

Výuka s RISC-V od instrukce po virtuální paměť, QtRvSim i FPGA

LinuxDays 2026

Pavel Píša

`pisa@fel.cvut.cz`

Petr Štěpán

`stepan@fel.cvut.cz`

založené na 10. přednášce z předmětu B35APO – Architektura počítačů na ČVUT FEL
a bakalářské práci Adama Němce implementující MMU pro QtRvSim

License: CC-BY-SA



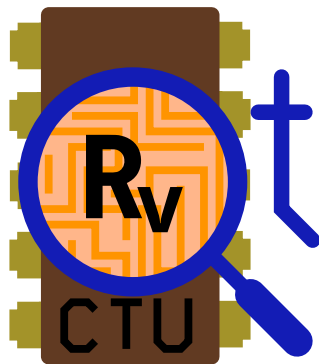
4. October, 2026

Outline

- 1 QtRvSim
- 2 Módy procesoru
- 3 Virtuální paměť
- 4 Virtualizace paměti – příklad
- 5 Translation Lookaside Buffer
- 6 64 bit mód

QtRvSim – simulátor RISC-V procesoru pro výuku principů počítačů

- Simulátor RISC-V procesoru s vizualizací průchodu instrukcí stupni zřetěženého zpracování, predikce skoků, stavu vyrovnávacích pamětí a dalších prvků
- Vyrobený jako pomůcka pro pochopení principů činnosti a architektury počítače
- Autoři: Karel Kočí, Pavel Píša, Jakub Dupák, Jiří Štefan, Adam Němec
- Simulátor bude použitý v interaktivních demonstracích během přednášky
- Online verze simulátoru a odkazy na stažení desktopových verzí a další materiály souvisejících předmětů na Českém vysokém učení v Praze <https://comparch.edu.cvut.cz/>



QtRVSim - odkazy na zdroje a prezentace

- Instalace Windows, Linux, Mac <https://github.com/cvut/qtrvsim/releases>
- Ubuntu <https://launchpad.net/~qtrvsimteam/+archive/ubuntu/ppa>
- Suse, Fedora and Debian <https://software.opensuse.org/download.html?project=home%3Ajdupak&package=qtrvsim>
- Online version <https://comparch.edu.cvut.cz/qtrvsim/app/>
- *QtRVSim - RISC-V Simulator with Cache and Pipeline Visualization*, RISC-V International Academic and Training SIG meeting, 2023, recording https://youtu.be/J6AcPZZ_ISg
- *QtRVSim—Education from Assembly to Pipeline, Cache Performance, and C Level Programming*, FOSDEM 2023, Brussels, https://fosdem.org/2023/schedule/event/rv_qtrvsim/
- Dupák, J.; Píša, P.; Štěpanovský, M.; Kočí, K.: *QtRVSim – RISC-V Simulator for Computer Architectures Classes*, In: embedded world Conference 2022, Haar: WEKA FACHMEDIEN GmbH, 2022. p. 775-778. ISBN 978-3-645-50194-1. (PDF) (Slides)

Outline

- 1 QtRvSim
- 2 Módy procesoru**
- 3 Virtuální paměť
- 4 Virtualizace paměti – příklad
- 5 Translation Lookaside Buffer
- 6 64 bit mód

Základ operačního systému

Moderní operační systém se neobejde bez dvou základních funkcí procesoru:

- uživatelský a systémový mód procesoru.
 - Uživatelský mód zajistí, že uživatelské programy nemohou číst ani měnit nastavení systému - nemožnost přecházet tajné informace, klíče, atd.
 - Systémový mód zajistí, že jádro operačního systému (často bohužel i ovladače HW) může všechno
- Virtualizace paměti
 - je řešením, jak oddělit systémovou část paměti a uživatelské programy
 - navíc lze oddělit i uživatelské procesy navzájem

Různé procesory používají k ochraně programů různé přístupy.

My si probereme přístup RISC-V procesoru

Módy procesoru RISC V

RISC V procesor má tři úrovně ochrany (o virtualizaci nemluvě):

- Machine Mode (M-mode): Nejvyšší úroveň oprávnění. V tomto režimu běží kód s přímým přístupem k hardwaru (bootloader, firmware, zabezpečené monitorovací programy). M-mode je povinný pro všechny implementace RISC-V.
- Supervisor Mode (S-mode): Střední úroveň oprávnění, určená pro běh operačních systémů. S-mode může využívat virtuální paměť a mechanismy ochrany paměti.
- User Mode (U-mode): Nejnižší úroveň oprávnění, ve které běží uživatelské aplikace. Aplikace v tomto režimu nemají přímý přístup k hardwaru a pro privilegované operace musí volat služby operačního systému (v S-mode)

Všechny procesory RISC V musí implementovat M-mode, ostatní úrovně nemusí být implementovány.

Změny módu procesoru RISC V

Jak přechází RISC V mezi úrovněmi ochrany:

- Zvýšení úrovně
 - Přechod $U \rightarrow S$, $U \rightarrow M$ nebo $S \rightarrow M$
 - Nejnebezpečnější jsou přechody $U \rightarrow S$, $U \rightarrow M$.
 - Musíme zabránit tomu, aby uživatelský program běžel v módu S nebo M
 - Kdy je potřeba použít S nebo M mód?
 - Když potřebujeme vyřešit spolupráci s periférií na základě přerušení.
 - Když uživatelský program chce použít HW (zapsat/číst do/z souboru, poslat paket po síti, atd.) – Pro tuto činnost potřebujeme speciální instrukci ecall - systémové volání.
 - Nesmíme dovolit spustit uživatelský program v módu M nebo S
 - Adresa pro přechod do S módu je uvedena v registru stvec a pro přechod do M módu mtvec
 - Lze nastavit, zda se vybrané přerušení bude obsluhovat v M nebo S módu

Módy procesoru RISC V

Jak přechází RISC V mezi úrovněmi ochrany:

- Snížení úrovně
 - Přejít $M \rightarrow S$, $M \rightarrow U$ nebo $S \rightarrow U$
 - Toto není nebezpečné.
 - Snížení úrovně znamená ztrátu některých práv.
 - Speciální instrukce `mret` a `sret`, které zajistí přechod do nižší úrovně.
 - Úroveň je nastavena v systémovém registru `mstatus.mpp` nebo `sstatus.spp`.
 - Adresa, kam se má instrukce skočit je v registru `mepc` nebo `sepc`.
 - Tento přechod je vlastně návratem z obsluhy přerušení, nebo systémového volání
 - I v případě startu, kdy nikdo systémové volání nezavolal.

Přepnutí do U módu

- Po zapnutí procesoru je vždy nastaven M mód (Machine mód)
- Přepnutí do U módu se provede takto
 - do registru `mstatus` nastavíme, že se chceme přepnout do U módu, hodnota 0 v poli MPP (Machine Previous Privilege)
 - Poznámka: registr slouží k zaznamenání módu, ze kterého jsme se dostali do M módu, abychom se tam mohli vrátit
 - Protože náš procesor se právě spustil, přepnutí do U módu simulujeme jakoby návratem do U módu instrukcí `mret`
 - Pokud bychom se chtěli přepnout do S módu nastavili bychom do MPP=1
 - do registru `mepc` nastavíme, jakou adresu chceme v U módu spustit
 - opět tento registr slouží k zaznamenání adresy instrukce, která vyvolala výjimku nebo byla přerušena, před spuštěním obsluhy v M módu
- provedení instrukce `mret` nás přepne do U módu na adresu `mepc`

Přepnutí do S/M módu příklad

Pro přechod do S nebo M módu z U módu se musí nejdříve v M/S módu nastavit jediná vstupní adresa, která se vykoná v M/S módu:

- `stvec` nebo `mtvec` adresu programu, který se bude spouštět při všech přepnutích
 - jedná se o jednu obslužnou rutinu, která zjistí, co se stalo (HW přerušení, výjimka, `ecall`) a v případě `ecall` podle hodnoty registru `a7` rozlišuje, co uživatel chce provést
- pokud chceme přecházet z U módu do S módu, musíme nastavit v registru `medeleg` přepnutí obsluhy dané výjimky v S módu
- na závěr povolíme přechody do S nebo M módu v registru `sstatus`, nebo `mstatus`

Uživatelský program:

- provedení instrukce `ecall` přepne procesor do S/M módu (podle `medeleg`) a spustí nastavený program (`stvec/mtvec`)

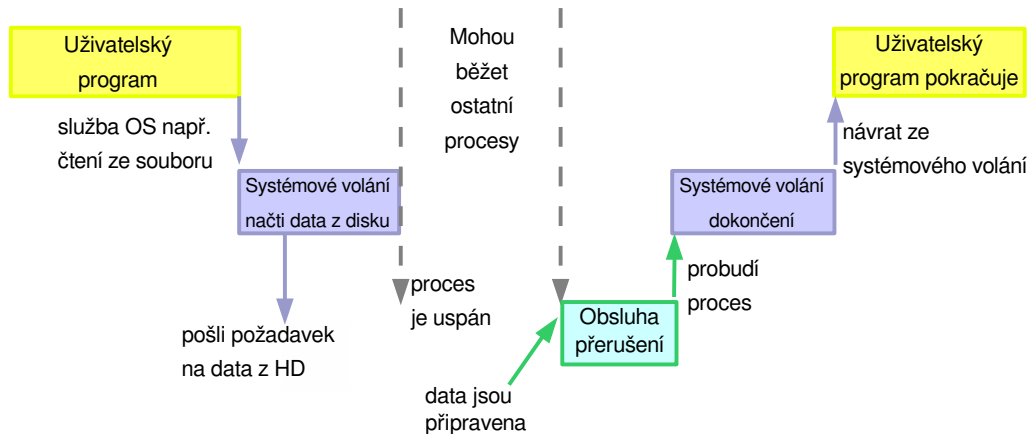
Základy ochrany OS

- Uživatelský proces nemůže přímo přistupovat k HW počítače.
- Uživatelský proces musí požádat OS o zpřístupnění HW, nebo vyřízení požadavku.
- To co OS umožňuje procesům jsou pouze systémová volání (system calls).
- Na rozdíl od volání funkce neznáme adresy funkcí v jádře.
- Systémové volání se vybírá číslem systémového volání.
- Vlastní zavolání funkce je vyvolání přerušení nebo výjimky specializovanou instrukcí:
 - x86 – 32 bitů používá přímo `int 0x80` – vyvolej přerušení číslo 0x80.
 - x86 – nověji specializované instrukce `sysenter/syscall` – podobný mechanismus, ale nevyužívá uživatelský zásobník, složitý mechanismus bran a přistupuje k méně paměti → je rychlejší.
 - RISC-V – specializovaná instrukce `ecall` – obdobně jako při přerušení.
- Při přerušení/výjimce se volá obslužná rutina v privilegovaném režimu (typicky systémovém) z adresy, která je nastavená OS (na RISC-V registry `mtvec/stvec`), uživatel ji nemůže měnit.

Systémové volání

Uživatelský program nastaví parametry volání do registrů vyvolá speciální přerušeni/výjimku (na RISC-V `ecall`).

OS podle čísla systémového volání zavolá odpovídající funkci.



ABI systémového volání

- API – application programming interface – popis, jaké funkce může Váš program zavolat z knihoven.
 - API je definováno pro jazyk C hlavičkovými soubory.
 - API definuje i co dané funkce dělají, jaké vrátí hodnoty, jak se chovají při chybě
 - zkuste např. `man 2 read`
- ABI – application binary interface – popis jaké registry a jaké instrukce použít
- ABI pro systémová volání na architektuře RISC-V
 - a7 – obsahuje číslo systémového volání (přehled např. <https://jborza.com/post/2021-05-11-riscv-linux-syscalls/> nebo <https://marcin.juszkiewicz.com.pl/download/tables/syscalls.html>)
 - a0 až a5 – parametry systémového volání (Linuxové systémové volání má maximálně 6 parametrů)
 - systémové volání se provede instrukcí `ecall`
 - a0 obsahuje návratovou hodnotu systémového volání
 - pokud by mělo vracet více dat (např. čtení ze souboru) musí uživatel zadat ukazatel na buffer a velikost bufferu, kam OS zapíše data.

Příklad Hello world!

```
.global _start
.text
_start:
#  write(1, "Hello world!\n", 13);
    addi a7, zero, 64
    addi a0, zero, 1
    la    a1, text_1
    addi a2, zero, 13
    ecall
```

```
final:
#  exit(0);
    addi a7, zero, 93
    addi a0, zero, 0
    ecall
    ebreak
    j final
.data
# store ASCII text, no termination
text_1: .ascii "Hello world!\n"
```

Outline

- 1 QtRvSim
- 2 Módy procesoru
- 3 Virtuální paměť**
- 4 Virtualizace paměti – příklad
- 5 Translation Lookaside Buffer
- 6 64 bit mód

Cíle virtualizace paměti

Cíle virtualizace:

- Ochrana jádra operačního systému
 - část virtuálního prostoru je chráněna proti přístupu v U módu
 - tato část prostoru je přístupná pouze z S nebo M módu
- Izolace procesů
 - Každý proces – běžící program – má svůj virtuální prostor.
 - Proces se pohybuje pouze ve virtuálním prostoru.
 - Součástí virtuálního prostoru je i paměť rezervovaná pro OS, aby mohl procesor skočit na obsluhu výjimky a přerušení.

Cíle virtualizace paměti

Další cíle virtualizace:

- Efektivní využití RAM
 - virtuální prostory všech procesů je potřeba poskládat do fyzické paměti - RAM
 - ukončením procesu vznikají díry volné paměti RAM různých velikostí.
- Podpora sdílení paměti
 - Text programů (strojové instrukce) mohou být sdílené různými procesy, které vykonávají stejný program.
 - Text knihoven (strojové instrukce) jsou sdíleny, aby se ušetřila fyzická paměť.
- Mapování zařízení a souborů do virtuálního procesu
 - Pro rychlejší přístup k zařízení je možné fyzickou adresu periferie mapovat do virtuálního prostoru procesu.
 - Při čtení nebo zápisu do souboru se obsah kopíruje do paměti OS, pro zrychlení je možné mapovat tuto část OS do virtuálního prostoru procesu.

Virtualizace paměti – stránkování

Nejpoužívanějším způsobem virtualizace je stránkování.

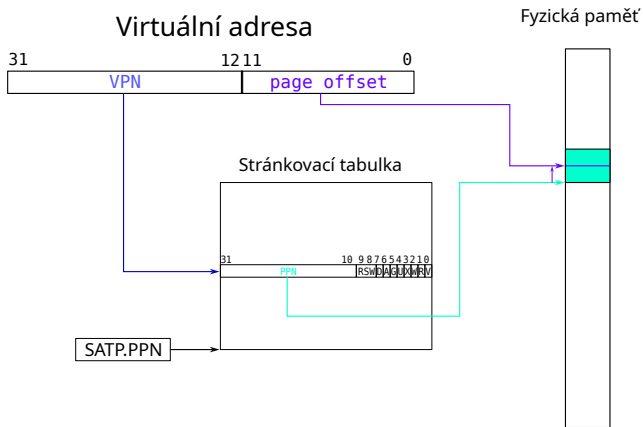
Základní princip stránkování:

- paměť se rozdělí na stejně velké části
 - část virtuální paměti se nazývá stránka - page.
 - část fyzické paměti se nazývá rámec - frame.
- velikost stránky a rámce je stejná a bývá 4 kiB tedy 4096 bajtů.
- díky tomu lze stránku umístit do libovolného rámce

Virtualizace paměti – stránkování

Základní princip stránkování:

- umístění stránky do rámce se definuje pomocí tabulky stránek (page table):
 - tabulka má položku pro každou stránku
 - položka obsahuje:
 - příznak V (validity bit), zda je stránka ve fyzické paměti
 - příznaky přístupu R, W, X, U, a další
 - číslo rámce fyzické paměti



Princip stránkování

Princip stránkování

- paměť se rozdělí od počátku na stejně velké části, většinou $4kiB = 4096B$
 - každá část paměti začíná na adrese $k \cdot 4096$ a končí na adrese $k \cdot 4096 + 4095$
 - pro zjištění čísla stránky/rámce se využije operace $adr // 4096 = adr \gg 12$
- velikost tabulky je dána velikostí virtuálního prostoru, protože každá stránka musí mít položku ve stránkovací tabulce:
 - velikost 32-bitového virtuálního prostoru je $2^{32} = 4GiB$
 - počet stránek $\frac{2^{32}}{2^{12}} = 2^{20} = 1Mi = 1024 \cdot 1024$
 - velikost jedné položky v tabulce jsou 4 bajty
 - velikost celé tabulky stránek je 4 MiB
- velikost tabulky v 64-bitovém prostoru
 - velikost 64-bitového virtuálního prostoru je $2^{64} = 16EiB$ (16 Exbibajtů = $16 \cdot 1024$ Pebibajtů)
 - počet stránek $\frac{2^{64}}{2^{12}} = 2^{52} = 4Pi$
 - velikost jedné položky je 8 bajtů
 - velikost tabulky stránek je 32 PiB

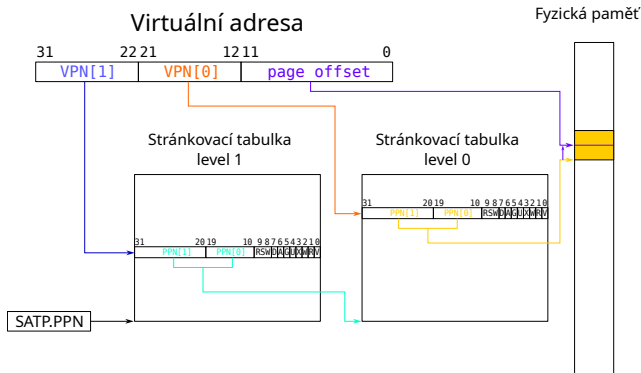
Stránkovací tabulka

Dvou-úrovňová stránkovací tabulka

- velikost stránkovací tabulky je nepřiměřeně velká
- většina položek je zbytečná, obsahuje samé 0
 - pouze stránky, které jsou použity pro program, nebo data jsou mapovány do fyzické paměti
 - ostatní stránky nejsou použity a nejsou mapovány do fyzické paměti
- tabulka stránek se rozdělí na stejně velké podtabulky a přidá se nová úroveň – tabulka, která bude určovat, zda a kde odpovídající podtabulka existuje

Stránkovací tabulka

- Adresní prostor Sv32 – 32bitové ukazatele, 32bitový prostor
 - počet stránek
 $\frac{2^{32}}{2^{12}} = 2^{20} = 1\text{Mi} = 1024 \cdot 1024$
 - rozdělíme tabulku stránek na 1024 malých tabulek, každá obsahuje 1024 položek
 - pokud by malá tabulka měla pouze nulové hodnoty, nemusí existovat a zabírat paměť.
 - přidáme další nadřazenou tabulku, která obsahuje 1024 položek, zda existuje odpovídající malá tabulka a ve kterém rámci tato malá tabulka je



Všimněte si, že velikost malé tabulky i velikost nadřazené tabulky je přesně velikost rámce
 (4 · 1024 = 4096 bajtů)!

RISC V Sv32



Položka stránkovací tabulky

- položka má velikost 4 bajty tj. 32 bitů
- potřebujeme zakódovat číslo rámce:
 - číslo rámce má v Sv32 – 22 bitů
 - to znamená, že velikost fyzické paměti může být až $22 + 12 = 34$ tedy 16 GiB (k číslu rámce přidáme posun uvnitř stránky)
 - K čemu je to dobré, když program může využít maximálně 4 GiB (každá virtuální adresa má pouze 32 bitů)?
 - V počítači může běžet hodně programů a celkové využití paměti tedy lehce přesáhne 4 GiB
- také potřebujeme zakódovat přístupová práva ke stránce – příznaky:
 - Příznak V - validity, nejdůležitější příznak definující, že číslo rámce je validní
 - Příznaky R,W,X,U - práva pro čtení, zápis, spuštění instrukce, přístup pro uživatele v U módu
 - Další příznaky slouží k zjištění používání stránky - A accessed, D dirty.

Outline

- 1 QtRvSim
- 2 Módy procesoru
- 3 Virtuální paměť
- 4 Virtualizace paměti – příklad**
- 5 Translation Lookaside Buffer
- 6 64 bit mód

Virtualizace paměti - příklad

Abychom mohli použít stránkování v 32 bitovém procesoru musíme provést následující kroky:

- Inicializujeme stránkovací tabulku
 - Do virtuálního prostoru namapujeme program, abychom měli kam skočit
 - Připravíme si také data, abychom se o ně pak nemuseli starat
- Nastavíme v registru satp číslo rámce, ve kterém se nachází kořenová tabulka úrovně 1
- Nastavíme v registru satp, že se má používat virtualizace v U i S módu
- Přepneme se do U nebo S módu
 - předpokládáme, že se procesor právě spustil a nachází se v M módu
 - v M módu nelze použít virtualizaci paměti

Virtualizace – příklad

Připravili jsme si jednoduchý program, který bude kopírovat data z jedné stránky do druhé. Nejdříve musíme vytvořit a inicializovat stránkovací tabulku

- Vytvoříme si dvě místa v paměti, která musí začínat na násobcích 4096 a obsahovat 4096 bajtů
- První bude pro tabulku úrovně 1, tedy nadřazenou tabulku
- Druhá bude pro tabulku úrovně 0, tabulku, která už obsahuje informace o mapování virtuální paměti do fyzické paměti

```
.org 0x3000
# the page table directory
# start to be set into satp
l1_table:
    .word 0, 0, 0, 0

.org 0x4000
# the last level page table
# with PTE entries
l0_table:
    .word 0, 0, 0, 0
```

Inicializace stránkovací tabulky

Kolik paměti maximálně můžeme namapovat pomocí těchto dvou tabulek

- A celý virtuální prostor
- B 4 MiB
- C 1 MiB
- D 4 kiB

Inicializace stránkovací tabulky - příklad

Nejdříve musíme propojit stránkovací tabulky (část funkce `map_page`)

- Tabulky propojíme tak, že do tabulky úrovně 1 na první 4 bajty zapíšeme pozici tabulky úrovně 0.
- $PPN = (\text{adr}(l0_table) \gg 12) \ll 10$
- nastavení příznaku U a V, bity R,W,X musí být nulové (použito pro velké stránky 4 MiB).

příprava položky do tabulky

```
la t5, l0_table
```

```
srli t5, t5, 12
```

číslo rámce má 22 bitů

```
slli t5, t5, 10
```

práva pro přístup

```
ori t5, t5, 0x11
```

t0 - virtual address

```
srli t1, t0, 22
```

```
andi t1, t1, 0x3ff
```

odkaz na l1_table[t1]

vynásob 4 - velikost položky

```
slli t1, t1, 2
```

přičti začátek tabulky

```
la t2, l1_table
```

```
add t3, t2, t1
```

ulož položku do tabulky

```
sw t5, 0(t3)
```

Inicializace stránkovací tabulky - příklad

Kterou část virtuální paměti můžeme díky tomuto propojení využít

- A celý virtuální prostor
- B 0 – 4 MiB
- C 4 MiB – 8 MiB
- D 8 MiB – 12 MiB

Plnění stránkovací tabulky - příklad

Nyní si do virtuálního prostoru namapujeme na stejné adresy program:

- Protože program začíná na adrese 0x200 a vejde se do 3584 bajtů ($0x1000 - 0x200 = 0xE00 = 3584$), tak stačí namapovat jednu stránku 0 do rámce 0

- VPN = 0, PPN = 0

- práva musíme nastavit X pro možnost vykonávání programu

t0 virtuální adresa

```
srli t1, t0, 12
```

```
andi t1, t1, 0x3ff
```

odkaz na l0_table[t1]

```
slli t1, t1, 2
```

```
la t2, l0_table
```

```
add t3, t2, t1
```

setup page table entry (PTE)

```
srli t4, a1, 12
```

```
slli t4, t4, 10
```

setup page privileges in PTE

D = 0, A = 0, G = 0, U = 1

X = 1, W = 1, R = 1, V = 1

```
li t5, 0x1f
```

```
or t4, t4, t5
```

```
sw t4, 0(t3)
```

Plnění stránkovací tabulky - příklad

Náš ukázkový program kopíruje data z virtuální adresy 0x6000 na adresu 0x7000.

Fyzicky jsou ale zdrojová data na adrese 0x1000 a kopírovat se budou do fyzické paměti 0x2000.

- zavoláme map_page VPN = 6, PPN = 1
- zavoláme map_page VPN = 7, PPN = 2
- práva nastavíme pro data R,W pro obě stránky
 - Pokud by zdroj dat byly konstanty, můžeme pro zdrojovou stránku nastavit pouze práva pro čtení a tím ochráníme data proti zápisu

```
# map source_page to virtual  
# page address 0x6000  
li a0, 0x6000  
la a1, source_page  
call map_page  
  
# map target_page to virtual  
# page address 0x7000  
li a0, 0x7000  
la a1, target_page  
call map_page
```

Zapnutí virtualizace paměti - příklad

Dalším krokem je nastavení módu virtualizace, nastavíme stránkování typ Sv32:

- v registru satp (Supervisor Address Translation and Protection) nastavíme mód na vybranou hodnotu, tedy Sv32
- jakmile je registr satp nastaven na nenulovou hodnotu módu, překlad adres a ochrana paměti jsou spuštěné pro S a U módy
 - M mód má neustále přímý přístup do paměti

```
# PPN tabulky úrovně 1
```

```
la t0, l1_table
```

```
srli t0, t0, 12
```

```
# Setup 32-bit RISC-V page
```

```
# translation mode (Sv32)
```

```
li t1, 0x80000000
```

```
or t0, t0, t1
```

```
csrw satp, t0
```

```
sfence.wma
```

Zapnutí virtualizace paměti - příklad

Po přepnutí do S módu spustíme náš program, který bude mít text programu na virtuální adrese 0x200 a data na virtuálních adresách 0x6000 a 0x7000. Žádné další stránky nejsou namapovány, takže nelze žádný jiný program, než ten náš, spustit

- Přepnutí do U/S módu provedeme instrukcí `mret` tak, že nejdříve nastavíme na jakou adresu se přepneme
- dále nastavíme, který mód po `mret` bude vybrán

adresa, kam se "vrátit" z M módu

```
la t0, s_mode_start
csrw mepc, t0
```

Nyní již náš program běží s virtualizací paměti v S módu procesoru.

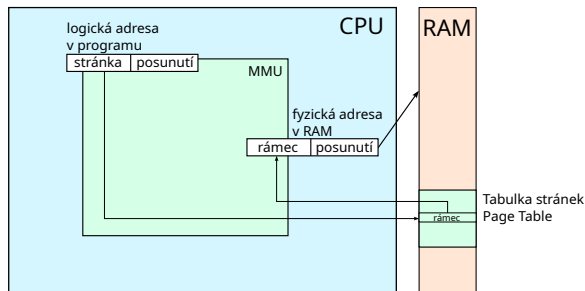
```
# načíst registr mstatus
csrr t1, mstatus
# clear MPP
li t2, 0xffffe7ff # ~(3 << 11)
and t1, t1, t2
# MPP = S
li t2, 0x800      # (1 << 11)
or t1, t1, t2
# SUM = 1 - enable CPU running in S-mode
# to access pages mapped as unprivileged
li t2, 0x40000    # (1 << 18)
or t1, t1, t2
# uloží hodnotu do registru mstatus
csrw mstatus, t1
# Switch CPU to S-mode, start S-mode code
mret
```

Outline

- 1 QtRvSim
- 2 Módy procesoru
- 3 Virtuální paměť
- 4 Virtualizace paměti – příklad
- 5 Translation Lookaside Buffer**
- 6 64 bit mód

Efektivita tabulky stránek

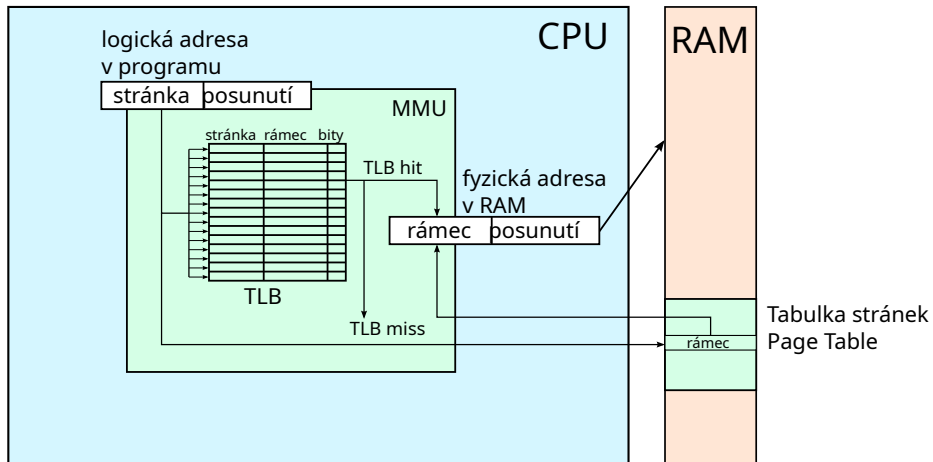
- každý přístup do paměti znamená, že je potřeba převést číslo stránky na číslo rámce, to znamená číst z RAM
- jedna instrukce může číst i dvakrát z paměti z různých stránek
- jedna instrukce tedy potřebuje $2 + 2 = 4$ přístupy do paměti
- při dvouúrovňovém stránkování se jedná o $3 + 3 = 6$ přístupů do paměti
- přístup do paměti velmi zdržuje



TLB

- Řešení – speciální rychlá cache paměť pro čísla rámců a čísel stránek – Translation Lookaside Buffer
- asociativní paměť – paměť adresovaná obsahem
 - oproti normální paměti, ptám se, kde v paměti je hodnota 15?
- TLB je asociativní paměť
- relativně malá kapacita, vysoká efektivita a zrychlení přístupu do paměti
- nevýhoda – obvodová složitost implementace TLB
- MMU se zeptá TLB znáš hodnotu rámce pro číslo stránky p ? Odpověď buď ano je to r (TLB hit), nebo neznám (TLB miss)

Tabulka stránek



Význam TLB

- Přístupová doba do RAM obvykle 10–100 cyklů procesoru
- Přístupová doba TLB – 0.5 – 1 cyklus procesoru
- Neúspěšnost TLB, TLB miss – 0.01 % – 1 %
- Příklad:
 - Přístupová doba do fyzické paměti $t = 30\text{cyklu}$, do TLB $t_{TLB} = 1\text{cyklus}$
 - Neúspěšnost TLB $\alpha = 0.01$ (jedno procento)
 - Stránkování bez TLB $t_{celkem} = 2 \cdot t = 60\text{cyklu}$
 - Průměrná doba přístupu do paměti s TLB
$$t_{celkem} = \alpha \cdot (t_{TLB} + 2 \cdot t) + (1 - \alpha) \cdot (t_{TLB} + t) = 31.3\text{cyklu}$$

Vliv TLB

- Velikost TLB 8–4096 položek
- Moderní procesory mají více-úrovň TLB – podobně jako úrovně cache
- Intel Core i7 má 64 TLB položek L1 první úrovně a 1536 TLB položek L2 úrovně
- I pro malé TLB je úspěšnost nalezení položky 99–99.99% (souvisí s principem lokality tj. prostorovou závislostí programů)
- Problém TLB je při změně procesu a tím i změně tabulky stránek
- Procesory umožňují speciálními instrukcemi ponechat stránku v TLB

Outline

- 1 QtRvSim
- 2 Módy procesoru
- 3 Virtuální paměť
- 4 Virtualizace paměti – příklad
- 5 Translation Lookaside Buffer
- 6 64 bit mód**

64-bitový mód

Co je nutné změnit pro 64-bitový mód?

- Rozsah čísla rámce

- V Sv32 je k dispozici 22 bitů pro číslo rámce, což umožňuje využít 16 GiB fyzické paměti (to není mnoho)
- Pro 64-bitový mód potřebujeme více bitů na číslo rámce, ale potřebujeme zachovat minimálně 10 bitů práv
 - Řešení: Velikost jedné položky v tabulce stránek bude 64 bitů tj. 8 bajtů

- Velikost tabulky

- V Sv32 je velikost virtuálního prostoru 2^{32} tj. 4 GiB, počet položek tabulky stránek je $2^{20} = 1024 \cdot 1024$ a velikost tabulky je 4 MiB
- Pro 64-bitový mód je velikost virtuálního prostoru 2^{64} , počet položek tabulky stránek je 2^{52} a velikost tabulky je 32 PiB
 - Řešení: Omezíme velikost prostoru a zavedeme více úrovní stránkovacích tabulek

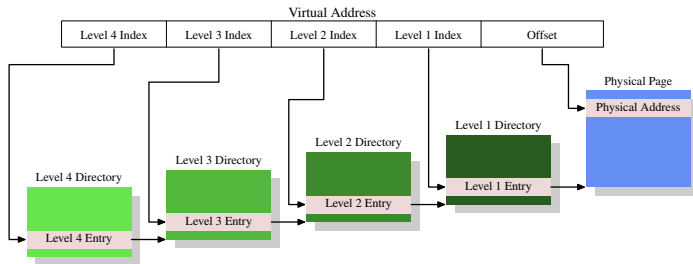
Stránkování Sv39

- velikost stránky je 4 KiB tj. 2^{12}
- každá tabulka všech úrovní se vejde přesně do jedné stránky
 - Velikost jedné položky v tabulce stránek je 8 bajtů, tzn. celkem se do jedné stránky vejde $512 = 2^9$ položek
- Stránkování Sv39 bude používat pouze spodních 39 bitů z adresy, tj. velikost virtuálního prostoru je 512 GiB.
 - Existují 3-úrovně stránkovacích tabulek - kódováno $3 \cdot 9$ bity
 - 12-bitů kóduje polohu uvnitř stránky
 - Celkem tedy 39 bitů virtuální adresy
 - Ostatní bity (nejvyšších 25 bitů) musí být rovny nejvyššímu použitému bitu (kanonický tvar adresy)

Proč nevzít rovnou 64 bitů virtuálního prostoru?

Stránkování Sv48, Sv57

- Každá další úroveň přináší zpomalení, protože je nutné načítat další informaci z paměti.
- Stránkování Sv48 zavádí 4-úrovně stránkovacích tabulek
 - Kódováno $4 \cdot 9 + 12 = 48$ bitů adresy
 - Velikost virtuálního prostoru je 256 TiB.



- Stránkování Sv57 zavádí 5-úrovň stránkovacích tabulek
 - Kódováno $5 \cdot 9 + 12 = 57$ bitů adresy
 - Velikost virtuálního prostoru je 128 PiB.

Konec prezentace a zopakování odkazu na simulátor QtRVSim a materiály

Online verze simulátoru, odkazy na stažení desktopových verzí a další materiály souvisejících předmětů na Českém vysokém učení v Praze

<https://comparch.edu.cvut.cz/>

