2.

Principio de diseño 1: **La simplicidad favorece la regularidad.**

**Ejemplo: Compilando dos asignaciones a código MIPS**  
Este segmento de un programa en C contiene cinco variables a, b, c, d y e:  
a = b + c;  
d = a – e;  
  
La traslación desde C al ensamblador MIPS la realiza el compilador correspondiente, mostrar el código MIPS que produciría para estas asignaciones.  
  
**Respuesta:**  
  
Una instrucción MIPS opera con dos operandos fuentes y pone el resultado en un operando destino. Para dos asignaciones simples, solo se requiere de dos instrucciones:  
  
add a, b, c  
sub d, a, e

Principio de diseño 2: **Si es mas pequeño es más rápido.**

**Ejemplo:**   
En este ejemplo se repite el ejemplo anterior, pero ahora se utilizarán los nombres de los registros en el código ensamblador.  
f = (g + h) – (i + j);   
  
**Respuesta:**   
Supongamos que las variables f, g, h, i y j se asocian con los registros $s0, $s1, $s2, $s3 y $s4, respectivamente.  
add $t0, $s1, $s2 # El registro $t0 contiene $s1+ $s2  
add $t1, $s3, $s4 # El registro $t1 contiene $s3+ $s4  
sub $s0, $t0, $t1 # f obtiene $t0 – $t1, que es (g + h) – (i + j)

Principio de diseño 3: **hacer el caso común más rápido.**

Resolver las variables que se inicialicen. Y lo que se utiliza frecuentemente.

Ejemplo:

ADD %g0,5,%L0

ADD %g0,-5,%L1

3.

Convertir a instrucciones de bajo nivel.

int x=0;

int y =8;

int z = 1;

y=x+3;

z=z+3;

x=(x-z)+(3+y);

inicializar variables:

x:%L0

y:%L1

z::%L2

add %g0,0,%L0

add %g0,8,%L1

add %g0,1,%L2

**y=x+3;**

add %L0, 3 , %O0

**z=z+3;**

add %L2, 3, %O0

**x=(x-z)+(3+y);**

add %L1, 3, %L1

sub %L0, %L2, %L0

add %L0,%L1, %O0

4.

**a[4]=a[2]+x**

a:%L0

x:%L1

ld [%L0+2\*4],%L2

add %L2,%L1,%L1

st %L1,[%L0+4\*2]

**y[0]=y[40]+13**

y:%L0

ld [%L0+13\*4],%L1

add %L1,3,%L1

st %L1,[%L0+4\*2]

5.

Convertir a lenguaje de máquina.

a.

int main(){

int i =3; p=2;

return i+3;

}

i:%L0

p:%L1

ADD %g0,3,L0 = OP:10, Rd:10000,OP3:000000,Rs1:00000,i:1,unused(zero) y Rs2:0000000000011

=**10100000000000000010000000000011**

ADD %g0,2,L1 = OP:10, Rd:10001,OP3:000000,Rs1:00000,i:1,unused(zero) y Rs2:0000000000010

=**10100010000000000010000000000010**

ADD %L0,3,%O0 = OP:10, Rd:01000,OP3:000000,Rs1:00011,i:1,unused(zero) y Rs2:0000000000011

=**10010000000000001110000000000011**

b.

int main(){

int p=3; x=1; z=4;

int w=0;

w=(p+40)+(x-z);

return 0;

}

p:%L0

x:%L1

z:%L2

ADD %g0,3,L0 = OP:10, Rd:10000,OP3:000000,Rs1:00000,i:1,unused(zero) y Rs2:0000000000011

=**10100000000000000010000000000011**

ADD %g0,1,L1 = OP:10, Rd:10001,OP3:000000,Rs1:00000,i:1,unused(zero) y Rs2:0000000000001

=**10100010000000000010000000000001**

ADD %g0,4,L2 = OP:10, Rd:10010,OP3:000000,Rs1:00000,i:1,unused(zero) y Rs2:0000000000100

=**10100100000000000010000000000100**

ADD %L0,40,%L0 = OP:10, Rd: 10000,OP3:000000,Rs1:00011,i:1,unused(zero) y Rs2:0000000101000

=**10100000000000001110000000101000**

SUB %L1,%L2,%L1 = OP:10, Rd:10001,OP3: 000100,Rs1:00001,i:0,unused(zero): 00000000,Rs2:00100

=**10100010001000000100000000000100**

ADD %L0,%L1,%O0 = OP:10, Rd:01000,OP3:000000,Rs1: 101000,i:0,unused(zero): 0000000,Rs2:00100

=**10010000000010100000000000000100**

6.

OR.

n=-12,

**n:%L1**

**OR %g0,-12,%L1**

a=-11,

**a:%L2**

**OR %g0,-11,%L2**

b=-14

**b:%L3**

**OR %g0,-14,%L3**