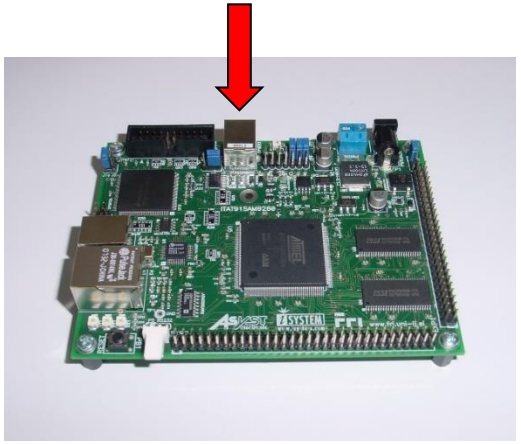
# *ARM*

*Vhodno / izhodne naprave*

*Prekinitve*

# *Delo na FRI-SMS razvojnem sistemu*



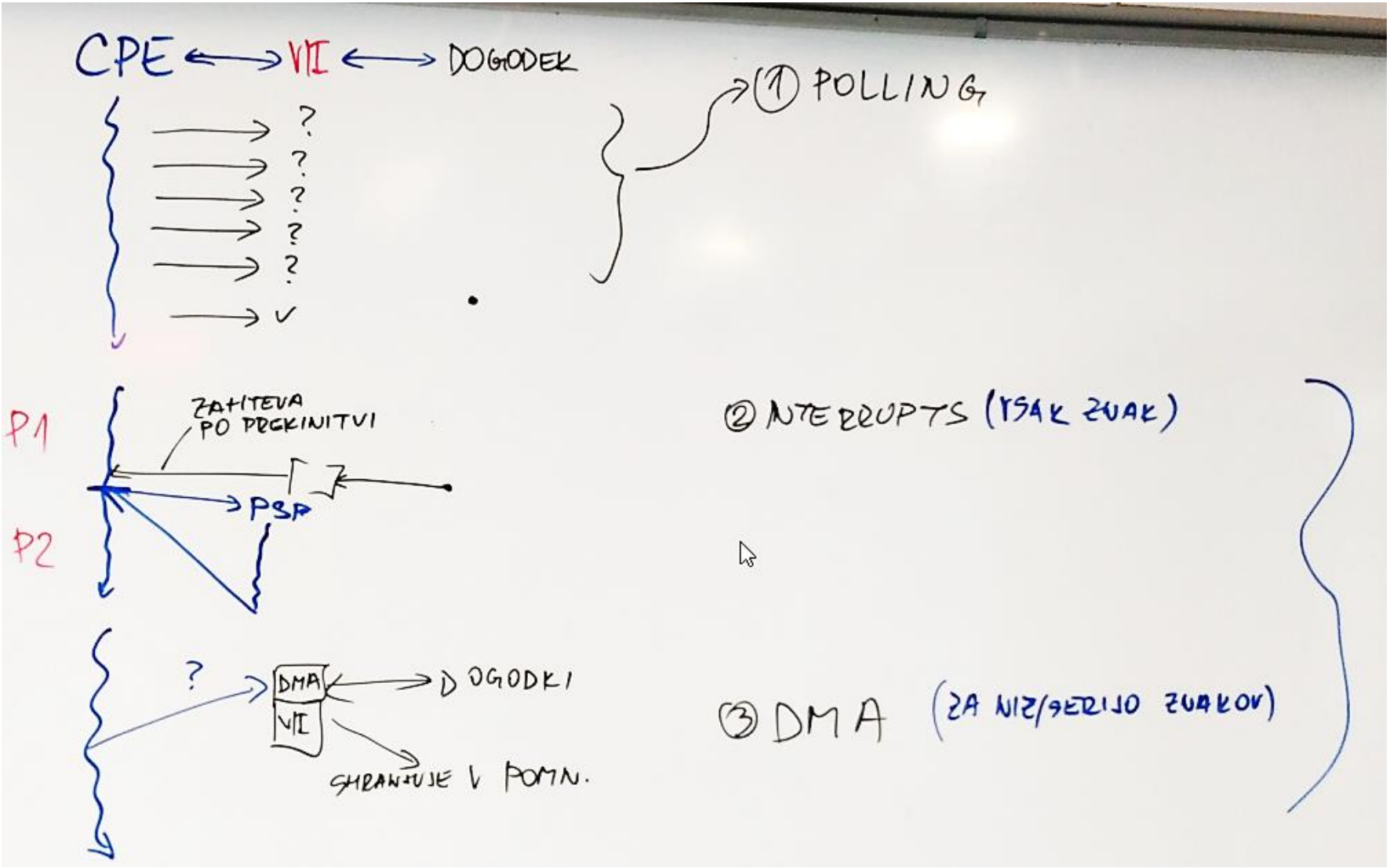
Priključitev :

- **USB** priklop na **daljši stranici**, sveti **zelena LED** dioda

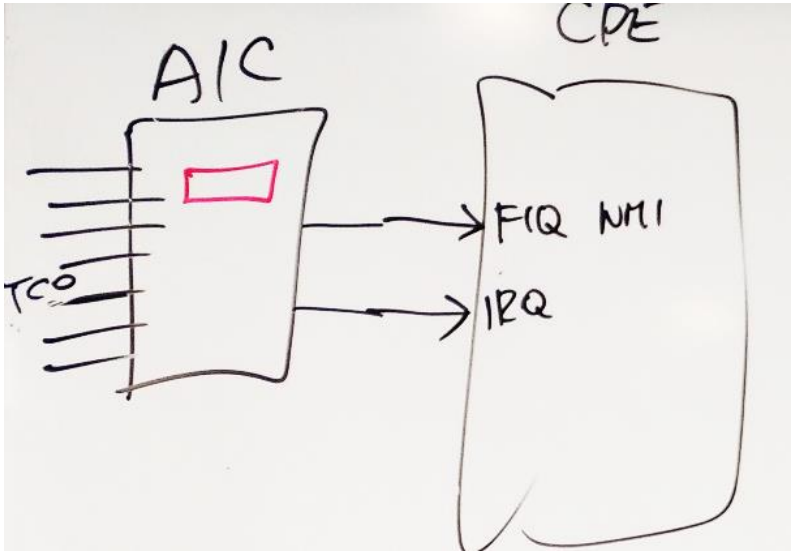
Poseben projekt za FRI-SMS (e-učilnica) :

- ***dodatne nastavitve*** (informativno) :
  - frekvenca urinega signala (višja poveča porabo!)
  - vklop predpomnilnikov
  - inicializacija sklada oz. SP – kazalca na sklad
- ***dodajanje vsebine (start.s):***
  - *podatki/operandi:*
    - dodamo v `/*constants*/` ,končamo z .align
  - *program* :
    - dodamo v `/* enter your code here */`
    - na koncu programa je mrtva zanka
    - podprograme dodamo za mrtvo zanko

# Prekinitve – Zakaj ?



# Prekinitve




**Prekinitev:**

- dogodek, ki povzroči, da procesor prekine izvajanje trenutno izvajajočega se programa
- vsaka prekinitev ima običajno svoj prekinitveno-servisni program (PSP)
- procesor ARM ima vektorske prekinitve: vsaki prekinitvi pripada svoj vektor
- prekinitveni vektor pri ARM: naslov pomnilniške besede s prvim ukazom PSP
- ob prekinitvi procesor spremeni način delovanja

**Prekinitve in izjeme pri ARM:**

- FIQ – Fast Interrupt Request - zunanja prekinitev, nastopi ob aktiviranju FIQ# priključka
- IRQ – Interrupt Request - zunanja prekinitev, nastopi ob aktiviranju IRQ# priključka
- SWI – programska prekinitev
- Reset
- *Prefetch abort – izjema, ki nastopi ob neuspešnem branju ukaza (npr. zgrešitev v predpomnilniku, napaka strani ipd.)*
- *Data abort – izjema, ki nastopi ob neuspešnem dostopu do operanda (npr. zgrešitev v predpomnilniku, napaka strani ipd.)*
- *Undefined instruction – izjema, ki nastopi ob branju nedefiniranega ukaza (sproži tudi ko izvajamo priviligirane ukaze v uporabniškem načinu)*
- *Reserved*

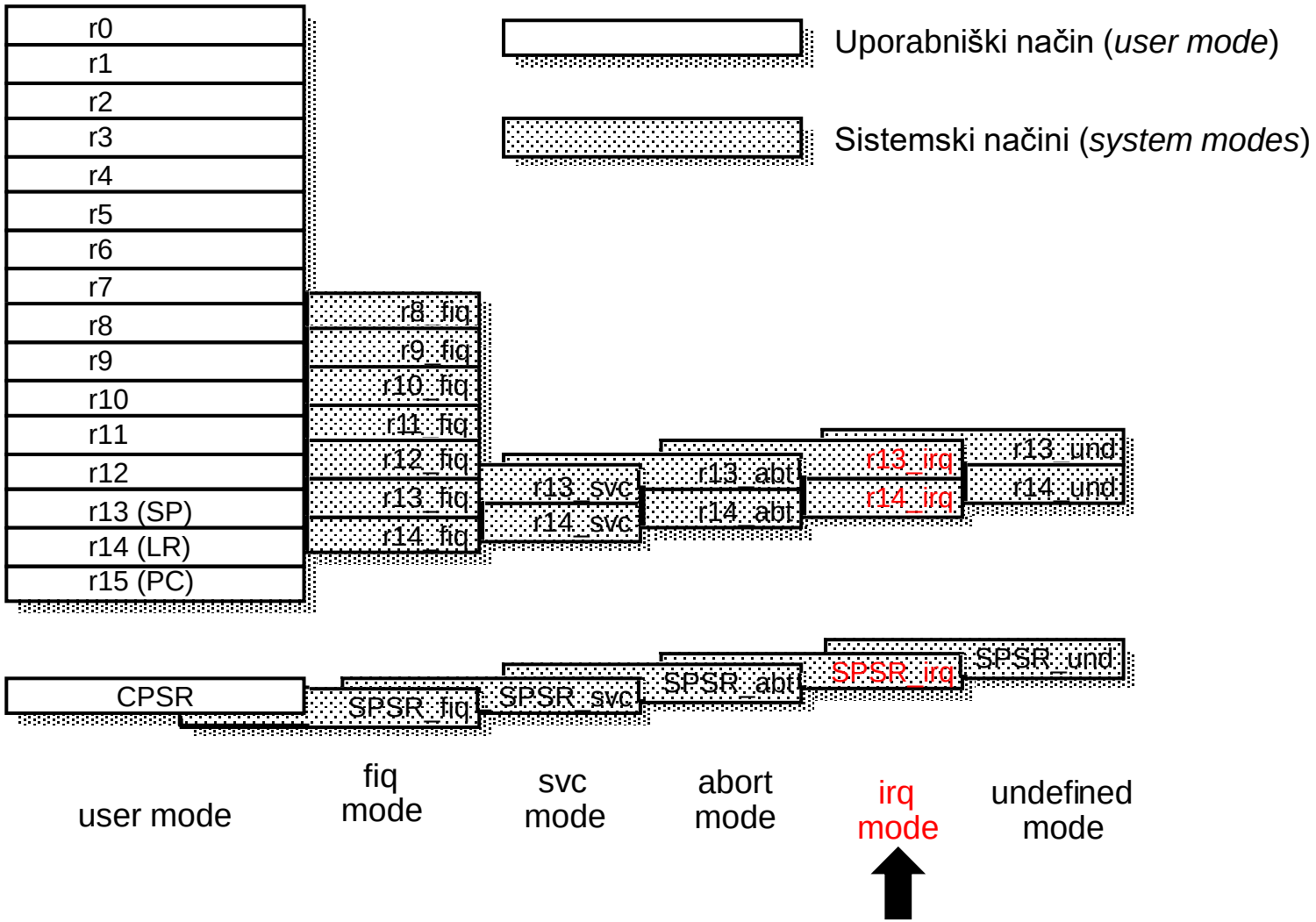
# Prekinitve in izjeme

| Prekinitev / Izjema | Način delovanja | Vektor |
|---|---|---|
| Reset | svc | 0x00000000 |
| Undefined Inst. | und | 0x00000004 |
| SWI | svc | 0x00000008 |
| Prefetch Abort | abt | 0x0000000C |
| Data Abort | abt | 0x00000010 |
| Reserved | | 0x00000014 |
| IRQ | irq | 0x00000018 |
| FIQ | fiq | 0x0000001C |

**WinIdea: vektorji so določeni v intvec.s :**

| Processor Mode | Description |
|---|---|
| User (*usr*) | Normal program execution mode |
| FIQ (*fiq*) | Fast data processing mode |
| IRQ (*irq*) | For general purpose interrupts |
| Supervisor (*svc*) | A protected mode for the operating system |
| Abort (*abt*) | When data or instruction fetch is aborted |
| Undefined (*und*) | For undefined instructions |
| System (*sys*) | Privileged mode for OS Tasks |

# Prekinitve in izjeme – načini delovanja CPE



# Prekinitve

Ob prekinitvi se zgodi naslednje:

- CPSR se prepiše v SPSR zahtevanega načina delovanja
- PC se shrani v LR zahtevanega načina delovanja
- v CPSR se zapiše zahtevani način delovanja
- v PC se prepiše prekinitveni vektor

Ob prekinitvi procesor spremeni način delovanja!

Zato ob resetu ne pozabimo nastaviti skladovnega kazalca za IRQ način delovanja (za IRQ prekinitve) !

Pri prepisu prekinitvenega vektorja v PC se:

- prevzame ukaz iz vektorske tabele in
- s tem se začne izvajanje določenega PSP v zahtevanem načinu delovanja.

# Prekinitve in izjeme – vektorska tabela

Vsak prekinitveni vektor vsebuje lahko le en ukaz. To je ukaz s katerim skočimo na PSP za pripadajočo prekinitev oz. izjemo. Ta ukaz je lahko:

- `B naslov` – s tem ukazom skačemo relativno glede na PC. PSP se mora začeti blizu vektorske tabele.

- `ldr pc, [pc, #odmik]` – s tem ukazom v PC naložimo naslov PSP iz pomnilnika. Naslov je tokrat 32-biten in lahko skočimo na poljuben naslov v pomnilniku, vendar se PSP začne izvajati pozneje zaradi dodatnega dostopa do pomnilnika.

- `ldr pc, [pc, #-0x0F20]` – tak ukaz običajno uporabimo pri rabi prekinitvenega krmilnika. V nadaljevanju bomo spoznali zakaj.

# Prekinitve – vektorska tabela

Vsak prekinitveni vektor vsebuje lahko le en ukaz, s katerim skočimo na PSP za pripadajočo prekinitev. Ta ukaz je pri uporabi AIC krmilnika običajno:

- `ldr pc, [pc, #-0x0F20]`
  - tak ukaz običajno rabimo pri uporabi prekinitvenega krmilnika.

•AIC je pomnilniško preslikan na naslov 0xFFFFF000

• register AIC_IVR je na odmiku 0x100, oz. na naslovu 0xFFFFF100

• ob IRQ prekinitvi se v PC zapiše 0x00000018. Na tem naslovu (v vektorski tabeli) moramo zapisati ukaz, s katerim v PC prepišemo AIC_IVR :

```
0x00000018:    ldr pc, [pc, #-0x0F20]
```

saj je 0x00000018 + 8 – 0x0F20 = 0xFFFFF100.

• tako z enim samim ukazom prenesemo vsebin iz AIC_IVR v PC in s tem skočimo na začetek PSP in s tem zmanjšamo latenco pri odzivu na prekinitve!

Kako AIC ve, ali se je CPE odzvala na kakšno prekinitev in jo servisira?

CPE mora ob odzivu na prekinitev prebrati prekinitveni vektor oz. naslov PSP iz registra AIC_IVR – branje iz tega registra je za AIC znak, da je CPE začela s servisiranjem prekinitve.

# Prekinitve in izjeme

| Prekinitev / Izjema | Prioriteta *1 .. najvišja 6 .. najnižja* | Postavi I bit v CPSR | Postavi F bit v CPSR |
|---|---|---|---|
| Reset | 1 | da | da |
| Undefined Inst. | 6 | da | ne |
| SWI | 6 | da | ne |
| Prefetch Abort | 5 | da | ne |
| Data Abort | 2 | da | ne |
| IRQ | 4 | da | ne |
| FIQ | 3 | da | da |

The **Interrupt flags** are as follows:

- **I**: when set, disables IRQ interrupts
- **F**: when set, disables FIQ interrupts

# Prekinitve in izjeme – Link Register

Pri odzivu CPE na prekinitveno zahtevo CPE:

- najprej dokonča izvajanje trenutno izvajajočega se ukaza in
- nato PC prepiše v LR.

Tako v LR hranimo "povratni" naslov. Vendar se zaradi cevovodnega izvajanja v LR praviloma ne zapiše naslov ukaza pri katerem je bil program prekinjen (v tabeli je to pc), ampak:

| Prekinitev / Izjema | Naslov, ki se zapiše v LR | Dejanski povratni naslov |
|---|---|---|
| Reset | -- | -- |
| Undefined Inst. | pc+4 | lr |
| SWI | pc+4 | lr |
| Prefetch Abort | pc+8 | lr-4 |
| Data Abort | pc+12 | lr-8 |
| IRQ | pc+8 | lr-4 |
| FIQ | pc+8 | lr-4 |

# Prekinitev IRQ

Pri odzivu na IRQ prekinitev se v CPE zgodi naslednje:

- CPSR se prepiše v SPSR_irq
- PC+8 se shrani v LR_irq:
  - v PC je naslov zadnjega izvedenega ukaza v prekinjenem programu (pravi povratni naslov je torej PC+4 oz. LR_irq - 4)
- v CPSR se zapiše irq način delovanja
- v PC se naloži naslov 0x00000018. Na tem naslovu je skočni ukaz na PSP.

Zasnova PSP - Prekinitveno-servisnega programa:

```
IRQ_hand:
        sub r14,r14,#4     /* popraviti moramo povratni naslov */
        stmfd r13!, {r14}  /* na sklad shrani povratni naslov.
                              Po potrebi še druge registre! */

        ...                /* servisiraj zahtevo ... */

        ldmfd r13!, {pc}^  /* vrni se na prekinjeni program,
                              pred tem obnovi CPSR iz SPSR_irq*/
```

Znak ^ povzroči obnovitev CPSR iz SPSR_irq.
Uporabi se lahko samo v privilegiranem načinu!

# Prekinitev IRQ

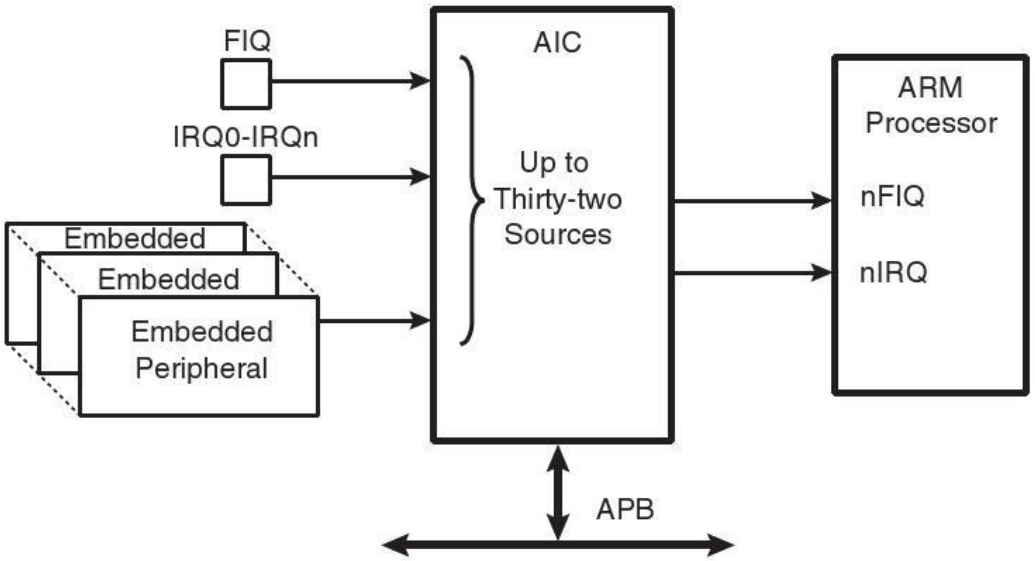
Takoj po resetu ne smemo pozabiti nastaviti skladovnega kazalca za irq način delovanja:

```
_start:
/* izberi irq način */
  mrs r0, cpsr
  bic r0, r0, #0x1F   /* pobriši zastavice načina delovanja */
  orr r0, r0, #0x12   /* nastavi irq način */
  msr cpsr, r0

/* nastavi irq kazalec na sklad */
  ldr sp,_Lirqstack_end

/* izberi uporabniški način način */
  mrs r0, cpsr
  bic r0, r0, #0x1F   /* pobriši zastavice načina delovanja */
  orr r0, r0, #0x10   /* nastavi uporabniški način */
  msr cpsr, r0

/* nastavi uporabniški kazalec na sklad */
  ldr sp,_Lstack_end

/* glavni program */
```

In na koncu še rezervirati ustrezni prostor za oba sklada :

```
/* constants */
  .align
_Lstack_end:
  .long __STACK_END__- 2*13*4  @ space for 26 registers on IRQ stack
_Lirqstack_end:
  .long __STACK_END__
```

# Prekinitveni krmilnik (Advanced Interrupt Controller)




**Osnovne lastnosti AIC:**

- Omogoča priključitev 32 prekinitvenih izvorov na procesor ARM
- Prekinitvene izvore priključi na prekinitvena vhoda FIQ# in IRQ# na procesorju ARM
- AIC sprejema prekinitvene zahteve iz teh 32 virov ter ustrezno aktivira signala FIQ# ali IRQ#
- AIC omogoča prioritetno razvrščanje prekinitvenih izvorov
- vsakemu izvoru lahko programsko določimo enega izmed 8 prioritetnih nivojev
- vsi prekinitveni vhodi so lahko proženi na nivo ali fronto – to določimo programsko
- na prekinitvene vhode je lahko povezana poljubna vgrajena V/I ali neka zunanja naprava

# Prekinitveni krmilnik – prekinitveni vhodi

**Priključitev prekinitvenih izvorov/vhodov:**

- **Prekinitveni izvori: IS0 – IS31 (Interrupt Sources)**
- **IS0 – vedno priključen na FIQ vhod vezja AIC**
- **IS1 – to je t.i. sistemska prekinitev – na ta izvor so preko OR vrat priključene naslednje sistemske naprave:**

| | |
|---|---|
| **System Timer** | **(sistemski časovnik)** |
| **RTC** | **(ura realnega časa)** |
| **Power Menagement Controller** | **(krmilnik za upravljanje porabe)** |
| **Memory Controller** | **(pomnilniški krmilnik)** |
| **Debug Unit** | **(enota za razhroščevanje)** |

- **IS2-IS31 – na te izvore so priključene vgrajene V/I naprave; mesto priključka je določeno z ID vgrajene naprave (npr. TC0 z ID 17 je priključen na IS17)**

**Privzeta nastavitev je, da vsi prekinitveni izvori, razen IS0, povzročijo IRQ prekinitev v CPE. Izvor IS0 povzroči FIQ prekinitev. Programsko sicer lahko določimo FIQ prekinitev tudi za izvore IS1-IS31.**

# Prekinitveni krmilnik – funkcionalni opis

**AIC_SMR[0..31]** : Source Mode Register – vsak IS ima pripadajoči AIC_SMR

PRIOR (biti 0-2) : določimo prioriteto prekinitvenega izvora

0 – najnižja, 7 - najvišja

SRCTYPE (bita 5-6) : določimo občutljivost IS

00 – prekinitev se proži ob visokem stanju na IS

01 – prekinitev se proži ob pozitivni fronti na IS

**AIC_SVR[0..31]** : Interrupt Source Vector Register

v te registre za vsak IS vpišemo prekinitveni vektor oz. naslov PSP, ki ga sami določimo

**AIC_IECR** : Interrupt Enable Comand Register

vpis 1 v bit omogoča prekinitve za ISi

**AIC_EOICR** : End of Interrupt Comand Register

vpis v register pomeni konec obravnave prekinitve

**AIC_IVR** : Interrupt Vector Register

v tem registru lahko preberemo prekinitveni vektor za tisti IS, ki je sprožil trenutno prekinitev. V tem registru dobi CPE dejanski prekinitveni vektor ali naslov PSP za IS, ki je prekinitev prožil.

# *Prekinitveni krmilnik – delovanje*

Predpostavimo:

- so registri ustrezno nastavljeni (prekinitev omogočena, določena prioriteta)
- PSP za ta izvor se nahaja na naslovu, ki je vpisan v AIC_SVR17

Ob zahtevi za prekinitev iz izvora IS17 se zgodi naslednje:

1. Prekinitveni krmilnik mora najprej ugotoviti ali se že servisira kakšna prekinitev.
   - Kako AIC ve, ali se je CPE odzvala na kakšno prekinitev in jo servisira?
     - CPE mora ob odzivu na prekinitev prebrati prekinitveni vektor oz. naslov PSP iz registra AIC_IVR – branje iz tega registra je za AIC znak, da je CPE začela s servisiranjem prekinitve.
   - Če CPE že servisira neko prekinitev:
     - in je prioriteta servisirane prekinitve nižja od IS17:
       - potem AIC aktivira prekinitveni vhod na CPE – s tem proži novo prekinitev in CPE prekine servisiranje trenutne (ob predpostavki, da v CPE prekinitve niso maskirane)
     - in je prioriteta servisirane prekinitve višja od prioritete IS17:
       - potem mora AIC počakati da CPE zaključi s servisiranjem prekinitve preden ponovno aktivira prekinitvena vhoda na CPE!
   - Kako AIC ve, ali je CPE zaključila s servisiranjem neke prekinitve?
     - CPE mora ob zaključku servisiranja prekinitve zapisati poljubno vrednost v register AIC_EOICR !
       - Gre za t.i. slepo pisanje. Pisalni dostop do AIC_EOICR pove AIC, da je CPE zaključila s servisiranjem, in jo lahko spet prekine.

# Prekinitveni krmilnik – vektorske prekinitve

AIC omogoča vektorske prekinitve. Ob vsaki prekinitvi tako CPE lahko ugotovi, kateri izvor je prožil prekinitev. To deluje na naslednji način:

- Za vsak prekinitveni izvor moramo v pripadajoči register AIC_SVR zapisati ustrezen vektor ali naslov PSP.

- Ko prekinitveni izvor i zahteva prekinitev, prepiše AIC vsebino iz registra AIC_SVRi v register AIC_IVR, tj:

```
AIC_IVR  <- AIC_SVRi
```

- CPE bo prejela prekinitveni signal preko vhodov IRQ# ali FIQ#. 'Odzvala' se bo torej na znanih vektorjih 0x00000018 ali 0x0000001C.

- CPE mora najprej prebrati vsebino AIC_IVR v PC, da bo lahko pričela z izvajanjem ustreznega PSP.

# Prekinitveni krmilnik – vektorske prekinitve

## Kako preberemo AIC_IVR v PC?

- AIC je pomnilniško preslikan na naslov 0xFFFFF000

- register AIC_IVR je na odmiku 0x100, oz. na naslovu 0xFFFFF100

- ob IRQ prekinitvi se v PC zapiše 0x00000018. Na tem naslovu (v vektorski tabeli) moramo zapisati ukaz, s katerim v PC prepišemo AIC_IVR :

```
0x00000018:    ldr pc, [pc, #-0x0F20]
```

saj je 0x00000018 + 8 – 0x0F20 = 0xFFFFF100.

- tako z enim samim ukazom skočimo na PSP in s tem zmanjšamo latenco pri odzivu na prekinitve!

# Prekinitveni krmilnik – nastavitve

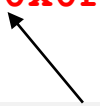
1) nastavimo prioriteto 4,
2) določimo prekinitveni vektor,
3) omogočimo prekinitve za IS17.

```
.equ AIC_BASE,    0xFFFFF000        /* Začetni (bazni) naslov AIC */
.equ AIC_SMR17,   0x044             /* odmiki posameznih registrov v AIC */
.equ AIC_SVR17,   0x0C4
.equ AIC_IECR,    0x120
.equ AIC_EOICR,   0x130


AIC_IS17_Init:
        ldr r0, =AIC_BASE
        ldr r2, #4
        str r2, [r0, #AIC_SMR17]  /* prioriteta=4 */
        ldr r2, =IRQ17_Hand
        str r2, [r0, #AIC_SVR17]  /* naslov PSP za IS17 */
        ldr r1, #1 << 17
        str r1, [r0, #AIC_IECR]   /* omogočimo prekinitve za IS17*/
        str r0, [r0, #AIC_EOICR]  /* slepo pisanje v EOICR */
```

# Prekinitveni krmilnik – prekitveni vektorji

Vektorska tabela:

```
B       _start          /* RESET INTERRUPT */
B       _undef          /* UNDEFINED INSTRUCTION INTERRUPT */
B       _swi            /* SOFTWARE INTERRUPT */
B       _abort_pref     /* ABORT (PREFETCH) INTERRUPT */
B       _abort_data     /* ABORT (DATA) INTERRUPT */
B       _reserved       /* RESERVED */
ldr pc, [pc, #-0x0F20]  /* IRQ INTERRUPT */
ldr pc, [pc, #-0x0F20]  /* FIQ INTERRUPT */
```



•Kako AIC ve, ali se je CPE odzvala na kakšno prekinitev in jo servisira?
CPE mora ob odzivu na prekinitev prebrati prekinitveni vektor oz. naslov PSP iz registra AIC_IVR – branje iz tega registra je za AIC znak, da je CPE začela s servisiranjem prekinitve.

# Prekinitveni krmilnik – zgled

PSP za IS17:

```
IRQ17_Hand :
        sub r14,r14,#4 /* ustrezno popravi povratni naslov */
        stmfd r13!, {r14} /* na sklad shrani povratni naslov. */
        /* po potrebi shrani še preostale registre na sklad … */
        /* po potrebi omogoči prekinitve … */
        …
        /* … PSP … */
        …
        /* maskiraj prekinitve, da se lahko varno vrneš … */

        /* slepo piši v AIC_EOICR: */
        ldr r0, =AIC_BASE
        str r0, [r0, #AIC_EOICR]

        /* obnovi vse registre s sklada … */

        /* vrni se iz PSP in obnovi CPSR: */
        ldmfd r13!, {pc}^
```

- Kako AIC ve, ali je CPE zaključila s servisiranjem neke prekinitve?

CPE mora ob zaključku servisiranja prekinitve zapisati poljubno vrednost v register AIC_EOICR ! Gre za t.i. slepo pisanje.

Pisalni dostop do AIC_EOICR pove AIC, da je CPE zaključila s servisiranjem, in jo lahko spet prekine.

# Prekinitve – povzetek sprememb v kodi 1

start.s:

```
/* izberi irq način */
  mrs r0, cpsr
  bic r0, r0, #0x1F   /* pobriši zastavice načina delovanja */
  orr r0, r0, #0x12   /* nastavi irq način */
  msr cpsr, r0

/* nastavi irq kazalec na sklad */
  ldr sp,_Lirqstack_end
```

Takoj po resetu ne smemo pozabiti nastaviti skladovnega kazalca za irq način delovanja:

```
  .equ AIC_BASE,          0xFFFFF000      /* Začetni (bazni) naslov AIC */
  .equ AIC_SMR17,         0x044           /* odmiki posameznih registrov v AIC*/
  .equ AIC_SVR17,         0x0C4
  .equ AIC_IECR,          0x120
  .equ AIC_EOICR,         0x130
```

Dodamo definicije registrov za prekinitveni krmilnik - AIC

```
/* user code here */
      bl INIT_IO
      bl INIT_TC0_PSP
      bl Init_AIC
      bl Enable_IRQ
```

V glavni program dodamo inicializacije :

- Časovnika (ki vključuje vklop prekinitvene zahteve, ko bo števec preštel)
- AIC prekinitvenega krmilnika
- Omogočanje IRQ prekinitev

# Prekinitve – povzetek sprememb v kodi 2a

start.s:

```
LED_Switch:
  stmfd r13!, {r0-r1, r14}

  ldr r0, LED_Counter
  eors r0, r0, #1
  str r0, LED_Counter
  mov r1, #2
  ldr r0, =PIOC_BASE
  strne r1, [r0, #PIO_SODR]
  streq r1, [r0, #PIO_CODR]
  ldmfd r13!, {r0-r1, pc}
```

Preveri trenutno stanje in ga negiraj

Glede na rezultat negacije (eors) vklopi ali izklopi LED diodo.

# Prekinitve – povzetek sprememb v kodi 2

start.s:

```
Enable_IRQ :
  stmfd r13!, {r0, r14}
  mrs r0, cpsr
  bic r0, r0, #0x80   /* clear I mask */
  msr cpsr, r0
  ldmfd r13!, {r0, pc}

Init_AIC :
  stmfd r13!, {r0-r2, r14}
  ldr r0, =AIC_BASE
  mov r1, #1 << 17
  mov r2, #4          /* high level sensitive, PRIOR=4 */
  str r2, [r0, #AIC_SMR17]
  ldr r2, =IRQ17_Hand
  str r2, [r0, #AIC_SVR17]    /* set interrupt vector for IS17 */
  str r1, [r0, #AIC_IECR]    /* enable interrupt for IS17 */
  str r0, [r0, #AIC_EOICR]
  ldmfd r13!, {r0-r2, pc}
```

Omogočanje IRQ prekinitev:
- zbrišemo bit I v statusnem registru

Vklop prekinitve za napravo 17 (TC – časovnik)

Nastavitev prioritete in naslova PSPja v prekinitveni vektor

Vklop prekinitve za napravo 17,
Slepo pisanje v **AIC_EOICR**

# Prekinitve – povzetek sprememb v kodi 3

start.s:

```
INIT_TC0_PSP:
  stmfd r13!, {r0, r2, lr}
  ldr r2, =PMC_BASE     /*Enable PMC for TC0 */
  mov r0, #(1 << 17)
  str r0, [r2,#PMC_PCER]

  /*Initialize TC0 MCK/128, RC=375 (1ms) */
  ldr r2, =TC0_BASE
  mov r0, #0b110 << 13 /*WAVE=1, WAVSEL= 10*/
  add r0, r0, #0b011              /* MCK/128 */
  str r0, [r2, #TC_CMR]
  ldr r0, =375                    /* 1 ms at 48 Mhz */
  str r0, [r2, #TC_RC]
  mov r0, #0b0101      /*TC_CLKEN,TC_SWTRG*/
  str r0, [r2, #TC_CCR]
  mov r0, #0x10                   /* CPSR triggers interrupt */
  str r0, [r2, #TC_IER]
  ldmfd r13!, {r0, r2, pc}
```

Aktivacija :

- Ko števec prešteje do željene vrednosti se bo sprožila tudi zahteva po prekinitvi

.

# Prekinitve – povzetek sprememb v kodi 4

start.s:

```
IRQ17_Hand:
  sub r14,r14,#4
  stmfd r13!, {r0-r1, r14} /* na sklad shrani povratni naslov.

  /* servisiraj zahtevo ... */
  ldr r1, =TC0_BASE
  ldr r0, [r1, #TC_SR]
  tst r0, #0b10000
  beq IRQ17_Exit

  ldr r0, IRQ17_Counter
  subs r0, r0, #1
  str r0, IRQ17_Counter
  bne  IRQ17_Exit

  moveq r0, #500          /* Counter reached 0 - start from 500 again */
  str r0, IRQ17_Counter
  mov r0,#1               /* it's 500 msecs: Set MSEC_500 to 1 */
  str r0,MSEC_500

IRQ17_Exit:

  ldr r0, =AIC_BASE
  str r0, [r0, #AIC_EOICR] /* slepo pisi v AIC_EOICR: */

  ldmfd r13!, {r0-r1, pc}^ /* vrni se na prekinjeni program,
                              pred tem obnovi CPSR iz SPSR_irq*/
```

Zmanjšaj vrednost števca milisekund.

Če že 0, potem MSEC_500 = 1

Pisalni dostop do AIC_EOICR pove AIC, da je CPE zaključila s servisiranjem, in jo lahko spet prekine.

Ob vstopu v PSP se CPSR shrani v SPSR_irq
^ ob izhodu povrne CPSR iz SPSR_irq

# Prekinitve – povzetek sprememb v kodi 5

start.s:

```
/* constants */

  .align
_Lstack_end:
  .long __STACK_END__- 2*13*4  @ space for 26 registers on IRQ stack
_Lirqstack_end:
  .long __STACK_END__

LED_Counter:
  .word 1

IRQ17_Counter:
  .word 500

MSEC_500:
  .word 0

.end
```

Zmanjšaj običajni sklad in dodaj še prekinitveni sklad.

Shranjuje trenutno stanje LED diode.

Števec, ki se zmanjšuje vsako prekinitev na 1ms.

Spremenljivka, ki se nastavi na 1 vsakih 500 ms oz. pol sekunde.

# Prekinitve – povzetek sprememb v kodi 6

start.s:

```
_main:

/* user code here */
      bl INIT_IO
      bl INIT_TC0_PSP
      bl Init_AIC
      bl Enable_IRQ

LOOP: ldr r0, IRQ17_Counter
      ldr r1, LED_Counter
      ldr r2, MSEC_500
      cmp r2,#0
      beq LOOP
      bl LED_Switch
      mov r2, #0
      str r2, MSEC_500

      b LOOP

/* end user code */
```

Potrebne inicializacije za delovanje prekinitev.

Nepotrebno, samo za debug …

Preteklo pol sekunde (MSEC_500 == 1)?

NE: čakaj naprej
DA: - preklopi LED diodo
- zbriši indikator: MSEC_500 = 0

večna zanka

# Prekinitve – povzetek sprememb v kodi 6

intvec.s:

```
.text
.code 32

.global _start
  B       _start                  /* RESET INTERRUPT */
  B       _error                  /* UNDEFINED INSTRUCTION INTERRUPT */
  B       _error                  /* SOFTWARE INTERRUPT */
  B       _error                  /* ABORT (PREFETCH) INTERRUPT */
  B       _error                  /* ABORT (DATA) INTERRUPT */
  B       _error                  /* RESERVED */
  ldr pc, [pc, #-0x0F20]   /* IRQ INTERRUPT */
  ldr pc, [pc, #-0x0F20]   /* FIQ INTERRUPT */

.end
```

•AIC je pomnilniško preslikan na naslov 0xFFFFF000

• register AIC_IVR je na odmiku 0x100, oz. na naslovu 0xFFFFF100

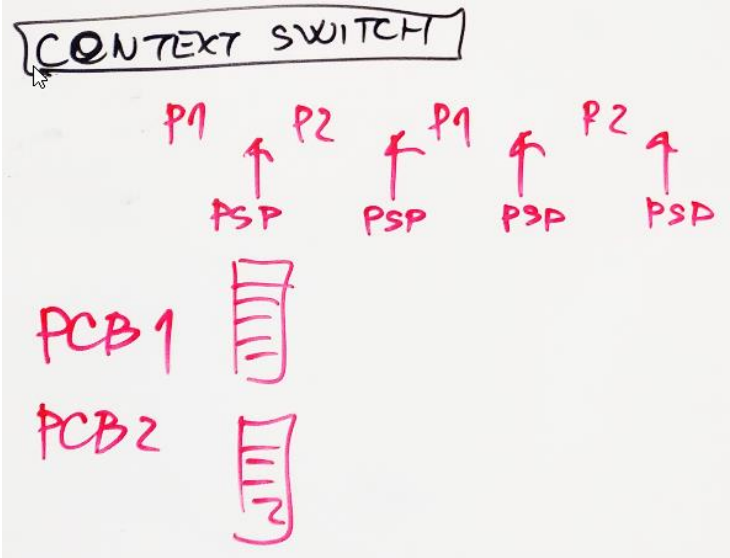
• ob IRQ prekinitvi se v PC zapiše 0x00000018. Na tem naslovu (v vektorski tabeli) moramo zapisati ukaz, s katerim v PC prepišemo AIC_IVR :

```
0x00000018:    ldr pc, [pc, #-0x0F20]
```

saj je 0x00000018 + 8 – 0x0F20 = 0xFFFFF100.

• tako z enim samim ukazom prenesemo vsebin iz AIC_IVR v PC in s tem skočimo na začetek PSP in s tem zmanjšamo latenco pri odzivu na prekinitve!

# Prekinitve – Izvedba več procesov




**Context switch**
The following example performs a context switch on the User mode process. The code is based around a list of pointers to ***Process Control Blocks*** **(PCBs)** of processes that are ready to run.
This figure shows the layout of the PCBs that the example expects.

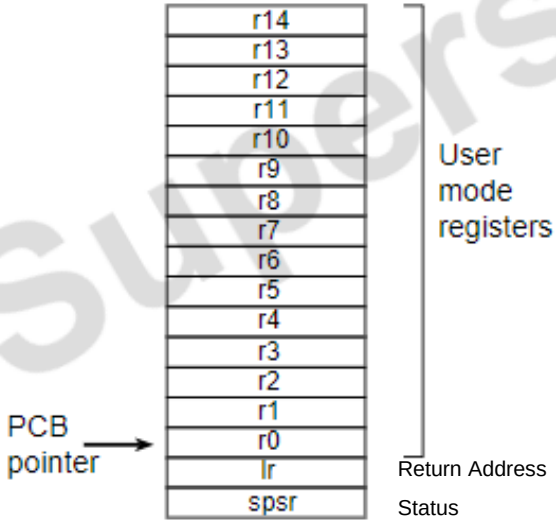
The pointer to the PCB of the next process to run is pointed to by R12, and the end of the list has a zero pointer. Register R13 is a pointer to the PCB, and is preserved between time slices, so that on entry it points to the PCB of the currently running process.

**Example 42. Context switch on the User mode process**

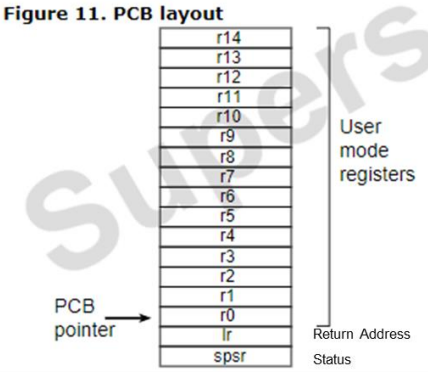
```
STM     sp,{R0-lr}^     ; Dump user registers above R13.
MRS R0, SPSR            ; Pick up the user status
STMDB sp, {R0, lr}      ; and dump with return address below.

LDR sp, [R12], #4       ; Load next process info pointer.
CMP sp, #0              ; If it is zero, it is invalid
LDMDBNE sp, {R0, lr}    ; Pick up status and return address.
MSRNE SPSR_cxsf, R0     ; Restore the status.
LDMNE   sp, {R0 - lr}^   ; Get the rest of the registers NOP
SUBSNE pc, lr, #4       ; and return and restore CPSR.
Insert "no next process code" here.
```

**VIR:** ARM Compiler Software Development Guide

**Figure 11. PCB layout**



# Prekinitve – Izvedba več procesov




start.s:

***Dodana koda :***

***Initialization: …***

```
        adr r12,PCB0+8      @ first process is PCB0
```

R12 = PCB pointer

***Variables: …***

PCB0, PCB1

```
ACT_PROC: .word 0           @ current process ID
PCB0:     .word 0, 0,               0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0  @ SPSR,lr, r0-11
PCB1:     .word 0b10000, PROC_1,    0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0  @ SPSR,lr, r0-11
CNT_PROC: .word 0           @ counter for process 1
```

SPSR ..SR procesa
lr .. PC procesa

***Process ID=1: …***

```
PROC_1:     ldr r0, CNT_PROC
            add r0, r0, #1
            str r0, CNT_PROC
            b PROC_1
```

Dodatni proces ID=1:
- Povečevanje CNT_PROC

(vzel bo pribl. polovico CPE časa)

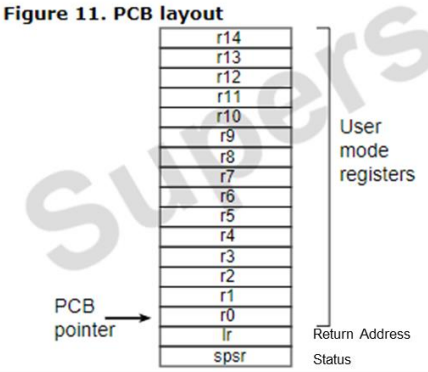
`0b10000` .. Zač. SR procesa (UserMode)

`PROC_1` .. Zač. naslov procesa

Osnovni proces ID=0,
enak kot prej:
- Utripanje LED diode

(vzel bo pribl. polovico CPE časa)

# Prekinitve – Izvedba več procesov




start.s:

```
PSP: …
IRQ17_Hand:
 sub lr,lr,#4
  stm     r12,{r0-r11}^        @ ^ -> Dump user mode registers above R13
  mrs     r0, SPSR             @ Pick up the user status register  (saved into SPSR)
  stmdb   r12, {r0, lr}        @ and dump with return address below sp.

          /* servisiraj zahtevo ... */

IRQ17_Exit:

  ldr  r12,ACT_PROC            @ change ID to next process
  eors r12, r12, #1
  str r12,ACT_PROC

  adreq r12,PCB0+8             @ New proc 0: set r12 to PCB0
  adrne r12,PCB1+8             @ New proc 1: set r12 to PCB1

  ldmdb r12, {r0, lr}          @ Pick up status and return address.
  msr   SPSR_cxsf, r0          @ Restore the status.
  ldm   r12, {r0 - r11}^       @ Get the rest of the user registers,
                                    ^ brez pc pomeni user registre

  stmfd r13!, {lr}             /* na sklad shrani povratni naslov.    */
  ldmfd r13!, {pc}^            /* vrni se na naslednji proces - prekinjeni program,
                                   ^ s pc -> pred tem obnovi CPSR iz SPSR_irq*/
                                      pred tem obnovi CPSR iz SPSR_irq*/
```

Shrani registre prekinjenega procesa

PCB0, PCB1

ID: 0 ali 1

R12 -> novi PCB

Povrni registre novega procesa

Vrnitev

# Prekinitve – dodatne informacije

**Razlaga ^ v zbirniku :**

**4.2.4. LDM and STM**

**Load and store multiple registers. Any combination of registers r0 to r15 can be transferred.**

…

**^**

**is an optional suffix. You must not use it in User mode or System mode. It has two purposes:**

- **If op is LDM and reglist contains the pc (r15), in addition to the normal multiple register transfer, the SPSR is copied into the CPSR. This is for returning from exception handlers. Use this only from exception modes.**
- **Otherwise, data is transferred into or out of the User mode registers instead of the current mode registers.**

# Prekinitve – dodatne informacije

**Odziv CPE na izjemo :**

**The processor response to an exception**

•This describes the processor response to an exception. You must ensure that the exception handler saves the system state when an exception occurs and restores it on return.

•Processors that support Thumb state use the same basic exception handling mechanism as processors that do not support Thumb state. An exception causes the next instruction to be fetched from the appropriate vector table entry.

•When an exception is generated, the processor performs the following actions:

- Copies the CPSR into the appropriate SPSR. This saves the current mode, interrupt mask, and condition flags.
- Switches state automatically if the current state does not match the instruction set used in the exception vector table.
- Changes the appropriate CPSR mode bits to:
- Change to the appropriate mode, and map in the appropriate banked out registers for that mode.
- Disable interrupts. IRQs are disabled when any exception occurs. FIQs are disabled when an FIQ occurs and on reset.
- Sets the appropriate LR to the return address.
- Sets the PC to the vector address for the exception.

Vir: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0471c/Cacdihic.html

# Prekinitve – dodatne informacije

**Vrnitev iz PSP:**

**Return from an exception handler**
This describes the processor response to an exception, and how to return to the main program after the exception has been handled. You must ensure that the exception handler saves the system state when an exception occurs and restores it on return.

The method used to return from an exception depends on whether the exception handler uses stack operations or not. In both cases, to return execution to the place where the exception occurred an exception handler must:
•restore the CPSR from the appropriate SPSR
•restore the PC using the return address from the appropriate LR.

For a simple return that does not require the destination mode registers to be restored from the stack, the exception handler carries out these operations by performing a data processing instruction with:
•the S flag set
•the PC as the destination register.
The return instruction required depends on the type of exception.

**Note**
You do not have to return from the reset handler because the reset handler executes your main code directly.
If the exception handler entry code uses the stack to store registers that must be preserved while it handles the exception, it can return using a load multiple instruction with the ^ qualifier. For example, an exception handler can return in one instruction using:

```
LDMFD sp!,{R0-R12,pc}^
```

To do this, the exception handler must save the following onto the stack:
•all the work registers in use when the handler is invoked
•the link register, modified to produce the same effect as the data processing instructions.

The ^ qualifier specifies that the CPSR is restored from the SPSR. It must be used only from a privileged mode.

Vir: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0471c/Caccfegc.html

# Prekinitve – dodatne informacije

•Enabling and disabling Interrupt

•This is done by modifying the ***CPSR***, this is done using only 3 ARM instruction:

- MRS To read *CPSR*
- MSR To store in *CPSR*
- BIC Bit clear instruction
- ORR OR instruction
- MRS r1, cpsr
- BIC r1, r1, #0x80/0x40
- MSR cpsr_c, r1
- MRS r1, cpsr
- ORR r1, r1, #0x80/0x40
- MSR cpsr

```
.end
```

## Enabling and disabling Interrupt

This is done by modifying the ***CPSR***, this is done using only 3 ARM instruction:

MRS To read *CPSR*
MSR To store in *CPSR*
BIC Bit clear instruction
ORR OR instruction

Enabling an IRQ/FIQ Interrupt:

```
MRS   r1, cpsr
BIC   r1, r1, #0x80/0x40
MSR   cpsr_c, r1
```

Disabling an IRQ/FIQ Interrupt:

```
MRS   r1, cpsr
ORR   r1, r1, #0x80/0x40
MSR   cpsr_c, r1
```