DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores

Almacenamiento de datos

©Alejandro Echeverría, Hans Löbel

1. Motivación

Una de las funciones básicas de un computador, además de realizar operaciones, es poder almacenar información. Existen distintas tecnologías para almacenar información que el computador puede ocupar, las cuales presentan distintas características que las hacen útiles para distintas funciones en el computador.

2. Variables y Arreglos

2.1. Variables

La unidad básica de almacenamiento de un programa se conoce como **variable**. Una variable corresponde a un contenedor que puede almacenar un valor que puede variar en el transcurso del programa (por eso su nombre). Las variables tienen tres características que las determinan: un **tipo de dato** que indica que información está almacenando; un **valor** que representa la información almacenada en un momento dado; y un **identificador** que se ocupa para reconocer a una determinada variable y diferenciarla de otras. Para poder representar variables de manera física en una máquina, por lo tanto, es necesario proveer estas tres características.

Como se ha visto anteriormente, el computador almacenará toda su información como números binarios (i.e. secuencias de bits). De esta forma, todas las variables se almacenarán finalmente como una secuencia de bits. La interpretación que se le de a esa secuencia de bits, sin embargo, variará, dependiendo del tipo de dato. De esta forma el valor de una variable es función tanto de la secuencia de bits como del tipo de dato. **Sin conocer el tipo de dato es imposible interpretar el valor de una determinada secuencia de bits.**

Los tipos de datos van a estar determinados por tres características principales: la **cantidad de bits** ocupados, y la **codificación** de la secuencia de bits y la **interpretación** que se le de a una codificación particular. La cantidad de bits determinará el rango de valores disponibles de un determinado tipo, es decir los valores máximo y mínimo que se pueden almacenar en un determinado tipo. La codificación determinara que está almacenando (la sintáxis del tipo) y la interpretación indicará que representa esa codificación particular para este tipo (la semántica del tipo).

Para entender mejor los diferentes elementos involucrados en un tipo de dato, analizaremos los tipos de datos básicos del lenguaje Java:

- Tipo `int`
  - Codificación: Número en base 2 con signo, ocupando representación de complemento a 2.
  - Interpretación: Número entero positivo o negativo.

- Cantidad de bits: 32 bits (4 bytes), lo que permite un rango de valores entre los números -2147483648 y 2147483647.

- Tipo `char`
  - Codificación: Número en base 2 sin signo.
  - Interpretación: Caracter o letra obtenida del estándar Unicode.
  - Cantidad de bits: 16 bits (2 bytes), lo que permite un rango de valores entre los caracteres asociados a los números 0 y 65535.

- Tipo `byte`
  - Codificación: Número en base 2 con signo, ocupando representación de complemento a 2.
  - Interpretación: Número entero positivo o negativo.
  - Cantidad de bits: 8 bits (1 byte), lo que permite un rango de valores entre los números -128 y 127.

- Tipo `boolean`
  - Codificación: Número en base 2 sin signo.
  - Interpretación: Valor lógico o de verdad, pudiendo solo ser verdadero (1) o falso (0).
  - Cantidad de bits: 8 bits (1 byte), lo que permite potencialmente un rango de valores entre los números 0 y 255, aunque solo se permite ocupar los valores 0 y 1, asociados a los valores `true` y `false`.

- Tipo `float`
  - Codificación: Single precission floating point, según especificado por el estándar IEEE754.
  - Interpretación: Número fraccional positivo o negativo, además de algunos símbolos especiales (+Infinito, -Infinito, NaN).
  - Cantidad de bits: 32 bits (4 bytes), lo que permite un rango aproximado de valores entre los números $\pm 1{,}5 \times 10^{-45}$ y $\pm 3{,}4 \times 10^{38}$.

- Tipo `double`
  - Codificación: Double precission floating point, según especificado por el estándar IEEE754.
  - Interpretación: Número fraccional positivo o negativo, además de algunos símbolos especiales (+Infinito, -Infinito, NaN).
  - Cantidad de bits: 64 bits (8 bytes), lo que permite un rango aproximado de valores entre los números $\pm 5 \times 10^{-324}$ y $\pm 1{,}7 \times 10^{308}$.

2.2. Arreglos

Una segunda forma de almacenamiento que es relevante en un programa son los **arreglos**. Un arreglo se define como un conjunto de datos agrupados por un identificador único, y que permite acceder de manera indexada a una cierto dato para leer o modificar su valor. Los arreglos tienen cuatro características que las determinan: un **tipo de dato** que indica que información está almacenando en todos los datos; un **conjunto de valores** distintos que representa la información almacenada en un momento dado; un

identificador que se ocupa para reconocer el arreglo y poder indexarlo para acceder a un valor; y un **largo** que indica cuantos valores se tienen almacenados.

Para poder representar arreglos de manera física en una máquina, por lo tanto, es necesario proveer estas tres características, lo que implica que además de los detalles descritos en la sección previa, para almacenar arreglos se requiere la capacidad de almacenar múltiples datos de manera ordenada.

3. Tecnologías de almacenamiento de números binarios

Una de las principales ventajas de usar números binarios es que su almacenamiento se puede realizar usando dos símbolos o estados por cada dígito o **bit**. Debido a esto existen diversas tecnologías que permiten almacenar números binarios, las cuales presentan distintas características.

3.1. Características

Los distintos tipos de almacenamiento se diferencia por una serie de características. A continuación se listan las más relevantes.

Tecnología

La primera característica diferenciadora es la tecnología que se ocupa para almacenar la información. Aunque en la actualidad existen una multitud de formas de guardar número binarios, son tres las tecnologías predominantes:

- Magnética: usada en los discos duros, se basa en modificar el estado magnético de una parte de una superficie para representar un 1. Las zonas no modificadas representa un 0. Para poder escribir y leer en esta superficie se utilizan cabezales especiales que son capaces de moverse en la posición donde se almacena la información y modificar el estado magnético para escribir o interpretar el estado para leer.
- Óptica: usada en los CD, DVD y discos Blu-ray, se basa en modificar la capacidad de reflectancia óptica de una sector superficie, de manera de que cuando se ilumine ocurra o no reflexión lo que se interpreta como un 1 o 0 lógico. Para leer se utilizan lasers de precisión que apuntan a una zona específica y contienen sensores que interpretan la reflectancia de la zona. Para escribir, se utilizan lasers más poderosos que son capaces de modificar las propiedades de reflectancia.
- Eléctrica: usada en las memorias flash, y otros tipos de memoria, se basa en almacenar la información usando componentes eléctricos, como transistores o condensadores que pueden ser leídos o escritos con señales eléctricas. Respecto a las anteriores presenta la ventaja de no requerir partes móviles para acceder a la información, ya que lo que se «mueve» es la corriente eléctrica.

Mutabilidad

Otra característica relevante es la «mutabilidad» o «posibilidad de cambiar». En dispositivos como los discos duros o memorias flash es posible tanto leer información como modificar información, es decir permiten **escritura y lectura**. En dispositivos como los CD y DVD no regrabables, sólo se puede obtener la información y no modificar, es decir son de **sólo lectura**.

Capacidad y Costo

La capacidad y el costo están directamente relacionado a la tecnología usada, y en general se cumple la regla de que a mayor capacidad, mayor costo. La razón $\frac{capacidad}{costo}$ es un índice relevante que indica para una tecnología cuanto cuesta almacenar una cierta cantidad de información lo que será un factor importante para determinar que tipo de almacenamiento ocupar en las distintas partes del computador.

Volatilidad

Los dispositivos habitualmente usados para almacenar nuestra información tienen la característica de ser **no volátiles**, es decir, si no estamos entregándoles alimentación eléctrica (i.e. no están enchufados, ni tiene baterías), son capaces de mantener la información guardada. Es evidente que esto es de suma relevancia, ya que en caso contrario podría ocurrir, por ejemplo, que al apagar el computador perderíamos toda nuestra información del disco duro, lo que no tendría mucho sentido práctico.

Existen, sin embargo, dispositivos de almacenamiento que no tienen esta capacidad, es decir, son **volátiles** y si pierden alimentación eléctrica, pierden la información. La volatilidad no presenta ninguna ventaja comparativa, pero ocurre que las tecnologías más rápidas de almacenamiento suelen tener esta característica, por lo que a pesar de contar con esta desventaja, son usadas.

Rendimiento

Una última característica relevante tiene relación con el rendimiento del dispositivo. Hay dos elementos principales para evaluar el rendimiento: la **latencia** que se refiere el tiempo en que se demora el acceso a una particular pieza de información almacenada, la cual se mide en segundos o nanosegundos, y el **throughput** que representa la cantidad de información que se puede sacar del dispositivo en un cierto tiempo, lo que se mide en $\frac{bytes}{segundos}$.

Como regla general los dispositivos de almacenamiento eléctrico tendrán menor latencia y mayor throughput, ya que no requieren estar moviendo partes mecánicas (como si ocurre en el almacenamiento magnético y óptico).

4. Circuitos de almacenamiento

Al considerar que dispositivo de almacenamiento usar como medio principal en un computador la principal característica a tomar en cuenta es el rendimiento. Los computadores realizan operaciones a altas velocidades y por tanto queremos que los dispositivos de almacenamiento ocupados para para estas operaciones sean también rápidos.

Como se señaló previamente, los dispositivos de almacenamiento eléctrico son los que presentan un mejor rendimiento, por lo que estos serán los elegidos para el almacenamiento principal del computador. A continuación se explicará como se pueden construir estos dispositivos de almacenamiento ocupando las mismas herramientas de diseño de circuitos basados en compuertas.

4.1. Flip-flops

El componente básico para almacenar información usando circuitos eléctricos se denomina flip-flop. A diferencia de los circuitos lógicos vistos previamente la salida de un flip-flop depende no sólo de las compuertas que lo componente, sino de su estado previo, lo que se denomina como circuito secuencial.

Existen distintos tipos de flip-flops, que presentan sutiles variaciones en su diseño. El diseño más utilizado en la actualidad corresponde al flip-flop D con señal de control, que se observa en la figura 1.

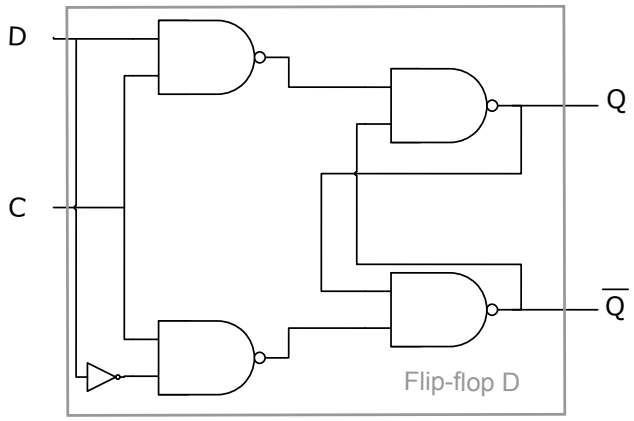

Figura 1: Flip-flop D con control.

El funcionamiento de este circuito es el siguiente: Si en la entrada de datos D llega un determinado valor y además la señal de control C está activa, el valor de salida Q de flip-flop se actualiza con el valor de D. Si en cambio, la señal del control C está desactiva, el valor de Q **es el mismo que tenía previamente**.

La tabla de valores del flip-flop D se observa a continuación:

C	D	Q
0	x	Q
1	0	0
1	1	1

Lo relevante de este circuito es que **tiene la capacidad de almacenar 1 bit**, dado que mientras la señal de control sea cero, se mantiene el valor guardado. Además, el circuito **permite modificar el valor almacenado** al activar la señal de control, y enviar un valor por la señal de datos.

De aquí en adelante interpretaremos un flip-flop con su abstracción de alto nivel: una caja que es capaz de almacenar 1 bit de información, y modificar su valor.

4.2. Registros

Un flip-flop D representa una unidad de almacenamiento de un bit. Si combinamos varias unidades de almacenamiento de 1 bit, podemos obtener unidades capaces de almacenar más información, las que se denomina **registros**. En la figura 2 se observa un registro de 4 bits formado por 4 flip-flops D.

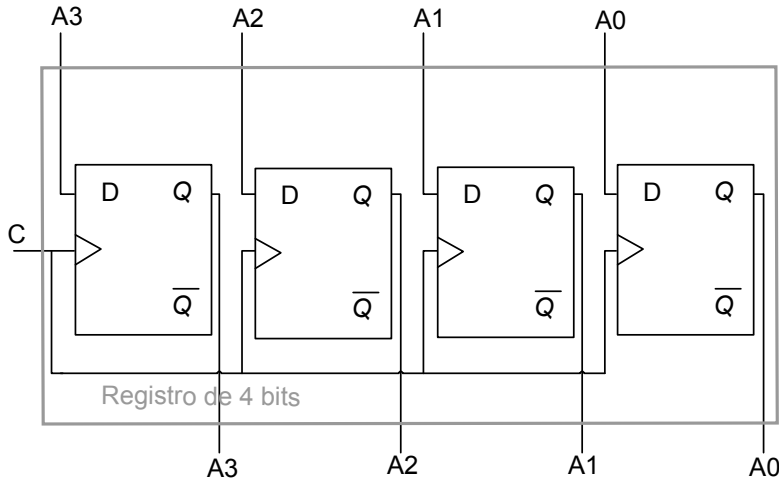

Figura 2: Registro de 4 bits formado por 4 flip-flops D.

El registro más simple, como el que se observa en la figura, sólo permite cargar un valor cuando la señal de control se activa. Podemos agregar más funcionalidades al registro, por ejemplo, un bit secundario de selección de carga L, y un bit que permita resetear el valor actual a 0, R, lo que se observa en la figura 3.

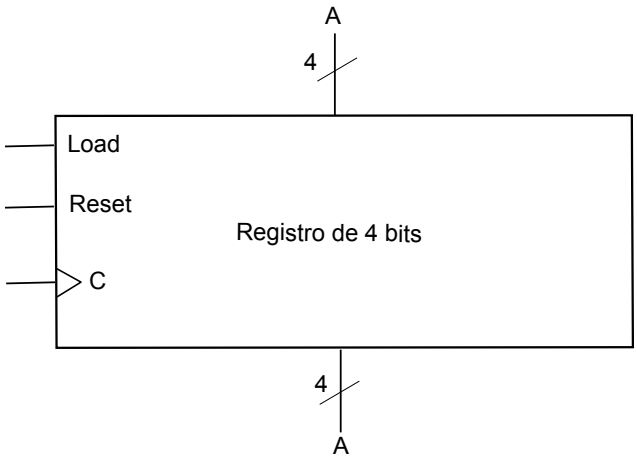

Figura 3: Registro de 4 bits con bit de carga y reset.

4.2.1. Contadores

A partir de flip-flops y registros podemos construir circuitos de almacenamiento más complejos, como por ejemplo un contador. Un **contador incremental** es un circuito que almacena un cierto número y luego de recibir una señal de incremento aumenta en 1 el valor actual. Un **contador decremental** será equivalente pero en vez de aumentar en 1 disminuirá el valor en 1. Podemos combinar ambas opciones en un contador incremental/decremental o up/down.

Al igual con el registro, podemos agregar más funcionalidades al contador, como bit de carga y reset, obteniendo el componente que se observa en la figura 4.

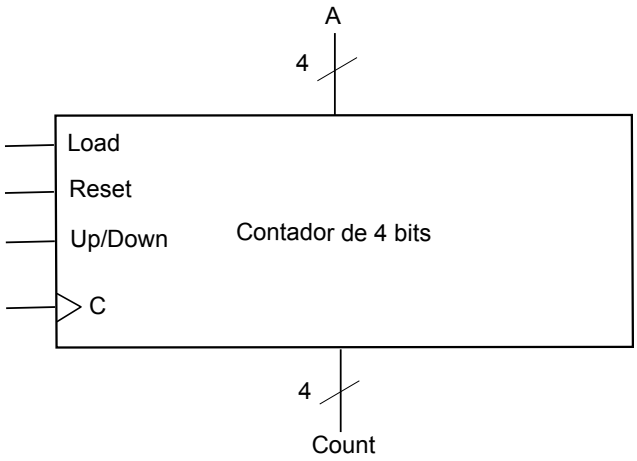

Figura 4: Contador incremental de 4 bits con carga y bit de reset.

4.3. Memorias

Los registros nos permiten almacenar cada uno una palabra o unidad de información. Nos gustaría poder escalar y tener un componente que almacene más información. Una opción simple es tener múltiples registros, pero para poder usarlos de esta manera tenemos que poder acceder a cada un de ellos, poder cargarlos y leer información, lo que hace complejo hacer el escalamiento.

El componente que queremos debe permitir almacenar varias palabras, a las cuales podamos acceder y también modificar fácilmente. El componente que cumple con esta función se denomina una **memoria**, las cuales almacenan una cierta cantidad de palabras de un determinado tamaño y permite acceder a estas asociando a cada palabra un número o identificador representado por un número binario.

El identificador numérico usado para acceder a una palabra se denomina **dirección de memoria** y por consiguiente el proceso de acceder a una de las palabras se conoce como **direccionamiento**. Dado esto, la memoria puede ser pensada como una tabla de pares de valores: para cada dirección se asocia una palabra.

Dirección en decimal	Dirección en binario	Palabra
0	000	$Palabra_0$
1	001	$Palabra_1$
2	010	$Palabra_2$
3	011	$Palabra_3$
4	100	$Palabra_4$
5	101	$Palabra_5$
6	110	$Palabra_6$
7	111	$Palabra_7$

4.3.1. Tipos de memoria

Revisaremos dos tipos de memorias relevantes: la **random access memory** o **RAM** y la **read only memory** o **ROM**.

Memorias de escritura-lectura: RAM

La memoria RAM es una extensión de los registros. Al igual que estos, ocupa como unidad de almacenamiento de un bit un flip-flop, sin embargo a este se le agregan componentes para permitir cumplir con las funciones anteriormente descritas, formando lo que se denomina una **celda de memoria** o **memory cell**.

El diagrama de una memoria RAM se observa en la figura 5. La memoria RAM tiene tres buses de entrada: un bus de direccionamiento de k bits, que indica que palabra se quiere seleccionar (entrada *Address*); un bus de datos de n bits, que tiene el dato que se guardará en la posición seleccionada (entrada *Data In*); y un bus de control de 1 bit que indica si la memoria está en modo lectura (0) o escritura (1) (entrada *Write Enable* o *WE*). Adicionalmente cuenta con un bus salida de datos de n bits, que tiene el dato indicado por la dirección recibida (salida *Data Out*). El número n será el tamaño de la palabra, el número k indica la cantidad de bits disponibles para direccionar, y por tanto 2^k será la cantidad de palabras distintas que se pueden almacenar.

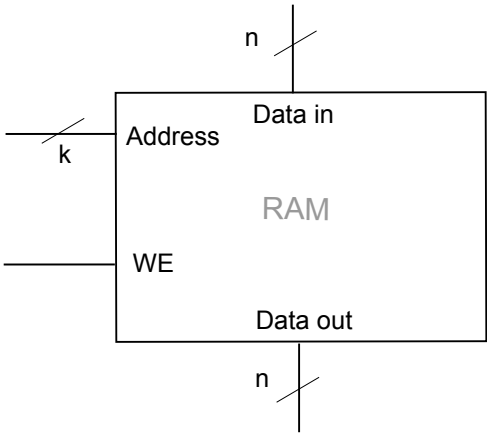

Figura 5: RAM de 2^k palabras de n bits.

La RAM tiene dos modos de funcionamiento: escribir o leer. Veamos ambos modos a través de un ejemplo: supongamos que tenemos una RAM de 2 bits de direccionamiento, es decir $2^2 = 4$ palabras, y que cada palabra es de 8 bits = 1 byte. El diagrama de esta memoria se puede observar en la figura 6.

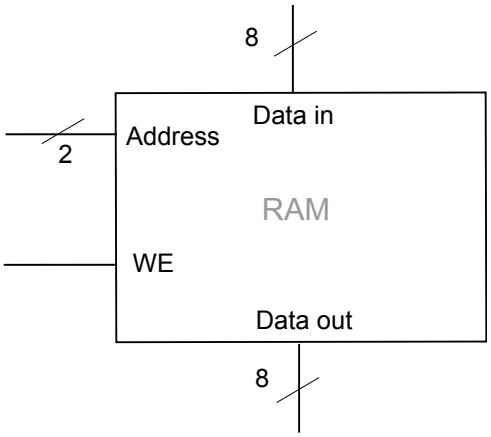

Figura 6: RAM de 2^2 palabras de 8 bits.

Supongamos que la información inicial que tenemos en la RAM es la siguiente:

Dirección en decimal	Dirección en binario	Palabra en binario
0	00	00011010
1	01	01000010
2	10	00000000
3	11	00111111

Para leer la dirección 1 debemos colocar el valor 01 en la entrada *Address*, el valor 0 en la entrada *WE*, y obtendremos el valor 01000010 en la salida *Data Out*. Al igual que en los registros, en este modo lo que está en la entrada *Data In* no va a entrar.

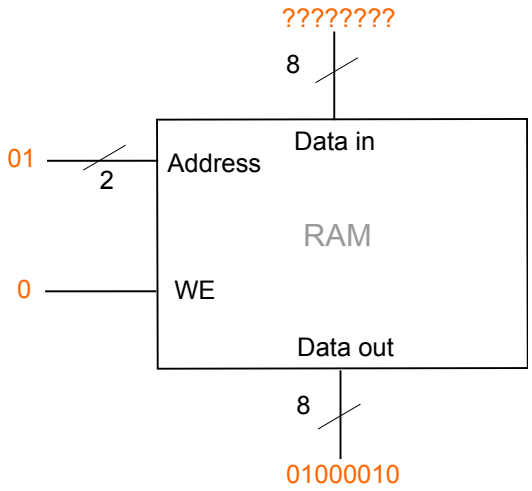

Figura 7: RAM en modo lectura.

Para escribir en la dirección 2 debemos colocar el valor 10 en la entrada *Address*, el valor 1 en la entrada *WE*, y el dato que queremos guardar en la entrada *Data In*, por ejemplo 11110000.

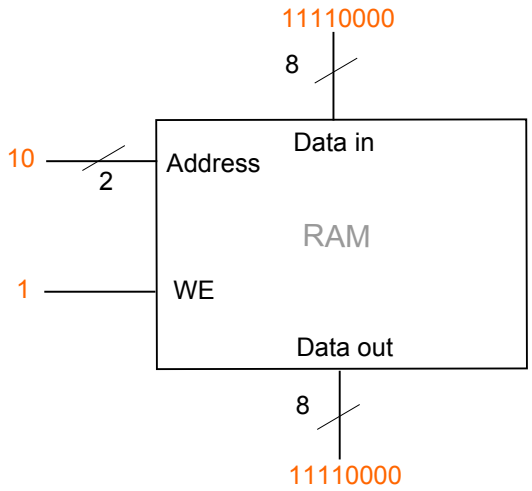

Figura 8: RAM en modo escritura.

Memorias de sólo lectura: ROM

Un segundo tipo de memoria relevante son las memorias de sólo lectura o ROM. El concepto de memoria de «solo lectura» suena confuso, ya que si no se puede escribir información inicialmente, no tiene mayor sentido como memoria. Hablar de «solo lectura» en la práctica se interpreta como «se puede escribir sólo una vez» o también en algunos casos «se puede escribir más de una vez, pero a una velocidad y costo mucho mayor que leer».

Dada esta definición, la utilidad de una memoria de este tipo es para almacenar datos que no variarán en el tiempo o que variarán poco. El hecho de que una memoria sea de «sólo lectura» tiene que ver con la tecnología usada. A diferencia de la RAM, no se utilizan flip-flops para almacenar bits, sino que otros mecanismo. En la memoria ROM original y más simple el proceso de escritura consiste en soldar o dejar abiertos conexiones entre cables, lo que se interpreta como un 1 o 0.

El diagrama de la memoria ROM se observa en la figura 17. La memoria ROM es más simple que la RAM: tiene un sólo bus de entrada, el bus de direccionamiento de k bits, que indica que palabra se quiere seleccionar (entrada *Address*) y un bus salida de datos de n bits, que tiene el dato indicado por la dirección recibida (salida *Data Out*). Al igual que en la RAM el número n será el tamaño de la palabra, el número k indica la cantidad de bits disponibles para direccionar, y por tanto 2^k será la cantidad de palabras distintas que se pueden almacenar.

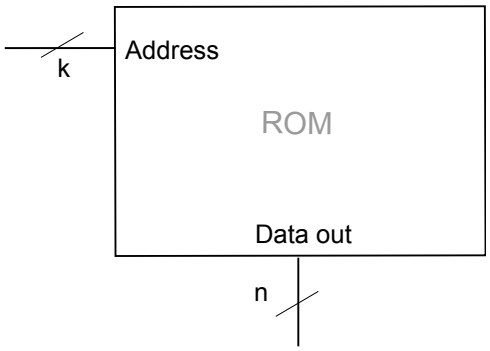

Figura 9: ROM de 2^k palabras de n bits.

5. Representación de Variables y Arreglos en Registros y Memorias

5.1. Variables y Registros

Para almacenar una variable necesitábamos una representación física que permita almacenar una secuencia de bits, del tamaño indicado por el tipo de dato, además de poder identificar de alguna forma la representación para diferenciarla de otras. Los registros son una primera alternativa para lograr esto. Como se observó previamente, un registro es básicamente un conjunto de flip-flops, es decir, cajas que almacenan bits. Dependiendo del tipo de dato, podríamos tener registros de distinto tamaño, que permitan representar los distintos tipos. Si queremos un tamaño único de registro, necesitamos que sea del tamaño del tipo de datos de mayor cantidad de bits. Por ejemplo, si queremos almacenar a lo más tipos de datos de 32 bits (como el tipo `int` de Java), necesitamos registros de 32 bits. Con estos registros también podemos representar tipos con menor cantidad de bits, por ejemplo de 8 bits (como el tipo `byte` de Java), para hacerlo basta con ocupar 8 bits de los 32, y no considerar el resto.

Para implementar la identificación, es necesario tener distintos registros para las distintas variables, de manera de poder decidir a que registro acceder conociendo el nombre de este. Por ejemplo si queremos tener dos variables, necesitamos que existan dos registros distintos al menos, que podríamos llamar

A y B, y poder acceder a estos de manera diferenciada. Esto muestra una limitante del uso de registros para almacenar variables: necesitaríamos tantos registros como variables posibles para permitir la identificación.

En la práctica, los registros son ocupados para almacenar variables, pero solo de manera temporal, justo antes de realizar operaciones sobre éstas. Dado que los registros representarán solo variables temporales, no será necesario tener tantos registros como variables, y por tanto necesitaremos otro mecanismo mas escalable que permita almacenar una cantidad mayor de variables.

5.2. Variables y Memorias

Las memorias RAM representan un mecanismo ideal para evitar los problemas de los registros y permitir almacenar una mayor cantidad de variables. Como se detalló previamente, las memorias pueden ser pensadas como tablas, en que por un lado se tiene una dirección y por otra parte una palabra, que corresponde a un a secuencia de bits de un cierto tamaño. Esta definición nos indica directamente que las memorias poseen los dos elementos que necesitamos para almacenar variables: un identificador, que en este caso serían las direcciones; y una lugar donde almacenar una secuencia de bits, asociado al identificador.

Una elemento importante que se debe considerar es que un número almacenado en memoria puede estar guardado en varias palabras. Por ejemplo, si el tamaño de las palabras de la memoria es 1 byte = 8 bits (que es el tamaño que habitualmente se utiliza), y queremos almacenar el número de punto flotante de tipo **single precisión** (`float`), siguiendo el estándar IEEE754 necesitamos 32 bits = 4 bytes = 4 palabras. De esta forma, para poder obtener un número de memoria, necesitamos saber a priori al menos de que tipo es el número, y cuantas palabras ocupa.

La información del tipo de dato y su tamaño puede no ser suficiente. Siguiendo el ejemplo anterior, tomemos un número de tipo `float`, por ejemplo el $0{,}5 = 0{,}1_2$ el cual se representa según el estándar con la secuencia de bits: 00111111000000000000000000000000. Como las palabras de nuestra memoria eran de tamaño 8 bits, debemos dividir nuestra secuencia en 4 partes: 00111111, 00000000 ,00000000 y 00000000, las cuales podríamos almacenar a partir de la dirección 0 de la siguiente forma:

Dirección en hexa	Dirección en binario	Palabra
0x00	000	00111111
0x01	001	00000000
0x02	010	00000000
0x03	011	00000000
0x04	100	$Palabra_4$
0x05	101	$Palabra_5$
0x06	110	$Palabra_6$
0x07	111	$Palabra_7$

El problema está en que esta es una de las formas en que podemos almacenar las 4 palabras que corresponde al número. De manera equivalente, podríamos almacenar las 4 palabras en el orden contrario:

Dirección en hexa	Dirección en binario	Palabra
0x00	000	00000000
0x01	001	00000000
0x02	010	00000000
0x03	011	00111111
0x04	100	$Palabra_4$
0x05	101	$Palabra_5$
0x06	110	$Palabra_6$
0x07	111	$Palabra_7$

Esto nos indica que además de saber el tipo del número y su tamaño asociado, debemos definir un orden en como se guardarán las palabras que compone el número. Este orden de las palabras se denomina **endianness**, y existen dos posibles formas de ordenar: **big endian**, en la que la palabra más significativa dentro del número (i.e. con los bits en las posiciones más altas) se almacena en la dirección menor (como el primer caso del ejemplo); **little endian**, en la que la palabra más significativa dentro del número (i.e. con los bits en las posiciones más altas) se almacena en la dirección mayor (como el segundo caso del ejemplo). Al diseñar un computador, se deberá definir cual de los dos órdenes se utiliza, y mantener la consistencia del orden para todos los tipos de datos.

5.3. Arreglos y Memorias

Las memorias también resultan útiles para almacenar estructuras de datos más complejas como los arreglos. Un arreglo requería un identificador de la lista de valores, la capacidad de guardar múltiples valores, y la posibilidad de acceder a estos múltiples valores de manera indexada. Veamos como almacenaríamos en memoria un arreglo unidimensional, comenzando con un arreglo de valores de tipo `byte`, con el siguiente contenido: $0x00, 0x05, 0x0A, 0x0F$.

Dirección en hexadecimal	Dirección en binario	Palabra
0x00	000	$0x00$
0x01	001	$0x05$
0x02	010	$0x0A$
0x03	011	$0x0F$

Se observa que podemos ocupar distintas direcciones de memoria para guardar los distintos valores. El identificador del arreglo va a corresponder a la dirección del primer valor, en este caso, estamos guardando el arreglo a partir de la dirección 0x00. Se observa también que para poder acceder al i-ésimo elemento, nos basta con sumarle i a la dirección del primer valor. Por ejemplo, para acceder al valor arreglo[1], accedemos a la dirección de arreglo (0x00) sumada con la posición (0x01), es decir accedemos a la dirección 0x01. Esta es en parte la razón por la cual los arreglos comienzan con índice 0.

El largo del arreglo en general se almacenará de manera independiente, y al acceder el arreglo habrá que estar al tanto de este largo para no pasarnos de entre los valores válidos.

Al igual que como se mencionó en el caso de las variables, un arreglo puede almacenar valores de tipos que tengan un tamaño mayor que la palabra. Revisemos un ejemplo de un arreglo de char (palabras de 2 bytes, 16 bits) $0x0000, 0x0005, 0x000A, 0x000F$:

Dirección en hexadecimal	Dirección en binario	Palabra
0x00	000	$0x00$
0x01	001	$0x00$
0x02	010	$0x00$
0x03	011	$0x05$
0x04	100	$0x00$
0x05	101	$0x0A$
0x06	110	$0x00$
0x07	111	$0x0F$

En este caso nuevamente surge el concepto de endianness: cada valor del arreglo deberá guardarse en un cierto orden (big o little endian), lo cual debe estar convenido previamente para poder interpretar correctamente el número. Otra diferencia de este caso es que ahora para poder indexar el i-ésimo valor del arreglo no basta con sumar la dirección del arreglo (0x00) con el índice. La fórmula correcta para el caso general de arreglos con tipos de tamaño distinto a 1 byte es: $dir(arreglo[i]) = dir(arreglo) + i \times sizeof(arreglo[i])$, donde $sizeof(arreglo[i])$ retorna el tamaño del tipo de dato del arreglo en bytes. Para el ejemplo del arreglo de valores de 2 bytes, si queremos acceder a la posición 2 del arreglo, debemos acceder a la dirección: $dir(arreglo[2] = 0x00 + 2 \times 2 = 0x04$.

5.3.1. Arreglos Multidimensionales

Un caso más complejo son los arreglos multidimensionales. Las memorias se prestan de manera natural, como se mostró anteriormente, para almacenar arreglos unidimensionales. Para arreglos de mayores dimensiones la solución no es tan directa y se debe definir la forma como se guarda el arreglo en memoria.

Para el caso de matrices (arreglos de 2 dimensiones), existen dos convenciones: guardar la secuencia de filas o guardar la secuencia de columnas. Por ejemplo suponiendo la matriz de 2x3:

0x02	0x04	0x07
0x05	0xA	0x09

Guardándola con la convención de filas obtendríamos en memoria:

Dirección en hexadecimal	Dirección en binario	Palabra
0x00	000	$0x02$
0x01	001	$0x04$
0x02	010	$0x07$
0x03	011	$0x05$
0x04	100	$0x0A$
0x05	101	$0x09$

Guardándola con la convención de columnas obtendríamos en memoria:

Dirección en hexadecimal	Dirección en binario	Palabra
0x00	000	$0x02$
0x01	001	$0x05$
0x02	010	$0x04$
0x03	011	$0x0A$
0x04	100	$0x07$
0x05	101	$0x09$

La convención de filas es la más habitualmente usada. En este caso, para acceder a un valor matriz[i,j] se debe ocupar la fórmula: $dir(matriz[i,j]) = dir(matriz) + i \times sizeof(matriz[i,j]) \times columnas + j \times sizeof(matriz[i,j])$

En nuestro ejemplo, para acceder al valor [1,1], en convención de filas, debemos acceder a la dirección: $dir(matriz[1,1]) = 0x00 + 1 \times 1 \times 3 + 1 \times 1 = 0x04$

6. Referencias

- Shannon, C.; A Symbolic Analysis of Relays and Switching Circuits, MIT Press, 1937.