PATENTOVÝ SPIS

(11) Číslo dokumentu:

279 873

ČESKÁ REPUBLIKA

(19)

(21) Číslo přihlášky: **932-91**

(22) Přihlášeno: 04. 04. 91

(30) Právo přednosti:
04. 04. 90 US 90/504910

(40) Zveřejněno: 18. 03. 92

(47) Uděleno: 26. 05. 95

(24) Oznámeno udělení ve Věstníku: 12. 07. 95

(13) Druh dokumentu: **B6**

(51) Int. Cl.[6]:
G 06 F 9/00
G 06 F 9/30

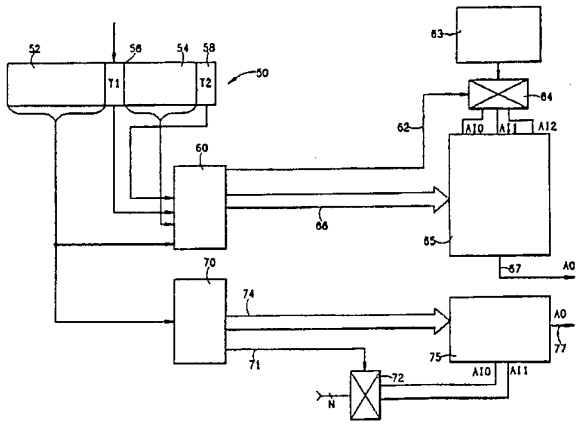
ÚŘAD PRŮMYSLOVÉHO VLASTNICTVÍ

(73) Majitel patentu:
International Business Machines Corporation, Armonk, NY, US;

(72) Původce vynálezu:
Blaner Bartholomew, Newark Valley, NY, US;
Vassiliadis Stamatis, Vestal, NY, US;
Phillips James Edward, Binghamton, NY, US;

(54) Název vynálezu:
Hardwarové zařízení pro překonávání datové závislosti v počítači

(57) Anotace:
Zařízení obsahuje instrukční registr (50) s nejméně dvěma paměťovými poli (52, 54), nejméně tři univerzální registry (63A, 63B, 63C), každý pro jeden operand, prvek (64) vzájemného spojení, který je spojen se společným výstupem univerzálních registrů, dekódovací a řídicí logiku (60), mající své vstupy připojené k výstupu instrukčního registru (50) a mající vyvolovací výstup (62) registrů spojený s prvkem (64) vzájemného spojení a mající svůj vyvolovací výstup (66) funkcí spojený s aritmetickou a logickou jednotkou (65). Aritmetická a logická jednotka (65) je dále opatřena třemi operandovými vstupy (AI0, AI1, AI2) spojenými s prvkem (64) vzájemného spojení a má jeden výstup (67).



CZ 279 873 B6

Hardwarové zařízení pro překonávání datové závislosti v počítači

Oblast techniky

Vynález se týká hardwarového zařízení pro vykonávání skalárních instrukcí v počítači s architekturou pro sériové vykonávání jejich sledu. Konkrétněji se vynález vztahuje na paralelní vykonávání skalárních instrukcí, když jedna z instrukcí vyžaduje jako operand výsledek získávaný při souběžně vykonávané instrukci.

Stav techniky

Zřetězené zpracovávání je standardní postup používaný návrháři počítačů pro zlepšení výkonu počítačových systémů. Při zřetězeném zpracovávání je instrukce dělena do několika kroků nebo stupňů, pro něž je přidělen jediný hardware pro vykonávání instrukce přiřazené tomuto stupni. Rychlost toku instrukcí zřetězením závisí na rychlosti, s níž nové instrukce vstupují do zřetězení, a to spíše, než na délce zřetězení. Při idealizované struktuře zřetězení, kde je maximum jedné instrukce zaváděno do zřetězení během jednoho cyklu, je propustnost zřetězení, která je mírou počtu instrukcí vykonávaných na jednotku času, závislá pouze na době cyklu. Je-li doba cyklu n-stupně implementace zřetězeného zpracovávání předpokládána jako $\frac{m}{n}$, kde je doba cyklu odpovídající implementace nepoužívající postupů zřetězeného zpracovávání, potom je maximální potenciální zdokonalení poskytované zřetězeným zpracováváním rovno n.

I když ze shora uvedeného vyplývá, že zřetězené zpracovávání poskytuje možnost n-násobného zlepšení výkonu počítačového systému, řada praktických omezení mají za následek, že skutečný zisk výkonu je menší, než platí pro ideální případ. Tato omezení vyplývají z existence hazardů při zřetězeném zpracovávání. Hazard ve zřetězeném zpracovávání je definován jakýmkoli aspektem struktury zřetězeného zpracovávání, který brání průchodu strukturou při maximální rychlosti. Hazardy zřetězeného zpracovávání mohou být způsobeny datovými závislostmi, strukturálními konflikty /vyplývajícími z hardwaru)/, řídícími závislostmi a jinými faktory.

Hazardy datové závislosti jsou často nazývány hazardy záznam-čtení /write-read hazards/, protože první instrukce musí zaznamenávat svůj výsledek dříve, než druhá instrukce může číst a následně používat tento výsledek. Aby se dovolil tento záznam před čtením, provádění čtení musí být blokováno do té doby, než došlo k zaznamenání. Toto blokování zavádí cyklus nečinnosti, často nazývaný "bublina" nebo "uváznutí" do výkonu blokované instrukce. Bublina přidává jeden cyklus do celkového prováděcího času uvázlé instrukce a tak snižuje propustnost zřetězeného zpracovávání. Při provádění v hardwaru může zajišťování a rozlišování hazardů strukturální a datové závislosti mít za následek nejen ztráty výkonu vzhledem k nedostatečnému využití hardwaru, ale může se také stát kritickou cestou stroje. Tento hardware bude potom omezovat dosažitelné doby cyklů stroje. Hazardy proto mohou negativně ovlivnit dva faktory, které přispívají k prostupnosti

zřetězení: počet instrukcí vykonaných v jednom cyklu a dobu cyklu stroje.

Existence hazardů ukazuje, že rozvrhování nebo seřazování instrukcí, když vstupují do struktury zřetězení, je velmi důležité při pokusu dosáhnout efektivního využití zřetězeného hardwaru. Efektivní využití hardwaru se potom projevuje ve výkonových ziscích. V podstatě je zřetězené rozvrhování pokus využívat zřetězení na jeho maximální potenciál tím, že se pokoušíme vyhnout se hazardům. Rozvrhování může být dosahováno staticky, dynamicky nebo s kombinací obou postupů. Statické rozvrhování se dosahuje přeskupováním sledu instrukcí před vykonáním na ekvivalentní proud instrukcí, který plněji využije hardware než v dřívějším případě. Jako příklad statického rozvrhování slouží tabulka I a tabulka II, v nichž se vyloučilo vzájemné blokování mezi dvěma instrukcemi zavádění.

Tabulka I

X1	žádná instrukce
X2	žádná instrukce
ADD R4, R2	R4 = R4 + R2
LOAD R1, /Y/	zaveď R1 z paměťového místa Y
LOAD R1, /X [R1]/	zaveď R1 z paměťového místa X do funkce R1
ADD R3, R1	R3 = R3 + R1
LCMP R1, R4	zaveď dvojkový komplement /R4/ do R1
SUB R1, R2	R1 = R1 - R2
COMP R1, R3	srovnej R1 a R3
X3	jakákoli slučovatelná instrukce
X4	jakákoli slučovatelná instrukce

Tabulka II

X1	jakákoli instrukce
X2	jakákoli instrukce
LOAD R1, /Y/	zaveď R1 z paměťového místa Y
ADD R4, R2	R4 = R4 + R2
LOAD R1, /X [R1]/	zaveď R1 z paměťového místa X do funkce R1
ADD R3, R1	R3 = R3 + R1
LCMP R1, R4	zaveď dvojkový komplement /R4/ do R1
SUB R1, R2	R1 = R1 - R2
COMP R1, R3	srovnej R1 a R3
X3	žádná slučovatelná instrukce
X4	žádná slučovatelná instrukce

I když rozvrhovací postupy mohou odstranit některé hazardy a přinést tak zlepšení výkonu, ne všechny hazardy mohou být odstraněny. Byla navržena řešení pro datové závislosti, které nemohou být odstraněny rozvrhováním. Tyto návrhy vykonávají více operací paralelně. Podle jednoho návrhu je proud instrukcí analyzován na základě použití hardwaru a je seskupován do sloučené instrukce určené k tomu, aby byla vydávána jako jediná jednotka. Tento přístup se odlišuje od "superskalárního stroje", v němž je

řada instrukcí seskupována přísně na základě metody "první zařazen, první vybrán" pro současný výstup. Předpokládáme-li, že hardware je určen pro podporování současného výstupu dvou instrukcí, stroj se sloučenými instrukcemi by pároval sled instrukcí z tabulky II následovně: /-X1/ /X2 LOAD/ /ADD LOAD/ /ADDLCMP/ /SUB COMP/ /X3, X4/, čímž se vyloučí datová závislost mezi druhou instrukcí LOAD a druhou instrukcí ADD. Srovnatelný superskalární stroj by však vydal následující dvojice instrukcí: /X1, X2/ /LOAD, ADD/ /LOAS, ADD/ /LCMP SUB/ /COMP X3/ /X4 -/, vytvářející nevýhodu datové závislosti LOAD-ADD.

Druhé řešení výpomoci proti vzájemnému blokování datovou závislostí bylo navrženo v Computer Architectural News, March 1988, v článku nazvaném "The WM Computer Architecture", W. A. Wulfa. Tento článek WM Computer Architecture navrhuje:

1. sestavení souboru instrukcí, který vkládá více než jednu operaci do jedné instrukce
2. připuštění vzájemného blokování registrů v sestavené instrukci a
3. zřetězení dvou aritmeticko-logických jednotek podle obr. 1 pro překonání /collapsing/ vzájemného blokování v rámci jedné instrukce.

Ve Wulfově návrhu musí být samozřejmě sestaveny jiné instrukce pro všechny sledové páry instrukcí, jejichž vzájemná blokování se mají potlačit. To má za následek buď to, že se definuje pro nový soubor instrukcí prohibitivní počet operačních kódů, anebo že se počet operačních sledů, jejichž vzájemná blokování se mají překonat, opatří mezí, vymezenou počtem operačních kódů, které jsou k dispozici. Kromě toho toto schéma nemusí být cílový kód slučitelný s dřívějšími implementacemi architektury. Jiné nedostatky tohoto schématu zahrnují požadavek na dvě aritmeticko-logické jednotky, jejichž zřetězení může mít za následek vykonávání násobných instrukcí, vyžadujících přibližně dvojnásobek času na vykonávání jediné instrukce. Takový vzrůst vykonávacího času by se projevil ve vzrůstu doby cyklu stroje a nepotřebně by znevýhodnil všechny vykonávání instrukcí.

V případě, kde byl existující stroj postaven pro postupné vydávání a vykonávání daného souboru instrukcí, bylo by prospěšné využít paralelnosti ve vydávání a vykonávání instrukcí. Paralelní vydávání a vykonávání daného souboru instrukcí, bylo by prospěšné využít paralelnosti ve vydávání a vykonávání instrukcí. Paralelní vydávání a vykonávání instrukcí by zvýšilo propustnost stroje. Dále je možné prospěšnost takové paralelnosti maximalizovat minimalizováním čekací doby na vykonávání instrukcí vyplývající z hazardů datové závislosti ve zřetězeném zpracovávání instrukcí. Adaptace na paralelnost by tak umožnila redukovat takové čekací doby překonáním vzájemného blokování vyplývajícího z těchto hazardů. Tyto kladné znaky je však třeba mít možnost využívat, aniž by bylo nutné platit cenu změny architektury v existujících strojích, vytváření nového souboru instrukcí pro vytvoření všech možných instrukčních dvojic a jejich kombinací majících vzájemné blokování a přidání značné části hardwaru. Dále by adaptace měla vykazovat jen velmi malý nebo vůbec žádný dopad na dobu cyklu stroje.

Podstata vynálezu

Uvedené cíle dosahuje vynález hardwarového zařízení pro překonávání datové závislosti v počítači s architekturou pro sériové vykonávání sledu skalárních instrukcí, jehož podstatou je, že obsahuje instrukční registr s nejméně dvěma paměťovými poli, nejméně tři univerzální registry, každý pro jeden operační prvek příčného spojení, který je spojen se společným výstupem univerzálních registrů, dekódovací a řídicí logiku mající své vstupy připojené k výstupu instrukčního registru a mající vyvolovací výstup registrů spojený s prvkem vzájemného spojení a mající svůj vyvolávací výstup funkcí spojený s aritmetickou a logickou jednotkou, která je dále opatřena třemi operandovými vstupy spojenými s uvedeným prvkem vzájemného spojení a má jeden výstup.

Zařízení podle vynálezu, uspořádané uvedeným způsobem, zajišťuje při současném vykonávání více skalárních instrukcí možnost přijímání více skalárních instrukcí pro souběžné vykonávání instrukcí, přičemž druhá ze skalárních instrukcí používá jako operand výsledek získaný vykonáním první skalární instrukce. Zařízení je uzpůsobeno pro přijímání tří operandů, které jsou používány první a druhou skalární instrukcí a má řídicí složku připojenou k opatření pro přijímání instrukcí, které generují řídicí signály, indikující operace, které vykonávají více skalárních instrukcí a které indikují pořadí jejich vykonávání. Vícefunkční aritmeticko-logická jednotka je připojena k operandům a řídicím opatřením a reaguje na řídicí signály a na operandy tím, že vytváří, paralelně s vykonáváním první instrukce, jediný výsledek odpovídající vykonání druhé instrukce.

Z jiného hlediska vytváří vynález zařízení, které podporuje současné vykonávání více skalárních instrukcí, kde výsledek získaný první ze současně prováděných instrukcí je použit jako operand ve druhé ze současně prováděných instrukcích. Zařízení vykonává druhou instrukci paralelně s vykonáváním první instrukce tím, že obsahuje aritmeticko-logickou jednotku překonávající datovou závislost, která je uzpůsobena pro přijímání tří operandů, které jsou používány první a druhou instrukcí pro poskytnutí výsledku druhé instrukce současně s výsledkem první instrukce.

Vynález proto přináší stroj, který usnadňuje paralelní provádění instrukcí pro zvýšení výkonu stávajícího počítače. Významnou výhodou zařízení podle vynálezu je redukce čekací doby na vykonávání instrukcí, vyplývající z hazardů datové závislosti, existujících ve vykonávaných instrukcích. Zařízení podle vynálezu umožňuje překonat vzájemná blokování vyplývající z hazardů datové závislosti, existujících mezi instrukcemi vykonávanými paralelně. Tyto účinky a výhody jsou dosahovány se zdokonalením výkonu a provádění instrukcí, jaké z toho vyplývají, pomocí zařízení slučitelného se skalárním počítačem určeným pro postupné vykonávání instrukcí.

Podle výhodného provedení vynálezu aritmetická a logická jednotka obsahuje sčítačku opatřenou třemi operandovými vstupy, připojenou k operandovým vstupům aritmetické a logické jednotky a jedním výsledkovým výstupem, spojeným s výstupem aritmetické a logické jednotky. Sčítačka s výhodou zahrnuje sčítačku s uchovávanými přenosy opatřenou třemi vstupy spojenými s operandovými

vstupy a se součtovým výstupem a přenosovým výstupem, a dále sčítačku s predikcí přenosů, opatřenou dvěma vstupy připojenými k příslušnému součtovému výstupu a přenosovému výstupu sčítačky a uchováváním přenosů a s výstupem spojeným s výsledkovým výstupem sčítačky.

Podle dalšího znaku vynálezu jsou dva ze tří vstupů aritmetické a logické jednotky připojeny ke třem logickým funkčním obvodům, které jsou dále spojeny s jedním z uvedených tří operandových vstupů sčítačky. Jeden z uvedených tří vstupů aritmetické a logické jednotky a výsledkový výstup sčítačky mohou být dále připojeny ke třem logickým funkčním obvodům, které jsou spojeny s výstupem aritmetické a logické jednotky.

První logické obvody na vstupní a výstupní straně sčítačky mohou být součinové obvody, druhé logické obvody mohou být součtové obvody a třetí logické obvody mohou být obvody výlučného součtu. Dva ze tří vstupů aritmetické a logické jednotky mohou být připojeny k multiplexoru, který může být dále připojen ke třem logickým funkčním obvodům. Tři logické funkční obvody mohou být připojeny k multiplexoru, který může být dále spojen s jedním ze tří operandových vstupů sčítačky.

Podle dalšího znaku vynálezu mohou být tři logické funkční obvody spojeny s multiplexorem, který může být dále spojen s výstupem aritmetické a logické jednotky.

Prvek vzájemného spojení může konečně obsahovat multiplexor pro zavádění obsahů tří univerzálních registrů na tři operandové vstupy aritmetické a logické jednotky.

Přehled obrázků na výkresech

Vynález je blíže vysvětlen v následujícím popise na příkladech provedení s odvoláním na připojené výkresy, kde na obr. 1 je stavba podle známého stavu techniky pro vykonávání instrukce párující operace, na obr. 2 je soubor časových sledů ilustrujících zřetězené vykonávání skalárních instrukcí, na obr. 3 je blokové schéma sčítačky, která přijímá až tři operandy a vytváří jediný výsledek, na obr. 4A, 4B je kategorizace instrukcí vykonávaných existujícím skalárním strojem, na obr. 5 je tabulka funkcí vytvořených v případech vzájemného blikování, kde instrukce jsou kombinovány typu "logical" a typu "add" v kategorii 1 z obr. 4A, na obr. 6A a 6B je specifikace operací, které mají být prováděny na operandech aritmeticko-logickou jednotkou podle vynálezu pro podporování instrukcí ve slučovatelných kategoriích na obr. 4A a 4B, na obr. 7A a 7B je shrnutí směrování operandů do aritmeticko-logické jednotky definované na obr. 6A a 6B, na obr. 8 je blokové schéma ukazující, jak je vynález používán pro vykonávání paralelního provádění dvou navzájem blikovaných instrukcí, na obr. 9 je blokové schéma vícefunkční aritmeticko-logické jednotky definované obr. 6A, 6B, 7A a 7B, na obr. 10 je tabulka schematicky ilustrující funkce vyžadující implementaci pro porušení vzájemných blokování vyplývajících z hazardů vyskytujících se při generování adres, na obr. 11 je logický diagram, ilustrující vícefunkční aritmeticko-logickou jednotku z obr. 10, na obr. 12 je schéma uspořádání funkcí podporovaných aritmeticko-logickou jednotkou pro překonání vzájemných blokování ve sloučených

instrukcích větvení, na obr. 13 je logický diagram znázorňující aritmeticko-logickou jednotku podle obr. 12 a na obr. 14 je schéma uspořádání sčítačky potřebné pro překonávání vzájemných blokování pro instrukce zahrnující devět operandů.

Provedení vynálezu

V následujícím popise je pojem "strojový cyklus" používán pro kroky zpracovávání zřetězením, potřebné pro vykonávání instrukce. Strojový cyklus zahrnuje jednotlivé intervaly, které odpovídají stupňům zpracování zřetězením. "Skalární instrukce" je instrukce, která je vykonávána při použití skalárních operandů. Skalární operandy jsou operandy reprezentující jednohodnotové veličiny. Pojem "slučování" se vztahuje na seskupování instrukcí obsažených ve sledu instrukcí, přičemž toto seskupování se provádí za účelem souběžného nebo paralelního vykonávání seskupených instrukcí. V minimálním případě je slučování reprezentováno "párováním" dvou instrukcí pro současné vykonávání. V popisovaném vynálezu jsou sloučené instrukce nezměněny vzhledem ke tvarům, které mají, nejsou předkládány pro vykonávání skalárních operací. Jak je vysvětleno níže, jsou slučované instrukce doprovázeny "příznaky", tj. bity připojenými k seskupeným instrukcím, které označují seskupování instrukcí pro paralelní vykonávání. Bity tak ukazují začátek a konec sloučené instrukce.

V dalším popise bude popsáno řešení hardwaru pro poskytnutí pomoci proti vykonávání vzájemných blokování jednotek, která nemohou být překonána při použití postupů známého podle stavu techniky. Cílem je minimalizovat hardware potřebný pro překonání těchto vzájemných blokování a vyvolat přitom jen malou nebo vůbec žádnou zátěž na dobu cyklu z přidaného hardwaru. Žádné změny architektury nejsou požadovány pro implementaci tohoto řešení. Slučitelnost cílového kódu pro existující architekturu je proto dodržována.

Příkladem existující architektury je sekvenční skalární stroj, jako je System/370, dodávaný společností International Business Machines Corporation, tj. přihlašovatelem vynálezu. Z tohoto hlediska takový systém může zahrnovat stavbu typu "System/370 extended architecture" /370-XA/ a "System/370 enterprise systems architecture" /370-ESA/. Je zde možné se odvolat na publikaci "principples of Operation of the IBM System/370" č. GA22-7000-10, 1987 a "the Principples of Operation, IBM Enterprise Systems Architecture/370", č. SA22-7200-0, 1988.

Instrukční soubor pro tyto existující skalární stavby systému System/370 je dobře znám. Tyto instrukce jsou skalární instrukce, přičemž jsou vykonávány operacemi prováděnými na skalárních operandech. Odvolávky, činěné níže na obzvláštní instrukce v souboru instrukcí vykonávaných shora uvedenými stroji jsou uváděny v obvyklé symbolické formě.

Předpokládejme, že se má vykonat následující sled instrukcí pomocí superskalárního stroje schopného vykonávat čtyři instrukce na cyklus:

Tabulka III

```
/1/ LOAD    R1, X  zaveď obsah X do R1
/2/ ADD     R1, R2 přičti R1 k R2 a vlož výsledek do R1
/3/ SUB     R1, R3 odečti R3 od R1 a vlož výsledek do R1
/4/ STORE   R1, Y  ulož výsledek do paměťového místa Y
```

Přes schopnost vícenásobného provádění instrukcí během cyklu bude superskalární stroj vykonávat shora uvedený sled sériově vzhledem ke vzájemnému blokování instrukcí. Předpokládá se, na základě analýzy sledovacích programů, že vzájemná blokování nastávají po přibližně jednu třetinu doby. Tolik superskalárních zdrojů by bylo vyplýtváno, což by vedlo k degradaci výkonu superskalárního stroje. Chování superskalárního stroje při vzájemně blokovaných skalárních instrukcích je znázorněno časovým sledem 8 na obr. 2. Na tomto obrázku se struktura zřetězení pro instrukce z tabulky III předpokládá následovná:

```
          /1/ LOAD:ID AG CA PA
/2/ a /3/ ADD and SUBTRACT:ID EX PA,
```

kde ID dekódování a přístup k registru, AG je generování adresy operandu, CA představuje přístup k rychlé vyrovnávací paměti, EX představuje vykonej a PA /put away/ představuje zaznamenání výsledku do registru. Pro zjednodušení výkladu se předpokládá u všech příkladů uváděných v tomto popise, pokud to není výslovně uvedeno, že se neimplementuje obcházení. V superskalárním stroji se vykonávání toku instrukcí provádí sériově vzhledem k tomu, že vzájemná blokování instrukcí snižují výkon superskalárního stroje na výkon skalárního stroje.

Na obr. 2 nevyžadují instrukce /2/ a /3/ žádné generování adresy /AG/. Tento stupeň však musí být ve zřetězení zohledněn. Z toho vyplývají neoznačené intervaly 7 a 9. Tato konvence se také týká ostatních tří sledů na obr. 2.

Shora uvedený příklad demonstruje, že vzájemná blokování instrukcí mohou omezit paralelnost, jaká je k dispozici na instrukční úrovni pro využití superskalárního stroje. Výkon je možné získat zřetězováním a obcházením výsledků jedné vzájemně blokované instrukci ke druhé. Vykonávání vzájemně blokovaných instrukcí však musí být nicméně prováděno sériově.

Má-li se vyloučit ztráta vykonávacích cyklů v důsledku vzájemného blokování, musí být vzájemné blokované instrukce prováděny "paralelně" a považovány za jedinou instrukci. To vede ke koncepci slučované vzájemně blokované instrukce, tj. souboru skalárních instrukcí, se kterými se má zacházet jako s jedinou instrukcí přes výskyt vzájemných blokování. Žádoucím znakem hardwaru vykonávajícího sloučenou instrukci je, že její vykonávání nevyžaduje více cyklů, než jakých je zapotřebí jednou ze slučovaných instrukcí. Jako důsledek slučování instrukcí a jeho žádoucích znaků musí stroj na soubor slučovaných instrukcí sledovat skalární instrukce užíváním hardwaru spíše než na základě popisu operačních kódů.

Myšlenka slučovaných vzájemně blokovaných instrukcí může být objasněna použitím instrukcí ADD a SUB v tabulce III. Tyto dvě instrukce mohou být považovány za typ jediné instrukce, protože používají stejný hardware. V důsledku toho jsou kombinovány a vykonávány jako jedna instrukce. Pro využití paralelnosti jejich vykonání vyžaduje provedení

R1 = R1 + R2 - R3

v jednom cyklu spíše než vykonávání sledu:

R1 = R1 + R2
R1 = R1 - R3

který vyžaduje vykonat více než jeden cyklus. Vzájemné blokování může být eliminováno, protože sčítání a odečítání používá totožný hardware. Kromě toho může být kombinovaná instrukce R1 + R2 - R3 vykonána při použití aritmeticko-logické jednotky obsahující sčítačku s uchovávanými přenosy CSA (carry save adder) a sčítačku s predikcí přenosů CLA (carry look-ahead adder), jak je znázorněno na obr. 3, za podmínky, že aritmeticko-logická jednotka ALU byla navržena pro vykonávání sčítací/odečítací funkce tři na jedna.

Jak by mělo být zřejmé, odpovídá kombinovaný tvar (R+ + R2 - R3) přepsání dvou operandů druhé instrukce v podmínkách tří operandů, z čehož vyplývá požadavek sčítačky, která může vykonat druhou instrukci na základě tří operandů. Na obr. 3 je znázorněna sčítačka s uchovávanými přenosy (CSA) jako sčítačka 10. Sčítačka 10 CSA (carry save adder) je běžná ze všech hledisek a přijímá tři operandy pro vytvoření dvou výsledků, a to součtu (S) na součtovém výstupu 12 a přenosu (C) na přenosovém výstupu 14. Ve shora uvedeném příkladě jsou vstupy do sčítačky 10 CSA operandy obsažené ve třech registrech R1, R2 a R3 (komplementované). Výstupy sčítačky 10 CSA jsou vynášeny v prvcích 16 a 17 pro dodání začáteční "1" nebo "0" ("horká" 1 nebo 0) na hodnotu přenosu přes vstup 20. Hodnota na vstupu 20 je nastavena obvykle podle funkce, která má být vykonávána sčítačkou 10 CSA.

Výstupy sčítačky 10 CSA s uchovávanými přenosy (s připsanou jedničkou nebo nulou) jsou upraveny jako dva vstupy do sčítačky 22 CIA s predikcí přenosů. Sčítačka 22 CLA také obvykle dostává "horkou" jedničku nebo nulu na vstupu 24 podle požadované operace a vytváří výsledek na výstupu 26. Na obr. 3 je výsledek vytvořený sčítačkou 22 CLA kombinací obsahů tří registrů R2, R2 a R3 (komplementovaných).

Sčítačky s uchovávanými přenosy (carry save adder CSA) a sčítačky s predikcí přenosů (carry lock-shead adder CLA) jsou běžné součástky, jejichž struktury a funkce jsou dobře známé. Hwang popisuje ve svých "Computer Arithmetic: Principples, Architecture and Design, 1979" sčítačky s predikcí přenosů na str. 88 až 93 a sčítačky s uchovávanými přenosy na str. 97 až 100.

Přestože sčítání tři na jedna vyžaduje přídavný stupeň, takový stupeň nemá negativní vliv na dobu cyklu stroje, protože délka jiných drah obvykle přesahuje délku dráhy aritmeticko-logické jednotky. Tyto kritické cesty lze obvykle nalézt na dra-

hách majících přístup polí, a generování adres vyžadující jednotku ALU tři na jedna a přechod čipu. Zpoždění vyplývající z přídavného stupně není proto na závadu a navržené schéma bude mít za následek zlepšení výkonu při srovnání se skalárními a superskalárními stroji. Zlepšení výkonu je znázorněno na obr. 2 souborem zřetězených schematických grafických čar označeným sledem 26. Tyto schematické grafické čáry ukazují provádění uvažovaného sledu instrukcí pomocí stroje se souborem slučovaných instrukcí, který obsahuje aritmeticko-logickou jednotku ALU se sčítačkou upravenou podle obr. 3.

Jak je znázorněno časovými sledy 8 a 26 na obr. 2, vykonání sledu strojem se souborem sloučených instrukcí vyžaduje osm cyklů nebo dva cykly na instrukci (CPI) při srovnání s jedenácti cykly nebo 2,75 CPI dosažitelnými pomocí skalárních a superskalárních strojů. Pokud se předpokládá, že ve všech strojích je možné zabezpečit obcházení, ukazují čárkové grafy sledů 28 a 30 z obr. 2 popis vykonávání instrukcí, dosažitelného se skalárními nebo superskalárními stroji. Z těchto souborů je patrné, že superskalární stroj vyžaduje osm cyklů nebo 2 CPI pro vykonání příkladového kódu, zatímco stroj se souborem slučovaných instrukcí vyžaduje šest cyklů nebo 1,5 CPI. Tato výhoda stroje se slučováním vůči jak superskalárním, tak i skalárním strojům je patrná spolu s malou výhodností superskalárního stroje vůči skalárnímu stroji pro uvažovaný instrukční sled.

Slučování instrukcí s jejich současným vykonáváním hardwarem není omezeno na aritmetické operace. Například většina logických operací může být slučována způsobem analogickým s aritmetickými operacemi. Slučování některých instrukcí by však mohlo mít za následek protahování doby cyklu, protože musí být pro vykonávání slučované funkce způsobována nepřijatelná zpoždění. Například sloučená instrukce ADD-SHIFT může protáhnout dobu cyklu nežádoucím způsobem, což by pokazilo veškerý výkonový zisk. Častost vzájemného blokování mezi těmito instrukcemi je však nízká vzhledem k nízké častosti výskytu posouvacích instrukcí, a proto mohou být vykonávány sériově bez podstatné ztráty výkonu.

Jak bylo popsáno, shora vyskytující se datové hazardy vzájemného blokování, když je zaznamenáván registr nebo paměťové místo a potom čten při následující instrukci. Navržená zařízení podle vynálezu překonávají tato vzájemná blokování odvozováním nových funkcí, které vznikají z kombinování provádění instrukcí, jejichž operandy přinášejí datové hazardy, při současném dodržování provádění funkcí vlastních souborů instrukcí. I když u některých kombinací instrukcí a operandů se neočekává jejich výskyt ve funkčním programu, jsou zvažovány všechny kombinace. Všeobecně všechny funkce odvozené ze shora uvedené analýzy jakož i funkce vyplývající ze skalární implementace souboru instrukcí by mohly být vykonány. V praxi se však vyskytují určité funkce, jejichž implementace se dobře nehodí pro schéma navržené pro toto zařízení. Následující výklad objasňuje tyto koncepce probráním toho, jak tyto nové funkce vznikají z kombinování provádění dvou instrukcí. Příklady sledů instrukcí, které se dají dobře zpracovávat podle vynálezu, jsou uváděny spolu s některými sledy, které nejsou dobře zpracovávány. Je také znázorněno logické schéma výhodného provedení vynálezu.

Zařízení podle vynálezu je navrženo pro usnadnění paralelního vydávání a vykonávání instrukcí. Příklad paralelního vydávání instrukcí je možné nalézt v superskalárním stroji podle známého stavu techniky. Vynález přitom umožňuje usnadnit paralelní vydávání instrukcí, které obsahují vzájemné blokování. Použití hardwaru podle vynálezu pro překonání datové závislosti však není omezeno na žádnou obzvláštní architekturu vydávání a vykonávání, ale má všeobecnou použitelnost pro schémata, která vydávají vícenásobné instrukce během cyklu.

Pro poskytnutí hardwarové základny pro tuto diskusi se předpokládá architektura s instrukční úrovní systému System/370, v níž se mohou vydávat až dvě instrukce na cyklus. Použití systému System/370, ani na dvoucestnou paralelnost. Diskuse je rozdělena do sekcí pro pokrytí operací aritmeticko-logické jednotky, generování paměťových adres a určování větvení.

Všeobecně může být soubor instrukcí systému System/370 rozdělen do kategorií instrukcí, které mohou být vykonávány paralelně. Instrukce v rámci těchto kategorií mohou být slučovány pro vytváření sloučených instrukcí. Níže popsané zařízení podle vynálezu podporuje provádění sloučených instrukcí paralelně a zajišťuje, že vzájemná blokování existující mezi členy sloučené instrukce budou pokryta, zatímco se instrukce současně vykonávají. Například může být architektura Systemu/370 rozdělena do kategorií znázorněných na obr. 4A a 4B.

Základní princip pro tuto kategorizaci byl založen na funkčních požadavcích instrukcí systému System/370 a jejich hardwarového použití. Zbytek instrukcí systému System/370 není uvažován pro to, aby byly slučovány pro vykonávání v této diskusi. Toto však nebrání tomu, aby byly sloučeny na budoucím stroji pro vykonávání slučovaných instrukcí a pro možné použití těchto závěrů pro "vyloučení výskytu" vzájemných zablokování, jak je vysvětlován v tomto spise.

Uvažujme nyní instrukce obsažené v kategorii 1 slučované s instrukcemi ze stejné kategorie, jak je příkladně uvedeno v následujícím sledu instrukcí:

```
AR R1, R2
SR R3, R4
```

Tento sled, který je prostý datových hazardů vzájemného blokování, přináší výsledky:

```
R1 = R1 + R2
R3 = R3 + R4
```

které zahrnují dvě nezávislé instrukce specifikované architekturou instrukční úrovně systému 370. Provádění takového sledu by vyžadovalo dvě nezávislé a paralelní aritmeticko-logické jednotky dvě na jedna, navržené pro architekturu instrukční úrovně. Tyto výsledky mohou být generalizovány do všech dvojic sledu instrukcí, které jsou prosté datových hazardů vzájemného blokování, v nichž obě instrukce specifikují operaci aritmeticko-logické jednotky. Dvě aritmeticko-logické jednotky jsou dostačující k vykonávání instrukcí vydávaných ve dvojicích, protože každá

instrukce specifikuje nanejvýše jednu operaci aritmeticko-logické jednotky.

Mnoho sledů instrukcí však není prosté vzájemného blokování s datovými hazardy. Tato vzájemná blokování s datovými hazardy vedou k "bublinám" zřetězení, které degradují výkon typického zřetězeného řešení. Řešením pro zvýšení výkonu procesoru je eliminovat tyto "bubliny" ze zřetězení použitím jediné aritmeticko-logické jednotky, která může pokrýt vzájemná blokování s datovými hazardy. Pro vyloučení těchto vzájemných blokování musí aritmeticko-logická jednotka vykonat několik funkcí vznikajících z párování instrukcí a z konfliktů operandů. Funkce, které vznikají, závisí na specifikovaných operacích aritmeticko-logické jednotky, sledu těchto operací a "konfliktů" operandů mezi operacemi /význam pojmu konflikty operandů budou patrné z následující diskuze/. Všechny sledy instrukcí, které mohou být vytvořeny párováním instrukcí, které jsou obsaženy uvnitř shora uvedeného slučovatelného seznamu a které budou specifikovat operaci aritmeticko-logické jednotky, musí být analyzovány pro všechny možné konflikty operandů.

Všeobecné zásady překonávání vzájemného blokování podle vynálezu byly uvedeny shora. Následující popis poskytuje konkrétnější příklad analýz, které je třeba vykonat pro určení požadavků aritmeticko-logické jednotky překonávající vzájemné blokování. Předpokládejme existenci sčítačky tři na jedna, jak je popsána shora, s odvoláním na obr. 3. OP1 a OP2 představují odpovídající první a druhou ze dvou operací, které se mají vykonávat. Například pro následující sled instrukcí

```
NR R1, R2
AR R3, R4
```

OP1 odpovídá operaci NR, zatímco OP2 odpovídá operaci AR /viz níže pro popis těchto operací/. Předpokládejme, že AI0, AI1 a AI2 představují vstupy odpovídající /R1/, /R2/ a /R3/ odpovídajícím vstupům sčítačky tři na jedna z obr. 3. Uvažujme analýzu slučování souboru instrukcí /NR, OR, XR, AR, ALR, SLR, SR/, podsoubory kategorie 1, jak je definovaná na obr. 4A a 4B.

Operace tohoto souboru instrukcí jsou specifikovány:

NR	po slabikách logická AND reprezentovaná $\wedge$
OR	po slabikách logická OR reprezentovaná $\vee$
XR	po slabikách EXCLUSIVE OR reprezentovaná $\oplus$
AR	32 bitové sčítání označené znaménkem, reprezentovaná +
ALR	32 bitové sčítání neoznačené znaménkem, reprezentované +
SR	32 bitové odečítání označené znaménkem, reprezentované --
SLR	32 bitové odečítání neoznačené znaménkem, reprezentované --

Tento soubor instrukcí může být rozdělen do dvou souborů instrukcí pro další úvahy. První soubor by zahrnoval logické instrukce NR, OR a XR, a druhý soubor by zahrnoval aritmetické instrukce AR, ALR, SR a SLR. Seskupování aritmetických instrukcí může být odůvodněno následovně. AR a ALR mohou být obě považovány za implicitní přičítání 33 bitového dvojkového komplementu při použití rozšiřování znaménka pro AR a rozšiřování nuly pro ALR a při přivedení "horké" nuly do sčítačky. I když nastavení stavo-

vého kódu a přeplnění jsou jedinečná pro každou instrukci, operace vykonávána sčítačkou, tj. binární sčítání, je společná pro obě instrukce. Podobně SR a SLR mohou být považovány za implicitní sčítání 33 bitového dvojkového doplňku při použití rozšiřování znaménka pro SR a rozšiřování nuly pro SRL, invertování menšitele a při přivedení "horké 1" do sčítačky. Invertování menšitele je považováno jako úkon vně sčítačky. Protože uvedené čtyři aritmetické operace v podstatě vykonávají tutéž operaci, tj. binární přičítání, budou označovány jako instrukce typu ADD /sčítání/, zatímco logické operace budou označovány jako instrukce typu LOGICAL /logické/.

Jako výsledek redukce shora uvedeného souboru instrukcí na dvě operace je třeba uvažovat následující sledy operací pro analyzování tohoto souboru instrukcí:

LOGICAL	následovaná ADD
ADD	následovaná LOGICAL
LOGICAL	následovaná LOGICAL
ADD	následovaná ADD

Pro každý z těchto sledů je nutno uvažovat všechny kombinace registrů. Kombinace zahrnují případy, kdy všechny čtyři registry jsou rozdílné, plus počet cest ven ze čtyřech možných specifikací registrů: 1/ dva jsou stejné, 3/ tři jsou stejné, a 3/ čtyři jsou stejné. Počet kombinací proto může být vyjádřen takto:

$$\text{Počet kombinací} = 1 + \sum_{i=2}^{4} {}_4C_i,$$

kde ${}_nC_r$ reprezentuje n kombinováno s r. Přitom

$${}_nC_r = n!/ ((n-r) ! r !),$$

přičemž z tohoto vzorce je možno odvodit počet kombinací jako 12. Těchto 12 kombinací registrů je:

1. R1 ≠ R2 ≠ R3 ≠ R4
2. R1 = R2 ≠ R3 ≠ R4
3. R2 = R3 ≠ R1 ≠ R4
4. R2 = R4 ≠ R1 ≠ R3
5. R3 = R4 ≠ R1 ≠ R2
6. R2 = R3 ≠ R4 ≠ R1
7. R1 = R3 ≠ R2 ≠ R4
8. R1 = R4 ≠ R2 ≠ R3
9. R1 = R2 = R3 ≠ R4
10. R1 = R2 = R4 ≠ R3
11. R1 = R3 = R4 ≠ R2
12. R1 = R2 = R3 = R4

Z těchto kombinací pouze sedm z dvanácti dává vznik vzájemnému blokování s datovou závislostí. Funkce vytvořené shora uvedenými případy vzájemného blokování pro shora uvedené sledy LOGICAL-ADD jsou uvedeny na obr. 5. Na tomto obrázku jsou operace typu LOGICAL označeny φ a operace typu ADD jsou označeny ξ .

Zatímco obr. 5 udává operace, které musí být vykonány na operandech instrukcí typu ADD a typu LOGICAL pro překonání vzájemného blokování,obr. 6A a 6B udávají operace aritmeticko-logické jednotky, které je třeba vykonat na vstupech AIO, AI1 a AI2 pro podporování všech instrukcí, které jsou obsaženy ve slučovatelných kategorií z obr. 4A a 4B. Na obr. 6A a 6B jednočlen - označuje dvojkový komplement a /x/ označuje absolutní hodnotu x. Tento obrázek byl odvozen při použití analýzy totožné s tou, jaká byla udána, přičemž však byla uvažována všechna slučování kategorií. Pro operace z obr. 5, které mají být vykonány aritmeticko-logickou jednotkou, musí jednotková řízení vykonávání operací směrovat požadované obsahy registrů na odpovídající vstupy aritmeticko-logické jednotky. Obr. 7A a 7B shrnují směřování operandů, které se musí objevit pro aritmeticko-logickou jednotku definovanou podle obr. 6A a 6B pro vykonávání operací z obr. 5. Spolu s těmito směrováními jsou udávány instrukce typu LOGICAL a ADD pro usnadnění mapování těchto výsledků na obr. 6A a 6B. Směrování sloučených operací ADD-ADD nebyla zahrnuta, protože tyto operace vyžadují čtyřvstupovou aritmeticko-logickou jednotku /viz "idiosynkrasie"/ a jsou tak vzaty na zřetel.

I když popis byl až dosud zaměřen na uvažování analýzy slučovaných instrukcí na čtyřech konkrétních uvedených registrech R1, R2, R3 a R4, je zřejmé, že praktické použití vynálezu není omezeno na čtyři určité registry. Volba těchto označení je spíše pouze pomůckou pro analýzu a porozumění. Naopak by mělo být zřejmé, že analýza může být zevšeobecněna, jak vyplývá ze shora uvedených rovnic.

Logické blokové schéma znázorňující zařízení pro implementaci vícefunkční aritmeticko-logické jednotky popsané v zásadě na obr. 5, 6A, 6B, 7A a 7B, je na obr. 8. Na obr. 8 dostává instrukční registr 50 sloučenou instrukci zahrnující instrukce paměťových polí 52 a 54. Sloučené instrukce jsou opatřeny příznaky 56 a 58. Instrukce a jejich příznaky jsou vedeny do dekódovací a řídicí logiky 60, která dekóduje instrukce a informaci obsaženou v jejich příznacích pro vytvoření signálů volby registrů na vyvolávacím výstupu 62 a signály volby funkce na vyvolávacím výstupu 66. Signály volby registru na vyvolovacím výstupu 62 konfigurují prvek 64 vzájemného spojení, který je připojen k nejméně třem univerzálním registrům 63A, 63B, 63C souboru 63 univerzálních registrů pro dodávání obsahů až tří registrů na tři operandové vstupy AIO, AI1 a AI2 aritmeticko-logické jednotky 65 pro překonávání datové závislosti. Aritmeticko-logické jednotky 65 pro překonávání datové závislosti. Aritmeticko-logická jednotka 65 je vícefunkční aritmeticko-logická jednotka, jejíž funkčnost se volí signály volby funkce přítomnými na výstupu 66 dekódovací a řídicí logiky 60. S operandy dodávanými z registrů připojených přes prvek 64 vzájemného spojení bude aritmeticko-logická jednotka 65 vykonávat funkce udávané signály volby funkce pro dosažení výsledku na výstupu 67.

Paralelně se shora uvedeným zařízením s aritmetickou a logickou jednotkou pracuje druhé zařízení s aritmeticko-logickou jednotkou, obsahující dekódovací a řídicí logiku 70, které dekóduje první instrukci v instrukčním paměťovém poli 52 pro poskytování signálů volby registru přes výstup 710 do obvyklého prvku 720 vzájemného spojení, který je také připojen k univerzálním

registrům 63A, 63B, 63C. Logika 70 také poskytuje signály volby funkce na výstupu 74 do běžné dvouoperandové aritmeticko-logické jednotky 65 . Jak je popisováno níže, aritmeticko-logická jednotka 65 může vykonávat druhou instrukci v obou případech, tj. kdy jeden z jejích operandů závisí nebo nezávisí na výsledných datech vytvářených vykonáváním první instrukce. Obě aritmeticko-logické jednotky pracují paralelně pro zajištění souběžného vykonávání dvou instrukcí, slučovaných nebo nikoliv.

Pokud jde o slučované instrukce paměťových polí 52 a 54 a instrukční registr 50, předpokládá se existence slučovače. Předpokládá se, že slučovač páruje nebo slučuje instrukce z toku instrukcí zahrnujícího sled skalárních instrukcí přiváděných do skalárního počítače, v němž je slučovač umístěn. Slučovač seskupuje instrukce podle shora uvedených zásad. Například instrukce kategorie 1 jsou seskupovány do LOGICAL/ADD, ADD/LOGICAL, LOGICAL/LOGICAL a ADD/ADD dvojic v souladu s tabulkou 5. Ke každé instrukci slučovaného souboru je připojen příznak obsahující řídicí informaci. Tento příznak obsahuje slučovací bity, které se týkají části příznaku použité specificky pro identifikování skupin slučovaných instrukcí. S výhodou v případě slučování dvou instrukcí se použije následující postup pro indikování, kde dochází ke slučování.

Ve strojích systému/370 jsou všechny instrukce uspořádány na půlslovové mezi a jejich délky jsou buď 2,4, nebo 6 bitů. V tomto případě je slučovací příznak zapotřebí pro každé půlslovo. Jednobitový příznak je dostatečný pro indikování, zda instrukce je nebo není slučována. S výhodou "1" značí, že instrukce, která začíná v uvažované slabice, je slučována s následující instrukcí. "0" označuje, že nedochází k žádnému slučování. Slučovací bit sdružený s půlslovy, které neobsahují první byte instrukce je zanedbáván. Slučovací bit pro první byte druhé instrukce ve slučované dvojici se také zanedbává. V důsledku toho je zapotřebí pouze jeden bit informace pro identifikování a vhodné vykonávání slučovaných instrukcí. Bity příznaků 56 a 58 jsou také dostatečné pro informování dekódovací a řídicí logiky 60 o tom, že instrukce v paměťových polích 52 a 54 instrukčního registru mají být slučovány, tj. vykonávány paralelně. Dekódovací a řídicí logika 60 potom vyšetřuje instrukce paměťových polí 52 a 54 pro určení, jaký je jejich sled vykonávání, jaké jsou podmínky vzájemného blokování, pokud existují, a jaké funkce jsou požadovány. Toto určování je znázorněno pro instrukce kategorie 1 na obr. 5. Dekódovací a řídicí logika také určuje funkce vyžadované pro překonání jakýchkoli vzájemných blokování s datovými hazardy, jako je na obr. 6A a 6B.

Tato určení jsou konzolidována na obr. 7A a 7B. Podle obr. 7A a 7B, při předpokladu, že dekódovací a řídicí logika 60 určila na základě příznakových bitů, že instrukce v paměťových polích 52 a 54 se mají slučovat, vyšle logika 60 signál volby funkce na výstupu 66 udávající požadovanou operaci podle levého krajního sloupce na obr. 7A. Operační kódy instrukcí jsou explicitně dekódovány pro zjištění na výstupu signálů volby funkce konkrétních operací ve sloupcích se záhlavím OP1 a OP2 na obr. 7A a 7B. Signály volby registru na výstupu 62 směrují registry na obr. 8 pomocí prvku 64 vzájemného spojení, jak je požadováno ve sloupcích vstupů AI0, AI1 a AI2 na obr. 7A a 7B. Předpokládejme tak

například, že první instrukce v paměťovém poli 52 je ADD R1, R2 a že druhá instrukce je ADD R1, R4. Osmnáctá řádka na obr. 7A ukazuje operaci aritmeticko-logické jednotky, které dekódovací a řídicí obvod indikuje pomocí OP1 = + a OP2 = +, zatímco registr R2 je směrován na vstup operandový AI0, registr R4 na operandový vstup AI1 a registr R1 na operandový vstup AI2.

Obr. 9 slouží pro objasnění struktury a činnosti aritmeticko-logické jednotky 65 pro překonávání datové závislosti. Na obr. 9 je znázorněna sčítačka 70 se třemi operandy a s jediným výsledkem, odpovídající sčítačce z obr. 3. Sčítačka 70 dostává vstupy přes obvody mezi vstupy 70A, 70B, 70C sčítačky a operandové vstupy AI0, AI1 a AI1 aritmeticko-logické jednotky. Z operandového vstupu AI2 je operand směrován přes tři logické funkční obvody 71, 72 a 73 odpovídající logickým operacím AND, OR a EXCLUSIVE-OR. Tento operand je kombinován v těchto logických funkčních obvodech s jedním z druhých operandů a je směrován do operandového vstupu AI0 nebo AI1 podle nastavení multiplexoru 80. Multiplexor 75 volí buď nezměněný operand připojený k operandovému vstupu AI2, nebo výstup jednoho z logických funkčních obvodů 71, 72 nebo 73. Vstup zvolený multiplexorem 75 je veden do invertoru 77, a multiplexor 78 připojuje k jednomu vstupu 70C sčítačky 70 buď výstup invertoru 77, nebo neinvertovaný výstup multiplexoru 75. Druhý vstup 70B sčítačky 70 je získáván z operandového vstupu AI1 aritmeticko-logické jednotky multiplexorem 82, který volí buď "0", nebo operand připojený k operandovému vstupu AI1 aritmeticko-logické jednotky. Výstup multiplexoru je invetován invertorem 84 a multiplexor 85 volí buď neinvertovaný, nebo invertovaný výstup multiplexoru 82 jako druhý vstup operandů do sčítačky 70. Třetí vstup 70A do sčítačky 70 se získá z operandového vstupu AI0, který je invertován invertorem 87. Multiplexor 88 volí buď "0", vstup operandu do vstupu AI0, nebo jeho inverzní hodnotu vedenou jako třetí vstup do sčítačky 70. Výstup aritmeticko-logické jednotky se získá multiplexorem 95, který volí výstup 70D sčítačky 70 nebo výstup jednoho z logických funkčních obvodů 90, 92 nebo 93. Logické funkční obvody 90, 92 a 93 kombinují výstup sčítačky pomocí indikované logické operace vstupem operandu operandového vstupu AI1.

Mělo by být zřejmé, že signál volby funkce spočívá v podstatě v signálech volby A B C D E F G multiplexoru a vstupu voleb "horké" 1/0 do sčítačky 70. Bude zřejmé, že multiplexor volí rozmezí signálů x od jediného bitu pro signály A, B, E a F do dvoubitových signálů pro C, D a G.

Stavy komplexního řídicího signálu (A B C D E F G 1/0 1/0) je možno snadno odvodit z obr. 7A a 7B. Například pro ADD R1, R2 ADD R1, R4 ze shora uvedeného příkladu, by signál OP1 nastavil signál C multiplexoru pro volbu signálu přítomného na AI2, zatímco signál F by zvolil neinvertovaný výstup multiplexoru 75, čímž by poskytl operand v registru R1 do vstupu sčítačky 70 nejdále vpravo. Podobně by signály B a E multiplexoru byly nastaveny pro poskytnutí operandu, který je k dispozici na vstupu AI1 v neinvertované formě do prostředního vstupu sčítačky 70, zatímco signál D multiplexoru by byl nastaven pro poskytnutí operandu na vstup AI0 sčítačky 70 nejvíce vlevo bez inverze. Konečně, oba "1/0" vstupy jsou nastaveny přiměřeně pro obě sčítací operace. S těmito vstupy se výstup sčítačky 70 jednoduše součet tří ope-

randů, který odpovídá požadovanému výstupu aritmeticko-logické jednotky. Řídicí signál G by proto měl být nastaven tak, že multiplexor 95 by vydal na výstupu výsledek dosažený sčítačkou 70, který by byl součet operandů v registrech R1, R2 a R3.

Při slučování sledu instrukcí LOGICAL/ADD by logická funkce byla zvolena multiplexorem 75 a poskytnuta multiplexorem 78 do sčítačky 70, zatímco operand, který se má přičítat k logické operaci, by byl veden jedním z multiplexorů 85 nebo 88 na jeden ze druhých vstupů sčítačky 70, při přivádění 0 na třetí vstup. V tomto případě by multiplexor 95 byl nastaven pro volbu výstupu sčítačky 70 jako výsledku.

Konečně při sledu ADD/LOGICAL by dva operandy, které by měly být nejprve přičítány, vedeny na dva ze vstupů sčítačky 70, zatímco 0 by byla vedena na třetí vstup. Výstup sčítačky je okamžitě kombinován s nezvoleným operandem v logických funkčních obvodech 90, 92 a 93. Řídicí signál bude nastaven na volbu výstupu prvku, jehož operace odpovídá druhé instrukci slučovaného souboru.

Všeobecněji řečeno představuje obr. 9 logické znázorněné aritmeticko-logické jednotky 65 pro překonávání datové závislosti. Při odvozování tohoto proudu dat je činěno rozhodnutí nepodporující vzájemná blokování, v nichž se výsledek první instrukce používá jako oba operandy druhé instrukce. Podrobnější rozbor je možné nalézt v sekci "Idiosynkracie". Skutečnost, že toto znázornění implementuje ostatní operace požadované slučováním LOGICAL-AND, je možno vidět ze srovnání datového toku s funkčním sloupcem z obr. 5. V tomto sloupci je operace typu LOGICAL na dvou operandech následována operací typu ADD mezi výsledkem LOGICAL a třetím operandem. Toto je prováděno směrováním operandů tak, aby byly logicky kombinovány na operandových vstupech AI0 a AI2 z obr. 9 vhodným blokem z logických funkčních obvodů 71, 72 nebo 73, směrováním tohoto výsledku do sčítačky 70 a směrováním třetího operandu přes operandový vstup AI1 sčítačky. Inverze a poskytnutí "horké" jedničky nebo nuly jsou zajišťovány jako část signálu volby funkce, jak je požadováno specifikovanou aritmetickou operací. V jiných případech je operace typu ADD mezi dvěma operandy následována operací typu LOGICAL mezi výsledkem operace ADD a třetím operandem. Toto je vykonáváno směrováním operandů pro operaci typu ADD na operandový vstup AI0 a AI2, směrováním těchto vstupů do sčítačky, směrováním výstupů sčítačky do logických funkčních obvodů 90, 92 a 93 za sčítačkou a směrováním třetího operandu přes AI3 do těchto logických bloků za sčítačkou. Operace typu LOGICAL sledovaná operací typu LOGICAL jsou vykonávány směrováním dvou operandů pro první operaci typu LOGICAL na vstup AI0 a AI2, které jsou směrovány do logických obvodů před sčítačkou, směrováním výsledků z logických obvodů před sčítačkou přes aritmeticko-logickou jednotku mez modifikace přičtením nuly do logického obvodu umístěného za sčítačkou, a směrováním třetího operandu do logického obvodu umístěného za sčítačkou. Pro operaci typu ADD následovanou operací typu ADD jsou tři operandy směrovány na vstupy sčítačky a výstup sčítačky je předkládán na výstup aritmeticko-logické jednotky.

Operace aritmeticko-logické jednotky 65 pro vykonání druhé instrukce v instrukčním paměťovém poli 54, když není datová zá-

vislost mezi první a druhou instrukcí, je přímočará. V tomto případě jsou do aritmeticko-logické jednotky přiváděny pouze dva operandy. Je-li druhá instrukce, instrukce typu ADD, budou proto oba operandy vedeny na sčítačku 70 spolu s nulou na místě třetího operandu, přičemž výstup sčítačky bude zvolen multiplexorem 95 jako výstup aritmeticko-logické jednotky. Je-li druhá instrukce logická instrukce, může být logická instrukce vykonávána směrováním dvou operandů do logických funkčních obvodů 71, 72 a 73 při volbě volného výstupu, a potom se nechá výsledek projít sčítačkou 70, přičemž se na oba ostatní vstupy sčítačky zavedou nuly. V tomto případě by byl výstup sčítačky rovný logickému výsledku a byl by zvolen multiplexorem 95 jako výstup aritmeticko-logické jednotky. Alternativně může být jeden operand veden sčítačkou při přičtení dvou nul, takže ze sčítačky 70 na výstupu vyjde stejný operand. Tento operand je kombinován s druhým operandem v logických funkčních obvodech 90, 92 a 93, přičemž vhodný výstup logického obvodu se multiplexorem 95 zvolí jako výstup aritmeticko-logické jednotky.

Jsou-li instrukce slučovány jak je znázorněno na obr. 8, bez ohledu na to, zda závislost neexistuje nebo ano, instrukce v instrukčním paměťovém poli 52 instrukčního registru 50 bude s výhodou vykonána dekódováním instrukce přes sčítačku 70 a výstup 74, volbou jejich operandů přes sčítačku 70, výstup 710 a prvek 720 vzájemného spojení a vykonáváním zvolené operace na zvolených operandech v aritmeticko-logické jednotce 75. Protože aritmeticko-logická jednotka 75 je určena pro provádění jediné instrukce, jsou dva operandy přiváděny ze zvoleného registru přes operandové vstupy AI0, AI1, přičemž indikovaný výsledek je poskytován na výstupu 77.

Při uspořádání znázorněném na obr. 8 tak aritmeticko-logická jednotka 65 pro překonávání datové závislosti, v kombinaci s běžnou aritmeticko-logickou jednotkou 75, podporuje souběžné (nebo paralelní) vykonávání dvou instrukcí, i když existuje datová závislost mezi instrukcemi.

Generování adres může být také ovlivňováno datovými hazardy, které mohou být označeny jako adresové hazardy AHAZ. Následující sled představuje slučovaný sled instrukcí systému System/370, který je prostý adresových hazardů:

```
AR R1, R2
S R3, D/R4, R5
```

kde instrukce AR znamená "sečti" (ADD), instrukce S znamená "odečti" (SUBTRACT) a D představuje posun o tři čtyřbitová slova. Neexistuje žádný hazard AHAZ, protože R4 a R5, které se používají ve výpočtu adres, nebyly změněny předcházející instrukcí. Adresové hazardy existují v následujících sledech:

```
AR R1, R2
S R3, D/R1, R5/
AR R1, R2
S R3, D/R4, R1/
```

shora uvedené sledy demonstrují slučování instrukce RR (kategorie 1 na obr. 5) s instrukcemi RX (kategorie 9) vykazujícími AHAZ.

Jiné kombinace zahrnují instrukce RR slučované s instrukcemi RS a SI.

Pro aritmeticko-logickou jednotku pro překonávání vzájemného blokování musí být odvozeny nové operace vznikající z překonávání vzájemných blokování s hazardy AHAZ, a to analyzováním všech kombinací sledů instrukcí a konfliktů adresových operandů. Analýza ukazuje, že společná vzájemná blokování, jaká jsou obsažena ve shora uvedených sledech instrukcí, mohou být překonávána pomocí aritmeticko-logické jednotky čtyři na jedna.

Funkce, které by bylo třeba podporovat aritmeticko-logickou jednotkou pro překonání všech vzájemných blokování s hazardy AHAZ pro architekturu instrukční úrovně systému System/370 jsou zaznamenány na obr. 10. Pro tyto případy, kde nejsou uvedeny čtyři vstupy, je třeba použít implicitní nuly. Logický diagram aritmeticko-logické jednotky pro překonávání vzájemného blokování s hazardy AHAZ je poskytnut na obr. 11. Velký dílčí soubor, ale nikoliv všechny z funkcí uvedených na obr. 10, je podporován znázorněnou aritmeticko-logickou jednotkou. Tento dílčí soubor sestává z funkcí uvedených v řádcích jedna až dvacet jedna na obr. 10. Rozhodnutí, jaké funkce zahrnout, je implementační rozhodnutí, jehož rozbor je odsunut do sekce "Idiosynkracie".

Jak ukazuje obr. 11, zahrnuje znázorněná aritmeticko-logická jednotka sčítačku 100 v níž jsou kaskádovitě uspořádány třívstupové a dvouvstupové sčítačky 101 a 102 s uchovávanými přenosy s dvouvstupovou a jednovýstupovou sčítačkou 103 s predikcí přenosů takovým způsobem, že sčítačka 100 je skutečně čtyřoperandová sčítačka s jedním výsledkem pro činnost aritmeticko-logické jednotky na obr. 11.

Při generování obrázku 10 byla složitost struktury aritmeticko-logické jednotky zjednodušena na účet řídicí logiky. Toto je nejlépe vysvětleno příkladem. Uvažujme dva následující sledy instrukcí Systému/370:

```
NR R1, R2       (4)
S R3, D/R1,R5/
NR R1, R2       (5)
S R3, D/R4,R1/
```

kde NR značí instrukci "AND" a S značí instrukci "odečti" (subtract).

Nechť je obecný záznam pro tento sled

```
NR R1, R2
S R3,D/R4,R5/
```

Pro tento první sled je adresa operandu:

$$OA = D + (R1 \wedge R2) + R5$$

zatímco pro druhý sled je

$$OA = D + R4 + (R1 \wedge R2)$$

kde OA značí "adresa operandu" (operand address).

Pro zjednodušení řízení provádění na účet složitosti aritmeticko-logické jednotky, by bylo zapotřebí, aby aritmeticko-logická jednotka vykonala následující dvě operace:

$$OA = AGIO + (AGI1 \wedge AGI2) + AGI3$$
$$OA = AGIO + AGI2 + (AGI1 \wedge AGI3)$$

kde D je přiváděno na vstup AGIO, R2 je přiváděno na vstup AGI1, R4 je přiváděno na vstup AGI2 a R5 je přiváděno na vstup AGI3. Smysl označení vstupů AGI0, AGI1, AGI2 a AGI3 je zřejmý z analogie provedení na obr. 11, s provedením z obr. 9, které je vysvětleno níže. Aritmeticko-logická jednotka by však mohla být zjednodušena, jestliže řídicí jednotky zjistí, které z R4 a R5 mají hazard s R1 a dynamicky směrují tento registr na AGI2. Druhý registr by byl veden do AGI3. Pro tento předpoklad musí aritmeticko-logická jednotka pouze podporovat operaci:

$$OA = AGI0 + (AGI1 \wedge AGI2) + AGI3$$

Záměny jako tyto jsou provedeny za účelem snížení složitosti aritmeticko-logické jednotky pro generování adres, jakož i aritmeticko-logických jednotek pro vykonávání instrukcí a určení větvení.

Aritmeticko-logická jednotka z obr. 11 může nahradit aritmeticko-logickou jednotku na obr. 8 a 9. V tomto případě by dekódovací a řídicí logika 60 přiměřeně odrážela funkce z obr. 10.

Podobné analýzy, jako byly provedeny pro aritmeticko-logické jednotky, překonávající vzájemné blokování, pro vykonávání instrukcí a generování adres, musí být provedeny pro odvozování požadavků slučování na aritmeticko-logické jednotce s určováním větvení, jak je uvedeno na obr. 12 a 13. Aritmeticko-logická jednotka pro určování větvení pokrývá funkce požadované instrukcemi srovnávajícími hodnoty registrů. To zahrnuje instrukce větvení BXLE, BXH, BCT a BCTR, v nichž je hodnota registru zvyšována o obsah druhého registru (BXLE a BXH) nebo je snižována o jednu (BCT a BCTR) před tím, než je srovnávána s hodnotou registru (BXLE a BXH) nebo nulou (BCT a BCTR) pro určení výsledku větvení. Podmíněná větvení nejsou vykonávána touto aritmeticko-logickou jednotkou.

Aritmeticko-logická jednotka znázorněná na obr. 13 obsahuje vícestupňovou sčítačku 10, v níž jsou dvě sčítačky 111 a 112 s uchovávanými přenosy kaskádovitě uspořádány, se dvěma výstupy sčítačky 112 s uchovávanými přenosy zajišťujícími dva vstupy pro sčítačku 113 s predikcí přenosů. Tato kombinace účinně zajišťuje čtyřvstupovou sčítačku s jediným výsledkem, použitou pro aritmeticko-logickou jednotku z obr. 13.

Jako příklad datových hazardů, které se mohou vyskytnout, je možno uvažovat následující sled instrukcí:

```
AL R1, D/R2,R3/
BCT R1,D/R2,R3/,
```

kde AL značí instrukci "ADD LOGICAL" a BCT instrukci "BRANCH ON COUNT".
Nechť (x) označuje obsah paměťového místa x. Výsledky následující provedení jsou

R1 = R1 + (D + R2 + R3) - 1
Větvi, jestliže (R1 + (D + R2 + R3) - 1 = 0.

Toto porovnání je možno provést vykonáním operace:

R1 + (D + R2 + R3) - 1 - 0

Výsledky analýz pro aritmeticko-logickou jednotku s určováním větvení jsou uvedeny v obr. 12 a 13 bez dalšího rozboru. Funkce podporované proudem dat zahrnují ty, které byly specifikovány řadami 1 až 25 na obr. 12. Aritmeticko-logickou jednotku 65 z obr. 8. V tomto případě by dekódovací a řídicí logika 60 přiměřeně odrážela funkce z obr. 12.

Některé z funkcí, které vznikají z konfliktů operandů, jsou složitější než ostatní. Například sled instrukcí:

AR R1, R2
AR R1, R1

kde AR značí instrukci "sečti" (add), vyžaduje aritmeticko-logickou jednotku čtyři na jedna, spolu se z toho vyplývající složitostí, pro překonání datového vzájemného blokování, protože její vykonávání má za následek:

R1 = (R1 + R2) + (R1 + R2).

Jiné sledy mají za následek operace, které vyžadují, aby do aritmeticko-logické jednotky bylo začleněno přídavné zpoždění za účelem překonání vzájemného blokování. Sled, který ilustruje toto zvýšené zpoždění je:

SR R1, R2
LPR R1, R1

kde S značí instrukci "odečti" a LP instrukci "zaveď kladné" (load positive), což má za následek operaci

R1 = (R1 = R2).

Tato operace se nehodí k paralelnímu provádění, protože výsledky odečítání jsou zapotřebí k nastavení výpočtu absolutní hodnoty.

Místo překonání všech vzájemných blokování v aritmeticko-logické jednotce je možné použít logiku vydávající instrukce nebo předprocesor, který je určen pro zjišťování sledů instrukcí, které vedou k těmto složitějším funkcím. Detekce předprocesorem odstraňuje potřebu přidávat zpoždění k logice vydávající instrukce, která je často v blízkosti kritické cesty. Když se zjistí takový sled, logika vydávající instrukce nebo předprocesor se přepnou na vydávání sledu ve skalárním modu při vyloučení potřeby překonávat vzájemné blokování. Rozhodnutí pokud jde o to, které instrukční sledy mají nebo nemají mít jejich vzájemná blokování

překonávána, je implementační rozhodování, které je závislé na faktorech nad rámec vynálezu. Je však třeba upozornit na možnost kompromisů mezi složitostí realizace aritmeticko-logické jednotky a složitostí logiky pro vydávání instrukcí.

Hazardy přítomné v generování adres také dávají vznik implementaci kompromisů. Například většina ze vzájemných blokování při generování adres může být překonána použitím aritmeticko-logické jednotky čtyři na jedna, jaká byla rozebrána shora. Následující sled

```
AR R1, R2
S R3,D/R1,R1/,
```

kde AR značí instrukci "sečti" a S instrukci "odečti", však nezapadá do této kategorie. Pro tento případ je zapotřebí aritmeticko-logické jednotky pět na jedna pro překonání vzájemného blokování s hazardem AHAZ, protože výsledná operace je:

$$OA = D + (R1 + R2) + (R1 + R2),$$

kde OA je adresa výsledného operandu. Jako dříve, je vřazení této funkce do aritmeticko-logické jednotky implementační rozhodnutí, které závisí na častosti výskytu takového vzájemného blokování. Podobné výsledky se také vztahují na aritmeticko-logickou jednotku pro určování větvení.

Analýzy podobné těm, jaké byly předvedeny shora, mohou být provedeny pro odvození hardwaru pro překonávání vzájemného blokování pro nejobecnější případ n vzájemných blokování. Pro tento rozbor je možno se odvolat na obr. 14. Předpokládáme jednoduchá datová vzájemná blokování, jako:

```
AR R1,R2
AR R3,R1
```

kde AR značí instrukci "sečti"
kde se změněný registr z první instrukce použije jako pouze jediný z operandů druhé instrukce, přičemž pro překonání vzájemného blokování by bylo zapotřebí aritmeticko-logické jednotky (n+1) na jedna. Pro překonání například tří vzájemných blokování při použití shora uvedeného předpokladu by byla zapotřebí aritmeticko-logická jednotka čtyři na jedna. To by vyžadovalo přídavný stupeň se sčítačkou s uchovávanými přenosy v aritmeticko-logické jednotce.

PATENTOVÉ NÁROKY

1. Hardwarové zařízení pro překonávání datové závislosti v počítači s architekturou pro sériové vykonávání sledu skalárních instrukcí, vyznačující se tím, že obsahuje instrukční registr (50) s nejméně dvěma paměťovými poli (52, 54), nejméně tři univerzální registry (63A, 63B, 63C), každý pro jeden operand, prvek (64) vzájemného spojení, který je spojen se společným výstupem univerzálních registrů, dekódovací a řídicí logiku (60), mající své vstupy připojené k výstupu instrukčního registru (50) a mající svůj vyvolovací výstup (62) registrů spojený s prvkem (64) vzájemného spojení a mající svůj vyvolovací výstup (66) funkcí spojený s aritmetickou a logickou jednotkou (65), která je dále opatřena třemi operandovými vstupy (AI0, AI1, AI2), spojenými s uvedeným prvkem (64) vzájemného spojení a má jeden výstup (67).

2. Hardwarové zařízení podle nároku 1, vyznačující se tím, že aritmetická a logická jednotka (65) obsahuje sčítačku (70), opatřenou třemi operandovými vstupy (70A, 70B, 70C) a připojenou k operandovým vstupům (AI0, AI1, AI2) aritmetické a logické jednotky (65) a jedním výsledkovým výstupem (70D) spojeným s výstupem (67) aritmetické a logické jednotky (65).

3. Hardwarové zařízení podle nároku 2, vyznačující se tím, že sčítačka (70) zahrnuje sčítačku (10) s uchovávanými přenosy, opatřenou třemi vstupy spojenými s operandovými vstupy (70A, 70B, 70C) sčítačky (70) a se součtovým výstupem (12) a přenosovým výstupem (14), a dále sčítačku (22) s predikcí přenosů, opatřenou dvěma vstupy připojenými k příslušnému součtovému výstupu (12) a přenosovému výstupu (14) sčítačky (10) s uchováváním přenosů a s výstupem (26), spojeným s výsledkovým výstupem (70D) sčítačky (70).

4. Hardwarové zařízení podle nároků 2 nebo 3, vyznačující se tím, že dva ze tří operandových vstupů (AI0, AI1, AI2) aritmetické a logické jednotky (65) jsou připojeny ke třem logickým funkčním obvodům (71, 72, 73), které jsou dále spojeny s jedním ze tří operandových vstupů (70C) sčítačky (70).

5. Hardwarové zařízení podle kteréhokoli z nároků 2 až 4, vyznačující se tím, že jeden z uvedených tří operandových vstupů (AI0, AI1, AI2) aritmetické a logické jednotky (65) a výsledkový výstup (70D) sčítačky (70) jsou připojeny ke třem logickým funkčním obvodům (90, 92, 93), které jsou dále spojeny s výstupem (67) aritmetické a logické jednotky (65).

6. Hardwarové zařízení podle nároků 4 nebo 5, vyznačující se tím, že první logické funkční obvody (71, 90) na vstupní a výstupní straně sčítačky (70) jsou součinové obvody, druhé logické funkční obvody (72, 92) jsou součtové

obvody a třetí logické obvody (73, 90) jsou obvody výlučného součtu.

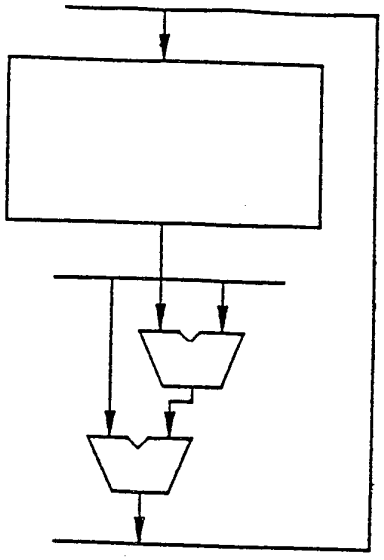
7. Hardwarové zařízení podle nároku 6, vyznačující se tím, že dva (AI0, AI1) ze tří operandových vstupů (AI0, AI1, AI2) aritmetické a logické jednotky (65) jsou připojeny k multiplexoru (80), který je dále připojen ke třem logickým funkčním obvodům (71, 72, 73).

8. Hardwarové zařízení podle nároku 7, vyznačující se tím, že tři logické funkční obvody (71, 72, 73) jsou připojeny k multiplexoru (75), který je dále spojen s jedním ze tří operandových vstupů (70A, 70B, 70C) sčítačky (70).

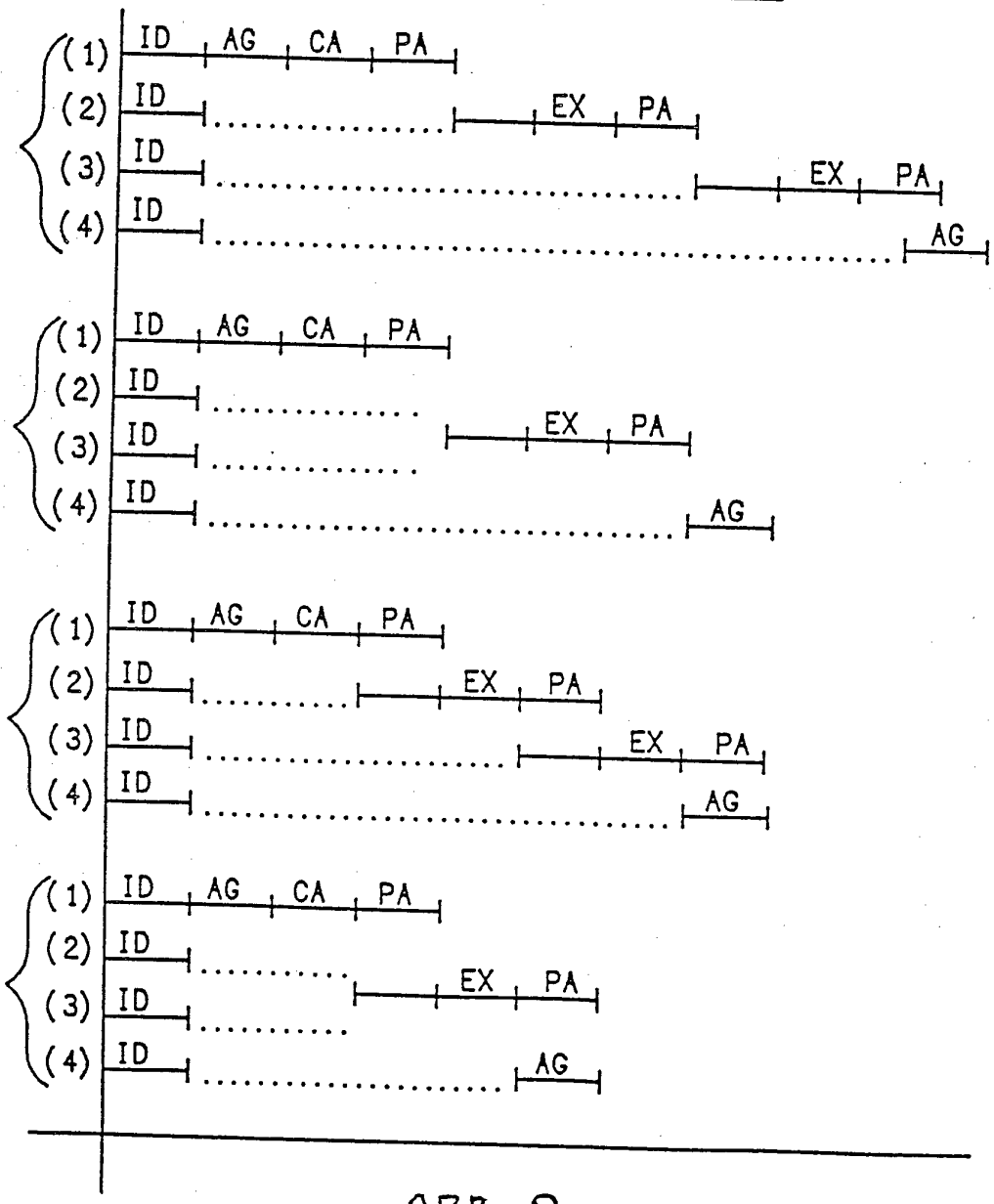
9. Hardwarové zařízení podle kteréhokoli z nároků 6 až 8, vyznačující se tím, že tři logické funkční obvody (90, 92, 93) jsou spojeny s multiplexorem (95), který je dále spojen s výstupem (67) aritmetické a logické jednotky (65).

10. Hardwarové zařízení podle kteréhokoli z nároků 1 až 9, vyznačující se tím, že prvek (64) vzájemného spojení obsahuje multiplexor pro zavádění obsahů tří univerzálních registrů (63A, 63B, 63C) na tři operandové vstupy (AI0, AI1, AI2) aritmetické a logické jednotky (65).

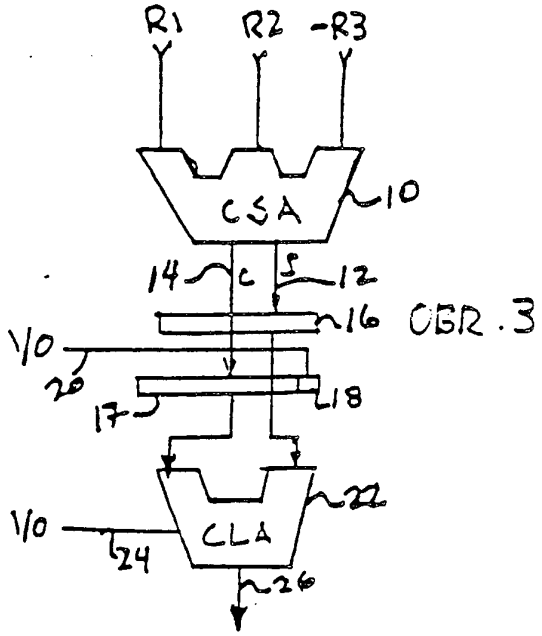
13 výkresů



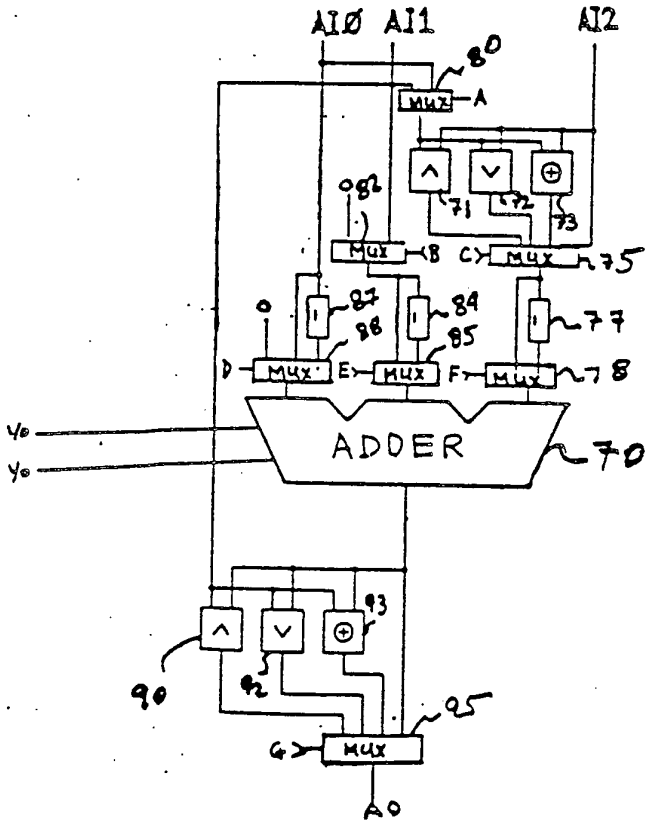
OBR. 1


(1) ID AG CA PA
(2) ID EX PA
(3) ID EX PA
(4) ID AG
(1) ID AG CA PA
(2) ID EX PA
(3) ID AG
(4) ID
(1) ID AG CA PA
(2) ID EX PA
(3) ID EX PA
(4) ID AG
(1) ID AG CA PA
(2) ID EX PA
(3) ID
(4) ID AG

OBR 2


R1
R2
-R3
CSA
10
14
C
S
12
16
I/O
20
17
18
CLA
22
I/O
24
26

OBR. 3


AIØ
AI1
AI2
80
MUX
A
82
71
72
73
MUX
B
C
MUX
75
87
84
77
86
85
D
MUX
E
MUX
F
MUX
78
ADDER
70
93
90
92
95
MUX
G
AO

OBR. 9

1. RR-FORMAT LOADS, LOGICALS, ARITHMETICS, COMPARES
   - LCR - LOAD COMPLEMENT
   - LPR - LOAD POSITIVE
   - LNR - LOAD NEGATIVE
   - LR - LOAD REGISTER
   - LTR - LOAD AND TEST
   - NR - AND
   - OR - OR
   - XR - EXCLUSIVE OR
   - AR - ADD
   - SR - SUBTRACT
   - ALR - ADD LOGICAL
   - SLR - SUBTRACT LOGICAL
   - CLR - COMPARE LOGICAL
   - CR - COMPARE

2. RS-FORMAT SHIFTS (NO STORAGE ACCESS)
   - SRL - SHIFT RIGHT LOGICAL
   - SLL - SHIFT LEFT LOGICAL
   - SRA - SHIFT RIGHT ARITHMETIC
   - SLA - SHIFT LEFT ARITHMETIC
   - SRDL - SHIFT RIGHT LOGICAL
   - SLDL - SHIFT LEFT LOGICAL
   - SRDA - SHIFT RIGHT ARITHMETIC
   - SLDA - SHIFT LEFT ARITHMETIC

3. BRANCHES - ON COUNT AND INDEX
   - BCT - BRANCH ON COUNT (RX-FORMAT)
   - BCTR - BRANCH ON COUNT (RR-FORMAT)
   - BXH - BRANCH ON INDEX HIGH (RS-FORMAT)
   - BXLE - BRANCH ON INDEX LOW (RS-FORMAT)

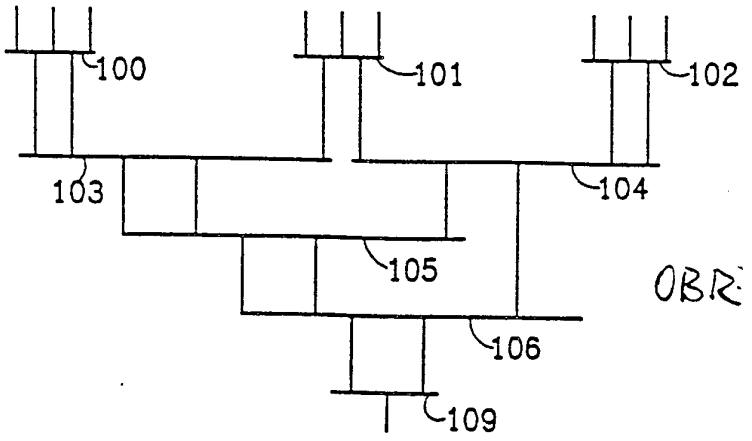
4. BRANCHES - ON CONDITION
   - BC - BRANCH ON CONDITION (RX-FORMAT)
   - BCR - BRANCH ON CONDITION (RR-FORMAT)

5. BRANCHES - AND LINK
   - BAL - BRANCH AND LINK (RX-FORMAT)
   - BALR - BRANCH AND LINK (RR-FORMAT)
   - BAS - BRANCH AND SAVE (RX-FORMAT)
   - BASR - BRANCH AND SAVE (RR-FORMAT)

OBR. 4A

6. STORES
   - STCM - STORE CHARACTERS UNDER MASK (0-4-BYTE STORE, RS-FORMAT)
   - MVI - MOVE IMMEDIATE (ONE BYTE, SI-FORMAT)
   - ST - STORE (4 BYTES)
   - STC - STORE CHARACTER (ONE BYTE)
   - STH - STORE HALF (2 BYTES)
7. LOADS
   - LH - LOAD HALF (2 BYTES)
   - L - LOAD (4 BYTES)
8. LA - LOAD ADDRESS
9. RX/SI/RS-FORMAT ARITHMETICS, LOGICALS, INSERTS, COMPARES
   - A - ADD
   - AH - ADD HALF
   - AL - ADD LOGICAL
   - N - AND
   - O - OR
   - S - SUBTRACT
   - SH - SUBTRACT HALF
   - SL - SUBTRACT LOGICAL
   - X - EXCLUSIVE OR
   - IC - INSERT CHARACTER
   - ICM - INSERT CHARACTERS UNDER MASK (0- TO 4-BYTE FETCH)
   - C - COMPARE
   - CH - COMPARE HALF
   - CL - COMPARE LOGICAL
   - CLI - COMPARE LOGICAL IMMEDIATE
   - CLM - COMPARE LOGICAL CHARACTER UNDER MASK
10. TM - TEST UNDER MASK

OBR. 4B



OBR. 14

FUNCTIONS ARISING FROM LOGICAL(ϕ), ADD(ζ) INSTRUCTION COMPOUNDINGS		
SEQUENCE	INTERLOCK CONDITION	FUNCTION
LOGICAL, ADD	R1=R3≠R2≠R4	(R1φR2)ζR4
LOGICAL, ADD	R1=R4≠R2≠R3	R3ζ(R1φR2)
LOGICAL, ADD	R1=R2=R3≠R4	(R1φR1)ζR4
LOGICAL, ADD	R1=R2=R4≠R3	R3ζ(R1φR1)
LOGICAL, ADD	R1=R3=R4≠R2	(R1φR2)ζ(R1φR2)
LOGICAL, ADD	R1=R2=R3=R4	(R1φR1)ζ(R1φR1)
ADD, LOGICAL	R1=R3≠R2≠R4	(R1ζR2)φR4
ADD, LOGICAL	R1=R4≠R2≠R3	R3φ(R1ζR2)
ADD, LOGICAL	R1=R2=R3≠R4	(R1ζR1)φR4
ADD, LOGICAL	R1=R2=R4≠R3	R3φ(R1ζR1)
ADD, LOGICAL	R1=R3=R4≠R2	(R1ζR2)φ(R1ζR2)
ADD, LOGICAL	R1=R2=R3=R4	(R1ζR1)φ(R1ζR1)
LOGICAL, LOGICAL	R1=R3≠R2≠R4	(R1φR2)φR4
LOGICAL, LOGICAL	R1=R4≠R2≠R3	R3φ(R1φR2)
LOGICAL, LOGICAL	R1=R2=R3≠R4	(R1φR1)φR4
LOGICAL, LOGICAL	R1=R2=R4≠R3	R3φ(R1φR1)
LOGICAL, LOGICAL	R1=R3=R4≠R2	(R1φR2)φ(R1φR2)
LOGICAL, LOGICAL	R1=R2=R3=R4	(R1φR1)ζ(R1φR1)
ADD, ADD	R1=R3≠R2≠R4	(R1ζR2)ζR4
ADD, ADD	R1=R4≠R2≠R3	R3ζ(R1ζR2)
ADD, ADD	R1=R2=R3≠R4	(R1ζR1)ζR4
ADD, ADD	R1=R2=R3≠R3	R3ζ(R1ζR1)
ADD, ADD	R1=R3=R4≠R2	(R1ζR2)ζ(R1ζR2)
ADD, ADD	R1=R2=R3=R4	(R1ζR1)ζ(R1ζR1)

OBR. 5

		OP1	OP2
1	AI2		
2	-AI2		
3	/AI2/		
4	-/AI2/		
5	AI1 OP1 AI2	∧,∨,⊕,+,-	
6	AI1 OP1 (-AI2)	∧,∨,⊕	
7	AI1 OP1 /AI2/	∧,∨,⊕	
8	AI1 OP1 (-/AI2/)	∧,∨,⊕	
9	(-AI2) OP1 (-AI2)	∧,∨,⊕	
10	/AI2/ OP1 /AI2/	∧,∨,⊕	
11	(-/AI2/) OP1 (-/AI2/)	∧,∨,⊕	
12	AI0 OP1 AI2	+,-	
13	/AI0/ OP1 AI2	+,-	
14	(-/AI0/) OP1 A12	+,-	
15	(-AI0) OF1 AI2	+,-	
16	/AI0/ OP1 /AI2/	+,-	
17	(-/AI0/) OP1 /AI2/	+,-	
18	AI2 OP1 AI2	∧,∨,⊕	
19	-(AI2 OP1 AI0)	∧,∨,⊕	
20	AI2 OP1 AI0	∧,∨,⊕	
21	/AI2 OP1 AI0/	∧,∨,⊕	
22	-/AI2 OP1 AI0/	∧,∨,⊕	
23	AI1 OP2 (AI2 OP1 AI0)	+,-	+,-
24	-AI1 OP2 (AI2 OP1 AI0)	+,-	+
25	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	∧,∨,⊕,+,-
26	(AI2 OP1 AI0) OP2 AI1	+,-	∧,∨,⊕
27	-(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+
28	AI2- 1		
29	(-AI2)- 1		

OBR. 6A

30	/AI2/ - 1		
31	(-/AI2/) - 1		
32	(AI2 OP1 AI0) - 1	∧,∨,⊕,+,-	
33	(AI2 - 1) OP1 AI1	∧,∨,⊕	
34	AI2 OP1 (AI0 - 1)	+,-	
35	(AI0 - 1) OP1 AI2	+,-	
36	-(AI2 - 1)		
37	(AI2 - 1) OP1 (AI2 - 1)	∧,∨,⊕	
38	/AI2 - 1/		
39	-/AI2 - 1/		
40	/AI2 OP1 AI0/	+,-	
41	-/AI2 OP1 AI0/	+	
42	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	∧,∨,⊕	∧,∨,⊕,+,-
43	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	+,-	∧,∨,⊕
44	AI0 + AI1 + AI2 + AI3		
45	-AI0 + AI1 - AI2 + AI3		
46	(AI2 - 1) OP1 (AI2 - 1)	+,-	

OBR. 6B

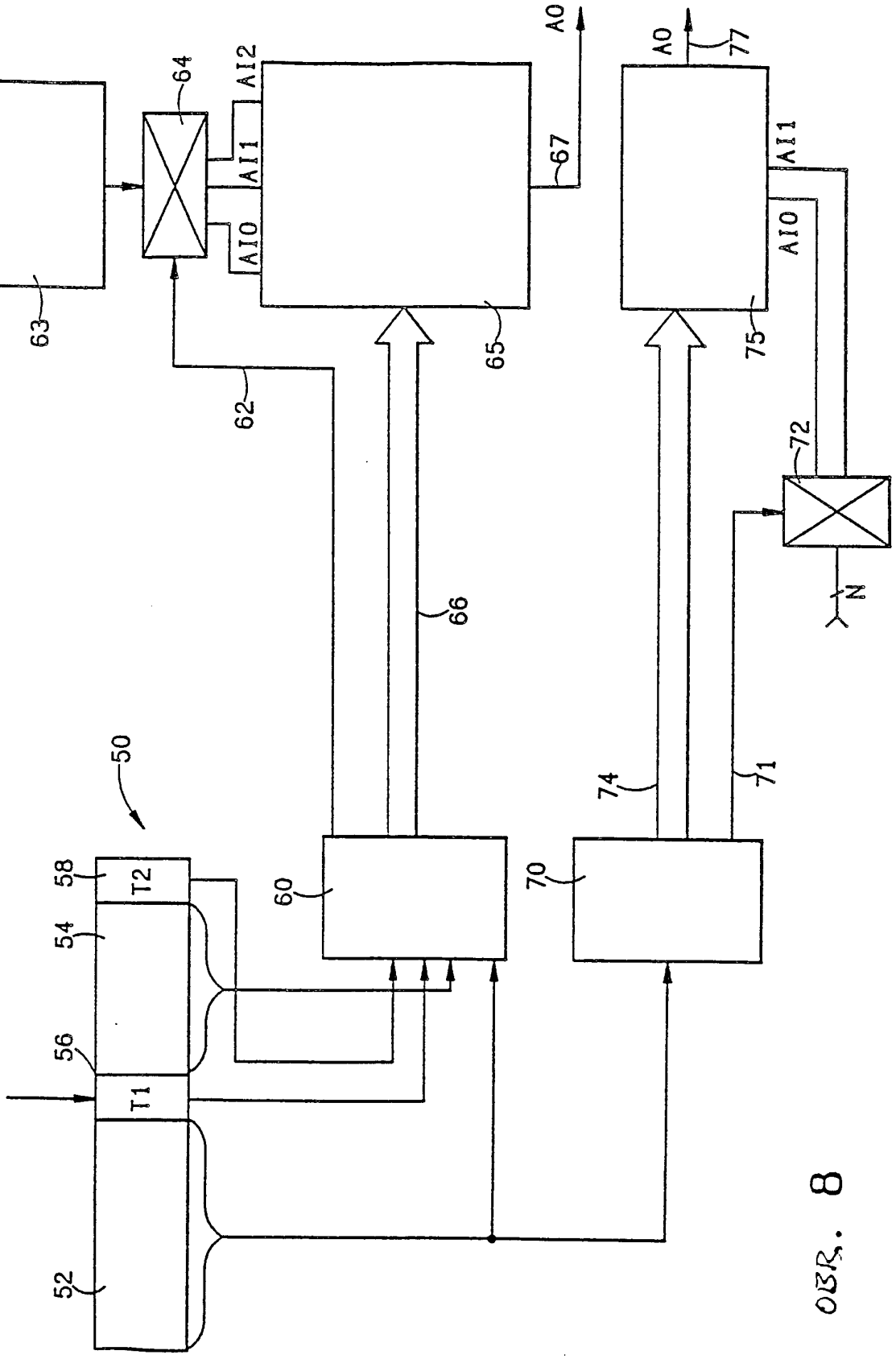
	ALU OP	OP1	OP2	AI0	AI1	AI2
(R1φR2)ζR4	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+,−	R2	R4	R1
R3ζ(R1φR2)	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+	R2	R3	R1
	−(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+	R2	R3	R1
(R1φR1)ζR4	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+,−	R2(=R1)	R4	R1
R3ζ(R1φR1)	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+	R2(=R1)	R3	R1
	−(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	+	R2(=R1)	R3	R1
(R1φR2)ζ(R1φR2)	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	∧,∨,⊕	+,−			
(R1ζR2)φR4	(AI2 OP1 AI0) OP2 AI3	+,−	∧,∨,⊕	R2	R4	R1
R3φ(R1ζR2)	(AI2 OP1 AI0) OP2 AI1	+,−	∧,∨,⊕	R2	R3	R1
(R1ζR1)φR4	(AI2 OP1 AI0) OP2 AI1	+,−	∧,∨,⊕	R2(=R1)	R4	R1
R3φ(R1ζR1)	(AI2 OP1 AI0) OP2 AI1	+,−	∧,∨,⊕	R2(=R1)	R3	R1
(R1ζR2)φ(R1ζR2)	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	+,−	∧,∨,⊕			
(R1φR2)φR4	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	∧,∨,⊕	R2	R4	R1
R3φ(R1φR2)	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	∧,∨,⊕	R2	R3	R1
(R1φR1)φR4	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	∧,∨,⊕	R2(=R1)	R4	R1
R3φ(R1φR1)	(AI2 OP1 AI0) OP2 AI1	∧,∨,⊕	∧,∨,⊕	R2(=R1)	R3	R1
(R1φR2)φ(R1φR2)	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	∧,∨,⊕	∧,∨,⊕			
(R1ζR2)ζR4	AI1 OP2 (AI2 OP1 AI0)	+,−	+	R2	R4	R1
	−AI1 OP2 (AI2 OP1 AI0)	+,−	+	R2	R4	R1
R3ζ(R1ζR2)	AI1 OP2 (AI2 OP1 AI0)	+,−	+,−	R2	R3	R1

ÖBR. 7A

CZ 279873 B6

(R1ζR1)ζR4	AI1 OP2 (AI2 OP1 AI0)	+, −	+	R2(=R1)	R4	R1
	−AI1 OP2 (AI2 OP1 AI0)	+, −	+	R2(=R1)	R4	R1
R3(R1ζR1)	AI1 OP2 (AI2 OP1 AI0)	+, −	+, −	R2(=R1)	R3	R1
(R1ζR2)ζ(R1ζR2)	AI2 OP1 AI0 + AI1 OP2 AI3	+, −	+, −			

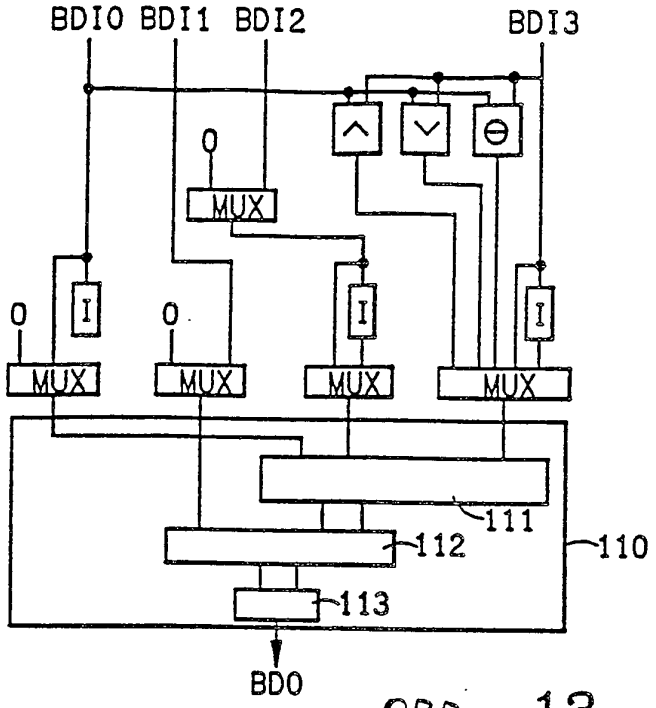
FIG. 7B


A0
64
AI2
AI1
AI0
67
63
65
62
66
A0
77
AI1
AI0
75
72
N
50
58
T2
54
56
T1
52
60
70
74
71

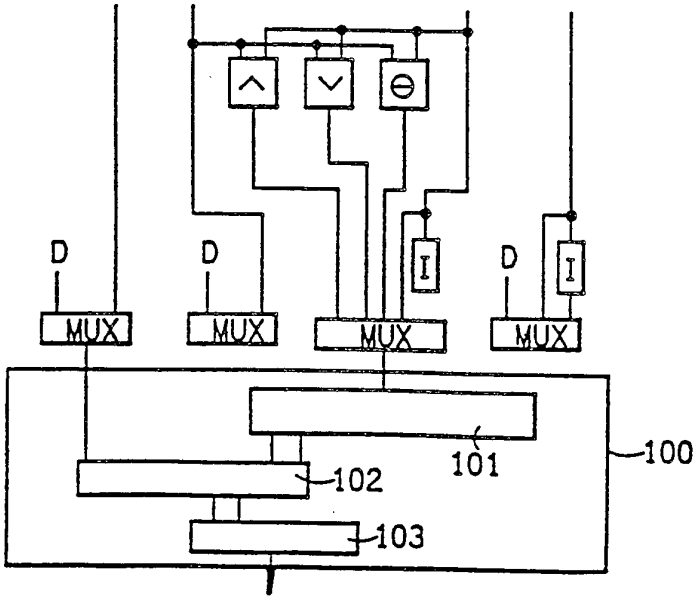
OBR. 8

		OP1
1	AGI2	
2	-AGI2	
3	/AGI2/	
4	-/AGI2/	
5	AGI1 OP1 AGI2	$\wedge, \vee, \oplus, +, -$
6	AGIO + AGI2	
7	AGIO - AGI2	
8	AGIO + AGI2 + AGI3	
9	AGIO - AGI2 + AGI3	
10	AGIO + AGI2 - AGI3	
11	AGIO - AGI2 - AGI3	
12	AGIO + /AGI2/	
13	AGIO - /AGI2/	
14	AGIO + /AGI2/ + AGI3	
15	AGIO + AGI2 + /AGI3/	
16	AGIO + /AGI2/ + /AGI3/	
17	AGIO - /AGI2/ + AGI3	
18	AGIO + AGI2 - /AGI3/	
19	AGIO - /AGI2/ - /AGI3/	
20	AGIO + (AGI1 OP1 AGI2)	$\wedge, \vee, \oplus, +, -$
21	AGIO + (AGI1 OP1 AGI2) + AGI3	$\wedge, \vee, \oplus, +, -$
22	AGIO + (AGI1 OP1 AGI2) + (AGI3 OP1 AGI4)	$\wedge, \vee, \oplus, +, -$

OBR. 10


BDI0 BDI1 BDI2
BDI3
0
MUX
0
I
0
MUX
MUX
MUX
MUX
111
112
110
113
BDO

OBR. 13


D
D
I
D
I
MUX
MUX
MUX
MUX
101
100
102
103

OBR. 11

		OP1
1	BDI1 + BDI2 - BDI3	
2	BDI1 + BDI2 + BDI3	
3	BDI1 + BDI2 + /BDI3/	
4	BDI1 + BDI2 - /BDI3/	
5	BDI1 - BDI2 - BDI3	
6	BDI1 + /BDI2/ - BDI3	
7	BDI1 - /BDI2/ - BDI3	
8	(-BDI0) + BDI2 - BDI3	
9	(-BDI0) - BDI2 - BDI3	
10	(-BDI0) + BDI2 + BDI3	
11	BDI0 + BDI2 - BDI3	
12	/BDI0/ + BDI2 - BDI3	
13	/BDI0/ + BDI2 - /BDI3/	
14	(-/BDI0/) + BDI2 - BDI3	
15	(-/BDI0/) + BDI2 + /BDI3/	
16	(-/BDI0/) - /BDI2/ - BDI3	
17	/BDI0/ + /BDI2/ - BDI3	
18	(BDI0 OP1 BDI3) + BDI1 - BDI2	
19	BDI1 + BDI2 - (BDI3 OP1 BDI0)	$\wedge,\vee,\oplus,+,-$
20	BDI1 - BDI2 + (BDI3 OP1 BDI0)	$\wedge,\vee,\oplus,+,-$
21	BDI3 - 1 - 0	
22	(-BDI3) - 1 - 0	
23	/BDI3/ - 1 - 0	
24	(-/BDI3/) - 1 - 0	
25	(BDI3 OP1 BDI0) - 1 - 0	$\wedge,\vee,\oplus,+,-$
26	(BDI3 OP1 BDI0) + BDI2 - (BDI3 OP1 BDI0)	$\wedge,\vee,\oplus,+,-$
27	(BDI3 OP1 BDI0) - BDI2 + (BDI3 OP1 BDI0)	$\wedge,\vee,\oplus,+,-$

OBR. 12

Konec dokumentu