

Avbrott

Avbrottsstyrd I/O

- I/O-enheten genererar ett avbrott som får processorns exekvering att hoppa direkt till en avbrottsrutin.

- Ett avbrott (interrupt/exception/trap) är
 - en yttre signal (external interrupt)

Alternativt

- en inre händelse (internal exception/trap)

som får processorn att avbryta pågående exekvering och hoppa till en avbrottsrutin

Olika typer av avbrott i ARM

- De olika typer av avbrott som kan ske i en ARM är:

Typ av händelse	Varifrån?	Prioritet (1=H, 6 = L)
Reset	Externt	1
IRQ	Externt	4
SWI (Software Interrupt, instr.)	Internt	6
Illegal adress (Abort)	Internt	2/5 (Data/instr.)
Odefinierad instruktion	Internt	6
FIQ	Externt	3

- IRQ och FIQ är de avbrott som kan tillåtas eller förbjudas, kallas *maskbara* avbrott
- De övriga kan inte stängas av

Avbrottsvektor(tabell) i ARM

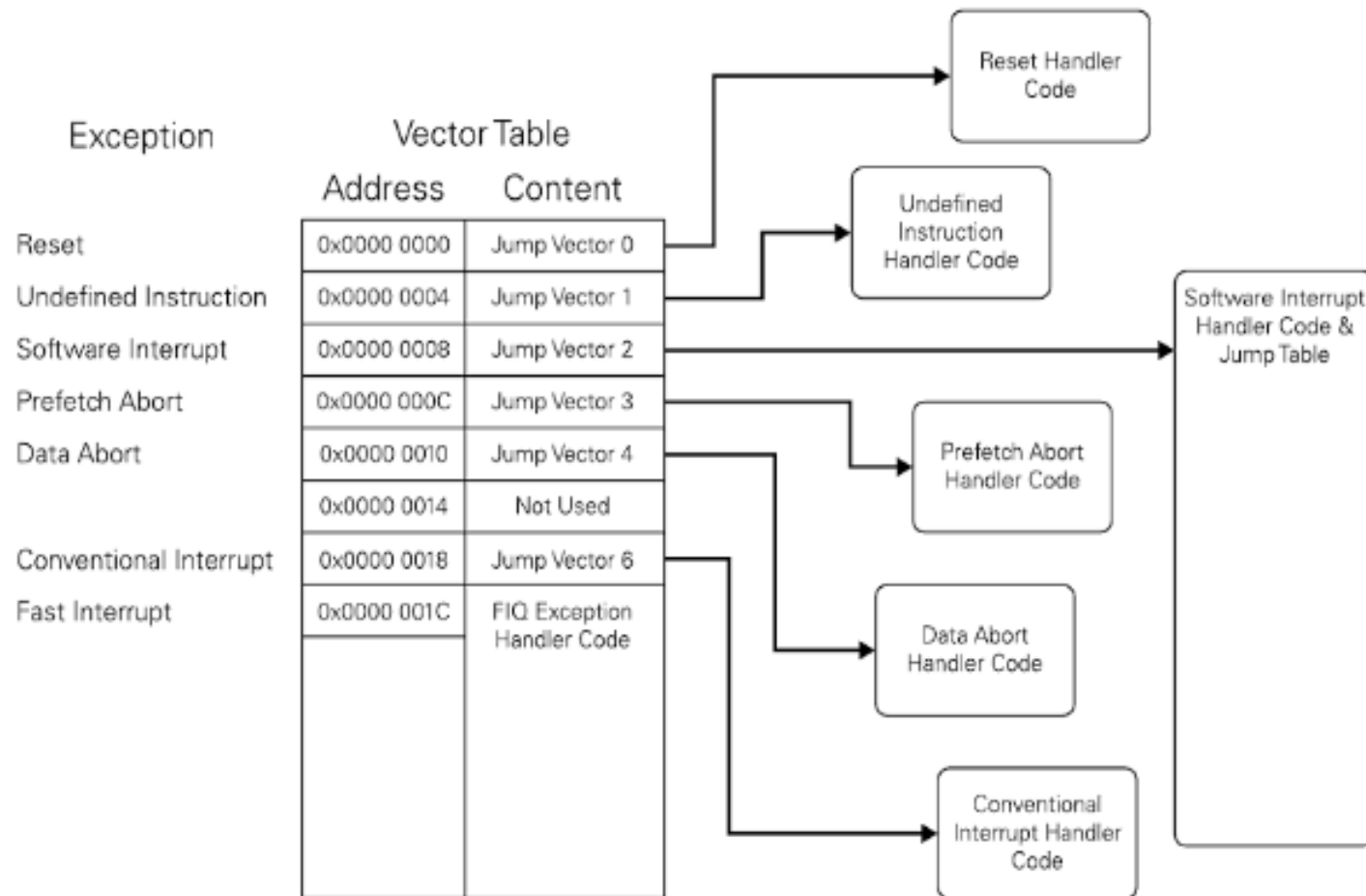


FIGURE 4-19: The ARM exception **vector table**

Jump vector = Hoppadress

Vektortabellens adress förutbestämd av hårdvara

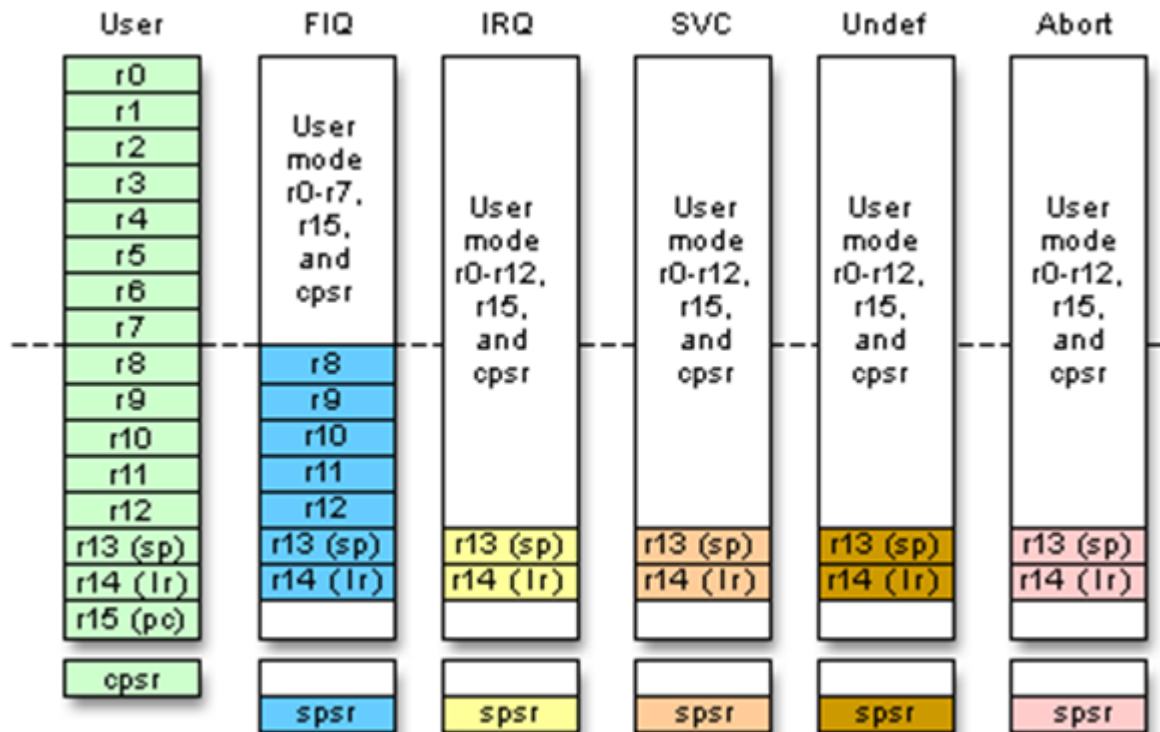
Vid avbrottshantering måste man veta

- När är det tillåtet att bli avbruten och av vad?
- Vad har orsakat avbrottet?
- Vad höll processorn på med när den blev avbruten?
- Vilken prioritet har olika avbrott relativt varandra?

Processorn har speciella register för att hantera avbrott

Hårdvarustöd för avbrott





















ARM har sex olika "tillstånd"



Användardefinierade avbrott

- Hur många och vilka externa avbrott som kan genereras beror på vilken hårdvara som kopplas till processorn
- Raspberry PI har en GPIO-enhet kopplad till ARM
- Vi har dessutom kopplat in en I/O-board till denna (GertBoard)
- Ni kommer att använda några av de generella I/O-pinnarna ur GPIO (GPIO_GEN)

Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Statusregister CPSR

31	30	29	28	27	26 - 25	24	23 - 20	19 - 16	15 - 10	9	8	7	6	5	4 - 0
N	Z	C	V	Q	RAZ	J	Reserved	GE	RAZ	E	A	I	F	T	M[4:0]

- I-biten kontrollerar om IRQ är tillåtet
- F-biten kontrollerar om FIQ är tillåtet
- Dessa bitas kallas processorns *avbrottsmask*
- Mode anger i vilken mode processorn arbetar (user, eller någon av de privilegierade)
- Sätts mha linux-kärnan i labben

Vad händer vid ett avbrott?

(Om avbrottet är tillåtet, annars händer inget ...)

1. Programmet avbryts mitt i eller efter den instruktion som för närvarande exekveras.
2. Avbrott stängs av och processorn försätts i ett privilegierat systemtillstånd.
3. Adressen till nästa instruktion läggs i aktuellt `1r`-register, finns ett för varje tillstånd (mode)
4. Processorn hoppar till att exekvera kod från en fördefinierad adress (avbrottsrutin)

Hur använder man avbrott?

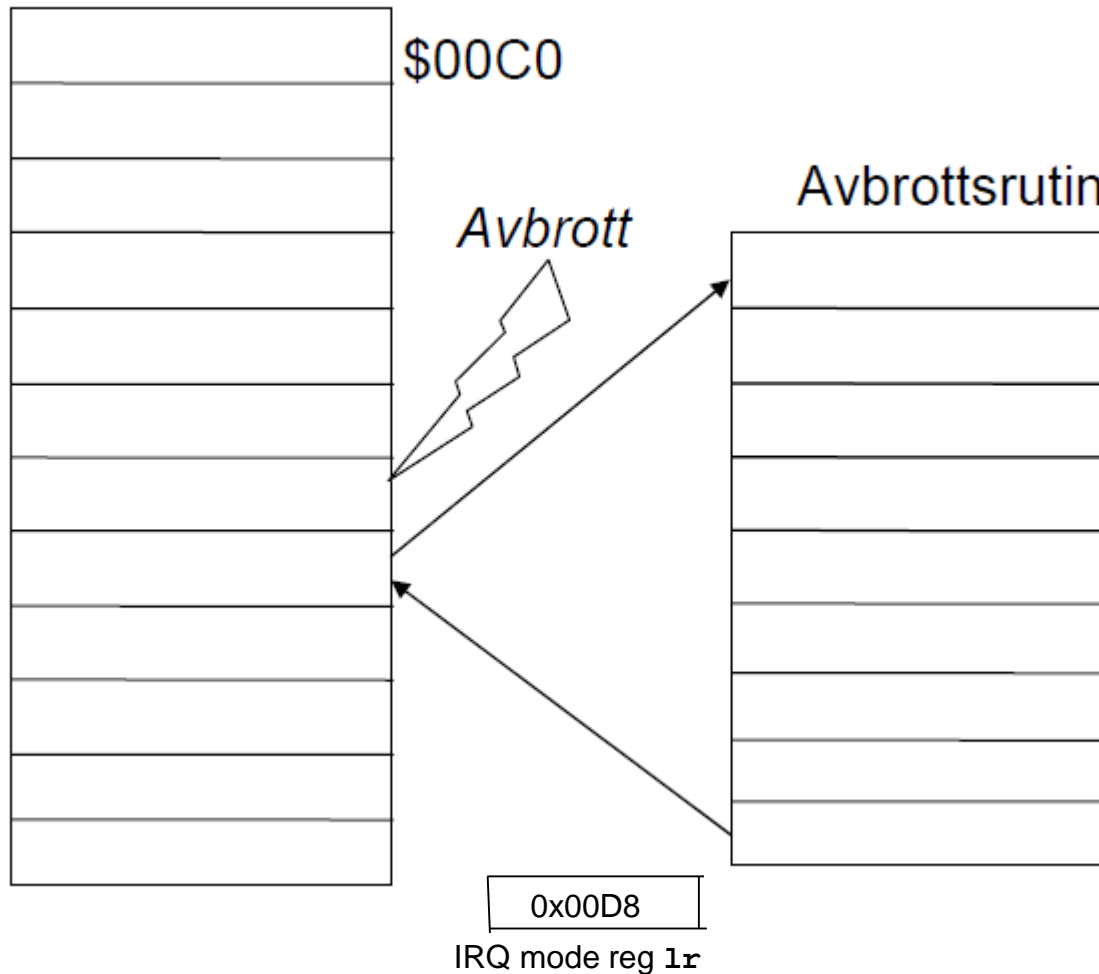
1. Koppla en enhet till någon avbrottsgenerande ingång på GPIO (t ex en knapp)
2. Skriv en rutin som ska köras när avbrottet kommer och lägg den på en specifik adress (Eftersom ni kommer att skriva avbrottshanteringen under linux ser ni aldrig den adressen explicit, den fastställs när avbrottsrutinen registreras i kärnan och associeras med namnet på den registrerade rutinen)
3. Använd `1r`-innehållet som återhoppadress (obs! det är inte "vanliga" `1r` i user mode, men det syns ingen skillnad i koden)
4. Tänk på att spara undan registerinnehåll i avbrottsrutinen, man vet ju inte på förhand när ett externt avbrott kommer att ske 😊

Använda avbrott, forts...

5. Initiera avbrottet i huvudprogrammet med hjälp av statusregistret (dvs tillåt det här avbrottet, överlåts till linux i labben).
6. Starta programmet.
7. Tryck på knappen så att avbrottsrutinen körs.

Avbrottshantering – schematiskt (för externt avbrott (IRQ))

Huvudprogram



Experiment med, SWI

- I C finns ett standardbibliotek har funktioner som använder OS:ets API för att göra systemanrop (mjukvaruavbrott som överlämnar kontrollen till OS:et)
- `write()` är exempel på en sådan funktion

Exempel på anrop:

```
write(1, text, sizeof(text));
```

1 är filedescriptor för stdout (terminaldisplayen)

- Detta kan vi också göra i assembler

Experiment med, SWI (forts)

```
.data
```

```
text: .asciz "Hello world\n"
```

```
after_text:
```

```
.text
```

```
...
```

```
MOV r0, #1 /* First argument: 1 */
```

```
LDR r1, =text
```

```
/* Second argument: string addr */
```

```
LDR r2, =after_text
```

```
SUB r2, r2, r1 /* Third argument: sizeof(text)*/
```

```
BL write
```

```
/* write(1, text, sizeof(text));*/
```

```
...
```

Om vi vet SWI-nr kan vi göra anropet helt själva i assembler

För att göra det behöver vi veta vilket nummer systemanropet **write** har i OSet på Pl.

- Den informationen finns listad för alla olika systemanrop i filen:
`/usr/include/arm-linux-gnueabi/hf/asm/unistd.h`
- Det numret ska placeras i r7 före **SWI** (write har 4 i linux),
- Operanden ignoreras vid **SWI**
- De andra parametrarna placeras som vid C-anropet. Vi får då följande kod.



Om vi vet SWI-nr kan vi göra systemanropet (PI ed)

`.data`

`text: .asciz "Hello world\n"`

`after_text:`

`.text`

`...`

```
MOV r0, #1          /* First argument: 1 */
LDR r1, =text
                    /* Second argument: string addr */
LDR r2, =after_text
SUB r2, r2, r1 /* Third argument: sizeof(text) */
MOV r7, #4        /* select system call 'write' */
SWI #0            /* perform the system call */
```




Motsvarande `swi`-anrop i ARMSim

Här funkar det lite annorlunda:

- Operanden till `swi` ska vara ett tal (2 för att skriva till stdout)
- Adressen till en nullterminerad sträng ska ligga i `r0`



Kod för ARMSim

.data

```
text: .asciz "Hello world\n"
```

.text

...

```
LDR r0, =text /* prepare system call */
```

```
SWI 2
```

```
/* perform the system call */
```

...

Avbrottsrutin vs subrutin

- Subrutinanrop sker i koden på samma ställe varje gång – predikterbart
- Avbrottsrutin anropas implicit genom yttre eller inre händelse, kan ske när som helst, kan ej förutses

dvs **extra viktigt** att avbrottsrutinen ombesörjer att spara undan och återställa *alla* använda register!!!