«به نام پروردگار مانا»

۱.

SRAM : در میکروی ما SRAM برای ذخیره ی داده ها استفاده می شود.

Flash : از فلش برای ذخیره برنامه و کدها استفاده می شود (فضای مختص برنامه ها)

EEPROM : از EEPROM نیز مانند SRAM برای ذخیره ی data استفاده می شود اما داده های نگهداری شده در EEPROM داده های حیاتی و data هایی می باشند که میخواهیم پس از خاموش شدن و قطع شدن برق سیستم از بین نروند.

یک شباهتی که فلش و EEPROM دارند آن هست که هردو غیرفرار(non-volatile) هستند.

تفاوت های Flash و EEPROM :

مزیت فلش آن هست که فضای کمی را در بر می گیرد ولی مشکل آن این هست که نوشتن (write شدن) در آن مستلزم آن است که همه ی داده های موجود در آن پاک شوند؛ به بیان دیگر وقتی میخواهیم یک داده ای را در یک بلاک از فلش تغییر دهیم، همه ی آن بلاک را باید پاک کنیم یعنی فرآیند پاک شدن در فلش بلاک به بلاک هست و نه بایت به بایت. پس استفاده از آن برای به روز رسانی داده ها (data) مناسب نیست ولی برای instruction خوب هست، زیرا یک برنامه را نیاز نیست دائم به روز رسانی کنیم و اگر هم نیاز به update باشد، باید کل برنامه را update کنیم.

یک مزیت دیگر فلش هم آن است که density بیشتری دارد و به نسبت به فضایی که می گیرد حجم بایتی بیشتری را نسبت به EEPROM شامل می شود.

درحالی که read و write در EEPROM، می تواند بایت به بایت انجام شود و برای به روز رسانی یک داده محدودیت بلاکی (block) نداریم.

تفاوت SRAM و EEPROM :

تفاوت عمده ی این دو آن هست که با قطع شدن انرژی و برق سیستم، اطلاعات موجود در EEPROM از بین نمی رود ولی اطلاعات SRAM پاک می شود. از طرفی مزیت SRAM آن است که سریع تر است و در آن با سرعت بالاتری می توان به داده ها دسترسی داشت و همچنین انرژی کمتری مصرف می کند.

از طرف دیگر EEPROM، محدودیت override شدن دارد و به تعداد محدودی (مثلا ۱۰ هزار بار) میتوان آنرا update کرد و مدام آنرا پاک کرد و روی آن نوشت ولی SRAM این محدودیت را ندارد.

۲.

در memory-mapped i/o، پردازنده با دستگاه های i/o درست مانند مکان هایی در حافظه برخورد می کند.در این روش برای پردازنده تفاوتی نمی کند که داده ها را از memory میخواهد تحویل بگیرد یا i/o؛ درواقع پردازنده تنها با memory سر و کار دارد و همه چیز را در قالب حافظه می بیند. پردازنده تنها با درستورات load/store کار می کند و هر دستگاه خارجی و که می خواهد با پردازنده داده رد و بدل کند، باید داده ی خود را در فضای حافظه قرار دهد.

دستگاه های i/o در سیستم memory، map می شوند و پردازنده از یک سری باس مشترک استفاده می کند. دستورات Load/Store برای خواند data از یک دستگاه i/o یا تحویل دادن data به آن استفاده می شوند.

مزیت های memory-mapped I/O :

- این روش رابط کاربری به ما یک تک فضای آدرس دهی می دهد تا به خوبی بتوانیم به راحتی از مجموعه ای از دستورات برای هردوی memory و عملگرهای i/o استفاده کنیم.
- استفاده از یک مجموعه opcode متفاوت برای دستورات i/o نیاز نیست و میتوانیم از همان دستورات دسترسی به memory استفاده کنیم.(دستورات کتری نسبت به isolated I/O نیاز دارد)
- از همان memory-mapping ای که برای memory استفاده می کردیم میتوانیم برای دسترسی به سایر دستگاه ها استفاده کنیم.
- از آنجایی که هیچ جابجایی (switching) ای میان دو فضای آدرس دهی وجود ندارد، نیازی به سیگنال های کنترلی جداگانه نیست. ما تنها یک فضای آدرس دهی یکپارچه برای memory و دستگاه های i/o داریم. که این کار به صرفه جویی در زمان کمک می کند.(برعکس isolated I/O که نیاز به خط سیگنال کنترلی جداگانه برای switch کردن میان فضاهای آدرس دهی متفاوت که در I/O mapped I/O ساخته ایم بشود)

از معایب آن نیز می توان گفت :

- سراسر address bus باید کاملا برای هر دستگاه خارجی (peripheral)، decode شود. برای مثال یک ماشین با یک address bus، ۶۴ بیتی، برای مشخص کردن وضعیت هرکدام از ۶۴ خط نیاز به گیت های منطقی دارد؛ تا آدرس بخصوص هرکدام از Peripheral ها را به درستی مشخص کند
- از I/O mapped-I/O کندتر عمل می کند.
- در memory-mapping، بخشی از فضای حافظه صرف آدرس های ورودی و خروجی می شود و فضای کمتری برای حافظه باقی می گذارد نسبت به isolated I/O.

هم چنین در یک مقایسه کلی میتوان گفت که :

در isolated-I/O برنامه نویس واضح تر هست؛ I/O خط آدرس جداگانه ی خود را دارد، بنابراین پردازنده می تواند از آن استفاده کند تا مشخص شود که آدرس memory است یا I/O که برنامه باید با آن کار کند. درصورتی که در memory-mapping هر دستوری که به memory اشاره می کند، می تواند به عنوان یک دستور به سیستم I/O تعبیر شود که باعث سردرگمی در memory-mapped I/O ها می شود.همچنین اختصاص دادن آدرس های I/O در isolated I/O راحت تر هست چون دستگاه های I/O فضای کمتری نسبت به memory در یک آدرس داده شده می گیرند.

.۳

```
ADDS R4,R2,R0
ADDC R5,R3,R1
```

.۴

فایل کد زده شده در ویژوال، به پوشه پیوست شده است.

```
START  B        FUNC
MAIN    END
FUNC    MOV     R0, #'3'
        MOV     R1, #'0'
        MOV     R2, #'9'
        CMP     R0, R1
        BCC     ENDP
        CMP     R2, R0
        BCC     ENDF
        MOV     R0, #0
ENDF    B       MAIN
```

۵.

فایل کد زده شده در ویژوال، به پوشه پیوست شده است.

الف)

```
LDR     R1, =0x400E0E60 ; load address of PIOA PIO_PUDR in R1
LDR     R2, =0x00000080 ; value of PIO_PUDR in PIOA8
LDR     R3, [R1] ; load value of register PIO_PUDR to R3
ORR     R4, R3, R2
STR     R4, [R1] ; disable pull up for 8th pin of PIOA
LDR     R1, =0x400E0E54 ; load address of PIOA PIO_MDDR in R1
LDR     R2, =0x00000080 ; value of PIO_MDDR in PIOA8
LDR     R3, [R1] ; load value of register PIO_MDDR to R3
ORR     R4, R3, R2
STR     R4, [R1] ; disable Multi Drive for 8th pin of PIOA
LDR     R1, =0x400E0E10 ; load address of PIOA PIO_OER in R1
LDR     R2, =0x00000080 ; value of PIO_OER in PIOA8
LDR     R3, [R1] ; load value of register PIO_OER to R3
ORR     R4, R3, R2
STR     R4, [R1] ; enable Out Put for 8th pin of PIOA
```

۵) ب)

```
LDR     R1, =0x400E0E64 ; load address of PIOA PIO_PUER in R1
LDR     R2, =0x00000080 ; value of PIO_PUER in PIOA8
LDR     R3, [R1] ; load value of register PIO_PUER to R3
ORR     R4, R3, R2
STR     R4, [R1] ; enable pull up for 8th pin of PIOA
LDR     R1, =0x400E0E14 ; load address of PIOA PIO_ODR in R1
LDR     R2, =0x00000080 ; value of PIO_ODR in PIOA8
LDR     R3, [R1] ; load value of register PIO_ODR to R3
ORR     R4, R3, R2
STR     R4, [R1] ; disable Out Put for 8th pin of PIOA
LDR     R1, =0x400E0E00 ; load address of PIOA PIO_ PER in R1
LDR     R2, =0x00000080 ; value of PIO_ PER in PIOA8
LDR     R3, [R1] ; load value of register PIO_ PER to R3
ORR     R4, R3, R2
STR     R4, [R1] ; disable pull up for 8th pin of PIOA
```

۶.

قطعه کد زیر دو عدد را باهم مقایسه می کند و عدد بزرگتر را در حافظه ذخیره می کند.

در اینجا مقادیر رجیستر های ما برابر است با :

R_4 = 0x20000000

R_0 = حافظه 0x20000000 مقدار موجود در خانه ی

R_4 = 0x20000004

R_1 = حافظه 0x20000004 مقدار موجود در خانه ی

R_4 = 0x20000008

اگر R_0 بزرگتر از R_1 بود، مقدار R_0 را در R_2 بریز و سپس R_2 را در خانه ی حافظه با آدرس موجود در R_4 قرار بده؛ اگر R_0 کوچکتر از R_1 بود، مقدار R_1 را در R_2 بریز و سپس R_2 را در خانه ی حافظه با آدرس موجود در R_4 قرار بده.

خوب هست به این نکته نیز اشاره شود که دستورات MOVGT و MOVLE در این کد دستورات conditional هستند که درواقع با گذاشتن کد پسوند های شرطی (condition code suffixes) مانند LE و GT پس از آنها بوجود می آید. در اینجا اگر R0>R1 باشد، پس از دستور SUBS و set شدن flagهای z و N و v اگر z = 0 بود و N = v بود، MOVGT اجرا می شود ولی اگر R0<R1 بود، یعنی z=0 و N!=v شده، پس MOVLE اجرا می شود.

۷.

فایل کد زده شده در ویژوال، پیوست شده است.

```
MYINT DCD       10, 10, 12
MYCHAR  DCB     'a', 'a'
        ADR     R0, MYINT
        LDR     R1, [R0]
        ADD     R0, R0, #4
        LDR     R2, [R0]
        ADD     R0, R0, #4
        LDR     R5, [R0]
        ADR     R6, MYCHAR
        LDRB    R3, [R6]
        ADD     R6, R6, #1
        LDRB    R4, [R6]
        CMP     R3, R4
        BNE     ENDP
        CMP     R1, R2
        BNE     ENDP
        ADD     R5, R5, #1
```

```
        STR     R5, [R0]
ENDP    END
```

۸.

در دستور اول ما مقدار رجیستر R1 را برابر مقدار 0x11121314 می کنیم. سپس آدرس 0x40000 را در رجیستر R2 می ریزیم و بعد از آن مقدار 0x11121314 را در چهار بایت از حافظه که آدرس شروع آنها 0x40000 هست می ریزیم.

اگر بخواهیم تنها مقدار آدرس 0x40000 را در نظر بگیریم (اولین خانه یا بایتی که این آدرس به آن اشاره می کند) مقدار آدرس 0x40000 برابر0x00000014 خواهد بود؛ اما اگر یک کلمه word را درنظر بگیریم (یعنی از 0x40000000 تا 0x40000004) مقدار آن برابر با 0x11121314 می شود.

و مقدار رجیستر R3 چون تنها یک بایت را می خواند برابر 0x00000014 می شود.

۹.

فایل کد زده شده در ویژوال، پیوست شده است.

```
                MOV     R1, #10 ;COUNTER
                MOV     R0, #0
START_OF_LOOP   ADD     R0, R0 , #1
                SUBS    R1, R1, #1
                BEQ     END_OF_LOOP ; if (R1 == 0) break
                B       START_OF_LOOP
END_OF_LOOP     END
```

پاسخ دیگری برای سوال ۵ با توجه به تمرین دوم می تواند این باشد :

۵.

فایل کد زده شده در ویژوال پیوست شده است.(لطفا آن فایل به عنوان جواب اصلی درنظر گرفته شود این کد موجود در pdf کمی اضافه تر و بر مبنای تمرین دوم نوشته شده)

```
LDR R1, =0x400E0E60 ; load address of PIOA PIO_PUDR in R1
LDR R2, =0x00000080; value of PIO_PUDR in PIOA8
LDR R3, [R1]; load value of register PIO_PUDR to R3
ORR R4, R3, R2
STR R4, [R1] ; disable pull up for 8th pin of PIOA
LDR R1, =0x400E0E54 ; load address of PIOA PIO_MDDR in R1
LDR R2, =0x00000080; value of PIO_MDDR in PIOA8
LDR R3, [R1]; load value of register PIO_MDDR to R3
ORR R4, R3, R2
STR R4, [R1] ; disable Multi Drive for 8th pin of PIOA
LDR R1, =0x400E0E10 ; load address of PIOA PIO_OER in R1
LDR R2, =0x00000080 ; value of PIO_OER in PIOA8
LDR R3, [R1]; load value of register PIO_OER to R3
ORR R4, R3, R2
STR R4, [R1] ; enable Out Put for 8th pin of PIOA
```

ب)

```
LDR R1, =0xE000E100; load address of NVIC ISER0 in R1
LDR R2, =0x00000400 ; setting port A interrupt enable in ISER0
LDR R3, [R1]; load value of register ISER0 to R3
```

```
ORR R4, R3, R2
STR R4, [R1] ; STORE new values of ISER0 in memory
LDR R1, =0xE000ED0C; load address of AIRCR in R1
LDR R2, =0x05FA0500 ; desired value for AIRCR in R2
LDR R3, [R1]; load value of register AIRCR to R3
ORR R4, R3, R2 ; setting desired value for AIRCR in R2
STR R4, [R1] ; STORE new values of AIRCR to memory
LDR R1, =0xE000E40C; load address of IPR2 in R1
LDR R2, =0xF0000000 ; desired value for IPR2 in R2
LDR R3, [R1]; load value of register IPR2 to R3
ORR R4, R3, R2 ; setting desired value for IPR2 in R2
STR R4, [R1] ; STORE new values of IPR2 to memory
LDR R1, =0xE000E40C; load address of IPR2 in R1
LDR R2, =0xF0000000 ; desired value for IPR2 in R2
LDR R3, [R1]; load value of register IPR2 to R3
ORR R4, R3, R2 ; setting desired value for IPR2 in R2
STR R4, [R1] ; STORE new values of IPR2 to memory
LDR R1, =0x400E0E64; load address of PIOA PIO_PUER in R1
LDR R2, =0x00000080; value of PIO_PUER in PIOA8
LDR R3, [R1]; load value of register PIO_PUER to R3
ORR R4, R3, R2
STR R4, [R1] ; enable pull up for 8th pin of PIOA
LDR R1, =0x400E0E14; load address of PIOA PIO_ODR in R1
LDR R2, =0x00000080; value of PIO_ODR in PIOA8
```

```
LDR R3, [R1]; load value of register PIO_ODR to R3
ORR R4, R3, R2
STR R4, [R1] ; disable Out Put for 8th pin of PIOA
LDR R1, =0x400E0E00; load address of PIOA PIO_ PER in R1
LDR R2, =0x00000080; value of PIO_ PER in PIOA8
LDR R3, [R1]; load value of register PIO_ PER to R3
ORR R4, R3, R2
STR R4, [R1] ; disable pull up for 8th pin of PIOA
```