КАТЕДРА: КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ
ДИСЦИПЛИНА: КОМПЮТЪРНИ АРХИТЕКТУРИ

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 10

ТЕМА: СПЕКУЛАТИВНО ИЗПЪЛНЕНИЕ НА ИНСТРУКЦИИТЕ
РАЗШИРЕН АЛГОРИТЪМ НА ТОМАСУЛО

Цел: Да даде теоретични и практически знания на студентите за начина на спекулативно изпълнение на инструкциите за преход и реализацията му на базата на алгоритъма на Томасуло за динамично планиране.

I. Теоретична част:

Зависимостите по управление, които следват след всяка инструкция за преход ограничават възможностите за максимално възползване от паралелизма на ниво инструкции. Използваните методи за предсказване на преходите намаляват застоите, но не са достатъчно ефективни за процесор, изпълняваш множество инструкции за един машинен цикъл (wide issue processor). Това налага да се разработи и приложи методът за спекулативно изпълнение на инструкции.

За преодоляване на зависимостите по управление, се прави предположение – как следва да продължи програмата след инструкцията за преход. Веднага след това инструкциите посочени от предположението започват да се извличат, допускат и изпълняват спекулативно до момента, в който се установи каква ще бъде действителната посока на прехода. Ако действителната посока съвпада с направеното предположение – изпълнените инструкции завършват, ако не – спекулативно изпълнените инструкции не завършват и биват премахнати от конвейера.

Спекулативното изпълнение включва три ключови идеи: (1) динамично предсказване на преходи, за да се избере инструкциите в кой клон на прехода да се изпълнят, (2) изпълнение на предложените инструкции преди да е преодоляна зависимостта по управление (с възможност за възстановяване на правилното изпълнение на програмата, ако предсказването на прехода е било невярно) и (3) динамично планиране за разпределяне на функционалните блокове, в които се изпълняват операциите на инструкциите.

При наличие само на динамичното планиране (без спекулативно изпълнение) е задължително да се установи каква е действителната посока на прехода, преди да започне изпълнението на каквато и да е инструкция след инструкцията за преход. Това ограничава възможностите за застъпване на инструкциите в блоковете за изпълнение.

За да поддържа спекулативно изпълнение, конвейерът с алгоритъм на Томасуло трябва да се разшири. За целта предаването на резултатите между инструкциите и действителното им завършване се разделя на два самостоятелни етапа. Това, позволява една инструкция да се изпълни и да предаде получения резултат към друга инструкция без да записва резултата си в целевия регистър. След като се установи че спекулативно изпълняваната инструкция е коректно предсказана, тя престава да е спекулативна и

резултатът се записва или в нейния целеви регистър или в паметта. Този допълнителен етап в изпълнението на инструкциите в конвейера с алгоритъм на Томасуло се нарича „окончателно завършване“ (instruction commit).

Използването на предадената стойност, а не на окончателно записаната е подобно на спекулативно четене от регистър. Така че, докато инструкцията не престане да е спекулативна, не е ясно дали осигурената, като входен операнд, стойност за инструкцията получател е коректна или не.

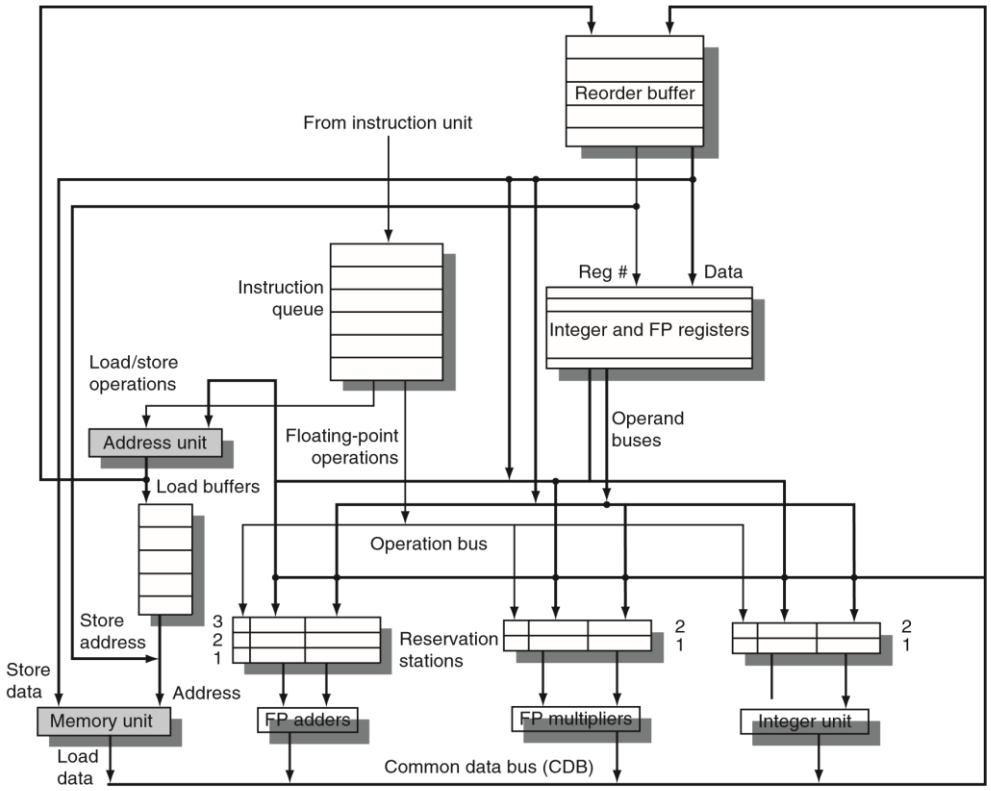
Основаната идея при реализацията на спекулативното изпълнение е да се позволи инструкциите да се изпълняват непоследователно (out of order), но окончателното им завършване да бъде в реда на постъпване (in order). Така се предотвратява извършването на операции, които не могат да бъдат отменени (обновяване на регистрите или паметта или възникването на изключение) докато инструкцията не завърши окончателно (commit). От тук и необходимостта да се раздели процесът на завършване на изпълнението на операцията от окончателното завършване на инструкцията. По този начин, изпълнението на операцията на една инструкция може да завърши значително по-рано от нейното окончателно завършване. Добавянето на фазата „окончателно завършване“ към последователността в изпълнението на инструкциите изисква допълнителна група от регистри, в които да се буферира резултатът от изпълнението на операциите, до окончателното завършване на инструкциите. Тази група от регистри се нарича ROB (reorder buffer) и освен това се използва за предаване на резултата към зависими инструкции, които могат да бъдат и спекулативни.

ROB буферът осигурява допълнителни регистри по същия начин, както станциите за резервиране разширяват набора регистри в конвейера с алгоритъм на Томасуло. ROB буферира резултата на една инструкция за времето от завършване на операцията на тази инструкция до времето за окончателното ѝ завършване. Следователно, ROB е източник на операнди за инструкциите, точни като и станциите за резервиране осигуряват операндите в алгоритъма на Томасуло. Основната разлика с алгоритъма на Томасуло е че след като веднъж инструкцията запише резултата в целевия си регистър, всяка последваща инструкция ще прочете този резултат от регистровия файл. При спекулативното изпълнение регистровия файл не се обновява докато инструкцията не завърши окончателно. Така ROB осигурява операндите в периода между завършване на изпълнението на операцията и окончателното завършване на инструкцията. ROB е подобен на „store“ буфера в алгоритъма на Томасуло, затова функциите на „store“ буфера се интегрират в ROB.

Всеки запис в ROB съдържа четири полета: тип на инструкцията, целеви регистър (destination), стойност (value) и флаг за готовност (ready). Полето тип на инструкцията показва дали инструкцията е: инструкция за преход (следователно липсва целеви регистър за запис на резултат), инструкция от тип „store“ (следователно резултата се записва в паметта) или е инструкция изпълняваща регистрова операция (АЛУ операция

или инструкция „load“, които запиват в целеви регистър). В полето “destination” се посочва номера на регистъра (за инструкциите от тип „load“ и АЛУ операциите) или адрес в паметта (за инструкциите от тип „store“), в който резултатът от инструкцията трябва да се запише. Полето “value” се използва за да се запише стойността на резултата от инструкцията докато нейното изпълнение завърши окончателно. Полето „ready“ показва, че е завършило изпълнението на операцията и резултатната стойност е налична.

На фигура 1 е показана хардуерната структура на процесор, включващ ROB. The ROB включва буферите за инструкциите „store“. Инструкциите „store“ все още се изпълняват в две стъпки, но втората стъпка се изпълнява при окончателното завършване в етапа „commit“. Въпреки, че преименуването на регистрите се премества от станциите за резервиране в ROB, все още е необходимо място за буфериране на операциите (и операндите) между момента на допускане и момента на започване на изпълнението. Тази функция все още се изпълнява от станциите за резервиране. Всяка инструкция има позиция в ROB, докато завърши окончателно. Номерът на тази позиция се използва, вместо номера на резервационната станция, за да се посочи къде трябва да се прехвърли резултатът от инструкцията. В резервационната станция се посочва същият ROB номер с цел проследяване на резултата.



Фиг. 1. Базова структура на конвейер с алгоритъм на Томасуло и ROB за спекулативно изпълнение.

Етапите при спекулативно изпълнение на инструкции са четири:

1. Issue – Взема се инструкция от опашката за инструкции. Ако има празна резервационна станция и свободно място в ROB, инструкцията се допуска. Ако операндите на инструкцията са налични в регистрите или в ROB, те се изпращат в станцията за резервиране. Посочва се кой регистър от ROB ще се използва за да се запише резултатът от инструкцията, когато стане готов и се появи на CDB. Номерът на този регистър също се изпраща до станцията за резервиране. Ако всички станции за резервиране са заети или ROB буферът е пълен, допускането на инструкцията се задържа, докато станат налични.

2. Execute – Ако един или повече от операндите все още не са налични, се следи шината CDB в очакване стойностите свързани със съответните регистри да бъдат изчислени. По този начин се предотвратяват конфликти от тип RAW. Когато и двата операнда са налични в станцията за резервиране, започва изпълнението на операцията. Изпълнението може да продължи повече от един машинен цикъл. Инструкциите от тип „loads“ се изпълняват за две стъпки в този етап – изчисляване на адреса и четене от паметта. На този етап за инструкциите от тип „stores“ е необходимо само да се изчисли ефективния адрес от паметта, в който ще се записва.

3. Write result – Когато резултатът стане готов, той се записва върху CDB шината и от там се прехвърля в съответния регистър в ROB, както и във всяка резервационна станция, очакваща този резултат. Резервационната станция се обозначава като налична. При инструкциите от тип „store“, съхраняваната стойност, ако е налична се записва в полето „value” на ROB буфера за последващо съхранение в паметта. Ако съхраняваната стойност не е налична, трябва да се следни CDB, докато стойността се изведе там. В този момент полето „value“ на ROB регистъра асоцииран с инструкцията store се обновява с появилата се на шината стойност. Приема се, че това се случва по време на етапа „write results“ за инструкцията от тип „store“.

4. Commit – Това е финалният етап от завършването на инструкцията, след този етап остава само нейния резултат. (В някой процесори този етап се нарича „completion” или „graduation.”) На тази фаза има три различни последователности от действия, в зависимост от това дали завършващата инструкция е инструкция за разклонение (branch) с некоректно предсказан преход, инструкция от тип „store“ или друга инструкция (нормално завършване). Нормалното завършване се получава когато инструкцията стане първа в ROB и резултатът е наличен в нейния ROB регистър; В този момент резултатът се прехвърля от ROB в целевия регистър и инструкцията се премахва от ROB. Завършването на „store“ е подобно с изключение на това, че резултата се записва не в регистър а в паметта. Когато инструкция от тип „branch“ с некоректно предсказан преход достигне до началото на ROB, то направената спекулация е грешна. ROB се изпразва и изпълнението се рестартира от коректната инструкция сочена от прехода. Ако инструкцията от тип „branch“ е била предсказана коректно, тя завършва.

След като една инструкция завърши окончателно, целевият регистър или целевата клетка в паметта се обновяват и нейната позиция в ROB се освобождава. Ако ROB се запълни спира допускането на инструкции, дкато не се освободи място в буфера.

II. Практическа част:

Ще разгледаме същата последователност от инструкции, която беше представена в предходното упражнение.

```
LD      F6, 34(R2)    # F6 ← MEM[R2+34]
LD      F2, 45(R3)    # F2 ← MEM[R3+34]
MULTD   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
```

Броят на процесорните цикли, необходими за изпълнението на операциите са: зареждане от паметта (*LD*) – 1 цикъл; събиране и изваждане (*ADD/SUB*) – 2 цикъла; умножение (*MULTD*) – 6 цикъла; деление (*DIV*) – 12 цикъла.

На фиг. 2 в трите таблици е представено състоянието на конвейера в момента, когато е завършило изпълнението на операцията *MULTD*.

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	No	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	Yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	Yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	Yes	DIV.D	F10,F0,F6	Execute	F10	
6	Yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mult1	No	MUL.D	Mem[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	Yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

FP register status

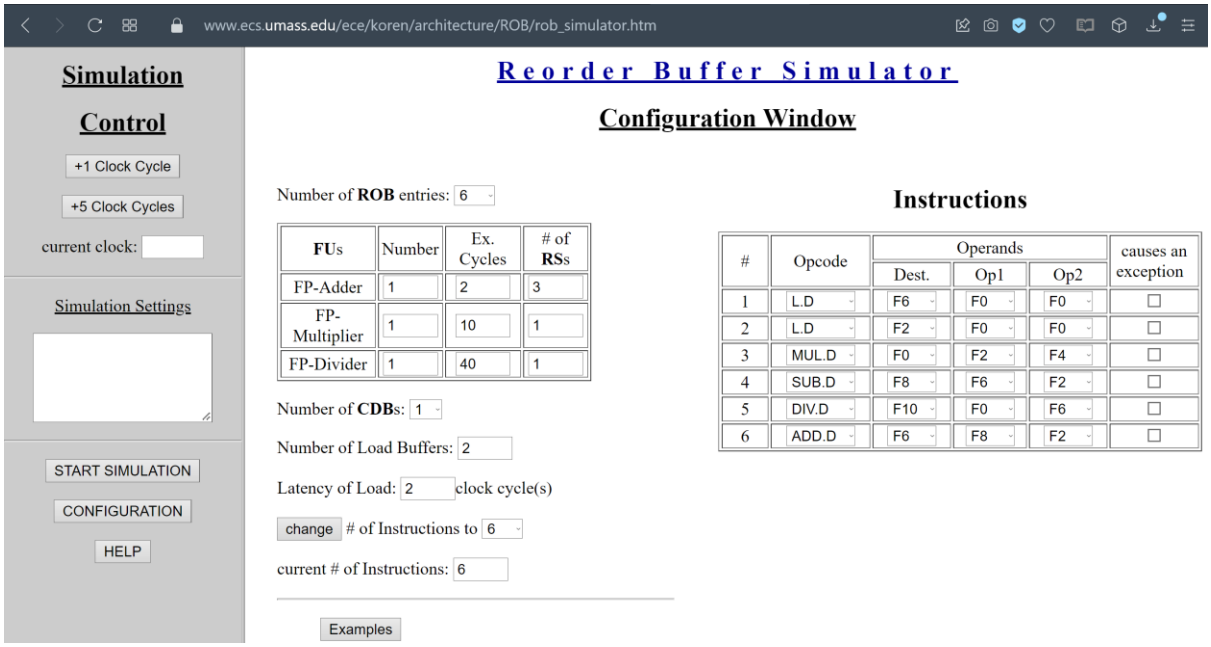
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	Yes	No	No	No	No	No	Yes	. . .	Yes	Yes

Фиг. 2. Състояние на конвейера в момента, в който MULD може да завърши окончателно. Само LD инструкциите за завършили окончателно, въпреки че има и други инструкции, които са завършили изпълнението на операциите си.

III. Задачи за изпълнение:

Задача 1:

Проследете как завършва изпълнението на програмата, представена в практическата част на това упражнение, като посетите следния адрес: https://www.ecs.umass.edu/ece/koren/architecture/ROB/rob_simulator.htm



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)
- Каква е разликата между алгоритъма на Томасуло с пренареждане и алгоритъма на Томасуло със спекулативно изпълнение?
- Възникват ли състезания от вида Write-after-Read (WAR)? Как се преодоляват?

Задача 2:

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
```

Добавете изключения и проследете какво влияние оказват на изпълнението на програмата.