دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

5/5/2021

Homework 4

Lec 9-12

MICROPROCESSOR
AND
ASSEMBLY LANGUAGE

Spring 2021

1) با بررسی خطوط ارتباطی و ذکر مقدار اولیه در صورت وجود، در هنگام استفاده از پروتکل UART امکان (ایجاد شبکه چندتایی) اتصال چندین دستگاه را بررسی کنید؟

طبق صحبت های استاد ما نمیتوانیم یک شبکه چندتایی با پروتکل UART بسازیم. درواقع برخلاف SPI و I2C اینجا ارتباط به صورتpoint-to-point میباشد.

درواقع به نقل قول از استاد دردقیقه‌ی 5 از لکچر9:

" وقتی یک خط راتباطی بیشتر وجود ندارد برخلاف SPI و I2C اینجا ارتباط، ارتباط point-to-point میباشد. با این یک خط فقط دوتا ماژول UART به هم وصل میشوند. ماژول UART سومی نمیتواند در شبکه باشد.

درواقع اینجا یک شبکه از پروتکل UART نمیتوانیم بسازیم. درحالی که در پروتکل I2C ما میتوانستیم بیش از دو واحد را به هم در یک ماژول یا شبکه‌ی I2C وصل کنیم. یا در SPI نیز به همین ترتیب میتوانستیم یک Master و Multiple Slave داشته باشیم. اما در پروتکل UART همچین امکانی وجود ندارد. یک فرستنده و یک گیرنده بیشتر در این شبکه و کانال ارتباطی وجود ندارد."

(2

الف) در پروتکل UART چرا گیرنده از اولین کلاک شروع به گرفتن دیتای ارسال شده نمی‌کند؟

ب) در کدام کلاک‌ها گیرنده بایت‌های اول تا سوم ارسال شده توسط فرستنده را دریافت خواهد کرد؟

الف) در UART میدانیم که ممکن است دو ماژول ارتباطی با کلاک یکسانی کار نکنند و دوتا کلاک مستقل از هم دارند. ماهم که یک سیم برای ارسال کلاک و synchronization نداریم. بنابراین یک Baud rate داریم که جفتشون کلاکشون را تبدیل به آن baud rate میکنند و نرخ ارسال و transmit دیتاشون براساس آن میباشد.

حالا ممکنه نتونن این دوتا ماژول دقیقا یک baud rate یکسان تولید کنند و طبق قرار داد برای اینکه UART درست کار کند تا 10 درصد جا دارد که baud rate شون با هم فرق کند. همچنین نرخ نمونه برداری در گیرنده‌ی هر ماژول 16 برابر baud rate میباشد.

با توجه به اینکه فرستنده و گیرنده کلاک مستقل دارند و صرفا سر توافقی که سر یک عدد کردند، دارند دیتا را میفرستند و میگیرند ، گیرنده برای اینکه خیالش راحت باشد همیشه دیتای درست را میگیرد ، همیشه باید سعی کند که جایی از 16 کلاکش نمونه برداری از دیتا کند که وسط کلاک فرستنده باشد. یعنی هر یک کلاک فرستنده یک بیت دیتا همزمان باهاش حمل میشود و هر 16 کلاک گیرنده معادل یک کلاک فرستنده است و تو هر 16 کلاک هم گیرنده دارد یک دیتای یکسان را روی خط میبیند ولی باید یکی از اونا را به عنوان دیتا بردارد و طوری هم بردارد که امکان خط در آن نباشد.

باتوجه به اینکه کلاک ها مستقل از هم هستند لزوما کلاک اول گیرنده ممکنه شروع کلاک فرستنده نباشد. برای همین یک ذره باید جلوتر برود. چون این 16 کلاک در گیرنده ممکنه یک ذره اختلاف فاز با اون یک کلاک فرستنده داشته باشد برای همین وسط 16 تای خودش یعنی کلاک 8 ام را به عنوان کلاک نمونه برداری در نظر میگیرد که خیالش راحت باشد که حتی اگر به اندازه 10 درصد کلاک هاشون باهم متفاوت بود بازهم دیتایی که میگیرد دیتای درستی باشد و همونی باشد فرستنده دارد الان با این کلاکش میفرستد.

همچنین این موضوع را هم میدانیم که در اولین کلاک فرستنده یک استارت بیت میفرستد و بنابراین گیرنده اول باید منتظر باشد تا استارت بیت تمام شود و در کلاک 8 ام خود آن را کپچر کند و بعد در کلاک 24 ام خود اولین بیت دیتا را طبق توضیحات بالا دریافت کند.

ب)میدانیم که در UART هر یک بایت دیتا با یک packet ارسال میشود. ما در اینجا فرض کردیم که فرستنده در هر packet، اول 1 بیت استارت دارد ، بعد 8 بیت دیتا دارد، بعد 1 بیت parity دارد و بعد 2 بیت به عنوان stop bit دارد. همچنین فرض کردیم packet ها بلافاصله و بدون هیچ تاخیری پشت سرهم ارسال میشوند.

در ابتدای کار که فرستنده به اندازه یک کلاک باید خط ارتباطی را 0 کند که به معنی شروع ارتباط است. پس در کلاک 8 ام گیرنده ، گیرنده 0 را از روی خط برمیدارد و میفهمد ارتباط شروع شده است. سپس روی کلاک 24 ام اولین بیت از بایت اول را دریافت میکند . در کلاک 40 ام خودش بیت دوم را برمیدارد و در کلاک 56 ام بیت سوم بایت اول را برمیدارد و .... به همین ترتیب. پس میتوان گفت برای بایت های اول تا سوم گیرنده در این کلاک ها دیتا را برمیدارد:

بایت اول: کلاک 24 ام بیت1، کلاک 40 ام بیت2، کلاک56 ام بیت3، کلاک 72ام بیت 4، کلاک 88 ام بیت 5، کلاک 104 ام بیت 5، کلاک 120 ام بیت 6، کلاک 136ام بیت 7، کلاک 152ام بیت 8

سپس در کلاک 168 ام بیت Parity مربوط به بسته‌ی اول را میگیرد بعد در کلاک های 184 و 200 بیت های stop بسته‌ی اول را میگیرد.

حال در کلاک 208 ام بیت استارت مربوط به packet دوم که شامل بایت دوم است را میگیرد.

بایت دوم: کلاک 224 ام بیت1، کلاک 240 ام بیت2، کلاک256 ام بیت3، کلاک 272ام بیت 4، کلاک 288 ام بیت 5، کلاک 304 ام بیت 5، کلاک 320 ام بیت 6، کلاک 336ام بیت 7، کلاک 352ام بیت 8

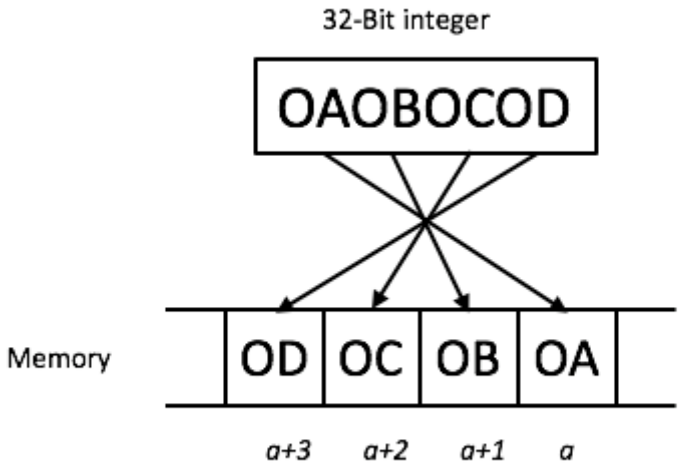
سپس در کلاک 368 ام بیت Parity مربوط به بسته‌ی اول را میگیرد بعد در کلاک های 384 و 400 بیت های stop بسته‌ی دوم را میگیرد.

حال در کلاک 408 ام بیت استارت مربوط به packet سوم که شامل بایت سوم است را میگیرد.

بایت سوم: کلاک 424 ام بیت1، کلاک 440 ام بیت2، کلاک456 ام بیت3، کلاک 472ام بیت 4، کلاک 488 ام بیت 5، کلاک 504 ام بیت 5، کلاک 520 ام بیت 6، کلاک 536ام بیت 7، کلاک 552ام بیت 8

سپس در کلاک 568 ام بیت Parity مربوط به بسته‌ی اول را میگیرد بعد در کلاک های 584 و 600 بیت های stop بسته‌ی سوم را میگیرد

3) با ذکر دلیل بیان کنید تصویر زیر کدام یک از روش‌های ذخیره‌سازی در حافظه را نشان می‌دهد؟



روش Big endian را نشان میدهد. زیرا همانطور که از تصویر پیداست مقادیر پرارزش‌تر داده(MSB) را در آدرس با شماره‌ی کمتر و و مقادیر کم ارزش تر داده(LSB) را در آدرس با شماره‌ی بیشتر ذخیره میکند.

مثلا OA که MSB است در آدرس پایین تری قرار گرفته و OD که LSB است در آدرس بالاتری قرار گرفته است.

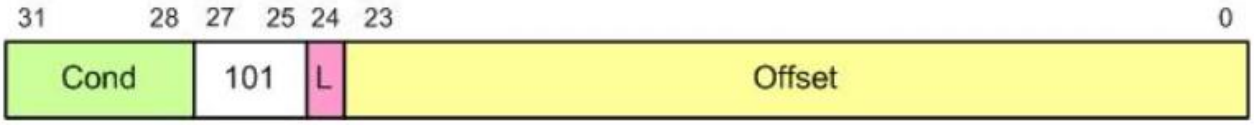
- **Big endian**
  - **Low byte goes to the high memory location**
  - **High byte goes to the low memory address**

4)

الف) توضیح دهید آیا برای اجرای دستور branch از کل 4 گیگابایت فضا می‌توانیم استفاده کنیم؟

خیر نمیتوانیم- طبق توضیحات استاد در اصفحه 5 لکچر18 ، ما مثلا از branch در HERE B HERE استفاده میکردیم. به کمک اون میگفتیم بپر به اون آدرسی که لیبل HERE میگوید.

میگفتیم آدرس ها 32 بیتی اند. اون آدرسی که در دستور branch قرار میگیرد طبق این شکل یک آدرس 24 بیتی است نه یک آدرس 32 بیتی . این معنیش اینه که اگرچه آدرس یک 32بیتی باید باشد ولی وقتی از branch استفاده میکنیم ، نمیتوانیم به هرجایی دلمون میخواهد branch کنیم و فقط به یک فضایی که با 24 بیت قابلیت آدرس دهی دارد، میتوانیم branch کنیم. بنابراین با branch نمیتوان از کل فضای 4 گیگ حافظه استفاده کرد.



ب) مقدار نهایی رجیسترها در قطعه کد زیر چیست؟ (مرحله‌ها را توضیح دهید.)

```
AREA myData, Data
L1      MOV     R0, #0
        ADD     R1,R0,#2
        B       L2
        ADD     R2,R0,R1
L3      RSB     R1,R2,#5
        B       L4
        ADD     R1,R1,R0
L2      MOV     R0,R1
        SUB     R2,R0,#3
        B       L3
L4      B       L4
```

در ابتدای کار یک خط نوشته شده که جزء خطوط کد نمیباشد. درواقع به کمک AREA اومدیم سکشن بندی کردیم. بعد گفتیم اسم این فضا myData میباشد و از جنس Data میباشد.

سپس به سراغ خط اول کد میرویم که به کمک دستور MOV مقدار constant صفر را در رجیستر R0 میریزد.

سپس به کمک دستور ADD مقدار ثابت 2 را با مقدار موجود در رجیستر R0 که برابر 0 بود جمع میکند و حاصل را در رجیستر R1 میریزد. بنابراین در R1 مقدار 2 داریم.

سپس branch میکند (بدون شرط) به L2 .

در آنجا به کمک دستور MOV مقدار موجود در رجیستر R1 یعنی 2 را در رجیستر R0 میریزد. پس مقدار R0 اکنون 2 میباشد.

به کمک دستور SUB ، مقدار موجود در R0 را منهای 3 میکند و حاصل را در R2 میریزد. در اینجا مقدار 1 را در R2 میریزد.

سپس branch میکند (بدون شرط) به L3 .

به کمک RSB که Reverse SUB میباشد مقدار ثابت 5 را منهای مقدار موجود در رجیستر R2 یعنی 1 میکند و حاصل که 4 هست را در رجیستر R1 مریزد.

سپس branch میکند (بدون شرط) به L4 .

در آنجا هم دوباره به L4 برنچ میکند و درواقع در یک loop قرار میگیرد.

پس مقادیر نهایی رجیسترها :

R0 میشود 2

R1 میشود 4

R2 میشود 1

5) شبه دستورات (Pseudo Instructions) چه دستوراتی هستند؟ 2 مورد را نام ببرید و توضیح دهید.

شبه دستورات ، دستوراتی هستند که ما در زبان اسمبلی میتوانیم استفاده کنیم ولی در ISA تعریف نشده است. ما تا الان فرضمون این بود که هر دستور اسمبلی معادل یک دستور 32 بیتی هست که در ISA تعریف شده و به پردازنده داده میشود و پردازنده Opcode آن را در می‌آورد، Operand هاش رو هم در می آورد و سپس اجرا میکند. توی زبان اسمبلی یک سری کارها ، خیلی کارهای رایجی است و خیلی لازممون میشود که انجام بدیم ولی براش دستور مستقیمی وجود ندارد. برای اینکه اون کار را انجام دهیم مجبوریم چندتا دستور را با هم ترکیب کنیم تا به اون هدفمون برسیم. حال به کمک شبه دستورات ، اسمبلر میگوید من خودم اون کارها رو براتون تبدیلاتش رو انجام میدهم و اگر برای مثال از یک شبه دستور استفاده کنید خود اسمبلر آن را به چندین دستور اصلی که در ISA هست تبدیل میکند و کار را برایمان انجام میدهد.

پس شبه دستورات میشود، instruction هایی که در سطح اسمبلی مجاز است ولی در ISA نیست.

برای مثال دو دستور LDR و ADR شبه دستور میباشند.

LDR : شکل کلی آن به صورت زیر میباشد و Rd رجیستر مقصد میباشد. یک constant ، 32 بیتی میتواند به عنوان ورودی و Op2 قرار بگیرد. در اینجا به کمک اون = ، اسمبلر میفهمد که یک شبه دستور است.

LDR Rd, =32-bit_immediate_value

کاری که میکند این است که این عدد constant 32 بیتی را برای ما به رجیستر مقصد Rd نتقل میکند و میریزد.

در حالت عادی میدانیم که برای برای انتقال یک مقدار ثابت به یک رجیستر از دستور MOV استفاده میکنیم. از طرفی میدانیم که دستورات ما 32بیتی میباشند. یک بخشیش Opcode میباشد. یک بخش دیگه هم برای اشاره به اون رجیستر مقصد میباشد و در نهایت از این 32 بیت طبق ISAی ARM فقط 8 بیت برای مقدار constant ورودی باقی میماند. پس درنهایت بزرگ ترین عددی که میتوان به کمک دستور MOV در یک رجیستر ریخت یک عدد 8 بیتی یا درواقع عدد 255 میباشد.

ADR : این دستور اجازه میدهد که یک آدرس 32 بیتی constant را داخل یک رجیستر بریزید. آدرس ها توی برنامه با label هایی مشخص میشوند و شما بعضی وقت ها میخواهید آدرس یک دستور را توی یک Reg بریزید. آدرس اون دستور یک مقدار constant میباشد چون حای دستور در حافظه جای ثابتی میباشد.

اگر خودتان بخواهید constant یvalue آن را بنویسید کار سختیه. از طرفی آن value را شما در زمانی که دارید اسمبلی کد میزنید ، شاید مقدار دقیقش را ندونید . پس به جای اینکه آدرس ها را حفظ کنید میگویید من فلان دستور را که روی فلان label هست میخواهم.

پس کاری که میکند اینه که زمانی که دستورات توی حافظه load شدن و آدرسشون مشخص شد، آدرس اون دستوری که بهش label چسبیده را load میکند در Rn. این دستور نیز در حالت کلی به صورت زیر است:

ADR Rn, label

6) کاربرد پایه‌های USART را توضیح دهید؟

پایه SCK : (Serial Clock) سیگنال کلاک هست که در مود سنکرون USART ، کلاک بین میکروی ما و ماژول اکسترنال ما مشترک میشود.

پایه TXD : پایه‌ای است که برای ارسال و Transmit دیتا استفاده میشود. در واقع ماژول موردنظر به کمک این پایه دیتا را به پایه‌ی RXD گیرنده و ماژول دیگر میفرستد.

طبق صحبت‌های استاد و مطالب صفحه 6 اسلاید: " برای transmit کردن دیتا میباشد.(Transmit Serial Data) اگر ما به عنوان USART یا UART از آن کنترلر استفاده کنیم. اما USART controller میتواند در نقش SPI controller هم باشد. اگر ما ببریمش به مود SPI در اینصورت اگر آن ماژول SPI به عنوان Master باشد ، این پایه‌ی TXD میشود همان MOSI ما و اگر این ماژول به عنوان Slave باشد آنگاه این پایه‌یTXD میشود MISOی موردنظر ما."

پایه RXD : پایه‌ای است که برای دریافت دیتا استفاده میشود. این پایه به TXD ماژول دیگر و فرستنده وصل میشود و وقتی آن از طریق TXD خود دیتا را ارسال میکند و در پایه‌ی RXD خود آن را دریافت میکنیم.

طبق صحبت‌های استاد و مطالب صفحه 6 اسلاید: " برای receive کردن دیتا میباشد.(Recieve Serial Data) . اگر ما ببریمش به مود SPI در اینصورت اگر آن ماژول SPI به عنوان Master باشد ، این پایه‌ی RXD میشود همان MISO ما و اگر این ماژول به عنوان Slave باشد آنگاه این پایه‌یRXD میشود MOSIی موردنظر ما."

پایه CTS : مخفف Clear To Send میباشد. یک پورت ورودی به USART است. Active Low میباشد. درواقع سیگنالی هست که برای Hand Shaking استفاده میشود و از طریق آن فرستنده میفهمد که گیرنده آمادگی دریافت دیتا را الان دارد و بافراش خالی هست و اگر دیتا براش بفرستیم میتواند هندل کند. پس فرستنده وقتی میخواهد دیتا بفرستد اول CTS که ورودیش هست را چک میکند اگر 0 شده باشد آنگاه یعنی ماژولی در سمت دیگر آمادگی دریافت را دارد و بنابراین دیتا ارسال میشود.

اگر هم در مود SPI باشیم و ماژول به عنوان Slave باشد آنگاه به عنوان Slave Select (NSS) استفاده میشود.

پایه RTS : مخفف Request To Send میباشد. یک پایه‌ی خروجی (output) میباشد. این سیگنال از سمت Receiver، زمانی که آمادگی دریافت دیتا را داشته باشد، فعال میشود. این پایه‌ی RTS درگیرنده به CTS در فرستنده وصل میشود و گیرنده در صورت آمادگی روی 0 میگذارد تا فرستنده براش دیتا بفرستد.

اگر هم در مود SPI باشیم و ماژول به عنوان Master باشد آنگاه به عنوان Slave Select (NSS) برای Master استفاده میشود و به کمک آن میتواند Slave مورد نظر خود را انتخاب و فعال کند که آماده‌ی دریافت اطلاعات شود.

جواب به صورت خلاصه در جدول زیر که از صفحه 6 لکچر10 برداشته شده است آمده است:

Name	Description	Type	Active Level
SCK	Serial Clock	I/O	
TXD	Transmit Serial Data or Master Out Slave In (MOSI) in SPI Master Mode or Master In Slave Out (MISO) in SPI Slave Mode	I/O	
RXD	Receive Serial Data or Master In Slave Out (MISO) in SPI Master Mode or Master Out Slave In (MOSI) in SPI Slave Mode	Input	
CTS	Clear to Send or Slave Select (NSS) in SPI Slave Mode	Input	Low
RTS	Request to Send or Slave Select (NSS) in SPI Master Mode	Output	Low

- مهلت ارسال تمرین ساعت 23.59 روز 28 اردیبهشت می‌باشد.
- سوالات خود را می‌توانید تنها از طریق ایمیل زیر بپرسید.
  - AUTMicroTA@gmail.com

- ارائه پاسخ تمرین به سه روش ممکن است:
  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 **HW1-G#-9531***.pdf** در مودل بارگزاری کنید.
  - نمونه: HW1-G2-9531747
- فایل زیپ ارسال **نکنید**.