Tema 2. Ensamblador MIPS y tipos de datos básicos

Modos de direccionamiento

- Forma en la que se especifica un operando en una instrucción
- MIPS soporta cinco modos de direccionamiento:
  - Modo registro
  - Modo inmediato
  - Modo memoria
  - Modo pseudodirecto
  - Modo relativo al PC

Operandos en modo registro

- El operando reside en un registro
- La instrucción especifica el identificador del registro
- Suma: `addu rd, rs, rt`
- Resta: `subu rd, rs, rt`
- MIPS incluye 32 registros de 32 bits

Registros

Número	Nombre	Descripción
$0	$zero	Contiene el valor 0 (solo lectura)
$1	$at	Registro temporal para pseudoinstrucciones
$2-$3	$v0-$v1	Resultados de subrutinas
$4-$7	$a0-$a3	Parametros de una subrutina
$8-$15	$t0-$t7	Temporales
$16-$23	$s0-$s7	Seguros (se preservan al llamar a una subrutina)
$24-$25	$t8-$t9	Temporales
$26-$27	$k0-$k1	Reservados para el SO
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Return address

Operandos en modo inmediato

- El operando se codifica en la propia instrucción
- Valor de 16 bits en Ca2
- ¿Cómo se convierte en un valor de 32 bits?
  - Extensión de signo
  - Extensión de ceros

`addiu rt, rs, imm16`	$rt = rs + SignExt(imm16)$
`lui rt, imm16`	$rt_{31..16} = imm16$ $rt_{15..0} = 0x0000$
`ori rt, rs, imm16`	$rt = rs\ OR\ ZeroExt(imm16)$

Ejercicio

- Dada la siguiente sentencia en C:

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

- Suponiendo que `f`, `g`, `h`, `i` son variables locales enteras almacenadas en los registros `$t0`, `$t1`, `$t2`, `$t3` respectivamente, ¿cuál será su traducción a lenguaje ensamblador MIPS?

Operandos en modo memoria

- Solo las instrucciones de tipo *load* y *store* admiten un operando que resida en memoria
  - MIPS es una arquitectura load-store
- Hay que cargar los datos de memoria en registros para poder utilizarlos en instrucciones aritmetico-lógicas
- Lectura/escritura de palabras en memoria:

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

Ejemplo

- Traducir a MIPS la siguiente asignación de valores enteros (words) en C, suponiendo que `g` y `h` ocupan $t1, $t2 y que la dirección base de `A` está en $t3:

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

Ejemplo

- Traducir a MIPS la siguiente asignación de valores enteros (words) en C, suponiendo que `g` y `h` ocupan $t1, $t2 y que la dirección base de `A` está en $t3:

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

lw  $t0, 32($t3)     # $t0 = A[8]
addu $t1, $t2, $t0   # g = h + $t0;
```

Ejercicio

- Traduce a MIPS la siguiente asignación de valores enteros (words) en C, suponiendo que `h` ocupa $t2 y que la dirección base de `A` está en $t3:

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

Acceso a halfword o byte - Enteros con signo

- Load halfword: `lh rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memoria a los 16 bits de menor peso del registro `rt`
  - Realiza extensión de signo (extiende el bit 15)

Acceso a halfword o byte - Enteros con signo

- Load halfword: `lh rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memoria a los 16 bits de menor peso del registro `rt`
  - Realiza extensión de signo (extiende el bit 15)
- Store halfword: `sh rt, off16(rs)`
  - Copia a memoria los 2 bytes de menor peso de `rt`

Acceso a halfword o byte - Enteros con signo

- Load halfword: `lh rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memoria a los 16 bits de menor peso del registro `rt`
  - Realiza extensión de signo (extiende el bit 15)
- Store halfword: `sh rt, off16(rs)`
  - Copia a memoria los 2 bytes de menor peso de `rt`
- Load byte: `lb rt, off16(rs)`
  - Copia 1 byte de memoria a los 8 bits de menor peso del registro `rt`
  - Realiza extensión de signo (extiendo el bit 7)

Acceso a halfword o byte - Enteros con signo

- Load halfword: `lh rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memoria a los 16 bits de menor peso del registro `rt`
  - Realiza extensión de signo (extiende el bit 15)
- Store halfword: `sh rt, off16(rs)`
  - Copia a memoria los 2 bytes de menor peso de `rt`
- Load byte: `lb rt, off16(rs)`
  - Copia 1 byte de memoria a los 8 bits de menor peso del registro `rt`
  - Realiza extensión de signo (extiendo el bit 7)
- Store byte: `sb rt, off16(rs)`
  - Copia a memoria el byte de menor peso del registro `rt`

Ejercicio

- Suponiendo que $t2 contiene la dirección 0x10010000 y que el contenido de la memoria es el que se muestra en la parte derecha, indica el resultado de cada instrucción y el contenido final de la memoria:

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)`

adreça	estat inicial
0x10010000	0x11
0x10010001	0x22
0x10010002	0xCC
0x10010003	0xDD

Acceso a halfword o byte - Naturales

- Load halfword unsigned: `lhu rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memoria a los 16 bits de menor peso del registro `rt`
  - Realiza extensión de ceros

Acceso a halfword o byte - Naturales

- Load halfword unsigned: `lhu rt, off16(rs)`
  - Copia un halfword (2 bytes) de la memoria a los 16 bits de menor peso del registro `rt`
  - Realiza extensión de ceros
- Load byte unsigned: `lbu rt, off16(rs)`
  - Copia 1 byte de memoria a los 8 bits de menor peso del registro `rt`
  - Realiza extensión de ceros

Ejercicio

- Suponiendo que $t2 contiene la dirección `0x10010000` y que el contenido de la memoria es el que se muestra en la parte derecha, indica el resultado de cada instrucción:

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

adreça	estat inicial
0x10010000	0x11
0x10010001	0x22
0x10010002	0xCC
0x10010003	0xDD

Acceso a doble palabra (dword)

- ¿Cómo se accede a un dato de 64 bits (long long) en la arquitectura MIPS de 32 bits?

Acceso a doble palabra (dword)

- ¿Cómo se accede a un dato de 64 bits (long long) en la arquitectura MIPS de 32 bits?

```
    .data
x:  .dword 0x7766554433221100

    .text
main:
    # $t2 contiene la direccion de memoria de x
    lw $t0, 0($t2)
    lw $t1, 4($t2)
```

Restricción de alineación

- Las direcciones utilizadas en las instrucciones `lw` y `sw` han de ser múltiplos de 4
- Las direcciones utilizadas en las instrucciones `lh`, `lhu` y `sh` han de ser múltiplos de 2
- En caso contrario se produce una excepción por dirección no alineada y el programa termina

Pseudoinstrucciones o macros

- Simplifican operaciones comunes para las que no existe una instrucción en MIPS
- Facilitan el desarrollo, la lectura y la depuración del código
- En el momento de ser ensamblada se traduce a una o varias instrucciones MIPS

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

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

li $t1, 0x0030D900     # lui $at, 0x0030
                       # ori $t1, $at, 0xD900
```

Pseudoinstrucciones o macros

- Simplifican operaciones comunes para las que no existe una instrucción en MIPS
- Facilitan el desarrollo, la lectura y la depuración del código
- En el momento de ser ensamblada se traduce a una o varias instrucciones MIPS

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

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

Ejercicio

- Traduce a MIPS el siguiente programa en C:

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

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

Formatos de representación de enteros

- Estudiaremos 4 formatos de enteros en EC:
  - Complemento a 2
  - Complemento a 1
  - Signo y Magnitud
  - Exceso

Formatos de representación de enteros

- Estudiaremos 4 formatos de enteros en EC:
  - Complemento a 2
  - Complemento a 1
  - Signo y Magnitud
  - Exceso
- Regla de representación
  - Indica cómo codificar un número entero en binario

Formatos de representación de enteros

- Estudiaremos 4 formatos de enteros en EC:
  - Complemento a 2
  - Complemento a 1
  - Signo y Magnitud
  - Exceso
- Regla de representación
  - Indica cómo codificar un número entero en binario
- Regla de interpretación
  - Indica cómo convertir una codificación binaria a número entero en base 10

Formatos de representación de enteros

- Estudiaremos 4 formatos de enteros en EC:
  - Complemento a 2
  - Complemento a 1
  - Signo y Magnitud
  - Exceso
- Regla de representación
  - Indica cómo codificar un número entero en binario
- Regla de interpretación
  - Indica cómo convertir una codificación binaria a número entero en base 10
- Rango de representación
  - Números enteros que se pueden codificar usando N bits

Complemento a 2 (Ca2)

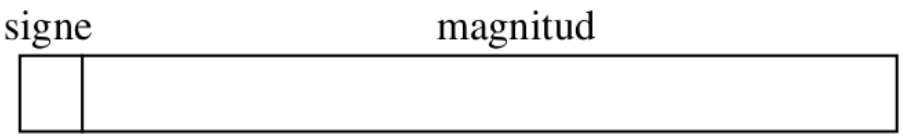
- Regla de representación para n bits:
  - Si es positivo: representar como natural
  - Si es negativo: sumar 2^n y representar como natural
- Regla de interpretación para n bits:
  - Interpretar como natural
  - Si el bit de mayor peso es 1 (negativo): restar 2^n
- Regla de cambio de signo
  - Complemetar bits y sumar 1
- Rango de representación con n bits: $[-2^{n-1}, 2^{n-1} - 1]$
- Rango NO simétrico
- Mismo circuito sumador para naturales y enteros en Ca2

Complemento a 1 (Ca1)

- Regla de representación para n bits:
  - Si es positivo: representar como natural
  - Si es negativo: sumar $2^n - 1$ y representar como natural
- Regla de interpretación para n bits:
  - Interpretar como natural
  - Si el bit de mayor peso es 1 (negativo): restar $2^n - 1$
- Regla de cambio de signo
  - Complemetar bits
- Rango de representación con n bits: $[-2^{n-1} + 1, 2^{n-1} - 1]$
- Dos representaciones para el cero
- Requiere un circuito de suma diferente al de los naturales

Signo y Magnitud (SyM)

- Regla de representación para n bits:
  - Codificar el signo en el bit de mayor peso (0 positivo, 1 negativo)
  - Codificar el valor absoluto (magnitud) en los $n-1$ bits restantes
- Regla de interpretación para n bits:
  - El bit de mayor peso indica el signo (0 positivo, 1 negativo)
  - Los $n-1$ bits restantes indican el valor absoluto
- Regla de cambio de signo
  - Complemetar el bit de mayor peso
- Dos representaciones para el cero
- Circuito específico para la suma



Exceso K

- Regla de representación para n bits:
  - Sumar K y representar el resultado como natural
- Regla de interpretación para n bits:
  - Interpretar como natural y restar K
- Rango de representación con n bits: $[-K, 2^n - K - 1]$
- Normalmente K=$2^{n-1} - 1$ para equilibrar positivos y negativos
- El bit de mayor peso NO indica el signo
- La comparación se puede hacer con el mismo circuito que compara naturales

Ejercicio

- Convierte los siguientes números enteros en base 10 a los formatos de representación Ca1, Ca2, Signo y Magnitud y Exceso 127 utilizando 8 bits en todos los casos:
  - -78
  - 125
  - 0
  - -1