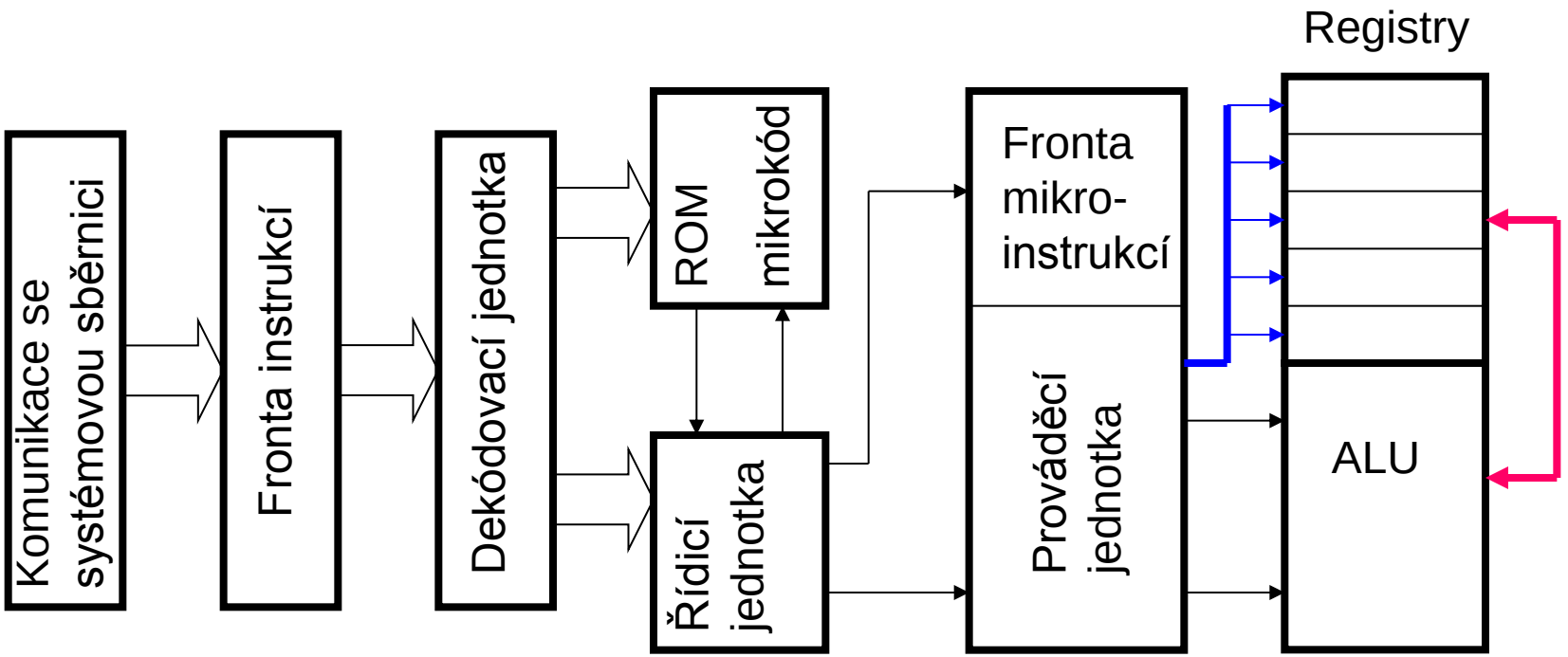
Architektury CISC a RISC, uplatnění v personálních počítačích

Cíl přednášky

- Vysvětlit, jak pracují architektury CISC a RISC, upozornit na rozdíly.
- Zdůraznit, jak se typické rysy obou typů architektur projevily v konstrukci PC.

Architektura procesoru CISC



Obr. 1

Význam zkratek:

CISC – Complex Instruction Set Architecture

RISC – Reduced Instruction Set Architecture

Posloupnost činností při provádění instrukce

1. Čtení instrukcí z paměti do fronty instrukcí (Prefetch Queue – fronta instrukcí).
2. Dekódování instrukcí v dekodéru (Decoding Unit – dekódovací jednotka).
3. Provedení instrukce (Execution Unit) - provedení mikroprogramu sestávajícího z mikroinstrukcí.
4. Mikroinstrukce jsou řazeny do fronty mikroinstrukcí (Microcode Queue - fronta mikroinstrukcí).
5. Mikroinstrukce pak vstupují do prováděcí jednotky (Execution Unit).

Existence mikroprogramu – typický rys architektur CISC.

Zřetězené zpracování instrukcí – typické pro oba typy architektur.

Prováděcí jednotka provádí mikroprogram složený z mikroinstrukcí.

Mikroprogram a mikroinstrukce

- Paměti mikroprogramů - v ní jsou uloženy mikroprogramy jednotlivých instrukcí.
- Každé **instrukci** odpovídá posloupnost **mikroinstrukcí**.
- Vyvolání mikroprogramu – řídicí jednotka (control unit) na základě informace z dekodéru (tato informace reprezentuje konkrétní instrukci).
- Všechny tyto činnosti je možné považovat za přípravné činnosti, které jsou poměrně časově náročné a srovnatelné s dobou provádění instrukce.
- Možnost realizovat mnohé činnosti automaty – v typické architektuře CISC není využito - rys architektur RISC.
- **Procesor vykonává rozsahem složité instrukce (complex), ty jsou implementovány formou mikroprogramu, různá délka a složitost instrukcí - typický rys architektur CISC.**

Zřetězené zpracování – problém instrukcí skoku

- Důležité: rozhodnutí o tom, zda bude realizován skok, bude realizováno v **prováděcí jednotce** – mezitím se rozpracují další instrukce**.**
- Při zpracovávání instrukce *n* v Execution Unit (EU) je v Prefetch Queue již načtena instrukce *n+1* a v dekodéru probíhá její rozdekódování na množinu budoucích příznaků, na jejichž základě bude v následujícím kroku zahájeno provádění jiného mikroprogramu.
- Čím vyšší úroveň rozpracování instrukce, tím více „práce“ procesoru přijde nazmar, pokud instrukce skoku změní posloupnost provádění instrukcí.
- Řešení: vytváření dvou front instrukcí, techniky předvídání skoku.

Rysy architektur CISC - shrnutí

- Procesor je řízen instrukcemi různé délky s mnoha parametry, modifikačními bity => složité dekodéry instrukcí. Operandy instrukce uloženy v registrech, pamětech (RVP, operační paměť, ….) => různá doba provádění operací v závislosti na tom, kde je uložen operand, nepevná pozice operandů v instrukci.
- Realizace instrukce – mikroprogram uložený v paměti PROM – rychlost závisela na vybavovací době paměti PROM (obecně platí: realizace funkce obvodově je rychlejší než realizace programem).
- Mikroprogramově řízený procesor je pomalejší než obvodově realizovaný procesor (hardwired).
- Realizace některých operací obvodově v architekturách RISC – využití postupně i v architekturách CISC.
- Problémy s instrukcemi skoku – řešeny důsledněji v architekturách RISC, postupně přenášeny do architektur CISC.
- Shrnutí: progresivní řešení problémů souvisejících se zřetězeným zpracováním v architekturách RISC, později uplatnění některých těchto principů v architekturách CISC (začalo to v procesorech Pentium).

Rysy architektur CISC – shrnutí

- Neustále se zdokonalující technologie výroby integrovaných obvodů a také pamětí – do paměti ROM bylo možné vkládat mikroprogramy s narůstajícím rozsahem.
- Mikroprogramově řízené procesory – byl nastaven trend tvorby komplexních a složitých instrukcí, jim odpovídaly složité mikroprogramy.
- Důsledek: stav, kdy bylo nutné zvětšovat kapacitu ROM paměti se stal v jisté fázi neúnosný.
- Paměti ROM jsou prvky, které měly horší rychlostní parametry než paměti RAM (u prvních verzí počítačů na bázi procesorů Intel řešeno stínováním paměti).
- Na zpracování 20 % nejčastěji používaných instrukcí se spotřebuje 80% strojového času => vznikla otázka optimalizace těchto instrukcí.
- Na době potřebné pro provedení instrukce se výrazným způsobem podílí doba na dekódování instrukce – u složitých instrukcí se to stává důležitým aspektem (dekódování složitých instrukcí je časově náročnější, několik stupňů dekódovacích obvodů, vyšší cena).

Řízení architektur CISC mikroprogramem – shrnutí

- Výhoda řízení procesoru mikroprogramem: přechod na vyšší verzi mikroprocesoru – doplnění o nové instrukce a vytvoření nových mikroprogramů reprezentujících činnosti, které mají tyto nové instrukce realizovat (nová instrukce = nový mikroprogram) - extensivní rozvoj.
- Mikroprocesory CISC se proto vyvíjely především tak, že se rozšiřovaly množiny instrukcí a vytvářely se jim odpovídající mikroprogramy (s neměnnou množinou mikroinstrukcí) – extensivní rozvoj.
- Pozitivum: možnost tvorby instrukcí šitých na míru konkrétní třídě aplikaci.
- Stav v moderních procesorech v PC: jednoduché instrukce jsou realizovány obvodově, složité instrukce jsou realizovány mikroprogramem (cenový aspekt).
- Architektury RISC – pokud jsou instrukce realizovány hardwarově, pak přechod na vyšší verzi procesoru s inovovanou množinou instrukcí znamená nový návrh.

Závěry nad typy programů

- Obsahují většinou aritmeticko-logické operace (instrukce) a přesuny (move).
- Velká frekvence činnosti typu „přečti/zapiš operand“ (průměrně 1,9x/jednu instrukci) – reflexe v architekturách RISC (aritmeticko – logické operace nemají zásadně operand uložen v paměti (nejprve se provede load/store).
- Většina operandů jsou takového typu, že mohou být uloženy v registru nebo jsou to lokální proměnné nebo parametry.
- Mechanismy realizace operací CALL/RETURN potřebovaly změnu.
- **Toto jsou závěry, které můžeme považovat za startovací bod úvah o architekturách RISC.**

Rysy architektur RISC

1. Redukce počtu instrukcí, jejichž jednotlivé fáze se při zřetězeném zpracování provádějí pokud možno v rámci jednoho strojového cyklu.
2. Zjednodušení instrukcí, jejich implementace obvodově (hardwired), pevná délka instrukce a jednotný formát.
3. Jednoduchá instrukce – jednoduchá řídicí jednotka, jednoduché datové cesty, možnost zvyšovat pracovní kmitočet.
4. Zřetězené provádění aritmeticko-logických instrukcí bez odkazů do paměti, snadnější naplnění požadavku, aby různé instrukce trvaly stejnou dobu => snadnější splnění požadavků na zřetězené zpracování instrukcí.
5. Složité a rozsáhlé instrukce (co do počtu kroků) neexistují, jsou nahrazeny jednoduššími instrukcemi.
6. Omezení komunikace s pamětí – pouze instrukce LOAD/STORE.
7. Implementace instrukcí logickými obvody – např. sekvenčním automatem (výrazně rychlejší alternativa než mikroprogram).
8. Nedestruktivní zpracování operandů.
9. To, co se provádělo v počítačích CISC jako složitá instrukce, se v architekturách RISC provádí jako posloupnost jednoduchých instrukcí.

Doba provádění programu – příklad

- Předpoklad: máme program, který obsahuje 80% instrukcí jednoduchých a 20% složitých.
  - **Počítač CISC**: jednoduchá instrukce 4 cykly, složitá instrukce 8 cyklů, doba cyklu – 100 ns – 10^{-7} s
  - **Počítač RISC**: jednoduché instrukce trvají jeden cyklus, složité operace jsou realizovány jako posloupnost jednoduchých instrukcí – budeme předpokládat průměrně 14 instrukcí (14 cyklů, doba 75 ns – 0,75 x 10^{-7} s)
- Máme program sestávající z 1 000 000 instrukcí, pak:

  CISC: $(10^6 \times 0{,}80 \times 4 + 10^6 \times 0{,}20 \times 8) \times 10^{-7} = 0{,}48$ ms

  RISC: $(10^6 \times 0{,}80 \times 1 + 10^6 \times 0{,}20 \times 14) \times 0{,}75 \times 10^{-7} = 0{,}27$ ms
- Závěr:

  složité instrukce trvají déle na počítačích CISC, jejich počet je ale menší

  RISC počítač pracuje na vyšší frekvenci, počítač CISC na této frekvenci není schopen pracovat (obvodová složitost).

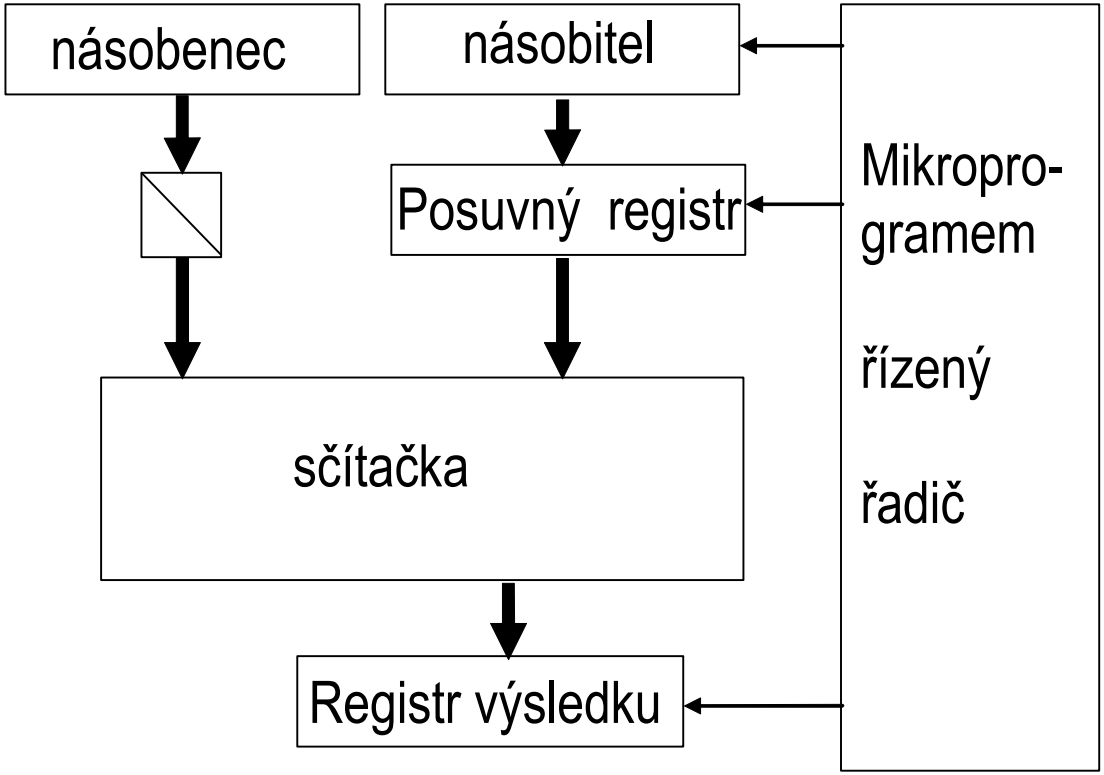
Násobení v procesoru CISC

Posloupnost operací:

1. Násobení je možné v procesoru CISC realizovat tak, že jeden operand (násobenec) posuneme o jednu pozici doleva a přičteme jej k mezivýsledku, pokud je odpovídající bit násobitele roven 1 – pokud je rovna 0, provede se další posun doleva.
2. Doba potřebná pro realizaci instrukce bude záviset na tom, kolik bitů v násobiteli bude mít hodnotu rovnu 0 (tzn. nebude pevná doba provádění instrukcí).
3. Pro realizaci násobení dvou operandů bude potřebná sčítačka, patřičný počet registrů a řadič řízený mikroprogramem provádějící jednotlivé kroky násobení.
4. Pro každý krok podle bodu 1) bude realizováno několik mikroinstrukcí.

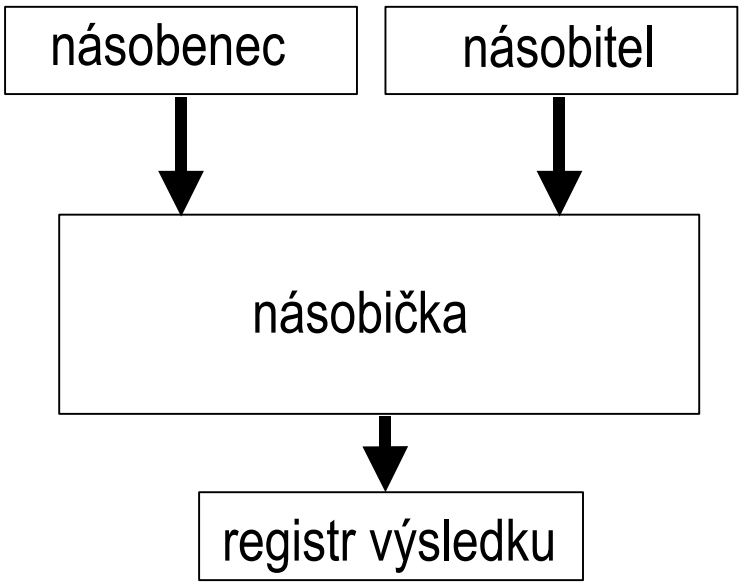
Závěr: realizace operace násobení bude potenciálně časově náročná.

Vybavení pro násobení v procesoru CISC



Obr. 2

Násobení v procesoru RISC



Obr. 3

V architekturách RISC je hardware cíleně navržen tak, aby optimalizoval zřetězené zpracování instrukcí.

Místo sčítačky se využívá specielní obvod (násobička), v němž je operace násobení realizována hardwarově a rychleji.

Doba potřebná pro realizaci násobení v násobičce se vždy provádí stejně dlouho, tzn. stejným počtem kroků (cyklů).

To je výhodné, protože při řetězeném zpracování instrukcí je dobře, když se v okamžiku zahájení nějaké činnosti přesně ví, kdy skončí.

Odraz těchto principů v architekturách procesorů Intel

- Procesory I8086 – I80386: převažovala implementace mikroprogramem uloženém v paměti ROM.
- Nevýhoda: proces, v němž byla ve hře paměť ROM, je pomalý.
- Doba procesorů I80286 – I80486: vybavovací doba RAM – 60 ns, ROM asi 200 ns.
- Řešení: stínování ROM – obsah ROM se při zavádění systému přenesl do paměti RAM.
- Dnes ROM BIOS je v paměti Flash – vybavovací doba 15 – 20 ns (první cyklus je výrazně pomalejší – kolem 100 ns), principielně je to paměť EEPROM (electrically erasable programmable read only memory), energeticky nezávislá.
- Se zvyšujícím se tlakem na zřetězené zpracování instrukcí (stejná doba trvání násobení nezávislá na hodnotách operandů) – snaha o realizaci procesu násobení pomocí hardwarových automatů => **implementace principů architektur RISC v architekturách CISC**.

Nedestruktivní operace v architekturách RISC

- RISC procesory umožňují tzv. **nedestruktivní provádění instrukcí**, což znamená, že obsahy paměťových míst, v nichž jsou uloženy operandy, se nemění.
- Stav v I80386 (CISC) – pokud operandy byly registry nebo paměťová místa, pak byly jejich obsahy provedením instrukce likvidovány.
- Příklad: ***ADD eax, mem32***

  Tato instrukce přičte obsah *mem32* k obsahu akumulátoru EAX a výsledek uloží zpět do EAX => původní hodnota uložená v EAX se přepíše novou hodnotou.

  Totéž platí pro instrukce s operandy registr – registr, registr – paměť nebo paměť – paměť.

  Důsledek: v konečném dopadu vyšší objem komunikace s pamětí v architekturách CISC.
- Architektury RISC – nedestruktivní režim, operandy zůstanou po provedení instrukce zachovány => zkratka RISC je také někdy překládána jako *Reusable Information Storage Computer* (neuchytilo se).

Nedestruktivní operace v architekturách RISC

- Důsledek – operandy mohou být používány opakovaně a nemusejí se tudíž znovu přenášet z paměti.
- Běžně pracují procesory RISC se 3 operandy, dvěma operandy reprezentujícími vstup a jedním pro uložení výsledku.
- Příklad: ***ADD dest, src1, src2***

  Dest – destination (místo určení)

  Src – source (zdroj)

  Procesor RISC touto instrukcí sečte dva operandy *src1, src2* a výsledek uloží do *dest*.

  => hodnoty uložené do *src1, src2* zůstanou zachovány.

  Výsledek: procesor RISC nepotřebuje tak často komunikovat s pomalou pamětí (příp. různými typy).
- Uplatnění architektur RISC – nepracuje se s jedním či dvěma universálními registry ale je k dispozici tzv. sada registrů (register file) – takto nejsou vybaveny procesoru Intel – cenové důvody. **Sady registrů nejsou určeny pro provádění aritmeticko-logických operací.**
- Architektura CISC (např. I80386) – pro ukládání výsledků aritmetických operací byl k dispozici pouze střadač.

Sady registrů v architekturách RISC

- Vyšší úroveň práce s registry a jejich využívání – **sady registrů** v procesorech RISC.
- Využití:
  - přepínání úloh,
  - předávání parametrů,
  - vyvolání podprogramu.
- Sady registrů obsahují až tisíce registrů – takový rys nemají zcela určitě architektury CISC a procesory v PC.

  Důvod: takový rys by výrazně zvýšil cenu počítače.

  Počítače CISC – nejčastěji využívaným registrem je **střadač**, v architekturách CISC má procesor k dispozici sadu universálních registrů.

Další odlišnosti v hardware mezi CISC a RISC

- Důležité: zatímco některé prvky se v architekturách CISC objevovaly pouze jednou (např. sčítačka), pak v architekturách RISC mohou být zapotřebí v každé fázi provádění instrukce, tzn. vyskytují se násobně.
- Příklad:

  Instrukce ***ADD eax, [ebx + ecx]*** – je zapotřebí provést 2 součty: určit adresu operandu a realizovat instrukci ADD.

  Původní procesory CISC – vše bylo realizováno jednou sčítačkou, bylo to možné, protože každý krok byl realizován v časově disjunktních okamžicích.

  Procesor RISC – každá jednotka realizující jednotlivé fáze provádění instrukce potřebuje své vlastní vybavení, což při současném stavu integrace není problém (v tomto případě musí sčítačka být v dekódovací jednotce a v prováděcí jednotce).
- Závěr: **ve snaze zrychlit činnost se rozšiřuje hardwarové vybavení,** v klasických architekturách CISC nic takového nebylo, dnes situace jiná, tyto principy jsou uplatňovány.

Další odlišnosti v hardware mezi CISC a RISC

- Příklad:

  Za sebou následují tyto instrukce:

  ADD eax, [ebx + ecx]

  MOV edx, [eax + ecx]

  V jistém okamžiku je instrukce ADD v prováděcí fázi => potřebuje sčítačku, instrukce MOV je ve fázi dekódování => potřebuje sčítačku na určení adresy operandu.

  Jiná alternativa – mít pouze jedinou sčítačku => dekódování instrukce MOV by se muselo zpozdit, dokud se nedokončí provedení sčítání (instrukce ADD).
- Další možná situace: předpokládáme tytéž instrukce jako v předcházejícím případě.

  ADD eax, [ebx + ecx]

  MOV edx, [eax + ecx]

  Konkrétní stav provádění této posloupnosti instrukcí: instrukce ADD je v prováděcí jednotce, instrukce MOV je v jednotce dekódovací.

  Problém: instrukce MOV potřebuje hodnotu uloženou v registru eax, ta je však známá až po provedení instrukce ADD.
- Technika co nejdokonalejšího vybavení jednotlivých komponent počítače vším, co potřebují, je v moderních procesorech CISC využita.

Řešení předcházejících situací

Řešení: 2 možnosti

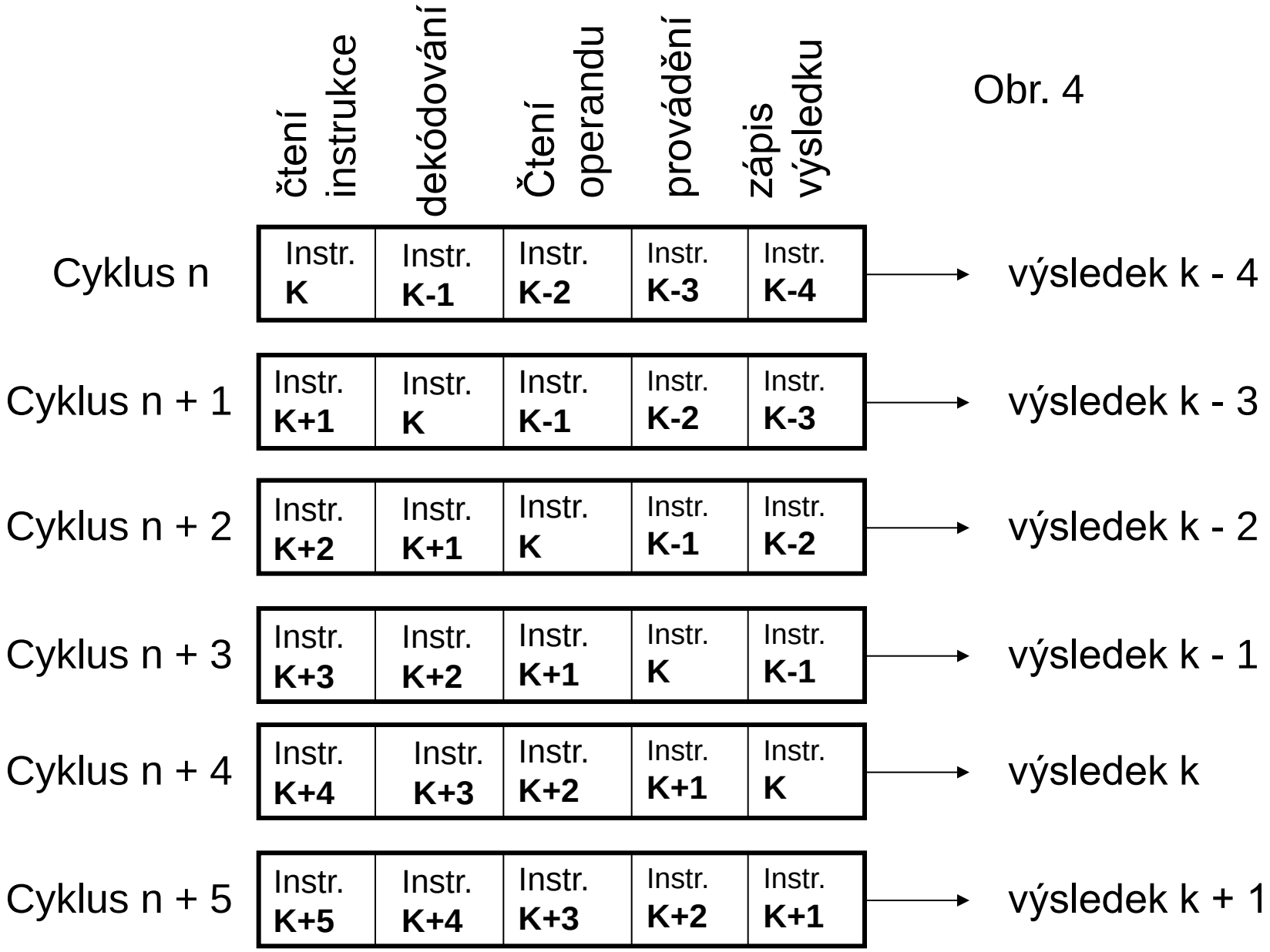
1. zajistit zpoždění výpočtu operandu v dekódovací jednotce (o jeden cyklus),
2. ve fázi překladu – překladač buď vloží do posloupnosti instrukcí NOP anebo provede restrukturalizaci programu => **v architekturách RISC tvoří hardware i software navzájem propojený celek, překladače musí respektovat problémy, které mohou nastat při implementaci hardware realizujícího instrukce**.

Výsledek: **optimalizace toku instrukcí při překladu programu v architekturách RISC,** nic takového není v současných architekturách CISC.

Vzájemné vazby mezi fázemi provádění instrukce a jejich blokování

- Důležitý aspekt zřetězeného zpracování – všechny fáze provádění instrukce by měly končit ve stejný okamžik.
- Ve struktuře podle obr. 4 to pak znamená, že všechny instrukce by měly skončit po 5 cyklech.
- Stav, kdy operandem instrukce jsou data z paměti – je potřeba je přečíst buď z rychlé vyrovnávací paměti (RVP) nebo z operační paměti (OP).
- Časové relace – přenos z RVP – kratší než z OP.
- Přenos operandu z OP => přechod z fáze „čtení operandu“ (operand fetch) do fáze „provádění instrukce“ (execution) se musí zpozdit => výrazným způsobem se naruší synchronizace jednotlivých kroků v jednotlivých fázích zpracování instrukce, vzniká problém se zajištěním stejné doby trvání zpracování instrukce v jednotlivých krocích.

Obr. 4


čtení instrukce
dekódování
Čtení operandu
provádění
zápis výsledku
Cyklus n
Instr. K
Instr. K-1
Instr. K-2
Instr. K-3
Instr. K-4
výsledek k - 4
Cyklus n + 1
Instr. K+1
Instr. K
Instr. K-1
Instr. K-2
Instr. K-3
výsledek k - 3
Cyklus n + 2
Instr. K+2
Instr. K+1
Instr. K
Instr. K-1
Instr. K-2
výsledek k - 2
Cyklus n + 3
Instr. K+3
Instr. K+2
Instr. K+1
Instr. K
Instr. K-1
výsledek k - 1
Cyklus n + 4
Instr. K+4
Instr. K+3
Instr. K+2
Instr. K+1
Instr. K
výsledek k
Cyklus n + 5
Instr. K+5
Instr. K+4
Instr. K+3
Instr. K+2
Instr. K+1
výsledek k + 1

Komentář k předcházejícímu obrázku

- Ideální případ – v každém stupni se provádí jedna fáze jedné z 5 instrukcí, pro realizaci jedné fáze je zapotřebí jeden cyklus => na provedení instrukce je zapotřebí 5 synchronizačních pulsů a s každým synchronizačním pulsem se na výstupu objeví jeden výsledek.
- Mezi jednotlivé stupně je vložen registr, který pracuje jako výstupní pro jeden stupeň a jako vstupní pro následující.
- Takové uplatnění paralelismu je dnes typické pro CISC i RISC procesory.
- První typy CISC procesorů – neměly takovou strukturu => instrukce k + 1 se zahájila poté, co skončila instrukce k (nebyly zpracovány zřetězeně).
- Pokud by byly 2 procesory - jeden RISC a druhý CISC - realizovány na shodné technologii (pracovaly by stejně rychle), pak by se instrukce v procesoru RISC prováděla 5 x rychleji než v původním procesoru CISC (prvních typech procesorů CISC bez zřetězení).
- **Zřetězené zpracování - pozdější verze procesorů CISC.**

Řešení problému s pamětmi v architekturách RISC

- Řešení:
- Zpoždění provádění instrukce - vkládáním NOP – tzv. „***delayed load“*** (zpožděné čtení).
- Jiná technika – „load forwarding“ (zavedení operandu dopředu) – operand se z paměti nečte do registru ale přivádí se přímo na vstup ALU, tzn. registry se obejdou (proto forwarding).
- Zásadní řešení v architekturách RISC - **snaha o vyřazení instrukcí, kde operandem je hodnota uložená v paměti** => s pamětí manipulují pouze instrukce typu ***load/store***, žádné jiné instrukce s pamětí nemanipulují => instrukce ***ADD*** odkazuje pouze na interní registry procesoru => instrukce ***ADD reg, mem*** neexistuje.

Řešení problému s pamětmi v architekturách RISC

- **Příklad**:

  LOAD reg1,mem

  ADD dest,reg1,reg2

- Stav, kdy instrukce LOAD je ve fázi provádění, zatímco ADD je ve fázi čtení operandů => zákonitě musí vzniknout ***„delayed load“,*** protože ADD se nemůže provést, dokud není v reg1 uložen operand z paměti (např. za LOAD se zařadí NOP).

  Instrukce ADD se zpozdí – existují na to další techniky.

- **Řešení**:

  Berkeley – ve hře je hardware

  Stanford – optimalizace při překladu

Řešení Berkeley

- Je zaveden pojem „scoreboarding“.
- Každému registru procesoru je přidělen jeden bit, který reprezentuje to, zda informace v něm uložená je/není platná.
- Princip: instrukce, která operuje s registrem, nejprve na začátku provádění instrukce nastaví tento bit do „1“ (obsah je neplatný), jakmile se do registru během fáze provádění (execution) vloží data, pak se tento bit vynuluje.
- Instrukce, která potřebuje platný operand v reg1 (instrukce ADD), je pak patřičným způsobem zpožděna, pokud je tento bit v 1.
- **Toto všechno zařídí řadič, tzn. hardware a řeší se to až při provádění instrukce.**
- Toto řešení je využíváno spíše v architekturách CISC (procesory instalované do PC).

Řešení Stanford

- Je řešeno již při překladu programu tak, že do posloupnosti instrukcí vkládá překladač instrukce typu NOP => výsledkem je tento kód:

  LOAD reg1,mem

  NOP

  NOP

  ADD dest,reg1,reg2

  Takové řešení má problémy, protože překladač nebere v úvahu tyto aspekty (v okamžiku překladu nejsou známy):

  zda je operand v RVP typu L1, L2 nebo hlavní paměti (OP),

  kmitočet, jímž budou tyto operace řízeny,

  kvalitu paměťových čipů (rychlost),

  režim činnosti paměti.

- Kompilátor musí počítat spíše s horší alternativou, takže počet vložených NOP může být příliš vysoký => snaha o optimalizaci kompilátoru, kompilátor musí respektovat principy uplatněné v procesoru => bude souviset s typem procesoru (typické pro RISC).
- Toto řešení je využíváno spíše v architekturách RISC.

Zpožděný skok a zpožděné větvení

- Instrukce skoku a podmíněného skoku tvoří asi 30 % všech instrukcí (?) – tyto skupiny instrukcí mají na běh programu nežádoucí efekt.
- Situace: zřetězené zpracování instrukcí, pak:

- Čítač adresy ukazuje průběžně na adresy načítaných instrukcí, ve frontě instrukcí (před instrukcí právě načítanou z paměti) je ale instrukce skoku nebo podmíněného skoku (bude se provádět dříve než instrukce právě načítaná z paměti), instrukce skoku mohou potenciálně změnit posloupnost provádění instrukcí.
- To, zda se bude měnit adresa (tzn. bude se přecházet na jinou instrukci) podle výsledku podmínky v instrukci podmíněného skoku, bude jasné, až se tato instrukce dostane do fáze provádění (execution) – proto zpožděný skok/zpožděné větvení.
- **Tzn. když se bude provádět instrukce skoku nebo podmíněného skoku, budou v jednotlivých fázích rozpracovány různé instrukce, které se budou potenciálně rušit (podle výsledku instrukce podmíněného skoku).**

Zpožděný skok a zpožděné větvení

- Má se realizovat skok, musí se zrušit všechny instrukce, které jsou již rozpracovány v jednotlivých fázích.
- Možné řešení – zařazení instrukcí NOP za instrukce skoku – musí se zařídit při překladu.
- Účinek vkládání NOP: neprovádějí se instrukce, které možná (podle výsledku instrukce podmíněného skoku – ten se ještě neví) bude zbytečné provádět.
- Výsledkem provedení instrukce podmíněného skoku je vygenerování ukazatele na další instrukci (2 možné výsledky) v kroku zápis výsledku (write-back).
- Jiná možnost: jakmile se v dekodéru rozpozná některá z těchto instrukcí, tak se začnou načítat dva sledy instrukcí respektující oba možné výsledky podmíněného skoku.
- Další možnost řešení: dostatečně dlouhá fronta instrukcí, takže se zvyšuje pravděpodobnost, že oba výsledky instrukce podmíněného skoku budou odkazovat na instrukci, která je ve frontě a nemusí se tudíž číst z paměti (RVP nebo OP).
- Progresivní způsob: předvídání výsledku instrukce podmíněného skoku (bude vysvětleno později) – technika BTB (Branch Target Buffer).

Aktuálnost obsahu registru – architektury RISC (další příklad)

- Problém s aktuálností obsahu registru existuje i v architekturách RISC (operandem není paměť, ale registr).
- Předpokládejme posloupnost těchto instrukcí:

  ADD reg1,reg2,reg7

  AND reg6,reg1,reg3

  Instrukce ADD - má se sečíst obsah registrů reg2 a reg7 a výsledek se má uložit do reg1.

  Instrukce AND – provést logický součin mezi obsahy registrů reg1 a reg3 a výsledek uložit do reg6.
- Podmínka korektní realizace instrukce AND – obsah reg1 musí být platný – pozor: zápis výsledku se odehrává až v poslední fázi provádění instrukce, instrukce ADD je v tom okamžiku ve fázi provádění a obsah reg1 nemusí být ještě zapsán.
- Řešení: Berkeley, Stanford (bylo už zmíněno při řešení v architekturách CISC).

  ADD reg1,reg2,reg7

  NOP

  NOP

  AND reg6,reg1,reg3

Posouzení výkonnosti

- Určení MIPS v situacích, kdy se vkládají NOP:

  NOP je instrukce, která nic nekoná – je potřeba takový algoritmus posuzovat podle efektivity výpočtu (jakoby instrukce NOP nebyly zařazeny do posloupnosti instrukcí).
- Problém správného obsahu registru se umocňuje u superskalárních procesorů – tento problém může vzniknout nejenom mezi fázemi v rámci jedné fronty ale také mezi fázemi různých front.
- Závěr: uplatnění principů zřetězeného zpracování způsobuje vznik problémů, které je nutno následně řešit.

Formáty instrukcí procesorů RISC, srovnání s CISC

- Výrazně kratší instrukce Procesorů RISC, všechny mají stejnou délku.
- Příklad kódování registrů v RISC procesorech: 8 registrů – 3 bity, na pevné pozici v instrukci.
- Kódování registrů v CISC procesorech (v začátcích): 8 registrů – 8 bitů (každý registr je reprezentován jedním bitem) – při malém počtu registrů snadno realizovatelné (méně náročné dekódování).
- Instrukce RISC – nedestruktivní (3 registry – 2 operandy, výsledek => hodnota žádného operandu se nepřepíše).
- Všechny instrukce mají stejnou délku – např. dvouregistrová instrukce ***MOV dest,src1*** bude mít nulové pole odpovídající druhému operandu src2, platí to i o instrukci NOP – bude mít všechna pole volná – délka těchto instrukcí je však stejná jako ostatní.

Formáty instrukcí procesorů RISC, srovnání s CISC

- Instrukce CISC – obsáhlá instrukce se značným množstvím informace (parametrů).
- Přeložené instrukce mikroprocesoru I80386 měly různou délku: 1 – 15 slabik.
- Takový složitý kód je možné rozumným způsobem realizovat pouze kombinací mikroprogramu a hardwarových dekodérů.
- Dekódování takového typu instrukce je náročné na podporu (hardware i mikroprogram), je ale i časově náročné.
- Závěr: principy konstrukce instrukcí pro procesory RISC jsou výrazně jednodušší (průhlednější) ve srovnání s principy využívanými v architekturách CISC – to má další pozitivní dopady při konstrukci obvodů procesorů RISC.

Příklady využití procesorů CISC

VAX 11 /70:

počet instrukcí: 303

délka instrukcí: 2 – 57

formát instrukce: různý

adresové režimy: 22

počet universálních registrů: 16

Pentium:

počet instrukcí: 235

délka instrukcí: 1 – 11

formát instrukce: různý

adresové režimy: 11

počet universálních registrů: 8

Příklady využití procesorů RISC, příp. s rysy RISC

Sun SPARC:

počet instrukcí: 52

délka instrukcí: 4

formát instrukce: pevný

adresové režimy: 2

počet universálních registrů: až 520

PowerPC:

počet instrukcí: 206

délka instrukcí: 4

formát instrukce: není pevný (malé diference)

adresové režimy: 2

počet universálních registrů: 32

Shrnutí dosavadních poznatků

- Jak RISC tak CISC architektury usilují o totéž: inovaci a zdokonalování výpočetních systémů – každý to řeší jinak.
- CISC: implementace složitých instrukcí, jimiž pokrývají požadované nové funkce.
- RISC: inovace na základě analýzy programů, snaha o efektivní zřetězené zpracování programů.
- Důležité rysy architektur RISC: redukovaný počet jednoduchých instrukcí, málo adresovacích režimů, využívání instrukcí load/store, instrukce definované délky a formátu, vyšší množství registrů.
- Současné architektury procesorů v PC vykazují znaky RISC i CISC.
- Závěr: RISC označuje procesory s redukovanou instrukční sadou, jejichž návrh je zaměřen na jednoduchou, vysoce optimalizovanou sadu strojových instrukcí, která je v protikladu s množstvím specializovaných instrukcí jiných architektur.