# Módulo 09

# Modos de Direccionamiento (Pt. 1)


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

# Copyright



# Contenidos

- Tipos de instrucciones
- Formato de instrucción
- Modos de direccionamiento
  - Directo vs. Indirecto
  - Absoluto vs. Relativo
- Código automodificable
- Código independiente de la posición
- La arquitectura **OCUNS**

# Tipos de instrucciones

- Las instrucciones que componen el set de instrucciones de una cierta arquitectura se clasifican en las siguientes categorías:
  - De procesamiento: las instrucciones usuales aritméticas y lógicas
  - De acceso a memoria: para transferir información desde y hacia memoria
  - De transferencia de datos: para enviar y recibir información de los dispositivos de entrada/salida
  - De control: las instrucciones que alteran el flujo de control del programa en ejecución

# Tipos de instrucciones

- Nótese que no todas las instrucciones hacen referencia al mismo número de argumentos:
  - Por caso, la instrucción **ADD** en una arquitectura registro a registro debe hacer referencia a tres argumentos, dos de entrada y uno de salida
  - En contraste, la instrucción **NEG** en una arquitectura de 1-address más registro sólo especifica un argumento, el cual hace las veces de argumento de entrada y también de salida

# Formato de instrucción

- Recordemos que las instrucciones a nivel de lenguaje máquina están compuestas por una secuencia de unos y ceros
- Esta secuencia debe codificar las siguientes cuestiones:
  - De qué instrucción se trata
  - El conjunto de argumentos, tanto origen como destino, sobre los cuales se ha de operar
  - Opcionalmente, una referencia a la próxima instrucción (arquitecturas de 4-addresses)

# Formato de instrucción

- Conceptualmente, una instrucción máquina se compone de los siguientes campos:
  - Un campo de opcode, de tamaño fijo o variable, el cual codifica de qué instrucción se trata
  - Cero o más campos argumento, dependiendo del tipo de instrucción codificado por el campo de opcode
- Los campos argumento describen cómo y dónde encontrar los argumentos
- Los modos de direccionamiento indican cómo se deben interpretar los campos argumento

# Diseño del set de instrucciones

- Definición del conjunto de operaciones:
  - Cuántas y cuáles operaciones considerar
  - Cuán complejas han de ser
- Tipos de datos nativos
- Codificación elegida para las instrucciones:
  - Longitud de la instrucción (variable vs. fija)
  - Cantidad de campos argumento en cada instrucción
  - Tamaño de los campos argumento en cada caso

# Características deseables

- Al diseñar el set de instrucciones máquina se aspira a satisfacer los siguientes objetivos:
  - Asegurar la completitud del set de instrucciones
  - Poder hacer uso de la totalidad de la memoria disponible
  - Que el set de instrucciones sea ortogonal con respecto a los modos de direccionamiento implementados por la arquitectura
  - Contar con instrucciones de un tamaño fijo

# Ortogonalidad

- Denominaremos a un set de instrucciones como ortogonal cuando todos los modos de direccionamiento implementados estén disponibles a todos los tipos de instrucciones
  - En otras palabras, set de instrucciones es ortogonal si el modo de direccionamiento está especificado dentro del campo argumento (por caso, el set de instrucciones de la arquitectura **PDP-11**)
  - En contraste, la practica usual es determinar el modo de direccionamiento a partir del campo de opcode (por caso, la arquitectura **OCUNS**)

# Ortogonalidad

- La ortogonalidad brinda sin duda una mayor libertad a los programadores (en particular a quienes implementen compiladores)
- No obstante, la ortogonalidad suele extender la longitud promedio de las instrucciones
  - A mayor tamaño promedio de instrucciones, hará falta un mayor ancho de banda a memoria
  - Recordemos que la tecnología de los procesadores avanza con mayor rapidez que la tecnología de la memoria principal (¡el gap se mantiene!)

# Tamaño fijo vs. variable

- Con respecto al tamaño de instrucción, surgen naturalmente dos alternativas:
  - Una posibilidad es que todas las instrucciones tengan el mismo tamaño, es decir, que el tamaño esté prefijado
  - Otra posibilidad es que cada instrucción tenga el menor tamaño posible, lo que implica que el tamaño de las instrucciones no estará prefijado, sino que será variable

# Tamaño fijo

- Ventajas:
  - Se simplifica notablemente la etapa de decodificación de la instrucción, ya que el opcode ocupa una posición conocida (y fija) dentro de la secuencia de unos y ceros que codifica la instrucción
- Desventajas:
  - El tamaño debe ser lo suficientemente grande como para codificar todas las instrucciones
  - Un formato fijo de gran tamaño requiere contar un amplio ancho de banda a memoria

# Tamaño variable

- Ventajas:
  - Cada instrucción ocupa el menor tamaño posible
  - Es posible contar con instrucciones de gran tamaño, por caso, con múltiples referencias a memoria
- Desventajas:
  - Se complica y posiblemente ralentiza la decodificación de las instrucciones
  - Esta alternativa no es del todo apropiada para arquitecturas superescalares, las cuales requieren decodificar múltiples instrucciones en cada ciclo

# El camino del medio

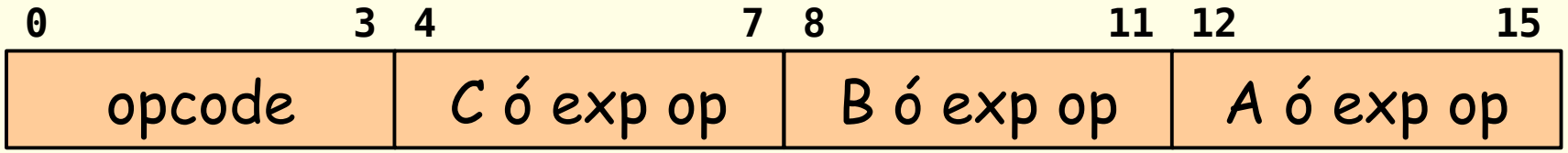
- Un cierto formato de instrucciones fijo de **k + n** bits, cuenta con **k** bits para el opcode y **n** bits para un único operando
- Este formato de instrucción permite a lo sumo $2^k$ opcodes y hacer referencia de a lo sumo $2^n$ direcciones de memoria
  - ¿Cómo elegir adecuadamente **k** y **n**?
  - Ciertamente, lo ideal sería que tanto **k** como **n** fueran variables, según el tipo de instrucción

# El camino del medio

- Otro ejemplo, supongamos que se dispone de un formato fijo de 16 bits, que debe abarcar lo siguiente:
  - 15 instrucciones de 3 operandos de 4 bits
  - 14 instrucciones de 2 operandos de 4 bits
  - 31 instrucciones de 1 operando de 4 bits
  - 16 instrucciones sin operandos
- Aparentemente quedan libres 4 bits para codificar apenas 16 de estas instrucciones...

# Expansión del opcode

- ¿Qué tal si los opcodes del **0000** (0) a **1110** (14) codificaran las operaciones de tres operandos, pero el opcode **1111** (15) se interpretara de una manera especial?
  - El opcode “reservado” **1111** podría ser interpretado como que se deben ir a buscar bits de opcode adicionales (ahora disponibles pues las restantes instrucciones tienen menor cantidad de operandos)



# Expansión del opcode

- Así siguiendo, se puede arribar por ejemplo a la siguiente codificación:

| 0 – 3 | 4 – 7 | 8 – 11 | 12 – 15 |
|---|---|---|---|
| 0000-1110 | operando C | operando B | operando A |
| 1111 | 0000-1101 | operando B | operando A |
| 1111 | 1110-1111 | 0000-1111 | operando A |
| 1111 | 1111 | 1111 | 0000-1111 |

# Tamaño variable de operandos

- En general, se requieren menos bits para codificar un registro que una dirección completa de memoria
- La expansión del opcode también permite la coexistencia de operandos de distinto tamaño
- Nótese que este problema está siendo resuelto sin comprometer el proceso de decodificación
  - La técnica de expansión de opcode sigue calificando como un formato de instrucción de tamaño fijo

# Modos de direccionamiento

- Muy pocos bits permiten codificar una gran cantidad de instrucciones que hagan uso de pequeñas constantes o de los registros del procesador
- No obstante, no se puede afirmar lo mismo al considerar el uso de operandos que residan en memoria principal
- En particular existe una cierta correlación entre el desempeño de un sistema en concreto y la cantidad de memoria principal disponible

# Modos de direccionamiento

- En internet se suele atribuir a un tal Bill Gates la siguiente frase:
  - *“640KB is enough for anyone”*
- Más allá de la veracidad o no de la atribución, el hecho que nos siga resultando simpática encierra una gran verdad
- Los diseñadores de la **PDP-11** acotaron:
  - *“el peor error que se puede cometer al diseñar una arquitectura es no contar con la suficiente cantidad de bits de direccionamiento”*

# Espacio de direccionamiento

- Un campo argumento de **n** bits posibilita hacer uso de un espacio de $2^n$ locaciones

| Arquitectura | Tam. de Palabra | Bits de address. | Espacio de Dir. |
|---|---|---|---|
| C=64 | 8 | 16 | 64KB |
| PDP-11 | 16 | 16 | 64KB |
| VAX | 32 | 32 | 4GB |
| Intel 8086 | 16 | 16/20 | 1MB |
| Intel 80286 | 16 | 16/24 | 16MB |
| Intel 80386 | 32 | 32 | 4GB |
| AMD64 | 64 | 36/40-52/48-64 | 64GB/4PB/6EB |

# Espacio de direccionamiento

- Al crecer **n**, se exploraron alternativas para minimizar la cantidad de bits ocupados por las instrucciones:
  - Evitar de ser posible las instrucciones que hagan referencia a locaciones de memoria
  - Implementar alguna forma de expansión de signo, para almacenar sólo una parte de la dirección de las locaciones de memoria referenciadas
  - Hacer uso de registros auxiliares implícitos

# ¿Preguntas?