Estructura de computadores

Unidad 5. El Procesador III: Diseño y control de la ruta de datos: Arquitectura multiciclo

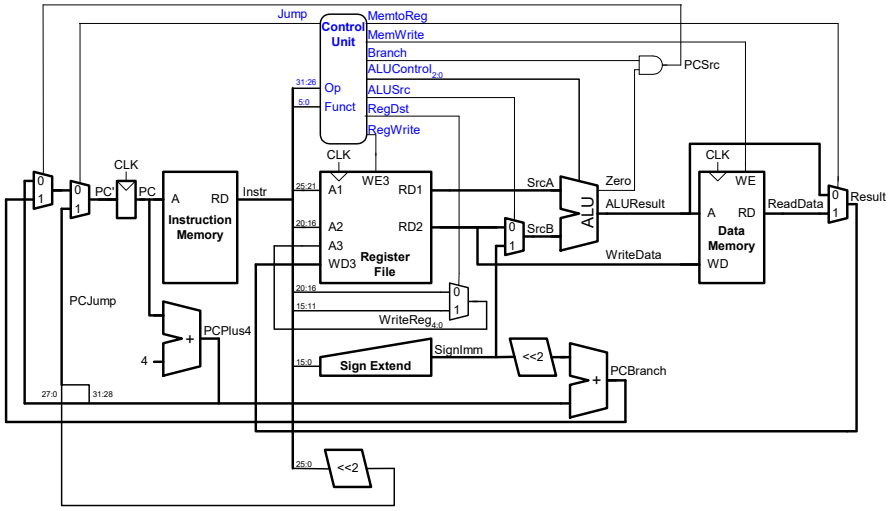
Escuela Politécnica Superior - UAM

Copyright © 2007 Elsevier, "Digital Design and Computer Architecture"

Índice

- **Resumen arquitectura MIPS uniciclo**
- Ruta de datos multiciclo
- Control multiciclo
- Añadir más instrucciones

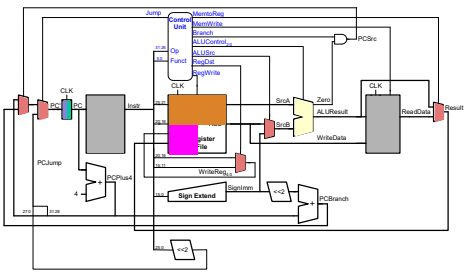
Resumen: MIPS uniciclo



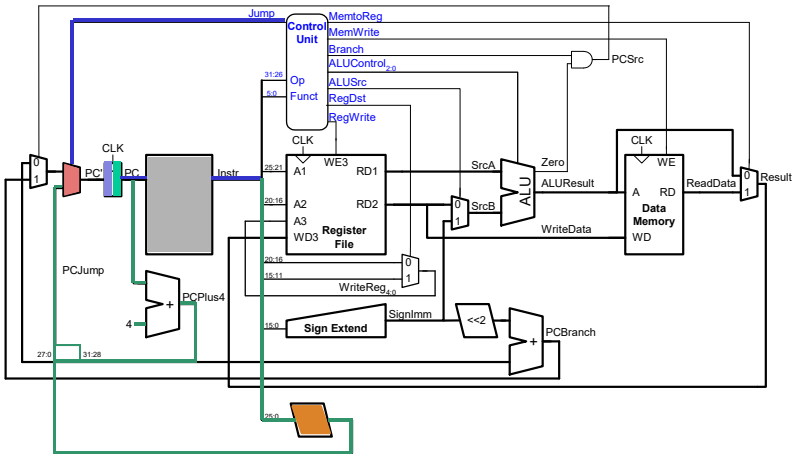
Parámetros temporales: Ciclo de reloj en uniciclo

- Todas las instrucciones utilizan el mismo ciclo de reloj, que tendrá que ser suficiente para la más lenta de todas.

Elemento	Parámetro	Retardo (ps)
$T_{CLK \rightarrow Q}$ Registro	t_{pcq_PC}	30
Leer de Memoria	t_{mem}	250
Leer del BancoReg	t_{RFread}	150
Multiplexor	t_{mux}	25
ALU	t_{ALU}	200
T_{SETUP} BancoReg	$t_{RFsetup}$	20
T_{SETUP} Registro	t_{setup}	20



Parámetros temporales: Ciclo de reloj en uniciclo. Ejemplo: camino crítico para ***jump(j)***



Camino crítico para la instrucción *jump* (j):

$T_c = t_{pcq\_PC} + t_{mem} + t_{<<} + t_{mux} + t_{PCsetup} =$

$= [30 + 250 + 150 + 25 + 20]$ ps = 475 ps (~2,1 GHz)

5

MIPS uniciclo vs multiciclo

	t_{pcq_PC}	t_{mem}	$T_{<<}$	t_{RFread}	t_{mux}	t_{ALU}	t_{mem}	t_{mux}	$t_{RFsetup}/t_{PCsetup}$	T_{total} (ps)
add	30	250	--	150	25	200	--	25	20/20	700 ps
lw	30	250	--	150	--	200	250	25	20/20	**925 ps**
j	30	250	150	--	--	--	--	25	20	475 ps

- Microarquitectura uniciclo:
  - \+ Simple
  - \- Ciclo de reloj limitado por instrucción más lenta (**lw**)
  - \- 3 sumadores/ALUs y 2 memorias (código y datos)
- Microarquitectura multiciclo:
  - \+ Reloj más rápido
  - \+ Instrucciones sencillas van más rápido (menos ciclos)
  - \+ Reutilizar hardware de aquellos elementos que antes se utilizaban en el mismo ciclo y ahora se utilizarán en distintos ciclos de reloj)
  - \- Si no se divide en etapas del mismo retraso se pierden prestaciones

6

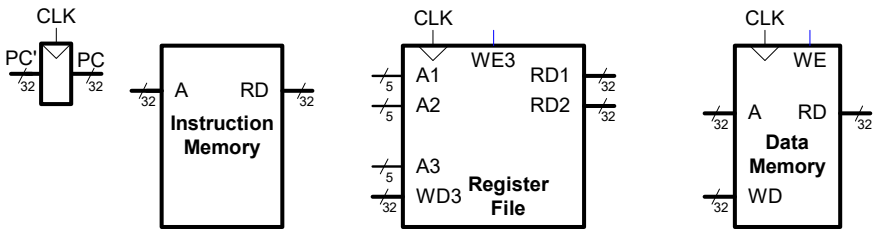
Índice

- Resumen arquitectura MIPS uniciclo
- **Ruta de datos multiciclo**
- Control multiciclo
- Añadir más instrucciones

7

Repaso uniciclo. Estado de la arquitectura

- Se puede conocer en qué situación se encuentra el micro conociendo los valores de:
  - PC
  - Banco de registros (los 32 registros)
  - Memoria (de código y de datos)
- Primeros elementos a considerar en la ruta de datos:



8

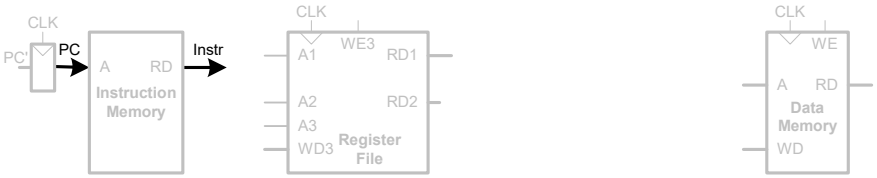
Repaso uniciclo.
Ruta de datos: captura `lw`

El análisis de la ruta de datos comienza con la instrucción:

(**0x8C112000**) `lw $s1,0x2000($0)`

y los pasos para ejecutarla:

- **PASO 1:** captura de instrucción (*fetch*)



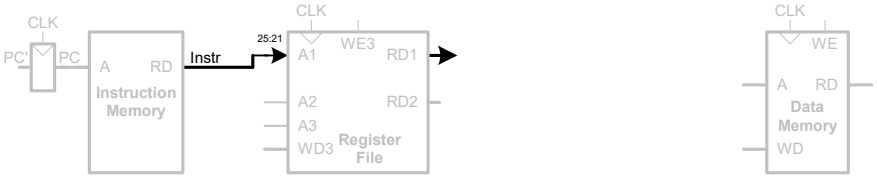
Instr <= "100011 00000 10001 0010000000000000"

op: Instr [31:26] =	"100011"	=>	`lw`
rs: Instr [25:21] =	"00000"	=>	`$0`
rt: Instr [20:16] =	"10001"	=>	`$s1`
imm: Instr [15:0] =	"0010000000000000"	=>	`0x2000`

9

Repaso uniciclo.
Ruta de datos: lectura de registros `lw`

- **PASO 2:** lectura de los operandos fuente del banco de registros

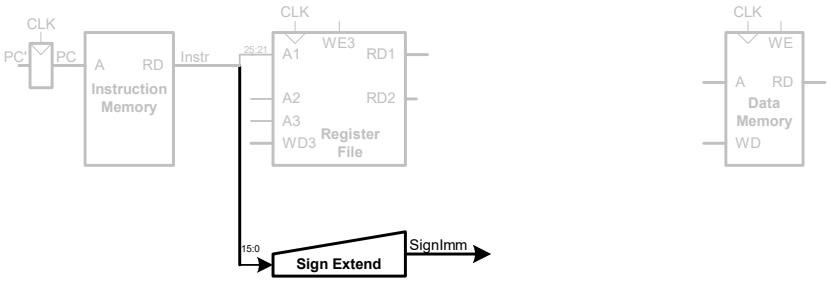


A1: Instr [25:21] = "00000"; RD1 <= "0x00000000" = ($0)

10

Repaso uniciclo.
Ruta de datos: dato inmediato lw

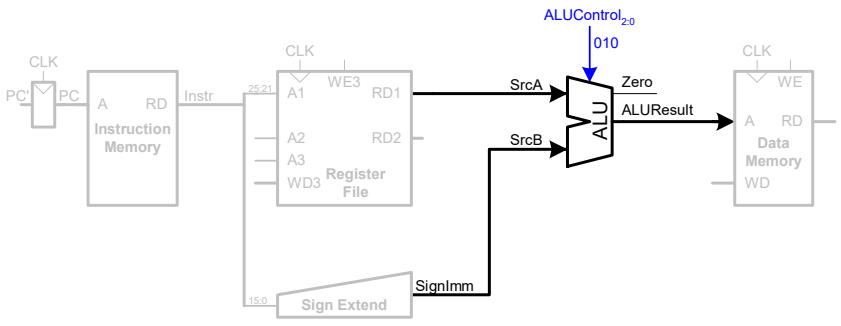
- **PASO 3:** extensión en signo del dato inmediato



SignImm <= 0x00002000 = Sign Extend (0x2000)

Repaso uniciclo.
Ruta de datos: dirección lw

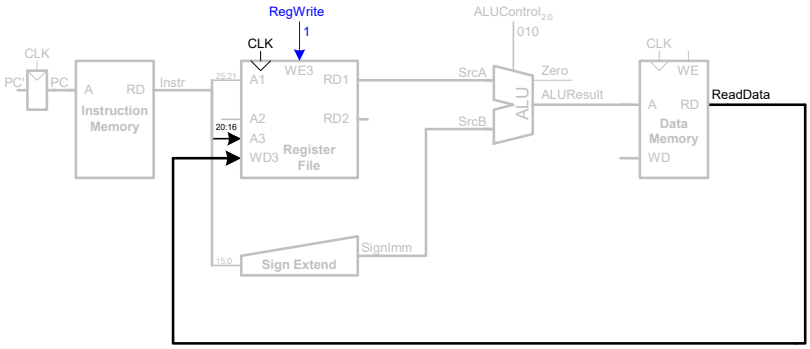
- **PASO 4:** calcular la dirección para acceso a memoria de datos sumando ([rs] + SignImm) en la ALU



ALUResult <= 0x00002000 = 0x00000000 + 0x00002000

Repaso uniciclo.
Ruta de datos: leer memoria lw

- **PASO 5:** leer el dato buscado de memoria y escribirlo en el registro destino, rt

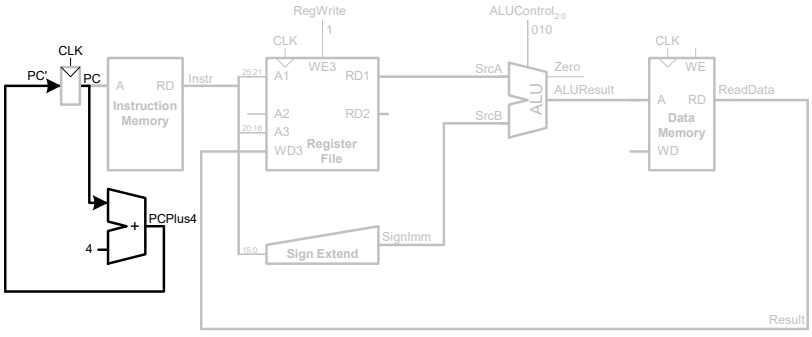


A3: Instr [20:16] = “10001” ($s1)

$s1 <= WD3 = MEM[0x00002000]

Repaso uniciclo.
Ruta de datos: incrementar PC

- **PASO 6:** incrementar el PC en 4 para tener la dirección de la próxima instrucción

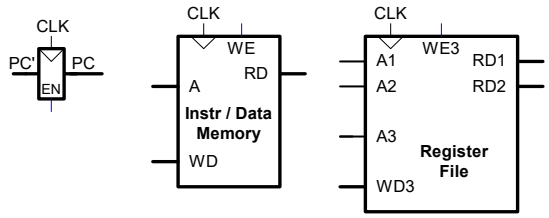


$PC <= $PC + 4

Elementos de estado (memorias)

Las memorias de instrucciones y de datos se juntan en una sola memoria.

– Se accede a cada elemento del sistema en ciclos distintos del reloj.



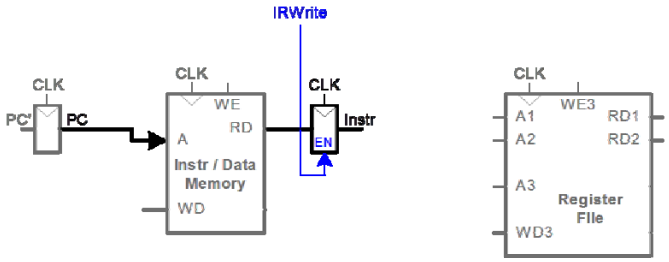
15

Ruta de datos: captura lw

De nuevo empezamos el análisis de la ruta de datos con la instrucción lw

➢ **CICLO 1:** captura de instrucción (*fetch*).

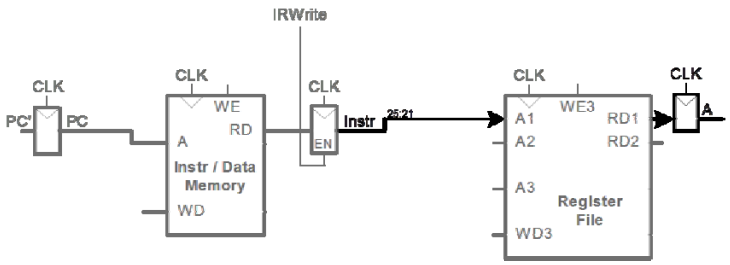
- Cada paso es un ciclo de reloj, pero mucho más corto que en el caso uniciclo (en un ciclo se ejecutaba todo).
- Ahora el resultado de cada paso se registra (registro **Instr**) habilitando la escritura (IRWrite, *enable*) en el ciclo de reloj en que se dispone de la información.



16

Ruta de datos: lectura de registros lw

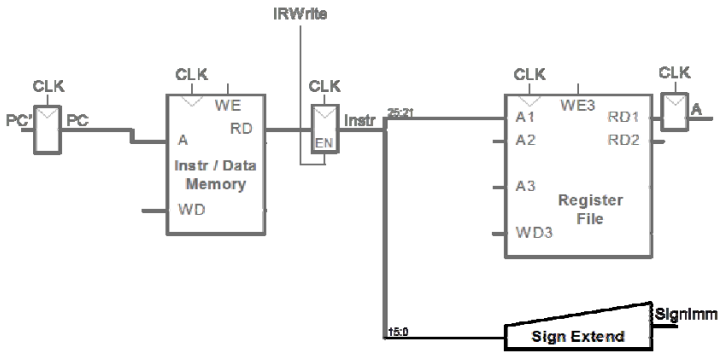
- **CICLO 2:** leer los operandos fuente del banco de registros.
  - El resultado (operando 1) se guarda en un nuevo registro (**A**). No tiene habilitación de escritura, así que se actualiza todos los ciclos (rs no cambia durante la instrucción).



17

Ruta de datos: dato inmediato lw

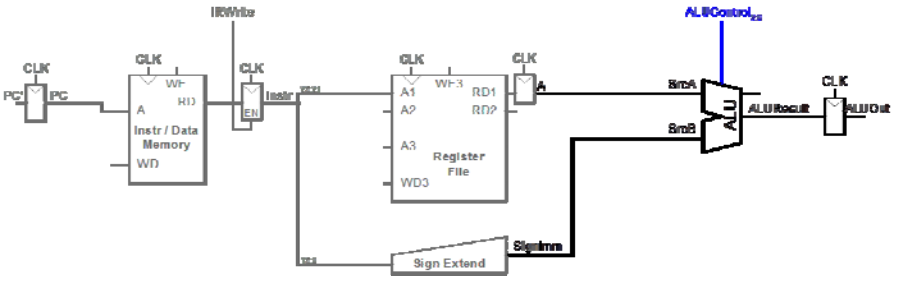
- **CICLO 2:** se hace en paralelo a la lectura del banco de registros, durante el ciclo 2.
  - No hace falta guardarlo en registro porque el dato inmediato ya está registrado en Instr.



18

Ruta de datos: dirección lw

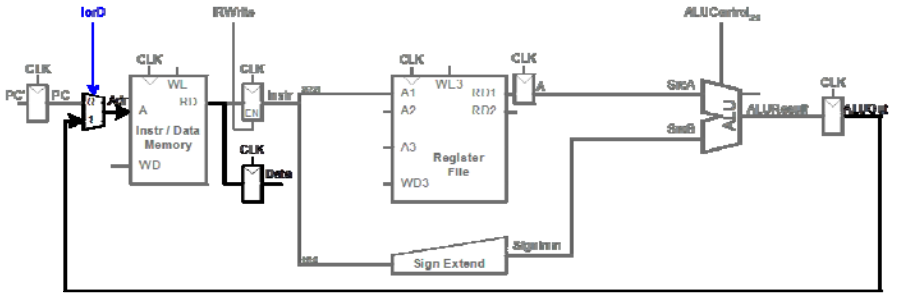
- **CICLO 3:** se calcula la dirección del dato sumando en la ALU registro y dato inmediato.
  - Se guarda el resultado en un nuevo registro (**ALUOut**), antes de cambiar de ciclo de reloj.



19

Ruta de datos: leer memoria lw

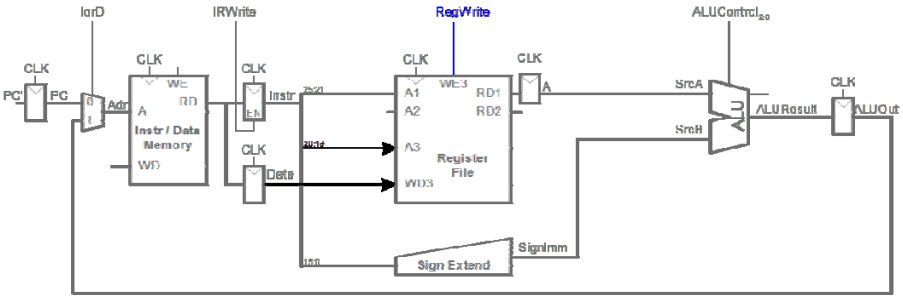
- **CICLO 4:** se usa la dirección calculada para leer el dato de memoria: nueva señal de control IorD (instrucción o dato).
  - Puede ser la misma memoria que la de código porque durante este ciclo de reloj no hay captura de instrucción.
  - El resultado hay que guardarlo en un nuevo registro (**Data**).



20

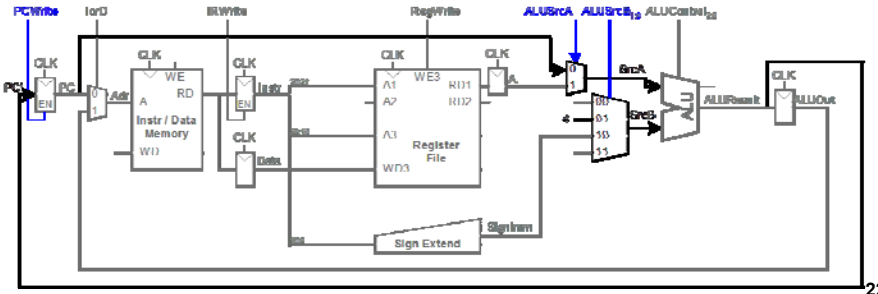
Ruta de datos: guardar registro lw

- **CICLO 5:** el dato leído se escribe en el registro destino.
  - No hay registro adicional, porque ya se guarda el resultado en el propio banco de registros.



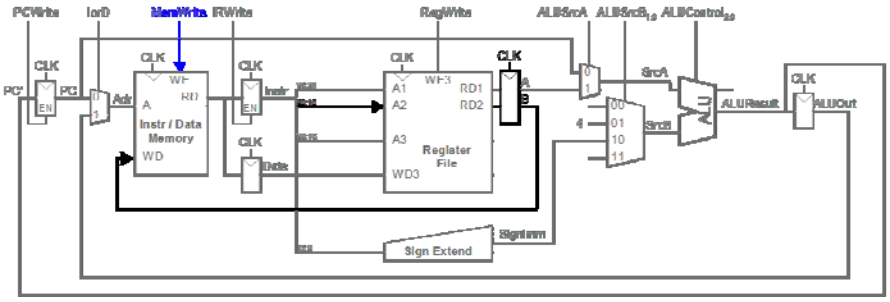
Ruta de datos: incrementar PC lw

- **CICLO 1:** durante el primer ciclo de cada instrucción se guarda el nuevo PC (provisional en los saltos) para la próxima instrucción, que es PC + 4.
  - En vez de utilizar un sumador específico para sumar 4 se usa la ALU, que no hace nada más durante este ciclo de reloj.
  - Requiere que PC y 4 lleguen a los dos operandos de la ALU (más multiplexores).



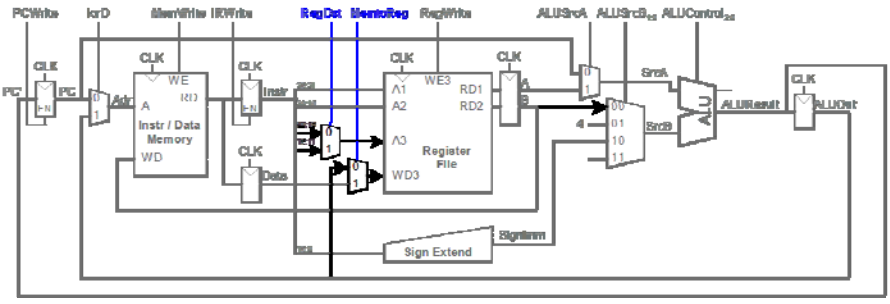
Ruta de datos: sw

- **Ciclo 2:** El operando tiene que ser registrado (registro **B**) porque la lectura de registro y la escritura en memoria se ejecutan en ciclos distintos.
- **Ciclo 3:** La dirección de memoria se guarda en (**ALUOut**)**.**
- **Ciclo 4:** El registro rt (ahora en B) se escribe en memoria (MemWrite), así que debe llegar a la entrada WD de memoria.



Ruta de datos: Tipo-R

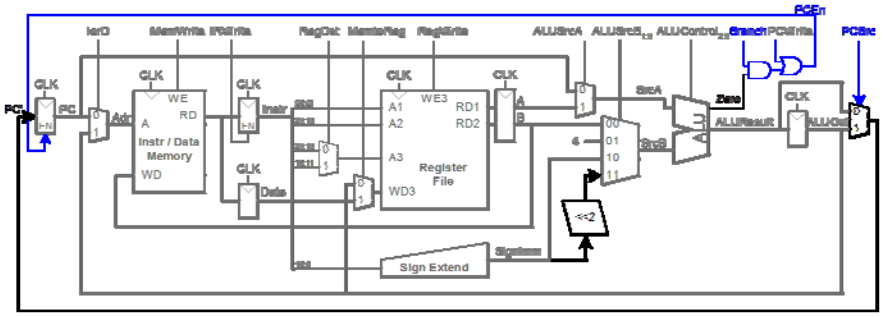
- **Ciclo 2:** Leer y registrar rs y rt, los dos operandos de entrada de la ALU.
- **Ciclo 3:** El resultado ALUResult, se registra en *ALUOut.*
- **Ciclo 4:** Ahora el contenido de ALUOut se escribe en el banco de registros (MemtoReg=‘0’). Se escribe en rd en lugar de rt (RegDst=‘1’).



Ruta de datos: beq

Cálculo de la dirección de salto: BTA = (PC+4) + (Sign Imm << 2)

- **Ciclo 1:** PC <= PC+4 (en la ALU), habilita PC (PCWrite='1') .
- **Ciclo 2:** ALUOut <= (PC+4) + (Sign Imm << 2) (en la ALU).
- **Ciclo 3:** Se restan los operandos para determinar Z. ALUOut se envía a PC (PCSrc='1') por si se produce el salto (Branch AND Z = '1').

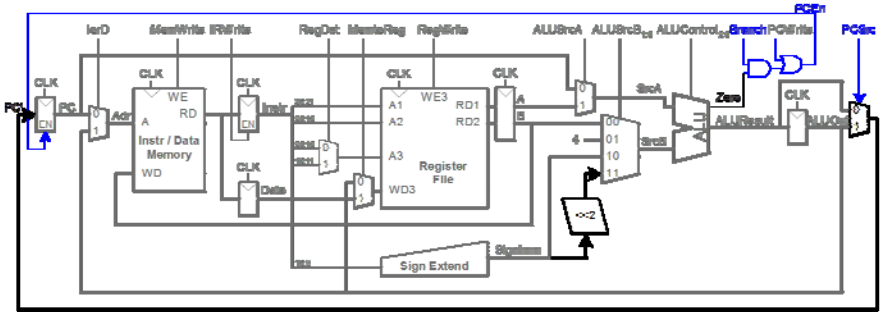


25

Ruta de datos: beq (decisión)

- Se decide si se salta o no con la bandera Z (Zero).
- Se actualiza el PC en dos casos:
  - ✓ PCWrite='1', cada ciclo 1 para conseguir PC+4 (**siempre**)
  - ✓ Zero and Branch (Z='1' y es un salto condicional).

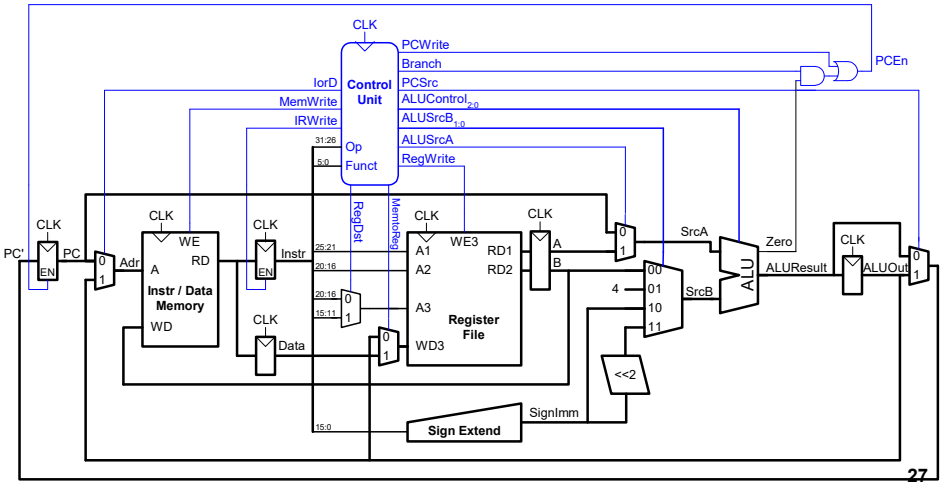
    Branch se debe generar en el ciclo 3 de beq, y si finalmente se salta o no depende de Zero.



26

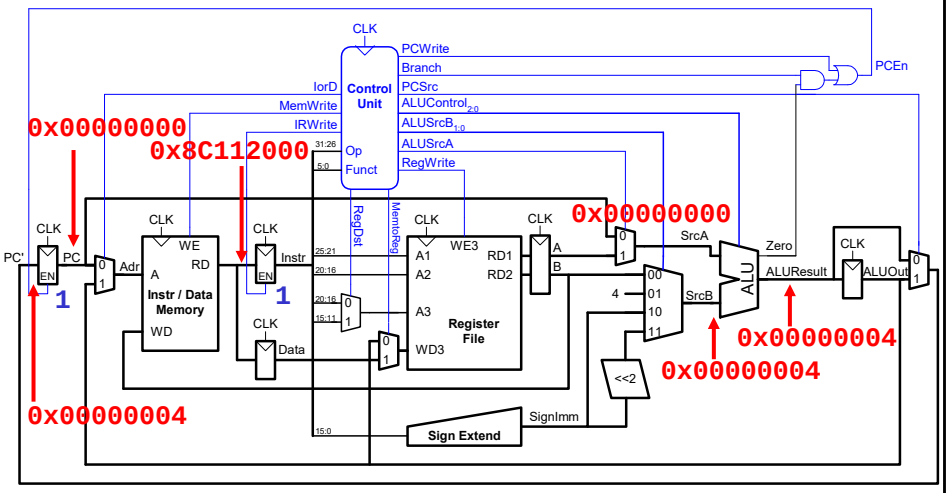
Ruta de datos y de control multiciclo

En la arquitectura multiciclo, las señales de control se generan según la instrucción (hay que decodificar opcode y funct) **y varían según el ciclo de reloj**.



27

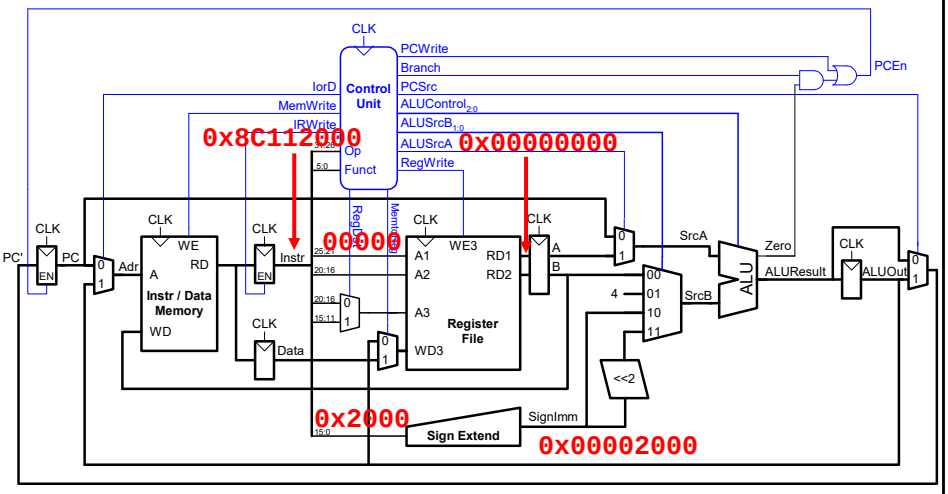
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 1



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

28

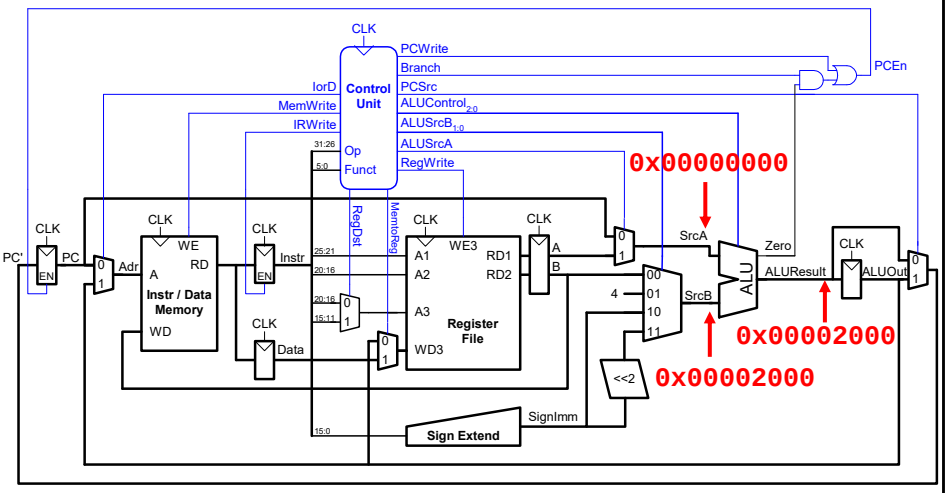
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 2



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

29

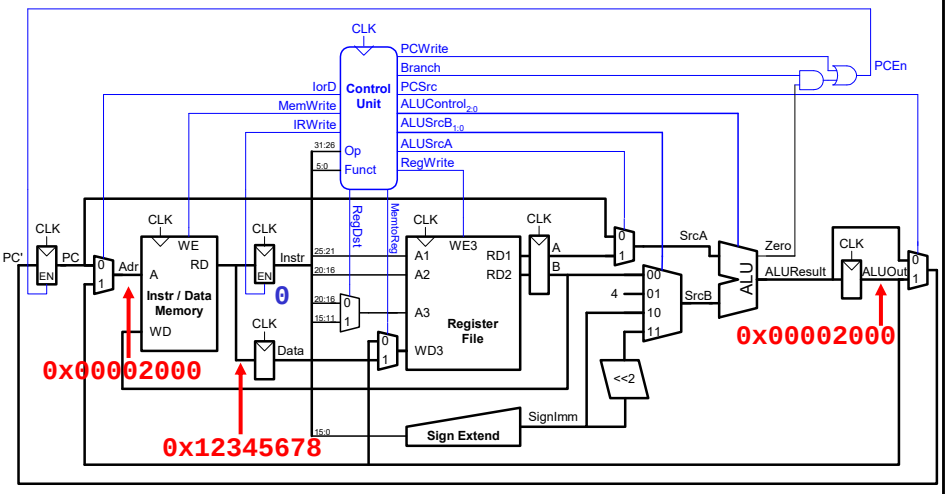
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 3



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

30

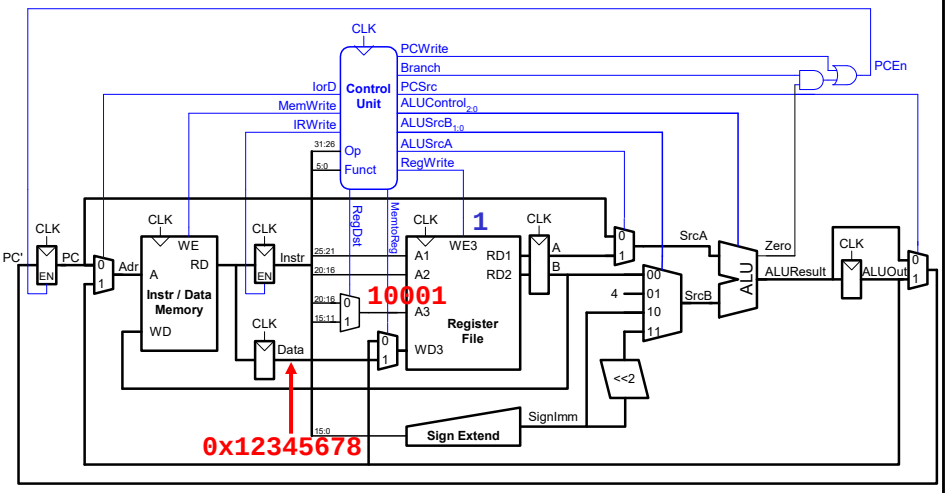
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 4



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

31

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 5



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

32

Índice

- Resumen arquitectura MIPS uniciclo
- Ruta de datos multiciclo
- **Control multiciclo**
- Añadir más instrucciones

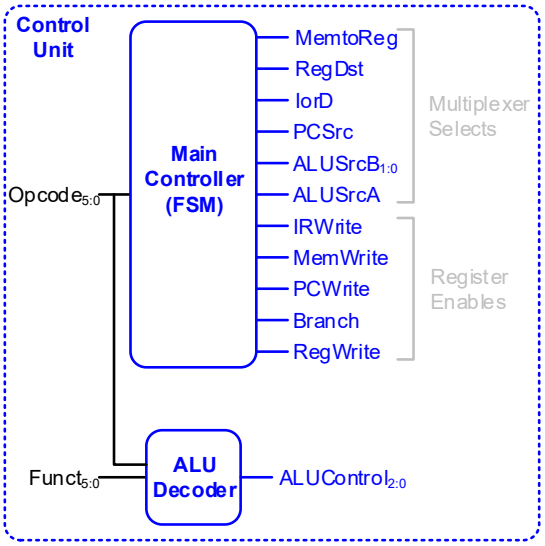
33

Planificación de instrucciones

<table>
<tr><th></th><th>lw</th><th>sw</th><th>R-type</th><th>beq</th></tr>
<tr><td>Ciclo 1</td><td colspan="4">Captura de la instrucción. PC <= PC + 4</td></tr>
<tr><td>Ciclo 2</td><td colspan="3">Lectura operandos.</td><td>a. Lectura operandos.
b. Cálculo BTA.
(no escribe en PC)</td></tr>
<tr><td>Ciclo 3</td><td colspan="2">Cálculo de dirección de Mem. Datos</td><td>Cálculo ALU</td><td>a. Resta en ALU
b. Si Z=1, actualiza PC.</td></tr>
<tr><td>Ciclo 4</td><td>Lectura
Mem. Datos</td><td>Escritura
Mem. Datos</td><td>Escritura
Reg.</td><td></td></tr>
<tr><td>Ciclo 5</td><td>Escritura
Reg.</td><td colspan="3"></td></tr>
</table>

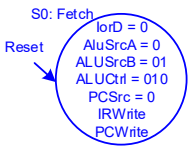
34

Unidad de control

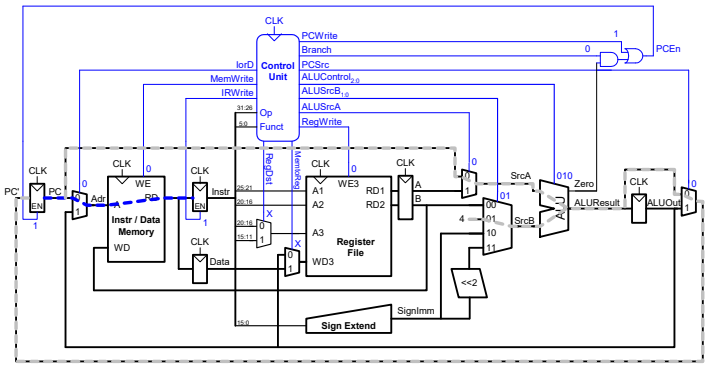


- Como hay que ir variando las señales de control según el ciclo de reloj, ya no puede ser combinacional.
- La parte de ALU Decoder sigue igual. Si se quiere que la ALU haga varias operaciones durante una instrucción se cambia ALUControl en el ciclo que corresponda.
- ***Main Controller*** es una máquina de estados finita (FSM).

Máquina de estados del control (ciclo 1)

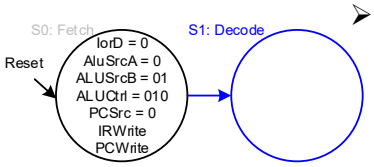


- **Ciclo-1:** es igual para todas las instrucciones.
  - ✓ Captura de la instrucción desde memoria.
  - ✓ Actualización de PC con PC+4 (ALU suma).

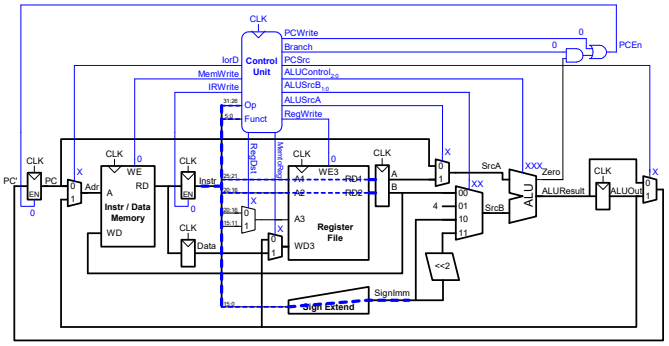


Nota: sólo se destacan las señales de selección de mux relevantes y los enables de registros si se activan (se ponen a '1').

Control: ciclo 2. Todas

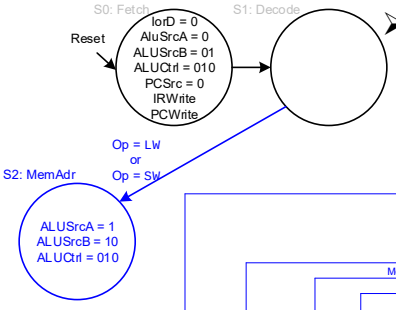


- **Ciclo-2:** Se decodifica la instrucción y se capturan operandos.
  - ✓ Como el primero, este ciclo es igual para todas las instrucciones. Se registran A y B (aunque no se vayan a usar). No tienen enable, así que siempre se registran.

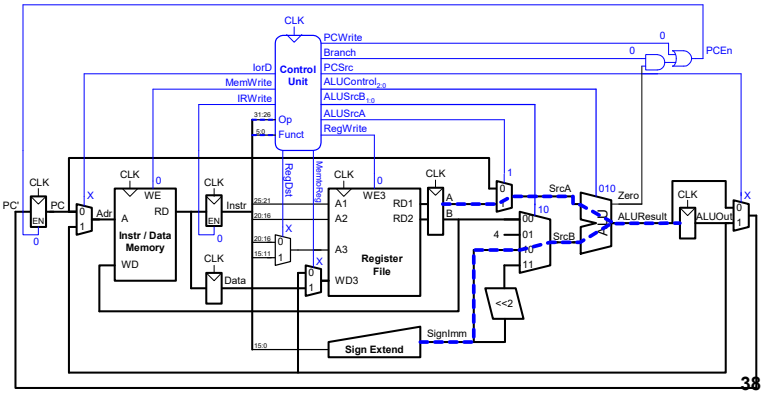


✓A partir de aquí, la evolución de la máquina es distinta para cada instrucción.

Control: ciclo 3, instrucciones lw/sw



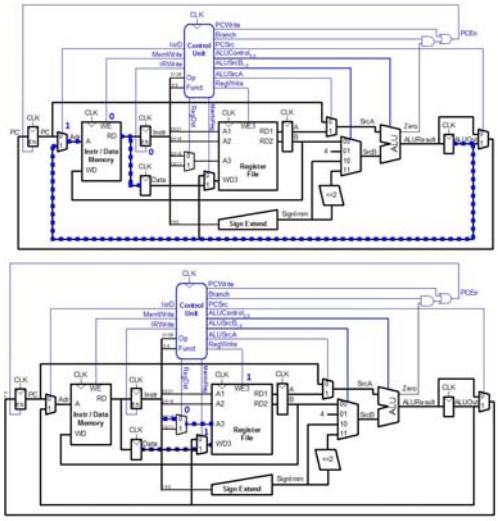
- **Ciclo-3** (para lw o sw): se calcula la dirección de memoria de datos, que se guarda en ALUOut (no tiene enable, siempre se guarda).



Control: ciclos 4 y 5, instrucción lw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria para leer el dato (IorD='1'), y se guarda el dato leído en Data (sin enable, siempre guarda).
- **Ciclo-5:** el dato (Data) se guarda en registro (registro rt es el destino).

Final de instrucción (vuelta a S0).

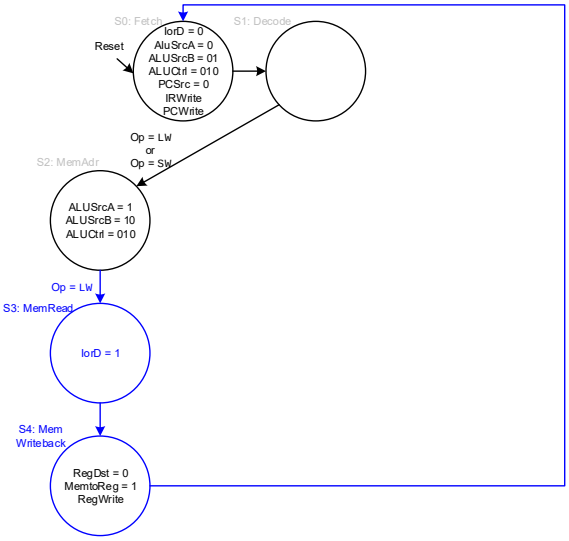


39

Control: ciclos 4 y 5, instrucción lw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria para leer el dato (IorD='1'), y se guarda el dato leído en Data (sin enable, siempre guarda).
- **Ciclo-5:** el dato (Data) se guarda en registro (registro rt es el destino).

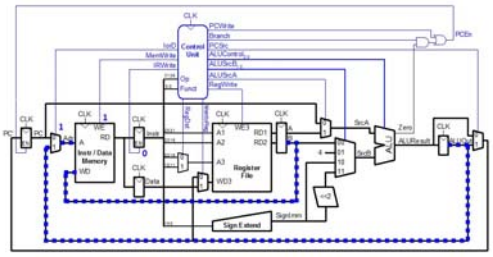
Final de instrucción (vuelta a S0).



40

Control: ciclo 4, instrucción sw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria (IorD='1') para escribir el dato. La memoria se activa para escritura. El dato escrito es rt, guardado en B.



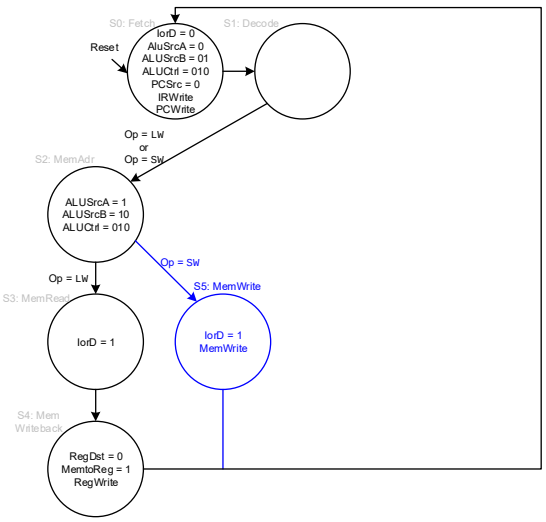
Final de instrucción (vuelta a S0).

La instrucción sw sólo necesita 4 ciclos, no 5.

41

Control: ciclo 4, instrucción sw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria (IorD='1') para escribir el dato. La memoria se activa para escritura. El dato escrito es rt, guardado en B.



Final de instrucción (vuelta a S0).

La instrucción sw sólo necesita 4 ciclos, no 5.

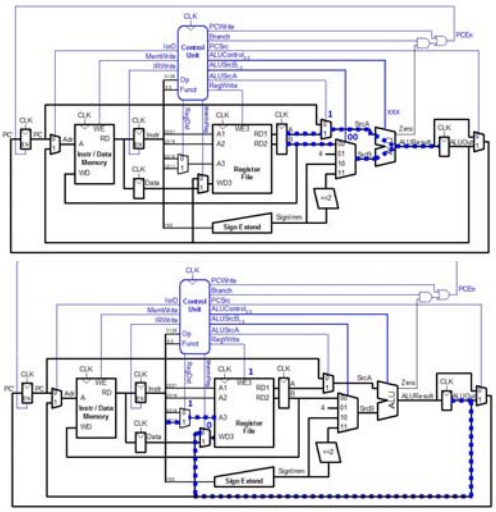
42

Control: ciclos 3 y 4, instrucciones Tipo-R

- **Ciclo-3:** se obtiene el resultado en la ALU. Se usan A y B (datos de registros) y la operación de la ALU depende de funct.
- **Ciclo-4:** el resultado, que ha quedado en ALUOut, se manda al registro rd.

  Final de instrucción (vuelta a S0).

  Se ejecutan en 4 ciclos.



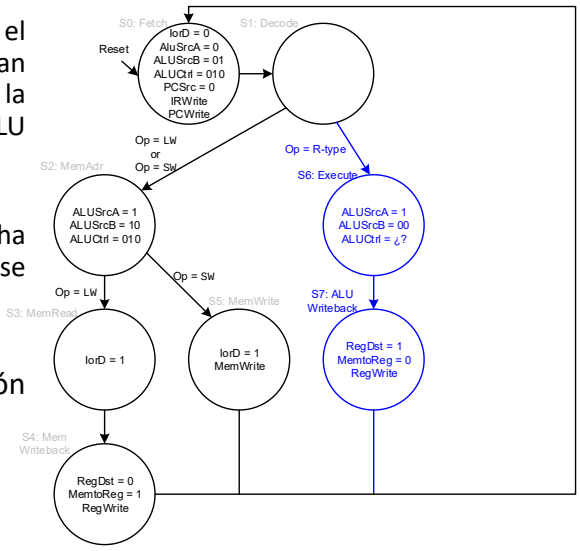
43

Control: ciclos 3 y 4, instrucciones Tipo-R

- **Ciclo-3:** se obtiene el resultado en la ALU. Se usan A y B (datos de registros) y la operación de la ALU depende de funct.
- **Ciclo-4:** el resultado, que ha quedado en ALUOut, se manda al registro rd.

  Final de instrucción (vuelta a S0).

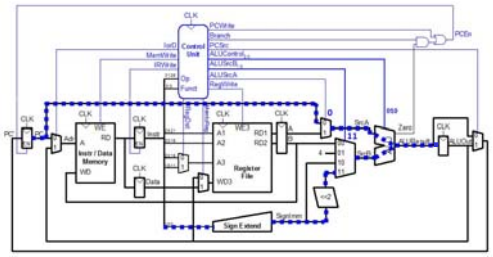
  Se ejecutan en 4 ciclos.

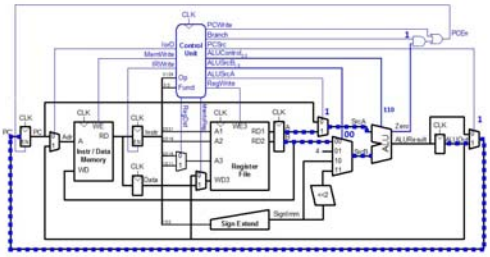


44

Control beq, ciclo 2

- **Ciclo 1:** la ALU calcula PC+4
- **Ciclo-2:**, se calcula la dirección de salto: SrcA=0 y SrcB=11. Se hace en **todas** las instrucciones, y si no es un salto, simplemente se descarta ALUOut, producido en ciclo 2.
- **Ciclo 3:** se activa Branch. En la ALU se restan los registros a comparar. De esta forma se activa o no la bandera Z, y por tanto se activará o no PCEn.





45

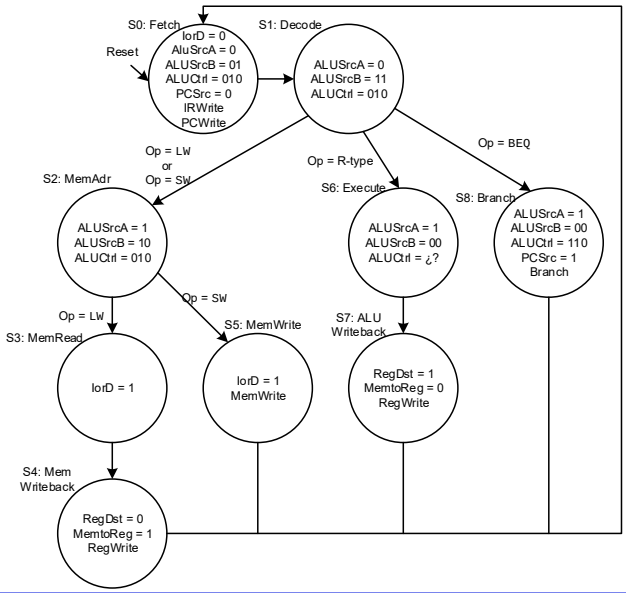
Control beq, ciclo 3

✓Sólo se usan 3 ciclos, a pesar de haber 3 usos de la ALU.

✓Si finalmente hay salto, PCEn estará activa. Hay que actualizar PC con ALUOut, dirección del salto calculada en ciclo 2.

46

Resumen: control multiciclo (sin instrucciones añadidas)



47

Planificación de instrucciones

	lw	sw	R-type	beq
Ciclo 1	Captura de instrucción. PC <= PC + 4			
Ciclo 2	**a.** Lectura operandos. **b.** Cálculo BTA (no se guarda en PC)			
Ciclo 3	Cálculo de dirección de Mem. Datos		Cálculo ALU	**a.** Resta en ALU **b.** Si Z=1, actualiza PC.
Ciclo 4	Lectura Mem. Datos	Escritura Mem. Datos	Escritura Reg.	
Ciclo 5	Escritura Reg.			

48

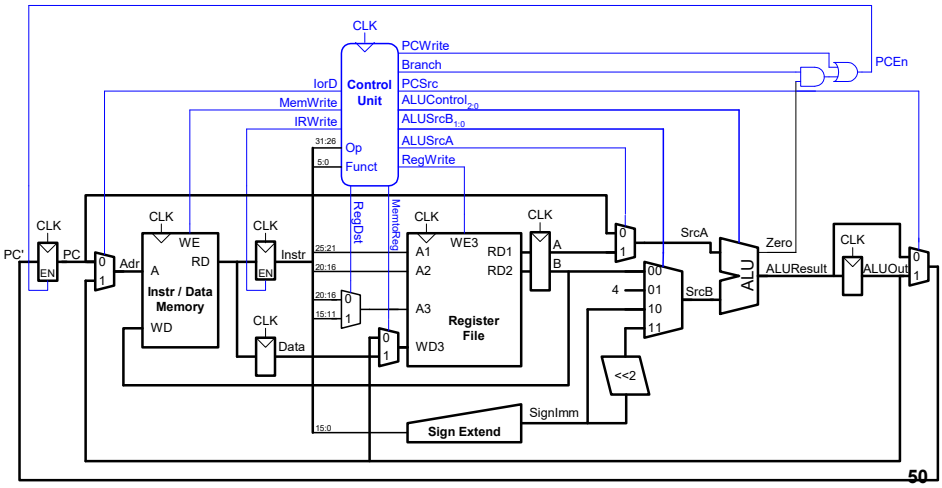
Índice

- Resumen arquitectura MIPS uniciclo
- Ruta de datos multiciclo
- Control multiciclo
- **Añadir más instrucciones**

49

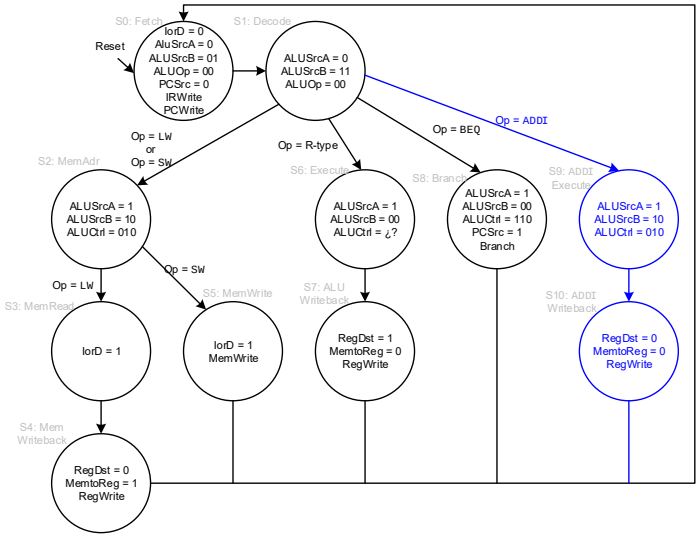
Añadimos addi

- La ruta de datos no necesita cambios (se puede sumar un registro y un dato inmediato como en lw, sw). Sólo hay cambios en el control.



50

Control: cambios para addi



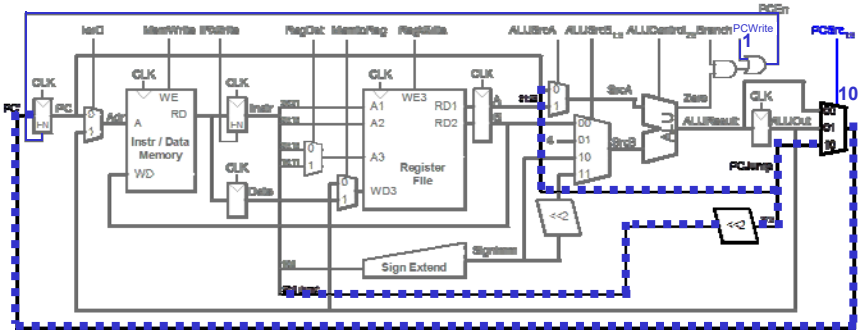
51

Añadimos j

- La ruta de datos sí necesita cambios: hay que poder generar JTA.
  JTA = (PC+4)[31:28] & addr & “00”.

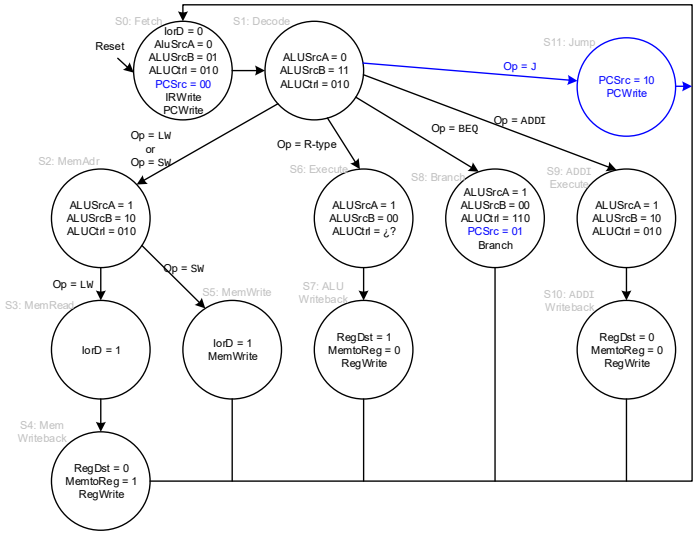
  En **Ciclo-1** PC <= (PC+4)

➢ **Ciclo-3:** Se concatena addr con dos ceros. PCSrc se convierte en una señal de dos bits, porque hay tres señales a elegir.



52

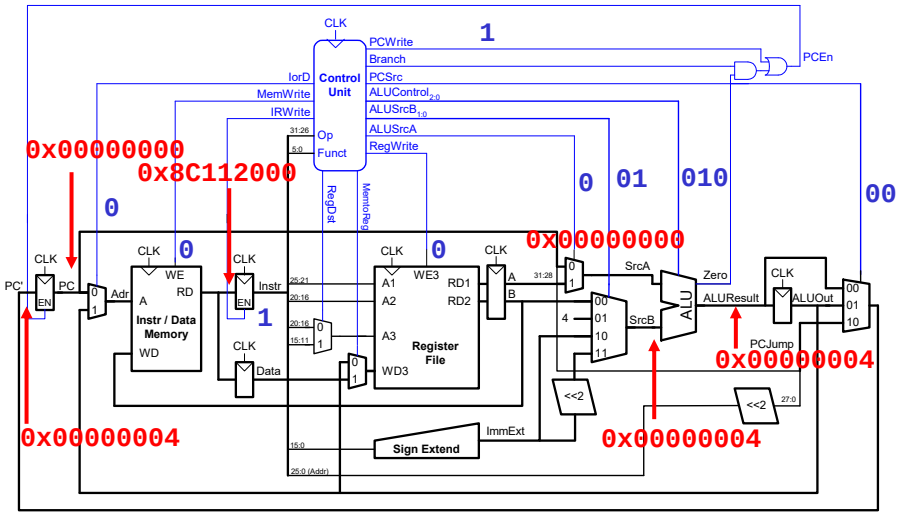
Control: cambios para j



Planificación de instrucciones

	lw	sw	R-type	beq	addi	j
Ciclo 1	Captura de instrucción. PC <= PC + 4					
Ciclo 2	Lectura operandos. Cálculo BTA (no se guarda en PC)					
Ciclo 3	Cálculo de dirección de Mem. Datos		Cálculo ALU	Resta en ALU. Salta si Z=1.	Cálculo ALU	Salta a JTA
Ciclo 4	Lectura Mem. Datos	Escritura Mem. Datos	Escritura Reg.		Escritura Reg.	
Ciclo 5	Escritura Reg.					

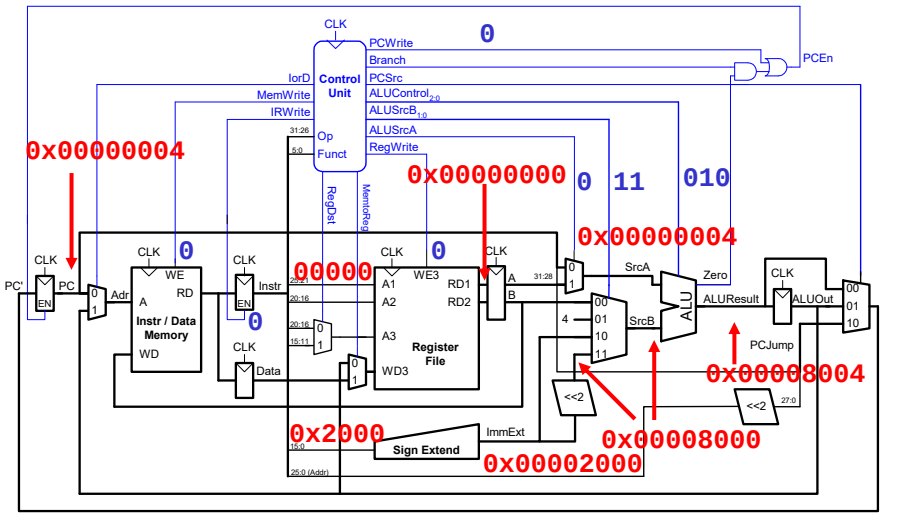
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 1



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

55

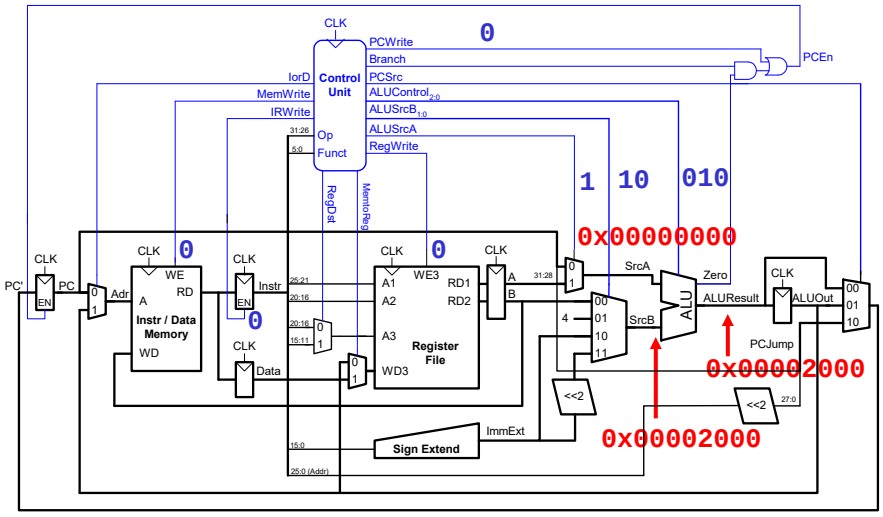
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 2



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

56

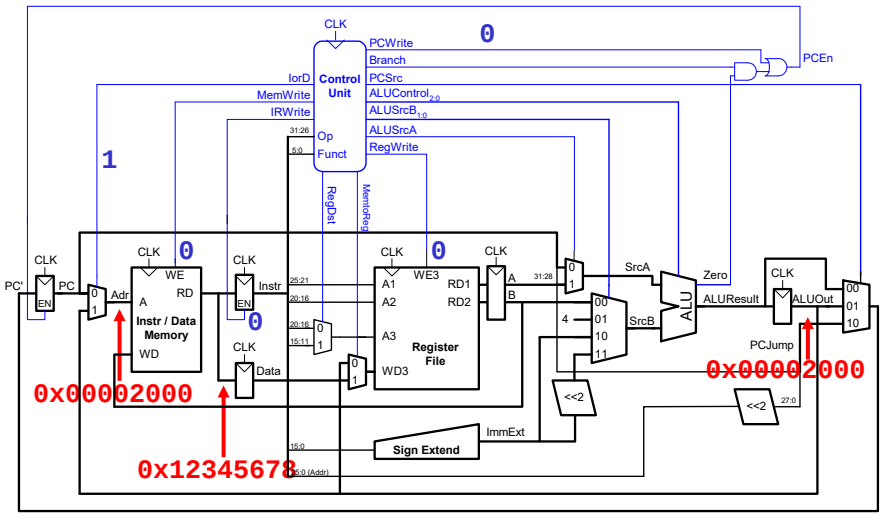
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 3



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

57

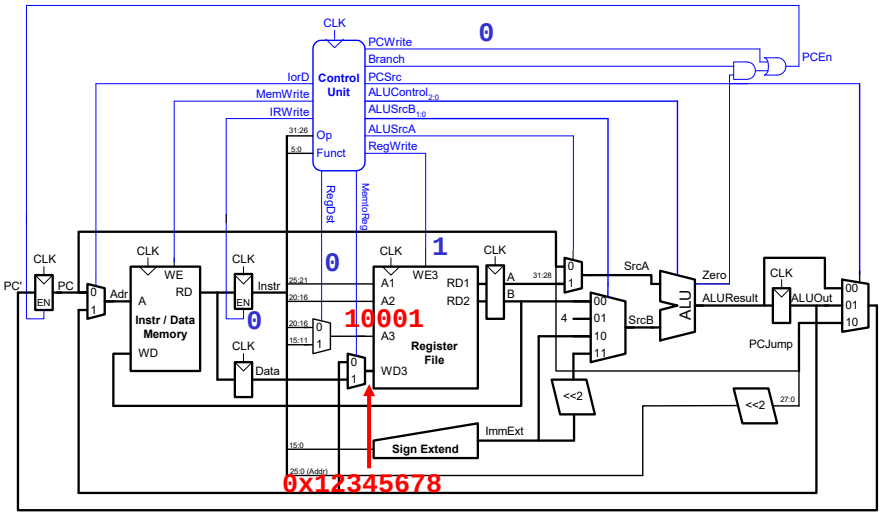
Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 4



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

58

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 5



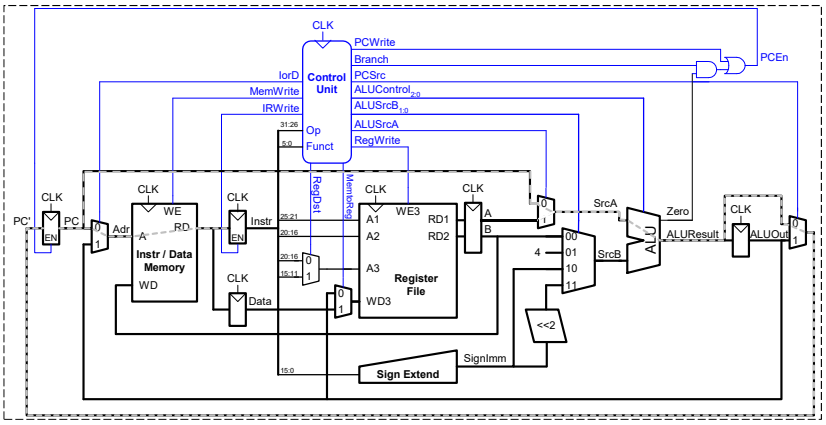
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

59

Ciclo de reloj en multiciclo

- Lo marca el ciclo de reloj más lento (ciclo 1):

$$T_c = t_{pcq} + t_{mux} + t_{mem} + t_{setup}$$



60

Ciclo de reloj en multiciclo

Elemento	Parámetro	Retardo (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + t_{mux} + t_{mem} + t_{setup} =$$
$$= [30 + 25 + 250 + 20] \text{ ps} = 325 \text{ ps}$$

61

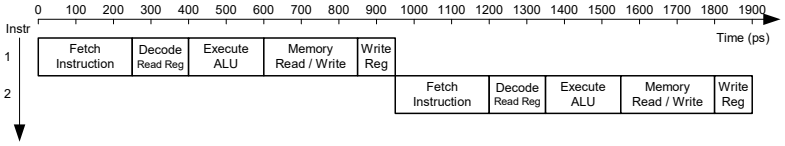
Análisis: uniciclo vs multiciclo

- En uniciclo, todas las instrucciones tardan lo mismo que la más lenta (lw), es decir, 925 ps.
- En multiciclo, todos los ciclos de reloj van al ritmo del ciclo más lento (ciclo 1), es decir, 325 ps.
- En multiciclo, cada instrucción tarda un número distinto de ciclos, entre 3 y 5. El tiempo de ejecución está entre 3·325 y 5·325, es decir, entre 975 y 1625 ps.
- Multiciclo sale más lento que uniciclo (en este ejemplo): la razón es que los ciclos de reloj no se han dividido uniformemente (lo uniforme habría sido 925/5).
- Sin embargo, multiciclo es la base de segmentación (*pipeline*), en donde cada ciclo "corto" de reloj se empieza una nueva instrucción, ejecutándose varias en paralelo.

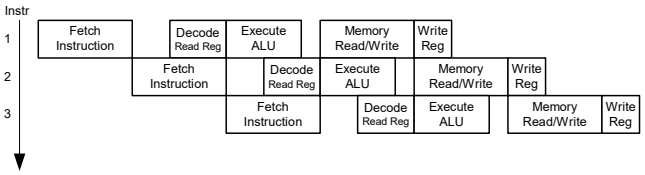
62

Uniciclo vs segmentado

Single-Cycle

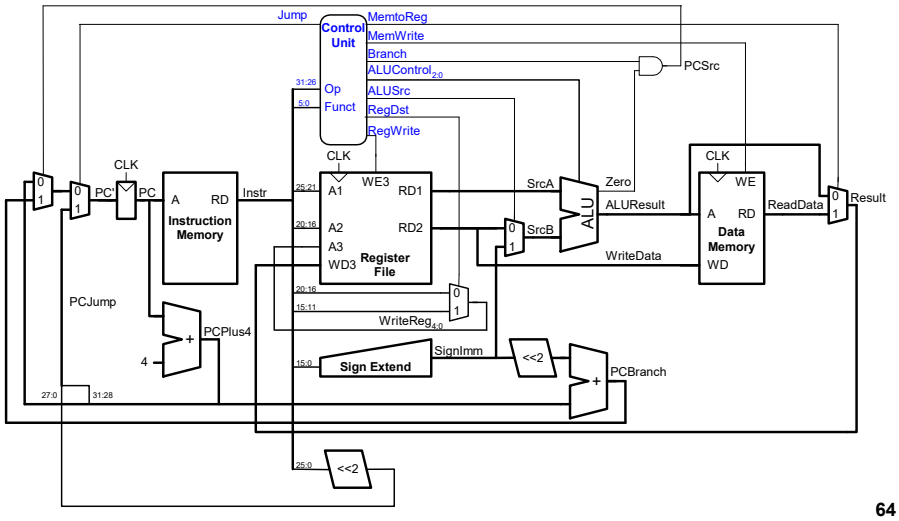


Pipelined



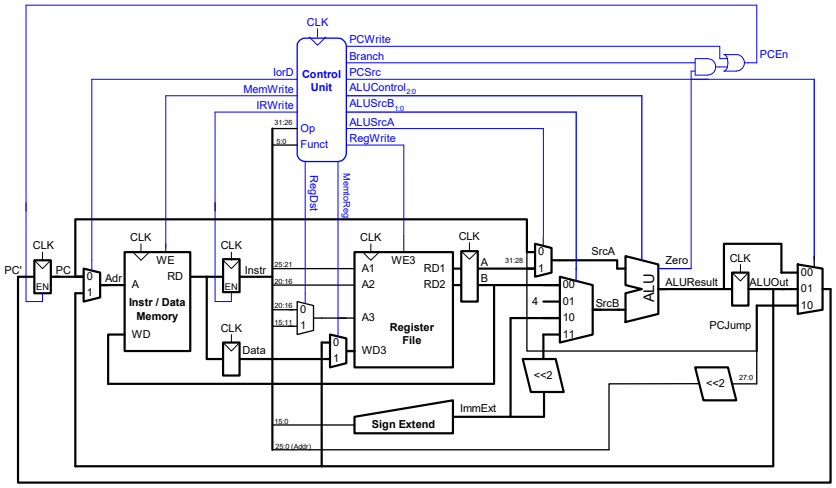
63

Resumen: MIPS uniciclo



64

Resumen: MIPS multiciclo



Estructura de computadores

Unidad 5. El Procesador III: Diseño y control de la ruta de datos: Arquitectura multiciclo

Escuela Politécnica Superior - UAM

Copyright © 2007 Elsevier, "Digital Design and Computer Architecture"