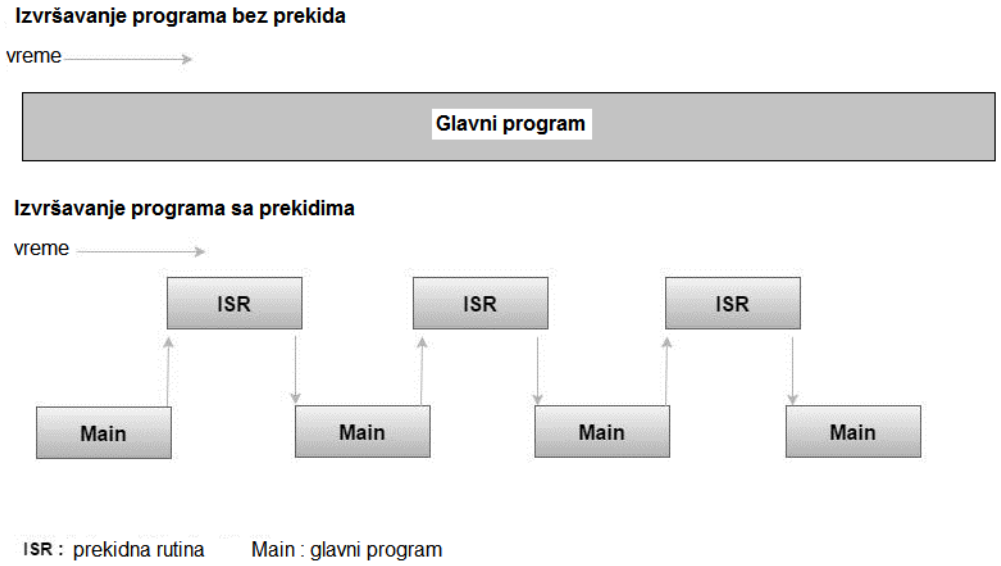
Mikroprocesorska elektronika

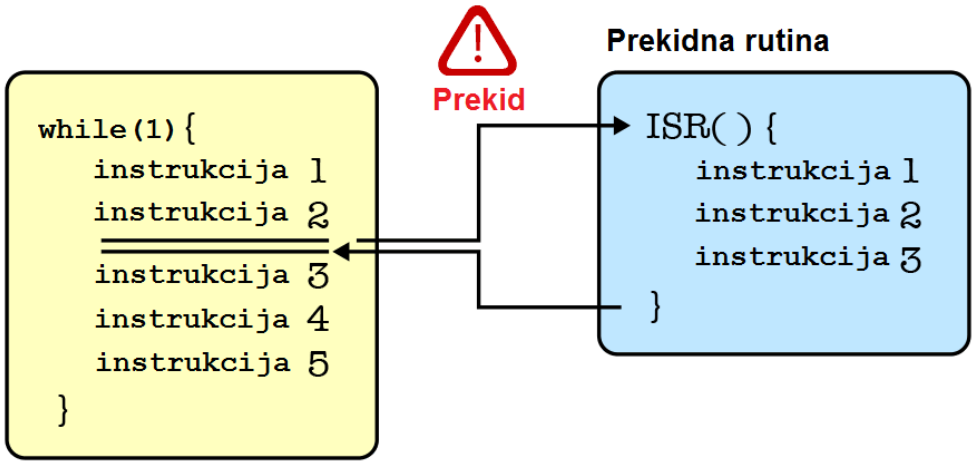
Prekidi

9 Prekidi

Generalno govoreći, prekid predstavlja događaj koji prethodni tok procesa prekida da bi se dešavalo nešto drugo (npr. telefonski poziv koji prekida prethodnu aktivnost). U embeded sistemima prekid predstavlja događaj koji je hardverski iniciran, koji zaustavlja procesor i potom program nastavlja rad na nekom drugom mestu. Nastavak programa na drugom mestu tiče se procesa koji nazivamo obrada prekida. Pri tome, procesor mora završiti barem tekuću instrukciju, obično automatski zabranjuje nove prekide (ili barem one koji su nižeg nivoa prioriteta), postavlja na stek (ako nije drugačije rešeno) adresu naredne instrukcije od koje će nastaviti izvršavanje programa nakon obrade prekida (tzv. povratnu adresu) i postavlja u programski brojač adresu početka programa za obradu prekida. Na kraju programa za obradu prekida uzima povratnu adresu i nastavlja redovno izvršavanje programa. Izvršavanje programa bez prekida i sa obradom prekida je ilustrovano na slici 9.1, a na slici 9.2 je ilustrovan rad prekidne rutine.



Slika 9.1 Ilustracija rada programa bez obrade i sa obradom prekida



Slika 9.2 Ilustracija rada programa i prekidne rutine

9.1 OBRADA PREKIDA

Mikrokontroleri u opštem slučaju mogu da rade sa više različitih periferija koje zahtevaju odgovarajuću obradu. U slučajevima kada periferija zahteva obradu od mikrokontrolera, postoje dva načina koja se koriste u obradi zahteva:

1) metod prozivke (eng. *polling*)
2) putem obrade zahteva za prekidom (eng. *interrupt request service*)

Kod metode prozivke mikrokontroler kontinuirano proziva periferiju (ili više njih) da proveri da li je potrebna obrada. Kada dobije informaciju da je potrebna obrada on je aktivira. Uobičajeno je da se ovo radi u petlji pa je zato mikrokontroler kontinuirano zauzet i nije u mogućnosti da radi nešto drugo ili da bude u režimu štednje energije. Ovo je glavna mana ovog metoda. Primer neefikasnosti ove metode je u slučaju telefonskog operatera koji treba da beleži poruke koje dobija telefonom, a pri tome telefon nema nikakvu indikaciju dolaznog poziva već operater mora stalno da podiže slušalicu i proverava da li je neko pozvao.

Kod metode koja koristi sistem prekida, periferija obaveštava mikrokontroler da je potrebna obrada. Dotle i nakon toga mikrokontroler može da radi druge stvari ili da bude u režimu štednje energije. Na ovaj način je mnogo lakše obraditi zahteve u slučajevima kada postoji više od jedne periferije. Ovaj metod je daleko efikasniji od metoda prozivke.

Pored ove prednosti, projektovanje embeded sistema baziranih na sistemu prekida ima i čitav niz dodatnih prednosti:

- Kompaktan i modularan softver – prekidni programi (eng. *Interrupt Service Routine*, ISR) nameću korišćenje modularizacije programa i njegovo ponovno korišćenje
- Smanjena potrošnja – kako korišćenje ISR rezultuje u izvršavanju manjeg broja CPU ciklusa, ovo ima direktan uticaj na količinu električne energije potrošenu od strane aplikacije
- Brže vreme odziva – Kada u sistemu postoji veći broj različitih periferijskih jedinica, odnosno spoljašnjih događaja, koje je potrebno servisirati, pažljivo projektovane ISR rutine obezbeđuju brzi odziv na zahteve za obradom. Pametno korišćenje sistema prioriteta prekida obezbeđuje brzo vreme odziva.

Primeri:

-sistem sa LED diodom i tasterom gde se na svaki pritisak tastera dioda uključuje/isključuje. Uključenje/isključenje se vrši u prekidnoj rutini koju aktivira pritisak tastera.

-dolazna poruka koja stiže putem serijskog porta aktivira prekid i u prekidnoj rutini se ona negde smešta ili se preduzima neka akcija.

-monitoring napona baterije gde se prekid generiše kada se dostigne napon koji je ispod nekog definisanog praga.

Da bi mogao da koristi sistem prekida, projektant embeded sistema mora da uključi odgovarajuće hardverske i softverske komponente. U zavisnosti od stepena integracije procesora na kome se bazira embeded sistem, hardverske komponente za podršku radu sa prekidima mogu se nalaziti:

- odvojene od CPU jedinice na posebnom integrisanom kolu, kao što je obično slučaj sa mikroprocesorima, ili

- zajedno sa CPU jedinicom, integrisane na istom čipu, kao što je obično slučaj kod mikrokontrolera.

Nezavisno od načina realizacije, kada postoji hardverska podrška radu sa sistemom prekida, potrebno je obezbediti korektno konfigurisanje hardvera i razviti odgovarajuće softverske module da bi se kompletirala implementacija sistema prekida u nekom embeded sistemu.

Zahtevi za prekidom najčešće su inicirani nekim hardverskim događajem. Događaji kao što su pritisak tastera, dostizanje neke granične vrednosti, isticanje nekog vremenskog intervala ili kraj A/D konverzije su neki od primera događaja koji mogu da iniciraju pojavu zahteva za prekidom. Kada se jednom neki hardverski događaj konfiguriše da generiše zahteve za prekidom, njihovo pojavljivanje je potpuno nepredvidivo i **asinhrono**. Upravo iz ovog razloga softverska komponenta koja pruža podršku radu sa prekidima mora biti napisana imajući ovu činjenicu na umu.

Zahtev za prekidom uvek se sistemu za obradu prekida saopštava korišćenjem jednog električnog signala. Generalno, postoje dva pristupa za indikaciju postojanja zahteva za prekidom unutar embeded sistema:

- indikacija bazirana na korišćenju nivoa signala (eng. *level-sensitive*) i
- indikacija bazirana na korišćenju ivice signala (eng. *edge-sensitive*)

Po pravilu, ovo je moguće konfigurisati softverski.

Tipovi prekida

Postoje **maskirani** i **nemaskirani** prekidi. Maskirane prekide je moguće zabraniti dok su nemaskirani prekidi rezervisani za sistemske aktivnosti i nije ih moguće zabraniti.

Većina zahteva za prekidom u sistemu spadaju u klasu maskirajućih zahteva i prosto se nazivaju prekidima. U slučaju maskirajućih zahteva za prekidom postoji odgovarajući mehanizam pomoću kojega programer može da zabrani (maskira) odgovarajući zahtev za prekidom. Najčešće se to postiže postavljanjem ili brisanjem odgovarajućih bitova koji se nalaze unutar statusnog registra (npr. *Global Interrupt Enable,* GIE) ili nekog posebnog registra namenjenog za tu svrhu (npr. *Interrupt Enable* registar).

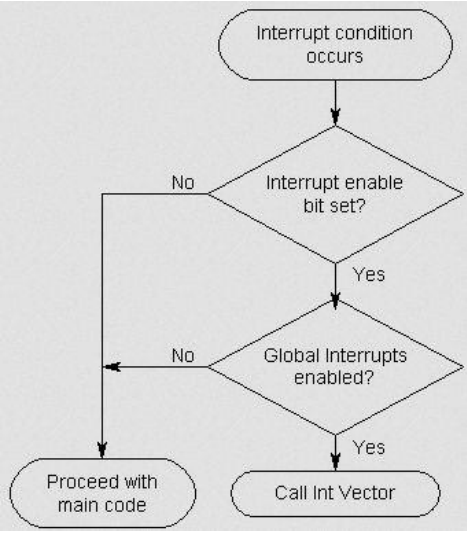
Kod ATmega328 se statusni registar zove SREG i njegov MSB bit (u oznaci I) predstavlja bit globalne dozvole/zabrane prekida.

Nemaskirajući zahtevi za prekidom ne mogu se zabraniti i kada se pojave procesor će pristupiti procesu obrade prekida. Ovaj tip zahteva za prekidom tipično je rezervisan za kritične događaje u embeded sistemu, čija se obrada ne može odložiti. Primer kritičnog događaja u sistemu mogla bi biti indikacija niskog nivoa baterije u nekom prenosivom uređaju. Nakon što se detektuje nizak nivo baterije pomoću naponskog komparatora, generisao bi se nemaskirajući zahtev za prekidom. Procesor bi započeo obradu ovog zahteva za prekidom i pristupio bi snimanju trenutnog stanja CPU-a kao i svih ostalih kritičnih podataka smeštenih u registrima i memoriji sa gubitkom sadržaja kako bi se sprečio gubitak podataka, i zatim bi započeo postupak isključenja sistema.

Druga podela prekida je bazirana na lokaciji izvora prekida u odnosu na CPU, pa tako imamo prekide uzrokovane **spoljašnjim** ili **unutrašnjim** događajima. Primeri spoljašnjih događaja su UART poslao ili primio karakter, promena signala na nekom pinu itd. Primeri prekida na bazi unutrašnjih događaja su izuzeci pri aritmetičkim operacijama (npr. deljenje sa nulom), prekidi tajmera, fluktuacija napajanja itd.

Sekvenca obrade zahteva za prekidom

Način reagovanja i algoritam obrade prekida prikazan je na slici 9.3. Dakle nakon pojave zahteva za prekidom imamo dva nivoa provere. Prvo se proverava da li je konkretan prekid dozvoljen setovanjem bita koji dozvoljava taj prekid. Ako je dozvoljen tada se još proverava i da li su prekidi globalno dozvoljeni.

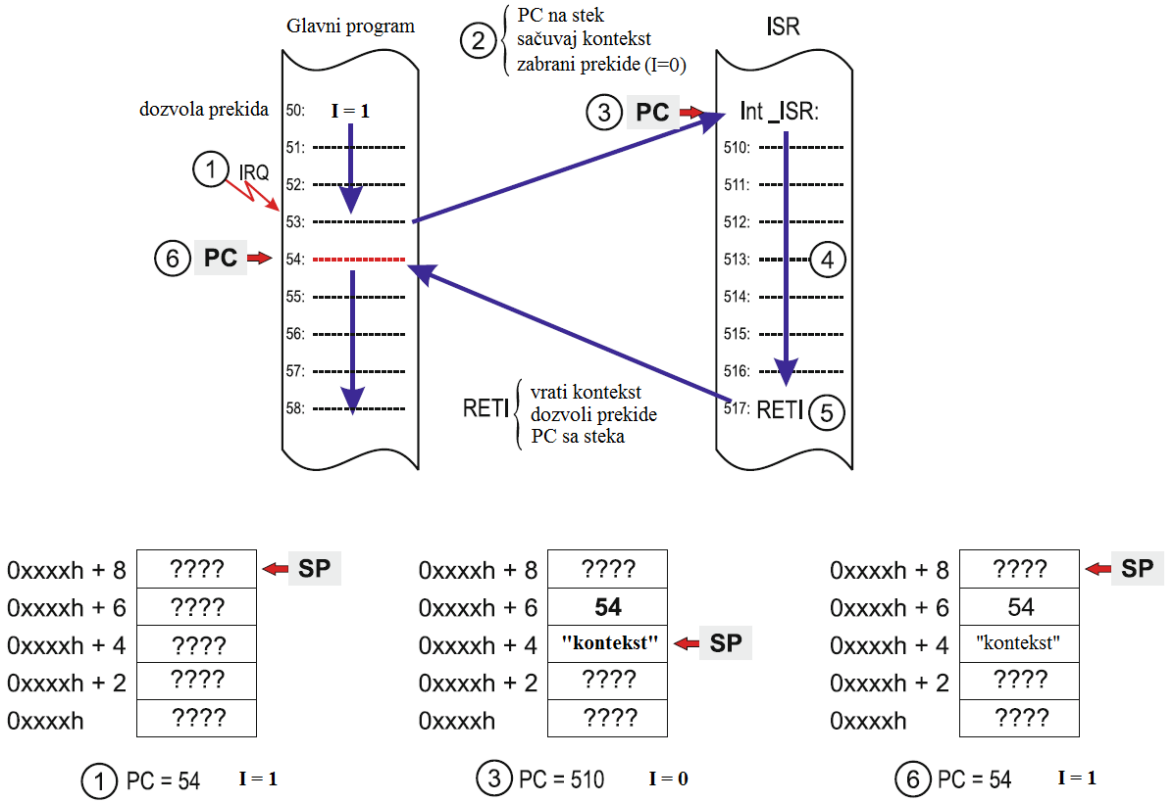


Slika 9.3 Algoritam obrade prekida

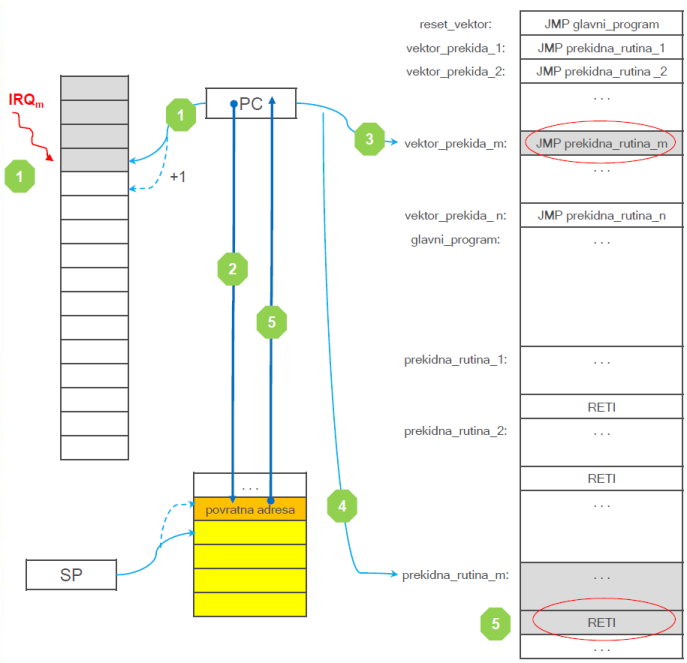
Prilikom obrade zahteva za prekidom dešavaju se sledeći koraci (videti sliku 9.4a):

1. Zahtev za prekidom dolazi do procesora u trenutku kada on izvršava instrukciju iz glavnog programa na adresi 53. Nakon detekcije zahteva za prekidom procesor najpre kompletira izvršavanje instrukcije sa adrese 53.
2. Procesor smešta trenutni sadržaj PC registra, koji pokazuje na adresu 54, trenutni "kontekst" (sadržaj registara koji se menjaju u toku rada prekida) na stek (stek je prikazan u donjem delu slike). Nakon toga procesor zabranjuje dozvolu svih novih prekida, brišući sadržaj I bita.
3. U PC registar smešta se adresa prve instrukcije prekidnog podprograma (ISR) koji je asociran izvoru prekida preko kojega je stigao zahtev za prekidom (u datom primeru to je adresa 510). Način na koji se ovo određuje biće objašnjen kasnije.
4. Procesor izvršava asocirani prekidni podprogram gde je na kraju RETI instrukcija.
5. Pojava RETI instrukcije uzrokuje vraćanje "konteksta" kao i vrednosti PC registra sa steka.
6. Procesor nastavlja sa izvršavanjem glavnog programa, izvršavajući instrukciju sa adrese 54

Pogledati na slici 9.4b ilustraciju slične sekvence, a na slici 9.4c kako bi izledala takva sekvenca u asembleru za ATmega328 mikrokontroler.



Slika 9.4a Sekvenca obrade zahteva za prekidom



Slika 9.4b Sekvenca obrade zahteva za prekidom

U okviru prethodne sekvence događaja prilikom obrade zahteva za prekidom nije odgovoreno na pitanje kako procesor zna koju početnu adresu prekidnog podprograma treba da smesti u PC. U slučaju samo jednog izvora prekida, rešenje je trivijalno. Međutim ukoliko imamo više izvora prekida ovaj postupak može biti složen. Drugi problem koji se tu može pojaviti je istovremeno stizanje više od jednog zahteva za prekidom. S obzirom da je procesor sekvencijalna mašina, jednom trenutku može da opsluži samo jedan zahtev. Jedno od rešenja ovog problema je postavljanje prirriteta prekida o čemu će biti reči kasnije.

```
Adresa  Labela      Kod             Komentar

0x0000              jmp RESET       ;skok na pocetak glavnog programa
0x0002              jmp EXT_INT0    ;skok na rutinu eksternog prekida 0
0x0004              jmp EXT_INT1    ;skok na rutinu eksternog prekida 1

...

0x0032              jmp SPM_READY
0x0034  RESET:      ...             ;glavni program

...

        EXT_INT0:   push SREG       ;kontekst (sadrzaji registara
                    push Rx         ;koji ce biti menjani)
                    push Ry         ;cuva se na steku
                    ...
                    sei             ;I <- 1 dozvola ugnezdenih prekida
                                    ;(opciono)

                    ...             ;telo prekidne rutine

                    pop Ry          ;restauracija konteksta
                    pop Rx
                    pop SREG
                    reti            ;povratak iz prekidne rutine
```

Slika 9.4c Sekvenca obrade zahteva za prekidom u asembleru

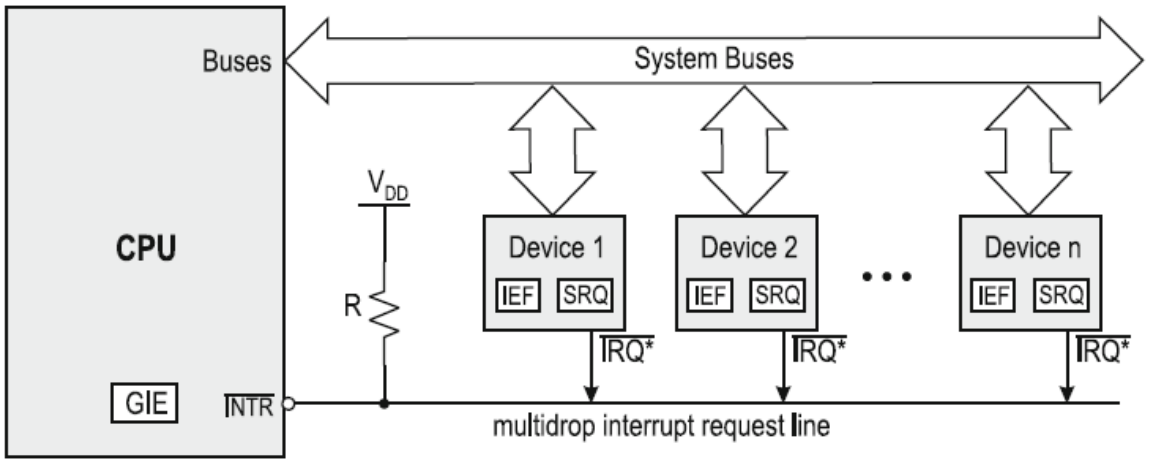
Identifikacija izvora prekida

Tokom evolucije embeded sistema, predloženi su različiti postupci za identifikaciju izvora zahteva za prekidom u slučaju kada u sistemu postoji više od jednog periferijskog uređaja koji može da generiše zahtev za prekidom.

U opštem slučaju, svi postupci mogu se klasifikovati u jednu od tri grupe:

- postupci bazirani na nevektorskim prekidima,
- postupci bazirani na vektorskim prekidima, i
- postupci bazirani na auto-vektorskim prekidima.

1) Nevektorski sistemi

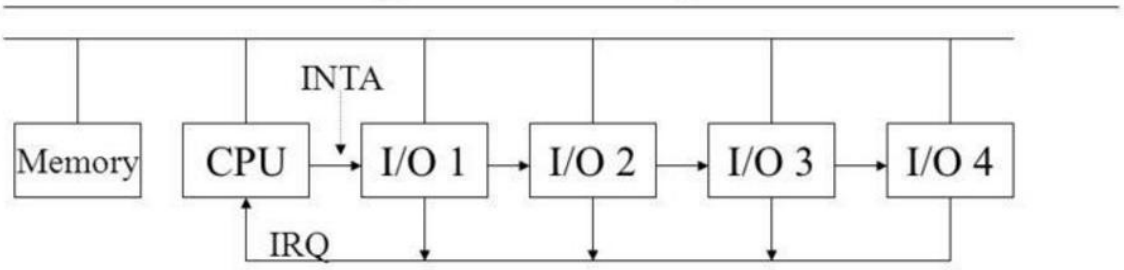


Slika 9.5 Nevektorski sistemi

CPU prima zahtev za prekidom preko jedne linije od svih periferija. Problem nastaje ako više od jedne periferije zahteva obradu prekida. Tada CPU proziva pojedinačno periferije i čita njihove SRQ (eng. *service request*) indikatore (videti sliku 9.5). Ovde se prioritet izvršavanja obrada prekida u slučaju više zahteva rešava redosledom prozivke.

2) Vektorski prekidi

Ovde se dodaje jedan takt za INTA (eng. *Interrupt Acknowledged*) potvrdu prijema zahteva za prekidom (videti sliku 9.6). Svaka periferija ima svoj ID identifikator na osnovu kojeg se računa adresa za početak prekidne rutine. Pod pojmom vektor se misli na adresu prekidne rutine.



Slika 9.6 Ilustracija INTA u vektorskim sistemima

Kod vektorskih sistema, u sistemskoj memoriji postoji posebna zona u koju se smešta tabela u kojoj se nalaze asocijacije između svakog izvora prekida i početnih adresa prekidnih podprograma koji su namenjeni za njihovu obradu. Ova memorijska zona se često naziva i sistemska vektor tabela. Procesori Intel x86 arhitekture koriste ovakav način rada sa prekidima.

3) Autovektorski sistemi

Svaki izvor prekida ili periferija ima fiksnu početnu adresu za obradu prekida i ona se ne može menjati. Nema računanja adrese niti je potreban INTA ciklus. Po pojavi zahteva za obradom prekida CPU automatski učitava početnu adresu za obradu prekida iz tabele

vektora prekida. Ukoliko je prekidna rutina veoma kratka moguće je kompletno smestiti unutar tabele vektora, a ako nije onda se obično postavi instrukcija skoka na adresu gde se nalazi prekidna rutina. Mikrokontroleri AVR, x51, PIC i MSP430 imaju ovakav način obrade prekida.

Vrste sistema identifikacije prekida kod mikrokontrolera

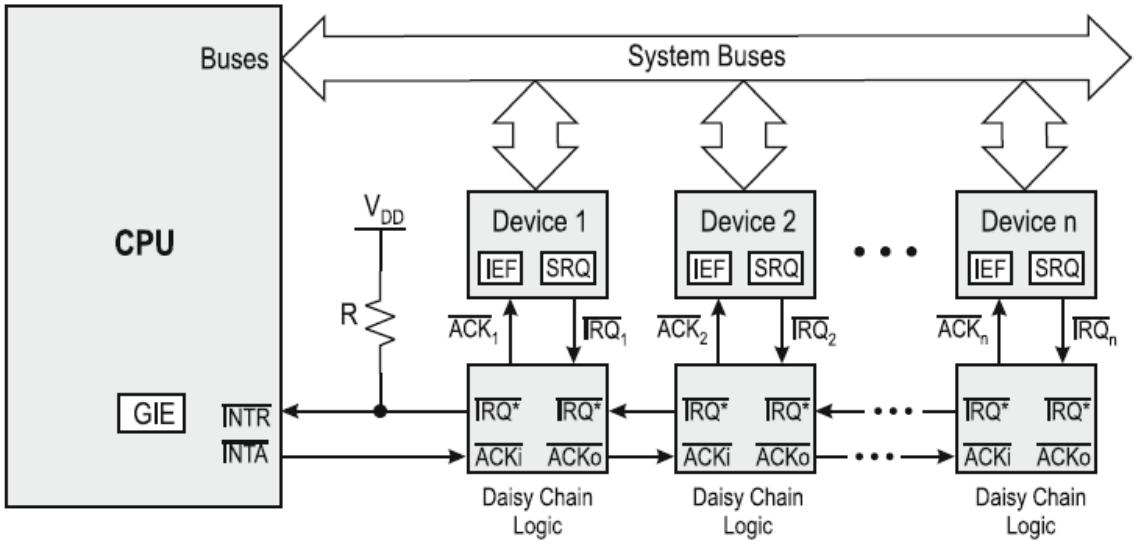
U zavisnosti da li je moguće modifikovati početne adrese prekidnih podprograma postoje dve najčešće vrste mikrokontrolera:

- Mikrokontroleri sa fiksnim početnim adresama prekidnih podprograma - u PC registar upisuje se predefinisana početna adresa prekidnog podprograma. Početna adresa prekidnog podprograma ne može se menjati od strane programera.

- Mikrokontroleri sa fiksnim vektorima prekida – u PC registar se upisuje početna adresa prekidnog podprograma iz sistemske vektor tabele na osnovu vektora pridruženog dotičnom prekidu. Programer ne može menjati vektore prekida ali može menjati sistemsku vektor tabelu. Ovo pruža mogućnost programeru da donekle proizvoljno organizuje pozivanje prekidnih podprograma unutar memorijskog adresnog prostora mikrokontrolera.

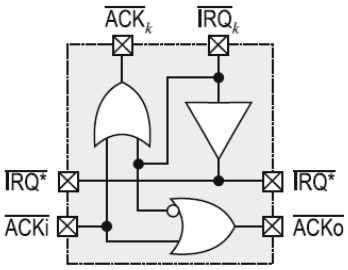
Razrešavanje problema prioriteta pri obradi prekida

1) *Daisy-chain* metod



Slika 9.7 Daisy chain metoda

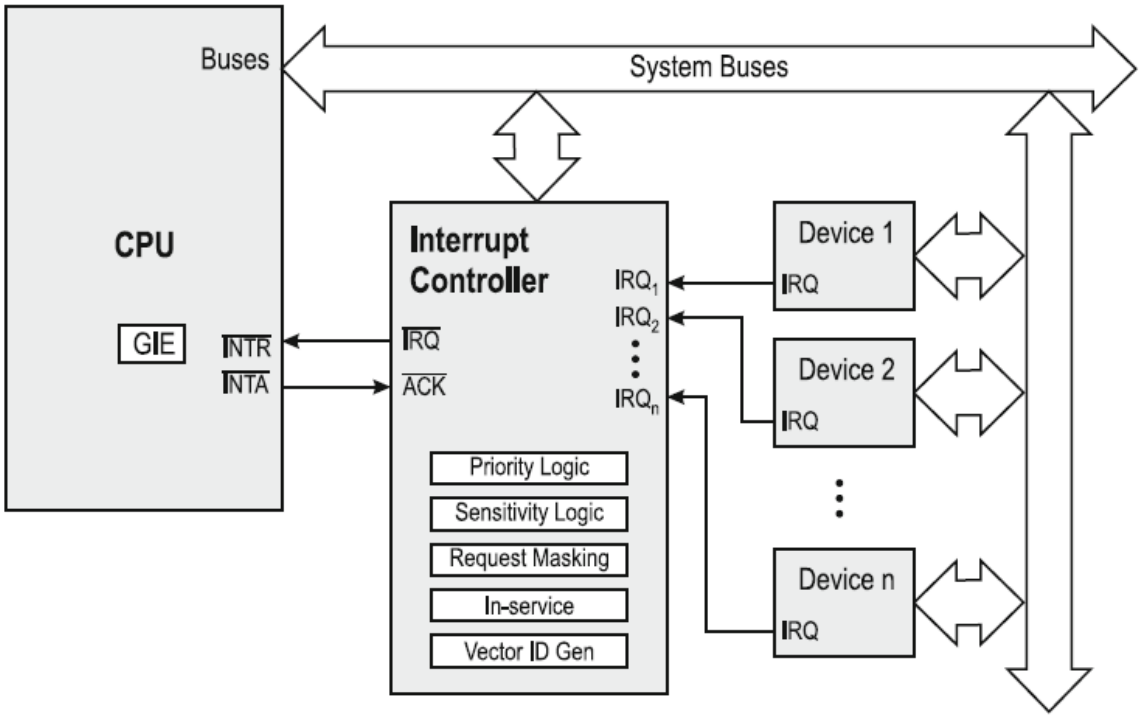
Logička struktura *daisy-chain* linka je data na slici 9.8:



Slika 9.8 Logička struktura *daisy-chain* linka

Ova metoda radi po principu: „što je uređaj bliži CPU ima viši prioritet“ i nije moguće menjati prioritet (videti sliku 9.7). Svrha dodatne logike (slika 9.8) kod ove metode je da blokira potvrdu (eng. *acknowledgment,* ACK) da ne ide dalje od najvišeg prioriteta onog ko je zahtev poslao.

2) Upotreba kontrolera prekida



Slika 9.9 Primena kontrolera prekida

Kontroler prekida (slika 9.9) vrši arbitražu u vezi prioriteta, ali i druge aktivnosti tipa dozvole/zabrane, podešavanja načina okidanja prekida i slično. Veoma konfigurabilno rešenje ali zahteva dodatnu komponentu u sistemu. Primer kontrolera prekida je Intel 8259A.

9.2 SISTEM PREKIDA MIKROKONTROLERA ATMEGA328

U mikroprocesorskim sistemima često je potrebno istovremeno pratiti rad više perifernih jedinica. To se može postići neprestanim prozivanjem jedne po jedne jedinice, proveravajući njihova stanja. Ako se utvrdi da je došlo do neke promene na nekoj od tih periferija na koju treba reagovati, preduzimaju se odgovarajuće akcije kao odgovor na promene. Na primer, nakon startovanja konverzije A/D konvertora neprekidno se vrši čitanje stanja BUSY nožice konvertora, koje označava da li je konverzija u toku ili je završena. Ako neko očitavanje pokaže da je konverzija završena, tada se učita odgovarajući podatak sa konvertora. U ovom slučaju mikrokontroler je neprestano zauzet proverom stanja na liniji BUSY A/D konvertora.

Drugi način praćenja rada više perifernih jedinica je da same jedinice jave kada je potrebno opsluživanje. Ta metoda se naziva metodom prekida, dok se prethodno opisana naziva metodom prozivanja (eng. *polling*). Dakle, kada se koristi metoda prekida, periferna jedinica posebnim signalom javlja kada je neophodna reakcija upravljačkog uređaja tj. mikrokontrolera. Taj signal se zove zahtev za prekidom odnosno interaptom (interrupt request). Slučaj iz prethodnog primera bi se mogao rešiti povezivanjem BUSY nožice konvertora na odgovarajuću INT (spoljašnji prekid) nožicu mikrokontrolera. Kada se BUSY deaktivira izaziva se prekid. Nakon što mikrokontroler uvaži zahtev, prelazi na podprogram za opsluživanje prekida (u ovom slučaju čitanje vrednosti konverzije), a nakon toga nastavlja sa izvršavanjem programa gde je prekinut u trenutku stizanja zahteva za prekidom. U ovom slučaju mikrokontroler se ne opterećuje proverom stanja na periferiji nego samo izvodi odgovarajuću akciju kada je to potrebno. Iz ovoga je jasno da u ovom slučaju mikrokontroler potroši mnogo manje vremena za opsluživanje periferija, zbog čega je i program može biti mnogo efikasniji.

Izvori prekida kod mikrokontrolera ATmega328

Kod ovog mikrokontrolera postoji dvadeset i šest izvora prekida koji su prikazani na slici 9.10. Ova tabela ujedno reguliše i prioritet prekida tako što je prekid sa nižim brojem višeg prioriteta (npr. RESET ima najviši prioritet).

Podeljeni u grupe ovi izvori prekida se mogu klasifikovati kao:

- spoljašnji prekidi (eng. *external interrupt*)
- prekidi na osnovu promena na pinu
- prekid *watch-dog* tajmera
- prekidi tajmera 0, 1 i 2 (eng. *timer interrupt*)
- prekidi serijskih interfejsa (SPI, USART i TWI) i
- prekid analognog dela (A/D konvertora i analognog komparatora)
- 3 sistemska prekida (Reset, EE *ready* i STM)

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

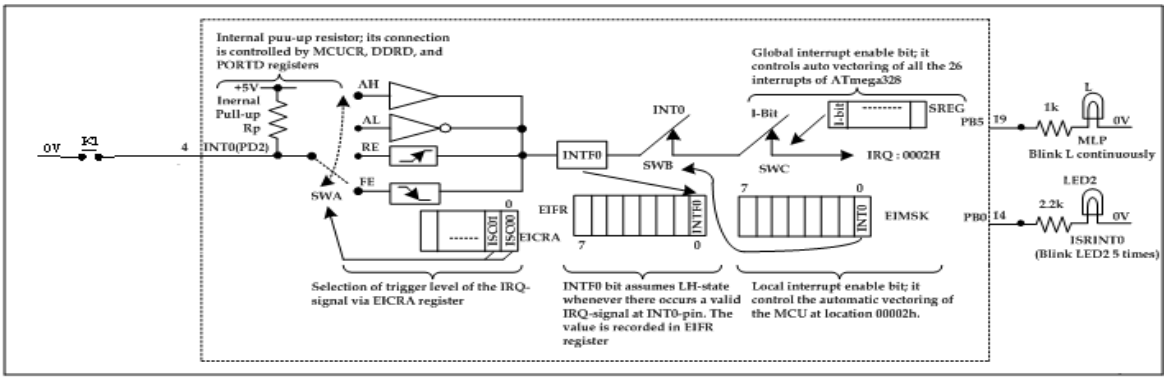
Slika 9.10 Izvori prekida kod ATmega328

Spoljašnji prekidi 0 i 1

Mikrokontroler poseduje dva ulaza – INT0 i INT1 (deo su porta D, PD2 i PD3). Na njih se može dovesti signal zahteva za prekidom sa bilo kog uređaja iz okruženja mikrokontrolera. Adresa od koje počinje potprogram za opsluživanje spoljašnjeg prekida 0 je 0002H. Za spoljašnji prekid 1 ta adresa je 0004H.

Na slici 9.11 prikazana je logička struktura spoljašnjeg prekida INT0 (isto je i za INT1). Unutrašnja struktura se može posmatrati u pet sekcija:

1- uključenje *pull-up* otpornika
2- postavljanje da li je reagovanje na nivo ili ivicu signala
3- memorisanje zahteva za prekidom
4- dozvola INT0
5- globalna dozvola prekida



Slika 9.11 Unutrašnja logička struktura INT0

MCUCR (eng. *MCU Control Register*), DDRD (eng. *Data Direction Register for PORTD*) i PORTD zajedno kontrolišu uključenje/isključenje unutrašnjeg *pull-up* otpornika.

Određivanje da li je reagovanje na nizak nivo, rastuću ili opadajuću ivicu ili bilo koju promenu se radi pomoću EICRA (eng. *External Interrupt Control Register A*) registra odabirom odgovarajućih kombinacija bita.

Da se desio zahtev za prekidom se može videti čitanjem odgovarajućeg bita EIFR (eng. *External Interrupt Flag Register*) registra.

Dozvola/zabrana spoljašnjeg prekida INT0 (ili INT1) se vrši setovanjem odgovarajućeg bita u EIMSK (eng. *External Interrupt Mask Register*) registru.

Globalna dozvola/zabrana prekida se vrši setovanjem/resetovanjem I-bita SREG (eng. *Status Register*) registra.

Primer programa

```
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    DDRD &= ~(1 << DDD2);      // PD2 pin je ulazni

    PORTD |= (1 << PORTD2);    // ukljuci Pull-up

    EICRA |= (1 << ISC00);     // INT0 reaguje na nizak nivo

    EIMSK |= (1 << INT0);      // uključi INT0

    sei();                     // dozvoli prekide

    while(1)
    {
        /*super petlja*/
    }
}


ISR (INT0_vect)
{
    /* kod prekidne rutine */
}
```

Prekidi usled promena na pinu

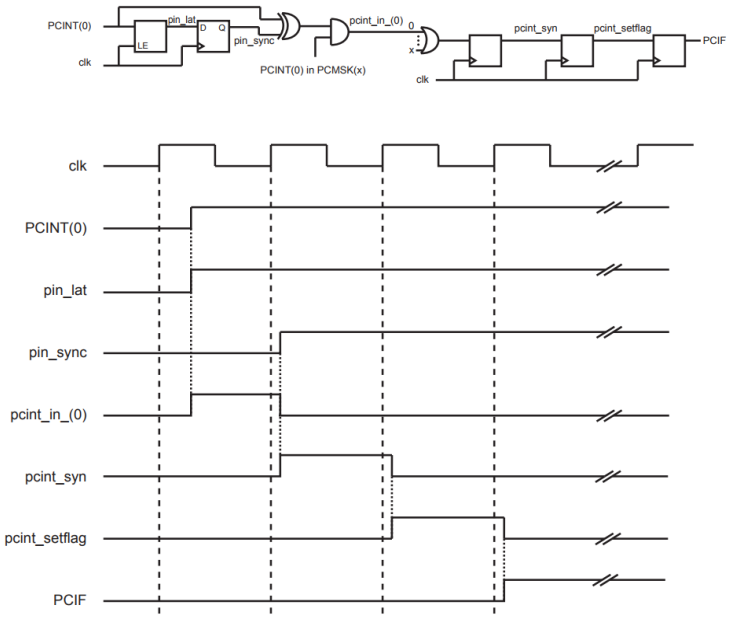
Mikrokontroler ATmega328 podržava aktivaciju prekida usled promena na bilo kojem pinu portova B, C ili D. Ovi prekidi su označeni kao PCINTx, x=0, 1 ili 2. Zbog neekonomičnosti da se svakom pinu dodeli poseban prekid oni su grupisani na sledeći način:

PCINT 7 .. 0 – PB 7..0

PCINT 14 .. 8 – PC 6..0
PCINT 23 .. 16 – PD 7..0

PCINT15 nije podržan.

Unutrašnja logička struktura reagovanja na ovu vrstu prekida kao i vremenski dijagrami sinhronizacije su dati na slici 9.12.



Slika 9.12 Unutrašnja logička struktura PCINTx prekida

Pri radu sa ovom vrstom prekida koriste se registri PCICR za dozvolu nekih od tri PCINT. Da se desio prekid ove vrste se može saznati čitanjem bita PCIFR registra. Dozvola pojedinih PCINT 23..0 se vrši pomoću PCMSKx,(x=0, 1 ili 2) registara, a globalna dozvola setovanjem I-bita u SREG registru.

Primer programa

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define PCINT_IN PORTB0
#define LED PORTB1

int main()
{
    DDRB &= ~(1 << DDB0);     // PB0 je ulaz
    PORTB |= (1 << PORTB0);   // postavi pull-up na PB0

    PCMSK0 = (1 << PCINT0);   //dozvoli pin-change prekid na PB0
    PCICR = (1 << PCIE0);     // dozvoli PCINT0 prekid
    Sei();                    // dozvoli prekide

    while (1)
    {
        // super petlja
    }
}

ISR(PCINT0_vect)              // ISR za prekid na promenu PB0
{
    // prekidna rutina
}
```