Терминът суперскаларен за първи път се използва през 1987г. и с него се обозначават машини проектирани да подобрят производителността при изпълнението на скаларните инструкции. При повечето приложения се извършват множество от операциите върху скаларни инструкции. Може да се каже че суперскаларният подход е следващата стъпка към създаването на високо-производителни прцесори.
Основата на СС подход е способността му да изпълнява инструкции независимо и едновременно в различните конвейри. Концепцията може да е разширим с това че инструкциите могат да бъдат изпълнени в различна от програмата последователност. На фиг.1 е показана идеята за този подход, а именно има множествофункционални устройства, всяко едно от тях е реализирано като конвейр, който може да изпълни паралелно няколко инструкции. За дадения пример 2 целочислени, 2 с плаваща запетая и една операция обръщение към паметта.
Въведен беше алтернативен подход-суперконвейр, с който се постигна по-голяма производителност-1988. [JOUP88]. На фиг.2 е показан и горния конвейр се състои от 4 фази. При суперконвейра имаме степен на 2. и най-долу стои суперконвейра, който в дадения пример може да изпълнява във всяка фаза по две инструкции. Степените на суперскаларният подход зависят от хардуерната реализация на регистровия файл, който е многопортов.
Понятието паралелизъм на ниво инструкции е свързано с степента с която инструкциите от програмата могат да бъдат изпълнение паралелно. За да се увеличи тази степен в съвремените СС архитектури се използва комбинацията от компилатор за оптимизация на програмата и хардуерни техники. Преди да разгледаме дизайна на СС машини които позволява да се използва ILP, трябва да ви покажа основните ограничения на този паралелизъм, с който СС процесори трябва да се справят. А именно зависимостити в програмите които възникват при тяхното изпълнение. Най-голям проблем са решаването на зависимостите по данни и по преход.
Зависимостите по данни се делят на такива в последователен код и итерации от цикъл.

Слайт 10 Машинният паралелизъм измерва способността на процесора да се възползва от предимството на ILP. Той се определя от паралелните конвейри и от скоростта на механизмите който процесора използва за да открива независими инструкции.
Тъй като машинният паралелизъм се нуждае от множество инструкции в всяка една фаза чрез термина политика на извличане се обозначава протокола който определя как се извличат инструкциите. Ние казваме че инструкциите подават от декод фазата към конвейрите. Има три типа подредби:

- последователност, по която инструкциите се извличат
- ---------- изпълняват
- ---------- записват

Условието на задачата:
Приемаме че се извличат и декодират по две.
Слайт 7: Зависимости по преход.
Във време 0 се извлича инструкция за условен преход от паметта, а с време 1 се изпълнява в резултат на което имаме нова стойност на програмния брояч. Едовременно с това обаче е извлечена инструкцията add която е следвала предишната инструкция. В такт 2 от адреса на таргета на условния преход е извлечена следващата инструкция, но тя не може да се изпълние на такт 3 защото тогава се изпълнява все още ADD. При RISC се използва техниката DELAY-BRANCH
За избягването на тези инструкции има различни подходи.
1. Multiple Stream – Тъй като инструкцията за преход когато се изплнява може да изиска да се извлече нов блок от инструкции то конвейра се организира да поддържа

двата потока от инструкции – последователния с инструкцията за преход и този с предсказания преход. Възникват два проблема:
- дублирането на конвейрите води до прекъсвания за да се достъпят регистрите и паметта;
- има опасност в конвейра да навлезе нова инструкция за преход; а всяка нова инструкция се нуждае от допълнителен поток.
Въпреки тези недостатъци тази стратегия може да подобри производителността. Пример за това е IBM 370/168 и IBM3033, при който конвейра е с 2 потока и повече.

2. Prefetch branch target – предсказва се прехода и се извлича предварително. Ако след изпълняването на последователността от инструкции тази инструкция се изпълни тогава се взима тага. IBM 360/91.

3. Loop Buffer – малка бърза памет, която се запълва с инструкциите от фазата за извличане и съдържа n най-често извличани инструкции. Ако прехода е предсказан HW първо проверява за branch target в този буфер. Предимствата на Loop Buffer са:
- ако има инструкция за преход при която таргета е няколко инструкции напред, то той ще се запише в буфера; това е подходящо когато имаме IF-THEN или IF-THEN-ELSE.
- тази стратегия е подходяща за цикли или итерации, от тук идва и наименованието loop buffer ; ако буфера е достатъчно голям, то той ще съдържа всички инструкции в цикъл и ще е необходимо само веднъж да се извлекат тези инструкции от паметта за първата итерация; вз следващите итерации всички необходими инструкции вече са в буфера; фиг.loop buffer е пример за него. Ако буфера има капацитет 256байта и се използва 1 B за адресиране, то тогава поне 8бита се използват за индексиране в буфера. Останалите битове от адреса участват в сравнението за хит.

4. Branch Target Prediction – различни техники могат да се използват за да се предскаже прехода дали ще се изпълни. Има следните възможности: ....
Първите три подхода са статични: те не се повлияват от историята на изпълненията на предходни инструкции за преход. Последните два подхода са динамични – те се основават на историята на изпълнението.
Първият подход е най-прост за реализация. Приема се че или преходите няма да се изпълнят и продължава да се извлича от опашката на следващите инструкции или винаги ще приемем че инструкциите ще се изпълнят и извличаме инструкциите от новата стойност на PC. Методът predict-never-taken в най-известен от останалите.
За останалите подходи, където се използва история на изпълнението е необходимо да се разбере програмното поведение когато имаме инструкция за преход – а именно че условните преходи се предсказват повече от 50% - тогава естествено че тази техника дава по-добри резултати. Обаче при страницирана ВО този метод на извличане и предсказване на предизвиква page fault.
В заключение статичните подходи взимат решение на базата на операционния код на инструкциите за преходи. Процесорът приема че прехода ще бъде предсказан за текущата инструкция за преход но не и за другите. Тази стратегия според проучванията дава до 75% производителност.
Динамичното предсказване на преходите се базира на воденето на статистика за историята на преходите в програмата.Например един или повече битове могат да бъдат свързани към инструкцията за преход която ще рефлектира върху текущата история. Тези битове са обозначени като предсказан/непредсказан switch който упътва процесорът да всеме самостоятелно решението при следващата инструкция за преход. Обикновенно тези битове за историята не са свързани с инструкциите от ОП. Единият

вариант е тези битове да са добавени в инструкциите в кеша и когато инструкцията се замести , то нейната история ще се загуби. Друга възможност е да се поддържа малка таблица за настоящите инструкции за преход, които ще се отбелязват с 1бит. Процесорът може да се обръща асоциативно към тази таблица.

Използват се битове с който да се отбелязва кога е предсказан прехода. Тези битове се обозначават като taken/not taken. Самите битове не са прибавени към инструкциите от ОП. Затова те се пазят в бърза памет. Първата възможмостта е да свържете тези битове с някоя инструкция за условен преход от кеша. Когато инструкцията се премахне от кеша нейния бит за история се губи. Друга възможност е да поддържа малка таблица за текущо изпълняваните инструкции за преход с един или повече бита за история. Процесорът може да достъпе таблицата асоциативно, както при кеша или чрез младшите битове от адреса на бранч инструкцията.

С един бит могат да се отбележат последните изпълнени инструкции дали са предсказани или не. Или накратко казано използването на един бит се прилага при инструкции за които предсказването е сигурно – напр. Циклите.
Докато 2 бита се използват за запис на резултата от последните инструкциите. На следната картина е показан алгоритъма. Той започва от горния ъгъл. Ако първата инструкция се предскаже грешно алгоритъма продължава да предсказва докато следващата бранч не стане такен.Ако само два успешни бранча не се предскажат алгоритъма се премества към горната дясна страна на блок-схемата. И ще се вътри докато не стане бита преказан. Т.е. иска да има два грешно предсказани прехода за да се смени политиката да предсказване.
На следващата схема е показан автомат. Ако бранч логиката е решила да предскаже прехода целевата инструкция не може да се извлече докато не се декодира таргет адреса, който пък е операнд на инст за преход. Най-голяма ефективност може да бъде постигната ако инструкцията започне да се извлича веднага след предсказването на прехода. За тази цел трябва да се събира повече инфо която се съхранява в BHTable. Това е малка кеш памет, която е свързана с фазата на извличане на конвейрите. Всеки ред от таблицата се състои от 3 елемента: адрес на бранч инст; хистори битове които записват състоянието на използване на тази инструкция; и инфо за таргет инструкцията. Има различни реализации къде да бъдат тези елементи – единия вариант е да са в адреса на таргет инструкцията; друга възможност е самата таргет инструкция да я съдържа.