RISC (*Reduced Instruction Set Computer*) İşlemciler:

CISC – RISC Örnekleri:

RISC: MIPS, SPARC, Alpha, HP-PA, PowerPC, i860, i960, ARM, Atmel AVR

CISC: VAX, PDP-11, Intel i86, Motorola 68K.

CISC (RISC benzeri iç yapıya sahip): Pentium, AMD Athlon.

CISC (*Complex Instruction Set Computer*) Özellikleri:

Yüksek düzeyli programlama dillerine yakın yetenekte komutlar ve adresleme kipleri sağlamaya çalışılmıştır.

- Çok sayıda komut (100 – 250)
- Bazı komutlar özel işlemler içindir, çok sık kullanılmazlar
- Çok sayıda ve karmaşık yapıda adresleme kipi (dolaylı adreslemeler)
- Değişken uzunlukta komut yapısı (komut çözme işi karmaşıktır)
- Doğrudan bellek üzerinde işlem yapan komutlar
- Mikropogramlı denetim birimi

Yüksek düzeyli bir dille ile yazılmış olan bir program az sayıda makine dili komut kullanılarak derlenebilir.

© 2005-2011 Dr. Feza BUZLUCA

RISC (*Reduced Instruction Set Computer*) İşlemcilerin Ortaya Çıkışı:

Yüksek düzeyli programlama dilleri ile yazılmış olan programların CISC makinelerde derlenmesi ile elde edilen kodlar incelendiğinde aşağıdaki noktalar belirlenmiştir:

- Çok sayıda atama (A = B) yapılmaktadır.
- Erişilen veri tipleri çoğunlukla yerel ve skaler (dizi, matris olmayan) verilerdir.
- Makine dili programlarda en büyük yükü alt program çağrıları oluşturmaktadır. Geri dönüş adresi, parametre aktarımı, yerel değişkenler, yığın (bellek) kullanımı
- Alt programların büyük çoğunluğu (%98) 6 ya da daha az parametre aktarmaktadır. [1]
- Alt programların büyük çoğunluğu (%92) 6 ya da daha az yerel değişken kullanmaktadır.[1]
- Alt program çağırma derinliği büyük çoğunlukla (%99) 8'den daha azdır. [2]

İncelenen programlardan elde edilen bu veriler dikkate alınarak merkezi işlem birimlerinin performanslarını arttırmak amacıyla daha az bellek erişimi yapan ve birazdan açıklanacak olan özelliklere sahip olan RISC işlemciler tasarlanmıştır.

1. Andrew S. Tanenbaum, Implications of structured programming for machine architecture, Communications of the ACM, Vol.21 , No.3 (1978),pp. 237 - 246
2. Yuval Tamir and Carlo H. Sequin, "Strategies for Managing the Register File in RISC," IEEE Transactions on Computers Vol. C-32(11) pp. 977-989, 1983.

© 2005-2011 Dr. Feza BUZLUCA

RISC Özellikleri:

- Daha az sayıda komut vardır, komutların işlevleri basittir.
- Daha az sayıda , basit adresleme kipi
- Sabit uzunlukta komut yapısı (komut çözme işi kolaydır)
- Doğrudan bellek üzerinde işlem yapan komutlar yoktur, işlemler iç saklayıcılarda yapılır.
- Belleğe sadece yazma/okuma işlemleri için erişilir (*load-store architecture*).
- Tek çevrimde alınıp yürütülebilen komutlar (komut işhattı (*pipeline*) sayesinde)
- Devrelendirilmiş (*hardwired*) denetim birimi.

Diğer Özellikler:

Aşağıdaki özelliklerin bazıları tüm RISC'lerde bulunmayabilir, bazısı ise CISC MİB'lerde de bulunabilir. Ancak bunlar RISC'ler için özellikle önemlidir.

- Çok sayıda saklayıcı (*register File*)
- Kesişimli (*overlapped register window*) saklayıcı penceresi
- Komutlar için optimize edilebilen iş hattı
- Harvard mimarisi
- Derleyici desteği

© 2005-2011 Dr. Feza BUZLUCA

RISC işlemcilerin kullanıldığı ürünlere ilişkin örnekler:

- ARM:
  - Apple iPod , Apple iPhone, iPod Touch, Apple iPad.
  - Palm and PocketPC PDA, smartphone
  - RIM BlackBerry smartphone/email device.
  - Microsoft Windows Mobile
  - Nintendo Game Boy Advance
- MIPS:
  - SGI computers, PlayStation, PlayStation 2
- Power Architecture (IBM, Freescale (eski Motorola SPS)):
  - IBM supercomputers, midrange servers and workstations,
  - Apple PowerPC-tabanlı Macintosh
  - Nintendo Gamecube, Wii
  - Microsoft Xbox 360
  - Sony PlayStation 3
- Atmel AVR:
  - BMW otomobillerde denetçi olarak kullanılıyor.

© 2005-2011 Dr. Feza BUZLUCA

Binişimli (Kesişimli) Saklayıcı Pencereleri (*Ovelapped Register Windows*):

Bu yapı, alt program çağrılarında yığına (bellek erişimine) gerek duymadan
- parametre aktarımını sağlamak ve
- yerel değişkenleri tutmak için kullanılır.

İşlemcinin çok sayıda saklayıcısı olmasına rağmen programcı belli bir anda bunlardan sadece belli bir adetini kullanabilir.

Bir anda kullanılabilen saklayıcıların oluşturduğu gruba **pencere** (*window*) denir.

Alt programa gidildikçe (ve geri dönüldükçe) pencere değişir.

Böylece programcı farklı saklayıcılara erişir.

İki pencere arasındaki ortak saklayıcılar parametre aktarımı için, ortak olmayanlar ise alt programların yerel değişkenleri için kullanılırlar.

Bir pencerede n saklayıcı varsa programlar yazılırken sadece R0 ve Rn-1 numaraları kullanılır.

Ancak pencere değiştikçe bu numaralar farklı fiziksel saklayıcılara denk düşerler.

Tüm RISC işlemciler bu yapıyı kullanmaz. Örneğin MIPS işlemcisinde yoktur.

© 2005-2011 Dr. Feza BUZLUCA

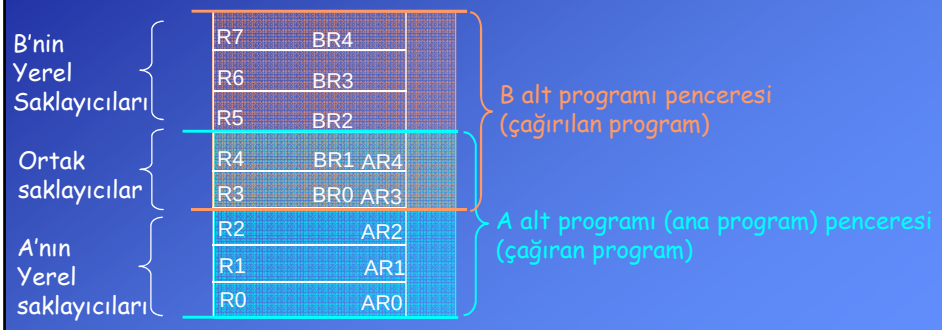
Örnek:

Aşağıdaki örnekte işlemcinin 8 adet saklayıcısı vardır. Ancak bir pencerede 5 saklayıcı olduğundan belli bir anda bunlardan sadece 5 tanesi kullanılabilmektedir.

Programlarda sadece R0-R4 kullanılır ancak pencere değiştikçe bunlar farklı saklayıcılara denk düşerler.

A'da programcı R0'a eriştiğinde işlemcinin R0'ına erişmiş olur.

B'de programcı R0'a eriştiğinde işlemcinin R3'üne erişmiş olur.



Tüm alt programların eriştiği ve numaraları değişmeyen global saklayıcılar da bulunur.

© 2005-2011 Dr. Feza BUZLUCA

Saklayıcı sayılarının belirlenmesi:

G: Global saklayıcı sayısı
L: Yerel saklayıcı sayısı
C: İki pencere arasındaki ortak saklayıcı sayısı
W: Pencere Sayısı
Pencere boyu = L+2C+G (2*C çünkü hem alttaki hem de üstteki pencere ile ortak saklayıcılar vardır.)
Saklayıcı sayısı = (L+C)W + G

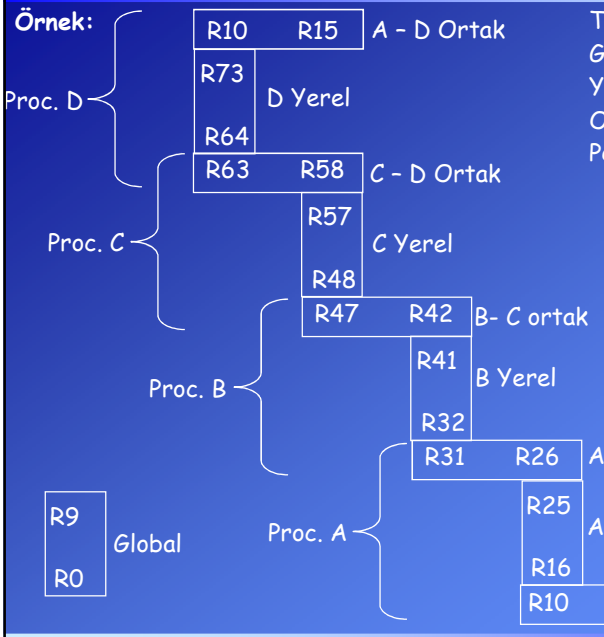
Pencere sistemi çevrel (*circular*) olarak tasarlanır.
Eğer işlemcinin 4 adet penceresi varsa, iç içe 5nci alt program çağırıldığında en eski programın 1nci pencerede yer alan bilgileri belleğe yazılır.
Bundan sonra 1nci pencere 5nci alt program tarafından kullanılır.
Geri dönüşte bellekteki bilgiler tekrar ilgili pencereye taşınır.

Örnek:
Sonraki örnekte, toplam 74 adet saklayıcısı bulunan, 32 saklayıcılı pencerelere sahip ve 4 derinliğinde alt program çağrılarına destek veren bir işlemcinin saklayıcı yapısı gösterilmiştir.

Bu örnekte alt programa gidildikçe pencerelerin artan numaralı saklayıcılara doğru ilerlediği var sayılmıştır. Gerçek işlemcilerde (RISC 1, SPARC) pencereler azalan adreslere doğru ilerlemektedir.

© 2005-2011 Dr. Feza BUZLUCA

Örnek:



Toplam 74 saklayıcı
Global saklayıcılar G=10 R0-R9
Yerel saklayıcı: L=10 adet
Ortak saklayıcı 2*C= 6+6=12 adet
Pencere boyu 32 saklayıcı

Bu sistemde, bir pencerede 32 adet saklayıcı olduğundan, simgesel dilde program yazılırken her alt programda R0 – R31 numaraları kullanılır. Pencerenin konumuna göre bu numaralar farklı fiziksel saklayıcılara karşı gelirler. Bkz. 3.14'teki örnek program

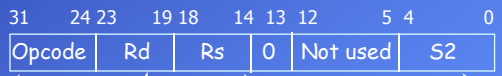
© 2005-2011 Dr. Feza BUZLUCA

RISC İşlemci Örneği: Berkeley RISC I

- 32 bit adres yolu
- von Neumann mimarisi: Komut ve veri belleği ortaktır.
- 8, 16, 32 bitlik veriler
- Komutlar sabit uzunlukta ve 32 bit
- Toplam 31 komutu var
- Toplam 138 saklayıcı (R0-R137),
  8 pencere her pencerede 32 saklayıcı, 10 adet global saklayıcı (R0-R9)
  Yerel saklayıcı: 10 adet, Ortak saklayıcı 6+6=12 adet
- 3 adet adresleme kipi: saklayıcı adresleme, ivedi adresleme, bağıl adresleme

Komut Yapısı:

Saklayıcı kipi:
(*Register mode*)



Kolay çözülür — Destination — Source — İkinci kaynak saklayıcı

Örnek: ADD R22, R21, R23 R23←R22+R21

Saklayıcı – ivedi (*Register-immediate mode*):

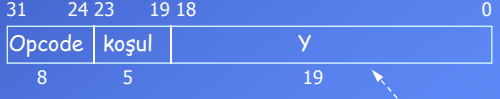
İvedi veri

Örnek: ADD R22, #150, R23 R23←R22+150

Bellek erişimi komutlarında, Rs adres saklayıcısı (işaretçi), S2 ise öteleme miktarı olarak kullanılır. Rs'nin içindeki 32 bitlik adres + S2'nin gösterdiği yere erişilir.

Örnek: LDL (R10)#5,R5 R5←M[R10 + 5] Load long: 32 bitlik veri aktarımı

Bağıl (*PC Relative mode*):



Örnek: JMPR EQ,Y Öteleme (*offset*)

Berkeley RISC I komutlarının kullanılması

R0 her zaman sabit 0 (sıfır) değerini taşır.

ADD	R0, R21, R22	R22←R21 (Move)
ADD	R0, #150, R22	R22←150 (İvedi yükleme)
ADD	R22, #1, R22	R22←R22 +1 (Increment)

Bellek erişimi için Load/Store komutları kullanılır.

LDL	(R22)#150,R5	R5←M[R22 +150]	Load long: 32 bitlik veri aktarımı
LDL	(R22)#0,R5	R5←M[R22]	
LDL	(R0)#500,R5	R5←M[500]	

Berkeley RISC I Komut Tablosu

Veri işleme komutları:

Opcode	Operands	Register Transfer
ADD	Rs,S2,Rd	*Rd* ← *Rs* + S2
ADDC	Rs,S2,Rd	*Rd* ← *Rs* + S2 + carry
SUB	Rs,S2,Rd	*Rd* ← *Rs* - S2
SUBC	Rs,S2,Rd	*Rd* ← *Rs* - S2 - carry
SUBR	Rs,S2,Rd	*Rd* ← *S2* - *Rs*
SUBCR	Rs,S2,Rd	*Rd* ← *S2* - *Rs* - carry
AND	Rs,S2,Rd	*Rd* ← *Rs* ∧ S2
OR	Rs,S2,Rd	*Rd* ← *Rs* ∨ S2
XOR	Rs,S2,Rd	*Rd* ← *Rs* ⊕S2
SLL	Rs,S2,Rd	*Rd* ← *Rs* shifted by S2
SRL	Rs,S2,Rd	*Rd* ← *Rs* shifted by S2
SRA	Rs,S2,Rd	*Rd* ← *Rs* shifted by S2

Veri aktarım komutları:

Opcode	Operands	Register Transfer	
LDL	(Rs)S2,Rd	*Rd←M[Rs + S2]*	Long load
LDSU	(Rs)S2,Rd	*Rd←M[Rs + S2]*	Short unsigned
LDSS	(Rs)S2,Rd	*Rd←M[Rs + S2]*	Short signed
LDBU	(Rs)S2,Rd	*Rd←M[Rs + S2]*	Byte unsigned
LDBS	(Rs)S2,Rd	*Rd←M[Rs + S2]*	Byte signed
LDHI	Y,Rd	*Rd←Y*	Immediate high
STL	(Rs)S2, Rm	*M[Rs + S2] ← Rm*	Store load
STS	(Rs)S2, Rm		
STB	(Rs)S2, Rm		
GETPSW	Rd	*Rd←PSW*	Load status word
PUTPSW	Rd	*PSW←Rd*	Set status word

© 2005-2011 Dr. Feza BUZLUCA

Program denetim komutları:

Opcode	Operands	Register Transfer	
JMP	COND,S2(Rs)	*PC←Rs + S2*	Mutlak (doğrudan) adresleme
JMPR	COND,Y	*PC←PC + Y*	Bağıl
CALL	S2(Rs),Rd	*Rd←PC* *PC←Rs + S2* *CWP ← CWP -1*	*Current window pointer*
CALLR	Y,Rd	*Rd←PC* *PC←PC + Y* *CWP ← CWP -1*	Bağıl
RET	(Rd)S2	*PC←Rd + S2* *CWP ← CWP +1*	

Berkeley RISC I işlemcisinde altprograma gidildiğinde pencere işaretçisi (CWP) azaltıldığından daha küçük numaralı saklayıcılara doğru gidilir.

Buna göre ana program (A prosesi) en yüksek numaralı saklayıcıları (R116-R137) ve global saklayıcıları (R0-R9) kullanır.

© 2005-2011 Dr. Feza BUZLUCA

Örnek Program:

500 ve 504 numaralı bellek gözlerinde bulunan 32 bitlik iki işaretli sayının toplamını gerçekleştiren ve sonucu 508 numaralı bellek gözüne yazan programı Berkeley RISC-1 simgesel dili ile kesişimli saklayıcı pencereler üzerinde parametre aktarımı gerçekleştirerek yazınız.

Toplama alt programı, R1 numaralı saklayıcıdaki adresin 20 ilerisinden başlamaktadır.

Çözüm:

	Program	Açıklama
	LDL (R0) #500, R10	R10 ← M[500] (1. parametre)
	LDL (R0) #504, R11	R11 ← M[504] (2. parametre)
	CALL (R1)#20, R15	R15 ← PC
		PC ← (R1)+20
		CWP ← CWP-1
	STL (R0) #508, R12	M[508] ← R12 (geri dönen değer)
	…	
	…	
[(R1)+20]	ADD R26, R27, R28	R28 ← R27+R26
	RET (R31)#0	PC ← (R31)+0
		CWP ← CWP+1

Not: Bu program yazılırken 4. bölümde anlatılan iş hattında (*pipeline*) çıkan sorunlar dikkate alınmamıştır.

© 2005-2011 Dr. Feza BUZLUCA