BİLGİSAYAR MİMARİSİNDE YENİ YAKLAŞIMLAR

Prof. Dr. Bülent Örencik

DÖNEM ÖDEVİ

ÖLÇEKLENEBİLİR MİMARİLER

Hazırlayan :
Zeynep Yurdakul
704061015

Aralık, 2007

1. GİRİŞ

Ölçeklenebirlik, bir bileşenin (donanım yada yazılım) gereksinimler doğrultusunda boyutunun veya kapasitesinin artırılmasına karşılık fonksiyonel özelliklerini aynı şekilde yerine getirebilmesidir. Ölçeklenebilirlik kavramı, yatay ve dikey olmak üzere iki şekilde düşünebilir. Yatay ölçeklenebilirlik, varolan sisteme aynı yada benzer bileşenlerin paralel olarak eklenerek, toplam sistem kapasitesinin artırılmasıdır. Dikey ölçeklenebilirlik, varolan sistem kaynaklarının gücünü, kapasitesini yada hızını artırabilmektir.

Ölçeklenebilir işlemci mimarisi : SPARC.

SPARC(Ölçeklenebilir İşlemci Mimarisi) bir RISC mikro işlemci komut kümesi mimarisidir ve 1985 yılında Sun Microsystems tarafından tasarlanmıştır. SPARC'ın tescilli ticaret markası olan SPARC International, Inc., bir organizasyon olarak 1989 yılında SPARC' ı duyurmak ve uyum testlerini yürütmek amacıyla kurulmuştur. SPARC International Sparc mimarisini, içinde Texas Instruments, Cypress Semiconductor, ve Fujitsu' nun da yeraldığı birkaç üreticiye lisanslı olan dizayn için büyük bir ekosistem yapmak amacıyla "açık" hale getirmeyi planladılar. SPARC International' in bir sonucu olarak, SPARC mimarisi tamamıyla açık ve sahipsizdir.

SPARC bir işlemci tasarım spesifikasyonu olarak görülebilir. Bu spesifikasyon bir şablon şeklindedir ve SPARC mimarisini uygulamasında kullanacak olanlar, Sun'un UltraSPARC ile yapmakta olduğu gibi, bu spesifikasyondan gerçek bir donanım ürünü oluştururlar.

SPARC Versiyon 8 (V8), standart 32-bit SPARC mimari tanımı, SPARC International tarafından 1994 yılında piyasaya çıkarılmıştır. 2006' nın başlarında, Sun süren mimari çalışmalarını piyasaya sürdü, UltraSPARC Mimari 2005. UltraSPARC Mimari 2005, sadece ayrıcalıksız ve biçok bölümü ayrıcalıklı olan SPARC V9' u değil, aynı zamanda bütün mimari uzanımlarıda kapsıyordu.

SPARC mimarisinin gerçekleştirilmesi özelleştirilmiş ve özelleştirilmemiş bileşenler olarak ikiye ayrılır: Özelleştirilmemiş bileşenleri gerçekleyen işlemciler SPARC kullanıcı uzayı uygulamalarını çalıştırabilirken, genel amaçlı SPARC işletim sistemini çalıştıramazlar.

SPARC isimlendirmesindeki ölçeklenebilirlik, SPARC spesifikasyonunda belirtildiği üzere aynı çekirdek komut kümesinin gömülü sistemlerden büyük sunucu işlemcilerine kadar farklı büyüklüklerdeki gerçeklenmelerde kullanılabilir olmasından ileri gelmektedir. Ölçeklenebilir olan mimarisel parametrelerden biri de gerçeklenen saklayıcı penceresi sayısıdır: Spesifikasyona göre 3 ila 32 arasında pencere gerçeklenebilir. Eğer tasarımda 32 pencerenin tamamına yer verilirse fonksiyon çağrısı yığını en yüksek verimlilik seviyesinde çalışacaktır yada 3 pencere tercih edilirse konteks değişim zamanı en kısa süreye inecektir.

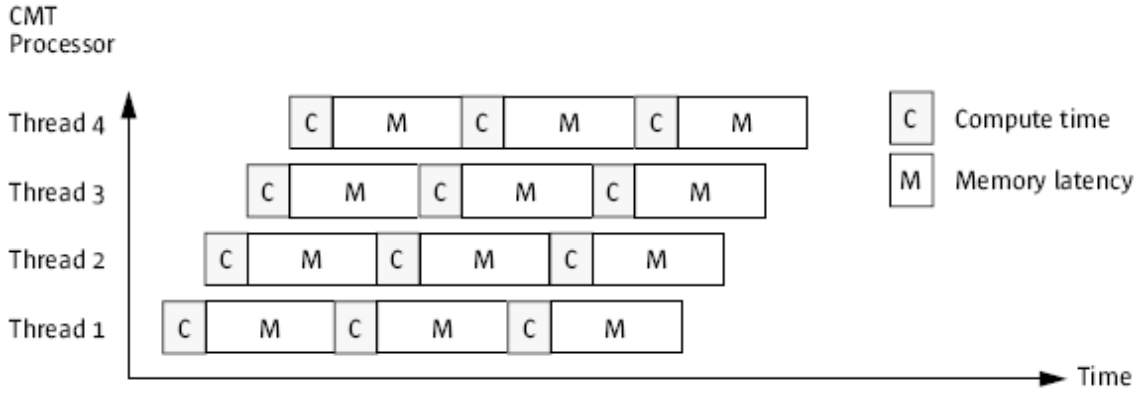
32 bitlik bir mimari olan SPARC v8, bir big-endian mimaridir. 64 bitlik olan SPARC v9 mimarisi big-endian komutları kullanır ancak veriye hem big-endian hem de little-endian dizilimiyle erişebilir. Hangi erişim metodunun kullanacağı ya uygulama komutu (load/store) seviyesinde yada bellek sayfalaması seviyesinde (MMU tarafından) seçilir. İkinci durum daha ziyade little-endian olan cihazlara –örneğin PCI veriyoluna bağlı olanlar- erişmek için kullanılır.

2. SPARC Mimarisi

Aşağıda SPARC mimarisinin ölçeklenebilirlik açısından önemli özellikleri incelenmiştir.

2.1 Donanımsal Çok İşçiklilik

Klasik tek işçikli işlemcilerden ve hatta yeni nesil çok çekirdekli işlemcilerden farklı olarak, donanımsal çok işçikli işlemciler, , bir işçik bellek erişimi için duraksadığında diğer işçikleri devam ettirerek aktif işçikler arasında yapılan geçişlerin oldukça hızlı olmasını sağlar. Şekil 2.1 'de teorik olarak bir işçik bellek erişimi için durakladığında dört kadar farklı işçik arasında geçiş yapabilen bir çok işçikli işlemci çekirdeğinin çalışması gösterilmiştir. Bu yaklaşımla işlemcinin çalışma iş hattı duraksamış işçikler için bellek işlemleri devam ederken aktif kalarak işe yarayan işlemler yapabilmektedir.



Şekil 2.1 Çok İşçikli İşlemci Çekirdeğinin Çalışması

2.2 Kırmık çok işçikliliği (CMT)

Aynı tek işçikli işlemcilerde olduğu gibi , çoklu işlem (çoklu çekirdek) teknolojisi donanımsal çok işçikliğinin faydalarını artırmak ve ölçeklemek için kullanılabilir. Bu uygulamaya Sun kırmık çok işçikliliği (chip multithreading – CMT) adını vermiştir. Karmaşık tek işçikli işlemcilerin aksine, CMT işlemcileri kullanılabilir transistör miktarından yararlanarak birden fazla çok işçikli işlemci çekirdeğini tek bir kırmık üzerinde oluştururlar. Bu ayrık işlemci çekirdekleri çok daha basit iş hatları (ağırlıklı olarak TLP üzerine ILP) oluşturduklarından dolayı hem de daha az enerji ile çalışırlar ve daha az ısı üretirler. CMP teknolojisi kırmık üzerinde bir çok işçikli Simetrik Çokluişlem(SMP) sistemi oluşturulmasını gerekli kılar.

2.3 Komut Seti

SPARC'ın komut seti, RISC mimarisinin bir çeşit gerçeklenmesidir. İşlemlerin yalnızca saklayıcılarda tutulan değerler üzerinden yapıldığı bir yükle-sakla tasarımına dayanmaktadır. Komutlar basittir, genelde iki girdi bir çıktıdan oluşurlar ve 32 bitlik sabit uzunluklu işlem kodlarından oluşurlar.

Yükleme ve saklama için adresleme iki şekilde yapılır:

- saklayıcı + saklayıcı
- saklayıcı + 13 bit işaretli değer

32 bitlik komut uzunluğu kod boyutunu düşük tutmak için verimli olsa da 32 bitlik değerler bir saklayıcıya doğrudan yazılamazlar. Bu nedenle SPARC bir saklayıcının üst 22 bitine yazmak için *sethi* isimli bir komut sağlar.

Örneğin Şekil 2.2'de 32 bitlik değer iki parçada yüklenmektedir; 22 bit g1 saklayıcısına seth1 komutuyla yüklenir, kalan 10 bit ise *or* komutunun sağladığı maske ile yüklenir.
SPARC kendisiyle aynı dönemde tasarlanmış, diğer kayda değer mimarilere (örneğin MIPS) benzer şekilde iş hattı yapısında gecikmeli dallanma kullanır. Şekil 2.3'deki örnekte de dolu olmayan dallanma gecikmesi slotlarını görebiliriz.

SPARC v8 aynı zamanda karşılaştırmalar için koşul kodları yöntemini kullanır. Ancak daha sonra ortaya çıkan mimarilerde bu teknikten, darboğazlar oluşturması nedeniyle kaçınılmıştır. Bu handikapı azaltmak için aritmetik işlemler iki sınıfa ayırılır: koşul koduna bakılanlar ve bakılmayanlar. SPARC v9 karşılaştırma işlemlerini tam sayı tutan saklayıcılar üzerinde de yapabilen komutlar tanımlamıştır.

Komut setini küçük tutabilmek için, çoğu komut *sentetik komut* olarak gerçekleştirilmiştir, yani semantik olarak aynı olan ancak daha az sezgisel olan komutlara indirgenirler. Şekil 2.3'de de görülebileceği üzere *cmp* komutunun çözülmüş halinin işlem alanı 010100'e açılmıştır, bu da *subcc* (çıkar ve koşul kodunu ayarla) işleminin işlem kodudur. Dolayısıyla cmp aslında subcc %g1, 0, %g0, için bir sentetik komuttur ve ardından gelen dallanma işlemi için koşul kodunu set eder.

```
void addr(void) {
int i = 0xdeadbeef;
}

00000054 <addr>:
54: 9d e3 bf 90     save %sp, -112, %sp
58: 03 37 ab 6f     sethi %hi(0xdeadbc00), %g1
5c: 82 10 62 ef     or %g1, 0x2ef, %g1  ! deadbeef <addr+0xdeadbe9b>
60: c2 27 bf f4     st %g1, [ %fp + -12 ]
64: 81 e8 00 00     restore
68: 81 c3 e0 08     retl
6c: 01 00 00 00     nop
```

Şekil 2.2: SPARC üzerinde çalışan basit bir fonksiyonun disassembly hali

2.4 Saklayıcılar

SPARC saklayıcı pencelerini gerçekleştiren ilk ticari işlemci mimarisidir. Bu teknik saklayıcıları soyutlayarak fonksiyon giriş ve çıkışlarında kaydedilmeleri ve geri alınmaları zorunluluğunu ortadan kaldırır. Saklayıcıların tamamına saklayıcı dosyası adı verilir. Anlık çalışmakta olan fonksiyon tarafından erişilebilir saklayıcılara saklayıcı penceresi adı verilir.

SPARC saklayıcı penceresi şeması; analizler sonucu ortaya çıkan, fonksiyonların genelde altı veya daha az giriş parametresi alması ve sadece birkaç seviye iç içe geçmesi gerçeğinin kullanılmasıyla oluşturulmuştur. Şekil 2.4'de gösterildiği gibi 32 genel amaçlı tam sayı saklayıcıya işlemci her an ulaşabilir:

Genel	Pencereli	Tanım
%r0 - %r7	%g0 - %g7	Global Saklayıcılar (tüm pencelerelerden erişilebilir)
%r8 - %r15	%o0 - %o7	Pencere çıktı saklayıcıları
%r16 - %r23	%l0 - % l7	Pencere yerel saklayıcıları
%r24 - %r31	%i0 - %i7	Pencere girdi saklayıcıları

Konsept olarak, bir fonksiyon çağrısı yapıldığında çağrıyı yapan fonksiyonun çıktı saklayıcıları çağrılan fonksiyon için girdi saklayıcıları olurlar. Saklayıcıların pencereli kısaltmaları (g, o, l, i) assembler (çevirici) tarafından programcıya yardımcı olması için sağlanırlar. İşlemci, genel isimleri (r), saklayıcı dosyası içindeki gerçek fiziksel saklayıcılara denk gelecek şekilde ele alır.

Saklayıcı penceresinin döngüsü *save* ve *restore* komutlarıyla programcının kontrolündedir. Bu derleyicinin bir fonksiyonun daha çok saklayıcıya erişmesine izin vermek için saklayıcı penceresini taşımasına izin vermesi nedeniyle, yalnızca prosedür giriş ve çıkışlarında saklayıcı penceresinin taşınmasına izin veren Berkeley RISC mimarisine ters bir uygulamadır.

Şekil 2.4'de gösterilen doğrusal dizilimden ziyade saklayıcı penceresi döngüsel bir şekilde çalışır. SPARC v8'de işlemci o andaki aktif pencereye işaretçi tutar (CWP). Bir *save* komutuyla saklayıcılar yeni bir pencere oluşturacak şekilde yeniden isimlendirilirler. Bir *restore* komutu ise bunun tam tersini yapar.

Döngüsel tampon bellek sonsuz olmadığından, birden fazla *save* işlemi yapıldığında eski değerlerin üzerine yazılır. Bu duruma taşma (overflow) adı verilir ve bu durumda bir taşma hatası yaratılır. Bu noktada kontrol tekrar işletim sistemine verilir ve pencerede bulunan üzerlerine yazılmak üzere olan değerler bölünerek bellekte kalması sağlanır.

```
int branch(int i)
{
 if (i == 0)
    return 0;
 else
    return i;
}
```

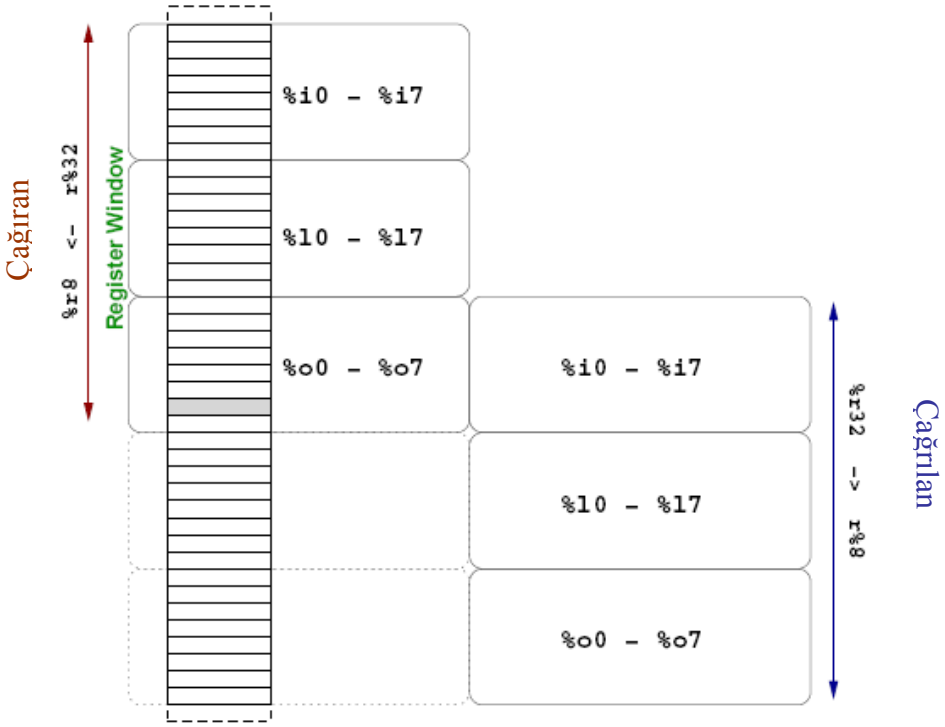
```
00000070 <branch>:
70: 9d e3 bf 90          save %sp, -112, %sp
74: f0 27 a0 44          st     %i0, [ %fp + 0x44 ]
78: c2 07 a0 44          ld     [ %fp + 0x44 ], %g1
7c: 80 a0 60 00          cmp  %g1, 0
80: 12 80 00 05   bne    94 <branch+0x24>
84: 01 00 00 00   nop
88: c0 27 bf f4          clr     [ %fp + -12 ]
8c: 10 80 00 04          b       9c <branch+0x2c>
90: 01 00 00 00   nop
94: c2 07 a0 44          ld      [ %fp + 0x44 ], %g1
98: c2 27 bf f4          st      %g1, [ %fp + -12 ]
9c: c2 07 bf f4          ld      [ %fp + -12 ], %g1
a0: b0 10 00 01   mov    %g1, %i0
a4: 81 e8 00 00   restore
a8: 81 c3 e0 08          retl
ac: 01 00 00 00          nop
```

```
Input Bitmap Type > sparc_int_op_rs1_rs2_rd

sparc_int_op_rs1_rs2_rd value > 0x80a06000

Decoded output for a Sparc Integer manipulation: register, register, destination
            Register 2 | 0 0000
               Unused | 0000 0000
                 Zero | 0
            Register 1 | 00 0011
            Operation  | 01 0100
Destination Register | 0 0000
                   10 | 10
```

Şekil 2.3 : SPARC üzerinde karşılaştırma, dallanma ve dallanma gecikmesi slotlarını gösteren bir disassebmly örneği . En alttaki cmp komutunun çözümlenmesinin gerçekte subcc için bir sentetik komut olduğu görülmektedir.



Şekil 2.4 SPARC saklayıcı pencerelerinin görünümü

Bu durum saklayıcıların nereye açılacağı sorusuna da yol açar. Kural olarak fonskiyonun dış saklayıcılarından biri yığın işaretçisi olarak kullanılır (%sp, Şekil 2.4'de gri ile gösterilmiştir). Assembler her bir fonksiyona kendi saklayıcı penceresini saklamak için yeterince alan sağlamakla sorumludur.

Restore işlemleri genelde store işlemlerinin tersidir, ve bir underflow hatası oluşturabilir. Underflow durumunda tekrar yığın işaretçişi takip edilir ancak bu sefer kontrolü program geri vermeden önce pencerenin içerisine saklayıcıları tekrar alırken aynı zamanda saklayıcıları da doldururuz.

2.5 SPARC v8 Saklayıcı Pencereleri

SPARC v8, işlemci durum saklayıcısında en fazla 5 bitlik bir alanı o anki pencereyi (anlık pencere işaretçisi - CWP) saklamak için kullanır. Bu nedenle en çok kullanılacabilecek

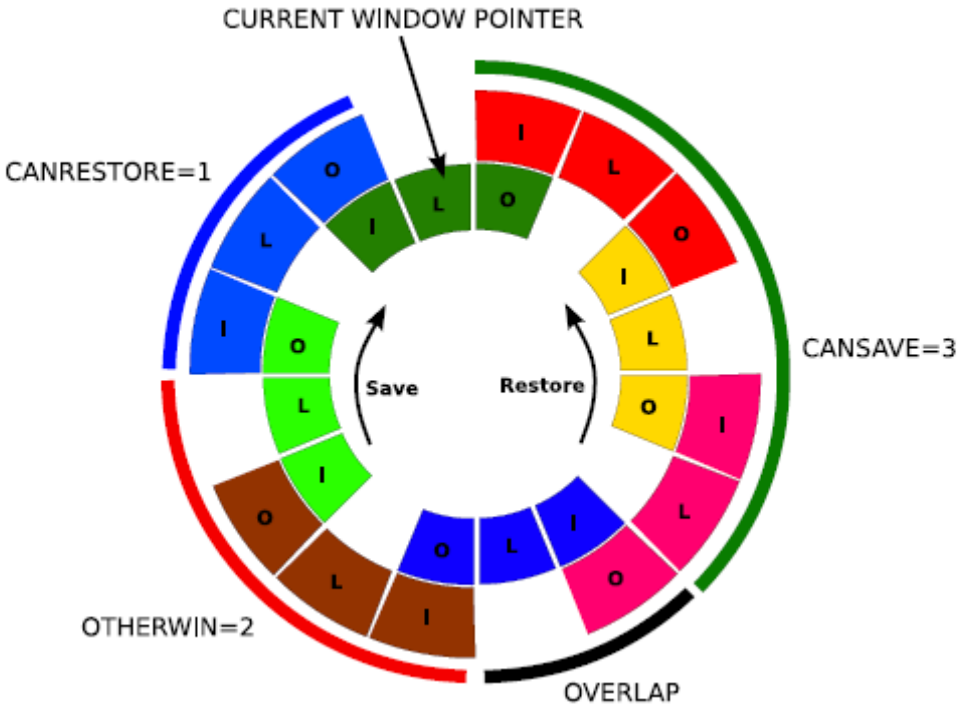
saklayıcı penceresi genelde daha az olabilse de sayısı 32'dir. Bunlarla ilgili bulunan bir eşleştirme tablsounda geçersiz saklayıcı pencerelerinin (Window Invalid Mask - WIM) bir kaydı tutulur.

İşlemciler üzerinde bulunan ve durumu daha karmaşık hale getiren bir başka etken de işlemcinin çökmesi (traplemesi) durumunda saklayıcı penceresi koşulsuz olarak yer değiştirir, ayrıca iç içe çökmeler desteklenmez (bir çökmenin sonunda başka bir çökme olamaz). Saklayıcıların üzerine yazmayı engellemek için bir çökme durumunda durumu kontrol edecek bir pencere her zaman rezerve olarak tutulmalıdır. Bunun için de bir pencere işletim sistemi tarafından geçersiz olarak işaretlenmelidir.

Bağlam (context) değişiminde, diğer işlerin saklayıcı pencereleri, geçersiz olarak işaretlenir. CWP her *save* işlemiyle birlikte bir azaltılır. Yeni pencerenin geçerliliği WIM'e karşı kontrol edilir ve eğer geçersiz bir pencereyse bir trap oluşturulur. Benzer bir işlem *restore* için de gerçekleşir.

2.6 SPARC v9 Saklayıcı Pencereleri

SPARC v9 ile birlikte saklayıcı penceresinin çalışmasının çalışma şeklinde birtakım değişiklikler yapılmıştır. İlk olarak, v8'de iç içe trap yaratılamamaktadır ve bir saklayıcı penceresi trapte kullanılabilmesi için kullanılmadan kenarda tutulmalıdır. V9'da, trap yakalayıcılara ayrılmış fazladan global saklayıcılar eklenerek 5 seviye içiçe trap oluşması desteklenecek şekilde mimaride değişiklik yapılmıştır.



Şekil 2.5 SPARC V9 Saklayıcı Pencereleri

V8'de bulunan ana problemlerden biri, WIM yapısının birden fazla adres uzayını tam olarak destekleyememesidir. Geçersiz saklayıcı pencerelerinin ve veri kayıplarının olmaması için pencere tamponun her bağlam değişikliğinde temizlenmesi gerekmektedir. Bu nedenle v9

saklayıcı pencerelerini yönetmek için gelişmiş bir yapı barındırır. Basit bir şekilde sadece geçerli-geçersiz olarak tutulan eşleştirme tablosu, her bir sınır saklayıcı serisi için aşağıdakilerle değiştirilmiştir:

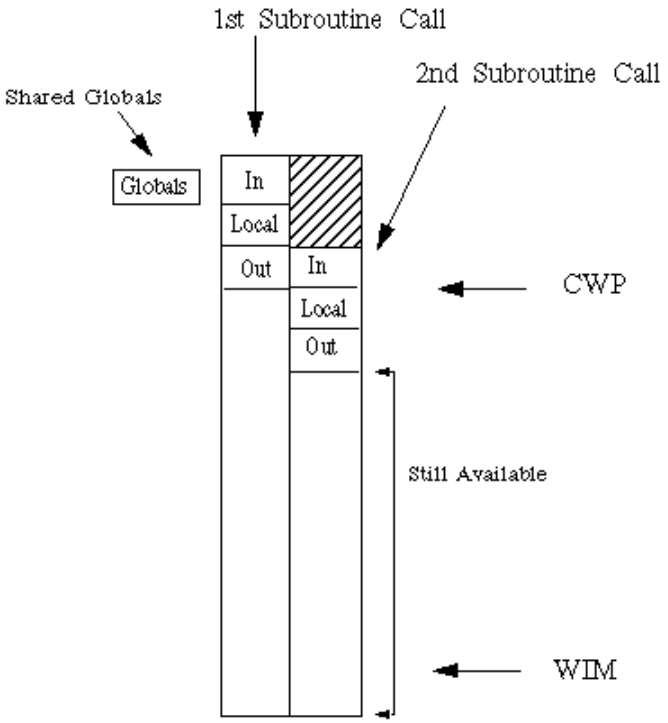
- CANSAVE: bir overflow trapi oluşana kadar gerçekleştirilen toplam *save* komutlarının sayısı
- CANRESTORE: bir underflow trapi oluşana kadar gerçekleştirilen toplam *restore* komutlarının sayısı
- OTHERWIN: *alternatif* spill (dağıtma) /fill (birleştirme) aykırı durumu oluşturacak pencerelerin sayısı

Bu mekanizmanın birkaç yararı vardır. Eğer kullanıcı OTHERWIN pencerelerine save veya restore yapmaya çalışırsa, alternatif hata yakalayıcıları çağırılabilir.Bu şekilde daha hızlı trap ayıklayıcılar ve daha iyi işlemler arası güvenli geçiş sağlanabilir. Bu mekanizma daha hızlı sistem çağrıları gerçekleşmesine de olanak sağlar. Örneğin; CANRESTORE değerini sıfıra ayarlayarak, sistem çağrılarından dönüşleri daha hızlı tespit edebilir ve daha hızlı trap ayıklayıcılar kullanılabilir.

2.7 Altprogram Saklayıcı Tahsisi

SPARC işlemcileri genelde 128 genel amaçlı saklayıcı bulundururlar. Bunlardan sadece 32 tanesi yazılım tarafından erişilebilirdir. Bunlar giriş (%i0 - %i7), çıkış (%o0 -%07), yerel (%l0 - %l7) ve global (%g0 - %g7) saklayıcılarından oluşurlar (g0 donanımsal olarak 0'a bağlıdır, bu nedenle sadece 7 tanesi saklayıcı olarak kullanıbilir. Global olmayan 24 saklayıcı, saklayıcı penceresi olarak adlandırılan yapıyı oluştururlar. Her bir altprogram çağrısında ve çıkışında bu pencere saklayıcı yığını üzerinde yukarı ve aşağı doğru kayar. Her bir pencerenin 8 yerel saklayıcısı ve komşu pencerelerle paylaşılan 8 saklayıcısı vardır. Paylaşılan saklayıcılar altprogram parametreleri aktarmak, değer döndürmek için kullanılırken, yerel saklayıcılar altprogram çağrıları esnasında yerel değişkenlerin korumak için kullanılırlar.

Bir altprogram çağrısı yapıldığında çıkış (%o0 - %o7) saklayıcıları tekrardan isimlendirilerek çağrılan altprogram için giriş (%i0 - %i7) saklayıcıları olarak tanmlanır. Ayrıca altprogram için yeni yerel (%l0 - %l7) ve çıkış (%o0 - %o7) saklayıcıları oluşturulur. Bu çıkış saklayıcıları daha sonra bu altprogram içerisinden çağrılacak başka bir altprogram için giriş saklayıcıları olacaklardır. Bu durum Şekil 2.6'da gösterilmiştir:

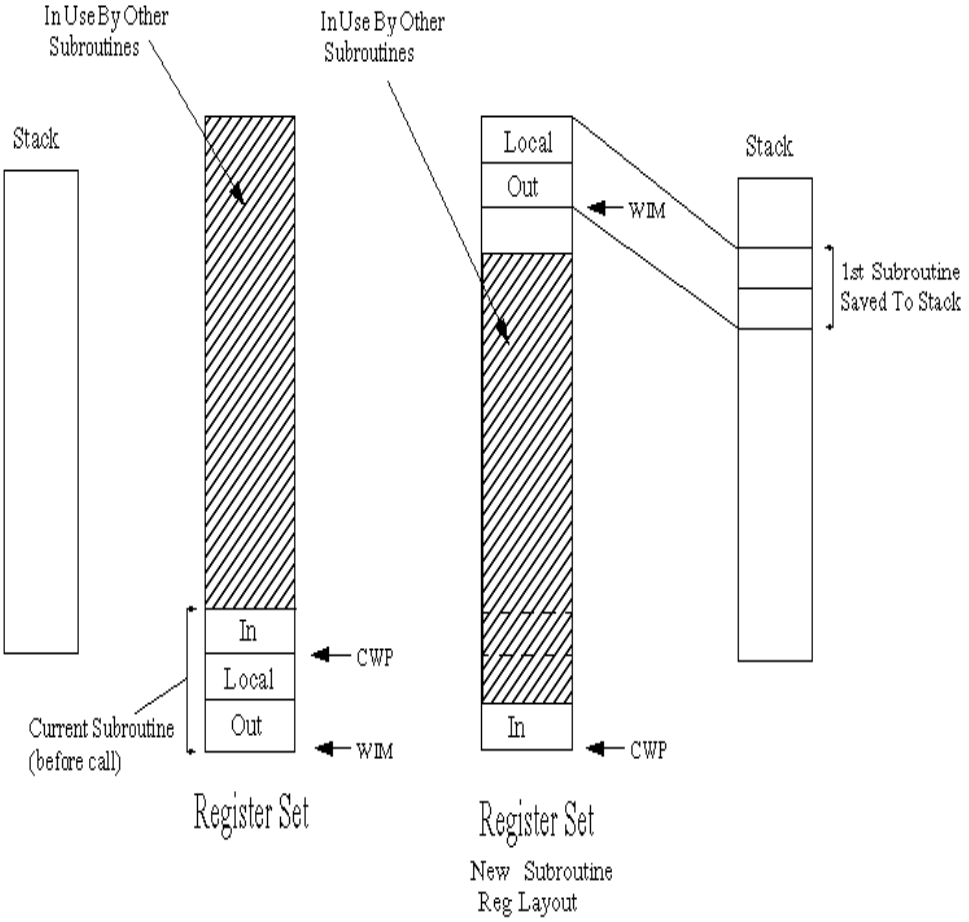


Şekil 2.6 Altprogram1 Altprogram2'i Çağırırken Yığının Gösterimi

Bu şekilde işlemci 8'den daha az parametre (%i0 - %i7) olması ve saklayıcı dosyasının dolu olmaması durumunda hızlı bir altprogram çağrısı yapılmasını garanti etmiş olur. Eğer altprograma çok fazla parametre geçiliyorsa, işlemci durumunun yığınına yazılım tarafından aktarılması gerekir. Eğer işlemci ayıracak yeni bir saklayıcı bulamazsa, en eski altprogram çağrısını yığına saklayarak ihtiyaç duyulan 32 saklayıcıyı serbest bırakmış olur. Bu serbest bırakılan saklayıcılar artık yeni altprogram için ayrılabilecektir.

SPARC işlemcisi saklayıcı dosyasının dolu olduğunu iki ek saklayıcı, "Anlık Pencere İşaretçisi - CWP" ve "Pencere Geçersizlik Biti - WIM", kullanarak algılar.CWP o anda kullanılmakta olan saklayıcılar kümesine işaret ederken, WIM en son kullanılabilir saklayıcı kümesini gösterir. CWP ile WIM aynı pencereyi işaret ederken, bir altprogram çağrısı yapılması durumunda, bir donanım trapi yaratılır ve işlemci en eski pencereyi yığına saklar ve pencereye ait saklayıcıları serbest bırakarak çağrılan altprograma bırakır. Saklayıcılar yığına kaydedilince WIM boşta kalan yeni saklayıcıları işaret edecek şekilde güncellenir.

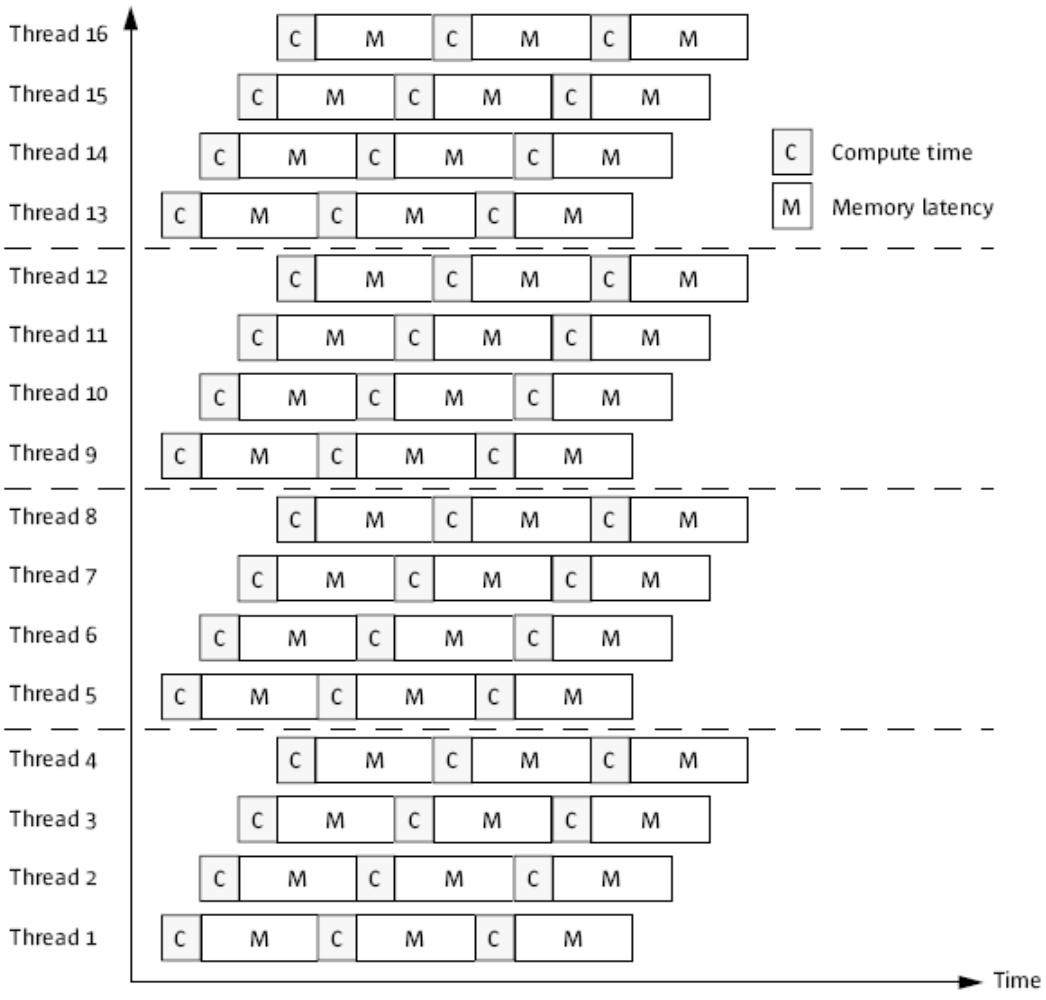
Saklayıcı dosyası Şekil 2.7'deki gibi gösterilebilir:



Şekil 2.7 Yığına saklayıcı taşmasının gösterimi

3. ULTRASPARC T1

UltraSPARC işlemcisinin birden fazla işçiği paralel çalıştırma yeteneğinin bir sonucu olarak bellek sistemi üzerinde daha fazla yük oluşur. Şekil 3.1'deki durum göz önüne alınırsa işçik1 için bir cep ıskası oluştuğunda, sonraki işçik (işçik 2) ve daha sonrasındakiler için de verinin hazır olduğu varsayılır. 32 işçiğin verisinin de hazır olmadığı en kötü senaryoyu düşünecek olursak, verilerin bellekten kırmığın cep belleğine taşınması esnasında toplam olarak 20.5 Gbayt/sn lik bir bantgenişliğine ihtiyaç olacaktır. Çoğu durumda 32 işçiğin hepsi de her çevrimde ana belleğe erişmeye çalışmayacaktır ve dolayısıyla daha düşük bir bantgenişliği ihtiyacı olacaktır.UltraSPARC T1 işlemcisinin 20 Gbayt/sn bellek bantgenişliğini destekleyen dört kırmık üstü bellek kontrolörü sayesinde, gerekli bantgenişliği ve T1 işlemcisinin verebildiği bantgenişliği dengesi sağlanır.

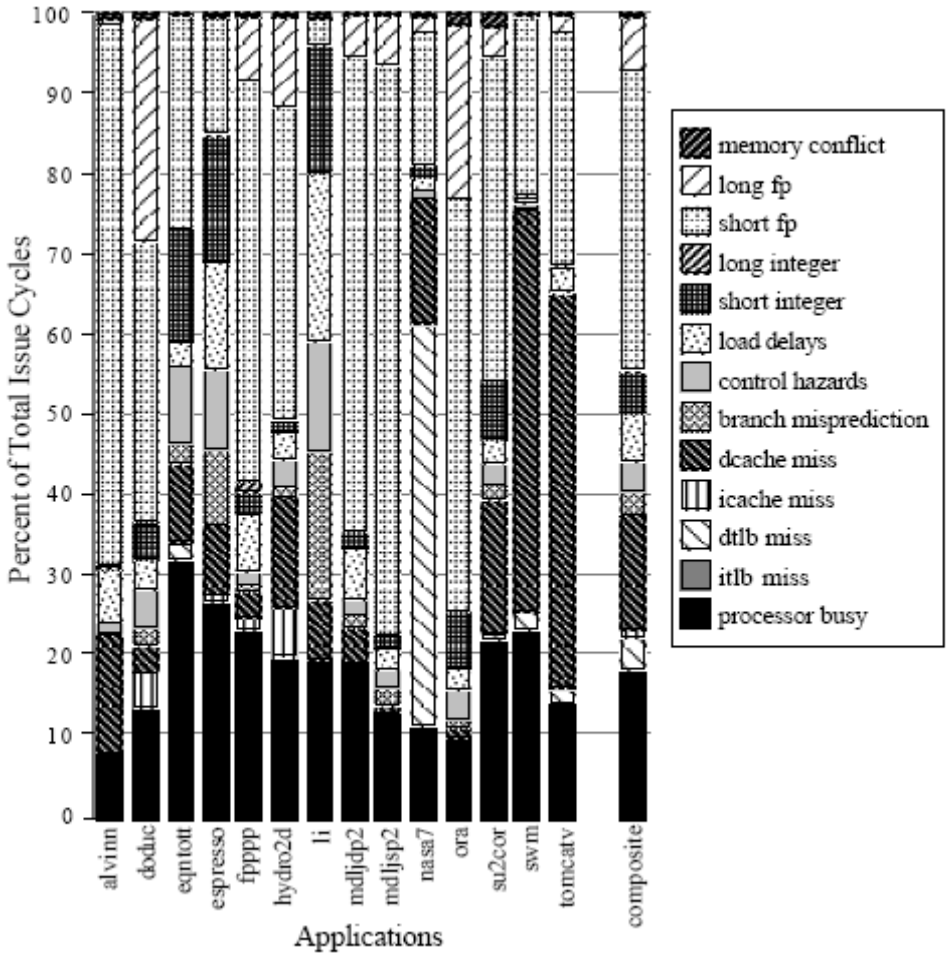


Şekil 3. 1 UltraSPARC T1 işlemcisinin çoklu işçikli kavramsal diyagramı. Onaltı aktif çalışma işçiğini destekleyen dört donanımsal çok işçikli çekirdeğin bir CMT işlemcinde birleştirilmiş halidir.

3.1. Verim (Throughput) Hesaplaması

Modern süperskaler işlemciler, mevcut fonksiyonel birimlerini çok iyi kullanmamaktadırlar. Süperskaler bir işlemcinin faydalı olması için, komut başına çevrim sayısı (CPI) 1'den daha küçük olmalıdır.

Şekil 3.2 teorik 8-issue süperskaler makine yapısını göstermektedir. Bu 8-issue süperskaler yapının 1'den küçük CPI değerine sahip olabilmesi için, potansiyel sorun olabilecek slotların 1/8' inden fazlası her bir çevrimde doldurulmalıdır(Birden fazla fonksiyonel birim üzerinde paralel çalışılıyor). Böylece potansiyel sorun slotların %12.5'ten fazlası doldurulabilir.



Şekil 3. 2 SPEC92 Testinde 8-issue süperskaler işhattı yapısının çevrimlerde kullanımı [6]

CPI'ın 1 veya daha büyük olduğu durumda, çoklu fonksiyonel birimlerin paralel kullanılmasının avantajı kalmamaktadır. Çünkü onu sürekli çalıştıracak yeteri kadar komut akışı sağlanamamaktadır. Ayrık komut akışlarında çalışacan tekil issue işlemcilerin birleştirilmesi daha verimli olmaktadır.

UltraSparc T1, simetrik çoklu işçiklerin (SMT) kullanımına öncülük etmiştir. SMT, kontrol ve saklayıcı mantığı, belirli sayıdaki yürütme birimini paylaşan işçikler için tekrarlanır. Yürütme birimini paylaşarak, belirli bir işçik yüksek CPI sergilemezse, başka bir işçik kullanılmayan kaynakları kullanmaya başlar.

UltraSparc T1, birden çok çekirdeği tek bir kırmık üzerinde 4-yollu SMT olarak birleştirmiştir. Sun firması, bu modele "Verim Hesaplaması" ismini vermiştir. Çünkü bireysel işçik performansındansa, toplam sistem veriminin yüksek olmasına odaklanmıştır. Bazı karşılaştırmalı değerlendirme testleri IPC'nin 5.76 olduğunu iddia ediyorlar [4] (CPI = 0.17).

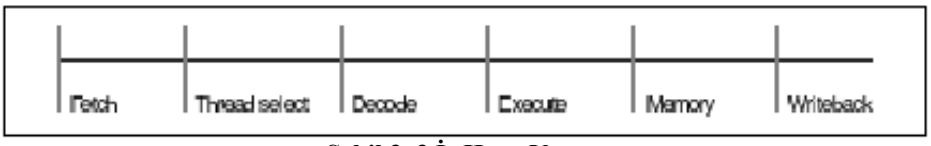
3.2. İşhattı Yapısı

UltraSparc T1 mimarisi, 8 çekirdek destekleyen bir yapıda oluşturulmuştur. Her bir çekirdek 16KiB 4-yollu kümeli çağrışımsal cebe, 32 byte büyüklüğünde L1I'ya ve 8KiB 4-yollu kümeli çağrışımsal cebe, 16KiB büyüklüğünde L1D'e sahiptir.

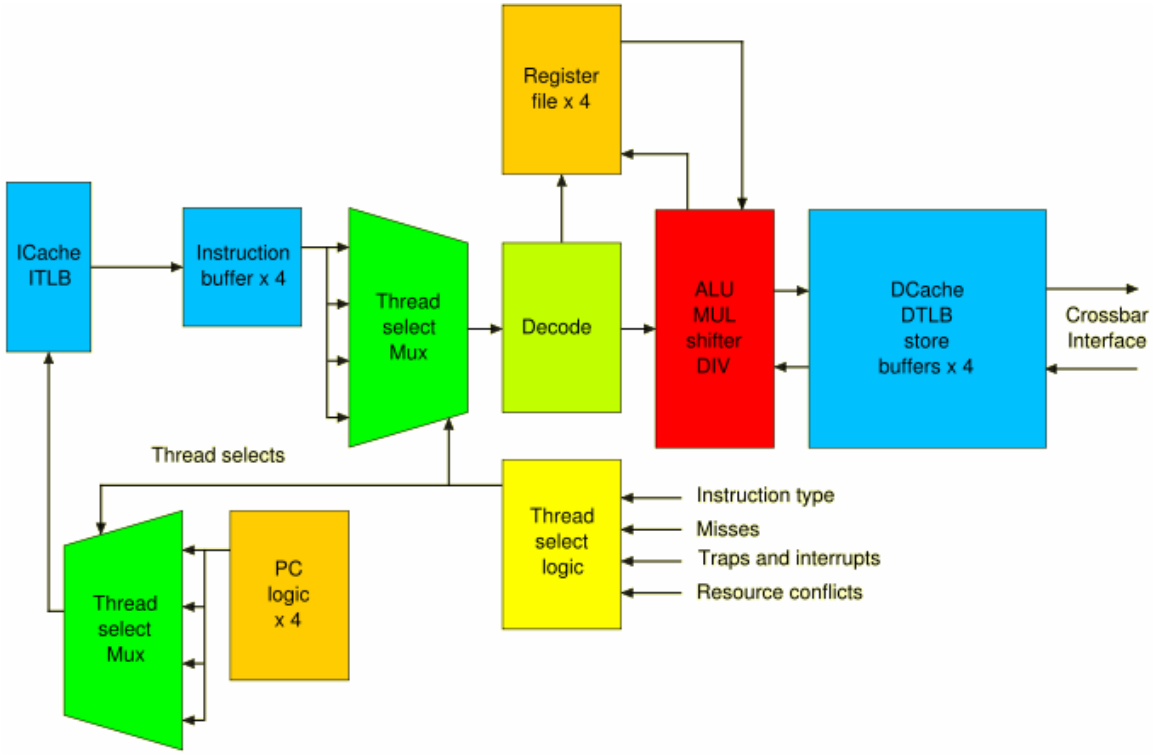
Sekiz çekirdeğin her biri 4 işçiği (işçik grubu) destekler. Gruptaki her bir işçik, ayrık bir saklayıcı kümesine, komut tamponuna sahiptir. Fakat işçik grubu L1 cebini, komut ve veri TLB değerlerini, yürütme birimlerini ve diğer işhattı kaynaklarını paylaşır.

İşhattı, kısa tek-issue işhattıdır. İş hattı; kod alma (fetch), işçik seçme (thread select), kod çözme (decode), yürütme (execute), bellek (memory), ve geri yazım(write back) katlarından oluşur (Şekil 3.3). Yürütme birimi, tekil-çevrim gecikmeli ALU'ları, kaydırıcılar(shifters), ve çoklu-çevrim gecikmeli çarpma ve bölme birimlerinden oluşur. Her bir işçik, ayrık bir komut tamponuna sahiptir. Bu komut tamponları yürütme kaynakları etkin olmadığı zamanlarda da doldurulabilir. Sekiz çekirdek arasında paylaşılmış bir kayan noktalı birimi (FPU) vardır.

İşhattı, RAW bağımlılıklarını önlemek için komut atlamayı (bypassing) destekler. Çoğu süperskaler işlemcideki ortak olan sırasız komut çalıştırma veya tahmin gibi diğer özellikler bu mimaride bulunmamaktadır. Böylece daha basit ve minumum enerji gerektiren bir iş hattı mimarisi olmaktadır[4].



Şekil 3. 3 İş Hattı Yapısı



Şekil 3. 4 UltraSPARC T1'in iş hattı yapısı

3.3. İşçik Anahtarlama

İş hattının ikinci katı işçik seçim katıdır. Bu katta, işhattının sonraki katlarına hangi işçiğin geçirileceğine karar verilir.

İşçik seçim katı, işçik değişimi için birçok kendi kendine öğrenime dayanan methodlar kullanır. Varsayılan method, her çevrimde LRU (en yeni kullanılan) algoritmasına göre uygun işçiğe geçilmesidir. Uzun gecikme süreli bir komut, bir işçik değişiminin iş hattını dolu tutmasına sebep olmalıdır.

Temel İşçik Değişimi Kuralları :
Predecode Bilgisi : Uzun gecikme süreli komutlar, örneğin div, komut çözme ve almadan önce, predecoded bitleri sayesinde önceden tespit edilip işaretlenebilirler. Bitler eşlenir eşlenmez, bir işçik değişimi gerçekleşebilir. Örneğin; bir load komutu verinin tekrar kullanılabilmesi için üç çevrim gerektirir. Bu yüzden load komutu, iki komut için stalling (kritik) olarak ve işçik değişimi olarak işaretlenebilir.

Cep Iskaları : Cep ıskaları uzun gecikmelidir ve işçik değişimine neden olurlar. Sıralayıcı, load komutlarını cep vurusu olarak kabul eder, ve bu yüzden daha detaylı bağlı komutları riskli gibi işler. Buna rağmen, bir riskli işçik birlikte çalıştığı başka birinden daha düşük önceliğe sahiptir.

Traps : Trapler, işhattının sadece ilerleyen katlarında oluşabilir. Trapler, daha yeni komutların tampondan temizlenmesini gerektirirler.

Kaynak Çakışması : Kaynak çakışması bir işçiği durdurabilir. Her bir işçik, sınırlı sayıdaki komutu tutabilen ayrık bir komut tamponuna sahiptir.

4. SONUÇ

SPARC, açık kaynaklı bir ölçeklenebilir işlemci mimarisidir.

SPARC isimlendirmesindeki ölçeklenebilirlik, SPARC spesifikasyonunda belirtildiği üzere aynı çekirdek komut setinin gömülü sistemlerden büyük sunucu işlemcilerine kadar farklı büyüklüklerdeki gerçeklenmelerde kullanılabilir olmasından ileri gelmektedir. Ölçeklenebilirliğin kaynağı, genişletilebilir saklayıcı pencereleri yapısı, artırılabilir çekirdek sayısı ve her bir çekirdek için donanımsal çokluişçikli yapının kullanımıdır.

Altprogram sayısı çok fazla olan programlar için, SPARC mimarisi tercih edilmelidir. SPARC çok sayıda saklayıcıya sahiptir. SPARC'ın dairesel saklayıcı pencereleri yapısı, altprogram çağrılarının yoğun olduğu durumlarda diğer mimarilerine göre daha başarılı olmasını sağlamaktadır. Dairesel saklayıcı pencereleri yapısı ile, SPARC altprogramların parametrelerini ve dönüş değerlerini RAM'e kopyalamak yerine saklayıcılarda tutabilir. Ayrıca tüm programlara ortak olan global saklayıcılar kullanması performansı artırır.

SPARC'ın dallanmalar için koşul kodlarını kullanır. SPARC'ın dallanmaları kontrol etme methodu, MIPS'ten daha iyidir.

SPARC iş hattı yapısını destekler. SPARC mimarisi, bazı load/store koşullarında derleyicilerin dallanmaları izleyen gecikme slotlarını doldurmasını gerektirir.

SPARC mimarisinde, her bir işhattı katı kendi kontrolüne sahiptir. Bu sayede, SPARC, her bir işhattı katının performansını diğer katları etkilemeden maksimize etmeye çalışabilir.

5. SÖZLÜK

Thread : İşçik
Process : İşlem
Trap : Çökme
ILP(Instruction-Level Parallelism) : Komut düzeyinde paralellik

6. KAYNAKLAR

1. Weaver, David L., and Germond, Tom: **The SPARC Architecure Manual**, Version 9, Prentice Hall, 2000
2. "The scalable processor architecture (SPARC)", *Garner, R.B.; Agrawal, A.; Briggs, F.; Brown, E.W.; Hough, D.; Joy, B.; Kleiman, S.; Muchnick, S.; Namjoo, M.; Patterson, D.; Pendleton, J.; Tuck, R.;* Compcon Spring '88. Thirty-Third IEEE Computer Society International Conference, Digest of Papers, Pages: 278 – 283

3. The SPARC Architecture Manual Version 8, *Sun Microsystems*

4. Ana Sonia Leon, Jinuk Luke Shin, Kenway W. Tam, William Bryg, Francis Schumacher, Poonacha Kongetira, David Weisner, and Allan Strong. A power–efficient high–throughput 32–thread SPARC processor. In IEEE International Solid State Circuit Conference, 2006.

5. http://www.sparc.org/history.html

6. Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In ISCA, pages 392–403, 1995.