DOĞU AKDENİZ ÜNİVERSİTESİ BAHAR 2012-2013
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BLGM-324 BİLGİSAYAR MİMARİSİ
DENEY #2
DİZİLERE ERİŞİMDE MIPS BELLEK TALİMATLARI

Amaç: Veri bölütü kullanımını ve tek-modüllü dizi kullanan çevirici programlarını tanımak. Değişkenlerde adres yerine sembol kullanarak anlaşılabilir çevirici programların geliştirilmesi.

Giriş: Yüksek seviyeli dillerdeki programlarda değişkenleri aritmetik komutlarda doğrudan kullanabilsek te, MIPS çeviricide aritmetik komutun işlenenleri bellekteki değişkenler olamaz. Bu komutların işlenenleri sadece az sayıdaki genel amaçlı yazmaçlar olabilir. MIPS mimarisinde ($0, $1, $2,…., $31) ile gösterilen toplam 32 yazmaç bulunur. Bu yazmaçlar sınırlı sayıda olmaları nedeniyle bellekte tutulan programlama dili değişkenlerinden farklıdır. Sınırlı sayıda olan bu yazmaçların içeriklerini okuma ve yazma hızı çok yüksektir. Yazmaç sayısı daha da fazla olsaydı, adres kodlarını çözmek için daha fazla kapı gerekecek, ve bu yüzden yüksek gecikmeler oluşacaktı. Bu gibi nedenlerle adres kodunu çözen kapılarda oluşan gecikmeler saat çevrim zamanını arttırır ve işlemciyi yavaşlatır.

Çoğu programlama dillerinde tek öğeli basit veri tiplerinin yanısıra tamsayı diziler gibi daha karmaşık veri yapıları da yaygın olarak kullanılır. Bu karmaşık veri yapıları işlemcideki yazmaç sayısından fazla veri elemanları içerebilir. Verileri sınırlı sayıda yazmaca saklayan işlemcinin böyle bir işlemci büyük veri yapılarına erişebilmesi ve bu yapıları işleyebilmesi işlemcinin milyonlarca veri elemanını saklayabileceği bellek sistemine erişebilmesiyle çözülür. Böylece, dizi gibi büyük veri yapıları bellekte saklanır ve verinin sadece işlenecek olan bölümü gerekli yazmaça yüklenir. MIPS mimarisi dizilere yönelik bellek aktarma komutlarını da sağlar.

Bellekten yazmaça veri aktarımı **load word (lw)** komutu ile sağlanır.

lw $hedef, offset($baz)

Bellekte **offset+$baz** adresindeki değer **$hedef** 'e yükleniyor. Örneğin;

lw $20, 10000 ($0) : (10000+$0) adresindeki veriyi yazmaç $20 ye yükler.

Yazmaçtan belleğe veri aktarımı **store word(sw)** komutu ile yapılır.

sw $kaynak, offset($baz)

$kaynak yazmaçını bellekte **offset+$baz** adresinden başlayan 32 bite saklar. Örneğin;

sw $10, 20000($2): yazmaç **$10** daki değeri bellekte adres (**20000+$2) '**e saklar.

Hem **lw** hem de **sw** komutlarının ofseti 16-bitlik işaretli tamsayıdır.

Ancak, sözde-komutlara izin verildiğinde, **lw** ve **sw** komutlar 32-bitlik adresleri de kabul eder ve 16-biti geçen adresler için sözde-komutları gerçekleştirmek üzere 16-bitlik ofset için **lui, add**, **sw** yada **lw** komutlarının birleşimini kullanır.

DENEYSEL ÇALIŞMA:

Bölüm 1:

Bellek kullanan komutları (**lw** ve **sw**) anlamak üzere aşağıdaki programı **"exp2a.s"** dosyasına yazarak MIPS çevirici programını çalıştırınız. Bu programda örneğin:

"la $dest,<label>" gibi MIPS sözde-komutları da kullanılmıştır.

Lw ve **sw** sözde-komutları 32 bitlik ofsete sahiptir. Bundan dolayı, bu sözde-komutları kullanabilmek için SPIM'deki pseudo-code diyalog-kutusu işaretlenmiş olmalıdır. **(alt-s,s)**

Not: program sayacı PC yi 0x00400000 adresinden başlayacak ilk değeri vermeyi unutmayınız.

```
.data  0x10000000
# A[] is array with a trailing zero.
A: .word  100,120,35,50,83,530,0,0
Count:
  .word 0
Sum:
  .word 0
Result:
  .word 0
Remainder:
  .word 0
EndofData:
  .word -1
# End of data segment, code starts here from address 0x00400000
# Turn-off bare-machine, and turn-on pseudo-instruction options.
```

```
.text
  .globl main
main:
# finding average of A[]
  la   $2,A          # pseudo-instruction load-address A[.]
  or   $8,$0,$0      # 0 -> $8 , count
  or   $10,$0,0      # 0 -> $10, sum
slp:
  lw   $11,0($2)     # A[i] -> $11
  beq  $11,$0,slx
  add  $10,$10,$11   # A[0]+...+A[i] -> $2
  addi $8,$8,1       # ++count
  addi $2,$2,4       # address of(A[i+1]) -> $2
  j    slp           # loop until getting the zero
slx:
# save the count
  sw   $8,Count($0)
  sw   $10,Sum($0)
# divide $10 by $8, use count of repeating subtractions.
  div  $10,$8            # quotient is in LO, reminder is in HI
  mflo $11               # move from LO to $destination
  mfhi $10               # move from HI to $destination
  sw   $11,Result($0)    # mean
  sw   $10,Remainder($0)  # and this is the reminder of division
  syscall       # for the sake of SPIM add below one empty line
```

Bölüm 2-PROGRAM YAZMA

A[] dizisinde verilen dört 32-bitlik sayının (30, 40, 50, 60 gibi) her elemanı için toplamlarının elemana oranını res[] dizisine koyacak bir program yazın.
Örneğin A[]=(30, 40, 50, 60) için toplamları 180 ediyor. Program çalışınca
(res[0] ← 180/30; res[1] ← 180/40; res[2] ← 180/50; res[3] ← 180/60) olacak.

```
.data  0x10000000
A:
   .word  20,40,60,80
res:
   .word 0,0,0,0
   .text
   .globl main
main:

        PROGRAMI TAMAMLAYINIZ!



syscall         # for the sake of SPIM add below one empty line
```

SPIM çeviricinin doğru çalışabilmesi için en sona bir boş satır ekleyiniz.

Programı derleyip çalıştırdıktan sonra elde ettiğiniz sonucu raporunuza doldurup asistanınıza iletiniz.

İsim: _____ Öğrenci No: _____

DOĞU AKDENİZ ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ **BLGM-324 BİLGİSAYAR MİMARİSİ**

DENEY #2

Bölüm-1: Veri bölütünün satırlarını aşağıdaki gibi doldurun

```
A: .word ____, ____, ____, ____, ____, ____, ____, ____
________:
   .word 0
________:
   .word 0
________:
   .word 0
________:
   .word 0
EndofData:
   .word -1
```

program çalıştıktan sonra veri bölütünden okuduğunuz değerleri 16 lı sayı olarak yazın.

0x10000000					
0x10000010					
0x10000020					
0x10000030					
0x10000040					
0x10000050					
....					

Bölüm-2: Programlama Deneyi

1- Veri kümeniz:

0x10000000					
0x10000010					

2- Değişikliklerden sonra:

0x10000000					
0x10000010					
0x10000020					
0x10000030					
0x10000040					
0x10000050					
....					

Notlandırma: