Manual de Instalación y Utilización Máquina Simulada SPARC

Proyecto Alfa1
Simulación de un procesador SPARC con fines educativos
http://www.sce.carleton.ca/faculty/wainer/usenix/homepage.html
Director: Dr. Gabriel Wainer
Luis Fernando De Simoni (ldesimon@dc.uba.ar)

3 de agosto de 2001

Parte I

Resumen

Este manual tiene como finalidad detallar todos los pasos necesarios para realizar la instalación de la maquina simulada SPARC sobre los sistemas operativos UNIX, en particular se detallan los pasos sobre un sistema LINUX. Este manual es autocontenido, no se requieren conocimientos de sistemas UNIX para su instalación ya que se detalla paso a paso su instalación, no obstante algunos conceptos no se desarrollan en su totalidad. Este documento puede obtenerse y redistribuirse realizando la mención a la página principal del proyecto [1]. Asismísmo se detalla la instalación de un ensamblador para la máquina, un compilador, la compilación de la máquina simulada, la creación de archivos assembler para esta máquina, su proceso de ensamblado, compilación y ejecución en la maquina simulada. Luego se analiza los resultados obtenidos del vuelco de memoria como asi también la ejecución de tests sobre la maquina con la ayuda de una herramienta.

1 Instalación en sistemas UNIX

1.1 Requerimientos

Para la instalación en sistemas UNIX, se debe contar con aproximadamente 75 Mb. en disco, se recomienda utilizar procesadores con velocidad mayor a 166 Mhz. Se han de definir para este documento dos directorios absolutos:

- [alfa1-dir] *Directorio donde se instalara la maquina (Ej: /home/alfa1*

[1] http://www.sce.carleton.ca/faculty/wainer/usenix/homepage.html

- [binutils-dir] *Directorio donde se instalara las binutils (Ej: /usr/local/bin/sun4*[2]

Luego, se haran referencias absolutas al ejecutar comandos realizando la referencia *[alfa1-dir]* o *[binutils-dir]*

1.2 Obtención del código fuente

La última versión del código fuente de la máquina simulada puede obtenerse en la página web del proyecto. De este lugar podra obtenerse el archivo *alfa1-dist.tar.gz*, en formato compactado, para su instalación.

1.3 Instalación del código fuente

Utilizaremos el directorio *[alfa1-bin]* para descompactar el código fuente de la máquina simulada, el archivo *alfa1-dist.tar.gz* se debe encontrar en el directorio *[alfa1-dir]* y sobre este directorio ejecutar el siguiente comando: [3]

```
[user@machine [alfa1-dir]]$gunzip -c alfa1-dist.tar.gz | tar xvf -
```

Una vez que se ha ejecutado este comando se creará sobre *[alfa1-dir]* un directorio *alfa1*.

Luego, debemos compilar el código fuente, para ello ingresamos al directorio *./alfa1/source* y dentro de este directorio encontraremos el archivo *makefile* este archivo contiene la información que requiere nuestro compilador. Hemos utilizado para la compilación las herramientas provistas por GNU, en particular la versi'on *egcs 2.91.66 19990314*, no obstante se ha probado exitosamente la compilación con versiones anteriores.

Para compilar el código fuente entonces ejecutamos el comando:

```
[user@machine [alfa1-dir]/alfa1/source]$make
```

Esto generará el archivo *alfa1-simu* en el directorio *[alfa1-dir]/alfa1/source*.

1.4 Ensamblador y Linker

Para poder utilizar la máquina simulada debemos poder alimentarla con los programas, para generar estos archivos desde los archivos assembler utilizaremos un ensamblador y un linker.

En primer lugar debemos conseguir el código fuente de un ensamblador y de un linker, para poder generar código objeto para nuestra arquitectura. Afortunadamente se pueden conseguir el código fuente de estos programas y configurarlo para que en nuestra arquitectura genere código para la arquitectura simulada (del tipo SUN4).

El paquete de herramientas que nos permite hacer esto son las *binutils* [4]

[2] No se deberia utilizar /usr/local/bin como directorio

[3] Debe tenerse especial atención en poseer el permiso correspondiente de escritura sobre el directorio en el cual se descompacta el archivo y lectura sobre el archivo.)

[4] ver apéndice para mas información sobre donde obtener el paquete binutils

En nuestro caso utilizaremos las bintuils versi'on 2-10 [5] debemos descompactar este archivo, para ello copiaremos este archivo al directorio *[binutils-dir*] y sobre este directorio ejecutamos:

```
[user@machine [binutils-dir]$gunzip -c binutils-2.10.tar.gz | tar xvf -
```

De esta manera nos creará un directorio *binutils-X.Y* [6] en *[binutils-dir]* con el código fuente. Una vez que lo hemos descompactado debemos configurar el paquete para que opere en nuestra plataforma y en particular debe, como resultado de la compilacion de archivos assembler, generar código SUN4, para ello utilizaremos dentro del directorio *[binutils-dir]/binutils-X.Y* el comando:

```
[user@machine [binutils-dir]/binutils-X.Y$./configure --target=sun4
```

Esto configurará el código fuente de forma óptima para nuestra arquitectura y para que en nuestro assembler y linker generen código sun4, generando el *Makefile* apropiado.

Una vez que hemos logrado configurarlo debemos compilar las binutils. Esto se realiza con el comando:

```
[user@machine [binutils-dir]/binutils-X.Y]$make
```

posicionados sobre el directorio sobre el cual hemos ejecutado el *./configure*. Se debe tener especial cuidado en no ejecutar el comando *make install*, de ser asi se instalaría en nuestro directorio */usr/local/bin/* y */usr/local/lib* reemplazando al ensablador y al linker de nuestro sistema.

Luego de haber ejecutado el make, y haber logrado compilar nuestro ensamblador y linker, para poder ejecutar cómodamente desde cualquier directorio debemos crear un link simbólico al ensamblador y al linker[7]. En nuestro caso el programa de ensamblado es *as* y el linker *ld*. Debemos buscar estos archivos dentro del directorio *[binutils-dir]/binutils-X.Y*, para ello:

```
[user@machine [binutils-dir]/binutils-X.Y]$find -name as

[user@machine [binutils-dir]/binutils-X.Y]$find -name as*
```

En nuestro caso se han encontrado dentro del directorio relativo */as* y */ld*. Para crear un link simbólico utilizaremos el comando UNIX *ln*. Llamaremos al ensamblador para nuestra arquitectura *alfa1-as* y al linker *alfa1-ld*.

```
[user@machine [binutils-dir]/binutils-X.Y]$ln -s /usr/local/bin/alfa1-as
 [binutils-dir]/binutils-X.Y/[gas]/as
```

[5] Recordar que se ha tomado como convension que los números impares luego del primer punto del número de version son inestables, por lo que se recomienda tomar las versiones de número par

[6] Donde X.Y representa el número de version

[7] Este paso es opcional pero muy recomendado

[8] Luego hacemos lo mismo para el linker:

```
[user@machine [binutils-dir]/binutils-X.Y]$ln -s /usr/local/bin/alfa1-ld
 [binutils-dir]/binutils-X.Y/[gas]/ld
```

[9]

De esta manera si vemos el directorio */usr/local/bin* tendremos:

```
lrwxrwxrwx 1 [user] [user] 48 alfa1-as -> [binutils-dir]/binutils-X.Y/[gas]/as
lrwxrwxrwx 1 [user] [user] 47 alfa1-ld -> [binutils-dir]/binutils-X.Y/[gas]/ld
```

Es decir que ahora tendremos en el path de nuestro sistema el programa *alfa1-as* y *alfa1-ld*.

1.5 Proceso de ensamble y link-edición de código assembler

Para comenzar compilaremos algunos ejemplos que se adjuntan con la distribución de la máquina simulada. Dentro del directorio *[alfa1-dir]/alfa1/tests* se encuentran ejemplos.

Tomaremos el ejemplo *store.s*

Archivo 1 Archivo store.s

```
set 0x12345678, %r1
st %r1, [dest]
sth %r1, [dest+4]
sth %r1, [dest+10]
stb %r1, [dest+12]
stb %r1, [dest+17]
stb %r1, [dest+22]
stb %r1, [dest+27]
unimp
dest: .ascii
```

Primero realizaremos el ensamble del archivo, ejecutando sobre el directorio que contiene el archivo *store.s*

```
[user@machine [alfa1-dir]/alfa1/tests]$alfa1-as store.s -o store.ens
```

Luego el linker:

```
[user@machine [alfa1-dir]/alfa1/tests]$alfa1-ld store.ens -o store.map -Ttext 0x20
```

Esto creará en este directorio el archivo *store.map* a ser utilizado por la máquina simulada.

[8] Donde [gas] es el directorio relativo donde se encuentra el archivo ejecutable as

[9] Donde [gas] es el directorio relativo donde se encuentra el archivo ejecutable ld

1.6 Configuración para ejecución de la máquina simulada

Una vez que hemos creado el .map debemos configurar la máquina para que utilice una memoria de x cantidad de bytes y que utilice este archivo (*store.map*) como entrada. Para ello editaremos el archivo *iu.ma* este se encuentra dentro del directorio *[alfa1-dir]/alfa1/source/coupled/alfa1* En este archivo se define:

```
[mem]
preparation : 0:0:0:50
memsize : 256
memfile : call.map
dumpfile : memory.dmp
```

donde *memsize* es nuestro tamaño de la memoria en la simulación, por defecto 256 bytes, luego nuestro archivo de entrada (aquí editaremos y pondremos *store.map*) y el archivo de vuelco de memoria de salida, por defecto *memory.dmp* [10].

1.7 Simulación

Una vez que hemos realizado estos cambios y guardado el archivo podemos comenzar la simualción, el ejecutable de la maquina simulada es *alfa1-simu*, dentro del directorio *[alfa1-dir]/alfa1/source/*.

Ejecutaremos:

```
[user@machine [alfa1-dir]/alfa1/source]$alfa1-simu -m[alfa1-dir]/coupled/alfa1/iu.ma  -o
```

[11] Esto ejecutara la simulación y gracias a la opción *-o* veremos el estado del PC [12] la opción *-m* ubica el archivo de definición de acoplamiento antes editado.

Como resultado se obtendrá un archivo llamado *memory.dmp* con el estado de la memoria al finalizar la ejecución, este archivo tendrá el tamaño en bytes que ha utilizado la memoria.

1.7.1 Parámetros adicionales alfa1-simu

- -l*archivo* Permite crear un archivo con los mensajes entre los componentes at'omicos del simulador
- -o*archivo* Crea un archivo con la salida del simulador, si no se especifica archivo la salida standard es utilizada
- -h Muestra los comandos adicionales del simulador

[10] Se recomienda utilizar memory.dmp
[11] No se deben dejar espacion entre la opción m y el directorio.
[12] Program Counter

2 Test de la máquina

Aunque se han realizado test de las operaciones de la máquina simulada, es posible realizar un test de la ejecución verificando el resultado en forma sistemática para un conjunto de archivos.

2.1 Ejecuión del test para múltiples archivos

Para realizar el test deberemos generar automáticamente muchos archivos de código assembler, esto se deja a su criterio.Por ejemplo se puede realizar archivos assembler cada uno realiza una suma de dos enteros, el código fuente del algoritmo de suma seria:

Archivo 2 Suma de dos enteros entre el registro 27 y el registro 4(add27-4.s)

```
!Reliza la suma entre dos registros y guarda el resultado en memoria
!Archivo: add27-4.s

set 8134766, %r27 !8134766 en reg 27
set 9720765, %r4 !9720765 en reg 4
add %r4, %r27, %r10 !realizo la suma resultado en reg 10
st %r10, [dest] !Guardo el resultado en memoria

unimp

.align 4
valor : .ascii "VALOR:"
dest : .word FFFFFFFF !Resultado del Test 11
```

Siendo los valores *8134766*, *9720765*, y los registros todos valores aleatorios[13], luego este resultado es guardado en una posición de memoria, con la etiqueta VALOR: antes de este resultado, como al generar este archivo sabemos el resultado de esta suma[14] , al mismo momento que creamos este archivo *add27-4.s* creamos el archivo *add27-4.tst*:

Archivo 3 Suma de dos enteros entre el registro 27 y el registro 4(add27-4.s)

```
VALOR:int32,8052980
```

siendo *8052980* [15] el resultado esperado luego de la ejecución de la simulación. Una vez que se ha ejecutado la simulación se puede utilizar un programa para verificar este resultado automáticamente, esto es muy útil para realizar sobre un conjunto de archivo, por lo que el programa retorna un archivo con

[13]recordar que el número de registros de uso general se encuentra acotado por 32

[14]El generador de archivos debe ser lo suficientemente inteligente como para predecir exactamente el resultado

[15]Se debe tener especial cuidado con el overflow o underflow dependiendo de la operación

los nombres de archivo y el valor OK si el valor esperado es el obtenido por la simulación y DIF con la diferencia númerica en el caso de existir un error, para cada uno de los archivos de este conjunto de test. Se ha utilizado con este fin el programa para entorno win32 *tester* [16]

para verificar sobre todo un directorio se debe ejecutar:

```
tester *.map *.tst > result.log
```

Obteniéndose el archivo result.log con el resultado detallado de las simulaciones y por cada archivo que se han encontrado diferencias se creara el archivo *nombre de archivo.dif*

De esta manera se pueden verificar de una manera sistemática un conjunto de instrucciones.

3 Apéndice

3.1 Obtención de Herramientas

http://www.sce.carleton.ca/faculty/wainer/usenix/homepage.html

3.2 Bibliografia recomendada

Referencias

[1] Sun Microsystem, Assembler User Documentation,

[2] Sparc Manual Reference
http://docsun.cso.uiuc.edu:80/cgi-bin/nph-dweb/ab2/coll.45.13/SPARC/@Ab2PageView/10399?

[16] Desarrollado por Pablo Revert(prevert@dc.uba.ar) y Luis Fernando De Simoni(ldesimon@dc.uba.ar) (Ver apéndice de programas)

3.3 Directorios alfa1

```
.
|-- source
|   |-- atomic
|   |   |-- common
|   |   `-- models
|   |       |-- ALU
|   |       |-- AND
|   |       |-- Adder
|   |       |-- AlignL
|   |       |-- AlignS
|   |       |-- Bus
|   |       |-- CCLogic
|   |       |-- CS
|   |       |-- CWPlogic
|   |       |-- Clock
|   |       |-- Cmp
|   |       |-- FromDeci
|   |       |-- IRQLogic
|   |       |-- Inc4
|   |       |-- IncDec
|   |       |-- Latch
|   |       |   `-- old
|   |       |-- Mem
|   |       |-- MulDiv
|   |       |-- Mux
|   |       |-- MuxN
|   |       |-- NOT
|   |       |-- RegBlock
|   |       |-- RegGlob
|   |       |-- Shifter
|   |       |-- SignExtN
|   |       |-- TestIn
|   |       |-- TestOut
|   |       |-- ToDeci
|   |       |-- TrapLogic
|   |       |-- UC
|   |       |   `-- bck
|   |       |-- UCTest
|   |       `-- WIMCheck
|   |-- coupled
|   |   `-- alfa1
|   `-- simu
`-- test
```

Índice General