PRINCIPI PROGRAMIRANJA

1. POJAM REGISTARSKOG PRENOSA I POJAM MIKRO–OPERACIJE

Digitalni sistem je sastavljen od hardverskih komponenti čiji je zadatak da obavljaju određene obrade nad podacima. Primjeri komponenti su registri, dekoderi, aritmetička kola i upravljačka logička kola. Za radnju koja se nad podacima upisanim u jednom ili u više registara obavi tokom jednog otkucaja časovnika (tokom jednog takta) kaže se da predstavlja jednu mikro–operaciju. Primjeri mikro–operacija su pomijeranje, brojanje, brisanje i upisivanje. Organizacija digitalnog računara najbolje se opiše ako se odrede a) – c) kako slijedi:

a) Skup registara koje računar sadrži i moguće radnje koje se odnose na registre,

b) Skup mikro–operacija koje se obavljaju nad podacima sadržanim u registrima,

c) Upravljačke radnje tj. kontrolni signali koji naređuju da počne niz mikro–operacija.

Obični primjeri registara u procesoru su AC, MAR i MBR, pa krenimo redom. Akumulator (Accumulator, AC) služi prilikom izvođenja aritmetičkih i logičkih operacija. Adresni registar (Memory Address Register, MAR) čuva adresu memorijske lokacije nad kojom se radnja vrši. Pomoću memorijske magistrale, MAR je povezan sa memorijskom jedinicom. Prihvatni memorijski registar (Memory Buffer Register) služi da prihvati sadržaj memorijske lokacije nad kojom se radnja vrši. Radnja može biti čitanje lokacije ili upis u lokaciju. Pomoću memorijske magistrale, MBR je povezan sa memorijskom jedinicom.

Mikro–operacije mogu da se podijele u četiri grupe 1) – 4) kako slijedi:

1) one koje služe da se ostvari prenos između registara,

2) aritmetičke, 3) logičke i 4) za pomijeranje.

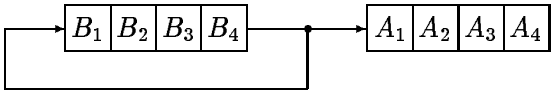
U nastavku će biti riječi posebno o svakoj od četiri vrste.

Znamo da se taktovanje u digitalnom sistemu sprovodi od strane glavnog generatora otkucaja čiji taktni impulsi se primjenjuju na sve flipflopove u sistemu (na sve flipflopove koji rade sinhrono). Primjenjivanje taktnog impulsa samo po sebi ne izaziva promjenu sadržaja registra, nego još treba da bude primijenjen i odgovarajući upravljački signal.

U slučaju 1), podaci iz jednog registra prenose se u drugi registar (podaci se prenose iz izvora na odredište). Razlikujemo nekoliko oblika prenosa, o čemu se govori u nastavku.

U slučaju paralelnog prenosa, svi bitovi izvora jednovremeno se prenose na odredište. Dakle, ta radnja obavi se tokom jednog otkucaja časovnika i ona (znači) predstavlja jednu mikro–operaciju. Ta mikro–operacija bilježi se kao $A \leftarrow B$. Pretpostavlja se da postoji mreža koja sa izlaza flipflopova registra B vodi prema ulazima registra A. Po pravilu, mi želimo da se prenos obavi samo ako je ispunjen neki unaprijed definisani uslov. Uslov ili svejedno upravljačka radnja može da bude označena kao P. Tada se bilježi kao P: $A \leftarrow B$.

Prepisivanje sadržaja procesorovog registra B u procesorov registar A može da bude ostvareno na paralelni način ili na serijski način.



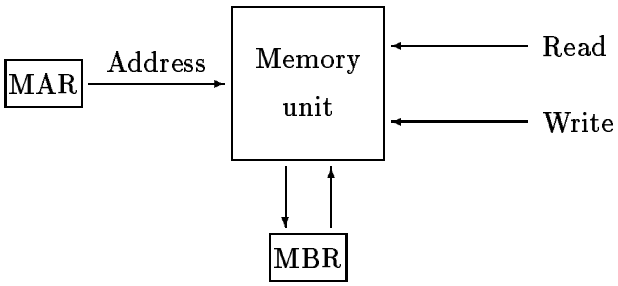
Pogledajmo slučaj serijskog prenosa. Neka su A i B pomerački registri i neka i jedan i drugi imaju po četiri flipflopa (i jedan i drugi su veličine četiri bita), $A = A_1A_2A_3A_4$ i $B = B_1B_2B_3B_4$. Radnja prenosa sadržaja može da bude ostvarena pomoću sljedećeg niza mikro–operacija:

$(A_4, A_3, A_2, A_1) \leftarrow (A_3, A_2, A_1, B_4)$, $(B_4, B_3, B_2, B_1) \leftarrow (B_3, B_2, B_1, B_4)$,

gdje navedeni niz treba da bude ponovljen četiri puta. Drugim riječima, izvršavanje radnje $A \leftarrow B$ traje četiri takta. V. sliku. Vidimo da se u registru B odvija kružno pomijeranje udesno. Vidimo da se u registru A odvija pomijeranje udesno. Što se tiče registra B, na kraju će njegov sadržaj ostati po starom.

U slučaju prenosa na magistralu izvrši se radnja recimo $BUS \leftarrow C$ ili $R1 \leftarrow BUS$. Ovdje su C i R1 registri u procesoru, a BUS je registar na magistrali.

Pogledajmo prenos koji se odnosi na memoriju. Prenošenje podataka iz memorije njenom okruženju naziva se operacijom čitanja. Prenošenje nove informacije memoriji naziva se operacijom upisivanja. Za jednu i drugu operaciju, treba da bude naznačena odgovarajuća memorijska riječ M. Da bi radnja prenosa bila definisana, treba da bude navedena adresa riječi M. Ako se u izjavi o prenosu prosto pojavljuje samo slovo M onda se time kaže da je adresa izabrane memorijske lokacije jednaka tekućem sadržaju registra MAR. Dakle, mikro–operacija čitanja predstavlja prenošenje iz izabrane memorijske riječi M u registar MBR. Bilježi se kao MBR $\leftarrow$ M ili svejedno detaljnije kao MBR $\leftarrow$ M[MAR]. Mikro–operacija upisivanja predstavlja prenošenje iz MBR u predviđenu riječ M. Bilježi se kao M $\leftarrow$ MBR ili svejedno detaljnije kao M[MAR] $\leftarrow$ MBR. Na slici je prikazana šema komunikacije sa memorijom. Jedna i druga mikro–operacija (read i write) obaviće se pod uslovom prisustva odgovarajućeg upravljačkog signala.



Ukupno, za 1) smo nabrojali mogućnosti: paralelno, serijski, sa magistralom i sa memorijom.

Pogledajmo 2). Primjer aritmetičke mikro–operacije jeste mikro–operacija sabiranja R2 $\leftarrow$ R1 + R2, gdje su R1 i R2 dva registra unutar procesora. Izjava R2 $\leftarrow$ R1 + R2 podrazumijeva postojanje logičkih kola koja su u stanju da obave aritmetičku radnju sabiranja.

Primjer oznake je P: A $\leftarrow$ A + B, a njen smisao je: sabiranje će se izvršiti samo ako je P = 1. Primjer oznake je isto tako EA $\leftarrow$ A + B, gdje su A i B registri, a E je flipflop. Flipflop E namijenjen je da čuva prenos koji prilikom sabiranja nastaje na mjestu najveće težine. Tako da EA $\leftarrow$ A + B ima smisao sabiranja sa pamćenjem posljednjeg prenosa.

Navedimo i druge obične primjere aritmetičkih mikro–operacija. Radnja brojanja ili inkrementiranja (povećavanja) za jedan sadržaja registra A zapisuje se kao A $\leftarrow$ A+1. Odgovarajuća hardverska komponenta za tu operaciju jeste očito brojač. Radnja komplementiranja bit–po–bit sadržaja registra A zapisuje se kao A $\leftarrow \overline{A}$. Na primjer, ako je bilo A = 0010 onda će se dobiti A = 1101. Radnja A $\leftarrow \overline{A}+1$ odgovara promjeni predznaka broja, ako se sadržaj registra tumači kao zapis broja u obliku potpunog komplementa (u obliku PK). Na prmjer, ako je staro stanje bilo A = 0010 onda će se poslije izvršavanja radnje A $\leftarrow \overline{A} + 1$ dobiti novo stanje A = 1110. Prevedeno na dekadni, od A = 2 postaće A = -2.

Nešto o hardverskoj realizaciji. Mikro–operacija brisanja P: A $\leftarrow$ 0 može da se ostvari za registar A sastavljen od JK flipflopova tako što će na sve K ulaze flipflopova biti direktno primijenjena upravljačka radnja P. Mikro–operacija komplementiranja P: A $\leftarrow \overline{A}$ može da se ostvari dovođenjem signala P na sve J i K ulaze.

Pogledajmo 3). Primjer logičke mikro–operacije jeste A $\leftarrow$ A $\oplus$ B. Ako se radi o registrima od po četiri bita tj. A = $A_1A_2A_3A_4$ i B = $B_1B_2B_3B_4$ onda navedena radnja ima smisao $A_i \leftarrow A_i \oplus B_i$ za i = 1, 2, 3, 4.

U vezi oznaka. Umjesto p $\vee$ q ponekad se piše p + q (disjunkcija, logičko sabiranje). Takođe, umjesto $\overline{p}$ ponekad se piše p′ (negacija).

Pogledajmo još i 4). Vidjeli smo ranije da postoje brojne vrste pomeračkih registara. Koje su odgovarajuće mikro–operacije? Navedimo samo četiri osnovne operacije pomijeranja (šiftovanja). Upravo: pomijeranje ulijevo, pomijeranje udesno, kružno pomijeranje ulijevo i kružno pomijeranje udesno. Ako se radi o četvorobitnom registru A = $A_1A_2A_3A_4$ onda formule za te četiri osnovne operacije glase redom: $A_1A_2A_3A_4 \leftarrow A_2A_3A_40$, $A_1A_2A_3A_4 \leftarrow 0A_1A_2A_3$, $A_1A_2A_3A_4 \leftarrow A_2A_3A_4A_1$ i

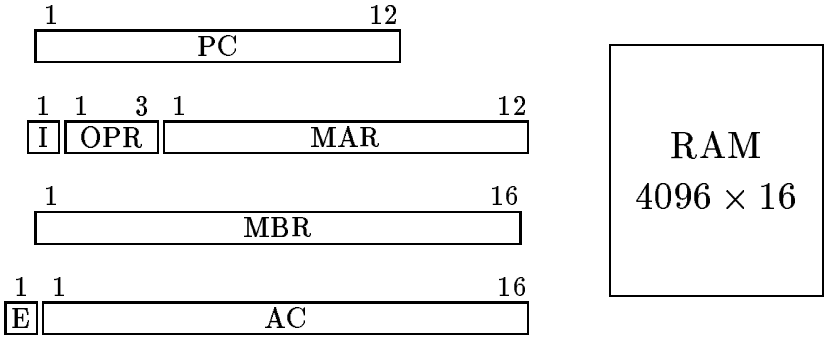
$A_1A_2A_3A_4 \leftarrow A_4A_1A_2A_3$.

Završen pregled mikro–operacija 1) – 4).

Nešto o terminima. Važi jednakost CU + ALU = CPU, gdje je CU control unit (upravljačka jedinica), ALU je arithmetic–logic unit (aritmetičko–logička jedinica), a CPU je central processor unit (jedinica centralnog procesora ili jednostavno procesor).

Nešto o pojmu registra. Znamo da više povezanih flipflopova čini registar. Gdje se registar nalazi? Ako se registar nalazi u CPU onda se naravno kaže da je to jedan procesorov registar. Druga mogućnost: registar se nalazi u memoriji. Umjesto memorijski registar, radije se kaže memorijska riječ. Znamo da memorijska riječ ima svoju adresu, tj. da su memorijske riječi numerisane.

2. ARHITEKTURA OSNOVNOG RAČUNARA



Principe organizacije računara izložićemo na primjeru organizacije jednog konkretnog računara. Za taj konkretni računar kažemo da je osnovni računar ili prosti računar (basic computer). Odgovara stvarnom računaru PDP–8.

Na slici je grubo prikazana arhitektura osnovnog računara, odnosno prikazani su njegovi glavni registri.

Kada se govori o organizaciji nekog računara onda se prvo govori o njegovoj arhitekturi. Upravo, nabroje se svi njegovi registri. Tako ćemo sada opisati registre CPU osnovnog računara, zasad ne sve registre, kasnije će biti kompletirano. Opisaćemo i memorijske registre.

O procesorovim registrima. Registar akumulatora AC sastoji se od 16 bita. Za komunikaciju sa memorijom služe dva registra: prihvatni registar MBR od 16 bita i adresni registar MAR od 12 bita. Flipflop E je pridružen akumulatoru u nekom smislu, odnosno E pomaže akumulatoru. Prisutan je i flipflop I, za koga se kaže da je mode bit. Vidjećemo da za adresiranje argumenta postoje dvije mogućnosti (neposredno i posredno) i vidjećemo da se pomoću I jedna mogućnost razlikuje od druge. Prisutan je i registar OPR, operation register, sadrži kod naredbe u užem smislu. A kaže se da I+OPR sadrži kod naredbe (sadrži kod naredbe u širem smislu). Možemo reći da I + OPR + MAR predstavlja kontrolni registar, control register. Registar PC sastoji se od 12 flipflopova. PC je skraćenica za program counter, što se prevodi kao programski brojač ili kao brojač naredbi. Vidjećemo da sadržaj registra PC određuje – koja naredba programa je u datom trenutku na redu da bude izvršena. Takođe ćemo vidjeti da se, nakon izvršavanja te naredbe, vrijednost PC poveća za 1, osim ako je ta (upravo izvršena) naredba bila naredba za grananje (bila naredba skoka).

O memoriji. Memorijska jedinica sastoji se od 4096 riječi tj. registara. Pojedina riječ ima 16 bita. Radi se o memoriji sa slučajnim pristupom (sa proizvoljnim pristupom, sa adresnim pristupom), RAM, random access memory. Dakle, u svakom trenutku, može se pročitati sadržaj bilo kog registra ili nešto upisati u registar. Kratko, memorija predstavlja RAM oblika 4096×16.

Šta sadrži memorija tokom izvršavanja programa? Sadržaj memorije dijeli se na dva dijela. Prvi dio: program napisan na mašinskom jeziku, po kome računar i funkcioniše. Program se sastoji od naredbi (od mašinskih naredbi). Pojedina naredba ima dužinu 16 bita = 2 bajta. Tako da pojedina naredba zauzima jednu memorijsku riječ. Drugi dio: podaci koji su predmet obrade koja je u toku.

Ili svejedno: podaci nad kojima se obrada vrši. Podaci (argumenti) mogu da budu ulazni podaci ili među–rezultati ili rezultati rada programa.

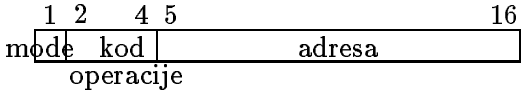
(Za radnju unošenja = upisivanja izvršnog = mašinskog programa u memoriju upotrebljava se izraz loadovanje, od engleske riječi load. Program se unosi u memoriju (u radnu memoriju). Ta radnja po pravilu neposredno prethodi početku izvršavanja programa.)

Očito je $2^{16} = 4096$. Tako da treba 12 bita za definisanje jedne memorijske riječi. Zato PC ima 12 bita. Isto, zato MAR ima 12 bita. S druge strane, dužina pojedine memorijske riječi je 16 bita. Zato MBR ima 16 bita.

3. OBLIK NAREDBE U MAŠINSKOM JEZIKU OSNOVNOG RAČUNARA

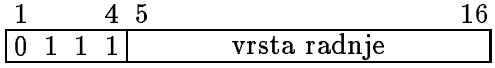
Tri vrste naredbi:

1) naredba koja se odnosi na memoriju

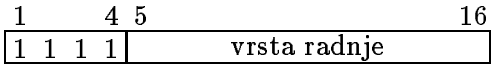


1 / 0 neposredna adresa; 1 / 1 posredna adresa, 2 4 / $\neq 111$

2) naredba koja se odnosi na registre



3) ulazno–izlazna naredba



Pogledajmo kakav je oblik naredbe i koje vrste naredbi postoje.

Znamo da se pojedina naredba mašinskog jezika osnovnog računara izražava pomoću 16 bita. Tih 16 bita dijelimo na dio 1–4 i dio 5–16. Postoje tri vrste naredbi, v. sliku. Upravo, prva vrsta: naredbe koje se odnose na memoriju, druga vrsta: naredbe koje se odnose na registre i treća vrsta: ulazno–izlazne naredbe. Ima 7 naredbi prve vrste, 12 naredbi druge vrste i 6 naredbi treće vrste, kao što ćemo raditi kasnije. Tako da je program napisan na mašinskom jeziku sastavljen od naredbi 25 vrsta. Po čemu se vidi kojoj od tri moguće vrste pripada jedna konkretna naredba? Ako je u dijelu 1–4 upisano 0111 onda je to neka od naredbi koje se odnose na registre. Ako je sadržaj bita 1–4 jednak 1111 onda je to neka ulazno–izlazna naredba. Ako je sadržaj dijela 2–4 različit od 111 onda je u pitanju jedna od 7 naredbi koje se odnose na memoriju (prva vrsta).

Znamo da pripremni korak za izvršavanje naredbe jeste njeno prepisivanje iz memorije u procesor. Vidjećemo da se bit 1 prepisuje u I, bitovi 2–4 u OPR, a bitovi 5–16 u MAR. Naredbe koje se odnose na memoriju (naredbe prve vrste) su najvažnije u nekom smislu. Neki izrazi koji se upotrebljavaju odražavaju tu okolnost. (1) Tako se kaže da bit 1 odnosno flipflop I jeste mode bit (bit za način) i da je njegova uloga da se razlikuju dva načina definisanja argumenta neke operacije (neposredni i posredni način). (2) Kaže se da bitovi 2–4 odnosno registar OPR služi za kod operacije, tj. da oni definišu koja radnja treba da bude obavljena (koja vrsta radnje treba da bude obavljena). (3) Takođe se kaže da 5–16 odnosno MAR određuju adresu. Mada je i (1) i (2) i (3) istinito u punom smislu samo ako je u pitanju naredba koja se odnosi na memoriju (naredba prve vrste).

Šta je to neposredno ili obično a šta je to posredno adresiranje. Isto se kaže i direktno odnosno indirektno. Argument neke operacije dolazi iz memorije, v. sliku. Slika ilustruje na primjeru naredbe kojom se vrši uvećavanje sadržaja akumulatora. Pogledajmo prvo osnovni ili obični slučaj I = 0, neposredno adresiranje. Svako zna da kod aritmetičke radnje sabiranja učestvuju tri broja: $x + y = z$, čiji je smisao redom prvi sabirak, drugi sabirak i rezultat (x, y i z). Kod računara imamo sljedeće okolnosti. Prije izvršavanja naredbe, x se nalazi u akumulatoru, a y se nalazi u memoriji. Nakon izvršavanja naredbe, u akumulatoru će biti z. Tako da MAR sadrži adresu veličine y.

Objasnimo i na drugi način. Čemu je jednak drugi sabirak? Pročitaj sadržaj jednog registra RAM–a. A kog registra? Vidi šta piše u MAR. Sve ovo svakako kada je I = 0.

Princip neposrednog ili posrednog adresiranja odnosi se na sve naredbe koje se obraćaju memoriji. Da se jedna takva naredba izvrši, ako je I = 0 onda se prema memoriji ide jednom. Ako je I = 1 onda se ka memoriji ide dvaput (ima jedan korak više).

Osnovni slučaj I = 0 nije dovoljan za neke situacije. Zato je uvedeno i tzv. posredno adresiranje, kome odgovara I = 1. Posredno adresiranje služi za rad sa nizovima, primjera radi var a:array[1..100]of integer;

Ako je I = 0 onda čitanjem dijela memorije kako upućuju bitovi 5–16 saznajemo argument. Ako je I = 1 onda tim čitanjem saznajemo tek adresu argumenta. Tako da je sada očito potrebno i dodatno drugo čitanje da se sazna sami argument. Prilikom tog drugog čitanja, mi ćemo se ravnati svakako po maločas spomenutoj adresi (po veličini koju smo saznali prilikom prvog čitanja).

Neposredno i posredno adresiranje:

pogledajmo jednu naredbu koja se odnosi na memoriju

sabiranje tj. uvećavanje akumulatora

naredba ADD, 2–4 [0 0 1]

razmotrimo naredbu:

0 0 0 1	63

prije njenog izvršavanja:

AC	2000
63	3000

a poslije:

AC	5000
63	3000

razmotrimo naredbu:

1 0 0 1	63

prije njenog izvršavanja:

AC	2000
63	3000
3000	14500

a poslije:

AC	16500
63	3000
3000	14500

4. KONTROLNA JEDINICA OSNOVNOG RAČUNARA I VREMENSKI CIKLUSI

Na slici je prikazan blok–dijagram kontrolne jedinice. Vidi se da glavni dio kontrolne jedinice čine "logička kola". To je jedan splet logičkih kola (upravljačkih logičkih kola). Izlazi iz spleta su upravljački signali (kontrolne radnje). Ti signali definišu mikro–operacije. A na osnovu čega ih definišu? Drugim riječima, koji su ulazi u splet? Sa slike se vidi da ima pet vrsta ulaza, upravo: t_0–t_3, c_0–c_3, I, q_0–q_7 i B_5–B_{16}, pa krenimo redom.

Registar SC (sequence counter, brojač taktova) ima dva bita. Taj registar definiše vremensko vođenje rada računara, odnosno on predstavlja generator taktova (izvor otkucaja). Sadržaj dvobitnog brojača SC daje se na ulaz tzv. vremenskog dekodera koji ima oblik 2 puta 4. Izlazi dekodera označeni su kao t_0, t_1, t_2 i t_3. Brojač SC radi stalno na jedan te isti način, mirno i spokojno, mada dosta brzo. Njegov rad neposredno se odražava na vrijednosti promjenljivih t_0 do t_3. Tako da se i te četiri vrijednosti (tokom cijelog vremena rada računara) mijenjaju stalno po jednom te istom nepromjenljivom šablonu. Upravo, iz takta u takt ima se redom ovako:

$t_0 = 1, t_1 = 0, t_2 = 0, t_3 = 0$
$t_0 = 0, t_1 = 1, t_2 = 0, t_3 = 0$
$t_0 = 0, t_1 = 0, t_2 = 1, t_3 = 0$
$t_0 = 0, t_1 = 0, t_2 = 0, t_3 = 1$

U jednom redu prikazan je jedan takt. Dalje se ponavlja. Postoji samo jedna mogućnost da se ovaj šablon poremeti. Flipflop S ima ulogu prekidača za start–stop. Taj flipflop djeluje na ulaz za dozvolu

(Enable) vremenskog dekodera. Vremenski signali t_i se generišu samo kada je S = 1. Ako je S = 0 onda će biti $t_0 = t_1 = t_2 = t_3 = 0$. Tada nijedan upravljački signal više neće biti izdat, računar će se zaustaviti.

Vremenski ciklusi:

F = 0, R = 0 slijedi $c_0 = 1$, $c_1 = 0$, $c_2 = 0$, $c_3 = 0$, ovo je Fetch cycle
F = 0, R = 1 slijedi $c_0 = 0$, $c_1 = 1$, $c_2 = 0$, $c_3 = 0$, ovo je Indirect cycle
F = 1, R = 0 slijedi $c_0 = 0$, $c_1 = 0$, $c_2 = 1$, $c_3 = 0$, ovo je Execute cycle
F = 1, R = 1 slijedi $c_0 = 0$, $c_1 = 0$, $c_2 = 0$, $c_3 = 1$, ovo je Interrupt cycle

Digitalni računar radi u diskretnim koracima. Mikro–operacije se izvršavaju tokom svakog koraka. Naredbe se čitaju iz memorije i izvršavaju se u registrima CPU. One se izvršavaju preko mikro–operacija. Kada se jednom aktivira startni prekidač onda računar stalno postupa po svom šablonu. Iz memorije se pročita naredba čija se adresa nalazi u registru PC, naredba se upisuje u registar MBR. Mode bit naredbe prepiše se u flipflop I, a kod naredbe u užem smislu prepiše se u registar OPR (tri bita). Sadržaj registra OPR dekodira se u upravljačkoj jedinici (u kontrolnoj jedinici). Ako se vidi da se naredba odnosi na memoriju (pa prema tome zahtijeva argument iz memorije) onda kontrola gleda sadržaj flipflopa I. Ako je I = 1 onda se kontrola obraća memoriji da bi se pročitala adresa argumenta, pa se zatim ponovo obraća memoriji da bi se i sami argument pročitao. Ako je I = 0 onda se memoriji pristupa radi čitanja argumenta. Prema tome, ukupno, riječ koja se nalazi u registru MBR je jednog od sljedeća tri moguća tipa: to je naredba ili adresa argumenta ili argument. Kada se iz memorije čita naredba onda se kaže da se računar nalazi u ciklusu dohvatanja naredbe (u ciklusu primanja naredbe, u prijemnom ciklusu, u fetch ciklusu). Tokom vremena čitanja (iz memorije) adrese argumenta računar se nalazi u indirektnom ciklusu (indirect cycle). A tokom izvršnog ciklusa (execute cycle) obavi se, pored ostalog, i čitanje argumenta. Kasnije ćemo vidjeti da postoji i četvrti vremenski ciklus – interrupt cycle (prekidni ciklus).

Kažimo unaprijed da se prekidni ciklus koristi samo prilikom ulazno–izlazne komunikacije. Kasnije ćemo vidjeti da ulazno–izlazna komunikacija može da bude ostvarena bez upotrebe ili sa upotrebom pogodnosti prekida (program odlučuje–bira). Pogodnost prekida pruža mogućnost programeru da komunikaciju sa ulaznom i izlaznom jedinicom organizuje na sasvim efikasan način.

Kontrola vodi računa o raznim vremenskim ciklusima (o raznim mašinskim ciklusima). Drugim riječima, zadatak kontrolne jedinice je, pored ostalog, i da reguliše smjenjivanje vremenskih ciklusa navedena četiri tipa. U kontrolnoj jedinici, dva flipflopa služe da se ciklusi razlikuju. To su flipflopovi F i R. Sa slike se vide da se izlazi flipflopova F i R dovode na ulaze tzv. ciklusnog dekodera koji ima oblik 2 puta 4. Izlazi dekodera su c_0 do c_3. Očito je da se dodjeljivanjem vrijednosti flipflopovima F i R definiše jedan od četiri moguća ciklusa. A onda veličine c_0 do c_3 učestvuju u definisanju upravljačkih signala, kao što je već rečeno.

Izvršavanje jednog ciklusa (izvršavanje bilo kog ciklusa) traje tačno četiri takta. Već je rečeno da se veličine t_0 do t_3 mijenjaju sasvim monotono: iz takta u takt je $t_0 = 1$ pa $t_1 = 1$ pa $t_2 = 1$ pa $t_3 = 1$, a dalje periodično (kružno). Za promjenljive c_0 do c_3 možemo reći samo da se mijenjaju u osnovi monotono (obično je poslije $c_0 = 1$ $c_1 = 1$, zatim $c_2 = 1$, onda $c_3 = 1$, itd). Na primjer, ako se ne koristi mogućnost prekida onda će biti preskočen prekidni ciklus $c_3 = 1$. Drugim riječima, poslije $c_2 = 1$ postaće $c_0 = 1$. Slično, tokom izvršavanja naredbe koja se odnosi na memoriju a čija je adresa napisana neposredno (I = 0) biće preskočen indirektni ciklus $c_1 = 1$. Drugim riječima, poslije $c_0 = 1$ postaće $c_2 = 1$.

Za ilustraciju, uzmimo da se izvršava dio mašinskog programa sastavljen od niza naredbi koje se odnose na memoriju, gdje svaka naredba ima neposrednu adresu. Dodatno, uzmimo da izvršavanje tog dijela programa nije spolja ometano (prekidano), tj. da nema spoljašnjih događaja koji mogu da nastupe asinhrono a koji bi uticali. Kraće rečeno, uzmimo da se ne zalazi u prekidni ciklus. Tada će se vrijednosti promjenljivih c_0 do c_3 (gledano po vremenskoj osi) smjenjivati ovako:

$c_0 = 1$, $c_1 = 0$, $c_2 = 0$, $c_3 = 0$, tokom četiri takta
$c_0 = 0$, $c_1 = 0$, $c_2 = 1$, $c_3 = 0$, tokom četiri takta
dalje se periodično ponavlja

... gdje svaka naredba ima posrednu (indirektnu) adresu ...

$c_0 = 1$, $c_1 = 0$, $c_2 = 0$, $c_3 = 0$, tokom četiri takta
$c_0 = 0$, $c_1 = 1$, $c_2 = 0$, $c_3 = 0$, tokom četiri takta
$c_0 = 0$, $c_1 = 0$, $c_2 = 1$, $c_3 = 0$, tokom četiri takta
dalje se periodično ponavlja

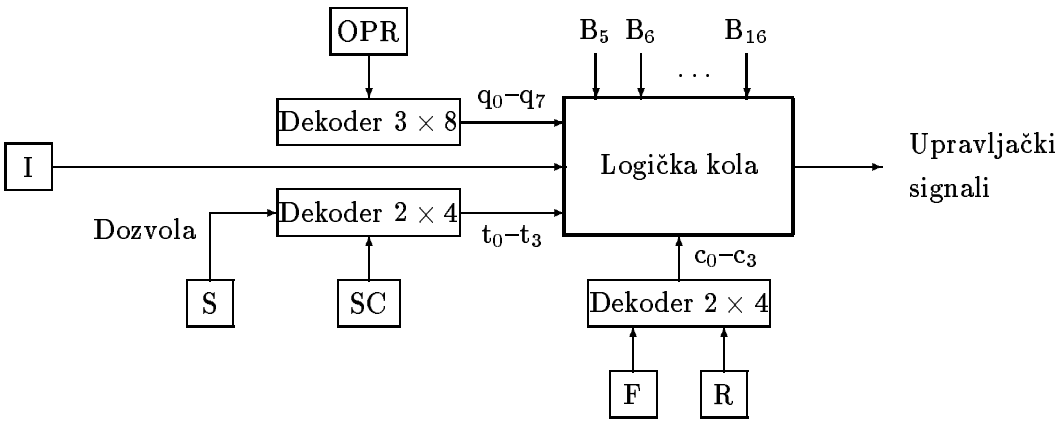
Izvršavanje jedne bilo koje naredbe traje četiri ciklusa ili manje. Tako da izvršavanje jedne bilo koje naredbe traje 16 taktova ili manje. Kasnije ćemo vidjeti da izvršavanje pojedine naredbe (pojedine mašinske naredbe) traje zavisno od okolnosti dva ili tri ili četiri ciklusa, tj. 8 ili 12 ili 16 taktova.

Vrijednost flipflopa I takođe utiče na definisanje kontrolnih signala. Toj vrijednosti nije potreban dekoder. Sa slike se vidi da se izlaz flipflopa I sprovodi pravo na upravljačku kombinacionu mrežu.

Sa slike se vidi da se sadržaj registra OPR uvodi u jedan dekoder oblika 3 puta 8 (to je tzv. dekoder stanja), čiji su izlazi označeni redom kao q_0 do q_7. Sa svoje strane, ti izlazi dovode se na upravljačku kombinacionu mrežu. Ako je $q_0 = 1$ ili itd. ili $q_6 = 1$ (sedam mogućnosti) onda se radi o naredbi prve vrste (ima sedam naredbi prve vrste). Ako je $I = 0$ i $q_7 = 1$ onda se radi o naredbi druge vrste. A ako je $I = 1$ i $q_7 = 1$ onda se radi o naredbi treće vrste.

U kombinacionu upravljačku mrežu uvode se i signali B_5 do B_{16}. Uvedena je oznaka $B_i = MBR(i)$, za $1 \leq i \leq 16$. Vidjeće se da se ti signali koriste kada se radi o naredbama druge vrste (naredbe koje se odnose na registre) i naredbama treće vrste (ulazno–izlazne naredbe). Kao što znamo, za te naredbe je karakteristično da one ne traže argument iz memorije. Zato se adresno polje koristi da se tu upiše vrsta naredbe. Drugim riječima, pomoću B_i sazna se vrsta naredbe (sazna se o kojoj vrsti radnje je riječ). Na primjer, ima 12 naredbi druge vrste. Svakoj od tih 12 naredbi odgovara jedan određeni raspored vrijednosti B_5 do B_{16}. U svakom rasporedu, samo jedna od tih vrijednosti je $= 1$, a sve ostale su $= 0$. Slične okolnosti važe i kada se radi o naredbama treće vrste. Samo što naredbi treće vrste ima manje od 12, tako da nisu iskorišćene sve mogućnosti. Ponovimo da mogućnosti ima ukupno 12 na broju: raspored u kome je $B_5 = 1$, ..., raspored u kome je $B_{16} = 1$.

Šema kontrolne jedinice. Uvedena je oznaka: $B_5 = MBR(5)$ = sadržaj petog bita registra MBR. Slično i B_6, ..., B_{16}:



5. IZ KOJIH MIKRO–OPERACIJA SE SASTOJI POJEDINI VREMENSKI CIKLUS

Prijemni ciklus (Fetch cycle):

c_0t_0: MAR $\leftarrow$ PC	prenošenje adrese naredbe
c_0t_1: MBR $\leftarrow$ M, PC $\leftarrow$ PC + 1	naredba se donosi, programski brojač se prilagođava
c_0t_2: I $\leftarrow$ MBR(1), OPR $\leftarrow$ MBR(2 – 4)	djelovi naredbe se raspoređuju
c_0t_3: if $q_7'I = 1$ then R $\leftarrow$ 1 else F $\leftarrow$ 1	podešavanje vrijednosti F i R

Izvršavanje prijemnog ciklusa traje četiri takta. Prijemni ciklus poznaje se po promjenljivoj c_0. Način izvršavanja prijemnog ciklusa je svaki put jedan te isti, ne zavisi od vrste naredbe, ne zavisi od naredbe. Kao što znamo, zadatak tog ciklusa je da se iz memorije donese naredba i da se pripremi teren za naredni ciklus. U prethodnoj tabeli prikazano je sasvim detaljno kako se odvija prijemni ciklus, tj. prikazane su četiri njegove mikro–operacije. Izrazimo riječima. Dok je $t_0 = 1$ sadržaj programskog brojača PC prepisuje se u adresni registar MAR. Zato što PC sadrži adresu naredbe koja je stigla na red za izvršavanje. I zato što MAR (u saradnji sa MBR) učestvuje u ostvarivanju komunikacije sa memorijom. U toku $t_1 = 1$, u MBR se donosi sadržaj memorijske lokacije M (radnja čitanja memorije). Tokom tog istog vremena $t_1 = 1$ PC se uveća za 1, čime je PC već u tom trenutku postao pripremljen za iduće čitanje memorije, za izvršavanje iduće mašinske naredbe (što će uslijediti kasnije). U periodu $t_2 = 1$ izvršiće se dvije radnje. Prvi bit registra MBR prepisuje se u flipflop I. Bitovi 2–4 registra MBR prepisuju se u registar OPR. Zapaziti da je adresni dio naredbe 5–16 ostao u MBR (čitava naredba je ostala u 1–16). Vrijeme $t_3 = 1$ posvećeno je definisanju radnji koje treba da uslijede. Ako je $B_1B_2B_3B_4 = 0111$ naredba druge vrste ili $B_1B_2B_3B_4 = 1111$ naredba treće vrste onda treba preći u izvršni ciklus. Ako je $B_2B_3B_4 \neq 111$ i $B_1 = 0$ naredba prve vrste sa direktnom adresom onda takođe treba preći u izvršni ciklus. Jedino ako je $B_2B_3B_4 \neq 111$ i $B_1 = 1$ naredba prve vrste sa indirektnom adresom treba preći u indirektni ciklus. Veličina B_1 vidi se po I. Veličine B_2, B_3 i B_4 vide se (posredstvom OPR) po $q_0, \ldots, q_7$. Da bi se razumjela dešavanja tokom $t_3 = 1$ treba znati da je prilikom ulaska u prijemni ciklus bilo F = 0 i R = 0. S druge strane, ako na završetku prijemnog ciklusa bude F = 0 i R = 1 onda će se preći u indirektni ciklus, a ako bude F = 1 i R = 0 onda u izvršni. Znamo da F i R determinišu vrijednosti promjenljivih $c_0, \ldots, c_3$. Poznata je jednakost $(q_7'I)' = q_7 + I'$. Očito je da $q_7'I = 1$ znači $q_7 = 0$ i I = 1. Isto tako, $q_7'I = 0$ znači $q_7 = 1$ ili I = 0.

U drugom redu (kada je $c_0 = 1$ i $t_1 = 1$) piše MBR $\leftarrow$ M, PC $\leftarrow$ PC+1. Te dvije radnje MBR $\leftarrow$ M i PC $\leftarrow$ PC + 1 možemo smatrati jednom mikro–operacijom, jer se one izvedu tokom jednog takta.

Naš opis rada osnovnog računara predviđa da čitanje memorije (ili upis u nju) može da se obavi tokom jednog takta. Mi tako predviđamo da bi se izlaganje pojednostavilo. Samo se napominje da je kod mnogih računara za čitanje ili upis potrebno vrijeme koje je duže od jednog takta.

Indirektni ciklus (Indirect cycle):

c_1t_0: MAR $\leftarrow$ MBR(5 – 16)	adresni dio naredbe donosi se u adresni registar
c_1t_1: MBR $\leftarrow$ M	mikro–operacija čitanja memorije
c_1t_2:	ništa se ne dešava
c_1t_3: F $\leftarrow$ 1, R $\leftarrow$ 0	determiniše se prelazak u izvršni ciklus

Indirektni ciklus poznaje se po promjenljivoj c_1. Govoreći formalno, tokom tog ciklusa vrši se čitanje memorijske lokacije. Govoreći po smislu, tokom tog ciklusa vrši se čitanje adrese argumenta. U tabeli od maločas prikazano je sasvim detaljno kako se odvija indirektni ciklus. Ispričajmo riječima. Prilikom $t_0 = 1$ izvrši se radnja registarskog prenosa kojom se pripremi adresni registar MAR. Tokom $t_1 = 1$ izvrši se radnja čitanja memorijske riječi M ili svejedno M[MAR]. Riječ M ima 16 bita, iako u konkretnom slučaju M ima smisao adrese; znamo da za definisanje adrese treba 12 bita. Tokom $t_2 = 1$ imamo prazan hod. Zato što se prilagođavanje dva flipflopa F i R uvijek vrši tokom vremena $t_3 = 1$, kako bi idući ciklus počeo u situaciji kada je $t_0 = 1$. Tokom $t_3 = 1$ setuje se F (postavlja se F) i resetuje se R (briše se R). Naime, iz indirektnog ciklusa uvijek se prelazi u izvršni ciklus.

Izvršni ciklus (Execute cycle):

Tokom izvršnog ciklusa dolazi do izvršavanja (u užem smislu) jedne mašinske naredbe. Tako da četiri mikro–operacije koje se izvedu tokom tog ciklusa zavise od vrste mašinske naredbe. Drugim riječima, budući da ima 25 vrata naredbi to bi sada trebalo napisati 25 tabela (jedna tabela ima četiri reda), ako želimo da kompletno opišemo računarov izvršni ciklus. Mi ćemo samo ilustrovati na jednom primjeru. Razmotrimo naredbu za uvećavanje akumulatora ADD. Toj naredbi odgovara $B_2B_3B_4 = 001$ tj. $OPR = 001$ ili svejedno $q_1 = 1$.

Za ADD

$q_1c_2t_0$: $MAR \leftarrow MBR(5-16)$	adresa se donosi u adresni registar
$q_1c_2t_1$: $MBR \leftarrow M$	mikro–operacija čitanja memorije
$q_1c_2t_2$: $EAC \leftarrow AC + MBR$	izvršava se sabiranje i pritom se sačuva prenos
$q_1c_2t_3$: $F \leftarrow 0$	sada će biti izvršen prelazak u prijemni ciklus

Izvršni ciklus poznaje se po tome što je $c_2 = 1$. Mašinska naredba ADD poznaje se po tome što je $q_1 = 1$. Naredba ADD želi da ostvari jedno sabiranje. Flipflop E pomaže prilikom sabiranja, time što prihvata prenos sa mjesta najveće težine. Na E i AC ovdje se gleda kao na jednu cjelinu.

Može da bude $I = 0$ ili $I = 1$, što za odvijanje izvršnog ciklusa nema značaja. Zaista, ako je $I = 1$ onda je računar maločas prošao kroz indirektni ciklus. Dakle, u trenutku kada je postalo $q_1 = c_2 = t_0 = 1$ imamo sigurno da dio 5–16 registra MBR sadrži m, gdje je m adresa argumenta. A mi želimo da u MBR dospije argument M. Znamo da M ima smisao drugog sabirka y.

Tokom $t_0 = 1$ pripremi se adresni registar MAR. Tokom $t_1 = 1$ pročita se argument. Tokom $t_2 = 1$ izvrši se samo sabiranje. Tako da učinak izvršnog ciklusa naredbe ADD možemo da izrazimo kao $EAC \leftarrow AC + M$. Zapis $EAC \leftarrow AC + M$ odražava jednu veliku operaciju ili jednu makro–operaciju.

Tokom $t_3 = 1$ učini se da postane $F = 0$ i $R = 0$, što će očito izazvati prelazak u novi prijemni ciklus. Odnosno, uradi se $F \leftarrow 0$, a i inače je bilo $R = 0$.

Naglasimo da je u prethodnoj tabeli radnja tokom $t_3 = 1$ prikazana lažno ili uprošćeno. Upravo, u stvarnosti se tokom $t_3 = 1$ vrši prilagođavanje vrijednosti dva flipflopa F i R. Kasnije ćemo detaljno vidjeti od čega zavisi prilagođavanje. Bez obzira, nastupiće jedan od dva moguća ishoda prilagođavanja: da se pređe u prekidni ciklus ili da se pređe u novi prijemni ciklus. Mi smo zasad prikazali kao da se svaki put prelazi u novi prijemni ciklus, a kasnije će biti prikazano kako treba.

Prekidni ciklus (Interrupt cycle):

O prekidnom ciklusu biće riječi kasnije, kada budemo govorili o ulazno–izlaznim naredbama.

Ukratko o pojmu prekida (o pojmu interapta). Računari koriste tzv. pogodnost prekida ili sistem prekida.

Sistem prekida stavlja na raspolaganje mogućnost da program P (čije izvršavanje je u toku) može da prepusti upravljanje računarom drugom jednom programu Q. Pomoćni program Q (rutina Q) služi po pravilu da obavi neku kratku i rutinsku obradu, najčešće služi da obavi neko ulazno ili izlazno prenošenje. Kada se rutinska obrada obavi onda će se upravljanje vratiti programu P. Rad programa P biće obnovljen, izvršavanje programa P biće nastavljeno kao da P nije ni bio prekidan.

Odluka o eventualnom prepuštanju upravljanja zavisi od samog programa P, ali i od spoljašnjih okolnosti (od spoljašnjeg signala). Spoljašnji signal nastupa asinhrono, u odnosu na rad programa P.

Ako su se stekli uslovi onda će se tokom $c_3 = 1$ upravljanje računarom prepustiti rutini Q. Drugim riječima, tada će se izvršiti skok na rutinu Q.

6. NAREDBE KOJE SE ODNOSE NA MEMORIJU

U ovom naslovu definišu se naredbe prve vrste osnovnog računara. Znamo da je za te naredbe karakteristično da se one obraćaju memoriji. Prvo se o tim naredbama govori grubo:

	oznaka	2–4	smisao
1	AND	000	izvrši AND (izvrši &) između memorijske riječi i akumulatora
	and – i		
2	ADD	001	uvećaj akumulator za memorijsku riječ
	add – dodati		
3	LDA	010	donesi iz memorije u akumulator
	load to accumulator – optereti akumulator		
4	STA	011	pošalji iz akumulatora u memoriju
	store from accumulator – pohrani		
5	BUN	100	bezuslovni skok (bezuslovno grananje)
	branch unconditionally – granaj bezuslovno		
6	BSA	101	skok i spašavanje povratne adrese
	branch and save return address – granaj i spasi povratnu adresu		
7	ISZ	110	povećaj za jedan i preskoči ako nula
	increment and skip–if–zero – inkrementiraj i poskoči–ako–nula		

Memorijska riječ definisana je adresnim dijelom 5–16 same naredbe. Bolje rečeno, u definisanju memorijske riječi učestvuje i prvi bit naredbe (učestvuje i flipflop I). Pomoću naredbe AND izvrši se konjunkcija bit–po–bit između memorijske riječi i AC, a rezultat te radnje ostaje upisan u AC. Naredba ADD vrši obično sabiranje brojeva (brojevi su prikazani binarno). Pomoću LDA vrši se prenošenje u smjeru iz memorije u procesor. Pomoću STA vrši se prenošenje u suprotnom smjeru.

Sada su na redu definicije naredbi. Sljedeća tabela definiše učinak izvršnog ciklusa pojedine naredbe na jeziku velikih operacija (na jeziku makro–operacija). U tabeli, m označava tekući sadržaj dijela 5–16 registra MBR (m označava sadržaj dijela 5–16 registra MBR u trenutku ulaska u izvršni ciklus). Dakle, m je adresa ili efektuvna adresa. U tabeli, M označava sadržaj odgovarajuće memorijske lokacije. Dakle, M je memorijska riječ. Formulom: $M = c(m)$, gdje c znači contents (sadržaj).

Pogledajmo posebno slučajeve $I = 0$ i $I = 1$. Ako je $I = 0$ onda je m jednako dijelu 5–16 same naredbe, a M je sadržaj odgovarajuće memorijske lokacije. Možemo reći da se učinak čitave naredbe poklapa sa učinkom njenog izvršnog ciklusa (kada je $I = 0$). Međutim, ako je $I = 1$ onda je izvršnom ciklusu prethodio indirektni ciklus. Tako da m i M imaju drugi smisao; sada je m sadržaj memorijske lokacije koju definiše dio 5–16 same naredbe.

Poslužimo se ranijim primjerom iz naslova 3. da ovo objasnimo. Neka treba da se izvrši naredba čiji dio 2–4 glasi 001 (naredba ADD). Neka je u datom trenutku AC 2000, 63 3000 i 3000 14500. U slučaju naredbe 0001 | 63 ima se $m = 63$ i $M = 3000$. A u slučaju naredbe 1001 | 63 ima se $m = 3000$ i $M = 14500$.

1	AND	$AC \leftarrow AC \wedge M$	bit–po–bit
2	ADD	$EAC \leftarrow AC + M$	$AC \leftarrow AC + M$, s tim da prenos najveće težine ide u E
3	LDA	$AC \leftarrow M$	iz memorije u akumulator
4	STA	$M \leftarrow AC$	iz akumulatora u memoriju
5	BUN	$PC \leftarrow m$	u programski brojač upisuje se određeni sadržaj
6	BSA	$M \leftarrow PC$, $PC \leftarrow m + 1$	ostavi dosadašnji sadržaj, a zatim upiši novi sadržaj
7	ISZ	$M \leftarrow M + 1$, if $M + 1 = 0$ then $PC \leftarrow PC + 1$	povećaj za 1 sadržaj lokacije; ako je novi sadržaj lokacije = 0 onda povećaj za 1 programski brojač

Sada bi trebalo napisati mikro–operacije za svaku od sedam naredbi, što nećemo raditi. Za naredbu ADD to je urađeno u prethodnom naslovu. Sada su sve naredbe prve vrste definisane. U nastavku navodimo primjere, dajemo komentare i govorimo o upotrebi naredbi (govorimo o namjeni naredbi).

Naredba AND. Na primjer, neka prije izvršnog ciklusa naredbe AND imamo $AC = 0110\ldots0$ i $M = 1100\ldots0$. Tada će poslije izvršavanja naredbe biti $AC = 0100\ldots0$.

Naredba ADD. Pogledajmo na primjeru izvršavanja naredbe 0 | 001 | 000000011111. Uzmimo da je prije njenog izvršavanja AC bio jednak 0...00101 i da je sadržaj lokacije 000000011111 bio jednak

0...00100. Tada će njenim izvršavanjem da postane AC = 0...01001 i E = 0. Znači da je (govoreći dekadno) 5 + 4 = 9 i da nije došlo do prekoračenja.

Dvije naredbe LDA i STA služe za komunikaciju između akumulatora i memorije (služe za komunikaciju između procesora i dijela RAM–a koji je namijenjen da čuva podatke). Pomoću naredbe LDA iz memorije se donosi neki podatak. Pomoću naredbe STA neki podatak se ostavlja u memoriju. Prevodilac programa sa višeg programskog jezika na mašinski jezik dodjeljuje svakoj promjenljivoj koja se u programu pojavljuje jednu lokaciju u memoriji, tj. prevodilac definiše "fizičku adresu" promjenljive.

Naredba BUN. Svako zna da u višim programskim jezicima postoji naredba za bezuslovni skok GOTO. Ono što postoji softverski (kao GOTO) mora prethodno da bude položeno hardverski. Znamo da sadržaj PC definiše redosljed izvršavanja naredbi. U običnom slučaju, naredbe se izvršavaju po redu. Tome odgovara inkrementacija registra PC urađena tokom prijemnog ciklusa. Međutim, ponekad treba da se odstupi od običnog redosljeda. Pomoću naredbe BUN u PC se prepiše dio 5–16 registra MBR. Znamo da će na početku sljedećeg prijemnog ciklusa (kada bude $c_0 = t_0 = 1$) to biti prepisano u MAR. Onda će u idućem taktu biti donesena odgovarajuća naredba.

Naredba BSA odgovara naredbi za prelazak iz glavnog programa u potprogram u slučaju viših programskih jezika (CALL). Ta naredba upisuje vrijednost m + 1 u programski brojač PC, čime se očito izvrši bezuslovni skok. Osim toga, ona prethodno zapamti–spasi staru vrijednost programskog brojača (zatečenu vrijednost programskog brojača). Zatečena vrijednost pošalje se na adresu m.

Zatečena vrijednost programskog brojača ima smisao povratne adrese. Spašavanjem povratne adrese stvaraju se preduslovi da kasnije (kada potprogram uradi svoje) bude obnovljeno izvršavanje glavnog programa. Od potprograma se očekuje da na kraju svog rada skoči na povratnu adresu.

Naredba ISZ. Lako se vidi da je mašinska naredba ISZ slična naredbi IF u višim programskim jezicima. Vidi se da se izvršavanje naredbe ISZ odvija u dvije faze. U prvoj fazi inkrementira se vrijednost promjenljive koja se drži na lokaciji m. Ta promjenljiva ima smisao nekog našeg brojača. U drugoj fazi, u zavisnosti od toga da li je nova vrijednost našeg brojača $\neq 0$ ili $= 0$, ne inkrementira se ili se inkrementira PC. Ukupno, na kraju će biti PC = k + 1 ili PC = k + 2, gdje sama naredba ISZ zauzima lokaciju k. Programer obično plasira na k + 1 i k + 2 dvije naredbe za bezuslovni skok.

Definicije naredbi u asemblerskoj notaciji: AND n $AC \leftarrow AC \wedge c(n)$, ADD n $AC \leftarrow AC + c(n)$, LDA n $AC \leftarrow c(n)$, STA n $c(n) \leftarrow AC$, BUN n $PC \leftarrow n$, BSA n $c(n) \leftarrow PC$, $PC \leftarrow n + 1$, ISZ n $c(n) \leftarrow c(n) + 1$, if $c(n) + 1 = 0$ then $PC \leftarrow PC + 1$, AND* n $AC \leftarrow AC \wedge c(c(n))$, ADD* n $AC \leftarrow AC + c(c(n))$, LDA* n $AC \leftarrow c(c(n))$, STA* n $c(c(n)) \leftarrow AC$, BUN* n $PC \leftarrow c(n)$, BSA* n $c(c(n)) \leftarrow PC$, $PC \leftarrow c(n) + 1$, ISZ* n $c(c(n)) \leftarrow c(c(n)) + 1$, if $c(c(n)) + 1 = 0$ then $PC \leftarrow PC + 1$, gdje je $0 \leq n \leq 4095$ i gdje $*$ znači da je adresa indirektna.

7. NAREDBE KOJE SE ODNOSE NA REGISTRE

	oznaka naredbe	smisao naredbe
1	CLA	clear AC, izbriši AC
2	CLE	clear E, izbriši E
3	CMA	complement AC, komplementiraj AC bit–po–bit
4	CME	complement E, komplementiraj E
5	CIR	circulate right E and AC, kružni desni šift (pomijeranje) nad E i AC
6	CIL	circulate left E and AC, kružni lijevi šift nad E i AC
7	INC	increment AC, inkrementiraj akumulator
8	SPA	skip if AC is positive, preskoči jednu naredbu ako je AC pozitivan
9	SNA	skip if AC is negative, preskoči jednu naredbu ako je AC negativan
10	SZA	skip if AC is zero, preskoči jednu naredbu ako je AC jednak nuli
11	SZE	skip if E is zero, preskoči jednu naredbu ako je E jednak nuli
12	HLT	halt computer, zaustavi računar

oznaka naredbe	sama naredba	naredba izražena heksadekadno	njena operacija

1	CLA	0111100...00	7800	$AC \leftarrow 0$
2	CLE	0111010...00	7400	$E \leftarrow 0$
3	CMA	...	7200	$AC \leftarrow \overline{AC}$ tj. $a_k \leftarrow \overline{a_k}$ za $k = 1, 2, \ldots, 16$
4	CME		7100	$E \leftarrow \overline{E}$
5	CIR		7080	cir EAC, v. niže
6	CIL		7040	cil EAC, v. niže
7	INC		7020	$EAC \leftarrow AC + 1$
8	SPA		7010	ako je $AC(1) = 0$ onda $PC \leftarrow PC + 1$
9	SNA		7008	ako je $AC(1) = 1$ onda $PC \leftarrow PC + 1$
10	SZA		7004	ako je $AC = 0$ onda $PC \leftarrow PC + 1$
11	SZE	...	7002	ako je $E = 0$ onda $PC \leftarrow PC + 1$
12	HLT	0111000...01	7001	$S \leftarrow 0$

$AC = a_1 a_2 \ldots a_{16}$, $a_1 = AC(1)$

Za CIR. Ako je prije izvršavanja naredbe CIR bilo E = $\boxed{e}$ i AC = $\boxed{a_1}\boxed{a_2}\boxed{a_3}\boxed{\ldots}\boxed{a_{15}}\boxed{a_{16}}$ onda će poslije njenog izvršavanja postati E = $\boxed{a_{16}}$ i AC = $\boxed{e}\boxed{a_1}\boxed{a_2}\boxed{\ldots}\boxed{a_{14}}\boxed{a_{15}}$.

Za CIL: prije izvršavanja: E $\boxed{e}$ AC $\boxed{a_1}\boxed{a_2}\boxed{a_3}\boxed{\ldots}\boxed{a_{15}}\boxed{a_{16}}$

poslije izvršavanja: E $\boxed{a_1}$ AC $\boxed{a_2}\boxed{a_3}\boxed{a_4}\boxed{\ldots}\boxed{a_{16}}\boxed{e}$

U ovom naslovu govori se o naredbama druge grupe, o naredbama koje se odnose na registre unutar procesora. Kontrola poznaje naredbu iz druge grupe po tome što naredba ima oblik 0 1 1 1 ____ ____ ____ ili kodirano heksadekadno 7___ . Tako da je $I = 0$ i $q_7 = 1$. U dijelu 5–16 definiše se o kojoj vrsti naredbe je riječ. U dijelu 5–16 pojavljuje se samo jedna jedinica.

Kažimo nešto o izvršnom ciklusu u slučaju ovih naredbi. Nećemo detaljno govoriti o postupku izvršavanja pojedine naredbe. Vidimo da se radi o vrlo jednostavnim naredbama. Tako da je za izvršavanje jedne bilo koje naredbe dovoljna jedna mikro–operacija. Bliže, naredba se izvrši sa vremenskom promjenljivom t_3. Ostale vremenske promjenljive t_0, t_1 i t_2 se ne upotrebljavaju. Drugim riječima, tokom $t_0 = 1$, $t_1 = 1$ i $t_2 = 1$ ima se prazan hod. Još se tokom $c_2 = t_3 = 1$ odlučuje o prelasku u idući ciklus, naravno.

Kod naredbe CIR vrši se desni kružni šift za jedno mjesto nad EAC tj. ovom prilikom flipflop E i registar AC treba smatrati jednom cjelinom. Slično kod naredbe CIL.

Cijeli brojevi zapisuju se u obliku PK (potpuni komplement). Tako da se znak zapisanog broja poznaje po prvom bitu a_1. Ako je $a_1 = 0$ onda se radi o pozitivnom broju ili nuli. Ako je $a_1 = 1$ onda se radi o negativnom broju. Broj koji je zapisan u akumulatoru jeste nula ako su svi bitovi $= 0$. Pogledajmo četiri naredbe SPA, SNA, SZA i SZE. Svaka od njih odgovara naredbi IF u višem programskom jeziku.

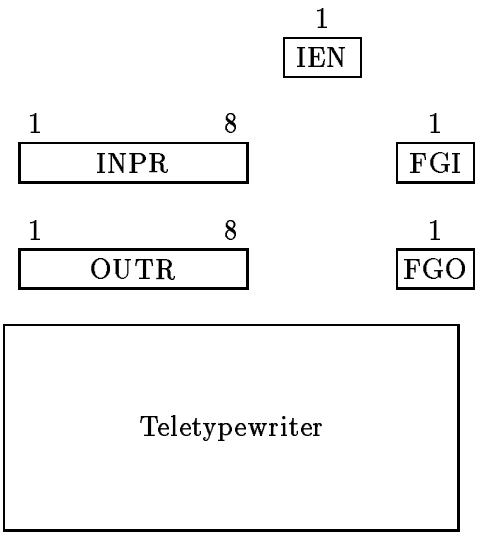
Pomoću naredbe HLT resetuje se flipflop S. Time se okonča vremenska serija.

Definicije naredbi ukratko: CLA $AC \leftarrow 0$, CLE $E \leftarrow 0$, CMA $AC \leftarrow \overline{AC}$, CME $E \leftarrow \overline{E}$, CIR circulate right $E + AC$, CIL circulate left $E + AC$, INC $AC \leftarrow AC + 1$, SPA if $AC \geq 0$ then $PC \leftarrow PC + 1$, SNA if $AC < 0$ then $PC \leftarrow PC + 1$, SZA if $AC = 0$ then $PC \leftarrow PC + 1$, SZE if $E = 0$ then $PC \leftarrow PC + 1$, HLT stop the computer.

8. ULAZNO–IZLAZNE NAREDBE I MOGUĆNOST PREKIDA

Registri za ulaz, izlaz i interapt

Slika: Ulazno–izlazni hardver:
INPR Input register, ulazni registar, 8 bita
OUTR Output register, izlazni registar, 8 bita
FGI Input flag, ulazni flag, 1 bit
FGO Output flag, izlazni flag, 1 bit
IEN Interrupt enable, flipflop za dozvolu prekida, 1 bit
Teletypewriter (teletype), teleprinter (dalekopisač): tastatura + štampač



Komercijalni računari sadrže više različitih ulaznih i izlaznih uređaja (jedinica). Mi želimo samo da ilustrujemo glavne zahtjeve za ulazno–izlaznu komunikaciju. Zato uzmimo da osnovni računar ima jedinicu teleprintera kao svoj ulazno–izlazni uređaj. Ta jedinica sadrži tastaturu koja predstavlja ulazni uređaj i sadrži štampač koji predstavlja izlazni uređaj. Ako se taster pritisne onda je jedinica u stanju da proizvede jedan niz impulsa koji je ekvivalentan binarnom kodu karaktera čiji je taster pritisnut. Taj niz biva proslijeđen u pomerački registar i tako čini ulazni karakter. S druge strane, jedna serija impulsa alfa–numeričkog koda karaktera može da bude poslata štampaču. Tamo serija biva dekodirana, da bi se odredio karakter koji treba da bude odštampan.

Teleprinter radi veoma sporo. Njegova brzina iznosi obično deset karaktera u sekundi otprilike. Teleprinter šalje i prima podatke serijski. Osam bita alfa–numeričkog koda predstavljaju jednu količinu informacije. Serijska informacija sa tastature biva upisana (biva šiftovana) u ulazni registar. Serijska informacija za štampač biva unesena u izlazni registar. Ta dva registra komuniciraju sa teleprinterom na serijski način, a sa akumulatorom paralelno.

Raspored registara prikazan je na slici. Ulazni registar INPR sastoji se od osam bita i čuva ulaznu informaciju. FGI je jedan upravljački flipflop, to je jedan jednobitni flag; flag – zastavica. Kada je nova informacija na raspolaganju u ulaznom uređaju onda je taj flag setovan. A bude resetovan kada računar prihvati informaciju. Postoji velika razlika u brzini rada između ulaznog uređaja i računara. Recimo da je ritam računara četiri reda veličine brži. Flag služi da uskladi tu razliku. Prenošenje informacije odvija se kako slijedi. Na početku rada računara ulazni flag je izbrisan (jednak nuli). Kada se pritisne taster, u INPR se upisuje kod dužine osam bita i ulazni flag se postavlja (dobija vrijednost jedan). Pored ostalog, sve dok je flag postavljen, pritisak na drugi taster ne može da promijeni sadržaj registra INPR. Dalje, računar može da pogleda vrijednost flaga. Ako je FGI = 1 onda informacija iz INPR može da bude prenesena na paralelni način u AC. Kada se ona prenese onda se FGI izbriše. Kada je flag izbrisan, nova informacija može da bude proslijeđena u INPR, pritiskom na taster. Neka na tastaturi postoji svjetlosni indikator kada je FGI = 1.

Teleprinter je štampač koji štampa na papirnu traku. Izlazni registar OUTR funkcioniše na sličan način, samo što je preokrenut smjer kretanja informacije. Na početku rada računara izlazni flag je postavljen. Računar može da pogleda vrijednost flaga FGO. Ako je FGO = 1 onda računar ima pravo da prenese informaciju iz AC u OUTR. U okviru tog prenošenja računar još i izbriše flag (uradi FGO $\leftarrow$ 0). Sada izlazni uređaj prihvata kodiranu informaciju i štampa odgovarajući karakter. Kada odštampa onda postavi flag (uradi FGO $\leftarrow$ 1). Tako da je tokom štampanja FGO = 0, što upravo i

pokazuje računaru da je štampač u datom trenutku zauzet. Tokom vremena FGO = 0 računar neće slati novu informaciju u OUTR.

Mi smo upravo opisali jedan mogući način ulazno–izlazne komunikacije. Za taj način kaže se da je programski upravljan. Program upravlja prenosom, ali se prenos izvodi samo u slučaju pripravnosti spoljašnjeg uređaja. Za prenos se obično upotrebljavaju dvije naredbe koje se ponavljaju u okviru tzv. petlje čekanja. Pomoću prve naredbe ispita se pripravnost. Zatim se pomoću druge naredbe izvrši prenos ako je uređaj pripravan ili se pak vrati na ispitivanje pripravnosti ako uređaj nije bio pripravan. Nedostatak ovog modela jeste gubitak vremena u petlji čekanja. Može se desiti da računar u cilju recimo štampanja jednog karaktera izvrši nekoliko hiljada puta radnju čitanja vrijednosti flaga FGO (provjere vrijednosti flaga FGO). Bolje bi bilo da procesor može ostaviti karakter na dogovoreno mjesto i kazati da je to karakter za štampanje. Procesor se oslobađa čekanja, on može odmah da počne da radi nešto drugo.

Taj model je moguć ako se izvrši određena hardverska dogradnja (prekidni ciklus i flipflop IEN) i određena softverska dogradnja (rutina prekida). Neka se mehanizmom komunikacije sa ulazom i izlazom upravlja od strane programa i hardvera periferne jedinice zajedno. Neka spoljašnji uređaj obavještava računara (obavještava procesora) kada je spreman za prenošenje. Kaže se da mehanizam prenošenja koristi mogućnost prekida; interrupt facility.

Čemu služi IEN? U pojedinim periodima svog rada, program se unaprijed izjasni da ne želi da vrši nikakvo ulazno ili izlazno prenošenje pomoću sistema prekida. Izjasni se putem podešavanja vrijednosti flipflopa IEN; interrupt enable – dozvola prekida. Flipflopu može da se dodijeli vrijednost mašinskom naredbom. Ako je IEN = 0 onda računar ne želi da bude prekidan. Ako je IEN = 1 onda računar dopušta da prekid nastupi, što se njega tiče. Ako je IEN = 1 i ako je još neki od dva flaga setovan onda dolazi do prekida! Bez obzira do kog mjesta se stiglo u datom trenutku u programu (u izvršavanju programa). Dakle, izvršavanje osnovnog ili pravog ili izvornog programa biva privremeno prekinuto. A počinje izvršavanje jednog drugog programa, tzv. uslužnog programa ili rutine. Zadatak uslužnog programa jeste da obavi ulaznu ili izlaznu radnju. Poslije uslužnog programa biće obnovljeno izvršavanje izvornog programa od odgovarajućeg mjesta. Tako da očito treba sastaviti i uslužni program. A izvorni program treba prilagoditi korišćenju mogućnosti prekida koje se predviđa. Drugim riječima, dva programa treba da se razumiju uzajamno i razne okolnosti treba da budu usaglašene.

Važno je i da plan upotrebe memorije bude usaglašen. Važni dio tog plana jeste kako slijedi. Prilikom prelaska sa izvršavanja izvornog programa na uslužni program, povratnu adresu upisati na lokaciju 0. Još, prilikom sastavljanja jednog i drugog programa, upisati na lokaciju 1 naredbu za skok na prvu naredbu uslužnog programa. Vidimo da dvije adrese $a = 0$ i $a = 1$ osnovnog računara imaju posebnu namjenu (imaju unaprijed definisanu namjenu). Programer neka ne upotrebljava po svome te dvije lokacije. Znamo da se adrese kreću od $a = 0$ do $a = 4095$.

Ulazno–izlazne naredbe

oznaka naredbe	naredba izražena hex	njen smisao	njena operacija
INP	F800	Input character to AC; upiši slovo u AC	AC(9–16) ← INPR, FGI ← 0
OUT	F400	Output character from AC; izdaj slovo iz AC	OUTR ← AC(9–16), FGO ← 0
SKI	F200	Skip on input flag; preskoči ako ulazni flag	ako je FGI = 1 onda PC ← PC + 1
SKO	F100	Skip on output flag; preskoči ako izlazni flag	ako je FGO = 1 onda PC ← PC + 1
ION	F080	Interrupt on; uključi mogućnost prekida	IEN ← 1 (odloženo za 16 taktova)
IOF	F040	Interrupt off; isključi mogućnost prekida	IEN ← 0

Dvije naredbe INP i OUT odnose se na donjih osam bitova akumulatora. Pomoću dvije uzastopne ulazne informacije sastavi se jedna riječ (dva bajta).

Kod naredbe SKI provjerava se vrijednost flaga FGI. Ako je FGI = 1 onda se jedna naredba preskoči, pa se zatim po pravilu izvrši naredba za učitavanje (INP). Ako je FGI = 0 onda nema

preskakanja, tako da na red za izvršavanje dolazi po pravilu naredba za skok koja vraća na ponovno provjeravanje vrijednosti flaga FGI. Svakako da je programer odlučio šta će raditi naredba koja dolazi na red. Slične okolnosti važe za naredbu SKO.

Još jednom definicije naredbi: INP $AC(9\text{–}16) \leftarrow INPR$, $FGI \leftarrow 0$, OUT $OUTR \leftarrow AC(9\text{–}16)$, $FGO \leftarrow 0$, SKI if $FGI = 1$ then $PC \leftarrow PC+1$, SKO if $FGO = 1$ then $PC \leftarrow PC+1$, ION $IEN \leftarrow 1$, IOF $IEN \leftarrow 0$.

Prekidni ciklus (Interrupt)

Prvo i prethodno, da vratimo dug. Posljednja mikro–operacija u izvršnom ciklusu ne glasi

c_2t_3: $F \leftarrow 0$

nego ona u stvarnosti glasi

c_2t_3: ako je IEN $\wedge$ (FGI $\vee$ FGO) $= 1$ onda $R \leftarrow 1$ a inače $F \leftarrow 0$

Smisao: ako je mogućnost prekida uključena IEN $= 1$ i ako je bar jedan od dva uređaja ulazni ili izlazni slobodan FGI $\vee$ FGO $= 1$ onda pređi u prekidni ciklus, a inače pređi u prijemni ciklus.

Ulazno–izlazne naredbe izvršavaju se (izvršni ciklus) kada je $t_3 = 1$. U istom taktu ($c_2 = t_3 = 1$) odlučuje se u koji će se ciklus preći iz izvršnog ciklusa (u prekidni ili u prijemni). Zato izvršavanje naredbe ION ima odloženo dejstvo; setovanje flipflopa IEN odlaže se dok se sljedeća naredba ne privede kraju (tako je definisana naredba ION). Tako da povratna adresa neće biti izgubljena.

Mikro–operacije prekidnog ciklusa su:

c_3t_0: MBR(5–16) $\leftarrow$ PC, PC $\leftarrow 0$	prenosi se zatečena vrijednost PC i briše se PC
c_3t_1: MAR $\leftarrow$ PC, PC $\leftarrow$ PC $+ 1$	u MAR se upisuje 0, u PC se upisuje 1
c_3t_2: M $\leftarrow$ MBR, IEN $\leftarrow 0$	zatečena vrijednost upisuje se na lokaciju 0, briše se IEN
c_3t_3: $F \leftarrow 0$, $R \leftarrow 0$	određuje da se pređe u prijemni ciklus

Vidimo da je namjena lokacije 0 da sačuva ili da zapamti povratnu adresu. Ta adresa biće iskorišćena kasnije prilikom nastavljanja izvršavanja izvornog programa. Vidimo da i lokacija 1 ima posebnu ulogu. Tamo treba da bude plasirana naredba za bezuslovni skok na uslužni program. Vidimo da se u okviru $c_3 = t_2 = 1$ uradi i IEN $\leftarrow 0$. Time unaprijed bivaju odbačeni (barem u datom trenutku) novi zahtjevi za prekidom koji bi eventualno mogli da dođu od jednog ili drugog flaga.

Kada se računar uključi onda je IEN $= 0$.

Primjeri za ulaz i izlaz

U ovom podnaslovu daju se primjeri ulaznog i izlaznog prenošenja u slučaju osnovnog računara.

Kod računara prve generacije, ulazno–izlazna komunikacija bila je riješena ovako: dok se ulazna naredba izvršava, procesor prosto čeka da se okonča njeno izvršavanje, a onda prelazi na iduću naredbu; isto za izlazne naredbe. Kod računara druge generacije: procesor je oslobođen čekanja, ali prije nego što naredi ulaz ili izlaz, on mora da ispita da li je ulazni odnosno izlazni uređaj slobodan; programski upravljano prenošenje. Kod računara druge i treće generacije i dalje: postoji mogućnost programski upravljanog prenošenja, a i mogućnost prenošenja pomoću sistema prekida. Tako je i kod našeg osnovnog računara. On će ponešto učitati na jedan način, a ponešto na drugi način; isto naravno prilikom štampanja.

♣ Pogledajmo prvo djelove programa i programe u kojima se koristi programski upravljano prenošenje.

Programski kontrolisano učitavanje jednog karaktera:

```
240  SKI       preskoči jednu naredbu ako je INPR pun
241  BUN 240   bezuslovni skok na prethodnu naredbu
242  INP       prepiši iz INPR u AC
```

A štampanje. Isti je šablon: prva kolona – adresa na kojoj se naredba nalazi, druga kolona – sama naredba i treća kolona – komentar:

```
260  SKO       preskoči jednu naredbu ako je OUTR slobodan
261  BUN 260   bezuslovni skok na prethodnu naredbu
262  OUT       prepiši iz AC u OUTR (štampaj karakter)
```

Primjer za ulaz–izlaz bez upotrebe prekida. Sastaviti program za osnovni računar koji štampa sadržaj dvije zadate memorijske lokacije (298 i 299), i to program ne koristi mogućnost prekida.

Rješenje. Redosljed štampanja je: donja polovina od 298, gornja polovina od 298, donja polovina od 299, gornja polovina od 299.

Biće odštampana četiri karaktera. Treba znati kakav sistem kodiranja se koristi za izlazni uređaj (štampač). Smatraćemo da je to ASCII kod. Slične okolnosti važe za rad sa ulaznim uređajem (tastaturom).

Strelica pokazuje ulaznu tačku programa. Pomoću osam naredbi za desni šift (CIR), gornja polovina akumulatora premjesti se u donju polovinu (na mjesta od 9 do 16).

```
→ 200  IOF       IEN ← 0 interrupt off
  201  LDA 298   AC ← c(298)
  202  OUT       OUTR ← AC(9–16) print
203 CIR 204 CIR 205 CIR 206 CIR 207 CIR 208 CIR 209 CIR 210 CIR right shift
right shift right shift right shift right shift right shift right shift right shift
  211  SKO       if FGO = 1 then PC ← PC + 1 skip on output flag
  212  BUN 211   PC ← 211 goto 211
  213  OUT       OUTR ← AC(9–16) print
  214  LDA 299   AC ← c(299)
  215  SKO       FGO = ?
  216  BUN 215   goto 215
  217  OUT       print
218 CIR 219 CIR 220 CIR 221 CIR 222 CIR 223 CIR 224 CIR 225 CIR
  226  SKO       FGO = ?
  227  BUN 226   goto 226
  228  OUT       print
  229  HLT       stop the computer
```

♣ Pogledajmo sada slučaj ulazno–izlaznog prenošenja koje je prekidom vođeno.

Samo se napominje da bi kao minimum minimuma trebalo dodati još četiri ulazno–izlazne naredbe, i to: prva FGI ← 0, druga AC(9–16) ← INPR, treća FGO ← 0 i četvrta OUTR ← AC(9–16), da bi sistem prekida postao osmišljen u potpunosti.

Sistem prekida računara temelji se na hardveru (hardver je dat) i na softveru (softver će sada biti definisan). Sporost perifernih uređaja predstavlja problem. Želimo da problem ublažimo što je više moguće.

Neka se karakteri koji pristižu preko tastature odmah (bez čekanja) upisuju u dio memorije koji je za to namijenjen. Za taj dio memorije kaže se da je ulazni bafer. Vidjećemo da će bafer biti organizovan kao kružni bafer (kružni red). A onda neka naš aplikativni program P preuzima iz bafera. Slične okolnosti važe za izlazno prenošenje. Naime, neka P šalje u izlazni bafer. A iz izlaznog bafera neka se šalje izlaznom uređaju (štampaču), po mjeri pripravnosti štampača, odnosno tempom koji štampač može da prati (može da opsluži).

Ključnu ulogu ima sistemska rutina R, rutina za obradu prekida, engl. interrupt service routine, ISR. Neka R zauzima adrese a u granicama $100 \leq a \leq 199$ za svoje naredbe i podatke. Od toga,

$100 \leq a \leq 109$ za ulazni bafer i $110 \leq a \leq 119$ za izlazni bafer. Neka $a = 120$ bude ulazna tačka rutine tj. adresa od koje treba da počne njeno izvršavanje. Znamo da $a = 0$ čuva vrijednost koju je programski brojač PC imao u trenutku skoka na rutinu. Tako da $a = 0$ čuva povratnu adresu; na kraju svog rada, R treba da obezbijedi da se izvršavanje programa P nastavi upravo od te adrese. Znamo da $a = 1$ sadrži naredbu koja izvodi skok na rutinu. Oba mjesta $a = 0$ i $a = 1$ su namjenska mjesta, tako je hardverski definisano. Kažimo unaprijed da će $a = 198$ i $a = 199$ biti namijenjeni za spašavanje AC i E, prilikom skoka na rutinu. Prilikom povratka iz rutine, u registre AC i E biće upisane dvije spašene vrijednosti. Da bi se izvršavanje programa P obnovilo u istim okolnostima kakve su bile prije prekida. Da asinhroni događaj (skok na rutinu) ne bi ništa poremetio u računanju koje je u toku u trenutku nastupanja asinhronog događaja.

Za ulazni bafer, $c(100) = \text{ib}$ i $c(101) = \text{in}$. ib = input begin = mjesto gdje se nalazi prvi (po logičkom redosljedu) član bafera. in = input number = broj članova bafera. Sami članovi bafera imaju 8 lokacija namijenjenih, od $a = 102$ do $a = 109$. To znači da je uvijek $0 \leq \text{in} \leq 8$. Ako je $\text{in} = 0$ onda je bafer prazan. Ako je $\text{in} = 8$ onda je bafer pun. Biće uvijek $102 \leq \text{ib} \leq 109$, osim ako je bafer prazan. Poslije 105 dolazi 106 očito. Smatra se da poslije 109 dolazi 102. Prvi član bafera je davno nekad upisan u bafer i on će prvi (gledano po vremenskoj osi) izaći iz bafera. Ako je bafer prazan onda neka bude $\text{ib} = \text{in} = 0$. Ponovimo, R puni bafer (stavlja u bafer), a P prazni bafer (uzima iz bafera). Ako je recimo $\text{ib} = 102$ i $\text{in} = 4$ onda to znači da je stanje u baferu sljedeće. Ima četiri člana i to su oni redom na adresama 102, 103, 104 i 105. Ako $\text{ib} = 104$ i $\text{in} = 4$. Četiri člana i to na 104, 105, 106 i 107 redom. Ako $\text{ib} = 108$ i $\text{in} = 4$. Četiri člana i to na 108, 109, 102 i 103. Prilikom svakog upisivanja jednog člana u bafer treba (osim samog upisivanja člana) još i ažurirati vrijednosti ib i in. Slično naravno i prilikom bilo kog uzimanja člana iz bafera. Lako se vidi da je bafer organizovan kružno da bi se za njega raspoloživi memorijski prostor (8 lokacija, za 8 članova) maksimalno koristio, a da se ne mora premještati članove kod bilo kog dodavanja ili oduzimanja člana.

Slične okolnosti važe i za izlazni bafer. Dakle, izlaznom baferu je dato 10 lokacoja $110 \leq a \leq 119$. Od toga dvije lokacije služe za evidenciju: ob = output begin = adresa prvog člana i on = output number = tekući broj članova, a preostale služe za same članove. Program P stavlja u izlazni bafer karaktere koji treba da budu odštampani. Rutina R uzima iz bafera i šalje štampaču. P ažurira što P uradi, R ažurira što R uradi.

Sada smo u mogućnosti da opišemo radnje koje rutina R treba da obavlja, koraci a)–f), mada neće biti dati svi detalji tj. neće biti napisan program R kao takav.

a) Zatečene vrijednosti AC i E treba da budu spašene. Zapiši te dvije vrijednosti negdje u memoriju. Da bi se kasnije izvršavanje programa P obnovilo u okolnostima koje se nisu izmijenile. Prvo spašavamo AC a zatim i E:

STA 198	$c(198) \leftarrow$ AC akumulator je spašen
CLA	AC $\leftarrow 0$ izbriši AC
CIL	left shift E + AC, tako da $a_{16} \leftarrow e$
STA 199	$c(199) \leftarrow$ AC flipflop E je spašen

Na redu je dio posla b)–c) koji predstavlja zadatak rutine R u užem smislu riječi.

Slovo c dolazi od engl. contents, što znači sadržaj.

Uobičajeno je da se pomoću znaka $*$ označava da je indirektna adresa, za razliku od običnog načina adresiranja.

b) Ako je FGI = 0 onda pređi na c). Ako je in = 8 ulazni bafer je pun onda izvrši naredbu za zvučni signal i pređi na c). Vidimo da ovdje uvodimo novu mašinsku naredbu – naredbu za zvučni signal, obično se ona naziva "beep". Sljedeće, ako je FGI = 1 ulazni uređaj ima nešto da kaže i ako je još $\text{in} < 8$ ulazni bafer nije pun (u njemu ima mjesta) onda R preuzima jedan karakter. Sada bi trebalo napisati odgovarajuće naredbe. Poslije izvršavanja tih naredbi, u ulaznom baferu imaćemo jedan član više. Nećemo pisati odgovarajuće naredbe u opštem slučaju, već samo pogledajmo dva posebna slučaja, da bi bar nešto bilo jasno.

Ako je bilo in = 0 (bafer je bio prazan) onda treba uraditi:

LDA 197	AC $\leftarrow c(197) = 102$
STA 100	$c(100) \leftarrow$ AC, ib je podešen
CLA	AC $\leftarrow 0$
INC	AC $\leftarrow$ AC $+ 1$
STA 101	$c(101) \leftarrow$ AC, in je podešen
INP	AC $\leftarrow$ INPR učitavanje karaktera
STA 102	$c(102) \leftarrow$ AC karakter je stavljen u bafer

s tim da se na adresi 197 nalazi odranije upisana brojna vrijednost 102. Uradili smo: ib $\leftarrow 102$, in $\leftarrow 1$ i $c(102) \leftarrow$ character.

Čisto za ilustraciju, ako je bilo in > 0 bafer nije prazan i ib $+$ in ≤ 109 kružno svojstvo sada ne utiče onda treba uraditi: $c(\text{ib} + \text{in}) \leftarrow$ character i in $\leftarrow$ in $+ 1$, a ib se ne mijenja:

LDA 100	AC $\leftarrow c(100)$, AC $\leftarrow$ ib
ADD 101	AC $\leftarrow$ AC $+ c(101)$, AC $\leftarrow$ AC $+$ in
STA 196	$c(196) \leftarrow$ AC, adresa je pripremljena
LDA 101	AC $\leftarrow c(101)$, AC $\leftarrow$ in
INC	AC $\leftarrow$ AC $+ 1$
STA 101	$c(101) \leftarrow$ AC, in je podešen
INP	AC $\leftarrow$ INPR učitavanje karaktera
STA* 196	$c(c(196)) \leftarrow$ AC karakter je stavljen u bafer

c) Ako je FGO $= 0$ štampač je zauzet onda pređi na d). Ako je FGO $= 1$ štampač je slobodan (pripravan) i on $= 0$ izlazni bafer je prazan onda pređi na d). Ako je FGO $= 1$ štampač je slobodan i on > 0 izlazni bafer nije prazan onda uradi sljedeće radnje. Pošalji odgovarajući karakter štampaču (naredba OUT) i zatim ažuriraj ob i on, pređi na d).

d) Vratiti dvije spašene vrijednosti (iz 198 i 199) u AC i E. Prvo E onda AC.

e) Naredba ION čiji je efekat IEN $\leftarrow 1$, da bi ostale iste okolnosti, jer je u trenutku prelaska na rutinu očito bilo IEN $= 1$.

f) Naredba BUN* 0 čiji je efekat PC $\leftarrow c(0)$, odnosno obnavlja se izvršavanje programa P.

Još treba reći o uslovima koje P treba da ispunjava da bi se uklopio u ukupnu šemu prekida. Dakle, program P rješava svoju ulaznu i izlaznu komunikaciju u potpunosti ili djelimično pomoću sistema prekida. O njemu je ustvari već sve praktično rečeno, jer znamo da on tada za komunikaciju mora da se povinuje šemi koju diktira sistemski program R (rutina R koja pripada sistemskom softveru, pripada operativnom sistemu). Zapaziti da u P nema naredbe za ulaz INP nego umjesto nje stoji LDA, preuzimanje iz ulaznog bafera u AC (ako ima šta da se preuzme). Slično, u programu P nema naredbi OUT već umjesto njih ima prosto STA za stavljanje u izlazni bafer karaktera koji će biti odštampani (rutina će ih odštampati). Posebno, prije stavljanja vidi da se ne prepuni (overflow), tj. vidi da nije bafer već pun, u tom slučaju bi se sačekalo dok se ne oslobodi mjesto u izlaznom baferu.

Ulazni bafer je prazan:

100	101	102	103	104	105	106	107	108	109
0	0								
ib	in								

Izlazni bafer sadrži četiri člana:

110	111	112	113	114	115	116	117	118	119
112	4	član	član	član	član				
ob	on								

Dvije adrese u memoriji:

$a = 0$		mjesto za spašavanje PC
$a = 1$	BUN 120	naredba za skok na rutinu

Tokom prekidnog ciklusa uradi se:

$\left\{ \begin{array}{l} c(0) \leftarrow \text{PC (spašavanje) i} \\ \text{PC} \leftarrow 1 \text{ (skok)} \end{array} \right.$

Primjer: ulaz–izlaz pomoću pogodnosti (facility) prekida. Sastaviti program P za osnovni računar koji želi da obavi dva posla za što je moguće kraće ukupno vrijeme. Prvi posao: da odštampa donje polovine lokacija od 296 do 299 (četiri lokacije ukupno). Za štampanje neka se koristi rutina za obradu prekida R. Drugi posao: da izvrši niz aritmetičkih i logičkih operacija, puno operacija ima.

Rješenje:

```
    1   BUN 120   goto 120, stvoreni su preduslovi za skok na R
→ 200   IOF       IEN ← 0, interrupt off
  201   CLA       AC ← 0, clear AC
  202   STA 100   c(100) ← AC, ib ← 0
  203   STA 101   c(101) ← AC, in ← 0 (ulazni bafer je prazan)
  204   LDA 296   AC ← c(296), počinje upisivanje u izlazni bafer
  205   STA 112   c(112) ← AC
  206   LDA 297   AC ← c(297)
  207   STA 113   c(113) ← AC
  208   LDA 298   AC ← c(298)
  209   STA 114   c(114) ← AC
  210   LDA 299   AC ← c(299)
  211   STA 115   c(115) ← AC, četiri vrijednosti su upisane u izlazni bafer
  212   LDA 294   AC ← c(294) = 112
  213   STA 110   c(110) ← AC, ob ← 112, ažuriranje
  214   LDA 295   AC ← c(295) = 4
  215   STA 111   c(111) ← AC, on ← 4, ažuriranje
  216   ION       IEN ← 1, interrupt on
  217   ADD 254   počinju aritmetičke i logičke operacije
  218   ADD 253
  219   AND 252
  220   AND 251
  .........
  294   DEC 112   ova lokacija neka sadrži brojnu vrijednost
  295   DEC 4     ova lokacija neka sadrži brojnu vrijednost
```

Ono što piše od "rješenje" dovde ima sljedeći smisao. Na naznačene adrese 1, 200, 201, ... loadovati naznačene naredbe BUN 120, IOF, CLA, ... U adrese 294 i 295 loadovati naznačene brojne vrijednosti 112 dekadno, odnosno 4 dekadno, DEC znači dekadno. Startovati računar od adrese 200 (strelica). Još, podrazumijeva se da je i rutina R loadovana (zauzima $100 \leq a \leq 199$).

Diskusija o sistemu prekida osnovnog računara

Rekli smo da osnovni računar ima dva periferna uređaja i da svaki uređaj ima svoj flag, to su FGI i FGO. Ako bi bila četiri uređaja tada bi bila četiri flaga FG1 do FG4. Tokom prekidnog ciklusa neke naredbe, izvršiće se skok na R ako je makar jedan flag podignut i ako je još IEN = 1.

Mogući dodatni periferni uređaji su linijski štampač i jedinica spoljašnje memorije (sa magnetnom trakom).

Preko FG1 do FG4 uređaji ispostavljaju procesoru svoje zahtjeve za prekid (zahtjeve da budu opsluženi). Može se desiti da u istom trenutku ima više zahtjeva. Da li postoji hijerarhija, tj. da li je definisan redosljed prioriteta među raznim vrstama zahtjeva? O tome se može govoriti jedino ako je rutina R sastavljena tako da među tim zahtjevima izdvoji jedan zahtjev i opsluži samo taj jedan izabrani zahtjev (a ostale ostavi da čekaju).

Tokom rada rutine, stalno je IEN = 0. Zaista, tokom prekidnog ciklusa urađeno je IEN $\leftarrow$ 0, kao što znamo. A u samom tekstu rutine nema naredbe ION, osim na kraju teksta. Kaže se da je sistem prekida isključen tokom rada rutine (tokom rada rutine za obradu prekida). Znači, zahtjevi za prekid koji se eventualno pojave tokom rada rutine biće odbačeni. Drugim riječima, rutina neće biti prekidana.

Kod mnogih računara postoje posebne rutine za različite vrste prekida. U opštem slučaju, dopušteno je i da rutina bude prekinuta. Prilikom prekidanja rutine, spasiti procesorove registre (kao AC i E) i spasiti povratnu adresu (kao $c(0)$). To zajedno čini jednu cjelinu koja se spašava. Prekidanju jedne rutine odgovara jedna cjelina koja se spašava. Cjeline stavljati na stek, a kod kasnijeg povratka skidati sa steka. Ukupna šema je nešto složenija od onog kako je rađeno.

Novi izlazni uređaj: linijski štampač

U ovom podnaslovu pogledajmo šta piše u priručniku za računar PDP–5, iz 1964. godine, o automatskom linijskom štampaču i njegovom kontroleru.

Linijski štampač može da odštampa 300 linija od po 120 karaktera u minutu. Svakom karakteru odgovara njegov 6–bitni kod. Svaki karakter pojedinačno se loaduje u printerov bafer iz AC[6–11]. Printerov bafer podijeljen je u dvije sekcije, svaka sekcija može da sadrži 120 kodova. Prema tome, treba da bude izvršeno 120 load naredbi da bi se jedna sekcija štampačevog bafera popunila. Neka program broji koliko je izdato takvih naredbi (LLD) i neka pamti, da bi znao u kojoj mjeri je sekcija popunjena. Naredba za štampanje LPR izaziva da se kodovi specifikovani sadržajem štampačevog bafera posljednja loadovana sekcija odštampaju u jednoj liniji. Dok štampanje napreduje, alternativna sekcija štampačevog bafera može da bude loadovana iznova. Kada se odštampa posljednji karakter linije, sekcija štampačevog bafera iz koje su karakteri upravo odštampani biva automatski izbrisana. Sekcija štampačevog bafera u koju se loaduje i iz koje se štampa smjenjuje se automatski unutar štampača i ne specifikuje se programom. Loadovanje 6–bitnog koda u štampačev bafer potroši približno 1,6 milisekundi. Kada se loadovanje (prenošenje) jednog koda završi, podigne se flag linijskog štampača čime se pokazuje da je štampač spreman da primi sljedeći kod. Flag linijskog štampača konektovan je sa mogućnošću prekida programa. Na redu su ulazno–izlazne naredbe koje upravljaju linijskim štampačem:

LCF Clear line printer flag
LPR Print the line
LSF Skip if line printer flag is a 1
LCB Clear both sections of the printing buffer
LLD Load printing buffer from the AC[6–11]

LCF – Izbriši flag linijskog štampača.

LPR – Odštampaj liniju sadržanu u onoj sekciji štampačevog bafera koja je posljednja loadovana i neka papir napreduje (neka se papir pomjeri po visini), u zavisnosti od AC[8–9], kako je opisano pravilo u nastavku. Ako je AC[8–9] = 00 onda nema napredovanja, ako je AC[8–9] = 01 onda pomjeri papir za jedan red (ovo je obični slučaj), ako je AC[8–9] = 10 onda pomjeri papir za dva reda, a ako je AC[8–9] = 11 onda neka se papir postavi na početak iduće strane.

LSF – Preskoči jednu naredbu ako je flag linijskog štampača = 1.

LCB – Izbriši obe sekcije štampačevog bafera.

LLD – Loaduj u štampačev bafer iz AC[6–11].

Svaka od pet naredbi ima svoj kod (16–bitni). Ako bi se osnovni računar hardverski proširio i bilo bi dodato pet navedenih vrsta naredbi onda bi on mogao da koristi i linijski štampač; dodati i sami štampač i štampačev kontroler.

9. Završni pogled na osnovni računar, o njegovoj izradi

Osnovni računar sastoji se od memorijske jedinice, teleprintera, glavnog časovnika, osam registara, osam flipflopova, tri dekodera i jednog broja upravljačkih logičkih kola (upravljačkih logičkih mreža).

Memorija i teleprinter su standardne jedinice koje mogu da budu nabavljene kao gotovi proizvodi. Registri su AC, PC, MAR, MBR, OPR, SC, INPR i OUTR. Flipflopovi su F, R, E, I, S, IEN, FGI i FGO. Glavni časovnik je obični taktni izvor. Obično je to oscilator koji generiše periodičnu povorku impulsa. Ti impulsi razvode se kroz čitav sistem posredstvom bafera i invertora. Svaki impuls mora da dostigne svaki registar i svaki flipflop u istom trenutku.

Ostale komponente računara su registri i dekoderi koji mogu da se nabave kao standardna MSI integrisana kola. Ostale komponente flipflopovi i mreže mogu da se nabave kao standardna SSI integrisana kola.

Za računar je razvijen spisak izjava o registarskom prenošenju. Na osnovu tog spiska moguće je definisati registre, flipflopove i mreže potrebne za izradu računara. Uzmimo mali konkretni primjer da ilustrujemo. Registar operacija OPR ima tri bita. Njemu je pridružena izjava o registarskom prenošenju

$$c_0 t_2\text{: OPR} \leftarrow \text{MBR}(2\text{–}4)$$

Po ovoj izjavi vidimo da računaru treba jedno AND kolo čiji ulazi c_0 i t_2 dolaze od ciklusnog dekodera, odnosno od vremenskog dekodera. Izlaz iz AND kola treba da se dovede na Load ulaz trobitnog registra koji ima sposobnost paralelnog loadovanja. Ulazi za podatke u registar OPR su bitovi 2, 3 i 4 registra MBR.

Neposredno prije uključivanja u rad (neposredno prije početka izvršavanja programa) treba podesiti bar neke registre i flipflopove, inicijalizacija. Recimo, brojač taktova SC podesiti na nulu, flipflop S setovati, resetovati F i R, itd.

Stvarni računar PDP–8 razlikuje se od našeg modela osnovnog računara najviše po tome što PDP–8 ima i jedinicu spoljašnje memorije, odgovara hard–disku. Tako da svakako ima i odgovarajuće naredbe, tj. mašinski jezik je proširen. Magnetna traka predstavlja jedinicu spoljašnje memorije.

10. PRIMJERI APLIKATIVNIH PROGRAMA ZA OSNOVNI RAČUNAR

1. Aritmetički program. Sastaviti program za osnovni računar za računanje vrijednosti izraza $y = 2a + b + 1$.

U nastavku je prikazano stanje u memoriji u trenutku kada počinje izvršavanje programa, s tim da izvršavanje treba da počne od adrese 200. Tako da se pretpostavlja da je program već loadovan na 200–205, a također da je i njegov radni prostor 210–212 popunjen. Rezultat y ostaje na 212.

$\rightarrow$	200	LDA 210	20D2	AC $\leftarrow a$ or AC $\leftarrow c(210)$
	201	CIL	7040	left shift AC
	202	ADD 211	10D3	AC $\leftarrow$ AC $+ b$ or AC $\leftarrow$ AC $+ c(211)$
	203	INC	7020	AC $\leftarrow$ AC $+ 1$ or increment AC
	204	STA 212	30D4	$y \leftarrow$ AC
	205	HLT	7001	stop the computer or halt
	210	DEC 300	012C	$a = 300$
	211	DEC 44	002C	$b = 44$
	212	DEC 0	0000	initially $y = 0$ (finally $y = 645$)

Vidimo da je $a = 300$ i $b = 44$. Na kraju će biti $y = 645$.

Program je prikazan na simboličkom jeziku, gdje DEC x znači sadržaj dekadno x. Zatim je isti program prikazan i u heksadekadnom obliku, što je vrlo blisko stvarnom binarnom sadržaju lokacija. Posljednje su komentari.

$\rightarrow$ 200 znači da na 200 počinje izvršavanje.
$c(n)$ znači sadržaj lokacije n.
$210 = (\mathrm{D2})_{16}$, $300 = (\mathrm{12C})_{16}$.

Na primjer, sadržaj lokacije 200 izražen heksadekadnim sredstvima glasi 20D2, a stvarni sadržaj te lokacije jeste 0010 0000 1101 0010.

2. Uslovni skok (naredba IF). Sastaviti program za osnovni računar za računanje vrijednosti izraza

$$y = \begin{cases} a + b + 10 \text{ ako je } c < 0 \\ a^2 + b^2 + 100 \text{ ako je } c \geq 0 \end{cases}$$

Napraviti raspored za memoriju, tj. napraviti plan upotrebe memorije. Drugim riječima, odrediti koje lokacije služe za naredbe, a koje služe za promjenljive (za brojne vrijednosti). Definisati ulaznu tačku programa, gdje počinje izvršavanje, $\rightarrow$.

Sami za vježbu.

Treba isprogramirati množenje.

Vidimo da zasad ostaju otvorena dva pitanja: a) kako loadovati naredbe i brojne vrijednosti i b) kako ostvariti start na definisanu ulaznu tačku. L.

3. Petlja. Sastaviti program za osnovni računar za računanje vrijednosti izraza $y = \sum_{k=a}^{b} k$, gdje su a i b ulazni podaci.

Sami za vježbu. Važe slične okolnosti kao u prethodnim primjerima.

4. Rad sa nizom. Sastaviti program za osnovni računar koji ilustruje rad sa nizom brojnih vrijednosti. Recimo, sa nizom od 10 brojeva $a_1, a_2, \ldots, a_{10}$ koji zauzimaju redom adrese $1001, 1002, \ldots, 1010$. Neka program učitava vrijednosti članova niza i upisuje učitane vrijednosti u memoriju na 1001–1010. Ili neka program računa a_k po nekoj formuli i upisuje a_k u memoriju. Ili neka program računa $y = \sum_{k=1}^{10} a_k$.

Uputstvo. Koristiti naredbu STA sa indirektnom adresom. Na simboličkom jeziku bismo rekli: naredba STA$*$. Ta naredba glasi B___ (hex). Recimo, naredba može da glasi STA$*$ 1000, gdje kroz adresu 1000 prolaze vrijednosti $1001, 1002, \ldots, 1010$. Koristiti i naredbu LDA sa indirektnom adresom (LDA$*$).

ADD n AC $\leftarrow$ AC + $c(n)$. ADD* n AC $\leftarrow$ AC + $c(c(n))$. BUN n PC $\leftarrow n$. BUN* n PC $\leftarrow c(n)$.

Ponovimo spisak svih naredbi osnovnog računara:

prva vrsta AND or AND* 0___ or 8___, ADD or ADD* 1___ or 9___, LDA or LDA* 2___ or A___, STA or STA* 3___ or B___, BUN or BUN* 4___ or C___, BSA or BSA* 5___ or D___, ISZ or ISZ* 6___ or E___

druga vrsta CLA 7800, CLE 7400, CMA 7200, CME 7100, CIR 7080, CIL 7040, INC 7020, SPA 7010, SNA 7008, SZA 7004, SZE 7002, HLT 7001

treća vrsta INP F800, OUT F400, SKI F200, SKO F100, ION F080, IOF F040

5. Rad sa potprogramom. Sastaviti program za osnovni računar za računanje vrijednosti izraza $y = (a/16 + b)/16$, gdje je prisutan potprogram za dijeljenje sa 16.

Rješenje. Ulazna tačka je 200. Potprogram 250–255 vrši radnju AC $\leftarrow$ AC/16:

$\rightarrow$	200	LDA 206	AC $\leftarrow c(206)$ or AC $\leftarrow a$
	201	BSA 250	call: $c(250) \leftarrow 202$ and PC $\leftarrow 251$
$\rightarrow$	202	ADD 207	AC $\leftarrow$ AC + $c(207)$ or AC $\leftarrow$ AC + b
	203	BSA 250	call: $c(250) \leftarrow 204$ and PC $\leftarrow 251$
$\rightarrow$	204	STA 208	$c(208) \leftarrow$ AC or $y \leftarrow$ AC
	205	HLT	stop the computer
	206	DEC 900	$a = 900$
	207	DEC 800	$b = 800$
	208	DEC 0	initially $y = 0$, finally $y = 53$
	250	DEC 0	place for return address
$\rightarrow$	251	CIR	right shift
	252	CIR	right shift
	253	CIR	right shift
	254	CIR	right shift
	255	BUN* 250	return: PC $\leftarrow c(250)$ i.e. goto $c(250)$

Dopuna. Svakako da u opštem slučaju i potprogram ima svoj radni prostor. U našem primjeru, on ga nema (osim 250 za povratnu adresu). U našem primjeru, radni prostor glavnog programa je 206–208.

U našem primjeru, razmjena parametara između glavnog programa i potprograma vrši se pomoću AC. Postoje i drugi načini da se razmjena ostvari. Neka se za razmjenu izdvoji nekoliko adresa u okviru potprograma. Ili neka za razmjenu služe neke fiksirane adrese (zajednička zona, čije su adrese apsolutne).

11. PRIMJERI SISTEMSKIH PROGRAMA ZA OSNOVNI RAČUNAR

1. Apsolutni loader L

Program L jednostavno popunjava memorijske lokacije. Lokacije su specifikovane pomoću svojih fizičkih (apsolutnih) adresa.

Neka loader zauzima lokacije 0–99. Vidimo da koristi (nenamjenski) i dvije namjenske lokacije 0 i 1. Tokom njegovog rada neće se koristiti mogućnost prekida. Uostalom, važi IEN = 0 u trenutku uključivanja računara.

Znamo da je ukupni zadatak loadera da popuni lokacije i onda da preda štafetu (da preda upravljanje na specifikovanu adresu).

Za ukupnu šemu rada računara pretpostavlja se da važe sljedeće dvije okolnosti. Prva okolnost: u trenutku uključivanja računara prostor 0–99 već je popunjen. Upisivanje loadera ne obavlja se programski! Druga okolnost: u trenutku uključivanja računara važi PC = 0.

Poslužimo se primjerom 1 iz prethodnog naslova da izrazimo kako treba da glase ulazni podaci loadera. Za taj aritmetički program ulazni podaci su:

Učestvuju vrijednosti $200 = (C8)_{16}$, $210 = (D2)_{16}$, $200 = (C8)_{16}$. Ulazni podaci su:

00C8 loaduj na $200, 201, \ldots$
20D2 7040 10D3 7020 30D4 7001 ono što se loaduje
7777 završen odsječak
00D2 loaduj na $210, 211, \ldots$
012C 002C 0000 ono što se loaduje
7777 završen odsječak
FFFF nema više
00C8 štafeta (PC $\leftarrow 200$)

Sada treba definisati izgled ulaznih podataka loadera u opštem slučaju. Vidimo da loader redom čita cjeline, on čita u grupama od po četiri karaktera. Prva cjelina definiše mjesto gdje počinje popunjavanje, a iduće cjeline naravno definišu sami sadržaj koji treba da bude upisan u memoriju. Tako se nastavlja sve dok se ne pročita cjelina 7777. Znači da je završeno popunjavanje jednog odsječka u memoriji. Program L je u stanju da popuni više odsječaka. FFFF znači da su svi odsječci preuzeti. Još ostaje jedino da se preda štafeta. Drugim riječima, sa zadnjom cjelinom definiše se ulazna tačka aritmetičkog programa P. Dakle, na kraju svog rada L će izvršiti jedan bezuslovni skok.

Sada ćemo da napišemo kako glasi L na jednom nestrogom jeziku koji je blizak paskalu. l dolazi od load point (tačka punjenja), a e dolazi od entry point (ulazna tačka):

1: read four; if four = FFFF then goto 3;
   $l \leftarrow$ four;
2: read four; if four = 7777 then goto 1;
   $c(l) \leftarrow$ four; $l \leftarrow l + 1$; goto 2;
3: read four; PC $\leftarrow$ four ($e \leftarrow$ four)

Korisno je napisati pravi tekst programa, tj. riješiti postavljeni zadatak u pravom smislu riječi. Znamo da učitati karakter preko ulaznog uređaja znači upisati odgovarajuću brojnu vrijednost u ulazni registar INPR, gdje ASCII tabela definiše korespondenciju između karaktera i brojne vrijednosti.

Ne smije da se zada $l < 100$. Kada je štafeta predata onda prostor 0–99 može da bude prebrisan (može slobodno da se koristi).

Govoreći uopšteno, loader dobija svoje ulazne podatke preko ulaznog uređaja ili ih preuzima od jedinice spoljašnje memorije.

2. Loader koji vrši relokaciju

Razmotrimo program L_r koji vrši premještanje, odnosno koji loaduje u relativnim adresama. Za svaki odsječak navodi se i iznos pomijeranja Δ. Ako je $\Delta = 0$ onda se L_r svodi na L. L_r ne vrši jednostavno prepisivanje. L_r ne upisuje u memoriju ono što je preuzeo preko ulaznog uređaja. Umjesto toga, L_r uvećava adresno polje 5–16 svake naredbe prve vrste za Δ. Dodati Δ naredbi ADD znači uzeti drugi sabirak y sa adrese $a + \Delta$, a ne više sa adrese a. Dodati Δ naredbi BUN znači izvršiti bezuslovni skok na $a + \Delta$, a ne više na a. Navedimo jedan običan razlog kada je relokacija korisna. Više stvari utiče na mjesto koje će u memoriji pripasti potprogramu: veličina glavnog programa, veličine drugih potprograma, kao i eventualne zajedničke zone za podatke. L je u stanju da reguliše premještanje potprograma, ali je tek L_r u stanju da reguliše prilagođavanje adresnih polja 5–16, da bi smisao potprograma ostao isti. Da ne bismo morali svaki put ponovo da pišemo tekst potprograma.

Kako treba da budu pripremljeni ulazni podaci za program L_r? Za L, pojedini odsječak specifikovao se pomoću sljedećeg niza cjelina (order = naredba):

l order order ... order 7777

Za L_r, niz cjelina za jedan odsječak glasi (ima jedna cjelina više):

l Δ order order ... order 7777

Neka kao primjer opet posluži raniji aritmetički program. Učestvuju $\Delta = (0010)_{16} = 16$, $\Delta = (0000)_{16} = 0$ vrijednosti. Neka ulazni podaci za program L_r glase

00C8 0010 20D2 ... 7001 7777 00D2 0000 012C 002C 0000 7777 FFFF 00C8

Šta će se dobiti u rezultatu izvršavanja programa L_r sa ovim ulaznim podacima? Razlika u odnosu na prethodni slučaj loadovanja je sljedeća: na mjesta 200–205 upisaće se 20E2 ... 7001. Ili (govoreći po smislu)

umjesto ranijeg	200	LDA 210	sada je	200	LDA 226
	201	CIL		201	CIL
	202	ADD 211		202	ADD 227
	203	INC		203	INC
	204	STA 212		204	STA 228
	205	HLT		205	HLT

Na nestrogom jeziku, jedan dio programa L_r glasi kako slijedi (ovdje je four $= x_1x_2x_3x_4$):

if $x_1 \neq 7$ & $x_1 \neq$ F then $c(l) \leftarrow$ four $+ \Delta$ else $c(l) \leftarrow$ four

Korisno je i da se napiše rješenje u pravom smislu riječi.

3. Bootstrap loader

Govoreći uopšteno (nevezano za osnovni računar), razmotrimo program koji pokreće računara, isto se kaže i startni loader ili inicijalni loader (inicijalna kapisla). Taj program djeluje kada je memorija prazna, tj. kada je u memoriji samo on prisutan. Čuva se u ROM–u i uzima upravljanje kada dugme "on". Šta radi bootstrap loader? Najviše, bootstrap loader učita (loaduje) jedan malo jači program i preda mu štafetu. Šta radi taj malo jači program? Provjerava periferne uređaje, inicijalizuje registre, nešto računa i nešto loaduje. Itd.

Vidimo da se pokretanje računara izvodi u dvije–tri faze.

U slučaju DOS, bootstrap loader učita tzv. boot block (to je prvi sektor na disku) i preda štafetu.

4. Program za obradu prekida R. Interrupt Service Routine

O ovoj rutini je već bilo dosta riječi, v. naslov 8.

5. Asembler A

V. idući naslov.

6. Interpreter ili debager D

To je jedan program na jeziku osnovnog računara. Njegov zadatak je da pomaže u otkrivanju grešaka u nekom programu P (u bilo kom programu P) na jeziku osnovnog računara. To se postiže time što se P izvršava na kontrolisani način (P se izvršava pod kontrolom programa D). Kaže se da se P izvršava korak po korak.

Dakle, pored programa D, tokom rada debagera D, u memoriji se nalazi upisan i program P, a i ulazni podaci programa P biće naravno učitani kada za to dođe vrijeme.

Tokom rada debagera D stalno se smjenjuju dvije faze rada: faza oponašanja ili interpretacije i faza proučavanja ili analize. U fazi oponašanja učini se da se jedna naredba programa P izvrši. U fazi proučavanja gleda se kakve su okolnosti nastupile (gleda se trag, engl. trace) i štampaju se neke informacije preko izlaznog uređaja ili se te informacije šalju na određeno mjesto u memoriji (da bi se obrazovao izvještaj).

Možda staviti da P ne koristi mogućnost prekida.

12. JEZIK ASEMBLERA ZA OSNOVNI RAČUNAR

U ovom naslovu govori se u glavnim crtama o važnom sistemskom programu za osnovni računar – o asembleru (prevodiocu). Zadatak tog programa A (asembler) jeste da generiše mašinski kod (program na mašinskom jeziku) P_2 koji odgovara ulaznom podatku P_1, P_1 je izražen na jeziku asemblera, Počinjemo sa primjerom programa P_1.

Zadatak. Sastaviti program P_1 na jeziku asemblera osnovnog računara za računanje vrijednosti izraza $y = 2a + b + 1$.

Rješenje. V. *Tabelu 1.*

Sintaksa jezika asemblera? Pojedina naredba jezika asemblera prikazuje se u posebnom redu. Naredba može da bude pseudo–naredba (asemblerska direktiva) ili obična naredba. Postoje četiri vrste pseudo–naredbi ORG, END, DEC i HEX. ORG n govori da P_2 treba da zauzima adrese n, n + 1, itd, origin. END prosto označava kraj teksta programa P_1. Naredba DEC n govori da (na odgovarajućem mjestu u P_2) treba da piše sadržaj koji odgovara $(n)_{10}$. HEX ... $(n)_{16}$. Običnih naredbi ima 25 vrsta, koliko ima i vrsta mašinskih naredbi. Pojedina naredba jezika asemblera sastoji se od tri polja, odnosno pojedina naredba glasi label, opcode address. U primjeru u naredbi A, DEC 300 čiji je smisao a $\leftarrow$ 300 labela je A, kaže se da je A simboličko ime adrese 206. Ako se u P_1 pojavljuje naredba recimo BUN MJESTO (bezuslovni skok) onda će se u P_1 pojaviti po svoj prilici i naredba čija je labela MJESTO. Dio naredbe label, može i da odsustvuje. Za drugo polje opcode obične naredbe postoje mogućnosti AND, AND*, ADD, ADD*, ..., CLA, CLE, ..., INP, OUT, ... U slučajevima CLA, CLE, ... naredbe druge vrste i INP, OUT, ... naredbe treće vrste odsustvuje treće polje address. Mogući primjeri naredbi koje imaju treće polje su AND A i AND* A i BUN MJESTO i BUN* MJESTO. Ovdje su A i MJESTO mogući primjeri tzv. simboličkih adresa.

Zadatak. Neka se program A izvršava i neka je pritom P_1 njegov ulazni podatak. Šta će se dobiti kao rezultat? Drugim riječima, kako će da glasi P_2?

Rješenje. V. *Tabelu 2.*

Prije nego što formulišemo glavni zadatak, da razjasnimo do kraja o ulazu i izlazu programa A (o ulazu i izlazu asemblera A). Ulazni podatak asemblera jeste P_1 ili rekli bismo `primjer.asm`. Asembler preuzima svoj ulazni podatak preko ulaznog uređaja ili sa spoljašnje memorije. Rezultat rada asemblera jeste P_2 ili rekli bismo `primjer.exe`. Neka ga saopštava preko izlaznog uređaja ili neka ga ostavlja na spoljašnju memoriju. Ili neka ga loaduje o istom trošku i neka još preda štafetu.

Zadatak. Sastaviti program A za osnovni računar, da zadovoljava gore nabrojane uslove.

Rješenje je teško i obimno, pa se zato izostavlja. Samo dajemo neka zapažanja. Jednoj običnoj naredbi u P_1 odgovara jedna naredba u P_2, što predstavlja jednu povoljnu okolnost. Rad asemblera odvija se u dvije faze: prvi prolaz i drugi prolaz, first pass and second pass. Tokom prvog pregledanja svog ulaznog podatka P_1, on obrazuje tzv. tabelu simbola. Svakoj simboličkoj adresi (A ili MJESTO ili slično) koja se pojavljuje u P_1 pridružuje se fizička adresa. U našem primjeru, tabela simbola glasi: A 206, B 207, C 208, to jest: A 0000 1100 1110, B 0000 1100 1111, C 0000 1101 0000. Tokom drugog pregledanja, on generiše objektni kod P_2. Obraća se tabeli simbola i tabeli opkodova.

Tabela 1

```
   ORG 200
   LDA A
   CIL
   ADD B
   INC
   STA C
   HLT
A, DEC 300
B, DEC 44
   END
```

Tabela 2

na adresi 200 tj.	na adresi 0000 1100 1000 neka piše 0010 0000 1100 1110
na adresi 201 tj.	na adresi 0000 1100 1001 neka piše 0111 0000 0100 0000
na adresi 202 tj.	na adresi 0000 1100 1010 neka piše 0001 0000 1100 1111
na adresi 203 tj.	na adresi 0000 1100 1011 neka piše 0111 0000 0010 0000
na adresi 204 tj.	na adresi 0000 1100 1100 neka piše 0011 0000 1101 0000
na adresi 205 tj.	na adresi 0000 1100 1101 neka piše 0111 0000 0000 0001
na adresi 206 tj.	na adresi 0000 1100 1110 neka piše 0000 0001 0010 1100
na adresi 207 tj.	na adresi 0000 1100 1111 neka piše 0000 0000 0010 1100

13. RAČUNARSKI SOFTVER

Računarski sistem sastoji se od svog hardverskog dijela i svog softverskog dijela. Hardver čine fizičke komponente i oprema koja je sa njima povezana. Softver čine programi koji su napisani za računar.

Jedan određeni računar ima svoj jedan jedini mašinski jezik.

Viši programski jezici služe da definišemo obradu koju računar treba da ostvari. Primjeri viših programskih jezika su paskal, C i C++.

Prevodilac ili programski prevodilac ili kompajler je jedan poseban sistemski program. On prevodi–pretvara tekst našeg programa napisanog (recimo) na paskalu u ekvivalentni program na mašinskom jeziku. Prevođenje je potrebno zato što računar može neposredno da izvršava samo program napisan na njegovom mašinskom jeziku.

Ulazni podatak za prevodilac P jeste tekst nekog programa P_1 (source code) na jednom jeziku, a njegov izlazni podatak tj. rezultat njegovog rada jeste tekst programa P_2 (object code) na nekom drugom jeziku, čiji je nivo niži. Tako da u definiciju programa P ulaze i ta dva nivoa. Primjer: prevodilac P sa paskala na mašinski jezik procesora Intel 8086 (operativni sistem DOS).

Postoji i jezik asemblera ili asemblerski jezik ili simbolički jezik. Taj jezik se po svom nivou nalazi između mašinskog jezika i višeg programskog jezika, a znatno je bliži mašinskom jeziku. On se od mašinskog jezika razlikuje uglavnom po sljedeće dvije stvari. Prvo: naredbe se ne zapisuju binarno nego pomoću skraćenica, recimo umjesto 0111 0000 0000 0001 piše se HLT. Drugo: adrese podataka ne zapisuju se binarno nego kao "imena promjenljivih", recimo umjesto 12 bita piše se A ili B ili slično.

Poseban sistemski program (zvani asembler) svodi bilo koji program napisan na jeziku asemblera na odgovarajući program napisan na mašinskom jeziku.

Sada ćemo da ilustrujemo ova tri nivoa pomoću jednog primjera. Neka treba da se sastavi program za sabiranje dva cijela broja. U nastavku je prikazano: a) kako taj program glasi ako je izražen na mašinskom jeziku osnovnog računara, b) na jeziku asemblera osnovnog računara i c) na višem programskom jeziku (na paskalu).

Oko a). Prva kolona pokazuje memorijsku adresu, tj. redni broj memorijske lokacije. Drugim riječima, prva kolona definiše položaj u memoriji svake naredbe i podatka, položaj zapisujemo binarno. Umjesto recimo adresa 110 trebalo bi da bude zapisano potpunije adresa 0000 0000 0110 i slično. Druga kolona pokazuje sadržaj te memorijske lokacije (sadržaj te memorijske riječi), sadržaj se izražava binarno.

Objašnjenja za a). Prva četiri reda odnose se na naredbe, a posljednja tri reda odnose se na podatke. Što se tiče prva četiri reda, imamo redom sljedeće za kod naredbe i smisao koda naredbe:

```
0010    LDA    AC ← M
0001    ADD    AC ← AC + M
0011    STA    M ← AC
0111 0000 0000 0001    HLT    zaustavljanje
```

Što se tiče posljednja tri reda, prvi sabirak nalazi se na adresi 100, a drugi sabirak nalazi se na adresi 101, dok će rezultat rada programa (zbir) biti na adresi 110. Vrijednost prvog sabirka jednaka je 83, a drugog -23, budući da se brojevi prikazuju u obliku potpunog komplementa (PK).

Program na mašinskom jeziku a) može da bude unesen u naznačeni dio memorije. Zatim on može da bude izvršen, startovanjem računara od adrese 0. Računarski hardver će izvršiti ove naredbe i time obaviti predviđeni posao. Ipak, kada se gleda tekst programa, teško se razumije šta će se dešavati tokom izvršavanja.

Oko b). Sada adrese više ne izražavamo u binarnom obliku kakav je recimo 0000 0000 0110 već ih sada izražavamo simbolički, kao recimo C. Skraćenica DEC znači dekadno.

Oko c). Ovaj program sastavljen je na programskom jeziku paskal, tako da ne traži poseban komentar.

Paskalov prevodilac pretvara treći tekst (c) u prvi tekst (a).

Da se primjer ne bi uveličavao, u programima su izostavljene ulazno–izlazne naredbe. Naime, u slučaju (a), učitavao bi se karakter–po–karakter (cifra–po–cifra) ulaznog podatka, tako da bi trebalo sastaviti i dio programa koji od učitanih karaktera stvara prvi sabirak, odnosno stvara drugi sabirak. Slično prilikom štampanja. Završen primjer.

U nastavku se uvodi opšta podjela za računarski softver. Računarski softver sastoji se iz dva dijela – sistemskog i aplikativnog. Sistemski softver sastoji se od niza izvršnih programa koji treba da učine da korišćenje računara bude lako i efikasno. Tako prevodilac za paskal pripada sistemskom softveru. I asembleri pripadaju sistemskom softveru. Za program koji pripada sistemskom softveru kaže se da je – sistemski program. Tako da prevodilac za paskal očito predstavlja jedan primjer sistemskog programa. Možemo reći da hardver zajedno sa sistemskim softverom čini komercijalni računar. Aplikativni softver sastoji se od programa koje je sam korisnik računara sastavio za neke svoje ciljeve (sastoji se od aplikativnih programa). Navedimo dva primjera a) i b) aplikativnih programa. a) Program za rješavanje sistema linearnih jednačina. b) Jedan program ili više programa za rješavanje nekih zadataka iz statistike. Kada se ima više programa onda se kaže da je to jedan paket programa.

Vratimo se sistemskom softveru i uvedimo jednu podjelu za sistemski softver. Dvije glavne komponente sistemskog softvera su: i) operativni sistemi i ii) prevodioci.

i) Operativni sistem jeste skup programa koji omogućavaju korišćenje hardvera. To je posrednik između računarskog hardvera i korisnika računara. To je dio softvera koji je najbliži hardveru. Može se reći da posredstvom operativnog sistema svi ostali programi pristupaju hardveru.

Operativni sistemi se dijele na jednokorisničke i višekorisničke. Kod jednokorisničkih sistema, kao što i sama riječ kaže, samo jedan programer može da upotrebljava računar u datom vremenu. Primjer: DOS, za personalne računare. Primjer višekorisničkog sistema: Unix, za velike računare. Postoje i operativni sistemi koji odgovaraju mreži personalnih računara. Operativni sistem Windows služi za personalne računare i za mreže personalnih računara.

Posljednjih godina postoje u svijetu dva operativna sistema čija se upotreba gledano po vremenskoj osi iz godine u godinu povećava. To su Windows i Unix. Potražnja za Windowsom raste po eksponencijalnom zakonu, a potražnja za Unixom raste po linearnom zakonu.

ii) Prevodioci (ili programi za prevođenje ili jezički procesori) vrše prevođenje na mašinski jezik programa koji je sastavljen na višem programskom jeziku (programa koji je sastavljen na korisnički–orijentisanom jeziku). Primjer prevodioca jeste prevodilac za paskal. Drugi primjer prevodioca jeste interpreter za bezik.

Postoje i druge komponente sistemskog softvera. Recimo, iii) programi koji služe za obradu teksta. Primjeri takvih programa (takvih paketa programa) su TEX i Word.

a)

adresa	naredba
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

b)

```
    LDA A     donesi u ak. broj sa adrese A
    ADD B     uvećaj ak. za broj sa adrse B
    STA C     broj iz ak. pošalji na adresu C
    HLT       zaustavi računar
A,  DEC 83
B,  DEC -23
C,  DEC 0
```

c)

```
program progra;
var a,b,c: integer;
begin
a:=83; b:=-23;
c:=a+b
end.
```

14. ORGANIZACIJA CENTRALNOG PROCESORA

U ovom naslovu i u iduća dva naslova govorimo uopšteno o načelima organizacije računara.

Računari čiji CPU ima mali broj registara koriste jedan akumulator za izvođenje aritmetičkih i sličnih radnji. Sa pojavom integrisanih kola veće složenosti (većeg stepena integracije) povećao se i broj registara CPU. Drugim riječima, napredak tehnologije omogućio je da jedno integrisano kolo može da sadrži veliki broj komponenti. Tako da umjesto jednog akumulatora ima više akumulatora, recimo 4 ili 16.

Bilo bi korisno da se rezultat neke operacije (recimo neke aritmetičke operacije) ne šalje u memoriju pa zatim poslije kratkog vremena čita iz memorije. Drugim riječima, pogodno je da neki među-rezultati ostanu u CPU. Isto tako, pogodno je da u CPU ostanu brojači, indeksi i povratne adrese.

Jasno je da je organizacija CPU koji ima znatan broj registara složenija od organizacije osnovnog računara, mašinski jezik takvog računara je složeniji od mašinskog jezika osnovnog računara. Takođe je i prevodilac za paskal takvog računara složeniji. A paskal je jedan te isti u jednom i drugom slučaju.

CPU se izrađuje iz djelova; djelovi CPU su registri, dekoderi, multiplekseri i slično. Tehnologija omogućava da se čitav CPU izradi u jednom komadu, kao jedna fizička cjelina, kao jedno integrisano kolo. Tada se kaže da je to jedan mikroprocesor. Mikroprocesoru se može pristupiti samo posredstvom njegovih spoljašnjih izvoda (njegovih nožica, njegovih pinova).

Mogu da budu razvijeni elementi paralelizma u radu CPU. Na primjer, dok ALU izvršava jednu naredbu (execute cycle) neka se iz memorije čita sljedeća naredba (fetch cycle).

U nastavku se govori o ALU i o izvođenju aritmetičkih operacija.

ALU jeste dio CPU. Uzmimo da u ALU ima nekoliko akumulatora. Tada je moguća naredba oblika $R3 \leftarrow R1 + R2$, gdje su R1, R2 i R3 – neka tri akumulatora.

Kako se vrši množenje dva broja? Prva mogućnost: pomoću jedne velike kombinacione mreže. Druga mogućnost: preko šiftovanja i sabiranja, uz upotrebu memorijskih komponenti (registara).

Pogledajmo drugu mogućnost. Da bi čitavo množenje moglo da se obavi unutar procesora, tj. bez obraćanja memoriji treba da imamo na raspolaganju četiri–pet registara. Tokom izvršavanja množenja, svaki od tih registara pamti jedan broj (jedan među–rezultat). Koji su to među–rezultati? Prvi množitelj, drugi množitelj, šiftovani oblik drugog množitelja i djelimični proizvod, na kraju će djelimični proizvod postati jednak proizvodu (rezultatu operacije množenja). Još eventualno i jedan brojač. O ovom postupku za množenje – v. tabelu 1 u nastavku.

Cijeli brojevi zapisuju se u obliku potpunog komplementa (PK), radili smo ranije. Kako se izvode četiri osnovne aritmetičke operacije? Sabiranje – v. šemu sabirača, radili smo ranije. Slično oduzimanje. Množenje – preko šiftovanja i sabiranja. V. tabelu 1 u nastavku koja prikazuje jedan primjer. Slično dijeljenje.

Realni brojevi (decimalni brojevi) zapisuju se u obliku pokretnog zareza (floating point), radili smo ranije. Da objasnimo postupak za sabiranje, poslužimo se primjerom, v. tabelu 2. Da objasnimo postupak za množenje, poslužimo se primjerom, v. tabelu 3.

Tabela 1. $a, \ldots, f \in \{0, 1\}$:

$abcdef \cdot 1101 = abcdef000 + abcdef00 + abcdef = \ldots$

Brojevi $abcdef000$ i $abcdef00$ dobijeni su šiftovanjem. Treba još izvršiti dva naznačena sabiranja.

Tabela 2. U pokretnom zarezu, broj x prikazuje se preko para brojeva m i e, po propisu $x = m \cdot 16^e$, $\frac{1}{16} \leq m < 1$, ako je $x > 0$. Mi želimo samo da izložimo princip sabiranja, pa nam je pogodno da 16 zamijenimo sa 10. Dakle, dopustimo da prikazivanje ima oblik $x = m \cdot 10^e$, $\frac{1}{10} \leq m < 1$.

Za mantisu se kaže da je normalizovana ako se njena vrijednost kreće u dozvoljenom okviru $\frac{1}{16} \leq m < 1$ (ili dopustili smo $\frac{1}{10} \leq m < 1$).

Naš primjer jeste očito $9.500.000 + 850.000 = 10.350.000$

$m_1 \cdot 10^e + m_2 \cdot 10^e = (m_1 + m_2) \cdot 10^e$

I sabirak		II sabirak		zbir		
m	e	m	e	m	e	
0,95	7	0,85	6			
		0,085	7			poravnavanje eksponenata
				1,035	7	sabiranje mantisa
				0,1035	8	normalizacija mantise zbira

Tabela 3. Naš primjer jeste očito $20 \cdot 3.000 = 60.000$

$(m_1 \cdot 10^{e_1}) \cdot (m_2 \cdot 10^{e_2}) = (m_1 \cdot m_2) \cdot 10^{e_1 + e_2}$

I množitelj		II množitelj		proizvod		
m	e	m	e	m	e	
0,2	2	0,3	4			
				0,06	6	množenje mantisa, sabiranje eksponenata
				0,6	5	normalizacija mantise proizvoda

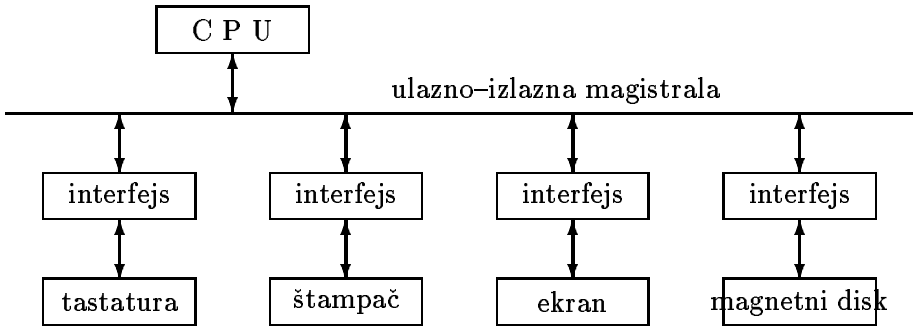
15. ORGANIZACIJA ULAZNO–IZLAZNE JEDINICE

Jedan računar povezan je sa nekoliko ulaznih i izlaznih uređaja (perifernih uređaja). S jedne strane, centralni procesor i unutrašnja memorija su elektronski uređaji. S druge strane, periferije su elektro–mehanički uređaji. Zato su centralni procesor i periferni uređaji povezani posredstvom izvjesnog hardvera koji omogućava skladno funkcionisanje, povezani su pomoću tzv. interfejsa.

Kada se podaci iz jedinice centralnog procesora CPU šalju jednom perifernom uređaju onda odgovarajući interfejs prihvata signale koji mu pristižu, prerađuje te signale u oblik koji odgovara uređaju i predaje signale samom uređaju. Onda periferni uređaj (recimo da je to štampač) izvrši odgovarajuću radnju (odštampa slovo). Interfejs štampača određuje koje slovo će biti odštampano, reguliše vremenske intervale štampanja, čini da se papir pomijera i slično. Kada se podaci prenose u suprotnom smjeru (od perifernog uređaja prema procesoru) onda imamo slične okolnosti.

Jedna magistrala služi da CPU komunicira sa više perifernih uređaja. Tipična veza koja služi za komuniciranje prikazana je na slici. Vidimo da su na slici prikazana četiri periferna uređaja: tastatura (ulazni uređaj), štampač (izlazni uređaj), ekran (izlazni uređaj) i magnetni disk (ulazno–izlazni uređaj) (jedinica spoljašnje memorije).

U najprostijem slučaju interfejs se svodi na tzv. ulazno–izlazna vrata. Primjer: ulazna vrata za tastaturu, to je praktično jedan prihvatni registar. Nasuprot tome, neki interfejsi predstavljaju složena integrisana kola koja manipulišu sa podacima. Za takav interfejs kaže se da predstavlja kontroler. Primjeri: kontroler ekrana i kontroler magnetnog diska.



O običnim perifernim uređajima, na popularnom nivou.

1. Matrični štampač sadrži integrisana kola. Procesor mu šalje jednu po jednu riječ. Kako jedna riječ pristigne tako on uradi jednu radnju (odštampa jedno slovo).

Laserski štampač sadrži puno više integrisanih kola i njegova saradnja sa procesorom je znatno unapređena. On predstavlja jedan podsistem ukupnog računarskog sistema ili se kaže jedan računar specijalne namjene. Procesor mu šalje podatke u velikim paketima. Jedan paket odgovara štampanju jedne stranice.

Naravno da i štampač bilo matrični bilo laserski šalje neka obavještenja procesoru. U tom cilju štampač gleda da izazove (hardverski) prekid.

Slične okolnosti važe i za ostale periferne uređaje: procesor i periferni uređaj razmjenjuju kako kontrolne informacije tako isto i obavještenja u užem smislu (podatke).

Dopunimo cijelu priču u hardverskoj ravni, u niskoj ravni. Neka procesor treba da pošalje perifernom uređaju jednu veliku količinu podataka (paket), a slične okolnosti važe i kada se vrši prenos podataka u suprotnom smjeru. Podaci odnosno paket nalaze se u unutrašnjoj memoriji (u RAM–u) a treba da budu prosleđeni perifernom uređaju. Postoji poseban čip **DMA** (direct memory access) kontroler koji omogućava da se prenos obavi za kratko vrijeme. Ako se koristi DMA onda put kojim idu podaci više nije memorija – procesor – periferija nego je sada memorija – DMA – periferija. Dakle, ako procesor to želi, procesor može da se obrati za uslugu DMA integrisanom kolu. Funkcionisanje DMA integrisanog kola organizovano je na jedan od sljedeća dva načina. Prvi način: dok DMA radi, procesor ne radi ništa (procesor je u fazi čekanja). Drugi način: dok DMA radi i procesor radi, po sistemu malo ja malo ti, kaže se da periferni uređaj DMA pozajmljuje cikluse.

Dopunimo cijelu priču u softverskoj ravni, u visokoj ravni. (Softverski) **drajver** perifernog uređaja je program koji je zadužen za taj periferni uređaj. Recimo, kada neki program želi nešto da odštampa onda se on obraća drajveru a onda se drajver obraća štampaču. Slično se štampač obraća svom drajveru a drajver eventualno programu.

2. Na tvrdom disku uočavamo dva važna područja: a) boot record ili boot sector i b) FAT (File Allocation Table) i Directory. Boot record zauzima jedan sektor (512 bajta) i to prvi po redu sektor na disku. FAT + Directory zauzima sljedeće po redu sektore na disku, a potroši približno 1% ukupnog prostora na disku. Sve ovo, u slučaju Intel + Dos.

Boot record ima značajnu ulogu prilikom pokretanja računara. Na početku pokretanja, taj sektor se loaduje na određeno mjesto u unutrašnjoj memoriji i njemu se predaje upravljanje. U rezultatu predaje upravljanja očito će početi da se izvršavaju naredbe iz tog dijela unutrašnje memorije. Njihovim izvršavanjem učine se dalji koraci u okviru pokretanja sistema. Najviše se loaduju novi programi i podaci (najviše se loaduju novi fajlovi). U boot recordu odnosno u tom dijelu unutrašnje memorije nalaze se i neki podaci o disku. Recimo labela diska. Isto kolika je veličina jednog sektora u bajtima, koliko sektora ima na jednoj traci, koliko ukupno traka sadrži čitavi disk. I slično.

FAT + Directory sadrži spisak naziva fajlova koji se čuvaju na disku i podatke o rasporedu fajlova. Directory sadrži za svaki fajl: naziv fajla, da li je fajl ili je poddirektorijum = subfolder, datum i vrijeme, veličina u bajtima, gdje počinje u smislu broj sektora i slično. Fajl počinje u jednom sektoru, zatim se nastavlja u nekom drugom sektoru koji nije obavezno susjedni, zatim zauzima još jedan sektor, itd. sve do svog "završnog" sektora. Tako da svakom fajlu odgovara jedan "lanac" sektora. Govoreći približno, FAT sadrži za svaki fajl podatke o lancu (podatke o lancu sektora) tog fajla.

3. Tastatura je ulazni uređaj i služi za prenošenje podataka na način riječ po riječ (karakter po karakter, bajt po bajt, jedna po jedna memorijska riječ). Obično je organizovano tako da podaci prvo dospijevaju u dio unutrašnje memorije koji je za to namijenjen (bafer veličine 20 bajta), a odatle ih program preuzima.

16. ORGANIZACIJA UNUTRAŠNJE I SPOLJAŠNJE MEMORIJE

Memorije sa kojima raspolaže računarski sistem dijele se u dvije vrste i) i ii), kako slijedi. Za jednu i drugu vrstu koriste se razni nazivi, kako slijedi:

i) Memorija ili unutrašnja memorija ili glavna memorija ili radna memorija ili operativna memorija ili RAM i

ii) Spoljašnja memorija ili pomoćna memorija ili periferna memorija ili velika memorija ili jedinica diska ili skladište.

Memorijski uređaj koji neposredno komunicira sa CPU naziva se unutrašnjom memorijom. Svi ostali memorijski uređaji odgovaraju spoljašnjim memorijama. U unutrašnjoj memoriji nalaze se samo oni programi i podaci koje procesor u datom trenutku koristi (koji su potrebni programu čije izvršavanje je u toku). A sve ostale informacije smještene su u spoljašnjoj memoriji, odakle mogu da budu po zahtjevu prebačene u unutrašnju memoriju.

O organizaciji unutrašnje memorije govorili smo ranije. Sada ćemo nešto više reći o organizaciji spoljašnje memorije.

Obični spoljašnji memorijski uređaji su magnetni disk, magnetna traka i magnetni doboš. Nešto više ćemo reći o magnetnom disku. Magnetnih diskova opet ima dvije vrste i) i ii), kako slijedi:

i) Tvrdi disk ili hard disk ili veliki disk ili stalni disk i

ii) Savitljivi disk ili floppy disk ili disketa.

Memorija sa magnetnim diskovima predstavlja sistem sa glavama i diskovima. Disk je metalna ili plastična ploča kružnog oblika čije su obje strane presvučene magnetnim materijalom. Ploča je podijeljena na koncentrične krugove (trake) koje sadrže memorijske ćelije (bitove). Trake su podijeljene na sektore jednakog kapaciteta. Jedan sektor obično sadrži 512 bajta. Na vertikalnoj osovini smješteno je više ploča. Ploče rotiraju konstantnom brzinom. Za zapisivanje i očitavanje podataka koriste se magnetne glave. Magnetna glava lebdi iznad ploče na vrlo malom rastojanju. Za zapisivanje, propusti

se struja kroz zapisne namotaje glave. Struja stvara magnetno polje koje prodire u magnetni materijal nanesen na ploču. Tako da se u elementu medijuma (materijala) stvori magnetisanje u jednom ili drugom smjeru (zavisno od smjera zapisne struje), a odgovara logičkoj nuli ili logičkoj jedinici. Slično se dešava kada glava vrši očitavanje stanja elementa (ćelije). Upisno–čitajuća glava može da se kreće linearno prema vertikalnoj osovini ili od vertikalne osovine, a široka je toliko da može upisivati ili čitati podatke sa tačno jedne trake. Obično je mehanička koncepcija sistema izvedena tako da svakoj ploči pripada jedan par magnetnih glava (iznad i ispod ploče). Neka treba da se izračuna vrijeme pristupa lokaciji, radi upisivanja ili čitanja podataka. Treba uračunati: vrijeme potrebno za linearno pozicioniranje glave, vrijeme potrebno da se rotacijom diska odgovarajući sektor postavi ispod glave i vrijeme potrebno za sami prenos podataka. Na slici je prikazan jedan magnetni disk (ploča). Kapacitet tvrdog diska personalnog računara kreće se od 100 MB do 4 GB. Adresa riječi upisane na disku određuje se obično pomoću: broja ploče, brojeva trake i sektora i broja riječi.

Potreba za distribucijom softvera dovela je do pojave savitljivih diskova (disketa). Disketa se može odvojiti od jedinice diska i na taj način se može prenositi softver. Ime su dobile po fizičkoj savitljivosti diskova. Postoji više vrsta disketa i disketnih jedinica za diskete. Dvije vrste disketnih jedinica su tipične: jedna za diskete prečnika 5,25 inča i druga za diskete prečnika 3,5 inča. Treba napomemuti da su diskete prečnika 3,5 inča zaštićene čvrstim omotom i da ustvari i nisu savitljive. Standardni kapacitet jednog sektora diskete je 512 bajta. Na primjer, uzmimo jednu disketu koja je dvostrana, ima 40 traka i ima 9 sektora po traci. Ukupni kapacitet te diskete iznosi očito:

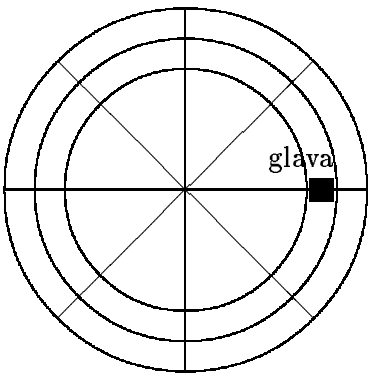
40 traka $\times$ 9 sektora $\times$ 512 bajta $\times$ 2 strane = 360 KB.

Dalje, kod nekih računara prisutna je i tzv. prikrivena memorija ili keš memorija; cache. To je jedna vrsta unutrašnje memorije. Karakteriše se velikom brzinom (i malim kapacitetom). Upotrebljava se za smještanje podataka koje tekući program često koristi i smještanje onih djelova teksta tekućeg programa koji se često koriste. Stvar je programiranja da se izaberu podaci koji će (umjesto u unutrašnjoj memoriji) biti smješteni u keš memoriji. Kod personalnih računara, keš memorija je smještena na jednom čipu.

Na primjer, memorija personalnog računara može da se sastoji od sljedeća tri dijela, s tim da kapaciteti pojedinih djelova iznose redom: keš 256 KB, unutrašnja 32 MB i spoljašnja 1 GB.

Keš memorija je brža tj. ima kraće vrijeme pristupa od unutrašnje memorije. Slično je i unutrašnja memorija brža od spoljašnje memorije. Na primjer, keš memorija može da bude jedan red veličine brža od unutrašnje memorije (to znači 10 puta brža), a unutrašnja memorija može da bude tri reda veličine brža od spoljašnje memorije (to znači 1000 puta brža).

Što je memorijski uređaj sporiji to je cijena po jedinici informacije manja.



17. PREGLED RAZVOJA MIKROPROCESORA

Na slici je prikazana uprošćeno arhitektura računarskog sistema zasnovanog na mikroprocesoru. Magistrala ili sistemska magistrala sastoji se iz tri dijela: adresna magistrala, magistrala podataka i upravljačka magistrala. CPU je mikroprocesor. U memoriji (u unutrašnjoj memoriji) M razlikujemo dva dijela: RAM i ROM, ROM sadrži tzv. jezgro operativnog sistema. Prikazane su tri obične

periferne jedinice (peripheral unit, PU): tastatura (ulazni uređaj), ekran (izlazni uređaj) i hard disk (spoljašnja memorija).

Na detaljnijoj slici bile bi nacrtane dvije magistrale: memorijska koja povezuje CPU sa M i sistemska koja povezuje CPU sa PU.

Za procesor koji je proizveden kao jedna fizička cjelina preciznije se kaže da predstavlja mikroprocesor. To je jedna pločica veličine kutije šibica koja ima spoljašnje izvode (pinove). Na jednom kristalu raspoređene su komponente zajedno sa linijama veze. Linije veze, kao uostalom i pinovi, propuštaju binarne signale. Komponenti, tj. tranzistora ima nekoliko hiljada. Glavni proizvođač mikroprocesora u svijetu jeste firma Intel.

Za računar koji je zasnovan na mikroprocesoru kaže se da predstavlja personalni računar. Prvi takvi računari pojavili su se krajem sedamdesetih godina. Firma IBM je među prvima počela da ih proizvodi.

Prvi mikroprocesor pojavio se 1971. godine. To je bio četvorobitni Intel 4004. Sadrži 2300 tranzistora. Šta znači "četvorobitni"? Podatak veličine 4 bita obrađuje se "odjednom", dužina registara unutar procesora iznosi 4 bita.

Jedan od prvih osmobitnih mikroprocesora jeste Intel 8080 iz 1974. godine. Navedimo glavne osobine ovog mikroprocesora. Širina magistrale podataka jednaka je 8 bita (1 bajt). Širina adresne magistrale jednaka je 16 bita, tako da je predviđeno da radi sa memorijom veličine 64 KB.

U razvoju 16–bitnih mikroprocesora mogu se razlikovati dvije faze. Predstavnik prve faze razvoja jeste Intel 8086 iz 1978. godine. Služi za PC XT. Ima 40 pinova. Predstavnik druge faze razvoja jeste Intel 80286 iz 1983. godine. Služi za PC AT. Ima 68 pinova. Uporedimo ovu vrstu mikroprocesora sa osmobitnim. Veći je stepen integracije, tj. na kristalu ima više tranzistora. U procesoru ima više registara. Proširen je set (skup) mašinskih naredbi.

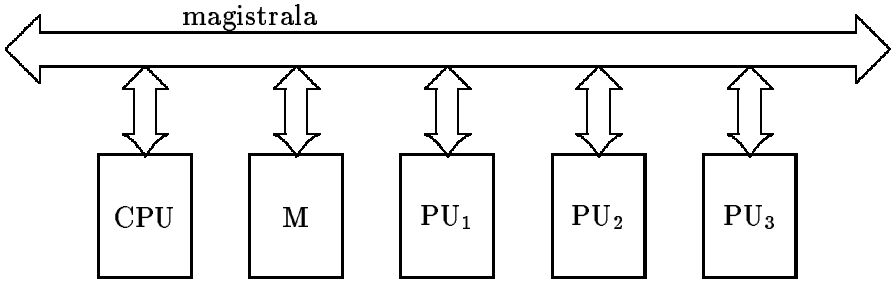
Magistrala podataka ima 16 bita. Adresna magistrala ima 20 do 24 bita. To očito odgovara memoriji veličine 1 do 16 MB.

Učestanost iznosi recimo 20 MHz.

Danas se koriste 32–bitni mikroprocesori. Primjeri ovakvih mikroprocesora su Intel 80386 iz 1985. godine i Intel 80486 iz 1989. godine.

Danas se koristi 32–bitni mikroprocesor Intel Pentium iz 1993. godine. Učestanost takta iznosi 60 MHz. Glavnu tehničku novinu predstavlja ugrađena keš–memorija. Pentium II 1997. godine 233 MHz. Pentium III 1999. godine 450 MHz.

Šta je to procesor ili mikroprocesor tipa CISC a šta je to tipa RISC? CISC je skraćenica za Complex instruction set computer – računar čiji je skup naredbi složen, a RISC je skraćenica za Reduced instruction set computer – računar čiji je skup naredbi sveden. Slabo koristi što je mikroprocesor složen, tj. što ima veliki broj mogućnosti ako oni koji sastavljaju programe na njegovom mašinskom jeziku ne koriste sve te mogućnosti. Razvojna linija ili razvojna koncepcija RISC govori da treba napraviti mikroprocesor koji ima manji broj naredbi, naredbe neka budu jednostavne, neka imaju fiksiranu dužinu, svesti broj mogućih načina adresiranja i slično. A zauzvrat (budući da broj tranzistora nije smanjen) može se povećati broj registara u mikroprocesoru, može se bolje organizovati pipeline (dok izvršavanje naredbe još nije završeno počinje obrada iduće naredbe), može se bolje organizovati upotreba keš–memorije i slično. Primjer za CISC je Intel Pentium. Primjer za RISC je DEC Alpha.



18. Primjer mikroprocesora: i8080

U ovom naslovu govori se o 8–bitnom mikroprocesoru Intel 8080: skup registara, skup naredbi i smisao pinova.

Pojavio se na tržištu 1974. godine. Sadrži 6000 tranzistora tipa NMOS. Ima učestanost takta 2 MHz. Na njemu zasnovani računari koristili su čuveni operativni sistem 70–tih godina CP/M – Control Program for Microcomputers. Glavni konkurenti bili su Motorola 6800 iz 1974. godine i Zilog Z–80 iz 1976. godine. Koristio se kod tzv. kućnih računara (home computers), naročito u periodu od 1980. do 1985. godine.

Pogledajmo arhitekturu mikroprocesora Intel 8080: ima 10 registara, ima 74 vrste naredbi i ima 40 pinova.

Pogledajmo registre. Akumulator A 8 bita. Registri opšte namjene B, C, D, E, H i L po 8 bita, gdje se parovi BC, DE i HL mogu koristiti kao registri od po 16 bita. Brojač naredbi PC 16 bita. Pokazivač steka SP 16 bita. I registar flagova F 8 bita.

Ima 5 flagova (uslovnih bita) i oni su okupljeni u registru F po sljedećem rasporedu:

7–0	SF	ZF	—	AF	—	PF	—	CF

SF Sign Flag pokazuje kakav je predznak akumulatora. ZF Zero Flag pokazuje da li akumulator sadrži nulu. AF Auxiliary Carry Flag pokazuje da li je došlo do prenosa na mjestu koje je za četiri niže od mjesta najveće težine. PF Parity Flag pokazuje da li akumulator sadrži paran broj bita 1. CF Carry Flag pokazuje da li je došlo do prenosa na mjestu najveće težine.

19. 8080 – NAREDBE

U tabeli naredbi koriste se sljedeće tri skraćenice:
Addr znači adresa (16 bita),
D8 znači neposredni argument (8 bita),
reg znači A, B, C, D, E, H, L ili (HL),
gdje (x) znači sadržaj od x.

Početak tabele Naredba i njen smisao: 8 naredbi aritmetičke `ADD reg` $A \leftarrow A + reg$ `ADC reg` $A \leftarrow A + reg + CF$ `ADI D8` $A \leftarrow A + D8$ `ACI D8` $A \leftarrow A + D8 + CF$ slično `SUB reg` $A \leftarrow A - reg$ `SBB reg` $A \leftarrow A - reg - CF$ `SUI D8` $A \leftarrow A - D8$ `SBI D8` $A \leftarrow A - D8 - CF$

8 logičke `ANA reg` $A \leftarrow A \,\&\, reg$ `ANI D8` $A \leftarrow A \,\&\, D8$ `ORA reg` $A \leftarrow A \vee reg$ `ORI D8` $A \leftarrow A \vee D8$ `XOR reg` $A \leftarrow A \oplus reg$ `XRI D8` $A \leftarrow A \oplus D8$ `CMP reg` compare, flagovi se namjeste zavisno od reg `CPI D8` compare immediate, flagovi se namjeste zavisno od D8

4 šift `RAL` ulijevo A `RLC` ulijevo $A + CF$ `RAR` udesno A `RRC` udesno $CF + A$

2 inkrement i dekrement `INR reg` $reg \leftarrow reg + 1$ `DCR reg` $reg \leftarrow reg - 1$

2 dodjela vrijednosti `MOV reg1, reg2` (move) $reg1 \leftarrow reg2$ `MVI reg, D8` (move immediate) $reg \leftarrow D8$

27 predaja upravljanja `JMP Addr` skok `CALL Addr` poziv `RET` povratak, ove tri naredbe su bezuslovne. Postoji osam trojki uslovnih naredbi za skok, poziv i povratak. U osam slučajeva uslov se definiše kao: ako je $PF = 0$, ako je $PF = 1$, ako je $CF = 0$, ako je $CF = 1$, ako je $ZF = 0$, ako je $ZF = 1$, ako je $SF = 0$, ako je $SF = 1$. Skok se razlikuje od poziva po tome što se u slučaju poziva još spasi povratna adresa (upiše se u stek)

2 upravljačke `HLT` halt (stop) `NOP` no operation (prazna naredba)

2 ulazna i izlazna `IN n` $A \leftarrow$ port broj n `OUT n` port broj n $\leftarrow A$ gdje je $0 \leq n \leq 255$

1 restart `RST n` bezuslovni poziv potprograma koji počinje na (apsolutnoj) adresi $8 \cdot n$, pri čemu je $0 \leq n \leq 7$

1 aritmetička `DAD x` $HL \leftarrow HL + x$ gdje $x \in \{BC, DE, HL, SP\}$

3 dodjela vrijednosti `LXI x, D16` $x \leftarrow D16$ gdje $x \in \{BC, DE, HL, SP\}$ `LHLD Addr` $HL \leftarrow$ (Addr) `SHLD Addr` (Addr) $\leftarrow HL$

4 dodjela vrijednosti `LDAX x` $A \leftarrow (x)$ gdje $x \in \{BC, DE\}$ `STAX x` $(x) \leftarrow A$ `LDA Addr` $A \leftarrow (Addr)$ `STA Addr` $(Addr) \leftarrow A$

5 posebne naredbe

`XCHG` $DE \longleftrightarrow HL$ exchange

`DAA` A kao dvije dekadne cifre decimal adjust accumulator

`CMA` $A \leftarrow (FF)_{16} - A$ complement accumulator

`STC` $CF \leftarrow 1$ set carry

`CMC` $CF \leftarrow 1 - CF$ complement carry

`PCHL` $PC \leftarrow HL$ HL to program counter

2 oko prekida `DI` disable interrupt `EI` enable interrupt

4 rad sa stekom `PUSH x` stavi x na stek gdje $x \in \{BC, DE, HL, AF\}$ `POP x` uzmi x iz steka (skini x sa vrha steka) `XTHL` $HL \longleftrightarrow (SP)$ `SPHL` $SP \longleftrightarrow HL$

2 inkrement i dekrement `INX x` $x \leftarrow x + 1$ gdje $x \in \{BC, DE, HL, SP\}$ `DCX x` $x \leftarrow x - 1$

Kraj tabele

Potrebna su neka objašnjenja za tabelu.

Stek? Ako na stolu imamo nekoliko knjiga naslaganih jedna na drugu onda možemo da stavimo još jednu knjigu gore (push) ili možemo da uzmemo knjigu koja je u datom trenutku gornja (pop). Rekli smo da se u steku drže povratne adrese potprograma. Dakle, stek služi da reguliše call i return za potprograme (odlazak u potprogram i povratak iz potprograma). SP je adresa tekućeg gornjeg člana steka. Naredbe PUSH i POP automatski podešavaju vrijednost SP ($SP \leftarrow SP - 2$, odnosno $SP \leftarrow SP + 2$). Isto važi i za naredbe CALL i RET.

Neposredna, obična i indirektna adresa? Vidimo da u mašinskom jeziku ovog procesora postoje tri načina da se definiše argument recimo aritmetičke operacije (primjera radi drugi sabirak prilikom sabiranja). Neposredni način: u samoj naredbi piše bukvalno argument, immediate. Za indirektni način koristi se registar HL, pa se za HL ponekad kaže da je indeks–registar.

Pogledajmo kroz primjere naredbi za dodjelu vrijednosti (c – contents – sadržaj):

neposredni argument (immediate operand): MVI A, broj znači $A \leftarrow$ broj
obični ili direktni argument (direct operand): LDA broj znači $A \leftarrow$ c(broj)
indirektno definisan argument (indirect operand): MOV A, (HL) znači $A \leftarrow$ c(HL) (prethodno smo odgovarajući broj upisali u HL)

Uporediti sa naredbama za bezuslovni skok:

obični bezuslovni skok: JMP broj znači program counter $\leftarrow$ broj
indirektni bezuslovni skok: PCHL znači program counter $\leftarrow$ HL (prethodno smo odgovarajući broj upisali u HL)

Sistem prekida? Ima nekoliko perifernih uređaja i svaki od njih može da ispostavi zahtjev. Ako je zahtjev za prekidom usvojen od strane procesora onda periferni uređaj koji je tražio šalje (preko magistrale podataka) jedno obavještenje procesoru. Da bi procesor znao ko je tražio, odnosno da bi znao šta sad treba da uradi. To obavještenje je jedna naredba. Sada će procesor izvršiti tu naredbu. Po pravilu, to je naredba RST n.

Dakle, možemo reći da ima više vrsta prekida (osam). Svaka vrsta ima svoju rutinu. Povratna adresa je spašena. Rutina spašava dva registra A i F, po pravilu.

20. 8080 – PINOVI

Prelazimo na spoljašnje izvode (nožice, pinove) mikroprocesora Intel 8080. Na slici je prikazan spoljašnji izgled mikroprocesora. Dakle, ima 40 pinova i to:

A_0–A_{15} D_0–D_7 −5V +5V +12V 0V(GND) CLK1 CLK2

SYNC DBIN READY WAIT $\overline{\text{WR}}$ HOLD HLDA INTE INT RESET

Koje funkcije imaju izvodi kućišta?

Izlazi sa tri stanja A_0 do A_{15} čine adresnu magistralu. Oni omogućavaju adresiranje radne memorije do maksimalnog kapaciteta 64 KB ili adresiranje ulaza–izlaza (na A_0–A_7) sve do maksimalnog kapaciteta od 256 ulaza i 256 izlaza.

Ulazi–izlazi sa tri stanja D_0 do D_7 čine magistralu podataka. Oni omogućavaju prenošenje naredbi i podataka između mikroprocesora, radne memorije i ulazno–izlaznih uređaja.

Izvor napajanja treba da daje napajanja od -5V, $+5$V i $+12$V, a prisutno je i uzemljenje.

Generator taktova i8224 šalje mikroprocesoru preko linija CLK1 i CLK2 dva taktna signala. Dva signala imaju jednaku periodu, ali se razlikuju po trenutku kada počinje prelazak sa niske na visoku vrijednost i po dužini trajanja visoke vrijednosti.

Izlaz SYNC predstavlja sinhronizacioni signal. On ukazuje na početak svakog mašinskog ciklusa.

Izlaz DBIN – data bus in –podatak na magistrali podataka. Objavljuje spoljašnjim uređajima da je magistrala podataka u ulaznom načinu rada. Drugim riječima, obavještava ih da je u toku učitavanje podataka iz radne memorije ili iz ulazno–izlaznih uređaja.

Ulaz READY – spreman. Ovaj signal upotrebljava se za sinhronizaciju mikroprocesora sa memorijom ili sa ulazno–izlaznim uređajima. Služi da se 8080 zadrži dok je čitanje ili upisivanje u toku.

Izlaz WAIT – čekanje. Ovaj signal pokazuje da je mikroprocesor u stanju čekanja. On čeka signal READY.

Izlaz $\overline{\text{WR}}$ – write. Obavještava da je magistrala podataka u izlaznom načinu rada.

Ulaz HOLD – zadržavanje. Signalom HOLD traži se od mikroprocesora da stupi u stanje zadržavanja. Stanje zadržavanja omogućava drugim djelovima računara da preuzmu upravljanje adresnom magistralom i magistralom podataka od trenutka kada mikroprocesor završi mašinski ciklus koji je u toku.

Ako mikroprocesor stupi u stanje zadržavanja onda su adresna magistrala i magistrala podataka u trećem stanju, tj. u stanju visoke impedanse. Drugim riječima, one su odspojene od mikroprocesora.

Sada drugi čipovi (drugi djelovi računara) koriste te dvije magistrale po svome.

Koji su to drugi čipovi? To je obično DMA kontroler. DMA – direct memory access – direktni pristup memoriji. Kada treba iz memorije na periferiju ili obrnuto da se prenese veliki blok podataka. Ili to može da bude eventualno prisutni drugi mikroprocesor.

(Procesor će izaći iz stanja zadržavanja kada nastupi spoljašnji prekid.)

Izlaz HLDA – hold acknowledge – potvrda zadržavanja. Pojavljuje se kao odziv na signal HOLD i pokazuje da su dvije magistrale odspojene od mikroprocesora.

Izlaz INTE – interrupt enable – prekid dopušten. Pokazuje kakav je sadržaj unutrašnjeg flipflopa kome može da bude dodijeljena vrijednost 0 ili 1 mašinskim naredbama. Kada je taj flipflop u stanju nula onda je onemogućeno ostvarivanje prekidanja programa (usvajanje zahtjeva za prekid).

Ulaz INT – interrupt request – zahtjev za prekid. Po ovoj liniji dolaze zahtjevi.

Ulaz RESET – dovođenje na nulu. Ako je signal RESET aktivan onda se sadržaj programskog brojača svodi na nulu, takođe se i neki flipflopovi dovode na nulu, dok se recimo sadržaj akumulatora ne dira.

Pregled pinova je završen. Tokom vremena SYNC $= 1$ procesor saopštava (preko linija D_0 do D_7) koja vrsta njegovog mašinskog ciklusa počinje da se izvršava u datom trenutku.

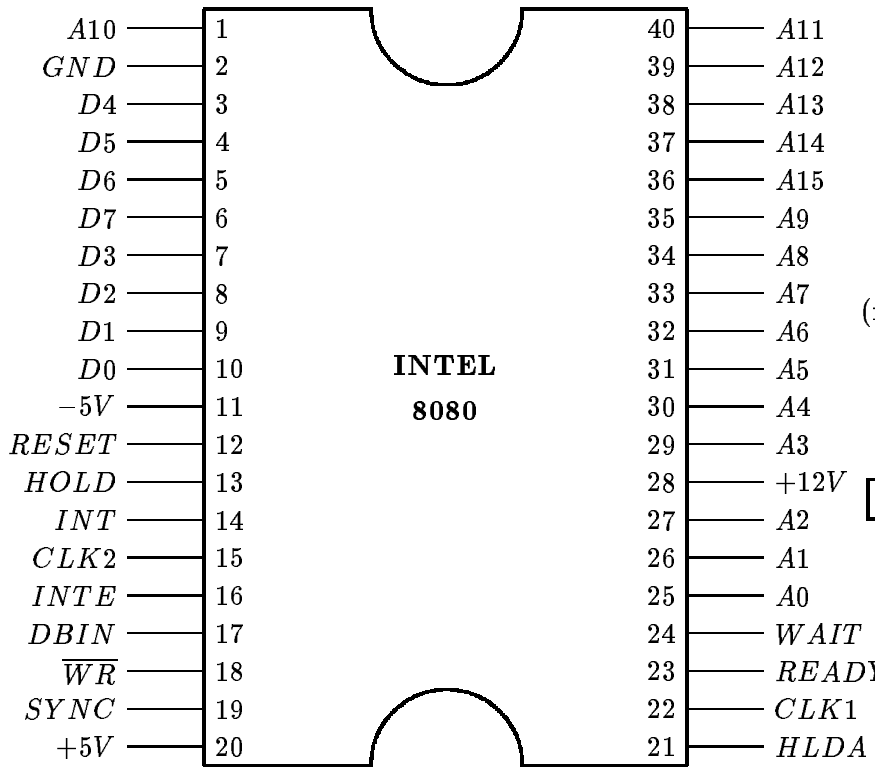
Ima 9 vrsta mašinskih ciklusa (vremenskih ciklusa). Samo prvi od njih se obavezno ostvari (fetch). Izvršavanje jedne naredbe obuhvata od 1 do 5 ciklusa, zavisno od vrste naredbe. S druge strane, za izvršavanje ciklusa treba 3 ili 4 ili 5 taktova. Ukupno, izvršavanje naredbe traje od najmanje 4 do najviše 17 taktova, što zavisi od vrste naredbe. Od ovoga postoje tri izuzetka kako slijedi. Postoje tri slučaja kada izvršavanje naredbe traje po volji dugo, odnosno zavisi od spoljašnjih događaja. To su slučajevi kada procesor uđe u stanje čekanja (WAIT) ili u stanje zadržavanja (HOLD) ili u stanje zaustavljanja (HALT). Uzima se da treba u prosjeku 6 taktova da se jedna naredba izvrši.

Ciklus je dobio naziv po najvažnijoj stvari koja se tokom njega dešava. U nastavku je nabrojano svih 9 vrsta mašinskih ciklusa. FETCH donošenje naredbe. MEMORY READ čitanje iz memorije.

MEMORY WRITE upisivanje u memoriju. STACK READ čitanje iz steka. STACK WRITE zapis u stek. INPUT ulaz iz perifernog uređaja. OUTPUT izlaz na periferni uređaj. INTERRUPT prekidni ciklus. HALT zaustavljanje.

Znamo da je veličina naredbe jednaka 1 ili 2 ili 3 bajta.

Spoljašnji izgled (pinovi):



Procesor Intel 8080 ima registre:

A accum.	F flags
B	C
D	E
H	L
SP stack pointer	
PC program counter	

(mogu parovi BC, DE, HL i AF)

Registar F sadrži flipflopve 7–0:

SF	ZF	—	AF	—	PF	—	CF

SF sign flag
ZF zero flag
AF aux. carry flag
PF parity flag
CF carry flag

21. Primjer mikroprocesora: i8086

Intel je 1978. godine ponudio tržištu prvi mikroprocesor od 16 bita: Intel 8086. 1979. godine pojavio se Intel 8088 (vrlo sličan sa 8086), a 1980. godine pojavila su se dva koprocesora: 8087 (matematički) i 8089 (za ulaz–izlaz). Prvi proizvođač koji je razvijao softver i hardver za te čipove bio je sami Intel. Potrebni softver obuhvatao je asemblere, prevodioce za više programske jezike i pomoćne programe. I druge firme su proizvodile računare zasnovane na mikroprocesoru 8086 ili 8088. Kao operativni sistem koristio se CP/M.

Firma IBM je ušla u polje personalnih računara 1981. godine. IBM je izabrao 8088 da bude CPU. Računar sa 48 KB RAM memorije i sa jedinicom za 160 KB diskete koštao je oko 2.000 dolara. IBM je sklopio ugovor sa firmom Microsoft, da Microsoft napravi operativni sistem (DOS) za te računare. Tako da se prva verzija DOS–a pojavila 1981. godine. Ugovor je predviđao da Microsoft pravi i ostali softver.

22. INTEL 8086 – ARHITEKTURA

U ovom naslovu govori se o arhitekturi mikroprocesora Intel 8086: registri, flagovi, adresiranje i pinovi.

Na slici su prikazani glavni registri, tj. registri koji su vidljivi programeru. Vidi se da ima 14 glavnih registara i da je svaki veličine 16 bita. Mikroprocesor se sastoji iz dva dijela: EU – Execution unit – izvršna jedinica i BU – Bus interface unit – jedinica za vezu sa magistralom. Opišimo barem ukratko smisao pojedinih registara.

Prva četiri registra označavaju se kao AX, BX, CX i DX. Kod svakog od njih, može posebno da se pristupi njegovoj gornjoj ili donjoj polovini; H High gornji, L Low donji. AX je akumulator, accumulator register. Ima veliki broj funkcija: aritmetičko–logičke radnje, ulazno–izlazne radnje i slično. BX je bazni registar, base register. Sličan je akumulatoru. Na primjer, pomaže prilikom indirektnog adresiranja, kao što ćemo vidjeti kada budemo govorili o naredbama. CX je brojač, counter register. Koristi se za petlje, za rad sa nizovima slova i za drugo. DX je registar za podatke, data register.

Predviđeno je da procesor radi sa memorijom veličine 1 MB $= 2^{20}$ bajta. Znamo da se procesor obraća memoriji radi čitanja naredbe i radi čitanja ili upisivanja podatka. Da bi se pristupilo adresi u memoriji treba očito saopštiti adresu te lokacije. Adresa ili svejedno fizička adresa a obrazuje se kombinovanjem adrese segmenta s i dodatka d po formuli $a = 16s + d$. Veličine s i d čuvaju se u dva registra. Recimo, ako je $s = 0 \ldots 011$ (16 bita) i $d = 0 \ldots 01$ (16 bita) onda će adresa biti $a = 0 \ldots 0110001$ (20 bita). Ako dvije veličine s i d obrazuju adresu na opisani način onda je uobičajeno da se o toj adresi govori kao o adresi s:d.

Sljedeća četiri registra SP, BP, SI i DI čuvaju dodatni dio adrese. SP je pokazivač steka, stack pointer. Jedino se koristi da čuva dodatak (offset) adrese gornjeg člana steka. Znamo da se stek koristi da čuva povratne adrese potprograma. Može da se koristi i za pamćenje nekih među–rezultata. BP je pokazivač osnove, base pointer. SI je indeks izvora, source index. Prilikom operacije sa nizom slova, tu se čuva dodatak adrese izvora. DI je indeks odredišta, destination index. Prilikom operacije sa nizom slova, tu se čuva dodatak adrese odredišta. Mi kažemo destination $\leftarrow$ source ili svejedno odredište $\leftarrow$ izvor.

Za dosad nabrojanih osam registara kaže se da su registri opšte namjene. Svaki od njih može da se upotrebljava bilo kao izvor bilo kao odredište u aritmetičko–logičkim naredbama. Ne može se isto tvrditi za bilo koji od šest registara koji slijede. Za svaki od tih šest registara kaže se da predstavlja jedan registar posebne namjene.

Na redu su četiri tzv. segmentna registra CS, DS, SS i ES. CS je segment za kod (segment za program), code segment. Sadrži segmentni dio adrese naredbe. Imamo u vidu naredbu mašinskog programa koja je došla na red za izvršavanje, tj. naredbu koju iz memorije treba donijeti u procesor. DS je segment za podatke, data segment. Sadrži segmentni dio adrese podatka koji se šalje u memoriju ili se iz memorije donosi u procesor. SS je segment za stek, stack segment. Sadrži segmentni dio adrese gornjeg člana steka. ES je ekstra segment, extra segment. Takođe sadrži segmentni dio adrese nekog podatka.

Ako dva registra sarađuju onda se iz ta dva dijela obrazuje jedna 20–bitna adresa. Ako SS i SP sarađuju onda se dobije adresa gornjeg člana steka. U programu je uobičajeno da se ta adresa zapisuje kao SS:SP. Na isti način, ako DS ili ES sarađuje sa BP ili SI ili DI onda može da bude izračunata adresa podatka.

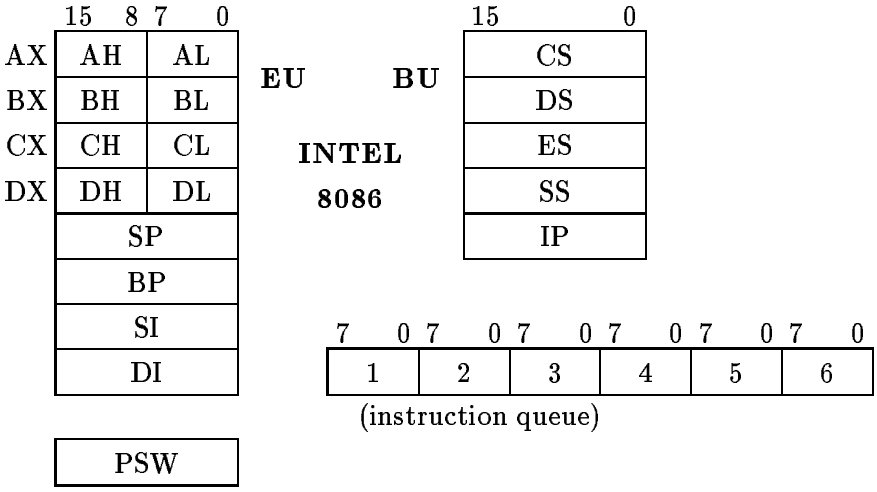
IP je pokazivač naredbe, instruction pointer. Čuva dodatni dio adrse naredbe. Ako CS i IP sarađuju onda oni određuju jednu adresu, određuju adresu jedne naredbe. Upravo, adresa naredbe je CS:IP, to je adresa prvog bajta naredbe. s:d = CS:IP.

PSW je registar flagova ili statusna riječ procesora, program status word. Pojedini bit registra sadrži vrijednost jednog flaga. Neke od tih vrijednosti mogu da utiču na redosljed izvršavanja naredbi mašinskog programa.

Red naredbi, instruction queue? Dok je magistrala besposlena, unaprijed se donose naredbe (fetch)

i stavljaju se u tzv. red naredbi koji se sastoji od šest jedno–bajtnih registara. Oni su na slici prikazani pomoću brojeva od 1 do 6. Red naredbi služi da se ubrza ukupni rad procesora. Dobro pomaže kada se izvršavaju pravolinijski djelovi programa (gdje nema skokova).

Pored 14 glavnih registara, u procesoru ima još (pomoćnih) registara. Šest jedno–bajtnih registara spadaju u pomoćne registre.



Pinovi? Ima 40 pinova, kao i i8080, ali je opšti raspored složeniji nego kod i8080, jer veliki broj pinova ima više od jedne funkcije. Samo dvije stvari o pinovima a) i b).

a) Adresna magistrala je 20–bitna, pinovi čije su oznake od AD0 do AD15 i od A16 do A19. To odgovara memoriji čija je veličina 2^{20} B = 1 MB. Magistrala podataka je 16–bitna a koristi pinove od AD0 do AD15. Tako da su adresna magistrala i magistrala podataka samo logički razdvojene, a ne i fizički. Zato, dok je u toku prenošenje nekog podatka ne može da se vrši prenošenje adrese, kao i obrnuto naravno.

S jedne strane, procesorovi registri imaju po 16 bita. S druge strane, za definisanje adrese treba 20 bita. Očito da sadržaj jednog registra nije dovoljan da definiše adresu. Adresa se definiše kombinovanjem dva registra, kao što smo već vidjeli detaljno.

Linije od AD0 do AD15 koriste se na multipleksni način za adresnu magistralu i za magistralu podataka. Upravo, ako je ALE = 1 onda te linije sadrže adresu, a ako je ALE = 0 onda one sadrže podatak. ALE je oznaka za jedan izlazni pin. Slično se i neke druge linije koriste na multipleksni način.

b) Tri pina NMI, INTR i INTA odnose se na prekide. Po liniji NMI not–maskable interrupt procesor prima obavezujući zahtjev za prekid, zahtjev kome će svakako biti udovoljeno, zahtjev koji ne može da bude maskiran. Po liniji INTR interrupt request prima (obični) zahtjev za prekid. INTA interrupt accepted je izlazna linija. Ako je usvojio zahtjev onda po toj liniji saopštava upravo da je usvojio zahtjev.

Kao odgovor na njegov signal INTA, procesoru se preko magistrale podataka dostavlja odgovarajući broj prekida n, gdje je $0 \leq n \leq 255$.

Po čemu se 8088 razlikuje? 8088 iako je kasnije proizveden je slabiji od 8086. Ima istu izvršnu jedinicu EU i ima nešto slabiju jedinicu za vezu sa magistralom. Upravo, 8088 je predviđen da radi sa magistralom podataka čija je širina svega 8 bita. Još, njegov red naredbi sastoji se od svega četiri jedno–bajtna registra.

23. 8086 – FLAGOVI

Zanimljivo je pogledati smisao pojedinih flagova. Iako registar PSW ima 16 flipflopova, samo 9 flipflopova se koristi, tj. ima samo 9 flagova, kako slijedi. V. sliku registra PSW.

CF Carry flag zastavica za prenos. Ako je CF = 1 onda to znači da se nešto pamti u slučaju sabiranja, odnosno da se nešto pozajmilo od bita najveće težine u slučaju oduzimanja. Koristi se prilikom sabiranja ili oduzimanja brojeva čija je veličina nekoliko bajta. Koriste ga i naredbe za kružno pomijeranje.

PF Parity flag flag parnosti. Ako je PF = 1 onda to znači da je rezultat parne parnosti, tj. da rezultat sadrži paran broj bitova 1. Ovaj indikator može da se koristi za otkrivanje greške prilikom prenošenja podataka.

AF Auxiliary carry flag pomoćni flag za prenos. Ako je AF = 1 onda to znači da se nešto pamtilo, odnosno da se nešto pozajmilo od prethodnog nibble na nibble najveće težine, nibble = 4 bita. Ovaj flag koriste naredbe za tzv. dekadno podešavanje (vezano za BCD kod).

ZF Zero flag nula–flag. Ako je ZF = 1 onda to znači da je rezultat operacije jednak nuli.

SF Sign flag flag za predznak. Preslikava bit najveće težine rezultata. Budući da se negativni brojevi prikazuju u komplementu do dva, to ovaj bit predstavlja predznak: 1 ako je rezultat negativan, a 0 ako nije negativan.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSW	—	—	—	—	OF	DF	IF	TF	SF	ZF	—	AF	—	PF	—	CF

TF Trap flag zamka–flag. Ako je TF = 1 onda će doći do prekida, kada procesor završi tekuću naredbu. Dolazi do skoka na jednu tačno određenu rutinu za obradu prekida. Kaže se da procesor prelazi u stanje korak po korak. On će izvršavati naredbu po naredbu. Da bi se otklonile greške u programu (debug), odnosno da bi se vidio trag (trace).

Kasnije ćemo vidjeti da ukupno postoje tri vrste prekida. Hardverski – generiše ih spoljašni uređaj, preko pinova NMI i INTR. Softverski – generiše ih program koji se izvršava, pomoću naredbe INT. I unutrašnji. Primjer unutrašnjeg prekida je prekid do koga dolazi kada je TF = 1.

IF Interrupt enable flag flag za dozvolu prekida. Ima ulogu maske kod maskirajućih (hardverskih) prekida, tj. kod prekida čiji zahtjev dolazi na liniju INTR. Ako preko upravljačke linije INTR nastupi zahtjev za prekid i ako je IF = 1 onda procesor pristupa obradi prekida, a ako je pak IF = 0 onda se zahtjev ne usvaja. Ovaj flag nema uticaja na ostale vrste prekida. Postoje posebne mašinske naredbe koje se koriste za dodjeljivanje vrijednosti 1 odnosno 0 ovom flagu.

DF Direction flag flag za smjer. Pomaže prilikom rada sa nizom slova. Utiče na sadržaj dva registra SI source index i DI destination index. Ako je DF = 1 onda izvršavanje naredbe za rad sa nizom slova izaziva da se sadržaj oba ta registra umanji za po jedan; kada se niz slova obrađuje u smjeru od viših memorijskih adresa prema nižim memorijskim adresama, auto–dekrement. Ako je DF = 0 onda auto–inkrement, povećava se za po jedan.

OF Overflow flag flag prekoračenja. Ako je OF = 1 onda to znači da je došlo do prekoračenja prilikom izvršavanja aritmetičke operacije (rezultat je prevelik), a u suprotnom je sve u redu. Dakle, ovaj flag služi kao indikator greške prilikom izvršavanja aritmetičke operacije. U slučaju OF = 1 može se pozvati posebna rutina za obradu prekida (naredba INTO, softverski prekid).

24. JEDNOSTAVNI PRIMJERI PROGRAMA NA JEZIKU ASEMBLERA. primjera

Svaki DOS ili Windows računar sadrži program debug.exe. Taj program služi za četiri primjera koji su sada na redu.

1. Aritmetički program. Sastaviti program za računanje četiri broja y=a·b·2+1, y=y−2, y=y−2 i y=y−16. Neka se na kraju odštampaju ta četiri broja.

Rješenje. Umjesto 100 na ekranu piše 0F6E: 0100. Umjesto 102 na ekranu piše 0F6E:0102. Itd.

```
    A 100 loaduj na 100,101,... u tekućem s.
100 JMP 108       goto 108
102 DB E          a=14 (1 bajt)
103 DB 3          b=3
104 DB 0          mjesto za y1
105 DB 0          y2
106 DB 0          y3
107 DB 0          y4
108 MOV AL,[102]  y=a=14
10B MOV CL,[103]
10F MUL CL        AX←AL·CL y=a·b=42
111 SHL AX,1      shift left AX y=2y=84
113 INC AX        increment y=y+1=85
114 MOV [104],AL
117 SUB AX,2      y=y−2=83
11A MOV [105],AL
11D SUB AX,2      y=y−2=81
120 MOV [106],AL
123 SUB AX,10     y=y−16=65
126 MOV [107],AL
129 MOV AH,2      INT 21 & AH=2: štam-
12B MOV DL,[104]  panje slova (bajta) DL
12F INT 21        poziva se rutina prekida
131 MOV DL,[105]
135 INT 21        štampa se drugi broj
137 MOV DL,[106]
13B INT 21        štampa se treći broj
13D MOV DL,[107]
141 INT 21
143 INT 20        rutina koja okončava rad
145 ␣             prazan red (izađi iz l.)
    G             ajde ili run executable
```

Na kraju će se odštampati USQA. Znamo da slovo U ima ASCII kod 85 S 83 Q 81 A 65. Još će se odštampati Program terminated normally. Ovo je bio program primjera.

25. primjerb

2. Rad sa stekom. Sastaviti program za računanje jednog broja y po nizu formula y=0, y=(y+c)·2−1, y=(y+b)·2−1, y=(y+a)·2−1. Neka se na kraju programa dvaput odštampa broj y. Staviti a=4, b=5 i c=6, tako da treba da se dobije y=69.

Rješenje. Ovo je program primjerb. Na početku rada programa primjerb stek je prazan. Tokom izvršavanja programa primjerb stanje u steku se mijenja i to (kada se izvrši naredba):

(push) 4; (push) 4,5; (push) 4,5,6; (pop) 4,5; (pop) 4; (pop) prazan

```
A 100 loaduj na 100,... u tekućem segmentu
MOV AX,4    a=4
PUSH AX     stavi AX na stek
MOV AX,5    b=5
PUSH AX     stavi AX na stek
MOV AX,6    c=6
PUSH AX     stavi AX na stek
MOV AX,0    y=0
POP CX      skini sa steka i stavi u CX
ADD AX,CX   AX←AX+CX (y=y+6=6)
SHL AX,1    left shift 1 place AX (y=2y=12)
DEC AX      decrement AX (y=y−1=11)
POP CX      skini sa steka i stavi u CX
ADD AX,CX   y=y+5=16
SHL AX,1    y=2y=32
DEC AX      y=y−1=31
POP CX      skini sa steka i stavi u CX
ADD AX,CX   y=y+4=35
SHL AX,1    y=2y=70
DEC AX      y=y−1=69='E'
MOV DX,AX DX←y, da bi se odštampalo y
MOV AH,2 priprema za štampanje jednog slova
INT 21      štampaj
INT 21      štampaj
INT 20      halt
␣           izađi iz loadovanja
G           run executable
```

Kada sve ovo otkucamo (otkucamo pošto smo pozvali debug.exe) onda će na ekranu pisati EE.

26. primjerc

3 primjerc. Učitavanje pojedinačnih slova i štampanje niza slova.

Sastaviti program koji će na ekranu prikazati poruku NIZ␣SLOVA. Koristiti prekid broj $(21)_{16}$ tj. koristiti naredbu INT 21 i to njegovu funkciju

$(9)_{16}$ – štampanje niza slova. Neka na ekranu piše u prvom redu `NIZ␣SLO` a u drugom redu `VA`. Prva dva slova N i I učitati preko tastature tokom izvršavanja programa. Koristiti prekid broj $(21)_{16}$ i to njegovu funkciju $(1)_{16}$ – učitavanje jednog slova sa odjekom. Ostala slova zadati programski.

Znamo da se funkcija prekida INT 21 zadaje upisivanjem odgovarajućeg broja u AH. Znamo da definicija funkcije AH=9 glasi: štampaj redom slova počev od mjesta DX (počev od mjesta DS:DX) pa naprijed, sve dok se ne naiđe na mjesto u kome piše slovo $. Funkcija AH=1: slovo koje se učitava dolazi u AL. Funkcija AH=8: učitavanje jednog slova bez odjeka, slovo koje se učitava dolazi u AL.

Naš raspored upotrebe memorije: na 100,101, ... naredbe, a na 200,201,... podaci (u tekućem segmentu). Znamo da `debug.exe` koristi heksadekadni brojni sistem.

Rješenje:

```
A 100           load point = 100
MOV AH,1        funkcija prekida
INT 21          prekid
MOV [200],AL    prvo slovo ide na svoje mjesto
INT 21          prekid
MOV [201],AL    drugo slovo ide na svoje mjesto
MOV AH,9        funkcija prekida
MOV DX,200      gdje počinje niz slova
INT 21          prekid
INT 20          halt
␣               nema više
A 202           load point = 202
DB 5A           'Z'
DB 20           '␣'
DB 53           'S'
DB 4C           'L'
DB 4F           'O'
DB D            CR carriage return
DB A            LF line feed
DB 56           'V'
DB 41           'A'
DB 24           '$'
␣               nema više
G               run executable (go)
```

Otkucati NI. Sada na ektanu piše

`NINIZ␣SLO`

`VA`

Promijenimo prvu naredbu: neka umjesto MOV AH,1 sada bude MOV AH,8. Otkucati NI. Sada na ekranu piše

`NIZ␣SLO`

`VA`

27. primjerd

4 `primjerd`. Rad sa nizom. Ovaj primjer služi da se ilustruje mogućnost indirektnog adresiranja.

Sastaviti program za sabiranje svih članova jednog niza brojeva (za računanje zbira četiri broja).

Plan. Zbir se drži na adresi 200 a sabirci se drže na adresama 202, 204, 206 i 208. Sabirci se zadaju programski, a zbir se ostavlja u AX. Registar BX služi kao indeks–registar.

Imamo da je BX=202, BX=204, BX=206 i BX= 208 tokom rada programa. Tako da je [BX]= [202], ..., [BX]=[208]. Prema tome, glavnu ulogu u programu ima naredba ADD AX,[BX].

U programu se ponavlja: dodaj sabirak i ispitaj treba li još.

Rješenje:

```
      A 100          load point = 100
      MOV BX,202     BX←202
→ 103 MOV AX,[200]   AX←c(200)
      ADD AX,[BX]    AX←AX+c(BX)
      MOV [200],AX   c(200)←AX
      CMP BX,208     compare
      JE 115         jump if equal
      INC BX         BX←BX+1
      INC BX         BX←BX+1
      JMP 103        jump
→ 115 INT 20         halt
      ␣              nema više
      A 200          load point = 200
      DW 0      y=0, y=a1+a2+a3+a4=3CA
      DW F1          a1=F1
      DW F2          a2=F2
      DW F3          a3=F3
      DW F4          a4=F4
      ␣              nema više
      T ponovo T itd. trace
```

Na svako T (izvršavanje korak po korak) imamo na ekranu sadržaj procesorovih registara.

Vidimo da je na kraju AX=03CA.

Poslije izvršavanja naredbe INT vidimo adresu CS:IP na kojoj počinje kod rutine.

28. UPOTREBA PROGRAMA DEBUG.EXE

Mi koristimo program debug.exe za propuštanje programa na jeziku asemblera. Preliminarno nabrajamo neke glavne mogućnosti koje program pruža. U memoriju mogu direktno da se upisuju naredbe ili podaci. Ovo se ostvaruje pomoću naredbe A (assemble). Može da se pročita–vidi na ekranu sadržaj jedne oblasti memorije. Ovo se ostvaruje pomoću naredbe D (dump). U programski brojač CS:IP može da se upiše određena vrijednost i da računar počne da izvršava počev od te vrijednosti. Ovo se ostvaruje pomoću naredbe G (go). Program može da se izvršava korak po korak, s tim da se poslije svakog izvršenog koraka, tj. naredbe na ekranu prikaže tekući sadržaj procesorovih registara. Ovo se ostvaruje pomoću njegove naredbe T (trace). Itd. Vidimo da program omogućava dobar uvid u dešavanja u računaru, odnosno otklanjanje grešaka. Program debug.exe upravo i jeste jedan debager. Vidjećemo da pomoću programa mogu da se proizvode izvršni programi čija je ekstenzija .com, kao recimo primjera.com. Znamo da u DOS–u postoje dvije ekstenzije koje odgovaraju izvršnim programima i to .com i .exe. Znamo da su programi vrste .com jednostavniji od onih .exe. U slučaju .com program čitav stane u jedan segment memorije, tj. RAM–a, gdje je jedan segment = 64 KB. Pored toga, izvršavanje će početi na naredbi čiji je ofset (Hex) 0100. U slučaju .exe, program uopšte uzev zauzima nekoliko segmenata, ima posebne segmente za naredbe, za podatke i za stek, startna adresa treba da bude definisana, itd. Ako je već generisan fajl primjera.com onda se naravno može izaći iz debagera i onda pozvati izvršni program primjera.com (iz operativnog sistema).

Postoje i drugi programi za propuštanje i popravljanje–debagiranje programa sastavljenih na jeziku asemblera. Oni pored ostalog omogućavaju da se proizvede izvršni program čija je ekstenzija .exe. Razlikuju se od našeg izbora po tome što je njihova upotreba složenija i što se teže razumiju (od strane programera). Primjeri takvih programa su symdeb, masm firme Microsoft, tasm firme Borland, nasm i a86.

Intelovi procesori familije x86 do uključeno Pentiuma su međusobno (hardverski) kompatibilni naviše: 80286 razumije program koji je nekad ranije bio sastavljen za 8086 i slično. Isto tako, razne verzije operativnih sistema DOS i Windows su sve zajedno međusobno (softverski) kompatibilne naviše.

Kako se poziva program debug.exe? U slučaju računara čiji je operativni sistem DOS, dovoljno je da se računar uključi. Ako imamo računar koji radi pod operativnim sistemom Windows 98 onda treba pozvati MS–DOS prompt. To se uradi posredstvom opcije "Programs". Kada je riječ o operativnom sistemu Windows XP, treba pozvati command prompt. To se uradi posredstvom "Programs" i "Accessories". Bez obzira na slučaj, sada je ekran u jednostavnom tekstualnom modu. I ostaće tokom našeg rada u tom tzv. režimu karaktera. Sada na ekranu piše primjera radi `C:\WINDOWS>` tj. napisano je ime tekućeg foldera. Program se može pozvati iz bilo kog foldera. Radi se o uobičajenom načinu pozivanja, kada smo prešli u prompt. Treba jednostavno otkucati debug (i pritisnuti enter). Nakon toga, na ekranu će se pojaviti debagerov prompt, a upravo znak crtica (minus). Nakon prompta unosimo naredbu, na početku rada ili uopšte tokom rada. Recimo otkucamo A 0100 (enter). Jedno slovo čini ime jedne naredbe, a ukupno ima cirka 24 vrste naredbi.

Postoji i drugi način da se program pozove. U slučaju Windows 98: "Start", "Run", command (enter), debug (enter). U slučaju Windows XP: "Start", "Run", cmd (enter), debug (enter).

U nastavku se daju definicije naredbi, počev od onih koje se više koriste i o kojima se ovdje više govori, prema onima koje se rijetko koriste. Zatim slijedi opis ukupnog rada prilikom upotrebe programa debug.exe na dva primjera.

Sintaksa A [address] Smisao Počinje unošenje naredbi i podataka u memoriju i to naravno počinje od naznačene adrese. Naredba ili svejedno podatak pretvara se iz simboličkog, tj. asemblerskog oblika kako ga mi kucamo u binarni (mašinski) oblik i unosi se, tj. loaduje se na odgovarajuće mjesto u memoriji. Primjer A 100. Vidimo da je na ekranu prikazana adresa (fizička adresa) na koju će

naredba da bude unesena. Vidimo da se tokom unošenja niza naredbi ta adresa postepeno povećava, odnosno da se adresa sama podešava. Recimo, mi unosimo naredbu MOV AX, [200] (pritisnuti enter). Ili unosimo podatak DB FF. I slično. Iz naredbe A izlazi se tako što se otkuca jedna prazna linija (pritisne se jedno enter više).

Uobičajeno je da loadovanje počinje od adrese 100. Obavezni smo da tako radimo jedino ako želimo da proizvedemo izvršni fajl vrste .com. Naime, za takve fajlove, DOS traži da se njemu ostavi prvih 256 bajta u segmentu, bajtovi od Hex 0 do Hex FF, jedna i druga granica su uključene. DOS koristi taj prostor za neke svoje potrebe. Taj prostor naziva se Program Segment Prefix ili skraćeno PSP. Recimo, vidi se da prva naredba u tom prostoru (počinje na adresi 0) jeste INT 20. Sistemski prekid broj 20 okončava rad po izvršnom programu i vraća upravljanje operativnom sistemu.

U programu debug.exe važi heksadekadni brojni sistem.

Sintaksa G [=address] [breakpoints] Smisao G [=address] znači da će početi izvršavanje od naznačene adrese. Primjer G =100. Smisao G [=address] [addresses] znači da je još definisano i nekoliko (od 1 do 10) tzv. breakpoints. Na breakpoint će štampati sadržaj procesorovih registara (kao u slučaju naredbe T) i onda će čekati da se pritisne enter.

Sintaksa D [range] Smisao Prikazuje sadržaj naznačene oblasti memorije. Ako se otkuca samo jedna adresa onda se ona tumači kao prva adresa i onda se prikaže sadržaj 128 bajta, tako je u našim primjerima. Sadržaj vidimo na ekranu u heksadekadnom obliku i u ASCII obliku. Primjer D 0 Prikazaće od početka tekućeg segmenta, prikazaće ukupno 128 bajta. Primjer D 100 Prikazaće od mjesta gdje je po pravilu bilo počelo loadovanje naredbi i podataka. Primjer D 0F6E:0100 Prikazaće od naznačenog mjesta u memoriji (u RAM–u). Primjer D 0:0 Prikazaće od samog početka memorije (memorijskog prostora).

Posljednji primjer D 0:0 je zanimljiv. Mi vidimo tabelu vektora prekida. Redom četiri po četiri vrijednosti čine jednu punu adresu, čine jednu RAM adresu. Prva četvorka odnosi se na INT 0, sljedeća na INT 1, itd. To je adresa na kojoj počinje rutina prekida. Ako četvorku čine vrijednosti a, b, c i d onda adresa glasi CS:IP = dc:ba. Jedna vrijednost – dva heksadekadna znaka.

Sintaksa T [=address] [number] Smisao Izvršiti naredbu koja stoji na naznačenoj adresi, itd. Ukupno izvršiti onoliko naredbi koliko kazuje number. Onda stati. Tada prikazati na ekranu: sadržaje svih procesorovih registara, posebno i sadržaje flagova, tekuću naredbu (naredbu koja je došla na red za izvršavanje) i još – ako se ta naredba obraća memoriji – i sadržaj odgovarajuće memorijske lokacije. Primjer T =0123:89AB 40 Primjer T =0123:89AB Izvršiće jednu naredbu. Primjer T =140 Izvršiće jednu naredbu. Upravo, izvršiće naredbu u tekućem segmentu čija je adresa (čiji je ofset) 140. Primjer T Ovo je obični slučaj. Izvršiće jednu naredbu, tj. izvršiće tekuću naredbu i još će naravno prikazati sadržaje procesorovih registara. Ako je u naredbi T izostavljeno number onda se podrazumijeva da je number jednako jedan. Ako je izostavljeno =address onda se podrazumijeva tekuća vrijednost, tj. podrazumijeva se trenutna vrijednost programskog brojača CS:IP.

Sintaksa U [range] Smisao Unassemble. Dizasemblira sadržaj naznačenog dijela memorije, odnosno prevodi taj sadržaj iz binarnog oblika u simbolički oblik, simbolički oblik prikazuje se preko ekrana. Simbolički oblik je čitljiv za programera. Primjer U 100 110 Prikazaće sadržaj 16 bajta memorije, od adrese 0100 u tekućem segmentu CS do znači adrese 010F u istom segmentu. Ako se izostavi druga adresa (bila je 110 maločas) onda će dizasemblirati 16 bajta. Primjer U 100 Primjer U 0123:89AB Ako se izostavi i prva adresa onda se podrazumijeva da je prva adresa jednaka tekućoj vrijednosti CS:IP. Primjer U

Pomoću U možemo da vidimo šta smo sami ukucali u memoriju (recimo U 100). Pomoću U možemo da pročitamo kako glasi nekakvi program, samo treba da znamo njegovu prvu adresu. Pomoću U možemo da pročitamo kako glasi neka rutina za obradu prekida (recimo U 00C9:0F9E), jer smo prethodno saznali vektore prekida (jer smo prethodno saznali prve adrese) pomoću D 0:0 i D 0:80 i slično.

Sintaksa Q Smisao Quit. Izlazi se iz debagera.

Sintaksa ? Smisao Help. Na ekranu se prikazuju moguće naredbe debagera kao i njihova sintaksa.

Sintaksa R [register] Smisao Displays or changes the contents of one or more of the processor registers.

Sintaksa N itd. Smisao Name. Specifies the name of a file that will be used by the L or the W command.

Sintaksa W itd. Smisao Write. Writes a file or a specific number of sectors to the specified disk.

Prvi primjer. Obični način rada sa programom debug.exe. Pozvati program. Otkucati A 100 (enter). Redom otkucati sve naredbe programa, a upravo:

JMP 108 (enter) DB E (enter) DB 3 (enter) itd. INT 20 (enter)
Otkucati enter (da se izađe iz A). Otkucati G =100 (enter). Dobiće se rezultati i poruka Program terminated normally. Otkucati Q (enter). Izašli smo iz programa.

Drugi primjer. Drugi način rada sa programom debug.exe, kada se proizvodi izvršni fajl. Poslužimo se opet primjerom primjera. Kada smo izašli iz A onda otkucati:

```
RCX
45
N PRIMJERX.COM
W
Q (enter)
```

Vrijednost 45 vezana je za konkretne okolnosti. Počeli smo da unosimo na adresu 100. Kada smo unijeli posljednju naredbu (to je bila naredba INT 20) onda je program spreman da iduću naredbu unese na adresu 145, to vidimo na ekranu. $145 - 100 = 45$. Pomoću R se registru CX dodjeljuje pravilna vrijednost. Ta vrijednost interveniše prilikom proizvodnje .com fajla.

Vidimo da smo između "W" i "Q" na ekranu dobili poruku Writing 00045 bytes.

Sada se fajl primjerx.com nalazi zapisan na hard disku.

29. 8086 – NAREDBE, NAREDBE ZA PRENOS PODATAKA

Biće navedene sve vrste naredbi mikroprocesora Intel 8086. Naredbe mogu da budu podijeljene u sljedećih šest grupa: a) naredbe za prenos podataka, b) naredbe za rad sa nizom slova, c) aritmetičke naredbe, d) logičke naredbe, e) naredbe za prenošenje upravljanja i f) naredbe za upravljanje procesorom. Skraćenica src znači source ili izvor, a skraćenica dest znači destination ili odredište. Za razumijevanje smisla naredbi, važne su sljedeće oznake:

reg – procesorov registar opšte namjene (8 bita ili 16 bita)

mem – sadržaj memorijske lokacije

immed – immediate tj. neposredna vrijednost

reg16 – procesorov registar opšte namjene čija je veličina 16 bita

mem16 – sadržaj memorijske lokacije čija je veličina 16 bita

Mi vježbamo programiranje na jeziku asemblera korišćenjem DOS–ovog (Windows–ovog) programa debug.exe. Znamo da debug.exe interpretivno izvršava naš program na jeziku asemblera čije je ime recimo primjer.asm. U programu primjer.asm pojavljuju se naredbe. U tom programu još mogu da se pojave i sljedeće dvije tzv. asemblerske direktive. To su direktive za definisanje podatka. Pomoću direktive se u memoriji izdvaja–rezerviše prostor za podatak i podatku se dodjeljuje početna vrijednost n. U prvom slučaju izdvaja se prostor čija je veličina jedan bajt, a u drugom slučaju izdvajaju se dva bajta. DB dolazi od define byte, a DW dolazi od define word:

`DB n` gdje je $0 \leq n \leq 255$

`DW n` gdje je $0 \leq n \leq 65535$

Vrijednost n prikazuje se u heksadekadnom brojnom sistemu. Ako piše DB ? ili DW ? onda to znači da početna vrijednost nije dodijeljena.

a) Naredbe za prenos podataka (ne utiču na flagove)

Sintaksa `MOV dest, src` Smisao dest ← src. Mogućnosti su sljedeće:

```
MOV reg, {reg|mem|immed}
MOV mem, {reg|immed}
MOV {reg16|mem16}, {CS|DS|ES|SS}
MOV {DS|ES|SS}, {reg16|mem16}
```

Navedimo nekoliko primjera za prvu mogućnost:

`MOV AX, BX` znači AX ← BX

`MOV AL, BL` znači AL ← BL

`MOV AX, [4]` znači AX ← c(4) (dva bajta), drukčije zapisano AH ← c(5), AL ← c(4)

`MOV AL, [7]` znači AL ← c(7) ili svejedno AL ← c(DS:7)

`MOV AX, 1C` znači AX ← 28 dekadno

`MOV AL, 1E` znači AL ← 30 dekadno

`MOV AX, [BX]` znači AX ← c(BX)

`MOV AX, [BX+1]` znači AX ← c(BX + 1)

(vidimo da u naredbi [n] znači sadržaj od n,
vidimo da u našem objašnjenju c znači contents tj. sadržaj)

Sintaksa `PUSH src` Smisao Stavlja vrijednost na vrh steka. Upravo SP ← SP – 2, [SP] ← src (dva bajta), gdje je src = {reg16|mem16|CS|DS|ES|SS}.

Sintaksa `POP dest` Smisao Uzima vrijednost sa vrha steka i stavlja je na naznačeno mjesto. Upravo dest ← [SP] (dva bajta), SP ← SP + 2, gdje je dest = {reg16|mem16|DS|ES|SS}.

Sintaksa `XCHG reg, {reg|mem}` Smisao Dvije vrijednosti zamijene mjesta (exchange).

Sintaksa `IN {AL|AX}, {DX|immed(1byte)}` Smisao Input, ulaz preko ulaznih vrata. Donesi u akumulator (to je prvi argument naredbe) vrijednost koju sadrži port čiji broj figuriše (to je drugi argument naredbe). Drugim riječima, `IN acm, port` znači unesi bajt ili riječ u akumulator. Primjer IN AL, 80 znači ulaz kroz vrata $(80)_{16} = 128$, zapaziti da je $0 \leq 128 \leq 255$. Primjer IN AL, DX znači ulaz preko vrata navedenih u DX, u ovom slučaju može se adresirati 65536 ulazno–izlaznih uređaja.

Sintaksa `OUT {DX|immed(1byte)}, {AX|AL}` Smisao Output, stavlja u naznačeni port vrijednost akumulatora. Primjer OUT 20, AL znači izlaz kroz vrata $(20)_{16}$.

Sintaksa `XLAT` Smisao AL $\leftarrow$ [BX + AL]. Dakle Ostvaruje radnju prevođenja jednog koda od jednog bajta u drugi kod od jednog bajta, po tabeli.

Simtaksa `LEA reg, mem` Smisao reg $\leftarrow$ adresa od mem. Load effective address. Dakle Efektivna adresa drugog argumenta naredbe (to je mem) dodjeljuje se da bude vrijednost prvog argumenta naredbe (to je reg).

Sintaksa `LDS reg, mem32` Smisao reg $\leftarrow$ [mem], DS $\leftarrow$ [mem + 2]. Dakle Iz memorije se donose 4 bajta. Dva niža bajta upisuju se u naznačeni registar reg. A dva viša bajta upisuju se u registar DS.

Sintaksa `LES reg, mem32` Smisao reg $\leftarrow$ [mem], ES $\leftarrow$ [mem + 2].

Sintaksa `LAHF` Smisao U registar AH preslikavaju se vrijednosti osam nižih bita flag–registra (osam nižih bita registra PSW). Upravo AH $\leftarrow$ SF:ZF:X:AF:X:PF:X:CF.

Sintaksa `SAHF` Smisao U osam nižih bita registra flagova PSW upisuje se vrijednost registra AH. Upravo SF:ZF:X:AF:X:PF:X:CF $\leftarrow$ AH.

Sintaksa `PUSHF` Smisao U stek upisuje flagove. Upravo SP $\leftarrow$ SP − 2, [SP] $\leftarrow$ PSW. Znsmo da je stek usmjeren naniže: recimo, ako se steku dodaje onda se pokazivač steka smanjuje.

Sintaksa `POPF` Smisao U flagove se upisuju vrijednosti iz steka. Upravo PSW $\leftarrow$ [SP], SP $\leftarrow$ SP+2.

b) Naredbe za rad sa nizom slova

Ove naredbe služe da olakšaju i ubrzaju radnje sa nizovima slova (radnje sa niskama, radnje sa stringovima). Drugim riječima, one obuhvataju radnje koje su uobičajene kada se ima posla sa nizom uzastopnih memorijskih lokacija. Gdje jedan član niza zauzima jedan bajt. I gdje se sadržaj lokacije tumači kao karakter.

Samo ćemo nabrojati imena naredbi koje pripadaju ovoj grupi:

```
MOVS  MOVSB  MOVSW
LODS  LODSB  LODSW
STOS  STOSB  STOSW
CMPS  CMPSB  CMPSW
SCAS  SCASB  SCASW
REP REPE/REPZ  REPNE/REPNZ
```

30. 8086 – NAREDBE, ARITMETIČKE NAREDBE

Nastavlja se spisak naredbi.

c) Aritmetičke naredbe (utiču na flagove AF, CF, OF, PF, SF, ZF)

Sintaksa	Smisao
`ADD dest, src`	dest $\leftarrow$ dest + src
`ADC dest, src`	dest $\leftarrow$ dest + src + CF
`SUB dest, src`	dest $\leftarrow$ dest − src
`SBB dest, src`	dest $\leftarrow$ dest − src − CF
`CMP dest, src`	Compare, dest se ne mijenja; utiče samo na flagove, kso prilikom SUB dest, src
`INC dest`	dest $\leftarrow$ dest + 1, ne utiče na CF
`DEC dest`	dest $\leftarrow$ dest − 1, ne utiče na CF
`NEG dest`	dest $\leftarrow$ −dest

gdje je `dest={reg|mem}`, `src={reg|mem|immed}` i ne mogu oba da budu mem.

Naredbe

```
DAA  DAS  AAA  AAS AAD  AAM
```

služe za sabiranje i oduzimanje brojeva prikazanih u BCD kodu.

Sintaksa `MUL {reg8|mem8}` Smisao Unsigned multiply, ostvaruje množenje dva neoznačena broja od po 8 bita. Upravo AX $\leftarrow$ AL $\cdot$ `{reg8|mem8}`. CF = OF = 0 ako je AH = 0. CF = OF = 1 u suprotnom slučaju. AF, PF, SF i ZF ostaju nedefinisani.

Sintaksa `MUL {reg16|mem16}` Smisao Unsigned multiply, ostvaruje množenje dva neoznačena broja od po 16 bita. Upravo DX:AX $\leftarrow$ AX $\cdot$ `{reg16|mem16}`. CF = OF = 0 ako je DX = 0. CF = OF = 1 u suprotnom slučaju. AF, PF, SF i ZF ostaju nedefinisani.

Sintaksa `IMUL {reg8|mem8}` Smisao Signed multiply, ostvaruje množenje dva označena broja od po 8 bita. Upravo AX $\leftarrow$ AL $\cdot$ `{reg8|mem8}`, proizvod sa predznakom. CF = OF = 0 ako rezultat može da stane u jedan bajt. CF = OF = 1 u suprotnom slučaju. AF, PF, SF i ZF ostaju nedefinisani.

Sintaksa `IMUL {reg16|mem16}` Smisao Signed multiply, ostvaruje množenje dva označena broja od po 16 bita. Upravo DX:AX $\leftarrow$ AX $\cdot$ `{reg16|mem16}`, proizvod sa predznakom. CF = OF = 0 ako rezultat staje u dva bajta. CF = OF = 1 u suprotnom slučaju. AF, PF, SF i ZF ostaju nedefinisani.

Slično, naredbe za dijeljenje su `DIV src` i `IDIV src`. Ako se računa riječ podijeljeno sa riječ onda je prije izvršavanja naredbe bilo djeljenik = AX i djelilac = src a poslije izvršavanja naredbe će biti količnik = AL i ostatak = AH. Ako se računa dvostruka riječ podijeljeno sa dvostruka riječ onda je prije djeljenik = DX:AX i djelilac = src a poslije će biti količnik = AX i ostatak = DX. Znamo da riječ znači dvobajt.

Sintaksa `CBW` Smisao Convert byte to word, proširenje predznaka od AL na AX. Upravo AX $\leftarrow$ AL. Ova naredba ne utiče na flagove.

Sintaksa `CWD` Smisao Convert word to doubleword, proširenje predznaka od AX na DX:AX. Upravo DX:AX $\leftarrow$ AX. Ova naredba ne utiče na flagove.

d) Logičke naredbe (utiču na flagove AF, CF, OF, PF, SF i ZF)

Sintaksa `AND dest, src` Smisao Računa se konjunkcija bit po bit između odredišta i izvora. Upravo dest $\leftarrow$ dest and src.

Sintaksa `TEST dest, src` Smisao Ova naredba ne mijenja sadržaj odredišta. Jedini učinak ove naredbe jeste to što ona utiče na flagove, i to isto kao prilikom izvršavanja naredbe AND dest, src.

Sintaksa `OR dest, src` Smisao Disjunkcija bit po bit. Upravo dest $\leftarrow$ dest or src.

Sintaksa `XOR dest, src` Smisao Ekskluzivna disjunkcija bit po bit. Upravo dest $\leftarrow$ dest xor src.

U četiri prethodne naredbe uradi se CF = OF = 1, AF ostaje nedefinisan, a vrijednosti PF, SF i ZF zavise od rezultata.

Sintaksa `NOT dest` Smisao Negacija bit po bit. Upravo dest $\leftarrow$ not dest.

Sintaksa `SHL/SAL dest, {1|CL}` Smisao Pomijeranje ulijevo za jedno mjesto odnosno za CL mjesta. Upravo (kada je za jedno mjesto) CF $\leftarrow a_{k-1}$, $a_{k-1} \leftarrow a_{k-2}$, ..., $a_1 \leftarrow a_0$, $a_0 \leftarrow 0$, gdje je dest = $a_{k-1} \ldots a_0$.

Sintaksa `SHR dest, {1|CL}` Smisao Logičko pomijeranje udesno za jedno mjesto odnosno za CL mjesta. Upravo (kada je za jedno mjesto) CF $\leftarrow a_0$, $a_0 \leftarrow a_1$, ..., $a_{k-2} \leftarrow a_{k-1}$, $a_{k-1} \leftarrow 0$.

Sintaksa `SAR dest, {1|CL}` Smisao Aritmetičko pomijeranje udesno za jedno mjesto odnosno za CL mjesta. Upravo (kada je za jedno mjesto) CF $\leftarrow a_0$, $a_0 \leftarrow a_1$, ..., $a_{k-2} \leftarrow a_{k-1}$, $a_{k-1} \leftarrow a_{k-1}$.

Sintaksa `ROL dest, {1|CL}` Smisao U dest se vrši kružno pomijeranje ulijevo za jedno mjesto odnosno za CL mjesta. Biće CF = a_{k-1} (ako za jedno mjesto).

Sintaksa `ROR dest, {1|CL}` Smisao U dest se vrši kružno pomijeranje udesno za jedno mjesto odnosno za CL mjesta. Biće CF = a_0 (ako za jedno mjesto).

Sintaksa `RCL dest, {1|CL}` Smisao U CF:dest vrši se kružno ponijeranje ulijevo za jedno mjesto odnosno za CL mjesta.

Sintaksa `RCR dest, {1|CL}` Smisao U dest:CF vrši se kružno ponijeranje udesno za jedno mjesto odnosno za CL mjesta.

31. 8086 – NAREDBE, NAREDBE ZA PRENOŠENJE UPRAVLJANJA

U ovom naslovu se nastavlja i završava spisak naredbi procesora Intel 8086, uz napomenu da te naredbe važe i za procesore koji su kasnije došli (Intel 80286, ...).

e) Naredbe za prenošenje upravljanja (ne utiču na flagove)

ea) Naredba za bezuslovni skok

Sintaksa `JMP n` Smisao Bezuslovni skok na adresu n. Ova naredba odgovara naredbi GOTO u višim programskim jezicima.

Navedimo nekoliko primjera naredbe za bezuslovni skok:

`JMP d` znači IP $\leftarrow$ d, a CS se ne dira (ostajemo u istom segmentu). Recimo, `JMP 4` znači IP $\leftarrow$ 4

`JMP s:d` znači CS $\leftarrow$ s, IP $\leftarrow$ d. Recimo, `JMP 5:6` znači CS $\leftarrow$ 5, IP $\leftarrow$ 6

`JMP AX` znači IP $\leftarrow$ AX, a CS se ne dira (ostajemo u istom segmentu). Ovo je naredba za indirektni skok

`JMP [7]` znači IP $\leftarrow$ c(7), a CS se ne dira (ostajemo u istom segmentu). Ovo je naredba za indirektni skok

eb) Uslovni skokovi, aritmetički označeni (sa brojevima koji su pozitivni, nula ili negativni)

`JL/JNGE etiketa`	Skok na etiketu ako je manji. Ako je SF $\neq$ OF
`JLE/JNG etiketa`	Skok ako je manji ili jednak. Ako je ZF = 1 ili SF $\neq$ OF
`JG/JNLE etiketa`	Skok ako je veći. Ako je ZF = 0 i SF = OF
`JGE/JNL etiketa`	Skok ako je veći ili jednak. Ako je SF = OF

ec) Uslovni skokovi, aritmetički neoznačeni (sa brojevima koji su pozitivni ili nula)

`JA/JNBE etiketa`	Skok ako je veći. Ako je CF = 0 i ZF = 0
`JBE/JNA etiketa`	Skok ako je manji ili jednak. Ako je CF = 1 ili ZF = 1
`JCXZ etiketa`	Skok ako CX sadrži nulu. Skok ako je CX = 0

ed) Uslovni skokovi koji zavise od vrijednosti flagova

`JAE/JNB/JNC etiketa`	Skok ako je veći ili jednak. Ako je CF = 0
`JB/JNAE/JC etiketa`	Skok ako je manji. Ako je CF = 1
`JE/JZ etiketa`	Skok ako je jednak. Ako je ZF = 1
`JNE/JNZ etiketa`	Skok ako je različit. Ako je ZF = 0
`JNO etiketa`	Skok ako nije prekoračenje. Ako je OF = 0
`JO etiketa`	Skok ako je prekoračenje. Ako je OF = 1
`JNS etiketa`	Skok ako nije predznak. Ako je SF = 0
`JS etiketa`	Skok ako jeste predznak. Ako je SF = 1
`JP/JPE etiketa`	Skok ako je parnost parna. Ako je PF = 1
`JNP/JPO etiketa`	Skok ako je parnost neparna. Ako je PF = 0

Objašnjenje za eb) – ed):

Ove naredbe dolaze u programu po pravilu poslije aritmetičke ili logičke operacije. Rezultat operacije utiče na flagove, a flagovi sa svoje strane (kao što smo maločas vidjeli) odlučuju da li će ili neće biti izvršen skok, kao naredba IF u višim programskim jezicima. Često u programu ove naredbe dolaze poslije naredbe za upoređivanje CMP (compare), koja pripada aritmetičkim naredbama.

Objašnjenje za eb) – ed):

Šta je to etiketa ili svejedno lokalna labela ili svejedno kratka labela? Etiketa ne može biti dalje od -128 bajta ili $+127$ bajta od mjesta gdje se nalazi naredba prelaska u kojoj se etiketa pojavljuje. To praktično znači da možemo da skočimo na naredbu koja je približno tridesetak naredbi ili manje udaljena od naredbe pomoću koje se skače.

ee) Naredbe za petlje ili svejedno uslovni skokovi koji koriste registar CX u svojstvu brojača

Sintaksa `LOOP etiketa` Smisao CX $\leftarrow$ CX $-$ 1 i skoči na etiketu ako je CX $\neq 0$. Dakle Sve dok je CX $\neq 0$, CX se smanjuje za 1 i ponavljaju se naredbe između etikete i naredbe LOOP.

Sintaksa `LOOPE/LOOPZ etiketa` Smisao CX $\leftarrow$ CX $-$ 1 i skoči na etiketu ako je ZF $= 1$ i CX $\neq 0$. Dakle Sve dok je ZF $= 1$ i CX $\neq 0$, CX se smanjuje za 1 i ponavljaju se naredbe između etikete i naredbe LOOPE.

Sintaksa `LOOPNE/LOOPNZ etiketa` Smisao CX $\leftarrow$ CX-1 i skoči na etiketu ako je ZF $= 0$ i CX $\neq 0$. Dakle Sve dok je ZF $= 0$ i CX $\neq 0$, CX se smanjuje za 1 i ponavljaju se naredbe između etikete i naredbe LOOPNE.

Već je rečeno da etiketa ne može biti dalja od -128 bajta ili $+127$ bajta od mjesta gdje se nalazi naredba prelaska u kojoj se etiketa pojavljuje.

ef) Naredbe za rad sa potprogramima

Sintaksa `CALL n` Smisao Idi na potprogram koji počinje na mjestu n. Upravo Upisati odgovarajuću povratnu adresu u stek i prenijeti upravljanje na prvu naredbu potprograma. Potprogram n može da bude oblika NEAR ili oblika FAR. Ako je u pitanju potprogram oblika NEAR onda se u stek upisuje samo IP, a ako je oblika FAR onda se u stek upisuju CS i IP. Primjeri:

`CALL 55` znači staro IP ide u stek (tako da SP $\leftarrow$ SP $-$ 2) i IP $\leftarrow$ 55, CS ostaje po starom

`CALL 77:99` znači stari CS i IP idu u stek (tako da SP $\leftarrow$ SP $-$ 4), CS $\leftarrow$ 77 i IP $\leftarrow$ 99

Sintaksa `RET` Smisao Return, return near, povratak iz potprograma. Upravo Uzima sadržaj iz steka i prenosi upravljanje na povratnu adresu. Drugim riječima, IP dobija vrijednost iz steka (tako da SP $\leftarrow$ SP $+$ 2), CS ostaje po starom.

Sintaksa `RET immed` Smisao Ima jedna radnja više u odnosu na RET, a upravo to je radnja SP $\leftarrow$ SP$+$immed. Dakle, IP dobija vrijednost iz steka i SP $\leftarrow$ SP$+2+$immed, CS ostaje po starom.

Sintaksa `RETF` Smisao Return far, povratak iz potprograma. Upravo IP i CS dobijaju vrijednosti iz steka (tako da SP $\leftarrow$ SP $+$ 4).

eg) Naredbe za rad sa rutinama prekida

Sintaksa `INT number`, gdje je number u granicama od 0 do 255. Smisao Pozivanje rutine, idi na rutinu kojoj odgovara broj number. Upravo:

U stek se upisuje PSW (u stek se upisuju flagovi)
Uradi se TF $\leftarrow$ 0 i IF $\leftarrow$ 0
U stek se upisuju CS i IP
CS i IP dobijaju odgovarajuće vrijednosti upravo CS $\leftarrow [0 : 4 \cdot \text{number} + 2]$, IP $\leftarrow [0 : 4 \cdot \text{number}]$

Sintaksa `INTO` Smisao Uslovni poziv rutine. Upravo Ako je OF $= 1$ onda uraditi INT 4. Dakle, poziva se prekid za obradu prekoračenja (overflow). Prilikom poziva postaće CS $=$ [0:12] i IP $=$ [0:10], heksadekadno.

Sintaksa `IRET` Smisao Povratak iz rutine za obradu prekida. Upravo Iz steka se redom ispisuju sadržaji za IP, CS i PSW, pa se tako procesor vraća na prekinuti program.

f) Naredbe za upravljanje procesorom

`CLC` CF $\leftarrow$ 0, briše se flag prenosa
`STC` CF $\leftarrow$ 1, postavlja se flag prenosa
`CMC` CF $\leftarrow 1 -$ CF, komplementira se flag prenosa
`CLD` DF $\leftarrow$ 0, briše se flag smjera
`STD` DF $\leftarrow$ 1, postavlja se flag smjera
`CLI` IF $\leftarrow$ 0, briše se flag za dozvolu prekida, čime se onemogućavaju maskirajući prekidi
`STI` IF $\leftarrow$ 1, postavlja se flag za dozvolu prekida, čime se dopuštaju maskirajući prekidi
`NOP` No operation, naredba bez dejstva, ništa se ne radi

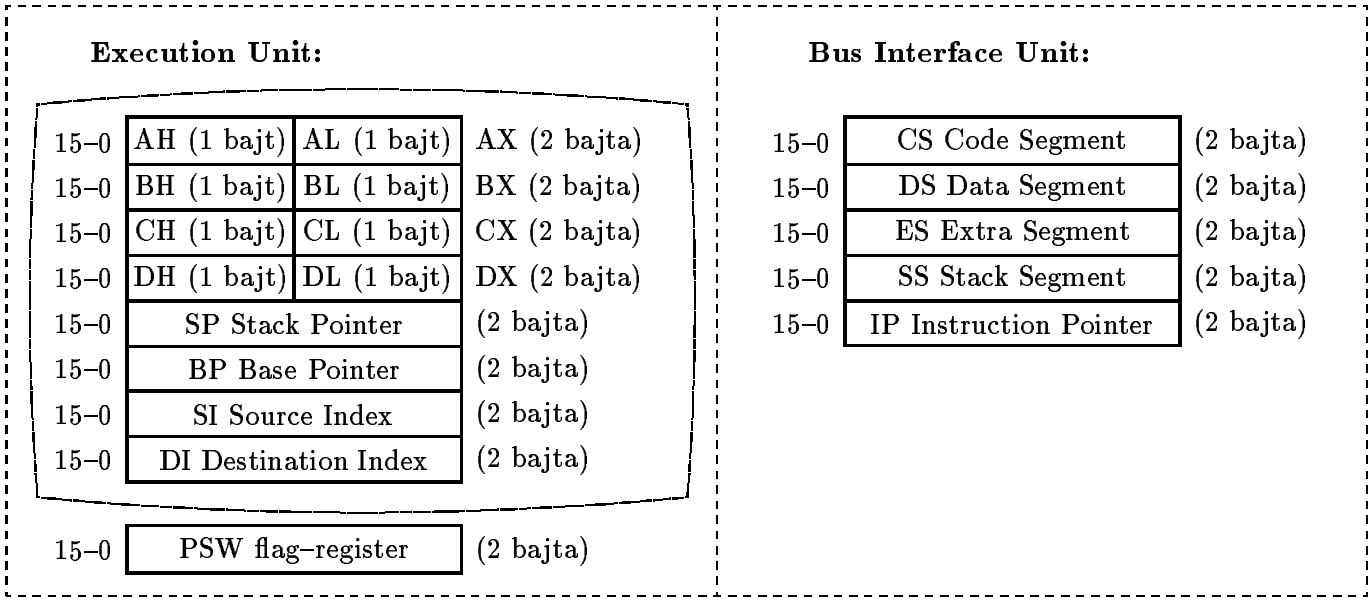
`HLT` Procesor se dovodi u stanje mirovanja. Sada procesor čeka spoljašnji prekid ili signal za restartovanje procesora.

`WAIT` Procesor se dovodi u stanje mirovanja. Svakih 5 taktova on gleda da li je aktivna njegova ulazna linija (njegov pin) TEST. On će preći na sljedeću naredbu iza naredbe WAIT kada ta linija postane aktivna.

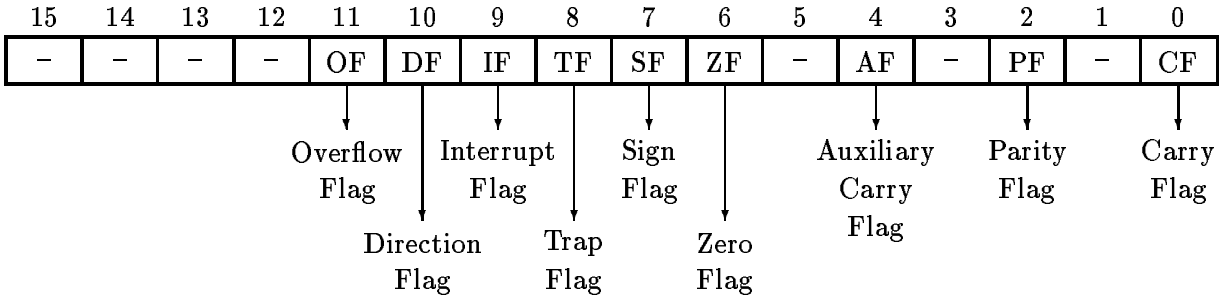
`LOCK` Ako ispred neke naredbe stoji naredba LOCK (stoji prefiks LOCK) onda to znači da je onemogućeno da bi drugi procesori koristili magistrale, sve dok se ne završi izvršavanje te naredbe. Drugi procesori – koprocesor ili DMA kontroler ili slično. Naredba LOCK može da prethodi bilo kojoj naredbi. Smisao Aktivira se pin LOCK. Time se drugim procesorima šalje obavještenje da oni ne smiju da koriste magistrale.

* * * * * * * * *

Slika 1. Glavni registri procesora Intel 8086. Moglo bi se pisati da je AX = AH:AL i slično za BX, CX i DX. Registri opšte namjene su AX, AH, AL, BX, BH, BL, CX, CH, CL, DX, DH, DL, SP, BP, SI i DI (zaokruženi dio na slici).



Slika 2. Struktura registra flagova (Program Status Word), 15–0 PSW:



Slika 3. Kako se definiše adresa segment:offset = s:d. $a = 16s + d$

Prilikom definisanja adrese u memoriji (u RAM–u) važi jednakost $a = 16 \cdot s + d$, gdje je a address, adresa, s segment (broj segmenta) i d dodatak ili svejedno offset. Tako da se a sastoji od 20 bita, jer se s i d sastoje od po 16 bita. Dakle, mogućih adresa ima 2^{20}.

Pojedina adresa pokazuje na jedan bajt u memoriji. Tako da je predviđeno da veličina memorije bude jednaka 2^{20} bajta. Drugim riječima, veličina memorije iznosi 1 MB, jedan mega–bajt.

32. SISTEM PREKIDA PROCESORA INTEL 8086

U ovom naslovu daje se opšta slika sistema prekida u slučaju procesora i8086 ili nekog nasljednika. Biće riječi redom o tabeli vektora prekida, izvorima zahtjeva za prekid, obradi prekida i redosljedu prioriteta među zahtjevima.

Prvih 1024 bajta u memoriji rezervisano je za tzv. tabelu vektora prekida, adrese od 0000:0000 do 0000:03FF. Pojedini član tabele zauzima 4 bajta, tako da ukupno ima mogućnosti za 256 vrsta prekida. Za pojedini član tabele, prva dva bajta definišu vrijednost za IP, a druga dva bajta definišu vrijednost za CS. Tako da jedan član tabele definiše jednu adresu CS:IP u memoriji. To je adresa u memoriji na kojoj počinje kod rutine za obradu prekida. Kada se pristupi obradi prekida čiji je broj n (gdje je $0 \leq n \leq 255$) onda će se izvršiti skok na odgovarajuću adresu.

Pogledajmo posebno slučajeve od $n = 0$ do $n = 4$. Prekid $n = 0$ naziva se divide error ili divide by zero. Zahtjev za prekid nastupiće u rezultatu izvršavanja naredbe za dijeljenje ako je imenilac jednak nuli ili ako je količnik prevelik. Računar će saopštiti "greška prilikom dijeljenja" i neće se vratiti u program koji ima naredbu DIV odnosno IDIV. Prekid $n = 1$ nastupiće poslije izvršavanja bilo koje naredbe ako je u datom trenutku TF = 1, trap flag je setovan. Naziva se single step. Služi za izvršavanje programa naredba po naredba. Očekuje se da je u tabeli vektora prekida (na odgovarajućem mjestu) upisana adresa koja pripada debageru. Prekid $n = 2$ nastupiće ako je pristigao zahtjev preko pina NMI, nemaskirajući hardverski. Taj zahtjev će svakako biti usvojen. Služi kao najava skorog nestanka napona napajanja i za druge slične hitne situacije. Prekid $n = 3$ takođe se odnosi na debager. Nećemo govoriti detaljno o njemu. Ukratko, neke naredbe u programu P koji ima grešaka označe se kao breakpoints naredbe. Debager će dobiti posebno obavještenje kada se tokom debagiranja dođe do takve naredbe. Prekid $n = 4$ odgovara naredbi INTO. To je tzv. prekid prekoračenja. Znamo da on nastupa prilikom izvršavanja naredbe INTO ako je u datom trenutku OF = 1, overflow flag je setovan. Za prekide od $n = 0$ do $n = 4$ kaže se da su rezervisani, da ih je firma Intel rezervisala.

Postoje tri moguća izvora–uzroka da dođe do prekida. Saglasno tome, sve vrste prekida ukupno dijele se u tri kategorije: a) hardverski prekidi, b) softverski prekidi i c) izuzeci. Pogledajmo redom.

a) Ako je prekid izazvan spoljašnjim događajem, gledano iz ugla procesora, onda je to hardverski prekid. Drugim riječima, ako je prekid izazvan signalom poslatim procesoru od strane nekog drugog dijela računarskog sistema. Zahtjev za hardverski prekid predstavlja asinhroni događaj, gledano iz ugla procesora, gledano iz ugla programa tj. niza naredbi čije izvršavanje je u toku. Dakle, zahtjev će se pojaviti tokom izvršavanja neke naredbe, ali se ne može unaprijed determinisati – koja je to naredba.

Zahtjevi pristižu preko pina NMI non–maskable interrupt ili preko pina INTR interrupt request. U slučaju NMI, zahtjev ne može da bude maskiran. A zahtjevi koji pristižu preko pina INTR mogu da budu maskirani: zahtjev će biti usvojen ili će biti odbačen. Za pojedinu instancu zahtjeva oblika INTR, o tome odlučuje tekuća vrijednost interrupt flaga. Ako je IF = 1 onda se ne maskira, usvaja se. Ako je IF = 0 onda se maskira, ne usvaja se. Vidimo da IF ima ulogu maske.

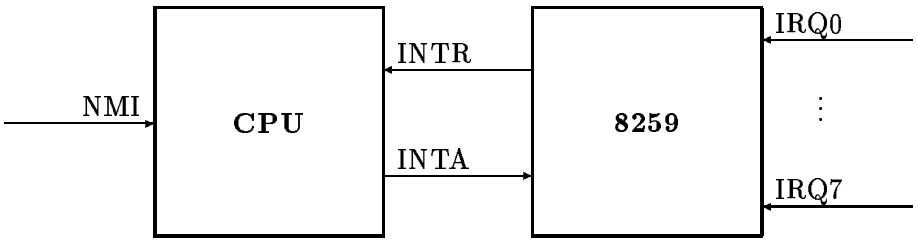
Ulogu posrednika između procesora i drugih djelova računarskog sistema, za potrebe maskirajućih hardverskih prekida, ima jedna posebna hardverska komponenta (integrisano kolo). To je tzv. kontroler prekida. Obično je to komponenta Intel 8259 (ili 8259A). Ima 28 pinova. Naime, može se desiti da nekoliko perifernih uređaja (hard–disk, tastatura i slično) u istom trenutku istaknu svoje zahtjeve. Svi ti zahtjevi idu na kontroler prekida. On ima upisano pravilo o prioritetu zahtjeva (o hijerarhiji). Izabraće jedan zahtjev, odnosno propustiće samo jedan zahtjev. Naravno da će procesoru uputiti zahtjev čiji je prioritet najveći.

Pored ostalih pinova, kontroler ima osam pinova označenih kao IRQ0, IRQ1, ..., IRQ7, interrupt request. Jasno je da jednoj periferiji odgovara jedan od tih pinova. U slučaju više istovremenih zahtjeva IRQk, propustiće se onaj od njih koji ima najmanje k. Vidimo da se pravilo o prioritetu definiše rasporedom na nožicama. Treba vezati periferije na odgovarajuće nožice onako kako to odgovara datom računarskom sistemu. Vidimo da bi se promjenom rasporeda promijenili prioriteti, imali bismo

drukčiju konfiguraciju sistema.

Pored drugih pinova koji povezuju kontrolera sa procesorom, postoje i dva pina koji su označeni (sa strane procesora) kao INTR i INTA. INTR je ulazna nožica. INTA je izlazna nožica, interrupt acknowledgement. Izložimo ukratko mehanizam za saopštavanje broja prekida. Procesor je dobio signal po liniji INTR. Ako je IF $= 1$ onda procesor šalje signal po liniji INTA. Kao odgovor, kontroler šalje procesoru osam binarnih vrijednosti (zavisno od vrste perifernog uređaja), preko osam linija adresne magistrale. Znamo da adresna magistrala ima ukupno 20 linija. Tih osam binarnih vrijednosti obrazuju jedan broj n u granicama $0 \leq n \leq 255$. Naravno, zatim će procesor preći na odgovarajuću rutinu za obradu prekida, ako nema prečih zahtjeva.

Pravilo o korespondenciji (vrsta uređaja $\leftrightarrow$ broj n) nalazi se upisano u kontroleru. Pravilo se upisuje prije njegovog povezivanja sa drugim komponentama sistema. Isto tako, ono može da bude izmijenjeno po potrebi. Zato se za kontroler kaže da je on programabilan, PIC. V. sliku: Hardver za prekide.



Osim glavnog kontrolera, u sistemu je obično prisutan i drugi pomoćni kontroler. Pomoćni kontroler se vezuje na nožicu IRQ2 glavnog kontrolera, po pravilu. U sistemu sa ovakvom konfiguracijom, očito je da 15 perifernih uređaja može da ispostavlja maskirajuće zahtjeve.

b) Znamo da opšti oblik naredbe za pozivanje rutine za obradu prekida glasi INT n. Vidimo da broj prekida n figuriše u samoj naredbi. Dakle, ako je prekid nastupio kao rezultat izvršavanja naredbe INT onda je to softverski prekid. Ovoj kategoriji pripada i INTO.

c) Može se desiti da neke unutrašnje okolnosti u procesoru izazovu prekid. To su tzv. izuzeci. Ovdje pripadaju $n = 0$ divide error i $n = 1$ single step. Završen je pregled kategorija a)–c).

Kada je izvršavanje tekuće naredbe pri kraju, procesor pregleda zahtjeve koji su ispostavljeni. On donosi odluku o tome da li će neki biti usvojen. Ako će biti, onda on i odabira broj n, odnosno odabira jedan zahtjev u slučaju da ih je bilo više. Uzmimo da je neki zahtjev usvojen i da je broj n odabran. Tada dalja dešavanja ne zavise od kategorije prekida ili od čega drugog. Tada, vrše se radnje:

PUSHF, PUSH CS, PUSH IP,	u stek se upisuju PSW, CS i IP
IF $\leftarrow 0$, TF $\leftarrow 0$,	brišu se flagovi IF i TF
IP $\leftarrow [4 \cdot n]$, CS $\leftarrow [4 \cdot n + 2]$.	skok na rutinu

Kasnije će se izvršiti povratak, pomoću naredbe IRET. Odgovarajuće radnje su:

POP IP, POP CS, POPF.	obnavlja se sadržaj tri registra

Rang raznih zahtjeva, odnosno prioritet prekida hardverski je založen u samom procesoru i ne može da bude promijenjen. U slučaju više istovremenih zahtjeva, biće odobren najjači. Spisak mogućih zahtjeva od jačih prema slabijima glasi: i) divide error, softverski prekid, ii) NMI zahtjev, iii) INTR zahtjev, iv) single step. U ovom spisku, "softverski prekid" znači naredba INT ili naredba INTO. i > ii > iii > iv.

33. NEKE FUNKCIJE SISTEMSKOG PREKIDA 21H

U ovom naslovu govori se o DOS–ovom prekidu čiji je broj $(21)_{16}$. Dakle, prekid se poziva pomoću naredbe INT 21. Vidjećemo da prekid 21H ima veliki broj funkcija (da je u stanju da obavi veliki broj raznovrsnih korisnih radnji), mada ćemo mi nabrojati samo nekoliko njegovih funkcija. Kao što znamo iz ranijih primjera programa na jeziku asemblera, za ovaj prekid 21H, njegova funkcija se definiše tako što se u registar AH plasira odgovarajući broj (odgovarajuća vrijednost). Plasira se prije izvršavanja naredbe INT 21. Plasirana vrijednost predstavlja broj funkcije tj. vrstu radnje. Kao što znamo iz prethodnog naslova (o prekidima uopšte), četiri bajta u memoriji (u RAM–u) čije su adrese redom od 0000:0084 do 0000:0087 čine jedan član tabele (tabele vektora prekida). Taj član odnosi se naravno na INT 21. Šta piše na ta četiri bajta (koji je sadržaj upisan u ta četiri bajta)? Odgovor na ovo pitanje zavisi od verzije operativnog sistema DOS, odnosno Windows. Protumačimo taj sadržaj kao da on znači jednu adresu, njegov smisao i jeste adresa upravo. Na jednom slučajno izabranom 486 računaru pročitano je da sadržaj govori segment:offset = 00C9:0FA8. Dakle, tekst rutine broj 21H (kod rutine) počinje upravo na toj adresi segment:offset. Kod rutine zauzima u memoriji iduće bajtove, sve do zaključno bajta čija je adresa segment:offset = ... (može se i ova adresa saznati). Rutina zauzima u memoriji ukupno nekoliko kilo–bajta. Sada počinje nabrajanje funkcija, funkcije su podijeljene u grupe. Počinjemo sa prvom grupom koja se odnosi na jednostavne ulazno–izlazne radnje.

1. Funkcija AH = 2 izdavanje jednog znaka. Znak (karakter) koji želimo da bude odštampan (da bude prikazan na ekranu) treba prethodno plasirati u registar DL. Navedimo mali primjer. Razmotrimo dio programa na jeziku asemblera koji se sastoji od tri naredbe kako slijedi:

```
MOV DL, 41   (DL ← 41, plasira se znak, znamo da je ORD('A') = (41)16 po ASCII)
MOV AH, 2    (AH ← 2, plasira se broj funkcije, tj. navodi se funkcija–radnja prekida)
INT 21       (poziva se rutina prekida, navodi se vrsta prekida)
```

U rezultatu izvršavanja ove tri naredbe, imaćemo veliko slovo A prikazano na ekranu.

2. Funkcija AH = 1 ulaz znaka sa odjekom. Učitava se (preko tastature) jedan znak. Taj znak će da dospije u registar AL. "Sa odjekom" ili "sa ehom" znači da će znak još biti i prikazan na ekranu. Navedimo mali primjer. Razmotrimo dio programa na jeziku asemblera:

```
MOV AH, 1    (AH ← 1, navodi se funkcija prekida)
INT 21       (poziva se rutina)
```

Prilikom izvršavanja ovih naredbi, ako otkucamo recino X onda će u AL biti upisana odgovarajuća vrijednost (biće upisana ASCII vrijednost koja odgovara slovu X, konkretno to je vrijednost $(58)_{16}$, odnosno 88 dekadno) i još će slovo X biti dato na ekranu.

3. Funkcija AH = 8 ulaz jednog znaka bez odjeka. Sve je isto kao kod prethodne funkcije AH = 1, samo što sada nema odjeka, tj. sada nema prikazivanja na ekranu. Za mali primjer napisali bismo dvije naredbe MOV AH, 8 i INT 21.

4. Funkcija AH = 9 izdavanje niske. Pomoću ove funkcije na ekranu će biti prikazan jedan niz slova. Članovi niza zauzimaju u memoriji niz uzastopnih bajta, jedno slovo tj. jedan znak tj. jedan karakter = jedan bajt. Položaj prvog bajta definišu tekuće vrijednosti dva registra DS i DX. Upravo, adresa prvog bajta je segment:offset = DS:DX. Biće prikazan taj bajt i redom idući bajtovi. Sve dok se na tom putu ne naiđe na znak $ kome odgovara ASCII broj $(24)_{16}$. Taj znak $ neće biti prikazan. Pogledajmo mali primjer. Šta će se desiti u rezultatu izvršavanja sljedećih naredbi:

```
MOV DS, 1C    (DS ← (001C)16)
MOV DX, EEE   (DX ← (0EEE)16)
MOV AH, 9     (AH ← 9, izabrana je funkcija prekida)
INT 21        (poziva se rutina za obradu prekida)
```

Za odgovor, treba pogledati šta piše na adresi 001C:0EEE i na idućim adresama i posebno treba pronaći gdje tu piše $ prvi put. U svakom slučaju, na ekranu će biti odštampano nekoliko karaktera.

Nisu sve funkcije prekida INT 21 tako proste kao one iz prve grupe. Vidjećemo da postoje i puno moćnije funkcije. Prelazimo na drugu grupu funkcija: rad sa fajlovima koji su upisani na hard–disku (rad sa datotekama). Biće opisano samo nekoliko funkcija. A ima još puno funkcija u okviru INT 21 za rad sa fajlovima.

1. Funkcija AH = 0F otvaranje fajla. U DS i DX treba pripremiti adresu DS:DX tzv. bloka za upravljanje fajlom FCB, File Control Block. I treba još naravno pripremiti sami FCB. FCB sadrži osnovne podatke o fajlu (naziv fajla i slično). Fajl će da postane dostupan. Ova funkcija vraća tzv. status fajla u AL.
2. Funkcija AH = 10 zatvaranje fajla. Veličine DS, DX i AL imaju isti smisao kao maločas kod prethodne funkcije. Fajl će prestati da bude dostupan.
3. Funkcija AH = 13 brisanje fajla. Veličine DS, DX i AL imaju isti smisao kao maločas.
4. Funkcija AH = 16 kreiranje fajla. Veličine DS, DX i AL imaju isti smisao kao maločas.
5. Funkcija AH = 14 Sequential Read. DS, DX i AL isto. Returns 1 block in DTA. DTA = Disk Transfer Area.
6. Funkcija AH = 15 Sequential Write. DS, DX i AL isto. Block written from DTA.

[Ovaj dio izlaganja treba da bude dopunjen–razrađen. FCB ima tačno određenu strukturu: koliko bajta zauzima, koje podatke sadrži i gdje se (u bloku) koji podatak nalazi. Rutine se tokom svog rada obraćaju samom hard–disku. One čitaju sa hard–diska podatke o samom hard–disku: koliko traka ima, koliko sektora ima, kolika je veličina jednog sektora i drugo. Sami hard–disk sadrži i svoju tzv. tabelu alokacije fajlova, FAT, engl. File Allocation Table.]

Završena priča o drugoj grupi funkcija. Postoji i treća grupa, četvrta grupa, itd. Drukčije rečeno, prekid broj 21 ima desetine i desetine svojih funkcija. Samo ćemo još reći ukratko na šta se te funkcije odnose između ostalog. Upravljanje informacijama, upravljanje direktorijumima, upravljanje procesima, upravljanje memorijom, itd.

34. PREGLED BIOS–OVIH I DOS–OVIH PREKIDA

U ovom naslovu daje se kratak pregled jednog obimnog spiska. To je spisak svih servisa–usluga koje operativni sistem stavlja na raspolaganje programeru. Drugim riječima, to je spisak svih rutina za obradu prekida. BIOS znači Basic Input Output System. DOS znači Disk Operating System (MS–DOS znači MicroSoft Disk Operating System). Znamo da prekidi čiji su brojevi od n = 0 do n = 31 pripadaju BIOS–u. Znamo da DOS–ovim prekidima odgovaraju brojevi od n = 32 do n = 255.

Among the BIOS interrupts there are these categories:
1. Keyboard service 2. Video service 3. Diskette service 4. Fixed disk service 5. Serial communications service 6. System service 7. Parallel printer service 8. Time–of–day and data service 9. Single function service.
Among the DOS interrupts there are these categories:
1. Character input 2. Character output 3. Disk management 4. File management 5. Information management 6. Directory management 7. Process management 8. Memory management 9. Miscellaneous system management.

Pogledajmo bar neke slučajeve broja prekida n.

♠ INT 5 Print Screen. Šalje ASCII sadržaj video bafera (dijela memorije koji čuva sadržaj ekrana) na štampač. Ova rutina nema svojih ulaznih veličina, tj. ne treba pripremiti bilo koji registar. Isto ona nema izlaznih veličina. Njen učinak je da se odštampa trenutni sadržaj ekrana (u karakter modu, u tekstualnom modu, ne u grafičkom modu). Rutina se poziva jednostavno pomoću INT 5 asemblerske naredbe.

♠ INT 10 funkcija AH = 2 Set Cursor Position (postavljanje kursora na određeno mjesto na ekranu). Ulazne veličine su: BH – broj video stranice (osnovnoj stranici odgovara broj 0), DH – vrsta i DL – kolona u koju se želi postaviti kursor. Izlaznih veličina nema. Navedimo sada mali primjer:

```
MOV BH, 0    neka bude strana broj 0
MOV DH, 8    neka broj vrste bude 8
MOV DL, 6    neka broj kolone bude 6
MOV AH, 2    bira se funkcija
INT 10       poziva se rutina (kursor će biti postavljen kako je rečeno)
```

Redni broj vrste može da se kreće u granicama od 0 do 24. Redni broj kolone može da se kreće u granicama od 0 do 79.

♠ INT 20 Terminate program (okončavanje programa). U CS treba da bude adresa segmenta gdje počinje kod (tekst) programa. Oslobađa memoriju koju je program dotad zauzimao i vraća upravljanje računarovom OS–u (operativnom sistemu).

Itd.

35. JEDAN KONKRETAN PRIMJER RUTINE ZA OBRADU PREKIDA

Pogledajmo kako glasi sami tekst (sami kod) rutine koja se poziva naredbom INT 5, a služi za štampanje ekrana (služi da se ono što u datom trenutku piše na ekranu preslika na papir pomoću štampača). Zavisno od verzije DOS–a, tekst rutine (kod rutine) nije uvijek jedan te isti. Kako su se iz godine u godinu hardver i softver razvijali tako se i rutina pomalo mijenjala i dopunjavala, od jedne verzije DOS–a do druge. Može se mijenjati i njena početna adresa (odgovara promjeni petog člana tabele vektora prekida). Mehanizam tabele daje korisniku kompatibilnost ranijeg i novog (stare verzije OS–a i nove verzije OS–a). Aplikativni program koji poziva INT 5 i ne primijeti promjene.

Detaljno razumijevanje čitavog koda rutine je zamorno i ostavlja se za neku bolju priliku.

Iz koda se vidi da rutina na početku stavi na stek i time ih sačuva neke registre

(PUSH DX, PUSH CX, PUSH BX, PUSH AX, PUSH DS),

a na kraju svog rada ona ih vrati, odnosno ona ih uzme iz steka

(POP DS, POP AX, POP BX, POP CX, POP DX, vidimo da je sada redosljed obrnut).

To su registri koje rutina tokom svog rada koristi i (znači) mijenja njihov sadržaj. Rutina je obavezna da ostavi po starom procesorove registre kada IRET (kada povratak u program, povratak iz rutine).

Iz koda rutine se vidi da ona sa svoje strane poziva neke druge rutine. Konkretno, na više mjesta pojavljuju se naredbe INT 10 i INT 17.

Kod rutine počinje na adresi F000:FF54 (to smo saznali čitajući šta piše na adresama od 0000:0014 do 0000:0017). Kod rutine završava se na adresi F000:FFD3. Tako da rutina u memoriji zauzima ukupno 128 bajta dekadno. Kada se izbroji, rutina se sastoji od ukupno 66 naredbi.

Pogledajmo barem u glavnim crtama šta se dešava tokom izvođenja (tokom rada, tokom izvršavanja) rutine. Ona prvo pita kakav je oblik ekrana. Dobije obavještenje da ekran ima oblik 25 vrsta puta 80 kolona. Zatim redom i polako u dvostrukoj petlji (za vrste i kolone) uzima jedan po jedan karakter iz video bafera i odmah šalje štampaču taj karakter.

36. O MIKROPROCESORU INTEL 80286

Procesor 286 pojavio se 1982. godine. Sadrži 134.000 tranzistora. Ima 68 pinova. Procesor obezbjeđuje podršku za tzv. multitasking (za više poslova), tj. da se nekoliko programa izvršava u isto vrijeme. U okviru toga predviđen je i sistem zaštite, da se ne desi da jedan program izmijeni podatke ili naredbe drugog programa. U tom smislu procesor ima dva moda rada (dva načina rada) – realni način (real mode) i zaštićeni način ili protekted način (protected mode) (ponekad se kaže i režim supervizora ili režim operativnog sistema). U realnom načinu on se ponaša potpuno isto kao procesor 8086. U zaštićenom načinu stvari se potpuno mijenjaju. Treba razumjeti da hardver μP samo pruža preduslove za recimo multiprogramski režim rada, odnosno da hardver ovdje unaprijed izlazi u susret (očekivanim) potrebama softvera, a da je zadatak softvera odnosno operativnog sistema da te preduslove iskoristi ili ne iskoristi i kako će ih iskoristiti. Proširen je skup mašinskih naredbi u odnosu na 8086. Proširen je i skup registara. Takođe je i mogući memorijski prostor povećan: postoji mogućnost da se adresuje čitavih 16 MB RAM–a, adresi jednog bajta u memoriji odgovaraju 24 bita. Mikroprocesor Intel 80286 je jedan 16–bitni procesor, kao i njegov prethodnik Intel 8086, dok je već njegov nasljednik Intel 80386 jedan 32–bitni procesor.

Jedan primjer za sistem zaštite. U zaštićenom načinu rada procesora, nikad se ne dopušta da korisnik nešto upisuje u segment koda (u segment koji je namijenjen da sadrži naredbe). A u realnom načinu to može da bude urađeno.

Već je rečeno da nekoliko programa (nekoliko procesa) može da se izvršava istovremeno (oni mogu da rade prepleteno, oni mogu da se izvode prepleteno). Pojedini proces ima ograničenje od 1 MB (jedan mega–bajt) za veličinu svog zauzeća memorije.

Kada se računar uključi (kada se procesoru pošalje signal RESET) onda se procesor nalazi u realnom načinu rada. Ako je u nekom trenutku procesor prešao u zaštićeni način rada onda se više ne može vratiti u realni način rada na jednostavan način, već se može vratiti jedino resetovanjem. Kod procesora nasljednika ovo pitanje je riješeno na povoljniji način za korisnika.

Predviđeno je da se u memoriji drže tokom rada računara tri tabele (tri vrste tabela). One su vezane za zaštićeni način rada. Hardver ima sredstva (ima naredbe) da se sa tim tabelama manipuliše. To su GDT – General Description Table, LDT odnosno LDTs – Local Description Tables (množina) i IDT – Interrupt Description Table. GDT služi kada se sa izvođenja jednog procesa prelazi na izvođenje nekog drugog procesa i slično. Tabela LDT ima onoliko koliko u datom trenutku ima procesa čije je izvođenje u toku. Drugim riječima, svaki proces ima svoju lokalnu tabelu deskriptora. LDT govori gdje se zaista u memoriji nalaze naredbe i podaci tog procesa i slično.

Da se ostvari zaštita, svakom procesu je pridružen njegov tzv. nivo privilegije. Postoje četiri moguća nivoa privilegije, označavaju se brojevima od 0 do 3, gdje je broj 0 najviši nivo (ima se najviše povjerenja). Proces sa manjim brojem ima veća prava. Koriste se sljedeći nazivi:

nivo 0 – jezgro operativnog sistema,
nivo 1 – servisi operativnog sistema,
nivo 2 – proširenja operativnog sistema i
nivo 3 – aplikativni programi.

37. TRIDESETDVOBITNI MIKROPROCESORI

U ovom naslovu govori se o 32–bitnim Intelovim procesorima 80386, 80486 i Pentium. Polazimo sa 386. Znamo da 386 predstavlja proširenje procesora 286, slično 486 predstavlja proširenje od 386 i isto tako Pentium je proširenje od 486.

386 se pojavio 1985. godine. Pogledajmo koje registre ima, kako se vrši adresiranje memorije (imamo u vidu naravno unutrašnju memoriju, tj. RAM) i koje načine (ili modove) rada ima, odnosno može da podrži.

Ranija četiri registra AX, BX, CX i DX (akumulatori) (bili su 16–bitni kod 286) sada su zamijenjeni 32–bitnim registrima EAX, EBX, ECX i EDX, E – extended. U programu na jeziku asemblera mogu

da se pojavljuju naredbe kao MOV EAX, 003F ili kao ADD EAX, EBX. Može se pisati i samo AX ili BX ili CX ili DX, odnosno može se pristupiti posebno donjoj polovini bilo kog od četiri akumulatora. Slično važi i za ostale registre čija skraćenica počinje sa E. Može se isto tako pisati AH ili AL ili ... ili DL, kao u slučaju 8086 asemblera.

Znamo da četiri akumulatora i četiri ofsetna registra čine zajedno osam registara opšte namjene. Sada su četiri ofsetna registra ESP, EBP, ESI i EDI. Svaki od njih je 32–bitni. Program može da pristupi donjoj polovini SP odnosno BP odnosno SI odnosno DI.

386 ima 32–bitni registar flagova EFLAG. Njegova donja polovina (16 bita) FLAG poklapa se po svojoj strukturi sa registrom flagova prethodnika (koji je označavan ako ćemo tačno kao PSW). Dakle, EFLAG je 31–0, a FLAG je njegov dio 15–0. Nisu sva 32 flipflopa u registru EFLAG iskorišćena, odnosno flagova ima manje od 32 ustvari.

EIP – extended instruction pointer – 32 bita. Program može da pristupi donjoj polovini čija je oznaka IP.

Četiri segmentna registra CS, DS, SS i ES ostali su po starom 16–bitni. Pojavila su se još dva nova segmentna registra FS i GS, takođe su 16–bitni.

U našem izlaganju mi ćemo izostaviti zamorne tančine. Tako da sada odustajemo od potpunog spiska svih registara.

Adresa naredbe obrazuje se kao CS:EIP ili kao CS:IP. Adresa podatka izražava se recimo kao address = segment:offset = DS:EDX ili kao DS:DX. I dalje važi formula address = 16 $\times$ segment + offset. Budući da su sada ofsetni registri ESP, EBP, ESI i EDI kao i registar EIP 32–bitni to je već drugi sabirak u formuli (sabirak "offset") dovoljan da se adresuje svih 2^{32} bajta RAM–a. To je 4 GB, četiri giga–bajta. Dakle i ukupno, 386 je predviđen da radi sa memorijom (sa RAM–om) čiji je kapacitet jednak 4 GB (ili manje).

U našem izlaganju mi ćemo izostaviti zamorne tančine. Tako sada izostavljamo priču o upravljanju memorijom. I priču o straničenju memorije (o podjeli memorije na stranice od po 4 KB).

Što se tiče 386, kažimo još o njegovim mogućim načinima rada (modovima rada). Postoje realni i zaštićeni način, isto kao kod 286. U realnom načinu, 386 radi kao jedan puno brzi 8086, proširen na 32 bita ako se želi. Znamo da u zaštićenom načinu može da se izvršava nekoliko procesa istovremeno. Neki od tih procesa mogu da rade u tzv. virtualnom 86 modu, engl. virtual 8086 mode (virtual 86 mode) ili svejedno V86 mode. Dakle, omogućeno je da se koristi softver koji je nekad ranije bio sastavljen za 8086, a da pritom računar ne napušta zaštićeni način rada (protected mode). Dakle, virtualni 86 način predstavlja jedan slučaj (podslučaj) zaštićenog načina.

Sada prelazimo na 486. Pojavio se 1989. godine. Sa stanovišta softvera, sve je praktično ostalo isto kao kod 386. Sa stanovišta hardvera, procesor 486 ima jedinicu za rad sa brojevima u pokretnom zarezu, tako da je u računaru postao suvišan poseban čip, tzv. koprocesor, koji se ranije koristio za izvođenje operacija sa brojevima prikazanim u pokretnom zarezu.

Sada prelazimo na Pentium. Pojavio se 1993. godine. Sa stanovišta softvera, sve je praktično ostalo isto kao kod 486 (iz ugla programera gledano). A koje su novine u pogledu hardvera? Ima ugrađenu keš memoriju od 16 KB: 8 KB za naredbe i 8 KB za podatke. Ima dvije aritmetičko–logičke jedinice, tako da omogućava da se dvije naredbe izvršavaju istovremeno.

Pentium ima usavršene tzv. tekuće trake, engl. pipeline. To znači da se istovremeno radi na nekoliko naredbi koje se nalaze u različitim fazama izvršavanja (po mašinskim ciklusima), da bi se dobilo na brzini rada računara.

Slično, Pentium vrši i tzv. dohvatanje unaprijed, engl. prefetch. Procesor prognozira koje će naredbe uskoro trebati. Unaprijed ih donosi iz memorije. Možda one neće ni biti potrebne (neće ni doći na red da se izvršavaju). Ono što se iz memorije donosi – smješta se u jedan bafer, smješta se u tzv. red naredbi.

Više o Pentiumu

Oktobra 1992. godine Intel je objavio da će se peta generacija njegove linije kompatibilnih procesora (čiji je interni naziv bio P5) zvati Pentium umjesto 586 ili 80586, kao što su svi očekivali. To je bila Intelova strategija da se omogući registrovanje marke i da se time zaštiti naziv spram konkurenata (najviše AMD).

Mikroprocesor se pojavio 1993. godine. Učestanost takta iznosi 60 MHz, a može da izvrši 100 miliona naredbi u sekundi. Ima ugrađenu keš memoriju od 16 KB: 8 KB za naredbe i 8 KB za podatke. Ima poboljšanu jedinicu za rad sa brojevima u pokretnom zarezu. Najvažnije, omogućava da se dvije naredbe izvršavaju istovremeno.

U martu 1994. godine Intel je plasirao na tržište drugu generaciju procesora Pentium.

Jedan matematičar je objavio preko Interneta u oktobru 1994. godine da Pentium ima bug kod dijeljenja brojeva prikazanih u pokretnom zarezu (naredba FDIV) za neke kombinacije brojeva. Naime, za određene kombinacije brojeva, računar saopštava količnik koji je pogrešan, i to relativna greška iznosi $0{,}5 \cdot 10^{-4}$. Ova greška je kasnije otklonjena.

U januaru 1997. godine pojavila se treća generacija Pentiuma. Dodato je 57 novih vrsta naredbi. Udvostručen je kapacitet keš memorije. Ima ugrađenu tehnologiju MMX (multimedia extensions), kako je Intel to nazvao.

U konstrukciji procesora odavno je poznata tzv. ideja tekuće trake ili engl. pipeline. To znači da se simultano radi na nekoliko naredbi koje se nalaze u različitim fazama izvršavanja (po mašinskim ciklusima), da bi se dobilo na brzini rada računara. Pentium ima usavršene tekuće linije, teži se da opšta performansa računara bude: jedna izvršena naredba za jedan takt, ima dvije aritmetičko–logičke jedinice za koje se koriste oznake U i V redom. Lako se zapaža da uzastopne naredbe nisu uvijek nezavisne. Recimo, prva glasi INC AX, a druga glasi INC AX takođe. Ili kada se i jedna i druga naredba obraćaju istoj lokaciji u memoriji. Slična je situacija prilikom izvršavanja naredbe za uslovni skok. U takvim situacijama, procesor uočava zavisnost i odustaje od istovremenog izvršavanja.

Slična je i tzv. ideja dohvatanja unaprijed. Kada se naredba n dohvati (sa lokacije ℓ) onda počinje njeno izvršavanje. Istovremeno sa tim izvršavanjem donosi se u procesor sadržaj lokacija $\ell + 1$, $\ell + 2$, ... (nekoliko bajta). To se smješta u tzv. instruction queue. Za ovo je zadužena posebna jedinica, prefetch unit (PU). Dobiće se na vremenu ako naredba n nije JMP, JA, CALL ili LOOP ili slično. Kod Pentiuma postoji i jedinica za predviđanje odredišta (BPU) zajedno sa svojim baferom. Ta jedinica odlučuje o tome šta se unaprijed donosi, sa ciljem da se od donošenja unaprijed ima koristi u slučaju da je n naredba za skok. Intel tvrdi da pomenuta jedinica sama za sebe povećava efikasnost izvršavanja programa za nekih 25%.

Postoje brojne verzije (varijante, podtipovi) procesora Pentium čiji su nazivi često slični. Iz godine u godinu pojavljuju se nove verzije. Npr. Itanium čija je proizvodnja počela 2001. godine. Ima 64b arhitekturu, tj. radi se o jednom 64–bitnom procesoru.

38. Temeljni pojmovi o operativnim sistemima

U ovom naslovu biće riječi o operativnim sistemima računara: definicija, struktura i jezik.

Znamo da operativni sistem predstavlja dio računarskog softvera koji je najbliži hardveru računara. Zadatak operativnog sistema je da omogući korišćenje hardvera računara. Dakle, operativni sistem upravlja svim resursima računara, kao što su: procesor, memorija, diskovi, štampači, mrežni adapteri i slično. U okviru toga, on omogućava programima da rade sa hardverom bez znanja svih detalja hardvera. Drukčije se kaže da zahvaljujući operativnom sistemu programi manje zavise od osobina konkretnog hardvera. Kao da operativni sistem pretvara konkretni hardver u zamišljenu mašinu, virtualnu mašinu.

Govoreći uprošćeno, operativni sistem upravlja hardverskim resursima računarskog sistema i nadgleda izvršavanje korisnikovih programa (aplikativnih programa). Za bližu definiciju operativnog sistema računara korisno je imati u vidu da li se govori o velikom računarskom sistemu (mainframe) ili o jedno–programskom personalnom računaru ili o mreži personalnih računara.

Prelazimo na strukturu operativnog sistema. Operativni sistem se sastoji od niza modula, gdje modul može da bude program ili rutina (dio programa). Module operativnog sistema dijelimo na rezidentne i tranzitne. Moduli koji se često upotrebljavaju prisutni su po pravilu čitavo vrijeme u unutrašnjoj memoriji. Oni čine rezidentni dio operativnog sistema. Moduli koji nisu čitavo vrijeme prisutni u unutrašnjoj memoriji (već se pamte na nekoj spoljašnjoj memoriji) čine tranzitni dio operativnog sistema. Moduli tranzitnog dijela prepisuju se po potrebi u unutrašnju memoriju, a kasnije se uklanjaju iz unutrašnje memorije kada prestanu da budu potrebni, kada se završi njihovo izvođenje (njihov rad).

Kada se računar uključi, rezidentni dio operativnog sistema treba da se upiše u unutrašnju memoriju, treba da se prepiše sa spoljašnje memorije. Za ove svrhe služi poseban modul operativnog sistema, ima ulogu inicijalne kapisle. Naziva se bootstrap loader ili punilac operativnog sistema. Drži se zapamćen u ROM–u računara.

Obično se govori i o tzv. jezgru operativnog sistema, u opštem smislu, zamišljajući da se jezgro nalazi u središtu. Jezgru pripadaju moduli za ostvarivanje elementarnih radnji.

Prelazimo na jezik operativnog sistema. Korisnik uspostavlja komunikaciju sa operativnim sistemom preko jezika operativnog sistema. Zavisno od načina saopštavanja zahtjeva računaru, operativni sistemi se mogu podijeliti u dvije grupe. Prva grupa – tekstualni način – command line mode. Korisnik treba da otkuca jednu naredbu (jednu liniju). Druga grupa – grafički način – graphical user interface, GUI. Korisnik djeluje na sličicu ili na meni, pomoću miša ili pomoću tastature. Primjer za tekstualni način je DOS. Primjer za grafički način je Windows. Sada ćemo navesti male primjere. DOS:
`copy fajl.doc a:` ili `copy c:\folder\subfolder\fajl.doc a:` kopiranje fajla
`print fajl.txt` da se odštampa
`ne` ili `ne kucam.tex` poziva se Norton Editor

Windows: ako se klikne na sličicu na kojoj piše Internet Explorer onda se time poziva taj program. Program služi za kretanje po Internetu.

Vodeći primjeri savremenih operativnih sistema su Unix, DOS i Windows 98. Takođe Linux, Windows XP i slično.

39. O OPERATIVNOM SISTEMU DOS

Nastanak i razvoj DOS–a

Operativni sistem za personalne računare sreće se pod nazivom MS DOS ili PC DOS, zavisno od proizvođača. Verziju MS DOS proizvodi softverska firma Microsoft, pa su prva dva slova u nazivu skraćenica naziva ove firme. Verziju PC DOS prodaje firma IBM koja je i proizvođač personalnih računara, pa otuda skraćenica PC. Razlike između ovih verzija su neznatne, pa ćemo dalje govoriti samo o DOS–u.

Do sada se pojavilo više od dvadeset verzija DOS–a. Naime, krajem sedamdesetih godina najpopularniji operativni sistem za mikro–računare bio je CP/M. Kada je IBM odlučio da započne sa proizvodnjom svojih mikro–računara zasnovanih na 16–bitnim mikroprocesorima, potražio je i odgovarajući operativni sistem. Mnogi su vjerovali da će to biti CP/M, međutim, IBM se odlučio za firmu Microsoft sa kojom je sklopljen ugovor o proizvodnji i ostalog pratećeg softvera. Tako se 1981. godine pojavila prva verzija DOS–a. Iza ove slijede verzije 1.1 i 1.25. Godine 1983. pojavljuje se verzija 2.0 za PC XT računare. Verzija 2.0 sadržala je niz značajnih novina, između ostalog rad sa tvrdim diskom. Iza nje slijede verzije 2.11, 2.25, 3.0, 3.1, 3.2, 3.3, itd. Svaka verzija u principu donosi neke novine i poboljšanja. Dograđivanje i usavršavanje nije samo karakteristično za DOS, već i za ostale operativne sisteme, odnosno za bilo koji značajniji softverski proizvod. Na taj način softver se prilagođava novim potrebama i tehnologijama.

Glavne komponente DOS–a

Tri glavne komponente DOS–a su tri fajla io.sys, msdos.sys i command.com. Fajl io.sys vrši radnje inicijalizacije tokom vremena pokretanja računara (boot). Pored toga, on sadrži važne ulazno–izlazne funkcije i drajvere uređaja (programe za upravljanje uređajima). Time se dopunjava primitivna ulazno–izlazna podrška sadržana u BIOS ROM čipu. Taj sistemski fajl poznat je pod imenom ibmbio.com, u slučaju verzije PC DOS. Drugi fajl msdos.sys djeluje kao DOS–ovo jezgro. On se bavi upravljanjem fajlovima, upravljanjem memorijom i ulazom–izlazom. Taj sistemski fajl poznat je pod imenon ibmdos.com, u slučaju verzije PC DOS. Treći fajl command.com predstavlja tzv. komandni interpreter ili shell operativnog sistema. On djeluje kao posrednik između korisnika i operativnog sistema. Tako on prikazuje DOS–ov prompt, nadgleda tastaturu i obrađuje korisnikove naredbe, kao što su recimo izbriši neki fajl ili loaduj neki program radi izvršavanja. Vidjećemo da sve tri komponente pripadaju rezidentnom dijelu operativnog sistema, odnosno da se tokom rada računara one nalaze upisane u unutrašnjoj memoriji. Takođe, da se tri fajla nalaze stalno zapisana na tvrdom disku i da se tokom vremena pokretanja računara tri fajla prepišu sa diska u memoriju. Koliku veličinu imaju glavni fajlovi? Pogledajmo u slučaju verzije MS DOS 3.3 iz 1987. godine:

io.sys 22KB msdos.sys 30KB command.com 25KB

Znamo da io.sys sadrži BIOS–ove rutine prekida (od n=0 do n=31), a da msdos.sys sadrži DOS–ove rutine prekida (od n=32 do n=255). BIOS je razvijala firma IBM, dok su DOS pravili stručnjaci Microsofta, zbog toga neki autori BIOS i ne ubrajaju u DOS, mada je veza između BIOS–a i DOS–a neraskidiva.

Nacrtati sliku. Unutar malog kruga piše **BIOS**. Oko njega prsten u kome piše **jezgro**. Oko njega drugi prsten, tj. još veći krug gdje piše **komandni procesor**.

Prilikom rada sa računarom koji ima DOS, ekran se nalazi u tekstualnom načinu rada i korisnik šalje svoje naredbe računaru (operativnom sistemu) preko tastature. Za ostvarivanje naredbe zadužen je command.com. On gleda da li je naredba unutrašnja (dir, erase i slično) ili spoljašnja (print, diskcomp i slično). Ako je unutrašnja onda sami command.com ostvari potrebne radnje. Ako je spoljašnja onda command.com poziva sa tvrdog diska odgovarajući program (taj program se loaduje i onda se izvodi). Spoljašnja naredba – tranzitni dio operativnog sistema. Dakle, neki program iz tranzitnog dijela

operativnog sistema po svemu je sličan nekom bilo kakvom aplikativnom (korisnikovom) programu. Lako se možemo uvjeriti da u računaru u direktorijumu koji sadrži DOS postoje fajlovi print.com i diskcomp.com (ili su .exe) (to je DOS računar). Tranzitni dio DOS–a predstavlja sljedeći četvrti prsten (četvrti sloj).

Danas postoji veliki broj tzv. pomoćnih programa ili utility programa, kao što su Norton Commander ili PC Tools koji obavljaju mnoge funkcije operativnog sistema, pružajući pritom veću udobnost korisniku. Kao da čine sljedeći prsten (peti prsten). Za takve programe možemo reći da predstavljaju nadgradnju DOS–a. To isto možemo reći i za prve verzije programa MS Windows.

Jezik DOS–a i naredbe DOS–a

Samo skicirajmo par naredbi:

CLS unutrašnja Brisanje ekrana
COPY unutrašnja recimo copy f1 f2 Kopira jednu ili više datoteka u navedeno odredište
DATE unutrašnja Prikazuje tekući datum ili se mijenja tekući datum
DIR unutrašnja recimo dir kat (ili recimo dir c:\folde) Ispisuje fajlove navedenog direktorijuma
ERASE unutrašnja recimo erase netreba.tex Briše navedenu datoteku sa diska ili diskete
REN unutrašnja recimo ren f1 f2 (ili recimo ren staroime.tex novoime.tex) Mijenja ime fajla f1 u novo ime f2 (Rename)
TYPE unutrašnja recimo type f1 Ispisuje sadržaj specifikovanog fajla f1 na ekran

DISKCOMP spoljašnja recimo diskcomp d1: d2: Upoređuje sadržaj dvije diskete
DISKCOPY spoljašnja recimo diskcopy d1: d2: Kopira se sadržaj jedne diskete na drugu
PRINT spoljašnja recimo print fil.txt Štampa sadržaj navedenog fajla

Redosljed dešavanja prilikom pokretanja sistema (boot)

Koja su najvažnija dešavanja, gledano redom po vremenskoj osi, prilikom uključivanja ("on") računara koji ima DOS? S jedne strane, prvi sektor na hard disku (to je sektor broj 1 trake broj 0) sadrži neke podatke koji će se koristiti (trebalo bi da sadrži). Za taj sektor obično se kaže da predstavlja boot sector ili boot block. S druge strane, jedan dio unutrašnje memorije računara rralizovan je kao ROM na posebnom čipu. Kaže se da je to BIOS čip, jer sadrži jedan dio BIOS–a. Veličina čipa je 8KB ili slično. Ima ulogu inicijalnog programa (inicijalne kapisle). Koje adrese pripadaju ovome? Obično su to adrese od FE00:0000 pa naprijed (do FFFF:000F, tj. do kraja 1MB RAM–a). Jednu adresu u memoriji možemo da smatramo najvažnijom adresom. To je adresa address = segment:offset = FFFF:0000. Hardverski je podešeno da se prilikom uključivanja računara odmah prvo (samo po sebi) izvrši naredba koja je plasirana na toj adresi. A koja je naredba plasirana? Naredba za bezuslovni skok na inicijalnu kapislu.

Uključivanjem računara u napon dolazi do podizanja sistema. Kada se sistem baziran na Intel 8086 uključi, on automatski počinje da izvršava kod sa fiksirane memorijske lokacije. On to uradi dodjeljujući registrima CS i IP vrijednosti FFFF i 0000 redom, što se prevodi u fizičku adresu $(FFFF0)_{16}$. Po dizajnu se lokacija FFFF0 nalazi u ROM–u i taj kod prvo prođe kroz set rutina koje provjeravaju ispravnost raznog hardvera u sistemu. To je poznato kao power–on–self–test, POST. BIOS–ova rutina koja počinje na FFFF0 zatim gleda razne portove da prepozna uređaje koji su prikopčani računaru i da ih inicijalizuje. Zatim BIOS uspostavlja podatke u dvije oblasti u memoriji: tabelu vektora prekida koja počinje u niskom dijelu memorije na adresi 0 i sadrži adrese za prekide koji nastupaju i BIOS–ovu oblast podataka koja počinje na adresi 400 a koja se najviše tiče prikopčanih uređaja. Zatim BIOS utvrđuje da li je prisutan disk koji sadrži sistemske fajlove DOS–ove. Ako jeste onda ga zanima bootstrap loader koji se nalazi na disku. Kopira sektor 1 trake 0 (but sektor) sa flopi disk drajva A ili sa hard disk drajva C, ako je drajv A prazan. Kopira ga u RAM. I skače na taj kod. Sa svoje strane, taj program loaduje sa diska u memoriju sistemske fajlove io.sys i msdos.sys i predaje upravljanje

ulaznoj tački programa io.sys koji sadrži drajvere uređaja i drugi za hardver specifičan kod. Program io.sys relocira sebe u memoriji i sa svoje strane predaje upravljanje programu msdos.sys. U slučaju da ti programi nisu prisutni onda bi se dobila poruka

NonSystem disk or disk error
Replace and strike any key when ready

Program msdos.sys vrši inicijalizaciju unutrašnjih DOS–ovih tabela i DOS–ovog dijela tabele prekida. Pored toga, on čita fajl config.sys i izvrši njegove naredbe. Najzad, msdos.sys predaje upravljanje programu command.com koji izvršava fajl autoexec.bat, prikaže svoj prompt i onda nadgleda tastaturu radi ulaza.

Da rezimiramo, koraci tokom butovanja računara su redom:

1. POST radi
2. ROM bootstrap program loaduje sektor 1 trake 0 (but sektor) u memoriju
3. U memoriju se loaduju io.sys i msdos.sys
4. DOS–ovo jezgro se premjesti na finalnu lokaciju
5. Program sysinit (to je dio programa io.sys) loaduje command.com

40. O OPERATIVNOM SISTEMU WINDOWS

By 1987, Microsoft was already at work on producing a GUI (Graphical User Interface) for MS DOS. Early forms of this were the 16–bit Windows 1, 2, and 3.x versions, which overlaid the MS DOS operating system carrying out much of the system work beneath. The various versions of Windows are all single–user multi–tasking systems: the only user can have more than one task (process) active at any moment. Later versions of Windows supported networking. Later Windows versions, such as Windows 95, 98, and Windows NT, take advantage of the more powerful features of current 32–bit Intel chips (and other 32–bit chips in the case of NT) and are complete operating systems (not relying on DOS). Nonetheless, Microsoft regards its Windows versions as a family of operating systems, and applications written for earlier versions can be run on later versions. This retains its customer base. This downwards compatibility, however, has caused Microsoft numerous development problems.

Windows provides the programmer with an accessible application programming interface (API).

41. KRATAK PREGLED RAZVOJA OPERATIVNIH SISTEMA

U ovom naslovu biće riječi o istoriji operativnih sistema računara. Umjesto operativni sistem računara ponekad se kaže sistem za eksploataciju računara. Razvoj sofyvera uopšte i razvoj operativnih sistema posebno usko je povezan sa napretkom tehnologije računara, tj. napretkom hardvera. Nove tehnologije su redovno uticale na stvaranje novih operativnih sistema.

Podjela operativnih sistema na generacije

Slično kao u razvoju računarskih sistema, možemo razlikovati nekoliko generacija u razvoju operativnih sistema. Razvoj prve generacije operativnih sistema počinje oko 1950. godine, kada su se i pojavili prvi elektronski računari u pravom smislu riječi "računar" (računari sa upisanim programom). Primjer takvog računara je Edsac (Cambridge, Engleska). Koristi se naziv *batch system* ili redna obrada za sistem rada takvih računara. To znači da su se programi izvršavali pojedinačno (jedan za drugim, redom). Kada jedan program počne da se izvršava onda on upravlja ukupno svim (hardverskim) resursima računara (računarskog sistema). Šta sadrži memorija tokom izvođenja programa? U memoriji možemo jednostavno da uočimo dva dijela. Jedan dio pripada operativnom sistemu, a drugi dio pripada tom progranu (drugi dio pripada korisnikovom = aplikativnom programu). Za prvi dio se kaže da predstavlja *monitor*. Po pravilu, taj dio je stalno prisutan u memoriji (on je

rezidentan). Iz čega se sastojao sistemski softver? Znamo da su dvije glavne komponente sistemskog softvera: operativni sistemi i programski prevodioci (govoreći savremenim rječnikom). U doba računara prve generacije nije postojala takva podjela sistemskog softvera. Pogledajmo da nabrojimo neke sistemske rutine i programe koji su postojali. a) Loader – služi da unese u memoriju naredbe i podatke korisnikovog programa. b) Asembleri – služe za prevođenje sa jezika asemblera na mašinski jezik. c) Potprogrami iz biblioteke – predstavljaju podršku za izvođenje ulazno–izlaznih radnji i radu sa spoljašnjim memorijama. d) Programi za debagiranje – da se fiksiraju–pronađu greške u korisnikovom programu, izvodeći ga interpretivno korak po korak.

Razvoj druge generacije operativnih sistema počinje oko 1960. godine. To su operativni sistemi koji podržavaju tzv. *multiprogramski* režim rada. To znači da se u memoriji računara tokom rada računara nalazi upisano nekoliko programa i da se ti programi izvršavaju istovremeno, prepleteno. U to doba pojavljuju se računarski sistemi koji imaju nekoliko terminala, gdje jednom korisniku odgovara jedan terminal. Jednu varijantu multiprogramskog režima rada predstavlja tzv. rad u podijeljenom vremenu. Uzima se da rad u podijeljenom vremenu karakteriše operativne sisteme druge generacije. Na engleskom se kaže *time–sharing*. Svaki korisnik sa svog terminala upućuje računaru jedan program (jedan proces, jedan posao, jedan zadatak) da bi ga računar izvršio. Računar prepleteno izvršava te programe. On svakom programu posvećuje po djelić vremena, on na smjenu posvećuje pažnju programima. Otuda riječ razdijeljeno vrijeme. Vidimo da se ovdje radi o više–korisničkom operativnom sistemu.

Razvoj treće generacije operativnih sistema počinje oko 1965. godine. Znamo da se 1964. godine pojavio računar 360 firme IBM (u originalu IBM System/360) i da njegova pojava predstavlja veliki napredak u razvoju računara. Predviđeno je da veliki broj korisnika istovremeno koristi računar. Za taj računar (za taj računarski sistem) prvi put se koristi naziv – veliki računarski sistem, ili na engleskom *mainframe computer*. Uzima se da je za operativne sisteme treće generacije karakteristično da su to operativni sistemi namijenjeni za mainframe računare. Postoji nekoliko operativnih sistema koji služe za IBM System/360. Najpoznatiji među njima je OS/360. Kažimo bar nekoliko riječi o operativnom sistemu OS/360. To je jedan veliki softverski proizvod. Procjenjuje se da je potrošeno hiljadu "čovjek godina" da se on sastavi. Poznat je bio po velikom broju bagova–grešaka, u svakoj svojoj verziji. To pokazuje da nije lako sastaviti veliki softverski proizvod.

Razvoj četvrte generacije operativnih sistema počinje oko 1975. godine. To su operativni sistemi koji služe prilikom rada sa kasnijim mainframe računarima. Primjer takvog operativnog sistema je VM firme IBM. VM je skraćenica od Virtual Machine, što bismo preveli kao zamišljena mašina. Kažimo bar jednu riječ o operativnim sistemima četvrte generacije. VM pruža korisniku privid da može da bira između više mašina–računara (i više operativnih sistema). Na taj način, korisnik "bira" mašinu koja mu odgovara, a VM ima zadatak da od zamišljene mašine stvori realnu.

Operativni sistem Unix

Krajem šezdesetih godina pojavili su se mini–računari, imaju daleko manje hardvera od velikih računara, a ipak mogu dobro da posluže za neke obrade. Sa njima su se pojavili i odgovarajući operativni sistemi. Među njih spada i slavni Unix. Napravljen je početkom sedamdesetih godina. Njegov tvorac je Dennis Ritchie. Računari PDP–11 firme DEC koristili su operativni sistem Unix. Podržavao je multiprogramski režim rada, a isto i rad u razdijeljenom vremenu.

Dennis Ritchie je u isto vrijeme stvorio i čuveni programski jezik – jezik C. Operativni sistem Unix sastoji se (računajući samo jezgro, ne računajući tranzitni dio) od otprilike 10.000 linija na jeziku C i još otprilike 1.000 naredbi na jeziku asemblera. Dakle, svega 5–10% je napisano na jeziku asemblera (na jeziku niskog nivoa). Drugim riječima, Unix je u najvećoj mjeri napisan na višem programskom jeziku. Ovo je jedno dobro svojstvo operativnog sistema. Zato što puno olakšava prenosivost operativnog sistema na drugu vrstu računara. Naime, samo 5–10% treba da bude ponovo napisano. Ono što je napisano na jeziku C dovoljno je da jednostavno bude prevedeno pomoću prevodioca, tako da odgovara toj drugoj vrsti računara (dovoljno je imati prevodilac odgovarajući).

Razna dobra rješenja iz operativnog sistema Unix dosta su uticala na sve operativne sisteme koji su se poslije njega pojavili.

Postoje razne varijante Unixa, namijenjene za razne klase–vrste računara.

Operativni sistemi DOS i Windows

Prvi mikro–računari pojavili su se sedamdesetih godina. Bili su zasnovani na 8–bitnim mikroprocesorima. Imali su daleko manje hardvera od ostalih vrsta računara. Prvi personalni računari pojavili su se 1977. godine (firma Apple), odnosno 1981. godine (firma IBM). Kao da je sada razvoj hardvera i softvera počeo iznova.

Operativni sistem CP/M koristio se na mikro–računarima koji su bili zasnovani na Intelovim osmobitnim mikroprocesorima, kakav je recimo Intel 8080. CP/M je skraćenica od Control Program for Microcomputers. Mali ali lijepo projektovan operativni sistem. Jedno–korisnički i jedno–programski: korisnik može da izvršava jedan program u datom vremenu, dok se program ne okonča. CP/M se pojavio 1974. godine. Početkom osamdesetih godina njegova upotreba postepeno zalazi i onda potpuno nestaje. Danas je najviše poznat kao "operativni sistem prije DOS–a".

Intelovi 16–bitni mikroprocesori 8086, 8088 i 80286 pojavili su se 1978. 1979. i 1983. godine. Prvi personalni računari pojavili su se na prelasku iz sedamdesetih u osamdesete godine. Prva verzija operativnog sistema DOS pojavila se 1981. godine, firma Microsoft. Taj operativni sistem koriste IBM–ovi personalni računari zasnovani na 16–bitnim Intelovim mikroprocesorima. Prva verzija bila je jedno–korisnička i jedno–programska. Druga verzija pojavila se 1983. godine. Podržava rad sa hard diskom. Podržava i hijerarhijsku strukturu direktorijuma (stablo direktorijuma). Pored toga, podržava i multiprogramski režim rada, mada na jedan primitivan način. Treća verzija DOS–a pojavila se 1984. godine. Kasnije se pojavilo još niz novih verzija. Svaka nova verzija donijela je neke novine. DOS je osposobljen da podržava rad mreže personalnih računara, kao i da podržava rad nekih novih uređaja.

Znamo da u razvoju personalnih računara postoje dvije glavne linije koje su konkurentske. Jednu liniju čine Intel i Microsoft. Drugu liniju čine Motorola i Apple. Njihov personalni računar naziva se Macintosh ili kraće Mac. Odgovarajući operativni sistem je Mac OS. Apple je prvi uveo 1984. godine GUI za personalne računare. GUI je skraćenica za Graphical User Interface. To znači ekran u grafičkom modu, prozori i miš.

Od 1987. godine Microsoft je već radio da bi proizveo GUI za DOS. Tako su se pojavile prve verzije Windowsa, 1, 2 i 3. One su bile 16–bitne. Predstavljale su samo dopunu ili nadgradnju DOS–a. Preuzimale su od DOS–a neke poslove, preklapale su ga. Sve te verzije bile su jedno–korisničke i više–programske: jedan korisnik mogao je da ima više od jednog posla (jednog procesa) aktivnog u bilo kom datom trenutku. Kasnije verzije Windowsa podržavaju rad mreže personalnih računara. Kasnije verzije Windowsa su Windows 95 i Windows 98. One su iskoristile to što su tekući Intelovi 32–bitni čipovi puno moćniji. To su potpuni operativni sistemi (ne oslanjaju se na DOS). Čitava ukupna linija DOS i Windows je softverski kompatibilna naniže. Težnja da kompatibilnost bude očuvana stvara teškoće prilikom projektovanja novih verzija.

Microsoft i dalje proizvodi nove verzije operativnog sistema Windows. Tako da Windows nastavlja da se razvija.

Kažimo još samo nekoliko riječi o dva operativna sistema OS/2 i Windows NT. Firme IBM i Microsoft predložile su 1987. godine operativni sistem pod nazivom OS/2 za mikroprocesor Intel 80286. Tokom devedesetih godina taj operativni sistem je evoluirao u Windows NT (Windows New Technology). Ova dva operativna sistema čine posebnu razvojnu liniju u odnosu na razvojnu liniju DOS i Windows (nema kompatibilnosti).

Operativni sistem Windows XP pojavio se na tržištu 2001. godine. Sa sobom je donio značajna poboljšanja u odnosu na svoje prethodnike. Na primjer, ima mnogo bolju podršku za dodatne uređaje. To je realizovano u vidu veće količine njihovih drajvera.

Windows Vista 2006. godine, Windows 7 2009. godine, Windows 8 2012. godine.

42. ŠTA JE TO LISTA

Prilikom sastavljanja raznih programa, koriste se strukture podataka. Kada se kaže struktura podataka onda to znači složeni podatak ili složena promjenljiva ili vrsta podataka koja nije data kao ugrađena u programskom jeziku koji se koristi. Kao suprotan primjer, vrste podataka niz i skup date su kao ugrađene u paskalu. Glavne vrste struktura podataka su lista, stek, red, graf, binarno drvo i drvo. O njima će biti riječi u nastavku, u ovom naslovu i u sljedećim naslovima.

U ovom naslovu definiše se pojam liste i daje se jedan mogući način da se lista implementira u paskalu, što je ilustrovano jednim primjerom.

Kaže se lista ili povezana lista ili spisak. Lista se definiše kao konačan niz oblika x_1, x_2, ..., x_n, gdje je $n \geq 0$. Ako je $n = 0$ onda imamo tzv. praznu listu. Za x_k se kaže da je član liste. Član liste je podatak određenog tipa. Recimo, član liste može da bude broj, može da bude podatak tipa `integer`. Po pravilu, svi članovi liste su podaci jednog te istog tipa. Budući da se radi o nizu (o konačnom nizu), to među članovima liste može da bude ponavljanja.

Tokom izvršavanja programa održava se jedna lista L. To znači da se lista mijenja tokom rada programa (iz koraka u korak konačan niz nije uvijek jedan te isti) i da se tokom rada programa vodi evidencija o listi. Drugim riječima, tokom rada programa vrše se operacije sa listom (sa članovima liste), tako da se lista mijenja kako vrijeme protiče (mijenja se stanje u listi).

Uzmimo da je tekuće (trenutno) stanje u listi koju održavamo $x_1, x_2, \ldots, x_n$. Ako se iz liste ukloni član čiji je indeks $k = 3$ onda će novi sastav liste da glasi očito $x_1, x_2, x_4, \ldots, x_n$. Ako bi se uklonio (izbacio, isključio, izbrisao) posljednji član onda bi novo stanje u listi postalo $x_1, x_2, \ldots, x_{n-1}$. Ako bi se uklonio prvi član onda bi novo stanje u listi postalo $x_2, x_3, \ldots, x_n$. Uzmimo sada da listi treba da se doda jedan objekat y. Objekat y treba da postane član liste. Kaže se da se objekat y umeće u listu i time on postaje novi član liste. Tako da se broj članova liste za jedan povećava. Bilo je n članova, a odsad će imati $n + 1$ člana. Recimo, neka y treba da bude umetnut između susjednih članova na pozicijama $k = 2$ i $k = 3$. Novo stanje u listi biće $x_1, x_2, y, x_3, \ldots, x_n$ kada se ostvari umetanje. Recimo da treba dodati y na samom kraju liste. Novo stanje u listi biće $x_1, x_2, \ldots, x_n, y$. I slično. Dakle, postoje dvije glavne operacije sa listom. To su operacija brisanja člana i operacija umetanja novog člana. Za te operacije koriste se da budu njihovi nazivi engleske riječi DELETE i INSERT redom.

U nastavku će biti dat primjer – program na paskalu. U primjeru se vodi evidencija o jednoj listi L. Drugim riječima, iz koraka u korak ažurira se stanje u listi L. To znači da se tokom rada programa na listu primjenjuje niz naredbi DELETE i INSERT. Naravno da svaka naredba treba da bude u potpunosti definisana: koji objekat se umeće, na koje mjesto se umeće, koji član se briše i slično. Dvije vrste naredbi DELETE i INSERT su pomiješane u nizu naredbi. Pretpostavlja se da je na početku rada programa lista – prazna lista. Uveden je mehanizam da se saopšti da je niz naredbi završen: kada se kaže da je, kada se otkuca da je `ch = 'p'`.

Naš program na paskalu služi samo da se izloži kako se izvode glavne operacije sa listom. Treba zamisliti veći program gdje bi između dvije uzastopne operacije sa listom bile vršene druge radnje (koje se tiču liste L ili se ne tiču liste L). Treba zamisliti da se u tom većem programu ažurira stanje u nekoliko listi, čega u našem programu takođe nema.

U našem primjeru koji slijedi uzeto je da lista služi za održavanje jednog promjenljivog skupa (za održavanje skupa čiji se sastav tokom rada programa mijenja). Zato među članovima liste ne treba da bude ponavljanja.

U našem primjeru, prilikom izvršavanja naredbe DELETE(y) razlikujemo dva slučaja. Prvi slučaj: objekat y nije prisutan u listi. Tada, u rezultatu izvršavanja naredbe DELETE, stanje u listi očito ostaje po starom, lista se ne mijenja. Drugi slučaj: u listi je prisutan objekat y. Treba ga ukloniti, lista će se promijeniti. Slično imamo dva slučaja i kod naredbe INSERT(y). Ako y–a nema u listi onda ga treba umetnuti. A ako je y već prisutan onda listu treba ostaviti po starom, budući da lista služi za evidenciju o **skupu**, tako da ne treba da bude ponavljanja (dupliranja).

Kako da se lista implementira ili ostvari ili stavi u memoriju u slučaju programskog jezika paskal. Postoji nekoliko mogućnosti, a mi ćemo razmotriti jedan način, pomoću pokazivača (poentera).

Tokom rada programa, za račun svakog novog člana liste, u memoriji se stvori jedna cjelina koja se sastoji iz dva dijela. Kaže se da se stvori cjelina ili da se generiše cjelina ili da se dinamički generiše jedan memorijski objekat ili da se u memoriji izdvoji odgovarajući prostor ili da se alocira. Dakle, jednom članu odgovara jedna cjelina čiji prvi dio sadrži sami član liste, a drugi dio je namijenjen za veznik, drugi dio sadrži pokazivač. On pokazuje na cjelinu koja odgovara sljedećem članu liste. Ako u drugom dijelu piše `nil` onda znači da smo došli do kraja liste. U slučaju INSERT(y) i y–a nema u listi uraditi NEW u programu. Time se generiše jedan memorijski objekat (jedna nova cjelina). U slučaju DELETE(y) i y je član liste uraditi DISPOSE u programu. Naredbu DISPOSE primijeniti naravno na odgovarajuću cjelinu. Prostor se dealocira. Posebnu ulogu ima promjenljiva `glava`, to je tzv. glava liste. Ona pokazuje na prvu cjelinu (na cjelinu gdje je prvi član liste upisan). Ako je `glava` = `nil` onda to znači da je lista u datom trenutku prazna. Iz ugla programera gledano, lista to je `glava`, odnosno listi se pristupa posredstvom promjenljive `glava`. Ako lista treba da se pregleda onda se polazi od glave liste i onda se redom prolazi lanac cjelina, sve dok se ne dođe do kraja liste.

U programu se redom izvršava jedna po jedna naredba DELETE ili INSERT. Operacije se učitavaju sa ulaza (kao ulazni podaci programa). Prilikom izvršavanja pojedine naredbe DELETE razlikujemo nekoliko slučajeva: lista je prazna ili y je prvi član liste ili y–a nema u listi ili y je član ali nije prvi član (tada iz lanca izostavi jednu cjelinu). Slično za INSERT: lista je prazna ili y je prisutan ili y nije prisutan (tada ga dodaj da bude zadnji član).

Zadatak. Sastaviti program na paskalu za ilustraciju rada sa listom. Ulazni podaci programa sastoje se iz nekoliko redova, gdje pojedini red ima oblik `d` x ili `i` x, gdje je x objekat, `d` – DELETE, `i` – INSERT. Predviđeno je da se na kraju rada programa štampa finalni sadržaj liste.

U nastavku je dat izdvojeno tekst rješenja, dat je program `lista.pas`.

`glava` je statička promjenljiva, a cjeline su dinamičke promjenljive. Da bi se u cjelini nešto promijenilo, treba usmjeriti recimo promjenljivu `tekuca` na željenu cjelinu i onda promjenljivoj `tekuca^` dodijeliti odgovarajuću vrijednost.

Naknadno je program dopunjen tako da prije svake operacije štampa tekući sadržaj liste. Dio programa od `write('p')` do `writeln('k')`.

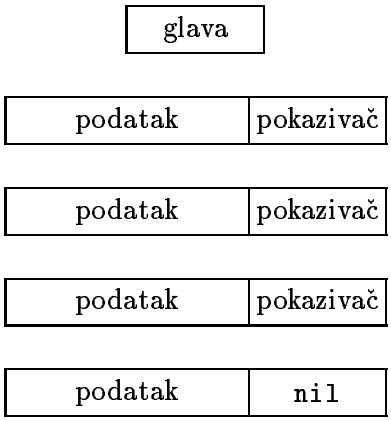
U nastavku su prikazani izdvojeno ulazni podaci koji su bili otkucani tokom rada programa, kao i rezultati koje je računar saopštio.

U nastavku je data izdvojeno jedna mala slika (od glava do `nil`), to je slika o tome kako se lista implementira.

```
    program lista(input,output);
    label 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
    const INSERT='i'; DELETE='d'; PRINT='p';
    type pokazivac=^cjelina;
    cjelina=record podatak: integer; nastavak: pokazivac end;
    var ch: char;
    broj: integer;
    glava, p, tekuca, brisanje, prethodna: pokazivac;
    begin
    glava:=nil;
 1: {DA VIDIMO STANJE U LISTI}
    write('p');
    tekuca:=glava;
 9: if tekuca=nil then goto 10;
    write(tekuca^.podatak,'x'); tekuca:=tekuca^.nastavak; goto 9;
10: writeln('k');
    {DA URADIMO JEDNU OPERACIJU SA LISTOM}
    read(ch);
    if ch=INSERT then goto 2;
```

```
   if ch=DELETE then goto 4;
   if ch=PRINT then goto 6;
 2: readln(broj);
   if glava=nil then
   BEGIN new(p); p^.podatak:=broj; p^.nastavak:=nil; glava:=p;
   goto 1 END;
   tekuca:=glava;
 3: if tekuca^.podatak=broj then goto 1;
   if tekuca^.nastavak=nil then
   BEGIN new(p); p^.podatak:=broj; p^.nastavak:=nil; tekuca^.nastavak:=p;
   goto 1 END;
   tekuca:=tekuca^.nastavak; goto 3;
 4: readln(broj);
   if glava=nil then goto 1;
   if glava^.podatak=broj then
   BEGIN brisanje:=glava; glava:=glava^.nastavak; dispose(brisanje);
   goto 1 END;
   prethodna:=glava; tekuca:=prethodna^.nastavak;
 5: if tekuca=nil then goto 1;
   if tekuca^.podatak=broj then
   BEGIN prethodna^.nastavak:=tekuca^.nastavak; dispose(tekuca);
   goto 1 END;
   prethodna:=tekuca; tekuca:=tekuca^.nastavak; goto 5;
 6: write('pocetak liste ');
   tekuca:=glava;
 7: if tekuca=nil then goto 8;
   write(tekuca^.podatak,' '); tekuca:=tekuca^.nastavak; goto 7;
 8: writeln('kraj liste')
   end.
```

```
PRVO RACUNAR ODSTAMPA pk
ONDA MI UNESEMO i 10
ONDA RACUNAR ODSTAMPA p10xk
ONDA MI UNESEMO i 20
ONDA RACUNAR ODSTAMPA p10x20xk
ONDA MI UNESEMO i 30
ONDA RACUNAR ODSTAMPA p10x20x30xk
ONDA MI UNESEMO i 30
ONDA RACUNAR ODSTAMPA p10x20x30xk
ONDA MI UNESEMO d 20
ONDA RACUNAR ODSTAMPA p10x30xk
ONDA MI UNESEMO d 10
ONDA RACUNAR ODSTAMPA p30xk
ONDA MI UNESEMO d 22
ONDA RACUNAR ODSTAMPA p30xk
ONDA MI UNESEMO i 40
ONDA RACUNAR ODSTAMPA p30x40xk
ONDA MI UNESEMO i 50
ONDA RACUNAR ODSTAMPA p30x40x50xk
ONDA MI UNESEMO p
ONDA RACUNAR ODSTAMPA DEFINITIVNO pocetak liste 30 40 50 kraj liste
```



Napomena. Program `lista.pas` može da se izvršava iz softverskog paketa za programski jezik paskal, što predstavlja prvi način. Drugi način jeste da se u okviru softverskog paketa generiše izvršni fajl `lista.exe` (i onda se izađe iz softverskog paketa), pa da se zatim izvršni fajl pozove. Izda se naredba izvrši `lista.exe`. Očito je da se poziva iz operativnog sistema. Kada se kaže operativni sistem onda se ima u vidu DOS. Bolje rečeno, ima se u vidu DOS ili Windows. Jer znamo da u Windowsu postoji `ms dos prompt` (kada je riječ o Windows 98), odnosno `command prompt` (kada je riječ o Windows XP).

Znamo da u DOS–u postoje tri vrste izvršnih fajlova i to `.com`, `.exe` i `.bat`.

Napomena. Program se može prilagoditi tako da ulazni podaci dolaze iz nekog fajla, čije je ime `listaula.dat` recimo. Taj fajl se pripremi unaprijed. A izlazni podaci (rezultati) mogu da se šalju u fajl `listaizl.dat` recimo. Tako da rezultati ostaju zapisani na disku.

Da bi se razumjelo naprijed izloženo, treba znati o pokazivačima u paskalu: pojam pokazivača, prostor u memoriji na koji on pokazuje i dva potprograma `new` i `dispose`. Izložimo ukratko i na najprostiji mogući način. Deklaracija `var p: ^integer` govori da je `p` adresna promjenljiva, da je `p` pokazivač na cio broj. Treba dodijeliti vrijednost promjenljivoj `p`. Naredba `new(p)` ima sljedeći efekat: u memoriji se zaduži prostor veličine dva bajta (za cio broj), a adresa tog prostora dodjeljuje se promjenljivoj `p` da bude njena vrijednost. Treba popuniti ta dva bajta, tj. tamo treba upisati neku vrijednost. Pomoću `p^:=50` recimo. Na kraju, naredba `dispose(p)` ima sljedeći efekat: razduži se prostor na koji `p` pokazuje, a vrijednost promjenljive `p` vraća se na `nil` = adresa koje nema (ili njena vrijednost ostaje nedefinisana). Znamo da se pokazivači na cijele brojeve rijetko koriste u praksi, a da se često koriste pokazivači na zapise (na `record`).

43. ŠTA SU TO STEK I RED

Dva važna specijalna slučaja liste su stek i red. Na engleskom se kaže stack, odnosno queue. Razlika u odnosu na listu sastoji se u tome što je ograničeno manipulisanje sa strukturom podataka (sa stekom, sa redom). Drukčije rečeno, manji je skup dozvoljenih operacija ili su manje mogućnosti pojedine operacije u odnosu na slučaj liste. Prvo govorimo o steku, a kasnije ćemo govoriti o redu. Dakle, i u jednom i u drugom slučaju radi se o konačnom nizu $x_1, x_2, \ldots, x_n$, gdje je $n \geq 0$.

Za stek je karakteristično da se sve operacije vrše na njegovom kraju, što se uzima kao neka definicija steka. Posljednji ili krajnji član steka $x_1, x_2, \ldots, x_n$ jeste x_n. Još se kaže da je x_n gornji član steka ili da se x_n nalazi na vrhu steka. Slično tome, kaže se da je x_1 donji član steka ili da se x_1 nalazi na dnu steka.

Prva operacija sa stekom jeste operacija POP – ukloniti iz steka člana koji je na vrhu steka. Kada se ova operacija izvrši onda će novo stanje u steku biti očito $x_1, x_2, \ldots, x_{n-1}$. Moguće je da je stek bio prazan (da je bilo $n = 0$) u trenutku kada je izdata naredba POP. Obično se kaže da je djelovanje naredbe POP u tim okolnostima nedefinisano. Vidimo da je bolje da se ispita da li je u datom trenutku stek prazan, prije nego što naredba POP bude izdata. Druga operacija sa stekom jeste EMPTY (prazan), što predstavlja skraćenicu od "da li je stek prazan". U rezultatu izvršavanja naredbe EMPTY dobićemo odgovor jeste prazan tj. sada je $n = 0$ ili nije prazan tj. sada je $n > 0$. Sljedeća operacija sa stekom jeste WRITE – pročitaj čemu je jednak gornji član steka. U rezultatu izvršavanja naredbe WRITE biće nam saopštena vrijednost x_n. Naredba WRITE samo odštampa x_n, ona ne uklanja x_n iz steka, sastav steka ostaje po starom. Operacija PUSH ima jedan ulazni parametar, odnosno ima jedan argument, označimo ga sa y recimo. Tako da se naredba zapisuje kao PUSH(y) ili kao PUSH y. Stavi y na vrh steka, odnosno dodaj y da postane novi član steka. Kada se naredba izvrši onda stek ima jedan član više. Dakle, novo stanje u steku biće očito $x_1, x_2, \ldots, x_n, y$.

Izložili smo matematički pojam steka, a sada je na redu dio koji se tiče programiranja. Koji su mogući načini za memorijsko predstavljanje steka. I kako se onda izvode operacije, u zavisnosti od izabranog načina. Postoje dva glavna načina. Ukratko o prvom načinu. Može se postupiti kao u slučaju liste. Dakle, pored glave steka, ima onoliko cjelina u memoriji (oblika član steka + pokazivač) koliko stek ima članova u datom trenutku. Za pojedinu naredbu PUSH y treba uraditi jednu radnju NEW, ako se programira u paskalu. A za pojedinu naredbu POP pozvati jednom potprogram DISPOSE. Međutim, implementacija steka na drugi način je puno jednostavnija i puno češće se koristi. Tako da se u nastavku govori o drugom načinu, a drugi način se koristi i u primjeru koji dolazi kasnije.

Za stek se izdvoji prostor u memoriji veličine $k \cdot m$ bajta, gdje je k – koliko bajta traži pojedini član, a m je maksimalni predviđeni broj članova u steku. U primjeru je izabrano da bude $m = 10$. Dakle, stek se predstavlja pomoću jednog niza, gdje jedan član niza služi da čuva jedan član steka. U primjeru imamo niz `a`, a članovi steka su cijeli brojevi (podaci tipa `integer`). Potrebna je još samo jedna promjenljiva u oznaci obično top (vrh), pa da memorijsko predstavljanje naše strukture podataka (steka) bude zaokruženo, završeno. Moguće vrijednosti promjenljive top su top = 0, top = 1, ..., top = m. Smisao promjenljive top je sljedeći: njena vrijednost jednaka je poziciji u nizu (u nizu `a` u našem primjeru) na kojoj se u datom trenutku nalazi gornji član steka. Na primjer, ako je top = 3 onda to očito znači da stek ima $n = 3$ člana i da je sastav steka a_1, a_2, a_3 u datom trenutku. Ako je top = 0 onda je stek prazan. Ako je top = m onda je stek pun. Broj članova steka uvijek je jednak tekućoj vrijednosti promjenljive top. Može se reći da top određuje gdje je logički kraj steka, a da je veličina prostora izdvojenog (alociranog) za stek stalno tokom rada programa jedna te ista ($k \cdot m$ bajta), odnosno da ta veličina ne zavisi od trenutnog broja članova steka. Jer je alociranje predviđeno još tokom prevođenja programa, alociranje je izvršeno statički, to znači da je ono izvršeno na početku rada programa ili bolje rečeno prije početka rada programa. A ne alocira se dinamički (tokom rada programa, tokom izvršavanja programa), jer ne koristimo pokazivače.

Do sada su navedene četiri operacije sa stekom i to POP, EMPTY, WRITE i PUSH, a sada će spisak mogućih operacija da bude kompletiran navođenjem još dvije preostale operacije FULL i inicijalizacija.

Prije dodavanja novog člana (naredba PUSH) bolje je da se pogleda da li ima mjesta da se doda. Treba pogledati da li je u datom trenutku stek pun, tj. da li je top $= m$. Operacija FULL saopštava "stek je pun" ako je trenutno top $= m$, odnosno ona saopštava da stek nije pun ako je u datom trenutku top $< m$.

Posao oko inicijalizacije steka sastoji se u tome da se na početku promjenljivoj top dodijeli vrijednost nula, na početku rada programa stek je prazan. Pored radnje top $\leftarrow 0$, u posao oko inicijalizacije ponekad se ubraja i radnja alociranja memorijskog prostora za članove steka, o čemu je bilo riječi maločas.

Zadatak. Sastaviti program na paskalu za ilustraciju rada sa stekom. Neka se tokom rada programa na jedan stek primjenjuju razne operacije EMPTY, POP, FULL, PUSH i WRITE u proizvoljnom redosljedu.

U nastavku je dat izdvojeno tekst rješenja, dat je program `stack.pas`.

U nastavku su prikazani izdvojeno ulazni podaci koji su bili otkucani tokom rada programa, kao i rezultati koje je računar saopštio.

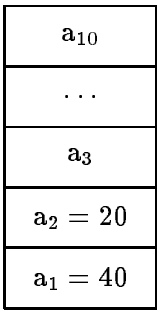
U nastavku je data izdvojeno jedna mala slika: finalno stanje steka za ulazne podatke koji su bili uneseni. Prikazane su vrijednosti $a_1, a_2, \ldots, a_{10}$ i top.

```
   program stack(input,output);
   label 1, 2, 3, 4, 5, 6, 7, 8;
   var a: array[1..10]of integer;
   top: integer;
   {a + top = struktura podataka}
   podatak: integer;
   ch: char;
   begin
   {prvo inicijalizovati stek}
   top:=0;
8: {moze write top, a[1], ..., a[top]}
       {if top>0 then
       for podatak:=1 to top do
       write(a[podatak],' ');
       writeln(top);}
   {redom jedna po jedna operacija}
   readln(ch);
   if ch='u' then goto 1;
   if ch='o' then goto 2;
   if ch='w' then goto 3;
   if ch='e' then goto 4;
   if ch='f' then goto 5;
   if ch='x' then goto 6;
1: readln(podatak);
   top:=top+1;
   a[top]:=podatak;
   writeln('uradio sam push');
   goto 7;
2: top:=top-1;
   writeln('uradio sam pop');
   goto 7;
3: writeln('na vrhu je ',a[top]);
   goto 7;
4: if top=0 then writeln('jeste prazan')
   else writeln('nije prazan');
```

```
   goto 7;
5: if top=10 then writeln('jeste pun')
   else writeln('nije pun');
   goto 7;
7: goto 8;
6: writeln('kraj programa')
   end.
```

STEK GLASI	MI ŽELIMO	MI OTKUCAMO	RAČUNAR ODŠTAMPA
	prazan?	e	jeste prazan
	pun?	f	nije pun
	push 40	u ENTER 40	uradio sam push
40	prazan?	e	nije prazan
40	pun?	f	nije pun
40	push 50	u ENTER 50	uradio sam push
40, 50	push 60	u ENTER 60	uradio sam push
40, 50, 60	pop	o	uradio sam pop
40, 50	pop	o	uradio sam pop
40	push 20	u ENTER 20	uradio sam push
40, 20	write	w	na vrhu je 20
40, 20	write	w	na vrhu je 20
40, 20	prazan?	e	nije prazan
40, 20	pun?	f	nije pun
40, 20	završeno	x	kraj programa

e – empty – prazan, f – full –pun, u – push – stavi (dodaj), o – pop – skini (oduzmi), w – write – odštampaj gornji član, x – nema više

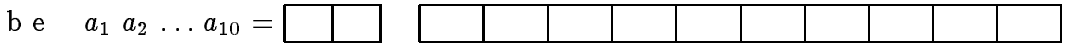


Kod savremenih računara, rad sa stekom je hardverski podržan. Kaže se da je SS:SP adresa gornjeg člana steka, u slučaju Intelovog procesora 8086. Takođe znamo da sistemski stek služi za spašavanje povratnih adresa u slučaju prekida. Što se tiče aplikativnog programera, bolje je da za svoje potrebe organizuje poseban stek.

Znamo da opšti oblik liste glasi $x_1, x_2, \ldots, x_n$, gdje je $n \geq 0$. Kaže se da se x_1 nalazi na početku liste i da se x_n nalazi na kraju liste. Već je rečeno da red predstavlja jedan poseban slučaj liste. Za red je karakteristično da se operacija unetanja člana izvodi na njegovom kraju, a operacija brisanja člana izvodi se na njegovom početku; ili obrnuto. Na primjer, ako se izda naredba `insert` y onda će poslije njenog izvršavanja novo stanje u redu da bude $x_1, x_2, \ldots, x_n, y$. Na primjer, ako se izda naredba `delete` onda će se u rezultatu izvršavanja te naredbe dobiti $x_2, \ldots, x_n$ kao novo stanje u redu. Kaže se red ili bafer.

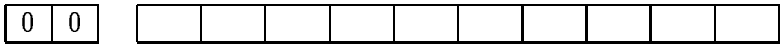
Za memorijsko predstavljanje reda obično služi jedan niz (da sadrži članove reda) zajedno sa dvije promjenljive b begin i e end. b govori koja je pozicija u nizu prvog, a e posljednjeg člana reda. b i e su promjenljive tipa indeks. U malom primjeru koji slijedi uzeto je da su moguće vrijednosti za

indeks od 1 do 10. Uzeto je da b = e = 0 znači da je red prazan. Prirodno je da predstavljanje bude dograđeno dodavanjem kružnog svojstva, da bi se izbjeglo premještanje članova reda kada se dođe do kraja u nizu. Ako je $a_{10} = x$ i $a_1 = y$ onda red glasi $(\ldots, x, y, \ldots)$ u većini slučajeva. Mali primjer koji slijedi služi i da ilustruje kružno svojstvo. Šablon oznaka je:



red glasi prazan red

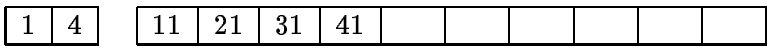
u memoriji



izvršiti naredbe `insert` 11, `insert` 21, `insert` 31 i `insert` 41

red glasi 11, 21, 31, 41

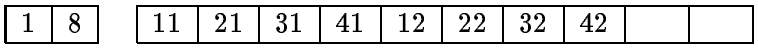
u memoriji



izvršiti naredbe `insert` 12, `insert` 22, `insert` 32 i `insert` 42

red glasi 11, 21, 31, 41, 12, 22, 32, 42

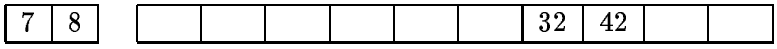
u memoriji



izvršiti naredbe `delete`, `delete`, `delete`, `delete`, `delete` i `delete`

red glasi 32, 42

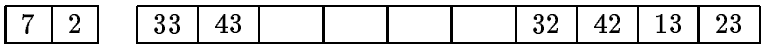
u memoriji



izvršiti naredbe `insert` 13, `insert` 23, `insert` 33 i `insert` 43

red glasi 32, 42, 13, 23, 33, 43

u memoriji



Obični primjer za upotrebu reda u sistemskom softveru jeste tzv. kružni bafer tastature čija veličina iznosi recimo 20 bajta. 20 karaktera može da se otkuca unaprijed.

44. Pojam grafa i zadatak o Ojlerovom ciklusu

Graf $G = (V, E)$ definiše se kao konačan neprazan skup vrhova V zajedno sa skupom ivica E. Ivica je dvočlani podskup skupa V. Tako da graf sa n vrhova ima najviše $\frac{n(n-1)}{2}$ ivica.

Ciklusom u grafu naziva se niz međusobno različitih nadovezanih ivica koje čine "zatvorenu liniju". Dakle, ciklus je $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}, \{v_k, v_1\}$, gdje među nabrojanim ivicama nema ponavljanja i gdje je $k \geq 3$.

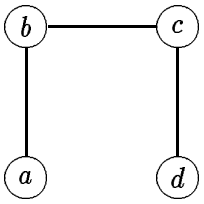
Za ciklus se kaže da je Ojlerov ako njegove ivice čine čitavi skup E. Zadatak o Ojlerovom ciklusu glasi: dat je graf, odgovoriti sa da ili ne na pitanje: postoji li Ojlerov ciklus. Drugim riječima, da li čitavi graf može da bude nacrtan bez podizanja olovke sa papira. Čitavi – svaka ivica se obuhvati. I da se ivica samo jednom povuče. S tim da crtanje počinje i završava se u jednom te istom vrhu.

Teorema. Ojlerov ciklus postoji ako i samo ako je stepen svakog vrha paran broj i još da je graf povezan.

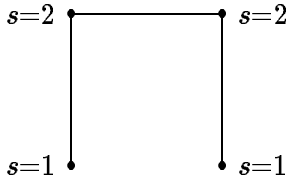
Stepen vrha pokazuje u koliko ivica učestvuje taj vrh. Graf je povezan ako postoji niz nadovezanih ivica da vodi od bilo kog vrha ka bilo kom vrhu.

Izostavlja se dokaz teoreme.

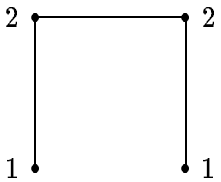
Slika 1 služi da ilustruje definiciju grafa: vrhovi su a, b, c i d, tj. skup vrhova je $V = \{a, b, c, d\}$, a ivice su $\{a, b\}$, $\{b, c\}$ i $\{c, d\}$, tako da je skup ivica $E = \{\{a, b\}, \{b, c\}, \{c, d\}\}$. Slika 2 služi da ilustruje pojam stepena pojedinog vrha (s – stepen). Naredne slike služe da ilustruju teoremu. Slika 3: Ojlerov ciklus ne postoji. Slika 4: postoji Ojlerov ciklus. Slika 5: Ojlerov ciklus ne postoji. Na slikama od 1 do 3 očito je prikazan jedan te isti graf.



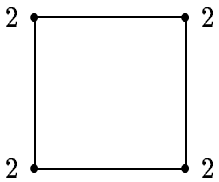
Slika 1



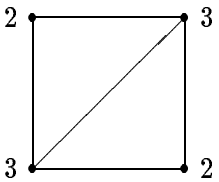
Slika 2



Slika 3 (ne postoji)



Slika 4 (postoji)



Slika 5 (ne postoji)

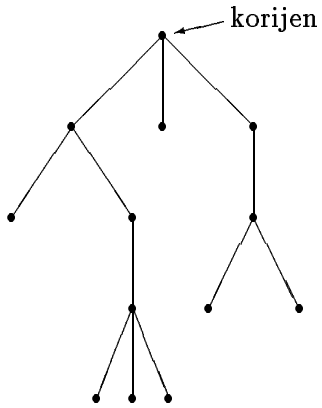
45. Pojam binarnog drveta

Slika 1: Primjer drveta: $w = w(T)$ = težina = broj vrhova, $h = h(T)$ = visina = broj nivoa mimo korijena, $w = w(T) = 13$, $h = h(T) = 4$.

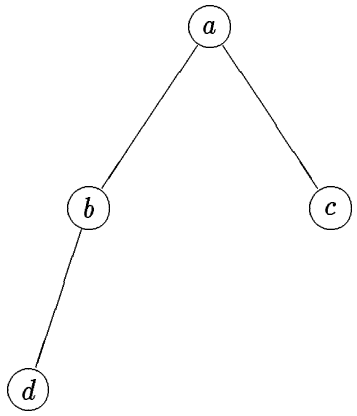
Slika 2: Primjer binarnog drveta: Vrh a je korijen drveta, vrh b je lijevi sljedbenik vrha a, vrh c je desni sljedbenik vrha a, vrh d je lijevi sljedbenik vrha b, $w = w(T) = 4$, $h = h(T) = 2$.

Za graf koji je povezan, u kome nema ciklusa i u kome je jedan vrh posebno izdvojen (naziva se korijenom drveta) kaže se da predstavlja drvo. Tako da korijen ima (može da ima) nekoliko sljedbenika, svaki sljedbenik sa svoje strane ima (može da ima) svoje sljedbenike, itd. U programima, drveta se koriste da odraze hijerarhiju koja postoji među objektima koji se razmatraju (odnos nadređenosti i podređenosti) ili odnose nadskup i podskup. Primjer za upotrebu drveta u sistemskom softveru jeste drvo svih fajlova prisutnih na disku, zajedno sa folderima i subfolderima.

Za drvo se kaže da predstavlja binarno drvo ako je za svaki vrh njegov broj sljedbenika ograničen na najviše dva i ako je svaki sljedbenik deklarisan. Upravo, za svaki vrh postoji najviše jedan sljedbenik koji je deklarisan kao lijevi i postoji najviše jedan sljedbenik koji je deklarisan kao desni, razumije se da je lijevi različit od desnog.



Slika 1 $w(T) = 13$, $h(T) = 4$



Slika 2 $w(T) = 4$, $h(T) = 2$

Razmotrimo program u kome se vodi evidencija o nekim podacima. Jedan vrh binarnog drveta čuva jedan podatak. Dakle, treba generisati binarno drvo koje ima onoliko vrhova koliko ima podataka. Treba na pogodan način rasporediti podatke po pojedinim vrhovima. Prije toga, pogodno odabrati strukturu binarnog drveta; nisu sva binarna drveta sa n vrhova međusobno izomorfna.

Binarna drveta najviše se koriste u programima u kojima se često izvodi radnja traženja podatka, engl. search. Binarna drveta koriste se u bazama podataka.

Zadatak. Sastaviti program na paskalu za ilustraciju pojma binarnog drveta. Neka se u programu generiše binarno drvo sa $n = 4$ vrha čija struktura treba da bude onako kako pokazuje sljedeća slika. Neka se vrhovima drveta pridruže podaci 100, 200, 400 i 600, opet na način kako to slika pokazuje.

Koristiti pokazivače. Neka pokazivač `glava` odgovara korijenu drveta, tj. neka glava pokazuje na cjelinu koja odgovara korijenu drveta.

U nastavku programa izvršiti jednu obradu sa drvetom: štampati sve podatke na neki sistematičan način.

Rješenje:

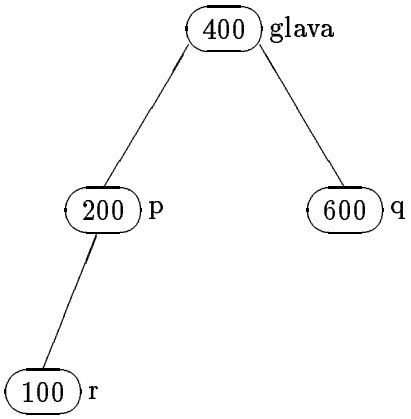
```
program binarytree(input,output);
type pokazivac=^cjelina;
cjelina=record podatak:integer;
lijevi:pokazivac; desni:pokazivac end;
var glava,p,q,r:pokazivac;
begin
new(glava); glava^.podatak:=400;
new(p); p^.podatak:=200;
new(q); q^.podatak:=600;
new(r); r^.podatak:=100;
glava^.lijevi:=p; glava^.desni:=q;
p^.lijevi:=r; p^.desni:=nil;
q^.lijevi:=nil; q^.desni:=nil;
r^.lijevi:=nil; r^.desni:=nil;
writeln(glava^.podatak);
writeln(glava^.lijevi^.podatak);
writeln(glava^.desni^.podatak);
```

```
writeln(glava^.lijevi^.lijevi^.podatak)
end.
```

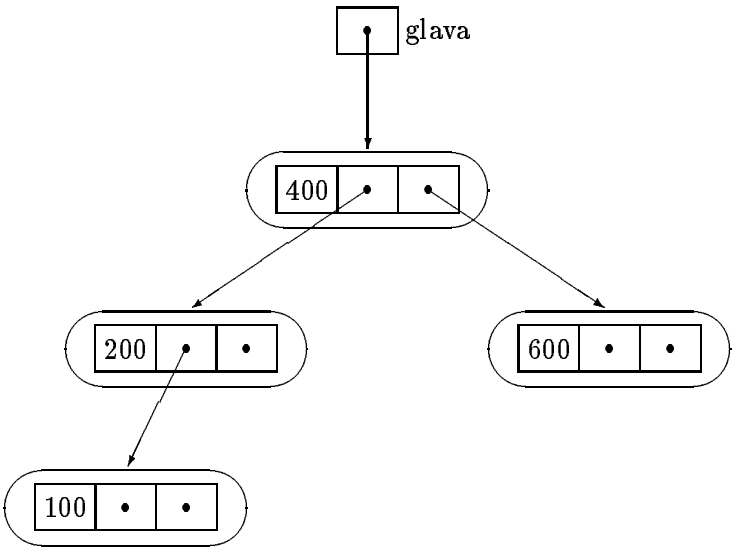
Kao rezultat izvršavanja programa, imaćemo odštampano na ekranu 400 200 600 100 i to sve jedno ispod drugog.

Program se može dopuniti tako da izvrši i neku drugu obradu sa drvetom: da izračuna težinu drveta i da izračuna visinu drveta, $w(T) = 1 + w(T_1) + w(T_2)$, $h(T) = 1 + \max\{h(T_1), h(T_2)\}$.

Binarno drvo:



U memoriji:



46. NAKNADNO DODATO: JEDNOSTAVNI PRIMJERI ZA ULAZ–IZLAZ

1. Rad sa ekranom (display). Napišimo program koji će na ekranu da prikaže u prvom redu slova A, ..., J (deset slova), u drugom redu B, ..., K (isto deset slova), itd. i u četvrtom redu D, ..., M (isto deset). U rješenju se koristi BIOS servis broj 10 i to njegova funkcija AH = E: prikaži na ekranu slovo iz AL, i to ga prikaži na teletype mode (teleprinter način). U nastavku je prikazan program, zajedno sa rezultatima koje je računar saopštio. U rješenju, u spoljašnjoj petlji BX uzima vrijednosti od BX = 0 do BX = 3. U unutrašnjoj petlji, SI uzima vrijednosti od SI = 0 do SI = 9. U pojedinoj iteraciji – štampa se jedan karakter. Na kraju unutrašnje petlje, na ekran se pošalju karakteri AL = A (line feed) i AL = D (carriage return).

```
-A 100                    loadovati na 100, 101, ...
0100 XOR BX, BX           BX ← 0
0102 XOR SI, SI           SI ← 0
0104 LEA AX, [BX+SI+41]   AX ← BX + SI + 41
0107 MOV AH, E            AH ← E (funkcija prekida)
0109 INT 10               poziva se rutina
010B INC SI               SI ← SI + 1
010C CMP SI, A            uporedi SI i A
010F JL 104               ako je SI < A (SI < 10) onda 104
0111 MOV AL, A            AL ← A = line feed
0113 INT 10               poziva se rutina
0115 MOV AL, D            AL ← D = carriage return
0117 INT 10               poziva se rutina
0119 INC BX               BX ← BX + 1
011A CMP BX, 4            uporedi BX i 4
011D JL 102               ako je BX < 4 onda 102
011F INT 20               return to DOS
0121                      sve je loadovano
-G =100                   izvršavanje, od 100
ABCDEFGHIJ
BCDEFGHIJK
CDEFGHIJKL
DEFGHIJKLM
Program terminated normally
-Q                        izlazi se iz debug.exe
```

Ekvivalentno rješenje pomoću DOS servisa: umjesto INT 21 treba INT 10 u tri naredbe, umjesto AH ← E staviti AH ← 2 i umjesto AX ← BX + SI + 41 staviti DX ← BX + SI + 41. Još DL ← A i DL ← D.

2. Rad sa tastaturom (keyboard). Program komunicira sa tastaturom preko dva porta: 60H i 64H. Prvi port služi za prenos podataka. Pomoću naredbe IN AL, 60 u akumulator se donosi karakter, donosi se scan kod tastera koji je pritisnut, AL ← port 60H. Drugi port služi za prenos podataka o posebnim tasterima. To su tasteri Alt, Ctrl, CapsLock i drugi. Pomoću naredbe IN AL, 64 učitava se trenutno stanje posebnih tastera. Za svaki takav taster, program će saznati da li je trenutno pritisnut ili otpušten. Pored toga, preko drugog porta, mogu kontrolne informacije da se šalju tastaturi, naredbom OUT 64, AL. AL definiše radnju koja će nad tastaturom biti urađena. Zašto su potrebne kontrolne radnje? Na primjer, da se podese svjetlosni indikatori na tastaturi, da se podesi perioda ponavljanja (duže vremena pritisnut taster), da se izvrše neke radnje u cilju ispitivanja ispravnosti rada tastature itd.

Šablon: (procesor) ——— (portovi 60H i 64H) ——— (interfejs tastature).

Na nožicu IRQ1 kontrolera prekida dolazi signal čim se neki taster pritisne ili otpusti. Ako se zahtjev odobri onda će biti generisan hardverski prekid broj 9.

Kako glasi kod rutine broj 9, odnosno koje radnje izvodi ta BIOS rutina za obradu prekida. Pogledajmo u osnovnim crtama. Sa IN AL, 60 rutina učitava trenutni sadržaj porta, najjači bit = 0 ako pritisak, a = 1 ako otpuštanje. Dakle, ako pritisak, učitava se karakter. Onda se iz AL prepiše u bafer tastature, tako da je karakter dospio u bafer tastature. Sa IN AL, 64 rutina učitava stanje posebnih tastera, a zatim i to stanje upisuje u memoriju. Koje adrese (od–do) pripadaju baferu tastature? Na kojim adresama se drže pokazivači "glava" i "rep"? Gdje se smješta stanje posebnih tastera? Odgovori na ova pitanja zavise od verzije BIOS–a. Na primjer, može da bude a = 0000:0417, dva bajta, za stanje posebnih tastera. Adresa glave može da bude na a = 0000:0480, dva bajta, a adresa repa na a = 0000:0482, dva bajta. U svakom slučaju, za jedan dio memorije kaže se da predstavlja tzv. područje za BIOS–ove podatke. Jedan dio tog područja služi za bafer, dva pokazivača i stanje posebnih tastera. Na kraju svog rada, rutina broj 9 uradi još jednu radnju. Upravo, ona izvrši dvije naredbe: MOV AL, 20 i OUT 20, AL. Što se tiče prve naredbe, broj 20 ima smisao (u našem kontekstu) EOI, end–of–interrupt. Što se tiče druge naredbe, port broj 20 odnosi se na kontroler prekida. Dakle, kontroleru prekida šalje se signal "EOI". Drugim riječima, kontroleru prekida pošalje se obavještenje: obrada prekida je završena.

Vidjeli smo ranije da se BIOS servis broj 16H odnosi na tastaturu. Mogli bismo da pogledamo kako glasi kod, odnosno koje radnje se izvode kada se servis pozove. Najkraće rečeno, taj sistemski potprogram, tokom svog rada, obraća se baferu tastature. Dakle, vrše se obrade koje se tiču područja za BIOS–ove podatke.

DOS rutina broj 21H, kada su u pitanju njene funkcije koje se tiču tastature, poziva rutinu 16H.

Znamo da BIOS–ovim rutinama odgovaraju brojevi prekida $n < 20$H, H znači hex. Neke od njih se direktno obraćaju hardveru. Noviji operativni sistemi ograničavaju upotrebu takvih rutina u aplikativnim programima. Zato je bolje da se, umjesto njih, koriste DOS–ove rutine. Znamo da njima odgovara $n \geq 20$H.

3. Rad sa ugrađenim zvučnikom (speaker). Brojna vrijednost sadržana u portu 42H determiniše visinu tona. Treba u taj port upisati brojnu vrijednost između 0 i FF, manji broj – viši ton. Port 61H je kontrolni port (kontrolni port sistema), ima osam bita. Dva donja bita odnose se na ugrađeni zvučnik, a ostali bitovi odnose se na neke druge uređaje. Pomoću ovog porta uključuje se odnosno isključuje se ugrađeni zvučnik. Upravo, ako su dva donja bita = 1 onda je uključen, a ako su = 0 onda je isključen. Rješenje je prikazano u nastavku. Kroz CX prolaze vrijednosti C, B, ..., 2, 1 a za visinu tona uzima se $8 \cdot$ CX. Prisutna je još i dvostruka petlja, da bi jedan ton trajao bar par sekundi. Tokom rada programa, mijenja se ton, ukupno će se čuti 12 različitih zvukova.

```
-A 100              loadovati na 100, 101, ...
0100 IN AL, 61      AL ← port 61
0102 OR AL, 3       AL ← AL OR 00000011
0104 OUT 61, AL     port 61 ← AL
0106 MOV CX, C      CX ← C (brojač ← C)
0109 PUSH CX        stavi CX na stek (stavi brojač na stek)
010A SHL CX, 1      shift left CX 1 place
010C SHL CX, 1      shift left CX 1 place
010E SHL CX, 1      shift left CX 1 place
0110 MOV AL, CL     AL ← CL
0112 OUT 42, AL     port 42 ← AL (za visinu tona)
0114 MOV CX, FFF    CX ← FFF
0117 PUSH CX        stavi CX na stek
0118 MOV CX, FFFF   CX ← FFFF
011B NOP            no operation
```

```
011C LOOP 11B      CX ← CX – 1, if CX ≠ 0 then 11B
011E POP CX        uzmi sa steka i stavi u CX
011F LOOP 117      CX ← CX – 1, if CX ≠ 0 then 117
0121 POP CX        uzmi sa steka i stavi u CX (brojač)
0122 LOOP 109      CX ← CX – 1, if CX ≠ 0 then 109
0124 IN AL, 61     AL ← port 61
0126 AND AL, FC    AL ← AL AND 11111100
0128 OUT 61, AL    port 61 ← AL
012A INT 20        return to DOS
012C               sve je loadovano
-G =100            izvršavanje, od 100
Program terminated normally
-Q                 izlazi se iz debug.exe
```

U nastavku je prikazano i rješenje na neformalnom jeziku: 100–104 pročitaj port 61H, setuj dva donja bita, vrati modifikovanu vrijednost, 109–112 izračunaj novu visinu tona i pošalji na port 42H, 114–11F dvostruka petlja za čekanje (za pauzu), 121–122 pređi na računanje nove visine tona ili izađi iz petlje, 124–128 pročitaj port 61H, izbriši dva donja bita u AL i vrati modifikovanu vrijednost (isključi speaker).

Program se može modifikovati, da se produži trajanje pojedinog tona, time što će se povećati broj taktova u pauzi. Npr. da na adresi 114 umjesto CX ← FFF piše CX ← 3FFF, čime se povećava četiri puta. Ili da bude CX ← FFFF (puta 16).

4. Rad sa matričnim štampačem (printer), u osnovnim crtama. Matrični štampač priključen je na paralelni port LPT1 računarskog sistema. On ispostavlja zahtjeve za prekid na nožicu IRQ7 kontrolera prekida. Time on saopštava da je pripravan. Ako se zahtjev odobri onda će početi da se izvršava rutina za obradu prekida broj F.

BIOS servis broj 17H služi za komunikaciju sa štampačem. Pogledajmo tri njegove funkcije. Ako je AH = 0 onda se štampaču šalje jedan karakter, funkcija štampanja karaktera. Karakter o kome se radi treba da bude pripremljen u akumulatoru AL. Funkcija AH = 1: da se štampač inicijalizuje (resetuje). Funkcija AH = 2: get printer status, saznavanje statusa štampača. Rezultat će biti upisan u registar AH. U rezultatu (odgovoru) sadržane su sljedeće informacije: da li je štampač isključen ili uključen, ima li papira, da li je štampač pripravan ili zauzet, da li je možda došlo do ulazno–izlazne greške generalnog tipa i slično.

Program debug.exe predstavlja jedan proizvod firme Microsoft. Pored njega, za vježbu asemblera, može se koristiti i program tasm firme Borland. Takođe se preporučuje i program, tj. programski paket masm firme Microsoft.

U slučaju operativnog sistema Windows 8 (na 64 bita procesoru), potreban je poseban postupak ako želimo da koristimo program debug.exe.

Principi programiranja
Sadržaj predavanja:

1 Pojam registarskog prenosa i pojam mikro–operacije prinb.tex
2 Arhitektura osnovnog računara prinb.tex
3 Oblik naredbe u mašinskom jeziku osnovnog računara prinb.tex
4 Kontrolna jedinica osnovnog računara i vremenski ciklusi prinb.tex
5 Iz kojih mikro–operacija se sastoji pojedini vremenski ciklus princ.tex
6 Naredbe koje se odnose na memoriju (osnovni računar) princ.tex
7 Naredbe koje se odnose na registre (osnovni računar) princ.tex
8 Ulazno–izlazne naredbe i mogućnost prekida (osnovni računar) prinv.tex 8 Primjeri za ulaz i izlaz (osnovni računar), programski upravljano prenošenje prinv.tex 8 Primjeri za ulaz i izlaz (osnovni računar), prekidom vođeno prenošenje prinv.tex 8 Novi izlazni uređaj: linijski štampač (osnovni računar) prinv.tex
9 Završni pogled na osnovni računar, o njegovoj izradi prinv.tex
10 Primjeri aplikativnih programa za osnovni računar (aritmetički program i rad sa potprogramom) prine.tex
11 Primjeri sistemskih programa za osnovni računar (loaderi) prine.tex
12 Jezik asemblera za osnovni računar prine.tex
13 Računarski softver (uopšte) (tri nivoa programa, podjela softvera) prine.tex
14 Organizacija centralnog procesora (uopšte) prinf.tex
15 Organizacija ulazno–izlazne jedinice (uopšte) pring.tex
16 Organizacija unutrašnje i spoljašnje memorije (uopšte) pring.tex
17 Pregled razvoja mikroprocesora pring.tex
18 Primjer mikroprocesora: i8080 prinh.tex
19 8080 – naredbe prinh.tex
20 8080 – pinovi prinh.tex
21 Primjer mikroprocesora: i8086 prinh.tex
22 i8086 – arhitektura prinu.tex
23 8086 – flagovi prinu.tex
24 Jednostavni primjer programa na jeziku asemblera, aritmetički program, primjera.asm prinu.tex
25 Jednostavni primjer programa na jeziku asemblera, rad sa stekom, primjerb.asm prinu.tex
26 Jednostavni primjer programa na jeziku asemblera, učitavanje pojedinačnih slova i štampanje niza slova, primjerc.asm prinu.tex
27 Jednostavni primjer programa na jeziku asemblera, rad sa nizom, primjerd.asm prinu.tex
28 Upotreba programa debug.exe prini.tex
29 8086 – naredbe, naredbe za prenos podataka prinj.tex
30 8086 – naredbe, aritmetičke naredbe prinj.tex
31 8086 – naredbe, naredbe za prenošenje upravljanja prinj.tex
32 Sistem prekida procesora Intel 8086 prink.tex
33 Neke funkcije sistemskog prekida 21H prink.tex
34 Pregled BIOS–ovih i DOS–ovih prekida prink.tex
35 Jedan konkretan primjer rutine za obradu prekida prink.tex
36 O mikroprocesoru Intel 80286 prinl.tex
37. Tridesetdvobitni mikroprocesori prinl.tex
38 Temeljni pojmovi o operativnim sistemima prinm.tex
39 O operativnom sistemu DOS prinm.tex
40 O operativnom sistemu Windows prinm.tex
41 Kratak pregled razvoja operativnih sistema prinm.tex
42 Šta je to lista prinp.tex
43 Šta su to stek i red prinq.tex
44 Pojam grafa i zadatak o Ojlerovom ciklusu prinr.tex
45 Pojam binarnog drveta prinr.tex
46 Naknadno dodato: jednostavni primjeri za ulaz–izlaz prinr.tex

Fajlovi (gsview): prinb.tex 7 pages princ.tex 5 pages prinv.tex 9 pages prine.tex 7 pages prinf.tex 2 pages pring.tex 4 pages prinh.tex 4 pages prinu.tex 5 pages prini.tex 3 pages prinj.tex 6 pages prink.tex 5 pages prinl.tex 3 pages prinm.tex 6 pages prinp.tex 4 pages prinq.tex 4 pages prinr.tex 6 pages. $\sum = 16$ files & 80 pages