1986

## DOĞU AKDENİZ ÜNİVERSİTESİ

**BİLGİSAYAR MÜHENDİSLİĞİ BİLİMDALI**

**Bahar 19-20**

# CMPE 325 – Bilgisayar Mimarisi II

**DENEY 3:**

**Mıps içinde Modüler Proglama Atlama ve Linki Kullanarak (jal) ve Jump-Return (jr) Talimatları**

1. **Giriş:**

Biz programcılar prosedürleri kullanın veya sınıfta tartıştık programları yazarken alt yordamlar. Modüler bir program daha kolaydır sözde spagetthy veya düz programlardan daha iyi anlamak. Ayrıca, modüler programlama daha kompakt kodlama sağlar çünkü yeniden kullanma modüller mümkündür. Modüler programlamayı desteklemek için komut kümesi sonra bir prosedür aramak için bir teknik sağlamaktır gerekir ve prosedürden arama talimatının hemen yanındaki adrese dönün.MIPS komut seti modüler programlama için iki talimatları vardır.

***Jump-and-link*** (**jal**) and ***Jump-register*** (**jr**).

**...**

**jal ProcedureAddress**

**add $2,$5,$9 # next-instruction after the call**

**...**

**ProcedureAddress:**

**...**

**jr $31 AnotherProcedure:**

**...**

**jal** komutu yürütme sırasını belirtilen değere değiştirir

işlem adresi ve aynı anda aşağıdaki talimatın adresini **$31**. siciline kaydeder. Talimat adındaki "link" sözcüğü, çağrıdan sonra bir sonraki talimatın yerine getirilmesi için arayan adrese bir link oluşturulduğunu belirtir. Bu bağlantıya geri dönüş adresi denir ve **$31** kayıt defterinde saklanır.

Jr $addr komutu, jump-return veya jump-to-register olarak adlandırılır. Talimatların yürütme sırasını $addr register, kayıtlı olan adrese değiştirerek, o adrese bir atlamaya neden olur. **jr $31**  jal talimatıyla kaydedilen adrese **$31** atlayarak prosedürün yürütülmesini sonlandırır.

MIPS'de, yürütülecek bir sonraki talimatın adresi **Program Counter** (**PC**) tutulur. Sıralı uygulamada **PC** otomatik olarak 4 arttırılır, böylece hafızadaki bir sonraki talimatı, bir sonraki çalıştırma talimatı olarak gösterir. Yeni bir adres **PC** kopyalanarak bir atlama gerçekleştirilir Jump (**j**) ve Jump-and-link (**jal**) belirtilen adresin **PC**, kopyalanmasıyla gerçekleştirilir; bu, yürütme sırasını bu adrese yönlendirir. Jump-register belirtilen yazıcının içeriğinin **PC** yazmacına kopyalar. **jr $31** içeriğini kopyalayarak alt rutinden geri dönüşü gerçekleştirir **$31** ve bu **PC**, geri dönüş adresine bir sıçrama sağlar.

Bir prosedür başka bir prosedür çağırırsa, o zaman eski kaydın değeri **$31**

yığınına **last-in-first-out** (**LIFO**) *kaydedilmektedir*.

MIPS, parametrelerin nasıl iletileceğini ve prosedür çağrılarının yuvalanmasını nasıl destekleyeceğini düzenleyen iki sözleşme sunar. Kayıtlar belirli amaçlar için gruplandırılmıştır.

## $0 içerir 0, $1 assembler-psödo kodları için ayrılmıştır

* $2 **ve** $3 **, prosedürün geri dönüş değerinin değerini veya göstergesini arayandan arayan kişiye döndürmede kullanılır.**
*  $4**,** $5**,** $6 **ve** $7 **argümanları arayandan cihaza transfer etmede kullanılır.**
  + $8 **...** $15 **callee'deki yerel çalışmalar için kaydedilen kayıtlı kayıtlardır.**
  + $16 **...** $23 **arayanlar arasında uzun mesafeli çalışmalar için arayanların kaydedilen kayıtlarıdır.**

 $28 **statik veri segmentini gösteren global veri işaretçisidir.**

* + $29 **yığının en üstünde yer alan yığının işaretçisidir.**

Bir derleyici genellikle çağrıları hem callee hem de caller save konvansiyonunda uygular ve hızlı ve kompakt kodlamaya sahip en iyisini seçer.

Bu deneyde **sadece callee-save convention** sözde kodları uygulamak için izin verilen seçeneğiyle birlikte kullanacağız.

# Deney Çalışması:

**1. İç İçe Aramalar:**

Yapısal bir programda **jal** ve **jr** komutlarının nasıl kullanıldığını görmek için, MIPS montajına dönüştürülen aşağıdaki C programını inceleyin. Program, bir dizinin toplamını, özyinelemeli toplama çağrılarıyla belirtilen bir başlangıç ​​dizininden bir bitiş dizinine bulur. Toplamı veri segmentine yazan ilgili MIPS montaj programını yazın ve uygulayın.

// Function computes sum of the array elements which starts from first element (0) and ends when reaches to (size-1)th index

int sum (int arr[ ], int size)

{

If (size==0)

return 0;

else

return sum(arr, size-1) + arr[size-1];

}

The following code is the corresponding MIPS assembly source.

|  |  |  |
| --- | --- | --- |
| .data 0x10000000 A: .word 3,5,6,2,0,4  .text 0x00400000  .globl main  main:  la $a0, A li $a1, 6 jal fun  move $s0, $v0 syscall  fun: addi $sp, $sp, -8 # Adjust sp  addi $s0, $a1, -1 # Compute size - 1  sw $s0, 0($sp) # Save size - 1 to stack sw $ra, 4($sp) # Save return address bne $a1, $zero, L1 # branch ! ( size == 0 ) li $v0, 0 # Set return value to 0  addi $sp, $sp, 8 # Adjust sp  jr $ra # Return | | |
| L1: | move $a1, $s0 | # update second arg |
|  | jal fun |  |
|  | lw $s0, 0($sp) | # Restore size - 1 from stack |
|  | li $t7, 4 | # t7 = 4 |
|  | mult $s0, $t7 | # Multiple size - 1 by 4 |
|  | mflo $t1 | # Put result in t1 |
|  | add $t1, $t1, $a0 | # Compute & arr[ size - 1 ] |
|  | lw $t2, 0($t1) | # t2 = arr[ size - 1 ] |
|  | add $v0, $v0, $t2 | # retval = $v0 + arr[size - 1] |
|  | lw $ra, 4($sp) | # restore return address from stack |
|  | addi $sp, $sp, 8 | # Adjust sp |
|  | jr $ra | # Return |

Bilgisayarı **ØxØØ4ØØØØØ** konumuna getirdikten sonra **f1Ø** tuşunu kullanarak kodu adım adım yükleyin ve izleyin (dosyayı Laboratuarda zaman kazanmak için laboratuvar saatinden önce hazırlamak iyi bir fikirdir). Yığın listesinin içeriğini, **jr** **$ 31** komutu ilk kez yürütüldüğünde değerin hangi kayıttan kaynaklandığını gösteren raporunuza yazın. Bir kayıt yığmaya basıldığında, kayıtların hangi kaynaklardan kaynaklandığını doldurabilmeniz için her seferinde kayıt numarasını not etmeniz gerekir.

## Bölüm 2 Proglama Alıştırması

Bu bölümde, yardımcınız size içinde bir işlev veya prosedür içeren bir C programı kaynağı verecektir. İlgili kodu yazmanız ve programınızın doğru çalıştığını doğrulamanız gerekir.

İsim: Öğrenci Numarası:

Gönderen (Yrd.): Tarih:dd/mm/yy / /

1986

## Doğu Akdeniz Üniversitesi

**Bilgisayar Mühendisliği Bilimdalı**

**Bahar 19-20**

# CMPE 325 – Bilgisayar Mimarisi II DENEY 1 – Rapor Formu

## Bölüm1:

Yığın içeriği, jr $ 31 komutunun ilk uygulanmasında.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Address** |  | **wordØ** | **word1** | **word2`** | **word3** |
| **addr.** |  |  |  |  |  |
| register |  |  |  |  |  |
| **addr.** |  |  |  |  |  |
| register |  |  |  |  |  |
| **addr.** |  |  |  |  |  |
| register |  |  |  |  |  |
| **addr.** |  |  |  |  |  |
| register |  |  |  |  |  |
| **addr.** |  |  |  |  |  |
| register |  |  |  |  |  |

İç içe aramaların derinliği nedir?..........

Yığını işgal eden kelimelerin sayısı nedir?..........

## Bölüm 2: Program Alıştırması

1. MIPS koduna çevrilecek C programı....................
2. Buraya hesaplanan dizi elemanlarının içeriğini doldurun,(onaltılık olarak):

................... , ...................... , ................... , ....................

................... , ...................... , ................... , ....................

................... , ...................... , ................... , ....................

................... , ...................... , ................... , ....................

................... , ...................... , ................... , ....................

Derecelendirme:

Sınav Performansı:

Labaratuvar Performansı:

Yrd. Gözlemler: