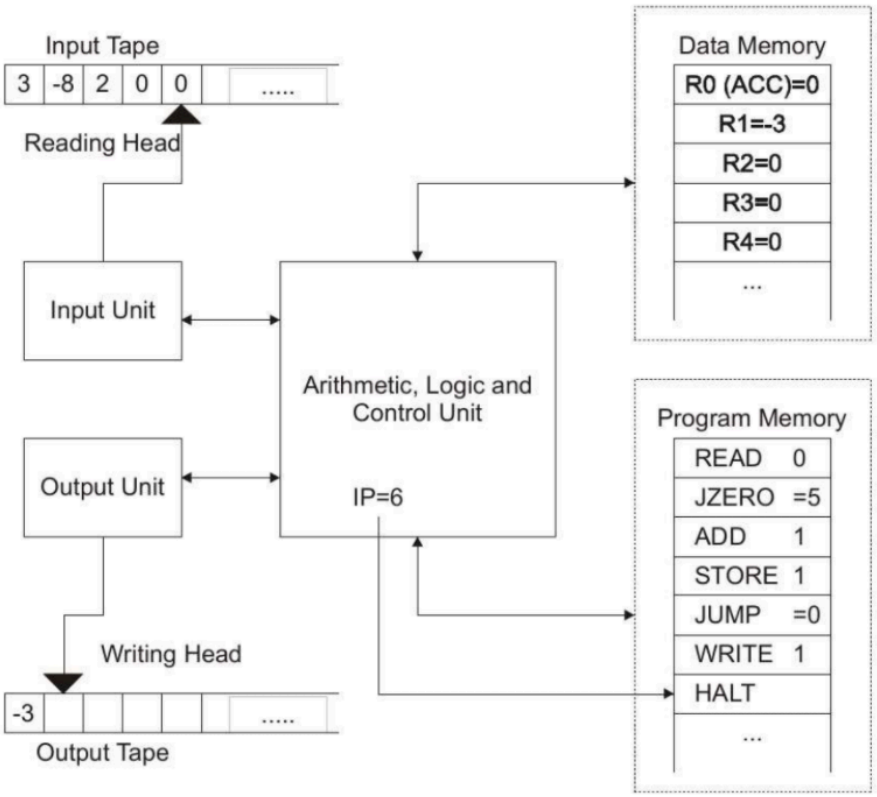
Algoritmo de Inserción y BubbleSort

Práctica 2/3 - Simulador de la máquina RAM



Realizado por …… Stephan Brommer Gutiérrez
Contacto……..… alu0101493497@ull.edu.es

Universidad de La Laguna

ÍNDICE

1.- Comparando el número de ejecuciones para instancias de diferentes tamaños

Cinta de entrada	Inserción (Número de ejecuciones)	BubbleSort (Número de ejecuciones)
0	4	17
1 0	19	31
2 1 0	59	87
3 2 1 0	113	167
4 3 2 1 0	181	271
5 4 3 2 1 0	262	399
6 5 4 3 2 1 0	359	551
7 6 5 4 3 2 1 0	469	727
8 7 6 5 4 3 2 1 0	593	927
9 8 7 6 5 4 3 2 1 0 (Totalmente desordenado)	731	1151
1 2 3 4 5 6 7 8 9 0 (Totalmente ordenado)	258	971

NOTA: Cabe destacar que primero, realizo una inicialización previa en la que cargo la secuencia de entrada en el registro R3. Este proceso de carga consume 7 ciclos de instrucción por cada elemento en la secuencia, más 3 ciclos adicionales para la detección del valor 0, que se utiliza como indicador del final de la secuencia. Esta etapa de inicialización es esencial para determinar la longitud del vector de entrada, que es un parámetro clave para el funcionamiento del algoritmo.

En lugar de seguir el enfoque tradicional de BubbleSort, que implica mover los elementos más pequeños (o “burbujas”) hacia el principio de la secuencia, he optado por una estrategia alternativa. En esta variante, se mueven los elementos más grandes (o “piedras”) hacia el final de la secuencia. Este enfoque puede describirse como una variante descendente del algoritmo BubbleSort.

Aquí muestro el pseudocódigo de lo que he implementado en mi máquina RAM:

```
1  for i = 1 to n-1 do
```

```
2     for j = 2 upto n - i do          <= Esta línea sería la que difiere del algoritmo de
3        if A[j-1] > A[j] then begin      bubbleSort
4           temp = A[j-1]
5           A[j-1] = A[j]
6           A[j] = temp
7        end
8     end
9  end
```

2.- Conclusiones

En el análisis comparativo de los algoritmos de ordenamiento BubbleSort e Inserción, se observa que, en general, BubbleSort tiende a incrementar la cantidad de instrucciones a un nivel más elevado que el algoritmo de Inserción para una secuencia de entrada desordenada. Esto se debe a la naturaleza del algoritmo BubbleSort, que realiza comparaciones y posibles intercambios para cada par de elementos adyacentes en la secuencia, independientemente de su estado de orden.

Por otro lado, en el caso de una secuencia de entrada ya ordenada, el algoritmo de Inserción es significativamente más eficiente, reduciendo el número de instrucciones ejecutadas a menos de ⅓ en comparación con el algoritmo de BubbleSort. Esto se debe a la característica del algoritmo de Inserción que permite finalizar la ejecución de su bucle interno cuando encuentra un elemento que ya está en su lugar correcto. En código, esto se refleja en la condición del bucle while i > 0 and A[i] > key. Si A[i] no es mayor que key, el bucle se termina y no se recorre completamente.

En contraste, BubbleSort continúa recorriendo la secuencia hasta que llega a los elementos que ya han sido “hundidos” al final de la secuencia, incluso si la secuencia ya está ordenada. Esto se refleja en la instrucción for j = 2 upto n - 1, que no tiene una condición de salida temprana similar a la del algoritmo de Inserción.