گزارش تحلیلی دستورکار دهم.

امیرحسین ادواری ۹۸۲٤۳۰۰٤ – زهرا حیدری ۹۸۲٤۳۰۲۰

سوال اول)

نوع اول REGESTER ADDRESSIG است و نیازی به دسترسی به Memory نداریم و دیتا در رجیستر ذخیره می شود .

```
MOV BX,DX
```

نوع دوم Immediate Addressing است که در آن یک دیتا8 یا 16 بیت را مستقیما داخل یک جیستر میریزیم.

```
MOV DL ,08H
```

این دستور عدد 08 در مبنای 16 را در رجیستر DL میریزد .

نوع سوم Direct Addressing است که آدرس Memory مورد نظر که یک عدد 16 بیتی است به کار می بریم.

```
MOV BX, [1368H]
```

در این دستور محتوایی که در آدرس 1368 در مموری وجود دارد را در رجیستر BX ذخیره می کند .

روش چهارم register indirect Addressing است که در آن به رجیستری که Effective Address EA را داخل خود نگه می دارد اشاره می کنیم .

رجیستر های خاصی EA را نگه می دارند که شامل DI , SI , BP , BX هستند .

MOV BX , [CX]

این دستور محتویات داخل آدرسی که در CX ذخیره شده است را در BX ذخیره می کند .

روش پنجم Based Addressing است که در آن BX یا BP آدرس Based از Effective Address را نگه می دارد که با یک عدد 8 یا 16 بیت جمع می شود تا آدرس نهایی را به ما بدهد .

MOV AX, [BX + 08H]

این دستور مقدار داخل BX + 08H خانه را داخل AX می ریزد .

روش ششم INDEXED Addressing است که در آن SI یا DI یک Index value را داخل خود دارند که باز با یک عدد 8 یا 16 بیتی جمع می شود تا آدرس نهایی را بدهد .

MOV CX , [SI + 0A2H]

این دستور محتویات آدرس مقدار داخل SI منها ی 0A2 خانه را داخل CX میریزد .

روش هفتم BASED INDEXED ADDRESSING مقدار Effective Address از مجموع base register و index register و یک عدد 8 یا 16 بیتی است .

MOV DX,[BX,SI,0AH]

این دستور محتویات آدرس مقدار داخل BX به اضافه SI به علاوه 0A2 را داخل DX میریزد .

روش هشتم آدرس دهی String Addressing است که برای عملیات رشته ای به کار می رود . Effective Address مبدا در SI و Effective address مقصد در DI ذخیره شده .

MOVS BYTES

این دستور مقدار SI را در DI ذخیره می کند .

روش نهم از روشهای IO PORT ADDRESSING برای دسترسی به Device های IO استفاده می کنیم . در روش Direct یک 8 port بیتی در دستور می اید .

IN AL , [09H]

این دستور محتویات پورت با ادرس 09H را داخل AL قرار می هد .

روش دهم Relative Addressing است که آدرس به صورت نسبی نسبت به instruction pointer IP تعیین می شود .

JZ 0AH

در این کد اگر ZF یک باشد به آدرس 0AH تا خانه بعد از Instruction pointer جامپ می کنیم .

روش یازدهم Implied Addressing است که در این مد Operand نداریم و خود دستور دیتا مورد نظر را تعیین می کند .

CLC

این دستور Carry Flag را صفر می کند .

سوال دوم)

الف) در این دستور میان عملوند اول و دوم ویرگول (کاما) قرار ندارد.

ب) در این دستور تلاش شده تا نیمه‌بالای CX در خود CX قرار گیرد، جدا از آنکه به‌طور منطقی ایراد دارد، ریختن مقدار ۸ بیتی در ۱۶ بیتی مولد خطاست.

پ) در این دستور عملوند مقصد یک عدد (immediate) است (dest یک لیترال عددی است) و مشخصا جنبه حافظه‌ای ندارد.

ADD	REG, memory
	memory, REG
	REG, REG
	memory, immediate
	REG, immediate
		Add.
		
		Algorithm:
		
		operand1 = operand1 + operand2
		
		Example:
		
		MOV AL, 5 ; AL = 5
		ADD AL, -3 ; AL = 2
		RET
		
		C Z S O P A
		r r r r r r

در این رفرنس مربوط به دستورات اسمبلی نیز این حالت برای این دستور وجود ندارد. (عملوند اول یا بایستی حافظه باشد و یا رجیستر)

ت) این دستور صرفا یک عملوند میپذیرد! کاراکتر نقطه در انتهای دستور نیز اضافی‌است و مولد خطا.

ث) در این دستور نیز مثل مورد پ عملوند اول نمی‌تواند immediate باشد (جنبه حافظه‌ای ندارد) شکل زیر حالات مجاز استفاده از دستور را نشان می‌دهد.

MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate SREG, memory memory, SREG REG, SREG SREG, REG

ج) دستوری به‌نام MOVE در اسمبلی 8086 وجود ندارد.

چ) این دستور نیز طبق تصویری که در مورد پ آمده است (و حالات مختلف عملوندهای دستور جمع را نشان می‌دهد). نامعتبر است، عملوند اول نمی‌تواند immediate باشد.

ح) در این دستور سعی شده تا مقدار ۸ بیتی در ۱۶ بیت ریخته شود که مولد خطاست.

خ) معکوس حالت قبل، در این دستور سعی شده یک مقدار (رجیستر) ۱۶ بیتی در ۸ بیتی ریخته شود که واضحا مولد خطاست.

د) در این دستور سعی شده که یک مقدار ۸ بیتی (AL) با یک immediate ۱۶ بیتی جمع شود که یک خطاست و این امکان موجود نیست.

ذ) طبق تصویر مورد ث که حالت مختلف عملوندها را در دستور MOV نشان می‌دهد، این دستور نامعتبر است چراکه عملوند اول نمیتواند immediate باشد.

ر) عملوند اول دستور IN صرفا می‌تواند AX یا AL باشد.

سوال سوم)

این برنامه به‌طور کلی ۱۲ کاراکتر اول رشته STRING1 را به صورت معکوس در رشته STRING2 قرار می‌دهد.

```
1  DATA SEGMENT
2          STRING1 DB 'MICROLAB OF SBU'
3          STRING2 DB 15 DUP(0)
4  DATA ENDS
5  CODE SEGMENT
6          ASSUME CS : CODE, DS : DATA,ES : DATA
7  START :
8      MOV AX, DATA
9      MOV DS, AX
10     MOV ES, AX
11     MOV BX, OFFSET STRING1
12     MOV SI, BX
13     MOV DI, OFFSET STRING2
14     ADD DI, 0CH
15     CLD
16     MOV CX, 0CH
17 UP :
18     MOV AL, [SI]
19     MOV ES : [DI] , AL
20     INC SI
21     DEC DI
22     LOOP UP
23     REP MOVSB
24     INT 03H
25 CODE ENDS
26 END START
27 ENDS
```

1. در ابتدا مقادیر ES و DS با آدرس Data مقدار دهی می‌شوند. (خطوط ۹ و ۱۰)
2. بعد آدرس شروع STRING1در BX و پس از آن BX در SI قرار می‌گیرد (خطوط ۱۱و ۱۲)
3. مشابه مرحله قبل مقدار STRING2 در DI قرار میگیرد، سپس DI با مقدار ۱۲ جمع می‌شود تا به ۱۲امین بایت در STRING2 اشاره کند (قرار است کاراکتر اول STRING1 در کاراکتر ۱۲ام STRING2 قرار گیرد و با تکرار این روال رشته معکوس شود) (خطوط ۱۳و۱٤)
4. سپس direction flag در خط ۱۵ام clear می‌شود (DF=0)
5. بعد در مقدار CX که عملا گام حلقه است، مقدار ۱۲ قرار داده می‌شود (خط ۱٦)

6. در این مرحله در یک حلقه مرتبا مقدار بایتی که SI در حال حاضر به آن اشاره می‌کند را در AL قرار داده (خط۱۸) و سپس این AL را در بایتی در حافظه که DI در حال حاضر به آن اشاره می‌کند، می‌نویسیم (خط ۱۹)
7. سپس در انتهای حلقه مقدار SI که عملا اشاره‌گر به STRING1 است را increment کرده تا به کاراکتر بعدی اشاره کند (خط ۲۰) و نیز مقدار DI که عملا اشاره‌گر به STRING2 است را decrement کرده تا به بایت پیش از بایت فعلی اشاره‌کند (طبق کارکردی که شرح داده شد، بایتهای STRING1 از ابتدا به ترتیب از انتها در STRING2 کپی می‌شوند)
8. دستور LOOP نیز کارکرد حلقه را فراهم می‌کند. در هرمرحله CX را یک واحد کم کرده و به ابتدای حلقه بازمی‌گردد. (خط ۲۲)
9. این دستور به تعداد CX دستوران MOV را تکرار کرده و مقادیر SI و DI را با توجه به اینکه DF=1 است increment میکند.

در نهایت نیز از اینتراپت ۳ برای ایجاد breakpoint (که برای اهداف دیباگینگ به‌کار میرود استفاده شده‌است)

INT 03h (3) **Breakpoint**

INT 3 is the breakpoint interrupt.

Debuggers use this interrupt to establish breakpoints in a program that is being debugged. This is normally done by substituting an INT 3 instruction, which is one byte long, for a byte in the actual program. The original byte from the program is restored by the debugger after it receives control through INT 3.