Objetivo: Comprender la estructura de un programa en lenguaje ensamblador MIPS básico. Introducción a la interfaz `mipsx` y al conjunto de instrucción MIPS.

Recursos y Bibliografía:

Arq. MIPS Vol I, II and III.

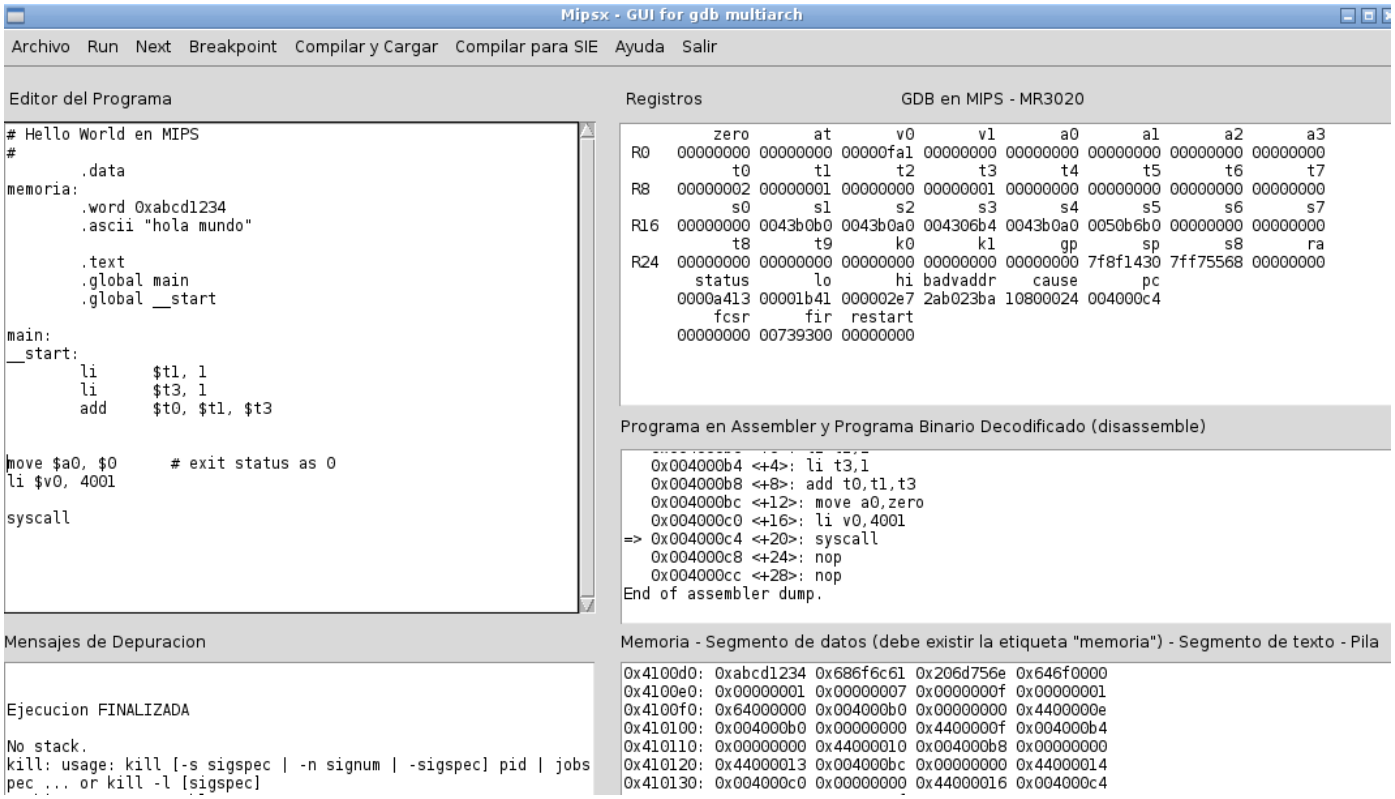
Programa `mipsx` desarrollado por la cátedra.

Introducción a mipsx

El programa `mipsx`, ha utilizar en este trabajo práctico, es una interfaz gráfica para desarrollar programas en lenguaje ensamblador MIPS. Es desarrollado por la cátedra y trabaja en conjunto con sistemas MIPS emulados y reales.

`mipsx` permite ensamblar y vincular los programas desarrollados. También ejecutar, y al mismo tiempo analizar, los programas compilados a través del debugger gdb. A partir de estas características, es posible realizar todo el proceso de desarrollo y verificación de programas en lenguaje ensamblador con una única herramienta, mientras que los programas pueden ser ejecutados y analizados en diferentes sistemas `MIPS`.

A continuación se detallan las partes fundamentales de esta interfaz.



El programa presenta cinco paneles:

Editor del Programa: es el editor ha utilizar para el desarrollo de los programas en lenguaje ensamblador.

Registros: presenta el contenido de los registros del procesador (CPU).

Programa en assembler y Programa Binario Decodificado: muestra el listado del programa con sus

respectivos números de línea. Permite seguir la traza de las instrucciones ejecutadas del programa siendo procesado por gdb.

También presenta un "disassemble" del programa en memoria, muy útil para reconocer pseudoinstrucciones y comparar con el programa original.

Memoria: permite visualizar el contenido de la memoria de la máquina MIPS. En particular, los segmentos del programa en memoria, de datos, de texto, y pila.

Mensajes de depuración: muestra información del estado del ensamblaje y vinculación, carga, y mensajes del programa siendo ejecutado por gdb, y la salida estándar.

El panel superior contiene los botones que controlan a `mipsx` :

Run : Ejecuta el programa siendo analizado hasta su finalización.
Next : Ejecuta la siguiente instrucción del programa en ejecución.
Breakpoint : Define o elimina breakpoints (NO IMPLEMENTADO).
Compilar y Cargar : copia, ensambla y vincula, en el sistema `MIPS` remoto, el programa siendo editado. Carga el programa ejecutable con gdb e inicia la ejecución de la primera instrucción del programa si no se presentaron errores.

NOTA: Para ejecutar `mipsx` debe contar con un usuario de red en los laboratorios de Pcs Linux. Si todavía no tiene una cuenta de usuario de red puede solicitar una a la cátedra. La cuenta será de utilidad para el resto de las materias en la carrera.

1. Ejecutar el programa `mipsx`: En una terminal Linux debe conectarse remotamente a una computadora de la facultad utilizando el siguiente comando, y reemplazando `USUARIO` por el suyo. Tenga en cuenta que al momento de ingresar la contraseña, no se hace eco de las teclas presionadas (no se muestra nada, ni siquiera asteriscos). Para usuarios de windows, ver el anexo al final del documento.

```
$ ssh -X -p 60173 USUARIO@aula-ssh.fi.uncoma.edu.ar
```

Una vez conectado, para ejecutar la herramienta mipsX debe ingresar el siguiente comando:

```
$ /export/home/extras/mipsx/mipsx.sh
```

2. Desarrollar el siguiente programa utilizando `mipsx`

IMPORTANTE: No utilice la función COPIAR y PEGAR para transferir el programa a `mipsx`. Esta acción suele "copiar" caracteres no-visibles, que puede resultar en un programa en `mipsx` que no está apto para ser ensamblado.

```
# Hello World en MIPS
#
        .data           # Zona de datos
memoria:
        .word 2, 3
        .word 0xabcd1234
        .ascii "hola mundo"


        .text            # Zona del programa
        .global main     # La etiqueta main debe ser global
        .global __start # La etiqueta __start debe ser global
__start:
main:
        la      $t0, memoria
```

```
        lw      $t1, 0($t0)
        lw      $t2, 4($t0)
        add     $t3, $t1, $t2
        sw      $t3, 8($t0)

# finaliza el programa solicitando un `exit` al sistema Linux
        move $a0, $0        # exit status as 0
        li $v0, 4001        # __NR_exit include/asm-mips/unistd.h
        syscall
```

Las palabras que comienzan con un punto: `.data, .word, .text` y `.globl` son directivas al compilador. Se utilizan para dar ordenes especiales acerca de cómo debe generarse el código. Los significados de cada una de las directivas utilizadas en este programa son:

- **`.data`**: los siguientes elementos han de situarse en el segmento de datos.
- **`.word, .byte, .float, .ascii, etc.`**: especifican el tipo de los datos que se detallan a continuación de la directiva.
- **`.text`**: indica al ensamblador que los siguientes elementos han de situarse en el segmento de texto (programas) del usuario.
- **`.globl`**: declara la etiqueta que le sigue como global, de forma tal que pueda ser referenciada desde el código del sistema que inicia nuestro programa.

3. Compilar y Cargar el programa. Verificar en el panel de Depuración que no se han presentado errores al compilar el programa.

Si el programa se cargó con `gdb` sin errores puede analizar el contenido de los diferentes paneles de información. Conteste :

1. ¿Cuál es el valor de registro del procesador $pc?
2. ¿Cuál es el valor del registro $0 ($zero)?
3. Analice con el registro $pc y los diferentes paneles cuál es la próxima instrucción que se ejecutará en la CPU MIPS. ¿Puede indicar la línea de su programa que corresponde a la próxima instrucción a ejecutar por la CPU? (Indicar la línea del programa que corresponde a esta instrucción y la instrucción en lenguaje ensamblador).

4. Ejecución paso a paso: Compilar y cargar el programa nuevamente. Luego, ejecute el programa paso a paso utilizando la orden Next. Cada vez que demos la orden de ejecutar un paso, se ejecutará una única instrucción y se detendrá la ejecución del programa, para que podamos observar el efecto producido por la instrucción. **Tener en cuenta que al Compilar y Cargar el programa, `mipsx` ya ejecuta la primera instrucción del programa y detiene la ejecución**. Contestar:

1. ¿Cuál es la dirección en memoria de la primera instrucción del programa?
2. ¿Cuál es la dirección en memoria de la última instrucción del programa?
3. Con la información anterior responda: ¿Cuántos bytes ocupa el segmento de código en memoria? (de instrucciones).

5. Ejecución del programa. Cargue nuevamente y ejecute el programa completo utilizando la orden Run. La máquina MIPS ejecutará el programa por completo hasta su finalización. Cuando la máquina MIPS termine la ejecución se indicará en el panel de Depuración con la leyenda “Ejecución FINALIZADA”. Conteste:

1. ¿Cuál es el valor del registro t1, t2 y t3 al finalizar el programa?
2. Explique el funcionamiento del programa.

3. Explique qué sucedió con la palabra de texto “hola” que se encontraba en memoria.

6. Cree un programa que calcule el en enésimo elemento de la sucesión fibonacci, y guarde el resultado en una variable “resultado”. El número del elemento a ser calculado debe ser tomado de la variable “N”. Recuerde que la secuencia de fibonacci está definida como:

- F(1) = 1
- F(2) = 1
- F(n) = F(n-1) + F(n-2), para “n” distinto de 1 y 2.

Los primeros 10 de la sucesión: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…

No olvide agregar el siguiente segmento de código al final del programa para que el proceso pueda terminar correctamente, además de la declaración de `__start`. Se recomienda utilizar la ejecución paso a paso para detectar los errores.

```
# finaliza el programa solicitando un `exit` al sistema Linux
        move $a0, $0      # exit status as 0
        li $v0, 4001      # __NR_exit include/asm-mips/unistd.h
        syscall
```

7. Sabiendo que X e Y son dos variables declaradas como enteros, y utilizando solo las instrucciones de salto beq, bne y j, y slt para comparar entre registros, cree fragmentos de código máquina que implementen las siguientes estructuras de control:

```
1. if(X==Y){/*código*/}
2. if(X<Y){/*código*/} else {/*código2*/}
3. while(X!=11){/*código*/}
4. do{/*código*/}while(X!=0)
5. for(X=0; X<Y; X++){/*código}
```

Ejemplo:

La estructura de control `while(true){/*código*/}` puede ser implementada con el siguiente código `MIPS`:

```
salto1:

     #código

j salto1
```

Anexo

Para aquellas personas que necesiten utilizar el programa mipsx desde Windows e internet pueden instalar putty y xming y conectarse.

Instrucciones :

1. Bajar putty : http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
2. Instalar xming : http://sourceforge.net/projects/xming/
3. En las opciones de instalación marcar "No instalar el cliente ssh"
4. Una vez instalado se debe tener "en ejecución" el programa xming (debería estar el ícono en la barra de programas de ejecución). Con xming en ejecución ejecutan putty que bajaron previamente.
5. Este paso es importantísimo. Si no se tiene en EJECUCIÓN el programa xming, las instrucciones que siguen debajo no funcionan:
    5.1. Colocan como nombre de host : aula-ssh.fi.uncoma.edu.ar
    5.2. PORT : 60173
    5.3. En las opciones de SSH -> X11 : habilitan "X11 forwarding".
    5.4. Le dan click a "open". En ese punto putty se conecta y les pide el nombre de usuario y la clave. Utilizan su usuario y clave de los laboratorios (la clave no aparece cuando la escriben).
    5.5. Una vez que ingresan pueden ejecutar `mipsx` con el comando: `/export/home/extras/mipsx/mipsx.sh`