Historie

Computer – latinsky *computare* = počítat; 1646 sir Thomas Browne – stroj provádějící náročné výpočty
Dnešní chápání počítače – univerzální programovatelný stroj
Nutné základy: Formalizace – filosofický přístup k formální logice – Aristoteles 4 BC
Matematizace – „*kalkulus*“ latinsky *calculus* = kamínek – čísla jako taková
Mechanizace

? Stonehenge, abacus, tabulky, pravítka, strojky, mechanické tabulátory a kalkulátory
1642 B. Pascal – Pascalina – počítání daní; asi 50 kusů, uměl jen on
1671 Gotlieb Wilhelm von Leibnitz – mělo by to počítat ve dvojkové soustavě
1805 Joseph Marie Jacquard – Tkalcovský stav, který pomocí kartiček měnil vzor
1820 Charles Xavier Thomas de Colmar – Thomasův arithmometr
1822 Difference Engine – C. Babbage
1904 elektronka
1947 tranzistor
1958 integrovaný obvod
1971 mikroprocesor 4004

Charles Babbage: (1791-1871)
Algorytmizace – pozoroval továrny a navrhoval zlepšení – mnoho práce zvládnou tupé stroje
1822 *Difference Engine* – hodnoty polynomů do šestého stupně; nedokončil; paměť, řídící jednotka, I/O
1842 *Analytical Engine* –univerzální, řízení programem na děrných štítcích; podmíněný skok

Herman Hollerith: (1860-1929); 1890 stroj na sčítání lidu; vstup na děrných štítcích

Konrád Zuse: (1911-1995); Studium stavebního inženýrství
Z1 : 1936-1938 – mechanický, binární kód
Z2 : 1940 – experiment
Z3 : 1938-1941 – 2400 relé, děrná páska; považován za „první funkční počítač na světě“
binární, 64 slov po 22 bitech; floating-point (14 mantissa, 7exp., 1 znam.)
Z4 : 1941-1945 – podobná architektura jako Z1 a Z3; paměť 1024 slov/ 1m^2; 2 registry R1 a R2

Alan Mathison Turing: (1912-1954) základy oboru *umělé inteligence*; 1936 hypotetická zařízení *Turingův stroj*
během 2. sv. války v Británii, pokusy o dešifrování Enigmy -> 1943 Colossus
1950 „Turingův test“; práce z oblasti vztahu člověka a stroje, položil základy vědy o „umělé nteligenci“

Grace Murray Hopper: Harvard Mark I – mechanický, 5t, 530 mil drátu, program na děrné pásce bez návratu
Zavedla pojem „bug“, když vlétla můra do relé a připekla se tam

ENIAC: uveden do provozu na Pensylvánské univerzitě v r. 1946
používal desetimístná dekadická čísla; použita „rychlá“ registrová paměť; 17 468 elektronek
ALU obsahovala dvacet sčítaček, speciální násobičku, děličku a obvody pro výpočet druhé odmocniny
programoval se pomocí propojování speciálních programových jednotek

EDVAC: Vnitřní paměť na zpožďovacích rtuťových linkách; instrukce jiný formát než data

John Louis von Neumann: (1903-1957); zakladatel „teorie her“
Myšlenka univerzálního stroje; struktura nezávisí na řešení
Program v paměti, nerozlišitelný od dat
Paměť tvořena buňkami stejné velikosti; instrukce dle pořadí v ní

Počítač EDSAC: Podle von Neumannovy koncepce; programování pomocí telefoního číselníku; elektronkový

zdokonalování von Neumannovy koncepce: 1947 formulace von Neumannovy koncepce

indexregistry; jednotka pro operace v pohyblivé řádové čárce; přerušení, univerzální registry, asynchronní činnost I/O;nepřímé adresování virtuální paměť
Harvardská univerzita – zvlášť program a zvlášť data; počítač řízený tokem dat???

Počítače IBM: elektronkový s registry/ paměťovou elektronkou/ magnetickou bubnovou pamětí
feritové paměti; první počítač vybavený polovodičovou technologií
„Prodáme zákazníkovi, co chce“ zbytek zablokován, lze odblokovat třeba jen na určité dny…

IBM 360: (1964); postaven na integrovaných obvodech; původně přesně podle von Neumanna
zásadní změny výstavby: stavebnicová konstrukce; jednotná struktura dat a instrukcí
jednotný způsob připojování periferií; ochrana dat v paměti

Vývoj v Čechách: první návrh počítače představen na Badatelském ústavu matematickém (1947)
některé části návrhu pokusně realizovány, např. elektronková násobička
založeno oddělení počítacích strojů v Ústředním ústavu matematickém ČSAV
SAPO: 400 elektronek, 700 relé; binární aritmetika v pohyblivé řádové čárce; délka slova 32 bitů
3-5 operací za sekundu; pětiadresové instrukce

Generace počítačů: Relé; elektronky + bubnová paměť; tranzistory + feritová paměť; SSI; MSI; LSI
Současné třídy počítačů: Vestavěné – spotřební elektronika (mobily, hodinky…)
Osobní – PC, notebooky…
Servery – podpora infrastruktury, náročnější aplikace
Superpočítače – orientace na výpočetní výkon
Mainframe – orientace na dostupnost, spolehlivost, propustnost, hodně malých operací
Vysoké objemy dat – 60 % netu; vysoké objemy dat, výkonu, uživatelů…

Von Neumannova architektura
Paměť: Posloupnost buněk stejné velikosti
Adresa dána pořadovým číslem bez ohledu na obsah
Program: Program je uložen v paměti, nelze rozlišit od dat
Pogram se nemění při změně vstupních dat
Posloupnost elementárních příkazů v paměti
Pořadí provádění se mění pouze instrukcemi skoku
Univerzální počítač: Struktura počítače nezávisí na typu úlohy

Architektura

Vliv na návrh počítače: abstraktní úroveň – matematika, filozofie, logika
technologická úroveň – fyzika, elektronika, ekonomie

Víceúrovňová organizace počítače :
L_0 – mikroprocesorová úroveň (nad abecedou {0,1}); přímo technické vybavení počítače
L_1 – strojový jazyk počítače (nad abecedou {0,1}); virtuální stroj nad obvodovým řešením
Jak spolu mají jednotky komunikovat; zpřístupňuje základní logické celky
L_2 – úroveň operačního systému; doplnění L_1 o soubor makroinstrukcí a novou organizaci paměti
L_3 – úroveň assembleru; nejnižší úroveň lidsky orientovaného jazyka
L_4 – úroveň vyšších programovacích jazyků; první nestrojově orientovaná úroveň; nezávislé na PC
L_5 – úroveň aplikačních programů; animace…

Architektura: pojem není jednoznačně definován ani používán
někdy se chápe jako jeden z bodů postupu: architektura – organizace - implementace
někdy zastřešuje všechny podstatné vlastnosti: funkce, organizace, struktura, realizace

Architektura IBM systému 360 :
architektura je popis chování tak jak ho vidí programátor, a koncepční struktura a funkční chování

Architektura dnes: Návrh prostředků k navrhování PC komponentů, abychom vytvořili PC splňující požadavky
Návrh: často protichůdné požadavky; funkce, propustnost, zpoždění; cena, výkon, spotřeba
váha, velikost, rozšiřitelnost, spolehlivost; dobrý návrh vyžaduje dobře zvolené kompromisy
Architektura je: vše co by měl znát ten, kdo programuje v Assembleru/ tvoří operační systém
tedy: z jakých částí se skládá; význam částí; jak se jednotlivé části ovládají; jak spolu části komunikují

Architektura pohledem programátora (= Instruction Set Architecture)
Formát a reprezentace dat: Způsob uložení dat v PC; mapovací fce mezi světem a vnitřnim uložením
Adresové konvence: Jak se k datům přistupuje (segment-offset, lineární adresace); velikost paměti a její šířka; paměťový model („povolená“ místa X vyhrazená paměť); virtualizace
Instrukční soubor a registrový model: definice přechodové funkce mezi stavy počítače; formát instrukce;
formát zápisu; možnosti adresování operandů
(malá rychlá paměť pro procesor); řídící registr/ registr operandu
explicitní (zapíšu, s kterými)/ implicitní (pevně dané)
Řízení chodu, stavy počítače: spolupráce procesoru a ostatních jednotek; interakce PC s okolim
přerušení vnitřní (zařízeními na stejném čipu jako procesot)/ vnější
Vstupy a výstupy: metody a způsoby přenosu dat mezi procesorem a ostatními jednotkami/ PC a okolím
zahrnuje: definice datových struktur, identifikace zdroje & cíle, identifikace datových cest

Architektura z hlediska vývojáře
Datové / přenosové cesty: data / řídící signály; koncové body; šířka
Sdílení: na úrovni obvodů – sdíl. obvodů procesoru, I/O; na úrovni jednotek – sdíl. ALU více procesory
Specializované jednotky: na různých úrovních; FLOPS; načtení/ chápání/ vykonání instrukce jednotky
Paralelismus: rozklad na úlohy, které lze zpracovávat současně - granularita; lze jen u nezávislých dějů
Programy, části programů, podprogramy, cykly, iterace …, instrukce
Organizace paměti a I/O: hierarchie paměti; způsob vyhledávání a předávání dat
rozdělení funkcí I/O zařízení, počet I/O jednotek, vztahy
Predikce: schopnost připravit se na očekávaný děj - načtení instrukce, nastavení přenosu dat
Realizace: explicitní predikce = předpoklad že se bude vykonávat následující instrukce
statistika = „když se jedná o přenos dat, bude nejspíš následovat i další přenos dat“
heuristiky = předpoklad co dál
adaptivní predikce = změna předpokládání dynamicky

Implementace architektury
Fyzická realizace: logická vrstva (funkční bloky nahrazeny logickými celky); obvodová vrstva (logické obvody na úrovni hradel); fyzická vrstva (na úrovni tranzistorů)

význam technologie: složitost architektury odpovídá stavu technologie: menší, rychlejší, levnější, ekonomičtější

Moorův zákon: Ke změně základních technologických parametrů dochází přibližně po 18ti měsících, kdy se některý z parametrů změní v poměru 1:2

Reprezentace dat

Vícebytové sekvence – jak ukládat a přenášet – po jakých částech, od vyšších/ nižších řádů
Big-endian (rok měsíc den); Little-endian (den měsíc rok); Middle-endian (měsíc den rok…)
NUXI problem – problém s přenosem - propojení big a little-endian - vzniklo připojením UNIXu
Byte-sexual / bi-endian – umí oboje; komunikace – Little-endian, PC- Big-endian
Problémy: portabilita software, přenos dat mezi zařízeními, 8 / 16 bitové architektury, telekomunikace

Číselné soustavy: Polyadické: soustava může mít jeden nebo více základů (radix number system)
Nepolyadické: římské číslice – pro počítání nevhodné
soustava zbytkových tříd: *k*-tice různých základů - prvočísla, číslo *k*-ticí zbytků po dělení

Typy dat: čísla, instrukce, literály (logické hodnoty, znaky, symboly, nečíselná data)
Logické hodnoty: reprezentace jedním bitem – problém s adresací/ celou datovou jednotkou – veliké
Znaky: tisknutelné znaky – reprezentace informace – abeceda, číslice, interpunkce
řídící znaky - ovládání vzdáleného terminálu (CR, LF, FF)
Kódování: zobrazení znaků / číslic na hodnoty v paměti
EBCDIC: 256 závazných znaků; abeceda není v jednom bloku
ASCII: původně pro 7 bitů ~ 128 znaků; malá a velká písmena se liší jediným bitem
rozšíření na 8 bitů + národní znaky - Latin 2; Windows codepage (1250); ISO / IEC 8859 (-2)
Unicode: 1 znak kódován posloupností více bajtů; podporuje (téměř) všechny národní znaky
Diakritika; možnost uživatelských symbolů
Problém se staršími programy; standardizovaná reprezentace (ISO 10646-1, 1993)
Grafické symboly: reprezentace se znaky; jazyky pro popis grafických symbolů
Čísla: různé typy hodnot, různé typy reprezentací; problémy omezené délky prezentace - přetečení

$\mathbb{N}$: převedením do nativní soustavy a přímým uložením

zobrazením BCD - převod po cifrách; snadné zpracování I/O vs. aritmetika & neúsporné

$\mathbb{Z}$: zobrazení se znaménkem - absolutní hodnota čísla + znaménkový bit

problém: dvě reprezentace nuly, složitější operace
zobrazení s posunutím - k číslu se přičte konstanta prezentující nulu
výsledné nezáporné číslo se zobrazí přímo vs. složitější aritmetické operace
zobrazení v jedničkovém doplňku - kladná čísla přímo, záporná inverze bitů absolutní hodnoty
dvě reprezentace nuly
zobrazení v dvojkovém doplňku – jako předchozí, ale záporná čísla + 1

$\mathbb{R}$: čísla s pevnou řádovou čárkou – pevná pozice řádové čárky – jen omezeně – finance…

s plovoucí řádovou čárkou – koeficient, základ, přesnost, mantisa; nerovnoměrná hustota
nejednoznačnost > normalizace – první číslo není nula
Používané zápisy: Znaménko exponentu | znaménko mantisy | exponent | mantisa
Znaménko mantisy | exponent s posunem | mantisa
Znaménko mantisy | exponent | (skryté 1.) mantisa vždy normalizovaná

IEEE standard 754 – formát čísel (speciální hodnoty ±0, ±∞, NaN), operace, zaokrouhlení

Precesion formáty: lze přenášet mezi platformami; posunutí o 2^N-1

single precision: z=2, p=24, exp=8; double precision: z=2, p=53, exp=11

quadruple precision: z=2, p=113, exp=15 - nestandartní

extended formats: nepoužívají skrytý bit, původně pro kalkulačky

single extended: z=2, p≥32, exp≥11; double extended: z=2, p≥64, exp≥15

178,125 → 10110010,001 → $1{,}0110010001\times10^{111}$ → 0 10000110 01100100010…0

exponent	mantisa	Význam
e_{min}-1 0…0	m=0	±0
e_{min}-1	$m\neq0$	$0{,}m\times2^{e_{min}}$
$\langle e_{min}\ ;\ e_{max}\rangle$	m	$1{,}m\times2^{e}$
e_{max}+1 1…1	m=0	±∞
e_{max}+1	$m\neq0$ 1xxx	QnaN
e_{max}+1	$m\neq0$ 0xxx	SnaN

NaN nemusí generovat přímo procesor

Operace	Vznik NaN
+,-	∞+(-∞)
×	0×±∞
/	0/0, ±∞/±∞
REM (mod)	x REM 0, ∞ REM y
√	√x pro x<0

Signální NaN např. hlídání neinicializovaných proměnných; tiché nezpůsobí problém

Pozor na chování konkrétního procesoru a kompilátoru

IEEE standard 854: povoluje z=2 a z=10; neurčuje konkrétní zápis, ani hodnoty p

pro single a double precision zavádí omezující podmínky možných hodnot p

Aritmetické a logické operace

Logické operace: operace s jednotlivými bity reprezentace čísla; kontrola/ nastavení bitu

Základní: unární (NOT), binární (OR, AND); Odvozené (XOR, NOR, NAND, XNOR, …)

Operace posunu: SHL, SHR; ROL, ROR, SAR, SAL

Logický posun: posunutí bitů operandu o *N* míst, na volná místa 0, ~ násobení/ dělení dvěmi

Aritmetický posun vpravo: zleva hodnota nejvyššího bitu, ~ dělení 2 čísla ve dvojkovém doplňku

Rotace: cyklický posun o *N* míst vpravo/ vlevo

Aritmetické operace: sčítání, odečítání, násobení, dělení; porovnávání

INSTRUKCE

Instrukční sada: styčný bod mezi návrhářem počítače a programátorem
v abstraktním smyslu programátor nemá znát fyzický návrh – obvodové řešení
musí ale znát logický návrh, tedy např. registrový a adresový model
musí být: funkčně úplná = dovolit formulovat libovolnou úlohu pro zpracování dat na vyšší úrovni
účinná = často opakované fce by měly být provedeny rychle a za pomoci malého počtu instrukcí
jasně definované požadavky na zdroje
měla by být: ortogonální = definice instrukcí, datových typů a způsobů adresování jsou nezávislé
kompatibilní s existujícím softwarem i hardwarem

Obsah instrukce: Každá instrukce musí obsahovat: operační kód (jaká instrukce má být vykonána); odkaz a určení zdrojových operandů; umístění a určení výsledků; odkaz na další instrukci

Syntaxe instrukční sady: nutná součást návrhu instrukční sady; pro dekodér musí být co nejjednodušší
určuje části, ze kterých je instrukce složena: operační kód (instrukční kód) + argumenty

Modalita instrukce: adaptace základní instrukce
modalita operačního kódu: směr a vzdálenost rotace; směr load/store; použití operandů (paměť/registry)
modalita operandů: velikost a umístění; způsob adresace; velikost a typ operandů; příprava instrukce…
modalita ochrany (paměti): úroveň procesu; přístupová práva k operandům; právo vykonání instrukce

Operační kód: co má instrukce provést; z návrhu PC plyne, co znamenají jednotlivé části operačního kódu

Zápis výrazu: infixová notace: (A – B) / (C + D * E)
postfixová notace (někdy nazývána „reverzní polská notace“): AB – CDE * + /
snadný výpočet pomocí zásobníku: číslo - uložit na zásobník, operátor - provést operaci
prefixová notace (někdy nazývána „polská notace“): / – AB + C * DE

Dělení instrukcí: podle počtu operandů – bezadresové, jednoadresové, dvouadresové, tříadresové, (čtyřadr.)
Bezadresové instrukce: bez operandů: NOP, RET
operandy dány implicitně - konkrétní operand dán operačním kódem: CLI, TBA
operace nad zásobníkem – zásobníková architektura: všechny operandy jsou na zásobníku, instrukce ze zásobníku vyzvedne potřebný počet, instrukce je vykonána, případný výsledek je uložen zpět na zásobník
Jednoadresové instrukce: zbývající dvě adresy (operandy) dány implicitně
obvyklé pro akumulátorovou architekturu - ADD x znamenalo Acc:= Acc+x
Dvouadresové instrukce: jedna z adres je použita jak pro operand, tak pro výsledek
velmi obvyklé - SUB A,4 znamenalo A := A – 4
Tříadresová instrukce: určeny jak oba operandy, tak umístění výsledku
S rostoucí *v* paměti roste flexibilita PC a dává velké možnosti dobrým kompilátorům
Pro větší délku instrukcí a výsledků délku instrukčního slova se nepoužívá často
podle typu operace: aritmetické operace; logické instrukce; přesuny dat; I/O; řídící instrukce
Aritmetické o.: základní (+ - * /, abs, neg, inc, dec), speciální (FLOP, ln, √), konverze
Logické o.: booleovské (AND…), porovnávání (nastavení příznaků), posuny a rotace
Přesuny dat: typické pro Load/Store architekturu – co chce počítat má v registrech…
Vstup/výstup: přesuny; obsluha (start, test)
Řídící instrukce: skoky (nepodmíněný, podmíněný); volání podprogramu, návrat; řízení

Určení argumentů: nutno jasně definovat při návrhu instrukční sady
Implicitně (parametry dány použitou instrukcí); Explicitně (součástí zápisu instrukce je odkaz na par.)

Způsoby adresace: Immediate – bezprostřední zápis v instrukci, data jsou za běhu kódu konstantní
po načtení instrukce není třeba přistupovat do paměti; velikost operandu je omezená
Direct – v instrukci zapsána adresa operandu; k vykonání je třeba navíc 1 přístup k paměti

rozsah adres limitován velikostí instrukce; adresa operandu je konstantní, data se mohou měnit
Indirect – odkaz do paměti, kde je adresa operandu
k vykonání instrukce potřeba 2 přístupy navíc: načtení adresy operandu, přístup k operandu
Indexed – k adrese je přičten index; adresa indexu a základ - index se může měnit; v programech
Based – adresa tvoří posunutí vzhledem k bázi; adresa základu a index; ochrana paměti, segmentace
Relative – operand je určen relativně vzhledem k programovému čítači
Adresace s použitím registrů: některá ze účastněných adres je z domény registrů, nikoli hlavní paměti
Register addressing: jako direct, ale adresová část určuje registr s operandem
Register indirect: jako indirect, ale adresová část určuje registr, který obsahuje adresu operandu

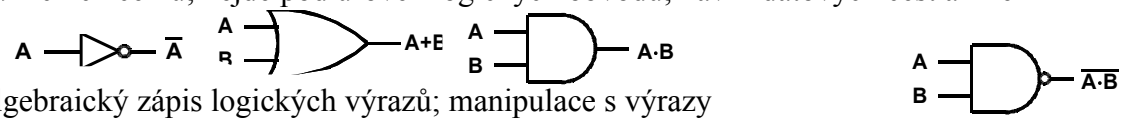
Automatické změny argumentů: např. pre/ post (před/ po vykonání), increment/ decrement (přičtení/ odečtení)
automatický posun na další zpracovávaná data je součástí logiky konstrukce
typické pro přesuny dat a konverze; typické v kombinaci s nepřímým adresováním

Přístupy možno kombinovat

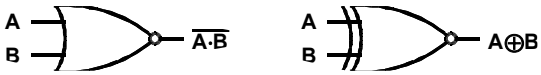
Mikroarchitektura

Architektura z pohledu návrháře: návrh a uspořádání funkčních bloků realizující konkrétní ISA
kompozice funkčních celků; nejde pod úroveň logických obvodů; návrh datových cest a řízení

Číslicové systémy

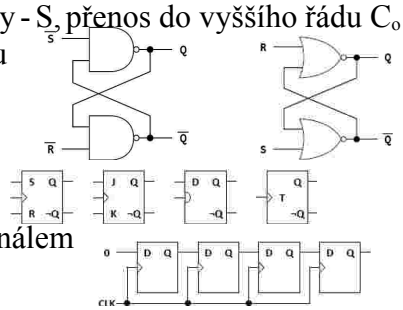


Booleova algebra: algebraický zápis logických výrazů; manipulace s výrazy
Logické obvody: realizace logických funkcí; hradla realizující základní funkce (NOT, OR, AND)
Odvozené funkce: hradla realizující funkce NOR, NAND, XOR



Praktická realizace: stačí NOT + 1 základní operace
NAND / NOR zahrnují negaci
pouze hradla NAND / NOR – závisí na technologii výroby
Kombinační obvody: skupiny hradel; funkce logických proměnných; výstup závisí pouze na aktuálním vstupu
Jednoduché funkční bloky: multiplexory, demultiplexory; kodéry, dekodéry
Složitější funkční bloky: části ALU
Sekvenční obvody: Kombinační obvody + paměťové prvky; paměťové prvky určují stav
Vstup a aktuální stav určuje výstup a následující stav
Synchronní (pomocí hodinového signálu, který mění periodicky hodnotu)/ Asynchronní

Poloviční sčítačka: součet dvou 1bitových čísel; vstupy - operand A, operand B; 2 výstupy - součet S, přenos C
Úplná sčítačka: součet tří čísel: 3 vstupy - A, B, přenos z nižšího řádu C_i; 2 výstupy - S, přenos do vyššího řádu C_o
Klopný obvod typu R-S (latch) – umožňuje si zapamatovat hodnotu jednoho bitu
Složitější klopné obvody: R-S s hodinovým vstupem (clocked R-S latch)
R-S master / slave (R-S flip-flop)
J-K master/slave (umí invertovat vlastní stav)
Odvozené obvody a označení : D latch, D flip-flop, T flip-flop
n-bitový registr: blok klopných obvodů typu D řízených stejným hodinovým signálem
- vstupy: data $d_{n-1} \ldots d_0$, hodiny CLK; výstupy: data $q_{n-1} \ldots q_0$



ALU: Vstupy – operandy, operace - sčítání, odečítání, násobení, dělení, …, porovnání
Výstupy - příznaky (přenos, nulový výsledek, …), výsledek

Datová cesta: uspořádání funkčních bloků umožňující zpracovávat instrukce a data uvnitř procesoru
Návrh datové cesty: vychází z cílové instrukční sady; identifikace prvků; propojení prvků; návrh řízení

Procesor MIPS

Registry – 32 obecných registrů
Operace: Aritmetika: co nejjednodušší: registr-registr, registr-přímá hodnota
Načítání, ukládání dat
Přesuny: registr-registr, registr-paměť
Skoky: nepodmíněné, do podprogramu & návrat
Speciální instrukce, které vyplynou v průběhu tvorby
Základní typy instrukcí: základní formát 32-bitů: | 6b | 5b | 5b | 5b | 5b | 6b |
r-formát: aritmetické instrukce s registry: | op | rs | rt | rd | shamt | funct |
i-formát: přesuny, větvení, přímé odkazy: | op | rs | rt | address / immediate |
j-formát: nepodmíněný skok: | op | target address |
Obecný postup zpracování instrukcí: přečíst instrukci, načíst registry, provést operaci (čtení/zápis/operace /skok)
Řízení datové cesty: Řídící signály - programový čítač, registrové pole, paměť pro data, ALU
Podporované instrukce: load / store word, branch equal; add / substract, logical and / or, set on less than
Kombinační obvod: nastavuje řídící signály datové cesty; dekodér z instrukčního kódu
Formát instrukčního kódu: významně ovlivňuje složitost a rychlost dekodéru
Víceúrovňové dekódování: rozdělení řídícího obvodu do více bloků - zjednodušení řízení ALU ze 4 bitů na 2

Návrh hlavního řadiče
Jednocyklový řadič: instrukce trvá 1 takt; kombinační obvod
Hlavní nevýhody: délka cyklu odpovídá délce nejdelší instrukce - spor s „optimize for common case“
duplicitní prvky v datové cestě
Vícecyklový řadič: instrukce rozdělena do kroků (různý počet), v každém taktu proveden 1 krok
Vyšší výkon a efektivita, není třeba duplikovat některé prvky vs. nutno uschovávat mezivýsledky
Rozdělení instrukcí do kroků - sekvenční a paralelní části vykonávání instrukce
Instrukční cyklus: načtení; dekódování a přetečení registrů, výpočet adresy podmíněného skoku; vykonávání, výpočet adresy, dokončení větvení; přístup do paměti a zapsání výsledku; dokončení čtení z paměti

Řadič

Instrukční cyklus: načíst instrukci, vykonat instrukci, uložit výsledek → fragmentace na menší části
výpočet instrukční adresy
| zjištění typu instrukce
| dekódování instrukce
| výpočet adresy operandů
| |--vyhledání operandů – je-li více operandů
| | vlastní operace
| | výpočet adresy výsledku
| | ---uložení výsledku – je-li víc výsledků
|---------|-------------kontrola přerušení – v případě že není přerušení: hotovo/ návrat pro data
---------------------------přerušení – hotovo/ návrat pro data
Základní schéma řadiče: direktivní/ zpětnovazební

Návrh řadiče
Hard-wired: Obvodová realizace - přechody stavovým diagramem, standardní metody sekvenční logiky
Méně komponent/ pro změnu potřeba změnit celé
Mikroprogramování: Stavový diagram jako přechodová funkce v paměti, vyzvedávání sekvencí z paměti
Převedení problému o úroveň niž – jak provádět mikroinstrukce
Horizontální formát: mikroinstrukce obsahuje přímo hodnoty řídících signálů
Není třeba dekódovat → rychlost; libovolná kombinace → pružnost; objemné
Vertikální: některé kombinace se vylučují navzájem → možno zakódovat → menší objem
Nutno dekódovat → zpomalení, zesložitění, pevný návrh → méně pružné

Load (ACC⇐mem[adr]); add (ACC⇐ACC+mem[adr]); store (mem[adr] ⇐ACC); brz (if (ACC=0) PC⇐adr)

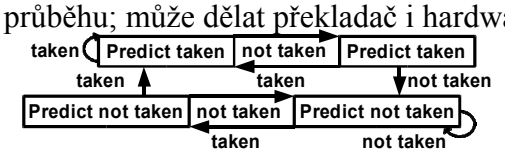
Nanoprogramování: Na místě instrukce index do pamětí → silná redukce velikosti / zpomalení
Vertikální formát pro indexaci druhé úrovně; horizontální pro nanoprogram ve druhé úrovní

Zvyšování výkonu procesoru

Pipeline = linka proudového zpracování; současné vykonávání nezávislých dějů

k kroků, n instrukcí, zrychlení: $s=n \cdot k/(k+n-1)$; pro $n \gg k$: $s \rightarrow k$

Problémy: Bez ohledu na to, co se děje uvnitř se musí procesor navenek tvářit striktně sekvenčně
Strukturální hazard: Hardware nepodporuje danou kombinaci instrukcí
Současný přístup k prostředku z více stupňů pipeline
Datový hazard: Instrukce nemá k dispozici data pro vykonání - čeká se na výsledek předchozí instrukce
Čtení dat z externí paměti způsobí zpoždění ve fázi fetch; r3←r2+r1, r5←r4-r3…
Odstranění vložením *neškodných* instrukcí (nepracují s ničím, na čem závisí výpočty), přerovnání
Řídící hazard: nutno učinit rozhodnutí před vykonáním instrukce (podmíněný skok)
Řešení stejným způsobem jako problém se závislostí
Skok si vynutí vyprázdnění pipeline
Řešení: Multiple streams: více pipeline, jedna hlavní, další počítá větve a v případě skoku se překopíruje
Look ahead / look behind buffer: zapamatuje si, jak to bylo minule
Delayed branch: zpoždění skoku, pár instrukcí za skok, provedou se než se skok vyhodnotí
Branch prediction: predikce skoku, v instrukci uložena *nápověda*. Podle té se instrukce načítají.
Je-li správně, nedojde ke ztrátě. Je-li špatně, vyprázdní se piperine a načte se správný obsah.
statická predikce: rozhodnutí bez znalosti skutečné historie průběhu; může dělat překladač i hardware
cykly obvykle skáčí na začátek, jen 1x neskočí
dynamická predikce: mění se dle předchozích stavů



Vývoj Pipeline
SuperPipelining: rozdělení kroků pipeline na menší → jednodušší → rychlejší; problém - přenos dat je pomalý
Superscalar pipelining: více paralelních částí - různé cesty pro integer a float-point; někdy i více celých pipeline
Dynamic piperine: přeplánování problematických částí: fetch and decode unit (rozplánování), functional unit (5-10; čekání a provedení), commit unit (čekání na ostatní části)

Druhy zrychlování
Simultánní multithreading: do pipeline se zavádějí instrukce různých vláken - nebudou na sobě skoro jistě závislé
Predikce hodnoty: pozorování ukazuje, že instrukce Load zavede ve více než ½ případů stejnou hodnotu → budeme to předpokládat

Zvyšování výkonnosti

Statistika užívání instrukcí (IBM / 360):

Přesuny dat	45,28%
Řízení	28,73%
Aritmetika	10,75%
Porovnávání	5,92%
Logické operace	3,91%
Posuny, rotace	2,93%
Bitové operace	2,05%
I/O a ostatní	0,43%

Další pozorování: 56% konstant je v rozsahu ±15 (5 bitů)
98% konstant je v rozsahu ±511 (10 bitů)
95% podprogramů potřebuje pro předání parametrů méně než 24 bytů
V typickém programu bylo použito 85% typických instrukcí
Pro 98% instrukcí stačilo 15% firmware

Instrukční sady
Do 80. let trend budovat rozsáhlé instrukční sady, složité instrukce, překlenutí mezery mezi assemblerem a …
Důraz na to nejpotřebnější a nejdéle trvající; spolehnout se na vyšší jazyky a optimalizující kompilátory
Snaha o jednoduché instrukce → rychlé provádění; rychlejší paměť → složité instrukce nejsou klíčové

MIPS: 80. léta, John Hennessy – Stanford University – silný v kompilátorech → vývoj procesoru, jehož architektura by vyjadřovala snížení kompilátoru na úroveň HW (místo povyšování HW na úroveň SW)

CISC vs. RISC: Complex Instruction Set Computer (zpětné označení) vs. Reduced Instruction Set Computer

Proč se CISC vlastně vyvinuly: První stroje měly jednoduchou architekturu
Cena HW klesla, cena SW stoupla
Přesun složitosti do HW usnadní programování
Méně instrukcí pro daný úkol = méně přístupů do (pomalé a drahé) paměti
Implementace pomocí mikrokódu se dá snadno změnit

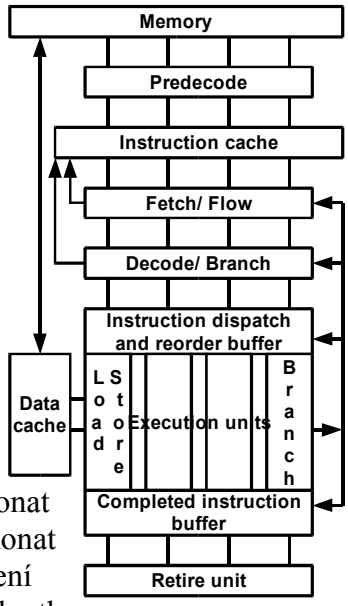
Strategie pro návrh RISCu: Zjištění nejpoužívanějších instrukcí (simulace a analýza programů)
Optimalizace datových cest pro tyto instrukce
Přidání dalších instrukcí, pokud jsou rozumně využitelné a nezpomalí procesor
Přesunutí komplexních činností do kompilátoru
Charakteristické rysy pro RISC: Jedna instrukce na cyklus (skok dvě – vyprázdnění pipeline)
Operace registr-registr (nečekání na paměť)
Architektura load-store
Malý počet a jednoduché adresovací režimy a instrukce
Pevný formát instrukce
Velké množství registrů (levná paměť; ukládání mezivýsledků)
Využití pipeline
Zvláštní zpracování skoků
Hard-wired návrh obvodového řešení
Silná závislost na překladači

Návrh procesoru: Pevná / proměnná délka instrukcí
Pevný / proměnný počet a typ operandů
Počet adresovacích režimů
Mikrokód / hard-wired řadič
Stupeň paralelismu
Důraz na vyšší programovací jazyky
Schopnosti kompilátoru
Zvolit RISC nebo CISC: Neexistuje jednoznačná odpověď
Kvantitativní přístup : porovnat velikost programů a rychlostí provádění
Kvalitativní přístup : vyhodnotit podporu vyšších programovacích jazyků
Problémy: Žádné dva procesory RISC a CISC nejsou přímo porovnatelné
Není konečná sada testovacích programů
Obtížné odlišit HW a vliv kompilátoru

Od 80-let se technologie i znalosti změnily: Mnohé techniky jsou dnes používány procesory z obou táborů :

CISC: Schopnost vykonávat v 1 taktu více instrukcí/
RISC – se zlepšením technologií zbývá místo, vyplnění komplikovanějšími instrukcemi
Vznik *post-RISC* návrhů, kombinující oba přístupy s metodami, které nejsou použity v žádné z těchto kategorií

Post-RISC: Rozdíl od superskalárních RISCů:
Přidání ne-RISCových instrukcí (pro zvýšení výkonu) → architektura FISC (Fast Instruction Set Comp.)
Agresivní přerovnávání instrukcí v průběhu zpracování - „out-of-order execution" „speculative execution"; odklon od závislosti na kompilátoru
Nové uspořádání, nové jednotky; větší míra paralelismu
Nové komponenty: predecode unit; renaming register; reorder buffer- retire unit



Predecode unit: Instrukce načítány po blocích
Částečné dekódování zjistí vlastnosti a uloží je do vyrovnávací paměti
Identifikace skoků, typ potřebné exekuční jednotky, potřeba paměti,
I-cache: Ve vyrovnávací paměti: blok instrukcí, historie skoku, predekódované příznaky, historie vykonávání skoku (u cyklů)
Fetch/Flow: Načítání z I-cache musí být chytré
Vyhodnocování historie, záznam v I-cache ukazuje na další blok instrukcí
Ukáže-li se odkaz jako špatný, je změněn
Decode/Branch: Vlastní dekódování instrukce – pro jakou jednotku
Rozdělení na dva proudy do doby, než je znám skutečný cíl skoku
Barvení instrukcí – příznak větví u větvení
Výsledky z Branch Unit: Fetch/Flow: aktualizace flow history, načtení správných I
Branch/decode unit: aktualizace branch history
Dispatch and Reorder buffer: zahození I, které se neměly vykonat
Completed instruction buffer: zahození I, které se neměly vykonat
Instruction Dispatch and Reorder Buffer: Dekódované instrukce „čekají" na provedení
K provedení dojde, jsou-li k dispozici: vstupní hodnoty, výstup, exekuční jednotka
Přednost mají: starší instrukce, instrukce load
Datové závislosti při přerovnání – Read After Write, WAW, WAR; WAW a WAR lze řešit přeznačením
Execution units: podobně jako u RISC - jednotky, které nepočítají v 1 cyklu, používají pipeline
Jednotky load/store obvykle umožňují rozpracovat více přístupů najednou
Completed Instruction Buffer: Ukládání vykonaných instrukcí, spolu s příznakem vygenerované výjimky a mapováním (přejmenováním) použitých registrů
Retire unit: Odebírá instrukce z bufferu v takovém pořadí, v jakém by byly sekvenčně zpracovány
Může *uvolnit* v 1 taktu více instrukcí

Problém – výjimky: *Precizní přerušení*: Při výskytu výjimky musí procesor „zastavit činnost" na tomto místě
Instrukce, které jsou po této instrukci, nesmí ovlivnit stav stroje
Nesmí existovat nezpracované instrukce ležící před touto instrukcí
Všechny výjimky způsobené předchozími instrukcemi jsou vyřízeny

RISC vs. post-RISC: výkon je dán stupněm paralelismu vs. počtem najednou odkládaných instrukcí

Přerušení

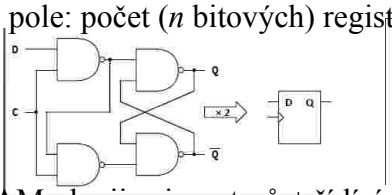
Příčiny: vnější (exception): Význačné stavy I/O zařízení, význačné stavy čítačů, zásahy uživatele, porucha HW
vnitřní (interrupt): Neznámá instrukce, speciální instrukce (INT, SYSCALL), výpadek (stránky/segmentu)
Průběh přerušení: vznik, vyslání žádosti; rozhodnutí o přijetí/nepřijetí (+maskování); identifikace zdroje; určení adresy obslužného programu; úschova aktuálního stavu CPU; provedení obslužného programu; obnova stavu CPU; návrat do přerušeného programu
Přijetí přerušení: vnitřní ihned, vnější jen mezi instrukcemi (nutno zachovat definovaný stav CPU)
Výběr mezi žadateli o přerušení:
Zřetězená zařízení: „společná žádost“, chytrá zařízení zřetězena a přerušení zastaví první žadatel
Malé nároky na CPU a program/ priorita dána zapojením, složitější zařízení
Řadič přerušení: hloupá jednodušší zařízení, pružné
Identifikace zdroje: Čistě programová - jediný obslužný program, zjistí si sám (pružné, ale pomalé)
S pomocí technických prostředků: žadatel poskytne vektor přerušení, podle něj se rozhodne o obslužném programu: adresa obslužného programu; index do tabulky adres; strojová instrukce

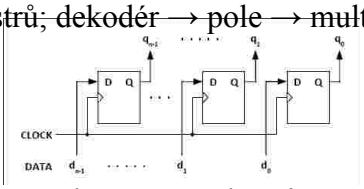
Paměťový subsystém

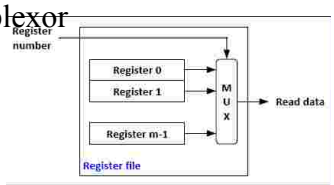
Parametry paměti: Kapacita - objem informace, který je možno uchovat v jedné paměťové jednotce (*slova*/ *byty*)
Přenosová rychlost - rychlost, kterou lze data přenášet do/z paměti (špičková vs. zaručená)
Vybavovací doba - čas, za který je paměť schopna vyřídit požadavek
Velikost slova - velikost adresovatelné jednotky paměti (obvykle počet bitů na slovo)
Přenosová jednotka - počet datových elementů, který je možno převést v 1 kroku
(obvykle počet bitů (hl. paměť) nebo bloků (sek. paměť))
Cyklus paměti - doba mezi dvěma bezprostředně za sebou jdoucími požadavky
Přístupová metoda

Dělení pamětí: fce, způsob přístupu, technologie, umístění v systému, vnitřní organizace, detekce/oprava chyb...
Dělení pamětí dle funkce: ROM – read only memory
Mask ROM – „jediná pravá ROM“, obsah z výroby
PROM – programovatelná ROM
EPROM – programovatelná ROM smazatelná UV světlem
EEPROM – elektronicky smazatelná EPROM
FLASH – ???
RWM – read-write memory
DRAM – dynamická RAM – pro udržení dat není třeba je obnovovat
SRAM – statická RAM – potřeba obnovovat data
Speciální: IRAM (Inteligent RAM) – paměť + CPU
CRAM (Crypting RAM)
WORM – write once, read many
Dělení podle způsobu přístupu: RAM – Random Access Memory: všechna paměťová místa mají svou adresu
k paměťovým místům může být přistupováno v libovolném pořadí
doba přístupu nezáleží na předchozí adrese, je konstantní
SAM – Sequential Access Memory: paměťová místa nemusejí míst svou adresu
přístup sekvenční
doba přístupu je závislá na vzdálenosti od počátku
DAM – Direct Access Memory: paměťová místa mají jednoznačné adresy
Kombinace předchozích dvou
AAM – Associative Access Memory
CAM – Contents-Addressable M.: přístup k datům podle (části) obsahu (ne adresy)
obvykle paralelní prohledávání
Technologie pamětí: Pre-elektronické: relé, zpožďovací linky, ferritová pole
Magnetické: bubny, pásky, disky
Elektronické: RAM, FLASH, …
Optické: CD, DVD, MD
(chemické, biologické, …)

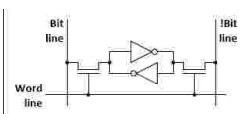
Statická pamět: Ráchytné a klopné obvody: klopný obvod typu D ~ 9 hradel ~ 18 tranzistorů
Registry: šířka slova (*n* bitů)
Registrové pole: počet (*n* bitových) registrů; ~~dekodér → pole → multiplexor~~

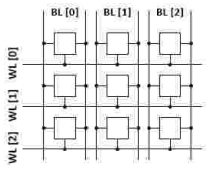


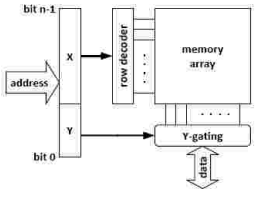




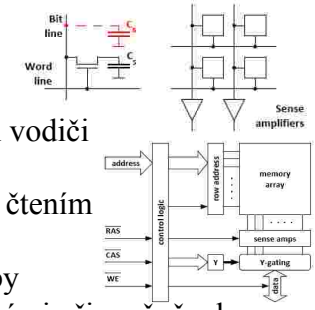
Buňka paměti SRAM: dvojice invertorů + řídící tranzistory; tranzistorů na 1 buňku
SRAM v maticovém uspořádání: vysoká hustota
Adresace: výběr řádku, čtení sloupců
Kapacita: výška (počet slov) * šířka (v bitech)







Dynamická paměť: Destruktivní čtení, limitovaná doba uložení



Malý počet součástek - kondenzátor + řídící tranzistoru
Nabití – na bitový vodič hodnota, tiknu adresovým, pokud 11, nabije se
Čtení – bitový vodič na polovinu, otevřu tranzistor a sledují změnu na bitovém vodiči
RAS – řádek; CAS – řádek; WE – zda čtu
Zvyšování výkonu DRAM: Nibble Mode access: 4 po sobě jdoucí data jedním čtením
Page Mode access: signál RAS podržen, signál CAS měněny
Fast Page Mode access: řádek není držen celou dobu → snížení spotřeby
HyperPage Mode – Extended DataOut: datový výstup může být zachován i při změně adresy
Burst EDO: čtení / zápis formováno po čtyřech → interní změna dvou bitů adresy místo celé
Video DRAM = VRAM: na výstupu posuvný registr, kam je zkopírován vystavený řádek

Časování: Asynchronní – pro dokončení operace je třeba časové kvantum
Synchronní – operace zcela řízena jednotným hodinovým signálem
eliminace dodatečných signálů, jednotné časování, zjednodušení interface

RDRAM vs. SDRAM: rychlé nastavení přístupu (rychle malá data) vs. dlouhé nastavení/ rychle data

Hierarchie pamětí

Vnitřní	Registry	Procesor	Polovodiče na procesoru	B	~1ns
	Cache	L1, L2, L3 cache	Polovodiče na/ mimo procesor	kB	~10ns
	Hlavní paměť	RAM	Polovodiče	MB	~10-100ns
Vnější	Odkládací paměť	Pevný disk	Magnetický záznam	GB	~1-10ms
	Archivní paměť	CD, DVD, pásek	Optický záznam	TB	>100ms

Vyrovnávací paměti – <u>cache</u>

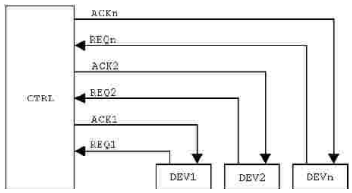
Vyrovnávání výkonových problémů – procesor chce rychle data/ hlavní paměť pomalá
využití lokality přístupu: Zapamatování si dat v blízkosti posledního přístupu (velká šance potřeby)
Organizace cache: data v cache uložena spolu se svou adresou
Vyhledávání dat v cache:
Plně asociativní mapování: při vyhledávání adresy je adresa vyhledávána cache přímo; vysoká cena
Přímé mapování: každý blok má v cache své místo; na jednom místě v cache může být více bloků
Skupinově asociativní mapování: kompromis – v několika oblastech zkoumám určité místo
Uvolňovací mechanismy cache: Je-li potřeba tam něco uložit a není místo – řešení: „něco“ se „vyhodí“
Přímé mapování: určeno jednoznačně
Random: Náhodně
LRU: last recently used
FIFO: first in first out
LFU: last frequently used
Konzistence dat: jak zajistit, aby data v cache odpovídala datům hlavní paměti
Write-through – zápis změny na obě místa zároveň; časté zápisy do hlavní paměti, mezivýsledky zbytečně
Write-back – zápis do hlavní paměti až při uvolnění z cache; problém při používání více procesorů
Write-once – napoprvé WT, později WB

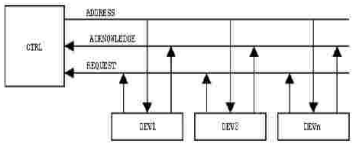
Architektura počítačového systému

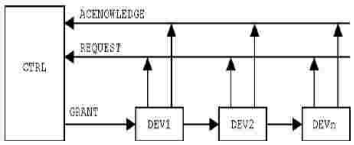
Uspořádání na úrovni systému: Funkční bloky (procesory, paměti; periferie zařízení)
Propojovací systémy (přenos dat v rámci systému a mimo něj)
Podpůrné obvody (přidělování prostředků; přerušení a přímý přístup do paměti)

Propojovací systém
Dvoubodové spoje: Přímé spojení (port), křížový přepínač (crossbar switch), propojovací síť (switch fabric)
Vícebodové spoje: Účastníci sdílejí přenosové médium (broadcast : 1 vysílač, více přijímačů)
Sběrnice = sada sdílených vodičů (adresové / datové vodiče, řídící vodiče)

Sběrnicové systémy: nízká cena a flexibilita; lehké zvládnutí složitosti systému
Potenciální úzké hrdlo v systému; rychlost omezena délkou a počtem zařízení; potřeba spojit různorodá zařízení - různé rychlosti, různé objemy přenášených dat
Princip: vystavení adresy cílového zařízení (adress); potvrzení cíle a výběr typu přenosu (control); přenos (data)
Fyzická charakteristika sběrnice: vodiče, napěťové úrovně, frekvence hodin, …
Logická charakteristika sběrnice: sběrnicový protokol
Sběrnicový protokol: Definuje povolený průběh transakce na sběrnici – logický význam signálů na vodičích - popis pomocí stavových a časových diagramů
Synchronní/ asynchronní protokol: přechody mezi fázemi transakce určeny hodinovým signálem/ signál
Centralizované/ distribuované přidělování: o přidělení sběrnice rozhoduje centrální arbitr/ dle protokolu
Přidělování sběrnice: Centrální přidělování sběrnice: Fyzické uspořádání: Samostatné žádosti, cyklické výzvy, prioritní zřetězení
Řešení současných požadavků: Náhodné,dle pořadí vzniku,prioritní







Distribuované přidělovací sběrnice: Kolizní (CSMA / CD): Poslouchám a když je volno začnu vysílat
Začnou-li dva zároveň, na náhodně dlouho se odmlčí
Token bus: Předávají si právo vysílat „právo vysílat“ (nechce-li, pošle dál
Adresová priorita: Chce-li jich vysílat víc, vysílá ten s vyšší prioritou

Ovládání periferních zařízení
Periferie ovládané programem: Vykonání specifické operace: Výběr zařízení, zápis příkazů specifických pro zařízení, čtení stavové informace specifické pro zařízení
Rozhraní paměť / procesor / periferie: Sdílený adresový prostor zařízení a paměti
Zařízení mapována do paměti (memory-mapped I/O)
Přístup pomocí běžných instrukcí typu load/store
Oddělený adresový prostor zařízení a paměti
Přístup pomocí speciálních instrukcí typu in/out
Informace o stavu zařízení: Polling: Periodická kontrola stavu zařízení - značná režie u pomalých zařízení
Interrupt-driven I/O: zařízení signalizuje procesoru změnu stavu, obslužná rutina přerušení reaguje
Přenos dat z/do zařízení (z/do paměti): S účastí procesoru/ bez účasti procesoru
Podpora interrupt-driven I/O: Procesor: Signalizace požadavků na přerušení: Jeden nebo více signálů
Identifikace zdroje přerušení: Vlastní vodič/ jinak
Výběr obslužné rutiny přerušení: Statická adresa vs. vektor přerušení
Systém: Řadič přerušení: Více přerušovacích signálů, vyhodnocení priority

Přenos dat po sběrnici
Za účasti procesoru: Rychlý přenos, vytěžuje hlavní procesor
Přenos iniciován při změně stavu zařízení: Polling, interrupt-driven I/O
Přenos realizován cyklem v programu: Přečíst/ zapsat slovo z/ do periferie/ paměti; „programmed I/O“
Bez účasti procesoru: Přenos iniciován při změně stavu zařízení

Přenos realizován řadičem nebo zařízením – program nastaví pouze parametry přenosu
Přenos zařízení / paměť: Direct memory access; řadič DMA, bus mastering
Přenos zařízení / zařízení: Bus mastering
Používané metody: Dávkový režim: Procesor se s řadičem domluví o používání sběrnice
Kradení cyklů: Řadič „uspí“ procesor
Transparentní režim: Řadič pozná, kdy procesor sběrnici nepoužívá

Řadič DMA: Chová se jako periferie, program nastavuje parametry, řadič je aktivuje
Přenos dat po sběrnici: Program nastaví řadič a periferii a spustí přenos; na začátku cyklu sběrnice řadič signalizuje HOLD; CPU testuje HOLD na začátku strojového cyklu, nepotřebuje-li sběrnici, signalizuje HLDA a uvolní ji; po přijetí HLDA řadič přenese slovo z/do paměti; řadič uvolní HOLD; procesor uvolní HLDA
Transparentní režim přenosu s pomocí DMA: Přenos probíhá nezávisle na procesoru
Sběrnice má 2 zdroje hodinového signálu, vzájemně fázově posunuté
Použije se náběžná i sestupná hrana hodinového signálu
Bus mastering: Sběrnici může řídit libovolný účastník - nutno zažádat o přidělení sběrnice
Procesor v roli normálního účastníka
Přenos dat mezi libovolnými účastníky; stále je nutné přenos nastavit z programu
Dávkový režim přenosu (burst mode): 1 adresový cyklus na blok dat
Přenos nesouvislých bloků (scatter / gather)
Obecné poznámky k DMA: přenos pomocí DMA nemusí nutně být rychlejší než PIO
Umožňuje procesoru dělat něco užitečnějšího
Systém musí zajistit platnost dat v cache: Write-back cache může mít novější data než paměť
V paměti mohou být novější data než v cache; procesor sleduje provoz z/do paměti

Operační systémy - úvod

Vývoj: 1. generace – žádný OS, konstruktér stroj spravuje i programuje
2. generace – dávkové systémy, sekvenční spouštění programů
3. generace – multitasking (zpracování více procesu na jednom procesoru), rozdělení rolí součástek
4. generace – cokoli (prostý zavaděč vs. distribuovaný, víceuživatelský, víceprogramový OS)

Hlavní role moderního operačního systému: Extended machina – program nemusí přesně vědět, na čem běží
Resource manager: Přiděluje prostředky; izolace aplikací, sdílení prostředků)

Hlavní koncepce

Monolytické systémy: Důraz na efektivitu (před robustností)
Subsystémy mají stejné oprávnění: nízká režie komunikace
Uživatelské procesy mají minimální oprávnění - systémové volání mění oprávnění
Ač nejstarší, běžně používaný - CP/M, DOS, UNIX, Windows, …

Virtuální stroje: Možnost běhu více systémů na jednom HW;
Měla být maximální abstrakce HW, dnes abstraktní stroj bez závislosti na HW ???
Úplná virtualizace, paravirtualizace (něco dělá HW)

Mikrojádrové systémy: Důraz na robustnost
Jádro co nejmenší - v jádře obsluha přerušení, komunikace; ostatní služby na uživatelské úrovni
Architektura klient-server; především speciální a experimentální systémy

Procesy

Proces: Spuštěný program - adresový prostor, prostředky, práva, signály; „kontext“
Vlákno / thread: „místo v procesoru, kde se vykonává program“ - IC, registry CPU, zásobník

Podpora vláken v OS: Původně jen procesy: Spouštění procesu je „drahé“, obtížnější komunikace

⇒ zavedení vláken na úrovni userspace (knihovny), v jádře, hybridní

Stavy procesu/vláken: Běžící: Běží li moc dlouho, přechází do stavu připraven; zde se též ukončuje a blokuje
Blokované
Připravené: Sem přichází nově vytvořené a odblokované

Plánování: Entita plánování (proces/vlákno), přidělování CPU entitám; plánovač určuje, co bude probíhat dle…
Cíle plánování: Spravedlnost (dostane se na všechny)
Efektivita (využití CPU tak, aby nevykonával zbytečnou práci)
Doba odezvy (pro interaktivní procesy)
Průchodnost (počet procesů ukončených v čase; hledí se na potřeby celého systému)
Co nejnižší režie (aby při přepínání procesů nedocházelo k velkým ztrátám)
Kritéria plánování – každý proces má vlastní, vázanost procesu na prostředky, typ procesu, výpadky, čas

Kategorie plánování
Nepreemptivní plánování: Proces běží, dokud uzná za vhodné; občas OS přeplánuje proces, pokud je to nutné, děje se tehdy, pokud proces žádá OS o nějakou službu
Preemptivní plánování: OS má kontrolu a je schopen vynutit si odebrání CPU

Plánovací algoritmy
FCFS (= first comes first served): „kdo dřív přijde, ten dřív mele“
SJF (= shortest job first): Kratší úlohy plánovány přednostně
Round Robin: Proces může být přeplánován – pracuje-li dlouho, není-li něco k dispozici, výpadek stránky
Více front se zpětnou vazbou: Při každém zpracování se proces propadá o úroveň níž, zpracovává se od nejvyšší úrovně, pokaždé když je proces zablokován dostává se o úroveň výš

Interakce mezi procesy: Přístup ke sdíleným prostředkům (i v rámci jednoho procesu): data, jednotky
Komunikace (mají-li společný cíl): Paralelní / distribuované aplikace; mezi vlákny/ procesy
Zablokování: na sdílených cestách

Implicitní sdílení dat: Synchronní události: Služby poskytované procesorům (procesy volají služby jádra)
Asynchronní události: Reakce na vnější přerušení (vstup dat z klávesnice, příjem dat…)
Modifikace dat v reakci na události: Kontext přerušení, systémové volání
Reentratní operační systém: Podpora souběžného zpracování více událostí
Explicitní sdílení dat: Souběžný přistup k datům (v rámci různých vláken jednoho procesu)
Důsledky chyb obvykle méně fatální

Problémy přístupu ke sdíleným prostředkům/ datům: operace nejsou atomické
Při souběžném přístupu může dojít k přerušení nebo přeplánování v nekonzistentním stavu
Výsledek může záviset na pořadí provádění operací
Pojmy: Race condition („chyba souběhu“): Výsledek operace závisí na konkrétním plánování (nenápadné)
Critical section („kritická sekce“): Část programu, kde se provádí kritická operace
Mutual exclusion („vzájemné vyloučení“): Kritickou operaci provádí nejvýše jeden proces
Vypořádání se: 1. žádné dva procesy nemohou být najednou v jejich kritické sekci

2. nemohou být kladeny žádné předpoklady o rychlosti nebo počtu CPU
3. žádný proces mimo kritickou sekci nesmí blokovat jiný proces
4. žádný proces nesmí čekat nekonečně dlouho v kritické sekci

Metody dosažení vzájemného vyloučení

Aktivní čekání: Spotřebovává čas procesoru, nespotřebovává prostředky OS, rychlejší, krátké čekání
Pasivní čekání/ blokování: Proces čeká ve stavu „blokován“, spotřebovává prostředky OS, pomalejší

Aktivní čekání: při uzamčeném zámku nedělá procesor nic užitečného

Zakázání přerušení: Vhodné pro jádro OS
Zámky: Přidání sdílené proměnné pro řízení přístupu do kritické sekce – nefunguje
Důsledné střídání: Porušuje podmínku 3; předává se v proměnné, kdo je nyní na řadě
Petersonovo řešení: Nepoužívá se pro více než 2 procesy
Pamatuji si, kdo má zájem, jedna fce koukne, zda je vše OK, kouknu, kdo může, 2. Fce oznámí konec
Instrukce TSL = spin-lock: nutná HW podpora, atomicky testuje a nastavuje zámek
Načti zámek do registru R a nastav zámek na 1, byl-li nenulový znovu, jinak …, ulož do zámku 0

Pasivní čekání: Procesor může při uzamčení zámku dělat něco užitečného; nutná podpora OS
SLEEP/ WAKEUP: SLEEP uspí proces, který ho zavolá, WAKEUP probudí uspaný proces
Nefunguje - test a uspání musí být atomické; k frontě je potřeba zámek, který se při SLEEP uvolní
Semafory: Implementovány OS; atomické operace UP, DOWN; čítač a fronta uspaných procesů
Dvě sémantiky chování podle možných hodnot čítače: čítač ≥ 0/ čítač v rozsahu celých čísel (i záporná)
DOWN: je-li čítač >0, sníží čítač o 1 a pokračuje dál; jinak zablokuje DOWN a proces do fronty
UP: se zvětší hodnota čítače o 1; je-li hodnota ≤ 0 a fronta neprázdná, odblokuje libovolný proces
Monitory: Implementovány překladačem – využívá semafory OS; vstoupit může víc ale jen jeden je aktivní
Podmíněné proměnné monitoru: slouží k zablokování uvnitř procesu
WAIT: Zablokuje proces uvnitř monitoru a umožní vniknout dovnitř jinému procesu
SIGNAL: Vybere libovolný z čekajících procesů a ten probudí
Problém s probuzením: Budící se musí uspat/ okamžitě opustit monitor
Zprávy: Atomické operace SEND a RECEIVE
SEND: Odešle zprávu a nezablokuje se; pokud na ní někdo čekal, odblokuje se
RECEIVE: Přijímá zprávy, pokud není žádná zpráva dostupná, zablokuje se
Adresace: Pomocí identifikace procesu - je možné pouze komunikace mezi dvěma procesy
Schránky (mailbox): Mají svojí velikost a identifikaci; možno k více procesům
Dostaveníčko(randezvous):Schránka nulové velikosti;po předání zprávy oba procesy naráz odblokovány

Ekvivalence primitiv: pomocí jednoho blokovacího primitiva lze implementovat jiné blokovací primitivum

Klasické synchronizační problémy

Producent-konzument: Producent produkuje, konzument spotřebovává, mezi nimi je sklad pevné velikosti
Konzument přestane prodávat když je sklad prázdný; továrna přestane produkovat když je sklad plný
Obědvající filozofové: 5 filosofů sedí okolo kulatého stolu, každý filosof má talíř špaget a vedle něj 1 vidličku
Když má filozof hlad, vezme dvě vidličky a jí; problém pokud všichni chtějí jíst současně
Ospalý holič: Holič má ve své oficírně křeslo na holení zákazníka a pevný počet sedaček pro čekající zákazníky
Pokud v oficírně nikdo není, holič se posadí a spí
Pokud přijde první zákazník a holič spí, probudí se a posadí si zákazníka do křesla
Pokud přijde zákazník a holič už stříhá a je volné místo v čekárně, posadí se, jinak odejde

Reálně používaná primitiva

Spin-lock: Aktivní čekání; vhodný pro očekávané krátké doby čekání; velmi rychlý vstup do kritické sekce
Kritická sekce: Blokovací primitivum; vhodný pro očekávané delší doby uvnitř kritické sekce
Semafory: Spouštění a zastavování vláken, čekání na události

Synchronizační primitiva – vyšší úroveň

RWL (Read-Write Lock): Vícenásobně/paralelně READ operace, exkluzivně WRITE operace (mění obsah)
Bariéry: Hlavní vlákno vypustí pomocná vlákna
Reentrantní zámky: Několik funkcí, které používají stejnou kritickou sekci; složité
Uzamčené (interlocked) operace: Atomicky prováděné jednoduché operace (přičtení, inkrement,…)

Prostředek: Cokoliv, k čemu je potřeba hlídat přístup; odnímatelné vs. neodnímatelné (odejmutí bez následků)
Požadavky na využití neodnímatelných prostředků mohou způsobit zablokování
Práce s prostředky: Žádost – blokující; výlučná práce s prostředkem

Coffmanovy podmínky zablokování
Vzájemné vyloučení (Exclusive use): Prostředek vlastní nejvýše jeden proces
Drž a čekej (Hold and wait): Proces může vlastnit prostředek a žádat o další
Neodnímatelnost: Přidělené prostředky nemohou být odebrány
Čekání do kruhu (Cyclic dependancy): Procesy čekají na prostředky v kruhu

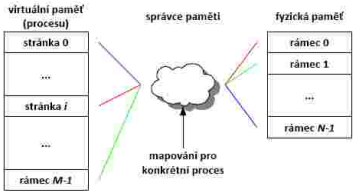
Řešení problému zablokování
Pštrosí algoritmus: Předstíráme, že problém není, uživatel sám rozhodne (zabije proces)
Detekce a zotavení: Řešíme, až když nastane problém: Zabití nepohodlného procesu/ restart vybraného procesu
Vyhýbání se zablokování: „Opatrným“ přidělováním – potřeba znát budoucnost, na main-framech
Předcházení zablokování: Napadení některé z HP: Nelze obecne, závisí na typu prostředku
Vzájemné vyloučení: Spooling – Fronta
Drž a čekej: O všechny prostředky požádat před startem, před další žádostí vše uvolnit
Neodjímatelnost – nepoužívá se - zmatky
Čekání do kruhu: Možnost žádat pouze o 1 prostředek
Očíslování prostředků, možno požádat pouze o prostředky s vyšším číslem

Správa paměti

Přidělování paměti: - spojité/ nespojité (1/ více bloků)
Přidělávání v prostředí více procesů: nutno řešit relokace (proměnné nejsou vždy na stejném místě) a ochranu
Pevné oddíly (*interní*; paměť dělena, vybere se díl správné velikosti)/ volné oddíly (*externí*; dle potřeby)
Řízení paměti: Bitová mapa (bloky stejné délky „paragrafy“)
Spojový seznam bloků: Na začátku velikost bloku a pozice dalšího
Je třeba řešit: Který blok; spojování volných bloků

Strategie přidělování: First fit, next fit, best fit (nejmenší větší), worst fit (z největšího)

Virtuální paměť: Oddělení adresového prostoru fyzické paměti a adresového prostoru procesu; převod adres
Ochrana paměti, cena, snížení nebezpečí uváznutí, snížená fragmentace; plochý adresový prostor; není třeba řešit relokace; rozšíření/ zmenšení paměti:
2 základní metody: Stránkování/ segmentace
Adresování nelze softwarově ale potřeba podpory



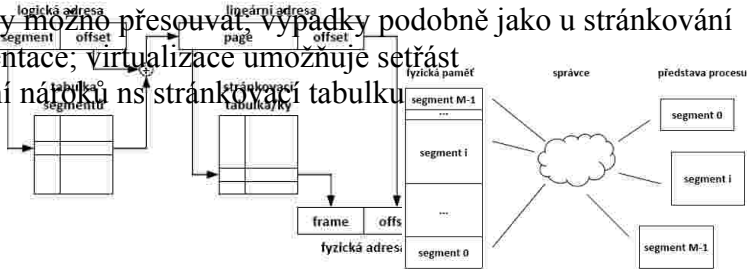
Stránkování
VAP i FAP rozděleny na bloky stejné velikosti; VAP – *stránky*, FAP - *rámce*
Pohled uživatele: Spojitý adresový prostor
Skutečnost: Stránky „rozházeny“ po hlavní i sekundární paměti, některé chybějí zcela
Adresa ve VAP je vyjádřena jako dvojice [p,d] (page, dispace mem); správce převede *p* na číslo rámce
Problémy stránkování: Interní fragmentace; rychlost přístupu do stránkovacích tabulek
Velikost stránkovacích tabulek: 32b adresy → 4GB, stránky 3kB → stránkovací tabulka má 1M položek
Výběr velikosti stránek: Malé stránky: Malé odkazy, menší fragmentace; velké stránkovací tabulky
Velké stránky: Malé stránkovací tabulky, lépe vyhovuje I/O; větší fragmentace
Úpravy: Víceúrovňové stránkování - Řeší problém velikosti tabulek, reálné jen do 4. úrovně
Asociativní paměť: Řeší problém rychlosti přístupu
Nulaúrovňové stránkování: Bez tabulek, jen asociativní paměť – malá & drahá
Inverzní stránkovací tabulky q: Organizace nad rámci, nikoli stránkami (prohledávání, ne přes index)

Algoritmus výměny stránek (replacement policy)
Optimální výměna: Vybrána bude stránka, na kterou přistoupíme za nejdelší dobu: Potřeba znát budoucnoust
FIFO: Klasická fronta: Nahrazení stránky, která byla v paměti nejdéle
Beladyova anomálie – zvýšení počtu rámců může vést ke zvýšení počtu výpadků
LRU: Dlouho nepoužívané stránky asi nebudu potřebovat; seznam, který je při každém přístupu upraven
HW realizace: 64b čítač, který CPU při přístupu uloží do PTE, vybírá se nejnižší
Matice *n*x*n*, při přístupu do rámce *k* nastavíme *k*-tý řádek na 1 a *k*-tý sloupec na 0.
NRU: Příznaky Accessed, Dirty (nastavuje HW, nuluje OS)

Algoritmus: Periodické vynulování A; při výpadku volím náhodně ze tříd v pořadí 0-0, 0-1, 1-0, 1-1

NFU: SW řešení LRU: Čítač u stránek, periodicky přičítáme A, vybere se nejnižší; problém – nezapomíná
Úprava – posunu doprava, přičítám doleva – stárnutí (aging)
Clock: Stránky jsou zařazeny do kruhového seznamu, při výpadku beru *A*=0, je-li *A*=1, nastavím 0 a posunu se

Segmentace: Nezávislý adresový prostor 0…limit; organizace podle potřeb a struktury programu
Segmenty různé a měnitelné velikosti; segmenty možno přesouvat, výpadky podobně jako u stránkování
Problém: dynamická alokace → externí fragmentace; virtualizace umožňuje setřást
Rozdělení stránkovací tabulky: Sdílené kódu; zmenšení nároků ns stránkovací tabulku



Segmentace vs. stránkování

programátor musí vědět
externí fragmentace
separátní ochrana
sdílený kód, separátní kompilace

programátor nemusí vědět
interní fragmentace

Souborové systémy

V hlavní paměti lza pracovat jen s omezeným množstvím informace; obsah paměti je při ukončení procesu ztracen; sdílení dat mezi více procesy
Soubor: Nezničí ho vypnutí; má jméno, aby bylo možné se na něj odkazovat i po skončení programu
Z pohledu OS: Jméno (lokální – pro proces/ globální – pro všechny); atributy; struktura; typ (bin/ text…)
Na souborech OS definuje druhy přístupu a povolené operace
Pojmenování souborů umožňuje lidskému uživateli přístup k jeho datům
Přesná pravidla určuje OS: Délka jména; malá vs. velká písmenka; speciální znaky; přípony a jejich význam
Atributy souborů definují vlastnosti a uchovávají informace o souboru
Struktura souborů: Sekvence bajtů (Windows, UNIX)/ sekvence záznamů (mainframes)/ strom
Typy souborů: Běžné soubory (informace)
Adresáře (tvoří strukturu nad soubory)
Speciální soubory (znakové/blokové; soft linky = odkazy)
Přístup k souborům: Sekvenční: Pouze pohyb vpřed, možné převinutí na začátek; umožňuje OS přednačítání
Přímý přístup: Umožňuje měnit aktuální pozici
Mapování do paměti: Využití stránkování/ segmentace
Soubory mapované do paměti: „pojmenovaná“ virtuální paměť
Program přistupuje k souboru pomocí běžných instrukcí pro práci s paměti
Vhodnou implementací lze ušetřit kopírování souboru po paměti
Problémy: Přesná velikost souboru → zvětšování souboru; velikost souborů
Proč potřebujeme adresáře: Udržení organizační struktury souborů; uchování atributů souboru
Adresář – zvláštní struktura souboru: Vnitřní struktura definována OS;
Speciální operace nad adresáři: hledání souboru, vypsání adresáře; přejmenování, vytvoření, smazání souboru
Cesta k souboru: Pojmenování souboru v hierarchickém uspořádání
Absolutní cesta: Cesta v grafu od kořene k souboru
Relativní cesta: Cesta z aktuálního adresáře k souboru
Aktuální adresář: Vlastnost procesu; jména souborů, která nezačínají kořenem, se hledají vzhledem k AA

Hard link: Na jedna data souboru se odkazuje z různých položek v (různých) adresářích
Soft link (symbolický odkaz, symlink): Speciální soubor, který odkazuje jméno souboru

Hierarchická struktura: Strom: Jednoznačné pojmenování
DAG = Directed Acyclic Graph = orientovaný acyklický graf: Víceznačné pojmenování, bez cyklů
Obecný graf: Víceznačné pojmenování; cykly vytváří problém při hledání

Implementace souborových typů
Správa souborů: Kde jsou data uložená v souboru
Správa adresářů: Mapování jména na jeho binární identifikaci; uložení atributů
Správa volného místa: které bloky jsou ještě volné

Ukládání souboru na disk: Disk → sektor → blok; průměrná velikost souboru ~1500B
Velké bloky: Rychlejší práce se souborem; nebezpečí velké vnitřní fragmentace;
Malé bloky: Pomalejší práce s diskem; větší režie na informaci o volných blocích
Souvislá alokace: Souvislý sled bloků; informace o uložení souboru sestává pouze z čísla prvního bloku
Lepší práce s diskem; problém při hledání volného místa; problém při zvětšování souboru
Spojovaná alokace: Pospojování bloků použitých pro soubor;
Indexová alokace: záznamy odkazují na záznamy a ty opět odkazují na záznamy….cosi zavánějící stromem

Implementace adresářů: Záznamy pevné velikosti - FAT
Spojový seznam: Delší jména; pomalé hledání
B-stromy: Rychlé hledání; NTFS

Volné místo na disku: Obdoba správy paměti; souvislý adresový prostor rozdělený do bloků
Bitmapa: NTFS, HPFS, Netvare

Spojový seznam volných bloků: UNIX, EXT (ne EXT2)