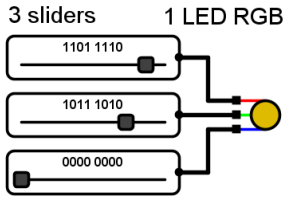
Pregunta 1



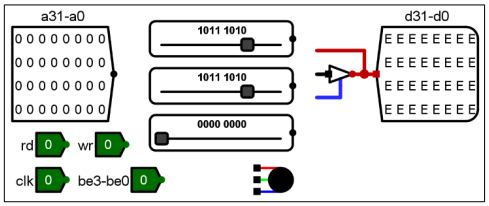
Parte a.- Suponga que Logisim dispone de 2 nuevos dispositivos: el *slider* y el *led rgb*. La figura muestra 3 sliders conectados al led rgb. Un slider genera una salida de 8 bits. El usuario hacer variar el valor de la salida moviendo la manilla con la mano de Logisim, desde 0 (manilla completamente a la izquierda) hasta 255 (completamente a la derecha). El led rgb posee 3 entradas de 8 bits cada una, que indican la intensidad del rojo (arriba), verde (al centro) y azul (abajo). En el ejemplo de la figura el usuario puede mostrar cualquier color moviendo las 3 manillas.

Diseñe una interfaz de entrada y salida para LRV32IM que permita a un programa leer el estado de los 3 sliders y especificar el color del led rgb invocando las funciones:

```
typedef unsigned char uchar;
void getSliders(uchar *pr, uchar *pg, uchar *pg);
void setRgbLed(uchar red, uchar green, uchar blue);
```

Ejemplo de uso:

```
uchar r, g, b;
getSliders(&r, &g, &b);
setRgbLed(r, g, b);
```



El diseño es libre: Ud. define el número de puertos y sus direcciones (puede usar un solo puerto de 32 bits o 3 puertos de 8). La figura de la derecha muestra las entradas y salidas que debe considerar en su interfaz de los dispositivos con los buses de direcciones, datos y control.

Parte b.- **Programe** las funciones *getSliders* y *setRgbLed*.

Pregunta 2

Parte i.- La figura muestra un extracto del estado actual de un *caché* de 4 KB (2^{12} bytes) de 1 grado de asociatividad con 256 líneas de 16 bytes. Por ejemplo en la línea 4a del caché (en hexadecimal) se almacena la línea de memoria que tiene como etiqueta 64a (es decir, la línea que va de la dirección 64a0 a la dirección 64af). Un programa accede a las siguientes direcciones de memoria (en hexadecimal): 64a0, d138, 7c1c, 64a4, 9c10, 5138, 9c14, 513c y 7c18. **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 64a0 es un acierto.

línea cache	*etiqueta*	*contenido*
c1	7c1	
4a	64a	
13	d13	

Parte ii.- Considere el diseño de LRV32IM microprogramado que usó en sus tareas. Al reverso del enunciado está el diagrama de bloques. El registro de instrucción *Inst* tiene cargada la instrucción *andi t0, a5, 21* (*t0* es el registro de destino). **Evalúe** de manera independiente si cada una de las 5 transferencias entre registros del cuadro de la derecha se puede realizar en *un solo ciclo del reloj*. Si la transferencia se puede realizar indique **cuáles son la señales de control** requeridas para llevarla a cabo. Si no se puede realizar, explique por qué.

a) t0 ← a5&21
b) Pc ← Pc+4
c) Inst ←Mem[a5&21]
d) a5 ← t0&21
e) Mem[a5&21] ← t0

Pregunta 3

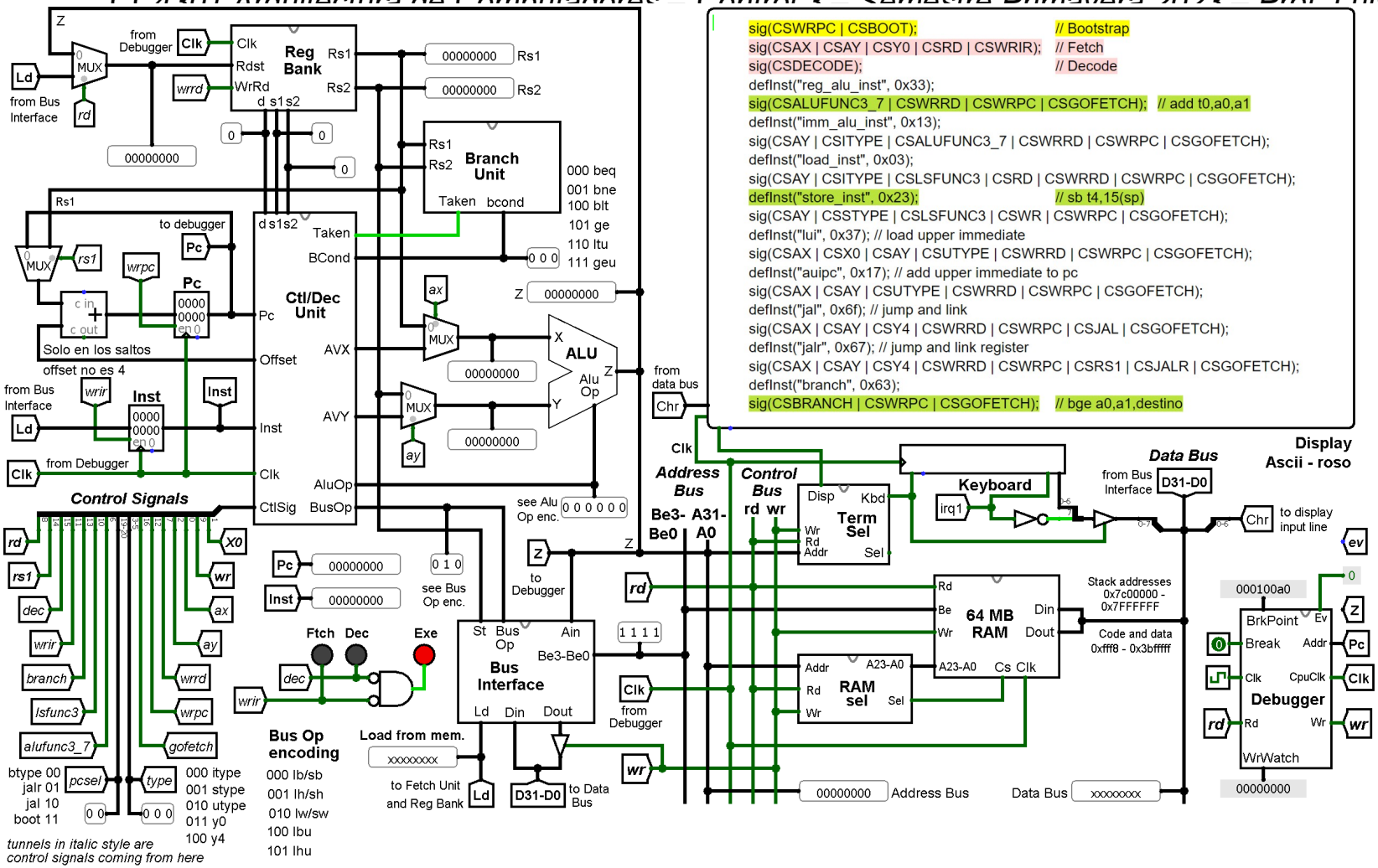
I. El cuadro de la izquierda muestra las instrucciones Risc-V ejecutadas por un programa.

Programa	*Arq. en pipeline*	*Arq. superescalar*
A add t0,a1,a2	Fetch Dec Exe	Fetch Dec Exe
B andi t1,a3,7	1 A	1 AB
C addi t2,a1,100	2 B A	2 CD AB
D sub t3,t2,t1	3 C B A	
E ori t5,t2,255		.
F beq t2,t1, P		.
G add ...	.	.
H sub ...	.	
I xor ...	.	
J andi ...		
...		
P ori t6,a1,15		
Q sub t7,t3,t5		

Considere una arquitectura en pipeline con etapas *fetch*, *decode* y *execute*. **Complete el cuadro del centro** de manera que muestre el ciclo en que se ejecuta cada etapa de las instrucciones hasta ejecutar completamente Q. Considere que la instrucción F sí salta a P, pero que la arquitectura predice que el salto no ocurriría.

II. Repita el mismo cuadro del centro considerando que el salto en F sí ocurre y la arquitectura **sí predice** la ocurrencia del salto en F.

III. Complete el cuadro de la derecha considerando una arquitectura superescalar: 2 pipelines idénticos al pipeline de la parte I. El salto en F sí ocurre, pero la arquitectura predice que no ocurriría.




```
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