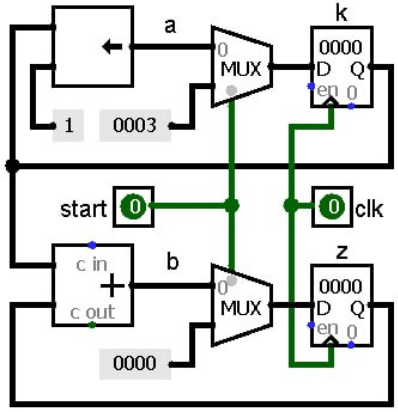
Pregunta 1

I. Considere el circuito de la figura. **Complete** la tabla de al lado con los valores que toma cada señal hasta el ciclo 6 del reloj. Use notación decimal.



ciclo	*start*	*k*	*z*	*a*	*b*
1	0	0	0	0	0
2	1	0	0	0	0
3	0				
4	0				
5	0				
6	0				

II. Traduzca la función *g* del cuadro de la derecha a assembler Risc-V.

```
int menores(int a[],
            int n, int x) {
  int *d=a;
  int *u= &a[n];
  for (int *p=a; p<u; p++){
    int y= *p;
    if (y<x)
      *d++= y;
  }
  return d-a;
}
```

Pregunta 2

Programe la función:

*void elim(char *s, int n);*

Esta función ve el string *s* como varias secuencias de n caracteres. En cada secuencia elimina el último caracter de esa secuencia. Ejemplo:

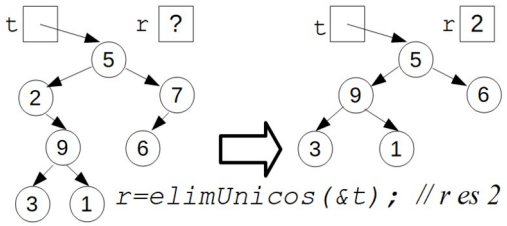
```
char s[]= "123456789";
elim(s,3); // s es "124578"
elim(s,4); // s es "12478"
```

Restricciones: Ud. no puede usar el operador de subindicación [ ], ni su equivalente **(p+i)*. Use aritmética de punteros como *p++* o *p+i*.

Pregunta 3

Programe la función *elimUnicos* con el encabezado del cuadro de la derecha. Esta función recibe en **pa* un árbol binario (no ordenado) y elimina todos los nodos que tengan un solo hijo. Debe retornar el número de nodos que fueron eliminados. En el ejemplo de la figura de la derecha el tipo de *t* es *Nodo** y de *r* es *int*. El nodo con etiqueta 2 es eliminado porque su único hijo es el nodo de etiqueta 9. Por la misma razón se elimina el nodo de etiqueta 7. Los nodos 3, 1 y 6 no se eliminan porque no tienen hijos.

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
} Nodo;
int elimUnicos(Nodo **pa);
```



Metodología obligatoria: Sea *a= *pa*. El caso en que *a* es el árbol vacío es trivial. Elimine recursivamente los nodos que tienen un solo hijo del subárbol izquierdo y del subárbol derecho. Si ahora *a* tiene un solo nodo hijo, haga que **pa* apunte hacia ese único nodo y libere la memoria ocupada por el nodo *a*. Calcule Ud. eficientemente el número de nodos eliminados.

Pregunta 4

La función *sumar* del cuadro de la derecha suma el vector *b* al vector *a*. Ambos vectores contienen *n* elementos. **Programe** la función *sumarPar* con la misma funcionalidad de *sumar*, pero realizando la suma en paralelo para un computador dual core. El encabezado es:

```
void sumar(double *a,
           double *b,
           int n) {
  for (int i=0; i<n; i++)
    a[i] += b[i];
}
```

```
void sumarPar(double *a, double *b, int n);
```

Metodología obligatoria: Invoque *fork* para crear un nuevo proceso. El hijo suma una mitad de los vectores y el padre suma la otra mitad. Use un pipe para que el hijo envíe al padre sus resultados. **¡Revise que haya paralelismo!**