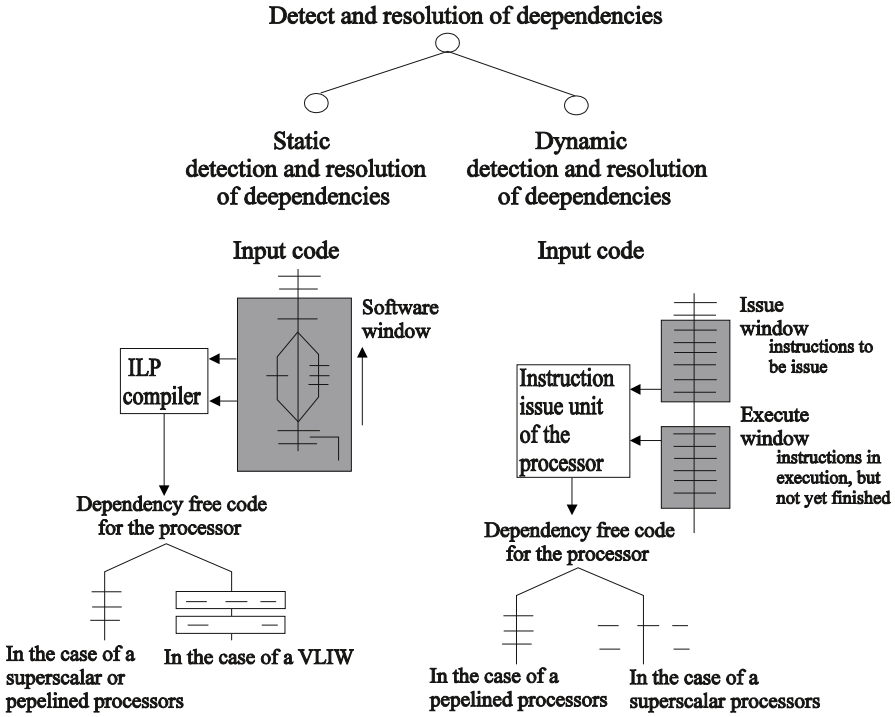
1фаза: Инструкцията се извлича от Cache или от оперативната памет, след това 2 фаза: декодиране, която се комбинира с фазата разпределяне, защото декодирането зависи от вида на инструкцията.
При Branch инструкцията втората фаза е всичко – и декодиране и изпълнение и т.н.
При фиксирана запетая – фазите са 4.
При Load/Store – тук се вмъква допълнителна фаза за генериране на адреси от които да се направи Load или Store – Add Gen
При плаваща запетая – след декодиране на инструкцията има 2 фази на изпълнение, защото при тези числа операциите са по-продължителни във времето. Затова те са разделени на 2: Execute1 и Execute2.
Диспечиращото устройство съдържа буфер. Той служи за реализиране на функцията поглеждане напред - във всеки един момент създава наличие от инструкции за изпълнене. Тези инструкции той ги получава от Fetch устройството. Буферът се състои от две части : Instruction и Dispatch buffer. Горната половина (Instruction) служи за поддържане на количество от инструкции, а долната половина осъществява "поглеждането напред".
Задачата на диспечера е да осигурява инструкции, които да подава към останалите устройства. За целта той ги съхранява във буфера и в момента, когато например пристигне инструкция за фиксирана запетая веднага я подава към съответното устройство за декодиране. Диспечера се опитва във всеки момент да осигури 3 инструкции за изпълнение едновременно (Instruction Completion, Integer Arithmetic write back, Integer Load write back) благодарение на буферите, където следи кодовете.

Откриването и преодоляването на зависимостите(фиг.10_8):

Има два начина за разкриване на зависимости:
- статични – изпълнението на програмата е статично (без движение). Компилатора разглежда програмата и се опитва да открие и компилира независими инструкции. Опитва се в една дълга инструкция да вмъкне всички инструкции и да ги подреди така, че те да са зависими колкото се може по-малко една от друга.
Зависимостта на инструкциите се изразява в това, че една инструкция трябва да създаде резултат, а друга да ползва този резултат, като това не може да бъде избегнато в никакъв случай. Но има и други зависимости между инструкции, които са само формални.
- динамични – има движение в процеса на изпълнение на програмата, което се изпълнява от компилатора. Използват се 2 прозореца.- в долния (Execution) се намират всички инструкции, които в момента се изпълняват, а в горния са всички инструкции, които вече са извлечени (изпълнени). Анализирайки двата прозореца устройството Dependency free трябва да подреди инструкциите според това дали трябва да чакат или могат да бъдат пуснати за изпълнение.

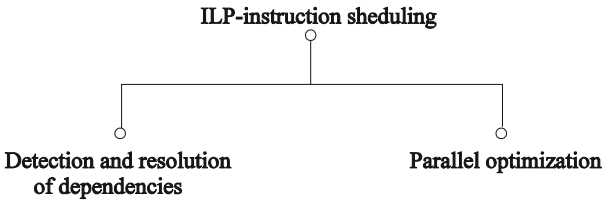
В първия случай трябва да се търси последователност на инструкциите, а в другия – компилаторът търси възможност за паралелно изпълняване на инструкциите, които са независими по между си.Фиг.10_8



Issue unit – генерира кода , който се подава на входовете на EUs.

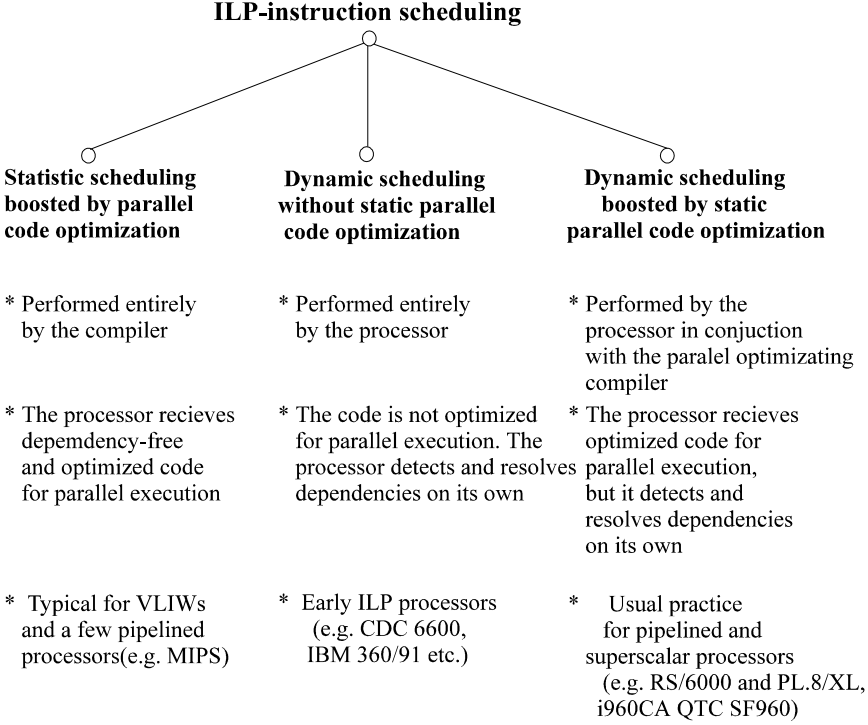
Ако зависимостите се откриват статично чрез компилатор ясно е, че ще се подобри оптимизацията за паралелизма. Но ако се открива и преодолява зависимостта динамично, то паралелната оптимизация ще се извърши след кодовата оптимизация. Прави се преподреждане на последователноста от инструкции.

На фиг.10_9 са показани двете работи свързани с ILP-процесори.



Трябва не само да открият инструкциите със зависимост, но и да се използват възможностите на паралелната машина. Прави се оптимизиране за паралелизъм, което може да се осъществи само от компилатора. Когато двата проблема – *откриване и преодоляване на зависимости* и *оптимизиран паралелизъм*, са на лице то се говори за разписание(*sheduling*). Технологията на разписанието(фиг.10_9) включва две задачи:

- откриване и преодоляване на зависимости



- паралелно оптимизиране.

Вследствие на двете задачи се появяват три класа машини(фиг.10_10):

1. Статистическо планиране, подпомогнато с паралелна оптимизация. Разчита се изцяло на компилатора. И в резултат от работата му, процесорът получава код за изпълнение, който е без зависимости и с паралелна оптимизация. Представители на този клас са всички VLIW машини.
2. Машини, които откриват динамично зависимостите и ги преодоляват, като не се извършва паралелна оптимизация. Процесорът по време на изпълнение открива и преодолява зависимостите.(IBM 360/91)
3. Динамично откриване на зависимости, подпомогнато от компилатора за паралелна оптимизация. Процесорът получава оптимизиран код от паралелен оптимизиращ компилатор и по време на изпълнение процесорът открива зависимостите между инструкциите.Типичен представител – RS/6000.

Трябва да се осигури паралелизъм на ниво инструкции. Ограничава ни зависимостта на инструкциите, защото трябва да спазваме последователността. На ниво инструкции трябва да има последователност.

Обикновено откриването и преодоляването на зависимостите е част от политиката на подаване на инструкции на процесора. Паралелизмът на ниво инструкции е определен от честотата на истинската даннова зависимост(RAW) и процедурните зависимости в кода.

фиг.10_10

При паралелното изпълнението на инструкциите, както при ILP-машина, трябва да се запази последователност на изпълняваната програма. Например:

```
div     r1, r2, r3;
ad      r5, r6, r7;
jz      anywhere;
```

при изпълнението инструкцията *ad* винаги ще се изпълнява след *div*. И *jz* следователно проверява знака на резултата от събирането, за да се осъществи преход(ако резултата е равен на нула) или не. Но при паралелно изпълнение на инструкциите *ad* и *div* без специално подреждане, кратката инструкция *ad* ще завърши изпълнението си първа, а по-дългата инструкция *div* – по-късно. И вследствие това - инструкцията jz ще провери резултата от *div*. Паралелното изпълнение нарушава последователността. Терминът последователна устойчивост е въведен(фиг.10_11).

При съхранение на последователността трябва да се спазват два аспекта:

(1) -последователния характер да бъде съхранен; Дели се на съхранение на истинската последователност на инструкциите и на съхранение на обръщенията към паметта.

В програмата може да има някаква логика в последователността на обръщенията към паметта. Има три вида консистенции:

- на процесора
- на паметта
- на прекъсванията.

(2) -обработване на прекъсвания(изключения) – Обработката на прекъсвания има смисъл, когато протича един процес, при асинхронен характер на прекъсванията. Има последователност на прекъсванията. Тази последователност трябва да бъде съхранена. Процесорът следи тази последователност и я спазва.

Фон Нойманова машина е последователна.

фиг.10_11 Implementation ot the concept of sequential consistency.