Operativni sistemi

dr.sc. Amer Hasanović

Osnovne informacije

- Kontakt:
  - Zgrada Stelekt, kancelarija br 2
- Email:
  - amer.hasanovic@fet.ba
- Predmetni asistent:
  - Džemal Memić

- Po završetku kursa, studenti će:
  - poznavati koncepte koji se koriste za dizajn i funkcionisanje operativnih sistema
  - naučiti implementaciju bitnih koncepata
- Preduslovi za slušanje predmeta su ostvareni ECTS krediti iz predmeta:
  - Arhitektura računara
  - Strukture podataka

Plan

- Tokom semestra studenti će raditi na implementaciji minimalnog, ali potpuno funkcionalog, operativnog sistema za *Intel* platformu.
- OS koncepti koji će biti tretirani:
  - Prekidi;
  - Virtuelna memorija;
  - Kernel i user mod;
  - Sistemski pozivi;
  - Procesi;
  - Koordinacija procesa;

Literatura i softver

- Za praćenje predavanje preporučuju se knjige:
  - A. Hasanović, *Principi operativnih sistema kroz analizu XV6 koda*, Izdavačka kuća Hamidović, Tuzla, 2015
  - R. Cox, F. Kaashoek, R. Morris, *Xv6: a simple, Unix-like teaching operating system*, http://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf
  - A.Silberschatz, P.B.Galvin, *Operating System Concepts*, Wiley
  - A.S.Tanenbaum, *Modern operating systems*, Prentice Hall
- Software:
  - gcc;
  - gdb;
  - git;
  - Qemu

Organizacija

- Bodovanje:
  - Predispitne aktivnosti – 100 bodova
    - Lab. projekti i vježbe – 65%
    - Test tokom semestra – 35%

Skala za konacne ocjene:

Bodovi	Ocjena
50-64	6
65-74	7
75-84	8
85-94	9
95-100	10

Operativni sistem (OS)

- Sa stanovišta običnog korisnika, OS diktira broj i tip aplikacija koje će korisnik moći instalirati na računar, kao i okruženje u kojem će se koristiti instalirane aplikacije.
- Pored stvaranja okruženja za izvršavanje programa, OS ima i slijedeće zadatka:
  - Upravljanje i komunikacija sa uređajima koji su povezani na računar.
  - Apstrakcija različitih hardverskih uređaja kroz implementaciju unificiranih standardnih biblioteka koje se stavljaju na raspolaganje programima.
  - Kontrola istovremenog izvršavanja proizvoljnog broja programa bez obzira na broj raspoloživih procesorskih jezgri.

- Raspoređivanje računarskih resursa, a naročito procesora i memorije, između svih programa u izvršenju.
- Sprečavanje situacija u kojim greške prilikom izvršavanja jedne aplikacija mogu utjecati na izvršenje ostalih aplikacija.
- Formiranje fajl sistema u kojim programi na uređen način mogu trajno pohranjivati podatke.

Kernel

- Pobrojane funkcije OS-a implementira **kernel**:
  - program koji se prvi učita u memoriju računara, u memoriji ostaje rezidentantan sve do gašenja računara, a aktivira se po potrebi kada se zahtijeva neka od funkcija koju pruža.
- Za razliku od razvoja običnih programa, programiranje kernela zahtijeva intimno poznavanje:
  - procesorske arhitekture,
  - kompletnog kompajlerskog lanca,
  - principa rada svih ključnih komponenti računara, a naročito memorijske hijerarhije.
- Kerneli modernih operativnih sistema, kao što su Linux ili BSD, spadaju u red najkompleksnijih sistema koji je dizajnirao čovjek.

XV6

- Za izučavanje osnovnih funkcija operativnih sistema industrijski kerneli su suviše kompleksni, zbog čega se u ovom predmetu koristi kernel pod imenom **XV6**, razvijen od strane MIT laboratorija za paralelne i distribuirane sisteme.
- XV6 je baziran na operativnom sistemu Unix koji je 80-tih godina razvijen u kompaniji AT&T , ali uz slijedeće razlike:
  - XV6 je dizajniran za IA-32 procesorsku platformu.
  - XV6 podržava više procesorskih jezgri.
  - XV6 implementira sopstveni fajl sistem sa žurnalom.
  - XV6 podržava jednog korisnika i nema mrežni stek.

- XV6 kernel i njegove aplikacije kompajliraju se na bilo kojoj Linux distribuciji uz pretpostavku da je na datom sistemu instaliran:
  - kompajlerski lanac *gcc*,
  - sistem za praćenje i kontrolu verzija *git*
- Iako je moguće instalirati XV6 OS na konkretan hardverski uređaj (Intel platforma), radi fleksibilnosti testiranja i debagiranja XV6 kernela i njegovih aplikacija koristi se simulirani računar (virtuelna mašina) kojeg stvara i kontroliše linux aplikacija *qemu*.

Proces

- Slično kao i za originalni Unix, XV6 kernel za svaki program koji se izvršava stvara okruženje koje se naziva **proces**.
- XV6 kernel uz podršku hardvera za svaki proces osigurava virtualni procesor i memoriju, čime program koji se izvršava unutar procesa dobija iluziju da u tom trenutku samostalno kontroliše dva ključna resursa računara.
- Ukoliko u nekom od procesa nastane greška, kernel intervenira na način da zatvara dati proces bez ometanja izvršavanja ostalih aktivnih procesa u sistemu.

Intel arhitektura

Pregled i poređenje sa MIPS

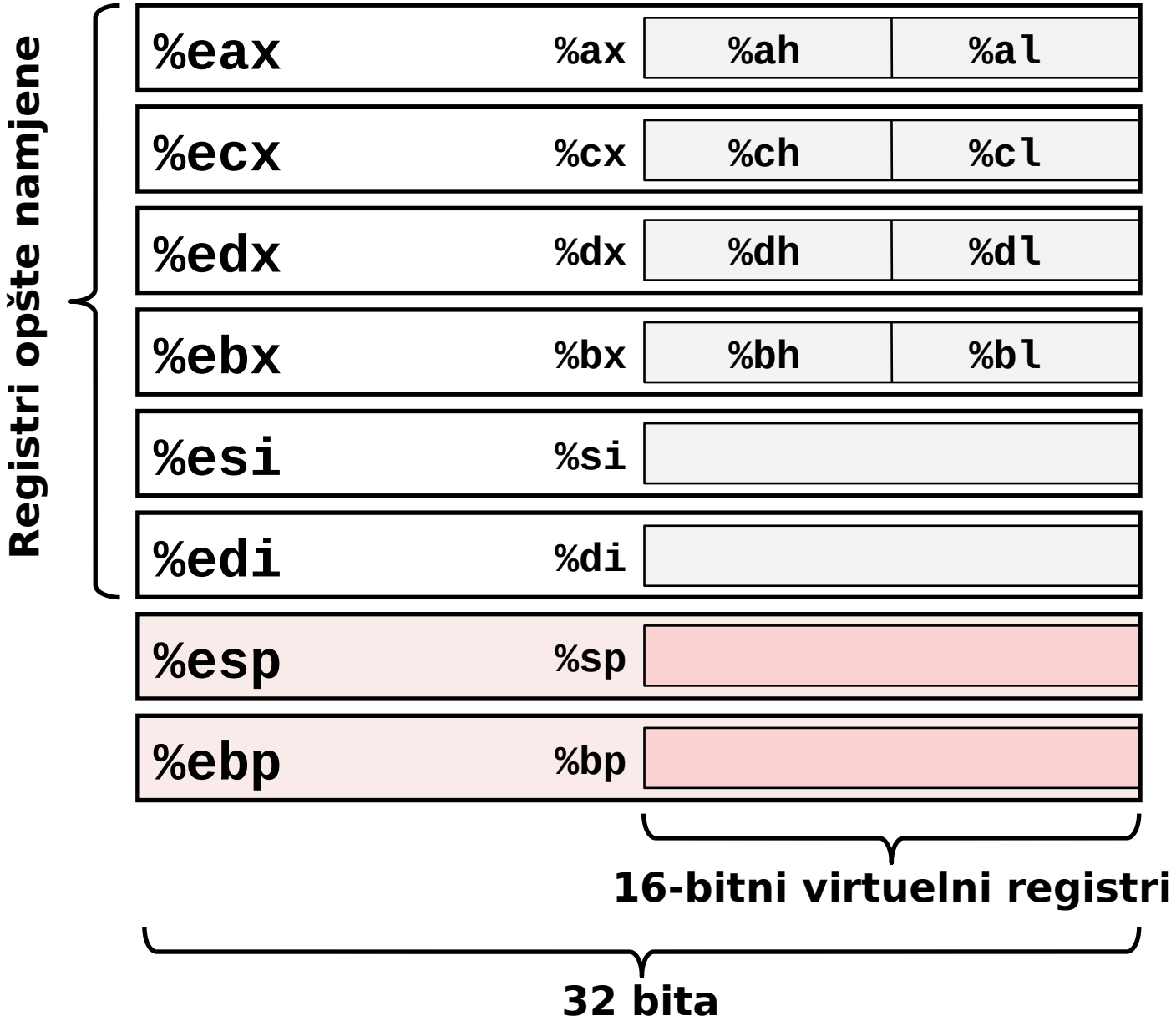
Intel x86 uvod

- 8086 – proizvodnja 1978 – 29K tranz. – 5 do 10MHz
  - 16 bitna arhitektura
  - 1MB adresni prostor
- 80386 – proizvodnja 1985 – 275K tranz. – 16 do 33 MHz
  - Prava 32 bitna arhitektura
  - Sposoban da izvršava Unix OS
- Pentium 4E – proizvodnja 2004 – 125M tranz. – 2.8 do 3.8 GHz
  - Prva 64 bitna arhitektura
- Core i7 – proizvodnja 2008 – 781M tranz. – 2.6 – 3.3 GHz

Intel razlike spram MIPS arhitekture

- CISC procesor;
  - Mali broj opštih registara (8 za IA32), veliki broj instrukcija.
- instrukcije nisu iste dužine (1B do 17B)
- pristup memoriji ne mora biti poravnat;
- najčešće dva operanda u instrukciji;
- većina instrukcija podržava jedan operand iz memorije;
- radi samo u little endian modu.

IA32 arhitektura registri



- Dodatni registri:
  - *eip*
    - Programski brojač (implictno manipularn tokom izvršenja)
  - *eflags*
    - Skup od 32 bita koji opisuju stanje procesora i rezultat operacija
  - 6 segmentnih registara
    - *cs*, *ss*, *ds*, *es*, *fs*, *gs* – svi po 16 bit
    - koriste se prilikom adresiranja, i manipulirani su od strane OS-a
  - FPU, MMX, SSE, SSE2
  - Sistemski
    - Npr *cr0*,.. *cr4*

Asembler instrukcije

- Tri klase instrukcija:
  - Aritmetičke i/ili logičke operacije nad podacima u memoriji ili registrima
  - Prebacivanje podataka između registara i memorije
  - Kontrola toka
- Dvije asembler notacije
  - AT&T (gcc) – koju koristimo u ovom kursu
  - Intel (microsoft)
- Podaci u tri veličine – gcc notacija dodaje tri sufiksa instrukcijama
  - b → byte, 8b
  - w → word, 16b
  - l → long, 32b

Operandi

- Većina instukcija *at&t* formata operira sa dva operanda i to u obliku:
  - `instrukcija op1,op2`
    - `op1` – izvor
    - `op2` – obično izvor i destinacija
  - Registri kao operandi imaju prefix % npr
    - `%eax`
    - `%ah`
  - Brojčane konstante imaju prefix `$` i koriste C notaciju, npr:
    - `$0xab`
    - `$-2`

- Memorija se može pojaviti kao operand u bilo kojem od oblika:

Oblik	Rezultat	Ime
Imm	M[Imm]	Absolute
(Ea)	M[R[Ea]]	Indirect
Imm(Eb)	M[Imm+R[Eb]]	Base+displacement
(Eb,Ei)	M[R[Eb]+R[Ei]]	Indexed
Imm(Eb,Ei)	M[Imm+R[Eb]+R[Ei]]	Indexed
(,Ei,s)	M[R[Ei]*s]	Scaled indexed
Imm(,Ei,s)	M[Imm+R[Ei]*s]	Scaled indexed
(Eb,Ei,s)	M[R[Eb]+R[Ei]*s]	Scaled indexed
Imm(Eb,Ei,s)	M[Imm+R[Eb]+R[Ei]*s]	Scaled indexed

- Gdje su:
  - `Imm` – broj
  - `Ea` – proizvoljni registar
  - `Eb` – tzv. bazni registar
  - `Ei` – tzv. indeks registar
  - `s` – tzv. faktor skaliranja (1, 2, 4 ili 8)
  - `R[Ea]` – vrijednost iz registra Ea

Prebacivanje podataka

- `mov{b,w,l} s, d`
  - Kopira podatak iz `s` u `d`
    - `s` – može biti memorija, registar ili konstanta
    - `d` – može biti memorija ili registar
  - Primjeri:
    - `movb %al,%bh`
    - `movl $10,4(%eax,%ebx)`
- `lea{w,l} s, d`
  - Slično kao `mov` samo što uzima sračunatu adresu memorijske lokacije a ne sadržaj memorije
    - `s` – može biti memorija, registar ili konstanta
    - `d` - registar

- `movz`*`xy`* `s, d` → `d=zero-extend(s)`
- `movs`*`xy`* `s, d` → `d=sign-extend(s)`
  - *x* veličina `s` : `b` ili `w`
  - *y* veličina `d` : `w` ili `l`
  - Primjeri:
    - `movzbl 4(%eax), %ebx`
    - `movswl %ax, %ecx`

Aritmetičko-logičke operacije

- `sal{b,w,l} s, d` → `d = d << s`
- `sar{b,w,l} s, d` → `d = d >> s` (predznak sačuvan)
- `shl{b,w,l} s, d` → `d = d << s` (isto kao sal)
- `shr{b,w,l} s, d` → `d = d >> s` (prazna mjesta 0)

- `and{b,w,l} s, d` → `d = d & s`
- `orl{b,w,l} s, d` → `d = d | s`
- `xor{b,w,l} s, d` → `d = d ^ s`
- `not{b,w,l} d` → `d = ~d`

- `inc(b,w,l} d` → `d = d + 1`
- `dec{b,w,l} d` → `d = d - 1`
- `neg(b,w,l} d` → `d = ^d`

- `add{b,w,l} s, d` → d = s + d
- `sub{b,w,l} s, d` → d = d – s

- `mulb s` → %ax = %al * s;
- `mulw s` → %dx:%ax = %ax * s;
- `mull s` → %edx:%eax = %eax * s;
- `imul{b,l,w} s` → isto kao mull samo sa predznakom
- `imul{b,l,w) s, d` → d = d*s;

- `cmp{b,w,l} s1, s2` → s2 – s1 (mijenja stanje eflags)
- `test{b,w,l} s1, s2` → s2 & s1 (mijenja stanje eflags)

Kontrola toka

- Gotovo sve instrukcije, u ovisnosti od njihovog rezultata, mijenjaju slijedeće bite u *eflags* registru:
  - `CF` carry flag; prekoračenje bez predznaka
  - `OF` overflow flag; prekoračenje sa predznakom
  - `SF` sign flag; negativna vrijednost
  - `ZF` zero flag; nula
- Na osnovu vrijednosti gornjih bita moguće je izvršiti uslovne skokove

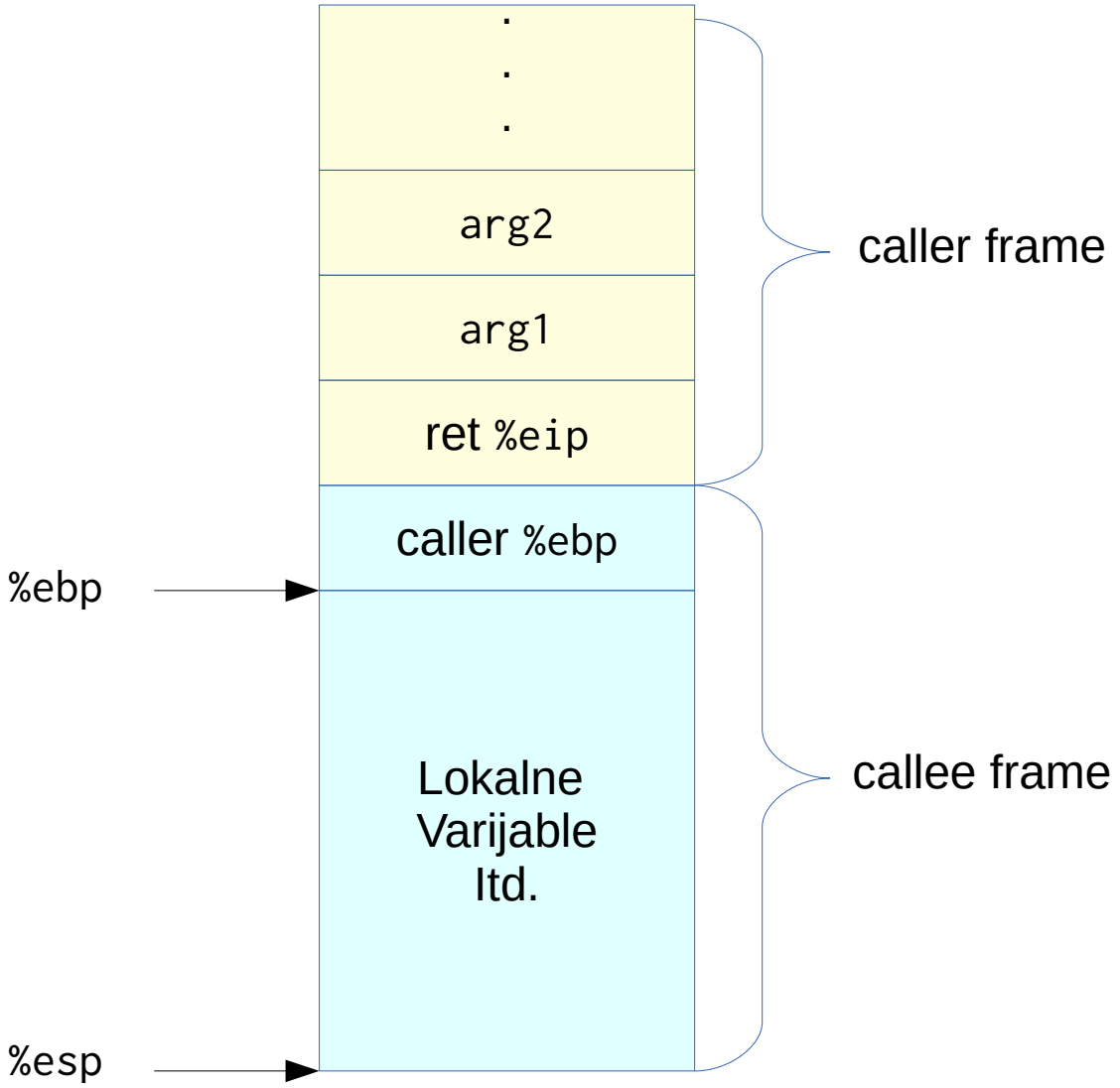
skokovi

- Bezuslovni skokovi
  - `jmp d`
- Uslovni skokovi
  - `je d` → `zero`
  - `jne d` → `nonzero`
  - `js d` → `negative`
  - `jns d` → `nonnegative`
  - `jg d` → `positive (signed >)`
  - `jge d` → `nonnegative (signed >=)`
  - `jl d` → `negative (signed <)`
  - `jle d` → `nonnegative (signed <=)`
  - `ja d` → `above (unsigned >)`
  - `jae d` → `above or equal (unsigned >=)`
  - `jb d` → `below (unsigned <)`
  - `jbe d` → `below or equal (unsigned <=).`

Funkcije i stek

- Stek segment (frame-a) funkcije označen je sa dva registra:
  - `%ebp` – pokazuje na početak;
  - `%esp` – pokazuje na kraj steka.
- Stek raste prema nižim adresama:
  - `push{w,l} s`
    - proširuje stek za 2 ili 4 bajta sa vrijednošću `s`
  - `pop{w,l} d`
    - kopira 2 ili 4 bajta podataka sa pozicije na koju pokzuje `%esp` u `d`, a zatim smanjuje stek

- Poziv funkcije f:
  - `call f`
    - push povratne adrese na stek (`pushl %eip`)
    - `%eip` postavljen na prvu instrukciju u `f`
- Povrat iz funkcije:
  - `ret`
    - Pop povratne adrese sa steka u `%eip` (`popl %eip`)
- Funkcija očekuje argumente na steku iznad svog stek segmenta
- Funkcija vraća vrijednost u registru `%eax`
- Prezervirani registri:
  - `%ebp, %ebx, %esi, %edi`
- Ne prezervirani registri:
  - `%eax, %ecx, %edx`


.
.
.
arg2
arg1
ret %eip
caller %ebp
%ebp
Lokalne
Varijable
ltd.
%esp
caller frame
callee frame

gcc -m32 -S -O0 prog.c

```
int foo(int x, int y) {
  return x+y;
}

int main() {
  return foo(10, 20);
}
```

```
foo:
    pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    movl    8(%ebp), %edx
    addl    %edx, %eax
    popl    %ebp
    ret
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $20, 4(%esp)
    movl    $10, (%esp)
    call    foo
    leave
    ret
```