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	///////////////		0x1001000F	0xFF
bb:	0x10010002	0xA8	dd:	0x10010010	0xA0
	0x10010003	0xFE		0x10010011	///////////////
	0x10010004	///////////////	ee:	0x10010012	0xA7
	0x10010005	///////////////		0x10010013	0x16
	0x10010006	///////////////		0x10010014	0x00
	0x10010007	///////////////		0x10010015	0x00
cc:	0x10010008	0xFD	ff:	0x10010016	0xFF
	0x10010009	0xFF		0x10010017	0xFF
	0x1001000A	0xFF		0x10010018	
	0x1001000B	0xFF		0x10010019	
	0x1001000C	0xFF		0x1001001A	
	0x1001000D	0xFF		0x1001001B	

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

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)
	Adreça efectiva d'accés a memoria	Núm bytes accedits	Valor llegit/escrit (hex 32/64 bits)

`main:`
`la $s0, aa`
`lb` `$s1, 0($s0)`
	0x10010000	1byte	$s1 = 0x000000FB

`la $s0, bb` **`lh`** `$s2, 0($s0)`	**0x10010002**	2bytes	$s2 = 0x0000FEA8
`la $s0, cc` **`lw`** `$s3, 0($s0)`	**0x10010008**	8bytes	$s3 = 0xFFFFFFFD
`lw` `$s4, 4($s0)`	**0x1001000C**		$s4 = 0xFFFFFFFF
`la $s0, dd` **`lbu`** `$s5, 0($s0)`	**0x10010010**	1byte	$s5 = 0x000000A0
`la $s0, ff` **`lh`** `$s6, 0($s0)`	**0x10010016**	2bytes	$s6 = 0x0000FFFF
`sh` `$s1, 0($s0)`	**0x10010016**	1byte	Mem_b[0x10010016] = 0xFB

Activitat 1.D: Operacions amb punters a variables global

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	0x10010001	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()`
`{`
`int temp;`
	`.text`
	`.globl main`
	`main:`

`punter = &A[2];`	la $t0, punter # $t0 = @punter la $t1, A # $t1 = @A[0] addiu $t1, $t1, 8 # $t1 = $t1 + 2 * 4 bytes sw $t1, 0 ($t0) # Memw[@punter] = @A[2]
`temp = *punter + 2;`	la $t1, punter # $t1 = @punter lw $s0, 0 ($t1) # $s0 = @A[2] lw $s0, 0 ($s0) # $s0 = A[2] = 7 addiu $s0, $s0, 2 # $s0 = A[2] + 2 # = 7 + 2 = 9
`temp = *(punter-2) + temp;`	la $t0, punter # $t0 = @punter lw $t0, 0 ($t0) # $t0 = Memw[@punter] addiu $t0, $t0, - 8 # $t0 = @[Memw[@punter] – 2] lw $t0, 0 ($t0) # $t0 = Memw[@punter] - 2 addu $s0, $s0, $t0 # $s0 = $s0 + $t0
`A[1] = temp;`	la $t0, A # $t0 = @A[0] addiu $t0, $t0, 4 # $t0 = @A[0] + 1*4 bytes = @A[1] sw $s0, 0 ($t0) # A[1] = temp = $s0
`print_integer(temp);` `// Consultar lectura prèvia`	li $v0, 1 # $v0 = 1 (identificador) move $a0, $s0 # $a0 = temp (enter a imprimir) syscall # mostra temp en pantalla
`// main retorna al codi de startup` `}`	jr $ra

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 *(tamany en 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          # $t0 = @vec[0]
sll $t1, $s2, 4      # $t1 = i * 4bytes
lw $s1, 0 ($s2)      # $s1 = Memw[@vec[i]] = vec[i]
```

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

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

`int i = 2;`
`fib[0] = 0;`
`fib[1] = 1;`
	li $s0, 2 # $s0 = 2
	la $t0, fib # $t0 = @fib[0]
	li $t1, 0 # $t1 = 0
	sw $t1, 0 ($t0) # fib[0] = 0
	addiu $t0, $t0, 4 # $t0 = @fib[1]
	li $t1, 1 # $t1 = 1
	sw $t1, 0 ($t0) # fib[1] = 1

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

`fib[i] = fib[i-1] + fib[i-2];`	la $t0, fib # $t0 = @fib[0]
	addiu $t1, $s0, - 4 # $t1 = i – 1*4bytes
	addiu $t2, $s0, - 8 # $t2 = i – 2*4bytes
	addu $t1, $t1, $t0 # $t1 = @fib[i-1]
	addu $t2, $t2, $t0 # $t2 = @fib[i-2]
	lw $t1, 0 ($t1) # $t1 = fib[i-1]
	lw $t2, 0 ($t2) # $t2 = fib[i-2]
	addu $t3, $t1, $t2 # $t3 = $t1 + $t2
	sll $t4, $s0, 2 # $t4 = i*4bytes
	addu $t0, $t0, $t3 # $t0 = @fib[i]
	sw $t3, 0 ($t0) # fib[i] = $t3
	# = fib[i-1] + fib[i-2]

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

Activitat 1.H: Cadenes de caràcters (strings)

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 $t1, vec # $t1 = @vec[0]
	li $t2, 4 # $t2 = 4
	subu $t2, $t2, $s0 # $t2 = 4 - i
	sll $t2, $t2, 2 # $t2 = (4 - i)*4 bytes
	addu $t1, $t1, $t2 # $t1 = @vec[4-i]
	lw $t1, 0 ($t1) # $t1 = vec[4-i]
	addiu $t1, $t1, 48 # $t1 = vec[4-i] + '0'
	la $t2, cadena # $t2 = @cadena[0]
	addu $t2, $t2, $s0 # $t2 = @cadena[i]
	sb $t1, 0 ($t2) # cadena[i] = $t1

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

`cadena[5]=0;`	la $t1, cadena # $t1 = @cadena[0] li $t2, 0 # $t2 = 0 sb $t2, 5 ($t1) # cadena[5] = $t2 = 0
`print_string(cadena);` `// consulteu lectura prèvia`	li $v0, 4 # $v0 = 1 (identificador) move $a0, $t1 # $a0 = @cadena[0] syscall # mostra cadena en pantalla
`// main retorna al codi de startup` `}`	jr $ra