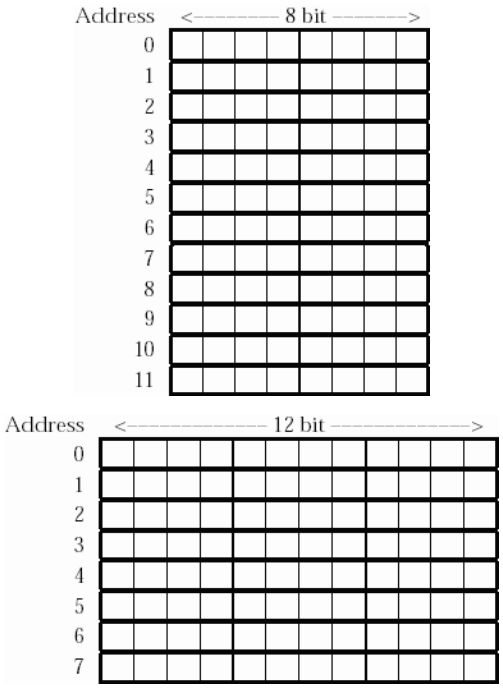
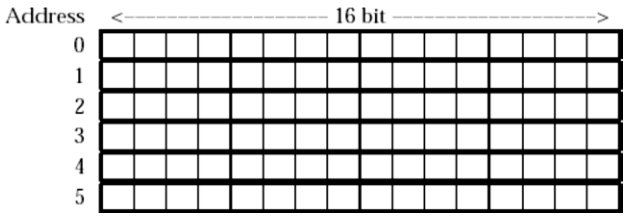
9.

ORGANIZACIJA GLAVNE MEMORIJE

Organizaciju glavne memorije možemo zamisliti kao matricu bitova. Svaka vrsta predstavlja memorijsku lokaciju, koja je obično jednaka veličini reči, iako može biti umnožak veličine reči (npr. 2 x *veličina reči*) ili deo reči (npr. polovina veličine reči). Radi jednostavnosti, pretpostavićemo da se podaci u glavnoj memoriji mogu samo čitati ili upisivati u jednom redu (memorijskoj lokaciji) u datom trenutku.

Za 96-bitnu memoriju, možemo napraviti organizaciju memorije kao 12 x 8 bitova, 8 x 12 bitova ili 6 x 16 bitova, ili čak 91 x 1 bit ili 1 x 96 bitova. Svaki red ima prirodan broj, adresu, koja se koristi za njegov odabir (slika 1).

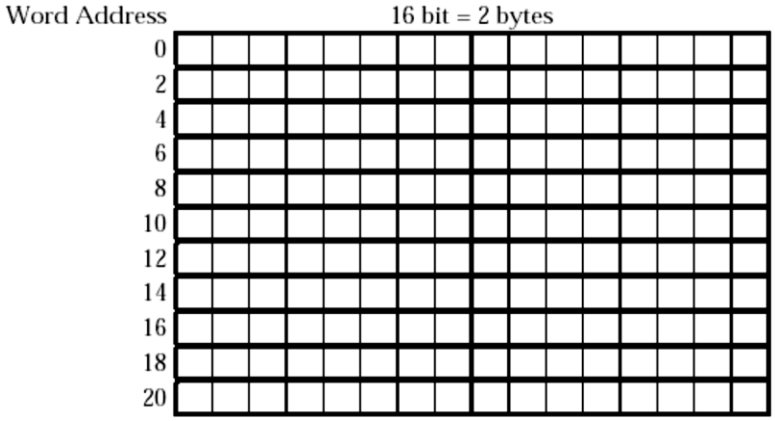




Slika 1. Različita organizacija 96-bitne memorije

9.1 Adresiranje na nivou bajta

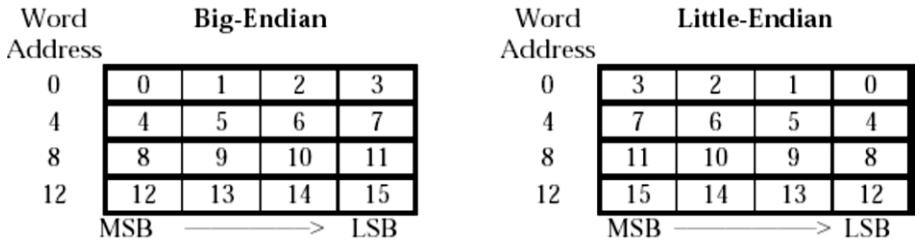
Glavne memorije uglavnom rade sa redovima, koji su veličine umnoška bajta (npr. 16-bitna reč = 2 bajta, 32-bitna reč = 4 bajta). Međutim, većina arhitektura podržava adresiranje glavne memorije na nivou bajta, umesto reči. U takvim arhitekturama, procesor i/ili glavna memorija je sposoban da čita/piše bilo koji pojedinačni bajt. Na slici ispod je prikazan primer glavne memorije sa 16-bitnim memorijskim lokacijama. Treba primetiti da memorijske lokacije (redovi) imaju neparne adrese (slika 2).



Slika 2. Organizacija memorije sa neparnim adresama

9.1.1 Redosled bajtova

Bajtovi unutar reči čija je veličina umnožak bajta se mogu brojati sa leva na desno (*Big-Endian*) ili sa desna na levo (*Little-Endian*). U sledećem primeru (slika 3), ćelije u tabeli predstavljaju bajtove, a broj u ćeliji označava adresu datog bajta u glavnoj memoriji.

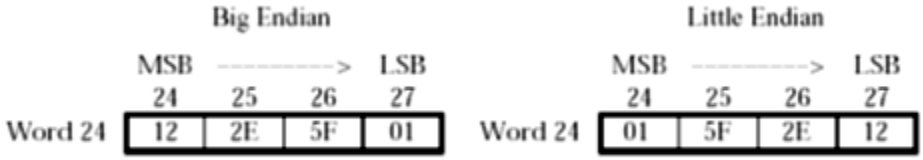


Slika 3. Big-Endian i Little-Endian pristup

U *Big-Endian* sistemima, bajt najveće težine ima najnižu adresu, dok bajt najmanje težine ima najveću adresu.

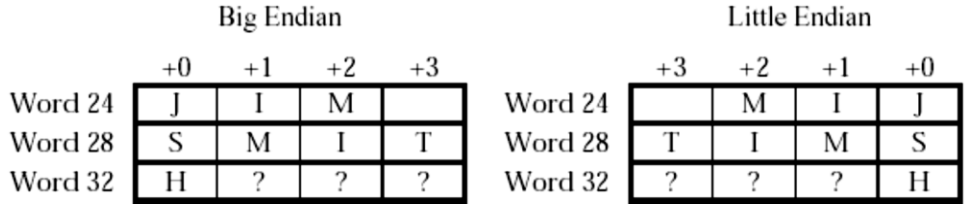
U *Little-Endian* sistemima, bajt najmanje težine ima najnižu adresu, dok bajt najveće težine ima najveću adresu. *Napomena*: vrednost stringa od *N* karaktera se ne tretira kao jedna velika vrednost od više bajtova, već kao *N* jedinstvenih vrednosti tipa karaktera, tj. prvi karakter u stringu uvek ima najnižu adresu, a poslednji ima najveću adresu. Ovo je zajedničko i za *big-endian* i za *little-endian* sisteme.

Primer 1. Prikazati sadržaj memorije na adresi reči 24 ako ta reč sadrži broj dat 122E5F01h, u *big-endian* i *little-endian* sistemima.



Napomena: po konvenciji, bajtovi unutar memorijske reči seređaju sa leva na desno za *big-endian* i sa desna na levo za *little-endian* sisteme.

Primer 2. Prikazati sadržaj glavne memoriji od adrese reči 24, ako te reči sadrže tekst JIM SMITH.



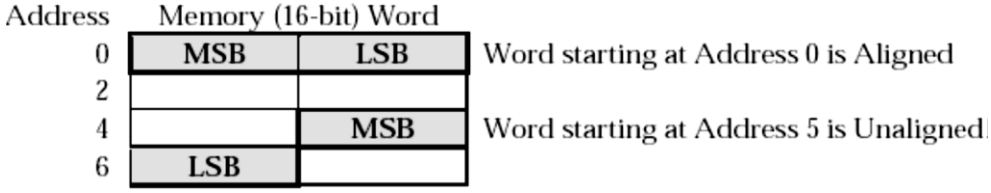
Bajtovi obeleženi sa ? su nepoznati. Oni mogu sadržati značajne podatke ili mogu biti nevažni – interpretacija je ostavljena programeru.

Nažalost, računarski sistemi koji se danas koriste su podeljeni na *big-endian* i *little-endian*. To dovodi do problema kada *big-endian* računar želi da šalje podatke ka *little-endian* računaru.

Neke skorije arhitekture (npr. PowerPC) omogućavaju da se promena sa *big* na *little endian* radi programski.

9.2 Poravnanje na nivou reči

Iako su glavne memorije uglavnom organizovane kao redovi reči adresirane na nivou bajta kojima se pristupa jedan po jedan, neke arhitekture omogućavaju procesoru da pristupi bilo kojoj grupi bitova veličine reči, nezavisno od njene adrese bajta (slika 4). Kaže se da pristup koji počinje na granici memorijske reči je poravnat pristup, dok pristup koji ne počinje na granici reči je neporavnat pristup.



Slika 4. Poravnat i neporavnat pristup reči

Čitanje neporavnate reči iz memorije zahteva:

- Čitanje susednih reči,
- Odabir potrebnog bajta iz svake reči,
- Spajanje tih bajtova.

Ova operacija je spora. Upisivanje neporavnate reči je još složenije i sporije. Iz ovog razloga, neke arhitekture zabranjuju pristup neporavnatim rečima. Na primer, u arhitekturi 68000, rečima se ne sme pristupiti počevši od neparnih adresa (npr. 1, 3, 5, 7, itd.). Neke arhitekture čak proširuju ovaj princip na pristup više reči. Na primer, u SPARC arhitekturi, 64-bitni podatak mora imati adresu bajta koja je umnožak broja 8.

9.3 Podela memorijskog prostora

Prefiksi koji se koriste za označavanje kapaciteta memorije imaju sledeću vrednost:

- 1 k = 2^{10} (npr. 1 kB = 1024 B = 2^{10} B),
- 1 M = 2^{20} (npr. 1 MB = 1024 kB = 2^{20} B),
- 1 G = 2^{30},
- 1 T = 2^{40},…

Zadatak 1. Memorijski adresni prostor je 256 adresa. Nižih 160 adresa je rezervisano za RAM memoriju, a viših 96 adresa je rezervisano za ROM memoriju. Naznačiti opseg adresa u memorijskom adresnom prostoru, kao i opseg adresa rezervisan za RAM i ROM memoriju.

Rešenje.

$256 = 2^8 \rightarrow n = 8$ bita (širina adrese)

Pošto je $160 = 5 * 32$ i $96 = 3 * 32$, sledi da se celokupan adresni prostor može podeliti na osam oblasti od po 32 lokacije, od kojih će donjih pet zauzimati RAM memorija, a gornjih tri ROM memorija.

Opseg adresa u memorijskom adresnom prostoru je:

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Heksadecimalna vrednost
0	0	0	0	0	0	0	0	00
…	…	…	…	…	…	…	…	
1	1	1	1	1	1	1	1	FF

Za predstavljanje osam oblasti od 32 lokacije je potrebno 3 bita ($2^3 = 8$), koji svojim vrednostima određuju pripadnost adresnog prostora RAM ili ROM memoriji.

A_7	A_6	A_5	
0	0	0	
0	0	1	R
0	1	0	A
0	1	1	M
1	0	0	
1	0	1	R
1	1	0	O
1	1	1	M

Opseg adresa RAM memorije je:

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Heksadecimalna vrednost
0	0	0	0	0	0	0	0	00
…	…	…	…	…	…	…	…	
1	0	0	1	1	1	1	1	9F

Opseg adresa ROM memorije je:

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Heksadecimalna vrednost
1	0	1	0	0	0	0	0	A0
…	…	…	…	…	…	…	…	
1	1	1	1	1	1	1	1	FF

Zadatak 2. Memorijski adresni prostor je 1k adresa. Nižih 256 adresa je rezervisano za ROM memoriju, a viših 768 adresa je rezervisano za RAM memoriju.

Naznačiti opseg adresa u memorijskom adresnom prostoru, opseg adresa rezervisan za RAM i ROM memoriju.

Rešenje.

$1k = 2^{10} \rightarrow n = 10$ bita (širina adrese)

Pošto je 768 = 3 * 256 i 256 = 1 * 256, sledi da se celokupan adresni prostor može podeliti na četiri oblasti od po 256 lokacija, od kojih će gornje tri zauzimati RAM memorija, a donju jednu ROM memorija.

Opseg adresa u memorijskom adresnom prostoru je:

A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Heksadecimalna vrednost
0	0	0	0	0	0	0	0	0	0	000
…	…	…	…	…	…	…	…	…	…	
1	1	1	1	1	1	1	1	1	1	3FF

Za predstavljanje četiri oblasti od 256 lokacija je potrebno 2 bita ($2^2 = 4$), koji svojim vrednostima određuju pripadnost adresnog prostora RAM ili ROM memoriji.

A_9	A_8	
0	0	ROM
0	1	R
1	0	A
1	1	M

Opseg adresa ROM memorije je:

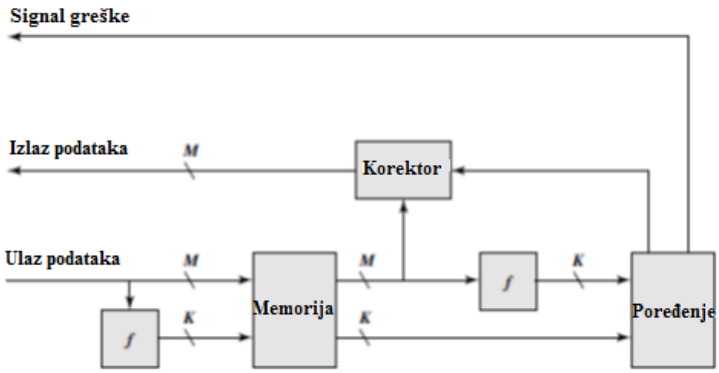
A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Heksadecimalna vrednost
0	0	0	0	0	0	0	0	0	0	000
…	…	…	…	…	…	…	…	…	…	
0	0	1	1	1	1	1	1	1	1	0FF

Opseg adresa RAM memorije je:

A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Heksadecimalna vrednost
0	1	0	0	0	0	0	0	0	0	100
…	…	…	…	…	…	…	…	…	…	
1	1	1	1	1	1	1	1	1	1	3FF

9.4 Ispravljanje grešaka

Poluprovodnička memorija je podložna greškama. One mogu da se okarakterišu kao hardverske i privremene. Hardverska greška je trajan fizički kvar, tako da memorijska ćelija ili ćelije koje su njom obuhvaćene ne mogu pouzdano da skladište podatke, nego se zakoče na 0 ili 1, ili pogrešno preskaču između 0 i 1. Hardverske greške mogu da budu prouzrokovane oštrim povredama iz okoline, greškama u proizvodnji i istrošenošću. Privremena greška je slučajan, nedestruktivan događaj, koji menja sadržaj jedne ili više memorijskih ćelija, bez oštećenja memorije. Privremene greške mogu da budu prouzrokovane problemima u napajanju električnom energijom, ili alfa česticama. Te čestice rezultuju iz radioaktivnog raspadanja i, nažalost, veoma su prisutne, zato što se radioaktivna jezgra u malim količinama nalaze u skoro svim materijalima. I hardverske i privremene greške su nepoželjne i većina savremenih memorijskih sistema uključuje logiku, kako za otkrivanje, tako i za ispravljanje grešaka.



Slika 13. Funkcija za korekciju greške

Na slici 13 ilustrovano je, u opštim pojmovima, kako se odvija proces. Kada podaci treba da se učitaj u memoriju, izvodi se proračun nad podacima, opisan kao funkcija *f* da bi se proizveo kod. Skladište se i kod i podaci. Dakle, ako treba da se uskladišti *M* – bitna reč podataka, a kod je dužine *K* bitova, onda je stvarna dužina uskladištene reći *M* + *K* bitova.

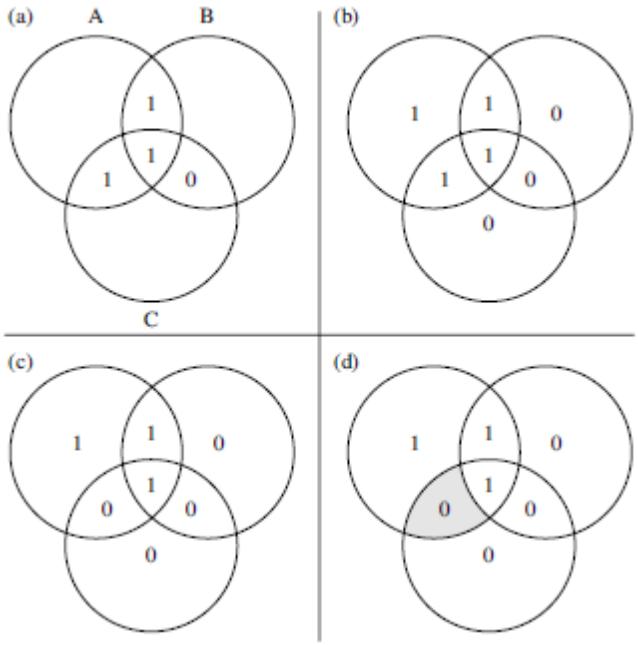
Kada se čitaju prethodno uskladišteni podaci, kod se koristi da bi se otkrile i možda ispravile greške. Novi skup od *K* bitova koda generiše se iz *M* bitova podataka i poredi sa bitovima donesenog koda. Poređenje može da dovede do jednog od sledeća tri rezultata:

1. Nije otkrivena nikakva greška. Doneseni podaci se šalju napolje.

2. Otkrivena je greška i moguće je da se ona ispravi. Bitovi podataka i bitovi za ispravljanje greške unose se u korektor, koji proizvodi ispravljen skup od M bitova da se pošalje napolje.
3. Otkrivena je greška, ali nije moguće da se ona ispravi. O tom uslovu se izveštava.

Kodovi koji rade na taj način, zovu se kodovi za ispravljanje grešaka. Kod se karakteriše brojem grešaka bitova koje može da ispravi i otkrije.

Najjednostavniji od kodova za ispravljanje grešaka je Hamingov kod, koji je izumeo *Richard Hamming* u *Bell* – ovim Laboratorijama. Na slici 14, upotrebljeni su Venovi dijagrami da bi se ilustrovala upotreba tog koda na 4 – bitnim rečima ($M = 4$). Kada imamo tri kruga koji se međusobno seku, postoji sedam odeljaka. Dodelićemo 4 bita podataka unutrašnjim odeljcima (slika 14a). Preostali odeljci se pune onim što zovemo bitovima parnosti. Svaki bit parnosti odabran je tako da je ukupni broj cifara 1 u njegovom krugu paran (slika 14b). Na taj način, zato što krug A uključuje tri cifre podataka 1, bit parnosti u tom krugu postavlja se na 1. Kada se zbog greške jedan od bitova podataka promeni (slika 14c), to se lako otkriva. Proveravanjem bita parnosti, otkrivaju se nesaglasnosti u krugovima A i C, ali ne i u krugu B. Samo jedan od sedam odeljaka je u A i C, ali ne i u B. Greška zato može da se ispravi promenom tog bita.



Slika 14. Hamingov kod za otklanjanje greški

Da bismo razjasnili koncepte s tim u vezi, razvićemo kod koji može da otkrije i ispravi greške na jednom bitu u 8-bitnim rečima.

Za početak, hajde da odredimo koju dužinu treba da ima kod. Ako pogledamo sliku 9, logika za poređenje prima kao ulaz dve K – bitne vrednosti. Poređenje bit po bit vrši se izračunavanjem ekskluzivnog – ILI za dva ulaza. Rezultat se zove *sindromska reč*. Dakle, svaki bit sindroma je 0 ili 1, u zavisnosti od toga da li postoji ili ne postoji podudarnost na toj poziciji bita za dva ulaza.

Sindromska reč je zato širine K bitova i ima opseg između 0 i $2^K - 1$. Vrednost 0 pokazuje da nije otkrivena nikakva greška, što ostavlja $2^K - 1$ vrednosti da pokažu, ako je bilo greške, koji od bitova je bio pogrešan. Sada, s obzirom na to da bi greška mogla da se dogodi na bilo kom od M bitova podataka ili K bitova za proveru, moramo da imamo:

$$2^K - 1 \geq M + K$$

Ova nejednačina daje broj bitova potreban da se ispravi jedna greška u reci koja sadrži M bitova podataka. Na primer, za reč od 8 bitova ($M = 8$), imamo

$$K = 3: 2^3 - 1 < 8 + 3$$

$$K = 4: 2^4 - 1 > 8 + 4$$

Prema tome, osam bitova podataka zahtevaju četiri bita za proveru. U prve tri kolone tabele 1 nalaze se brojevi bitova za proveru koji se zahtevaju za različite dužine reči podataka.

Tabela 1. Povećanje dužine reči sa ispravljanjem grešaka

	Ispravljanje jedne greške		**Ispravljanje jedne greške / otkrivanje dve greške**	
Bitovi podataka	**Bitovi za proveru**	**% povećanja**	**Bitovi za proveru**	**% povećanja**
8	4	50,00	5	62,5
16	5	31,25	6	37,5
32	6	18,75	7	21,875
64	7	10,94	8	12,5
128	8	6,25	9	7,03
256	9	3,52	10	3,91

Radi pogodnosti, želeli bismo da generišemo 4 – bitni sindrom za 8 – bitnu reč podataka sa sledećim karakteristikama:

- Ako sindrom sadrži sve cifre 0, nije otkrivena nijedna greška.
- Ako sindrom sadrži jedan i samo jedan bit postavljen na 1, onda se greška pojavila u jednom od 4 bita za proveru. Nije potrebna nikakva ispravka.

– Ako sindrom sadrži više od jednog bita postavljenog na 1, onda numerička vrednost sindroma pokazuje poziciju bita podatka sa greškom. Taj bit podatka se invertuje radi popravke.

Da bi se postigle te karakteristike, bitovi podataka i provere raspoređuju se u 12 – bitnu reč, kao što je prikazano na slici 15. Pozicije bitova su numerisane od 1 do 12. One pozicije bitova, čije pozicione brojeve predstavljaju stepeni od 2, označene su kao bitovi za proveru. Bitovi za proveru se proračunavaju na sledeći način, gde simbol ⊕ označava operaciju ekskluzivno – ILI:

$$C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7$$

$$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$$

$$C4 = D2 \oplus D3 \oplus D4 \oplus D8$$

$$C8 = D5 \oplus D6 \oplus D7 \oplus D8$$

Vrednosti koje operacija isključivo – ILI daje nad dva jednobitna operanda su:

A	**B**	**F = A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

Svaki bit provere radi na svakom bitu podatka čiji broj pozicije sadrži 1 u istoj poziciji bita kao što je broj pozicije tog bita za proveru. Prema tome, pozicije bitova podataka 3, 5, 7, 9 i 11 (Dl, D2, D4, D5, D7) svi sadrže 1 u najmanje značajnom bitu njihovog broja pozicije kao što radi C1; pozicije bitova 3, 6, 7, 10 i 11 sve sadrže 1 u drugoj poziciji bita, kao što radi C2; i tako dalje. Posmatrano na drugi način, pozicija bita *n* proverava se onim bitovima C_i, koji su takvi da je $\sum i < n$. Na primer, poziciju 7 proveravaju bitovi na pozicijama 4, 2 i 1; a 7 = 4 + 2 + 1.

Pozicija bita	12	11	10	9	8	7	6	5	4	3	2	1
Broj pozicije	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit podatka	D8	D7	D6	D5		D4	D3	D2		D1		
Bit za proveru					C8				C4		C2	C1

Slika 15. Raspored bitova podataka i bitova za proveru

Hajde da proverimo da li ta šema radi pomoću jednog primera. Pretpostavite da je 8 – bitna ulazna reč 00111001, gde je bit podatka Dl na krajnjoj desnoj poziciji. Proračuni su sledeći:

$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$

$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Sad pretpostavite da je bit 3 pretrpeo grešku i da se promenio sa 0 na 1. Kada se ponovo izračunaju bitovi provere, imamo:

$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$

$C4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$

$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Kada se novi bitovi provere uporede sa starim bitovima provere, formira se reč sindroma:

	C8	C4	C2	C1
	0	1	1	1
⊕	0	0	0	1
	0	1	1	0

Rezultat je 0110, što pokazuje da je pozicija bita 6, koja sadrži bit podatka 3, u grešci.

Na slici 16 ilustrovan je prethodni proračun. Bitovi podataka i provere su na odgovarajući način postavljeni u 12 – bitnoj reči. Četiri bita podataka imaju vrednost 1 (osenčeni u tabeli), i njihove pozicije bitova su podvrgnute operaciji ekskluzivno – ILI da bi se proizveo Hamingov kod 0111, koji formira četiri cifre za proveru. Ceo blok koji se skladišti je 001101001111. Sad pretpostavite da bit podataka 3, na poziciji bitova 6, pretrpi grešku i da se promeni sa 0 na 1. Rezultujući blok je 001101101111, sa Hamingovim kodom od 0111. Ekskluzivno – ILI Hamingovog koda i svih vrednosti pozicija bitova za bitove podataka različite od nule kao rezultat daje 0110. Rezultat različit od nule otkriva grešku i pokazuje daje greška u poziciji bita 6.

Pozicija bita	12	11	10	9	8	7	6	5	4	3	2	1
Broj pozicije	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit podatka	D8	D7	D6	D5		D4	D3	D2		D1		
Bit za proveru					C8				C4		C2	C1
Uskladištena reč	0	0	1	1	0	1	0	0	1	1	1	1
Donesena reč	0	0	1	1	0	1	1	0	1	1	1	1
Broj pozicije	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit za proveru					0				0		0	1

Slika 16. Proračun bitova za proveru

Zadatak 3. Bitovi podatka i koda su raspoređeni u 12-bitnoj reči 101110100101. Primenom Hamingovog koda, odrediti da li je došlo do greške u podatku.

Rešenje.

Pozicija bita	12	11	10	9	8	7	6	5	4	3	2	1
Broj pozicije	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit	1	0	1	1	1	0	1	0	0	1	0	1
Bit podatka	D8	D7	D6	D5		D4	D3	D2		D1		
Bit za proveru					C8				C4		C2	C1

Bitovi koda koji se nalaze u podatku su:

C1 = 1

C2 = 0

C4 = 0

C8 = 1

Ispravni bitovi koda prema Hamingovom kodu su:

$$Cl = Dl \oplus D2 \oplus D4 \oplus D5 \oplus D7 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 1 \oplus 0 \oplus 1 \oplus 0 = 1 \oplus 1 \oplus 0 = 0 \oplus 0 = 0$$

$$C2 = Dl \oplus D3 \oplus D4 \oplus D6 \oplus D7 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$C4 = D2 \oplus D3 \oplus D4 \oplus D8 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$

$$C8 = D5 \oplus D6 \oplus D7 \oplus D8 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$$

Formiranje sindroma:

	C8	C4	C2	C1
	1	0	0	1
⊕	1	0	1	0
	0	0	1	1

Pošto u sindromu postoje dve jedinice, to vrednost sindroma određuje broj pozicije koja je u grešci. U ovom slučaju, bit na poziciji 3, odnosno bit D1 je pogrešno prenet i treba da ima vrednost 0.

Zadatak 4. Bitovi podatka i koda su raspoređeni u 12-bitnoj reči 100100111100. Primenom Hamingovog koda, odrediti da li je došlo do greške u podatku.

Rešenje.

Pozicija bita	12	11	10	9	8	7	6	5	4	3	2	1
Broj pozicije	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit	1	0	0	1	0	0	1	1	1	1	0	0
Bit podatka	D8	D7	D6	D5		D4	D3	D2		D1		
Bit za proveru					C8				C4		C2	C1

Bitovi koda koji se nalaze u podatku su:

C1 = 0

C2 = 0

C4 = 1

C8 = 0

Ispravni bitovi koda prema Hamingovom kodu su:

Cl = Dl ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7 = 1 ⊕ 1 ⊕ 0 ⊕ 1 ⊕ 0 = 1

C2 = Dl ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7 = 1 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 0 = 0

C4 = D2 ⊕ D3 ⊕ D4 ⊕ D8 = 1 ⊕ 1 ⊕ 0 ⊕ 1 = 1

C8 = D5 ⊕ D6 ⊕ D7 ⊕ D8 = 1 ⊕ 0 ⊕ 0 ⊕ 1 = 0

Formiranje sindroma:

	C8	C4	C2	C1
	0	1	0	0
⊕	0	1	0	1
	0	0	0	1

Pošto u sindromu postoji jedna jedinica, to vrednost sindroma ukazuje da je greška u jednom od bitova za proveru, tako da nikakve korekcije nisu potrebne.