KÖZZÉTÉTELI PÉLDÁNY

~~53.965/MK~~

1100/91

KIVONAT

Kapcsolási elrendezés adatfüggőség megszüntetésére, valamint többfunkciós ALU a kapcsolási elrendezéshez

International Business Machines Corp., ARMONK, US

A bejelentés napja: 1991. 04. 04.

Uniós elsőbbsége: 1990. 04. 04. (504,910) US

Jelen találmány tárgya kapcsolási elrendezés adatfüggőség megszüntetésére egy skalár számítógépben, a skalár utasítások párhuzamos (végrehajtása) során, amikor az utasítások egyike egy párhuzamosan végrehajtott utasítás által létrehozott eredményt operandusként használ.

Egy digitális adatfeldolgozásban használható, többfunkciós ALU-t írunk le, amely megkönnyíti az utasítások párhuzamos végrehajtását, miáltal megnöveli a processzor teljesítményét. A javasolt elrendezés csökkenti az utasításvégrehajtás latenciáját, amit egy az utasításokat szekvenciálisan feldolgozó számítógépben az adatfüggőségi hazárdok okoznak. Ezt a lappangás csökkentést a fenti hazárdokból származó blokkolások feloldásával valósítjuk meg. A javasolt elrendezéssel teljesítmény növekedést érünk el úgy, hogy az azonos arhitetektúrára tervezett, korábbi megvalósításokkal a csereszabatosság megmarad.

(jellemző ábra: 9. ábra)

S.B.G. & K.
BUDAPESTI NEMZETKÖZI ÜGYVÉDI
ÉS SZABADALMI IRODA
1061 BUDAPEST, DALSZÍNHÁZ U. 10.
TELEFON: 153-3733

NSZO: G06F 9/00

Kapcsolási elrendezés adatfüggőség megszüntetésére, valamint többfunkciós ALU a kapcsolási elrendezéshez

International Business Machines Corp., ARMONK, US

Feltalálók: BLANER, Bartholomew, NEWARK VALLEY
VASSILIADIS, Stamatis, VESTAL,
PHILLIPS, James Edward, BINGHAMPTON, US

A bejelentés napja: 1991. 04. 04.
Uniós elsőbbsége: 1990. 04. 04. (504,910) US

A találmány tárgya kapcsolási elrendezés adatfüggőség megszüntetésére, valamint többfunkciós ALU a kapcsolási elrendezéshez egy skalár számítógépben, skalár utasítások párhuzamos végre-

hajtására, amikor az utasítások egyike egy párhuzamosan végrehajtott utasítás által létrehozott eredményt operandusként használ.

A számítógép tervezők a számítógép rendszerek teljesítőképességének növelésére a felfűzést szokásos megoldásként alkalmazzák. A felfűzésnél egy utasítást számos lépésre ill. fokozatra osztják szét, amelyekhez elhelyezik a szükséges speciális kapcsolási elrendezést, hogy az adott fokozathoz kijelölt funkciót megvalósítsák. Az utasításfelfűző egységen (csővonalon) átmenő utasításfolyam sebessége inkább függ az új utasítások felfűző egységbe történő belépési sebességétől, mintsem a felfűző egység hosszától. Ideális felfűző egység elrendezésnél, ahol a felfűző egységbe ciklusonként maximum egy utasítást táplálnak be, a felfűző egység átbocsátó képessége, azaz az egységnyi idő alatt végrehajtott utasítások számának nagysága, csak a ciklusidőtől függ. Ha egy n-fokozatban megvalósított csővonal ciklusidejét n/m-nek tételezük fel - ahol az m a felfűzést nem alkalmazó, megfelelő megvalósítás ciklusideje - akkor a felfűzés által nyújtott maximális, potenciális javulás n-szeres.

Annak ellenére, hogy a fentiek azt jelzik, hogy a felfűzés egy számítógép teljesítőképességében egy n-szeres, potenciális javulást ajánl, egy sor gyakorlati korlátozás azt eredményezi, hogy a tényleges teljesítmény-növekedés kisebb lesz, mint az ideális esetben. Ezen korlátozásokat a felfűzési hazárdok jelenléte eredményezi. Egy felfűzési hazárdot, amely meggátolja, hogy az utasítások a maximális sebességgel haladjanak át a struktúrán, a csővonal szerkezetének bármelyik szempontjából definiálhatjuk. A csővonalbeli hazárdokat okozhatják adat-függőségek, szerkezeti (hardver erőforrásbeli) ellentmondások,

vezérlés-függőségek és más tényezők.

Az adat-függőségi hazárdokat gyakran írás-olvasás hazárdoknak vagy írás-olvasás blokkolásnak nevezik, mert az első utasításnak be kell írnia eredményét, mielőtt a második utasítás kiolvassa és ezután felhasználja ezt az eredményt. Ahhoz, hogy ezt az írást az olvasás előtt tegyék lehetővé, az olvasás végrehajtását az írás végrehajtásának idejére blokkolni kell. Ezen blokkolás a blokkolt utasítás végrehajtásában egy inaktív ciklust eredményez - amelyet gyakran "zárványnak" vagy "sebességvesztésnek" titulálnak. A zárvány a leállított utasítás teljes végrehajtási idejéhez egy további ciklust ad hozzá és így csökkenti a csővonal áteresztő képességét. Ha hardverben valósították meg, a szerkezeti- és adatfüggőségi hazárdok észlelése és feloldása nem csak a hardver alulkihasználásából adódó teljesítményveszteségeket eredményezheti, hanem a számítógép kritikus útjává is válhat. Ez a hardver ezután lecsökkentheti a gép elérhető ciklusidejét. Így a hazárdok rossz irányban befolyásolhatják a csővonal áteresztőképességében résztvevő két tényezőt: a ciklusonként végrehajtott utasítások számát és a számítógép ciklusidejét is.

A hazárdok jelenléte jelzi, hogy a csővonal struktúrába jutó parancsok ütemezése ill. rendezése igen nagy fontossággal bír, amikor megpróbáljuk elérni a csővonal hardverének hatékony kihasználását. A hardver hatékony kihasználása viszont teljesítmény növekedésbe megy át. A csővonal ütemezés lényegében egy kisérlet arra nézve, hogy a hazárdok lehetséges elkerülésével megkíséreljük a csővonalat maximálisan kihasználni. Az ütemezést statikusan, dinamikusan vagy a két módszer kombinációjával lehet elérni. A statikus ütemezést az utasítások szekvenciájának, a végrehajtás előtti olyan újrarendezésével érjük el, amelynél az új utasítás-

folyam a hardvert az előzőnél jobban kihasználja. A statikus ütemezésre az I. és II. táblázat nyújt példát, amelyben a két Load (betöltés) utasítás között a blokkolást elkerültük.

I. Táblázat:

```
X1          ;bármilyen utasítás
X2                ;bármilyen utasítás
ADD  R4,R2              ;R4 = R4 + R2
LOAD R1,(Y)             ;R1 betöltése az Y memóriarekeszből
LOAD R1,(X[R1])         ;R1 betöltése az X memóriarekeszből
                         R1 függvényében
ADD  R3,R1              ;R3 = R3 + R1
LCMP       R1,R4        ;az (R4) 2 komplemensének betöltése
                         R1-be
SUB  R1,R2              ;R1 = R1 - R2
COMP R1,R3              ;R1 és R3 összehasonlítása
X3                      ;bármilyen bővíthető utasítás
X4                      ;bármilyen bővíthető utasítás
```

II. Táblázat:

```
X1          ;bármilyen utasítás
X2                      ;bármilyen utasítás
LOAD R1,(Y)             ;R1 betöltése az Y memóriarekeszből
ADD  R4,R2              ;R4 = R4 + R2
LOAD R1,(X[R1])         ;R1 betöltése az X memóriarekeszből
                         R1 függvényében
ADD  R3,R1              ;R3 = R3 + R1
LCMP R1,R4              ;az (R4) 2 komplemensének betöltése
                         R1-be
SUB  R1,R2              ;R1 = R1 - R2
COMP R1,R3              ;R1 és R3 összehasonlítása
```

X3 ;bármilyen bővíthető utasítás

X4 ;bármilyen bővíthető utasítás

Mivel az ütemezési módszerek csak néhány, teljesítmény-javítást eredményező néhány hazárdot szüntetnek meg, az összes hazárd nem szüntethető meg. Az ütemezéssel meg nem szüntethető adat-függőségekre javasoltunk megoldásokat. Ezen javaslatok a többszörös utasításokat párhuzamosan hajtják végre. Az egyik javaslat szerint egy utasításfolyamot a felhasználás szerint analizálják és összetett utasításként csoportba rendezik, hogy egyetlen egységként adhassák ki. Ez a megközelítés különbözik a "szuperskalár géptől", amelyben az egyidejű kiadáshoz az utasításokat szigorúan FIFO alapon csoportosítják. Ha feltételezzük, hogy a kapcsolási elrendezést két utasítás egyidejű kiadásának biztosítására tervezték, akkor egy összetett gép a II. táblázat utasítássorát a következőképpen párosítaná:
(-X1) (X2 LOAD) (ADD LOAD) (ADD LCMP) (SUB COMP) (X3 X4),
miáltal elkerüli a második LOAD utasítás és a második ADD utasítás közötti adatfüggőséget. Egy hasonló szuperskalár gép viszont a következő utasításpárokat adná ki: (X1, X2) (LOAD, ADD) (LOAD, ADD) (LCMP SUB) (COMP X3) (X4 -), viselvén a LOAD-ADD adatfüggőség hátrányát.

A Computer Architecture News 1988. márciusi számában lévő W. A. Wulf által írt "A WM számítógép szerkezet" című cikk az adatfüggőségek kiküszöbölésére egy másik megoldást javasol.

A WM Computer Architecture javaslata:

1. Egy egynél több műveletet magában foglaló utasításkészlet egyetlen utasításba történő szerkesztése
2. Egy megszerkesztett utasításon belüli regiszter blokkolások megengedése és

3. Két ALU 1. ábra szerinti összekapcsolása az egyetlen utasításon belüli blokkolások megszüntese érdekében.

Wulf javaslata szerint minden olyan utasítás-párhoz új utasításokat kell szerkeszteni, amelyek blokkolását meg akarjuk szüntetni. Ez egyrészt az új utasításkészlethez éppen definiált műveletkódok korlátozó számával, másrészt egy határral jár, amely a rendelkezésre álló műveletkódok száma által korlátozva van és azon műveleti sorok számához van hozzáadva, amelyek blokkolásait meg lehet szüntetni. Továbbá ez a séma a korábbi szerkezeti megvalósításokkal esetleg nem tárgykód kompatibilis. Ezen rendszer további hátrányai közé sorolható, hogy szükség van két ALU-ra, amelyek összekapcsolása többszörös műveleti utasítást eredményezhet, amelyhez egy egyedi utasítás végrehajtási idejének közel kétszerese is szükséges lehet. A végrehajtási idő ilyen növekedése a gép ciklusidejének növekedésében is kifejeződhet és szükségtelenül hátráltatná az összes utasításvégrehajtást. Abban az esetben, amikor egy meglévő gépet úgy terveztek, hogy az egy adott utasításkészletet szekvenciálisan ad ki és hajt végre, előnyösebb lenne a párhuzamosság alkalmazása. Az utasítások párhuzamos kiadása megnövelné a gép teljesítményét. Továbbá, az ilyen párhuzamosság előnyeit az utasítás csővonalbeli adatfüggőségi hazárdokkal járó utasítás - végrehajtás lappangás minimalizásával maximálni kell. Így a párhuzamosság alkalmazása az ezen hazárdok miatti blokkolások megszüntetésével az ilyen lappangás csökkentését biztosítaná. Mindamellett ezen előnyöket anélkül élvezhetjük, hogy a meglévő gép szerkezeti változtatásaiból, az összes, blokkolásokkal rendelkező, lehetséges utasítás-párt és ezek kombinációját biztosító új utasításkészlet létrehozásából, valamint nagyszámú hardver hozzáadásából eredő költségeket meg

kellene fizetnünk. Ezenkívül az adaptáció mérsékelt, ill. semmilyen hatást sem gyakorol a számítógép ciklusidejére.

A találmány ezen feladatokat azzal teljesíti, hogy egy skalár műveleti sor soros végrehajtására tervezett számítógéphez egy egyetlen gépi ciklusbeli, számos skalár utasítás párhuzamos végrehajtására szolgáló elrendezést biztosít. Az elrendezés megszünteti az egyidejűleg végrehajtott utasítások közötti adatfüggőséget, ami azt jelenti, hogy egy utasítás-párt még akkor is végre lehet hajtani, ha a pár egyik tagjának a pár másik tagjának végrehajtásával nyert eredményre operandusként van szüksége. Ezen találmányban a számos, skalár utasítás egyidejű végrehajtása alatti adatfüggőség negszüntetésére szolgáló elrendezés magában foglalja a párhuzamosan végrehajtandó, számos, skalár utasítás vételének biztosítását és az ezen utasítások végrehajtási sorrendjére vonatkozó információt, a skalár utasítások második utasítását, amely az első skalár utasítás végrehajtásával létrejött eredményt operandusként használja. Az elrendezés biztosítja továbbá az első és a második skalár utasítás által használt három operandus vételét valamint egy, az utasítások vételét biztosító egységhez csatlakoztatott vezérlő elemet, amely olyan vezérlő jeleket hoz létre, amelyek jelzik azokat a műveleteket, amelyek a skalár utasítások sorát végrehajtják és amelyek jelzik végrehajtásuk sorrendjét. Az operandusokhoz és a vezérlést biztosító egységhez egy többfunkciós ALU csatlakozik, amely a vezérlő jelekre és az operandusokra a második utasítás végrehajtásának megfelelő, egyetlen eredmény létrehozásával, az első utasítás végrehajtásával párhuzamosan reagál.

Másrészt a találmány egy olyan kapcsolási elrendezésre vonatkozik, amely több skalár utasítás egyidejű végrehajtását

teszi lehetővé, ahol az egyidejűleg végrehajtott utasításokból az első által létrehozott eredményt a párhuzamosan végrehajtott utasítások közül a másodikban operandusként használjuk. Az elrendezés a második utasítást az első utasítással párhuzamosan hajtja végre egy adatfüggőséget megszüntető ALU biztosításával, amely az első és a második utasítás által felhasznált, legfeljebb három operandus vételére szolgáló egységgel rendelkezik, amely a második utasítás eredményét az első utasítás eredményével párhuzamosan biztosítja.

Következésképpen, jelen találmány elsődleges célja, hogy egy olyan elrendezést biztosítson, amely a meglévő számítógép teljesítményének növelése érdekében megkönnyíti a párhuzamos utasításvégrehajtást.

Az elrendezés jelentős előnye a végrehajtott utasításokban meglévő, adatfüggőségi hazárdokból származó utasításvégrehajtási késedelem csökkentése.

Ezen elrendezésben a párhuzamosan végrehajtott utasítások között meglévő adatfüggőségi hazárdok miatti blokkolások megszüntetése a cél.

Ezen célokat és előnyöket, egy velejáró teljesítménybeli és utasítás-végrehajtásbeli javulással együtt, egy olyan elrendezéssel érjük el, amely az utasítások soros végrehajtására tervezett skalár számítógéppel kompatibilis.

Ezen és más célok és előnyök elérését szolgáló találmány szerinti megoldást az alábbiakban - hivatkozással az alábbi ábrákra - példaképpeni kiviteli alakok alapján ismertetjük részletesebben, ahol az

1. ábra egy műveleteket párosító utasítás végrehajtására szolgáló régebbi struktúrát mutat be, a

2. ábra egy időzítés szekvencia készlet, amely a skalár utasítások felfűzött végrehajtását illusztrálja, a

3.ábra egy legfeljebb három operandust fogadó és egyetlen eredményt szolgáltató összeadót szemléltet, a

4A és 4B ábra egy meglévő skalár gép által végrehajtott utasítások csoportosítását szemlélteti, az

5. ábra a blokkolási esetek által létrehozott olyan függvényeket szemlélteti, ahol a 4A ábra 1. csoportjának logikai összeadás-típusú utasításait kombináljuk, a

6A és 6B ábra specifikálja azokat a műveleteket, amelyeket a 4A és 4B ábra rendezhető csoportjaiban lévő utasítások biztosításához egy, a találmány szerinti ALU-val, az operandusokon végre kell hajtani, a

7A és 7B ábra a 6A és 6B ábrán megadott ALU-hoz történő operandus-irányítást összegzi, a

8. ábra egy blokkvázlat, amely azt mutatja meg,hogy a találmányt hogyan használjuk két összevont utasítás párhuzamos végrehjtásának megvalósítására, a

9. ábra egy, a 6A, 6B, 7A, 7B ábrával definiált, többfunkciós ALU, a

10.ábra azokat a függvényeket szemlélteti, amelyek a cím képzésben fellépő hazárdokban rejlő blokkolások feloldásához szükségesek, a

11.ábra egy, a 10. ábra szerinti többfunkciós ALU-t szemléltető logikai rajz, a

12.ábra kijelöli azokat a függvényeket, amelyeket egy ALU az összetett elágazási utasításokbeli blokkolások feloldásához biztosít, a

13.ábra egy , a 12. ábra szerinti ALU-t szemléltető logikai

rajz és a

14.ábra egy összegző konfigurációt szemléltet, amely kilenc operandust magában foglaló utasítások blokkolásainak feloldásához szükséges.

Az alábbi leírásban a "gépi ciklus" terminológia egy utasítás végrehajtásához szükséges felfűzési lépésekre vonatkozik. Egy gépi ciklus a felfűzési fokozatoknak megfelelő, egyedi intervallumokat foglalja magában. A "skalár utasítás" egy utasítás, amelyet skalár operandusok felhasználásával hajtanak végre. A skalár operandusok egyértékű mennyiségeket jelentő operandusok. A "rendezés" terminológia egy utasítás-sorban lévő utasítások csoportosítására vonatkozik, ahol a csoportosítás a csoportba rendezett utasítások egyidejű vagy párhuzamos végrehajtása érdekében történik. A rendezés minimálisan két utasítás "párosítását" jelenti az egyidejű végrehajtáshoz. A leírásban szereplő találmányban a rendezett utasítások nem különböznek a skalár utasításként megadott formájuktól. Amint azt az alábbiakban kifejtjük, a rendezett utasításokat "jelzőbitek" kisérik, azaz a rendezett utasításokhoz csatolt bitek jelzik az utasítások csoportosítását a párhuzamos végrehajtáshoz. Igy a bitek egy rendezett utasítás kezdetét és végét jelzik.

Az alábbi fejezetekben egy javított hardver megoldást írunk le, amely azoknak a végrehajtási egység blokkolásoknak megszüntetésére alkalmas, amelyeket a korábbi technikákkal nem lehet megszüntetni.

A cél ezen blokkolások megszüntetéséhez szükséges hardver minimalizálása és az, hogy a kiegészítő hardver csak csekély vagy semmilyen hátrányos befolyást ne gyakoroljon a ciklusidőre. Ezen megoldás megvalósításához semmilyen szerkezeti változtatás

nem szükséges; miáltal a meglévő rendszerrel a tárgykód kompatibilitás fennmarad.

A feltételezett meglévő struktúrát egy olyan sorrendi skalár gép példáján szemléltetjük, mint az International Business Machines Corporation által gyártott és forgalmazott System/370. Ebből a szempontból egy ilyen rendszer a System/370-t, a System/370 bővített struktúrát (370-XA) és a System/370 enterprise-rendszerek struktúrját (370-ESA) foglalhatja magában. Itt adjuk meg a referenciákat: IBM System/370 Principles of operation (Működési elvek), kiadványszáma GA22-7000-10, 1987, és IBM Enterprise Systems Architecture/370 Principles of Operation, kiadványszáma SA-22-7200-0, 1988.

Ezen meglévő, System/370-es skalár struktúrák utasításkészlete jól ismert. Ezen utasítások skalár utasítások, amelyeket skalár operandusokon végzett műveletekkel hajtunk végre. A fentiekben leírt számítógépekkel végrehajtott utasításkészletek konkrét utasításaihoz itt, az alábbiakban megadott referenciákat a szokásos assembly-szintű formában adjuk meg.

Tételezzük fel, hogy az alábbi utasítássort egy ciklusonként négy utasítás végrehajtására képes szuperskalár géppel kell végrehajtani:

(1) LOAD R1,X — X tartalmának betöltése R1-be

(2) ADD R1,R2 — R1 és R2 összeadása, az eredmény R1-be töltése

(3) SUB R1,R3 — R3 kivonása R1-ből, az eredmény R1-be töltése

(4) STORE R1,Y — az eredmény tárolása az Y memóriarekeszben

A ciklusonkénti többszörös utasítás-végrehajtási képesség

ellenére a szuperskalár gép a fenti szekvenciát az utasítás blokkolások miatt sorosan hajtja végre. A program nyomkövetések analízise alapján kiderült, hogy az idő kb. egy harmadában blokkolások lépnek fel. Így a szuperskalár gép sok erőforrása elvész, ami a szuperskalár gép teljesítményének csökkenését okozza. A blokkolt skalár utasítások a szuperskalár gép teljesítményét a 2. ábrán látható, 8 időzítés szekvenciával szemléltetjük. Ezen az ábrán a III. Táblázat utasításainak felfűzési sruktúráját az alábbiak szerint tételezzük fel:

(1) LOAD:ID AG CA PA

(2) és (3) ADD és SUBTRACT:ID EX PA

ahol ID (instruction decode) utasítás dekódolás és regiszter hozzáférés, AG (op address generation) operandus cím létrehozás, CA (cache access) gyorsító hozzáférést, EX (execute) végrehajtást, PA (put away) pedig az eredmény regiszterbe történő írását jelenti. A magyarázat egyszerűsítése érdekében az ebben a leírásban megadott összes példa feltételezi - kivéve ott, ahol kifejezetten állítjuk - , hogy kerülőút nincs megvalósítva. A szuperskalár gépben, a szuperskalár gép teljesítményét egy skaláréra csökkentő utasítás blokkolások miatt, az utasításfolyam végrehajtása sorba van rendezve.

A 2. ábra (2) és (3) utasítása nem igényel cím létrehozást (AG). Mindazonáltal a felfűzésnek ezt a fokozatát is meg kell magyarázni. Innen a 7 és 9 jelöletlen szakasz. Ez a megállapodás a 2. ábra másik három szekvenciájára is fenáll.

A fenti példák azt mutatják, hogy az utasítás blokkolások kényszeríthetik a párhuzamosságot, amely egy szuperskalár gép általi hasznosításhoz az utasítás szinten férhető hozzá. A teljesítményt egy blokkolt utasításnak a másikhoz történő felfűzé-

sével és kerülőúttal lehet növelni; mindazonáltal a blokkolt utasítások végrehajtását sorba kell rendezni.

Ha a blokkolások miatti végrehajtási ciklusok vesztesége elkerülendő, akkor a blokkolt utasításokat "párhuzamosan" kell végrehajtani, és egyetlen utasításnak kell tekinteni. Ez egy összetett blokkolt utasítás koncepciójához, egy skalár utasításkészlethez vezet, amelyet a blokkolások ellenére egyetlen, önálló utasításként kell kezelni. Egy összetett utasítást végrehajtó elrendezés megkívánt jellemzője, hogy végrehajtása nem igényel több ciklust, mint amennyit az összetett utasítások egyike megkíván. Az utasítás összetételének és megkívánt tulajdonságainak következtében egy összetett utasításkészletű gépnek a skalár utasításokat inkább hardver felhasználással, mint műveletkód leírással kell figyelembe vennie.

Az összetett, blokkolt utasítások koncepcióját a III. ábra ADD és SUB utasításainak használatával lehet megvilágítani. Ezt a két utasítást különleges utasításknak tekinthetjük, mert ezek ugyanazt a hardvert használják. Ennek következtében ezeket egy utasításként kombináljuk és hajtjuk végre. A párhuzamosság hasznosításához ezek végrehajtása inkább az

R1 = R1 + R2 - R3

egy ciklusban történő végrehajtását igényli, mint az

R1 = R1 + R2

R1 = R1 - R3

sorrendi végrehajtását, amely egynél több végrehajtási ciklust igényel. A blokkolást ki lehet küszöbölni, mert az összeadás és a kivonás azonos hardvert használ. A CSA, átvitelmegőrző összeadót és a CLA átvitelt előre látó összeadót használó ALU alkalmazásával, amint az a 3. ábrán látható, az R1 + R2 -R3 összetett

utasítást egy ciklusban lehet végrehajtani, feltéve, hogy az ALU-t háromból egy összeadás/kivonás függvényre tervezték.

Amint annak magától értetődőnek kell lenni, az (R1+R2+R3) összetett forma megfelel a második utasítás két operandusának három operandus szempontjából történő újraírásának, ami egy olyan összeadó követelményeit vonja maga után, amely válaszként a három operandusra, végre tudja hajtani a második utasítást.

A 3. ábrán látható, az átvitelt megőrző 10 összeadó (CSA) minden szempontból szokványos; a két eredmény - a 12 kimeneten az összeg (S) és a 14 kimeneten az átvitel (C) - létrehozásához három operandust fogad. A fentiekben megadott példánál a 10 CSA bemeneteire három, az R1, R2 és R3 (komplemens) regiszterben lévő operandus kerül. A 10 CSA kimenetei a 16 és 17-nél kerülnek színre, egy vezető "1"-nek vagy "0"-nak ("forró" "1" vagy "0") az átvitel értékre történő biztosítására, a 20 bemeneten keresztül. A 20 bemeneten az érték a 10 CSA által végrehajtandó függvénynek megfelelően, hagyományosan van beállítva.

A 10 CSA összeg és átvitel kimenetet (a járulélkos 1-gyel vagy 0-val) a 22 átvitelt előre látó összeadó (CLA) két bemeneteként biztosítjuk. A 22 CLA is hagyományosan veszi a 24 bemeneten a "forró" 1-et vagy 0-t, a kívánt üzemmódnak megfelelően, az eredményt pedig a 26 kimeneten hozza létre. A 3. ábrán a 22 CLA által létrehozott eredmény, a három, az R1, R2 és R3 (komplemens) regiszter tartalmának összege.

Az átvitelt megőrző és az átvitelt előre látó összeadók hagyományos alkotóelemek, amelyek felépítése és funkciója jól ismert. Hwang az átvitelt előre látó összeadókat és az átvitelt megőrző összeadókat a COMPUTER ARITMETHIC-ben, 1979-ben megjelent, Principles, Architecture and Design (Elvek, felépítés és

tervezés) című írásának 88-93 ill. 97-100 oldalán írja le.

Annak ellenére, hogy a háromból egy összeadás külön fokozatot igényel - a 3. ábrán látható CSA - az ALU kritikus útjában, egy ilyen fokozatnak nem kell a gép ciklusidejével megegyeznie, mivel az egyéb részek hossza általában meghaladja az ALU-ét. Ezen kritikus részek általában a tömb-hozzáféréssel rendelkező utakban, a cím létrehozásban találhatók, amely egy háromból egy ALU-t és egy morzsa keresztezést (chip-crossing) igényel; miáltal a plusz fokozat késleltetése nem korlátozó, és a javasolt elrendezés a skalár vagy szuperskalár gépekkel összehasonlítva teljesítményjavulást fog eredményezni, A teljesítményjavulást a 2. ábrán a 26-os hivatkozási számmal jelölt felfűzött grafikonkészlettel mutatjuk be. Ezen grafikonok az utasítássor végrehajtását mutatják meg, amit egy, a 3.ábrán látható módon konfigurált összeadóval rendelkező ALU-t magában foglaló, összetett utasításkészletű gép dönt el.

Amint az a 2. ábra 8 és 26 időzítési szekvenciáiból látható, a szekvenciának az összetett utasításkészletű gép általi végrehajtása nyolc ciklust vagy utasításonként két ciklust, CPI, igényel, összehasonlítva a skalár és szuperskalár gépek által elérhető 11 ciklussal vagy a 2,75 CPI-vel. Ha feltételezzük, hogy az kerülőút az összes gépben biztosítandó, a 2. ábra 28 és 30 grafikonkészlete szemlélteti a skalár/szuperskalár gépekkel ill. az összetett utasításkészletű gépekkel elérhető végrehajtást. Ezen készletekből a példa-kód végrehajtásához a szuperskalár gép számára nyolc ciklus vagy két CPI szükséges, míg az összetett utasításkészletű gép csak hat ciklust vagy 1.5 CPI-t igényel. A feltételezett utasítássor mellett az összetett gépnek a szuperskalár és a skalár géppel szembeni előnyét a szuperskalár

gépnek a skalár géppel szembeni előnyének hiányával együtt kell megjegyezni.

Az utasítások összetételének és egyidejű végrehajtásának hardverben történő megvalósítása nem korlátozódik az aritmetikai műveletekre. Például a legtöbb logikai műveletet az aritmetikai műveletekkel analóg módon lehet összevonni. Ezenkívül a legtöbb logikai műveletet aritmetikai műveletekkel is össze lehet vonni. Mindazonáltal néhány utasítás összevonása a ciklusidő megnövekedését okozhatná, mert az elfogadhatatlan késleltetéseknek el kell szenvedniük az összetett függvény végrehajtását. Például egy ADD-SHIFT összetett utasítás a ciklusidőt igencsak megnövelheti, amely viszont veszélyeztetné az egész teljesítményjavulást. Az ezen utasítások közötti blokkolások gyakorisága ugyan csekély a shift utasítások kis előfordulási gyakorisága miatt, így ezeket alapvető teljesítményveszteség nélkül sorosan is végre lehet hajtani.

Amint azt korábban már leírtuk, az adat hazárd blokkolások akkor fordulnak elő, amikor egy regisztert vagy egy memóriarekeszt írunk, és ezután egy ezt követő utasítással olvasunk. Az ezen találmányban javasolt elrendezés megszünteti ezeket a blokkolásokat azáltal, hogy új függvényeket vezet be, amelyek az olyan utasítások végrehajtásának kombinálásából erednek, amelyek operandusai adat-hazárdokat jelentenek, mivel az utasításkészletben lévő függvények végrehajtását visszatartják. Annak ellenére, hogy néhány utasítás- és operandus-kombináció nem várható egy működő programban, mégis minden kobinációt figyelembe veszünk. Általában a fenti analízisből eredő összes függvény és az utasításkészlet skalár megvalósításából keletkező függvények lennének megvalósítva. A gyakorlatban jelentkeznek bizonyos,

olyan függvények is, amelyek megvalósításához az ehhez a berendezéshez javasolt séma nem jól illeszkedik. Az alábbiakban ezt a koncepciót olymódon magyarázzuk meg, hogy bemutatjuk, hogy a két utasítás végrehajtásának kombinációjából hogyan keletkeznek új függvények. A találmánynak megfelelően, jól kezelt utasítássor példákat tárgyalunk olyan sorokkal együtt, amelyek nem jól kezeltek. A találmány javasolt megvalósításának logikai diagramját ábrázoltuk. A találmány szerinti elrendezést az utasítások párhuzamos kiadásának és végrehajtásának megkönnyítésére javasoljuk. Az utasítások párhuzamos kiadására már a korábbi szuperskalár gépben is található példa; az ilyen alkalmazású találmány megkönnyíti a blokkolásokat tartalmazó, kiadott utasítások párhuzamos végrehajtását. Mindazonáltal a találmány adatfüggőség megszüntető hardverének használata nem korlátozódik semmilyen konkrét kiadó és végrehajtó struktúrára, hanem olyan sémákra nyújt általános alkalmazhatóságot, amelyek ciklusonként többszörös utasításokat adnak ki.

Hogy jelenlegi fejtegetésünket hardver síkra tereljük, egy System/370 utasításszint felépítést tételezünk fel, amelyben ciklusonként maximum két utasítást lehet kiadni. Mindazonáltal ezen feltételezések felhasználása ezeket a megfontolásokat nem szűkítik le egy System/370-re vagy egy kétutas párhuzamosságra. A fejezetekre osztott tárgyalásunk kitér az ALU működésekre, a memória cím képzésre és az elágazás (branch) meghatározásra.

Általánosságban a System/370 utasításkészletet utasítások kategóriáira oszthatjuk fel, amelyeket párhuzamosan lehet végrehajtani. Ezen kategóriákon belüli utasításokat kombinálni, vagy rendezni lehet, hogy azok egy összetett utasítást képezzenek. A találmány alábbiakban leírt elrendezése támogatja az összetett

utasítások párhuzamos végrehajtását, és biztosítja, hogy az összetett utasítások tagjai között meglévő blokkolásokat kompenzáljuk, mialatt az utasításokat egyidejűleg végrehajtjuk. Például a System/370 felépítését a 4A és 4B ábrán látható kategóriákra oszthatjuk.

Ezen kategorizálás értelme a System/370 utasításainak funkcionális követelményein és azok harver hasznosításán alapult. A System/370 többi utasítását ebben a leírásban nem tekintjük rendezhetőnek. Ez azonban nem zárja ki azt, hogy egy jövőbeni, összetett utasításvégrehajtó gép összerakja őket, és azt sem, hogy a jelen találmány által bemutatott blokkolás "elkerülés" következtetéseit esetlegesen felhasználják.

Tekintsük az 1. kategóriában lévő utasításokat, amelyek ugyanabban a kategóriában található utasításokkal kerültek összetételre, amint azt az alábbi utasítássorban példaként megadtunk:

AR, R1, R2

SR, R3, R4

Ez a szekvencia, amelyben nincs adat hazárd blokkolás, az

R1 R1+R2

R3 R3+R4

eredményt adja, amely a 370 utasításszint struktúra által specifikált két független utasítást tartalmazza. Egy ilyen szekvencia végrehajtása két, független és párhuzamos, kettőből egy ALU-t igényelne, amelyek az utasításszint strukturához vannak kijelölve. Ezen eredményeket az összes olyan utasítássor-párhoz általánosítani lehet, amelyben nincs adat hazárd blokkolás, és ahol mindkét ALU elegendő ahhoz, hogy a párban kiadott utasításokat végrehajtsa, mivel az egyes utasítások legfeljebb egy ALU művele-

tet határoznak meg.

Mindamellett számos utasítássorban vannak adat hazárd blokkolások. Ezek az adat hazárd blokkolások csővonal buborékokhoz vezetnek, amelyek lerontják egy tipikus csővonal kivitel teljesítményét. A processzor teljesítményének növelése érdekében ezeket a buborékokat egyetlen olyan ALU biztosításával kell kiküszöbölni, amely ezen adat hazárd blokkolásokat képes kompenzálni. Ezen blokkolások kiküszöbölésére az ALU-nak az utasítások párosításából és az operandusok ellentmondásaiból adódó, új függvényeket kell végrehajtania. A keletkező függvények a meghatározott ALU műveletektől, ezen műveletek szekvenciájától és a műveletek közötti operandus "konfliktusoktól" függnek. (Az operandus terminus jelentése a későbbi leírásból világossá válik). Az összes utasítássort, amelyet az ebben a részben korábban már megadott rendezhetőségi listában szereplő és egy ALU műveletet meghatározó utasítás párosításával hozhatjuk létre, ki kell értékelni az összes lehetséges operandus ütközés szempontjából.

A fentiekben a blokkolások találmány szerinti megszüntetéséhez szükséges, általános szerkezetet ismertettük. Az alábbiakban egy blokkolást megszüntető ALU követelményeinek meghatározásában végrehajtandó kiértékelések konkrétabb példáját mutatjuk be. Tételezzünk fel egy, a fentiekben, a 3. ábránál leírtak szerinti három/egy összeadót. Legyen a két végrehajtandó utasítás közül OP1 az első, az OP2 pedig a második. Például a következő utasítássornál

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

OP1 az NR műveletek, míg OP2 az AR műveletnek felel meg (lásd később ezen műveletek leírását). AI0, AI1 és AI2 bemenetek a 3.

ábrán látható háromból egyes összeadónál lévő (R1), (R2) és (R3)-nak felelnek meg.

Tekintsük az (NR, OR, XR, AR, ALR, SLR, SR) utasításkészlet összevonásának analízisét, a 4A, 4B ábrában definiált 1. kategória alkészletét. Ezen utasításkészlet műveleteit az alábbiakkal határozzuk meg:

NR bitenkénti logikai ÉS, jele $\wedge$

OR bitenkénti logikai VAGY, jele $\vee$

XR bitenkénti kizáró VAGY, jele $\oplus$

AR 32-bites előjeles összeadás, jele +

ARL 32-bites előjel nélküli összeadás, jele +

SR 32-bites előjeles kivonás, jele -

SLR 32-bites előjel nélküli kivonás, jele -

Ezt az utasításkészletet további megfontolás céljából két készletre lehet osztani. Az első készlet az NR, OR és SLR logikai utasításokat, a második készlet pedig az AR, ALR, SR és SLR aritmetikai utasításokat tartalmazná. Az aritmetika csoportosítását az alábbiak szerint lehet megítélni. Az AR-t és az ALR-t is egy implicit 33-bites 2 komplemens összeadásnak tekinthetjük azáltal, hogy az előjel kiterjesztést az AR-nál és a 0 kiterjesztést az ALR-nél használjuk és az összeadókhoz "forró" "0"-t vezetünk. Annak ellenére, hogy a feltételkód és a túlcsordulás beállítása mindegyik utasításnál egyedi, az összeadó által végrehajtott művelet, a bináris összeadás, mindkét utasításra közös. Hasonlóképpen az SR és az SLR is egy implicit, 33-bites 2 koplemens összeadásnak tekinthető azáltal, hogy az előjel kiterjesztést az SR-nél, a 0 kiterjesztést pedig az SLR-nél használjuk, ivertáljuk a kivonandót és az összeadóhoz "forró" "1"-et vezetünk. A kivonandó inverzióját az összeadóhoz

képest külsőnek tekintjük. Mivel a négy aritmetikai művelet alapvetően ugyanazt a műveletet, a bináris összeadást, hajtja végre, ezeket ADD-típusú utasításoknak nevezzük, míg a logikai műveleteket LOGICAL-típusú utasításoknak fogjuk nevezni.

A fenti utasításkészlet két műveletre történő csökkentésének eredményeképpen a következő műveleti sort figyelembe kell venni, hogy ezen utasításkészlet összetételét (rendezését) analizáljuk:

LOGICAL, amelyet ADD követ

ADD, amelyet egy LOGICAL követ

LOGICAL, amelyet egy LOGICAL követ

ADD, amelyet egy ADD követ.

Ezen szekvenciák mindegyikéhez a regiszterek összes kombinációját figyelembe kell venni. A kombinációk mindegyike négy regiszter specifikációja, különböző, plusz az utak száma a négy lehetséges regiszter specifikáción kívül, amely:

1) kettő egyforma; 2) három egyforma; 3) négy egyforma. Így a kombinációk számát kifejezhetjük:

$$\text{Kombinációk száma} = 1 + \sum_{i=2}^{4} {}_4C_i,$$

ahol ${}_nC_r$ jelentése "n kombinálva r-rel egyszerre", de ${}_nC_r = n!/((n-r)!r!)$, amely kifejezésekből a kombinációk száma 12-re jön ki. Ez a 12 regiszter kombináció a következő:

1. $R1 \neq R2 \neq R3 \neq R4$

2. $R1 = R2 \neq R3 \neq R4$

3. $R2 = R3 \neq R1 \neq R4$

4. $R2 = R4 \neq R1 \neq R3$

5. $R3 = R4 \neq R1 \neq R2$

6. $R3 = R4 \neq R1 \neq R2$

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

Ezen kombinációk közül csak a héttől tizenkettőig terjedők okoznak adat-függőségi blokkolásokat. A korábbiakban felsorolt LOGICAL-ADD szekvenciák fenti blokkolási esetei által létrehozott függvényeket az 5. ábrán adtuk meg. Ezen az ábrán a LOGICAL-típusú műveleteket §-vel az ADD-típusú műveleteket pedig ç-vel jelöltük.

Míg az 5. ábra azokat a műveleteket határozza meg, amelyeket az ADD-típusú és a LOGICAL-típusú utasítások operandusain kell végrehajtani, hogy a blokkolásokat megszüntessük, a 6A és 6B ábra az AI0, AI1 és AI2 ALU bemeneteken végrehajtandó ALU műveleteket határozza meg, hogy a 4A és 4B ábra rendezhető kategóriáiban lévő, mind a 370 utasítást biztosítsa. A 6A és 6B ábrán egy egyalkotós - a 2 komplemest, míg |x|, az x abszolút értékét jelzi. Ez az ábra egy, a fentiekben megadottal azonos analízisből származik; mindamellett az összes lehetséges kategória kombinációt figyelembe vettük. Az 5. ábrának az ALU által végrehajtandó műveleteinél a végrehajtó egység vezérlőinek a kívánt regiszter-tartalmakat az ALU megfelelő bemeneteire kell irányítania. A 7A és 7B ábra azon operandusok irányítását foglalja össze, amelyeknek a meghatározott ALU-nál elő kell fordulniuk, úgy mint a 6A és 6B ábrában, hogy az 5. ábra műveleteit végrehajtsák. Ezen irányításokkal együtt megadtuk a LOGICAL és ADD-típusú utasításokat, hogy a 6A és 6B ábrához ezen eredmények iránykijelölését megkön-

nyítsük. Néhány ADD-ADD összetétel irányítását nem vettük bele, mivel ezek a műveletek egy négy bemenetű ALU-t igényelnek (lásd a "sajátságokat"), ezért így jegyeztük fel őket.

Míg a leírás elsősorban négy darab, speciálisan számba vett R1, R2, R3, R4 összetett utasítás visgálatára irányul, magától értetődő, hogy a találmány gyakorlati felhasználása nem korlátozódik semmilyen négy speciális regiszterre. Vagy méginkább, ezen megjelölések kiválasztása csupán az analízis és az érthetővé tétel segédeszköze. Nyilvánvaló továbbá, hogy az analízist általánosítani lehet, amint azt a fentiekben megadott egyenletekből következik.

A 8. ábrán, az 5, 6A, 6B, 7A, 7B ábrán fent elvében ismertetett többfunkciós ALU-t megvalósító elrendezést ábrázoló logikai blokkvázlatot mutatunk be. A 8. ábrán az 50 regiszter veszi az 52 és 54 utasítást magában foglaló összetett utasítást. Az összetett utasítások az 56 és 58 csatolt jelzőbittel rendelkeznek. Az utasításokat és jelzőbitjeiket a 60 logika dekódolásához és vezérléséhez biztosítjuk, amely az utasításokat és a jelzőbitjeikben lévő információt dekódolja, a 62 kimenet regiszter kiválasztó jeleket ill. a 66 kimeneten a funkció kiválasztó jeleket biztosítása érdekében. A 62 kimeneten a regiszter kiválasztó jelek a 64 átkötő elemet alakítanak ki, amely a 63 általános célú regiszterhez csatlakozik, hogy a 65 adatfüggőséget megszüntető ALU AI0, AI1, AI2 három operandus bemenetéhez a maximum három regiszter tartalmát biztosítsa. A 65 ALU egy többfunkciós ALU, amelynek függvényviszonyát a 60 dekódoló és vezérlő logika 66 kimenetén előálló függvénykiválasztó jelek határoznak meg. A 65 ALU a 64 átkötőn át csatlakoztatott regiszterekből biztosított operandusokkal végre fogja hajtani a függvény

kiválasztó jelek által jelzett függvényeket, az eredményt pedig a 67 kimenetre adja.

A fentiekben leírt ALU berendezéssel párhuzamosan egy második ALU berendezés is elhelyezkedik, amely magában foglalja a 70 dekódoló és vezérlő logikát, amely az 52 utasításmező első utasítását dekódolja, és biztosítja a 72 hagyományos átkötőhöz a regiszter kiválasztó jeleket, amely szintén a 63 általálnos célú regiszterekhez csatlakozik. A 70 logika is függvény-kiválasztó jeleket szolgáltat a 74 kimeneten a hagyományos, kétoperandusos 75 ALU számára. Ezt az ALU berendezést az 52 utasításmezőbeli utasítás végrehajtására használjuk, míg az 54 utasításmezőben lévő második utasítást a 65 ALU hajtja végre. Amint azt az alábbiakban bemutatjuk, a 65 ALU egy második utasítást is végre tud hajtani tekintet nélkül arra, hogy egy operandusa függ-e az első utasítás végrehajtásának eredményétől. Így a két ALU párhuzamosan működik és a két utasítás párhuzamos végrehajtása az összetételtől függetlenül biztosított.

Visszatérve az 52 és 54 összetett utasításhoz és az 50 regiszterhez, feltételezzük egy összeadó meglétét. Azt állítjuk, hogy az összetevő párosítja ill. összerakja az utasításfolyam utasításait, amely annak a skalár számítógépnek a bemenetéhez menő skalár utasítássort foglalja magában, amelyben az összerakó elhelyezkedik. Az összerakó az utasításokat a fenti fejtegetés szerint csoportosítja. Például az 1. kategóriájú utasításokat (5. ábra) az 5. táblázat szerinti logical/add, add/logical, logical/logical és add/add párok szerint csoportosítja.

Egy rendezett készlet mindegyik utasításához egy vezérlő információt tartalmazó kiegészítő cimkét adunk. A cimke azokat az összetételi biteket tartalmazza, amelyek egy, csak az összetett

utasítások csoportjainak azonosításához használt, cimke részre vonatkoznak. Két utasítás összevonásakor, annak jelzésére, hogy az összetétel hol fordul elő, lehetőleg a következő eljárást használjuk. A System/370 gépeknél az összes utasítás a fél-szó határon van rendezve, függetlenül attól, hogy hosszuk 2, 4 vagy 6 bájt. Ebben az esetben mindegyik fél-szóhoz egy összetételi cimke szükséges. Egy egy-bites cimke elegendő ahhoz, hogy jelezze, hogy egy utasítás összetett-e vagy sem. Lehetőleg "1" jelezze, hogy egy vizsgálat alatt lévő bájttal kezdődő utasítás a következő utasítással össze van vonva. A "0" azt jelzi, hogy nincs összevonás. Azt az összevonás bitet, amely olyan fél-szavakhoz kapcsolódik, amelyek nem tartalmazzák egy utasítás első bájtját, nem vesszük figyelembe. Az összetett pár második utasítás első bájtjának összetételi bitjét szintén figyelmen kívül hagyjuk. Ennnek megfelelően csak egyetlen bitnyi információ szükséges az összetett utasítások azonosításához és pontos végrehajtásához. Így az 56 és 58 cimke bit elegendő ahhoz, hogy a 60 dekódoló és vezérlő logikát informáljuk arról, hogy az 52 és 54 regiszter mezőben lévő utasításokat kombinálni kell, azaz párhuzamosan kell végrehajtani. Ezután a 60 dekódoló és vezérlő logika ellenőrzi az 52 és 54 utasítást, hogy meghatározza azok végrehajtási sorrendjét, blokkolási feltételeit - amennyiben talál ilyet - valamint a szükséges függvényeket. Ezt a meghatározást az 5. ábra 1. kategória utasításaira szemléltetjük. A dekódoló és vezérlő logika azokat a függvényeket is meghatározza, amelyek a 6A, 6B ábra bármelyik adat hazárd blokkolás megszüntetéséhez szükségesek. Ezeket a 7A, 7B ábrában vonjuk össze. A 7A, 7B ábrán - feltételezvén, hogy a 60 dekódoló és vezérlő logika a címkebitekből meghatározta, hogy az 52 és 54 mezőben lévő utasításokat

rendezni kell - a 60 logika a 66 kimeneten a kívánt műveletet jelző függvényt kiválasztó jelet bocsát ki, a 7A ábra bal szélső oszlopa szerint. Az utasítások operációs kódjait félreértést kizáró módon dekódoljuk, hogy a függvény kiválasztó kimeneten a 7A, 7B ábra OP1, OP2 fejlécű oszlopaiban lévő speciális függvényeket biztosítsuk. A 62 kimeneten a regiszter kiválasztó jelek, a 64 átkötő segítségével, a 8. ábra regisztereit irányítják, amint azt a 7A, 7B ábra AI0, AI1, AI2 oszlopaiban megköveteljük. Így például tételezzük fel, hogy az 52 mezőben az első utasítás ADD R1, R2, a második utasítás pedig ADD R1, R4. A 7A ábra 18. sora megmutatja azokat az ALU műveleteket, amelyeket a dekódoló és vezérlő áramkör OP1 = + -szal és OP2 = + -szal jelez, míg az R2 regiszter az AI0 bemenetre, az R4 regiszter az AI1 bemenetre, az R1 regiszter pedig az AI2 bemenetre van irányítva.

A 65 ALU adatfüggőség megszüntető felépítésének és működésének megértéséhez most tekintsük a 9. ábrát. A 9. ábrán egy, a 3. ábra összeadójának megfelelő, három operandusú, egy eredményű, 70 összeadó látható. A 70 összeadó bemeneteit az összeadó bemenetei és az ALU AI0, AI1 és AI2 bemenetei közé csatlakoztatott áramkörön keresztül kapja. Az AI2 bemenettől egy operandus három logikai funkcionális elemen, a 71 logikai AND a 72 logikai OR valamint a 73 logikai EXCLUSIVE-OR (kizáró vagy) kapun megy keresztül. Ez az operandus ezekben a logikai elemekben a további operandusok egyikével összeadódik és az AI0-ra vagy az AI1-re kerül, a 80 multiplexer beállításának megfelelően. A 75 multiplexer vagy az AI2-höz csatlakoztatott változatlan operandust, vagy a 71, 72 ill. 73 logikai elem egyikének kimenetét választja ki. A 75 multiplexer által kiválasztott bemenetet a 77 inverterre adjuk, a 78 multiplexer pedig a 70 összeadó egy beme-

netéhez vagy a 77 inverter kimenetét, vagy a 75 multiplexer nem-invertált kimenetét csatlakoztatja. A multiplexer kimenete a 84 inverteren át invertálódik, a 85 multiplexer pedig a 70 összeadó második ooperandus bemeneteként a 82 multiplexernek vagy a nem-invertált vagy az invertált kimenetét választja ki. A 70 összeadó harmadik bemenetét az AI0 bemenetből nyerjük, amelyet a 87 inverter invertál. A 88 multiplexer vagy "0"-t - az AI0 operandus bemenetét - vagy az inverzét választja ki, amelyet a 70 összeadó harmadik bemeneteként használunk. Az ALU kimenetét a 95 multiplexer kimenetén át nyerjük, amely a 70 összeadó kimenetét vagy pedig a 90, 92, 93 logikai elemek egyikének kimenetét választja ki. A 90, 92, 93 logikai elem a feltüntetett logikai művelet segítségével az összeadó kimenetét az AI1 operandus bemenettel összegzi.

Magától értetődőnek kell lennie, hogy a függvény kiválasztó jel alapvetően az A B C D E F G multiplexer kiválasztó jelből és a 70 összeadó "forró" 1/0 kiválasztás bemenetből áll. Nyilvánvalóvá válik, hogy a multiplexer kiválasztó jelei az A, B, E, F jelek egyetlen bitjétől a C, D, G kétbites jeleiig terjednek. A komplex vezérlő jelek állapotait (A B C D E F G 1/0 1/0) könnyen származtathatjuk a 7A, 7B ábrából. Például a fentiekben megadott ADD R1, R2 ADD R1, R4 példát végigkövetve az OP1 jel a C multiplexer jelet az AI2-n lévő jel kiválasztásához állítaná be, míg az F jel a 75 multiplexer nem-invertált kimenetét választaná ki, miáltal a 70 összeadó jobbszélső bemenetéhez az R1 operandust biztosítaná. Hasonlóképpen a B, E multiplexer jeleket is beállítanánk, hogy az AI1-nél elérhető operandust nem-invertált formában a 70 összeadó középső bemenetére vezessük, míg a D multiplexer jelet úgy állítanánk be, hogy a 70 összeadó balszélső

bemenetére az AI0 operandust invertálásmentesen alkalmazzuk. Végezetül a két "I/O" bemenetet a két összeadási műveletnek megfelelően állítjuk be. Ezen bemenetekkel a 70 összeadó kimenete egyszerűen a három operandus összege, amely megfelel az ALU kívánt kimenetének. Ennélfogva a G vezérlő jelet úgy állítanánk be, hogy a 95 multiplexer a 70 összeadó által szolgáltatott eredményt adná ki, amely az R1, R2, R3 regiszterben lévő operandusok összege lenne.

Egy logikai összeadás szekvencia alatt a logikai funkciót a 75 multiplexer választaná ki, és a 78 multiplexeren át a 70 összeadóra kerülne, mialatt a logikai művelethez az összeadandó operandust a 85 ill. 88 multiplexer közül az egyiken keresztül a 70 összeadó további bemeneteinek egyikére vezetnénk, egy 0-val együtt, amelyet a harmadik bemenetre juttatnánk. Ebben az esetben a 95 multiplexer a 70 összeadó kimenetének, mint eredménynek a kiválasztására lenne beállítva.

Végül, egy összetett add/logical szekvenciában az elsőként összeadandó két operandust a 70 összeadó két bemenetére adjuk, míg a harmadik bemenetre 0-t vezetünk. Az összeadó kimenetét azonnal összeadjuk a 90, 92, 93 logikai elemek ki nem választott operandusaival. A G vezérlő jelet azon elemek kimeneteinek kiválasztására állítjuk be, amelyek működése megfelel a rendezési készlet második utasításának.

Általánosabban, a 9. ábra a 65 adatfüggőség megszüntető ALU logikai bemutatását nyújtja. Ezen adatfolyamból eredően úgy határoztunk, hogy azokat a blokkolásokat nem tartjuk meg, amelyekben az első utasítás eredményét a második utasítás operandusaként is használjuk. Ennek részleteit a "sajátságok" című bekezdésben találhatjuk meg. Azt, hogy ez a megvalósítás javítja a

LOGICAL-ADD összetételek által megkívánt többi műveletet is, az adatfolyamnak és az 5. ábra függvényoszlopának összehasonlításából láthatjuk. Ebben az oszlopban egy két operandusos LOGICAL-típusú utasítást egy, a LOGICAL eredmény és egy harmadik operandus közötti ADD-típusú utasítás követi. Ezt az operandusoknak a 9. ábra AI0, AI2-vel történő logikai összegzésével és a 71, 72 ill. 73 logikai blokk közül a megfelelővel érjük el úgy, hogy ezt az eredményt a 70 összeadóra adjuk, a harmadik operandust pedig az AI1-en keresztül irányítjuk az összeadóra. Az invertálásokat valamint a "forró" 1-ek ill. 0-k biztosítását a specifikált aritmetikai művelet által megkívánt módon, a függvény kiválasztó jel részeként biztosítjuk. Más esetekben egy két operandus közötti ADD-típusú műveletet egy, az ADD-típusú művelet eredménye és egy harmadik operandus között lévő LOGICAL-típusú művelet követi. Ezt az ADD-típusú műveletek operandusainak az AI0-, AI2-höz történő irányításával, ezen bemeneteknek az összeadóra való eljuttatásával, az összeadó kimeneteinek a frissítő összeadó 90, 92, 93 logikai blokkra juttatásával és a harmadik operandusnak az AI3-on át, az ezen logikai blokkokhoz történő irányításával hajtjuk végre. Az olyan utasítást, amelyben egy LOGICAL-típusú utasítást LOGICAL-típusú utasítások követnek, az első LOGICAL-típusú utasítás két operandusának az AI0-, AI2-höz történő irányításával, amelyeket az elő-összeadó logikai blokkhoz irányítunk, az elő-összeadó logikai blokk eredményeinek az ALU-n át történő, módosítás nélküli, nullához való adásával, a frissítő összeadó logikai blokkhoz történő irányításával és a harmadik operandusnak, az AI3-n át, a frissítő logikai blokkhoz történő irányításával hajtjuk végre. Az olyan utasításoknál, amelyeknél egy ADD-típusú műveletet egy ADD-típusú művelet követ, a három

operandust az összeadó bemenetére adjuk, míg az összeadó kimenetét az ALU kimenetére vezetjük.

A 65 ALU feladata, hogy az 54 utasításmező második utasítását végrehajtsa, - amikor az első és a második utasítás között nincs adatfüggőség - egyértelmű. Ebben az esetben az ALU-ra csak a két operandust adjuk. Ennélfogva, ha a második utasítás "add" utasítás, a két operandust egy nullával - amit a harmadik operandus helyett adunk - és a 95 multiplexeren át, az ALU kimeneteként éppen kiválasztott összeadó kimenettel egyetemben a 70 összeadóra adjuk. Ha a második utasítás "logical" utasítás, a logikai műveletet a két operandusnak a 71, 72, 73 logikai elemre történő irányításával, a megfelelő kimenet kiválasztásával, majd a másik két bemenetre történő nullák biztosításával, az eredménynek a 70 összeadón át történő átvitelével hajtjuk végre. Ebben az esetben az összeadó kimenete egyenlő lenne a logikai eredménnyel, és a 95 multiplexer az ALU kimeneteként választaná ki. Vagy pedig egy operandust két nulla hozzáadásával át lehet juttatni az összeadón, amely a 70 összeadóban ezt az operandust kimenetként biztosítja. Ezt az operandust a 90, 92, 93 logikai elemben lévő másik operandussal összegezzük úgy, hogy a 95 multiplexer által kiválasztott megfelelő logikai elem kimenet legyen az ALU kimenete.

Amikor a 8. ábrán megadottak szerint utasításokat randezünk, függetlenül attól, hogy létezik-e függőség, az 50 regiszter 52 utasításmezejének utasítását a 70, 74 utasítás dekódolásával, az operandusainak a 70, 71, 72 által történő kiválasztásával és a 75 ALU-ban a kiválasztott operandusokon a kiválasztott művelet elvégzésével a szokásoknak megfelelően végrehajtjuk. Mivel a 75 ALU-t egyetlen utasítás végrehajtására használjuk, az AI0, AI1

kimeneten át a kiválasztott regiszterből két operandust biztosítunk úgy, hogy a jelzett eredményt a 77 kimeneten állítjuk elő.

Így a 8. ábrán bemutatott elrendezéssel, az adatfüggőséget megszüntető 65 ALU, a hagyományos 75 ALU-val kombinálva segíti két utasítás egyidejű (vagy párhuzamos) végrehajtását, még akkor is, amikor az utasítások között adatfüüggőség áll fenn.

Az adat hazárdok befolyásolhatják a cím generálást is; ezekre cím hazárdként (AHAZ) hivatkozunk. A következő szekvencia egy cím hazárdtól mentes összetett System/370 szekvenciát mutat be:

```
AR R1, R2
S R3,D(R4,R5)
```

ahol D három, négybites relatív címet jelent. AHAZ nincs jelen, mivel a cím számításánál használt R4-et és R5-öt az előző utasítással nem változtattuk meg. A következő szekvenciákban adat hazárdok nem lépnek fel:

```
AR R1, R2                    AR R1,R2
S R3, D(R1,R5)               S R3,D(R4,R1)
```

A fenti szekvenciák egy RR utasítás (5. ábra 1 kategória) és AHAZ-t mutató RX utasítások (9 kategória) összetételét mutatják be. Más kombinációk RS és SI utasításokkal összetett RR utasításokat tartalmaznak.

Egy blokkolást megszüntető ALU-nál az AHAZ blokkolások megszüntetéséből eredő új műveleteket az utasítás szekvenciák összes kombinációjának, valamint a cím- és operandus ütközések analíziséből kell származtatni. Az analízis megmutatja, hogy az olyan közös blokkolásokat, mint amilyeneket a fenti utasítás szekvenciák tartalmaznak, egy négyből egyes ALU-val meg lehet szüntetni.

A 10. ábrán azokat a függvényeket soroltuk fel, amelyeket egy ALU-nak támogatnia kell ahhoz, hogy egy System/370 utasítás-szintű struktúrájának összes AHAZ blokkolását megszüntesse. Azokra az esetekre, ahol a négy bemenetet nem specifikálják, implicit nullát kell biztosítani. A 10. ábrán definiált AHAZ blokkolást megszüntető ALU logikai rajza a 11. ábrán látható. A szemléltetett ALU-val a 10.ábrán specifikált függvényeknek egy nagy, de nem teljes alkészletét támogatjuk. Ezen alkészlet a 10. ábra 1-21 sorában megadott függvényekből áll. A döntés, hogy melyik függvényt tartalmazza, egy megvalósítási döntés, amelynek tárgyalását a "Sajátságok" című fejezetre halasztjuk. Amint azt a 11. ábrán láthatjuk, a bemutatott ALU magában foglalja a 100 összeadót, amelyben a három bemenetű, két kimenetű átvitel megőrző 101, 102 összeadó kaszkádba van kapcsolva a két bemenetű, egy kimenetű, az átvitelt előre látó 103 összeadóval olymódon, hogy a 100 összeadó egy, a 11. ábrán látható ALU működéséhez szükséges, négy bemenetű, egyetlen kimenettel rendelkező összeadóként működik.

A 10. ábra létrehozásakor az ALU bonyolult szerkezetét a vezérlő logika rovására egyszerűsítettük. Ezt a legegyszerűbb egy példán megmagyarázni. Tekintsük az alábbi két System/370 utasítás szekvenciát:

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

és

```
NR R1, R2                    (5)
S R3, D (R4, R1).
```

Legyen erre a szekvenciára az általános jelölés

```
NR r1, r2
```

S r3,D(R4,R5).

Az első szekvenciára az operandus címe:

OA = D + (R1∧R2) + 5

míg a második szekvenciára:

OA = D+ R4 (R1∧R2)

A végrehajtás vezérlők egyszerűsítéséhez az ALU bonyolultságának rovására, az ALU-nak az alábbi két műveletet kell végrehajtania:

OA = AG10 + (AG11 ∧ AG12) + AG13

OA = AG10 + AG12 + (AG11 ∧ AG13)

ahol D-t az AGI0-ba, r2-t az AGI1-be, r4-et az AGI2-be, r5-öt pedig az AGI3-ba tápláljuk. Mindazonáltal az ALU-t egyszerűsíteni lehetne, ha a vezérlők érzékelnék, hogy az r4, r5 közül melyiknek van az r1-gyel hazárdja, és ezt a regisztert dinamikusan az AGI2-höz irányítanák. A másik regisztert az AGI3-ra kell adni. Ezen feltételezéshez az ALU-nak csak támogatnia kell az

OA = AG10 + (AG11 ∧ AG12) + AG13

műveletet.

Az ilyen váltott használatokat a címgenerációs ALU valamint a végrehajtó és ugrás meghatározó ALU-k bonyolultságának csökkentése érdekében végezzük.

A 11. ábrán látható ALU-t a 8. ábrabeli 65 ALU helyettesítheti. Ebben az esetben a 60 dekódoló és vezérlő logika a 10. ábra függvényeit megfelelően tükrözné.

A blokkolást megszüntető ALU analíziseihez hasonlókat kell elvégezni a végrehajtásra és a cím generálásra is, hogy a 12. és 13. ábrán megadott elágazás meghatározó ALU összetételének hatásait megkapjuk. Az elágazás meghatározó ALU a regiszter értékeket összehasonlító utasításokhoz szükséges funkciókat

öleli fel. Ez magában foglalja a BXLE, BXH, BCT és BCTR ugrás utasítást, amelyben egy regiszter érték egy második regiszter (BXLE és BXH) tartalmával inkrementálódik, vagy eggyel dekrementálódik (BCT és BCTR), mielőtt egy regiszter értékkel összehasonlítódik (BXLE és BXH) vagy nullával, hogy az ugrás eredményét meghatározzuk. Ez az ALU nem hajt végre feltételes ugrásokat.

A 13. ábrán látható ALU a 110 többfokozatú összeadót foglalja magában, amelyben a 111, 112 átvitel megőrző összeadó kaszkádba van kapcsolva a 112 átvitel megőrző két kimenetével, amely a 113 átvitelt előre látó összeadó két bemenetét adja. Ez az összegzés hatásosan biztosítja a 13. ábra ALU-jához csatlakoztatott négy-bemenetű, egyetlen kimenetű összeadót.

Az előforduló adat hazárdokra példaként nézzük meg az alábbi utasítás szekvenciát:

AL R1,D (R2, R3)

BCT R1,D (R2, R3)

Jelölje az x memóriarekesz tartalmát [x]. A végrehajtást követő eredmények:

R1 = R1 + [D + R2 +R3] - 1

elágazás, ha (R1 + [D +R2 +R3]) - 1 = 0

Ezt az összehasonlítást az alábbi művelet végrehajtásával lehet megtenni:

R1 + [D + R2 + R3] - 1 - 0.

Az elágazást meghatározó ALU analíziseinek eredményeit minden további tárgyalás nélkül a 12. és 13. ábrán adjuk meg. Az adatfolyam által támogatott függvények a 12. ábra 1 - 25. sorában specifikáltakat foglalják magukban.

A 13. ábrán látható ALU-t a 8. ábra szerinti 65 ALU helyettesítheti. Ebben az esetben a 60 dekódoló és vezérlő logika a 12.

ábra funkcióit megfelelően tükrözi.

Az operandus ütközésekből adódó függvények közül néhány bonyolultabb a többinél. Például az alábbi utasítás szekvencia, azaz az:

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

egy négyből egyes ALU-t igényel, az ezzel járó bonyolultsággal együtt, az adatblokkolás megszüntetésére, mivel végrehajtásának eredménye:

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

Más szekvenciák olyan műveletekkel járnak, amelyek az ALU-ban foglalandó további késleltetéssel járnak, hogy a blokkolást megszüntessék. Egy a megnövekedett késleltetést ábrázoló szekvencia:

```
SR  R1, R2
LPR R1,R1
```

amely az alábbi műveletet eredményezi:

R1 = /R1 - R2/.

Ez a művelet nem alkalmas a párhuzamos végrehajtásra, mert a kivonás eredményeihez az abszolút érték felállítása szükséges.

Az ALU-beli összes blokkolás megszüntetése helyett inkább egy utasítás-kiadó logikát, vagy egy elő-feldolgozót lehet alkalmazni, amelyet azon utasítás szekvenciák detektálására terveztünk, amelyek ezekhez a bonyolultabb függvényekhez vezetnek. Az előfeldolgozó detektálás elkerüli a gyakran közel kritikus utat képező kiadó logika további késleltetését. Amikor egy ilyen szekvenciát észlelünk, a kiadó logika vagy az előfeldolgozó visszatérne a szekvencia skalár módban történő kiadásához, elkerülvén a blokkolás megszüntetésének szükségességét. Annak el-

határozása, hogy melyik utasítás szekvenciának kell ill. nem kell a blokkolásait megszüntetni, megvalósítási kérdés, amely ezen találmány körén kívül eső tényezőktől függ.

Mindazonáltal meg kell jegyezni az ALU megvalósításának bonyolultsága és a kiadó logika komplexitása közötti helycserét.

A cím generálásban lévő hazárdok is megvalósítási cseréket okoznak. Például a legtöbb cím generálási blokkolást egy, a fentiekben tárgyalt négyből egyes ALU alkalmazásával meg lehet szüntetni. Ámbár a következő szekvencia nem illeszkedik ehhez a kategóriához:

```
AR R1,R2
S  R3,D(R1,R1);
```

Ebben az esetben az AHAZ blokkolások megszüntetéséhez egy ötből egyes ALU szükséges, mert a végső művelet:

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

ahol OA az eredmény operandus címe. Ahogyan az előbbiekben is, az, hogy az ALU tartalmazza-e ezt a függvényt, az megvalósítási elhatározás, amely az ilyen blokkolások előfordulási gyakoriságától függ. Az eredmények az ugrás meghatározó ALU-ra is érvényesek.

A legáltalánosabb n blokkolási eset blokkolást feloldó hardverének származtatásához a bemutatott analízisekhez hasonlókat hajthatunk végre. Ehhez a tárgyaláshoz lásd a 14. ábrát. Tételezzük fel az alábbi egyszerű adat blokkolásokat:

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

ahol a megváltoztatott regisztert az első utasítástól a második utasításnak csak egy operandusaként használjuk, a (n +1) db ALU lenne szükséges a blokkolás feloldásához. Például három blokkolás

megszüntetéséhez, a fenti feltételezést alkalmazva, egy négyből egyes ALU-ra lenne szükség. Ez még egy további CSA fokozatot is igényelne az ALU-ban.

Mindazonáltal az ALU-ban szükséges CSA fokozatok számának növekedése nem lineáris. Egy kilenc operandus kezelésére tervezett ALU, mint egy egyedi végrehajtó egység, négy CSA fokozatot és egy CLA fokozatot igényelne. Ezt a 14. ábrából lehet látni, amelyen minden függőleges vonal egy összeadó bemenetet és minden vízszintes vonal egy összeadót jelent. Az átvitelt megőrző összeadókat a 100-106 vízszintes vonalakkal, míg az átvitelt előre látó összeadót a 109 vonallal jelöltük. Minden egyes CSA összeadó két kimenetet hoz létre három bemenetből. A kimenetek csökkenése fokozatról fokozatra folytatódik, míg az utolsó CSA kettőre nem csökkenti. A következő összeadó egy CLA, amely két bemenetből egy végső kimenetet képez. Csak aritmetikai műveleteket feltételezve, egy egyfokozatú CLA összeadót és egynégyfokozatú CSA összeadót, a javasolt elrendezést alkalmazó egyetlen egységben nyolc operandus végrehajtását lehet megvalósítani, egy elsőfokú közelítéshez, ugyanannyi idő alatt, mint a fent idézett referenciában a Wulf által javasolt megoldásnál.

Az adat hazárd blokkolások lerontják a felfűzött utalításokkal dolgozó számítógépekből nyert teljesítményt azáltal, hogy a felfűzésbe rögzített helyzeteket visznek be. Ezen blokkolások közül néhányat kód mozgatással és utasítás ütemezéssel fel lehet oldani. A teljesítmény leromlás csökkentésére egy másik javaslat, hogy adat blokkolásokat kezelő utasításokat kell definiálni. Ez a javaslat azonban azon blokkolások számával van korlátozva, amelyeket még egy ésszerű utasításmérettel kezelni lehet. Ezenfelül ez a megoldás nem hozzáférhető a 370 felépítéssel csereszaba-

tos számítógépeknél.

Jelen találmányban az utasítás blokkolások feloldására egy alternatív megoldást mutattunk be. A találmány azzal az előnnyel jár, hogy nem igényel szerkezeti változtatásokat, nem kell hozzá az összes lehetséges utasítás-párt és azok blokkolását egy utasításkészletbe struktrálni, a gép ciklusidejére csak csekély vagy semmilyen hatással nem bír, kevesebb hardvert igényel, mint az 1. ábrán látható korábbi megoldások, és a System/370 felépítésű számítógépekkel kompatibilis.

Mivel a találmányt részletesen bemutattuk és előnyben részesített megvalósítására való hivatkozással részletesen leírtuk, a szakember számára érthető lesz, hogy számos formai és a részleteket érintő változtatást lehet benne eszközölni anélkül, hogy a találmány lényegétől és tárgyától eltérnénk.

SZABADALMI IGÉNYPONTOK

1. Kapcsolási elrendezés több skalár utasítás párhuzamos végrehajtásának támogatására egy a skalár utasítás szekvenciák soros végrehajtására strukturált számítógépben, a z z a l j e l l e m e z v e, hogy tartalmaz

egy utasítás tárolót a skalár utasítások vételére, ahol az első skalár utsítás által létrehozott eredményt a második skalár utasítás operandusként használja;

egy operandus tárolót a skalár utasítások vételére, az első és a második skalár utasítás a fenti operandusok közül legalább hármat használ;

egy, az utasítás tárolóhoz csatlakoztatott vezérlő egységet a vezérlő jelek létrehozására, hogy jelezze azokat a műveleteket, amelyek végrehajtják a skalár utasításokat; és

egy, az operandus tárolóhoz és a vezérlő egységhez csatlakoztatott végrehajtó eszközt, amely a vezérlő jelekre és az operandusokra reagál, hogy a fenti számos operanduson történő, a fenti műveletek végrehajtásának megfelelő, egyetlen eredményt létrehozza.

2. Az 1. igénypont kapcsolási elrendezése, azzal jellemezve, hogy a végrehajtó eszköz tartalmaz egy összeadót, amely három operandusra válaszként egyetlen összegjelet ad.

3. Az 1. vagy 2. igénypont szerinti kapcsolási elrendezés, azzal jellemezve, hogy az összeadó egy átvitelt megőrző összeadót tartalmaz, amely a három operandusra válaszként két kimenetet ad, valamint egy az átvitelt megőrző összeadóhoz

csatlakoztatott, az átvitelt előre látó összeadót, amely az átvitelt megőrző összeadó két kimenetére válaszként egy kimenetet ad.

4. Az 1 - 3. igénypontok bármelyike szerinti kapcsolási elrendezése, azzal jellemezve, hogy a végrehajtó eszköz magában foglal még az operandus tárolóhoz és az összeadóhoz csatlakoztatott logikai elemeket is az operandusokon egy logikai függvény végrehajtására, amely egy logikai eredményt ad, az összeadó pedig a fenti egyetlen összeget adja eredményül, válaszként a logikai eredményre és az egyik operandusra.

5. Az 1 - 4. igénypontok bármelyike szerinti kapcsolási elrendezés, azzal jellemezve, hogy a végrehajtó eszköz magában foglal még az operandus eszközhöz és az összeadóhoz csatlakoztatott logikai eszközöket is, hogy az első és második operanduson egy logikai függvényt hajtson végre, amely egy logikai eredményt ad, és a végrehajtó eszköz az egyetlen összeget adja eredményül, válaszként a logikai eredményre és az egyetlen összeadási eredményre.

6. Az 1 - 5. igénypontok bármelyike szerinti kapcsolási elrendezés, azzal jellemezve, hogy az első skalár utasítás egy logikai utasítás, a második skalár utasítás pedig egy aritmetikai utasítás, a végrehajtó eszközök pedig az első és második operandus kombinációjához logikai eszközöket foglalnak magukban, hogy a fenti logikai utasítás és aritmetikai eszköz által megkívánt logikai eredményt adja, hogy a logikai eredményt egy harmadik operandussal kombinálja annak érdekében, hogy a fenti egyetlen

eredményt adja; a fenti egyetlen eredményt, amelyet az aritmetikai utasítás igényel.

7. Az 1 - 6. igénypontok bármelyike szerinti kapcsolási elrendezés, azzal jellemezve, hogy az első skalár utasítás egy aritmetikai utasítás, a második skalár utasítás pedig egy logikai utasítás, a végrehajtó eszköz pedig az első és második operandus kombinációjához logikai eszközöket foglal magában, hogy a fenti aritmetikai utasítás és logikai eszköz által megkívánt aritmetikai eredményt adja, hogy az aritmetikai eredményt egy harmadik operandussal kombinálja annak érdekében, hogy a fenti egyetlen eredményt adja; a fenti egyetlen eredményt, amelyet a logikai utasítás igényel.

8. Az 1- 7. igénypontok bármelyike szerinti kapcsolási elrendezés, azzal jellemezve, hogy az első skalár utasítás egy aritmetikai utasítás, a második skalár utasítás pedig egy logikai utasítás, a végrehajtó eszköz pedig a három operaandus összegzéséhez aritmetikai eszközöket foglal magában, hogy egyetlen aritmetikai eredményt adjon; a fenti egyetlen aritmetikai eredményt, amelyet a fenti egyetlen eredményként biztosítunk.

9. Az 1 - 8. igénypontok bármelyike szerinti kapcsolási elrendezés, azzal jellemezve, hogy az első skalár utasítás egy logikai utasítás, a második skalár utasítás is egy logikai utasítás, a végrehajtó eszköz pedig az első és második operandus kombinációjához logikai eszközöket foglal magában, hogy egy első logikai eredményt adjon; a fenti első logikai utasítás által megkívánt, a fenti első logikai eredmény, és egy második logikai

eszköz az első logikai eredmény és egy harmadik operandus kombinációjához egy második logikai eredmény produkálásához; a második skalár utasítás által megkívánt fenti második logikai eredmény és a fenti egyetlen eredményként biztosított fenti második logikai eredmény.

10. Többfunkciós aritmetikai logikai egység (ALU) három operandus kombinációjához, amely egy utasításpárra válaszként egyetlen eredményt ad, a z z a l j e l l e m e z v e, hogy tartalmaz

két operandust logikailag kombináló, első logikai elemkészletet az első logikai eredmény létrehozásához;

egy összeadót három operandus aritmetikai kombinációjához, hogy egyetlen aritmetikai eredményt hozzon létre;

egy áramkört az összeadó szelektív bemenet képzéséhez, vagy az összes, fenti operandussal, két fenti operandussal és egy nullával, vagy fenti operandussal, egy nullával és a fenti első logikai eredménnyel, vagy két nullával és a fenti első logikai eredménnyel;

egy, a fenti egyik logikai operandust és a fenti egyetlen aritmetikai eredményt kombináló, második logikai elemkészletet a második logikai eredmémy létrehozásához és

egy áramkört a fenti aritmetikai eredmény vagy a fenti logikai eredmény kiválasztására.

11. A 10. igénypont szerinti többfunkciós ALU, azzal jellemezve, hogy a fenti összeadó az alábbiakat foglalja magában:

egy átvitelt megőrző összeadót, amely három operandusra válaszként két kimenetet ad; és

egy, a fenti átvitelt megőrző összeadóhoz csatlakozó, átvitelt előre látó összeadót, amely a fenti két kimenetre válaszként egy kimenetet ad.

12. Kapcsolási elrendezés több skalár utasítás párhuzamos végrehajtásának támogatására egy skalár utasítássor soros végrehajtására szerkesztett számítógépben, a z z a l j e l l e m e z v e, hogy tartalmaz

egy utasítás tárolót az összetett skalár utasítások vételére, ahol a skalár utasítások közül legalább egy operandusként használja a számos skalár utasítás közül legalább egy skalár utasítás által léltrehozott eredményt;

egy operandus tárolót a fenti, skalár utasítások által használt, operandusok vételére;

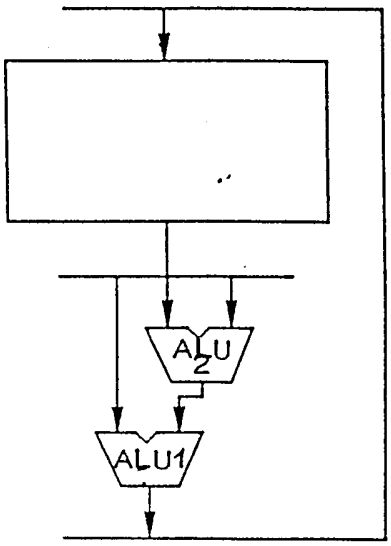
egy vezérlő jelek vételére szolgáló, az utasítás eszközhöz csatlakoztatott vezérlő egységet, hogy jelezze azokat a műveleteket, amelyek számos skalár utasítást hajtanak végre; és

egy, az operandus tárolóhoz és a vezérlő egységhez csatlakoztatott végrehajtó eszközt, amely reagál a vezérlő jelekre és a fenti, számos operandusra egy, a fenti operandusokon végzett, fenti műveleteknek megfelelő, egyetlen eredmény létrehozásához.

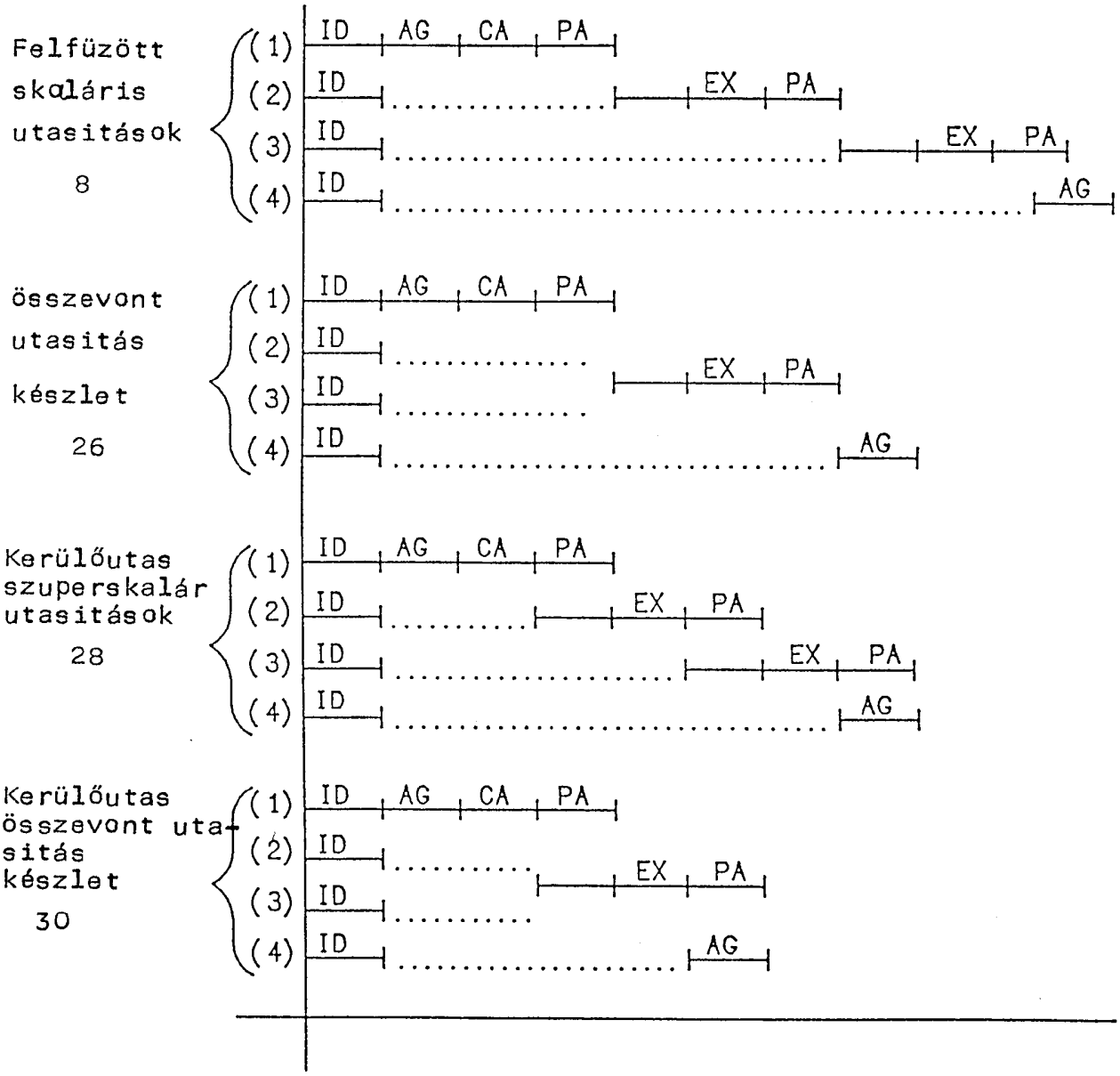
(13 rajz, 14 ábra)

A meghatalmazott:

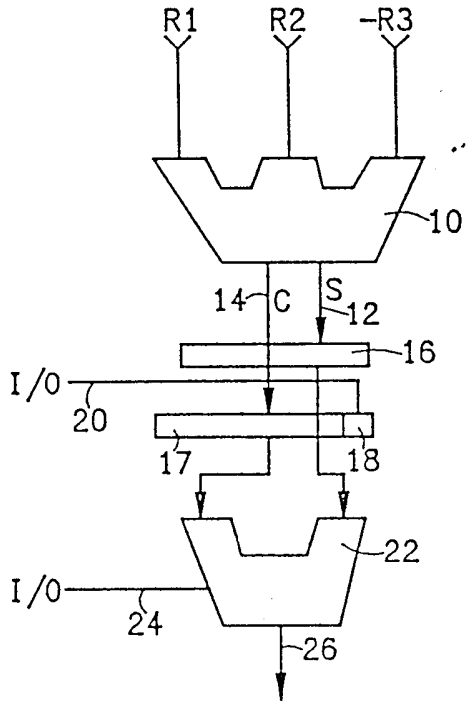
KÖZZÉTÉTELI PÉLDÁNY



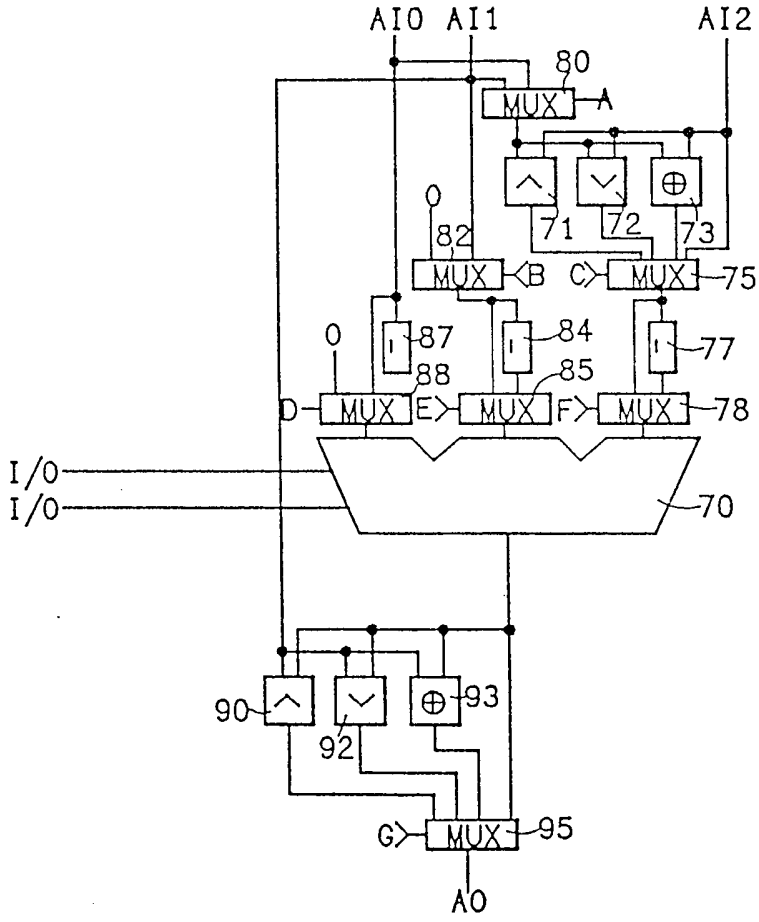
1. ÁBRA



2. ÁBRA



3. ÁBRA



9. ÁBRA

1.

- RR-formátum LOAD-, LOGICAL-, ARITHMETIC-, és COMPARE utasitások
- LCR- komplementer betöltés
- LPR- pozitiv betöltés
- LNR- negativ betöltés
- LR- regiszter betöltés
- LTR- betöltés és teszt
- NR - és
- OR - vagy
- XR - kizáró vagy
- AR - összeadás
- SR - kivonás
- ALR - logikai összeadás
- SLR - logikai kivonás
- CLR - logikai összehasonlitás
- CR - összehasonlitás

2. Formátumu léptetés (nincs tárhozzáférés)

- SRL - logikai jobbra léptetés
- SLL - lobikai balra léptetés
- SRA - aritmetikai jobbra léptetés
- SLA - aritmetikai balra léptetés
- SRDL - logikai jobbra léptetés
- SLDL - logikai balra léptetés
- SRDA - aritmetikai jobbra léptetés
- SLDA - aritmetikai balra léptetés

3. Elágazások érték és index függvényében

- BCT - elágazás értéktől függően (RX-formátum)
- BCTR - " " " (RR-formátum)
- BXH - " indextől " (RS-formátum)
- BXLE - " " " (RS-formátum)

4. Elágazás feltétel függvényében

- BC - elágazás feltételtől függően (RX-formátum)
- BCR - " " " (RR-formátum)

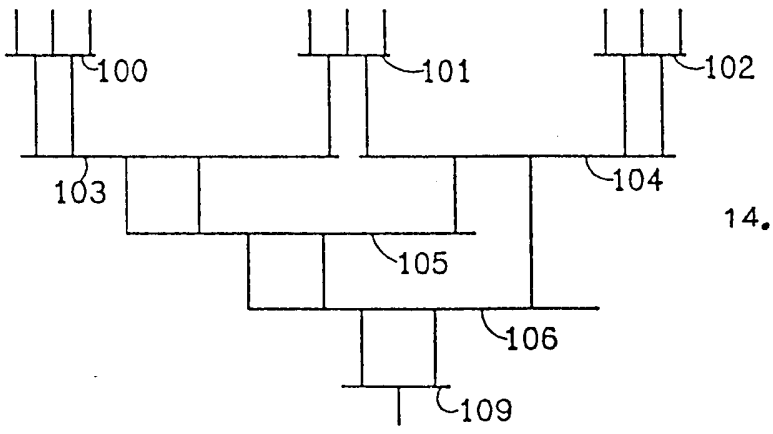
5. Elágazás és összekapcsolás

- BAL - elágazás és összekapcsolás (RX-formátum)
- BALR - " " (RR-formátum)
- BAS - elágazás és mentés (RX-formátum)
- BASR - " " (RR-formátum)

4A. ÁBRA

6. Tárolás
   - STCM - maszkolt karakterek mentése
   - MVI - közvetlen mozgatás (egy byte, SI-formátum)
   - ST - tárolás (4 byte)
   - STC - karakter tárolás (egy byte)
   - STH - félszó tárolás (2 byte)
7. Betöltés
   - LH - félszó betöltés (2 byte)
   - L - betöltés (4 byte)
8. LA - cim betöltés
9. RX/SI/RS-FORMAT ARITHMETICS,LOGICALS, BESZURÁS,ÖSSZEHASONLITÁS
   - A - összeadás
   - AH - félszó összeadás
   - AL - logikai összeadás
   - N - és
   - O - vagy
   - S - kivonás
   - SH - félszü kivonás
   - SL - logikai kivonás
   - S - kizáró vagy
   - IC - karakter beszurás
   - ICM - maszkolt karakter beszurás (0-4 byte)
   - C - összehasonlitás
   - CH - félszó összehasonlitás
   - CL - logikai összehasonlitás
   - CLI - közvetlen logikai összehasonlitás
   - CLM - maszkolt karakter logikai összehasonlitás
10. MASZKOLT TESZT

4B. ÁBRA



14. ÁBRA

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)

5. ÁBRA

1100.90 **KÖZZÉTÉTELI PÉLDÁNY**

ROW	FUNCTION	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) OP1 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		

6A. ÁBRA

30	/AI2/ - 1		
31	(-/AI2/) - 1		
32	(AI2 OP1 AI0) - 1	$\wedge,\vee,\oplus,+,-$	
33	(AI2 - 1) OP1 AI1	$\wedge,\vee,\oplus$	
34	AI2 OP1 (AI0 - 1)	+,-	
35	(AI0 - 1) OP1 AI2	+,-	
36	-(AI2 - 1)		
37	(AI2 - 1) OP1 (AI2 - 1)	$\wedge,\vee,\oplus$	
38	/AI2 - 1/		
39	-/AI2 - 1/		
40	/AI2 OP1 AI0/	+,-	
41	-/AI2 OP1 AI0/	+	
42	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	$\wedge,\vee,\oplus$	$\wedge,\vee,\oplus,+,-$
43	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	+,-	$\wedge,\vee,\oplus$
44	AI0 + AI1 + AI2 + AI3		
45	-AI0 + AI1 - AI2 + AI3		
46	(AI2 - 1) OP1 (AI2 - 1)	+,-	

6B. ÁBRA

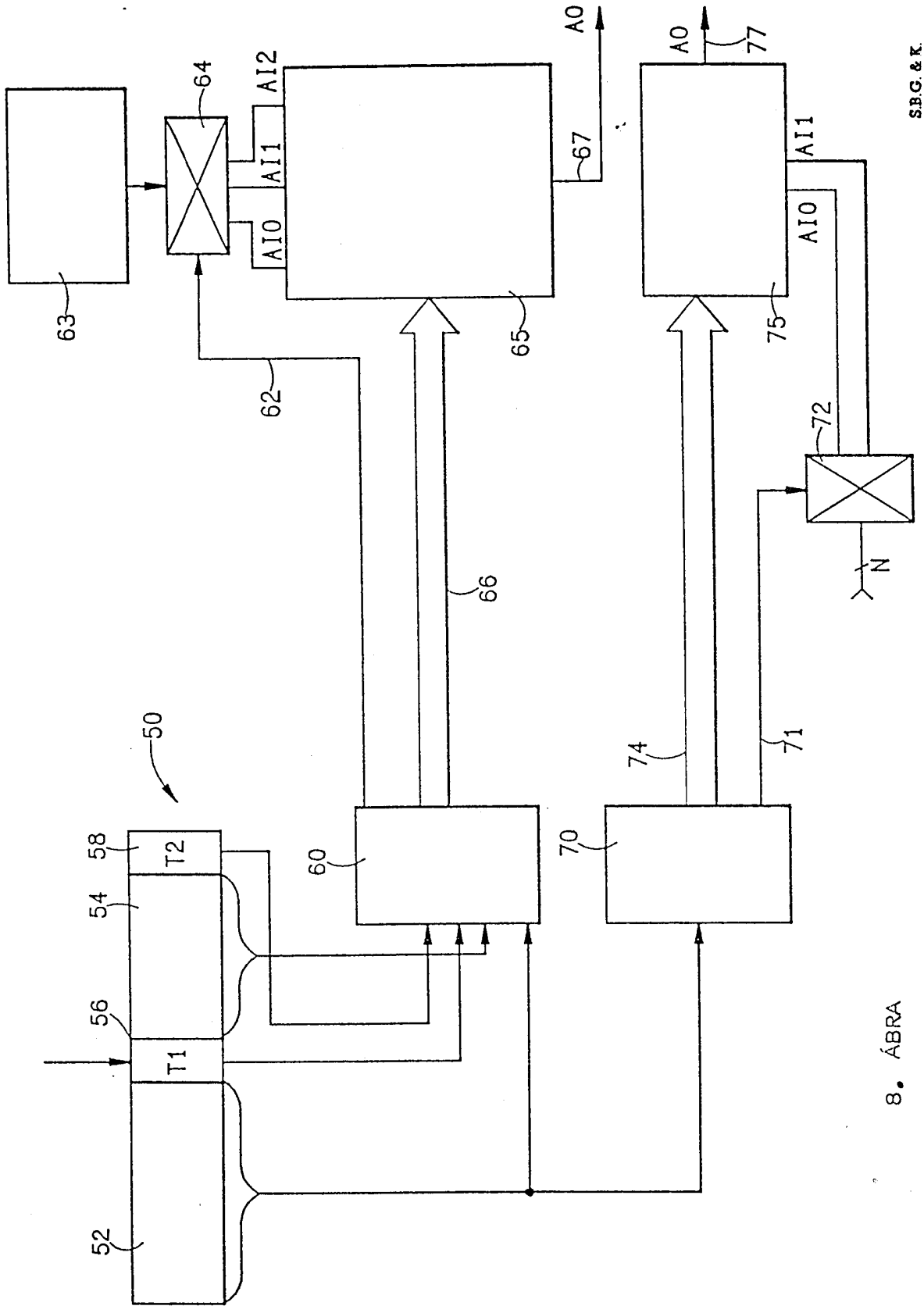
OPERAND ROUTINGS TO PRODUCE DESIRED OPERATIONS FOR LOGICAL-ADD						
DESIRED OP	ALU OP	OP1	OP2	AI0	AI1	AI2
$(R1 \varphi R2) \zeta R4$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	+, −	R2	R4	R1
$R3 \zeta (R1 \varphi R2)$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	+	R2	R3	R1
	−(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	+	R2	R3	R1
$(R1 \varphi R1) \zeta R4$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	+, −	R2(=R1)	R4	R1
$R3 \zeta (R1 \varphi R1)$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	+	R2(=R1)	R3	R1
	−(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	+	R2(=R1)	R3	R1
$(R1 \varphi R2) \zeta (R1 \varphi R2)$	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	$\wedge, \vee, \oplus$	+, −	SEE NOTE		
$(R1 \zeta R2) \varphi R4$	(AI2 OP1 AI0) OP2 AI3	+, −	$\wedge, \vee, \oplus$	R2	R4	R1
$R3 \varphi (R1 \zeta R2)$	(AI2 OP1 AI0) OP2 AI1	+, −	$\wedge, \vee, \oplus$	R2	R3	R1
$(R1 \zeta R1) \varphi R4$	(AI2 OP1 AI0) OP2 AI1	+, −	$\wedge, \vee, \oplus$	R2(=R1)	R4	R1
$R3 \varphi (R1 \zeta R1)$	(AI2 OP1 AI0) OP2 AI1	+, −	$\wedge, \vee, \oplus$	R2(=R1)	R3	R1
$(R1 \zeta R2) \varphi (R1 \zeta R2)$	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	+, −	$\wedge, \vee, \oplus$	SEE NOTE		
$(R1 \varphi R2) \varphi R4$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	$\wedge, \vee, \oplus$	R2	R4	R1
$R3 \varphi (R1 \varphi R2)$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	$\wedge, \vee, \oplus$	R2	R3	R1
$(R1 \varphi R1) \varphi R4$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	$\wedge, \vee, \oplus$	R2(=R1)	R4	R1
$R3 \varphi (R1 \varphi R1)$	(AI2 OP1 AI0) OP2 AI1	$\wedge, \vee, \oplus$	$\wedge, \vee, \oplus$	R2(=R1)	R3	R1
$(R1 \varphi R2) \varphi (R1 \varphi R2)$	(AI2 OP1 AI0) OP2 (AI2 OP1 AI0)	$\wedge, \vee, \oplus$	$\wedge, \vee, \oplus$	SEE NOTE		
$(R1 \zeta R2) \zeta R4$	AI1 OP2 (AI2 OP1 AI0)	+, −	+	R2	R4	R1
	−AI1 OP2 (AI2 OP1 AI0)	+, −	+	R2	R4	R1
$R3 \zeta (R1 \zeta R2)$	AI1 OP2 (AI2 OP1 AI0)	+, −	+, −	R2	R3	R1

7A. ÁBRA

KÖZZÉTÉTELI PÉLDÁNY

(R1ʃR1)ʃR4	AI1 0P2 (AI2 0P1 AI0)	+, –	+	R2(=R1)	R4	R1
	–AI1 0P2 (AI2 0P1 AI0)	+, –	+	R2(=R1)	R4	R1
R3(R1ʃR1)	AI1 0P2 (AI2 0P1 AI0)	+, –	+, –	R2(=R1)	R3	R1
(R1ʃR2)ʃ(R1ʃR2)	AI2 0P1 AI0 + AI1 0P2 AI3	+, –	+, –			

7B. ÁBRA



8. ÁBRA

		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)	$\wedge, \vee, \oplus, +, -$

10. ÁBRA

		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)	$\wedge,\vee,\oplus,+,-$

12. ÁBRA

1100.90

KÖZZÉTÉTELI PÉLDÁNY

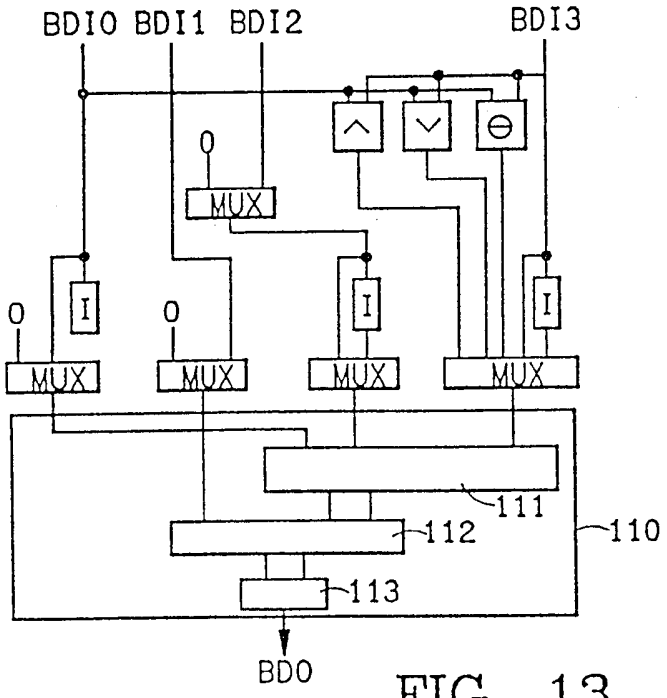
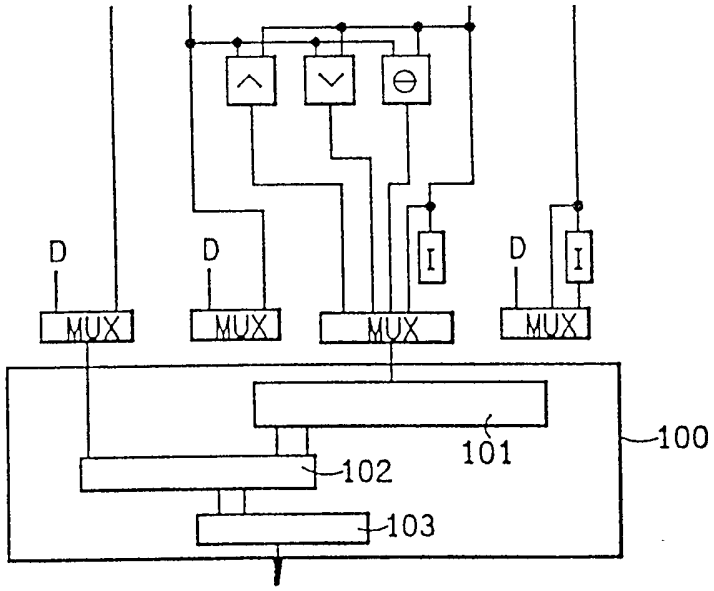


FIG. 13

13. ÁBRA



11. ÁBRA

S.B.G. & K.
BUDAPESTI NEMZETKÖZI ÜGYVÉDI
ÉS SZABADALMI IRODA
1061 BUDAPEST, DALSZÍNHÁZ U. 10.
TELEFON: 131-3733