QSim: Simulador de arquitecturas Q

Susana Rosito Tatiana Molinari

6 de noviembre de 2013

Índice general

Capítulo 1

Introducción

Capítulo 2

Contexto

2.1. Sobre la materia Organización de Computadoras

2.2. Conceptos importantes

2.3. Arquitecturas Q

2.3.1. Características generales

2.3.2. Repertorio de instrucciones

Instrucciones de 2 operandos

Instrucciones de 1 operando origen

Instrucciones de 1 operando destino

Instrucciones sin operandos

Instrucciones de salto condicional

2.3.3. Versiones de la arquitectura

Q1

Q2

Q3

Q4

2.4. Estado del arte

Capítulo 3

Simulador QSim

3.1. Funcionalidad del simulador

La funcionalidad del simulador puede caracterizarse mediante las siguientes partes importantes:

- Chequeo de sintaxis de los programas escritos en el lenguaje Q
- Ensamblado del código fuente de un programa en su correspondiente código máquina
- Cargado en memoria del código máquina
- Ejecución paso a paso de un programa cargado en memoria

3.1.1. Chequeo de sintaxis

El simulador provee al alumno de un editor de texto en el cual escribirá el programa en un lenguaje Qi, que desea cargar en memoria y ejecutar. Una vez que el usuario haya terminado la escritura, al momento de cargar el programa, el simulador utilizará un parser para detectar errores de sintaxis, tales como la falta de una coma o un corchete, o la presencia de símbolos que no pertenecen al lenguaje (como por ejemplo signos de pregunta y símbolos matemáticos); o bien errores semánticos como la combinación incorrecta de elementos del lenguaje, por ejemplo: modos de direccionamiento mal ubicados. El parser solo revisará lo escrito por el alumno y de acuerdo a las gramática del lenguaje, mostrará alguno de los siguientes estados:

OK Este mensaje se obtiene cuando no hubo ningún error de sintaxis. Si se da este resultado, es posible continuar con el ensamblado y cargado en memoria.

SyntaxError Este mensaje de error se obtiene cuando en alguna línea del programa se detectó algún error de sintaxis o de semántica, como se describió arriba. Cuando ocurre este error se lo acompaña con una descripción lo mas detallada posible para que el alumno detecte donde ocurrió y pueda corregirlo. Un programa con errores no puede ser ensamblado y cargado en memoria.

3.1.2. Ensamblado

Una vez que el programa es sintácticamente válido es posible traducir el código fuente del programa en código máquina (representado en cadenas binarias). Para esto se respeta un formato de instrucción que indica cómo se codifica cada operación y los operandos.

Mas detalle al respecto de este proceso en la sección de ...[1]

3.1.3. Cargado en memoria

Una vez ensamblado, la representación binaria (o código máquina) del programa será cargado en memoria a partir de una ubicación (celda de memoria) que el alumno puede elegir. Esto permite visualizar el contenido de la memoria (con el programa cargado) y el estado de los registros de la CPU. La decodificación con desensamblado permite al alumno experimentar otros escenarios y efectos laterares,por ejemplo[2]:

- Si la ejecución paso a paso excede los límites del programa, pueden tomarse instrucciones de otra rutina y procesarse como una nueva instruccion.
- Si en cambio, se intenta ejecutar el contenido de una celda con datos (y no una instrucción) podrá ocurrir que se encuentre una instrucción invalida (combinacion de modos, codigos, incorrecta) y el alumno verá el mensaje de error pertinente.

Durante la carga del programa en memoria puede producirse un OutOfMemoryError?, que significa que el programa no entra en la ubicación elegida en memoria ya que ocupa más celdas que las que se encuentran disponibles debajo de la[3] inicial anteriormente elegida. Si por el contrario, no se produce este error, el alumno podrá ver el programa cargado en memoria exitosamente.

3.1.4. Ejecución paso a paso

Se provee la funcionalidad de la ejecución paso a paso ya que se desea que el alumno pueda experimentar y así comprender los pasos del ciclo de ejecución de una instrucción[4]. Además puede ejercitarse situaciones que se denominan "errores conceptuales de programación" Esto es a lo que llamamos ConceptError?. Por ejemplo:[5]

- Tomar un dato de un sector de memoria equivocado.
- Que el programa sobrescriba su mismo código máquina.
- Permitir que la ejecución continue una vez terminado el programar cargado en memoria.

[1] Poner vínculo a donde se ponen ejemplos de Qi

[2] entre los cuales podemos enumerar

[3] (Hablar de la estructura de la memoria:) Como se mencionó en la sección 2.3.1, la memoria disponible tiene un tamaño limitado... Por este motivo la alocación en memoria del código máquina puede exceder el espacio disponible a partir de la celda ...

[4]

[5] Errores conceptuales, entre los cuales es posible mencionar:

-

El paso a paso que provee el simulador consiste en las siguientes estapas pertenecientes al ciclo de instruccion:

1. **FI (Fetch de instruccion)**[6]**:** El alumno podrá visualizar el valor que contiene PC (Program counter) donde se encuentra la dirección de la celda en memoria que contiene la próxima instrucción a ejecutar (por ejemplo, en caso de ser la primer instrucción del programa recién cargado, el pc tendrá la dirección de memoria elegida por el alumno para iniciar el cargado del programa en memoria). El simulador, toma de la memoria el código maquina correspondiente a la instrucción que comienza en esa dirección tomada de PC (una instrucción puede ocupar más de una celda de memoria) y los guarda en el registro IR (*Instruction Register*). Será observable también para el alumno el incremento del registro PC, tantas como [7] ocupe la instrucción actual, lo que conceptualmente es, prepararse[8] para tomar la siguiente instrucción.

2. **Decodificación:** En la decodificación el Interprete,se encarga de tomar lo que se encuentra en IR en representación hexadecimal y desensamblado[9] para mostrar el código fuente de la instrucción actual con sus respectivos operandos. Si el programa escrito por el alumno es sintacticamente y conceptualmente correcto, este paso le permite comprobar que la instrucción actual es la que él mismo escribió y no otra, visualizandola en pantalla. En esta etapa se provee también la oportunidad de que el alumno aprecie otros conceptos, tales como el anteriormente mencionado ConceptError?[10].

3. **EX Execute**[11] La ejecución lleva a cabo[12] los efectos de la instrucción y muestra en pantalla los cambios en el estado [13]: memoria, puertos, registros y flags. Dentro de esta misma etapa se encuentra el ST (Store)[14] que, cuando sea necesario, guardará el valor resultante de la operación descripta por la instrucción en el operando destino. Esto cambiará el valor de una celda de memoria o de un registro y será visto en pantalla por el alumno.

3.2. Implementación del simulador

3.2.1. Tecnología utilizada

3.2.2. (arq OO)

[6] Búsqueda de instrucción
[7] celdas
[8] preparar el contexto de ejecución
[9] se encarga de desensamblar el código máquina (abreviado en hexadecimal) que ya fue ubicado en el registro IR
[10] los errores conceptuales mencionados antes
[11] Ejecución
[12] El execute ejecuta
[13] de ejecución
[14] lleva a cabo el almacenamiento de resultados

Capítulo 4

Evaluación del desarrollo

4.1. Dificultades encontradas

4.1.1. Dificultades presentadas por el dominio

Las dificultades del dominio estuvieron relacionadas a la comprensión no solamente del modelo de arquitectura Q si no también a su función[1] didáctica[2], ya que el objetivo del simulador no era solamente la implementación del lenguaje[3] sino también el proveer a los alumnos la capacidad de realizar cosas[4] conceptualmente erróneas, por lo que se requería una comprensión didáctica del problema más allá de la teoría[5] de las arquitecturas Q.

4.1.2. Dificultades de diseño

En primera instancia se opto por implementar un modelo de objetos que utilizaba un objeto de la clase **Programa** a lo largo de toda la ejecución (procesaba las instrucciones no leyendo de la matriz memoria, si no, pidiendo la siguiente instrucción al objeto instancia de la clase Programa), sobreviviendo así las distintas etapas una vez que fue creado y evitando la creación de un objeto cuya responsabilidad sea interpretar el código máquina alojado en la memoria[6]. Luego, al caer en la cuenta de[7] que un programa no sólo podía modificar su entorno al ser ejecutado (otras celdas de memoria que no ocupen su código maquina, celdas de puertos, registros, etc) si no que también podría sobreescribir su código maquina (ya sea con ese propósito o sólo por un ConcepError[8]), o bien, que el alumno debía tener la posibilidad de seguir ejecutando más allá del código máquina alojado en memoria o más, inevitablemente se opto por refactorear todo el[9] modelo

[1] propósito
[2] o
[3] es solamente la simulación de la arquitectura y la ejecución de programas,
[4] ejercitar situaciones
[5] especificación
[6] (sacar los paréntesis y detallar mas)
[7] entendimos
[8] error conceptual
[9] necesitó corregir gran parte del

agregando una clase denominada **Intérprete**, cuya responsabilidad es interpretar la siguiente instrucción alojada en memoria para que luego sea ejecutada, y descartando el objeto instancia de **Programa** una vez que éste es cargado en memoria con éxito.

Tuvieron que ser solicitadas además extensiones al equipo de desarrolladores de Arena para poder realizar la interfaz con dicho framework.[10]

4.2. Casos de prueba

4.3. Ejemplos de uso

[10] (detallar)

Bibliografía

[1] Williams Stallings, *Computer Organization and Architecture*, octava edición, Editorial Prentice Hall, 2010.

[2] A. Tanenbaum, *Organización de Computadoras*, cuarta edición, Editorial Pearson.

[3] Hennessy, Patterson. *Arquitectura de Computadores - Un enfoque cuantitativo*, primera edición, Editorial Mc Graw Hill.

[4] Sitio oficial de la materia Organización de Computadoras: `http:\orga.blog.unq.edu.ar` (2013)