

BİLGİSAYAR ORGANİZASYONU ve TASARIMI

DR. FATİH KELEŞ

Merkezi İşlemci Birimi - MİB (CPU)

- ▶ Giriş
- ▶ Genel Saklayıcı Organizasyonu
- ▶ Yığın Organizasyonu
- ▶ Komut Formatları
- ▶ Adresleme Modları
- ▶ Data Transferi ve İşlemleri
- ▶ Program Kontrolü
- ▶ Azaltılmış Komut Kümeli Bilgisayar – RISC

Komut Kümesi Ne Sağlar?

- ▶ Zengin komut kümeleri derleyicilerin işini kolaylaştırır.
- ▶ Zengin komut kümeleri yazılım krizini önler.
Ya da mümkün mertebe daha fazla fonksiyonu donanımsal olarak gerçekleme yoluna gidilmelidir.
- ▶ Zengin komut kümeleri mimarının kalitesini artırır.

Azaltılmış Komut Kümeli Bilgisayarlar

RISCs

- ▶ 1970'li yılların sonları-1980'li yılların başlarında, CISC türü işlemcilerin problemlerine bir reaksiyon olarak RISC işlemciler ortaya çıkmıştır.
- ▶ RISC işlemcilerin ardında yatan temel düşünce, komut setini basitleştirmek ve komut icra süresini azaltmaktadır.

RISC

- ▶ Yükleme komutları, operandlar için sadece saklayıcı kullanırlar. Sadece birkaç adresleme moduna ihtiyaç gösterirler.
- ▶ Komutlar aynı bit uzunluğu ile kodlandıklarından okunmaları basit ve hızlıdır.
- ▶ Fetch ve decode aşamaları basittir. (TBO gibi)
- ▶ Komut ve adres formatları decoding için kolayca tasarlanabilir.
- ▶ RISC komutlarının opcode ve saklayıcı alanları aynı anda çözülebilir.
- ▶ Az sayıda komut ve adresleme moduna sahiptir.
- ▶ RISC işlemcinin kontrol lojiği basit ve hızlı olarak tasarlanabilir.
- ▶ Kontrol lojiği donanımsaldır (yazılımsal kontrole göre daha hızlıdır).

RISC İşlemcilerinin Özellikleri

- Az sayıda komut
- Az sayıda adresleme modu
- Belleğe erişimde sadece load ve store komutları
- Diğer bütün işlemler, işlemci saklayıcılarını kullanarak gerçekleşir
- Sabit uzunluklu komutlar
- Tek saat çevrim icra (execution) süreli komutlar
- Kontrol ünitesi yazılımsal yerine donanımsal

RISC Avantajları

- VLSI Gerçeklemeye uygunluk
- Hızlı Hesaplama
- Güvenilirlik ve ucuz maliyetli tasarım

Mimari Tasarım Prensipleri

- Bellek Teknolojisinin hızlı gelişimi, büyük mikroprogramların az bir donanımsal maliyetle gerçekleştirilebilmelerine neden olduğundan bu da Geniş Genel Amaçlı Komut Kümesi eldesini sağlamıştır.
- Mikroprogram belleği ana bellekten daha hızlı olduğundan, Mikroprogram, makine komutlarından daha hızlı icra edilir.
- Yüksek performanslı makineler için, mikroprogram içine çeşitli yazılım fonksiyonlarını gerçekleme imkanı vermiştir.
- Pahalı bir tasarım olduğundan CPU'daki saklayıcı sayısının sınırlı olması etkin olarak kullanımlarını zorlaştırmaktadır.

Kompleks Komut Seti Mimarisi - CISC

- ▶ Çok komutlu ve adresleme modlarına sahip bilgisayarlar olarak bilinirler (CISC).
- ▶ CISC bilgisayar için temel amaç, yüksek seviyeli dil ifadesine karşı gelen bir makine dili komutuna sahip olmaktadır.

Değişken Uzunluklu Komutlar

- ▶ Çok sayıda komut ve adresleme modları CISC makinelerinin **değişken uzunluklu komut formatlarına** sahip olmasına izin verir.
- ▶ Çok sayıda komut kodlamak için büyük uzunluklu bit kullanımı gerektirir.
- ▶ Etkin olarak çok sayıda opkodu yönetmek için, farklı uzunluklu kodlama kullanılmalıdır:
 - Sık kullanılan komutlar kısa opkodlar ile
 - Ender kullanılan komutlar daha uzun opcodlar ile kodlanırlar.
- ▶ Ayrıca, çoklu operandlı komutlar her operand için farklı adresleme modlarını belirler.

Örneğin,

- Operand 1 doğrudan adreslenebilir bir saklayıcı,
 - Operand 2 dolaylı adreslenebilir bir bellek alanı,
 - Operand 3 (hedef) dolaylı adreslemeli bir saklayıcı.
- ▶ Farklı durumlarda, kullanılan opkod ve operandlara bağlı olarak farklı uzunluklu komutlar gereklidir.

Değişken Uzunluklu Komutlar

- ▶ Örneğin, sadece register operandlarını belirleyen bir komut 2 byte uzunluklu olabilir
 - 1 byte komut ve adresleme modunu belirler.
 - 1 byte kaynak ve hedef saklayıcıları belirler.

Operandlar için bellek adreslerini belirleyen bir komutun 5 byte ihtiyacı bulunur:

 - 1 byte komut ve adresleme modunu belirler.
 - 2 byte her bir bellek adresini belirler.
- ▶ Değişken uzunluklu komutlar büyük oranda fetch ve decoding problemlerini karmaşıklığıdır.
- ▶ Çeşitli komutları tanıtmak ve operandlar için gerekli byte sayısını uygun bir şekilde fetch etmek çok kompleks bir iştir.

Kompleks Komut Kümeli Bilgisayar

- ▶ CISC bilgisayarlarının diğer bir karakteristiği, bellek adreslerine doğrudan erişen komutlara sahip olmaktadır.

Örneğin,

ADD L1, L2, L3

$M[L1]$ içeriği, $M[L2]$ nin içeriği ile toplanır ve sonuç $M[L3]$ bellek alanına depolanır.

- ▶ Bu komutun icrası 3 bellek varış çevrimi sürer.
- ▶ Bu potansiyel olarak çok uzun bir çevrim süresidir.
- ▶ CISC mimarilerindeki problemler:
 - Tasarım karmaşıklığının işlemciyi yavaşlatması
 - Tasarım karmaşıklığı işlemcinin tasarım ve gerçeklenmesinde maliyetsel hatalara neden olması
 - Komut ve adresleme modlarının pek çögünün nadiren kullanılması

CISC Önemli Karakteristikleri

- Çok sayıda komut içermesi (100-250)
- Nadiren kullanılan bazı özel komutlar içermesi
- Çok çeşitli adresleme modları (5-20)
- Değişken uzunluklu komut formatları
- Bellekteki operandlara erişmek için kullanılan komutlar

CISC Komut Kümesi Mimarisi

Kompleks komut kümesi

- Daha karmaşık opkodlar
- Daha fazla adresleme modu

Saklayıcı-Bellek Mimarisi

- Hesaplama komutlarındaki operandlar bellekte saklanır.

Kompleks komut kodlama

- Değişken uzunluklu komutlar
(farklı operand sayıları, farklı operand uzunlukları, değişik formatlar)

CISC Komut Kümesi Mimarisi

Daha karmaşık saklayıcı tasarımlı

- Özel amaçlı saklayıcı kullanımı
- Karma yığın mimarisi
(Kayan noktalı sayılar için)

Daha karmaşık bellek yönetimi

- Sayfa bölütleme (segmentation paging)

RISC ve CISC

RISC işlemcinin avantajı 2 önemli faktöre dayanır:

- Çip teknolojisine
- İşlemci karmaşıklığına

1990 öncesi: Çip yoğunluğu düşük & işlemci gerçeklemeleri basitti.

- Tek-çip RISC CPUs (1986) & çip üzerinde cep bellek
- Komut çözme CISC icra süresinde önemli bir paya sahipti

1990 sonrası: Çip yoğunluğu yüksek & işlemci gerçeklemesi karmaşıktır.

RISC & CISC gerçeklemeleri büyük bir L1 cep bellek içeren çip üzerine gerçekleşmiştir.

Saklayıcı

- ▶ Komutların ve adresleme modlarının basitleştirilmesi, RISC CPU içinde daha fazla devre tasarıımı için yer sağlar.
- ▶ Bu ekstra kapasite 2 önemli işlem için kullanılır:
 - Pipeline komut icrası (komut icrasını hızlandırır)
 - CPU'ya çok sayıda saklayıcı eklemek

Pipelining

- ▶ Çoğu RISC işlemcilerinin en önemli özelliği her saat çevriminde bir komut icra etmektir.
- ▶ Bir komutun icra aşamaları düşünülürse, (fetch, decode ve execute) bu işlem pek mümkün görünmez
- ▶ Pipelining denilen teknik bunu sağlar.
- ▶ Pipelining çoklu komutların farklı aşamalarını paralel olarak icra etmek için işlemcinin kullanılmasıdır.

Pipelining

- ▶ Örneğin, belirli bir anda, bir pipeline özelliği olan işlemci
 - *Komut icrası* i_t
 - Komut kodçözme i_{t+1}
 - Komut Fetching i_{t+2}
 - 3 komut aynı anda farklı aşamaları icra edilirse, saat çevrim başına 1 komut icra edilmiş olacaktır.
- ▶ Pipeline gerçeklemede bazı sorunlarla karşılaşılır:
 - Örneğin, işlemci dallanma yaparsa pipeline yapısına ne olur ?
 - Bununla birlikte pipeline icra özelliği modern işlemcilerde bulunan birzelliktir ve önemli rol oynar.

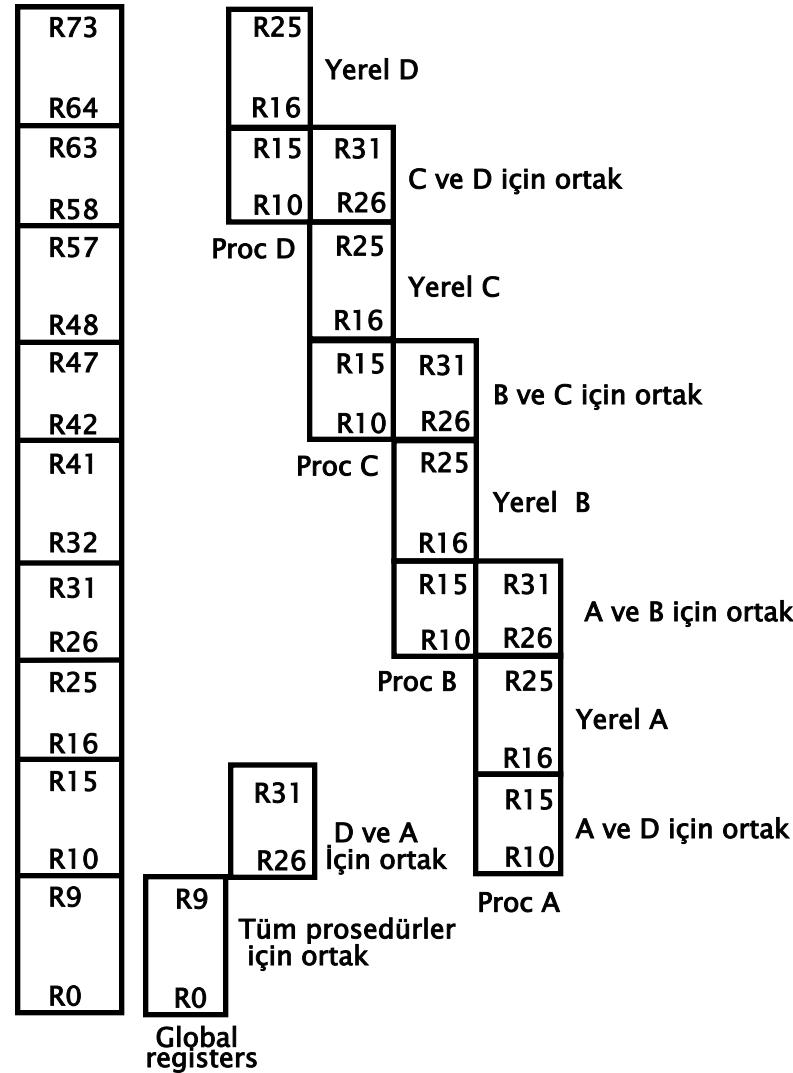
Saklayıcılar

- ▶ Çok sayıda genel amaçlı saklayıcı ile daha az belleğe erişime neden olması hız açısından önemli bir performans sağlar.
- ▶ Saklayıcı varış zamanı düşüktür, zira CPU bus kullanıcıları ve 1 saat çevriminde herhangi bir transferi gerçeklerler.
- ▶ Bellek yolu üzerinden belleğe okuma ve yazma daha çok saat çevrim gerektirir.
 - Bellek yolu donanımı işlemciden daha yavaş çalışır.
 - Bellek yoluna disk sürücüsü gibi diğer elemanlarda ulaşabileceğinden bir yarış paylaşım sorunu olabilecektir.
 - Saklayıcı kullanımı komutları basitleştirir
 - Load veya store komutlarının daha az kullanımına neden olur

Saklayıcı Penceresi Yaklaşımı

- Gözlemler
 - Procedure call / return en çok zaman tüketen işlemlerdir.
 - Tipik bir prosedür birkaç geçiş parametresi ve lokal değişken kullanır.
- Çözüm
 - Çoklu küçük saklayıcı kümesi kullan (windows), (herbiri farklı bir prosedüre tahsis edilen)
 - Bir prosedür otomatik olarak CPU yu farklı bir saklayıcı penceresi kullanmaya yöneltir.
 - Komşu prosedürler için pencereler parametre geçişine izin vermek için örtüşürler.

Örtüşmeli Saklayıcı Pencereleri



Örtüşmeli Saklayıcı Penceresi

► 3 tür saklayıcı kümesi vardır:

- Global saklayıcılar
 - Tüm prosedürler için mevcuttur
- Yerel pencere saklayıcıları
 - Herbir prosedüre has saklayıcı
- Pencere paylaşımı saklayıcı
 - İki prosedürün paylaştığı datayı tutan saklayıcı

Örtüşmeli Saklayıcı Penceresi

- ▶ G= Global Saklayıcı sayısı
 - ▶ L= Lokal Saklayıcı sayısı (her pencere için)
 - ▶ C= İki pencerenin ortak kullandığı pencere sayısı
 - ▶ W= Pencere sayısı

 - ▶ Window size= $L + 2C + G$
 - ▶ Register file= $(L+C)w + G$
- 
- ▶ G=10, L=10, C=6, W=4
 - ▶ Window size=32
 - ▶ Saklayıcı file=74

Berkeley RISC I

- 32-bit tümleşik devre CPU
- 32-bit adres, 8-, 16-, 32-bit data
- 32-bit komut formatı
- Toplam 31 komut
- 3 adresleme modu
 - register; ivedi; PC bağıl adresleme
- 138 registers
 - 10 global registers
 - 8 pencere (herbiri 32 register)

Berkeley RISC I Komut Formatları

Register mod: (S2 register belirler)

31	24	23	19	18	14	13	12	0
Opcode	Rd	Rs	0	Not used	S2			
8	5	5	1	8	5			

Register-ivedi mod (S2 operand belirler)

31	24	23	19	18	14	13	12	0
Opcode	Rd	Rs	1		S2			
8	5	5	1	13				

PC bağıl mod

31	24	23	19	18	0
Opcode	COND		Y		
8	5		19		

Berkeley RISC I

- ▶ Register 0 donanımsal olarak 0 yükülüdür.
- ▶ 8 bellek variş komutu vardır:
 - 5 load komutu
 - 3 store komutu
- ▶ load komutları:

LDL	load long
LDSU	load short unsigned
LDSS	load short signed
LDBU	load byte unsigned
LDBS	load byte signed

 - Burada long: 32 bit, short: 16 bit ve byte: 8 bit datayı gösterir.
- ▶ store komutları:

STL	store long
STS	store short
STB	store byte

Berkeley RISC I Komut Kümesi

Opcode	Operands	Register Transfer	Description
Data manipulation instructions			
ADD	Rs,S2,Rd	$Rd \leftarrow Rs + S2$	Integer add
ADDC	Rs,S2,Rd	$Rd \leftarrow Rs + S2 + \text{carry}$	Add with carry
SUB	Rs,S2,Rd	$Rd \leftarrow Rs - S2$	Integer subtract
SUBC	Rs,S2,Rd	$Rd \leftarrow Rs - S2 - \text{carry}$	Subtract with carry
SUBR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs$	Subtract reverse
SUBCR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs - \text{carry}$	Subtract with carry
AND	Rs,S2,Rd	$Rd \leftarrow Rs \wedge S2$	AND
OR	Rs,S2,Rd	$Rd \leftarrow Rs \vee S2$	OR
XOR	Rs,S2,Rd	$Rd \leftarrow Rs \oplus S2$	Exclusive-OR
SLL	Rs,S2,Rd	$Rd \leftarrow Rs \text{ shifted by } S2$	Shift-left
SRL	Rs,S2,Rd	$Rd \leftarrow Rs \text{ shifted by } S2$	Shift-right logical
SRA	Rs,S2,Rd	$Rd \leftarrow Rs \text{ shifted by } S2$	Shift-right arithmetic
Data transfer instructions			
LDL	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load long
LDSU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load short unsigned
LDSS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load short signed
LDBU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load byte unsigned
LDBS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load byte signed
LDHI	Rd,Y	$Rd \leftarrow Y$	Load immediate high
STL	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store long
STS	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store short
STB	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store byte
GETPSW	Rd	$Rd \leftarrow PSW$	Load status word
PUTPSW	Rd	$PSW \leftarrow Rd$	Set status word
Program control instructions			
JMP	COND, S2(Rs)	$PC \leftarrow Rs + S2$	Conditional jump
JMPR	COND,Y	$PC \leftarrow PC + Y$	Jump relative
CALL	Rd,S2(Rs)	$Rd \leftarrow PC$ $PC \leftarrow Rs + S2$ $CWP \leftarrow CWP - 1$	Call subroutine and change window
CALLR	Rd,Y	$Rd \leftarrow PC$ $PC \leftarrow PC + Y$ $CWP \leftarrow CWP - 1$	Call relative and change window
RET	Rd,S2	$PC \leftarrow Rd + S2$ $CWP \leftarrow CWP + 1$	Return and change window
CALLINT	Rd	$Rd \leftarrow PC$ $CWP \leftarrow CWP - 1$	Disable interrupts
RETINT	Rd,S2	$PC \leftarrow Rd + S2$ $CWP \leftarrow CWP + 1$	Enable interrupts
GTLPC	Rd	$Rd \leftarrow PC$	Get last PC