

1. La memoria de un ordenador tiene capacidad para 65536 palabras de 25 bits cada una. El código de instrucción de ese ordenador se divide en cuatro partes: un código de operación, un bit de modo de direccionamiento directo/indirecto, dos bits para especificar un registro del procesador y una parte de dirección.

- a) ¿Cuál es el máximo número de operaciones que se pueden codificar si cada una de ellas sólo puede ocupar una palabra?
- b) ¿Cuántos registros hay en el ordenador?
- c) ¿Cuántos bits tiene el PC (program counter)?

Código = op + bit modo + 2 bits reg. + direc.

$$\log_2 65536 = 16,32 \Rightarrow 20 \text{ bits para cada dirección.}$$

a) $2^2 = \text{op} + 20$

$\boxed{N^{\circ} \text{ op} = 4}$

b) Usando dos bits $\Rightarrow 2^2 \text{ reg.} = \text{prey. S.}$

c) 20 bits puesto que dirección usa instrucción

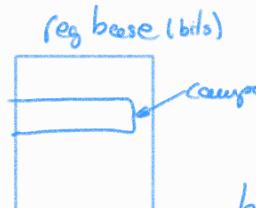
2. El direccionamiento por base ponen en juego 4 longitudes:

- L1: número de bits del campo que especifica un registro base.
- L2: número de bits del campo que especifica un desplazamiento.
- L3: número de bits de un registro base.
- L4: número de bits de una dirección de memoria.

Para un conjunto dado de valores de los registros base:

- a) ¿A cuántas direcciones distintas se puede hacer referencia?
- b) ¿Qué fracción de la memoria total se puede direccionar?

D(R_b, R_i, S)



bits reg base + campo + desplazamiento

a) Sería la suma de todos $\approx 2^{12}$

b) ?

3. En un ordenador hipotético se puede acceder directamente a 16 MBytes. La CPU de este ordenador dispone de 16 registros base. Determine el tamaño del campo desplazamiento de modo que sea posible direccionar el 6,25% del espacio total de direcciones sin cambiar el contenido de los registros base.

16 MBs en dirs

$\hookrightarrow 16384 \text{ KB}$

$\hookrightarrow 16777216 \text{ B}$

$\hookrightarrow 134217728 \text{ bits}$ calcular $\Rightarrow 27 \text{ bits}$

Registros 16 $\Rightarrow 4 \text{ bits}$

\downarrow
6,25%

\downarrow
838861 bits calcular $\Rightarrow 20 \text{ bits}$

} para direc. especiales

D(R_b, R_i, S)
12 bits 4 bits 1 bit 2 bits
↓ ↓ ↓ ↓ ↓ ↓

Son 12 bits sin contar la escritura

- 4.** Un procesador hipotético puede acceder a 4 GB de RAM. El procesador dispone de 16 registros base. Determine el tamaño del campo desplazamiento de modo que sea posible direccionar, mediante direccionamiento relativo a registro base, el 6,25% del espacio total de direcciones sin cambiar el contenido de los registros base en cada uno de los dos siguientes supuestos:

- a) Todos los registros base contienen el mismo valor. *← a qué se refiere?*
b) Todos los registros base contienen un valor tal que las zonas que se pueden direccionar a partir de ellos son disjuntas.

4GB $\xrightarrow{62.5\%}$ 0.25GB \rightarrow 2147483648 bits $\xrightarrow{\text{real}}$ 31 bits

16 reg \Rightarrow 4 bits

a) 31 bits - 4 bits = 27 bits para despl.

b) 4 bits + 27 bits = 31 bits \rightarrow 21 bits para el desplazamiento.
 \downarrow \downarrow
por por reg

- 5.** Un microprocesador dispone de 4 registros base y puede direccionar por bytes hasta 1 MB. El único modo de direccionamiento de memoria existente es relativo a registro base. La dirección de memoria efectiva se calcula según: EA = base * 16 + desplazamiento.

- a) ¿Cuál es el tamaño del campo desplazamiento si sabemos que es posible direccionar como máximo el 25% del espacio total de direcciones sin cambiar el contenido de los registros base?
b) ¿Cuál es el tamaño de los registros base?

a) 21 bits para direccionar toda la parte efectiva. Sup. cada reg tiene el mismo tam. \Rightarrow
2097152 bits de zona efectiva \Rightarrow por reg: 524288 bits \rightarrow 19 bits para el despl. y 2 para reg.

$$EA = 2 \text{ bits} \cdot 16 + 19 \text{ bits}$$

b) Tamaño de 64 kB

- 6.** Dados los valores de memoria que se indican a continuación y una arquitectura de acumulador, ¿qué valores cargan en el acumulador las instrucciones 1, 2, ..., 6? La palabra 20 contiene 40, la 30 contiene 50, la 40 contiene 60 y la 50 contiene 70.

1. LOAD(immediato) 20
2. LOAD(directo) 20
3. LOAD(indirecto) 20
4. LOAD(immediato) 30

?

5. LOAD(directo) 30
6. LOAD(indirecto) 30

7. Compare las máquinas de 0, 1, 2 y 3 direcciones elaborando con los repertorios de instrucciones de cada una de ellas un programa que permita realizar la operación:

$$X = (A+B \cdot C) / (D-E \cdot F)$$

sin cambiar el valor de A, B, C, D, E, F, almacenadas en memoria.

Los repertorios de instrucciones son:

M0:	M1:	M2:	M3:
PUSH MCARGA M	MOV X, Y (X:=Y)	MOV X, Y (X:=Y)	ADD X, Y, Z (X:=Y+Z)
POP M	ALMACENA M ADD X, Y (X:=X+Y)	SUB X, Y (X:=X-Y)	ADD X, Y, Z (X:=Y-Z)
ADD	ADD M	MUL X, Y (X:=X*Y)	SUB X, Y, Z (X:=Y*Z)
SUB	SUB M	MUL X, Y (X:=X/Y)	MUL X, Y, Z (X:=Y/Z)
MUL	MUL M	DIV X, Y (X:=X/Y)	DIV X, Y, Z (X:=Y/Z)
DIV	DIV M		

donde M es una dirección de memoria de 16 bits. X, Y, y Z son direcciones de memoria (16 bits) o de registros (4 bits). La máquina de 0 direcciones utiliza una pila y la de 1 un acumulador. Si se consideran códigos de operación de 8 bits. ¿Cuántos bits necesita cada máquina para calcular X?

L>?

$N \rightarrow 16 \text{ bits} \xrightarrow{\text{cod}} 4 \text{ bits}$

$x, y, z = 4$

$\text{reg} \rightarrow 2 \text{ bits cod.}$

$\text{codop} \rightarrow 3 \text{ bits cod}$

Entonces $1, 18 + 3 + 9 * 4 + 4 * 2$ bits para calcular X

as:

Igual con las claves

push B
push C

mult
push A
add

pop G
push E
push F
mult
pop H
push D
push H

sub
pop H
push G
push H
div
pop I.

8. Sea un computador de cero direcciones (pila) cuyo juego de instrucciones contiene push, pop, add, sub, mul y div.

Escriba la rutina que calcula el producto escalar de dos vectores de tres componentes (a_1, a_2, a_3) y (b_1, b_2, b_3). Las componentes de los vectores están en memoria, y el resultado debe almacenarse también en memoria.

$$(a_1, a_2, a_3), (b_1, b_2, b_3) = \sum_{i=1}^3 a_i b_i$$

push a ₁ push b ₁ mult pop s ₁	push a ₂ push b ₂ mult pop s ₂	push a ₃ push b ₃ mult push s ₁ add push s ₂ add pop res
--	--	---

Sup. puedo acceder memoria?

9. Una calculadora programable con arquitectura de pila, una memoria de 256 bytes para datos de 32 bits e instrucciones, y una pila de 256 palabras de 32 bits para almacenar resultados intermedios, dispone de las siguientes instrucciones de los bytes:

PUSH X, POP X, IN X, OUT X, JMP X, JZ X y JN X

(donde X es una dirección de memoria) y de las siguientes instrucciones de un byte:

ADD, SUB, MUL, DIV y SQRT

Escriba un programa que pida al usuario tres números A, B y C, y saque como resultado las dos soluciones de la ecuación de 2º grado $Ax^2 + Bx + C = 0$. Si no hay solución real el programa terminará sin sacar ningún resultado.

Comprobaremos si $b^2 - 4ac \geq 0$ si > 0 → tercios valores

si = 0 → una sol

si < 0 → dos sol

256 bytes.

Datos: 32 bits

Zust. → 32 bits

Pila: 256 pal → 8 bits
→ 32 bits

Hay que saber de pila.

10. Cierta máquina de pila implementada en memoria dispone de las siguientes instrucciones:

FETCH X	push: $tope \leftarrow tope + 1; C(tope) \leftarrow C(X)$
STORE X	pop: $C(X) \leftarrow C(tope); tope \leftarrow tope - 1$
ADD	$C(tope - 1) \leftarrow C(tope) + C(tope - 1); tope \leftarrow tope - 1$
SUB	$C(tope - 1) \leftarrow C(tope) - C(tope - 1); tope \leftarrow tope - 1$
MPY	$C(tope - 1) \leftarrow C(tope) * C(tope - 1); tope \leftarrow tope - 1$
DIV	$C(tope - 1) \leftarrow C(tope) / C(tope - 1); tope \leftarrow tope - 1$
DUP	$C(tope + 1) \leftarrow C(tope); tope \leftarrow tope + 1$
EXCH	intercambiar $C(tope)$ y $C(tope - 1)$

donde $tope$ es la dirección de la cima de la pila y $C(posición)$ es el contenido de esa posición de memoria.

Escriba el programa para arquitectura de tipo pila más corto posible para evaluar

$$Z \leftarrow ((B^*C) - A)^2 / (B^*C)$$

$$Z = \frac{(B^*C - A)^2}{B^*C}$$

Consiste en programar si siguiendo la tabla de instrucciones.

11. Un ordenador con 16 registros direccionables (R_0, R_1, \dots, R_{15}) de 16 bits y una memoria de 64 KB tiene instrucciones cuyo tamaño es un número entero de bytes. El código de operación (que incluye el modo de direccionamiento) ocupa un byte. El destino de las instrucciones con dos operandos es el primer operando.

a) Para cada una de las instrucciones del siguiente programa, indique qué modos de direccionamiento se utilizan y dibuje un esquema (cajitas y flechas) que muestre cómo funcionan esos modos de direccionamiento (con direcciones y valores para este programa concreto y suponiendo ubicación en

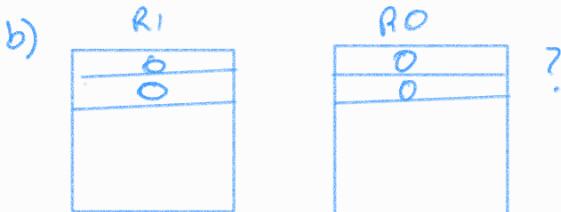
2/7

memoria little-endian):

```

1000: SUB R1, R1
1002: AND R0, R1
1004: MOV [++R1], R0
1006: XOR 0001h[R0], R1
100A: JNZ -CAh
100C: ADD [0002h], #0003h
    
```

b) ¿Cuáles son los contenidos de $R_0, R_1, M[1], M[2]$ y $M[3]$ cuando el programa finaliza para el caso de ubicación little-endian? ¿Y para big-endian?



$R_1, R_1 \rightarrow \text{registros}$

$R_0, R_1 \rightarrow \text{regs}$

[+R1]?

0001h [R0] → desplazamiento

- CAh → relativo a PC sub α_{100}

[0002h] inmediato

0003h?

little endian 00 00 00 todo 0 → big endian = little endian.

12. Un ordenador tiene las siguientes características:

- Microprocesador con 16 registros direccionables (r_0, r_1, \dots, r_{15}) de 32 bits cada uno.
- Memoria de 4 GB direccionable por bytes.
- El tamaño de palabra (tanto en el microprocesador como en memoria) es 32 bits.
- Utiliza la ubicación little-endian.
- El tamaño de las instrucciones es un número entero de bytes. → kbytes
- El código de operación de las instrucciones incluye los modos de direccionamiento y siempre ocupa un byte.
- Los desplazamientos son de 16 bits.
- El destino de las instrucciones con dos operandos es el primer operando.

a) Para cada una de las instrucciones del siguiente programa, indique qué modos de direccionamiento se utilizan y dibuje un esquema (cajitas y flechas) que muestre cómo funcionan esos modos de direccionamiento (con direcciones y valores para este programa concreto).

```

10000000: sub r1, r1
10000002: Bucle: and r0, r1
10000004: mov [r1++], r0
10000006: xor Tabla[r0], r1
1000000A: jnz Bucle
1000000D: add [00000002h], #00001000h
    
```

b) Muestre en una tabla cuáles son los contenidos de R_0, R_1 , y la primera palabra de memoria (la que se encuentra almacenada a partir de la dirección 0) desde el principio hasta que el programa finaliza.

→ Dir, reg → 4 bits

igual que arriba

- 13.** Un computador dispone de las 9 instrucciones adjuntas.

Para cada una de estas instrucciones y operandos indique su(s) modo(s) de direccionamiento y dibuje un esquema lo más didáctico, preciso y claro que pueda de su funcionamiento (se trata de explicar sólo con un dibujo, sin palabrería, lo que hace cada instrucción).

1.	ADD	Reg,#Dato	registos
2.	ADD	RegDst,RegSrc	registros
3.	LD	Reg,#Dato	reg
4.	LD	Reg,[Dir]	memoria reg
5.	LD	Reg,[Reg]	memoria reg
6.	MOVE	RegSrc,RegDst	reg
7.	ST	Reg,[Dir]	reg
8.	ST	Reg,[Reg]	reg + memoria
9.	Bcond	Desp	reg

- 14.** Un computador tiene 8 registros (R1, R2, ..., R8), de los cuales R1 actúa como contador de programa, y R2 como puntero de pila, y dispone solamente de las siguientes instrucciones:

ADD	Reg,#Dato
ADD	RegDst,RegSrc
LD	Reg,#Dato
LD	Reg,[Dir]
LD	Reg,[Reg]
MOVE	RegSrc,RegDst
ST	Reg,[Dir]
ST	Reg,[Reg]
Bcond	Desp

Las instrucciones que tienen direccionamiento inmediato o absoluto ocupan 2 palabras, y el resto una palabra. Los registros no se autoincrementan ni autodecrementan, excepto el contador de programa.

Programa, con este juego de instrucciones, el equivalente a las siguientes funciones:

- a) PUSH Reg y POP Reg
b) ADD [Dir], [Reg+]

Usar hacia arriba para programar estas instrucciones en ensamblador

3/7

- c) CALL [Dir] (llamada a la rutina que comienza en la dirección de memoria contenida en la dirección de memoria Dir)

El valor de los registros utilizados como almacenamiento temporal será el mismo que el que tuvieran antes de los fragmentos de programa correspondientes.

- 15.** Suponga un ordenador que tenga sólo las tres instrucciones de una dirección siguientes:

SUB X, que resta al acumulador A el contenido de la posición de memoria X.
STORE X, que almacena el contenido de A en la posición de memoria X.

JMPNEG X, que salta a la posición X si el contenido de A es negativo.

- a) Escriba un programa en ensamblador para esta máquina que implemente la operación

$$R \leftarrow M * N$$

mediante el siguiente algoritmo en pseudocódigo:

R ← 0
Repetir M veces la siguiente instrucción:
R ← R + N

Suponga lo siguiente:

- El acumulador A tiene inicialmente un valor arbitrario.

- M >= 0

- N >= 0

- El programa debe funcionar correctamente cuando M o N sean 0.

- Existe una posición de memoria llamada UNO que contiene la constante 1.

- Pueden usarse posiciones temporales de memoria y etiquetas.

- Al finalizar, M y N deben contener el mismo valor que al principio, y R contendrá el resultado.

b) Concurso: conseguir que el programa ejecute las mínimas instrucciones bajo los supuesto anteriores. Ganará el estudiante que diseñe el programa correcto que ejecute menos instrucciones para valores muy grandes de M.

c) Si le estuviera permitido implementar una instrucción más para este ordenador, ¿cuál escogería? Razone su respuesta escribiendo de nuevo el programa usando la nueva instrucción.

16. Consideré un ordenador hipotético con arquitectura de acumulador y un repertorio de instrucciones formado por sólo dos instrucciones de n bits. El primer bit especifica el código de operación, y los bits restantes direccionan una de las 2^n palabras de n bits de la memoria principal. Las dos instrucciones son:

SUB X Subtract. Restar al acumulador el contenido de la posición de memoria X, y memorizar el resultado en la posición X y en el acumulador.

RC X Branch if Carry. Saltar si acarreo (adeudo) en la última operación SUB, colocar la dirección X (el valor X, no el contenido de la posición X) en el acumulador de programa.

Una palabra de memoria puede contener una instrucción o un número binario en notación de complemento a dos. Demuestre que este repertorio de instrucciones es razonablemente completo especificando en ensamblador cómo se podrían programar las siguientes operaciones:

a)	CLR X	Clear. Poner a 0 el contenido de la posición X y el acumulador.	Pista: Utilizar SUB.
b)	LD X	Load. Cargar transferir el contenido de la posición de memoria X al acumulador.	Pista: Utilizar CLR, SUB, y una posición auxiliar.
c)	ST X	Store. Almacenar transferir el contenido del acumulador a la posición de memoria X.	Pista: Utilizar SUB y suponer que la posición de memoria CERO contiene 0. Nota: Después de esta instrucción se podría poner una instrucción CLR CERO para dejar CERO a 0.
d)	ADD X	Add. Sumar el contenido del acumulador con el contenido de la posición X. El resultado se almacena en la posición X y en el acumulador.	Pista: Utilizar ST, CLR CERO, SUB, y una posición auxiliar.
e)	SHL X	Shift Left. Desplazar la posición X hacia la izquierda. El resultado se almacena en la posición X y en el acumulador.	Pista: Utilizar LD y ADD.
f)	JMP X	Jump. Saltar incondicionalmente: colocar la dirección X en el contador de programa.	Pista: Utilizar CLR, SUB, JC, y suponer que la posición de memoria DOSN1 contiene 2^32.
g)	OR X	Or. Operación lógica OR entre el acumulador y el contenido de la posición X. El resultado se almacena en el acumulador.	Pista: Utilizar un bucle y utilizar desplazamientos y acarreos. Utilizar posiciones temporales y suponer que la posición de memoria UNO contiene 1. Consejo: Utilice pseudocódigo antes de empezar a escribir instrucciones.
h)	IN X	Input. Operación de entrada desde un puerto en el acumulador.	Pista: Suponer E/S mapeada en memoria.
	OUT X	Output. Operación de salida del acumulador en un puerto.	

operacion mult X que multiplica el contenido del acumulador por 8.

ubits: 1 octet + (u-1)bits → 2^u-1 palabras de u bits

Usar el repertorio, es sencillo

17. Dado el repertorio de instrucciones en ensamblador siguientes:

```

LOAD A, d ; Cargar en el acumulador el contenido de la posición de memoria d
LOAD A, [d] ; Cargar en el acumulador el contenido de la posición de memoria almacenada en la posición de memoria d
SUB A, d ; Restar del acumulador el contenido de la posición de memoria d
SUB A, [d] ; Restar del acumulador el contenido de la posición de memoria almacenada en la posición de memoria d
STORE d, A ; Almacenar en la posición de memoria d el contenido del acumulador
STORE [d], A ; Almacenar en la posición de memoria almacenada en la posición de memoria d el contenido del acumulador
JZ d ; Saltar a la posición de memoria d si el resultado de la última operación aritmética es cero
JZ [d] ; Saltar a la posición de memoria almacenada en la posición de memoria d si el resultado de la última operación aritmética es cero
JC d ; Saltar a la posición de memoria d si el resultado de la última operación aritmética provocó acarreo
JC [d] ; Saltar a la posición de memoria almacenada en la posición de memoria d si el resultado de la última operación aritmética provocó acarreo
JMP d ; Saltar a la posición de memoria d
JMP [d] ; Saltar a la posición de memoria almacenada en la posición de memoria d

```

- a) Escriba en ensamblador el programa para realizar la ordenación de menor a mayor de una lista de N bytes sin signo almacenados en memoria, supuestas las siguientes contenidos de memoria ya almacenados:

```

M[0] = 1
M[1] = N
M[2] = Posición de memoria inmediatamente siguiente al último elemento de la lista

```

Utilice el algoritmo siguiente:

```

for i:=N-1 downto 1 do
    for j:=i-1 downto 0 do
        if LISTA(i) < LISTA(j) then
            begin
                Temp := LISTA(i);
                LISTA(i) := LISTA(j);
                LISTA(j) := Temp;
            end;

```

- b) Realice una codificación del repertorio de instrucciones, teniendo en cuenta que hay 256 posiciones de memoria, cada una de un byte, y que sea lo más sencilla posible para el desarrollo de los siguientes apartados, aunque desperdicie espacio en memoria.

c) Diseñe el camino de datos (*datapath*) del ordenador que contenga sólo ese repertorio de instrucciones.

d) Diseñe la estructura de la unidad de control.

e) Escriba, en un lenguaje de alto nivel, el contenido de la memoria de control.

[] función como un puntero

{ intercambio = intercambiar el algoritmo con las instrucciones de arriba

8 bits para la welllona

Necesito n bits para el opcode, s bits para welllona y bits para direccionar el acar.

- 18.** a) Diseña una mínima exo-arquitectura de registros de uso general del tipo *load-store*, con un registro destino y dos registros fuente, que permita implementar el programa en C que se muestra a continuación. Se trata de decidir los registros, describir los modos de direccionamiento (con dibujos), y concebir un mínimo repertorio de instrucciones en lenguaje máquina (menos de 8). El programa ordena de menor a mayor una lista de N bytes sin signo almacenados en memoria.

```

for (i=N-1; i>=1; i--)
    for (j=i-1; j>=0; j--)
        if (LISTA[i] < LISTA[j])
            {
                temp = LISTA[i];
                LISTA[i] = LISTA[j];
                LISTA[j] = temp;
            }

```

- b) Escriba en ensamblador el programa completo utilizando esas instrucciones. Suponga que las constantes que necesita se encuentran ya almacenadas en posiciones de memoria.

Hacer un bucle for con mis instrucciones

a)
 dec reg
 swap reg1, reg2, temp (intercambia los registros usando temp como pivote).
 won sic, dest (dest=src)
 cuip reg1, reg2 (hace reg2-reg1 uod-flags)
 jl (saltasi es menor)
 jup (saltas)
 jge (salta si mayor o igual)
 jg (saltasi mayor)

19. Traduzca a ensamblador (el repertorio de instrucciones se encuentra en la Tabla 1) la siguiente función Pascal:

```

function min(i,j,k: integer): integer;
var m: integer;
begin
  if i < j then m:=i else m:=j;
  if k<m then m:=k;
  min := m;
end;

```

El tipo **integer** ocupa una palabra de 16 bits. Cada posición direccionable de memoria es una palabra de 16 bits. Los parámetros de la función están ya almacenados en la pila, antes de hacer la llamada, en direcciones superiores al contador de programa, siguiendo el mismo orden de los parámetros de la llamada, es decir, primero i, luego j, luego k y más abajo el PC. La función almacena la variable local m en la pila debajo del contador de programa. El resultado de la función debe quedar almacenado en el acumulado (AC). No es responsabilidad de la función quitar de la pila los parámetros de llamada.

Tabla 1. Repertorio de instrucciones máquina para el problema 5. En las instrucciones locales, X es un número de 12 bits en complemento a 2 (de -2048 a 2047); en las restantes X es un número positivo de 12 bits (0 a 4095).

Codificación	Ensambledor	Instrucción	Significado
0000xxxxxxxxxxxx	LODD X	Carga directa	AC := MX
0001xxxxxxxxxxxx	STOD X	Almacenamiento directo	MX := AC
0010xxxxxxxxxxxx	ADDD X	Suma directa	AC := AC + MX
0011xxxxxxxxxxxx	SUBD X	Resta indirecta	AC := AC - MX
0100xxxxxxxxxxxx	JPOS X	Salto si positivo	if AC > 0 then PC := X
0101xxxxxxxxxxxx	JZER X	Salto si cero	if AC = 0 then PC := X
0110xxxxxxxxxxxx	JUMP X	Salto incondicional	PC := X
0111xxxxxxxxxxxx	LOCO X	Carga de constante	AC := X (0 ≤ X ≤ 4095)
1000xxxxxxxxxxxx	LODL X	Carga local	AC := M(SP + X)
1001xxxxxxxxxxxx	STOL X	Almacenamiento local	M(SP + X) := AC
1010xxxxxxxxxxxx	ADDL X	Suma local	AC := AC + M(SP + X)
1011xxxxxxxxxxxx	SUBL X	Resta local	AC := AC - M(SP + X)
1100xxxxxxxxxxxx	JNEG X	Salto si negativo	if AC < 0 then PC := X
1101xxxxxxxxxxxx	JNZE X	Salto si no cero	if AC ≠ 0 then PC := X
1110xxxxxxxxxxxx	CALL X	Llamada a subrutina	SP := SP - 1; M(SP) := PC; PC := X
1111000000000000	PUSH	PUSH del acumulador en la pila	SP := SP - 1; M(SP) := AC
1111010000000000	POP	POP de la pila en el acumulador	AC := M(SP); SP := SP + 1
1111100000000000	RETN	Retorno de subrutina	PC := M(SP); SP := SP + 1
1111110000000000	SWAP	Intercambio de AC y SP	TMP := AC; AC := SP; SP := TMP

Tabla 2. Repertorio de instrucciones máquina para el problema 6. En las instrucciones locales, X es un n.º de 12 bits en complemento a 2 (de -2048 a 2047); en las restantes X es un n.º positivo de 12 bits (0 a 4095).

Codificación	Ensambledor	Instrucción int j, int k)	Significado
0000xxxxxxxxxxxx	JPOS X	Salto si positivo	if AC > 0 then PC := X
0001xxxxxxxxxxxx	JZER X	Salto si cero	if AC = 0 then PC := X
0010xxxxxxxxxxxx	JUMP X	Salto incondicional	PC := X
0011xxxxxxxxxxxx	LOGO X	Carga de constante 1 else	AC := X (0 ≤ X ≤ 4095)
0100xxxxxxxxxxxx	LODL X	Carga local / criterio m = X	AC := M(SP + X)
0101xxxxxxxxxxxx	STOL X	Almacenamiento local	M(SP + X) := AC
0110xxxxxxxxxxxx	ADDL X	Suma local	AC := AC + M(SP + X)
0111xxxxxxxxxxxx	SUBL X	Resta local	AC := AC - M(SP + X)
1000xxxxxxxxxxxx	JNEG X	Salto si negativo	if AC < 0 then PC := X
1001xxxxxxxxxxxx	JNZE X	Salto si no cero	if AC ≠ 0 then PC := X
1010xxxxxxxxxxxx	CALL X	Llamada a subrutina	SP := SP - 1; M(SP) := PC; PC := X
1011-----	PUSH	PUSH del acumulador en la pila	SP := SP - 1; M(SP) := AC
1100-----	POP	POP de la pila en el acumulador	AC := M(SP); SP := SP + 1
1101-----	RETN	Retorno de subrutina	PC := M(SP); SP := SP + 1

Usar lo deabajo para sacar la función

20. Traduzca a ensamblador (el repertorio de instrucciones se encuentra en la Tabla 2) la siguiente función escrita en C:

El tipo **int** ocupa una palabra de 16 bits. Cada posición direccionable de memoria es una palabra de 16 bits. Los parámetros de la función están ya almacenados en la pila, antes de hacer la llamada, en direcciones superiores al contador de programa, siguiendo el orden inverso de los parámetros de la llamada, es decir, primero k, luego j, luego i. Más abajo en la pila está almacenado el PC. La función almacena la variable local m en la pila, debajo del contador de programa. El resultado de la función debe quedar almacenado en el acumulador (AC). No es responsabilidad de la función quitar de la pila los parámetros de llamada.

Usar la tabla anterior para programar.

21. Sea un computador de arquitectura memoria-memoria, con dos operandos explícitos en las instrucciones aritmético-lógicas (el primero es el destino). Realice una subrutina reubicable que calcule la varianza y la media de una serie unidimensional de datos. Los parámetros de entrada y salida se pasan a través de la pila del sistema. Los de entrada serán el número de datos de la serie y la dirección de comienzo de la misma. Los de salida la varianza y la media. Puede suponer que dispone de cualquiera de las instrucciones máquina habituales, siempre que respete el modelo memoria-memoria.

$$\text{Varianza} = \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N}$$

Si es ensamblador normal → easy.

Es un bue, undiv y mucha paeria

22. Escriba un programa en lenguaje ensamblador para un microprocesador de 16 bits (con varios registros de uso general de 16 bits) que genere una secuencia de 32 números aleatorios con una distribución gaussiana. Un método para obtener dicha secuencia se basa en el teorema del límite central según el cual un conjunto de valores medios de secuencias de números sin correlacionar tiene una distribución gaussiana de valores. Suponga que tiene una tabla de 16 Kbytes compuesta de números de 16 bits con números sin correlacionar. Tome 256 datos de la tabla y súmelos (la suma de números de 16 bits puede producir un número de más de 16 bits). Divida por 256. De esta forma obtendrá el valor medio de esos 256 números. El número obtenido tendrá un valor aleatorio que sigue una distribución gaussiana. Genere así otra tabla, de 32 palabras de 16 bits ($8K / 256 = 32$), con los valores obtenidos a partir de la tabla original.

→ la función genera → cuadra de hora y obtiene 32 → me explican metodo para hacer numeros pseudoaleatorios → me explico metodo para hacer numeros pseudoaleatorios que consiste en trazar medidas anteriores de tablas de tabla gaussiana.

