Preguntas coloquio Orga1

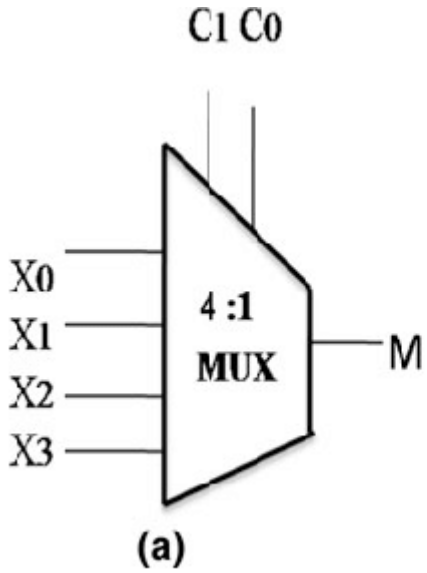
¿Cómo implementaría un circuito que se comporta de igual manera que una fórmula de lógica proposicional?
Respuesta:
En primer lugar, para implementar un circuito con igual comportamiento que una fórmula de lógica proposicional, tendríamos que entender la tabla de verdad de la fórmula que nos gustaría implementar.
Una vez que tenemos nuestra tabla de verdad, utilizamos compuertas lógicas (las cuales nos proporcionan *outputs* determinados dependiendo la compuerta que utilizamos) para combinarlas y llegar a nuestra expresión deseada.
A su vez contamos también con distintos métodos como:

- Suma de productos.
- Producto de sumas.

¿Qué función cumple la señal de reloj? ¿Qué circuito permite detectar el flanco de subida de la señal de clock?
Respuesta:
La señal de clock es el mecanismo utilizado por la UC para ordenar las operaciones que tendrá que realizar nuestra computadora. El clock se mide a partir de sus ciclos (en general se toma como unidad los *Hertz*) y cada operación que realizamos puede tardar más de un ciclo de clock en ejecutarse.
Al trabajar con cambios de estado, sin el componente del clock, nuestra computadora no podría actualizar su *estado* en función del tiempo.
Por otra parte, sabiendo que la operación se ejecuta en el flanco ascendente del ciclo de clock *rising edge*, necesitamos un dispositivo que pueda leer esas señales.
Para eso tenemos el *flip-flop* (y todas sus variantes dentro del mismo). La particularidad del *flip-flop* es que el cambio de estados del mismo se ve afectado no por el valor de la señal del clock en sí, sino por el cambio de flanco de la señal del clock
(transición de 1 -> 0 y 0 -> 1)

¿Qué es un multiplexor y un demultiplexor?

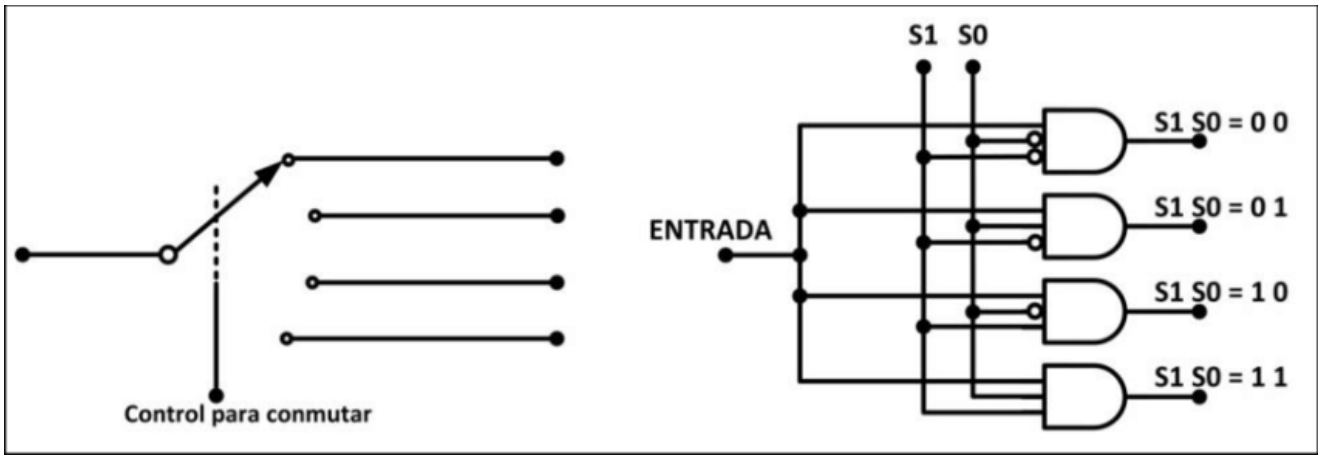
Respuesta:
Un *multiplexor* es un switch que utilizamos para filtrar valores de entrada a direcciones especificas dentro de la memoria. Los multiplexores siguen la forma 2^n a n, donde filtrar 2^n lineas de entrada requiere de n líneas de control.



C1	C0	M
0	0	X0
0	1	X1
1	0	X2
1	1	X3

(b)

Por otro lado, un *demultiplexor* realiza la operación inversa a un mulitplexor. Esto quiere decir que, dada una entrada y n señales de control, el mismo devuelve 2^n posibles salidas



¿A qué le decimos estado de un procesador?

Respuesta:

El estado de un procesador podría pensarse como una *foto* en un momento de tiempo t sobre la información que tenemos almacenada en nuestra computadora. Las computadoras almacenan ese estado a partir de flags (en general el flag Z, flag C, flag N, flag V. flag I *este último en nuestra arquitectura orga1I*)

Por otra parte, tenemos otros indicadores en arquitecturas más complejas para determinar el estado de nuestra computadora, como puede ser:

- Registros dentro de nuestra CPU.
- Registros de nuestra ALU.
- PC (program counter)
- Contenido dentro de la memoria. Esto abarca el estado de los registros dentro de la RAM al estado de nuestra memoria caché.

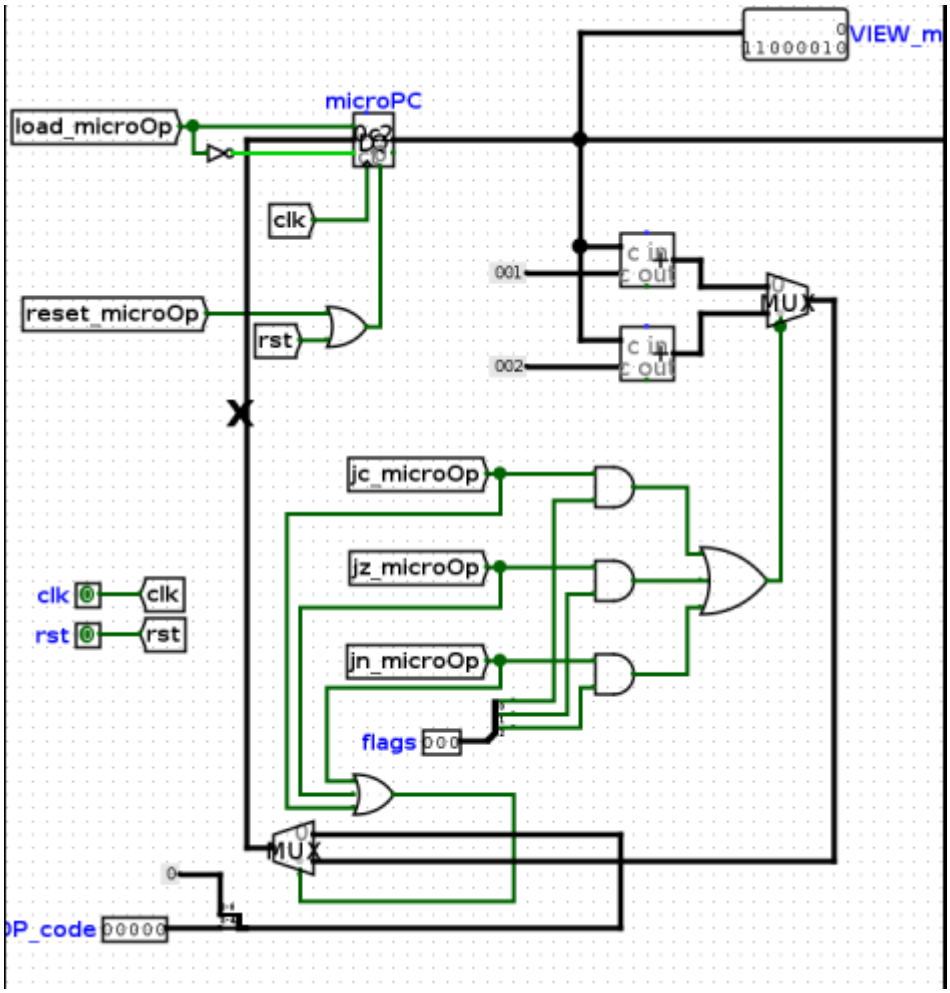
¿Qué función cumple la señal load_microop en orga1small? ¿Qué uso le damos al micropc?

Respuesta:
Cuando tenemos un set de instrucciones cargadas en memoria, cada una de ellas tiene un conjunto de *mini* operaciones asociadas llamadas Micro Instrucciones. Nuestra arquitectura de *orga1small* nos permite acceder, por cada instrucción, al conjunto de micro instrucciones para ejecutarlas y operar.

Nuestra instrucción *load_microop* nos permite cargar cada una de las micro operaciones asociadas a una instrucción e ir ejecutándolas a partir de los ciclos del microPC. Es importante tener en cuenta que cuando ejecutamos una instrucción en un ciclo de clock, dentro de la misma, podemos tener multiples ejecuciones de micro operaciones las cuales se llevan cuenta a partir del microPC; esto implica que en un ciclo de clock, podemos tener más de una micro instrucción siendo ejecutada.

¿Cuál es el mecanismo que utilizamos en orga1small para implementar los saltos condicionales?
Respuesta:
Cuando realizamos un salto condicional, nuestra arquitectura orga1small realiza el siguiente conjunto de operaciones:

- Recibe una instrucción.
- Si alguno de nuestros flags (Z,C,N) se encuentra prendido, nuestro microPC se encargará a partir de los adders de sumarle 2 a nuestra siguiente instrucción (es decir, salta al caso de True). Si nuestro salto condicional fuera False, simplemente ejecuta la siguiente microinstrucción.
- El multiplexor en la parte inferior se encarga de pasar la instrucción y el código de operación respectivo.



¿Qué efecto tiene activar la señal de reset_microop en la unidad de control de orga1small?

Si activamos la señal de reset_microop, simplemente estamos pisando el valor de nuestro microPC a cero. Si esto ocurre, la instrucción que se ejecutará a continuación sera la 0x00.

¿Cuál es el mecanismo que utilizamos en orga1smallI para comenzar a ejecutar una rutina de atención de interrupciones?
Respuesta:
El mecanismo que utilizamos en orga1smallI es el módulo de Interrupt Controller. Cuando se recibe una interrupción, el mismo se encarga de recibir la señal de interrupción del *mundo externo*, llevarla a la UC quien se encargará de dar el INTA (acknowledge). Una vez que la CPU se encuentra lista para atender esa interrupción, se realiza la transferencia de información.

Es importante verificar siempre que realizamos el fetch si tenemos una interrupción, ya que nuestra computadora deberá guardar el estado actual en el que se encuentra y atender esa interrupción. Guardamos el PC en un registro, seteamos I = 0, INTA = 1 y nuestro PC = RAI ya que la siguiente instrucción deberá ser nuestra *rutina de atención de interrupciones.*

¿Cuál es la ventaja de tener operaciones de tres registros en RISC-V en lugar de dos como suele ser el caso en otras arquitecturas?
Respuesta:

En RISCV, ¿qué significa que un registro sea temporal?

Respuesta:

Dentro de la arquitectura RISC-V, la convención nos dice que las variables que utilicemos deberán ser almacenadas en registros ya que el tiempo de acceso hacia ellas es mucho menor que tenerlas "guardadas" en memoria.

Podemos clasificar los registros según si los mismos mutan o no su valor en función de la ejecución de un programa. Aquellos registros que SI mutan dependiendo de la ejecución del código se dicen $temporales$.

Si tenemos una función que lee una variable y luego la va cambiando (por ejemplo, un sumador) la convención nos dice que esa *variable* que cambia deberá ser guardada en un registro temporal.

Registro	Nombre ABI	Descripción	¿Preservado en llamadas?
x0	zero	Alambrado a cero	—
x1	ra	Dirección de retorno	No
x2	sp	Stack pointer	Sí
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Link register temporal/alterno	No
x6–7	t1–2	Temporales	No
x8	s0/fp	Saved register/frame pointer	Sí
x9	s1	Saved register	Sí
x10–11	a0–1	Argumentos de función/valores de retorno	No
x12–17	a2–7	Argumentos de función	No
x18–27	s2–11	Saved registers	Sí
x28–31	t3–6	Temporales	No
f0–7	ft0–7	Temporales, FP	No
f8–9	fs0–1	Saved registers, FP	Sí
f10–11	fa0–1	Argumentos/valores de retorno, FP	No
f12–17	fa2–7	Argumentos, FP	No
f18–27	fs2–11	Saved registers, FP	Sí
f28–31	ft8–11	Temporales, FP	No