فربد فولادی-98243045

عرفان رفیعی اسکوئی-98243027

سوال اول

Register addressing

داده ها در رجیسترها نگهداری می شوند و نیازی به دسترسی به حافظه نیست.

```
MOV BX, DX        Copy the contents of DX into BX
MOV ES, AX        Copy the contents of AX into ES
```

Immediate Addressing

در حالت آدرس دهی فوری، 8 بیتی یا 16 بیتی داده به عنوان بخشی از دستورالعمل مشخص شده است.

```
MOV DL, 08H       The 8-bit data (08H) given in the instruction is moved to DL
MOV AX, 0A9FH     The 16-bit data (0A9FH) given in the instruction is moved to AX register
```

Direct Addressing

در اینجا effective address (EA) محل حافظه که عملوند داده در آن ذخیره می شود در دستورالعمل آورده شده است.

آدرس مؤثر فقط یک عدد 16 بیتی است که مستقیماً در دستورالعمل نوشته شده است.

```
MOV BX, [1354H]
MOV BL, [0400H]
```

براکت های اطراف H1354 نشان دهنده محتویات مکان حافظه است. پس از اجرا، این دستورالعمل محتویات مکان حافظه DS:1354H را در ثبات BX کپی می کند.

این حالت آدرس دهی مستقیم نامیده می شود زیرا جابجایی عملوند از segment section مستقیماً در دستورالعمل مشخص شده است.

Register Indirect Addressing

در Register indirect addressing نام register که دارای effective address (EA) است در دستورالعمل مشخص می شود.

رجیسترهایی که برای نگهداری EA استفاده می شوند یکی از رجیسترهای زیر هستند:

BX, BP, DI and SI

Example:

MOV CX, [BX]

Operations:

EA = (BX)
BA = (DS) x 16_{10}
MA = BA + EA

(CX) ← (MA) or,

Note : Register/ memory enclosed in brackets refer to content of register/ memory

Based Addressing

در Based Addressing، BX یا BP برای نگه داشتن مقدار پایه برای آدرس موثر استفاده می شود و جابجایی 8 بیتی یا بدون علامت 16 بیتی در دستورالعمل مشخص می شود.

در صورت جابجایی 8 بیتی، قبل از افزودن به مقدار پایه، علامت آن به 16 بیت افزایش می یابد.

وقتی BX مقدار پایه EA را نگه می دارد، آدرس فیزیکی 20 بیتی از BX و DS محاسبه می شود.
وقتی BP مقدار پایه EA را نگه می دارد، BP و SS استفاده می شود.

MOV AX, [BX + 08H]
Also: **MOV AX, [BX]+08H**

Operations:

0008_H ← 08_H (Sign extended)
EA = (BX) + 0008_H
BA = (DS) x 16_{10}
MA = BA + EA

(AX) ← (MA) or,

Indexed Addressing

رجیستر SI یا DI برای نگهداری یک مقدار شاخص برای داده های حافظه استفاده می شود و جابجایی 8 بیتی علامت دار یا 16 بیتی بدون علامت در دستورالعمل مشخص می شود.

جابجایی به مقدار شاخص در رجیستر SI یا DI اضافه می شود تا EA بدست آید.

در صورت جابجایی 8 بیتی، قبل از افزودن به مقدار پایه، علامت آن به 16 بیت افزایش می یابد.

MOV CX, [SI + 0A2H]

Also: **MOV CX, [SI]+0A2H**

Operations:

$FFA2_H \leftarrow A2_H$ **(Sign extended)**

$EA = (SI) + FFA2_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

(CX) ← (MA) or,

Based Index Addressing

در آدرس دهی شاخص پایه، آدرس موثر از مجموع یک ثبات پایه (BX یا BP) یک ثبات شاخص (SI یا DI) و یک جابجایی محاسبه می شود.

MOV DX, [BX + SI + 0AH]

Also: **MOV DX, [BX][SI]+0AH**

Operations:

$000A_H \leftarrow 0A_H$ **(Sign extended)**

$EA = (BX) + (SI) + 000A_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

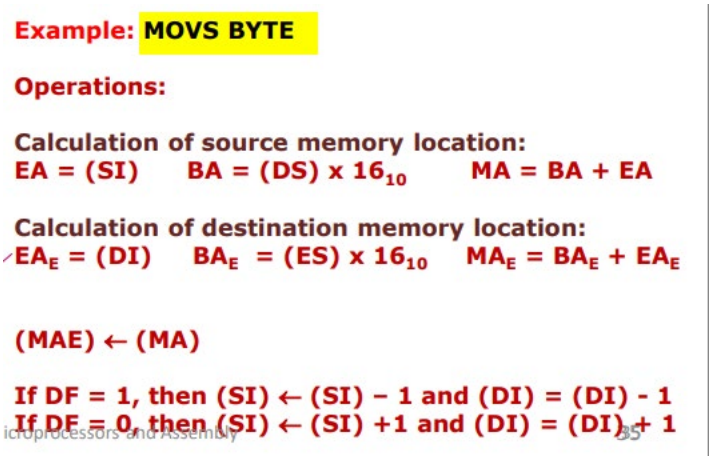
(DX) ← (MA) or,

String Addressing

در عملیات رشته ای برای کار بر روی داده های رشته ای استفاده می شود.

آدرس مؤثر (EA) داده مبدأ در ثبات SI و EA مقصد در رجیستر DI ذخیره می شود.

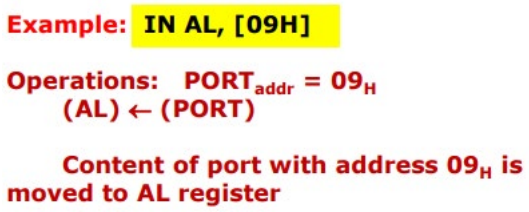
Segment register برای محاسبه آدرس پایه داده مبدا DS و داده مقصد ES است.



Direct I/O port Addressing AND Indirect I/O port Addressing

این حالت‌های آدرس‌دهی برای دسترسی به داده‌ها از دستگاه‌ها یا پورت‌های استاندارد I/O استفاده می‌شوند.

در حالت آدرس دهی پورت مستقیم، یک آدرس پورت 8 بیتی به طور مستقیم در دستورالعمل مشخص می شود.



Relative Addressing

در این حالت آدرس دهی، آدرس مؤثر یک دستورالعمل برنامه نسبت به نشانگر دستورالعمل (IP) توسط یک جابجایی علامت 8 بیتی مشخص می شود.

Example: JZ 0AH

Operations:

$000A_H \leftarrow 0A_H$ (sign extend)

If ZF = 1, then

EA = (IP) + $000A_H$
BA = (CS) x 16_{10}
MA = BA + EA

If ZF = 1, then the program control jumps to new address calculated above.

If ZF = 0, then next instruction of the program is executed.

Implied Addressing

دستورالعمل های استفاده از این حالت هیچ عملوندی ندارند.

خود دستورالعمل داده هایی را که باید توسط دستورالعمل اجرا شود را مشخص می کند.

CLC This clears the carry flag to zero.

سوال دوم

الف) نمی توانیم 3D را در AX برزیم باید یک مقدار هگز باشد.

ب) نمی توانیم یک 8 بیتی را در یک 16 بیتی بریزیم.

پ) destination در ADD می تواند رجیستر یا in memory باشد و نمی تواند عدد باشد.

ت) دستور INC فقط یک متغیر می گیرد و آن را یکی زیاد می کند پس 2 زیادی است.

ث) یک رجیستر را نمی توانیم در یک مقدار immediate بریزیم.

ج) دستور MOV، به شکل MOVE استفاده نمی شود.

چ) destination در ADD می تواند رجیستر یا in memory باشد و نمی تواند عدد باشد و src می تواند رجیستر و in memory و immediate باشد.

ح) نمی توانیم مقدار رجیستر 8 بیتی را در رجیستر 16 بیتی بریزیم.

خ) نمی توانیم مقدار رجیستر 16 بیتی را در رجیستر 8 بیتی بریزیم.

د) مقدار immediate 16 بیتی را نمی توانیم در رجیستر 8 بیتی بریزیم.

ذ) مقدار یک رجیستر را نمی توانیم در یک immediate بریزیم.

ر) دستور IN مقدار هگز دریافت نمی کند.

سوال سوم

این برنامه 12 کاراکتر اول رشته را reverse می کند.

در ابتدا data segment را مشخص می کنیم که string1 و string2 را مقدار دهی می کنیم که string2 یک آرایه به اندازه 15 است.

در code segment ابتدا با استفاده از assume، DS و ES اشاره می کنند به data segment ما و CS نیز به عنوان code شناخته می شود.

در START کار هایی که به ترتیب انجام می دهیم عبارتند از:

آدرس Data را در AX قرار می دهیم.

AX را در DS قرار می دهیم.

همچنین AX را در ES نیز قرار می دهیم.

آفست String1 را در BX قرار می دهیم.

BX را در SI که source index است قرار می دهیم.

آفست String2 را در DI که destination index است قرار می دهیم.

مقدار DI را 12 تا اضافه می کنیم تا در ادامه برای معکوس کردن String1 استفاده کنیم.

با دستور CLD، index registers ها را یکی اضافه می کنیم برای هر دفعه عملیات بر روی رشته.

در رجیستر CX مقدار 0CH که همان 12 است را ذخیره می کنیم.

در UP دستوراتی که انجام می شوند عبارتند از:

مقداری که در آدرس SI ذخیره شده است را در AL قرار می دهیم.

این مقدار را در DI که به خانه 12 ام آرایه اشاره دارد قرار می دهیم.

SI را یکی اضافه می کنیم.

DI را یکی کم می کنیم.

به لیلبل UP بر می گردیم که ساختار حلقه وار درست کنیم که همه رشته را در بر بگیریم.

REP MOVSB نیز شرط پایان حلقه را بررسی می کند. به این صورت که تا وقتی از رشته ای به رشته ای دیگر byte/word برای جا به جایی وجود دارد.

INT 03H نیز breakpoint است برای برنامه.