تمرین دهم ریزپردازنده

ارمغان سرور ۹۵۳۱۸۰۷

سوال ۱

buffer مقادیر مختلف را ابتدا تقویت و یا از آن محافظت می‌کند و سپس عبور می‌دهد. اما latch با دریافت مقادیر، آن ها را در خود نگه می‌دارد.از Latch برای پورت خروجی و از buffer برای پورت های ورودی میکرو استفاده می‌کنیم. در حالت کلی نمی‌توان زیرا این دو قطعه در مدار داخلی و به طور کلی اتصالات و کارکرد با هم تفاوت دارند . اگر در بافر 'G را به ۰ متصل کنیم و فعال باشد ، ورودی به خروجی متصل میشود و اگر ۰ نباشد هم در خروجی z می بینیم. در صورتی که باید همان مقدار قبل باقی بماند. در لچ این امکان وجود دارد و اگر غیرفعال شود مقدار را حفظ میکند. در عوض اگر بخواهیم به عنوان ورودی از لچ استفاده کنیم هم نمیتوانیم، زیرا در این صورت اگر چندین لچ را به ورودی متصل کنیم و مقدار آنها z نشود در واقع تداخل سیگنال خواهیم داشت که مشکلات الکترونیکی به دنبال دارد. اما اگر مقدار buffer را کنترل کنیم و خودمان بعد از دریافت صفر نگهش داریم میتوانیم از آن به عنوان خروجی استفاده نماییم.

سوال ۲

Latch:

tSHSL = the time STB is high to when it becomes low

tIVSL = input to STB setup time (from putting data until STB falling edge)

tSLIX = input to STB hold time (from falling edge until data is valid)

Buffer:

tPHL = propagation delay from output high to low

tPLH = propagation delay from output low to high

tPZH = propagation delay from output high z to high

tPZL = propagation delay from output high z to low

Decoder:

tPHL = propagation delay from giving address to the port until change in output from high to low

tPLH = propagation delay from giving address to the port until change in output from low to high

سوال ۳

روش سرکشی CPU را دائما در حالت busy نگه می‌دارد چون باید مرتبا وسیله جانبی را چک کند تا وقتی آن وسیله از حالت busy خارج شود و بتواند داده را دریافت کند و زمانی که چندین دستگاه باشد ، این روش عملا نارضایتی به همراه دارد.

اما در روش وقفه CPU دیگر مجبور به بررسی مداوم نخواهد بود و فقط بعد از هر دستور آمدن وقفه‌ها را چک خواهد کرد. اگر پرچم وقفه مربوطه set شده بود، pc به روتین وقفه jump خواهد کرد و پس از اجرای روتین دیگر منتظر دستگاه نمی ماند و در صورتی که دریافت دستگاه تمام شد و باز هم درخواستی داشت دوباره وقفه میدهد . و نهایتا بازدهی بیشتری خواهیم داشت چون پردازنده میتواند به پردازش سایر قسمت ها بپردازد. با اجرای روتین وقفه پرچم مذکور هم صفر می‌شود و آماده‌ی دریافت وقفه‌های بعدی هستیم.

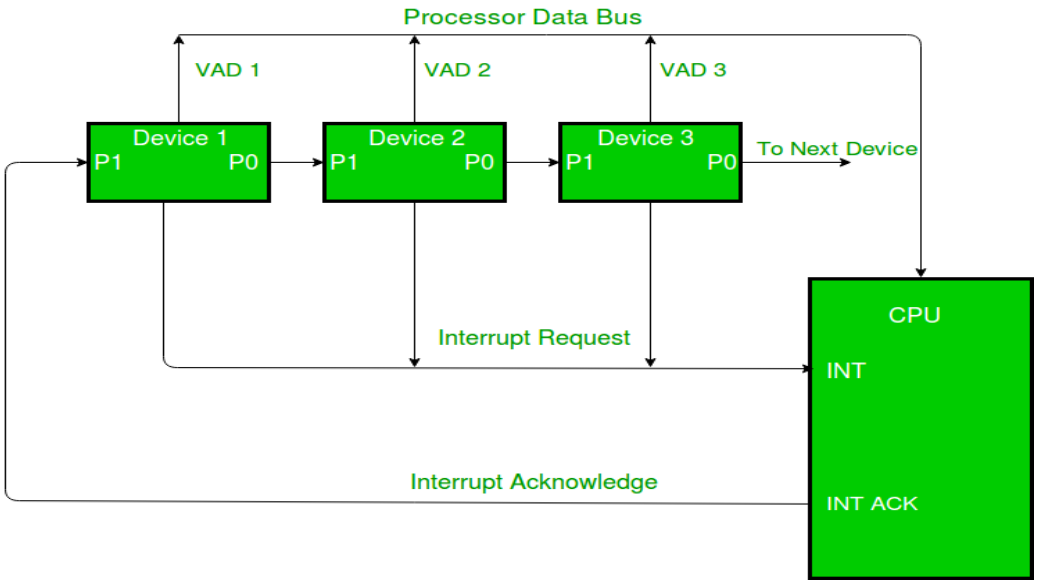
در سیستم های multitasking استفاده از روش وقفه بسیار مناسب تر است زیرا میتوان پس از هر روتین به وقفه جدیدی سرویس داد و دیگر لازم نیست منتظر اتمام کار دستگاه ارتباطی برای آزادکردن پردازنده بمانیم و از طرفی اگر چندین دستگاه داشته باشیم ، پرچمهای زیادی را هم باید در هر دوره چک کنیم که سربار را افزایش میدهد .

سوال ۴

روش های اولویت‌دهی عبارت اند از Priority Encoder و استفاده از Daisy chain.

بدین ترتیب اگر CPU متوجه چند وقفه همزمان شد، می‌تواند به ترتیب اولویت به آن‌ها پاسخ دهد.

سخت افزار daisy chain:



سوال ۵

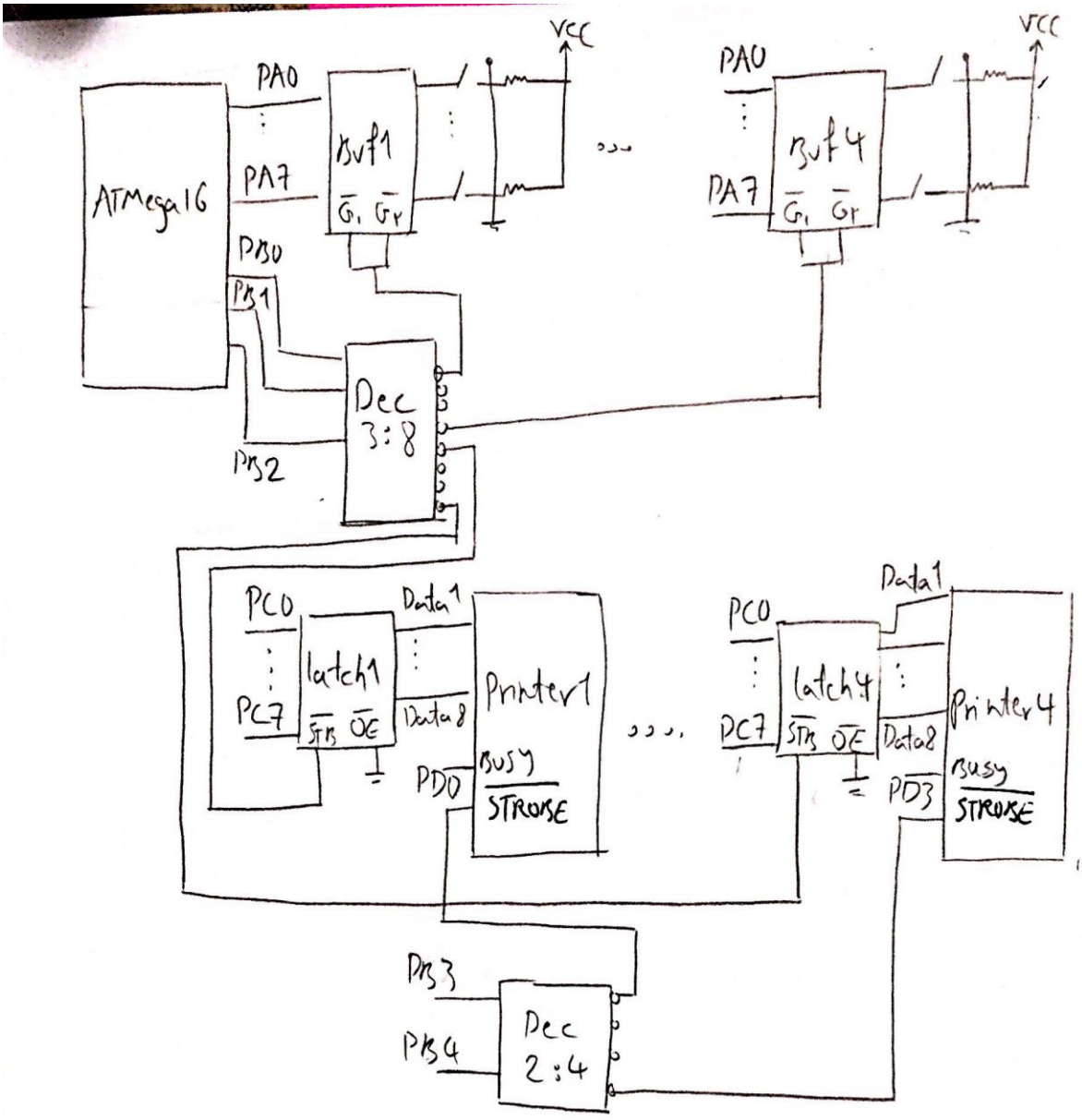
بله قابل انجام است. با ترتیبی که خودمان در کد برنامه و به صورت نرم‌افزاری پیاده می‌کنیم می‌توانیم برایشان ترتیب اولویت در نظر بگیریم. همه ی وسایل جانبی در روش سرکشی باید داخل یک loop دائمی بررسی شوند و ترتیب چک کردن آن ها بر اساس اولیتشان خواهد بود. اما این روش نسبت به روش سخت‌افزاری سرعت کمی خواهد داشت و با توجه به روند تکراری آن، هر وقفه نهایتا اگر به روتینش رسیده باشیم بررسی خواهد شد و اولویت بندی خاصی نمیتوان اعمال کرد چون هنگام کار روی یک تقاضا نمی‌شود به بقیه تقاضاها رسیدگی کرد.

سوال ۶

در هنگام رخ دادن وقفه ، میکرو محتوای ثبات ها و پرچم های مربوط به وقفه را ذخیره نمیکند و تنها ثبات های مربوط به پرش به روتین وقفه و محل بازگشت از روتین در استک ذخیره میشود. به عبارتی تنها PC را در استک ذخیره میکند تا در پایان بتواند دوباره به آن بازگردد.
اگر قرار باشد اطلاعاتی درون روتین وقفه تغییر کند (مثلا پرچم های ثبات status و یا سایر ثبات ها) باید قبل از ورود به روتین آن‌ها را ذخیره کنیم. با push کردنشان درون stack و در انتهای روتین pop نمودن، می‌توان آن‌ها را ذخیره و سپس بازیابی کرد.

سوال ۷

الف)

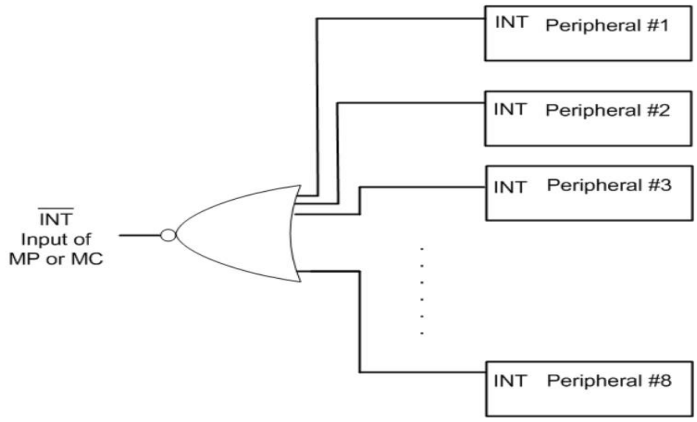


ب)

```
frequency = 16Mhz => Period = 62.5

PIN delay = 1.5 clocks

printer strobe active time = 500

start:

ldi r16, High(RAMEND)
out SPH, r16
ldi r16, Low(RAMEND)
out SPL, r16

call read_buff

call write_printer

read_buff:

ldi r17, 0x00
out DDRA, r16
ldi r16, 0xff
out DDRB, r16
out PORTB, r17  ;put address on port
nop
nop
nop

in r17, PINA  ; read the value
ret


write_printer:

ldi r17, 0xff
out DDRC, r17
out DDRB, r17
ldi r17, 0x00
out DDRD, r16
```

```
out PORTC, r18  ;this is the data register
ldi r16, 0x04
out PORTB, r16


loop:

sbic PIND, 0
rjmp loop
sbi PORTB, 3

sbi PORTB, 4

nop    ;write pulse width
nop
nop
nop
nop
nop
nop
nop

cbi PORTB, 3
cbi PORTB, 4

ret
```

ج) اگر INT0 را حساس به سطح پایین در نظر بگیریم از یک قطعه nor استفاده می کنیم که ورودی‌اش تمامی کلیدها و پایه‌های busy مربوط به پرینترها باشد و خروجی را به پایه اینتراپت میکرو وصل کنیم. چک کردن فشرده شدن کلیدها از طریق بافرها انجام شده است.



د)

```
.def counter = r24
.org 0x002
rjmp routine

.org 0x030
start:

ldi r16, high(RAMEND)
out SPH, r16
ldi r16, low(RAMEND)
out SPL, r16


ldi r16, 0xff
out DDRB, r16

ldi r16, 0x00
out DDRA, r16
out PORTB, r16

// MCUCR and GICR settings are written here

loop1: rjmp loop1

routine:
in r17, PINB

ldi counter, 4
ldi r20, 8

loop2:
        out PORTB, r17

        nop
        nop
        nop

        in r16, PINA
        cpi r16, 0xff
        breq next_buff
```

```
check_keys:
        dec r20
        lsl r16
        brcc read_key
        rjmp check_keys

next_buff:
        inc r17
        dec counter
        cpi counter, 0
        brne loop2

read_key:
        mov r0, r20

check_printers:
        sbic PORTD, 0
        rjmp printer1_busy_low
        sbic PORTD, 1
        rjmp printer2_busy_low
        sbic PORTD, 2
        rjmp printer3_busy_low
        sbic PORTD, 3
        rjmp printer4_busy_low
        reti
```