Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2018-2019

# Allocazione dinamica della memoria

*Turno A*
**Nicola Basilico**
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

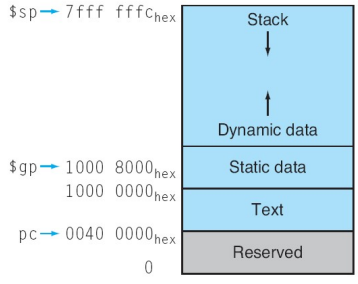
*Turno B*
**Marco Tarini**
Dipartimento di Informatica
Via Celoria 18 - 20133 Milano (MI)
Ufficio 4010
marco.tarini@unimi.it
+39 02.503.16217

1

# Utilizzo della memoria

In MIPS la memoria viene divisa in:

- **Segmento testo**: contiene le **istruzioni** del programma.
- **Segmento dati**:
  - ***dati statici***: contiene dati la cui dimensione è conosciuta a *compile time* e la cui durata coincide con quella del programma *(e.g., variabili statiche, costanti, etc.);*
  - ***dati dinamici***: contiene dati per i quali lo spazio è allocato dinamicamente a *runtime* su richiesta del programma stesso *(e.g., liste dinamiche, etc.).*
- **Stack**: contiene dati dinamici organizzati secondo una coda LIFO (Last In, First Out) *(e.g., parametri di una procedura, valori di ritorno, etc.)*.

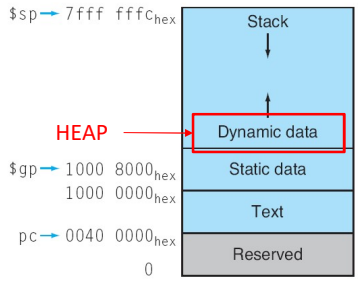



2

# Utilizzo della memoria

In MIPS la memoria viene divisa in:

- **Segmento testo**: contiene le **istruzioni** del programma.
- **Segmento dati**:
  - ***dati statici***: contiene dati la cui dimensione è conosciuta a *compile time* e la cui durata coincide con quella del programma *(e.g., variabili statiche, costanti, etc.);*
  - ***dati dinamici***: contiene dati per i quali lo spazio è allocato dinamicamente a *runtime* su richiesta del programma stesso *(e.g., liste dinamiche, etc.).*
- **Stack**: contiene dati dinamici organizzati secondo una coda LIFO (Last In, First Out) *(e.g., parametri di una procedura, valori di ritorno, etc.)*.




3

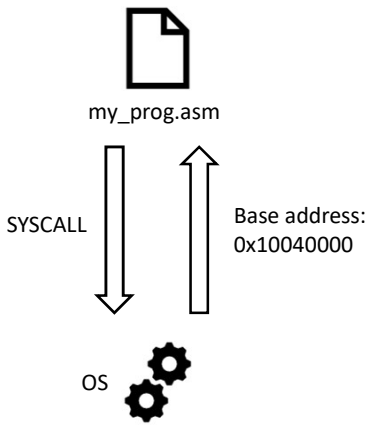
# Allocazione statica vs. allocazione dinamica

| STATICA | DINAMICA |
|---|---|
| • Allocata a **compile time** | • Allocata a **run time** |
| • Allocazione: a cura del compilatore | • Allocazione: a cura del programmatore |
| • Deallocazione: mai (solo al termine dell'esecuz) | • Deallocazione: a cura del programmatore |
| • Usata per: variabili (o costanti) *globali*, compresi array (di lughezza fissa) | • Usata per (esempi): stringhe di lunghezza non determinata, strutture dati dinamiche |

4

# Come avviene l'allocazione?

1. Il programma richiede al sistema operativo di allocare un certo numero di byte in memoria tramite una syscall
2. Il sistema operativo verifica l'effettiva disponibilità dello spazio in memoria e, se possibile, alloca lo spazio richiesto e restituisce al programma il base address dell'area di memoria allocata
3. In caso di memoria non allocabile, di norma, il sistema operativo restituirà un codice di errore (tipicamente un numero negativo) al posto del base address permettendoci di gestire il problema




*In MARS: la richiesta di allocare più spazio di quanto disponibile, genera un'eccezione a run time*

5

# SBRK (segment break)

La syscall utilizzata per allocare spazio dinamicamente sullo heap è la **SBRK** (codice **9**).

- Input: $a0 <- numero di byte da allocare
- Output: $v0 <- base address dell'area di memoria allocata

```
1  .text
2  .globl main
3
4  main:
5
6  li $v0, 9       # SBRK syscall
7  li $a0, 100     # Alloca 100 bytes in memoria
8  syscall
9  # $v0 ora contiene il base address dell'area
10 # di memoria allocata
11 li $t0, 5
12 sw $t0, 0($v0)
13 ...
```

6

# SBRK vs. malloc e free

- La syscall SBRK si occupa solo di «spostare» il puntatore alla fine dello heap più avanti aumentando lo spazio di memoria dinamica disponibile al nostro programma
- non ci permette di gestire come allochiamo la memoria all'interno dello heap, ma solo di richiedere al sistema operativo di darci più memoria
- Le funzioni «malloc» e «free» appartenenti alla libreria stdlib distribuita con libc (installata praticamente su tutti i sistemi operativi moderni) servono per gestire in maniera più fine lo heap e sono implementate «sopra» a SBRK, permettendo in maniera automatica di allocare e deallocare spazio per le nostre strutture dati dinamiche, gestendo automaticamente problemi di frammentazione della memoria e minimizzando le chiamate a SBRK utilizzando opportuni buffer di memoria
- In MARS e in SPIM non abbiamo accesso alle funzioni malloc e free, che sono disponibili solo su sistemi «reali», per cui o ne implementiamo una nostra versione oppure ne facciamo a meno

7

# Strutture Dati

Una struttura dati rappresenta un modo di **organizzare i dati** in memoria in modo da permettere l'implementazioni di **determinate operazioni** su di essi (per esempio l'accesso, la ricerca di elementi, la rimozione, di elementi, etc)

Sono state proposte molte strutture dati preposte a scopi diversi.

Una struttura dati è definita da:

- I **valori** dei dati in essa contenuti (interi, stringhe, sotto-strutture …)
- Le eventuali **relazioni** tra i dati in essa contenuti (ordine …)
- Le **operazioni** supportate
  (per es cancellazione, inserimento, ricerca …)

Una struttura dati può essere più o meno efficiente su ciascuna delle operazioni supportate. Ad esempio, due strutture dati diverse possono offrire

8

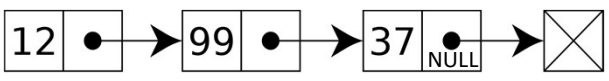
# Lista concatenata (struttura dati)

- Le liste concatenate hanno prestazioni peggiori degli array contigui
  - Meno *cache coherence*
  - Necessario storare puntatori (a "next") oltre ai dati stessi
  - Non sono Random Access (per accedere al 150esimo elemento dobbiamo scandire I primi 149 elementi)
- Tuttavia sono più flessibili
  - Diviene facile (tempo costante) rimuovere o aggiungere elementi in mezzo alla sequenza di elementi
  - E' possibile avere elementi eterogenei, ciascuno di dimensione diversa

9

# Lista concatenata (struttura dati)

- Ogni elemento contiene l'indirizzo ("next") dell'elemento successivo
- Nell'ultimo elemento, "next" è l'indirizzo invalido (adottiamo 0x00000000, detto anche NULL)
- Per passare una lista ad una funzione come argomento, si passa l'indirizzo del primo elemento
  - Tutti gli altri elementi sono raggiungibili da questo passando di "next" in "next"




10

# Esercizio: da array a lista concatenata (linked list)

- In questo esercizio, convertiamo
  un array di interi (in memoria statica)
  in una lista (in memoria dinamica)
- Per ogni elemento dell'array,
  costruiamo un elemento della lista, e lo concateniamo all'elemento precendete
- Per costruire un elemento, dobbiamo allocandolo dinamicamente (con una SBRK)
- Completiamo il nostro codice con due funzioni:
  (1) stamapre a schermo una linked list
  (2) cercare un elemento in una linked list
- Per i dettagli, vedere il codice sul sito

11