DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

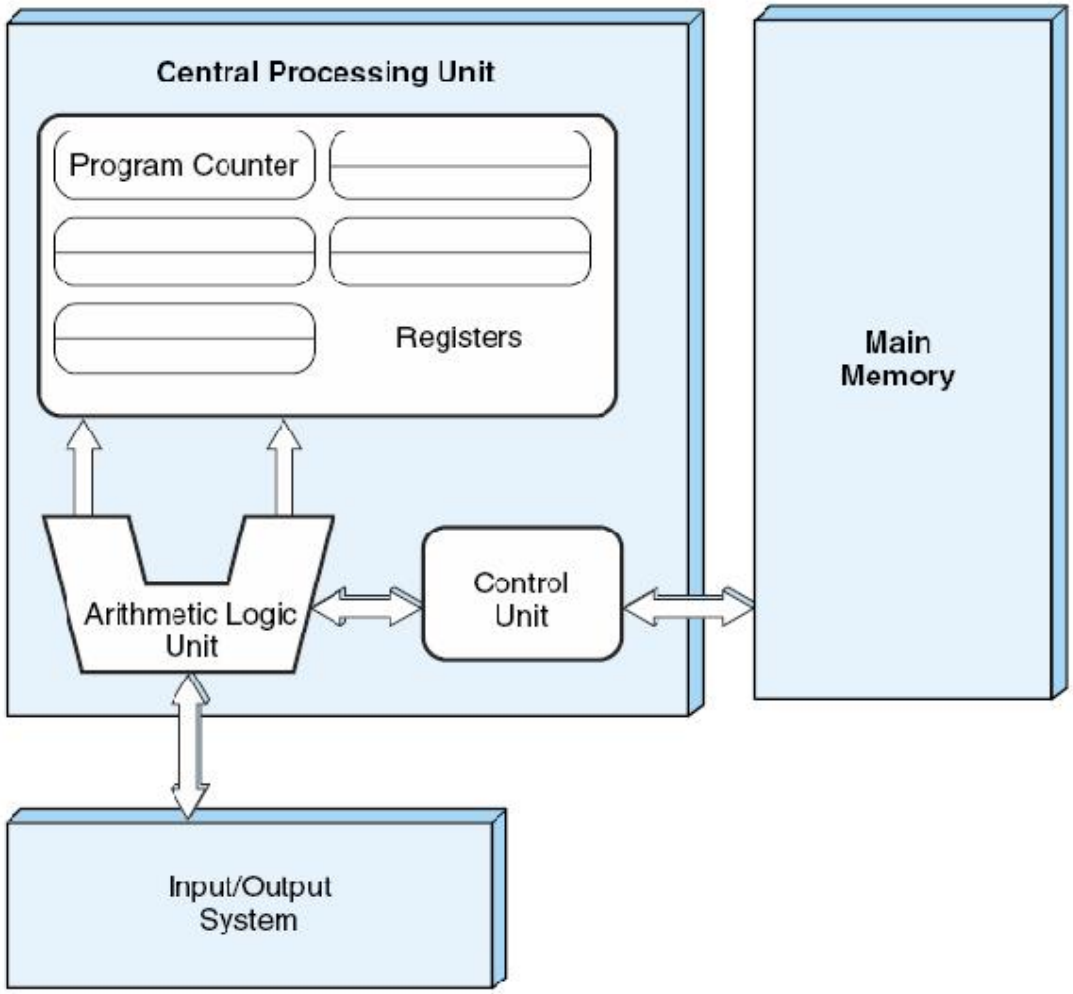
Organización del Computador 1
Modos de direccionamiento

Dr. Marcelo Risk

13 y 20 de septiembre de 2022

Índice

Modelo de Von Neumann



Ciclo de ejecución

1. UC (`Control Unit`) obtiene la **próxima** instrucción de memoria. Usa el registro PC (`Program Counter`).

Ciclo de ejecución

1. UC (`Control Unit`) obtiene la **próxima** instrucción de memoria. Usa el registro PC (`Program Counter`).

2. Se **incrementa** el PC.

Ciclo de ejecución

1. UC (`Control Unit`) obtiene la **próxima** instrucción de memoria. Usa el registro PC (`Program Counter`).

2. Se **incrementa** el PC.

3. La instrucción es **decodificada** a un lenguaje que entiende la ALU.

Ciclo de ejecución

1. UC (`Control Unit`) obtiene la **próxima** instrucción de memoria. Usa el registro PC (`Program Counter`).

2. Se **incrementa** el PC.

3. La instrucción es **decodificada** a un lenguaje que entiende la ALU.

4. Va a memoria para obtener los **operandos** requeridos por la operación.

Ciclo de ejecución

1. UC (`Control Unit`) obtiene la **próxima** instrucción de memoria. Usa el registro PC (`Program Counter`).

2. Se **incrementa** el PC.

3. La instrucción es **decodificada** a un lenguaje que entiende la ALU.

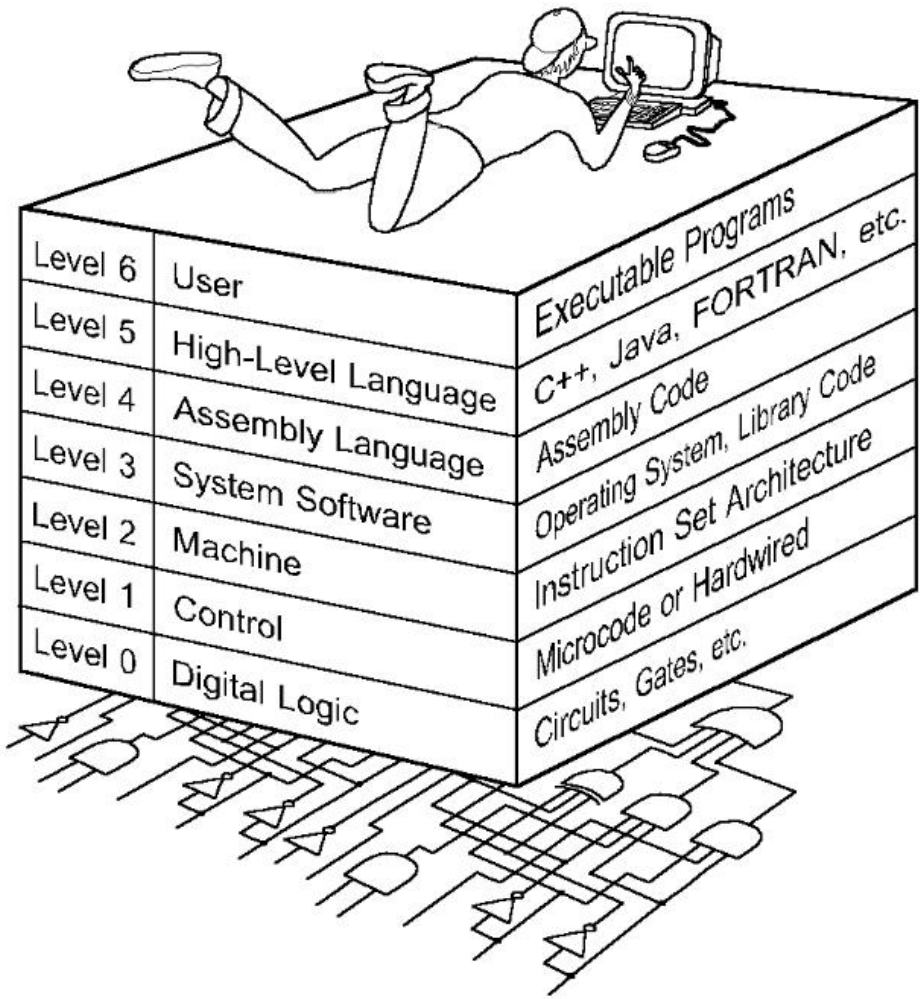
4. Va a memoria para obtener los **operandos** requeridos por la operación.

5. La ALU **ejecuta** y deja los resultados en registros o en memoria.

Ciclo de ejecución

1. UC (`Control Unit`) obtiene la **próxima** instrucción de memoria. Usa el registro PC (`Program Counter`).
2. Se **incrementa** el PC.
3. La instrucción es **decodificada** a un lenguaje que entiende la ALU.
4. Va a memoria para obtener los **operandos** requeridos por la operación.
5. La ALU **ejecuta** y deja los resultados en registros o en memoria.
6. **Repetir** a partir de paso 1.

Jerarquía de niveles



ISA

- Nivel de Lenguaje de Máquina (Instruction Set Architecture).
- Límite entre Hardware-Software.
- Es lo que vemos como programadores.
- Define:
  - Cómo se representan los datos, como se almacenan, como se acceden.
  - Qué operaciones se pueden realizar.
  - Cómo se codifican estas operaciones.
- No importa la implementación interna.

Características de una ISA

- Cantidad de memoria que un programa requiere.
- Complejidad del conjunto de instrucciones. Por ejemplo, RISC vs CISC.
- Longitud de las instrucciones (fija o variable).
- Cantidad total de instrucciones.

¿Cómo se representan los datos?

¿Qué soporte hay para distintos tipos de datos?:

- Enteros (8, 16, 32... bits, ¿complemento a 2?).
- Big-endian, Little endian.
- Punto Flotante.
- ¿BCD, ASCII, UNICODE?

Little vs Big endian

- *"endian"* se refiere a la forma en que la computadora guarda datos que ocupan más de un byte (i.e. *multibyte*).
- Por ejemplo, cuando se guarda un entero de dos bytes en memoria:
  - *"Little endian"*: el byte en una posición de memoria menor, es menos significativo.
  - *"Big endian"*: el byte en una posición de memoria menor, es el más significativo.

Little vs Big endian

Supongamos que tenemos un número entero de dos bytes. El `Byte0` es el menos significativo, `Byte1` el más significativo. ¿Cómo queda en memoria en cada caso?

Little vs Big endian

Supongamos que tenemos un número entero de dos bytes. El `Byte0` es el menos significativo, `Byte1` el más significativo. ¿Cómo queda en memoria en cada caso?

"Little endian":

`Base address` $+0 =$ `Byte0`

`Base address` $+1 =$ `Byte1`

"Big endian":

`Base address` $+0 =$ `Byte1`

`Base address` $+1 =$ `Byte0`

Little vs Big endian

Supongamos que tenemos un número entero de dos bytes. El `Byte0` es el menos significativo, `Byte1` el más significativo. ¿Cómo queda en memoria en cada caso?

"Little endian":

`Base address` +0 = `Byte0`
`Base address` +1 = `Byte1`

"Big endian":

`Base address` +0 = `Byte1`
`Base address` +1 = `Byte0`

Importante

Habitualmente, como programadores de alto nivel, no nos importa el *endianess* del sistema (si es *"Big Endian"* o *"Little endian"*). Nos comienza a importar cuando nos toca interpretar a nosotros qué es lo que hay en memoria, por ejemplo cuando hay que analizar un *dump* de memoria o entender qué hace una aplicación a bajo nivel.

Acceso a los datos

- ¿Dónde se almacenan?
  - Registros.
  - Memoria.
  - Stack.
  - Espacio de E/S.
- ¿Cómo se acceden?
  - Modos de Direccionamiento.

Operaciones

- Movimiento de datos (`Move`, `Load`, `Store`, ...)

Operaciones

- Movimiento de datos (`Move`, `Load`, `Store`, ...)
- Aritméticas (`Add`, `Sub`, ...)

Operaciones

- Movimiento de datos (`Move`, `Load`, `Store`, ...)
- Aritméticas (`Add`, `Sub`, ...)
- Lógicas (`And`, `Xor`, ...)

Operaciones

- Movimiento de datos (`Move`, `Load`, `Store`, ...)
- Aritméticas (`Add`, `Sub`, ...)
- Lógicas (`And`, `Xor`, ...)
- I/O: acceso a dispositivos de Entrada/Salida

Operaciones

- Movimiento de datos (`Move`, `Load`, `Store`, ...)
- Aritméticas (`Add`, `Sub`, ...)
- Lógicas (`And`, `Xor`, ...)
- I/O: acceso a dispositivos de Entrada/Salida
- Transferencia de control (`Jump`, `Skip`, `Call`, ...)

Operaciones

- Movimiento de datos (`Move`, `Load`, `Store`, ...)
- Aritméticas (`Add`, `Sub`, ...)
- Lógicas (`And`, `Xor`, ...)
- I/O: acceso a dispositivos de Entrada/Salida
- Transferencia de control (`Jump`, `Skip`, `Call`, ...)
- Específicas. Ejemplo: Multimedia

Codificación

Cada instrucción se representa de manera única y la codificación debe tener en cuenta lo siguiente:

- Códigos de operación (`OpCode`)
  - Representa la operación $\langle$`Add`, `Sub`, `Mult`, ...$\rangle$

Codificación

Cada instrucción se representa de manera única y la codificación debe tener en cuenta lo siguiente:

- Códigos de operación (`OpCode`)
  - Representa la operación ⟨`Add`, `Sub`, `Mult`, ...⟩
- Operando/s Fuente
  - A realizar sobre estos datos ⟨*registro*, *memoria*, ...⟩

Codificación

Cada instrucción se representa de manera única y la codificación debe tener en cuenta lo siguiente:

- Códigos de operación (`OpCode`)
  - Representa la operación ⟨`Add`, `Sub`, `Mult`, ...⟩
- Operando/s Fuente
  - A realizar sobre estos datos ⟨*registro*, *memoria*, ...⟩
- Operando Resultado
  - Pone la respuesta aquí ...

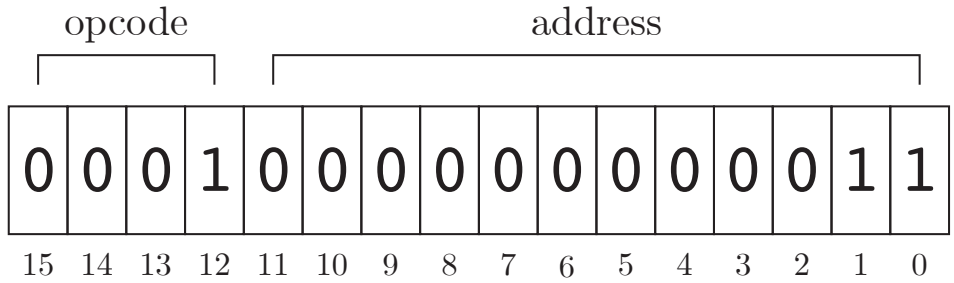
Codificación

Cada instrucción se representa de manera única y la codificación debe tener en cuenta lo siguiente:

- Códigos de operación (`OpCode`)
  - Representa la operación ⟨`Add`, `Sub`, `Mult`, ...⟩
- Operando/s Fuente
  - A realizar sobre estos datos ⟨*registro*, *memoria*, ...⟩
- Operando Resultado
  - Pone la respuesta aquí ...
- Referencia a la próxima instrucción
  - Cuando termina sigue por aquí ...

Jerarquía de niveles

- Instrucción LOAD en el IR:



- Opcode=1, Cargar en el AC el dato contenido en la dirección 3.

Modos de Direccionamiento

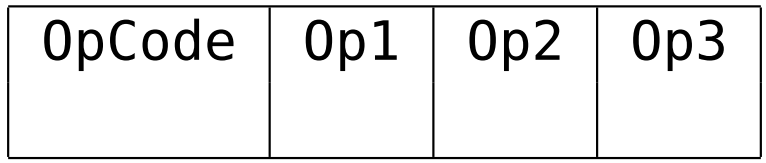
Instrucción: OpCode + Operandos

Modos de Direccionamiento

Instrucción: OpCode + Operandos

¿Qué tipos de cosas pueden ser los operandos?:

- Constantes
- Referencia a Variables
- Referencia a Arrays
- Referencias a subrutinas
- Estructuras de datos (Listas, Colas)



Modos de Direccionamiento

- Inmediato
- Directo (o absoluto)
- Indirecto
- Registro
- Indirecto con registro
- Desplazamiento (Indexado)

Inmediato

- El operando es parte de la instrucción (N).
- Ej en Marie: `ADD 5`
  - Efecto: `AC = AC + 5`
  - `5` es un operando.

Inmediato

- El operando es parte de la instrucción (N).
- Ej en Marie: `ADD 5`
  - Efecto: `AC = AC + 5`
  - `5` es un operando.
- Ej2: `Jump 110`
  - No hay referencia adicional a memoria.
  - Efecto: `PC = 110`
  - Rápido.
  - **Puede tener rango limitado.**

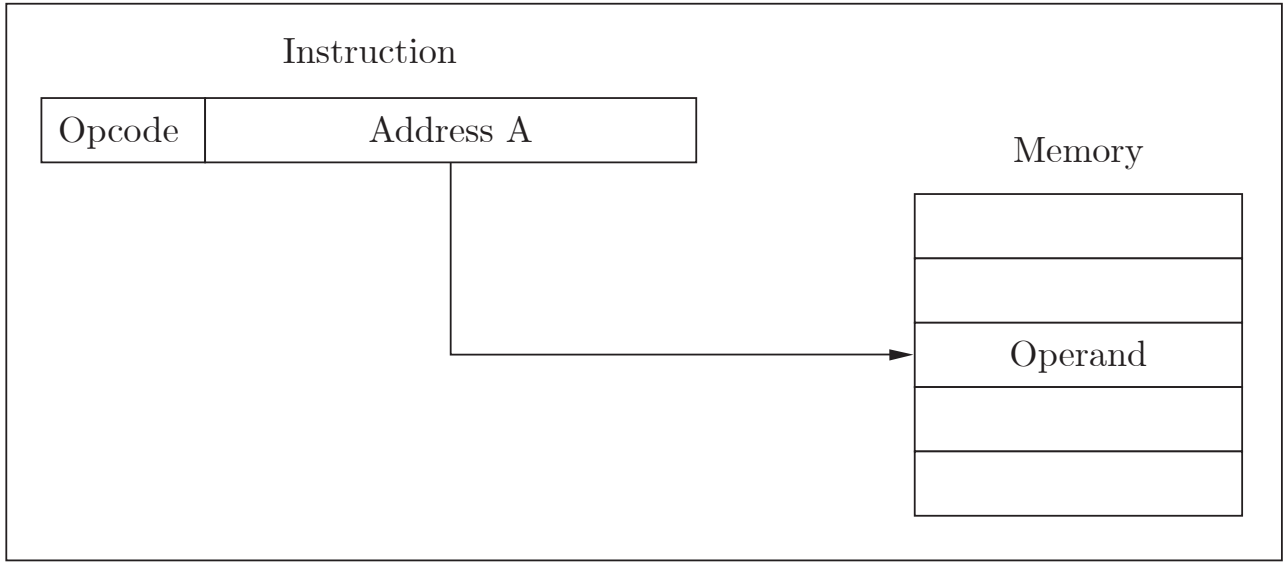
Directo

- El operando está en la dirección referenciada por A.
- Operando = [A]
- Ej: `ADD [941]` (`A = A + [941]`)
- Ideal para acceso a variables.
- Hay sólo un acceso a la memoria.
- Direccionamiento limitado a tamaño del operando.

Directo

Opcode	A

- El operando está en la dirección referenciada por A.
- Operando = [A]
- Ej: `ADD [941]` (`A = A + [941]`)
- Ideal para acceso a variables.
- Hay sólo un acceso a la memoria.
- Direccionamiento limitado a tamaño del operando.



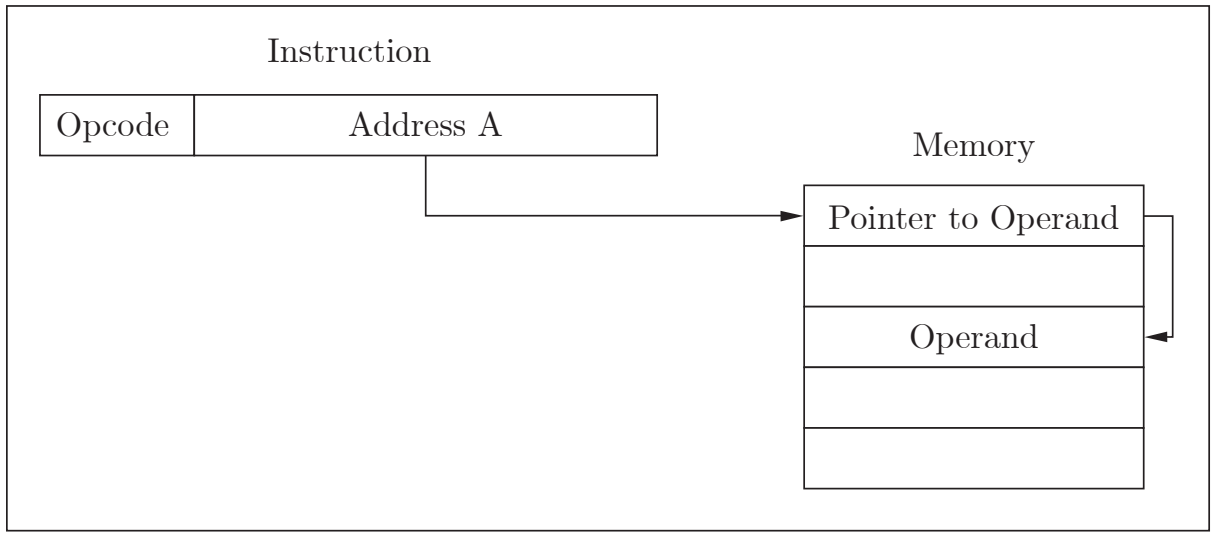
Indirecto

- A es un Puntero.
- Operando = [[A]]
- Usos:
  - Acceso a Arrays, Listas u otras estructuras.
  - Aumenta el espacio de direccionamiento.
- Existe acceso múltiple a la memoria para encontrar el operando.

Indirecto

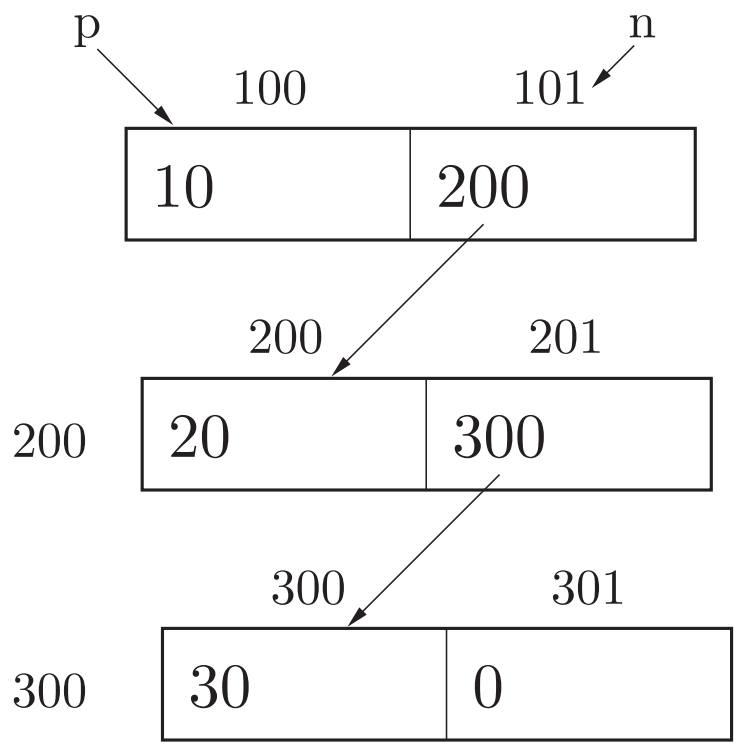
Opcode	A

- A es un Puntero.
- Operando = [[A]]
- Usos:
  - Acceso a Arrays, Listas u otras estructuras.
  - Aumenta el espacio de direccionamiento.
- Existe acceso múltiple a la memoria para encontrar el operando.



Ejemplo Lista indirecto

```
          MOV R1,0
Start:    CMP [p],0
          JE fin
          ADD R1,[[p]]
          MOV [p],[[n]]
          MOV [n],[p]
          ADD [n],1
          JMP Start
End:
p: DW 100
n: DW 101
```



Registro

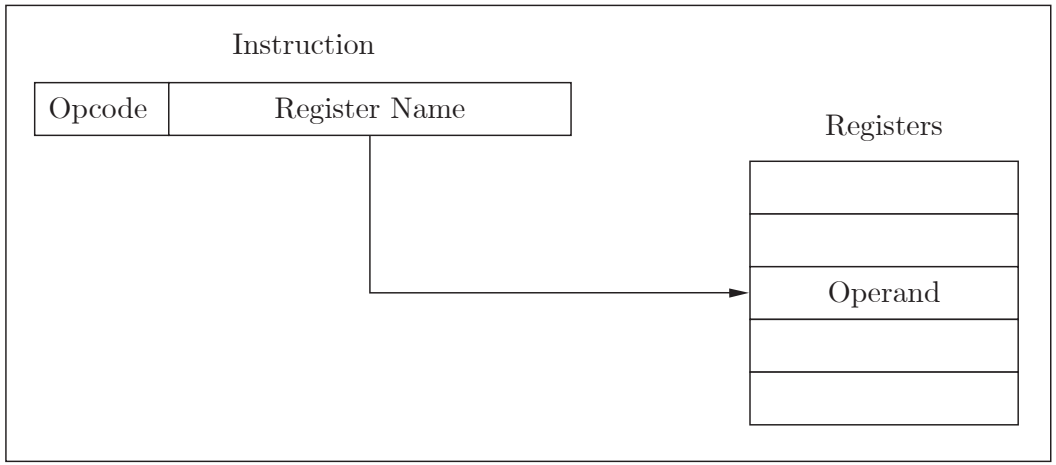
Opcode	Rn

- El operando es un registro de la CPU.
- Operando = Registro n
- Número **limitado** de registros.
- Instrucción rápida:
  - Ej: Mov R1,R2.
- No hay acceso a memoria.
- Instrucción **corta**.
- Espacio de direcciones limitado.

Registro

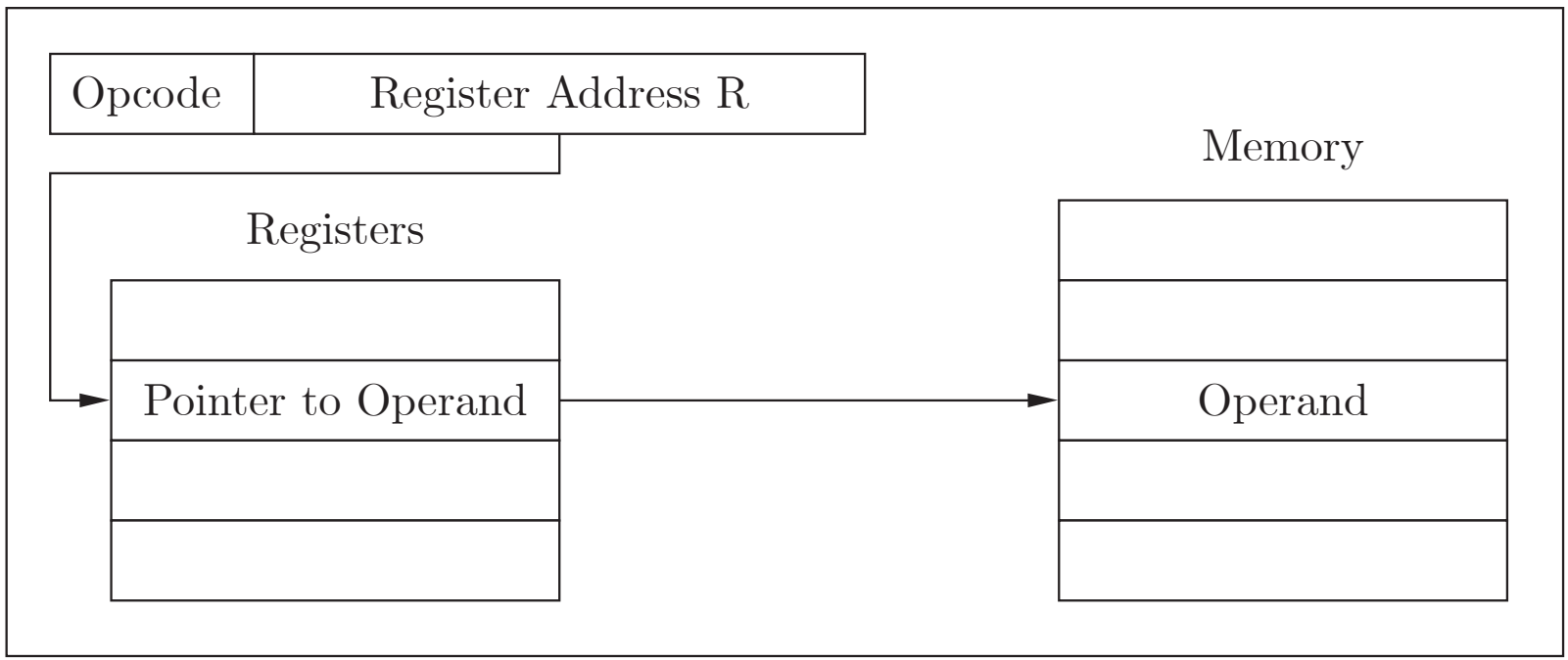
Opcode	Rn

- El operando es un registro de la CPU.
- Operando = Registro n
- Número **limitado** de registros.
- Instrucción rápida:
  - Ej: Mov R1,R2.
- No hay acceso a memoria.
- Instrucción **corta**.
- Espacio de direcciones limitado.



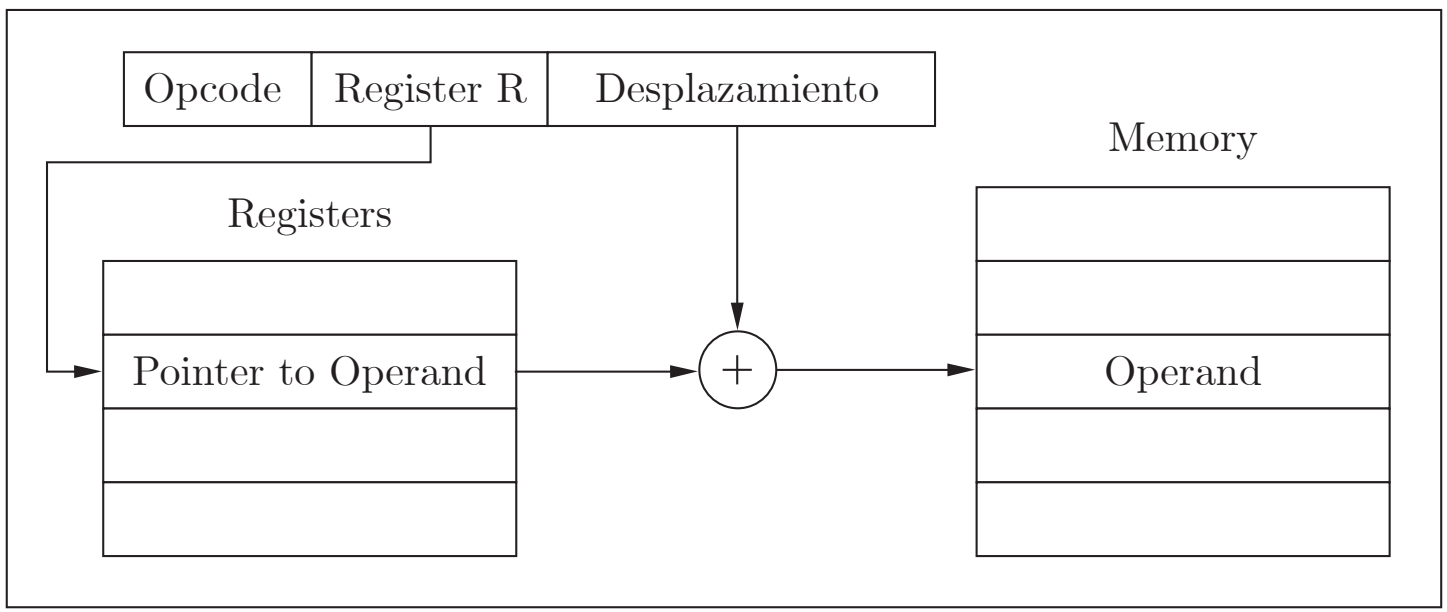
Registro Indirecto

- El operando está en la memoria direccionada por un registro.
- Operando = [Rn]
- Hay un acceso menos a memoria que en direccionamiento indirecto.
- Cómodo para acceder a arrays.



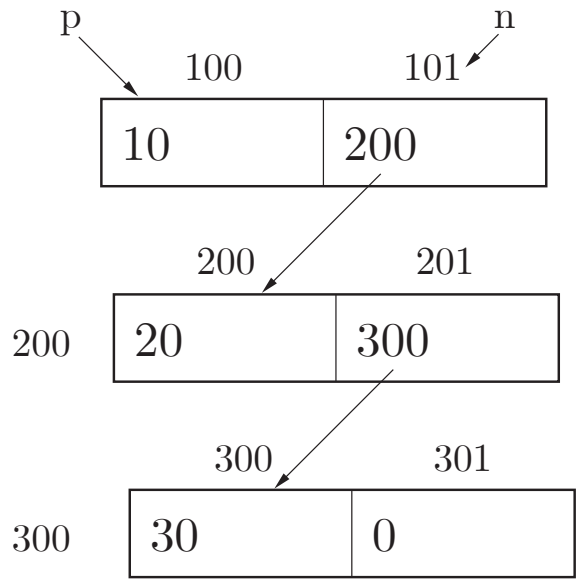
Desplazamiento

- El operando contiene una referencia a un registro y a un valor de desplazamiento.
- Operando = $[R_{N1} + D]$
- Ideal para acceder a campos de registros.
  - Moviendo D.
- También para arrays de estructuras.
  - R se mueve dentro del array.
  - D selecciona el campo.



Ejemplo Lista sin desplazamiento

```
            MOV R1,0
            MOV R2,[p]
            MOV R3,[n]
Start:      CMP [r2],0
            JE fin
            ADD R1,[R2]
            MOV R2,[R3]    // R2=[[n]]
            MOV R3,R2
            ADD R3,1       // R3 = [n]
            JMP Start
End:
p: DW 100
n: DW 101
```



Ejemplo Lista con desplazamiento

```
            MOV R1,0
            MOV R2,[p]
Start:      CMP [r2],0
            JE fin
            ADD R1,[R2]
            MOV R2,[R2+1]; R2=[[n]]
            JMP Start
End:
p: DW 100
```

Indexado

- Similar al desplazamiento.
- Un operando contiene una referencia a una dirección y a un registro que actúa como desplazamiento.
- Operando = $[D + R_{N1}]$
- Ideal para arrays.

Modos de direccionamiento

Modo de Direccionamiento	
Inmediato	`Mov R, cte` R ← N
Directo	`MOV R, [A]` R ← mem[A]
Registro	`MOV R1,R2` R1 ← R2
Registro Indirecto	`MOV R1,[R2]` R1 ← mem[R2]
Desplazamiento	`MOV R1, [R2+D]` R1 ← mem[R2+ D]
Desplazamiento basado en registro	`MOV R1, D (`R_{base}`)` R1 ← mem[R_{base} + D]
Indexado	`MOV R1, A[R]` R1 ← mem[A+R]
Escalado indexado (*Indexed Scaled*)	`MOV R1, A[R*Scale]` R1 ← mem[A+R*Scale]
Relativo a PC	Jump N PC ← PC + N

Ejemplo

- Completar el valor de AC según el modo de direccionamiento.

Memory

800	900
...	
900	1000
...	
1000	500
...	
1100	600
...	
1600	700

R1 800

```
LOAD 800
```

Mode	**Value Loaded into AC**
Inmediate	
Direct	
Indirect	
Indexed	

Ejemplo

Memory

800	900
...	
900	1000
...	
1000	500
...	
1100	600
...	
1600	700

R1 800

```
LOAD 800
```

Mode	**Value Loaded into AC**
Inmediate	800
Direct	900
Indirect	1000
Indexed	700

Diseñando una ISA

Temas a considerar:

- Tipos de operación.
- Número de bits por instrucción.
- Número de operandos por instrucción
  - Operandos implícitos y explícitos
  - Ubicación de los operandos
  - Tamaño y tipo de los operandos
- Uso de Stack, Registros.

Diseñando una ISA

Algunas características:

- Memoria principal ocupada por el programa.
- Tamaño de la instrucción (en bits).
  - Code density: tratar que las instrucciones ocupen poco.
- Complejidad de la instrucción.
- Número total de instrucciones disponibles.

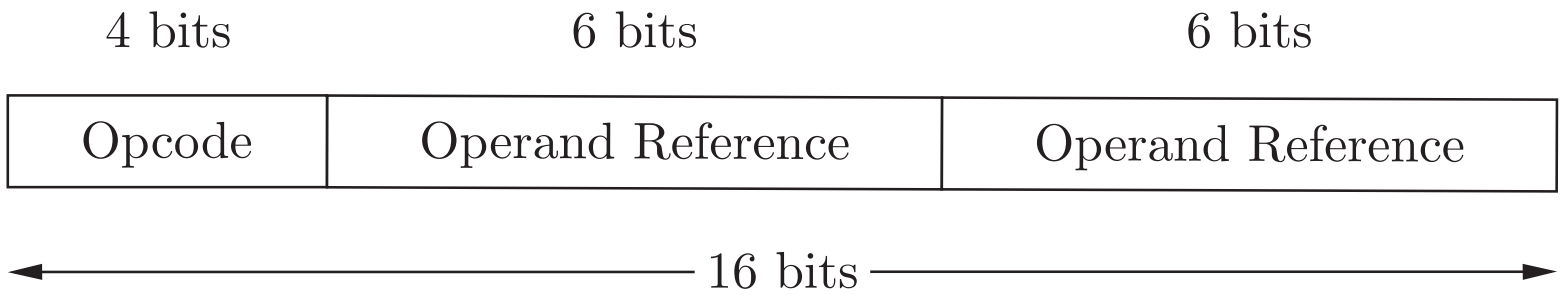
Criterios en diseños de ISA

- Tamaño de la instrucción
  - ¿Corto, largo, variable?
- ¿Cuántos operandos?
- ¿Cuántos registros?
- Memoria
  - ¿Direccionable por byte o por palabra (word)?
  - Big/Little Endian
  - ¿Cuántos modos de direccionamiento?
    - Directo, indirecto, indexado...
    - Muchos
    - Pocos
    - Uno solo

¿Cuántos Operandos?

- 3 operandos: RISC y Mainframes
  - A = B + C
- 2 operandos: Intel, Motorola
  - A = A + B
  - Uno suele ser un registro
- 1 operando: Acumulador
  - AC = AC + Dato
- 0 operandos: Stack Machines
  - Usan un stack
- Muchos operandos (VLIW)
  - Paralelismo implícito en la instrucción

Formato Posible



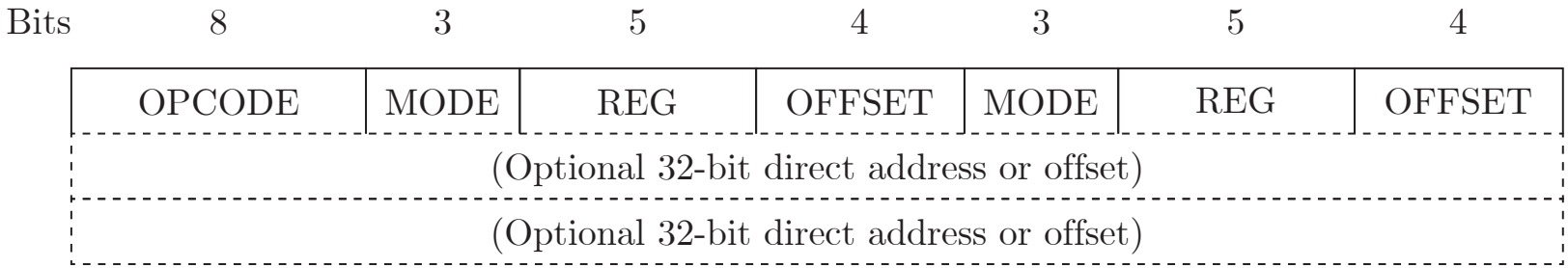
Algunas ISA

	CISC examples			**RISC examples**		**Superscalars**	
	IBM 370/168	**VAX 11/780**	**Intel 80486**	**88000**	**R4000**	**RS/6000**	**80960**
Year developed	1973	1978	1989	1988	1991	1990	1989
The number of instruction	**208**	**303**	**235**	**51**	**51**	**184**	**62**
Instruction size (bytes)	**2 - 6**	**2 - 57**	**1 - 11**	**4**	**4**	**4**	**4, 8**
Addressing modes	**4**	**22**	**11**	**3**	**1**	**2**	**11**
The number of GRPs	**16**	**16**	**8**	**32**	**32**	**32**	**32 - 256**
Control memory size (K bits)	420	480	246	-	-	-	-
Cache size (KB)	64	64	8	16	128	32 - 64	0.5

Ortogonalidad

- Cualquier instrucción puede ser usada con cualquier modo de direccionamiento.
- Es una cualidad “elegante”, pero costosa:
  - Implica tener muchas instrucciones.
  - Algunas quizás poco usadas o fácilmente reemplazables.

Formato Posible



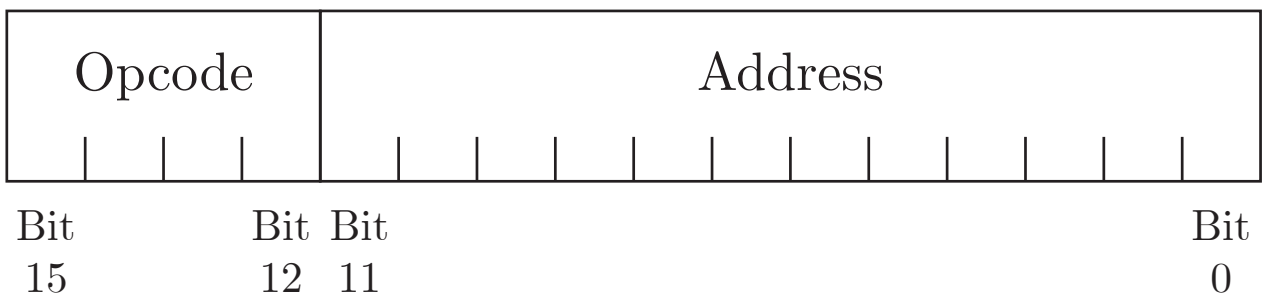
- Instrucción 32 bits.
- 256 instrucciones posibles.
- 2 operandos (Op1 = Op1 op Op2)
- 12 bits por operando
  - MODE = 8 Modos de Direccionamiento
  - Reg = 32 Registros
  - OffSet = Desplazamiento o Escala (4 bits)
- **Problema**: Para direccionamiento directo o inmediato hay que acceder a los campos opcionales.

Ejemplo: MARIE

- Máquina de Acumulador:
  - Representación binaria, complemento a 2.
  - Instrucciones de tamaño fijo.
  - Memoria accedida por palabras de 4K.
  - Palabra de 16 bits.
  - Instrucciones de 16 bits, 4 para el código de operación y 12 para las direcciones.
  - Una ALU de 16 bits.
  - 7 registros para control y movimiento de datos.

MARIE

- Registros Visibles:
  - AC: Acumulador
    - 16 bits
    - Operando implícito en operaciones binarias
    - También para condiciones
- Formato de instrucción fijo

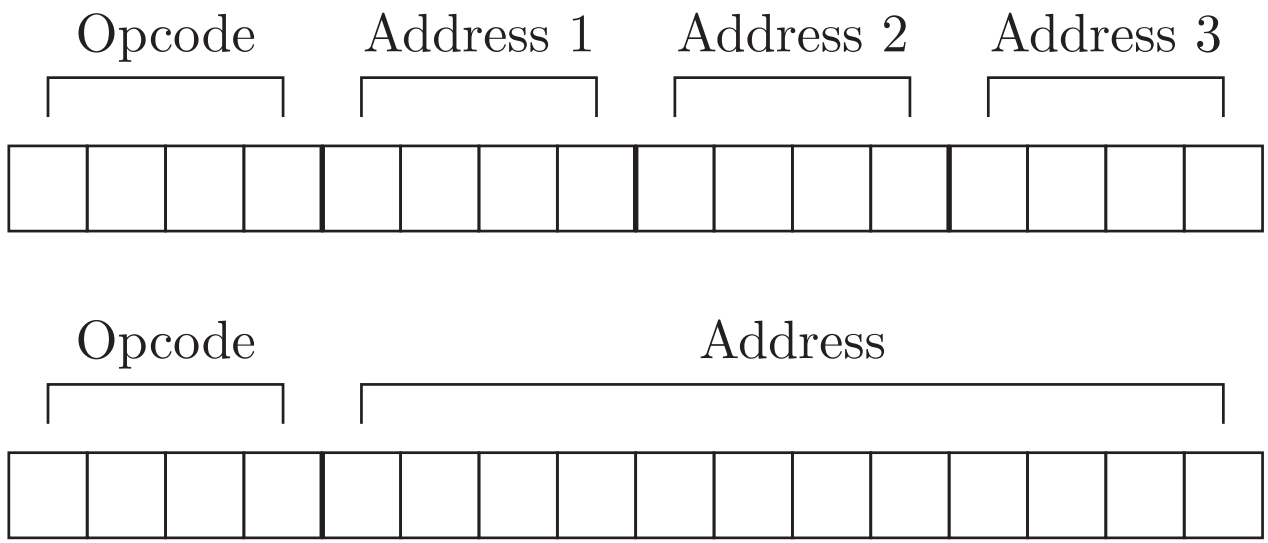


Formatos de instrucción

- El tamaño de las instrucciones está fuertemente influenciado por el número de operandos que soporta la ISA.
- No todas las instrucciones requieren el mismo número de operandos.
- Hay operaciones que no requieren operandos (ej: **HALT**)
  - ¿Qué hacemos con el espacio que sobra?
- Podríamos utilizar códigos de operación variables.

Ejemplo Máquina con Registros

- 16 Registros, 4K de memoria.
- Necesitamos 4 bits para acceder a un registro.
- Necesitamos 12 bits para acceder a memoria.
- Si las instrucciones son de 16-bits tenemos dos opciones:



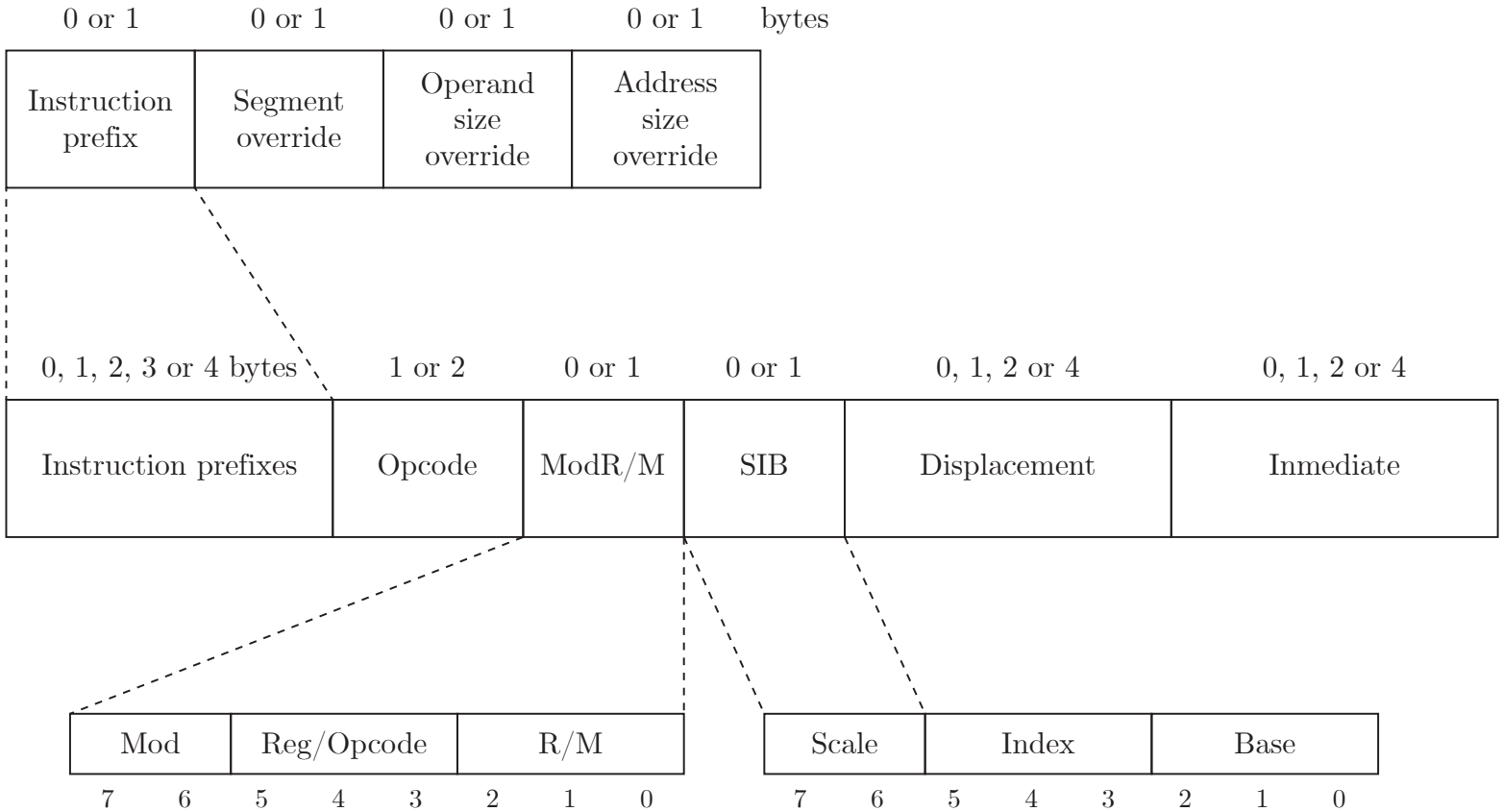
Ejemplo

- Si permitimos que varíe el opcode:

```
0000  R1   R2   R3        }  15 3-address codes
1110  R1   R2   R3        }
1111 0000  R1   R2        }  14 2-address codes
1111 1101  R1   R2        }
1111 1110 0000  R1        }  31 1-address codes
1111 1111 1110  R1        }
1111 1111 1111 0000       }  16 0-address codes
1111 1111 1111 1111       }
```

¿Falta algo?

Formato de Instrucción Pentium



Ejemplo

Prefijo	OpCode	MODR/M	SIB	Desplazamiento	Inmediato
0xF3	0xA4				

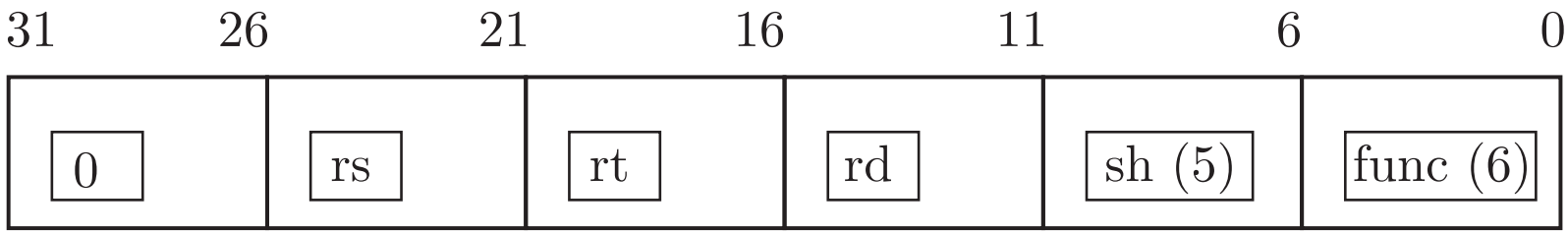
Instrucción: REP MOVS
Copia CX bytes de DS:SI, a ES:DI.

MOVS: Copia el dato en DS:SI, a ES:DI.
Dependiendo de un flag, SI y DI son incrementados (+1) o decrementados (-1)
REP decrementa CX y hace que se repita la operación hasta que CX llegue a 0

Formato MIPS de Instrucción

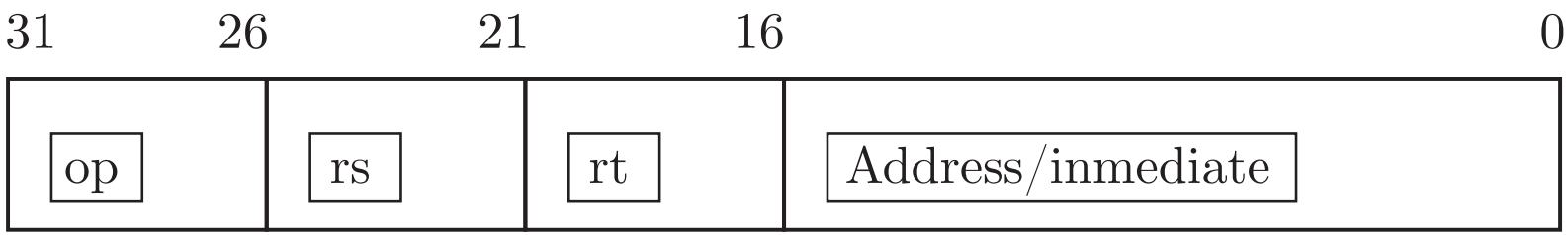
Son todas de 32 bits. Tres formatos:

- Tipo R
  - Aritméticas.



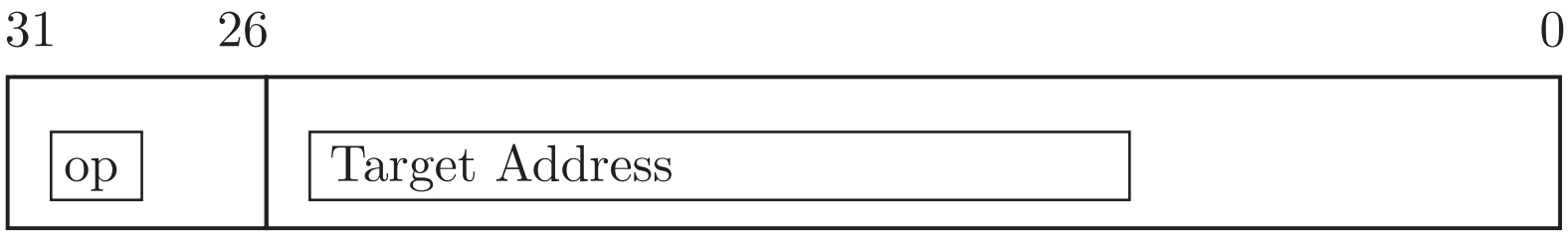
Formato MIPS de Instrucción

- Tipo I
  - Transferencia, salto.
  - Operaciones con operando inmediato.

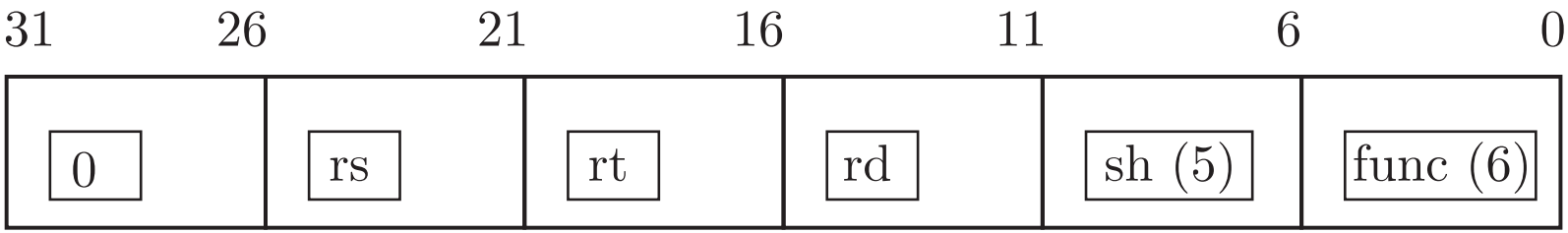


Formato MIPS de Instrucción

- Tipo J
  - Saltos.



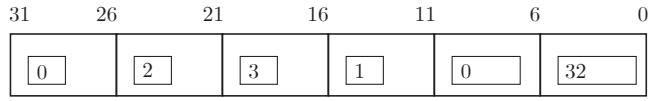
Formato R (registro)



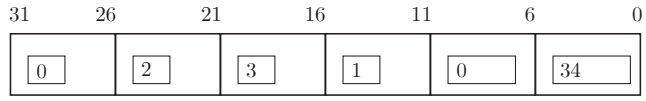
- Op= 0
- rs, rt = identificación de los registros fuente.
- rd = identificación del registro destino.
- sh= cantidad de shifts.
- func= identifica la operación (por ej. add=32, sub=34, sll=0,srl=10)

Formato R: Ejemplos)

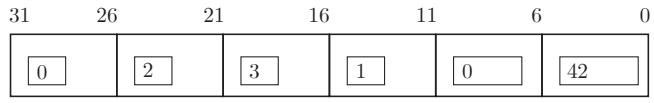
- Add \$1, \$2, \$3



- Sub \$1, \$2, \$3

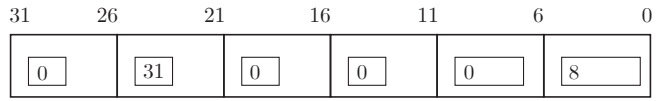


- Slt \$1, \$2, \$3



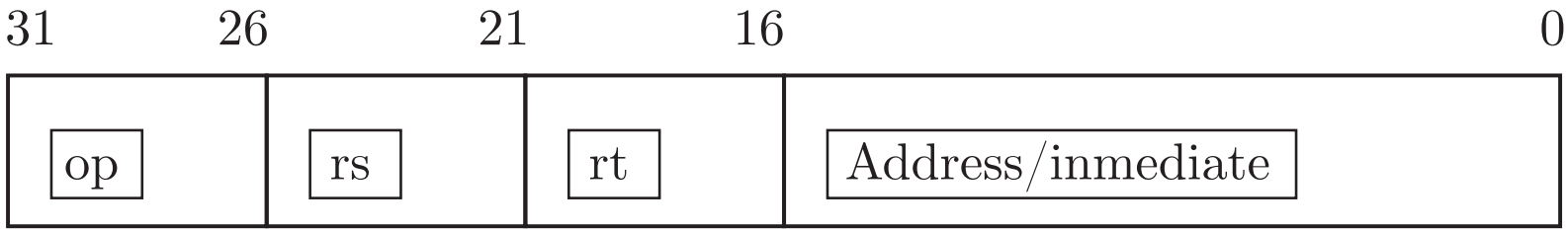
  Set Less Than
  si (\$2 < \$3) entonces \$1 = 1 sino \$1 = 0

- Jr \$31

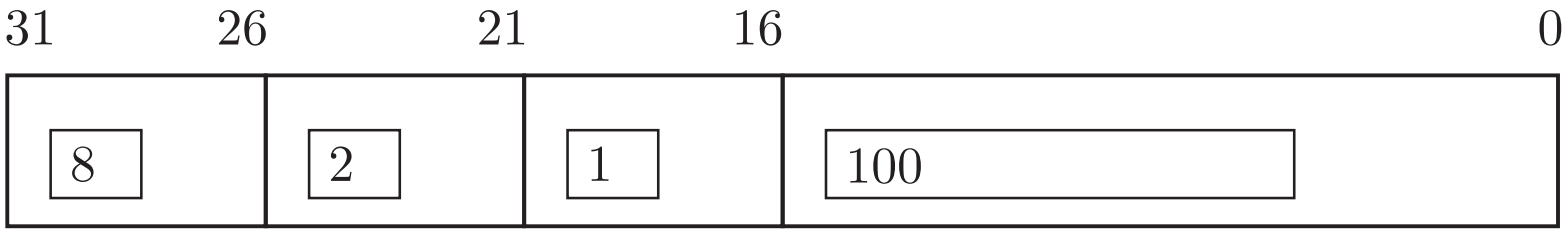


  Jump Register $PC <-- \$31$

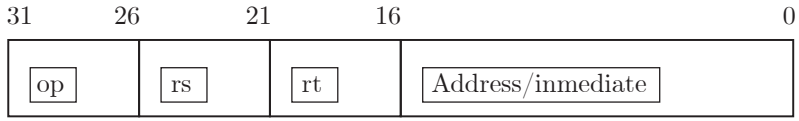
Formato I: Transferencias inmediatas



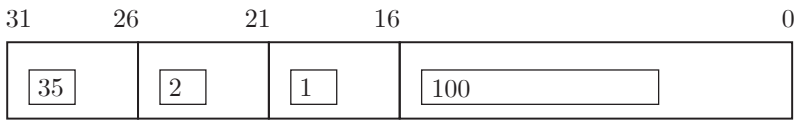
- Addi \$1,\$2,100
  \$1 = \$2 + 100



Formato I: Transferencias

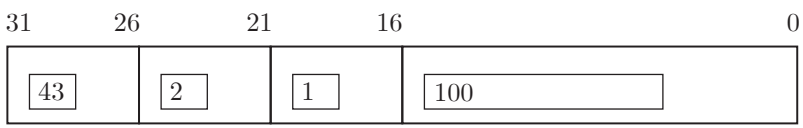


- Beq$1,$2,100



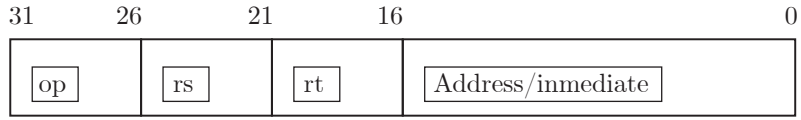
Load Word $\$1 = M[\$2 + 100]$

- Sw $1, 100($2)

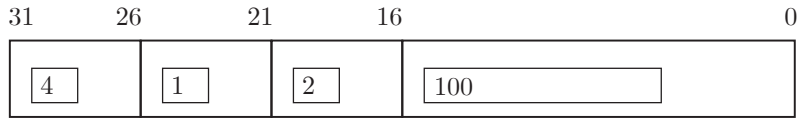


Store Word $M[\$2 + 100] = \$1$

Formato I: Saltos Inmediatos



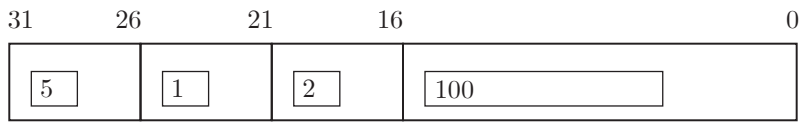
- Beq$1,$2,100



Branch Equal

si ($1 = $2) *entonces ir a* $PC + 4 + 100$

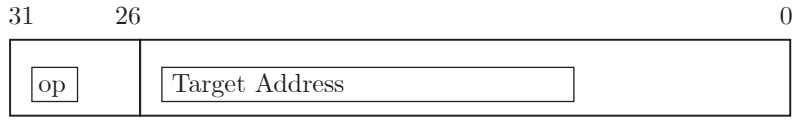
- Bne$1,$2,100



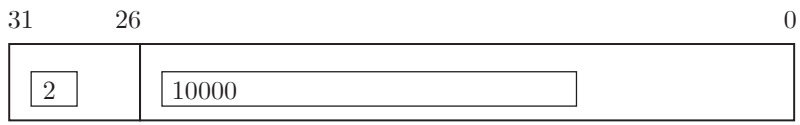
Branch Not Equal

si ($1 = $2) *entonces ir a* $PC + 4 + 100$

Formato J: Transferencias inmediatas

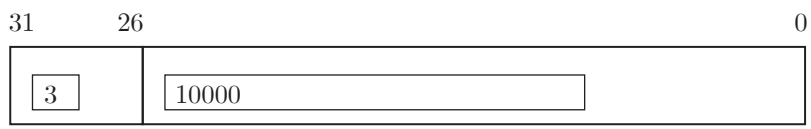


- J 10000



$PC_{27.,2} = 10000 \quad PC_{1.,0} = 00$

- Jal 10000



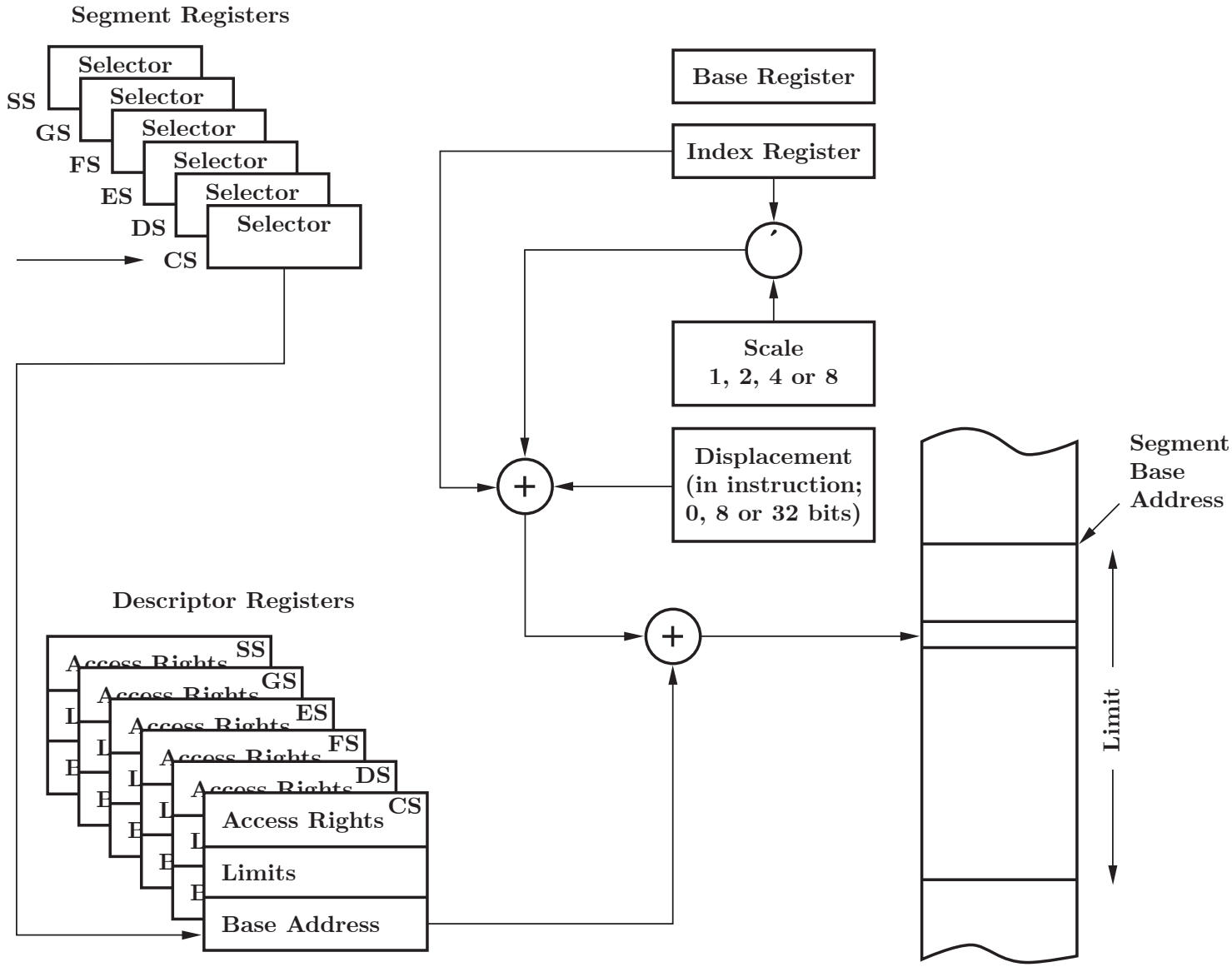
JUMP AND LINK $31 = PC + 4$

$PC_{27.,2} = 10000 \quad PC_{1.,0} = 00$

Pentium Addressing Modes

- Direcciona usando Segmento + Offset
  - Segmento + Offset = Dirección Plana
- 12 modos disponibles
  - Immediate
  - Register operand
  - Displacement
  - Base
  - Base with displacement
  - Scaled index with displacement
  - Base with index and displacement
  - Base scaled index with displacement
  - Relative

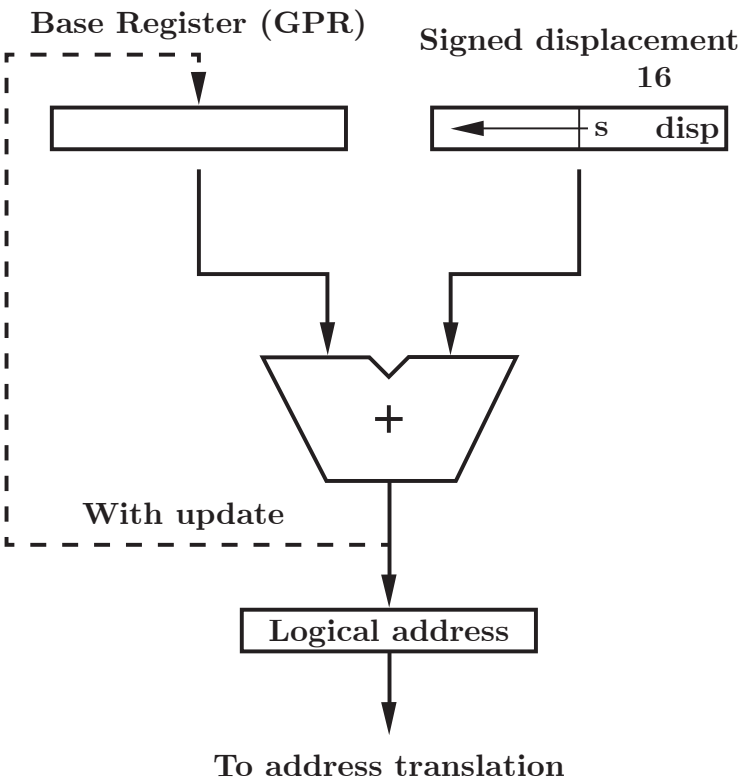
Pentium Addressing Modes



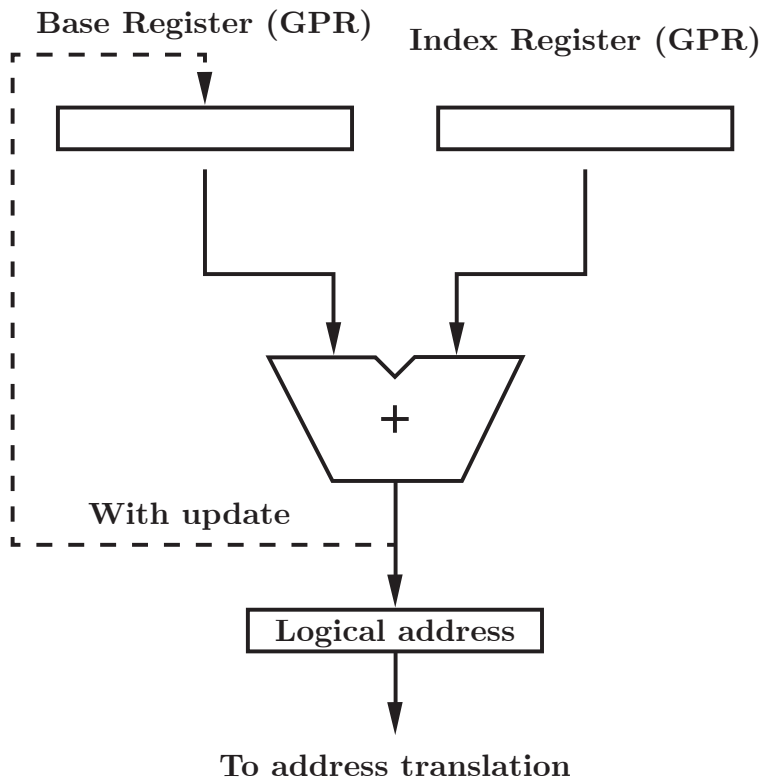
PowerPC Addressing Modes

- Load/store
  - Register Indirect + Desplazamiento
    - Las instrucciones incluyen 16 bit de desplanzamiento a ser sumado a un registro base (seteable)
  - Indirect indexed
    - Un registro de base y otro de indice
- Saltos
  - Absoluto
  - Relativo
  - Indirecto
- Arithmetic
  - Operandos en registros o en la instrucción

PowerPC Memory Operand Addressing Modes



(a) Indirect Addressing



(b) Indirect Indexed Addressing

PowerPC Addressing Modes

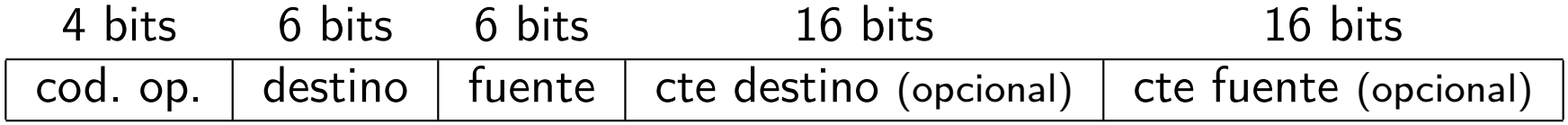
- Tanenbaum – Capítulo 5
- Stalling – Capítulo 11
- Null - Capítulo 5

Arquitectura Orga 1: descripción general

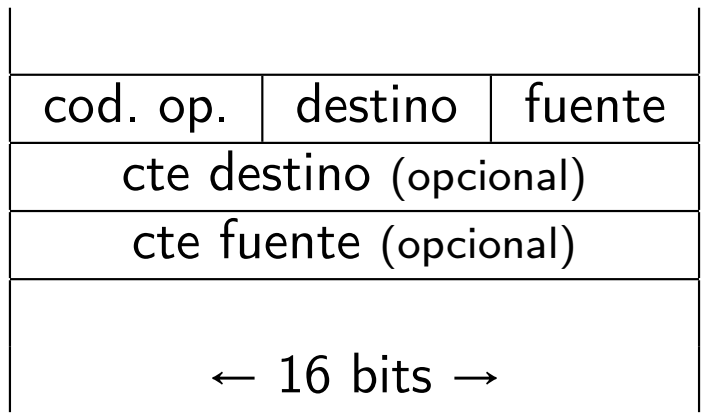
- Palabras de **16 bits**.
- Direccionamiento a palabra.
- Espacio direccionable de **65536** palabras (2^{16}).
- Memoria destinada para entrada/salida en las direcciones **FFF0-FFFF**.
- Ocho registros de propósito general de **16 bits**: R0 a R7.
- **Program counter** (**PC**) de **16 bits**.
- **Stack pointer** (**SP**) de **16 bits**. La pila comienza en la dirección **FFEF**.
- **Flags**: **Z** (**zero**), **N** (**negative**), **C** (**carry**), **V** (**overflow**).
- Todas las instrucciones alteran los **flags**, excepto **MOV**, **CALL**, **RET**, **JMP** y **Jxx**.
- De las que alteran los **flags**, todas dejan **C** y **V** en cero, excepto de **ADD**, **SUB**, **CMP** y **NEG**.
- Todas las operaciones en la ALU se realizan en notación **complemento a 2**.

Formato de instrucción

Formato:



Huella en memoria:



$cccc-ddd, ddd-fff, fff-xxxx, xxxx, xxxx, xxxx-yyyy, yyyy, yyyy, yyyy$

Modos de direccionamiento

Modo	**Codificación**	**Resultado**
Inmediato	000000	c16
Directo	001000	[c16]
Indirecto	011000	[[c16]]
Registro	100*rrr*	R*rrr*
Indirecto registro	110*rrr*	R*rrr*
Indexado	111*rrr*	[R*rrr* + c16]

- c16 es una constante de 16 bits.
- R*rrr* es el registro indicado por los tres últimos bits de la codificación.
- Las instrucciones que tienen como destino un operando de tipo *inmediato* son consideradas como inválidas por el procesador.

Tipo 1: Instrucciones de dos operandos

operación	**cod. op.**	**efecto**
MOV d, f	0001	$d \leftarrow f$
ADD d, f	0010	$d \leftarrow d + f$
SUB d, f	0011	$d \leftarrow d - f$
AND d, f	0100	$d \leftarrow d$ and f
OR d, f	0101	$d \leftarrow d$ or f
CMP d, f	0110	Modifica los *flags* según el resultado de $d - f$.

Tipo 1: Ejemplos instrucciones de dos operandos

- **MOV R2,0** ; llena el registro 2 con 0.
  modo de direccionamientos:
  - destino: *Registro*
  - fuente: *Inmediato*
  - Huella en memoria:
    0001 – 100,010 – 000,000 – 0000,0000,0000,0000

Tipo 1: Ejemplos instrucciones de dos operandos

- **MOV R2,0** ; llena el registro 2 con 0.
  modo de direccionamientos:
  - destino: *Registro*
  - fuente: *Inmediato*
  - Huella en memoria:
    0001 – 100,010 – 000,000 – 0000,0000,0000,0000
- **ADD R4,R5** ; suma el contenido del registro 4 con el contenido del registro 5, el resultado queda en R4.
  modo de direccionamientos:
  - destino: *Registro*
  - fuente: *Registro*
  - Huella en memoria:
    0010 – 100, 100 – 100, 101

Tipo 1: Ejemplos instrucciones de dos operandos

- **AND [PEPE],[[TONI]]** ; función lógica AND entre el contenido de la dirección de memoria PEPE con el contenido de la dirección de memoria en TONI, el resultado se guarda en la dirección de memoria PEPE.
  modo de direccionamientos:
  - destino: *Directo*
  - fuente: *Indirecto*
  - Huella en memoria:
    $0100 - 001,000 - 011,000 - PPPP, PPPP, PPPP, PPPP - TTTT, TTTT, TTTT, TTTT$

Tipo 2: Instrucciones de un operando

- *Tipo 2a*: instrucciones de un operando destino.

4 bits	6 bits	6 bits	16 bits
cod. op.	destino	000000	constante destino (opcional)

operación	**cod. op.**	**efecto**
NEG d	1000	$d \leftarrow 0 - d$
NOT d	1001	$d \leftarrow \overline{d}$ (bit a bit)

Tipo 2a: ejemplos instrucciones de un operando destino

- **NEG [PEPE]** ; niega el contenido de la dirección de memoria PEPE.
  modo de direccionamientos:
  - destino: *Directo*
  - fuente: NA
  - Huella en memoria:
    $1000 - 001,000 - 000,000 - PPPP, PPPP, PPPP, PPPP$

Tipo 2: Instrucciones de un operando

- *Tipo 2b*: instrucciones de un operando fuente.

4 bits	6 bits	6 bits	16 bits
cod. op.	000000	fuente	constante fuente (opcional)

operación	**cod. op.**	**efecto**
JMP f	1010	$PC \leftarrow f$
CALL f	1011	$[SP] \leftarrow PC$, $SP \leftarrow SP - 1$, $PC \leftarrow f$

Tipo 2b: ejemplos instrucciones de un operando fuente

- **JMP OTRO** ; salta a la dirección de memoria OTRO.
  modo de direccionamientos:
  - destino: NA
  - fuente: *Inmediato*
  - Huella en memoria:
    $1010 - 000,000 - 000,000 - RRRR, RRRR, RRRR, RRRR$

Tipo 3: Instrucciones sin operandos

- *Tipo 3*: instrucciones sin operandos.

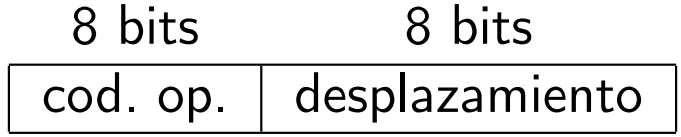
4 bits	6 bits	6 bits
cod. op.	000000	000000

operación	**cod. op.**	**efecto**
RET	1100	$PC \leftarrow [SP+1]$, $SP \leftarrow SP+1$

Tipo 3: ejemplo de instrucciones sin operandos

- **RET** ; retorna de una subrutina.
  modo de direccionamientos:
  - destino: NA
  - fuente: NA
  - Huella en memoria:
    $1100 - 000,000 - 000,000$

Tipo 4: Saltos condicionales



cod. op.	**Operación**	**Descripción**	**Condición**
1111 0001	JE	Igual / Cero	Z
1111 1001	JNE	Distinto	$\overline{Z}$
1111 0010	JLE	Menor o igual	$Z + (N \oplus V)$
1111 1010	JG	Mayor	$\overline{(Z + (N \oplus V))}$
1111 0011	JL	Menor	$N \oplus V$
1111 1011	JGE	Mayor o igual	$\overline{N \oplus V}$
1111 0100	JLEU	Menor o igual sin signo	$C + Z$
1111 1100	JGU	Mayor sin signo	$\overline{C + Z}$
1111 0101	JCS	Carry / Menor sin signo	C
1111 0110	JNEG	Negativo	N
1111 0111	JVS	Overflow	V

Tipo 4: ejemplo de instrucciones de saltos condicionales

Las instrucciones en este formato son de la forma *Jxx* (salto relativo condicional). Si al evaluar la condición de salto en los *flags* el resultado es *1*, el efecto es incrementar el *PC* con el valor de los *8 bits* de desplazamiento (*DDDD*, *DDDD*), representado en *complemento a 2* de *8 bits*. En caso contrario la instrucción no produce efectos, por lo cual el programa sigue con la instrucción siguiente.

Tipo 4: ejemplo de instrucciones de saltos condicionales

Las instrucciones en este formato son de la forma *Jxx* (salto relativo condicional). Si al evaluar la condición de salto en los *flags* el resultado es *1*, el efecto es incrementar el *PC* con el valor de los *8 bits* de desplazamiento (*DDDD*, *DDDD*), representado en *complemento a 2* de *8 bits*. En caso contrario la instrucción no produce efectos, por lo cual el programa sigue con la instrucción siguiente.

- **JE SIGUE** ; si la operación anterior dió "igual" (encendió el flag correspondiente) salta a SIGUE.
  modo de direccionamientos:
  - destino: NA
  - fuente: NA
  - Huella en memoria:
    1111, 0001 – *DDDD*, *DDDD*