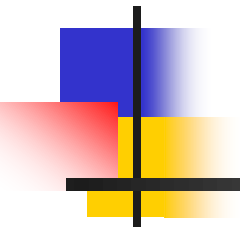


Ulazno/izlazni podsistem računarskih sistema

NRS predavanje br 8.

Zadatak i elementi UI podsistema

- Organizacija komunikacionih aktivnosti RS, odnosno kontrola razmene podataka između centralnog procesora i perifernih jedinica.
  - Kompletna komunikacija RS sa okruženjem, uključujući unos programa i podataka u OM, kao i prikaz rezultata obrade.
- Periferne U/I jedinice su neposredne tačke kontakta RS sa okolinom
  - U opštem slučaju to su električni ili elektromehanički uređaji
  - Performansa RS zavisi veoma mnogo od njihovih karakteristika
  - Često nose najveći deo cene u okviru računarskog sistema.
- Podela perifernih jedinica
  - Komunikacione jedinice
    - Ulazne, izlazne i ulazno-izlazne periferne jedinice,
    - Od tastature, miša, monitora i štampača, do serijskih i mrežnih uređaja
  - Spoljne memorije, ili spoljne memorijske jedinice
    - Namenjene za smeštanje informacija, kao što su
    - Diskovi, diskete, trake, optički diskovi i sl.

Sprežni podsistem

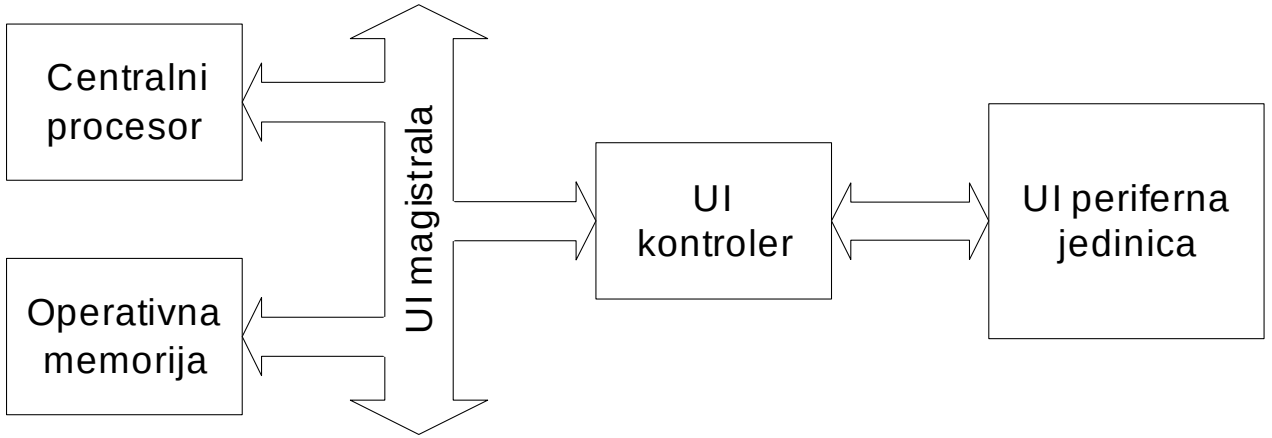
- Između perifernih jedinica, memorije i procesora,
- Zadatak - da upravlja radom U/I jedinica
  - prenos informacija sa njih (čitanje), odnosno ka njima (upis)
- Standardizacija sprežnog sistema
  - potrebno je povezati veći broj perifernih jedinica, različitih tipova i proizvođača
  - definiše se skup i struktura naredbi, upravljački signali i format informacionih reči potrebnih radi upravljanja U/I jedinicama.
- U/I sprega je metod prenosa podataka između internih memorija CP (OM i registara) i spoljnih U/I uređaja.
- Ovaj metod razrešava razlike između centralnog procesora i svake od periferijskih jedinica.

Glavne razlike između CP+OM i PJ

- Periferijske jedinice su generalno elektromehanički sklopovi, koji funkcionišu po principima različitim u odnosu na centralni procesor,
- Brzine prenosa podataka su mnogo manje za periferijske jedinice,
- Rad periferijske jedinice treba sinhronizovati sa centralnim procesorom i memorijom,
- Formati podataka se moraju uskladiti,
- Rad svake periferne jedinice mora biti kontrolisan tako da ne ugrozi rad centralnog procesora ili ostalih periferija u računarskom sistemu.

Opšta struktura UI podsistema u okviru računarskog sistema

- Standardno rešenje podrazumeva povezivanje svake od periferija sa namenskom upravljačkom jedinicom, koja vrši zadatke neposredne kontrole nad perifernim uređajem i komunikacije sa centralnim procesorom.

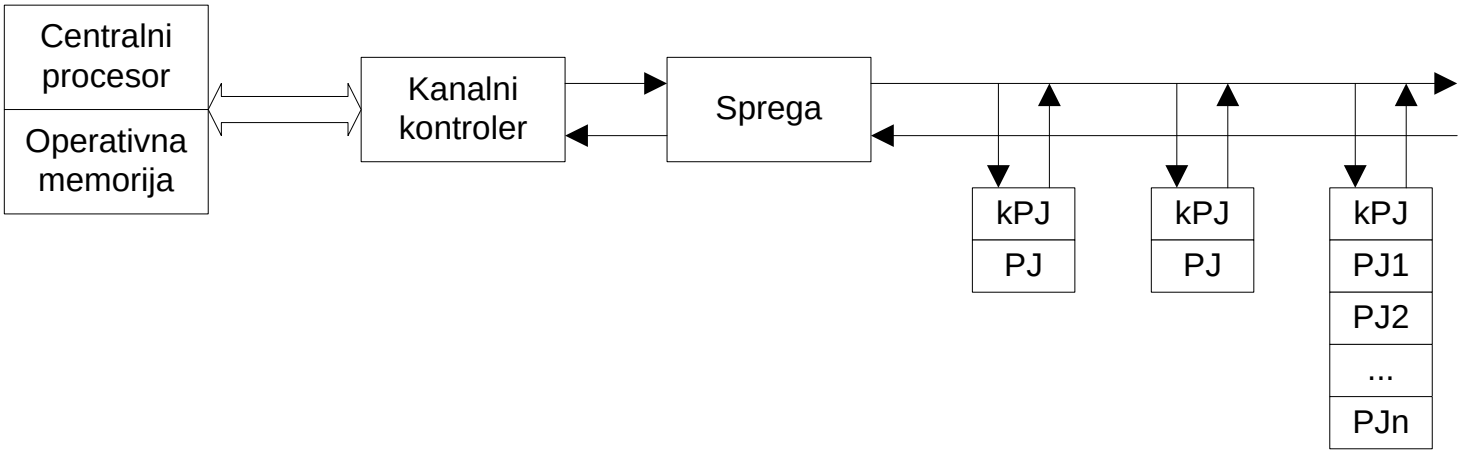


UI kontroler, U/I procesor

- Namenska upravljačka jedinica koja vrši pretvaranje standardnog skupa informacionih i upravljačkih reči i signala, u oblik koji obezbeđuje upravljanje radom konkretne periferne jedinice
- Vrši upravljanje razmenom informacija između perifernih i ostalih funkcionalnih jedinica računarskog sistema
- Realizacija UI kontrolera varira po složenosti, počev od prostog sprežnog sistema u obliku prihvatnih kola (bafera), do višeprocesorskih upravljačkih struktura
- Kompleksnost, naravno, zavisi od složenosti same periferne jedinice, njihovog broja, potrebne brzine prenosa i veličine elementa koji se prenosi (blok, reč, bajt)
- Realizuje paralelizam u radu procesora i perifernih jedinica
  - U slučaju serijskog obavljanja aktivnosti propusna moć sistema bila bi veoma mala

Model komunikacije centralnog procesora i perifernih jedinica

- Formira se **kanal** (realni ili virtualni) između CP i kontrolera periferne jedinice (kPJ)
- Adresa sa dva dela, (kanal, položaj u okviru kanala)
- Zavisno od brzine i obima podataka, kanal je
  - serijski ili paralelan,
  - asinhron ili sinhronizovan taktnim impulsima



Kanalni kontroler

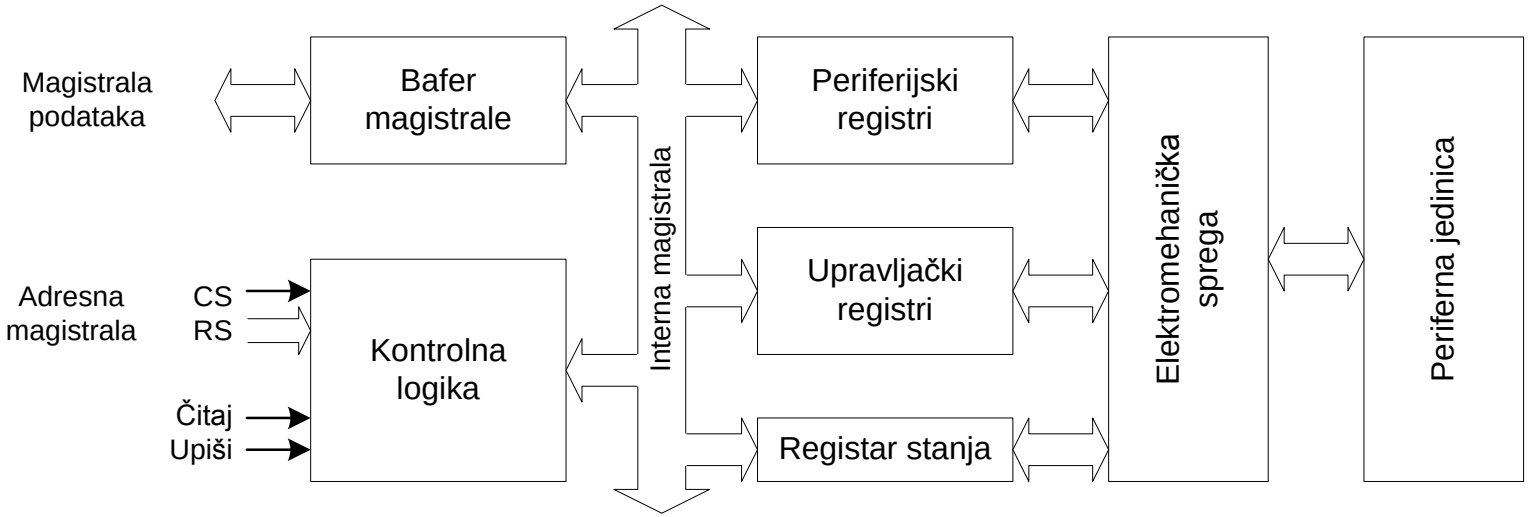
- **Multipleksorski kanal**
  - Po principu vremenske podele, u cilju opsluživanja nekoliko sporih U/I jedinica, a zahtevi za njihovim opsluživanjem su relativno retki
  - Izvršavanje UI operacija deli se na kratke vremenske intervale vremena
  - Brzina kanala >> od brzine periferijskog uređaja
- **Selektorski kanal**
  - Kanalni kontroler je vezan sa više PJ, i može u svakom momentu vremena izvršavati samo jednu UI operaciju
  - Za priključenje brzih PJ, sa intenzivnim režimom rada preko kanala
  - Ako je selektor priključen na jednu PJ, a do njega dođe komanda od procesora radi obraćanja drugoj, njeno izvršavanje se zadržava
  - Moguć je slučaj kad se realizuju komande koje ne traže zauzeće kanala i potom se autonomno izvršavaju u okviru periferne jedinice. Tako se može započeti redom nekoliko operacija koje se izvršavaju u perifernim jedinicama, paralelno po vremenu i međusobno nezavisno.
- **Krosbar veza -** svaki kanalni kontroler je spojen sa svakom od PJ

Komande centralnog procesora

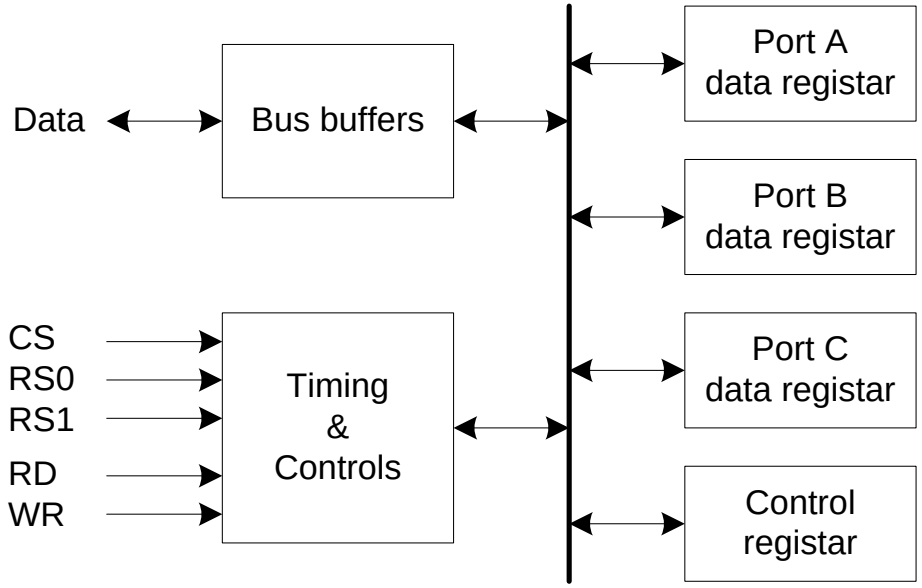
- Upravljačke - komanduju radom PJ
- Upitne (test) - u kom je stanju PJ
- Slanje podataka - praćeno upitom da li je prenos završen
- Prijem podataka - ukoliko je upitom utvrđeno postojanje podataka u perifernom uređaju
- Stoga U/I instrukcije moraju sadržati
  - adresu uređaja, i
  - kod funkcije koju kontroler treba da izvrši
- Kod savremenih procesora i U/I kontrolera, ovo se odrađuje selekcijom internih registara u okviru kontrolera

Blok dijagram kontrolera PJ

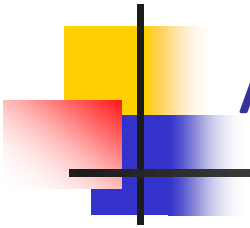
- Registri različitog tipa: naredbi, stanja, podataka
- Ulazno-izlazni prolaz (***port***)
- CS, RS, RD/WR, podaci



Sprežni signali PIO kola 8255



CS	RS1	RS2	Izbor
0	×	×	-
1	0	0	Port A
1	0	1	Port B
1	1	0	Port C
1	1	1	Control



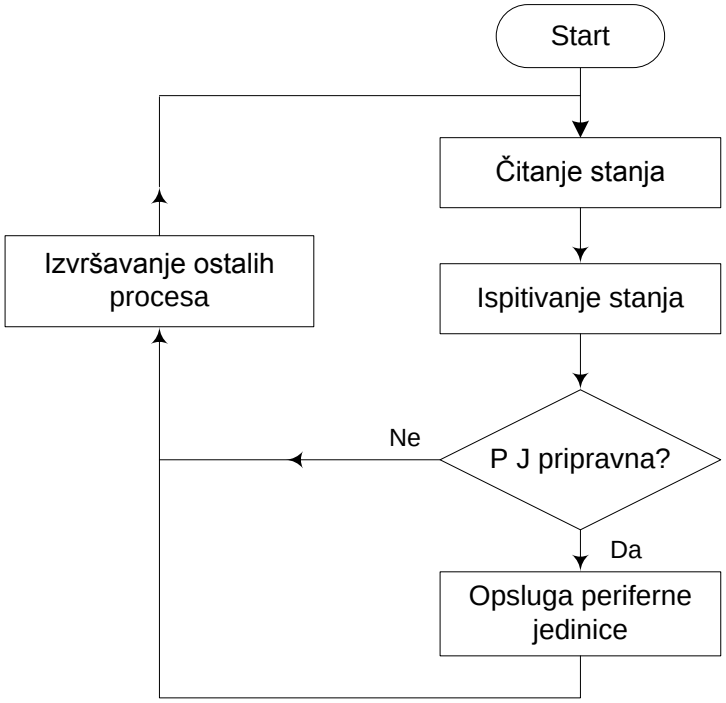
Adresiranje UI uređaja

- **Memorijski preslikani ulazi/izlazi** (*Memory-Mapped I/O*)
  - upotreba Standardnih RD/WR signala
  - U/I prolazima dodeljuju se adrese iz memorijskog adresnog prostora
- **Izolovan ulaz/izlaz** (*Isolated I/O*)
  - Posebne UI instrukcije (*in, out*)
  - Koje angažuju IORD, IOWR signale
  - Dva paralelna, međusobno izolovana adresna prostora

Tehnike komunikacije CP sa UI uređajima

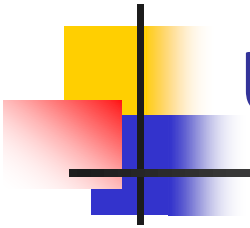
- Cilj komunikacije - uvek prenos podataka OM $\Leftrightarrow$ PJ
- **Indirektni prenos** - centralni procesor učestvuje kao posrednik
  - Tehnika programiranih ulazno-izlaznih aktivnosti, program ispituje stanje svake periferne jedinice i pokreće izvršenje adekvatnih U/I operacija – **polling**.
  - Prenos podataka počinje na zahtev UI jedinice, koja ukazuje procesoru da je spremna za izvršenje UI aktivnosti. Sistem prekida – ***interrupts***.
- **Direktni prenos** između UI podsistema i memorije (**DMA**)
  - Bez posredstva centralnog procesora, izuzev na početku i na kraju tog prenosa
  - Kanalni kontroler preuzima kontrolu magistrale i sam započinje memorijski ciklus.
  - Kontroler zakida memorijski ciklus centralnog procesora, jer su oba vezani za memoriju.
  - Kada i procesor i kanalni kontroler istovremeno zahtevaju memorijski ciklus, obično se prioritet daje kanalnom kontroleru, jer propuštanje ciklusa od strane kontrolera može da izazove velika kašnjenja.

Programirani ulaz-izlaz



- ***Polling*** - prozivanje, beleženje glasova
- Po inicijalizaciji sistema, pokreće se UI rukovaoc (*driver*), koji:
  - Periodično očitava stanje UI jedinice, ispitivajem bita stanja pripravnosti
  - Ako jedinica nije pripravna CP radi druge stvari, u suprotnom vrši prenos podataka
  - Vremenski interval zavisi od brzine UI jedinice

UI aktivnost na bazi prekida



- Prekidi su se u arhitekturi računara pojavili usled potrebe za povećanjem efikasnosti rada računarskog sistema, pojavom druge generacije računarskih sistema.
- Osnovni nedostatak rada sa programiranim ulazom-izlazom je nekorisno trošenje vremena CP na ispitivanje stanja PJ.
- UI aktivnosti koje se pokreću prekidima znatno umanjuju ovaj nedostatak.
- Osnovna ideja - iniciranje prenosa podataka od strane PJ.
- Signalom prekida UI jedinica ukazuje centralnom procesoru da je pripravna za prenos podataka.
- UI jedinica može da koristi mehanizam prekida i za druge namene, prijavu kvara ili završetka lokalne operacije.

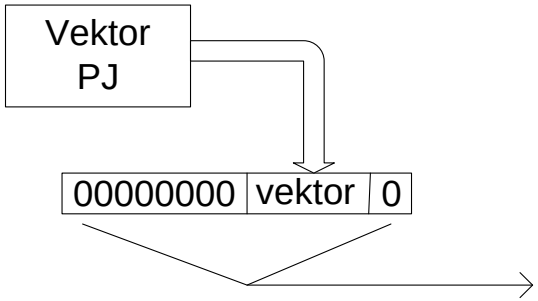
Klasifikacija prekida

- **Fizički (hardverski) prekidi**
  - ***Spoljne*** (eksterne) prekide izazivaju U/I uređaji, klasični prekidi
    - Maskirani ili nemaskirani
  - ***Unutrašnji*** (interni) prekidi su produkt fizičkih komponenti centralnog procesora, najčešće kao posledica nepravilnog izvršenja instrukcija.
    - Ovi prekidi se označavaju kao **zamke** ili **izuzeci** (*trap*, *exception*)
    - *Overflow*, *divide-overflow*, *stack-overflow*, *protection violation i* sl.
- **Programski (softverski) prekidi**
  - Izazivaju se upotrebom posebne instrukcije (***int***)
  - Pogodan način za povezivanje aplikativnog programa sa uslužnim rutinama operativnog sistema, gde je često potrebna promena režima rada od korisničkog u sistemski
  - Komunikacija koja je pri tom neophodna, tj. prenos argumenata i rezultata, odvija se posredstvom registara centralnog procesora.

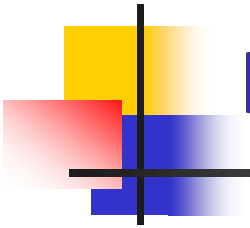
Izvršenje prekida

- Slično izvršenju potprograma, ali sada fizička arhitektura (HW) obezbeđuje adresu grananja. Dva su načina izbora ove adrese:
- **Nevektorski metod** podrazumeva da je adresa grananja fiksna memorijska lokacija, nakon čega je neophodna programska potraga za uređajem koji je prekid izazvao.
- **Vektorski metod** je rešenje kod koga se izvor prekida, posredstvom magistrale podataka, identifikuje centralnom procesoru.
  - Predajom svoje identifikacije, periferna jedinica istovremeno predaje i informaciju o adresi grananja.
  - Ta informacija se naziva **vektor prekida**, na osnovu koje procesor izračunava adresu početka prekidne rutine, ili adresu memorijske reči u kojoj je ta adresa zapisana.

Ilustracija upotrebe vektora prekida



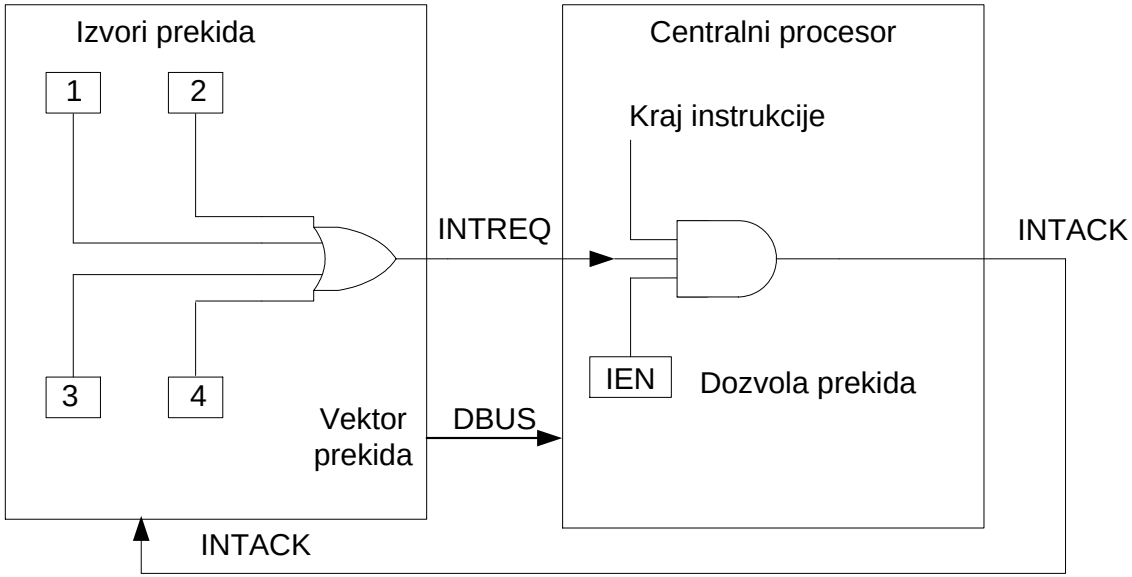
0	Adresa veze PA
1	SA
2	Adresa veze
3	SB
4	Adresa veze
	SC



Izvršenje prekida

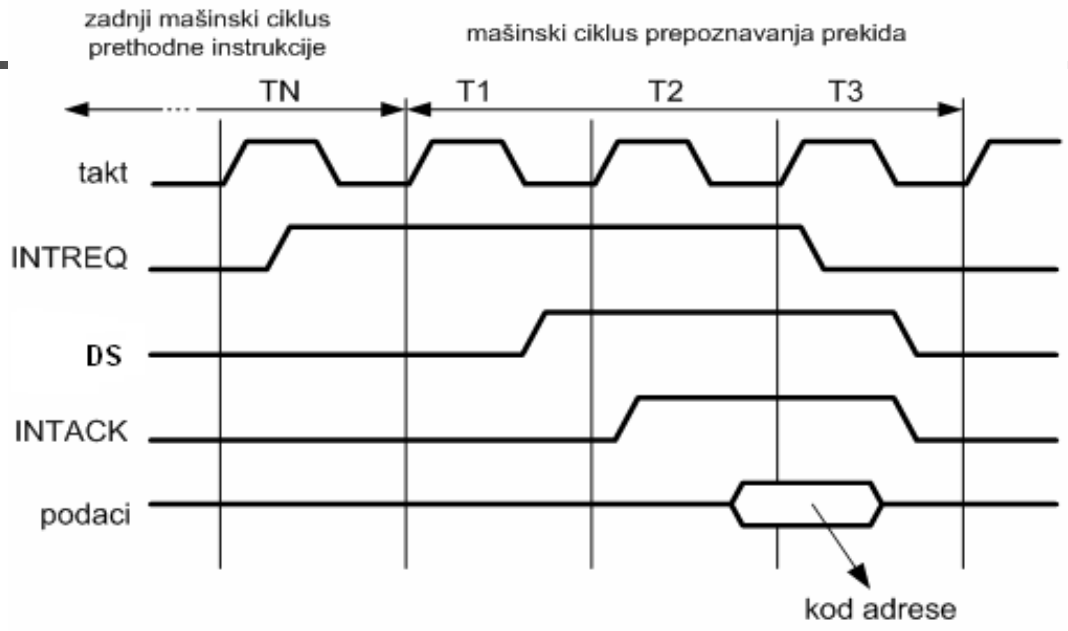
- Prekid prisiljava centralni procesor da preduzme određeni sled akcija.
- Pošto se prihvati od strane centralnog procesora:
  - prekida se izvršavanje tekućeg programa;
  - smešta se stanje tekućeg programa;
  - prelazi se na rutinu za opsluživanje prekida;
  - vraća se i obnavlja izvršavanje prekinutog programa
- Mehanizam prekida mora u celini biti automatizovan komponentama fizičke arhitekture
- Sadržaj izmenjenih registara se čuva u memoriji, za šta se koristi *stack* ili odvaja specijalna memorijska zona.
- Stanje upravljačkih signala se ne može očuvati, jer bi ih sama operacija smeštanja razrušila.

Sprega izvora prekida i centralnog procesora



- Očigledno je potreban metod za određivanje izvora prekida, kao i za određivanje međusobnog prioriteta perifernih uređaja
  - Programski metod
  - Fizički metod

Mašinski ciklus prekida



Ulazak u prekid (INT)

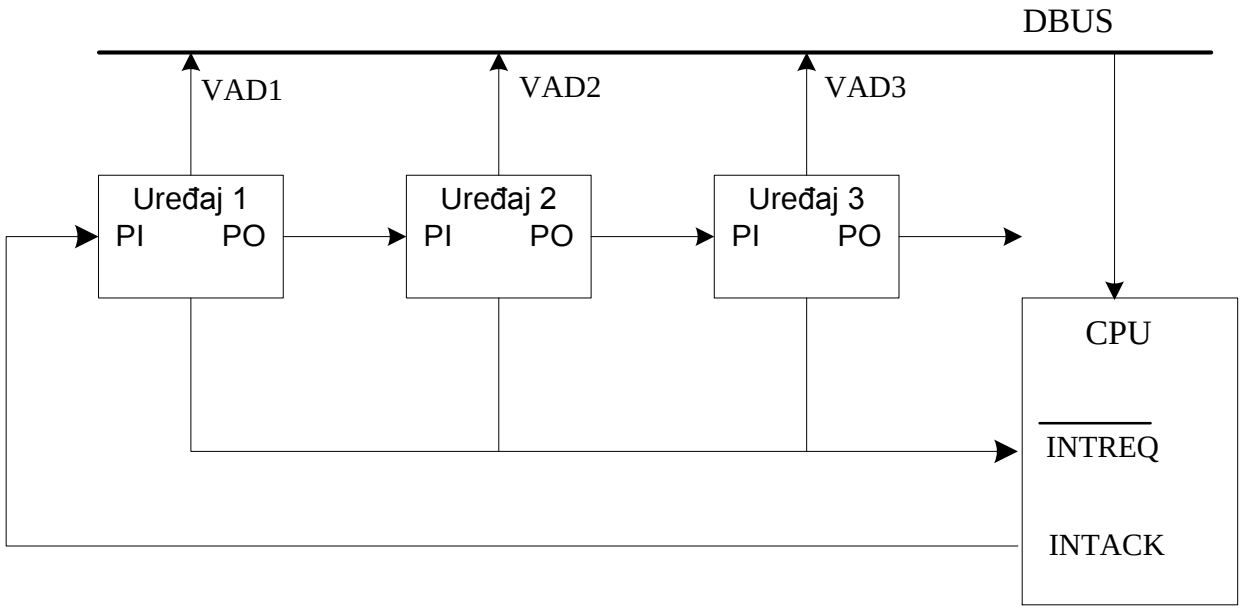
```
SP ← SP+2
M[SP] ← PC
INTACK ← 1
PC[H] ←0, PC[L] ← DBUS(VAD)
IEN←0, INTACK←0
```

Povratak iz prekida (RETI)

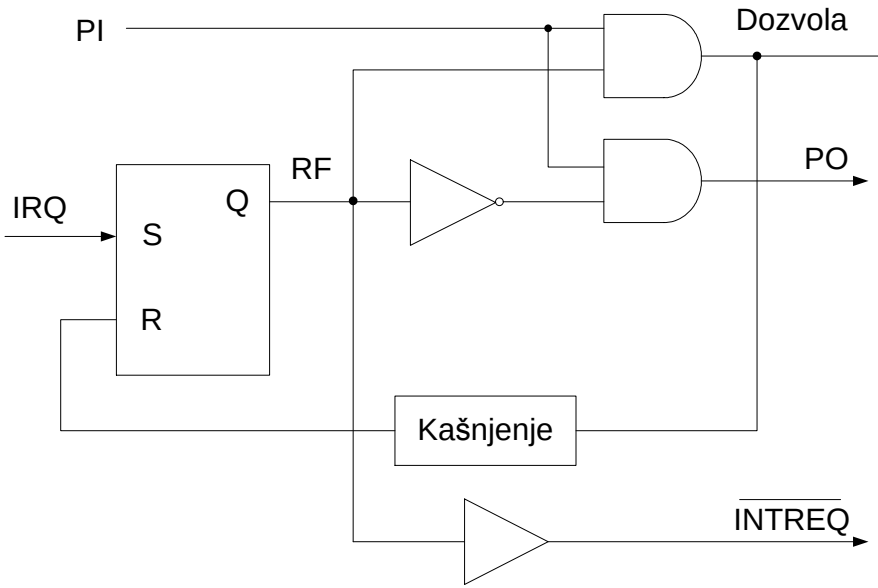
```
PC←M[SP]
IEN←1
```

Serijski kontroler prekida

- Podrazumeva serijsku vezu uređaja koji generišu prekide
  - *Daisy-chain* lanac prioriteta

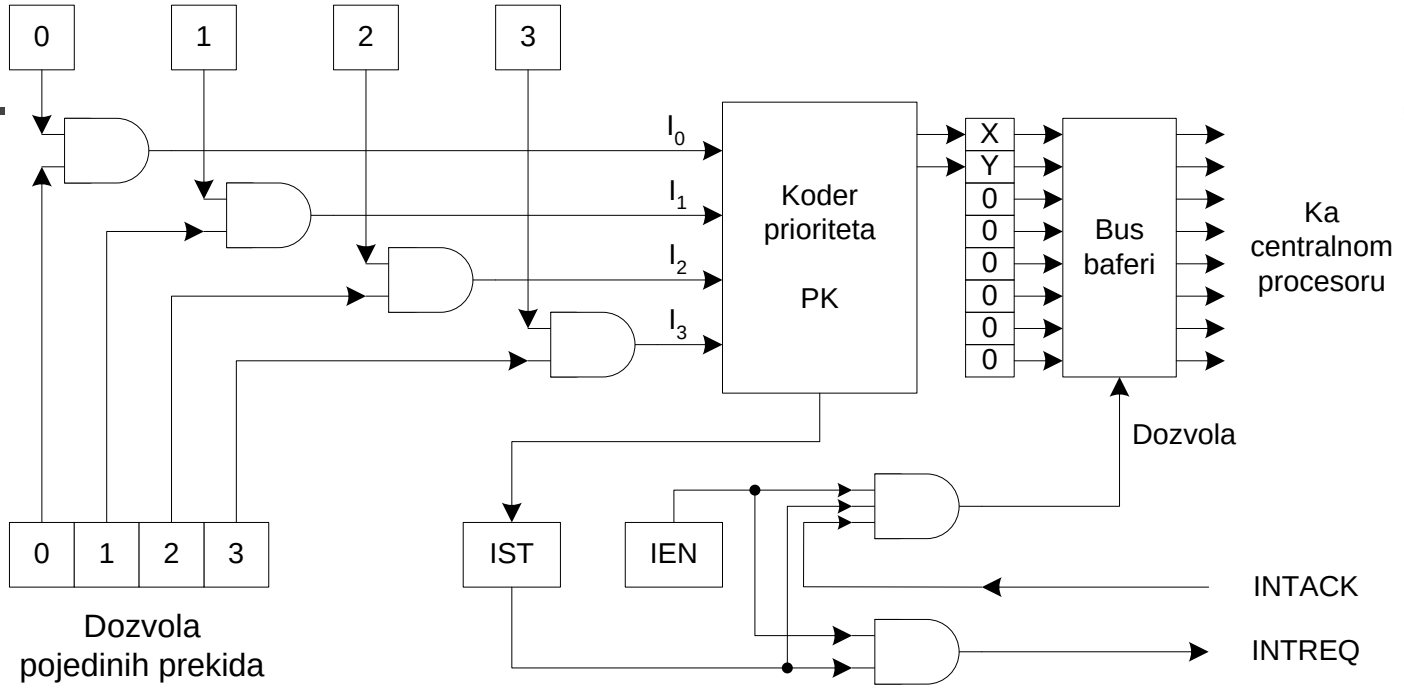


Logika za razrešenje prioriteta kod serijske sprege UI uređaja



PI	RF	PO	Dozvola
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Paralelni kontroler prekida



Paralelna veza prekidnih linija omogućuje dodatne funkcije, poput pojedinačne dozvole prekida i izmene njihovih prioriteta

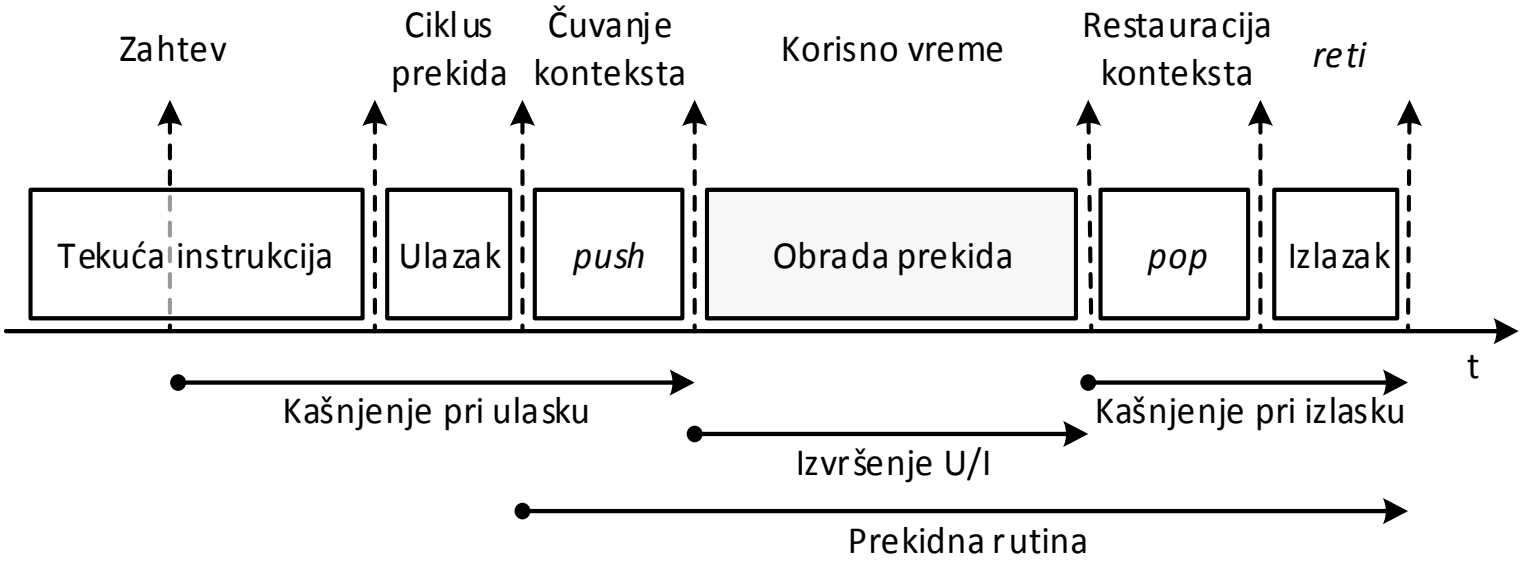
I_0	I_1	I_2	I_3	X	Y	IST
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1
0	0	0	0	x	x	0

$X = \overline{I_0}\,\overline{I_1}$

$Y = \overline{I_0} I_1 + \overline{I_0}\,\overline{I_2}$

$IST = I_0 + I_1 + I_2 + I_3$

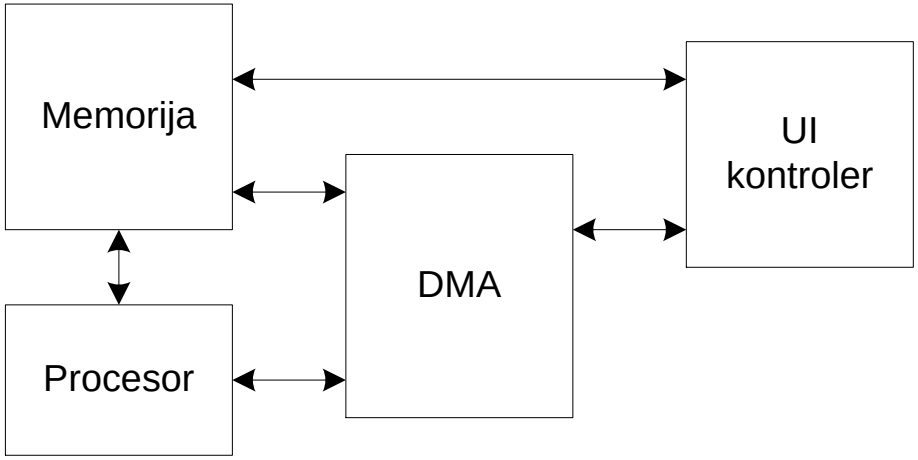
Efikasnost prenosa pomoću prekida



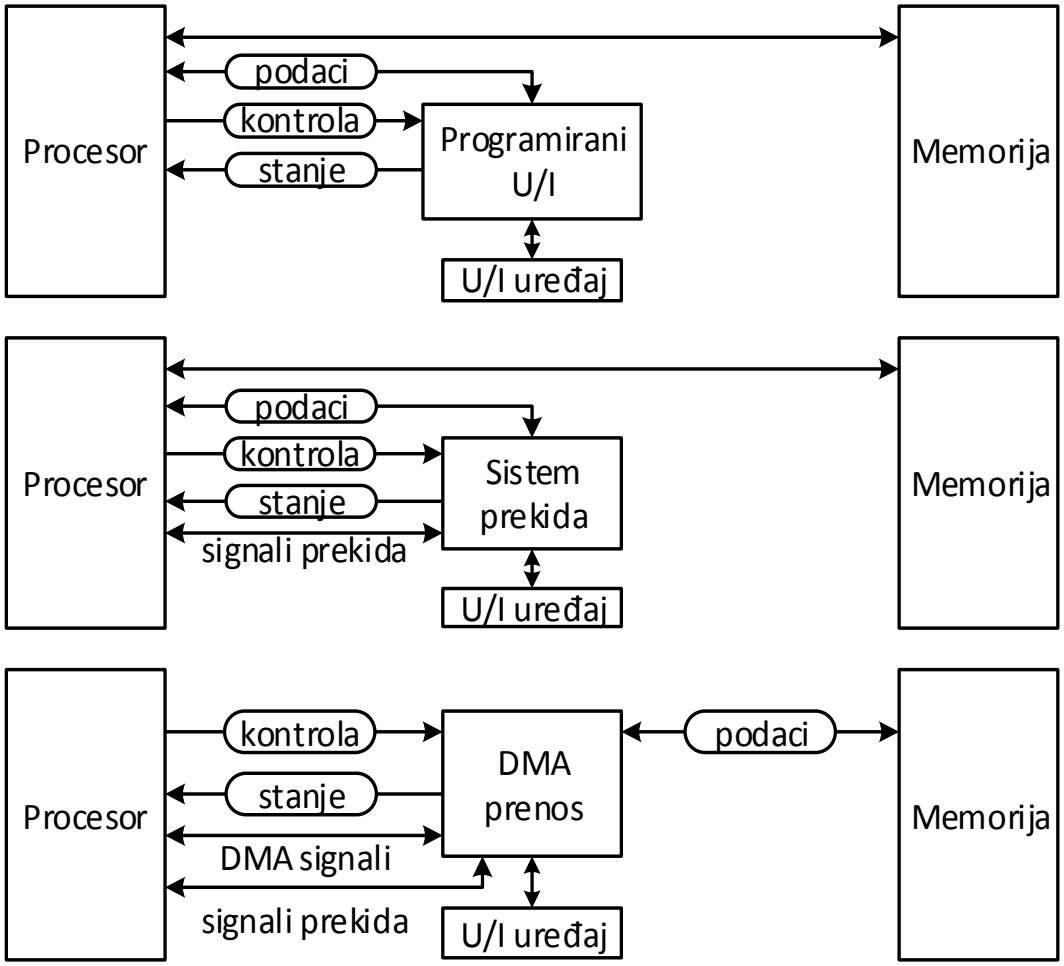
Direktan pristup memoriji

DMA - Direct Memory Access

- Sistem prekida nije dovoljno efikasan u slučaju brzih UI jedinica
- DMA kontroler prenosi blok podataka
- Suština ove tehnike je preuzimanje memorijskih ciklusa procesora
- Zato se ovakav način rada naziva **zakidanje**, ili **krađa ciklusa** (*cycle stealing*)



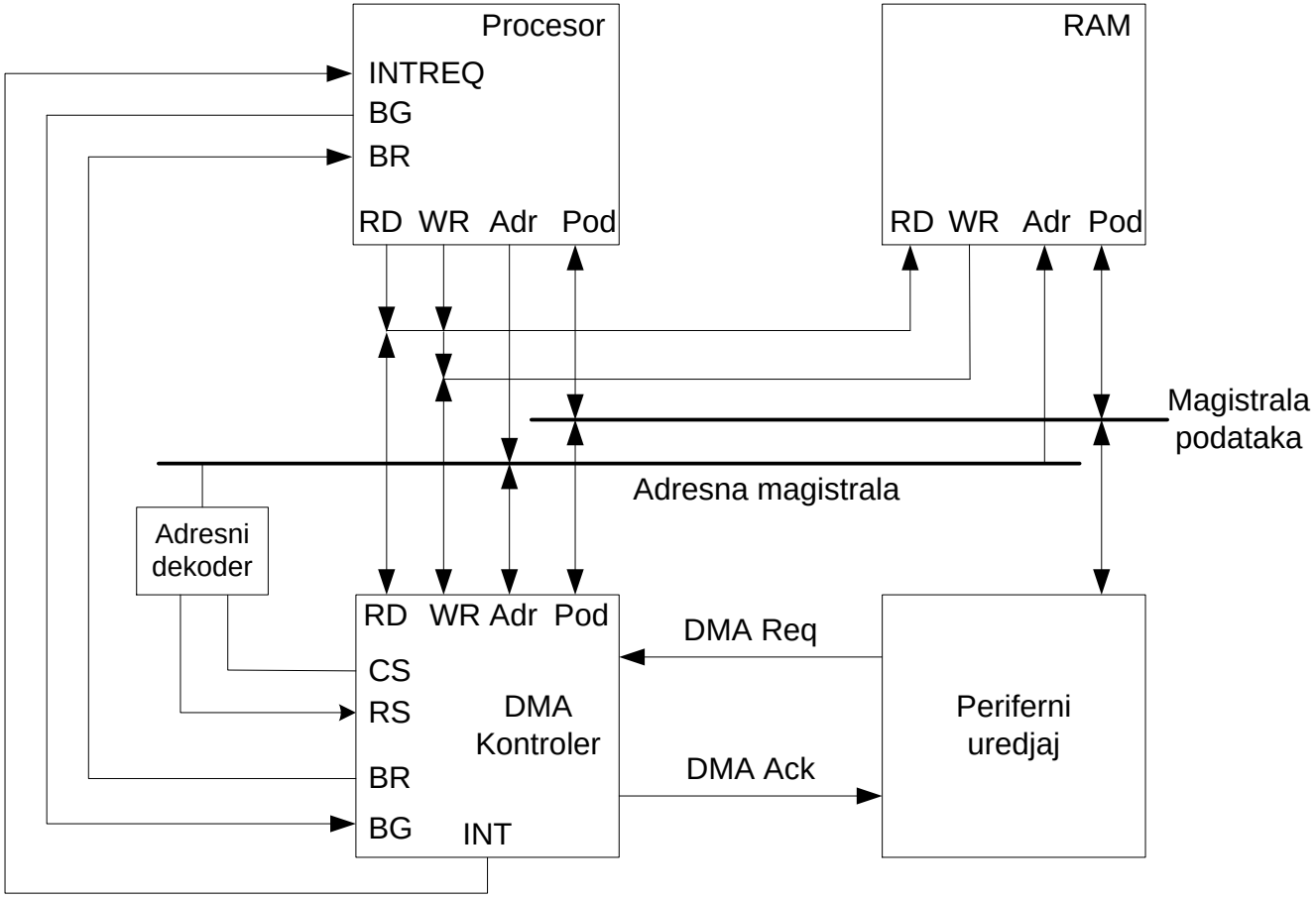
Ilustracija tri režima UI komunikacije



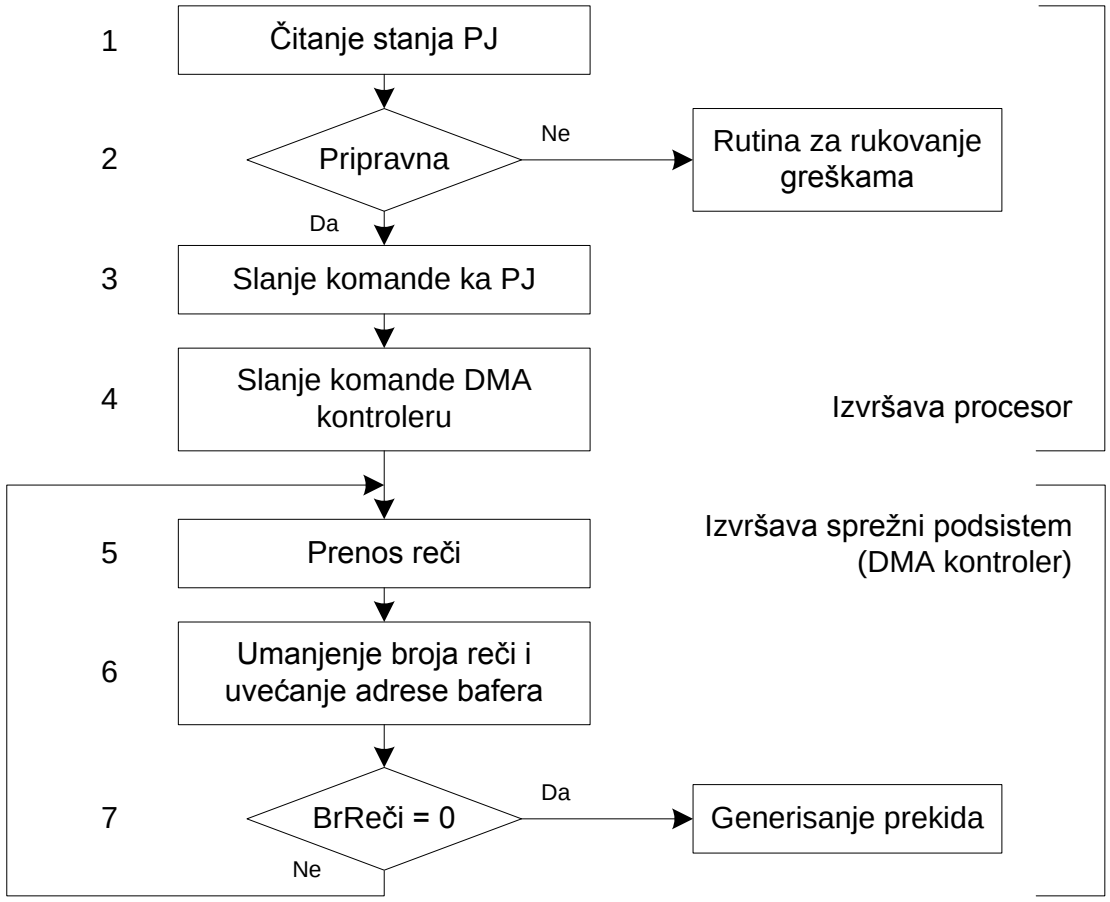
Inicijalizacija DMA prenosa

- DMA prenos se inicijalizuje pod kontrolom centralnog procesora, zadavanjem:
  - početne adrese zone podataka u memoriji (*start adress*)
  - broja znakova, tj. dužine bloka koji se prenosi (*byte count*)
  - smer prenosa (upis ili čitanje)
  - komanda za početak (*start DMA*)
- Signali zahteva i dodele magistrale (BR, BG)
- Pre prenosa upravljanja nad magistralom ka DMA kontroleru i perifernoj jedinici, centralni procesor mora da:
  - završi bilo koji mašinski ciklus na magistrali koji je u toku
  - da prebaci svoje izlazne linije na odspojeno stanje, tj. stanje visoke impedanse

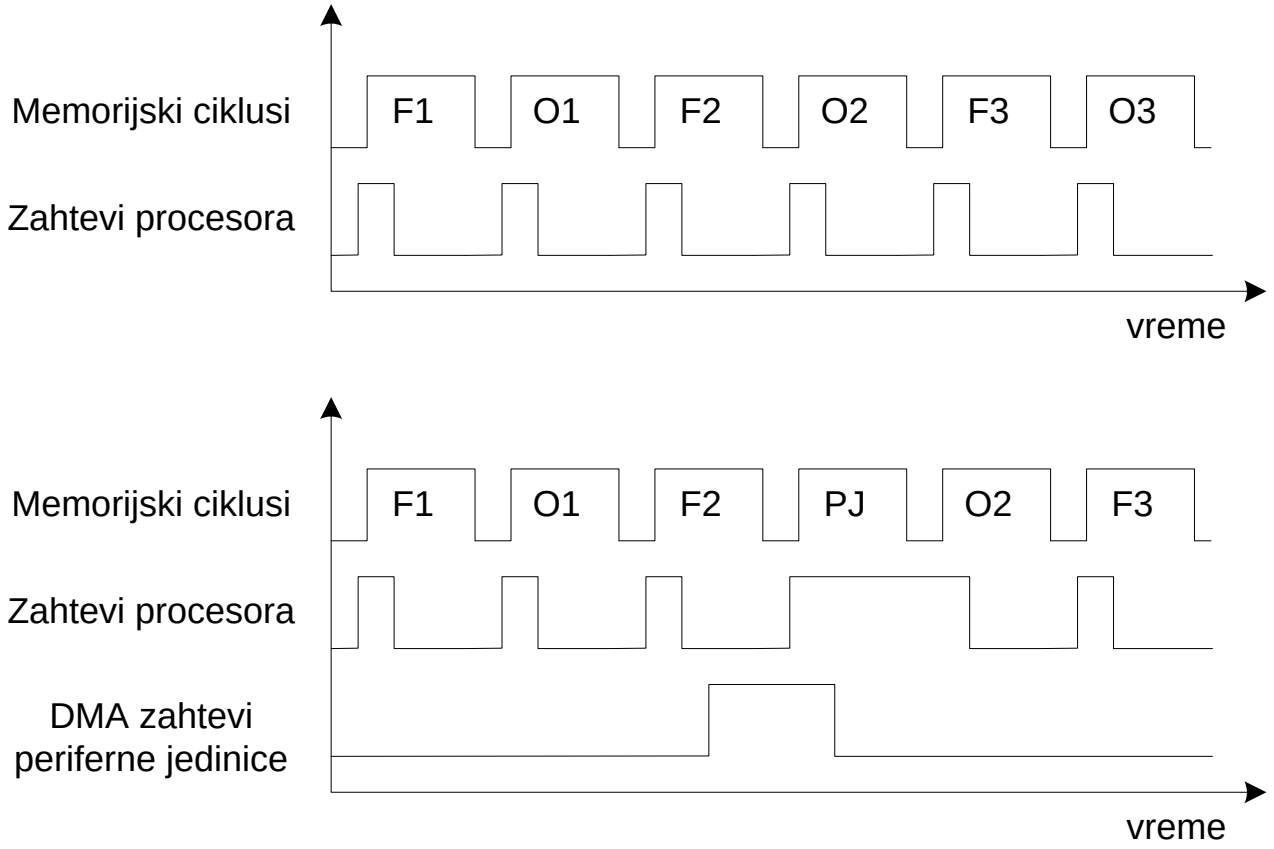
Organizacija DMA prenosa



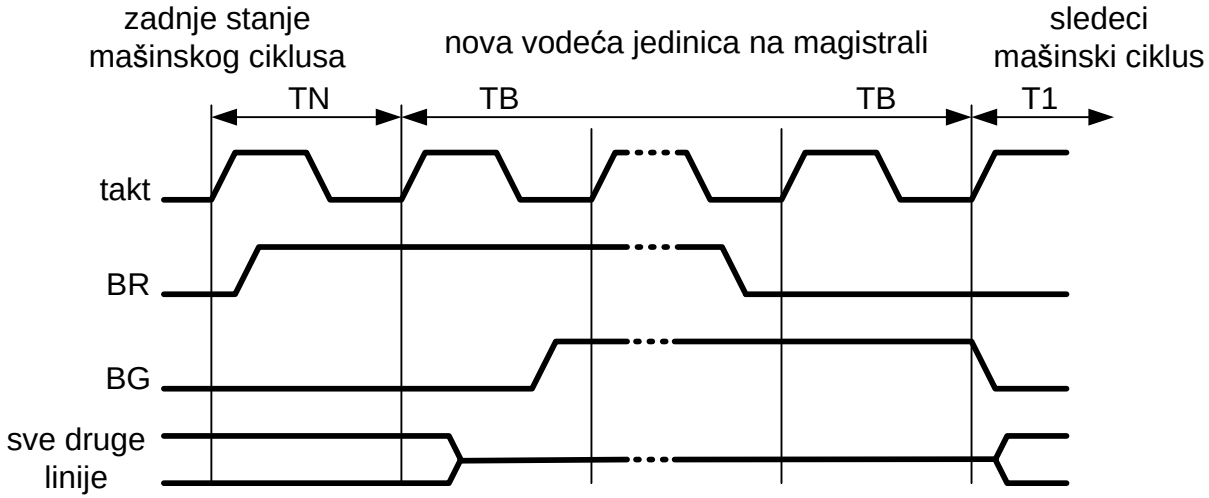
Organizacija aktivnosti u slučaju DMA prenosa



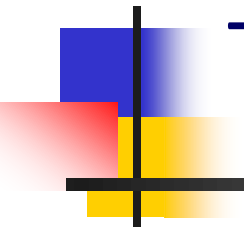
Zakidanje ciklusa



Vremenski redosled signala preuzimanja magistrale



- Radi razrešenja potencijalnih konflikata, mora postojati adekvatan mehanizam baziran na prioritetu svake od perifernih jedinica.
- Zahtevi od strane centralnog procesora tretiraju se kao i svi ostali, čak se njemu daje niži prioritet nego perifernim jedinicama.



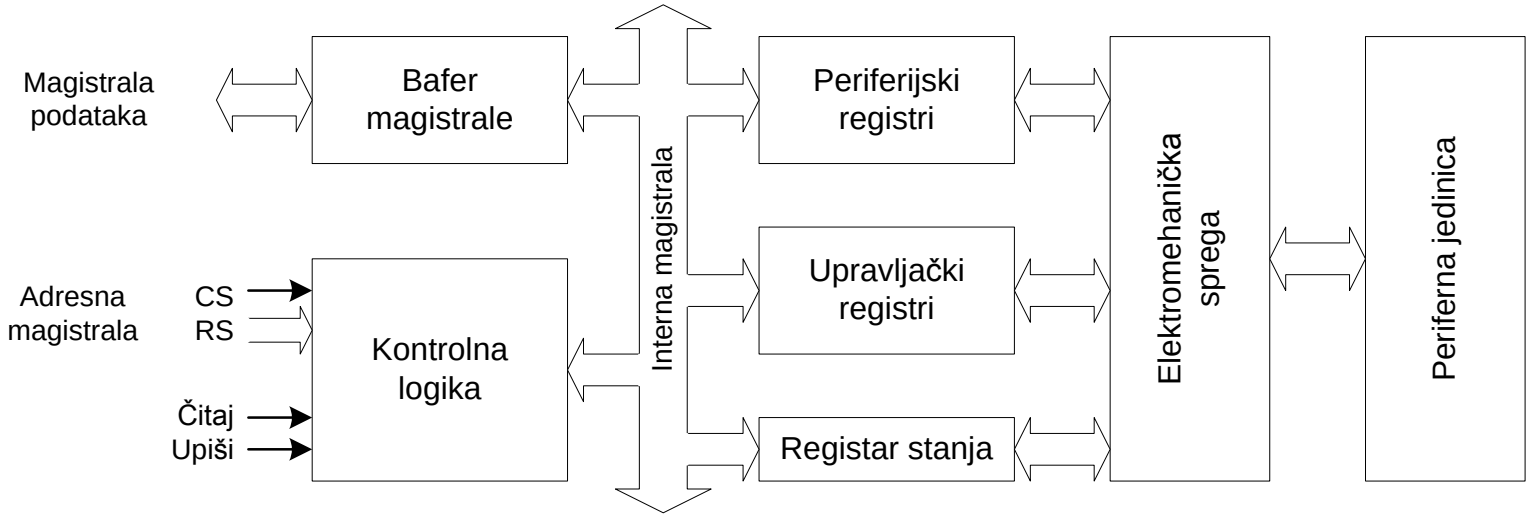
Tehnike UI programiranja

Programska kontrola UI uređaja

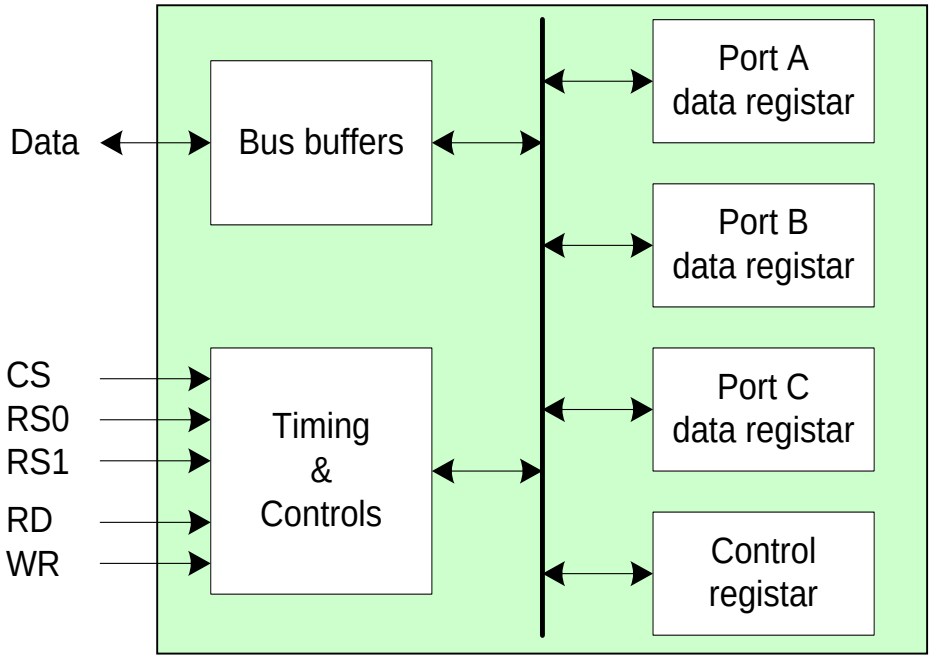
Rukovanje Ulazom/Izlazom, prekidi, DMA

Blok dijagram kontrolera PJ

- Registri različitog tipa: naredbi, stanja, podataka
- Ulazno-izlazni prolaz (***port***)
- CS, RS, RD/WR, podaci



Sprežni signali PIO kola 8255



CS	RS1	RS2	Izbor
0	×	×	-
1	0	0	Port A
1	0	1	Port B
1	1	0	Port C
1	1	1	Control

Adresiranje UI uređaja

- **Memorijski preslikani ulazi/izlazi**
  - Upotreba Standardnih RD/WR signala
  - U/I prolazima dodeljuju se adrese iz memorijskog adresnog prostora
  - Pristup iz C programa – putem pointera kao i svakoj drugoj promenljivoj
- **Izolovan ulaz/izlaz**
  - Posebne UI instrukcije (*in, out*)
  - Koje angažuju IORD, IOWR signale
  - Dva paralelna, međusobno izolovana adresna prostora
  - C funkcije: inport (_inp), outport (_outp) (byte, word)

Pristup 8255 kolu na oba načina

```
#define PORTA 0
#define PORTB 1
#define PORTC 2
#define PCNTR 3

// memorijski mapiran pristup
volatile unsigned char *gpIO = (unsigned char *) 0xBFF0000;

   gpIO[PCNTR] = 0x80;      // set ports OUT OUT OUT
   ...
   gpIO[PORTB] = 0x55;      // put 0x55 to port B

// Izolovani IO prostor
#define GPIO 0x03F0

   outportb( GPIO+PCNTR, 0x80 );   // set ports OUT OUT OUT
   ...
   outportb( GPIO+PORTB, 0x55 );   // put 0x55 to port B
```

Tehnike komunikacije CP sa UI uređajima

- Cilj komunikacije - uvek prenos podataka OM ⇔ PJ

- **Indirektni prenos** – CP učestvuje kao posrednik
  - Tehnika programiranih ulazno-izlaznih aktivnosti, (***polling***)
  - Sistem prekida (***interrupts***)
- **Direktni prenos** - DMA
  - Bez posredstva centralnog procesora, izuzev na početku i na kraju tog prenosa
  - Kanalni kontroler preuzima kontrolu magistrale i započinje samostalno memorijski ciklus.

Tehnika programiranih UI aktivnosti

- PIC 18F8722
- memorijski mapiran UI

```
#BYTE PIR1      = 0xF9E
...
   #BIT RC1IF   = PIR1.5
   #BIT TX1IF   = PIR1.4
...
#BYTE TXREG1    = 0xFAD
#BYTE RCREG1    = 0xFAE
```

```
void putc1( char ch)
{
   while( !TX1IF );
   TXREG1 = ch;
}

char getc1( void )
{
   while( !RC1IF )
      delay( 10_MSEC );
   ch = RCREG1;
}
```

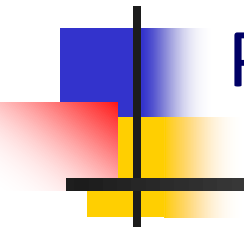
Sistem prekida

- Slično izvršenju potprograma, ali sada HW obezbeđuje adresu grananja, na jedan od dva načina:
  - **Nevektorski metod** – fiksna adresa grananja
  - **Vektorski metod** - izvor prekida šalje identifikaciju ka CPU
- Pošto se prekid prihvati od strane centralnog procesora:
  - prekida se izvršenje tekućeg programa (po završetku instrukcije);
  - smešta se stanje tekućeg programa;
  - prelazi se na rutinu za opsluživanje prekida;
  - vraća se i obnavlja izvršavanje prekinutog programa
- Sadržaj registara koji se menaju čuva se u memoriji
  - *stack* ili druga specijalna memorijska zona

Vektorski i nevektorski prekid

Ulazak u prekid (INT)	**Povratak iz prekida (RETI)**
SP ← SP + 1 M[SP] ← PC INTACK ← 1 PC ← AdrFun(VAD) IEN ← 0, INTACK ← 0	PC ←M[SP] SP ← SP-1 IEN ←1

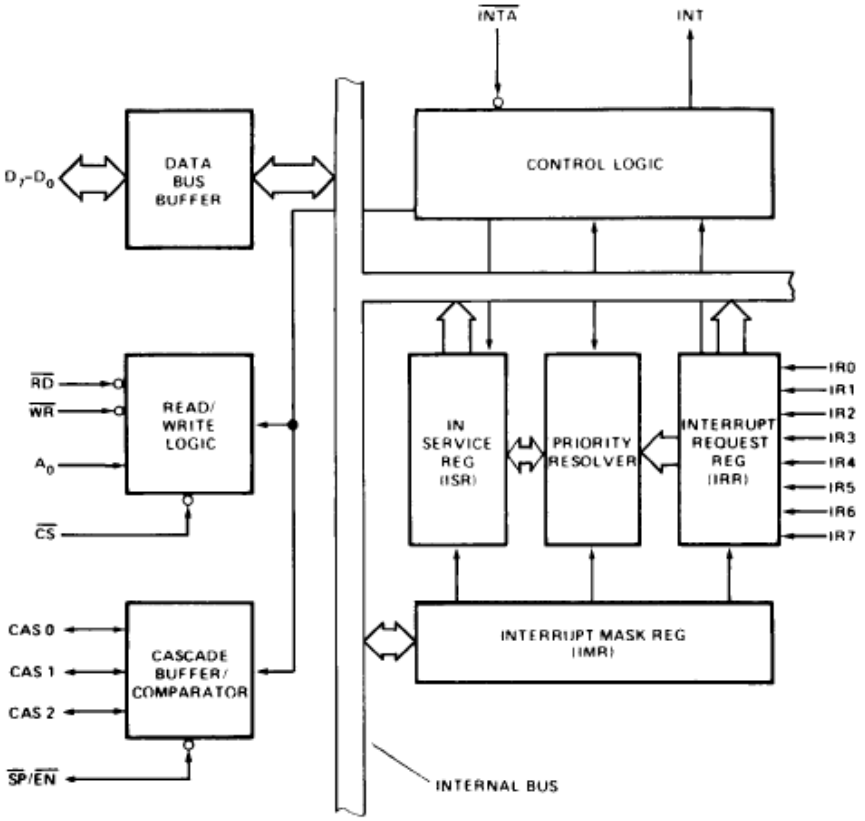
Ulazak u prekid (INT)	**Povratak iz prekida (RETI)**
SP ← SP + 1 M[SP] ← PC INTACK ← 1 PC ← CONST_ADDR IEN← 0, INTACK ← 0	PC ←M[SP] SP ← SP-1 IEN ←1

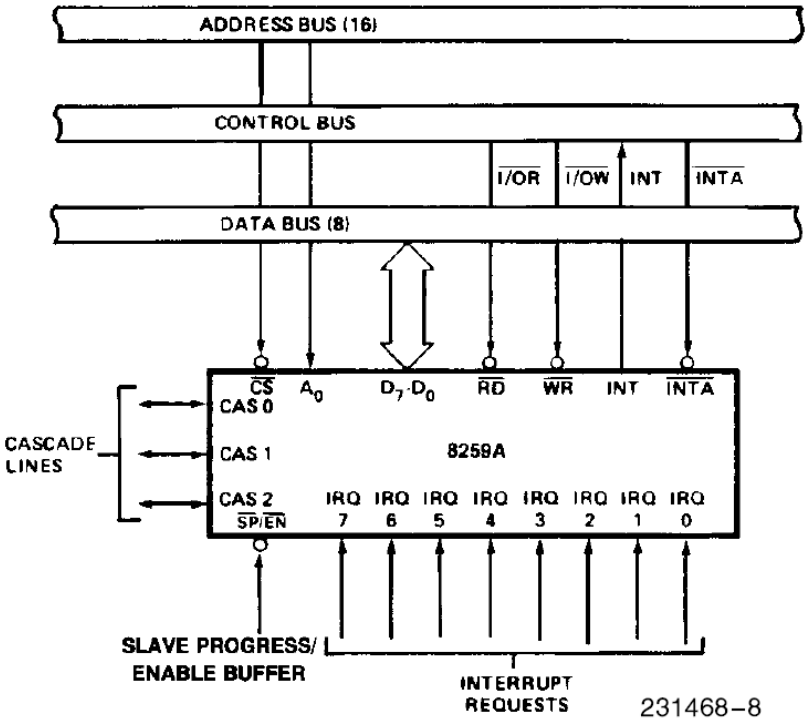


Primer rukovanja prekidima na PC platformi

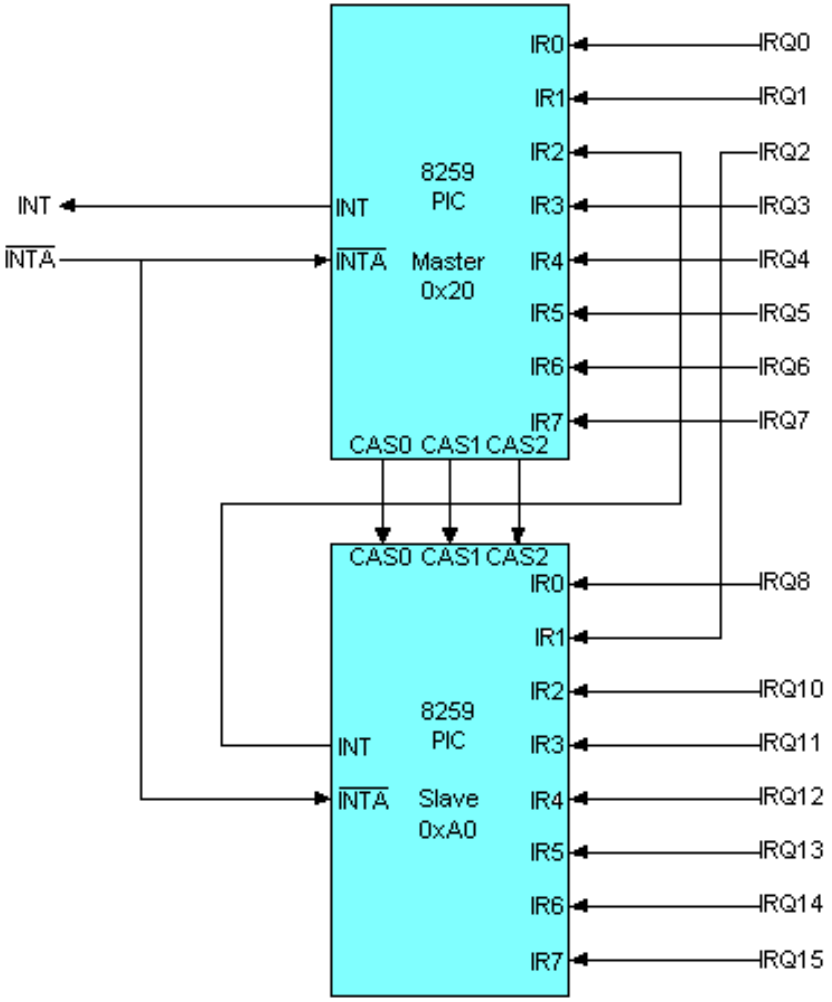
Real Time Clock Interrupt Handler

8259A Programmable Interrupt Controller





PC sistem prekida



INT	IRQ	Common Uses
00 - 01	Exception Handlers	-
02	Non-Maskable IRQ	NMI (Parity Errors)
03 - 07	Exception Handlers	-
08	Hardware IRQ0	System Timer
09	Hardware IRQ1	Keyboard
0A	Hardware IRQ2	Redirected
0B	Hardware IRQ3	COM2/COM4
0C	Hardware IRQ4	COM1/COM3
0D	Hardware IRQ5	Reserved/Sound Card
0E	Hardware IRQ6	Floppy Disk Controller
0F	Hardware IRQ7	Parallel Comms.
10 - 6F	Software Interrupts	-
70	Hardware IRQ8	Real Time Clock
71	Hardware IRQ9	Redirected IRQ2
72	Hardware IRQ10	Reserved
73	Hardware IRQ11	Reserved
74	Hardware IRQ12	PS/2 Mouse
75	Hardware IRQ13	Math's Co-Processor
76	Hardware IRQ14	Hard Disk Drive
77	Hardware IRQ15	Reserved
78 - FF	Software Interrupts	

Organizacija PC prekida

- x86 arhitektura dozvoljava do 256 prekida
  - Exceptions, NMI, HW, SW
  - Vektorska organizacija
- Interrupt Vector Table
  - 256 x 4 = 1024 byte
  - Sadrži 256 ukazivača na prekidne rutine
  - Zauzima prostor od 0000:0000h do 0000:03FCh
  - Inicijalizuje se po startu računara (BIOS, OS)
- Korisnik može prijaviti svoju funkciju kao novu ISR (*Interrupt Service Routine*)
  - getvect, setvect DOS uslužne rutine

Sadržaj IVT

- HW prikazani
- SW izostavljeni (DOS)
- Preslikavanje
  - Int# * 4 = Adresa

Interrupt	Address	Type	Function
00h	0000:0000h	Exc	Divide By Zero
01h	0000:0004h	Exc	Single Step
02h	0000:0008h	HW	Nonmaskable Interrupt (NMI)
03h	0000:000Ch	Exc	Breakpoint Instruction
04h	0000:0010h	Exc	Overflow Instruction
05h	0000:0014h	HW	Bounds Exception
06h	0000:0018h	HW	Invalid Op Code
07h	0000:001Ch	HW	Math Coprocessor Not Present
08h	0000:0020h	HW	Double Exception Error
08h	0000:0020h	HW	System Timer - IRQ 0
09h	0000:0024h	HW	Keyboard - IRQ 1
0Ah	0000:0028h	HW	IRQ 2 - Cascade from Second PIC
0Bh	0000:002Ch	HW	IRQ 3 - COM 2
0Ch	0000:0030h	HW	IRQ 4 - COM 1
0Dh	0000:0034h	HW	Parallel Printer (LPT 2)
0Eh	0000:0038h	HW	IRQ 6- Diskette Drive Controller
0Fh	0000:003Ch	HW	IRQ 7 – Parallel printer (LPT 1)
...	...	...	...
70h	0000:01C0h	HW	IRQ 8 - Real Time Clock
71h	0000:01C4h	HW	IRQ 9 - Redirect Cascade
72h	0000:01C8h	HW	IRQ 10 - General Adapter Use
73h	0000:01CCh	HW	IRQ 11 - General Adapter Use
74h	0000:01D0h	HW	IRQ 12 - PS/2 Mouse
75h	0000:01D4h	HW	IRQ 13 - Math Coprocessor Exception
76h	0000:01D8h	HW	IRQ 14 - Primary HDisk Controller
77h	0000:01DCh	HW	IRQ 15 – Secondary HD Controller

Redosled aktivnosti pri preuzimanju prekida

- Inicijalizacija
  - zabrani prekide
  - iniciraj HW uređaj (da zahteva prekid)
  - dozvoli prekid na PIC-u
  - sačuvaj stari i postavi novi vektor prekida
  - dozvoli prekide
- void interrupt NewISR
  - obrada uzroka prekida (brisanje zahteva)
  - eventualno pozovi staru rutinu (ako postoji default obrada)
  - EOI (end of interrupt) komanda PIC-u
- Izlazak iz programa
  - zabrani prekide
  - onemogući prekid na HW uređaju
  - zabrani prekid na PIC-u
  - vrati stari vektor prekida
  - dozvoli prekide

Primer: Periodični prekidi sa RTC kola

```
void InitRTC(void)
{
   disable();
   OldInt70 = getvect(0x70);                    //save old interrupt vector
   setvect( 0x70, newInt70 );                   // put new interrupt vector

   writeRTC( REG_A, DIV_BITS|RATE_SEL1 );       // set counting on 1ms
   writeRTC( REG_B, PIE|MODE_24 );

   outportb( 0xA1, inportb(0xA1) & 0xFE );
   enable();
}
void interrupt newInt70()
{
   RealTime++;
   enable();

   readRTC( REG_C );                  // Essential for recovering of RTC
   outportb( 0xA0, 0x20 );            // End of interrupt for higher 8259
   outportb( 0x20, 0x20 );            // End of interrupt for lower 8259
}
void DeinitRTC(void)
{
   disable();
   writeRTC( REG_A, DIV_BITS|RATE_SEL1 );       // return counting on 1ms
   writeRTC( REG_B, MODE_24 );
   outportb( 0xA1, ( inportb(0xA1) | 0x01) );
   setvect(0x70, OldInt70);                     // return old
   enable();
}
```

Da li je ovako čitljivije?

```
void InitRTC(void)
{
   disable();
   OldInt70 = getvect(0x70);      // save old int.vector
   setvect( 0x70, newInt70 );     // put new int.vector

   writeRTC( REG_A, DIV_BITS|RATE_SEL1 );    // 1ms
   writeRTC( REG_B, PIE|MODE_24 );

   outportb( 0xA1, inportb(0xA1) & 0xFE );
   enable();
}
```

```
void DeinitRTC(void)
{
   disable();
   writeRTC( REG_A, DIV_BITS|RATE_SEL1 );
   writeRTC( REG_B, MODE_24 );
   outportb( 0xA1, ( inportb(0xA1) | 0x01) );
   setvect(0x70, OldInt70);            //set old IV
   enable();
}
```

```
void interrupt newInt70()
{
   RealTime++;
   enable();

   readRTC( REG_C );           // obrada RTC
   outportb( 0xA0, 0x20 );     // EOI slave
   outportb( 0x20, 0x20 );     // EOI master
}
```

Asemblerski listing prekidne rutine

ulaz, obrada, izlaz

```
;void interrupt newInt70()
        assume  cs:_TEXT
_newInt70       proc            far
        push    ax
        push    bx
        push    cx
        push    dx
        push    es
        push    ds
        push    si
        push    di
        push    bp
        mov     bp,DGROUP
        mov     ds,bp
        mov     bp,sp
;{
```

```
        RealTime++;
;enable();
        ei
;readRTC( REG_C );              // Obrada RTC
        mov     al,12
        push    ax
        call    near ptr _readRTC
        pop     cx
```

```
; outportb( 0xA0, 0x20 );       // EOI slave
        mov     dx,160
        mov     al,32
        out     dx,al
; outportb( 0x20, 0x20 );       // EOI master
        mov     dx,32
        mov     al,32
        out     dx,al
;}
        pop     bp
        pop     di
        pop     si
        pop     ds
        pop     es
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        iret
_newInt70       endp
```

Primer rukovanja prekidima na MIPS platformi

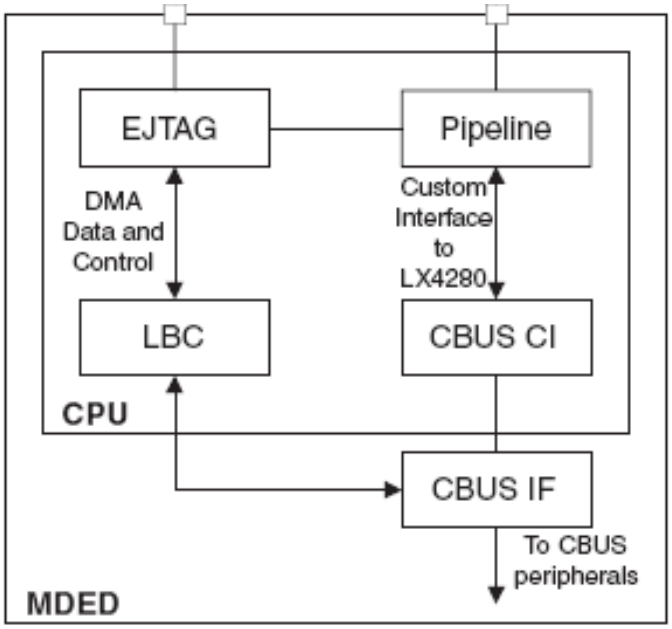
SW generisan HW

Interrupt Handler

Specifičnosti programiranja na *embedded* platformi

- Mešanje C i asemblerskog koda
- Neophodan je prisniji odnos sa HW osnovom
  - Instrukcije, adresiranja, memorijski prostor
  - Periferije – format registara i UI adrese
  - Zahtevi RISC protočne organizacije
  - Poznavanje organizacije stack-a i strukture podataka na njemu nije opšte obrazovanje, nego živa potreba
- Poznavanje nestandardnih biblioteka
- Pristup UI uređajima
- Rukovanje prekidima i DMA transferima
- Opcije kompajlera - pogotovo uticaj optimizacije
- U našem slučaju, interesantna je MIPS platforma

MIPS/MDE UI organizacija



- MIPS ne podržava izolovani UI
- Nedostaci MM pristupa peglaju se uvođenjem posebne sprežne magistrale
- MDE koristi CBUS (control?)
  - povezuje periferije
- Pristup MDE portovima
  - UINT32 cbus_read(address);
  - void cbus_write(address, value);

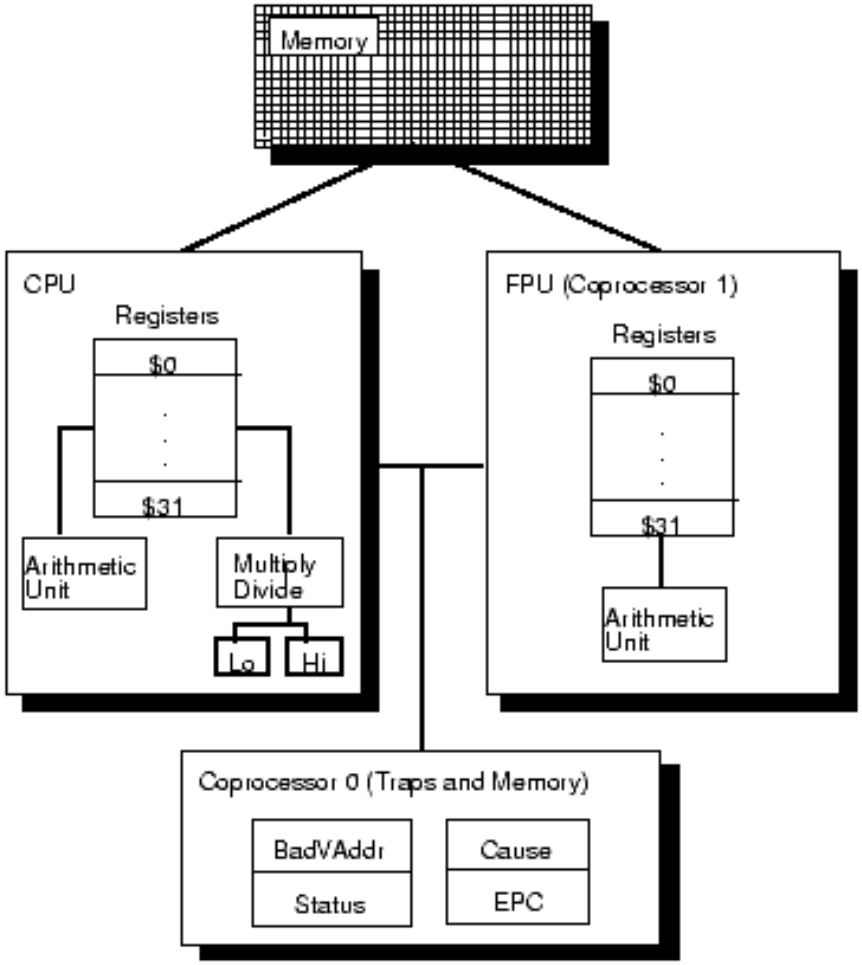
Protočna obrada i UI aktivnosti

- Pipeline organizacija relativizuje
  - pojam programskog brojača, tačnije
  - instrukcije koja se trenutno izvršava
- Pojava prekida zahteva zaustavljanje protočne obrade
  - slično grananju
  - korišćenje postojeće logike za pražnjenje “pipeline”-a
  - narušava performanse sistema
- DMA je dodatan problem
  - rizik podataka preko operativne memorije

MIPS sistem prekida

- Svi prekidi koriste isti vektor i nivo prioriteta
  - CPU prelazi u privilegovani režim
  - izvršenje se prenosi na adresu 0x80000180
  - pristup registrima koprocesora 0 radi analize uzroka prekida
- Zašto?
  - nedostatak uP-UJ se nadoknađuje brzinom RISC procesora
  - kašnjenje zbog programskog prihvata prekida je najčešće zanemarivo u odnosu na akciju obrade
- Podržani su regularni MIPS prekidi (exceptions)
  - Spoljni, interni, programski

Organizacija MIPS procesora

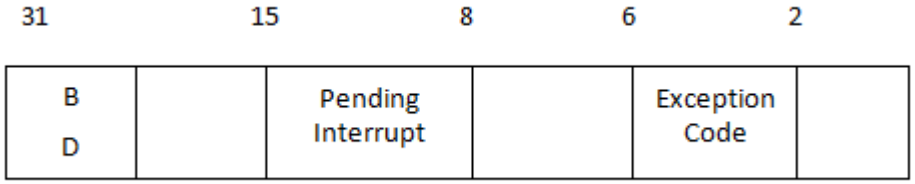


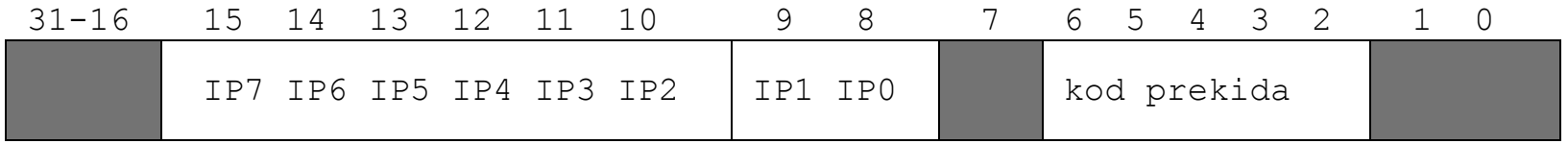
Registri koprocesora 0

Broj	Naziv	Opis
8	BadVAddr	Loša virtualna adresa (mem. adresa)
12	Status	Maska i status prekida
13	Cause	Vrsta prekida
14	EPC	Adresa instrukcije koja je izazvala prekid

- Navedeni registri se koriste za obradu prekida
- Njima se pristupa pomoću mfc0 i mtc0 instrukcija
- Move from/ to coprocessor 0, npr
  - mfc0 $k0, $13 #Cause
  - mfc0 $k1, $14 #EPC
- Registri k0 i k1 rezervisani su u ove svrhe

Cause registar



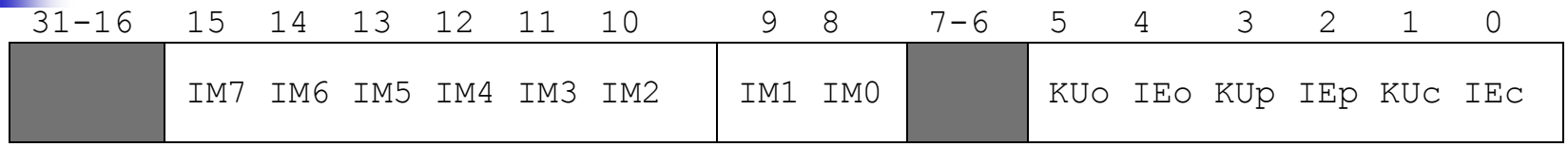


- **Cause registar** sadrži informacije o prekidima koji su se dogodili i čekaju na obradu (*interrupt pending*)
- Prekid na i-tom nivou, označava se bit IPi bitom
- Biti IP1 i IP0 se koriste za programske prekide
- Polje **kod_prekida** sadrži uzrok prekida

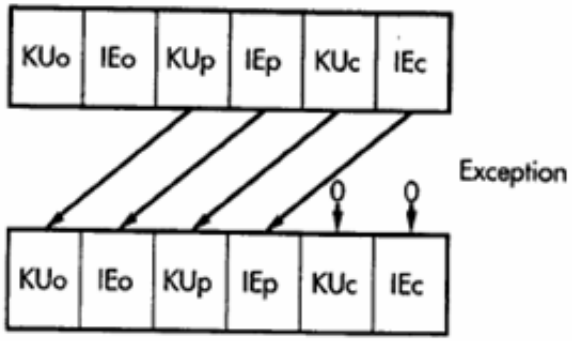
Kod prekida

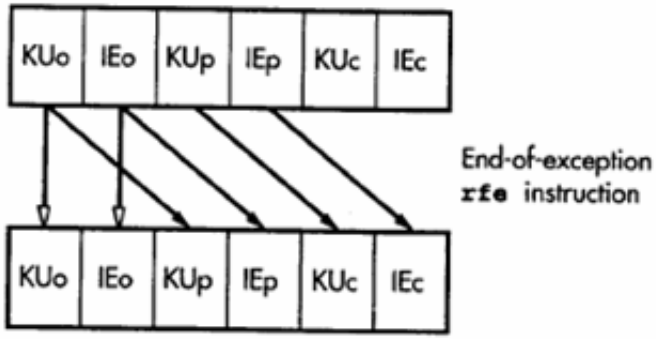
Kod	Naziv	Opis
0	INT	Spoljni prekid označen IPi bitima
4	ADDRL	Čitanje sa ilegalne adrese
5	ADDRS	Upis na ilegalnu adresu
6	IBUS	Greška na magistrali u toku zahvatanja instrukcije
7	DBUS	Greška na magistrali u toku pristupa podatcima
8	SYSCALL	Izvršenje syscall instrukcije
9	BKPT	Izvršenje break instrukcije
10	RI	Pokušaj izvršenja nepoznate instrukcije
12	OVF	Aritmetičko prekoreačenje

Status registar



- Biti IMi maskiraju (dozvola/zabrana) pojedinačne prekide
- KU/IE kontrolišu režim rada i prihvat prekida (globalna dozvola/zabrana)
- KUc/IEc ukazuju na trenutan (current) režim rada
- Pri prekidu oni se kopiraju u par KUp,IEp (previous), a trenutna vrednost ovih bita se pomera u KUo,IEo (old) par. Staro stanje se gubi.
- Ovo je neophodno jer se po povratku iz prekidne rutine CPU mora naći u istom režimu u kome je bio kada se prekid dogodio





EPC (*Exception Program Counter*) registar

- Čuva adresu naredne instrukcije, tj. adresu povratka
- MIPS arhitektura pravi razliku između spoljnog i programski generisanog prekida
  - U slučaju programskog prekida EPC sadrži adresu instrukcije koja je generisala prekid, a ne one iza nje
  - Zato povratna adresa mora biti uvećana za 4
- Priloženi kod ilistruje sekvencu pri povratku iz oba prekida

```
# spoljni prekid
mfc0 $k0,$14   #učitaj EPC
rfe            #ret iz prekida
jr $k0
nop
```

```
# programski prekid
mfc0 $k0,$14   #učitaj EPC
addiu $k0, 4   #uvecaj za 4
rfe            #ret iz prekida
jr $k0
nop
```

Restore From Exception - **rfe**

- Restauriše stanje pre prekida
  - vraćanje korisničkog režima, i sl.
  - podešavanjem Status registra
- NE vraća kontrolu na prekinuti program
- Povratak se radi **jr** instrukcijom u “delay slot”-u

Jednostavna prekidna rutina

```
    .section ".xcpn_text"  #0x80000080
    .set noreorder
    .set noat
xcptgen:
    la   ko,xcnt          # uzmi adresu brojača
    lw   k1,0(k0)         # napuni ga
    nop
    addu k1,1             # uvecaj
    sw   k1,0(k0)         # i sacuvaj
    mfc0 k0,$14           # učitaj EPC
    nop
    jr   k0               # vrati se
    rfe
    .set at
    .set reorder
```

void __attribute__((interrupt(),nomips16)) Timer1Handler(void)

```
{
  RDPGPR SP, SP
  MFC0 K1, EPC
  MFC0 K0, SRSCtl
  ADDIU SP, SP, -24
  SW K1, 20(SP)
  MFC0 K1, Status
  SW K0, 16(SP)
  MFC0 K0, Cause
  SW K1, 12(SP)
  SRL K0, K0, 10
  INS K1, K0, 10, 6
  INS K1, ZERO, 1, 4
  MTC0 K1, Status
  SW V1, 0(SP)
  SW V0, -4(SP)
  LW V1, 16(SP)
  ANDI V1, V1, 15
  SW S8, 4(SP)
  NOP
  ADDU S8, SP, ZERO
```

```
digitalWrite(pin, !digitalRead(pin));
// clear the interrupt flag
clearIntFlag(_TIMER_1_IRQ);
```

```
ADDU SP, S8, ZERO
  LW V0, 16(SP)
  ANDI V0, V0, 15
  LW S8, 4(SP)
  LW V1, 0(SP)
  LW V0, -4(SP)
  NOP
  DI ZERO
  EHB
  LW K0, 20(SP)
  LW K1, 12(SP)
  MTC0 K0, EPC
  LW K0, 16(SP)
  ADDIU SP, SP, 24
  MTC0 K0, SRSCtl
  WRPGPR SP, SP
  MTC0 K1, Status
  ERET   //Exception Return
}
```

void __attribute__((interrupt(),nomips16)) Timer1Handler(void)

`RDPGPR SP, SP`	Read GPR from Previous Shadow Set
`MFC0 K1, EPC`	Učitaj EPC
`MFC0 K0, SRSCtl`	Kao i Shadow register control
`ADDIU SP, SP, -24`	Pripremi stack
`SW K1, 20(SP)`	Sačuvaj K1 (EPC)
`MFC0 K1, Status`	Učitaj Status registar
`SW K0, 16(SP)`	Sačuvaj K0 (SRSCtl)
`MFC0 K0, Cause`	Učitaj Cause registar
`SW K1, 12(SP)`	Sačuvaj K1 (Status)
`SRL K0, K0, 10`	Shift Word Right Logical
`INS K1, K0, 10, 6`	Insert Bit Field
`INS K1, ZERO, 1, 4`	
`MTC0 K1, Status`	Upisi u Status
`SW V1, 0(SP)`	Push V1
`SW V0, -4(SP)`	Push V0
`LW V1, 16(SP)`	Napuni ucitani SRSCtl u V1
`ANDI V1, V1, 15`	ANDuj sa 0x0F
`SW S8, 4(SP)`	Sacuvaj S8
`NOP`	
`ADDU S8, SP, ZERO`	

Rukovanje prioritetima prekida

- Isključivo programski, korišćenjem IMi bita
  - Pojedinačnim maskiranjem prekida
- Osnovna ideja
  - Aplikacija podržava tekući nivo prioriteta, 0 – 6 npr.
  - Svaki od prekida vezan je za jedan od njih
  - Na najnižem nivou dozvoljeni su svi prekidi
  - Na najvišem nivou zabranjeni su svi prekidi
- Mogu se ostvariti politike fiksne ili promenljive dodele
- PROBLEM: rukovanje SR registrom
  - zahteva učitavanje i potom zapis

Ugnježdeni (nested) prekidi

- Ukoliko se dozvoli paralelno izvršenje prekida
- Pre dozvole, neophodno je
  - čuvanje EPC i SR u memoriji
  - kao i k0 i k1 registara
- Za njihovo čuvanje logično je koristiti stack
  - exception frame
  - može doći do prekoračenja opsega
- Obično se dozvoljavaju samo prekidi prioritetniji od tekućeg

Rukovanje SR registrom

```
   mfc0 t0,SR
1:
   nop
   or/and t0, maska    #set/reset bit
2:
   mtc0 t0,SR
   nop
   nop
```

- Ova sekvenca je prekidiva (nije atomska)
- Problem ukoliko se radi iz aplikacije ili sa ugnježdenim prekidima

Test and Set opcija

- Atomski zapis u memorijsku reč
  - prvo test na sadržaj, potom zapis nove vrednosti
- Podržana posebnom instrukcijom na većini arhitektura
- Osnova za pravljenje kritične sekcije

```
int TestAndSet(int* lockPtr)
{
    int oldValue;
    oldValue = SwapAtomic(lockPtr, 1);
    return oldValue != 0;
}
```

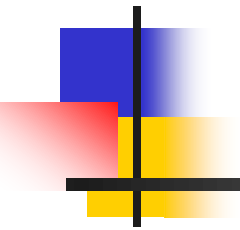
INTEL arhitektura

- BTx BitBase, BitOffset
- BT, BTS, BTC, BTR
  - bit test, test & set, test & compl, test & reset
- LOCK
  - prefiks instrukcije - atomsko izvršenje
- stara vrednost se sačuva u C bitu status registra
  - CF ← Bit(BitBase, BitOffset)
  - Bit(BitBase, BitOffset) ← nova vrednost;

MIPS kritična sekcija sa instrukcijama ll i sc

```
WaitSem:
   la   t0,sem
TryAgain:
   ll   t1,0(t0)
   bne  t1, zero, WaitSem
   li   t1, 1
   sc   t1,0(t0)
   beq  t1, zero, TryAgain
   # dobio semafor
   jr ra
```

- Ne garantuje neprekinuto izvršenje, ali daje status izvršenja
- *Load Linked*
  - puni registar, ali
  - pamti ll zahtev (flag + LLAddr)
- *Store Conditional*
  - prvo proveri da li je prethodno započeta sekvenca još neprekinuta
  - Ako jeste, zapisuje i vraća TRUE
  - Ako nije, vraća FALSE



Rukovanje prekidima u RTOS

Prekidi i Izuzeci

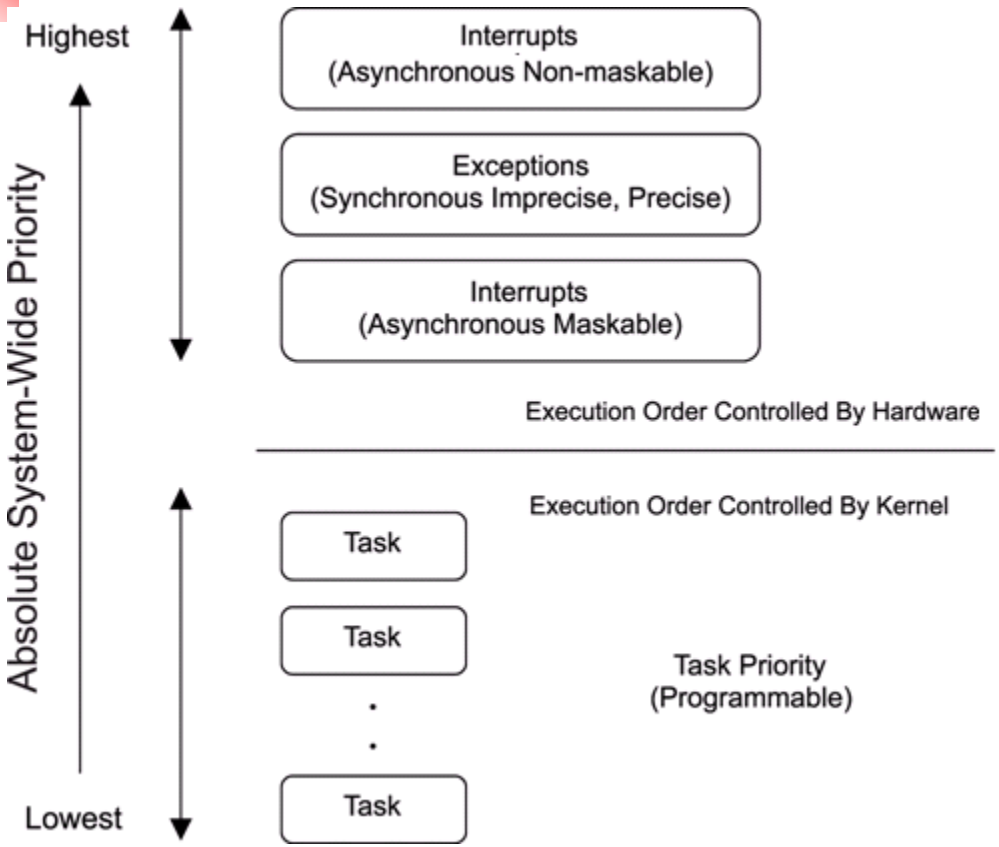
Interrupts & Exceptions

Asihnroni i Sinhroni izuzeci

Značaj prekida i načina njihove obrade

- RT aplikacije su generalno orijentisane na korišćenje prekida
- Efikasnost obrade prekida je jedna od osnovnih ocena kvaliteta RTOS
- Klasifikacija prekida
  - spoljni (*interrupts, asynchronous exceptions*) – UI prenos, HW
  - interni (*exceptions , synchronous exceptions*) – pristup memoriji, delenje nulom, SW
  - precizni – stanje procesora definisano (PC je iza izvršene instrukcije a nema započetih/nezavršenih)
  - neprecizni – svi ostali slučajevi tipični za RISC/pipeline
- RTOS najčešće direktno rukuju prekidima (ISR), a korisničkoj aplikaciji šalju signal koji pokreće nit (IST)
  - obrada prekida u dva konteksta

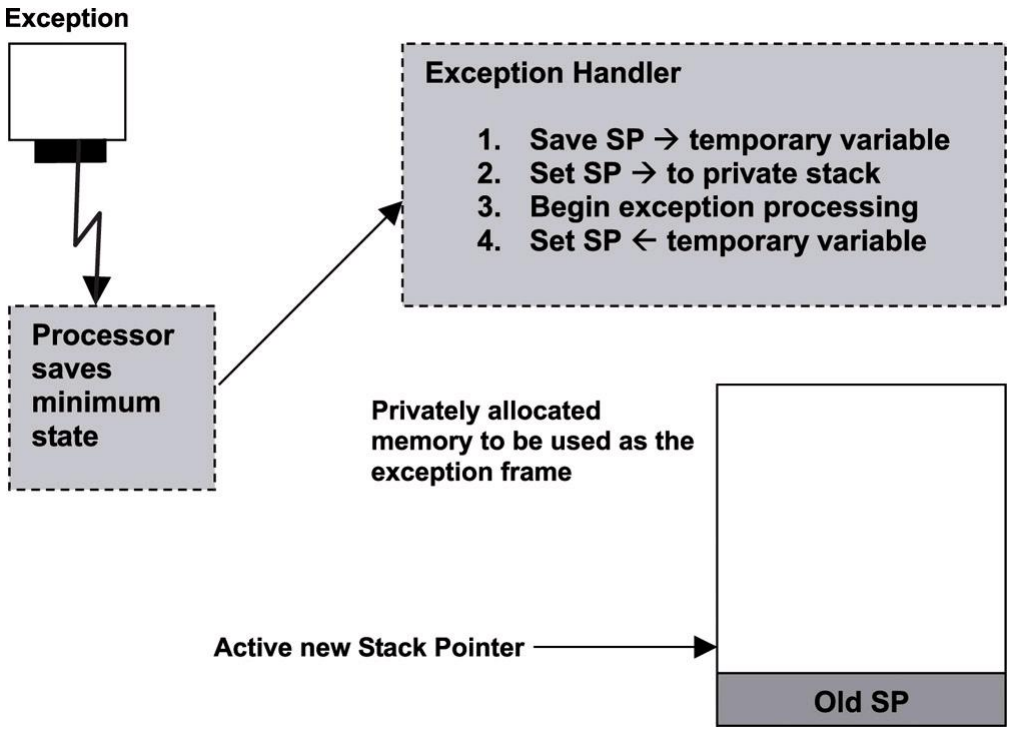
Prioritet izvršenja prekida



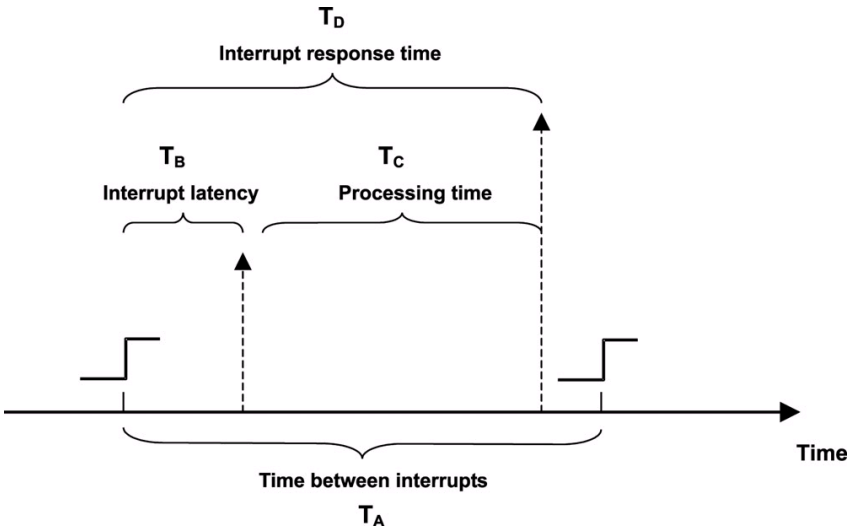
Highest	Interrupts	Non-maskable
↓	Exceptions	Precise
	Exceptions	Imprecise
Lowest	Interrupts	Maskable

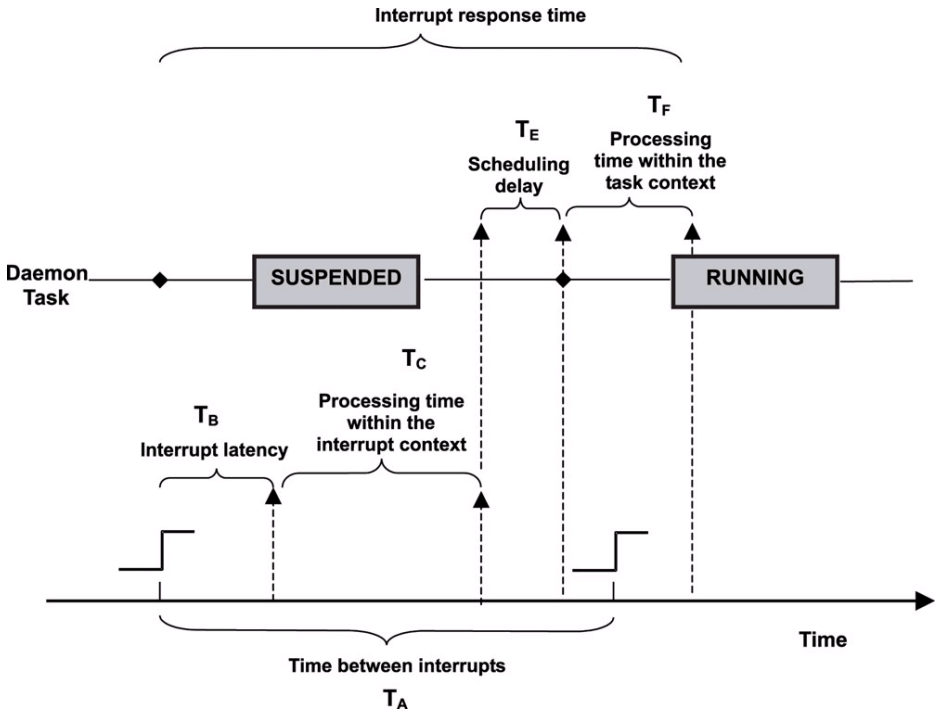
Sistemska ISR (*Interrupt Service Routine*)

- Rukuje svim prekidima u sistemu
  - default procedure obrade (reset prekida)
  - slanje signala ka registrovanim IST (*Interrupt Service Thread*)
  - sopstveni stack (*interrupt stack , exception frame*)



Vremensko izvršenje obrade prekida



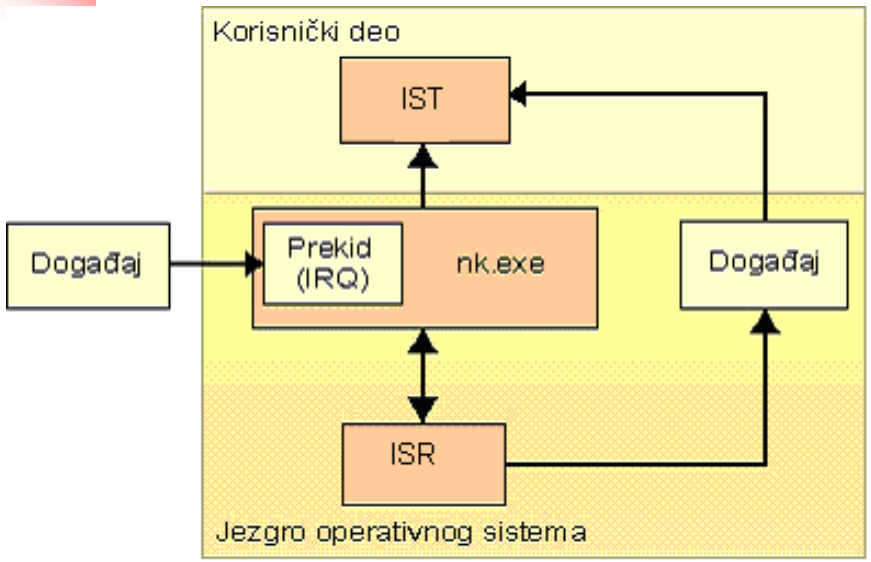


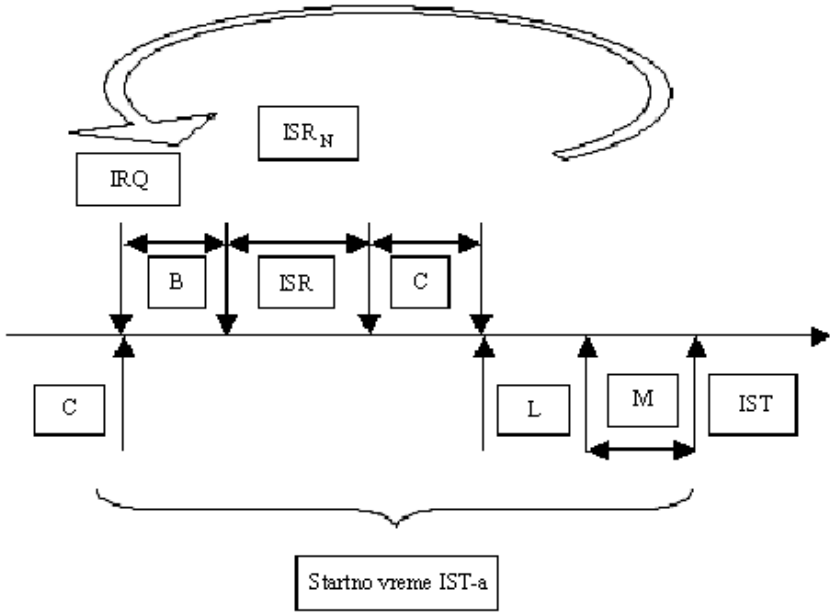
- ISR ima za cilj što brži reset prekida i zaštitu sistema
- IST radi aplikativnu obradu, bez restrikcija sistemskih poziva i sl.
  - produžuje se ukupno vreme obrade
  - mogućnost obrade po prioritetu prekida, preko prioriteta niti IST

Primer formiranja IST u Linuxu

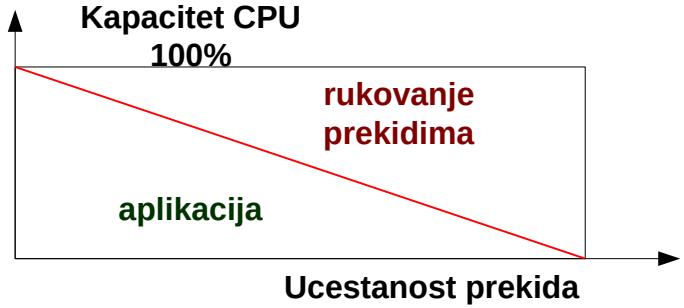
```
#include <linux/sched.h>;
....
int request_irq(unsigned int irq,
   void (*handler)(int, void *, struct pt_regs *),
   unsigned long flags,
   const char *dev_name,
   void *dev_id);

void free_irq(unsigned int irq, void *dev_id);
....

if (short_irq >= 0)
{
    result = request_irq(short_irq, short_interrupt, SA_INTERRUPT,
                         "short", NULL);
    if (result)
    {
        printk(KERN_INFO "short: can't get assigned irq %i\n", short_irq);
        short_irq = -1;
    }
    else
    { /* actually enable it -- assume this *is* a parallel port */
        outb(0x10,short_base+2);
    }
}
```

Rukovanje prekidima u Windows CE



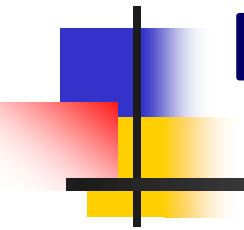


$$Start\ IST\text{-}a = C + L + M + \sum_{n=0}^{N_{ISR}} (B + C + T_{ISR_N})$$



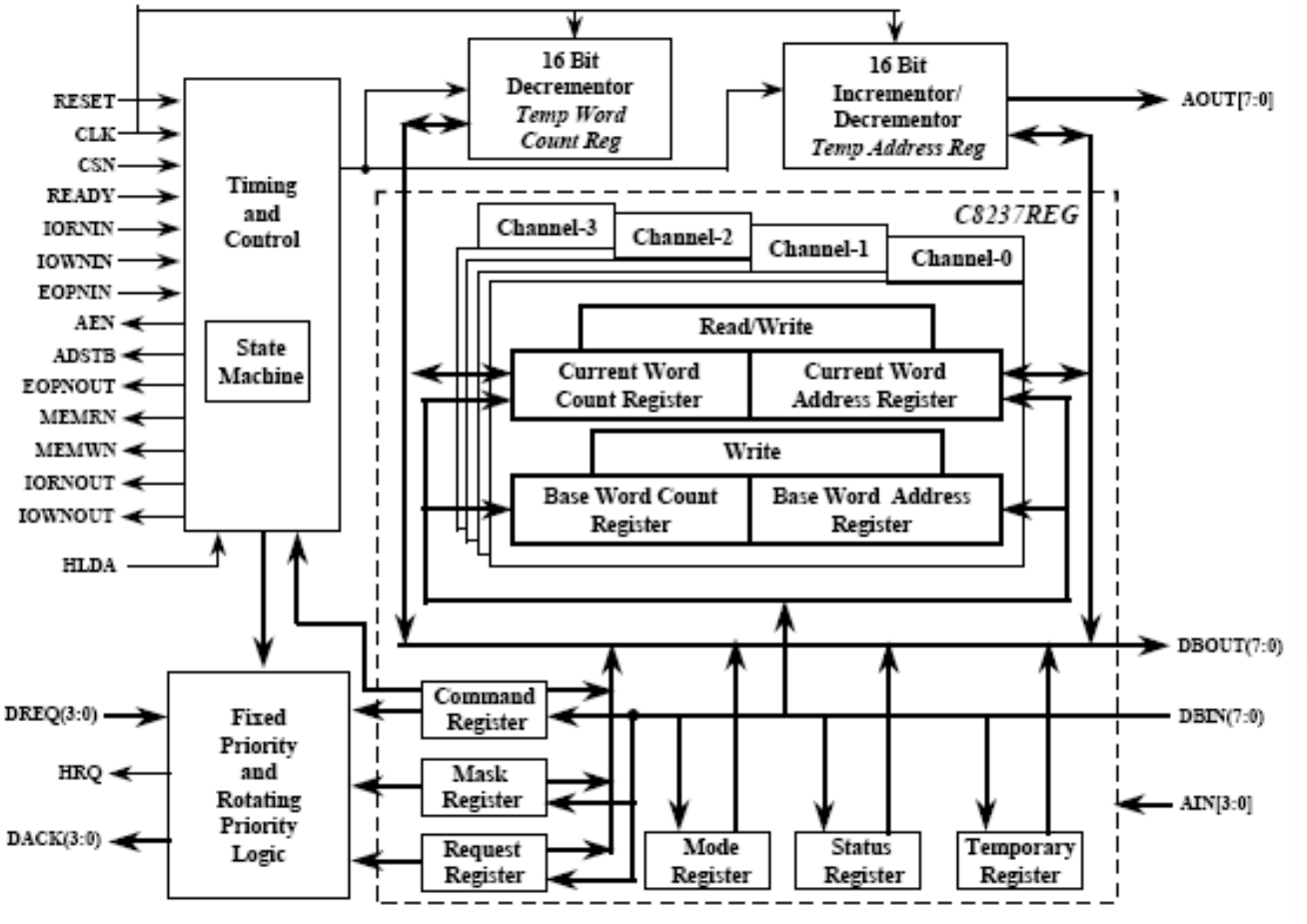
Inicijalizacija IST

```
HANDLE hEvent;

DWORD WINAPI IRQ_Thread( void *pValue )      // IST thread
{
   while( TRUE )
   {
      WaitForSingleObject( hEvent, INFINITE );
      InterruptDone( sysintr );
   }
}

int SetIST( int IRQn )     // inicijalizacija IST
{
   hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

   DWORD sysintr = MapIrq2SysIntr( IRQn );
   if( !InterruptInitialize(sysintr, hEvent, NULL, 0) )
      return -1;

   HANDLE hThread = CreateThread(NULL,0,IRQ_Thread,NULL,0,&ThreadID);
   SetThreadPriority( hThread, THREAD_PRIORITY_TIME_CRITICAL );

   return 0;
}
```

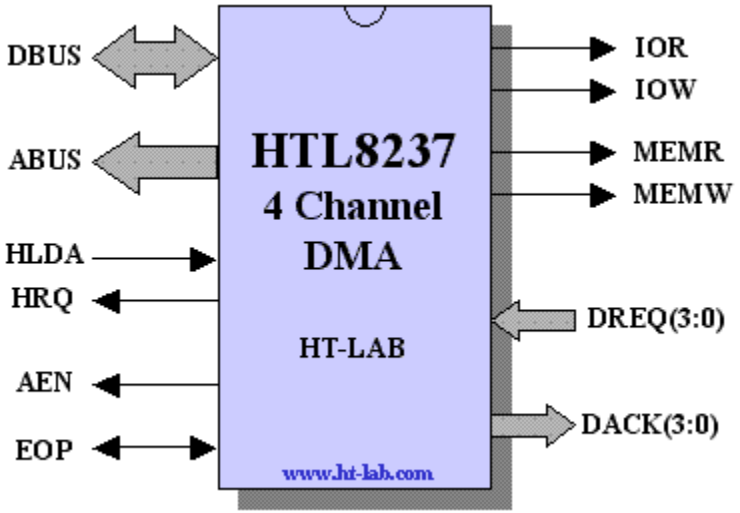


DMA primer

8237 DMA Controller (4-kanalni)



Funkcionalnost 8247 DMA kontrolera

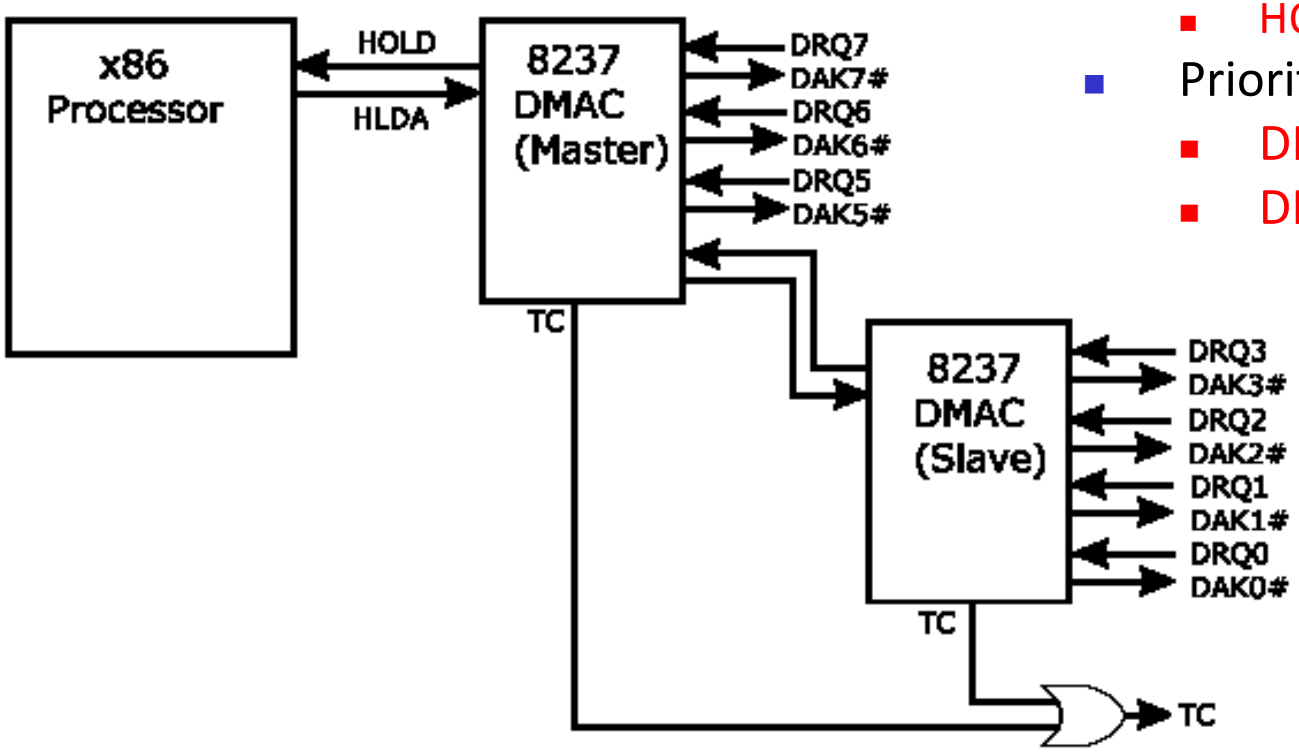


- 4 DMA channels, each with an address range of 64K bytes/words.
- An address can be auto incremented, decremented or put on hold.
- 4 modes: single, block, demand and cascade
- Single mode - one transfer before returning the bus to the processor
- Block mode - transfers a programmed number of words without returning the bus.
- Demand mode - peripheral continues to transfer data until exhausted or when the preprogrammed number of transfers has been reached or when EOP signal is asserted.
- Cascade mode which is used to extend the number of channels by cascading additional 8237 controller(s).

Režimi rada

- **Demand Transfer Mode**: The device will continue making transfers until a TC or external EOPN is encountered or until DREQ goes inactive.
- **Single Transfer Mode**: The device makes one transfer only. DREQ must be held active until DACK becomes active in order to be recognized.
- **Block Transfer Mode**: The device is active by DREQ or software request to continue making transfers during the service until a TC or an external EOPN is encountered. DREQ need only be held active until DACK becomes active.
- **Cascade Transfer Mode**: This mode is used to cascade more than one C8237 together for simple system expansion. The ready input is ignored in this cascade transfer mode.

DMA podsistem PC računara



- CPU sprega:
  - HOLD
  - HOLD Acknowledge
- Prioriteti:
  - DRQ0 najviši
  - DRQ7 najniži

Mapiranje UI adresa DMA kontrolera

DMA Controller	Slave				Master			
DRQ#	0	1	2	3	4	5	6	7
Memory Address Register I/O Address	00h	02h	04h	06h	C0h	C4h	C8h	CCh
Page Register	87h	83h	81h	82h	(none)	8Bh	89h	8Ah
Count Register I/O Address	1	3	5	7	(none)	C6h	CAh	CEh

DMA Controller	Slave	Master
Control Register I/O Address	08h	D0h
Mode Register I/O Address	0Bh	D6h
Mask Register I/O Address	0Ah	D4h
Clear Byte F/F I/O Address	0Ch	D8h

Podešavanje kanala 1 DMA kontrolera 8237 za prijem

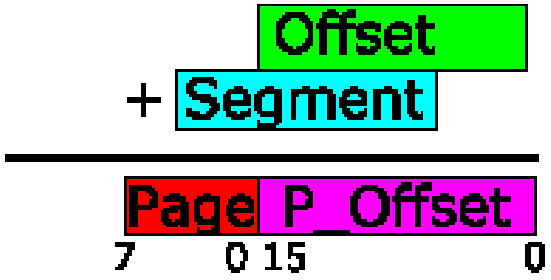
```
void SetRxDma( BYTE *SrcPnt )
{
   unsigned long RxAdr;
   unsigned char PgRx;

   RxAdr = ((FP_SEG(SrcPnt)<<4) & 0x000FFFF0L) + FP_OFF(SrcPnt);
   PgRx  = (RxAdr >> 16) & 0x0000000FL;

   outportb( Mode_Reg_37, Single_Mode | Write_Xfer | Ch1_Sel);
   outportb( Clr_Byte_Ptr_37,                              0);
   outportb( Ch1_Addr,                                RxAdr);
   outportb( Ch1_Addr,                          (RxAdr >> 8));
   outportb( Ch1_Page_Reg,                             PgRx);
   outportb( Ch1_Count,                             BUFA_SIZE);
   outportb( Ch1_Count,                                    0);
   outportb(Single_Mask,                             Ch1_Sel);
}
```

Podešavanje kanala 3 DMA kontrolera 8237 za slanje

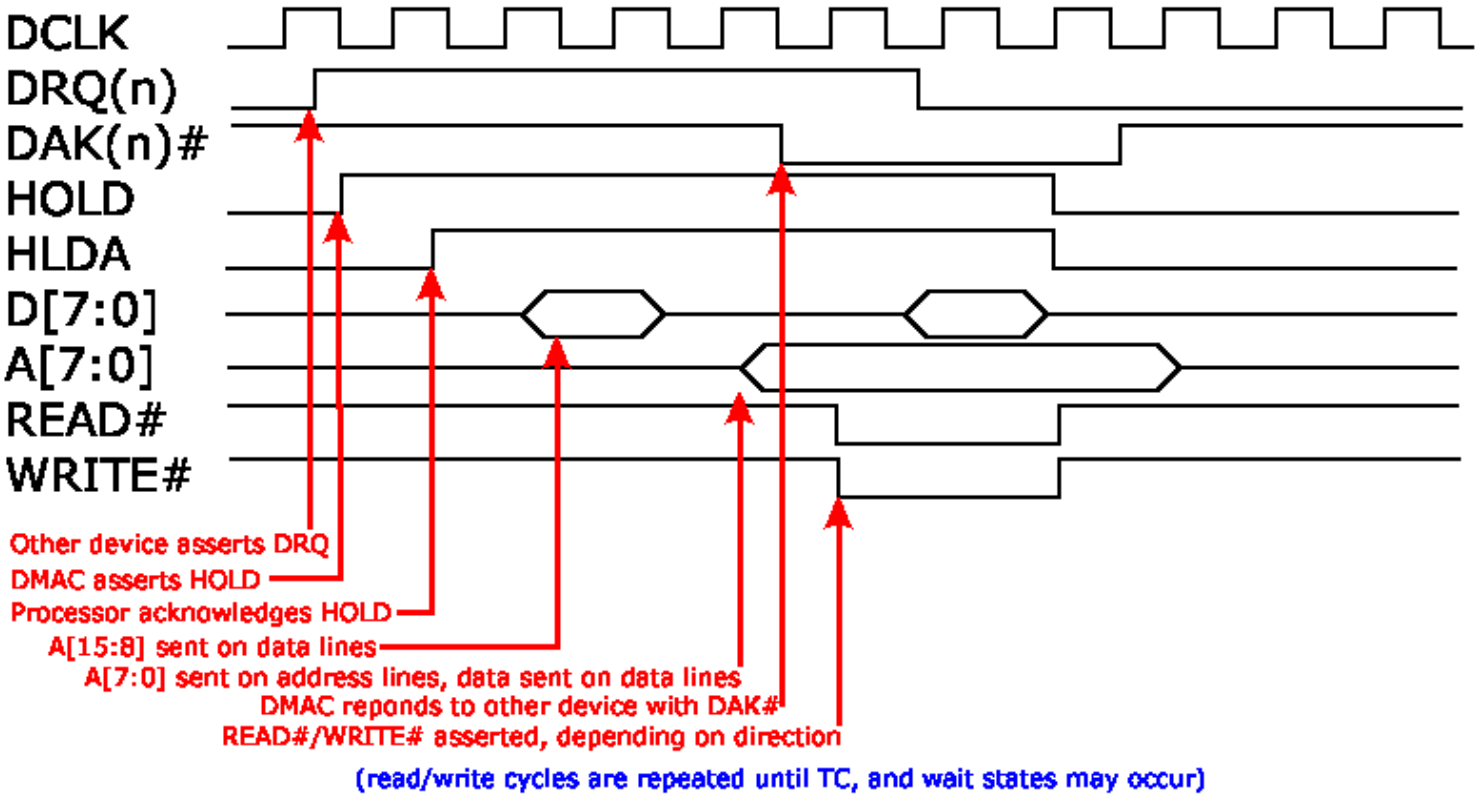
```
void SetTxDma( void )
{
   unsigned long TxAdr;
   unsigned char PgTx;
   unsigned char *P = &TporA[1];

   TxAdr = ((FP_SEG(P) << 4 ) & 0x000FFFF0L) + FP_OFF(P);
   PgTx  = (TxAdr >> 16) & 0x0000000FL;

   outportb( Mode_Reg_37, Single_Mode | Read_Xfer | Ch3_Sel);
   outportb( Clr_Byte_Ptr_37,                          0);
   outportb( Ch3_Addr,                              TxAdr);
   outportb( Ch3_Addr,                       (TxAdr >> 8));
   outportb( Ch3_Page_Reg,                           PgTx);
   outportb( Ch3_Count,                        TxDuzA - 2);
   outportb( Ch3_Count,                                0);
   outportb( Single_Mask,                         Ch3_Sel);
}
```

Izračunavanje efektivne adrese x86 procesora

- Par *Segment:Offset* definiše 20-bitnu adresu
- *Page* registar dopunjuje sa 4 bita donji deo adrese sačuvan u 16-bitnom registru (*Memory Address Register*)



DMA mašinski cilkus na ISA magistrali



Hvala na pažnji