4. Metodika syntézy operačných častí a špecifikácia riadenia

A. Prechod od špecifikácie na syntézu operačnej časti systému
B. Reprezentácia štruktúrnej implementácie operačnej časti a špecifikácie riadenia - **tok údajov** a **tok riadenia**
C. Zostavenie štruktúry operačnej časti
D. Zostavenie špecifikácie riadiacej časti

E. Prechod od špecifikácie na syntézu operačnej časti systému

Syntéza operačných častí (OČ) predstavuje náplň jednej časti procesu návrhu štruktúrnej implementácie primárnych architektúr pozostávajúcej z dvojice (**OČ**, **RČ**). **Východiskom** je (cieľová) **špecifikácia systému**, cieľom je **štruktúrna implementácia** OČ. Vstupnou informáciou návrhu takejto implementácie je množina procesov, alebo tzv. globálny proces, ktoré sa inicializujú zo štartovacieho mechanizmu špecifikácie. Pri vykonávaní P_S sa na OČ dosahuje špecifikované spracovanie informácie. V našich príkladoch možno globálny proces extrahovať a zostaviť zo špecifikácie.

Globálny proces P_S vyjadruje **celkové správanie** systému dané cieľovou špecifikáciou a to opisnými prostriedkami jedného procesu. Obsahuje agenty (obvykle mikrooperácie) definované na systéme (ako celku, čiernej skrinke, bez vyjadrenia informácie o obvodovej štruktúre).

P_S zostavíme na základe znalosti o štartovaní procesov (aj agentov) vonkajšími udalosťami. Globálny proces sa naštartuje udalosťou "ez" (reprezentujúcou pripojenie energie) a ako celok sa realizuje v **nekonečnom cykle** s "čakacím cyklom" na niektorú štartovaciu udalosť (danú štartovacím výrazom) tohto procesu. V príkladoch použijeme práve **P_S**

Prevod opisu správania v danej špecifikácií na proces **P_S** predstavuje jednoduchú transformáciu. Niekedy je tento proces už priamo opísaný v špecifikácií. Ukážeme to najskôr na príklade systému VSYS a neskoršie aj na príklade procesora CPU.

Vychádzame z vypracovanej cieľovej špecifikácie VSYS-2

VSYS-2 (tu vynecháme položku DT)

```
    PORTY
    vstup              D       udaj;
                  RD, CO, ACK, CLK      booleovska hodnota;
    vystup             DO      udaj;
                  RR, CR, REQ           booleovska hodnota;
    OPER STAV:    M                     pole M(0),...,M(15) udaj;
                  DO      udaj;
                  RR, CR, REQ           booleovska hodnota;
                  I       index;
  proces    PrReadIn = Reset.[(ACK=1)[EA].SetReq.[EA](ACK=1).ResReq.WriteM.
                        Inkrl ](I=0).SetRR ;
  proces    PrCompute = Reset.[Add.Inkrl](I=0).SetCR ;
```

```
agent    Reset
  TE  es, ef =up(CLK=1);
   KM   (u; es; u)(u; ef; DO=0, RR=0, CR=0, REQ=0);
          g        I:=0; DO:=0; RR:=0; CR:=0; REQ:=0
          TR      FA     e ∈ { up(I=0), up(DO=0), dw(RR=1), dw(CR=1), dw(REQ=1)}
           afto(E, ef, 0) ;
agent    InkrI
  TE   es, ef =up(CLK=1);
  KM    (u; es; u)(u; ef; u);
  g        I:=I+1;
  TR            afto(up(I=I+1), ef, 0);
agent   Add
  ES   es, ef =up(CLK=1);
  KM     (u; es; u)(u; ef; DO=d);
  vs        d = <DO> +< M(I)>;
  g         DO:=DO + M(I);
  TR            afto(up(DO=d), ef, 0);
 agent   SetCR
TE   es, ef =up(CLK=1);
 KM    (u; es; u)(u; ef; CR=1);
 g        CR:=1;
TR           afto(up(CR=1), ef, 0);

 agent   WriteM
TE   es, ef =up(CLK=1);
 KM    (D=d; es; u)(u; ef; u)
 g        M(I) := d
 TR       afto(up(M(I)=d), ef, 0)

agent   SetReq
 TE   es, es, ef =up(CLK=1);
 KM    (ACK=0; es; u)(u; ef; REQ=1);
 g        REQ:=1;
 TR      afto(up(REQ=1), ef, 0);

agent   ResReq
TE   es, es, ef =up(CLK=1);
 KM   (ACK=1; es; u)(u; ef; REQ=0);
 g        REQ :=0;
 TR      afto(dw(REQ=1),ef,0);

agent   SetRR
TE   es, es, ef =up(CLK=1);
 KM   (u; es; u)(u; ef; RR=1);
 g        RR:=1;
TR       afto(up(RR=1), ef, 0);

START          PrReadIn       RD and up(CLK=1) ;
               PrCompute      CO and up(CLK=1) ;
 restr     RD nand CO; RR => (not CO); CR => (not RD) ;
```

Alebo **alternatívne** pridáme ešte proces P1

P1 = [(RD=1)**:** PrReadIn, (CO=1)**:** PrCompute, ((RD=0 and CO=0))**:** EA]$^{\omega}$;

pričom START upravíme takto:

START P1 ez;
restr RD nand CO; RR => (not CO); CR => (not RD) ;

Globálny proces sa nachádza práve z uvedenej druhej alternatíve vo VSYS-2. Totiž, $\mathbf{P}_S$ zodpovedá práve tu uvedenému procesu **P1**.

P_S=P1= [(RD=1)**:** Reset**.** [$^{(ACK=0)}$[EA]**.**SetReq**.**[EA]$^{(ACK=1)}$**.** ResReq**.** WriteM**.**
. **.**InkrI]$^{(I=0)}$**.** SetRR,
(CO=1)**:** Reset**.**[Add**.** InkrI]$^{(I=0)}$**.**SetCR,
((RD=0)and(CO=0)): EA]$^{\omega}$;

Vo forme **programovej schémy** tento proces môžeme opísať aj s danými obmedzeniami v šecifikácií (restr) zapíať aj takto:

P_S =
L0: EA / (RD and (not CO) and (not RR))**:** L1**,** (CO and (not RD) and (not CR))**:** R1**,**
(RD nor CO) and (RD or (notCR)) and (CO or (notRR)) **:** L0
L1: Reset / L2;
L2: ACK**:** L3**,** L4
L3: EA / L2
L4: SetReq / L5;
L5: EA / ACK: L6**,** L5;
L6 :ResReq / L7
L7: WriteM / L8;
L8:InkrI/L9;
L9:(I /=0) and (ACK=0)**:** L4**,** (I /=0) and (ACK=1)**:** L3**,** (I=0)**:** L10;
L10: SetRR / L0
R1: Reset / R2;
R2: Add / R3;
R3: InkrI / R4;
R4:(I /= 0)**:**R2**,** R5;
R5: SetCR / L0

Je zrejmé, že ide práve o taký proces, pri exekúcii ktorého sa systém správa tak, ako to predpisuje špecifikácia VSYS-2. Prvé návestie L0 označuje **začiatočný** príkaz, ktorý predpisuje **čakať** (pri RD=0 a CO=0) v **cykle** z prázdnym agentom (prázdnou mikro-operaciou) **EA**, kým sa neobjaví udalosť up(RD=1) alebo up(CO=1). Po tejto udalosti sa inicializuje exekúcia príslušnej **sekcie procesu** (označená návestiami "**L**" alebo "**R**"), ktorá zodpovedá procesu **PrReadIn**, resp. **PrCompute** (a teda exekúcia pôvodných agentov **ReadIn** a **Compute**). Pri danej restr sú výrazy RR and (notRD) ; CR and (notCO) ; ktoré zabezpečujú, že ak skončil agent PrReadIn, tak nebude znova opakovane naštartovaný a podobne to platí aj pre PrCompute.

1. Reprezentácia štruktúrnej implementácie operačnej časti a špecifikácie riadenia - tok údajov a tok riadenia

Prostriedkami reprezentácie štruktúry OČ a globálneho procesu Ps, s ktorými sa pri návrhu v praxi najčastejšie pracuje, sú dva orientované grafy, vyjadrujúce priamo tzv. **tok údajov** a **tok riadenia** v navrhovanom systéme.

Tok údajov chápeme ako súbor funkčných závislostí medzi premennými, nositeľmi údajov (t.j. medzi **operandami** - vstupnými údajmi alebo údajovými konštantami a **výsledkami** - výstupnými údajmi) a teda súbor väzieb medzi premennými, údajovými konštantami a operáciami. **Operácie** obsiahnuté v toku údajov sú obyčajne alebo

-- **jednoduché funkcie** (vyjadrené operátormi +,-,*, / a pod.)
nad príslušnými údajmi, alebo

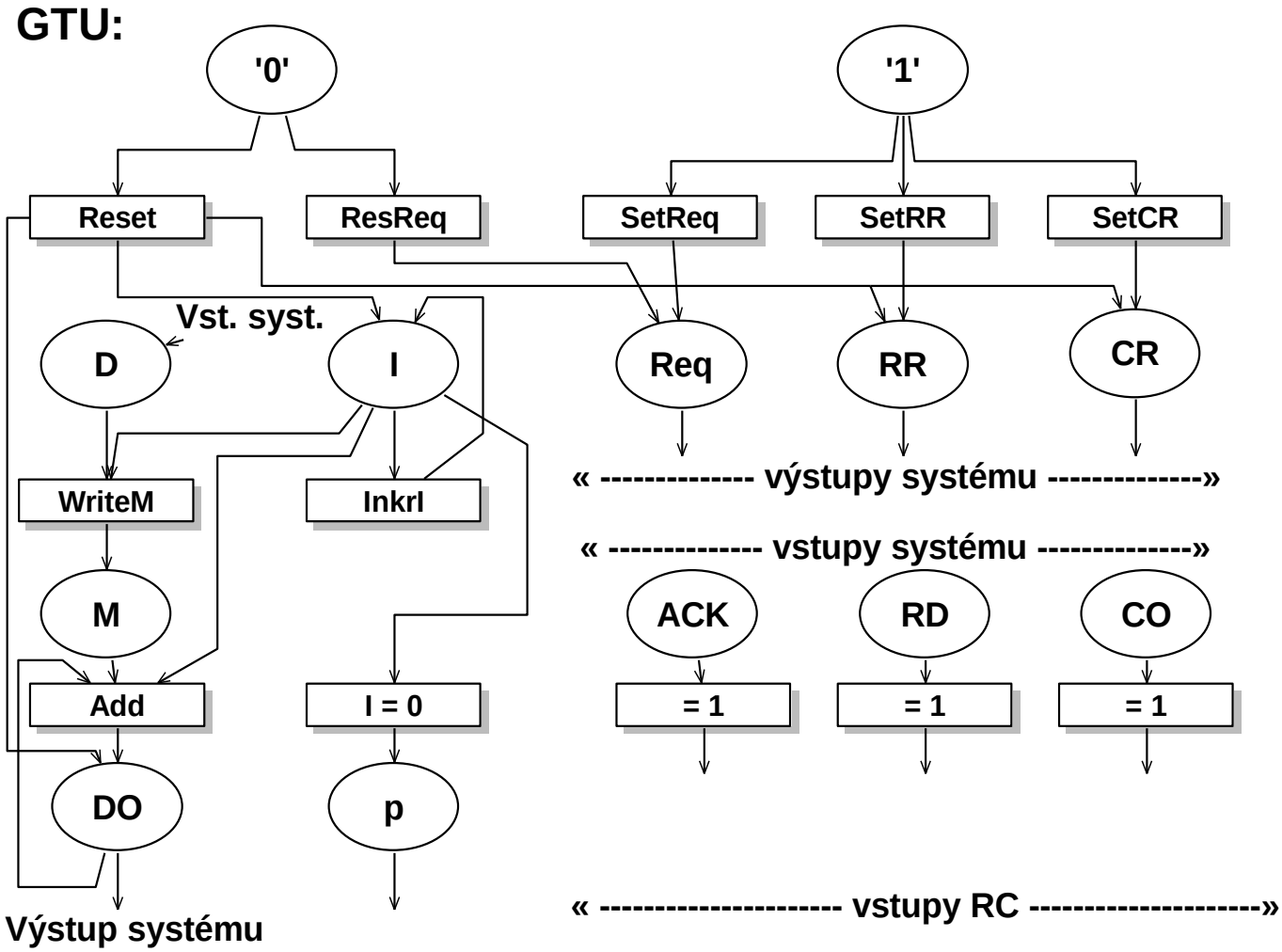
-- priamo **agenty** (najčastejšie, mikrooperácie).

Výstupmi (výstupnými premennými) daného agenta **A** v **toku údajov** sú práve tie **výstupné** a **operačné stavové** premenné systému, ktorých hodnoty sa v **A nastavujú**. **Vstupy** (vstupné premenné) agenta **A** v **toku údajov** sú pritom tie **vstupné** a **stavové** premenné systému a údajové konštanty, od ktorých **priamo závisia** hodnoty nastavovaných premenných v **A**. Tak napr. agent **Add** z VSYS-2 má tieto vstupy a výstupy: stavové premenné M, I a DO, resp. stavovú premennú DO. Značíme to takto: Add (M, I, DO **;** DO)

Graf toku údajov (**GTU**) sa zostaví tak, že premenným, konštantám a operáciam v danej špecifikácií sa priradia **vrcholy** orientovaného grafu. Orientované **hrany** sa usporiadajú tak, aby vyjadrovali realizované **údajové väzby** medzi vstupmi, operáciami (agentmi) a ich výstupmi.

Na nižšie uvedených obrázkoch sú **obidve** spomenuté formy **GTU** systému **VSYS** (prvá s agentmi, druhá, s jednoduchými funkciami;). Vrcholy priradené vstupným premenným operácií (a konštantám) a výstupným premenným operácií nerozlišujeme navzájom. Vrcholy priradené operáciam vyznačujeme jednoduchými **obdĺžnikmi** a vrcholy priradené premenným sú kružnice alebo elipsy. Konštanty označujeme ako určité hodnoty medzi dvoma apostrofmi „**’ ‘**, napr. konštanta: **‘ 0‘**.

V nasledujúcom obr. Je **GTU**, v ktorom sú operácie **priamo agenty** definované v špecifikácií VSYS-2. Tento spôsob vyjadrenia toku údajov nie je viazaný na úroveň abstrakcie; môže používať ľubovoľné agenty ako nedeliteľné entity.



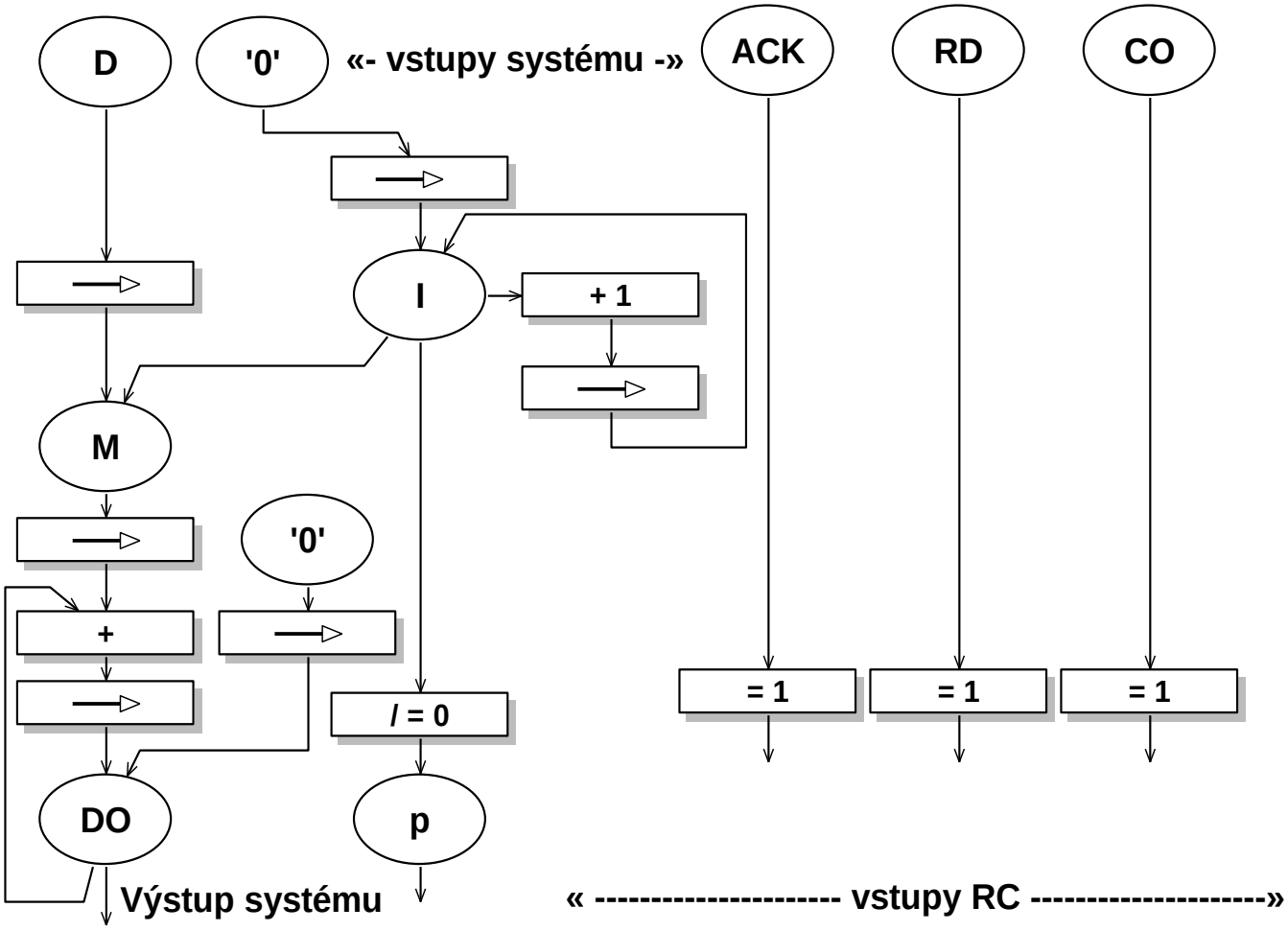
V grafe sú naznačené aj operačné vrcholy (funkcie) generujúce hodnoty predikátov. (napr. (I = 0): ide o binárnu funkciu, ktorá nastaví výstupnú premennú p na 0 alebo 1, nositeľa hodnoty predikátu, na hodnotu 1 práve vtedy, ak <I>=0). Nositeľmi predikátov (ACK=1), (RD=1) a (CO=1), ktoré majú hodnotu 0 alebo 1, sú priamo vstupné boolovské premenné systému (nie sú tu teda uvedené žiadne výstupné premenné pre tieto predikáty). V ďalšom využijeme **GTU** iba s **jednoduchými agentmi** (mikrooperaciami), v ktorých môže byť zahrnutý určitý paralelizmus v realizácii funkcii v nich obsiahnutých (napr. agent **Reset** zahrňuje 4 funkcie priradenia hodnoty 0 operačným stavovým premenným DO, I, RR, CR, čo sa vykonáva súbežne).

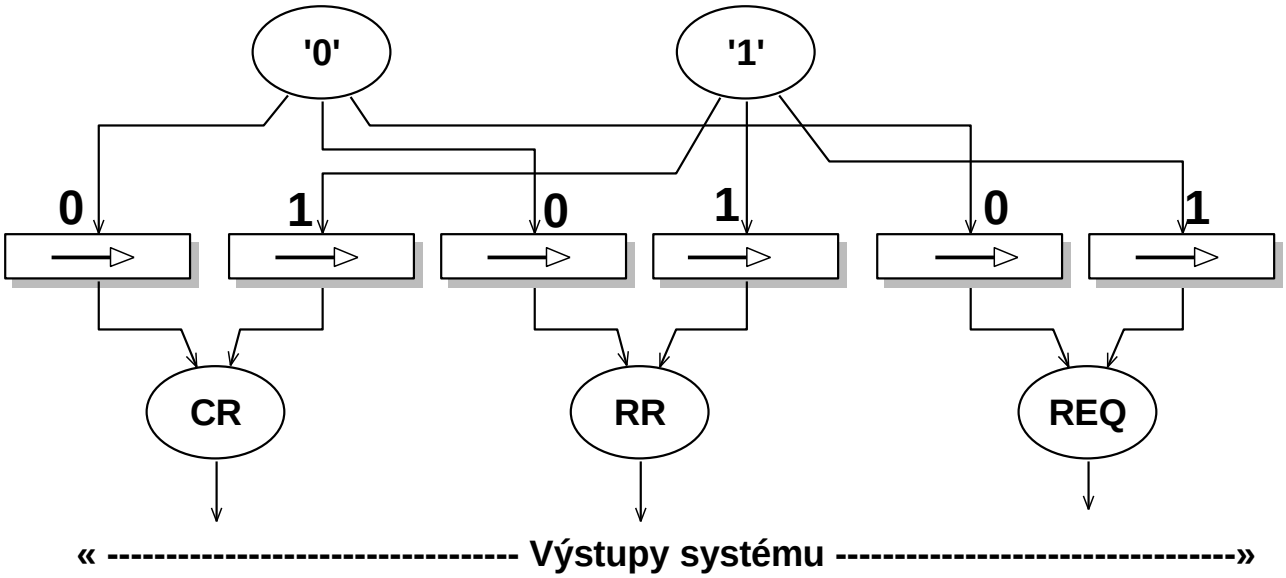
Spôsob zostavenia je zrejmý. Uvedený graf možno ľahko zostaviť ak vychádzame zo špecifikácie VSYS-2 a z jednotlivých agentov a stanovíme ich vyššie definované **vstupy** a **výstupy** v GT. Napr. agent **Add** z VSYS-2 má tieto vstupy a výstup - stavové premenné M,I a DO, resp. stavovú premennú DO. Medzi tieto vrcholy vložíme vstupujúce orientované hrany, resp vystupujúcu hranu do, resp z vrcholu **Add**.

Takto sme zostavili vstupujúce a vystupujúce hrany pre všetky vrcholy GTU priradené agentom.

Nasledujúci **GTU** reprezentuje tok údajov s **jednoduchými operáciami** vyjadrenými priamo známymi symbolmi operácií (operátormi) **+** (súčet údajov), **+1** (inkremento- vanie údaja), → (priame priradenie určitého údaja premennej).

GTU:



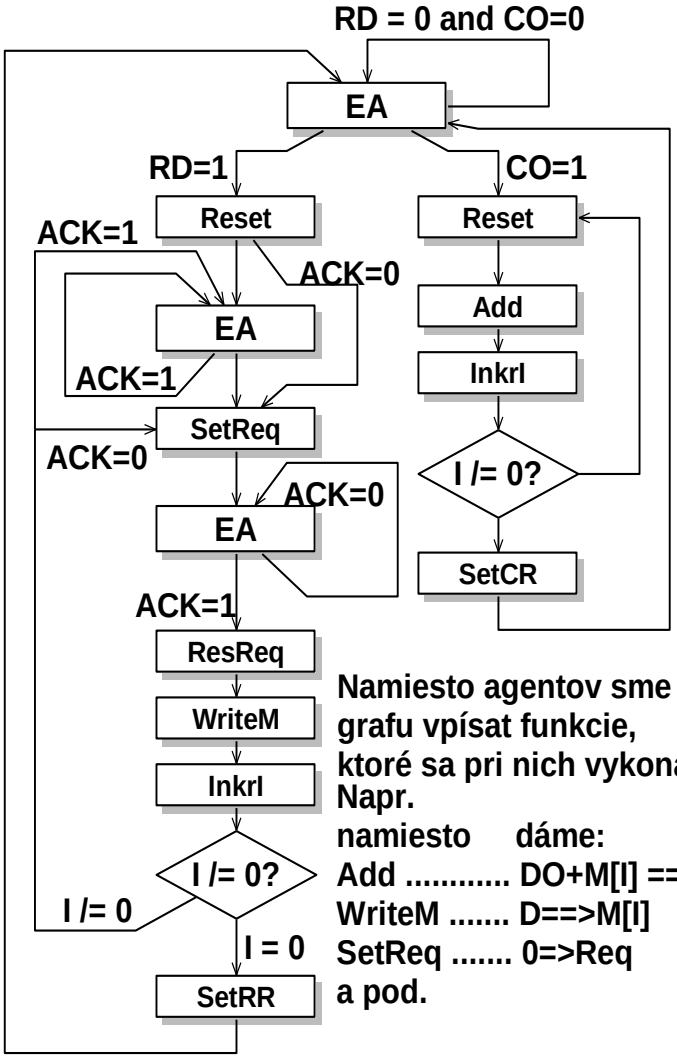


Graf zostavíme z predchádzajúceho GTU s tým, že analyzujeme jednotlivé agenty (tu mikrooperácie) a zistíme aké **jednoduché operácie** sa v nich **realizujú** nad vstupnými

operandami privedenými k vrcholu priradenému jednoduchej operácií (funkcií) a výsledky sa objavujú na výstupoch týchto vrcholov. Napr. v mikrooperácií Add sa realizuje jednoduchá operácia "**+**" nad údajmi (hodnotami) premenných M,I, a DO s nastavením výsledku ako (novej) hodnoty DO {t.j. DO := DO + M[I] }. Symbolom „→"sa vyznačujú **priraďovacie operácie** „:= " pre operačné stavové premenné. Často ich **vynechávame**.

Naostatok uvedený **GTU** s jednoduchými operáciami (funkciami) je vhodný pre **extrahovanie štruktúry** operačnej časti systému. **GTU** s agetmi použijeme (v zjednodušenej forme pri plánovaní vykonávania operácií v riadiacich cykloch **RČ**.

Tok riadenia a jeho grafická forma „graf toku riadenia" (**GTR**) vyjadruje **spriahnutie** operácií (agentov, jednoduchých funkcií) do **globálneho procesu** (ov) prebiehajúceho (prebiehajúcich) v danom systéme. **Vrcholom** v **GTR** priraďujeme **množiny operácií** (niektoré podmnožiny množiny operácií). **Orientovane hrany** opisujú vyžadované spriahnutie operácií v čase. Prítomnosť dvoch operácií v jednom vrchole **GTR** vyjadruje ich súbežnú (paralelnú) exekúciu. **GTR** extrahovaný zo špecifikácie **VSYS-2** opisuje spriahnutie agentov, ktoré v systéme nastáva pri jeho fungovaní v "nekonečnom" cykle v závilosti od vstupných a iných udalostí. **GTR** (tu teda) predstavuje určitú **grafickú formu** zápisu globálneho procesu (mikroprogramu) $\mathbf{P_S}$. V nasledujúcom obr. je **GTR**, ktorý zodpovedá východiskovému P_S vo **VSYS**.



Namiesto agentov sme mohli do vrcholov grafu vpísat funkcie, ktoré sa pri nich vykonajú. Napr. namiesto dáme:
Add ..... DO+M[I] ==>DO
WriteM ..... D==>M[I]
SetReq ..... 0=>Req
a pod.

Pre **porovnanie** uvedieme programovú schému P_S:

L0**:** EA / RD**:** L1, CO**:** R1, (-RD and -CO)**:** L0
L1**:** Reset / L2;
L2**:** ACK**:** L3**,** L4;
L3**:** EA / L2;
L4**:** SetReq / L5;
L5**:** EA / ACK**:** L6**,** L5;
L6**:** ResReq / L7;
L7**:** WriteM / L8;
L8**:** InkrI / L9;
L9**:** (I /=0) and (ACK=0)**:** L4**,**
(I /=0) and (ACK=1)**:** L3**,** (I=0)**:** L10;
L10**:** SetRR / L0;

R1**:** Reset / R2;
R2**:** Add / R3;
R3**:** InkrI / R4;
R4**:** (I /= 0)**:** R2**,** R5;
R5**:** SetCR / L0

2. Zostavenie štruktúry operačnej časti

Z gragfu **GTU**, ktorý obsahuje jednoduché funkcie možno vychádzať pri zostavení **štruktúry OČ** systému (na úrovni registrov, funkčných jednotiek a multiplexorov (zberníc) - t.j. na úrovni **RT**). Postupujeme takto:

Operačným stavovým premenným priradíme zodpovedajúce **pamäťové prostriedky**. Ak sa v cieľovej špecifikácií nachádzajú stavové premenné s nasledujúcimi **údajovými typmi** (ku ktorým pri vývoji špecifikácie najčastejšie smerujeme):

1. konečná množina symbolov, podmnožina množiny celých čísel, boolovské vektory
2. jednorozmerné pole údajov typu
3. záznam s položkami typu,

potom stavovej premennej typu 1, 2, 3 priradíme pamäťové prvky **1**, **2**, **3** takto:

1. **Register** s vhodnou dľžkou, ktorého obsah (boolovský vektor) zodpovedá binárne zakódovaným údajom daného typu; pričom zvolíme typ registra podľa potreby.
2. **Súbor registrov** s rovnakou dĺžkou ako v 1, pričom zvolíme požadovaný spôsob **prístupu** k jednotlivým prvkom poľa, k registrom (napr., priamy prístup miesta s danou adresou pamäti typu **RAM**, alebo prístup k vrcholovému miestu ako pri pamäti typu zásobník – stack a pod.).
3. **Súbor registrov**, z ktorých každý má potrebnú (spravidla inú dĺžku, danú príslušným údajovým typom; prípadne register s vymedzenými **vnútornými poľami** potrebnej dĺžky pre jednotlivé položky.

PRÍKLAD 1: Z GTU VSYS vyplýva priradenie pamäťových prostriedkov operačným stavovým premenným, ktoré môže vyzerať takto:

Stavové premenné D.. celé číslo z $< -2^{15}, +2^{15} - 1 >$
v doplnkovom kóde

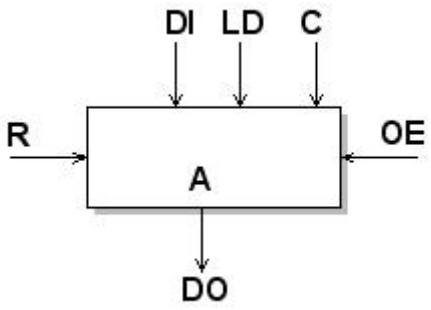
I .... nezáporné číslo z intervaliu < 0, 2^4>

M .... pole 16 celých čísel

REQ, RR, CR ... boolovská hodnota

Priradené pamäte: DO ==> register (RDO) s dĺžkou 16 bit so synchronizovaným riadeným zápisom a s nulovacím vstupom
I ==> register (RI) s dĺžkou 4 bit; typ ako pri DO
M ==> súbor registrov (MEM) s prístupom ako v pamäti RAM s adresovým vstupom 4 bit, údajovým vstupom DI a výstupom DO (16 bit).
REQ, RR, CR ==> preklápací obvod (register s dĺžkou 1) typu asynchrónny SR-PO

Špecifikácia registra s riadeným synchronizovaným zápisom údaja zo vstupu DI s nulovaním obsahu a s riadeným výstupom. Uvedieme to ako príklad špecifikácie prvku v knižnici. podobne možno špecifikovať iné prvky.



system REGISTER (T_{Dset}, T_{Dhold}, T_{AD},, T_{RA}) - - bez vstupu "OE"

DT údaj : boolovský vektor z $\{0,1\}^8$
signál: boolovská hodnota z {0,1}

PORTY
vstup DI udaj
LD,R,C signál
vstup DO udaj
TR stbi(DI, up(C=1), T_{Dset},T_{Dhold})

OPER STAV DO udaj

agent Update(T_{DO})
TE es, ef;
KM (DI=d, LD=a; es; u).(u ; ef; DO=o);
vs o = ak a=1 tak d
g DO := ak a=1 tak d inak DO
TR del(up(C=1), up(DO=d), T_{DO}),
afto(up(C=1), ef, T_{DO}),

agent Resetop(T_{RA})
TE es,ef;
KM (u;es;u)(u;;DO=0)
g DO := 0;
TR del(up(R=1), up(DO=0), T_{RD}) ; afto(up(C=1), ef, T_{RD}) ;
START Update up(C=1) ;
Reset up(R=1) ;

Alebo **druhý variant** (so vstupom "OE")

System REGISTER (T_{Dset},T_{Dhold},T_{SA},T_{SD},T_{RA})
DT udaj : boolovský vektor z $\{0,1\}^8$
Zsignál: Z // hodnota tretieho stavu, plávajúci výstup
Zvektor: (Z,Z,Z,Z,Z,Z,Z,Z)
signál: boolovská hodnota z {0,1}
PORTY
Vstup DI udaj
vystup LD, OE, R, C signál
DO udaj, Zvektor
TR stbi(DI, up(C=1), T_{Dset},T_{Dhold})
OPER STAV A udaj

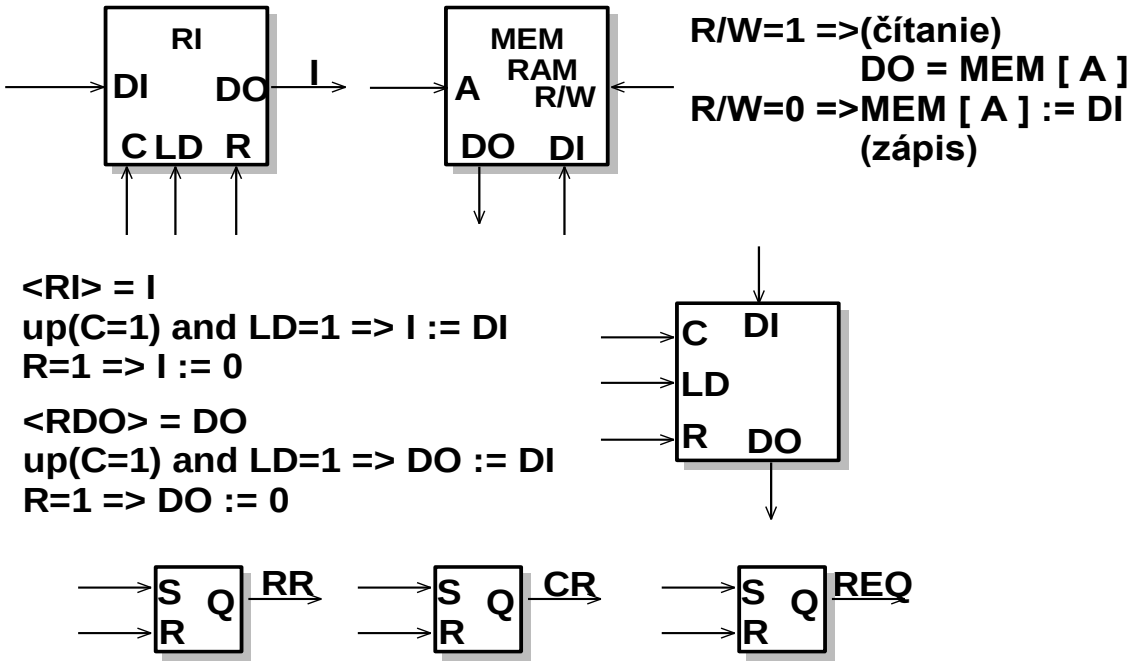
agent Update(T_{SA})
TE es, ef → up(C=1);
KM (DI=d, LD=a; es; u).(u;ef;u);
vs A = ak a=1 tak d inak <A>;
TR del(up(C=1), up(A=d), T_{SA}),
afto(up(C=1), ef, T_{SA});

agent SetOutput(T_{SD})
TE es, ef → up(C=1)
KM (OE=b;es; u)(OE=b;ef;DO=A);
g DO = ak b=1 tak <A> inak (Z,Z,Z,Z,Z,Z,Z,Z);
A := A;
TR del(es, up(DO=A), T_{SD}), afto(es, ef, T_{SD});
stbe(OE, es, ef)
agent Resetop(T_{RA})
TE es,ef → up(C=1)
KM (u;es;u)(u;ef;u) ;
g A := 0;
TR del(up(R=1), up(A=0), T_{RA}) ; afto(up(R=1), ef, T_{RA}) ;
START Update up(C=1);
SetOutput (up(OE=1) or up(C=1) ;
Reset (up(R=1) ;

restr **c**(Update,Setoutput) ; **c**(Setoutput, Update, Reset) ; **c**(Reset,Setoutput)

Pamäťové prostriedky sa vyberajú z **knižníc** (katalogov) prvkov zameraných na danú **technológiu** implementácie alebo z knižníc **virtuálnych súčiastok** (IP components, „Intellectual property components“). V prednáškach sa budeme orientovať na všeobecnejší prístup a budeme sa venovať iba štruktúrnej implementácií na úrovni RT so všeobecne špecifikovanými prvkami.

Pamäťové prostriedky priradené stavovým premenným v OČ vo **VSYS** vidno na obr.



Úlohy na riešenie:
Zostavte špecifikácie tu uvedených prvkov RI, MEM,..,SR-PO zavedenou metódou !!

2. Funkciám (jednoduchým priamo operáciam, ktoré možno jednoducho implementovať) priradíme **funkčné jednotky**. Na základe vzťahu medzi požiadavkami na rýchlosť spracovania informácie a cenu, prípadne spotrebu energie urobíme rozhodnutie o **súbežnom** alebo **sekvenčnom** vykonávaní operácií a podľa neho prideľujeme funkčné jednotky (vrátime sa k tomu neskoršie). Dva extrémne prípady sú:
a/ bez paralelizmov, spravidla s najnižšou cenou a rýchlosťou
b/ s maximálnym možným paralelizmom, spravidla s najvyššou cenou a rýchlosťou

Možná je optimalizácia podľa daných kritérií (pozri osobitnú kapitolu 6)

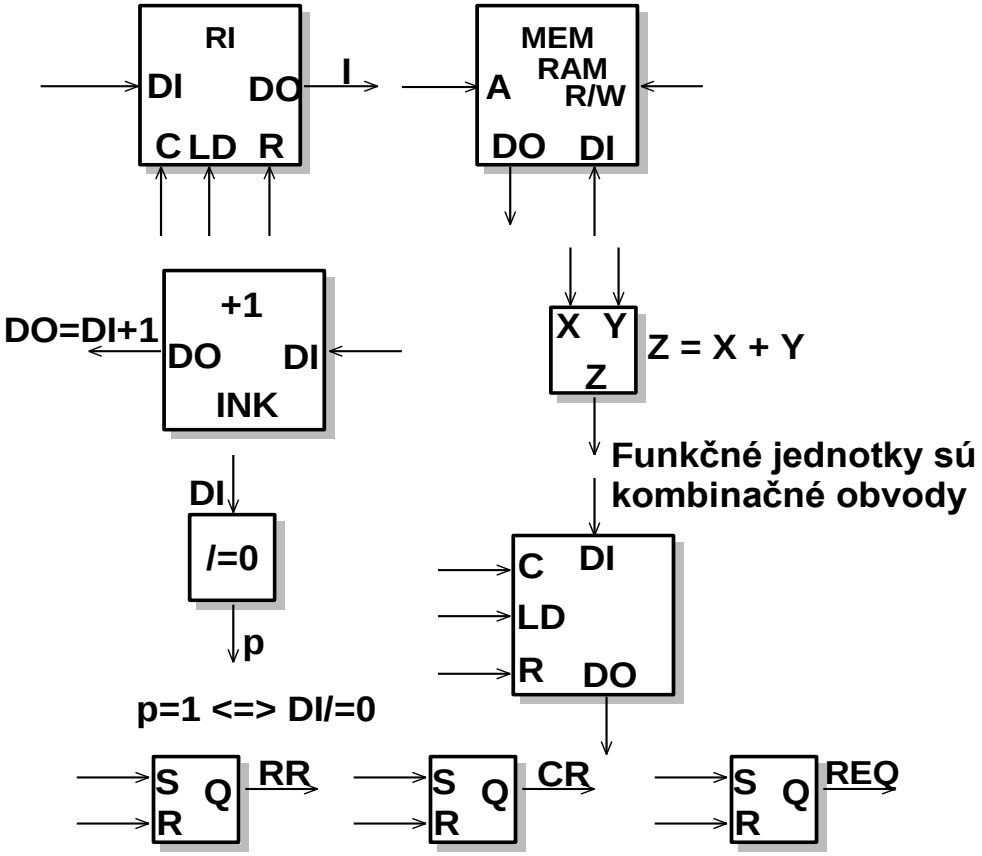
PRÍKLAD 1: Z GTU VSYS extrahujeme informáciu o potrebných funkčných jednotkách.

Funkcie: **+** ➔ sčítačka pre dve celé čísla v doplnkovom kóde s dĺžkou 16 bit;
+1 ➔ inkrementačný obvod pre 4-bitové nezáporné čísla
→ ➔ je automaticky riešiteľné pri registroch

s riadeným zápisom údaja, resp. pri
pamäti RAM s adresovacím vstupom a riadením zápisu
a čítania RAM a pri výstupoch systému

/=0 ➔ kombinačná logika s funkciou: p = 1 <=> DI /= 0

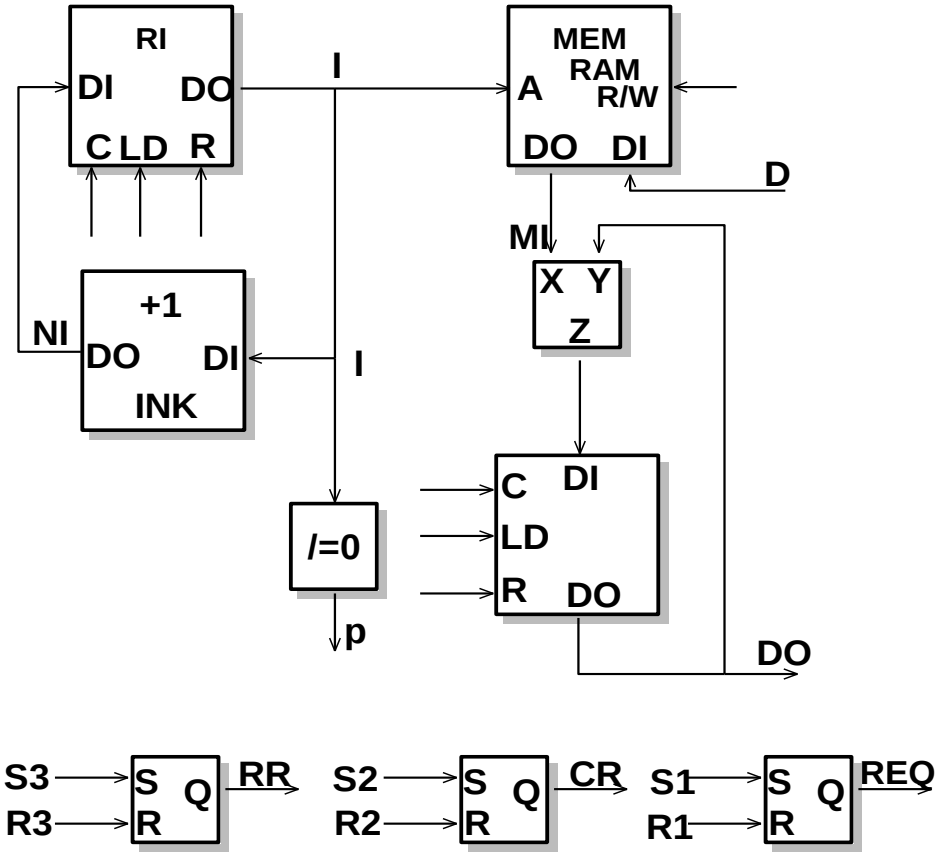
Pamäťové prostriedky a funkčné jednotky OČ vo VSYS sú na nasledujúcom obr.:



3. Vytvorenie údajových ciest podľa **GTU** podľa potreby pomocou multiplexných obvodov (t.j. štandardných multiplexorov, alebo zberníc s riadenou selekciou prístupu k linkám zbernice). Ide o prepojenie výstupov funkčných jednotiek so vstupmi pamäťových prostriedkov (registrov alebo súboru registrov), t.j. tzv. **výstupných väzieb** (fan-out); resp. o prepojenie výstupov pamäťových prostriedkov so vstupmi funkčných jednotiek, t.j. tzv. **vstupných väzieb** (fan-in). Informácia o požadovaných prepojeniach je **explicitne** obsiahnutá v **GTU**.

Poznámka: V jednoduchých prípadoch nie je potrebné **výberové prepínanie** údajových ciest, t.j. prepojenia dané v GTU sú **fixné**. Takýto prípad predstavuje i **OČ** vo **VSYS**.

PRÍKLAD 1: **Fixné** (neriadené) **prepojenia** OČ vo VSYS možno priamo extrahovať z **GTU** pre VSYS. Sú viditeľné v nasledujúcom obr.



Dostali sme štruktúru OČ. V danom prípade sa nevyskytuje potreba riadeného prepínania údajových ciest. Žiaden multiplexor nie je potrebný (Koniec PRÍKLADU 1).

Vo všeobecnosti, ak sa vyskytne potreba použitia viacerých multiplexorov (zberníc), možno počet mux **optimalizovať** podľa kritéria **minimálneneho počtu výberových spojení** (do jednoho vstupu niektorého HW prostriedku z viacerých výstupov iných prostriedkov), z ktorých každé vyžaduje **jeden** multiplexor. Ako ukazovateľ optimálnosti riešenia tu môže slúžiť **minimálny počet** multiplexorov alebo aj minimálny **celkový počet vstupov** všetkých multiplexorov (uvedieme v kapitole 6).

PRÍKLAD 2: Zostavíme implemetačnú štruktúru **OČ CPU** (na úrovni RT, t.j. s hotovými (knižničnými) pamäťovými a fumkčnými prvkami.

Výjdeme z gloálneho procesu P_S, ktorým je proces PrGlobal

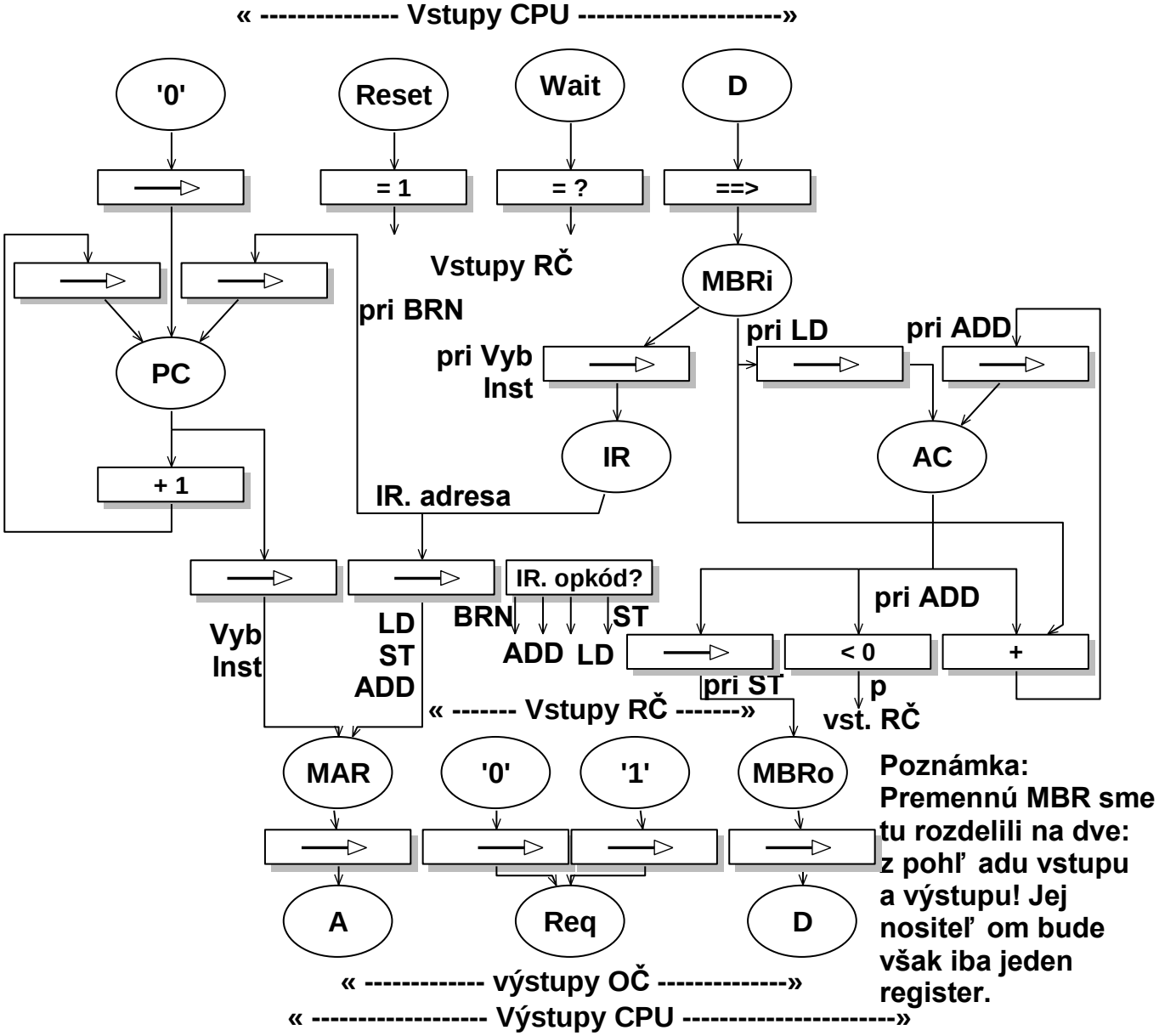
```
PrGlobal = Reset.[PrInstVyb. ((IR.opkód=BRN): PrBRN,
                              (IR.opkód=ADD): PrADD,
                              (IR.opkód=LD):  PrLD,
                              (IR.opkód=ST):  PrST )  ]ω;
```

Jednotivé sub-procesy v PrGlobal sú:

PrInstVyb = (LdMAR || IncrPC).[EA]$^{(Wait=1)}$ [SetMRd]$^{(Wait=0)}$.
[LdIR]$^{(Wait=1)}$;
PrST = (MovIR-MAR // LdMBR).[SetMWr]$^{(Wait=0)}$;
PrBRN = (AC<0): EA, (AC >=0): MovIR-PC ;
PrLD = MovIR-MAR.[SetMRd]$^{(Wait=0)}$.LdAC ;
PrADD = MovIR-MAR.[SetMRd]$^{Wait=0)}$.AddAC;

Po analýze PrGlobal možno priamo zostaviť **graf toku údajov** (**GTU**) s jednoduchými funkciami na mieste operácií :

GTU CPU:



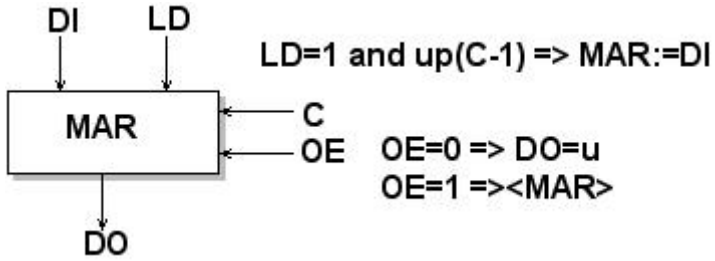
Poznámka: Premennú MBR sme tu rozdelili na dve: z pohľ adu vstupu a výstupu! Jej nositeľ om bude však iba jeden register.

Funkcie priradení (→) sme vložili pri registroch, čo značí, že ide o funkciu **zápisu do registra**. Priradenie hodnôt premennej **Req** sa deje priamo v **RČ**. Ak to nevedie na omyl priradenia → v **GTU** často vynechávame. Z odvodeného **GTU možno prejsť na**

štruktúru OČ

Alokácia HW prvkov:

MAR ... 32-bit register s riadeným synchronizovaným zápisom údaja a
s riadeným výstupom (s trojstavovými hradlami na výstupe prekl. obvodov).

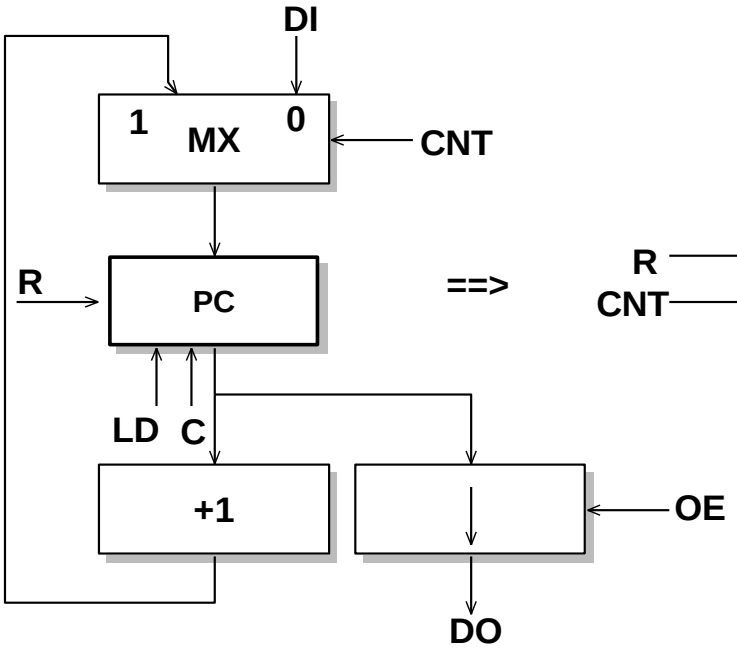


MBR 32-bit register (ešte spresníme)

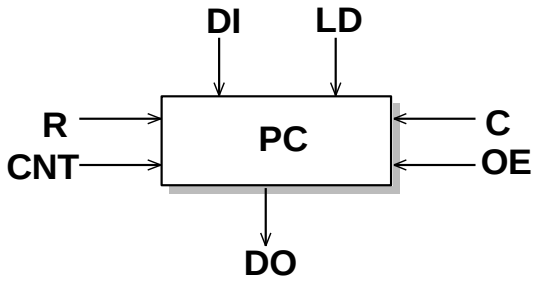
IR 32-bit register, detto ako MAR

AC 32-bit register s riadeným synchronizovaným zápisom,bez riadeného výstupu, musí mať **schopnosť pracovať v uzavretej sľučke** (ako v sekv. obvode) cez ALU.

PC,**+1**...Združenie registra a inkrementačného obvodu do **počítadla** (s menom **PC**) s možnosťou **zápisu** údaja,riadenia výstupu a **nulovania**. Nulovanie Realizuje priradenie PC := 0, ktorého potreba vyplýva z **GTU**.

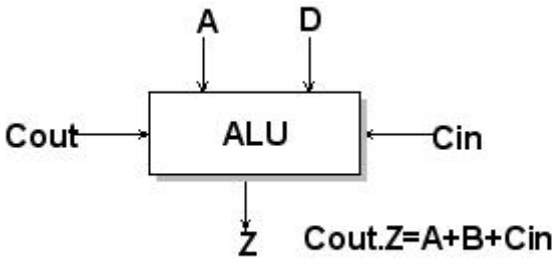


==>



CNT=1 and up(C=1) => PC := PC + 1
LD=1 and up(C=1) => PC := DI
R =1 => PC := 0
OE = 1 => DO = PC
OE = 0 => DO = u

+ **ALU**, 32-bitová binárna sčítačka



(Cout.Z tu značí zreťazenie hodnoty Cout s hodnotou Z !!!)

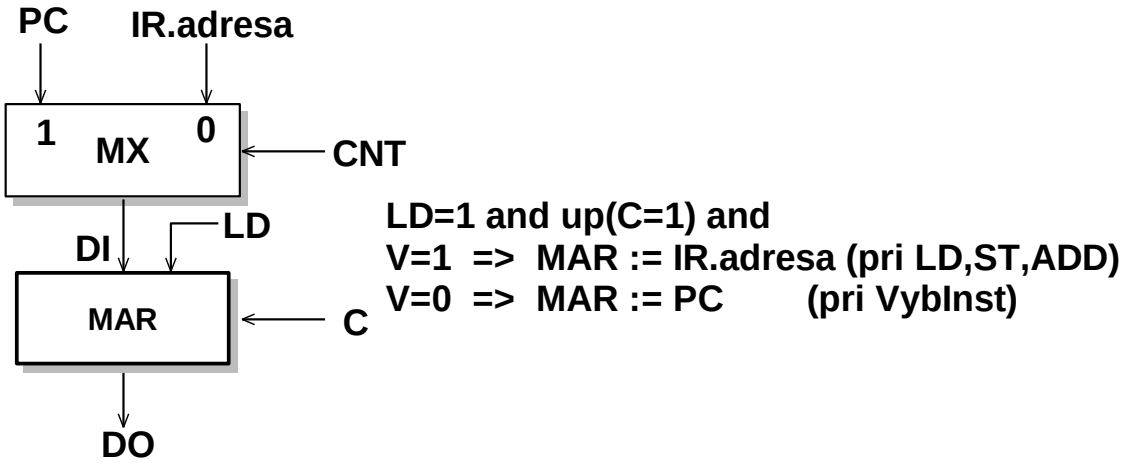
Poznámka: Dekódovanie **IR.opkod** a podobne aj vyhodnotenie predikátu **p** možno **presunúť** do **RČ**, čo urobíme. Teda vstupmi **RČ** bude príslušné **pole** v **IR** pre opkod a (vzhľadom na doplnkový kód) **znamienkový bit** z **AC**, ktorý je nositeľom hodnoty predikátu **p**.

Úlohy na riešenie: Zostavte (formálne) špecifikácie tu uvedených prvkov (v opísanom špecifikačnom modeli a vo VHDL) !!!!!

Zostavenie štruktúry operačnej časti:

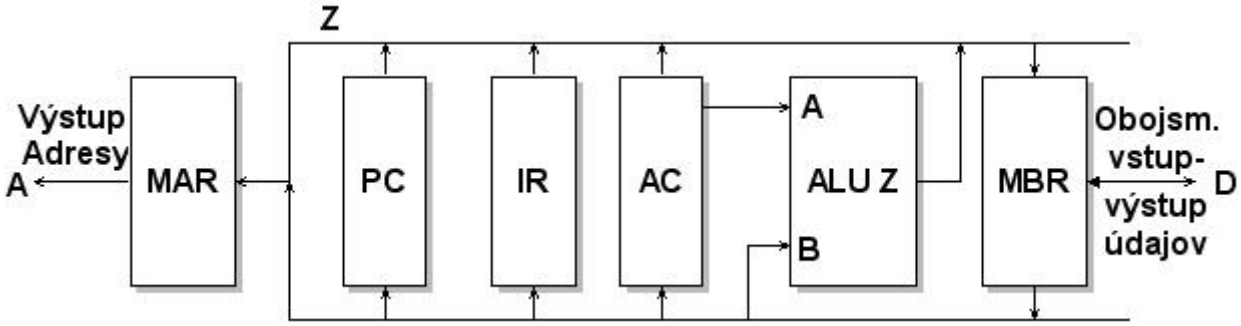
Mohli by sme postupovať aj tak, ako v PRÍKLADE 1 a vytvoriť štruktúru s **dvoj-bodovými prepojeniami** medzi registrami, počítadlom a sčítačkou; s použitím multiplexora na vstupe **MAR**, kde je potrebný **riadený výber** (pozri obr.).

Pripojenie vstupu registra (prvku) podľa výberu na viaceré vstupy (výstupy registrov PC a IR) pomocou MX s riadením je vidno na obr.



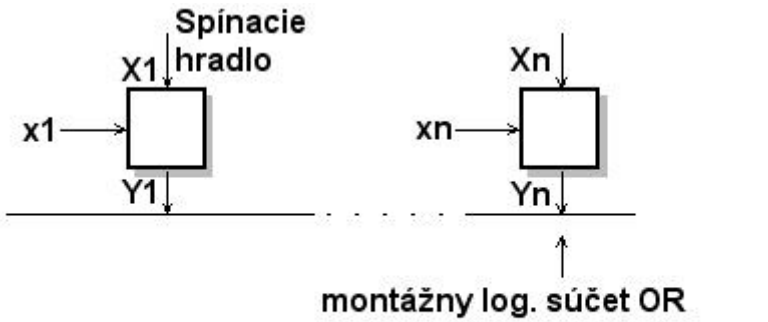
Využijeme tento príklad na uvedenie **všeobecnejších prepájacích štruktú**r typických pre jednoduché procesory daného typu. Ide nám o poukázanie na častý spôsob riešenia pri jednoduchých procesoroch na spracovanie dát. V našom príklade by sme, pravda, vystačili aj s prepájacím systémom so štandardnými multiplexormi (2 až 3 kusy s dvoma vstupnými údajovými kanálmi) šitým na mieru.

A. Prepájací systém s **jednou zbernicou** (zbernica <=> **bus**)



ALU (Aritmetic-Logic Unit, aritmeticko-logická jednotka) je v našom prípade iba sčítačkou

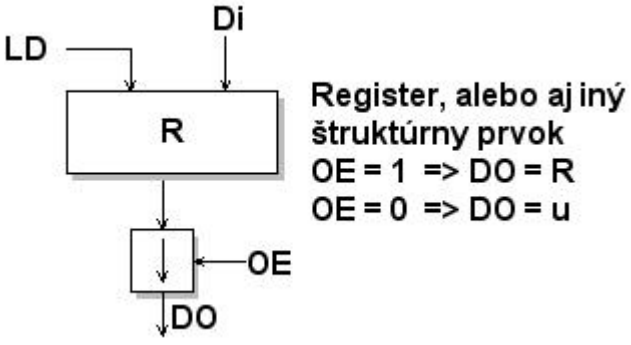
Pripojenie výstupov registrov X_i na zbernicu (**bitový rez**) vidno na nasledujúcom obr.



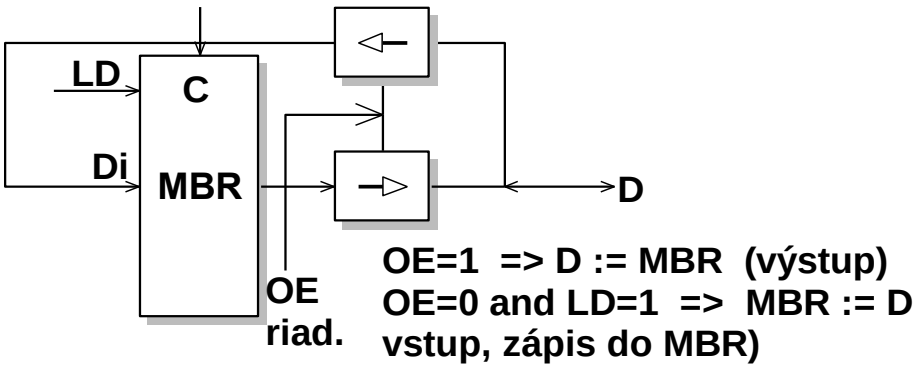
xi = 1 => Yi = Xi
xi = 0 => Yi a teda aj Z odpojená od Xi(tretí stav)

Vstupy DI registrov s **riadeným zápisom** sa pripojujú na Z bezprostredne. Inak je potrebné "demultiplexovanie" (pozri ďalej).

Spínacie hradlá na pripojenie výstupov registrov na zbernicu môžu byť aj ich súčasťou. Takéto registre (prípadne aj iné prvky) majú už spomenutý **riadený výstup** (pozri obr.).



Obojsmerný vstup-výstup údajov sa obyčajne rieši pomocou spínacích hradiel tak, ako to ilustruje nasledujúci obr.



Zbernica **Z** so systémom spínacích hradiel je vlastne **distribuovaný** (**neštandardný**) **multiplexor** s funkciou, ktorú možno vidieť na obr. Ide o „multiplexovanie" 5 vstupov zbernice (výstupov uvedených HW prostriedkov) na jeden vstup **Z** (na obr. sú znázornený bitový rez), ktorý je nositeľom údaja na zbernici a ktorý sa v danom prípade rozvetvuje do vstupov HW prostriedkov (PC, IR, …,vstup B sčítačky, a MAR). V danom prípade platí
$\mathbf{x_i = 1}$ tak $\mathbf{vstup_i}$ je prepojený na **Z**, teda **Z**= $\mathbf{vstup_i}$
V danom prípade je počet riadiacich vstupov $\mathbf{x_i}$ rovný počtu údajových vstupov
Treba poznamenať, že o zbernici sa nevhodne hovorí aj v prípadoch ak sa multiplexovanie nerobí, čo je nesprávne a neprofesionálne.
Je zrejmé, že **súbežnosť** (paralelizmus) prenosov medzi registrami **nie je** v tomto prípade **možná**. V danom základnom (hodinovom) cykle sa môže prenášať iba **jeden** vstupný **údaj** privádzaný na zbernicu **Z**.

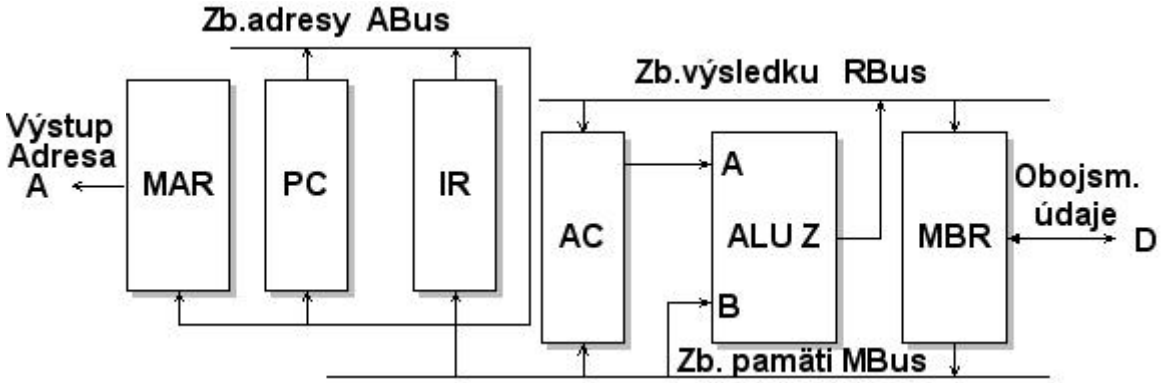
Jednoduché **rozvetvenie** zo **Z** do vstupov registrov je možné iba pri **riadení zápisu údaja** do nich. Pri neriadenom zápise do registrov je potrebné zvlášť riešiť tzv. **demultiplexovanie** (pozri ďalej).

Ak použijeme toto prepojenie v **OČ** CPU, potom napr. pre inštrukciu ADD potrebujeme **4** základné hodinové cykly (pri potrebe jedného cyklu pre čítanie operandu z pamäte).

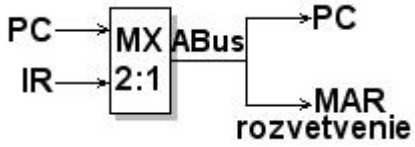
Cyklus **1**: Z := IR.adresa;	
MAR := Z;	| Adresovanie a
	| vybratie
Cyklus **2**: Čítanie údaja z pamäte M na D;	| operanda z **M**
MBR := D;	| a zápis do **MBR**
Cyklus **3**: Z := MBR;	|
vstup B sčít.:= Z; (vstup A sčít.= AC)	| **X+Y**
Cyklus **4**: Z := Výst.Z sčít.;	| Zápis
AC:= Z	| výsledku
	| X+Y do **AC**

Ak uvážime, že vybratie inštrukcie z pamäte potrebuje **3** cykly, potom celý inštrukčný cyklus potrebuje **6** základných cyklov

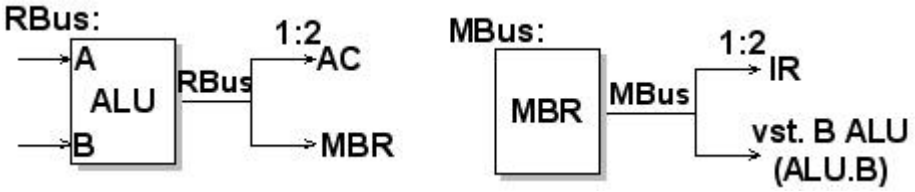
B. Prepájací systém s **viacerými** zbernicami (na obr. je bežný systém s **3**-mi zbernicami). Zvýšenie stupňa paralelizmu (súbežnosti)



Na uvedených **3** zberniciach sa môžu prenášať **3** údaje paralelne.
Adresová zbernica - ABus predstavuje **multiplexor** 2:1 (obr.)

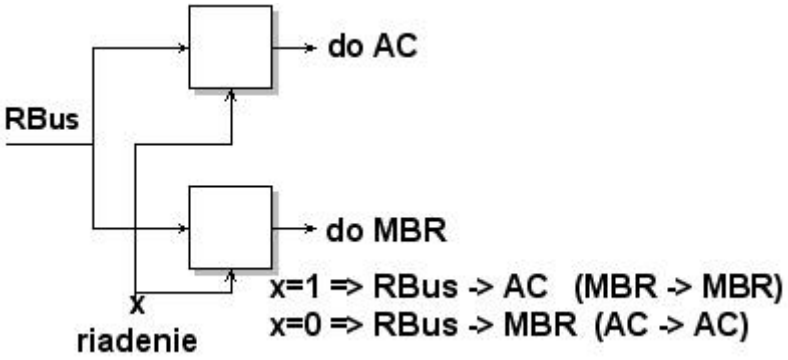


Ostatné dve zbernice v danom systéme realizujú **iba rozvetvenie** (pozri obr. dole) a nie sú to vlastne zbernice.**!!!!**



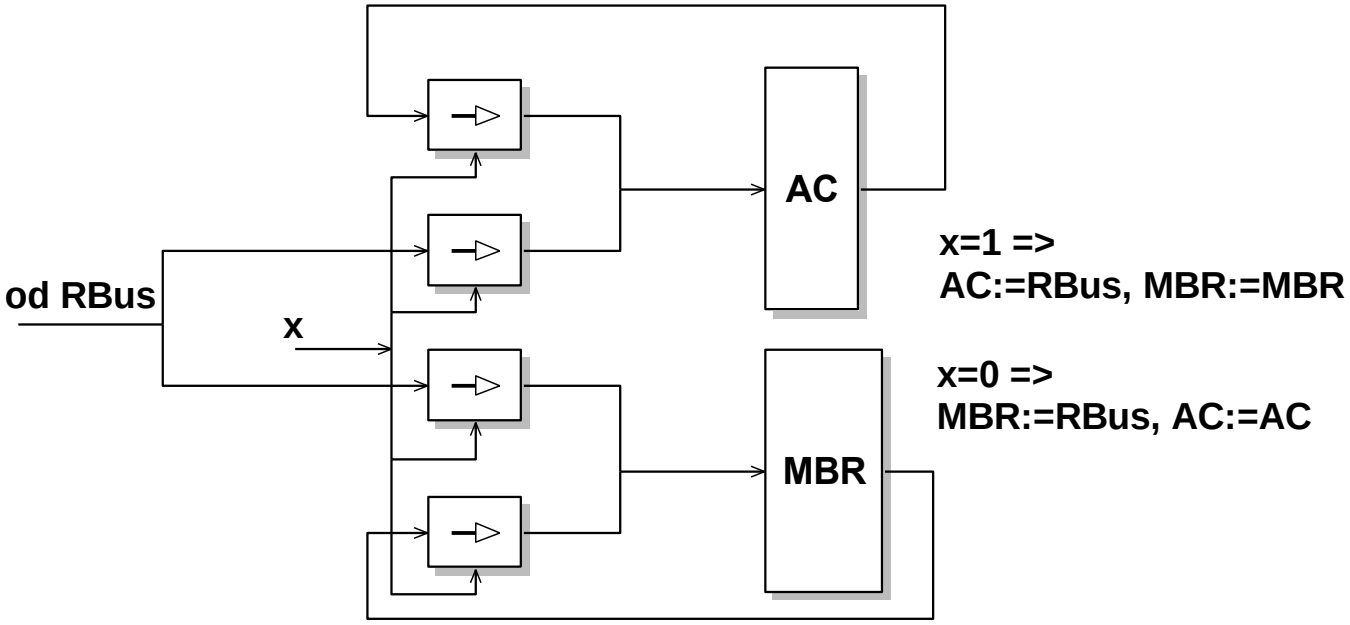
Treba však poznamenať, že jednoduché rozvetvenie tu možno použiť iba preto, lebo výber a zápis údaja do **cieľových** registrov sa uskutoční prostredníctvom **riadeného zápisu** (pri **LD**=**1**). Ak by zápis nebol riadený a údaj v každej vetve by sa automaticky zapisoval do cieľových registrov v každom cykle, potom by sme museli namiesto rozvetvenia použiť tzv. **demultiplexovanie** (1:n) napr. tak, ako je to na nasledujúcom obr.

Demultiplexovanie 1:2

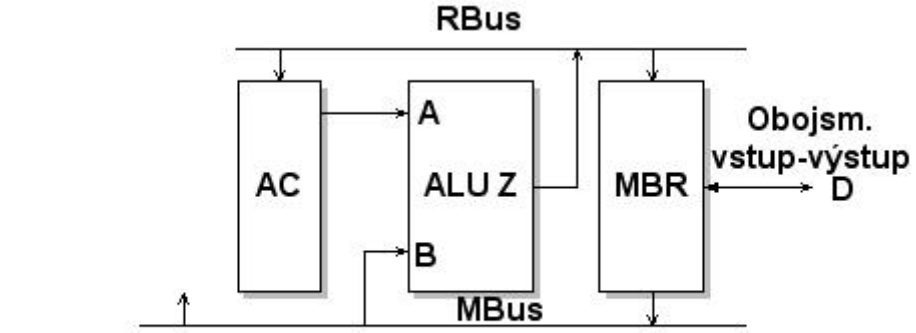


Pri riadenom zápise do registrov sa vlastne demultiplexovanie robí vo **vstupných obvodoch** registrov pre riadený zápis

Pri demultiplexovaní sa pre_nevybraté vstupy prostriedkov sa spravidla vyžaduje ich **odpojenie** (**tretí** stav) a pri nevybratých registroch sa tiež vyžaduje zabezpečenie uchovania ich pôvodného obsahu (pozri napr. obr. hore: AC:=AC). Kompletné riešenie, v ktorom je zabezpečenie uchovania pôvodného obsahu v nevybratom registri je ilustruje napr.nasledujúci obr.



V opísanom 3-zbernicovom prepájacom systéme možno prenos z registra AC do MBR, a podobne aj naopak, organizovať **cez ALU**, t.j. v našom prípade cez sčítačku.



Prenos údaja z MBR do AC, t.j. AC := MBR sa organizuje takto:
MBR --> MBus --> ALU.B --> ALU.Z --> RBus --> AC

Pri prenose je tu je potrebné ALU nastaviť tak, aby na vstupe **ALU.A** bol **nulový vektor**, t.j. **číslo 0**. Vtedy bude RBus=ALU.Z := Mbus = MBR.

Pri troj-zbernicovom systéme, napr. pri **exekúcií inštrukcie ADD** je situácia takáto:

```
Cyklus 1:   ABus := IR.adresa;
            MAR  :=  ABus;                          | Adresovanie a
                                                    | vybratie

Cyklus 2:   Čítanie údaja z pamäte M na D;          | operanda z M
            MBR := D;                               | a zápis do MBR

Cyklus 3:   MBus  := MBR;                           | Súčet
            ALU.B := MBus;      (vstup X sčít.= AC) |  X+Y;
            RBus  := ALU.Z;                         | zápis
            AC    :=  RBus                          |výsledku
                                                    | X+Y do AC
```

Ušetrí sa jeden základný cyklus (pri vykonaní **ADD** je to zrýchlenie Pri čítanie v 3

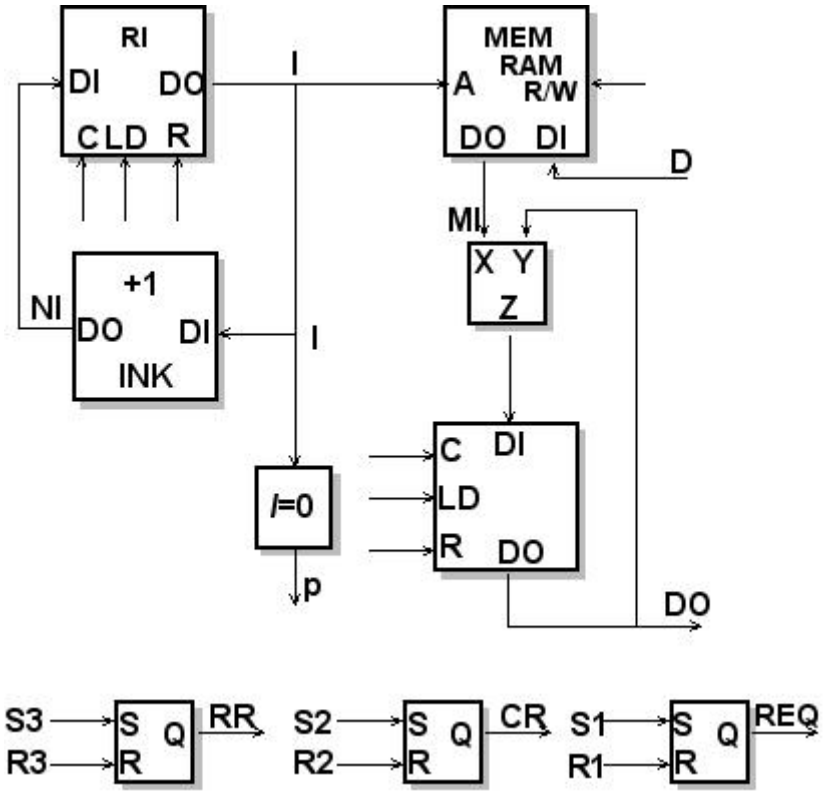
hodinových cykloch z vonkajšej pamäti budeme potrebovať **5** hodinových cyklov.

V našom prípade **nie je možné využiť** potenciálne daný **stupeň** paralelizmu (3 súbežné údaje) prepájacieho systému. Bránia tomu **údajové závislosti**, ktoré vyžadujú postupné, sekvenčné spriahnutie operácií.

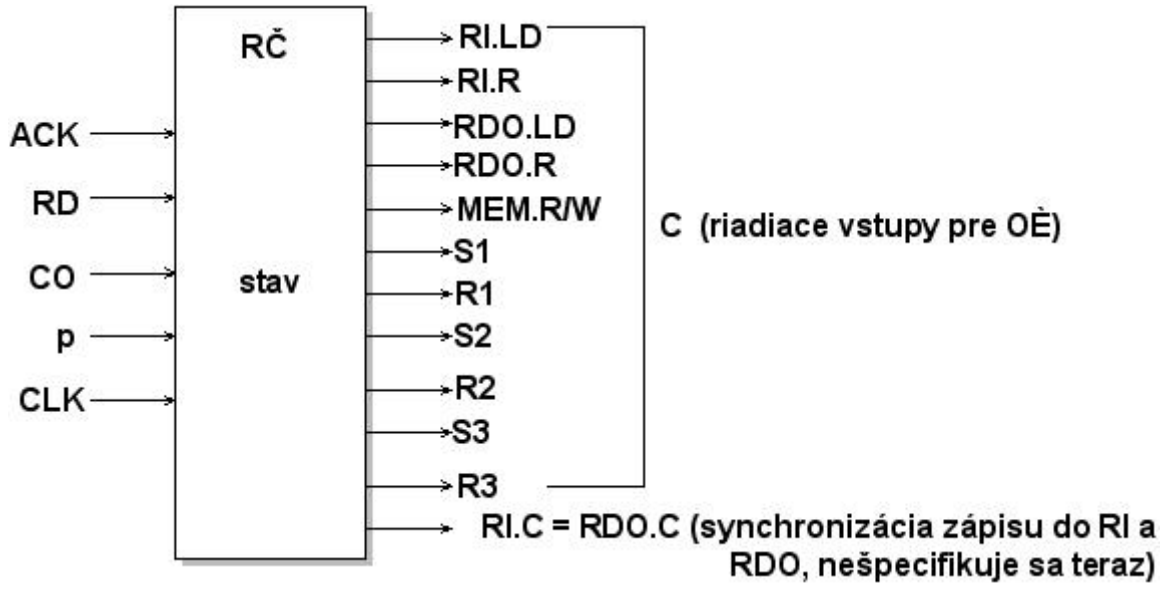
3. Zostavenie špecifikácie riadiacej časti

Globálny proces $\mathbf{P_S}$ a graf toku riadenia implicitne **špecifikujú správanie RČ** a sú **východiskami** pre jej syntézu. Odvodenie **kompletnej** špecifikácie **RČ** (t.j. nielen jej funkcie v d-čase) uvedieme pomocou príkladu **VSYS**. Základom špecifikácie funkcie **RČ** v d-čase je **prechodová** a **výstupná** funkcia stavového stroja, ktorý zodpovedá $\mathbf{P_S}$. Pri zostavení týchto funkcií budem v danom príklade predpokladať **Mooreov** FSM.
PRÍKLAD 1: Zostavíme úplnú špecifikáciu riadiacej jednotky systému VSYS.

Štruktúra OČ:



Vstupy a výstupy – rozhranie **RČ** s **OČ** a s okolím:



system RJ-VSYS // funguje ako Mealyho FSM táto špecifikácia sa dá prepísať do VHDL**!!**
PORTY // vynechame DT
vstup IV=(RD,CO,ACK,p) boolovský vektor; // p=1 <=> (I /=0);
CLK boolovská hodnota; // hodiny
vystup OV = (RI.LD, RI.R, RDO.LD, RDO.R, MEM. R/W,S1, R1, S2, R2, S3, R3)
booleovský vektor;
OPER STAV stav symbol ∈ { L0, L1, L3, L4, L5, L6, L7, L8, L10, C1, C2, C3, C5 }
proces RiadPr = [RiadCyklus]$^{\omega}$

agent RiadCyklus
TE es, ef → up(CLK=1);
KM (IV=v ;es ; u)(u; ef; OV=h);
vs pre prípad "stav,v" h
L0, RD=1 RDO.R=RI.R=R1=R2=R3=1 ;
L0, CO=1 RDO.R=RI.R=R1=R2=R3=1 ;
L0, RD=1 and C0=0 u ;
L1, ACK=0 S1=1 ;
L1, ACK=1 u
L3, ACK=1 u ;
L3, ACK=0 S1=1 ;
L4, u, u ;
L5, ACK=1 R1=1
L5, ACK=0 u ;
L6, u MEM.RD/WR=0 ;
L7, u RI.LD=1 ;
L8, p=1 and ACK=0 S1=1 ;
L8 p=1 and ACK=1 u ;
L8, p=0 S3=1 ;
L10 u, u ;
C1, u, DO.LD ;
EM.RD/Wr= ;
C2, u RI.LD=1 ;
C3, p=1 u ;

```
                    C3,  p=0            S2=1
                    C5,  u,             u

    g    stav :=  pre prípad "stav,v"      je
                  LO, RD=1               L1;
                  L0, CO=1               C1;
                  L0, RD=1 and C0=0      L0;
                  L1, ACK=0              L4;
                  L1, ACK=1              L3;
                  L3, ACK=1              L3;
                  L3, ACK=0              L4;
                  L4,  u                 L5;
                  L5, ACK=1              L6;
                  L5, ACK=0              L5;
                  L6,  u                 L7;
                  L7,  u                 L8;
                  L8, p=1 and ACK=0      L4;
                  L8  p=1 and ACK=1      L3;
                  L8, p=0                L10;
                  L10, u                 L0;
                  C1   u                 C2;
                   C2,  u                C3;
                  C3, p=1                C2;
                  C3, p=0                C5;
                  C5,  u                 L0;
// pre provnanie  uvadzame programovú schému P_S
// L0: EA / RD: L1,  CO: C1,   (RD nor CO): L0
// L1: Reset / L2;
// L2: ACK: L3, L4
// L3: EA / L2
// L4: SetReq / L5;
// L5: EA / ACK: L6, L5;
// L6:ResReq / L7
// L7: WriteM / L8;
// L8: InkrI / L9;
// L9: (I/=0) and (ACK=0): L4,  (I/=0) and (ACK=1): L3,  (I=0): L10;
// L10: SetRR / L0
// C1: Reset / C2;            povodne  pouzivane návestia R1,..,R5 sme označili
// C2:Add / C3;               C1,...,C5 kvôli vylúčeniu zámeny s menami "reset"
// C3:InkrI / C4;             vstupov R1,R2 a R3 prekl. obvodov
// C4:(I/=0): C2, C5;
// C5:SetCR / L0

TR…….afto (up(OV=h), ef, 0),
          stabi (IV, up(CK=1), Ts, Th)      //Ts,Th su generické parametre

START       RiadPr      ez and (stav=L0);
     restr      RD nand CO ;   RR=1 => (notRD); CR=1 => (notCO)
```

PRÍKLAD 2: Zostavíme úplnú špecifikáciu **riadiacej jednotky CPU**.
Vychádzame pritom z toku riadenia implicitne zadanom v globálnom procese

$\mathbf{P}_S$= PrGlobal

PrGlobal = Reset.[PrInstVyb. ((IR.opkód=BRN): PrBRN,
(IR.opkód=ADD): PrADD,
(IR.opkód=LD): PrLD,
(IR.opkód=ST): PrST)]$^{\omega}$; kde

PrInstVyb = (LdMAR || IncrPC).[EA]$^{(Wait=1)}$ [SetMRd]$^{(Wait=0)}$.[LdIR]$^{(Wait=1)}$;

PrST = (MovIR-MAR // LdMBR).[SetMWr]$^{(Wait=0)}$;

PrBRN = (AC<0):EA, (AC >=0): MovIR-PC;

PrLD = MovIR-MAR.[SetMRd]$^{(Wait=0)}$.LdAC;

PrADD = MoveIR-MAR.[SetMRd]$^{(Wait=0)}$.AddAC;

Stavový stroj, ktorý špecifikuje riadiacu jednotku sa najľahšie odvodí z **programovej schémy** procesu PrGlobal, ktorý obsahuje iba jednocyklových agentov (**mikrooperácie**) a zodpovedá vlastne **mikroprogramu**. **Návestia** zodpovedajú stavom automatu a prechodová a výstupná funkcia sa odvodí priamo z programovej schémy

PrGlobal (programová schéma)

RES: Reset/ IF0; // vedie na Moorov FSM
IF0: LdMAR//InkrPC/ IF1;
IF1: EA / Wait=1: IF2, IF1;
IF2: SetMRd / Wait=1: IF2, IF3;
IF3: LdIR / Wait=1: OD, IF3;
OD: EA / IR.opk=LD**:** LD0, IR.opk=ST**:** ST0, IR.opk=ADD**:** AD0, IR.opk=BRN**:** BR0;,
LDO: MovIR-MAR/ LD1;
LD1: SetMRd / Wait=1: LD1, LD2;
LD2: LdAC / IF0;
ST0: MovIR-MAR || LdMBR / ST1;
ST1: StMWr / Wait=1: ST1, IF0;
AD0: MovIR-MAR / AD1;
AD1: SetMRd / Wait=1: AD1, AD2;
AD2: AddAC / IF0;
BR0: EA / (AC < 0)**:** BR1, IF0;
BR1: MovIR-PC / IF0;

Tabelárny lineárny zápis FSM, ktorý odvodíme z progr.schémy procesu PrGlobal je:

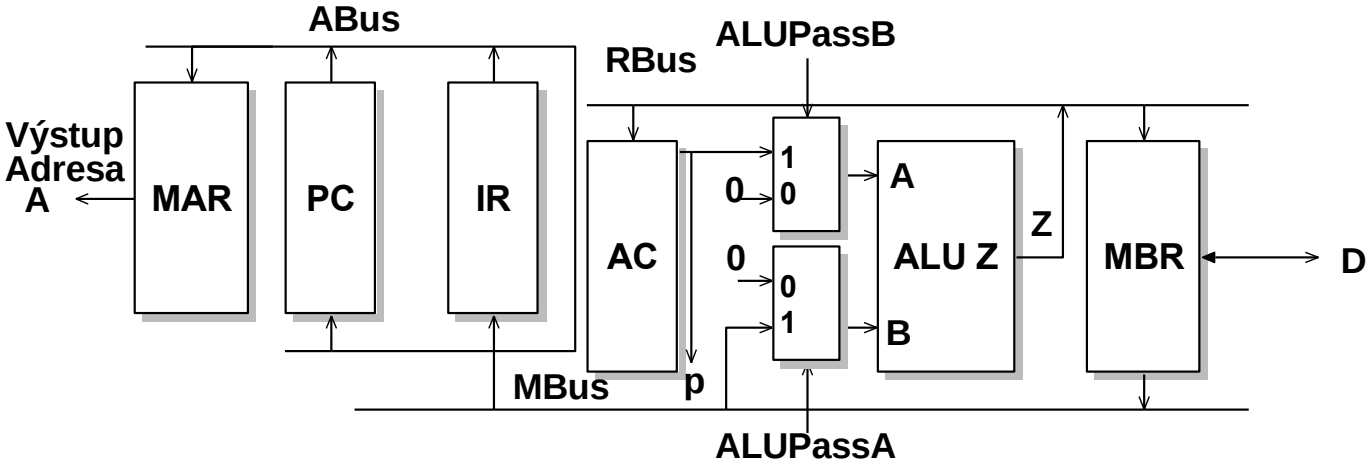
Stav q	Hodnoty vstupov, vst.vekt. v	Nasl.stav p(q,v)	Mikrooperácie R(q)	Procesy v PrGlobal	
RES	**u**	IF0	Reset		
IF0	**u**	IF1	LdMAR || InkrPC	PrIstrVyb	PrGlo
IF1	Wait =0	IF1	EA	PrIstrVyb	PrGlo
IF1	Wait =1	IF2		PrIstrVyb	PrGlo
IF2	Wait =1	IF2	SetMRd	PrIstrVyb	PrGlo
IF2	Wait=0, D=d	IF3		PrIstrVyb	PrGlo
IF3	Wait =0	IF3	LdIR	PrIstrVyb	PrGlo
IF3	Wait =1	OD		PrIstrVyb	PrGlo
OD	IR.opkód=LD	LD0	EA	PrIstrVyb	PrGlo
OD	IR.opkód=ST	ST0			PrGlo
OD	IR.opkód=ADD	AD0			PrGlo
OD	IR.opkód=BRN	BR0			PrGlo
LD0	Wait=1	LD1	MovIR-MAR	PrLD	PrGlo
LD1	Wait=1	LD1	SetMRd	PrLD	PrGlo
LD1	Wait=0, D=d	LD2		PrLD	PrGlo
LD2	**u**	IF0	LdAC	PrLD	PrGlo
ST0	Wait=1	ST1	MovIR-MAR|| LdMBR	PrST	PrGlo
ST1	Wait=1	ST1	SetMWr	PrST	PrGlo
ST1	Wait=0	IF0		PrST	PrGlo
AD0	Wait=1	AD1	MovIR-MAR	PrADD	PrGlo
AD1	Wait=1	AD1	SetMRd	PrADD	PrGlo
AD1	Wait=0, D=d	AD2		PrADD	PrGlo
AD2	**u**	IF0	AddAC	PrADD	PrGlo
BR0	AC<0 (p=1)	BR1	EA	PrBRN	PrGlo
BR1	AC >=0 (p=0)	IF0	MovIR-PC	PrBRN	PrGlo

Upozorňujeme, že z tohto FSM, ktorého výstupy sú agenty (mikroagenty - mikrooperácie na danej úrovni mikroarchitektúry našej CPU) možno priamo odvodiť behavioristický opis **CPU** vo **VHDL** (ako FSM) tak, že pri daných stavoch FSM zabezpečíme priradenie hodnôt deklarovaným stavovým premenným systému (ktoré budú spravidla súčasťou OČ) a výstupom tak, ako to predpisujú (v položke g, resp. vs) agenty (mikrooperácie). Napr. pri LdAC sa realizuje priradenie: AC := MBR.

Riadiace signály pre OČ

Štruktúra a požadované hodnoty riadiacich signálov, vysielaných z RČ pre riadenie OČ (pre exekúciu daných agentov), je priamo daná štruktúrnymi prvkami a 3-zbernicovým prepájacím systémom.

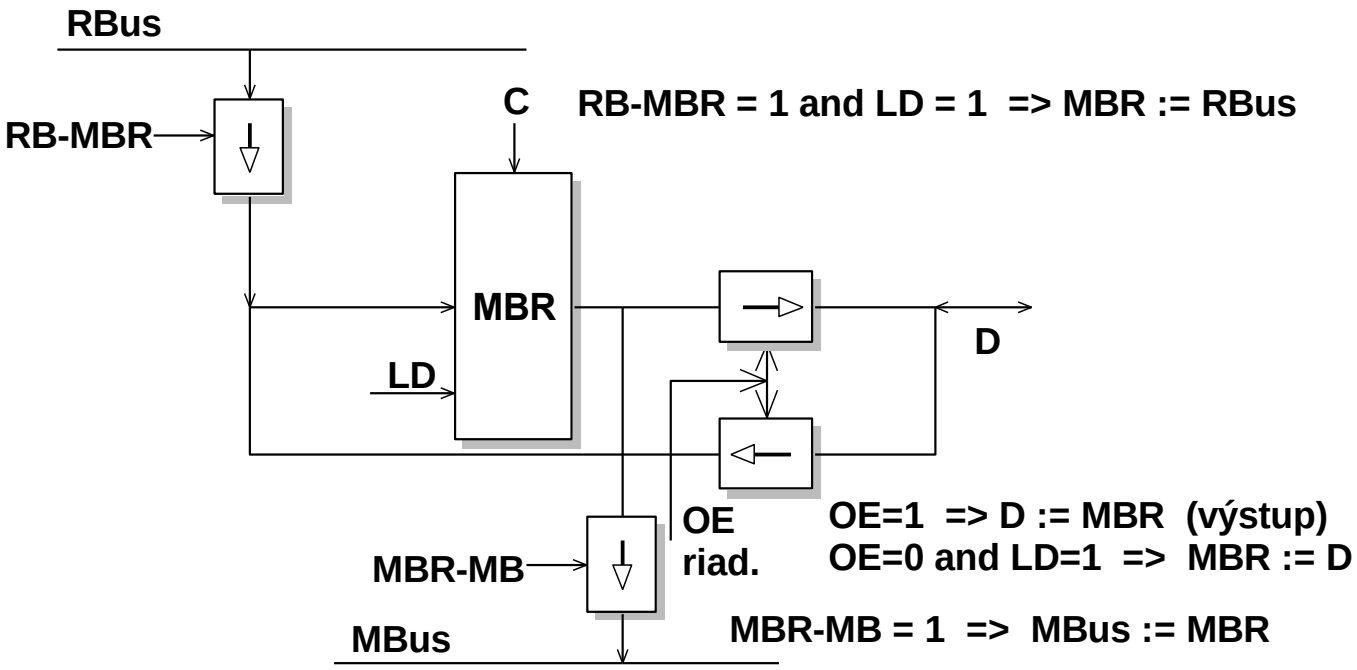
Budeme predpokladať 3-zbernicovú prepájaciu štruktúru (pozri obr.) s predtým špecifikovanými prvkami **MAR**, **PC**, **IR**, **AC**, ALU a **MBR**.



V danej štruktúre musíme ešte špecifikovať **riadenie obojsmerného údajového kanála D** a riadenie prechodu údaja zo vstupu A resp. B cez ALU, ktorý je potrebný v danej koncepcií prepájacich prostriedkov pri prenos údaja z AC do MBR a naopak.

Jeden spôsob využívajúci dva multiplexory 2:1 je na obr.
ALUPassB = 1 and ALUPassA = 0 ==> RBus = MBus + 0 = MBus
ALUPassA = 1 and ALUpasB = 0 ==> RBus = AC + 0 = Mbus

Zvolené riešenie "okolia" registra MBR, v ktorom možno dosiahnuť vyššie vyžadované prepojenia je na nasledujúcom obr. Nastavenie hodnôt riadiacich signálov pre jednotlivé z vyžadovaných prepojení sú zrejmé.



Jednotlivým agentom (mikrooperáciam) zodpovedajú potom nasledujúce **riadiace signály**:

LdMAR || InkrPC: PC.OE=1, MAR.LD=1, PC.CNT=1

SetMRd: MAR.OE=1, Rd/Wr=1, Req=1, MBR.OE=0, MBR.LD=1
RB-MBR=0

LdIR: MBR-MB=1, IR.LD=1

MovIR-MAR: IR.OE=1, MAR.LD=1

LdAC: MBR-MB=1, ALUPassB=1, AC.LD=1

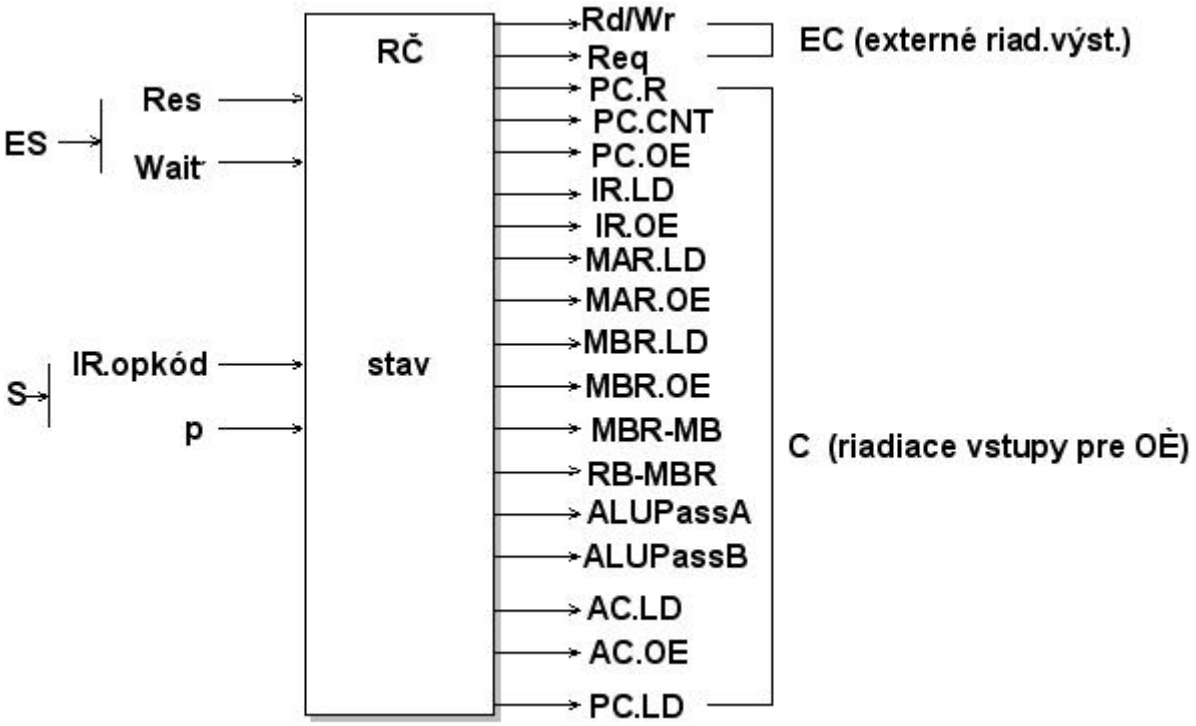
MovIR-MAR || LdMBR: IR.OE=1, MAR.LD=1, ALUPassA=1, RB-MBR=1,
MBR.LD=1

SeMWr: MAR.OE=1, Rd/Wr=0, Req=1, MBR.OE=1

AddAC: MBR-MB=1, AC.LD=1

MovIR-PC: IR.OE=1, PC.LD=1

Vstupy a výstupy - rozhranie **RČ** s **OČ** a s okolím



S,ES...státus: vstupy z operačnej časti (S), resp z okolia (ES)

Teraz zostavíme špecifikáciu **RJ** ako digitálneho systému použiívanými špecifikačnými prostriedkami. Opíšeme, vlastne, konečný stavový stroj (FSM), ktorý opisuje správanie RJ v reálnom čase v hodinovom cykle (ako sme to už urobili v pripade RJ pre VSYS). Pôjde o opis s jednocyklovými agentami RiadCzklus a Reset. Z takejto špecifikácie možno jednoducho prejsť na opis správania vo VHDL.

Kompletná špecifikácia RČ na úrovni RT s rešpektovaním štartovacieho mechanizmu môže byť takáto:

RiadJedn-CPU // vynechame kolonku DT

PORTY

vstup IV vstupny vektor (Res, Wait, IR.opkod, p)

Res boolovska hodnota

Wait boolovska hodnota

IR.opkód symbol ∈ { LD,ST,ADD,BRN }

p boolovska hodnota

CLK boolovska hodnota

vystup OV vystupny vektor (Rs/Wr,….., AC.OE)

Rd/Wr vsetky boolovska hodnota;

Req

PC.R

PC.CNT

PC.LD

PC.OE

IR.LD

IR.OE

MAR.LD

MAR.OE

```
                MBR.LD
                MBR.OE
                MBR-MB
                RB-MBR
                ALUPassA
                ALUPassB
                AC.LD
                AC.OE
OPER STAV
                stav      symbol ∈ { RES, IF0, IF1, IF2, IF3, OD,LD0, LD1,
                                     LD2, ST0, ST1, AD0,AD1,AD2,BR0,BR1 };
   proces     PrGlobal  =  Reset. [RiadCyklus]^ω;

agent   RiadCyklus
TE   es, ef → up(CK=1);
KM   (IV=v ; es; u)(u; ef; OV=h);
vs   v prípade  (stav, IV=v)                    h
                  RES,u          PC.OE=1, MAR.LD=1, PC.CNT=1;
                  IF0,u,                          u ;
                  IF1,Wait =0                     u ;
                 IF1,Wait =1       MAR.OE=1, RD/Wr=1, Req=1, MBR.OE=0,
                                       MBR.LD=1, RB-MBR=0 ;
                 IF2,Wait=1        MAR.OE=1, RD/Wr=1, Req=1, MBR.OE=0,
                                       MBR.LD=1, RB-MBR=0 ;
                 IF2,Wait=0,D=d      MBR-MB=1, IR.LD=1 ;
                 IF3,Wait =0         MBR-MB=1, IR.LD=1 ;
                  IF3,Wait =1                    u ;
                    OD,IR.opkód=LD          IR.OE=1, MAR.LD=1 ;
                    OD,IR.opkód=ST     IR.OE=1, MAR.Wr=1, RB-MBR=1,
                                         ALUPassA=1, MBR.LD=1 ;
                    OD,IR.opkód=ADD         IR.OE=1, MAR.LD=1 ;
                    OD,IR.opkód=BRN                 u ;`

                     LD0,Wait=1       MAR.OE=1, RD/Wr=1, Req=1, MBR.OE=0,
                                          MBR.LD=1, RB-MBR=0 ;
                     LD1,Wait=1       MAR.OE=1, RD/Wr=1, Req=1, MBR.OE=0,
                                          MBR.LD=1, RB-MBR=0 ;
                     LD1,Wait=0,D=d     MBR-MB=1, ALUPassB=1,AC.WR=1 ;
                     LD2,u,                         u ;
                    ST0,Wait=1       MAR.OE=1, RD/Wr=0, Req=1,MBR.OE=1 ;
                    ST1,Wait=1       MAR.OE=1, RD/Wr=0, Req=1, MBR.OE=1 ;
                    ST1,Wait=0                     u ;
                   AD0,Wait=1        MAR.OE=1, RD/Wr=1, Req=1,MBR.OE=0,
                                          MBR.LD=1, RB-MBR=0 ;
                   AD1,Wait=1        MAR.OE=1, RD/Wr=1, Req=1, MBR.OE=0,
                                          MBR.LD=1, RB-MBR=0 ;
                   AD1,Wait=0,D=d       MBR-MB=1, AC.LD=1 ;
                   AD2, u,                        u ;
                   BR0, AC <  0           IR.OE=1, PC.LD=1 ;
                   BR0, AC >= 0                   u ;
                   BR1, u,                        u ;
```

```
g       stav :=   v prípade (stav,v)     je
                  RES,u                  IF0;    //zaciatok IstrVyb
                  IF0,u                  IF1;
                  F1, Wait =0            IF1;
                  IF1, Wait =1           IF2;
                  IF2, Wait =1           IF2;
                  IF2, Wait=0            IF3;
                  IF3, Wait =0           IF3
                  IF3, Wait =1           OD;
                  OD,  IR.opkód=LD       LD0;    //začiatok LD
                  OD,  IR.opkód=ST       ST0 ;   //začiatok ST
                  OD,  IR.opkód=ADD      AD0 ;    // začiatok ADD
                  OD   IR.opkód=BRN      BR0 ;    //začiatok BRN
                  LD0, Wait=1            LD1 ;
                  LD1, Wait=1            LD1 ;
                  LD1, Wait=0            LD2 ;
                  LD2,   u               IF0 ;
                  ST0, Wait=1            ST1 ;
                  ST1, Wait=1            ST1 ;
                  ST1, Wait=0            IF0 ;
                  AD0, Wait=1            AD1 ;
                  AD1, Wait=1            AD1 ;
                  AD1, Wait=0            AD2 ;
                  AD2,   u               IF0 ;
                  BR0, AC <  0           BR1 ;
                  BR0, AC >= 0           IF0 ;
                  BR1,   u               IF0 ;
Reset
   TE   es, ef;
   KM    (u;es;u)(u; ef ; PC.R = 1);
   g          stav := RESET;
              PC := 0

START    PrGlobal        ez or up(Res=1);
```

Pretože v kolonke restr nie je žiadny procesný predikát pri objavení sa udalosti up(Res,1) sa teda skončí ten PrGlobal, ktorý bežal a naštartuje sa globálny proces PrGlobal odznova.

Dôležitá poznámka: V špecifikácií agenta **RiadCyklus**, pri výpočte funkcie **vs** je treba vychádzať zo začiatočného stavu (na začiatku komunikácie KM) agenta a zo vstupného vektora IV = **v** (teda z toho, aké je **v** v prvej akcií formuly KM). Hodota **h** výstupného vektora OV, ktorá sa indikuje v koncovej akcií f komunikačnej formuly KM sa teda musí vyjadriť prostredníctvom dvojice (**začiatočný stav**, **v**). V danom prípade špecifikácia vedie fakticky na Moorov FSM a hodnota **h** na koci KM agenta **RiadCyklus** je **jednoznačne daná** iba stavom, avšak na konci konci komunikácie, teda na konci cykla jednocyklového agenta – mikrooperácie **RiadCyklus**.

Funkcia **vs**: **DV*→ DH*** vracia výstupné slovo u h pre vsupné slovo v u, teda

vs(**v**, u) = u **h**

a funkcia **g**(**v**, u) = **q**, kde q je koncový stav agenta RiadCyklus.