Módulo 08

La Arquitectura von Neumann (Pt. 2)

Organización de Computadoras
Depto. Cs. e Ing. de la Comp.
Universidad Nacional del Sur

Copyright

- Copyright © **2011-2024** A. G. Stankevicius
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la **GNU** Free Documentation License, Versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera
- Una copia de esta licencia está siempre disponible en la página **http://www.gnu.org/copyleft/fdl.html**
- La versión transparente de este documento puede ser obtenida de la siguiente dirección:

http://cs.uns.edu.ar/~ags/teaching

Contenidos

- Organización multinivel
- Concepto de máquina virtual
- Niveles en una máquina contemporánea
- Evolución histórica de los niveles
- Tecnologías de memoria principal
- Organización del procesador
- Arquitecturas de n-direcciones
- Subsistema de Entrada/Salida

Organización de la memoria

- La memoria se organiza como un arreglo de **n** celdas, cada una de **k** bits
  - Cada celda o locación cuenta con un identificador de $\log_2(n)$ bits que la caracteriza denominado dirección
  - De igual forma, cada locación o celda almacena un contenido de **k** bits
  - Nótese que la menor unidad direccionable es la celda, si bien la unidad básica es el bit
  - El tamaño en bits de la celda es independiente de la cantidad de bits de las direcciones

Dirección	Contenido
0000:	01110010
0001:	11100101
0010:	00011001
0011:	01110010
0100:	00100101
⋮	⋮
1110:	00111001
1111:	10010110

Organización de la memoria

- La norma dicta que 8 bits componen un byte y que los bytes se agrupan en palabras
  - Una palabra suele tener 8, 16, 32 o 64 bits
  - La cantidad de bits por palabra está relacionada con el tamaño de los registros del procesador
  - Las instrucciones en general operan sobre palabras
- Las operaciones básicas con la memoria son:
  - Leer una palabra de la memoria
  - Escribir una palabra a la memoria

Ordenamiento de los bytes

- En caso de que una palabra esté compuesta por múltiples bytes, ¿cómo se deben ordenar?
  - El esquema big-endian postula que se debe numerar los bytes de izquierda a derecha
  - En contraste, el esquema little-endian postula que se deben numerar los bytes derecha a izquierda
- La decisión respecto a qué esquema utilizar poco afecta el desempeño del procesador
  - Puede que resulte conflictivo transmitir palabras entre computadoras que no usen el mismo esquema

Ordenamiento de los bytes

tamaño de palabra: 32 bits

mensaje original: Hola Mundo! 20/10/2024

	00	01	10	11
000	'H'	'o'	'l'	'a'
001	' '	'M'	'u'	'n'
010	'd'	'o'	'!'	0
011	20	'/'	10	'/'
100	232	7		

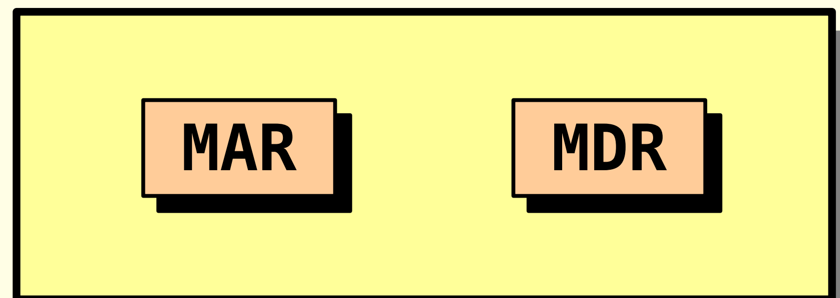
big-endian

	00	01	10	11
000	'a'	'l'	'o'	'H'
001	'n'	'u'	'M'	' '
010	0	'!'	'o'	'd'
011	'/'	10	'/'	20
100			7	232

little-endian

Interfaz con la memoria

- La unidad controladora de la memoria cuenta esencialmente con dos registros:
  - El registro **MAR** el cual almacena la dirección de una locación de memoria
  - El registro **MDR** el cual almacena el contenido de una locación de memoria

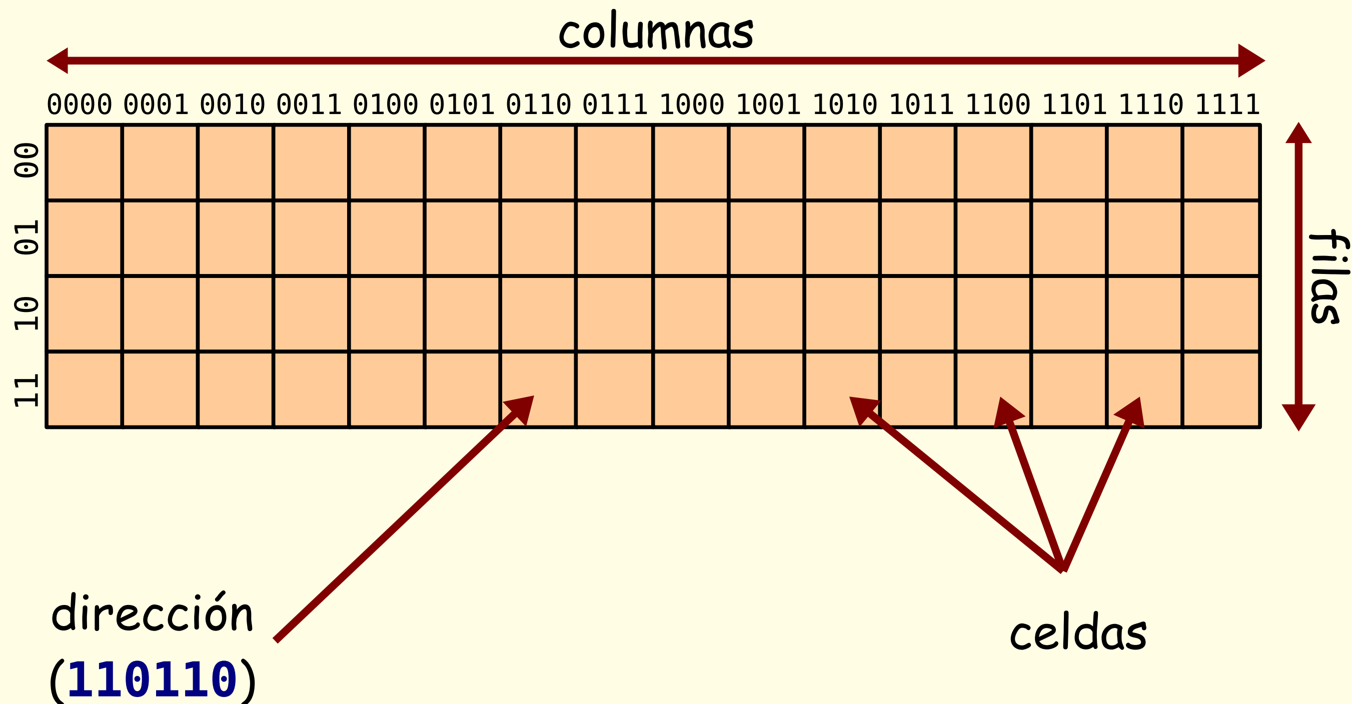


unidad controladora de la memoria

Interfaz con la memoria

- Operación de lectura (**load**):
  - El procesador escribe la locación que se desea acceder en el registro **MAR**, luego activa la señal de lectura y finalmente accede a la palabra en cuestión la cual estará disponible en el registro **MDR**
- Operación de escritura (**store**):
  - El procesador escribe la locación que se desea acceder en el registro **MAR** y el contenido que se desea almacenar en el registro **MDR** y por último activa la señal de escritura

Organización física



tamaño: (cant. filas) x (cant. cols) x (bits x celda)

Diagrama lógico de detalle

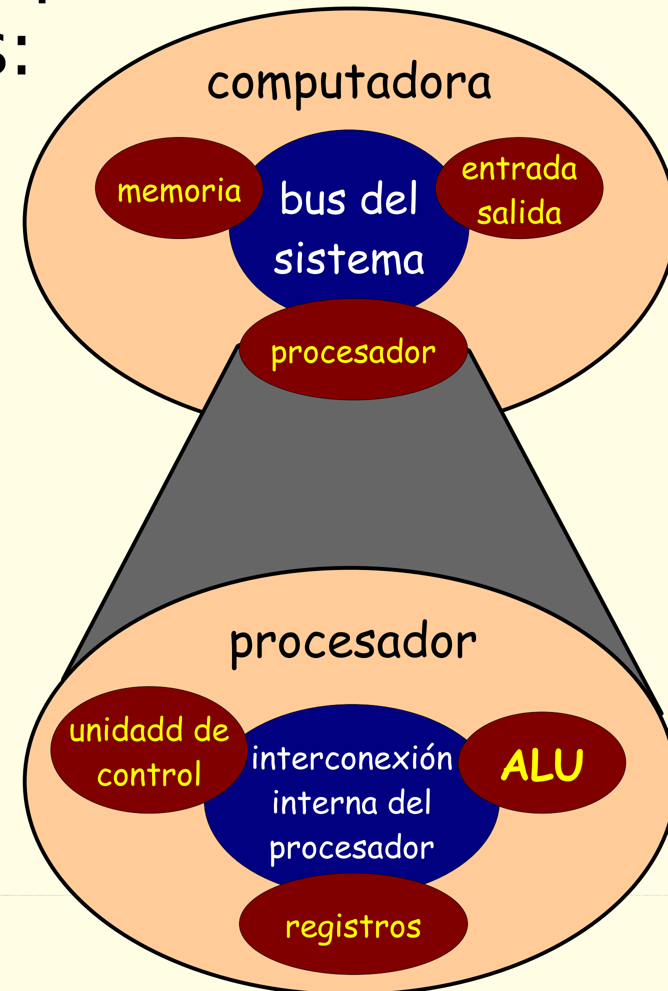
Data in
I_2
I_1
I_0
Write gate
Word 0 select line
Word 1 select line
Word 2 select line
A_1
A_0
Word 0
Word 1
Word 2
Word 3
D Q
CK
CS • $\overline{RD}$
CS
RD
OE
O_1
O_2
O_3
Output enable = CS • RD • OE

Organización del procesador

- El procesador incluye otros dos componentes básicos de la arquitectura von Neumann:
  - La unidad aritmético lógica (**ALU**)
  - La unidad de control
- El procesador tiene como principal tarea orquestar el adecuado funcionamiento de la computadora
  - La **ALU** llevará adelante el procesamiento de los datos de la manera en que el programa en ejecución le vaya indicando

Organización del procesador

- Internamente el procesador se puede descomponer en cuatro partes:
  - La unidad de control
  - La unidad aritmético lógica
  - Los registros del procesador
  - Las interconexiones entre estos componentes
- Nótese la similitud entre la organización del procesador y de la computadora



Data path vs. control path

Unidad Aritmético Lógica

Indicadores de estado

Desplazador

Complementador

Lógica Arirtmética y Boolenana

Bus interno de la CPU

Registros

Unidad de Control

Caminos de control

Funciones del procesador

- Funciones desempeñadas por el procesador:
  - Traer de memoria principal la próxima instrucción del programa en ejecución
  - Determinar de qué instrucción se trata (decodificar)
  - Obtener todos operandos referidos en la instrucción en curso
  - Llevar adelante el procesamiento de datos sancionado por esa instrucción
  - Almacenar dónde corresponda los resultados

Ciclo básico del CPU

- Para cumplir con las tareas asignadas el procesador repite infinitas veces el siguiente ciclo básico de operación:
  - Etapa Fetch: se almacena en el registro **IR** la instrucción apuntada por el registro **PC**
  - Etapa Decode: durante esta etapa se determina de qué instrucción se trata
  - Etapa Effective Address: se calcula la dirección efectiva referida por la instrucción (si es que alguna) y/o se recuperan todos los operandos necesarios para comenzar a ejecutar la instrucción en curso

Ciclo básico del CPU

- Continúa:
  - Etapa Execute: conociendo de qué instrucción se trata y contando con los operandos que sean necesarios, durante esta etapa se programa a la **ALU** para que lleve adelante el procesamiento correspondiente
  - Etapa Memory: las arquitecturas tipo **RISC** cuentan con una etapa específica donde se accede a memoria para leer o escribir una determinada locación
  - Etapa Write-Back: usualmente el resultado obtenido se almacena en alguno de los registros del procesador durante esta etapa

Ciclo básico del CPU

FETCH → DECODE → EFFECTIVE ADDRESS → EXECUTE → MEMORY → WRITE BACK → FETCH

Ejemplo

Datos
Direcciones
Control
PC
MAR
Dout
Din
IR
B. Reg.
Mem.
Perif.
Unidad de control

analicemos la evolución de
la computadora al ejecutar
este fragmento de código

```
add R1, R2, R3
load R3, [R5]
bnz fin, R3
```

Ejemplo

3
1
Datos
Direcciones
Control
PC
MAR
Dout
Din
IR
Mem.
Perif.
2
R2
B. Reg. R1
R3
4
Unidad de control

```
add R1, R2, R3
load R3, [R5]
bnz fin, R3
```

Ejemplo

3 5
1
Datos
Direcciones
Control
4
PC
MAR
Dout
Din
IR
Mem.
Perif.
2
R5
R3
B. Reg.
Unidad de control

```
   add R1, R2, R3
➡ load R3, [R5]
   bnz fin, R3
```

Ejemplo

Datos
Direcciones
Control

1
3

PC
MAR
Dout
Din
IR
Mem.
Perif.

2
4
R3
B. Reg.
fin

Unidad de control

```
add R1, R2, R3
load R3, [R5]
bnz fin, R3
```

Direccionamiento simbólico

- En un lenguaje de alto nivel es frecuente ver expresiones simbólicas tales como **$x = y + z$**
- Al bajar de nivel, estas referencias simbólicas deben ser reemplazadas por la dirección en memoria donde se almacenen esas variables
  - Debemos tener en cuenta que la dirección de una locación de memoria es a su vez un número

$$x = y + z \quad \rightarrow \quad 123 = 15 + 83$$

  - Para diferenciar una dirección de su contenido haremos uso de pares de paréntesis

$$(123) = (15) + (83)$$

Registros del procesador

- Los registros son un almacenamiento temporario de muy alta velocidad
  - Se organizan en un banco de registros, el cual suele contar con múltiples puertos de lectura y/o escritura
- La cantidad de bits de los registros coincide con el tamaño de palabra del procesador
  - El tamaño de palabra nos da una idea aproximada de la capacidad de procesamiento del hardware (en particular, de la **ALU**)

Registros del procesador

- Los registros del procesador se clasifican en dos categorías:
  - Registros accesibles por el usuario: estos registros puede ser accedidos por el usuario a través de las distintas instrucciones
  - Registros internos del procesador: estos registros son utilizados exclusivamente por la unidad de control durante el desarrollo del ciclo básico del procesador

Registros del procesador

- Los registros accesibles al usuario se clasifican a su vez en otras dos categorías:
  - Registros de propósito general: estos registros están disponibles para que el programador haga un uso discrecional de los mismos (por caso, el registro acumulador **EAX** o el registro índice **ECX**)
  - Registros de propósito específico: estos registros usualmente tienen asignado un rol por defecto (por caso, el registro **ESP** que siempre apunta al tope de la pila del programa)

Registros del procesador

- Uno de los registros internos del procesador es el **PSW** (processor status word), el cual codifica el estado actual del procesador
- Los registros internos del procesador sólo pueden ser alterados de manera indirecta
  - Por caso, el devenir de la ejecución de un programa va alterando los registros internos **PC** e **IR**
  - De manera análoga, el resultado de la última operación realizada afectará al registro interno **PSW**

Acceso a los operandos

- Una de las etapas del ciclo básico del procesador requiere ganar acceso a los operandos de la instrucción en curso
- A lo largo de la evolución de la computación se ensayaron muy diversas formas de especificar la ubicación de estos argumentos
- Estas marcadas diferencias permiten clasificar a las distintas arquitecturas en función cuántas referencias a memoria tienen las instrucciones de cómputo

Arquitectura 0-address

- La arquitectura 0-address sólo cuenta con instrucciones que refieren a sus operandos de manera implícita
  - También se la conoce como arquitectura pila
  - La pila del sistema es origen y destino implícito de todas las operaciones aritmético-lógicas
  - Naturalmente, las instrucciones de acceso a memoria son una excepción, ya que sí explicitan la locación que se está accediendo

Ejemplo

- Fragmento de código en una arquitectura 0-address que evalúa la siguiente expresión:

A, B, C, D y E son locaciones de memoria las cuales contienen los valores sobre los cuales se desean operar

A = B + C x D – C / E

```
push (B)
push (C)
push (D)
mpy
push (C)
push (E)
div
sub
add
pop (A)
```

se debe acordar el orden de apilado en las operaciones no conmutativas

Arquitectura 1-address

- La arquitectura 1-address cuenta con instrucciones en las que se especifica sólo uno de los argumentos
  - También se la conoce como arquitectura de acumulador único
  - Existe un registro especial llamado acumulador el cual es origen y destino implícito de todas las operaciones aritmético-lógicas
  - Puede contar con registros auxiliares los que se emplean para implementar modos de direccionamiento más complejos

Ejemplo

- Fragmento de código en una arquitectura 1-address que evalúa la siguiente expresión:

tmp es una locación arbitraria de memoria usada para almacenar temporalmente un resultado intermedio

A = B + C x D – C / E

A también puede ser usado como almacenamiento intermedio puesto que su contenido es destruido por la asignación

```
load C
mpy (D)
store tmp
load C
div (E)
store A
load B
add (tmp)
sub (A)
store A
```

Arquitectura 1-address + reg

- La arquitectura 1-address + registro cuenta con instrucciones en las que está permitido especificar un registro general a la par de una dirección de memoria
  - La arquitectura Intel **x86** pertenece a esta categoría
  - En las operaciones con dos operandos de entrada el registro especificado dentro de la instrucción se convierte en el destino implícito del resultado
  - Este tipo de arquitectura cuenta con la instrucción **MOV** para mover información desde y hacia memoria y también entre los registros

Ejemplo

- Fragmento de código en una arquitectura 1-address + reg para la siguiente expresión:

en algunas arquitecturas se utilizan corchetes en lugar de los paréntesis

A = B + C x D – C / E

asumiremos que el destino de las operaciones se indica primero (al igual que en la arquitectura **x86**)

```
mov R0, [C]
mov R1, R0
mul R0, [D]
div R1, [E]
add R0, [B]
sub R0, R1
mov [A], R0
```

Arquitectura 2-addresses

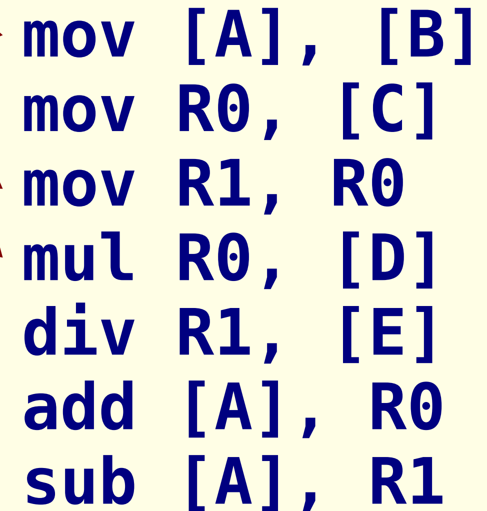
- La arquitectura 2-addresses cuenta con instrucciones en las que está permitido especificar hasta dos direcciones de memoria
  - En las operaciones con dos operandos de entrada, uno de los argumentos especificado dentro de la instrucción se convierte en el destino implícito
  - Este tipo de arquitectura también cuenta con la instrucción **MOV** para mover información libremente entre memoria y registros

Ejemplo

- Fragmento de código en una arquitectura 2-addresses para la siguiente expresión:

en esta arquitectura también asumiremos que el destino de las operaciones se indica primero

A = B + C x D – C / E



Arquitectura 3-addresses

- La arquitectura 3-addresses cuenta con instrucciones en las que está permitido especificar hasta tres direcciones de memoria
  - Este tipo de arquitectura evidentemente brinda la máxima flexibilidad a los programadores del sistema que estén a cargo de implementar los distintos compiladores
  - No obstante, el tamaño de las instrucciones ha de variar en gran medida en función de la cantidad de direcciones de memoria que se especifiquen

Ejemplo

- Fragmento de código en una arquitectura 3-addresses para la siguiente expresión:

```
A = B + C x D – C / E
```

```
mul [A], [C], [D]
div R0, [C], [E]
add [A], [B], [A]
sub [A], [A], R0
```

como se puede apreciar, este tipo de arquitectura genera código muy compacto... ¿pero a qué precio?

Arquitectura reg. a reg.

- La arquitectura registro a registro cuenta con instrucciones que sólo operan sobre registros
  - Todas las arquitecturas tipo **RISC** (por caso, la **ARM**) adoptan esta configuración
  - De manera análoga a la arquitectura pila, se cuenta con instrucciones especificas de acceso a memoria las cuales permiten especificar usualmente a lo sumo una dirección de memoria
  - Los modos de direccionamiento más avanzados se logran combinando esa dirección de memoria con uno o más registros

Ejemplo

- Fragmento de código en una arquitectura registro a registro para la siguiente expresión:

asumiremos que la única instrucción que permite referir una dirección de memoria es la instrucción **lda** (load address)

A = B + C x D – C / E

```
lda R0, B
load R1, (R0)
lda R0, C
load R2, (R0)
lda R0, D
load R3, (R0)
lda R0, E
load R4, (R0)
mul R5, R2, R3
div R6, R2, R4
add R1, R1, R5
sub R1, R1, R6
lda R0, A
store (R0), R1
```

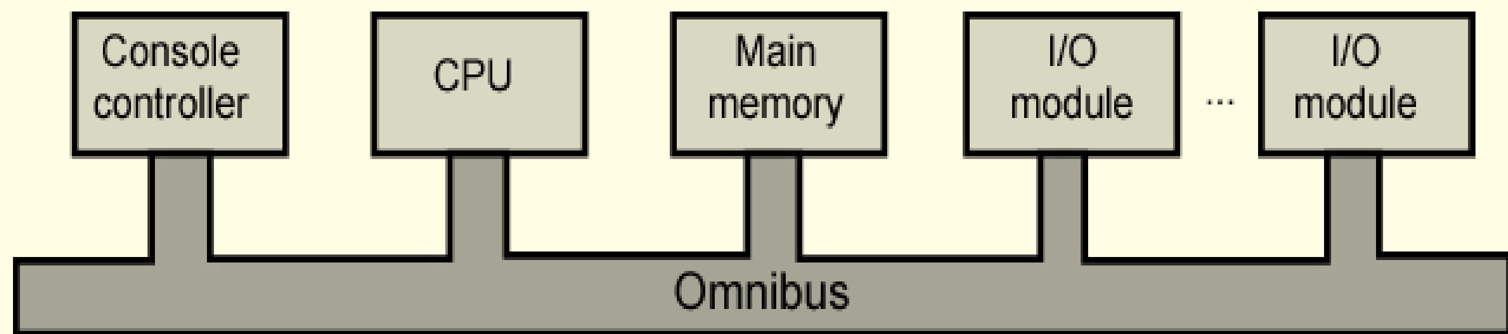
analizar este fragmento de código en busca de alguna posible optimización extra en el uso de los registros

Entrada / Salida

- El último componente de la arquitectura von Neumann que nos resta abordar se lo conoce como entrada/salida
- Las computadoras rápidamente demandaron almacenar más información de la que se podía almacenar en la memoria principal
- En consecuencia, a lo largo del tiempo se han ensayado distintas tecnologías para implementar esta memoria secundaria

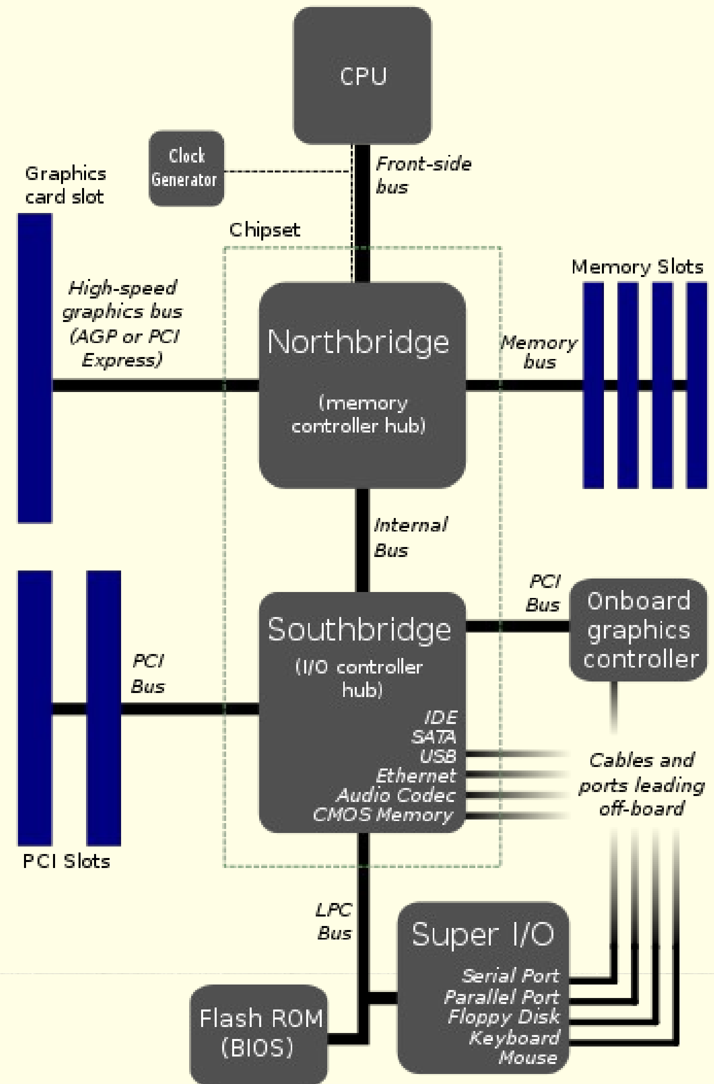
Bus del sistema

- La arquitectura von Neumann presenta un serio cuello de botella: el canal de comunicación entre el procesador en resto de la computadora
- Las primeras computadoras adoptaban un esquema de bus único
  - Por ejemplo, el Omnibus de la **PDP-8**:



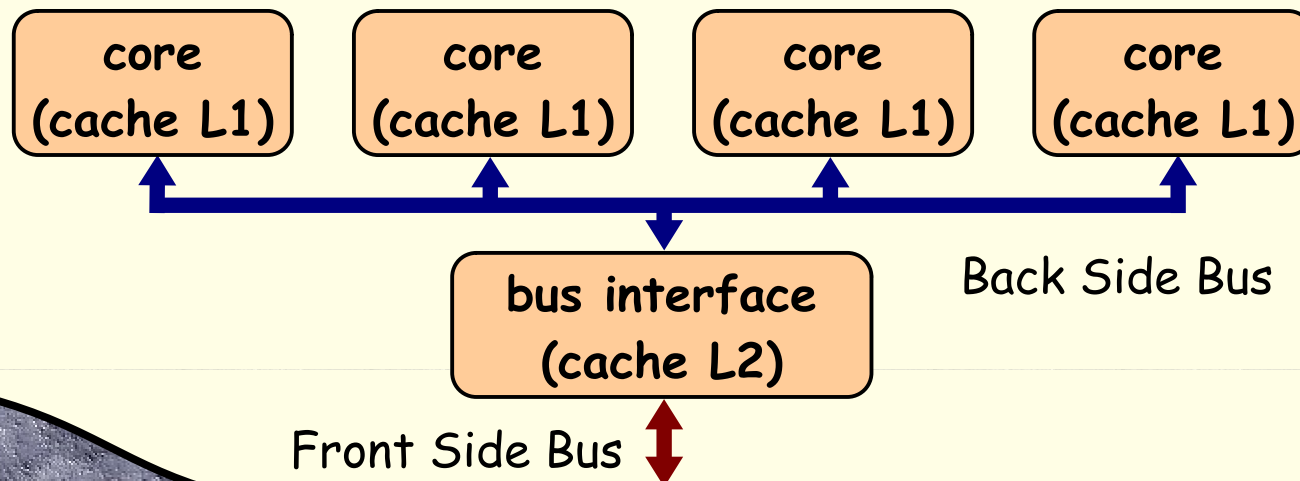
Bus del sistema

- Las arquitecturas más recientes empezaron a hacer uso de múltiples buses independientes
  - Por caso, una típica **PC** de escritorio cuenta con gran cantidad de buses de propósito específico
  - Hoy en día las funciones del northbridge se han incorporado al procesador



FSB vs. BSB

- En lo que al **CPU** respecta, cuenta básicamente con dos buses:
  - El Front Side Bus (**FSB**) para interconectar el procesador con el resto de la computadora
  - El Back Side Bus (**BSB**) para interconectar los componentes internos del procesador entre sí



¿Preguntas?