B35APO: Architektury počítačů

Lekce 09. Přerušení a události

Pavel Píša
pisa@fel.cvut.cz

Petr Štěpán
stepan@fel.cvut.cz

30. srpna, 2024

Obsah

Cíl dnešní přednášky

- Oblasti využití počítačů/propojení s reálným světem
- Obsluha periferií a vliv na zátěž procesoru
- Zpracování událostí, přerušení a výjimky
- Přímý přístup do paměti z periferií
- Konsekvence při použití skrytých pamětí, zpracování mimo pořadí, více procesorů

Zopakování již probraných stavebních bloků

- Centrální procesorová jednotka
- Paměť/úložiště organizované do hierarchie
  - zápisníková paměť (registry, rychlé)
  - skrytá paměť (cache, úroveň L1, L2, ...)
  - hlavní paměť (RAM – dnes typicky DDR)
  - vnější paměť, úložiště (disky, mžikové paměti – Flash)
- Propojení – sběrnice, sítě
  - ISA, PCI, PCIexpress, AXI

Oblasti, ve kterých lze bloky, procesory využít

- Zábava
  - histricky první sekvenční stroje, automatofon - hrací skříňky, hodiny, orloje, mechanismus z Antikythéry již 150 př. n. l.
  - dnes hrací automaty, herní konzole, simulátory, multimédia
- Výdělečná zaneprázdněnost/aktivity (busyness)
  - původně jen sčítačky, pokladny, dnes burzy cenných papírů, podnikové a bankovní informační systémy atd.
- Rozsáhlé matematické, vědecké a modelovací výpočty
  - globální klimatické prognózy a analýzy, jaderná fúze atd
  - strojové učení
- Komunikace
  - jako hlavní cíl síťové prvky, původně telefon pevný, mobilní
  - dnes součást téměř všech ostatních aplikací
- Automatizace, výroby, dopravy
  - původně automatické vzory tkalcovský stav, dnes sváření, robotika, autonomní automobily a vlaky
- a mnoho dalších oblastí

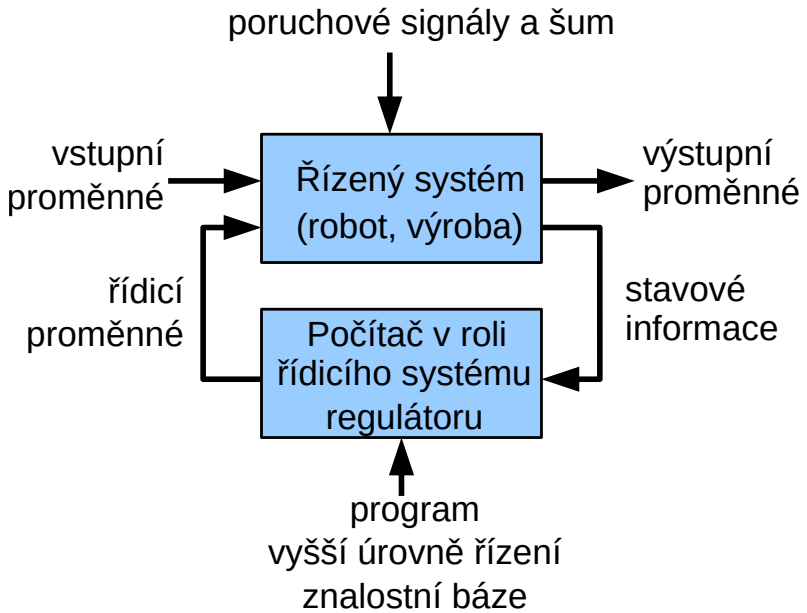
Počítač v řídicích aplikacích

Nahrazuje na míru sestavené regulátory.

Výhody:

1. často složitý řídicí proces, pravidla, vztahy
   - rychlý výpočet
2. levný díky sériové výrobě
3. velmi flexibilní
   - programování
4. snadná realizace hierarchického řízení
   - ovládání k dispozici
5. přesné/deterministické zpracování veličin
   - zobrazení hodnot a jeho limity
6. komplexní algoritmy
   - pouze paměťová a časová omezení

Počítač jako řídící systém



Různé požadavky na zpracování úloh, dat

- Dávkové zpracování
  - úloha řídí přístupy k datům podle toho, jak je potřebuje zpracovávat, stejně tak časování a pořadí uložení výstupů se řídí potřebami výpočtu
- Interaktivní
  - obvykle událostmi řízené v kombinaci podle požadavků, vstupů od uživatele nebo z jiného systému
- Řízení (technologií, robotů, automobilů) v reálném čase
  - sebepřesnější výstupy dodané pozdě nemají žádnou hodnotu (letadlo již havarovalo) nebo výrazně sníženou (nespojitý přenos videa)
  - stupeň integrity bezpečnosti (Safety Integrity Level – SIL) pokud hrozí újma na zdraví a majetku

Obsah

Vstupně-výstupní subsystém (opakování)

- Periferie pouze pro vstup
  - běžné: klávesnice, myš, videokamera
  - digitální vstupy, fyzikální veličiny – obvykle převedeny na analogové elektrickým signálem a následně A/D převodníkem na číselnou hodnotu přístupné na vstupním portu a dalších senzorech
- Pouze výstupné periferií
  - video výstup (2D, 3D + akcelerace), audio výstup
  - výstupy s fyzickým efektem, 3D tiskárna (rychlé prototypování), řízení technologických procesů (D/A převodníky, PWM) a mnohé jiné druhy pohonů
- Obousměrná komunikace
  - pevný disk, komunikační rozhraní
  - většina výše uvedených „jednosměrných“ periferií vyžaduje čtení a přístup pro zápis pro jejich nastavení, monitorování a parametry řízení

Metody přenosu dat mezi periferiemi a procesorem

- Programovaný vstup/výstup s dotazováním (anglicky programmed input/output – PIO with polling)
  - procesor se cyklicky dotazuje na stavové informace periferie jestli jsou dostupná vstupní data nebo místo ve výstupní vyrovnávací paměti
- Programovaný vstup/výstup řízený přerušením (anglicky programmed input/output PIO)
  - procesor/program/operační systém konfiguruje periferie, ale nečeká na data. Příchod dat je signalizován přerušením (asynchronní událost/výjimka). Data jsou čtena v obslužné rutině přerušení (interrupt service routine – ISR).
- Přímý přístup do paměti (anglicky direct memory access – DMA)
  - procesor/program/operační systém nastaví zdroj a cíl, přenos realizuje specializovaná jednotka, dokončení signalizované přerušením.
- Inteligentní periferie/řadiče
  - realizující operaci i s přenosem do paměti (bus master – DMA)

Obsah

Základní cyklus procesoru (opakování z lekce 2)

1. Počáteční nastavení – inicializace registru PC a PSW (případně i dalších) po zapnutí proudu nebo po resetu procesoru
2. Přečti instrukci z paměti z adresy PC
   - nastav PC na adresu sběrnice paměti
   - Přečti obsah ze sběrnice paměti do registru IR
   - $PC + l \rightarrow PC$, uprav PC, l je délka načtené instrukce
3. Dekóduj instrukci
4. Proveď instrukci
   - spočti adresu, vyber registry, načti operandy, proveď požadovaný výpočet ALU a ulož výsledky
5. **Zkontroluj zda není přerušení nebo výjimka**
   - pokud je výjimka důsledkem vykonávání dané instrukce, zajisti, že její výstupy nejsou uloženy a PC na ní dále ukazuje
   - přejdi na obslužnou rutinu tak, aby bylo možné po návratu v přerušené sekvenci instrukcí pokračovat
6. Opakuj od kroku 2

Výjimky a přerušení

- Výjimky (synchronní) – anomální nebo výjimečné situace znemožňují pokračování běhu sekvence instrukcí a vyžadující zvláštní zpracování, hlavní uvažované zdroje
  - nedefinovaná instrukce (RISC V – neznámý operační kód – unknown opcode)
  - aritmetická výjimka (dělení nulou, přetečení – na RISC-V neexistuje pro celočíselnou jednotku, pro plovoucí řádovou čárku ano)
  - systémové volání (vyvolává instrukce systémového volání – syscall)
  - data nejsou dostupná nebo došlo k chybě čtení/zápisu
  - chyba v adrese, stránka označená jako neplatná, chráněný rozsah atd.
  - nedovolená instrukce pro daný režim procesoru (uživatelský, systémový, strojový)
  - byla zjištěna chyba sběrnice (parita, kontroly a opravy chyb – ECC, překročen čas na potvrzení)
- Asynchronní/externí výjimky (přerušení)
  - nejsou přímým důsledkem vykonání dané aktuální instrukce
  - většinou nelze určit na instrukci, hodinový cyklus, přesně, kdy k události dojde

Asynchronní/externí výjimky (přerušení)

- maskovatelné, lze zakázat ve stavovém/řídicím slovu procesoru (processor status word – PSW)
  - buď všechna nebo do určité priority zdroje
  - periferní zařízení, čítače, časovače
  - meziprocesorová komunikace, notifikace dalších jader, že mají převzít úlohu a další
  - většinou i na vlastním zdroji nebo řadiči povolení a volitelně i nastavení priority
- nemaskovatelné
  - poruchy hardware, dohledové obvody (Watch Dog)

Kroky při zpracování výjimek nebo přerušení

- Výjimka se přijímá/zpracovává obvykle bezpodmínečně, externí přerušení pouze tehdy, není-li maskováno nebo není-li maskováno

1. Vektor stavu procesoru je uložen včetně čítače instrukcí (PC)
   - na systémový zásobník a nebo do specializovaných registrů procesoru (ELR, EPC)
   - kolik je uloženo již v první fázi hardwarem a kolik později obslužnou rutinou se mezi architekturami liší
   - při přerušení běhu v uživatelském režimu je režim přepnutý na systémový nebo strojový (machine)
   - obvykle další přerušení nebo nižší a rovné priority jsou zakázané
2. Čítač instrukcí je přednastaven na počáteční adresu obslužné rutiny
   - buď jedné pro všechny zdroje, případně podle typu výjimky nebo i jednotlivého čísla zdroje přerušení
3. Provede se servisní rutina začínající na této adrese
   - ta ukládá obvykle stav ostatních ohrožených registrů na zásobník, komunikuje s periferiemi, načte chybějící stránku, informuje o neobnovitelné chybě úlohy nebo celého systému atd.

roky při zpracování výjimek nebo přerušení – ukončení obsluhy

5. Spadá-li důvod mezi obnovitelné zdroje (recoverable) – obnoví rutina hodnoty registrů do stavu před spuštěním
6. Rutina je dokončena speciální instrukcí návratu výjimky
   - tím doje přepnutí procesoru do předchozího stavu včetně čítače instrukcí a umožní tak pokračování přerušeného kódu

- Dále již pokračuje buď zopakování instrukce, která výjimku způsobila, nebo se pokračuje od další instrukce, případ externího přerušení, systémového volání nebo emulace instrukce, která není přímo podporovaná

Zdroje výjimek na architektuře RISC-V

- Výjimky způsobené poruchou hardwaru:
  - kontrola stroje (Machine Check): Procesor detekuje vnitřní nekonzistenci
  - chyba sběrnice (Bus Error): při načtení nebo uložení instrukce nebo načtení instrukce
- Výjimky způsobené některými vnějšími příčinami (pro procesor):
  - znovunastavení (Reset): Signál aktivovaný na příslušném vývodu pouzdra, jádra
  - nemaskovatelné přerušení (NMI): Náběžná hrana signálu NMI na příslušném signálu
- Hardwarová přerušení: Aktivovaná příslušnými signály
  - hardwarová přerušení lze maskovat nastavením příslušných bitů stavového registru
- Výjimky, které se vyskytují v důsledku problémů s vykonáním instrukce

Zdroje výjimek na architektuře RISC-V – pokračování

- Výjimky, které se vyskytují v důsledku problémů s vykonáním instrukce
  - chyba v adrese (Address Error): odkaz na neplatnou oblast paměti, popř. odkaz na adresní prostor jádra z uživatelského režimu
  - neplatná instrukce: Nedefinované pole operačního kódu (nebo privilegované instrukce v uživatelském režimu)
- Výjimky způsobené prováděním k tomu určených instrukcí
  - systémové volání (Syscall): Provedená instrukce **ecall**
  - instrukce pozastavení (Break): Provedená instrukce **ebreak**

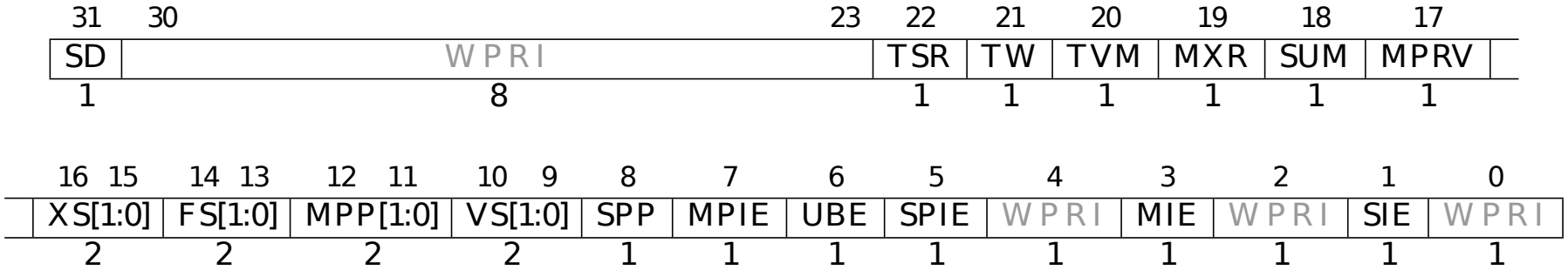
RISC-V – stavové a řídicí registry pro obsluhu výjimek

Jméno registru	Číslo	Význam/využití
mstatus	0x300	Machine status register
misa	0x301	ISA and extensions
mie	0x304	Machine interrupt-enable register
mtvec	0x305	Machine trap-handler base address
mscratch	0x340	Scratch register for machine trap handlers
mepc	0x341	Machine exception program counter
mcause	0x342	Machine trap cause
mtval	0x343	Machine bad address or instruction
mip	0x344	Machine interrupt pending
mtinst	0x34A	Machine trap instruction (transformed)

The RISC-V Instruction Set Manual – Volume II: Privileged Architecture
`https://riscv.org/technical/specifications/`

RISC-V – stavový registr v 32-bitové variantě

Machine Status Register – **mstatus** (RV32)



Pole	Bit(y)	Význam/využití
SIE	1	Supervisor global interrupt enable
MIE	3	Machine global interrupt enable (for handler atomicity)
SPIE	5	SIE before trapping to system mode
MPIE	7	MIE before trapping to machine mode
SPP	8	Priority mode before trapping to system mode
VS	10:9	Inform if floating point save is needed
MPP	12:11	Priority before trapping into machine mode

RISC-V – registr příčiny výjimky/přerušení

Machine Cause Register (mcause) – **mstatus**

Registr informuje obslužnou rutinu o zdroji výjimky, přerušení.
Pokud je nastavený nejvýznamnější bit (RV32 bit 31, RV64 bit 63), pak je zdroj asynchronní výjimka/periferní/externí přerušení. Kód výjimky odpovídá zdroji a odpovídající pozici bitu povolení a nevyřízeného zdroje v registrech **mie** a **mip**. **mepc** ukazuje na přerušenou (první nezpracovanou) instrukci. Návrat jednoduše instrukcí **mret** bude pokračovat v provádění přerušeného programu od další instrukce. Když je nejvýznamnější bit nulový, jedná se o synchronní zdroj výjimky. **mepc** ukazuje na instrukci která jí způsobila. Pokud je příčina (např. výpadek stránky) vyřešena, instrukci bude znova spuštěná jednoduše po návratu **mret**. Pokud je důvodem systémové volání (instrukce ecall), **ebreak** popř. neplatná instrukce, která je emulována obsluhou, pak je nutné **mepc** před návratem **mret** posunout za instrukci, která výjimku způsobila.

RISC-V – kódování příčin synchronních výjimek

Vnější (IRQ)	Číslo	Důvod/zdroj
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint exception
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	15	Store/AMO page fault
0	24...31	Designated for custom use

RISC-V – kódování příčin asynchronních přerušení

Vnější (IRQ)	Číslo	Důvod/zdroj
1	1	Supervisor software interrupt
1	3	Machine software interrupt
1	5	Supervisor timer interrupt
1	7	Machine timer interrupt
1	9	Supervisor external interrupt
1	11	Machine external interrupt
1	16	Designated for platform use

Registr **mie** povoluje jednotlivé zdroje přerušení, pozice bitu odpovídá zdroji
Registr **mip** informuje o aktuálně čekajících/nevyřízených zdrojích
mstatus.MIE globální povolení (1) / zakázání (0)
Existuje další sada registrů pro úroveň systému kontrola
sstatus, **sie**, **sip**, **sscratch**, **scause** atd. Jejich struktura je stejná/podobná, ale jsou přístupné z režimu systému (supervisor), kdy ovládání na úrovni stroje (machine) není povoleno

RISC-V – zpracování výjimek a přerušení

Procesor přijme přerušení, výjimku nebo **ecall** operační znak (ten v režimu uživatele, systému, stroje)

- mepc <= pc and switches to machine privilege mode
- mstatus.MPP and mstatus.MPV set to preceding privilege mode
- mcause <= exception code, mcause[XLEN-1] <= 1 if interrupt
- mstatus.MPIE <= mstatus.MIE, mstatus.MIE <= 0
- PC <= mtvec (for vectored mode and interrupts BASE+4×cause)

Počátek, prolog, obslužné rutiny je zodpovědný za zjištění příčiny `csrr rd, mcause`

Stav procesoru je sledovaný a řízený přes kontrolní a stavové registry (control an status registers – CSR)

`csrrw rd, csr, rs1`		`csrrwi rd, csr, uimm5`
`csrr(s/c)(i) rd, csr, rs1`		`csrr(s/c)(i) rd, csr, uimm5`
`csrr rd, csr`	odpovídá	`csrrs rd, csr, x0`
`csrw csr, rs`	odpovídá	`csrrw x0, csr, rs1`

RISC-V – zakončení zpracování výjimek a přerušení

Obslužná rutina informuje instrukcí **mret** (ze systémového režimu **sret**) o ukončení obsluhy výjimky/přerušení a procesor provede přepnutí do uloženého, obvykle původního režimu a programu

- priviledge mode from mstatus.MPP and mstatus.MPV
- mstatus.MPV <= 0, mstatus.MPP <= 0
- mstatus.MIE <= mstatus.MPIE
- pc <= mepc and continue execution in mode restored from mstatus.MPP

Precizní obsluha výjimek

Pokud je přerušení/výjimka úspěšně zpracována (tj. chybí stránka byla zaměněna atd.) tak provádění pokračuje instrukcí, která ho způsobila (na které byla výjimka přijata). Tok přerušeného kódu se nezmění a z pohledu programu nelze detekovat, že problému došlo (kromě zpoždění/časování a případů, kdy je modifikace stavu zamýšlena/způsobena obsluhou systémových volání)

Poznámka: Přesné zpracování výjimek je nejvíce komplikované zpožděnými zápisy (změna pořadí vykonávání instrukcí na superskalárních architekturách). To vede k detekci synchronních výjimek i o mnoho instrukcí později. Koncept obnovení stavu (rollback) nebo potvrzení „transakcí“ je pak nutný pro zajištění stránkování paměti atd.

Varianty vyhodnocení zdrojů výjimek

- Softwarové vyhodnocení příčiny
  - Obsluha vyhodnocení všech výjimek/přerušení začíná stejnou rutinou na stejné adrese – tj. pro RISC-V je *PC* nastavena na hodnotu **mtvec**. Pro systémový režim **stvec**.
  - Rutina čte zdroj ze stavového registru (RISC-V: **mcause** Registrovat)
- Vektorové zpracování výjimek (na RISC-V se většinou nepoužívá)
  - Identifikace zdroje příčiny v příčiny/zdroj/čísla přerušení v hardware procesoru
  - Pole počáteční adres obslužných rutin je vyplněno na pevné nebo přednastavené adrese (VBR – bázový registr tabulky) v hlavní paměti
  - CPU vypočítá index do tabulky na základě čísla zdroje
  - CPU načte slovo z dané adresy do PC
- Nevektorové zpracování výjimek s více rutinami/počátkem adresy přiřazené třídám výjimek a nebo prioritám přerušení

Na RISC-V buď čistě softwarově, rutina od **mtvec** nebo když **mtvec**.0=1, tak **mtvec** ukazuje na tabulku, pak na pozici nula odkaz na obsluhu (vnějších) přerušení, a dále položky pro jednotlivé synchronní výjimky.

Příklad obsluhy přerušení – počáteční nastavení

```
_start:
    addi    a0, zero, 0x101
    la      t0, skip
    csrrw   zero, mepc, t0
    mret    # test exception ret
    addi    a0, zero, 0x105
    addi    a0, zero, 0x106
skip:
    addi    a0, zero, 0x107
    csrrs   t0, mepc, zero
    ebreak
    la      t0, handle_exception
    csrrw zero, mtvec, t0
    la      t0, task_control_block
    csrrw zero, mscratch, t0
    csrrsi zero,mstatus,8 # MIE=1
    addi t0, zero, 16 # UART RX
    addi t1, zero, 1
    sll t1, t1, t0 # bit mask
    csrrs zero, mie, t1
    li    a0, SERIAL_PORT_BASE
    li    t0, SERP_RX_ST_REG_IE_m
    sw    t0, SERP_RX_ST_REG_o(a0)
    # Background task
    addi t0, zero, 0x0001
    li    a0, SPILED_REG_BASE
Loop:
    csrrs t1, mepc, zero # check
    sw    t1,SPILED_REG_LED_LINE(a0)
    srl t2, t0, 31
    sll t0, t0, 1
    or    t0, t0, t2
    lw    t2, ..._KNOBS_8BIT(a0)
    sw    t2, ..._LED_RGB1(a0)
    xori t2, t2, -1
    sw    t2, ..._LED_RGB2(a0)
    beq zero, zero, loop
```

Příklad obsluhy přerušení – obslužná rutina

```
handle_exception:
    csrrw   tp, mscratch, tp       # store previous and take system tp
    sw      sp, TCB_SP(tp)         # store stack pointer
    sw      ra, TCB_RA(tp)         # store return address
    sw      t0, TCB_T0(tp)         # store rest of clobberable regs
    sw      a0, TCB_A0(tp)
    ...
    csrr    t0, mcause             # is it Rx interrupt?
    blt     t0, zero, handle_irq   # branch to interrupts processing
    ...
    # handle synchronous exception
ret_from_exception:
    lw      sp, TCB_SP(tp)         # restore stack pointer
    lw      ra, TCB_RA(tp)         # restore return address
    lw      t0, TCB_T0(tp)         # restore rest of clobberable regs
    lw      a0, TCB_A0(tp)
    ...
    csrrw   tp, mscratch, tp       # Swap back TCB to mscratch
    mret                           # Return from exception pc <= mepc
```

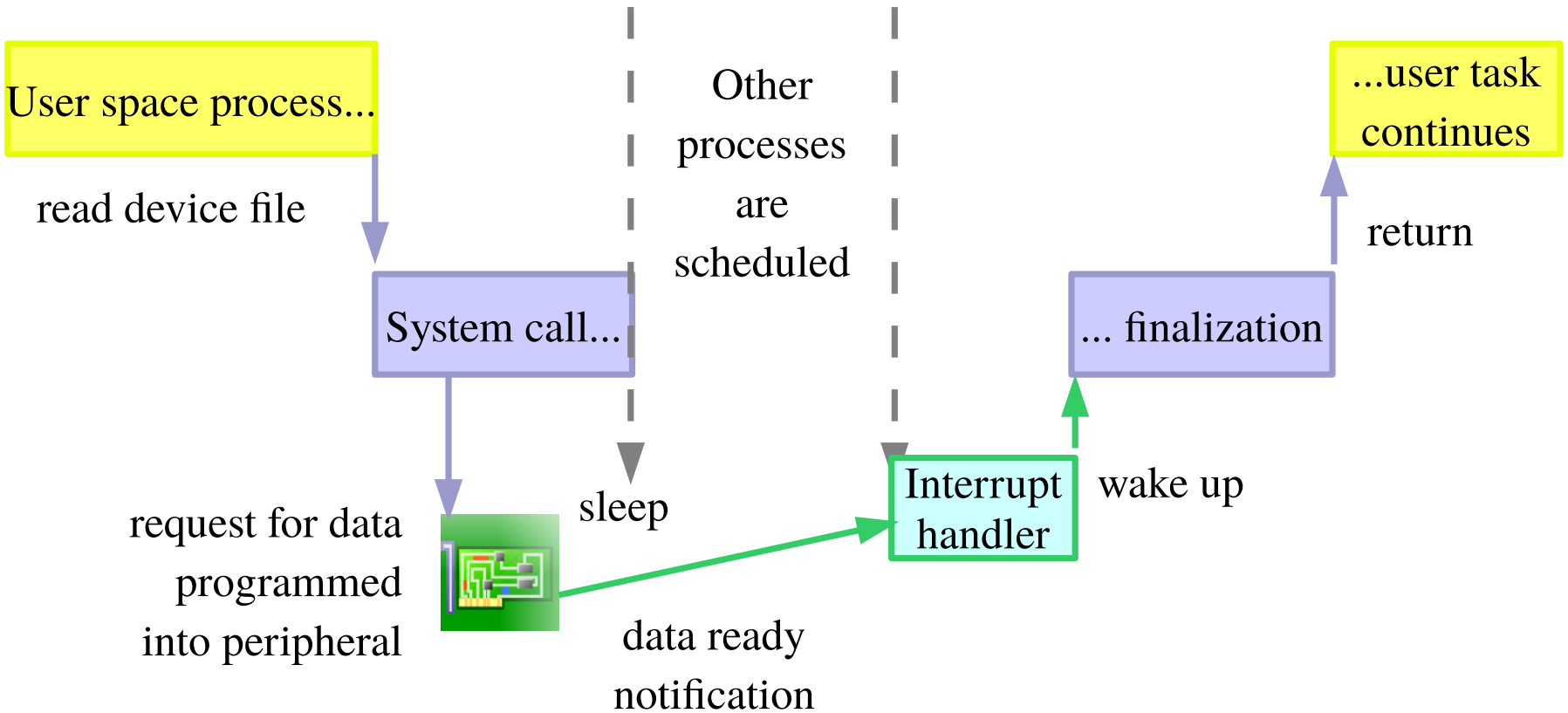
Příklad obsluhy přerušení – obsluha sériového portu

```
handle_irq:                             # t0 mcause
    slli    t0, t0, 2                   # shift out sign, left sources * 4
    # the t0 would be used to point into irq handlers table
    # check only for UART RX interrupt for simplicity 8
    addi    a0, zero, 16 * 4            # UART RX is the first platform irq
    beq     t0, a0, handle_uart_rx_irq # it is UART RX
    # mask out unknown sources
    srli    t0, t0, 2                   # make t0 back simple source index
    addi    a0, zero, 1
    sll     a0, a0, t0                  # generate bit mask for source
    csrrc   zero, mie, a0               # mie = mie & ~a0
    j       ret_from_exception
handle_uart_rx_irq:
    li      a0, SERIAL_PORT_BASE    # Setup base of UART
    lw      t0, SERP_RX_DATA_REG_o(a0) # Read received character
    sw      t0, SERP_TX_DATA_REG_o(a0) # echo it back to terminal
    j       ret_from_exception
```

https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/uart-echo-irq/uart-echo-irq.S
QtRVSim simulátor v RISC-V zatím podporu nemá, QtMips, v MIPS přerušení podporuje.

Přerušení k obsluze V/V v operačním systému

Když aplikace čeká na příchozí data nebo na prostor na odeslání dalších, tak je úloha pozastavena/čeká (procesor může obsluhovat jinou úlohu). Periferie po přijetí dat informuje procesor přerušením a ten dokončí přenos a uvolní původní úlohu pokračování



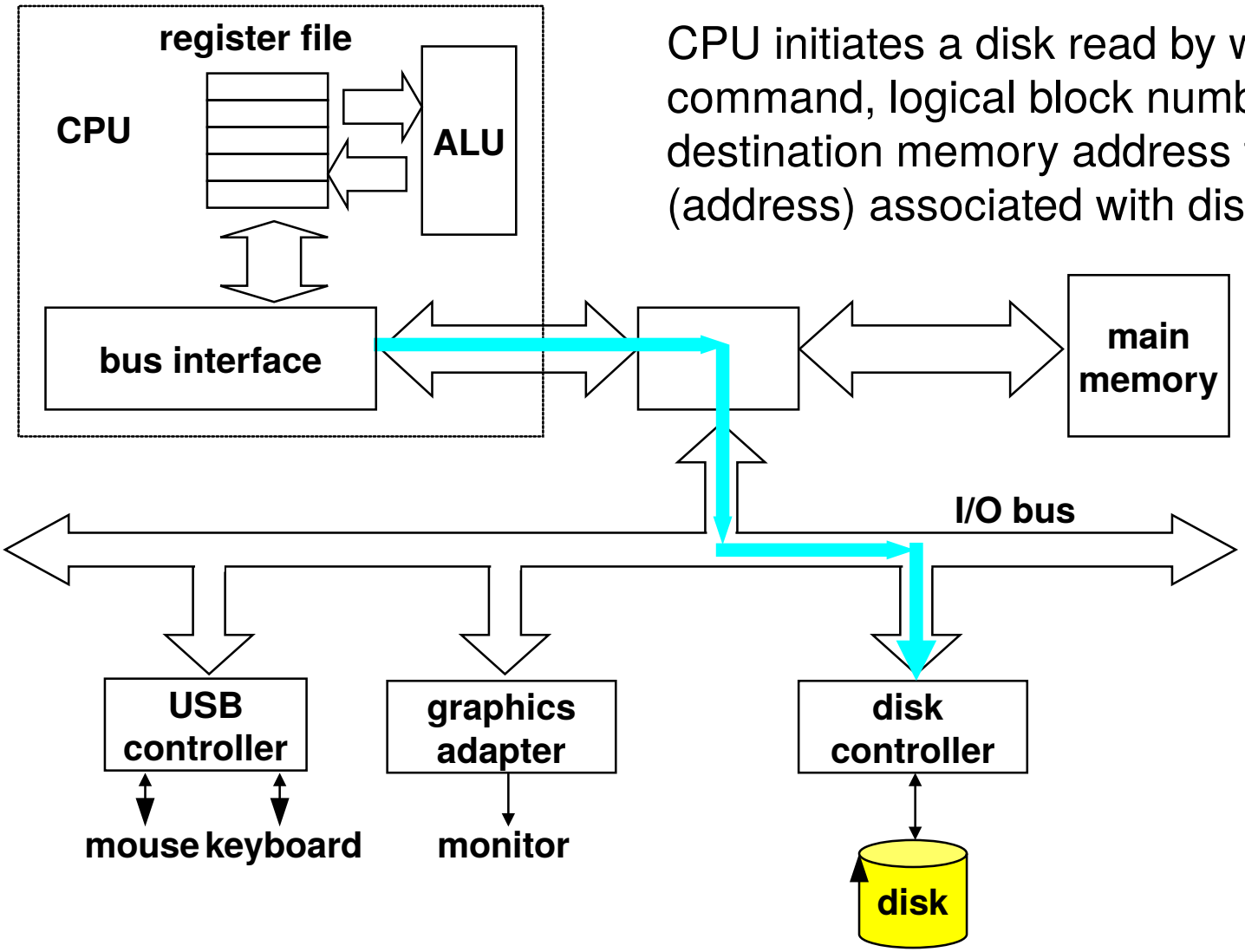
Zdroj: bootlin: Kernel, drivers and embedded Linux development https://bootlin.com/

Obsah

Přímý přístup do paměti z periferií

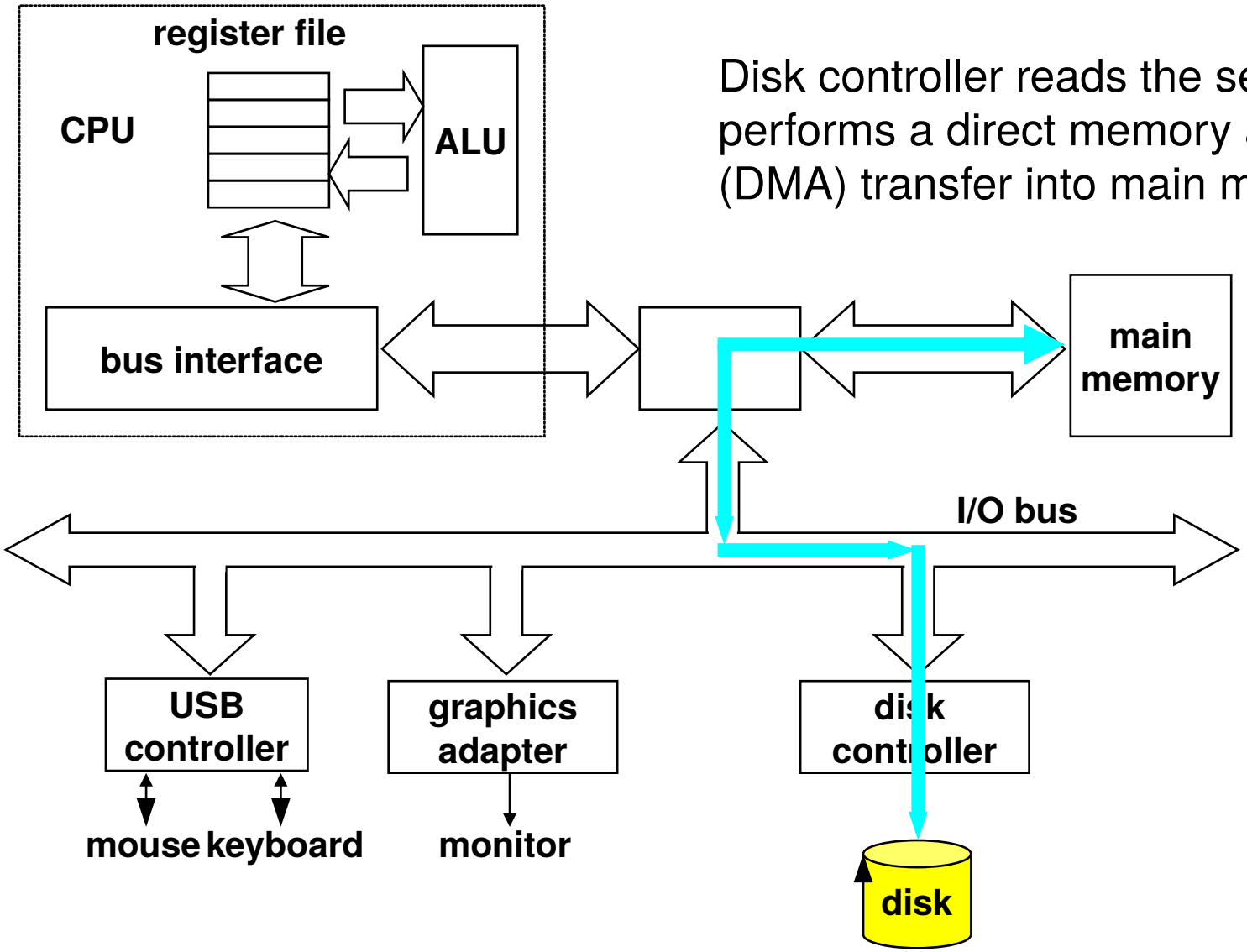
- Procesor nastaví periferii a adresu do paměti odkud/kam mají být data přenesena
- Poté je spuštěný přenos a buď přímo periferie provádí přístupy do hlavní paměti a nebo přes požadavek žádá obecný řadič přímých přenosů (DMA controller)
- Klasických sdílených sběrnic je nutné zajistit, že pro každý datový přenos, cyklus sběrnice dojde uvolnění řízení sběrnice centrálním prvkem, procesorem a volbu adresy a řízení přenosu dat provádí periferie případně obecný řadič
- Po přenesení nastaveného počtu bytů dojde k informaci procesoru o dokončení přenosu nebo o případané chybě mechanizmem vyvolání přerušení

Přímý přístup do paměti – čtení z disku, krok 1



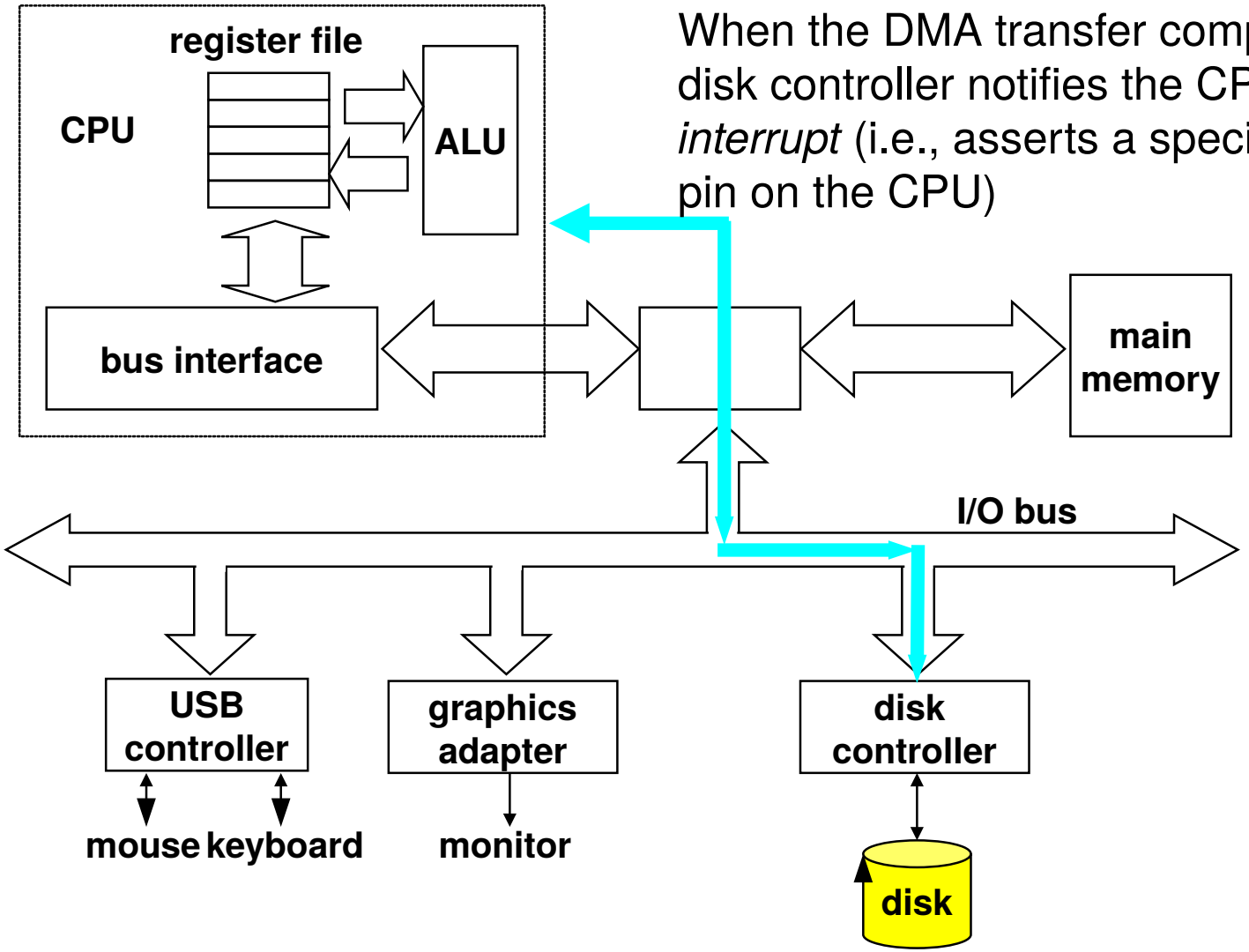
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Přímý přístup do paměti – čtení z disku, krok 2



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Přímý přístup do paměti – čtení z disku, krok 3



When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special “interrupt” pin on the CPU)

Obsah

Skryté paměti, více procesorů a nebo přístupy z periferií

- Při přístupu k paměti mimo dané procesorové jádro je nutné počítat s tím, že data se mohou nacházet ve skrytých pamětech (cache) na různých úrovních paměťové hierarchie
- Přístup bez zneplatnění nebo obnovení povede k porušen podmínek paměťové koherence
- O tu se pro většinu víceprocesorových/jádrových systémů minimálně mezi vlákny stará hardware procesoru
- Koherence při přenosech z/na periferie je na různých architekturách a jejich implementacích řešena různě. Pro oblasti určené pro příjem a vyslání dat by měl operační systém před odstartováním přenosu na periferii zavolat funkce, které zaručí synchronizaci nebo zneplatnění oblasti ve skrytých pamětech tak, jak je na dané architektuře potřeba. Například na architektuře x86 jsou většinou tyto funkce prázdné nebo obsahují jen bariéru pro synchronizaci jádra procesoru. Na ARM je například nutné vyslat instrukce pro synchronizaci s krokem délky řádky vyrovnávací paměti.

Vykonávání instrukcí mimo pořadí a paměťové bariéry

- Budeme předpokládat, že koherence jednotlivých lokací v paměti je vyřešená, ať v hardware nebo pro periferie softwarově
- Pokud však procesor vykonává instrukce mimo pořadí, tak i přístupy do paměti mohou pořadí měnit
- Může se stát, že například instrukce pro spuštění přenosu zapíše data dříve, než je nastavená cílová adresa
- Stejně tak může vláknu na druhém procesoru potvrdit zápisem do paměti, že jsou na jiné adrese data připravená, ale instrukce zápisů nejsou dokončené
- I kompilátor může libovolně přeorganizovat pořadí přístupů nebo opakované vynechat

Vykonávání instrukcí mimo pořadí a paměťové bariéry

- Je tedy nutné definovat sekvenční body přes které se žádné nebo jen určité přístupy v čase dopředu a dále nepřesouvají. K tomu slouží pro každý jazyk, běhové prostředí definované konstrukce a na úrovni procesoru pak specializované instrukce. Na MIPS například **sync**, na RISC-V **fence** na x86 přetížené historické instrukce. Pro x86 vyvolání bariéry z GCC

```
asm volatile("lock; addl $0,-4(%%esp)" ::: "memory", "cc")
```

Na novějších verzích v rámci SSE2 i přímo `mfence`, `lfence` a `sfence`. Další základ pro sychronizaci je operace porovnat a prohodit (Anglicky: Compare and Swap). Na x86

```
__asm__ volatile ("xchgl %0,%1" :"=r" (x) :"m" (y), "0" (x) :"memory");
```