КАТЕДРА: КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ
ДИСЦИПЛИНА: КОМПЮТЪРНИ АРХИТЕКТУРИ

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 8

ТЕМА: КОНВЕЙЕРНО ИЗПЪЛНЕНИЕ С ПРЕНАРЕЖДАНЕ НА ИНСТРУКЦИИТЕ

Цел: Да даде практически знания на студентите за работата на алгоритмите за динамично планиране (пренареждане) на инструкциите при тяхното паралелно изпълнение в конвейера на процесора.

I. Теоретична част:

Основното ограничение за постигане на по-висока производителност при конвейерното изпълнение е свързано това, че инструкциите се изпълняват в реда, в който постъпват на входа на конвейера. Ако една инструкция бъде задържана, то следващите, след нея, също не могат да бъдат изпълнявани. По този начин, ако има зависимост между две близко разположени инструкции, то в конвейера възниква състезание и извличането и изпълнението на следващите поред инструкции се преустановява, докато състезанието се прекрати.

Например:

DIVD F0, F2, F4
ADDD F10, F0, F8
SUBD F12, F8, F14

Въпреки, че инструкция *SUBD* не зависи от предходните две инструкции, тя не може да се изпълни, тъй като съществува зависимост между *ADDD* и *DIVD*, което води до задържане на изпълнението на инструкцията *ADDD* и всички след нея. Спадът на производителността в работата на конвейера, в следствие на това ограничение, може да бъде избегнат, като се даде възможност инструкциите да се изпълняват в различен ред от този, определен в изходния код на програмата.

За да може *SUBD* да започне да се изпълнява, въпреки задържането на предходната инструкция *ADDD*, процесът на допускане до изпълнение на инструкциите, който се осъществява в етапа *ID* на стандартния конвейер, трябва да се раздели на две части: 1) допускане след проверка за структурни зависимости и 2) изчакване при възможни състезания, произтичащи от зависимости по данни. След като инструкцията бъде извлечена и декодирана, т.е. веднага след като се установи каква операция трябва да се изпълни от функционалните устройства в конвейера, може да се провери и дали необходимите функционални устройства за изпълнението на операцията са свободни. Това, всъщност е проверка за структурни състезания. Освен това, изпълнението може да започне веднага щом операндите (данните) на инструкцията станат налични. По този начин. започването и завършването на изпълнението на инструкциите в конвейера, ще може да се извършва и в различен, от определения в първоначалната програма, ред.

И така, инструкциите могат да се изпълняват с пренареждане, ако етапът *ID* (познат ни от стандартния конвейер) се раздели на два етапа:

Допускане до планиране (Issue) – инструкциите се декодират и се прави проверка за структурни зависимости (респ. състезания);

Четене на операнди (Read operands) – изчаква се, докато отпаднат състезанията, породени от зависимости по данни, след което се прочитат операндите.

Между етапите *извличане от паметта* (*IF*) и *допускане до планиране* (*Issue*) инструкциите трябва да се буферират, или в регистър (само за една инструкция), или в опашка (за няколко инструкции). Етапът *EXE* е след етапа за *четене на операнди*, както е и в стандартния конвейер. Обикновено в конвейера могат да се извършват операции с числа с плаваща запетая. Всяка такава операция отнема по няколко процесорни цикъла. Това налага да се направи разграничение – кога дадена инструкция започва и кога завършва своето изпълнение.

1. Централизирано управление на инструкциите – алгоритъм „Scoreboard“

Алгоритъмът „Scoreboard“ е метод за централизирано динамично планиране. Той позволява инструкциите да се изпълняват пренаредено в конвейера. Алгоритъмът следи за състезания, причинени от зависимости по данни и за наличието на свободно функционално устройство (суматор, умножител и т.н.), което може да изпълни операцията.

Зависимостите по данни за всяка инструкция се записват и проследяват. Алгоритъмът не допуска текущата инструкция до планиране и изпълнение, докато не се установи, че състезанията между нея и предходните, вече допуснати и незавършили изпълнението си инструкции са отпаднали.

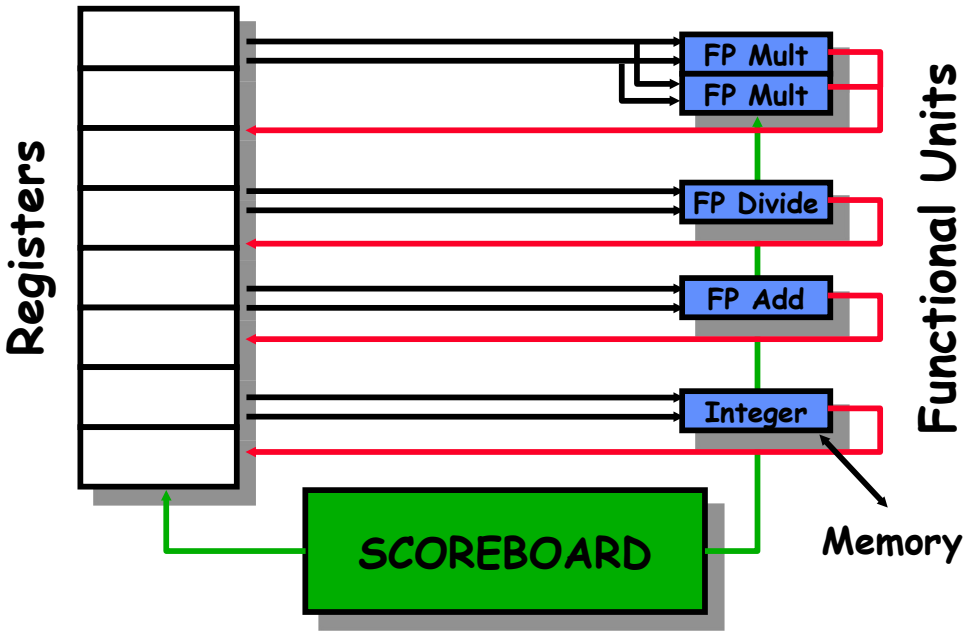
Инструкциите се извличат и декодират в реда, в който се намират в изходния код на програмата, след което преминават през следните четири етапа:

Допускане (Issue): Проверява се от кои регистри, инструкцията ще чете и в кои ще записва. Тази информация се запомня, тъй като ще е необходима в следващите етапи. За да се избегнат структурните зависимости (състезанията от типа WAW - Write after Write) инструкцията се задържа, докато предходните инструкции, които записват в същия регистър, не завършат изпълнението си. Освен това, инструкцията се задържа (остава в застой в етапа *Issue*), ако необходимите функционални устройства за нейното изпълнение са заети.

Четене на операнди (Read operands): След като инструкцията е допусната до планиране и е разпределена към необходимия за нейното изпълнение функционален модул, се изчаква, докато всички нейни операнди станат налични, т.е. докато в регистрите, в които се намират операндите не се запишат стойностите, които са резултат от изпълнението на други инструкции. Тази процедура предотвратява състезанията от вида RAW – Read after Write, причинени от зависимости по данни.

Изпълнение (Execution): Когато всички операнди бъдат прочетени, функционалното устройство започва изпълнението на инструкцията. След като изпълнението приключи, полученият резултат трябва да се запише в целевия регистър.

Запис на резултата (Write result): Изчаква се, докато има предходни инструкции, които още не са завършили етапа четене на операнди и не са прочели своите операнди от регистъра, в който тази инструкция трябва да запише резултата от своето изпълнение. По този начин се преодоляват, т.нар. анти-зависимости, водещи до състезания от вида WAR – Write after Read.



Фиг. 1. Схематично представяне на централизираното управление посредством „Scoreboard“

Фигура 1 представя схематично мястото на „Scoreboard“ при управлението на регистрите и функционалните устройства.

За да може да се управлява изпълнението на инструкциите, в алгоритъма „scoreboard“ се поддържат три таблици на състоянието:

Състояние на инструкциите (Instruction Status): Показва в кой от четирите етапа се намира всяка инструкция.

Състояние на функционалните устройства (Functional Unit Status): Показва състоянието на всяко функционално устройство. За всяко функционално устройство се поддържат 9 полета:

Busy: Показва дали функционалното устройството се използва или не;

Op: Операцията, която устройството изпълнява (умножение, деление и т.н.);

Fi: Регистърът, в който се записва резултата от операцията;

Fj, *Fk*: Регистрите, от които се четат входните операнди на инструкцията;

Qj, *Qk*: Функционалните единици, предават стойности за регистрите *Fj*, *Fk;*

Rj, *Rk*: Флагове, които показват дали *Fj*, *Fk* са готови или все още не може да се чете от тях.

Състояние на регистрите (Register Status): Посочва кой функционален елемент в кой регистър записва резултат.

Алгоритъм на работа:

След извличането на инструкцията и нейното декодиране са известни: вида на операцията (*op*), номерът на целевия регистър, в който ще се запише резултатът от тази операция (*dst*), входните операнди и номерата на регистрите от които ще бъдат прочетени (*scr1* и *src2*).

На етапа *issue* се проверява дали функционалното устройство (*FU*) е свободно и дали няма друга инструкция, в процес на изпълнение, която да записва своя резултат в целевия регистър на текущата инструкция.

Формално, работата в този и следващите етапи може да бъде описана със следния псевдо-код:

```
function issue(op, dst, src1, src2)
  wait until (!Busy[FU] AND !Result[dst]);
        //FU е всяко функционално устройство, в което операцията може да за изпълни
        operation op
        Busy[FU] ← Yes;
        Op[FU] ← op;
        F_i[FU] ← dst;
        F_j[FU] ← src1;
        F_k[FU] ← src2;
        Q_j[FU] ← Result[src1];
        Q_k[FU] ← Result[src2];
        R_j[FU] ← Q_j[FU] == 0;
        R_k[FU] ← Q_k[FU] == 0;
        Result[dst] ← FU;

function read_operands(FU)
  wait until (R_j[FU] AND R_k[FU]);
        R_j[FU] ← No;
        R_k[FU] ← No;

function execute(FU)
        // Извършва се операцията от страна на FU

 function write_back(FU)
  wait until (∀f {(F_j[f]≠F_i[FU] OR R_j[f]=No) AND (F_k[f]≠F_i[FU] OR R_k[f]=No)})
```

foreach f do

if $Q_j[f]=FU$ then $R_j[f] \leftarrow$ Yes;

if $Q_k[f]=FU$ then $R_k[f] \leftarrow$ Yes;

Result[$F_i[FU]$] ← 0;

// 0 означава, че няма FU, което да записва резултат в регистъра

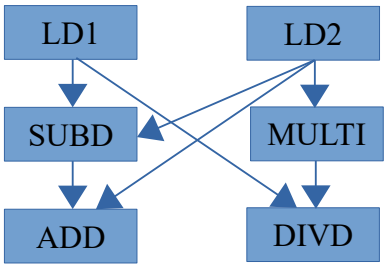
RegFile[$F_i[FU]$] ← *computed value*;

Busy[*FU*] ← No;

II. Практическа част:

На следващите фигури е представено управлението на инструкциите от страна на „scoreboard“ при изпълнението на следната примерна програма:

```
LD      F6, 34(R2)     #  F6 ← MEM[R2+34]
LD      F2, 45(R3)     #  F2 ← MEM[R3+34]
MULTI   F0, F2, F4     #  F0 ← F2 * F4
SUBD    F8, F6, F2     #  F8 ← F6 - F2
DIVD    F10, F0, F6    #  F10 ← F0 / F6
ADD     F6, F8, F2     #  F6 ← F8 + F2
```



В дясно от горната програма са показани зависимостите между инструкциите, както и възможността за тяхното пренареждане и паралелно изпълнение.

Броят на процесорните цикли, необходими за изпълнението на операциите са: зареждане от паметта (LD) – 1 цикъл; събиране и изваждане (ADD/SUB) – 2 цикъла; умножение (MULTI) – 10 цикъла; деление (DIV) – 40 цикъла.

Преди да започне изпълнението трите таблици на състоянието са празни.

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Cycle	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									

Таблиците на състоянието след първия процесорен цикъл изглеждат така:

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

В първия процесорен цикъл, първата LD инструкция е допусната по планиране.

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Състоянието на функционалното устройство Integer се променя на Busy (заето), докато първата LD инструкция извърши и запише резултата в регистър F6.

Register result status:

Cycle	F0	F2	F4	F6	F8	F10	F12	...	F30
1				Integer					
FU									

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Във втория процесорен цикъл, първата LD инструкция се намира на етапа „Read Operands“.

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Втората LD инструкция не се допуска до планиране, тъй като няма свободно функционално устройство за нейното изпълнение.

Register result status:

Cycle		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU				Integer					

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

В третия процесорен цикъл, първата LD инструкция се намира на етапа „Execution“.

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Втората LD инструкция все още не е допусната, тъй като функционално устройство за нейното изпълнение продължава да е заето.

Register result status:

Цикъл		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU				Integer					

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

В четвъртия процесорен цикъл, LD1 се намира на етапа „Write Result“.

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Остават същите ограничения за недопускане на LD2. Инструкцията MULT още не е извлечена от паметта, тъй като буфера преди етапа issue може да поеме само по един елемент и в този момент това е LD2.

Register result status:

Cycle		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU				Integer					

На следващите фигури продължава представянето на трите таблици: *Instruction Status, Functional Unit Status и Register Status,* в които се записва състоянието от изпълнението на инструкциите в програмата.

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

В петия процесорен цикъл, LD2 е допусната до паниране, а MULT – извлечена и поставена в буфера преди етапа issue.

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Cycle		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU		Integer							

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

В шестия процесорен цикъл, LD2 се намира на етапа „Read Operands“ Инструкцията MULT се допуска до планиране и се извлича SUBD.

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	MULT	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Инструкцията MULT зависи от резултата на LD2, който ще се запише в регистър F2. Rj=No показва, че операнда все още не готов, а Qj=Integer, че стойността му ще дойде от функционалното устройство с име Integer.

Register result status:

Cycle		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	Integer							

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	MULT	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	SUBD	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

Cycle		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	Integer			Add				

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status:

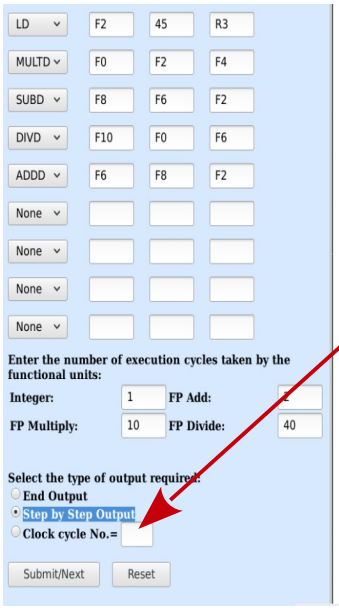
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	MULT	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	SUBD	F8	F6	F2		Integer	Yes	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1	Add	No	No

Register result status:

Cycle		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	Integer			Add	Divide			

III. Задачи за изпълнение:

Задача 1: Проследете как завършва изпълнението на програмата, представена в практическата част на това упражнение, като посетите следния адрес: http://www.ecs.umass.edu/ece/koren/architecture/scoreboard/demo/index1.htm



Използвайте опцията „Step by Step Output“ и бутона „Submit/Next“, за да проследите съдържанието на таблиците в различните процесорни цикли.

Имайте предвид, че част от процесорните цикли, в които съдържанието на трите таблици не се променя се пропускат.

Това се налага, тъй като операцията умножение отнема 10 процесорни цикъла, а операцията деление – 40.

Отговорете на слените въпроси:

- След колко процесорни цикъла може да започне повторно изпълнение на програмата? (т.е. кога първата инструкция *LD* може да бъде допусната до планиране, отново за втори път)
- Как влияе пренареждането на времето за изпълнение на тази програма?
- Възникват ли състезания от вида Write-after-Read (WAR)? Как се преодоляват?

Задача 2: Отидете на следния адрес: http://www.ecs.umass.edu/ece/koren/architecture/scoreboard/. Въведете кода на следващата програма от 6 инструкции. Изпълнете я постъпково и проследете състоянието на инструкциите в трите таблици.

```
LD    F0, 0(R1)
ADDD  F4, F0, F2
SD    F4, 0(R1)
LD    F0, -8(R1)
ADDD  F4, F0, F2
SD    F4, -8(R1)
```

Отговорете на следните въпроси:

- След колко процесорни цикъла тази програма може да започне да се изпълнява отначало?
- Как влияе пренареждането на времето за изпълнение на тази програма?
- Възникват ли състезания от вида Write-after-Read (WAR)? Как се преодоляват?

Задача 3: Въведете кода и проследете изпълнението на следващата програма. В нея се сумират произведения. Използва се за реализация на бързо преобразование на Фурие (Fast Fourier Transform).

```
LD    F0, 0(R1)
LD    F2, 4(R1)
MULTD F8, F0, F2
LD    F4, 8(R1)
LD    F6, 10(R1)
MULTD F10, F4, F6
ADDD  F10, F8, F10
LD    F8, 12(R1)
ADDD  F10, F10, F8
```

Отговорете на следните въпроси:

- След колко процесорни цикъла тази програма може да започне да се изпълнява отначало?
- Как влияе пренареждането на времето за изпълнение на тази програма?
- Възникват ли състезания от вида Write-after-Read (WAR)? Как се преодоляват?