Řadič

11. listopadu 2008
10:57

Vykonávání instrukcí

Instrukční cyklus

1) Načíst instrukci
2) Vykonat ji
3) Uložit výsledek /případně/

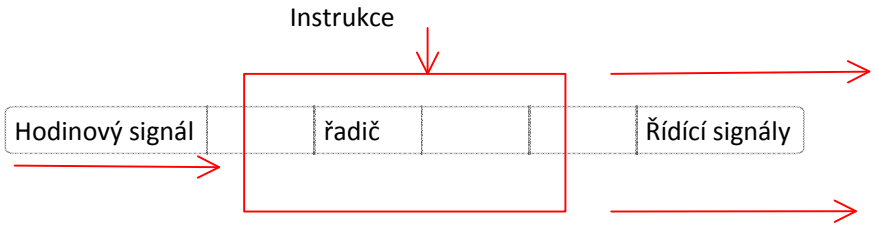
=> fragmentace úlohy na menší části - zjednodušení práce, méně náročné části můžou řešit části úloh

Cyklus 2

1. Zjistíme adresu instrukce, kterou vykonáme
2. Načteme ji (viz předchozí přednáška)
3. Dekódujeme operaci, abychom věděli co vlastne budeme dělat
4. Zjisime, kde leží data a načteme si je, pokud víc operandu načteme adresu, pak data a tak dokola dokud nemáme vše co je třeba (můžeještě být nepřímé adresování)
5. Pak samotné vykonání operace
6. Uložíme výsledek, vypadá stejně jako načítání.. Pokud víc výsledků, tak opakujeme, dokud neuložíme vše
7. Na konci ověříme jestli nedošlo k chybě (interruptu) - kontrola uz je az na konci, abych věděl kam přesně se mam vrátit, kdyby někde uprostřed, tak by to bylo nejasné
8. Pokud vektorová operace, tak se vrátím do fáze načítání operandů (4. bod) a opakuji dokud nedokončím vektor
9. Po vykonání se vracím zpět na začátek (1. bod)

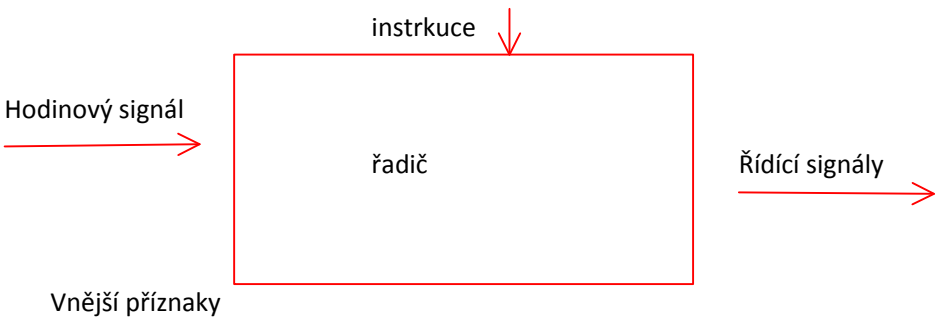
Pokud dojde k problému během cyklu, tak to zatím není řešeno

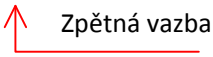
Základní schéma řadiče



Hodinový signál řídí kdy provádět
Tento řadič neví co se děje okolo, kromě své instrukce (např. semafory)
D**irektivní řad**ič.. Vždy rozhoduje bez ohledu na nic
Použití pro nejjednodušší úlohy

Zpětně vazební řadič





Stále velmi obecné, nepopisuje žádnou architekturu
Ale už se konkrátní řešení na tento model dá nacpat

Implementace řadiče

Jak je možno stroj s řadičem navrhnout

Hard-wired
- Přímo přenáší návrh stavového diagramu, pomocí sekvenční logiky,
- Používá hradla
- Ze stavového diagramu, který dělá nějakou opraci se vytvoří návrh který přímo popisuje HW
- Jde jen o to, vymyslet postup, protože obvodové řešení už je přímočaré
- Výhoda
  - Obvodové řešení přesně odpovídá schématu - nejsou žádné zbytečné součásti, proto i patřičně rychlé
  - Často se používá u strojů, kde se snaží o optimalizaci, protože toto řešení už ani rychlejší být nemůže.. Na úkor obecnosti (např. procesor risk)
- Nevýhoda
  - Žádná flexibilita, jakákoliv změna vyžaduje změnu návrhu a následně i HW a tedy i celé výroby
- Nepříjemné pro návrháře

Mikroprogramování
- Maurite v. Wilkes - jeho návrh: řídící signály produkované procesorem nebude přímo implementovat HW, ale uložím si je někam do paměti a udtud je budu brát..
  - Změnou obsahu řídící jednotky tedy budou i jiné řídící znaky.. = přeprogramování stroje
  - Přechod na tento mechanismus (např. IBM 370 se řídící signály načítali z pásky, při startu stroje se načetl mikrokód, který udělal jen testy stroje, a pak terve načetl mikrokód přímo pro ten stroj) ; (u VAX 11 bylo možné změnou kódu emulovat i úplně jiný stroj, pro jiné účely)
- Výhody
  - Můžu změnit instrukční sadu bez změny HE
- Nevýhody
  - Výrazně složitější návrh
  - Musím řešit obecnosti, tedy asi i na úkor rychlosti
- Vytvoření počítače uvnitř počítače--- v paměti máme instrukce které říkají co se m á dělat, takže zdánlivě jsme si nepomohli, ale zde máme plný přístup

Matrix A a B
- je řídící paměť
- Každá část ma jiný význam, jedna říká adresu pro mikroinstrukci

Provádění mikrokódu

Začneme tím, že mám zpracovat instrukci z vyšší úrovně
Z instrukčního registru vezmu instrukci, operační kód je index v tabulce
Operační kód může říkat adresu mikroinstrukce, tu si dekodér načte z hlavní paměti

Poslední mikroinstruce způsobí, že se do nstručního registrhu nahraje další instrukce

Podmínky se projevují jako podmíněný skok tak se použijí i slova do podmínky…

Řídící signály

Horizontální formát

Řídící signály si sepíšu, šířka je podle toho, kolik mám jednotlivých cílů řízení
Dost široké, ne vždy potřebuji všechny kombinace stavových kódu, některé se mohou navzájem vylučovat
Mikroinstrkuce obsahuje přímo hodnoty řídících signálů
Není třeba dekódovat => rychlost
Libovolná kombinace > pružnost

¨vertikální

Dekodér je výběr z tabulky, vstupuje index, vystupuje libovolné slovo délkou odpovídající tomu indexu
Není žádný vztah mezi počtem vstupů a výstupů
Některé operace se navzájem vylučuj -> možno zakódovat -> menši objem
Nutno dekódovat -> zpomalení
+ ušetřím pozice na řídícím slově, šířka
- dekódování
- Změnou dekodérů by se musel měnit návrh
Pevný návrh

Kdy vybrat jaký-- záleží na konkrétních požadavcích na ten konkrétní stroj

Nano programování

Dvou úrovňová řídící paměť
[viz prezentace]
Silná redukce velikosti, na úkor rychlosti programu
Mělo časté využití

Simple CPU

Do řadiče vstupuje čítač část T0 až T7 /maximálně potřebuji 8 mikroinstrukcí/
Pak vstupuje informace, jakou operaci mám vykonávat
A ten pak dává kontrolní signály
Řadič ví, v jaké fázi je, proto muže posílat konkrétní signály
Akumulátorová architektura - výpočty přes akumulátor

To šedé je sběrnice
PC - program counter, obsahuje adresu
MAR - adresový registr paměti
MDR - datový registr
ACC - akumulátor - hlavní pracovní registr
ALU - ze dvou vstupů umí udělat součet
A výsledek uloží do TEMP registru

Z adresy v PC načíst instrukci

Nesmím otevřít dvě brány na sběrnici, aby nedošlo ke konfliktu
Ale můžu víc bran, které ze sběrnice čtou

ACC_in - z internal busu uloží do akumulátoru

Implementace instrukce load:

- Instrukci načíst, vykonat, uložit výsledek a pak navýším PC o 1, abychom se dostali na další adresu instrukce
- Načteme z PC
- načtení
  - Fetch:
    - T0: otevřu brány PC_out , MAR_in
    - T1: read (řekneme paměti), pcincr (Navýšit PC mohu kdekoliv po T1¨,

uděláme to tedy hned jak to jde)
- ◇ T2: přečtu data z datového registru
  MDR_out, IR_in
  nyní data mám k dispozici v instručním registru

- Decode:
  - ◇ T3: decode [řadič řekne dekodéru, aby z toho co vidí na vstupu poslal ten správný příkaz load,add,store, brz] - dostanu load
- Execute:
  - ◇ T4: IR_out, MAR_in (poslu data do MAR)
  - ◇ T5: reknu pameti precist - read
  - ◇ T6: MDR_out, ACC_in -> vrátíme se zpět na T0 pro provedení další instrukce

Až podle dekódování zjišťujeme, co budeme dělat dal

Logicky popis řídících signálů
schéma všech instrukcí a popis kdy se do dělá
Snaha to mít co nejoptimálnější

Z toho můžeme dát dohromady řídící paměť

PIC12C5xx
Už reálný procesor