Tema 2
Assemblador i tipus de dades bàsics

Estructura de Computadors (EC)

Modes d’adreçament

- Manera en la que s’especifica un operand en una instrucció
- MIPS suporta cinc modes d’adreçament
  - **Mode registre**
  - **Mode immediat**
  - **Mode memòria**
  - Mode pseudodirecte
  - Mode relatiu al PC

Operands en mode registre

- L'operand es troba en un registre
- L'instrucció especifica l'identificador del registre
  - suma: `addu rd, rs, rt  # rd = rs + rt`
  - resta: `subu rd, rs, rt  # rd = rs - rt`

- L'arquitectura MIPS té 32 registres de 32 bits

Registres

Número	Nom	Descripció
$0	$zero	Conté el valor 0 (read-only)
$1	$at	Registre temporal per pseudoinstruccions
$2 - $3	$v0 - $v1	Resultats de subrutines
$4 - $7	$a0 - $a3	Paràmetres de subrutines
$8 - $15	**$t0 - $t7**	**Temporals**
$16 - $23	**$s0 - $s7**	**Segurs (es preserven al cridar a una subrutina)**
$24 - $25	**$t8 - $t9**	**Temporals**
$26 - $27	$k0 - $k1	Reservats per el SO
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Return address

Operands en mode immediat

- L'operand es codifica a la propia instrucció
- Valors de 16 bits en complement a 2 (Ca2)
  - Com es converteix a valors de 32 bits?
    - Extensió de signe
    - Extensió de zeros

```
addiu rt, rs, imm16          # rt = rs + SignExt(imm16)

ori rt, rs, imm16            # rt = rs OR ZeroExt(imm16)

lui rt, imm16                # rt31..16 = imm16

                             # rt15..0 = 0x0000
```

Exercici

- Donada la següent sentencia en C

```
f = (g + h) - (i - 100)
```

- Suposant que `f`, `g`, `h`, `i` son variables locals enteres emmagatzemades als registres `$t0`, `$t1`, `$t2`, `$t3` respectivament. Tradueix la sentencia a llenguatge assemblador MIPS.

Operands en mode memòria

- Només les instruccions de tipus *load* o *store* admeten un operand que resideixi en memòria.
  - MIPS és una arquitectura load-store
  - Loads per llegir de memòria
  - Stores per escriure a memòria
- S'han de llegir les dades de memòria per poder utilitzar-les en instruccions aritmetico-lògiques.
- Lectura/escriptura de paraules (words) en memòria

`lw rt, off16(rs)`	$rt = M_W[rs + SignExt(off16)]$	Load word
`sw rt, off16(rs)`	$M_W[rs + SignExt(off16)] = rt$	Store word

Exercici Load

- Traduiu a MIPS la següent assignació de valores enters (words) en C, suposant que `g` i `h` ocupen `$t1`, `$t2` i que la direcció base de `A` està a `$t3`

```
g = h + A[8];  # A és un vector de int
```

Exercici Load

- Traduiu a MIPS la següent assignació de valores enters (words) en C, suposant que `g` i `h` ocupen `$t1`, `$t2` i que la direcció base de `A` està a `$t3`

```
g = h + A[8];
```

```
lw $t1, 32($t3)         # $t1 = A[8]
addu $t1, $t2, $t1# g = h + $t1
```

Exercici Store

- Traduiu a MIPS la següent assignació de valores enters (words) en C, suposant que `h` ocupa `$t2` i que la direcció base de `A` està a `$t3`

```
A[12] = h + A[8]; # A és un vector de int
```

Exercici Store

- Traduiu a MIPS la següent assignació de valores enters (words) en C, suposant que h ocupa `$t2` i que la direcció base de `A` està a `$t3`

```
A[12] = h + A[8];
```

```
lw $t1, 32($t3)         # $t1 = A[8]
addu $t1, $t2, $t1# $t1 = h + $t1
```

Exercici Store

- Traduiu a MIPS la següent assignació de valores enters (words) en C, suposant que h ocupa `$t2` i que la direcció base de `A` està a `$t3`

```
A[12] = h + A[8];
```

```
lw $t1, 32($t3)        # $t1 = A[8]
addu $t1, $t2, $t1# $t1 = h + $t1
sw $t1, 48($t3)        # A[12] = $t1
```

Accés a halfword o byte - Enters amb signe

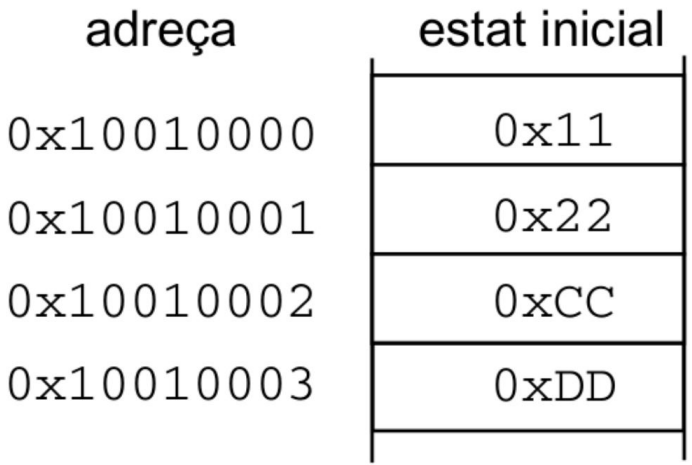
- Load halfword: `lh rt, off16(rs)`
  - Copia un half (2 bytes) de la memòria als 16 bits de menor pes del registre `rt`
  - Realitza extensió de signe (exten el bit 15)
- Store halfword: `sh rt, off16(rs)`
  - Copia a memòria els 2 bytes de menor pes de `rt`

- Load byte: `lb rt, off16(rs)`
  - Copia 1 byte de memòria als 8 bits de menor pes del registre `rt`
  - Realitza extensió de signe (exten el bit 7)
- Store byte: `sb rt, off16(rs)`
  - Copia a memòria el byte de menor pes del registre `rt`

Exercici

- Suposant que `$t2` conte la direcció `0x10010000` i que el contingut de la memòria és el que es mostra a la part dreta, indica el resultat de cada instrucció i el contingut final de la memòria

1. `lb $t1, 1($t2)`
2. `lb $t1, 2($t2)`
3. `lh $t1, 0($t2)`
4. `lh $t1, 2($t2)`
5. `sb $t1, 1($t2)`



Accés a halfword o byte - Naturals

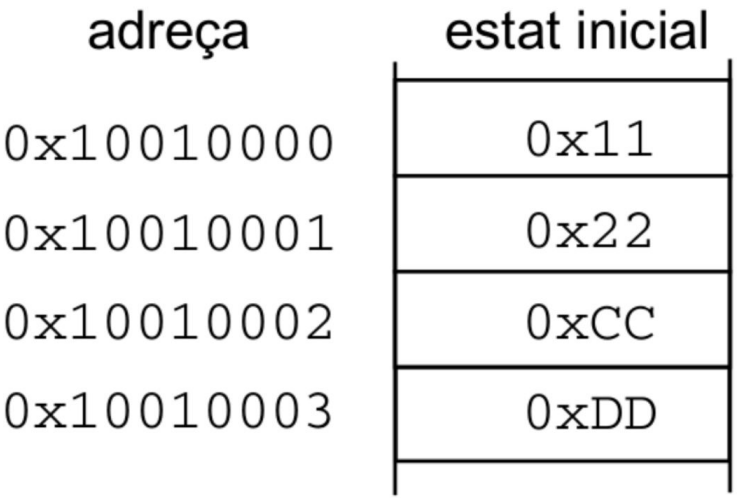
- Load halfword unsigned: `lhu rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memòria als 16 bits de menor pes del registre `rt`
  - Realitza extensió de zeros

- Load byte unsigned: `lbu rt, off16(rs)`
  - Copia 1 byte de memòria als 8 bits de menor pes del registre `rt`
  - Realitza extensió de zeros

Exercici

- Suposant que `$t2` conte la direcció `0x10010000` i que el contingut de la memòria és el que es mostra a la part dreta, indica el resultat de cada instrucció

1. `lbu $t1, 2($t2)`
2. `lhu $t1, 2($t2)`



Accés a doble paraula (dword)

- Com s'accedeix a una dada de 64 bits (long long) en la arquitectura MIPS32?

```
    .data
x:  .dword 0x7766554433221100

    .text
main:
    # $t2 conté l'adreça de memòria de x
    lw $t0, 0($t2)    # llegeix part baixa 0x33221100
    lw $t1, 4($t2)    # llegeix part alta 0x77665544
```

Restriccions d'alineament

- Les adreces utilitzades a les instruccions `lw i sw` han de ser múltiples de 4
- Les adreces utilitzades a les instruccions `lh, lhu i sh` han de ser múltiples de 2

- En cas contrari es produeix una excepció per adreça no alineada i el programa finalitza

Pseudoinstruccions o macros

- Defineixen operacions comuns per les que no existeix una instrucció MIPS
- Faciliten el desenvolupament, la lectura i la depuració del codi
- En el moment de ser traduïda a llenguatge màquina es tradueix per una o varies instruccions MIPS

```
move $t1, $t2                 # addu $t1, $t2, $zero

li $t1, 100                   # addiu $t1, $zero, 100

li $t1, 0x11223344            # lui $at, 0x1122
                              # ori $t1, $at, 0x3344
```

Pseudoinstruccions o macros

- Defineixen operacions comuns per les que no existeix una instrucció MIPS
- Faciliten el desenvolupament, la lectura i la depuració del codi
- En el moment de ser traduïda a llenguatge màquina es tradueix per una o varies instruccions MIPS

```
        .data
y:      .word 42        # Adreça de y = 0x10010024

        .text
        la $t0, y       # lui $at, 0x1001
                        # ori $t0, $at, 0x0024
```

Exercici

- Tradueix a MIPS el següent programa en C

```
short v[3] = {-31, 43, 77};
int sum;

int main(void){
    sum = v[0] + v[1] + v[2] - 91;
}
```

Representació de dades

Representació de naturals

- Declaració en C de variables naturals
  - unsigned char, unsigned short, unsigned int, unsigned long long
- Codificar un natural en binari
  - Algorisme de divisions successives
- De codificació binaria a decimal

$$x = \sum_{i=0}^{n-1} X_i \cdot 2^i$$

- Rang de representació per ***n*** bits: `[0, 2^n-1]`

Formats de representació d'enters

- Estudiarem 4 formats per enters a EC
  - **Complement a 2 (Ca2)**
  - Complement a 1 (Ca1)
  - Signe i Magnitud
  - Excés

- Regla de representació
  - Indica com codificar un número enter en binari
- Regla d'interpretació
  - Indica com convertir una codificació binaria a número enter en base 10
- Rang de representació
  - Rang de números enters que es poden codificar usant N bits

Complement a 2 (Ca2)

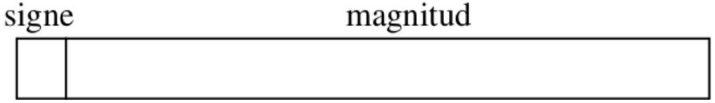
- Regla de representació per ***n*** bits
  - Si és positiu: representar com un natural
  - Si és negatiu: sumar 2^n i representar com un natural
- Regla d'interpretació per ***n*** bits
  - Interpretar com un natural
  - i si el bit de major pes és 1 (negatiu): restar 2^n
- Regla de canvi de signe
  - Complementar bits i sumar 1
- Rang de representació per ***n*** bits: $[-2^{n-1},\ 2^{n-1}-1]$
- Rang no és simètric
- Mateix circuit sumador per naturals i enters en Ca2

Complement a 1 (Ca1)

- Regla de representació per ***n*** bits
  - Si és positiu: representar com un natural
  - Si és negatiu: sumar $2^n - 1$ i representar com un natural
- Regla d'interpretació per ***n*** bits
  - Interpretar com un natural
  - Si el bit de major pes és 1 (negatiu): restar $2^n - 1$
- Regla de canvi de signe
  - Complementar bits
- Rang de representació per ***n*** bits: $[-2^{n-1}+1,\ 2^{n-1}-1]$
- Dos representacions per el zero
- Requereix un circuit sumador diferent al dels naturals

Signe i Magnitud (SiM)

- Regla de representació per ***n*** bits
  - Codificar el signe al bit de major pes (0 positiu, 1 negatiu)
  - Codificar el valor absolut (magnitud) als $n-1$ bits restants



- Regla d'interpretació per ***n*** bits
  - El bit de major pes indica el signe (0 positiu, 1 negatiu)
  - Els $n-1$ bits restants indiquen el valor absolut
- Regla de canvi de signe
  - Complementar el bit de major pes
- Rang de representació per ***n*** bits: $[-2^{n-1}+1,\ 2^{n-1}-1]$
- Dos representacions per el zero
- Requereix un circuit sumador diferent al dels naturals

Excés K

- Normalment $K = 2^{n-1}-1$ per equilibrar positius i negatius
- Regla de representació per ***n*** bits
  - Sumar *K* i representar el resultat com un natural
- Regla de interpretació per ***n*** bits
  - Interpretar com un natural i restar *K*
- Rang de representació per ***n*** bits: $[-K,\ 2^{n}-K-1]$
- El bit de major pes no indica el signe
- Requereix un circuit sumador diferent al dels naturals

Exercici

- Converteix els següents números enters en base 10 a els formats de representació Ca2, Ca1, Signe i Magnitud i Excés 127 utilitzant 8 bits
  - -78
  - 125
  - 0
  - -1
  - -127