Enunciats de la sessió

Activitat 1.A: Declaracions amb alineació en memòria automàtica

Exercici 1.1: Tradueix a assemblador la següent declaració de variables globals en C:

C	Assemblador MIPS
`.data`	`.data`
`char aa = -5;`	`aa: .byte -5`
`short bb = -344;`	`bb: .half -344`
`long long cc = -3;`	`cc: .dword -3`
`unsigned char dd = 0xA0;`	`dd: .byte 0xA0`
`int ee = 5799;`	`ee: .word 5799`
`short ff = -1;`	`ff: .half -1`

Exercici 1.2: Sabent que les dades globals s'emmagatzemen a partir de l'adreça 0x10010000, escriviu el contingut de memòria de la declaració de l'exercici anterior, byte per byte, en ordre little-endian, i escriviu cada etiqueta a la posició que correspongui. Indiqueu amb una 'X' les posicions de memòria que el compilador deixa sense ocupar a fi d'alinear les dades (per defecte l'alineació és automàtica):

Etiqueta	@Memòria	Contingut	Etiqueta	@Memòria	Contingut
aa:	0x10010000	0xFB		0x1001000E	0xFF
	0x10010001	X		0x1001000F	0xFF
bb:	0x10010002	0xA8	dd:	0x10010010	0xA0
	0x10010003	0xFE		0x10010011	X
	0x10010004	X		0x10010012	X
	0x10010005	X		0x10010013	X
	0x10010006	X	ee:	0x10010014	0xA7
	0x10010007	X		0x10010015	0x16
CC:	0x10010008	0xFD		0x10010016	0x00
	0x10010009	0xFF		0x10010017	0x00
	0x1001000A	0xFF	ff:	0x10010018	0xFF
	0x1001000B	0xFF		0x10010019	0xFF
	0x1001000C	0xFF		0x1001001A	
	0x1001000D	0xFF		0x1001001B	

Comprovació pràctica

Engegueu el simulador MARS i carregueu el fitxer **s1a.s**. Copieu la declaració de les variables de l'exercici 1.1 i premeu F3 per assemblar el programa. Comproveu els continguts de memòria de l'exercici 1.2 amb els valors que apareixen a la vista de dades. Recordeu que aquesta vista mostra el contingut de memòria en format word (paraules de 4 bytes ordenades en little-endian).

Activitat 1.B: Inicialització de registres amb immediats i adreces

Feu una còpia del fitxer **s1a.s** amb el nom **s1b.s**. Inserteu en aquest fitxer el següent codi en assemblador MIPS i assembleu-lo. Observant la vista de Codi desassemblat del MARS, indiqueu en quines instruccions s'expandeixen cada una de les següents macros:

Macros MIPS	Instruccions MIPS
`la $s3, aa`	lui $1, 0x00001001 ori $19, $1, 0x00000000
`li $s4, 65535`	ori $20, $0, 0x0000FFFF
`li $s5, 65536`	lui $1, 0x00000001 ori $21, $1, 0x00000000
`move $s0, $s1`	addu $16, $0, $17

Activitat 1.C: Accés a variables de tipus elemental en memòria

Exercici 1.3: Les instruccions en negreta del següent codi accedeixen a memòria per llegir (o escriure) les variables globals de l'exercici 1.1. Escriviu, per a cada variable del programa l'adreça i mida. També escriviu el valor final dels registres destinació de les instruccions de load que hi accedeixen (ressaltades en negreta) o el contingut de memòria (en cas d'escriptura) a partir dels resultats calculats a l'exercici 1.2:

Codi assemblador MIPS	Adreça efectiva d'accés a memòria	Núm bytes accedits	Valor llegit/escrit (hex 32/64 bits)

`main:`
`la $s0, aa`
`lb` `$s1, 0($s0)`
	0x10010000	1 byte	0xFFFFFFF8

`la $s0, bb` **`lh`** `$s2, 0($s0)`	0x10010002	2 bytes	0xFFFFFEA8
`la $s0, cc` **`lw`** `$s3, 0($s0)` **`lw`** `$s4, 4($s0)`	0x10010008	8 bytes	0xFFFFFFFFFFFFFFFD
`la $s0, dd` **`lbu`** `$s5, 0($s0)`	0x10010010	1 byte	0x000000A0
`la $s0, ff` **`lh`** `$s6, 0($s0)`	0x10010018	2 bytes	0xFFFFFFFF
`sh` `$s1, 0($s0)`	0x10010018	2bytes	0x0000FFF8

Comprovació pràctica

Feu una còpia del fitxer **s1a.s** amb el nom **s1c.s**. Afegiu-hi el codi anterior i executeu el programa pas a pas (tecla F7), tot comprovant que les respostes anteriors són correctes.

Activitat 1.D: Operacions amb punters a variables globals

Exercici 1.4: Donada la següent declaració de dades en assemblador MIPS (un punter inicialitzat amb l'adreça d'una altra variable global), i suposant que les variables estan emmagatzemades en memòria a partir de l'adreça 0x10010000, escriviu el valor en hexadecimal de cada una de les següents expressions en C:

```
        .data
dada:   .half 3
pdada:  .word dada
```

&pdada	0x10010004	&dada	0x10010000
pdada	0x10010000	dada	0x00000003
*pdada	0x00000003		

Activitat 1.E: Accés indirecte a una variable a través d'un punter

Exercici 1.5: Traduïu a assemblador MIPS el següent programa escrit en C, omplint les caselles en blanc. Considereu que la variable `temp` es guardarà al registre $s0:

C	Assemblador MIPS
`int A[3] = {3, 5, 7};`	`.data` `A: .word 3,5,7`
`int *punter = 0;`	`punter: .word 0`

`void main()`
`{`
	`.text`
	`.globl main`
	`main:`

`int temp;`	

`punter = &A[2];`	`li $s1, 2`
	`sll $s1, $s1, 2`
	`la $s2, A`
	`addu $s3, $s1, $s2`

`temp = *punter + 2;`	`lw $s4, 0($s3)` `addiu $s0, $s4, 2`

`temp = *(punter-2) + temp;`	`subu $s5, $s3, $s1`
	`lw $s6, 0($s5)`
	`addu $s0, $s6, $s0`

`A[1] = temp;`	`addiu $s7, $s2, 4` `sw $s0, 0($s7)`

`print_integer(temp);`
`// Consultar lectura prèvia`
	`li $v0, 1`
	`move $a0, $s0`
	`syscall`

`// main retorna al codi de startup` `}`	`jr $ra`

Comprovació pràctica

Copieu el codi anterior al fitxer **s1e.s.** Salveu-lo, assembleu-lo i executeu-lo. Comproveu que el programa mostra per la consola d'entrada/sortida del simulador MARS el número 12, valor de la variable temporal `temp`. Comproveu també a la vista de dades que `A[1]` val 12.

Activitat 1.F: Tipus estructurats de dades: el vector

Exercici 1.6:

Donat el següent vector global `vec` de 10 elements de tipus enter:

```
int vec[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
```

A continuació, escriviu la declaració del vector vec en assemblador MIPS:

```
vec: .word  9,8,7,6,5,4,3,2,1,0
```

Escriviu també la fórmula per al càlcul de l'adreça de l'element `vec[i]`, en funció de l'adreça inicial de `vec` i del valor de l'índex `i`:

```
@vec[i] = @vec[0] + i * T   * Tamaño T bytes.
```

A partir de la fórmula anterior, escriviu un fragment de codi en assemblador MIPS tal que copïi en el registre $s1 el valor de `vec[i]`, és a dir: `$s1 <- vec[i]`, suposant que el valor de i es troba al registre $s2.

```
la   $t0, vec
sll  $t1, $s2, 2
addu $t0, $t0, $t1
lw   $s1, 0($t0)
```

Activitat 1.G: Accés aleatori als elements d'un vector

Suposem un vector global de 10 elements enters `fib`. El codi en C mostrat a continuació escriu en els 10 elements del vector els 10 primers valors de la sèrie de Fibonacci

```
int fib[10];

void main() {
     int i  = 2;
     fib[0] = 0;
     fib[1] = 1;
     while (i < 10){
          fib[i] = fib[i-1] + fib[i-2];
          i++;
     }
}
```

Completeu a continuació l'exercici 1.7.

Exercici 1.7: Traduïu a assemblador MIPS el següent programa escrit en C, omplint les caselles en blanc. Considereu que la variable `i` es guardarà al registre $s0:

C	Assemblador MIPS
`int fib[10];`	`.data` `fib: .space 40`

`void main()`
`{`
`  int i = 2;`
`  fib[0] = 0;`
`  fib[1] = 1;`
	`.text`
	`.globl main`
	`main:`
	`li $s0,2`
	`la $t3, fib`
	`li $t1, 1`
	`sw $zero, 0($t3)`
	`sw $t1, 4($t3)`

`  while (i < 10)`
`  {`
	`while:`
	`slti $t0, $s0, 10`
	`beq $t0, $zero, fi`

`    fib[i] = fib[i-1] + fib[i-2];`	`addiu $t4, $s0, -1`
	`sll $t5, $t4, 2`
	`addu $t5, $t3, $t5`
	`lw $t5, 0($t5)`
	`addiu $t6, $s0, -2`
	`sll $t7, $t6, 2`
	`addu $t7, $t3, $t7`
	`lw $t7, 0($t7)`
	`sll $t8, $s0, 2`
	`addu $t8, $t3, $t8`
	`addu $t2, $t5, $t7`
	`sw $t2, 0($t8)`

`    i++;` `  }`	`addiu $s0, $s0, 1` `b while`
`  // main retorna al codi de startup` `}`	`fi:` `jr $ra`

Comprovació pràctica

Copieu el codi de l'exercici 1.7 a l'arxiu **s1g.s**. Salveu-lo, assembleu-lo i executeu-lo. Comproveu en la zona de memòria que el contingut del vector `fib` és 0 1 1 2 3 5 8 13 21 34.

Activitat 1.H: Cadenes de caràcters (strings)

Sigui el vector de naturals `vec`, el qual conté els dígits (números del 0 al 9) de la representació en decimal del número natural `num=19865`. El primer element del vector representa el dígit de menor pes. El següent programa en C converteix cada un dels elements del vector `vec` a la seva representació ASCII i els emmagatzema en el string `cadena`:

```
char cadena[6];
unsigned int vec[5] = {5, 6, 8, 9, 1};

void main() {
     int i=0;
     while (i<5)
     {
          cadena[i] = vec[4-i] + '0';
          i++;
     }
     cadena[5]=0;       // posa la marca de final de string

     print_string(cadena);
}
```

Observeu que el programa escriu a l'string `cadena` els dígits decimals de `num` començant pel de major pes, ja que volem que es pugui llegir el número correctament quan imprimim el string per pantalla. Per aquesta raó, a cada iteració del bucle es converteix el dígit `vec[5-1-i]` en comptes de convertir el dígit `vec[i]`. Noteu que la conversió a ASCII es fa sumant 48 al dígit en decimal, o el que és el mateix, el codi ASCII de '0'. Fixeu-vos també que quan es declara el vector de caràcters `cadena` es reserva espai per a `5+1` elements, per tal de poder guardar el valor *sentinella* 0, que assenyala el final del string.

Completeu a continuació l'exercici 1.8

Exercici 1.8: Traduïu a assemblador MIPS el següent programa escrit en C, omplint les caselles en blanc. Considereu que la variable `i` es guardarà al registre $s0:

C	Assemblador MIPS

`char cadena[6]={-1,-1,-1,-1,-1,-1};`
`unsigned int vec[5]={5, 6, 8, 9, 1};`
	`.data`
	`cadena: .byte -1,-1,-1,-1,-1,-1`
	`vec: .word 5,6,8,9,1`

`void main()`
`{`
	`.text`
	`.globl main`
	`main:`

`int i=0;`	`li $s0, 0`
`while (i < 5)` `{`	`while: li $t0,5` `bge $s0, $t0, fi`

`cadena[i]=vec[4-i] + '0';`	`la $s3, vec`
	`sll $s1, $s0, 2`
	`addiu $s3, $s3, 16`
	`subu $s4, $s3, $s1`
	`lw $s5, 0($s4)`
	`addiu $s6, $s5, 48`
	`la $s7, cadena`
	`addu $s1, $s7, $s0`
	`sb $s6, 0($s1)`

`i++;`
`}`
	`addiu $s0, $s0, 1`
	`b while`
	`fi:`

`cadena[5]=0;`	`addu $s2, $s7, $s0` `sb $zero, 0($s2)`

`print_string(cadena);`
`// consulteu lectura prèvia`
	`li $v0, 4`
	`move $a0, $s7`
	`syscall`

`// main retorna al codi de startup` `}`	`jr $ra`

Comprovació pràctica

Copieu el codi de l'anterior exercici a l'arxiu **s1h.s**. Verifiqueu el correcte funcionament del programa de manera que imprimeixi la cadena de caràcters: "19865". Comproveu també a la vista de dades que al final del programa la variable `cadena` representa aquest mateix número.

Alerta, perquè la vista de dades mostra la memòria en format word:

Per exemple, suposem el string "0123456". Estaria format pels 8 elements 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x00 (sentinella), i es guardarien en memòria en aquest mateix ordre. Però MARS mostra a la vista de Dades tot el contingut de la memòria suposant que tot són words (agrupant els bytes de 4 en 4). Així doncs, el primer word del string estaria format pels bytes 0x30, 0x31, 0x32, 0x33, amb el byte 0x30 guardat en primer lloc. El primer byte és el de menys pes dels quatre si els interpretem com un word, de manera que en l'escriptura normal en hexadecimal, tal com ho mostra MARS a la vista de dades, apareix escrit a la dreta: 0x33323130. De la mateixa manera, el segon word apareixeria escrit: 0x00363534.

Anàlogament, la variable `cadena` del nostre exercici hauria de contenir la seqüència "19865" que està formada pels bytes: 0x31, 0x39, 0x38, 0x36, 0x35, 0x00, guardats en memòria en aquest ordre. Així doncs, ¿com s'hauria de mostrar la cadena, en format word hexadecimal, a la vista de dades?

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0x36383931	0x00000035	...
0x10010020	0x00000000	0x00000000	...