UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED

Informačné technológie

Vypracované otázky na štátnu skúšku

Nitra 2018 Bc. Michal Kohútek

Obsah

1. Architektúra RISC a CISC

Korpus: *Definícia pojmu procesor, rozdiel medzi architektúrami RISC a CISC, príklady spracovávania inštrukcií, využitie architektúry CISC v súčasnosti, výhody a nevýhody malého počtu inštrukcií. Pri uvádzaní rozdielov medzi jednotlivými architektúrami je potrebné, aby študent vedel rozlíšiť od seba nielen architektúru RISC a CISC, ale aby vedel popísať hlavne prečo vznikla nová architektúra RISC a čo ňou je možné dosiahnuť z hľadiska používania tzv. inštrukčných mixov.*

1.1. CPU

Central Processing Unit, alebo procesor, interpretuje, vykonáva inštrukcie a spracúva dáta programu vo forme strojového kódu. V dnešnej dobe je zvyčajne realizovaný vo forme mikroprocesora. Historicky boli CPU implementované ako diskrétne komponenty a viacero malých integrovaných obvodov na jednej či viacerých doskách. Mikroprocesory sú CPU vyrábané zvyčajne na jednom integrovanom obvode. Zmenšenie veľkosti CPU prispieva k rýchlejšiemu prepínaniu, vďaka fyzickým faktorom, ako napríklad zníženej parazitickej kapacitancii. Hoci komplexnosť, veľkosť a výkon CPU sa od roku 1950 výrazne zmenili, základný dizajn a funkčnosť sa zachovali. Skoro všetky bežné CPU sa dajú pomerne presne popísať Von Neumannovou schémou.

Práca CPU Základnou funkciou CPU je vykonávať sekvenciu uložených inštrukcií[1]. Vykonávané inštrukcie bývajú uložené v nejakej forme **počítačovej pamäti**. Skoro všetky CPU nasledujú fetch, decode a execute kroky v ich operácii, ktoré sa spolu nazývajú **inštrukčným cyklom.**. Po vykonaní inštrukcie sa celý proces zopakuje s nasledujúcou inštrukciou označenou inkrementovanou hodnotou v registri **program counter**. Pokiaľ sa vykonala inštrukcia **skoku**[2], program counter sa nastavi na cieľovú inštrukciu a vykonávanie programu prebieha ďalej normálne. Vo viac zložitých CPU sa môže vykonávať inštrukčný cyklus na viacerých inštrukciach súčasne. Niektoré inštrukcie (napríklad *jump*) nemanipulujú dátami, ale zato upravujú program counter. Tým umožnujú praktiky, napr. cykly[3], podmienené vykonávanie inštrukcií[4] a existenciú funkcií.

Fetch Prvý krok zaisťuje získanie inštrukcie z pamäte programu. Adresa inštrukcie je určená hodnotou v **PC** (program counter). Keď je inštrukcia získaná, PC je inkrementovaný dĺžkou inštrukcie tak, aby obsahoval adresu nasledujúcej inštrukcie. Keďže inštrukcie bývajú uložené v relatívne pomalých pamätiach (RAM, swapfile na disku), existuje riziko, že procesor bude na inštrukciu musieť čakať. Tento problém sa v moderných architektúrach rieši použitím cache a pipeline architektúr.

Decode V tomto kroku sa inštrukcia konvertuje na signály, ktoré ovládajú ostatné časti CPU, za použitia **dekódera inštrukcií**[5]. Spôsob akým sa toto deje je definovaný architektúrou inštrukčnej sady[6]. Zvyčajne jedna skupina bitov, zvaná opcode, označuje, ktorá operácia sa má vykonať, kým ostatné polia poskytujú dodatočné informácie dôležité pre operáciu, napr. operandy. Tieto operandy môžu byť konštatné hodnoty, alebo lokácie hodnoty, ktorá môže byť buť register procesoru, alebo adresa pamäte. V niektorých dizajnoch CPU býva dekóder inštrukcií implementovaný ako nemeniteľný obvod. V iných sa na dekódovanie in-

[1] program
[2] jump
[3] for, while loop
[4] if, switch-case
[5] Instruction decoder
[6] ISA, Instruction Set Architecture

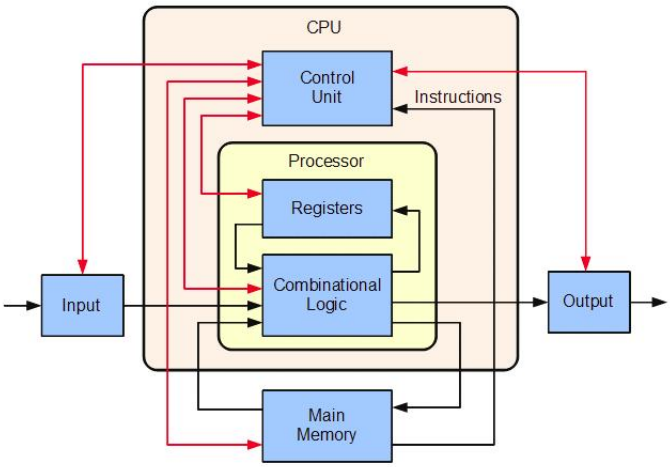
štrukcií používa mikroprogram, ktorý môže byť prepísaný, čím sa dá upraviť spôsob, ako CPU dekóduje inštrukcie. Takto sa napríklad riešia zraniteľnosti, napr. Spectre a Meltdown.

Execute Posledným krokom inštrukčného cyklu sa vykonávajú inštrukcie. V závislosti od architektúry CPU, toto môže byť spravené jednou akciou, alebo ich sekvenciou. Počas každej akcie sa rôzne časti CPU elektricky prepoja, aby mohli vykonať celú, či časť operácie v závislosti od tikotu hodín. Častokrát sa výsledny zapisujú do vnútorného registra CPU pre rýchly prístup z následujúcich inštrukcií. Pokiaľ sa k nim v skorej dobe nebude pristupovať, zapisujú sa do pomalšej, ale lacnejšej hlavnej pamäti. Napríklad ak sa má vykonať inštrukcia sčítania, vstupy do aritmeticko-logickej jednotky[7] sú prepojené s párom operandov, výsledok sa objaví na výstupe, ktorý je pripojený na úložisko (register, alebo pamäť). Keď nastane tik hodín, výsledok sa presunie do úložiska a pokiaľ je výsledok väčší ako veľkosť výstupneho slova ALU, nastaví sa arithmetic overflow flag.

Štruktúra CPU V CPU sú na pevno definované základné operácie, ktoré dokáže vykonávať, tzv. **inštrukčná sada**[8]. Medzi takéto operácie patrí napríklad sčítanie, odčítanie, porovnanie dvoch čísel či skok do inej časti programu. Každá základná operácia je reprezentovaná kombináciou bitov, ktorá sa nazýva **opcode** strojového jazyka. Program v strojovom jazyku je zoskupenie inštrukcií v storjovom jazyku, ktoré CPU vykonáva. Matematické operácie pre každú inštrukciu sa vykonávajú kombinatorickým logickým obvodom, ktorý sa označuje ako Aritmeticko logická jednotka, alebo ALU. Základnými časťami procesora teda sú:

- Aritmeticko-logická jednotka
- Riadiaca jednotka[9]

Viac komplexné mikroprocesory súčasnosti obsahujú aj tzv. **jednotku prideľovania pamäte**, alebo **MMU**[10]. Jednoduché procesory, napr. mikrokontroléry zvyčajne neobsahujú MMU.



Obr. 1.1. Základná schéma CPU. Čierne čiary označujú tok dát, červené tok príkazov, šípky smer.

[7] ALU, Arithmetic Logic Unit
[8] instruction set
[9] Control Unit
[10] Memory Management Unit

ALU Aritmeticko logická jednotka je kombinačný digitálny elektrický obvod, ktorý vykonáva aritmetické a bitové operácie na celočíselných binárnych číslach. Tento koncept navrhol John von Neumann v 1945 v správe o základoch nového počítača EDVAC. Na vstupe do ALU sú dáta, zvané operandy a kód operácie, ktorá má byť vykonaná. Na výstupe je výsledok vykonanej operácie. Vo viacerých dizajnoch má ALU aj stavové vstupy a výstupy, ktoré nesú informáciu o predchádzajúcej, alebo súčasnej operácie medzi ALU a externými stavovými registrami.

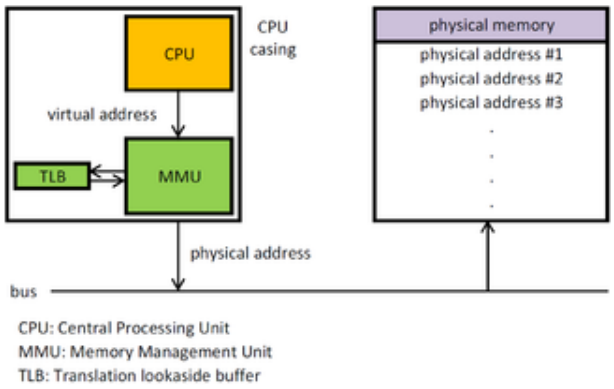
CU Ďalšou časťou von Neumannovej architektúry je riadiaca jednotka [11]. Je to komponent CPU, ktorý riadi činnosť procesora. Hovorí pamäti, ALU, a vstupno-výstupným zariadeniam ako majú reagovať na inštrukcie programu. Riadi operácie ostatných jednotiek poskytovaním časových a riadiacich signálov. CU riadi tok dát medzi CPU a ostatnými zariadeniami. Jednotlivé inštrukcie prechádzajú cez CU, ktorá ich dekóduje na niekoľko sekvenčných krokov, ktoré riadia a koordinujú vnútorné mechanizmi CPU pre správnu manipuláciu dát.

Registre Registre sú rýchlo dostupné oblasti prístupné procesoru. Zvyčajne obsahujú malé, no veľmi rýchle úložisko, hoci môžu mať aj iné, špecifické hardvérové funkcie. Pamäť v registri býva typicky adresovateľná mechanizmami oddelenými od hlavnej pamäte, no v istých prípadoch môžu mať pridelenú adresu pamäte. Skoro vo všetkých architektúrach sa dáta načítavajú z väčších pamätí do registrov, odkiaľ sa používajú v aritmetických operáciach. Výsledky sa bežne ukladajú do hlavnej pamäti. Poznáme rôzne kategórie registrov, líšiace sa účelom, použitím a množstvom bitov, ktoré môžu uchovávať.

- Používateľsky-prístupné registre
  - Dátové registre
  - Adresové registre
  - GPR [12] - môžu uchovávať dáta i adresy.
  - Stavové registre
  - Floating-point registre
  - Registre konštánt - napr. 0, 1, π
  - Vektorové registre - uchovávajú dáta pre SIMD[13]
  - SPR[14] - uchovávajú stav programu. Patrí sem program counter, status register.
  - MTRR[15] - sada registrov, ktoré poskytujú systému riadenie toho, ako sú prístupy CPU k pamäti cachované.
- Vnútorné registre
  - Register inštrukcií - drží informáciu o momentálne vykonávanej inštrukcií.
  - Registre oddelené od CPU súvisiace so získavaním informácie z RAM
    - Memory buffer register (MBR)
    - Memory data register (MDR)
    - Memory address register (MAR)
- Architektúrne registre

[11] Control Unit
[12] General-purpose registers
[13] Single Instruction, Multiple Data
[14] Special-purpose registers
[15] Memory Type Range Registers

Jednotka prideľovania pamäte Memory Management Unit je komponent, skrz ktorý prechádzajú všetky pamäťové referencie a ktorý predovšetkým vykonáva preklad adries virtuálnej pamäte na fyzické adresy. Spravidla býva implementovaná ako súčasť CPU, avšak môže sa vyskytovať aj ako samostatný integrovaný obvod. MMU sa stará o manažment virtuálnej pamäte, tzn. riadi ochranu pamäte, riadenie cache, a v jednoduchších architektúrach aj tzv. "bank switching"[16] Moderné MMU typicky delia adresný priestor na stránky, ktoré majú veľkosť v násobkoch 2, zvyčajne pár kilobytov, hoci môžu byť aj väčšie.



Obr. 1.2. Schéma práce MMU.

1.2. Harvardská a Princetonská architektúra

Princetonská architektúra Taktiež známa ako von Neumannová, podľa svojho autora, matematika a fyzika Johna von Neumanna. Popisuje dizajnovú architektúru pre elektronický digitálny počítač pozostávajúci z výpočtovej jednotky[17], riadiacej jednotky[18], pamäť na uchovávanie dát a inštrucií, externé úložisko, a vstupno-výstupné mechnizmy. V počítačoch von Neumannovej architektúry sú dáta a inštrukcie ukladané do jednej pamäti, čo znamená, že načítanie inštrukcií a vykonanie operácie sa nemôže diať naraz, keďže zdieľajú zbernicu. Táto limitácia sa nazýva **von Neumannove obmedzenie**[19] a častokrát býva limitujúcim faktorom výkonu systému. CPU musí často čakať na dodanie potrebných dát z pamäte. Keďže rýchlosť CPU a pamätí sa zvyšovali omnoho rýchlejšie ako rýchlosť zbernice medzi nimi, toto obmedzenie sa stalo väčším problémom a je nutné ho rôzne obchádzať. Najčastejšie spôsoby mitigácie obmedzenia sú použitím:

- vyrovnávacej pamäte (cache) medzi CPU a operačnou pamäťou,
- viacerých oddelených cache alebo oddelených prístupovýc ciest pre dáta a inštrukcie (tzv. Modifikovaná Harvardská architektúra),
- algoritmy predikcie vetvenia (branch predictor algorithms)
- limitovanej haldy alebo inej pamäte na čipe pre zníženie počtu prístupov k pamäti.

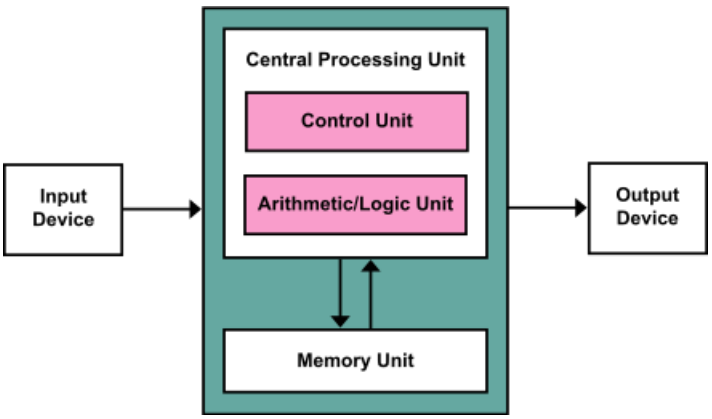
Tento problém sa tiež dá čiastočne obísť paralelným spracovaním. Príkladom je aj Non-Uniform Memory Access (NUMA) architektúra, častokrát používaná v superpočítačoch. Vo von Neumannovej koncepcií môže dochádzať k samomeniacému kódu, čo môže mať v nepriaznivých prípadoch za následok pád programu, či samotného operačného systému. Tomuto zabraňujú mechanizmy ochrany pamäte. Niekedy však môže byť modifikácia programu za jeho chodu aj žiaduca.

[16] Technika ktorou sa v starších osem bitových počítačoch rozširovala pamäť nad rámec normálne adresovateľného priestoru. Bloky pamäte, ku ktorej bolo potrebné pristupovať len raz, napr. pri boote systému sa mohli vymeniť s prázdnou časťou operačnej pamäte.

[17] Obsahujúcej aritmeticko-logickú jednotku a regitre procesoru

[18] Obsahujúcej register inštrukcií a program counter

[19] von Neumann bottleneck



Obr. 1.3. Schéma konepcie Johna Von Neumanna

Harvardská architektúra je architektúra s fyzicky oddeleným úložiskom a spojmi pre inštrukcie a dáta. Meno pochádza z počítača Harvard Mark I, postaveného na relé, ktorý uchovával inštrukcie na diernej páske a dáta v elektromechanických počítadlách. Počítače tejto éry mali dátové úložisko uzavreté v CPU a neposkytovali žiaden prístup k inštrukciam ako k dátam. Programy preto museli byť načítané operátorom, procesor sa nemohol sám inicializovať. V dnešnej dobe sa oddelené spoje používajú pre navýšenie výkonu, ale v skutočnosti sa jedná o tzv. Modifikovanú Harvardskú architektúru, ktorá umožňuje načítanie programov z diskov ako dáta a ich následné vykonanie.

V Harvardskej architektúre nemusia mať dátová a inštrukčná pamäť rovnaké vlastnosti. Šírka slova, časovanie, implementácia, adresová štruktúra môžu byť rozdielne. V niektorých systémoch bývajú inštrukcie uložené len v Read-Only pamäti, pričom dáta bývajú v Read-Write pamäti.

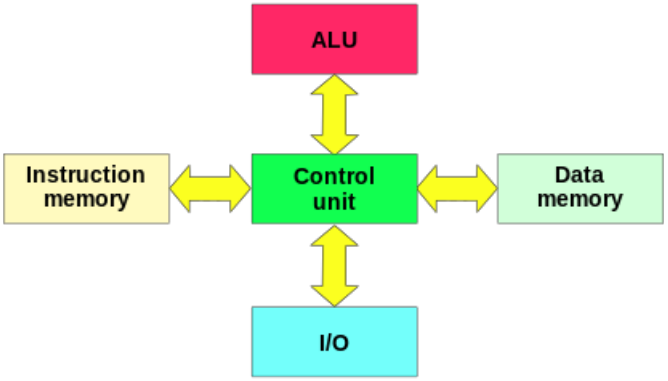
Na rozdiel od Princetonskej architektúry, v Harvardskej môže CPU naraz čítať inštrukcie i pristupovať k dátam, bez nutnosti použiť vyrovnávajúcu pamäť. Harvardský systém môže byť preto rýchlejší pri porovnateľnej komplexnosti.

Modifikovaná Harvardská architektúra sa líši od štandardnej tým, že sa v nej upúšťa od striktného oddelenia dát a inštrukcií, pričom stále umožnuje CPU naraz pristupovať k viacerým pamäťovým zberniciam. Najčastejšie modifikácie zahŕňajú oddelené vyrovnávajúce pamäte pre inštrukcie a dáta so spoločným adresným priestorom. Kým CPU pracuje s cache, správa sa ako pravý Harvardský stroj. Keď pristupuje k pamäti, jedná ako von Neumannov stroj, pričom programy môžu byť manipulované ako dáta. Táto modifikácia je bežná v moderných procesoroch, napríklad v architektúrach ARM, IBM Power, alebo x86.

Napriek rozšírenosti modifikovanej varianty, aj pravá Harvardská architektúra má dodnes uplatnenie v praxi. Používa sa najmä v systémoch, kde ušetrené energetické náklady a cena z vynechania cache prekonávajú vyššie programátorské nároky.

- Digital Signal Processor (DSP) častokrát vykonávajú malé, vysoko optimalizované audio, alebo video spracujúce algoritmy. Vynechávajú cache, keďže ich správanie musí byť extrémne zopakovateľné. Náročnosť pracovania s viacerými adresnými priestormi je vedľajšia oproti nárokom na rýchlosť vykonávania. Súčasne niektoré DSP obsahujú viacero dátových pamätí s oddelenými adresnými priestormi pre uľahčenie SIMD a VLIW. Príkladom takéhoto procesora je Texas Instruments TMS320 s dvoma zbernicami na zápis, troma na čítanie a jednou pre inštrukcie.

- Mikrokontroléry sú charakteristické tým, že majú malé programové a datové pamäte a využívajú Harvardskú architektúru kvôli jej rýchlosti a súčasný prístup k dátam a inštrukciam. Častokrát mávajú rozdielne bitové šírky pamätí, napr. 8-bit pre dáta, 16-bit pre inštrukcie. Taktiež môžu vykonávať predprípravu inštrukcií paralelne s ostatnými aktivitami. Takýmito mikrokontrolermi sú napríklad PIC od Microchip Technology a Atmel AVR.



Obr. 1.4. Schéma Harvardskej architektúry

1.3. RISC a CISC

ISA Architektúra inštrukčnej sady je abstraktný model počítača. Realizácia ISA sa nazýva jej implementáciou. ISA definuje všetko čo musí programátor strojového jazyka vedieť na programovanie počítača. To, čo jednotlivé ISA definujú sa líši prípad od prípadu. Vo všeobecnosti definuje podporované dátové typy, stavy a ich sémantiky, inštrukčnú sadu a I/O model. ISA sa líši od mikroarchitektúry, čo je sada dizajnových techník použitých v konkrétnom procesore na implementáciu inštrukčnej sady. Procesory rôznych mikroarchitektúr môžu mať spoločnú inštrukčnú sadu. Napríklad procesory Intel Pentium a AMD Athlon implementujú takmer totožné verzie x86 inštrukčnej sady, hoci majú radikálne odlišný vnútorný dizajn. Dve hlavné kategórie ISA sú CISC[20] a RISC [21]. Hoci sa jedná o protichodné koncepty, rozdiely medzi nimi sú však občas ťažko rozpoznateľné, keďže napríklad niektorí autori opisujú mikrokontroléry PIC ako RISC, iní zas ako CISC. Taktiež procesory x86, hoci sú CISC, mnoho z nich vnútorne používa mikrokód na konverziu komplexných inštrukcí na sekvenciu jednoduchších, ktoré potom vykoná. Hovorí sa preto, že hoci x86 **inštrukčná sada** je CISC, moderná x86 **architektúra** je RISC.

CISC	RISC
Dôraz na hardvér	Dôraz na softvér
Aj viac-taktové komplexné inštrukcie	Len jedno-taktové redukované inštrukcie
”LOAD” a ”STORE” vstavané v inštrukciách	”LOAD” a ”STORE” ako nezávislé inštrukcie
Menšia veľkosť kódu, viac cyklov za sekundu	Menej cyklov za sekundu, väčšia veľkost kódu
Tranzistory ukladajú komplexné inštrukcie	Viac tranzistorov v pamäťových registroch

CISC Starší z typov. Jednotlivé inštrukcie sú schopné vykonávať viacero nízko-úrovnňových operácií[22], alebo viackrokových operácií, či adresových módov počas jednej inštrukcie. Keďže sa tento názov začal používať retroaktívne po nástupe RISC procesorov, slúži ako kolektívny názov pre všetko, čo nie je RISC, či už sa jedná o komplexné sálové počítače, po najjednoduchšie mikrokontroléry, kde ukladanie a načítanie z pamäti nie sú oddelené od aritmetických inštrukcií. Z tohto dôvodu môže byť paradoxne moderný RISC procesor výrazne komplexnejší ako napríklad moderný CISC mikrokontroler, obzvlášť z hľadiska komplexnosti elektrických obvodov, ale aj počtom inštrukcií či ich zložitosti. Jediná typicky odlišujúca charakteristika je, že väčšina RISC dizajnov používa jednotnú inštrukčnú dĺžku pre skoro všetky svoje inštrukcie a striktne oddeluje inštrukcie load/store od ostatných.

Výhody CISC Predtým, než sa uplatnila dizajnová filozofia RISC, mnoho počítačových architektov sa snažilo zmenšiť takzvanú ”sémantickú priepasť”, tzn. navrnúť inštrukćné sady priamo podporujúce programovacie konštrukty vyššej úrovne, napr. volania procedúr, cykly a komplexné adresné režimy, umožňujúce dátovú štruktúru a prístupy k poliam skombinovať do jednotlivých inštrukcií. Inštrukcie sú typicky vysoko kódované, aby sa navýšila hustota kódu. Takáto kompaktnosť vedie k menším programom a menšiemu počtu prístupov do hlavnej (pomalej) pamäti, čo v dobe vzniku[23] viedlo k veľkému šetreniu na cene pamäte a úložiska, a aj rýchlosti výpočtov. Taktiež to viedlo k lepšej produktivite programátorov aj v assembly, keďže vysoko úrovňové jazyky ako Fortran a Algol neboli vždy dostupné, alebo vhodné.

Pridávanie inštrukcí V 70. rokoch sa predpokladalo, že pridanie ďalších inštrukcií by mohlo navýšiť výpočtový výkon. Kompilátory boli aktualizované, aby využili tieto inštrukcie

[20] Complex Instruction Set Computer
[21] Reduced Instruction Set Computer
[22] Napr. načítanie z pamäte, aritmetické operácie, či uloženie do pamäti.
[23] 60. roky 20. storočia

a takýto prístup sa používa v procesoroch dodnes, keďže prístupy do hlavnej operačnej pamäte sú stále niekoľko rádovo pomalšie ako činnosť jadra CPU.

Nevýhody dizajnu Kým úmyslom navyšovania počtu inštruikcí bolo zvýšiť výkon a umožniť vysoko úrovňovým jazykovým konštruktom vyjadrenie menším počtom inštrukcií, nie vždy sa to takto darilo. Architekti občas predizajnovali inštrukcie strojového jazyka, zahrňujúc funkcie, ktoré nemohli byť efektívne implementované na hardvéri.

RISC Táto architektúra umožnuje počítačom potrebovať menej cyklov na inštrukciu, ako porovnateľný CISC. Koncept RISC sa datuje do 80. rokov minulého storočia. O jeho popularizáciu sa zaslúžili najmä dva projekty zo Standfordskej univerzity a Univesity of California, Berkeley. Standfordská MIPS architektúra sa neskôr úspešne skomercializovala, kým architektúra vyvinutá v Berkeley, RISC, dala názov celému konceptu a ujala sa ako SPARC. Rôzne varianty RISC dizajnu zahrňujú ARC, Alpha, Am29000, ARM, Atmel AVR, Blackfin, i860, i960, M88000, MIPS, PA-RISC, Power ISA (including PowerPC), RISC-V, SuperH, and SPARC.

Hoci sa podobné koncepty a implementácie objavovali už skôr, napr. CDC 6600 navrhnuté Seymourom Crayom v 1964, Michael J. Flynn považuje za prvý RISC systém IBM 801. Do vývoja RISC sa dapojila aj DARPA skz svoj VLSI[24] program. Berkeley RISC bol navrhnutý tak, aby získal náskok vo výkone použitím pipeliningu a agresívnym používaním techniky register windowing. V tradičnom CPU je len malý počet registrov a program k nim môže kedykoľvek pristupovať. V CPU s oknami registrov je obrovský počet registrov, ale programy môžu využívať naraz len malý počet z nich. Program, ktorý sa limituje na osem registrov na procedúru môže vykonávať veľmi rýchle volania. Volanie jednoducho "posunie" okno nadol o osem, k sade registrov, ktoré procedúra vyoužíva a pri návrate posunie okno späť. RISC-I procesor, vyvinutý v 1982 pozostával z 44 420 tranzistorov (oproti cca 100 000 v CISC procesoroch tej doby), mal len 32 inštrukcií, no vo výkone prekonal všetky ostatné jednočipové dizajny. Neskôr ho predčili procesorom RISC-II s 40 760 tranzistormi a 39 inštrukciami, ktorý bol 3x rýchlejší ako RISC-I.

V 2010 sa objavila nová, open source ISA RISC-V, vyvinutá v Berkeley pre výskumné účely a ako slobodná alternatíva voči proprietárnym ISA. V súčasnosti sa objavujú prvé implementácie vo forme SiFive procesorov.

ZISC Zero Instruction Set Computer označuje architektúru založenú čiste na priraďovaní vzorov a absencií inštrukcií v klasickom zmyysle. Tieto čipy sú porovnateľné s neurónovými sieťami, rozlišované počtom synapsí a neurónov. Prvé komerčné ZISC boli vyvinuté firmou IBM v 90. rokoch.

[24] Very-Large-Scale Integration

2. Multiprocesorový počítač a multiprocesorový systém

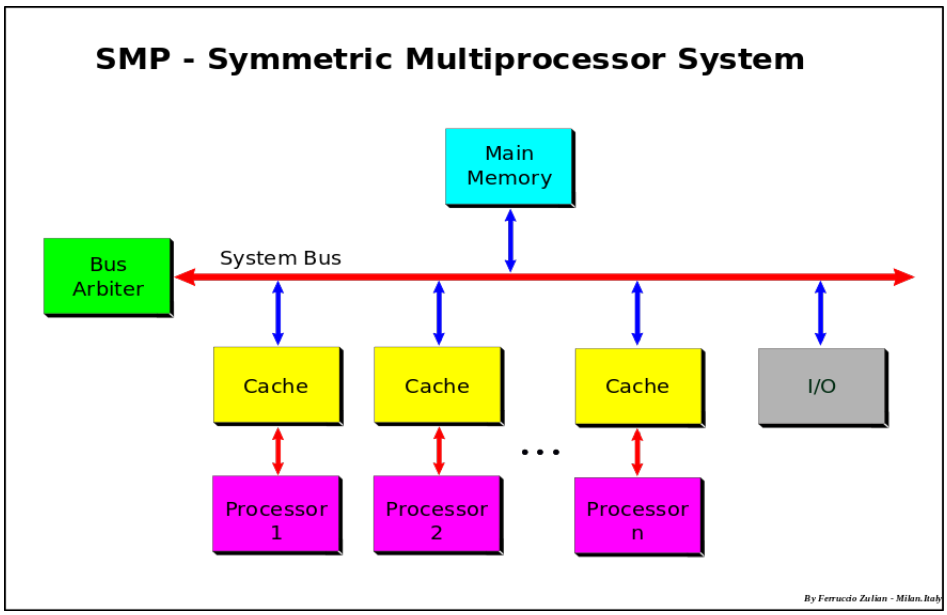
Korpus: *Definícia, rozdelenie podľa kategórií, stavba vnútornej pamäte. Využitie paralelných počítačov z hľadiska Flynnovej klasifikácie v súčasnosti, paralelizmus v procesoroch (porovnanie jedno a viacjadrovej architektúry, viacprocesorovej architektúry), Amdahlov zákon. Je potrebné, aby študent vedel preukázať opodstatnenosť využitia paralelizmu v súčasných typoch procesorov a počítačových systémov napr. formou Grid technológie, alebo iného spôsobu spracovania veľkého počtu požiadaviek v relatívne krátkom čase.*

Multiprocesorový počítač, systém je taký, kde sa na výpočtoch podiela viac ako jeden procesor. Ako taký sa za multiprocesorový systémy považujú počítače s viacjadrovými procesormi, ale aj clustere a gridy. V multiprocesorovom systéme môžu byť všetky jadrá rovnaké, alebo môžu byť niektoré rezervované pre špeciálne účely. Kombinácia hardvéru a operačného systému určuje symetriu systému. V niektorých systémoch môže byť jeden CPU reagovať na všetky Hardvérové prerušenia, kým všetka ostatná činnosť je rovnomerne distribuovaná na zvyšné jadrá. Z hľadiska symetrickosti preto takéto prípady delíme na dve kategórie:

- Symmetric Multiprocessor System - SMP,
- Asymmetric Multiprocessor System - AMP,

2.1. Kategorizácia podľa symetrie systému

SMP Symetrický multiprocesing vyžaduje hardvérovú a softvérovú architektúru, kde sú dva alebo viac identické procesory prepojené na jednu zdieľanú pamäť, majú plný prístup k všetkým vstupno-výstupným zariadeniam a sú ovládané jednou inštanciou operačného systému, ktorá sa správa ku všetkým procesorom rovnako bez vyhradenia žiadneho na špeciálne účely. Väčšina dnešných multiprocesorových systémov má symetrický charakter. V prípade viacjadových procesorov sa SMP aplikuje na jadrá, správajúc sa k nim ako k separátnym procesorom. SMP systémy sú spravidla **tesne viazané multiprocesorové systémy** s množstvom homogénnych procesorov pracujúcich nezávisle od ostatých. Každý procesor vykonáva iné programy a pracuje s inými sadami dát, má možnosť zdieľať spoločné zdroje, ako sú pamäť, I/O, systém prerušení, apod., ktoré sú prepojené či už zbernicou, krížovo, alebo ako tzv. "mesh network". V prípade prepojenia zbernicou alebo krížovo je škálovateľnosť systému obmedzená priepustnosťou a energetickou náročnosťou prepojení. Mesh siete obchádzajú všetky tieto obmedzenia a prinášajú takmer lineárnu škálovateľnošt, avšak za cenu výrazne náročnejšieho programovania.



Obr. 2.1. Symmetric Multiproccessing System

AMP V prípade asymetrického multiprocesingu sa systém nespráva ku všetkým procesorom rovnako. Systém môže, napríklad, iba jednomu procesoru prideliť vykonávanie kódu operačného systému, prípadne dovoliť len jednému CPU pristupovať k I/O. Iné AMP dizajny dovolujú ktorémukoľvek CPU vykonávať kód operačného systému aj vstupno-výstupné operácie (čím sú všetky CPU symetrické vzhľadom na role procesorov), ale pripojiť niektoré periférie k špecifickým CPU, takže sú asymetrické vzhľadom na pripojenie periférií. Asymetrický multiprocesing je starším prístupom ako SMP, avšak používal sa aj v systémoch, kde bolo možné SMP, ako menej drahá alternatíva. Taktiež sa používa v dedikovaných aplikáciach, napríklad vstavaných systémoch, kde môžu byť jednotlivé procesory dedikované určitým úlohám už v štádiu dizajnovania systému. Tu môžeme ako príklad brať dizajn procesora A11 Bionic v iPhone X. Jedná sa o 6-jadrový procesor s dvoma výkonnými Monsoon jadrami a štyrmi energeticky úspornými Mistral jadrami. Tie môžu aplikáci využívať v akýchkoľvek kombináciach, i všetkých šesť naraz. Okrem toho sú do SoC integrované aj rôzne koprocesory, ako napríklad M11 motion coprocessor[1], Neural Engine[2], ktorý je použitý na FaceID, Animoji a iné úlohy strojového učenia.

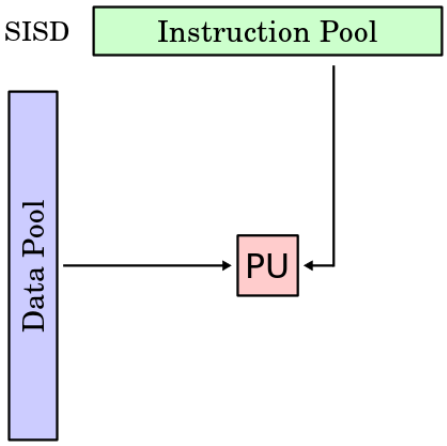
2.2. Rozdelenie z hľadiska Flynnovej klasifikácie

Flynnova taxonómia je klasifikáciou počítačových architektúr navrhnutá Michaelom J. Flynnom v 1966. Tento systém klasifikácie pretrváva a používa sa ako nástroj pri navrhovaní moderných procesorov a ich funkcionalít. Triedy tejto klasifikácie sú:

- Single Instruction stream, Single Data streams - SISD,
- Single Instruction stream, Multiple Data streams - SIMD,
- Multiple Instruction stream, Single Data streams - MISD,
- Multile Instruction stream, Multiple Data streams - MIMD,
- Single Instruction, Multiple Threads - SIMT.

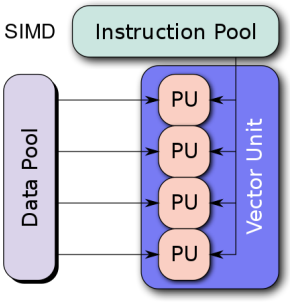
[1] Ktorý spracováva údaje z gyroskopu
[2] Integrujúci hardvérovú neurónovú sieť

SISD Jedná sa o sekvenčný počítač, ktorý nevyužíva žiadnu paralelizáciu či už v inštrukčnom, alebo dátovom toku. Jedna riadiaca jednotka načítava jeden inštrukčný tok z pamäte. CU potom generuje kontrolné signály na nasmerovanie jednej procesorovej jednotky na spracovanie jedného dátového toku. Prikladom sú staré jednoprocesorové PC a sálové počítače. Podľa Flynna, aj SISD môžu ma charakteristiky konkurentného spracovávania. V modernej aplikácií sa jedná procesory s pipeliningom[3] a superskalárne procesory[4].



Obr. 2.2. SISD

SIMD Popisuje počítače s viacerými procesnými jednotkami, ktoré vykonávajú tú istú operáciu na rôznych dátach súčasne. Takéto stroje využívajú paralelizmus na úrovni dát, avšak niek konkurenciu - vykonáva sa viacero paralelných výpočtov, ale len jeden proces (inštrukcia) v danom okamihu. SIMD sa aplikuje na bežné operácie ako napríklad úpravu kontrastu v digitálnych obrázkoch, alebo úpravu hlasitosti digitálnom zvuku. Väčšina moderných CPU pridávajú SIMD inštrukcie pre zlepšenie výkonu v multimédiach. Ako prvý prípad širokého nasadenia SIMD v osobných počítačoch sa berú MMX[5] rozšírenia v Intel procesoroch v roku 1996. Nevýhodou SIMD je, že nie všetky úlohy sa dajú jednoducho vektorizovať, SIMD vyžaduje veľké registre, ktoré zvyšujú veľkosť čipu a energetickú spotrebu a programovanie pre SIMD inštrukcie sa zväčša musí robiť ručne, keďže väčšina kompilátorov negeneruje SIMD inštrukcie.



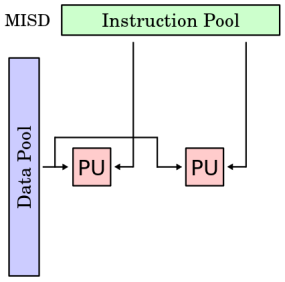
Obr. 2.3. SIMD

[3]Technika paralelizmu na úrovní inštrukcií v jednom procesore. Pipelining sa snaží zamestnávať všetky časti proesora tým, že rozdelí prichádzajúce inštrukcie na sériu sekvenčných krkokov, vykonávaných rôznymi procesorovými jednotkami s rôznymi časťami inštrukcií vykonávaných paralelne. Pipelining umožnuje vyššiu priepustnosť CPU, avšak môže mať za následok vyššiu latenciu, kvôli réžií, ktorú vyžaduje samotný proces pipeliningu.

[4]Procesory, ktoré na rozdiel od obyčajných skalárnych procesorov môžu vykonávať viacero inštrukcií v jednom cykle súčasným rozoslaním viacerých inštrukcií do rôznych výkonných jednotiek procesora. Výkonné jednotky (execution units) nie sú separátne procesory, ani jadrá, ale obvody v CPU, ako napríklad ALU.

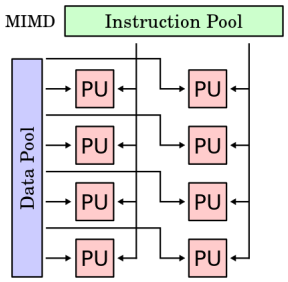
[5]Matrix Math eXtentions

MISD Typ architektúry paralelného výpočtu, kde mnoho funkčných jednotiek vykonáva rôzne operácie na jednej sade dát. V praxi nepríliš bežná architektúra, keďže MIMD a SIMD bývajú viac vhodné pre bežné paralelizačné techniky a umožňujú lepšie škálovanie. Jedným z príkladov použitia MISD boli počítače letovej kontroly vesmírnych raketoplánov NASA.



Obr. 2.4. MISD

MIMD Viacero autonómnych procesorov naraz vykonávajúcich rôzne inštrukcie na rôznych dátach. Patria sem viacjadrové superskalárne procesory a distribuované systémy používajúce buď jeden zdieľaný pamäťový priestor alebo distribuovaný pamäťový priestor. Procesory v MIMD strojoch pracujú nezávisle a asynchrónne. Používajú sa v počítačovom dizajne/výrobe, simulácií, modelingu a ako komunikačné prepínače. MIMD so zdieľanou pamäťou môžu byť zbernicové, rozšírené alebo hierarchického typu. Distribuované môžu mať schémy pripojenia buď typu hyperkocky (tesseract), alebo mesh siete.



Obr. 2.5. MIMD

SIMT

2.3. Rozdelenie podľa viazania multiprocesorového systému

3. Bezpečnosť technických prostriedkov, NAS servery

Korpus: Bezpečnosť technických prostriedkov a ukladania dát na Slovensku, problematika DAS (Direct Attached Storage), NAS (Network Attached Storage) a SAN (Storage Area Network) a Fault Tolerant systémy. Študent musí vedieť definovať výhody nasadenia SAN, z čoho pozostáva dnešné typické SAN riešenie.

4. Riadiace počítače a ich využitie v priemyselnej technike

Korpus: Definícia a architektúra riadiaceho počítača, základné požiadavky kladené na riadiace počítače z hľadiska ich implementácie do výrobného procesu. Je potrebné aby študent vedel definovať riadiaci počítač z pohľadu jeho vnútornej architektúry a zároveň vedel uviesť najbežnejšie príklady riadiacich počítačov používaných v súčasnosti v praxi (minimálne 3) a porovnal ich z pohľadu počtu a možnosti využitia analógových/digitálnych vstupov a výstupov, riadiacej jednotky, pamäte a pod.

5. Cloud computing

Korpus: Definícia pojmu cloud computing, hlavné charakteristiky a modely služieb Cloud computingu (Software as a Service (SaaS), Platforma as a Service (PaaS), Infrastructure as a Service (IaaS)), praktický príklad použitia cloudu, výhody, nevýhody a riziká využitia cloudu.

6. Distribuované systémy a distribuované výpočty

Korpus: Definícia distribuovaných systémov, úrovne transparentnosti v distribuovaných systémoch, architektúry a modely distribuovaných systémov.

7. Migrácia procesov

Korpus: Stavové modely procesov, definícia procesu a vlákna, mapovanie vlákien, migračné stratégie.

8. Medziprocesová komunikácia

Korpus: Základné princípy medziprocesovej komunikácie, synchrónna a asynchrónna komunikácia, vzdialené volanie procedúr, sockety, MPI komunikácia.

9. Synchronizácia hodín

Korpus: Dôvod synchronizácie hodín, mechanizmy šírenia časového signálu, logické hodiny (Lamportov algoritmus), synchronizácia fyzických hodín (Cristianov algoritmus), protokol NTP, Berkeley algoritmus.

10. Bezdrôtové senzorové siete

Korpus: Architektúry bezdrôtových senzorových sietí, lokácia v bezdrôtových senzorových sieťach, praktické využitie senzorových sietí.

11. Charakteristika expertných systémov

Korpus: Znalostný systém, silné a slabé metódy riešenia problémov, atribúty znalostí (poznatkov), charakteristické vlastnosti znalostných systémov, výhody oddelenej reprezentácie poznatkov, ohraničenia - problémy znalostných systémov, typy problémov pre znalostné systémy, základné architektúry znalostných systémov, minimálna kostra znalostného agenta.

12. Reprezentácia znalostí

Korpus: Reprezentácia poznatkov, požiadavky na jazyk pre reprezentáciu poznatkov, logika (výroková, predikátová), pravidlá, vizualizácia pravidiel pomocou AND/OR grafu, sémantické siete, rámce, konceptuálne mriežky.

13. Odovodzovací mechanizmus

Korpus: Odvodzovací mechanizmus, vnútorný a vonkajší pohľad na odovodzovací mechanizmus, usudzovanie s pravidlami, stratégie riešenia konfliktov, dopredné a spätné zreťazenie, optimalizácia tvorby aplikovateľných pravidiel, usudzovanie so sémantickými sieťami, usudzovanie s rámcami.

14. Zložky znalostného systému a tvorba znalostných systémov

Korpus: Vstupno-výstupný modul, vysvetľovací modul, životný cyklus ZS, etapy získavania znalostí (identifikácia a analýza problému, konceptualizácia, formalizácia, implementácia, validácia), účastníci v procese tvorby ES, techniky získavania znalostí, spoľahlivosť a neúplnosť znalostí, interview, voľné asociácie, monitorovanie, komentovanie postupu, dialógy, príznakové mriežky a postupy pri jej vytváraní.

15. Rozhodovacie stromy

Korpus: Extrakcia znalostí z údajov, rozhodovacie stromy, rozhodovací strom ako klasifikátor, algoritmus tvorby rozhodovacieho stromu, nájdenie vhodnej vlastnosti pre delenie, trénovanie rozhodovacieho stromu, prepis rozhodovacieho stromu na pravidlá

16. Základné pojmy v kybernetike

Korpus: Definícia pojmu kybernetika, historický prehľad, základné pojmy, štruktúra kybernetiky, teoretická, aplikovaná a technická kybernetika.

17. Kybernetický systém, riadenie dynamických systémov

Korpus: Definícia systému riadenia, riadený a riadiaci systém, regulačný obvod, regulovaná sústava, regulátor.

18. Typy kybernetických systémov a prvky ich riadenia

Korpus: Otvorený - uzavretý, lineárny - nelineárny, spojitý – nespojitý, jednoduchý – rozvetvený, jednoparametrový –viacparametrový, regulácia, cieľ riadenia, informáciu o stave okolia a riadeného systému, pôsobenie na objekt (vlastné riadenie), algoritmus riadenia (pravidlo udávajúce, aký spôsobom je možné dosiahnuť cieľ na základe informácie o stave okolia a riadeného systému).

19. Prostriedky automatického riadenia v kybernetike

Korpus: Získavanie, prenos, úprava a spracovanie informácií, akčné a riadiace systémy

20. Automatizované systémy

Korpus: Riadenie výrobných strojov, robotov a výrobných systémov. Automatizácia nevýrobnej sféry - inteligentné domy, zabezpečovanie, informačné systémy, trendy automatizácie.