تمرین ۱۰ ریزپردازنده - فاطمه غلامزاده - ۹۵۱۳۱۰۶۰

سوال ۱

Latch مقداری که در ورودی می‌گیرد را تثبیت می‌کند اما باز مقداری که در خروجی می‌گیرد را تقویت می‌کند و عبور می‌دهد. از لچ به شرطی می‌توان به عنوان درایور استفاده کرد که ورودی آن به اندازه کافی قوی باشد و نیاز به تقویت نباشد.

از بافر هم می‌توان به عنوان خروجی استفاده کرد به شرطی که کنترل بیشتری نسبت به Latch روی آن داشته باشیم.

سوال ۲

لچ:

tSHSL ← استروب high تا استروب low

tIVSL ← گذاشتن داده تا لبه بالارونده سیگنال استروب

tSLIX ← پایین رفتن پالس STB تا انتهای زمان معتبر بودن داده

بافر:

tPHL ← تاخیر انتشار از زمان گذاشتن آدرس تا تغییر وضعیت خروجی از high به low با فرض اینکه پایه های Enable از قبل فعال شده باشند

tPHL ← تاخیر انتشار از زمان گذاشتن آدرس تا تغییر وضعیت خروجی از low به high با فرض اینکه Enable ها از قبل فعال شده باشند

tPZL ← تاخیر انتشار در حال رفتن خروجی از حالت شناور به low

tPZH ← ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ high

دیکودر و انکودر:

tPHL ← مشابه موارد قبل

tPLH ← ~ ~ ~

سوال ۳ در روش polling پردازنده به طور مداوم باید به دستگاه های I/O سرکشی کند تا در صورت وجود درخواست یا آمادگی آن دستگاه کاری برای آن انجام دهد. این امر زمان زیادی از CPU را تلف می کند.

در روش مبتنی بر وقفه CPU دستورات عادی خودش را اجرا می کند و بعد از اجرای هر دستور چک می کند که آیا وقفه ای آمده یا نه. اگر آمده باشد دیگ روتین وقفه به آن رسیدگی می کند. این روش کارایی CPU را بالا می برد و زمان تلف شده آن را به شدت کاهش می دهد

سوال ۴

① استفاده از priority Encoder

② اینکه همه ی وقفه ها را دریافت کنیم و خودمان به ترتیبی که می خواهیم به آن ها پاسخ بدهیم

سوال ۵ بله امکان پذیر است. سرکشی را به ترتیب انجام دهیم و اگر یک دستگاه تقاضای عملیات داشت و انجام دادیم، ترتیب را از سر بگیریم و مشابه روش Round Robin باشد

سوال ۶

۱- مقدار PC را نگه داریم (program Counter)

۲- اگر مقدار رجیستر ها (به خصوص SREG) مهم است آن را هم در استک ذخیره کنیم

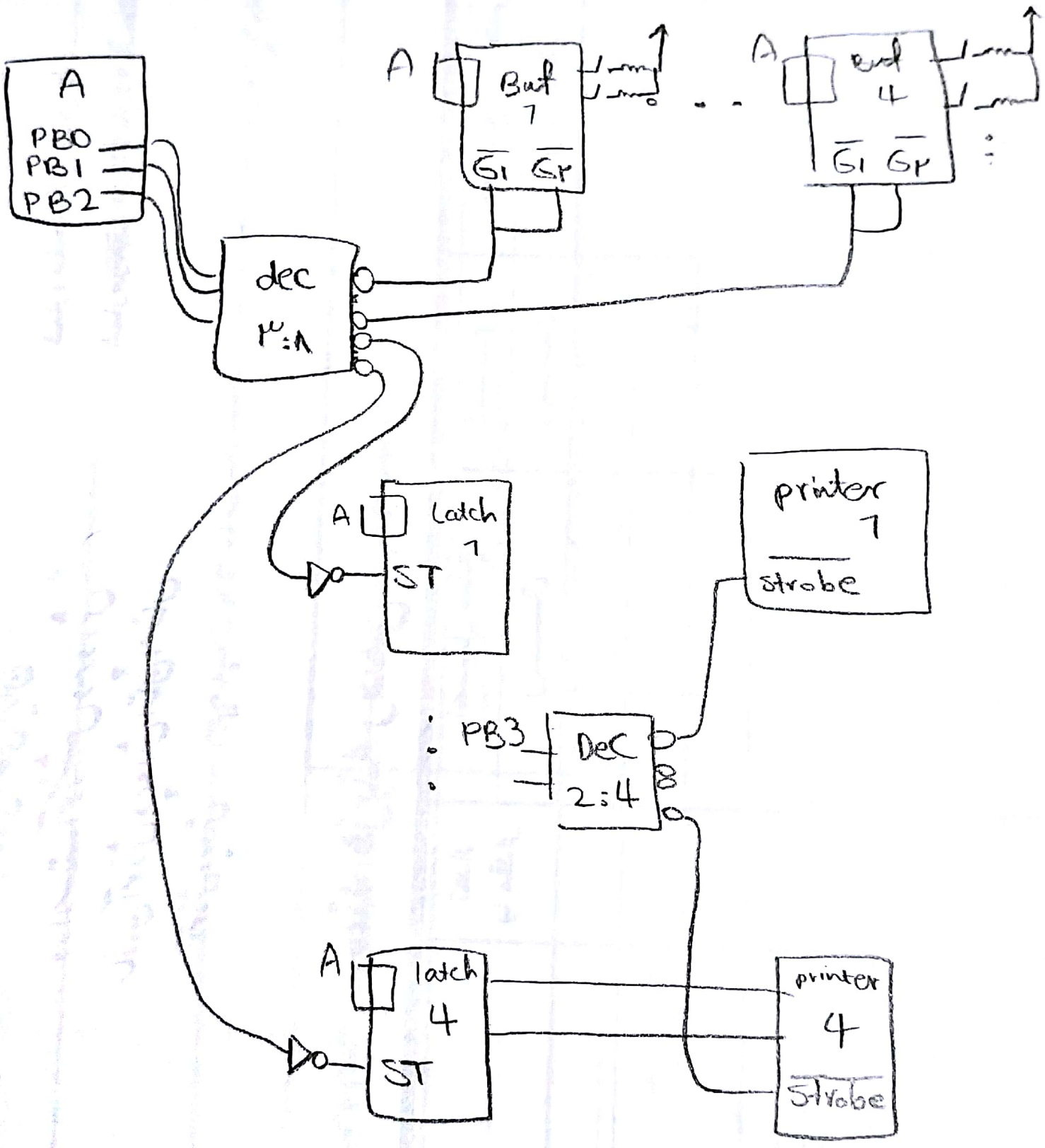
ذخیره و بازیابی در stack صورت می گیرد.

هنگام بازیابی هم باید PC و رجیسترها را از استک pop کنیم.

سوال ۷

تأخیر Strobe برای پرینترها بالاست ← دیکودر حرکاتی استفاده شده

دیکودر ۲:۴ ← با رسم یک Enable طراحی شود ← PB5 وصل شده



```
.org 0x00
JMP start


Start:
LDI R24, 0xFF
OUT DDRB, R24
LDI R24, 0x04
CALL BufferRead
CALL Printer
RJMP Start

BufferRead:
LDI R24, 0x00
OUT DDRA, R24
OUT PORTB, R24
NOP
NOP
NOP
LDI R20, 0x8
LOOP1:
IN R16, PINA
LDI R17, 0xFF
```

```
LOOP2:  DEC  R20
LSL R16
BRCC LOOP3
RJMP LOOP2
LOOP3:
MOV R0, R20;
RET


Printer:
LDI R24, 0xFF
OUT DDRA, R24
LDI R24, 0x04
OUT PORTB, R24 ;
LDI R24, 0xFF
OUT DDRD, R24
LDI R20, 0x00
LOOPp:
```

```
OUT DDRA, R24
LDI R24, 0x04
OUT PORTB, R24 ;
LDI R24, 0xFF
OUT DDRD, R24
LDI R20, 0x00
LOOPp:
SBIC  PIND, 06
RJMP LOOPp
MOV R21, R0
OUT PORTA, R21 ;
LDI R20, 0x01
OUT PORTB , R20
LDI R20, 0x00 ;
OUT PORTB , R20 ;
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
RET
```

```
.org 0x002
RJMP int0_routin


Start:
LDI R16, (0<<ISC00)|(0<<ISC01)
OUT MCUCR, R16
LDI R16, (1<<INT0)
OUT GICR, R16
SEI


main:
RJMP main

int0_routin:
RJMP BufferRead
RJMP Printer
```

```
BufferRead:
LDI R24, 0xFF
OUT DDRB, R24
LDI R24, 0x00
OUT DDRA, R24
LDI R24, 0x04
OUT PORTB, R24
NOP
NOP
NOP
LDI R20, 0x8
LOOP1:
IN R16, PINA
LDI R17, 0xFF
CP R16, R17
BREQ LOOP1
LOOP2:  DEC  R20
LSL R16
BRCC LOOP3
RJMP LOOP2
LOOP3:
MOV R0, R20
RET
```

```
Printer:
LDI R24, 0xFF
OUT DDRA, R24
LDI R24, 0x04
OUT PORTB, R24
LDI R24, 0xFF
OUT DDRD, R24
LDI R20, 0x00
LOOP1p:
SBIC  PIND, 06
RJMP main
MOV R21, R0
OUT PORTA, R21
LDI R20, 0x01
OUT PORTB , R20
LDI R20, 0x00
OUT PORTB , R20
NOP
NOP
NOP
NOP
NOP
```

```
OUT DDRA, R24
LDI R24, 0x04
OUT PORTB, R24
LDI R24, 0xFF
OUT DDRD, R24
LDI R20, 0x00
LOOP1p:
SBIC  PIND, 06
RJMP main
MOV R21, R0
OUT PORTA, R21
LDI R20, 0x01
OUT PORTB , R20
LDI R20, 0x00
OUT PORTB , R20
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
RET
```