SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1821

OSNOVNE RUTINE ZA IZGRADNJU OPERACIJSKOG SUSTAVA

Jure Budiša

Zagreb, studeni 2009.

Sažetak

U ovom diplomskom radu ostvarene su osnovne rutine koje olakšavaju izgradnju jednostavnih operacijskih sustava. Osnovne rutine namijenjene su za izgradnju operacijskih sustava za x86 arhitekturu. Korištenjem rutina izgrađen je jednostavni operacijski sustav. Operacijski sustav je monolitnog dizajna i podržava višezadaćnost. Idući koraci u razvoju operacijskog sustava su ostvarenje dinamičke dodjele memorije i podrške za datotečni sustav. U radu su opisani resursi koji su korišteni prilikom izgradnje operacijskog sustava.

Abstract

In this thesis work are achieved basic routines which make things easy at building simple operating systems. Basic routines are destined to build operating systems for x86 architecture. By using routines, simple operating system has built. Operating system is monolithic designed and support multitasking. Next steps in development of operating system are realizing dynamic memory allocation and file system support. There are described resources, at work, which were used when the operating system has been building.

Sadržaj

1. Uvod

Cilj rada je osmisliti i ostvariti rutine koje olakšavaju izradu jednostavnih operacijskih sustava. Korištenjem rutina potrebno je ostvariti neke funkcionalnosti operacijskog sustava.

Implementirane rutine namijenjene su za izgradnju operacijskog sustava za x86 arhitekturu. Ostvareni jednostavni operacijski sustav je monolitnog dizajna. Za izradu rutina i operacijskog sustava korišten je programski jezik C i strojni (*asemblerski*) jezik.

Ovaj rad sadrži osam poglavlja. Drugo poglavlje sadrži općenite informacije o operacijskom sustavu. U tom poglavlju opisan je najvažniji dio operacijskog sustava – jezgra operacijskog sustava. Prikazana su dva glavna modela jezgre: monolitna jezgra i mikro jezgra. Oba modela imaju određene prednosti ali i određene nedostatke. Većina operacijskih sustavi uglavnom nastoje iskoristiti prednosti obje metode.

U trećem poglavlju opisana je segmentacija i straničenje kod x86 arhitekture, prekidi, kontroler prekida (PIC), brojilo sustava (PIT), sat stvarnog vremena (RTC), ispis znakova na zaslon, upravljanje podacima tipkovnice itd.

U četvrtom poglavlju opisane su ostvarene rutine, a u petom poglavlju je na primjerima pokazano kako se mogu koristiti neke rutine za izgradnju operacijskog sustava.

2. Općenito o operacijskom sustavu

Operacijski sustav upravlja sklopovskim i programskim resursima sustava. Razni programi natječu se za pozornost procesora (CPU-a) i potražuju memoriju . Operacijski sustav brine se da svaki program dobije potrebne resurse.

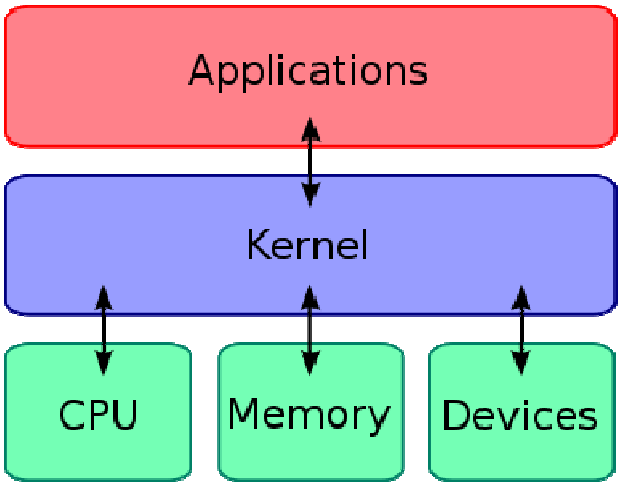
Isto tako operacijski sustav osigurava programima stabilan i dosljedan način korištenja sklopovlja. Programi pri tome ne trebaju poznavati sve njegove detalje.

Kategorije zadataka operacijskog sustava su :

- Upravljanje procesorom (raspoređivanje – engl. *scheduling*),
- Upravljanje memorijom,
- Upravljanje uređajima,
- Upravljanje pohranom (engl. *storage management*),
- Programsko sučelje (API – *Application Programming Interface*),
- Korisničko sučelje (GUI – *Graphical User Interface*).

2.1. Jezgra

Jezgra (engl. *kernel*) je središnji dio operacijskog sustava. Programe i sklopovlje povezuje jezgra. Upravo to prikazuje sljedeća slika.



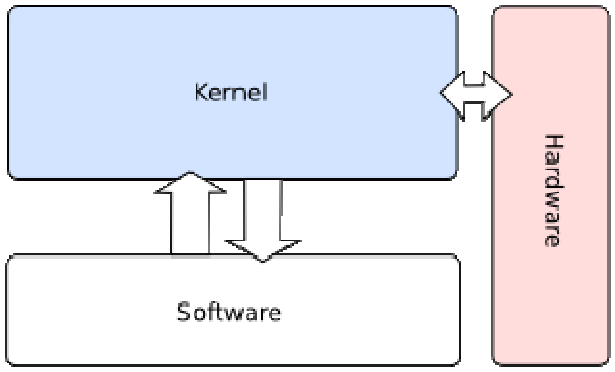
Slika 1. Jezgra povezuje programe sa sklopovljem [46]

Jezgra upravlja sklopovljem i pruža usluge aplikacijama. Obično se izvršava u nadglednom načinu rada (engl. *supervisor mode*).

Glavni modeli jezgre su:

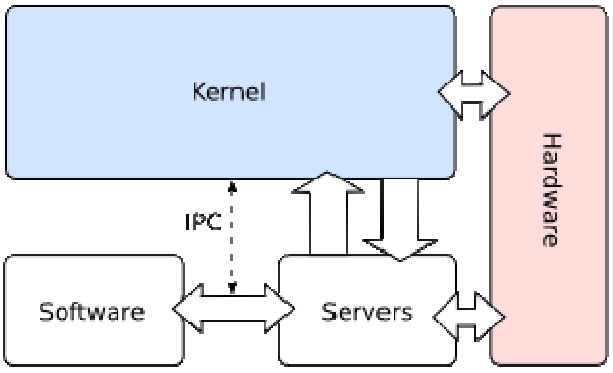
- monolitna jezgra i
- mikro-jezgra.

Monolitne jezgre obavljaju sve osnovne usluge sustava u jezgrinom prostoru. Sve jezgrine usluge ili većina njih dio su same jezgre. Cijela jezgra izvršava se u jezgrinom prostoru u nadglednom načinu rada (prsten 0). Jezgra je na primjer zadužena za upravljanje procesima, upravljanje memorijom, rukovanje prekidima, datotečni sustav, I/O komunikaciju itd. Veliki nedostaci ove jezgre su veličina jezgre, nedostatak proširivosti i loše održavanje. Ove jezgre su uobičajene za 80x86/PC arhitekturu. Neki od primjera monolitnih jezgara su Linux, Windows 9x (95, 98, Me), Solaris, BSD jezgre. Kod ovog pristupa sve komponente vjeruju jedna drugoj. Tako da u slučaju greške u bilo kojem dijelu jezgre često uzrokuje pad cijelog sustava. Na slici 2. prikazana je monolitna jezgra.



Slika 2. Osnovni pregled monolitne jezgre [50]

Kako bi se nadvladali nedostaci monolitne jezgre pojavila se ideja o mikro-jezgrama. Mikro-jezgra obavlja manji skup operacija. Većina usluga (npr. datotečni sustav, upravljački programi, itd.) se izvodi u korisničkom prostoru. Postoji servis za upravljanje memorijom, servis za upravljanjem procesima itd. Jezgra pruža samo osnovne funkcionalnosti. Mikro-jezgra obično podržava sljedeće funkcije: komunikacija među procesima (engl. *Inter-process Communication* - IPC), kratkoročno planiranje (engl. *short-term scheduling*), upravljanje memorijom na niskoj razini (engl. *low level*), ulaz/izlaz na niskoj razini, mrežna podrška na niskoj razini. Procesi unutar jezgre se izvode u jezgrinom načinu. Neki od primjera mikro-jezgara su Mach, QNX, L4, Minix, AmigaOS, Symbian OS. Na slici 3. prikazana je mikro-jezgra.



Slika 3. Osnovni pregled mikro jezgre [49]

Većina operacijskih sustava uglavnom nisu čisto monolitni ili čisto mikro-jezgreni. Nastoje se iskoristiti prednosti obiju metoda.

3. Upravljanje resursima sustava

U ovom poglavlju opisani su resursi koji su korišteni prilikom izgradnje operacijskog sustava.

3.1. Prikazivanje znakova na zaslonu

Kako bi se neki znak prikazao na zaslonu potrebno ga je zapisati u video međuspremnik (engl. *framebuffer*). Kod uobičajenog tekstualnog načina rada u boji video međuspremnik se nalazi na adresi 0xB8000, dok se kod jednobojnog tekstualnog načina rada nalazi na adresi 0xB0000. Video međuspremnik je dio video memorije VGA kontrolera. Međuspremnik se memorijski mapira putem sklopovlja u linearni adresni prostor (0xB8000-0xBFFFF kod uobičajenog tekstualnog načina u boji).

Tablica 1. 16-bitni element video međuspremnika

Bitovi	Opis
0 - 7	znak
8 - 10	boja znaka
11	intezitet boje znaka
12 - 14	boja pozadine
15	treperenje znaka

Video međuspremnik se sastoji od 16 bitnih elemenata. Nižih 8 bita svakog elementa je znakovni bajt, a viših 8 bita je atributni bajt. Znakovni bajt sadržava ASCII vrijednost znaka. Atributni bajt sadržava informacije o boji znaka, boji pozadine toga znaka i o treperenju znaka.

Za ažuriranje kursora i za dobivanje pozicije kursora bitni su indeksni i podatkovni registar koje koristi VGA kontroler. Indeksni bajt se zapisuje na port 0x3D4, a podatkovni bajt na port 0x3D5. Kod jednobojnog tekstualnog načina koriste se portovi 0x3B4 i 0x3B5.

Tablica 2. Registri koji se koriste za pristup CRTC registrima

Port	Registar
0x3D4 / 0x3B4	indeksni registar
0x3D5 / 0x3B5	podatkovni registar

Tablica 3. Boje

Boja	Vrijednost
crna	0
plava	1
zelena	2
cijan	3
crvena	4
ružičasta	5
smeđa	6
svijetlo siva	7
tamno siva	8
svijetlo plava	9
svijetlo zelena	10
svijetlo cijan	11
svijetlo crvena	12
svijetlo ružičata	13
svijetlo smeđa	14
bijela	15

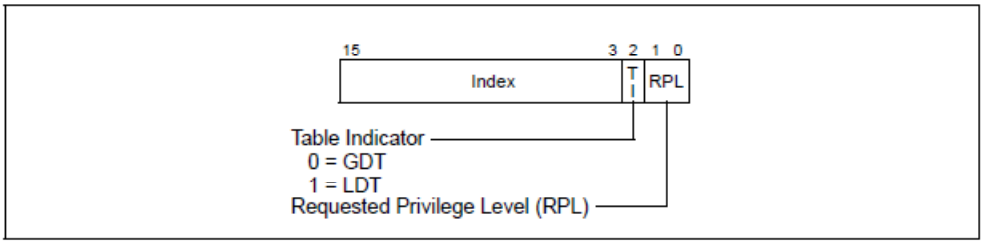
3.2. Segmentacija i straničenje

Kod x86 arhitekture postoje dvije metode za zaštitu memorije i za pružanje virtualne memorije. Te metode su segmentacija i straničenje.

3.2.1. Segmentacija

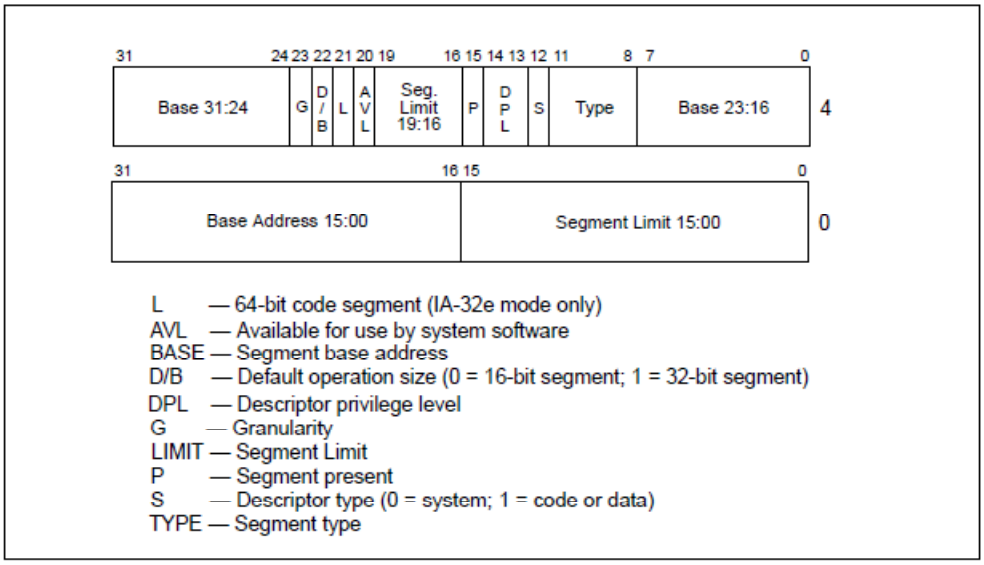
Segmentacija pruža mehanizam dijeljenja linearnog adresnog prostora u manje zaštićene adresne prostore. Ti manji adresni prostori nazivaju se segmenti. Tablica GDT (engl. *Global Descriptor Table*) sadrži informacije o segmentima.

Izbornik segmenta (engl. *segment selector*) je identifikator segmenta. Pokazuje na opisnik segmenta (engl. *segment descriptor*) koji definira segment. Sastoji se od indeksa (od 3. do 15. bita), TI zastavice (bit 2) i zahtjevane razine privilegija RPL (engl. *requested privilege level*) (bit 0 i 1). Sa indeksom se odabire jedan od opisnika u tablici GDT ili LDT (engl. *Local Descriptor Table*). Zastavicom TI se određuje tablica koja se koristi (0 = GDT, 1 = LDT), a sa RPL određuje se razina privilegija izbornika. Na slici 4. prikazan je izbornik segmenta.



Slika 4. Izbornik segmenta [10]

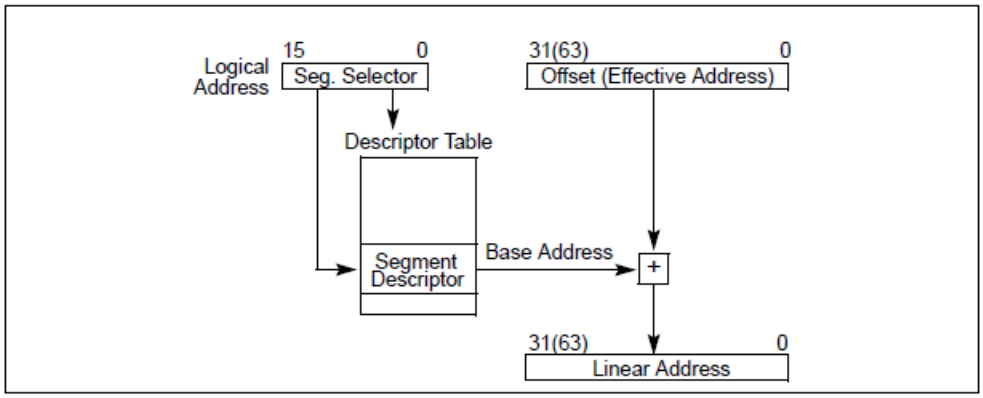
Segmentni opisnik je podatkovna struktura u tablicama opisnika GDT i LDT. Svaki segmentni opisnik je 8 bajtni. Opisnik opisuje karakteristike segmenta.



Slika 5. Opisnik segmenta [10]

Na slici 5. prikazan je opisnik segmenta. Polje opseg segmenta (engl. *segment limit*) označava veličinu segmenta. Početna adresa (engl. *base address*) je početak adrese segmenta u memoriji. Polje tip (engl. *type*) određuje tip segmenta. Sa zastavicom S određuje se tip opisnika. Razina privilegija opisnika DPL (engl. *descriptor privilege level*) određuje razinu privilegija segmenta. Polje P označava da li je segment prisutan u memoriji ili nije. Zastavica zrnatosti G (engl. *granularity*) određuje skaliranje polja opsega segmenta. Ako je zastavica G obrisana (G = 0) opseg segmenta je izražen u bajtovima, a ako je postavljena onda je opseg izražen u blokovima od 4KB. Detaljniji opis opisnika segmenta može se pronaći u literaturi [10] .

Na slici 6. prikazano je kako se prevodi logička adresa u linearnu adresu. Pomoću izbornika segmenta odabire se segmentni opisnik u tablici GDT ili LDT. Zatim se provjerava segmenti opisnik. Potrebno je ispitati pravo pristupa i raspon segmeneta. Linearna adresa se dobije dodavanjem početne adrese segmenta pomaku koji određuje adresu bajta u segmentu u odnosu na početnu adresu segmenta.



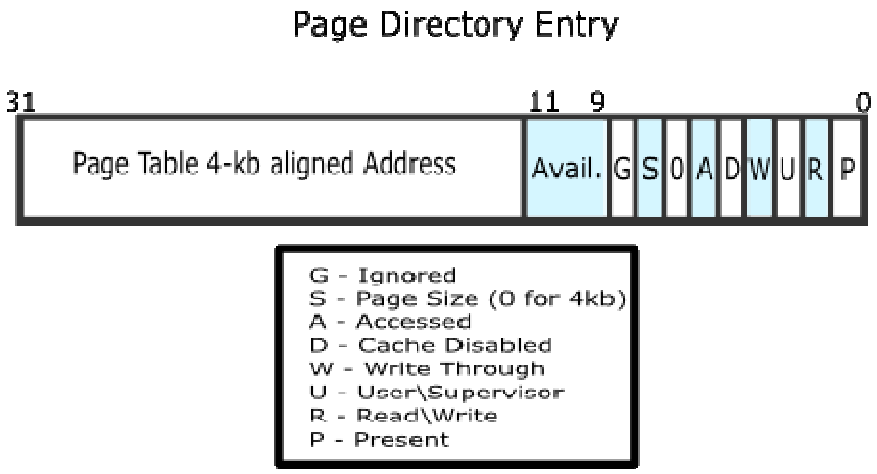
Slika 6. Prevođenje logičke adrese u linearnu adresu [10]

3.2.2. Straničenje

Virtualna memorija može se implementirati sa metodom segmentacije i sa metodom straničenja (engl. *paging*). Segmentacija postaje zastarjela metoda, dok je straničenje novija i bolja zamjena kod x86 arhitekture.

Kod straničenja se virtualni adresni prostor dijeli na blokove. Ovi blokovi nazivaju se stranicama (engl. *pages*) i obično su veličine 4 KB. Blokovi jednake veličine u fizičkoj memoriji nazivaju se okviri (engl. *frames*). Stranice se mapiraju u okvire.

Jedinica za upravljanje memorijom MMU (engl. *memory management unit*) pretvara virtualne adrese u fizičke adrese. Odnosno virtualna adresa se mapira u fizičku adresu. Kod x86 arhitekture MMU mapira memoriju kroz sljedeće dvije tablice: direktorij stranica (engl. *page directory*) i tablicu stranica (engl. *page table*). Direktorij stranice i tablica stranica sadrže po 1024 32-bitna ulaza (engl. *entries*). Svaki ulaz u direktorij stranica pokazuje na tablicu stranica, a svaki ulaz u tablicu stranica pokazuje na fizičku adresu.

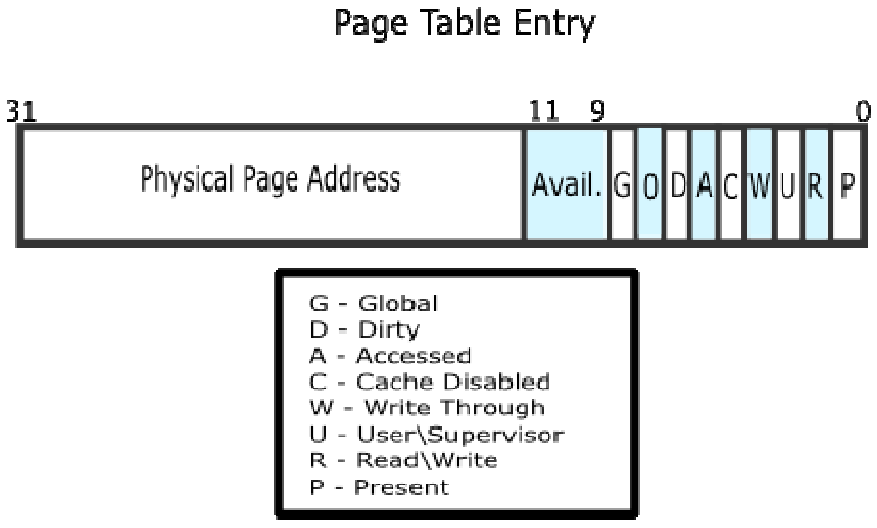


Slika 7. Ulaz direktorija stranica [55]

Slika 7. prikazuje ulaz direktorija stranica. U tablici 4. detaljnije su opisane zastavice i polja ulaza direktorija stranica.

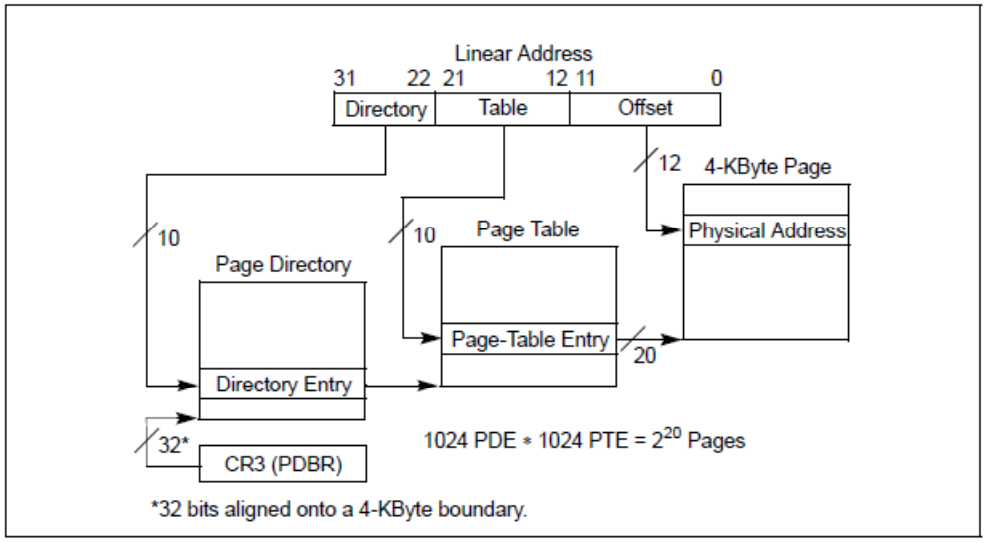
Tablica 4. Opis zastavica i polja kod ulaza direktorija stranica

Polje/Zastavica	Opis
P	P određuje da li je stranica u fizičkoj memoriji. Ako je zastavica postavljena stranica je u fizičkoj memoriji, a ako je obrisana stranica nije u fizičkoj memoriji.
R	R određuje privilegij čitanja-pisanja. Ako je zastavica postavljena stranica se može čitati i dopušteno je pisanje, a ako nije postavljena stranica se može samo čitati.
U	Zastavica U određuje korisnik-nadzornik (engl. *supervisor*) povlastice. Ako je zastavica postavljena stranici mogu pristupiti svi, a ako zastavica nije postavljena samo nadzornik može pristupiti.
W	W kontrolira pisanje kroz (engl. *write-thorugh*) sposobnosti stranice. Ako je bit postavljen omogućeno je pisanje kroz, a ako nije postavljen omogućen je upis natrag (engl. *write-back*).
A	A označava da li je stranici pristupano. Ako je zastavica postavljena, pristupano je.
D	D određuje spremanje u priručnu memoriju (engl. *cache*). Ako je bit postavljen stranica ili tablica stranice se neće spremiti, a ako je bit obrisan onda će se spremiti.
S	Sa S određuje se veličina stranice. Ako je bit postavljen stranice su veličine 4 MB, a ako nije postavljen stranice su veličine 4 KB.
adresa tablice stranica	Adresa tablice stranica je fizička adresa tablice stranica.



Slika 8. Ulaz tablice stranica [55]

Slika 8. prikazuje ulaz tablice stranica. Većina bitova odgovara prethodnom opisu za ulaz direktorija stranica. Procesor postavlja zastavicu D (engl. *dirty*) ako je pisano u stranicu.



Slika 9. Prevođenje linearne adrese u fizičku adresu [10]

Slika 9. prikazuje prevođenje linerne u fizičku adresu adresu. Upravljački registar CR3 sadrži fizičku adresu direktorija stranica. Linerna adresa dijeli se na tri dijela: ulaz direktorija stranica (od bita 22 do bita 31), ulaz tablice stranica (od bita 12 do bita 21) i pomak stranice (engl. *page offset*) (od bita 0 do bita 1). Dio ulaz direktorija stranica pruža pomak na ulaz direktorija stranica. Taj odabrani ulaz pruža fizičku adresu odgovarajuće tablice stranica. Dio ulaz tablice stranica pruža pomak na ulaz u odabranoj tablici stranica, a taj ulaz pruža početnu fizičku adresu stranice u fizičkoj memoriji. Pomak stranice osigurava pomak na fizičku adresu u stranici.

3.3. Tablice opisnika

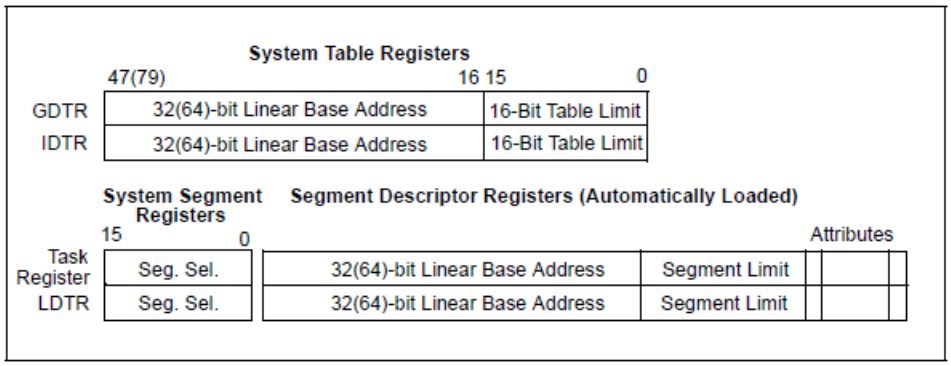
U ovom dijelu opisane su dvije tablice koje su svojstvene za x86 arhitekturu. Opća tablica opisnika sastoji se od opisnika segmenata koji govore procesoru o memorijskim segmentima. Tablica opisnika prekida ukazuje gdje se nalaze prekidne rutine ISR (engl. *Interrupt Service Routines*).

3.3.1. Opća tablica opisnika - GDT

Opća tablica opisnika GDT (engl. *Global Descriptor Table*) je podatkovna struktura u linarnom adresnom prostoru. Bitno je napomenuti da GDT nije segment.

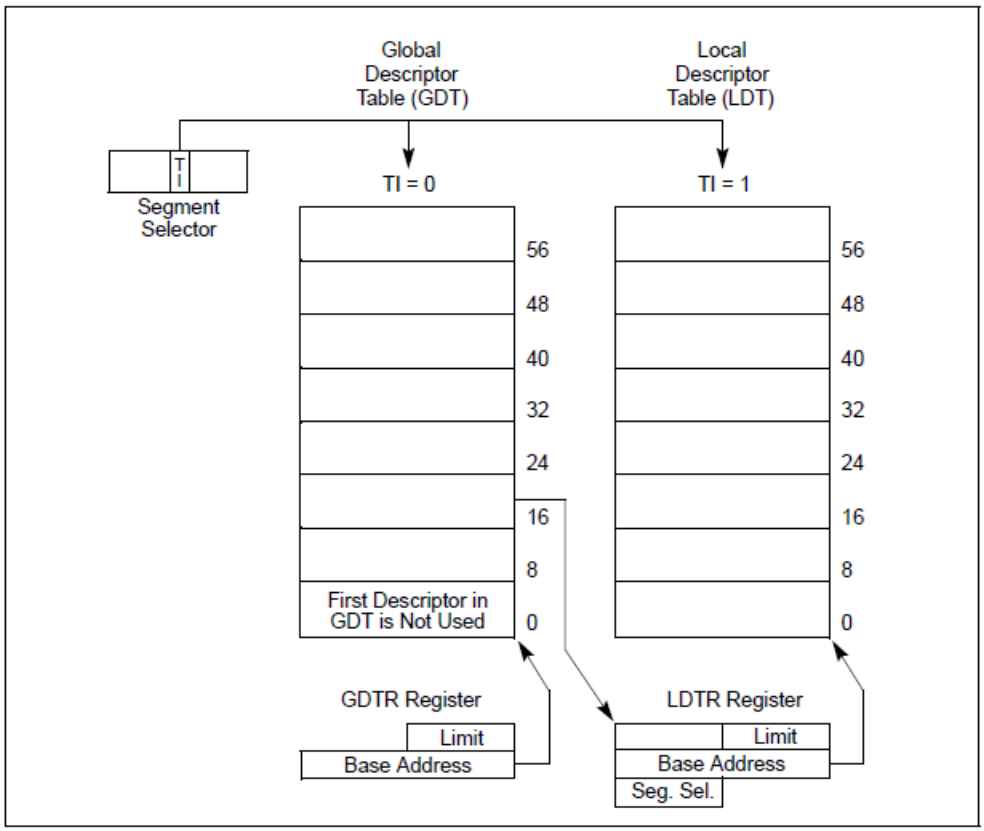
GDT tablica sastoji se od ulaza (engl. *entries*) koji se nazivaju opisnicima segmenata. Veličina GDT tablice je varijabilna i može imati maksimalno 8192 opisnika segmenata. Opisnici segmenata definiraju svojstva segmeneta. Svaki opisnik segmenta je duljine 8 bajta i svaki opisnik ima odgovarajući izbornik segmenta. Na str. 7. opisan je i prikazan opisnik segmenta.

GDTR registar sadrži adresu i veličinu GDT tablice. Radi se 48 bitnom registru u kojem je nižih 16 bitova namijenjeno veličini tablice u bajtima, a viših 32 bita adresi tablice u memoriji. Na sljedećoj slici prikazan je GDTR registar.



Slika 10. Registri GDTR, IDTR, TR i LDTR [10]

Naredba LGDT puni GDTR registar sa GDT adresom i veličinom iz memorije. Spremanje GDT adrese i veličine iz GDTR registra u memoriju obavlja naredba SGDT.



Slika 11. Tablica opisnika GDT i LDT [10]

Prvi opisnik segmenta u GDT-u je nulti opisnik (engl. *null descriptor*). Ovaj opisnik sastoji se od nula. Nulti opisnik ne koristi procesor. Ako se pokuša pristupiti memoriji koristeći ovaj opisnik javlja se iznimka općenite zaštite (engl. *general-protection exception*).

Pristup memorijskim segmentima obavlja se preko tablice opisnika (GDT ili LDT). Izbornik segmenta ovisno o zastavici TI određuje koja se tablica koristi GDT ili LDT. Ako je TI nula odabire se GDT, a ako je jedan odabire se LDT. Izbornik segmenta sadrži indeks kojim se odabire jedan od 8192 opisnika u odabranoj tablici. Na slici 11. prikazana je globalna i lokalna tablica opisnika.

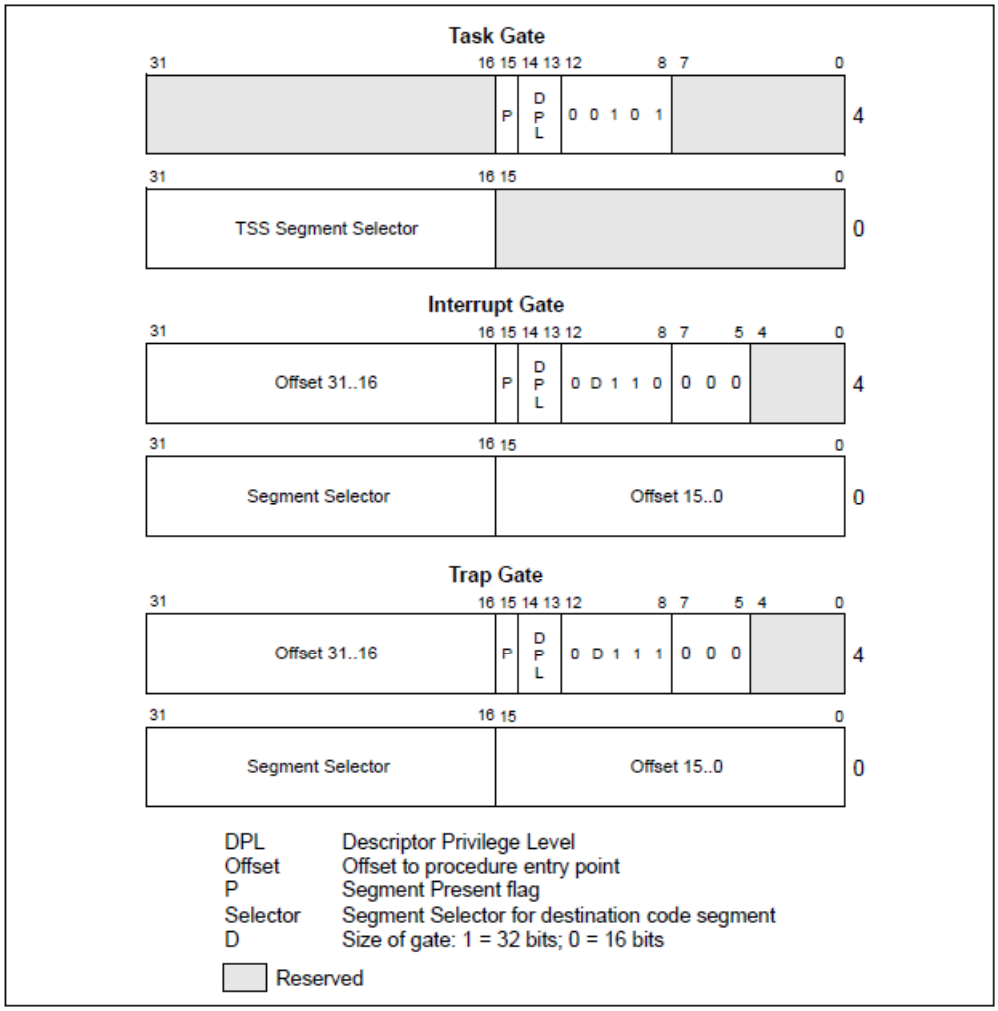
Kod postavljanja GDT tablice potrebno je:

- izgraditi tablicu koja se sastoji od opisnika segmenata,
- sa lgdt instrukcijom reći procesoru gdje se nalazi tablica,
- ponovo učitati segmente registre.

3.3.2. Tablica opisnika prekida - IDT

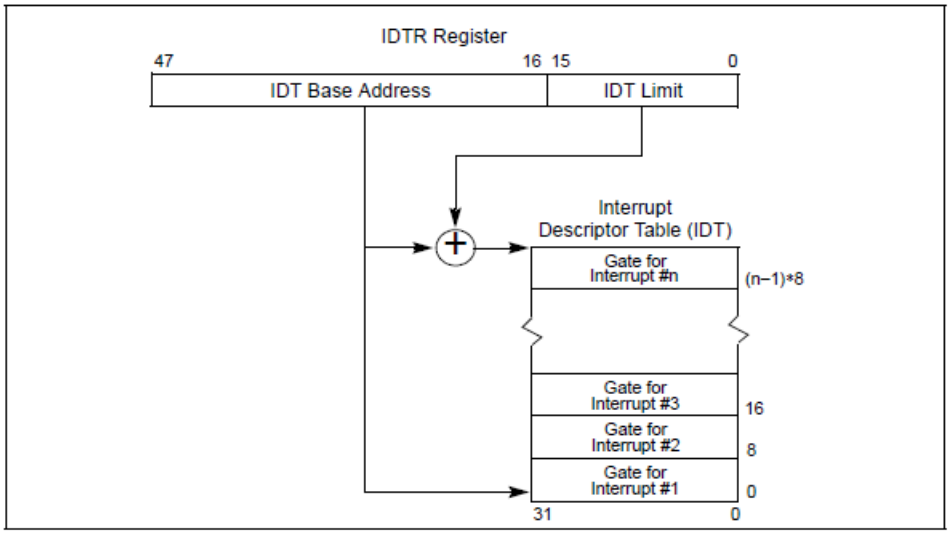
Tablica opisnika prekida IDT (engl. *Interrupt Descriptor Table*) je polje 8-bajtnih opisnika. IDT može sadržavati maksimalno 256 opisnika. Upravo je toliki i broj vektora za prekide i iznimke.

Postoje tri tipa opisnika vrata (engl. *gate descriptors*) koje IDT može sadržavati: vrata zadataka (engl. *task gate*), vrata prekida (engl. *interrupt gate*) i vrata zamke (engl. *trap gates*). Na sljedećoj slici prikazani su formati IDT opisnika vrata.



Slika 12. Formati IDT opisnika vrata [10]

Osnovne podatke o tablici IDT sadrži procesorski registar IDTR. Radi se o registru koji sadrži 32 bitnu adresu i 16 bitnu veličinu u bajtima IDT tablice. Pomoću ovoga registra procesor zna gdje je IDT.



Slika 13. Veza registra IDTR i tablice IDT [10]

Naredba LIDT učitava IDT adresu i veličinu iz memorije u IDTR registar, a naredba SIDT sprema IDT adresu i veličinu iz IDTR registra u memoriju.

Procesor pomoću tablice IDT može saznati gdje se nalaze rukovatelji (engl. *handlers*) prekida i iznimki. Vektor prekida i iznimke koristi se kao indeks u IDT. Kada se dogodi prekid procesor će na temelju toga indeksa umnoženog sa 8 i osnovne adrese tablice iz registra IDTR saznati osnovnu adresu opisnika. Ako je dobivena adresa prevelika, odnosno ako nije unutar tablice, procesor će generirati iznimku.

3.4. Prekidi i iznimke

Prekidi prekidaju upravljački tok procesora (prekidaju CPU). Prema izvoru prekidi mogu biti vanjski (sklopovski) i programski.

3.4.1. Sklopovski prekidi

Sklopovski prekidi (engl. *hardware interrupts*) su oni prekidi koje generiraju sklopovski uređaji. Za upravljene sklopovskim prekidima koristi se kontroler prekida (PIC, APIC). Kontroler prekida prihvaća od uređaja zahtjeve za prekidom. Za zahtjev najvećeg prioriteta kontroler prekida će objaviti prekid procesoru.

Postoje dvije vrste sklopovskih prekida:

- maskirani prekidi (engl. *maskable interrupts*) i
- nemaskirani prekidi NMI (engl. *non-maskable interrupts*).

Maskirani prekidi se mogu maskirati. Brisanjem zastavice IF registra EFLAGS onemogućavaju se ovi prekidi. Za onemogućavanje prekida koriti se naredba *cli*. Prekidi se mogu omogućiti naredbom *sti*.

Nemaskirani prekidi se ne mogu maskirati (sa IF zastavicom). NMI je često rezultat težih sklopovskih problema.

Tablica 5. Sklopovski prekidi

Prekidni broj	IRQ	Opis
0x08	IRQ 0	Brojilo (PIT)
0x09	IRQ 1	Tipkovnica
0x0A	IRQ 2	Kaskada sa 8259A pomoćnim kontrolerom
0x0B	IRQ 3	Serijski port 2
0x0C	IRQ 4	Serijski port 1
0x0D	IRQ 5	Kod AT sustava paralelni port 2, a kod PS/2 sustava rezervirano
0x0E	IRQ 6	Disketni uređaj
0x0F	IRQ 7	Paralelni port 1
0x70	IRQ 8	RTC
0x71	IRQ 9	CGA
0x72	IRQ 10	Rezervirano
0x73	IRQ 11	Rezervirano
0x74	IRQ 12	Kod AT sustava rezervirano, a kod PS/2 sustava pomoćni uređaj
0x75	IRQ 13	FPU
0x76	IRQ 14	Tvrdi disk
0x77	IRQ 15	Rezervirano

3.4.2. Programski prekidi

Programski prekidi (engl. *software interrupts*) generiraju se izvođenjem određene instrukcije u programu. Instrukcije *INT*, *INT 3*, *INTO* i *BOUND* generiraju ove prekide.

3.4.3. Iznimke

Iznimke (engl. *exceptions*) se javljaju ako procesor tijekom izvođenja programa uoči određene greške. Na primjer dijeljenje sa nulom uzrokovat će iznimku.

3.5. Programirljivi kontroler prekida - PIC

Sklopovski prekidi su oni prekidi koji su generirani sklopovljem. U ovom dijelu opisano je upravljanje sklopovskim prekidima.

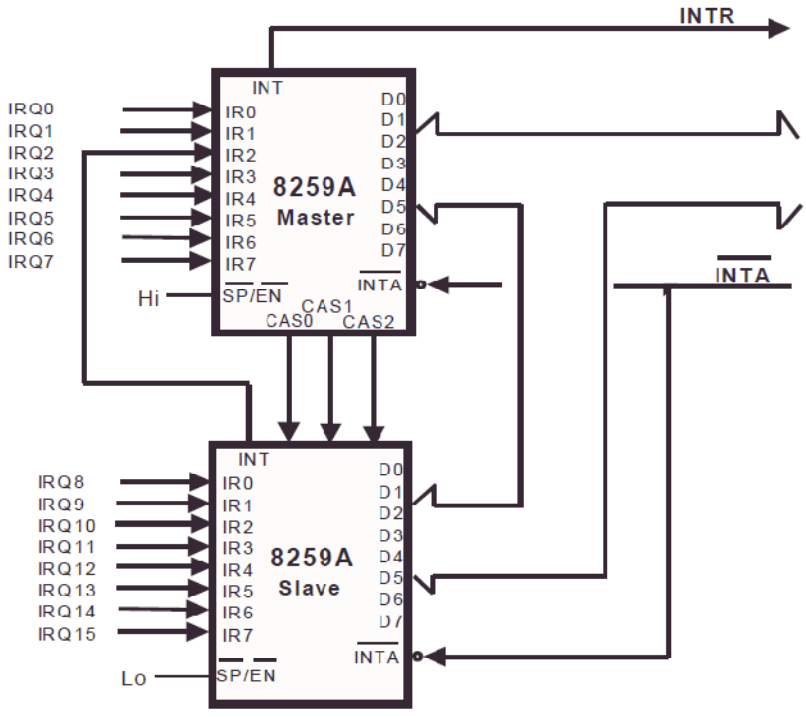
PIC (engl. *Programmable Interrupt Controller*) upravlja sklopovskim prekidnim zahtjevima. Jedan od najpoznatijih kontrolera prekida je 8259A.

Kontroler prekida PIC prihvaća prekidne zahtjeve sa vanjskih uređaja (npr. tipkovnice) i javlja to procesoru (CPU). Procesor prvo dovršava instrukciju koju trenutni izvodi. Zatim procesor dohvaća rutinu za posluživanje uređaja. Nakon izvršavanja rutine, procesor će nastaviti raditi ono što je radio prije prekidnog zahtjeva.

Stara računala (IBM PC i XT) koristila su samo jedan PIC. IBM PC/AT računala imaju dva kontrolera prekida koja mogu međusobno komunicirati. Korištenjem dva PIC-a moguće je posluživati 15 prekidnih uređaja.

Svaki PIC ima po 8 ulaza. Glavni PIC (engl. *Master PIC*) barata sa linijama IRQ0 – IRQ7 (engl. *Interrupt Request*), a pomoćni PIC (engl. *Slave PIC*) sa linijama IRQ8 – IRQ15. U tablici 5. na str. 15. može se vidjeti na temelju linija IRQ za koje uređaje je zadužen glavni, a za koje pomoćni PIC.

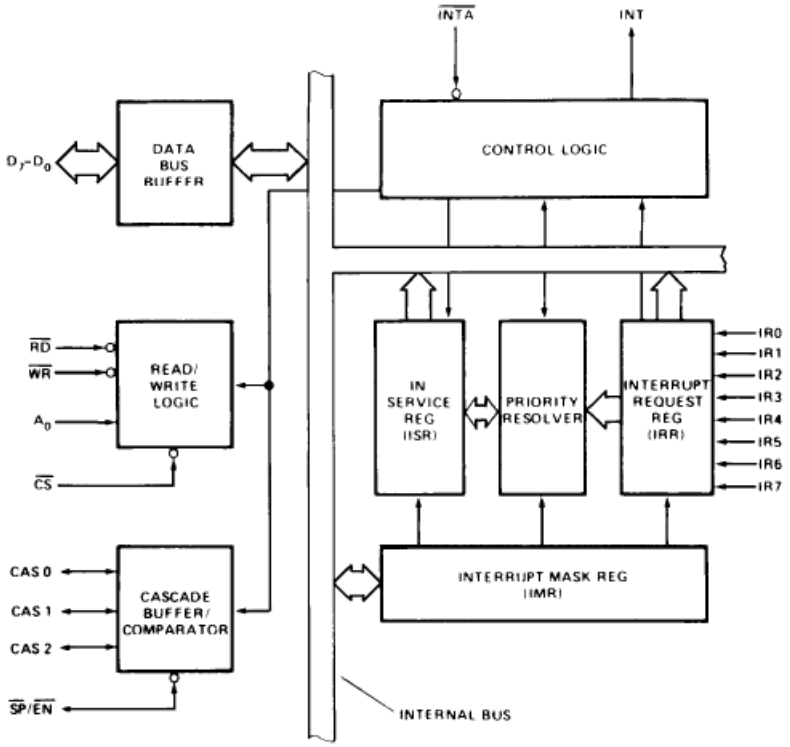
Pomoćni PIC povezan je sa glavnim PIC-om. Linija IRQ2 koristi se za povezivanje pomoćnog i glavnog PIC-a. Glavni PIC povezan je direktno sa procesorom.



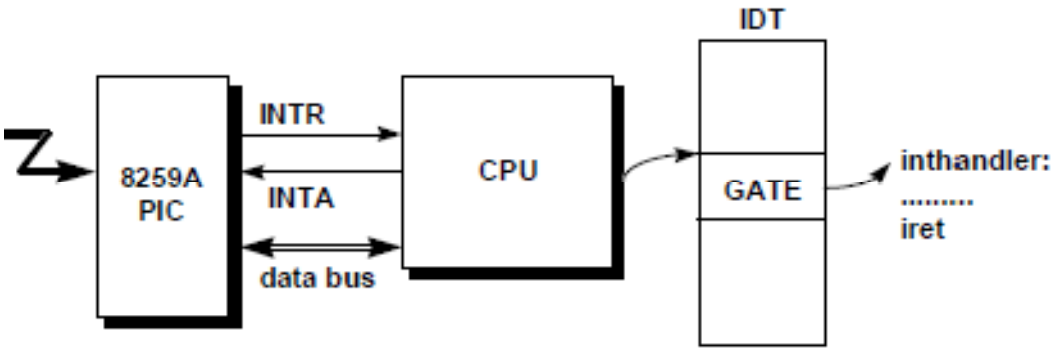
Slika 14. Kaskada dva 8259A kontrolera prekida [59]

Kada neki uređaj treba biti poslužen javlja to kontroleru prekida PIC. Uređaj signalizira PIC-u preko odgovarajuće linije za zahtjev prekida (IR). Postavlja se odgovarajući bit u registru IRR (engl. *Interrupt Request Register*). Ako je zahtjev odgovarajući PIC će javiti procesoru aktiviranjem linije INT. CPU će završiti instrukciju koju trenutno izvodi. Zatim će ispitati zastavicu IF u registru RFLAGS. Ako prekidi nisu maskirani CPU će poslati PIC-u impuls priznavanja prekida (INTA). PIC će primiti INTA impuls. Nakon toga postavit će bit najvećeg prioriteta u registru ISR (engl. *In-Service Register*) i resetirati odgovarajući IIR bit. CPU će zatim inicirati drugi impuls, a tijekom impulsa PIC će postaviti 8 bitni broj (broj vektora prekida) na podatkovnu sabirnicu. Na temelju pročitanog broja procesor će odrediti adresu prekide rutine ISR. Za određivanje adrese rutine koristi se tablica IVT (engl. *Interrupt Vector Table*) kod stvarnog načina rada (engl. *real mode*), a kod zaštićenog načina rada (engl. *protected mode*) koristi se tablica IDT. Prekidna rutina se zatim izvršava.

Na slici 15. prikazan je blok dijagram kontrolera prekida, a na slici 16. prikazan je prekidni mehanizam.



Slika 15. 8259A blok dijagram [42]



Slika 16. Prekidni mehanizam [61]

IRQ 0-7 se originalno mapira u IDT ulaze 0x8 – 0xF, a IRQ 8-15 se mapira u IDT ulaze 0x70 – 0x78. Kod zaštićenog načina rada postoji konflikt između IRQ 0-7 i procesorskih iznimaka, jer su IDT ulazi od 0 do 31 rezervirani za iznimke. Preporučava se napraviti ponovno postavljanje PIC-ova (engl. *remapping PICs*). Pri ponovnom postavljanju mijenjaju se pomaci vektora (engl. *vector offset*) PIC-ova tako da IRQ-ovi koriste nerezervirane vektore. Pomaci vektora koriste se za dobivanje broja prekida. Uobičajeno se pomak glavnog PIC-a postavlja na 0x20, a pomak pomoćnog PIC-a na 0x28.

Moderna računala podržavaju noviju verziju kontrolera prekida - APIC (engl. *Advanced Programmable Interrupt Controller*).

3.6. Brojilo sustava - PIT

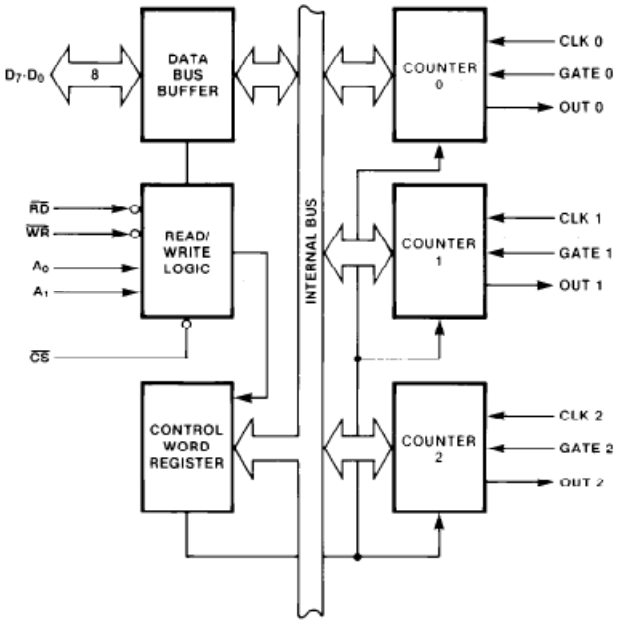
PIT (engl. *Programmable Interval Timer*) je čip koji je koristan za točno generiranje prekida u redovitim vremenskim razmacima. Ovaj čip još se naziva 8252/8254 čip. Moderna računala ne sadrže više PIT kao zasebni čip, već se nalazi integriran na matičnoj ploči. PIT se može koristiti kod implementacije višezadaćnosti. Ideja je da se prilikom prekida izvrši zamjena konteksta procesa.

PIT ima tri kanala, a to su kanal 0, kanal 1 i kanal 2. Svaki kanal ima vlastiti izlazni signal. Izlazni signali kanala imaju različite svrhe.

Kanal 0 je preko svoga izlaza povezan sa IR0 linijom čipa PIC. Generirani prekidi će imati najveći prioritet. Zadana (uobičajena) izlazna frekvencija ovoga kanala je 18.2065 HZ, što znači da će IRQ0 opaliti svakih 54.9254 ms. Frekvencija se može podesiti slanjem djelitelja PIT-u. Vrijednost djelitelja se može dobiti dijeljenjem ulazne frekvenciju (1193180 HZ) i tražene frekvencije. Kanal 0 koristi se implementaciju sata sustava (engl. *System Clock*).

Kanal 1 više se ne upotrebljava. Prije se koristio kod osvježavanja DRAM memorije.

Kanal 2 je preko svoga izlaza povezan sa PC zvučnikom (engl. *PC Speaker*). Koristi se za generiranje tona (engl. *beep*).



Slika 17. 8254 blok dijagram [37]

U sljedećoj tablici nalaze se portovi koje koristi PIT čip.

Tablica 6. Portovi koje koristi PIT čip

Port	Čitanje/Pisanje	Uporaba
0x40	čitanje/pisanje	kanal 0 podatkovni port
0x41	čitanje/pisanje	kanal 1 podatkovni port
0x42	čitanje/pisanje	kanal 2 podatkovni port
0x43	pisanje	registar načina/naredbe

Na slici 18. prikazan je format kontrolnog bajta, a u tablici 7. nalazi se opis. Bitovi 6 i 7 služe za odabir kanala koji se konfigurira (npr. 00_2 = kanal 0, 01_2 = kanal 1). Bitovima 4 i 5 može se odabrati način pristupa (engl. *access mode*) odabranog kanala (npr. 01_2 je za čitanje ili pisanje najmanje značajnog bajta LSB). Bitovima 1-3 određuje se način rada odabranog kanala (npr. 000_2 = način rada 0). Sa bitom 0 odabire se da li će kanal raditi u BCD ili binarnom načinu.

7 - 6	5 - 4	3 - 1	0
odabir kanala	način pristupa	način rada	BCD/bin

Slika 18. Format naredbenog (kontrolnog) bajta

Tablica 7. Opis polja naredbenog bajta

Polje	Opis
Odabir kanala	00 = kanal 0 01 = kanal 1 10 = kanal 2 11 = naredba (read-back)
Način pristupa	00 = naredba (latch count value) 01 = čitanje/pisanje LSB 10 = čitanje/pisanje MSB 11 = čitanje/pisanje prvo LSB pa MSB
Način rada	000 = način rada 0 001 = način rada 1 010 = način rada 2 011 = način rada 3 100 = način rada 4 101 = način rada 5 110 = način rada 2 111 = način rada 3
BCD/binarni način	0 = bin 1 = BCD

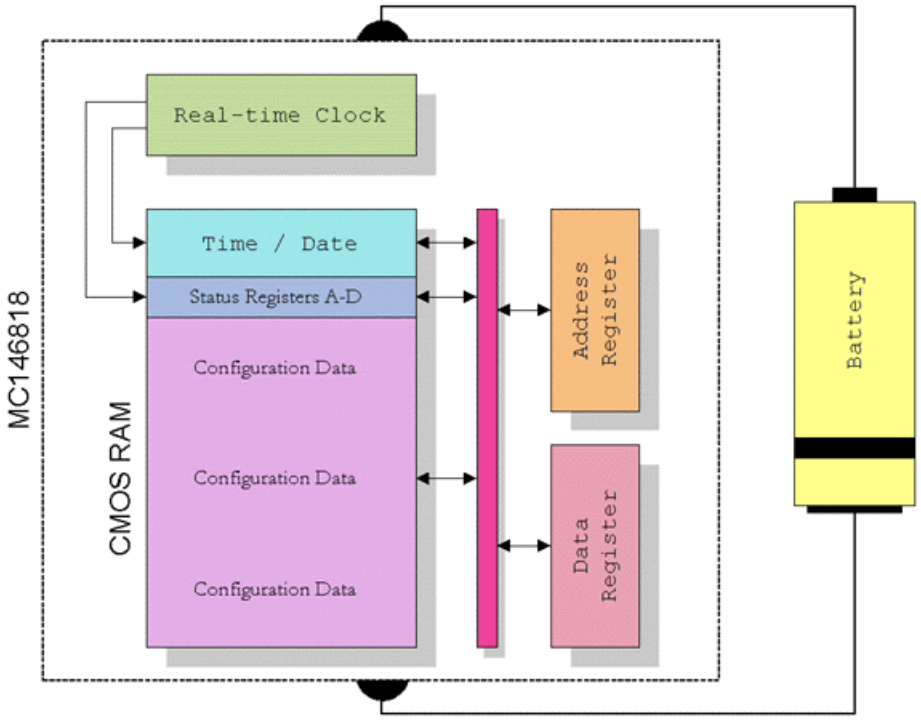
Svaki kanal podržava 6 načina rada. Ovdje neće biti opisani načini rada kanala. Opis načina rada kanala može se pronaći u literaturi [36].

3.7. Sat stvarnog vremena – RTC

Kako bi računala imala podatak o točnom vremenu i datumu važnu ulogu ima čip koji sadrži RTC (engl. *Real Time Clock*) i postojani RAM (engl. *non-volatile RAM*). Nazivi koji su karakteristični za ovaj čip su RTC/NRAM čip, RTC čip, CMOS čip i CMOS RAM čip .

RTC ima zadaću da sačuva trenutno vrijeme i datum. Postojana memorija služi za spremanje osnovne konfiguracije sustava. U memoriji su sadržane informacije o tipu tvrdog diska, količini memorije i druge informacije. Memorija se koristi za spremanje BIOS postavki.

Kako bi se sačuvale informacije u postojanoj memoriji i nastavilo RTC otkucavanje kada je računalo ugašeno koristi se mala baterija. Baterija se obično nalazi na matičnoj ploči. Na sljedećoj slici prikaza je struktura originalnog čipa ovoga tipa (Motorola MC146818).



Slika 19. MC146818 struktura [33]

Za direktan pristup RTC/CMOS čipu koriste se I/O portovi 0x70 i 0x71. Kod pristupa unutrašnjim registrima čipa potrebno je postaviti indeks na port 0x70, a zatim obaviti čitanje sa ili pisanje na port 0x71.

Tablica 8. Portovi za pristup RTC/CMOS čipu

Port	Opis
0x70	indeksni port – za specifikaciju indeksa
0x71	podatkovni port – za čitanje i pisanje bajta u memoriju

RTC registri mogu se podijeliti u tri grupe: sat/kalendar, statusni registri i konfiguracijski podaci. Registri grupe sat/kalandar se ažuriraju pomoću sata. Statusni registri utječu na rad samog čipa.

Tablica 9. RTC registri

Adresa	Funkcija
00	trenutna sekunda
01	sekunda alarma
02	trenutna minuta
03	minuta alarma
04	trenutni sat
05	sat alarma
06	trenutni dan u tjednu
07	trenutni dan u mjesecu
08	trenutni mjesec
09	trenutna godina
0A	statusni registar A
0B	statusni registar B
0C	statusni registar C
0D	statusni registar D
0E	POST dijagnostički statusni bajt
0F	statusni bajt isključivanja
10	vrsta disketnog uređaja
11	rezervirano
12	vrsta tvrdog diska
13	rezervirano
14	bajt opreme
15	niži bajt osnovne memorije
16	viši bajt osnovna memorija
17	proširene memorija u kilobajtima (niži bajt)
18	proširene memorija u kilobajtima (viši bajt)
19	disk 0 tip ako (CMOS adresa 12h & 0fh) je 0fh
1A	disk 1 tip ako (CMOS adresa 12h & f0h) je f0h
1B - 2D	rezervirano
2E	ispitni zbroj CMOS adresa 10h-20h (MSB)
2F	ispitni zbroj CMOS adresa 10h-20h (LSB)
30	LSB veličina proširene memorije
31	MSB veličina proširene memorije
32	stoljeće (BCD)
33	razne zastavice
34 - 3F	rezervirano

RTC čip može generirati tri tipa prekida:

- periodički prekid (frekvencija od 2 Hz do 8192 Hz),
- alarmni prekid,
- prekid ažuriranja.

Frekvencija periodičkog prekida može se podesiti u statusnom registru A. Alarmni prekid generira se u zakazanom vremenu, a prekid ažuriranja generira se svake sekunde poslije ažuriranja sata. Za korištenje alarmnih prekida i prekida ažuriranja potrebno je prvo podesiti određene bitove u statusnom registru B.

Čitanje vremena i datuma sa RTC-a je jako sporo. Kako bi se zaobišla sporost čitanja može se samo jednom dohvatiti vrijeme i datum sa RTC čipa, a za praćenje vremena može se koristit brojilo (npr. PIT).

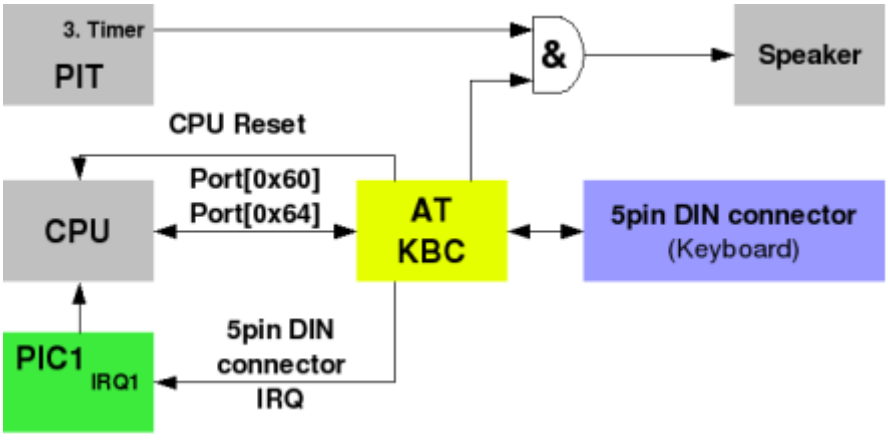
3.8. Tipkovnica

Kod izgradnje operacijskog sustava potrebno je napraviti upravljački programi za tipkovnicu. Pri tome je dobro poznavati princip rada tipkovnice.

Koder tipkovnice skenira stanja tipki. Ukoliko ustanovi da je neka tipka promijenila stanje šalje podatkovni paket kontroleru tipkovnice. Odnosno ako je neka tipka pritisnuta, otpuštena ili zadržana dolje koder će poslati paket kontroleru tipkovnice. Kontroler tipkovnice se obično nalazi na matičnoj ploči. Radi se o komunikaciji između kontrolera koji se nalazi unutar tipkovnice (koder tipkovnice) i kontrolera tipkovnice na matičnoj ploči.

Podatkovni paket koji se šalje kontroleru tipkovnice naziva se skenirajući kod (engl. *scan code*). Koder tipkovnice saznaje koji skenirajući kod odgovara određenoj tipci pomoću mape znakova koja se nalazi u njegovoj ROM memorije.

Kada kontroler tipkovnice primi skenirajući kod sprema ga u ulazni spremnik i aktivira linju za zahtjev prekida IRQ1. PIC prihvaća zahtjev i javlja procesoru.

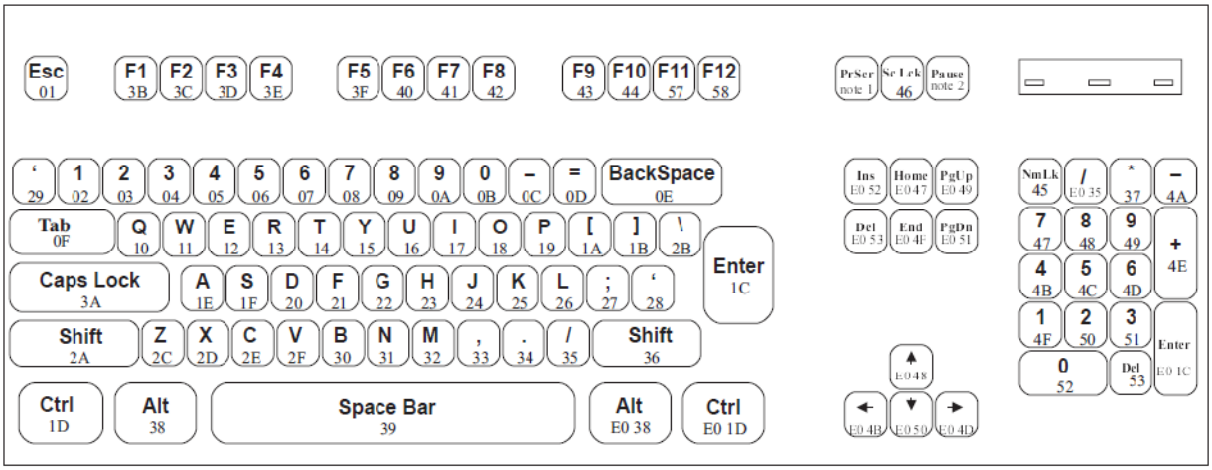


Slika 20. Prikaz AT kontrolera tipkovnice (AT-KBC) [15]

Postoje dvije vrste skenirajućih kodova, a to su kod pritiskanja (engl. *make code*) i kod otpuštanja (engl. *break code*). Kod pritiskanja se šalje kada je tipka pritisnuta ili zadržana dolje, a kod otpuštanja se šalje kada je tipka otpuštena. Svaka tipka ima svoj jedinstveni kod pritiskanja i

otpuštanja. Ako se neka tipka pritisne i ostavi tako zadržana dolje, tada se kod pritiskanja te tipke šalje računalu sve dok se tipka ne otpusti ili se neka druga tipka stisne.

Postoje različite grupe skenirajućih kodova. Standardne grupe su: grupa 1 skenirajućih kodova (XT skenirajući kodovi), grupa 2 skenirajućih kodova (AT skenirajući kodovi) i grupa 3 skenirajućih kodova (PS/2 skenirajući kodovi). Skenirajuće kodove grupe 1 koristila su starija računala. Moderne tipkovnice obično koriste grupu 2. Kako bi se sačuvala kompatibilnost sa ranijim programima kontroler tipkovnice pretvara skenirajuće kodove grupe 2 u skenirajuće kodove grupe 1. Skenirajući kodovi grupe 3 se rijetko koristi. Kod programiranja niske razine (engl. *low level*) obično se koriste skenirajući kodovi grupe 1.



Notes:
1. Print Screen Key has the scan code E0 2A E0 37.
2. Pause Key has the scan code E1 1D 45 E1 9D C5.
3. All the keys that have upper and lower cases are shown with lower case characters

Slika 21. XT heksadecimalni kodovi pritiskanja [21]

8042 kontroler tipkovnice sadrži sljedeće registere: ulazni spremnik, izlazni spremnik, statusni registar i upravljački registar.

Tablica 10. Portovi kontrolera tipkovnice

Port	Čitanje/Pisanje	Opis
0x60	čitanje	čitanje ulaznog spremnika
0x60	pisanje	pisanje u izlazni spremnik
0x64	čitanje	čitanje statusnog registra
0x64	pisanje	slanje naredbi

U tablici 10. nalaze se portovi kontrolera tipkovnice. Skenirajući kod može se pročitati sa I/O porta 0x60. Statusni bajt kontrolera tipkovnice čita se sa porta 0x64. Pisanjem naredbenog bajta na port 0x64 šalje se naredba. Popis naredbi može se pronaći u literaturi [5].

4. Osnovne rutine za izgradnju operacijskog sustava

Tablica 11. Rutine za izgradnju operacijskog sustava

Funkcija	Opis funkcije
halt	zaustavlja procesor
nop	ne obavlja nikakvu operaciju (služi za trošenje vremena)
stop_int	onemogućava prekide
start_int	omogućava prekide
outb	zapisuje bajt na port
inb	čita bajt sa porta
outw	zapisuje 2 bajta na port
inw	čita 2 bajta sa porta
kputch	ispisuje jedan znak na zaslon
kputs	ispisuje niz znakova
kputi	ispisuje cijeli broj
kprintf	formatirani ispis podataka
set_fgcolor	postavlja boju samog znaka
set_bgcolor	postavlja boju pozadine znaka
set_cursor	sprema položaj kursora
get_cursor	dohvaća položaj kursora
kscroll	pomiče sadržaj zaslona za jednu liniju
kclear	briše sadržaj zaslona
gdt_install	postavlja tablicu GDT
PIC_remap	ponovno postavlja PIC-ove
idt_install	postavlja tablicu IDT i postavlja ulaze tablice za iznimke
irq_install	postavlja ulaze tablice IDT za sklopovske prekide
set_irq_handler	postavlja rukovatelja prekida
get_scancode	dohvaća skenirajući kod
update_led	ažurira lampice tipkovnice
kb_update	ažurira stanja određenih tipki (npr. shift) i lampica tipkovnice
kgetch	dohvaća ASCII vrijednost za neki skenirajući kod
modulate_pit_freq	postavlja frekvenciju brojila PIT
get_RTCtime	dohvaća vrijeme sa RTC čipa
set_RTCtime	postavlja vrijeme
get_RTCdate	dohvaća datum
set_RTCdate	postavlja datum
reboot	resetira i ponovo pokreće sustav

Kako bi se olakšala izgradnja jednostavnih operacijskih sustava ostvarene su rutine. U tablici 11. nalazi se popis rutina. U nastavku je ukratko opisana svaka rutina.

4.1. Zaustavljanje procesora

Funkcija *halt* zaustavlja procesor. Pozivom ove funkcije procesor se postavlja u stanje zaustavljanja (engl. *halt state*). U stanju zaustavljanja procesor će biti dok se ponovo ne pokrene prekidom ili resetiranjem. Funkcija *halt* temelji se na privilegiranoj asemblerskoj instrukciji *hlt*, prema tome i funkcija *halt* je privilegirana.

Deklaracija funkcije halt:

```
void halt();
```

4.2. NOP

Funkcija *nop* ne obavlja nikakvu operaciju. Može se koristiti za trošenje vremena. Pozivom funkcije *nop* troše se instrukcijski ciklusi u procesoru.

Deklaracija funkcije nop:

```
void nop();
```

4.3. Omogućavanje i onemogućavanje prekida

Funkcija *stop_int* onemogućava prekide. Pozivom funkcije *stop_int* briše se prekidna zastavica IF (u registru FLAGS), odnosno zastavica se postavlja na 0. Postavljanjem zastavice na nulu prekidi su onemogućeni. Ovom funkcijom onemogućavaju se samo maskirajući sklopovski prekidi.

Deklaracija funkcije stop_int:

```
void stop_int();
```

Funkcija *start_int* omogućava prekide. Pozivom funkcije *start_int* postavlja se zastavica IF (u registru FLAGS). Ovom funkcijom omogućavaju se maskirajući sklopovski prekidi.

Deklaracija funkcije start_int:

```
void start_int();
```

4.4. I/O funkcije

Često su potrebne funkcije za slanje podataka na neki uređaj i za primanje podataka sa nekog uređaja (npr. tipkovnice). Funkcije *outb* i *outw* služe za pisanje na I/O portove, a funkcije *inb* i *inw* za čitanje sa I/O portova.

Funkcija *outb* zapisuje bajt na port. Prvi argument funkcije je port, a drugi argument je podatak koji se zapisuje.

Deklaracija funkcije outb:

```
void outb(unsigned short port, char data);
```

Funkcija *inb* čita jedan bajt sa porta. Kao argument funkcije potrebno je navesti port. Povratna vrijednost funkcije je pročitani podatak.

Deklaracija funkcije intb:

```
unsigned char inb(unsigned short port);
```

Funkcije *outw* zapisuje 2 bajta na port. Prvi argument funkcija je port, a drugi argument je podatak koji se zapisuje.

Deklaracija funkcije outw:

```
void outw(unsigned short port, unsigned short data);
```

Funkcija *inw* čita 2 bajta sa porta. Kao argument funkcije potrebno je navesti port. Povratna vrijednost je pročitani podatak.

Deklaracija funkcije inw:

```
int inw(unsigned short port);
```

4.5. Funkcije za rad sa zaslonom

U ovom dijelu opisane su funkcije za ispis znakova na zaslon, postavljanje boje ispisa, brisanje sadržaja zaslona i funkcija za pomicanje sadržaja zaslona.

4.5.1. Ispis na zaslon

Za ispis na zaslon na raspolaganju su sljedeće rutine: *kputch*, *kputs*, *kputi* i *kprintf*.

Tablica 12. Funkcija za ispis na zaslon

Funkcije za ispis	Opis
kputch	ispis jednog znaka
kputi	ispis cijelog broja
kputs	ispis niza znakova
kprintf	formatirani ispis

Za upravljanje ispisom dopuštene su sljedeći prekidni znakovi: \n, \b, \r i \t. Prekidni znakovi mogu se koristiti u rutinama *kputch*, *kputi* i *kputs*. Dakle, u svim rutinama za ispis osim rutine *kputi*.

Tablica 13. Prekidni nizovi

Prekidni niz	Opis
\n	prijelaz u novi red
\b	pomak unatrag za jedno mjesto
\r	pomak na početak reda
\t	horizontalni tab

Funkcija *kputch* ispisuje jedan znak na zaslon. Argument funkcije je znak koji se ispisuje na zaslon. Sve ostale funkcije za ispis znakova temelje se na ovoj funkciji, odnosno ostvarene su pomoću funkcije *kputch*.

Deklaracija funkcije kputch:

```
void kputch(char c);
```

Funkcija *kputs* služi za ispis niza znakova. Argument funkcije je niz znakova koji se ispisuju na zaslon.

Deklaracija funkcije kputs:

```
void kputs(char *s);
```

Cijele brojeve ispisuje funkcija *kputi*. Kao argument funkcija prima cijeli broj kojeg treba ispisati.

Deklaracija funkcije kputi:

```
void kputi(int dec);
```

Funkcija *kprintf* služi za formatirani ispis podataka. Broj argumenata funkcije je varijabilan. Prvi argument je kontrolni znakovni niz kojim se može odrediti način ispisa ostalih argumenata funkcije.

Deklaracija funkcije kprintf:

```
void kprintf(char *f, ...);
```

Dozvoljeni su sljedeći znakovi konverzije: b, c, d, o, s, u i x. U tablici 14. opisani su znakovi konverzije.

Tablica 14. Znakovi konverzije

Format	Opis
%c	1 znak
%d	decimalni cijeli broj
%s	niz znakova
%b	binarni broj
%o	oktalni broj
%u	decimalni cijeli broj bez predznaka
%x	heksadecimalni broj

4.5.2. Postavljanje boje znaka

Atributi znaka mogu se podesiti sa *set_fgcolor* i *set_bgcolor*. Funkcije kao argument primaju boju koju treba postaviti. Ako vrijednost argumenta nije ispravna (veća od 15) povratna vrijednost funkcije je 0, a inače 1.

Funkcija *set_fgcolor* postavlja boju znaka.

Deklaracija funkcije set_fgcolor:

```
short set_fgcolor(unsigned short fgcolor);
```

Funkcija *set_bgcolor* postavlja boju pozadine znaka.

Deklaracija funkcije set_bgcolor:

```
short set_bgcolor(unsigned short bgcolor);
```

4.5.3. Dohvaćanje i postavljanje položaja kursora

Za dohvaćenje i postavljanje položaja kursora namijenjene su funkcije *set_cursor* i *get_cursor*. Postavljanje položaja kursora obavlja se sa *set_cursor*. Argument funkcije je položaj kursora.

Deklaracija funkcije set_cursor:

```
void set_cursor(short cursor);
```

Položaj kursora može se dobiti pozivom funkcije *get_cursor*. Funkcija kao povratnu vrijednost vraća položaj kursora.

Deklaracija funkcije get_cursor:

```
short get_cursor();
```

4.5.4. Pomicanje sadržaja zaslona

Pomicanje sadržaja zaslona obavlja funkcija *kscroll*. Pozivom funkcije sadržaj se pomiče prema gore za jednu liniju.

Deklaracija funkcije kscroll:

```
void kscroll();
```

4.5.5. Brisanje sadržaja zaslona

Nekad je potrebno izbrisati sadržaj zaslona. Brisanje sadržaja cijelog zaslona obavlja funkcija *kclear*.

Deklaracija funkcije kclear:

```
void kclear();
```

4.6. Postavljanje tablice GDT

Funkcija *gdt_install* postavlja tablicu GDT. Postavljena tablica sadržava sljedeće ulaze: nulti opisnik, opisnik segmenta koda i opisnik segmenta podataka.

Deklaracija funkcije gdt_insall (u C-u):

```
extern void gdt_install();
```

4.7. Ponovno postavljanje kontrolera prekida PIC

Funkcija *PIC_ramap* obavlja ponovno postavljanje PIC-ova. Reprogramiranje PIC-a se obavlja kako bi IRQ-ovi koristili nerezervirane vektore. Prvi argument funkcije je pomak vektora glavnog PIC-a, a drugi argument pomak vektora pomoćnog PIC-a. Uobičajeno se pomak glavnog PIC-a postavlja na 0x20, a pomak pomoćnog PIC-a na 0x28.

Deklaracija funkcije PIC_remap:

```
short PIC_remap(short pic1, short pic2);
```

4.8. Postavljanje tablice IDT

Funkcija *idt_install* instalira tablicu IDT i postavlja prva 32 ulaza u IDT tablici koja su namijenjena za iznimke. Ako se dogodi neka iznimka na temelju postavljenih ulaza pozvat će se odgovarajući rukovatelj iznimke ISR (engl. *interrupt service routine*). Rukovatelj iznimke ispisuje odgovarajuću poruku i zaustavlja sustav.

Deklaracija funkcije idt_install (u C-u):

```
extern void idt_install();
```

Funkcija *irq_install* obavlja postavljanje ulaza (od 32. do 47.) u IDT tablici. Ovi ulazi namijenjeni su za sklopovske prekide. Da bi neki sklopovski prekid mogao biti poslužen potrebno je još sa funkcijom *set_irq_handler* postaviti rukovatelja toga prekida. Prije poziva funkcije *irq_install* potrebno je pozvati funkciju *PIC_remap*.

Deklaracija funkcije irq_insall (u C-u):

```
extern void irq_install();
```

Postavljanje rukovatelja za određeni IRQ obavlja funkcija *set_irq_handler*. Pozivom ove funkcijom sprema se adresa one funkcije (rukovatelja) koja će biti pozvana kada se dogodi određeni tip prekida. Na primjer za tipkovnicu se poziva *set_irq_handler(1, keyboard_handler)*. Pri čemu je keyboard_handler adresa funkcije. Kada se dogodi prekid sa tipkovnice funkcija keyboard_handler ce biti pozvana za obradu toga sklopovskog prekida.

Deklaracija funkcije set_irq_handler :

```
void set_irq_handler(unsigned int irq, unsigned int handler);
```

4.9. Funkcije za rad sa tipkovnicom

Kada se neka tipka pritisne moguće je pročitati skenirajući kod, a zatim iz toga koda može se saznati ASCII vrijednost znaka za tu tipku. Upravo to rade funkcije *get_scancode* i *kgetch*. U nastavku su opisane rutine koje olakšavaju izradu upravljačkog programa za tipkovnicu.

4.9.1. Čitanje skenirajućeg koda

Funkcija *get_scancode* služi za čitanje skenirajućeg koda iz podatkovnog registra tipkovnice. Povratna vrijednost funkcije je pročitani skenirajući kod.

Deklaracija funkcije get_scancode:

```
int get_scancode();
```

4.9.2. Dohvaćanje ASCII koda znaka

Funkcija *kgetch* kao argument prima skenirajući kod. Povratna vrijednost funkcije je ASCII vrijednost znaka. Za skenirajući kod tipki kao što su shift, ctrl, alt povratna vrijednost je nula.

Deklaracija funkcije kgetch:

```
int kgetch(int scancode);
```

4.9.3. Ažuriranja kod tipkovnice

Promjenu stanja lampica tipkovnice obavlja funkcija *update_led*. Funkcija kao argumente prima stanje lampica za Scroll Lock, Num Lock i Caps Lock. Ako se želi uključiti neka lampica vrijednost argumenta koji predstavlja stanje te lampice treba biti 1. Ako se lampica želi isključiti vrijednost argumenta za tu lampicu treba biti 0. Pri unosu neodgovarajućih vrijedosti (vrijednost manja od 0 ili veca od 1), funkcija neće obaviti promjenu stanja lampica.

Deklaracija funkcije update_led:

```
void update_led(unsigned short scroll, unsigned short num_lock,
                unsigned short caps_lock);
```

Kako bi se dohvatio ispravan ASCII kod nekog znaka prije poziva funkcije *kgetch* potrebno je pozvati funkciju *kb_update* koja ažurira stanja tipkovnice (npr. ažurira lampice, pamti stanje tipke shift i slično).

Deklaracija funkcije kb_update:

```
void kb_update(int scan_code);
```

4.10. Postavljanje frekvencije brojila sustava

Postavljanje željene frekvencije brojila obavlja se sa funkcijom *modulate_pit_freq*. Funkcija kao argument prima frekvenciju.

Deklaracija funkcije mudulate_pit_freq:

```
int modulate_pit_freq(unsigned int frequency);
```

4.11. RTC funkcije

Vrijeme i datum može se dohvatiti sa RTC-a. U nastavku su opisane funkcije za dohvaćanje i namještanje vremena i datuma.

4.11.1. Dohvaćanje i spremanje vremena

Funkcije *get_RTCtime* dohvaća vrijeme. Nakon poziva funkcije sat će biti pohranjen na mjesto na koje pokazuje prvi argument, minuta na mjesto koje pokazuje drugi argument, a sekunda na mjesto na koje pokazuje treći argument.

Deklaracija funkcije get_RTCtime:

```
void get_RTCtime(short *hour, short *min, short *sec);
```

Vrijeme se može podesiti sa funkcijom *set_RTCtime*.

Deklaracija funkcije set_RTCtime:

```
void set_RTCtime(short hour, short min, short sec);
```

4.11.2. Dohvaćanje i spremanje datuma

Datum se može saznati sa funkcijom *get_RTCdate*. Nakon poziva funkcije dan će biti pohranjen na mjesto na koje pokazuje prvi argument, mjesec na mjesto koje pokazuje drugi argument, a godina na mjesto na koje pokazuje treći argument.

Deklaracija funkcije get_RTCdate:

```
void get_RTCdate(short *day, short *month, int *year);
```

Datum se može podesiti sa funkcijom *set_RTCdate*.

Deklaracija funkcije set_RTCdate:

```
void set_RTCdate(short day, short month, int year);
```

4.12 Ponovno podizanje sustava

Funkcija *reboot* služi za ponovno podizanje sustava.

Deklaracija funkcije reboot:

```
void reboot();
```

5. Korištenje osnovnih rutina

U ovom poglavlju pokazano je korištenje funkcija za ispis na zaslon, instaliranje rukovatelja prekida i instaliranje tablica sustava.

5.1. Korištenje funkcija za ispis na zaslon

Korištenje funkcija za ispis na zaslon je dosta jednostavno. Jedan od razloga tome je velika sličnost sa C funkcijama za ispis. Funkcijom *test_print* pokazuje se korištenje svih funkcija za ispis na zaslon.

```
void test_print()
{
	char str[] = "EFG";
	int num1 = 1;
	int num2 = 2;

	kprintf("Testni ispis: \n");
	kputs("\nABCD\n");
	kputi(num1);
	kprintf("\t%d \n%s", num2, str);
	kputch('H');
}
```

Pozivom funkcije *test_print* dobiva se sljedeći ispis:

```
Testni ispis:

ABCD
1	2
EFGH
```

5.2. Postavljanje rukovatelja prekida

Kako bi se u slučaju prekida pozvao odgovarajući rukovatelj toga prekida potrebno je postaviti adresu rukovatelja prekida. Postavljanje adrese obavlja funkcija *set_irq_handler*. Kao prvi argument potrebno je navesti IRQ broj, a kao drugi adresu rukovatelja tog prekida.

U sljedećem primjeru pokazano je kako se postavlja rukovatelj za tipkovnicu. Funkcija *keyboard_han*dler je rukovatelj prekida tipkovnice. U primjeru se prvo poziva funkija *get_scancode* kako bi se dohvatio skenirajući kod tipke. Zatim se poziva funkcija *kb_update* koja obavlja ažuriranja kod tipkovnice. Pozivom funkcije *kgetch* dohvaća se ASCII kod znaka, koji

ako je različit od nule ispisuje se na zaslon. Sa nulom se provjerava da li se radi o tipkama za koje uopće nije potreban ispis (npr. ctrl, alt, shift).

```
void keyboard_handler()
{
	int scancode = get_scancode();
	kb_update(scancode);
	int key = kgetch(scancode);

	if(key != 0)
		kputch(key);
}
```

Postavljanje rukovatelja prekida za tipkovnicu obavlja se pozivom sljedeće funkcije:

```
set_irq_handler(1, keyboard_handler);
```

5.3. Postavljanje tablica sustava

Prvo je potrebno instalirati tablicu GDT. To se obavlja pozivom funkcije *gdt_install*. Prije instaliranja tablice IDT potrebno je napraviti ponovno postavljanje PIC-ova pozivom funkcije *PIC_remap*. Zatim se instalira tablica IDT i postave rukovatelji za iznimke pozivom funkcije *idt_install*. Sa funkcijom *irq_install* postavljaju se IDT ulazi za sklopovske prekide. U sljedećem primjeru pokazano je postavljanje tablica sustava.

```
void main()
{

	gdt_install();
	PIC_remap(0x20, 0x28);
	idt_install();
	irq_install();

	for( ; ; );
}
```

6. Zaključak

U ovom diplomskom radu ostvarene su osnovne rutine koje olakšavaju izradu jednostavnih operacijskih sustava. Rutine su namijenjene za izgradnju operacijskih sustava za x86 arhitekturu. Postoji još dosta mjesta za poboljšanje ovih funkcija (npr. optimizacija). Moguće su i neke greške kod rutina.

Korištenjem rutina ostvaren je jednostavni operacijski sustav. Operacijski sustav je monolitnog dizajna i podržava višezadaćnost. Korišten je algoritam kružnog raspoređivanja (engl. *round-robin scheduling algorithm*). Moguća su poboljšanja implementiranjem boljeg algoritma za raspoređivanje poslova.Trebalo bi još omogućiti i korištenje prstena 3 za procese.

Mehanizam straničenja je omogućen (aktiviran). Pri tome je implementiran i upravitelj fizičke memorije (engl. *physical memory manager*). Straničenje služi za zaštitu memorije i implementiranje virtualne memorije. Postupak omogućavanja straničenja se pokazao dosta složenim.

Idući koraci u razvoju operacijskog sustava mogu biti u smjeru ostvarenja dinamičke dodjele memorije (engl. *dynamic memory allocation*), podršci za datotečni sustav, upravljački program (engl. *driver*) za miš, itd.

Razvoj operacijskog sustava je dosta složen zadatak, ali radi se o jako zanimljivom iskustvu.

7. Literatura

1. Bran's Kernel Development, http://www.osdever.net/bkerndev/Docs/title.htm
2. Multitasking tutorial, http://hosted.cjmovie.net/TutMultitask.htm
3. Tutorials, http://hem.passagen.se/gregge/index_hires.html
4. Roll your own toy UNIX-clone OS, http://www.jamesmolloy.co.uk/tutorial_html/
5. Operating System Development Series, http://www.brokenthorn.com/Resources/OSDevIndex.html
6. OSDev.org, http://wiki.osdev.org
7. Bare Bones, http://wiki.osdev.org/Bare_Bones
8. Programming the PIC, http://www.osdever.net/tutorials/pic.php
9. 8259 PIC, http://wiki.osdev.org/PIC#What_does_the_8259_PIC_do.3F
10. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, 2008.
11. Setting Up Paging, http://wiki.osdev.org/Setting_Up_Paging#Basic_Paging
12. Hardware Level VGA and SVGA Video Programming Information Page, http://www.stanford.edu/class/cs140/projects/pintos/specs/freevga/vga/crtcreg.htm
13. OSD: Putting text on the screen, http://bos.asmhackers.net/docs/textmode/OSD%20Putting%20text%20on%20the%20screen.htm
14. Clock and Timer Circuits, http://jno.glas.net/data/prog_books/lin_kern_2.6/0596005652/understandlk-CHP-6-SECT-1.html
15. PS2 Keyboard, http://wiki.osdev.org/PS2_Keyboard
16. IBM PC keyboard information for software developers http://www.osdever.net/documents/pdf/kbd.pdf

17. Keyboard scancodes, http://www.win.tue.nl/~aeb/linux/kbd/scancodes.html

18. 8259A/82C59A-2 Programmable Interrupt Controller, Technical questions and Answers http://ftp.utcluj.ro/pub/users/nedevschi/PMP/WLab/io/

19. Upravljanje procesima u jednostavnim operacijskim sustavima, Antonio Arbutina

20. 8253 PIT, http://heim.ifi.uio.no/~stanisls/helppc/8253.html

21. IBM AT and XT Keyboards Operation, http://vgcontrols.com/Products/encoders/pdf/pckeybrd.pdf

22. Onyxkernel , http://code.google.com/p/onyxkernel/

23. Paging: Memory Mapping With A Recursive Page Directory, http://www.rohitab.com/discuss/index.php?s=d8a43d31230553511bdab419e5470538&showtopic=31139

24. Linux Device Drivers, www.cs.usfca.edu/~cruse/cs635s03/lesson2.ppt

25. Leap year, http://en.wikipedia.org/wiki/Leap_year

26. PC Keyboard Scan Codes, http://www.barcodeman.com/altek/mule/scandoc.php

27. The AT-PS/2 Keyboard Interface, Adam Chapweske, http://www.tayloredge.com/reference/Interface/atkeyboard.pdf

28. Interrupts, http://wiki.osdev.org/IRQ

29. Understanding Hardware Device Resources Part One - IRQ's, http://www.pcnineoneone.com/howto/irq1.html

30. RTC/NVRAM (CMOS RAM) Batteries, http://www.freeopenbook.com/upgrading-repairing-pc/ch21lev1sec15.html

31. Real Time Clock / CMOS Setup Reference, Tom Przeor, http://www.nondot.org/sabre/os/files/MiscHW/RealtimeClockFAQ.txt

32. Periodic Interrupts with the Real Time Clock, http://www.geocities.com/SiliconValley/Campus/1671/docs/rtc2.htm

33. What is Real Time Clock ? , http://www.ust.hk/itsc/y2k/Y2k_pc/RTC_detail.html

34. Intel 8253, http://en.wikipedia.org/wiki/Intel_8253

35. Programmable Interval Timer, http://en.wikipedia.org/wiki/Programmable_Interval_Timer

36. Programmable Interval Timer, http://wiki.osdev.org/Programmable_Interval_Timer

37. 8254 Programmable Interval Timer, http://www.scs.stanford.edu/09wi-cs140/pintos/specs/8254.pdf

38. Programmable Interrupt Controller, http://en.wikipedia.org/wiki/Programmable_Interrupt_Controller

39. Using Interrupts, http://www.beyondlogic.org/interrupts/interupt.pdf

40. X86 Assembly, http://upload.wikimedia.org/wikibooks/en/1/11/X86_Assembly.pdf

41. CMOS RTC - Real Time Clock and Memory, http://docs.huihoo.com/help-pc/hw-CMOS_RAM.html

42. 8259A Programmable Interrupt Controller (8259A/8259A-2), http://www.scs.stanford.edu/09wi-cs140/pintos/specs/8259A.pdf

43. Interrupts and Handlers, http://www.delorie.com/djgpp/doc/ug/interrupts/inthandlers1.html

44. Interrupt descriptor table, http://en.wikipedia.org/wiki/Interrupt_descriptor_table

45. How Operating Systems Work, http://computer.howstuffworks.com/operating-system.htm

46. Kernel (computing), http://en.wikipedia.org/wiki/Kernel_(computing)

47. Monolithic kernel vs. Microkernel, Benjamin Roch, TU Wien, http://www.vmars.tuwien.ac.at/courses/akti12/journal/04ss/article_04ss_Roch.pdf

48. Microkernels, http://educationalstuff1.tripod.com/microkernels.pdf

49. Microkernel, http://wiki.osdev.org/Microkernel

50. Monolithic Kernel, http://wiki.osdev.org/Monolithic_Kernel

51. Uvod u kernel, Dinko Korunić, http://sistemac.carnet.hr/node/75

52. Understanding the Linux Kernel, Daniel P. Bovet, Marco Cesati, http://jno.glas.net/data/prog_books/lin_kern_2.6/0596005652/main.html

53. Interrupts, Exceptions, and IDTs, http://www.osdever.net/tutorials/pdf/interrupts_2.pdf

54. Global Descriptor Table, http://wiki.osdev.org/GDT

55. Paging, http://wiki.osdev.org/Paging

56. NASM – The Netwide Assembler, http://www.et.byu.edu/groups/ece425web/stable/labs/nasmmanual.pdf

57. Lecture 10, http://www.eecs.wsu.edu/~ee314/lectures/lecture10.pdf

58. Interrupt Descriptor Table, http://wiki.osdev.org/IDT

59. PC Architectures, http://www.electronballet.com/Datasheets/ET4508_L10.pdf

60. Reference All In One, http://www.run.montefiore.ulg.ac.be/~martin/osfaq-copy/ReferenceAllInOne.html

61. Interrupts, http://www.eecs.utoledo.edu/~ewing/Real_Time/RTOS/RMX/IntelNotes/Chapter_7.pdf

62. Time And Date, http://wiki.osdev.org/Time_And_Date