Uvod u Organizaciju i Arhitekturu Racunara 2B — Cheat Sheet

Andrija Urošević

16. septembar 2019

1 Fon-Nojmanovi Racunari

1.1 ISA

ISA (eng. *Instruction Set Architecture*) je jedan apstraktan model racunara. ISA obuhvata skup registara, skup instrukcija, formati instrukcija, nacini adresiranja operanada, velicina memorijskog prostora, rezim rada, sistem prekida itd.

1.2 Instrukcioni Formati

Struktura masinske instrukcije:

OPCODE | OPERANDS

Operandi mogu biti registarski, memorijski i neposredni. Razmotrimo sledeci izraz u zavisnosti od vrste instrukcije: $(a + b) * (c + d)$

1.2.1 Tro-adresne Instrukcije

OPCODE | DEST | SRC | SRC

```
ADD     R1, A, B
ADD     R2, C, D
MUL     X, R1, R2
```

Imaju tri adresna polja za specifikovanje registra ili memorijske lokacije. Program je mnogo manji po duzini, ali broj bitova po instrukciji se povecava, sto ne znaci da ce biti brzi jer puno stvari mora da uradi u jednoj instrukciji (load, execute, store, itd.).

1.2.2 Dvo-adresne Instrukcije

OPCODE | DEST | SRC

```
MOV     R1, A
ADD     R1, B
MOV     R2, C
ADD     R2, D
MUL     R1, R2
MOV     X, R1
```

Najpristupljeniji u komercijalnim racunarima.

1.2.3 Jedno-adresne Instrukcije

OPCODE | OPERAND

```
LOAD    A
ADD     B
STORE   T
LOAD    C
ADD     D
MUL     T
STORE   X
```

Koristi *akumulator* registar za manimulisanjem podacima. Jedan operadn je akumulator a drugi je u registru ili na memorijskoj lokaciji.

1.2.4 Nula-adresne Instrukcije

```
PUSH    A
PUSH    B
ADD
PUSH    C
PUSH    D
MUL
POP     X
```

Ovi racunari su bazirani na staku. Da bi se izracunao izraz prvo se prevede u obrnutu poljsku notaciju.

1.3 Load/Store Arhitektura

Load/Store arhitektura je ISA koja deli instrukcije u dve kategorije: pristupanje memoriji (load/store izmedju memorije i registra) i ALU operacije (koje se izvrsavaju samo nad registarima). Implementira ih RISC.

Na primer, oba operanda i destinacija za ADD operaciju moraju biti u registrima.

1.4 RISC/CISC

CISC (eng. *Complex Instruction Set Computer*):

- Veliki broj instrukcija
- Komplekse instrukcije promenljive duzine
- Instrukcije mogu trajati vise otkucaja radnog takta
- ISA radi sto je vise moguce koriscenjem hardvera
- Efikasno koriscenje RAM-a

RISC (eng. *Reduced Instruction Set Computer*):

- Mali broj instrukcija
- Jednostavne instrukcije fiksirane duzine
- Instrukcije traju tacno jedan otkucaj radnog takta
- Kompajleri visokog nivoa preuzimaju kodiranje kompleksnih instrukcija
- Tesko koriscenje RAM-a (izaziva problem uskog grla)

1.5 Adresiranje

1.5.1 Neposredno Adresiranje

Operand se zadaje najlakse ako se zada na mestu gde se zadaje adresa. Takav operand se zove *neposredni operand* (eng. *immediate operand*). Primer:

```
MOV     R1, 4
```

Koristi se u radu sa malim konstantama.

1.5.2 Direktno Adresiranje

Operand se nalazi na potpunoj adresi koja je zadata u instrukciji. Koristi se u radu sa globalnim promenljivim, jer se adresa globalnih promenljiva ne menja dok njihova vrednost moze da se menja.

1.5.3 Registarko Adresiranje

Slicno je kao i direktno adresiranje sem sto se ovde ne zadaje memorijska lokacija vec registar. Primer

```
MOV     R1, R2
```

Prevodioci se trude da otkriju koje promenljive se najvise koriste i njih smestaju u registre.

1.5.4 Indirektno Adresiranje

Zadati operand dolazi iz memorije ili odlazi u nju, ali njegova adresa nije zabelezena unutar instrukcije kao u direktnom adresiranju, vec se nalazi u registru.

```
ADD     R1, [R2]
```

1.5.5 Indeksno Adresiranje

Adresiranje memorije tako sto se navede registar i konstantno rastojanje zove se *indeksno adresiranje* (eng. *indexed addressing*)

```
ADD     R1, [R2 + R3]
```

1.5.6 Relativno Adresiranje

U instrukcijama grananja potrebna je i odredisna adresa koju takodje treba zadati. Vrednost operanda je relativna adresa u odnosu na adresu tekuce instrukcije, tj. PC registra.

To je u stvari indeksno adresiranje, s registrom PC.

2 x86–64 Arhitektura

2.1 Nacini Adresiranje

```
MOV     rax, 10
MOV     rax, rbx
MOV     rax, 0x603829
MOV     rax, [rbx + rcx]
MOV     rax, [rbx + 4 * rcx]
MOV     rax, [rbx + 4 * rcx + 8]
```

2.2 Tipovi Instrukcija

2.2.1 Instrukcije Transfera

```
MOV     dst, src
LEA     dst, src
PUSH    src
PULL    dst
```

2.2.2 Aritmeticke i Bitovske Instrukcije

```
ADD     dst, src  ;dst += src
SUB     dst, src  ;dst -= src
IMUL    dst, src  ;dst *= src
NEG     dst       ;dst = -dst

AND     dst, src  ;dst &= src
OR      dst, src  ;dst |= src
XOR     dst, src  ;dst ^= src
NOT     dst       ;dst =~dst

SHL     dst, count ; dst <<= count
SAR     dst, count ; dst >>= count
SHR     dst, count ; dst >>= count

MUL    src        ;rax *= src
SHL     dst       ;dst <<= 1
```

2.2.3 Flag-ovi

PSW registar je registar koji sadrzi informacije o stanju procesora.

- **ZF** — Nula (eng. *Zero Flag*) pokazuje da li je rezultat aritmeticke ili logicke operacije bio nula.
- **CF** — Prenos (eng. *Carry Flag*) dopusta brojevima vecim od jedne redi da se dodaju ili oduzimaju, tako sto cuva prenos.
- **SF** — Znak (eng. *Sign Flag*) pokazuje da li je rezultat matematicke operacije negativan.
- **OF** — Prekoracenje (eng. *Overflow Flag*) pokazuje da li je oznaceni rezultat operacije prevelik da stane u registar.

2.2.4 Instrukcije poredjenja

```
CMP     op1, op2  ;op1 - op2
TEST    op1, op2  ;op1 & op2

JMP     target
JE      target   ;(ZF=1)
JNE     target   ;(ZF=0)
JL      target   ;(SF!=OF)
JLE     target   ;(ZF=1 ili SF!=OF)
LG      target   ;(ZF=0 i SF=OF)
LGE     target   ;(SF=OF)
JA      target   ;(CF=0 i ZF=0)
JB      target   ;(CF=1)
JS      target   ;(SF=1)
JNS     target   ;(SF=0)
```

2.3 Funkcije

2.3.1 Pozivanje Prodecura

Pre poziva procedure mora se sacuvati povratna adresa. Ona moze da se cuva u registar ili na stek.

Ako bi postojao poseban registar koji bi cuvao povratnu adresu, nakon zavrsetka procedure znali bi smo gde da se vratimo. Ali recimo da procedura P poziva proceduru Q i onda procedura Q poziva proceduru R, nakon izvrsavanja procedure R i vracanja u proceduru Q izgubili bi adresu za vracanje u proceduru P.

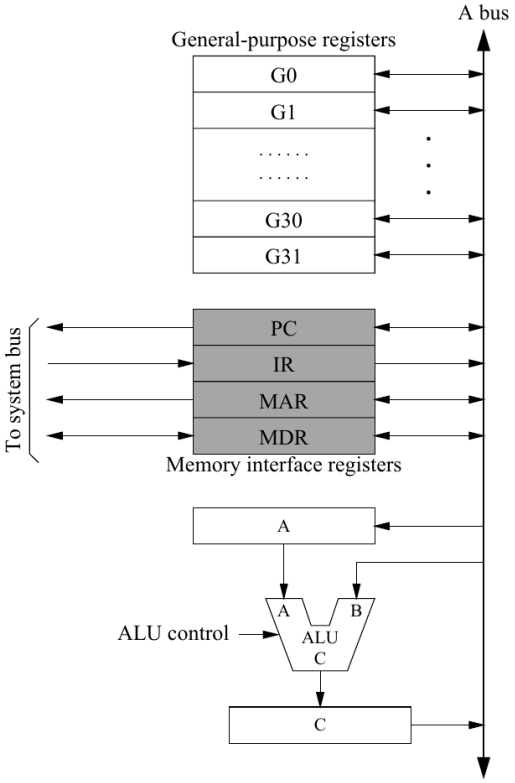
Druga mogucnost jeste cuvanje povratne adrese na steku. Pozivajuca procedura gura adresu na stek, nakon izvrsavanja pozvane procedure ona skida adresu sa steka. Time se resava problem gubljenja povratne adrese i omogucava se rekurentan poziv procedure.

2.3.2 Prenos Argumenata

Argumenti se takodje mogu prenositi na dva nacina preko registara i preko steka.

Argumenti se prenose preko registra na prethodno dogovoren nacin i to po sledecem redosledu: RDI, RSI, RDX, RCX, R8, R9. Ovaj metoda ne radi za pozivanje funkcije u trenutno pozvanoj funkciji i kada su parametri veci od 8 bajta.

Argumenti mogu da se prenose i preko steka tako sto ih pozivajuca procedura gurne na stek u obrnutom poretku, tj. prvo se gurne param n, onda param (n-1), ..., param 1. Onda nakon izvrsavanja pozvane procedure, pozivajuca procedura skida sa steka prethodno dodate argumente.



2.3.3 Vracanje Vrednosti

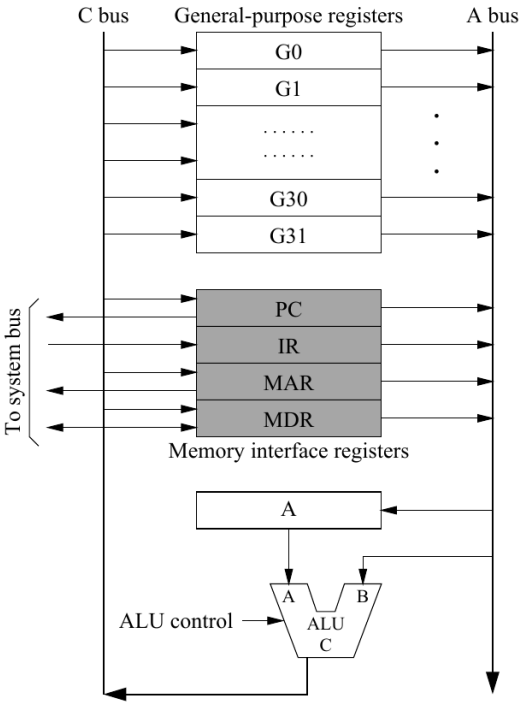
Povratna vrednost ako je 8 bajta ili manje vraca se preko registra RAX, u slucaju da je vrednost sa pokretnim zarezom koristi se *floating-point register*.

Vrednost ukoliko je neka strutura ona se gura na stek.

2.3.4 Pozivanje funkcije x86–64

Skeleton pozivanja funkcije:

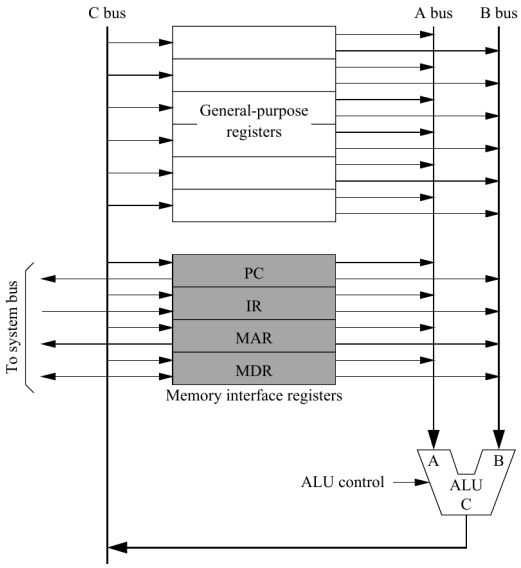
```
function:
    # prologue
    PUSH    ebs
    MOV     ebp, esp
    SUB     esp, N
     :::
    # epilogue
    MOV     ESP, EBP
    POP     EBP
    RET     0

main:
     :::
    MOV     RDI, 1st arg
    MOV     RSI, 2nd arg
     :::
    MOV     R9, 6th arg
    PUSH    7th arg
    PUSH    8th arg
     :::
    CALL function
    POP     n-th arg
     :::
    POP     7th arg
     :::
```



3 Procesor

Osnovne komponente procesora su putanja podataka, kontrolna jedinica, memorijske komponente (registri i kes memorija), radni takt, logicke kapije.

Putanja podataka (eng. *data path*) je deo procesora koji sadrzi aritmeticko-logicku jedinicu (ALI), njene ulaze i izlaze. Sadrzi i niz registara opste namene i memorijske registre



3.1 Interni Registri Procesora

- PC — programski brojac (eng. *Program Counter*)
- IR — instrukcioni registra (eng. *Instruction Registar*)
- MAR — registar memorijskih adresa (eng. *Memory Address Register*)
- MDR — registar memorijskih podataka (eng. *Memory Data Register*)
- PSW — statusni registart (eng. *Program Status Word*)
- CW — kontrolni registar (eng. *Control Word*)

3.2 Faze Izvrsavanja Instrukcije

1. *Dohvatanje instrukcije* (eng. *Fetch*): Instrukcije se ucitava iz memorije sa adrese na koju ukazuje PC registar i prebacuje na MDR registar. U MAR registar se prebaci vrednost PC registra, izdaje se komanda za citanje memoriji. Vrednost dolazi sa magistrale podataka do MDR registra, i ona se u njemu zapamti.
2. *Dekodiranje instrukcije* (eng. *Decode*): Instrukcija se prebacuje u IR registar cija se vrednost salje na ulaze kontrolne jedinice. Dekodiranje se vrsti kada kontrolna jedinica utvrdjuje koju instrukciju zapravo izvrsava i koje korake treba da preduzme u nastavku.
3. *Izvrsavanje instrukcije* (eng. *Execute*): Izvrsava se odredjenja operacija.

   Ukoliko instrukcija sadrzi memorijske operande, u ovoj fazi sve vrsi i dohvatanje operanada iz memorije (na slican nacin kao i dohvatanje instrukcije).

   U slucaju aritmeticko-logickih instrukcija, vrednost operanada se propusta kroz ALU da bi se izracunao rezultat.

   U slucaju transfera izmejdu registra, vrednost izvornog registre se propusta korz ALU bez ikakve operacije i prebacuje u odredisni registar.

   U slucaju transfera iz registra u memoriju, vrednost iz izvornog registra se kopira u MDR registra, a adresta odredista u MAR registar. Nakon toga se preko magistrale podatak upisuje na zadatu adresu.

   U Slucaju instrukcije skoka, izvrsavanje zavisi od ispunjenosti uslova. Ukoliko flegovi ukazuju da je uslov ispunjen, vrsi se upis adrese na koju se skace u PC registar, u suprotnom ne radi se nista.

3.3 Nacini Implementacije Kontrolne Jedinice

3.3.1 Tvrdo Ozicena Implementacija

Tvrdo Ozicena Implementacija (eng. *Hardwired Control Unit*) ili "hardverska implementacija".

- Dizajnira se sekvencijalno kolo koje funkcionise kao konacni trasduktor
- Na ulazu se nalaze vrednosti IR i PSW registra, a na izlazu se nalazi skup kontrolnih signala koji se salju putanji podataka, momoriji, i ostalim delovima racunara.
- Stanja odgovaraju fazama izvrsavanja instrukcija
- Mogu postojati i medjustanja u slucaju komplikovanih operacija.
- Pogodan za procesore sa jednostavnim skupom instrukcija (RISC).
- Brz je, ali zato ne moze lako da za modifikuje

3.3.2 Mikroprogramirana Implementacija

Mikroprogramirana implementacija (eng. *Microprogrammed Contrl Unit*) ili "softverska implementacija".

CONTROL SIGNAL | ADDRESS BITS

- Kontrolna jedinica sadrzi *kontrolnu memoriju* (eng. *control store*). Svaka adresibilna lokacija u ovoj memoriji sadrzi jendu mikroinstrukciju.
- Svaka mikroinstrukcija se izvrsava u jednom ciklusu.
- Adresni bitovi iz tekuce mikroinstrukcije se kombinuju na odgovarajuci nacin sa vrednoscu IR i PSW registra cime se dobija adresa sledece mikroinstrukcije u kontrolnoj memoriji.
- MIR registar (eng. *Micro Instruction Register*) sadrzi tekucu mikroinsrukciju.
- Niz mikroinstrukcija u kontrolnoj memoriji naziva se *mikroprogram*.
- PSW registar takodje utice na izbor mikroprograma koji ce se izvrsavati, jer se pojedinacne instrukcije izvrsavaju drugacije u zavisnosti da li je niki uslov ispunjen ili ne.
- Mikroprogramirana kontrolna jedinica omoguca da se dizajn kontrolne jedinice svede na mikroprogramiranje. Jednostavnost u slucaju slozenih skupova instrukcije (CISC).
- Losa strana je sto je sporiji od tvrdo-ozicenih implementacija

Struktura mikroinstrukcije:

1. *Horizontalna struktura*: Svi kontrolni signali postoje kao pojedinacni bitovi mikroinstrukcije i mogu se direktno usmeriti ka odgovarajucim komponentama procesora.
2. *Vertikalne strukture*: Ideja je da se duzina mikroinstrukcije smanji tako sto se prepoznaju neke tipicne kombinacije kontrolnih signala koje se javljaju u mikroinstrukcijama.

4 Memorija

4.1 Karakteristike Memorije

4.1.1 Vrste Pristupa

Proizvoljan Pristup: Kod ovih memorija postoji mehanizam za pristupanje svakoj adresibilnoj jedinici u priblizno konstantnom vremenu. Koristi se jos i *slucajan pristup* (eng. *random access*).

Sekvencijalni Pristup: Adresibilnim jedinicama u momoriji se moze pristupiti iskljucivo redom. Da bismo pristupili 10-toj adresibilnoj jedinici, moramo pre toga da pristupimo svim adresibilnim jedinicamo od 0-te do 9-te redom.

Direktan Pristup: Kod ovih memorija posotji moguc-nost direktnog pristupa nekoj okolini adresibilne jedinice u memoriji na osnovu zadate adrese. Kada se pristupi okolini, pretraga trazene adresibilne jedinice se nastavlja sekvencijalno unutar ukoline.

Asocijativni Pristup: Podatku se ne pristupa preko adrese, vec tako sto se zadaje neka maska koja se na neki nacin pretrazuje u memoriji.

4.1.2 Kapacitet

Kolicina podataka koja se moze sacuvati u memoriji. Izrazava se u (KiloByte, MegaByte, GigaByte, TeraByte, ...).

4.1.3 Brzina

Na brzinu uticu *kasnjenje* (eng. *memory latency*) i *brzina transfera* (eng. *bandwitdth*). Kasnjene se izrazava vremenom koje protekne od trenutka iniciranja transfera do trenutka kada taj transfer zaista zapocne. Brzina transfera se izrazava kolicinom podataka koji se mogu preneti u jednoj sekundi.

Vreme pristupa je prosecno ukupno vreme potrebno da se pojedinacni podatak prenese iz memorije u procesor.

Vreme memorijskog ciklusa predstavlja vreme koje je potrebno da prodje izmedju dva uzastopna zahteva za pristupom. Ukljucuje vreme pristupa, ali i dodatno vreme koje mora proci nakon zavrsetka transfera do ponovnog zahteva za pristupom.

4.1.4 Postojanost

Memorija je *postojana* (eng. *persisten/non-valatile*) ako se iskljucenjem napajanja njen sadrzaj ne gubi.

Memorija je *nepostojana* (eng. *non-persistent/valatile*) ako se iskljucenjem napajanja njen sadrzaj gubi.

4.1.5 Promenljivost

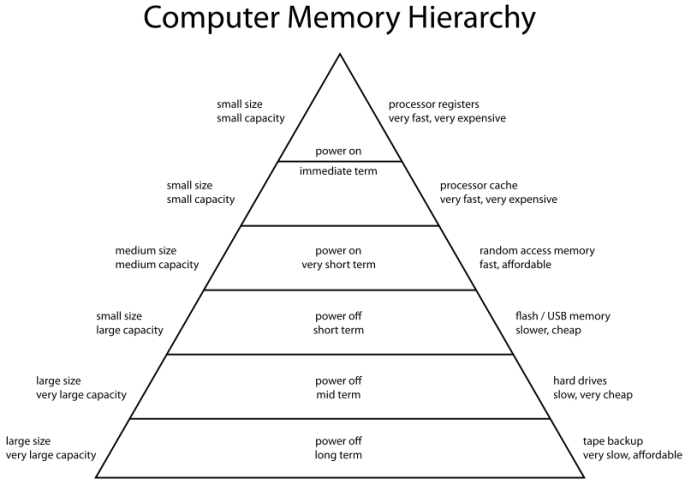
Memorija ciji je sadrzaj nepromenljiv nazivaju se *nepromenljive memorije* (eng. *read-only memory*).

Memorija ciji je sadrzaj promenljiv naziva se *promenljiva memorija* (eng. *read-write memory*).

4.2 Memorijske Hijerarhije

Univerzalna memorija je memorija koja ima sve pozeljne karakteristike. Ovakva memorija ne postoji.

Memorije su organizovane u *memorijsku hijerarhiju.*



4.2.1 Unutrasnje memorije

Read Only Memory (ROM): ROM sadrzi programe i podatke koji su nepromenljivi i koji treba da su stalno prisutni da bi racunar mogao da radi. ROM ima proizvoljan pristup. Implementiraju se kao kombinatorno kolo.

PROM (Programmable ROM) koji se moze programirati samo jednom, spaljivanjem odgovarajucih zica.

EPROM (Erasable Programmable ROM) koji se moze programirati uz pomoc posebnog uredjaja za programiranje. Moze se programirati vise puta, ali je pre svakog programiranja potrebno obrisati prethodni sadrzaj.

EEPROM (Electronically Erasable Programmable ROM) koji se moze brisati elektronskim putem i zatim ponovo programirati.

Random Access Memory (RAM): RAM memorija je osnovna memorija u kojoj se nalaze podaci i programi koji se trenutno izvrsavaju.

SRAM (Staticki RAM) se konstruise pomocu kola slicnih osnovnim D flip-flopovima. Cuva svoj sadrzaj sve dok postoji napajanje. Vreme pristupanja nekoliko milisekundi.

DRAM (Dinamicki RAM) se sastoji od niza celija, a svaka sadrzi jedan tranzistor i mali kondenzator. Kondenzatori se pune i prazne cime se omogucava cuvanje logickih nula i jedinica. Kako elektricni napon tezi disipaciji, svaki bit u DRAM mora se *osvezavati.*

SDRAM (Synchronous DRAM) je hibrid staticke i dinamicke memorije kojom upravlja glavni radni takt. Prednost je sto radni takt uklanja potrebu za upravljacke signale, sto je cini brzom.

DDR (Double Data Rate) SDRM je memorija koja isporucuje podatke i na ulaznoj i na silaznoj ivici radnog takta.

Da bi se smanjilo kasnjenje prilikom pristupa susednim memorijskim adresama je koriscenje *isprepletanih memorija* (eng. *interleaved memory*) Ideja je da se memorija podeli na nekoliko manjih memorijskih jedinica koje nazivamo *bankama.* Banke se organizuju slicno kao organizovanje memorijskih cipova u vece cipove. Nizi bitovi se koriste za izbor banke. Ovo daje vremena bankama da pripreme svoje podatke, kasnjenja pojadinacnih banki se medjusobno preklapaj. Ovaj metod radi samo ukoliko se pristupa podacima sekvencijalno.

4.2.2 Spoljasnje memorije

Hard Disk: Sadrze veci broj diskova premazanih magnetnim materijalom koji su postavljeni na istu osovinu. Imaju veliki kapacitet, ali su dosta sporiji od najsporije unutrasnje memorije. Podeljen je na sektore cija je velicina 512 bajtova. Sektori se grupisu u staze koje predstavljaju koncentricne krugove na diskovima razlicitih precnika. Svaki sektor ima svoju adresu.

Floppy Disk: Slicni kao hard diskovi koji su elektromagnetni. Veoma malog kapaciteta. Danas se ne koriste.

CD i DVD ROM: Uredaju koji omogucuju samo citanje. Zapisane na optickom principu.

Flash Memorije i SSD: Zasnovani su na slicnoj tehnologiji kao i EEPROM, s razlikom sto ne omogucuju brisanje pojedinacnih bajtova, vec iskljucivo prisanje i upis citavih blokova. Dosta su brzi od hard diskova, ali su dosta skuplji a i zivotni ciklus im je dosta kraci.

4.3 Mapiranje Memorijskih Adresa

Adresni prostor procesora je skup adresa koje procesor moze direktno da adresira. Procesori se dizajniraju tako je adresni procesor veci nego sto je to zapravo potrebno (jer se pretpostavlja da ce se RAM uvecavati). Memorija ce zauzimati samo deo adresnog prostora procesora. *Mapiranje memorijskih adresa* je pridruzivanje adrese procesorskog adresnog procesora, adresibilnim lokacijama u memoriji.

Mapiranje adresa se vrsi tako sto se adresa podeli na nizi i visi deo. Visi deo adrese aktivira memorijski modul u kome ce se traziti odgovarajuca adresa, a nizi deo adrese odredjuje adresu u okviru modula.

Jednostavan nacin za selektovanje memorijskog modula na osnovu viseg dela adrese je da se odgovarajucom kombinatornom logikom implementira funkcija koja ima vredno 1 samo za odgovarajucu kombinaciju visih bitova adrese.

Ukoliko svakom memorijskom modulu odgovara jedinstvena kombinacija visih bitova adrese, tada ovakvo mapiranje nazivamo *potpuno mapiranje*.

Postoji mogucnost da se memorijski modul aktivira za veci broj kombinacija visih adresnih bitova. U tom slucaju je istom memorijskom modulu pridruzeno vise opsega adresa u okviru adresnog prostora procesora, te se svakoj njegovoj memorijskoj lokaciji moze pristupiti pomocu razlicitih adresa. Ovaj nacin mapiranja zovemo *parcijalno mapiranje*

4.4 Memorijsko Poravnanje

Memorije su *bajt-adresibilne* ako svaki bajt ima svoju adresku preko koje mu se moze pristupiti. Medjutim, magistrala podataka je po pravilu sira. Ono znaci da prilikom transfera prebacujemo vise od jednog bajta pocev od nekog na odredjenoj adresi. Zbog toga cesto postoji zahtev da se podaci u memoriji organizuju u blokove, tako da je pocetni bajt na adresi koja je deljiva nekim stepenom dvojke. Neke arhitekture to zahtevaju, dok ce druge u slucaju da transfer nije adekvatno poravnat podeliti ga u 2 dela sto smanjuje performanse.

5 Kes Memorija

Kes memorija se koristi kako bi se prevazisla razlika u brzini procesora i RAM-a. Ideja je da se podaci koji se najcesce koriste drze u kes memoriji koja je zasnovana na brzoj statickoj memoriji i time smanjilo kasnjenje.

Princip Vremenske Lokalnosti: (eng. *temporal locality*) Odnosi se na ponovno pristupanje nedavno posecenim memorijskim lokacijama. Memorijske lokacije blizu vrha steka ili instrukcije koje se ponavljaju u petlji.

Koristi se uglavno tako sto se bira koji ce se sadrzaj iz kesa izbaciti tako sto se analizira koriscenje tog sadrzaja u bliskoj proslosti.

Princim Prostorne Lokalnosti: (eng. *spatial locality*) Odnosi se na adrese koje su veoma bliske adresama kojima je nedavno pristupano, pa je vrlo verovatno da ce se i njima pristupiti.

Kes koristi ovo svojstvo da pripremi vise podataka nego sto se trenutno traziz, u nadi da ce oni uskoro biti iskorisceni.

Kes memorija je podeljena na blokove fiksirane velicine koje zovemo *linije kesa*. Prilikom transfera podataka iz memorije u kes, uvek se ucitava ceo blok koji se upisuje u neku liniju kesa.

Ukoliko se podatak, koji procesor zahteva, nalazi u nekoj od linija kesa on se dohvata i transferuje u procesor, to se naziva *pogotak kesa* (eng. *cache hit*).

Ukolio se podatak ne nalazi u nekoj od linija kesa, to se naziva *promasaj kesa* (eng. *cache miss*).

5.1 Razdvojeni i Jedinstveni Kes

Da li instrukcije i podatke treba drzati u istom kesu ili odvojeno?

Jedinstveni kes (eng. *unified cache*), u kome se nalaze i instrukcije i podaci, jednostavan je i dobar je za velike memorije.

Razdvojeni kes (eng. *split cache*), koji je podeljen na deo sa instrukcijama i deo sa podacima, je dobar za male memorije.

5.2 Nivoi Kesa

Kes memorija se obrazuje hjerarhijski, u vise nivoa. Svaki sledeci nivo je sve veci, ali je i sve sporiji. Na taj nacin se najcesce korisceni podaci nalaze u najblizem kesu, oni malo redje u sledecem itd.

Kod *inkluzinvne organizacije* kesa svaki sledeci nivo kesa sadrzi sve podatke koje je sadrzai i prethodni, ali i dodatne podatke koji u prethodnom nisu postojali.

Kod *ekskluzivne organizacije* kesa sledeci nivoi sadrze iskljucivo druge podatke.

L1 kes: Nalazi se u cipu procesora. Realizuje se kao razdvojeni 8-asocijativni kes. Tipicna velicina 32K + 32K.

L2 kes: Nalazi se u cipu procesora. Realizuje se kao jedinstveni 4-asocijativni kes. Tipicna velicina 256K.

L3 kes: Nalazi se na procesorskoj ploci i zajednicki je za sva jezgra procesora. Realizuje se kao jedinstveni 8/16-asocijativni kes. Tipicna velicina: 2M-4M.

5.3 Direktno Mapiranje

Direktno mapiranje odredjuje liniju kesa u koju memorijski blok treba da bude postabljen. Ako je C broj linija kesa, memorijski blok i se mapira na liniju kesa $c \equiv i \pmod{n}$

ADRESA: | TAG | BROJ_LINIJE | POMERAJ |

POMERAJ: poslednjih $b = \log_2 B$ bita, gde je B velicina bloka memorije.

BROJ_LINIJE: $c = \log_2 C$ bita, gde je C broj linija kesa.

TAG: Ostatak adrese, koji sluzi za identifikovanje u kesu.

LINIJA KESA:
| dirty_bit | valid_bit | TAG | PODACI |

Da bi pronasli memorijski blok u kesu, prvo koristimo modulo funkciju za pronalazenje linije kese. Onda posmatramo da li je ta linija kesa validna. Ako jeste uporedjujemo TAG. Ako se oni slazu trazeni blok je pogodjen, u suprotnom je promasen.

Prednosti direktnog mapiranja su njegova jednostavno implementacije, ali bas to moze da dovede do gubljenja performansi, jer se gubi mogucnost lokalnosti.

5.4 Asocijativno Mapiranje

Memorijsko blok moze da se postavi na bilo kojoj liniji kesa. Sto omogucava maksimalnu fleksibilnost.

ADRESA: | TAG | POMERAJ |

LINIJA KESA:
| dirty_bit | valid_bit | TAG | PODACI |

Da bi pronasli memorijski blok u kesu potrebno je uporedjivati TAG polja. To znaci da nam treba 2^c uporedjivanja, gde je 2^c broj linija kesa.

Dobra strana je fleksiblilnost, dok je losa strana sukpa pretraga i slozenija logika politike zamene.

5.5 Set—Asocijativno Mapiranje

Set—asocijativno mapiranje je nesto izmedju direktnog i asocijativnog. Deli linije kesa u odvojene skupove. Ukoliko ti skupovi sadrze po n linija, taj kes nazivamo *n-trostruki set-asocijativni kes* (eng. *n-way set associative cache*). Mapiranje memorijskog bloka je isto kao i kod direktnog mapiranja, samo sto se ovde mapiranje vrsi na skupove. Memorijski blok moze da se smesti u bilo koju liniju kesa unutar dodeljenog skupa.

ADRESA: | TAG | BROJ_SKUPA | POMERAJ |

LINIJA KESA:
| dirty_bit | valid_bit | TAG | PODACI |

BROJ_SKUPA: Oznacava redni broj skupa i ima $s = \log_2 S$ bita, gde je S broj skupova

5.6 Politika Zamene

Politika zamene definise nacin na koji se bira linija kesa koja ce biti zamenjena i veoma je vazna zbog dobrog iskoriscenja vremenske lokalnosti. Kod direktnog mapiranje nema potebe za politiku zamene. U Slucaju asocijativnog mapiranja politika zamene treba da odabere jednu kes liniju iz skupa svih kes linija u kesu. U slucaju set—asocijativnog mapiranja, potrebno je odabrati jednu kes liniju iz odgovarajuceg skupa kes linija u koje dati podatak ide.

LRU (Least Recently Used): Zamenjuje se ona linija kesa koja najduze nije bila koriscena. Najbolji rezultati, najteza implementacija. Za n kes linija moze se implementirati konacni automat sa $n!$ stanja. Problem je sto $n!$ brzo raste.

Pseudo—LRU: Aproksimacija LRU politike koja se manje precizna, ali se lakse implementira. Neka imamo n linija kesa. Ovaj skup delimo na dve polovine. Jednim bitom odredjujemo u kojoj se od te dve polovine nalazi linija kesa kojoj je najskorije pristupano. Na slican nacin se deli svaka od ovih polovina.

FIFO (First In First Out): One linije kesa koje su u kesu najduze najmanje se koriste, dok se one koje su u kesu stigle kasnije vise koriste. Jednostavna je implementacija, dovoljno je cuvati brojac koji predstavlja odredjenu liniju kesa, i koji se svaki put uvecava kada se doda nova linija kesa.

LFU (Least Frequently Used): Najredje koriscena linije se izbacuje iz kesa. Potrebno je da postoji brojac za svaku liniju kesa, koji se uvecava pri svakom pristupu te linije kesa, kao i slozena logika koja je potrebna za uporedjivanje brojaca. Implementacija je dosta slozena.

5.7 Politika Pisanja

Ako procesor azurira neko promenljivu ili neku strukturu podata, kako se ovo resava? Imamo dve kopije podatka jedan je u kesu, a drugi u memoriji.

1. *Write-Back*: Ovde se azurira samo podatak u kesu. Kada se azurira u glavnoj memoriji? Azuriranje se vrsi za vreme zamene. Zbog toga tokom upisivanja u kesu, prvo mora da se taj podatak upise u memoriju, pa na njegovom mestu se upisuje novi memorijski blok.

   Vecina write-back keseva implementira *dirty-bit*, koji oznacava da li je linija kesa azurirana.

2. *Write-Throught*: Svaki put kada procesor azurira podatak u kesu, on ga i azurira u glavnoj memoriji.

5.8 Odluke Pri Projektovanju Kesa

Velicina kesa: Povecanjem kesa do neke granice imamo poboljsanje performansi, iznad te granice performanse opadaju.

Velicina linije kesa: Povecanjem veliine linije kesa do neke mere performanse se poboljsavaju, ali se u nekom trenutku performanse pocinju da opadaju.

Asocijativnost : Povecanjem stepena asocijativnost povecava se procenat pogodaka kesa. Potpuno asocijativni kes je u tom smislu najbolji, ali je njegova implementacija previse slozena, pa se zbog toga koriste set-asocijativno mapiranje.

6 Magistrale

Magistrala (eng. *bus*) je linija koja povezuje dva ili vise uredjaja. Prego magistrale se prenose podaci i/ili instrukcija izmedju dva uredjaja. Pod *transakcijom* na magistrali podrazumevamo bilo koju zaokruzeno aktivnost na magistrali. Svaka transakcija moze ukljucitati jednu ili vise *operacija* na magistrali (citanje, pisanje, ...). U svakom trenutnu se putem magistrale moze obavljati najvise jedna transakcija. U svakoj transakciji imamo dva uredjaja: jedan je aktivni *glavni* (eng. *master*), koji inicira transakciju, a drugi je pasivni *sporedni* (eng. *slaves*), koji odgovara na zahtev. Komunikacija izmedju uredjaja pri svakoj transakciji regulisana je skup pravila koje nazima *protok magistrale* (eng. *bus protocol*).

Magistrale mogu biti *unutrasnje*, koje sluze za razmenu podataka izmedju razliticith uredjaja u procesorskom sistemu, i mogu bit *spoljanje*, koje sluze za razmenu podataka izmedju procesora i memorije ili ostalih I/O uredaja.

Da bismo povezali vise uredjaja na magistralu, potrebno je spreciti istovremeno pustanje signala od strane razlicitih uredjaja. Ovo se postize koriscenjem bafera sa tri stanja (eng. *tristate buffer*). Odredjuju da li ce ulaz biti prosledjen na izlaz ili ce na izlazu biti vrednost visoke impedanse.

6.1 Serijske i Paralelne Magistrale

Serijske magistrale se sastoje od jedne linije preko koje se podaci prenose bit po bit. *Paralelne* magistrale se sastoje od vise linije preko kojih se prenose podaci rec po rec.

Postoje dva problema sa paralelnim magistralama:

1. *Iskprivljenost* (eng. *bus skew*) — ne stizu svi bitovi na odrediste u isto vreme. Pri velikim frekvencijama ovo je dosta izrazeno.
2. *Interferencija* — pri velikoj frekvenciji nastaje elektromagnetna indukcija, sto dovodi do toga da signali sa razlicitih paralelnih linija uticu jedni na druge.

Zbog tih razloga serijske magistrale su dosta brze i vise se koriste.

6.2 Vrste Paralelnih Magistrala

- *Multipleksirane* (eng. *multiplexed*) — iste linije se koriste i za adrese i za podatke tako sto se adrese i podaci prenose u razlicitim ciklusima
- *Razdvojene* (eng. *Dedicated*) — Odvojene linije se koriste za adrese, podatke i kontrolne signale (*magistrala podataka, adresna magistrala, kontrolna magistrala*).

6.3 Sirina Paralelne Magistrale

Broj bitova adrese magistrale odredjuje velicinu adresibilnog prostora. Broj bitova magistrale podataka odredjuje velicinu podataka koji se moze preneti u jednom transferu.

6.4 Vrste Transakcije na Magistrali

Citanje — Citanje jedne reci iz memorije.

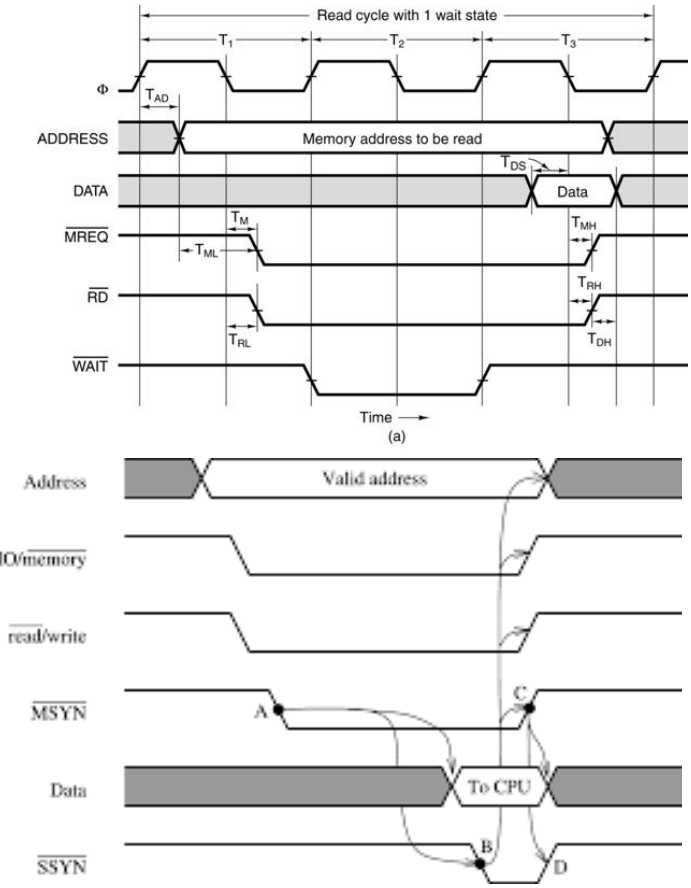
Upis — Upis jedne reci u memoriju.

Citanje bloka podataka — Citanje vise uzastopnih reci iz memorije. Podrazumeva vise ciklusa, ali je dosta brza nego citanje pojedinacnih reci vise puta.

DMA pristup — Kada je potrebno da neki I/O uredjaj pristupi memoriji, za to postoji poseban kontroler direktnog pristupa, kako se nebi prekidao rad procesora.

Read-modify-write — Omogucava atomicko citanje i upis nekog podatka u memoriju. Ova operacija ujedinjuje citanje i pisanje, i nijedan drugi uredjaj nema pristup magistrali.

6.5 Sinhronizacija Magistrale



6.6 Arbitriranje Magistralom

Sta se dogadja kada dva ili vise uredjaja istovremeno hoce da upravljaju magistralom? Mora da postoji mehanizam *arbitriranja magistralom.*

Centralizovan mehanizam ukljucuje posebna uredja *arbitar* koji u slucaju vise zahteva odredjuje kome ce magistrala biti dodeljena u skladu sa politikom dodele.

Decentralizovan mehanizam ne ukljucije nikakav uredjaj, vec se masteri izmedju sebe dogovaraju o redosledu.

6.6.1 Politika Dodele Magistrale

1. *Politika fiksiranih prioriteta*: Svaki master uredjaj ima fiksirani prioritet i u skladu sa tim prioritetom se odredjuje koji ce uredjaj dobiti magistralu.
2. *Politika rotirajucih prirориteta*: Jedna varijatna je da se prioritet uredjaja uvecava srazmerno vremenu koji je prveo u cekanju na magistralu. Druga varijanta je *kruzna politika*, (eng. *round-robin*), kod koje se uredjaju koji je upravo koristio magistralu dodeljuje najmanji moguci prioritet, cime se stavlja na kraj reda.

6.6.2 Politika Oslobadjanja Magistrale

1. *Politika bez preuzimanja* (eng. *non-preemptive*): Kada neki uredjaj dobije pristup magistrali, tada mu taj pristup ne moze boti oduzet dok on sam dobrovoljno ne oslobodi magistralu.
2. *Politika sa preuzimanje* (eng. *preemptive*): Uredjaju se moze oduzeti magistrala u slucaju da neki drugi uredjaj viseg prioriteta zahteva pristup magistrali.

6.6.3 Implementacija Arbitraze

Ulancavanje uredjaja (eng. *daisy chaining*) je implementacija u kojoj se formira jedan signal kao disjunkcija signala svih pojedinacnih uredjaja. Sa druge strane signal za dodelu magistrale se prosledjuje od jednog do drugog uredjaja. Ako je uredjaj zatrazio magistralu on onda prestaje da salje signal i ima pristup magistrali, a ukoliko nije zatrazio magistralu on signal prosledjuje dalje drugom uredjaju u nizu.

Nezavisni signal za zahtev i dodelu magistrale je implementacija u kojoj svaki uredjaj salje posevan signal za zahtev. Takodje, svaki uredjaj ima sopstveni grant signal.

Hibridni pristup je kompromis izmedju prethona dva. Uredjaji se dele u grupe. Svaka grupa ima svoj jedinstveni signal za zahtev. Uredjaji u svakoj grupi se povezuju ulancavanjem.

6.7 Elektricne Karakteristike Serijskih Signala

Za prenos signala se ne koristi jedna zica (gde $+0V$ predstavlja logicku nulu, a $+5V$ logicku jedinicu), vec se koristi jedan par zica, koje se oznacavaju sa $D-$, i $D+$. Logicka jedinica se prenosi tako sto se na $D+$ dovede pozitivan napon $+5V$, a na $D-$ dovede negativan napon $-5V$. Logicka nula se prenosi tako sto se na $D+$ dovede negativan napon $-5V$, a a $D-$ dovede pozitivan napon $+5V$. Ova tehnika se naziva *diferencijalno signaliziranje*.

6.8 Kodiranje Podataka Kod Serijskih Magistrala

1. *Povratak na nulu* (eng. *return-to-zero RZ*): Nakon svakog prenetog bita napon izmedju zica se vraca na nulu pre nego sto zapocne slanje sledeceg bita. Produzava duzinu ciklusa i ogranicava frekvenciju prenosa.
2. *Bez povratka na nulu* (eng. *non-return-to-zero NRZ*): Izmedju prenosa napon se ne vraca na nulu, vec odmah zapocinje prenos sledeceg bita. Ukoliko imamo dugacak niz istih bitova koje prenosimo, nece biti nikakve promene napona u duzem vremenskom periodu, pa moze dodji do gubljenja sinhronizacije.
3. *NRZI kodiranje* (eng. *non-return-to-zero-inverted*): Pocinjemo sa jedinicom i kodiramo dalje redom niz bitova tako da kad god naidjemo na jedinicu prepisujemo prethodni bit, a kada naidjemo na nulu prethodni bit invertujemo.

   BIN: 00110110111000111110000110000000 1
   ENC: 10001110000101111110101110101010 0

4. *8b/10b kodiranje*: Problem se javlja ukoliko neki podatak ima previse nula ili previse jedinica te prenos nije ujednacen. Nemaju dobre elektricne karakteristike pa se pokusava ostvariti DC-balans (razlika u broju jedinica i broju nula treba da bude sto bliza nuli). Ovo kodiranje resava DC-balans i sinhronizaciju. 8-bitni zapis se kodira kao 10-bitna rec.

7 Sistemi Prekida

Osnovna uloga sistema prekida je omogucavanje procesoru da reaguje na asinhrone dogadjaje koji zahtevaju hitnu obradu. Omogucava da se trenutni program prekine, da se opsluzi odgovarajuci zahtev, a da se zatim program nastavi tamo gde je stao.

7.1 Vrste Prekida

Hardverski prekid: Procesor ima jedan ili vise fizickih prikljucaka na koji se moze dovesti signal kojim se izaziva prekid. Mogu se dogoditi u bilo kom trenutku pa se jos i zovu *asinhroni*.

- *Maskirajuci* (eng. *maskable*): Procesor ima mogucnost da privremeno ignorise prekide.
- *Nemaskirajuci* (eng. *non-maskable*): Procesor nema mogucnost da ih ihnorise

Softverski prekid: Izazivaju se od strane procesora posebnim instrukcijama. Nazivaju se jos i *sinhroni*, zato sto se desavaju nda kada softver to zahteva, tj. predviljivi su.

Izuzeci: (eng. *exceptions*) Prekidi koji nastaju kao rezultat neke greske prilikom izvrsavanja neke instrukcije.

- *Greska* (eng. *faults*): Greske koje su otklonjive, pre pozivanja procedure za obradu izuzetaka stanje programa se vraca na prethodno stanje. Nakon obrade izuzetka program ce biti nastavljen sporednom instrukcijom koja nece da izazove gresku.
- *Zamke* (eng. *traps*): Greske koje su otklonjive, ali se prilikom pozivanja procedure za obradu izuzetaka stanje programa ne vraca ne menja.
- *Zaustavljanja* (eng. *aborts*): Kada dodje do greske koja nije otklonjiva i jedino smisleno resenje je zaustavljanje programa.

7.2 Obrada Prekida

Vektorski prekid: Svaki tip podataka ima svoj redni broj koji nazivamo *vektor prekida*. U slucaju prekida uredjaj na magistralu postaljva ovaj vektor i na taj nacin ga dostavlja procesoru. Procesor odrzava u memoriji *tabelu deskriptora prekida* (eng. *interrupt descriptor table, IDT*). Ova tabela sadrzi za svaki prekid adresu procedure za obradu tog prekida.

1. Prelazi se u odgovarajuci rezim privilegije.
2. Vrednost programskog brojaca i statusa registra procesora se cuva u memoriji.
3. Vektor prekida se koristi kao indeks u tabeli prekida, kako bi se odredila i pozvala procedura za obradu prekida.

8 Ulazno-Izlazni Uredjaji

Ulazni uredjaji omogucavaju onos podataka u RAM od strane drugih uredjaja.

Izlazni uredjaji omogucavaju da se podaci iz RAM-a prenesu na druge uredjaje.

Ulazno-Izlazni uredjaji omogucavaja i ulaz i izlaz.

Procesor sa U/I uredjajima komunicira preko *U/I kontrolera*. Kontroler kontrolise i obavlja posao komunikacije niskog nivoa sa uredjajima, pruzajuci procesoru unifikovani intefejs, koji se sastoji iz skupa registara:

1. *Registri podataka*: Sluze da procesor u njih upise podatke ili da iz njih procita podatke koje je uredjaj dostavio.
2. *Kontrolni registar*: Sluzi da procesor u njega upise komandu namenjenu uredjaju.
3. *Statusni registar*: Sluzi da procesor iz njega procita status U/I operacije.

8.1 Pristupanje Procesora Registrime U/I Kontrolera

1. *Memorijski mapirani U/I* (eng. *memory-mapped*): Procesor ima isti adresni prostor za U/I kontrolere kao i za glavnu memoriju. Procesor ima pristup na isti nacin kao i lokacijama u glavnoj memoriji. Koriste se iste instrukcije kao i za pristupanje u glavnoj memoriji.
2. *Izlolovani U/I*: Procesor ima poseban adresni prostor za U/I kontrolere i njihove registre. Za pristup to adresnom prostoru koriste se posebne instrukcije, kojima se zadaje adresa u okviru tog adresnog prostora..

8.2 Komunikacija Izmejdu Procesora i U/I

8.2.1 Programiranje U/I

1. Najpre proverada da li je uredjaj zauzet ili ne. Ukoliko jeste procesor ceka dok se uredjaj ne oslobodi.
2. Procesor upisuje u registre podatke, a zatim u komandni registar upisuje odgovarajucu komandu.
3. Procesor ceka da se obavi operacija.
4. U slucaju ulazne operacije procesor prebacuje procitani podatak iz registra podataka.

8.2.2 U/I Vodjen Prekidima

Procesor ne ceka da U/I uredjaj zavrsi sa radom, vec samo izda naredbu i nastavi da radi nesto drugo. Kada U/I obavi svoj posao, salje signal procesoru da je zavrsio i u slucaju ulaznog uredjaja on preuzima ulazne podatke.

8.2.3 Direktan Pristup Memoriji

Kod *direktnog pristupa memorija* (eng. *Direct Memory Access, DMA*) postoji poseban *DMA kontroler* koji obavlja posao transfera umesto procesora.

1. Procesor posalje DMA kontroleru broj koji identifikuje U/I uredjaj sa kojim zeli da komunicira, pocetnu adresu u memoriji, broj bajtova za transfer, kao i da li je u pitanju citanje ili pisanje. Nakon toga procesor izdaje komandu DMA kontroleru da zapocne transfer.
2. DMA kontroler zahteva pristup magistrali. DMA kontroler postablja adresu na adresnu magistralu, a zatim salje kontrolne signale U/I uredjaju i memoriji, cime obezbedjuje da se podatak prenese preko magistrale podataka.
3. DMA kontroler salje signal za prekid procesoru, cime ga obavestava da je transfer zavrsen.

9 Virtualna Memorija

Virtuelna memorija podrazumema da svaki program ima sopstveni virtualni adresni prostor fiksirane velicine. Svaki program misli da je sam na tom racunaru i da je sva njemu vidljiva virtualna memorija njegova.

Neki delovi programa ce biti prisutini u fizickoj memoriji. Drugi delovi ce biti prisutni na disku, ali ne i u operativnoj memoriji. Kada program zahteva pristup nekoj virtualnoj adresi, operativni sistem proverava da li se odgovarajuci podatak nalazi na toj adresi. Ako se on nalazi na adresi izvrsice se prevodjenje iz virtualne u fizicku adresu. Ukoliko se podatak ne nalazi na toj adresi, tada se on dohvata sa diska i smesta u operativnu memoriji.

- Program moze da ima znatno vecu virtualnu memoriju na raspolaganju nego sto ima fizicke memorije.
- Sprecava se mogucnost da jedan program pristupi drugom programu ili operativnom sistemu.
- Problem se javlja kada neke delove memorije koriste vise programa (np. biblioteke).
- Funkcija prevodjenja virtualnih u fizicke adrese ne mora biti linearna, tj. susedne lokacije u virtualnoj memoriji ne moraju biti susedne lokacije u virtualnoj memoriji.

9.1 Nacin Organizacije Virtualne Memorije

Virtualni adresni prostor je podeljen na *stranice* (eng. *pages*) jednake velicine.

Fizicka memorija je takodje podeljena na *okvire* (eng. *page frames*) jednake velicine.

Virtualnu adresu treba prevesti u fizicku adresu:

BROJ STRANICE	POMERAJ
20 bit	12 bit

BROJ OKVIRA	POMERAJ
20 bit	12 bit

Postize se pomocu *tabele stranica* (eng. *Page Tables*). Sastoji se iz *stavik* (eng. *Page Table Entry, PTE*). Za svaku stranicu postoji jedna stavka, pa se broj stranice koristi kao indeks u tabeli. Tabela se nalazi u fizickoj memoriji, a njena adresa se cuva u posebnom registru procesora.

Prilikom pristupanja virtualnoj adresu, procesor automatski pronalazi stavku i tabeli stranice. Ukoliko je stranica prisutna u RAM-u, tada broj okvira i pomeraj formiraju fizicku adresu. U suprotom, dolazi do izuzetka koji se naziva *greska stranice* (eng. *page fault*) Ovaj izuzetak automatski pokrece izvrsavanja funkcije za obradu prekida, i prebacuje podatke sa diska u RAM, azurira tabelu stranica i vraca kontrolu programu.

Kada vise nije moguce alocirati nove okvire na stranice vrsi se *zamena stranica*. Stranica se izbacuje iz fizicke memorijem, sacuvava se na disku, a okvir oslobadja za drugu stranicu.

Za politiku zamene se koristi varijanta pseudo-LRU strategije, procesor najcesce odrzava informaciju o tome koje su stranice koriscene, a koje ne. Na osnovu toga izbacuje odgovarajuce stranice.

Za politiku pisanja se koristi write-back strategija: Uz svaku stranicu se u PTR cuva dirty-bit koji procesor automatiski azurira prilikom promene sadrzaja stranice u RAM-u. Prilikom zamene stranice, ako je bilo promena, stranica ce biti sacuvana na disku, a u suprotnom nece, jer je kopija koju vec imamo na disku azurirana.

9.2 Stranicenje Na Vise Nivoa

Umesto da tabela stranica mora da zauzima 1024 uzastopnih okvira u RAM-u, mozemo dozvoliti da se i samo okviri raspse po memoriji u proizvoljne okvire. Da bismo znali gde se koja stranica nalazi, uvodi se dodatna tabela koja se zove *direktorijum tabele stranice* (eng. *Page Directory*). Ovaj direktorijum sadrzi *stavke* (eng. *Page Directory Entry, PDE*). Svaka stavka sadrzi broj fizickog okvira u kome se nalazi odgovarajuca stranica tabele stranica.

BROJ DIR	BROJ STRANICE	POMERAJ
10 bit	10 bit	12 bit

9.3 Velicina Stranice

Velike stranice resavaju probelm velikih tabela stranica, i efikasnije koriscenje diska.

Lose strane velikih stranica su to sto mogu postojati neke koje su polu prazne i bez razloga koriste prostor na memoriji ili ako nam treba neka adresa sa neke stranice, velika je verovatnoca da nam nece trebati sve te adrese na njoj, pa takodje zauzimaju prostor bez razloga.

9.4 Translation Lookahead Buffer

Translation Lookahead Buffer (TLB) je specijalni kes u procesoru koji skladisti delove tabele stranica i koristi se kako bi se brze pronasla adresa na fizickoj memoriji. On je mali i veoma brz kes, obicno je razdvojen. Kod manjih se koristi potpuno asocijativno mapiranje, a kod vecih se koristi set-asocijativno mapiranje. U slucaju da se adresa ne nalazi na TLB odlazi se u memoriji u prebacuje se iz nje.

10 Neke Napredne Tehnologije Kod Savremenih Procesora

10.1 Preklapanje Instrukcija

Preklapanje instrukcija (eng. *pipelining*) omogucava izvrsavanje vise instrukcija odjednom svaka u razlicitoj fazi. Pretpostavimo najjednostavniji slucaj, da u svakoj instrukciji treba po jedan ciklus za svaku od gore navedenih faza izvrsavanja. To znaci da ce se svaka instrukcija izvrsiti u 4 ciklusa.

```
Sekvencijalno:
    |-|-|-|-|-|-|-|-|-|-|-|-|
1     F D E W
2             F D E W
3                     F D E W

Pipelining:
    |-|-|-|-|-|-|-|-|-|-|-|-|
1     F D E W
2       F D E W
3         F D E W
4           F D E W
5             F D E W
6               F D E W
7                 F D E W
8                   F D E W
9                     F D E W
```

Ali ovo je idealan slucaj. U praksi i nije bas uvek tako. Moze doci do poremetanja "pokretne trake" tako sto neke instrukcije traju duze od jednog ciklusa.

```
Pipelining:
    |-|-|-|-|-|-|-|-|-|-|-|-|
1     F D E W
2       F D E W
3         F D E E E E W
4           F D         E W
5             F         D E W
6                       F D E W
```

Moze se resiti tehnikom *prethodnog dohvatanja* (eng. *prefetching*). Kod ove tehnike postoji bufer koji moze primiti veci broj instrukcija. U ovaj bafer se dohvata vise instrukcija odjednom, tako da proceosr u svakom trenutu ima vise narednih instrukcija na raspolaganju.

10.2 Izvrsavanje Van Redosleda

Izvrsavanje van redosleda (eng. *out-of-order execution*). Procesor moze utvrditi da dve ili vise instrukcija ne zavise jedna od druge, pa im moze promeniti redosled.

```
add  esi, edi        add  esi, edi
mov  eax, x   --\    add  ecx, edx
add  ecx, edx --/    mov  edx, esi
mov  edx, esi        mov  eax, x
```

Ukoliko imamo instrukcije uslovnog koka, tada ne znamo tacno koje ce instrukcije slediti nakon skoka. Ukoliko ucitamo pogresan niz instrukcije, sve sto je uredjeno sa njima bice odbaceno, a pokretna traka bice zaustavljena dok se ne ucitaju prave instrukcije. Ovo se naziva *kazna skoka* (eng. *branch penalty*).

Ovakve situacije koje mogu dovesti do zaustavljanja pokretne trake usled pogresno ucitanih instrukcija prilikom uslovnog skoka nazivaju se *rizici kontrole* (eng. *control hazards*).

Rizici kontrole se razresavaju *predikcijom grananja* (eng. *branch prediction*). Ideja je da se unutar hardvera prilikom dohvatanja instrukcije skoka nekako predvidi koji ce rezultat biti, i da se za njom dohvataju odgovarajuce instrukcije. Obicno se koristi *dinamicko predvidjanje*, gde se koriste rezultati prethodnih n izvrsavanja te instrukcije da se predvidi sta ce se desiti u sledecem izvrsavanju. Ova jednostavna tehnika dobra je u 90% slucajeva.

10.3 Superskalarno Izvrsavanje

Superskalarno izvrsavanje podrazumeva postojanje veceg broja odredjenih funkcionalnih jedinica u kojima se in-

strukcije izvrsavaju. Znaci da vise instrukcija mogu istovremeno biti u fazi izvrsavanja.

11 Verilog

11.1 Osnovni Tipovi Podataka u Verilog

Podaci predstavljaju signale koji se prenose kroz kolo. Svaki signal moze biti jednobitni ili visebitni. Visebitni signali se zovu *vektori*. Svaki signal moze imati sledece vrednosti

- 0 — logicka nula
- 1 — logicka jedinica
- x — nedefinisana vrednost
- z — vrednost visoke inpedanse

Dva osnovna tipa podataka su *zice* (eng. *net types, wires*) i *registri* (eng. *registers*).

Na svaku zicu se moze povezati izlaz nekog kola A, i ulaz nekog kola B. Vrednost zice je uvek odredjena vrednoscu signala koji se na nju dovodi.

```
wire x;
wire [7:0] y;
wire [31:0] z;
wire [0:7] x;

reg x;
reg [7:0] y;
```

Integer — 32-bitni reg tip, vrednost se tumaci kao oznaceni ceo broj u potpunom komplementu.

Time — 64-bitni reg tip, cuva informaciju o proteklom vremenu, tumaci se kao neoznaceni broj.

Vektor je jedan podataka koji se sastoji iz vise bitova. Niz je sekvenca vise podataka

```
wire x[0:99]; // niz od 100
              // jednobitnih signala
reg [7:0] y[0:255]; // niz od 256
                    // 8-bitnih podataka
reg [7:0] z[0:99][0:256] // 2d niz
```

11.2 Izrazi i Operatori u Verilogu

- Operator indeksovanja ([ ]).
- Operator za pristup ugnjezdenim imenima (.)
- Aritmeticki operatori $(+, -, *, /, \%)$
- Logicki operatori $(!, \&\&, ||)$
- Relacioni operatori
- Bitovski operatori
- Operatori pomeraja
- Operatori redukcije
- Operatori grupisanja $(\{\}, \{\{\}\})$
- Uslovni operatori (?:)

11.3 Koncept Modula

Modul predstavlja osnovnu jedinicu dizajna i tipicno predstavlja hardversku komponentu koju zelimo da napravimo. Svaki mogul moze imati ulazne i izlazne signale. Manji moguli se mogu instancirati u okviru vecih modula.

```
module <ime_modula>(<lista_portova>);
<deklaracije_portova>;
// kod
endmodule

module my_module(x, y, z);
input [3:0] x;
output y;
inout z;
```

Modul se moze instancirati u okviru drugog modula:

```
<ime_modula> <ime_instance>(<signali>);
```

11.4 Modelovanje Na Nivou Gejtova

```
wire x, y;
wire z;
and(z, x, y); // and kolo
not(z, x);    // not kolo
buf(x, y);    // baferi koji prosledjuju
              // vrednost y i salju je na x
              // simulacije kasnjenja
bufif0(x, y, e);
bufif1(x, y, e);
notif0(x, y, e);
notif1(x, y, e);
```

Logickim kolima se moze zadati kasnjenje po zelji:

```
and #(5) and(z, x, y);
```

- #n — kasnjenje od n vremenskih jedinica
- #(n) — kasnjenje od n vremenskih jedinica
- #(n, m) — kasnjenje n se odnosi na prelaz ja 0 na 1, a kasnjenje m se odnosi na prelaz sa 1 na 0
- #(n, m, k) — kasnjenje n se odnosi na prelaz ja 0 na 1, a kasnjenje m se odnosi na prelaz sa 1 na 0, kasnjenje k se odnosi na prelazak u stanje visoke impedense.

11.5 Modelovanje Na Nivou Protoka Podataka

```
wire [3:0] x;
wire y, z;
wire [2:0] p;

assign x = {y|~z, {y,{2{z}}}^~p};
```

Sa desne strane se formiran 4-bitni signal, koji se pridruzuje podatku x. Ova naredba se zove naredba *kontinuirane dodele*. Signal (zica) x trajno se povezuje na izlaz kola koje izracunava signal na desnoj strani.

Naredba assign takodje dozvoljava definisanje kasnjenja:

```
assign #5 x = y;
```

Umesto naredbe assign moguce je koristiti inicijalizaciju prilikom deklaracije wire podataka. Sledece stvari su ekvivalentne:

```
wire x = y | z;

wire x;
assign x = y | z;
```

11.6 Modelovanje Ponasanja

11.6.1 Procesi

Postoje dva osnovna tipa procesa: inital i always

Initial proces je proces koji se izvrsava samo jednom, pocev od nulte vremenske jedinice u radu simulatora. Sintaksa ovog procesa je:

```
initial
begin
    <naredbe>
end
```

Always proces je proces koji se iznova pokrece cim se zavrsi i tako dokle god traje simulacija.

```
always
begin
    <naredbe>
end
```

Dodavanje kasnnjenja u naredbe u procesu:

```
initial
    clk <= 0;

always
    #20 clk <=~clk;
```

Specifikacija dogadjaja koji aktivira proces:

```
always @(p or q)
begin
    p <= q;
    q <= p;
end
```

11.6.2 Naredba Proceduralne Dodele

Postoje dve vrste naredbe proceduralne dodele: blokirajuce i neblokirajuce.

Blokirajuce se izvrsavaju tako sto se izracuna izraz na desnoj strani, a zatim se azurira vrednost promenljive na levoj strani.

```
<promenljiva> = <izraz>;
```

Naziv blokirajuca potice od toga sto se naredbe koje slede u bloku naredbi iza te naredbe nece izvrsavati pre nego sto se ona ne izvrsi.

Neblokirajuca naredba ima sledecu sintaksu:

```
<promenljiva> <= <izraz>;
```

Ova naredba se izvrsava u dve etape: Najpre se izracuna izraz na desnoj strani, ali se izracunata vrednost ne upisuje odmah u promenljivu na levoj strani. Umesto toga, vrednost izraza se zapamti, a nastavlja se dalje sa sledecom naredbom u bloku. Na kraju tekuce vremenske jedinice, kada se aktivni red isprazni, vrsi se azuriranje promenljive sa leve strane. Ova naredba ne blokira izvrsavanja narednih naredbi u bloku.

11.6.3 Vremenska Kontrola Naredbi I Procesa

```
#<broj>
```

Odlaze izvrsavanje naredbe ili procesa za <broj> vremenskih jedinica.

```
inital #20
begin
    #10 x <= y;
end
```

Drugi tip vremenske kontrole je zasnovan na promeni vrednosti signala po zelji.

```
@(<uslov>)
```

Primer simulacija T flip-flopa:

```
always @(pasedge clk)
begin
    q <=~q;
end
```

Postoji jos i sintaksa @* ili @ (*) sto je skraceni zapis za @ (x1 or x2 or ...or xn)

Treci tip vremenske kontrole je:

```
wait(<uslov>) <naredba ili blok>
```

Primer brojaca

```
integer x;

initial
begin
    x <= 0;
    wait (clk) x <= x + 1;
end
```

11.6.4 Kontrola Vremena Unutar Dodele

```
x <= #10 y; // desna strana se izracuna
            // odmah dok se upis odlaze

#10 x <= y; // odlaze kompletnu naredbu
```

11.6.5 Naredba Grananja

```
if(<uslov>)
    <naredba>
else
    <naredba>

case(<izraz>)
    <lista_vrednosti>: <naredba>
    <lista_vrednosti>: <naredba>

    default: <naredba>
endcase
```

11.7 Petlje u Verilogu

```
while(<uslov>) <naredba>

for(<init>; <uslov>; <inc>) <naredba>
```