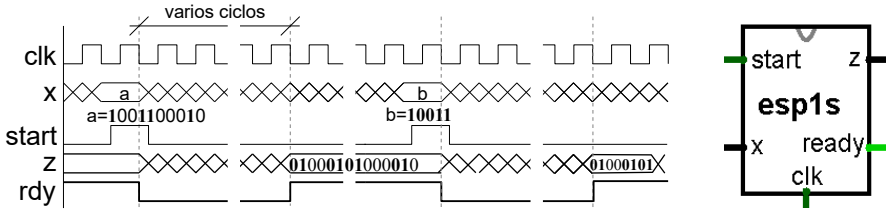
Pregunta 1

Diseñe el circuito *esp1s* (espaciar 1s). Recibe como entrada un entero *x* de 32 bits y entrega como resultado un entero *z* de 32 bits que se obtiene reemplazando en *x* todos los bits en 1 por la secuencia 01. Por ejemplo si *x* es 0b**1**00**11**00**1**0, *z* será 0b**01**000**101**000**01**0. Use el algoritmo del cuadro de la derecha. Cuando *start* se pone en 1, debe llevar la salida *ready* a 0 y calcular *z* en varios ciclos del reloj. Al terminar el cálculo debe colocar *ready* en 1 y entregar el resultado en *z*. No se especifica el resultado si hay desborde en la magnitud. Las salidas se deben mantener constantes hasta que se inicie una nueva búsqueda colocando *start* en 1. A continuación se muestra un ejemplo de uso y las entradas y salidas de *esp1s*.

```
typedef unsigned int uint;
uint esp1s(uint x) {
  uint z= 0;
  int i= 0;
  while (x!=0) {
    int bit= x&1;
    z += bit<<i;
    i += bit ? 2 : 1;
    x = x>>1;
  }
  return z;
}
```



Pregunta 2

línea cache	*etiqueta*	*contenido*	*etiqueta*	*contenido*
b5	2b5		6b5	
78	d78		478	
2a	92a		f2a	

Parte i.- La figura muestra un extracto del estado actual de un *caché* de 8 KB (2^{13} bytes) de 2 grados de asociatividad, cada uno de los 2 bancos con 256 líneas de 16 bytes. El caché no está vacío. Por ejemplo en la línea 2a del caché (en hexadecimal) se almacenan las líneas de memoria que tienen como etiqueta 92a y f2a (es decir, la línea que va de la dirección f2a0 a la dirección f2af). Un programa accede a las siguientes direcciones de memoria (en hexadecimal): 6b58, d789, f2ac, a2a0, 2b54, 1788, 92a0, eb5c y 32a0. **Indique** qué accesos a la memoria son aciertos en el caché, cuáles son desaciertos y **rehaga la figura** mostrando el estado final del caché. Por ejemplo el acceso 6b58 es un acierto. En caso de desaciertos en la misma línea de la memoria caché Ud. debe reemplazar alternadamente en los 2 bancos. Por ejemplo si acaba reemplazar la línea 478, si ocurre otro desacierto en la línea 78 del caché deberá reemplazar la línea d78.

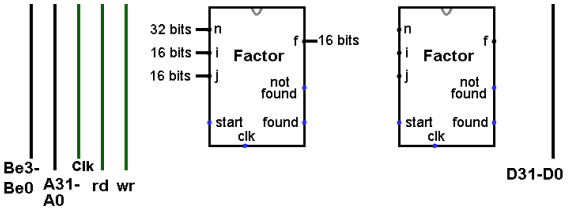
Parte ii.- **Traduzca** la función *esp1s* de la pregunta 1 a assembler Risc-V.

Pregunta 3

a) a3 ← t2+4
b) Inst ← Mem[t2+s5]
c) Mem[t2] ← s5

I. Al reverso del enunciado está el diagrama de bloques del LRV32IM microprogramado estudiado en cátedra. El registro de instrucción *Inst* tiene cargada la instrucción *add a3, t2, s5* (*a3* es el registro de destino). **Indique** de manera independiente cuáles son las señales de control necesarias para realizar las 3 transferencias entre registros del cuadro de arriba en *un solo ciclo del reloj*. *Ayuda*: use la señal de control *type* que regula cuál será la salida de AVY de la unidad de control y decodificación (puede ser *itype*, *stype*, *utype*, *y0* o *y4*). Se descontará puntaje por señales innecesarias.

II. El circuito *Factor* de la tarea 2 busca un factor de *n* en el intervalo [*i*,*j*] cuando la entrada *start* se pone en 1. Mientras tanto las salidas *found* y *not found* son 0. *Found* se pone en 1 si se encuentra un factor, entregándolo en la salida *f*. Si no, *not found* se pone en 1. Considere que Ud dispone de 2 instancias de *Factor* como dispositivos de E/S, que operan como coprocesadores de LRV32IM. ***Diseñe libremente*** una interfaz de estas 2 instancias de Factor con los buses de LRV32IM que muestra la figura. La interfaz debe permitir especificar los valores de *n*, *i*, *j* y *start* y obtener los valores de *f*, *found* y *not found* para ambas instancias. Ud. elige las direcciones de los puertos.

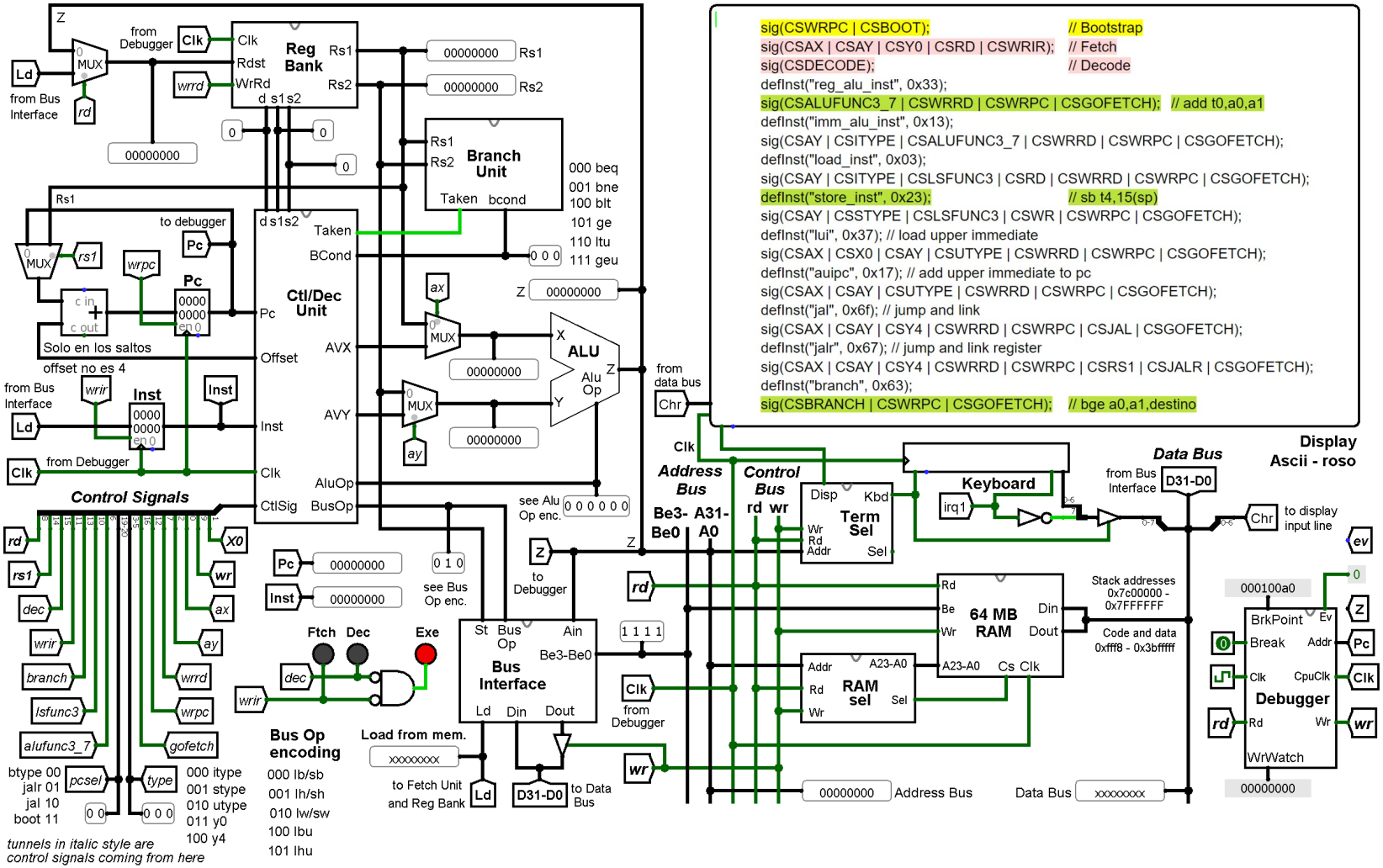


Ayuda: Necesitará 4 registros de 32 bits para almacenar *n*, *i* y *j* para ambas instancias de *Factor* (colocando *i* y *j* en un solo registro). Decodifique *A3* y *A2* para seleccionar qué registro escribe. Haga que al escribir el *n* que ingresa a alguna de las instancias de *Factor*, se lleve *start* a 1 por un ciclo del reloj en esa instancia para iniciar la búsqueda. Si lo prefiere puede usar 6 registros y decodificar A4, A3 y A2. Puede obtener *f*, *found* y *not found*, leyendo un solo puerto de 32 bits.

III.- Programe la función: `uint searchFactor(uint n);`

Esta función debe buscar un factor de *n* en el intervalo [3, *sqrt*(*n*)]. Para realizar la búsqueda **Ud. debe usar la interfaz de E/S de la parte II**, buscando la mitad del intervalo en la 1era. instancia de *Factor*, en paralelo con la búsqueda de la otra mitad del intervalo en la 2da. instancia de *Factor*. Use *busy-waiting* para esperar a que alguna de las instancias termine exitosamente. En tal caso retorne de inmediato el factor encontrado. No espere a que termine la otra instancia. Si ambas instancias fracasan, retorne 0.

CC4301 Arquitectura de Computadores – Examen – Semestre Primavera 2023 – Prof. Luis Mateu




```
sig(CSWRPC | CSBOOT);                          // Bootstrap
sig(CSAX | CSAY | CSY0 | CSRD | CSWRIR);       // Fetch
sig(CSDECODE);                                 // Decode
defInst("reg_alu_inst", 0x33);
sig(CSALUFUNC3_7 | CSWRRD | CSWRPC | CSGOFETCH);  // add t0,a0,a1
defInst("imm_alu_inst", 0x13);
sig(CSAY | CSITYPE | CSALUFUNC3_7 | CSWRRD | CSWRPC | CSGOFETCH);
defInst("load_inst", 0x03);
sig(CSAY | CSITYPE | CSLSFUNC3 | CSRD | CSWRRD | CSWRPC | CSGOFETCH);
defInst("store_inst", 0x23);                   // sb t4,15(sp)
sig(CSAY | CSSTYPE | CSLSFUNC3 | CSWR | CSWRPC | CSGOFETCH);
defInst("lui", 0x37); // load upper immediate
sig(CSAX | CSX0 | CSAY | CSUTYPE | CSWRRD | CSWRPC | CSGOFETCH);
defInst("auipc", 0x17); // add upper immediate to pc
sig(CSAX | CSAY | CSUTYPE | CSWRRD | CSWRPC | CSGOFETCH);
defInst("jal", 0x6f); // jump and link
sig(CSAX | CSAY | CSY4 | CSWRRD | CSWRPC | CSJAL | CSGOFETCH);
defInst("jalr", 0x67); // jump and link register
sig(CSAX | CSAY | CSY4 | CSWRRD | CSWRPC | CSRS1 | CSJALR | CSGOFETCH);
defInst("branch", 0x63);
sig(CSBRANCH | CSWRPC | CSGOFETCH);            // bge a0,a1,destino
```

Alu Op encoding

bit 5 (left): mul/div operation
bit 4: immediate format
bit 3: substract / sra
bit 2-0 (right): **func3**

Alu Op func3 encoding

000 add/sub	100 xor
001 sll	101 srl/sra
010 slt	110 or
011 sltu	111 and

LRV32IM CPU

Risc-V 32 bit integer base instruction set with multiply/divide extension

Simple microprogrammed physical architecture

64 MB RAM

No Memory Management Unit

By Luis Mateu

Previous work by leijurv

GPL Licence