

Tema 2

Assemblador i tipus de dades bàsics

Estructura de Computadors (EC)



Codificació ASCII



ASCII

- Sistema de codificació de caràcters
- Assigna un codi numèric a cada símbol
- A EC estudiem l'ASCII de 7 bits
 - Els caràcters s'emmagatzemen utilitzant 1 byte (8 bits)
 - El bit de major pes sempre val 0
 - Els primers 32 codis son de control
 - La resta son símbols tipogràfics (imprimibles)

Decimal	Codi (hex)	Símbol	En C i MIPS
9	0x09	TAB	'\t'
10	0x0A	LF	'\n'
48	0x30	0	'0'
65	0x41	A	'A'
97	0x61	a	'a'

Propietats de la codificació ASCII

- **Ordre alfabètic**
 - 'a' = 97, 'b' = 98, 'c' = 99, ...
 - 'A' = 65, 'B' = 66, 'C' = 67, ...
 - '0' = 48, '1' = 49, '2' = 50, ...
- **Conversió minúscula / majúscula**
 - De majúscula a minúscula sumant 32: 'a' = 'A' + 32
 - De minúscula a majúscula restant 32: 'B' = 'b' - 32
- **Conversió dígit ASCII a valor numèric**
 - Caràcter '1' representa el dígit 1 i té un codi ASCII de 49: '1' = 49
 - Es pot convertir un dígit (número natural) al seu codi ASCII sumant 48
 - $1 + 48 = '1'$ $0 + 48 = '0'$
 - També es pot convertir sumant '0'
 - $1 + '0' = '1'$

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Declaració de variables de tipo caràcter

- En C, usem el tipus char
 - `char lletra = 'L'`
- En MIPS
 - `lletra: .byte 'L'`



Format de les instruccions MIPS



Format de les instruccions MIPS

- Les instruccions es representen amb cadenes de bits i s'emmagatzemen a memòria - com les dades
- MIPS32 - instruccions de 32 bits
- Tres formats d'instrucció diferents
 - R (register), I (immediate) i J (jump)

Format	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				

Format de les instruccions MIPS

- opcode: codi de la operació
- funct: extensió del codi de l'operació
- rs, rt, rd: operands en mode registre
- imm16: operand en mode immediat de 16 bits
- shamt: shift amount, immediat per desplaçaments
- target: adreça destí d'un salt

Format	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				

Examples

		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	001001	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				

Exercici

- Codifica en llenguatge màquina les següents instruccions en llenguatge ensamblador MIPS
 - `addu $t4, $t3, $t5`
 - `addiu $t7, $t6, 25`
 - `lw $t3, 0($t2)`

Exercici

- Codifica en llenguatge màquina les següents instruccions en llenguatge ensamblador MIPS
 - `addu $t4, $t3, $t5` -> `0x016D6021`
 - `addiu $t7, $t6, 25` -> `0x25CF0019`
 - `lw $t3, 0($t2)` -> `0x8D4B0000`

Exercici

- Codifica en llenguatge màquina les següents instruccions en llenguatge ensamblador MIPS
 - `addu $t4, $t3, $t5`
 - `addiu $t7, $t6, 25`
 - `lw $t3, 0($t2)`
- Desassembla les següents instruccions
 - `0xAE0BFFFC`
 - `0x8D08FFC0`
 - `0x0233A823`

Exercici

- Codifica en llenguatge màquina les següents instruccions en llenguatge ensamblador MIPS
 - `addu $t4, $t3, $t5`
 - `addiu $t7, $t6, 25`
 - `lw $t3, 0($t2)`
- Desassembla les següents instruccions
 - `0xAE0BFFFC` `sw $t3, -4($s0)`
 - `0x8D08FFC0` `lw $t0, -64($t0)`
 - `0x0233A823` `subu $s5, $t1, $t3`



Punters



Punters

- Variable que conté una adreça de memòria
 - 32 bits en MIPS32
 - Similar a una variable de tipus enter (word)
- Si `p` conté l'adreça de memòria de la variable `v`
 - Diem que `p` **apunta a** `v`
- Declaració de punters

```
// Codi C
int *p1, *p2;
char *p3;
```

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


Inicialització de punters

- Assignant un altre punter del mateix tipus
- Assignant l'adreça d'una variable (operador unari &)

// Codi C

char a = 'A';

char b = 'B';

① char *p1 = &a;

void func() {

② char* p2 = &a; //p2 a \$t0

③ p1 = &b;

}

Assemblador

.data

a: .byte 'A'

b: .byte 'B'

① p1: .word a # p1 apunta a a

.text

func: la \$t0, a ② # inicialitza amb l'adreça

la \$t1, b # \$t1 = &b

③ { la \$t2, p1 # \$t2 = &p1

sw \$t1, 0(\$t2) # p1 = &b

p1 apunta a b

Desreferència de punters

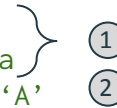
- Els punters serveixen per accedir a les variables apuntades
- La desreferència consisteix en accedir a l'adreça de memòria apuntada per el punter
- A C s'utilitza l'operador unari *
 - *p1: contingut de l'adreça de memòria apuntada per p1
 - No confondre amb la declaració de punters

```
// Codi C
char a = 'A';
char *p1 = &a;

void func() {
    char tmp = *p1; // tmp = 'A'
}
```

```
# Assemblador
        .data
a:      .byte 'A'
p1:     .word a

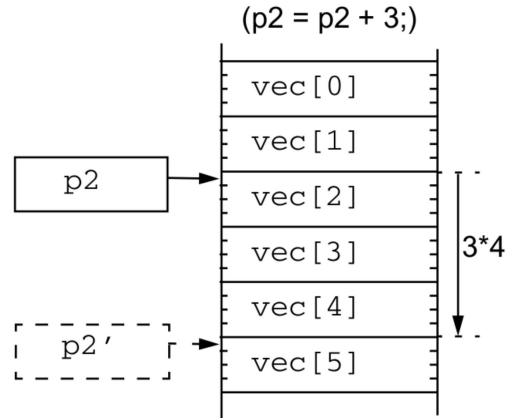
func:   la $t0, p1      # $t0 = &p1
        lw $t1, 0($t0)  # $t1 = p1 = &a
        lb $t2, 0($t1)  # tmp = *p1 = 'A'
```



- ① Carreguem el valor del punter a un registre (és una adreça de memòria)
- ② Desreferenciar el punter (s'ha de fer servir el tipus correcte!)

Aritmètica de punters

- Suma de un punter p més un enter N
- Dona com a resultat un altre punter q del mateix tipus
 - $q = p + N$
- q apunta a un altre enter situat N elements més endavant
- Exemple
 - `int *p2 = &vec[2];`



Exemples aritmética de punters

- Moure un punter $N = 3$ elements

```
// Codi C
char *p1;
int *p2;
long long *p3;
```

```
p1 = p1 + 3;
p2 = p2 + 3;
p3 = p3 + 3;
```

```
# Assemblador MIPS
p1:  .word 0
p2:  .word 0
p3:  .word 0
```

```
addiu $t1, $t1, 3    # variable p1 guardada a $t1
addiu $t2, $t2, 12   # variable p2 guardada a $t2
addiu $t3, $t3, 24   # variable p3 guardada a $t3
```

- Multiplicar N per la mida dels elements apuntats

Exercici

- Donades les següents declaracions globals en C
 - `int dada = 0x22222222;`
 - `int *pdada;`
- Tradueix a MIPS les següents sentències en C
 1. `pdada = &dada;`
 2. `pdada = pdada + 1;`
 3. `*pdada = *pdada + 1;`
 4. `dada = dada - 1;`

Exercici

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 assembleador MIPS les següents sentències:

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

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