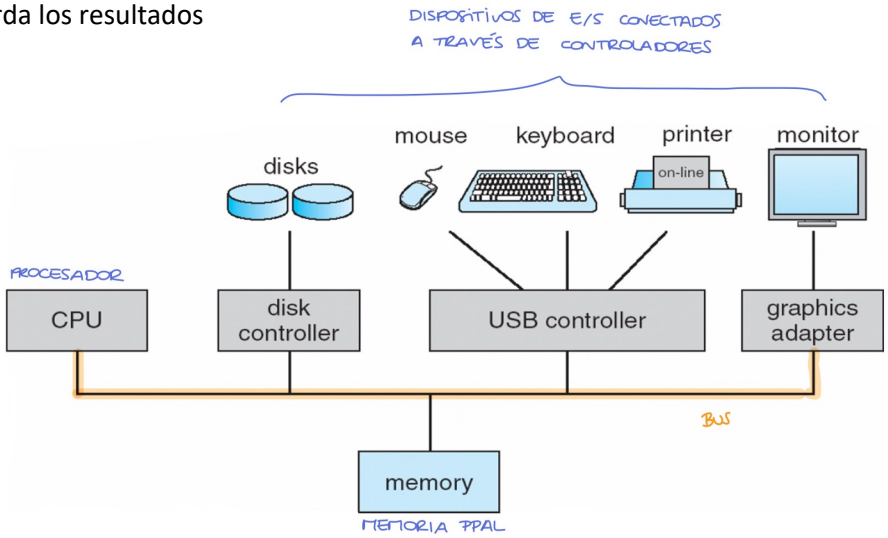
3. PROGRAMACIÓN

3.1. ARQUITECTURA DE COMPUTADOR DE VON NEUMANN

Para ejecutar un programa, los datos e instrucciones tienen que estar en memoria.

Proceso de ejecución:

1) El procesador va a la memoria y se trae una instrucción
2) La decodifica
3) La ejecuta
4) Guarda los resultados



3.2. LENGUAJES DE PROGRAMACIÓN

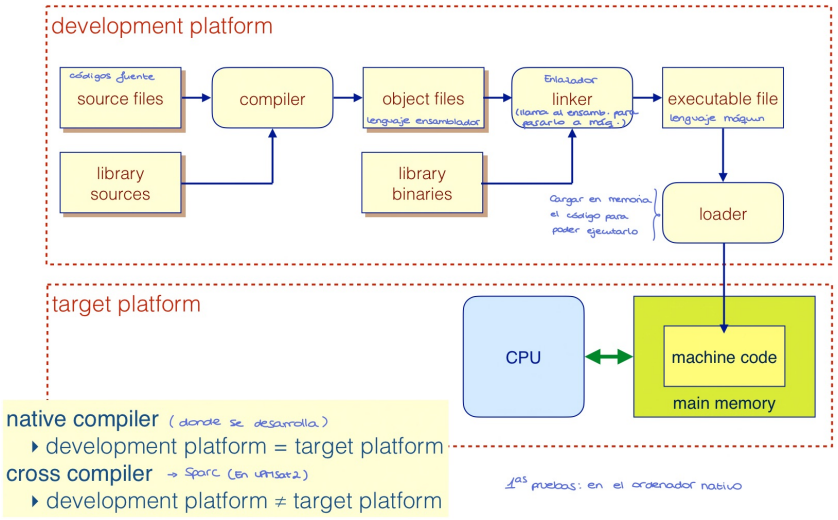
- **Lenguajes de alta abstracción o de alto nivel** (C, Ada, Python): Son adecuados para resolver el problema. Permiten productividad y portabilidad. Estos códigos son independientes del procesador. Se desarrollaron para facilitar la elaboración de programas, dado que el código binario o ensamblador son complicados de leer y entender.
- **Lenguaje ensamblador.** Representación textual de las instrucciones de máquina. Relación 1 a 1 entre lenguaje de ensamblador y lenguaje del código. Al compilar, se traduce el lenguaje de alto nivel en un lenguaje ensamblador.
- **Lenguaje máquina** (32 o 64 bits). Dígitos binarios, que contienen los datos e instrucciones codificados de forma que sean interpretables por el ordenador. Al ejecutar, se pasa de lenguaje ensamblador a lenguaje máquina, y éste se traduce para un procesador completo. Un procesador diferente no podría entender el lenguaje de maquina de un procesador determinado.

3.3. PROCESO DE COMPILACIÓN

La mayoría de los programas se escriben como un conjunto de *source files* (paquetes o módulos). Se divide el problema en “pequeños trozos”.

- Los ***source files*** se **compilan**, de forma que se traducen a un conjunto de ***object files*** binarios.
- Los ***object files*** son **enlazados (linked)** para producir un ***executable file*** binario, habitualmente incluyendo *library files* pre-compilados.
- Los ***executable files*** se almacenan en disco y se cargan a la memoria principal cuando se desean ejecutar.

CADENA DE COMPILACIÓN



3.4. COMPILACIÓN CRUZADA

En un proceso de compilación cruzada, el **compilador nativo** es donde se desarrolla el código, que se hace en otro ordenador porque en el *target* no hay pantalla, ni teclado, y la potencia está limitada.

El lenguaje máquina del procesador donde se desea ejecutar (*target*) el programa es completamente al lenguaje máquina del procesador nativo. Por tanto, se utiliza un **compilador cruzado** que compila el programa para el lenguaje máquina del target platform. Algunas bibliotecas pueden cambiar (por ejemplo, hay bibliotecas para Intel y otras para Sparc).

El código fuente es independiente del computador, es el compilador el que se encarga de traducir al procesador en concreto.

3.5. ESTRUCTURA DEL PROGRAMA

Un programa se suele dividir en módulos o paquetes. Dentro de esos paquetes se tienen subprogramas, y suele haber un subprograma principal (*main*) que es el que llama el sistema operativo cuando se carga el programa y se encarga de llamar al resto de subprogramas de los otros módulos.

Hay dos niveles de abstracción: paquetes y éstos se subdividen a su vez en paquetes más pequeños. Los paquetes siguen una estructura jerárquica.

Los distintos paquetes o subprogramas son unidades de compilación. Se suelen dividir en dos partes:

- **Especificación.** Sólo contiene las mínimas definiciones que necesitan otras partes del programa. Actúa como interfaz. Explica qué hace esa determinada unidad de compilación. No hay código fuente, sólo contiene la información de qué cosas se pueden hacer.
- **Cuerpo.** Contiene todos los detalles que se necesitan para ejecutar la unidad. Contiene la información de cómo hace las cosas.

Especificación:

```
procedure Write (S : in String);
```

Cuerpo:

```
procedure Write (S : in String) is
  c : Character;
begin
  for i in S'Range loop
    c := S(i);
  end loop;
end Write;
```

3.6. PROCEDIMIENTOS Y FUNCIONES

Los subprogramas se ejecutan cuando son llamados desde otras unidades del programa.

- Un **procedimiento** es la abstracción de una acción (escritura/lectura). Se nombran con un verbo.
- Una **función** es la abstracción de un valor. Se nombra con un nombre, devuelve un valor.

Los procedimientos y funciones se dividen en:

- **Declaraciones**: definen los nombres, tipos y otras propiedades de los datos u otros elementos del programa.
- **Instrucciones**: definen las operaciones que ejecuta el programa.

Cuando una unidad de programa se ejecuta -> se elaboran las declaraciones (por ejemplo, crear las variables locales) -> las instrucciones se ejecutan.

```
procedure P (…) is
  -- declarations go here
begin
  -- instructions go here
end P;
```

3.7. PAQUETES

Un paquete es un módulo de programa donde los datos y operaciones se pueden declarar. Son el mecanismo que utiliza Ada para la encapsulación y ocultación de datos.

La **especificación** del paquete contiene su interfaz visible: declaraciones de datos, tipos de datos y subprogramas. No se incluyen cuerpos o instrucciones.

El **cuerpo** del paquete contiene los datos de la implementación. Incluye los cuerpos de los subprogramas declarados en la especificación.

Las especificaciones y el cuerpo suelen ir en diferentes *source files*: *.ads (spec), *.adb (body). De este modo, la implementación se puede cambiar sin tener que alterar las interfaces.

```
package P is
   -- declarations
   -- subprogram specs
end P;
```

```
package body P is
   -- declarations
   -- subprogram bodies
end P;
```

Compilación separada

manager.h1_interface.ads
↳ 4 proc d. interfaz

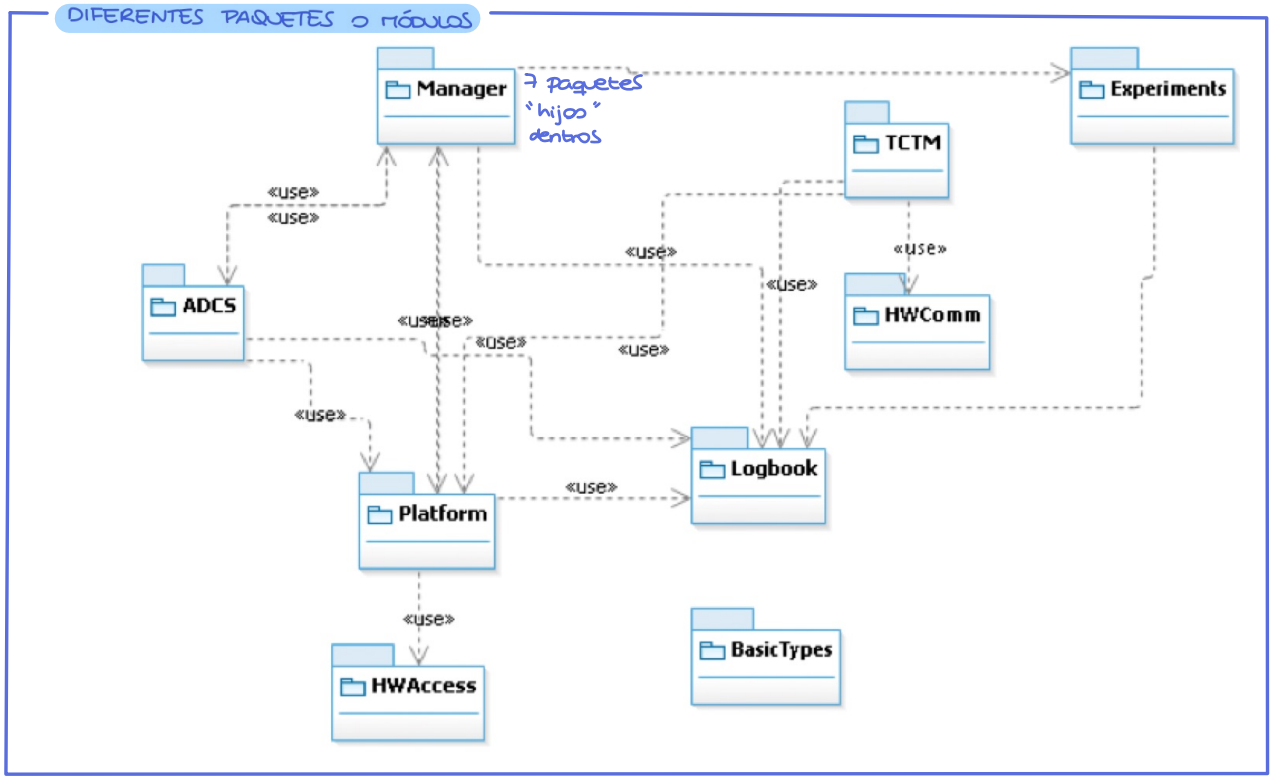
→ paquete ADCS
sin haber desarrollado todo el manager, y sólo con el interfaz, puedo:
→ Desarrollar ADCS
→ Compilar
→ NO puedo ejecutar

Unidades de compilación : Elementos más pequeños que se pueden compilar.

+ alto nivel

Paquetes → contienen → Subprogramas (funciones o procedimientos)

Se suelen dividir en 2 partes:
- Especificación
- Cuerpo



Si el ADCS se desea comunicar con el Manager, no desea conocer los su paquetes que hay dentro ni debería poder acceder a todo. Sólo debe extraer la información que necesita.

Subpaquetes de Manager.