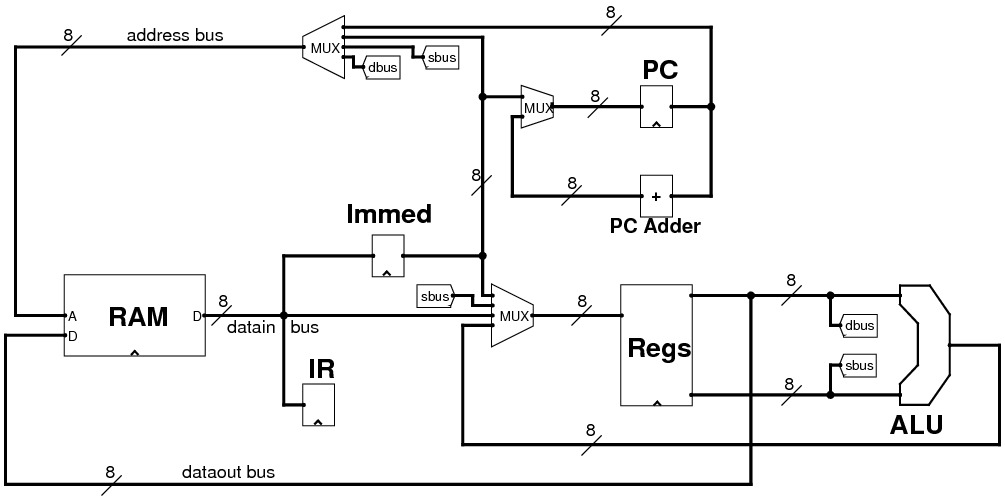
**Un ejemplo de CPU cableada**

**1   Introducción**

* Veremos un Procesador MIPS cableada simple, para demostrar que la lógica de control en una CPU puede construirse utilizando algunas puertas y multiplexores simples.
* El ejemplo de Procesador MIPS se diseñó e implementó en Logisim,
* El objetivo era crear una arquitectura razonablemente tradicional pero con la menor lógica de control posible.

**2.   Arquitectura**

* La CPU tiene un bus de datos de 8 bits y un bus de direcciones de 8 bits, por lo que solo puede admitir 256 bytes de memoria para almacenar tanto instrucciones como datos.
* Internamente, hay cuatro registros de 8 bits, R0 a R3, más un registro de instrucciones, el contador del programa y un registro de 8 bits que contiene valores inmediatos.
* La ALU es la misma que diseñamos la semana pasada. Realiza las cuatro operaciones AND, OR, ADD y SUB en dos valores de 8 bits, y admite ADD y SUB firmados.
* La CPU es una arquitectura de carga / almacenamiento: los datos deben incorporarse a los registros para su manipulación, ya que la ALU solo lee y escribe en los registros.
* El siguiente diagrama muestra las rutas de datos en la CPU:



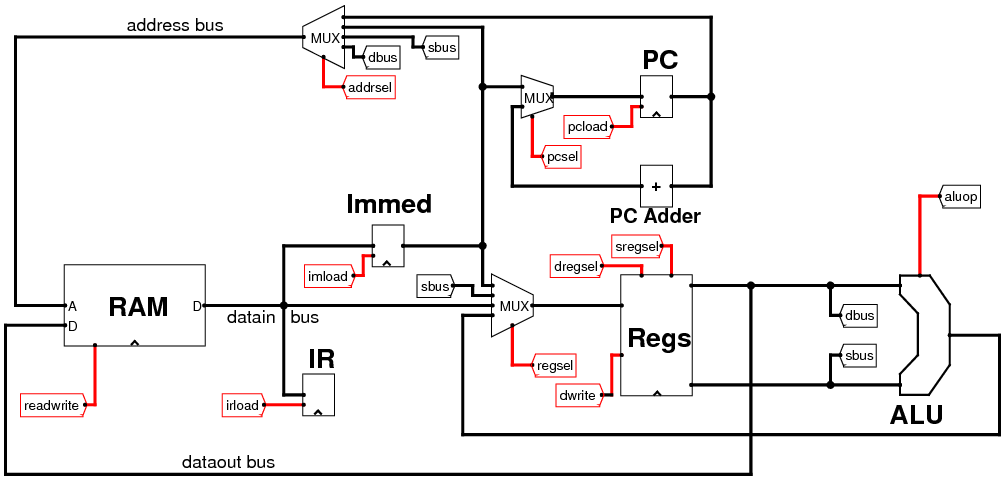
* Las etiquetas *dbus* y *sbus* indican las líneas que salen del archivo de registro que contienen el valor de los registros de destino y fuente.
* Tenga en cuenta el bucle de datos que involucra los registros y la ALU, cuya salida solo puede volver a un registro.
* El bus de datos solo está conectado a la línea *dbus* , por lo que el único valor que se puede escribir en la memoria es el registro de destino.
* También tenga en cuenta que solo hay 3 multiplexores:
  + el multiplexor de bus de dirección puede obtener una dirección de memoria de la PC, el registro inmediato (para direccionamiento directo) o de los registros de origen o destino (para direccionamiento indirecto de registro).
  + el multiplexor de PC permite al PC incrementar o saltar al valor en el registro inmediato.
  + El multiplexor delante de los registros determina de dónde proviene una escritura de registro: la ALU, el registro inmediato, otro registro o el bus de datos.

**4   fases de instrucciones**

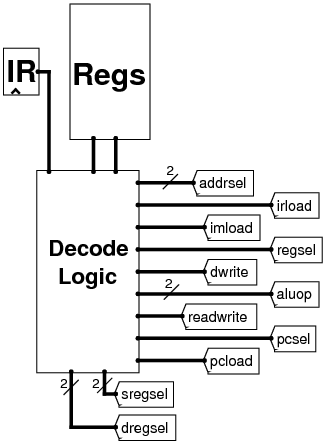
* La CPU tiene internamente tres fases para la ejecución de cada instrucción.
* En la fase 0, la instrucción se recupera de la memoria y se almacena en el Registro de instrucciones.
* En la fase 1, si la instrucción recuperada es una instrucción de dos bytes, el segundo byte se recupera de la memoria y se almacena en el Registro Inmediato. Para instrucciones de un byte, nada ocurre en la fase 1.
* En la fase 2, todo lo demás se realiza según sea necesario, que puede incluir:
  + Una operación ALU, leyendo de dos registros.
  + Una decisión de salto que actualiza la PC.
  + un registro de escritura.
  + una lectura desde una ubicación de memoria.
  + una escritura en una ubicación de memoria.
* Después de la fase 2, la CPU inicia la siguiente instrucción en la fase 0.
* La lógica de control será simple para el trabajo de la fase 0, no difícil para el trabajo de la fase 1, pero complicada para el trabajo de la fase 2.

**5   líneas de control de CPU**

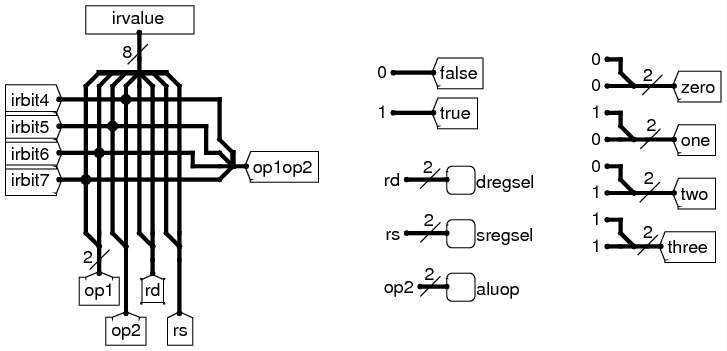
* A continuación se muestra nuevamente el diagrama de la CPU principal, esta vez con las líneas de control mostradas.



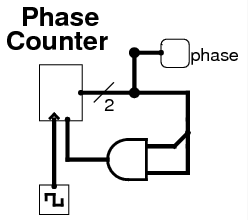
* Hay varias líneas de control de 1 bit:
  + *pcsel* , incrementa PC o carga el valor de salto del Registro Inmediato.
  + *cargue* , carga la PC con un nuevo valor, o no carga un nuevo valor.
  + *cargue* , carga el registro de instrucciones con una nueva instrucción.
  + *imload* , carga el registro inmediato con un nuevo valor.
  + *lectura-escritura* , lectura de la memoria, o escribir en la memoria.
  + *escriba* , *vuelva a* escribir un valor en un registro o no escriba un valor.
* También hay varias líneas de control de 2 bits:
  + *addrsel* , seleccione una dirección de la PC, el registro inmediato, el registro de origen o el registro de destino.
  + *Regsele* , seleccione un valor para escribir en un registro desde el Registro inmediato, otro registro, el bus de datos o desde la ALU.
  + *dregsel* y *sregsel* , seleccione dos registros cuyos valores se envíen a la ALU.
  + *aluop* , que son los bits *op2* que controlan el funcionamiento de la ALU.
* Los valores para todas estas líneas de control son generados por la unidad de control, que obtiene como entrada el valor del registro de instrucciones, y las líneas cero y negativas del registro de destino.



**6   Dentro de la unidad de control**

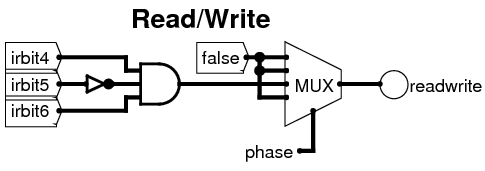


* Dentro del bloque de lógica de decodificación, el valor del registro de instrucciones se divide en varias líneas individuales *irbit4* , *irbit5* , *irbit6* e *irbit7* . *op1* y *op2* se dividen, con *op2* exportado como *aluop* . Finalmente, los 4 bits de código de operación de la instrucción se dividen como la línea *op1op2* .
* Varios de los bits del valor del registro de instrucciones se conectan directamente a estas salidas de 2 bits: *dregsel* , *sregsel* y *aluop* .
* El diagrama parece un poco feo, ya que esta es la forma en que Logisim divide los grupos de líneas en líneas individuales.
* En la implementación de Logisim de la CPU, hay dos líneas "constantes" de 1 bit definidas: *verdadera* y *falsa* , así como varias líneas de 2 bits: *cero* , *una* , *dos* y *tres* .
* También necesitamos algo de lógica para generar la fase actual de ejecución. Esto se hace con un simple contador de 2 bits que está controlado por el ciclo del reloj y que genera una línea de *fase de* 2 bits .



**6.4   Lógica de lectura / escritura**

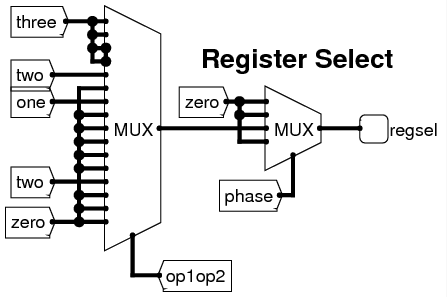
* La línea de lectura / escritura en la memoria solo debe habilitarse cuando estamos realizando operaciones de SW (store word), y solo en la fase 2.
* Los valores *op1op2* para las dos instrucciones de SW son 0101 y 1101, por lo que podemos tratar esto como x101, y establecer *readwrite* true cuando *irbit6* está *activado* , *irbit5* está desactivado e *irbit4*está *activado* .



**6.5   Registrar la lógica de selección**

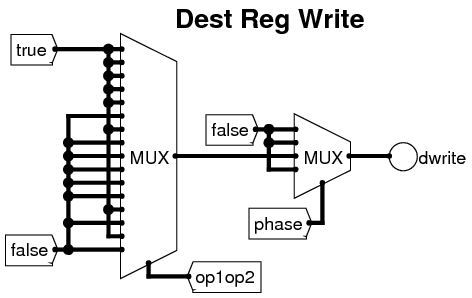
* Ahora llegamos a las líneas de control que están desordenadas. La primera es la línea *regsel* , que selecciona la entrada que se escribirá en el registro de destino. Esto puede ser:

|  |  |
| --- | --- |
| 00 (cero) | Registro inmediato |
| 01 (uno) | *sbus* , es decir, el registro fuente |
| 10 (dos) | bus de datos |
| 11 (tres) | Salida ALU |



**6.6   Destino Registro Escribir Lógica**

* Siguiendo con *Regsel* , necesitamos controlar el modo de *escritura* , es decir, cuando se escribe el registro de destino.
* Nuevamente, podemos usar un multiplexor de 16 entradas, con *op1op2* para elegir el valor de *dwrite* correcto para la salida en la fase 2.

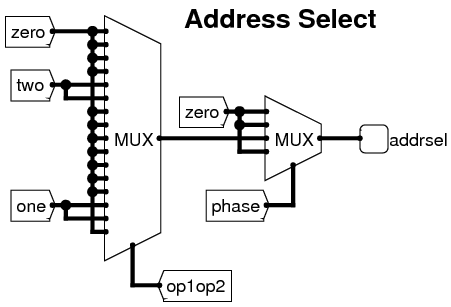


**6.7   Lógica de selección de dirección**

* Cuando queremos leer datos de la memoria principal, la dirección de la que queremos leer se puede seleccionar desde estas entradas:

|  |  |
| --- | --- |
| 00 (cero) | Contador de programa |
| 01 (uno) | Registro inmediato |
| 10 (dos) | *sbus* , es decir, el registro fuente |
| 11 (tres) | *dbus* , es decir, el registro fuente |

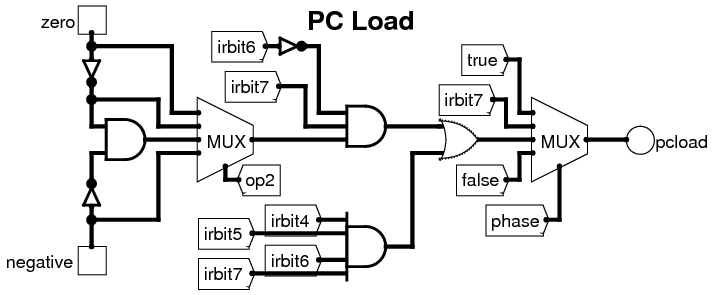
* Al igual que con las dos líneas de control anteriores, no existe una lógica simple para producir el valor en esta línea basado en el código de operación de la instrucción, por lo que recurrimos nuevamente a un multiplexor de 16 entradas.



* Verifique el diagrama lógico anterior con la tabla grande de arriba para asegurarse de que producirá la salida correcta para *addrsel,* dado el valor *op1op2* de la instrucción.

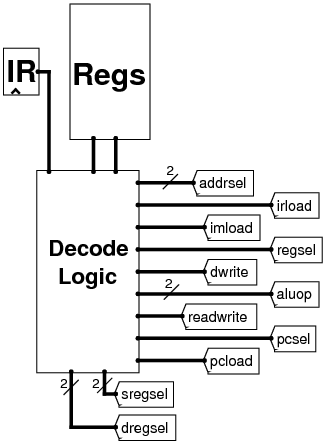
**6.8   Jump Logic**

* Hemos llegado a la última y probablemente la línea de control más complicada de la CPU, *pcload* , que determina cuándo se actualiza el contador de programas.
* En realidad, no es tan complicado, solo se ve feo. Usted ha visto la lógica para la fase 0 y la fase 1. Ahora debemos observar la lógica para las instrucciones de salto.
* Aquí está la lógica para configurar *pcload* para todas las fases:

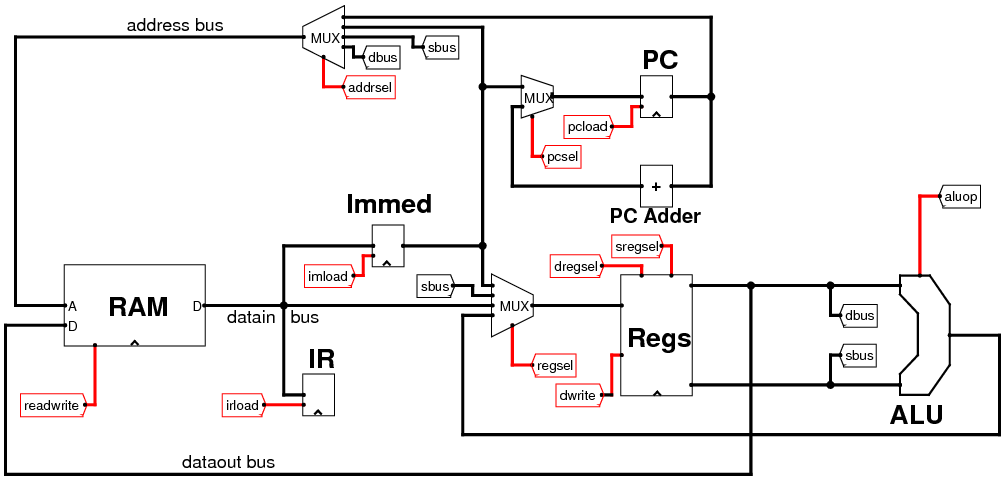


**6.9   finalmente ...**

* Poniendo todo esto de nuevo juntos, ahora tenemos este dispositivo:



que controla el flujo de datos para toda la CPU:



**Implementando la CPU**