Máquina de Acceso Aleatorio

Diseño y Análisis de Algoritmos
Francisco Almeida

Una máquina de acceso aleatorio (Random Access Machine – RAM) modeliza un computador de un único acumulador, en el que las instrucciones no pueden modificarse a si mismas.

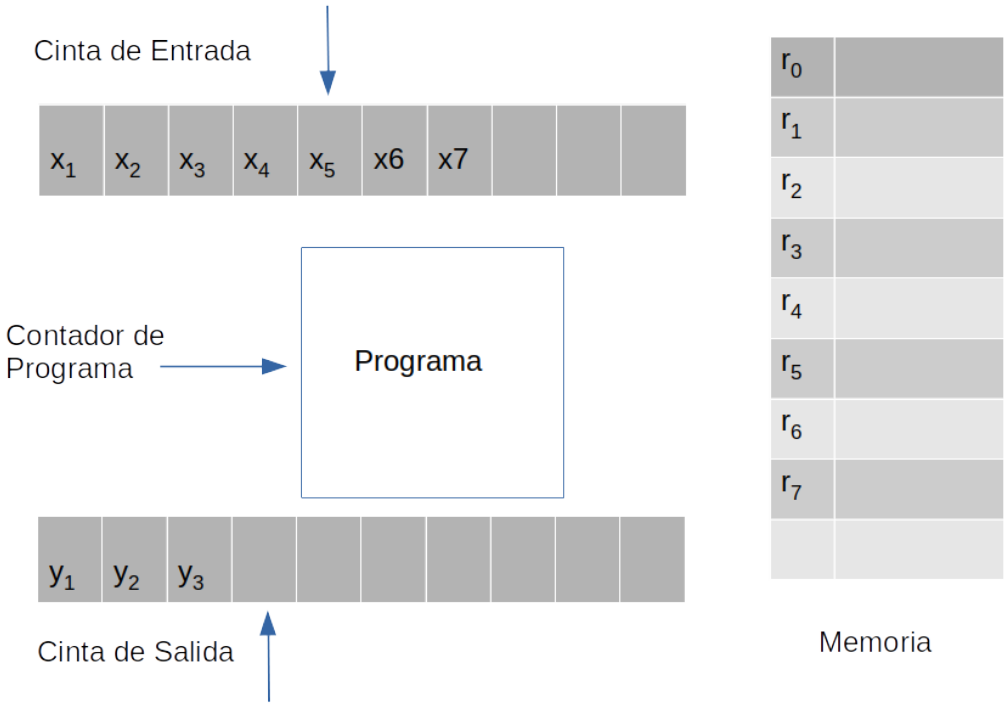
Una RAM consta de una cinta de entrada, solo de lectura, una cinta de salida, solo de escritura, un programa y una memoria.

La cinta de entrada es una sucesión de celdas; cada una de ellas puede almacenar un entero (Que puede ser negativo). Cuando un símbolo se lee de la cinta de entrada, la cabeza de lectura de la cinta de entrada se mueve a la derecha.

La cinta de salida, también formada por una sucesión de celdas que pueden almacenar enteros, inicialmente se encuentra con todas sus casillas en blanco. Cuando se ejecuta una instrucción WRITE, se imprime un entero en la celda de la cinta de salida y, la cabeza de escritura, se mueve una celda a la derecha. Una vez que un símbolo se ha escrito, no puede modificarse.

La memoria consiste en una serie de registros r_0, r_1, …, r_i, …, con capacidad para almacenar un entero de tamaño arbitrario. No acotaremos superiormente el número de registros que podemos utilizar. Esta abstracción es válida para los casos donde:

1. El tamaño del problema es suficientemente pequeño y cabe en la memoria principal del ordenador.
2. Los enteros utilizados en la computación son tan pequeños que caben en la palabra del ordenador.



El programa para la RAM no se almacena en la memoria. Esto es, estamos suponiendo que el programa no se modifica a si mismo. El programa consiste básicamente en una sucesión de instrucciones etiquetadas (opcionalmente). La

naturaleza exacta de las instrucciones no es demasiado importante, siempre que se parezca a aquellas que encontramos en ordenadores reales. Suponemos que hay instrucciones aritméticas, instrucciones de E/ES, direccionamiento indirecto (para indizar conjuntos, por ejemplo) e instrucciones de bifurcación. Todos los cálculos tienen lugar en el primer registro r_0 llamado acumulador que, como cualquier otro registro, puede almacenar un entero arbitrario. Cada instrucción consta de un código de operación y un operando (dirección o etiqueta). Un conjunto de instrucciones para la RAM puede ser el siguiente:

Código de operación	Operando
1. LOAD	operando
2. STORE	operando
3. ADD	operando
4. SUB	operando
5. MULT	operando
6. DIV	operando
7. READ	operando
8. WRITE	operando
9. JUMP	etiqueta
10. JGTZ	etiqueta
11. JZERO	etiqueta
12: HALT	

En principio, podría aumentarse este conjunto de instrucciones con cualquier otro, tales como instrucciones lógicas, operaciones con caracteres,…, sin alterar el orden de complejidad de los problemas.

Un operando puede ser de las formas:

1. =i (modo inmediato), indica el entero i.
2. i (modo directo), un entero no negativo i, que indica el contenido del registro i.
3. *i (modo indirecto), indica que el operando es el contenido del registro j, donde j es el entero contenido en el registro i. Si j < 0, la máquina se para.

Estas instrucciones son bastante familiares a cualquiera que haya programado en un lenguaje ensamblador. Podemos definir el significado de un programa P con la ayuda de dos cantidades, una aplicación $c:Z^{+}\dashrightarrow Z$ y un contador de programa que indica la siguiente instrucción a ejecutar. La función c es una función de memoria, de manera que c(i) nos proporciona el contenido del registro i.

Inicialmente c(i) = 0 para todo i >= 0, y el contador de programa apunta a la primera instrucción del programa P, una vez ejecutada ésta, el contador de programa apunta automáticamente a la siguiente instrucción, a menos que la instrucción ejecutada sea JUMP, JGTZ, JZERO o HALT.

Para especificar el significado de una instrucción, se define el valor del operando v(a) como sigue:

v(=i) = i
v(i) = c(i)
v(*i) = c(c(i))

Por ejemplo, si el contenido de la memoria fuera:

r0	
r1	7
r2	5
r3	2
r4	2

v(2) toma el valor 5
v(3) toma el valor 2
v(*3) toma el valor 5

De esta forma el significado de cada instrucción queda definido como sigue:

Instrucción	Significado
1. LOAD a	c(0) <-- v(a)
2. STORE i	c(i) <-- c(0)
STORE *i	c(c(i)) <-- c(0)
3. ADD a	c(0) <-- c(0) + v(a)
4. SUB a	c(0) <-- c(0) - v(a)
5. MULT a	c(0) <-- c(0) * v(a)
6. DIV a	c(0) <-- c(0) div v(a)
7. READ i	c(i) <-- símbolo leido
READ *i	c(c(i)) <-- símbolo leído
	En ambos casos se trata del símbolo sobre el que se encuentra la cabeza de lectura. La cabeza de lectura se mueve una posición a la derecha después de la operación.
8. WRITE a	v(a) se imprime sobre la la cinta de salida. La cabeza de escritura se mueve una posición a la derecha después de la operación.
9. JUMP b	El contador de programa apunta a la instrucción etiquetada con b.
10. JGTZ b	El contador de programa apunta a la instrucción etiquetada con b si c(0) > 0. En otro caso el contador de programa apunta a la siguiente instrucción.
11. JZERO b	El contador de programa apunta a la instrucción etiquetada con b si c(0) = 0.En otro caso, el contador de programa apunta a la siguiente instrucción.
12. HALT	Fin de ejecución.

Las instrucciones no definidas como STORE =i, se consideran equivalentes a HALT. Lo mismo ocurre cuando se produce una división por cero y otra operación no definida.

En las ocho primeras operaciones el contador de programa se incrementa en una unidad. Esto es, las instrucciones en el programa se ejecutan en orden secuencial hasta que aparezca una instrucción JUMP, JGTZ con el acumulador mayor que cero, JZERO con el acumulador igual a cero o HALT.

Ejemplos:

LOAD =3; c(0) = 3

r0	3
r1	7

r0	3
r1	

LOAD 3; c(0) = c(3)

r0	2
r1	7
r2	5
r3	2

LOAD *3; c(0) = c(c(3))

r0	5
r1	7
r2	5
r3	2

- Se deja como ejercicio al lector, analizar como quedan los registros de la máquina cuando se ejecuta la siguiente secuencia de instrucciones sobre la configuración de memoria que presenta la figura anterior:

```
LOAD 1
ADD =1
STORE 1
LOAD *3
ADD 1
STORE *3
```

- Si utilizamos el registro r1 para almacenar el contenido de la variable x y, r2 para almacenar el contenido de la variable y, traducir las siguientes expresiones a

instrucciones ejecutables por la máquina RAM:

a) x = 5
b) y = 4
c) x = y
d) x = x + 1
e) y = x * y
f) x = (y + 3) * y * 4
g) x = (y + 3) * (y + 4)

a) x = 5 es equivalente a almacenar el valor 5 en r1, por lo que

```
LOAD =5
STORE 1
```

producen la asignación deseada.

- Si utilizamos el registro r1 para almacenar la variable x y, r2 para almacenar el contenido de la variable y, traducir el siguiente segmento de código C++ a instrucciones ejecutables por la máquina RAM:

```
if (x <= 0)
  cout << (int) 0;
else
  y = x;
```

Código para la máquina RAM:

```
      LOAD 1            ; if (x <= 0) cout << (int) 0;
      JGTZ else
      WRITE =0
      JUMP endif
else: LOAD 1            ; y = x
      STORE 2
endif:HALT
```

- Si utilizamos el registro r1 para almacenar el contenido de la variable x y, r2 para almacenar el contenido de la variable y, traducir las siguientes expresiones a instrucciones ejecutables por la máquina RAM:

```
while (x > 0) {
  y = y * x;
  x = x – 1;
}
cout << y;
```

Código para la máquina RAM:

```
   while: LOAD 1           ; while x > 0 do
          JGTZ continue
continue: LOAD 2           ; y = y * x
          MULT 1
          STORE 2
          LOAD 1           ; x = x -1
          SUB =1
          STORE 1
          JUMP while
endwhile: WRITE 2          ; cout << y
          HALT
```