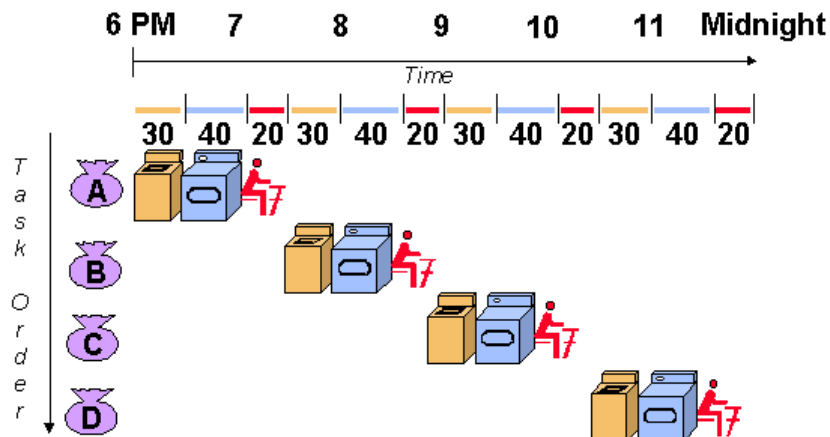


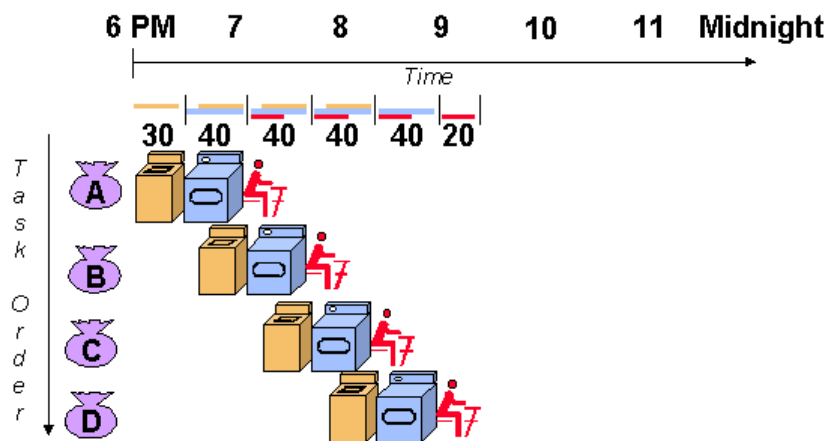
M12 Pipeline CPU

#technicke_vybaveni_pocitacu

- výhoda paralelizace
 - částečné zrychlení (obvykle $\times 2$ až $\times 3$)



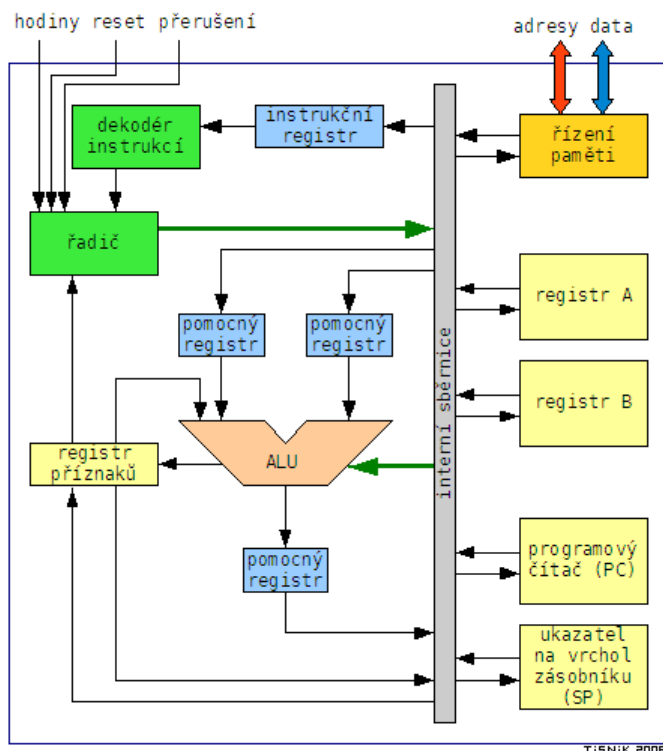
- před paralelizací



- po paralelizaci
- proč proudově pracující CPU?
 - využívá technologii pipeline k urychlení vykonávání instrukcí
 - umožňuje provádění několika instrukcí najednou v různých fázích zpracování
 - efektivnější využití zdrojů procesoru
 - zkrácení doby potřebné k vykonání instrukcí
 - má 4 fáze (*fetch*, *decode*, *execute*, *zásah do paměti*) - instrukce procházejí skrz tyto fáze postupně
 - mohou se vyskytnout problémy
 - je důležité zajistit předcházení konfliktům

Schéma pipeline CPU

Popis jednotlivých komponent a fází



Fáze plnění

- týká se způsobu jakým jsou instrukce vloženy do pipeline a jak jim prochází
- fetch - instrukce jsou načteny z paměti do pipeline
- decode - instrukce je přeložena na sérii mikroinstrukcí

provozu

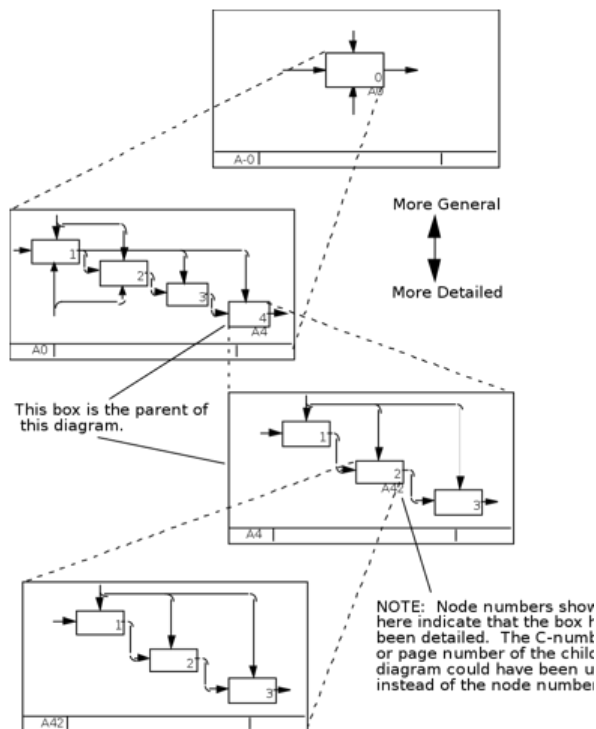
- **execute** - procesor vykonává instrukce
- memory - jen pokud instrukce vyžaduje přístup k paměti (čtení/zápis)
- write back - výsledek instrukce je zaslán zpět do registru či paměti

vyprazdňování

- vztahuje se k situacím, kdy je nutné náhlé přerušení nebo zastavení běhu pipeline (např. změna toku instrukcí) nebo po dokončení instrukcí
- detekce skoku - pokud je podmínka splněna, instrukce v pipeline (načteny, nedokončeny) jsou "zahozeny"
- může způsobit zpoždění
- zahození instrukcí
 - zajišťuje, že neproběhnou neplatné operace
 - pipeline je zaplněna nulami
- aktualizace stavu procesoru - procesor je aktualizován na novou hodnotu PC na základě skoku nebo větvení; nové instrukce jsou načteny z nové pozice v programu
- zahájení nového provozu (plnění → provoz)

Dekompozice systému

- rozklad složitého problému na více menších jednodušších
- ve strukturovaném programování *algoritmická dekompozice* rozkládá proces na dobře definované kroky
- v objektově-orientované dekompozici je rozklad řešen rozkladem velkého systému na progresivně menší třídy nebo objekty způsobil za nějakou část hlavního problému
- dekompoziční paradigma je strategie organizující program jako počet částí a určuje jak bude text programu uspořádán; obvykle využíván k optimalizaci programů pro vylepšení modularity nebo jeho udržitelnosti



Vliv na výkon

- specializované moduly navrženy pro efektivní řešení konkrétních úloh
- dekompozice usnadňuje správu a údržbu systému → aktualizace a záplaty mohou být snáze implementovány
- umožňuje přenositelnost a škálování systému; jednotlivé komponenty mohou být jednoduše vyměněny nebo rozšířeny
- neefektivní komunikace a synchronizace mohou ovlivnit rychlost zpracování

Konflikty

skokové

- vzniká v situacích kdy je třeba skočit na jinou část programu ale instrukce skoku je závislá na výsledku nějakého předchozího výpočtu; některé RISC arch. mají zpožděné skoky (skok je proveden po následujícím instrukčním cyklu) → pokud je v pipeline zahrnut další skok nebo větev během tohoto zpoždění, může dojít k konfliktu
- při konfliktu je třeba pipeline vyprázdnit a znovu naplnit novými instrukcemi na základě skoku
- řešení
 - předběžné zpracování skoku
 - předpovídá výsledek skoku
 - pipeline se začne plnit v předpokladu, že skok proběhne; správná předpověď = pokračování dál; špatná předpověď = vyprázdnění pipeline
 - vykonání instrukce mimo pořadí - za určitých podmínek vykoná další instrukce, dokud se nevyřeší skok
 - spekulativní vykonání - instrukce jsou prováděny na předběžném odhadu; špatný odhad = provádí se korekce

datové

- vzniká když dvě nebo více instrukcí soutěží o přístup ke stejným datům nebo registrům a alespoň jedna z nich provádí zápis
- Read after Read
 - když instrukce čte hodnotu z registru a následující instrukce čte stejný registr
 - není problém pokud první instrukce neprovede zápis
- Write after Read
 - když jedna instrukce čte registr a následující instrukce provede zápis do stejného registru
 - konflikt nastane když první instrukce nedokončila čtení a druhá začne přepisovat
- Write after Write

- když více instrukcí provádí zápis do stejného registru
- procesor musí zajistit, aby se zápisy do registru staly v pořadí; aby žádná instrukce nezapsala novou hodnotu před dokončením zápisu předchozí instrukce
- řešení
 - předběžné načítání
 - předpovědění výsledku
 - použití virtuálních registrů

Vliv na výkon

- umožňuje paralelní zpracování instrukcí → vyšší propustnost procesoru; každá pipeline může pracovat na jiné instrukci
- instrukce se zpracovávají ve fázích → nová instrukce se může začít zpracovávat dříve než se dokončí celý cyklus
- různé fáze mohou pracovat s různými instrukcemi současně
- skoky a závislosti mohou způsobit čekání na dokončení ostatních fází, což sníží rychlost CPU
- konflikty při zápisu a čtení mohou vyžadovat přerušení řešené člověkem
- správa pipeline a optimalizace jej může vyžadovat složité techniky a nástroje, což může zvýšit náklady na návrh a výrobu CPU