7 Centralna procesorska jedinica

U Glavi 6 opisana je struktura mašinskih instrukcija. Za izvršavanje instrukcija, potrebno je da postoji u računaru poseban dio (jedinica), koji uzima instrukcije iz memorije, prepoznaje ih i obavlja akcije u skaldu sa značenjem instrukcije. Takva jedinica naziva se centralnom procesorskom jedinicom (CPU).

Kao što je već rečeno, rješavanje nekog problema uz pomoć računara vrši se definisanjem algoritma. Algoritam se satoji od niza koraka koji se nazivaju mašinskim instrukcijama. CPU vrši svoj posao na sličan način. Da bi uzeo iz memorije i izvrčio instrukciju CPU mora, za svaku instrukciju, da obavi niz elementarnih operacija. Te operacije i način na koji se izvršavaju od strane CPU biće opisane u ovoj glavi.

7.1 Komponente CPU-a

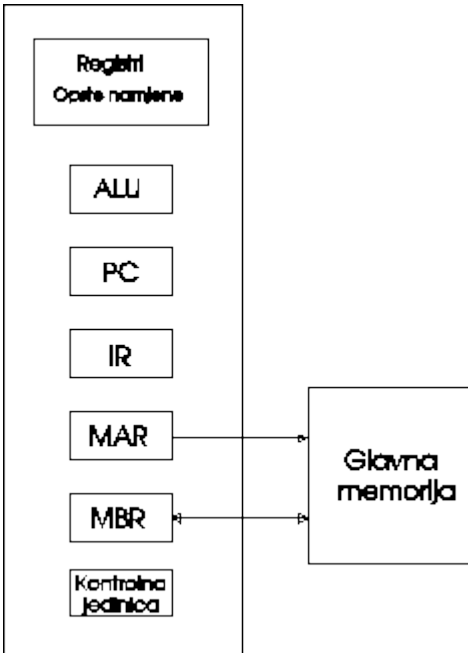
CPU možemo posmatrati kao skup komponenti sa različitim funkcijama, kako je ilustrovano na Slici 7.1.

Na Slici 7.1 se mogu uočiti tri osnovne komponente CPU-a:

Specijalni registri PC, IR, MAR, MBR, kao i skup registrara opšte namjene.

Aritmneticko-logička jedinica ALU.

Kontrolna jedinica.



Slika 7.1 Struktura CPU-a

Specijalni registri, kao što će se vidjeti, služe za prihvatanje i analizu instrukcija. Registri opšte namjene stoje na raspolaganju programeru da u njih dovodi potrebne podatke. Aritmetičko-logičku jedinicu možemo posmatrati kao "crnu kutiju" u kojoj se zapravo obavljaju računske i logičke operacije. Kontrolna jedinica je "nervni centar" računara koja salje upravljačke signale svim ostalim jedinicama.

7.1.1 Interni registri

Svaki program je sačinjen od niza instrukcija smještenih u glavnoj memoriji računara. Prilikom izvršavanja programa CPU uzima jednu po jednu instrukciju iz memorije, i priprema akcije (operacije) potrebne da se instrukcija obavi. Instrukcije se uzimaju iz memorije redom iz uzastopnih lokacija, osim u slučaju instrukcija grananja.

Kao sto je već rečeno u Glavi 6, sve komunikacije CPU-a i memorije odvijaju se pomoću dva registra: memorijskog adresnog registra (MAR) i memorijskog baferskog registra (MBR). Da bi uzeo instrukciju iz memorije CPU najprije smješta njenu adresu u MAR i izvršava operaciju čitanja memorije čiji rezultat je "kopiranje" sadržaja memorijske lokacije u MBR registar. Izvršavanje instrukcije, koja je sada u MBR-u, može zahtijevati podatak iz memorije, čija adresa mora biti smještena u MAR što bi uništilo adresu instrukcije koja se izvršava. CPU bi tako

izgubio orjentaciju i ne bi mogao da zna adresu sljedeće instrukcije. Kao posljedica ovog problema, CPU mora imati poseban registar za čuvanje adrese sljedeće instrukcije. Taj registar je poznat kao PC (program counter - programski brojač). Na početku izvršavanja programa PC se napuni sa adresom prve instrukcije programa, a zatim nakon uzimanja svake instrukcije iz memorije adresa smještena u PC-u se automatski uvećava da pokazuje adresu sljedece instrukcije (sem u slučaju instrukcije grananja kada se adresa u PC-u modifikuje na drugi način).

Nakon uzimanja instrukcije iz memorije ona će se naći u MBR registru, kako je već rečeno. Međutim, MBR registar može biti zahtijevan za preuzimanje podatka iz memorije koji je potreban za izvršavanje instrukcije. Znači, pojavljuje se sličan problem kao i sa MAR registrom. Zato je potrebno da u CPU-u postoji poseban registar koji će držati instrukciju za vrijeme njenog izvršavanja. Takav registar naziva se instrukcionim registrom (IR).

Sada možemo čitav proces izvršavanja programa prikazati kao sljedeći niz događaja:

(1) Kopiraj PC sadrzaj u MAR

(2) Učitaj u MBR memorijsku lokaciju sa adresom u MAR

(3) Kopiraj sadrzaj MBR-a u IR.

(4) Dekodiraj sadrzaj IR (prepoznaj instrukciju)

(5) Izvrši instrukciju

(6) Ponovi sve od koraka (1)

Ovaj proces se moze prikazati simbolickom notacijom kako slijedi:

[PC] ---> MAR

[M] ---> MBR

[PC]+1 ---> PC

[MBR] ---> IR

Dekodiraj IR

Izvrši instrukciju

Uglaste zagrade označavaju sadržaj memorije (M) ili registra.

Operacija "Dekodiraj instrukciju" se obavlja pomoću dekodera, koji može biti izveden kao kombinatorno logičko kolo.

Operacija "Izvrsi instrukciju" takođe se može sastojati od niza koraka, što će biti detaljnije analizirano u poglavlju 7.2.

7.1.2 Aritmetičko-logička jedinica

Aritmetičko-logička jedinica (ALU) je dio CPU u kojem se izvršavaju sve aritmetičke i logičke operacije. Gradi se od vrlo brzih elektronskih komponenti i predstavlja složeno "parče" elektronike koje je sposobno da sa podacima koje joj se daju izvrši elementarne aritmetičke i logičke operacije. Tipične operacije koje se obavljaju u ALU-u su:

Aritmetičke operacije kao što su ADD (dodavanje) i COMPLEMENT (komplement binarnog broja).

Logičke operacije kao što su AND, OR, EXCLUSIVE OR.

Manipulacione operacije kao što su SHIFT, TEST.

Skup aritmetičko logičkih operacija koje mogu da se obavljaju u ALU varira od računara do računara. ALU, najčešće, aritmetičke operacije obavlja samo sa cjelobrojnim vrijednostima. Pa i tada, vrlo ograničen skup operacija se implementira, a složenije operacije se najčešće izvode programskim putem. Tako, recimo, operacije množenja i dijeljenja, najčesce, nijesu realizovane hardverski u ALU-u, već se izvode softverski korišćenjem sabiranja i oduzimanja ili šift operacijama.

ALU sadrži i niz tzv. test bitova (flip-flop signala) koji signaliziraju rezultat aritmetičkih i logičkih

oparacija. Ovi bitovi (flegovi - zastavice) se najčešće označavaju sa N, Z i O. Tako fleg N (Negative) kada je postavljen na 1, označava da je rezultat aritmetičke operacije koja je upravo izvršena u ALU negativan broj, Z (Zero) označava da je rezultat 0, a O oznacava pojavu rezultata veceg od maksimalnog broja sa kojim računar može da radi (Overflow - prekoračenje).

7.1.3 Floating point jedinica

Kako je već rečeno, ALU najčešće obavlja aritmetičke operacije sa cijelim brojevima. Međutim brojne aplikacije zahtijevaju i rad sa decimalnim brojevima (floating point - pokretni zarez). Naravno, sve operacije sa pokretnim zarezom mogu biti softverski simulirane, ali se pri tome značajno povećava vrijeme izvršavanja takvih operacija. Zato se, često i za ove operacije gradi posebna elektronska jedinica poznata kao FPU (Floating point unit).

7.2 Izvršavanje instrukcija

Kao što smo vidjeli u 7.1.1, program smješten u glavnoj memoriji računara, izvršava se u sljedećim koracima:

(1) [PC] ---> MAR

(2) [M] ---> MBR

(3) [PC]+1 ---> PC

(4) [MBR] ---> IR

(5) Dekodiraj IR

(6) Izvrsi instrukciju

(7) Ponovi od koraka (1)

Koraci (1) do (5) se nazivaju fazom uzimanja (FETCH) instrukcije, a korak (6) fazom izvršavanja (EXECute). Svaki od ovih koraka se naziva mikro-instrukcijom.

Jasno je da se FETCH faza izvršava uvijek na isti način, bez obzira o kojoj je mašinskoj instrukciji riječ. Međutim faza izvršavanja biće različita za različite mašinske instrukcije.

Sada će biti pokazana faza izvršavanja u skladu sa tipovima instrukcija.

7.2.1 Izvršavanje ne-memorijskih instrukcija

Ne memorijske instrukcije su, kako je rečeno u Glavi 6, one instrukcije za čije izvršavanje nijesu potrebni podaci koji se nalaze u glavnoj memoriji. To su najčešće instrukcije koje se izvršavaju sa podacima koji su već u CPU registrima. Za izvršavanje takvih instrukcija potrebno je samo ALU-u saopštiti koji bitovi i iz kojeg registra će biti korišćeni i signalom iz dekodera aktivirati potrebnu operaciju ALU-a. Kada ALU završi operaciju rezultat rada biće ponovo u nekom od registara.

7.2.2 Izvršavanje memorijski zavisnih instrukcija

Ovdje je proces izvršavanja instrukcija nešto složeniji, jer prije izvršavanja operacije mora se iz memorije u CPU dovesti i podatak. Faza izvršvanja instrukcija koje uzimaju podatke iz memorije i nad njima vrše aritmetiču operaciju može biti prikazana na sljedeć način:

$[IR]_{\text{adresno polje}}$ ---> MAR

[M] ---> MBR

Izvrsi aritmeticko-logicku operaciju koriscenjem ALU i MBR.

Ako je instrukcija takvog tipa da piše u memoriju, kao npr. operacija "Smjesti sadržaj registra A u adresiranu memorijsku lokaciju", sekvenca mikro-instrukcija može biti sljedeća:

$[IR]_{\text{adresno polje}}$ ---> MAR

[A] ---> MBR

[MBR] ---> M

Neke instrukcije i čitaju i pišu u memoriju. Posmatrajmo npr. instrukciju "Povećaj sadržaj zadate memorijske lokacije za 1 i preskoči sljedeću instrukciju ako je rezultat 0". Takva instrukcija se

izvršava na sljedeći način:

(1) $[IR]_{\text{adresno polje}}$ **---> MAR**

(2) **[M] ---> MBR**

(3) **[MBR]+1 ---> MBR**

(4) **[MBR] ---> M**

(5) **Ako je [MBR] = 0, onda [PC]+1 ---> PC**

Nakon što je učitan podatak iz memorije u koraku (2), korakom (3) se povećava njegova vrijednost za 1, a korakom (4) upisuje nazad u istu memorijsku lokaciju iz koje je uzet (jer se MAR nije promijenio). Instrukcijom (5) se testira rezultat operacije i u slučaju da je jednak 0, PC se povećava za 1. Time se preskače sljedeća instrukcija u memoriji, jer je PC već povećan za 1 za vrijeme FETCH faze instrukcije.

7.2.3 Instrukcije grananja

Instrukcije grananja su poseban podskup instrukcija koje se odnose na memoriju i služe za određivanje adrese sljedeće instrukcije koja će se izvršavati (mijenjaju redosled izvršćavanja).

Posmatrajmo tzv. instrukciju bezuslovnog skoka. Ovom instrukcijom se definiše, u adresnom polju, adresa sljedeće instrukcije koju treba izvršiti. Za vrijeme FETCH faze instrukcije bezuslovnog skoka PC je povećan za jedan i pokazuje na adresu instrukcije koja se u memoriji nalazi odmah iza instrukcije bezuslovnog skoka. Međutim, sama instrukcija skoka zahtijeva izvrašavanje sljedeće instrukcije, ne kako je definisano u PC, već kako nalaže adresno polje instrukcije skoka. Zato se tok izvršavanja instrukcije bezuslovnog skoka može prikazati kako slijedi:

$[IR]_{\text{adresno polje}}$ **---> PC**

Ako je grananje uslovno, prije izvršavanja skoka provjerava se neki zadati uslov, kao što je prikazano i poglavlju 7.2.2.

7.2.4 Indirektno i indeksno adresiranje

U primjerima prikazanim do sada, smatrali smo da je adresiranje bilo direktno, to jest da je adresno polje instrukcije pokazivalo stvarnu (efektivnu) adresu glavne memorije.

Ako se u instrukciji koristi indirektno adresiranje tada je za izvršavanje instrukcije potrebno još jedno dodatno čitanje memorije.

Kao primjer posmatrajmo instrukciju kojom se indirektnim adresiranjem uzima podatak iz memorije i nad njim izvršava neka aritmetička operacija. Niz koraka je tada sljedeći:

$[IR]_{\text{adresno polje}}$---> MAR

[M] ---> MBR

[MBR] ---> MAR

[M] ---> MBR

Izvrsi aritmeticku oparaciju koriscenjem ALU i MBR.

Kod indeksnog adresiranja, mora postojati i korak u kome se na adresno polje dodaje vrijednost index registra da se dobije efektivna adresa u memoriji. Za primjer uzmimo istu instrukciju kao malo prije. Sada će niz koraka biti:

$[IR]_{\text{adresno polje}}$ ---> MAR

[MAR]+[indeksni registar] ---> MAR

[M] ---> MBR

Izvrsi aritmeticku oparaciju koriscenjem ALU i MBR.

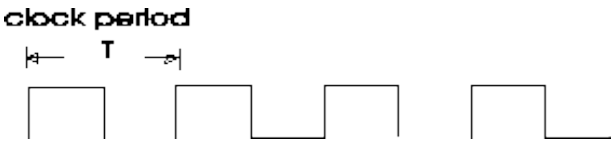
7.3 Arhitektura CPU-a

U ovoj glavi smo, do sada, razmatrali osobine i funkcije koje CPU većine računara mora da sadrži. Sa tačke gledišta korisnika, ono što razlikuje računare jednosg od drugog jeste zapravo samo skup mašinskih instrukcija. Sa aspekta performansi CPU potrebno je pored skupa instrukcija, pogledati i kako su bazične komponente (registri, ALU, kontrolna jedinica) struktuirane i međusobno povezane. Od toga će zavisiti efektivna snaga i brzina rada CPU-a. Unutrašnja struktura i međusobna veza komponenti CPU-a se nazivaju i arhitekturom CPU. Riječ arhitektura se koristi upravo zato da ukaže na sličnost sa uobičajenim njenim značenjem. Kao što se od opeke, cementa, stakla, drveta, gradi kuća,. tako se i od bazičnih elemenata (memorije, registara, ALU, itd.) gradi računar.

7.3.1 Sinhronizacija rada CPU-a

Ukupno vrijeme potrebno za izvršavanje neke instrukcije zavisi od složenosti same instrukcije. Trajanje FETCH faze je isto za sve instrukcije, ali faza izvršavanja (kao što smo već vidjeli) može zahtijevati jedan ili više koraka. Takođe i vrijeme za izvršavanje pojedinačnih koraka u FETCH/EXEC fazama može biti različito. U zavisnosti od toga kako se rješava problem usklađivanja vremena izvršavanja elementarnih koraka razlikujemo dvije vrste procesora: sinhrone i asinhrone.

Kod sinhronih procesora postoji tzv. satni mehanizam (clock). To je elektronsko kolo koje generiše impusle u pravilnim i tačnim vremenskim intervalima kao što je prikazano na Slici 7.2.



Slika 7.2 Sinhronizujući klok signal

Svaki korak pri izvršavanju instrukcije mora započeti sa pojavom novog clock signala. Neki

koraci zahtijevaju kraće a neki duže vrijeme, tako da će za neke korake biti dovoljan jedan period (T) clock signala dok će za druge biti potrebno više takvih perioda. Na taj način se postiže da se ne može započeti sljedeći korak dok predhodni nije završen. Ovim mehanizmom se na jednostavan način rješava problem sinhronizacije rada, a glavni mu je nedostatak što se uvijek koristi fiksno vrijeme koje nije potrebno za sve instrukcije.

Drugo rješenje, sa asinhornim radom procesora, kod kojeg se početak koraka pri izvršavanju instrukcije vezuje za kraj izvršavanja predhodnog koraka ima značajnu uštedu u vremenu. Međutim, sada mora postojati posebna elektronika koja će kontrolisati trenutak završetka koraka, što usložnjava arhitekturu CPU-a.

7.3.2 Povezivanje komponenti CPU-a

Kao sto je već rečeno, CPU se sastoji od niza komponenti. Komponente mogu biti međusobno povezane na više načina, a način njihovog povezivanja značajno utiče na ukupnu brzinu rada CPU-a. Zato je način povezivanje komponenti CPU-a važna arhitektonska karakteristika računara.

Spojni putevi medju komponentama CPU mogu biti ostvareni na tri načina:

point-to-point (od-tačke-do-tačke - svako sa svakim)

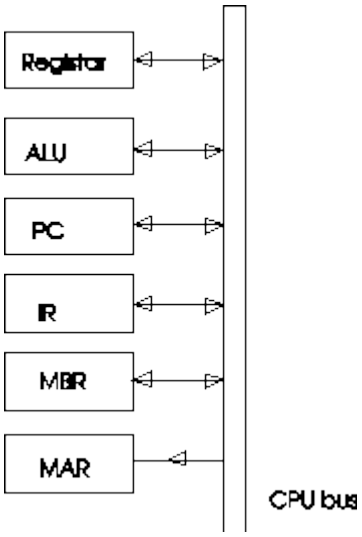
zajedničkom magistralom (bus),

sa više magistrala (multibus).

S druge strane, da bi transfer podataka među komponentama CPU-a bio efikasan potrebno je da se odvija u paralelnom modu (svi bitovi istovremeno) što znači da svaka putanja mora imati više žica (linija). Skup paralelnih žica koje povezuju dvije ili više komponenti naziva se magistralom (bus-om).

U slučaju point-to-point magistrala, između svake dvije komponete koje treba da razmjenjuju podatke postoji po jedna posebna magistrala. Ovakav način, naravno čini transfer podataka veoma efikasnim, ali je složenost veoma velika i skupa. Zato se ovakva mogućnost skoro nikada ne koristi kod standardnih arhitektura.

Kao drugi ekstremni slučaj posmatraćemo postojanje samo jedne magistrale, kao što je prikazano na Slici 7.3.

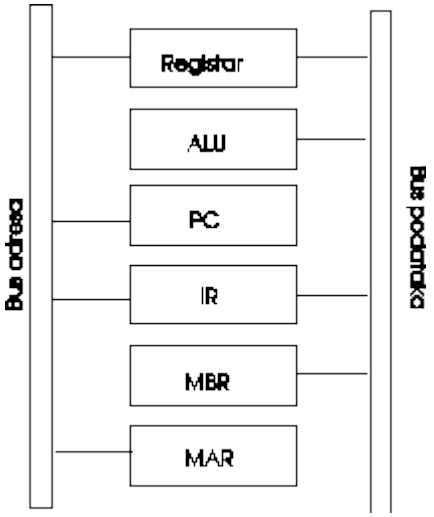


Slika 7.3 Interna CPU magistrala

Kako se sav prenos podataka odvija preko zajedničke magistrale, to se u jednom trenutku moze vršiti samo jedan transfer podataka izmedju dvije komponente CPU-a. Takođe, u ovakvoj arhitekturi, mora postojati i posebna elektronika koja će upravljati koji od djelova mogu slati/primati podatak u nekom trenutku. Ovakvim rješenjem se postiže jednostavnost izgradnje veza među komponentama, ali je značajno redukovana efiksanost prenosa podataka (u odnosu na point-to-point).

Najčesće korišćen način za internu komunikaciju među komponentama CPU je kompromisno rješenje između gore pomenuta dva načina. To je multibus način prikazan na Slici 7.4

Multibus sistemom se omogućava više od jednog transfera istovremeno, ali po različitim bus-ovima, a složenost elektronike je prihvatljiva.



Slika 7.4 Interni multibus

7.4 Kontrolna jedinica

Za izvršavanje instrukcija CPU mora posjedovati neko sredstvo za generisanje odgovarajućih kontrolnih signala za svaki elmentarni korak. Postoje dva načina koja se koriste u ovu svrhu:

- hardverska kontrola
- mikroprogramska kontrola

7.4.1 Hardverska kontrola

Hardverska kontrola se ostvaruje posebnim elektronskim sklopom i poznata je kao "hardwired" (ožičavanje).

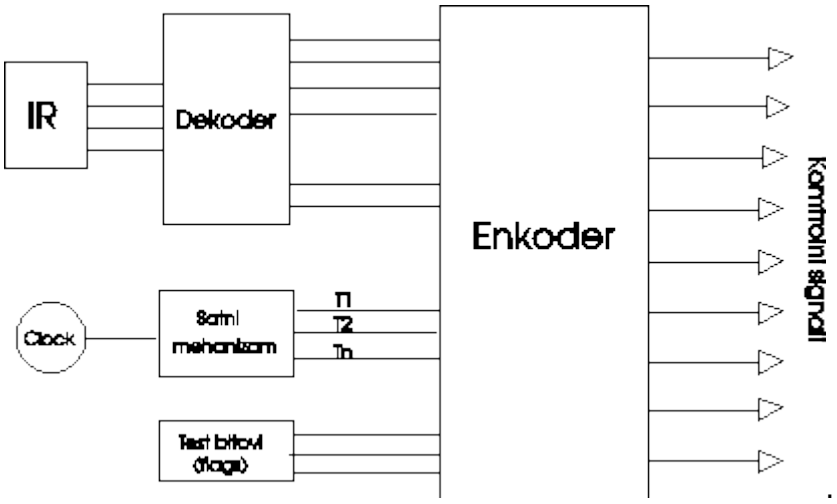
Posmatrajmo niz koraka u FETCH/EXEC ciklusu izvršavanja instrukcija. Svaki korak tog ciklusa zahtijeva odredjeni period vremena koji se definiše putem sata (clock) ako se radi o sinhronom procesoru. Akcija koju CPU izvršava za svaki korak definise se pomoću sjedećih stanja:

(1) izlaza iz dekodera - ovime se definiše mašinska instrukcija koja se izvršava;

(2) clock signala - ovime se definiše početak i trajanje instrukcije,

(3) test bitova (flags) koje postavlja ALU.

Kod "hardwired" kontrolne jedinice stanja (1) - (3) se dovode kao ulazi na složeno kombinatorno kolo koje se naziva enkoderom (slika 7.5).



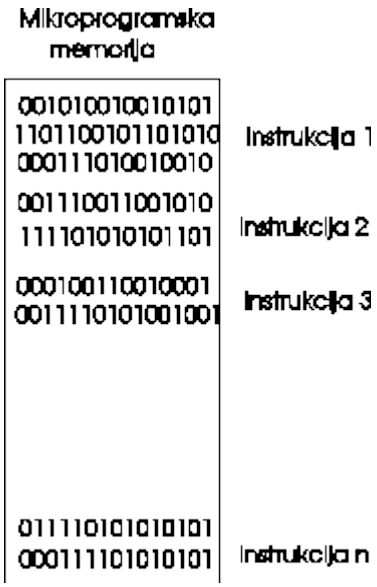
Slika 7.5 Hardverska kontrolna jedinica

Na enkoder se dovode ulazi iz dekodera, satnog mehanizma i ALU-a.

Samo jedna linija iz dekodera će imati izlaza 1 i označavaće instrukciju koju treba izvršiti. Takođe, samo jedna linija iz satnog mehanizma će imati 1 na izlazu i označavaće koji je trenutni period u izvršavanju instrukcije. Samo se na test signalima mogu pojaviti više jednica na izlazu. Izlaz iz enkodera zavisi od ovih ulaza i definiše se kombinatornom logikom unutar enkodera. Kako je već rečeno, svi registri su povezani medju sobom pomoću bus-a. Povezivanje registra na bus se ostvaruje posebnim prekidačem (gate) tako da se može njime upravljati (prekidač uključen registar povezan na bus, prekidač isključen registr otkačen od bus-a). Tako se dovođenjem kontrolnog signala na ovaj prekidač upravlja spajanjem i rastavljanjem registara na bus. Isti princip se koristi i za povezivanje ALU-a.

7.4.2 Mikroprogramska kontrola

Alternativni način za generisanje kontrolnih signala je softverska tehnika poznata kao mikroprogramirmska kontrola. Izlaze iz enkodera posmatrajmo kao jednu memorijsku riječ sa određenom kombinacijom nula i jedinica. Za izvršavanje mašinske instrukcije potrebno je generisati jednu ili više kontrolnih riječi u zavisnosti od tipa mašinske instrukcije. Tako se svaki elementarni kork mašinske instrukcije, kako je to definisano u poglavlju 7.2, može predstaviti sa jednom memorijskom riječi, a cijela mašinska instrukcija sa više takvih riječi, Slika 7.6.



Slika 7.6 Mikroprogramska kontrolna memorija

Niz riječi koje definišu kontrolne signale za neku mašinsku instrukciju naziva se mikroprogramom te instrukcije. Prema tome, za izvršavanje mašinske instrukcije potrebno je redom iz mikroprogramske memorije uzimati riječi koje definišu kontrolne signale za datu instrukciju.

Ovakav način ima pogodnost u tome što se za istu arhitekturu CPU-a mogu definisati različiti skupovi mašinskih instrukcija jednostavnom promjenom sadržaja mikroprogramske memorije. Hardversko rješenje, međutim, ima prednost u brzini rada.