Estructura de Computadores

Tema 2. Ensamblador MIPS y tipos de datos básicos

ASCII

- Sistema de codificación de caracteres
- Asigna un código numérico a cada símbolo
- En EC estudiaremos el ASCII de 7 bits
  - Los caracteres se almacenan utilizando 1 byte (8 bits)
  - El bit de mayor peso siempre vale 0
  - Códigos del 0 al 31 son de control
  - El resto de códigos son símbolos tipográficos (imprimibles)

Código	**Símbolo**	**En C y MIPS**
0x09	TAB	'\t'
0x0A	LF	'\n'
0x30	1	'1'
0x41	A	'A'
0x61	a	'a'

Propiedades de la codificación ASCII

- Orden alfabético
  - 'a' = 97, 'b' = 98, 'c'= 99, etc.
  - 'A' = 65, 'B' = 66, 'C'=67, etc.
  - '0' = 48, '1' = 49, '2'=50, etc.
- Conversión minúscula/mayúscula
  - De mayúscula a minúscula sumando 32: 'a' = 'A' + 32
  - De minúscula a mayúscula restando 32: 'B' = 'b' - 32
- Conversión dígito ASCII/valor numérico
  - Caracter '1' representa el dígito 1 y tiene un código ASCII de 49
  - Se puede converir un dígito (número natural) a su código ASCII sumando 48
    - 1 + 48 = '1'
  - También se puede convertir sumando el '0'
    - 1 + '0' = '1'

Declaración de variables de tipo caracter

- En C, usamos el tipo `char`:
  - `char letra = 'R'`
- En MIPS:
  - `letra: .byte 'R'`

Formato de las instrucciones MIPS

- Las instrucciones se representan con cadenas de bits y se almacenan en memoria
- MIPS32: Instrucciones de 32 bits
- Formatos de instrucción en MIPS: R (register), I (immediate) y J (jump)

Formato	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	imm16		
J	opcode	target				

Formato de las instrucciones MIPS

- `opcode`: código de operación
- `funct`: extensión del código de operación
- `rs`, `rt`, `rd`: operandos en modo registro
- `imm16`: operando en modo inmediato de 16 bits
- `shamt`: shift amount, inmediato para desplazamientos
- `target`: dirección de destino de un salto

Formato	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	imm16		
J	opcode	target				

Ejemplos

		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
addu rd, rs, rt	R	000000	rs	rt	rd	00000	100001
sra rd, rt, shamt	R	000000	rs	rt	rd	shamt	000011
addiu rt, rs, imm16	I	001000	rs	rt	imm16		
lui rt, imm16	I	001111	00000	rt	imm16		
lw rt, offset16(rs)	I	100011	rs	rt	offset16		
j target	J	000010	target				

Ejercicio

- Codifica en lenguaje máquina las siguientes instrucciones en lenguaje ensamblador MIPS:
  - `addu $t4, $t3, $t5`
  - `addiu $t7, $t6, 25`
  - `lw $t3, 0($t2)`

Ejercicio

- Codifica en lenguaje máquina las siguientes instrucciones en lenguaje ensamblador MIPS:
  - `addu $t4, $t3, $t5`
  - `addiu $t7, $t6, 25`
  - `lw $t3, 0($t2)`
- Desensambla las siguientes instrucciones:
  - 0xAE0BFFFC
  - 0x8D08FFC0
  - 0x0233a823

Punteros

- Variable que contiene una dirección de memoria
  - 32 bits en MIPS32
  - Similar a una variable de tipo entero
- Si `p` contiene la dirección de memoria de la variable `v` decimos que `p` apunta a `v`
- Declaración de punteros:

```
int *p1, *p2;
char *p3
```

```
.data
p1: .word 0
p2: .word 0
p3: .word 0
```

Inicialización de punteros

- Asignando otro puntero del mismo tipo
- Asignando la dirección de una variable (operador &)

```
char a = 'E';
char b = 'K';
char *p = &a;

void f() {
    p = &b;
}
```

```
.data
a:  .byte 'E'
b:  .byte 'K'
p:  .word a

.text
f:
    la $t0, b
    la $t1, p
    sw $t0, 0($t1)
```

Desreferencia (indirección) de punteros

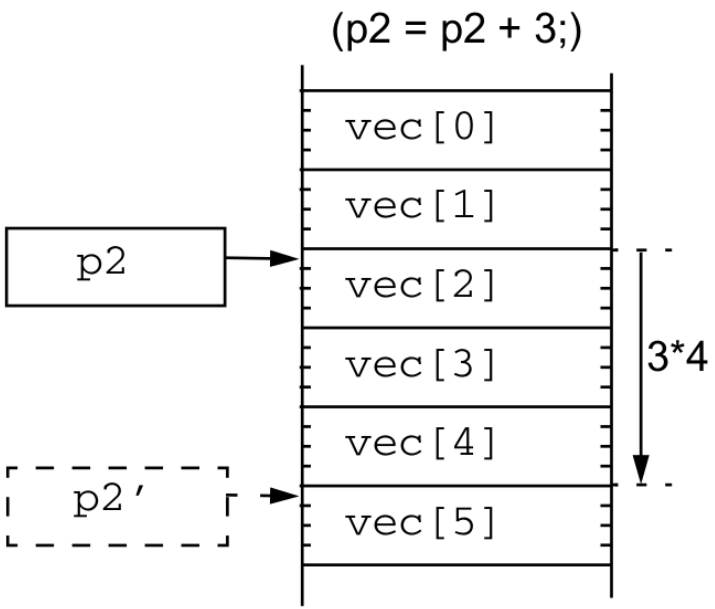
- Los punteros sirven para acceder a las variables apuntadas
- La desreferencia consiste en acceder a la dirección de memoria apuntada por el puntero
- En C se utiliza el operador *
  - *p: contenido de la dirección de memoria apuntada por p
  - No confundir con la declaración de punteros

```
char a = 'E';
char *p = &a;

void f() {
    char tmp = *p;
}
```

```
.data
a: .byte 'E'
p: .word a

.text
f: la $t0, p
   lw $t1, 0($t0)
   lb $t2, 0($t1)
```

Aritmética de punteros

- Suma de un puntero `p` más un entero `N`
- Da como resultado otro puntero `q` del mismo tipo: `q = p + N`
- `q` apunta a otro entero situado `N` elementos más adelante



Aritmética de punteros

- Ejemplo

En C

```
char *p1;
int *p2;
long long *p3;
...
p1 = p1 + 3;
p2 = p2 + 3;
p3 = p3 + 3;
```

En MIPS

```
p1: .word 0
p2: .word 0
p3: .word 0
...
addiu $t1, $t1, 3
addiu $t2, $t2, 12
addiu $t3, $t3, 24
```

Ejercicio

- Dadas las siguientes declaraciones en C:
  - `int dada;`
  - `int *pdada;`
- Traduce a MIPS las siguientes sentencias en C:
  1. `pdada = &dada;`
  2. `*pdada = *pdada + 1;`
  3. `pdada = pdada + 1;`
  4. `dada = dada - 1;`

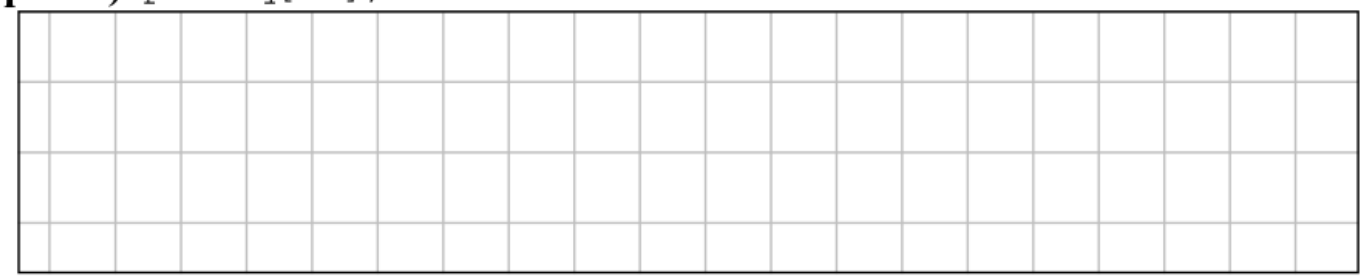
Ejercicio

Donades les següents declaracions de variables locals

```
short *p, *q, x, y;
```

Sabent que les variables `p`, `q`, `x`, `y` estan guardades en els registres $t0, $t1, $t2, $t3 respectivament, tradueix a assemblador MIPS les següents sentències:

a) (0,25 punts) `p = &q[10];`



b) (0,25 punts) `y = *(q + x);`