- Virtuelne mašine
  - VMM mora imati mogućnost da
    - omogući izvršavanje guest OS-a kao da je na pravom hardveru
    - onemogući direktno zauzimanje resursa host sistema
      - presretanje svih operacija koje mogu dovesti do ovoga: prekidi, sistemski pozivi, rad sa virtuelnom memorijom, izuzeci
        - neophodnost postavljanja zamke (trap; prekidna tačka) za sve ovakve naredbe
        - podrška host CPU-a
  - Uvođenje još jednog nivoa virtuelne memorije
    - stranice virtuelne memorija unutar guest OS-a se mapiraju na fizičke stranice unutar guest OS-a; ove “fizičke” stranice su za VMM realne stranice (real page)
    - VMM zatim mapira realne stranice na fizičke stranice host sistema
      - može se ubrzati ako postoji podrška u procesoru
  - Virtuelizacija I/O uređaja - najteži deo
    - diskovi su npr. najčešće fajlovi na host sistemu

- Virtuelne mašine
  - Podrška u današnjim procesorima
    - ubrzanje rada sa vitruelnom memorijom (tabele stranica i njihovo keširanje)
    - ubrzanje I/O operacija (prekidi i DMA)
  - Značajno povećani zahtevi za sigurnošću zbog sve veće upotrebe cloud sistema
    - često se ne zna šta se sve fizički izvršava na istom računaru na kome je i vaša VM
  - Paravirtuelizacija
    - Manje izmene guest OS-a kako bi bolje koristio resurse host računara
      - npr. direktno zauzimanje stranica memorije od host OS-a
      - Izmena Linux kernela da koristi Xen je na oko 3000 linija koda, što je oko 1% x86 koda
    - Korišćenje više nivoa privilegija

- Virtuelne mašine
  - Pwn2Own 2017, tim Qihoo 360, prvi zvanični virtual machine escape napad, $105k
    - Situacija: Microsoft Edge browser, izvršavan na Windows-u 10, unutar VMWare virtuelne mašine, sav softver sa najnovijim patch-evima
    - Edge je poslat na posebno pripremljen sajt
    - Kod na sajtu je iskoristio bag u JavaScript implementaciji, kako bi izašao van sandbox okruženja i izvršio kod na guest OS-u (Windows 10)
    - Zatim je iskorišten bag u kernelu Windowsa i izvršen je kod na sistemskom nivou
    - Nakon toga je iskorišten bag u VMWare-u i izvršen je kod u host operativnom sistemu!
    - Ceo napad je trajao oko minut i po. Sve što je bilo potrebno je da se otvori web stranica...

- Paralelizam unutar procesora
  - ILP - Instruction-Level Parallelism
  - Većina procesora napravljenih posle 1985 koristi protočnu strukturu (pipeline)
  - Generalno dve vrste
    - ILP zasnovan na hardveru (dinamički ILP)
    - ILP zasnovan na pomoći kompajlera (statički ILP, određuje se u vreme prevođenja programa)
      - pokazalo se da je upotrebljivo samo za specifične primene paralelizma zasnovanog na podacima
  - Za procesor sa protočnom strukturom, broj ciklusa po instrukciji (cycles per instruction):

    $Pipeline_{CPI} = IdealPipeline_{CPI} + StructuralStalls + DataHazardStalls + ControlStalls$

- Paralelizam unutar procesora
  - Bazni blok (basic block) je sekvencijalni niz naredbi bez grananja (osim ulaska i izlaska iz bloka)
  - Mogućnosti za paralelizam unutar jednog baznog bloka su relativno male
    - najviše zbog toga što su naredbe u bloku često međuzavisne
    - za RISC, veličina bloka je u proseku 3-6 naredbi
  - Za bitnije unapređenje, paralelizam se mora tražiti preko više baznih blokova
    - jedan od osnovnih je paralelizam između različitih iteracija petlje

- Paralelizam unutar procesora
  - loop-level parallelism

```
for (i=0; i<=999; i=i+1)
    x[i] = x[i] + y[i];
```

    - sve iteracije se mogu odraditi u paraleli
      - u okviru pojedinačne iteracije maltene nema mesta paralelizmu
    - loop-unrolling - razvijanje petlje u pojedinačne iteracije
      - statički, od strane kompajlera
      - dinamički, od strane procesora
    - SIMD naredbe
      - sabiranje dva vektora za gornji primer
  - Međuzavisnost podataka
    - Bitno je utvrditi da li/kako jedna naredba zavisi od druge, kako bi se znalo koje naredbe mogu ići u paraleli (ili delimično u paraleli)

- Paralelizam unutar procesora
  - Međuzavisnost podataka
    - posmatraju se sekvencijalne naredbe sa oznakama i,j,...
    - generalno 3 vrste zavisnosti naredbe j od naredbe i:
      - data dependences (true data dependences)
      - name dependences
      - control dependences
  - Međuzavisnost podataka (data dependences, true data dependences)
    - naredba i generiše rezultat koji može biti korišten u naredbi j
    - naredba k zavisi od naredbe j, a naredba j zavisi od naredbe i (dependence chain)

- Paralelizam unutar procesora
  - Međuzavisnost podataka (data dependences, true data dependences)
    - ako je jedna naredba na ovaj način zavisna od druge, ona moraju biti izvršene u redosledu i mogu se eventualno delimično preklapati (druga naredba se može izvršiti do momenta kada rezultat prve postane neophodan)
    - ako procesor krene da paralelno izvršava dve ovakve naredbe, može se desiti zastoj u protočnoj strukturi; procesor treba da detektuje ovakve situacije
      - kod procesora kod kojih je za paralelizam odgovoran kompajler, uloga kompajlera je da naredbe rasporedi tako da do ovakvih situacija ne može da dođe
    - ovakve zavisnosti su posledica zavisnosti podataka u izvornom kodu i praktično su osobina programa
      - i samim tim određuju i granice paralelnog izvršavanja naredbi unutar procesora

- Paralelizam unutar procesora
  - Međuzavisnost podataka (data dependences, true data dependences)
    - dva osnovna načina rešavanja su:
      - očuvanje međuzavisnosti, uz detektovanje zastoja u protočnoj strukturi
      - eliminisanje međuzavisnosti putem transformacije koda
    - između dve nardebe, podatak može ići od jedne do druge na dva načina:
      - preko registra
      - preko memorijske lokacije
    - detekcija međuzavisnosti
      - preko registara je jednostano - ako je isti registar
      - preko lokacije je dosta teže
        - 100(R1) i 20(R2) mogu biti ista lokacija (a ne moraju)
        - 100(R1) u jednom trenutnku može biti različito od 100(R1) u nekom narednom trenutku, a može biti i isto

- Paralelizam unutar procesora
  - Međuzavisnost imena (name dependences)
    - dešava se kada dve naredbe koriste isti registar ili memorijsku lokaciju (ime), ali ne postoji tok podataka vezan za taj registar, odnosno lokaciju
    - generalno dve vrste; ako naredba i prethodi naredbi j:
      - anti-zavisnost (antidependence) je kada naredba j upisuje u ime koje naredba i čita
        - redosled se mora zadžati, da bi naredba i pročitala korektnu vrednost
      - izlazna zavisnost (output dependence) je kada obe naredbe upisuju u isto ime
        - redosled se mora zadžati, kako bi na kraju ostala vrednost koju je upisala naredba j
    - iako nema prenošnja vrednosti sa jedne naredbe na drugu, u oba slučaja se redosled mora zadržati
    - može se rešiti reimenovanjem - izmenom registra ili memorijske lokacije za jednu od naredbi, što omogućava njihovo paralelno izvršavanje; reimenovanje može odraditi kompajler (statičko) ili procesor (dinamičko)

- Paralelizam unutar procesora
  - Opasnosti u radu sa podacima (data hazards)
    - postoji uvek kada postoji međuzavisnost podataka ili imena između naredbi; preklapanje izvršavanja takvih naredbi može dovesti do nekorektnog izvršavanja programa
    - da bi se izbegla opasnost od nekorektnog izvršavanja, mora se očuvati redosled izvršavanja (program order), odnosno redosled izvršavanja međuzavisnih naredbi mora biti isti kao kada se program izvršava isključivo sekvencijalno
      - treba naći situacije kada je ovo neophodno raditi, odnosno treba očuvavati redosled izvršavanja samo ako to utiče na rezultat rada (dela) programa
    - ako posmatramo dve uzastopne naredbe “i” i “j” koje koriste isti resurs, imamo tri vrste opasnosti kod paralelnog izvršavanja:
      - čitanje nakon pisanja (RAW - read after write) - naredba j može pokušati čitanje pre nego je naredba i upisala vrednost
      - pisanje nakon pisanja (WAW - write after write) - naredba j može pokušati upis pre naredbe i
      - pisanje nakon čitanja (WAR - write after read) - naredba j može pokušati da upiše vrednost pre nego ju je naredba i pročitala

- Paralelizam unutar procesora
  - Kontrolne međuzavisnosti (control dependences)
    - dešavaju se kada izvršavanje neke naredbe zaisi od uslova nekog prethodnog grananja

```
if p1 { S1; };
if p2 { S2; };
```

      - S1 je zavisno od p1, S2 je zavisno od p2, ali ne i od p1
    - dve vrste kontrolnih međuzavisnosti:
      - naredba koja je zavisna od grananja se ne može premestiti pre grananja (čime bi se izgubila zavisnost od grananja)
      - naredba koja nije zavisna od grananja se ne može pomeriti iza grananja tako da posle toga bude zavisna od grananja
    - mogu se izvršiti i naredbe koje ne bi trebalo, pod uslovom da ne menjaju korektnost programa

- Paralelizam unutar procesora
  - Kontrolne međuzavisnosti (control dependences)
    - prilikom izvršavanja i naredbi koje ne bi trebalo, mora se voditi računa o
      - ponašanju izuzetaka (exception behavior)
      - toku podataka (data flow)
    - ponašanje izuzetaka
      - bilo kakva izmena u redosledu izvršavanja ne sme izmeniti dešavanje izuzetaka u programu; nešto slabiji uslov je da izmena redosleda izvršavanja ne sme izazvati nove izuzetke u programu

```
        add x2,x3,x4   // x2=x3+x4
        beq x2,x0,L1   // skok ako x2==x0
        ld x1,0(x2)    // x1=sadržaj adrese x2+0
    L1:
```

        - ako ignorišemo kontrolnu međuzavisnost ld i beq, pa pomerimo izvršavanje ld pre beq, može doći do izuzetka zbog pogrešnog adresiranja memorije
        - ono što se može uraditi je da se izuzetak ignoriše ako se uslovni skok izvrši (spekulativno izvršavanje)

- Paralelizam unutar procesora
  - Kontrolne međuzavisnosti (control dependences)
    - tok podataka
      - prolazak vrednosti od naredbi koje ih generišu do naredbi koje ih koriste
      - grananja ove tokove čine dinamičkim, pa je kontrolna međuzavisnost ključna za očuvanje redosleda izvršavanja

```
        add x1,x2,x3    // x1=x2+x3
        beq x4,x0,L     // skok ako x4==x0
        sub x1,x5,x6    // x1=x5-x6
    L:  ...
        or x7,x1,x8     // x7=x1|x8
```

      - or naredba je zavisna od podataka u add i sub naredbama, ali sama ta činjenica nije dovoljna da odredi korektno paralelno izvršavanje:
        - vrednost x1 u or naredbi je zavisna od toga da li je skok izvršen ili ne

- Paralelizam unutar procesora
  - Kontrolne međuzavisnosti (control dependences)
    - tok podataka
      - još jedan primer

```
        add x1,x2,x3       // x1=x2+x3
        beq x12,x0,skip    // skok ako x12==x0
        sub x4,x5,x6       // x4=x5-x6
        add x5,x4,x9       // x5=x4-x9
    skip:
        or x7,x8,x9        // x7=x8|x9
```

        - ako znamo da se vrednost x4 neće koristiti posle skip, tada se x4 može izmeniti i pre uslovnog skoka i to neće poremetiti tok podataka (i naravno pod uslovom da sub ne može generisati izuzetak)
        - u tom slučaju, ako je uslov skoka ispunjen, sub će se svejedno izvršiti, ali rezultat te naredbe nije bitan
  - RISC-V naredbe:
    - Computer Architecture - A Quantitative Approach 6th Edition, Appendix A, A-34 do A-41