Computabilidad y Complejidad (CMC)

Práctica 2: Máquinas de registros

José M. Sempere

Valencian Research Institute for Artificial Intelligence (VRAIN)
Valencian Graduate School and Research Network of Artificial Intelligence (valgrAI)
Department of Informatics Systems and Computation (DSIC)
Universitat Politècnica de València (UPV)

http:// jsempere.webs.upv.es

jsempere@dsic.upv.es

Índice

1. Generalidades
2. El modelo RAM, RASP y la máquina contador
3. Operativa de las máquinas
4. Diseño de macros
5. Algunos ejemplos de programas RAM
6. Equivalencias entre los modelos
7. Implementación en *Mathematica*
8. Actividades propuestas

Bibliografía básica recomendada

- Computation: Finite and Infinite Machines (M.L. Minsky)
- The Language of Machines. An Introduction to Computability and Formal Languages (R.W. Floyd, R. Beigel)
- Computability. (K. Weihrauch)

Introducción

En este tema describiremos diferentes modelos de cálculo basados estructuralmente en los computadores digitales ordinarios.

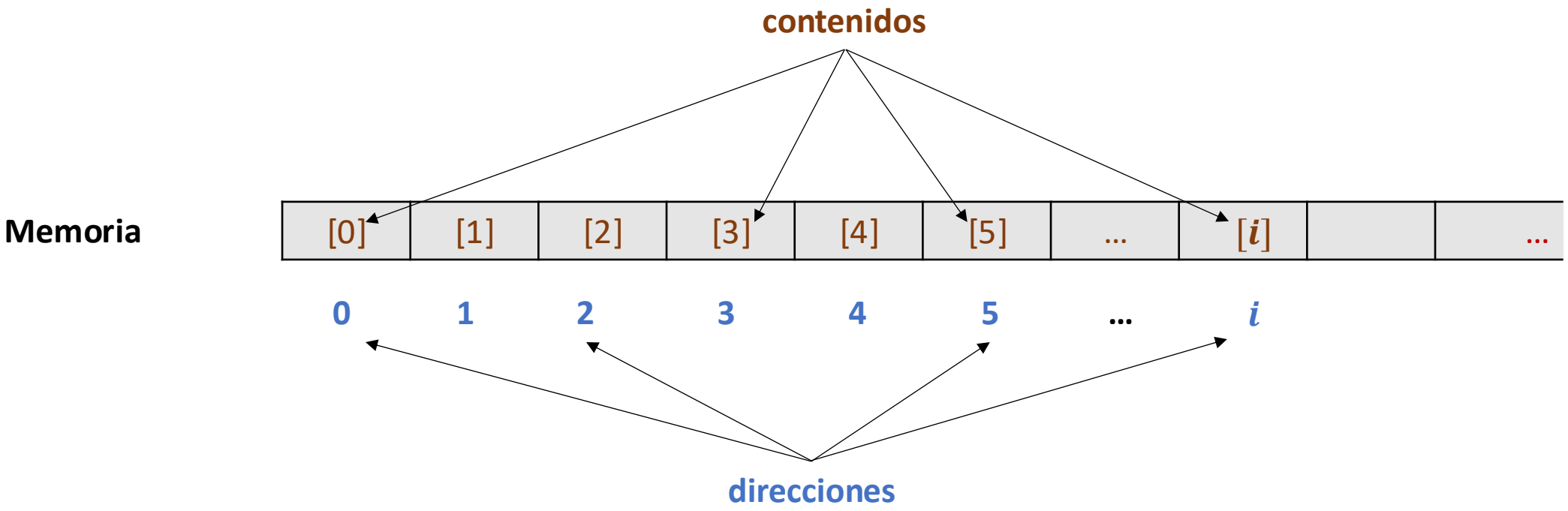
Estos modelos consisten en máquinas de registros que abstraen e idealizan los elementos constituyentes de los computadores digitales, de modo que, por su idealización, no tienen existencia real y permiten, al igual que la máquina de Turing, definir la familia de las funciones computables. De hecho todos los modelos que describiremos son computacionalmente equivalentes al modelo de Turing.

Los modelos descritos a continuación son:

- el modelo RAM,
- el modelo RASP,
- el modelo de la máquina contador.

Generalidades

Cualquier máquina de registros consta de una memoria potencialmente infinita organizada en registros. Cada registro tiene una dirección de memoria $i \in \mathbb{N}$ y puede almacenar un número natural arbitrariamente grande $[i] \in \mathbb{N}$

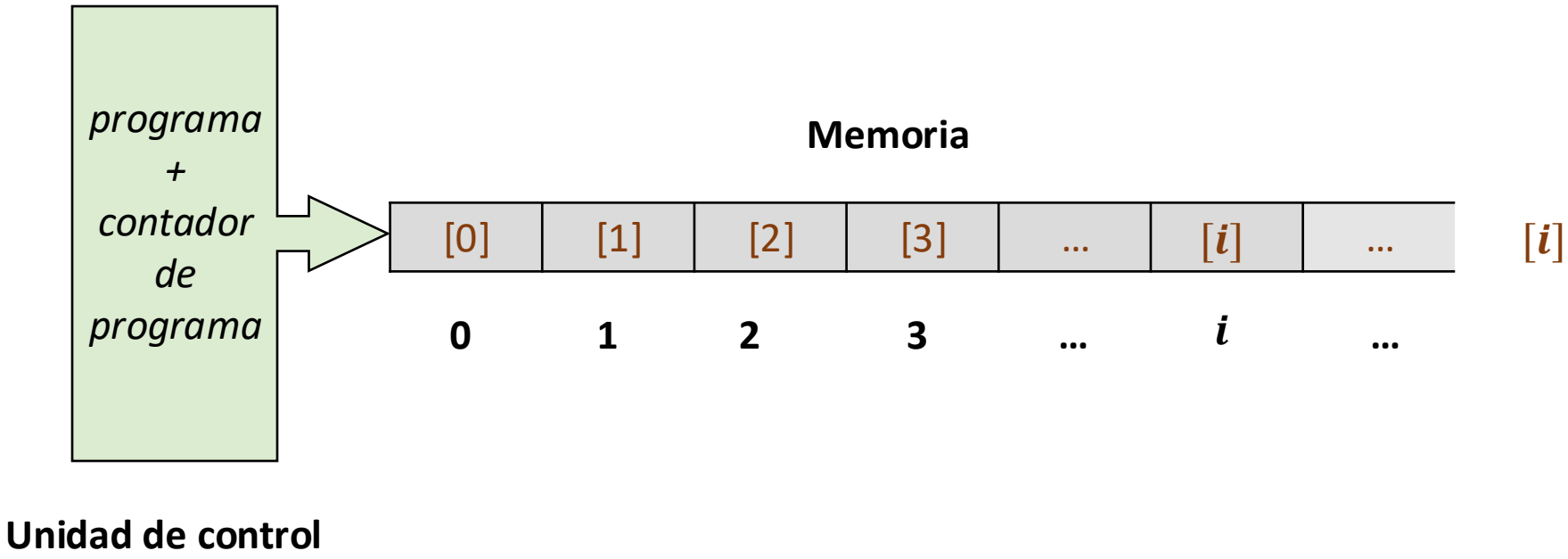


Generalidades

- Un programa de una máquina de registros es una secuencia finita de instrucciones enumeradas. La primera instrucción se enumera como la instrucción 0.
- Cada instrucción puede ser de un tipo predefinido de instrucciones básicas.
- Las instrucciones permiten el direccionamiento indirecto (salvo que se indique lo contrario).
- La unidad de control de la máquina consiste en el programa junto con un contador de programa que indica la instrucción en ejecución.
- Opcionalmente, cada instrucción puede llevar una etiqueta asociada.
- La máquina se detiene cuando intenta acceder a una instrucción inexistente. Adicionalmente, se puede definir una instrucción de parada.
- El resultado de la ejecución de un programa se almacena en uno (o varios) registros predefinidos.

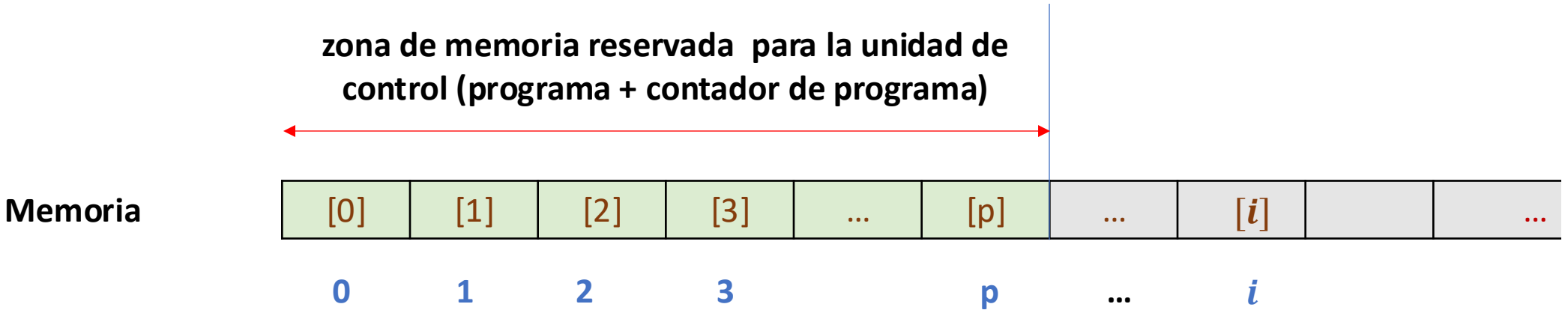
La maquina RAM (Random Access Machine)

(arquitectura de *Harvard*)



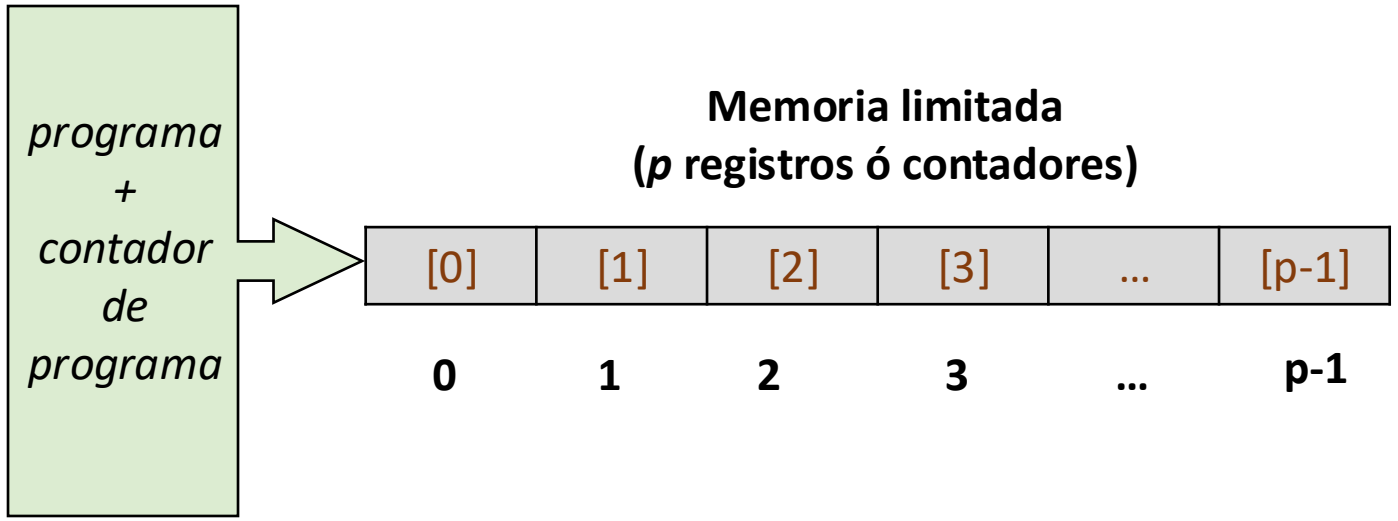
La maquina RASP (Random Access Stored Program Machine)

(arquitectura de *von Neumann*)



- La máquina RASP puede simular a cualquier máquina RAM y, en consecuencia, es un modelo **universal**

La maquina contador



Unidad de control

- La máquina no admite instrucciones con direccionamiento indirecto

Operativa de las máquinas (I)

- Cada registro será designado mediante la notación $R_i,\ i \in \mathbb{N}$; donde i es su dirección. Su contenido lo denotaremos mediante $[i]$.
- Las máquinas las utilizaremos para computar funciones numéricas de la forma $g: \mathbb{N}^m \longrightarrow \mathbb{N}^k$
- Para su cómputo la máquina arranca con los m primeros registros con los argumentos de la función y el resto de registros con valor nulo. Cuando la máquina termine el cómputo, su resultado será el contenido de los k primeros registros de la memoria (en las máquinas RASP, no se toman en cuenta la zona de memoria habilitada para la unidad de control); en otro caso, si el cómputo no termina el resultado no está definido.

Operativa de las máquinas (II)

La máquina computa la función g definida de modo que $g(n_1, \ldots, n_m) = (j_1, \ldots, j_k)$ siempre que la máquina arranca con:

$$[i] = n_{i+1}, \qquad i = 0, \ldots, m-1$$
$$[i] = 0, \qquad \forall\, i \geq m$$

y termina con

$$[i] = j_{i+1}, \qquad i = 0, \ldots, k-1$$

Operativa de las máquinas (III)

El conjunto básico de instrucciones es el siguiente

- **suc(i)** Incrementa en una unidad el contenido de R_i es decir $[i] \leftarrow [i] + 1$. La ejecución continúa en la siguiente instrucción del programa si existe, si no termina.

- **pre(i, k)**

  - Si $[i] > 0$, entonces $[i] \leftarrow [i] - 1$. La ejecución continúa en la siguiente instrucción del programa si existe, si no termina.

  - Si $[i] \leq 0$, entonces la ejecución continúa en la k-ésima instrucción si existe, si no termina

Este par de instrucciones junto con la existencia de un registro predefinido con contenido nulo son suficientes para realizar cualquier computación.

Ejemplos

Seguidamente veremos algunos fragmentos de programa que ejecutan cálculos sencillos que ilustran la operatividad de este modelo. En lo inmediato que sigue R_c tiene asignado permanentemente el valor nulo.

$[i] \leftarrow 0$

$$\begin{aligned} n&: \ \mathrm{pre}(i, n+2) \\ n+1&: \ \mathrm{pre}(c, n) \end{aligned}$$

$[i] \leftarrow [i] + [j] \ \wedge \ [j] \leftarrow 0 \ \wedge \ R_i \neq R_j$

$$\begin{aligned} n&: \ \mathrm{pre}(j, n+3) \\ n+1&: \ \mathrm{suc}(i) \\ n+2&: \ \mathrm{pre}(c, n) \end{aligned}$$

Operativa de las máquinas (IV)

Para evitar el tener que mantener un registro predefinido permanentemente con valor nulo añadiremos una nueva instrucción de direccionamiento incondicional

- **goto(n)** Ejecuta incondicionalmente la instrucción n-ésima si ésta existe, si no termina la ejecución.

Las instrucciones $\mathrm{suc}(i)$, $\mathrm{pre}(i,k)$ y $\mathrm{goto}(n)$ son suficientes para realizar cualquier computación.

Ejemplo

$$[i] \leftarrow [i] + [j] \ \wedge \ [j] \leftarrow 0 \ \wedge \ R_i \neq R_j$$

$$\begin{aligned} n&: \ \mathrm{pre}(j, n+3) \\ n+1&: \ \mathrm{suc}(i) \\ n+2&: \ \mathrm{goto}(n) \end{aligned}$$

Diseño de macros

Una macro es una secuencia de instrucciones que se puede invocar mediante un nombre y unos parámetros que hacen referencia a direcciones de registros.
La definición de macros facilita el diseño de programas tal y como sucede en los lenguajes de programación de alto nivel.

- Poner el contenido del registro R_i a cero $[i] \leftarrow 0$

$$\begin{array}{|r|}\hline n\colon\ \mathrm{pre}(i, n+2) \\ n+1\colon\ \mathrm{goto}(n) \\ \hline\end{array} \quad \mathrm{cer}(i)$$

Diseño de macros

- Asignar el valor k al registro R_i $[i] \leftarrow k$

$$\begin{array}{r} \text{cer}(i) \\ n+1\colon\ \text{suc}(i) \\ \ldots \\ n+k\colon\ \text{suc}(i) \end{array} \qquad \text{asi}(k,i)$$

- Asignar el contenido del registro R_j al R_i (siendo R_i y R_j dos registros distintos) $[i] \leftarrow [j] \ \wedge \ R_i \neq R_j$

Consideremos R_k un registro auxiliar distinto de los anteriores ($R_i \neq R_k \ \wedge \ R_j \neq R_k$)

$$\begin{array}{r} \text{cer}(i) \\ \text{cer}(k) \\ n\colon \text{pre}(j, n+4) \\ n+1\colon \text{suc}(i) \\ n+2\colon \text{suc}(k) \\ n+3\colon \text{goto}(n) \\ n+4\colon \text{pre}(k, n+7) \\ n+5\colon \text{suc}(j) \\ n+6\colon \text{goto}(n+4) \end{array} \qquad \text{cop}(j,i)$$

Diseño de macros

- Sumar el contenido de los registros R_i y R_j y almacenar el resultado en el registro R_k $[k] \leftarrow [i] + [j]$

  Consideremos R_p y R_q dos registros auxiliares distintos de los anteriores

$$
\begin{array}{r}
\mathrm{cop}(i,p) \\
\mathrm{cop}(j,q) \\
n\colon \mathrm{pre}(q, n+3) \\
n+1\colon \mathrm{suc}(p) \\
n+2\colon \mathrm{goto}(n) \\
n+3\colon \mathrm{cop}(p,k)
\end{array}
\qquad \mathrm{sum}\ (i,j,k)
$$

Diseño de macros

- Multiplicar el contenido de los registros R_i y R_j y almacenar el resultado en el registro R_k $[k] \leftarrow [i] \times [j]$

  Consideremos R_p, R_q, R_s y R_t cuatro registros auxiliares distintos de los anteriores (se podría prescindir de R_q)

$$
\begin{array}{r}
\text{cer}(t) \\
\text{cer}(s) \\
\text{cop}(i,p) \\
\text{cop}(j,q) \\
n\text{: pre}(p, n+8) \\
n+1\text{: pre}(q, n+5) \\
n+2\text{: suc}(t) \\
n+3\text{: suc}(s) \\
n+4\text{: goto}(n+1) \\
n+5\text{: pre}(s,n) \\
n+6\text{: suc}(q) \\
n+7\text{: goto}(n+5) \\
n+8\text{: cop}(t,k)
\end{array}
$$

ó alternativamente

$$
\begin{array}{r}
\text{cer}(t) \\
\text{cop}(i,p) \\
n\text{: pre}(p, l+1) \\
\text{sum}(t,j,t) \\
l\text{: goto}(n) \\
l+1\text{: cop}(t,k)
\end{array}
$$

$\text{mul}(i,j,k)$

Diseño de macros

- Realizar la división entera del contenido de los registros R_i y R_j y almacenar el resultado en el registro R_k $[k] \leftarrow [i]/[j]$

Consideremos R_p, R_q, R_s y R_t cuatro registros auxiliares distintos de los anteriores

$$\begin{array}{r} \text{cer}(t) \\ \text{cer}(s) \\ \text{cop}(i,p) \\ \text{cop}(j,q) \\ n\text{: pre}(q,\text{error}) \\ n+1\text{: suc}(q) \\ n+2\text{: pre}(q,n+6) \\ n+3\text{: pre}(p,n+10) \\ n+4\text{: suc}(s) \\ n+5\text{: goto}(n+2) \\ n+6\text{: suc}(t) \\ n+7\text{: pre}(s,n+2) \\ n+8\text{: suc}(q) \\ n+9\text{: goto}(n+7) \\ n+10\text{: cop}(t,k) \\ \dots \\ \text{error:} \end{array}$$

$\text{div}(i,j,k)$

Diseño de macros

- Realizar la comparación menor o igual del contenido de los registros R_i y R_j e ir a direcciones de programa condicionadas si $([i] \leq [j])$ entonces goto m_1
  sino goto m_2

Consideremos R_p y R_q dos registros auxiliares distintos de los anteriores

$$\begin{aligned} &\text{cop}(i,p)\\ &\text{cop}(j,q)\\ n:\ &\text{pre}(p,m_1)\\ n+1:\ &\text{pre}(q,m_2)\\ n+2:\ &\text{goto}(n) \end{aligned}$$

$$\text{mei}(i,j,m_1,m_2)$$

Diseño de macros

- Realizar la comparación igual del contenido de los registros R_i y R_j e ir a direcciones de programa condicionadas si $([i] = [j])$ entonces goto m_1
  sino goto m_2

Consideremos R_p y R_q dos registros auxiliares distintos de los anteriores

$$\begin{aligned} &\text{cop}(i,p)\\ &\text{cop}(j,q)\\ n:&\ \text{pre}(p,n+3)\\ n+1:&\ \text{pre}(q,m_2)\\ n+2:&\ \text{goto}(n)\\ n+3:&\ \text{pre}(q,m_1)\\ n+4:&\ \text{goto}(m_2) \end{aligned}$$

$\text{igu}(i,j,m_1,m_2)$

Instrucciones y macros

instrucción	semántica
$\mathrm{suc}(i)$	$[i] \leftarrow [i] + 1$
$\mathrm{pre}(i,k)$	si $[i] > 0$ entonces $[i] \leftarrow [i] - 1$ sino goto$[k]$
$\mathrm{goto}(k)$	ir a la instrucción k

macro	semántica
$\mathrm{cer}(i)$	$[i] \leftarrow 0$
$\mathrm{asi}(c,i)$	$[i] \leftarrow c$
$\mathrm{cop}(j,i)$	$[i] \leftarrow [j]$
$\mathrm{sum}(i,j,k)$	$[k] \leftarrow [i] + [j]$
$\mathrm{mul}(i,j,k)$	$[k] \leftarrow [i] \times [j]$
$\mathrm{div}(i,j,k)$	$[k] \leftarrow [i]/[j]$
$\mathrm{mei}(i,j,m_1,m_2)$	si $[i] \leq [j]$ entonces goto(m_1) sino goto(m_2)
$\mathrm{ig}(i,j,m_1,m_2)$	si $[i] = [j]$ entonces goto(m_1) sino goto(m_2)

Algunos ejemplos de programas RAM

Proporcione el código de programa para una máquina RAM que calcule $[i] \leftarrow \text{factorial}([j])$ siendo R_i y R_j dos registros no necesariamente distintos. Considere $\text{factorial}(0) = 1$.

Consideraremos dos registros R_q y R_p distintos a R_i y R_j

```
        cop(j, q)
        cer(p)
        ig(q, p, cero, nocero)
cero:   suc(p)
        goto(fin)
nocero: asi(1, p)
bucle:  mul(q, p, p)
        pre(q, fin)
        pre(q, fin)
        suc(q)
        goto(bucle)
fin:    cop(p, i)
```

Algunos ejemplos de programas RAM

Proporcione el código de programa para una máquina RAM que calcule $[i] \leftarrow [\log_2[j]]$ siendo R_i y R_j dos registros no necesariamente distintos

Consideraremos dos registros R_q R_p y R_m distintos a R_i y R_j

```
        cer(q)
        ig(j, q, error, sig)
sig:    asi(1, q)
        ig(j, q, uno, sig2)
uno:    cer(p)
        goto(fin)
sig2:   asi(2, q)
        asi(2, m)
        asi(1, p)
        mei(j, q, fin, bucle)
bucle:  mul(q, m, q)
        suc(p)
        ig(j, q, fin, menor)
menor:  mei(j, q, resta1, bucle)
resta1: pre(p, error)
fin:    cop(p, i)
error:  ...
```

Algunos ejemplos de programas RAM

Proporcione el código de programa para una máquina RAM que calcule $[i] \leftarrow numdivisores([j])$ siendo R_i y R_j dos registros no necesariamente distintos. ($numdivisores([j])$ indica el número de divisores del contenido del registroR_j)

Consideraremos dos registros R_q R_p y R_m distintos a R_i y R_j

```
          asi(1, q)
          cer(m)
test:     ig(j, q, fin, division)
division: div(j, q, p)
          mul(p, q, p)
          ig(j, p, divisor, nodivisor)
divisor:  suc(q)
          suc(m)
          goto(test)
nodivisor: suc(q)
          goto(test)
fin:      suc(m)
          cop(m, i)
```

Algunos ejemplos de programas RAM

Sea la función f definida como sigue

$$f(n,m) = \begin{cases} n^3 + m^2 & \text{si } n \leq m \\ m^n & \text{en cualquier otro caso} \end{cases}$$

Proporcione el código de programa para una máquina RAM que calcule $[i] \leftarrow f([j],[k])$ siendo R_i, R_j y R_k tres registros no necesariamente distintos.

Consideraremos cinco registros R_n R_m R_s R_t y R_l distintos a R_i R_j y R_k

```
        cop(j, n)
        cop(k, m)
        mei(j, k, menorig, mayor)
menorig: cop(n, s)
        mul(n, s, n)
        mul(n, s, n)
        cop(m, t)
        mul(m, t, m)
        sum(m, n, l)
        goto(fin)
mayor:  asi(1, l)
        cop(m, t)
bucle:  pred(n, fin)
        mul(l, t, l)
        goto(bucle)
fin:    cop(l, i)
```

Equivalencias entre los modelos

Sea una función (parcial) de la forma $g: \mathbb{N}^m \to \mathbb{N}^k$. Diremos que la función g es:

- RAM computable si y sólo si existe una máquina RAM que la compute.
- RASP computable si y sólo si existe un programa para la máquina RASP con el que esta máquina puede computarla.
- MC computable si y sólo si existe una máquina contador que la compute.

Teorema: Sea una función (parcial) de la forma $g: \mathbb{N}^m \to \mathbb{N}^k$. Los siguiente enunciados son equivalentes:

- g es RAM computable.
- g es RASP computable.
- g es MC computable.
- g es Turing computable.

Implementación en Mathematica

La **memoria** de una máquina RAM se implementará en Mathematica mediante una lista. Obsérvese que en Mathematica, la posición 0 de las listas es una posición reservada a la que no se debe acceder. Por lo tanto, el primer registro de la memoria lo ocupará la posición 1 de la lista:

Memoria

i_0	i_1	i_2	…	i_p
0	**1**	**2**	**…**	**p**

En Mathematica

M=$\{i_0, i_1, i_2, \ldots, i_p\}$

$\forall\, j \geq 1$ M[[j]]= i_{j-1}

Implementación en Mathematica

El **flujo de programa** se puede implementar mediante las instrucciones *Goto* y *Label* de Mathematica

Goto [*tag*]
scans for Label [*tag*], and transfers control to that point.

Label [*tag*]
represents a point in a compound expression to which control can be transferred using Goto .

```
f[a_] := Module[{x = 1., xp = 0},
          módulo
  Label[begin];
  etiqueta
  If[Abs[xp - x] < 10^-8, Goto[end]];
  si  valor absoluto          ve a
  xp = x;
  x = (x + a / x) / 2;
  Goto[begin];
  ve a
  Label[end];
  etiqueta
  Return[x]]
  retorna
```

Implementación en Mathematica

Pueden emplearse las macros implementadas en el notebook Macros.nb

Ejemplo: Calcular el factorial de un valor entero positivo almacenado en el registro j de una máquina RAM. Considere j distinto de los registros 1, 3 y 4.

```
factorial[j_Integer] := Module[{},
                          cop[j, 3];
                          cer[4];
                          ig[3, 4, cero, nocero];
  Label[cero];            suc[4];
                          Goto[fin];
  Label[nocero];          asi[1, 4];
  Label[bucle];           mul[3, 4, 4];
                          pre[3, fin]];
                          pre[3, fin]];
                          suc[3];
                          Goto[bucle];
  Label[fin];             cop[4, 1]
  ];
```

memoria = {0, 6, 0, 0, 0, 0, 0, 0, 0, 0}

In[]:= factorial[2]

Out[]= 720

In[]:= memoria

Out[]= {720, 6, 0, 720, 0, 0, 0, 0, 0, 0}

Actividades propuestas

Desarrolle módulos en Mathematica que simulen a máquinas RAM que calculen las siguientes funciones (puede asumir que los registros j y k son registros distintos entre sí y distintos del registro 1)

1. $f(n,m) = n^{\lfloor\sqrt{m}\rfloor}$ $\quad [1] \leftarrow f([j],[k])$
2. $f(n,m) = \lfloor \log_m n \rfloor$ $\quad [1] \leftarrow f([j],[k])$
3. $f(n) = \lfloor\sqrt{n}\rfloor$ $\quad [1] \leftarrow f([j])$
4. $f(n,m) = \begin{cases} n^m & \text{si } n \leq m \\ m^n & \text{en otro caso} \end{cases}$ $\quad [1] \leftarrow f([j],[k])$

- Al final de cada módulo se debe mostrar el contenido del registro 1 de la memoria que contendrá el resultado de la función.
- Sólo se pueden utilizar las estructuras de programación formuladas mediante las Macros que se proporcionan en el notebook correspondiente.