Sistemski ukazi

BRANKO ŠTER

ARHITEKTURA RAČUNALNIŠKIH SISTEMOV

- Sistemski ukazi spreminjajo parametre delovanja računalnika in nadzorujejo njegovo delovanje.
- Parametri:
  - omogočanje prekinitev in pasti,
  - način delovanja PP,
  - način delovanja navideznega pom.,
  - zaustavitev delovanja,
  - prehod na višji ali nižji nivo privilegiranosti,
  - itd.

Prekinitve in pasti

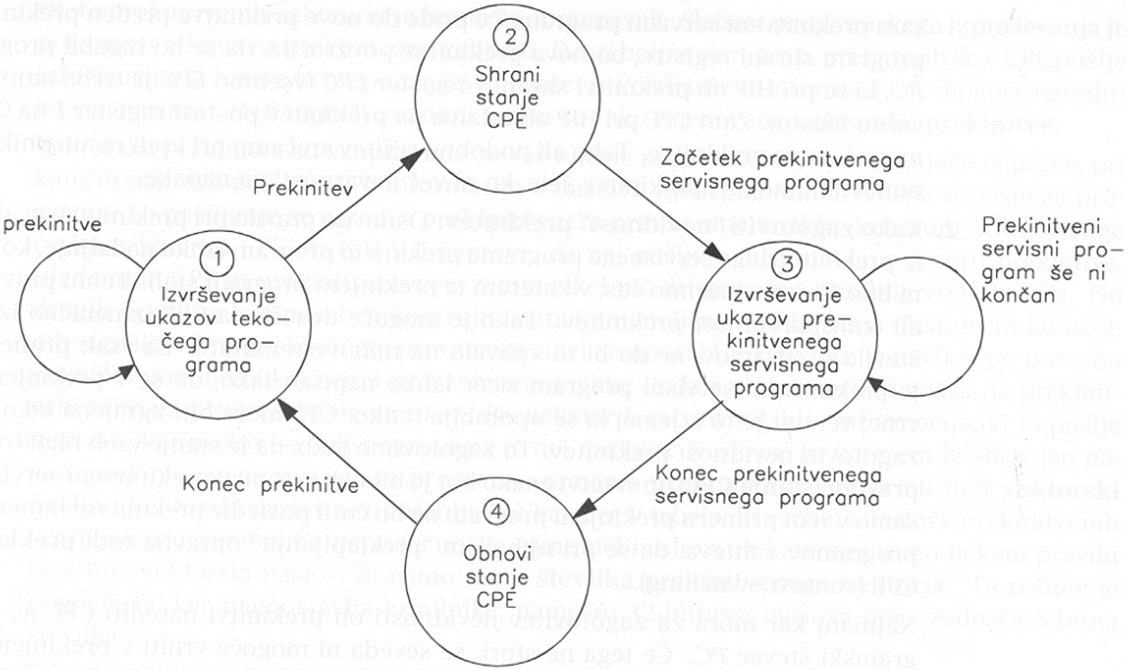
- **Prekinitev** (interrupt) je dogodek, ki povzroči, da CPE začasno preneha izvajati tekoči program ter prične izvajati t.i. **prekinitveni servisni program** (PSP)
  - Zahteva za prekinitev pride v CPE od zunaj, npr. od neke vhodno/izhodne naprave
- **Past** (trap) je posebna vrsta prekinitve, ki jo zahteva sama CPE ob nekem nenavadnem dogodku ali celo na zahtevo programerja
  - pasti pridejo od znotraj
- Če ne bi bilo prekinitev in pasti, bi morala CPE stalno preverjati stanje mnogih naprav

Ni enotne definicije

- Žal je terminologija prekinitev, pasti in izjem zelo zmedena:
  - RISC-V pasti deli na
    - izjeme (neveljeven/neporavnan ukaz, breakpoint, sistemski klici (ecall), napaka strani) in
    - prekinitve (hw oz. zunanje, časovniki, swi, preliv števca)
  - x86 prekinitve deli na
    - hw-prekinitve in
    - izjeme (napake oz. faults, pasti, aborts)
  - ARM izjeme deli na
    - prekinitve
    - sistemske klice
    - napake in
    - pasti

- Primeri uporabe:
  - zahteve V/I naprav ob različnih dogodkih
  - napaka v delovanju nekega dela računalnika
  - aritmetični preliv
  - napaka strani ali segmenta (pri navideznem pomnilniku)
  - dostop do zaščitene pomnilniške besede
  - dostop do neporavnane pomnilniške besede
  - uporaba nedefiniranega ukaza
  - klic programov operacijskega sistema

- Pri prekinitvah razlikujemo 4 stanja:
  - Normalno izvrševanje ukazov programa
  - Shranjevanje stanja CPE ob pojavu zahteve za prekinitev
  - Skok na prekinitveni servisni program in njegovo izvajanje
  - Vrnitev iz prekinitvenega servisnega programa in obnovitev stanja CPE



5 dejavnikov:

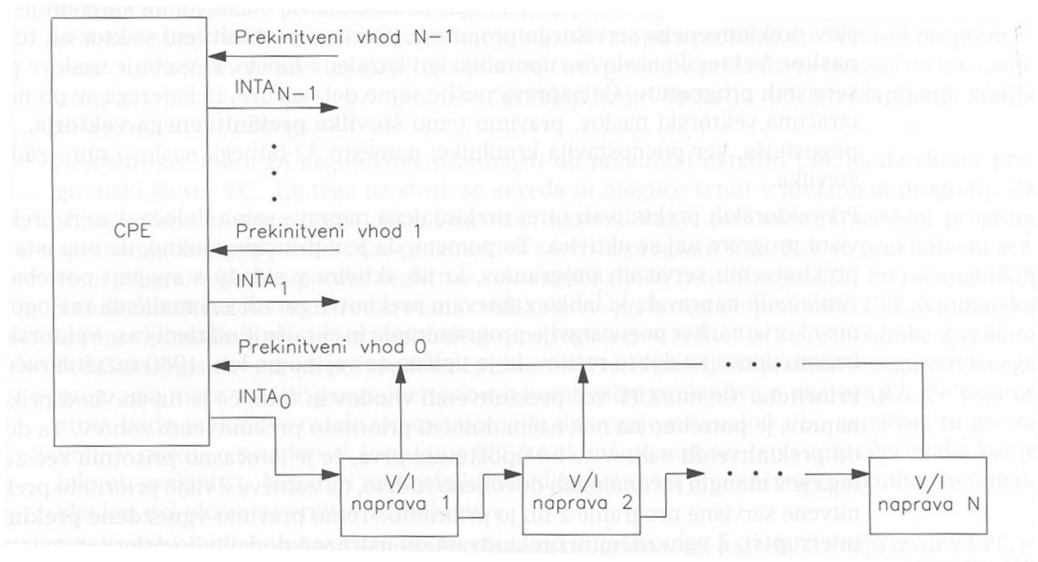
1. **Kdaj CPE reagira na prekinitveno zahtevo**
   - najenostavneje je po izvrševanju tekočega ukaza
     - v tem primeru se mora ohraniti samo stanje programsko dostopnih registrov (*x0-x31, PC, ...*)
   - programer lahko onemogoči odziv CPE na prekinitvene zahteve (bit *I*, ukaza DI in EI)
     - po vklopu so V/I prekinitve onemogočene, dokler se V/I naprave ne inicializirajo
     - če pride do nove prekinitve, preden prekinitveni servisni program shrani registre, lahko pride do izgube *PC*, ki se ob prekinitvi shrani v *EPC*
2. **Kako zagotoviti "nevidnost" prekinitev**
   - treba je zagotoviti, da je stanje (registrov) CPE enako kot prej

3. Kje se dobi naslov prekinitvenega servisnega programa

- to je pomembno pri prekinitvah, ki prihajajo od zunaj
- najprej je treba ugotoviti, katera naprava je zahtevala prekinitev
  - če je na vsakem prekinitvenem vhodu samo ena naprava, je problem trivialen
  - drugače je, če je na enem prek. vhodu več naprav, ali če ima ista naprava več PSP
- najpreprostejše je **programsko izpraševanje** (polling)
  - CPE bere registre vsake V/I naprave, v katerih je bit, ki pove, ali je naprava zahtevala prekinitev
  - če je, izvrši skok na njen prek. servisni program
  - polling je zamuden
- običajen način pa so **vektorske prekinitve**
  - naprava pošlje v CPE naslov njenega PSP v **prekinitvenem prevzemnem ciklu**, s katerim CPE obvesti V/I naprave, naj pošljejo informacijo o izvoru prekinitve
  - ta naslov se imenuje **prekinitveni vektor** ali **vektorski naslov**, ki se običajno izračuna iz **številke prekinitvenega vektorja** po nekem pravilu (tu zadošča npr. že 8-bitno število)
  - možno je tudi, da ima ena naprava več PSP

4. Prioriteta

- če ima CPE več prek. vhodov in več naprav na posameznem vhodu, potrebujemo neko prioriteto.
- **vgnezdene prekinitve** (nested interrupts), pri katerih zahteve z višjo prioriteto prekinejo prek. servisne programe z nižjo prioriteto
- **prekinitveni krmilnik** omogoča računalnikom, ki imajo CPE z enim samim bitom za omogočanje prekinitev, bolj fleksibilno obravnavo prioritete
- določanje prioritete je možno izvesti tudi z **marjetično verigo** (daisy chain)
  - naprava, ki ni zahtevala prekinitve, spusti določen signal v naslednjo napravo, tista pa, ki jo je, zapre signalu pot in vrne CPE ustrezno informacijo, da jo CPE lahko prepozna (na starejših ali na bolj preprostih sistemih)



5. Potrjevanje prekinitve

- potrebno zato, da naprava spusti prekinitveni vhod (da se prekinitev ne servisira večkrat)
- dva načina:
  - programsko: prekinitveni servisni program piše v nek register krmilnika naprave
  - strojno: z nekim signalom (ali kombinacijo večih) se obvesti napravo

Prekinitve in pasti pri cevovodu

- Kdaj skočiti na servisni program?
  - istočasno se izvaja več ukazov
  - delno izvršeni ukazi lahko povzročijo napake
- 3 primeri
  1. **Vhodno/izhodne prekinitve**
     - običajno je, da cevovod izvrši ukaze (ki so že v njem) do konca
       - V/I prekinitve so razmeroma redki dogodki, zato izguba ni velika
     - prekinitveno-prevzemni cikel je najbolje izvesti izven cevovoda (sicer bi rabili 6 stopenj v cevovodu)

2. Programske pasti

- v bistvu gre za klic procedure
  - poseben brezpogojni skok

3. Pasti, do katerih pride med izvrševanjem ukaza

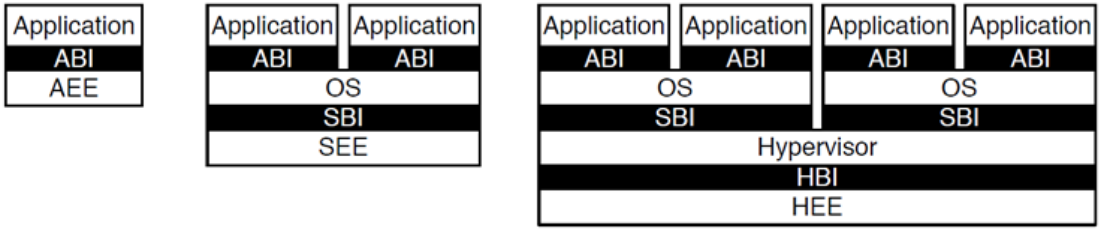
- najtežje
- zgodijo se na sredi ukaza
  - ukaz se ne more dokončati
  - potrebno ga je ustaviti, izvršiti servisni program in ga ponovno začeti
    - treba je tudi paziti, da del ukaza, ki se je (bil) že izvršil, ne povzroči napake

Stopnja cevovoda	Problematične pasti pri RISC procesorjih
IF	napaka strani (pri branju ukaza), zaščita pomnilnika
ID	nedefiniran ukaz (illegal instruction)
EX	preliv (overflow)
MEM	napaka strani (pri dostopu do operanda), zaščita pomnilnika neporavnan operand,
WB	nobena

- napaka strani (page fault): pri navideznem (virtualnem) pomnilniku, kadar stran ni (fizično) v GP (ne gre za resnično napako)
- zaščita pomnilnika: dostop do naslova, ki ne pripada programu (segmentation fault)
- pri napaki strani se po servisiranju program nadaljuje na prekinjenem mestu
- pri ostalih pasteh se običajno zaključi z diagnostičnim sporočilom

Arhitektura nivojev privilegiranosti v RISC-V

- Programi lahko tečejo na procesorjih RISC-V v različnih načinih:
  - application execution environment (AEE)
  - supervisor execution environment (SEE)
  - hypervisor execution environment (HEE)



4 nivoji privilegiranosti:

Nivo	Kodiranje	Ime	Kratica
0	00	User/App.	U
1	01	Supervisor	S
2	10	Reserved	
3	11	Machine	M

- Posamezna HW-nit (HW thread - hart) teče na nekem nivoju privilegiranosti
  - Hart normalno teče v načinu U, dokler ga neka past (sistemski klic ali prekinitev) ne prisili, da gre izvajati PSP (rokovalnik), ki običajno teče na višjem nivoju.
  - Privilegirani ukazi se lahko izvajajo le na dovolj visokem nivoju privilegiranosti.
- Uporabniški programi delujejo na najnižjem nivoju privilegiranosti.
  - Če računalnik prevzame privilegiran ukaz v v manj privilegiranem načinu, se sproži past.
  - Prehod na višji nivo je možen preko posebnih ukazov (supervisor calls, sistemski klici).

V načinu M (M-mode) je možno vse.
Način S (S-mode) se uporablja, če na procesorju teče OS.

Različne opcije:

- M: bare-metal, vgrajeni sistemi (preprosti), nobene zaščite
- M+U: vgrajeni sistemi z zaščito
- M+S+U: sistemi z OS

Debug mode (D-mode) je višji celo od načina M.

CSR

- 12-bitov (csr[11:0]) določa enega od 4096 CSR.
- Zgornji 4 biti naslova (csr[11:8]) določajo način dostopa (R/W) CSR glede na nivo privilegiranosti.
- Bita csr[11:10] določata, ali je register read/write (00, 01, ali 10) ali read-only (11).
- Bita csr[9:8] določata najnižji nivo privilegiranosti, ki lahko dostopi do CSR.

Primeri registrov CSR:

- m: mstatus (0x300), medeleg (0x302)
- s: (0x180) satp, stval (0x143)
- u: cycle (0xC00), time(0xC01), instret (0xC02), fflags (0x001)

Statusni register mstatus (najpomembnejši CSR)

- obravnava pasti, prekinitve omogočene ali ne,
- Konfiguracija: big/little endian, velikost registrov,

Past je sinhrona – sproži se zaradi nekega ukaza
Prekinitve so tipično asinhrone – zaradi zunanjega dogodka (V/I)

Primer preprostega PSP, ki se vrne kar na naslednji ukaz:

```
      .text
      j main
handler:
      # samo premakne uepc na naslednji ukaz
      csrrw t0, uepc, zero          # naloži EPC (naslov ukaza, ki je sprožil izjemo) v t0
      addi t0, t0, 4                # inkrementira t0 (naslednji ukaz)
      csrrw zero, uepc, t0          # posodobi EPC na t0
      uret                          # vrnitev na uepc
main:
      la t0, handler
      csrrw zero, utvec, t0         # utvec dobi naslov rokovalnika
      csrrsi zero,ustatus,1 #nastavi previous U int. enable bit (UPIE) v reg. ustatus (naslov 0)
      lw zero, 0(zero)              # sproži past za napako dostopa za load z naslova 0
      li a7, 10
      ecall
```

Sistemski ukazi RISC-V

CSR – Control and Status Register

- 12-bitni takojšnji operand določa enega od možnih 4096 registrov CSR

Format	Opkoda	funct3	Ukaz		Opis (ze ... zero extended)
I	1110011	001	csrrw	Atomic Read/Write CSR	rd ← ze(CSR), razen če rd==x0. CSR ← rs1
I	1110011	010	csrrs	Atomic Read and Set bits in CSR	rd ← ze(CSR). Biti CSR, ki imajo v rs1 (=maska) 1, se postavijo na 1, razen če rs1==x0
I	1110011	011	csrrc	Atomic Read and Clear bits in CSR	rd ← ze(CSR). Biti CSR, ki imajo v rs1 (=maska) 1, se brišejo (0), razen če rs1==x0
I	1110011	101	csrrwi	CSRRW immediate	rd ← ze(CSR), razen če rd==x0. CSR ← ze($uimm_{4:0}$) (v polju rs1)
I	1110011	110	csrrsi	CSRRS imm.	rd ← ze(CSR). Biti CSR, ki imajo v rs1 (=maska) 1, se postavijo na 1, razen če $uimm_{4:0}$==0
I	1110011	111	csrrci	CSRRC imm.	rd ← ze(CSR). Biti CSR, ki imajo v rs1 (=maska) 1, se brišejo (0), razen če $uimm_{4:0}$==0

- Sistemski ukazi shranjujejo podani register CSR v podani splošnonamenski register rd, v CSR pa naložijo novo vrednost (nove bite)
  - pogosto pa ne potrebujemo obeh ‘storitev’, ampak le eno

- Psevdoukazi za enostavnejše primere:

Psevdoukaz	Ukaz	Opis
CSRR rd, csr	CSRRS rd, csr, x0	samo branje CSR
CSRW csr, rs1	CSRRW x0, csr, rs1	samo pisanje CSR
CSRWI csr, uimm	CSRRWI x0, csr, uimm	samo pisanje CSR iz immed.
CSRS csr, rs1	CSRRS x0, csr, rs1	nastavljanje (set) bitov v CSR, kadar stare vrednosti ne rabimo
CSRC csr, rs1	CSRRC x0, csr, rs1	brisanje (clear) bitov v CSR, kadar stare vrednosti ne rabimo
CSRSI csr, uimm	CSRRSI x0, csr, imm	nastavljanje (set) bitov v CSR iz imm., kadar stare vrednosti ne rabimo
CSRCI csr, uimm	CSRRCI x0, csr, imm	brisanje (clear) bitov v CSR iz imm., kadar stare vrednosti ne rabimo

Če pride do pasti (ecall) v načinu U:

- rokovalnik (handler) teče v načinu S

Registri načina S:

- sstatus – statusni register
- stvec – trap vector (naslov rokovalnika)
- sepc – exception PC
- scause – koda vzroka
- stval – dodatne info
- sscratch – delovni reg.

sstatus vsebuje bite:

- sie – Supervisor, Interrupts Enabled
- spie – Supervisor, Previous Interrupts Enabled (prejšnji sie)
  - ob pasti v način S spie dobi vrednost sie, sie pa 0 (na koncu ob sret sie dobi spie, spie pa 1)
- spp – Supervisor, Previous Priority (0: U, 1: S)

Obdelava pasti

- HW-faza:
  - past
  - onemogoči prekinitve (sstatus: SIE) oz. (mstatus: MIE)
  - shranjen prejšnji status (sstatus: SPP, SPIE) oz. (mstatus: MPP, MPIE)
  - scause (8: U, 9: S) oz. mcause
  - sepc (naslov ukaza) oz. mepc
  - PC <- stvec oz. mtvec
- SW-faza:
  - shranjevanje registrov (v sscratch oz. mscratch)
  - obdelava kode scause oz- mcause
  - sepc++
  - ohnovitev registrov
  - ukaz sret (SIE = SPIE, mode = SPP, PC = sepc) oz. mret

Kode izjem

Koda	Pomen	Koda	Pomen
0	Neporavnan naslov ukaza	8	ecall iz načina U
1	Napaka dostopa ukaza	9	ecall iz načina S
2	Neveljaven ukaz	11	ecall iz načina M
3	Breakpoint	12	Napaka strani pri ukazu
4	Neporavnan naslov pri load	13	Napaka strani pri load
5	Napaka pri operandnem dostopu	15	Napaka strani pri store
6	Neporavnan naslov pri store	18	SW
7	Napaka dostopa pri store	19	HW-napaka

- Rokovalnik pasti lahko teče v načinu S (ali M)
  - rokovalnik dostopa do CSR registrov načina S
  - rokovalnik konča z ukazom `sret`
- Prekinitev/izjema se zgodi v načinu U ali S
- HW-faza
  - spremeni način v S
  - shrani prejšnje stanje
  - skok na rokovalnik pasti
- Podobno lahko rokovalnik teče v načinu M
- Način M lahko ‘delegira’ nekatere pasti v način S (če le-ta obstaja)

Izvori prekinitev

- Zunanje prekinitve (EI – external int.)
  - SEI (S-mode EI)
  - MEI (M-mode EI)
- Prekinitve časovnika (TI – timer int.)
  - STI (S-mode TI)
  - MTI (M-mode EI)
- SW prekinitve (SI – SW int.)
  - SSI (S-mode SI)
  - MSI (M-mode SI)
- Prekinitev preliva lokalnega števca (LCOFI – local counter overflow int.)

Prekinitveni krmilnik PLIC

- PLIC – Platform level interrupt controller
  - Prekinitveni krmilnik prejme prekinitev
  - Odloči se, katero jedro bo obvestil (če je več jeder)
  - Eno jedro ‘prizna’ prekinitev (kot pri osebni napaki pri košarki)
    - to jedro zažene rokovalnik
    - jedro obvesti PLIC, da ‘upokoji’ prekinitev
  - Druga jedra ignorirajo to prekinitev
  - CSR v jedru ima bita ‘pending’ in ‘enabled’ za vsak tip prekinitve
    - Če prekinitev trenutno ni omogočena, postane pending

Preostali sistemski ukazi

- FENCE
  - pomnilniška pregrada zagotavlja, da se pred pomnilniškim dostopom dokončajo vsi morebitni prejšnji dostopi
  - to je pomembno predvsem v kontekstu večnitenja in spremenjenega vrstnega reda izvajanja ukazov (out-of-order)
- FENCE.I
  - pregrada za ukaze zagotavlja, da se pred branjem ukaza izvedejo vsa morebitna prejšnja pisanja
- ECALL (environment call)
  - implementacija sistemskih klicev
  - sistemski klici omogočajo uporabniku, do dobi usluge od jedra OS (privilegiran način delovanja), tipično dostop do HW (pomnilnik, disk, terminal, ...)
- EBREAK
  - med izvajanjem programa vrne kontrolo razhroščevalniku

Kaj mi bo zbirni jezik?

Za prevedbo iz višjenivojskega ali ‘srednjenivojskega’ jezika (C) v zbirni jezik poskrbi prevajalnik

- npr.: gcc, clang, lcc, IAR, Visual C, Watcom, ...
- danes so prevajalniki že zelo dobri

Kljub temu pa je včasih potrebno napisati kako zbirniško kodo – v takem primeru ni potrebno prevajati konstruktov višjega jezika v zbirni jezik

- Torej, ni treba začeti z višjenivojsko kodo in jo prevajati, temveč lahko neposredno pišemo v zbirnem jeziku, saj lahko kaj naredimo tudi bolj učinkovito

Primeri uporabe programiranja v zbirnem jeziku

- **Zagonski programi** - nizkonivojska koda v bralnem oz. bliskovnem pomnilniku za inicializacijo in testiranje strojne opreme pred zagonom operacijskega sistema, npr. BIOS
- **Deli jedra OS**, sistemski klici za določeno arhitekturo
- Nekateri jeziki in prevajalniki omogočajo vključevanje delov zbirniške kode (**inline** assembly), npr. za specifično CPE
- **Disassembly** – koda v zbirnem jeziku, ki jo je ustvaril prevajalnik ob prevajanju iz višjega jezika – lahko se uporabi za razhroščevanje in/ali optimizacijo
- V zgodnjih računalnikih je bilo možno v zbirnem jeziku napisati **bolj učinkovito** kodo.
- **Vzvratno inženirstvo** (Reverse engineering) - strojne kode ni težko prevesti (disassembler) v zbirni jezik. Na ta način je *v principu* možno rekonstruirati izvorno kodo.

- Zanimiva uporaba zbirnika je tudi preverjanje, ali je indeks polja znotraj obsega (bounds check)
  - Če želimo preveriti na čimkrajši način, ali je neka spremenljivka x v obsegu $0 \leq x < y$, lahko predznačeno število obravnavamo kot nepredznačeno.
  - Negativna števila v 2'K izgledajo kot velika števila v nepredznačenem formatu!
  - Tako nam nepredznačena primerjava x < y preverja tako tudi, če je x < 0
  - Npr.: če x20 >= x11 ali x20 < 0, potem skoči na oznako IndexOutOfBounds :

```
bgeu x20, x11, IndexOutOfBounds
```

- Nenazadnje, in morda najpomembneje, zbirni jezik uporabljamo kot uporabniku prijazen zapis strojne kode pri **študiju delovanja računalnikov!**