Tema 2. Instruccions i tipus de dades bàsics

Estructura de Computadors (EC)

Rubèn Tous

rtous@ac.upc.edu
Computer Architecture Department
Universitat Politecnica de Catalunya

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índex

2.7 Representació de caràcters en ASCII

- Els llenguatges d'alt nivell acostumen a incloure un tipus caràcterper treballar amb textos.
- Fan servir alguna *codificació de caràcters* (character encoding).
- caràcter $\Longleftrightarrow$ representació (binari en el nostre cas).
- Actualment Unicode però nosaltres farem servir ASCII.
- La primera edició del codi ASCII és de 1963! (telegrafia).

2.7 Representació de caràcters en ASCII

- ASCII: 7 bits.
- 128 caràcters (33 no imprimibles).

codi	símbol	en C i MIPS
0x00	null	'\0'
...		
0x09	TAB	'\t'
0x0A	LF	'\n'
...		
0x0D	CR	'\r'
...		
0x20	space	' '

codi	símbol	en C i MIPS
0x30	0	'0'
0x31	1	'1'
...		
0x41	A	'A'
0x42	B	'B'
...		
0x61	a	'a'
0x62	b	'b'

2.7 Representació de caràcters en ASCII

Propietats de la codificació ASCII:

- Símbols decimals del '0' al '9' a partir del codi 48 (0x30).
- Majúscules a partir del codi 65 (0x41) i en en ordre alfabètic.
- Minúscules a partir del codi 97 (0x61) i en en ordre alfabètic.
- Caràcter null (0 o '\0' en C) és el 0x00.
- Espai (' ') el 0x20, salt de línia ('\n') el 0x0A.

2.7 Representació de caràcters en ASCII

Propietats de la codificació ASCII:

- Conversió majúscula/minúscula:

```
1 cmin = cmaj + 32;
```

- Conversió dígit ascii/valor numèric.

```
1 caracter_digit = '0' + numero;
```

2.7 Representació de caràcters en ASCII

- Declaració de variables de tipus caràcter (C):

```
1 char lletra = 'R';
```

- Donat que el bit de més pes sempre valdrà 0 no canviarà res si ho declarem com unsigned.
- En MIPS:

```
1 lletra: .byte 'R'
```

2.7 Representació de caràcters en ASCII

Exemple:

```
1   .data
2 c1: .byte 'A'
3 c2: .byte 0
4   .text
5   .globl main
6 main:
7   la $t0, c1
8   lb $t1, 0($t0)
9   addiu $t1, $t1, 32 # maj -> min
10  ...
11  li $t1, 7
12  addiu $t1, $t1, '0'
13  la $t0, c2
14  sb $t1, 0($t0)
```

2.8 Format de les instruccions MIPS

3 formats: R (register), I (immediate) i J (jumps).

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

2.8 Format de les instruccions MIPS

Exemples:

		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
`addu rd, rs, rt`	R	0x00	rs	rt	rd	0x00	0x21
`sra rd, rt, shamt`	R	0x00	rs	rt	rd	shamt	0x03
`addiu rt, rs, imm16`	I	0x08	rs	rt	imm16		
`lui rt, imm16`	I	0x0F	0x00	rt	imm16		
`lw rt, offset16(rs)`	I	0x23	rs	rt	offset16		
`j target`	J	0x02	target				

2.8 Format de les instruccions MIPS

Exemple: Codificar en binari la instrucció:

```
1 addu $t0, $s2, $zero
```

```
opcode = 0x00 = 000000
rs = $s2 = $18 = 10010 ($s0-$s7 -> $16-$23)
rt = $zero = $0 = 000000
rd = $t0 = $8 = 01000  ($t0-$t7 -> $8-$15)
shamt = 00000
funct = 0x21 = 100001
0000 00|10 010|0 0000| 0100 0|000 00|10 0001 = 0x02404021
```

Vectors

Vector (C array)

Agrupació unidimensional d'elements del mateix tipus i dentificats per un índex [0, N-1].

Declaració i emmagatzematge

En C:

```
1 ...
2 int vec[100];
3 ...
4 int vec2[5] = {0, 1, 2, 3, 4};
```

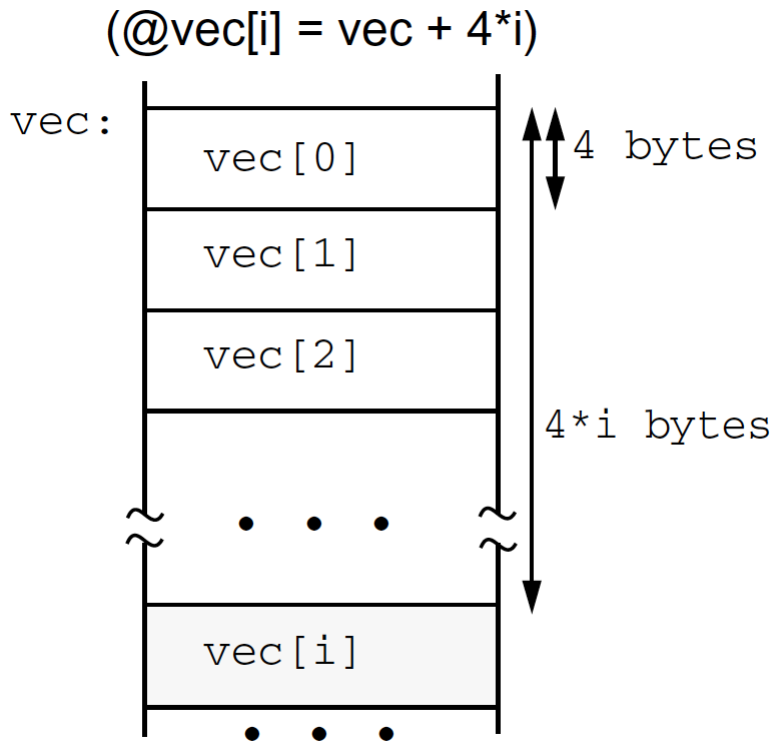
En MIPS:

```
1       .data
2 ...
3       .align 2
4 vec:  .space 400
5 ...
6 vec2:.word 0, 1, 2, 3, 4
```

Accés (aleatori) a un element

Per accedir a l'element i-èssim cal calcular la seva adreça:

$$@vec[i] = @vec[0] + i \cdot T$$



Strings

Emmagatzematge.

- En Java: com una tupla (un enter i un vector de caràcters).
- En C: vector de caràcters amb sentinella (caràcter '\0').

Strings

Declaració, en C (formes equivalents):

```
1 char cadena[20] = "Una␣frase";
2 char cadena[20] =
3       {'U', 'n', 'a', '␣', 'f', 'r', 'a', 's', 'e', '\0'};
```

En MIPS (formes equivalents):

```
1 cadena:.ascii "Una frase" # 9 car.
2         .space 11                # El sentinella i 10
              zeros
3 cadena:.asciiz "Una frase" # 10 (inclou sentinella)
4         .space 10
```

Strings

Accés als elements.

```
1  char nom[80];
2  main() {
3  int num=0;
4  ...
5  while (nom[num] != '\0')num += 1;
6     ...
7  }
```

```
1  main:...
2    move   $t0, $zero     # num = 0
3    la     $t1, nom
4  while:
5    addu   $t3, $t1, $t0 # $t3 = @nom[0] + num*1
6    lb     $t2, 0($t3)   # $t2 = nom[num]
7    beq    $t2, $zero, fiwhile
8    addiu  $t0, $t0, 1   # num += 1
9    b while
10   fiwhile:
11 ...
```

Punters

Punter

Un punter és una **variable** que conté una adreça de memòria. Per tant, en MIPS ocupa 32 bits. Si el punter p conté l'adreça de la variable v diem també que *p apunta a v*.

Punters

Com tota variable un punter pot ser variable **global** o **local**:

```
1  int *p1; // global
2
3  void main()
4  {
5     int *p2; // local
6     ...
7  }
```

Índex

Declaració

Declaració (si global):

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

```
1     .data
2 p1:.word 0
3 p2:.word 0
4 p3:.word 0
```

Si local no cal reservar espai, només decidir a quin registre anirà.

Índex

Inicialització

Operador adreça-de (&)

En C l'operador & davant una variable retorna l'**adreça de** la variable.

Inicialització

Inicialització dins la declaració (si global):

```
1 int v[100];
2 char a='E', b='K';
3 char *pglob = &a;
```

```
1 v:     .space 400
2 a:     .byte 'E'
3 b:     .byte 'K'
4 pglob:.word a    # char *pglob = &a
```

Inicialització

Inicialització dins una sentència (si punter global):

```
1  void f()
2  {
3     pglob = &b;
4  }
```

```
1  f:
2     la $t1, b    # &b
3     la $t2, pglob # &pglob
4     sw $t1, 0($t2)# pglob = &b
5     ...
```

Inicialització

Inicialització dins una sentència (si punter local):

```
1  void f()
2  {
3     int *ploc; /* en $t0 */
4     ploc = &b;
5  }
```

```
1  f:
2     la $t0, b   # ploc = &b
3     ...
```

Índex

Operació desreferència (indirection)

Operador adreça-de (&)

En C l'operador * davant un punter retorna l'**el valor de la variable a la que apunta (l'adreça continguda en)** el punter.

Operació desreferència (indirection)

Exemple desreferència (si punter global):

```
1  char *pglob;
2
3  void g()
4  {
5     char tmp; // a $t0
6     tmp = *pglob;
7     ...
8  }
```

```
1  g: ...
2     la $t2, pglob   # &pglob
3     lw $t3, 0($t2)  # pglob
4     lb $t0, 0($t3)  # *pglob
5     ...
```

Operació desreferència (indirection)

(escriptura)

```
1  char *pglob;
2
3  void g()
4  {
5     *pglob = 3;
6     ...
7  }
```

```
1  g:  ...
2     la  $t2, pglob   # &pglob
3     lw  $t3, 0($t2)  # pglob
4     li  $t4, 3
5     sb  $t4, 0($t3)  # *pglob = 3
6     ...
```

Operació desreferència (indirection)

Exemple desreferència (si punter local):

```
1  void g()
2  {
3     char tmp;     //$t0
4     char *ploc;   //$t1
5     ...
6     tmp = *ploc;
7     ...
8  }
```

```
1  g:
2     ...
3     lb $t0, 0($t1)
4     ...
```

Operació desreferència (indirection)

(escriptura)

```
1  void g()
2  {
3     char *ploc; //$t1
4     ...
5     *ploc = 3;
6     ...
7  }
```

```
1  g:
2     ...
3     li $t0, 3
4     sb $t0, 0($t1)
5     ...
```

2.11.3 Operació desreferència (indirection)

Operador de desreferència vs. declarador

No s'ha de confondre l'operador de desreferència * amb el declarador de punters explicat (mateix símbol).

```
1  int *p1;
2  void g(int *p2)
3  {
4     int *p3;
5     ...
6     *p1 = *p2;
7     ...
8  }
```

Índex

Aritmètica de punters

Aritmètica de punters

Sumar un enter N a un punter p, que apunta a un tipus de T bytes, dóna com a resultat un altre punter q = p + N * T.

```
1  char *p1;
2  int *p2;
3  long long *p3;
4  ...
5  p1 = p1 + 3;
6  p2 = p2 + 3;
7  p3 = p3 + 3;
```

Aritmètica de punters

```
1  char *p1;        //$t1
2  int *p2;         //$t2
3  long long *p3;   //$t3
4  ...
5  p1 = p1 + 3;
6  p2 = p2 + 3;
7  p3 = p3 + 3;
```

```
1 addiu $t1, $t1, 3
2 addiu $t2, $t2, 12
3 addiu $t3, $t3, 24
```

Índex

Relació entre punters i vectors

Aritmètica de punters

En C un vector és en realitat un punter que apunta al primer element.

```
1  int vec[100];
2  int *p;
```

Relació entre punters i vectors

Aritmètica de punters

En C un vector és en realitat un punter que apunta al primer element.

```
1  int vec[100];
2  int *p;
3
4  main()
5  {
6      ...
7      p = vec;  //OK
8      ...
9  }
```

Relació entre punters i vectors

L'expressió

```
1  *(p + i) = 0;
```

És equivalent a:

```
1  p[i] = 0;
```

Relació entre punters i vectors

```
1  int vec[100];
2  int *p;
3
4  main()
5  {
6     *v = 3;      /* v[1] = 3;
7     ...
8     p = vec;
9     p[8] = 10; /* element 8 del vector */
10    ...
11 }
```