Organización del Computador
Descripción de ARCSim y ARCAssembler

24 de abril de 2000

1 El Simulador ARC

1.1 ¿Qué es el Simulador ARC?

El Simulador ARC es una aplicación Java 1.1 que simula la ejecución de un programa en el lenguaje de máquina ARC. Es un simulador "funcional" dado que sólo simula a nivel ISA el comportamiento de la arquitectura ARC. Sirve para mostrar la ejecución de los programas ARC, pero no contiene información alguna sobre tiempos, interrupciones, retardos, excepciones, manejo de memoria, estados internos de la CPU, etc. Esta se presenta como una máquina de estados finitos, donde el estado está representado por la memoria y los registros y las transiciones ocurren de acuerdo a la palabra de la memoria apuntada por el *Program Counter*.

1.2 ¿Qué computadora y sistema operativo son necesarios para correr este programa?

Este simulador está escrito en Java 1.1, por lo tanto cualquier combinación arquitectura/sistema operativo que soporte la ejecución de código Java podrá correr este programa. Ha sido probado en x86/Win9x, x86/Linux y ppc/MacOS.

1.3 ¿Cómo ejecuto el simulador?

Al no ser un Applet, esta aplicación Java no funciona desde un Navegador Web, y se deberá tener una JRE (Java Runtime Enviroment) para la arquitectura/sistema operativo usado.

Tanto para x86/Win9x como para x86/Linux basta con invocar desde la línea de comandos[1]:

```
jre -cp path/to/ARCTools.jar ARCTools/ARCSim
```

Al finalizar la carga aparecerá la pantalla principal con el título **ARCSimulator v1.xx**. Esta se divide en 4 zonas:

- Una zona de campos editables que muestran el contenido de los registros `%pc`, `%psr` y los 32 registros numerados desde `%r0` hasta `%r31`. Los registros `%pc`, `%psr` y `%r0` no son modificables.
- Dos filas de botones de control.

[1]También hay que establecer las variables de entorno `PATH` y `CLASSPATH` apropiadamente, pero esto normalmente es realizado por los paquetes instaladores de JRE o de JDK (Java Development Kit).

- Ocho localizaciones de memoria consecutivas, botones para establecer puntos de ruptura (*break-points*) en esas direcciones, las palabras almacenadas en cada una de ellas y el código assembler correspondiente a esa palabra de memoria.

  También se incluyen 6 botones para desplazar la ventana de memoria que estamos observando.

- En la parte inferior de la pantalla se muestran posiciones de memoria que si pueden ser editadas, asi como una ventana de texto con mensajes de error.

1.4 ¿Qué tipo de archivos de entrada acepta el simulador ARC?

El simulador ARC acepta un formato especial de entrada que sería el equivalente de un código binario ejecutable en representación ASCII. Esta representación facilita la inspección del código ejecutable por cualquier tipo de editor o visor de archivos de texto.

El formato es muy sencillo: cada línea contiene 2 columnas, ambas son palabras de 32 bits en hexadecimal, donde la primera indica la dirección de memoria y la segunda el contenido de esa dirección.

La forma usual de generar estos archivos, que tendrán extensión `.bin`, es a través del **ARCAssembler**, el cual se accede desde el botón **Edit** de la ventana principal.

1.5 ¿Cómo corro programas en el simulador?

Utilizando los botones del centro de la pantalla principal. La funcionalidad de cada uno se describe a continuación:

Exit Termina la aplicación ARCSim.

Print Imprime una instantánea de la ventana principal. *(¿Funciona?)*

Load Permite la carga de un archivo binario ARC.

Reload Recarga el archivo binario ARC actual, reinicializando la memoria, registros y puntos de ruptura.

Edit Muestra la ventana de edición de programas ARC en código assembler.

Step Ejecuta la instrucción apuntada por `%pc` y se detiene. Esta instrucción está resaltada en la parte central de la pantalla.

Run Comienza a ejecutar instrucciones ARC desde `%pc`. Sólo se detiene cuando se encuentra una instrucción `halt` o un punto de ruptura en una posición de memoria. También puede ser detenido con el botón Stop.

Stop Detiene una ejecución iniciada con Run.

Hex↔Dec Cambia la representación de los datos del ARCSim de base-16 a base-10 y viceversa.

Clear RegFile Pone a 0 todo el conjunto de registros.

Clear BreakPts Borra todos los puntos de ruptura que pudiera haber en la memoria.

Clear Memory Escribe toda la memoria con el valor 0.

2 El Ensamblador ARC

2.1 ¿Qué es el ARCAssembler?

El ARCAssembler es un ensablador simbólico de 2 pasadas que ensabla programas escritos en el lenguaje de máquina ARC (`.asm`) y genera el código ejecutable en su representación hexadecimal (`.bin`) junto con un archivo de listado (`.lst`) donde se muestra el código fuente junto al código objeto.

El lenguaje ensamblador soporta referencias simbólicas y expresiones aritméticas en tiempo de compilación. Los comentarios se inician en cualquier lugar de una línea con un signo de exclamación (`!`) y finalizan con el EOL.

2.2 ¿Qué instrucciones están soportadas?

Todo el conjunto de instrucciones dadas en el libro (`ld`, `st`, `sethi`, `and{cc}`, `or{cc}` `orn{cc}`, `srl{cc}`, `add{cc}`, `call`, `jmpl`, `be`, `bneg`, `bcs`, `bvs`, `ba`) junto con estas otras instrucciones: `xor{cc}`, `sll{cc}`, `sra{cc}`, `nop` y `halt`.

En la figura 1 de la página 4 vemos una tabla que muestra de manera resumida la funcionalidad de cada instrucción.

2.3 ¿Qué pseudo-ops acepta?

La situación es inversa a la de los opcodes, pues del conjunto mostrado en el libro sólo se implementan:

`.equ` Asigna valores a símblos. Por ejemplo podemos hacer[2] `TRUE .equ 1`.

`.begin` Comienza a ensamblar.

`.end` Finaliza de ensamblar.

`.org` Cambia el contador de locación.

`.dwb` Reserva un bloque de palabras.

Quedan afuera `.global`, `.extern`, `.macro`, `.endmacro`, `.if` y `.endif` cuyas funcionalidades permiten darle un poco de estructura y *programming-in-the-large* al código ensamblador.

2.4 ¿Cómo ensamblo un programa escrito en ARC Assembler?

El ensablador ARC se accede desde el simulador ARC presionando el botón Edit. Cuando esto sucede un editor de texto será mostrado. Si un archivo binario había sido cargado, entonces el archivo asociado `.asm` será abierto, en caso de no haber existido un binario en el simulador, el editor estará vacío y podremos escribir un nuevo programa para ARC.

Con el programa listo en la ventana del editor basta con presionar Assemble, lo que producirá una llamada al ensamblador con el archivo `.asm` que estamos trabajando en el editor. En caso de una compilación exitosa, se generarán los archivos `.lst` y `.bin` asociados.

En el cuadro de texto que está por debajo del editor, se muestran los mensajes generados por el ensamblador.

[2] Sin embargo el código `%sp .equ %r14`, que resulta súmamente práctico para codificar más claramente programas que utilizan la pila, no es aceptado por el ensamblador. No obstante existe una declaración interna que permite escribir `%sp` y `%r14` de manera indistinta.

Instr.	Semántica	Ejemplo	Significado
`nop`	No Operation, sólo incrementa el `%pc` para que apunte a la siguiente instrucción		
`halt`	Para la ejecución del programa		
`sethi`	Establece los 22 bits más significativos y pone a 0 los 10 de menor orden	`sethi 1111b, %r1`	Establece a 1 los bits 10 al 13 del registro `%r1`
`be` (`bne`)	Salta si `%psr.z` $= 1(0)$	`be isEqual`	Salta a `isEqual` si `z` es 1
`bcs`	Salta si `%psr.c` $= 1$	`bcs isCarry`	Salta a `isCarry` si `c` es 1
`bneg` (`bpos`)	Salta si `%psr.n` $= 1(0)$	`bneg isNegative`	Salta a `isNegative` si `n` es 1
`bvs`	Salta si `%psr.v` $= 1$	`bneg isOverflow`	Salta a `isOverflow` si `v` es 1
`ba`	Salta siempre	`ba loop`	Salta incondicionalmente a `loop`
`call`	Salta al inicio de una subrutina y almacena el `%pc` corriente en `%r15`	`call add64`	Llama a la subrutina `add64`
`add{cc}`	Suma los operandos fuente en el operando destino. El sufijo `cc` indica que se modifica el `%psr` de acuerdo al resultado	`add %r1,%r2,%r6` `add %r6,-2,%r6`	`%r6:=%r1+%r2` `%r6:=%r6+(-2)`
`sub{cc}`	Resta utilizando artimética de complemento a 2's	`subcc %r1,%r2,%r6` `sub %r1,0xA,%r2`	`%r6:=%r1-%r2` `%r2:=%r2-10`
`and{cc}`	Bit a bit AND	`andcc %r0,%r0,%r1` `and %r1,0xFF,%r1`	Pone a 0 `%r1` y `%psr` Enmascara los 8 LSB de `%r1`
`or{cc}`	Bit a bit OR	`orcc %r1,%r2,%r4` `or %r1,1,%r1`	`%r4:=%r1 OR %r2` Pone a 1 el LSB de `%r1`
`orn{cc}`	Bit a bit NOR	`orn %r1,%r0,%r1`	Complemento de `%r1`
`xor{cc}`	Bit a bit XOR	`xorcc %r1,%r2,%r0`	Compara `%r1` y `%r2`
`srl{cc}`	Desplaza a la derecha en 0-31 bits. Las posición libre se completa con 0 y el sufijo `cc` indica que `%psr` se modifica con el resulado	`srl %r1,%r2,%r1`	Desplaza `%r1` `%r2` veces a la derecha
`sll{cc}`	Desplaza a la izquierda en 0-31 bits. Las posición libre a la derecha se completa con 0	`sll %r1,%r2,%r1`	Desplaza `%r1` `%r2` veces a la izquierda
`sra{cc}`	Desplaza a la derecha en 0-31 bits. El bit de signo es replicado en la posición vacante más a la izquierda	`sra %r1,%r2,%r1`	Desplaza artiméticamente `%r1` `%r2` veces a la derecha

`ld`	Carga un registro con el contenido de una posición de memoria que debe estar alineada con una palabra	`ld [x], %r1`
		`ld %r0+x, %r1`
		`ld [x],%r0,%r1`
		`ld %r0,x,%r1`
			`%r1:=M[%r0+x]`

`st`	Almacena un registro en la memoria. La posición de memoria debe estar alineada con una palabra	`st %r1, [x]`
		`st %r1,%r0+x`
		`st %r1,%r0,x`
			`M[%r0+x]:=%r1`

`jmpl`	Control de transferencia incondicional controlado por registro. Salta a una dirección nueva y almacena la actual	`jmpl %r15+4,%r2`	Almacena la dirección actual en `%r2` y salta a `%r15+4`

Figura 1: Conjunto de instrucciones del ARC ISA.

2.5 ¿Cuál es la funcionalidad de los otros botones en el ARCAssembler?

Show Asm File Muestra el código fuente del archivo cargado.

Show Lst File Muestra el archivo de listado (memoria, código objeto y código fuente) correspondiente al ensamblado del archivo cargado.

Show Binary File Muestra el código binario que resulta del ensamblado.

Bin → Sim Carga el código binario en el simulador.