2. NIVOI MAŠINE

Već smo naglasili da se, sa korisničke tačke gledišta, arhitektura računarskog sistema može posmatrati kao funkcionalni opis tog računarskog sistema. Postavlja se sada jedno pitanje: Ko je taj korisnik? Da li je to lice koje koristi aplikacioni program? Programer tog aplikacionog programa? Ili sistemski programer? U suštini, pitanje samo po sebi daje objašnjenje: uvek kada se govori o korisniku glavnu razliku predstavlja nivo na kome korisnik vidi sistem. U daljem tekstu ukazaćemo na ove nivoe detaljnije.

2.1. Identifikacija nivoa mašine

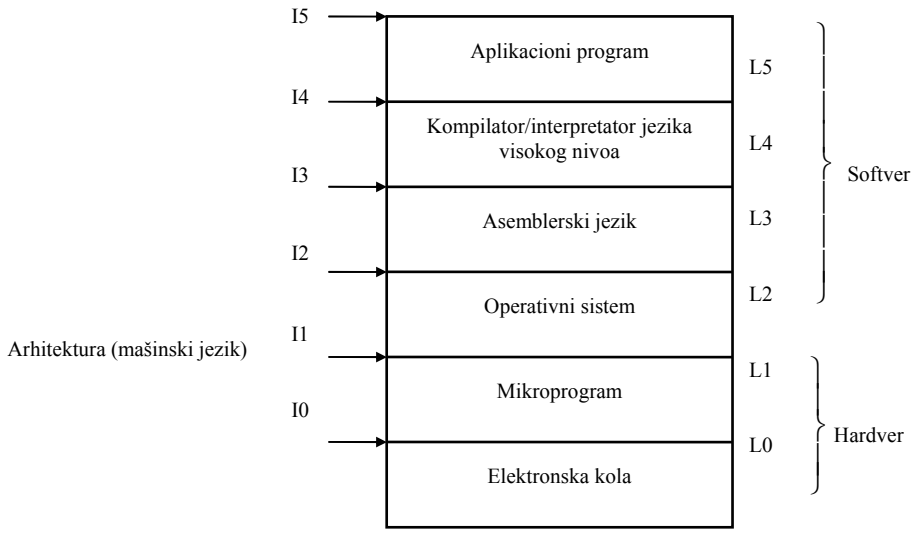
Kod donošenja odluke koje će instrukcije formirati "mašinski jezik" računarskog sistema, projektant treba da sagleda balans između performanse sa jedne strane i kompleksnosti mašine i ukupne cene hardvera i softvera, sa druge strane. Istorijski posmatrano, zahtevi za visokim performansama su uslovili potrebu za kreiranjem moćnih instrukcija, a to je uslovilo ugradnju moćnih elektronskih kola i, naravno, višu cenu. Zahtevi da se projektuje mašina opšte namene uslovili su korišćenje instrukcija koje po svojoj prirodi nisu specijalizovane. Kompromis da se realizuje računar opšte namene uslovljava da mašinski jezik mora da bude jednostavan a to sa druge strane dovodi da napisani programi za tu mašinu budu teško razumljivi korisniku. Nasuprot ovakvom pristupu, programski jezici višeg nivoa (HLL-*High Level Languages*) kao što su Pascal, Algol i dr., projektovani su tako da oslobode korisnika najsitnijih detalja i da obezbede veću produktivnost kod razvoja programa, kao i veću produktivnost u toku održavanja. Jedan od razloga je taj što su programi na HLL-u razumljiviji i imaju konstrukcije koje omogućavaju da se problem može lako rešiti.

Na slici 2.1 prikazana je jednostavna operacija dodele zbira dva broja promenljivoj, kada je ta operacija programirana na HLL (Pascal), asemblerskom jeziku i mašinskom jeziku za mikroprocesor MC68020.

Kao što se vidi sa slike 2.1, postoje tri jezička nivoa kod računarskog sistema. Na slici 2.2 prikazan je jedan dodatni mašinski nivo (ne postoji na slici 2.1), lociran između asemblerskog jezika i mašinskog jezika, poznat kao *nivo operativnog sistema*.

```
        Nivo jezika visokog nivoa (Pascal):
                A:=B+C;
        Nivo asemblerskog jezika (MC68020):
                MOVE.W                  B,D1
                ADD.W           C,D1
                MOVE.W                  D1,A
        Nivo mašinskog jezika (MC68020) (u bitovima)
1000:   MOVE.W          (0x2002).W,D1   0011 0010 0011 1000
                                        0010 0000 0000 0010
1004    ADD.W           (0x2004).W,D1   1101 0010 0111 1000
                                        0010 0000 0000 0100
1008:   MOVE.W          D1,(0x2000).W   0011 0001 1100 0001
                                        0010 0000 0000 0000
2000:   A
2002:   B
2004:   C
```

Sl. 2.1. Različiti nivoi jezika.



Sl. 2.2. Nivoi mašine u računarskom sistemu.

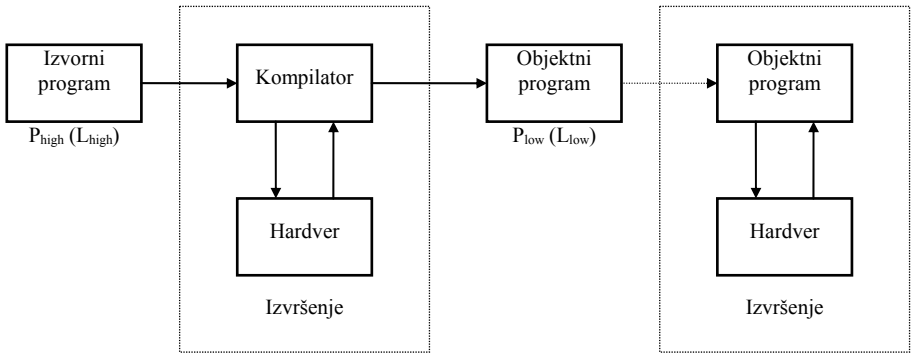
Za najveći broj računarskih sistema se može smatrati da sadrže pet mašinskih nivoa, označenih sa L1 do L5. Postoji šest interfejsa između ovih nivoa označenih sa I0-I5. Nivoi L0 i L1 formiraju hardver sistema. Ponekad je termin hardver nejasan, jer se nivo L0 smatra hardverom, a nivo L1 je "*firmware*" (softver vezan za taj hardver). Nivoi L2-L5 se zovu softver. Nivo L5 čine namenski aplikacioni programi. Nivo L2 se često zove hibridni, jer sadrži dve različite klase funkcija koje se izvršavaju na različite načine. Jednu klasu čine instrukcije koje se izvršavaju na mašinskom jeziku, dok druga klasa omogućava kreiranje novih zadataka (procesa), sinhronizaciju, U/I, upravljanje memorijom, itd.

2.2. Konverzioni mehanizmi između mašinskih nivoa

Analizirajući sliku 2.2, postavlja se osnovno pitanje: Kako se vrši konverzija iz jednog nivoa u drugi? Kod odgovora na ovo pitanje razmatraćemo samo konverziju sa višeg nivoa na niži, pošto je obrnuta konverzija skoro nemoguća. Na nižem nivou, nije više moguće (ili je vrlo teško) da se prepozna koja je operacija bila početno zahtevana za izvršenje od strane programera. Sa slike 2.1 možemo zaključiti da, analizirajući instrukcije na najnižem nivou, nije više moguće ukazati kom iskazu HLL-a one pripadaju. Ovo se često zove *gubitak semantike*. Ova velika razlika u semantikama između mašinskog jezika i HLL se zove "semantička praznina" (semantic gap). Da bi prešli sa jednog nivoa na drugi koriste se kompilacija, interpretacija ili njihova kombinacija. Kombinacija podrazumeva uvođenje međujezika.

2.2.1. Kompilacija

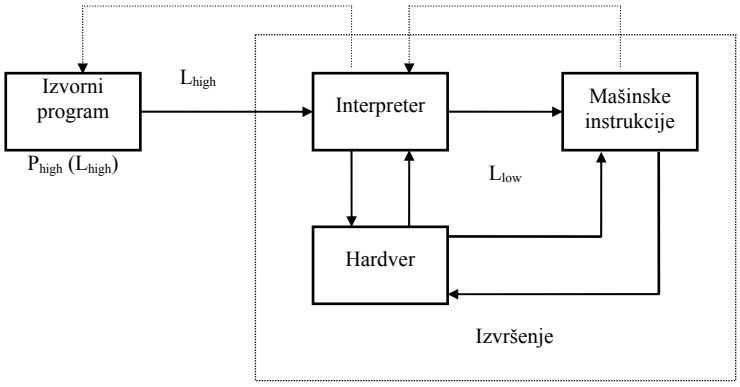
U toku kompilacije *izvornog* programa P_{high} (napisan na jeziku višeg nivoa L_{low}) u *objektni* program na nižem jezičkom nivou L_{low} (koji se prihvata od strane mašine), svaki iskaz programa P_{high} se zamenjuje nizom instrukcija iz L_{low}, tako da one zajedno imaju isti efekat kao i početni iskaz iz P_{high}. Ovaj konverzioni proces obično rezultira u kreiranju programa P_{low} na jeziku L_{low}. Proces konverzije se vrši pomoću programa koji se zove *kompilator*. Kompilator prihvata početni program P_{high} kao ulazne podatke i generiše objektni program P_{low} kao izlazne podatke. Nakon kompilacije objektni program se puni (loaduje) u mašinu radi izvršenja. (slika 2.3).



Sl. 2.3. Kompilacija iz L_{high} u L_{low}.

2.2.2. Intetrpretacija

Kada program P_{high}, napisan na jeziku L_{high}, se interpretira na nivou Ll_{ow}, zahteva se izvršenje specijalnog programa koji se zove *interpreter*. Interpreter uzima jednu instrukciju iz programa P_{high}, analizira je, i generiše niz instrukcija na nivou L_{low} koje se izvršavaju tako da imaju isti efekat kao i iskaz na višem nivou. Proces produžava sve dok se kompletni program ne završi (slika 2.4).



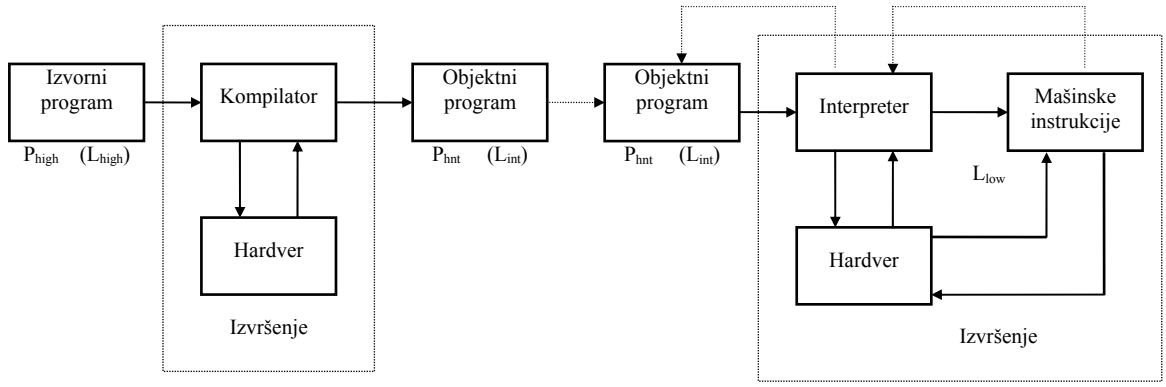
Sl. 2.4. Interpretacija L_{high} u L_{low} nivo.

Prednost ovog konverzinog mehanizma je takva da je interpreter relativno mali, što čini da se lakše prenosi na novu mašinu u odnosu na kompilator. Drugi razlog korišćenja interpretacije ogleda se u sledećem: izvornim jezikom se može zahtevati dinamičko ponašanje tipova podataka i struktura što je kod kompilacije nemoguće; promenljive menjaju tip podatka u toku izvrženja sa celobrojne vrednosti (*integer*) na pokretni zarez (*floating point*). Jezik APL omogućava ovaj tip promene. Ovo znači da "*early binding*", tj. određivanje tipa podataka i strukture kompilacije u toku same kompilacije, neće biti moguće kod APL, već se to izvodi u toku izvršenja programa.

Nedostatak interpretacije ogleda se u tome što je izvršenje izvornog programa, u opštem slučaju, sporije u odnosu na izvršenje objektnog programa generisanog od strane kompilatora. Naime HLL iskazi se moraju analizirati nakon čega se generiše ekvivalentna sekvenca LLL (*Lower Level Languages*) instrukcija. Razlika u brzini je tipično deset puta. Interpretacija se sastoji od velikog broja komplikovanih aktivnosti pa zbog toga dugo traje.

2.2.3. Kombinacija kompilacije i interpretacije

Kombinacija mehanizama za kompilaciju i interpretaciju koristi međujezički nivo L_{int} čiji je nivo između L_{low} i L_{high}. Program P_{high} napisan na izvornom jeziku L_{high} se kompiluje u program P_{int} na jeziku L_{int}. Programa P_{int} se zatim interpretira na nivou L_{low} pomoću interpretera (slika 2.5).



Sl. 2.5. Kombinovana kompilacija i interpretacija.

Kombinovanjem mehanizama kompilacije i interpretacije poboljšava se programska prenosivost, a zadržavaju se prednosti obe metode. Na primer, međujezički nivo L_{int} se može tako odabrati da proces prevođenja bude jednostavan, objektni program P_{int} mali, a kompilator takođe mali (kratak program). Sa druge strane jezik L_{int} je lakše interpretirati u odnosu na L_{high}. Na žalost, vreme izvršenja kompletno kompilovanih programa se ne može dostići jer je interpretacija relativno spor proces.

2.2.4. Standardno korišćeni mehanizmi za konverziju

Konverzioni mehanizmi između nivoa na slici 2.2 su:

- L0: instrukcije mikroprograma (definisane interfejsom I0, gde je L_{high}=I0) se interpretiraju pomoću elektronskih kola na nivou L0.
- L1: Mikroprogram na L1 interpretira mašinski jezik interfejsa I1 sa instrukcijama od I0, tako da L_{high}=I1 a L_{low}=I0.
- L2: L2 je hibridni nivo. Neke instrukcije od I2 su identične sa onim od I1, na primer instrukcija ADD. Ove identične instrukcije se prenose od I2 na I1 radi interpretacije od strane L1. Ostale instrukcije od I2 su tipične instrukcije operativnog sistemsa, kao što su Time-of-Day, ili Print-a-Message, koje se interpretiraju od strane L2.
- L3: Na nivou L3 moguće je govoriti o virtuelnoj L3 mašini, sa jezikom L3 kao mogućim interfejsom sa korisnikom. Ipak, najveći broj korisnika uzdržava se od korišćenja asemblerskog jezika i voli da koristi jezike koji su na višem nivou od L3. Asembler prevodi (to je prost oblik kompilacije) jezik definisan interfejsom I3 (nazvan asemblerski jezik) u jezik definisan sa I2 (koji je interfejs operativnog sistema).
- L4: HLL kompilator, interpreter ili kombinacija pomoću koje se konvertuje HLL interfejs I4 u asemblerski jezik interfejs I3. Ovaj interfejs se takođe zove virtuelna mašina, jer je on za korisnika vidljiv kao da računar direktno izvršava programe na jeziku definisanom od strane tog interfejsa.
- L5: aplikacioni program interpretira aplikaciono orijentisane komande visokog nivoa kao što su Kucaj-Platu-Gospodina-Krtolice, u zavisnosti od jezika definisanog sa I4.