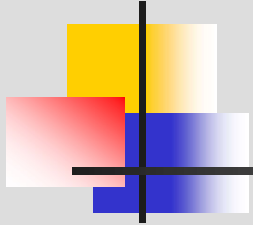




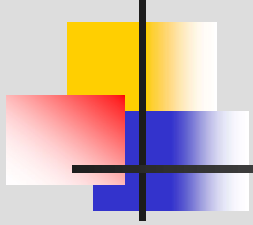
# Bilgisayar Organizasyonu

---

Prof. Dr. İsmail Kadayıf  
Çanakkale Onsekiz Mart Üniversitesi  
Bilgisayar Mühendisliği



<http://www.ecs.umass.edu/ece/vspgroup/burleson/courses/232/spim/>

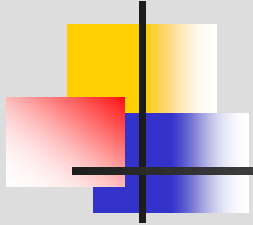


# Programlama

---

## ➤ Programlama dilleri

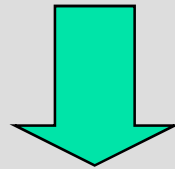
- High-level Languages (Yüksek seviyeli diller) (C, C++, Java, Fortran, ...)
  - kompleks işlemler ile ifade edilebilir
  - programlama işlemini kolaylaştırır
- Low-level Languages (Alçak Seviyeli diller) (C, assembly, makine dili, ...)
  - işlemleri basit (program yazımı zor ve kolayca hata yapılmaya müsait)



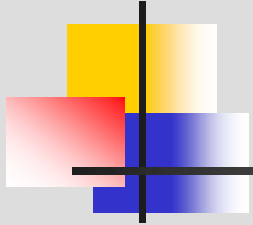
# Programlama

---

```
hypotenuse := sqrt (sqr(a) + sqr(b) );
```

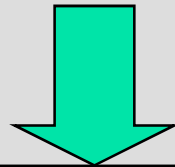


```
mul  asquare, a, a  
mul  bsquare, b, b  
mdd  sumsquare, asquare, bsquare  
sqrt hypotenuse, sumsquare
```

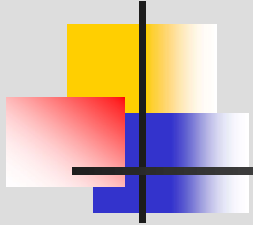


# Programlama

```
hypotenuse := sqrt (sqr(a) + sqr(b) );
```



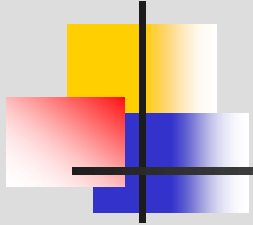
```
l.s    $f0, aa
mul.s  $f0,$f0,$f0
l.s    $f2, bb
mul.s  $f2,$f2,$f2
add.s  $f4,$f0,$f2
s.s    $f4, sumsquare
neg.s  $f6,$f4
s.s    $f6, hypotenuse
```



# Programlama

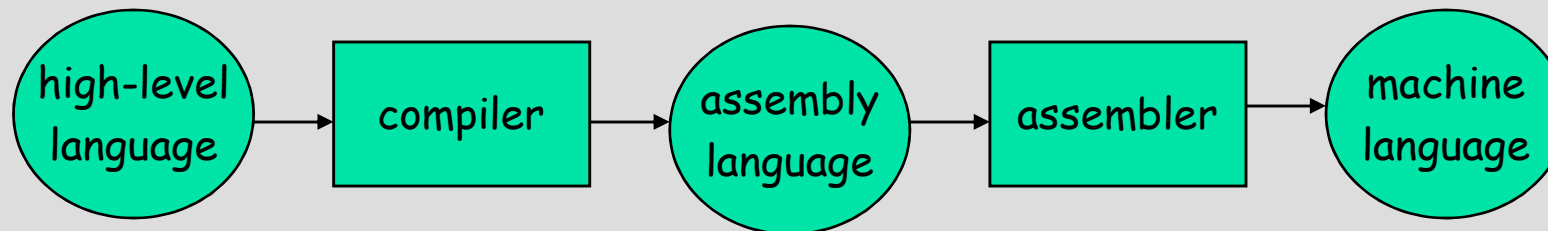
---

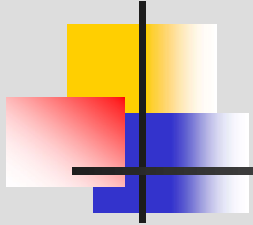
```
0011 1100 0000 0001 0001 0000 0000 0000
1100 0100 0010 0000 0000 0000 0000 0000
0100 0110 0000 0000 0000 0000 0000 0010
.....
0011 1100 0000 0001 0001 0000 0000 0000
```



# Compilation (Derleme)

---





# Bilgisayar Mimarisi

---

- Assembly dilinde programlama bilgisayar mimarisine bağlı olarak değişiklik gösterir (Instruction Set Architecture, ISA)
- ISA az sayıda bir işlemlerden (instruction, komut) oluşur. Komutlar hardware tarafından doğrudan çalıştırılabilirler.
- bilgisayarın makine dili üreticisi tarafından belirlenir
- MIPS RISC architecture

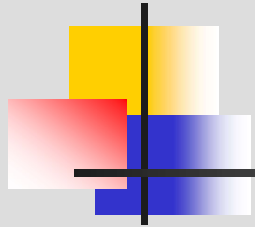




# Low-level Languages (MIPS Mimarisinde)

---

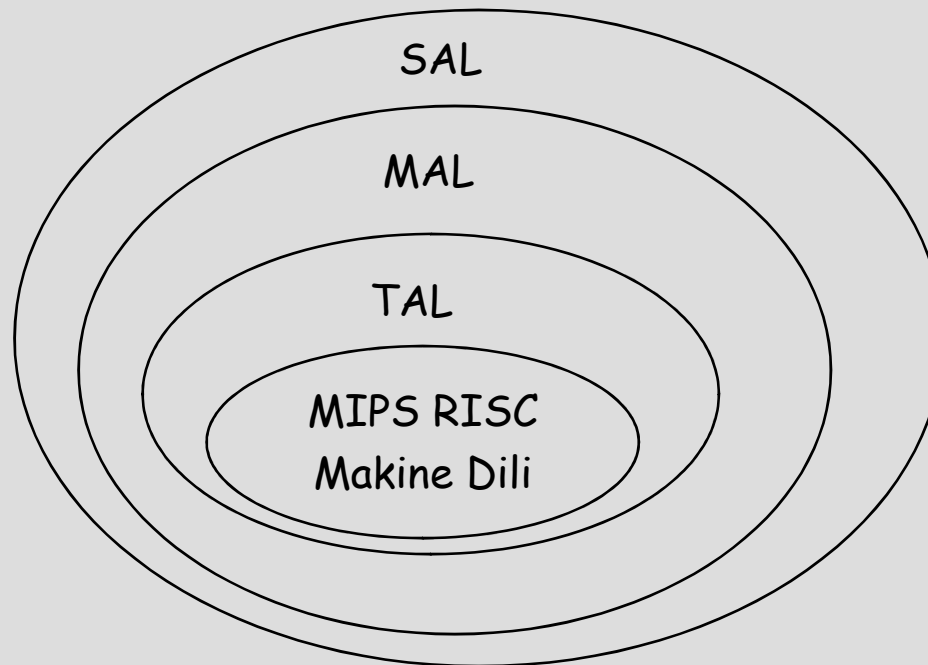
- True Assembly Language (TAL): ISA daki komutlardan oluşur.
  - TAL dan daha aşağı seviyede programlama yapılamaz
- More Abstract Language (MAL) (MIPS Assembly Language)
  - MIPS RISC assembler, MAL programı önce TAL programa, daha sonra TAL daki her komutu MIPS RISC makine koduna çevirir
- MAL programlar TAL programlar kadar etkindir.
- MAL da programlama TAL da programlamadan daha kolay
- Simple Abstract Language (SAL). MAL dan daha yüksek seviyelidir. Yüksek Seviyeli Dillerle Assembly Dili arasındaki boşluğu doldurmak için kullanılır.

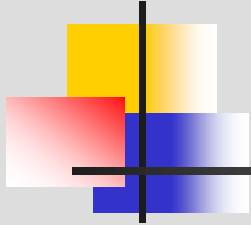


# MIPS RISC

## Mimarisindeki Soyutlama

---

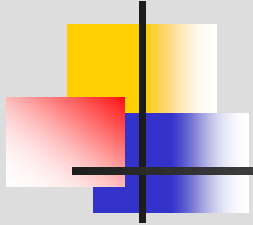




# Bilgisayar Organizasyonu

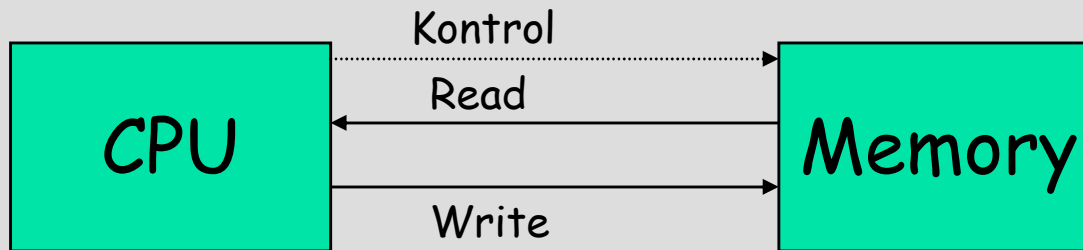
---

- Programın çalışması sırasında, işlemci (processor) komutları yürütür.
  - Komutlar değişkenlere nasıl değerler atandığını (bellek) belirler.
- Memory (Bellek) boyu sabit fakat oldukça geniştir.
  - programda tanımlanan her bir variable (değişkene) karşı bir bellek gözü (memory location ) tahsis edilir.
  - herhangi bir değişkenin değerinin okunması/değişkene değer atanması, bu değişkenin bulunduğu bellek gözünün okunması/yazılması anlamına gelir (load/store).
  - **binding** :bellek gözünün bir değişkene tahsis edilmesi

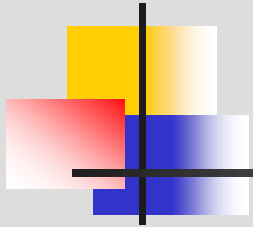


# Bilgisayar Organizasyonu

---



CPU (Central Processing Unit):  
Merkezi İşlem Birimi



```
part1:  x1:=x2+x3;
        x4:=x2-x5;
        if(x4=x1) then goto 10;
        x1:=x2+x5;
```

```
10:
```

```
part1:  add x1, x2, x3
        sub x4, x2, x5
        beq x4, x1, part2
        add x1, x2, x5
```

```
part2:
```



# Program Yürütme Sırasında Temel Adımlar

---

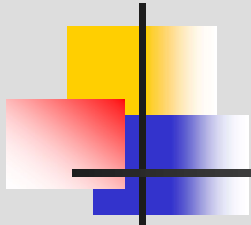
1. Instruction Fetch (Komut Getirimi)
2. Program Counter (PC) update (Program Sayacını güncelleme)
3. Instruction Decode (Komut Çözümü)
4. Operand Load (Operand Yükleme)
5. Operation Execution (İşlem Yürütümü)
6. Storage of results (Sonuçların belleğe yazılması)



# SAL'da Programlama

---

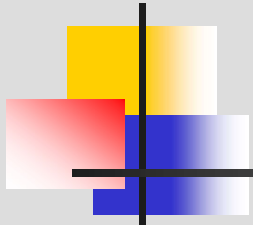
```
program summation(input, output);  
var  
    n:      integer;  
    sum:    integer;  
    i:      integer;  
  
begin  
    write('Please Enter a positive integer: ');  
    readln(n);  
    writeln;  
    sum:=0;  
    for      i:=0 to n do  
        sum:=sum+i;  
    write('The sum of the first '); writeln(n);  
    write(' integers is '); writeln(sum);  
  
end.
```



```
                                .data
str1:    .asciiz    "Please enter a positive integer: "
str2:    .asciiz    "The sum of the first "
str3:    .asciiz    "integers is "
newline: .byte      '\n'

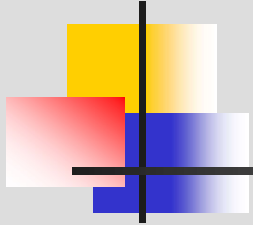
n:       .word      0
sum:     .word      0
i:       .word      0
temp:    .word
```





```
__start:  puts    str1
          get      n
          put      newline

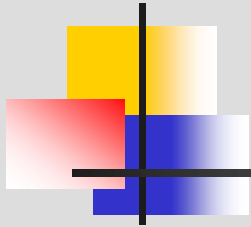
          for:      sub    temp, n, i
                   bltz    temp, endfor
                   add     sum, sum, i
                   add     i, i, 1
                   b       for
          endfor:  puts    str2
                   put      n
                   puts     str3
                   put      sum
                   put      newline
          done
```



## Branch Instructions (SAL)

---

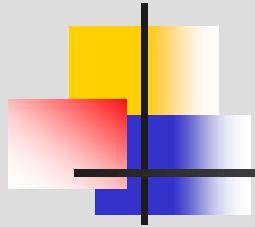
b	label	# goto label
beq	x, y, label	# if x = y then goto label
bne	x, y, label	# if x <> y then goto label
blt	x, y, label	# if x < y then goto label
bgt	x, y, label	# if x > y then goto label
ble	x, y, label	# if x <= y then goto label
bge	x, y, label	# if x >= y then goto label



## Branch Instructions (SAL)

---

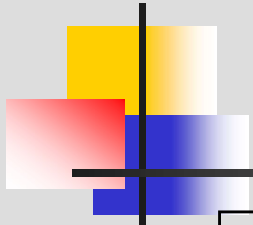
bltz	x, label	# if $x < 0$ then goto label
bgtz	x, label	# if $x > 0$ then goto label
blez	x, label	# if $x \leq 0$ then goto label
bgez	x, label	# if $x \geq 0$ then goto label
beqz	x, label	# if $x = 0$ then goto label
bnez	x, label	# if $x \neq 0$ then goto label



# Communication Instructions

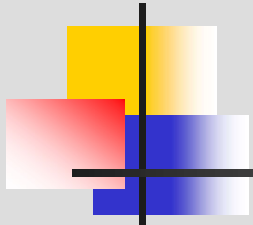
---

```
get    x      # readln(x) (x .word veya .float)
get    x      # read(x) (x .byte)
put     x      # write(x) (x, . word, .float, .byte)
puts   stringname  # write(string) (.asciiz)
```

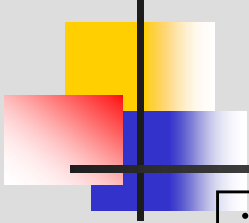


```
if (A > 0) then  
  B := C div A  
else  
  B := A+10;
```

```
                blez    A, elsepart  
                div     B, C, A  
                b       endif  
elsepart:      add     B, A, 10  
endif:
```



	bgtz	A, ifpart
	add	B, A, 10
	b	endif
ifpart:	div	B, C, A
endif:		

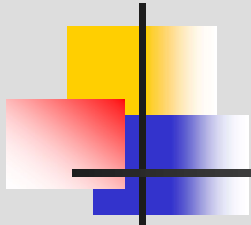


```
if ( (A = B) or ( C < D ) ) then
begin
A := A + 1;
B := B - 1;
D := A + C;
end;
```

```
                beq    A, B, do_if
                blt    C, D, do_if
                b      end_if

do_if:  add    A, A, 1
        add    B, B, -1
        add    D, A, C

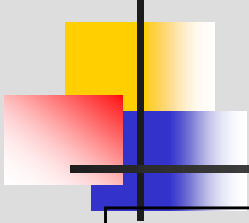
endif:
```



```
if ( (A = B) and ( C = D ) or ( E < 0 ) ) then
begin
    A := A + 1;
    C := E;
end;
```

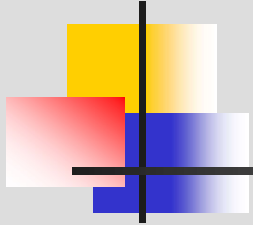
```
                                bne    A, B, check_E
                                beq    C, D, do_if
check_E:                        bgez    E, end_if
do_if:                          add     A, A, 1
                                move    C, E
end_if:
```





```
result := 1;
counter := exponent;
while (counter > 0 ) do
begin
    result := result * base
    counter := counter - 1;
end;
```

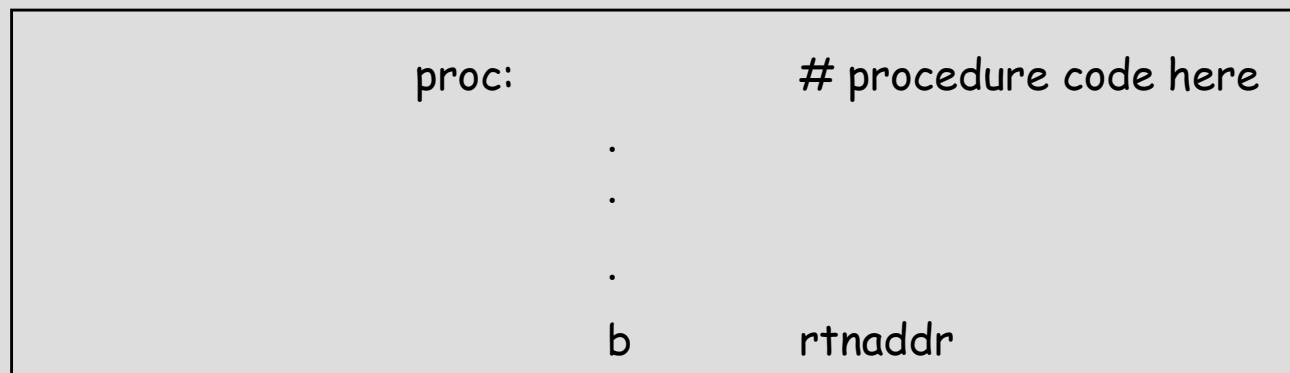
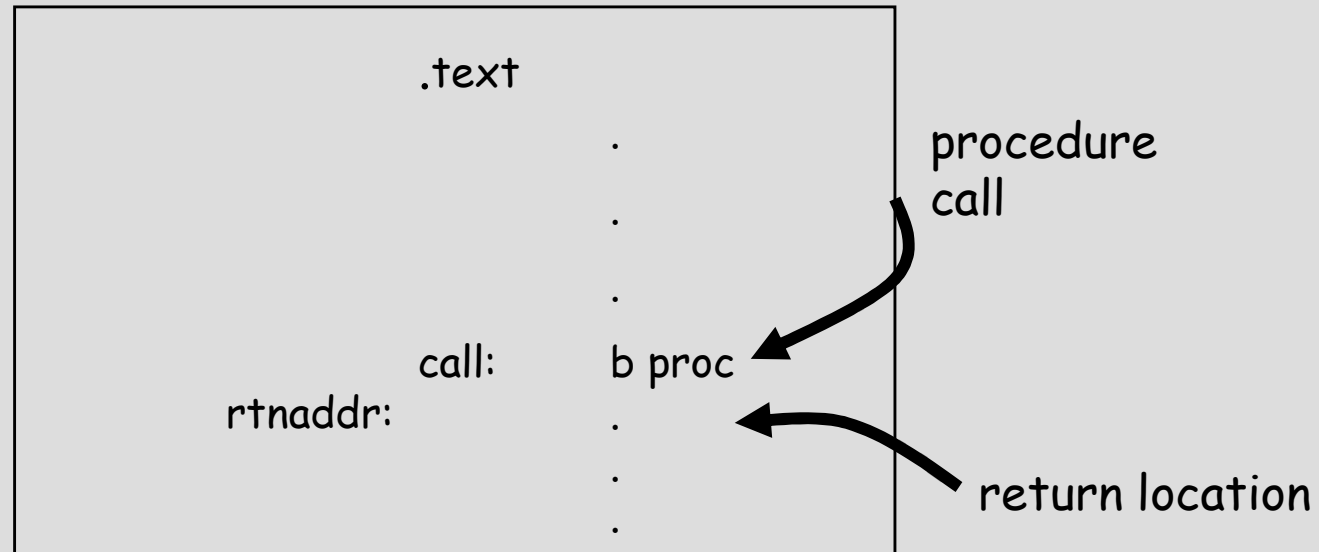
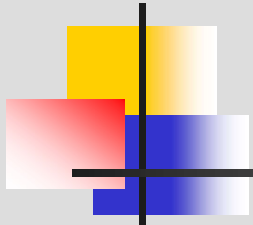
```
while:          move    result, 1
                move    counter, exponent
                blez     counter, endwhile
                mul      result, result, base
                sub      counter, counter, 1
                b         while
endwhile:
```

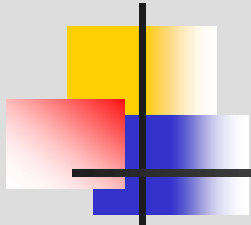


# Procedure

---

- Programın modüler olmasına yardımcı olurlar.
- Procedure kullanımı
  - SAL sınırlamaları
    - Procedure parametre aktarma olanağı yoktur
    - Recursive procedure çağırma olanağı yoktur
- Temel adımlar
  - Save return address
  - Procedure call
  - Execute procedure
  - Return





```
                                la      proc1_ret, ret_addr1
                                b      proc1
return_addr1:
```

```
                                la      proc1_ret, ret_addr2
                                b      proc1
return_addr2:
```

```
proc1:                          # procedure code here
.
.
.
b      (proc1_ret)
```