Test-P1P2-ARQO-21-22.pdf

osnofla

Arquitectura de Ordenadores

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

El siguiente programa ensamblador quiere probar la versión del MIPS pipeline con soporte de riesgos de datos.

```
lw $t1, 0($zero) # lw $r9, 0($r0)
lw $t2, 4($zero) # lw $r10, 4($r0)
lw $t3, 8($zero) # lw $r11, 8($r0)
nop
nop
nop
add $t3, $t1, $t2 # en r11 = r9 + r10
add $t1, $t3, $t2 # resultado en R9
```

Seleccione una:

- a.
  El código prueba el funcionamiento de la unidad de detección de riesgos (hazard detection unit), es decir LOAD (LW) y posterior uso del dato. Este código no permite detectar problemas de adelantamiento de datos de ningún tipo.
- b.
  El código permite detectar un mal funcionamiento en el adelantamiento desde la etapa MEM. Pero no permitirá detectar problemas con adelantamientos desde WB. El código no prueba el funcionamiento de la unidad de detección de riesgos (hazard detection unit) ✔
- c.
  El código permite detectar un mal funcionamiento en el adelantamiento debido a riegos de tipo WAR (Write After Read) y WAW (Write After Write). Es imposible detectar riesgos de tipo RAW (Read After Write) con este código.
- d.
  El código permite detectar un mal funcionamiento en el adelantamiento desde la etapa WB. Pero no permitirá detectar problemas con adelantamientos desde MEM. El código prueba el funcionamiento de la unidad de detección de riesgos (hazard detection unit), es decir LOAD (LW) antes del uso del dato.
- e.
  El código permite detectar un mal funcionamiento en el adelantamiento desde la etapa MEM y/o WB. El código prueba el funcionamiento de la unidad de detección de riesgos (hazard detection unit), es decir LOAD (LW) antes del uso del dato.

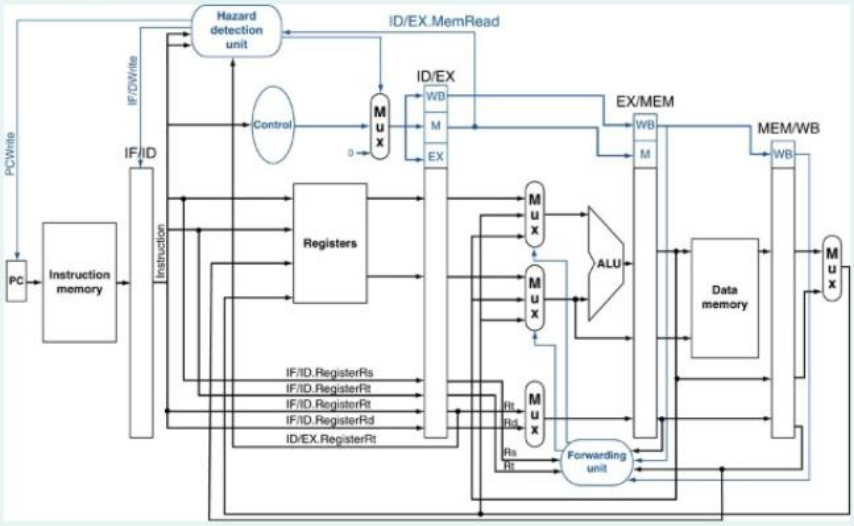
Respecto del banco de Registros provisto por la asignatura en el procesador MIPS y modificado en la P2.

Seleccione una:

- a. Se lee de forma combinacional y escribe en flanco de bajada. No se requiere modificar el banco de registros en Práctica 2, el adelantamiento de MEM a EX está ya resuelto.
- b. El banco de registro lo hemos tenido que diseñar para evitar los riesgos RAW (Data Hazards), adelantando datos de las etapas MEM y WB a EX.
- c. Se lee de forma combinacional y escribe en flanco de subida. La modificación en la Práctica 2 lo hace escribir en flanco de bajada para que se adelanten datos de WB a DE.
- d. Se lee y escribe de forma combinacional en flanco de subida. La modificación en la Práctica 2 lo hace todo secuencial en flanco de bajada para que se adelanten datos de WB a DE.
- e. Se lee con flanco de bajada y escribe en flanco de subida. Se modifica el banco de registros en Práctica 2 para producir el adelantamiento de MEM a EX.

La respuesta correcta es: Se lee de forma combinacional y escribe en flanco de subida. La modificación en la Práctica 2 lo hace escribir en flanco de bajada para que se adelanten datos de WB a DE.

La unidad de detección de riesgos (hazard detection unit) en la versión segmentada implementada durante la práctica 2 tiene por objetivos: (seleccione la respuesta que corresponde).



Seleccione una:

Seleccione una:

a. La unidad de detección de riesgos (hazard detection unit) es completamente opcional. El procesador funciona perfectamente sin esta unidad. El objetivo de la unidad es el aumento del rendimiento haciendo que el CPI (ciclos por instrucción) sea mayor que 1.

De esta manera al detectar ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt)) ejecuta dos operaciones en paralelo.

b. La unidad debe actuar en la etapa de captura (IF) y/o decodificación (ID). De forma combinacional detecta si hay datos en caché. En ese caso adelanta los datos directamente a las etapas EX, MEM y WB. Con esto se evitan burbujas incensarias en el pipeline. Esta es la condición

Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

Luego el procesador funciona a mayor frecuencia de reloj.

c. Se utiliza para detectar la situación que una instrucción utiliza un dato como operando fuente y este es generado por una instrucción de carga de memoria LOAD que está inmediatamente por delante en el pipeline. La acción requerida es detener el pipeline y generar una burbuja.

La detección de la situación se realiza en la etapa de decodificación (ID), por tanto, la instrucción LOAD estará en EX.

ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt))

En el siguiente ciclo de reloj al riesgo, la instrucción LOAD estará en MEM y en EX habrá una burbuja.

d. Se utiliza para detectar la situación que se da cuando una instrucción utiliza un dato como operando fuente y este es generado por una instrucción de carga de memoria LOAD que está inmediatamente por detrás. La acción requerida es detener pipeline y generar nueva dirección en el contador de programa (PC).

La detección de la situación se realiza en la etapa de acceso a memoria (MEM), por tanto, la instrucción LOAD estará en la etapa de captura (IF).

ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt))

En el siguiente ciclo de reloj, la instrucción LOAD grabará el resultado en el banco de registros.

e. La unidad debe actuar en la etapa de ejecución (EX). De forma combinacional detecta la necesidad de adelantar datos desde las etapas MEM y WB. Esta es la condición:

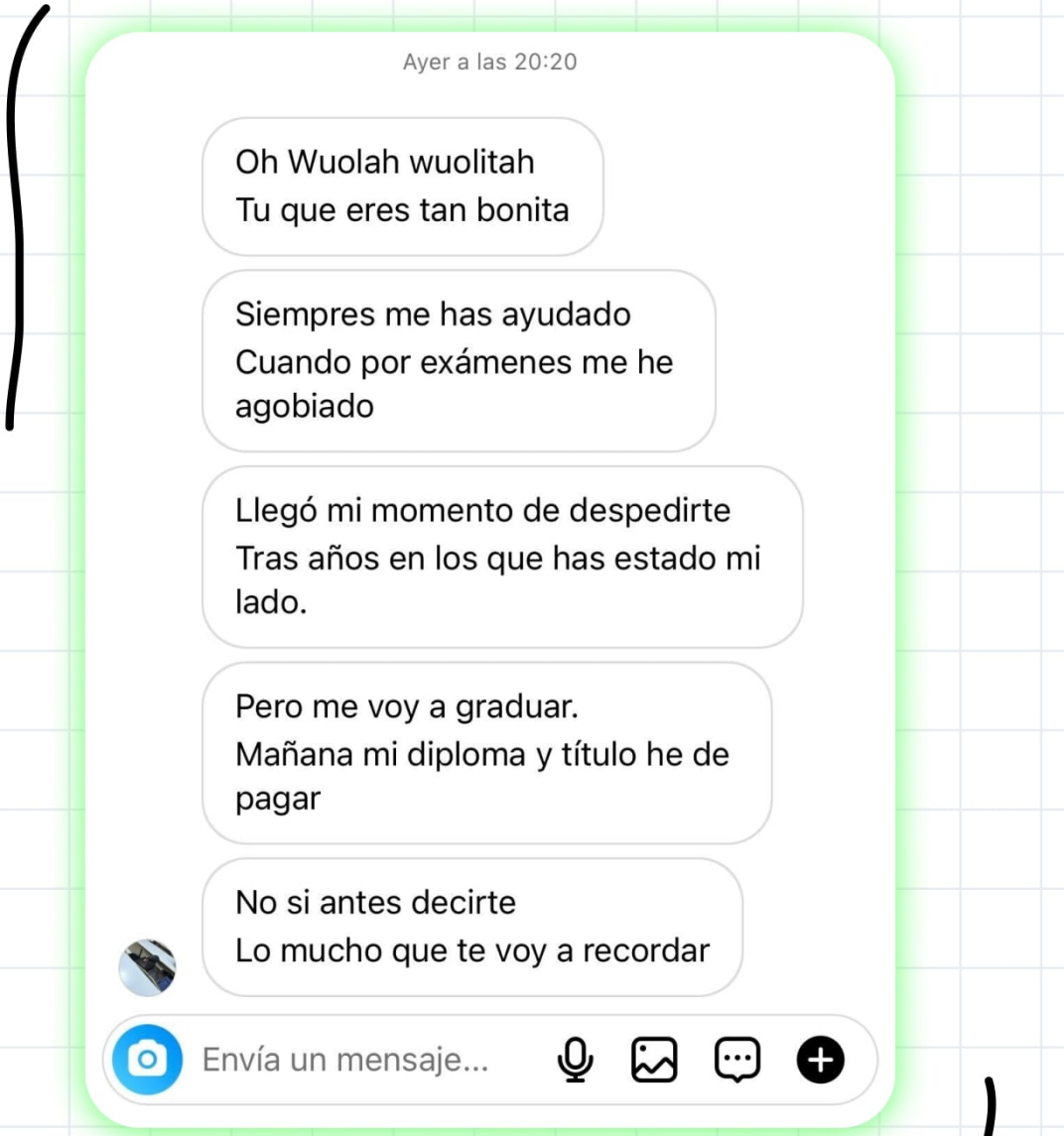
Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA).

De esta manera en la etapa EX se utiliza el dato presente en la etapa MEM en vez de lo leído en el banco de registros

si lees esto me debes un besito

Que no te escriban poemas de amor cuando terminen la carrera 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)



WUOLAH

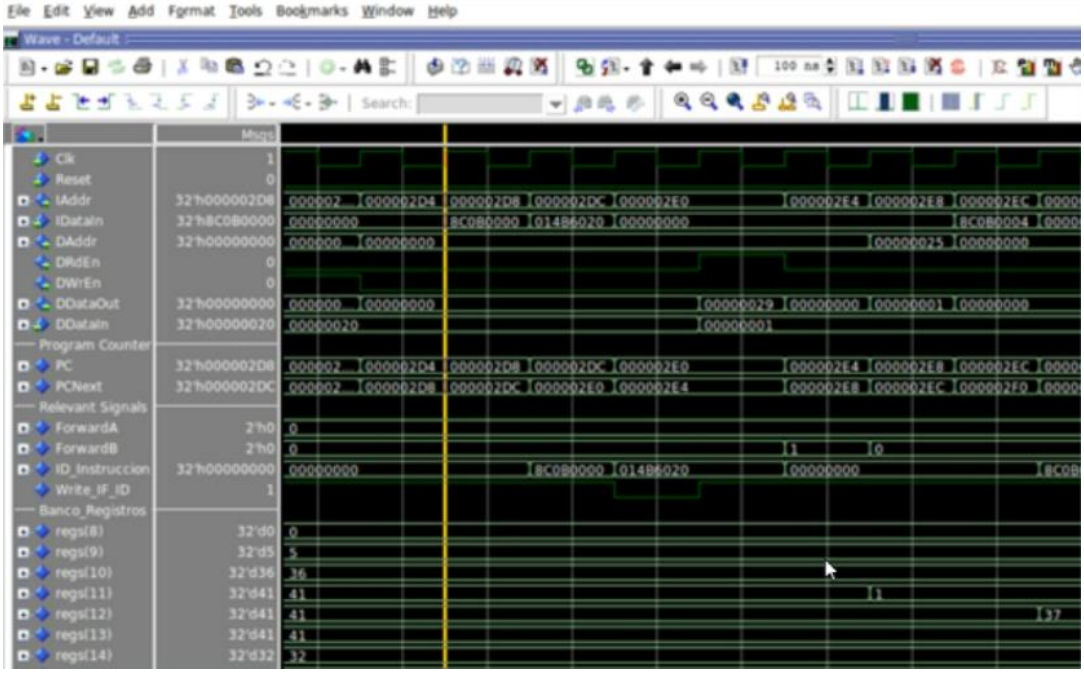
Se está depurando el procesador MIPS en su versión pipeline con la unidad de adelantamiento (data Forwarding) y la detección de riesgos (Hazard Detection Unit). El ensamblador del programa es:

```
nop
nop
nop
# RIESGOS MEMORIA REGISTRO
lw $t3, 0($zero) # en r11 un 1
add $t4, $t2, $t3 # dependencia con la anterior # en r12 37 = 36 + 1
nop
nop
nop
nop
```

El código máquina correspondiente es:

```
000002cc        00000000
000002d0        00000000
000002d4        00000000
000002d8        8c0b0000
000002dc        014b6020
000002e0        00000000
000002e4        00000000
```

El código a evaluar tiene instrucciones NOP por delante y por detrás por simplicidad.

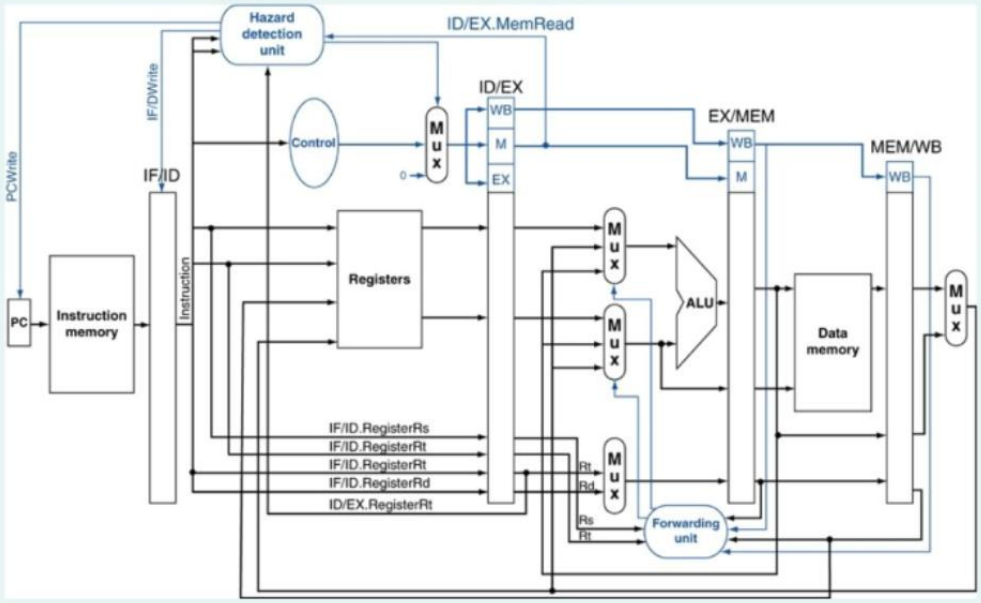


Seleccione una:

- a.
  El funcionamiento NO es correcto. Falla dado que: La señal de habilitación Write_IF_ID se pone a cero cuando el LW está en EX y la tipo-R (el add) está en ID generando una burbuja. Se puede ver que la instrucción en ID (ID_instruccion) permanece 2 ciclos, así como el PC. Adicionalmente se ve que la suma en R12 es correcta (valor 37), sino hubiese generado la burbuja tendría el valor 77.
- b.
  El funcionamiento es correcto. Dato que: El registro 8 nunca cambia. El registro 11 se vuelve 1 gracias a la suma de 1+0. Las señale de ForwardA y Forward B tienen los valores correctos para adelantar el resultado. El registro 14 guarda la suma de r3+r4 (16 + 16). Además se ve que no hay ninguna burbuja dado que el contador de programa (PC) se incrementa continuamente.
- c.
  El funcionamiento NO es correcto. Falla dado que: El registro 8 nunca cambia. El registro 11 se vuelve 1 gracias a la suma de 1+0. Las señale de ForwardA y Forward B tienen los valores correctos para adelantar el resultado. El registro 14 guarda la suma de r3+r4 (16 + 16). Además se ve que no hay ninguna burbuja dado que el contador de programa (PC) se incrementa continuamente.
- d.
  El funcionamiento es correcto. Dado que: La señal de habilitación Write_IF_ID se pone a cero cuando el LW está en EX y la tipo-R (el add) está en ID generando una burbuja. Se puede ver que la instrucción en ID (ID_instruccion) permanece 2 ciclos, así como el PC. Adicionalmente se ve que la suma en R12 es correcta (valor 37), sino hubiese generado la burbuja tendría el valor 77✔
- e.
  No se puede determinar, dado que necesitaría al menos conocer el valor de los registros involucrados en la operación, así como todas las señales del sistema. Por otra parte la simulación con forma de ondas no permite depurar errores de diseño.

si lees esto me debes un besito

La unidad de adelantamiento de datos (forwarding unit) detecta la necesidad de adelantar datos y modifica las señales de control a los multiplexores de entrada a la ALU. Seleccione la respuesta correcta.



○ a. La unidad de adelantamiento se coloca en la etapa de decodificación (ID) o en la de ejecución (EX). De forma secuencial usando una máquina de estados detecta la necesidad de adelantar datos que han sido generados en la etapa de captura (IF) o la de escritura (WB). Estas son las condiciones para tener que adelantar datos

1. Hay necesidad de adelantar el dato desde la etapa MEM:

Si (IF/MEM.RegWrite and (IF/MEM.RegisterRd ≠ 0) and (IF/MEM.RegisterRd = IF/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (IF/MEM.RegWrite and (IF/MEM.RegisterRd ≠ 0) and (IF/MEM.RegisterRd = IF/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

2. Es necesario adelantar datos desde la etapa WB:

Si (IF/WB.RegWrite and (IF/WB.RegisterRd ≠ 0) and not (IF/MEM.RegWrite and (IF/MEM.RegisterRd ≠ 0) and (IF/MEM.RegisterRd = ID/EX.RegisterRs)) and (IF/WB.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (IF/WB.RegWrite and (IF/WB.RegisterRd ≠ 0) and not (IF/MEM.RegWrite and (IF/MEM.RegisterRd ≠ 0) and (IF/MEM.RegisterRd = IF/EX.RegisterRt)) and (IF/WB.RegisterRd = IF/EX.RegisterRt)) Entonces adelantar el operando A (ForwardA)

○ b. La unidad de adelantamiento se coloca en la etapa de decodificación (ID). De forma combinacional detecta la necesidad de adelantar datos desde las etapas EX, MEM y WB. Estas son las condiciones

1. Hay necesidad de adelantar el dato desde la etapa EX a ID:

Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

2. Es necesario adelantar datos desde la etapa WB a ID:

Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

c. La unidad de adelantamiento se coloca en la etapa de ejecución (EX). De forma combinacional detecta la necesidad de adelantar datos desde las etapas MEM y/o WB. Estas son las condiciones

1. Hay necesidad de adelantar el dato desde la etapa MEM:

Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

2. Es necesario adelantar datos desde la etapa WB:

Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

d. La unidad de adelantamiento se coloca en la etapa de ejecución (EX). De forma secuencial usando una máquina de estados detecta la necesidad de adelantar datos desde las etapas MEM y/o WB. Estas son las condiciones

1. Hay necesidad de adelantar el dato desde la etapa MEM a ID:

Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

2. Es necesario adelantar datos desde la etapa WB a ID:

Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

e. La unidad de adelantamiento se coloca en la etapa de ejecución (EX). De forma combinacional detecta la necesidad de adelantar datos desde las etapas MEM y/o WB. Estas son las condiciones

1. Hay necesidad de adelantar el dato desde la etapa MEM:

Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)) Entonces adelantar el operando A (ForwardA)
y
Si (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt)) Entonces adelantar el operando B (ForwardB)

2. Es necesario adelantar datos desde la etapa WB:

Si (MEM/WB.RegWrite) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)) and (MEM/WB.RegisterRd ≠ ID/EX.RegisterRs))
Entonces adelantar el operando A (ForwardA)
y
Si (MEM/WB.RegWrite and (MEM/WB.RegisterRd 0) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt)) and (MEM/WB.RegisterRd ≠ ID/EX.RegisterRt))
Entonces adelantar el operando B (ForwardB)

Respecto a la simulación y verificación del correcto funcionamiento del procesador MIPS desarrollado. Seleccione la aseveración correcta.

Seleccione una:

a. Se simula con un simulador RTL (Modelsim, QuestaSim, VivadoSimulator, etc). El tesbench instancia el procesador y dos memorias. Una será la memoria de datos y la otra la de instrucciones, las que se conectan al procesador diseñado. El contenido de cada memorias se carga desde un fichero de texto separado. El contenido de estos ficheros se genera compilando el ensamblador ya sea usando MARS o el compilador provisto desde la asignatura. El testbench solo genera la señal de reloj y un pulso de reset al comienz

b. Se utiliza VirtualBox o VMWare. Se carga la MV con el procesador a simular. Se ven las formas de ondas en cualquier navegador web que ejecute en Windows o Linux. El programa en ensamblador se tras-compila en el tesbench provisto por la asignatura.

c. Se simula con cualquier compilador de C/C++ (GCC, Clang, Visual C++, etc). El tesbench instancia el procesador y dos memorias. Una será la memoria de datos de nivel y la otra la de segundo nivel, las que se conectan al procesador diseñado. El contenido de cada memorias se carga desde un fichero formato Excel separado. El contenido de estos ficheros se genera compilando el ensamblador ya sea usando MARS o el compilador provisto desde la asignatura. El testbench solo genera la señal de reloj, el resto es por linea de comandos.

d. Se utiliza Xilinx ISE. Para ejecutar el programa se lanza desde lineas de comandos. Tras crear un proyecto en ISE, este genera un fichero de script tcl que se puede usar para poder simular. El testbech provisto se encarga de compilar el codigo ensamblador y cargarlo en la memoria del programa.

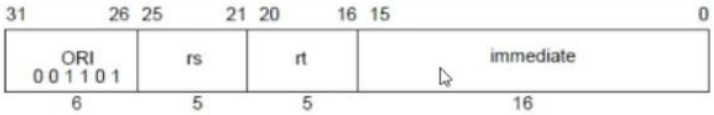
e. Se simula con un simulador RTL (Modelsim, QuestaSim, VivadoSimulator, etc). El tesbench instancia el procesador y la memoria unificada. Existe una única memoria la que se conecta al procesador diseñado. El contenido de esta memoria es un fichero escrito en ensamblador de MIPS. El testbench ejecuta hasta que detecta una instrucción "stop" en el código ensamblador.

si lees esto me debes un besito

Se desea agregar la instrucción ORI al procesador desarrollado durante las prácticas.

Responda los cambios que se deberían realizar. Seleccione la opción que más se ajuste.

Or Immediate **ORI**



Format: ORI rt, rs, immediate **MIPS I**

Purpose: To do a bitwise logical OR with a constant.

Description: rd ← rs OR immediate

The 16-bit *immediate* is zero-extended to the left and combined with the contents of GPR *rs* in a bitwise logical OR operation. The result is placed into GPR *rt*.

Restrictions:

None

Operation:

GPR[rt] ← zero_extend(immediate) or GPR[rs]

Exceptions:

None

Seleccione una:

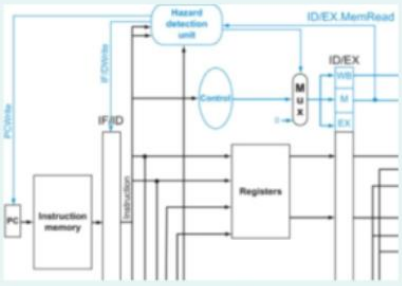
- a. La unidad control de la etapa DE deberá reconocer este nuevo código de operación. En la etapa EX el 1er operando proviene del registro RS, el 2do operando surge de los 16 bits de menor peso de la instrucción extendido con signo a 32 bits. En la etapa MEM la instrucción no realiza nada, en la etapa RW se escribe en el registro destino de la instrucción (RT) ✔
- b. La unidad control de la etapa DE deberá reconocer este nuevo código de operación. En la etapa EX el 1er operando proviene del registro RS, el 2do operando del contenido del registro RT. En la etapa MEM la instrucción no realiza nada, en la etapa RW se escribe en el registro destino de la instrucción (RT).
- c. La unidad control de la etapa DE deberá reconocer este nuevo código de operación. En la etapa EX el 1er operando proviene del registro RT, el 2do operando surge de los 16 bits de mayor peso de la instrucción extendido con signo a 32 bits. En la etapa MEM la instrucción no realiza nada, en la etapa RW se escribe en el registro destino de la instrucción (RS).
- d. La unidad control de la etapa DE deberá reconocer este nuevo código de operación. En la etapa EX el 1er operando proviene del registro RT, el 2do operando del registro RS. En la etapa MEM la instrucción no realiza nada, en la etapa RW se escribe en el registro destino de la instrucción (RS).
- e. La unidad control en la etapa DE no necesita reconocer el código de operación ya que ya esta incluido en las instrucciones tipo-R. En la etapa EX el 1er operando proviene del registro RS, el 2do operando surge de los 16 bits de menor peso de la instrucción extendido con signo a 32 bits. En la etapa MEM la instrucción no realiza nada, en la etapa RW se escribe en el registro destino de la instrucción (RT).

La memoria de instrucciones utilizada en la implementación del procesador segmentado en la P1 y P2. Con el objetivo de alcanzar el mejor CPI funciona de la siguiente forma:

Seleccione una:

- a. Funciona con lectura síncrona (2 ciclos) y escritura combinacional.
- b. Es combinacional en lectura y escritura.
- c. Funciona con lectura y escritura síncrona, requiere 2 ciclos.
- d. Es combinacional en lectura ✔
- e. Funciona con lectura y escritura síncrona, requiere 3 ciclos.

si lees esto me debes un besito



Se ha modelado en VHDL los registros IF/ID del procesador MIPS segmentado implementado en la práctica 2. Los registros unen las epatas de IF (Instruction Fetching) y la etapa ID (Instruction Decode). La semántica de las señales es la siguiente:

Reset: es el reset asíncrono global del sistema.

clear_ID: señal de reset síncrona activo en alto utilizada para eliminar la instrucción de la etapa de decodificación (ID), usado en los saltos.

Write_IFID: señal de habilitación, utilizada para bloquear una instrucción cuando es necesario agregar una burbuja. Activo en Alto.

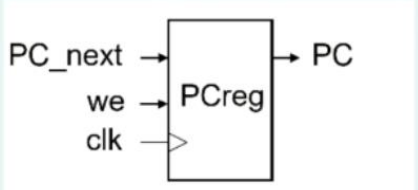
Instrucciones_IF: señal que contiene la instrucción capturada desde la memoria de instrucciones en la etapa IF.

PCPlus4_IF: señal que contiene el contador de programa (PC), sumado 4. Es decir la dirección de la siguiente instrucción.

Señale el código correcto.

b.
```
regs_IF_ID: process(Clk, Reset)
begin
   if Reset = '1' then
      PCPlus4_ID <= (others => '0');
      Instruccion_ID <= (others => '0');
   elsif rising_edge(Clk) then
      if clear_ID = '1' then
         PCPlus4_ID <= (others => '0');
         Instruccion_ID<= (others => '0');
      elsif Write_IFID = '1' then
         PCPlus4_ID <= PCPlus_IF;
         Instruccion_ID<= IDataIn;
      end if;
    end if;
end process✔
```

¿Cuál es la descripción correcta para un Registro Flip-Flop tipo D con reset asincrono activo alto (reset), con señal de habilitación (ce) activa alta y reloj activo en flanco ascendente (clock) ? Este registro se usa para modelar el registro PC (program counter) de microprocesador



d.
```
PC_reg_proc: process(Clk, Reset)
begin
  if Reset = '1' then
     PC <= (others => '0');
  elsif rising_edge(Clk) then
     if we = '1' then
        PC <= PC_next;
     end if;
  end if;
end process✔
```