دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

12/4/2021

Homework 4

Lec 13-18

MICROPROCESSOR
AND
ASSEMBLY LANGUAGE

Fall 2021

1) به سوالات زیر در مورد اسمبلر پاسخ دهید:

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

جواب:

کامپایلر کد نوشته شده توسط برنامه نویس که اغلب سطح بالا میباشد را به کد به زبان سطح ماشین تبدیل میکند؛ البته بعضی از کامپایلرها کد را به زبان اسمبلی تبدیل میکنند. در حالی که اسمبلر کد نوشته شده به زبان اسمبلی را به زبان ماشین تبدیل میکند.

کامپایلر هوشمندی بیشتری را برای تبدیل کد به زبان ماشین دارد. برای مثال کامپایلر ممکن است بر روی کد یک سری بهینه سازی را انجام دهد در حالی که اسمبلر چنین کاری را انجام نمیدهد.

کامپایلر به یکباره کد را به زبان سطح ماشین تبدیل میکند. اما اسمبلر نمیتواند به یکباره کد را به زبان ماشین تبدیل کند.

از شباهت های کامپایلر و اسمبلر میتوان به این اشاره کرد که هر دوی آنها کد را به عنوان ورودی میگیرند و آنرا به کد زبان سطح ماشین تبدیل میکنند.

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

جواب:

اسمبلر شبه دستور ها را به یک یا چند دستور اسمبلی تبدیل میکند. چرا که ISAایی برای شبه دستور طراحی نشده.

2) 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؟

جواب:

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 دشوار تر است، داده هایی را بر روی آن مینویسیم که زیاد تغییر پیدا نمیکنند. مثلا دستورالعمل ها را در فلش ذخیره میکنیم. و یا داده هایی که زیاد تغییر نمیکنند. همچنین یکی دیگر از دلایل 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. همچنین SRAM میزان انرژی کمتری را مصرف میکند و برخلاف EEPROM محدودیت تعداد دفعات نوشتن ندارد.

3) به سوالات زیر در مورد Directiveها توضیح دهید:

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

جواب:

این Directive یک section را مشخص میکند و نوع آنرا نیز مشخص میکند. برای بخش بندی و منظم تر شدن کد از آن استفاده میکنند.

یکی از attributeهای این directive، ReadOnly میباشد که مشخص میکند این قسمت از کد تنها قابل خواندن است. یکی دیگر از attributeهای آن Code میباشد که مشخص میکند این قسمت مربوط به کد برنامه میباشد؛ این Attribute، Readonly نیز میباشد. یعنی اگر از Code استفاده کنیم، آن قسمت تنها قابل خواندن خواهد بود.

یکی دیگر از attributeهای 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.

و ALIGN از attributeهای دیگر این directive میباشد. این attribute مشخص میکند که داده ی مربوط به این قسمت را به صورت align در حافظه قرار دهد.(ضریبی از 2 یا 4 چرا که در حالت دیفالت به صورت بایتی میباشد)

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
```

جواب:

در صورت نبودن HERE B HERE و پس از اجرا شدن دستورالعمل ها، pc دوباره افزایش پیدا میکند و خانه های دیگر 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 دستورالعمل های 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

جواب:

DCB یک بایت از مموری یا بیشتر را allocate میکند و به value نوشته شده اختصاص میدهد. اینکه یک بایت یا بیشتر را Allocate کند بستگی به مقداری دارد که پس از DCB نوشته میشود. مانند DCB 0X11; که یک بایت از حافظه به مقدار 0X11 اختصاص پیدا میکند.

SPACE یک بلاک صفرشده از مموری را رزرو میکند. مقدار بایتی که باید رزرو شود پس از اسم این directive نوشته میشود. مثلا: data SPACE 255 که 255 بایت صفرشده[1] از حافظه را رزرو میکند.

EQU یک نام را به یک مقدار constant میدهد و میتوان از آن نام به جای استفاده ی مستقیم از آن constant استفاده نمود. از مزایای آن میتوان به آپدیت کردن راحت کد اشاره کرد. مثلا یک constant که در جاهای مختلف کد استفاده شده و با تغییر مقدار آن، تمام جاهای دیگر که از این نام استفاده کرده نیز آپدیت میشوند.

مثال test EQU 2 که نام test را برای مقدار constant 2 قرار میدهد.

[1] Zeroed

4) 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
```

جواب:

در ابتدا لیبل Over-Data مقدار آدرس اولین خانه ی allocate شده برای Micro_HW را در رجیستر R2 میریزد. سپس مقدار constant 9 را در R0 قرار میدهد. و بعد مقدار آدرس که در رجیستر R2 موجود است را با 9 جمع میزند و به 9 بایت بعدتر از از آن آدرس اشاره میکند. حال مقدار بایتی که رجیستر R0 به آن اشاره میکند، بایت دوم کم ارزش مربوط به 0x50 میباشد که مقدار آن برابر با 0 هست. پس مقدار رجیستر R10 برابر با صفر میباشد.(توجه شود که در اینجا با توجه به استفاده از DCD چهار بایت برای 0X50 رزرو شده و تنها بایت اول مورد استفاده قرار گرفته شده است. و سه بایت دیگر مقدار صفر را دارند.)

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.)

جواب:

اگر قبل از DCD از ALIGN 4 استفاده کنیم، آدرس مموری که برای این دستور اختصاص می‌یابد از ضریب 4 میباشد.

حال با توجه به اینکه 5 دستور داریم که هر کدام 32بایت و از ضریب چهار هستند و سپس استفاده از DCB که 8بایت را اشغال میکند و از ضریب 4 هست، پس استفاده از ALGIN در اینجا تاثیری ندارد و مانند حالت قبل میباشد.

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

بله. یک دستور دیگر به اسم ADDR وجود دارد که آدرس حافظه با یک Range خاص را در یک رجیستر load میکند.

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

پردازنده ی arm7 بعد از روشن شدن از خانه ی 0حافظه شروع به خواندن دستورالعمل ها میکند.
اما پردازنده ی arm cortex m بعد از روشن شدن شروع به خواندن خانه های 3تا7 حافظه میکند و آنها را در pc قرار میدهد. حال برنامه که در flash، load شده است و pc که مقدار دهی شده به یکی از خانه های flash اشاره میکند و سپس دستورالعمل ها اجرا میشوند.

6) 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.

جواب:

برای branch یک محدودیت وجود دارد و 24 بیت برای مشخص کردن جایی که قرار است branch انجام شود لازم است. مقدار این 24بیت نسبت به مکان فعلی کد میباشد. مثلا اگر pc یک مقداری را دارد، پس از اجرای branch مقدار pc با مقدار این 24بیت جمع میشود.

- مهلت ارسال تمرین تا ساعت 23.55 روز **سه شنبه بیست سه آذر** می‌باشد.
- سوالات خود را می‌توانید از طریق تلگرام از تدریسیارهای گروه خود بپرسید.
- ارائه پاسخ تمرین به بهتر است به روش‌های زیر باشد:
  1) استفاده از فایل docx. تایپ پاسخ‌ها و ارائه فایل Pdf
  2) چاپ تمرین و پاسخ دهی به صورت دستنویس خوانا
- فایل پاسخ تمرین را تنها با قالب **HW4 -9731***.pdf** در مودل بارگزاری کنید.
  - نمونه: HW4-9731121
- فایل زیپ ارسال **نکنید**.