

Osnovi mikroprocesorskih i mikrokontrolerskih sistema - Prekidi

prof. dr Ivan Mezei

13. maj 2022.

Glava 8

Prekidi

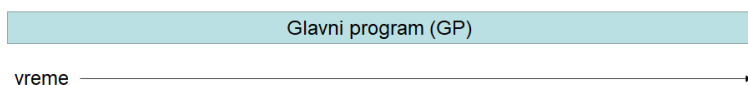
Generalno govoreći, prekid (eng. *interrupt*) predstavlja događaj koji prethodni tok procesa prekida da bi se dešavalo nešto drugo (npr. telefonski poziv koji prekid prethodnu aktivnost). U embeded sistemima prekid predstavlja događaj koji je najčešće hardverski iniciran, koji zaustavlja mikroprocesor u redovnom izvršavanju programa i potom program nastavlja rad na nekom drugom mestu. Nastavak programa na drugom mestu tiče se procesa koji nazivamo obrada prekida (ili prekidna rutina, eng. *interrupt service routine* - ISR). Pri tome, mikroprocesor 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 8.1.

8.1 Obrada prekida

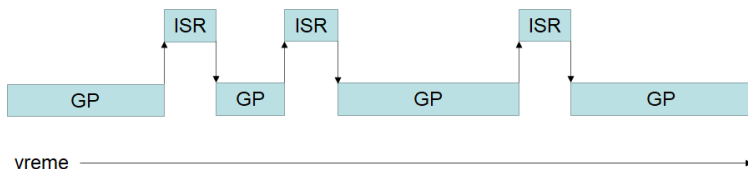
Mikrokontroleri u opštem slučaju mogu da rade sa više različitih periferija koje zahtevaju odgovarajuću obradu. U slučajevima kada periferija zahteva obradu od mikrokontrolera, postoje dva načina koji se koriste u obradi zahteva:

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

Izvršavanje programa bez prekida:



Izvršavanje programa sa prekidima:



Slika 8.1: Ilustracija rada programa bez obrade i sa obradom prekida

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. Primer iz embeded sistema bi bio da se nakon startovanja konverzije A/D konvertora neprekidno 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.

Kod metode koja koristi sistem prekida, periferija obaveštava mikrokontroler da je potrebna obrada. Dole 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. Slučaj iz prethodnog primera bi se mogao rešiti povezivanjem BUSY nožice A/D 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 program može biti mnogo efikasniji. Očigledno, prekidni metod je daleko efikasniji od metoda prozivke.

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

- Kompaktan i modularan softver – prekidni programi 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, i
- Brže vreme odziva – kada u sistemu postoji veći broj različitih periferijskih jedinica, odnosno spoljašnjih događaja, koje je potrebno servisirati, pažljivo projektovane ISR rutine obezbeđuju brži odziv na zahteve za obradom. Pametno korišćenje sistema prioriteta prekida obezbeđuje brzo vreme odziva.

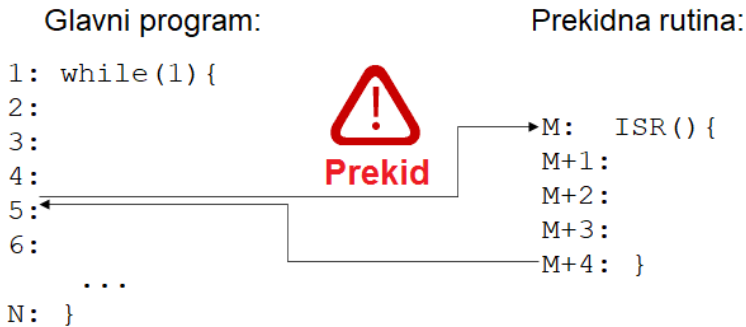
U nastavku ćemo dati nekoliko primera sistema sa obradom prekida. Pođimo od sistema sa LED diodom i tasterom gde se na svaki pritisak tastera dioda uključuje/isključuje. Uključenje/isključenje se vrši u prekidnoj rutini koju aktivira pritisak tastera. Ukoliko posmatramo sistem koji koristi serijski port, dolazna poruka koja stiže putem serijskog porta aktivira prekid i u prekidnoj rutini se ona negde smešta ili se preduzima neka akcija. U savremenim baterijski napajanim sistemima monitoring napona baterije gde se prekid generiše kada se dostigne napon koji je ispod nekog definisanog praga takođe je jedan od primera.

Da bi mogao da koristi sistem prekida, projektant embedded sistema mora da uključi odgovarajuće hardverske i softverske komponente. Pored toga, neophodno je i da konfiguriše sistem za pravilan rad sa prekidima. U zavisnosti od stepena integracije mikroprocesora na kome se bazira embedded 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.

Kao što je već napomenuto, 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 neki

su 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** sa programerske tačke gledišta. Drugim rečima prekid može da se desi u bilo kom trenutku, u bilo kom delu izvršavanja glavnog programa. Pored sličnosti sa pozivom potprograma ovo je ujedno i suštinska razlika. Iz ovih razloga softverska komponenta koja pruža podršku radu sa prekidima mora biti pažljivo napisana. Na slici 8.2 prekid se desio između 4. i 5. komande glavnog programa. Po završetku obrade prekida, glavni program nastavlja od 5. komande.



Slika 8.2: Ilustracija rada glavnog programa i prekidne rutine

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

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

Po pravilu, ovo je moguće konfigurisati softverski podešavanjem odgovarajuće vrednosti u registru koji je za to namenjen.

8.1.1 Tipovi prekida

Postoje **maskirani** i **nemaskirani** prekidi. Maskirane prekide je moguće zabraniti dok su nemaskirani prekidi rezervisani za systemske aktivnosti i nije ih moguće zabraniti. Većina zahteva za prekidom 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 bita koji se nalaze unutar statusnog registra (npr. *Global Interrupt Enable* - GIE) ili nekog posebnog registra namenjenog za tu svrhu (npr. *Interrupt Enable* - IE registar). Kod ATmega328P 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 mikroprocesor 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, generiše se nemaskirajući zahtev za prekidom. Mikroprocesor započinje obradu ovog zahteva za prekidom i pristupa 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 započinje postupak isključenja sistema.

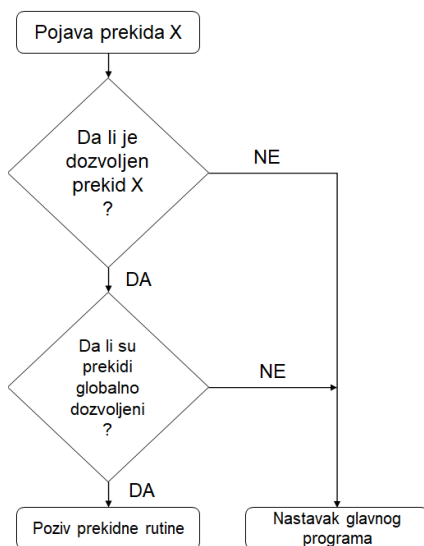
Druga podela prekida je bazirana na lokaciji izvora prekida u odnosu na mikrokontroler, pa tako imamo prekide uzrokovane spoljašnjim ili unutrašnjim događajima. Primeri spoljašnjih događaja su sledeći: kada je na serijski port stigao ili se šalje neki karakter, promena signala na nekom pinu, aktivacija zahteva za prekidom na pinu za spoljašnji prekid itd. Primeri prekida na bazi unutrašnjih događaja su izuzeci pri aritmetičkim operacijama (npr. deljenje sa nulom), prekidi tajmera, fluktuacija napajanja itd.

8.1.2 Sekvenca obrade zahteva za prekidom

Nakon pojave zahteva za prekidom (eng. *interrupt request* - IRQ), da bi moglo da dođe do prelaska sa glavnog programa na obradu prekida vrši se tzv. dvofazna provera dozvole prekida. Po pojavi nekog prekida, ova provera se sastoji od provere da li je dozvoljen konkretan prekid i ako jeste da li je aktivirana globalna dozvola prekida. Algoritam dvofazne provere dozvole prekida prikazan je na slici 8.3

Ukoliko su prekidi dozvoljeni tada prilikom obrade zahteva za prekidom dešava se sledećih šest koraka (videti sliku 8.4):

1. Zahtev za prekidom dolazi do CPU u trenutku kada on izvršava instrukciju iz glavnog programa na adresi 3. Nakon detekcije zahteva za prekidom (IRQ) CPU najpre kompletira izvršavanje instrukcije sa adrese 3.
2. CPU smešta trenutni sadržaj PC registra, koji pokazuje na adresu 4, trenutni "kontekst" (sadržaj registara koji se menjaju u toku obrade prekida) na stek.

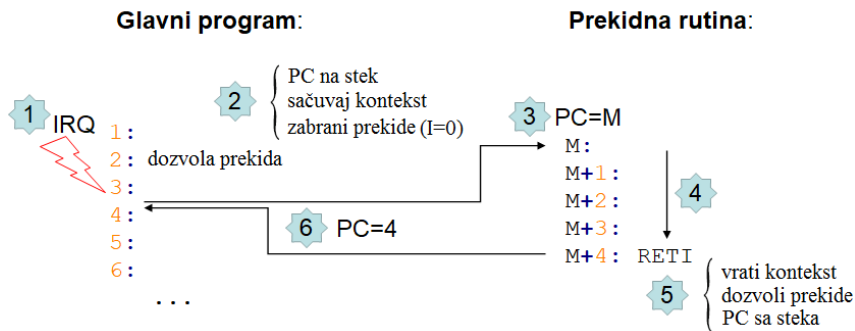


Slika 8.3: Algoritam dvofazne provere dozvole prekida

Nakon toga, CPU zabranjuje obradu svih novih prekida¹, brišući sadržaj I bita.

3. U PC registar smešta se adresa prve instrukcije prekidnog podprograma (ISR) koji je asociiran izvoru prekida preko kojega je stigao zahtev za prekidom (u datom primeru to je adresa M). Način na koji se ovo određuje biće objašnjen kasnije.
4. CPU izvršava asociirani prekidni podprogram na čijem kraju je RETI instrukcija.
5. Pojava RETI instrukcije uzrokuje vraćanje "konteksta" kao i vrednosti PC registra sa steka.
6. CPU nastavlja sa izvršavanjem glavnog programa, izvršavajući instrukciju sa adrese 4.

¹osim ukoliko su dozvoljeni tzv. ugneždeni prekidi



Slika 8.4: Obrada zahteva za prekidom

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 ugnjezdenih prekida
			; (opciono)
		...	;telo prekidne rutine
		pop Ry	;restauracija konteksta
		pop Rx	
		pop SREG	
		reti	;povratak iz prekidne rutine

Slika 8.5: Obrada zahteva za prekidom realizovana u assembleru

U okviru prethodne sekvence događaja prilikom obrade zahteva za prekidom nije odgovoreno na pitanje kako CPU zna koju početnu adresu prekidnog podprograma treba da smesti u registar 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 CPU sekvencijalno digitalno kolo, u jednom trenutku može da opsluži samo jedan zahtev. Jedno od rešenja

ovog problema je postavljanje pririteta prekida o čemu će biti reči kasnije.

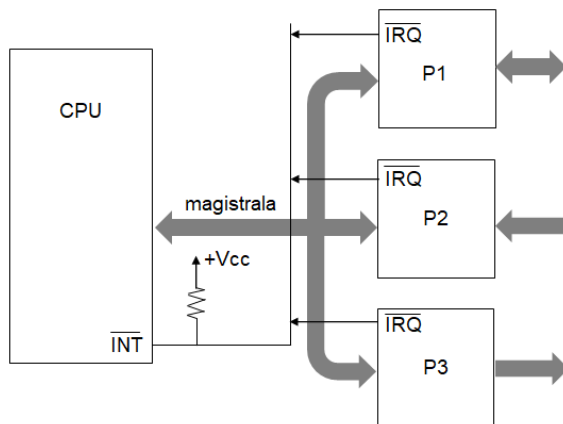
Sekvenca obrade spoljašnjeg prekida (EXT_INT0) realizovana u assembleru za ATmega328P je data na slici 8.5. Od adrese 0x0000 do adrese 0x0032 se nalazi tabela vektora prekida. Nakon toga kreće glavni program. Po pojavi prekida se prelazi na obradu EXT_INT0. Prvo se čuva kontekst na steku i potom opcionalna dozvola ugnežđenih prekida. Nakon završetka obrade se vraća kontekst i potom pojavom RETI se vrši povratak u glavni program.

8.1.3 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 perifernog uređaja koji može da generiše zahtev za prekidom.

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

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



Slika 8.6: Nevektorski sistem

Kod nevektorskih sistema CPU prima zahtev za prekidom preko jedne linije od svih periferija (vidi sliku 8.6). Ovo se realizuje preko linije koja radi u tzv. "ožičenoj

I logici” jer je dovoljno da jedna periferija spusti napon do nule spuštanjem svoje IRQ linije koja je aktivna na niskom nivou. Ukoliko ni jedna periferija ne traži obradu prekida tada je INT ulaz na CPU neaktivan jer preko otpornika imamo visok nivo signala. Problem nastaje ako više od jedne periferije zahteva obradu prekida jer CPU ne zna koja je to periferija. Tada CPU proziva pojedinačno periferije i čita koja od njih je tražila zahtev za prekidom (IRQ). Ovo se obično rešava čitanjem nekog indikatora. U ovakvim sistemima se prioritet izvršavanja obrada prekida u slučaju više zahteva rešava redosledom prozivke.

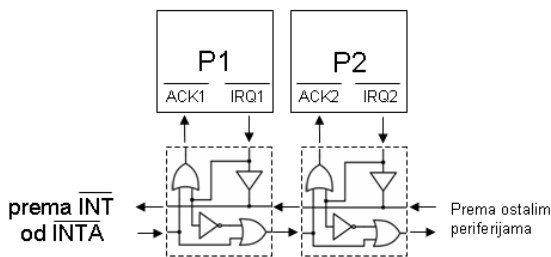
Kod vektorskih sistema se dodaje još jedna linija INTA (eng. *Interrupt Acknowledged*) koja služi za potvrdu prijema zahteva za prekidom. Svaka periferija ima svoj ID identifikator na osnovu kojeg se računa adresa za početak prekidne rutine. Pod pojmom vektor se misli na adresu prekidne rutine. Kod vektorskih 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. Mikroprocesori Intel x86 arhitekture koriste ovakav način rada sa prekidima.

Kod autovektorskih sistema 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.

8.1.4 Razrešavanje problema prioriteta pri obradi prekida

Kao što je već rečeno, kod nevektorskih sistema je prioritet prekida rešen redosledom prozivke periferija. Kod vektorskih sistema može da se koristi metoda ulančavanja (eng. *daisy chain*). Kod svake periferije pored postojeće linije za zahtev za prekidom INT dodaje se i linija za potvrdu zahteva INTA koja poseduje i dodatnu logiku prikazanu na slici 8.7. Prikazan je primer sa dve periferije. Ova logika obezbeđuje rad po principu: „što je uređaj bliži CPU ima viši prioritet“ i nije moguće menjati prioritet. Svrha dodatne logike kod ovakvih sistema je da blokira potvrdu (eng. *acknowledgment* - ACK) da ne ide dalje od najvišeg prioriteta onog ko je zahtev poslao.

Kod vektorskih sistema se ponekada koristi i kontroler prekida. Ovo je veoma konfigurabilna komponenta koja vrši arbitražu u vezi prioriteta, ali i druge aktiv-

Slika 8.7: Dodatna logika kod *Daisy-chain* metode

nosti tipa dozvole/zabrane, podešavanja načina okidanja prekida i slično. Ovakvo rešenje zahteva dodatnu komponentu u sistemu pa se danas ređe koristi. Primer kontrolera prekida je Intel 8259A.

Kod autovektorskih sistema se prioritet prekida može rešavati putem registra za postavku višeg prioriteta nekog od prekida u odnosu na ostale. Primer je registar IP (eng. *Interrupt Priority*) kod mikrokontrolera Intel 8051. Drugo rešenje je da pozicija prekida u tabeli vektora prekida automatski određuje i prioritet. Ovakvo je rešenje kod ATmega328P kog koga RESET prekid ima najviši prioritet (adresa 0).

8.2 Sistem prekida mikrokontrolera ATmega328P

Kod mikrokontrolera ATmega328P postoji dvadeset i šest izvora prekida koji su prikazani u tabeli 8.1. Pored broja prekida i odgovarajuće adrese u tabeli vektora prekida, dati su i izvor prekida, objašnjenje kao i način poziva upotrebom ISR() makroa. 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 interrupts*),
- 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),

Tabela 8.1: Tabela vektora prekida kod ATmega328P

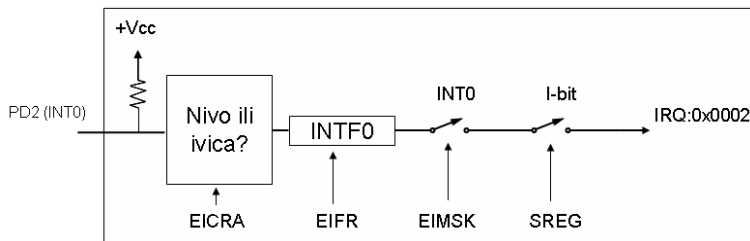
#	Adresa	Izvor	Opis	Ime ISR() makroa
1	0x0000	RESET	Reset	
2	0x0002	INT0	Spolj. prekid 0 (pin D2)	(INT0_vect)
3	0x0004	INT1	Spolj. prekid 1 (pin D3)	(INT1_vect)
4	0x0006	PCINT0	Pinovi D8 to D13	(PCINT0_vect)
5	0x0008	PCINT1	Pinovi A0 to A5	(PCINT1_vect)
6	0x000A	PCINT2	Pinovi D0 to D7	(PCINT2_vect)
7	0x000C	WDT	Prekid Watchdog tajmera	(WDT_vect)
8	0x000E	T2 COMPA	T/C2 Compare Match A	(TIMER2_COMPA_vect)
9	0x0010	T2 COMPB	T/C2 Compare Match B	(TIMER2_COMPB_vect)
10	0x0012	T2 OVF	Pretek T/C2	(TIMER2_OVF_vect)
11	0x0014	T1 CAPT	T/C1 Capture Event	(TIMER1_CAPT_vect)
12	0x0016	T1 COMPA	T/C1 Compare Match A	(TIMER1_COMPA_vect)
13	0x0018	T1 COMPB	T/C1 Compare Match B	(TIMER1_COMPB_vect)
14	0x001A	T1 OVF	Pretek T/C1	(TIMER1_OVF_vect)
15	0x001C	T0 COMPA	T/C0 Compare Match A	(TIMER0_COMPA_vect)
16	0x001E	T0 COMPB	T/C0 Compare Match B	(TIMER0_COMPB_vect)
17	0x0020	T0 OVF	Pretek T/C0	(TIMER0_OVF_vect)
18	0x0022	SPI, STC	SPI transfer završen	(SPI_STC_vect)
19	0x0024	USART, RX	Prijem (Rx) završen	(USART_RX_vect)
20	0x0026	USART, UDRE	Data Registar Prazan	(USART_UDRE_vect)
21	0x0028	USART, TX	Predaja (Tx) završena	(USART_TX_vect)
22	0x002A	ADC	A/D konverzija završena	(ADC_vect)
23	0x002C	EE READY	EEPROM spreman	(EE_READY_vect)
24	0x002E	AN. COMP	Analogni komparator	(ANALOG_COMP_vect)
25	0x0030	TWI	2-žični ser. interfejs(I2C)	(TWI_vect)
26	0x0032	SPM READY	SPM spremna	(SPM_READY_vect)

- prekid analognog dela (A/D konvertora i analognog komparatora), i
- 3 sistemska prekida (Reset, EE ready i STM).

U nastavku će biti obrađeni spoljašnji prekidi i prekidi usled promena na pinu dok će prekidi tajmera biti obrađeni u narednom poglavlju. Ostali prekidi nisu od interesa za ovaj predmet pa se čitalac upućuje na dodatnu literaturu [38].

8.2.1 Spoljašnji prekidi

Mikrokontroler poseduje dva ulaza – INT0 i INT1 (deo su porta D, PD2 i PD3) koji mogu biti ulazi za klasične spoljašnje prekide. Na njima 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 0x0002. Za spoljašnji prekid 1 ta adresa je 0x0004.



Slika 8.8: Unutrašnja logička struktura spoljašnjeg prekida

Na slici 8.8 prikazana je logička struktura spoljašnjeg prekida INT0 (isto je i za INT1). Unutrašnja struktura se može posmatrati u pet sekcija:

1. uključanje pull-up otpornika: MCUCR (eng. *MCU Control Register*), DDRD (eng. *Data Direction Register for PORTD*) i PORTD zajedno kontrolišu uključanje/isključanje unutrašnjeg *pull-up* otpornika.
2. postavljanje da li je reagovanje na nivo ili ivicu signala: 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.
3. memorisanje zahteva za prekidom: Da se desio zahtev za prekidom se može videti čitanjem odgovarajućeg bita EIFR (eng. *External Interrupt Flag Register*) registra.
4. dozvola INT0: Dozvola/zabrana spoljašnjeg prekida INT0 (ili INT1) se vrši setovanjem odgovarajućeg bita u EIMSK (eng. *External Interrupt Mask Register*) registru.
5. globalna dozvola prekida: Globalna dozvola/zabrana prekida se vrši setovanjem/resetovanjem I-bita SREG (eng. *Status Register*) registra.

Primer programa u programskom jeziku C koji realizuje prethodno prikazan je na slici 8.9.

```
#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);     // dozvoli INT0 prekid

    sei();                    // dozvoli prekide

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

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

Slika 8.9: Primer programa za rad sa spoljašnjim prekidom INT0

Posebna vrsta spolješnjih prekida su *prekidi usled promena na pinu*. Mikrokontroler ATmega328P podržava aktivaciju prekida usled promena na bilo kojem pinu portova B, C ili D. Ovi prekidi su označeni kao PCINT_x, 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

Dakle, svi prekidi usled promena na pinu su grupisani u tri grupe gde svaka ima svoj vektor prekida (ukupno 3). Treba primetiti da nije podržan prekid na pinu 15, tj. PCINT 15 ne postoji.

Pri radu sa ovom vrstom prekida koriste se registri PCICR (eng. *Pin Change Interrupt Control Register*) za dozvolu nekih od tri PCINT. Provera da li se desio prekid ove vrste se može uraditi čitanjem bita PCIFR (eng. *Pin Change Interrupt Flag Register*) registra. Dozvola pojedinih PCINT 23..0 se vrši pomoću PCMSK_x, (x=0, 1 ili 2) (eng. *Pin Change MaSK*) registara, a globalna dozvola setovanjem I-bita u SREG registru. Primer programa koji koristi prekide usled promene na pinu PB0 prikazan je na slici 8.10.

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

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
    }
}
```

Slika 8.10: Primer programa za rad sa prekidom usled promene na pinu

8.3 Pitanja

1. Šta predstavljaju prekidi generalno govoreći, a šta u slučaju embeded sistema?
2. Koja 2 načina postoje u obradi zahteva periferija?
3. Uporediti metodu prozivke i prekidnu metodu pri radu sa periferijama.
4. Navesti prednosti sistema koji rade sa prekidima.

5. Koji načini za indikaciju postojanja zahteva za prekidom unutar embedded sistema postoje?
6. Navesti koji tipovi prekida postoje.
7. Objasniti maskirajuće prekide.
8. Objasniti nemaskirajuće prekide.
9. Objasniti dvofaznu proveru dozvole prekida.
10. Objasniti i nacrtati sekvencu obrade zahteva za prekidom.
11. Kako se klasifikuju postupci za identifikaciju izvora prekida?
12. Opisati nevektorske sisteme za identifikaciju izvora prekida.
13. Opisati vektorske sisteme za identifikaciju izvora prekida.
14. Opisati autovektorske sisteme za identifikaciju izvora prekida.
15. Nacrtati i objasniti svrhu dodatne logike kod Daisy-chain metoda za rešavanje problema prioriteta prekida.
16. Objasniti upotrebu kontrolera prekida za rešavanje problema prioriteta prekida.
17. Kako se kod mikrokontrolera rešava pitanje prioriteta prekida?
18. Za ATmega328P, koliko ima izvora prekida i u koje grupe spadaju?
19. Kako je razrešen prioritet prekida kod ATmega328P?
20. Nacrtati i objasniti unutrašnju logičku strukturu spoljašnjeg prekida INT0.
21. Kako se vrši globalna dozvola/zabrana prekida kod ATmega328P?
22. Kako je realizovana mogućnost rada sa prekidima usled promena na pinu kod ATmega328P?

Bibliografija

- [1] Arthur W. Burks, Herman H. Goldstine, and John von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument", The Institute of Advanced Study, Princeton, USA, 1946-47.
- [2] J. Biggs, J. Myers, J. Kufel et al. A natively flexible 32-bit Arm microprocessor, *Nature* 595, pp. 532–536, 2021.
- [3] F. Arute, K. Arya, R. Babbush et al. Quantum supremacy using a programmable superconducting processor. *Nature* 574, pp. 505–510, 2019.
- [4] J. Hochstetter, R. Zhu, A. Loeffler et al. Avalanches and edge-of-chaos learning in neuromorphic nanowire networks. *Nat Commun* 12, 4008, 2021.
- [5] B. B. Brey, *Intel Microprocessors*, 8th Edition, Pearson, 2009.
- [6] S. Harris, D. Hariss, *Digital Design and Computer Architecture*, Arm edition, Morgan Kaufmann, 2015.
- [7] D. Patterson, J.L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, The Morgan Kaufmann Series in Computer Architecture and Design, 2nd Edition, 2021.
- [8] M. M. Mano, C. R. Kime, T. Martin, *Logic and Computer Design Fundamentals*, Fifth edition, Pearson, 2015.
- [9] R. S. Sandige, M. L. Sandige, *Fundamentals of Digital and Computer Design with VHDL*, McGraw Hill, 2012.
- [10] L. Null, J. Lobur *Essentials of Computer Organization and Architecture*, 5th Edition, Jones and Barlett Publishers, 2018.

- [11] E.O. Hwang, Digital Logic and Microprocessor Design With VHDL with Interfacing, Cengage Learning, Second Edition, 2017.
- [12] I. Mezei "Evolution of an educational microprocessor", Computer Applications in Engineering Education, 28(5), Wiley, pp. 1265-1277, 2020.
- [13] W. Stallings, Computer organization and architecture, 9th Edition, Pearson, 2013.
- [14] M. Abd-El-Barr, H. El-Rewini, Fundamentals Of Computer Organization And Architecture, John Wiley and Sons, Inc., 2005.
- [15] C. Hamacher, Z. Vranesic, S. Zaky, N. Manjikian, Computer Organization and Embedded Systems, 6th Edition, McGraw-Hill, 2012.
- [16] Manuel Jiménez, Rogelio Palomera, Isidoro Couvertier, Introduction to Embedded Systems, Springer, 2014.
- [17] D. Patterson, "Reduced Instruction Set Computers Then and Now" in Computer, vol. 50, no. 12, pp. 10-12, 2017. <https://doi.ieeecomputersociety.org/10.1109/MC.2017.4451206>
- [18] D. A. Patterson, J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Fourth Edition, Morgan Kaufmann, 2011.
- [19] J.L. Gustafson, Moore's Law. In: D. Padua D. (eds) Encyclopedia of Parallel Computing. Springer, 2011.
- [20] S.A. McKee, R.W. Wisniewski, Memory Wall. In: D. Padua (eds) Encyclopedia of Parallel Computing. Springer, 2011.
- [21] P. Bose, Power Wall. In: D. Padua (eds) Encyclopedia of Parallel Computing. Springer, 2011.
- [22] A. Tsakyridis, T. Alexoudi, A. Miliou, N. Pleros, and C. Vagionas, "10 Gb/s optical random access memory (RAM) cell," Opt. Lett. 44, pp. 1821-1824, 2019.
- [23] W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques", IEEE Wescon Proc., Aug. 1970.
- [24] C.G. Bell, A. Newell, Computer structures: Readings and Examples, McGraw-Hill, 1971.

- [25] S. Brown, Z. Vranesic, Fundamentals of Digital Logic with VHDL Design, McGraw Hill, 2009.
- [26] D. L. Perry, VHDL: Programming by Example, Fourth edition, McGraw Hill, 2002.
- [27] L. Null, J. Lobur, Essentials of Computer Organization and Architecture, Jones and Bartlet Learning, 3rd Ed., 2010.
- [28] D. Patterson, J.L. Hennessy, Computer Organization and Design RISC-V Edition: The Hardware Software Interface, The Morgan Kaufmann Series in Computer Architecture and Design, 1st Edition, 2017.
- [29] S. L. Harris, D. Harris, Digital Design and Computer Architecture, RISC-V Edition, Morgan Kaufman, 1st edition, 2021.
- [30] D. A. Patterson, J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, ARM edition, The Morgan Kaufmann Series in Computer Architecture and Design, 2016.
- [31] D. Kushner, "The making of arduino", IEEE Spectrum 26, 2011.
- [32] M. Banzi, "How Arduino is open-sourcing imagination", TEDtalk, Scotland, 2012. <https://www.youtube.com/watch?v=UoBUXOOdLXY> (datum pristupa: 31.01.2022.)
- [33] Sparkfun, "Arduino shields v2", <https://learn.sparkfun.com/tutorials/arduino-shields-v2>, (datum pristupa: 01.02.2022.)
- [34] Arduino Uno Rev3 schematic, https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf (datum pristupa: 02.02.2022.)
- [35] Arduino software, <https://www.arduino.cc/en/Main/Software> (datum pristupa: 02.02.2022.)
- [36] 74HC595 datasheet, Philips, 1998, <https://www.arduino.cc/en/uploads/Tutorial/595datasheet.pdf> (datum pristupa: 03.03.2022.)
- [37] 74HC165 datasheet, Texas Instruments, 2015 <https://www.ti.com/lit/ds/symlink/sn74hc165.pdf> (datum pristupa: 03.03.2022.)
- [38] ATmega328 datasheet, Atmel, 2015, https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (datum pristupa: 15.03.2022.)

- [39] Tutorials 83, Fundamentals of RS-232 serial communications, Analog Devices, 2001. <https://www.maximintegrated.com/en/design/technical-documents/tutorials/8/83.html> (datum pristupa: 31.03.2022.)