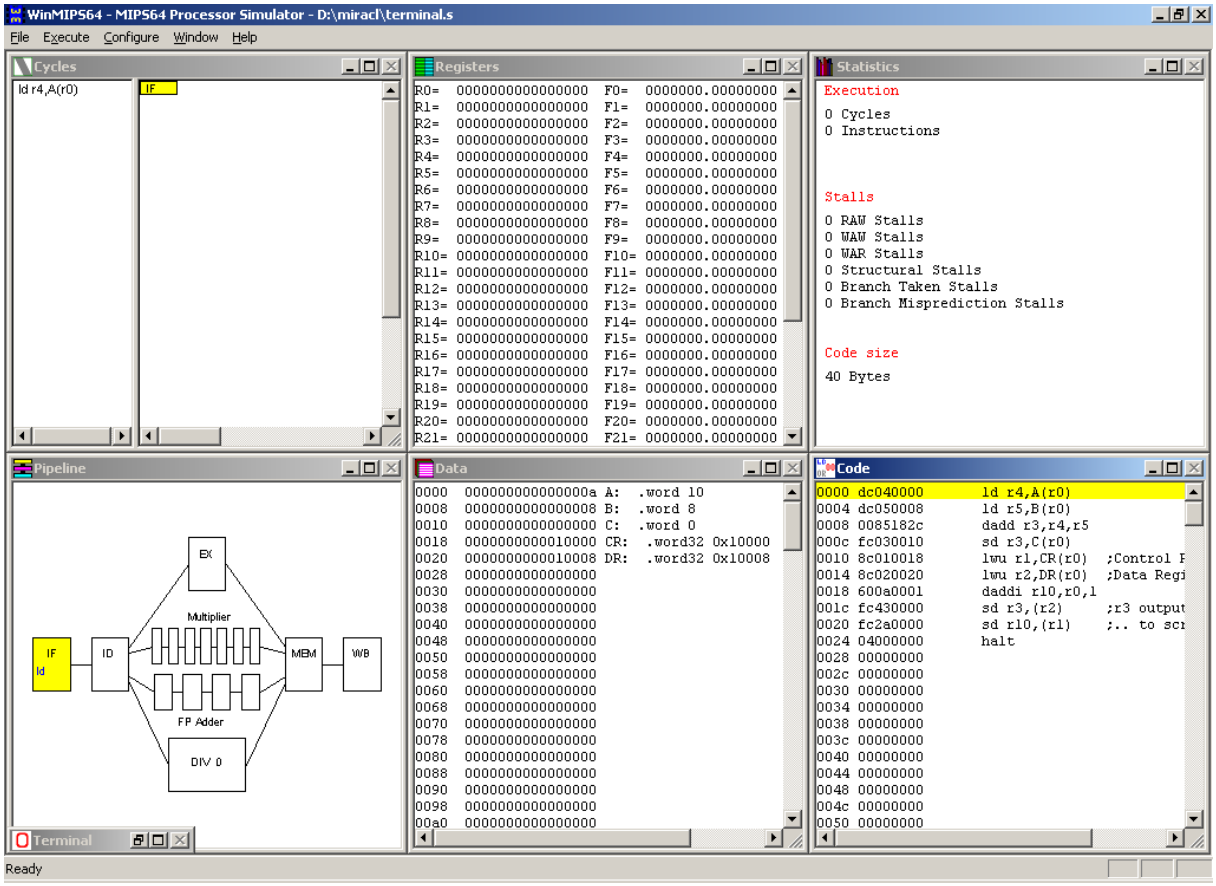
Usando el simulador WinMIPS64

Este documento presenta al WinMIPS64, un simulador para Windows de una implementación del procesador MIPS64 de 64 bits que utiliza segmentación (pipeline). Las actividades que se presentan están orientadas a mostrar las distintas características del simulador.

1. Iniciando y configurando el WinMIPS64

Una vez que se inicie el WinMIPS64, aparecerá una ventana, denominada ventana principal o *main*, que contiene siete ventanas más y una barra de estado en la parte inferior de la misma.

Las siete ventanas se denominan **Pipeline**, **Code**, **Data**, **Registers**, **Statistics**, **Cycles** y **Terminal.**

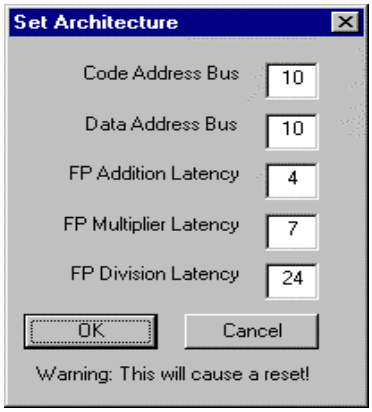


Pipeline	Esta ventana muestra una representación esquemática de las cinco etapas del pipeline del procesador MIPS64 y de las unidades para operaciones de punto flotante (adición/sustracción, multiplicación y división). En ella se indica visualmente que instrucción está en cada una de las etapas.
Code	Esta ventana muestra el contenido de la memoria de código mediante una representación en tres columnas. La de la izquierda muestra la dirección correspondiente en la memoria. La del centro muestra la representación en código de máquina de 32 bits de la instrucción contenida en dicha dirección. Finalmente, la de la derecha muestra la instrucción en lenguaje *assembly*. Haciendo doble clic sobre alguna de las instrucciones de esta ventana establece o quita un punto de parada (*breakpoint*).

Data	Esta ventana muestra el contenido de la memoria de datos. Si bien la memoria puede ser direccionada de a bytes, el contenido es mostrado en porciones de 64 bits, ya que resulta más apropiado para un procesador de 64 bits. Haciendo doble clic es posible editar el valor entero contenido en esa dirección de memoria. Para editar un valor en punto flotante, es necesario hacer doble clic con el botón derecho.
Register	Esta ventana muestra los valores almacenados en cada registro del procesador. Si el registro está mostrado en gris es porque se encuentra en proceso de ser escrito por alguna instrucción. Si se muestra con algún color, dicho color indica la etapa de la segmentación para cuál el valor del registro está disponible para hacer *forwarding*. Esta ventana permite cambiar el contenido de los registros enteros y de punto flotante siempre que no se encuentren en proceso de ser escritos o de *forwarding*. Para hacerlo, basta con hacer doble clic sobre el registro que se desea cambiar, tras lo cual aparecerá una ventana preguntando el nuevo valor.
Clock Cycle diagram	Esta ventana muestra el comportamiento temporal del pipeline. Registra la historia de las instrucciones a medida que entran y salen de él. Una instrucción que causa un parón (stall) es remarcada en color azul mientras que una instrucción que debe ser demorada como resultado de un parón es mostrada en color gris.
Statistics	Esta ventana provee estadísticas sobre el número de ciclos simulados, cantidad de instrucciones procesadas, el promedio de Ciclos Por Instrucción (CPI), el tipo de parones, el número de saltos condicionales y la cantidad de instrucciones de carga (Load) y almacenamiento (Store).
Terminal	Esta ventana simula ser una terminal de entrada/salida que además de procesar texto, posee algunas capacidades gráficas limitadas.
Status Line	La barra de estado ubicada en la parte inferior de la ventana principal normalmente muestra el texto "Ready" pero durante la ejecución de un programa, muestra información útil respecto de la simulación.

Al iniciar, uno puede asegurarse que la simulación se encuentre en un estado inicial conocido haciendo clic en la opción del menú ***File→Reset MIPS64***.

WinMIPS64 puede ser configurado de muchas formas. Se puede cambiar la estructura y los requerimientos de tiempo del pipeline de punto flotante y el tamaño de la memoria de código y de datos. Para ver o cambiar la configuración por defecto, es necesario acceder al menú ***Configure→Architecture***. La siguiente ventana aparecerá:



Puede cambiarse la configuración editando los valores de los campos apropiados. Cualquier cambio en las demoras de punto flotante será reflejado en la ventana **Pipeline**. La opción *Code Address Bus* se refiere al número de cables conectados al bus de direcciones. De esa manera, un valor de 10 significa que 2^{10} = 1024 bytes de memoria de código van a estar disponibles y van a ser mostrados en la ventana **Code**. Algo similar ocurre con la opción *Data Address Bus*, solo que referida a la memoria de datos.

Tres opciones más pueden ser elegidas en el menú ***Configuration***: ***Multi-Step***, ***Enable Forwarding***, ***Enable Branch Target Buffer*** y ***Enable Delay Slot***. Para los siguientes ejemplos de todas ellas debería estar marcada solamente ***Enable Forwarding*** y el resto deberán estar desmarcadas.

Es posible cambiar el tamaño y/o posición de las ventanas contenidas en la ventana principal al igual que minimizar o maximizar cada una de ellas.

2. Cargando un programa de prueba.

Utilice un procesador de texto cualquiera (el Bloc de notas, por ejemplo) para crear un archivo llamado **sum.s**, que va a contener un programa para el MIPS64 que calculará la suma de dos enteros almacenados en las posiciones de memoria **A** y **B** y va a almacenar el resultado de la suma en la posición de memoria **C**.

```
        .data
A:      .word 10
B:      .word 8
C:      .word 0

        .text
main:
        ld r4,A(r0)
        ld r5,B(r0)
        dadd r3,r4,r5
        sd r3,C(r0)
        halt
```

El simulador provee una utilidad de línea de comando llamada `asm.exe` que permite verificar que un programa no contenga errores de sintaxis. Para verificar el programa de este ejemplo, es posible ejecutar el siguiente comando:

```
C:\winmips64> asm sum.s
```

Para poder comenzar con la simulación, el programa debe cargarse en la memoria principal. Para lograr eso, se debe usar la opción ***File→Open***. Al abrir la ventana de diálogo, se mostrará una lista con los programas assembly que se encuentren en el directorio actual. Busque y seleccione a **sum.s**, el archivo que acaba de escribir. Luego de seleccionarlo, es necesario hacer clic en el botón ***Open***.

Ahora, el programa se encuentra cargado en memoria y todo está listo para comenzar la simulación.

Es posible ver el contenido de la memoria de código usando la ventana **Code** y observar los datos del programa en la ventana **Data**.

3. Simulación

3.1 Simulación Ciclo-por-Ciclo.

En cualquier momento se puede presionar la tecla ***F10*** para reiniciar la simulación desde el comienzo.

Al iniciar la simulación, se puede observar que la primera línea en la ventana **Code**, correspondiente a la dirección 0000h se encuentra remarcada con color amarillo. La etapa IF en la ventana **Pipeline** también se encontrará coloreada en amarillo y contendrá el nombre de la primera instrucción de assembly del programa.

Luego de hacer esto, observe la ventana **Code** y vea la primera instrucción **`ld r4,A(r0)`**. Observe la ventana **Data** y encuentre la variable etiquetada como **`A`**.

Clock 1:

Haciendo clic en ***Execute→Single Cycle*** (o simplemente presionando ***F7***) se avanza la simulación un paso en el tiempo o, lo que es lo mismo, un pulso de reloj. En la ventana **Code**, el color de la primera instrucción cambiará a color azul y la segunda instrucción se marcará ahora en amarillo. Estos colores indican la etapa del pipeline en la que cada instrucción se encuentra (amarillo para IF, azul para ID, rojo para EX, verde para MEM y púrpura para WB).

Si se observa la etapa IF en la ventana **Pipeline**, se verá que la segunda instrucción **`ld r5, B(r0)`** se encuentra en la etapa IF y la primer instrucción **`ld r4,A(r0)`** ha avanzado a la segunda etapa ID.

Clock 2:

Presionando ***F7*** de nuevo, se van a reacomodar los colores en la ventana **Code**, mostrando por primera vez el color rojo correspondiente a la tercera etapa, EX. La instrucción **`dadd r3,r4,r5`** ingresará al pipeline. Nótese que el color de la instrucción indica en que etapa del pipeline la instrucción va a ser completada cuando ocurra el siguiente pulso del reloj.

Clock 3:

Presionando ***F7*** de nuevo, se van reacomodar nuevamente los colores en la ventana **Code**, mostrando por primera vez el color verde correspondiente a la etapa MEM. La instrucción **`sd r3,C(r0)`** va a ingresar al pipeline. Observe el diagrama en la ventana **Cycle**, el cual muestra la secuencia en que cada una de las instrucciones fue pasando por cada etapa en cada pulso de reloj.

Clock 4:

Al presionar nuevamente ***F7***, cada etapa del pipeline estará activada con una instrucción. El valor que va a ir a parar al registro **`r4`** ha sido leído desde la memoria pero aún no se ha escrito en **`r4`**. Sin embargo, está disponible para *forwarding* desde la etapa MEM. Observar que en la ventana **Registers** **`r4`** está mostrado en verde (el color correspondiente a MEM). ¿Puede explicar el valor que contiene **`r4`**? Note que la última instrucción **`halt`** ha entrado en el pipeline.

Clock 5:

Presione ***F7*** nuevamente. Algo interesante ha ocurrido. El valor destinado para el registro **`r5`** pasa a estar disponible para hacer *forwarding*. Sin embargo, el valor para **`r5`** no estaba disponible en el momento que la instrucción **`dadd r3,r4,r5`** se ejecutaba en la etapa EX. Por lo tanto, se mantiene demorada en EX. La barra de estado muestra *"RAW stall in EX (R5)"*, que indica donde ha ocurrido el parón, cual fue el motivo y que registro, al no estar disponible, fue el responsable de causarlo.

El diagrama en la ventana **Cycle** y la imagen en la ventana **Pipeline** muestran claramente que la instrucción **`dadd`** se encuentra demorada en la etapa EX y que las demás instrucciones detrás de ella tampoco pueden continuar. En la ventana **Cycle**, la instrucción **`dadd`** se encuentra remarcada en azul y las instrucciones detrás de ella se muestran en gris.

Clock 6:

Presione ***F7***. La instrucción **`dadd r3,r4,r5`** se ejecuta y su salida, destinada al registro **`r3`**, queda disponible para hacer *forwarding*. El valor de salida es 12h, que resulta ser la suma de 10+8 = 18 en decimal. Esta es nuestra respuesta.

Clock 7:

Presione ***F7***. La instrucción **`halt`** que entra en la etapa IF tiene el efecto de "congelar" el pipeline, de manera que no se acepte ninguna instrucción nueva.

Clock 8:

Presione ***F7***. Si se examina la ventana **Data**, se observará que la variable **`C`** ahora contiene el valor 12h. La instrucción **`sd r3,C(r0)`** lo ha escrito en la memoria en la etapa MEM del pipeline, usando el valor recibido mediante *forwarding* del registro **`r3`**.

Clock 9:

Presione ***F7***.

Clock 10:

Presione ***F7***. El programa ha terminado

Al mirar en la ventana **Statistics**, se observa que se han contado 2 cargas (loads), 1 almacenamiento (store) y un parón (stall) RAW. Se necesitaron 10 ciclos de reloj para ejecutar 5 instrucciones así que el CPI=2. Esto es altamente artificial debido al tiempo que se necesitó en llenar el pipeline.

La ventana **Statistics** es extremadamente útil para comparar los efectos que traen los cambios en la configuración. Para examinar los efectos de utilizar *forwarding*, que hasta ahora estaba habilitando, sería interesante conocer el tiempo de ejecución del programa de ejemplo sin contar con esta característica activada.

Para lograrlo, se debe hacer clic en ***Configure*** y desmarcar la opción ***Enable Forwarding***.

Tras repetir la ejecución ciclo-por-ciclo, si se vuelve a examinar la ventana **Statistics**, se podrán comparar los resultados. Note que hay más parones debido a que las instrucciones son retenidas en la etapa ID esperando por un registro y por tanto, esperando a que una instrucción anterior complete la etapa WB. Las ventajas de utilizar *forwarding* deberían ser más que obvias.

3.2 Otros modos de ejecución.

Antes de comenzar, es necesario hacer clic en ***File→Reset MIPS64***. Debe notarse que esto borrará la memoria de datos, por lo que será necesario repetir el procedimiento para cargar el programa. Sin embargo, haciendo clic en ***File→Reload*** o presionando ***F10*** se conseguirá el mismo efecto para reiniciar la simulación de una forma más rápida. Para este caso, usa la segunda opción.

Es posible correr la simulación por un número especificado de ciclos. Utilice la opción ***Execute→Multi cycle…*** para ello. El número de ciclos que se van a ejecutar se especifica a través de la opción ***Configure→Multi-step*.**

También, es posible ejecutar el programa completo presionando la tecla ***F4*** o haciendo clic en ***Execute→Run to***.

Por último, existe la posibilidad de establecer puntos de parada (breakpoints). Para establecer un nuevo punto de parada, basta con hacer doble clic en la ventana **Code** sobre la instrucción en la que uno quiere que se detenga la ejecución. Por ejemplo, si luego de presionar ***F10*** para reiniciar la simulación, se pone un punto de parada haciendo doble clic sobre la instrucción **`dadd r3,r4,r5`** y se presiona ***F4***, el programa va a detenerse cuando esta instrucción entre la etapa IF. Para quitar un punto de parada, se debe hacer nuevamente doble clic sobre la misma instrucción.

3.3 Salida de la Terminal

El simulador soporta un dispositivo de entrada/salida básico, que trabaja como una pantalla de terminal simple, con soporte para algunas operaciones gráficas. La salida de un programa puede aparecer en esta pantalla de esta terminal. Para mostrar la salida del programa anterior, es necesario modificarlo de la siguiente manera:

```
        .data
A:      .word 10
B:      .word 8
C:      .word 0
CR:     .word32 0x10000
DR:     .word32 0x10008

        .text
main:
        ld r4,A(r0)
        ld r5,B(r0)
        dadd r3,r4,r5
        sd r3,C(r0)

        lwu r1,CR(r0)  ;Registro Control
        lwu r2,DR(r0)  ;Registro Data
        daddi r10,r0,1
        sd r3,(r2)     ;Enviar r3...
        sd r10,(r1)    ;... a la pantalla

        halt
```

Después de que este programa sea ejecutado, se podrá ver el resultado de la suma impreso en decimal en la ventana **Terminal**. Para ver una demostración más completa de las capacidades de entrada/salida de la terminal, vea los programas de ejemplo *testio.s* y *hail.s* que trae el simulador.

El conjunto de instrucciones.

Son reconocidas las siguientes *directivas* para el ensamblador:

```
.text                 - comienzo del código
.code                 - comienzo del código (igual que .text)
.data                 - comienzo de los datos
.org    <n>           - dirección de inicio
.space  <n>           - deja n bytes vacíos
.ascii  <s>           - almacena una cadena ASCII
.asciiz <s>           - almacena una cadena ASCII terminada en 0.
.word   <n1>,<n2>..   - almacena número(s) de 64-bits
.byte   <n1>,<n2>..   - almacena byte(s)
.word32 <n1>,<n2>..   - almacena número(s) de 32 bits
.word16 <n1>,<n2>..   - almacena número(s) de 16 bits
.double <n1>,<n2>..   - almacena número(s) en punto flotante
```

donde <n> denota un número como 24, <s> denota una cadena como "fred" y <n1>,<n2>… denota números separados por coma. Los registros enteros pueden ser referidos como r0-r31 o R0-R31 o $0-$31 o usando pseudonimos MIPS estandar como $zero para r0, $t0 para r8, etc. Es importante notar que el tamaño de un valor inmediato está limitado a 16 bits. Además, el valor inmediato para una instrucción de desplazamiento de registro está acotado a 5 bits, por lo tanto, un desplazamiento mayor a 31 bits no está permito.

Los registros de punto flotante pueden ser referidos como f0-f31 o F0-F31.

Las siguientes *instrucciones* son soportadas (*):

```
ld      - load 64-bit double-word
sd      - store 64-bit double-word
l.d     - load 64-bit floating-point
s.d     - store 64-bit floating-point
halt    - stops the program
daddi   - add immediate
andi    - logical and immediate
ori     - logical or immediate
xori    - exclusive or immediate
slti    - set if less than or equal immediate
beq     - branch if pair of registers are equal
bne     - branch if pair of registers are not equal
beqz    - branch if register is equal to zero
bnez    - branch if register is not equal to zero
j       - jump to address
jr      - jump to address in register
jal     - jump and link to address (call subroutine)
jalr    - jump and link to address in register (call subroutine)
dsll    - shift left logical
dsrl    - shift right logical
dsra    - shift right arithmetic
dsllv   - shift left logical by variable amount on register
dsrlv   - shift right logical by variable amount on register
dsrav   - shift right arithmetic by variable amount on register
nop     - no operation
and     - logical and
or      - logical or
xor     - logical xor
slt     - set if less than
dadd    - add integers
dsub    - subtract integers
```

```
dmul    - signed integer multiplication
ddiv    - signed integer division
add.d   - add floating-point
sub.d   - subtract floating-point
mul.d   - multiply floating-point
div.d   - divide floating-point
mov.d   - move floating-point
cvt.d.l - convert 64-bit integer to a double floating-point format
cvt.l.d - convert double floating-point to a 64-bit integer format
mtc1    - move data from integer register to floating-point register
mfc1    - move data from floating-point register to integer register

(*) el simulador soporta mas instrucciones que las mencionadas.
```

Área de Entrada/Salida ubicada en la memoria.

Las direcciones de los registros CONTROL y DATA son:

```
CONTROL: .word32 0x10000
DATA:    .word32 0x10008

Set CONTROL = 1, Set DATA to Unsigned Integer to be output
Set CONTROL = 2, Set DATA to Signed Integer to be output
Set CONTROL = 3, Set DATA to Floating Point to be output
Set CONTROL = 4, Set DATA to address of string to be output
Set CONTROL = 5, Set DATA+5 to x coordinate, DATA+4 to y coordinate,
                 and DATA to RGB colour to be output
Set CONTROL = 6, Clears the terminal screen
Set CONTROL = 7, Clears the graphics screen
Set CONTROL = 8, read the DATA (either an integer or a floating-point)
                 from the keyboard
Set CONTROL = 9, read one byte from DATA, no character echo.
```

Instalación

El programa no posee un instalador. Es su computadora, descomprima el archivo del simulador en cualquier lugar conveniente y cree un acceso directo que apunte a él para facilitar el uso del mismo. Deberá saber que el programa WinMIPS64 va a escribir dos archivos de configuración en el directorio donde se encuentra, uno llamado winmips64.ini que almacena detalles de la arquitectura y otro, llamado winmips64.las, que recuerda el último archivo .s accedido.