Facultad de Informática
Universidad Complutense de Madrid

INTRODUCTION TO COMPUTERS II
MODULE 3

Basic problems:

1. Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of the variables.

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

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

2. Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of the variables.

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

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

3. Write a RISC-V assembly program that implements the following code. Use the **`.bss`** section to reserve memory space for the variables.

```
int a, b;

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

4. Write a RISC-V assembly program that implements the following code. Use the **`.bss`** section to reserve memory space for the variables.

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

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

5. Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of variables f and n, and the **`.bss`** section to reserve memory space for variable i.

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

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

6. The following program calculates the greatest common divisor of two numbers a and b according to the Euclidean algorithm. Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of variables a and b, and the **`.bss`** section to reserve memory space for variable gcd.

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

7. Write a RISC-V assembly program that implements the following code:

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

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

Additional problems:

8. The following code increments the components of a vector with 10 elements. Translate it into RISC-V assembly code.

```
#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. The following code counts the number of components greater than 0 within a vector with 6 elements. Translate it into RISC-V assembly code.

```
#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. The following code calculates the Fibonacci sequence and stores it in a vector V with size N. Translate it into RISC-V assembly code.

```
#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. The following program, given a vector A with 12 elements, generates another vector B such that B only contains the elements of A that are even numbers greater than 0. For example, if A = (0,1,2,7,-8,4,5,12,11,-2,6,3), then B = (2,4,12,6). Translate it into RISC-V assembly code.

```
#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] is even) {
    B[j] = A[i];
       j++;
  }
}
countB = j;
```

12. Given two vectors A and B with 10 elements each, implement another vector, C, such that: C(i) = |A(i) + B(9-i)| with i = 0, 1, ... 9. Write a RISC-V assembly program that perform this calculation.

13. Translate the following program into RISC-V assembly code, considering that `swap(a,b)` swaps the values of variables a and 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. Write a RISC-V assembly program that calls a `swap` function that swaps the content of two memory positions. This function will receive the memory addresses of variables a and b as input parameters and will preserve the content of the appropriate registers according to the RISC-V function call convention.

15. Write a RISC-V assembly program to count the number of zeros in a vector with an arbitrary length. Use a function called `count0s` that receives all the needed information as input parameters.

16. Write a RISC-V assembly program to implement a variant of the bubble sort algorithm. This variant sorts the elements of the vector according to the following code.

```
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. Write a RISC-V assembly program to calculate the factorial of a non-negative number, n, using a loop that performs an iterative sequence of multiplications.

Exam problems:

18. (September 2015) Given two points P1(x1, y1) and P2(x2, y2), their Chebyshev distance can be calculated with the following algorithm:

```
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) Write a RISC-V assembly function, `chebyshev(x1,x2,y1,y2)`, which will receive the coordinates of two points P1 and P2 and will return their Chebyshev distance. This function will call another function that calculates the absolute value of a given number.

b) Translate the following program into RISC-V assembly code, which stores (into a vector D) the Chebyshev distances of a point P to each of the points within a vector V with N elements. P, V y D will be global variables. Vector V will contain 2N integers such that the i-th point will have coordinates (x, y) = (V[2*i], V[2*i + 1])

```
#define N, ...

int Px, Py; // x , y coordinates of point P
int V[2N];  //Vector with N points V=[x0,y0,x1,y1,...]
int D[N];   //Vector with N distances

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

19. (June 2016) Given a vector A with 3*N elements, we want to obtain a new vector B with N elements so that each element of B is the mod-32 addition of three consecutive elements of A:

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

a) Write a RISC-V assembly program to implement the described calculation, according to the following C code:

```
#define N 4

int A[3*N] = {a list with 3*N values};
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
  }
}
```

Where `sum_mod_32(V, p, m)` returns the mod-32 addition of m consecutive elements of vector V, starting at position p.

b) Write the RISC-V assembly code of the `sum_mod_32` function, according to the following C code:

```
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;
}
```

Where `mod_power_of_2(num, exp)`, returns the value of (num mod (2^exp)), being num a positive integer and exp and integer greater than 0 and less than 32. For example, if the function is called as `mod_power_of_2(34, 5)`, it will return: (34 mod (2^5)) = (34 mod 32) = 2.

20. (September 2016) Given a vector A with N 32-bit unsigned elements, we want to calculate its Cyclic Redundancy Check (CRC) according to the Fletcher's checksum. This algorithm outputs two unsigned integers in order to check data integrity. The 64-bit version of this algorithm (Fletcher64) can be implemented with the following code:

```
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;
}
```

Where the first argument is the data vector, the second one is the length vector and the third one is the vector that stores the result (i.e., the two unsigned integers that form the CRC). Assume that function `unsigned int sum_mod64(unsigned int A, unsigned int B)` is already implemented.

a) Write the RISC-V assembly code that implements the `Fletcher64` function.
b) Write a RISC-V assembly program that calls the `Fletcher64` function and calculates the CRC of the following vectors:

V={0x12340000, 0x00005678}

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

The `Fletcher64` function must be called twice. The first call will return the CRC of V and the second one the CRC of W.

21. (June 2013). Assume that we define a number as "nice" if its value is less than one hundred thousand and it can be obtained as the sum of natural numbers as 1+2+3+4+5+…

a) Write a RISC-V assembly program that receives a natural number N and decides whether it is nice or not. The program will write a 1 in variable B if N is nice and a 0 otherwise.
b) Convert the previous program in a function that receives a natural number N and returns a 1 if N is nice and a 0 otherwise. Write a RISC-V assembly program that calls this function. This program will receive a vector A with M natural numbers and will find out how many nice numbers there are in such vector. The program will store the amount of nice numbers in a `count_nice` variable.

22. (June 2014) Given a vector V with N elements, we define it as Melchioriform if it has at least one Blonde element. An element V[i] is Blonde if it meets the following expression:

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

c) Write a RISC-V assembly function, `AddVector(V, N)`, which adds the N elements of vector V. This function must follow the RISC-V function call convention.
d) Write a RISC-V assembly program that, given a vector V and its size N, decides whether V is Melchioriform or not, using the `AddVector` function.

23. (September 2014) A vector V with N natural elements is Noelian if it is a monotonic increasing sequence and its elements add up to 45. For example: {0,1,2,3,4,5,6,7,8,9} is Noelian because 0≤1≤2≤3≤4≤5≤6≤7≤8≤9 and 1+2+3+4+5+6+7+8+9=45. Also {3,5,5,7,10,15} is Noelian, because 3≤5≤5≤7≤10≤15 and 3+5+5+7+10+15=45.

a) Write a RISC-V assembly function, `Sum45(A, N)`, which receives the initial address of a vector A as its first parameter, the number of elements N as its second parameter and returns 1 if its sum is 45 (0 otherwise). This function must be programmed following the RISC-V function call convention.
b) Write a RISC-V assembly program that, using the previous function, determines whether an input vector is Noelian or not.