Prekinitve

→ to so dogodki, ki prekinejo izvajanje trenutno izvajajočega se programa in CPE takrat preklopi na rutino podprogram, ki se potem vrne v prekinjeni program

→ Prekinitve morajo biti TRANSPARENTNE

↘ "prekinjeni program ne sme vedeti, da je bil prekinjen"

↙ Kaj to pomeni?

1. da vsak program, ki ga CPE izvaja, le-ta hrani lastni kontekst

↳ - vsebina vseh programsko vidnih registrov
- vsebina PC
- vsebina SP
- vsebina statusnega registra
⋮

Kako CPE prekinemo?

↙ ↓ ↘

aktiviramo t.i. IRQ signal (CPE ima vhod za ta signal)

⇓

OD ZUNAJ (INTERRUPTS)

interni dogodki v CPE med izvajanjem ukazov

- npr:
- Neveljaven ukaz
- napaka pri dostopu do pomnilnika
- deljenje z 0
⋮

programski prekinitve

↙ 'SW' ukazi

↘ postavimo en namenski bit v kont. registru

⋮

OD ZUNAJ
PASTI (FAULTS) +TRAPS

OD ZNOTRAJ
(SOFTWARE INTERRUPTS)

pri ARM-u vsem pravijo IZJEME (EXCEPTIONS)

Kako CPE PSP? pridobi naslov

→ Vektorske prekinitve

PC ← M[definirani_naslov + 4 × ID (8)]

PC ← definirani_naslov + 4 × ID (8)

npr.:
PC ← 0x0000 0000 + 4 × ID

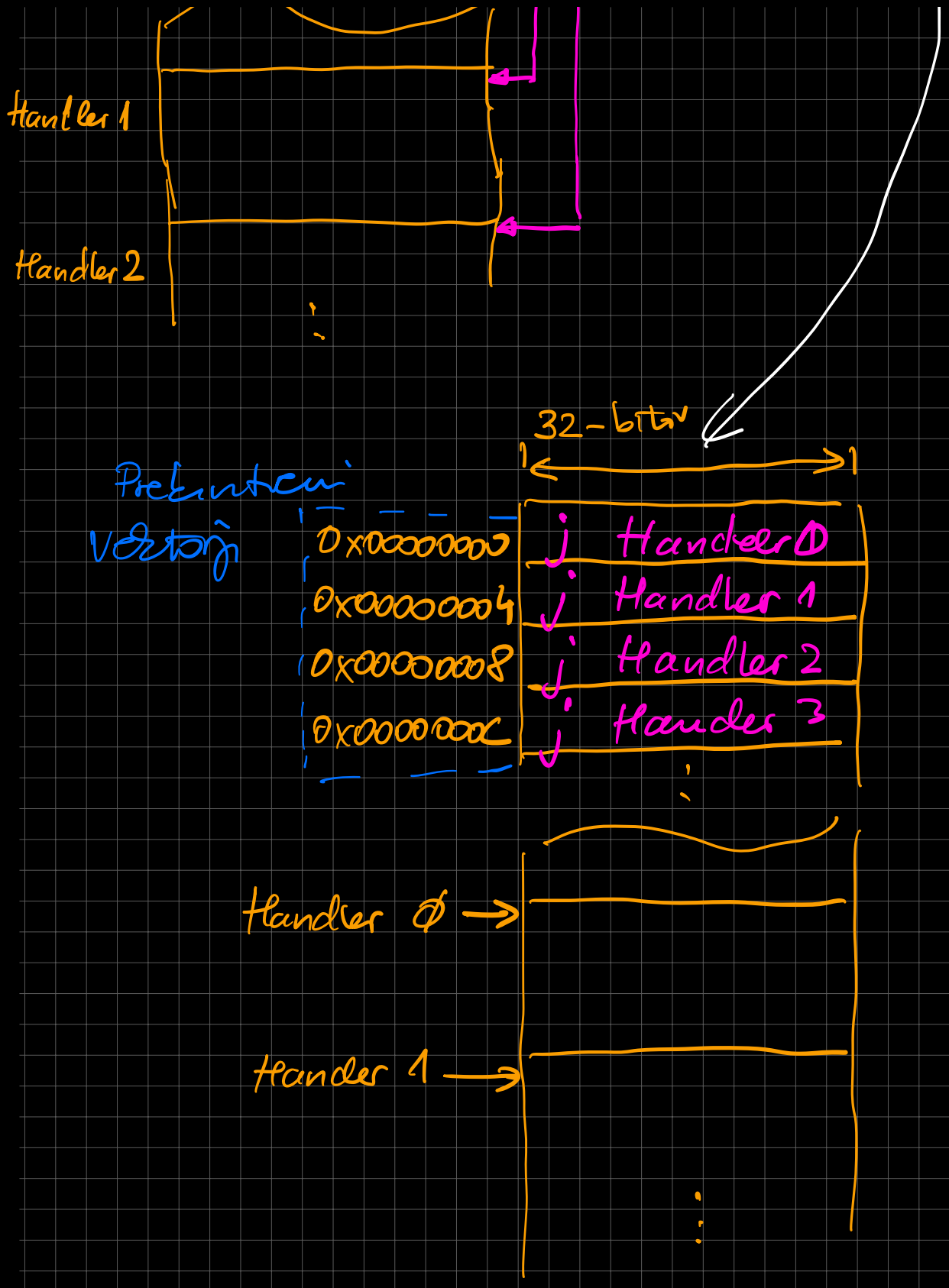
npr.:
PC ← M[0x0000 0000 + 4 × ID]

ARM Cortex

RISC-V

0x0000 0000 Handler 0
0x0000 0004 Handler 1
0x0000 0008 Handler 2
Handler 3

→ Tabela prekinitvenih vektorjev


Handler 1
Handler 2
32-bitov
Prekinitveni vektorji
0x00000000
0x00000004
0x00000008
0x0000000C
j Handler 0
j Handler 1
j Handler 2
j Handler 3
Handler 0
Handler 1

Izjeme pri ARM Cortex-M procesorjih

ARM Cortex-M jedra imajo:

1. do 16 notranjih izjem
2. do 240 zunanjih izjem

Vsaka izjema ima: ID (1 ... 256), Prioriteto

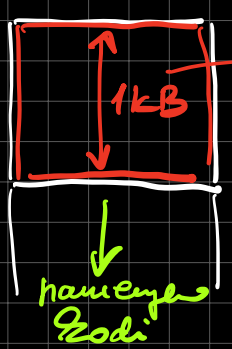
ID	Ime	Prioriteta	Naslov vektorja	Aktivacija
1	Reset	-3	0x00000004	ASINH.
2.	NMI	-2	0x00000008	AS INH
3.	Hard Fault	-1	0x0000000C	SINH.
4	Mem Manage	nastavljiva (0-15)	0x00000010	SINH
5	Bus Fault	nastavljiva (0-15)	0x00000014	SINH
6	Usage Fault	nastavljiva (0-15)	0x00000018	SINH
7-10	—	—	—	—
11	SVCall	nastavljiva (0-15)	0x0000002C	nastavljiva
12-13	—	—	—	—
14	PendSV	nastavljiva (0-15)	0x00000038	nastavljiva
15	SysTick	nastavljiva (0-15)	0x0000003C	Async.
16 →	IRQs	nastavljiva	0x00000040 →	Async.

Prioriteta: - nižje število prioritete ⇒ višja prioriteta

- če prideta 2 ali več prekinitev z isto prioriteto, CPE vzame tisto z manjšim ID-jem ⇒ manjši ID = višja prioriteta

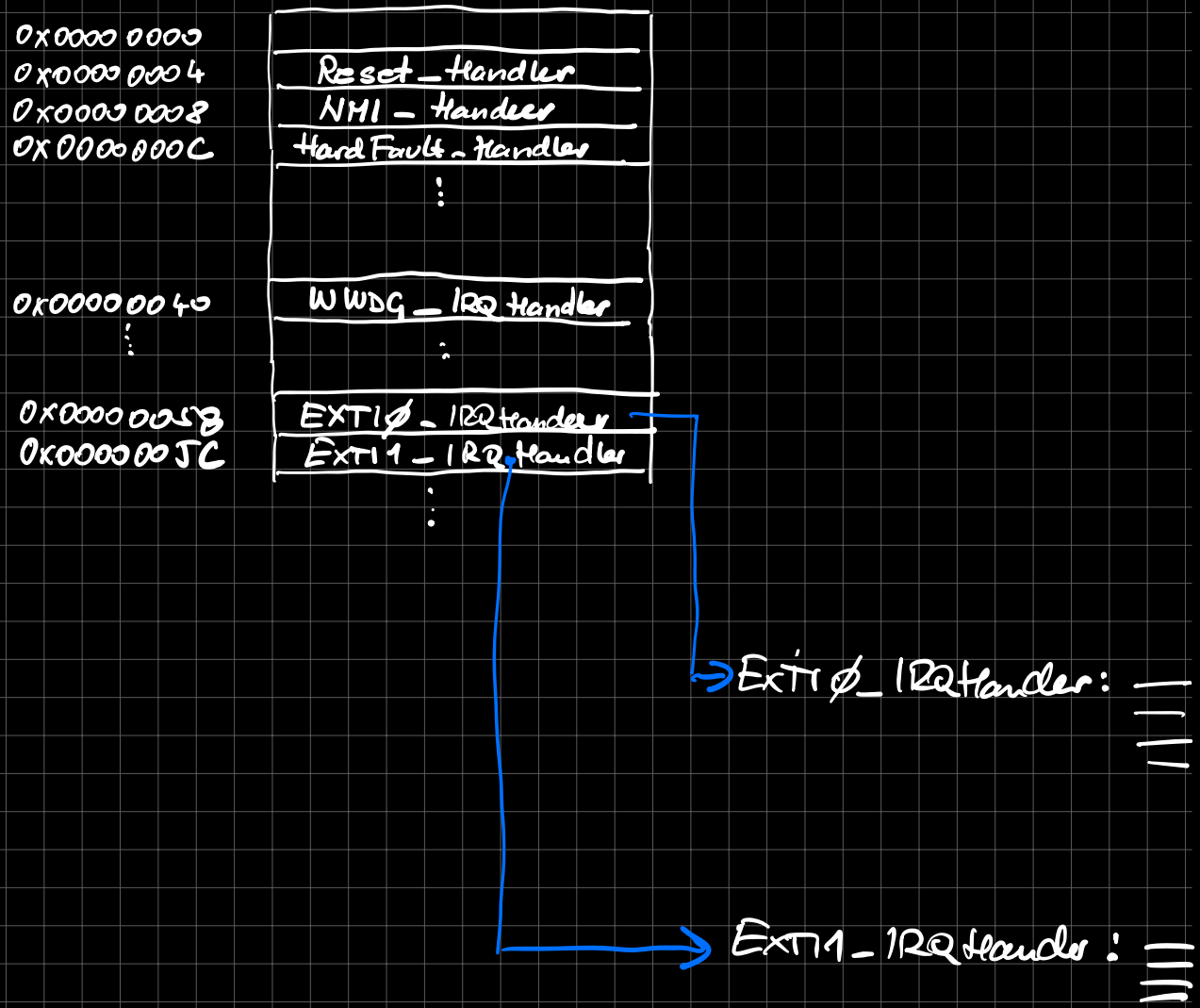
Vektorska tabela

ARM Cortex-M jedro predvideva, da se tabela prekinitvenih vektorjev začne na naslovu 0x0000 0000



PC ← M[0x0000 0000 + 4 x ID]

Tabela prekinitvenih vektorjev



VSTOP IN IZSTOP IZ PREKINITVE (IZJEME)

Programski model CPE:

16 splošno-namenskih registrov:

R0, R1, ..., R12, R13, R14, R15

- R13 ⇓ SP
- R14 ⇓ LR
- R15 ⇓ PC

Še en statusni register: xPSR (Program Status Register)

→ Kontekst

1. VSTOP:

– Cortex shrani polovico konteksta, in sicer tisto polovico za katero obstaja velika verjetnost da se bo v PSP umazala:

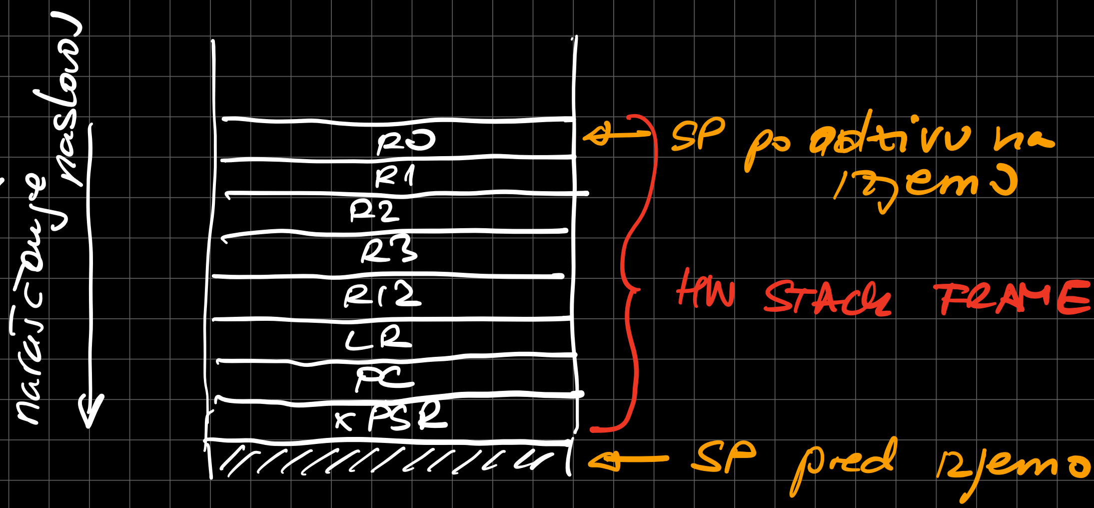
NA SKLAD (FULL-DESCENDING):

- R0, R1, R2, R3 (argumenti)
- R12
- LR
- PC
- xPSR

FULL → SP kaže na zadnji vstavljen podatek

DESCENDING → sklad narašča v smeri padajočih naslovov

ob vstopu v prekinitev:



To CPE naredi sama, HW, ne da bi izvajala ukazov

↳ tej operaciji pravimo STACKING

Istočasno s stacking operacijo:

PC ← M[0x0000 0000 + 4x ID]

↳ v PC se zapiše prek. vektor

LR ← 0xFFFF FFF x

28 enic (Exception Return vrednost)

2. CPE začne izvajati PSP

→ če PSP uporablja zaščitene registre R4-R11, mora prepisane (?) na začetku PSP porinuti.

Ne registar na sklad in jim predhodno obhrani s sklada

3. VRNITEV:

Za vračanje iz PSP se uporabi ukaz, ki izvede:

PC ← LR

CPE ima trdo-ožičeno logiko, ki preveri, ali je v LR zapisanih 28 bitov enakih 1

Če je, potem v PC ne prepiše LR (kljub ukazu), temveč začne izvajati DESTACKING OPERACIJO

sam s sklada pobere (v tem vrstnem redu)

R0, R1, R2, R3, R12, LR, PC, xPSR

povratni naslov, ki zdaj v PC