Facultad de Informática
Universidad Complutense de Madrid

PROBLEMAS DE FUNDAMENTOS DE COMPUTADORES II
TEMA 3

Problemas básicos:

1. Escriba un programa en lenguaje ensamblador del RISC-V que implemente el siguiente bloque de código. Use la sección **`.data`** para asignar el valor inicial a las variables.

```
int x = 10, y = 5;

if (x >= y) {
  x = x + 2;
  y = y - 2;
}
```

2. Escriba un programa en lenguaje ensamblador del RISC-V que implemente el siguiente bloque de código. Use la sección **`.data`** para asignar el valor inicial a las variables.

```
int x = 5, y = 10;

if (x >= y) {
  x = x + 2;
  y = y + 2;
}
else {
      x = x - 2;
      y = y - 2;
}
```

3. Escriba un programa en lenguaje ensamblador del RISC-V que implemente el siguiente bloque de código. Use la sección **`.bss`** para reservar espacio en memoria para las variables.

```
int a, b;

a = 81;
b = 18;
do {
  a = a - b;
} while (a > 0);
```

4. Escriba un programa en lenguaje ensamblador del RISC-V que implemente el siguiente bloque de código. Use la sección **`.bss`** para reservar espacio en memoria para las variables.

```
int n, fprev, f, i;

n = 5;
fprev = 1;
f = 1;
i = 2;
while (i <= n) {
  faux = f;
  f = f + fprev;
  fprev = faux;
  i = I + 1;
```

```
}
```

5. Escriba un programa en lenguaje ensamblador del RISC-V que implemente el siguiente bloque de código. Use la sección **`.data`** para asignar el valor inicial a las variables f y n, y la sección **`.bss`** para reservar espacio de memoria para la variable i.

```
int f = 2, n = 5;
int i;

for (i = 2; i <= n; i++)
  f = f + f;
```

6. El siguiente programa calcula el máximo común divisor de dos números a y b según el algoritmo de restas de Euclides. Tradúzcalo a ensamblador del RISC-V. Use la sección **`.data`** para asignar el valor inicial a las variables a y b, y la sección **`.bss`** para reservar espacio de memoria para la variable mcd.

```
int a=5, b=15, mcd;
while (a ≠ b) {
  if (a > b)
    a = a - b;
     else
       b = b - a;
}
mcd = a;
```

7. Escriba un programa en lenguaje ensamblador del RISC-V que implemente el siguiente bloque de código:

```
int f, g=2, h=3;
int B[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

f = g + h + B[4];
```

Problemas adicionales:

8. El siguiente bloque de código suma los componentes de un vector de 10 componentes. Tradúzcalo a ensamblador del RISC-V.

```
#define N 10

int V[N] = {12, 1, -2, 15, -8, 4, -31, 8, 8, 25};

for (i = 0; i < N; i++)
  V[i] = V[i] + 1;
```

9. El siguiente bloque de código cuenta el número de componentes mayores de 0 que contiene de un vector de 6 componentes. Tradúzcalo a ensamblador del RISC-V.

```
#define N 6

int V[N] = {14, 1, -2, 7, -8, 4};
int count = 0;

for (i = 0; i < N; i++) {
  if (V[i] > 0)
    count = count + 1;
}
```

10. El siguiente programa calcula la sucesión de Fibonacci y la almacena en un vector V de longitud arbitraria N. Tradúzcalo a ensamblador del RISC-V.

```
#define N 12

int V[N];

V[0] = 0;
V[1] = 1;
for (i = 0; i < N-2; i++)
  V[i+2] = V[i+1] + V[i];
```

11. El siguiente programa, dado un vector A de 12 componentes, genera otro vector, B, tal que B sólo contiene las componentes de A que son números pares mayores que cero. Por ejemplo, si A = (0,1,2,7,-8,4,5,12,11,-2,6,3), entonces B = (2,4,12,6). Tradúzcalo a ensamblador del RISC-V.

```
#define N 12

int A[N] = {0, 1, 2, 7, -8, 4, 5, 12, 11, -2, 6, 3};
int B[N];
int countB = 0;

j = 0;
for (i = 0; i < N; i++) {
  if (A[i] > 0 && A[i] es par) {
    B[j] = A[i];
       j++;
  }
}
countB = j;
```

12. Dados dos vectores A y B de 10 componentes cada uno, se desea construir otro vector, C, tal que: C(i) = |A(i) + B(9-i)| con i = 0, 1, ... 9. Escriba un programa que efectué este cálculo y tradúzcalo a ensamblador del RISC-V.

13. Traduzca el siguiente programa a ensamblador del RISC-V sabiendo que la orden `swap(a,b)` intercambia los valores de las variables a y b.

```
int a = 13, b = 16;

while (a > 10) {
  a=a-1;
  b=b+2;
}
if (a < b)
  swap(a, b);
else
  b = a - 1;
```

14. Escriba un programa en ensamblador del RISC-V que llame a una subrutina swap encargada de intercambiar el contenido de dos posiciones de memoria. La subrutina recibirá como parámetros de entrada las posiciones de memoria correspondiente a las variables a y b y deberá preservar el contenido de todos los registros que obligue el estándar de llamadas estudiado en clase.

15. Escriba un programa para el ensamblador del RISC-V que cuente el número de 0's de un vector de longitud arbitraria. Emplee para ello una subrutina llamada cuenta0s que reciba como parámetros de entrada toda la información necesaria para llevar a cabo la tarea.

16. Escriba un programa en ensamblador del RISC-V que implemente una variante del algoritmo de ordenación de la burbuja. Esta variante ordena los elementos de un vector de menor a mayor recorriendo repetidas veces el vector, intercambiado posiciones

sucesivas si V(i)>V(i+1), hasta que no se realiza ningún cambio.

```
do {
  swapped = false
  for (i = 0; i <= N-2; i++){
  if (V[i] > V[i+1]){
    swap( V[i], V[i+1] )
    swapped = true
    }
} while swapped
```

17. Escriba un programa que calcule el factorial de un número no negativo, n, por medio de un bucle que realiza una secuencia iterativa de multiplicaciones. Tradúzcalo a ensamblador del RISC-V.

Problemas de examen:

18. (Septiembre 2015) Dados dos puntos P1(x1, y1) y P2(x2, y2), la distancia de Chebyshev entre ellos puede calcularse con el siguiente algoritmo:

```
int chebyshev(int x1, int y1, int x2, int y2)
{
  int d1, d2;
  d1 = abs(x1 - x2)
  d2 = abs(y1 - y2)
  if (d2 > d1)
    d1 = d2;
  return d1;
}
```

a) Codifique en ensamblador del RISC-V la subrutina chebyshev(x1, x2, y1, y2) que recibe las coordenadas de dos puntos P1 y P2 y devuelve la distancia de Chebyshev que los separa. En su cuerpo, invocará a la subrutina valor absoluto.

b) Codifique también el programa siguiente que almacena en un vector D la distancia de Chebyshev desde un punto P a todos los puntos de un vector V de N puntos, donde P, V y D serían variables globales. El vector V tendrá 2N enteros de forma que el i-ésimo punto tendrá coordenadas (x, y) = (V[2*i], V[2*i + 1])

```
#define N, ...

int Px, Py; //coordenadas x e y del punto P
int V[2N];  //Vector con N puntos V=[x0,y0,x1,y1,...]
int D[N];   //Vector de N distancias

void main(void)
{
   int i;
   for (i = 0; i < N; i++)
     D[i] = chebyshev(Px, Py, V[2*i], V[2*i + 1]);
}
```

19. (Junio 2016) Dado un vector A de 3*N componentes, se desea obtener un nuevo vector B de N componentes donde cada componente de B es la suma módulo 32 de una tripleta de elementos consecutivos de A. Es decir:

B[0] = (A[0]+A[1]+A[2]) mod 32, B[1] = (A[3]+A[4]+A[5]) mod 32, etc

Se pide:

a) Escribir un programa en lenguaje ensamblador del RISC-V que implemente el

cálculo descrito, de acuerdo al siguiente código C equivalente

```
#define N 4

int A[3*N] = {una lista de 3*N valores};
int B[N];
int i, j = 0;

void main(void)
{
  for (i = 0; i < N; i++){
    B[i] = sum_mod_32(A, j, 3);
    j = j+3
  }
}
```

Donde `sum_mod_32(V, p, m)` devuelve como resultado la suma módulo 32 de m elementos consecutivos del vector V tomados a partir de la posición p.

b) Escribir el código ensamblador de la subrutina sum_mod_32, de acuerdo al siguiente código C equivalente:

```
sum_mod_32(int A[], int j, int len)
{
  int i, sum=0;
  for (i = 0; i < len; i++)
    sum = sum + A[j+i];
  sum = mod_power_of_2(sum, 5);
  return sum;
}
```

Donde `mod_power_of_2(num, exp)`, siendo num un entero positivo y exp un entero mayor que 0 y menor que 32, devuelve como resultado el valor de (num mod 2^exp). Por ejemplo, si invocamos a la función como: `mod_power_of_2(34, 5)`, la función devolvería como salida: ((34) mod (2^5))=(34 mod 32)=2.

20. (Septiembre 2016) Dado un vector A de N enteros de 32 bits sin signo, se desea calcular su Código de Redundancia Cíclico o CRC siguiendo el algoritmo de Fletcher. Dicho algoritmo genera como salida dos enteros sin signo que servirán para comprobar la integridad de los datos. El algoritmo de Fletcher de 64 bits se puede implementar a partir del siguiente código:

```
void Fletcher64(
      unsigned int data[], int length, unsigned int crc[] )
{
  unsigned int sum1 = 0;
  unsigned int sum2 = 0;
  int index;
  for (index = 0; index < length; index++) {
    sum1 = sum_mod64(sum1, data[index]);
    sum2 = sum_mod64(sum1, sum2);
  }
  crc[0] = sum1;
  crc[1] = sum2;
}
```

Donde el primer argumento es el vector de datos, el segundo es la longitud del vector y el tercero es el vector que contiene el resultado (es decir, los dos enteros sin signo que componen el CRC). Se supone que la rutina `unsigned int sum_mod64(unsigned`

`int A, unsigned int B)` está ya implementada.

Se pide:

a) Escribir el código ensamblador del RISC-V de la rutina Fletcher64.
b) Escribir un programa en lenguaje ensamblador del RISC-V que, invocando a la rutina Fletcher64, calcule el CRC de los siguientes vectores:

V={0x12340000, 0x00005678}

W={0xAB000000, 0x00CD0000, 0x0000EF00, 0x00000011}

Debe llamar a la rutina Fletcher64 dos veces. La primera llamada retornará el CRC de V y la segunda el CRC de W.

21. (Junio 2013). Supongamos que definimos que un número natural es "bonito" si es menor que cien mil y además su valor puede obtenerse como una suma de números naturales de la forma 1+2+3+4+5+…

Se pide:

a) Escribir un programa en lenguaje ensamblador del RISC-V tal que dado un número natural N decida si es o no bonito. El programa escribirá en la variable B un 1 si el número es bonito y un 0 en caso contrario.
b) Convertir el código anterior en una subrutina que reciba como entrada un número natural N y devuelva como salida un 1 si el número N es bonito y un 0 si no lo es. Escribir un programa en lenguaje ensamblador del RISC-V que llame a la subrutina, y tal que dado un vector A de M números naturales sea capaz de hallar cuántos números bonitos hay en el vector. El programa debe almacenar la cantidad de números bonitos hallada en la variable "cuenta_bonitos".

22. (Junio 2014) Dado un vector V de N componentes se dice que es Melchoriforme si posee al menos un elemento Rubio. Un elemento V[i] es Rubio si satisface la siguiente expresión:

$$\sum_{j=0}^{N-1} \mathrm{v[j]} = 2 * \mathrm{v[i]}$$

Se pide:

c) Una subrutina SumaVector(V, N) que sume los N elementos del vector V, respetando el convenio de llamadas a subrutinas visto en clase.
d) Un programa que dado un vector V y su dimensión N decida si es Melchoriforme, utilizando la subrutina SumaVector.

23. (Septiembre 2014) Un vector V de N números naturales es noeliano si es una secuencia monótona creciente y sus elementos suman en total 45. Por ejemplo: {0,1,2,3,4,5,6,7,8,9} es noeliano porque 0≤1≤2≤3≤4≤5≤6≤7≤8≤9 y 1+2+3+4+5+6+7+8+9=45. También {3,5,5,7,10,15} lo es, ya que 3≤5≤5≤7≤10≤15 y 3+5+5+7+10+15=45.

Se pide:

a) Escribir en ensamblador de RISC-V una subrutina Sum45(A, N) que reciba la dirección de comienzo de un vector A como primer parámetro, el número N de elementos del vector como segundo parámetro y devuelva 1 si su suma es 45 y 0 en otro caso. La subrutina debe programarse de acuerdo con el estándar de llamadas a subrutinas que hemos estudiado en clase.

b) Escribir un programa RISC-V que, utilizando la subrutina anterior, determine si un vector de entrada es noeliano o no.