Estructura de Computadores

Tema 2. Ensamblador MIPS y tipos de datos básicos

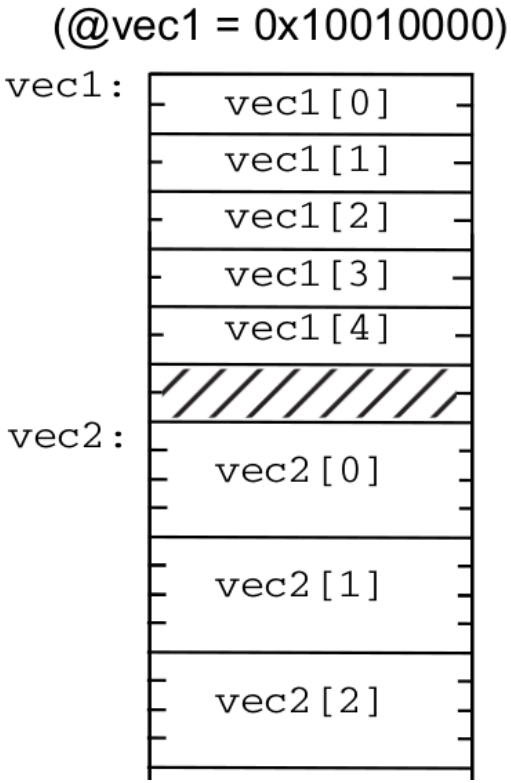
Vectores

- Agrupaciones unidimensionales de elementos del mismo tipo, los cuales se identifican por un índice
- Los elementos se almacenan en posiciones consecutivas a partir de la dirección inicial del vector
- El primer elemento tiene índice 0
- En MIPS, los elementos han de respetar la reglas de alineación
  - Como todos los tipos tienen tamaño múltiplo de 2, si el primer elemento del vector está alineado el resto también
  - No es necesario insertar espacio intermedio entre los elementos

Vectores

```
/* En C */
short vec1[5] = {0, -1, 2, -3, 4};
int vec2[100];
```

```
# En MIPS
.data
vec1: .half 0, -1, 2, -3, 4
      .align 2
vec2: .space 400
```



Acceso a un elemento de un vector

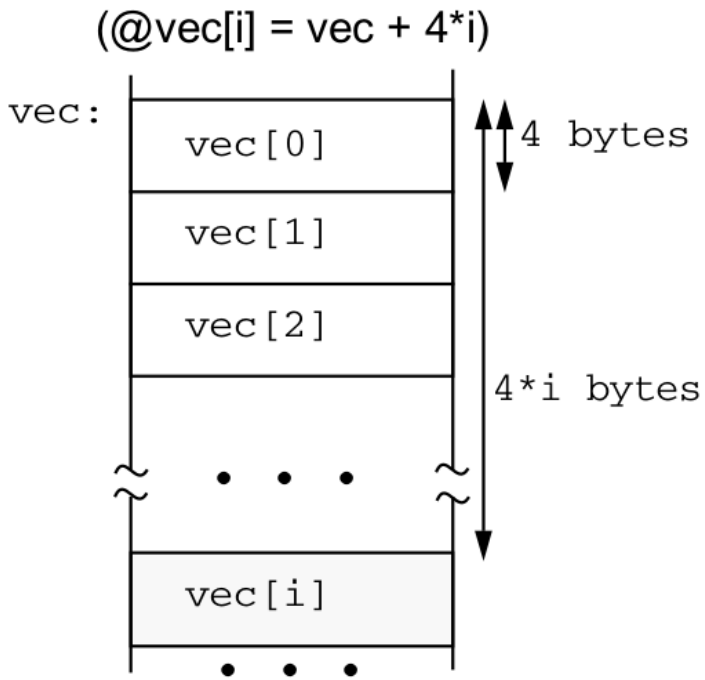
- Para acceder a un elemento `i` de un vector hay que calcular su dirección

Acceso a un elemento de un vector

- Para acceder a un elemento `i` de un vector hay que calcular su dirección
- Si los elementos tienen tamaño `T` bytes, la dirección del elemento `i` es:
  - `@vec[i] = @vec[0] + i * T`

Acceso a un elemento de un vector

- Para acceder a un elemento `i` de un vector hay que calcular su dirección
- Si los elementos tienen tamaño `T` bytes, la dirección del elemento `i` es:
  - `@vec[i] = @vec[0] + i * T`



Ejemplo

- Dada la siguiente sentencia en C:
  - `vec[i] = 10;`
- Indica el código en lenguaje ensamblador del MIPS asumiendo que `i` está en `$t0` y que `vec` es un vector global de enteros de 32 bits.

Ejemplo

- Dada la siguiente sentencia en C:
  - `vec[i] = 10;`
- Indica el código en lenguaje ensamblador del MIPS asumiendo que `i` está en `$t0` y que `vec` es un vector global de enteros de 32 bits.

```
la   $t2, vec          # $t2 = @vec[0]
sll  $t1, $t0, 2       # $t1 = 4 * i
addu $t2, $t2, $t1     # $t2 = @vec[0] + 4 * i
li   $t1, 10
sw   $t1, 0($t2)
```

Ejercicio

- Dadas las siguientes declaraciones globales en C:

```
int val[100], vec[100];
int elem;
```

- Traduce las siguientes sentencias en C a lenguaje ensamblador MIPS:
  1. `elem = val[5] + vec[10];`
  2. `elem = vec[10 + val[5]];`

Vectores y punteros

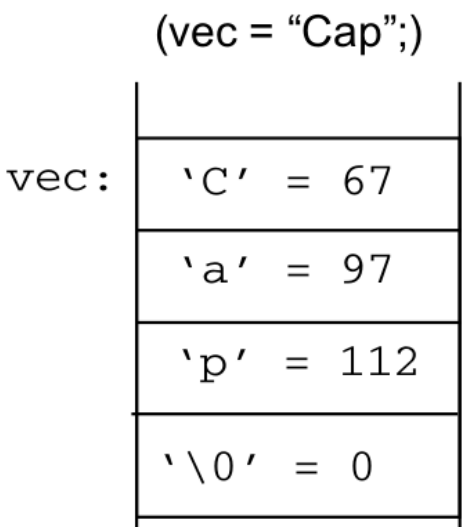
- En C, los vectores tienen el mismo tipo que un puntero y siempre apuntan al primer elemento del vector
  - `int vec[100];`
  - `int *p;`
- Podemos inicializar un puntero asignandole un vector pero no al revés
  - `p = vec;`
- Podemos utilizar el operador [] con un puntero
  - `p[8] = 0;`
- Podemos utilizar el operador de indirección * con un vector
  - `*vec = 0;`

Vectores y punteros

- Un puntero se puede considerar como un vector, el primer elemento del cual es el apuntado por `p`
- La siguiente expresión es válida:
  - `*(p + i) = 0;`
- Y es equivalente a la expresión:
  - `p[i] = 0;`

Strings (cadenas de caracteres)

- Vectores con un número variable de caracteres
- El último caracter de la cadena (centinela) siempre vale 0 ('\0')
- String "Cap" se representa con los caracteres 67, 97, 112, 0



Declaración de strings

- En C:

```
char cadena[] = "Una frase";
```

Declaración de strings

- En C:

```
char cadena[] = "Una frase";
```

- El compilador reserva 10 bytes: 9 caracteres más el centinela

Declaración de strings

- En C:

```
char cadena[] = "Una frase";
```

- El compilador reserva 10 bytes: 9 caracteres más el centinela
- En MIPS:

```
        .data
cadena: .ascii "Una frase"
        .byte 0                 # centinela
```

Declaración de strings

- En C:

```
char cadena[] = "Una frase";
```

- El compilador reserva 10 bytes: 9 caracteres más el centinela
- En MIPS:

```
        .data
cadena: .ascii "Una frase"
        .byte 0                 # centinela
```

- Alternativa en MIPS:

```
        .data:
cadena: .asciiz "Una frase"
```

Acceso a los elementos de un string

- En C los caracteres se codifican en ASCII (1 byte)

Acceso a los elementos de un string

- En C los caracteres se codifican en ASCII (1 byte)
- Se acceden utilizando las instrucciones `lb` y `sb`

Acceso a los elementos de un string

- En C los caracteres se codifican en ASCII (1 byte)
- Se acceden utilizando las instrucciones `lb` y `sb`
- Mismo método que se utiliza para acceder a vectores:
  - `char cadena[] = ‘‘Una frase’’;`
  - `@cadena[i] = @cadena[0] + i;`
  - Tamaño de elemento es siempre 1 para los strings

Acceso a los elementos de un string

- En C los caracteres se codifican en ASCII (1 byte)
- Se acceden utilizando las instrucciones `lb` y `sb`
- Mismo método que se utiliza para acceder a vectores:
  - `char cadena[] = ‘‘Una frase’’;`
  - `@cadena[i] = @cadena[0] + i;`
  - Tamaño de elemento es siempre 1 para los strings
- Traduce a ensamblador MIPS la siguiente sentencia en C, asumiendo que las variables `i` y `j` están en los registros `$t0` y `$t1` respectivamente:
  - `cadena[i] = cadena[j] - 32;`

Problema (Parte I)

- Dadas las siguientes declaraciones de variables en lenguaje C:

```
short a[] = {513, 17, -5};
long long b = 1030;
short c = 0;
char d[] = ``2017'';
short *e = &c;
```

- Escribe el programa MIPS equivalente
- Indica el contenido de la memoria en hexadecimal, asumiendo que las variables globales se almacenan a partir de la dirección 0x10010000. El código ASCII del '0' es 0x30.

Continuación Problema (Parte II)

- Dado el siguiente código en ensamblador MIPS, indica el valor final en hexadecimal del registro $t1

```
la      $t0, d
lbu     $t0, 1($t0)
addiu   $t1, $t0, -0x31
```

Continuación Problema (Parte III)

- Traduce a lenguaje ensamblador MIPS la siguiente sentencia en C:

$$*e = a[0] - a[2]$$

Problema

- Dada la siguiente declaración de variables globales en un programa MIPS, almacenadas a partir de la dirección 0x10010000:

```
        .data
a:      .byte       19, 0x91
b:      .space      2
c:      .asciiz     "ABCD"          # ASCII 'A' = 0x41
d:      .half       -33
e:      .word       a+1
f:      .dword      0xffffffff
```

- Escribe una declaración equivalente en lenguaje C
- Indica el contenido de las primeras 24 posiciones de memoria

Problema

- Traduce a MIPS el siguiente programa en C:

```
unsigned short v[10];
unsigned short *p = &v[2];

void main() {
    *(p + 5) = *p + 5;
}
```