Pregunta 1

Parte a.- (2 puntos) Programe la función *concat_bits* declarada como:

```
typedef unsigned int uint; // enteros sin signo
uint concat_bits(uint x, int n, uint y, int m);
```

Si los bits de x son $x_{31}\ x_{30}\ ...\ x_1\ x_0$ en donde x_0 es el bit menos significativo y los de y son $y_{31}\ y_{30}\ ...\ y_1\ y_0$, la función *concat_bits* debe retornar $0\ \dots\ 0\ x_{n-1}\ ...\ x_0\ y_{m-1}\ ...\ y_0$. Considere que n y m son menores que 32 y $n+m<=32$. Ejemplos de uso:

```
uint z1= concat_bits(0b101,3,0b11001, 5);   // z1=0b10111001
uint z2= concat_bits(0b11011, 3, 0b10101, 2); // z2=0b01101
```

Restricciones: No use los operadores de multiplicación, división o módulo (* / %). Use eficientemente + y -, y los operadores de bits.

Parte b.- (4 puntos) Programe la función *convertir* declarada como:

*void convertir(unsigned int x, int b, char *s, int n);*

Esta función recibe el entero sin signo x y lo convierte a base b (con $2 \le b \le 10$) dejando el resultado en el string s de largo n alineado a la derecha. *Ayuda*: divida x por b reiteradamente hasta obtener 0; en cada iteración $x\ \%\ b$ le entregará un dígito que debe colocar en s. Ejemplos:

```
char s1[6]; convertir(13, 2, s1, 5);  // s1 es " 1101"
char s2[7]; convertir(134, 8, s2, 6); // s2 es "  206"
```

Restricciones: No puede invocar a otras funciones dentro de su código. No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros.

Pregunta 2

La función *menores* del cuadro de la derecha mueve al inicio del arreglo a todos los elementos de a que son menores que x y retorna la cantidad de elementos menores que x. Este es un ejemplo de uso:

```
double a[5]={ 5.2, 4.1, 7.3,
              8.2, 1.6 };
int k= menores(a, 5, 5.2);
       // k es 2, a[0] es 4.1 y a[1] es 1.6
```

```
int menores(double a[],
      int n, double x) {
  double *p= a;
  double *q= a;
  for (int i=0;i<n;i++){
    double y= *p++;
    if (y<x)
      *q++= y;
  }
  return q-a;
}
```

Programe la función *menoresPar* con los mismos parámetros y valor retornado que *menores*, pero en paralelo para un computador dual core.

Métodología obligatoria: Use *fork* para crear un proceso hijo que se comunica con el proceso padre por medio de un pipe. Sea $h=(n+1)/2$. En el proceso hijo llame a *menores* para determinar los menores que x desde $a[h]$ hasta $a[n-1]$. Considere que encuentra kh elementos menores que x. Envíe por el pipe kh y los elementos menores encontrados. En el proceso padre llame a *menores* para determinar los menores que x desde $a[0]$ hasta $a[h-1]$. Considere que encuentra kp elementos menores y por lo tanto quedan almacenados en $a[0]$ hasta $a[kp-1]$. Lea kh del pipe y reciba los menores encontrados por el hijo a partir de $a[kp]$ (use la función *leer*). La cantidad de menores que x en todo el arreglo a es $kh+kp$.

Pregunta 3

I. (2 puntos) Construya un circuito que dados a, b y c de 32 bits calcule $max(a+b,c)$.

II. (4 puntos) El programa en assembler Risc-V de la derecha es el resultado de compilar la función *incognito*. **Programe** la función *equivalente* a *incognito* en C sin usar la instrucción **goto** de C. Preocúpese de *reproducir* en C todos los aspectos de la función original en assembler, en particular el valor retornado. El encabezado de *incognito* es:

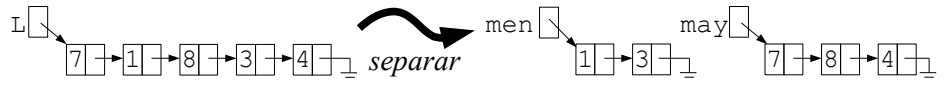
```
int *incognito(int a[]);
```

```
incognito:
    lw   a2,0(a0)
    addi a0,a0,4
    mv   a1,a0
    j    .L2
.L1:
    lw   a3,0(a1)
    addi a1,a1,4
    beq  a2,a3,.L2
    mv   a2,a3
    sw   a3,0(a0)
    addi a0,a0,4
.L2:
    bne  a2,zero,.L1
    ret
```

Pregunta 4

Programe la función *separar* con el encabezado del cuadro de la derecha. Esta función recibe en L una lista simplemente enlazada desordenada en donde cada nodo almacena un entero. Ud. debe desarmar la lista L de tal forma que en **pmen* queden los nodos que almacenan enteros menores que z y en **pmay* queden los nodos que almacenan enteros mayores o iguales que z. En el cuadro también se muestra un ejemplo de uso, en donde la lista L ha sido creada con 5 nodos que almacenan 7, 1, 8, 3 y 4. La figura muestra a la izquierda la lista L original y a la derecha las 2 listas resultantes de invocar separar.

```
typedef struct nodo {
  int x;
  struct nodo *prox;
} Nodo;
void separar(Nodo *L, int z,
  Nodo **pmen, Nodo **pmay);
Nodo *L= ...; // 7 1 8 3 4
Nodo *men, *may;
separar(L, 4, &men, &may);
```



Restricciones: No puede usar *malloc*. Debe reutilizar los nodos que recibe en la lista L. Los nodos en **pmen* y **pmay* deben seguir el mismo orden en que aparecían en L. Por claridad Ud. **debe usar recursividad**.