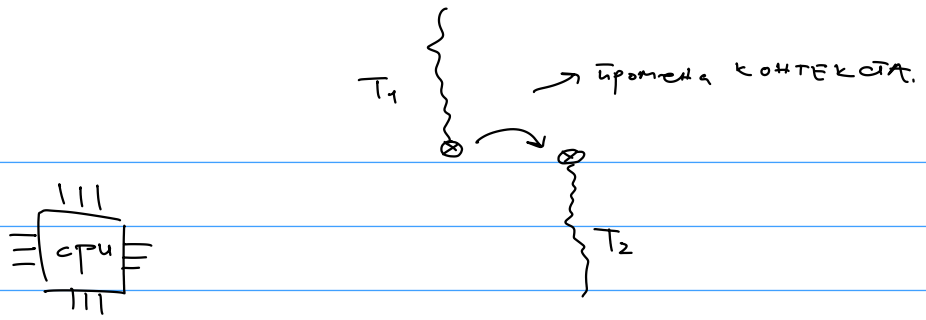
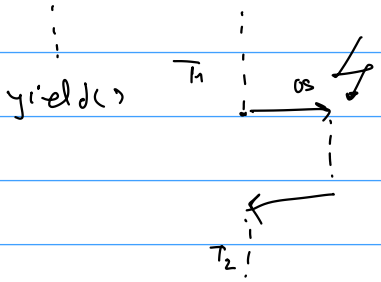


promena kontekta.

kontekst čine registri

PC, PSW



1 sinhrono yield

2 asinhrono interr.

3. (10 poena)

a)(7) Na asembleru nekog RISC procesora sa *load/store* arhitekturom, poput onog opisanog na predavanjima, napisati prevod sledeće rekurzivne C funkcije koja izračunava najveći zajednički delilac (engl. *greatest common divisor*, GCD) dva data nenegativna cela broja x i y Euklidovim algoritmom, pri čemu je uvek $x \geq y \geq 0$ (pretpostaviti da su argumenti uvek ovakvi i ispravni). Funkcija `remainder(x,y)` je implementirana i vraća ostatak pri deljenju x sa y.

```
unsigned int gcd (unsigned int x, unsigned int y) {
  if (y==0) return x;
  else return gcd(y,remainder(x,y));
}
```

x%y

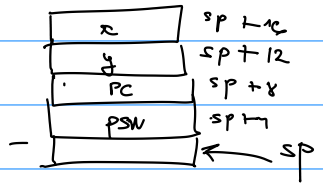
sp

b)(3) Da li je ova funkcija bezbedna za uporedne pozive iz konkurentnih niti? Obrazložiti.

Rešenje:

call gcd

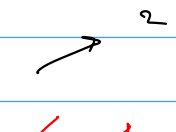
```
gcd:  ld   r1, [sp+16]      r1 := x
      ld   r2, [sp+12]      r2 := y
      jnz  r2, REC
      mv   r0, r1
      ret
REC:  push r1
      push r2
      call remainder        r0 := remainder(x,y)
      pop  r2
      pop  r1
      push r1
      push r0
      call gcd              r0 := gcd(x, remainder(x,y))
      pop  r1
      pop  r1
      ret
```



```
int x = 0;
void inc(){
    x++;
}
```

→

```
ld r0, x
inc r0
sd r0, x
```



t_1 : inc() t_2 : inc().

ld r0, x r0 := 0

inc r0 r0 = 1

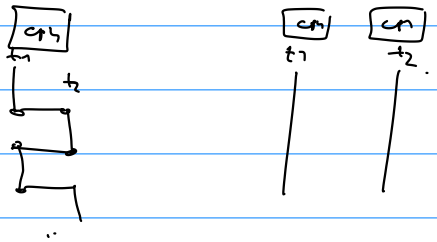
sd r0, x x := 1

ld r0, x r0 := 0

inc r0 r0 := 1

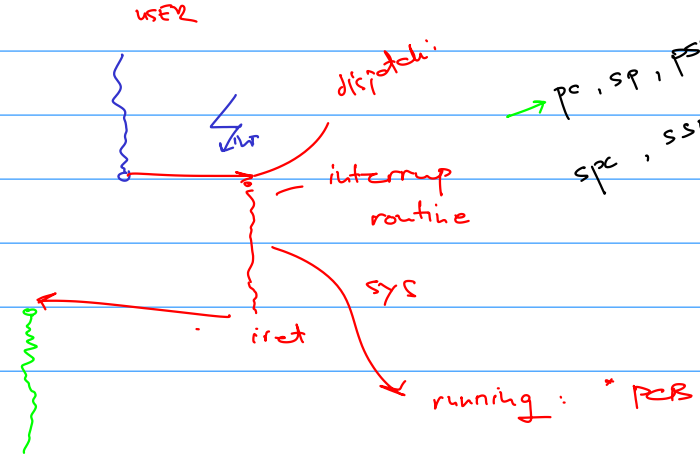
sd r0, x x := 1

Конкурентно / паралелно.



FIFO = FCFS

```
struct PCB {
    ...
    int pc;
    int psw;
    int sp;
    ...
    int reg;
    ...
}
```



2. (10 poena)

Neki procesor obrađuje prekide (hardverske i softverske) u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina), koristeći posebne registre SPC (*System Program Counter*), SPSW (*System Processor Status Word*) i SSP (*System Stack Pointer*) koji nisu dostupni u korisničkom (neprivilegovanom) režimu. Registre PC, PSW i SP, kao ni sve druge programski dostupne registre koji se koriste u neprivilegovanom režimu, procesor ne čuva nigde implicitno prilikom obrade prekida, jer se oni ni ne menjaju implicitno tokom skoka u prekidnu rutinu niti tokom izvršavanja te rutine (izvršavanje instrukcija u privilegovanom režimu koristi SPC, SPSW i SSP umesto PC, PSW i SP); njihove vrednosti se mogu menjati eksplicitno, uobičajenim instrukcijama u privilegovanom režimu (npr. `load` i `store`). Prema tome, izvršavanje u privilegovanom režimu koristi i poseban stek alociran u delu memorije koju koristi kernel. Taj stek je jedan i kernel ga ne menja.

Prilikom povratka iz prekidne rutine instrukcijom `iret` procesor ništa ne restaurira, samo se prebacuje na korišćenje registara PC, SP i PSW umesto njihovih sistemskih parnjaka SPC, SSP i SPSW.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće registre dostupne u neprivilegovanom režimu: 32 registra opšte namene (R0..R31), SP, PSW i PC. Poseban registar RX je dostupan samo u privilegovanom režimu rada procesora, pa ga korisnički procesi ne koriste i kernel ga može koristiti samo za svoje potrebe.

Registre dostupne u neprivilegovanom režimu treba sačuvati u odgovarajuća polja strukture PCB. U strukturi PCB postoje polja za čuvanje svih tih registara; pomeraji ovih polja u odnosu na početak strukture PCB označeni su simboličkim konstantama `offsPC`, `offsPSW`, `offsSP`, `offsR0`, ..., `offsR31`.

U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `scheduler`, koji nema argumente, realizuje raspoređivanje, tako što smešta PCB na koji ukazuje pokazivač `running` u listu spremnih procesa, a iz nje uzima jedan odabrani proces i postavlja pokazivač `running` na njega.

Na asembleru datog procesora napisati kod prekidne rutine `dispatch` koja vrši promenu konteksta korišćenjem potprograma `scheduler`. Ova prekidna rutina poziva se sistemskim pozivom iz korisničkog procesa, pri čemu se sistemski poziv realizuje softverskim prekidom.

Rešenje:

```
                ld  Rx, running
dispatch:       st  PC,  [Rx + offsPC]
                st  PSW, [Rx + offsPSW]
                st  SP,  [Rx + offsSP]
                st  R0,  [Rx + offsR0]
                 ⋮
                st  R31, [Rx + offsR31]
                call scheduler
                ld  Rx, running
                ld  PC,  [Rx + offsPC]
                ld  PSW, [Rx + offsPSW]
                ld  SP,  [Rx + offsSP]
                ld  R0,  [Rx + offsR0]
                 ⋮
                ld  R31, [Rx + offsR31]
                iret.
```

2. (10 poena)

Neki procesor obrađuje prekide (hardverske i softverske) tako što tokom izvršavanja prekidne rutine koristi poseban stek koji se koristi u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina).[1] Taj stek alociran je u delu memorije koju koristi kernel, a na vrh tog steka ukazuje poseban registar procesora koji je dostupan samo u privilegovanom režimu. Taj stek je uvek isti i kernel ga ne menja.

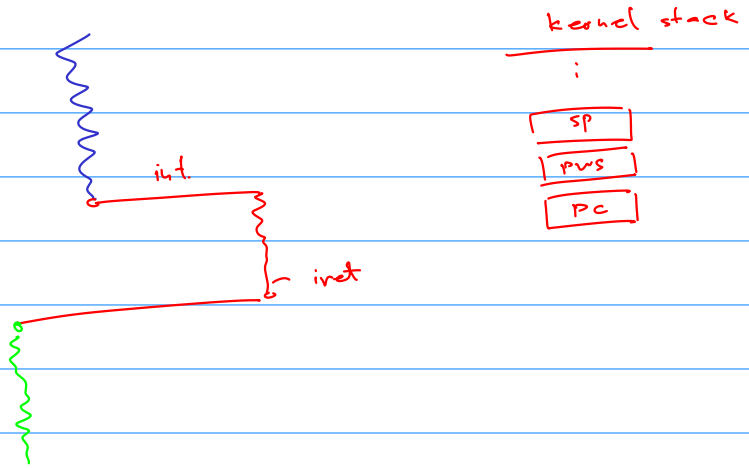
Prilikom obrade prekida, procesor na ovom sistemskom steku čuva: pokazivač vrha korisničkog steka (SP) koji je koristio prekinuti program, programsku statusnu reč (PSW) i programski brojač (PC), tim redom, ali *ne* i ostale programski dostupne registre. Prilikom povratka iz prekidne rutine instrukcijom `iret`, procesor restaurira ove registre sa sistemskog steka i vraća se u korisnički režim, a time i na korisnički stek.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće programski dostupne registre: 32 registra opšte namene (R0..R31), SP, PSW i PC. Poseban registar RX je dostupan samo u privilegovanom režimu rada procesora, pa ga korisnički procesi ne koriste i kernel ga može koristiti samo za svoje potrebe.

Registre PC, SP, PSW i R0..R31 treba sačuvati u odgovarajuća polja strukture PCB. U strukturi PCB postoje polja za čuvanje svih tih registara; pomeraji ovih polja u odnosu na početak strukture PCB označeni su simboličkim konstantama `offsPC`, `offsPSW`, `offsSP`, `offsR0`, ..., `offsR31`.

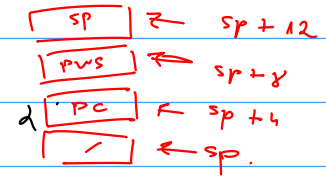
U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `scheduler`, koji nema argumente, realizuje raspoređivanje, tako što smešta PCB na koji ukazuje pokazivač `running` u listu spremnih procesa, a iz nje uzima jedan odabrani proces i postavlja pokazivač `running` na njega.

Na asembleru datog procesora napisati kod prekidne rutine `dispatch` koja vrši promenu konteksta korišćenjem potprograma `scheduler`. Ova prekidna rutina poziva se sistemskim pozivom iz korisničkog procesa, pri čemu se sistemski poziv realizuje softverskim prekidom.



R_0
R_1
⋮
R_{31}.

kernel stack



dispatch : ld Rx, running

st R_0, [Rx + offs R_0]

ld R_0, [sp + 4]

st R_0, [Rx + offs PC]

ld R_0, [sp + 8]

st R_0, [Rx + offs PSW]

ld R_0, [sp + 12]

st R_0, [Rx + offs SP]

st R_1, [Rx + offs R_1]

⋮

st R_{31}, [Rx + offs R_{31}]

call scheduler

ld Rx, running

pop R_0

pop R_0

pop R_0

ld R_0, [Rx + offs SP]

push R_0

ld R_0, [Rx + offs PSW]

push R_0

ld R_0, [Rx + offs PC]

push R_0.

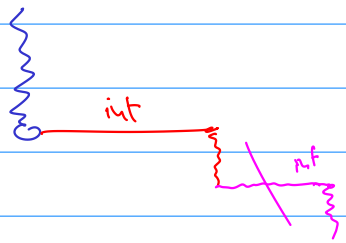
ld R_0, [Rx + offs R_0]

ld R_1, [Rx + offs R_1]

⋮

ld R_{31}, [Rx + offs R_{31}]

iret.



2. (10 poena)

Neki procesor obrađuje prekide (hardverske i softverske) tako što tokom izvršavanja prekidne rutine koristi poseban stek koji se koristi u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina).[1] Taj stek alociran je u delu memorije koju koristi kernel, a na vrh tog steka ukazuje poseban registar SSP procesora koji je dostupan samo u privilegovanom režimu.

Prilikom obrade prekida, procesor na ovom steku čuva programski brojač (PC) i programsku statusnu reč (PSW), tim redom, ali *ne* i ostale programski dostupne registre. Prilikom povratka iz prekidne rutine instrukcijom `iret`, procesor restaurira ove registre sa sistemskog steka i vraća se u korisnički režim, a time i na korisnički stek.

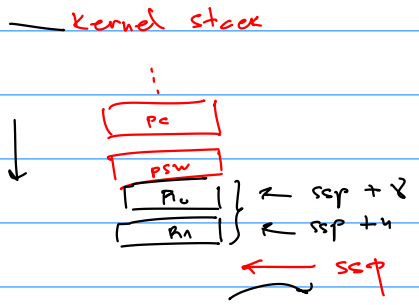
Postoji samo jedan kernel stek koji se koristi za izvršavanje celog koda kernela. Svi potprogrami kernela pisani su tako da na ovom steku čuvaju (i sa njega potom restauriraju) sve registre procesora koje koriste. Za sve vreme izvršavanja kernel koda prekidi su maskirani (procesor ih implicitno maskira prilikom prihvatanja prekida, a kernel kod ih ne demaskira).

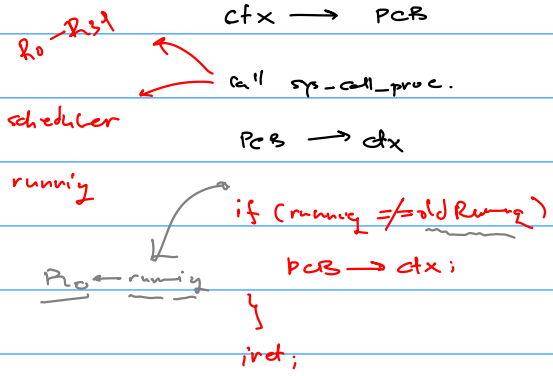
U strukturi PCB postoje polja za čuvanje vrednosti svih programski dostupnih registara procesora; pomeraj polja za neki registar R*i* u odnosu na početak strukture PCB označen je simboličkom konstantom `offsRi` (npr. `offsSP`, `offsPC`, `offsPSW`, `offsR0`, ..., `offsR31`).

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće programski dostupne registre: 32 registra opšte namene (R0..R31), SP, PSW i PC. Svi registri su 32-bitni.

U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `sys_call_proc`, koji nema argumente, realizuje obradu sistemskog poziva, kao i raspoređivanje, tako što, nakon obrade samog sisteskog poziva, postavlja pokazivač `running` da ukazuje na PCB odabranog novog tekućeg procesa.

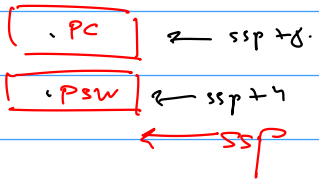
Na asembleru datog procesora napisati kod prekidne rutine `sys_call` za sistemske pozive softverskim prekidom, s tim da ona, zbog efikasnijeg rada, ne treba da vrši promenu konteksta (čuvanje i restauraciju registara) ako za tim nema potrebe, odnosno ukoliko procedura `sys_call_proc` nije promenila pokazivač `running`.

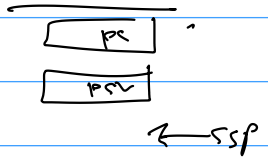





```
sys_call:   st   R0, [ssp]
            addi ssp, ssp, -4
            ld   R0, running
            call sys_call_proc
            ld   R1, [ssp]
            addi ssp, ssp, -4.
            ld   R1, running
            sub  R1, R0, R1
            jnz  R1, swap ctx
            ld   R0, [ssp + 8]
            ld   R0, [ssp + 4]
            addi ssp, ssp, 8
            iret.
```

```
swapctx:  sd   R2 , [R0 + offs R2]
          ld   R2 , [ssp + 4]          R2 := Rp
          sd   R2 , [R0 + offs R1]
          ld   R2 , [ssp + 8]
          ld   R1 , [R0 + offs R0]
          addi ssp , ssp , 8.
          ld   R2 , [ssp + 4]
          sd   R12 , [R0 + offs psw]
          ld   R2 , [ssp + 8]
          sd   R2 , [R0 + offs PC]
          addi ssp , ssp , 8.
          sd   sp , [R0 + off sp]
          sd   R3 , [R0 + offs R3]
           :
          sd   R31 , [R0 + offs R31].
          ld   R0 , [R1 + offs PC]
          sd   R0 , [ssp]
          addi ssp , ssp , -4.
          ld   R0 , [R1 + offs psw]
          sd   R0 , [ssp]
          addi ssp , ssp , -4.
          ld   sp , [R1 + offs sp]
          ld   R0 , [R1 + offs R0]
          ld   R2 , [R1 + offs R2.]
           :
          ld   R31 , [R1 + offs R31]
          ld   R1 , [R1 + offs R1]
          iret.
```





2. (10 poena)

Neki procesor obrađuje prekide (hardverske i softverske) tako što tokom izvršavanja prekidne rutine koristi poseban stek koji se koristi u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina).[1] Taj stek alociran je u delu memorije koju koristi kernel, a na vrh tog steka ukazuje poseban registar SSP procesora koji je dostupan samo u privilegovanom režimu.

Prilikom obrade prekida, procesor na ovom steku čuva: pokazivač vrha korisničkog steka (SP) koji je koristio prekinuti proces, programsku statusnu reč (PSW) i programski brojač (PC), tim redom, ali *ne* i ostale programski dostupne registre. Prilikom povratka iz prekidne rutine instrukcijom `iret`, procesor restaurira ove registre sa sistemskog steka i vraća se u korisnički režim, a time i na korisnički stek.

Svaki proces ima takav sopstveni sistemski stek. Prilikom promene konteksta, ostale programski dostupne registre treba sačuvati na ovom sistemskom steku prekinutog procesa. U strukturi PCB postoji polje za čuvanje vrednosti SSP steka procesa; pomeraj ovog polja u odnosu na početak strukture PCB označen je simboličkom konstantom `offsSSP`.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće programski dostupne registre: 32 registra opšte namene (R0..R31), SP, PSW i PC.

U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `scheduler`, koji nema argumente, realizuje raspoređivanje, tako što smešta PCB na koji ukazuje pokazivač `running` u listu spremnih procesa, a iz nje uzima jedan odabrani proces i postavlja pokazivač `running` na njega.

Na asembleru datog procesora napisati kod prekidne rutine `dispatch` koja vrši promenu konteksta korišćenjem potprograma `scheduler`. Ova prekidna rutina poziva se sistemskim pozivom iz korisničkog procesa, pri čemu se sistemski poziv realizuje softverskim prekidom.

iret

Rešenje:

```
dispatch:   addi  ssp, ssp, -128
            st    R0, [ssp + 1.4]
            st    R1, [ssp + 2.4]
                  :
            st    R31, [ssp + 32.4]
            ld    R0, running
            st    ssp, [R0 + offsSSP]
            call  scheduler
            ld    R0, running
            ld    ssp, [R0 + offsSSP]
            ld    R0, [ssp + 1.4]
            ld    R1, [ssp + 2.4]
                  :
            ld    R31, [ssp + 32.4]
            addi  ssp, ssp, 128
            iret
```