Universidad de La Laguna

Grado en Ingeniería Informática
Diseño y Análisis de Algoritmos
Curso 2017-2018

PRÁCTICA 1

Implementación de un simulador de máquina RAM

Fecha de defensa: 20 de febrero de 2018

1. Introducción

La *Máquina de Acceso Aleatorio* (del inglés, *Random Access Machine* - RAM) consiste en cuatro componentes bien diferenciados, tal y como se puede observar en la Figura 1:

1. **Unidad de memoria**.
2. **Unidad de aritmética, control y lógica**.
3. **Unidad de entrada**.
4. **Unidad de salida**.

La memoria de la máquina RAM se divide en la *memoria de datos* y la *memoria de programa*. La memoria de datos consiste en un número infinito de registros indexados desde el valor cero, es decir, R_0, R_1, R_2, etc. Cada registro puede almacenar un **número entero**. Además, el registro especial R_0, también conocido como *acumulador*, permite llevar a cabo operaciones aritméticas, lógicas y de control.

La memoria de programa también consiste de un número infinito de registros P_0, P_1, P_2, etc., los cuales pueden almacenar instrucciones (y sus correspondientes argumentos: operandos o etiquetas) pertenecientes al repertorio de instrucciones de la máquina RAM (véase Tabla 1). Cada instrucción puede tener asociada una *etiqueta*. Las instrucciones de control reciben una etiqueta como argumento, en lugar de un operando. Existen tres tipos diferentes de operandos.

- Operando $=i$: constante cuyo valor es i. Por ejemplo, la instrucción LOAD =10 carga en el acumulador el valor entero 10. Este tipo de operando no tiene sentido para las instrucciones STORE y READ.
- Operando i: esquema de *direccionamiento directo*. Por ejemplo, la instrucción LOAD 10 carga en el acumulador el valor almacenado en el registro R_{10}.

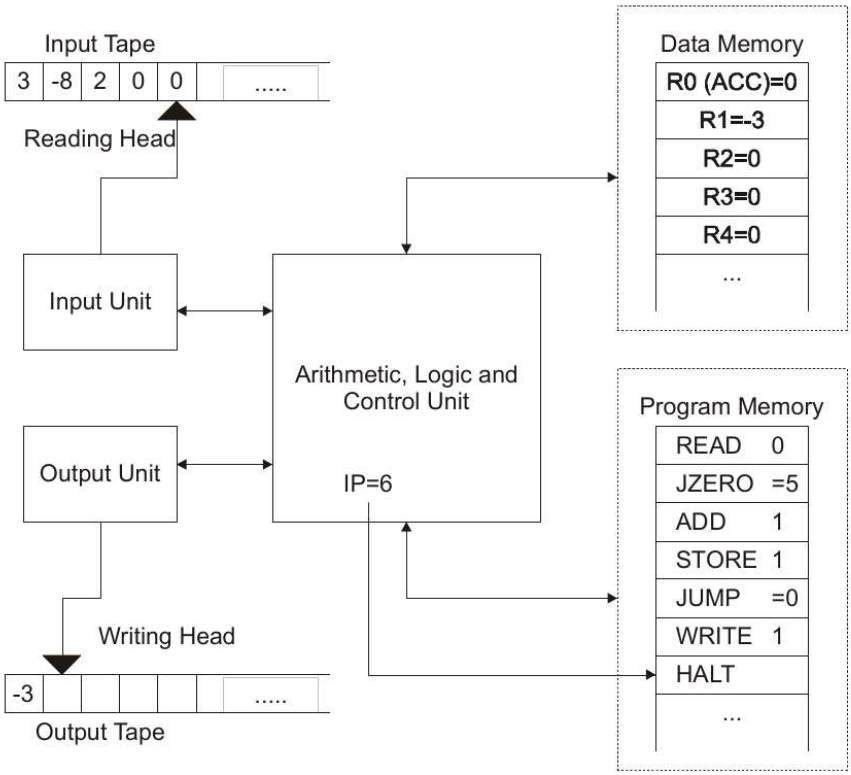

Figura 1: Componentes de una máquina RAM

- Operando $*i$: esquema de *direccionamiento indirecto*. Por ejemplo, la instrucción LOAD $*10$ carga en el acumulador el valor almacenado en el registro R_j, siendo j el valor contenido en el registro R_{10}.

La unidad de control cuenta con un *registro apuntador de instrucción* (IP), que apunta a la instrucción en la memoria de programa que se ejecutará en el siguente paso. Dicho registro IP puede modificarse mediante las instrucciones de control.

Un programa RAM consiste en una secuencia de instrucciones almacenadas en la memoria de programa. Computar un programa RAM sobre una entrada x (abreviadamente, un *cómputo RAM*) es una secuencia finita o infinita de instrucciones que se ejecutan de acuerdo a las siguientes reglas:

- La ejecución comienza con el puntero IP apuntando al registro de programa cero, es decir, $IP = P_0$, con todos los registros de la memoria de datos vacíos, las diferentes instrucciones cargadas en la memoria de programa, y con la entrada x almacenada en la *cinta de entrada*.

- La siguiente instrucción a ejecutar (que debe existir en el repertorio), y que se encuentra apuntada por IP, se lee junto con sus argumentos de la memoria de programa y se ejecuta.

- Una vez ejecutada dicha instrucción, el contenido de los registros de memoria afectados se habrán modificado y el puntero IP pasará a apuntar, o bien al siguiente

Cuadro 1: Instrucciones válidas de una máquina RAM

Instrucción	**Comportamiento**
LOAD op	El operando se carga en R_0
STORE op	El contenido de R_0 se almacena en la memoria según el operando
ADD op	El operando se suma a R_0 y el resultado se almacena en R_0
SUB op	El operando se resta a R_0 y el resultado se almacena en R_0
MUL op	El operando multiplica a R_0 y el resultado se almacena en R_0
DIV op	El operando divide a R_0 y el resultado se almacena en R_0
READ op	Se lee un valor de la cinta de entrada y se almacena en la memoria según el operando
WRITE op	Se escribe el operando en la cinta de salida
JUMP etiq	El valor del registro IP se modifica para apuntar a la instrucción identificada por la etiqueta
JZERO etiq	El valor del registro IP se modifica para apuntar a la instrucción identificada por la etiqueta (si $R_0 == 0$)
JGTZ etiq	El valor del registro IP se modifica para apuntar a la instrucción identificada por la etiqueta (si $R_0 > 0$)
HALT	Detiene la ejecución del programa

registro en la memoria de programa, o bien, al valor resultante de haber ejecutado alguna instrucción de control (por ejemplo, una instrucción JUMP).

- La instrucción READ lee de la cinta de entrada, mientras que la instrucción WRITE permite escribir resultados en la *cinta de salida*. La lectura/escritura de un valor, conlleva el avance de posición en las cintas. No se puede utilizar como operando de dichas instrucciones el acumulador R_0, es decir, las instrucciones READ 0 y WRITE 0 no son válidas.

- El programa termina si el registro IP apunta, o bien a una instrucción HALT, o bien, a ninguna instrucción.

La potencia de cómputo de una máquina RAM es equivalente a la potencia de cómputo de una *máquina de Turing*. Un resultado directo de lo anterior es que cualquier algoritmo secuencial puede implementarse como un programa RAM.

La complejidad temporal de un cómputo RAM sobre cierta entrada x viene dada por la suma de los costes de todas y cada una de las instrucciones ejecutadas durante dicho cómputo. Como simplificación, asumamos que todos los costes son unitarios (aunque podrían considerarse costes logarítmicos). Por lo tanto, la complejidad temporal vendrá dada por el numero total de instrucciones ejecutadas durante el cómputo RAM.

2. Objetivos, requisitos y evaluación

2.1. Objetivos

Será **condición necesaria, pero no suficiente para aprobar** la práctica, llevar a cabo las siguientes tareas:

1. Programar un simulador de máquina RAM atendiendo a la especificaciones indicadas en la Sección 1 y a los requisitos de la Sección 2.2. Se debe utilizar el paradigma de desarrollo orientado a objetos, teniendo en cuenta los diferentes componentes de la máquina RAM, tal y como muestra la Figura 1. Además, el lenguaje de programación utilizado será C++ o Java.

2. Desarrollar un programa que pueda ejecutarse en el anterior simulador RAM. En concreto, deberá implementar un programa que calcule x^y, donde x e y son enteros que deben venir especificados en la cinta de entrada. El resultado del cálculo deberá almacenarse en la cinta de salida. Por último, si $y < 0$, entonces se escribirá en la cinta de salida algún número que por convención indique que la operación no puede llevarse a cabo.

2.2. Requisitos

Los requisitos que debe cumplir el simulador son los enumerados a continuación:

1. Todo el código deberá estar adecuadamente comentado y desarrollado atendiendo al paradigma de programación orientada a objetos [1 punto].

2. El simulador debe funcionar con los ejemplos de prueba proporcionados junto con este enunciado [1 punto].

3. El programa RAM a cargar en la memoria, así como el contenido de la cinta de entrada y de la cinta de salida, se modelarán mediante tres ficheros independientes [1 punto].

4. El simulador debe ejecutarse de la siguiente manera [1 punto]:

   C++: `./ram_sim ram_program.ram input_tape.in output_tape.out debug`
   Java: `java ram_sim ram_program.ram input_tape.in output_tape.out debug`

   - `ram_program.ram`: fichero con el programa RAM.
   - `input_tape.in`: fichero con el contenido de la cinta de entrada.
   - `output_tape.out`: fichero con el contenido de la cinta de salida.
   - `debug`: Si el valor de este parámetro es 1, se llevará a cabo una simulación paso a paso, mostrando por consola en cada uno de ellos el contenido del registro IP, de la memoria de datos, de la memoria de programa, de las cintas de entrada y salida, y del número total de instrucciones ejecutadas hasta ese momento. El valor 0 lleva a cabo la simulación completa. Al finalizar la misma, sólo deberá mostrarse por consola el número total de instrucciones ejecutadas.

5. Las instrucciones del programa RAM pueden estar escritas en mayúscula o en minúscula [1 punto].

6. Se pueden especificar comentarios en un programa RAM. Una línea de comentario comienza con el caracter '#'. Estas líneas se ignorarán a la hora de cargar el programa RAM en la memoria [1 punto].

7. Se deben contemplar los tres tipos de operandos: constante, direccionamiento directo y direccionamiento indirecto [1 punto].

8. Se debe comprobar que las instrucciones sean válidas. Por un lado, deberán ser aquellas especificadas en la Tabla 1, y por el otro, sus correspondientes argumentos deben ser correctos. Por ejemplo, la instrucción STORE =3 no es válida (STORE no permite operando de tipo constante) [1 punto].

9. Si se detecta algún error, se debe imprimir en la consola un mensaje que indique la instrucción y número de línea dentro del programa RAM que lo provocó [1 punto].

10. Cuando se alcance una instrucción HALT o se detecte algún error, el contenido de la cinta de salida se volcará al fichero correspondiente [1 punto].

2.3. Evaluación

La **calificación máxima** de la práctica será de **Sobresaliente (10.0)**. Por cada uno de los requisitos mencionados en la Sección 2.2 que se cumpla, se obtendrá el número de puntos indicado para cada uno de ellos. Además, cabe mencionar que durante la defensa de la práctica **se podrá solicitar algún tipo de modificación o prueba adicional**, la cual afectará en mayor o menor grado a la calificación final dependiendo de la dificultad.

2.4. Entregables

Los siguientes entregables deberán estar disponibles en el **repositorio asociado a la correspondiente tarea de Github Classroom**:

1. Código fuente del simulador.

2. Programa RAM implementando el cálculo x^y.

El enlace para poder clonar el repositorio se encuentra disponible en la correspondiente tarea de entrega de esta práctica en el aula virtual. Dicha tarea estará abierta hasta el **20 de febrero de 2018 a las 08:00 horas, y en la misma deberá indicar la** URL **del repositorio donde ha desarrollado la práctica**.