Mikroprocesorska elektronika

Predavanje VI

Sadržaj predavanja

- Organizacija ulazno/izlaznog podsistema
- Sistem prekida

Organizacija ulazno/izlaznog podsistema

Organizacija ulazno/izlaznog podsistema I

- Ulazno/izlazni podsistem sastoji se iz svih periferijskih uređaja povezanih na sistemske magistrale, izuzimajući centralnih procesor i memorije.

- Oznaka U/I (I/O) dodeljuje se svim uređajima koji služe kao ulazi, izlazi ili oboje unutar nekog mikroračunarksog sistema, ali takođe se dodeljuje i svim registrima posebne namene ili uređajima koji služe za upravljanje radom sistema bez potrebe za komunikacijom sa njegovim okruženjem

- CPU predstavlja referencu kada se neki od uređaja označava kao ulazni ili izlazni:
  - Ulazna transakcija prenosi informacije ka procesoru od periferije, čineći tu periferiju *ulaznom periferijskom jedinicom*
  - Izlazna transakcija prenosi podatke od procesora ka periferiji, čineći je *izlaznom periferijskom jedinicom*

Organizacija ulazno/izlaznog podsistema II

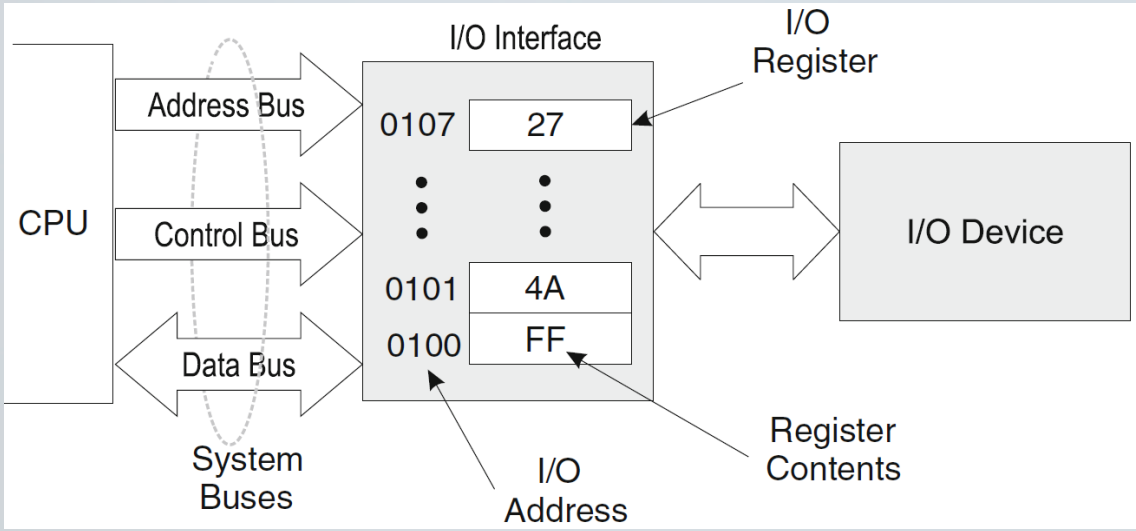
- Primeri ulaznih periferijski jedinica uključuju:
  - Prekidače, tastere, tastature
  - Čitače bar kodova
  - Razne vrste senzora (temperature, pritiska, blizine, pozicije, brzine, itd...)
  - Analogno/digitalne konvertore
- Primeri izlaznih periferijskih jedinica uključuju:
  - LED
  - Displeje (LED, LCD)
  - Zvučnike
  - Motore (DC, AC)
  - Digitalno/analogne konvertore
- Primeri ulazno/izlaznih periferijskih jedinica:
  - Razne vrste komunikacionih uređaja (serijski port, paralelni port, uređaji masovne memorije kao što su flash memorijske kartice, hard diskovi, itd...)

Organizacija ulazno/izlaznog podsistema III

- Periferijski uređaji koji se najčešće sreću u embeded sistemima su:
  - **Tajmeri** – ove periferijske jedinice mogu se programirati za merenje proteklog vremena: za merenje vremena proteklog između dva događaja, za generisanje događaja u tačno određenim trenucima, za generisanje signala odgovarajuće frekvencije (kao što su PWM signali), itd.
  - **Watchdog tajmer (WDT)** – ovaj tajmer ima posebnu namenu, obezbeđivanje pouzdanog rada embeded sistema. Ukoliko WDT ne dobije signal koji se generiše programski od strane procesora svakih X vremenskih jedinica, WDT generiše reset signal ili signal prekida koji se zatim koristi za izvođenje korektivnih akcija u sistemu.
  - **Komunikacioni interfejsi** – koriste se za razmenu informacija između embeded sistema i nekog drugog uređaja ili sistema. Prilikom prenosa podataka koriste različite protokole kao što su **SPI (Serial Peripheral Interface**), **USB (Universal Serial Bus),** itd.
  - **Analogno/digitalni konvertori (ADC)** – koristi se za konverziju analognih signala u odgovarajuće digitalne. Vrlo često se koristi jer je većina signala u prirodi analogna.
  - **Digitalno/analogni konvertori (DAC)** – koristi se za konverziju digitalnih signala u odgovarajuće analogne signale.

Organizacija ulazno/izlaznog podsistema IV

- Organizacija I/O podsistema podseća na organizaciju memorije
- Svaki periferijski uređaj zahteva I/O interfejs kako bi mogao da komunicira sa procesorom
- Ovaj interfejs služi kao “most” između I/O uređaja i sistemskih magistrala
- Svaki I/O interfejs sadrži jedan ili više registara koji omogućavaju razmenu podataka, upravljačkih i statusnih informacija između procesora i I/O uređaja
- Kao posledica ovakve organizacije, što se tiče procesora, svaki I/O uređaj “izgleda” vrlo slično nekom memorijskom modulu, gde se registri I/O interfejsa “ponašaju” kao memorijske lokacije povezane na magistralu podataka, adresirane pomoću adresne magistrale



I/O mapirane i memorijski mapirane periferijske jedinice

- Neki od komercijalnih procesora, kao na primer Intelovi x86 procesori, imaju odvojene adresne prostore za memoriji i I/O uređaje
- U ovom slučaju u skupu instrukcija moraju postojati **posebne instrukcije** za pristup periferijskim jedinicama mapiranim u I/O adresni prostor
- Ovaj pristup, poznat pod nazivom ***Standard I/O*** ili ***I/O-mapped I/O***, zahteva posebne instrukcije i kontrolne signale koji signaliziraju da li je trenutna adresa "namenjena" memoriji ili I/O podsistemu
- Alternativa gornjem pristupu je da se I/O uređaji uključe unutar postojećeg adresnog prostora za memoriju, obzirom da se I/O registri mogu posmatrati kao neke dodatne memorijske lokacije
- Ovaj pristup poznat je pod nazivom *Memory-mapped I/O* i danas predstavlja dominatnu šemu povezivanja I/O uređaja u savremenim embeded sistemima

Anatomija I/O interfejsa I

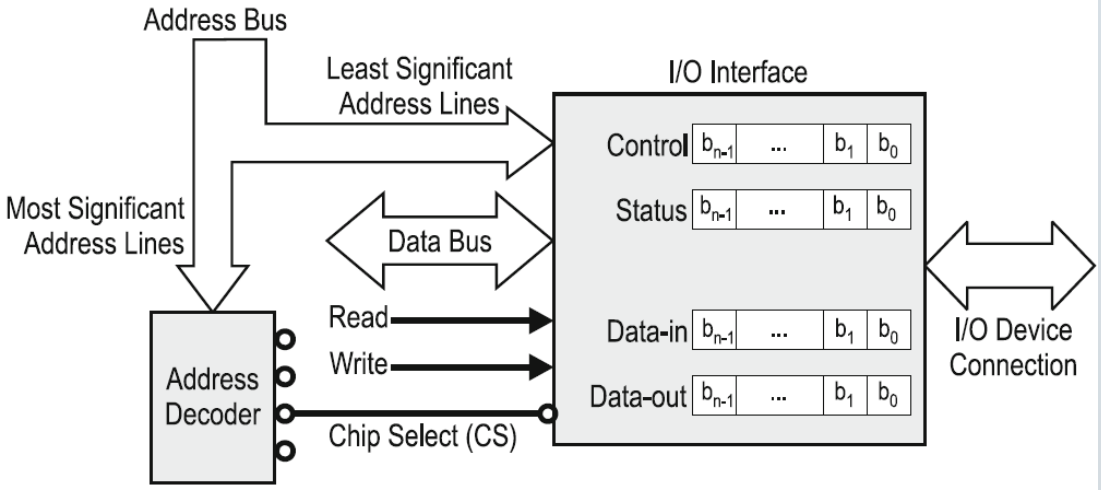
- I/O interfejs sastoji se iz:
  - Linija koje se povezuju na adresnu, kontrolnu i magistralu podataka procesora
  - Linija koje se povezuju sa I/O uređajem
  - Skupa unutrašnjih registara

- Linije magistrale podataka (obično je reč o 8-bitnim ili 16-bitnim magistralama) koriste se za prenos podataka od i ka unutrašnjim registrima

- Nekoliko adresnih linija najmanje značajnosti koristi se za selekciju internih registara I/O interfejsa

- Preostale adresne linije obično su povezane na eksterni adresni dekoder koji omogućava rad sa više različitih I/O uređaja

- Obično postoje barem dva kontrolna signala (read i write) koji služe za selekciju i sinhronizaciju operacija čitanja i upisa



Anatomija I/O interfejsa II

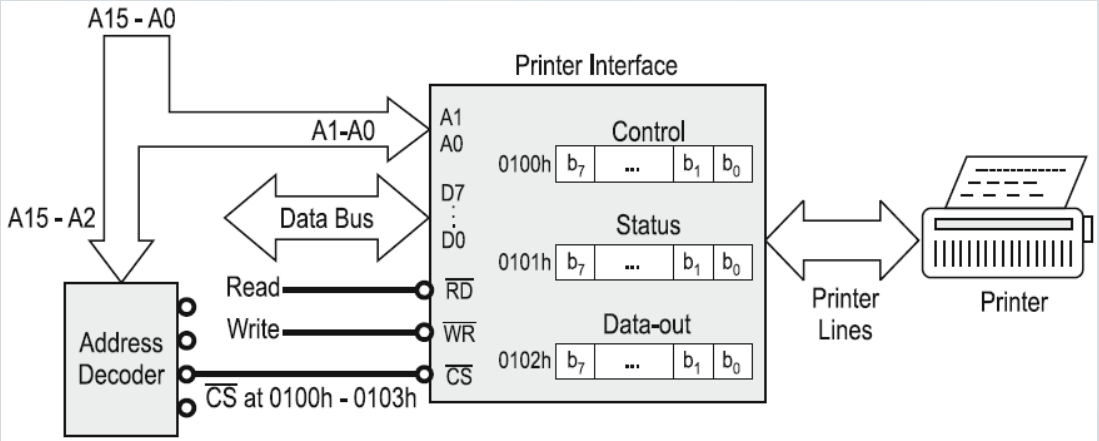
- Unutrašnji registri mogu biti *read-only*, *write-only* ili *read-write* tipa u zavisnosti od funkcije registra i vrste interfejsa

- U većini I/O interfejsa mogu se pronaći tri tipa unutrašnjih registara:
  - **Kontrolni registri** – omogućavaju konfigurisanje I/O uređaja ali i samog interfejsa. Može postojati jedan ili više ovakvih registara. Često imaju naziv *Mode Register* ili *Configuration Register*.
  - **Statusni registri** – omogućavaju uvid u trenutni status I/O uređaja i interfejsa. Posebna polja unutar ovih registara prenose informaciju o specifičnom statusu, kao što je na primer spremnost uređaja na prihvat novih komandi ili signalizacija greške prilikom izvršavanja poslednje komande
  - **Registri podataka** – omogućavaju prenos podataka između procesora i I/O uređaja. Unidirekcioni I/O uređaji mogu imati samo jedan registar podataka (*Data-in* za ulazne periferije, *Data-out* za izlazne periferije). Bidirekcioni I/O uređaji po pravilu uključuju oba tipa registara podataka.

Primer I/O interfejsa za štampač I

- Razmotrimo strukturu hipotetičkog I/O interfejsa za štampač

- Interfejs sadrži tri 8-bitna registra:
  - **Control** – omogućava procesoru da upravlja radom štampača
  - **Status** – služi za saopštavanja informacija o tekućem stanju štampača
  - **Data-out** – write-only registar u koji procesor upisuje kod simbola koji je potrebno odštampati



- Obzirom da postoji tri unutrašnja registra, potrebno je koristiti barem dve adresne linije, A1 i A0, za njihovo adresiranje

- Moguće mapiranje unutrašnjih registara u adresnom prostoru prikazano je u tabeli desno

A1	A0	Registar
0	0	Control
0	1	Status
1	0	Data-out
1	1	Ne koristi se

Primer I/O interfejsa za štampač II

- Adresna magistrala procesora biće povezana na I/O interfejs štampača na sledeći način:
  - Adresni biti A1 i A0 biće povezani direktno na adresnu magistralu I/O interfejsa štampača
  - Preostali adresni biti (u našem primeru adresna magistrala je 16-bitna) A15-A2 povezani su na ulaze adresnog dekodera koji je projektovan na takav način da generiše nizak CS signal kada je bazna adresa I/O interfejsa štampača jednaka 0x0100.

- Imajući ovo na umu adrese unutrašnjih registara I/O interfejsa štamapača su:
  - Control registar = 0x0100
  - Status registar = 0x0101
  - Data-out registar = 0x0102

- Adresa 0x103 će biti neiskorišćena, obzirom da ova adresa nije pridružena ni jednom unutrašnjem registru I/O interfejsa

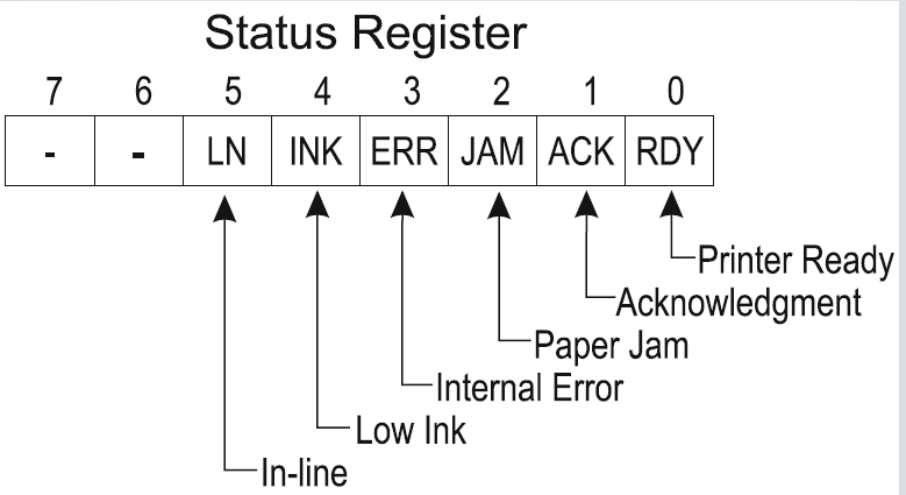
Primer I/O interfejsa za štampač III

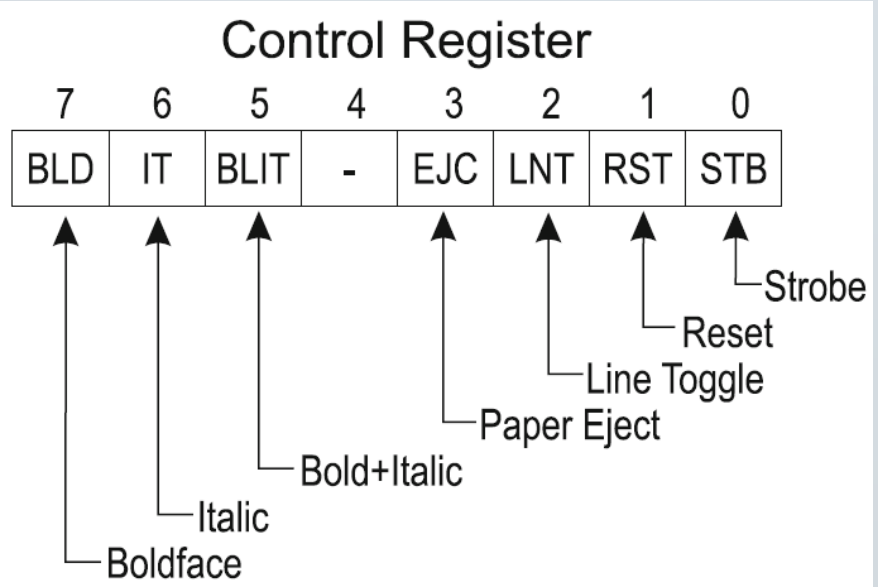
- Uz pretpostavku da kontrolni i statusni biti unutar kontrolnog i statusnog registra imaju značenja kao na slikama desno, postupak korišćenja štampača bio bi sledeći:

  - Koristeći adresu 0x0100 procesor upisuje odgovarajuću konfiguracionu reč u kontrolni registar kojom definiše kako će se naredni karakter odštampati (normalno, boldovano, itd.) ili se izbacuje papir iz štampača, itd.

  - Kada želi da odštampa sledeći karakter, procesor upisuje njegov kod u Data-out registar, na adresi 0x0102

  - Čitajući statusni registar, mapiran na adresu 0x0101, procesor može da dobije informacije o trenutnom statusu štampača, na primer da li je štampač spreman da prihvati naredni karakter, da li se zaglavio papir, da li je nivo mastila ispod minimuma, itd.





Paralelni i serijski I/O interfejsi I

- Ukoliko se embeded sistemi projektuju tako da se većina periferijskih jedinica nalazi na istom integrisanom kolu na kojem se nalazi i procesor, periferijske jedinice su sa procesorom povezane preko magistrale podataka sa paralelnim interfejsom

- Pod paralelnim interfejsom podrazumevamo interfejs preko kojega se čitava digitalna reč prenosi istovremeno, zahtevajući po **jedan signal za svaki bit**

- Međutim, povezivanje procesora i periferijskih jedinica koje nisu na istom integrisanom kolu može se izvesti na dva načina:
  - **Korišćenjem paralelnog interfejsa** (paralelni I/O portovi)
  - **Korišćenjem serijskog interfejsa** (serijski I/O portovi)

- Za razliku od paralelnog interfejsa, podaci se preko serijskog interfejsa prenose bit po bit, zahtevajući samo jedan signal za prenos podataka

Paralelni i serijski I/O interfejsi II

- Serijski interfejsi su tip interfejsa koji se prvi koristio prilikom prenosa podataka između komponenti sistema
- Najpoznatiji serijski interfejs je RS232 interfejs, koji se i dan danas koristi za prenos podatka pogotovo ako je udaljenost između uređaja velika
- Glavna motivacija za korišćenje serijskih interfejsa bila je njihova niska cena realizacije, jer zahtevaju samo jednu signalnu liniju (plus liniju za masu)
- Međutim, u ranim mikroračunarskim sistemima brzina serijskih interfejsa bila je vrlo mala, te su se za brzi prenos podataka na malim rastojanjima preferirali paralelni interfejsi
- U ranim sistemima paraleni interfejsi masovno su se koristili za povezivanje štampača, uređaja masovne memorije, itd...
- Neki od najpoznatijih paralelnih protokola su PCI, SCSI, ISA, EISA, GPIB

Paralelni i serijski I/O interfejsi III

- Međutim, paralelni interfejsi imaju ozbiljne probleme ukoliko se brzina signalizacije povećava
- Na primer, usled brzo promenljivih signala, dolazi do elektromagnetske interferencije između susednih linija paralelnog interfejsa
- Takođe, osigurati istovremeni dolazak različitih signala koji čine paralelni interfejs na odredište u slučaju brzih konekcija je vrlo komplikovano
- Upravo iz ovih razloga, u modernim embeded sistemima **dominiraju ultra-brzi serijski interfejsi**, sa brzinama prenosa koje mogu iznositi i nekoliko desetina Gbit/s
- Danas većina štampača, skenera, hard diskova, displeja i mnogo drugih vrsta periferijskih uređaja koriste serijske interfejse za komunikaciju sa procesorom
- Najpoznatiji serijski interfejsi koji se koriste u savremenim embeded sistemima su UART, USB, FireWire, SATA, Ethernet, SPI, I2C, PCIe, itd...

Sinhronizacija procesora i I/O uređaja I

- Sinhronizacija između procesora i periferijske jedinice neophodna je u slučaju kada je brzina rada periferijske jedinice manja od brzine rada procesora
- U ovom slučaju, procesor mora da “čeka” dok periferijska jedinica ne bude spremna za sledeću transakciju
- Na primer, štampanje karaktera na papiru je sporiji proces od njihovog prenosa u vidu električnih signala
- Zbog toga, štampač poseduje bafer, koji se realizuje kao odgovarajuća RAM memorija, u koje smešta karaktere koje je neophodno odštampati
- Štampač ne može primati nove karaktere kada je njegov bafer pun
- Drugi primer su A/D i D/A konvertori, koji pre nego što su spremni da pošalju odnosno prime novi podatak moraju završiti konverziju tekućeg podatka

Sinhronizacija procesora i I/O uređaja II

- Da bi “javile” procesoru svoju spremnost ili nespremnost za sledeću transakciju periferijske jedinice generišu odgovarajući statusni *ready* signal
- Ready signal se postavlja ili briše u zavisnosti od toga da li je periferijska jedinica spremna za sledeću transakciju sa procesorom ili nije
- Sinhronizacija između procesora i I/O uređaja ostvaruje se preko *ready* signala na dva načina:
  - Korišćenjem **metode prozivke** (*polling*)
  - Korišćenjem **metode prekida** (*interrupt*)
- Kod metode prozivke, procesor periodično “proziva” I/O uređaje kako bi utvrdio koji od njih je spreman ili zahteva novu transakciju
- Kod metode prekida, procesor komunicira sa I/O uređajem samo ukoliko je on “zatražio” komunikaciju

Vremenski dijagrami I

- Podaci između procesora i I/O uređaja prenose se preko skupa odgovarajućih magistrala: magistrale podataka, adresne i kontrole magistrale

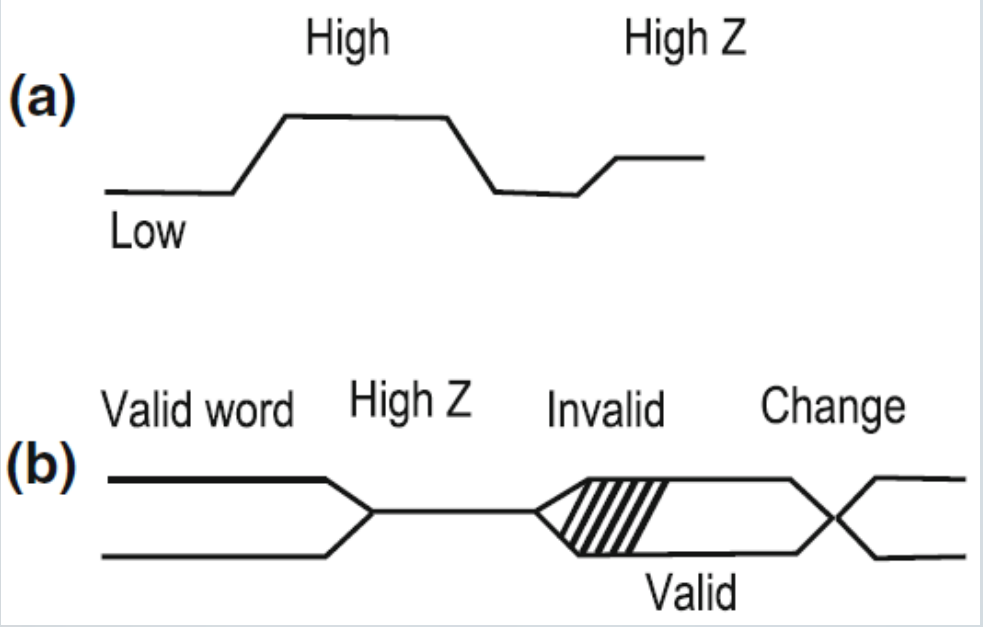
- Međutim, da bi se podaci pouzdano prenosili između dva uređaja potrebno je definisati **tačan redosled koraka** i njihovo trajanje na svakoj od magistrala koja čini **sistem za prenos podataka**

- Informacija o redosledu koraka i njihovom trajanju definiše se preko skupa pravila koja čine ***komunikacioni protokol***

- Protokoli se najčešće opisuju pomoću vremenskih dijagrama

Vremenski dijagrami II

- U vremenskim dijagramima vreme teče sa leva na desno
- Za 1-bitne signale obično se definišu dva moguća stanja:
  - stanje logičke jedinice (*High*) i
  - stanje logičke nule (*Low*)
- Opciono, može postojati i treće stanje, stanje visoke impedanse (*High Z*)
- Do promene vrednosti signala dolazi tokom konačnog intervala vremena, što je na dijagramu prikazano pomoću kosih linija
- U slučaju magistrala, koje se sastoje iz većeg broja individualnih bitova, validne reči se reprezentuju pomoću **paralelnih linija** unutar kojih opciono stoji upisana validna vrednost

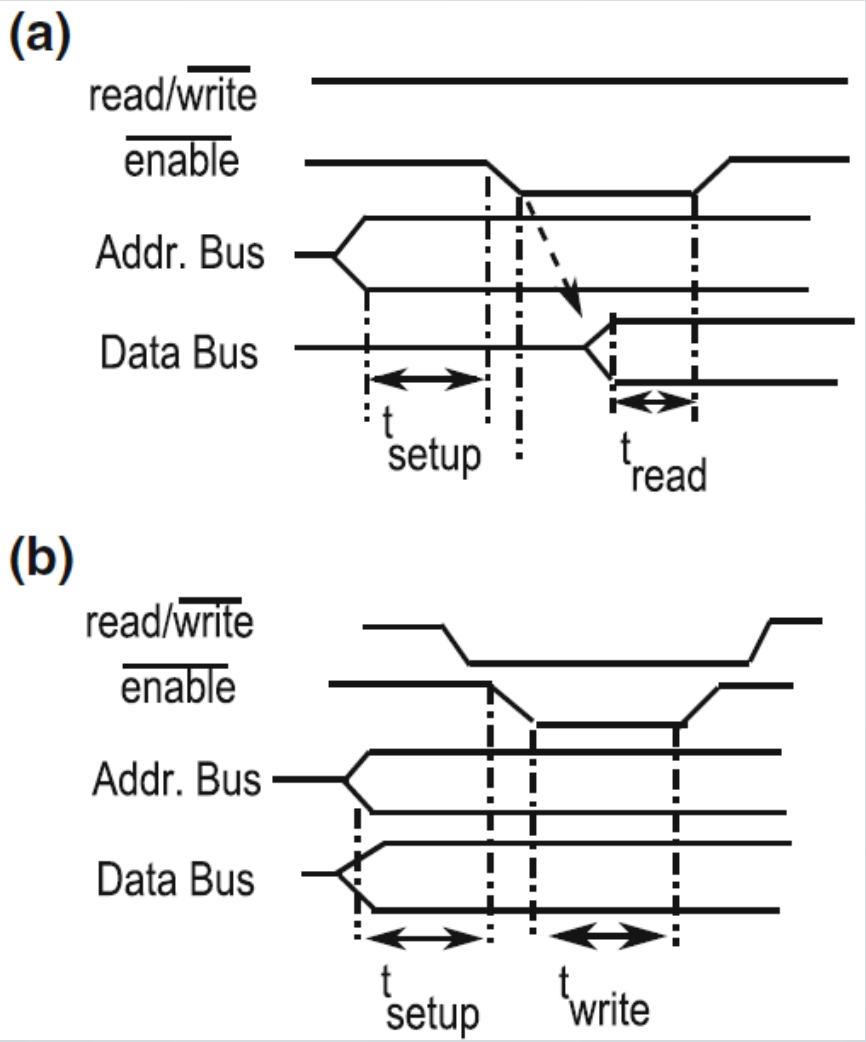


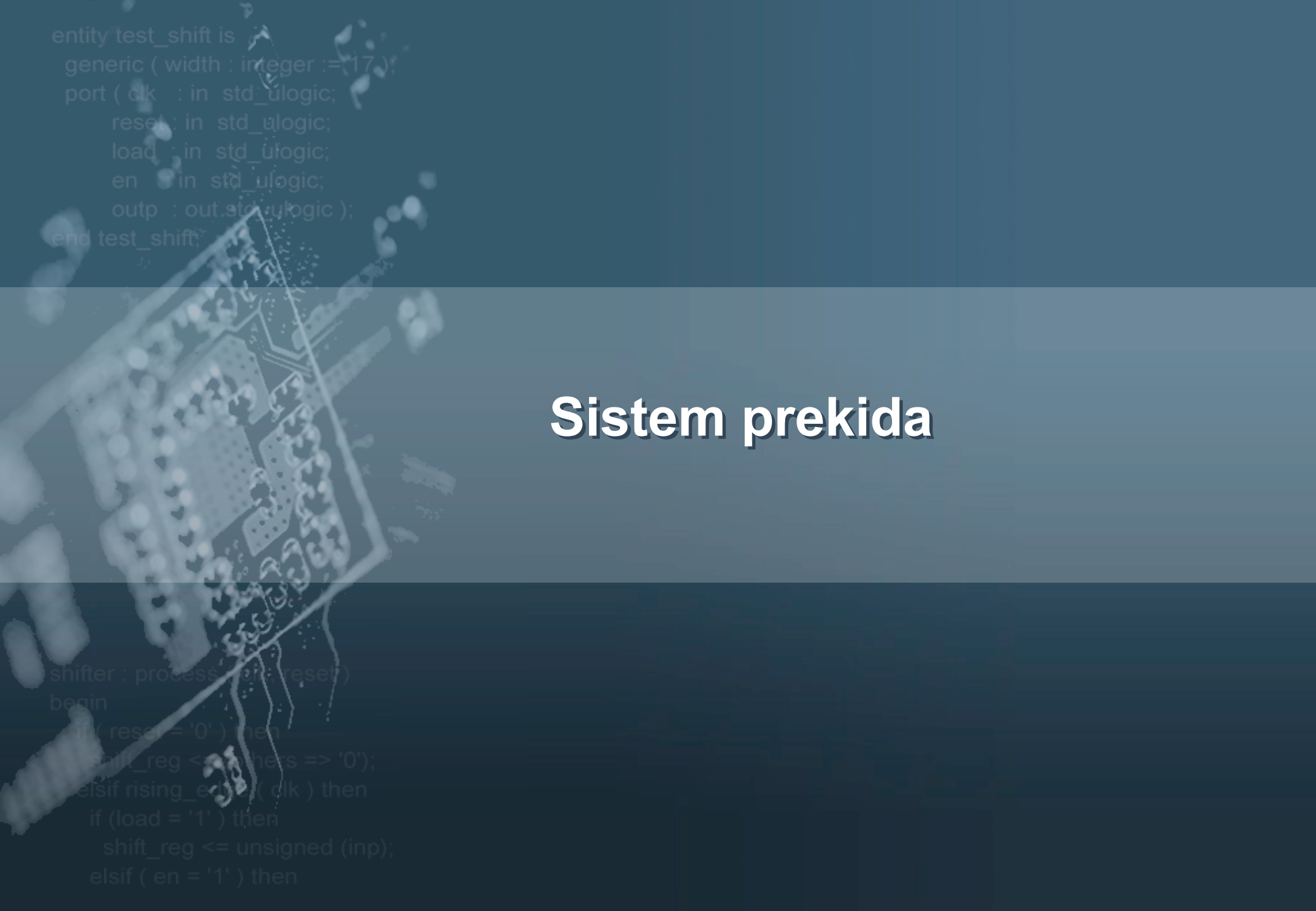
Primer vremenskog dijagrama

- Slika desno prikazuje jedan pojednostavljeni **vremenski dijagram** mogućeg protokola za čitanje (a) i upis (b) podataka u memoriju

- U ovom slučaju postoje dva kontrolna signala:
  - **enable** – signal aktivan na logičkoj nuli koji aktivira memorijski modul
  - **read/write** – signal koji određuje da li je trenutna transakcija upis (read/write=0) ili čitanje (read/write=1) podatka iz memorije

- Većina realnih vremenskih dijagrama je daleko složenija od prikazanog





Sistem prekida

Sistem prekida I

- Svaki procesor je u suštini sekvencijalna mašina
- Upravljačka jedinica procesora prihvata, dekoduje i izvršava instrukcije iz programske memorije prema redosledu koji je naveden u programu
- Imajući ovo u vidu jedan procesor u svakom trenutku može da izvršava samo jednu instrukciju
- Ukoliko se javi potreba za servisiranjem periferija od strane procesora, izvršavanje programa se mora prekinuti i programska kontrola se mora preneti na poseban blok instrukcija koji je napisan na takav način da na efikasan način može da opsluži posmatranu periferijsku jedinicu
- Kao primer, razmotrimo šta je potrebno da se opsluži zahtev izdat od strane tastature

Sistem prekida II

- Tastatura se na najnižem nivou može posmatrati kao niz tastera, pri čemu sotfver daje odgovarajuće značenje svakom tasteru
- Tasteri su organizovani u mrežnu strukturu tako da svaki pritisnuti taster rezultuje u generisanju različitog koda na izlazu tasture
- Nakon svakog pritiska, procesor mora preuzeti generisani kod sa tastature
- Pod terminom *opsluživanje tastature* podrazumevamo akciju preuzimanja koda nakon svakog pritiska tastature i njegovo prosleđivanje programu koji će zatim interpretirani primljeni kod
- Kada govorimo o servisiranju periferijskih jedinica, postoje dva pristupa:
  - Servisiranje bazirano na prozivkama (engl. polling)
  - Servisiranje bazirano na prekidima (engl. interrupt handling)

Servisiranje bazirano na prozivkama I

- Kod ovog načina servisiranja procesor neprekidno “ispituje” ili “proziva” status periferijske jedinice kako bi utvrdio da li ona zahteva servisiranje
- Kada utvrdi da periferijska jedinica zahteva servisiranje, procesor preduzima potrebne akcije kako bi servisirao periferiju
- Da bi ilustrovali princip servisiranja baziranog na prozivkama pretpostavimo da je naš zadatak da odgovaramo na telefonske pozive i preuzimamo poruke
- Pretpostavimo dalje da nemamo informaciju o tome kada će pozivi doći, i da na raspolaganju imamo telefon kome ne radi zvono, niti ima vibraciju tako da nemamo nikakav način da ustanovimo da je poziv stigao
- U ovom slučaju jedini način da saznamo da li je neki poziv stigao je da periodično podižemo slušalicu i proverimo da li je neko na liniji

Servisiranje bazirano na prozivkama II

- Ovo bi bio prilično frustrirajući posao, pogotovo ako imamo i neke druge obaveze

- Međutim, ukoliko ne želimo da propustimo ni jedan poziv, moramo obustaviti sve ostale aktivnosti i posvetiti se neprekidnom ponavljanju sledeće sekvence:
  - podizanju slušalice,
  - prinošenju slušalice uvu,
  - proveri da li je neko na liniji

- Obzirom da linija mora biti slobodna, kako bi poziv mogao da dođe, na kraju svake sekvence moramo spustiti slušalicu i ponoviti celu sekvencu od početka

- Kakvo gubljenje vremena! Ali to je prinicip rada sistema baziranog na prozivkama

Servisiranje bazirano na prekidima I

- Kod ovog sistema, svaka periferijska jedinica koristi poseban signal da signalizira procesoru kada ima potrebu da bude servisirana
- Ovaj signal poznat je pod nazivom ***zahtev za prekidom*** (interrupt request, IRQ)
- Procesor može biti zauzet obavljanjem nekih drugih aktivnosti, ili može biti u režimu spavanja, međutim kada stigne neki od zahteva za prekidom, ukoliko su oni dozvoljeni, procesor će obustaviti aktivnost koju je trenutno izvodio kako bi se posvetio obradi prekidnog zahteva, izvršavajući specifičan blok instrukcija
- Ovaj događaj se zove ***prekid***
- Blok instrukcija koje se izvršavaju u cilju opsluživanja periferijske jedinice, kao odgovor na zahtev za prekidom, naziva se ***prekidni potprogram*** (**interrupt service routine, ISR**)

Servisiranje bazirano na prekidima II

- Razmotrimo kako bi radio sistem za odgovaranje na pozive u slučaju da je baziran na sistemu prekida
- Pretpostavimo da naš telefon ima zvono pomoću kojega nam može signalizirati da je stigao novi poziv
- Dok čekamo na poziv, možemo obavljati neke druge aktivnosti, jer znamo da će nam zvono signalizirati dolazni poziv
- Kada se zvono oglasi, zaustavićemo tekuću aktivnost kako bi smo podigli slušalicu i prihvatili poruku
- U ovom slučaju zvono predstavlja naš indikator zahteva za prekidom
- Jasno je da je ovaj sistem daleko efikasniji metod prijema poruka od prethodnog

Vrste prekida

- U opštem slučaju prekidi unutar mikroračunarskog sistema dele se u dve kategorije:

  - **Maskirajući prekidi** – ovi prekidi se mogu zabraniti (maskirati) od strane programera. Način maskiranja varira od procesora do procesora. Na ovaj način moguće je kontrolisati koji izvori prekida mogu prekinuti procesor u datom trenutku.

  - **Nemaskirajuće prekide** – ovi prekidi ne mogu biti zabranjeni od strane programera i svaki put kada se generiše neki prekid ovog tipa procesor će zaustaviti izvršavanje tekuće akcije i preći na opsluživanje prekida. Ovi prekidi su obično rezervisani za signalizaciju kritičnih događaja u sistemu koji se moraju opslužiti odmah.

Anatomija obrade zahteva za prekidom I

- Iako različite vrste procesora mogu imati različite mehanizme obrade prekida, postoje koraci koji su svima njima zajednički

- U trenutku kada se pojavi zahtev za prekidom, pretpostavljajući da je procesor bio u toku izvršavanja proizvoljne instrukcije iz programa, procesor izvršava sledeće korake kako bi pristupio obradi prekida:
  1. Procesor prvo završava tekuću instrukciju čije je izvršavanje bilo u toku kada je stigao zahtev za prekidom. Procesor nikada ne skraćuje izvršavanje tekuće instrukcije. Zahtevi za prekidom uvek se obrađuju između instrukcija.

  2. Procesor smešta tekuću vrednost programskog brojača (PC) i u većini procesora tekuću vrednost statusnog registra (SR) na stek. Neki od procesora mogu na stek smestiti tekući sadržaj i drugih unutrašnjih registara.

  3. Procesor zabranjuje pojavu novih zahteva za prekidom dok se obrađuje tekući zahtev za prekidom. Kod nekih procesora moguće je kontrolisati koji novi zahtevi za prekidom mogu prekinuti obradu tekućeg zahteva za prekidom, uvođenjem sistema ***prioriteta prekida***.

Anatomija obrade zahteva za prekidom II

4. Procesor smešta adresu prve instrukcije prekidnog potprograma asociranog tekućem zahtevu za prekidom u programski brojač. Na ovaj način se kontrola izvršavanja programa prenosi na prvu instrukciju prekidnog potprograma.

5. Procesor izvršava sve instrukcije prekidnog potprograma. Prekidni potprogram završava se kada se izvrši posebna instrukcija koja označava povratak iz prekidnog potprograma (Interrupt Return, IRET, RETI).

6. Procesor vraća vrednost programskog brojača, statusnog registra, i eventualno još nekih registara, na vrednosti koje su bile sačuvane na steku u koraku 2. Ova akcija obično predstavlja deo izvršavanja instrukcije za povratak iz prekidnog potprograma. Postavljanje programskog brojača na staru vrednost omogućava procesoru da nastavi proces izvršavanja instrukcija na onom mestu gde je bio prekinut nailaskom zahteva za prekid.

Teorijska pitanja P6

Predavanja 6- Organizacija mikroračunarskog sistema 4

Spisak teorijskih pitanja uz Predavanja 6

1. Organizacija ulazno/izlaznog podsistema
2. I/O mapirane i memorijski mapirane periferijske jedinice
3. Anatomija I/O interfejsa. Navesti primer jednog realnog interfejsa.
4. Paralelni i serijski I/O interfejsi. Prednosti i mane svakog tipa.
5. Sinhronizacija procesora i I/O uređaja
6. Servisiranje zahteva periferijskih jedinica. Navesti i ukratko objasniti dva osnovna pristupa.
7. Servisiranje bazirano na prozivkama
8. Servisiranje bazirano na prekidima
9. Vrste prekida
10. Anatomija obrade zahteva za prekidom


entity test_shift is
generic (width : integer :=
shift_reg <= unsigned (inp);
elsif (en = '1') then