# D . Špecifikácia časovania

Korektné fungovanie číslicového systému v definovanom diskrétnom čase vyžaduje určitú **disciplínu** v časovaní rozličných udalostí (vrátane časovacích udalostí), t.j. ich **vhodné** vzájomnom rozmiestnenie na osi **spojitého času**.

**Disciplína časovania** sa opisuje v špecifikáciách rozličným spôsobom. V týchto prednáškach používame opis pomocou tzv. **časovacích pravidiel**. Uvedieme pravidlá, ktoré budeme používať. Ide o **predikáty** (ktoré sú pravdivé alebo nepravdivé, t.j. majú logickú hodnotu 1 resp. 0). **t(e)** značí čas udalosti **e**.

## *1. Pravidlá o oneskoreniach príčinne viazaných udalostí*

**del(e1, e, d) <=> t(e) = t(e1) + d**
**del*m*([e1,e2,...,er], e, [d1,d2,...,dr]) <=> t(e) = *m*(tm(e1)+d1, ..., t(er)+dr),**

kde **[e1,e2,...,er ]** je zoznam tzv. **vstupných** udalostí, **e** je tzv. **výstupná** udalosť, **d** a **[d1,d2,...,dr ]** sú kvantitatívne **časove parametre** – reálne (racionálne) čísla, ,viazané na vstupné udalosti, a "***m***" značí typ výberovej funkcie; ***m*** ε {***min***, ***max***}.
Pri funkcii "***min***" sa vyberá **najbližší** a pri "***max***" **najvzdialenejší** čas výskytu výstupnej udalosti **e**. Časovacie pravidlá vyjadrujú **oneskorenie** (angl.: delay) výskytu výstupnej udalosti e oproti vstupnej udalosti, pričom výstupná udalosť je **príčinne viazaná** (!) na vstupné udalosti.

Ak uvedené predikáty majú hodnotu 1 (sú pravdivé), potom medzi vstupnými a výstupnými udalosťami platia časove relácie na pravej strane implikácii. Tieto pravidlá pevne stanovujú časové body výskytu výstupných udalostí vzhľadom na časové parametre "**d**" (vo forme tvrdej rovnosti).

## 2. *Obmedzovacie pravidlá (obmedzenia)*

**aft(e1, e2, d) <=> t(e2) >= t(e1) + d**
**bef(e1, e2, d) <=> t(e2) =< t(e1) – d**
**afto (e1, e2, d) <=> t(e2) > t(e1) + d**
**befo(e1, e2, d) <=> t(e2) < t(e1) - d**

Je zrejmé, že platí: **bef(e1,e2,d ) ≡ aft(e2,e1,d)**
**afto (e2,e1,d) ≡ befn(e1,e2,d)**

Časovacie pravidlá typu **aft** a **bef** viažu čas výskytu dvoch udalostí s parametrom "**d**" vo forme **nerovnosti**. Pravidla **aft** a **bef** teda vyžadujú, aby sa udalosť **e2** vyskytla **po** (after) resp. **pred** (before) udalosťou **e1** so vzdialenosťou (na časovej osi) aspoň (res. väčšou ako) **d**. Pri **afto** a **befo** ide o **ostré** (angl. sharp) nerovnosti.
**Poznamenávame**, že časovacie pravidlá bef a befo sme zaviedli iba z praktickych dôvodov. Po zavedení pravidiel aft a afto sú pravidlá typu bef **redundantne.**

## *3. Pravidlá o stabilite signálu* v analyzovanom intervale „Int“

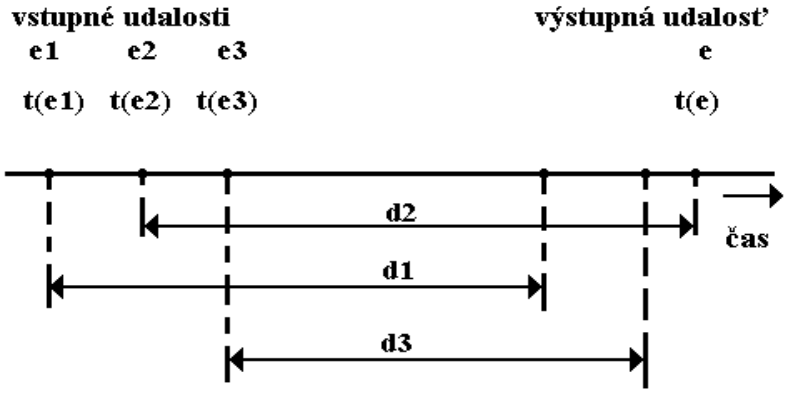
**stb(X=v) <=> v každom čase ma X stabilnú hodnotu <X>=v**
**stb(X) <=> detto, ale premenná X ma nešpecifikovanú stabilnú hodnotu**

**stbi(X=v, e, d1, d2) <=> pre všetky t <t(e)-d1, tm(e)+d2) je <X>=v**
**stbi(X, e, d1, d2) <=> detto avšak X má nešpecifikovanú stabilnú hodnotu**
**stbe(X=v, e1, e2) <=> pre všetky t <t(e1), t(e2)): <X>=v**
**stbe(X, e1, e2) <=> detto len X má nešpecifikovanú stabilnú hodnotu**

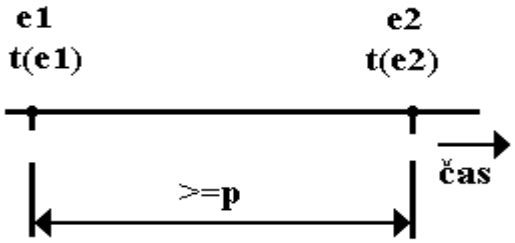
Posledné štyri pravidlá sú **redundantné**. Možno ich nahradiť pravidlami typu aft a afto. Zaviedli sme ich iba z praktických dôvodov jednoduchosti zápisov.

**Pravidlá sú ilustrované** na nasledujúcich obrázkoch

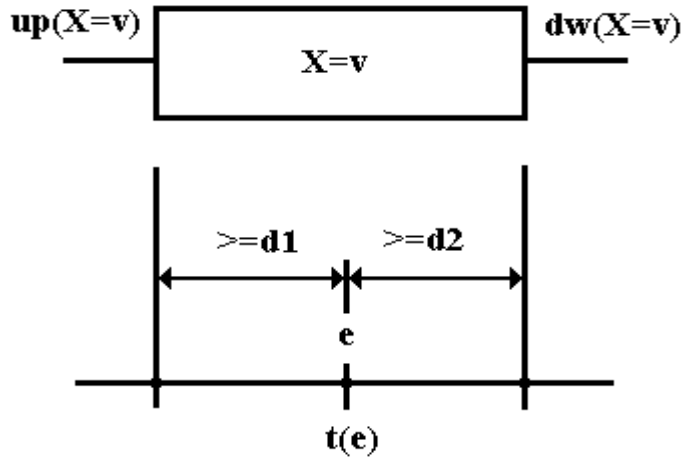
@ **del*max* ( [e1, e2, e3], e, [d1, d2, d3] )**



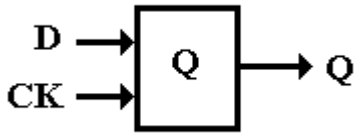

@ **aft(e1,e2, p)**



@ **stbi(X=v, e, d1, d2)**




**PRÍKALD 8:** Na obr. sú naznačené vstupy a výstupy **záchytného** preklápacieho obvodu (latch). Všetky premenné sú boolovské. Výstupná premenná Z je zároveň jedinou stavovou premennou. V tomto systéme definujeme agent **Nastav** takto:




Dátové typy
**DV={(<D>,<CK>},**
**DH = {<Q >} = {0,1}**
**SI ={ <Z>} = {0,1}**

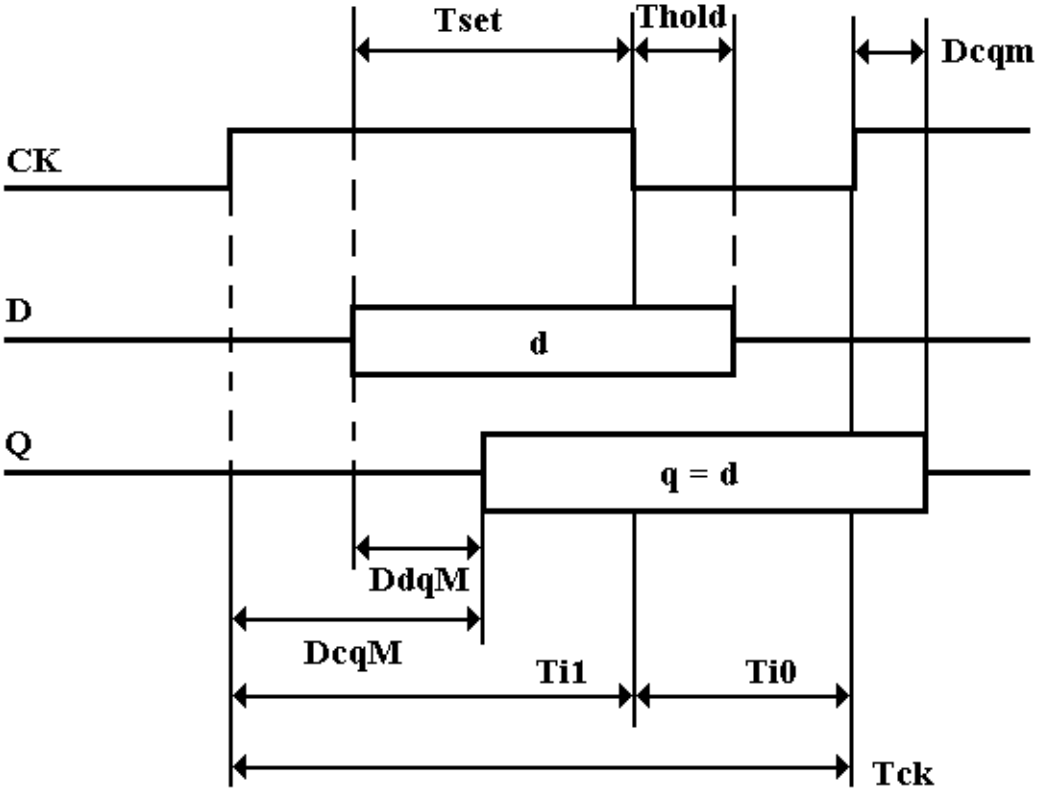
Časovacie udalosti **{ up(CK=1), up(D=d), up(Q=d), dw(Ck=1), ef }**

KM [ **(D=d, CK=0; ; u)+(D=d, CK=1; ;Q=d) ]( CK=0, D=d; ;Q=d)(u; ef ;Q=d)**
g **Q := d**

**Poznámka: X = a,** kde **X** je **vtupný port** značí, že časovacia udalosť je daná **up(X=a)**.

Časovacie pravidlá
**stbi(D, dw(CK=1), Tset, Thold)** // stabilita D = d v okolí dw(CK=1)
**aft (up(CK=1), dw(CK=1), Ti1)** // vyžadovaná dĺžka impulzu 1 v CK
**aft(up(CK=1, j), (up(CK=1, j+1)), Tck)** // vyžadovaná najktratšia dĺžka cyklu CK
**delM( [up(CK=1), up(D=d)], up(Q=d), [DcqM, DdqM] )** // max oneskorenie – obr.
**afto(up(Q=d), dw(CK=1, 0**) // dodržanie fundamentálneho režimu

kde Tset, Thold sú známe časy **predstihu** a **presahu** vstupného údaja (**d**) pred resp. za udalosťou **dw(CK=1)**, t.j. pred resp. za zadnou hranou impulzu 1 v **CK**; **Ti1** a **Tck** sú minimálne časy trvania impulzu 1 resp. vzdialenosti po sebe nasledujúcich zmien CK z 0 na 1 (medzi udalosťou **up(CK=1, j)** a udalosťou **(up(CK=1, j+1)**); DcqM a DdqM sú maximálne časy oneskorení zmeny výstupnej premennej **Z** (**up(Z=d**) vzhľadom na zmeny vstupných premenných **CK** a **D** (**up(CK=1)** a **up(D=d**));




Časovacie pravidlá **doplňujú** špecifikáciu systému. V danom príklade záchytného obvodu sme použili jeden agent **Nastav.** Časovacie pravidlá spojené s týmto agentom tvoria jeho súčasť. Vkladáme ich do agentov, keď je to potrebné, teda ak chceme rozšíriť (spresniť) špecifikáciu s cyklov - z diskrétneho času - na spojitý čas.

V ďalšom, pri **formálnom opise agentov**, ak zahrnieme do nich aj množinu **TR** časovacích pravidiel (angl.:Timing Rules), agent **A** opíšeme ako formálny systém

**A** = $_{def}$ (**S**, **SI**, **KMM**, **Q**, **g, TR**)

**KSM** = (**SI**, DV, DH, **R** ,**vs**),

kde **TR** je množina časovacích pravidiel.

Pravidlá sú definované pri určitých **časových parametroch**. Vo všeobecnosti tieto parametre môžu byť funkcie rozličných fyzikálnych parametrov (ako sú napr. čas, teplota, počet rozvetvení - "fan-out" a pod.). Môžu mať:

- pevnú, známu číselnú hodnotu,
- hodnotu z daného intervalu <dmin, dmax>, kde dmin a dmax sú hranične hodnôt parametra
- symbolickú hodnotu p, ktorá sa pri vývoji špecifikácie nahradí reálnym číslom

# E. Štandardný zápis špecifikácie systému s agentmi a procesmi

## *Zápis agentov*

Pri zápisoch agentov v špecifikáciách systémov podľa nášho modelu pri opise položiek agentov a behavioristickej špecifikácie digitálneho podsystému ako celku, budeme používať určitý systém (rámec).

Preberané položky agenta označujeme nasledujúcimi symbolmi:

**SI**.. zápis množiny **SI začiatočných stavov** vyjadrený pomocou začiatočných hodnôt stavových premenných (napr.<Zj>=d) alebo charakteristickým logickým výrazom, ktorý jednoznačne definuje stavy v SI

**TE**.. opis množiny TE typov **časovacích udalostí** v komunikačných výrazoch (formulách)

**KM**…opis **komunikačnej množiny** agenta výrazom – **komunikačnou formulou** s trojicami typu (v; e ; h). Ak je KM zapísaná v tvare komunikačnej formuly, v ktorej sú iba **mená** akcií namiesto samotných akcií (známych trojíc); **potom musí nasledovať deklarácia akcií**

**akcia** meno akcie opis trojice
**akcia** meno akcie opis trojice

**vs**...výrazy určujúce symbolické hodnoty **primárnych výstupov** (výstupných portov) systému v komunikačných slovách

**g**….priraďovacie výrazy určujúce hodnoty **lokálnych stavových premenných** (v budúcej operačnej časti) na konci exekúcie agenta

**TR**... zápis **časovacích pravidiel**

Nepotrebné položky obyčajne vynechávame (tak napr. vynecháme **SI** ak **SI = S** (kde **S** je globálna množina stavov **RxQ**)alebo od začiatočného stavu sú výstupné slová v komunikačnom výraze nezávislé, alebo vynecháme **vs** ak hodnoty výstupov sú už uvedené komunikačných slovách a pod.).

Predpokladáme, že opis množín DV a DH ako aj množiny operačných stavov **Q** systému, ktoré sú v svojej podstate určené množinami primárnych vstupov, primárnych výstupov, resp. lokálnych stavových premenných a príslušnými údajovými typmi, **sú dané v opise systému ako celku**.

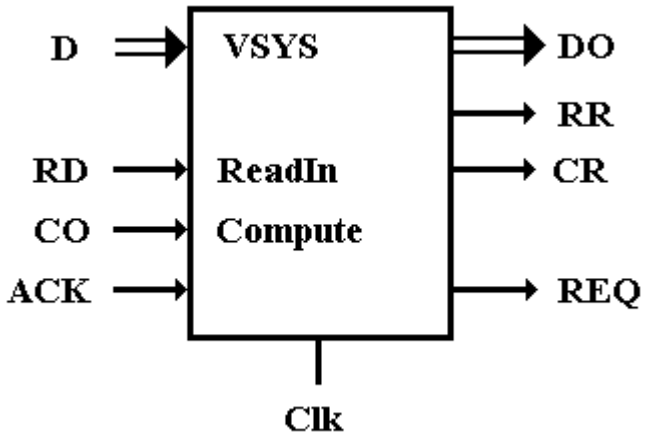
Pri označení položiek sme použili skratky odvodené z anglických názvov: state, (TE) timing events, (TR) timing rules.

Agent môže byť zadaný **generickým** spôsobom, t.j. napr. ako

**agent** meno agenta (..,$x_i$,..;...,$y_j$,..;...,$z_r$,..;...,$T_k$ ),

kde x, y a z sú generické vstupné, výstupné a stavové premenné a T sú časové parametre daného generického agenta.

**PRÍKLAD 3 (pokračovanie**): Uvedieme zápis agenta **ReadIn** systému **VSYS**




agent ReadIn

SI <RR> = 0 // vsetky stavy z Q, pri ktorych je <RR>=0

TE es, up(CLK=1), ef // es je zaciatocná casovacia udalost

KM (RD=1;es;REQ=0)(ACK=1;;REQ=0)$^*$[(ACK=0;;REQ=0)(ACK=0;REQ=1)$^+$

.(ACK=1, D=$d_j$ ; ; REQ=1)( ACK=1, D=$d_j$ ; ;REQ=0)(ACK=1; ;REQ=0)$^*$]$^{1\text{-}16\ (D)}$

. (u; ef; RR=1)

// hodnoty 0 alebo 1 relevantnych vystupov REQ a RR su

// priamo uvedene v komunikacnom vyraze KM vs vynechame

g

M := $d_1$,...,$d_{16}$;

RR := 1

TR

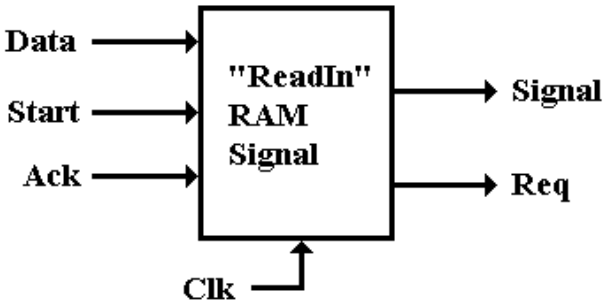
del(up(RD=1), up(RR=1), $T_{RD}$);

befo(up(RR=1), up(M=$d_1$,...,$d_{16}$),0);

afto(up(RR=1),ef,0)

Tento agent ReadIn môže byť zadaný ako generický (všeobecný) so všeobecnými premennými a parametrami.

Uvedieme ešte príklad **generického** spôsobu zápisu agenta **ReadIn**




agent ReadIn (Start, Data, Ack, Clock; Req, Signal; RAM,Signal; $T_{StSi}$)

SI Signal = 0

TE es, up(Clk=1), ef; //"es" a "ef" sú standardné
// casovacie udalosti a Clk je
// ctanardné meno pre hodiny

KM

(Start =1;es;Req=0).(Ack=1; ;Req=0)$^{*}$[(Ack=0 ;;Req=0).

.(Ack=1, Data=$d_j$ ; ; Req=1)$^{+}$.(Data=$d_j$, Ack=1; ; Req=1)

.(Ack=1, Data=$d_j$ ; ; Req=0 ). .(Ack=1; ; Req=0)$^{*}$]$^{1\text{-}16(Data)}$.(u; ef; Signal=1);

g

RAM := $d_1$,...,$d_{16}$;

Signal := 1;

TR

del(up(Start=1), up(Signal=1), $T_{StSi}$);

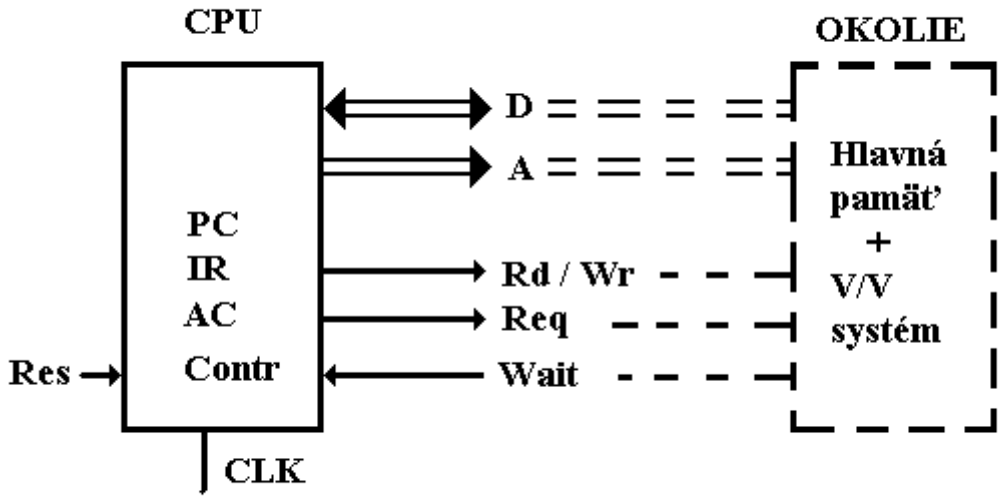
befo(up(Start=1), up(RAM=$d_1$,...,$d_{16}$),0);

afto(up(Signal=1),ef,0);

Pri vytvorení konkrétnej inštancie agenta pri aplikácií sa potom uvedie s príslušným mapovaním premenných a časových parametrov ako

ReadIn(RD, D,ACK, CLK; REQ, RR; M, RR; $T_{RD}$)

**PRÍKLAD 4**: Procesor **CPU**, agent **Agent-1** (čítanie inštrukcie)




agent Agent-1
SI
Contr = RES;
TE
es, up(CLK=1), ef;

KM
(Res=1; es; u)$^{*}$.(Res=0; ; u).(Wait=0; ; u)*.(Wait=1; ; A=PC, RdWr=1, Req=1)$^{+}$
.(Wait=0, D=d ; ; u).(Wait=0 ; ; u)$^{*}$.(Wait=1; ;ef ; u);

g
IR := d;
PC := PC+1;
Contr := OD;

***Zápis Špecifikácie Systému***

V používanom behavioristickom špecifikačnom modeli budú nasledujúce položky:

**DT**……špecifikácii údajových typov

**PORTY**…deklarácia vstupných a výstupných portov
**vstup** deklarácie vstupných portov
**vystup** deklarácie výstupných portov
**OPER STAV**….deklarácia lokálnych **stavových** premenných ktorých vektor hodnôt tvorí operačný stav . Ak sa premenná z tejto položky nachádza aj v položke porty, tak sa to interpretuje tak, že táto premenná je lokálna stavová a zároveň primárna výstupná

**TR** zoznam časovacích pravidiel platných pre špecifikovany system **ako celok**

Nasleduje deklarácia procesov
**proces** meno procesu = opis procesu výrazom ;
**proces** meno procesu = opis procesu výrazom;
Nasleduje deklarácia agentov
**agent** meno agenta opis agenta;
**agent** meno agenta opis agenta;

**START** // štartovacie mechanizmus pre agenty a procesy vo forme
// predikátov „štartovacích výrazov**"** a „obmedzení" (reštrikcií)

**meno procesu** štartovací výraz;
**meno procesu** štartovací výraz;
**:**
**:**
**meno agenta** štartovací výraz;
**meno agenta** štartovací výraz;
**:**
**:**

**restr** // je množina predikátov, nazývaná obmedzeniami
// (reštrinkciami) definovaných na premenných systému
// platiaca v celom systéme.
**predikat**;
**predikat**;
**:**
**:**

**Štartovací** výraz ktorý zodpovedá **prve**j, t.j., **štartovacej časovacej** udalosti **es** v komunikačnej množine **KM** (výraze pre **KM**) prvého agenta v procese **P**, resp. ak je na začiatku procesu P paralelná kompozícia agentov, alternatíva behu agentov, alebo výberová alternatíva agentov, potom **es** je vo všetkých agentoch týchto kompozícií v **KM**.

V položke **reštrikcie** (**restr**) sa môže nachádzať aj jeden alebo viacej špeciálnych tzv. **procesných predikátov**, v ktorých ako premenné vystupujú mená procesov (agentov) systému. napr. **c**(P1,P2). Jednotlivé premenné (mená procesov alebo agentov, napr. P1 a P2 v danom predikáte c, predstavujú tiež predikáty, ktoré sú pravdivé, ak príslušný proces (agent) je práve vo vykonávaní. V každom predikáte (napr. **c**(P1,P2)) je menami uvedená podmnožina procesov (agentov), ktoré sú tzv. kompatibilné. **Kompatibilné** nazývame procesy (agenty), ktoré sa môžu v systéme vykonávať paralelne (súbežne).

Tu upozorňujeme, že **agent** chápeme ako **triviálny proces** s jedným prvkom. Priradiť agentovi štartovací výraz vlastne znamená to isté ako priradiť štartovací výraz procesu, ktorý obsahuje iba tento (jeden) agent.

Ak napr. v časti **restr** sa nachádza predikát **c**(P, P'), potom proces P môže prebiehať v systéme **súbežne** s procesom P' a naopak. Teda štartovacím výrazom možno spustiť aj P' ak P bol už spustený a beží alebo nebol spustený (pozri obrázok nižšie) a to isté platí pre proces P ak beží P'. Pre vyhodnocovanie predikátu **c** zrejme platí:

**c(Pi1,Pi2,…,Pik) ↔ or [ nor(Pi1, Pi2, …., Pik), or (Pi1, Pi2, …., Pik) ]**

Je však prirodzené, že procesy musia spĺňať **podmienky korektnosti exekúcie paralelných (súbežných) procesov** vzhľadom na nastavovanie a čítanie hodnôt spoločných výstupných a stavových premenných musia byť v špecifikácií zabezpečené (napr. pomocou známych semafórov)

**Štartovacie výrazy,** sú logické predikáty (formuly) obsahujúce ako svoje premenné – **predikáty** a **udalosti** definované na digitálnom systéme ako celku. Jednou z udalostí môže byť aj udalosť "**ez**", **univerzálna štartovacia udalosť**, ktorá sa objaví vždy na začiatku činnosti systému S (pri pripojení energie).

Štartovací výraz predikát**,** ktorý môže nadobudnúť pravdivú hodnotu (hodnotu1) iba ak niektorá premenná tohto výrazu, ktorá zodpovedá udalosti (tzv. **udalostná premenná**) je pravdivá, teda nadobudne hodnotu 1.

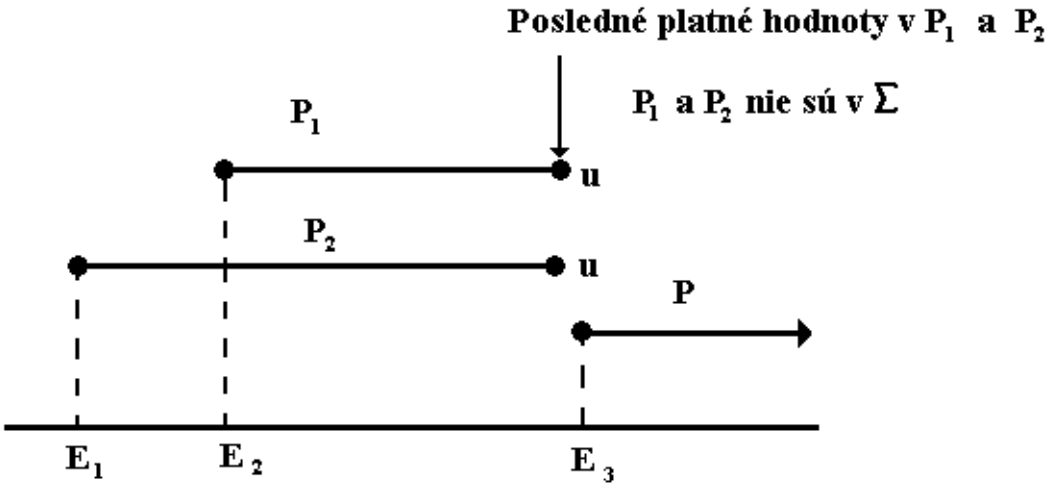
Udalostnej premennej sa pritom formálne prisudzuje hodnota 1 práve len v jednom d-bode času, v ktorom nastane. Syntax výrazov vyplýva priamo z uvedenej definície.

***Interpretácia*** (sémantika) práve zavedených položiek pri agentoch a procesoch je zrejmá a bola už vlastne opísaná predtým v našom modeli spracovania informácie. Ostáva ešte spresniť interpretáciu **štartovacieho mechanizmu START.**

- Treba pripomenúť skutočnosť, že ak sa niektorý P' nachádza v procese P a proces P bol naštartovaný, potom P' je automaticky naštartovaný (**z vnútra**) procesu P a to vtedy, keď **príde na rad** podľa daného spriahnutia. Je vidno, že procesy môžu byť naštartované v inom proces z **vnútra** procesu. Pri implementácií systému štruktúrnym HW spôsobom sa exekúcia procesu P_naštartovaná **z vnútra** tohože procesu P **nepripúšťa.**
- Štartovací mechanizmus **START** vyjadruje spôsob štartovania procesov **zvonku** (pomocou udalostí a predikátov).Vyjadruje to prostredníctvom **štartovacích výrazov**, ktoré **START** priraďuje procesom, pričom sa rešpektujú obmedzenia v položke **restr**.
- Systém "funguje" od d-času **0,** v ktorom nastane t(**ez**), kde **ez** je udalosť na spojitej časovej osi, ktorá signalizujúca **nabehnutie energie** do ustáleného stavu. Štartovací výraz môže podľa definície nadobudnúť hodnotu 1 iba v **jednom diskrétnom bode** spojitého času. Ak tento výraz nadobudne hodnotu 1 ( je pravdivý), určí sa tým **d-bod** času, v ktorom sa **začne** vykonávať príslušný proces (prvý jeho agent, resp. jeho prvé, paralelne bežiace agenty) Zodpovedá mu **aj prvý bod** d-času naštartovaného procesu.
- Naštartovanie procesu závisí aj od predikátov v položke **restr** (reštrikcia). Proces (agent) P sa **spustí** (naštartuje) práve vtedy, ak každý predikát p ε **restr** je pravdivý (p=1). Hovoríme, že sú splnené obmedzenia. Ak sa položke **restr** nenachádza procesný predikát c(P,P',P'',….) , ktorý by obsahoval meno procesu (agenta) P spolu s inými procesmi P',P'',… , ktoré sa môžu vykonávať súbežne, tak potom keď sa má P spustiť, tak **sa spustí** nezávisle od toho či **je** alebo **nie je** v predikáte c.

Avšak, ak **nie je** (a teda nemôže bežať paralelne s P', P'',…) a práve sa vykonáva jeden alebo viacej procesov zo zoznamu P',P'',…, tak tieto bežiace procesy **predčasne skončia**.(pozri obr. dole)

Na obrázku je znázornená situácia, pri ktorej sa v položke **restr** nachádza **iba jeden** procesný predikát c(P1,P2), kde P1 a P2 sú mená dvoch procesov systému. Beží P1 ( bol spustený v štartovacím výrazom E1), potom E2 spustí P2. Obidva procesy sa môžu podľa c(P1,P2) vykonávať a aj sa vykonávajú súbežne. Pri naštartovaní procesu P štartovacím výrazom E3 vykonávanie oboch bežiacich procesov P1 a P2 sa musí však predčasne skončiť, pretože v **restr** sa nenachádza predikát c'(P, P1, P2) a P sa nemôže vykonávať v systéme súbežne s P1 a/alebo P2 ale tiež preto, lebo spustenie P **má prednosť**. Stavové a výstupné premenné predčasne zakončených procesov P1 a P2, ktorých hodnoty sa v nich nastavujú, ostanú **po tomto bode nešpecifikovane** - "u".




Zápis špecifikácie číslicového systému pomocou práve opísaného rámca uvedieme na príkladoch. Tu uvedieme aj určitú schému zápisu, ktorú budeme v prednáškach používať. Použijeme pritom už uvedenú schému zápisu agentov.

**PRÍKLAD 3** (pokračovanie): Uvedieme špecifikáciu systému **VSYS**

system VSYS

ÚDAJOVÉ TYPY

[udaj ∈ $< -2^{15}, +2^{15}-1 >$;
operacia "+" v doplnkovom kode];
pole U(0),...,U(15) udaj;
boolovska hodnota ∈ {0,1};

PORTY

| | | |
|---|---|---|
| vstup | D | udaj; |
| | RD,CO | booleovska hodnota; |
| výstup | DO | udaj; |
| | RR,CR | booleovska hodnota; |

```
OPER  STAV
            M           pole M(0),...,M(15)  udaj;
            RR,CR    booleovska hodnota;

 agent    ReadIn
    SI   RR = 0; //všetky stavy z Q, pri ktorých je <RR>=0
    TE   es, up(CLK=1), ef;          // es je štartovacia časovacia udalosť
    KM
         (RD=1;es;REQ=0)(ACK=1;;REQ=0)*[(ACK=0;;REQ=0)
         (ACK=1;;REQ=1)+(D=dj,ACK=1;;REQ=1)(D=dj,ACK=1;;REQ=0)
         (ACK=1;;REQ=0)*]1-16(D)(u;ef;RR=1);
    g
          M := d1,...,d16;
          RR := 1;
    TR       del(up(RD=1), up(RR=1), TRD);
             befo(up(RR=1), up(M=d1,...,d16),0);
             afto(up(RR=1), ef, 0);

 agent    Compute
     SI  CR = 0;
     TE    es,ef;
     KM
          (u; es; u)(u; ef; DO=do, CR=1);
     vs     do = M(0) + ...+ M(15);
     g      DO := do;
            CR := 1;
     TR
            del(up(CO=1), up(CR=1), TCO);
            befo(up(CO=1),up(DO=do),0);
            afto(up(CR=1),ef,0);                  // up(CR=1) je signal
                                                  // zakoncenia vypoctu
                                                  // TCO ie čas potrebný
                                                  // na vykonanie Compute
          // vo VSYS sú dva triviálne procesy, každý s  jedným agentom
          // ReadIn resp. Compute. Procesy tu teda neuvádzame

START
            ReadIn      up(RD=1);
            Compute    up(CO=1);
      restr     nand (RD,CO);
                afto( up(CO=1), up(RD=1), TCO) ;
                afto( up(RD=1), up(CO=1), TRD);
// Vzhladon na to, ze v „restr“ (pozri nnizsie) nie je procesny predikat
// c(ReadIn, Compute) agenty sa nemozu vykonavat subezne
```

**Alternatívne** možno VSYS špecifikovať inými položkami **proces** a **START** aj takto:

```
   Proces P1 = [(RD=1): ReadIn, (CO=1): Compute
                ((RD=0) and (CO=0)):EA]^ω;       // nekonečný proces
START
        Proces  P1      ez;
      restr          nand (RD, CO);
                     afto( up(CO=1), up(RD=1), T_CO) ;
                     afto( up(RD=1), up(CO=1), T_RD);
```

**PRÍKLAD 4:** (pokračovanie) Uvedieme špecifikáciu procesora s menom **CPU**.

```
system    CPU
DT    udaj        [cele cislo ε < -2^31, + 2^31-1 >;
                   operacia  "+" v doplnkovom kode];
      adresa      [kladné cislo ε <0, 2^30-1>;
                   operacia "+" v module 2^30 ] ;
      opcode       symbol  ε {ADD, BRN, LD, ST};
      instrukcia   record
                      opcode,
                      adresa;
       r_stav      symbol  ε { IF, OD};
       signál      boolovska hodnota ε {0,1};
PORTY
      vstup      D        udaj,  instrukcia    // vstupom D sa prenasaju obdva DT
                 Wait     signál;
      výstup     D        udaj,                // D je obojsmerny vstup - vystup
                 A        adresa,
                 Req      signál;
OPER STAV
                 PC       adresa,
                 IR       instrukcia,
                 AC       udaj,
                 Contr    r_stav;
 proces    Instcyclus = ResOper .[InstrVyb.( (IR.opkód=BRN):BRNop
                                             (IR.opkód=AD): ADDop,
                                             (IR.opkód=LD): LDop,
                                             (IR.opkód=ST): STop )]^ω

   agent  InstrVyb
      SI    Contr = IF;
      TE    es, up(CLK=1), ef;
      KM      (u;es;u).(u; ;u)(Wait=0; ;u)*.(Wait=1; ;u).(Wait=1; ;A=PC, Req=1, RdWr=1)*
              .(Wait=0, D=d ; ; A=PC, Req=1, RdWr=1).(Wait=0;;u)*.(Wait=1;;u).(u; ef; u);

      g           IR := d;     // IR obsahuje inštrukciu v IR.opkód a
```

```
                PC := PC+1;  // adresu v IR.adresa
                Contr := OD; // OD ynačí "Operation Decode"

agent   BRNoper
   SI   Contr = OD;
   TE   es, up(CLK=1), ef;
   KM    (u; es; u).(u; ef; u);
   g        ak AC < 0 tak  PC := IR.adresa inak PC := PC + 1;
                           Contr := IF;
agent   ADDoper
   SI   Contr = OD;
   TE    es, up(CLK=1), ef;
    KM      (u; es; u)(Wait=1; ; u).(Wait=, D=d0; ;A=IR.adresa, Req=1, RdWr=1)*
            .(Wait=0, D=d; ; A=IR.adresa, Req=1, RdWr=1).(u; ef; Req=0);
    g          PC := PC + 1;
               AC :=  AC + d;
               Contr := IF;
agent   LDoper
   SI   Contr = OD;
   TE    es, up(CLK=1), ef;
    KM  .   (u;es;u).(Wait=1;;u).(Wait=1,D=d0;;A=IR.adresa,Req=1,RdWr=1)*
    .(Wait=0,D=d; ; A=adresa, Req=1, RdWr=1).(u; ef ; Req=0);
    g          PC := PC + 1;
               AC := d;
               Contr := IF;
agent   SToper
   SI   Contr = OD;
   TE   es, up(CLK=1), ef;
   KM      (u;es;u).(Wait=1;;u).(Wait=1;;A=IR.adresa,D=AC,RdWr=0,Req=1)*
           .(Wait=0; ef; A=IR.adresa, D=AC, RdWr=0, Req=1)
   g           PC := PC + 1;
               Contr := IF;
agent   ResOper
   TE    es,ef;
    KM    (u;es;u)(u;ef;u);
    g       PC := 0;
            Contr := IF;
START
            InstrCyclus          (ez or ((up(Clk=1) and Res)) ;
```

**PRÍKLAD 5:** Ako ukážku uvedieme príklad zápisu špecifikácie **VSYS** v jazyku **HSSL** vypracovanom na **STU FIIT**, založenom na v týchto textoch použitom modeli. Ide o rozpracovanú behavieristickú špecifikáciu v procese vývoja (zjemňovania) **smerom k** špecifikácií na úrovni **RT**. Nachádza sa tu agent **ReadIn** a zjemnený agent **Compute** (pozri Kap.3) implementovaný procesom **Pcompute** = Comp_Reset.[Add; Incr_I]$^{(I=0)}$ **.**Comp_Ready. Definícia HSSL je koncepčne odvodená z jazyka C.  Pri znalosti nášho modelu je text  možno pochopiť.

Položka OPER STAV je opísaný pod **State Vars**; v špecifikáciách v agentoch **LV** deklaruje domény hodnôt lokálnych stavových premenných, ktorých „symbolické“ hodnoty sa používajú; „KM“ je pod **CM** (Communication Set), a „g“ je pod **FS** (Final States). V deklarácií „akcií“ v **CM** sú opisy vstupných a výstupných vektorov pod **iv** resp. **ov** a časovacia udalosť pod **te**. Ak v tomto opise chýba niektorá zložka akcie, tak nie je špecifikovaná – je „u“. „START“ je tu pod **Starting Structure**, „štartovací výraz“ sa uvádza pod **Stf** (Starting function).

```
System Vsys {
  const Trd = 65;
  const Tcomp = 500;

  typedef int:16 DATA;
  typedef DATA DATAARRAY[16];
  typedef unsigned:1 BOOL;
  typedef unsigned:5 RANGE_0_15;
  typedef clock(1,1) CLOCK;

  Inputs {
    DATA DI;
    BOOL RD,CO,ACK;
    CLOCK CLK;
  }

  Outputs {
    DATA DO;
    BOOL RR,CR,REQ,COMP;
  }

  State Vars {
    DATAARRAY M;
    DATA DO;
    BOOL RR,CR,REQ,COMP;
    RANGE_0_15 I;
  }
 Agent Readin {
    LV { DATA d[16];
         int:8 j,k;
    }
    IS { RR==0 }
    CS { start_rd. (req1 . d_in . req0 . ack0 )#(j=0:15) .rr }
    action start_rd { iv:RD=1, ACK=0; te:up(RD,1); }
    action req1 { iv:ACK=0; te:up(REQ,1); }
    action d_in { iv:d[j]=DI; te:up(ACK,1); ov:REQ=1; }
    action req0 { iv:ACK=1; te:up(REQ,0); }
    action ack0 { te:up(ACK,0); ov:REQ=0; }
    action rr { te:ef; ov:RR=1; }
    FS { M[k]=d[k] for (k=0:15); }
```

```
 TR {
     del(up(RD,1), up(RR,1), Trd);
          bef(up(RR,1), up(M), 0);
          aft(up(RR,1), ef, 0);
    }
  }
  Agent Comp_Reset {
    IS { CR==0 }
    DefaultTE: up(CLK,1);
    CS { start . reset_do }
    action start { iv:CO=1; ov:COMP=1; }
    action reset_do { ov:DO=0, CR=0; }
    FS { DO=0; I=0; CR=0; COMP = 1; }
  }
  Agent Incr_I {
    DefaultTE: up(CLK,1);
    CS { start . finalAction }
    action start { }
    action finalAction { }
    FS { I=I+1; }
  }
  Agent Add {
    LV {DATA dout;}
    DefaultTE: up(CLK,1);
    CS { start . d_out }
    OV { dout=DO+M[I]; }
    action start { }
    action d_out { ov:DO=dout;}
  }
  Agent Comp_Ready {
    DefaultTE: up(CLK,1);
    CS { start . comp_ready }
    action start { }
    action comp_ready { ov:CR=1 ;}
    FS { CR=1; COMP = 0; }
  }
  Process PCompute {
    Comp_Reset; do {Add; Incr_I} while (I<16); Comp_Ready
}
  Starting Structure {
    Stf Readin: up(RD,1);
    Stf PCompute: (CO) and (!COMP) and up(CLK,1);  // process
    Restrictions { !(RD and CO) }
  }
}
```

**PRÍKLAD 6:** Ukážka behavioristickej špecifikácie **VSYS** v jazyku **VHDL**.na úrovni **RT**. Systém je opísaný ako konečný stavový stroj.

```
--datove typy
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use types.all;

--entita
entity VSYS is
  port
  (
    p_D       : in    udaj;
    p_RD      : in    std_logic;
    p_CO      : in    std_logic;
    p_ACK     : in    std_logic;
    p_CLK     : in    std_logic;
    p_DO      : out   udaj;
    p_RR      : out   std_logic;
    p_CR      : out   std_logic;
    p_REQ     : out   std_logic
  );
end VSYS;

--architektura,  opis spravania
architecture behavior of VSYS is
  begin
  process(p_CLK)
     variable   stav : stavy := L0;
     variable   M:RAM;
     variable  DO:udaj;
     variable  I:Inc;
     variable  REQ:STD_LOGIC;
     variable  RR:STD_LOGIC;
     variable  CR:STD_LOGIC;
  begin
    if  (p_CLK'Event and p_CLK= '1')   then
      case Stav is
        when L0  => if p_RD = '1' then
                        stav := L1;
                         elsif p_CO = '1' then
                           stav := C1;
                         else
                           stav := L0;
                           DO   :="0000000000000000";
                           I    :="0000";
                           RR   :='0';
                           CR   :='0';
                           REQ  :='0';
                         end if;
```

```
  when L1  => if p_ACK = '1' then
                  stav := L3;
                   elsif p_ACK = '0' then
                     stav := L4;
                end if;
  when L3  => if p_ACK = '1' then
                  stav := L3;
                   elsif p_ACK = '0' then
                     stav := L4;
               end if;
  when L4  => REQ:='1';
                stav := L5;
  when L5  => if p_ACK = '1' then
                  stav := L8;
                   elsif p_ACK = '0' then
                     stav := L5;
               end if;
  when L8  => M(conv_integer(I)):= p_D;
                  I:= I + "0001";
                  REQ := '0';
                   if (I /= 0)and(p_ACK = '0')then
                     stav := L4;
                   elsif (I/=0)and(p_ACK = '1') then
                     stav := L3;
                   elsif I = 0 then
                     stav := L10;
                   end if;
  when L10 => RR:='1';
              stav:= L0;
  when C1  => DO   :="0000000000000000";
                   RR   :='0';
                   CR   :='0';
                          I:="0000";
                   REQ  :='0';
                   stav := C2;
  when C2  => DO := DO + M(conv_integer(I));

                     I:= I + "0001";
                   Stav := C3;
  when C3  => if I/=0 then
                     stav := C2;
                   elsif I = 0 then
                     stav := C5;
                end if;
  when C5  => CR := '1';
                stav := L0;
end case;
 p_DO   <= DO;
 p_RR   <= RR;
 p_CR   <= CR;
 p_REQ  <= REQ;
```

```
    elsif p_Clk='U' then
         DO   :="0000000000000000";
         RR   :='0';
        I    :="0000";
         CR   :='0';
         REQ  :='0';
    end if;
  end process;
end VSYS;
```

**PRÍKLAD 7**: Tu je ukážkový príklad špecifikácie **CPU** v **HSSL** ktorá leží **relatívne vysoko na systémovej úrovni**. Je hrubá a teda špecifikuje iba **hrubé správanie** CPU v inštrukčnom cykle, bez detailov protokolov pri čítaní a zápise dát z, resp. do externej pamäti RAM a pri exekúcií jednotlivých inštrukcií.

```
System CPU {
 typedef int:32 DATA;    //typ pre datovu zbernicu
 typedef unsigned:30 ADDRESS; //typ pre adresovu zbernicu
 typedef unsigned:1 SIGNAL;   //typ pre signaly
 typedef enum:2 { ADD, BRN, LD, ST } OPCODE; //druhy instrukcii

 typedef struct {
   OPCODE  Opcode;
   ADDRESS Addr;
 } INSTRUCTION;  //typ reprezentujuci instrukciu

 typedef union {
   DATA  asData;
   INSTRUCTION asInst;
 } COMPLEXDATA; //typ reprezentujuci spolocnu datovuzbernicu
                //pre instrukcie aj data
 Inputs {
   COMPLEXDATA D;
   SIGNAL Reset;
 }

 Outputs {
   COMPLEXDATA D;
   ADDRESS A;
 }
 State Vars {
   ADDRESS PC;
   DATA AC;
 }

 //agent na vyber a exekuciu instrukcie
 Agent GlobOper {
   LV {
     INSTRUCTION d1;
     DATA d2;
   }
```

```
   CS {
     set_inst_addr.read_inst.
     if (d1.Opcode == d1.Opcode.ADD) set_addr.get_data.final
     else
     if (d1.Opcode == d1.Opcode.LD) set_addr.get_data.final
     else
     if (d1.Opcode == d1.Opcode.ST) set_addr.set_data.final
     else
     if (d1.Opcode == d1.Opcode.BRN) final
   }
   action set_inst_addr {
     te: up(A, "PC");
     ov: A = "PC11";
   }
   action read_inst {
     iv: d1 = D;
     te: up(D);
   }
   action set_addr {
     te: up(A, "set_addr");
     ov: A = d1.Addr;
   }
   action get_data {
     iv: d2 = D.asData;
     te: up(D);
   }
   action set_data {
     te: up(D, "set_data");
     ov: D.asData = AC;
   }
   action final {
    te: ef;
   }

   FS {
     PC = if (d1.Opcode == d1.Opcode.BRN and AC < 0)
            d1.Addr
          else  PC + 1;
     AC = if (d1.Opcode == d1.Opcode.ADD)
            AC + d2
          else
           if (d1.Opcode == d1.Opcode.LD) d2
           else AC;
   }
 }
//agent vykonavajuci reset CPU
 Agent ResOper {
   CS { start.final }
   action start {
     te: up(Reset,1);
   }
```

```
    action final {
      te: up(Reset,0);
    }
      FS {
      PC = 0;
    }
  }

  //process reprezentujuci instrukcny cyklus CPU
  Process PrInstrCycle {
    ResOper;loop GlobOper  // nekonecny proces
  }
  Starting Structure {
    Stf PrInstrCycle: up(Reset,1) or eon;
  }
}
```

**PRÍKLAD 8**: Ukážka špecifikácie **CPU** v **HSSL** po **zjemnení špecifikácie** v jej vývoji smerom k úrovni **RT** (pozri Kap. 3). Je tu špecifikovaný **komunikačný protokol** pri čítaní dát (inštrukcie) a zápis dát z, resp. do externej RAM pamäti.

```
System CPU {

  typedef int:32 DATA;    //typ pre datovu zbernicu
  typedef unsigned:30 ADDRESS; //typ pre adresovu zbernicu
  typedef unsigned:1  SIGNAL;  //typ pre signaly
  typedef enum:2 { ADD, BRN, LD, ST } OPCODE; //druhy instrukcii
  typedef enum:1 { RES, OD } R_STATE; //stavy spracovania
                                      // instrukcie

  typedef struct {
    OPCODE  Opcode;
    ADDRESS Addr;
  } INSTRUCTION; //typ reprezentujuci instrukciu

  typedef union {
    DATA  asData;
    INSTRUCTION asInst;
  } COMPLEXDATA; //typ reprezentujuci spolocnu datovy vstup pre
                 //  instrukcie aj data

  typedef clock(1,1) CLOCK; //typ reprez. synchronizacny
hodinovy signal

  Inputs {
    COMPLEXDATA D;
    SIGNAL Wait;
    SIGNAL Reset;
    CLOCK  CLK;
  }
```

```
Outputs {
  COMPLEXDATA D;
  ADDRESS A;
  SIGNAL Req;
}

State Vars {
  ADDRESS PC;
  DATA AC;
  INSTRUCTION IR;
  R_STATE Contr;
}

//agent na vyber instrukcie
Agent Fetch {
  LV {
    INSTRUCTION d1;
  }
  IS { Contr == Contr.RES }
  DefaultTE: up(CLK, 1);
  CS {
     wait0**.wait1.read_data++.after_read.wait0**.wait1.final }

  action wait0 {
    iv: Wait = 0;
  }
  action wait1 {
    iv: Wait = 1;
  }
  action read_data {
    iv: Wait = 1;
    ov: Req = 1, A = PC;
  }
  action after_read {
    iv: Wait = 0, d1 = D;
    ov: Req = 0;
  }
  action final {
  }

  FS {
    IR = d1;
    PC = PC + 1;
    Contr = Contr.OD;
  }
}
```

```
//agent na vykonanie instrukcie BRN
  Agent BRNoper {
    IS { Contr == Contr.OD }
    DefaultTE: up(CLK, 1);
    CS { start.final }

    action start {
    }
    action final {
    }


    FS {
      PC = if (AC < 0)
             IR.Addr
           else
             PC;
      Contr = Contr.RES;
    }
  }

  //agent na vykonanie instrukcie ADD
  Agent ADDoper {
    LV {
      DATA d2;
    }
    IS { Contr == Contr.OD }
    DefaultTE: up(CLK, 1);
    CS { read_data++.after_read.final }

    action read_data {
      iv: Wait = 1;
      ov: Req = 1, A = IR.Addr;
    }
    action after_read {
      iv: Wait = 0, d2 = D.asData;
    }
    action final {
      ov: Req = 0;
    }

    FS {
      AC = AC + d2;
      Contr = Contr.RES;
    }
  }
```

```
//agent na vykonanie instrukcie LD
Agent LDoper {
  LV {
    DATA d2;
  }
  IS { Contr == Contr.OD }
  DefaultTE: up(CLK, 1);
  CS { read_data++.after_read.final }
  action read_data {
    iv: Wait = 1;
    ov: Req = 1, A = IR.Addr;
  }
  action after_read {
    iv: Wait = 0, d2 = D.asData;
  }
  action final {
    ov: Req = 0;
  }

  FS {
    AC = d2;
    Contr = Contr.RES;
  }
}

//agent na vykonanie instrukcie ST
Agent SToper {
  IS { Contr == Contr.OD }
  DefaultTE: up(CLK, 1);
  CS { write_data++.after_write.final }

  action write_data {
    iv: Wait = 1;
    ov: D.asData = AC, Req = 1;
  }
  action after_write {
    iv: Wait = 0;
  }
  action final {
    ov: Req = 0;
  }

  FS {
    Contr = Contr.RES;
  }
}
```

```
  //agent vykonavajuci reset CPU
  Agent ResOper {
    DefaultTE: up(CLK, 1);
    CS { start.final }

    action start {
      iv: Reset = 1;
    }
    action final {
      iv: Reset = 0;
    }

    FS {
      PC = 0;
      Contr = Contr.RES;
    }
  }

  //process reprezentujuci instrukcny cyklus CPU
  Process PrInstrCycle {
    ResOper;
    loop {
      Fetch;
      if (IR.Opcode == IR.Opcode.BRN) BRNoper
      else
      if (IR.Opcode == IR.Opcode.ADD) ADDoper
      else
      if (IR.Opcode == IR.Opcode.LD) LDoper
      else
      if (IR.Opcode == IR.Opcode.ST) SToper
    }
  }

  Starting Structure {
    Stf PrInstrCycle: up(Reset,1) or eon;
  }
}
```

**PRÍKLAD 9**: Opis **CPU** v jazyku **VHDL** na úrovni **RT**

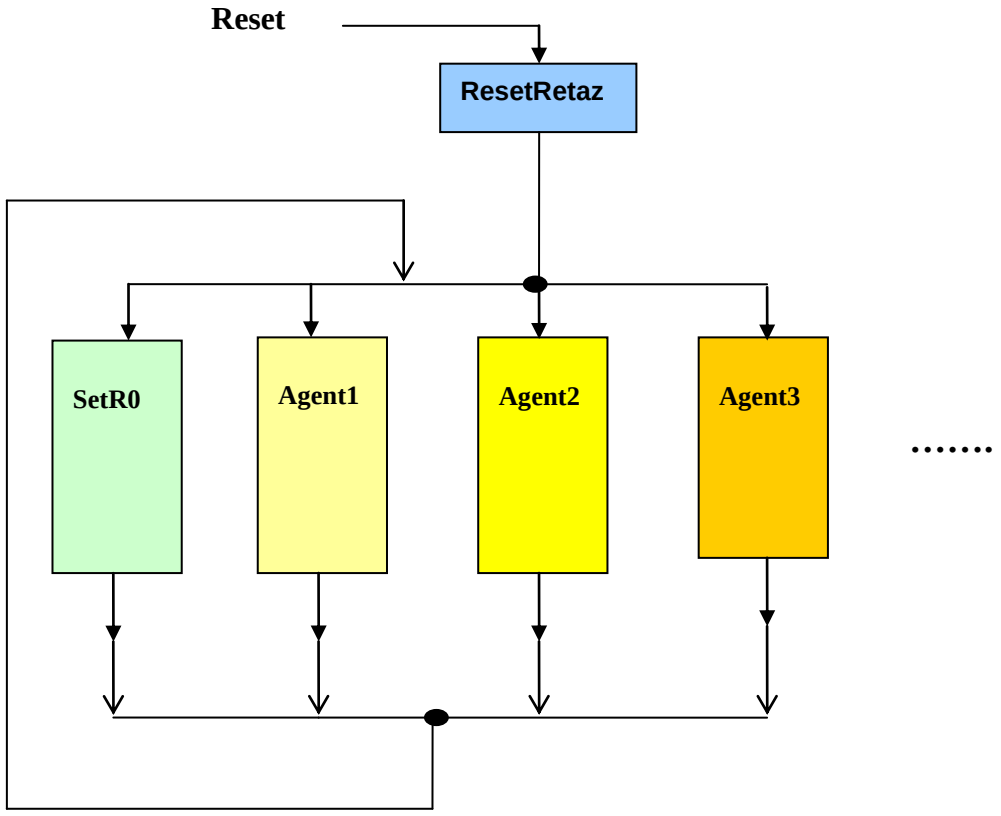
```
-- Opis spravania jednoducheho procesora (pomocou FSM)
-- datove typy
-- balik types
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use types.all;
```

```
-- entita
entity CPU is
  port
  (
    p_D     : inout udaj;
    p_Wait  : in    std_logic;
    p_Res   : in    std_logic;
    p_CLK   : in    std_logic;
    p_A     : out   adress;
    p_Req   : out   std_logic;
    p_Rd_Wr : out   std_logic
  );
end CPU;
-- architektura, opis spravania CPU
architecture behavior of CPU is
 signal  stav : stavy := RES;
 begin
  process(p_CLK, p_Res)
  begin
    if(p_Res = '1') then
      stav <= RES;
    elsif  (p_CLK'Event and p_CLK= '1')   then
      case Stav is
        when RES => stav <= IF0;
        when IF0 => stav <= IF1;
        when IF1 => if p_Wait = '0' then
                      stav <= IF1;
                    elsif p_Wait = '1' then
                      stav <= IF2;
                    end if;
        when IF2 => if p_Wait = '1' then
                      stav  <= IF2;
                    elsif p_Wait = '0' then
                      stav <= IF3;
               end if;
        when IF3 => if p_Wait = '0' then
                      stav <= IF3;
                    elsif p_Wait = '1' then
                      stav <=  OD;
                    end if;
        when  OD => case IR.opcode is
                      when  LD => stav <= LD0;
                      when  ST => stav <= ST0;
                      when ADD => stav <= AD0;
                      when BRN => stav <= BR0;
                    end case;
        when LD0 => stav <= LD1;
        when LD1 => if p_Wait = '1' then
                       stav <= LD1;
                     elsif p_Wait = '0' then
                       stav <= LD2; end if;
```

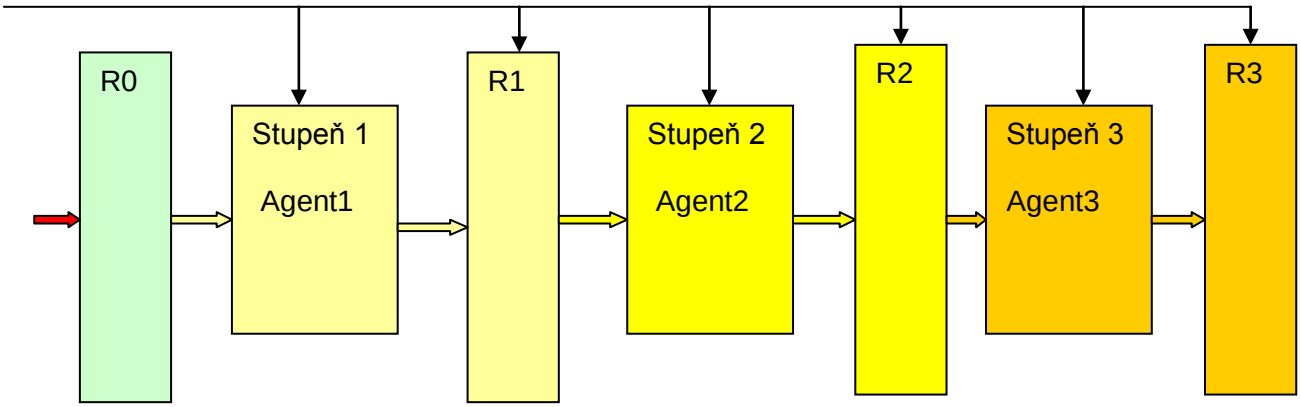
```
        when LD2 => stav <= IF0;
        when ST0 => stav <= ST1;

        when ST1 => if p_Wait = '1' then
                      stav <= ST1;
                    elsif p_Wait = '0' then
                      stav <= IF0;
                    end if;
        when AD0 => stav <= AD1;
        when AD1 => if p_Wait = '1' then
                      stav <= AD1;
                     elsif p_Wait = '0'  then
                      stav <= AD2;
                     end if;
        when AD2 => stav <= IF0;
        when BR0 => if AC >= 0 then
                      stav <= IF0;
                    else
                      stav <= BR1;
                    end if;
        when BR1 => stav <= IF0;
      end case;
    end if;
  end process;
out_proc:
process(stav)
    variable PC   : adress;
    variable MAR  : adress;
    variable MBR  : udaj;
    variable AC   : udaj;
    variable IR   : instrukcia;
    variable Output:std_logic_vector(1 downto 0):="ZZ";
    -- variable Data  :udaj:="ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
  begin
    case stav is
     when RES => PC   := "00000000000000000000000000000000";
                 Output := "ZZ";
                 p_D<="ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
     when IF0 => Output := "Z0";
                    p_D<="ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
                 MAR  := PC;
                 PC   := PC + 1;
     when IF2 | LD1 | AD1 =>
                     p_A   <= MAR;
                  Output := "11";
                  MBR   := p_D;
     when IF3 =>  Output := "Z0";
                IR  := mbr_to_ir(MBR);
     when LD0 =>  MAR  := IR.adresa;
     when LD2 => Output := "Z0";
               AC := MBR;
```

```
      when ST0 => MAR  := IR.adresa;
                  MBR  := AC;
        when ST1 => p_A <= MAR;
                    p_D <= MBR;
                    Output := "01";
                 -- Data   :=MBR;
      when AD0 => MAR  := IR.adresa;
      when AD2 => Output := "Z0";
                  AC := AC + MBR;
         when BR1 => PC := IR.adresa;
         when others => null;
       end case;
      p_Req   <= Output(0);
      p_Rd_Wr <= Output(1);
  end process;
end CPU_behavior;
```

**PRÍKLAD 8:** Koncepcia formálneho opisu **prúdového spracovania** v našom modeli (HSSL). Uvedieme koncepciu opisu prúdového spracovania v zreťazenom systéme (pozri nasledujúci obr. )




Na obrázku je v grafe znázornená spriahnutie agentov, ktoré sú priradené jednotlivým stupňom prúdového reťazca a jednotlivo, behavioristicky opisujú ich správanie.
Prúdový reťazec je načrtnutý na nasledujúcom obr. :




R0,….,R3, … sú registre prúdového reťazca Stupeň „j“ j podsystém v j-tom stupni, ktorého správanie je opísané agentom Aj (j= 1, 2, 3, ….). Vnútorné vstupy a výstupy subsystém „j“ v reťazci označíme Dij, resp. Doj. Vstup registra R0 je DI0 z okolia.

Správanie systému ako celku je behavioristicky opísané v zavedenom modeli (HSSL) koncepčne takto:

proces PrudovaOper = ResetRetaz. [SetR0 || Agent1 || Agent2 || Agent3 ||…] $^{\omega}$

agent Agent**j** //genericky agent pre jednotlive stupne **j** = 1, 2, 3, ….
SI <Rj-1> =qj-1
TE es, ef
KM (DIj = vj ; es ; u ).( u ; ef ; Doj = hj )
vs hj = v(qj-1, vj) // v je vystupna funkcia FSM stupna j
g Rj := hj

agent SetR0
TE es, ef
KM (DI0 =v0; es ; u ).( u ; ef ; u)
g FA j <R0> := v0

agent ResetRetaz
TE es, ef
KM (u ; es ; u ).( u ; ef ; u)
g FA j <Rj> := (0, 0, …., 0) // j = 1, 2, 3, …

Pri synchrónnom systéme, napr. s hodinami up(Clk,1) časovacie udalosti dané, alebo odvodené z hodín.

**Konkrétny príklad**: Uvedieme modifikovanú **CPU** s prúdovým spracovaním. Pri modifikácií budú nasledujúce zmeny

- komunikácia s externou pamäťou, teda čítane inštrukcie a dát z externej pamäti sa uskutoční iba v **jednom** hodinovom cykle. Ide o realistický prípad, ak CPU číta, resp zapisuje do vyrovnávacej pamäti(í) typu **cache**.
- Dodáme inštrukciu **ADI**, pri vykonávaní ktorej sčíta obsah AC s konštantou (K), ktorá je v inštrukcií ADI s formátom (opkod, konstanta)

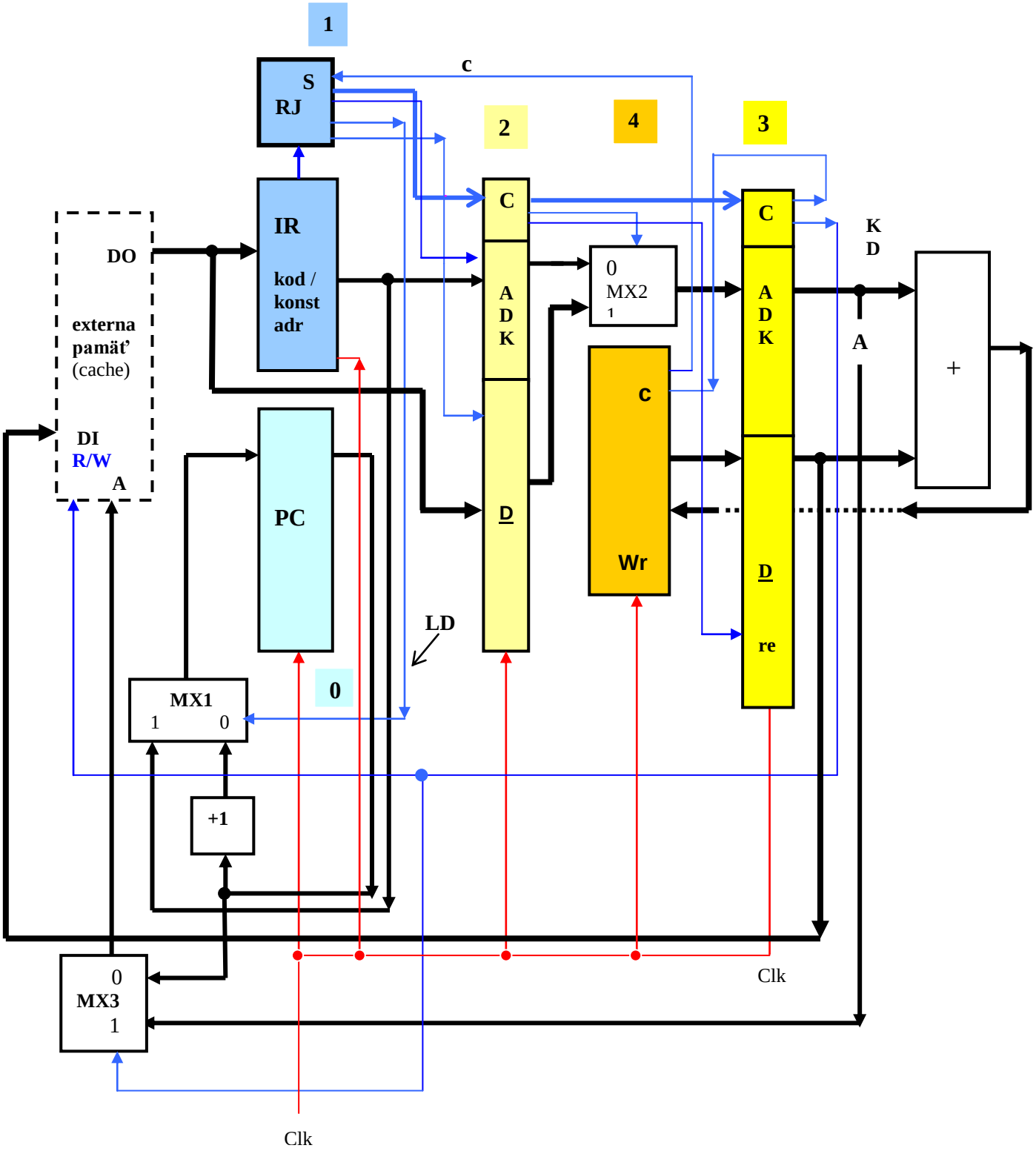
Pri prúdovom spracovaní rozdelíme inštrukčný cyklus na fázy 0, 1, 2, 3, 4. Bude 5 agentov: Agent0, Agent1, Agent2, Agent3, Agent4, ktoré sú paralelne spriahnuté a jeden agent ReRetaz ktorý je sekvenčne spriahnutý s so skupinou súbežných agentov. V systéme CPU budú v tomto (školskom) príklade v reťazci prúdového spracovania inštrukcií, vyberaných z externej pamäti, 5 prúdových registrov, ktoré označíme R0, R1, R2 , R3 a R4. Prúdové registre majú segmenty podľa obrázku dole. Na tomto obrázku je vidno použitú štruktúrnu koncepciu. R0 a podobne R1 a R4 majú iba jednu sekciu, ktorou je register PC, resp. IR a AC. Špecifikácia v našom modeli je takáto (pre zjednodušenie **vynecháme** časti DT, PORTY a OPER STAVY zápisu špecifikácie).

system CPU – so zretazenim

```
proces   PrudovyProces =   ResetRetaz. [Agent0 || Agent1  || Agent2 || Agent3  ||
                                                  || Agentč4 ] ω
 agent   Agent0   // nastavuje register PC na adresu nasled.instrukcie v pamati
    TE     es, ef
    KM      (u; es ; u ).( u ; ef ; PC=d0 );
    vs        d0 = <R0.PC> ;
    g          R0.PC :=   FA <R1.(IR.opkod)>      R0.PC+1
                          ak  <R3.[IR.opkod] >  ε  {LD,ST, BRN}  tak     R3.ADK;

 agent   Agent1  // nastavuje prudovy register R1, ktorz obsahuje iba register RI
    TE     es, ef ;
    KM      (<PC> ; es ; u ).( u ; ef ; RI = d);
    vs         d = <M[PC]> ;
    g          R0.IR := MO = M[PC] ;    // do RI sa zapise vystup externej pamati M

 agent   Agent2   // nastavuje obsah segmentov prudoveho registra R2
    SI   <R1.IR>, <R0.PC> ;
    TE     es, ef ;
    KM      (u; es ; u ).( u ; ef ;ADK = d1, D = d2 >;
    vs        d1 =  <R(IR.ADK)>,  d2 = <M[PC]>  ;
    g          R2.C := R1.(IR.opkod);
               R2.ADK : = R1.(IR.ADK)
                          // (ADK oznacuje segment prudovych registrov R1,R2 a
                          // R3,  v ktorych sa uchovava adresa alebo udaj alebo
                          // konstanta  z instrukcie zapisanej z pamati do R1.IR
               R2.D :=  DO = <M[R0.PC]
                          // D segment prudovych registrov pre udaj D (aj adresu) s
                          // plnou dlzkou udaja CPU

 agent   Agent3   // nastavuje segmenty prudoveho registra R3
    SI   <R2>
    TE     es, ef
    KM      (u; es ; u ).( u ; ef ;R3.ADK= d1, R3.D =d2) ;
    vs         d1 = ak <R2.C>  ε  {ST, LD, ADI, BRN }    tak <R2.ADK>   inak
                      ak  <R2.C> =  ADD    tak <R2.D>
               d1 =  ak  <R2.C> ε  {ADD, ADI }   tak  <AC>  inak
                      ak   <R2.C> ε {LD, BRN}  tak   0
    g          R3.C      :=  R2.C ,
```

```
                R3.ADK :=  ak <R2.C>  ε  {ST, LD, ADI, BRN }    tak R2.ADK   inak
                           ak  <R2.C> =  ADD    tak R2.D
                R3.D     :=  ak  <R2.C> ε  {ADD, ADI }   tak  AC  inak
                           ak   <R2.C> ε {LD, BRN}  tak   0
    agent   Agent4   // nastavuje prudovy register R4 obsahujuci iba register AC
                     // a nastavuje vstupy DI a A (vstup dat, adresa) externej pamati M
        SI   <R3 >;
        TE     es, ef;
        KM      (u; es ; u ).( u ; ef ; AC =d)
        vs          d = ak <R3.C>  ε  {ADD, ADI }   tak  <R2. ADK + R2. D >  inak
                                ak <R3.C>  = LD  tak  <R3.D > ;
        g          AC :=  ak <R3.C>  ε  {ADD, ADI }   tak  R2. ADK + R2. D   inak
                          ak <R3.C>  = LD  tak  R3.D ;
                   M.IN := ak < R3.C> ST  tak  R3.D ;
                   M.A  := ak < R3.C> ST  tak  R3. ADK ;


agent   ResetPipe
        TE      es, ef ;
        KM       (u ; es ; u ).( u ; ef ; u);
        g              FA  j=1,2,3,4    <Rj> :=  (0, 0, …., 0);          // j = 1, 2, 3, …
                                        <R0> := sA;  // sA je zaciatocna adresa, konstanta

START       PrudovyProces   (eon or up(RestRetaz,1))
```

Jedna y koncepcií štruktúry operačnej časti reťayca a riadenia prúdového spracovania je uvedená na obrázku dole:


1
S
RJ
c
2
4
3
C
IR
kod /
konst
adr
A
D
K
0
MX2
1
C
A
D
K
K
D
DO
externa
pamäť
(cache)
DI
R/W
A
PC
D
C
Wr
D
re
+
LD
0
MX1
1
0
+1
0
MX3
1
Clk
Clk