

Procesory

Petr Jedlička, Cyril Klimeš, Tomáš Pitner

Předmluva

HTML · Slides · PDF | Přehled předmětu

Cíle

- Seznámit se s rolí **procesoru** v architektuře počítače.
- Pochopit, jak CPU vykonává program pomocí **instrukčního cyklu**.
- Poznat hlavní části CPU: **řadič**, **ALU/FPU**, **registry** a **cache**.
- Rozlišit **ISA**, přístupy **RISC** a **CISC** a současné architektury **x86-64**, **Arm/AArch64** a **RISC-V**.
- Vysvětlit hlavní techniky zvyšování výkonu a energetické efektivity moderních procesorů.

🔑 **Procesor** je řídicí a výpočetní jednotka počítače. Vykonává **instrukce programu** uloženého v paměti, pracuje nad daty a koordinuje komunikaci s dalšími částmi systému. U současných CPU je důležitá nejen frekvence, ale hlavně **architektura jádra**, **cache**, **počet jader**, **instrukční sada**, práce s pamětí a **spotřeba energie**.

Základní model počítače

Procesor v počítači

- Počítač vykonává **program** nad **daty**.
- Program i data musí být někam uloženy, typicky do **paměti**.
- Procesor je část, která instrukce **načítá**, **dekóduje** a **provádí**.

Co ještě počítač potřebuje

Paměť	uchovává program a data
Vstupy	dodávají informace dovnitř systému
Výstupy	předávají výsledky ven
Sběrnice a řadiče	propojují CPU s pamětí a periferiemi

Uložený program

- Moderní počítač je založen na myšlence **uloženého programu**.
- Program lze změnit pouhou **změnou obsahu paměti**, přeprogramováním.
- Tím se univerzální počítač liší od jednoúčelového automatu.

Program napevno vs. uložený program

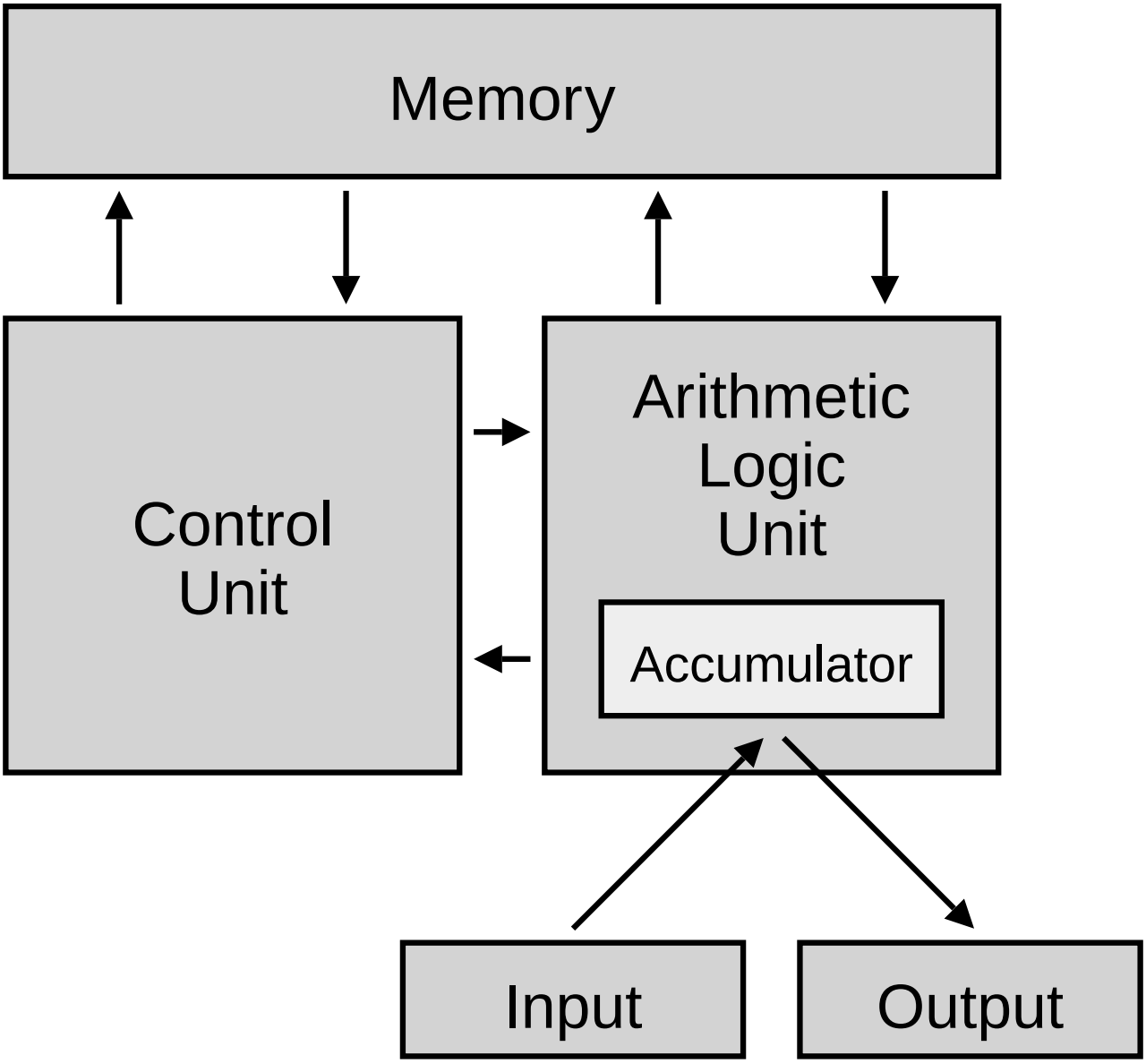


Obr. 1. Orchestrion: skladba, tj. mechanický „program“ je pevně zabudovaný

Von Neumannova architektura

- Program i data sdílejí **jednu hlavní paměť**.
- Procesor používá společné cesty pro přístup k instrukcím i datům.
- Jde o klasický model běžných počítačů.

Schéma von Neumannovy architektury



Obr. 2. Von Neumann Computer Architecture

Výhody a nevýhody von Neumannova modelu

Pro

- Jednodušší celkové uspořádání.
- Flexibilní využití jedné paměti pro program i data.

- Dobře odpovídá obecným počítačům a operačním systémům.

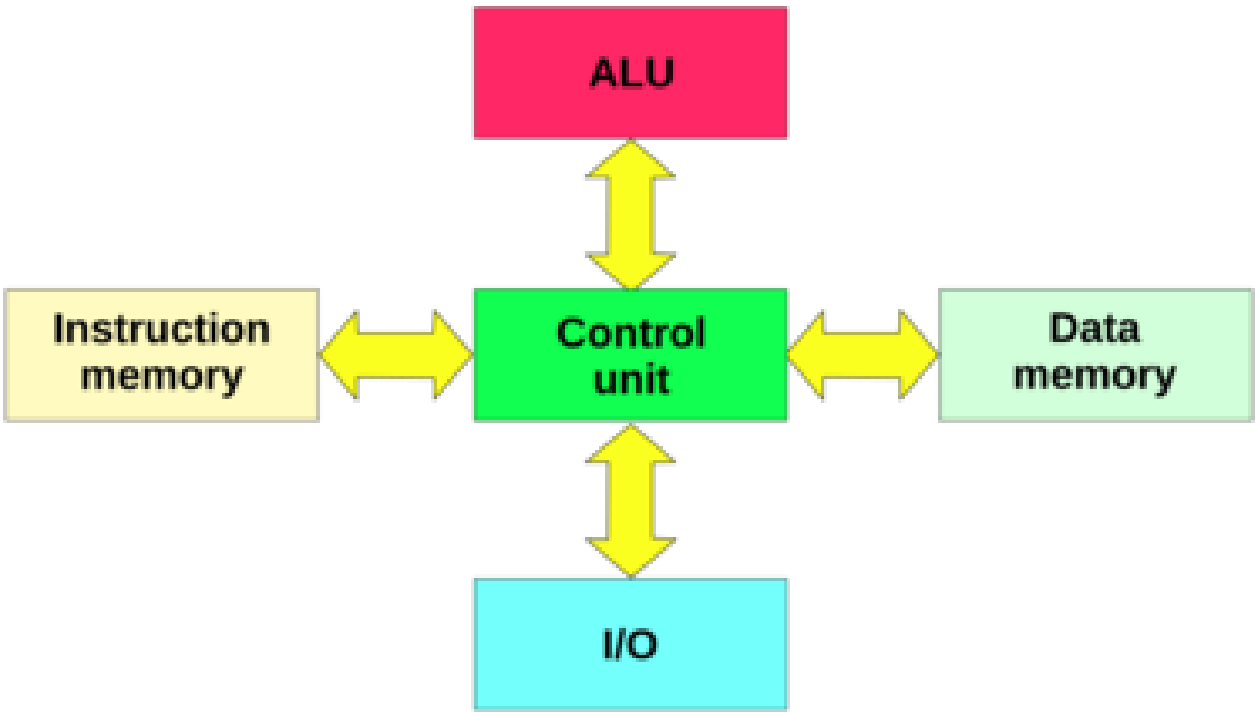
Contra

- Přístup k instrukcím a datům si může vzájemně překážet.
- Výkon může omezovat komunikace mezi CPU a pamětí.
- Proto se používají cache a další optimalizace.

Harvardská architektura

- Instrukce a data jsou **odděleny**.
- Mohou mít oddělené paměti nebo alespoň oddělené cesty.
- Často se používá u mikrokontrolérů a jednodušších vestavěných systémů.

Schéma Harvardské architektury



Obr. 3. Harvard Computer Architecture

Výhody Harvardského modelu

- **Současný přístup** k instrukcím i datům → rychlost.
- Snadnější **oddělení programu** od zapisovatelných dat → bezpečnost.
- Dobře se hodí pro specializovaná a úsporná zařízení → např. stačí malá paměť dat.

Dnešní praxe: hybrid

- Většina dnešních CPU *není* čistě von Neumann ani čistě Harvard.
- Časté je společné adresování paměti, ale oddělené **instrukční** a **datové cache**.
- Mluví se proto i o **modifikované Harvardské architektuře**.

Stavba a činnost CPU



Skutečná vnitřní architektura je dnes extrémně složitá, u CPU jde o obvody s miliard hradel a stovkami funkčních jednotek. U moderních CPU se často detailní vnitřní architekturu ani nedozvíme, je předmětem duševního vlastnictví, licencovaného designu.

Co je uvnitř CPU

- **Řadič** rozhoduje, co se má stát v dalším kroku.
- **ALU** provádí celočíselné a logické operace.
- **FPU** zpracovává operace s čísly v plovoucí řádové čárce.
- **Registry** drží právě používané hodnoty.
- **Cache** zrychluje práci s pamětí.

Řadič

- Řadič **načítá** instrukce, **dekóduje** je a vytváří **řídicí signály**.
- Koordinuje **přesuny dat** mezi registry, cache, ALU a pamětí.
- Aktivuje další jednotky, např. výpočetní (ALU).
- Bez řadiče by jednotlivé části CPU nepracovaly ve správném pořadí.

ALU a FPU

Aritmeticko-logická jednotka

- provádí sčítání, odčítání, porovnání, posuny a logické operace;
- pracuje hlavně s **celými čísly** a bity;
- je přítomna v každém obecném procesoru i ve velmi jednoduchých.

Floating Point Unit

- provádí výpočty s čísly v plovoucí řádové čárce, kterými se obvykle aproximují **reálné** hodnoty;
- je důležitá pro vědecké výpočty, grafiku, simulace i strojové učení;
- v moderních CPU bývá běžnou součástí jádra.

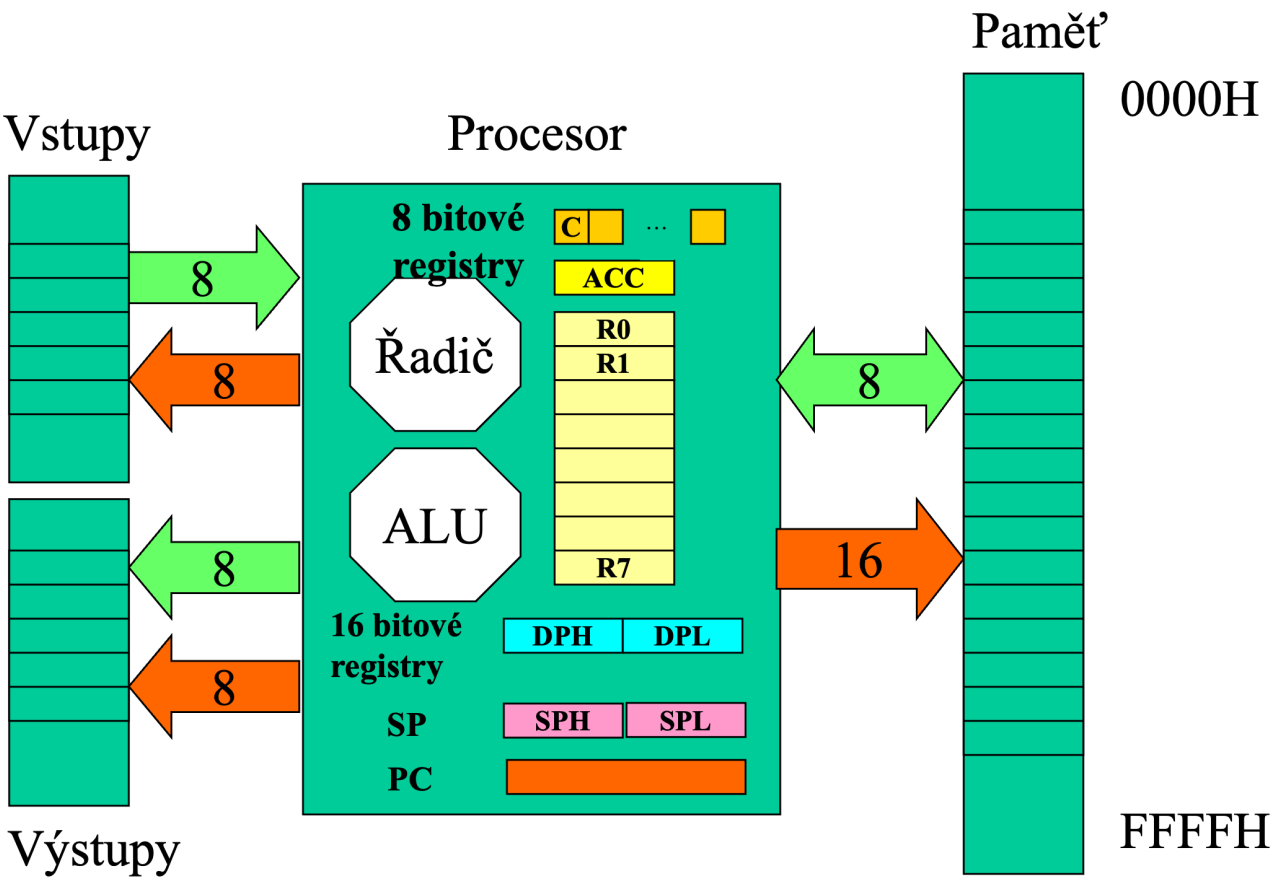
Registry

- Registry jsou nejrychlejší pamětí přímo uvnitř procesoru.
- Nejjednoduší historická CPU měla jen malé jednotky registrů
  - provozní registry jako je čítač instrukcí (instruction pointer)
  - obvykle akumulátor

- indexové (adresové) registry

- Ukládají operandy, adresy, mezivýsledky a stavové informace.
- Přístup do registru je výrazně rychlejší než přístup do operační paměti.

Programátorský model jednoduchého 8-bit CPU



Typické druhy registrů

Obecné registry	obsahují běžná data a adresy (například akumulátor a další)
Programový čítač	(Program Counter/Instruction Pointer) drží adresu další instrukce
Příznakový registr	obsahují stavové bity (příznaky, flags) signalizující výsledek provedení operace
Speciální registry	například pro správu paměti nebo řízení procesoru

Příznakový registr

Popis

- Obsahuje bity, které popisují výsledek předchozí operace nebo obecně operací (některé operace příznaky nemění).

Typické příznaky

Zero

(nulovost) výsledek je nula,

Carry

(přenos) nastal přenos,

Sign

(znaménko) výsledek je záporný,

Overflow

(přetečení) výsledek se nevešel do daného rozsahu.

Hodiny a časování

- Běžné CPU je *převážně* **synchronní systém** řízený jedním nebo více hodinovými signály.
- **Hodiny** určují okamžiky, kdy se mění stav sekvenčních obvodů v příslušné hodinové doméně → změna vnitřního stavu (ale i komunikace zvenku/ven) je synchronizovaná.
- **Latence** (přibližně doba od započetí vykonávání jedné instrukce až po dostupnost jejího výsledku) může být několik taktů, ale pipeline a superskalární vykonávání umožňují současně zpracovávat více instrukcí.

Příklad časového průběhu zpracování

Frekvence není vše

- Frekvence = Rychlost "tikání" systémových hodin = kolik taktů za sekundu.
- Vyšší frekvence obvykle pomáhá výkonu, ale sama nestačí.
- Dva procesory se stejným taktem mohou mít velmi rozdílný výkon.
- Rozhodujícím faktorem rychlosti jsou zejména potřebný **počet taktů** na provedení instrukce a **počet výpočetních toků**, které procesor souběžně provádí.
- Rozhodují i cache, pipeline, šířka vykonávání, predikce větvení a práce s pamětí.

Instrukce, ISA a mikroarchitektura

Program jako posloupnost instrukcí

- Strojový program je řada **instrukcí**.
- Instrukce určují, jakou operaci CPU provede a s jakými operandy.
- Normální pořadí mohou měnit skoky, volání podprogramů a přerušení.

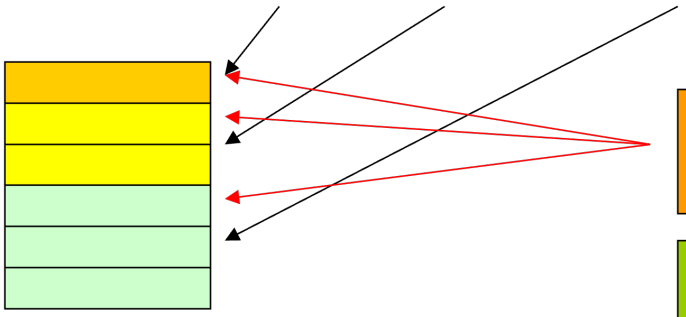
Struktura instrukce

Část	Význam
Operační kód	Určuje, jaká operace se má provést
Operandy	Hodnoty nebo odkazy na data, se kterými CPU pracuje
Adresovací informace	Určuje, odkud se mají data vzít nebo kam uložit výsledek

Instrukce jako binárně zakódovaný příkaz

Skládá se z operačního kódu a operandů/adres

Dělení podle velikosti oblasti paměti, kterou instrukce zabírá:
jednobytové, dvoubajtové, tříbajtové, …



Prvním bytem instrukce je vždy operační kód

Vícebytové instrukce mají v dalších bytech upřesněny své operandy

Operandy instrukcí mohou být zcela určeny operačním kódem, i jednobytové instrukce tedy mohou mít své operandy.

Počet bytů pro operandy je vždy dán operačním kódem

Instrukční cyklus

- Vykonání jedné instrukce probíhá v několika krocích.
- Tyto kroky se opakují znovu a znovu po celou dobu běhu programu.

Fáze instrukčního cyklu

1. načtení instrukce,
2. dekódování,
3. získání operandů,
4. provedení operace,
5. zapsání výsledku.

Načtení instrukce

- Procesor přečte instrukci z adresy uložené v programovém čítači.
- Instrukce bývá načtena z cache, případně z hlavní paměti.
- Poté se čítač posune na další instrukci.



Ideální je, když lze instrukci načíst najednou a nepotřebuje další parametry v následujících slovech v paměti programu.

Dekódování instrukce

- **Určení operace:** řadič zjistí, kterou operaci (případně i operace) instrukce nese.
- **Identifikace operandů:** současně se zjišťuje, které registry nebo paměťové adresy budou potřeba.
- **Mikroinstrukce u CISC:** u složitějších CPU typu CISC (Complex Instruction Set Computer/CPU) se instrukce může převést na jednodušší interní operace tvořené **mikroinstrukcemi**.

Získání operandů

- Operandy mohou být v registrech, v paměti nebo přímo v instrukci jako konstanta.
- Přístup do registrů je rychlý.
- Přístup do paměti je pomalejší, proto je důležitá cache.

Provedení instrukce

- V této fázi proběhne samotná operace.
- Může jít o aritmetiku, logiku, přesun dat, skok nebo přístup do paměti.
- Výsledek se připraví pro zápis do registru nebo paměti.

Zápis výsledku

- Výsledek se uloží tam, kam instrukce určuje.
- Může jít o registr, paměť nebo aktualizaci stavových příznaků.
- Tím se instrukční cyklus uzavírá.

Odkud se berou operandy

Registr	nejrychlejší zdroj dat
Paměť	větší kapacita, ale vyšší latence
Konstanta v instrukci	hodnota je přímo součástí kódu
I/O nebo speciální registr	komunikace s okolím systému

Instrukční sada

- **ISA** (*Instruction Set Architecture*) je rozhraní mezi hardwarem a softwarem.
- Definuje, jak procesor vypadá z pohledu programátora a překladače.
- Dvě různá CPU mohou mít stejnou ISA, ale odlišnou vnitřní realizaci.

Co ISA určuje

- sadu instrukcí,
- formáty instrukcí,
- registry a datové typy,
- adresovací režimy,
- práci s pamětí,
- chování při přerušení a výjimkách.

Proč je ISA důležitá

- Umožňuje kompatibilitu programů mezi různými procesory.
- Překladače a operační systémy se opírají právě o ISA.
- ISA odděluje software od detailů konkrétní mikroarchitektury.

Mikroarchitektura

- Mikroarchitektura je konkrétní vnitřní realizace procesoru.
- Dva procesory se stejnou ISA mohou mít jinou pipeline, cache, šířku vykonávání i spotřebu.
- Výkon proto neurčuje jen ISA, ale i způsob realizace.

RISC a CISC

- Jde o dva klasické přístupy k návrhu instrukční sady a procesoru.
- V dnešní praxi se hranice mezi nimi částečně stírá.

RISC

- Menší a jednodušší instrukce.
- Důraz na rychlé dekódování a efektivní pipeline.
- Typické pro **Arm** a **RISC-V**.

CISC

- Bohatší a složitější instrukční sada.
- Historicky se snažila zkrátit program a zjednodušit ruční programování.
- Typické pro rodinu **x86**.

Dnešní realita

- Moderní procesory často kombinují myšlenky z obou světů.
- Například x86 procesory interně převádějí složitější instrukce na jednodušší mikrooperace.
- Prakticky je důležitější kompatibilita, výkon a spotřeba než čistá nálepka RISC nebo CISC.

Současné instrukční sady

Vymezení

- Tato kapitola se věnuje procesorům určeným pro **počítače**: desktopy, notebooky, pracovní stanice a servery.
- Mikrokontroléry a jednoduché vestavěné systémy používají zpravidla jiné, jednodušší instrukční sady; ty zde neprobíráme.

x86-64

- Dominantní ISA v osobních počítačích, pracovních stanicích a části serverů.
- Navazuje na starší x86, ale dnes je běžná v 64bitové podobě.
- Silnou stránkou je dlouhodobá zpětná kompatibilita.

Silné stránky x86-64

- široká softwarová kompatibilita,
- silná pozice v desktopu a notebooku,
- vyspělý ekosystém nástrojů a operačních systémů.

Arm/AArch64 (ARM64)

- **AArch64** je 64bitový vykonávací stav architektury Arm a používá instrukční sadu **A64**; **ARM64** je běžné označení v operačních systémech a nástrojích.
- AArch64 je součástí aplikačních architektur Armv8-A a Armv9-A.
- Výrazná v mobilech, tabletech, vestavěných systémech a stále více i v noteboocích a serverech.
- Typicky klade důraz na poměr výkonu a spotřeby.

Kde se Arm/AArch64 hodí

- telefony a tablety,
- úsporné notebooky,
- vestavěná zařízení,
- servery zaměřené na efektivitu na watt.

RISC-V

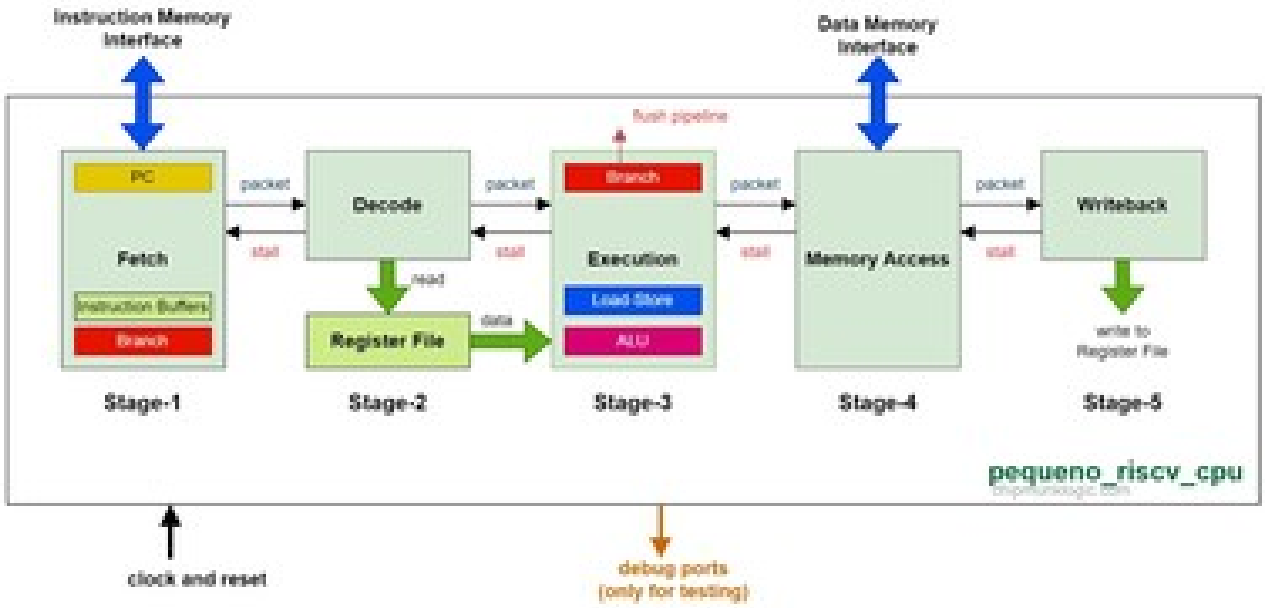
- **Otevřený standard ISA**, který lze používat bez licenčních poplatků.
- Je zajímavá pro výuku, výzkum, vestavěná zařízení i nové komerční návrhy.

- Výhodou je modularita tvořená základní instrukční sadou a standardizovanými rozšířeními.

Proč je RISC-V důležitý

- Otevřená je specifikace ISA,
- konkrétní procesorové implementace mohou být otevřené i proprietární.
- Dobře se hodí pro akademické i průmyslové experimenty.
- Není zatím tak rozšířený v osobních počítačích jako x86-64 nebo Arm/AArch64.

Příklad blokového schématu RISC-V CPU



Obr. 4. PEQUENO: jednoduchý RISC-V procesor

Paměťová hierarchie

- v dřívějších architekturách nebo v dnešních malých kontrolerech se používal jen jeden typ operačních pamětí
- při vyšších frekvencích začíná brzdit CPU
- vytváří se proto hierarchie paměti jako technika zvýšení výkonu
- a optimalizace poměru cena/výkon

Cache

- Cache je malá, ale rychlá paměť mezi CPU a hlavní pamětí.
- Ukládá často používaná data a instrukce.
- Cílem je snížit počet pomalých přístupů do větší RAM.

Princip lokality

- Programy často opakovaně používají stejná data nebo blízké adresy.
- Toho využívá cache.
- Proto může relativně malá cache výrazně pomoci výkonu.

Úrovně cache

L1 velmi malá a velmi rychlá, obvykle přímo u jádra

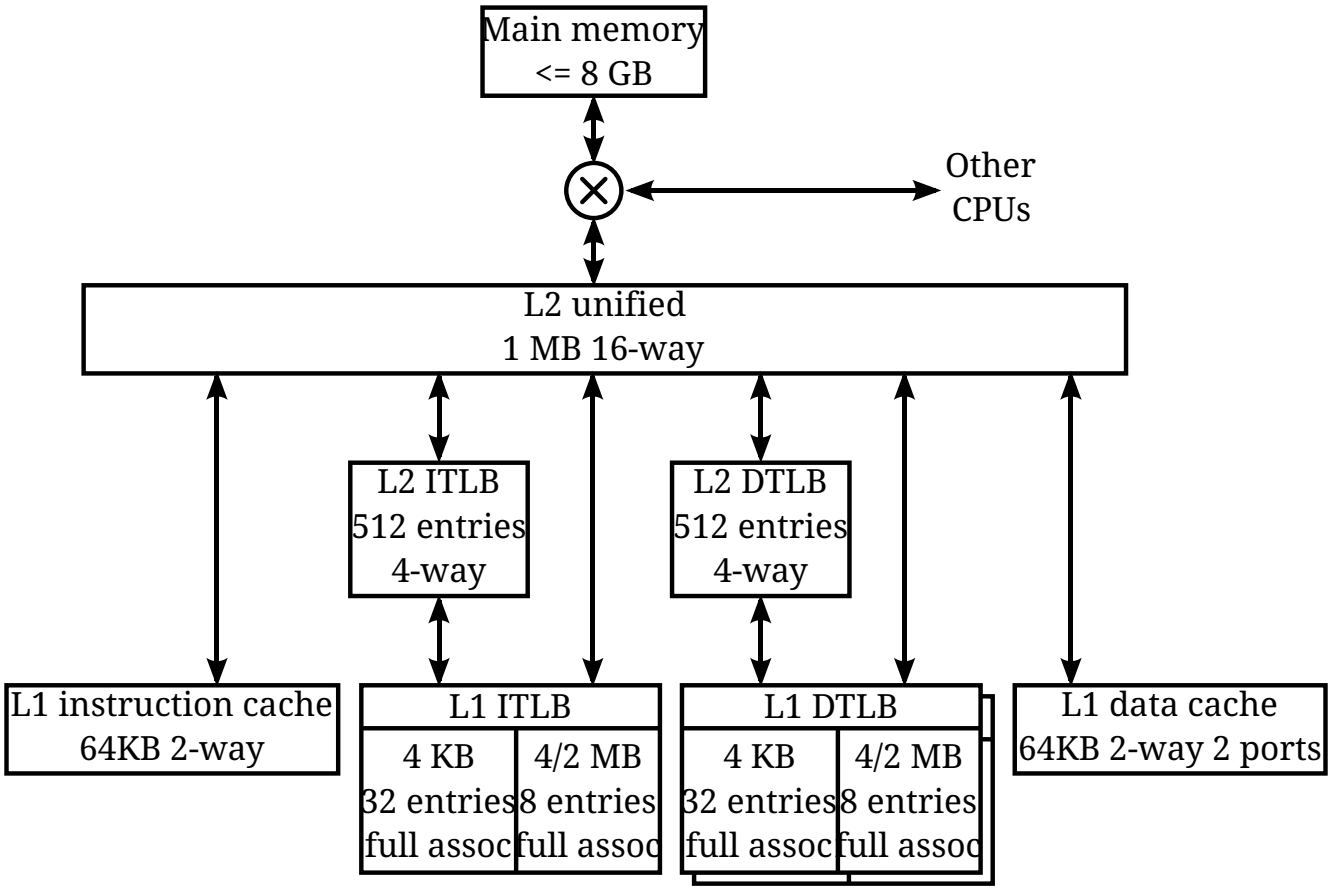
L2 větší, o něco pomalejší

L3 ještě větší, často sdílená mezi více jádry

Instrukční a datová cache

- Mnoho procesorů odděluje **I-cache** a **D-cache**, typicky na úrovni L1.
- Instrukce a data se tak mohou číst paralelně.
- Je to jeden z důvodů, proč moderní CPU připomínají modifikovanou Harvardskou architekturu.

Vizualizace oddělené cache



Obr. 5. Oddělené L1 instrukční a datové cache, sjednocená L2 a hlavní paměť (příklad dle AMD K8)

Techniky zvyšování výkonu jednoho jádra

Pipeline

- **Proudové zpracování**
- Zpracování instrukce se rozdělí do více fází.
- Různé instrukce pak mohou být v různých fázích současně.
- Tím roste propustnost procesoru.

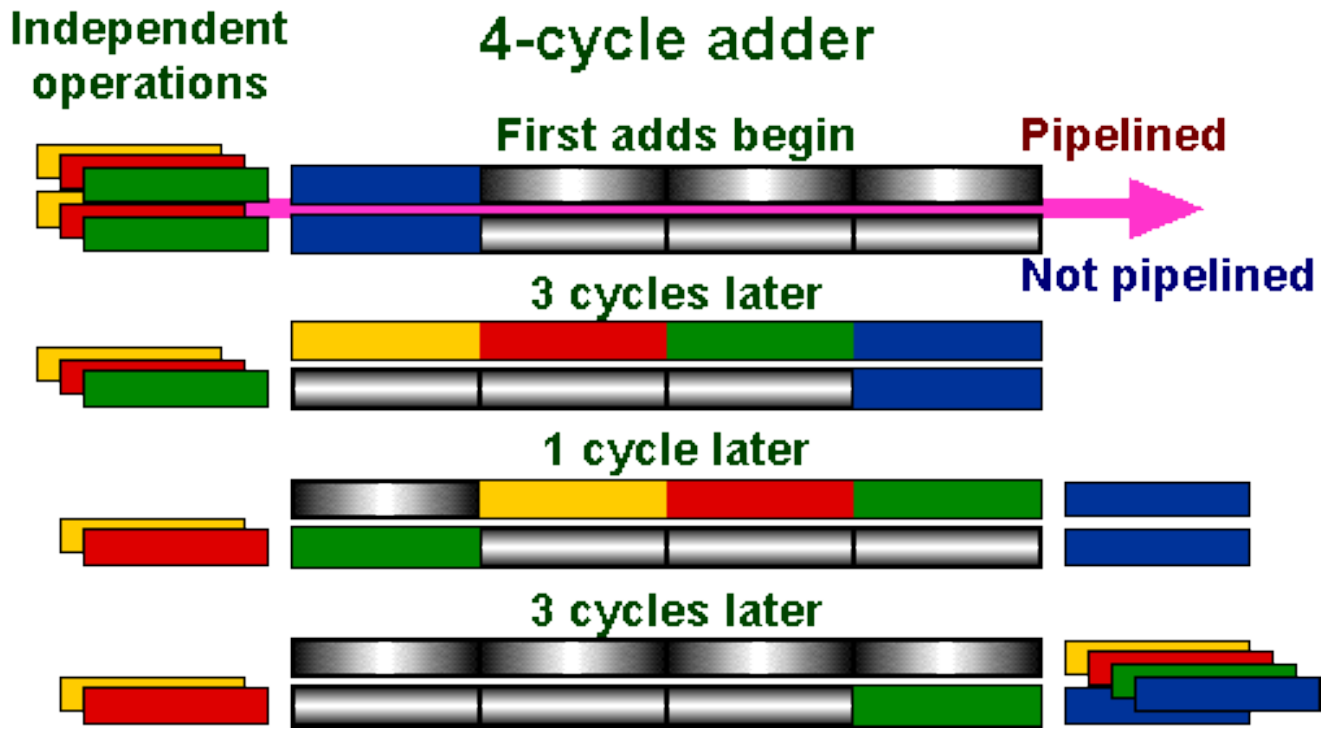
Typické fáze pipeline

1. **fetch** = výběr
2. **decode** = dekódování
3. **execute** = provedení
4. **memory** = přístup do paměti
5. **write-back** = zápis výsledku

Co pipeline přináší

- Lepší využití jednotlivých částí CPU.
- Vyšší počet dokončených instrukcí za jednotku času.
- Zároveň ale přináší komplikace při skocích a závislostech mezi instrukcemi.

Pipeline jako překrytí fází



Problémy pipeline

Datové konflikty

instrukce potřebuje výsledek předchozí instrukce.

Strukturální konflikty

dvě instrukce chtějí stejný prostředek.

Řídicí konflikty

není jisté, kam povede podmíněný skok.

Predikce větvení

- **Branch Prediction**
- Procesor se snaží odhadnout výsledek podmíněného skoku dříve, než jej skutečně dopočítá.
- Když odhad vyjde, pipeline se nemusí zbytečně zastavit.
- Když nevyjde, část práce se zahodí.

Spekulativní vykonávání

- **Speculative Execution**
- CPU může začít provádět instrukce předem podle odhadu dalšího běhu programu.
- Pokud byl odhad správný, ušetří se čas.
- Pokud ne, výsledky se zahodí.

Bezpečnostní poznámka

- Mikroarchitektonické útoky mohou využívat vedlejší efekty spekulativního vykonávání a cache.
- Mezi nejznámější patří třídy zranitelností **Spectre** a **Meltdown**.
- Riziko se omezuje kombinací úprav hardwaru, firmwaru, operačního systému a překladačů; některá opatření mohou ovlivnit výkon.

Vykonávání mimo pořadí

- **Out-of-order Execution**
- Instrukce se nemusí interně provádět přesně v pořadí, v jakém jsou v programu.
- CPU hledá operace, které mohou běžet dříve bez porušení správnosti.
- Tím se lépe využijí výpočetní jednotky.

Přejmenování registrů

- **Register Renaming**
- Interně lze odstranit některé falešné závislosti mezi instrukcemi.
- Procesor používá více fyzických registrů, než kolik jich vidí programátor.
- To zlepšuje paralelismus uvnitř jádra.

Superskalární procesory

- **Superscalar Processors**
- Jedno jádro může začít nebo dokončit více instrukcí během jednoho taktu.
- Má k dispozici více paralelních vykonávacích jednotek.
- Dnešní výkonná CPU jsou běžně superskalární.

Paralelismus a akcelerátory

Vícejádrové procesory

- **Multi-core**
- Dnešní procesory obsahují více jader na jednom čipu.
- Každé jádro může zpracovávat vlastní vlákno nebo proces.
- Výkon pak roste hlavně tam, kde software umí využít paralelismus.

Sdílení mezi jádry

- Některé prostředky jsou pro jádra samostatné, jiné sdílené.
- Typicky má každé jádro vlastní registry a L1 cache.
- Vyšší úrovně cache nebo paměťová propustnost mohou být sdílené.

SMT a Hyper-Threading

- Jedno fyzické jádro může obsluhovat více vláken.
- Cílem je lépe využít jednotky, které by jinak část času čekaly.
- Přínos závisí na typu zátěže a není stejný jako přidání dalšího fyzického jádra.
- **Hyper-Threading** je obchodní označení technologie SMT společnosti Intel.

SIMD a vektorové instrukce

- Jedna instrukce může provádět stejnou operaci nad více daty najednou.
- To je užitečné pro multimédia, grafiku, šifrování, vědecké výpočty i AI.
- Příklady jsou x86 rozšíření **SSE**, **AVX** a **AVX-512**, technologie Arm **Neon** a **SVE/SVE2** nebo standardní vektorové rozšíření **V** architektury RISC-V.

Specializované akcelerátory

- Vedle klasických CPU se dnes používají i GPU, NPU nebo další akcelerátory.
- CPU zůstává univerzální řídicí jednotkou.
- Specializované úlohy je ale výhodné přesouvat na vhodnější hardware.

Přerušení, ochrana procesů

Přerušení

- Hardwarové přerušení umožňuje CPU asynchronně reagovat na událost z periferie, časovače nebo jiného procesoru.
- Procesor dočasně přeruší běžný kód a spustí obsluhu přerušení.
- Po dokončení se vrátí k původní práci.

K čemu přerušení slouží

- obsluha klávesnice, síťové karty nebo disku,
- periodické časovače,
- signalizace dokončení vstupní nebo výstupní operace,
- meziprocesorová přerušení ve vícejádrových systémech.

Přerušení a výjimky

- **Přerušení** je zpravidla asynchronní vůči právě prováděné instrukci a obvykle přichází z hardwaru.
- **Výjimka** je synchronní událost vyvolaná prováděnou instrukcí, například dělením nulou, výpadkem stránky nebo instrukcí systémového volání.
- Oba mechanismy přesměrují řízení na příslušnou obsluhu, ale liší se zdrojem a okamžikem vzniku.

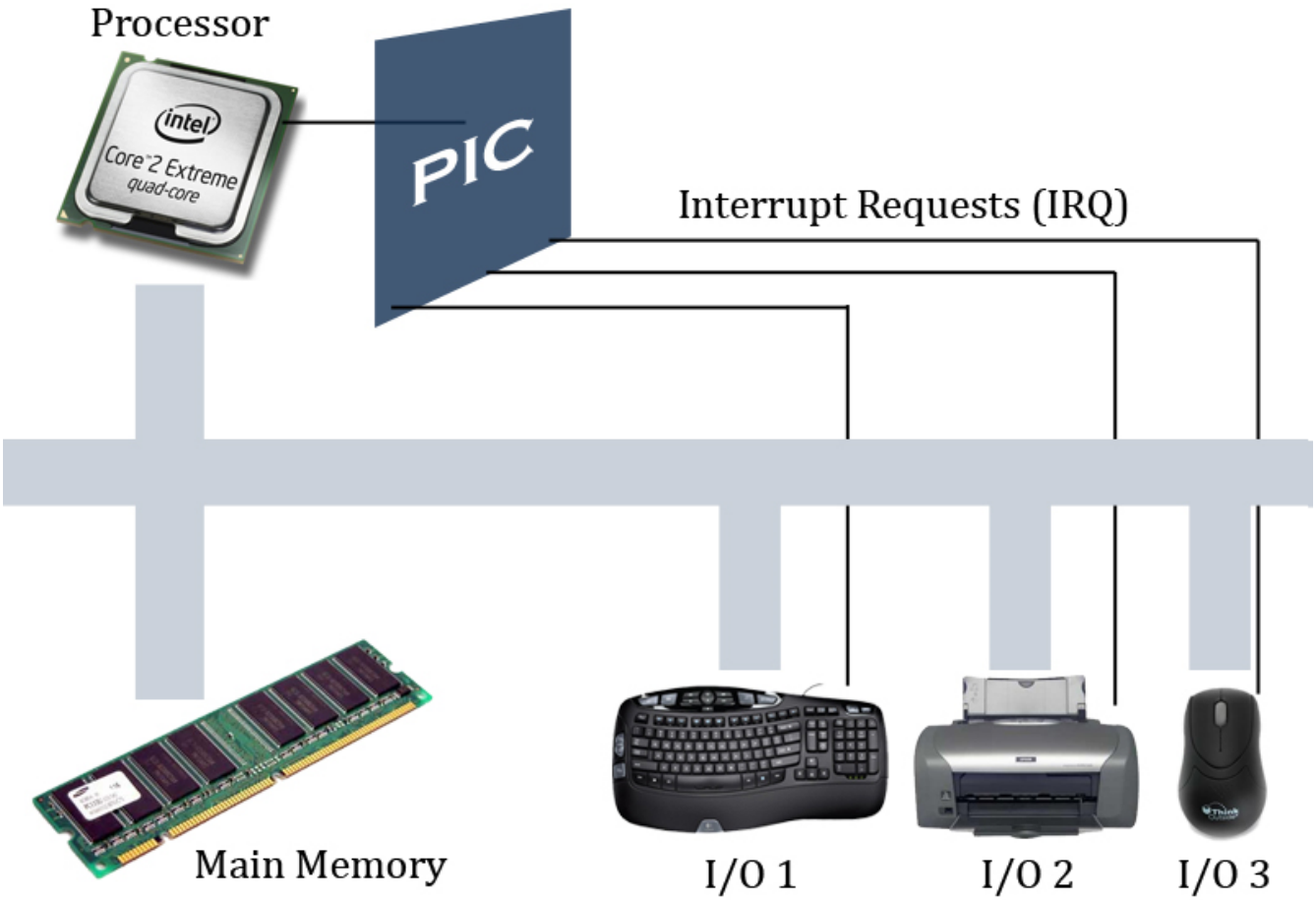
Tabulka vektorů přerušení

16.1. Interrupt Vectors in ATmega328/P

Table 16-1. Reset and Interrupt Vectors in ATmega328/P

Vector No	Program Address(2)	Source	Interrupts definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event

Schéma obsluhy přerušení



Privilegované režimy

- Procesory rozlišují běžný a privilegovaný režim.
- Operační systém potřebuje větší oprávnění než běžná aplikace.
- To je základ ochrany paměti a bezpečného běhu systému.

Virtuální paměť

- Moderní CPU spolupracují s operačním systémem na převodu virtuálních adres na fyzické.
- Každý proces tak zpravidla dostává vlastní virtuální adresový prostor; vybrané oblasti mohou procesy záměrně sdílet.
- Zvyšuje to bezpečnost i flexibilitu správy paměti.

Energetická efektivita procesoru

Paměťový řadič a propojení systému

- Běžné moderní procesory obsahují integrovaný **paměťový řadič**.
- Tím se zkracuje cesta do RAM a zlepšuje výkon.
- Důležitá je také propustnost spojení mezi CPU, pamětí a dalšími zařízeními.

Vývoj propojení CPU se systémem

Starší platforma s odděleným řadičem	Současné obvyklé uspořádání
CPU systémová sběrnice severní můstek RAM a grafika	CPU/SoC obsahuje paměťový řadič a u PC či serverů zpravidla také PCIe řadič; RAM a rychlá zařízení jsou připojena přímo
Severní a jižní můstek zajišťují většinu komunikace s periferiemi	Pomalejší I/O může obsluhovat samostatný čipset nebo integrovaná část SoC
Paměťový řadič leží mimo CPU, což prodlužuje cestu do RAM	Vícečipové a vícesocketové systémy používají vysokorychlostní koherentní propojení

Spotřeba a teplo

- Výkon CPU je omezen i spotřebou a chlazením.
- Vysoký takt bez odpovídajícího chlazení není udržitelný.
- Proto se řeší řízení napětí, frekvence a vypínání nevyužitých částí.

Výkon na watt

- U mobilních a serverových systémů je poměr výkonu a spotřeby klíčový.
- Proto mají moderní procesory sofistikované režimy úspory energie.
- Energetická efektivita je dnes stejně důležitá jako hrubý výkon.

Procesory jako SoC

- V mnoha zařízeních není CPU samostatný čip, ale součást **SoC**.
- SoC integruje CPU a další logiku do jednoho pouzdra nebo čipu.
- To snižuje spotřebu, cenu i nároky na desku zařízení.

Co může SoC obsahovat

- CPU jádra,

- GPU,
- řadiče pamětí,
- síťová a multimediální rozhraní,
- specializované akcelerátory.

Typy procesorů dle použití

Procesory podle použití

Desktop a notebook	univerzální použití, důraz na výkon i efektivitu
Server	výkon, škálování, spolehlivost, virtualizace
Mobilní zařízení	co nejlepší poměr výkonu a spotřeby
Vestavěné systémy	nízká spotřeba, jednoduchost, dlouhá dostupnost

Mikrokontroléry

- Mikrokontrolér je velmi integrovaný procesorový systém pro řízení zařízení.
- Často obsahuje CPU, paměť i periferní rozhraní na jednom čipu.
- Hodí se pro senzory, regulaci, auta, spotřebiče nebo průmyslovou automatizaci.

CPU, GPU a NPU

- **CPU** je univerzální a dobře zvládá různorodý kód.
- **GPU** je vhodné pro masivně paralelní výpočty.
- **NPU** nebo podobný akcelerátor cílí na úlohy strojového učení.

Superpočítače

- Současné superpočítače kombinují mnoho procesorů a akcelerátorů.
- Výkon nevzniká jen silným jedním jádrem, ale rozsáhlým paralelismem.
- Důležitá je i rychlá komunikace mezi uzly a efektivní chlazení.

Jak se CPU hodnotí

- Samotná frekvence nestačí.
- Důležité jsou:
  - výkon na jedno vlákno,
  - výkon ve více vláknech,
  - latence a velikost cache,
  - paměťová propustnost,
  - spotřeba,
  - kompatibilita a cena.

Co je dnes zastaralé měřítko

- Samotný počet MHz nebo GHz už nevystihuje kvalitu procesoru.
- Ani jednoduché srovnání počtu jader bez kontextu nestačí.
- Procesory je nutné posuzovat podle typu úlohy, architektury a energetických limitů.

🏁 Shrnutí

- CPU vykonává instrukce programu uloženého v paměti.
- Klíčové části jsou **řadič**, **ALU/FPU**, **registry** a **cache**.
- Z pohledu software je zásadní **ISA**, z pohledu hardware **mikroarchitektura**.
- Výkon zvyšují **pipeline**, **predikce větvení**, **vykonávání mimo pořadí**, **SIMD**, **SMT** a **více jader**.
- V současnosti jsou nejdůležitějšími obecnými architekturami **x86-64**, **Arm/AArch64** a **RISC-V**.
- V mnoha zařízeních je CPU jen jednou částí širšího **SoC**.

Na další kapitolu: Programátorský model procesoru, assembler

HTML · Slides · PDF | Přehled předmětu