SPARC v8 İşlemci Simülasyonu Manueli

SPARC v8 işlemcisi; RISC (reduced instruction set computer) mimarisine bağlı kalınarak yaratılmış bir ISA(instruction set architecture) işlemcisidir. İşlemcinin; veri tipleri, durum saklayıcıları, pencere, iş hattı ve komut seti bilgileri detaylarıyla aşağıda verilmiştir.

1. Veri Tipleri

İşlemcide tüm saklayıcılar 32 bitlik olduğundan ve işlemci bellek arasında 32 bit paralel iletişim üzerinden haberleşme sağlandığından işlemcinin bir kelimesi 32 bit olarak kabul edilmiştir. Bu noktadan yola çıkılarak işlemcide kullanılan temel veri tipleri Tablo 1'deki gibi gösterilebilir.

Tablo 1 - İşlemcide Kullanılan Temel Veri Tipleri

İşaret	Veri Tipi	Bit Sayısı	Kapsadığı Aralık
İşaretli	Byte	8	-2^{7} / $2^{7}-1$
	Halfword	16	-2^{15} / $2^{15}-1$
	Word	32	-2^{31} / $2^{31}-1$
	Doubleword	64	-2^{63} / $2^{63}-1$
İşaretsiz	Byte	8	0 / $2^{8}-1$
	Halfword	16	0 / $2^{16}-1$
	Word	32	0 / $2^{32}-1$
	Doubleword	64	0 / $2^{64}-1$

Yukarıda verilen veri tipleri bellekten okunup, belleğe yazılırken dikkat edilmesi gereken bazı noktalar bulunmaktadır.

- Bellekten **byte** tipinde veri okunuyorsa; kaynak adres olarak herhangi bir bellek gözü seçilebilir. Alınan sayı saklayıcının düşük anlamlı 8 bitine yerleştirilir. Aynı şekilde belleğe byte tipinde veri yazılıyorsa; hedef adres herhangi bir bellek gözü seçilebilir. Bellek gözüne saklayıcının düşük anlamlı 8 bitindeki veri yazılır.

- Bellekten **halfword** tipinde veri okunuyorsa; kaynak adres belleğin ikiye bölünebilen gözlerinden biri (çift numaralı bellek adresleri) olmak zorundadır. Alınan sayı saklayıcının düşük anlamlı 16 bitine yazılır. Aynı şekilde belleğe halfword tipinde veri yazılıyorsa; hedef adres çift numaralı bellek adreslerinden biri olmalıdır. Bellek gözlerine saklayıcının düşük anlamlı 16 bitindeki veri yazılır.

- Bellekten **word** tipinde veri okunuyorsa; kaynak adres belleğin dörde bölünebilen adreslerinden biri olmak zorundadır. Alınan sayı saklayıcının tamamına yazılır. Aynı şekilde belleğe word tipinde veri yazılıyorsa; hedef adres belleğin dörde bölünebilen adreslerinden biri olmak zorundadır. Bellek gözlerine saklayıcının tamamındaki veri yazılır.

- Bellekten **doubleword** tipinde veri okunuyorsa; kaynak adres belleğin dörde bölünebilen adreslerinden biri olmak zorundadır. Alınan sayının yüksek anlamlı 32 biti verilen saklayıcının tamamına yazılır. Düşük anlamlı 32 biti ise bir sonraki saklayıcının tamamına yazılır. Aynı şekilde belleğe doubleword tipinde veri yazılıyorsa; hedef adres belleğin dörde bölünebilen adreslerinden biri olmak

zorundadır. Verilen bellek gözünden itibaren ilk 4 bellek gözüne (yüksek anlamlı 32 bit) verilen saklayıcının tamamındaki veri yazılır. Daha sonraki 4 bellek gözüne ise (düşük anlamlı 32 bit) bir sonraki saklayıcının tamamı yazılır.

2. Durum Saklayıcıları

İşlemci Simülasyonu'nda tanımlı olan 4 temel durum saklayıcısı bulunmaktadır. Bu saklayıcılar ve işlemci için önemleri aşağıda listelenmiştir.

- **PSR (Processor State Register – İşlemci Durum Saklayıcısı):** 32 bit büyüklüğündeki PSR bitlerine ayrılarak kullanılmıştır.
  - 31:24 numaralı bitler işlemcinin uygulama/versiyon bilgilerinin içerir. Kullanıcı tarafından görülemeyen ve değiştirilemeyen bu bitlerin değerleri işlemcinin üretimi aşamasında verilmiştir.
  - 23 numaralı bit n (negatif) bayrağıdır.
  - 22 numaralı bit z (sıfır) bayrağıdır.
  - 21 numaralı bit v (taşma) bayrağıdır.
  - 20 numaralı bit c (elde) bayrağıdır.
  - 19:14 numaralı bitler sonraki versiyonlarda kullanılmak üzere rezerve edilmiştir.
  - 13 numaralı bit ec (enable co-processor) bayrağıdır.
  - 12 numaralı bit ef (enable floating-point) bayrağıdır.
  - 11:5 numaralı bitler sonraki versiyonlarda kullanılmak üzere rezerve edilmiştir.
  - 4:0 numaralı bitler cwp (current window pointer) olarak kullanılmaktadır.

- **Y (Multiply/Divide Register – Çarpma/Bölme Saklayıcısı):** 32 bit büyüklüğündeki Y saklayıcısı işlemcide yapılan bölme ve çarpma işlemlerinde yardımcı saklayıcı olarak kullanılır.

  İki adet 32 bitlik veri saklayıcılarındaki değerler çarpıldığında sonuç 32 biti aşabilir. Böyle durumlarda veri kaybının önlenmesi amacıyla hedef saklayıcı olarak verilen saklayıcıya sonucun düşük anlamlı 32 biti yazılırken, Y saklayıcısına da yüksek anlamlı 32 bit yazılır.

  Benzer şekilde bölme işlemi yapılırken bölünen 32 bitten büyük bir sayı verilmek istenirse; kaynak saklayıcıya bölünenin düşük anlamlı 32 biti, Y saklayıcısına ise yüksek anlamlı 32 biti yazılır. İşlemci bölme işlemini yaparken bölüneni [Y:%RK1] çifti olarak alır.

- **PC (Program Counter – Program Sayacı):** 32 bit büyüklüğündeki PC durum saklayıcısı işlemci tarafında işlenmekte olan (iş hattına alınmakta olan) komutun bellek adresini barındırır.

- **nPC (Next Program Counter – Sonraki Program Sayacı):** 32 bit büyüklüğündeki nPC durum saklayıcısı işlemci tarafından bir sonraki adımda işlenecek olan (iş hattına alınacak olan) komutun bellek adresini barındırır. Böyle bir saklayıcıya ihtiyaç duyulmasının sebebi; dallanma, altprogram gidiş/dönüşleri gibi komutların sıralı işleyişinin değiştiği durumlarda iş hattının aksamadan çalışmasının sağlanmasıdır.

3. Pencere Yapısı

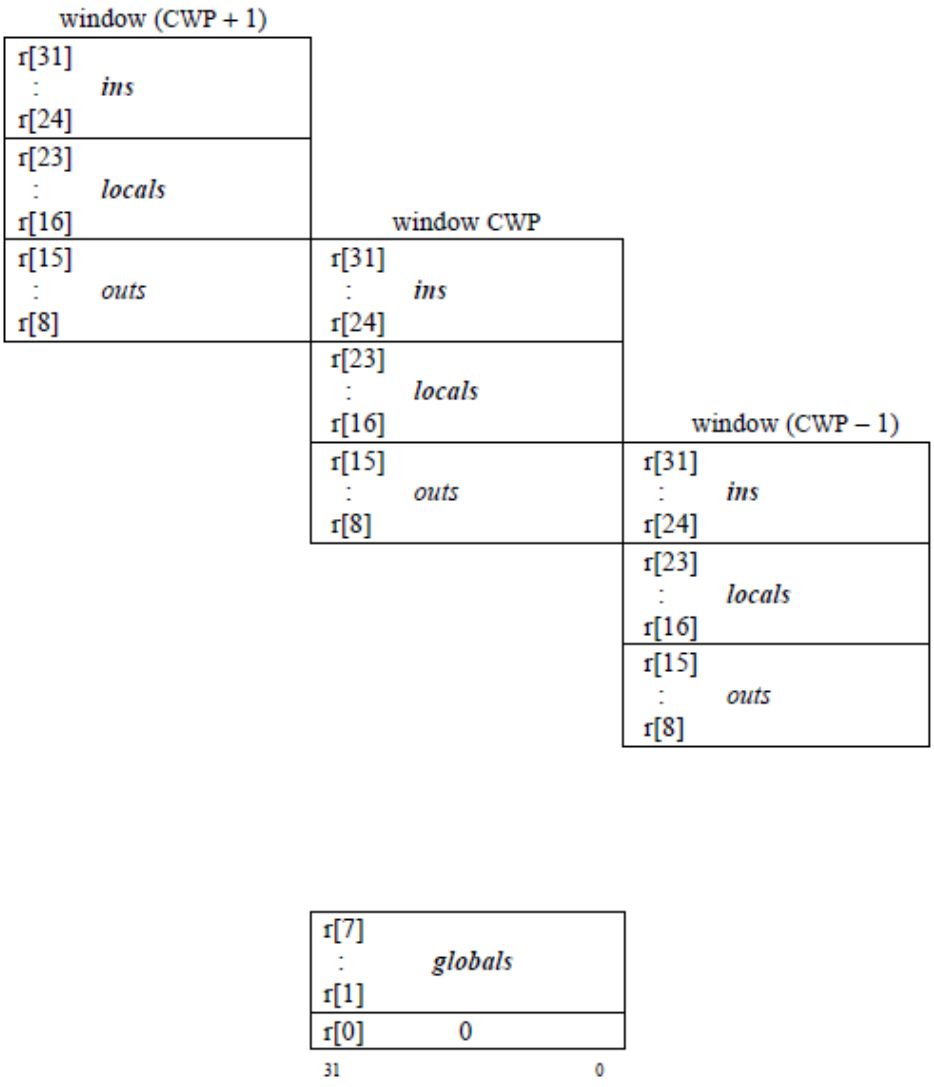
İşlemci Simülasyonu'nda tanımlı olan 520 adet genel amaçlı veri saklayıcı vardır. Ancak çalışma esnasında herhangi bir anda sadece 32 tanesi görülebilmektedir. Görülebilen bu 32 saklayıcıya pencere adı verilir (Şekil 1). Yani işlemcide 32 adet pencere tanımlıdır.

Bahsedilen 32 saklayıcıdan;

- 0:7 arasındakiler "**global saklayıcılar**" olarak adlandırılan ve tüm pencerelerde ortak olan saklayıcılardır.
- 8:15 arasındakiler "**dış saklayıcılar**" olarak adlandırılırlar ve bir önceki pencere ile ortak kullanılırlar.
- 16:23 arasındakiler "**lokal saklayıcılar**" olarak adlandırılırlar ve sadece mevcut pencere tarafından kullanılabilirler.
- 24:31 arasındakiler "**iç saklayıcılar**" olarak adlandırılırlar ve bir sonraki pencere ile paylaşılırlar.

Bu saklayıcılardan 2 tanesinin özel değerleri vardır.

- R0 saklayıcısı; her zaman 0 değerini gösterir.
- R15 saklayıcısı alt programdan dönüş adresinin değerini içerir.



Şekil 1 - Pencere Yapısı

4. İş Hattı

SPARC v8 İşlemci Simülasyonu'nda üç aşamalı iş hattı kullanılmaktadır. Kullanılan iş hattının aşamaları I(Instruction Fetch), A(ALU/Address Decode Operations) ve D(Data/Memory Access Operations) olarak belirlenmiştir.

- **I (Instruction Fetch - Komut Al):** Alma aşamasında PC de bellek adresi verilen komut alınarak iş hattının ilk adımına yerleştirilir. Komutun işlenmesiyle ilgili herhangi bir adımda bulunulmaz. Ancak iş hattının diğer birimleri arasında gerekli kaydırma işlemleri yapılır. I'daki komut A'ya; A'daki komut D'ye verilir.

- **A (ALU/Address Decode Operations – Alu /Adres Çöz İşlemleri):** Komut çözme aşamasında eğer komut bellek erişim komutu ise bellek hesaplamaları yapılır. ALU komutu ya da durum saklayıcısı komutu ise işlemin tamamı yapılır. Eğer kontrol transfer komutu ise gerekli PC ve nPC atamaları yapılır.

- **D (Data/Memory Access Operations – Veri/Bellek Erişimi İşlemleri):** Çalıştırma adımı sadece bellek erişim komutları için anlamlıdır. Bellek okuma işlemi yapılıyorsa ilgili adresten veri bu adımda getirilir; belleğe yazma işlemi yapılıyorsa ilgili adrese veri yine bu aşamada yazılır.

Komut yazılırken iş hattında oluşabilecek sorunlara dikkat edilmelidir. Bahsedilen sorunlar iki başlık altında toplanabilir:

- **Veri çakışması:** Bellekten veri okunurken, verinin hedef saklayıcıya iş hattının D aşamasında alındığından bahsedilmişti. Eğer bir veri okuma komutundan sonra, hedef saklayıcı kullanılarak bir ALU komutu çalıştırılıyorsa (ALU işlemleri A aşamasında yapılır.) saklayıcı değeri henüz kesinleşmediğinden işlem hatası oluşabilir. Simülasyon Derleme aşamasında kullanıcıyı bu hataya karşı uyarır. Bu iki komutun arasına "nop" komutu ekleyerek ya da başka bir komut taşıyarak bu sorun engellenebilir. Tablo 2'de veri çakışması ve çözümüne örnek verilmiştir.

Tablo 2 - Veri Çakışması Örneği

Adres	**Veri Çakışması**	**"nop" Ekleme**	**Komut Taşıma**
1000	ld [%r2] %r3	ld [%r2] %r3	ld [%r2] %r3
1004	add %r1 %r3 %r4	nop	add %r2 4 %r2
1008	add %r2 4 %r2	add %r1 %r3 %r4	add %r1 %r3 %r4

- **Dallanma Hatası:** Kontrol/Transfer komutlarında PC ve nPC düzenlemeleri A aşamasında yapıldığından, bir sonraki komut bu düzenlemelerden önce I aşamasında iş hattına alınmış olur. Simülasyon Derleme aşamasında kullanıcıyı bu hataya karşı uyarır. Kontrol/Transfer komutlarından sonra "nop" komutu kullanarak ya da daha önceki veri bağımlılığı bulunmayan bir komutu sonrasına taşıyarak sorun oluşması engellenebilir. Tablo 3'te dallanma hatası ve çözümüne örnek verilmiştir.

Tablo 3 - Dallanma Hatası Örneği

Adres	**Dallanma Hatası**	**"nop" Ekleme**	**Komut Taşıma**
1000	sethi a %r1	sethi a %r1	sethi a %r1
1004	add %r1 2 %r3	add %r1 2 %r3	call altP
1008	call altP	call altP	add %r1 2 %r3
100C	st %r2 [%r3]	nop	st %r2 [%r3]

5. Komut Seti

İşlemci Simülasyonu'nda tanımlı olan komutlar ve detayları anlatılmadan önce komut setinin daha iyi anlaşılabilmesi için bazı açıklamalar yapılacaktır.

Komut setinde; hiç argüman almayan, 1 argüman alan, 2 argüman alan ve 3 argüman alan komutlar mevcuttur. Hangi komutun kaç argüman aldığının detayı Tablo 4'te verilmiştir. Komut sentaksında; birimler TAB ile ayrılırlar. Her komutta;

- İlk birim etiket için ayrılmıştır ve etiketler ':' (iki nokta üst üste) karakteri ile bitmek zorundadırlar.
- İkinci birim komut adını içerir. Etiket yok ise dahi satır başından TAB bırakılarak ikinci birime komut adı yazılmalıdır.
- Argümanlarda birbirinden TAB ile ayrılırlar; üçüncü, dördüncü ve beşinci birimler argümanlara ayrılmıştır.

Yorum yazılmadan önce; ister tüm satır yorum olsun, ister komutun ardına yorum eklensin; '!' (ünlem işareti) karakteri kullanılmalıdır.

SPARC v8 İşlemci Simülasyonu komut setinde atamalar soldan sağa gerçekleştirilir. Yani kaynak birimler solda; hedef birimler sağda verilmiştir. Tablo 6'da kaynak saklayıcılar **%rk1** ve **%rk2** şeklinde gösterilirken; hedef saklayıcılar **%rh** şeklinde gösterilmiştir. Sayı değerleri ve data etiketleri ise **sd** şeklinde gösterilmiştir.

Komut yazılırken verilen sayılar başlarında '$' karakteri yoksa hexadecimal (16'lık) olarak kabul edilirler. Başına '$' karakteri eklenen sayılar decimal (10'luk) olarak alınırlar.

Komut yazılırken kullanılabilecek 4 tane segman etiketi bulunmaktadır. Bu etiketler .data, .bss, .equ ve .text şeklinde isimlendirilirler ve satırın ikinci biriminde verilmeleri gerekir. Kendilerinden sonra gelen tüm satırlar başka bir segman etiketi görülene kadar o segmana ait sayılır.

- **.data :** Program başlangıcında yapılan belli değer atamalarının yapıldığı segman data segmanıdır. Bu segmanda satırın ilk biriminde verilen etikete, ikinci biriminde verilen veri tipinde, üçüncü birimde verilen değer atanır. Veri tipleri byte, halfword ve word olabilir. Bu segmanda yer alabilecek bazı atamalar aşağıdaki gibidir.

  ```
  diziA:  .word   0700
  n:      .byte   02
  ```

- **.bss :** Belli bir bellek adresinin isimlendirilerek program boyunca bahsi geçen bellek adresine ulaşılmak istendiğinde bu isimin kullanılmasını sağlayan segmandır. Belleğe verilen isim bellek erişimi esnasında [] içerisinde kullanılabilir. Köseli parantez içine saklayıcı adı ve bss etiketi dışında değer yazılamaz. Bu segmanda yer alabilecek bazı atamalar aşağıdaki gibidir.

  ```
  fibBas: 1000
  diziA:  10c0
  ```

- **.equ :** İlk birimde verilen bellek adresinden itibaren üçüncü bürümde verilen elemanlarının ikinci birimdeki veri tipinden değerler halinde belleğe yazılmasını sağlayan segmandır. Byte, halfword ve word tipinden değerler belleğe yazılabilir.

Yazılacak değerler ‘,’ (virgül) karakteriyle ayrılmalı ancak aralarında boşluk olmamalıdır. Bu segmanda yer alabilecek bazı atamalar aşağıdaki gibidir.

```
700:    .word       ff,ee,dd,cc,bb,aa,99,88,77,66,55,44,33,22,11,00
800:    .halfword   12a0
```

- **.text :** Komutların sıralandığı segmandır. Yanına (satırdaki üçüncü birim) bir bellek adresi alabilir. Bu durumda komutlar belleğe verilen bellek adresinden itibaren yazılacaktır. Yanına adres verilmemesi halinde komutlar 0 numaralı bellek gözünden itibaren yazılır. Altprogramlardan önce tekrar kullanılarak; altprogramın farklı bir bellek bölgesine yazılması sağlanabilir.

Tablo 4 - Komut Seti

KOMUT TANIMI				AÇIKLAMA	TÜR
sethi	sd	%rh		Saklayıcıya değer atama	BELLEK ERİŞİM KOMUTLARI
ldd	[%rk1]	%rh		Saklayıcıya doubleword yükle	
ld	[%rk1]	%rh		Saklayıcıya word yükle	
ldsh	[%rk1]	%rh		İşaretli halfword yükle	
lduh	[%rk1]	%rh		İşaretsiz halfword yükle	
ldsb	[%rk1]	%rh		İşaretli byte yükle	
ldub	[%rk1]	%rh		İşaretsiz byte yükle	
std	%rk1	[%rh]		Belleğe doubleword kaydet	
st	%rk1	[%rh]		Belleğe word kaydet	
sth	%rk1	[%rh]		Belleğe halfword kaydet	
stb	%rk1	[%rh]		Belleğe byte kaydet	
swap	[%rk1]	%rh		Saklayıcı/Bellek değeri değiştir	
and	%rk1	%rk2 / sd	%rh	Ve	ARİTMETİK LOJİK BİRİM KOMUTLARI
andn	%rk1	%rk2 / sd	%rh	Ve değil	
or	%rk1	%rk2 / sd	%rh	Veya	
orn	%rk1	%rk2 / sd	%rh	Veya değil	
xor	%rk1	%rk2 / sd	%rh	Özel veya (ya da)	
xnor	%rk1	%rk2 / sd	%rh	Özel veya (ya da) değil	
sll	%rk1	%rk2 / sd	%rh	Mantıksal sola kaydırma	
srl	%rk1	%rk2 / sd	%rh	Mantıksal sağa kaydırma	
sra	%rk1	%rk2 / sd	%rh	Aritmetik sağa kaydırma	
add	%rk1	%rk2 / sd	%rh	Toplama	
addx	%rk1	%rk2 / sd	%rh	Eldeli toplama	
sub	%rk1	%rk2 / sd	%rh	Çıkarma	
subx	%rk1	%rk2 / sd	%rh	Eldeli çıkarma	
umul	%rk1	%rk2 / sd	%rh	İşaretsiz çarpma	
smul	%rk1	%rk2 / sd	%rh	İşaretli çarpma	
udiv	%rk1	%rk2 / sd	%rh	İşaretsiz bölme	
sdiv	%rk1	%rk2 / sd	%rh	İşaretli bölme	
rdy	%rh			Y'den veri oku	DURUM SAKLAYICISI KOMUTLARI
wry	%rk1			Y'ye veri yaz	
nop				Hiç bir şey yapma	
ba	sd			Her zaman dallan	KONTROL TRANSFER KOMUTLARI
bn	sd			Asla dallanma	
bne	sd			Eşit değilse dallan	
be	sd			Eşitse dallan	
bg	sd			Büyükse dallan	
ble	sd			Eşit veya küçükse dallan	
bge	sd			Eşit veya büyükse dallan	
bl	sd			Küçükse dallan	
bcc	sd			Elde yoksa dallan	
bcs	sd			Elde varsa dallan	
bpos	sd			Pozitifse dallan	
bneg	sd			Negatifse dallan	
call	sd			Altprogram çağır	
jmpl	sd			Altprogramdan dön	
ta	sd			Yazılım kesmesi	