

# Tema 3. Traducció de Programes

## Estructura de Computadors (EC)

Rubèn Tous

[rtous@ac.upc.edu](mailto:rtous@ac.upc.edu)

Computer Architecture Department  
Universitat Politècnica de Catalunya



UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH

# Índex

1 3.8 Estructura de la memòria

2 3.9 Compilació, assemblatge, enllaçat i càrrega

## 3.8 Estructura de la memòria

Variables amb mode d'emmagatzemament estàtic:

- Es guarden al mateix lloc durant tot el temps d'execució del programa.
- En C, les variables externes (o globals).
- La secció `.data` d'un programa en assemblador.
- Mida fixa.

## 3.8 Estructura de la memòria

Variables amb mode d'emmagatzemament dinàmic:

- Es guarden en una àrea de memòria de manera temporal.
- En C, les variables automàtiques (o locals).
- Les podem guardar en registres o la pila.

## 3.8 Estructura de la memòria


'variables dinàmiques':




- Aquelles on la reserva i alliberament de de memòria la fa explícitament el programador (malloc o free).
- Com que poden invocar-se en qualsevol punt del programa, aquest espai no es reserva i s'allibera de forma ordenada com succeeix amb la pila.
- Estructura de dades que anomenem **heap**.
- Gestionat pel sistema operatiu.
- malloc i free són dues funcions que fan d'interfície amb el sistema operatiu.
- malloc: retorna una adreça al fragment de memòria concedit.


## 3.8 Estructura de la memòria

Vegem els tres tipus de variables esmentats en un programa:

```

int gvec[100];  globals: dades estàtiques
int *pvec;

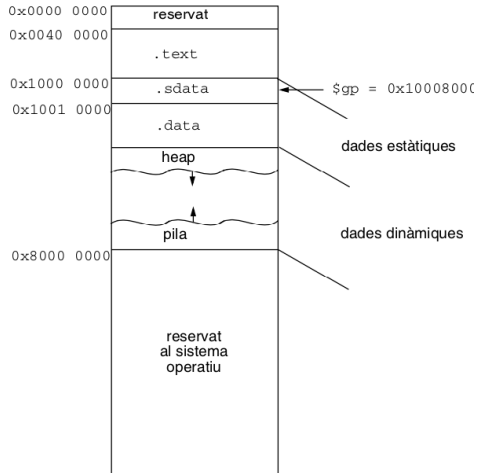
int f()
{
    int lvec[100];  local: dades dinàmiques
    ...                                     (reservem espai a la pila)
    pvec = malloc(400);  global: dades dinàmiques
    ...                                     (reservem espai al heap)
    pvec[10] = gvec[10] + lvec[10];
    ...
}  (alliberem espai a la pila)

int g()
{
    ...
    free(pvec);  (alliberem espai al heap)
    ...
}

```

## 3.8 Estructura de la memòria

Diferents àrees de memòria, en MIPS::



## 3.8 Estructura de la memòria

El 'global pointer' i .sdata:

- Per estalviar-nos fer 'la' per accedir a les dades estàtiques, podem usar un punter (global pointer) que sempre apunti a una adreça fixa de les dades estàtiques (emmagatzemat a \$gp):

```
1 lw $t0 , offset_dada($gp)
```

- Amb l'offset de 16 bits només podem abarcar una secció de 64KB, l'anomenada .sdata (small data).
- S'hi guarden variables globals d'ús freqüent de petites dimensions.
- El global pointer s'inicialitza al punt mig de .sdata, a l'adreça 0x10008000.



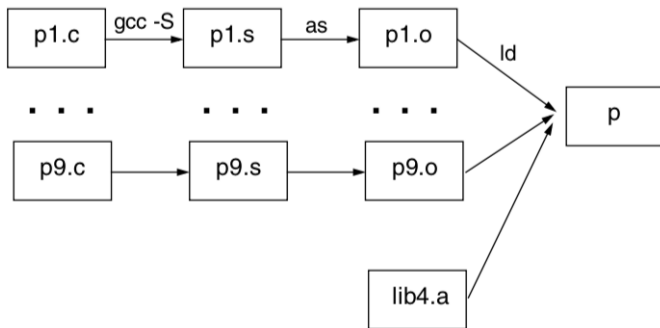
## 3.9 Compilació, assemblatge, enllaçat i càrrega

alt nivell -> baix nivell en 4 passos:

- Compilació: codi font en C -> compilador -> llenguatge ensamblador.
- Assemblatge: llenguatge ensamblador -> ensamblador -> codi màquina (fitxer objecte).
- Enllaçat: múltiples fitxers objecte -> enllaçador (linker) -> fitxer executable.
- Càrrega: fitxer executable -> carregador (loader, part del sistema operatiu) -> memòria

## 3.9.1 Compilació separada

- Es generen fitxers objecte (o de biblioteca) independents.
- L'assemblador generat fa servir etiquetes per referir-se a subrutines o dades d'altres mòduls.



## 3.9.2 Assemblatge

- Expandir les macros.
- Traduir cada instrucció al seu corresponent codi binari.
- Però no podem codificar les instruccions si no tenim les adreces!

## 3.9.2 Assemblatge

Instruccions amb adreces relatives (beq, bne):

- Primera passada de tot el codi font: taula de símbols = etiqueta <-> adreça.
- No adreces absolutes, relatives (offsets) a la secció on està definida l'etiqueta.
- Segona passada: es codifiquen les instruccions.

## 3.9.2 Assemblatge

Instruccions amb adreces absolutes (la, j, jal):

- No podem saber l'adreça absoluta ja que no estem mirant els altres mòduls.
- Es codifica provisionalment un zero.
- L'enllaçador ho corregirà (reubicació).
- Es genera una llista amb la posició de cada instrucció a reubicar, el seu tipus, i l'adreça provisional a què fa referència.
- Si són referències a altres mòduls al final del fitxer les posarem en una llista de referències externes no-resoltes.
- Per a les etiquetes declarades com a `.globl` es genera una taula de símbols globals amb les seves corresponents adreces provisionals.

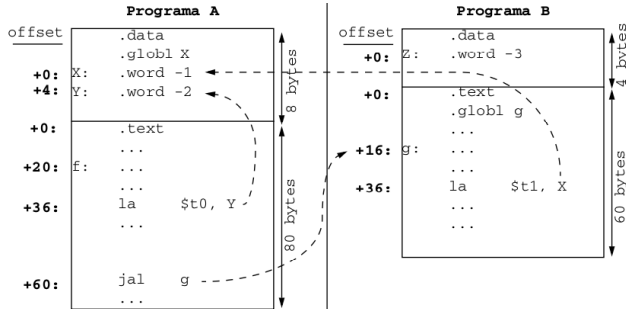
## 3.9.2 Assemblatge

En UNIX:

- Capçalera, amb informació sobre la ubicació dels restants components.
- Secció de text amb el codi màquina de les subrutines del mòdul.
- Secció de dades estàtiques, amb les dades globals definides al mòdul.
- Llista de reubicació (posició de la instrucció, tipus, i adreça provisional)
- Taula de símbols globals, i llista de referències externes no-resoltes.
- Informació de depuració (e.g. números de línia del codi font).

## 3.9.2 Assemblatge

Exemple:



Reubicar:

- posició +36 (text), tipus la, offset +4 (data)

Símbols globals:

- posició +0 (data), label="X"

Referències no-resoltes:

- posició +60 (text), tipus jal, label="g"

Reubicar:

- 

Símbols globals:

- posició +16 (text), label="g"

Referències no-resoltes:

- posició +36 (text), tipus la, label="X"

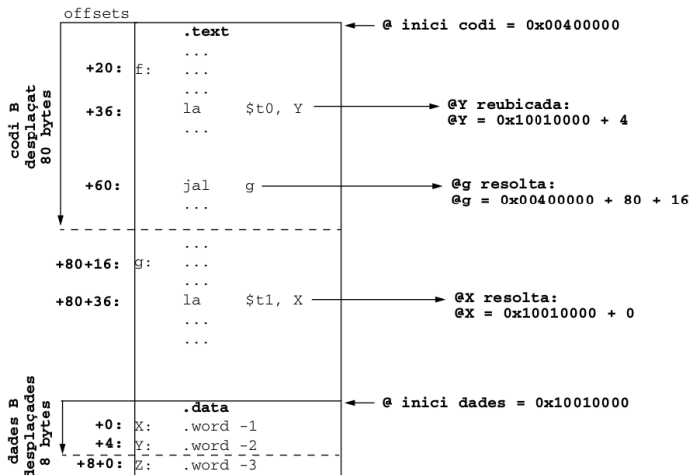
### 3.9.3 Enllaçat (o muntatge o 'linkatge')

- 1 Es busquen, en cascada, els fitxers de biblioteca necessaris (del programa o del sistema).
- 2 Es concatenen el codi i les dades dels diversos mòduls:
  - S'anota el desplaçament de cada secció.
  - S'assignen adreces definitives a les etiquetes que figuren en les taules de símbols globals, sumant-los-hi els desplaçaments de les seves corresponents seccions.
- 3 Es reubiquen les instruccions amb adreces absolutes sumant l'adreça inicial de la secció (0x00400000 per al .text o 0x10010000 per al .data, en MIPS) amb el desplaçament sofert per la secció i amb l'offset provisional dins la secció que havia assignat l'assemblador.
- 4 Es resolen les referències creuades no-resoltes, consultant les adreces definitives a la taula de símbols globals.



## 3.9.3 Enllaçat (o muntatge o 'linkatge')

Exemple:



## 3.9.4 Càrrega en memòria

- El programa resideix en un fitxer del disc.
- El 'Loader' del sistema operatiu el carrega a la memòria.
  - 1 Llegir la capçalera per determinar la mida de les seccions de codi i dades.
  - 2 Reservar memòria principal.
  - 3 Copiar les instruccions i dades del fitxer executable a la memòria (al Tema 7 veurem que no caldrà carregar tot el programa).
  - 4 Copiar a la pila els paràmetres del programa principal.
  - 5 Inicialitzar els registres, deixant \$sp apuntant al cim de la pila.
  - 6 Saltar a 'startup' (el compilador la posa dins executable), que copia els paràmetres de la pila als registres (\$a0, etc.). A continuació, la rutina startup fa una crida a 'main'.
- Quan 'main' retorna, se segueix executant 'startup', que crida a la rutina exit del sistema (allibera els recursos assignats).