



CS120 - ORGANIZACIJA RAČUNARA

Sloj skupa instrukcija – ISA sloj

Lekcija 09

PRIRUČNIK ZA STUDENTE

CS120 - ORGANIZACIJA RAČUNARA

Lekcija 09

SLOJ SKUPA INSTRUKCIJA – ISA SLOJ

- ✓ Sloj skupa instrukcija – ISA sloj
- ✓ Poglavlje 1: Pregled nivoa ISA
- ✓ Poglavlje 2: Registri
- ✓ Poglavlje 3: Formati instrukcija mikroprocesora
- ✓ Poglavlje 4: Tipovi instrukcija mikroprocesora
- ✓ Poglavlje 5: Intel Core i7
- ✓ Poglavlje 6: Pokazne vežbe
- ✓ Poglavlje 7: Zadaci za samostalni rad
- ✓ Poglavlje 8: Domaći zadatak
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

❖ Uvod

UVOD

Pod pojmom arhitektura računara danas se po pravilu podrazumeva tzv. Arhitektura skupa instrukcija.

U ovoj lekciji ćemo detaljno govoriti o nivou arhitekture skupa instrukcija **ISA** (Instruction Set Architecture).

Ovaj nivo pozicioniran je između nivoa mikroarhitekture i nivoa operativnog sistema.

Istorijski posmatrano, ovaj nivo je razvijen pre bilo kog drugog nivoa, i u početku je to bio i jedini nivo. I danas nije neobično da se ovaj nivo ponekad naziva jednostavno "**arhitektura računara**" ili ponekad (netačno) kao "**asemblerški jezik računara**".

ISA nivo ima poseban značaj za projektante sistema jer predstavlja vezu između softvera i hardvera.

Iako je moguće kreirati hardver koji direktno izvršava programe napisane na C, C++, Java ili nekom drugom programskom jeziku visokog nivoa, to ne bi bila dobra ideja. U tom slučaju prednost prevodenja (**kompajliranja**) nad interpretiranjem se ne bi mogla iskoristiti. **Da bi bili od praktične koristi, računari bi uglavnom morali da budu u stanju da izvršavaju programe napisane na više programskega jezika, a ne samo na jednom.**

Pristup koji u suštini svi projektanti sistema imaju je da programi na različitim jezicima visokog nivoa budu prevedeni u formu zajedničku za sve – na jezik nivoa ISA – a hardver treba da nauči samo jezik nivoa ISA.

ISA nivo definiše vezu između kompjulera i hardvera. Njegov jezik moraju da razumeju obe strane.

U lekciji su dati primeri procesora **Core i7 ISA modela i instrukcija, registri, tipovi podataka i formati instrukcija.**

▼ Poglavlje 1

Pregled nivoa ISA

PROGRAMERSKI POGLED NA ARHITEKTURU MIRKOPROCESORA

Program se sastoji od niza instrukcija, od kojih svaka ukazuje na jednu elementarnu operaciju, a sve zajedno definišu izračunavanja koja računar treba da obavi.

Program sastavlja čovek (programer) i zajedno sa ulaznim podacima smešta ih u memoriju računara. CPU pribavlja iz memorije instrukcije i podatke, i izvršava instrukcije, transformišući tako ulazne podatke u izlazne, koje, po okončanju rada, predaje okruženju.

CPU jedinica upravlja radom računara tako što izvršava programske instrukcije. CPU izvršava program instrukciju-po-instrukciju pri čemu izvršenje svake pojedinačne instrukcije odgovara jednom instrukcijskom ciklusu.

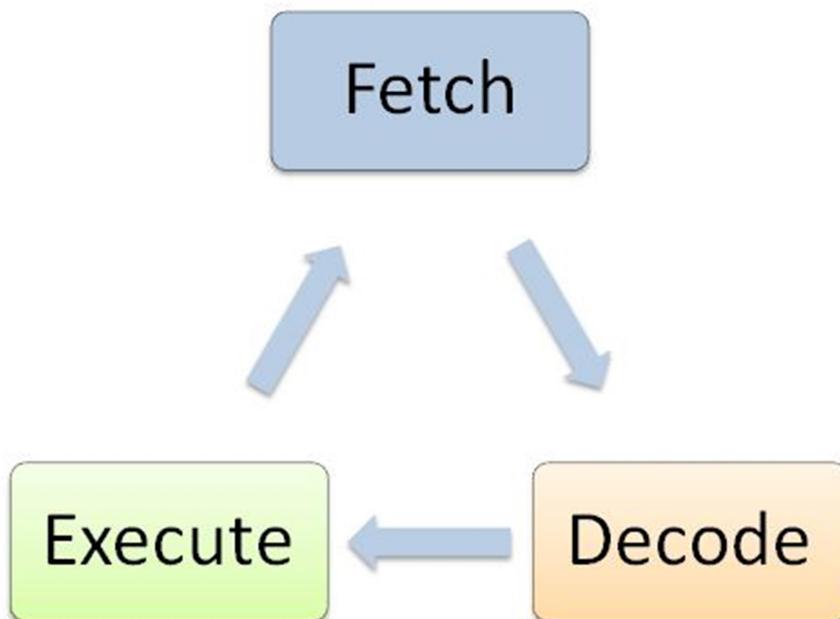
Svaki instrukcijski ciklus uključuje tri glavne aktivnosti (ili faze):

- **Pribavljanje instrukcije,**
- **Dekodiranje instrukcije,**
- **Izvršenje instrukcije.**

Pribavljanje instrukcije odgovara **čitanju instrukcije iz programske memorije**. Da bi se instrukcija izvršila potrebno je obaviti neku specifičnu sekvencu operacija.

Kada CPU dekodira instrukciju, on zapravo utvrđuje o kojoj se instrukciji radi, kako bi izabrao korektnu sekvencu operacija koju treba obaviti.

Procedura koja opisuje rad CPU jedinice u toku svakog instrukcijskog ciklusa zove se algoritam „pribavi-izvrši“ (en. Fetch-and-Execute). Odgovara CISC (en. Complex Instruction Set Computer) modelu CPU jedinice.



Slika 1.1 Algoritam "pribavi - izvrši" [Izvor: Autor]

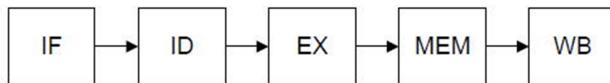
PROGRAMERSKI POGLED NA ARHITEKTURU MIRKOPROCESORA - RISC MODEL

Za razliku od **CISC modela**, **RISC model** CPU jedinice propisuje donekle drugačiji oblik algoritma pribavi-izvrši.

Za razliku od **CISC modela**, **RISC model** CPU jedinice propisuje donekle drugačiji oblik algoritma pribavi-izvrši koji se sastoji iz 5 faza:

1. **Pribavi tekuću instrukciju iz memorije sa adresu na koju ukazuje programski brojač i uvećaj programski brojač za 1.** (**Instruction Fetch - IF**).
2. **Dekodiraj instrukciju i pribavi (prikupi) operande specificirane instrukcijom.** Operandi se pribavljaju iz registarskog fajla. (**Instruction Decoding - ID**).
3. **Izvrši operaciju specificiranu instrukcijom.** (**Execution - EX**).
4. **Ako se radi o instrukciji za pristup memoriji (Load/Store), pristupi eksternoj memoriji radi upisa/čitanja podatka.** (**Memory Access - MEM**).
5. **Upiši rezultat operacije u odredišni registar registarskog fajla.** (**Write Back - WB**).

Kod protočnog izvršenja instrukcija, svi stepeni su upošljeni u svakom mašinskom ciklusu, tako što svaki stepen radi na izvršenju različite instrukcije.



Instrukcijski ciklus	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
n	IF	ID	EX	MEM	WB					
n+1		IF	ID	EX	MEM	WB				
n+2			IF	ID	EX	MEM	WB			
n+3				IF	ID	EX	MEM	WB		
n+4					IF	ID	EX	MEM	WB	
n+5						IF	ID	EX	MEM	WB

Slika 1.2 Protočno izvršenje instrukcija [Izvor: Autor]

PROTOČNO II SUPERSKALARNO IZVRŠENJE INSTRUKCIJA

Kod protočnog izvršenja instrukcija, svi stepeni su upošljeni u svakom mašinskom ciklusu, tako što svaki stepen radi na izvršenju različite instrukcije.

Situacije kada je izvršenje jedne instrukcije privremeno zaustavljeno zato što operandi koje ona zahteva još uvek nisu izračunati od strane prethodne instrukcije nazivaju se hazardi.

Primer:

- n+1) $A=B+C$
- n+2) $D=A \cdot E$
- n+3) $F=G \cdot H$

Instruction	1	2	3	4	5	6	7	8	9	10	11
n	IF	ID	EX	MEM	WB						
n+1		IF	ID	EX		MEM	WB				
N+2			IF	XXX	XXX	XXX	ID	EX	MEM	WB	
N+4							IF	ID	EX	MEM	WB

Slika 1.3 Prikaz kada je izvršenje jedne instrukcije privremeno zaustavljeno [Izvor: Autor]

Dodatno ubrzanje izvršenja programa postiže se tehnikom koja se zove superskalarno izvršenje .

U ovom slučaju CPU istovremeno pribavlja i paralelno izvršava dve ili više instrukcija.

Instrukcijski ciklus	t1	t2	t3	t4	t5	t6	t7
n	IF	ID	EX	MEM	WB		
n+1	IF	ID	EX	MEM	WB		
n+2		IF	ID	EX	MEM	WB	
n+3		IF	ID	EX	MEM	WB	
n+4			IF	ID	EX	MEM	WB
n+5			IF	ID	EX	MEM	WB

Slika 1.4 Superskalarni izvršenje instrukcija [Izvor: Autor]

NIVOI ISA

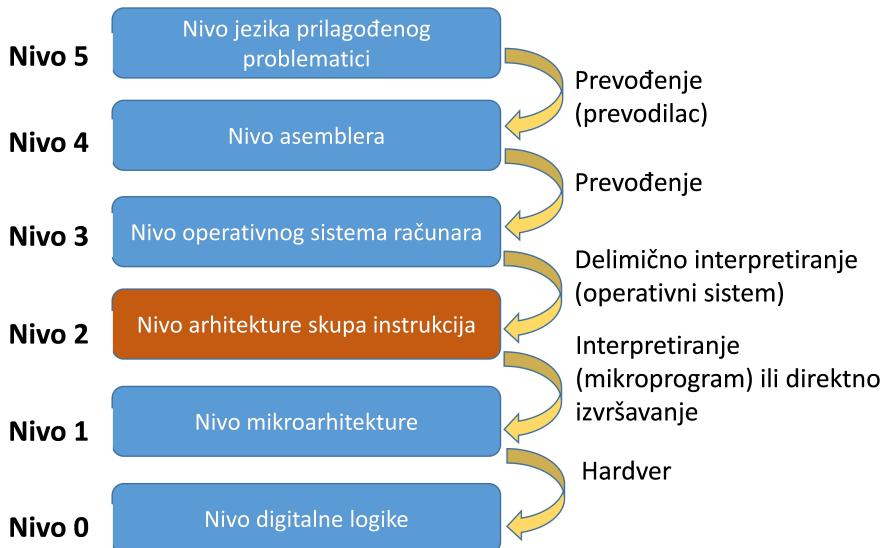
ISA (Instruction Set Architecture) model predstavlja skup instrukcija i formata koji su podržani od strane mikroprocesora i dostupni programeru za razvoj softvera.

ISA model obuhvata tipove instrukcija koje su podržane, kao i format koji će biti korišćen za kodiranje tih instrukcija. ISA model takođe opisuje kako će mikroprocesor obrađivati i izvršavati instrukcije, kao i druge funkcije koje su podržane.

Polazeći od činjenice da računarski program faktički prikazuje – računarskim sredstvima izražen algoritam rešenja nekog problema može se videti da neke od operacija koje su specificirane algoritmom mogu da se realizuju direktno na nivou hardvera, a neke moraju da budu predmet programiranja, tj. moraju se realizovati u vidu određenog softverskog modula.

Takođe, polazeći od tipova i struktura podataka koji se obrađuju na nivou računarskog programa može se uočiti da se neki od tih podataka mogu direktno predstaviti na nivou hardvera i shodno tome direktno obrađivati.

Mnogi podaci zahtevaju realizaciju softverskih modula da bi mogli da budu obrađeni u skladu sa definisanim algoritmom. Iz ovoga sledi da između **HARDVERA** i **SOFTVERA** postoji granica, koju određuju skup operacija i podataka koji se direktno mogu realizovati na nivou hardvera.



Slika 1.5 Računar sa šest nivoa i načinom izvršavanja [Izvor:Autor]

Nivo Arhitekture skupa instrukcija (ISA) nalazi se između nivoa mikroarhitekture i nivoa operativnog sistema i definiše vezu između programskih prevodilaca i hardvera. Njegov jezik moraju da razumeju obe strane.

SVOJSTVA NIVOA ISA

Arhitektura skupa instrukcija (ISA) najčešće se bavi najnižim slojem softverske i najvišim slojem hardverske hijerarhije.

Fokus računarskih arhitektura je u okolini tanke linije koja predstavlja granicu između hardvera i softvera. Tu se nalazi skup instrukcija koji određuje osnovne operacije procesora, sa jedne strane i glavne komponente procesora koje razumiju te instrukcije, sa druge strane.

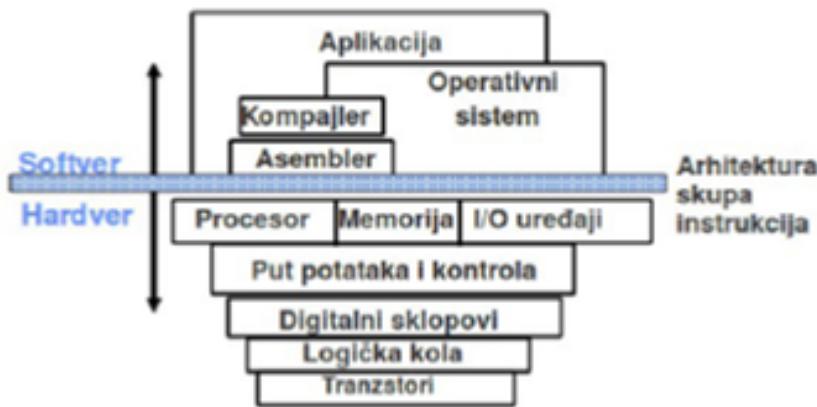
Programer vidi procesor kroz skup instrukcija, dok dizajner hardvera vidi softver kroz sekvencu mašinskih instrukcija koje treba **interpretirati** (izvršiti).

Sa obe strane navedene tanke linije se nalazi više slojeva apstrakcija -kako u softveru, tako i u hardveru.

Arhitektura računara je definisana interfejsom između softvera i hardvera i, kao nauka, najčešće se bavi najnižim slojem softverske i najvišim slojem hardverske hijerarhije apstrakcija - arhitekturom ISA skupa instrukcija i mikroarhitekturom hardvera procesora.

ISA se odnosi na najniži nivo apstrakcije vidljiv programeru.

Mikroarhitekturom se bavi dizajner hardvera i njen zadatak je da se brine o logičkom povezivanju instrukcija i hardverskih modula.



Slika 1.6 Položaj ISA nivoa između hardvera i softvera [Izvor: Autor]

Na vrhu su aplikativni programi koji se izvršavaju na procesoru pomoću sistemskog softvera. Kompajler prevodi program iz jezika visokog nivoa u mašinski kod, dok operativni sistem upravlja resursima i omogućava rad kompajliranje, punjenje i izvršavanje programa.

Ispod ISA nivoa se nalazi dizajn centralne procesne jedinice (CPU), a na nižem nivou logički dizajn sklopova.

Na dnu je fizički dizajn sa razmeštajem komponenti na silicijumskoj pločici.

MEMORIJSKI MODELI

ISA nivo obezbeđuje dva režima rada

Važno svojstvo nivoa ISA jeste da on većini računara obezbeđuje dva radna režima.

Rezim rada jezgra (engl. **kernel mode**) namenjen je izvršavanju operativnog sistema i u njemu se mogu izvršavati sve instrukcije i

Korisnički režim rada (engl. **user mode**) namenjen je izvršavanju aplikacija i u njemu nije dozvoljeno izvršavanje određenih osetljivih instrukcija (kao što je direktno manipulisanje kešom).

Mesto u memoriji gde se nalaze podaci, tj. odgovarajuća adresa povezuju operacije i podatke koji se obrađuju tako da način kako se pristupa podacima u memoriji takođe utiče na definisanje granice između hardvera i softvera.

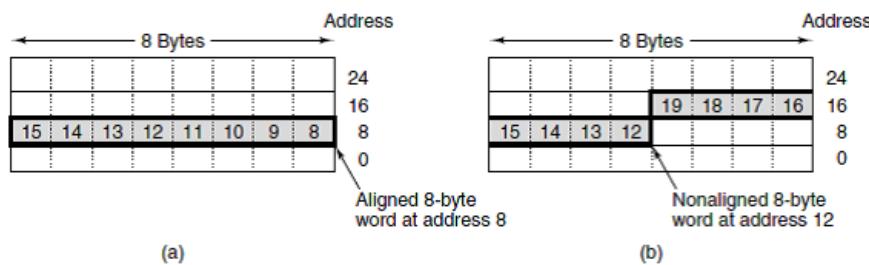
Imajući sve ovo u vidu osnovne elemente instrukcijske arhitekture računara čine:

1. **Operacije** koje se direktno mogu izvršavati na nivou hardvera računara;
2. **Tipovi i strukture podataka** koje hardver računara direktno podržava, t.j. omogućava njihovu obradu u izvornom obliku;
3. **Načini adresiranja informacija** u memoriji računara.

Na svim računarima memorija je podeljena u ćelije sa uzastopnim adresama. Trenutno je uobičajena veličina ćelije 8 bitova, ali su u prošlosti korišćene ćelije veličine od jednog, pa do 60 bitova.

Osmobitna ćelija zove se bajt(engl. byte).

Bajtovi se grupišu u četvorobajtne (32-bitne) ili osmobajtne (64-bitne) reči pomoću instrukcija koje: manipulišu čitavim rečima. U mnogim arhitekturama neophodno je da reči budu poravnate na svojim prirodnim granicama. tako na primer: četvorobajtna reč može da počne na adresama 0, 4, 8 itd... ali ne i na adresi 1 ili 2. Slično tome osmobajtna reč može da počne na adresama 0, 8 ili 16, ali ne i na adresama 4 ili 6. Poravnavanje osmobajtnih reči prikazano je na Slici 3.



Slika 1.7 Osmobajtna reč u memoriji a) Poravnata i b) Neporavnata [Izvor: Autor]

MEMORIJSKI MODELI NA NIVOU ISA

Na nivou ISA računari imaju jedinstven linearan adresni prostor.

Računari po pravilu imaju jedinstven linearan adresni prostor na nivou ISA, koji se proteže od adrese 0 do nekog maksimuma (često 2^{32} bajtova ili 2^{64} bajtova).

Mali broj računara ima zasebne adresne prostore za instrukcije i podatke, pa se preuzimanje instrukcije sa adrese 8 obavlja u jednom adresnom prostoru, a preuzimanje podataka sa adrese 8 u drugom. Ova šema je složenija od jedinstvenog adresnog prostora, ali ona ima dve prednosti.

Prvo, tako je moguće imati 2^{32} bajtova programa i 2^{32} bajtova podataka a da adrese i dalje ostanu 32-bitne.

Drugo, pošto svako upisivanje autornatski ide u prostor za podatke, onemogućuje se slučajno brisanje delova programa što je jedan od izvora programske grešaka.

Potrebno je obratiti pažnju da dva adresna prostora -- jedan za instrukcije i drugi za podatke nisu isto što i podeljeni kešprvog nivoa.

U prvom slučaju ukupan broj adresa se udvostručava i čitanje sa bilo koje adrese daje drugačiji rezultat zavisno od toga da li se čitaju podaci ili instrukcije. U podeljenom kešu adresni prostor je i dalje jedinstven, samo različiti keševi smeštaju podatke u njegove različite delove.

Drugi aspekt memoriskog modela na nivou ISA vise je semantičke prirode. Sasvim je logično očekivati da ćete instrukcijom **LOAD** koja se nalazi iza instrukcije **STORE** na istoj adresi, učitati upravo smeštenu vrednost.

Međutim u mnogim slučajevima mikro-instrukcije se izvršavaju preko reda. Otuda postaje sasvim realno da se memorija neće ponašati na očekivani način.

Taj problem se još više zaoštrava kod multi-procesora gde svaki od više procesora šalje tok (preuređenih) zahteva za čitanje i upisivanje istoj deljenoj memoriji.

Projektanti sistema mogu da pristupe ovom problemu na više načina.

U jednom slučaju svi memoriski zahtevi mogu se staviti u niz, tako da se svaki mora završiti pre nego što se izda sledeći zahtev.

U drugom slučaju da bi memoriju "doveo u red", program mora da izvrši instrukciju **SYNC** koja ne dozvoljava izdavanje nijedne nove instrukcije za rad s memorijom dok se ne završe sve prethodno započete memoriske operacije.

✓ Poglavlje 2

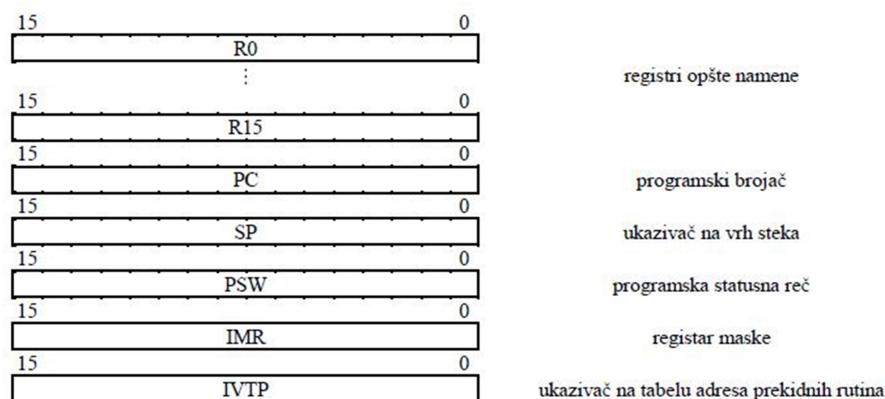
Registri

ARHITEKTURA PROCESORA

Arhitekturu procesora možemo posmatrati kao skup registara, podataka, instrukcija, ...

Arhitekturu procesora čine:

- programski dostupni registri,
- tipovi podataka,
- formati instrukcija,
- načini adresiranja,
- skup instrukcija i
- mehanizam prekida.



Slika 2.1 prikaz registara [Izvor: Autor]

Registri procesora

- programski brojač - **PC** (Program Counter) - sadrži adresu naredne instrukcije, inkrementirajući brojački registar
- adresni registar memorije - **MAR** (Memory Address Register) - sadrži adresu mem. lok. kojoj treba pristupiti (upis/čitanje), vodi se na adresne linije memorije
- prihvatni registar podatka - **MDR** (Memory Data Register) - sadrži podatak pročitan iz mem.lok. sa izlaznih linija podataka memorije ili podatak za upis koji se vodi na ulazne linije podataka memorije

- prihvatni registar instrukcije – **IR** (**Instruction Register**)
sadrži instrukciju, tj. sadržaj **MDR** koji je pročitan iz mem.lok. čija je adresa u **PC**
- prihvatni registri izvorišnih operanada (**A,B**)
vezani na ulazne linije podataka **ALU**
- prihvatni registar rezultata (**C**)
vezan na izlaz **ALU**, sadržaj ovog registra se vodi u **MDR**

INTERNI REGISTRI

U svakom procesoru postoji skup registara na nivou ISA, koji služe za privremeno čuvanje malih količina podataka neophodnih za pravilan rad procesora.

Skup procesorskih registara definiše projektant procesora tokom procesa projektovanja.
Prema mogućnostima pristupa, procesorski registri se mogu svrstati u dve grupe:

- **interni registri i**
- **programski dostupni registri.**

Interni registri su registri procesora kojima se ne može programski pristupati.

Ovim registrima se pristupa samo iz ugrađenih algoritama po kojima se instrukcija izvršava, pa stoga pripadaju organizaciji računara. Služe za čuvanje sadržaja u različitim fazama izvršavanja neke instrukcije, a njihov sadržaj se može koristiti samo u okviru te instrukcije.

Interni registri obuhvataju:

1. **kontrolni registri i**
2. **statusni registar.**

Kontrolni registri

Tokom izvršavanja instrukcije, za smeštaj sadržaja koji se prenosi između memorije i procesora, koriste se sledeći kontrolni registri:

- **programski brojač (PC)** – sadrži adresu naredne instrukcije u memoriji
- **instrukcijski registar (IR)** – sadrži instrukciju
- **adresni registar memorije (MAR)** – sadrži adresu memorijске lokacije kojoj se pristupa
- **registar podatka memorije (MDR)** – sadrži podatak pročitan iz memorije ili podatak koji treba upisati u memoriju

Treba napomenuti da se PC obično inkrementira implicitno, nakon svake instrukcije, ali se može postaviti i eksplicitno instrukcijama skoka. To znači da mu se može pristupiti i programski, tj. da se u slučaju skokova ponaša kao programski dostupan registar.

Da bi bili procesirani, podaci se moraju dovesti na ulaze aritmetičko-logičke jedinice. **MDR** može direktno da bude priključen na **ALU**.

STATUSNI REGISTAR

Nivo mikroarhitekture sastoji se od skupa registara koji kordiniraju jedinice računarskog sistema.

Statusni registar Statusni registar sadrži određen broj indikatora, tj. flegova kojima se opisuje trenutno stanje procesora.

Indikatori odgovaraju pojedinačnim bitovima u registru.

Postavljaju se nezavisno jedan od drugog i predstavljaju različite statusne informacije. **Često se statusni registar naziva PSW** (Program Status Word) registrom, što je preuzeto iz IBM arhitekture.

Indikatori se mogu svrstati u dve grupe:

1. **statusni indikatori**
2. **upravljački indikatori**

Upravljački indikatori se postavljaju softverski tokom izvršavanja posebnih instrukcija programa, a proveravaju se hardverski. Zbog upravljačkih indikatora, može se reći da je statusni registar delom programske dostupan.

Za mehanizam prekida, značajan upravljački indikator je **I** (**Interrupt flag**) koji se postavlja na 1 ako su dozvoljeni maskirajući prekidi.

Bitovi upravljačkog karaktera se postavljaju softverski kao rezultat izvršavanja posebnih instrukcija, a proveravaju hardverski u saglasnosti sa algoritmom izvršavanja instrukcija.

Statusni indikatori se postavljaju hardverski na osnovu rezultata dobijenog izvršavanjem instrukcije, a proveravaju se softverski u instrukcijama uslovnog skoka. Uobičajeni statusni indikatori mogu biti:

Indikator	Opis
<i>N (negative flag)</i>	postavlja se na 1 ako je rezultat operacije negativan
<i>Z (zero flag)</i>	postavlja se na 1 ako je rezultat operacije 0
<i>C (carry flag)</i>	postavlja se na 1 ako ima prenosa ili pozajmice pri aritmetičkim operacijama nad neoznačenim veličinama
<i>V (overflow flag)</i>	postavlja se na 1 ako ima prekoračenja pri aritmetičkim operacijama nad celobrojnim veličinama sa znakom

Slika 2.2 Statusni indikatori [Izvor:Autor]

Bitovi statusnog karaktera N, Z, C i V, se postavljaju hardverski na osnovu rezultata izvršavanja instrukcija, a proveravaju softverski instrukcijama uslovnog skoka.



Slika 2.3 Struktura registra PSW. [Izvor:Autor]

L1, L0, koji binarnim vrednostima od 00 do 11 određuju nivo prioriteta tekućeg programa. E i P indikatori upravljačkog karaktera.

PROGRAMSKI DOSTUPNI REGISTRI

Programski dostupan registar je registar procesora u koji se programskim putem, tj. prilikom izvršavanja instrukcije, može upisati/čitati sadržaj.

Programski dostupni registri procesora su registri u koje je moguće programskom putem izvršavanjem instrukcija procesora upisivati vrednosti i iz kojih je moguće programskom putem izvršavanjem instrukcija procesora očitavati vrednosti.

Vrednost koja se u neki od ovih registara upisuju u nekom od koraka izvršavanja instrukcije koriste se u nekom ili nekim kasnijim koracima izvršavanja te iste instrukcije, ali ne neke od sledećih instrukcija.

Funkcije i broj programske dostupnih registara se razlikuje od procesora do procesora.

Programski dostupni registri koji se najčešće sreću kod komercijalno raspoloživih procesora su:

- programski brojač **PC**,
- registri podataka **DR**,
- adresni registri **AR**,
- bazni registar **BR**,
- indeksni registri **XR**,
- registri opšte namene **GPR**,
- programska statusna reč **PSW**,
- ukazivač na vrh steka **SP** i
- ukazivač na okvir steka **FP**.

Upisivanje vrednosti u ove registre i čitanje vrednosti iz ovih registara je nemoguće specificirati instrukcijama. To je određeno usvojenim algoritmima izvršavanja svake instrukcije posebno.

Ovim registrima se može pristupati na dva načina:

- **eksplicitno**, specificiranjem njegove adrese u adresnom polju instrukcije
- **implicitno**, izborom koda operacije instrukcije koji automatski ukazuje na određeni registar

FUNKCIJE I BROJ PROGRAMSKI DOSTUPNIH REGISTARA

Funkcije i broj programske dostupne registre se razlikuju od procesora do procesora.

Registri podataka - **DR** služe samo za čuvanje podataka i ne mogu se koristiti pri računanjima adresa operanada.

Procesor ovim podacima pristupa znatno brže nego podacima koji se nalaze u memoriji, jer je pristup memoriji skoro za red veličine sporiji. U registre podataka se obično smeštaju među rezultati obrade, kao i podaci koji se više puta koriste prilikom nekih izračunavanja.

Razlozi za uvođenje registara podataka su:

- **sekvencijalna priroda obrade, pri kojoj se često rezultat jedne operacije koristi kao ulazni podatak za narednu operaciju (rezultat prve operacije se smešta u DR, a ne u memoriju, pa mu se pri izvršavanju naredne operacije brže pristupa)**
- **pri obradi je velika verovatnoća da, ako se jednom pristupi nekom podatku, isti podatak bude uskoro opet korišćen**

Adresni registri - **AR** omogućavaju brže generisanje adresa, jer se tokom izvršavanja programa adrese (ili njihovi delovi) uzimaju iz ovih registara, a ne iz memorije, što bi bilo znatno sporije.

Adresni registri mogu biti opšteg karaktera, ali mogu biti i specijalizovani u zavisnosti od primjenjenog načina (moda) adresiranja mogu biti i specijalizovani u zavisnosti od primjenjenog načina (moda) adresiranja. Specijalizovani adresni registri su, na primer, bazni i indeksni registri.

Bazni registri - **BR** se koriste kod baznog i bazno-indeksnog adresiranja. Zbir sadržaja specificiranog baznog registra i pomeraja kod baznog adresiranja, odnosno baznog registra, indeksnog registra i pomeraja kod bazno-indeksnog adresiranja, predstavlja adresu memorijske lokacije na kojoj se nalazi izvorišni ili odredišni operand.

Indeksni registri - **XR** se koriste kod indeksnog i bazno-indeksnog adresiranja. Zbir sadržaja specificiranog indeksnog registra i pomeraja kod indeksnog adresiranja, odnosno baznog registra, indeksnog registra i pomeraja kod bazno-indeksnog adresiranja, predstavlja adresu memorijske lokacije na kojoj se nalazi izvorišni ili odredišni operand.

Registri opšte namene - **GPR** se koriste na isti način kao registri podataka, adresni registri, bazni registri i indeksni registri. Registri opšte namene GPR se javljaju kod onih procesora kod kojih ne postoje posebni registri podataka, adresni registri, bazni registri i indeksni registri.

Za razliku od procesora sa posebnim registrima podataka, adresnim registrima, baznim registrima i indeksnim registrima, gde se registri iz svake grupe registara koriste samo uz odgovarajuća adresiranja, procesori sa registrima opšte namene mogu da koriste bilo koji od registara uz bilo koje adresiranje.

✓ Poglavlje 3

Formati instrukcija mikroprocesora

FORMATI INSTRUKCIJA

Formatom instrukcije procesora: OPERACIJA opedand1, operand2

Formatom instrukcije se specificiraju dve vrste informacija neophodne za izvršavanje instrukcija programa i to:

- **operacija i tip podatka sa kojim operacija treba da se realizuje i**
- **izvorišni i odredišni operandi.**

Za izvršavanje instrukcija programa treba da se zna i odakle uzeti sledeću instrukciju po završetku izvršavanja tekuće instrukcije.

U svim daljim razmatranjima prepostavice se danas uobičajeni pristup da je to određeno tekućom vrednošću programske memorije, koju se označava sa **PC**.

Kod ovog pristupa kod čitanja instrukcije kao adresa memorije uvek se koristi tekuća vrednost programske memorije, koja se automatski inkrementira.

Kao rezultat ovakvog pristupa instrukcije se mogu jedino sekvencialno čitati i izvršavati.

To ima za posledicu da u nekim situacijama ne treba preći na čitanje i izvršavanje prve sledeće instrukcije u sekvenci, već na neku instrukciju van sekvenca, što je moguće postići jedino ako se u programske memorije upiše adresa memorije sa koje treba produžiti sa čitanjem i izvršavanjem instrukcija programa.

Šta je arhitektura skupa instrukcija?

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poljem za izvorišni i odredišni operand se eksplisitno specificiraju operandi. Postoje više varijanti ovog polja koje nastaju kao posledica sledeća dva elementa:

- broj eksplisitno specificiranih operanada,
- moguće lokacije operanada.

POLJA INSTRUKCIJA ZA OPERACIJU I TIP PODATKA

Informacije i operacije u tipu podatka sa kojim operacija treba da se realizuje i izvorišnim i odredišnim operandima se specificiraju odgovarajućim poljima instrukcije.

Danas je uobičajen pristup da u formatu instrukcije ne postoji informacija o sledećoj instrukciji već da se tokom sekvenčnog izvršavanja instrukcija implicitno kao adresa sledeće instrukcije koristi tekuća vrednost programskog brojača koja se inkrementira posle svakog čitanja instrukcije, a da se kada treba odstupiti od sekvenčnog izvršavanja instrukcija eksplisitnom instrukcijom skoka vrši upis adrese sledeće instrukcije u programski brojač **PC**.

Interesantno je da se u mikroprogramima kod mikroprogramskih realizacija upravljačkih jedinica skokovi u mikroprogramu javljaju u proseku posle svake druge ili treće mikroinstrukcije, pa je stoga dosta uobičajeno da u formatu mikroinstrukcije pored dela kojim se specificira koje mikrooperacije treba da se realizuju postoji i deo sa adresom mikroinstrukcije na koju treba preći u slučaju da se odstupa od sekvenčnog izvršavanja mikro-instrukcija. Informacije o operaciji i tipu podatka sa kojim operacija treba da se realizuje i izvorišnim i odredišnim operandima se specificiraju odgovarajućim poljima instrukcije.

U zavisnosti od toga kako se ove dve vrste informacija specificiraju, zavisi kakve je struktura polja u formatu instrukcije. Na osnovu toga se govori o različitim formatim instrukcija. U daljim razmatranjima se daju funkcije i struktura najpre polja sa specifikacijom operanda i tipa operacije, a zatim i polja sa specifikacijom izvorišnih i odredišnih operanada.

U daljim razmatranjima se daju funkcije i struktura najpre polja sa specifikacijom operanda i tipa operacije, a zatim i polja sa specifikacijom izvorišnih i odredišnih operanada.

OPERACIJA I TIP PODATKA

Ovim poljem, koje se obično naziva polje koda operacije, se specificira operacija koju treba izvršiti i tip podatka nad kojim datu operaciju treba izvršiti. Pod operacijom se misli na operacije iz skupa instrukcija, kao što su operacije prenosa, aritmetičke operacije, logičke operacije, operacije pomeranja i rotiranja i upravljačke operacije.

Pod tipom podatka se misli iz koliko susednih memorijskih lokacija treba pročitati binarne i na koji, od više mogućih načina, ih treba interpretirati. Procesor treba tako realizovati da se na osnovu vrednosti koda operacije utvrđuje:

- iz koliko susednih memorijskih lokacija, počev od one koja je specificirana poljem sa specifikacijom izvorišnih i odredišnih operanada, treba očitati memorijskih reči da bi se dobio operand potrebne dužine,
- kako očitane binarne vrednosti treba interpretirati i
- koju operaciju treba realizovati.

IZVORIŠNI I ODREDIŠNI OPERANDI

U slučaju binarnih operacija, kao što su aritmetičke i logičke operacije, potrebne su dve izvorišne i jedna odredišna lokacija.

Kao ilustracija ovog pristupa može se uzeti da u procesoru koji od tipova podatak podržava celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita, celobrojne veličine sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita i veličine u pokretnom zarezu dužine 32 i 64 bita mora da postoji 10 kodova operacija za operaciju množenja.

U slučaju binarnih operacija, kao što su aritmetičke i logičke operacije, potrebne su dve izvorišne i jedna odredišna lokacija .

U zavisnosti od toga koliko se lokacija eksplisitno definiše u formatu instrukcije, procesori se dele na **troadresne, dvoadresne, jednoadresne i nulaadresne** procesore.

Kod procesora kod koji se u formatu instrukcije eksplisitno ne definišu sve tri lokacije, za one lokacije kojih nema eksplisitno definisanih u formatu instrukcije zna se implicitno gde su. To znači da je za izvršenje najvećeg broja operacija na ulaz operacione jedinice treba dovesti dva ulazna operanda i specificirati adresu memorijске lokacije u koju se smešta rezultat operacije. U opštem slučaju ovo se najlakše obezbeđuje preko opšteg formata instrukcije koji je prikazan na Slici.



Slika 3.1 Opšti format instrukcije [Izvor: Autor]

OC: Specificira operaciju

SRC1, SRC2: adrese memorijskih lokacija (mem.lok.) u kojima se nalaze operandi (izvorišni operandi)

DST: adresa memorijске lokacije u koju treba smestiti rezultat odredišni operand.

Iako ovaj format instrukcije pruža najviše slobode u zadavanju svih parametara jedne instrukcije on sa sobom nosi i određene nedostatke.

Pre svega to je dužina takve instrukcije, jer ako se želi obezrediti adresiranje na nivou pojedinačne memorijске lokacije onda dužina adresnog polja mora da bude jednak dužini adresne reči računara.

TROADRESNE INSTRUKCIJE

Osim koda operacije OC, u troadresnom formatu se nalaze još tri adresna polja za operative koji su eksplisitno definisani.

Kod procesora sa troadresnim formatom instrukcija (Slika) eksplisitno su poljima A1, A2 i A3 definisane sve tri lokacije.

Kod nekih procesora polja A1 i A2 definišu adrese izvorišnih lokacija, a A3 adresu odredišne lokacije. Međutim, ima procesora kod kojih polja A2 i A3 definišu adrese izvorišnih lokacija, a A1 adresu odredišne lokacije.

Izvršavanje instrukcije u troadresnom procesoru biće prikazano na primeru izračunavanja izraza

A + B = C.

Za računanje ovog izraza je dovoljna samo jedna instrukcija:

ADD A, B, C

Data instrukcija se izvršava tako što se izvorišni operandi, koji se nalaze u memorijskim lokacijama čije su adrese A i B, sabiju i dobijeni zbir smesti u memorijsku lokaciju čija je adresa C.

Operacija sabiranja definisana je kodom operacije **ADD**.

OC	A1	A2	A3
----	----	----	----

Slika 3.2 Troadresni format instrukcije. [Izvor: Autor]

U poljima namenjenim izvorišnim operandima, umesto adresa mogu da se nalaze i sami operandi. U polju koje odgovara odredišnom operandu uvek mora da bude adresa, jer rezultat operacije nije unapred poznat.

Dobra strana ovog formata je u tome što se binarna operacija (nad dva operanda) izvršava jednom instrukcijom.

Loša strana formata je u velikoj dužini instrukcije.

DVOADRESNE INSTRUKCIJE

Ovaj format se interpretira tako što se smatra da oba adresna polja predstavljaju izvorišne operative.

Lokacija odredišnog operanda se zadaje implicitno kao jedna od izvorišnih lokacija (za svaki procesor se definiše koja je to lokacija).

Na primer, može se uzeti da se rezultat operacije smešta u memorijsku ćeliju sa adresom prvog izvořišnog operanda. To znači da se pre izvršenja instrukcije u datoj ćeliji nalazi izvořišni operand, a nakon izvršenja instrukcije odredišni operand (izvořišni operand se gubi).

U poređenju sa troadresnim formatom, **dobra strana dvoadresnog** formata je u tome što je instrukcija kraća, **loša strana** u tome što je potrebno više instrukcija za izračunavanje izraza.

Povećanje broja instrukcija biće ilustrovano na ranijem primeru izračunavanja izraza:

$$\mathbf{A + B = C.}$$

Kod dvoadresnih procesora, za računanje ovog izraza su potrebne dve instrukcije:

MOV A,C

ADD C,B

Kod operacije **MOV** definiše operaciju prenosa podatka, a kod operacije **ADD**, operaciju sabiranja.

Neka u datom procesoru lokacija odredišnog operanda prilikom sabiranja odgovara adresi prvog izvořišnog operanda.

Prva instrukcija se izvršava tako što se sadržaj lokacije na adresi A pročita i smesti u lokaciju sa adresom C. Ovo je bilo potrebno uraditi da bi se za drugu instrukciju obezbedilo da prvi izvořišni operand bude na odgovarajućem mestu.

U drugoj instrukciji, izvořišni operandi iz lokacija sa adresama C i B se sabiraju i rezultat smešta u lokaciju sa adresom C.



Slika 3.3 Dvoadresni format instrukcije [Izvor: Autor]

JEDNOADRESNE INSTRUKCIJE

U ovom formatu, eksplicitno je definisan samo jedan izvořišni operand, dok su drugi izvořišni operand i odredišni operand implicitno definisani.

Podrazumeva se da se oni čuvaju u posebnom procesorskom registru koji se naziva **akumulator**.

Dobra strana ovog formata je u maloj dužini instrukcije, a

Loša strana je u tome što je potrebno još više instrukcija za izračunavanje izraza nego u slučaju dvoadresnog formata.

Za raniji primer računanja izraza

$$\mathbf{A + B = C,}$$

kod jednoadresnih procesora, potrebne su tri instrukcije:

LOAD A

ADD B

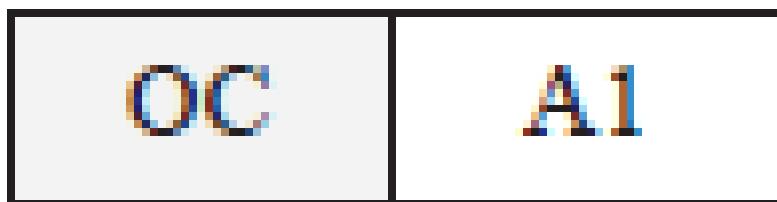
STORE C

Kod operacije **LOAD** definiše operaciju prenosa podatka u akumulator, kod operacije **ADD** operaciju sabiranja, a kod operacije **STORE** operaciju prenosa podatka iz akumulatora.

Prva instrukcija se izvršava tako što se sadržaj lokacije na adresi A pročita i smesti u akumulator.

Druga instrukcija čita sadržaj iz lokacije sa adresom B, sabira ga sa sadržajem akumulatora i rezultat smešta u akumulator.

Treća instrukcija sadržaj akumulatora prenosi u lokaciju sa adresom C.



Slika 3.4 Jednoadresni format instrukcije [Izvor: Autor]

NULAADRESNE INSTRUKCIJE

U ovom formatu, postoji samo polje koda operacije i ne postoji ni jedno polje kojim bi se eksplicitno definisale izvorišne i odredišne lokacije.

Kod ovih procesora vrh steka je implicitno izvorište za oba operanda i implicitno odredište za rezultat.

Izuzetak su dve jednoadresne instrukcije i to instrukcija **PUSH** kojom se sadržaj iz neke memorijske lokacije smešta na vrh steka i instrukcija **POP** kojom se sadržaj sa vrha steka smešta u neku memorijsku lokaciju.

Ukoliko kod procesora sa nulaadresnim formatom instrukcija treba sračunati izraz:

C = A + B

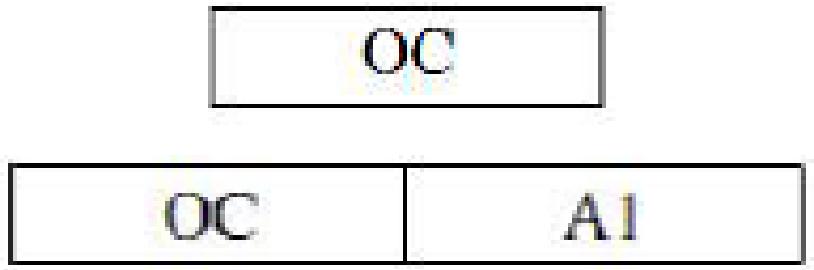
kojim se sabiraju sadržaji memorijskih lokacija čije su adrese simbolički označene sa A i B i rezultat smešta u memorijsku lokaciju čija je adresa simbolički označena sa C, tada se to mora realizovati instrukcijama:

PUSH A ;

PUSH B ;

ADD ;

POP C



Slika 3.5 Nulaadresni format instrukcije [Izvor: Autor]

Simbolički su sa **PUSH, ADD** i **POP** označene vrednosti polja koda operacija prenosa na vrh steka, sabiranja i prenosa sa vrha steka, respektivno, a sa A, B i C adrese eksplisitno definisanih izvorišnih i odredišnih lokacija.

Najpre se tokom izvršavanja instrukcije **PUSH** iz memorijske lokacije sa adresi A čita operand i smešta na vrh steka, potom se tokom izvršavanja instrukcije **PUSH** iz memorijske lokacije sa adresi B čita operand i smešta na vrh steka, zatim se tokom izvršavanja instrukcije sabiranja **ADD** sa vrha steka čitaju dva operanda, izvršava operacija sabiranja i dobijeni rezultat smešta na vrh steka i na kraju se tokom izvršavanja instrukcije **POP** skida rezultat sa vrha steka i smešta u memorijsku lokaciju na adresi C.

NAČINI ADRESIRANJA

Načini adresiranja određuju da li je operand sadržaj neke memorijske lokacije, nekog od registara podataka ili registara opšte namene procesora ili veličina u samoj instrukciji

Načini adresiranja specifičiraju i kako treba formirati adresu memorijske lokacije u koliko je operand sadržaj neke memorijske lokacije.

Postoje:

- **Registarsko direktno adresiranje** je adresiranje kod koga se operand nalazi u jednom od registara podataka ili registara opšte namene procesora.
- **Registarsko indirektno adresiranje** je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se nalazi u jednom od adresnih registara.
- **Memorijsko direktno adresiranje** je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a u adresnom delu instrukcije u kome se specificira operand.

- **Memorijsko indirektno adresiranje** se javlja i kod onih procesora kod kojih postoje posebno ***registri podataka (DR)***, ***adresni registri (AR)***, ***bazni registri (BR)*** i ***indeksni registri (XR)***.
- **Bazno adresiranje sa pomerajem:** operand se nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja baznog registra i pomeraja.
- **Indeksno adresiranje sa pomerajem:** operand se nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja indeknog registra i pomeraja.
- **Bazno- indeksno adresiranje sa pomerajem** je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem registra i pomeraja.

▼ Poglavlje 4

Tipovi instrukcija mikroprocesora

TIPOVI INSTRUKCIJA

Skup instrukcija specificira operacije koje mogu da se izvršavaju u procesoru. Skup instrukcija čine standardne instrukcije i nestandardne instrukcije.

Standardne instrukcije uključuju operacije čijim kombinovanjem svaki problem koji treba da se reši u računaru može da se predstavi programom.

Standardne instrukcije se u nekom vidu nalaze u svakom procesoru.

Skup standardnih instrukcija čine:

- **instrukcije prenosa,**
- **aritmetičke instrukcije,**
- **logičke instrukcije,**
- **instrukcije pomeranja i rotiranja,**
- **instrukcije skoka i**
- **mešovite instrukcije.**

Tipovi instrukcija

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

INSTRUKCIJE PRENOSA

Instrukcije prenosa su instrukcije koje realizuju prenos podatka sa jednog mesta u računaru na drugo.

Podaci se nalaze u memorijskim lokacijama, registrima procesora i u instrukciji. Zbog toga mora da postoje instrukcije prenosa za prebacivanje podataka između lokacija na kojima mogu da se nađu.

Treba napomenuti da podaci mogu da budu i u registrima kontrolera periferija, pri čemu se njima pristupa na dva načina.

Kod nekih procesora registri kontrolera periferija se ne izdvajaju kao posebno mesto gde mogu da budu podaci već se tretiraju na isti način kao i memorijske lokacije, pa ne postoje posebne instrukcije za prenos podatak u i iz registara kontrolera periferija već se registrima kontrolera periferija pristupa istim instrukcijama kojima se pristupa memorijskim lokacijama.

Kod ovih procesora se kaže da je ulazno izlazni adresni prostor memorijski preslikan.

Kod drugih procesora registri kontrolera periferija se izdvajaju kao posebno mesto gde mogu da budu podaci i postoje posebne instrukcije za prenos podatak u i iz registara kontrolera periferija.

Kod ovih procesora se kaže da su ulazno izlazni i memorijski adresni prostori razdvojeni.

Pored toga kod procesora sa jednoadresnim formatom instrukcija akumulator je implicitno izvorište i odredište.

Zato postoji potreba za instrukcijom prenosa kojom bi podatak preneo u akumulator i dalje mogao da koristi kao izvorišni operand.

Akumulator je i implicitno odredište, pa postoji potreba za instrukcijom prenosa kojom bi se podatak prebacio iz akumulatora u željenu lokaciju.

Slična je situacija i kod procesora sa nulaadresnim formatom instrukcija kod kojih je vrh steka implicitno izvorište i odredište.

Zato postoji potreba za instrukcijom prenosa kojom bi podatak preneo na vrh steka i dalje mogao da koristi kao izvorišni operand.

Vrh steka je i implicitno odredište, pa postoji potreba za instrukcijom prenosa kojom bi se podatak prebacio sa vrha steka u željenu lokaciju.

INSTRUKCIJE PRENOSA - MOV

Podaci se nalaze u memorijskim lokacijama, registrima procesora i u instrukciji. Zbog toga mora da postoje instrukcije prenosa za prebacivanje podataka između lokacija.

U daljem tekstu se razmatraju moguće instrukcije prenosa.

MOV a, b

Ova instrukcija se javlja kod procesora sa dvoadresnim formatom instrukcija.

Sa **MOV** je simbolički označeno polje koda operacije, a sa **a** i **b** specifikacije jednog izvorišnog i jednog odredišnog operanda.

U daljim razmatranjima će se uzeti da je a izvorište, a b odredište, mada može da bude i obrnuto.

U zavisnosti od specificiranog načina adresiranja a može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji.

Slična je situacije i sa b, pre čemu, zbog toga što se radi o odredištu, b samo ne može da bude neposredna veličina u instrukciji.

Specificiranjem odgovarajućih adresiranja moguće je realizovati prenose iz memorije (registra kontrolera periferije) u memoriju (registar kontrolera periferije), iz memorije (registra kontrolera periferije) u registar procesora, iz registra procesora u memoriju (registar kontrolera periferije), iz registra procesora u registar procesora, neposredno iz instrukcije u memoriju (registar kontrolera periferije) i neposredno iz instrukcije u registar procesora.

Kod procesora sa troadresnim formatom instrukcija sve je isto, samo što se polje za specifikaciju drugog izvorišta ne koristi.

INSTRUKCIJE PRENOŠA - IN I OUT

Ove instrukcije se javljaju kod procesora sa dvoaddrēsnim formatom instrukcija, ukoliko su ulazno izlazni i memorijski adresni prostori razdvojeni.

IN regper, regproc

OUT regproc, regper

Ove instrukcije se javljaju kod procesora sa dvoaddrēsnim formatom instrukcija, ukoliko su ulazno izlazni i memorijski adresni prostori razdvojeni.

Sa **IN** je simbolički označeno polje koda operacije za prenos iz registra kontrolera periferije u registar procesora, pri čemu je sa regper direktno data adresa registra kontrolera periferije, dok je sa regproc direktno data adresa registra procesora.

Sa **OUT** je simbolički označeno polje koda operacije za prenos iz registra procesora u registar kontrolera periferije, pri čemu regproc i regper imaju identično značenje kao i u slučaju instrukcije **IN**. Uobičajeno je kod ovih instrukcija da su prenosi samo između registara kontrolera periferija i registara procesora i da se njihove adrese direktno daju.

IN regper

OUT regper

Ove instrukcije se javljaju kod procesora sa jednoaddrēsnim formatom instrukcija, ukoliko su ulazno izlazni i memorijski adresni prostori razdvojeni.

Oznake **IN, OUT** i **regper** imaju identično značenje kao i u slučaju procesora sa dvoaddrēsnim formatom instrukcija.

Kod instrukcije IN prenos je iz registra kontrolera periferije čija je adresa direktno data sa **regper** i akumulatora koji je implicitno odredište, dok je kod instrukcije OUT prenos iz akumulatora koji je implicitno izvorište i registra kontrolera periferije čija je adresa direktno data sa **regper**.

INSTRUKCIJE PRENOSA - LOAD, STORE, PUSH, POP

Instrukcije prenosa su: LOAD a, STORE b, PUSH a i POP b.

LOAD a

STORE b

Ove instrukcija se javlja kod procesora sa jednoadresnim formatom instrukcija. Sa **LOAD** i **STORE** je simbolički označeno polje koda operacije, a sa a i b specifikacije izvorišnog i odredišnog operanda, respektivno. Instrukcijom **LOAD** se operand sa izvorišta a prenosi u akumulator kao implicitno odredište, dok se instrukcijom **STORE** sadržaj akumulatora kao implicitno izvorište prenosi u odredište b. U zavisnosti od specificiranog načina adresiranja a može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji.

Slična je situacija i sa b, pre čemu, zbog toga što se radi o odredištu, b samo ne može da bude neposredna veličina u instrukciji. U slučaju instrukcije **LOAD** specificiranjem odgovarajućih adresiranja za izvorište a moguće je u akumulator preneti operand iz memorije (registra kontrolera periferije), iz registra procesora i neposredno iz instrukcije.

U slučaju instrukcije **STORE** specificiranjem odgovarajućih adresiranja za odredište b moguće je iz akumulatora preneti operand u memoriju (registar kontrolera periferije) i registar procesora, dok prenos u instrukciju korišćenjem neposrednog adresiranja nije dozvoljen.

PUSH a

POP b

Ove instrukcija se javlja kod procesora sa nulaadresnim formatom instrukcija. Sa **PUSH** i **POP** je simbolički označeno polje koda operacije, a sa a i b specifikacije izvorišnog i odredišnog operanda, respektivno. Instrukcijom **PUSH** se operand sa izvorišta a prenosi na stek kao implicitno odredište, dok se instrukcijom **POP** sadržaj sa steka kao implicitno izvorište prenosi u odredište b.

✓ 4.1 Aritmetičke instrukcije

ARITMETIČKE INSTRUKCIJE - ADD

Aritmetičkim instrukcijama se realizuju standardne aritmetičke operacije sabiranja, oduzimanja, množenja i deljenja.

Aritmetičkim instrukcijama se realizuju standardne aritmetičke operacije sabiranja, oduzimanja, množenja i deljenja.

ADD a, b, c

Ovo je format instrukcije sabiranja kod procesora sa troadresnim formatom instrukcija. Sa **ADD** je simbolički označeno polje koda operacije, a sa a, b i c specifikacije dva izvorišna i jednog odredišnog operanda. U daljim razmatranjima će se uzeti da su a i b izvorišta, a c odredište, mada se često dešava i da je a odredište, a b i c izvorišta. Tokom izvršavanja ove instrukcije sa lokacija specificiranih sa a i b najpre se čitaju dva izvorišna operanda, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa c.

Aritmetičke instrukcije sa primerima

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

U zavisnosti od nezavisno specificiranih načina adresiranja za a i b, izvorišni operandi mogu da budu bilo koja kombinacija registara procesora, memorijskih lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registara kontrolera periferije) i neposrednih veličina u instrukciji.

Slična je situacija i sa c, pri čemu, zbog toga što se radi o odredištu, c samo ne može da bude neposredna veličina u instrukciji. Neke od mogućih kombinacija su:

- a, b i c su memorijske lokacije (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan a, b i c mogu da budu bilo koja kombinacija stvarnih memorijskih lokacija i registara kontrolera periferije),

- a, b i c su registri procesora,

- a je neposredna veličina, b je registar procesora i c memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije),

ARITMETIČKE INSTRUKCIJE - ADD - VARIJACIJE

Aritmetičke instrukcije su: ADD a, b i ADD a.

ADD a, b

- a je memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije), b je neposredna veličina i c registar procesora, itd.

To predstavlja format instrukcije sabiranja kod procesora sa dvoadresnim formatom instrukcija.

Sa **ADD** je simbolički označeno polje koda operacije, a sa a i b specifikacije dva izvorišna operanda, pri čemu je a ili b i implicitno odredište. U daljim razmatranjima će se uzeti da je a jedno izvorište i implicitno odredište, a b drugo izvorište, mada se često dešava i da je a prvo izvorište, a b drugo izvorište i implicitno odredište.

Tokom izvršavanja ove instrukcije sa lokacija specificiranih sa a i b najpre se čitaju dva izvorišna operanda, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa a.

U zavisnosti od specificiranog načina adresiranja za b, drugi izvorišni operand može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji.

Slična je situacija i sa a, pri čemu, zbog toga što se radi ne samo i prvom izvorištu nego i o implicitnom odredištu, a samo ne može da bude neposredna veličina u instrukciji. Neke od mogućih kombinacija su:

- a i b su memorijske lokacije (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan a i b mogu da budu bilo koja kombinacija stvarnih memorijskih lokacija i registara kontrolera periferije),
- a i b su registri procesora,
- a je registar procesora i b memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije),
- a je memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i b je neposredna veličina, itd.

ADD a

Ovo je format instrukcije sabiranja kod procesora sa jednoadresnim formatom instrukcija.

ARITMETIČKE INSTRUKCIJE SA NULAADRESNIM FORMATOM

Sa ADD je simbolički označeno polje koda operacije, pri čemu je vrh steka implicitno izvorište za oba izvorišna operanda i implicitno odredište.

Tokom izvršavanja ove instrukcije se najpre iz akumulatora implicitno čita prvi izvorišni operand i sa lokacije specificirane sa a se čita drugi izvorišni operand, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta u akumulator kao implicitno odredište.

U zavisnosti od specificiranog načina adresiranja za a, drugi izvorišni operand može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije) i neposredna veličina u instrukciji.

ADD

Ovo je format instrukcije sabiranja kod procesora sa nulaadresnim formatom instrukcija.

Sa **ADD** je simbolički označeno polje koda operacije, pri čemu je vrh steka implicitno izvorište za oba izvorišna operanda i implicitno odredište. Tokom izvršavanja ove instrukcije sa vrha steka se implicitno čita najpre prvi a potom i drugi izvorišni operand, zatim se nad njima realizuje operacija sabiranja i na kraju se dobijeni rezultat smešta na vrh steka kao implicitno odredište.

Slična je situacija i sa formatima instrukcija za operacije oduzimanja, množenja i deljenja.

Formati instrukcija za ove operacije za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

- **SUB a, b, c**
- **SUB a, b**
- **SUB a**
- **SUB**
- **MUL a, b, c**
- **MUL a, b**
- **MUL**
- **DIV a, b, c**
- **DIV a, b**
- **DIV**

ARITMETIČKE INSTRUKCIJE ZA OPERACIJE MNOŽENJA I DELJENJA

Implicitna izvorišta i odredišta u formatima instrukcija za množenje gde ih ima, su, takođe, ista kao i kod instrukcije sabiranja.

- **MUL a, b**
- **MUL a**
- **MUL**
- **DIV a, b, c**
- **DIV a, b**
- **DIV a**
- **DIV**

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a, b i c je ista kao i kod instrukcije sabiranja.

Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takođe, ista kao i kod instrukcije sabiranja.

Podaci nad kojima se izvršavaju operacije sabiranja, oduzimanja, množenja i deljenja mogu da budu predstavljeni na više načina, kao na primer celobrojne veličine bez znaka, celobrojne veličine sa znakom i to obično u drugom komplementu, pokretni zarez, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd.

Za svaki mogući način predstavljanja podataka i određenu dužinu podatka mora da postoji poseban kod operacije za svaku od operacija sabiranja, oduzimanja, množenja i deljenja.

Kao ilustracija može se uzeti da u procesoru, koji od tipova podataka podržava celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita, celobrojne veličine sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita i veličine u pokretnom zarezu dužine 32 i 64 bita, mora da postoji 10 kodova operacija za operaciju množenja.

Isto važi i za operaciju deljenja.

Međutim, kod instrukcija sabiranja i oduzimanja nad celobrojnim veličinama bez znaka i celobrojnim vrednostima sa znakom u drugom komplementu to nije neophodno, jer se istim postupkom sabiranja i oduzimanja binarnih reči određene dužine dobija korektni rezultat i kada se binarne reči interpretiraju kao celobrojne veličine bez znaka i kao celobrojne veličine sa znakom u drugom komplementu.

▼ 4.2 Formati instrukcija

FORMAT INSTRUKCIJE INKREMENTIRANJA - INC

INC a, b je format instrukcije inkrementiranja koji se javlja kod procesora sa dvoadresnim formatom instrukcija.

INC a, b

Ova je format instrukcije inkrementiranja koji se javlja kod procesora sa dvoadresnim formatom instrukcija.

Sa **INC** je simbolički označeno polje koda operacije, a sa a i b specifikacije izvorišnog i odredišnog operanda, respektivno. U daljim razmatranjima će se uzeti da je a izvorište a b odredište, mada se često dešava i da je a odredište a b izvorište.

Tokom izvršavanja ove instrukcije sa lokacije specificirane sa a najpre se čita izvorišni operand, zatim se njegova vrednost uveća za 1 i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa b.

U zavisnosti od specificiranog načina adresiranja za a izvorišni operand može da bude registar procesora, memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registara kontrolera periferije) i neposredna veličina u instrukciji.

Slična je situacija i sa b, pri čemu, zbog toga što se radi o odredištu, b samo ne može da bude neposredna veličina u instrukciji.

INC a

Ovo je format instrukcije inkrementiranja koji se javlja kod procesora sa jednoadresnim formatom instrukcija.

Sa **INC** je simbolički označeno polje koda operacije, a sa a specifikacija izvorišnog i implicitnog odredišnog operanda.

Tokom izvršavanja ove instrukcije sa lokacije specificirane sa a najpre se čita izvorišni operand, zatim se njegova vrednost uveća za 1 i na kraju se dobijeni rezultat smešta u lokaciju specificiranu sa a.

U zavisnosti od specificiranog načina adresiranja za a izvorišni i implicitni odredišni operand može da bude registar procesora i memorijska lokacija (a kod procesora kod kojih je ulazno izlazni prostor memorijski preslikan i registar kontrolera periferije).

Zbog toga što je a ne samo izvorište već i implicitno odredište, a ne može da bude neposredna veličina u instrukciji.

FORMAT INSTRUKCIJE DEKREMENTIRANJA - DEC

DEC je format instrukcije dekrementiranja kod procesora sa nulaadresnim formatom instrukcija.

DEC a, b

DEC a

DEC

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a i b je ista kao i kod instrukcije inkrementiranja.

Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takone, ista kao i kod instrukcije inkrementiranja.

Podaci nad kojima se izvršavaju operacije inkrementiranja i dekrementiranja mogu da budu predstavljeni kao celobrojne veličine bez znaka i celobrojne veličine sa znakom i to obično u drugom komplementu, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd.

Međutim, inkrementiranje se za određenu dužinu operanda realizuje na isti način i nad celobrojnim veličinama bez znaka i nad celobrojnim veličinama sa znakom u drugom komplementu, pa nisu potrebna 2 već samo 1 kod operacije.

Ovo je objašnjeno za instrukciju sabiranja i važi i za instrukciju inkrementiranja kao poseban slučaj instrukcije sabiranja.

Zbog toga različiti kodovi operacije treba da postoje samo za različite dužine operanada.

Kao ilustracija ovog pristupa može se uzeti da u procesoru koji od tipova podataka podržava celobrojne veličine bez znaka dužine 8, 16, 32 i 64 bita i celobrojne veličine sa znakom predstavljene u drugom komplementu dužine 8, 16, 32 i 64 bita mora da postoje 4 operacije za operaciju inkrementiranja. Isto važi i za operaciju dekrementiranja.

FORMATI INTRUKCIJE UPOREĐIVANJA - CMP

Formati instrukcije upoređivanja za procesore sa troadresnim, dvoaдресним, jednoaдресним i nulaadresnim formatima instrukcija su: CMP a, b, c ; CMP a, b ; CMP a i CMP.

Među aritmetičkim instrukcijama se pojavljuje i poseban slučaj operacije oduzimanja kod koje se rezultat oduzimanja nikuda ne upisuje, već se samo vrši provera dobijene vrednosti i na osnovu toga vrši postavljanje indikatora N, Z, C i V u programskoj statusnoj reči PSW procesora.

Ova instrukcija se naziva aritmetičko upoređivanje.

Formati instrukcije upoređivanja za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

CMP a, b, c

CMP a, b

CMP a

CMP

Sa **CMP** je simbolički označeno polje koda operacije, a sa a i b specifikacije dva izvorišna operanda. Sa c je označeno polje za specifikaciju odredišnog operanda koje postoji kod procesora sa troadresnim formatom instrukcija, mada se u slučaju operacije upoređivanja polje c ne koristi.

Razlika između instrukcija upoređivanja kod procesora sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija je samo u tome kako se dolazi do dva izvorišna operanda.

U slučaju procesora sa troadresnim i dvoadresnim formatima instrukcija izvorišni operandi se dobijaju sa lokacija specificiranih sa a i b, u slučaju procesora sa jednoadresnim formatom instrukcija jedan izvorišni operand se dobija implicitno iz akumulatora a drugi izvorišni operand iz lokacije specificirane sa a, dok se u slučaju procesora sa nulaadresnim formatom instrukcija oba izvorišna operanda dobijaju implicitno sa vrha steka.

✓ 4.3 Logičke instrukcije

LOGIČKE INSTRUKCIJE - I, ILI, EKSCLUZIVNO ILI I NOT

Logičkim instrukcijama se realizuju standardne logičke operacije I, ILI, ekskluzivno ILI i komplementiranja

Logičkim instrukcijama se realizuju standardne logičke operacije I, ILI, ekskluzivno ILI i komplementiranja.

Formati instrukcija za logičke operacije I, ILI i ekskluzivno ILI za procesore sa troadresnim, dvoadresnim, jednoadresnim i nulaadresnim formatima instrukcija su:

- **AND a, b, c**
- **AND a, b**
- **AND a**
- **AND**
- **OR a, b, c**
- **OR a, b**
- **OR a**
- **OR**

- **XOR a, b, c**
- **XOR a, b**
- **XOR a,**
- **XOR**

pri čemu su sa **AND, OR i XOR** simbolički označena polja koda operacije za logičke operacije I, ILI i ekskluzivno ILI, respektivno.

Podaci nad kojima se izvršavaju logičke operacije I, ILI i ekskluzivno ILI su binarne reči, pri čemu to može da se čini na više različitih dužina, kao na primer bajt, dva bajta, četiri bajta itd.

Za svaku moguću dužinu binarnih reči mora da postoji poseban kod operacije za svaku od operacija I, ILI i ekskluzivno ILI. Kao ilustracija može se uzeti da ako u procesoru binarne reči mogu da budu dužine 8, 16, 32 i 64 bita, mora da postoje 4 kodova operacija za operaciju **I**.

Formati instrukcija za logičku operaciju komplementiranja za procesore sa dvoaddrēsnim, jednoaddrēsnim i nulaaddrēsnim formatima instrukcija su:

- **NOT a, b**
- **NOT a**
- **NOT**

pri čemu su sa **NOT** simbolički označeno polje koda operacije za logičku operaciju komplementiranja.

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a i b je ista kao i kod instrukcije inkrementiranja. Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takone, ista kao i kod instrukcije inkrementiranja.

LOGIČKE INSTRUKCIJE - LOGIČKO UPOREĐIVANJE I INSTRUKCIJE POMERANJA

Među logičkim instrukcijama se pojavljuje i poseban slučaj operacije logičko I kod koje se rezultat logičke I operacije nikuda ne upisuje, već se samo vrši provera dobijene vrednosti

Formati instrukcije upoređivanja za procesore sa troaddrēsnim, dvoaddrēsnim, jednoaddrēsnim i nulaaddrēsnim formatima instrukcija su:

TST a, b, c

TST a, b

TST a

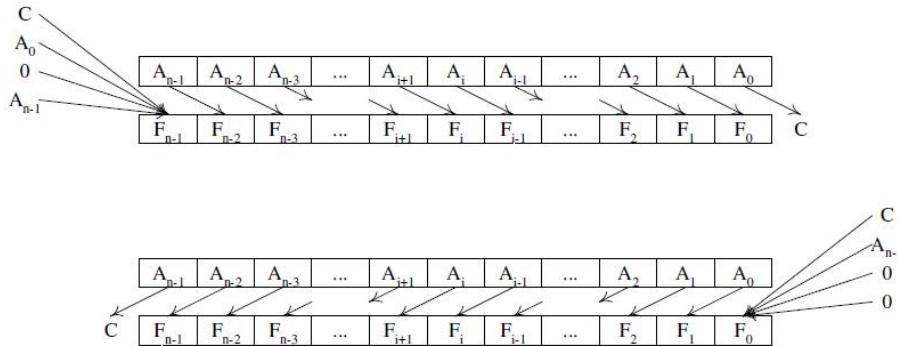
TST

pri čemu je sa **TST** simbolički označeno polje koda operacije za operaciju logičkog upoređivanja.

Eksplisitna specifikacija izvorišnih i odredišnih operanada poljima a, b i c je ista kao i kod instrukcije **CMP**.

Implicitna izvorišta i odredišta u formatima instrukcija gde ih ima, su, takone, ista kao i kod instrukcije **CMP**.

Instrukcijama pomeranja se realizuje pomeranje binarne reči za jedno mesto udesno ili jedno mesto uлево (Slika).



Slika 4.1.1 Pomeranje u desno i u levo [Izvor: Autor]

INSTRUKCIJE SKOKA

Korišćenje brojača PC kao adresu memoriske lokacije sa koje se čita instrukcija i njegovo inkrementiranje prilikom svakog čitanja instrukcije ima kao posledicu sekvensijalnost izvršenja.

Ukoliko se binarna reč 1000 pomeri logički udesno dobija se binarna reč 0100.

Ukoliko se sada i binarna reč 0100 pomeri logički udesno dobija se 0010. Ukoliko se binarne reči 1000, 0100 i 0010 interpretiraju kao celobrojne veličine bez znaka, i time predstavljaju vrednosti 8, 4 i 2, uočava se da se logičkim pomeranjem udesno realizuje deljenje sa 2.

Međutim, ukoliko se binarne reči 1000, 0100 i 0010 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, i time predstavljaju vrednosti 8, 4 i 2, uočava se da se logičkim pomeranjem udesno ne realizuje deljenje sa 2.

Ukoliko se binarna reč 1000 pomeri aritmetički udesno dobija se binarna reč 1100.

Ukoliko se sada i binarna reč 1100 pomeri aritmetički udesno dobija se 1110.

Ukoliko se binarne reči 1000, 1100 i 1110 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, i time predstavljaju vrednosti -8, -4 i -2, uočava se da se aritmetičkim pomeranjem udesno realizuje deljenje sa 2.

Ukoliko se binarna reč 0010 pomeri aritmetički ili logički uлево dobija se binarna reč 0100.

Ukoliko se sada i binarna reč 0100 pomeri aritmetički ili logički uлево dobija se 1000.

Ukoliko se binarne reči 0010, 0100 i 1000 interpretiraju kao celobrojne veličine bez znaka, i time predstavljaju vrednosti 2, 4 i 8, uočava se da se aritmetičkim ili logičkim pomeranjem uлево realizuje množenje sa 2.

Ukoliko se binarna reč 1110 pomeri aritmetički ili logički uлево dobija se binarna reč 1100.

Ukoliko se sada i binarna reč 1100 pomeri aritmetički ili logički uлево dobija se 1000.

Ukoliko se binarne reči 1110, 1100 i 1000 interpretiraju kao celobrojne veličine sa znakom u drugom komplementu, i time predstavljaju vrednosti -2, -4 i -8, uočava se da se aritmetičkim ili logičkim pomeranje uлево realizuje množenje sa 2.

INSTRUKCIJA BEZUSLOVNOG SKOKA

Format instrukcije bezuslovnog skoka je JMP

Instrukcije skoka se svrstavaju u sledeće grupe:

- **instrukcije bezuslovnog skoka,**
- **instrukcije uslovnog skoka,**
- **instrukcije skoka na potprogram i povratka iz potprograma,**
- **instrukcija skoka u prekidnu rutinu i**
- **instrukcija povratka iz prekidne rutine.**

Instrukcija bezuslovnog skoka

Format instrukcije bezuslovnog skoka je:

JMP adresa

pri čemu je sa **JMP** simbolički označeno polje koda operacije, a sa adresa polje sa vrednošću adrese memorijske lokacije na kojoj se nalazi instrukcija na čije izvršavanje treba preći. Izvršavanja instrukcije **JMP** se sastoji u upisivanju vrednost iz polja adresa u programski brojač **PC**.

Kao ilustracija se može uzeti instrukcija JMP 3579 u kojoj 3579 predstavlja vrednost polja adresa.

Izvršavanja instrukcije JMP se sastoji u upisivanju vrednost 3579 iz polja adresa u programski brojač PC. Ova instrukcija se koristi u vreme preponenja poznata adresa instrukcije na koju treba skočiti.

U slučaju kada adresa instrukcije na koju treba skočiti nije poznata u vreme proveravanja, već se izračunava tokom izvršavanja programa, koristi se instrukcija bezuslovnog skoka, čiji je format

JMPIND a

pri čemu je sa **JMPIND** simbolički označeno polje koda operacije, a sa a polje sa specifikacijom adrese memorijske lokacije na kojoj se nalazi instrukcija na čije izvršavanje treba preći. Polje a ima identičnu strukturu kao polja u prethodno razmatranim instrukcijama kojima se eksplicitno specificiraju izvorišni i odredišni operandi korišćenjem različitih načina adresiranja (odeljak o adresiranju), pri čemu su dozvoljena samo memorijska adresiranja, dok direktno registarsko i neposredno adresiranje nisu dozvoljeni.

Izvršavanje instrukcije **JMPIND** se sastoji u sračunavanju adrese memorijske lokacije na osnovu specificiranog memorijskog adresiranja i upisivanju sračunate adrese memorijske lokacije u programski brojač **PC**.

Kao ilustracija se može uzeti da je najpre nekim instrukcijama pre instrukcije JMPIND a kao adresa memorijske lokacije na koju treba skočiti sračunata vrednost 3579, da je ta vrednost upisana u adresni registar AR5 i da instrukcijom JMPIND a vrednost 3579 iz adresnog registra AR5 treba upisati u programski brojač PC.

INSTRUKCIJA USLOVNOG SKOKA

Tokom izvršavanja instrukcije uslovnog skoka se sabiranjem tekuće vrednost programskog brojača PC i pomeraja pom izračunava adresa memorijske lokacije skoka.

Instrukcije uslovnog skoka

Format instrukcija uslovnog skoka je:

- **BEQL pom,**
- **BNEQ pom,**
- **BGRTU pom,**
- **BGREU pom,**
- **BLSSU pom,**
- **BLEQU pom,**
- **BGRT pom,**
- **BGRE pom,**
- **BLSS pom,**
- **BLEQ pom,**
- **BNEG pom,**
- **BNNG pom,**
- **BOVF pom i**
- **BNVF disp**

Dosta je uobičajeno da mnemonici instrukcija uslovnog skoka kod kojih se realizuje relativni skok počinju slovom B (Branch).

Standardne instrukcije uključuje operacije čijim kombinovanjem svaki problem koji treba da se reši u računaru može da se predstavi programom.

pri čemu je sa
BEQL,BNEQ,BGRTU,BGREU,BLSSU,BLEQU,BGRT,BGRE,BLSS,BLEQ,BNEG,BNNG,BOVF
i ***BNVF*** simbolički označeno polje koda operacije, a sa pom polje sa vrednošću pomeraja sa kojim treba napraviti relativan skok u odnosu na tekuću vrednost programskog brojača ukoliko je uslov za skok ispunjen.

Tokom izvršavanja instrukcije uslovnog skoka se sabiranjem tekuće vrednost programskog brojača PC i pomeraja pom izračunava adresa memorijske lokacije skoka, na osnovu vrednosti indikatora N, Z, C i V se proverava da li je uslov za skok specificiran poljem koda operacije ispunjen ili ne i na osnovu toga sračunata adresa memorijske lokacije skoka upisuje ili ne upisuje u programski brojač **PC**, respektivno.

instrukcija	značenje	uslov
BEQL	jednako	Z = 1
BNEQ	nejednako	Z = 0
BGRTU	veće nego bez znaka	C ∨ Z = 0
BGREU	veće nego ili jednako bez znaka	C = 0
BLSSU	manje nego bez znaka	C = 1
BLEQU	manje nego ili jednako bez znaka	C ∨ Z = 1
BGRT	veće nego sa znakom	(N ⊕ V) ∨ Z = 0
BGRE	veće nego ili jednako sa znakom	N ⊕ V = 0
BLSS	manje nego sa znakom	(N ⊕ V) = 1
BLEQ	manje nego ili jednako sa znakom	(N ⊕ V) ∨ Z = 1

Slika 4.1.2 Prikaz instrukcija i njihovo značenje [Izvor: Autor]

INSTRUKCIJA USLOVNOG I BEZUSLOVNOG SKOKA

Prikaz instrukcije bezuslovnog i uslovnog skoka

```

Address      Instruction
004937F7    MOV EAX,200
004937FC    MOV EDX,50
00493801    ADD EAX,67F0
00493806    MOV ECX,490AB3
0049380B    JMP 00497000

;Pretend there is a lot of code inbetween here.

00497000   DEC EDX
00497001   MOV DWORD [49E6CC],EDX
00497007   MOV EAX,EDX

```

Jump to address 497000

then continue the code.

Slika 4.1.3 Prikaz instrukcije bezuslovnog skoka [Izvor: Autor]

JE loc //jump if equal	JNE loc //jump if not equal
JG loc //jump if greater	JGE loc //jump if not greater
JA loc //jump if above	JAE loc //jump if above or equal
JL loc //jump if lesser	JLE loc //jump if less or equal
JB loc //jump if below	JBE loc //jump if below or equal
JO loc //jump if overflow	JNO loc //jump if not overflow
JZ loc //jump if zero	JNZ loc //jump if not zero
JS loc //jump if signed	JNS loc //jump if not signed

Slika 4.1.4 Prikaz instrukcije bezuslovnog skoka [Izvor: Autor]

INSTRUKCIJA SKOKA NA POTPROGRAM I POVRATAK IZ POTPROGRAMA

Format instrukcije skoka na potprogram je JSR

Format instrukcije skoka na potprogram je:

JSR adresa

pri čemu je sa **JSR** simbolički označeno polje koda operacije, a sa adresa polje sa vrednošću adrese memorijske lokacije na kojoj se nalazi prva instrukcija potprograma na čije izvršavanje treba preći.

Izvršavanje instrukcije JSR se sastoji u stavljanju na stek tekuće vrednosti programskog brojača PC i upisivanju vrednosti iz polja adresa u programske brojač PC.

Ovde se podrazumeva da se prilikom čitanja instrukcije **JSR** vrši inkrementiranje vrednosti programskog brojača **PC** i da tekuća vrednost programskog brojača koja se stavlja na stek predstavlja adresu prve sledeće instrukcije posle instrukcije **JSR**.

Format instrukcije povratka iz potprograma je:

RTS

pri čemu je sa **RTS** simbolički označeno polje koda operacije. Instrukcija RTS mora da bude zadnja instrukcija potprograma.

Izvršavanje instrukcije RTS se sastoji u skidanju vrednosti sa steka i upisivanju u programske brojač PC.

Tokom izvršavanja potprograma broj stavljanja vrednosti na stek jednak je broju skidanja vrednosti sa steka, čime se obezbeđuje da instrukcija **RTS** skida sa steka i upisuje u programske brojač PC onu vrednost koju je instrukcija **JSR** stavila na stek.

U skupu instrukcija mogu da se nađu i instrukcije kojima se omogućava za različiti režimi rada računara, pruža podršku za testiranje programa i dr.

Format instrukcije zadavanja režima rada u kome se reaguje na zahteve za prekid je **INTE**;

Format instrukcije zadavanja režima rada u kome se ne reaguje na zahteve za prekid je **INTD**;

Format instrukcije bez dejstva je **NOP**;

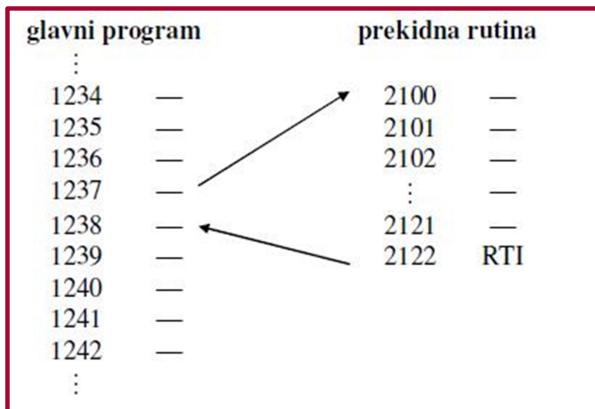
MEHANIZMI PREKIDA

Aktivnosti u procesoru kojima se prekida izvršavanje glavnog programa i skače na prekidnu rutinu nazivaju se opsluživanje zahteva za prekid

Aktivnosti u procesoru kojima se prekida izvršavanje glavnog programa i skače na prekidnu rutinu nazivaju se **opsluživanje zahteva za prekid**, a aktivnosti kojima se obezbeđuje povratak iz prekidne rutine u glavni program i produžavanje izvršavanja glavnog programa

sa mesta i pod uslovima koji su bili pre skoka na prekidnu rutinu nazivaju se **povratak iz prekidne rutine**. Mehanizam prekida kod procesora omogućuje prekid u izvršavanju tekućeg programa, koji će se nazivati glavni program, i skok na novi program, koji će se nazivati prekidna rutina.

Poslednja instrukcija u prekidnoj rutini je instrukcija **RTI**.



Slika 4.1.5 Prikaz prekidne rutine i povratak iz nje [Izvor: Autor]

Zahteve za prekid mogu da generišu:

- Kontroleri periferija da bi procesoru signalizirali spremnost za prenos podataka
- Procesor kao rezultat izvršavanja instrukcije prekida **INT**.

Prekidi pod 1. se nazivaju spoljašnji prekidi jer ih generišu uređaji van procesora.

Prekidi pod 2. se nazivaju je i maskirajući prekidi, jer će procesor u zavisnosti od toga da li se u razredu I registra **PSW** nalazi vrednost 1 ili 0 da reaguje (zahtev za prekid nije maskiran) ili da ne reaguje (zahtev za prekid je maskiran) na ove zahteve za prekid, respektivno.

▼ Poglavlje 5

Intel Core i7

PREGLED NIVOA ISA PROCESORA INTEL CORE I7

Procesor Intel Core i7 je evoluirao u ono što jeste preko brojnih generacija, a njegovo poreklo se može pratiti unazad sve do prvih mikroprocesora.

Osnovni nivo ISA ne samo da i dalje potpuno podržava izvršavanje programa pisanih za procesore 8086 i 8088 (koji su imali isti skup !ISA instrukcija), već nosi i tragove koje je ostavio 8-bitni procesor 8080, popularan sedamdesetih godina prošlog veka.

Procesor 8080 je napravljen uz strogo poštovanje kompatibilnosti sa još starijim procesorom 8008 koji je pak bio zasnovan na modelu 4004, 4-bitnum čipu korišćenom na samom početku kreiranja prvog procesora.

Sa stanovišta softvera. 8086 i 8088 su pravi 16-bitni procesori (mada 8088 ima 8-bitnu magistralu podataka). Njihov naslednik, procesor 80286, bio je takođe 16-bitni. Glavna prednost mu je bila veći adresni prostor, iako je tu preciznost koristilo malo programa zato što se adresni prostor sastojao od 16.384 dela od po 64 KB, umesto kontinualne memorije od 2^{30} bajtova.

Sledeći logičan korak bio je pravi 32-bitni CPU na jednom čipu, 80386, predstavljen 1985. Kao i 80286, i ovaj je bio manjeviše kompatibilan sa svim prethodnicima sve do 8080. Ova kompatibilnost unazad bila je posebna pogodnost za korisnike zbog starijeg softvera koji je mogao da se pokreće bez problema.

Intel Core i7 radi u tri režima od kojih u dva radi kao procesor 8088.

U realnom režimu rada isključuju se sve mogućnosti koje su dodavane posle procesora 8088. Core i7 se ponaša kao taj procesor. Ako neki program uradi nešto nepredviđeno, ceo računar će se blokirati.

Virtuelni režim rada procesora Core i7, koji omogućava izvršavanje programa za stari procesor 8088 u zaštićenom režimu. U ovom režimu računarom upravlja stvarni (postojeći) operativni sistem.

Treći režim je zaštićeni režim rada u kome se Core i7 ponaša kao Core i7, a ne kao procesor 8088.

Postoje četiri nivoa privilegija koje određuju bitovi u registru PSW.

Nivo 0 odgovara režimu jezgra na drugim računarima i u njemu je omogućen potpun pristup resursima. Njega koristi operativni sistem.

Nivo 3 je namenjen korisničkim programima. U njemu je blokiran pristup izvesnim osetljivim instrukcijama i upravljačkim registrima da neki nelegalan program ne bi "srusio" računar.

Nivoi 1 i 2 se retko koriste.

REGISTRI PROCESORA INTEL CORE I7

Intel Core i7 ima ogroman adresni prostor s memorijom podeljenom na 16.384 dela, i sa adresama od 0 do 2 na 30.

Registri **Intel Core i7** prikazani su na Slici. Prva četiri (**EAX**, **EBX**, **ECX** i **EDX**) su 32-bitni registri, opšte namene, iako svaki ima određene posebnosti.

- **EAX** je glavni aritmetički register;
- **EBX** je zadužen za čuvanje pokazivača (na adresu u memoriji);
- **ECX** igra glavnu ulogu u programskim petljama;
- **EDX** je neophodan za množenje i deljenje, gde zajedno s registrom EAX čuva 64-bitne proizvode i količnike.

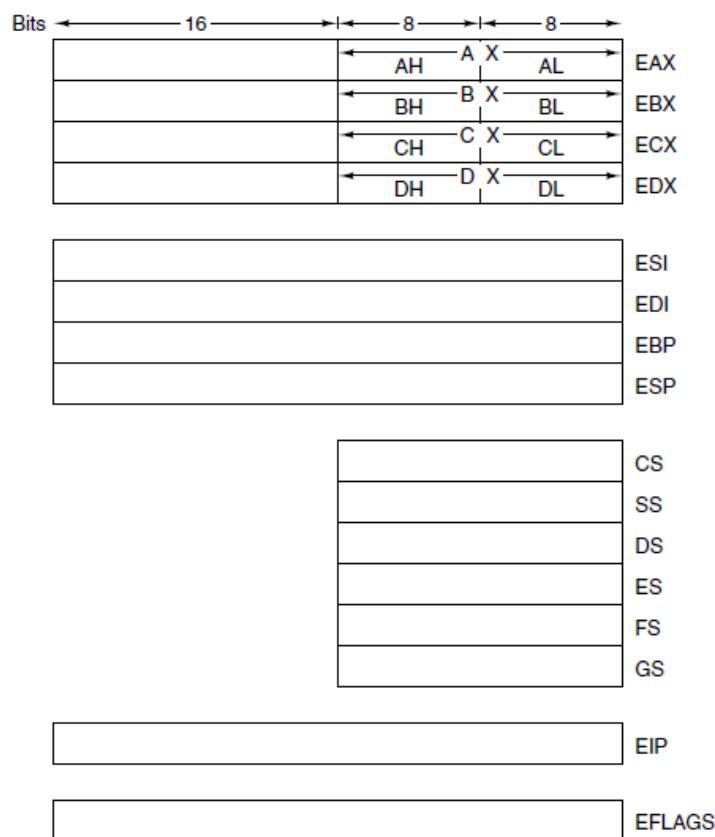
Svaki od ovih registara sadrži 16-bitni register u svojih 16 najmanje značajnih bitova i 8-bitni register u svojih 8 najmanje značajnih bitova.

Sledeća tri registra su i dalje opšte namene, ali i sa više specifičnosti. Registri **ESI** i **EDI** u načelu čuvaju pokazivače na adresu u memoriji naročito za instrukcije pomoću kojih se hardverski manipuliše znakovnim nizovima pri čemu **ESI** ukazuje na izvorišni a **EDI** na odredišni.

Registrar EBP takođe je namenjen čuvanju pokazivača.

Kada se register (kao što je **EBP**) koristi za pokazivanje na početak okvira lokalnih promenljivih na steku, zove se pokazivač okvira.

Na kraju, **ESP** je pokazivač vrha steka.



Slika 5.1 Osnovni registri procesora Core i7 [Izvor: Autor]

TIPOVI PODATAKA PROCESORA INTEL CORE I7

Core i7 podržava označene cele brojeve kao komplernente dvojke, neoznačene cele brojeve, binarno kodirane decimalne brojeve i brojeve u fonnatu pokretnog zareza (IEEE 754)

Zahvaljujući tome što vodi poreklo od skromnih osmobilnih odnosno šesnaestobilnih procesora Core i7 sa celim brojevima ove dužine radi dobro i ima brojne instrukcije za aritmetičke i logičke operacije sa njim kao i za njihovo poređenje.

Operandi ne moraju da budu poravnati u memoriji ali se postižu bolje performanse ukoliko su adrese reči umnošci od 4 bajta.

Tip	8-bitni	16-bitni	32-bitni	64-bitni
Označen ceo broj	X	X	X	
Neoznačen ceo broj	X	X	X	
Binarno kodiran decimalni broj	X			
Broj u formatu pokretnog zareza			X	X

Slika 5.2 Numerički tipovi podataka Core i7. Podražani tipovi označeni simbolom X [Izvor: Auor]

Evolucija Intel procesora (video):

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

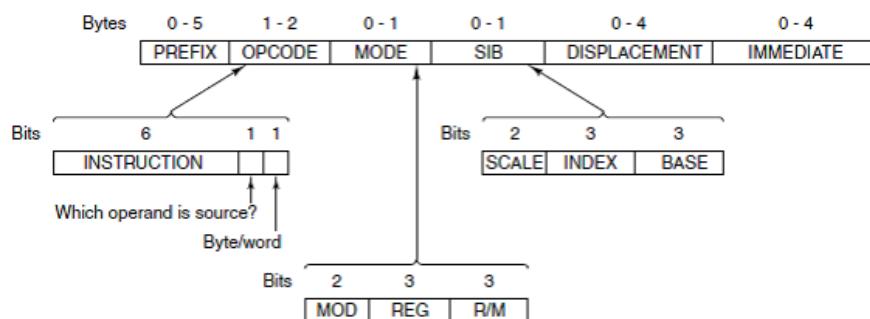
FORMATI INSTRUKCIJA PROCESORA INTEL CORE I7

Formati instrukcija za Core i7 veoma su složeni i nepravilni - imaju do šest polja promenljive dužine, od kojih pet nisu obavezna.

Takvo stanje je nastalo zbog toga što se arhitektura ovog procesora razvijala kroz brojne generacije i što je zadržala neka stara, loša rešenja koja se zbog kompatibilnosti sa starijim procesorima kasnije nisu mogla menjati.

U instrukcijama sa dva operanda ako je jedan operand u memoriji, onaj drugi ne mora da bude u memoriji.

Tako postoje instrukcije za sabiranje sadržaja dva registra, za dodavanje sadržaja registra memorijskom sadržaju i za dodavanje memorijskog sadržaja sadržaju registra, ali ne i za dodavanje jedne memorijske reči drugoj.



Slika 5.3 Formati instrukcije Core i7 procesora [Izvor: Autor]

U ranijim Intelovim arhitekturama svi opkodovi su bili dužine 1 bajt, iako su neke instrukcije često menjane prefiksnim bajtom.

Prefiksni bajt je dodatni opkod ispred instrukcije koji menja njenu namenu.

U većini instrukcija koje preuzimaju operand iz memorije iza bajta sa opkodom nalazi se drugi bajt koji daje informacije o operandu.

Ovih 8 bitova su podeljeni u dvobitno polje **MOD** i dva trobitna registarska polja **REG** i **R/M**. Ponekad se prva 3 bita prvog bajta koriste za proširenje opkoda što daje ukupno 11 bitova za opkod.

Međutim, dvobitno polje za režim rada (**MOD**) znači da postoji ukupno četiri načina adresiranja operanada i da jedan operand uvek mora biti u registru adresiranja operanada i da jedan operand uvek mora biti u registru.

Logično je da se registri **EAX**, **EBX**, **ECX**, **EDX**, **ESI**, **EDI**, **EBP**, **ESP** mogu zadavati kao i drugi registri, ali pravila kodiranja zabranjuju neke kombinacije koje se koriste za specijalne slučajeve.

INSTRUKCIJA PROCESORA INTEL CORE I7

Skup instrukeija Intel Core i7 predstavlja mešavinu instrukcija koje imaju smisla u 32-bit nom režimu rada i onih koji se vraćaju na njegovu istoriju kao procesor 8088

Mnoge od instrukcija Core i7 referenciraju jedan ili dva operanda koji se nalaze u registru ili u memoriji. Na primer, instrukcija **ADD**, koja ima dva operanda, dodaje izvorište odredištu, a instrukcija **INC**, koja ima jedan operand, uvećava svoj operand za 1. Neke od instrukcija postoje u nekoliko blisko povezanih varijanti. Na primer, instrukcije za pomeranje bitova to mogu da rade uлево ili удесно i mogu da tretiraju bit za znak na poseban ili na uobičajen način. Većina instrukcija se kodira na vise načina, prema prirodi operanada.

Polja **SRC** su izvorišta informacija i ne menjaju se. Nasuprot tome, polja **DST** su odredišta i menjaju se pod dejstvom instrukcije. Postoje pravila o tome šta može biti izvorište, a sta odredište, koja se prilično razlikuju od instrukcije do instrukcije, ali u to nećemo sada ulaziti.

Mnoge instrukcije imaju tri varijante: za osmobilne, za šesnaestobilne i za tridesetdvo bitne operative. One se razlikuju po opkodu i(ili) određenom bitu u instrukciji.

Prva grupa sadrži instrukcije koje premeštaju podatke između registara memorije i steka. **Drugu grupu** čine aritmetičke instrukcije sa označenim i neoznačenim brojevima. **Treća grupa** instrukcija obuhvata aritmetiku s binarno kodiranim decimalnim (**Binary Coded Decimal**, **BCD**) brojevima, gde se svaki bajt tretira kao spoj dva četvoro bitna dela (engl. **nibble**).

Logičke (bulove) instrukcije i instrukcije za pomeranje/rotiranje rade s pojedinačnim bitovima u reči ili bajtu. Postoji više kombinacija ovih instrukcija.

Sledeće dve grupe čine **instrukcije za testiranje i poređenje** a potom i grananje programskog toka prema dobijenom rezultatu.

Core i7 ima više **instrukcija za učitavanje, smeštanje, premeštanje**, poređenje i pretraživanje znakovnih nizova iii reči.

Sledeća grupa **instrukcija bavi se kodovima uslova**.

Poslednja grupa sastoji se od instrukcija koje se ne mogu svrstati ni u jednu drugu grupu. Tu su **instrukcije za pretvaranje jednog formata u drugi**, za rad sa okvirom promenljivih na steku, za zaustavljanje procesora i za ulazno-izlazne operacije.

▼ Poglavlje 6

Pokazne vežbe

PRIMER 1: OPERACIJA SABIRANJA - 0 I 1 ADRESNA INSTRUKCIJA

Primeri operacije sabiranja kod nulto i jedno adresnih instrukcija (10 min)

Zadatak (10 minuta): Izračunati dati izraz u procesoru sa nultoadresnim instrukcijama. $A = (X + Y) * Z$

Na raspolaganju su sledeće instrukcije: ADD, MUL, MOV, PUSH/POP i LOAD/STORE.

REŠENJE:

U ovom slučaju se koristi stek, kome se pristupa putem instrukcija PUSH(stavljanje podatka na stek) i POP (uzimanje podatka sa steka).

Za izračunavanje vrednosti zadatog izraza primenom 0 – adresnih instrukcija, instrukcija ima oblik:

PUSH X \ stavlja operand X na vrh steka
PUSH Y \ stavlja operand Y na vrh steka
ADD \ sabira X+Y i stavlja na vrh steka
PUSH Z \ stavlja operand Z na vrh steka
MUL \ množi $(X+Y)*Z$ i stavlja na vrh steka
POP A \ uzima sa vrha steka i stavlja u Y

Pri izvršavanju instrukcija ADD i MUL, uzimaju se dva podatka koja setrenutno nalaze na vrhu steka, sabiraju/množe, a onda se dobijeni rezultat upisuje na vrh steka.

Zadatak (10 minuta): Izračunati dati izraz u procesoru sa jednoadresnim instrukcijama. $A = (X + Y) * Z$

Na raspolaganju su sledeće instrukcije: ADD, MUL, MOV, PUSH/POP i LOAD/STORE.

REŠENJE:

Kod ovog slučaja format zahteva upotrebu akumulatora za smeštaj operanada. Podaci se u akumulator upisuju instrukcijom LOAD, a iz njega čitaju instrukcijom STORE.

Za izračunavanje vrednosti zadatog izraza primenom 1 – adresnih instrukcija, instrukcija ima oblik:

LOAD X \ stavlja operand X u akumulator
ADD Y \ sabira X+Y i stavlja u akumulator
MUL Z \ množi $(X+Y)*Z$ i stavlja u akumulator
STORE A \ sadržaj akumulatora stavlja u A

Instrukcije ADD i MUL uzimaju sadržaj akumulatora, izvršavaju njegovosabiranje/množenje sa operandom navedenim u instrukciji i dobijeni rezultat smeštaju u akumulator.

PRIMER 2: OPERACIJA SABIRANJA - 2 I 3 ADRESNA INSTRUKCIJA

Primeri operacije sabiranja kod dvo i troadresnih instrukcija (10 min)

Zadatak (10 minuta): Izračunati dati izraz u procesoru sa dvoadresnim instrukcijama.

$$A = (X + Y) * Z$$

Na raspolaganju su sledeće instrukcije: ADD, MUL, MOV, PUSH/POP i LOAD/STORE.

REŠENJE:

U ovom slučaju, korišćen je registar R1 da bi sadržaj lokacija predviđenih za čuvanje operanada (X, Y i Z) ostao nepromenjen.

Za izračunavanje vrednosti zadatog izraza primenom 2 – adresnih instrukcija, instrukcija ima oblik:

MOV X, R1 \ stavlja operand X u registar R1

ADD R1, Y \ sabira X+Y i stavlja u registar R1

MUL R1, Z \ množi (X+Y)*Z i stavlja u registar R1

MOV R1, A \ sadržaj R1 stavlja u lokaciju A

Instrukcije ADD i MUL stavlju rezultat operacije sabiranja/množenja namesto prvog operanda u instrukciji (R1) zato što je tako zahtevano upolaznim pretpostavkama za dvoadresni format u postavci zadatka.

Zadatak (10 minuta): Izračunati dati izraz u procesoru sa troadresnim instrukcijama.

$$A = (X + Y) * Z$$

Na raspolaganju su sledeće instrukcije: ADD, MUL, MOV, PUSH/POP i LOAD/STORE.

REŠENJE:

Za izračunavanje vrednosti zadatog izraza primenom 3 – adresnih instrukcija, instrukcija ima oblik:

ADD X, Y, R1 \ sabira X+Y i stavlja u R1

MUL R1, Z, A \ množi (X+Y)*Z i stavlja u A

Kao što se vidi, što je veća adresnost instrukcija, programi su kraći.

PROJEKTNI ZADATAK IZ PREDMETA CS120

Predmet CS120 od predispitnih obaveza sadrži i PZ koji se sastoji od tri mini-projekta.

Na predmetu CS120 - Organizacija računara svaki student samostalno radi projektni zadatak (PZ), koji se sastoji od tri mini-projekta.

Mini-projekti brane se u petoj, desetoj i petnaestoj nedelji semestra!

Svaki student dužan je da tok izrade projekta izveštava svake nedelje u obliku kratkih izveštaja (do jednog pasusa) preko LAMS lekcija u okviru dodatnih aktivnosti interaktivnih lekcija.

O formatu mini-projekata svi studenti (tradicionalni i Internet) biće obavešteni mejlom od strane predmetnog profesora/asistenta.

✓ Poglavlje 7

Zadaci za samostalni rad

DODATNI PRIMERI ZA SAMOSTALNO VEŽBANJE 3 I 2 ADRESNI FORMAT

Koristeći instrukcije ADD, MOV, DIV, MUL, moguće je jednostavno prikazati izraze.

ZADATAK (10 minuta) :

Odredi izraze (A) čije vrednosti računaju dati programi:

za tro-adresni format:

- **ADD Y, Z, A**
- **DIV X, A, A**

REŠENJE:

Do traženih izraza za A se najlakše može doći komentarisanjem svake od instrukcija.

Za 3 - adresni format:

- ADD Y, Z, A \ sabira Y+Z i stavlja u A
- DIV X, A, A \ deli X / (Y+Z) i stavlja u A

Dakle, izraz je:

$$\underline{A = X / (Y + Z)}$$

ZADATAK (10 minuta):

Odredi izraze (A) čije vrednosti računaju dati programi:

za dvo-adresniformat:

- **MOV Y, A**
- **ADD A, Z**
- **MUL A, X**

REŠENJE:

Do traženih izraza za A se najlakše može doći komentarisanjem svake od instrukcija.

Za 2 - adresni format:

- MOVY,A \ stavlja operand Y u A
- ADDA,Z \ sabira Y i Z i stavlja u A
- MUL A, X \ množi X sa (Y + Z) i stavlja u A

Dakle, izraz je:

$$A = (Y + Z) * X$$

DODATNI PRIMERI ZA SAMOSTALNO VEŽBANJE - 1 ADRESNI FORMAT

Koristeći instrukcije DIV, LOAD, STORE, moguće je jednostavno prikazati izraze.

ZADATAK (10 minuta):

Odredi izraze (A) čije vrednosti računaju dati programi: za 1-adresni format:

- LOAD X
- DIV Y
- STORE A
- LOAD Z
- DIV A
- STORE A

REŠENJE:

Do traženih izraza za A se najlakše može doći komentarisanjem svake od instrukcija.

Za 1 - adresni format:

LOAD X, \ stavlja operand X u akumulator

DIV Y, \ deli Y / X i stavlja u Y

STORE A, \ sadržaj akumulatora stavlja u A

LOAD Z, \ stavlja operand Z u akumulator

DIV A, \ deli Y * Z / X i smešta u akumulator

STORE A, \ sadržaj akumulatora smešta u A

Dakle, izraz je:

$$A = Z * Y / X$$

DODATNI PRIMERI ZA SAMOSTALNO VEŽBANJE - 0 ADRESNI FORMAT

Koristeći instrukcije ADD, MUL, POP, PUSH, moguće je jednostavno prikazati izraze.

ZADATAK (10 minuta):

Odredi izraze (A) čije vrednosti računaju dati programi: za 0-adresni format:

- **PUSH X**
- **PUSH Y**
- **ADD**
- **PUSH Z**
- **PUSH W**
- **ADD**
- **MUL**
- **POP A**

Rešenje:

Do traženih izraza za A se najlakše može doći komentarisanjem svake od instrukcija. Za 0 – adresni format:

PUSH X, \ stavlja operand X na vrh steka

PUSH Y, \ stavlja operand Y na vrh steka

ADD, \ sabira X + Y i stavlja na vrh steka

PUSH Z, \ stavlja operand Z na vrh steka

PUSH W, \ stavlja operand W na vrh steka

ADD, \ sabira Z + W i stavlja na vrh steka

MUL, \ množi (X + Y) * (Z + W) i stavlja na vrh steka

POP A, \ uzima sa vrha steka i stavlja ga u A

Dakle, izraz je:

$$A = (X + Y) / (Z + W)$$

✓ Poglavlje 8

Domaći zadatak

DOMAĆI ZADATAK #9

Izrada domaćeg zadatka #9 okvirno traje 30 minuta.

Domaći zadatak #9, zadatak #1 (20 minuta)

Registri A, B, C i D sadrže vrednosti i, j, k i l, respektivno. Ako se sadržaji ovih registara najpre stave na stek u redosledu A, D, B, C, a zatim se sadržaj steka pročita i smesti u C, A, D, B, koje vrednosti tada sadrže ovi registri?

- i ima vrednost broj_indeksa
- j ima vrednost broj_indeksa %10 +1
- k ima vrednost broj_indeks % 14*10
- l ima vrednost zadnje cifre u broj_indeksa -ukoliko je 0 dodati broj 2

Domaći zadatak #9, zadatak #2 (10 minuta) Na raspolaganju je stek orientisan procesor koji može da izvršava instrukcije PUSH i POP, kao i 0 - adresne instrukcije. Prikazati sadržaj steka po izvršenju sledećih instrukcija:

- **PUSH #5**
- **PUSH #2**
- **PUSH #8**
- **SUB**
- **PUSH #7**
- **MUL**
- **ADD**

Predaja domaćeg zadatka

Tradicionalni studenti:

Domaći zadatak treba dostaviti najkasnije 7 dana nakon vežbi, za 100% poena. Nakon toga poeni se umanjuju za 50%.

Domaći zadatak poslati predmetnim asistentima, sa predmetnim profesorima u CC.

Predati domaći zadatak koristeći .doc/.docx uputstvo dato u prvoj lekciji.

Internet studenti treba poslati domaće zadatke najkasnije do 10 dana pred izlazak na ispit predmetnom asistentu zaduženog za internet studente.

Napomena:

Svaki domaći zadatak treba da bude napisan prema dokumentu za predaju domaćih zadataka koji je dat na kraju interaktivne lekcije.

✓ Poglavlje 9

Zaključak

ZAKLJUČAK

Rezime lekcije #9 - Sloj skupa instrukcija - ISA sloj

U lekciji #9 najpre je bilo reči o svojstvima ISA modela.

Pokazana je struktuirana višenivoovska organizacija savremenih računarskih sistema uz osvrt na svaki nivo.

Zatim, predstavljen je skup procesorskih registara koje definiše projektant procesora tokom procesa projektovanja.

Definisani su različiti formati i tipovi instrukcija o opkodova.

Objašnjene su troadresne, dvoadresne, jednoadesne I nultoadresne instrukcije.

Konačno, dat je primer Core i7 ISA modela i instrukcija, regista, tipova podataka i formata instrukcija.

Literatura:

1. A. Tanenbaum, Structured Computer Organization, Chapter 05, pp. 343 – 404