Práctica 2

–

Simulación de segmentación

Nombre: David López Pereira

Índice de Contenidos

Índice de Figuras

1. Introducción

En este trabajo se realizará una explicación detallada sobre la abstracción llevada a cabo para implementar la simulación de segmentación de instrucciones mediante el lenguaje de programación Python. En el proyecto se adjuntará un documento de texto con un programa escrito en ensamblador para poder probar la funcionalidad de la simulación.

En la explicación solo me centrare en explicar las clases implementadas como abstracción del circuito, pero el proyecto también cuenta con una clase para leer y poder traducir a una lista el documento de texto y una clase main para poder ejecutar la segmentación desde esta clase.

Si se quiere introducir otra entrada de documento de texto se debe cumplir una serie de restricciones. Se debe tener un espacio entre la instrucción y los registros, los registros se deberán escribir con $ seguido de la letra y el número, y para escribir una etiqueta en una instrucción se hará de la siguiente manera e**tiqueta: instrucción $registro** los dos puntos deben ir pegados a la etiqueta. Las instrucciones implementadas son **lw, sw, li, addi, add, sub, mul, div, j y beq**.

Además, en la conclusión explicare brevemente alguno de los problemas encontrados para realizar esta práctica.

2. Explicación de abstracción del circuito en Python

Como se puede ver en la Figura 1, el diagrama de clases del circuito consiste en 8 clases, donde 7 de ellas forman parte de la clase más importante de esta simulación, la clase segmentación, esta clase la explicare en un punto aparte, ya que presenta más complejidad que las otras.

La primera clase que comentare es la clase **MemInstrucciones** que hace referencia a la estructura instruction memory del circuito mostrado en la Figura 2, esta clase se forma por dos atributos, una lista de instrucciones y un diccionario para guardar las etiquetas encontradas con su posición en la lista de instrucciones, así cuando se inicialice un objeto de la clase MemInstrucciones le tendremos que pasar una lista de instrucciones validas y para cada instrucción de la lista, obtendrá la instrucción en formato **instrucción reg1, reg2**, mediante el uso de split y replace y añadiré la etiqueta al diccionario si se encuentra en la instrucción con el método contains(":"), como se puede ver detallado en el código.

La siguiente clase será **Registros** que como su propio nombre indica contendrá los 32 registros característicos de mips32, estos registros se guardaran en un diccionario llamado registros que contendrá como clave los nombres de los diferentes registros de mips32, todos ellos inicializados a 0.

La clase **MemDatos** hará referencia en el circuito a la estructura Data memory (Figura2), esta clase será un diccionario con 32 posiciones donde cada posición representara una palabra, para que funcione correctamente los desplazamientos deberán ser múltiplos de 4.

La unidad de control del circuito se implementa en la clase **Control**, esta clase contará con un atributo que será un diccionario donde cada clave será uno de los 9 bits de control del circuito. Estos bits de control son **"RegDst", "AluSrc", "MemtoReg", "RegWrite", "MemRead", "MemWrite", "Branch", "AluOp1" y "AluOp2"**. Para inicializar estos bits y que no entre en conflicto con otras condiciones se inicializaran a un valor de 3, pero cuando sean utilizados los valores son 0 si esta desactivado, 1 si esta activado el bit y 2 si no es relevante para esa instrucción, con el método getSeñales le pasaremos una instrucción y en función de la operación de la instrucción pondrá los valores correspondientes a cada bit.

La unidad aritmeticológica y su control se referencian en el circuito en la clase **Alu**. En esta clase se definen 4 atributos enteros que representaran el primer operando, el segundo operando, el resultado y el valor zero, este valor hace referencia al bit PcSource necesario para la instrucción beq, y 1 atributo String que representara la operación que se quiera realizar en cada caso. Para declarar la operación deseada se hace mediante el método setOperacion donde en función del valor de aluOp1, aluOp2 y la instrucción seleccionara una operación. Una vez que contemos con una operación podemos llamar al método calcular para que comprobando la operación la realicemos en lenguaje Python, si el resultado de la resta es 0 se pone zero a 1 para identificar un posible beq.

La clase **Instrucción** se compone de atributos para identificar su tipo, operación (si es sw, add…), el registro destino, el primer registro fuente, el segundo registro fuente, la etiqueta, el desplazamiento, el inmediato (desplazamiento e inmediato inicializados a un número distinto a 0 para saber si ha sido modificado) y la etapa en la que se encuentra. Para poder obtener todos estos datos se tiene el método obtenerInstruccion que pasando una instrucción la divide en dos listas una con la operación y un registro y otra lista con registros o inmediatos, en la primera

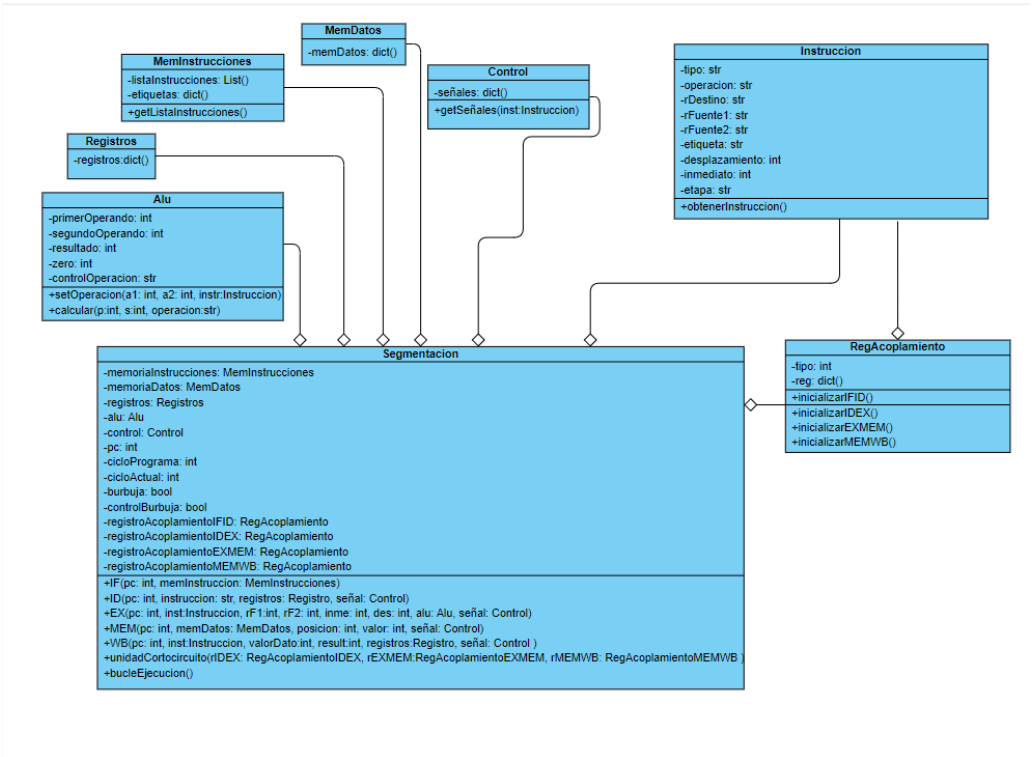

Figura 1 : Diagrama de clases de la abstracción del circuito

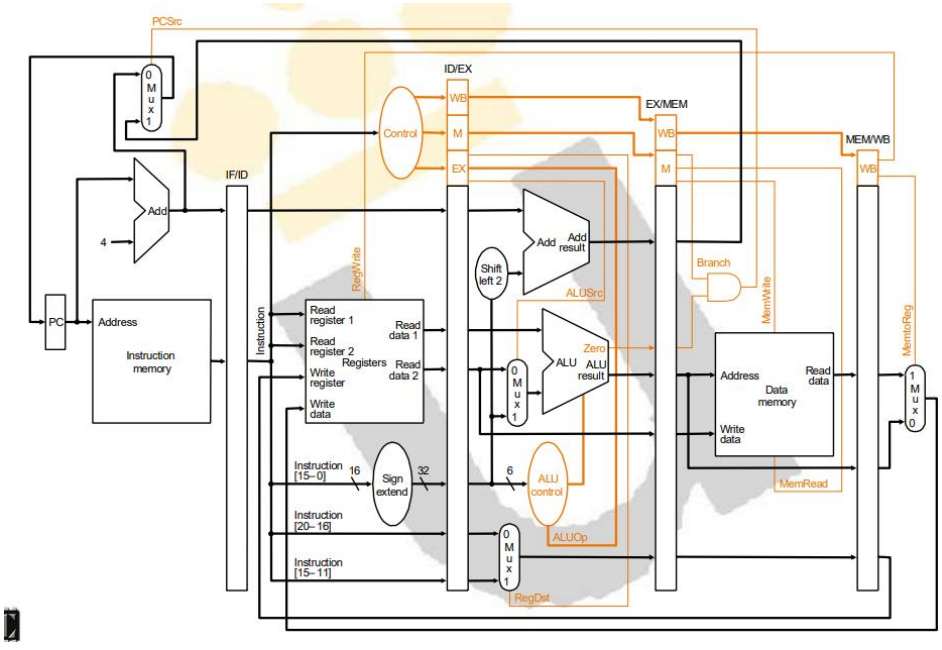

Figura 2 : Circuito (falta unidad de cortocircuito) en el que me he basado para realizar la abstracción en el lenguaje Python

lista volvemos a hacer un Split para separar los espacios y así obtener la operación en la primera posición de la lista resultante, con esto podemos comprobar la operación y en función de cual sea rellenar sus atributos correspondientes, en las operaciones lw y sw el desplazamiento se deberá dividir por 4, para que tenga sentido con la memoria de datos del circuito.

La clase **RegAcoplamiento** hace referencia a los registros IF/ID, ID/EX, EX/MEM y MEM/WB, presentes en la Figura 2, esta clase constará de un tipo que será necesario para identificar cada tipo de registro siendo:

- **Tipo = 0**, se corresponde con el registro IF/ID que será simplemente un String donde se guardará la instrucción,
- **Tipo = 1**, se corresponde con el registro ID/EX que será un diccionario con claves pc, señales de control de IDEX, registro destino, registro fuente 1, registro fuente 2, una instrucción, el desplazamiento y el inmediato.
- **Tipo = 2**, se corresponde con el registro EX/MEM que será un diccionario con claves pc, señales de control de EXMEM, registro destino, registro fuente 2, una instrucción, el resultado y zero.
- **Tipo = 3**, se corresponde con el registro MEM/WB que será un diccionario con claves pc, señales de control de MEMWB, registro destino, una instrucción, resultado de la alu y el resultado de memoria.

Además, esta clase contara con métodos para inicializar cada uno de los registros de acoplamiento y métodos para obtener y modificar los valores, al ser diccionario la modificación se hará mediante update, ya que si se iguala apuntaran al mismo lugar y se modificaran los valores simultáneamente, este fallo me ocurrió en el desarrollo de la práctica.

2.1. Clase Segmentación

Como ya he comentado antes, esta clase es la más importante del programa porque reúne todas las clases explicadas anteriormente, para empezar, explicare cada uno de los atributos presentes en esta clase y luego comentare los métodos presentes en la misma.

Esta clase debe tener todos los datos disponibles del circuito por lo que los atributos se corresponderán con las clases, como es el caso del atributo de **memoria de instrucciones**, **memoria de datos**, **registros**, **alu**, **control** y los **4 tipos de registros de acoplamiento**. Además, contara con más atributos como son **pc** (hace referencia al program counter del circuito), **ciclos de programa** (sirve para controlar el bucle de ejecución), **ciclo actual** (ciclo en el que se encuentra el programa), **burbuja** (booleano para identificar si se realiza burbuja) y **control de burbuja** (si se ha realizado la burbuja controla la no entrada a etapas anteriores). Ciclo actual se encuentra inicializado a -1 para que el primer ciclo corresponda con el número 0.

El primer método implementado es el correspondiente a la etapa **IF**, que se encarga de buscar la instrucción en la memoria de instrucciones y devolverla en modo de String.

La siguiente etapa se encuentra implementada en el método **ID**, en esta etapa el objetivo es decodificar toda la instrucción anteriormente obtenida para poder tener los distintos valores de registros, para ello tendremos que pasarle como entrada el pc, la instrucción (en modo string), los registros y la señal de control, una vez dentro del método creo un objeto de la clase Instrucción para poder llamar a obtener instrucción con el String de instrucción (pasado en la entrada) y con los datos obtenidos en esa variable relleno los atributos de instrucción y obtengo la señal de control de la instrucción, todos estos valores son devueltos por el método.

La etapa de ejecución en la alu se implementa con el método **EX**, en este método pasaremos como variables de entrada la instrucción, también pasaremos los datos relevantes de la instrucción (registros, inmediatos y desplazamiento), la alu y la señal de control, esta señal nos permitirá diferenciar si se tiene que realizar fase EX o no, ya que si el AluSrc es 2 no tendrá que

realizar ninguna operación, si en otro caso AluSrc es 1 el valor de registroFuente1 se tiene que calcular con el valor del inmediato o de desplazamiento, si los valores de alguno de los dos son distintos a -1 realizaremos el cálculo con la operación calcular(), para la que antes habremos calculado la operación correspondiente, si el valor de AluSrc es 0 entonces se calculara el valor con los registros Fuente1 y Fuente2. En este método devolveremos el resultado de la operación y el valor de zero.

El método **MEM** controlara si la instrucción tiene que leer de memoria o escribir en la memoria de datos, para ello le pasamos como argumentos de entrada el pc, la memoria de datos, la posición de la memoria que deseamos consultar, el valor que queramos introducir o guardar y la señal de control que permitirá conocer en función de si el MemRead o el MemWrite están en 1 si tenemos que leer o escribir en memoria, respectivamente. Si en cualquier otro caso MemRead y MemWrite se encuentran a 1 los dos o a 0 no se realizará la etapa MEM para esa instrucción y se devolverá el valor anterior.

La última etapa de la segmentación en mips se corresponde con la escritura de la instrucción, esta etapa se encuentra implementada con el método **WB**, aquí se le pasará como valor de entrada el pc la instrucción el valor de datos de la fase MEM, el resultado de la alu, los registros y la señal de control que decidirá en función del valor de MemtoReg y de RegWrite la operación a realizar. Para que se realice esta etapa el valor de RegWrite tiene que ser 1, si tiene otro valor diferente no se hará ninguna operación, si el valor de MemtoReg es 1 se escribirán los datos en memoria del registro destino el valor de memoria de datos, si es 1 y la instrucción es li se escribirá el inmediato y si es 0 se escribirá en el registro destino el valor del resultado de la alu.

El siguiente método que he llevado a cabo es **la unidad de cortocircuito** que recibe como entrada los registros de acoplamiento IDEX, EXMEM y MEMWB, para poder detectar si existe riesgos de datos en las instrucciones. Para detectarlo compruebo primero que la instrucción de IDEX no sea una instrucción lw o li, si es distinta tengo que comprobar que los registros de la instrucción situados en IDEX sean iguales que los de EXMEM o que los de MEMWB y que los registros de IDEX sean distintos a zero (que tengan algún valor valido). En el caso que los registros de IDEX sean iguales a los de EXMEM si la instrucción es lw tengo que realizar una burbuja, si es li actualizo el valor del registro correspondiente por el inmediato y si es cualquier otra instrucción se actualiza con el valor de la alu. En el caso que los registros coincidan con MEMWB si es una instrucción lw se actualiza el valor del registro con el resultado de memoria, si es li se actualiza con el inmediato y si es cualquier otra instrucción con el resultado de la alu. Este método devolverá un booleano indicando si ha realizado burbuja o no.

El método con la funcionalidad de la segmentación propiamente dicha es el método **bucleEjecucion** en el que el orden de las etapas será el explicado en clase primero la etapa WB, luego la etapa MEM, la siguiente será la fase EX, la siguiente la fase ID y por último la fase IF. Este orden tiene que ser así por el modo en el que esta implementado la máquina segmentada. El método consta de un bucle while que estará activo mientras que ciclo de programa sea 1, este bucle en su interior contará con 5 condiciones if cada una representando las etapas de la segmentación.

En el primer condicional se comprueba si la etapa de la instrucción es WB, si es así se realiza el método WB pasando como entrada los valores del registro de acoplamiento MEMWB, además en esta etapa tengo que comprobar si se trata de la última instrucción si es así establezco el valor de ciclo de programa a 0 para finalizar el bucle while.

En el segundo condicional if compruebo si la etapa de la instrucción es MEM y si el ciclo de programa es distinto a 0, para comprobar si lo he actualizado en el condicional anterior, dentro del if realizare el método MEM con parámetros de entrada correspondientes al registro de acoplamiento EXMEM, el valor devuelto por este método lo guardare en el resultadoMem del registro de acoplamientoMEMWB, se establece la etapa de la instrucción de EXMEM como WB y luego se igualan todos los valores del registroMEMWB con los correspondientes del registro EXMEM. Además, compruebo si la burbuja esta activada si es así establezco la etapa de la instrucción del registro IDEX a ID, inicializo el registro EXMEM, disminuyo el pc en 1 y establezco la instrucción de IFID a una instrucción auxiliar.

EL tercer condicional se corresponde con la etapa EX, donde comprobare si la etapa de la instrucción del registro de acoplamiento IDEX es EX y si el ciclo de programa es distinto a 0, realizare el método EX pasando como argumentos de entrada los valores correspondientes del registro de acoplamiento IDEX y las salidas del método se igualarán a resultado y zero del registro EXMEM, además, la etapa de la instrucción se cambia a MEM y los valores del registro EXMEM se actualizan con los de IDEX. Dentro de este condicional también se comprueba si es la última instrucción entrando en fase EX para lo que se activara controlBurbuja, también se comprobara si se realiza el salto a una etiqueta o no.

La cuarta condición if hace referencia a la etapa ID donde además de comprobar si no es la última instrucción y el valor de cicloPorgrama se comprueba si controlBurbuja es falsa, si se cumple la condición si el pc coincide con la instrucción leída en el registro IFID entonces se realiza el método ID con valores el String guardado en IFID y los atributos de la clase, para luego actualizar el registro IDEX con la salida del método ID. Por último, en este condicional se llama a la unidad de cortocircuito para comprobar que los registros son distintos a los de las otras instrucciones.

Para terminar con el método se comprueba el **último condicional if** que representara la fase IF, aquí se guardara el valor de IFID en un auxiliar y se cambiara el valor de este registro de acoplamiento con el método IF, el if se terminara sumando uno al pc.

3. Conclusión

La principal barrera encontrada para iniciar esta práctica ha sido el hecho de tener que abstraer el circuito en distintas clases para poder entender y programarlo, para ello realice el diagrama de clases de la Figura 1 que me permite ver las clases de una forma visual, entendiendo así la relación de cada componente.

El hecho de utilizar diccionarios para los registros de acoplamiento, al principio, debido a fallos como el comentado anteriormente de la actualización de los datos mediante = en vez del método update, me ha complicado un poco el desarrollo de la práctica, pero una vez solucionado los errores me parece que es la estructura de datos adecuada porque permiten aumentar la legibilidad del código.

Por último, la gran dificultad encontrada ha sido el hecho de realizar el bucle while siguiendo el orden explicado en clase comprobando primero la fase WB.