دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

12/4/2021

Homework 4

Lec 13-18

MICROPROCESSOR
AND
ASSEMBLY LANGUAGE

Fall 2021

۱) به سوالات زیر در مورد اسمبلر پاسخ دهید:

الف) اسمبلر و کامپایلر چه تفاوت و شباهتی باهم دارند؟

پاسخ:

شباهت کامپایلر و اسمبلر در این مورد است که هردو کدی را به زبان ماشین تبدیل می‌کنند. یعنی یک دستور را ورودی می‌گیرند و در خروجی کد ۰و۱ به ما می‌دهند.

نکته: در داخل کامپایلرها اسمبلر وجود دارد و ابتدا دستور زبان های سطح بالا به اسمبلی تبدیل می‌شود و سپس توسط اسمبلر به ۰و۱ تبدیل خواهد شد.

کامپایلر و اسمبلی در چند مورد زیر باهم تفاوت دارند:

۱- تفاوت اسمبلر و کامپایلر آن است که کامپایلر کد Level-High را به کد ماشین تبدیل میکند در صورتی که اسمبلر کد اسمبلی را به کد ماشین تبدیل میکند.

۲- تفاوت دیگر آن است که کامپایلر تمام کد را یکجا به زبان ماشین تبدیل می‌کند در صورتی که اسمبلر نمیتواند این کار را انجام دهد و هر دستور به ترتیب به زبان ماشین تبدیل می‌شود.

۳- کامپایلر باهوش تر از اسمبلر است زیرا فرایند Optimization بر روی تبدیل کد به زبان ماشین انجام می‌دهد اما اسمبلر صرفا کد ماشین خروجی می‌دهد و بهینه سازی انجام نمی‌دهد. می‌توان گفت اسمبلر سریع‌تر است زیرا برخی فرایند های بهینه سازی کامپایلر را انجام نمی‌دهد.

۴- اسمبلر هر دستور اسمبلی به جز شبه دستورات رو به یک دستور سطح ماشین تبدیل می‌کند درحالی که کامپایلر لزوما این کار را انجام نمی‌دهد و عموما دستورات زبان‌های سطح بالا به چند دستور سطح ماشین تبدیل می‌شود.

ب) با توجه به تفاوت‌های ذکرشده در قسمت الف اسمبلر چگونه Pseudo Instructions (شبه دستورات) را پیاده‌سازی می‌کند.

پاسخ:

دستوراتی مانند LDR و ADR توسط اسمبلر به دستورات بیشتری شکسته می‌شوند (البته دستور LDR معادل یک دستور اسمبلی است). این دستورات برای پردازنده نیستند و صرفا مختص به اسمبلر هستند که برای ساده تر شدن کار برنامه نویس طراحی شده و خود اسمبلر آنها را به چند دستور قابل فهم برای پردازنده تبدیل می‌کند. برای مثال در دستور MOV ماکزیمم یک عدد ۸ بیتی را می‌توان در رجیستر ذخیره کرد اما با شبه دستور LDR می‌توان اعداد بیشتر از ۸ بیت تا ۳۲ بیت را هم لود کرد.

۲) شباهت‌ها و تفاوت‌های سه مدل حافظه On chip ای که در میکرو درس وجود دارد را نام ببرید و به چه دلایلی برای ساخت میکروکنترل به این سه مدل حافظه نیاز داریم؟

پاسخ:

سه مدل حافظه ی گفته شده SRAM و EEPROM و FLASH می‌باشد.

دو مدل حافظه ی FLASH و EEPROM حافظه های غیرفرار هستند. یعنی در صورت قطع شدن منبع انرژی، داده از حافظه پاک نمی‌شود. اما در سمت دیگر حافظه ی SRAM یک حافظه ی فرار یا VOLATILE می‌باشد و در صورت قطع جریان برق تمام داده های آن از بین می‌رود.

حافظه ی EEPROM قابلیت WRITE کردن محدودی را دارد و پس از یک مقدار محدودی نوشتن بر روی آن دیگر قابلیت استفاده را ندارد. نحوه ی نوشتن بر روی FLASH و EEPROM متفاوت است. در فلش داده به صورت BLOCK نوشته می‌شود و در ابتدا باید داده ی قبلی یا همان BLOCK قبلی پاک شود و سپس بلوک جدید داده نوشته شود. در صورتی که در EEPROM نوشتن داده به صورت بایتی می‌باشد. به دلیل اینکه نوشتن بر روی فلش یک خرده دشوارتر است، داده‌هایی را بر روی آن می‌نویسیم که زیاد تغییر پیدا نمی‌کنند. مثال دستورالعمل ها را در فلش ذخیره می‌کنیم. و یا داده هایی که زیاد تغییر نمی‌کنند. همچنین یکی دیگر از دالیل استفاده از فلش میزان DENSITY یا تراکم آن می‌باشد چرا که میزان حجم حافظه ی بیشتری را به ما می‌دهد.

در سمت دیگر داده را در EEPROM ذخیره می‌کنیم. چرا که خواندن و نوشتن بصورت بایتی می‌باشد. از لحاظ سرعت نیز SRAM دارای سرعت بیشتری نسبت به EEPROM می‌باشد. هم چنین SRAM میزان انرژی کمتری را مصرف می‌کند و برخالف EEPROM محدودیت تعداد دفعات نوشتن ندارد.

۳) به سوالات زیر در مورد Directiveها توضیح دهید:

الف) Directive Area و انواع Attributesهای آن را شرح دهید.

پاسخ:

از این Directive برای بخش‌بندی کد و مرتب کردن و کمک به خوانایی آن استفاده می‌شود.
یکی ازAttribute های این Directive ویژگی ReadOnly می‌باشد که مشخص می‌کند این قسمت از کد تنها قابل خواندن است و بعد از ذخیره شدن دیگر تغییر نمی‌کند و این بخش در Flash ذخیره خواهد شد. در مقابل ReadOnly ما قابلیت ReadWrite را داریم که مشخص می‌کند این بخش قابلیت خواندن و نوشتن دارد و در SRam ذخیره خواهد شد. یکی دیگر ازAttribute های آن Code می‌باشد که مشخص می‌کند این قسمت مربوط به کد برنامه می‌باشد ؛ این ویژگی، ReadOnly نیز می‌باشد. یعنی اگر از Code استفاده کنیم، آن قسمت تنها قابل خواندن خواهد بود. یکی دیگر ازAttribute های این Directive ، DATA می‌باشد و مشخص میکند که این قسمت مربوط به داده ی برنامه می‌باشد. این Attribute قابلیت ReadWrite را به این قسمت از کد می‌دهد. آخرین ویژگی این Directive، ویژگی Align است. این Attribute مشخص می‌کند که داده به صورت Align شده (یعنی مثلا از ضرایب ۲ یا ۴ در حافظه ذخیره شود) در حافظه ذخیره شود.

ب) چرا دو دستور زیر را در کنارهم در پایان برنامه‌های خود استفاده می‌کنیم و صرفا استفاده از Directive End به تنهایی کافی نیست؟

```
Here    B Here
        End
```

پاسخ:

Directive End صرفا برای اسمبلر کاربرد دارد و مشخص می‌کند بعد این خط دیگر جزو دستورات برنامه ما نیست و اسمبلر نباید آنها را به زبان ماشین تبدیل کند. درحالی که در صورت نبودن HERE B HERE و پس از اجرا شدن دستورالعمل ها، PC دوباره افزایش پیدا می‌کند و خانه های دیگری از حافظه را نیز به عنوان دستور العمل تعبیر می‌کند و آنها را fetch می‌کند و به عنوان دستورالعمل اجرا می‌کند. این کار که ناخواسته می‌باشد موجب اجرا شدن دستورالعمل‌های valid یا invalid می‌شود. برای جلوگیری از این کار از branch و لیبل HERE استفاده می‌کنیم. یعنی یک حلقه به وجود میآید و HERE یک لیبل هست که به خانه ی فعلی حافظه اشاره می‌کند. پس این دستور موجب می‌شود که در این خانه از حافظه تا ابد! بماند.

ج) فرق بین سه Directive زیر چیست و از هرکدام برای چه کاربردی استفاده می‌شود(برای هر Directive یک مثال بزنید)؟

Directives: DCB, SPACE, EQU

پاسخ:

EQU: برای تعریف یک Constant در برنامه خود از EQU استفاده می‌کنیم و اسمبلر در هر جای برنامه مانند مثال اگر عبارت Count را ببیند آن را به 0x25 تبدیل می‌کند.

Count EQU 0x25

DCB: بعد از اینکه اسمبلر به این خط می‌رسد به اندازه یک بایت بعد از آن دستور در حافظه فضا رزرو می‌کند و برای آن اسم مشخص شده را قرار می‌دهد. باید حتما مقدار اولیه مشخص باشد.

Myvalue DCB 5

SPACE: به اندازه حافظه مشخص‌شده از فضای حافظه ذخیره می‌کند و اسم داده شده را برروی آن می‌گذارد. نیازی نیست به آن مقدار اولیه بدهیم.

Long_var Space 4

۴) به سوالات زیر در مورد نگاشت حافظه پاسخ دهید:

الف) در صورت اجرا برنامه زیر در نهایت در رجیستر R10 چه چیزی ذخیره می‌شود؟ (اعداد با متد Little Endian در رجیسترها ذخیره می‌شوند.)

```
Area Exercise4_Code, Readonly, Code
    LDR  R2, =Our_Data;
    MOV R0, #9
    ADD R2, R2, R0;
    LDRB  R10, [R2];
HERE    B HERE; stay here forever

Area Exercise4_Data, Data
Our_Date
    DCB   "Micro_HW"
    DCD   0x50, 0x30
    END
```

پاسخ:

باتوجه به اینکه آدرسی که Our_Data به آن اشاره می‌کند در ابتدا همان آدرس بایتی است که کاراکتر m در آن ذخیره شده پس بعد از اضافه کردن #9 به مقدار آن به بایت دوم پر ارزش از Word ای که در آن 0x50 ذخیره شده است اشاره خواهد کرد و چون مقدار 0x50 در آن Word به صورت little endian ذخیره شده است مقدار بایت دوم پرارزش صفر خواهد بود و در نهایت مقدار صفر در R10 لود می‌شود.

ب) در صورتی که قبل از دستور DCD، دستور Align 4 اضافه کنیم نگاشت حافظه ما به چه صورت خواهد بود با فرض اینکه از خانه شماره صفر حافظه شروع به ذخیره دستورات کنیم؟ (همانطور که در ویدیوها مطرح شده، شبه دستور LDR نیز معادل با یک دستور اسمبلی خواهد بود و در یک 32 Bit سیو می‌شود.)

پاسخ:

با توجه به اینکه همه دستورات ما در ۴ بایت ذخیره می‌شوند و حتی رشته ما در ۸ بایت (هر کاراکتر به ۱ بایت برای ذخیره شدن نیاز دارد.) سیو می‌شود و در نهایت دستور DCD یک ۴ بایتی را رزرو می‌کند، در نتیجه Align ۴ تاثیری در نگاشت حافظه ما ایجاد نمی‌کند و مشابه حالت قبل خواهد بود.

ج) آیا امکان دارد بتوان شبه دستور LDR را با یک دستور دیگر جایگزین کرد؟

پاسخ:

بله، می‌توانیم از شبه دستور ADR استفاده کنیم که به صورت زیر خواهد شد دستور معادل:

```
ADR R2, Our_Data
```

نکته: درصورت استفاده از شبه دستور ADR دیگر نیازی نیست از = استفاده کنیم.

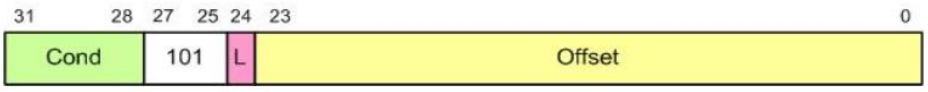
۵) فرآیندی که پردازنده میکرو درس (Arm Cortex-M) بعد از شروع مجدد یا ری استارت برای شروع به کار طی می‌کند را شرح دهید.

پاسخ:

برخلاف باقی پردازنده‌های شرکت Arm که از خانه صفر شروع به خواندن دستورات از حافظه می‌کنند، در Arm Cortex-M ابتدا پردازنده از خانه شماره ۴ تا ۷ حافظه را می‌خواند که در این ۳۲ بیت آدرسی ذخیره شده است که باید PC از آن شروع کند. در نتیجه این مقدار را در PC ذخیره می‌کند و به سراغ اجرای برنامه خود می‌رود. پس بعد از ذخیره برنامه خود در میکرو باید ادرس دستور اول برنامه خود را در خانه شماره ۴ تا ۷ حافظه ذخیره کنیم تا میکرو به درستی Program شود.

۶) با توجه به این نکته که طول دستورات در پردازنده Arm، 32 Bit است چگونه آدرس خانه‌ای از حافظه که باید به آن Branch شود در این دستور ذخیره می‌شود.

پاسخ:



همانطور که مشاهده می‌کنید ساختار دستورات Branch به شکل بالا است و تنها می‌توان یک آدرس ۲۴ بیتی را در بخش Offset ذخیره کرد در نتیجه ما با استفاده از دستور Branch به تمامی حافظه دسترسی نداریم و تنها به بخشی از آن دسترسی خواهیم داشت. Arm برای اینکه ما بتوانیم به خانه های بیشتری از حافظه دسترسی داشته باشیم نه صرفا خانه‌هایی از حافظه که آدرس آن ها ۲۴ بیت یا کمترست، امکانی را ایجاد کرده که این بخش Offset با آدرس کنونی ما جمع می‌شود و این مدلی این آدرس ۲۴ بیتی فاصله ما به خانه حافظه Branch را تعیین می‌کند نه آدرس مطلق آن خانه حافظه و ما می‌توانیم به بخش بیشتری از حافظه دسترسی داشته باشیم.