Sistemas Digitales

Diseño de
Ruta de Datos y
Unidad de Control

Implementación de un MIPS Básico

- Para empezar, vamos a implementar lo necesario para realizar el siguiente conjunto de instrucciones:
  - Instrucciones de memoria (lw y sw)
  - Instrucciones aritmético lógicas (add, sub, and, or y slt)
  - Instrucciones de ramificación (beq) y la instrucción salto (j) que se agregará luego al diseño.
- Al implementar podemos ver que la arquitectura del set de instrucciones determina varios aspectos de la implementación.

Descripción de la Implementación

- Mucho de lo que se requiere hacer no depende del tipo de instrucción.
- Para cualquier instrucción los dos primeros pasos son idénticos:
  - Enviar el contador de programa (PC) a la memoria que contiene el código y obtener la instrucción.
  - Leer uno o dos registros usando campos de la instrucción para seleccionar los registros a leer. Para la instrucción lw necesitamos leer un solo registro, pero muchas de las otras requieren leer dos registros.
- Luego de estos pasos, las acciones para completar el proceso dependen de la instrucción misma.

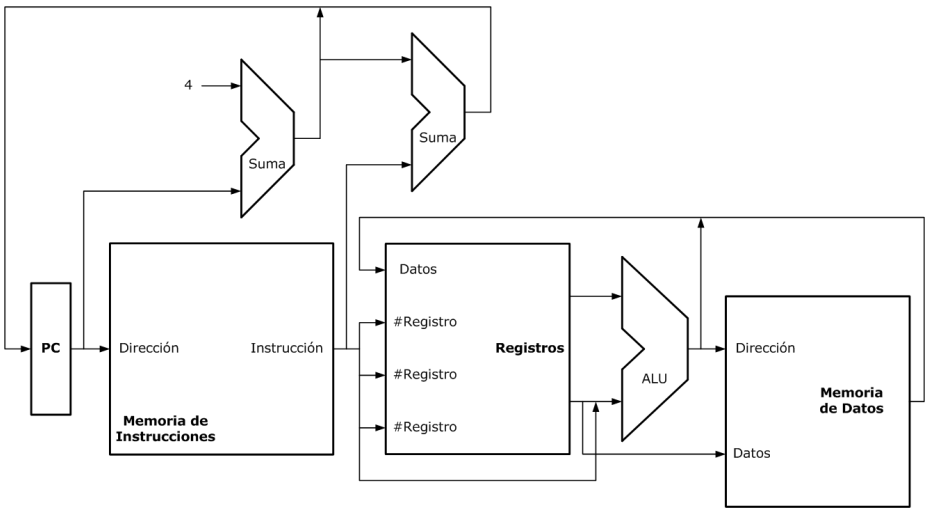
Descripción de la Implementación

- A pesar de las diferentes clases de instrucciones, hay ciertas similitudes en el uso de los elementos de la ruta de datos. Por ejemplo todas las instrucciones mencionadas, excepto jump, usan el ALU:
  - Las instrucciones de memoria para calcular la dirección.
  - Las instrucciones aritmético lógicas para ejecutar operaciones.
  - Las instrucciones de ramificación para efectuar la comparación mediante la resta.

Descripción de la Implementación

- Después de utilizar el ALU, las acciones son distintas:
  - Las instrucciones de memoria necesitan acceder a memoria para leer o escribir datos.
  - Las instrucciones aritmético lógicas escriben el resultado desde el ALU hacia un registro.
  - Las instrucciones de ramificación necesitan cambiar la dirección de la siguiente instrucción a ejecutar basado en el resultado de la comparación.
- A continuación un esquema base de acuerdo a lo descrito sobre las instrucciones.

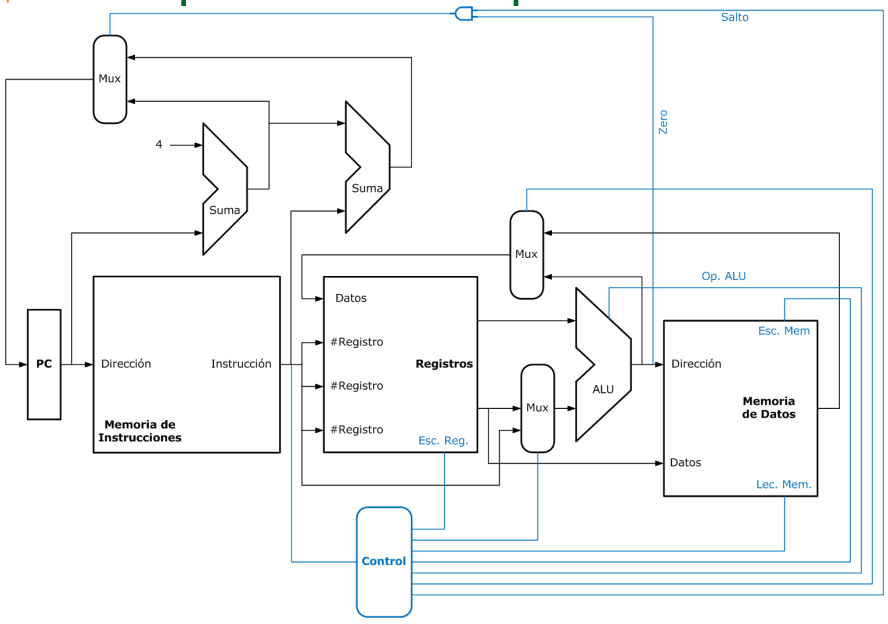
Descripción de la Implementación



Descripción de la Implementación

- Algunos datos que van hacia algún bloque vienen de dos fuentes. Esto no se puede hacer, por lo que se agrega un elemento llamado *selector de datos*. Las líneas de selección vienen de la información tomada a partir de la instrucción que se ejecuta.
- Varias de las unidades requieren un control y esto depende de la instrucción. Por ejemplo:
  - La memoria de datos debe leer en un load (lw) y escribir en un store (sw).
  - El registro debe escribirse en un load u operación.
- Estas operaciones adicionales son dirigidas por líneas de control que se establecen de acuerdo con los formatos y campos de las instrucciones.

Descripción de la Implementación



Descripción de la Implementación

- Se agregaron los multiplexores necesarios así como las líneas de control. Vemos que uno de los multiplexores no es controlado por esta unidad sino por el valor cero del ALU (usado en la comparación de la instrucción beq).
- En este diseño cada instrucción empieza con un flanco de reloj y termina con el siguiente flanco.
- Este diseño no es práctico pues es más lento que aquel donde los distintos tipos de instrucción usan diferentes números de reloj.

Descripción de la Implementación

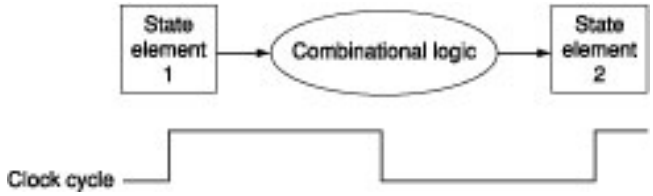
- La ruta de datos de ciclo simple (monociclo)descrita debe tener memoria de datos e instrucciones separados porque:
  - El formato de los datos e instrucciones es diferente en MIPS y por consiguiente se necesitan diferentes memorias.
  - Tener memorias separadas es más barato.
  - El procesador opera en un ciclo y no puede usar una memoria de un solo puerto para dos accesos distintos dentro de ese ciclo.

Metodología de Temporización

- Define cuando las señales se pueden leer o escribir. Es importante definirlos pues si una señal se escribe en el mismo momento que se lee, el resultado puede ser incierto.
- Por simplicidad se asume la metodología de disparo por flanco. Como solo los elementos de estado pueden almacenar datos, la lógica combinacional debe tener sus entradas provenientes de un conjunto de elementos de estado y sus salidas escritas en un conjunto de elementos de estado.

Metodología de Temporización

- Cuando los elementos de estado se escriben, además del flanco de reloj se debe tener una señal de control de escritura. Solo cuando esta señal esté activa y ocurra el flanco, el elemento de estado cambiará.
- En el primer ciclo se activan las salidas del bloque 1, pasan por el combinacional y llegan a las entradas del bloque 2.
- En el siguiente ciclo las salidas del bloque 2 se activan.

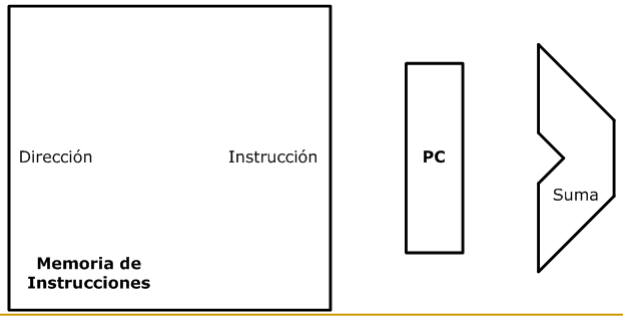


Metodología de Temporización

- La metodología del disparo por flanco permite leer un registro, pasarlo a la lógica combinacional y escribir en el mismo ciclo de reloj. Con esta metodología no hay realimentación dentro de un solo ciclo de reloj.

La Ruta de Datos

- Para empezar necesitamos conocer los principales componentes necesarios para ejecutar cada instrucción. Veamos que elementos de ruta de datos necesita cada instrucción.

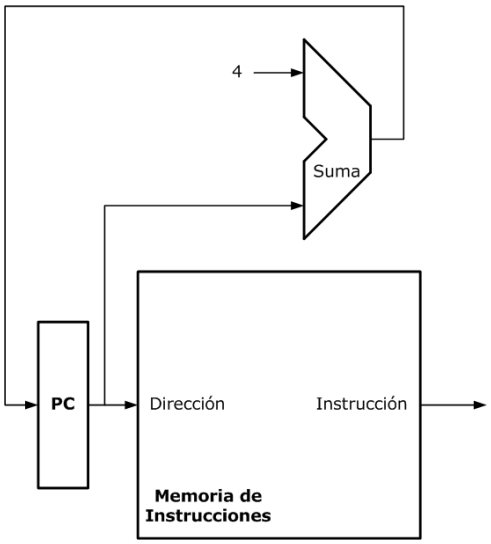


La Ruta de Datos

- Primer elemento: Una unidad memoria para almacenar las instrucciones del programa y entregarlas para una dirección dada.
- Segundo elemento: Un registro llamado Contador de Programa (PC) que se usa para retener la dirección de la instrucción actual.
- Tercer elemento: Un sumador para incrementar el PC a la dirección de la siguiente instrucción.
- Para ejecutar una instrucción, se debe recoger la instrucción desde memoria.
- Para prepararse a ejecutar la siguiente instrucción se debe incrementar el PC.

La Ruta de Datos

- El incremento del PC es distinto en instrucciones de 32 bits pues al pasar de una dirección a otra el salto debe ser de cuatro posiciones. Esto sucede cuando la memoria es de 8 bits por posición.

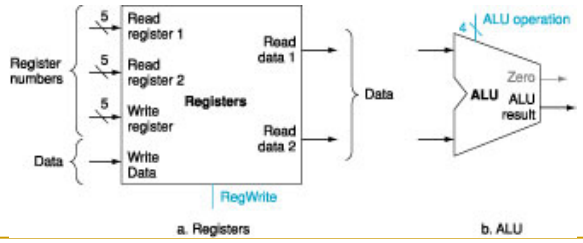


Inst. Aritmético Lógicas (tipo R)

- Las instrucciones tipo R requieren leer dos registros, realizar la operación en el ALU y escribir el resultado.
- Los registros del procesador están en una estructura llamada archivo de registros y se acceden especificando su número (selector). También se necesita el ALU para operar estos registros.
- Como las instrucciones tienen 3 operandos, se necesitan leer dos datos y escribir un dato (resultado) en los registros durante la ejecución.
- Por cada dato leído necesitamos una entrada que especifique el número de registro a leer y una salida que muestre el dato leído.

Inst. Aritmético Lógicas (tipo R)

- Para escribir un dato necesitamos dos entradas:
  - Una para seleccionar el registro donde se escribe.
  - Una para el dato que se va a escribir.
- El archivo de registros siempre muestra en su salida el contenido del registro seleccionado.
- La escritura está controlada por una señal (RegWrite) que se valida en cada flanco de reloj.

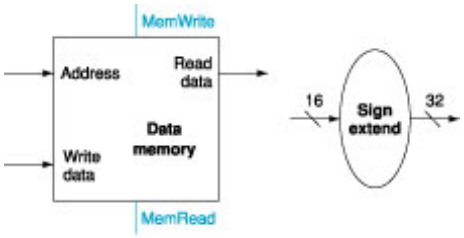


Inst. Load y Store

- Las instrucciones tienen la forma general:
  - lw $t1,offset($t2)
  - sw $t1,offset($t2)
- Los offset son números de 16 bits con signo.
- Si la instrucción es load, el valor leído desde memoria se guarda en $t1 (archivo de registros)
- Si la instrucción es store, el valor se lee de $t1 y se guarda en la memoria.
- En estas instrucciones se calcula la dirección de memoria sumando al registro base $t2 el valor del offset contenido en la instrucción.
- Por lo tanto se requiere también el archivo de registros y el ALU.

Inst. Load y Store

- Además de lo mencionado, se va a necesitar de una unidad para extender el del campo de offset (16 bits) a un valor de 32 bits para operarlo con el valor del registro.
- También se requiere de una unidad de memoria de datos que debe tener control de lectura y escritura así como buses de dirección y datos.



Inst. de ramificación (beq)

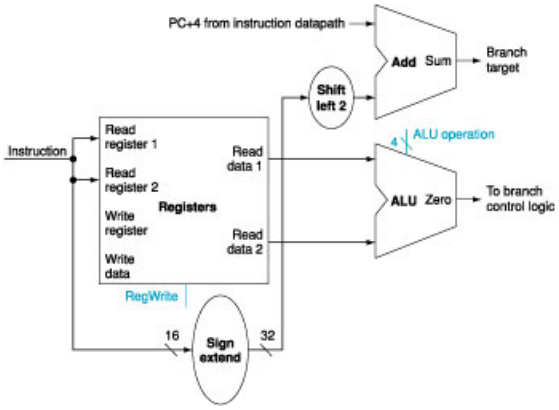
- El formato es: beq $t1,$t2,offset
- Tiene tres operandos:
  - Dos registros que se comparan.
  - Un offset de 16 bits empleado para calcular la dirección destino a partir de la dirección de la instrucción actual.
- Para implementar la instrucción debemos calcular la dirección de salto sumando el offset a la dirección actual almacenado en el PC.

Inst. de ramificación (beq)

- Hay dos detalles que debemos tener en cuenta:
  - La arquitectura especifica que las direcciones en memoria están siempre saltando de a 4 y por consiguiente se usa este esquema en beq.
  - El offset indica cuantas instrucciones se deben saltar, pero al ser de 4 en 4 el salto en memoria por cada instrucción,debemos multiplicar el offset por 4 y eso se logra desplazando el número hacia la izquierda dos veces.
- En el cálculo de la dirección ramificada se debe determinar cual es la siguiente instrucción a ejecutar:
  - Si la condición es verdadera la dirección calculada para la ramificación se carga en el PC.
  - Si la condición es falsa, se utilizará el PC que cambió naturalmente.

Inst. de ramificación (beq)

- Entonces, la ruta de datos de beq hace las operaciones:
  - Calcular la dirección de salto.
  - Comparar los contenidos de los registros.



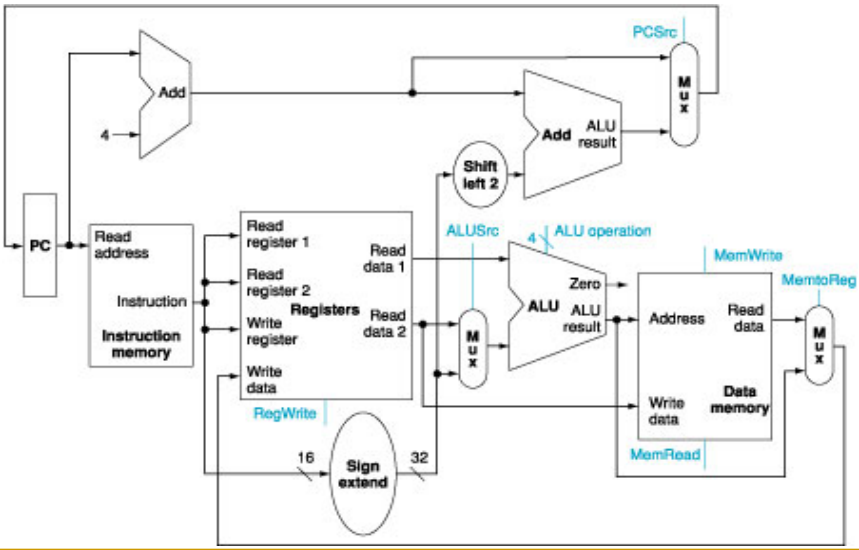
Construcción de la Ruta de Datos

- Los análisis de las instrucciones ahora se combinan para formar la ruta de datos que se completará con el control necesario.
- Las instrucciones se ejecutarán en un ciclo de reloj, es decir, ningún recurso del sistema se usará más de una vez por instrucción. Entonces se necesitan memorias separadas para instrucciones y datos.
- El uso de funciones similares para las distintas instrucciones nos permite compartirlas entre ellas agregando el control necesario.

Construcción de la Ruta de Datos

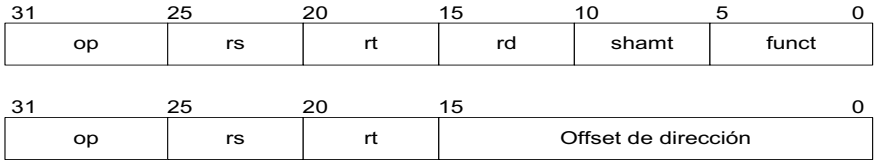
- En la construcción se van a combinar las siguientes rutas de datos parciales:
  - Busqueda de instrucciones (fetch).
  - Instrucciones tipo R
  - Instrucciones de memoria
  - Instrucciones de ramificación
- Con la ruta de datos completa se incorpora la unidad de control. Esta debe ser capaz de tomar las entradas y generar una señal de salida para cada elemento de estado, selector de multiplexor y control del ALU.

Construcción de la Ruta de Datos



Incorporando el Control

- El control que vamos a incorporar nos permitirá:
  - Seleccionar las operaciones que se efectuarán.
  - Controlar el flujo de los datos.
- Toda la información viene de la instrucción de 32 bits.
- Revisamos los formatos de las instrucciones para determinar los controles necesarios.



Control para el ALU

- El ALU posee cuatro entradas para elegir la operación que se va a realizar (Creo que esto ya lo conocen…). En el diseño actual, los códigos son:

Líneas de ALU	Función
0000	AND
0001	OR
0010	Suma
0110	Resta
0111	Set on less than
1100	NOR

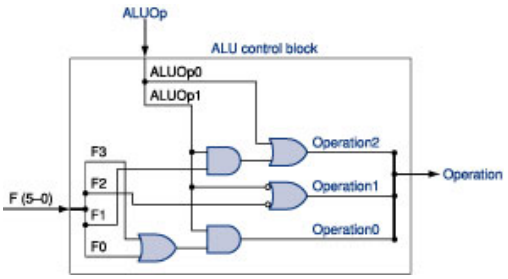
Control para el ALU

- Para generar estos 4 bits necesitamos solo 2 bits a los que llamamos ALUOp. Con estos bits se define si la operación será de suma para lw y sw; resta para beq o aquella que se determina según el valor del campo "función" de la instrucción R.

Opcode	ALUOp	Instrucción	Función	Acción del ALU	Control ALU
LW	00	Load word	XXXXXX	Suma	0010
SW	00	Store word	XXXXXX	Suma	0010
BEQ	01	Branch if equal	XXXXXX	Resta	0110
Tipo R	10	Add	100000	Suma	0010
Tipo R	10	Substract	100010	Resta	0110
Tipo R	10	AND	100100	And	0000
Tipo R	10	OR	100101	Or	0001
Tipo R	10	Set on less than	101010	Set on less than	0111

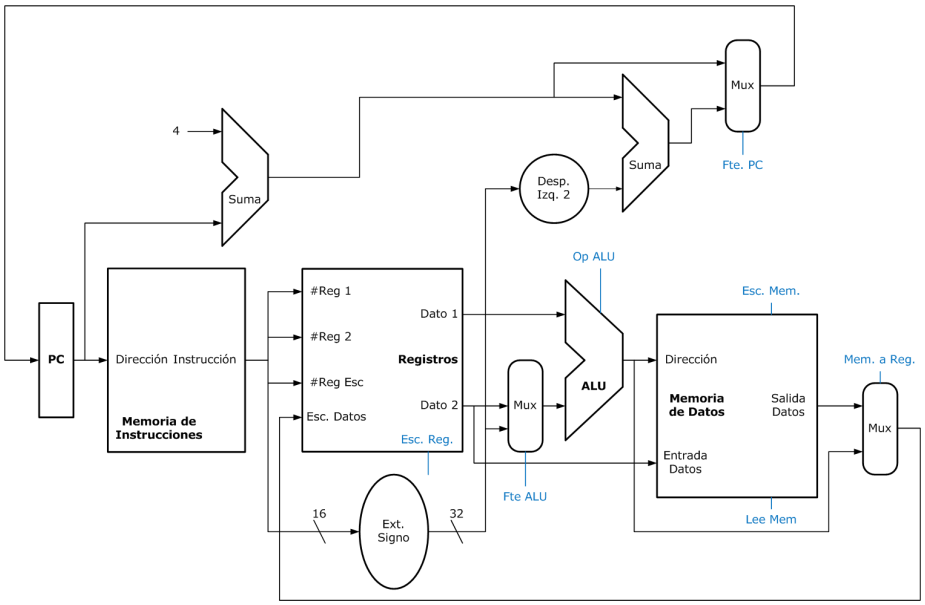
Implementación del Control de ALU

- La tabla de verdad y el circuito de control del ALU se muestran. Tener en cuenta que F5 y F4 no se utilizan. Este también ya lo conocen!



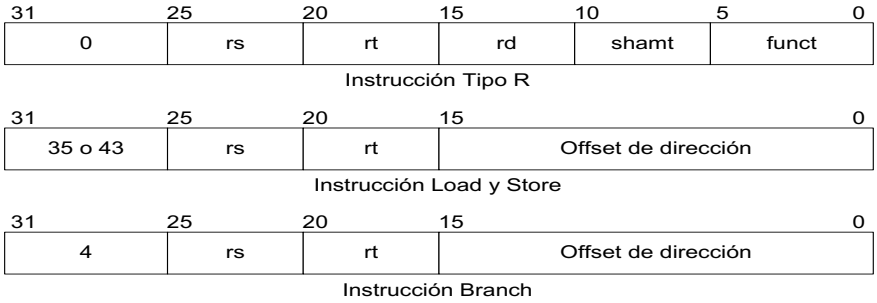
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operación
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Ruta de Datos con Control de ALU



Unidad de Control Principal

- En la implementación se incorporan los códigos de operación de las instrucciones. Estos nos servirán para enviar las señales de control a los bloques funcionales de nuestro diseño.



Líneas de Control del Diseño

- En la implementación de la Unidad de Control debemos considerar, además de los bits ALUOp, otros siete bits de, cuyas funciones se resumen a continuación.

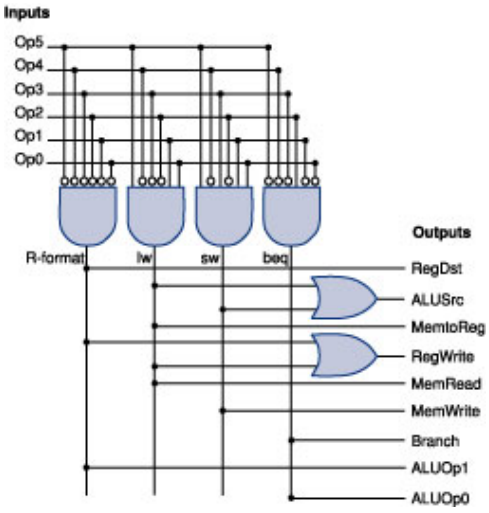
Señal	Efecto cuando se desactiva	Efecto cuando se activa
RegDst	El número del registro destino para escritura viene del campo rt	El número del registro destino para escritura viene del campo rd
RegWrite	Ninguno	El registro seleccionado en Write Register se escribe con el valor en Write Data
ALUSrc	El segundo operando del ALU viene de la segunda salida de registros	El segundo operando del ALU son los 16 bits de la instrucción, extendidos en signo
PCSrc	El PC es reemplazado por PC+4	El PC es reemplazado por el calculo del desplazamiento para el salto
MemRead	Ninguno	Se lee memoria
MemWrite	Ninguno	Se escribe en memoria
MemtoReg	El dato en Write Data viene del ALU	El dato en Write Data viene de la memoria de datos.

Selección de las Líneas de Control

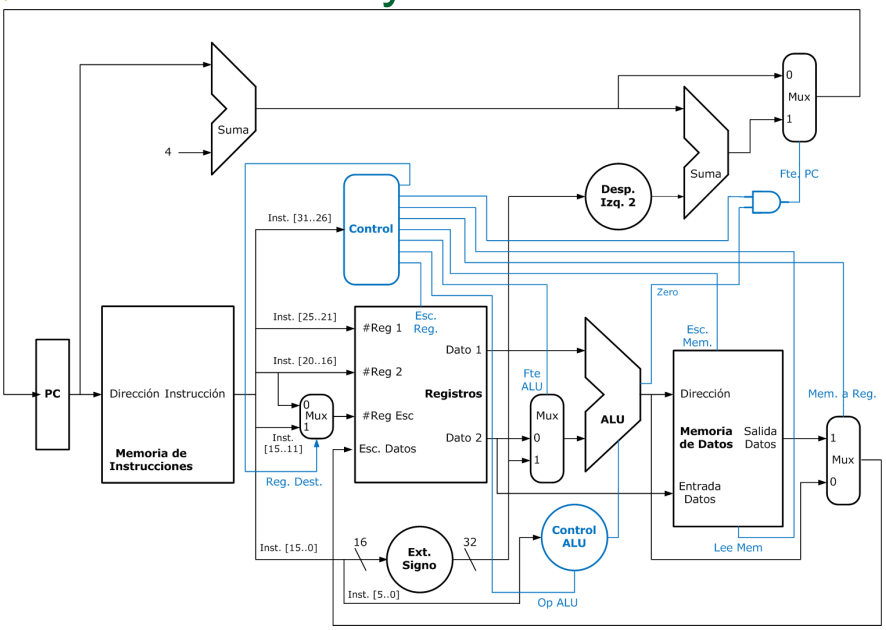
- Las líneas de control vistas anteriormente así como la subunidad para branch y el control del ALU se activarán de acuerdo al opcode y así tenemos:

Instrucción	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

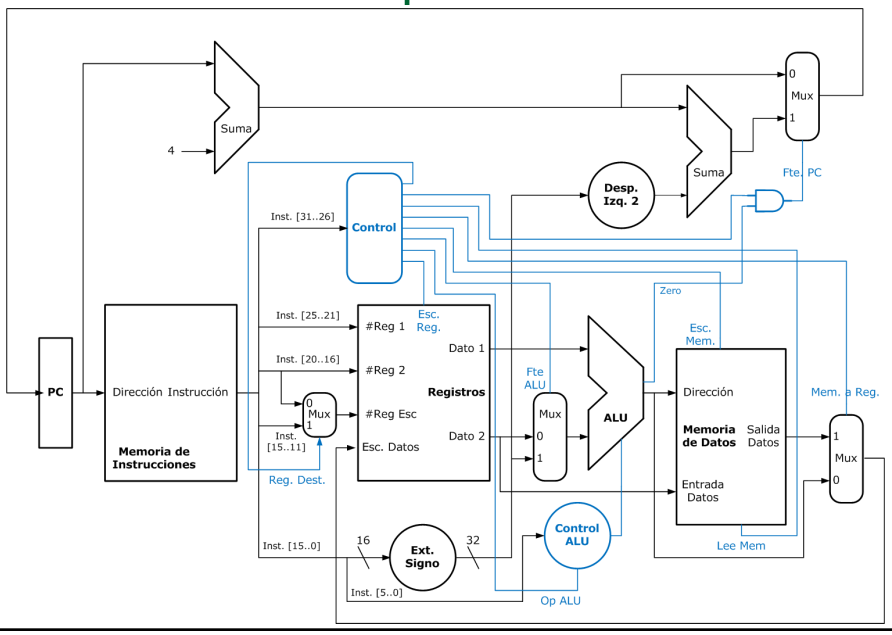
Implementación Unidad de Control



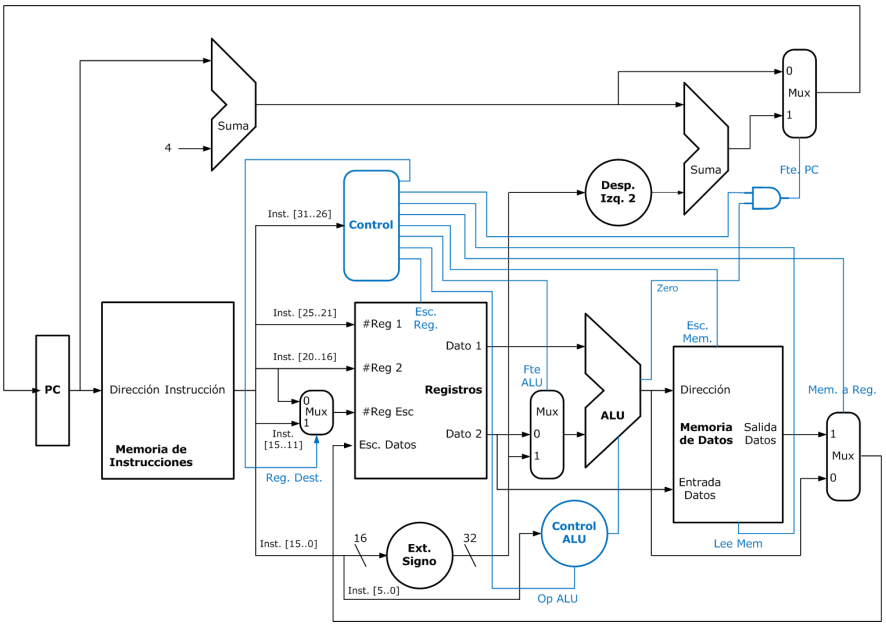
Ruta de Datos y Unidad de Control



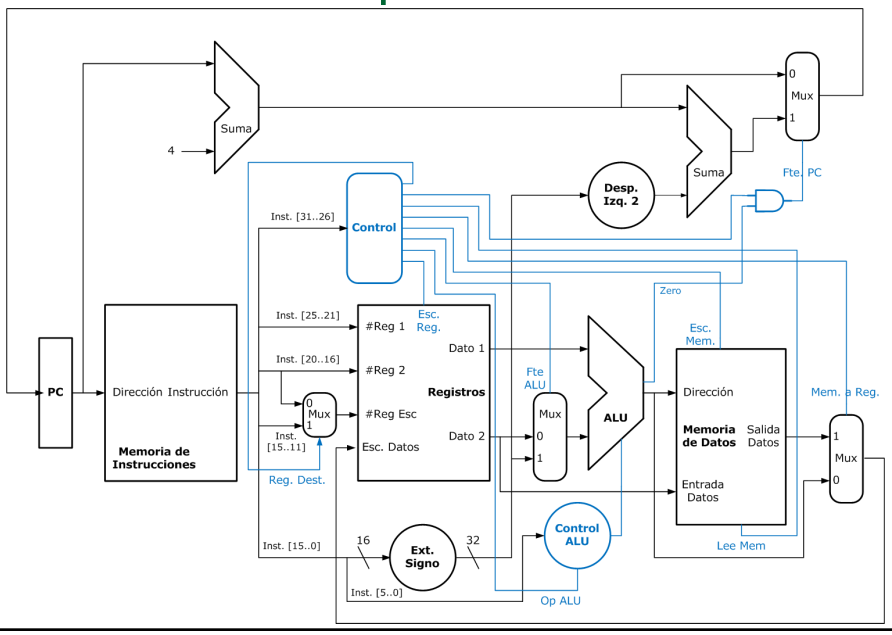
Instrucciones Tipo R



Instrucción lw



Instrucción beq

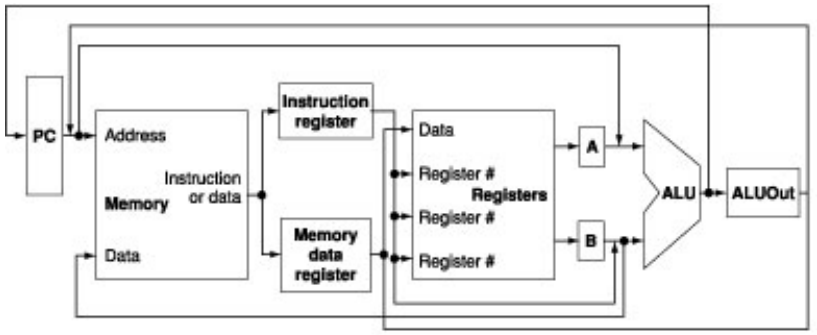


Rendimiento de la Ruta de Datos

- Para determinar el rendimiento consideremos los siguientes tiempos para cada unidad:
  - Unidades de Memoria: 200 picosegundos
  - ALU y sumadores: 100 picosegundos
  - Archivo de Registros (lectura o escritura)
- Asumimos también que los multiplexores, la unidad de control, accesos a PC, unidad de extensión de signo y cables no tienen retardo.

Implementación de un Multiciclo

- En una implementación multiciclo, cada paso en la ejecución dura un ciclo de reloj. Esto permite utilizar una unidad funcional más de una vez dentro de una instrucción, siempre y cuando se use en distintos ciclos de reloj.
- Al compartir estas unidades se puede reducir el hardware.



Implementación de un Multiciclo

- Las principales diferencias que podemos encontrar frente al monociclo son:
  - Se usa una sola unidad de memoria para instrucciones y datos.
  - Hay un solo ALU en lugar de un ALU y dos sumadores.
  - Se agregan uno o más registros después de cada unidad funcional principal para retener el valor que se empleará en el siguiente ciclo de reloj.
- Al final de un ciclo de reloj, todos los datos que se usarán en el siguiente ciclo deben ser almacenado en un elemento de estado.

Implementación de un Multiciclo

- Los datos usados por una siguiente instrucción se deben almacenar en alguno de los elementos de estado visibles al programador: el archivo de registros, la PC o la memoria.
- Los datos usados dentro de una misma instrucción deben ser almacenados en los registros agregados.
- La ubicación de los registros adicionales se deben a:
  - Que unidades combinacionales entrarán en un ciclo de reloj.
  - Que data se necesita en ciclos después implementando la instrucción.

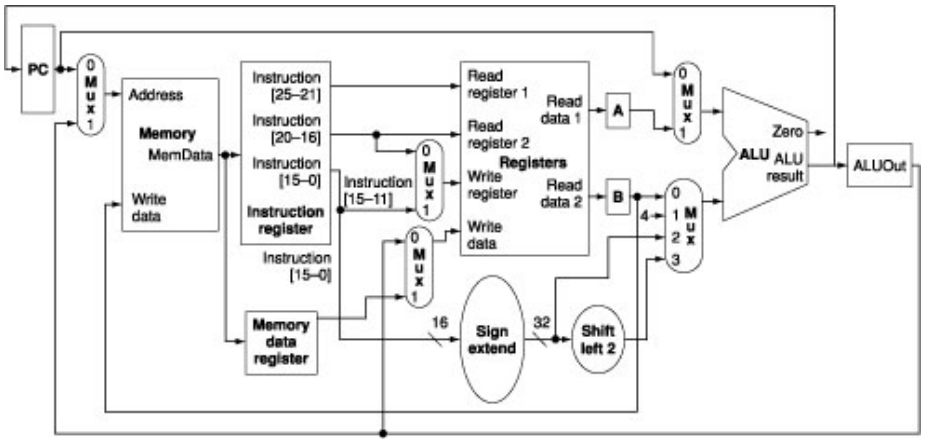
Implementación de un Multiciclo

- Para cumplir estos requerimiento, se añaden los siguientes registros:
  - El registro de instrucción y el registro de memoria de datos, se usan para guardar la salida de memoria de una lectura de instrucción y una lectura de datos respectivamente.
  - Los registros A y B se usan para retener los valores de operandos leídos desde el archivo de registros.
  - El registro ALUOut que retiene la salida del ALU.
- Todos los registros excepto el IR retienen datos solo entre dos ciclos de reloj adyacentes y no necesitará señal de control.

Implementación de un Multiciclo

- Las unidades ahora se usan para diferentes propósitos y por lo tanto se deben agregar multiplexores y a los ya existentes, expandirlos.
- Reemplazar los tres ALUs en uno solo implica que los ALUs deben acomodar sus entradas. Para manejar estas nuevas entradas, se necesitan dos cambios:
  - Un multiplexor adicional para la primera entradas del ALU, para elegir entre el registro A y el PC.
  - El multiplexor de la segunda entrada se cambia de 2x1 a 4x1. Las entradas adicionales son la constante 4 y el campo extendido y desplazado 2 bits.

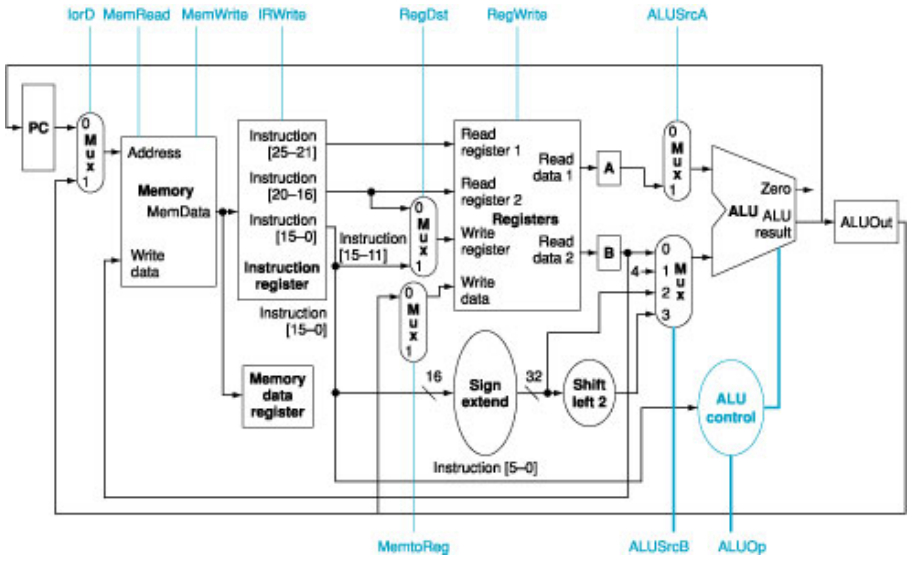
Implementación de un Multiciclo



Implementación de un Multiciclo

- Debido a que esta ruta de datos toma varios ciclos de reloj para cada instrucción, los controles son distintos.
- La unidades de estado (PC, memoria y registros) así como la IR van a necesitar señales de control de escritura.
- Además el nuevo multiplexor de cuatro entradas va a requerir de dos líneas de control.
- Los controles del ALU se mantienen como en el caso de monociclo.

Implementación de un Multiciclo



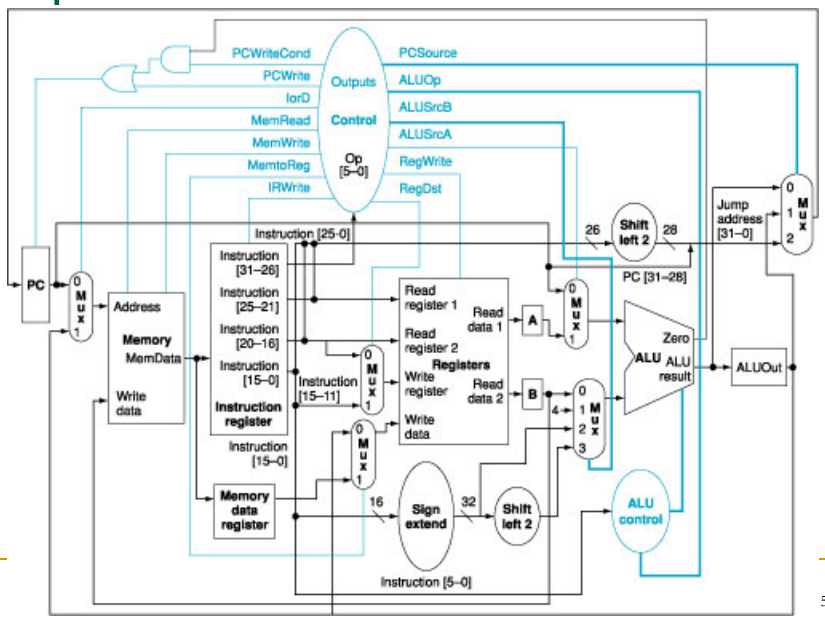
Implementación de un Multiciclo

- Por último, esta ruta de datos necesita el soporte para saltos y ramificaciones.
- Con las intrucciones jump y branch hay tres posibles fuentes para el valor que se escribirá en el PC:
  - La salida del ALU para el fetch. Debe ir directo al PC.
  - El registro ALUOut, donde se guarda la dirección destino de ramificación antes de ser calculada.
  - Los 26 bits menos significativos del IR desplazados en dos y concatenados con los 4 bits más significativos del PC ya incrementado.

Implementación de un Multiciclo

- Como se vio en monociclo, el PC se puede escribir condicional e incondicionalmente:
  - Condicional: Cuando se verifica si la condición de una ramificación es valida.
  - Incondicional: Cuando simplemente se carga el valor de incremento natural, o el determinado por el jump.
- Entonces la implementación empleará dos señales de control:
  - PCWrite, para escritura incondicional en el PC.
  - PCWriteCond, para escritura condicional.
- Entonces la señal de escritura del PC se obtendra de PCWrite, PCWriteCond y la señal Zero de ALU.

Implementación de un Multiciclo



Ejecución en Multiciclo

- Vamos a dividir la ejecución de cada instrucción en una serie de pasos, cada uno tomando un ciclo de reloj. Por ejemplo, se restringira que un paso contenga una operación del ALU, acceso a registro o acceso a memoria.
- Recordemos que al final de un paso hay valores que deben preservarse para el siguiente y quedarán almacenados en alguno de los elementos de estado.
- En un monociclo, el acceso a registros esta contenido dentro del ciclo, sin embargo en este diseño uno de los pasos será dicho acceso.

Ejecución Paso por Paso

1. Fetch de Instrucción: Recoge la instrucción de memoria y calcula la dirección de la siguiente instrucción en secuencia.
   1. Operación: Envia el contenido del PC a la memoria como dirección, realiza la lectura y escribe la instrucción en el IR, donde será almacenado. También incrementa el PC.
   2. Para implementar, necesitamos validar las señales de control MemRead y IRWrite y setear IorD a 0 para seleccionar el PC como la fuente de la dirección. También incrementamos el PC, para lo cual ALUSrcA debe ser 0, ALUSrcB 01 y ALUOp 00. Finalmente vamos setear PC source en 00 y PCWrite.

Ejecución Paso por Paso

2. Decodificación de Instrucción y fetch de registro: Podemos leer los dos registros como rs y rt; y almacenarlos en los registros temporales A y B. También calculamos la dirección de ramificación con el ALU, guardando el resultado en ALUOut.
   1. Operación: Acceso al archivo de registros para leer rs y rt, guardando en A y B. Se calcula la dirección de ramificación y se guarda en ALUOut.
   2. Para implementar,fijamos ALUSrcA en 0, ALUSrcB en 11 y ALUOp en 00.

Ejecución Paso por Paso

3. Ejecución, calculo de dirección de memoria o finalización de ramificación:.
   1. Referencia de Memoria: Para implementar, vemos que el ALU suma los operandos para formar la dirección de memoria. Se necesita que ALUSrcA=1 y ALUSrcB=10. ALUOp debe ser 00.
   2. Instrucción Aritmético Lógica: Para implementar, el ALU realiza la operación que se solicita (ALUOp=01), con ALUSrcA=1 y ALUSrcB=0.
   3. Bifurcación: El ALU compara los registros, con la señal Zero para determinar el salto (ALUOp=01). ALUSrcA=1 y ALUSrcB=00.

Ejecución Paso por Paso

4. Acceso a memoria o finalización de instrucción R: En este paso, una instrucción load o store acede a memoria y una instrucción aritmético lógica escribe el resultado.
   1. Referencia de memoria: Para load o store la dirección es la calculada en el paso anterior y almacenada en ALUOut.
   2. Instrucción Aritmético Lógica: Coloca el valor de ALUOut en el registro Result.
5. Finalización de lectura en memoria:
   1. Escribe el dato cargado, que fue almacenado en MDR en el ciclo anterior, en el banco de registros.