سوال اول)

الف) تفاوت Assembler و Compiler آن است که Compiler کد High-Level را به کد ماشین تبدیل میکند در صورتی که Assembler کد اسمبلی را به کد ماشین تبدیل میکند. تفاوت دیگر آن است که کامپایلر تمام کد را یکجا به زبان ماشین تبدیل میکند در صورتی که Assembler نمیتواند این کار را یکجا انجام دهد بلکه به ترتیب انجام میشود. همچنین کامپایلر باهوش تر از Assembler است زیرا فرایند Optimization بر روی تبدیل کد به زبان ماشین انجام میدهد اما Assembler صرفا کد ماشین خروجی میدهد و بهینه سازی انجام نمیدهد. میتوان گفت Assember سریع تر است زیرا برخی فرایند های بهینه سازی کامپایلر را انجام نمیدهد.

شباهت این دو آن است که هر دو خروجی زبان ماشین میدهند که برای کامپیوتر قابل فهم تر است. همچنین در هر دو دستورات سطح بالایی که یک خط هستند به دستورات چند خطی ماشین تبدیل میشوند. پس این دستورات فقط در کامپایلر و Assembler هستند و نه در ماشین. (توضیح بیشتر در بخش ب)

ب) دستوراتی مانند LDR و ADR توسط Assembler به دستورات بیشتری شکسته میشوند. این دستورات برای پردازنده نیستند و صرفا مختص به Assembler است که عملا برای ساده تر شدن کار برنامه نویس طراحی شده و خود Assember آنها را به دستورات را شکسته و به دستورات قابل فهم برای پردازنده تبدیل میکند. برای مثال در دستور LDR ماکزیمم یک عدد 8 بیت است اما با این دستور میتوان اعداد بیشتر از 8 بیت (تا 32 بیت) را هم لود کرد. به گونه ای که خود Assembler از عدد 8 بیت من کرده، داخل رجیستر گذاشته و آن را شیفت میدهد. این کار را تکرار میکند تا بتوانیم عدد 32 بیت را لود کنیم.

سوال دوم)

در کل پنج مدل وجود دارد که سه مدل آن: On-chip data SRAM, On-chip EEPROM, on-chip FLASH

مقایسه: به دلیل پروتکل نوشتن در Flash که برای هر نوشتن باید دیتا قبلی پاک شود و اینکار بلوک به بلوک انجام میشود سرعت انتقال اطلاعات در Flash پایین است. پس بهتر است داده ای را در آن ذخیره کنیم که کمتر دچار تغییر میشوند از جمله ***Instruction*** ها. از طرفی SRAM بر خلاف Flash بایت به بایت اطلاعات را پاک میکند که منجر به افزایش سرعت آن میشود پس انتخاب مناسب تری برای ذخیره ***data*** است. اما مشکلی وجود دارد که این نوع حافظه موقت است یعنی با قطع جریان برق حافظه آن پاک میشود. برای حل این مشکل از EEPROM استفاده میشود که حافظه دائمی است. همچنین نوع پاک کردن آن بایت به بایت است که منجر به افزایش سرعت آن میشود پس انتخاب مناسبی برای ذخیره ***data*** برای طولانی مدت است. هرچند سرعت SRAM از EEPROM بیشتر است.

شباهت: EEPROM و SRAM هر دو بایت به بایت هستند که باعث میشود سرعت بیشتری نسبت به Flash داشته باشند. همچنین EEPROM و Flash هر دو حافظه دائمی هستند.

هر سه نوع حافظه همانطور که بالاتر توضیح داده شد مورد استفاده قرار میگیرند و خصوصیات خود را دارند.

سوال سوم)

الف) این دستور یک بخش یا section در کد ایجاد میکند. مانند تابع است از نظر Abstraction اما متفاوت از تابع است. معمولا برای جدا کردن بخش های کد و دیتا استفاده میشود. انواع Attribute های آن:

READONLY: زمانی که در بخش میخواهیم دیتا ذخیره کنیم که نباید تغییر کند استفاده میشود. اجازه تغییر آن را نداریم و فقط میتوانیم از آن بخوانیم.

READWRITE: زمانی که کد و دستورات را میخواهیم ذخیره کنیم استفاده میشود. زیرا دستورات میتوانند تغییر کنند پس هم نیاز به خواندن و هم نوشتن آن داریم.

CODE: وقتی این عبارت استفاده میشود دسترسی به صورت پیش فرض READONLY است.

DATA: وقتی این عبارت استفاده میشود دسترسی به صورت پیش فرض READWRITE است.

ALIGN: این دستور فواصل برای ذخیره داده را مشخص میکند. برای مثال اگر آخرین داده در خانه 26 ذخیره شده بود و ALIGN = 4 بود اولین خانه بعدی که برای ذخیره کردن داده استفاده میشود باید مضرب 4 باشد که میشود 28. زمانی استفاده میشود که احتمال میدهیم دیتای ذخیره شده میتواند از 4 بیت بیشتر باشد پس برای احتیاط 4 بیت به 4 بیت جلو میریم که دیتا overwrite نشود.

ب) HERE اول به عنوان label و B دستور branching و HERE دوم برای صدا کردن label استفاده میشود. این یکی دو خط عملا یک حلقه بینهایت است که برنامه حافظه های بعدی را نخوانده و به عنوان دستور حساب نکند. زیرا اگر این حلقه را نگذاریم برنامه دستورات بعدی را هم خوانده و سعی بر اجرا آنها میکند. خروجی ارور، یا انجام شدن دستورات رندوم است. پس برای پیشگیری این اتفاق از این خط استفاده میکنیم.

ج)

EQU: برای نگه داری constant آدرس استفاده میشود به طوری که نیاز به نوشتن مداوم آن آدرس نباشد و صرفا از اسم ثابت آن استفاده کنیم. مقدار آدرس میتواند عوض شود اما این دستور فقط یک pointer به آدرس را ذخیره میکند. پس با مقدار آن کاری ندارد. مثال:

```
TEMP EQU 2_0100011

ADD R1, #EQ2
```

DCB: برای تعریف و نگه داری constant یک بایت استفاده میشود (برخلاف EQU که آدرس ذخیره میکرد) و به عنوان متغیر ثابت در برنامه است. میتوان به آن یک string هم پاس داد که کد ASCII آن ذخیره میشود مانند آرایه در موری. مثال:

```
TEMP_VALUE DCB 21
```

SPACE: زمانی استفاده میشود که میخواهیم برای یک متغیر موری allocate کنیم اما نمیخواهیم مقدار اولیه ای به آن بدهیم. صرفا جای آن را رزرو میکنیم. برای آن تعداد بایت یک پارامتر است. مثال:

```
TEMP_VAR SPACE 8
```

سوال چهارم)

الف) میدانیم hfjnhمقدار ASCII کاراکتر های Micro_HW در R2 ذخیره شده، سپس مقدار 9 در R0 ذخیره شده و 9 را به ارقام ASCII جمع میکنیم. به دلیل Little Endian بودن low به high ذخیره میشود. حال اگر 9 را به آن اضافه کنیم باید به بخش high اضافه شود یعنی به مقدار ASCII کاراکتر 87+9 = 96.W

ب) با اینکار بین دو 0x50 و 0x30 3 فاصله می افتد.

ج) دستور LDR که برای operand2 مقدار Our_Data= داده شده را میتوان با دستور ADR و به دنبال آن Our_Data خالی (بدون علامت =) جایگزین کرد زیرا اضافه کردن عدد 9 باعث overflow شدن (بیشتر از 8 بیت immediate شدن) نمیشود.

سوال پنجم)

در پردازنده ARM Corttext-7 اولین آدرس حافظه که خوانده میشود از 0x00000004 تا 0x00000007 است. داخل این آدرس ها دستوری ذخیره نشده بلکه آدرس ذخیره شده است. آدرسی که سپس داخل program counter قرار میگیرد و بعد پردازنده دستورات را از program counter میخواند. اگر بخواهیم برنامه خود را موقع شروع اجرا کنیم باید آدرس اولین دستور آن را داخل حافظه های 0x00000004 تا 0x00000007 بگذاریم.

سوال ششم)

ما در ARM به همه فضای حافظه دسترسی نداریم. تنها 24 بیت به عنوان offset مقصد branch را مشخص میکنند که به آن دسترسی داریم و مابقی از کنترل مستقیم ما خارج است. تمام تنظیمات در این 24 بیت گنجانده میشود