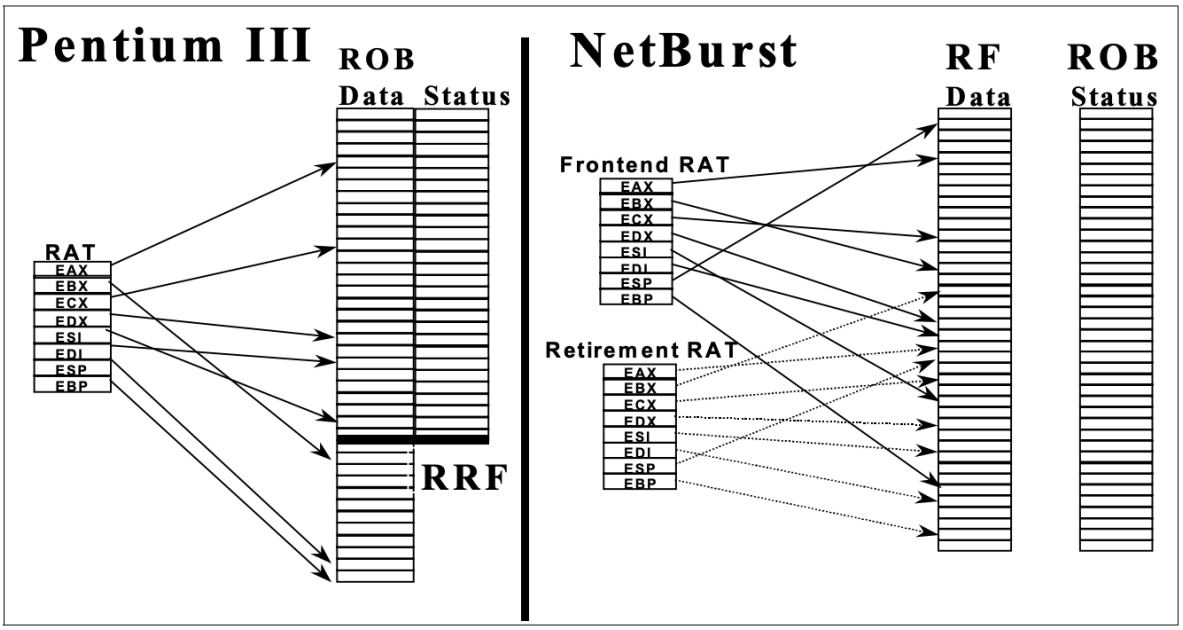

https://courses.cs.washington.edu/courses/cse378/10au/lectures/Pentium4Arch.pdf

Pentium 3'te geçici değerler PST(Re-Order Buffer) nin içinde tutuluyor ve yazmaçlar yeniden adlandırılırken bu PST yi işaret ediyor. Kullanılan numaralar bu tablonun numaraları. Ayrıca bir de mimari yazmaçları(RRF- Retirement Register File) var. ROB dan bitince RRF ye aktarıyor veriyi.

NetBurst Pentium 4'ün mimarisidir. Bir tane yazmaç birimi var(RF). Program sırasını tutan tablo ayrı(ROB). Veri aktarımı olmasın diye tek bir yerde tek bir yazmaç tutuyorlar ama 2 tane tablo var. 1 tanesi ön tarafta olan tahmine dayalı(Frontend RAT), tahmine dayalı denilen değişebilir, geçici değerleri gösteren tablo, bir de dışarıdan görülen yazmaçların nerede olduğunu gösteren tablo(Retirement RAT). Retirement RAT'ı nasıl kullanabiliyoruz? Bekliyoruz bir hata olana kadar, hata olduğunda en yaşlı buyruğa geliyor, en yaşlı buyruğa gelince bütün geri kalan buyrukları atıyoruz, o sırada Retirement Rat değeri ile Frontend RAT değerleri aynı oluyor. Retirement RAT'i Frontend RAT'a kopyalıyoruz. Atılanlardan sonra yeni gelen buyruklar yazmaçların yerlerini biliyor.

3 ve 4'ün farkı, 3'te ROB diye bir şey var değer saklıyor, pentium 4'te değer saklamıyor ve 3'te ayrıca bir mimari yazmaçlarının olduğu tablo var. Pentium 4'te tek bir yazmaç birimi var ve mimari yazmaçların ne olduğu Retirement RAT'ta tutuluyor.

Çok Yollu(-ing. Superscalar) İşlemciler

Her çevrim birden çok buyruk getir, çöz, yürüt, tamamla. Biraz geri saralım burada, Program sırasını dağıtmadan önce yapılan bir şey var. Program sırasını neden önce anlattık, çünkü Tomasulo 1967'de ilk program sırasını dağıtmayı söylüyor işlemcileri hızlandırmak için. Ama intel 486'dan sonra ilk iş program sırasını dağıtmıyor, ilk iş biz önce bir vuruşta birden fazla buyruk getirelim diyor. Buna superscalar deniyor. Yani bir vuruşta birden fazla buyruk getiren çok yollu işlemci. İlk pentium'lar 2'şer 2'şer getiriyor, Pentium 4'e gelindiğinde 3'er 3'er getiriyor. Boru hattının içerisinde koyduğumuz bekleme alanlarına(reservation station) ne kadar fazla buyruk sokarsak ve PST'yi ne kadar büyük yaparsak programda o kadar ileriye bakabiliriz. Ve o kadar birbirinden bağımsız buyrukları bulma olasılığımız artıyor.

- N-yollu boru hattı → Çevrim başına N buyruk

- aynı anda gelen buyruklar arasındaki bağımlılığı donanım denetler:
    - 2 tane buyruk aldık ve 2'si birbirine bağlı ise ne olacak? Birinin diğerinden sonra gitmesi gerekecek yani öndeki yürüyecek arkadaki duracak, yine boru hattını durdurmak gerecek. O yüzden donanımın aldığı bu birden fazla buyrukları kendi içerisinde denetlemesi gerekiyor. Ve örneğin 3 tane buyruk çekildi ve 2. buyruk dallanma buyruğu çıktı. Eğer 2.buyruğun öngörüsü atlayan dallanma ise getirme işlemi durdurulur genelde(instructions fetch stops after first taken branch). Çünkü dallanma buyruğu gelip atlamaya kalktığı zaman, 3. gelen buyruk anlamsız oluyor ve orda durduruyorsun 3. olan buyruğu atıyorsun. Bu şekilde yazmaç anlamlandırması yaparken tabloyu hem güncelleyeceksin hem de aynı anda okuyacaksın birden fazla buyrukla ve bundan dolayı daha fazla kapıdan okuyup yazmak gerekecek. Karmaşıklık artmış olacak. 2 tane boru hattı yapmaktan daha karmaşık buyrukların birbiri ile olan bağımlılıklarını denetlemek.
    - yazmaç yeniden adlandırma yönteminin uygulaması karmaşık

- Çok yollu işlemciler ***sıralı ve sırasız yürütme*** gerçekleştirebilir.
- Tek yollu işlemciler sırasız yürütme gerçekleştirebilir.
    - Tomasulo Örneği

- Tek yollu işlemciler sıralı yürütme gerçekleştirebilir.
    - ilk boru hattı örneği

Çok Yollu, Sıralı Yürüten İşlemci

Aynı çevrimde birden çok buyruğu yürütebilmek için karmaşık donanım gerekir: Denetim biriminin bağımlılıkları anlaması gerekiyor. Yazmaçları da okurken birden fazla buyrukla aynı anda okumamız gerektiği için bütün saklama alanlarının karmaşıklığı artıyor. Hepsine aynı anda birden fazla kapı koymamız gerekiyor. Elbette bu karmaşıklığı oldukça arttırır. Daha Sonra işlemcileri 2-yollu, 3-yollu, 4-yollu yapmışlar ve fark etmişler ki arttırsak da sonsuz gitsek de artık başarım artışında belli bir yere geliyoruz. Buyruk düzeyinde koşutluğun bir sınır var. Program içindeki buyruklar bir süre sonra bağımsızlığı bulamıyor ve bir yerden sonra bağımlı hale geliyor. Artık belli bir yerden sonra çok buyruk çekmenin bir anlamı olmadığı için çok çekirdekli işlemcilere geçildi. Tek başına çalışan işlemciler bir süre sonra artık başarımı arttıramadığı için, yani buradan daha fazla transistör koyarak sağlanamadığı için bir çekirdek daha konması düşünülmüş.

2-yollu işlemciden bahsediliyorsa ÇBB en fazla 2 olur. ÇBB → çevrim başına buyruk. Yani her vuruşta her yolunda giderse 2 buyruk işleniyor.

Buyruk Düzeyinde Çok İş Parçacığı Çalıştırma(-ing. Fine-Grained Multithreading)

Donanımda her iş parçacığı için program sayacı ve yazmaç öbekleri ayrı ayrı bulundurulur. Bir işlemcinin içerisinde birden fazla iş parçağının çalıştırılma yöntemi. Bu yöntemi kullanmak genellikle program sırasını dağıtma ile beraber oluyor. Program sırasının dışında çalışıyorsun, amaç bir program içerisinde bağımsızlığı bulmak. Ama bir yerden sonra bu bağımsızlık bulunamıyor, eğer elimizde aynı işlemcinin içinde 2 tane program olursa bu 2 programın birbirinden bağımsız olacağı bellidir çünkü buyrukları birbirinden bağımsızdır. Bir programda tıkanırsak diğer programdan içeriyi(işlem birimlerini) doldurabiliyoruz. Bunu yapmanın birden fazla yöntemi var. Şu anda bahsedilen fine-grained multithreading. Bir o iş parçacığından bir o iş parçacığından getiriyor. Böyle yapmak için

işlemci içerisinde 2 tane program sayacı tutulması gerekiyor. Yazmaç birimi aynı olabilir Pentium 4'te olduğu gibi ama yazmaçları yeniden adlandırma tablosu(YAT) 2 tane olması gerekiyor. Fine-grained dediğimiz ince-ayar. Bir ondan bir ondan al ki sürekli işlemcinin içinde birden fazla programdan buyruklar olsun ve birbirinden bağımsız olduklarına emin olalım. Bekleme alanlarında da bu buyrukların hazır olup işlem birimlerini, işlem kaynaklarını kullanmalarının yolunu açalım. Buradaki amaç budur.

- İşlemci her çevrim bir diğer iş parçacığından buyruk getirir.
- dallanmalara bağlı gecikmeler diğer iş parçacıklarından getirilen buyrukların gecikmeleri ile çakışır.

+ Denetim ve veri bağımlılıklarını takip etmek için donanım gerekmez. Her çevrim farklı iş parçacığından buyruklar getiriliyor, yazmaç öbekleri ayrı. Sadece buyrukların hangi programa ait olduklarını denetlemek gerekir. PST'den 2 tane olması gerekir. Yazmaç öbekleri eğer mimari yazmacı kullanıyorsanız ayrıdır, Pentium 4 gibi tek bir yazmaç öbeği kullanıyorsanız ayrı olması gerekmez. Bu durumda da yeniden adlandırma yapılan tablolar ayrıdır.

Birden fazla iş parçacığını aynı anda işlemcinin içine koyarsanız, diyelim işlem birimi 4 tane, ikisini birisi ikisini birisi kullanabilir. Olağan şartlarda bir iş parçacığı yani 1 program 1 işlemciyi kullandığında belki o işlem birimlerinin hepsini(dördünü) birden kullanacaktı. Başka bir programla birlikte çalışıyor diye onun başarımı düşebilir. Yani Tek bir iş parçacığının başarımı düşebilir ama 2 iş parçacığını beraber çalıştırıldığında toplamda daha verimli olabilir.

- tek iş parçacığının başarımı düşük
- yazmaç öbekleri, program sayacı gibi iş parçacığına özel birimlerin kapladığı toplam alan artar.

Simultaneous Multithreading(-Intel. Hyperthreading)

Çok yollu işlemcilerde tek iş parçacığı tüm yürütme birimlerini yüksek verimlilikle kullanamayabilir.

- Buyruk düzeyinde koşutluk düşükse

Ana fikir: Yürütme birimlerini ve bazı diğer kaynakları (örn. dallanma öngörücü, yükle/kaydet kuyruğu) iş parçacıkları arasında paylaştır. Birden çok iş parçacığından gelen buyrukları aynı anda ***yürüt***.

- İş parçacıklarına özel birimler hala tutuluyor: yazmaç öbeği, program sayacı

It is not power efficient, it is termal efficient.