فاطمه سادات سیفی – 98243035

پارسا محمدپور – 98243050

سوالات تحلیلی:

۱- کدهای مختلف آدرس دهی پردازنده ۸۰۸۶ را با ذکر مثال توضیح دهید.

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

MOV BX,DX

- Immediate Addressing است که در آن یک دیتا۸ یا ۱۶ بیت را مستقیما داخل یک جیستر میریزیم.

MOV DL ,04H

این دستور عدد 04 در مبنای ۱۶ را در رجیستر DL میریزد .

- Direct Addressing است که آدرس Memory مورد نظر که یک عدد ۱۶ بیتی است به کار می بریم.

MOV BX, [1234H]

در این دستور محتوایی که در آدرس 1234 در مموری وجود دارد را در رجیستر BX ذخیره می کند .

- 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 BX , BP , SI , DI هستند .

MOV BX , [CX]

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

- Based Addressing است که در آن BX یا BP آدرس Based از Effective Address را نگه می دارد که با یک عدد ۸ یا ۱۶ 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 .

- INDEXED Addressing است که در آن SI یا DI یک Index value را داخل خود دارند که باز با یک عدد ۸ یا ۱۶ 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 .

- BASED INDEXED ADDRESSING مقدار Effective Address از مجموع base register و index register و یک عدد ۸ یا ۱۶ بیتی است .

MOV DX,[BX,SI,0A2H]

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

- 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 .

- از روشهای IO PORT ADDRESSING برای دسترسی به Device های IO استفاده می کنیم . در روش Direct یک port ۸ بیتی در دستور می اید .

IN AL , [09H]

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

- 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 .

- 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 .

۲- خطاهای موجود در هر یک از موارد زیر را بیان کنید.

ADD 2, CX	MOV CX, CH	MOV AX 3D
MOVE AX, 1H	MOV 23, AX	INC AX, 2.
MOV BH, AX	MOV DX,CL	ADD 3, 6
IN BL, 04H	MOV 7632H, CX	ADD AL, 2073H

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 واضح است که جنبه حافظه‌ای ندارد.

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.

ج) دستوری به‌نام MOVE در اسمبلی ۸۰۸۶ 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.

سوالات کدی:

سوال اول:

متغییر ها:

- **STRING:**

در این سوال، رشته ورودی را در قسمت DATA ذخیره می کنیم. این رشته هر مقدار دلخواهی می تواند داشته باشد و آدرس ابتدایی آن در متغییر STRING می باشد. تنها نکته موجود در این STRING این است که در انتهای آن باید یک کاراکتر $ قرار بگیرد تال مشخص کننده انتهای رشته باشد. برای استفاده از رشته، یا باید طول آن را نگهداری می کردیم در یک متغییر دیگر، یا انتهای این رشته را با یک کاراکتر خاص مشخص می کردیم که ما نیز همین کار را انجام دادیم.

- **YES:**

در این متغییر، مقدار YES را ذخیره می کنیم که بعدا در صورت پالیندورم بودن رشته، آن را چاپ کنیم. (در این متغییر هم مانند متغییر قبلی، در انتهای این رشته، کاراکتر $ را قرار می دهیم.)

- **NO:**

در این متغییر مقدار NO را قرار دادیم تا در صورت پالیندورم نبودن رشته، این متغییر را چاپ کنیم. (در این متغییر هم مانند متغییر قبلی، در انتهای این رشته، کاراکتر $ را قرار می دهیم.)

Main:

در این روتین، ابتدا آدرس شروع STRING را در رجیستر BX قرار می دهیم.

سپس باید در داخل یک حلقه، کل رشته را پیمایش کنیم تا به انتهای رشته (این مطلب را از کاراکتر $ متوجه می شویم) برسیم. تا بعدا بتوانیم از آدرس انتهایی آن برای بررسی کردن پالیندورم بودن یا نبودن این رشته استفاده کنیم.

برای طی کردن این حلقه، از یک سری رجیستر هایی استفاده می کنیم که در جدول زیر هرکدام از آنها به همراه کارکردشان، آمده اند:

رجیستر	کاربرد
BX	در این رجیستر در هر مرحله، آدرس ایندکسی از STRING قرار دارد که در حال بررسی آن می باشیم.
CL	این رجیستر، کاربردی مشابه با یک متغییر temp دارد که در آن در هر مرحله کاراکتری در آن قرار می گیرد که در ایندکس BX قرار دارد.

در هر مرحله اجرای این حلقه، این مقدار BX اضافه می شود تا به سراغ کاراکتر بعدی برویم. سپس در هر مرحله اجرای حلقه، چک میکنمی که آیا کاراکتر CL ، $ می باشد یا خیر؛ که اگر $ باشد، بررسی تمام می شود و به انتهای رشته رسیده ایم.

حال می خواهسم بررسی کنیم که آیا رشته ورودی پالیندورم می باشد یا خیر.
برای اینکار ابتدا یک pointer به ابتدا و یک pointer به انتهای رشته (این pointer را با استفاده از حلقه قبلی بدست آورده ایم و صرفا باید آن pointer را یکی کم کنیم تا دیگر به جای $ به آخرین کاراکتر موجود در رشته، point کنیم) را ست می کنیم. سپس از دو سمت شروع به حرکت می کنیم تا به یکدیگر برسیم.(شرط خاتمه این است که آدرس شروع و پایان، با یکدیگر برابر شوند یا آدرس شروع بیشتر از آدرس پایان باشد.)
در هر سری، کاراکتر های موجود در آدرس های ابتدایی و انتهایی رشته را با یکدیگر چک می کنیم که اگر با یکدیگر برابر باشند، به اجرای حلقه می پردازیم، در غیر این صورت، باید به state یی برویم که در آن می دانیم که رشته ورودی، پالیندورم نیست و باید رشتهNO را چاپ کنیم.

در اجرای این کار، از رجیستر هایی استفاده کرده ایم. در جدول زیر، رجیستر ها و کاربردشان آمده است:

رجیستر	کاربرد
BX	در این رجیستر در هر مرحله، آدرس ایندکسی قرار دارد که مربوط به حرکت از ابتدای رشته می باشد.
CX	در این رجیستر در هر مرحله، آدرس ایندکسی قرار دارد که مربوط به حرکت از انتهای رشته می باشد.
DI	برای اینکه بتوانیم در move_instruction از از کاراکتر موجود در آدرس CX استفاده کنیم، باید آن را در این رجیستر قرار دهیم، وگرنه به ارور می خوریم.
DL	در این رجیستر، کاراکتری قرار دارد که آدرس انتهایی ای که از آن می آییم، به آن اشاره می کند.

در هر یک از قسمت های مربوط به انتهای حلقه، ابتدا مقدار رجیستر AH را به گونه ای ست می کنیم که interrupt تولیدی، منجر به چاپ کردن شود و سپس آدرس شروع رشته ای را که قصد چاپ آن را داریم، در رجیستر BX قرار می دهیم. سپس از instruction مربوط به interrupt استفاده می کنیم.

سپس در یک حلقه دیگری که قرار دادیم، تمام طول رشته ورودی را پمایش می کنیم و کاراکتر های داخل این رشته را بررسی می کنیم و آنها را با y مقایسه می کنیم؛ در صورتی که برابر با y باشند، تعداد کاراکتر های y را (که مقدارشان را در رجیستر DL نگه می داریم) یکی زیاد می کنیم.

در انتهای اجرای این حلقه نیز، ابتدا (فرض می کنیم که تعداد کاراکتر های y موجود در رشته، یک رقمی می باشد) تعداد کاراکتر های y را به کد ASCII عه ’0‘ اضافه می کنیم، تا کد ASCII تعداد y های داخل رشته تولید شود. سپس آن را با دستورات لازم، چاپ می کنیم.

سوال دوم:

در این سوال، روتین های خواسته شده را پیاده سازی کردیم. برای هر کدام از آنها نیز، در قسمت DATA، متغییر های مورد نیاز به همراه سایز آنها را تعریف کردیم.

اما برای اینکه بتوانیم عملیات های 32 بیتی را انجام دهیم، ابتدا باید در ابتدای کد ، خط 386. را اضافه می کردیم.

سپس با توجه به رجیستر ها، هر کدام از روتین های مورد نظر را پیاده سازی کردیم.