دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

3/15/2021

Homework 2

Lec 5-6

MICROPROCESSOR
AND
ASSEMBLY LANGUAGE

Spring 2021

1) از کدام رجیسترها برای پیشگیری از وقفه ها استفاده می‌کنیم؟ هر یک را به اختصار توضیح دهید.

رجیسترهای PRIMASK و BASEPRI و FAULTMASK برای پیشگیری از وقفه ها استفاده می شوند.

PRIMASK: از این رجیستر برای mask کردن اینتراپت های configurable استفاده میشود. یک رجیستر یک بیتی است که اگر set شود (مقدار آن 1 بشود) جلوی فعال کردن همه‌ی exceptionهای با اولویت های configurable را می‌گیرد. درواقع هیچ کدام از اینتراپت‌ها به جزReset و NMI و Hard Fault Exception نمیتوانند مزاحم شوند و وقفه ایجاد کنند. در حقیقت میتوان گفت اگر PRIMASK روی 1 تنظیم شود اولویت process درحال اجرا را به 0 می‌رساند.

BASEPRI : به طور خلاصه برای mask کردن وقفه ها از یک اولویت مشخص به بعد به کار می رود.

یک رجیستر 8 بیتی است که به کمک آن 8 بیت یکی از شماره های اولویت را مشخص میکنیم. همه ی اینتراپت‌هایی که اولویتشان از آن شماره و به بالا میباشد را Mask میکند و نمیگذارد مزاحم کار ما شوند. درواقع BASEPRI مینیمم اولویت را برای exception processing مشخص میکند و وقتی روی یک مقدار غیر صفر تنظیم میشود از فعال شدن همه‌ی وقفه ها با سطح اولویت مساوی یا کوچکتر از مقدار BASEPRI جلوگیری میکند.(سطح اولویت پایین تر یا همان شماره‌ی اولویت بزرگ تر)

دقت شود که اگر مقدار آن را 0 تنظیم کنیم کلا عملیات mask کردن را به کمک BASEPRI درواقع disable میکند.

FAULTMASK: یک رجیستر یک بیتی میباشد . شبیه PRIMASK میباشد با این تفاوت که Hard Fault Exception نیز نمیتواند اینتراپت دهد و مزاحم شود.

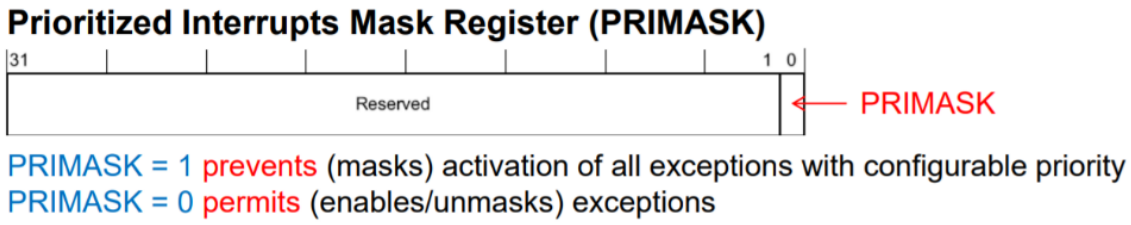
پس درواقع اگر این رجیستر ست شود اینتراپت های با شماره‌ی اولویت1- به بالا نمیتوانند برنامه را متوقف کنند و Mask میشوند. در واقع با ست شدن آن اولویت برنامه‌ی درحال اجرا 1- میشود. پس به جز NMI و Reset اینتراپت‌های دیگه نمیتوانند مزاحم شوند.

2) پیکربندی رجیستر primask از اجرای کدام وقفه ها میتواند پیشگیری کند؟ چگونه اینکار را انجام می‌دهد؟ هنگام استفاده از آن چه وقفه‌هایی می‌تواند رخ دهد؟

پیکربندی رجیستر primask از اجرای همه‌ی وقفه های configurable و درواقع آن هایی مقدار priority آن‌ها از 0 به بالا باشد میتواند پیشگیری کند.

یک رجیستر یک بیتی است (نه به این مفهوم که یک رجیستر با 1 بیت باشد بلکه بیت اول از یک رجیستر 32 بیتی برای آن است)و زمانی که set شود (مقدار آن 1 بشود) اولویت process درحال اجرا را به 0 می‌رساند. ازطرفی میدانیم وقتیپردازنده درحال سویس دادن به یک وقفه میباشد چنانچه وقفه ای با اولویت کم تریا مساوی (شماره‌ی اولویت بزرگ تر یا مساوی) بیایددر این حالت پردازنده آن وقفه ی جدید را نادیده میگیرد و روال قبلی را متوقف نمیکند. حال در اینجا هم وقتی شماره اولویت process ما 0 شود دیگر از وقفه های با شماره اولویت بزرگتر مساوی 0 پیشگیری میشود.

در زمان استفاده از آن وقفه های با اولویت های 1- و 2- و3- میتوانند رخ هند چرا که اولویت بالاتری دارند. این وقفه ها به ترتیب عبارتند از: Hard fault exception وNon-Maskable Interrupt (NMI) و Reset. اگر هرکدام از این 3 وقفه رخ دهند، پردازنده موظف است روال قبلی را متوقف کند و به آن ها رسیدگی کند.



3) فرض کنید در یک سیستم تنها یک نوع وقفه داریم. حال در این سیستم فرضی CPU با فرکانس ۱۵۰مگاهرتز کار می‌کند. مجموع کلاک‌هایی که برای ذخیره کردن اینتراپت‌ها در حافظه و برگرداندن آن‌ها نیاز داریم هم ۱۰۰ تا می‌باشد. حال اگر در هر ثانیه این وقفه ۱۵۰هزار بار به سیستم بیاید برای اینکه بتوانیم به این وقفه رسیدگی کنیم باید اجرای ISR آن حداکثر به چند کلاک نیاز داشته باشد؟

ابتدا داده های صورت سوال را مینویسیم:

فرکانس CPU : $F_{cpu} = 150\ MHz$

مجموع کلاک‌هایی که برای ذخیره کردن اینتراپت‌ها در حافظه و برگرداندن آن‌ها نیاز داریم: $C_{overhead} = 100$

فرکانس اینتراپت : $F_{int} = 150\ KHz$

حال ما میخواهیم بیشینه مقدار C_{ISR} (تعداد کلاک های موردنیاز برای رسیدگی برای اجرای ISR) را بیابیم.

طبق فرمول داریم :

$$F_{int} = \frac{F_{cpu}}{C_{ISR} + C_{overhead}} \rightarrow C_{ISR} = \frac{F_{cpu}}{F_{int}} - C_{overhead}$$

$$\rightarrow C_{ISR} = \frac{150\ MHz}{150\ KHz} - 100 = 900$$

پس بیشینه تعداد کلاک های موردنیاز برای رسیدگی برای اجرای ISR طبق روابط بالا 900 کلاک میباشد.

(اگر مقدار کلاک کم تر شود حتی اگر فرکانس اینتراپت بالاتر هم برود میتوانیم باز به آن رسیدگی کنیم ولی بیشینه مقدار کلاک برای این مقدار از فرکانس اینتراپت همین مقدار 900 کلاک میباشد.)

4) توضیح دهید که Glitch Filter و Debouncing Filter چه قابلیت‌هایی هستند و چه کاری انجام می‌دهند

Glitch Filter :

Glitch Filtering فرآیند حذف پالسهای ناخواسته از یک سیگنال ورودی دیجیتال است که میتواند زیاد یا کم باشد. تداخل الکتریکی و یا حتی در برخی موارد حتی مکانیکی می تواند باعث ایجاد پالس glitch ناخواسته از گیرنده شود.

درGlitch Filter زمانی خروجی 1 میشود که نمونه‌ی کنونی و N نمونه‌ی قبلی 1 بوده باشند و زمانی خروجی 0 میشود که نمونه‌ی کنونی و N نمونه‌ی قبلی 0 بوده باشند در غیر اینصورت خروجی روی همان مقدار فعلی میماند و تغییر نمیکند. این مقدار N و طول فیلترینگ نیز قابل برنامه ریزی میباشد.

همچنین یکی از I/O mode ها میباشد که درصورتی که آن را برای یک پین فعال کنیم پالس های ناخواسته را از سیگنال ورودی دیجیتال ما حذف میکند. پس درواقع کامپوننتی هست که میتوان با هر ورودی دیجیتال استفاده کرد.

Debouncing Filter:

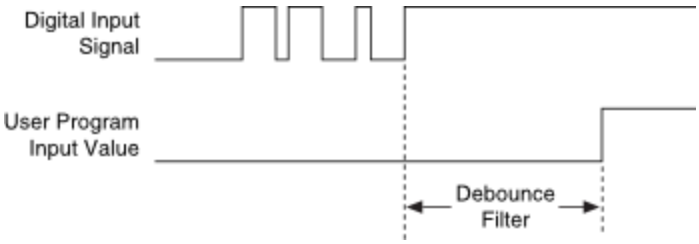
میدانیم یکی از I/O mode ها مود debouncing میباشد. برای هر پین ورودی دیجیتال میتوان debounce filter را فعال کرد.

Debouncing Filter یک تایمر است که میتواند سوییچ‌های مکانیکی را حذف کند یا noise و transition ها را فیلتر کند. تایمر فیلتر از لبه‌ی بالارونده یا پایین رونده‌ی سیگنال ورودی فیلتر نشده آغاز میشود.. برنامه User مقدار قبلی سیگنال را برای مدت زمان فیلتر می خواند. بعد از سپری شدن زمان فیلتر و عدم وجود لبه های جدید روی سیگنال ورودی ، برنامه User مقدار سیگنال جدید را می خواند. تایمر فیلتر در لبه بعدی سیگنال ورودی فیلتر نشده دوباره راه اندازی می شود.

برای مثال وقتی ما یک دکمه‌ی کیبورد را فشار میدهیم، به صورت مداری اتصال ممکن است در آن حین هزاران بار قطع و وصل شود تا ثابت بماند و درواقع یک بار فشار دادن دکمه توسط ما برای پردازنده فقط یک بار ارسال

سیگنال نیست. این debouncing filter این تغییرات مداوم را فیلتر میکند و باعث میشود با وجود آن همه قطع و وصل شدن ، به میکروی کا فقط یک بار خبر داده شود که دکمه فشار داده شده است.

برای مثال در شکل زیر میتوانید debounce filter را روی ورودی Active High ببینید.



5) Race Condition چیست؟ با آوردن مثال توضیح دهید چرا در هنگام دسترسی به داده ها با آن روبه‌رو می‌شویم و راه حلی برای پیشگیری از این مشکل ارائه دهید.

یک Race Condition زمانی ایجاد میشود که یک برنامه‌ی کامپیوتری برای اینکه به درستی کار کند به ترتیب یا زمانبندی پردازه های برنامه و thread ها بستگی داشته باشد. Race Condition باعث اجرای نادرست برنامه و به وجود آمدن خروجی‌های نادرست میشود.

در واقع Race Condition زمانی ایجاد میشود که پردازه ها و فانکشن های ما به یک سری حالت ها و دیتاهای shared وابسته باشند.

مثال:

فرض کنید دو تا فانکشن داریم. یکی GetDateTime است که هر وقت call شود به زمان را می دهد. یک فانکشن هم DateTimeISR میباشد. در واقع ISR مربوط به یک اینتراپت میباشد که هر یه ثانیه یک بار صدا زده میشود و این ISR اجرا میشود و باعث میشود به متغیر زمان ما یک ثانیه اضافه شود و در صورتی که ثانیه به 60 رسید به دقیقه یکی اضافه کند و اگر دقیقه به 60 رسید یکی به ساعت و اگر ساعت به 23 رسید یکی به روز اضافه کند. هر دو فانکشن یک ساختار(struct) shared دارند که TimerVal میباشد که یک سری فیلد ثانیه و دقیقه و ساعت و روز دارد. در زیر میتوانید این دو قطعه کد را مشاهده کنید:

```
void GetDateTime(DateTimeType * DT){
 DT->day = TimerVal.day;
 DT->hour = TimerVal.hour;
 DT->minute = TimerVal.minute;
 DT->second = TimerVal.second;
}
```

```
void DateTimeISR(void){
 TimerVal.second++;
 if (TimerVal.second > 59){
   TimerVal.second = 0;
   TimerVal.minute++;
   if (TimerVal.minute > 59) {
     TimerVal.minute = 0;
     TimerVal.hour++;
     if (TimerVal.hour > 23) {
     TimerVal.hour = 0;
       TimerVal.day++;
       … etc.
     }
```

اگر اینتراپت در زمانی که درحال خواندن تایم به کمک تابع GetDateTime هستیم بیاید باعث میشود که دیتای داخل DT به صورته نصفه آپدیت شود و همه ی فیلدها آپدیت نشوند.

یعنی وقتی یک سری از فیلدهای struct مشترک TimerVal را که می خوانیم و داخل DT آپدیت کردیم و برای مثال در زمان {23:59:59, 10^{th} day } تابع GetDateTime صدا زده میشود و بعد تا خط سوم تابع GetDateTime می رویم و درواقع داخل متغیر های **DT->day** و **DT->hour** به ترتیب 10 و 23 ذخیره میشود. در همین لحظه اینتراپت می آید که تایم آپدیت شود و سراغ ISR مربوط به آن میرود و پس از اجرای آن تایم میشود: {0:0:0, 11^{th} day }

حال وقتی برمیگردیم که ادامه ی تابع قبل را اجرا کنیم داخل متغیرهای **DT->minute** و **DT->second** به ترتیب مقادیر 0 و 0 قرار میگیرد و در آخر چیزی که به عنوان خروجی این تابع به ما گزارش میشود مقدار زیر میباشد: {23:0:0 , 10^{th} day }

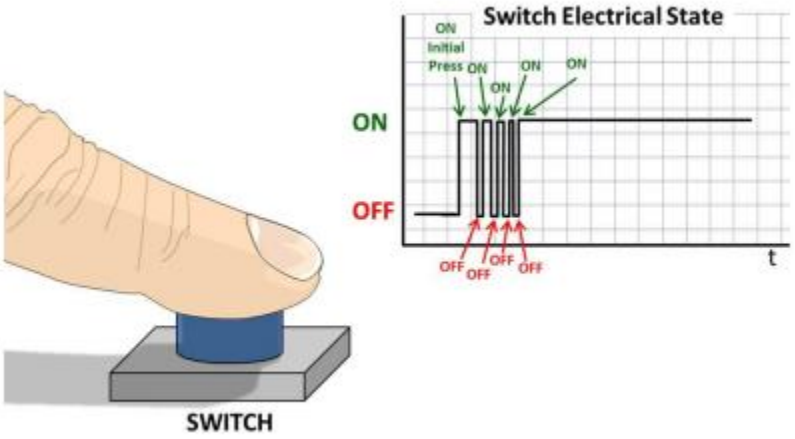
همانطور که مشاهده میشود مقداری غلط میباشد. این موضوع به خاطر Race Condition میباشد.

راه حل:

راه حل این دست که دسترسی به دیتاهای مشترک را atomic کنیم. در این حالت کار ما در یک فانکشن با این متغیر آغاز شد باید تا آخرش برویم و نگذاریم کسی در این بین اینتراپت دهد و روی این دیتای مشترک over write انجام دهد. برای این کار باید اینتراپت را غیر فعال کنیم و در آخر وضعیت اینتراپت ها را به همان حالت قبل برمیگردانیم.

```
void GetDateTime(DateTimeType * DT){
 uint32_t m;
 m = __get_PRIMASK();
 __disable_irq();
 DT->day = TimerVal.day;
 DT->hour = TimerVal.hour;
 DT->minute = TimerVal.minute;
 DT->second = TimerVal.second;
 __set_PRIMASK(m);
}
```

6) تصویر زیر پدیده‌ی Switch Bounce را نشان می‌دهد. نمونه‌ای از مشکلات احتمالی ناشی از این نوسان‌ها را بیان کنید و برای دوری از این مشکلات چه راه‌حلی وجود دارد؟ توضیح دهید.



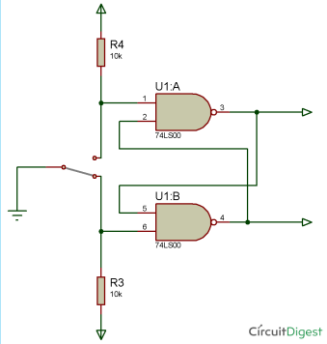
وقتی pushbutton را فشار میدهید تا اتصال برقرار شود در واقع در مدار آن دوتا قطعه‌ی فلزی به هم وصل شوند تا اتصال موردنظر برقرار شود. ولی این دوتا قطعه بلافاصله با فشردن دکمه به هم وصل نمیشوند بلکه وقتی خیلی نزدیک میشوند جرقه هایی زده میشود و چندین بار قطع و وصل میشوند تا سپس به حالت پایداری برسند و اتصال انجام شود. این موضوع هنگام رها کردن دکمه نیز رخ میدهد. این موضوع باعث false triggering و multiple triggering میشود که درواقع انگار دکمه چندین بار فشرده شده است. البته از دید کاربر انگار همون لحظه اتصال برقرار شده است چون این قطع و وصل شدن ها شاید در حد چند میکرو ثانیه رخ دهد. ولی از دید سخت افزار درواقع چند بار قطع و وصل شده است.

خب این موضوع میتواند دلخواه ما نباشد و در یک مدار دیجیتال که با 0 و 1 کار داریم این موضوع باعث قطع و وصل شدن قطعه‌ی متصل میشود (برای مثال اگر یک LED به آن وصل باشد چندین بارخاموش و روشن میشود تا به حالت پایدار موردنظر برسد. که البته از دید کاربر مشخص نمیشود ولی برای کارهای حساس از دید سخت افزاری همچین مشکلی وجود دارد) و میتواند یک پروژه ی الکترونیکی را خراب کند.

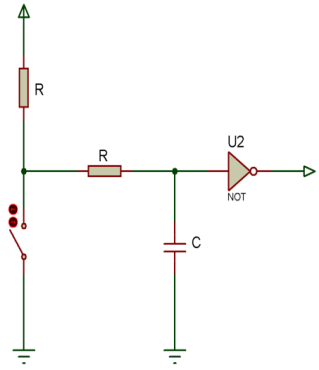
برای رفع این مشکل در میکرویی که استفاده میکنیم برای هر پین میتوان از debouncing filter استفاده کرد و باید به کمک تنظیمات آن را برای هرپین دیجیتال ورودی دلخواه فعال کرد تا از این مشکل جلوگیری کند.

در حالت کلی برای debouncing کردن و جلوگیری از switch bouncing طبق سایتی که در انتهای جواب این سوال قرار گرفته است، راه های زیر وجود دارند:

1. Hardware Debouncing
2. RC Debouncing
3. Switch Debouncing IC

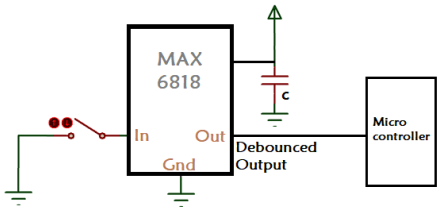


- در روش Hardware Debouncing از یک S-R flip flop برای جلوگیری از switch bouncing استفاده میشود. مقاومت مورد استفاده در این مداربه صورت pull-up میباشد. اگر bouncing رخ دهد، فلیپ فلاپ خروجی را تغییر نمی دهد چراکه منطق 0 از خروجی به داخل گیت‌های NAND برمیگردد. درواقع فقط یک لحظه را میگیرد و قبل و بعد آن در نظر گرفته نمیشود و یک موج تمیز به عنوان ورودی میدهد



- در روش RC Debouncing همانطور که از نامش پیداست از یک مدار RC برای جلگیری از این مشکل استفاده میکند. خازن موجود در مدار تغییرات ناگهانی در سیگنال سوییچ را فیلتر میکند.درواقع اون قطع و وصل شدن ها را به میکرو منتقل نمیکند و با آن ها فقط خازن شارژ میشود.

- در روش Switch Debouncing IC دربازار IC هایی برای Debouncing وجود دارد مانند MAX6816 که میتوان از آن ها نیز استفاده کرد.



https://circuitdigest.com/electronic-circuits/what-is-switch-bouncing-and-how-to-prevent-it-using-debounce-

circuit#:~:text=When%20we%20press%20a%20pushbutton,actual%20stable%20connection%20is%20made.

همچنین در یکی از ویدیو های آزمایشگاه گفته میشود که یک روش نرم افزاری وجود دارد و یک delay در نظر گرفته میشود و از زمانی که بار اول برای مثال سیگنال یک میشود هرچی بگذرد و از آن تاخیر گفته شده کمتر باشد ما همچنان سیگنال را 0 میگیریم و درواقع در اون زمان ما تغییرات و قطع و وصل شدن ها را تا برقراری تماس کامل درنظر نمیگیریم.

7) چرا در PIO_ODSR برخلاف بقیه‌ی رجیسترهای وضعیت دسترسی نوشتن داریم؟

ما در فرستادن دیتا روی پایه ها نیاز داریم تا به صورت parallel دیتا روی آن ها بفرستیم. یعنی اگر میخواهیم یک دیتای n بیتی بفرستیم باید n بیت به صورت سنکرون همه باهم همزمان روی پایه ها ظاهر شوند.

میدانیم که برای هر بیت دیتا باید در حالت عادی مقادیر PIO_SODR و PIO_CODR را تنظیم کنیم تا از ترکیب آن ها مقدار رجیستر PIO_ODSR جنریت و تولید شود و سپس دیتا از روی آن منتقل شود به خروجی. خب در این n بیت دیتا، یک سری بیت ها 0 و یک سری 1 هستند. برای 0 ها باید بیت متناظر در PIO_CODR برابر 1 شود و برای 1 ها باید بیت متناظر در PIO_SODR برابر 1 شود. از آنجایی که نمیتوان همزمان هم روی PIO_CODR و هم روی PIO_SODR مقدار نوشت در اینصورت ممکن است مثلا در پین های خروجی اول 0 ها را فقط ببینیم و بعد از زمانی 1 ها را ببینیم. خب این موضوع باعث اذیت و کندی میشود.

برای رفع این مشکل ما اجازه داریم که برخلاف بقیه ی رجیسترها مستقیما روی PIO_ODSR دیتا بنویسیم

- مهلت ارسال تمرین ساعت 23.55 روز 15 فروردین می‌باشد.
- سوالات خود را می‌توانید تنها از طریق ایمیل AUTMicroTA@gmail.com بپرسید.
- ارائه پاسخ تمرین بهتر است به روش های 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 **HW2 - 9731***.pdf** در مودل بارگزاری کنید.
  - نمونه: HW2- 9731747
- فایل زیپ ارسال **نکنید**.