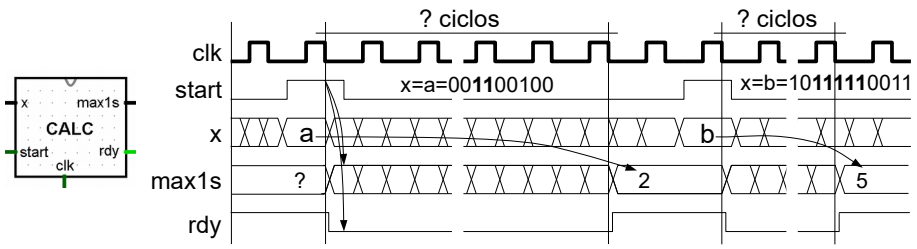
## Pregunta 1




Use diseño modular para implementar el circuito *CALC* de la figura. Este circuito entrega en la salida *max1s* el máximo número de bits consecutivos en 1 encontrados en la entrada *x*, corresponiente a un entero de 32 bits sin signo. Por ejemplo si x=00...00**11**00100, *CALC* debe entregar 2 y si x=00..0010**1111**10011, *CALC* debe entregar 5.

Resuelva el problema usando este algoritmo:

```
unsigned int calc(unsigned int x) {
  int max1s= 0;
  while (x!=0) {
    x = x & (x>>1);
    max1s++;
  }
  return max1s;
}
```

Observe que el cálculo debe iniciarse cuando la entrada *start* se pone en 1. Por simplicidad considere que *start* se pone en 1 por un solo ciclo del reloj. Después vuelve a 0. Mientras se realiza el cálculo, la salida *rdy* debe permanecer en 0 y colocarse en 1 cuando en la salida *max1s* está el resultado definitivo, tal cual lo muestra el diagrama de tiempo. Las salidas deben mantenerse constantes hasta que *start* se ponga en 1 nuevamente. Puede implementar su solución en el módulo *calc* del archivo *max1s.circ*, para así probar si funciona correctamente con ayuda del módulo *tester*.

## Pregunta 2

***Parte a.-*** (2 puntos) Simplifique la siguiente fórmula booleana a su mínima expresión como suma de productos. Justifique su resultado usando álgebra de boole, recurriendo específicamente a las 2 reglas de simplificación de fórmulas vistas en el minuto 37:12 del video de la clase del viernes 20 de agosto.

$$\overline{x}\,\overline{y}\,\overline{z}\,w + x\,\overline{y}\,z\,w + x\,y\,\overline{z}\,\overline{w} + x\,\overline{y}\,z\,\overline{w} + x\,y\,\overline{z}\,w$$

*Ayuda*: Puede usar los mapas de Karnaugh generados por Logisim para encontrar las simplificaciones convenientes. Pero lo relevante en esta pregunta es justificar las simplicaciones usando algebra de boole. Sin esas justificaciones no tendrá puntaje.

***Parte b.-*** (4 puntos) El programa de la izquierda está en assembler Risc-V. El lado derecho muestra el encabezado de la función *incognito* cuando se invoca desde C. Escriba el programa *equivalente* a *incognito* en C sin usar la instrucción **goto** de C. Preocúpese de *reproducir* en C todos los aspectos del programa original en assembler, en particular el valor retornado.

```
incognito:
        slli    a3,a1,2
        add     a3,a0,a3
        lw      a2,0(a0)
        li      a4,0
        j       .L4
.L3:
        sw      a4,4(a2)
        mv      a4,a2
        addi    a0,a0,4
        lw      a2,0(a0)
.L4:
        bgtu    a3,a0,.L3
        mv      a0, a4
        ret
```

```
typedef struct nodo {
  int x;
  struct nodo *prox;
} Nodo;

Nodo *incognito(Nodo **p,
                int n);
```

Programe la función en el archivo *incognito.c* y compílelo con *make incognito.s* para verificar que su solución al menos no tiene errores sintácticos. Incluso, en el archivo *incognito.s* puede ver el assembler Risc-V generado a partir de su código en C. Pero tenga presente que el assembler generado no va a coincidir con el del enunciado.