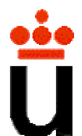
ARQUITECTURA DE COMPUTADORES

Tema 2: Lenguaje Máquina, Instrucciones y Modos de Direccionamiento.

Licesio J. Rodríguez-Aragón

Programa

1. Introducción
   - Lenguajes de Programación: Alto y Bajo nivel
   - Lenguaje Ensamblador
   - Lenguaje Máquina
2. Instrucciones
   - Repertorio de Instrucciones
   - Propiedades
   - Procesadores MIPS
3. Formato de Instrucción
4. Modos de Direccionamiento
   - Directo, Indirecto, Relativo a PC, Inmediato, Pseudodirecto, Implicito.
5. Tipos de Instrucciones
   - Tipo R
   - Tipo I
   - Tipo J
6. Ejemplos
7. Conclusiones



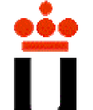
1. Introducción

➲ Los computadores ejecutan programas.

➲ Programa: secuencia de operaciones conducentes a resolver un problema determinado.

➲ Características de los programas
- Están compuestos por secuencias de instrucciones o sentencias.
- Se escriben utilizando una notación formal conveniente.
- Pueden ser escritos por personas (programadores), o bien pueden ser generados automáticamente mediante una herramienta adecuada.
- Un programa en ejecución se encuentra cargado en memoria principal.

➲ Lenguaje de programación: una notación formal para describir algoritmos o funciones que serán ejecutadas por un computador.



Lenguajes de alto nivel y lenguajes de bajo nivel

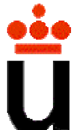
➲ La codificación de programas en binario es conveniente y natural para la circuitería del computador, pero es difícil para un programador humano.

➲ El lenguaje ensamblador surgió para facilitar la escritura de programas de computador.
- Es un lenguaje simbólico que da nombres a las instrucciones de máquina, y permite dar nombres a posiciones de memoria que contienen instrucciones o datos.

➲ Los lenguajes de alto nivel facilitan la tarea de los programadores, ya que se encuentran más próximos a la forma de pensar de los humanos.
- Control estructurado de flujo.
- Comprobación de tipos.

➲ La programación en lenguajes de alto nivel es más productiva, ya que los programas son más cortos (en cuanto a líneas de código).

➲ Hoy en día la práctica totalidad de los programadores trabaja utilizando lenguajes de alto nivel.



Tipos de lenguajes de programación

➲ Lenguajes de bajo nivel: cercanos a la arquitectura de la máquina.

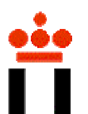
➲ Lenguajes de alto nivel: cercanos a la forma de pensar del programador.

➲ Lenguaje máquina: el único que la circuitería de la máquina es capaz de interpretar.

- Sus instrucciones se encuentran codificadas en binario.

➲ Lenguajes simbólicos: no son directamente interpretables por la circuitería de la máquina.

- Se codifican mediante símbolos alfanuméricos, de puntuación, paréntesis, separadores, etc.



Lenguajes de alto nivel

➲ Son métodos convenientes y sencillos de describir las estructuras de información y las secuencias de acciones precisas para ejecutar tareas concretas.

➲ Los lenguajes de alto nivel se acercan de alguna manera a la forma en que las personas resolvemos los problemas.

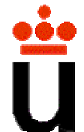
➲ Características:

- Posibilidad de traducción automática a lenguaje máquina.
- Independencia de la arquitectura del computador.
- Transportabilidad entre diferentes computadores.

➲ Algunos tipos de lenguajes de alto nivel:

- Lenguajes de propósito general.
- Lenguajes de propósito específico (comerciales, científicos, educativos, etc).
- Lenguajes de diseño de sistemas de información.

➲ Los lenguajes de alto nivel son lenguajes simbólicos no comprensibles directamente por la circuitería del computador.



Lenguajes de bajo nivel

➲ Se encuentran totalmente vinculados a la estructura del computador.

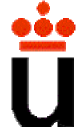
➲ Están diseñados para sacar el máximo partido de las características físicas del computador.

➲ Características:

- Dependencia absoluta de la arquitectura del computador.
- Imposibilidad de transportar programas entre distintas máquinas, salvo que sean de la misma familia o compatibles.
- Instrucciones poco potentes.
- Programas muy largos.
- Códigos de operación, datos y referencias en binario.

➲ Tipos:

- Lenguaje máquina.
  - Códigos de operación, datos y referencias en binario.
  - Directamente interpretable y ejecutable por la circuitería del computador.
- Lenguaje ensamblador.



Lenguaje ensamblador

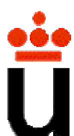
➲ El lenguaje ensamblador (o lenguaje de ensamble, *assembly language*) es la representación simbólica de la codificación binaria de un computador.

- Códigos de operación representados mediante códigos nemotécnicos.
- Datos y referencias codificadas mediante nombres simbólicos (símbolos o etiquetas).

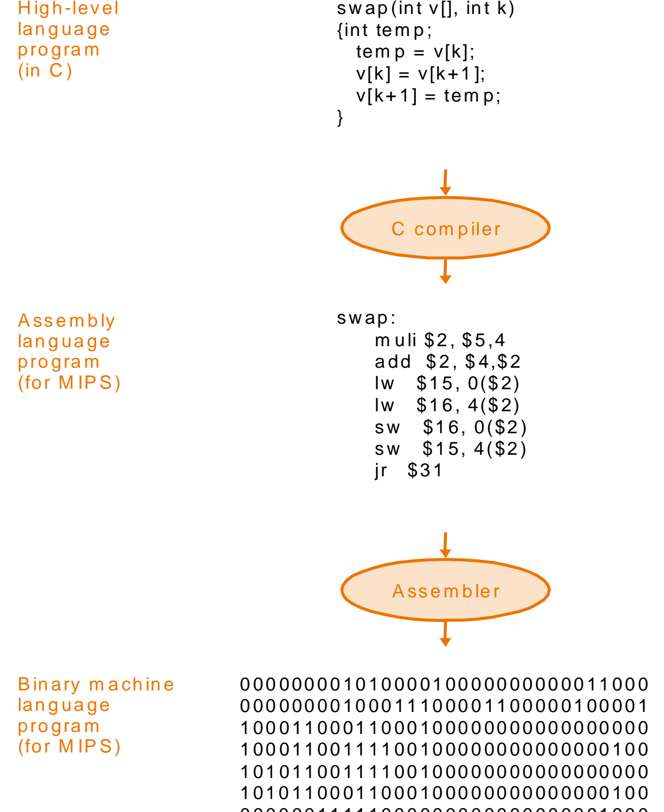
➲ Existe una correspondencia biunívoca entre las instrucciones de máquina y las instrucciones de un lenguaje ensamblador.

- Cada instrucción ensamblador es una codificación simbólica de una instrucción de máquina.
- Excepción: ensambladores que proporcionan una máquina virtual con pseudoinstrucciones.

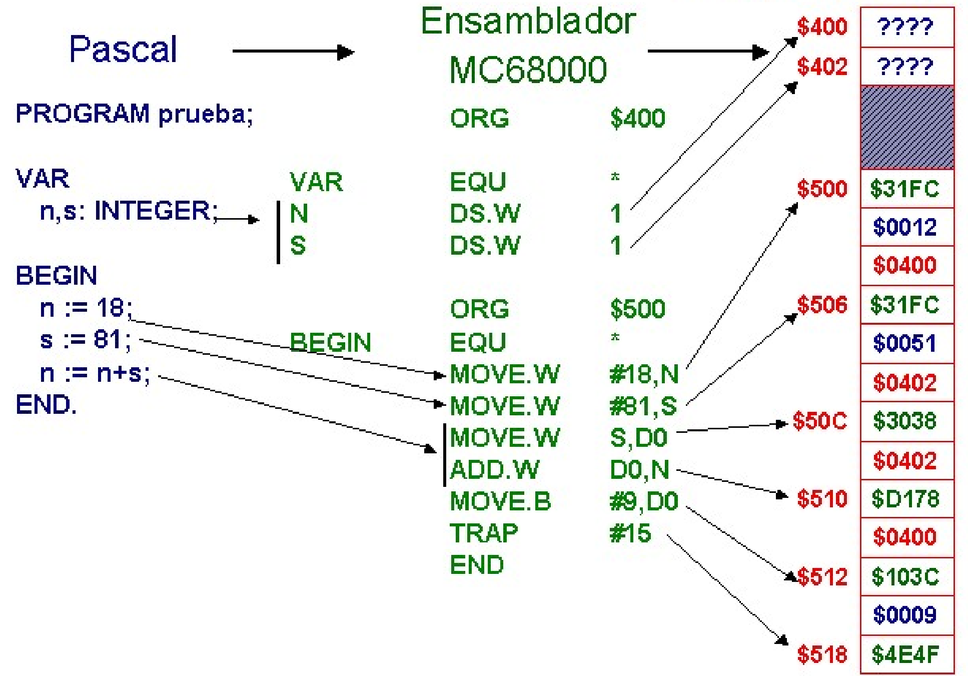
➲ El lenguaje ensamblador debe ser traducido a lenguaje máquina para poder ser interpretado y ejecutado directamente por el computador.

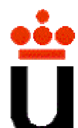


Lenguajes de alto nivel y bajo nivel



Lenguajes de alto nivel y bajo nivel



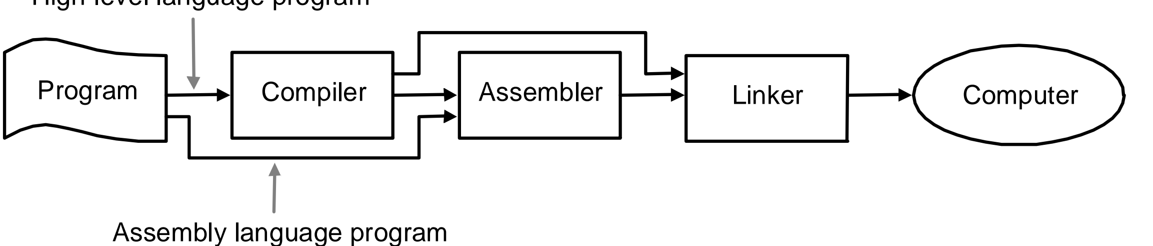


Lenguajes de alto nivel y bajo nivel

➲ La circuitería del computador tan solo comprende los programas escritos en lenguaje máquina. ⚠

➲ Los programas escritos en lenguajes simbólicos deben ser traducidos a lenguaje máquina antes de ser ejecutados.

- •Compilador (*compiler*): traductor de lenguaje de alto nivel a lenguaje ensamblador.
- •Hoy día los compiladores pueden traducir los programas directamente a lenguaje máquina.
- •Ensamblador (*assembler*): traductor de lenguaje ensamblador a lenguaje máquina.
- •Montador (*linker*): crea el código máquina ejecutable final.
- •Cargador (*loader*): carga el código ejecutable en memoria y lo prepara para su ejecución.



Terminología

➲ Código (programa) fuente: código escrito por el programador.

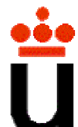
- •Puede estar escrito en cualquier lenguaje: alto nivel, ensamblador, código máquina (inusual).
- •Puede contener errores sintácticos porque el programador haya escrito mal el programa.

➲ Código (programa) objeto: código obtenido al traducir el código a lenguaje máquina.

- •No contiene errores sintácticos.
- •A veces no es directamente ejecutable.

➲ Código (programa) ejecutable: listo para ser ejecutado en el computador.

- •Puede contener errores lógicos debidos a que el programa no esté bien diseñado y no realice correctamente la función para la cual se creó.



Lenguajes de alto nivel y bajo nivel

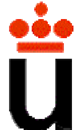
➲ Factores que miden la calidad de los programas ejecutables:
- Tamaño en número de palabras de memoria.
- Velocidad.

➲ Tradicionalmente los compiladores generaban código máquina de inferior calidad que el que podían escribir programadores humanos.
- Las memorias son mucho mayores hoy en día: el tamaño ha dejado de ser crítico.

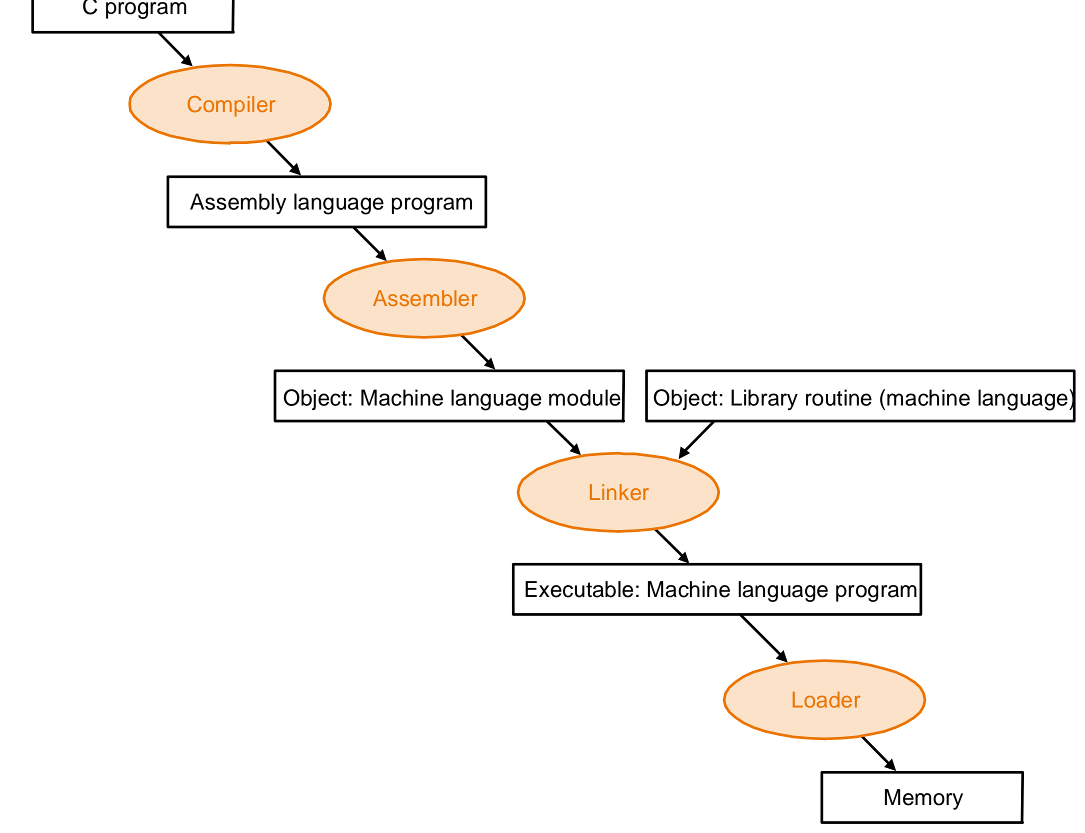
➲ Los compiladores proporcionan hoy en día un código máquina de alta calidad pequeño y rápido, haciendo poco atractiva la programación en ensamblador.
- Los programadores de ensamblador siguen teniendo ventaja en cuanto a que disponen de un mayor conocimiento global del programa que les permite realizar determinadas optimizaciones del código que resultan muy difíciles para los compiladores.

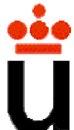
➲ Puede ser recomendable programar en ensamblador cuando la velocidad del programa y su tamaño sean críticos.
- Caso especial: computadores empotrados (*embedded computers*). ⚠

➲ Solución mixta:
- Programar en alto nivel la mayor parte del código.
- Programar en ensamblador las partes críticas en cuanto a velocidad.
- Programar en ensamblador los sistemas con un tamaño de memoria muy reducido.



Jerarquía de traducción

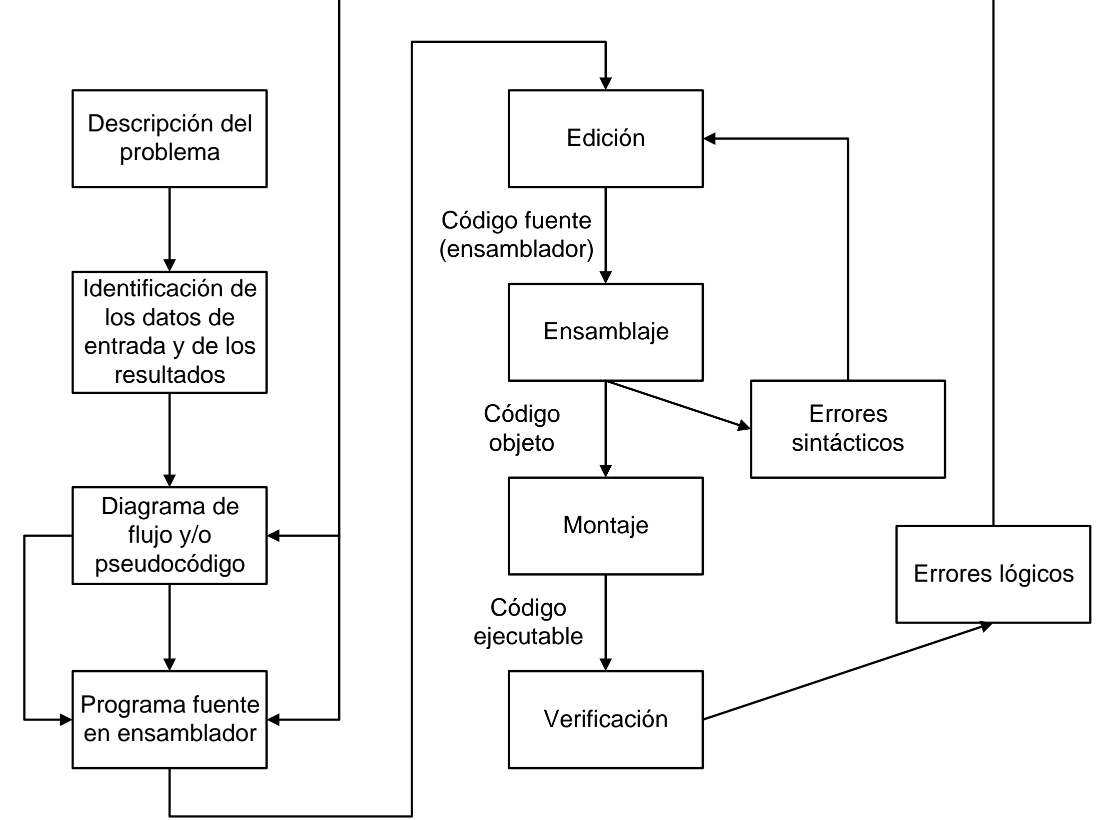


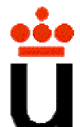


Desarrollo de programas en ensamblador: fases

1. Especificación del problema.
2. Elaboración del algoritmo de la solución.
3. Diseño del diagrama de flujo.
4. Codificación en ensamblador.
5. Edición del programa fuente.
6. Traducción del programa fuente a código máquina.
7. Montaje del programa ejecutable.
8. Carga y ejecución del programa.
9. Verificación del funcionamiento del programa.

Desarrollo de programas en ensamblador: fases





Desarrollo de programas en ensamblador: fases

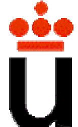
Traducción del programa fuente a código máquina.

➲ Ensamblador
- Traductor de lenguaje ensamblador a lenguaje máquina.
- Genera un fichero con el código objeto equivalente al código fuente completo, junto con información necesaria para el montaje.

➲ Compilador
- Traductor de lenguaje de alto nivel a ensamblador.
- Hoy todos los compiladores traducen directamente a lenguaje máquina.
- En tal caso generan un fichero con el código objeto equivalente al código fuente completo, junto con información necesaria para el montaje.

➲ Si el código fuente contiene errores sintácticos, no se genera código objeto.

➲ Intérprete: traductor de lenguaje de alto nivel a lenguaje máquina.
- Un intérprete traduce y ejecuta las instrucciones del programa fuente una por una, sin generar fichero alguno con código objeto.
- Los intérpretes son propios de los llamados **lenguajes interpretados** (BASIC, LISP, etc).



Desarrollo de programas en ensamblador: fases

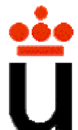
Carga y Ejecución

➲ Consiste en la transferencia del programa ejecutable a la memoria del computador desde el fichero en disco, y en el posterior lanzamiento de su ejecución.

➲El programa ejecutable formado por instrucciones en lenguaje máquina se encuentra almacenado en posiciones consecutivas de memoria.

➲ El Contador de Programa (PC) es un registro que contiene la dirección de la posición de memoria que contiene la instrucción que va a ser ejecutada a continuación.

➲ Herramienta utilizada: cargador.
- Pertenece al sistema operativo.



2. Instrucciones

➲ **Repertorio o juego de instrucciones:** conjunto de instrucciones de máquina que es capaz de ejecutar el computador. Debe ser:

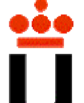
- **Completo:** debe permitir resolver cualquier problema.
- **Eficaz:** los programas deben ser ejecutados en un tiempo razonable.

➲ Está relacionado con:

- El número de registros disponibles.
- El tamaño de los datos.
- Los modos de direccionamiento (maneras de acceder a los datos).

➲ Clasificación de los computadores según su repertorio:

- **RISC:** Reduced Instruction Set Computer (ej.: MIPS, PowerPC).
- **CISC:** Complex Instruction Set Computer (80x86).
- **VLIW:** Very Long Instruction Word (Itanium).



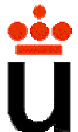
➲ El repertorio de instrucciones debe especificar:

- **Formato de las instrucciones:** tamaño fijo, variable o híbrido.
- **Localización de operandos y resultado**, así como **modos de direccionamiento** de la memoria.

- **Tipos de datos y tamaños:** Enteros (complemento a 2 o BCD, de 8, 16, 32 y 64 bits) y números en coma flotante de 32 y 64 bits (IEEE 754).
- **Operaciones soportadas:** lógicas, aritméticas, etc.
- **Mecanismos de bifurcación:** instrucciones de salto, subrutinas, etc., que modifican el flujo normal de ejecución.

➲ Propiedades de las instrucciones:

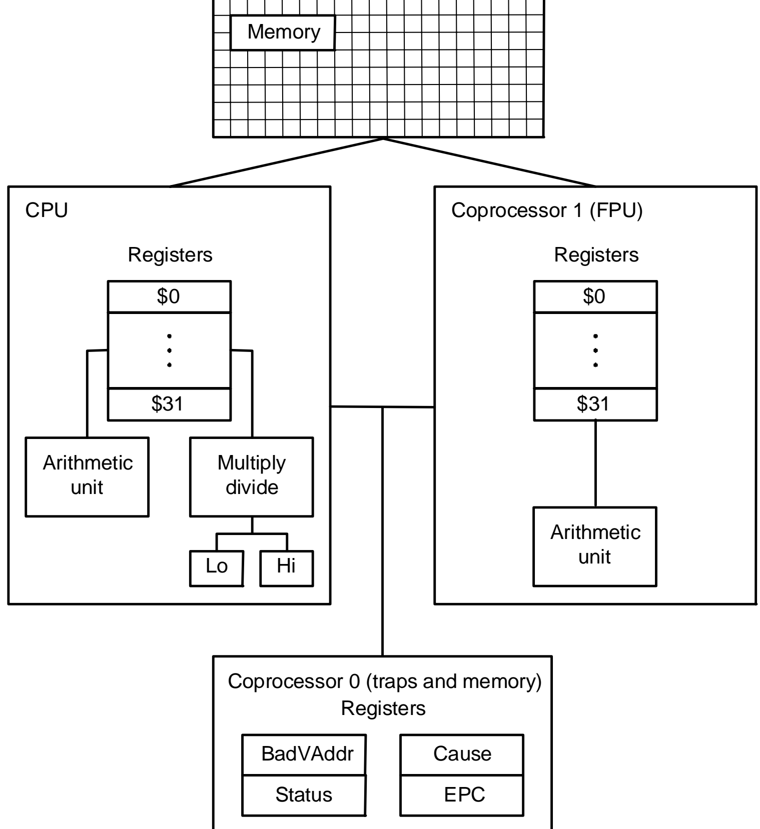
- Realizan una función única y sencilla.
- Emplean un número fijo de operandos en una representación determinada.
- Su codificación binaria es bastante sistemática, o al menos es recomendable que lo sea (con objeto de facilitar su decodificación).
- Son autocontenidas, es decir, contienen toda la información necesaria para su ejecución:
  - Código de operación.
  - Identificación de los operandos.
  - Destino.
  - Ubicación de la siguiente instrucción (normalmente implícita, pues suele ser la siguiente).

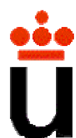


Características principales de MIPS32

- Máquina RISC (computador con repertorio de instrucciones reducido).
- Ancho de palabra y tamaño de los buses: 32 bits.
- Tamaño de los datos en las instrucciones:
  - Bit (*b*): en muy pocas instrucciones.
  - Byte (8 bits, B)
  - Halfword (16 bits, H)
  - Word (32 bits, W)
  - Doubleword (64 bits, D)
- Arquitectura de carga / almacenamiento:
  - Antes de ser utilizado en una instrucción aritmética, todo dato debe ser cargado previamente en un registro de propósito general.
  - Instrucciones aritméticas con 3 operandos de 32 bits en registros.
- Esquema de bus único para memoria y E/S.
- Modos de funcionamiento: usuario, núcleo (*kernel*), supervisor y depuración.

Modelo de programación de MIPS R2000





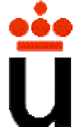
Registros del procesador

31   0
r0 (hardwired to zero)
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15
r16
r17
r18
r19
r20
r21
r22
r23
r24
r25
r26
r27
r28
r29
r30
r31

31   0
HI
LO

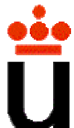
31   0
PC

- 32 registros de propósito general
- Registros propósito específico:
  - HI-LO: para productos y divisiones.
  - PC.
- Ancho de los registros: 32 bits.



Las instrucciones en MIPS

- Máquina RISC (computador con repertorio de instrucciones reducido).
- Arquitectura de carga / almacenamiento.
  - Antes de ser utilizado en una instrucción aritmética, todo dato debe ser cargado previamente en un registro de propósito general.
- Tamaño de las instrucciones: 32 bits = 4 bytes.
- Número de operandos: 0, 1, 2 ó 3.
  - Instrucciones de carga o almacenamiento: 2 operandos, uno en registro y el otro en memoria (de 8, 16 ó 32 bits).
  - Instrucciones aritméticas o lógicas: 3 operandos de 32 bits en registros.
  - Instrucciones aritméticas o lógicas con un inmediato: 3 operandos, dos de 32 bits en registros, y el tercero inmediato de 16 bits con extensión de signo (ceros o unos).
- Máquina virtual: el programador puede utilizar instrucciones y direccionamientos que no están incorporados en el hardware: pseudoinstrucciones.
  - Las pseudoinstrucciones son proporcionadas por el ensamblador, que se encarga de traducirlas a código máquina.



3. Formato de Instrucción en lenguaje máquina

➲ Formato de instrucción: representación en binario de la misma.

- El formato de instrucción especifica el significado de cada uno de los bits que la constituyen.
- Longitud del formato de instrucción: número de bits que lo componen.

➲ La información contenida en el formato de la instrucción es:

- Código de operación (COP).
- Dirección de los operandos (OP1 y OP2).
- Dirección del resultado (RES).
- Dirección de la siguiente instrucción (casi siempre implícita).
- Tipos de representación de los operandos (casi siempre implícitos en el código de operación).
- Modificador (MD): suele completar al CO, y sirve para especificar ciertas particularidades de la instrucción:
  - Tamaño y tipo de los operandos.
  - A veces se usa para distinguir entre operaciones similares.

COP	MD	OP1	OP2	RES

Lenguaje Máquina, Instrucciones y Modos de Direccionamiento

➲ Un computador contendrá instrucciones con diferentes formatos (= no todas las instrucciones del repertorio de un computador tienen el mismo formato).

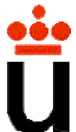
- Un computador dispone de pocos formatos de instrucción diferentes para simplificar su decodificación.
- Los formatos son sistemáticos: campos del mismo tipo suelen ocupar la misma longitud y la misma posición.

➲ El código de operación:

- Permite distinguir entre los distintos formatos de instrucción de un computador.
- Indica la longitud y formato de los operandos (a veces en campos modificadores asociados al mismo).

➲ Las longitudes de los formatos son fracción o múltiplo del tamaño de la palabra del computador.

- Para acortar la longitud de los formatos se utiliza **direccionamiento implícito**: ninguna instrucción, salvo las de salto o bifurcación, contiene la dirección de la siguiente instrucción que se va a ejecutar.

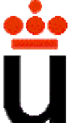


Formato de Instrucciones

➲ MIPS presenta tres formatos básicos de instrucción (32 bits):

- o Tipo R, o instrucciones de registro.
- o Tipo I, instrucciones de transferencia de datos o ramificación condicional.
- o Tipo J, o de salto incondicional.

➲ El compromiso elegido por los diseñadores del MIPS es guardar todas las instrucciones con la misma longitud, por eso se requieren diferentes clases de formatos de instrucción para diferentes clases de instrucciones.



4. Modos de direccionamiento

➲ Los operandos no están contenidos en la instrucción de forma directa por:

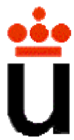
- Ahorro de espacio.
- Empleo de código reubicable y reentrante.
- Hay datos formando estructuras más o menos complejas.

➲ Los operandos pueden ubicarse en los siguientes lugares:

- Dentro de la propia instrucción (operandos “inmediatos”):
  - En el registro de instrucción.
  - En palabras de extensión o ampliación.
- En registros visibles para el programador.
- En variables ubicadas en posiciones de memoria.

➲ Dirección efectiva de un operando: ubicación exacta del mismo.

➲ Los operandos pueden ser referenciados desde la instrucción de múltiples formas, dando lugar a los modos de direccionamiento.

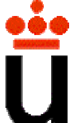


➲ **Modo de direccionamiento:** mecanismo que permite conocer la ubicación de un objeto (dato o instrucción).

➲ Un computador debe disponer de varios modos de direccionamiento.

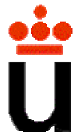
➲ No todos los modos de direccionamiento están implementados en todos los computadores.

➲ Los modos de direccionamiento disponibles están determinados por la arquitectura interna de la máquina y por el repertorio de instrucciones.



Modos de direccionamiento en MIPS

- ➲ Direccionamiento directo a registro.
- ➲ Direccionamiento indirecto a registro con desplazamiento.
- ➲ Direccionamiento relativo a PC con desplazamiento.
- ➲ Direccionamiento inmediato.
- ➲ Direccionamiento pseudodirecto.
- ➲ Direccionamiento implícito.



Direccionamiento directo a registro

➲ El campo tiene 5 bits.

➲ Permitido para operando fuente o destino.

Register addressing

op	rs	rt	rd	. . .	funct

Registers

Register

➲ Notación: $n

➲ Los registros tienen alias, indicando la función para la que están dedicados según los convenios adoptados por programadores y compiladores.

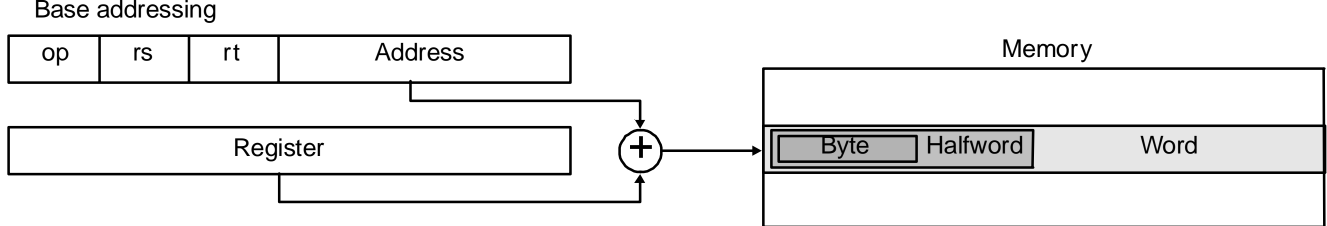
➲ Ejemplo: `add $16, $17, $18` → suma el contenido del registro 17 con el registro 18 y lo almacena en el registro 16.

Direccionamiento indirecto a registro con desplazamiento

➲ Campos:

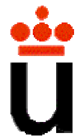
- Registro: 5 bits.
- Desplazamiento: 16 bits.

➲ Permitido para operando fuente en almacenamientos y destino en cargas.



➲ Notación: desplazamiento($n)

➲ Ejemplo: `lw $16, 20($17)` → carga (“load”) en el registro 16 la palabra contenida 20 bytes más allá de la dirección almacenada en el registro 17.



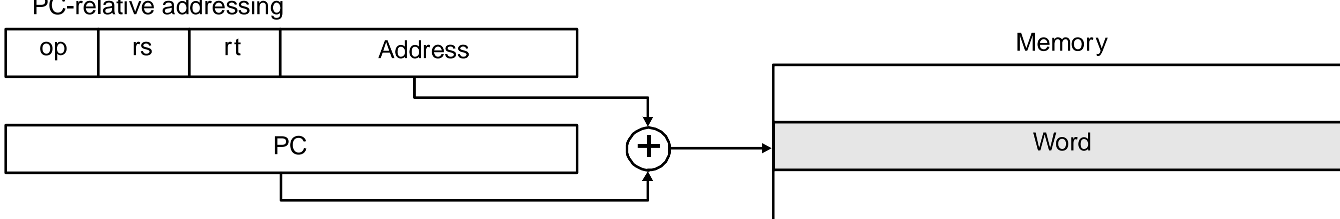
Direccionamiento relativo a PC con desplazamiento

➲ Campos:

- Desplazamiento: 16 bits.

➲ El desplazamiento se alinea a múltiplo de 4 bits y se extiende en signo a 32 bits para calcular la dirección efectiva.

➲ Se utiliza para direcciones en bifurcaciones.



➲ Notación: oculto en etiqueta.

➲ Ejemplo: `bne $16, $17, Fin` → Si el contenido del registro 16 es ≠ del 17, entonces salta a la dirección indicada por la etiqueta “Fin”.

Direccionamiento inmediato

➲ Campos:

- Inmediato de 16 bits.

➲ Los datos inmediatos se extienden a 32 bits.

- En ciertas instrucciones se hace extensión de signo.
- En otras instrucciones se hace extensión con ceros.

➲ También se usa para indicar la longitud de un desplazamiento.

- En este caso el inmediato es de 5 bits.

➲ Permitido sólo para operandos fuente.

Immediate addressing

op	rs	rt	Immediate

➲ Notación: dato (sin prefijo).

➲ Ejemplo: `addi $16, $17, 1` → suma una unidad al contenido del registro 17 y el resultado lo almacena en el registro 16.

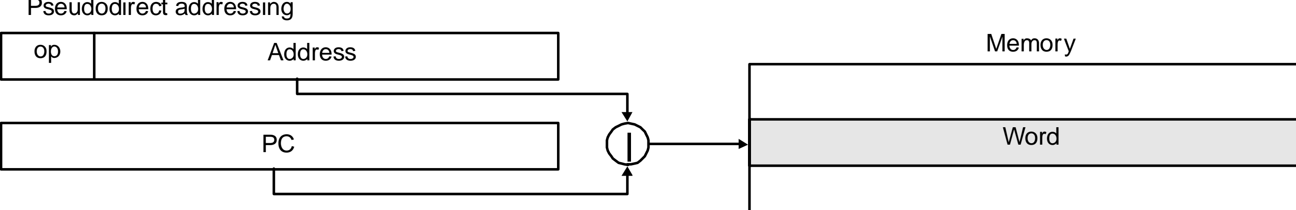
Direccionamiento pseudodirecto

➲ Campos:

- Dirección de 26 bits.

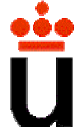
➲ La dirección se desplaza dos bits a la izquierda, y se concatena a los cuatro primeros bits del contador de programa.

➲ Se utiliza en instrucciones de salto.



➲ Notación: direccion (sin prefijo)

➲ Ejemplo: `j 2500` → salta a la dirección de memoria correspondiente a la palabra 2500 (dirección 10000).

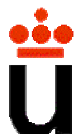


Direccionamiento implícito

➲ Un operando tiene direccionamiento implícito cuando está determinado por el código de operación.

➲En MIPS hay pocos casos.

➲ Ejemplo: `jal 2500` → almacena la dirección de la siguiente instrucción del programa (PC + 4) en el registro de retorno $ra y salta a la dirección de memoria correspondiente a la palabra 2500 (dirección 10000), donde empieza una subrutina.



5. Tipos de Instrucciones

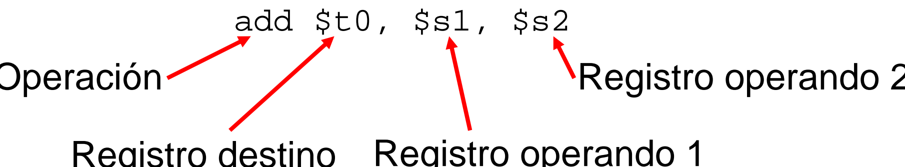
- **Transferencia de datos:** mover, cargar, almacenar.
- **Aritméticas para enteros:** sumar, restar, multiplicar, dividir, etc.
- **Aritméticas para coma flotante:** sumar, restar, multiplicar, dividir, etc.
- **Operaciones lógicas:** and, or, not, xor, etc.
- **Activación condicional:** si es r1 igual r2 entonces, saltar; si r1 es menor que r2, entonces saltar, etc.
- **Instrucciones de rotación y desplazamiento:** desplazar hacia la izquierda/derecha, etc.
- **Instrucciones de control de programa:** saltar, saltar si menor, saltar y enlazar dirección, etc.
- **Instrucciones de control de sistema:** provocar excepción, llamar al sistema operativo, etc.

Tipo R

Tipo R

(shamt: *shift amount* en instrucciones de desplazamiento)

Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino		Funct
xxxxxx	rs	rt	rd	shamt	funct
6	5	5	5	5	6
31-26	25-21	20-16	15-11	10-6	5-0



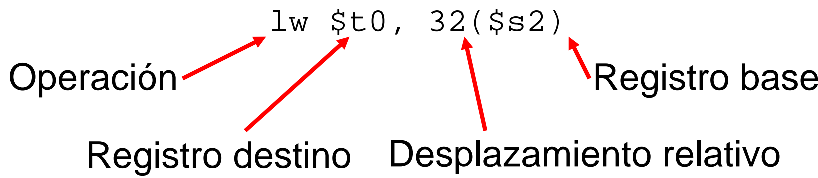
Código operación	Operando 1	Operando 2	Destino	Shamt	Función
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

Notación compacta hexadecimal: 02324020

Tipo I

Tipo I (carga o almacenamiento, ramificación condicional)

Cód. Op.	Registro base	Registro destino	Desplazamiento
xxxxxx	rs	rt	Inmediato
6	5	5	16
31-26	25-21	20-16	15-0

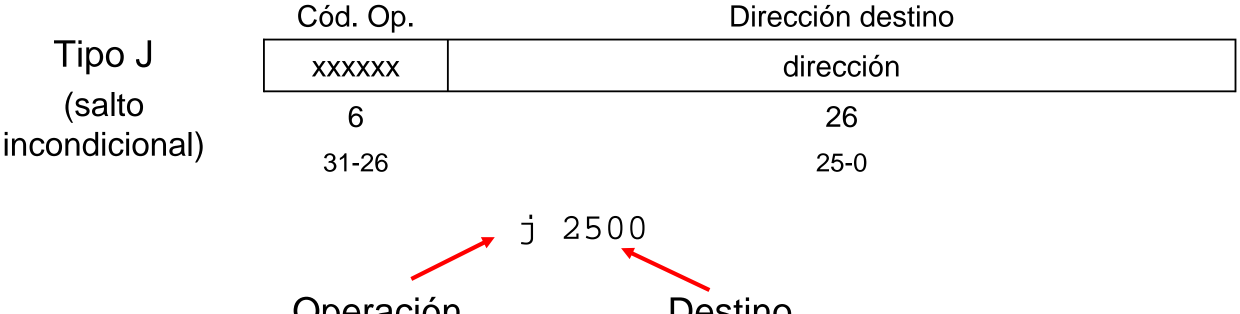


Código operación	Base	Destino	Desplazamiento
35	18	8	32
100011	10010	01000	0000000000010000

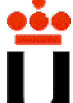
Notación compacta hexadecimal: 8E480010

Tipo J

Tipo J (salto incondicional)

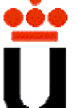


Código operación	PC	Destino
2	4000	2500
000010	**0000**0000000000000000111110100000	**0000**00000000000000100111000100**00** ←



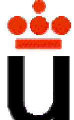
Instrucciones aritméticas y de desplazamiento para enteros e instrucciones lógicas

Sintaxis		T	Descripción
add	rd,rs,rt	I	rd = rs+rt
addi	rd,rs,inm16	I	rd = rs+ext_signo(inm16,32)
addu	rd,rs,rt	I	rd = rs+rt
addiu	rd,rs,inm16	I	rd = rs+ext_signo(inm16,32)
div	rs,rt	I	lo = rs/rt; hi = rem(rs/rt)
divu	rs,rt	I	lo = rs/rt; hi = rem(rs/rt)
mult	rs,rt	I	hi-lo = rs1*s2
multu	rs,rt	I	hi-lo = rs1*s2
sub	rd,rs,rt	I	rd = rs-rt
subu	rd,rs,rt	I	rd = rs-rt
and	rd,rs,rt	I	rd = rs AND rt
andi	rd,rs,inm16	I	rd = rs AND ext_ceros(inm16,32)
nor	rd,rs,rt	I	rd = rs NOR rt
or	rd,rs,rt	I	rd = rs OR rt
ori	rd,rs,inm16	I	rd = rs OR ext_ceros(inm16,32)
xor	rd,rs,rt	I	rd = rs XOR rt
xori	rd,rs,inm16	I	rd = rs XOR ext_ceros(inm16,32)
sll	rd,rt,shamt5	I	rd = desp_log(rt,shamt5,izquierda)
sllv	rd,rt,rs	I	rd = desp_log(rt,rs4..0,izquierda)
sra	rd,rt,shamt5	I	rd = desp_arit(rt,shamt5,derecha)
srav	rd,rt,rs	I	rd = desp_arit(rt,rs4..0,derecha)
srl	rd,rt,shamt5	I	rd = desp_log(rt,shamt5,derecha)
srlv	rd,rt,rs	I	rd = desp_log(rt,rs4..0,derecha)



Instrucciones de transferencia de datos

Sintaxis		T	Descripción
lb	rt,desp(rs)	I	rt = ext_signo(Mem[desp+rs]$_{7..0}$,32)
lbu	rt,desp(rs)	I	rt = ext_ceros(Mem[desp+rs]$_{7..0}$,32)
lh	rt,desp(rs)	I	rt = ext_signo(Mem[desp+rs]$_{15..0}$,32)
lhu	rt,desp(rs)	I	rt = ext_ceros(Mem[desp+rs]$_{15..0}$,32)
lui	rd,inm16	I	rd$_{31..16}$ = inm16; rd$_{15..0}$ = 0
lw	rt,desp(rs)	I	rt = Mem[desp+rs]
lwcZ	rt,desp(rs)	I	coprocesadorZ(rt) = Mem[desp+rs]
lwl	rt,desp(rs)	I	rt$_{31.16}$ = Mem[desp+rs]
lwr	rt,desp(rs)	I	rt$_{15..0}$ = Mem[desp+rs]
sb	rt,desp(rs)	I	Mem[desp+rs] = rt$_{7..0}$
sh	rt,desp(rs)	I	Mem[desp+rs] = rt$_{15..0}$
sw	rt,desp(rs)	I	Mem[desp+rs] = rt
swcZ	rt,desp(rs)	I	Mem[desp+rs] = coprocesadorZ(rt)
swl	rt,desp(rs)	I	Mem[desp+rs] = rt$_{31.16}$
swr	rt,desp(rs)	I	Mem[desp+rs] = rt$_{15..0}$
mfcZ	rt,rd	I	rd = rt; rd: registro UCP; rt: registro del coprocesador Z
mfhi	rd	I	rd = hi
mflo	rd	I	rd = lo
mtcZ	rd,rt	I	rd = rt; rd: registro UCP; rt: registro del coprocesador Z
mthi	rd	I	hi = rd
mtlo	rd	I	lo = rd



Instrucciones de activación condicional, control de programa y misceláneas

```
Sintaxis                    T  Descripción
--------------------------  -  --------------------------------------
slt     rd,rs,rt            I  Si rs < rt, rd = 1; si no, rd = 0
slti    rd,rs,inm16         I  Si rs < ext_signo(inm16,32), rd = 1; si no, rd = 0
sltu    rd,rs,rt            I  Si rs < rt, rd = 1; si no, rd = 0
sltiu   rd,rs,inm16         I  Si rs < ext_signo(inm16,32), rd = 1; si no, rd = 0

bcZf    etiqueta            I  Si flag(coprocesadorZ) = 0, ramificar a etiqueta
bcZt    etiqueta            I  Si flag(coprocesadorZ) = 1, ramificar a etiqueta
beq     rs,rt,etiqueta      I  Si rs = rt, ramificar a etiqueta
bgez    rs,etiqueta         I  Si rs >= 0, ramificar a etiqueta
bgezal  rs,etiqueta         I  Si rs >= 0, ramificar a etiqueta y enlazar ($ra = PC)
bgtz    rs,etiqueta         I  Si rs > 0, ramificar a etiqueta
blez    rs,etiqueta         I  Si rs <= 0, ramificar a etiqueta
bltz    rs,etiqueta         I  Si rs < 0, ramificar a etiqueta
bltzal  rs,etiqueta         I  Si rs < 0, ramificar a etiqueta y enlazar ($ra = PC)
bne     rs,rt,etiqueta      I  Si rs <> rt, ramificar a etiqueta
j       objetivo            I  PC = PC31..28 || (objetivo << 2)
jal     objetivo            I  ra = PC; PC = PC31..28 || (objetivo << 2)
jalr    rs,rd               I  rd = PC; PC = rs
jr      rs                  I  PC = rs

rfe                         I  Restaurar desde excepción (Restaura el registro Status)
syscall                     I  Llamada a un servicio del sistema ($v0: número del servicio)
break   codigo20            I  Provoca una excepción (código 1 reservado para el depurador)
nop                         I  No operación
```

Operaciones Aritméticas Básicas

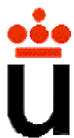
- MIPS es una máquina registro-registro (arquitectura de carga-almacenamiento): para usar un dato almacenado en memoria, primero hay que pasarlo a un registro.
- Las operaciones aritméticas básicas en MIPS se caracterizan por:
  - Utilizar tres registros (2 para los operandos y 1 para el resultado).
  - Sintaxis: `operacion resultado op1 op2`
  - El último de los operandos puede ser una constante de 16 bits (“inmediato”).
- Ejemplo: El mnemónico para la suma es `add`.

➲ C:

```
int a, b, c;
c = a + b;
```

➲ MIPS:

```
# Suponiendo que los datos a, b, c
# están asignados a los registros
# $s0, $s1, $s2 respectivamente:
add $s2, $s0, $s1    # $s2 = $s0 + $s1
```



• Si sólo podemos sumar dos registros en cada instrucción, ¿cómo hacemos operaciones más complicadas?

• Ejemplo: El mnemotécnico para la resta es `sub`.

➲ C:

```
int a, b, c, d, e;
a = ( b + c ) - ( d + e );
```

➲ MIPS:

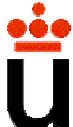
```
# Suponiendo que los datos a, b, c, d, e
# están asignados a los registros
# $s0, $s1, $s2, $s3 y $s4, respectivamente:
add $t0, $s1, $s2  # $t0 = $s1 + $s2
add $t1, $s3, $s4  # $t1 = $s3 + $s4
sub $s0, $t0, $t1  # $s0 = $t0 - $t1
```

Accesos a Memoria: Carga y Almacenamiento

• El ancho del bus de datos (“palabra”) es 32 bits (4 bytes).

• De igual forma, el ancho del bus de direcciones es 32 bits. Por tanto, en la memoria hay capacidad para 2^{32} posiciones, a cada una de las cuales le corresponde 1 byte.

- **Memoria total:** 2^{32} bytes = 4294967296 bytes = 4 Gigabytes = 2^{30} palabras.
- **“Big-Endian”:** dirección de palabra = dirección de byte más significativo.
- **Restricción de alineamiento:** Las direcciones de palabra son obligatoriamente múltiplos de 4.



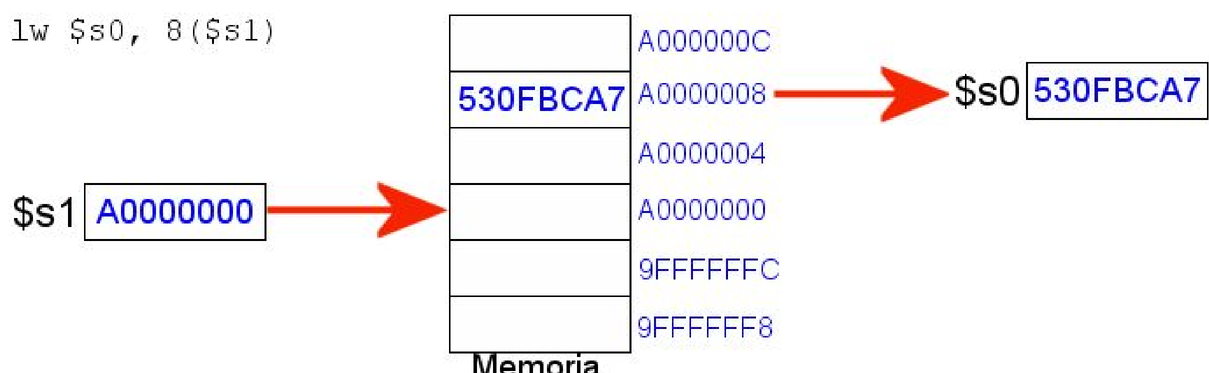
Accesos a Memoria: Carga y Almacenamiento

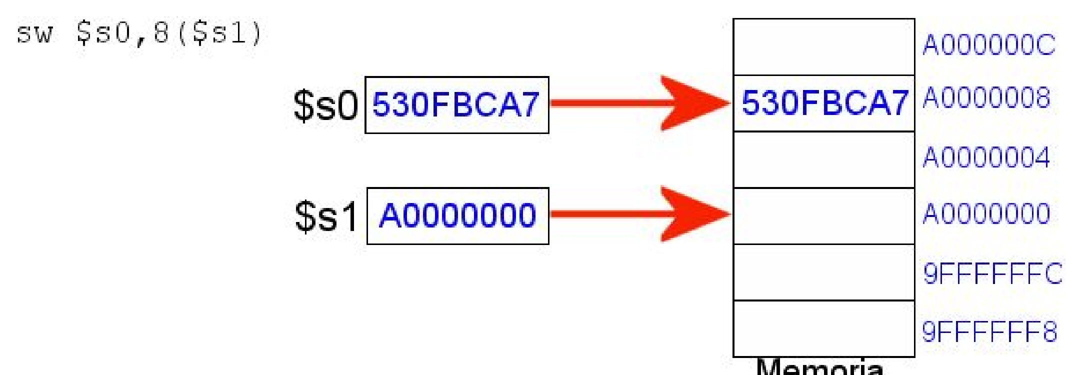
- Transferencia de datos entre memoria y registros:
  - **Carga ("load"):** Memoria → Registro.
  - **Almacenamiento ("store"):** Registro → Memoria.
- Los mnemotécnicos `lw` (load word) y `sw` (store word) permiten realizar transferencias entre memoria y registros de palabras enteras, utilizando la dirección de memoria almacenada en un registro base. Sintaxis:

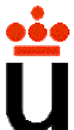
```
lw registro_destino, desplazamiento(registro_origen)

sw registro_origen, desplazamiento(registro_destino)
```

- La posición de memoria exacta se indica mediante un desplazamiento en bytes relativo a la dirección de memoria contenida en el registro origen (`lw`) o destino (`sw`).

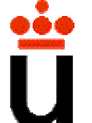






Sentencias de Control

- Destacamos las siguientes instrucciones:
  - `beq r1, r2, etiqueta` ("branch if equal")
    - Compara los valores contenidos en ambos registros.
    - Si son iguales, el flujo de programa salta a la instrucción que corresponde a la etiqueta.
    - Si no lo son, la instrucción que se ejecuta es la siguiente a ésta.
  - `bne r1, r2, etiqueta` ("branch if not equal")
    - En este caso, si lo valores de ambos registros no son iguales, el programa salta a la instrucción que corresponde a la etiqueta.
    - Si son iguales, la instrucción que se ejecuta es la siguiente a ésta.
  - `slt r1, r2, r3` ("set on less than")
    - El registro r1 se cargará con el valor 1 si el valor contenido en r2 es menor que el de r3.
    - En caso contrario, r1 valdrá 0.



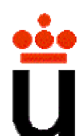
Ejemplos

➲ C:

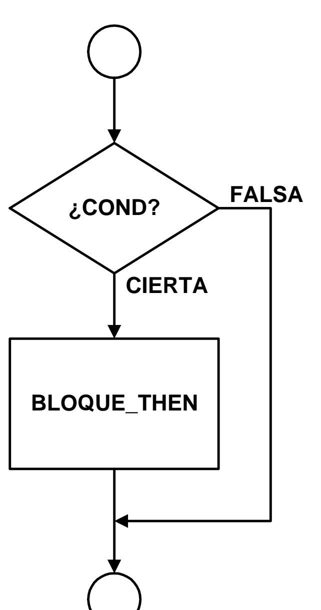
```
if (i==j) goto L1;
f = g + h;
L1: f = f - i;
```

➲ MIPS:

```
# Suponiendo que a las 5 variables de f a j les
# corresponden los registros de $s0 a $s4
     beq $s3, $s4, L1   # if ($s3 == $s4) goto L1
     add $s0, $s1, $s2  # f = g + h
L1:  sub $s0, $s0, $s3  # L1: f = f - i
```



6. Ejemplos: Condición IF-THEN

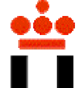


➲ C (variables enteras):

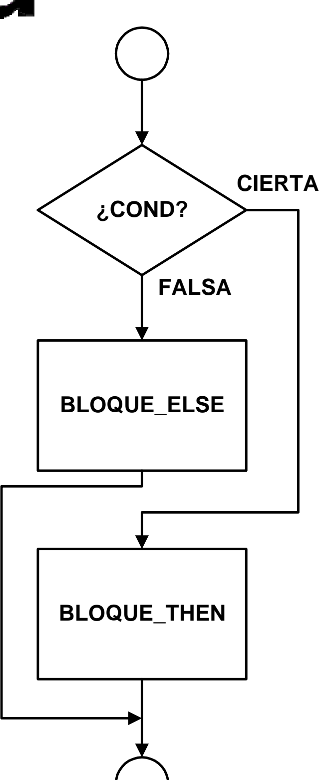
```
if (x >= y) {
    x = x+2;
    y = y-2;
};
```

➲ MIPS:

```
IF:
        lw     $s0,X
        lw     $s1,Y
        blt    $s0,$s1,END
        addiu  $s0,$s0,2
        addiu  $s1,$s1,-2
        sw     $s0,X
        sw     $s1,Y
END:
```

Condición IF-THEN-ELSE

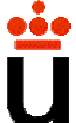


➲ C (variables enteras):

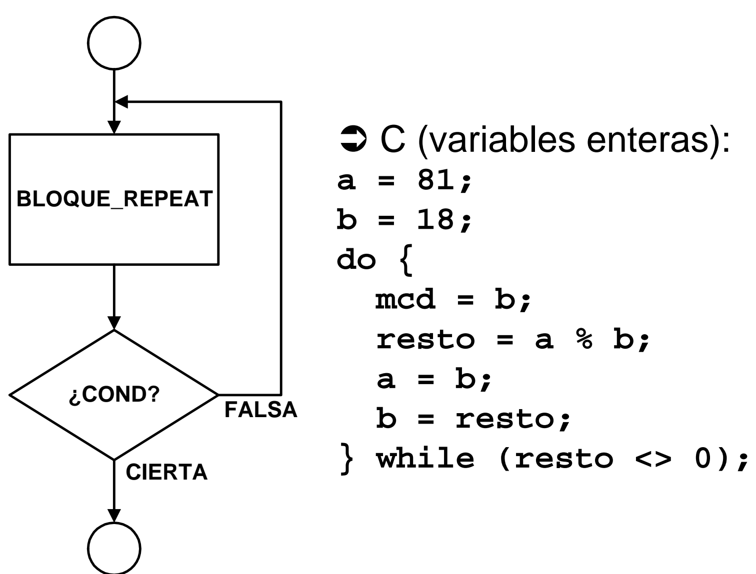
```
if (x >= y) {
    x = x+2; y = y+2;
}
else {
    x = x-2; y = y-2;
}
```

➲ MIPS:

```
IF:
        lw     $s0,X
        lw     $s1,Y
        bge    $s0,$s1,THEN
ELSE:   addiu  $s0,$s0,-2
        addiu  $s1,$s1,2
        sw     $s0,X
        sw     $s1,Y
        j      END
THEN:   addiu  $s0,$s0,2
        addiu  $s1,$s1,-2
        sw     $s0,X
        sw     $s1,Y
END:
```



Condición REPEAT-UNTIL



➲ C (variables enteras):

```
a = 81;
b = 18;
do {
  mcd = b;
  resto = a % b;
  a = b;
  b = resto;
} while (resto <> 0);
```

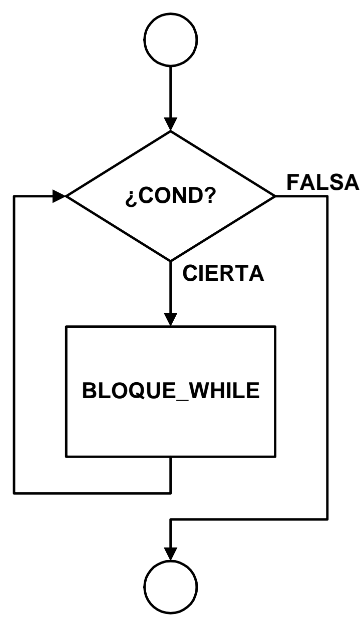
➲ MIPS:

```
# Variable A: $s0
# Variable B: $s1
# Variable MCD: $s2
# Variable RESTO: $s3

        li      $s0,81
        li      $s1,18
REPEAT:
        move    $s2,$s1
        div     $s0,$s1
        mfhi    $s3
        move    $s0,$s1
        move    $s1,$s3
UNTIL:  bnez    $s3,REPEAT
# Habría que salvar
# las variables A, B, MCD
# y RESTO en memoria
```

Algoritmo ejemplo: cálculo del máximo común divisor, con gestión de variables mediante registros.

Condición WHILE-DO



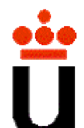
➲ C (variables enteras):

```
n = 5; fant = 1;
f = 1; i = 2;
while (i <= n) {
     faux = f;
     f = f + fant;
     fant = faux;
     i = i+1;
}
```

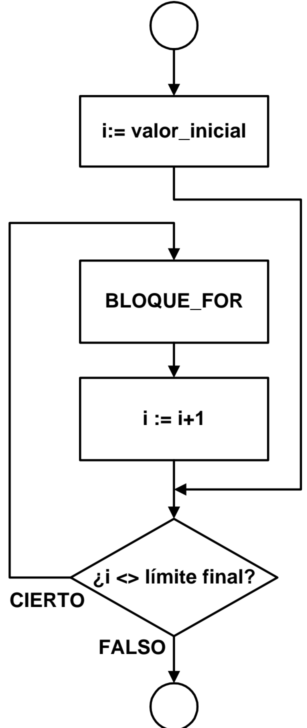
➲ MIPS

```
# Variable N: $s0
# Variable F: $s1
# Variable FANT: $s2
# Variable I: $s3
# Variable FAUX: $s4
        li      $s0,5
        li      $s2,1
        li      $s1,1
        li      $s3,2
WHILE:  bgt     $s3,$s0,END
        move    $s4,$s1
        addu    $s1,$s1,$s2
        move    $s2,$s4
        addiu   $s3,$s3,1
        j       WHILE
END:
# Salvar variables
# en memoria
```

Algoritmo del ejemplo: cálculo del término enésimo de la serie de Fibonacci.



Condición FOR

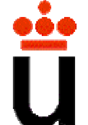


➲ C (variables enteras):

```
n = 5;
fant = 1;
f = 1;
for (i=2; i<=n; i++) {
  faux := f;
  f := f + fant;
  fant := faux;
}
```

➲ MIPS (versión PASCAL):

```
# Variable N: $s0
# Variable F: $s1
# Variable FANT: $s2
# Variable I: $s3
# Variable FAUX: $s4
        li      $s0,5
        li      $s2,1
        li      $s1,1
FOR:    move    $t0,$s0
        li      $s3,2
        bgt     $s3,$t0,END
        j       BODY
INC:    addiu   $s3,$s3,1
BODY:   move    $s4,$s1
        addu    $s1,$s1,$s2
        move    $s2,$s4
COND:   bne     $s3,$t0,INC
END:
# Salvar variables en memoria
```

Ejemplo

➲ C:

```
int a, V[100];
int i, j, k;
while (V[i] == k)
{
  i = i + j;
}
```

➲ MIPS:

```
# Suponiendo que a corresponde a $s0, V[0] a $s1,
# y las variables i, j, k a $s2 - $s4:
Bucle: add $t1, $s2, $s2      # t1 = 2*i
       add $t1, $t1, $t1      # t1 = 4*i
       add $t1, $t1, $s1      # t1 = dir. de V[i]
       lw $t0, 0($t1)         # t0 = V[i]
       bne $t0, $s4, Fin      # si i!=k salta
       add $s2, $s2, $s3      # i = i + j
       j Bucle
Fin:
```

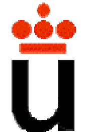
Ejemplo

➲ C:

```
int a, b, c, i, j;
if (i == j)
{
  a = b + c;
}
else
{
  a = b - c;
}
```

➲ MIPS:

```
# Suponiendo que a, b, c corresponde a $s0 - $s2
# y las variables i, j a $s3 - $s4:
        bne $s3, $s4, SiNo      # Si i!=j ir a SiNo
        add $s0, $s1, $s2       # se evita si i!=j
        j Fin                   # Salta a Fin
SiNo:   sub $s0, $s1, $s2       # se ejecuta si i!=j
Fin:
```

7. Conclusiones

➲ Programa y Lenguaje de Programación: Alto y bajo nivel.

➲ Lenguaje Ensamblador y su correspondencia con el Lenguaje Máquina.

➲ Aplicaciones de la programación en Ensamblador.

➲ Repertorio de Instrucciones del Lenguaje Máquina (características).

➲ Formato de Instrucciones.

➲ Modos de Direccionamiento.

➲ Ejemplos en los procesadores MIPS.