UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tema 3
Traducció de programes

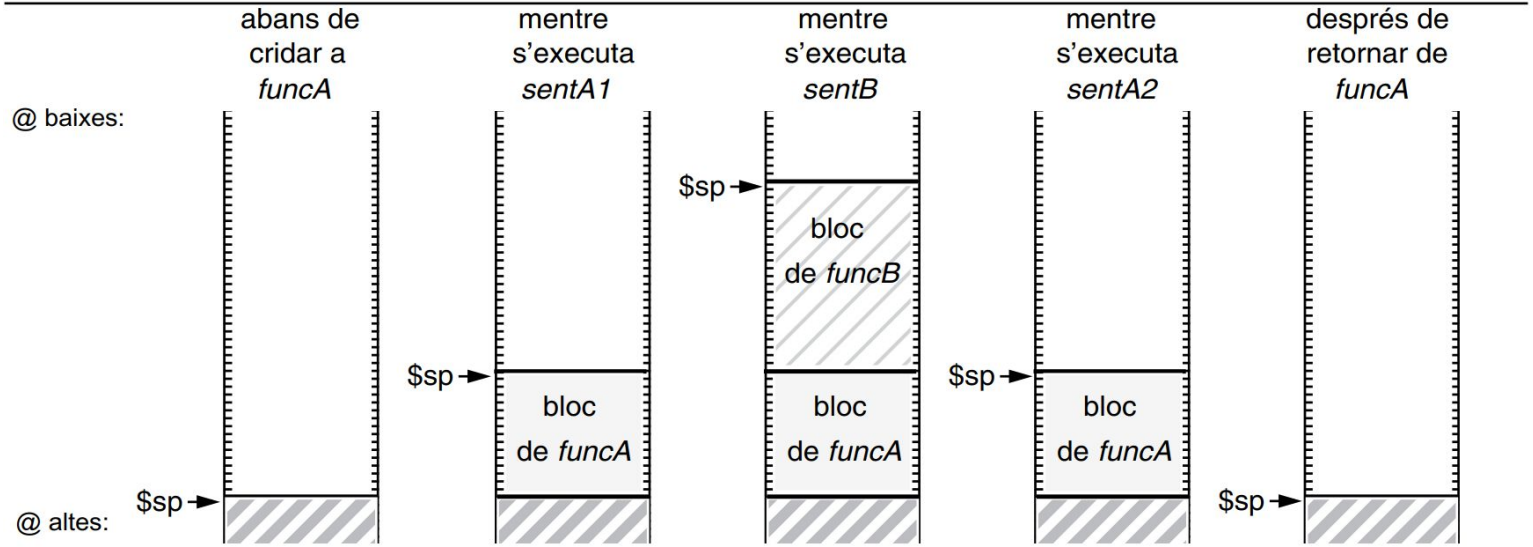
Estructura de Computadors (EC)
2023 - 2024 Q2
Adrià Armejach (adria.armejach@upc.edu)

Subrutines

La pila i els blocs d'activació

```
funcA()
{       sentA1;
        funcB();
        sentA2;
}
```

```
funcB()
{
        sentB;
}
```



Subrutines multinivell

- Subrutines que criden a altres subrutines
- Context d'una subrutina - conjunt de dades locals
  - Paràmetres ($a0 - $a3)
  - Adreça de retorn ($ra)
  - Punter de pila ($sp)
  - Càlculs intermedis (guardats en registres o a la pila)
- Problema
  - Com preservem el context de la rutina que ens ha cridat?
  - Com sabem quins registres podem modificar?

Problema: Subrutines multinivell

```
funcA:
    …
    addu    $t1, $a0, $a1
    move    $a0, $a2
    jal     funcB
    addu    $v0, $v0, $t1
    …
```

```
funcB:
    …
    li $t1, 10
    …
```

- No es preserva el context de `funcA` - `$t1`
- Com sap una subrutina quins registres no ha de modificar?

Salvar i restaurar registres

- Solució trivial i ineficient
  - Garantir que tots els registres tenen el mateix estat que tenien al invocar la subrutina
  - Obliga a guardar al bloc d'activació (pila) tots els registres que es modificaran a la subrutina
    - "Còpia de seguretat" al principi de la subrutina
    - "Restaurar registres" al final de la subrutina

Salvar i restaurar registres

- Solució de l'ABI de MIPS
  - Dividir els registres en temporals i segurs

Temporals	Segurs
`$t0 - $t9`	`$s0 - $s7`
`$v0 - $v1`	`$sp`
`$a0 - $a3`	`$ra`

Quan una subrutina acaba, ha de deixar els registres segurs en el mateix estat que tenien quan s'ha invocat

Salvar i restaurar registres

- Permet preservar el context salvant a la pila el mínim número de registres
- Requereix dos passos:
  1. Determinar els registres segurs
     - Identificar quines dades emmagatzemades en registres es generen ABANS d'una crida a subrutina i s'utilitzen DESPRÉS de la crida
  2. Salvar i restaurar els registres segurs
     - Salvar al inici de la subrutina el valor del registre segur al bloc d'activació (pila)
     - Restaurar al final de la subrutina el valor salvat del registre segur

Exemple: Salvar i restaurar registres

```
funcA:
    …
    addu    $t1, $a0, $a1
    move    $a0, $a2
    jal     funcB
    addu    $v0, $v0, $t1
    …
```

```
funcB:
    …
    li $t1, 10
    …
```

```
funcA:
    #salvar valor actual de $s0
    …
    addu    $s0, $a0, $a1
    move    $a0, $a2
    jal     funcB
    addu    $v0, $v0, $s0
    …
    #restaurar valor salvat de $s0
```

```
funcB:
    …
    li $t1, 10
    …
```

Exemple: Salvar i restaurar registres

```
funcA:
    …
    addu    $t1, $a0, $a1
    move    $a0, $a2
    jal     funcB
    addu    $v0, $v0, $t1
    …
```

```
funcB:
    …
    li $t1, 10
    …
```

```
funcA:
    #salvar valor actual de $s0
    …
    addu    $s0, $a0, $a1
    move    $a0, $a2
    jal     funcB
    addu    $v0, $v0, $s0
    …
    #restaurar valor salvat de $s0
```

```
funcB:
    #salvar valor actual de $s0
    …
    li      $t1, 10
    addu    $s0, $t1, $t1
    …
    #restaurar valor salvat de $s0
```

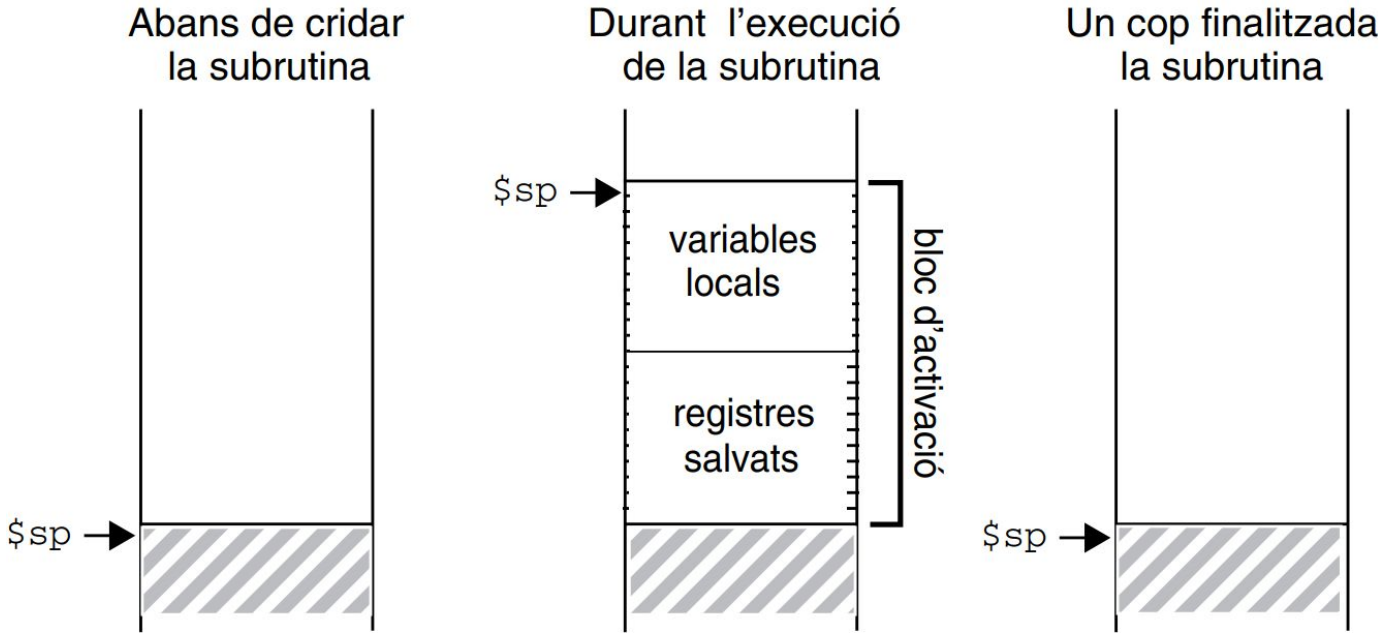
Estructura del bloc d'activació

- El bloc d'activació està ubicat a la pila i inclou la següent informació
  - Variables locals
    - De tipus estructurat (vectors, matrius, …)
    - Escalars si s'aplica l'operador unari &
  - Valors inicials dels registres segurs ("copia de seguretat")
    - Només es guarden els que es modifiquen durant la subrutina
    - S'assigna un registre segur a cada dada que ha de "sobreviure" una crida a subrutina

Estructura del bloc d'activació

- El bloc d'activació ha de respectar les següents normes
  - **Posició**
    - Variables locals al principi, seguides de els registres segurs
  - **Ordenació**
    - Les variables locals es col·loquen seguint l'ordre de declaració
  - **Alineació**
    - Les variables locals han de respectar les normes de alineament
    - Els registres segurs (words) han de anar alineats a adreces múltiples de 4
    - El tamany total del bloc d'activació ha de ser multiple de 4

Estructura del bloc d'activació



Exercici

- Analitza, dibuixa el bloc d'activació, i tradueix a MIPS la subrutina `multi`

```
int multi(int a, int b, int c) {
     int d, e;
     d = a + b;
     e = mcm(c, d);
     return c + d + e;
}
```

Exercici

- Analitza, dibuixa el bloc d'activació, i tradueix a MIPS la subrutina `exemple`

```
int f(int m, int *n);
int g(char *y, char *z);

char exemple (int a, int b ,int c) {
     int d, e, q;
     char v[18], w[20];
     d = a + b;
     e = f(d , &q) + g(v ,w);
     return v[e + q] + w[d + c];
}
```

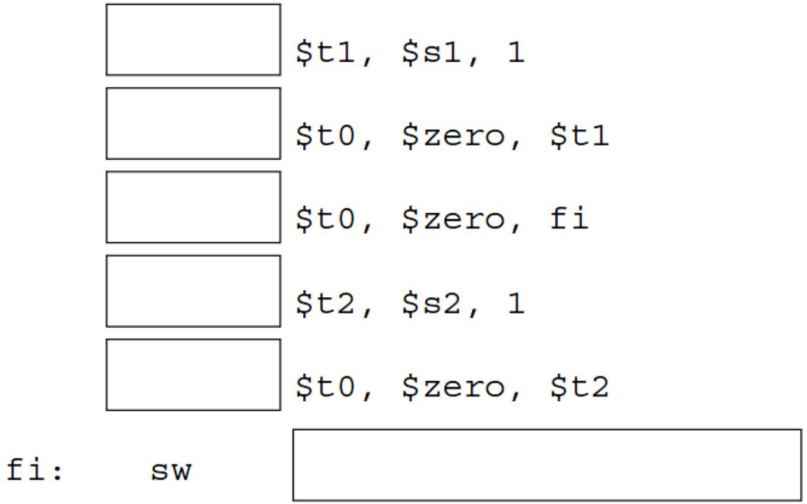
Per fer a casa....

Per fer a casa...

- Els farem a la sessió de teoria del Dijous 7 de Març

Exercici 1

- Una funció té les següents variables locals guardades a `$s0, $s1 i $s2`
  - `int *a, b, c;`
- Tradueix la següent sentencia de C a assemblador MIPS
  - `*(a+1) = b & 1 || c >> 1;`



Exercici 2

```
func:           ble     $a0, $a2, etiq1
                ble     $a1, $a2, etiq2
etiq1:          beq     $a2, $zero, etiq3
etiq2:          li      $v0, 0
                b       etiq4
etiq3:          li      $v0,1
etiq4:
                jr      $ra
```

```
int func(int x, int y, int z)
{
        int res;

        if ( ((          )&&(          )) || (          ) )

        {
                res = 0;
        } else
        {
                res = 1;
        }
        return res;
}
```

Exercici 3 - a,b,c,d a $t0, $t1, $t2, $t3

```
unsigned int a, b, c;
int d;

if (((a<=b) && (a>c)) || (d!=0))
        d = -1;
else
        d = 0;
```

```
et1:    [    ]  $t0, $t1, [    ]
et2:    [    ]  $t0, $t2, [    ]
et3:    [    ]  $t3, $zero, [    ]
et4:    li      $t3, -1
et5:    b       et7
et6:    move    $t3, $zero
et7:
```

Exercici 4

- Escriu el valor final en hexadecimal del registre `$t0`

```
li   $t0, 0x0020A040
sw   $t0, 0($sp)
lb   $t0, 1($sp)
srl  $t0, $t0, 4
ori  $t0, $t0, 0x0099
```

Exercici 5

- Escriu el valor final en hexadecimal del registre `$t0`

```
addiu    $t0, $zero, -1
sltu     $t0, $zero, $t0
addiu    $t0, $t0, -1
```

Exercici 6

- Donades les següents declaracions en C

```
char a;
int b;
long long int c;
main() {
  char *p;           /* punter guardat en $t0 */
  int *q;            /* punter guardat en $t1 */
  long long int *h; /* punter guardat en $t2 */
}
```

- Tradueix a MIPS les següents sentències en C que formen part de la funció `main`:

1. `q = q + 1;`
2. `a = *p;`
3. `h = &c;`
4. `b = *(q + b);`
5. `p[*q + 10] = a;`
6. `h = &h[*p];`

Exercici 7

- Tradueix la següent subrutina a assemblador MIPS

```
int fib(int n) {
    int tmp;
    if (n<2)
        tmp = n;
    else
        tmp = fib(n-1) + fib(n-2);
    return tmp;
}
```

Exercici 8

- Quins elements de `subr1` s'han de guardar en registres de tipus segur `$s`
- Dibuixa el bloc d'activacio de `subr1` i tradueix-la a MIPS

```
void subr2(short *x, int *y, char *z);
void subr1(int a[], int b) {
  char k;
  short v[7];
  int i;
        for (i = 0; i < b; i++)
              subr2(v, &a[3], &k);
}
```

Exercici 9

- Tradueix la següent subrutina a assemblador MIPS

```
short s3(unsigned long long *p1) {
    short v1;
    if (*p1 != 0)
        v1 = 1 + s3(p1+1);
    else
        v1 = 0;
    return v1;
}
```