12.

Účel dekompozice systémů-

Dekompozici definujeme jako proces rozkladu systému na jeho dílčí části, a to podle určitých hledisek a s určitým cílem. Dekompoziční metody zajišťují optimální rozčlenění systému z hlediska zajištění jeho funkce. Pojmem dekompozice je také nazýván výsledek procesu.

Důvody, které vedou k použití tohoto procesu, jsou spojeny s problematikou systémové analýzy nebo i systémového návrhu.

Prvním důvodem je technická zvládnutelnost systémových operací. Rozklad systému na jeho podsystémy (a dále na jeho prvky) přináší větší přehlednost systémové struktury. Umožňuje analýzu jeho vlastností vhodnou analýzou vlastností chování jednotlivých podsystémů.

Druhým důvodem je zjednodušení návrhu systému podle funkcí jeho podsystémů s přehlednějším popisem vazeb mezi takovými dílčími spolupracujícími a vzájemně provázanými subsystémy

Třetím důvodem je skutečnost, že proces dekompozice je přímou součástí procedury strukturovaného přístupu systémové analýzy i systémového návrhu. Popis struktury, sestavený na určité rozlišovací úrovni, může být v dalších krocích zpodrobňován na rozlišovací úrovni nižší a opakován ať po dosažení úrovně základních prvků

Čtvrtým důvodem je skutečnost, že dekompozice systému může být součástí jeho integrační úlohy, tj. úlohy, zabývající se slučováním skupin prvků do nových integrovaných prvků.

Zásady provádění dekompozice systémů

-Zásada integrity

-V procesu dekompozice musí být zohledněna skutečnost, že je třeba stále sledovat a respektovat vlastnosti systému jako celku. Ve výsledné struktuře nesmí chybět žádná část (např. prvek) systému.

-Zásada soudržnosti

-V procesu dekompozice musí být stále sledován požadavek možnosti opětovného spojování dekomponovaných částí do větších celků. Musí být věnována pozornost sledování vazebv systému a zabráněno jejich ztrátě.

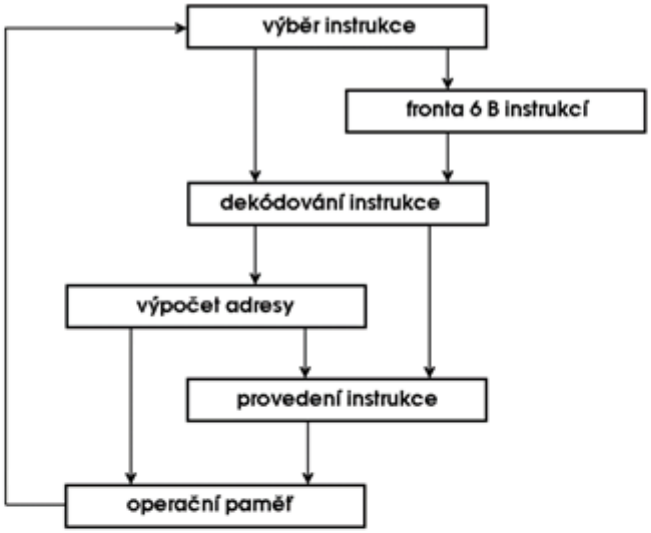
Vektorové proudové zpracování (pipeline)

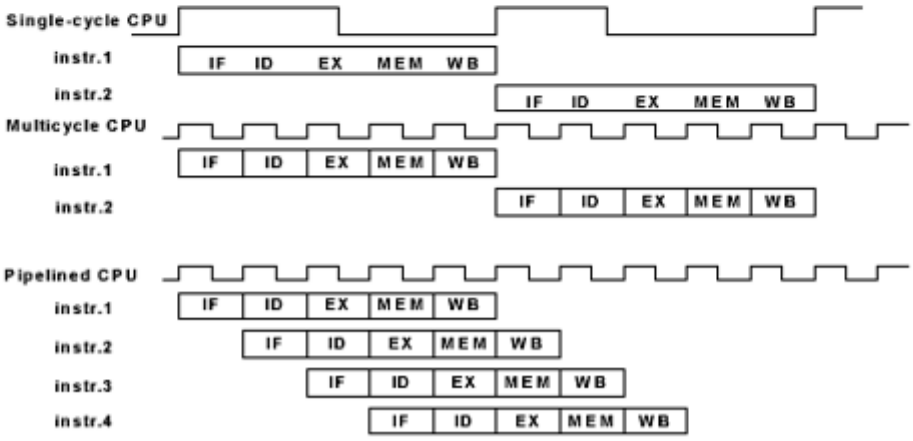
při zpracování dat je vykonávána posloupnost velmi jednoduchých úkonů seřazených tak, že na sebe navazují a doplňují se. Jeden každý operand pak prochází řadou operací, ale každá operace se vykonává až poté, co jsou všechny předchozí operace ukončeny

v každém časovém okamžiku se v každém bloku zpracovává jiný operand, který se po zpracování bude předávat do jiného bloku. Je to však možné jen za té podmínky, že rychlosti ve všech blocích budou stejné (synchronní zpracování) a čas bude odpovídat nejpomalejšímu bloku! Pak se tomuto zpracování informací říká **proudové zpracování** (pipelining)

nejjednodušší proudové zpracování řady Intel začíná procesorem 8086, který byl rozdělen na 2 samostatné, ale spolupracující části. Předvýběr tam zajišťuje část BIU. Díky tomuto kroku se výkon procesoru zvýšil cca o 35 % a díky 6 B frontě instrukcí o dalších 15 %. To z toho důvodu, že procesor nemusí čekat na předání instrukce, ale odebere si ji sám z fronty instrukcí maximální možnou rychlostí.

členění zpracování instrukce v procesoru 8086





Operační kód

Operační kód (také opkód) je ta část instrukčního pole, která určuje typ instrukce a jejích operandů, tedy zda půjde například o sčítání, odečítání, skok, atp. U delších instrukcí bývá zpravidla umístěn na začátku.

Operandy

Operand instrukce je jejím parametrem. Jestliže operační kód říká co se bude provádět, operandy definují s čím, s jakými daty. V bitových polích vyhrazených pro operandy (neboli parametry) instrukce může být zakódována:

konstanta – jako operand se použije přímo hodnota bitového pole

registr – číslo v bitovém poli určuje registr

adresa určená konstantou – operandem je hodnota na adrese určené bitovým polem

adresa určená registrem – operandem je hodnota na adrese obsažené v registru určeném bitovým polem

adresa určená kombinací registrů a konstant – v bitovém poli může být dohodnutým způsobem zakódován i složitější adresový výraz V instrukční sadě i386 lze například adresu zadat i jako součet bázového registru, indexovaného registru vynásobeného jednou z konstant 1,2,4 a konstanty.

Šířka instrukčního pole

Počet bitů kódu instrukce, tj. základní šířka instrukčního slova je většinou násobkem osmi, takže instrukce zaujímá v paměti celý počet bytů. Výjimkou jsou např. některé řady mikrokontrolérů PIC, které používají dvanáctibitové nebo čtrnáctibitové instrukční slovo.

Čím výkonnější má být určitý procesor, tím větší šířku instrukčního slova používá. Do širšího instrukčního slova je možné zakódovat více informací, takže instrukce pak mohou být výkonnější. Nevýhodou širšího instrukčního slova je nižší hustota kódu. Široké instrukční slovo sice dovoluje udělat více věcí najednou. Například DSP mohou mít instrukci, která vynásobí dvě buňky z kruhových bufferů, výsledek násobení posune o daný počet bitů doprava a přičte do akumulátoru, přičemž současně posune indexy kruhových bufferů na následující prvek. V případě že potřebujeme jednoduchou akci (přičíst do registru jedničku), bude pravděpodobně kód schopný vyjádřit složité operace zbytečně dlouhý.

Broken pipeline

Je to instrukce, která se nebude vykonávat (neměla tam být)

Muže vzniknou třeba smyčkou

```
Cpi r16,r17
Brne loop
Mov r16,r17
```
Možná oprava -Dát sem instrukci, která se bude vykonávat ve smyčce