل می شوند و باید وقت زمانی ثابت بماند (setup time)

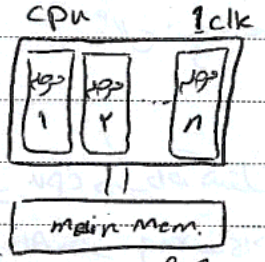
زیاد شدن تعداد cpu ها مشکلی ایجاد می کند که داده را کدام بگیرد، هر کسی خرابی را تشخیص دهد در نتیجه مشکل کردن احتیاج دارد و مشکل را زیاد می کند.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

super scalare: یک دستور را به چندین instruction تبدیل کرده و به طور همزمان چندین دستور را می‌گیرد و اجرا می‌کند. (چند خط pipe line با هم‌زمان اجرا می‌کند.)

pipe line: هنگامی که دستوری گرفته می‌شود و درون پردازنده decode می‌شود و باید چندین مرحله را برروی آن‌ها انجام دهد: ۱- گرفتن دستور ۲- decode کردن دستور ۳- گرفتن operator ۴- اجرای دستور ۵- بازنویسی حافظه. این مراحل را از هم جدا کردند و در هر لحظه همه مراحل در حال انجام هستند و دستورهای زیادی در یک لحظه و در گام‌های متفاوت بررسی می‌شوند.



m دستور ← n×m clock (کاهش سرعت نسبت به حالت معمولی)

pipeline:
1 → n clock
2 → 1 +
3 → 1 +
⋮
m → 1 +
n+(m-1)

← این هم n مرحله طول می‌کشد ولی n-1 با بقیه هم‌پوشانی شده

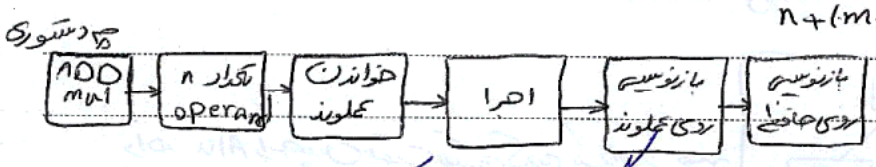
افزایش سرعت کند شد:

$$\text{speed up} = \frac{n \times m}{n+m-1}$$

$m \to \infty \quad = n$

⇓

هر چه تعداد مراحل بیشتر باشد با بی‌نهایت دستور (ایده‌آل) سرعت بازا تر می‌رود.



معایب pipeline:

لزومی به ترتیب instruction ها وجود ندارد مانند برنامه‌های سیالتی یا دستورهایی که goto و jump دارند دیگر مراحل برای هم دستورها به ترتیب اجرا نمی‌شود. البته با روش‌هایی می‌توان مشکلات goto را تا حدی برطرف کرد ولی در بعضی موارد انجام یک دستور به مقدار دستور قبلی مشکلی دارد و باید صبر کنیم تا آن کامل انجام شود.

goto بدون شرط مشکلی ندارد و با شرط در pipeline مشکل ایجاد می‌کند.

```
n: ____
   ____
   ____
   ____
   goto n
```

$a = 2^n - n \times n;$

goto n if a=0 → BP

برای حل این مشکل راهی به اسم BP (Branch prediction) اضافه کردند که با گرفتن احتمال قبل از رسیدن به دستور goto پیش‌بینی می‌کند احتمال اجرای این دستور چقدر است.

label2:

مثال: $R_3 \leftarrow R_1 + R_2$: ADD R_3, R_1, R_2

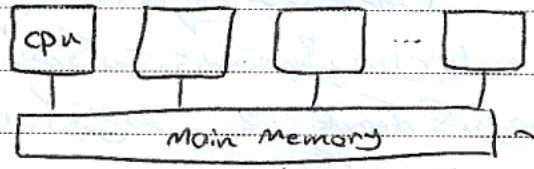
JNZ label2 → وقتی می‌خواهد اجرا کند به شرط است که دستور قبلی صفر نشده باشد (jump not zero) و چون مقدار R_3 مشخص نیست نمی‌تواند دستور را اجرا کند

JMP label2 → مستقیم و بدون شرط می‌پرد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

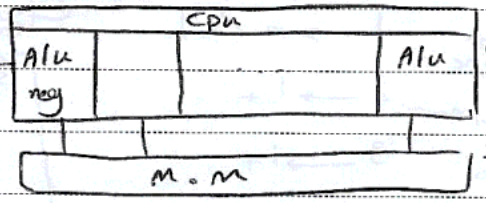
MIMD : multi instruction multi data — چندین دستور و data می‌خوانند عملیات را انجام می‌دهد.



هرکدام از این‌ها دستورها و داده‌های خود را می‌گیرند و عملیات را انجام می‌دهند مانند کارکردن چندین کامپیوتر ولی با یک memory

مثال: image processing ، game processing ← عملیات‌های موازی انجام می‌شوند

SIMD : single instruction multi data

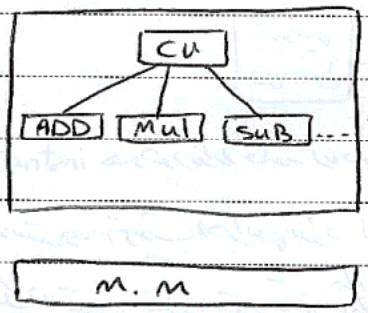


فقط یک cpu و یک واحد کنترل وجود دارد ولی هرکدام Alu ها reg های خود را دارند چندین داده می‌گیرند و یک عملیات را بر روی آن‌ها انجام می‌دهند

Alu ← دستورها با Alu اجرا می‌شوند

مثال: ضرب ماتریس‌ها

MISD : multi instruction single data



واحد Alu را به چندین قسمت تقسیم کردیم یک داده را می‌گیریم و بر اساس قسمت‌های مختلف عملیات گوناگونی را انجام می‌دهیم.

مثال: داده را می‌گیریم و چندین کار بر روی آن انجام می‌دهیم ← * کارهای آماری (یک سری داده داریم و برای محاسبه میانه، مد و ...)

* نقشه‌کشی و محاسبه محیط و مساحت با هم

* محاسبه سری نمایی

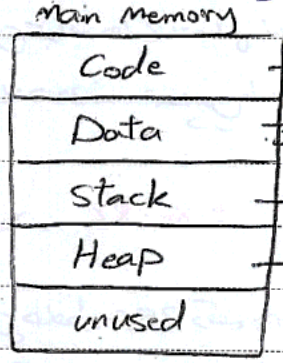
$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\dots$$

www.jozveha.ir

Subject:
Year. Month. Date. ( )

اسمبلی:

در نوشتن برنامه بوسیله اسمبلی باید دقیقاً مشخص کرد کدام قسمت از حافظه کد قرار دارد و کدام قسمت داده‌ها قرار دارند (یا چه داده مورد نیاز آن. که در مدل 80286 به بعد از طریق segment محل چیزهای مورد نیاز را مشخص می‌کند.



می‌توان جدا کرد

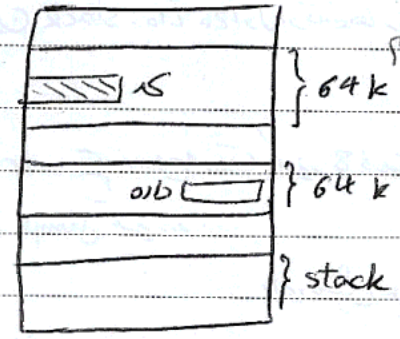
ولی برای کی باید تعداد است

(ES) باید جدا باشد چون به صورت فراخوانی procedure

دنیا یک است و از آخر پر می‌شود

برای pointer

Heap هم چون تعداد pointer ها متفاوت است باید جدا باشد.

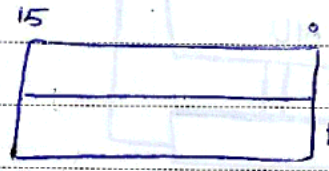


این است که به طور دستی محل ذخیره شدن code یا data را مشخص کنیم بلکه به طور مقدار اولیه خود اسمبلر دنبال می‌گردد و حالا می‌توانیم آن را عوض کنیم در نتیجه مشکلی روی هم افتادن داده‌ها به وجود نمی‌آید.

CPU 80286:

رجیسترهای موجود در این نوع cpu به صورت زیر:

14 register
- 2
  - Instruction pointer = PC
  - Flag register = (Flags)
- 12 →
  - داده
  - کد
  - سلحت



می‌توانیم هم با 8 تای بالا کار کنیم هم با 8 تای پایین

12: 15 ... 0

Ax	AH	AL
Bx	BH	BL
Cx	CH	CL
Dx	DH	DL

اندیس

قسمت high و low دارند ولی دستور نداریم باید 16 تایی بگیریم

SP : stack pointer	
BP : Base pointer	
SI : source index	
DI : destination index	

segment register: قرار دارد instruction

15

CS		code seg.
DS		Data seg.
SS		stack seg.
ES		extra seg.

PAPCO

بسیار پرکاربرد برای متغیرها

نسبت به اینکه seg چه چیزی را می‌خواهیم مشخص کنیم از یکی از این register ها استفاده می‌کنیم.

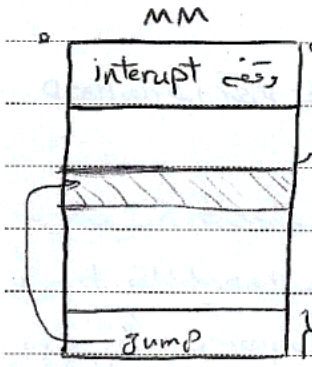
www.jozveha.ir

Subject:
Year. Month. Date. ( )

می‌شود چندین segment مربوط به قسمت‌های مختلف برنامه بر روی همدیگر قرار گیرند مثلاً در فایل‌هایی با پسوند com قسمت‌های code و data بر روی هم قرار دارند. در این نوع register file داریم همان reg هستند.

در نوشتن برنامه‌های اسمبلی باید تمام متغیرها را از بین ۴ رجیستر AX و BX و ... استفاده کنیم پس تعداد متغیرها محدودند. همچنین باید مقادیر segment ها را خودمان مشخص کنیم (code، data، ...)

حافظه در 80286:



در این نوع حافظه 256 وقفه وجود دارد.

شکل‌هایی که باید به پردازنده داده شود که خطایی اتفاق افتاده را وقفه می‌گویند مثل تقسیم بر صفر یا پر شدن stack. ۱۰ تا ۲۰ اول وقفه‌ها رزرو است ولی می‌توانیم با ایجاد وقفه از بقیه استفاده کنیم.

مربوط به سیستم عامل است و اگر در 8 بایت جا نشود به قسمت‌های دیگر حافظه jump می‌زند.

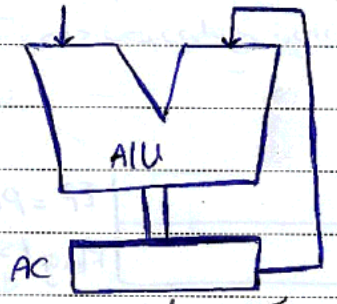
$FF\ FFFF = 2^{24} - 1 = 16^6 - 1$

16^6

برای نمایش این ۶ F استفاده کردیم.

AC (Accumulator):

انباشتگرهایی هستند که در مقابل ALU قرار دارد و مقادیر خروجی ALU را در خود نگه می‌دارد و معمولاً Feed Back می‌شوند. در هر پردازنده‌ای چند وجود دارد.



stack pointer: آدرس آخرین خانهٔ stack را مشخص می‌کند تا داده در آن قرار گیرد.

Base pointer: یک نقطهٔ ثابت را به عنوان مبنا قرار می‌دهیم و بقیه آدرس‌ها را با مقایسه نسبت به آن بدست می‌آوریم.

Sourse pointer: زمانی که می‌خواهیم انبوهی از داده‌ها را در حافظه قرار دهیم از دستورات پیچیده‌تری استفاده می‌کنیم. مثلاً برای کپی کردن یک آرایه از یک جا به یک جای دیگر از حافظه این آرایه ذخیره شود که نقطهٔ شروع آن sourse pointer و نقطهٔ پایان آن distination pointer است.

www.jozveha.ir

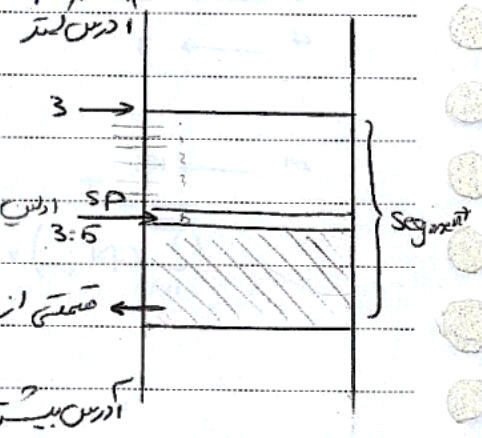
Subject:
Year. Month. Date. ( )

مثال: اگر register ها در شرایط زیر باشند در چنین شرایطی کدام خانهٔ stack آدرس دهی می‌شود یا آدرس خانهٔ مشخص شده چیست؟

segment reg.

4	code	5
18	data	
3	stack	
23	extra	

stack pointer



آدرس خانهٔ خالی را نشان می‌دهد

قسمتی از stack که پر شده

در واقع segment که stack در آن قرار دارد توسط stack seg. مشخص می‌شود و آخرین خانه از stack که داده در آن قرار دارد توسط stack pointer. باید توجه داشت که stack از آخر پر می‌شود پس خانهٔ خالی نشان داده شده دارای آدرس زیر است.

$3 \times 16 + 5 = 63$

از پایین stack پر می‌شود ولی آدرس دهی از آدرس کمتر به سمت آدرس بیشتر است

یک داده ممکن است بیشتر از یک segment باشد ولی در یک لحظه فقط با یک segment می‌توان کار کرد و با عوض کردن مقدار code segment به خانهٔ مورد نظر رسید.

دستوراتی که در اسمبلی می‌نویسیم کار کردن با رجیستردهاست یا قرار دادن یا برداشتن اطلاعات به/از رجیسترها

MOV Ax, Bx ⟶ Ax ← Bx (کپی می‌شود) Ax := Bx;

دستوری است که سریع اجرا می‌شود و تنها 2 کلاک زمان می‌گیرد

DIV Ax, Bx ⟶ تقسیم Ax بر Bx است که تقسیم 16 بیتی بر 16 بیتی کندترین دستور است که 25 clk طول می‌کشد.

CPI (clock per instruction):

یک واحد اندازه‌گیری یا قدرت پردازشی است و می‌گوید هر instruction چند clock طول می‌کشد. که در 80286 بین 2 تا 25 است.

زمان اجرای برنامه = $m \times CPI \times \frac{1}{f}$

m: تعداد دستور، f: فرکانس clock

$\frac{1}{f}$ = clock time = duration

زمانی که یک clock طول می‌کشد

www.jozveha.ir

Subject:
Year. Month. Date. ( )

دستور		تعداد clock
1	⟶	2
2	⟶	3
3	⟶	2
4	⟶	3
⋮		
m	⟶	15

برای مشخص کردن سرعت اجرا از CPI در کنار فرکانس استفاده می‌کنیم که باید مفهوم سرعت مشخص شود.

$$\left(\sum_{i=1}^{m} CPI_i\right) \times \frac{1}{f} =$$ مدت زمانی که اجرای برنامه طول می‌کشد.

تکنولوژی ارتباط دو بخش

1- wait: درخواست را به طرف مقابل می‌دهیم و آن قدر صبر می‌کنیم تا جواب را بگیریم. این روش خوبی نیست زیرا ۱- resource را بدون استفاده نگه می‌داریم ۲- نمی‌دانیم چقدر طول می‌کشد باید wait کنیم.

2- polling (سرکشی): درخواست را می‌دهیم و مشغول کار دیگری می‌شود و هر دفعه سر می‌زند که ببیند نوبت خودش شده یا نه. این هم روش خوبی نبود زیرا از دستگاه مقابل اطلاعی نداشتیم و باید چندین بار سر می‌زدیم و وقت تلف می‌شد.

3- interrupt (وقفه): بعد از فرستادن درخواست مشغول کار دیگری می‌شود و هر وقت داده آماده شد خودش خبر می‌دهد. زمان استفاده از این از سخت افزار شروع شد مثل ارتباط دستگاه‌های IO (پرینتر، صفحه‌کلید، ...) و cpu. از این روش در نرم‌افزار هم استفاده شد: زمانی که می‌خواهیم یک دستور تحت هر شرایطی اجرا شود و سایر کارها متوقف شود مثل تقسیم بر صفر یا گرفتن Alt + Ctrl + del

از pentium به بالا در پردازنده‌های 386 همه رجیسترهای آن ۳۲ بیتی است.

Extended

	0		Am	15
EAx				
EBx				
ECx				
EDx				

www.jozveha.ir

Subject:
Year. Month. Date. ( )

PC: یک Pointer است که درون code seg یکی یکی جلو می‌رود (programming counter)

Flag: وضعیت حال پردازنده را نشان می‌دهد و هر خانه از رجیستر 16 بیتی نشان دهنده یک عضو است.

وقتی در حالت protected mode است از این دو استفاده می‌کنیم.

مشخص می‌کند آخرین عملیات carry داشته یا نه (carry flag) برای عملیات جمع و تفریق

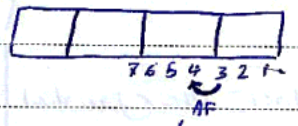
$$\begin{array}{r} 1101 \\ +\ 0101 \\ \hline (1)\ 0010 \end{array}$$

carry

1 ← دارد carry
0 ← ندارد carry

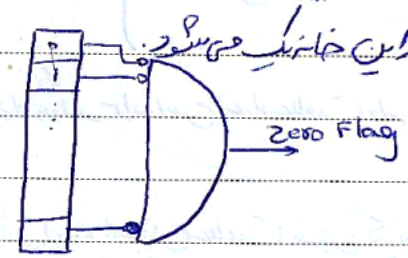
PF (parity flag): از نوع even است و برای کشف خطا از آن استفاده می‌شود.
1: تعداد یک نتیجه فرد
0: تعداد یک نتیجه زوج

(ACF) Auxilary Flag: carry داخلی را در نمایش به صورت BCD مشخص می‌کند

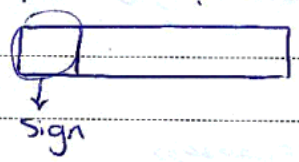


carry از ۳ به ۴ را در خود نگه می‌دارد.

(ZF) zero Flag: هرگاه نتیجه محاسبات صفر شود مقدار این خانه یک می‌شود.



(SF) Sign Flag: اگر عدد منفی و مثبت داشته باشد با صفر و یک شدن این خانه نمایش می‌دهد. همیشه بیت آخر نتیجه را در خود نگه می‌دارد و اگر فرض بر این باشد که همه اعداد مثبت هستند MSP را نشان می‌دهد.



PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Trap Flag (TF): با یک شدن این خانه به mode می‌رود که به آن single step می‌گوییم.

single step: (تک گام) اگر اجرای برنامه به این حالت برود پردازنده این قابلیت را می‌دهد که دستور به دستور انجام شود و بتوانیم مقادیر را بعد از اجرای هر دستور ببینیم.

Interrupt Flag (IF): بیت وقفه است که به آن (Interrupt Enable) هم گفته می‌شود. اگر مقدار این بیت ۱ باشد وقفه‌ها را دریافت می‌کند و اگر مقدار صفر باشد وقفه‌ها disable می‌شود. مقدار default آن ۱ است.

Direction Flag (DF): براساس مقادیر source index و destination index بعد از اجرای عمل بین این دو فاصله هر دو مقدار یکی به جلو می‌رود که برای مقایسه آرایه‌ها مناسب است. انجام این کار از طریق DF امکان‌پذیر است.

SI, DI ← -1 ← 1 } Direction Flag

SI, DI ← +1 ← 0

OF (OverFlow Flag):

سرریز:
- اعداد علامت‌دار: اگر علامت‌دار باشد بیت MSB را بیت علامت فرض می‌کنیم
  - جمع دو عدد مثبت ← منفی
  - جمع دو عدد منفی ← مثبت
- اعداد بدون علامت: در این حالت carry دو بیت آخر مشخص کننده سرریز بودن یا نبودن است.

پردازنده براساس اعداد علامت‌دار OF را مقداردهی می‌کند.

```
  0110 ....
 +1011 ....
 (1) 0001
```

اگر اعداد را بدون علامت فرض کنیم چون یک عدد مثبت بزرگ و با علامت مثبت دیگر جمع شده و حاصل یک عدد کوچک است پس overFlow اتفاق افتاده ولی برای اعداد علامت‌دار چون یکی مثبت و دیگری منفی است و حاصل یک عدد مثبت شده overflow اتفاق نیافتاده و (1) carry است.

```
  1011
 +1001
 (1) 0100
```

دو عدد منفی جمع آن‌ها یک عدد مثبت شده است

www.jozveha.ir

Subject:
Year. Month. Date. ( )

برنامه‌سازی به زبان اسمبلی:

۱- این زبان key sensitive نیست یعنی به حروف کوچک و بزرگ حساس نیست.

۲- برای نمایش اعداد به صورت زیر عمل می‌کنیم:

- Decimal ← 198D
- Binary ← 10110 1B
- Hex ← 4F5 H
- "ALI" ← می‌توان برای نمایش عدد از کلاکت‌های آن استفاده کرد.

اسکی	اسکی	اسکی
A	L	I

۳- برای نام متغیرها نیز می‌توان از 31 character زیر استفاده کرد:

- 0 → 9
- a → z
- A → Z
- @ _ ? . $

۴- برای مشخص کردن Function یا Procedure از دو ساختار زیر می‌توان استفاده کرد که بر اساس شرایط می‌توان برای آن Arg ورودی یا return مشخص کرد.

- PROC → هر جا که تابع صدا زده شود برنامه‌ی Procu. پرش می‌کند
- ENDP

- MACRO → هر جا که تابع صدا زده شود کل دستورهای شامل آن در محل مربوطه کپی می‌شود.
- ENDM

۵- از میان دستورهای کنترلی فقط Jump وجود دارد.

- Jmp x → به خانه‌ای که label مشخص می‌کند پرش می‌کند.
- JNZ x → اگر مقدار قبلی صفر باشد عمل پرش صورت نمی‌گیرد. jump not zero

www.jozveha.ir

بر حسب آدرس دستور در حافظه است و نام متغیر نیز آدرس اطلاعات در حافظه است.

Subject:
Year. Month. Date. ( )

۶- برای نمایش دستورات شرطی از Jmp استفاده می‌کنیم:

```
if a>0 then
    [ A ]
else
    [ B ]
```

```
MOV  Ax : 0
JNZ  y
[ B ]
Jmp  z
y: [ A ]
z:
```

اگر این جمله نبود بعد از اجرای B ادامه می‌داد و A را دوباره اجرا می‌کرد برای همین بعد از A پرش می‌کنیم.

کد اسمبلی فقط شامل دستورات اجرایی توسط cpu نیست بلکه به دو قسمت زیر تقسیم می‌شود:

۱- Directive: توسط cpu اجرا نمی‌شود و دستور نیست. در واقع دستوراتی هستند که هیچ کمکی به پشت آن‌ها نیست که cpu اجرا کند. تنها برای مهم کار برد compiler است.

```
c:      #define MAXNUM 9999
pascal: CONST MAXNUM 50
```

همه دستورات PROC، ENDP و ... از این نوع هستند.

۲- Instruction: برای ارتباط با cpu هستند و توسط آن پردازش می‌شوند مثل:

```
MOV Ax, Bx
```

اگر در یک برنامه هر دو Directive و Instruction وجود داشته باشد عملیات اجرای آن در یک مرحله اتفاق نمی‌افتد و نمی‌توان گفت فقط complite یا Asamble می‌شوند بلکه شامل دو مرحله زیر است.

۱- directive ها شناسایی و مخفیده می‌شوند.

۲- مجموعه instruction اسمبل می‌شود.

۷- comment از هر جایی که علامت ; گذاشته شود تا آخر خط محسوب می‌شود.

Instruction : [label :] op code [operand] [;comment]

لیبل جایی که امکان پرش به این خط وجود دارد باید بنویسیم.

mnemonic: دستورهایی که می‌خواهیم به وسیله آن کاری انجام دهیم

STD → DF = 1 از دستورهایی است که operand ندارد.

set direct Flag

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Directive : [name] Directive [operand] [; comment]

MACRO

می‌تواند یک اسم قرار گیرد.

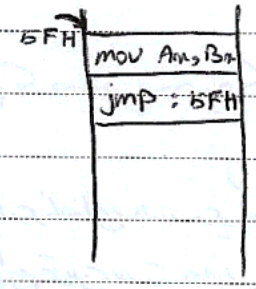
اگر دستور نوشته شده Directive یا Instruction باشد با حروف بزرگ می‌نویسیم و متغیرها را با حروف کوچک مشخص می‌کنیم.

ISA (instruction Set Architecture): مشخص می‌کند که دستورها و عملیات استاندارد برای پردازنده‌ها وجود چه خصوصیاتی دارند. در ضمن در مدل‌های بالاتر مشخصات ISA مدل‌های قبلی را شامل می‌شود.

86 ISA 1
286 ISA 2
386
486
pentium

نکته: label ها خودشان دارنده چیزی نیستند بلکه نشان دهنده یک عدد هستند.

```
Label 1:  MOV  An, Bn
          jmp  Label 1
```



directive :

```
# define n 50
void main()
{
  int a;
  a = n;
}
```

سرعت بیشتر و ثابت کمتری مصرف می‌کند

instruction:

```
void main()
{
  int a, n;
  n = 50;
  a = n;
}
```

بهتر است از تلفیق dir و instruction استفاده شود.

#include<conio.h> کل فایل مربوط را در اول برنامه کپی می‌کند

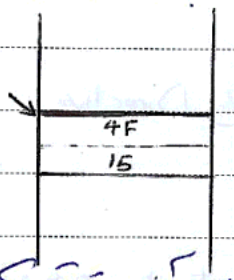
www.jozveha.ir

Subject:
Year. Month. Date. ( )

نحوه ذخیره سازی word ها:

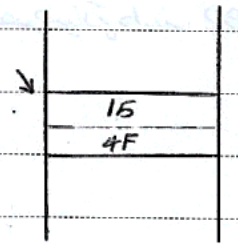
1- Little_endian : در هنگام ذخیره سازی عدد برعکس از آخر به اول ذخیره می کند. در عمل little_endian روش مناسب تری برای ذخیره است.

مثال: 154F H
۸ بیت ۸ بیت



2- Big_Endian : همان روش ابتدایی است که اعداد از اول به آخر و به ترتیبی که هست ذخیره می شود.

مثال: 154F H



انتخاب روش ذخیره سازی به دو شیوه صورت می گیرد:

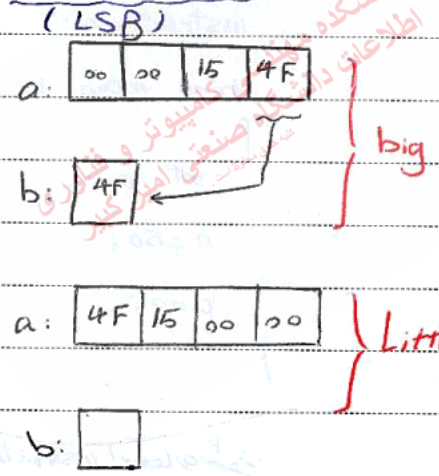
۱- پردازنده روش را مشخص می کند.

۲- سیستم عامل (OS)، حتی اگر پردازنده شیوه ای را انتخاب کرده باشد سیستم عامل می تواند آن را برعکس کند.

برای تغییر روش ذخیره سازی باید برنامه ای بنویسیم که دو بایت ذخیره کند و یک بایت بخواند.

little_endian روش مناسب تری برای ذخیره است زیرا در برنامه ای مانند زیر اگر به روش دوم ذخیره شود باید جدا پیدا کردن عدد مورد نظر ۴ بایت جلو رفت تا به کم ارزش ترین بیت ها رسید.

```
int a;
char b;
a = 154F H;
b = (char) a;
```



دانشکده فنی و مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

و زمانی از استفاده از DB ، DW یا DD اگر از حروف هگزادسیمال A ، B و C ... استفاده می‌کنیم. در صورتی

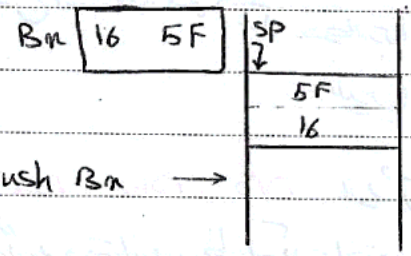
Subject: Data DB 0A5H

Year. Month. Date.

که رقم اول باشد قبل از آن 0 می‌گذاریم که مشخص شود این حرف یک عدد است.

H ← هگزادسیمال B ← باینری D یا بدون علامت ← دهدهی

نکته: نحوه ذخیره در stack نیز به همین صورت است ولی با وجود اینکه از آخر پر می‌شود هنگام push کردن به تعداد بایت‌ها بالا می‌رود و بعد به همان روش ذخیره می‌کند.



Directives:

۱- INCLUDE: می‌توان فایل‌های دیگری که از قبل نوشته شده را به برنامه منتقل کرد، دقیقاً مانند به اول برنامه کپی می‌کند.

```
INCLUDE    Fact.ASM
```

۲- PROC , ENDP: برای نوشتن تابع استفاده می‌شود ولی return و آرگومان نمی‌تواند داشته باشد و برای اینکار می‌توانیم خودمان سازی کنیم مثلاً رجیستر AX را مختص آرگومان قرار دهیم و قبل از استفاده از FACT ، AX مقداردهی شود و همین طور برای return. نمی‌توان از PROC های تودرتو استفاده کرد و باید برنامه flat باشد.

```
FACT PROC
   :
   :
   RET ⟶ به جای اصلی برنامه برمی‌گردد.
ENDP
```

۳- MACRO , ENDM: نوشتن ماکرو سخت‌تر از PROC است و لزوماً با عوض کردن کلمه MACRO و PROC برنامه درست کار نمی‌کند. از این روش در source generator ??? استفاده می‌کنیم برنامه‌هایی که برای خود یک منبع هستند. همچنین می‌توان با macro یک برنامه recursive نوشت. چگونه؟؟

```
FACT  MACRO
   =
   =
   =
ENDM
```

۴- ASSUME: به وسیله این دستور می‌توان segment های مورد نظر را مقداردهی کرد.

```
ASSUME  CS : My code ⟶ می‌توان یک seg با کد را نام‌گذاری کرد.
        PS : ____
        ES :
        SS : NOTHING ⟶ هیچ مقداری در دستش قرار نگیرد و خالی شود
```

دانشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year: Month: Date:

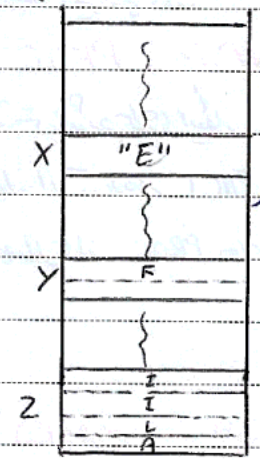
می‌توانیم از دستور (=) هم استفاده کنیم با این تفاوت که از طریق = می‌توان دوباره مقدار متغیر را عوض کرد ولی با EQ نمی‌شود.

۵- EQ : اگر بخواهیم متغیر global بگیریم یا همانند define مقداری را به یک متغیر نسبت دهیم از این روش استفاده می‌کنیم.

NAME EQ "ALI" → هر جا در برنامه Name باشد "ALI" جایگزین می‌کند.

۶- DB , DW , DD : با نوشتن این دستورها متناسب با مقدار بایت هر کدام خانه‌هایی از حافظه را در نظر می‌گیرد و مقدار مربوط را به آن اختصاص می‌دهد.

۱ DB define byte — X DB "E"
۲ DW define word — y DW 15 (در مبنای ۱۰)
۴ DD define double — Z DD "ALI"



می‌توان از طریق این روش Data را در جایی که قرار داد و با اختصاص دادن اسم به آنها برای صدا زدن آنها استفاده می‌کنیم.

قرار داشتن داده‌ها در جایی که شکل سازی می‌شود چون instruction نیستند و اگر دستور دیگر داشته باشد به اشتباه عمل می‌کند پس باید از روی آن jmp کنیم.

یا همهٔ define ها را در اول کد قرار دهیم.

یکی از استفاده‌های آن زمانی است که می‌خواهیم بدون استفاده از اسمبلر مستقیم داده‌ها را درون حافظه قرار دهیم.

۷- SEGMENT : از طریق این دستور قسمتی از کد به اندازهٔ 1 seg را با مشخصات خاص و نام مشخص می‌کنیم و در هنگام استفاده از آن می‌توانیم نام آن را صدا زده و دستورهای مربوط را اجرا کنیم.

SEGMENT 16 ← para (paragraph دقیقه) — public 'data'
1 ← Byte — common
2 ← word — stack
256 ← page

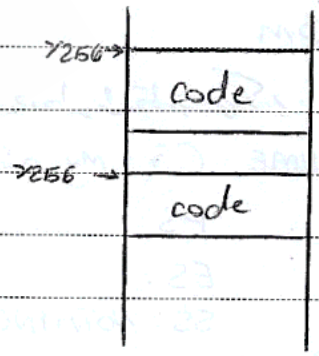
اسم را اینجا تعیین می‌کنیم

نوع seg است.

ذخیره آدرس دهی و ذخیره سازی درون حافظه را مشخص می‌کند.

لیست قرار دادن seg های شبیه

scope که معادل یک seg باشد از طریق SEGMENT انجام می‌شود.



ENDS

دانشکده مهندسی کامپیوتر و مشاوری دانشگاه صنعتی امیرکبیر

www.jozveha.ir

کلمه stack فقط در سگمنت پشته استفاده می‌شود و گذاشتن آن در برنامه اجباری است. در این حالت سگمنت پشته برنامه کاربر با سگمنت پشته سیستم عامل ترکیب می‌شود

Subject:
Year. Month. Date. ( )

para: مخفف paragraph و مشخص می‌کند seg ی که در وسش ذخیره شود مضرب 16 باشد

public: ممکن است که مربوط بیش از یک seg باشد و خودمان باید آن‌ها را تقسیم کنیم و نام مشترک برای آنها قرار دهیم public مشخص می‌کند که این seg های مشابه در حافظه به یکدیگر بچسبند.

stack ← پشته سرهم

common ← overwrite

۸- ORG: بوسیله‌ی آن می‌توان تنظیم کرد که داده‌ها را دقیقاً در کدام آدرس قرار دهد.

ORG 100H ⟶ کد را در آدرس 256 قرار دهد.

16^2 16^1 16^0

DB 'A' ⟶ در آدرس 256 کد 'A' را قرار می‌دهد.

۹- public , Extern: گاهی برنامه آنقدر بزرگ است که در چندین فایل ذخیره شود ولی متغیرهای global در هر دو مشترک است برای استفاده از آن از این واژه‌ها استفاده می‌کنیم.

A.asm		B.asm
public Name	DD 'A'	Extern Name ⟶ به همان متغیری اشاره می‌کند که مقدار آن 'A' است.

برای وصل کردن این فایل‌ها و ساختن یک برنامه exe به کمک اسمبلر و لینکر این کار را انجام می‌دهیم.

لیست دستورهایی که در sourse code نوشته شده نشان می‌دهد قابل فهم برای کاربر است.

Borland (در دوره‌ی لینکر ساخته می‌شود)
- tasm.exe: a.asm ⟶ a.obj , a.Lst
- tlink.exe: a.obj ⟶ a.exe

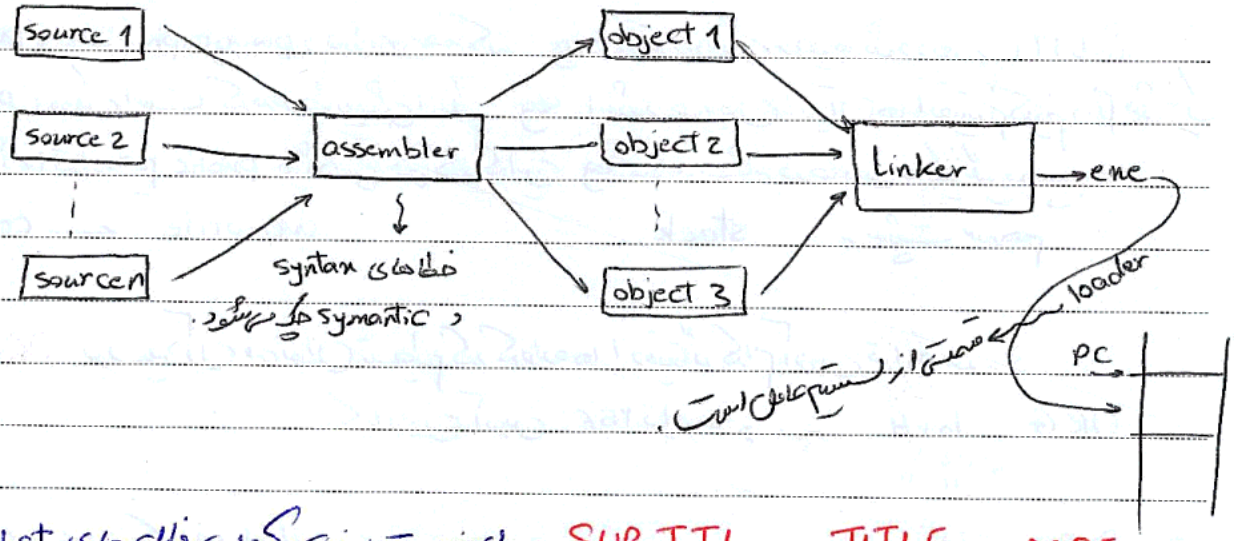
می‌توان با انتخاب چندین sourse برای هر کدام یک obj. ساخت و بعد همه‌ی این‌ها را با لینکر به exe تبدیل کنیم

Microsoft
- Masm.exe
- Link.exe

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )



۱۰. PAGE , TITLE , SUBTTL : برای فرمت بندی کردن فایل های .lst به کار می رود تا برای کاربری که استفاده می کند قابل فهم تر باشد.

PAGE 60 , 20 → صفحه شامل این اعداد در این تعداد خط می‌شود
(تعداد خطوط در صفحه ، تعداد حروف در خط)

TITLE 'My Prog' → این نام به کل document اختصاص داده می‌شود (حداکثر تعداد طول کلمات ۶)

SUBTTL 'SUB' → به هر صفحه یک عنوان اختصاص می‌دهد و تکرار می‌شود.

۱۱. .286C , .286P , .8086 : مشخص می‌کند که instruction های نوشته شده مربوط به کدام پردازنده و متناسب با کدام است.

.8086 → دستورها مربوط به پردازنده 8086 است

.286C → real mode
.286P → protected mode

اگر مشخص نشود به صورت پایه .286C در نظر گرفته می‌شود.

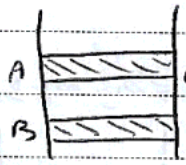
دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر
اطلاعات

www.jozveha.ir

Subject:
Year. Month. Date. ( )

12. EVEN : دستورهای مربوط به بعدی را به آدرس های زوج اختصاص می دهد.

```
EVEN
A   DB   5
B   DB   10
```



با وجود اینکه هر کدام یک بایت مصرف می کنند ولی زوج در خانه های زوج قرار می دهد.

13. RET , IRET : هنگامی که یک proc صدا زده می شود با وجود اینکه ENDP دارد ولی باید دوباره به خط که مورد نظر انتقال پیدا کرد که این کار از طریق RET انجام می دهیم

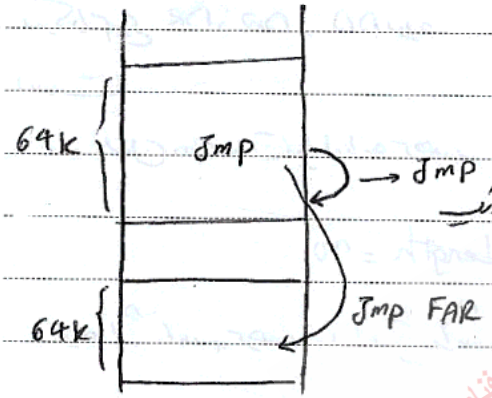
RET → call → صدا زدن معمولی

IRET → INT → صدا زدن با interrupt

اگر proc بودیم interrupt صدا زده شده باشد هنگام خارج شدن از آن نیز از IRET استفاده می کنیم.

صفات آدرس ها (NEAR , FAR) :

از این صفات برای آدرس دهی های دور و نزدیک استفاده می کنیم اگر آدرسی که می نویسیم در همان seg. باشد NEAR استفاده می کنیم ولی اگر بیش از یک seg. فاصله داشته باشند از آدرس FAR استفاده می کنیم.



می توان برای proc از آن استفاده کرد. اگر قرار بود از segment دیگری نیز صدا زده شود Far و اگر یک روال داخلی باشد از near استفاده می کنیم.

```
Main proc Far
Main ENDP
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

نماد ها : (keyword)

1) NUM DB ? → حسابی در مورد مقدار نداریم هم نیست چه مقداری قرار گیرد در زمانی که نمی خواهیم مقدار دهی کنیم استفاده می کنیم

2) MYARR DW 256 DUP(0) → به تعداد 256 عدد صفر را تکرار کن

⇓

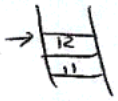
MYARR DW 0, 0, ..., 0 (256)

یعنی عمل DW، 256 بار تکرار می شود و 0 در بایتی است. زمانی که MYARR را صدا می زنیم اولین خانه را صدا می زنند برای دسترسی به خانه های بعدی باید عددی را با MYARR جمع کنیم

www.jozveha.ir

Subject:
Year. Month. Date. ( )

DW 1112H



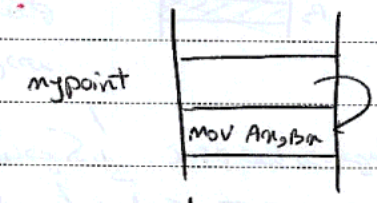
20 DUP(54??3?)

3) MYVAR DB 20 DUP(5,4, 2DUP(?), 3,?) →

20 بار ساختار داخل پرانتز تکرار می‌شود.

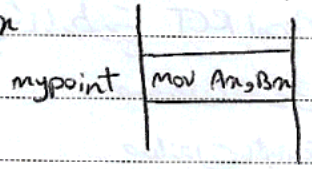
4) mypoint DW HERE → آدرس با مقدار program counter را در mypoint قرار می‌دهد یا آدرس خودش.

MOV AX, BX



→ آدرس خانه بعدی در آن قرار می‌گیرد

mypoint : MOV AX, BX



5) SEGMENT Num → آدرس seg. متغیر مربوط را برمی‌گرداند.

MOV AX, Segment num

6) OFFSET Num → offset متغیر Num در segment را نشان می‌دهد.

7) TYPE Num →
- 1 → از نوع بایت
- 2 → از نوع word
- 4 → از نوع double

مشخص می‌کند type متغیر مورد نظر از کدام نوع DB، DW، DD بوده است.

8) LENGTH Num → تعداد المان‌ها است یا اندازه برحسب type

MYVAR DW 20 DUP(?, 'A') → Length = 40

9) SIZE num → اندازه شناسه برحسب بایت را نشان می‌دهد.

MYVAR DW 20 DUP(?, 'A') → Size = 80

SIZE = LENGTH * TYPE

Data DB 'tehran_iran' → که اصلی محاوره عبارت را در خانه‌های پشت سرهم قرار می‌دهد

Data DB 5,6,9,2,3 → " " " " با شروع از آدرس data اعداد را در خانه‌های

Data DB '5','6','9' → " " " " " " کد اسکی اعداد را

رشته‌ها توسط DB در حافظه قرار می‌گیرند که برای هر حرف یک خانه در نظر می‌گیرد.

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

نکته: می‌توان شناسه‌های دستورات قبل را برای یک آدرس نیز مشخص کرد.

```
mycode: MOV  Ax, Bx
```

Length mycode : 1 ← یک آدرس آرایه‌ای ندارد در یک خانه قرار می‌گیرد.

Size mycode : X معنی ندارد

Type mycode : NEAR ← offset ذخیره می‌شود -۱ ، FAR ← offset : seg ذخیره می‌شود -۲

حداقل دستوراتی که باید در یک برنامه اسمبلی نوشته شود مانند زیر است:

```
SEGMENT  para  public  Dataseg

ENDS
SEGMENT  para  public  Codeseg
   ASSUME   cs: Codeseg , Ds: Dataseg , ss: stackseg , Es: Nothing
   MOV      Ax, Dataseg
   MOV      Ds, Ax
ENDS
SEGMENT  para  ~~public~~ stack  STACKseg

ENDS
END
```

نمی‌توان یک label را مستقیم درون seg قرار داد و باید حتماً همه یک طرف MOV یک reg باشد.

حتماً باشد

فقط برای Dataseg باید جداجدا Make کنیم.

```
MOV  Ax, 4C00H
INT  21H
```

برای آزاد شدن حافظه خارج شدن از برنامه باید در آخر یک دو دستور داشته باشد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

ساده‌تر برای مشخص کردن سگمنت‌ها می‌توان از نماد (.) استفاده کنیم:

```
.data
.code
   MOV  Ax, @data
   MOV  Ds, Ax
   mov  Ax, 4C00H
   INT  21H
.STACK
```

در این حالت اختصاص سگمنت Data به چند صورت زیر می‌شود:

در این حالت فقط یک سگمنت از Data, code و Stack داریم و نمی‌توان با برنامه‌های دیگر لینک کرد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

می‌توان علت جدا قرار دادن کردن آدرس segment برای Data این طور گفت که برای stack در تعریف SEGMENT مشخص شده است ولی Data به طور دیگر باید مشخص شود.

عملکرد linker :

در هنگام نوشتن کدها از label ها برای دسترسی به خطوط دیگر کد استفاده می‌کنیم زمانی که برنامه اسمبل می‌شود در فایل‌های obj هم label ها از بین می‌رود و به جای آن‌ها آدرس‌ها را قرار می‌دهد.

A.obj

0	MOV Am, Bn
	jmp 100
100	call B
	⋮
400	

B.obj

0	MOV Am, B
150	jmp 200
200	call C
500	

C.obj

0	DB 8
	jmp 400
400	END

وظیفه linker وصل کردن این obj ها به یکدیگر و تعریف آدرس‌ها زمانی که همه این فایل‌ها در اختیار هم قرار گیرند.

D.EXE

0	MOV Am, Bn
100	jmp 100
	call 401
400	
401	MOV Am, Bn
550	jmp 660
660	CALL
900	
1300	

پس وظیفه لینکر خیلی حساس است باید جدولی داشته باشد تا تمام آدرس‌هایی که وجود دارد را براساس obj های متصل شده به هم تغییر دهد. که پس از آماده شدن فایل exe و اجرا کردن آن وظیفه loader است که آدرس‌های موجود در فایل exe را متناسب با جایی که در حافظه قرار می‌گیرد تغییر می‌دهد ولی برای اینکه loader مجبور نباشد برای هر بار اجرا تمام آدرس‌ها را عوض کند زمانی که لینکر آدرس‌های داخلی را مشخص می‌کند از relative adressing استفاده می‌کنیم یعنی

jmp 4
4 call B

هر آدرس متناسب با اجزای خود مشخص می‌شود نه خانه‌های حافظه

دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

نکته: می‌توانیم تمام کارهایی را که به صورت روتین هر بار می‌خواهیم انجام دهیم را درون یک batch file ذخیره کنیم، که برای اینکار داخل یک notepad کارهای مورد نظر را نوشته و فایل را با پسوند .bat ذخیره می‌کنیم.

```
Source  DB  'ABCD'
```

خودش می‌فهمد که باید هر حرف را در یک بایت قرار دهد.
'A','B','C'

مثال: تشخیص اینکه حافظه‌ها Little_end یا Big_Endian است از طریق برنامه زیر صورت می‌گیرد:

```
PAGE  40,60
TITLE 'Hello world'
```

نوع seg را نشان می‌دهد.

```
myStack  SEGMENT  para  stack  'stack'
    DB  256 DUP(0)
```

اگر این جمله نباشد حافظه‌ای به stack اختصاص نمی‌دهد.

```
ENDS
myData  SEGMENT  para  public  'data'
Source  DB  "ABCD"
X       DW  "XY"
```

باید عدد باشد تا درست کار کند.

متناسب با آن ۴ بایت DB می‌کند.

```
LITEND  DB  'Little-Endian$'
BIGEND  DB  'Big-Endian$'
ENDS
```

به صورت string قرار می‌دهد و می‌فهمد که باید تا $ بخواند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

```
myCode  SEGMENT  para  public  'code'
    Assume  CS:myCode, DS:MyData, SS:MyStack, ES:Nothing
    MOV  Ax, MyData
    MOV  DS:Ax

    MOV  AL, 'Y'
```

www.jozveha.ir

Subject:
Year. Month. Date. ( )

```
LEA BP, X
```
آدرس شروع X را درون BP قرار می‌دهد

```
MOV BL, [BP]
```
محتوای درون آدرس BP را نشان می‌دهد.

```
CMP AL, BL
```
دو عدد را با هم مقایسه می‌کند

```
JE Equal
```
jump if equal

```
;Big_EDDian
LEA Dx, Bigend
MOV AH, 09H
INT 21H
```
بوسیله این سه دستور string مربوط به روی صفحه چاپ می‌شود.

```
JMP ENDPROG
```
باید jump کند تا شرط دوم نیز اجرا نشود زیرا شرط اول برقرار شده

```
EQUAL:
;Little-ENDian
LEA DX, Litend
MOV AH, 09H
INT 21H

ENDPROG:
MOV AX, 4C00H
INT 21H

ENDS

END
```

این برنامه درست اجرا نمی‌شود زیرا string ها همان ترتیبی که نوشته می‌شوند در حافظه ذخیره می‌شوند. باید DW کنیم ولی یک بار آنا بتوانیم little endian را درست تشخیص دهیم

```
X DW FE80H
```

89
FE

دانشکده مهندسی کامپیوتر
اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

نکته: کامپایلرها به دو صورت عمل می کنند: در برخورد با داده های static و dynamic متفاوت هستند. اگر داده static در برنامه تعریف شده باشد در حجم برنامه تاثیر گذار است ولی اگر dynamic باشد در اندازه برنامه تاثیری ندارد. (static یعنی در طول برنامه مقدار مشخصی داشته باشد.) ولی ظرفیتی که به داده ها اختصاص می دهند محدود است.

COM , EXE: فایل های باینری هستند که تنها از کدهای صفر و یک ساخته شده است. معروف ترین فایل com مورد استفاده command.com است ولی محدودیت های زیادی دارند. مشخصات آن عبارت است از:

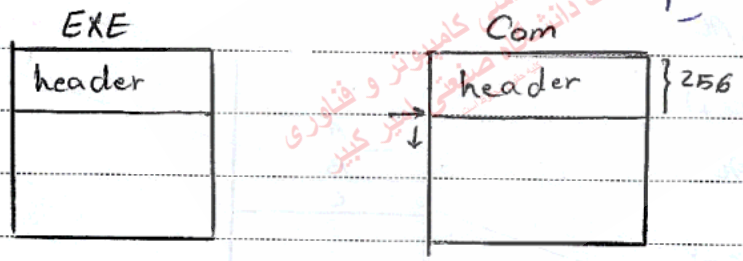
۱- تنها از یک segment استفاده می کند (segment های داده، کد و ... یکی هستند.)

۲- اندازه فایل محدودیت دارد و نباید از 64K (اندازه یک seg) بیشتر شود.

۳- segment های مختلف با یکدیگر یکسانند.

Data segment
Code segment
Stack Segment
Extra segment
} یکی هستند

۴- اولین دستور (directive) که باید نوشته شود: ORG 100H است. زیرا در این ها لازم است تا حجمی در اول برنامه header اختصاص داده شود. در header اطلاعاتی در مورد اندازه، طول خط و ... از برنامه برای اطلاع سیستم عامل نوشته می شود.




```
myseg   Segment   para   public   'code'
        Assume    cs:myseg , Ds:myseg , Es:myseg , SS:myseg
ends
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

stack در طول برنامه متناسب با حجم تو در تو بودن PROC ها هستند و برای استفاده از stack مناسب نیستند زیرا حجم زیادی را مصرف می‌کنند.

5. stack ندارد.

6. قرار دادن برچسب شروع برای END الزامی است.

7. همه PROC ها و Jump / call باید از نوع Near باشند.

دلیل استفاده از com سرعت زیادتر آن نسبت به exe است.

چگونه کامپایلر تبدیل دارد یا نه

exe2bin.exe → هر فایل exe قابل تبدیل به com توسط این برنامه به com تبدیل می‌شود.

مثال: نمونه‌ای از برنامه com در نظر داریم. برعکس کردن داده‌ها از source به Dest

```
page 140
TITLE 'com_file'
CSeg SEGMENT para public 'code'
    Assume Cs:Cseg, Ds:Cseg, SS:Cseg, ES:Cseg
    ORG 100H
main: Jmp start
    source DB 'ABC'
    DEST DB 3 Dup(?)
start: Mov AL, Source
    MOV DEST+2, AL
    MOV AL, Source+1
    MOV Dest+1, AL
    MOV AL, Source+2
    MOV Dest, AL
    MOV Ax, 4C00H
    INT 21H
ends
END Main
```

به اولین دستوری که در حافظه قرار می‌گیرد اختصاص می‌دهیم

Assembler قبل از اجرای برنامه همه directive ها را جایگزین می‌کند و اعمال می‌کند و بعد پردازش برنامه را اجرا می‌کند. همچنین مکانیزم اسمبلر حالا این نیست که دستورات را به ترتیب بخواند. در نتیجه DB می‌شود و در هنگام اجرای دستورات از روی داده‌ها jump می‌کنیم که به عنوان کد خوانده نشوند.

A
B
C
?
?
?
jmp start

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

شیوه آدرس دهی در 80x86 و 80286:

شیوه های متفاوت برای دسترسی به داده مورد نظر را شیوه آدرس دهی می گوییم.

1) RISC: Reduced instruction set computer

دستهای از پردازنده ها هستند که دستورات را ریز کرده و بسیار ساده هستند ولی source code بزرگ می شود.

2) CISC: Complex instruction set computer

دستورات پیچیده است و یک دستور می تواند کارهای زیادی را انجام دهد. بعد از مدتی استفاده از CISC متوقف شد زیرا از بسیاری از امکانات آن استفاده نمی شد. قدرت CISC از RISC بیشتر است.

از ویژگی های CISC عبارتند از:

۱- تعداد دستورات ها خیلی زیاد است.

۲- دستورات پیچیده تر است.

۳- سخت افزار پیچیده تر است.

۴- شیوه های آدرس دهی متنوع تر و بیشتر است.

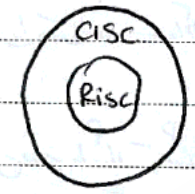
۵- حجم کمتر است.

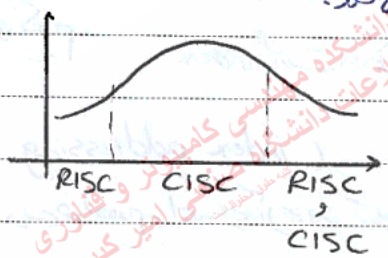
۶- سریعتر است.

۷- طول دستورات متغیر است.

۸- طول دستورات بیشتر است.

در Pentium از ترکیب هر دو استفاده می کند.





دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

انواع شیوه های آدرس دهی:

۱- شیوه آدرس دهی با فصل Immediate addressing:

عین داده ای که احتیاج داریم درون خانه مورد نظر قرار می گیرد. این روش ناقص است چون در بسیاری از موارد مقدار داده را نمی دانیم و در حین اجرا مشخص می شود.

MOV AL, 5

0111	AL	5

max مقداری که می توان دستیابی قرار داد 2^{32} است.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

2- شیوه آدرس دهی ثباتی Register addressing :

داده از یک ثبات به ثبات دیگر منتقل می‌شود.

```
Mov AL, BL
```

3- شیوه آدرس دهی مستقیم (Direct addressing) :

مستقیم داده‌ای در حافظه که اسم label آن را می‌دانیم خوانده و درون ثبات قرار می‌دهیم.

```
Mov AL, Source
```

4- آدرس دهی غیر مستقیم (Indirect addressing) :

در خانه‌ای از حافظه آدرس داده دیگری قرار دارد که از طریق آن به داده مورد نظر دست رسی داریم.

```
Mov Ax, Source
Mov Bx, [Ax]
```

5- آدرس دهی غیر مستقیم با ثبات (Register indirect addressing) :

آدرس داده‌ای در حافظه را داریم که درون یک ثبات است و محتوای آن را داخل ثبات دیگری قرار می‌دهیم.

```
Mov Ax, [Bx]
```

6- آدرس دهی نسبی با ثبات پایه (base relative addressing)

یک ثبات که آدرس پایه را مشخص می‌کند و از طریق آن به داده‌های بعدی دسترسی می‌یابد.

```
Mov Ax, [BP]+4
```

برای jump به آدرس دیگری از حافظه به کار می‌رود.

```
Mov Ax, [BP+4]
Mov Ax, 4[BP]
```

در لیست کردن آرایه‌ها به کار می‌رود که آدرس آرایه را می‌دانیم و هستیم عناصر را متناسب با آن پیدا می‌کنیم.

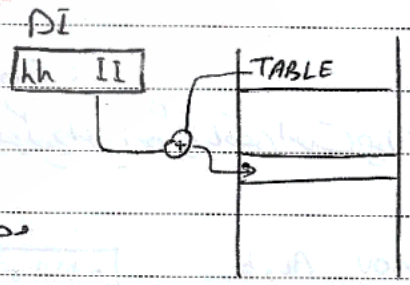
index ثابت و محل متغیر

7- آدرس دهی با ثبات شاخص (index addressing) :

زمانی که مکان آرایه متغیر است و می‌خواهیم به اعضای آن دسترسی داشته باشیم.

DI, SI

```
Mov DI, 5;
Mov Ax, TABLE[DI]
```



محل ثابت و index متغیر

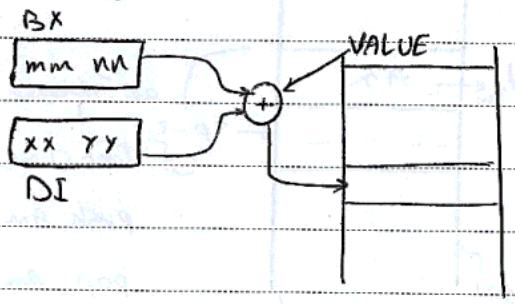
در این روش هنگام دسترسی به خانه‌ای از جدول یا ماتریس از آن استفاده می‌کنیم که خانه اول جدول + اندیس مورد نظر به خانه مورد نظر می‌رود.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

8- آدرس دهی شاخص دار با ثبات پایه (Base index addressing):

MOV An, value [BX][DI]



Instruction Set 80286 :

حدود ۱۰۰ دستور برای این نوع پردازنده وجود دارد.

دستورات انتقال داده:

1) MOV مبدأ ، مقصد

- MOV An, Bn ⟶ An ← Bn

این دستور محدودیت هایی دارد مثل ۱- همزمان دو حافظه قابل استفاده نمی باشد.

MOV An, TABLE
MOV SCR, DEST X ⟶ غلط است

استفاده از دو label امکان پذیر نیست.

۲- همزمان دو seg. register قابل استفاده نمی باشد.

MOV An, CS
MOV DS, An ⟶ MOV DS, CS X ⟶ غلط است

۳- نمی توان مستقیماً درون CS مقداری را قرار دهیم.

۴- نمی توان از عدد همراه seg register استفاده کرد.

- MOV DS, 4 X ⟶ غلط است

نکته: برای استفاده از CS در انتقال داده بجای از دستور Assume استفاده می کنیم.

2) PUSH An
POP An

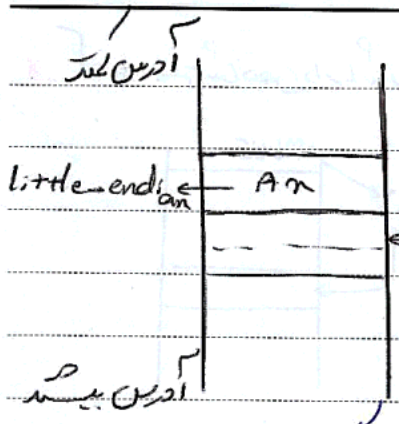
یک مقدار دو بایتی را درون stack قرار می دهد برای استفاده از این دستورها باید قبلاً حافظه ای به stack اختصاص داده شود.

درون این pop reg می شود.

دانشکده فنی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )



زمانی که push می‌کنیم دوتا از sp کم می‌شود و مقدار آن به صورت little-endian درون حافظه قرار می‌گیرد.

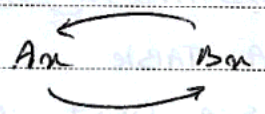
می‌توان به جای نوشتن دستور MOV از push و pop سرهم استفاده کرد. با این تفاوت که MOV سریعتر است و برای استفاده از مکانیزم stack آن باید حافظه‌ای به stack اختصاص داده شده باشد.

push Am
pop Bm

3) pushF — یک Flag register که ۱۶ بیت دارد درون حافظه یا stack قرار می‌گیرد و
popF — سپس با pop کردن ۱۶ بیت از آن خوانده می‌شود.

4) XCHG → exchange — جای Am و Bm را با یکدیگر عوض می‌کند.

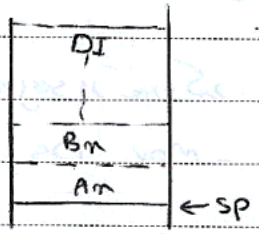
XCHG Am, Bm → { Reg / mem.



XCHG TABLE, Am

5) PushA — می‌توان reg هایی را که می‌خواهیم درون stack قرار دهیم سرهم بنویسیم.
popA — یعنی به جای اینکه یکی یکی درون stack push کنیم، یک جا این کار را انجام
↓ Push All — می‌دهیم. فرق این دو حالت این است که عمل pop کردن به صورت برعکس می‌خواند.

PUSHA { Am, Bm, Cm, Dm } → { SP, BP, SI, DI }



در این حالت زمانی که sp درون stack قرار می‌گیرد مقدار تغییر نکرده sp درون آن قرار می‌گیرد. یعنی ابتدا از تمام مقادیر کپی می‌گیرد و سپس درون stack قرار می‌گیرد.

از این روش وقتی استفاده می‌کنیم که می‌خواهیم تمام مقادیر reg را نگه داریم و این کار را متوقف کنیم و یک برنامه دیگر اجرا شود. در این حالت ترتیب قرار گرفتن درون stack مهم نیست.

push و pop دو عمل برعکس و قرینه هستند و متغیرهای روی روی آنها باید برعکس نوشته شوند.

PAPCO

یا زمانی که وسط stack خالی نداشته باشیم مقداری را push می‌کنیم و داده‌ای بعدی را قرار می‌دهیم.

دانشکده مهندسی کامپیوتر
اطلاعات دانشگاه صنعتی امیرکبیر
مسعود شاوری

www.jozveha.ir

اگر اسم متغیر را بنویسیم با حافظه در ارتباط هستیم.

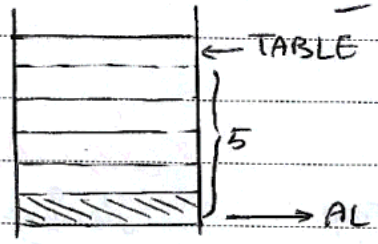
Subject:
Year. Month. Date. ( )

6) XLAT (translate) زمانی که می‌خواهیم به خانه یا درایه ای از یک label دسترسی پیدا کنیم فقط به ۵ خانه بعد از table باید به صورت زیر عمل کنیم:

```
MOV  AL, 5
MOV  Bx, OFFSET TABLE
XLAT TABLE
```

رجیستر های زرد و نگره برای این کار هستند

در این حالت آدرس table را می‌گیرد و به مقدار AL جلو می‌رویم و درایه AL از TABLE را درون AL ذخیره می‌کند.



7) port → IN شماره پورت ، داده
→ out داده ، شماره پورت

برای ارتباط با دستگاه های ورودی و خروجی از این دستور استفاده می‌کنیم هر یک از device ها یک شماره پورت دارند که از سخت افزار خوانده می‌شود. (در این حالت مستقیم CPU با I/O Device ارتباط دارد.

```
IN  Ax, Dx        IN  Ax, 200H
OUT Dx, Ax        OUT 10H, Ax
```

یک ۲ بایتی از دستگاه مورد نظر می‌خواند یا بر روی شماره پورت آن می‌نویسیم.

8) CMP (compare) → Ax - Bx ⇒ Flag

برای مقایسه کردن دو عدد به کار می‌رود نتیجه را درون اولین reg ذخیره نمی‌کند و فقط روی Flag تأثیر می‌گذارد. زمانی که می‌خواهیم داده ها خراب نشوند استفاده می‌کنیم

SUB Ax, Bx → نتیجه درون Ax ذخیره می‌شود

Ax ← Ax - Bx ⇒ { FLAG, Ax }

دانشگاه صنعتی امیرکبیر
اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

$Am = Bm \rightarrow$ equal

$Am < Bm \rightarrow$ Less, Below

$Am > Bm \rightarrow$ Grater, Above

این دو دستور بر روی Flag تأثیر می‌گذارد و می‌توانیم بگوییم روی همه اعضای Flag تأثیر می‌گذارد.
(... , zero flag , overflow F. , signed F.)
چون عملیات تفریق انجام می‌شود.

دستورات پرش:

۱- غیرشرطی : Jmp

1) Jmp

۲- شرطی : JXX ← براساس هر وقتی که به جای XX قرار می‌گیرد شرط مربوط روی دستور قبلی چک می‌شود.

```
start: MOV  Cx, 0          MOV  CX, 5
       Jmp  start          WHILE:
                                DEC  Cx
                                JNZ  WHILE
```

(هر بار Cx یکی از Cx کم می‌کند. (روی Flag ها تأثیر می‌گذارد.)

Cx بار تکرار می‌شود

اگر مقدار دستور قبلی صفر باشد پرش می‌کند. (هرگاه Flag مربوط صفر شود شرط برقرار شده)

```
start:
      MOV  Am, 8BBH
      MOV  BX, FFFFH
      CMP  Am, Bm
      JG   start
      JA   start
```

نمی‌توانند هر دو عدد ثابت باشند.

greater

اعداد را علامت‌دار در نظر می‌گیرد

اعداد را مثبت در نظر می‌گیرد.

Above

دانشکده مهندسی کامپیوتر مشاوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

A G
B L

اگر JG نوشته شود Jump صورت می‌گیرد ولی اگر JA باشد Jump صورت نمی‌گیرد
زیرا عدد FFFF با وجود اینکه منفی است ولی مثبت در نظر گرفته می‌شود.

JA → Jump if Above اگر اولی بزرگتر است و بدون علامت

JAE → Jump if Above equal (") اگر اولی بزرگتر و مساوی

JB → Jump if below اگر اولی کوچکتر است و بدون علامت

JBE → Jump if below equal

JC → Jump if carry = 1

JCXZ → Jump if CX = 0 (zero) اصولاً از CX به عنوان Counter استفاده می‌شود.

JE → Jump if equal

با علامت دار
* JG → Jump if greater

* JGE → Jump if greater equal

* JL → Jump if Less

* JLE → Jump if Less equal

not
JNA → Jump if not Above ≡ JBE

JNAE → Jump if not Above equal ≡ JB

JNB → Jump if not Below

JNBE → Jump if not Below equal

JNC → Jump if not carry

JNE → Jump if not equal

* JNG → Jump if not greater

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

*JNGE → Jump if not greater equal

JNLE → Jump if not less equal

JNO → Jump if not overflow

JNP → Jump if not parity = parity = 0

JNS → Jump if not signed = signed = 0

JNZ → Jump if not zero ZF = 1

overflow ↔ JO → Jump if overflow => OF = 1

parity ↔ JP → Jump if parity => PF = 1

JPE → Jump if even parity

JPO → Jump if odd parity

sign ↔ JS → Jump if signed ⇝ اگر عدد منفی است

JZ → Jump if zero

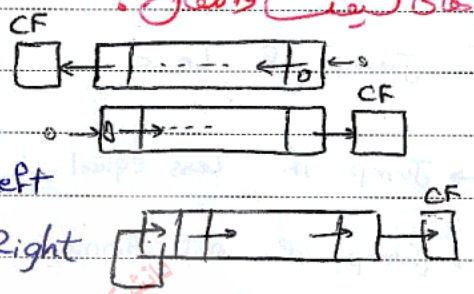
دستورالعمل های شیفت و انتقال:

SHL → shift left

SHR → shift Right

SAL → shift Arithmatic left

SAR → " " Right



از طریق shift left می‌توان مقدار sign flag را بفهمیم پس مقدار CF ، sign را آسان می‌دهد و در خانه‌ای که خالی شده صفر را قرار می‌دهیم.

ولی در دو حالت بعدی SAL و SAR برای اینکه در حالت مثبت استفاده می‌کنیم و نمی‌خواهیم ماهیت آن عوض شود پس مقدار قبلی همین خانه را در دو خانه خالی ایجاد شده قرار می‌دهیم.

shift left → ×2

shift right → ÷۲

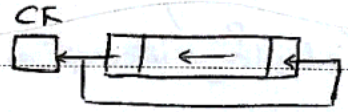
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

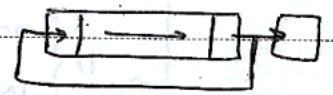
http:\\ ceit.aut.ac.ir\course

بارسی ۸۲

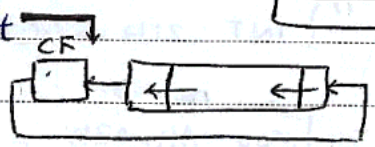
ROL → Rotate Left



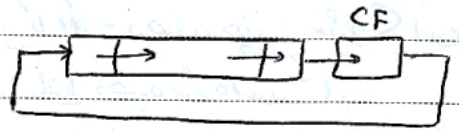
ROR → Rotate right



RCL → Rotate with carry Left



RCR → Rotate with carry right



جلوی این دستورات یک رجیستر قرار می‌دهیم که عملیات shift و Rotate را روی رجیستر مربوط انجام می‌دهیم.

در دو حالت آخر CF هم به عنوان یک بیت از رجیستر در نظر گرفته می‌شود و ۱۷ بیت Rotate می‌شوند.

	10116	CF 1
SHL	0110	1
SAR	1101	1
RCR	1101	1
RCL	0111	1

	00011011	CF=1
SHL	00110110	0
SHR	00001101	1
SAL	00110110	0
SAR	00001101	1
ROL	00110110	0
ROR	10001101	1
RCL	00110111	0
RCR	10001101	1

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

A 25 | A DB 25
B | B DB 3DUP(2)

Subject: B 25/5/94
Year. Month. Date. ( )

* تمرین:
✓ برنامه‌ای بنویسید که عدد صحیح را برعکس تبدیل کند.

- INT 21H: interrupt سیستم عامل (dos) است. هر interrupt یک شماره دارد که متناسب با آن کار خاصی را انجام می‌دهد.

1)
```
mov AH, 01H
INT 21H
```
→ منتظر یک کلید از کیبورد می‌ماند و مقدار آن را در AL ذخیره می‌کند.

2)
```
mov DL, '?'
mov AH, 02H
INT 21H
```
→ یک کاراکتر در صفحه چاپ شود که این بایت در DL ذخیره شده است.

3)
```
mov Dx, آفست پیغام
mov AH, 09H
INT 21H
```
→ آن پیغامی که offset آن در Dx قرار داده شده است چاپ می‌کند تا به $ برسد.

- LEA حافظه و مقصد

LEA Ax, Mem[10] → offset حافظه داده شده را برمی‌گرداند و seg برابر با data seg است.

LEA Ax, Mem
LEA CX, TABLE

Mov Bx, offset Mem

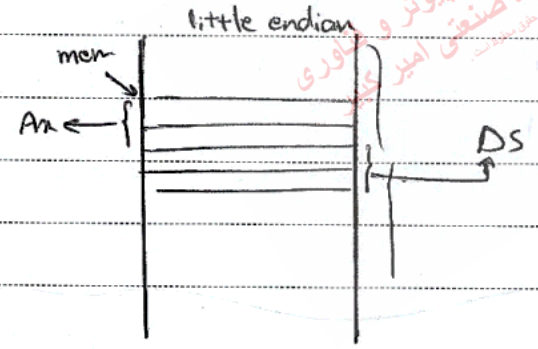
در حالت دوم تنها می‌توان از این دستورها در آرایه‌ها قابل استفاده نیست. (مثل حالت اول)

- LDS (load pointer using DS) → MOV AX, X ≡ LES Ax, X
  MOV ES, AX+2

LES

LDS Ax, Mem
(Ax: 16 بیتی، Mem: 32 بیتی)
→ چون 32 بیتی در 16 بیتی جا نمی‌شود برای اینکار 4 بایت کافی است. در Ax یا رجیستر 16 بیتی ذخیره می‌شود و 2 بایت بالا ارزش‌تر در DS (data seg) قرار می‌گیرد.

little endian



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject: - insertion sort
- merge sort
Year. Month. Date. ( )

* تمرین:
هر یک از برنامه های زیر را: زبان اسمبلی بنویسید برای آرایه ۱۰ بایتی

X DD Here
here LES AX, X → offset خود دستور در AX قرار می‌گیرد و seg درون ES ذخیره می‌شود. چون آدرس‌ها را داریم به آن pointer می‌گوییم.

load
- LAHF → مقادیر داخل ۸ flag زیر را درون AH قرار می‌دهد
SAHF → مقادیری که درون ۸ بیت AH قرار دارد را درون خانه‌های flag قرار می‌دهد.
store

	7	6	5	4	3	2	1	0
AH								

CF	PF	AF	ZF	SF
0	2	4	6	7

برای اینکه هر ۱۶ بیت را بتوانیم ذخیره کنیم از pushA و popA استفاده می‌کنیم.

- ADD Axn, Bxn → Axn ← Axn + Bxn

ADC Axn, Bxn
→ Axn ← Axn + Bxn + CF
برای اینکه carry هم ذخیره شود از دستور روبه‌رو استفاده می‌کنیم.

- SUB Axn, Bxn = CMP Axn, Bxn
→ Axn ← Axn − Bxn
در عمل منفی ممکن است overflow اتفاق بیفتد اگر اعداد علامت‌دار در نظر گرفته شود. که در پیش فرض برای OF اعداد علامت‌دار است.

SBB Axn, Bxn
Axn ← Axn − Bxn − CF

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

	OF	

$S_A S_B \bar{S}_F + \bar{S}_A \bar{S}_B S_F$ ← مواری که در بیت OF وجود دارد براساس علامت‌دار بودن اعداد است
اگر حاصل دو عدد مثبت منفی یا حاصل دو عدد منفی مثبت باشد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

- INC Ax → Ax ← Ax+1 (INC Mem →)
یکی به محتوای داخل Mem اضافه می‌شود

DEC Bx → Bx ← Bx-1

- NEG Bx → Bx ← 0 - Bx علامت Bx را عوض می‌کند که برای آن یک عمل SUB انجام می‌دهد.

نکته: برای انجام عملیات ضرب و تقسیم از جمع و تفریق متوالی در یک حلقه for استفاده می‌کنیم

اگر بیت BX صفر باشد CX را با صفر جمع می‌کنیم و اگر یک باشد CX را با AX جمع می‌کنیم. برای به دست آوردن بیت‌های BX می‌توان از shift استفاده کرد که بیت مربوط را در CF ذخیره می‌شود ولی اصل داده از بین می‌رود.

AX
* BX
CX

- XOR BX, CX : Bx ← BX ^ CX
AND DX, Mem : BX ← DX & Mem
OR AX, Mem : AX ← AX || Mem
NOT BX : BX ← $\overline{BX}$

بیت به بیت در دو رجیستر عملیات خواسته شده را انجام می‌دهد.

- Test AX, BX → همان عملیات AND را انجام می‌دهد ولی حاصل فقط تغییرات را بر روی Flag اعمال می‌کند و AX و BX تغییری نمی‌کنند.

(PF, SF, ZF, CF, OF تأثیری ندارد ولی این مقادیر را صفر می‌کند.)

راه دیگر برای صفر کردن CF استفاده از shift در یک رجیستر که مقدار آن صفر است.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

- proc هیچ ارگومانی در جلوی آن قرار نمی‌گیرد. آخرین دستور قبل از ENDP برای ret (ret) است. این دستور عمل برعکس فراخوانی proc را انجام می‌دهد اگر ret وجود نداشته باشد نمی‌داند باید کجا برگردد.

```
ENDP        FACT proc <------------ call FACT
            ENDP
```

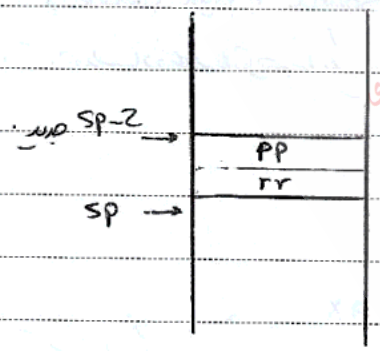
با call کردن فقط یک اشاره‌گر به proc مربوط است که به محل آن تابع jump می‌کند. حتی ممکن است proc از نوع recursive باشد و درون خودش هم call داشته باشد. برای اینکه درست عمل کند باید Ret وجود داشته باشد.

ذخیرهٔ proc های تودرتو درون stack صورت می‌گیرد در نتیجه وقتی که می‌خواهیم از proc استفاده کنیم باید stack seg را set کنیم.

برای داشتن پارامتر ورودی و یا خروجی باید خودمان استاندارد سازی کنیم و رجیسترهای خاصی را برای ورودی و خروجی فرض کنیم و قبل از صدا زدن تابع آن رجیسترها مقداردهی شوند.

- نکته call بر روی stack :

هنگامی که call صورت می‌گیرد باید آدرسی که call صورت گرفته درون stack قرار گیرد. تا بدانیم به کجا باید ret کنیم. برای ذخیرهٔ آدرس call تنها به ۲ بایت احتیاج داریم و کافی است offset آن ذخیره شود.



اطلاعات مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

+ برنامه‌ای بنویسید که تشخیص دهد عدد داشته شده اول است یا خیر

X DB 17

packed & unpacked decimal :

برای نمایش اعداد BCD به کار می‌رود که در حالت packed decimal هر رقم در ۴ بیت قرار می‌گیرد.
یعنی در یک بایت دو رقم قرار می‌گیرد ولی unpacked decimal هر رقم را در ۸ بیت ذخیره می‌کند.

packed Decimal 49 ⟶ [4 | 9] (۴ بیت، ۴ بیت)

unpacked decimal ⟶ [0 0 . . 0 0 0] [9] (۸ بیت، ۸ بیت)

- AAA (ASCii Adjustment for Add)

هنگامی که دو عدد BCD با یکدیگر جمع می‌شوند لزوماً حاصل جمع در BCD درست نیست AAA این حاصل را به BCD می‌برد (زیرا عملیات جمع در باینری درست است فقط باید به BCD تبدیل شود.)

```
MOV  AX, 65H      65    0110 0101   BCD
MOV  BX, 18H     +18    0001 1000   +
ADD  AX, BX              0111 1101
AAA                      (7  13)
                          AAA
                          8  3
```

0111 1101
+ 0110 ⟵ ۶ جمع می‌کنیم.
1000 0011
8 3

حتی اگر فرمول AAA را نداشتیم چکار می‌کنیم اگر رقم‌ها بزرگ‌تر از ۹ بود با ۶ جمع می‌کنیم در نتیجه بیان آن در رقم اول و carry آن همان ۱۰ دهدهی است که با رقم بعدی جمع می‌شود.

- MUL ⟶ اعداد بدون علامت MUL Source (byte\word)

IMUL ⟶ Signed عملیات ضرب دو عدد با توجه به اینکه در ضرب طول کلمات دو برابر می‌شود
& integer

بایت: Source x AL ⟶ AX

word: Source x AX ⟶ DX : AX ⟶

همیشه وقتی مقدار یک عملیات در رجیستر مربوطه جا نشود بقیه آن در DX ذخیره می‌شود.

چگونه دو عدد double را در هم ضرب کنیم؟

می‌توان یک عدد یک بایتی را در دو بایتی ضرب کنیم فقط باید AH را از قبل مقدارش را درست کنیم.

دانشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject: اگر باقی مانده داشته باشد در این رجیستر ها ذخیره می شود (پس باید یکی داده را در جایی نگه داریم زیرا در حالت دوم خراب می شود.)
Year. Month. Date.

- DIV source — بایت : AX ÷ source → AL (خارج قسمت) AH ← remainder
- IDIV source — کلمه : DX:AX ÷ source → AX DX ← "
  & signed

تقسیم دو عدد بر یک دیگر را انجام می دهد

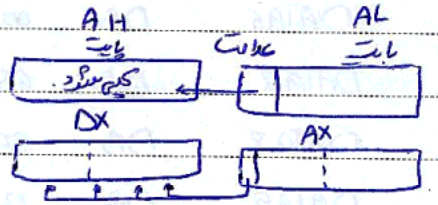
باید توجه داشت که در استفاده از این دستورات (MUL) مقادیر داخل DX خراب می شود و در IDIV در نظر گرفته می شود
در صورت عدم استفاده باید مقدار DX را صفر کنیم و در ضرب اگر از قبل مقداری در DX قرار داریم باید آن را
در جای دیگر ذخیره کنیم که خراب نشود

نکته: عمل ضرب و تقسیم در اعداد علامت دار با بدون علامت برابر است. (در واقع در عمل ضرب بیت علامت در DX ذخیره شده است.)

1 byte × 1 byte → 2 byte

1 word × 1 word → 4 byte
2 word

- CBW → convert byte to word
- CWD → convert word to double




```
MOV  AL, 5   → CBW } MOV AL, 5
MOV  BX, 100        MOV AH, 0
ADD  AX, BX
```

ممکن است درون AH از قبل مقادیری وجود داشته باشد که حاصل را اشتباه کند برای اینکه قبل از جمع کردن از CBW استفاده کنیم تا مقادیر داخل AH درست شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مثال: برنامه‌ای بنویسید و شیوه‌های آدرس‌دهی را نشان دهید.

```
stack Segment  para  Stack  'stack'
    DB   256 DUP (?)
ENDS

DATASEG  SEGMENT  para  public  'data'
    DATA1   DB   0
    DATA2   DB   20H
    DATA3   DB   30H
    DATA4   DB   0FEH
    DATA5   DB   5
    DATA6   DB   0011b
    DATA7   DB   65H
    DATA8   DB   256
    DATA9   DB   13H
    DATA10  DB   ?
ENDS

CodeSeg SEGMENT  para  public  'code'
  START  PROC  FAR
      ASSUME  CS:CODE SEG, DS:DATA Seg, ES:NOTHING, SS:STACKSEG
      MOV  AX, DATASEG
      MOV  DS, AX
      MOV  AL, DATA1
      MOV  BL, AL
```

چون در برنامه از PROC استفاده شده باید تعیین کنیم.

→ مستقیم AL = 0

→ register BL = 0

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

تمرین* همه حروف را به بزرگ تبدیل کند.

Subject:
Year. Month. Date. ( )

```
MOV   BX, offset DATA4        مستقیم
MOV   SI, 3   ⟶               یا فصل
MOV   AX, [BX][SI+2]          AX=13H
```

?	13H
AH	AL

شاخص دار با ثبات پایه

```
MOV   BX, OFFSET DATA2
MOV   AL, [BX]   غیرمستقیم با ثبات   ⟶ register indirect   AL=20H
MOV   AL, [BX+2]   ⟶ آدرس دهی نسبی به پایه ثابت   AL=0FEH
      با ثبات پایه   data4
MOV   SI, 2   یا فصل
MOV   AL, DATA3[SI]   ⟶ شاخص دار   AL=5
      data5
```

مثال: برنامه ای بنویسید که بزرگترین عدد درین داده مشخص کند.

```
Datasg   SEGMENT   para public 'data'
    MyARR   DB   5, 8, 10, 'a', 'c', 21H, 0FEH, 8, 9, 0
ENDS
Codeseg   SEGMENT   para public 'code'
    {
    MOV   DI, 0
    MOV   AL, 0
start:
    CMP   AL, MyARR[DI]
    JBE   NEXT
    MOV   AL, MyARR[DI]
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

```
NEXT:
        inc  DI
        DEC  CX
        JNE  start
```

لزوماً نیاز به CMP نداریم بلکه JMP براساس Flag ها صورت می‌گیرد که در JNE مقدار ZF چک می‌شود.

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مثال: برنامه‌ای بنویسید که حروف کوچک را به بزرگ تبدیل کند.

```
datasg  SEGMENT  para  public  'data'
        Source   DB  'my Name $'
        Dest     DB  7DUP(?),'$'
ENDS

Codesg  SEGMENT  para  public  'code'
        Assume  Cs: codesg , Ds: datasg , ES: nothing , SS: nothing

Main:   Mov  Ax, datasg
        Mov  Ds, Ax
        MOV  SI , 0
        MOV  Cx , 7

start:  Mov  AL , Source[SI]
        CMP  AL, 'a'
        JB   Next
        CMP  AL, 'z'
        JA   Next
        SBB  AL, 'a'
        ADD  AL, 'A'
        Mov  Dest[SI], AL
next:
        inc  SI
        DEC  CX
        CMP  CX , 0
        JNE  start
```

مقادیر قبل و بعد از حروف کوچک بود نیازی به تغییر نیست

a
⋮
'z'
next

$X + 'A' - 'a'$

دستور DEC بر روی ZF تاثیر می‌گذارد پس می‌توان از همه JNZ نیز استفاده کرد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

* تمرین: برنامه‌ای از نوع com بنویسید که نام خودتان را بر روی صفحه نمایش دهد.

Subject:
Year. Month. Date. ( )

* برنامه زیر را CSICC2009.aut.ac.ir

```
    MOV  AH, 09H
    LEA  DX, Dest
    INT  21H

    MOV  AX, 4C00H
    INT  21H
ENDS
END
```

مثال: برنامه‌ای بنویسید که FACT به صورت بازگشتی محاسبه کند.

```
stackseg SEGMENT para stack 'stack'
        DB  65537 (DUP)(?)
ENDS
codeseg SEGMENT para public 'code'
        :

FACT PROC
        CMP  BL, 0
        JNE  possitive
        MOV  AX, 1
        JMP  Finish
possitive:
        MOV  CL, BL
        SUB  BL, 1
        CALL FACT
        MOV  BL, CL
        MUL  AL, BL
Finish: RET
ENDP
```

یک seg برای stack در نظر می‌گیریم.

if BX = 0
FACT = 1
else
FACT = FACT(BX-1) * BX

می‌توانستیم بدون استفاده از متغیر کمکی نیز برنامه را بنویسیم زیرا
اگر چه در دون FACT BX مقدار BL خراب شود ولی دقیقاً از
call خارج می‌شود و با اضافه کردن BL به همان مقدار می‌رسیم.

```
SUB  BL, 1
Call FACT
INC  BL
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date.

debug.exe
(turbo debuger) TD.exe
(emulator 8086) Emu86.exe
(Disassembler Debuger) IDA

```
Main:   MOV   BL,5
        Call  FACT
        MOV   AX,4C00H
        INT   21H
ENDS
```

نکته: هر آدرس call درون stack اگر از نوع near باشد ۲ بایت و اگر از نوع far باشد 4 بایت (offset)
درون stack اختیار می‌کند که اول seg و بعد offset ذخیره می‌شود

مثال: برنامه زیر چه کاری انجام می‌دهد؟

```
SSEG   SEGMENT  para  stack  'stack'
       DW   32H  Dup(0)
ENDS
DSEG   SEGMENT  'Data'
       Count   EQU   05
       DATA1   DW    20060, 25000, 30000, 35000, 40000
               ORG   20H
       SUM     DW    2Dup(?)
ENDS
CSEG   SEGMENT  'code'
start  proc  FAR
       Assume  CS:CSEG, SS:SSEG, DS:DSEG, ES:nothing
       MOV   AX,DSEG
       MOV   DS,Ax

       MOV   CX,Count
       MOV   SI, OFFSET DATA1
       MOV   AX,0
       MOV   BX,AX
```

در seg (data) 32 بایت جلو می‌رود و DW می‌کند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir


```
BACK1:  ADD  AX, [SI]
        ADD  BX, [SI]
        inc  SI
        INC  SI
        DEC  CX
        JNZ  Back1
        MOV  SUM, AX
        MOV  SUM+2, BX
        MOV  AX, 4C00H
        INT  21H
START   END
ENDS
END     start
```

چون هدف آن از نوع word است و ۲ بایت است و برای رسیدن به آن بعدی دو بایت جلو می‌رویم.

⇒ Loop Back1

می‌توان آدرس را نیز با عدد جمع کنیم تا به خانه دیگری از حافظه دسترسی یابیم.

Loop : همان کار jmp را انجام می‌دهد. رجیستر خاص برای loop، cx است که به تعداد دهدهی در cx بلوک مربوط در وسط اجرا می‌شود و بعد از یک بار اجرا مقدار cx را یکی کم می‌کند.

```
        MOV  CX, 100
Back:   [          ]
        Loop Back
```

LoopE / LoopZ : در loop فقط به تعداد اختصاص داده تکرار می‌شود و هیچ شرطی را چک نمی‌کند.
در loopE علاوه بر شرط برای تمام رسیدن تکرار حلقه چک می‌کند که ZF چه مقداری دارد

loop while ZF = 1 & CX ≠ 0

```
        MOV  CX, 100
Back:   INC  SI
        CMP  0, [SI]
        LoopE  Back
```

علاوه بر شرط CX که به صفر نرسیده باشد تا زمانی که مقدار SI صفر باشد loop اجرا می‌شود.

دانشکده مهندسی کامپیوتر
اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Loop NE , loop NZ : Loop while ZF=0 , CX≠0

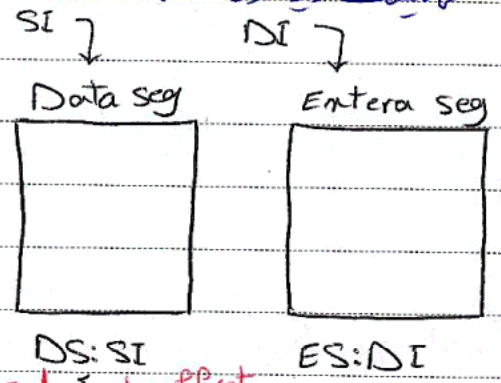
عمل برعکس دو دستور قبل را اجرا می کند.

: String Operation

مجموعه ای از بایت ها و word ها هستند که در حافظه ذخیره می شود ولی به علت استفاده زیاد آن ها در شاخه جدایی در CPU قرار می گیرد. در واقع عمل CISC را انجام می دهد.

در این حالت operand هایی که می گیرد از نوع آرایه است. در دستورات رشته ای نیاز به دو pointer دارد که برای copy کردن یا compare کردن کاراکترها را یکی یکی مقایسه و اجرا کند.

دستورات رشته ای با entra seg کار می کند و نوع خاصی نیست زیرا طول رشته ها متفاوت است و بهتر است که در seg جدا قرار گیرد.



اگر بخواهیم یک String را از DS در DS کپی کنیم باید ES را برابر DS قرار دهیم

```
push DS
pop ES      MOV AX, DS
            MOV ES, AX
```

seg ← مبدا → offset

ولی اگر عمل برعکس را انجام دهیم باید مراقب باشیم که مقدار DS را سریع عوض کنیم که دچار اشتباه نشود.

1) MOVS → MOVSB / MOVSW → word

مشخص می کند که آرایه ای که در نظر گرفتیم عناصر آن بایت یا word است و شیوه انتقال عملیات را نشان می دهد.

2) CMPS = compare string

3) SCAS = scan string

4) LODS = Load string

5) STOS = Store string

دانشگاه مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

6) INS

7) outS → { outSB, outSW

بهتر است که برای استفاده از خود کلمه های SB ، Version و SW استفاده کنیم که تایپ اعلام و مشخص شود ولی زمانی که از دستور کلی استفاده می کنیم باید حتماً operand ها را بنویسیم تا از روی آنها type مشخص شود.

example: (MOVS)

```
DataSeg
    source   DB, 'AIT'
    DEST     DB, 3DUP(?)
DataSeg ends
    CLD
    LEA  SI, Source
    LEA  DI, DEST
    mov  AX, DS
    mov  ES, AX
    mov  CX, 3
REP MOVS Dest, Source
```

باید مقادیر SI و DI که برای کپی کردن string ها به کار می روند را مشخص کرد.

در MOVS باید مقدار از ES انتخاب شود برای همین باید ES را با DS مقداردهی کنیم.

→ MOVSB

نیازی به operand نداره زیرا با B مشخص شده که از چه نوعی باید جابجا شود در غیر این صورت با نوشتن dest و source نوع انتقال مشخص می شود.

نکته: دستور MOVSB تنها یک بایت را انتقال می دهد و تفاوت آن با MOV این است که می توانیم از دستورات Rep استفاده کنیم

Rep
RepE \ RepZ
RepNE \ RepNZ

→ prefix هستند و باید قبل از instruction مربوط به string قرار بگیرند و دستور مربوطه را به تعداد CX اجرا می دهد.

دانشجویی مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

* تمرین: برنامه‌ای بنویسید که از دستورات رشته‌ای استفاده نماید و بررسی کند که آیا رشته اول در دوم وجود دارد یا خیر.

برای اینکه در هر بار Rep کردن SI و DI جلو یا عقب بروند باید DF مقداردهی شود که این کار بوسیله دستورات زیر صورت می‌گیرد:

STD (store) → DF=1 → SI و DI کم می‌شوند و عقب می‌روند

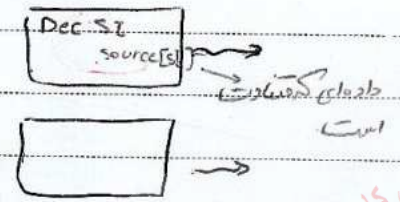
CLD (clear) → DF=0 → SI و DI اضافه می‌شوند و جلو می‌روند

example: (cmps)

```
CLD
MOV AX, DS
MOV ES, AX
LEA SI, Source
LEA DI, Dest
MOV CX, 3
RepE CMPSB
```

نمی‌توان از دستور Rep استفاده کرد زیرا فقط بایت‌ها را با هم مقایسه می‌کند پس از RepE استفاده می‌کنیم یعنی کجایی جلو برود که بایت‌ها برابر باشند و با چک کردن CX می‌توان فهمید علت بیرون آمدن تمام شدن string بوده یا برابر نبودن بایت‌ها

```
JCXZ LA
```



در یک کاراکتر با هم متفاوت هستند

دو آرایه با هم برابر بوده‌اند

نکته: همیشه بعد از مقایسه کردن SI و DI تحت هر شرایطی زیاد یا کم می‌شوند (متناسب با CLD و STD) پس برای تغییر دادن کاراکترهای متفاوت باید SI و DI را به مقدار قبلی تغییر دهیم و از طریق همین SI و DI به المان دسترسی پیدا می‌کنیم.

اگر دو رشته در المان آخر با هم متفاوت باشند مقدار CX صفر می‌شود ولی دو رشته برابر بوده‌اند در چنین شرایطی باید ZF نیز مقایسه شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شریف

www.jozveha.ir

Subject:
Year. Month. Date. ( )

چون RepE دارد ۱۰۰ بار تکرار می‌شود.

example: (SCAS)

```
CLD
LEA DI, ES:B_STRING
MOV AL, ' '
MOV CX, 100
RepE SCAS B_STRING
```

از آدرسی که DI نشان می‌دهد CX تا می‌خواند و آنقدر جلو می‌رود که AL و AX مقایسه می‌کند اگر برابر بود ZF را set می‌کند و اگر برابر نبود ZF را صفر می‌کند.

از type آن می‌فهمد یا AL باید مقایسه کند یا AX ⟶ SCASB بدون operand

example: (LODS)

```
CLD
LEA DI, ES:DEST
LEA SI, Source
MOV CX, 500
RepE CMPSB
JCXZ MATCH
DEC SI
LODS Source
:
:
match
```

مقدار موجود در آدرس SI را درون AL ذخیره می‌کند که مقدار که AL باشد یا AX از روی type، source می‌فهمد.

در LODS مقدار SI مهم است.

برای اصلاح cmp یکی SI عوض شده. ⟶ LODSB

example: (STOS)

```
CLD
LEA DI, ES:W_STRING
MOV AX, 0
MOV CX, 200
RepNE SCASW
JCXZ ALL0
SUB DI, 2
MOV CX, 6
Rep STOS W_String
ALL0: ____
```

مقدار موجود در AX یا AL را درون رشته ای که آدرس آن در DI است قرار می‌دهد.

این برنامه تا جایی جلو می‌رود که به اولین صفر برسد بعد از آن که کار الت صفر جلو دن W_String ذخیره می‌کند.

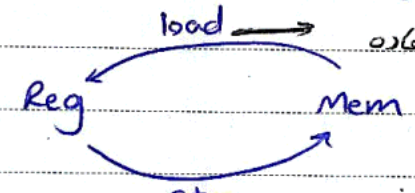
چون رشته از نوع word است پس باید ۲ تا به عقب برگردیم

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

تذکر: بسته به اینکه instruction مقدار مبدأ یا مقصد را تغییر می‌دهد از SI و DI استفاده می‌کنیم.



load → مبدأ مهم است پس از SI استفاده می‌کنیم.

store → مقصد مهم است پس از DI استفاده می‌کنیم.

example: (INS)

```
INS   destinationString , DX
MOV   DX, شماره پورت
CLD
LEA   DI, DEST
MOV   CX, 100
REP   INS   Dest, DX
```

مثال:

```
Dseg   segment
A      DB   1, 5, 8, 0BH, 'a', 5, 8, 9, 0
B      DB   8, 3, 2, 1, 8, 'b', 'z'
C      DB   9 Dup(?)
main:
mov    AX, dataSeg
mov    DS, AX
mov    AH, 0
mov    BX, 8
mov    CX, 8
SHL    AH, 1
```

دو آرایه برمبنای ۲۵۶ در نظر می‌گیرد و عناصر را دو به دو از هم کم می‌کند و درون C ذخیره می‌کند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

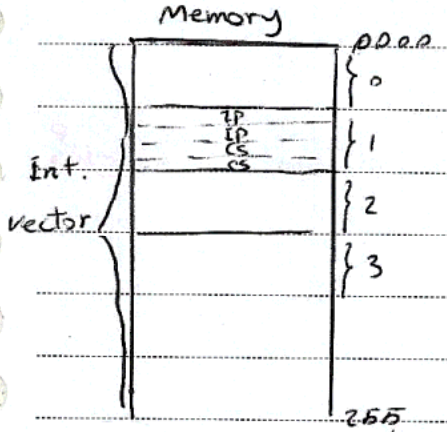
Subject:
Year. Month. Date. ( )

```
myloop:  mov   AL, [A+BX]
         sbb   AL, [BX+B]
         mov   [BX+C], AL
         Dec   BX
         loop  myloop
         mov   AX, 4C00H
         INT   21H
```

وقفه ها (interrupt):

به جدولی که در حافظه وجود دارد و مشخص می کند کدام interrupt service متناظر با interrupt هست به آن interrupt vector می گویند که متناسب با interrupt داده شده از روی جدول مربوط مشخص می دهد که کدام service باید اجرا شود.

interrupt vector در پردازنده های 286 ثابت است و در اول حافظه قرار دارد.



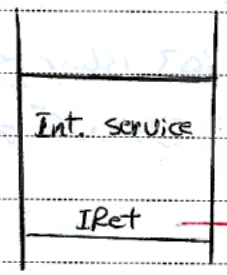
با داشتن شماره interrupt آن را در 4 ضرب می کنیم و به خانه مربوطه وقفه مربوط می رسد و با خواندن 4 بایت که شامل segment و offset است به service مربوط می رسیم.

$seg \times 16 + offset$ → آدرس interrupt service

$1KB = 2^{10} B \longrightarrow 4Byte$

$2^8 \times 2^2 = 2^{10} B$

هر کدام ۴ بایت



باید در پایان interrupt با برگشت به ادامه برنامه برگردد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

۱- interrupt ها نسبت به call اولویت بیشتری دارند و خود interrupt ها اولویت دهی دارند.
یعنی خود interrupt ها زودتر و دیرتر از بقیه اجرا می‌شوند.
از طریق datasheet مربوط به پردازنده ها می‌توان سطح خاص دارالویت شماره های interrupt چگونه است.
۲- interrupt ها هر وقت صدا زده شوند حتماً اجرا و جواب داده می‌شود و interrupt های تو در تو وجود ندارند و تا زمانی که وسط interrupt است هیچ وقفه دیگری پذیرفته نمی‌شوند.
ولی در درون خود interrupt service می‌توان یک interrupt دیگر را صدا بزنیم.

نکته: اگر درون برنامه interrupt ی داده شود و stack نداشته باشیم از stack سیستم عامل استفاده می‌کند در غیر این صورت از stack خود برنامه استفاده می‌کند.

اگر بخواهیم کاری را با interrupt service کامپیوتر استفاده کنیم. ابتدا کار خود را انجام می‌دهیم و سپس jmp به interrupt اصلی می‌دهد.

هنگام صدا زدن یک interrupt مراحل زیر اتفاق می‌افتد:
۱- پردازنده Flag ها را درون stack قرار می‌دهد.
۲- پردازنده TF و IF را غیرفعال می‌کند. (TF=0 ، IF=0)
وقفه های سخت افزار ← وقفه مربوط به سرافزار
۳- پردازنده seg دستور بعدی را در stack قرار می‌دهد.
۴- پردازنده offset دستور بعدی را در stack قرار می‌دهد.
۵- شماره Int. x 4 و محل سرویس وقفه مشخص می‌شود.
۶- دو بایت اول ← آفست دستوری که باید اجرا می‌شود.
۷- دو بایت بعدی ← سگمنت دستوری که باید اجرا می‌شود.

نکته: اگر وسط اجرای یک دستور interrupt داده شود ابتدا دستور را تا آخر اجرا می‌کند و سپس وقفه داده می‌شود. پس بهتر است IP را پس از اجرای آن دستور درون stack قرار می‌دهیم زیرا ممکن است این دستور jmp بوده باشد و IP دقیقاً IP+1 نیست.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

pop کردن این مقادیر از درون stack توسط IRet انجام می‌شود و این خود یکی از تفاوت‌های Ret و IRet است.

IRet:

۱- آفست دستورهای را از stack برمی‌دارد.

۲- سگمنت دستورهای را از stack برمی‌دارد.

۳- Flag را به مقدار اصلی بازگرداند (از طریق stack)

نکته: interrupt‌ها باید source code کم و سرعت بالا داشته باشند.

- STC → set carry flag
CLC → clear carry flag
CMC → complement carry flag (not)

مربوط به carry Flag هستند

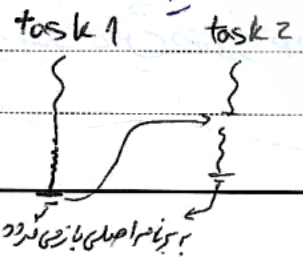
- STI
CLI

interrupt Flag را set و clear می‌کند زمانی که interrupt غیرفعال می‌شود یعنی mask شده‌اند که به این عمل Interrupt masking می‌گوییم.
اگر یک interrupt از نوع maskable باشد یعنی می‌توانیم جلوی آن را بگیریم.

Interrupt masking

نتیجه: تنها می‌توان مقادیر CF، IF و DF را عوض کرد.

یکی از روش‌های multitasking تقسیم کردن زمان‌ها بین چندین برنامه است که در اینجا دقتی که یک task به انتها می‌رسد و IRet می‌شود آدرس نقطه بازگشت را در جای دیگری داریم و آدرس interrupt بعدی را به جای آن در stack قرار دهیم و سپس به اصل اصلی برنامه بازمی‌گردیم.
به این شیوه می‌توان multitasking را اجرا کرد.



www.jozveha.ir

Subject:
Year. Month. Date. ( )

MACRO :

proc در زمان runtime مشخص می‌شود که چه کاری باید انجام دهد اما macro در designtime مشخص می‌شود از macro در زمان خلاصه نویسی استفاده می‌شود

تفاوت MACRO و proc :

۱- ماکرو مربوط به اسمبلر است و عیناً عمل کپی را انجام می‌دهد. یعنی ماکروها code generator هستند یعنی خودشان source را بیشتر می‌کنند.

۲- سایز برنامه را زیاد می‌کند.

۳- با stack کاری ندارد.

۴- نه call و نه Ret دارد.

۵- سرعت بالا

۶- می‌تواند آرگومان بگیرد.

بدی‌های MACRO :

۱- کد برنامه زیاد می‌شود.

۲- نسبتاً پیچیده است (بازگشتی)

در MACRO قبل از اینکه برنامه اجرا شود باید recursive را انجام دهد و کد را ایجاد کند.

۳- مقداری باید در زمان اسمبل شدن مشخص شود. یعنی نمی‌توان یک مقدار را در طول اجرا مشخص شود و macro با آن کار کند. (برای بازگشتی) در واقع مقادیری که به عنوان آرگومان به ماکرو داده می‌شود باید imp باشد نه حسیت یا خانه‌ای از حافظه.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

ماکروها برای اسمبلر خوب نیستند زیرا باید دستورات زیادی را ترجمه و جایگزین ماکرو کنند ولی این کار یک بار انجام می‌شود و با ساخته شدن exe دیگر مشکلی وجود ندارد.

Subject:
Year. Month. Date. ( )

نکته: تعریف MACRO هیچ سازگاری از فایل exe را شامل نمی‌شود. (مخصوصاً هنگام پیاده‌سازی خود ماکرو است.)

```
swap  MACRO  From , To
      mov  AX , From
      mov  From , To
      mov  To , AX
ENDM
```

From و To نیاز به تعریف شدن ندارند و تنها سمبل هایی هستند که به کار رفته اند و اصلاً From و To را نمی‌بینند و هر بار macro صدا زده شده باشد با این دستورات جایگزین می‌کند.

مثال:

```
myAdd  MACRO  value1 , value2 , Sum
       mov  AX , value1      ---> { push AX
                                    push BX
       mov  BX , value2
       ADD  AX , BX
       mov  Sum , AX         ---> { pop BX
                                    pop AX
ENDM
```

در برنامه از رجیستر ها استفاده کردیم که اگر ورودی های ماکرو عدد ثابت بودند مشکلی ایجاد نشود.

```
myAdd  1,5 , num
```

مثال: فاکتوریل یک عدد را به صورت بازگشتی با MACRO بنویسید.

```
FACT  MACRO  n
IFE  n
     F = 1
ELSE
     FACT  n-1
     F = F * n
ENDF
ENDM
```

انجام محاسبات مربوط به فاکتوریل توسط اسمبلر اجرا می‌شود نه پردازنده

چون برای F از اعمال ضرب و مساوی استفاده کردیم باید از نوع dir. باشد و همچنین چون مقدار آن تغییر می‌کند نمی‌توان از EQ استفاده کرد.

باید به این صورت F = 1 در برنامه تعریف شود.

دانشکده مهندسی کامپیوتر و مشاوری اطلاعات دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

اگر ماکروها بازگشتی نباشند می‌توانیم از قید نیز در آن استفاده کنیم ولی در صورت بازگشتی بودن چنین چیزی غیرممکن است.

تعدادی از directive ها که برای صحبت با اسمبلر به کار می‌رود به صورت زیر است. یعنی دستور نیستند که پردازنده آنها را اجرا کند فقط برای مشخص کردن وضعیت برای اسمبلر و قبل از اجرا به کار می‌رود

- IF
  → هرگاه شرط جلوی IF یک باشد دستورات مربوط به آن اجرا می‌شود.
- IFE
  → هرگاه عبارت روبروی آن برابر یک باشد دستورات بعد از آن اجرا می‌شود.
- ELSE
  → می‌توانیم همراه با if از دستور else نیز استفاده کنیم.
- ENDF
  انتهای if و else با ENDF مشخص می‌شود. یعنی برای شرط مربوطه تا رسیدن به endif اجرا می‌شود.
- IFDEF
  اگر آرگومان قبلاً تعریف شده باشد دستورات اجرا می‌شود. یعنی آرگومان DB یا EQ شده باشد.
- IFDIF
  اگر آرگومان های جلوی IF با یکدیگر متفاوت باشند اجرا می‌شود در غیر این صورت اجرا نمی‌شود.
- IF1
  اگر در path اول کامپایل است دستورات را اجرا کند (خواندن اولیه برنامه و جایگزین کردن dir با کد)
- IF2
  اگر در path دوم کامپایل باشد دستورات را اجرا می‌کند (زمان ترجمه دستورات و تبدیل به صفر و یک)

```
IF 0
    mov AX, BX
ENDF
```

هیچ وقت دستور mov اجرا نمی‌شود.

نکته: عملیات ضرب و تقسیم برای اسمبلر قابل انجام است ولی در انجام instruction ها نمی‌توان از این عملیات استفاده کنیم. تنها برای اعدادی که از نوع EQ تعریف شده‌اند یعنی directive هستند می‌توان استفاده کرد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject: $\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{n}$

Year. Month. Date. ( )

* برنامه بازگشتی به دو روش زیر بنویسید

- الف) با استفاده از PDL - ب) با استفاده از MACRO

```
A EQ 5
C EQ 6
D EQ A+C/2
mov AX, C
IF A
  MOV AX, 5
ELSE
  MOV BX, 5
  MOV AX, 6
ENDIF
```

ست به اینکه مقدار A چه باشد source code متفاوتی را ایجاد و اجرا می‌کند.

مثال: یک time consumer که زمان را از بین می‌برد.

```
Wait MACRO count
  push CX
    MOV CX, count
Next: Loop Next
    pop CX
ENDM
```

Local Next

این برنامه فقط به تعداد count یک حلقه تکرار می‌شود و زمان از بین می‌رود.

اگر این ماکرو در دو قسمت از برنامه استفاده کنیم، دو تا label هم نام داریم و اسمبلر خطا می‌گیرد. برای رفع این مشکل از دستور Local Next استفاده می‌کنیم که خود اسمبلر شماره label ها را عوض می‌کند.

- IRP

دستورات بین IRP و ENDM را برای آرگومان‌هایی که می‌گیرد تکرار می‌کند.

```
IRP dummy, <argoman_List>
```

- REPT

محتوای داخل REPT و ENDM به تعداد جلوی REPT اجرا می‌کند. باید ENDM را هم برای ماکرو و هم برای REPT بنویسیم.

- EXITM

هر جایی که این directive به کار رود بلافاصله از ماکرو خارج می‌شود.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

_IF1 یعنی در path اول عملیات انجام شود (زمانی خواندن برنامه و تعریف directive ها)

_IF2 در path دوم دستورات مربوط اجرا شود.

_IFB (blank) مشخص می‌کند آیا به ماکرو آرگومانی داده شده است در صورت اینکه آرگومانی داده نشود وارد حلقه می‌شود.

_IFNB

EXAMPLE 8 (IFB)

```
WAIT  MACRO  count
    Local  Next
    IFB  count        ---> if count is blank
        ExitM
    ENDIF
    push  cx
    MOV  cx, COUNT
NEXT:  Loop  NEXT
        pop  cx
ENDM
```

نکته: جلوی دستورهای directive مربوط به ماکرو فقط expression باید بنویسیم و نمی‌توانیم از رجیستر یا حافظه استفاده کنیم. از این directive ها فقط در ماکرو می‌توان استفاده کرد.

```
CELL  MACRO  N, X
    REPT  N
    DB  X
    ENDM
ENDM
```

هر جا cell صدا زده شود یک آرایه N تایی از X ایجاد می‌کند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

EXAMPLE: (REPT)

```
MYMUL   MACRO   X,Y
    MOV   AX,0
    REPT  X
    ADD   AX,Y
    ENDM
ENDM
```

برنامه ای با REPT که عمل جمع را انجام می دهد.

به تعداد X ، AX را با Y جمع می کند

EXAMPLE: (IRP)

```
IRP   X,<1,2,3,5,7>
    DW  X*X*X
ENDM
```

→

```
DW   1*1*1
DW   2*2*2
DW   3*3*3
DW   5*5*5
DW   7*7*7
```

نکته: برای دسترسی به DW هایی که صورت می گیرد از آدرس دستور قبلی آن می توان استفاده کرد.

```
A DB 5
IRP  X,<1,2,3>
```

→ A+1 آدرس آن

⋮

- IRPC

از رشته ی جلوی آن کاراکتر به کاراکتر می خواند و در X قرار می دهد و دستورات مربوط تا ENDM را برای آن اجرا می کند.

```
IRPC  X,1234
    DB  X
ENDM
```

'1'	'2'	'3'	'4'

در این دستورات تکرار شونده نمی توانیم برای X اسمی انتخاب کنیم (DB)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

8 DEF_TABLE MACRO N, value

TABLE & N DB N DUP (value)

ENDM

عمل & دو سمبل TABLE و N را به هم وصل می‌کند و یک label می‌سازد.

EXM: IRPC X,1235 7

myByte&X DB X

ENDM

mydata1 mydata

'1'	'2'	'3'	'5'

EXM: DEF_TABLE 10,5 ⇝ TABLE10 DB 10 DUP(5)

مثال: سری فیبوناچی را به صورت بازگشتی بنویسید.

$Fib_n = Fib_{n-1} + Fib_{n-2}$

```
FIBO MACRO N
  Local F1
  IFE N-1
    F=1
  ELSE
    IFE N-2
    F=1
    ELSE
      FIBO N-2
      F1=F
      FIBO N-1
      F=F1+F2
    ENDIF
  ENDIF
```

چون دو مرحله پشت سر هم صدا زدن ماکرو مقدار F1 به هم می‌ریزد از local استفاده می‌کنیم تا به طور موقتی فرض اسم آن را عوض کند.

نتیجه‌ی نهایی در F قرار دارد

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

از میان directive های موجود برای اسمبلر تنها دستورات repeat مخصوص ماکرو هستند.

Subject:
Year. Month. Date. ( )

مثال: ضرب اعداد ۳۲ بیتی در ۳۲ بیتی

16	16
A	B

$= B + A \times 2^{16}$

16	16
C	D

$\times = D + C \times 2^{16}$

$\Rightarrow BD + (AD + BC) \times 2^{16} + AC \times 2^{32}$

BD
AD
BC
AC

```
Num1  DD  15
Num2  DD  16

Lea  BP, Num1
Mov  AX, [BP]
~~Mov BX, [BP+2]~~
Lea  BP, Num2
Mov  BX, [BP]
ADD  A2, AX
~~Mov DX, [BP+2]~~
ADC  A3, 0
MUL  BX
ADD  A3, DX
ADC  A4, 0
Mov  AX, A
Mov  BX, C
MUL  BX
ADD  A3, AX
ADC  A4
ADD  A4, DX
```

```
Mov  AX, D
Mov  BX, B
MUL  BX
Mov  AI, AX
Mov  A2, DX
Mov  AX, A
Mov  BX, D
MUL  BX
Mov  A3, DX
ADD  A2, AX
ADC  A3, 0
Mov  AX, B
Mov  BX, C
MUL  BX
```

نتیجه

A4	A3	A2	A1

```
A4  DW 0
A3  DW 0
A2  DW 0
A1  DW 0
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مثال: محاسبه $y=\sqrt{x}$ از طریق اسمبلی

$y_2 = (x/y_1 + y_1)/2$ ← y_1 یک مقدار حدس زدنی

روال تا زمانی پیش می‌رود که $2 \geq |y_{i+1} - y_i|$

```
est proc
    mov temp, DX
    mov AX, X
    DIV CX
    mov temp2, AL
    mov temp3, word ptr temp2
    Add temp3, CX
    mov AX, temp3
    mov BL, 2
    DIV BL
    mov temp2, BL
    mov AX, word ptr temp2
endp
main:
continue: INC DX
          INC DX
          CMP AX, DX
          JBE -break
          Call est
          Jmp continue;
-break:
```

```
SUM AX, BX
CMP AX, BX 2
SUB AX, BX
JBE A
MOV
CMP AX, 2
JBE Finish
Div
```

DX ← y_1
X ← x
CX ← y_1

```
temp3 DW
temp2 DB
temp1 DW
```

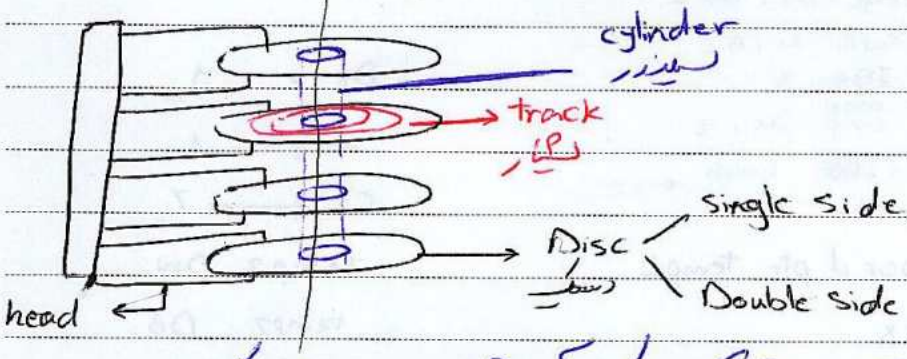
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

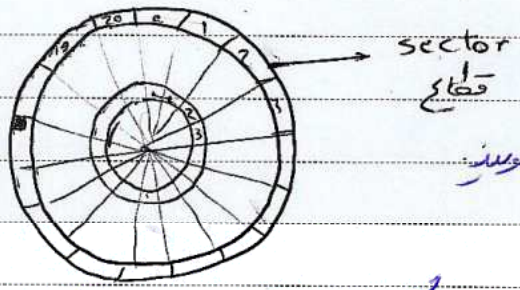
Subject:
Year. Month. Date. ( )

ساختار درونی hard disk :

مجموعه‌ای از صفحات که به آنها disc می‌گوییم حافظه را تشکیل می‌دهند که بر روی هر disc یک head وجود دارد که عمل خواندن از روی حافظه را به صورت مغناطیسی انجام می‌دهند که بوسیله یک پایه که می‌تواند متحرک یا ثابت باشد بر روی دیسک قرار گرفته است.



هر دیسک به دوایر هم مرکزی تقسیم می‌شود که به هر یک از آن شیارها track می‌گویند به مجموعه‌ای از شیارها با شماره یکسان یک سیلندر (cylinder) را ایجاد می‌کند.



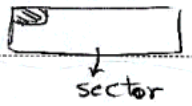
به قسمت کوچکتر از یک track، قطاع می‌گویند.

براساس هدف مربوط به سرعت و حافظه هر حافظه نمی‌توان گفت hard disc ها کم سرعت‌ترین memory هستند زیرا باید ابتدا disc و سپس track مورد نظر و بعد از آن به sector مربوط می‌رود. برای افزایش سرعت حافظه دیسک‌ها از یک بافر استفاده می‌کنند و sector خوانده شده را درون آن قرار می‌دهیم تا دفعات دسترسی از بافر بخوانیم.

(Side , cylinder , sector)

در هر عمل خواندن از درون حافظه ابتدا مشخص می‌شود کدام head خوانده شود و سپس متناسب با سیلندر مورد نظر head جلو و عقب می‌رود و سپس با چرخیدن دیسک sector مورد نظر زیر head قرار می‌گیرد.

byffer



فایل چند بایت از یک sector احتیاج داریم ولی با پیش بینی نیاز به سایر داده‌ها کل sector را در بافر قرار می‌دهد.

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

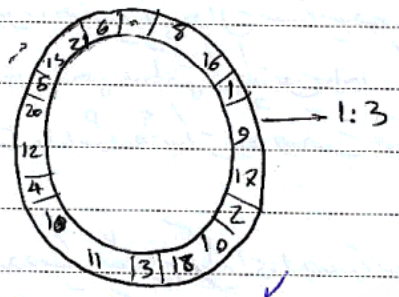
در هارد دیسک ها تعداد قطاع ها در track های مختلف با یکدیگر برابر نیست و یکسان است
در این روش میزان دوران head برای همهٔ track ها یکسان است.
برای اینکه حجم یکسانی در sector های track های مختلف قرار بگیرند باید track های داخلی فشرده تر شوند تا بتوانند تعداد یکسانی داشته باشند. ← Sectors per track

به دلیل حرکت فیزیکی hard head ممکن است بعد از فرستادن دستور به دلیل ایندیس دیرتر از زمان ممکن توقف اتفاق بیفتد و از دادهٔ مورد نظر برای خواندن در مرحلهٔ بعدی رد می شود.
برای برطرف کردن این کار می توان از روش های زیر استفاده کرد:

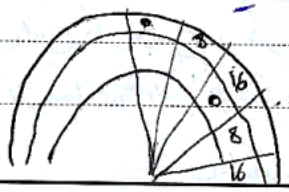
۱- interleaving :
ترتیب قرارگرفتن اطلاعات بر روی هارد پراکنده باشد. در واقع قطعات مختلف پشت سر هم می آیند ولی پراکنده بین هر دو قسمت پشت سر هم چندین خانه فاصله دارد.

1:n
↳ هر n تایی را قرار دهیم

کمترین فاصله برای قرارگرفتن 1:6 است.



۲- Track Skew :
داده ها را بر روی track های مختلف یک disk قرار می دهیم ولی با فاصله ای از یکدیگر.



در این روش اگر مسافتی هم رد شود قطاع بعدی یک خانه فاصله دارد.

۳- divice driver : اولین چیزی که بر روی هارد دیسک ها قرار می گیرند divice driver ها هستند.

Application
OS
Divice driver
HW

دانشکده مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

۳- Cylinder skew :

همان شیوه قرارگیری قبلی است با این تفاوت که هدهای مختلف بر روی سیلندرهای گوناگون قرار دارد.
توجه داشته باشید که head ها با یکدیگر حرکت می‌کنند و در یک هارد با یکدیگر می‌چرخند.

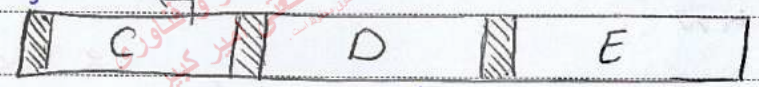
این سه روش باید موازی با یکدیگر کار کنند اگر داده بر روی یک track (sector) قرار بگیرد از روش اول اگر بر روی یک track جا بگیرد بر روی track های مختلف و روش دوم و اگر بر روی track های مختلف ها بگیرد از روش سوم و سیلندرهای مختلف استفاده می‌شود.

File System :

اولین فایل سیستم موجود با نام FAT بود که مخفف file allocation table است ولی به دلیل محدودیت‌ها موجود از NTFS استفاده می‌کنند.
آدرس دهی بر روی حافظه به دو صورت است ۱- آدرس دهی فیزیکی ۲- آدرس دهی منطقی
۱- آدرس دهی فیزیکی همان سه تایی مرتب (side, cylinder, sector) است که مستقیماً با حافظه حرف می‌زند.
۲- در آدرس دهی منطقی کوچکترین واحد سکتور است ولی این سکتورها به صورت پی در پی شماره گذاری می‌شوند بیشتر ما با شماره سکتورها صحبت می‌کنیم و با این که این سکتور در کجای حافظه قرار دارد کاری نداریم.

تمام هارد دیسک شامل آرایه‌ای از صفحات بی‌نهایت است که چگونگی تعریف یک نگاشت برای قرار دادن این ساختار درختی که محتوای آن فایل است بر روی این نوار آرایه‌ای شامل از وظایف فایل سیستم است.
کل هارد دیسک به صورت یک نوار یا آرایه دیده می‌شود که به چند قسمت تقسیم می‌شود و هر قسمت آن partition یا volume یا drive می‌گویند که به ترتیب نام گذاری می‌شوند و حروف A و E در آنها نوشته شده است.

partition, volume, drive



برای دسترسی به هر پارتیشن باید شماره سکتور شروع شوند و پایانی و همچنین سایز partition را داشته باشیم. برای این کار اولین سکتور در هر پارتیشن را نرم می‌کنیم تا اطلاعات دو دیسک را برای فایل دارند. که به این سکتورها boot sector می‌گویند.

سایز هر سکتور 512k یا 1024k است.

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

و برای این به آن boot sector می‌گویند که در آن‌ها اطلاعات مورد نیاز برای بوت شدن وجود دارد. در واقع در هر پارتیشن می‌تواند یک برنامه بزرگتر به اسم سیستم عامل وجود داشته باشد که ارتباط ما با کامپیوتر را manage می‌کند.

چون در درون boot sector اطلاعاتی در مورد bootable بودن آن پارتیشن دارد این نام را گرفته است در واقع این سکتور حساس‌تر است که از روی آن می‌توان فهمید که اطلاعات معتبر هست یا نه هر پارتیشن اگر OS داشته باشد bootable است.

آیا محل فیزیکی boot sector ثابت است یا نه؟

boot sector فقط بر روی هارد ذخیره می‌شود. از نظر فیزیکی ثابت نیست ولی از نظر logic ثابت است و با تغییر محل پارتیشن‌ها محل اولین سکتور از آن نیز متفاوت است (اولین خانه از پارتیشن است)

: bad sector

هر سکتور قابلیت ذخیره اطلاعات یا برداشتن اطلاعات از روی آن‌ها را دارند. اگر یک سکتور به دلیل خاصیت آهنربایی یا ضربه این دو گی خود را از دست بدهد به آن bad sector می‌گوییم و با داشتن شماره سکتورهای خراب می‌توان موقعیت جغرافیایی آن‌ها را ذخیره کرد.

چون سکتور کوچکترین واحد اندازه‌گیری در هارد است در نتیجه اگر تنها یک بیت از آن خاصیت ذخیره یا بازیابی خود را از دست بدهد کل سکتور از بین رفته است. پس هرچه سایز sector ها کوچک‌تر باشد حایز bad sectors نیز کم‌تر می‌شود.

: partition Sector

چند سکتور اول هارد دیسک قبل از قرار گرفتن partition ها رزرو شده است تا اطلاعاتی در مورد تعداد partition های موجود و محل فیزیکی قرار گرفتن آن‌ها و نقطه شروع و پایان آن‌ها را در خود نگه دارند.

(0 و 0 و 1) ⟶ آدرس فیزیکی partition sector

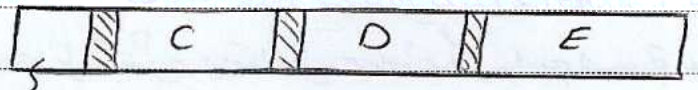
cylinder — side (head) — sector

از آنجایی که محل فیزیکی partition sector ثابت است اگر به bad sector تبدیل شود دیگر هارد قابل استفاده نیست.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شریف

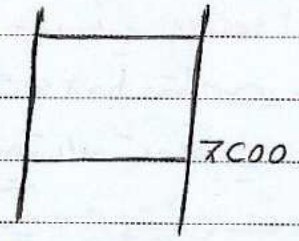
www.jozveha.ir

Subject:
Year. Month. Date. ( )

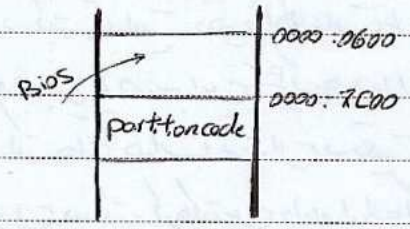


اولین سکتور هارد (partition sector)

با روشن کردن کامپیوتر در ابتدا اطلاعات موجود در پارتیشن سکتور درون حافظه قرار می گیرد و سپس به اجرا شدن می کند. در واقع ترابیتی ای به اسم BIOS در آن وجود دارد که حاوی برنامه ای است که با روشن شدن کامپیوتر درون حافظه قرار می گیرد و اطلاعات موجود در پارتیشن سکتور را به حافظه RAM می آورد.



BIOS در طول اجرا شدن partition code را می خواند و در جای خود overwrite می کند و خودش به آدرس 0600 از حافظه می رود.



ساختار partition sector به صورت زیر است.

Address	contents (30 bytes)
+000h	partition code
1BEh	1_entry in the partition table
1CEh	2_entry in the partition table
1DEh	3_entry in the partition table
1EEh	4_entry in the partition table
1FEh	ID code (AA55h)

صفحه بعد

در ابتدا خوانده می شود که اطلاعاتی در مورد تعداد پارتیشن ها نیز در آن وجود دارد و دستورهای بعدی را ترجمه می کند.

تعداد partition های موجود را نشان می دهد که max ۴ تا است و سپس اینکه کدام ها فعال اند در par. code مشخص می شود و اطلاعات هر پارتیشن را دارد.

بعد از partition C. به این جا می رود که در صورت برابر بودن محتوا با AA55 مشخص می شود که محتوای داخل هارد معتبر است و partition وجود دارد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شاهرود

PAPCO

www.jozveha.ir

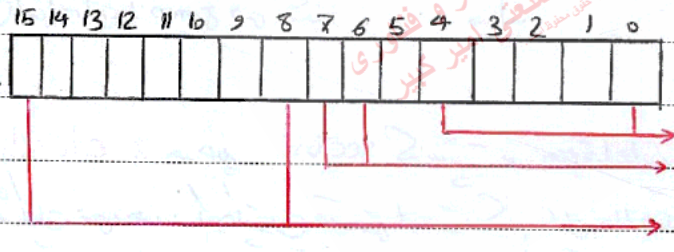
Subject:
Year. Month. Date. ( )

نکته: در صورت نیاز به پارتیشن های بیشتر از extended استفاده می کنیم که در واقع هر قسمت از پارتیشن خود یک پارتیشن دیگر است که آن نیز برای خود یک partition table دارد.
درون هر پارتیشن تنها یک سیستم عامل می تواند وجود داشته باشد ولی یک سیستم عامل می تواند پارتیشن های گوناگونی را شامل شود.

Entry partition table

contents (16 bytes)	
partition status 00h = inactive = non bootable 80h = boot partition = bootable	تشخیص bootable بودن پارتیشن را مشخص می کند
head which the partition begins	نقطه شروع پارتیشن (head, cylinder, sector) را مشخص می کند.
sector and cylinder " "	
partition type 00h = Entry not allocated (سیستم عامل نیست) 01h = Dos with 12 bit FAT 02h = XENIX 03h = XENIX 04h = Dos with 16-bit-FAT 05h = extended Dos 06h = Dos_4 0Bh = concurrent Dos	اگر پارتیشن boot شونده است از چه نوع سیستم عامل استفاده می کند که هر شماره نشان دهنده نوعی از OS است که می تواند ۲۵۶ حالت مختلف داشته باشد
head which the partition ends	سه تایی دوم نقطه پایانی partition را نشان می دهد.
sector and cylinder " "	
removal ...	اطلاعات اضافی
Number of sectors in the partition	تعداد کل سکتورهای هر پارتیشن را نشان می دهد

1 word



شیوه قرارگرفتن شماره sector و cylinder نقطه شروع و پایان درون ۲ بایت به صورت بالا است که در ابتدا قرار بوده بایت اول برای sector و بایت دوم برای cylinder باشد ولی با کمبود جا برای سیلندر دو بیت از سکتور به آن اختصاص داده شده است.

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

پس دستیابی به اصلی partition code از طریق partition table به اطلاعاتی در مورد فرمت کلی پارتیشن های دسترسی یافتیم. پس با خواندن اطلاعات موجود در boot sector از هر partition که آدرس آن همان نقطه شروع پارتیشن است که در درون partition table مشخص شده علاوه بر اطلاعات کلی در مورد هر پارتیشن، سیستم عامل مربوطه نیز اندوی آن boot می شود.

ساختار یک boot sector به صورت زیر است:

boot sector

contents	
Jump to boot routine (E9xx یا EBxx90)	→ opcode jump
Manufacturer's name & version number	→ نام سازنده هارد و version آن
Bytes per sector	→ هر sector چند بایت است.
Sectors per cluster	→ تعداد sector ها در هر cluster
Number of reserved sector	
Number of FATS	→ تعداد backup از FAT در partition
Number of entries in root directory	
Number of sectors in volume	→ تعداد کل sector های موجود در پارتیشن
Media descriptor	
Number of sectors per FAT	→ هر FAT چند sector را به خود اختصاص می دهد
Sectors per track	→ تعداد sector ها در هر track
Number of read/write heads	
Number of hidden sectors	
BOOT ROUTINE	→ در همان ابتدا به اینجا jump می کند.

cluster: هر چند sector که پشت سر هم قرار دارند (از نظر فیزیکی و هم از نظر منطقی) یک cluster را تشکیل می دهند. کوچکترین قسمتی است که به فایل ها اختصاص یافته است.

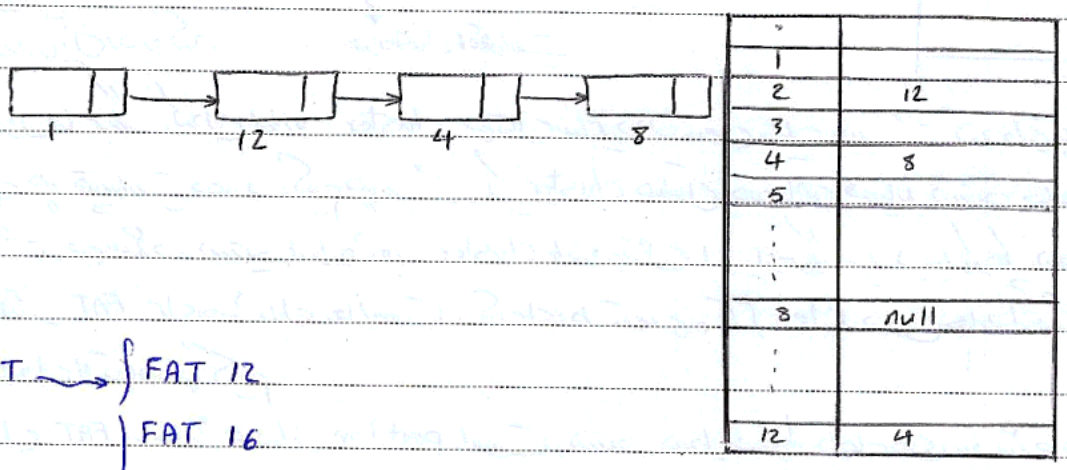
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

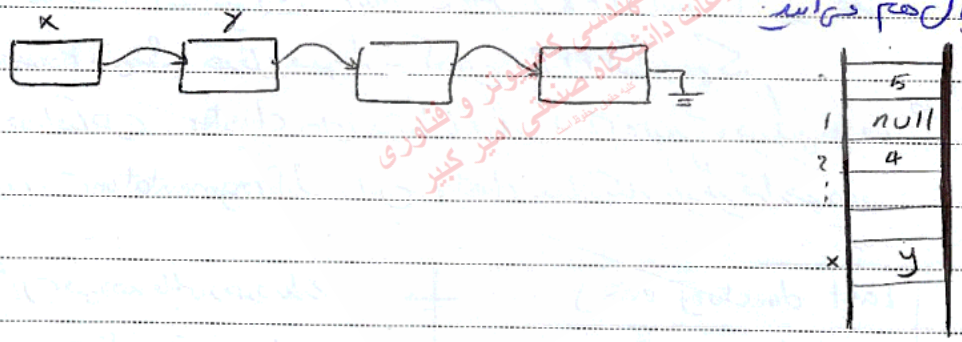
FAT:

در نمایش link list ها می‌توان از یک آرایه استفاده کرد که در هر خانه شماره گره بعدی نوشته شده باشد. این ایده‌ای است که در FAT به کار رفته و در فایل سیستم‌ها از آن استفاده کرده‌اند.
در واقع کوچکترین واحد را cluster در نظر گرفته و هر cluster خانه بعدی خود را نشان می‌دهد.
در نتیجه در این روش لزومی به کنار هم بودن خانه‌ها وجود ندارد.



FAT → FAT 12
FAT 16
FAT 32

تعداد بیت اختصاص یافته به cluster ها در boot sector مشخص می‌شود که فایل‌ها حاوی cluster های گوناگونی هستند که به صورت link list به یکدیگر متصل شده‌اند.
برای ذخیره link list از جدولی استفاده می‌کنند که به آن FAT می‌گویند. در واقع آرایه‌ای است که از صفر شروع می‌شود و در هر خانه نام گره بعدی نوشته می‌شود. در نتیجه با داشتن آدرس اولین cluster بقیه آنها نیز به دنبال هم می‌آیند.

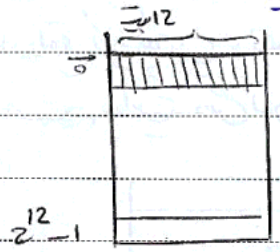


دانشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

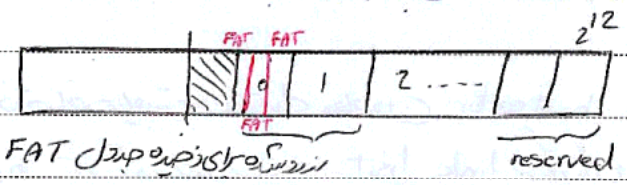
Subject:
Year. Month. Date. ( )

براساس قدرت FAT به دسته های گوناگونی تبدیل می شود. مثلاً در FAT 12 هر خانه از جدول شامل ۱۲ بیت است و دامنهٔ اعدادی که می تواند در هر خانه قرار گیرد از ۰ تا $2^{12}-1$ است.
در چنین شرایطی برای سایز فایل محدودیت وجود دارد و تعداد کل cluster هایی که می توان استفاده کرد 2^{12} است.



$2^{12} \simeq 4k \simeq 4000$ cluster $\simeq 1kB$ ⇒ در مجموع می توان 4mb فایل بر روی آن ذخیره کرد.
هر cluster ۱ کیلوبایت

می توان با بزرگ کردن اندازهٔ cluster می توان سایز در دسترس فایل را بیشتر کرد ولی چون برای هر فایل چه چندبایت مصرف کنیم چه بیشتر یک cluster حداقل اختصاص می یابد در نتیجه حافظه هدر رفته بیشتر می شود. در نتیجه اندازهٔ هر cluster را نه می توان بسیار بزرگ یا بسیار کوچک در نظر گرفت.
ساختار FAT برای همهٔ فایل ها است. که برای دسترسی به تمام File درون هارد از شماره های موجود در جدول استفاده می کنیم.
جدول FAT در خانهٔ صفر از partition است. در نتیجه جدول مربوط در خانه های رزرو شده صفر و یک است.



حجم جدول FAT $= \left\lceil 2^{12} \times 12 / 8 \right\rceil$ بایت

در هنگام قرار گرفتن جدول در حافظه به صورت پی در پی در حالتی که وجود دارد درون هارد ذخیره می شود.
به دلیل حساس بودن جدول FAT چند کپی از آن در cluster شمارهٔ صفر قرار می گیرد
دو cluster آخر نیز برای null که برابر (FF8) است رزرو شده است. زمانی که جدول مربوط Format می شود جدول مربوط به آدرس FF8 اشاره می کند.
با دور افتادن cluster های مرتبط با یکدیگر انجام عملیات فیزیکی هارد بیشتر می شود. در نتیجه از طریق عملیات difregmentation، این cluster ها را در کنار یکدیگر قرار می دهد.

کد مربوط به بایت اول از ۳۲ بایت مربوط به هر فایل

00 H	Last directory entry	→ نشان می دهد فایلی وجود ندارد
05H	First charecter of filenam has E5H	→ فایلی با این مشخصات وجود دارد ولی حذف شده و امکان بازیابی وجود دارد.
2EH	File applies to current directory	
E5H	File deleted →	→ فایل delete شده.

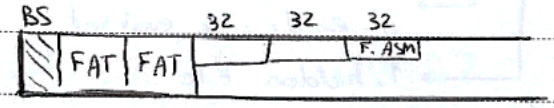
www.jozveha.ir

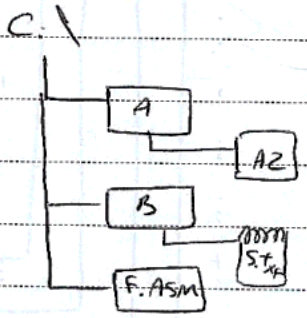
Subject:
Year. Month. Date. ( )

هنگام delete کردن یک فایل تنها آدرس اولین cluster را به null تبدیل می‌کند و نتیجه link list هم خیال وجود دارد.

شیوه قرار گرفتن file ها در hard:

درون هارد بعد از خواندن جدول های FAT در ابتدا فایل ها یا فولدرهایی که در root وجود دارند در خانه های ۳۲ بایتی قرار می‌گیرند که هر بایت نشان دهنده یکی از مشخصات فایل است.





با توجه به جدول زیر برای نام فایل ۸ بایت اختصاص داده است حال اگر اسم فایل بیشتر از ۸ بایت باشد در هنگام نمایش به جای انتهای نام از ~ استفاده می‌کند اگر دو فایل وجود داشته باشند که در ۸ بایت اول نام آنها با هم مشترک باشند دو حرف را حذف می‌کند و با عدد تفاوت ایجاد می‌کند.

ABCDEF~1 } IJ.txt
ABCDEF~2 } JK.txt

اولی را حس با ! نشان می‌دهد.

32 بایت مربوط شامل اطلاعات زیر می‌باشد:

*	Filename	8 bytes
	File extension	3 bytes
*	File attribute	1 byte → خصوصیات از فایل را نشان می‌دهد
	Reserved	10 bytes
*	Time of last change	1 word
*	Date of last change	1 word
*	First cluster of file	1 word
*	File size	2 words

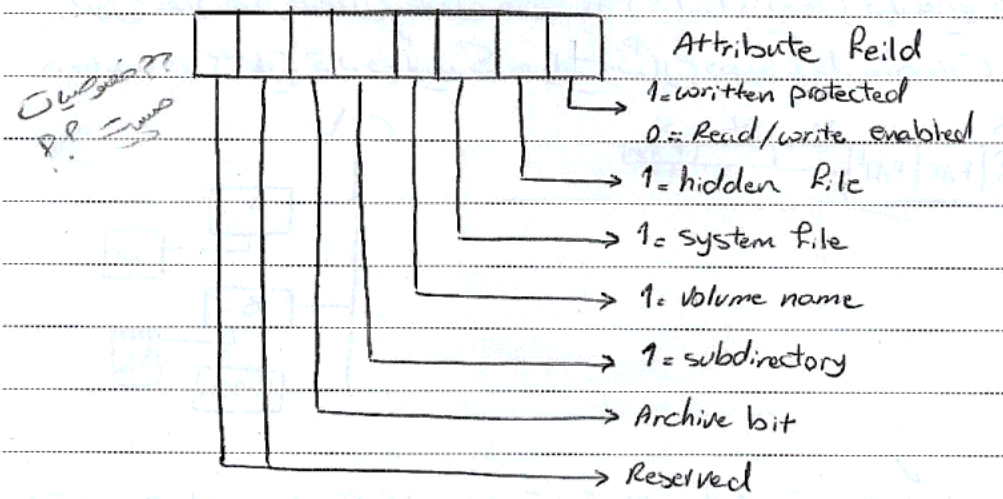
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

بایت اول از این 32 بایت اختصاص داده به هر فولدر یا فایل براساس جدول دو صفحه قبل مقداردهی می‌شود که نشان می‌دهد فایلی در این ۳۲ بایتی وجود دارد یا نه.

برای رفع مشکل نام فایل در دو اتوکد که بیش از ۸ بایت است به هر فایل دو تا 32 بایتی اختصاص می‌دهد که از طریق First cluster of file این را مشخص می‌دهد زیرا هیچ دو فایلی نمی‌توانند شروع یکسانی داشته باشند.



اگر فایلی که در هارد مشخص شده خود یک فولدر باشد در چنین شرایطی اولین cluster آن خود شامل ۳۲ بایتی هایی است که اجزای آن فولدر را نشان می‌دهد، که شناسه فیلم یا فولدر بودن آن از طریق attribute مشخص می‌شود.

جدول زیر مربوط به دو صفحه قبل است که در مورد هر cluster نشان دهنده چیست:

000H	cluster is available	→ کلاستر خالی است
FF0H _ FF6H	reserved cluster	→ چند cluster آخر که reserve شدند
FF7H	cluster damage, not used	→ cluster خراب شده
FF8H _ FFFH	Last file cluster	→ null = cluster نشان دهنده آخرین
XXXH	Next file cluster	→ در غیر این صورت cluster بعدی را در خود نگه می‌دارد

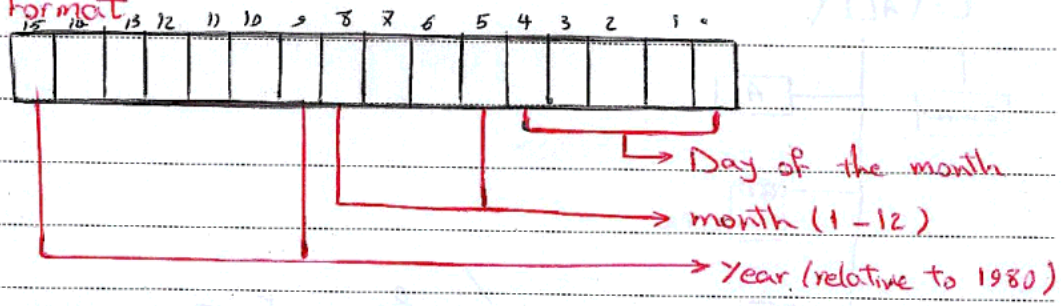
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

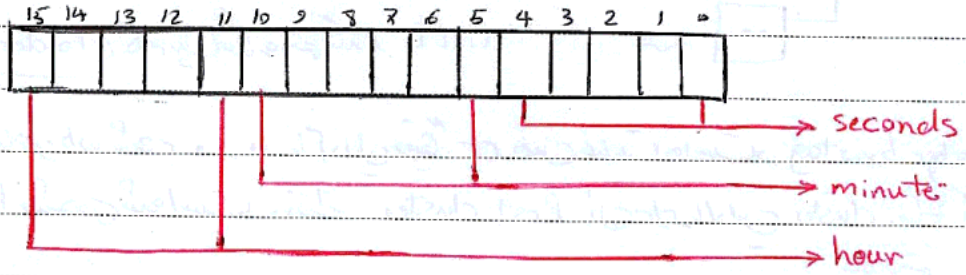
Subject:
Year. Month. Date. ( )

شیوهٔ نمایش زمان و تاریخ در یک word دارای ۳۲ بایتی به صورت زیر است:

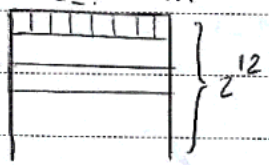
Date Feild Format



Time Feild Format:

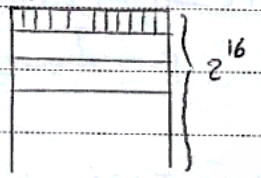


FAT ۱۲ بیتی



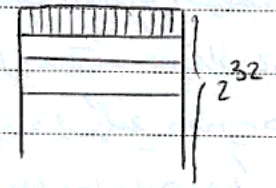
$$Size = \lceil 2^{12} \times 12 / 8 \rceil \text{ byte}$$

FAT ۱۶ بیتی



$$Size = 2^{16} \times 16 / 8 \text{ B}$$

FAT ۳۲ بیتی



$$Size = 2^{32} \times 32 / 8 \text{ B}$$

Byte per sector = 512

sector per cluster = 4

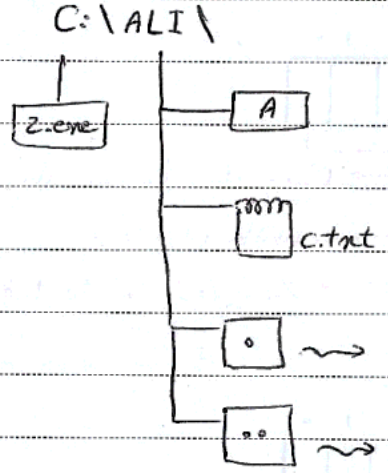
$\xrightarrow{\text{۱۲ بیتی}} \simeq 2^{12} \times 512 \times 4$ حداکثر سایز فایل

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

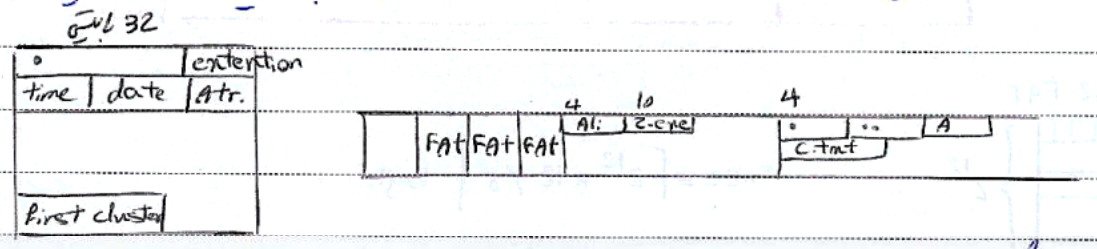
در هر فولدر علاوه بر زیرشاخه‌های موجود در فایل یا فولدر با نام‌های . و .. نیز ایجاد می‌کند



. ← فولدری است که به شاخه‌های اشاره می‌کند

.. ← فولدری که به پدر اشاره می‌کند (ALI)

در 32 بایتی‌های اختصاص داده شده به . و .. نام فایل دقیقاً برابر همین عبارت‌هاست و tag مربوط به subdirectory آنها از نوع Folder مشخص شده است و مقدار first cluster آن برابر با اولین cluster رشته آن است.



نکته: . و .. برگرفته از سیستم عامل dos است که اولین بار توسط IBM مشخص شد. علت وجود این دو فایل این است که در یک درخت بزرگ برای بازگشت به مرحله قبلی نیاز نباشد از ریشه شروع کنیم تا به پدر فایل مربوطه برسیم.

دانشکده مهندسی کامپیوتر شاهرود
اطلاعات دانشگاه صنعتی امیرکبیر

در دل این 32 بایتی‌ها تا جایی جلو می‌رود تا به 00 برسد و همین طور برای مشخص کردن long name ها چندین 32 بایتی اختصاص می‌دهد و نام آن‌ها را به ترتیب 01 و 02 و 03 و ... قرار می‌دهد که مشخص می‌کند اینها به یکدیگر وصل هستند. اگر تعداد 32 بایتی‌های موجود در یک Folder بیشتر از یک cluster شود ادامه آن در cluster دیگری قرار می‌گیرد و در دل جدول FAT این دو cluster به یکدیگر link می‌شوند.

تعداد فایل‌های موجود در یک directory با تعداد 32 بایتی‌های موجود در آن برابر نیست زیرا ممکن است به خاطر long name ها بیش از 32 بایت به یک فایل اختصاص دهد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

صفحه نمایش:

اگر تنها با سیستم عامل Dos در دست داشته باشیم و بخواهیم با صفحه نمایش ارتباط برقرار کنیم به صورت زیر است:

استاندارد موجود برای صفحه نمایش ماتریس با ابعاد 25 x 80 است.

monitor — text mode → 25 x 80
monitor — Graphic mode → x × y

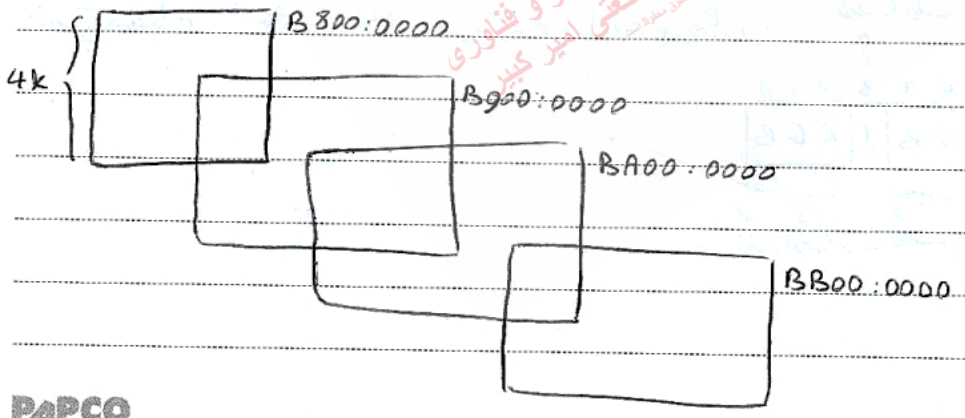
صفحه نمایش اطلاعاتش را از روی کارت گرافیک می‌خواند. کارت گرافیک هم حافظه‌ای روی RAM دارد.

تعداد کاراکتر قابل نمایش = 80 x 25 = 2000 ≃ 2k

هر کاراکتر نیز یک مشخصه دارد که رنگ کاراکتر، رنگ زمینه و حالت چشمک زن آن را مشخص می‌کند. در نتیجه برای هر کاراکتر یک بایت نیز برای attribute آن در نظر می‌گیریم در نتیجه داریم:

تعداد کاراکتر = 80 x 25 = 2000 ≃ 2k
تعداد صفات = 2000 B ≃ 2k
} 4k ← به 4k حافظه برای نگه داشتن اطلاعات صفحه نمایش نیاز داریم.

این اطلاعات در آدرس B800:0000 از حافظه به صورت ماتریسی با حجم 4k قرار دارد که به آن bitplane می‌گویند. در کارت گرافیک‌ها چهار تا از این bitplane ها وجود دارد که در واقع ارتباط مستقیم بین صفحه نمایش و این آدرس وجود دارد که هر چند ms یکبار refresh می‌شود.



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

در مبنای 16 معادل یک شیفت به سمت چپ است.

B900 :0000 → B900 x16 → B9000

− B800 :0000 → B800 x16 → B8000

01000

$16^3 = 2^{12} = 4K$ → فاصله دو bitplane از یکدیگر

درون حافظه به ازای هر کاراکتر attribute آن قرار دارد در نتیجه برای دسترسی به عنصر (i,j) از جدول آدرس آن برابر است با:

$2 * (j + 80 * i)$

خصوصیت کاراکتر

A		L		I	

(24,79)

مثال: می‌خواهیم عبارت ALI موجود در صفحه نمایش را به BLI تغییر دهیم داریم:

```
MOV BL, 'B'
CLD
MOV AX, B800H
MOV ES, AX
MOV DI, 0
MOV ES:[DI], BL
```

بایت مربوط به Attribute به صورت زیر قرار داده می‌شود:

شدت رنگ

blinking

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

رنگ پس زمینه نوشتار — رنگ نوشتار

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Monitor programming
- text → INT 21 H (DOS)
- text → INT 10 H (BIOS)
- graphic

سیستم عامل DOS شماره های گوناگونی را به عنوان پارامتر ورودی برای interrupt 21 ایجاد کرده که مشخص می کند چه کاری انجام دهد:

```
MOV  AH , ?      AH ← شماره سرویس مورد نظر
INT  10 H
```

interrupt با شماره های زیر مربوط به BIOS است.

00H : تعیین نوع صفحه نمایش
01H : تعیین اندازهٔ مکان نما
02H : تغییر محل مکان نما (cursor)
03H : پیدا کردن موقعیت و اندازه مکان نما
05H : انتخاب صفحه فعال
06H : پاک کردن صفحه نمایش و scroll up
08H : خواندن حرف و رنگ آن
09H : نوشتن تعدادی حرف و رنگ بدون تغییر مکان نما
0AH : نوشتن تعدادی حرف بدون تغییر مکان نما
0CH : روشن کردن یک pixel
0DH : خواندن مشخصات یک pixel
0EH : نوشتن یک حرف روی مانیتور و تغییر محل
0FH : پیدا کردن حالت صفحه نمایش

Graphic

اطلاعات دانشکده فنی کامپیوتر و فناوری دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

00H : (تعیین نوع صفحه نمایش)

Resolution	color	تعداد حرف	تعداد pixel برای هر حرف	حالت
400x360	mono تک رنگ	25x40	9x16	0
400x360	16	25x40	9x16	1
400x720	mono	25x80	9x16	2
400x720	16	25x80	9x16	3
400x720	mono	25x80	9x16	7

برای استفاده از این وقفه و مشخص کردن حالت مورد نظر و قرار دادن در AL تغییرات مربوط به روی صفحه نمایش اعمال می شود.

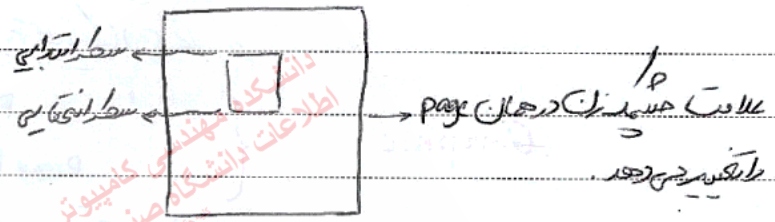
مثال:
AH ← 00H
AL ← 2
INT 10H

AH ← 00H
AL ← شماره حالت
INT 10H

01H : (تغییر اندازه مکان نما)

برای تغییر اندازه مکان نما به کار می رود و به صورت زیر عمل می کنیم:

AH ← 01H
CL ← سطر ابتدایی
CH ← سطر انتهایی
INT 10H



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

تغییر اندازه مکان نما تا زمانی بر روی فونت کاراکتر هاست و تنها باعث علامت چشمک زن را تغییر می دهد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

02 H : (تغییر محل مکان نما)

برای عوض کردن محل مکان نما باید رجیسترهای زیر را مقداردهی کنیم:

AH ← 02 H
DH ← سطر (y)
DL ← ستون (x)
BH ← page ⇝ یکی از ۴ bitplane مربوط است.
INT 10H

03H : (پیدا کردن موقعیت و اندازهٔ مکان نما)

برای یافتن موقعیت و اندازهٔ مکان نما به کار می‌رود:

AH ← 03 H
BH ← page
INT 10H

DH ⇝ محل مکان نما
DL ⇝
CL ⇝ مشخص شدن
CH اعداد مکان نما

05 H : (انتخاب صفحه فعال)

AH ← 05 H
AL ← page
INT 10H

یکی از ۴ bitplane مربوط را انتخاب می‌کنیم.

0	B800 : 0000
1	B900 : 0000
2	BA00 : 0000
3	BB00 : 0000

نکته: با اعمال هر یک از این function ها تغییر در محل ذخیرهٔ bitplane اعمال می‌شود و لزوماً بر روی صفحه نمایش نشان داده نمی‌شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

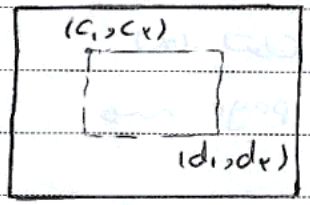
www.jozveha.ir

Subject:
Year. Month. Date. ( )

06H: (پاک کردن صفحه نمایش)

کاری در دن صفحه نمایش را انتخاب کرده و به تعداد AL از آخر کادر خالی می کند
و خط های جدیدی که می آورد را به رنگ BH تبدیل می کند
بیشترین مقدار scroll برابر با طول کادر مربوطه است

AH ← 06H
CH ← c1
CL ← c2
DH ← d1
DL ← d2
BH ← رنگ
AL ← تعداد خطوطی که باید scroll شوند
INT 10H



08H: (خواندن حرف و رنگ)

کاراکتر و ویژگی های حرف موجود در محل مکان نما را در AH و AL می ریزد

AH ← 08H
BH ← page
INT 10H

AH ⇝ att.
AL ⇝ character

09H: (نوشتن تعدادی حرف و رنگ بدون تغییر مکان نما)

به تعداد CX حرف مربوطه را بدون تغییر مکان نما چاپ می کند.

AH ← 09H
AL ← character
BL ← attr.
BH ← page
CX ← counter
INT 10H

0AH: (نوشتن تعدادی حرف بدون attr)

تنها کاراکتر مربوطه را به تعداد CX می نویسد و attr به صورت default تعیین می شود.

AH ← 0AH
AL ← char
BH ← page
CX ← counter
INT 10H

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مثال: برنامه‌ای بنویسید که صفحه نمایش را پاک کند.

```
CLS  MACRO
     MOV  AH, 06H
     MOV  CX, 0
     MOV  DH, 24
     MOV  DL, 79
     MOV  BH, 0
     MOV  AL, 50
     INT  10H
ENDM
```

mov cl, 0
mov ch, 0
یک چاهدد را مقداردهی کرده است

اگر بیشتر از خطوط صفحه بنویسیم اتفاق خاصی نمی‌افتد و فقط بیهوده scroll می‌کند.

0EH: (نوشتن یک حرف روی مانیتور)

AH ← 0EH
AL ← char.
BL ← att.
BH ← page
INT 10H

یک حرف را با att. موجود در BL بر روی page مشخص شده می‌نویسد و مکان نما را جابه‌جا می‌کند.

0FH: (پیدا کردن حالت صفحه نمایش)

AH ← 0FH
INT 10H

AL ← شماره حالت
AH ← تعداد سطر
BH ← حروف در هر page

حالت فعلی صفحه نمایش را مشخص می‌کند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Graphic Mode :

mode	Resulation	Color
04H , 05H	200 x 320	4
06H	206 x 640	Mono
0D H	200 x 320	16
0EH	200 x 640	16
0F H	350 x 640	Mono
10 H	350 x 640	16
11 H	480 x 640	2
12 H	480 x 640	16
13 H	200 x 320	256

از طریق حالت انتخابی mode صفحه نمایش انتخاب می‌شود که به صورت text یا یا Graphic.

pixel = picture element

0CH : (روشن کردن pixel)

AH ← 0CH
AL ← attr.
CX ← ستون
DX ← سطر
BH ← page
INT 10H

برای روشن کردن یک pixel به کار می‌رود.

فعلن است ابعاد صفحه در یک بایت جا نشود پس از word استفاده می‌کنیم.

0DH : (خواندن attr. یک pixel)

AH ← 0DH
CX ← ستون
DX ← سطر
BH ← page
INT 10H

با مشخص کردن محل pixel مشخصات آن را می‌خوانیم.

AL ← attr.

دانشکده مهندسی کامپیوتر و مشاوره اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

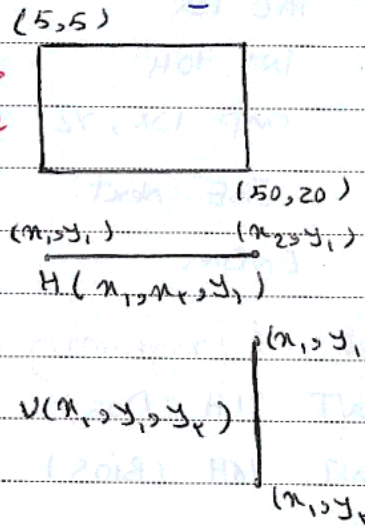
Subject:
Year. Month. Date. ( )

از طریق function شماره 00H می‌توانیم مشخص کنیم که در mode گرافیک یا text باشد.
و هم‌زمان نیز می‌توانیم از function های هر دو استفاده کنیم.

مثال: برای رسم یک خط از طریق بدست آوردن معادله خط با دادن x های گوناگون y مربوط را تعیین و pixel را روشن می‌کنیم.

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

مثال: یک مستطیل بر روی صفحه نمایش چاپ کنید.




```
H   MACRO   X1, X2, Y1
    local   AGAIN
    mov     BH, 0
    mov     AH, 0CH
    mov     AL, 0FFH
    mov     DX, Y1
    mov     CX, X1
    DEC     CX
AGAIN:
    INC     CX
    INT     10H
    CMP     CX, X2
    JBE     AGAIN
ENDM
```

چند بار حلقه تکرار می‌شود پس باید local باشد.

رنگ سفید انتخاب می‌شود

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

```
Rectangle   MACRO   X1, X2, Y1, Y2
    H   X1, X2, Y1
    H   X1, X2, Y2
    V   X1, Y1, Y2
    V   X2, Y1, Y2
ENDM
```

www.jozveha.ir

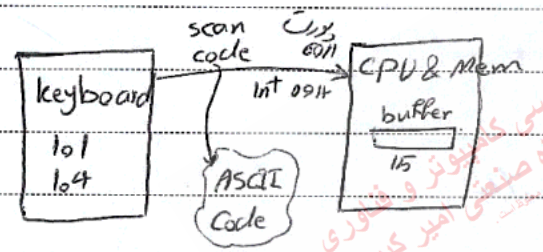
Subject:
Year. Month. Date. ( )

```
V MACRO X1, Y1, Y2
    Local Next
    mov AH, 0CH
    mov AL, 0FFH
    mov CX, X1
    mov DX, Y1
    Dec DX
Next:
    inc DX
    INT 10H
    CMP DX, Y2
    JBE Next
    ENDM
```

Keyboard Programing :

- INT 21H (DOS)
- INT 16H (BIOS)

هر از طرف یک دکمه از کیبورد scan code فرستاده می‌شود همزمان با آن int. 09 نیز اتفاق می‌افتد که بوسیله پورت 60H انتقال می‌یابد.



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

interrupt 09 مربوط به صفحه کلید است که مقدار موجود بر روی پورت 60H را می‌خواند و درون buffer که 16 بیتی است ذخیره می‌کند و به ASCII تبدیل می‌کند.
scan code شماره‌ای است که به هر دکمه از کیبورد اختصاص داده شده است.

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

INT 21 H → Function

1) 01 H : خواندن یک کاراکتر از صفحه کلید و نمایش

06 H : تشخیص فشار دادن صفحه کلید

07 H : خواندن کلید و بدون نمایش

08 H : خواندن کلید و بدون نمایش (ctrl + Break)

0A H : خواندن یک رشته

0B H : تست بافر صفحه کلید

0C H : پاک کردن بافر کلید + فراخوانی یک سرویس دیگر

1) 01 H : (خواندن یک کلید و نمایش آن)

AH ← 01 H

INT 21 H

AL → ASCII کلید

همانند دستور getche() عمل می‌کند که کاراکتر علاوه بر خوانده شدن نمایش هم داده می‌شود.

2) 06 H : (تشخیص فشار دادن صفحه کلید)

AH ← 06 H

DL ← 0 FF H

INT 21 H

AL → { 0 : بافر خالی است و کلید فشار داده نشده ، ZF = 1
کلید : کلید فشار داده شده ، ZF = 0

3) 07 H : (خواندن کلید بدون نمایش)

AH ← 07 H

INT 21 H

AL → ASCII کلید

برای خواندن password مناسب است همانند دستور getch() می‌باشد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

4. 08H : (خواندن کلید بدون نمایش)

AH ← 08H

INT 21H

AL → ASCII کلید

تفاوت با 07 این است که اگر Ctrl + break را بزنیم دستور اجرا نمی‌شود.

?? تفاوت اصلی با 0A

5) 0AH : (خواندن یک رشته)

AH ← 0AH

DX ← آدرس بافر (بافر باید ساخته شده باشد)

INT 21H

برای استفاده از این دستور باید یک buffer تعریف کنیم و طول آن را در خانهٔ اول آن قرار دهیم.
سپس با وارد کردن کاراکترها توسط کاربر وارد buffer می‌شود تا زمانی که enter زده شده باشد
و در دومین خانهٔ buffer تعداد کاراکترهای وارد شده نوشته می‌شود.

buffer

10	3	A	L	I	

buff DB _____

mov DX , offset buff

6) 0BH : (تست بافر صفحه کلید)

AH ← 0BH ?? تفاوت با 06

INT 21H

AL → { 0 ← بافر خالی است
0FFH ← بافر پر است

تست خالی بودن بافر صفحه کلید از این دستور استفاده کنیم و در صورت پر بودن اگر بخواهیم آن را خالی کنیم از interrupt بعدی استفاده می‌کنیم.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

7) 0CH : (پاک کردن بافر صفحه کلید)

AH ← 0C H

AL ← شماره سرویس (int 21H)

DX ← تنظیم

INT 21H

در این حالت بافر پاک می‌شود و سرویس AL فراخوانی می‌شود که باید تنظیمات سرویسی که می‌خواهیم بعد از آن اجرا شود را درون DX تنظیم کنیم که این سرویس فقط از interr. های 21 می‌تواند باشد.

INT 16H : function

00 H : خواندن یک حرف بدون نمایش

01 H : بررسی فشار دادن یک حرف

10 H : همان 00H ولی برای کیبوردهای جدید

11 H : آیا حرفی در بافر است یا خیر

02 H : گزارش وضعیت صفحه کلید

1) 00 H : (خواندن یک حرف بدون نمایش)

AH ← 00 H

INT 16H

AL → ascii

AH → scan code

AL → 0

AH → شماره ⇐ F1 تا F12

یک حرف از کیبورد می‌خواند و با توجه به مقدار AL و AH نوع آن را به شما می‌دهد.

2) 01H : (بررسی فشار دادن یک کلید)

AH ← 01 H

INT 16H

ZF { 1 هیچ کلیدی زده نشده
ZF { 0 کلیدی زده شده است

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

02H : (گزارش وضعیت صفحه کلید)

AH ← 02H

INT 16H

AL:
- $AL_0 = 1$ Right Shift is pressed
- $AL_1 = 1$ Left shift is pressed
- $AL_2 = 1$ ctrl
- $AL_3 = 1$ ALt
- $AL_4 = 1$ Scroll Lock
- $AL_5 = 1$ Num Lock
- $AL_6 = 1$ caps Lock
- $AL_7 = 1$ Insert

مقدار هر bit از AL نشان می‌دهد که کدام یک از این کلیدها زده شده باشد ولی وضعیت آن را نمی‌توانیم تشخیص دهیم.

11H : (وجود حرفی در بافر)

AH ← 11H

INT 16H

ZF:
- 0 : بافر پر
- 1 : بافر خالی

AL → ASCII

AH → Scan Code

دو آرگومان برای یک حرف در بافر است؟!

نکته: دستورات 10H نیز همانند 00H است و هیچ تفاوتی با هم ندارند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Timer / Date / Time :

برای انجام عملیاتی که در زمان مشخصی می‌خواهیم انجام دهیم یا ایجاد تأخیر در برنامه از این دستورات استفاده می‌کنیم.

CX : DX

[ ] ← clock = 18.2 Hz (در هر ثانیه 182 clock می‌زند)

32 bit

↳ $Max = 2^{32} - 1$

min = 0

که در هر 18.2 Hz یکی به مقدار درون آن اضافه می‌شود تا به max خود برسد و پس از اول شروع می‌کند به این نوع از وقفه‌ها، timer های سخت افزاری می‌گویند. نوشتن یک for که چندین بار تکرار شود به سیستم عامل و مشخصات دستگاه وابسته است و ممکن است در زمان برنامه به نوع کامپیوتر متفاوت باشد (مقدار clock یک استاندارد در هر نوع دستگاه است)

نکته: وقفه‌های زیر 20 مربوط به BIOS است و وقفه‌های مربوط به timer از نوع Bioc است.

1) 00H : (خواندن شمارنده)

AH ← 00H

INT 1AH

CX : DX مقدار جاری شمارنده

2) 01H : (تنظیم شمارنده)

AH ← 01H

CX : DX ← مقدار مورد نظر

INT 1AH

در این حالت در هر ثانیه 18 بار clock می‌خورد پس برای ایجاد تأخیر 1 ثانیه باید مقدار درون آن 182 تا زیادتر شده باشد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

02 H : (خواندن ساعت جاری)

AH ← 02 H

INT 1AH

CH → ساعت (BCD)

CL → دقیقه (BCD)

DH → ثانیه (BCD)

CF → { 0 ساعت کار می‌کند / 1 ساعت کار نمی‌کند

باید هر بایت از رجیسترها را به عنوان دو رقم بخوانیم چون از نوع BCD است

CL [5 | 9] → (هر بایت نشان دهنده یک رقم است) دقیقه 59 است

03 H : (تنظیم ساعت)

AH ← 03 H

CH ← ساعت

CL ← دقیقه

DH ← ثانیه

INT 1AH

04 H : (خواندن تاریخ)

AH ← 04 H

INT 1A H

CH → قرن (BCD)

CL → سال (BCD)

DH → ماه (BCD)

DL → روز (BCD)

CF { 0 کار می‌کند / 1 کار نمی‌کند

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

از چهار رقم مربوط به سال دو رقم اول وارد CH و دو رقم دوم وارد CL می‌شود.

05 H : (تنظیم تاریخ)

AH ← 05 H

CH ← قرن

CL ← سال

DH ← ماه

DL ← روز

INT 1A H

تاریخ را بوسیله این دستور تنظیم می‌کنیم.

INT 21 H (DOS) :

1) 2CH : (خواندن ساعت)

AH ← 2CH

INT 21H

CH ↔ ساعت (binary)

CL ↔ دقیقه (binary)

DH ↔ ثانیه (binary)

DL ↔ صدم ثانیه (binary)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

2) 2DH : (نوشتن ساعت)

AH ← 2DH

CH ↔

CL ↔

DH ↔

DL ↔

INT 21H

AL { 0 ← تنظیم شد
FF ← تنظیم شد

www.jozveha.ir

Subject:
Year. Month. Date. ( )

3) 2AH : (خواندن تاریخ)

AH ← 2AH

INT 21H

AL → روز هفته → { ۰ : یکشنبه ، ۱ : دوشنبه ، ... }

CX → سال ۴ رقمی

DH → ماه

DL → روز

4) 2BH : (تنظیم تاریخ)

AH ← 2BH

CX ← سال

DH ← ماه

DL ← روز

INT 21H

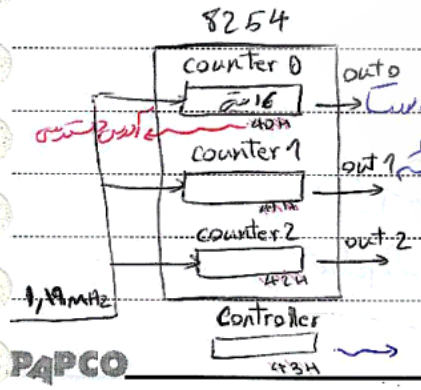
AL { ۰ تنظیم نشده ، ۱ تنظیم شده }

در این حالت نیاز نیست AL را تنظیم کنیم چون روز خودش تنظیم می‌شود.

Sound:

برای ایجاد صدا اگر فرکانس زیاد باشد صدا زیر و اگر فرکانس کم باشد صدا بم است.

در کامپیوترها IC با شماره 8254 وجود دارد که وظیفه‌ی آن تولید صداست و به آن PIT نیز می‌گویند (programmable Interval Timer). یکی از وظایف این IC ایجاد 18.2 Hz برای timer است.



درون این IC چندین counter ۱۶ بیتی است که قابل برنامه‌ریزی است و می‌توان مشخص کرد از کدام counter استفاده کنیم. درون این تراشه یک رجیستر وجود دارد که خروجی را کنترل می‌کند.

اینکه از کدام counter و چگونه استفاده کنیم از controller استفاده می‌کنیم.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

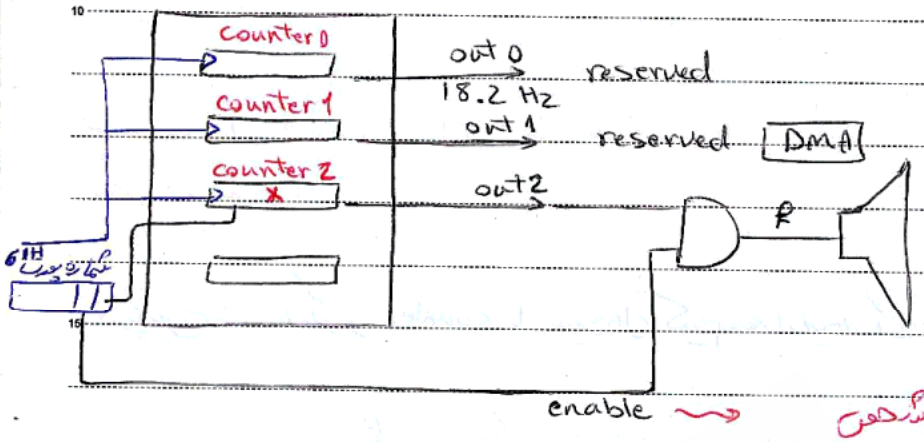
www.jozveha.ir

Subject:
Year. Month. Date. ( )

برای اینکه ورودی فرکانس این ترانسیستور در هر نوع کامپیوتری ثابت باشد از طریق تقسیم کننده فرکانس ثابت 1,19MHz را برای آن می‌فرستیم.

بهتر است ترانسیستورها را تنها در زمانی که به آنها احتیاج داریم روشن کنیم زیرا در زمانی که روشن هستند توان را به خود اختصاص می‌دهد. برای انجام این کار از chip enable استفاده می‌کنیم که با قرار گرفتن مداری در مقابل Address bus و دادن خروجی آن به chip enable این کار انجام می‌شود که شماره port آن را از روی همین مدار می‌توان تشخیص داد.

نوع شمارش counter ها (up / down) توسط Controller مشخص می‌شود. دو counter شماره 0 و 1 خروجی گره شده دارند و وارد دارهای داخلی خود بسته شده است. فعلاً ایجاد فرکانس 18.2 Hz از out0 استفاده می‌کنیم و از out1 برای دستگاه IO یا با DMA استفاده می‌شود.

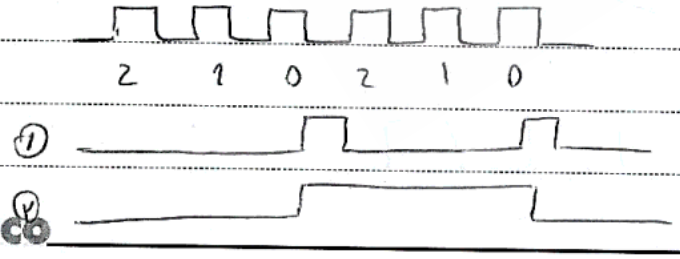


پایه فرمان برای ایجاد صدا در زمان مشخص

سؤال: برای تولید فرکانس f چه مقداری داخل counter 2 باید قرار داده شود؟

$$f = \frac{1,19 \times 10^6}{X}$$

نکته: هرگاه مقدار counter به صفر رسید یک پالس خروجی ایجاد می‌کند که نوع پالس را نیز می‌توان مشخص کرد. ورودی counter، clock خود counter را مشخص می‌کند و با تنظیم داده اولیه counter فرکانس خروجی را تنظیم می‌کنیم.



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

برای ایجاد صدا در بلندگو باید مراحل زیر را انجام دهیم:

1. مقداردهی B6H بر روی controller
2. مقداردهی X به counter2
3. فعال کردن بلندگو

```
MOV  AL, B6H
OUT  43H, AL
MOV  AX, 2943H
```

تنظیم 8254 برای اینکه بعد از شمارنده ۲ استفاده کنیم.

$$x = \frac{1.19 \times 10^6}{f}$$ (2943)

```
out  42H, AL
out  42H, AH
```

bus داده این تراشه ۸ بیتی است پس برای مقداردهی کردن counter ابتدا بایت کم ارزش و سپس بایت پرارزش ترا می فرستیم.

```
IN   AL, 61H
OR   AL, 00000011b
OUT  61H, AL
```

بیت صفر و یک از پورت 61H را یک می کند تا counter و بلندگو فعال شوند.

در پایان باید بلندگو و counter را غیرفعال کنیم و بعد از ایجاد یک delay این دستگاه را غیرفعال می کنیم

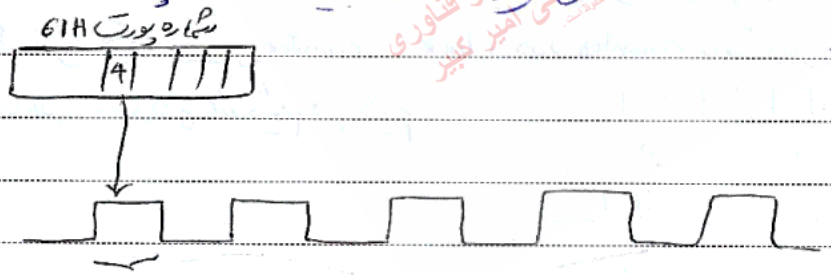
```
MOV  CX, 01H
while  loop  while
```

زمان : 65537 × تعداد کلاک × f_{clock}

این تأخیر نرم افزاری مناسب نیست زیرا بستگی به نوع کامپیوتر دارد.

چگونه تأخیر ایجاد کنیم؟

یکی از روش ها استفاده از timer است. روش دیگر استفاده از بیت شماره 4 پورت 61H زیرا با clock ثابتی تغییر می کند.



دانشکده مهندسی کامپیوتر دانشگاه علامه طباطبائی
اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

* تمرین: برنامه‌ای بنویسید که الف) 1s تأخیر سخت افزاری ایجاد کند. ب) طبق جدول زیر را تولید کند.

Subject:
Year. Month. Date. ( )

f	مدت
20	1s
~~120~~	~~0.5s~~
400	2s
1000	5s

```
        mov     Cx , عدد من
        mov     AH, 0001 0000b

Read:   IN      AL, 61H
        AND     AL, 0001 0000 b
        CMP     AL, AH
        JE      Read
        XOR     AH , 0001 0000 b
        Loop    Read
```

15.08μs

در ابتدا تا زمانی که لبه یک را طی می‌کند تکرار می‌شود و پس از آن لبه صفر تکرار می‌شود.

به هر لبه که رسید مقداری که با AL مقایسه می‌کند را عوض کند.

$1ms = Cx * 15.08\mu s$ $\qquad Cx \simeq \frac{1000}{15.08}$

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:

Year. Month. Date. ( )

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

PAPCO

www.jozveha.ir

BIOS ROM

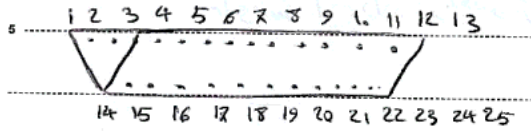
parallel port : (درگاه موازی)

port موجود بر روی کامپیوترهای کنونی که از نوع parallel هستند به نام LPT هستند

(long print terminal)

شماره پورت	اسم مجازی	اسم واقعی
0	LPT1	
1	LPT2	
2	LPT3	
3	LPT4	

LPT



این پورت حاوی ۲۵ پایه است و یک شماره port به آن اختصاص داده شده است. از اسم مجازی برای interrupt ها استفاده می کنیم و از شماره پورت برای (IN / OUT) استفاده می شود.
اگر بخواهیم به صورت نرم افزاری شماره پورت را مشخص دهیم از آدرس زیر از حافظه می توان این مقدار را خواند.

0040 : 0000

مربوط به پورت سریال است ←

0040 : 0008 } LPT1
0040 : 0009

0040 : 000A } LPT2
0040 : 000B

0040 : 000C } LPT3
0040 : 000D

0040 : 000E } LPT4
0040 : 000F

شماره پورت مربوطه در ← این خانه قرار دارد.
(مقادیر little_endian هستند.)

78H
03H } ⇒ 0378H = port(LPT1)

اگر مقدار موجود در این دو بایتی ها صفر باشد یعنی چنین پورتی موجود نیست و می توان از روی این مقدار تعداد پورت های موازی را مشخص دهیم.
در واقع هرگاه به اطلاعاتی نیاز داشتیم که از دنیای بیرون با سرعت نسبتاً بالایی انتقال یابد از پورت parallel استفاده می کنیم.

دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

www.jozveha.ir

Defect : عیب
Fault : اشکال
error : خطا
Failure : خرابی

Subject:
Year. Month. Date. ( )

Pin

1 : strobe → اعلام آمادگی کامپیوتر

2 : D_0
3 D_1
4 D_2
5 D_3
6 D_4
7 D_5
8 D_6
9 D_7

Data → برای انتقال داده از کامپیوتر به printer

10 $\overline{ACK}$ acknowledge ← سیگنال تصدیق از printer به کامپیوتر (Active low)

11 Busy ← مشغول بودن printer

12 out of paper ← نبودن کاغذ در پرینتر

13 Select ← چاپگر انتخاب شد Conline_offline

14 $\overline{\text{Auto Feed}}$ → تنظیم برای سرصفحه رفتن (Active low)

15 $\overline{\text{Error}}$ ← درصورت بروز خطا در چاپگر (Active low)

16 Initialize → راه اندازی اولیه

17 $\overline{\text{Select Input}}$ → انتخاب یک چاپگر

11 کامپیوتر منتظر می‌ماند تا سیگنال busy از طرف printer غیرفعال شود.
12 کامپیوتر سیگنال strobe را فعال می‌کند
13 داده‌ها از طریق $D_0 - D_7$ ارسال می‌شوند
14 پرینتر سیگنال ACK را فعال می‌کند. (نشان دهنده دریافت داده از طرف چاپگر است)

debug.exe

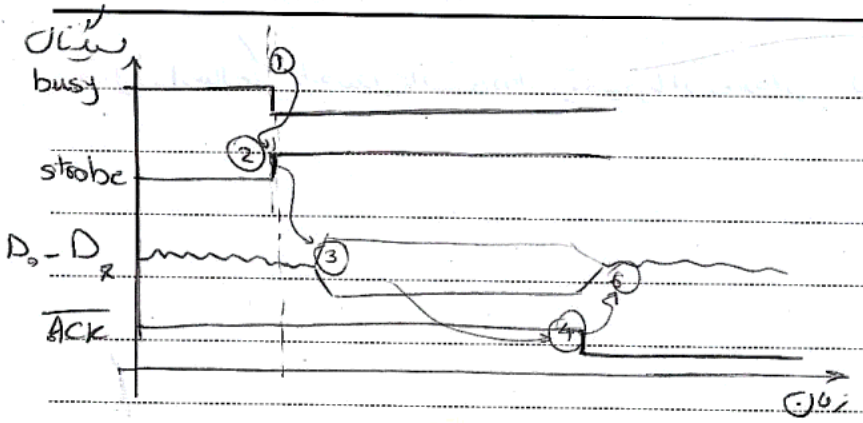
-d 0040:0008 L8

سایر پایه‌ها از 18 تا 25 زمین شده هستند و به زمین وصل می‌شوند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )



برای استفاده از parallel port از دو روش زیر می‌توان استفاده کرد:

۱- استفاده از وقفه (INT 17) :

۲- استفاده از دستورات IN / OUT

INT 17H :

① 00H (ارسال داده) :

AH ← 00H

AL ← data

DX ← LPT شماره سنبی { 1, 2, 3 }

INT 17H

AH → وضعیت فرستادن را در خود نگه می‌دارد مانند 02H .

② 01H (راه‌اندازی اولیه) :

AH ← 01H

DX ↔ LPT شماره سنبی

INT 17H

③ 02H (خواندن وضعیت پرینتر) :

AH ← 02H

DX ← LPT شماره سنبی

INT 17H

AH → اطلاعات فرستاده شده ولی در مدت زمان معین به دلیلی پاسخ داده نشده

$AH_0 = 1$ → time out

$AH_3 = 1$ → I/O Error

$AH_4 = 1$ → printer is activated

$AH_5 = 1$ → No paper

$AH_6 = 1$ → No Ack

$AH_7 = 1$ → busy error

روشن یا خاموش بودن آن را مشخص می‌کند

اگر صفر باشد یعنی busy است و اگر ۱ باشد آزاد است.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

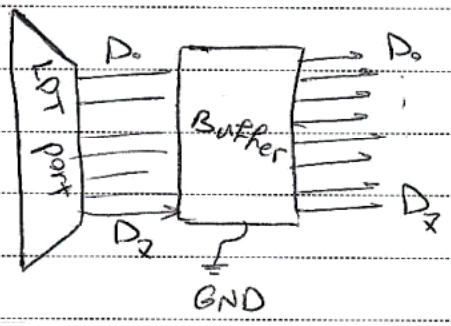
Subject:
Year. Month. Date. ( )

مثال: ارسال یک داده بدون استفاده از وقفه و با استفاده از IN / OUT :

```
MOV  AX, 40H
MOV  ES, AX
MOV  DI, 08H
MOV  DX, ES:[DI]
CMP  DX, 0
JZ   skep
MOV  AL, '2'
OUT  DX, AL

skep:
```

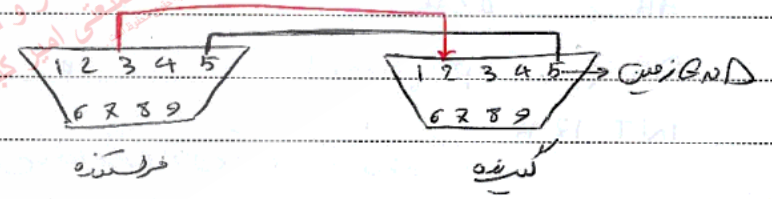
می‌توان در مقابل LPT port یک بافر قرار داد تا مقدار خروجی پورت بیرون آن ذخیره شود. مقداری که می‌تواند در خروجی دارد یک بایت است و دیتا را تقویت نیز می‌کند در نتیجه می‌توان از خروجی آن اشکال‌های زیادی ایجاد کرد.



Serial port:

پورت‌های com که از آدرس 0000 تا 0007 در سگمنت 0040 است که 9 pin دارد.

```
0040 : 0000 } Com1
       0001
       0002 } Com2
       0003
       0004 } Com3
       0005
       0006 } Com4
       0007
```



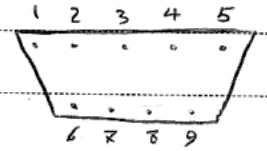
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

حداقل با دو سیم می‌توان از طریق پورت سریال دو کامپیوتر را به یکدیگر وصل کرد و بتوانند تبادل اطلاعات کنند.

برای ارسال به صورت سری بر روی یک خط استفاده می‌شود. نقطه ضعف آن این است که سرعت آن از parallel کم‌تر هست ولی سریال مصرف بیشتری دارد زیرا برای وصل کردن دو دستگاه تنها به دو سیم احتیاج داریم.



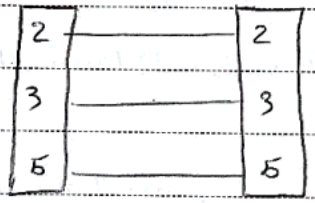
RS.232 ⇝ به استاندارد برای دریافت و ارسال اطلاعات می‌گویند

2: ⟶ receive (دریافت)

3: ⟵ send (ارسال)

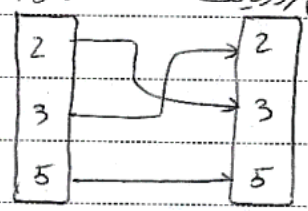
5: GND باید منبع پایه دو دستگاه با یکدیگر برابر باشند تا مبدأ هر دو شاسی باشد و صفر یک برای هر دو یک معنا داشته باشد. (منبع تغذیه مشترک)

1)



کابل ارتباط مستقیم اتصال رابط

2)



کابل send recieving کابل ارسال

Cross Cable

دستگاه‌های کار با RS.232:

1) Modem ⟶ کاربردهای com
2) Mouse
3) Monitor
4) Joy stick
5) Reciever's
6) Card reader
7) Barcode Reader

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

آدرس شروع شماره پورت com

	شماره پورت	شماره سنتی (مورد استفاده از وقفه)	
0040 : 0000	com1	0	← com1
0040 : 0001		1	← com2
0040 : 0002	com2	2	← com3
0040 : 0003		3	← com4
0040 : 0004	com3		
0040 : 0005			
0040 : 0006	com4		
0040 : 0007			

UART: Universal Asyncrone Receive and Transmit

این IC بقیه پایه‌ها و نحوه ارتباط و signaling را کنترل می‌کند و شیت پورت com قرار می‌گیرد.
که شماره پورت‌ها نیز در کنار آن نوشته شده است.

روش‌های کار کردن با پورت سریال:

1. استفاده از دستورات IN/out

2. استفاده از وقفه (BIOS 14H)

1) IN / out :

```
cmd ↵
debug.exe
-d 0040:0000 L8 ⇝ 78 03 78 02 00 00 00 00
```

com1 com2

0378H = com1 شماره پورت com2 شماره پورت = 0278H

```
mov AX, 40H
mov ES, AX
mov DI, 0
mov DX, ES:[DI]
cmp DX, 0
JZ no_com
```

```
mov AL, کاراکتر مورد نظر
OUT DX, AL
```
⇝ برای ارسال

```
IN AL, DX
no_com:
```
⇝ برای دریافت

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

Pin: (modem)

1 DCD: data Carrier Detect ← ورودی که دستگاه خارجی (مودم) مشخص می‌دهد که وصل است

2 Recieve Data ← انتقال اطلاعات از مودم به کامپیوتر

3 transmit Data → اعلام آمادگی برای انتقال در دریافت اطلاعات توسط کامپیوتر و مودم می‌خورد که دستگاه مقابل وصل شده

4 DTR: Data terminal Ready → مودم اعلام می‌کند به دستگاه دیگری وصل شده

5 ground

6 DSR: Data set ready → مودم درخواست فرستادن اطلاعات می‌کند با فعال کردن این پایه

7 RTS: Request to send ← اطلاعات فرستاده شد و آماده برای اطلاعات بعدی است.

8 Clear to send (CTS) ← (کامپیوتر از مودم سؤال می‌کند که آیا آماده دریافت اطلاعات است)

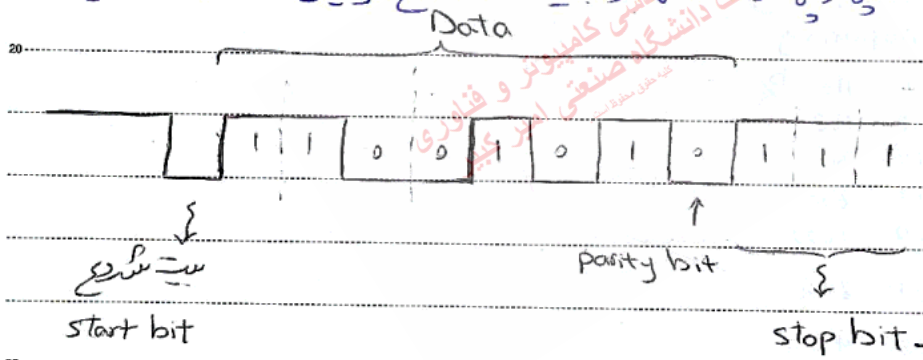
9 Ring Indicator (Ring) ← مودم در جواب به درخواست کامپیوتر این شاخه را فعال می‌کند

در این روش باید داده‌ها با یک زمان مشخصی فرستاده شود. و رعایت کردن delay بر عهده فرستنده و گیرنده است. این ارتباط زمانی است و در زمان‌های گوناگون نمونه‌گیری می‌کند پس باید گیرنده و فرستنده در clock زمانی توافق کنند.

اولین طراحی برای com از نوع modem بود و ارتباطات دقیقا برای مودم است ولی می‌توان برای سایر وسایل نیز به کار برد.

در انتقال اطلاعات چون به صورت خطی است در ابتدا سیگنال یک است هنگام فرستادن ابتدا یک glitch ایجاد شده که نشان می‌دهد قرار است اطلاعات فرستاده شود و بعد از آن ۸ بیت اطلاعات فرستاده می‌شود.

در این روش چون اطلاعات پی‌درپی فرستاده می‌شود باید زمان شروع و پایان ارسال مشخص شود.



سه نمونه اضافه برای دارد که نشان می‌دهد به درستی خوانده شده است.

(نرخ سیگنال)
boundRate = f_1 ——زمان——→ f_2
فرستنده — گیرنده

PAPCO

www.jozveha.ir

ابتدا DTR و DSR فعال شده سپس کامپیوتر باید RTS را فعال کرده زمانی که دوم برای دریافت اطلاعات آماده شود
با فعال شدن CTS مشخص می‌شود اگر در حین ارسال CTS غیرفعال شود ارسال اطلاعات متوقف شده تا دوباره فعال شود.

Subject:
Year. Month. Date. ( )

برای هماهنگ شدن باید فرستنده و گیرنده قبل از دریافت یا ارسال داده در فرکانس با هم توافق کنند.

نکته: اگر بسیار یک اختلاف باشد نشان می‌دهد یا فرکانس‌ها با یکدیگر هماهنگ نیستند یا noise بوده و در نتیجه اطلاعات خوانده شده درست دریافت نمی‌شود.

Data فرستاده شده در بعضی مواقع ۷ بیتی و گاهی ۸ بیتی است. (به دلیل وجود parity bit)
برای ارسال داده باید نقطه شروع، طول ارسال و نرخ سیگنال را مشخص کرد و سپس داده برای ارسال آماده می‌شود.

INT 14:

① 00H (initialize):

AH ← 0

AL ← تنظیمات سریال

INT 14H ← DX ← شماره سری پورت

AL:

- بیت 0 و 1
  - → 10 داده‌ها ۷ بیتی است
  - → 11 داده‌ها ۸ بیتی است ← تنظیمات معمول برای رایانه
- 2
  - → 0 : عدد stop bit دارد ←
  - → 1 : ۲ عدد stopbit دارد
- 4,3
  - → 00 : no parity ←
  - → 01 : odd parity
  - → 10 : even parity
  - → 11 : noparity
- 5_7
  - → 000 : 110 b/s
  - → 001 : 150
  - → 010 : 300
  - → 011 : 600
  - → 100 : 1200
  - → 101 : 2400
  - → 110 : 4800
  - → 111 : 9600 ←

سرعت ارسال داده را مشخص می‌کند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

در ارسال از طریق سریال کامپیوتر سریع تر از دو آی او می‌کند زیرا دو آی او زمانی که buffer آن پر شده اطلاعات را ارسال می‌کند یا دریافت

Subject: پر شدن کامل CTS یا غیرفعال می‌کند زمانی که RTS غیرفعال شود CTS نیز بلافاصله low تبدیل می‌شود

Year. Month. Date. (/)

یا فعال شدن DTR توسط کامپیوتر دوم اطلاعاتی را برای کامپیوتر ارسال می‌کند.

مثال: اگر دستگاه بردی 4800 تنظیم باشد ولی داده با سرعت 2400 فرستاده شود هر دو بایت را یک بایت فرض می‌کند.

② 01 H : (ارسال داده)

AH ← 01 H

AL ← data

DX ← شماره سری

INT 14 H

↳ $AH_7 = 0$: successful

$AH_8 = 1$ → وضعیت درست ارسال گردیده

AL:
- change to CTS line
- change to DSR line
- change to RI line
- change to RLSD line
- CTS
- DSR
- telephone ringing
- carrier detect

③ 02 H : (دریافت داده)

AH ← 02 H

DX ← شماره سری

INT 14 H

AL → data

AH_8 → = 0 داده صحیح است

AH:
- input character available
- overrun error
- parity error
- transmission error
- alarm detected from the other
- transmission hold register
- " shift reg. is empty
- time out of recieving a ch

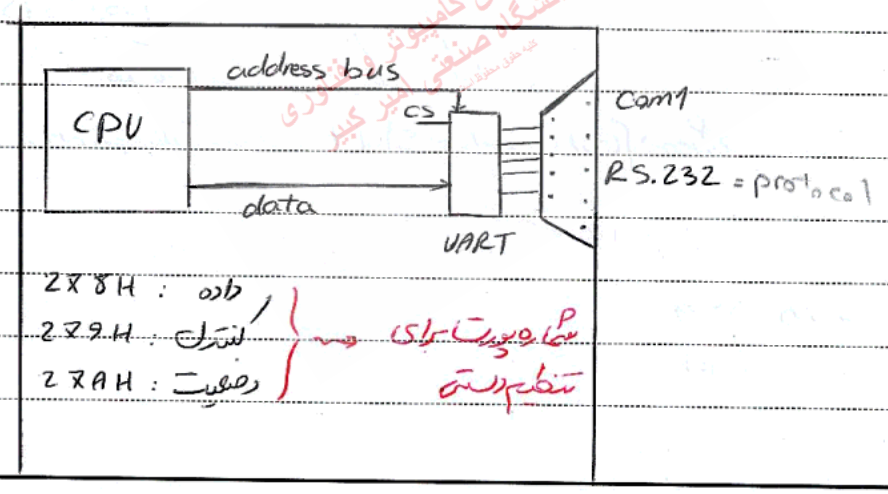
④ 03 H : (دریافت وضعیت)

AH ← 03 H

INT 14 H

AX → status

اصلاً اشکال در دریافت داده است نه ارسال داده و هنگامی که داده‌ای دریافت می‌شود هر چه درون بافر باشد می‌خواند از طریق AH_8 صحت آن را مشخص می‌دهد.



www.jozveha.ir

Subject:
Year. Month. Date. ( )

وظیفه UART : رجیستری است که در پشت پورت سریال مربوط به ارسال و دریافت اطلاعات را دریافت می‌کند و در خود ذخیره می‌کند زمانی که تمام بیت‌های یک کاراکتر دریافت شد در صورتی که خطای اتفاق نیافتاده بود داده را ارسال می‌کند.

TSR:

برنامه‌های مقیم در حافظه به طوری که بخواهیم یک برنامه در چند وقت یکبار تکرار شود.

TSR : Terminate program & stay Resident

1) Stay Resident:

اولین مرحله قرار دادن برنامه در حافظه است در برنامه‌های TSR از دو خط زیر استفاده می‌کنیم چون این دو خط سبب برگشتن برنامه به سیستم عامل می‌شود.

```
mov AX, 4C00H
INT 21H
```

→ سیستم عامل

در واقع با این دستورات برنامه خاتمه می‌یابد و کنترل اجرا به سیستم عامل برگردانده می‌شود.

برای مقیم کردن برنامه در حافظه از int زیر استفاده می‌کنیم

```
mov AH, 31 H
mov DX, اندازه برنامه برحسب پاراگراف
INT 21H
```

نکته: برای محاسبه برنامه برحسب پاراگراف باید اندازه را تقسیم بر 16 کنیم و رو به بالا گرد کنیم و حاصل را در DX قرار دهیم.

هر 16 بایت ← یک پاراگراف

تقسیم بر 16 در اسمبلی با استفاده از shift به راست انجام می‌شود.

(اندازه برنامه + 15) / 16 → خارج قسمت → DX

+ 15 >> 4
shift

www.jozveha.ir

Subject:
Year. Month. Date. ( )

21 Activation ← ببینیم هر چه وقت بیدار شود

برای اجرای برنامه در زمان های متفاوت می توان از یکی از روش های زیر استفاده کرد:

۱- وصل کردن به timer

۲- وصل کردن به صفحه کلید. (هر بار کلیدی زده می شود اجرا شود.)

در این روش ها باید برنامه را جایگزین یکی از وقفه های سخت افزاری کنیم.

نکته: برنامه هایی که به صورت TSR اند باید به صورت com نوشته شوند.

activation → keyboard → 09H

activation → Timer → 08H

در این حالت سرعت فراخوانی وقفه 18.2Hz است یعنی در هر ثانیه 18 بار اجرا می شود.

باید یک برنامه نوشته شود که در هر بار فراخوانی ابتدا برنامه ما اجرا شود و سپس routine interrupt چون نمی دانیم در حین اجرای routine interrupt چه می شود و کنترل برنامه دست ما نیست.

در نتیجه درون interrupt vector که دو بایت اول offset و دو بایت دوم segment را شامل می شود باید آدرس seg برنامه خود را قرار دهیم.

مراحل activation:

۱- مشخص کردن وقفه فعال سازی (شماره وقفه ای که می خواهیم برنامه خود را به جای آن بنشانیم.)

2- خواندن segment و آفست روتین وقفه قدیمی و نگهداری آن

3- جایگزین کردن segment و آفست برنامه مقیم در حافظه در interrupt vector

4- قرار دادن پرش در برنامه مقیم در حافظه به segment و آفست (که با هر بار صدا زدن وقفه آن هم انجام شود)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شاهرود

www.jozveha.ir

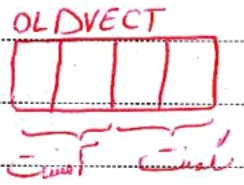
Subject:
Year. Month. Date. ( )

مرحله‌ی اول و دوم:

Example: n = 08H (timer)

```
OLDVECT   DD ?
mov       AX, 0
mov       ES, AX
mov       DI, 32   (08 x 4 = 32)
mov       OLDVECT, ES:[DI]
mov       DI, 34
mov       OLDVECT+2, ES:[DI]
```

۱- روش مستقیم:



۲- روش غیرمستقیم (وقفه‌ی ۱):

```
mov       AH, 35H
mov       AL, شماره وقفه مورد نظر
INT       21H
```

در این مثال 08H است

ES ⇝ سگمنت روتین مورد نظر در AL

BX ⇝ آفست روتین مورد نظر در AL

به صورت کامل:

```
mov       AH, 35H
mov       AL, 08H
INT       21H
mov       OLDVECT, BX
mov       OLDVECT+2, ES
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

نکته: چون برنامه TSR باید با فرمت COM باشد در نتیجه proc در برچسب نمی‌تواند از نوع FAR باشد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مرحلهٔ سوم:

در این مرحله باید offset و seg برنامهٔ خود را درون int. vector جایگزین کنیم که برای بدست آوردن این دو دو کلمه آفست را از دستور زیر و سگمنت نیز همان CS است

```
newproc proc Near
{
mov ES, OLDVECT +2
JMP ES: OLDVECT
ENDP
```

offset آن را می‌دهد ← offset Newproc

seg آن را می‌دهد ← Segment Newproc

برای جایگزین کردن این مقادیر می‌توان به روش مستقیم و با برعکس کردن مرحلهٔ قبل مقادیر را قرار داد و یا از یک int زیر استفاده کنیم:

```
mov AH, 25H
mov AL, شماره وقفه ← 08H
mov DX, آفست که ← OFFSET newproc
INT 21H
```

این interrupt سگمنت و آفست خود نقطه را در سگمنت و آفست interrupt vector ذخیره می‌کند مقدار seg را نمی‌رسد چون همان سگمنت که برنامه است

مرحلهٔ چهارم:

اضافه کردن دستور jump به انتهای STR که در بالا نشان داده شد این jump از نوع FAR است. این برنامه به طور طبیعی در هر ثانیه 18 بار اجرا می‌شود که با قرار دادن counter می‌توانیم برنامه بفهمانیم که در هر ثانیه یک بار اجرا شود.

نکته: routine interrupt ها باید کم باشند و زمان زیادی طول نکشند که عملکرد کامپیوتر زیر سؤال نرود

www.jozveha.ir

Subject:
Year. Month. Date. ( )

مثال: برنامه‌ای بنویسید که با هر بار فشار دادن صفحه کلید، اسمی را در صفحه نمایش چاپ کند.

```
mySEG   SEGMENT  para  public  'code'
        Assume   CS: mySEG , DS: myseg
        ORG      100 H
main:   jmp  load1
        OLDVECT  DD  ?
        new1  proc  Near
        {  ⟶  عملیات مربوطه
                 word ptr
        mov   ES , OLDVECT + 2
        jmp   ES : OLDVECT
        ENDP

Load1:
        mov   AH , 35H
        mov   AL , 09 H
        INT   21 H
        mov   OLDVECT , BX
        mov   OLDVECT , ES
        ⟶ نگه داشتن offset و seg در حافظه

        mov   AH , 25H
        mov   AL , 09 H
        mov   DX , offset  new1
        INT   21 H
        قرار دادن seg و offset جدید در int. vector

        mov   AH , 31 H
        mov   DX , *
        INT   21 H
        باقی ماندن برنامه در حافظه
ENDS
END main
```

دانشکده مهندسی عامل و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

برای بدست آوردن اندازهٔ برنامه می‌توان گفت با تفریق offset دو label، main و load1 این مقدار بدست می‌آید. مهم این است که کل proc در حافظه باشد و دایرکتیو خطوط نیز قرار گیرد تا مشکلی بوجود نمی‌آید.

* → (offset load1 - offset main) + 15 / 16

چون تمام اجزای عبارت directive است می‌توان مستقیماً این عبارت را قرارداد در عمل این صورت از کدی استفاده می‌کنیم:

```
mov DX, offset load1 - offset main
ADD DX, 15
mov CL, 4
SHR DX, 4
```

حال چگونه می‌توان ۱- برنامه را از Active بودن خارج کرد؟
۲- از stay بودن خارج کرد؟

اولین راه restart کردن کامپیوتر است دومی اینکه از طریق برنامه debugger، dump کنیم و تا جایی در برنامه پیش برویم که به jump آخر newPRG برسیم و از طریق آن seg و offset interrupt قبلی را پیدا کنیم.

راه سوم استفاده از یکی از وقفه های 21H که مستقیماً این مقدار را به ما می‌دهد و حافظه را release می‌کند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:

Year. Month. Date. ( )

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

تدریس یار اسمبلی:

CO_processor : علاوه بر پردازشگرهایی که وجود دارند از پردازشگرهای کمکی استفاده می کنیم که فقط عملیات ریاضی را انجام می دهد و وقت زمان انجام محاسبات را کم می کند.

General purpose Registers:

An : برای نگه داشتن نتیجه محاسبات از آن استفاده می کنیم یا زمانی که محاسبات داخل stack را انجام می دهیم در هر مرحله pop کردن نتیجه درون این قرار می گیرد.

Bn : اشاره کننده به داده درون data segment است.

Cn : شمارنده در حلقه های تکرار شونده و برای string ها است.

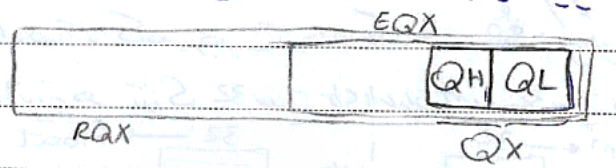
Dn : اشاره کننده برای I/O است.

BP : اشاره کننده به داده مورد نظر درون stack است.

Registers in 64-bit Mode:

همان رجیسترهای 16 بیتی به رجیسترهای 64 بیتی تبدیل شده و نام آن ها به RAX و RBX و RCX و RDX تغییر می یابد و ۸ رجیستر دیگر نیز اضافه شده (R8-R15)

Rxn



Segment Registers in 64-bit Mode:

در این حالت CPU در سطح بالاتری عمل می کند و جلوگیری از تداخل اطلاعات درون حافظه از طریق کامپایلر کنترل می شود در نتیجه خطای آن از خطای انسان بسیار کم تر است. در این مد نیز رجیسترهای CS، DS، ES و SS وجود دارند ولی مقدار آن ها صفر است و توجهی به این رجیسترها ندارد و برای انجام محاسبات تنها از رجیسترهای FS و GS استفاده می کند که از 32-bit اضافه شده.

Flag register در این mode، 64 بیتی است با این تفاوت که 32 بیت بالا ارزش ترآن رزرو شده است و 32 بیت کم ارزش تر همانند FR درون ۳۲ بیتی می باشد.

www.jozveha.ir

اگر رجیستر در برنامه نداشته باشیم مجبوریم از حافظه استفاده کنیم که بسیار زمان گیر است. برای crack کردن برنامه به کد اسمبلی آن نیاز داریم. برای نوشتن برنامه در سیستم های کم و کم حافظه بهترین کار این است که ما قسمتی از برنامه را با C++ و قسمتی از آن که حافظه و کلاردار دارد با اسمبلی بنویسیم. سرعت پردازنده موبایل کم و حافظه آن بسیار محدود است پس بهتر است برای آن با کد اسمبلی برنامه بنویسیم.

Subject:
Year. Month. Date.

Arithmatic Logic Unit (ALU):

عملیات منطقی و ریاضی زیر را انجام می دهد:

۱- عملیات ریاضی بر روی اعداد باینری (ADD ، SUB ، DIV ، MUL ، INC ، ...)

۲- عملیات ریاضی بر روی اعداد Decimal (به صورت BCD)

۳- عملیات منطقی

۴- شیفت دادن

۵- محاسبات بر روی بایت و بیت (مثلاً یک بیت از بایت را ضرب در عدد هشت می کند)

ماشین ها به دو دسته زیر تقسیم می شوند:

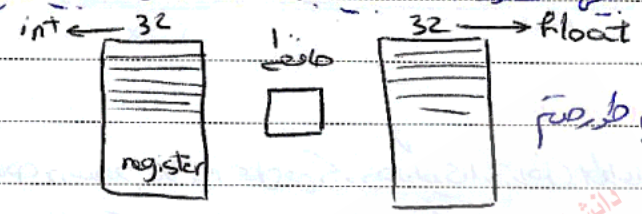
۱- stack Mashine: می خواهیم ماشینی داشته باشیم که دستورات را برای ما ذخیره کند ولی نیازی به داشتن آدرس ها نداشته باشد.

$(112+4)*(4+2))^2$ ⟶ push کردن درون stack

نیازی به داشتن آدرس نداریم.

۲- register Mashine: نوعی از ماشین است که تعداد زیادی رجیستر دارد ولی همه آن ها با همدیگر یکسان هستند ($R0 = $R32 رجیستر شماره صفر با رجیستر 32 فرقی ندارد).

نوع مشابه این ها پردازنده های MIPS هستند که همه رجیسترهای آن به جز $R0 با یکدیگر برابرند و کاربرد یکسان دارند. نوعی از این پردازنده ها ۶۴ بیتی هستند که 32 بیت برای اعداد int و 32 بیت دیگر برای اعداد float استفاده می شود.

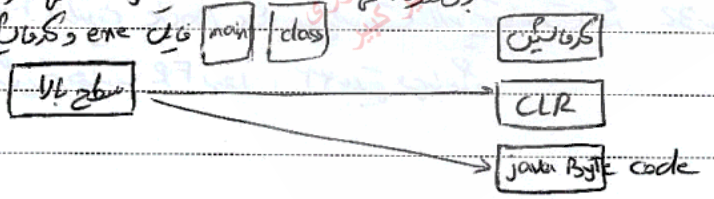


اینتل بین این دو نوع ماشین قرار دارد و به طور جدی ویژگی هیچ یک از آنها را ندارد.

مثال: برای نوشتن یک driver می توان از دو راه استفاده کرد: ۱- Visual Basic user Mode

که فایل های (com) را شامل می شوند. ۲- kernel Mode

برای هر کلاسی که header دارد یک .obj می سازد و در نهایت به فایل exe و کد ماشین تبدیل می شود



component object Modeling

Visual Basic, .net

مستقیماً با استفاده از کامپایلر .net اجرا می شود

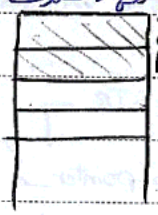
java ⟶ javac.exe ⟶ Byte code

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

نکته: زمانی که در Real Mode هم پوشانی segment ها اتفاق می افتد نمی توانیم برنامه های multitask داشته باشیم زیرا ممکن است داده ها و کدهای دو برنامه بر روی هم بیفتند. ولی حافظه آدرس را مصرف می کند.



ممکن است در طول برنامه داده تعریف کنیا آرایه بکشید درستیم ← دو داده بر روی هم می افتند.

برای مقابله با این مشکل ~~[illegible]~~ ، overwrite کردن interrupt های Bios توانستند در هنگام اجرای یک برنامه همزمان مثلاً ماشین حساب نیز فعال شود.

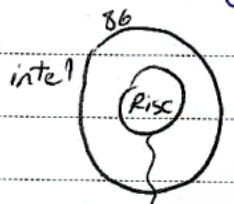
Floating point: استانداری برای IEEE 754 برای نمایش اعداد double و float را پیشنهاد استفاده می کنیم

پردازنده های CISC , RISC:

CISC (complex instruction set computer): پردازنده ها را پیچیده کردند و دستورات را پیچیده تر قابل محاسبه است ولی زمان بیشتری صرف می شود (AMD , intel)

RISC (Reduce instruction set computer): برای هر دستور cpu ها را چیده شده ولی دستورات ساده تر را می تواند محاسبه کند (مثلاً عملیات ضرب را نمی تواند انجام دهد.) در این مدل می توانیم pipeline را اجرا کند. (عملیات ریز ترند)

با وجود اینکه ساختار intel از نوع cisc است ولی با استفاده از سیستم وداشتن هسته RISC سرعت عملیات چندین برابر می شود.



از یک هسته RISC استفاده می کند در حالیکه خودش CISC است

نکته: قسمت اصلی برنامه با برچسب label ، Main شروع شود و در پایان بعد از ENDS باید این label بسته شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

: 21H interrupt

interrupt های مربوط به سیستم عامل DoS هستند

LEA → آدرس پارامتر دوم در دون پارامتر اول قرار می‌دهد.

LEA dx, MySTR

یک pointer در ابتدای label مربوطه قرار می‌دهد.

برای فراخوانی یک interrupt به عنوان پارامتر ورودی باید مشخص کنیم که می‌خواهیم interrupt مربوط چه کاری برای ما انجام دهد که در اسمبلی این پارامتر توسط یک عدد مشخص می‌شود.

برای فراخوانی یک ← Mov AH, 9

```
DSEG Segment public para 'data'
    MySTR DB "Hello world!!", 10, 13, "Assembly is the coolest$"
DSEG ENDS
```

10: ابتدای خط
13: خط بعدی → با این اعلان به اول خط بعدی می‌رود.
carriage return
1. line feed (LF)
2. new line ('\n')

```
CSEG Segment public para 'code'
    Assume Ds: Dseg, CS: Cseg, ES: Nothing, SS: nothing

Main:
    Mov Ax, DSEG
    Mov ds, Ax

load effective address
    LEA dx, MYSTR
    Mov AH, 9
    INT 21H
```

→ چاپ یک رشته بر روی صفحه

```
    Mov Ax, 4C00h
    int 21 H
CSEG ENDS
END main
```

→ این عدد مشخص می‌کند که interrupt از خانواده 21H برای خارج شدن از برنامه است.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

0XCD → آپکود مربوط به اینترایپت است.

در مقابل این نماد یک بایت قرار می‌گیرد که نشان می‌دهد کدام interrupt اتفاق بیفتد.

0XEA /9 → برای far jump به کار می‌رود.

EA [seg][offset]

دو بایتی دو بایتی

چون مبدأ است نمی‌توانیم از DI استفاده کنیم.

b = A[4] →	Mov SI, 4	i = 2	mov di, 2
	LEA BX, A	b[i] = 9	mov dx, a
	Mov AX, [BX+SI]		Lea bx, b
	LEA SI, b		mov DS:[BX+Di], dx
	Mov [SI], AX		

: Operating Modes

۱- real-address mode : یک seg register و یک offset دارد که آدرس حقیقی در حافظه از طریق seg×16 + offset بدست می‌آید با این روش تنها حدود 1M را می‌توان آدرس‌دهی کرد.

۲- protected mode : mode پیش‌فرض سیستم عامل cpu protected m. است البته با روشن شدن کامپیوتر در حالت real mode فعال می‌شود و سپس از طریق یک interrupt به حالت protected می‌رود.

۳- system management mode (SMM) : با پردازنده‌های intel 386 وارد شد و در mouse و اقسام آن استفاده می‌شود. برای سیستم عامل با عملکرد خاص مناسب است.

۴- compatibility mode : مناسب برای کارهای 16 و 32 بیتی است ولی هدر زمانی نوشته شده در قسمت 16 و 32 بیتی در این حالت قابل اجرا نیست. برای سیستم عامل با عملکرد خاص مناسب است. در هر سطح CPU (4 level) برای دسترسی معرفی می‌شود که عبارت اند از:

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

kernel: دل وجود سیستم عامل است و امکان دسترسی به همه بخش ها را دارد.
سروی آن در level دیگر قرار می گیرد که device driver است و برای دسترسی driver هاست. آخرین و بیرونی ترین level با نام user mode است که کمترین دسترسی را دارد. که براساس این level بندی ها می توانیم مشخص کنیم به کدام قسمت ها دسترسی داشته باشد.

به این سطح بندی privilege level می گویند که در compatibility mode همه این level ها support می شوند. همانند protected mode امکان دسترسی به 4GB از حافظه در یک برنامه وجود دارد. ۴ سطح دسترسی وجود دارد در protected mode و compatibility
??? PAE

64-bit mode:

سایز پیش فرض برای آدرس ها 64 بیت و برای داده ها 32 بیت است. در این حالت و حالت compatibility می توان به گونه ای برنامه نویسی کرد که هر کدام در mode مربوط به خود (32 یا 64 بیتی) اجرا شوند.

Memory Management modes:

1- Flat memory: در این حالت CPU می تواند به صورت خطی به همه قسمت های حافظه دسترسی داشته باشد. در این segmentation mode و paging وجود ندارد. علت استفاده از این mode به خاطر این است که در حالت paging برای دسترسی به یک داده دقیق است و مجبور باشیم چندین بار به memory مراجعه کنیم و این سرعت CPU را به شدت کاهش می دهد برای رفع این مشکل از Flat mode استفاده می کنند.

از مزایای آن مکانیزم ساده آن است که می تواند در همه پردازنده ها وجود داشته باشد و دسترسی سریع به داده ها است. ولی برای سیستم های single tasking مناسب است ولی در windows با وجود multi tasking از آن استفاده می کنند.

زیرا در این روش بظاهر برنامه ها از Flat memory استفاده می کنند ولی شیوه اصلی به غیر از این روش اتفاق می افتد.

دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

غالباً تمام پردازنده‌ها Flat model را پیاده‌سازی می‌کنند به جز 8086 که استثنا است.
در این حالت فقط یک seg برای کد و داده وجود دارد.
Similar to Dos (Disk operating System) com file ??

۲- Real_Addressing Mode: (segmented model)

یک مدل از حافظه برای intel 8086 است و طول هر آدرس برابر ۲۰ بیت است
ولی چون ۲۰ بیت در یک رجیستر جا نمی‌گیرد از دو رجیستر ۱۶ بیتی استفاده می‌کنیم و آدرس
از ترکیب زیر بدست می‌آید.

تقسیم بندی

Address = 16 x Selector + offset

از معایب آن این است که هر خانه از حافظه به یک صورت واحد آدرس‌دهی نمی‌شود.
اگر برنامه بیش از 64kB باشد باید به صورت دو قسمت اجرا شود

۳- protected Mode (segmented Model)

دوم

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:

Year. Month. Date. ( )

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

PAPCO

www.jozveha.ir

Subject:
Year. Month. Date. ( )

File System :

روشی است که بوسیله آن فایل ها را داخل هارد، فلش یا دیسک ذخیره و سازمان دهی می کنند و محتوای آن نیز ذخیره می شود.

File System ها به ۶ دسته تقسیم می شود.

۱- Disk file system : مربوط به ذخیره سازی فایل ها در هارد می باشد که به سیستم عامل ها می گویند چگونه فایل ها را روی disck ذخیره، بازیابی و سازمان دهی کند. مثل NTFS، FAT (12، 16، 32)

که خود به دو دسته تقسیم می شوند: journaling file system_1 versioning file system_2

ext* ← برای Linux, unix کاربرد دارد.

FAT : در این روش جدولی به اسم file allocation table در ویندوز و I know در لینوکس وجود دارد. هر فایل از مجموع بایت ها تشکیل می شود. این بایت ها به گروه ها و دسته هایی تقسیم می شوند که به آن ها sector می گویند. معمولاً هر sector شامل ۵۱۲ بایت است. این sector ها با یکدیگر ادغام می شوند و cluster ها را بوجود می آورند. cluster واحدی است که بایت ها براساس آن مدیریت می شود و براساس اینکه فایل سیستم از نوع FAT ۱۲، ۱۶ یا ۳۲ بیتی باشد جدول مربوط دارای تعداد بایت های مشخص است. مثلاً اگر ۱۶ بیتی باشد این جدول 2^{16} خانه دارد.

از مجموع cluster ها فایل بوجود می آید و ویندوز نیز با همین cluster ها کار دارد.

byte → sector → cluster → file

اگر cluster ها بصورت link list به هم متصل شده باشند و آدرس اولین cluster را داشته باشیم می توانیم به file دسترسی یابیم که در جدول مربوط شماره cluster ها قرار دارد. به کل این روش FAT می گویند. هر فایل یک جدول ویژه خود را دارد که آدرس cluster های آن مشخص شده است و هر cluster بعدی خود را نشان می دهد.

۲- Flash memory file system :

روشی برای سازماندهی اطلاعات در Flash memory است. علت اصلی جدا کردن فایل سیستم hard و Flash memory به صورت زیر است:

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شریف کبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

۱- ساختار پاک کردن در این دو با هم فرق می کند. در hard disk برای پاک کردن شماره و آدرس اولین cluster آن به چند دیگری تغییر می دهد ولی در Flash memory باید block مربوطه به طور کامل پاک شود. به خاطر همین پاک کردن از روی hard بسیار سریعتر از پاک کردن صورت می گیرد.

در گذشته نمی توانستیم فایل های FAT را بر روی لینوکس باز کنیم ولی الان NFAT این کار را انجام می دهد. در کل دو معماری برای سیستم عامل وجود دارد:

۱- micro kernel memory: سیستم عامل را به بخش های کوچک تری تقسیم می کند و windows به ظاهر چنین روشی را دارد. بسیار سخت تر از روش بعدی است.

۲- monolotic memory: linux و unix و solaris که در واقع به روش اول هستند ولی ادعا کردند که monolotic می باشند و به سوی روش micro پیش می روند.

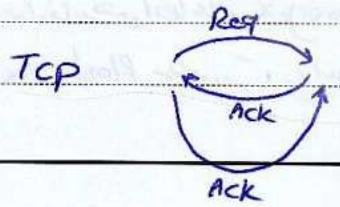
۲- seeking latency: چون حجم هارد ها تقریباً ۱۰ تا ۲۰ برابر حجم فلش ها هستند موضوع جستجوی زمان بسیار وابسته به این حجم است و در این دو تفاوت و پیچیده است.

۳- wear out: فایل سیستم هایی که برای Flash memory استفاده می شوند به صورت دیگری است که به آن log - structure file system می گویند مثل JFFS

۴- Transactional file system:

در بحث database بحثی به اسم Transaction processing است که هنگام انتقال داده بین دو چیز کاربرد دارد و برای جلوگیری از از دست رفتن اطلاعات به کار می رود. یعنی درخواست داده می شود و در ازای آن از دستگاه مقابل Acknowlegment داده می شود که درخواست را گرفتیم و با یک Acknowlage که داده Acknowlage گرفته شد و بعد از آن داده برداشته می شود

اما در روش database از همان اول داده از مقصد برداشته می شود در نتیجه اگر داده به مقصد نرسد از بین می رود.



www.jozveha.ir

Subject:
Year. Month. Date. ( )

4_ Database file system :

- Data model : توصیف می‌کند که داده‌ها چه هستند و چگونه باید سازمان‌دهی شوند.
- meta data : اطلاعاتی درمورد اطلاعات که به آن data dictionary گفته می‌شود.

Data model
- ER diagram
- Relational model
- object_Relational model

تبدیل ← توسط ORM

5_ Network file system :

برای انتقال داده‌ها بر روی شبکه مثل FTP یا webdav باید بر روی client ها فایل سیستمی وجود داشته باشد تا این‌ها را ترجمه کند که این فایل سیستم‌ها به صورت زیر است

- SMB → microsoft
- NFS → Sun

فایل سیستم‌های شبکه قواعدی هستند که در انتقال اطلاعات در شبکه کاربرد دارند.

6_ special_purpose file sys :

زمانی است که نمی‌خواهیم از هیچ یک از فایل سیستم‌های استفاده کنیم و خود به دلخواه انتخاب می‌کنیم.

معروف‌ترین آن‌ها:

GFS : google file system یا procfs

نوعی سیستم عامل برای نمایش دادن process های سیستم عامل به کار می‌رود

زمانی که از search engine مربوط به google استفاده می‌کنیم از نوع data base نیست

در گذشته component مربوط به file management جدا بوده است یعنی به صورت جدا از سیستم عامل load می‌شد و بر روی hard بود و هر وقت که احتیاج داشتیم load می‌شد که مربوط به سیستم عامل را انجام می‌داد و store می‌شد. این روش مناسب نبود و file system بوجود آمد که خود سیستم عامل

www.jozveha.ir

Subject:
Year. Month. Date. ( )

سازمان‌دهی فایل‌ها را انجام می‌دهد. در نتیجه باید interface ی بین ما و سیستم عامل وجود داشته باشد زیرا ما از فایل سیستم اطلاعاتی نداریم. این interface ها به دو دسته تقسیم می‌شوند:

- command line : shell = linux shell, unix shell, open VMS DCL, ...
- graphical → file browser

فایل‌ها به دو صورت می‌توانند ذخیره شوند:

۱- Flat : همه یک root داریم و همه فایل هستند. اولین سیستم عامل‌ها به این صورت هستند ولی این روش مشکل است زیرا اگر تعداد فایل‌ها زیاد شود ذخیره‌سازی دشوار می‌شود.

۲- hierachial (سلسله مراتبی): حالت پیشرفته‌تری نسبت به Flat ها هستند و شامل سلسله مراتب گونا گون می‌شوند.

: Format

زمانی که درایوی را format می‌کنیم، جدول قبلی پاک می‌شود و جدید ساخته می‌شود.

windows → map: همان فضای mount در لینوکس در windows می‌گویند.

linux → mount: باید filesystem مورد نظر را به سیستم عامل بفهمانیم و نشان بدهیم.

این کار در لینوکس به صورت خودکار انجام می‌شود. نقطه‌ای که در لینوکس به سیستم عامل اعلام می‌کنید که فایل سیستم را از روی آن بخواند mounting point می‌گوییم.

: boot stap مراحل و کارهایی که انجام می‌شود تا سیستم عامل بالا بیاید.

: boot loader اگر چندین سیستم عامل وجود داشته باشد از بین آن‌ها امکان انتخاب یکی را می‌دهد.

bootloader (در لینوکس)
- grub ← پروژه اختیاری
- LILO
- ELILO

باید با فایل سیستم‌های (FAT ، NTFS ، EXT) کار کند.
- آدرس آدرس را نشان می‌دهد.
- آخرین آدرس را نشان می‌دهد.

(سیستم [redacted] اینتل)

دانشکده مهندسی عمران دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

الگوریتم insertion sort :

اولین عنصر را کوچکترین عنصر در آرایه و عناصر تا انتها با عناصر قبلی مقایسه می‌شود و در صورت کوچکتر بودن جای آنها عوض می‌شود. در واقع لیست را تا عنصر i ام sort می‌کند و بعد عناصر بعدی را با عنصر i ام در سمت چپ. فرق این روش با bobblesort این است که در bobblesort این عنصر با همه عناصر قبلی چک می‌شود ولی در این روش چون تا قبل از عنصر i ام sort شده است از i به اول جلو می‌رویم تا جای اصلی را پیدا کنیم.

```
for( int i=1 ; i<n ; i++){
int j=i-1;
int r=A[i];
while ((j>=0) & A[j]>r) {
    A[j+1]=A[j];
    j--;
}
A[j]=r;
}
```

6, 3, 2, 5, 4, 1, 3, 2

r = 3: 6, 6, 2, 5, 4, 1, 3, 2

r 3, 6, 2, 5, 4, 1, 3, 2

r = 2: 3, 6, 6, 5, 4, 1, 3, 2

3, 3, 6, 5, 4, 1, 3, 2

r 2, 3, 6, 5, 4, 1, 3, 2

الگوریتم merge sort :

براساس روش تقسیم و دستیابی است عناصر ۰ تا q ، sort شده‌اند و عناصر q تا n-1 نیز sort می‌شود. عناصر اول L_1 و L_2 را مقایسه می‌کنیم و عنصر کوچکتر را به آرایه p اضافه می‌کنیم. آرایه‌ای که عنصر اول آن استفاده شده بود آن عنصر را حذف می‌کنیم و این الگوریتم را برای این آرایه جدید به صورت بازگشتی فراخوانی می‌کنیم.

p ⇐ ابتدای آرایه (اندیس اولین عنصر آرایه)

A ⇐ اندیس آرایه

r ⇐ انتهای آرایه ⇐ اندیس آخرین عنصر آرایه

یکی از سوالات merge s. این است که الزاماً باید از آرایه کمکی استفاده کنیم.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات
دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

```
merge Sort (A, p, r) {
  if (p<r) {
    q = (p+r)/2;
    mergeSort (A, p, q);
    mergeSort (A, q+1, r);
    merge (A, p, r, q);
  }
}
```

دو آرایه Sort شده به ما می‌دهد.

```
merge (A, p, r, q) {
  int L[];
  n1 = q-p+1;
  n2 = r-q;
  L = new int [n1+n2];
  int k=0, i=0, j=0;
  while ((i<n1) & (j<n2))
  {
    if (A[p+i] < A[q+j])
    { L[k] = A[p+i];
      k++;  i++;
    }
    if (A[p+i] > A[q+j])
    { L[k] = A[q+j];
      k++;  j++;
    }
    if (A[p+i] == A[q+j]) {
      L[k] = A[p+i];
      k++;  i++;
      L[k] = A[q+j];
      k++;  j++;
    }
```

آرایه کلی سه‌شکل داده شد.

```
while (i<n1) {
  L[k] = A[p+i];
  k++;  i++;
}
```

دو عنصر مساوی باید هر دو وارد شوند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

char = 1 byte → ASCii
= 2 byte → UTF

```
val1  DB  68H
val2  DB  3FH
mov   AL, val1
      68H
mov   BL, val2
      3FH
AND   AL, B6H
      20H
cmp   AL, BL
JA    label1
mov   AL, BL
jmp   exit
label1:
  mov   BL, AL
exit:
```

AL 0110 1000
10 1110 110
10 0000 1110

AL = BL = 3F

تمرین:
برنامه‌ای که کلاسیم پایه ۲ یک عدد مثبت را تعیین کند.

مثال:

```
void main {
int a = 8;
char c = getch();
if (c == 'a') {
    a = 10;
} else {
    a = 20;
}
cout >> a;
}
```

```
        mov   AH, 01
        int   21
        cmp   AL, 'a'
        JZ    Label
        mov   a, 6
        jmp   print
label:  mov   a, 5
print:  mov   AH, 09H
        ADD   a, 48
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

تمرین: فقط با اعمال شیفت محاسبه کنید $12 * AX + 7 * BX - 3$

تمرین: برنامه‌ای بنویسید که عددی مثل N را از ورودی خوانده تمام اعداد کامل قبل از آن را چاپ کند.
(عدد کامل عددی است که مجموع تمام مقسوم علیه‌هایش به غیر از خودش برابر خودش باشد.)

تمرین:

```
val1    db    35H
val2    dB    3FH
mov     AL, val2
mov     BL, val1
nor     BL, 0FFH
test    AL, 3
JZ      label1
mov     AL, 1
JMP     exit
label:
mov     BL, 1
exit:
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

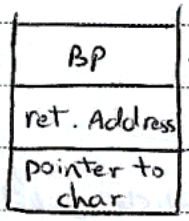
SP ← top stack را نشان می‌دهد.

BP ← برای استفاده از پارامترهایی که به تابع پاس داده می‌شود به کار می‌رود و نیازی به back up گرفتن نداریم.

نکته ۱: برای درست اجرا شدن برنامه و دسترسی به داده‌های درون data segment باید حتماً در اول codsg مقادیر seg ها با Assume کنیم در غیر این صورت با mov کردن Data به DS درست کار نمی‌کند.

نکته ۲: زمانی که مقداری درون stack ، push می‌شود و می‌خواهیم آن را pop کنیم باید مواظب باشیم که صدا زدن یک proc نیز آدرس بازگشت را درون stack قرار می‌دهد پس برای pop کردن داده مورد نظر باید از این آدرس عبور کنیم.

نکته ۳: می‌توان به جای pop کردن آدرسی که بر روی داده قرار گرفته از BP استفاده کنیم به این صورت که با نگه داشتن مقدار BP در stack آن را به محل sp انتقال دهیم و با محاسبه خانه مربوطه مقدار آن را در متغیر بریزیم بدون اینکه pop ی صورت گیرد و در نهایت مقدار BP را به اولیه بازگردانیم.



← BP

SS:[BP] ← به این خانه اشاره دارد.

SS:[BP+2] ←

SS:[BP+4]

```
_start:
        call  read param
        mov   CHAR_READ, al

        Lea   ax, CHAR_READ
        push  ax
        call  PrintNumber
```

دانشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

```
printNumber PROC NEAR
    push BP
    MOV BP, SP

    LEA dx, MSG
    MOV AH, 9
    INT 21H

    MOV SI, [BP+4]
    MOV DI, [SI]
    MOV AH, 2
    INT 21H

    POP BP
    ret 2
ENDP
```

به مقداری که از ورودی گرفته شده از طریق این دستور دسترسی داریم

مقدار گرفته شده از ورودی چاپ می‌شود

BP را به مقدار اولیه خود که انتهای stack است برمی‌گردانیم

به تعداد عدد مربوط علاوه بر برداشتن آدرس بازگشت proc به این تعداد بایت از stack pop می‌کند. با این کار داده‌ی قرار گرفته که pop شده از stack برداشته می‌شود.

یکی از کاربردهای این مکانیزم در جمع دو عدد است که قبل از صدا زدن proc مربوط به خواندن اعداد از ورودی و جمع آنها حافظه‌های خالی را درون stack قرار می‌دهد و حاصل را درون آنها push می‌کند. تنها نکته در آن مشخص آدرس خانه‌ی مربوط در stack و تعداد بایت بیرون ریخته شده توسط ret است.

(برنامه‌ی مورد نظر 5- Simple procedure \ 8.asm)

$$\begin{cases} \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} & \leftarrow \text{recursive step} \\ \binom{a}{a} = \binom{a}{0} = 1 & \leftarrow \text{Base (پایه)} \end{cases}$$

دانشکده مهندسی کامپیوتر - دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:
Year. Month. Date. ( )

برای برگرداندن مقدار می‌توان از shared memory استفاده کرد که برای مودی مناسب است که تابع فقط یک بار احضار صدا زده شود (ساختار سری داریم) ولی برای ساختار درختی مناسب نیست

mov BP, SP → BP را به جایی می‌برد که SP در آن قرار دارد.

برای اینکه اطلاعات را از دست ندهیم می‌توانیم از آنها back up گرفته و در جایی مانند stack ذخیره کنیم.

نکته: می‌توان segment های مورد نظر را به صورت زیر overwrite کنیم:

MOV ES:[BX], AX

↓

یعنی مقدار داخل BX را از extra seg حساب و صرف کند.

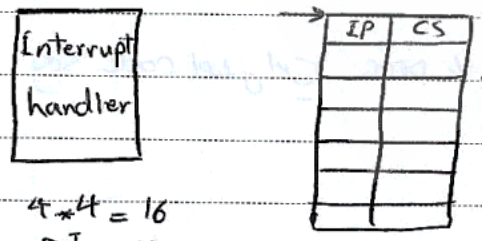
prefix:

علاوه بر prefix های موجود زمانی که دستوری به فرم زیر نوشته شود زبان اسمبلی آن را تغییر می‌دهد.

mov Ax, ES:[Bx] ⟶ ES:
mov Ax, [Bx]

در این جا به ES یک prefix می‌گوییم.

برای ذخیره آدرس یک interrupt service در interrupt vector می‌توان با داشتن آدرس ابتدای جدول وقفه‌ها و آدرس خانه مربوط به این interrupt، IP و CS را ذخیره می‌کنیم.



جدول مربوط را درون DS قرار می‌دهیم.

4*4 = 16
DI ← 16

DS:[BX+DI+0] → IP
+1
+2 → CS
+3

www.jozveha.ir

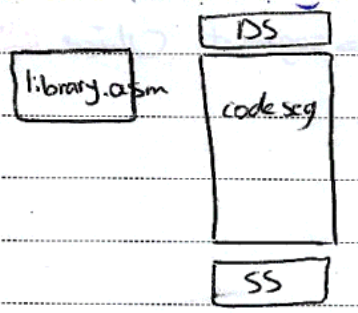
* تمرین ① proc Far که پارامتر ورودی هم داشته باشد. ② این دو فایل چگونه درون حافظه قرار می‌گیرد.
bat, lib.asm, call.asm

Subject:
Year. Month. Date. ( )

باید interrupt vec. را درون DS قرار دهیم تا بتوانیم به آن دسترسی پیدا کنیم برای اینکار آدرس ابتدای vector به صورت ثابت وجود دارد و از روی آدرس می‌توانیم seg و off آن را در بیاوریم و seg را برابر DS قرار دهیم و مقدار قبلی DS را درون stack نگه داریم.

مثال: استفاده از lib و Far proc برای اتصال دو فایل به یکدیگر

در این جا یک برنامه کوچک را به عنوان library تعریف می‌کنیم و از آن در یک برنامه بزرگتر با استفاده از Jmp Far دسترسی می‌یابیم.




```
public s
codesg segment page public 'code'
    s proc FAR
    mov bp, sp
    mov AH, 9
    int 21H
    s ENDP
codesg ENDS
END
```

احتیاجی به assume کردن نداریم و برای استفاده از آن در کلاس دیگری باید این proc را از نوع public تعریف کنیم.

public → EXT RN

همواره یکدیگر به کار می‌روند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیر کبیر

باید در code seg اصلی این proc را extrn کنیم.

```
EXTRN    S: [type]
```

Far / near

باید در seg دیگر از آن استفاده کنیم.

www.jozveha.ir

Subject: .lib ⇝ static library / در زمان compile به برنامه متصل می‌شود.
.dll ⇝ dynamic library در زمان runtime هر کدام که مورد نیاز باشد را load می‌کند

Year. Month. Date. ( )

نکته: از includeها برای اضافه کردن library های داخلی خود برنامه به کار می‌رود و زمانی که می‌خواهیم این دو فایل را به یکدیگر متصل کنیم بنویسیم tlink و نوشتن نام هر دو فایل exe و شروع ساخته می‌شود که نام فایل exe برابر با نام اولین فایلی است می‌نویسیم

```
tlink.exe test.obj lib.obj
```

می‌توان ترکیبی از یک فایل اسمبلی و زبان سطح بالا داشت باشیم برای اینکار باید فایل asm را درون کد extern "C" کنیم.

```
extern "C" void clear();
```

در انتهای نام تابع یک underline اضافه می‌کند که باید در هنگام لینک کردن این علامت حتماً باشد.

EXAMPLE:

```
#include <iostream>
using namespace std;
extern "C" void clear();
int main(){
  clear();
  return 0;
}
```

در این جا clear یک proc به زبان اسمبلی است که درون یک فایل asm در source files ذخیره شده است.
در این جا دستورات 32 بیتی نوشته شده اند (9_Hybrid code)

برای اجرا شدن برنامه باید تنظیمات مربوط به اسمبل شدن فایل asm را انجام دهیم زیرا لینکر تنها دو کد را به هم متصل می‌کند در حالی که هنوز هیچ obj file ی از این ساخته نشده است. به صورت زیر عمل می‌کنیم:

clear.asm → properties → Custom build step → general → command Line

→ C:\ML\ml /c /Cx /coff "$(inputpath)"
سیدیافتن فایل در حین شکوفایی در فایل جایی می‌گردد

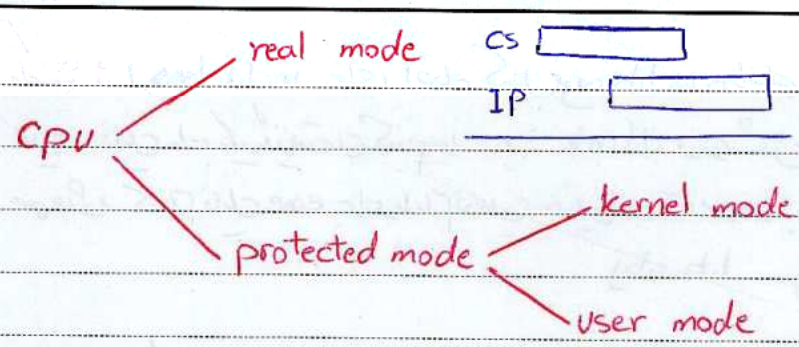
outputs → $(InputName).obj
نامی که فایل با آن ذخیره شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شریف

www.jozveha.ir

تمرین: این فایل ها چه هستند چه فرقی با هم دارند obj و elf و coff

Subject:
Year. Month. Date. ( )

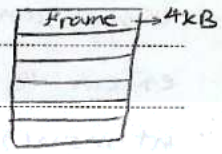


protected mode: دستورات در کامپیوتر previlage هستند که باید توسط برنامه‌های خاصی اجرا شوند مانند سیستم عامل که در حالت real mode هیچ حفاظتی نیست مانند برنامه‌هایی که در C یا pascal نوشته می‌شوند (به راحتی به interrupt vector دسترسی دارند.) ولی در protected m. اینگونه نیست. در حالت kernel mode تنها برنامه‌های خاصی دسترسی دارند ولی در user mode برنامه‌های معمولی وجود دارد.

Appl $\xrightarrow{\text{request}}$ OS $\longrightarrow$ port $\rightsquigarrow$ برمی port مربوط رفته و دستور را می‌گیرد.

در protected mode امکانی برای paging وجود دارد.

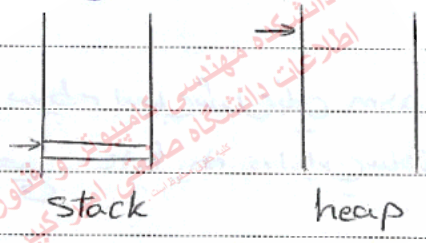
$\rightsquigarrow$ زمانی که در java برنامه نوشته می‌شود به صورت مجازی به این حافظه 4 GB دسترسی دارد ولی در نهایت به صورت فیزیکی به frame تبدیل شده. و به نظر می‌رسد که تنها برنامه در حال اجرا برنامه ماست. virtual address memory



8 Garbage collection

زمانی که یک object پاک می‌شود در اینجا قرار می‌گیرد

```
customer c;
c=new customer();
```



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

stack از آدرس های پایین شروع می‌کند و به بالا می‌آید ولی heap از بالا به پایین شروع می‌شود. هنگام new کردن یک object آدرس آن را در stack قرار می‌دهد و خود object را در heap قرار می‌دهد.

www.jozveha.ir

Subject:
Year. Month. Date. ( )

```
class Temp {
    void sample() {
    }
}
```

```
void temp::sample() {
    }
```

مانند فاکتور است و به ازای هر بار new کردن کلاس یک بار فضا در حافظه قرار می‌گیرد

تنها یک بار در حافظه heap قرار می‌گیرد مانند procudure رفتار می‌کند.

API :

هر سیستم عامل شامل دو بخش kernel و shell است که kernel قابل دسترسی نیست و کلیه اعمال سیستم عامل را آن انجام می‌دهد ولی shell قابل دسترسی است و هر کاری که می‌خواهد انجام دهد را باید به kernel درخواست دهد که به آن system call می‌گویند.

هر system call در windows را API می‌گویند. این API ها واسط به سیستم عامل هستند به مجموعه این کتابخانه‌ها API می‌گویند. می‌توان API ها را به چندین دسته تقسیم کرد:

1) base access : به قسمت های حساس سیستم مانند thread ، processes و file system دسترسی دارد.

2) Advanced service : مانند دسترسی به windows registery که یک سیستم خوشه‌ای و پیچیده است.

3) graphical device int : تمام اجزای گرافیکی از جمله bottom ها و window ها API هستند. تمام dialog Box مانند API هستند.

4) control library : تمام menubar ها و ابزارهای کنترلی را شامل می‌شود.

5) windows shell : شامل sucket هاست که راه دسترسی به شبکه است.

6) web : خدماتی که internet explorer ارائه می‌دهد.

7) multimedia : مانند MCI که اجازه می‌دهد به هر وسیله multimedia دسترسی داشته باشد.

(بدون اهمیت نوع آن)

دانشکده فنی کامپیوتر شهرکی
اطلاعیه دانشگاه صنعتی شریف

www.jozveha.ir

Subject:
Year. Month. Date. ( )

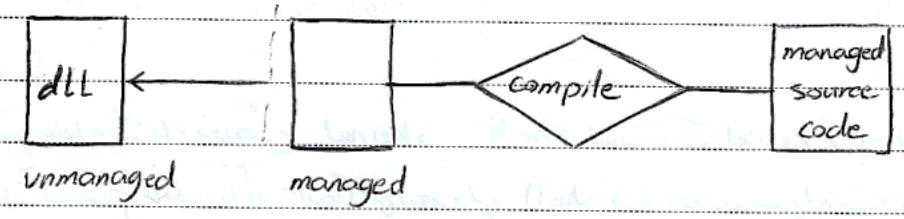
kernel32.dll
User32.dll
Gdi32.dll
} → شامل همه ی APIهای windows است.

13. wrapper library: به جای APIها یک سری library ایجاد کرده که به جای آنها استفاده می شود و کار را راحت تر می کند. COM یک wrapper library است.

```
Socket S=new Socket();
```

یک lib wrapper برای دلفی به شبکه است.

APIها درون فایل هایی به نام dll قرار دارد که برای استفاده از آنها باید پارامترهای ورودی آن را بدهیم تا برای استفاده در برنامه آن را از حالت unmanaged به managed تبدیل می کنیم.




```
Using System.Runtime.InteropServices;
class API{
    [dllImport("kernel32.dll", EntryPoint="CreatFile", charset=charSet.unicode)]
    public static extern SafeFileHandle Creat(
```

نام فایل dll شامل API مورد نظر

نام API مورد استفاده

برای استفاده از کاراکترهای فارسی

خارج از کلاس قابل استفاده باشد.

نوع خروجی متد

نام prototype مورد نظر که در صورت هم نام بودن با خود API نیاز به نوشتن entrypoint نیست.

دلیل ۵: برنامه های دیگر managed به زبان CLR تبدیل می شود که زبان اسمبلی اصلاح شده است و ویژگی آن این است که متناسب با سیستم عامل بروی کامپیوتر نصب می شود و مختص ماشین خاصی نیستند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

در #C نمی‌توان از pointer استفاده کرد و همه چیز call by reference است که در استفاده از pointer به حالت unsafe می‌رود و از یک جا به بعد خود compiler نیز محدودیت هایی را برای unsafe ایجاد می‌کند.

Subject:
Year. Month. Date. ( )

فایل های DLL:

دسته‌ای از Function ها که ویژگی های مشترکی دارند درون یک فایل باسیوند dll ذخیره می‌شود که قابل فهم برای سیستم عامل است. به هر یک از Function های موجود در آن API می‌گویند. هر فایل dll شامل جدولی است که به آن dll export table می‌گویند و به هر Function عددی را اختصاص می‌دهد، که برای دسترسی به هر Function از دو طریق می‌توان دسترسی داشت:

عدد	Function

۱- عدد مربوط به آن که به آن ordinal می‌گویند.

۲- از طریق نام آن Function

entry point = "#123"

→ مکانیزم آن به این سادگی نیست و در واقع یک binary tree است.

MAssageBoxA → ANSII (۱ بایت)

MAssageBox W → UNICODE (۲ بایت)

با قرار دادن A و W در ادامه API Funct. ها به دو حالت ANSII و Unicode می‌رود.

هر dll File شامل دو section به نام‌های ۱- code section و ۲- data section است که ویژگی های خود را دارد.

برای فهمیدن نام Function های موجود در یک فایل dll می‌توان از windebug استفاده کرد.

Linking:

۱- Static: که آنها دقیقاً درون برنامه کپی می‌شود مانند library های موجود در زبان C

۲- Runtime linking: در زمان اجرا به فایل خارجی دسترسی پیدا می‌کند که در صورت وجود داشتن dll درون حافظه آن را اجرا می‌کند و در غیر این صورت به سیستم عامل درخواست می‌دهد که dll مربوطه را load کند.

برای فهمیدن آدرس dll load شده از کدی استفاده می‌کنیم که حافظه را می‌گردد و از جدول AIt استفاده می‌کند که آدرس تمام برنامه‌ها در آن وجود دارد و با load شدن هر فایل خانه‌های این جدول تنظیم می‌شود. code section همه این ها یکی است ولی data sec. جدا از هم هستند و فقط کد خود برنامه‌ها یکدیگر share کنند.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

www.jozveha.ir

Subject:

Year. Month. Date. ( )

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

PAPCO