El nivel ISA
(Instruction Set Architecture)

Posicionado entre la microarquitectura y el sistema operativo

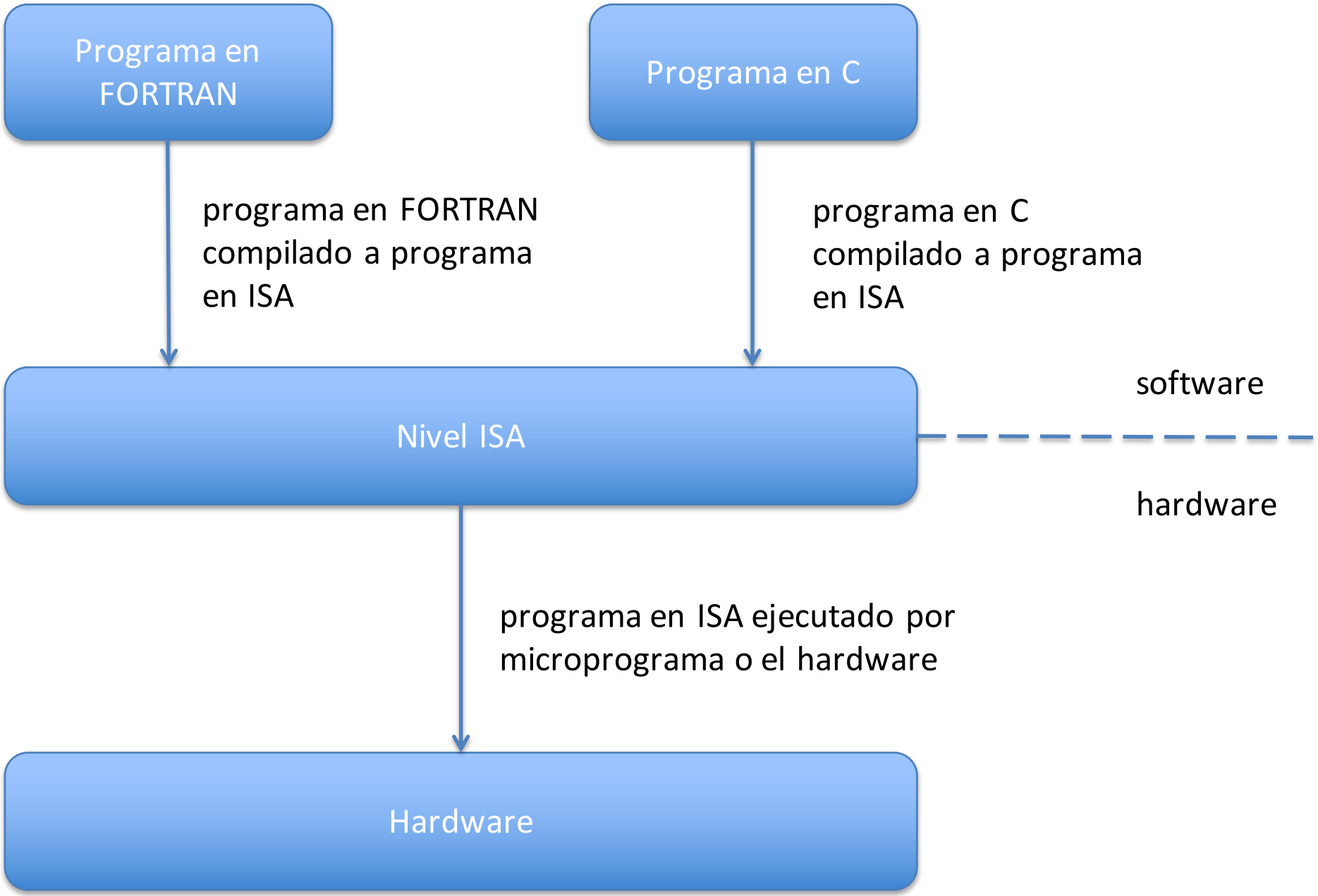
Interfaz entre el software (los compiladores) y el hardware:

- es el lenguaje que ambos tienen que entender

Se podría hacer que el hardware ejecutara directamente programas en C, C++ o Java

… pero no sería una buena idea:

- se perdería la ventaja de desempeño de la compilación sobre la interpretación
- para ser útiles, los computadores tienen que poder ejecutar programas escritos en múltiples lenguajes, y no solo uno


Programa en FORTRAN
Programa en C
programa en FORTRAN compilado a programa en ISA
programa en C compilado a programa en ISA
Nivel ISA
software
hardware
programa en ISA ejecutado por microprograma o el hardware
Hardware

Los programas, en diversos lenguajes de alto nivel, son traducidos al nivel ISA —una forma intermedia común

… y se diseña y construye el hardware que pueda ejecutar directamente programas en este nivel

Problema: Compatibilidad con versiones anteriores

Las dos características de un buen ISA:

- define un conjunto de instrucciones que puede ser implementado eficientemente con las tecnologías vigentes y futuras

  … resultando en diseños *cost-effective* (eficaces en relación a su costo) a lo largo de varias generaciones

  debe hacer felices a los diseñadores de hardware

- proporciona un objetivo sin irregularidades para el código compilado

  debe hacer felices a los diseñadores de software

Propiedades

El nivel ISA se podría definir según cómo ve a la máquina un programador de lenguaje de máquina:

- pero hoy no se programa mucho en lenguaje de máquina

Hoy lo definimos como …

el código del nivel ISA es lo que produce un compilador:

- el programador del compilador tiene que saber cuál es el modelo de memoria,

  … qué registros hay,

  … qué tipos de datos están disponibles

  … qué instrucciones están disponibles, etc.

Hay aspectos que no son parte del nivel ISA

… porque no son visibles para el programador del compilador:

- si la microarquitectura es microprogramada o no
- si es “pipelined” o no
- si es superescalar o no
- (… aunque algunas de estas propiedades afectan el desempeño, que sí es visible para el programador)

Modelos de memoria

Todos los computadores dividen la memoria en celdas, normalmente de 8 bits (**byte**), que tienen direcciones consecutivas:

- los caracteres ASCII son de 7 bits + bit de paridad
- otros códigos usan múltiplos de 8 bits para representar caracteres

Los bytes son agrupados en **palabras** de 4 u 8 bytes

… y hay instrucciones para manejar palabras enteras

Muchas arquitecturas requieren que las palabras estén alineadas según su límite natural —**alineación**:

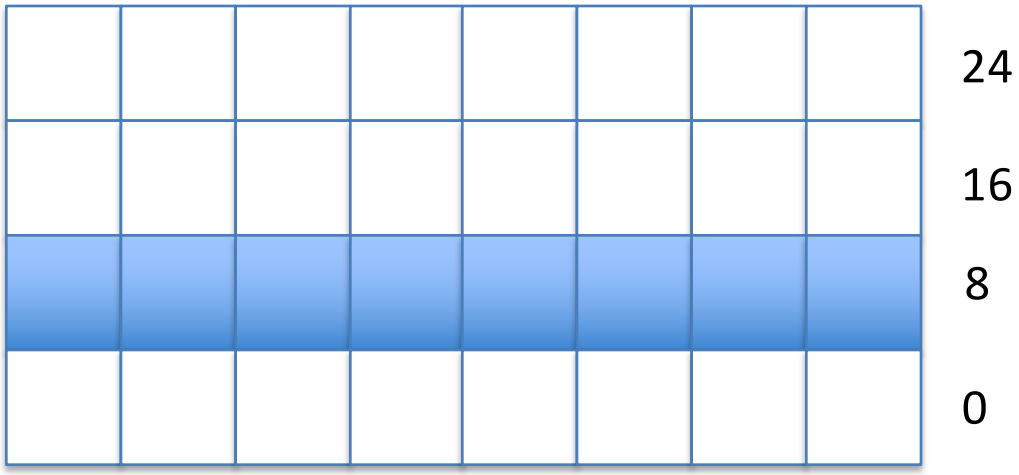
- p.ej., las de 4 bytes comienzan en las direcciones 0, 4, 8, etc.
- las memorias operan más eficientemente

  … p.ej., el Core i7 lee 8 bytes a la vez,

  … pero la interfaz de memoria requiere direcciones que sean múltiplos de 8

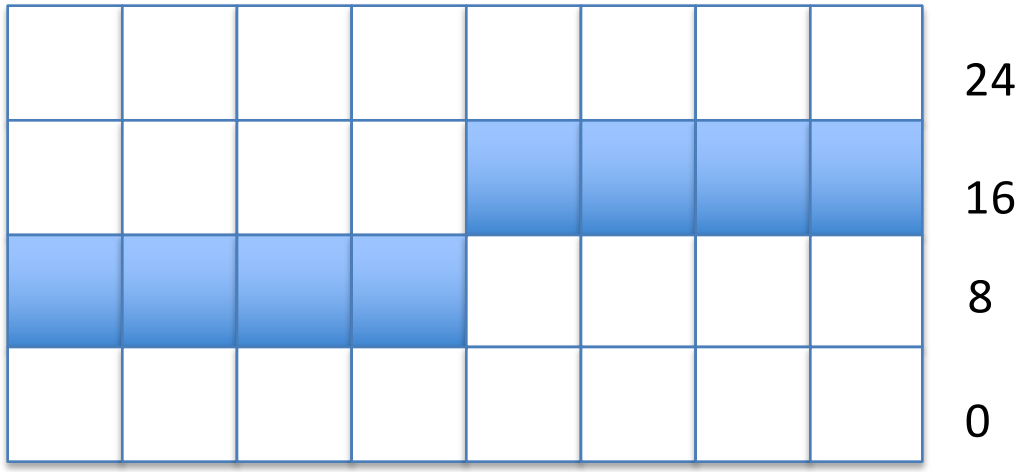
… aunque a veces esto causa problemas:

- en el Core i7, los programas pueden hacer referencia a palabras que empiezan en cualquier dirección (ya que en el 8088 el bus de datos era de 1 byte)



palabra alineada de 8 bytes que empieza en la dirección 8: un acceso a memoria

palabra no alineada de 8 bytes que empieza en la dirección 12: dos accesos a memoria y luego armar la palabra



Espacio de direcciones único: 0 a $2^{32} - 1$, 0 a $2^{64} - 1$ bytes

… aunque algunas máquinas tienen espacios de direcciones separados para instrucciones y datos:

- más complejo
- permite tener 2^{32} bytes de programa y otros 2^{32} bytes de datos usando solo direcciones de 32 bits
- es imposible que un programa se modifique accidentalmente a sí mismo —todas las escrituras van automáticamente al espacio para datos
- "malware" no puede cambiar un programa, ni siquiera puede direccionarlo

Los registros

Registros visibles al nivel ISA:

- controlan la ejecución del programa
- almacenan resultados temporales

Los hay de propósito especial:

- “program counter”, “stack pointer”

… y de propósito general:

- variables locales clave, resultados intermedios

Los registros proporcionan acceso rápido a datos muy usados (evitando accesos a memoria)

Algunas veces, algunos registros de propósito general son un poco especiales:

- p.ej., EDX en el Core i7 es un registro general, pero que también recibe la mitad del producto en una multiplicación y almacena la mitad del dividendo en una división

Incluso cuando los registros generales son intercambiables,

… el sistema operativo o los compiladores pueden usarlos de manera especial:

- p.ej., algunos registros pueden almacenar parámetros para una función, otros pueden ser usados como almacenamiento temporal

Hay registros de propósito especial disponibles solo para el sistema operativo —ni los compiladores ni los usuarios tienen que conocerlos:

- controlan los caches, la memoria, los dispositivos de I/O, y otras características del hardware

PSW (*program status word*) es un registro de control que almacena bits de varios tipos requeridos por la CPU

… entre ellos, los códigos de condición:

- N, Z, V, C, A, P
- son usados por las instrucciones de comparación y de salto condicional

Los 16 registros del Core i7

EAX, **EBX**, **ECX** y **EDX**: 32 bits, más o menos generales:

- EAX es el principal registro aritmético
- EBX almacena punteros (direcciones de memoria)
- ECX se usa en los "loops"
- EDX se usa en multiplicaciones y divisiones, para almacenar productos y dividendos de 64 bits junto a EAX
- cada uno contiene un registro de 16 bits (AX, BX, CX y DX) e incluso uno de 8 bits (AL, BL, CL y DL)

ESI, **EDI**, **EBP** y **ESP**: 32 bits, más o menos generales:

- ESI y EDI almacenan punteros para manejo de strings
- EBP típicamente apunta a la base del registro de activación vigente
- ESP es el "stack pointer"

CS, SS, DS, ES, FS y GS: 16 bits, pueden ser ignorados al usar un único espacio de direcciones de 32 bits (recuerdos del 8088)

EIP: 32 bits, "program counter"

EFLAGS: 32 bits, PSW

Las instrucciones

El componente principal del nivel ISA es el conjunto de instrucciones de máquina:

- controlan lo que la máquina puede hacer
- siempre hay instrucciones LOAD y STORE para mover datos entre la memoria y los registros

  … e instrucciones MOVE para copiar datos entre registros
- siempre hay instrucciones arirméticas y lógicas

  … e instrucciones para comparar datos y “saltar” a una determinada instrucción dependiendo de los resultados

IA-32, de Intel, como está implementada en el Core i7

Soporte completo para ejecutar programas escritos para los procesadores 8086 y 8088 (que tenían el mismo ISA):

- el 8086 y el 8088 eran máquinas de 16 bits (solo que el 8088 tenía un bus de 8 bits)

  … seguidos por el 80286 (16 bits)

Todos los procesadores que siguieron (el 80486, la familia Pentium, Core 2 duo, y hasta el Core i7) **tienen esencialmente la misma arquitectura de 32 bits** —llamada IA-32:

- los cambios arquitectónicos posteriores principales han sido la adición de instrucciones especializadas para mejorar el desempeño en aplicaciones multimediales

  … y la versión de 64 bits, en realidad introducida por AMD

El Core i7 tiene tres modos de operación:

- *real*: todas las propiedades agregadas desde el 8088 son desabilitadas y se comporta como un 8088
- *virtual 8086*: el sistema operativo crea un ambiente aislado especial que se comporta como un 8088
- *protegido*: el Core i7 propiamente tal

El espacio de direcciones está organizado en 16,384 segmentos, cada uno con direcciones de 0 a $2^{32} - 1$

… aunque la mayoría de los sistemas operativos (UNIX y Windows incluidos) manejan solo un segmento:

- la mayoría de las aplicaciones ve un espacio de direcciones lineal de 2^{32} bytes

Cada byte tiene su propia dirección

… y las palabras tienen 32 bits en formato “little endian”:

- el byte de más a la derecha (low-order) tiene la menor dirección

Los registros del Core i7 aparecen en la diap. #16

Tipos de datos

Un tema clave es si hay soporte de hardware para un tipo de datos en particular:

- si hay instrucciones que esperan datos en un formato particular

  … entonces el usuario no es libre de elegir otro formato

- p.ej., los números enteros con signo exigen que el signo sea el bit más significativo

Pueden ser divididos en dos categorías:

- numéricos
- no numéricos

El principal tipo de datos numérico son los **números enteros**:

- 8, 16, 32 y 64 bits
- sin signo, con signo
- todas las combinaciones están en el Core i7

… también se usan **números de punto flotante**:

- 32, 64 y 128 bits
- muchos computadores tienen registros separados para operandos enteros y para operandos de punto flotante
- el Core i7 implementa el estándar IEEE 754, en 32 y 64 bits

Algunos lenguajes de programación —p.ej., COBOL— manejan números decimales:

- dos dígitos decimales por byte, de 4 bits cada uno (¿es eficiente?)

Los computadores manejan mucha información no numérica:

- e-mail, Web, fotos digitales, multimedia

P.ej., los **caracteres**:

- ASCII, de 7 bits (más 1 bit de paridad)
- Unicode, de 16 bits —codificación universal de los alfabetos de la mayoría de los lenguajes humanos; usado por Java

El nivel ISA suele tener instrucciones especiales para *strings*:

- copia, búsqueda, edición, etc.

P.ej., los **valores Boolean**:

- 0 es falso; todo lo demás, verdadero
- bastaría con un bit, pero se usa un byte o una palabra (los bits no tienen dirección propia)
- … excepto cuando hay un arreglo de valores Boolean, o “bit-map” (p.ej., para seguirle la pista a los bloques del disco)

P.ej., los **punteros**, es decir, las direcciones de memoria:

- *stack pointer* y *program counter* (o *instruction pointer*) son punteros
- tener acceso a una variable a una distancia fija de un puntero es muy común en todos los computadores
- útiles, pero también son la fuente de muchos errores de programación con graves consecuencias —hay que usarlos con mucho cuidado

El formato de las instrucciones

Una instrucción consiste en un *opcode* —qué hace la instrucción

… y, normalmente, información adicional:

- p.ej., de dónde vienen los operandos o a dónde va el resultado (una, dos o tres direcciones —*direccionamiento*)

Todas las instrucciones podrían ser del mismo largo:

- más simple y facilita la decodificación, pero desperdicia espacio

En general, el largo de las instrucciones varía:

- pueden ser del mismo largo de una palabra, de la mitad o de un cuarto del largo de una palabra (en cuyo caso caben dos o cuatro instrucciones en una palabra), o pueden ocupar dos palabras

Direccionamiento

La mayoría de las instrucciones tienen operandos

Direccionamiento (*addressing*), o cómo especificar dónde están los operandos

Cómo se interpretan los bits de una dirección para encontrar el operando —los **modos de dirección**

Direccionamiento **inmediato**

La parte de la instrucción en que se especifica una dirección contiene al operando propiamente tal (en lugar de su dirección):

- el operando es automáticamente traido de la memoria al mismo tiempo que la instrucción

  … por lo que está **inmediatamente** disponible para ser usado

  … y no requiere una referencia adicional a la memoria
- p.ej., MOV R1, #4 carga el registro R1 con el valor 4
- ¿desventajas?

Direccionamiento **directo**

Se especifica explícitamente la dirección de memoria del operando:

- la instrucción siempre va a tener acceso a la misma dirección de memoria —puede cambiar el valor almacenado en la dirección, pero no la dirección
- solo para tener acceso a variables globales cuya dirección es conocida al momento de compilar
- ¿es útil?

Direccionamiento **por registro** (*modo registro*)

Conceptualmente igual a direccionamiento directo,

… pero especifica explícitamente un registro en lugar de una ubicación de memoria:

- modo de direccionamiento más común
- las variables que van a ser usadas más a menudo (p.ej., el índicde de un *loop*) van en los registros

En la arquitectura ARM, casi todas las instrucciones usan únicamente este modo:

- el "casi" es debido a las instrucciones que llevan un dato de la memoria a un registro (LDR), o viceversa (STR)