Arreglos y recorridos

Organización de Computadoras 2019

Universidad Nacional de Quilmes

Mediante el lenguaje de Q podemos resolver muchos de los problemas que se abordan con los lenguajes de mas alto nivel, pero hasta ahora carece de una estructura de datos que permita representar una colección de datos.

1. Estructura de arreglos

Un arreglo es un conjunto homogéneo de valores. Esto quiere decir que todos los valores tienen la misma naturaleza (el mismo tamaño y tipo). Los arreglos ocupan un bloque de celdas de memoria consecutivas y cada uno de sus valores puede ocupar una sola celda, o podrían ser más grandes y ocupar varias celdas. El **tamaño** del arreglo se determina por la cantidad de elementos que tiene (notar que no siempre es la cantidad de celdas que ocupa).

Tamaño del arreglo

Por ejemplo, las ventas diarias de un comercio pueden organizarse dentro de un arreglo como el siguiente

	...
2000	0050
2001	008A
2002	00C5
2003	000F
	...

En la celda `2000` se tiene la cadena `0050`, que representa el valor 80, correspondiente a las ganancias del día lunes. En la celda `2001` se tiene la cadena `008A` que representa lo ganado el día martes ($138). Y así se pueden interpretar las siguientes celdas.

A la hora de trabajar con arreglos es necesario conocer dónde comienza y cuántos elementos tiene. Para lo segundo (tamaño del arreglo) es posible usar diferentes criterios, por ejemplo sabiendo la cantidad de elementos (de manera relativa), o conociendo la posición del último elemento (de forma absoluta) o bien estableciendo una condición de fin (un valor inválido). Ver figura 1 con ejemplos de estas estrategias.

Los arreglos son colecciones de datos que deben recorrerse para ser procesados, por ejemplo para acumular un resultado (sumatoria, promedio, etc) o para aplicarles a todos los elementos una transformación (aplicarles un descuento, invertir el signo, etc). Ejemplos de estos recorridos de arreglos pueden verse en la figura 2

Recorrido

Para llevar a cabo esta tarea de recorrer un arreglo es natural pensar en una repetición, donde en cada vuelta de ciclo se procese un elemento. Pensemos

Estrategia	**A** **Cant fija de elementos**	**B** **Fin en celda fija**	**C** **Condicion de fin**
Inicio	celda 2000	celda A000	celda 0090
Fin	4 elementos	celda A003	valor FFFF
Elementos	16b	16b	16b
	...	...	...
	2000 0050	A000 0050	0090 0050
	2001 008A	A001 008A	0091 008A
	2002 00C5	A002 00C5	0092 00C5
	2003 000F	A003 000F	0093 FFFF
	...	...	...

Figura 1: Ejemplos de arreglos

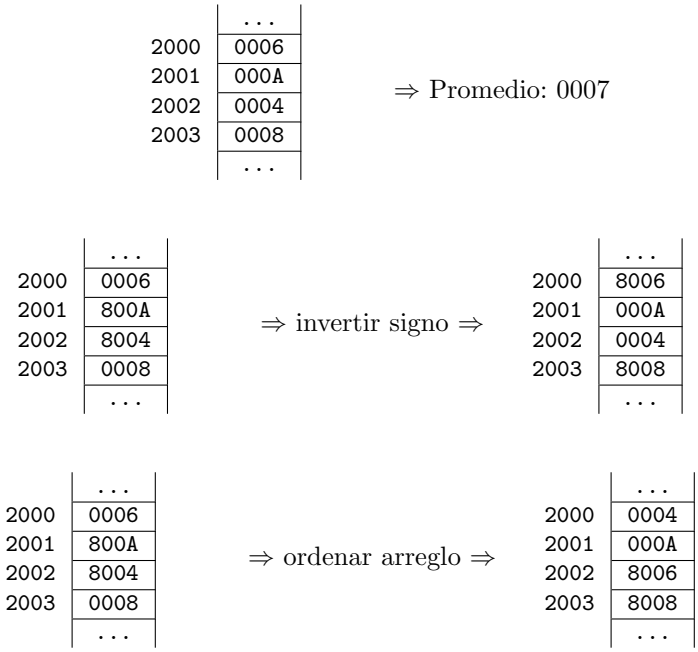


Figura 2: Ejemplos de recorrido

en el caso de calcular la sumatoria de los elementos del arreglo, asumiendo que contamos con el arreglo A de la figura 1:

Recorrido con problema

```
recorrido: MOV R0, 0x0000 <-- inicializar el acumulador
           MOV R1, 0x0000 <-- inicializar el contador
    ciclo: CMP R1, 0x0004
           JG fin
           ADD R0, [0x2000] <-- problema!
      fin: RET
```

En el enfoque anterior se ve claramente que con la instrucción `ADD R0, [0x2000]` siempre se suma el mismo elemento, llevándonos a pensar que necesitamos poder cambiar dinámicamente la dirección (en este caso `2000`) del elemento. Para solucionar esta problemática es necesario incorporar los modos de direccionamiento indirectos que se explican en el siguiente apartado.

2. Modos de direccionamiento indirectos

La motivación para incorporar un nuevo modo de direccionamiento a la arquitectura Q es la necesidad de un mecanismo de acceso que permita **indicar el operando mediante una expresión**. Una situación similar se vió antes en la gestión de la pila a través del registro SP o *stack pointer*.

En los modos de direccionamiento indirectos, el operando indicado en la instrucción es en realidad una dirección de un operando, y tiene dos posibles implementaciones: indirecto por registro e indirecto por memoria.

En la vida cotidiana encontramos ejemplos de indirecciones también. Supongamos que necesitamos saber donde se tramita la simultaneidad de carreras de programación y le preguntamos a lxs profes de orga. A pesar de que ellxs no tienen esa información, si saben quién la tiene y nos redirigen a preguntarle a la asistente de la carrera.

Indirecto por registro:

En este caso se indica el nombre de un registro que, en lugar de contenerlo, tiene la dirección del operando. Esto es aplicable al operando origen como al operando destino y la sintaxis necesaria para hacerlo es `[R0]`. Veamos el siguiente ejemplo:

Entonces el efecto de la ejecución de esta instrucción es

```
R0 <-- F678
```

Para analizar los accesos a memoria que provoca la ejecución de esta instrucción, es necesario primero poder ensamblarla y para eso se necesita agregar un código para este modo de direccionamiento (ver cuadro 1)

Modo	Codificación
Inmediato	000000
Directo	001000
Indirecto memoria	011000
Registro	100rrr
Indirecto registro	110rrr

Cuadro 1: Modos de direccionamiento en Q6

De esta manera, el código máquina de la instrucción es 0001 100000 110101 y comprimido en hexadecimal 1835. Esto quiere decir que esta instrucción ocupa una celda en memoria.

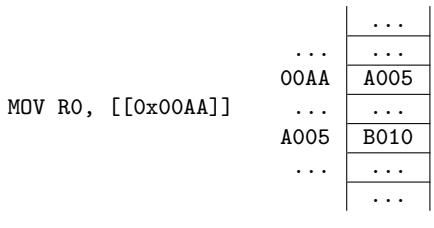
Volviendo al análisis de los accesos a memoria en las distintas etapas del ciclo de ejecución, se debe definir donde está ensamblada la instrucción. Asumamos que lo está en la celda 0000, y por lo tanto los accesos a memoria pueden describirse en un cuadro como el que sigue:

Instrucción	B.Inst.	B.Op.	Alm.Op.
MOV R0, [R5]	0000	A004	–

Es importante notar que el modo indirecto de esta instrucción provoca un acceso a memoria (en la celda A004 durante la búsqueda de operandos.

Indirecto por memoria:

De manera similar al caso de la indirección por registro, la indirección por memoria agrega un paso mas para alcanzar el operando, utilizando la sintaxis [[0x00AA]]. Nuevamente, este modo es aplicable tanto al operando origen como al destino. Veamos el siguiente ejemplo



Para llevar adelante el análisis del ciclo de ejecución es necesario analizar cómo es el código máquina de una instrucción que utilice este nuevo modo de direccionamiento. Teniendo en cuenta nuevamente el cuadro 1, vemos que la instrucción MOV R0, [[0x00AA]] se ensambla: 0001 100000 011000 0000 0000 1010 1010 y compactado en hexadecimal esto es 1818 00AA, es decir que el código máquina ocupa 2 celdas. Asumiendo que se lo ubica a partir de la celda 0000, confeccionamos un cuadro como el que sigue:

Instrucción	B.Inst.	B.Op.	Alm.Op.
MOV R0, [[0x00AA]	0000, 0001	00AA,A005	–

3. Recorrido de arreglos

Supongamos una situación donde se cuenta con el siguiente arreglo que contiene las edades de 4 niñxs, y donde cada edad (es decir cada elemento del arreglo) ocupa una celda:

	...
2000	0005
2001	0006
2002	0006
2003	0005
	...

Entonces supongamos que nos dicen que a partir de la celda `2000` está almacenado el arreglo y que su tamaño es 4. El objetivo es calcular la sumatoria de las edades de lxs niñxs, y para esto habíamos realizado un primer abordaje en la sección 1, cuando vimos que los modos de direccionamiento conocidos hasta entonces (directo, registro, inmediato) no eran suficientes para realizar el recorrido. A partir de contar con los modos indirectos, utilizaremos una variable (registro o celda de memoria) para llevar cuenta de la posición del elemento a procesar en cada paso del recorrido. Esta variable se denomina **índice del arreglo**.

Indice del arreglo

Con este objetivo, cargaremos en el registro `R2` el valor `2000`, pues esta es la dirección de inicio del arreglo. De esta manera la rutina es como sigue:

```
recorrido: MOV R0, 0x0000 <-- inicializar el acumulador
           MOV R1, 0x0000 <-- inicializar el contador
           MOV R2, 0x2000 <-- inicializar el INDICE
    ciclo: CMP R1, 0x0004 <-- Condición de fin
           JG fin
           ADD R0, [??]   <-- problema!
           ADD R1, 0x0001 <-- contar un elemento
           ADD R2, 0x0001 <-- mover el índice
      fin: RET
```

Como último desafío debemos denotar el acceso a cada elemento del arreglo dentro del ciclo (`problema`), y para esto utilizaremos un modo indirecto vía registro:

```
ADD R0, [R2]
```

Finalmente el código de la rutina `recorrido` se muestra a continuación:

```
recorrido: MOV R0, 0x0000 <-- inicializar el acumulador
           MOV R1, 0x0000 <-- inicializar el contador
           MOV R2, 0x2000 <-- inicializar el INDICE
    ciclo: CMP R1, 0x0004 <-- Condición de fin
           JG fin
           ADD R0, [R2]   <-- se acumula el elemento actual
           ADD R1, 0x0001 <-- contar un elemento
           ADD R2, 0x0001 <-- mover el índice
      fin: RET
```

Recorrer un arreglo de tamaño variable

Para este segundo ejemplo consideraremos que el tamaño del arreglo no es fijo por lo cual se delimita el arreglo con un valor especial `FFFF`, pues no tiene sentido que se almacene que algún niñx tiene 65.535 años de edad. Se debe recorrer el arreglo para calcular la sumatoria entre los valores almacenados a partir de la celda `2000` hasta el primer valor `FFFF` (y ese valor no debe sumarse).

```
recoVariable: MOV R0, 0x0000 <-- inicializar el acumulador
              MOV R2, 0x2000 <-- inicializar el INDICE
              MOV R1, [R2]   <-- Cargar el primer valor
       ciclo: CMP R1, 0xFFFF <-- Condición de fin
              JE fin         <-- Ojo: es otro salto
              ADD R0, R1     <-- se acumula el elemento actual
              ADD R2, 0x0001 <-- mover el índice
              MOV R1, [R2]   <-- Cargar un nuevo valor valor
     fin: RET
```