Instituto Tecnológico de Costa Rica

San José

Compiladores e Intérpretes

Apuntes de la clase del día viernes 17 de febrero del 2017

Apuntador:
Kevin Lobo Chinchilla (2015088135)

Profesor:
Dr. Francisco Torres Rojas

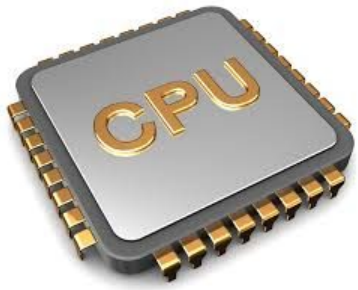
I Semestre 2017

Tabla de Contenido

Sistemas Operativos y Máquinas Virtuales

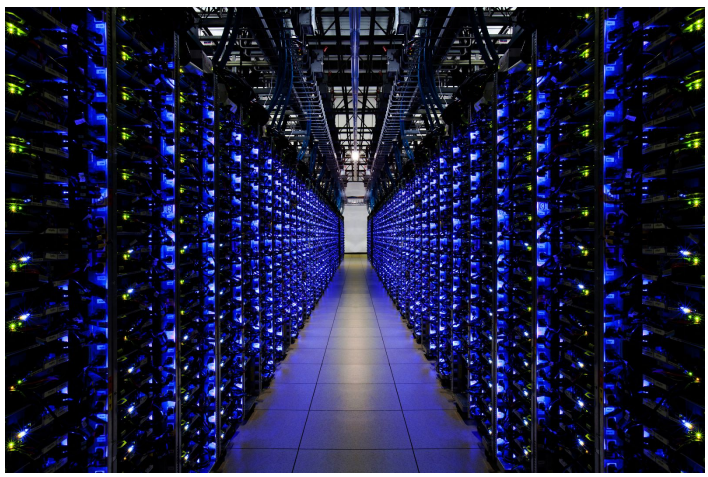
Definiciones

- **Real:** Se ve y existe. Por ejemplo, los circuitos del CPU.



- **Virtual:** Se ve, pero no existe. Por ejemplo, el Sistema Operativo.

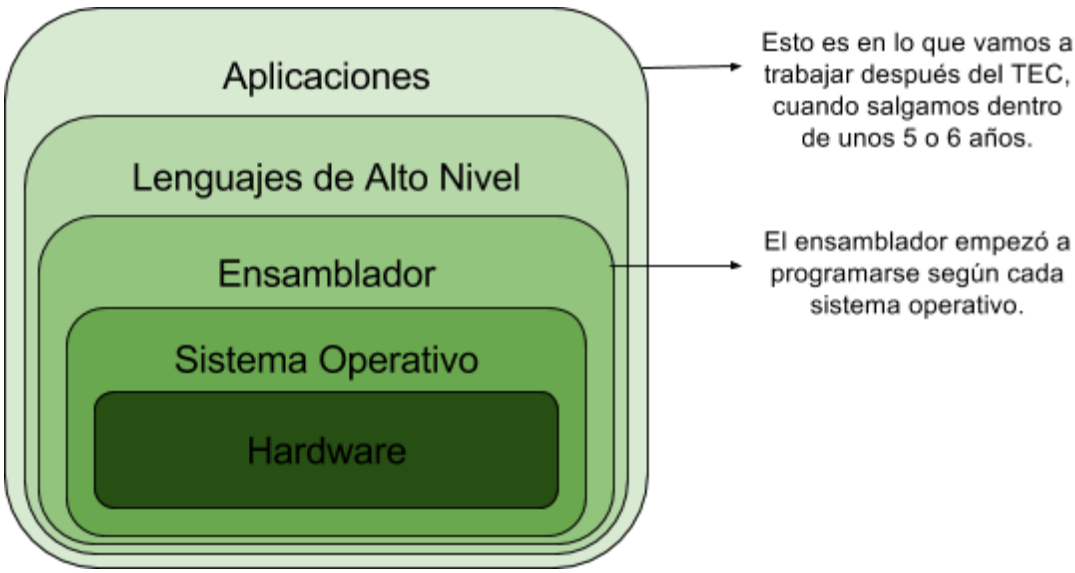
- **Transparente:** No se ve, pero existe. Por ejemplo, los servidores cuando utilizamos servicios en la nube.



- **Máquina:** Dispositivo (real, virtual o transparente) que realiza una tarea computacional. Por ejemplo: servidores, las computadoras que corren en algunos autos, sistemas empotrados.

Sistema Operativo como Máquina Virtual

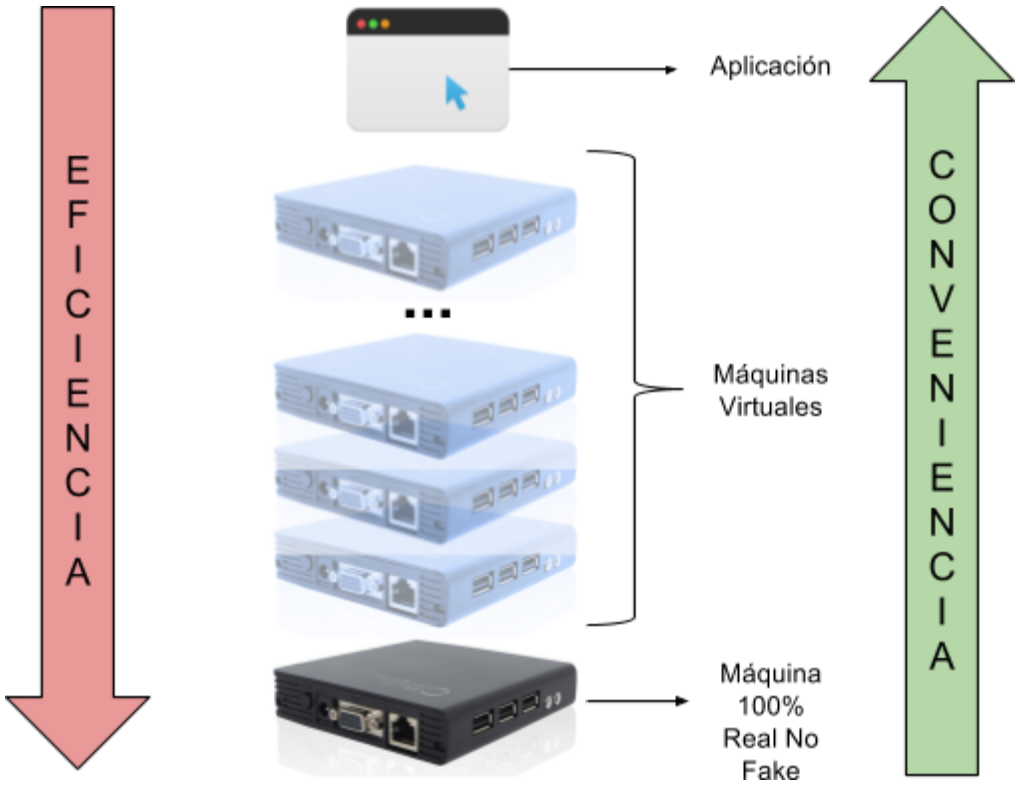
- Se puede decir que el sistema operativo es una máquina que vemos, pero no existe.
- La idea de utilizar un sistema operativo fue tan exitosa que las capas se redefinen:



- El SO secuestra nuestro hardware, lo hace suyo. A cambio nos brinda conveniencia y simpleza.

Máquinas Multiniveles

El concepto de máquina virtual se generaliza y surgen las máquinas multiniveles.



Las aplicaciones también son máquinas virtuales. Por ejemplo, en un supermercado el cajero que nos atiende ve la computadora que utiliza como una caja registradora, se abstrae mucho más la computadora real. La caja como tal en realidad no existe, es virtual. Nuestro trabajo será, por lo tanto, crear máquinas virtuales.

Máquinas y Lenguajes

Definiciones I

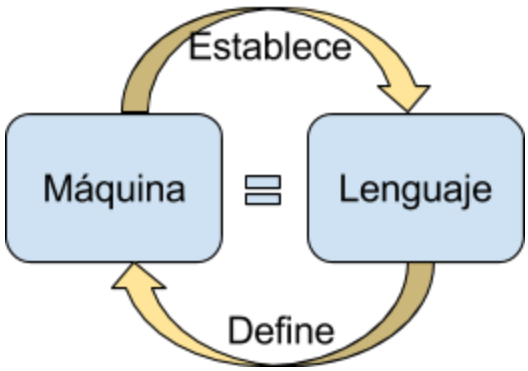
> **Programa:** Texto formado por símbolos que describe y prescribe el comportamiento de una máquina.

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```

La máquina “entiende” directamente lo que indica el programa que debe ejecutar.

> **Lenguaje de Programación:** Lenguaje apropiado para escribir programas, un sistema de notación que permite describir cómputos.

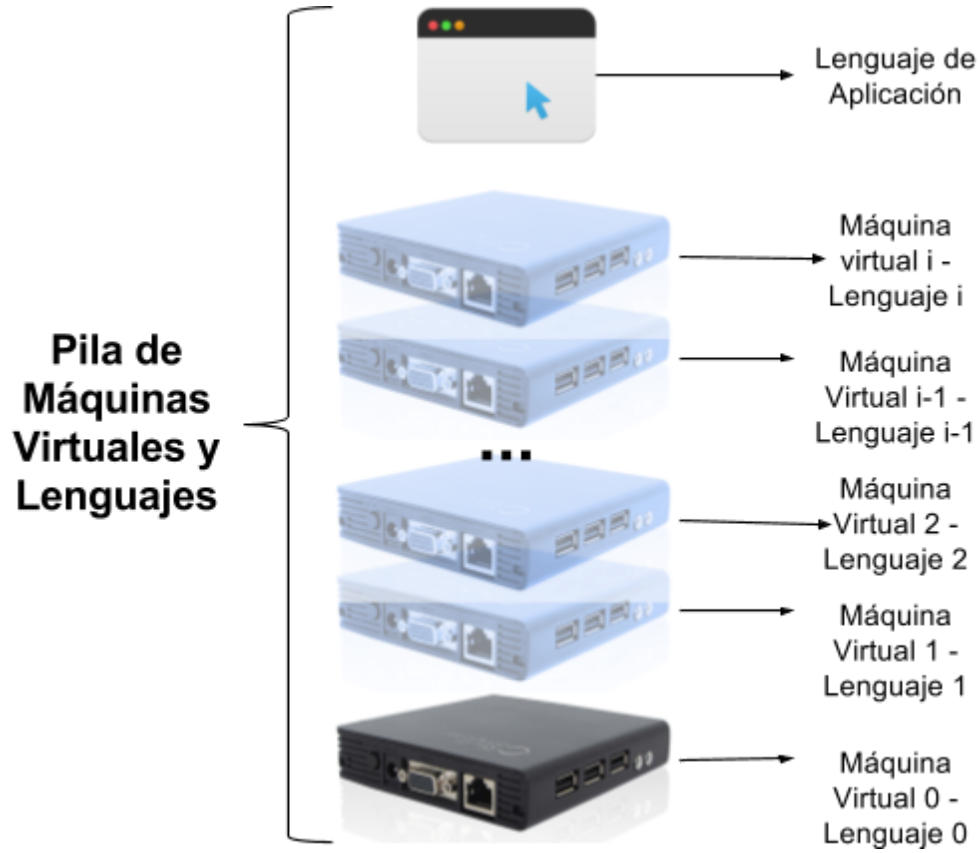
Cada máquina tiene un lenguaje que “entiende” directamente. El lenguaje define las capacidades de esta máquina.



Existe una equivalencia entre máquina y lenguaje.

¿Qué es crear una máquina virtual?

- Diseñar el lenguaje que la define.
- Lograr que el lenguaje sea entendido por otra máquina virtual o real.



El lenguaje de aplicación es cómo usar la aplicación. Por ejemplo, si no sabemos usar Waze, no conocemos lo suficiente el lenguaje de aplicación de Waze.

Así mismo, cuando utilizamos Netflix, buscamos en su catálogo y reproducimos una película o serie, nos estamos comunicando mediante el lenguaje de aplicación de Netflix.

Creación de Máquina Virtual

- Como se aprecia en la imagen anterior, cada máquina virtual se crea sobre la anterior. M_i -> M_{i-1}
- M_i supone que M_{i-1} es real.

Problema de Lenguajes

Dado que existe un lenguaje diferente para cada máquina virtual, surgen las siguientes dudas:

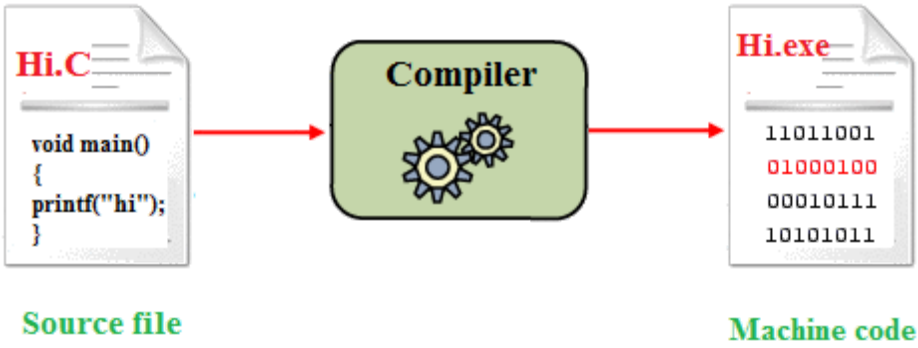
- ¿Cómo lograr su comunicación?
- ¿Cómo hacerlos equivalentes?

Por lo que surge el mapeo de lenguajes, que se lleva a cabo mediante:

- Traducción
    - Convertir completamente el programa en L_i a un programa en L_{i-1}.
    - Hay que esperar a que la traducción termine.
    - Se realiza una sola vez.
    - Rápido en ejecutar.
- Interpretación:
    - Tomar cada instrucción que la máquina M_i quiera ejecutar.
    - Simular la ejecución de esa instrucción, ejecutando código en M_{i-1}.
    - No hay que esperar la traducción.
    - Lento en ejecutar. Se debe interpretar de nuevo cada vez que se ejecuta.

Definiciones II

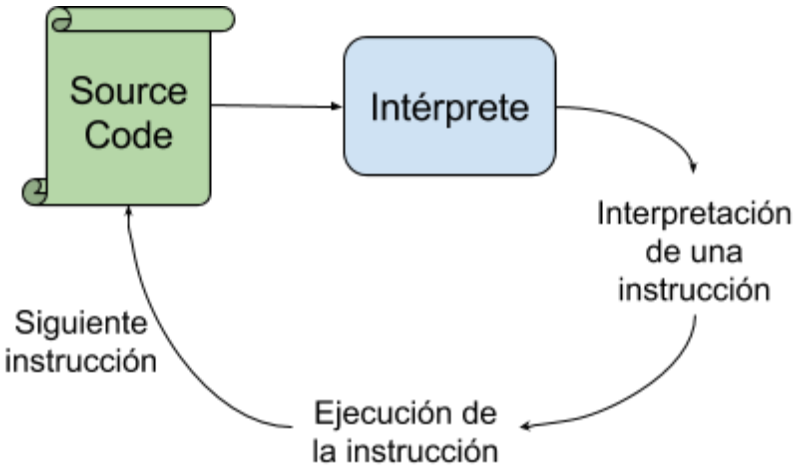
Compilador: Software que **traduce** un programa escrito en lenguaje de una máquina virtual a un programa equivalente escrito en el lenguaje de otra máquina.



Source file

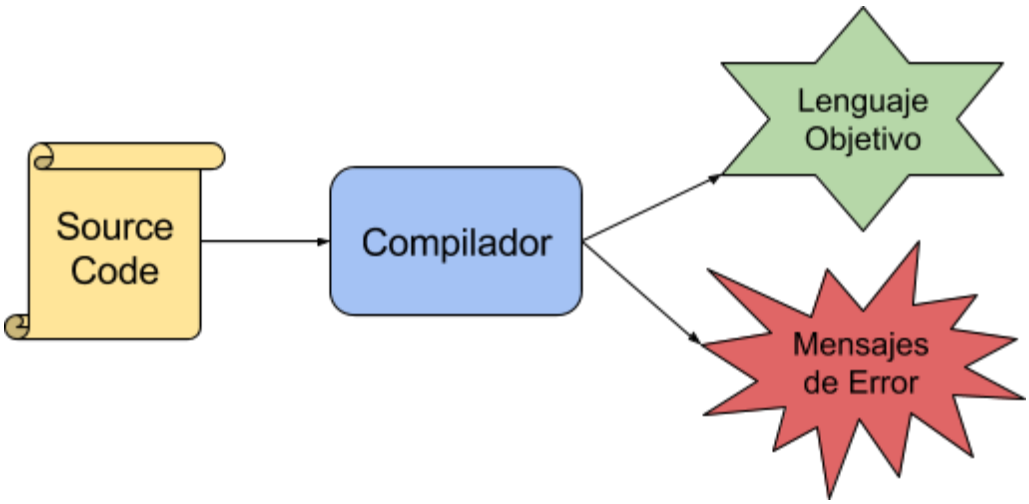
Machine code

Intérprete: Software que **ejecuta** un programa escrito en el lenguaje de una Máquina Virtual corriendo una simulación de dicha máquina sobre otra Máquina Virtual.



Computadora: Intérprete implementado en hardware.

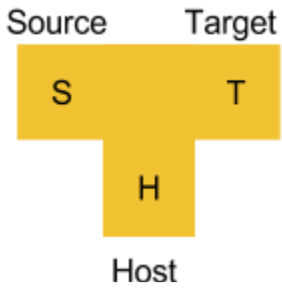
Diagramas T



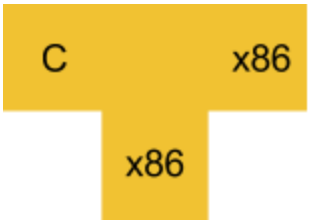
Hay 3 lenguajes asociados a un compilador:

- **Lenguaje Fuente (Source):** Entrada que debe ser traducida. Por ejemplo: C, COBOL, C++, Java.
- **Lenguaje Objeto (Target):** Salida hacia el que se traduce. Por ejemplo: Ensamblador, C.
- **Lenguaje de Implementación (Host):** Es el lenguaje en el que está escrito el compilador.

Estos lenguajes puede representarse gráficamente mediante una T



Ejemplo 1: GCC



El compilador de este ejemplo toma un programa escrito en C, escupe un programa en lenguaje máquina y está escrito en lenguaje máquina.

Este diagrama T corresponde al de GCC.

¿Qué es GCC?

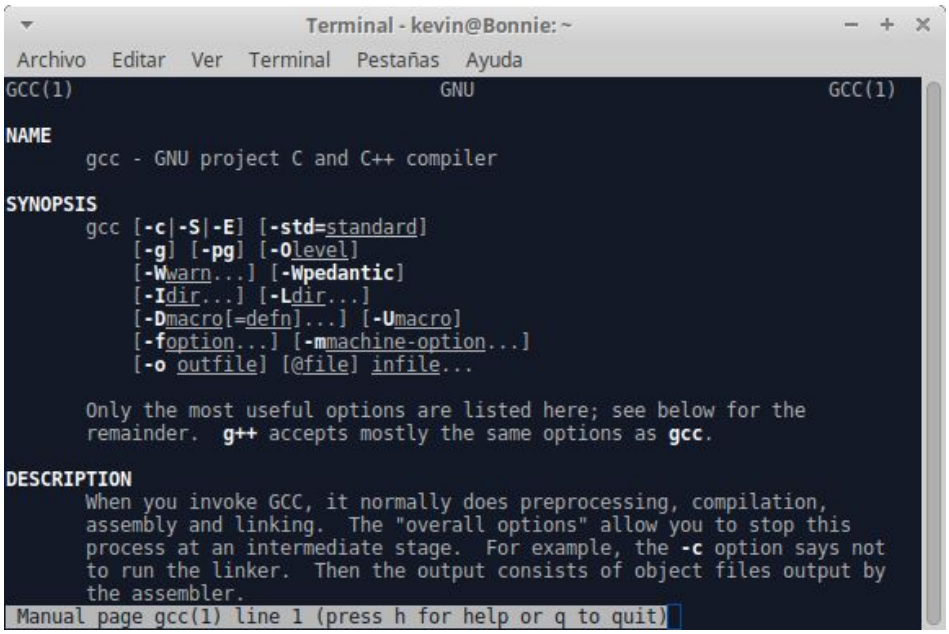
El GCC es el GNU Compiler Collection, es decir, un sistema de compiladores que soporta varios lenguajes de programación.
Fue escrito por el loco pero no idiota Richard Stallman, quien ama el software libre hasta extremos terroristas y tiene un buen español. Además, se asustó con el cronómetro del profesor.

```c
#include<stdio.h>
#include <stdlib.h>

int addsub(int op1, int op2, char what, unsigned short *flags);
unsigned short flags;

int main(int argc,char *argv[]) {

        int op1, op2;
        char operand;

        if(argc!=4){
                printf("Parameter Error\n");
                return -1;
        }

        op1 = atoi (argv[1]);
        op2 = atoi (argv[3]);
        operand = argv[2][0];

        printf("%d\n",(unsigned short)addsub(op1,op2,operand,&flags));
        printf("%d\n",(signed short)addsub(op1,op2,operand,&flags));
        printf("%d\n",flags);

        return 0;
}
```

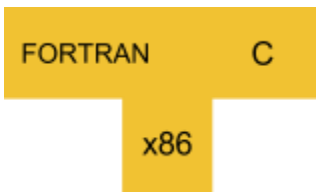
GCC

```asm
section .text
global addsub

addsub: push    ebp
        mov     ebp, esp

        mov     al, [ebp+16]
        cmp     al, '-'
        je      dif
        cmp     al, '+'
        je      sum

        jmp     ende

dif:    mov     ax, [ebp+8]
        mov     cx, [ebp+12]
        sub     ax, cx
        jmp     ende

sum:    mov     ax, [ebp+8]
        mov     cx, [ebp+12]
        add     ax, cx
ende:
        push    eax
        mov     eax, 1
        mov     [ebp+20], eax
        pop     eax
        pop     ebp
        ret

section .data
```

Man de GCC



Ejemplo 2: F2C



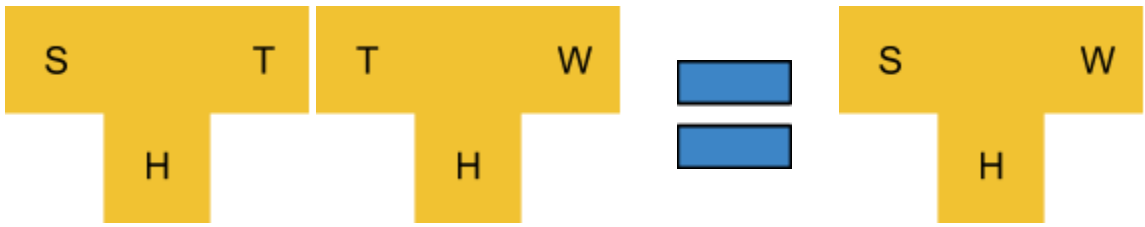
Este compilador recibe como entrada código FORTRAN, brinda como salida código C y está escrito en lenguaje máquina.
Se llama F2C (*Forchan…* FORTRAN to C), recibe código de FORTRAN 77 y genera código C o C++.

¿Para qué genera C? (Because I Can)

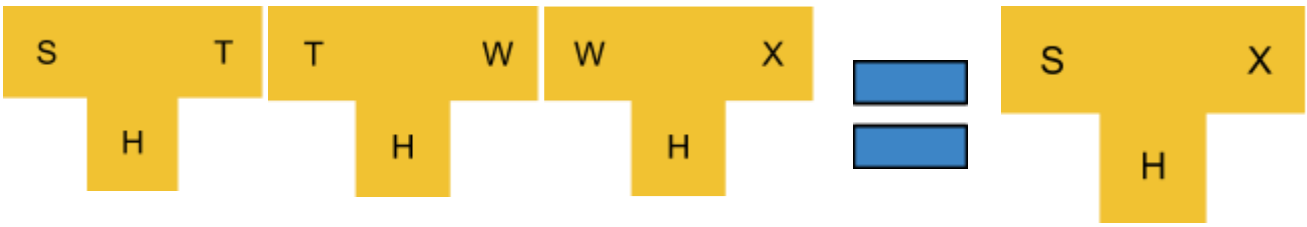
Tranquilo compañero Forchan, ya casi respondo a esa pregunta.

Operaciones con Diagramas T

Los diagramas T se pueden combinar para representar procesos de traducción más complejos.

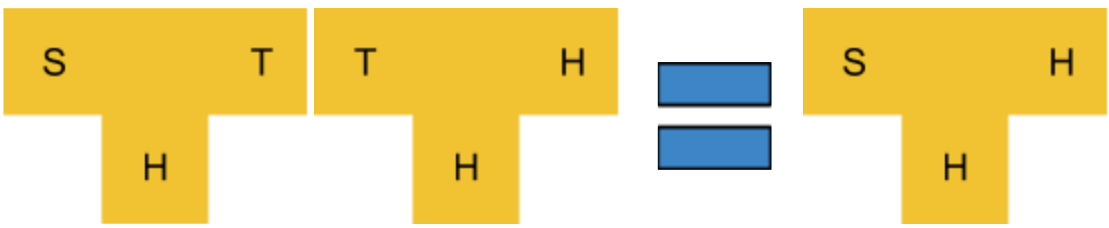


En este caso, tenemos un compilador que recibe S, escupe T y está escrito en H. Además, se tiene otro que recibe T, escupe W y está escrito en H. Esto nos facilita el proceso de traducción de un lenguaje a otro, y nos ahorra tiempo. *Ahora sí Forchan, aquí está el porqué F2C compila FORTRAN a C en lugar de x86 directamente.*



1. Compilador que recibe S, libera T y está escrito en H.
2. Compilador que recibe T, libera W y está escrito en H.
3. Compilador que recibe W, libera X y está escrito en H.
4. Esta combinación da como resultado un compilador que recibe S, escupe X y está escrito en H.

¿Y qué pasa si H es diferente? No sean necios, ya casi les contesto.



1. Compilador que recibe S, libera T y se ejecuta en H.
2. Compilador que recibe T, libera H y se ejecuta en H.
3. Obtenemos un compilador que recibe S, libera H y se ejecuta en H.

Volviendo al ejemplo de F2C, obtenemos lo siguiente mediante operaciones con diagramas:

Algo similar hacen algunas empresas para hacer conversiones de programas escritos en un lenguaje a otro lenguaje. Por ejemplo cuando se intenta convertir un programa escrito en COBOL a Java, o escrito en RPG *(Sí, sí hay algo mucho peor que COBOL)*.

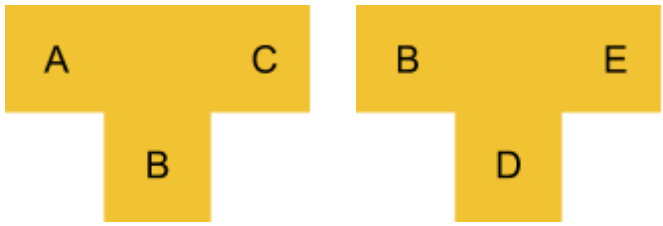
Lenguaje de Implementación

Ahora sí, ¿Qué pasa si el host es diferente? Bueno, el host puede encadenarse con otro traductor.

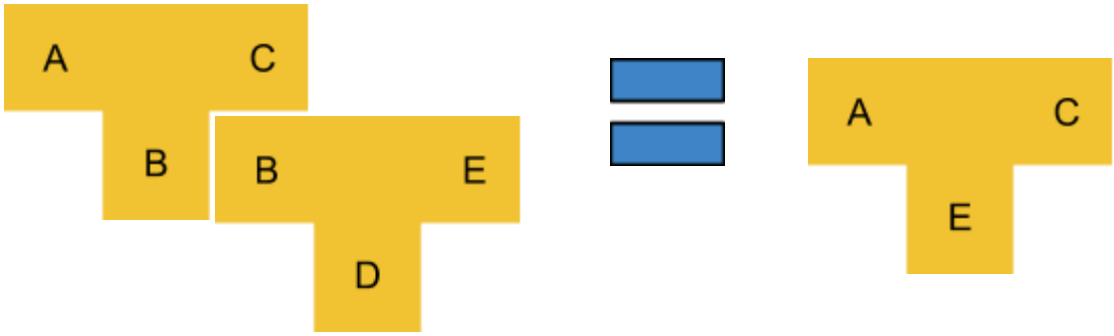
Mind = Blown

Veamos ejemplos de esto.

Tenemos los siguientes compiladores

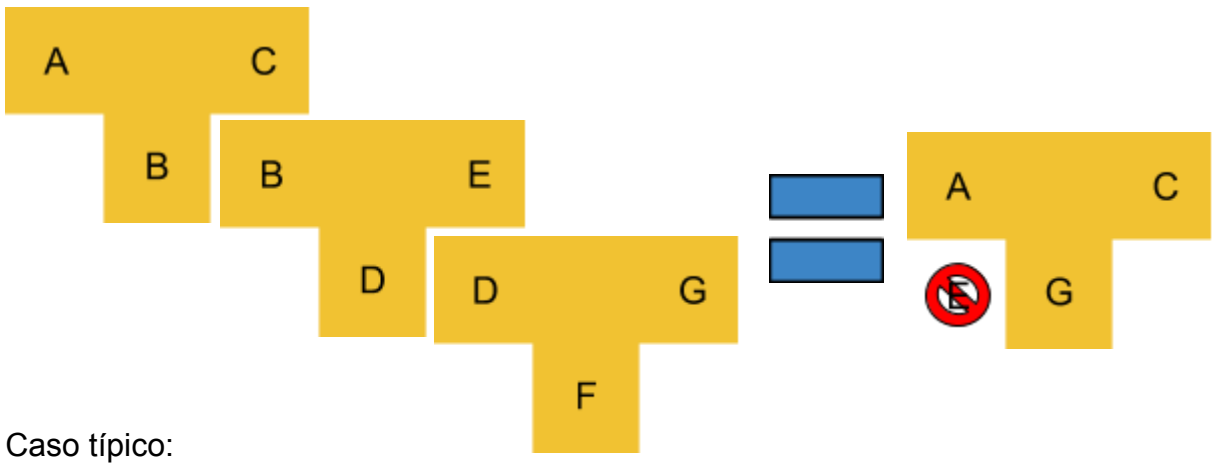


¿Qué podemos hacer con ellos?

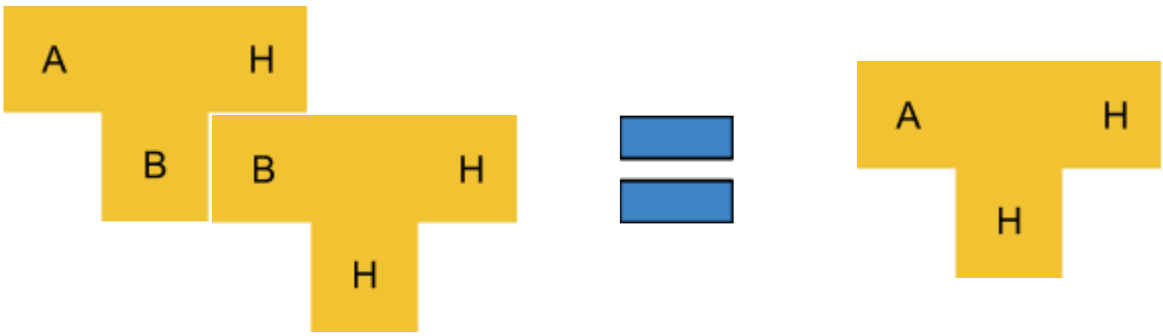


1. Compilador que recibe código de A, obtenemos a partir de él código en C, y corre en B.
2. Compilador que recibe código de B, libera E y está escrito en D.
3. El segundo compilador se utiliza para compilar el lenguaje sobre el que está escrito el primer compilador.
4. Obtenemos un compilador que recibe como entrada código escrito en A, genera como salida código en C, y corre en E.

Además, podemos utilizar varios niveles.

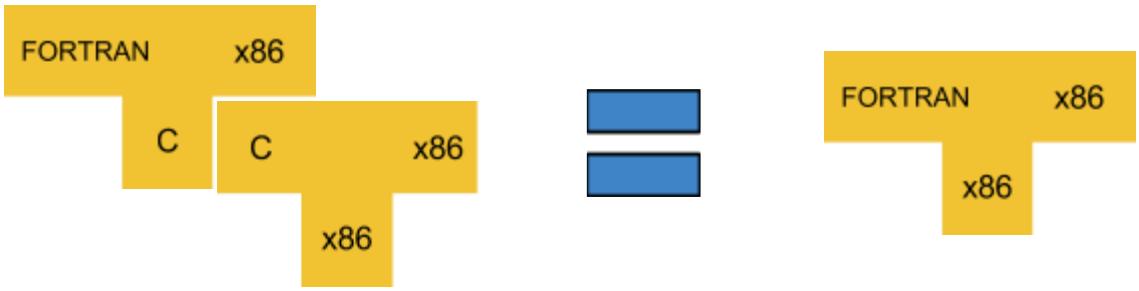


Caso típico:



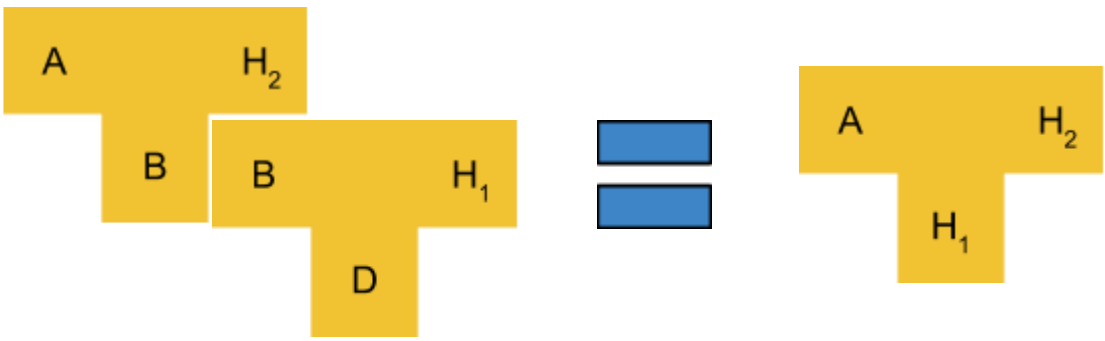
Como se puede observar mediante los diagramas, primeramente tenemos un compilador que toma como entrada un código en A, produce como salida código en C, y está escrito en B. Ahora bien, tenemos otro compilador que toma B, genera H y está escrito en H. Esto nos da la posibilidad de crear un compilador que genere H a partir de A y que a su vez esté escrito en H.

Veamos esto mediante un ejemplo en el que participa FORTRAN, C y Lenguaje Máquina.



Lenguaje Objeto

En ocasiones el lenguaje objeto (target) de un compilador podría ser diferente al de la máquina en la que corre el compilador (host).



Se puede observar que hay compiladores escritos en una máquina con lenguaje H_1 y que genera código para otra máquina con lenguaje H_2. Una vez corra en H_2, nos deja de interesar el código original de dicho programa, ya que lo que realmente importa es el funcionamiento de este en la máquina destino.

A este tipo de compilación, en el que utilizamos una máquina para compilar código que correrá en otra máquina distinta, se le llama **Cross Compiler**.

Cross Compiler

- Situación donde el código objeto generado por un compilador no es el mismo que el lenguaje máquina de la máquina donde corre.
- Máquina 1 (host) ≠ Máquina 2 (Target)
- Genera código que se ejecutará en sistemas empotrados: Aires acondicionados, microondas, lavadoras, sistemas de audio, calculadoras, entre otros.
- Se necesita cuando:
    - La máquina 1 todavía no existe, es decir, el código para la máquina 1 inexistente podría probarse en simuladores.
    - Los recursos de la máquina 1 son limitados.

El cross compiler se utiliza mucho para generar programas que ejecutan en la arquitectura ARM. Esta arquitectura es más simple que la x86 y se utiliza en muchos dispositivos móviles en la actualidad.

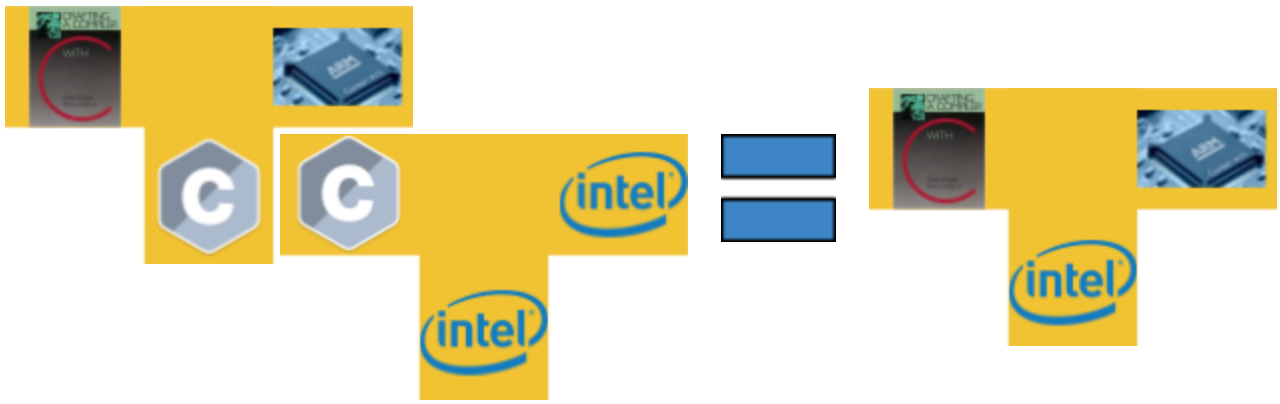
Entre las finalidades de usar cross compiler es no gastar recursos de procesadores no muy potentes en la compilación, y además nos permite compilar para procesadores que aún no existen y ejecutar sus futuros programas en simuladores. Esta técnica es muy utilizada por Intel.

Proyecto 1

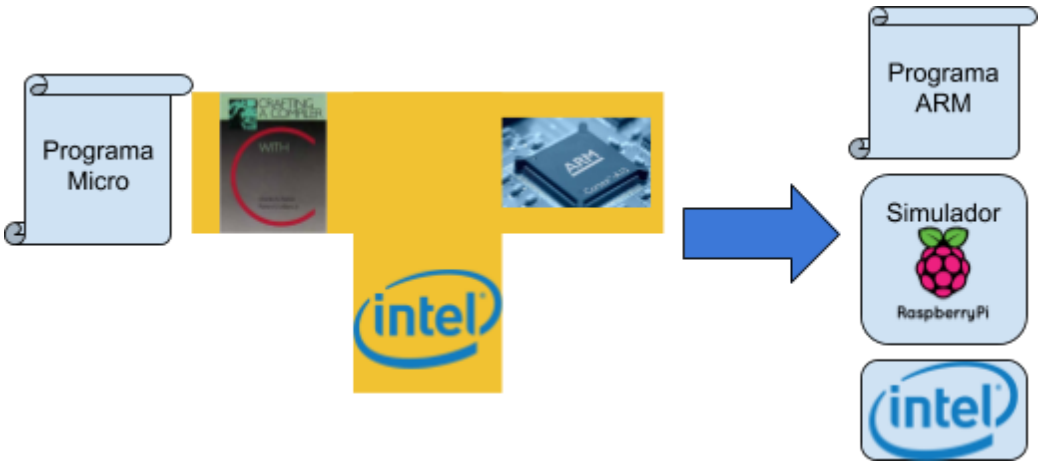
En el proyecto 1 de este curso, técnicamente lo que haremos es un cross compiler. El compilador en cuestión tiene las siguientes características:

- Recibe código Micro.
- Genera código ARM, que se ejecutará en un simulador de Raspberry PI cuya arquitectura es ARM.
- Está escrito en C.

Debemos compilar el compilador con GCC, en una arquitectura Intel o AMD, por lo que el host será x86. Podemos representar nuestro proyecto mediante los siguientes diagramas T:



El proyecto funcionará de la siguiente manera:

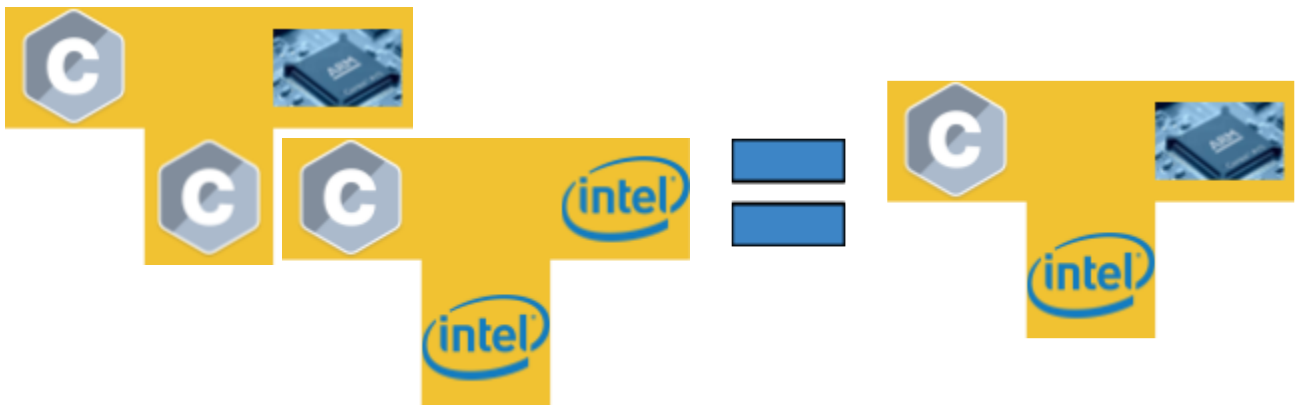


Arquitectura ARM

- Es una familia de conjuntos de arquitectura de lenguajes.
- Creada en 1980 por ARM Holdings. Esta empresa solo se encarga del diseño de la arquitectura y no de su fabricación.
- ARM Holdings vende los derechos a otras empresas para que fabriquen los procesadores.
- Es una arquitectura RISC (Reduced Instruction Set Computing).
- Muy eficiente en el uso de energía y recursos.
- Se utiliza en celulares, tabletas, laptops, sistemas empotrados.
- Existen modelos de 32 y 64 bits.
- Es la arquitectura más usada del mundo.
- ARM significaba Acorn RISC Machine. Actualmente significa Advanced RISC Machine.
- Según reportes de ARM del tercer trimestre fiscal del 2016 (Q3 2016), más de 95 mil millones de chips basados en ARM se han vendido.
- Según la página web de ARM, más del 95% de los teléfonos inteligentes utilizan esta arquitectura.

Compilador para ARM en x86

Generalmente el código de los programas que se ejecutan en ARM, son escritos y compilados en una arquitectura distinta, como x86. Esto es un claro ejemplo de Cross Compiler. Este compilador se incluye en las distribuciones de linux.



Lenguaje de Implementación

Es muy corriente escribir un compilador en el mismo lenguaje de implementación (Bootstraping). Veamos un ejemplo utilizando versiones diferentes de GCC.

GCC Timeline

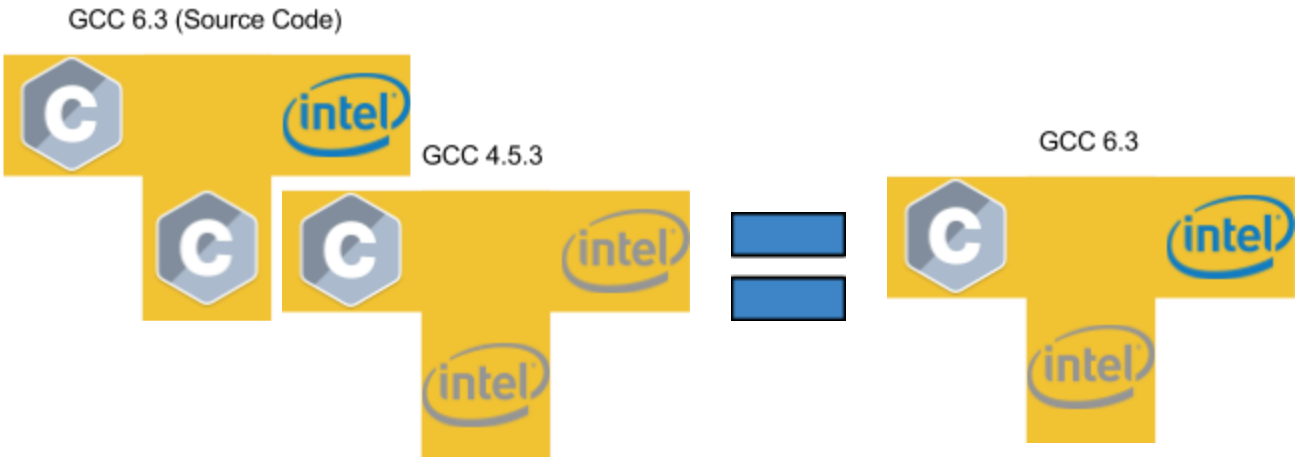
The table is sorted by date. Note that starting with version 3.3.4, we provide bug releases for older rele of stability.

Please refer to our development plan for future releases, and an alternative view of the release history.

Release	Release date
GCC 6.3	December 21, 2016
GCC 6.2	August 22, 2016
GCC 4.9.4	August 3, 2016
GCC 5.4	June 3, 2016
GCC 6.1	April 27, 2016
GCC 5.3	December 4, 2015
GCC 5.2	July 16, 2015
GCC 4.9.3	June 26, 2015
GCC 4.8.5	June 23, 2015
GCC 5.1	April 22, 2015
GCC 4.8.4	December 19, 2014
GCC 4.9.2	October 30, 2014
GCC 4.9.1	July 16, 2014
GCC 4.7.4	June 12, 2014
GCC 4.8.3	May 22, 2014

Parte del Timeline de GCC actualizado

Supongamos que descargamos el ejecutable de la versión 4.5.3 de GCC del 28 de abril del 2011 (baja calidad) y también la versión 6.3 del 21 de diciembre del 2016 pero en forma de código fuente (alta calidad). Ahora procedemos a compilar el código fuente de la versión 6.3 con el compilador de la versión 4.5.3:

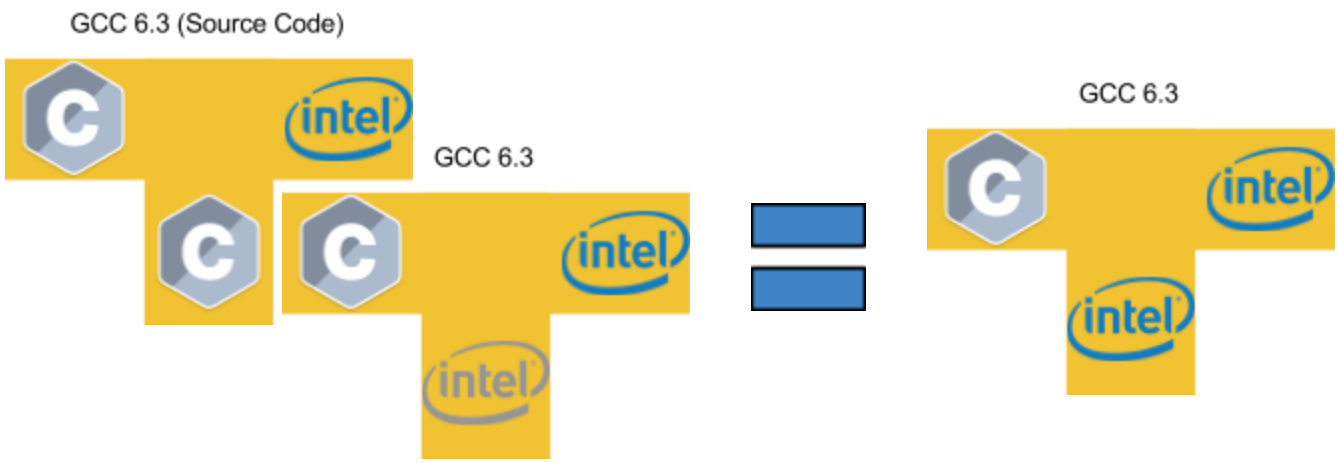


¿Qué acaba de suceder?

1. Se tiene GCC 6.3 que recibe lenguaje C, escupe x86 de alta calidad, pero está escrito en C.
2. Se tiene GCC 4.5.3 que recibe lenguaje C, escupe x86 de baja calidad dado que es una versión vieja, y está escrito en x86 de baja calidad también.
3. Utilizamos el ejecutable de GCC 4.5.3 para compilar el código fuente de GCC 6.3, por lo que obtenemos un GCC que recibe C, escupe x86 de alta calidad, y ahora está escrito en x86 pero de baja calidad.

¿Cómo logramos que GCC 6.3 esté escrito en alta calidad?

Volvemos a compilar el código fuente de GCC 6.3, pero esta vez utilizamos el mismo GCC 6.3.



Recordatorios

- La entrega del proyecto es el 1ero de marzo. Si gustan, pueden compilar el compilador a ARM, para correr el compilador en el simulador, luego compilar MICRO en el simulador y ejecutar el programa (gracias Izcar por la idea).
- Todos dijimos “Yes” y nos dejaron de tarea resumir el capítulo 3 (casi el 4 también pero el profesor tuvo piedad) del libro *The Language Instinct* de Steven Pinker. Recordar que se debe hacer en LaTeX y se entrega el miércoles 1ro de marzo (Sí, el mismo día de la entrega del proyecto). Si se hace en inglés hay puntos extra.
- Recordar la tarea en grupos que se encuentra en el foro http://www.ic-itcr.ac.cr/index.php/foro/ic5701sj-g40-1s2017/1757-tarea-en-grupos. Es extremadamente larga (palabras de nuestro profesor) y se debe entregar el 8 de marzo (Sí, justo una semana después de entregar el primer proyecto). Se debe realizar en LaTeX.
- Quiz el miércoles 22 de febrero.
- El viernes 24 no hay clases.
- Richard Stallman está loco, pero no es idiota.
- Tarjetas perforadas.
- Todo es un lenguaje.