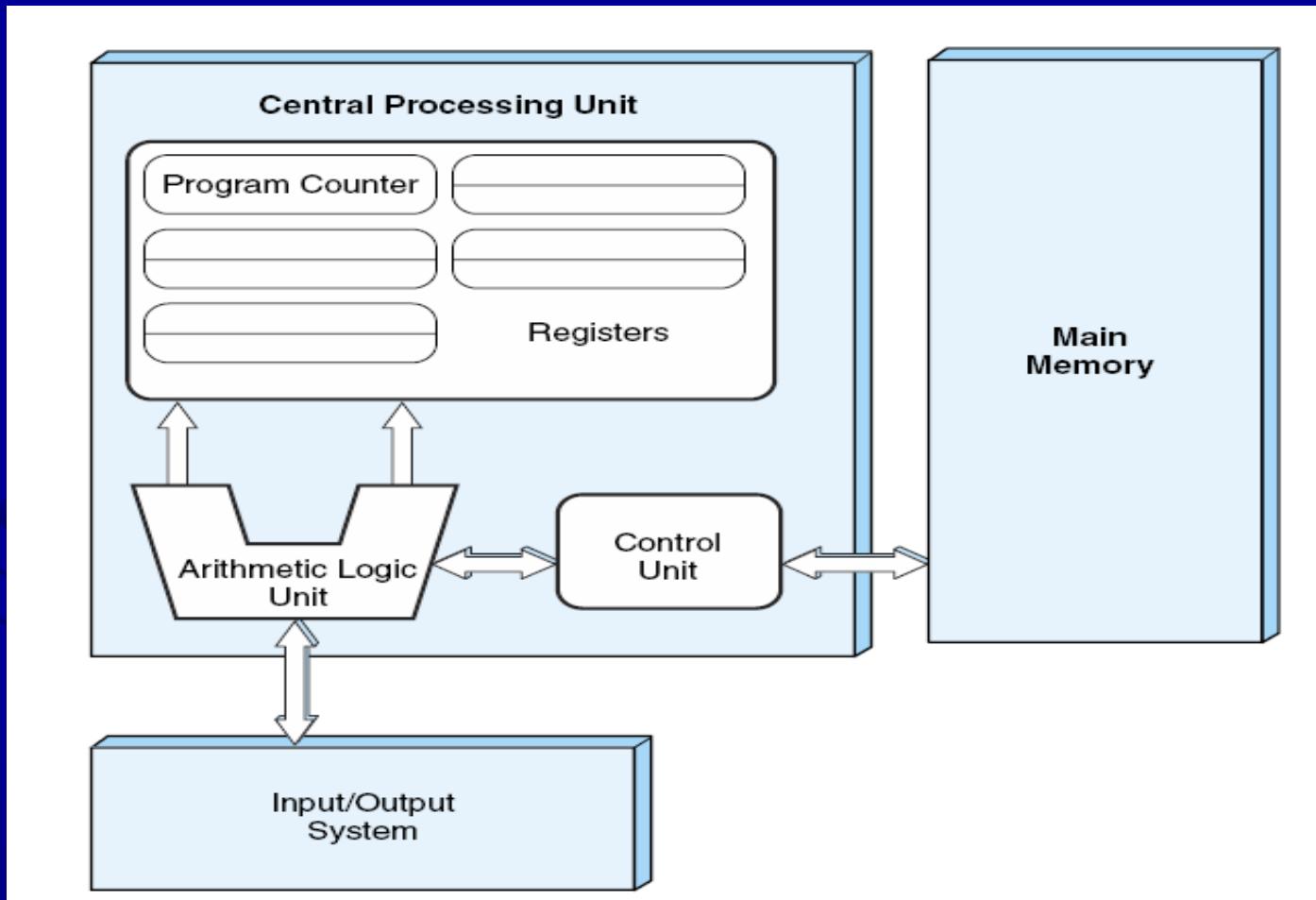


Organización del Computador 1

CPU (ISA)– Conjunto de
Instrucciones de la Arquitectura

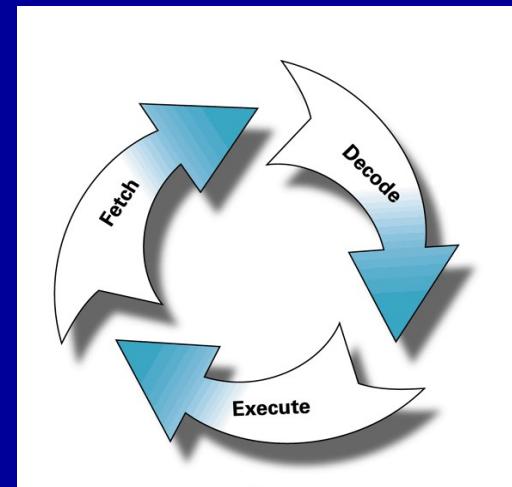
2017

Estructura de una máquina von Neumann

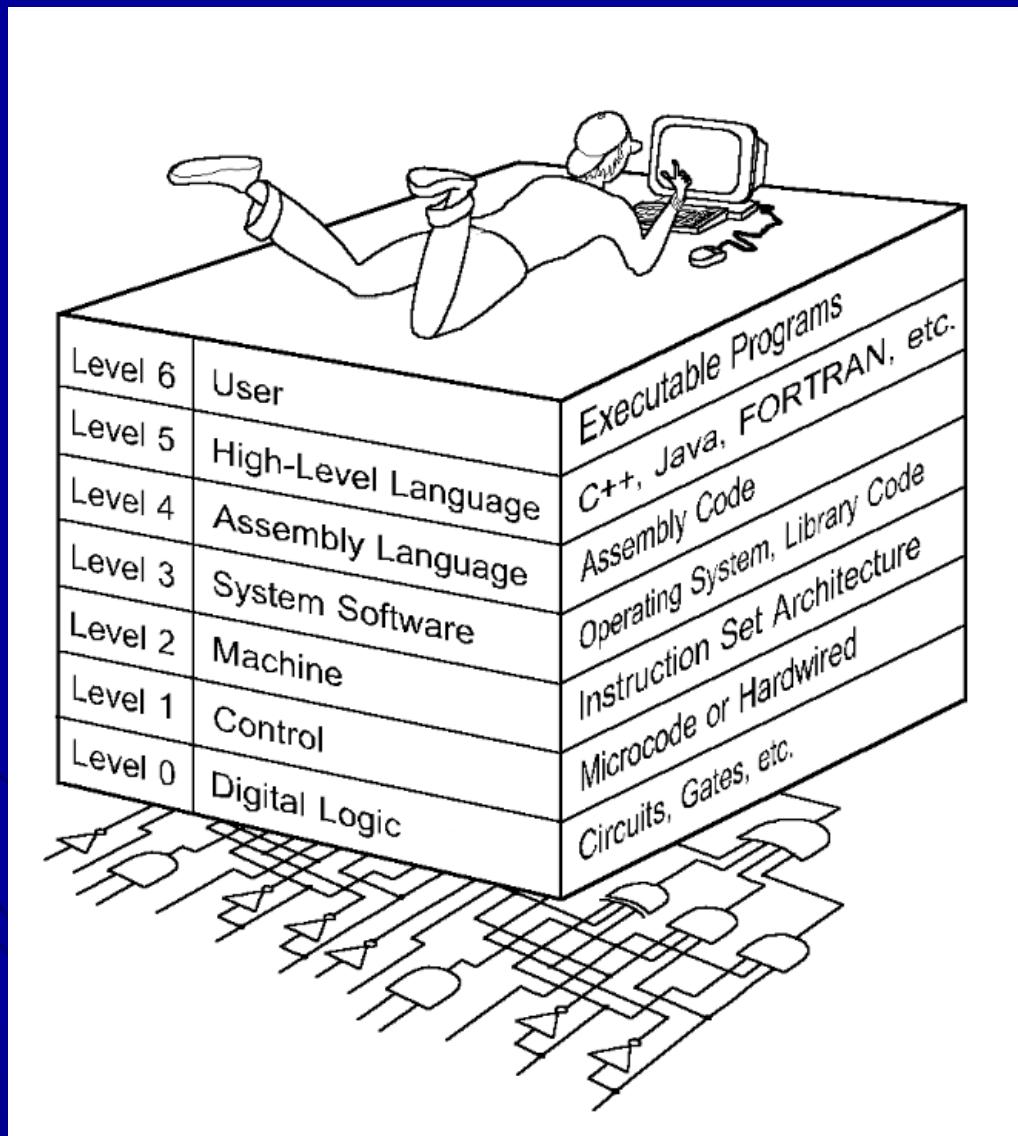


Ciclo de Ejecución

1. UC obtiene la próxima instrucción de memoria (usando el registro PC).
2. Se incrementa el PC.
3. La instrucción es decodificada a un lenguaje que entiende la ALU.
4. Obtiene de memoria los operandos requeridos por la operación.
5. La ALU ejecuta y deja los resultados en registros o en memoria.
6. Repetir paso 1.



Los niveles de una computadora



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.

Métricas de una ISA

- Cantidad de memoria que un programa requiere.
- Complejidad del conjunto de instrucciones (por ejemplo RISC vs CISC).
- Longitud de las instrucciones.
- Cantidad total de instrucciones.

Cómo se representan los datos?

- 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 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

- Ejemplo: entero de dos bytes, Byte 0 menos significativo, Byte 1 más significativo.
- “*Little endian*”:
Base address + 0 = Byte 0
Base address + 1 = Byte 1
- “*Big endian*”:
Base address + 0 = Byte 1
Base address + 1 = Byte 0

Acceso a los datos

- Donde se Almacenan?
 - Registros
 - Memoria
 - Stack
 - Espacio de I/O
- Como se acceden?
 - Modos de Direccionamiento

Operaciones

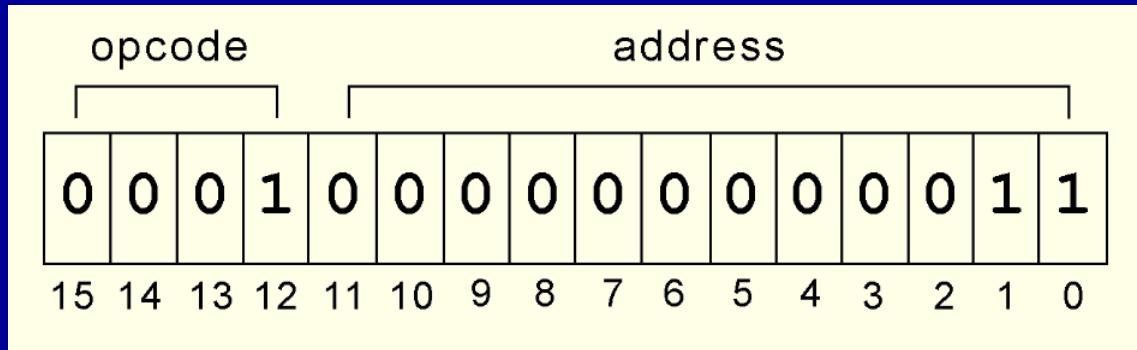
- Movimiento de datos (Move, Load, Store,...)
- Aritméticas (Add, Sub, ...)
- Lógicas (And, Xor, ...)
- I/O.
- Transferencia de Control (Jump, Skip, Call...)
- Específicas
 - Ejemplo: Multimedia

Codificación

- Códigos de operación (**OpCode**)
 - Representa la operación ...
- Operando/s Fuente
 - A realizar sobre estos datos ...
- Operando Resultado
 - Pone la respuesta aquí ...
- Referencia a la próxima instrucción
 - Cuando termina sigue por aquí ...

Ejemplo: Marie

- Instrucción LOAD en el IR:



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

Características de ISAs

- Tipos típicos de arquitecturas:
 1. Orientada a Stack.
 2. Con acumulador.
 3. Con registros de propósito general.
- Los “tradeoffs”:
 - Simpleza del hardware.
 - Velocidad de ejecución.
 - Facilidad de uso.

Características de ISAs

- Arquitectura Stack: las instrucciones y operandos son tomados implícitamente del *stack*
 - El *stack* no puede ser accedido de forma aleatoria (sino a través de un orden).
- Arquitectura acumulador: En cualquier operación **binaria** un operando esta implícito (el acumulador)
 - El otro operando suele ser la memoria, generando cierto tráfico en el bus.
- Arquitectura con registros de propósito general (GPR): los registros pueden ser utilizados en lugar de la memoria
 - Más rápido que la de acumulador.
 - Implementaciones eficientes usando compiladores.
 - Instrucciones más largas... (2 ó 3 operandos).

Instrucciones (Posibles)

	Stack	Acumulador	Registros
Movimiento	PUSH X POP X	LOAD X STORE X	Mov R1,R2 Load R1,X Store X,R1
Aritméticas	Add Sub	Add X Sub X	Add R1,R2,R3 Sub R1,R2,R3
Lógicas	And Or Not LE GE Eq	And X Or X Not	And R1,R2,R3 Or R1,R2,R3 Not R1, R2 Cmp R1,R2
Control	Jump X JumpT X JumpF X Call/Ret X	Jump X Jump (AC) SkipCond Call X/Ret	Jump X Jump R Jump Cond R Jump Cond X Call X/R Ret

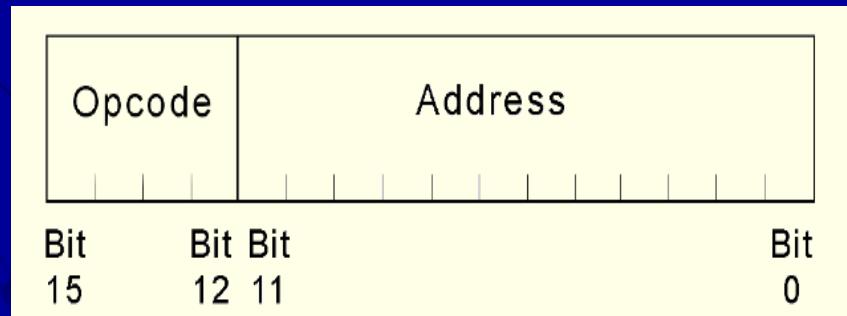
MARIE

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



MARIE

OpCode	Instrucción	
0000	JnS X	Almacena PC en X y Salta a X+1
0001	Load X	AC = [X]
0010	Store X	[X]= AC
0011	Add X	AC = AC + [X]
0100	Subt X	AC = AC - [X]
0101	Input	AC = Entrada de Periférico
0110	Output	Enviar a un periférico contenido AC
0111	Halt	Detiene la Ejecución
1000	SkipCond Cond	Salta una instrucción si se cumple la condición (00=>AC<0; 01=>AC=0; 10=>AC>0))
1001	Jump Dir	PC = Dir
1010	Clear	AC = 0
1011	Addi X	AC = AC + [[X]]
1100	Jumpi X	PC = [X]

Stack Machines

- Las Stack Machines **no usan** operandos en las instrucciones
- Salvo **PUSH x** y **POP x** que requieren una dirección de memoria como operando.
- El resto obtienen los operandos del stack.
- **PUSH** y **POP** operan sólo con el tope del stack.
- Operaciones binarias (ej. **ADD**, **MULT**) usan los 2 primeros elementos del stack.

Stack Machines

- Utilizan notación polaca inversa
 - Jan Lukasiewicz (1878 - 1956).
 - Notación infija: $Z = X + Y.$
 - Notación postfija: $Z = X Y +$
- No necesitan paréntesis!
 - infija $Z = (X \times Y) + (W \times U),$
 - postfija $Z = X Y \times W U \times +$

Comparativa (Asignaciones)

$$Z = X \times Y + W \times U$$

Stack	1 Operando	Registros 2 Operandos	Registros 3 Operandos
-------	------------	--------------------------	--------------------------

PUSH X	LOAD X	LOAD R1 , X	
PUSH Y	MULT Y	MULT R1 , Y	
MULT	STORE TEMP	LOAD R2 , W	MULT R1 , X , Y
PUSH W	LOAD W	MULT R2 , U	MULT R2 , W , U
PUSH U	MULT U	ADD R1 , R2	ADD Z , R1 , R2
MULT	ADD TEMP	STORE Z , R1	
ADD			
POP Z			

Modos de Direcccionamiento

Instrucción: OpCode + Operandos

Que tipos de cosas pueden ser los operandos?

Constantes

Referencia a Variables

Referencia a Arrays

Referencias a subrutinas

Estructuras de datos (Listas, Colas)

OpCode Op1

Op2 Op3

Modos de Direcccionamiento

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

Inmediato

OP	N
----	---

- El operando es parte de la instrucción (N)
- Ej: ADD 5
 - $AC = AC + 5$
 - 5 es un operando
- Ej2: Jump 110
- No hay referencia adicional a memoria
- Rápido
- Rango limitado

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

Directo

Instruction

Opcode	Address A
--------	-----------

Memory



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

Indirecto

Instruction

Opcode

Address A

Memory

Pointer to operand

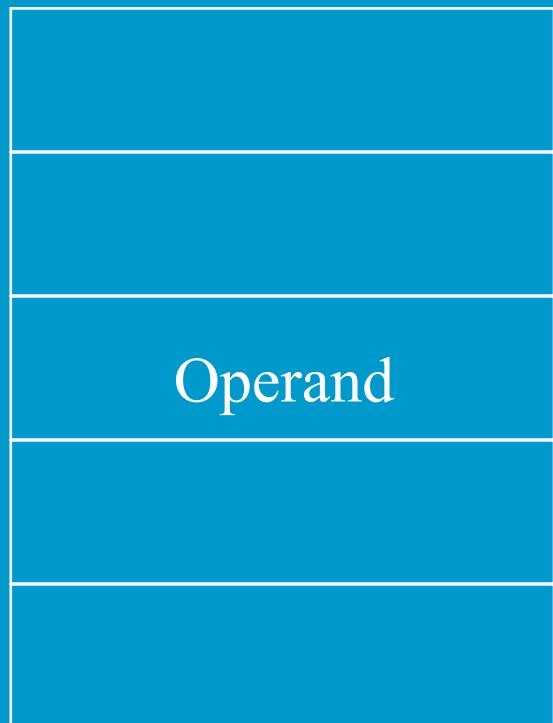
Operand

Registro

Instruction

Opcode	Register Name
--------	---------------

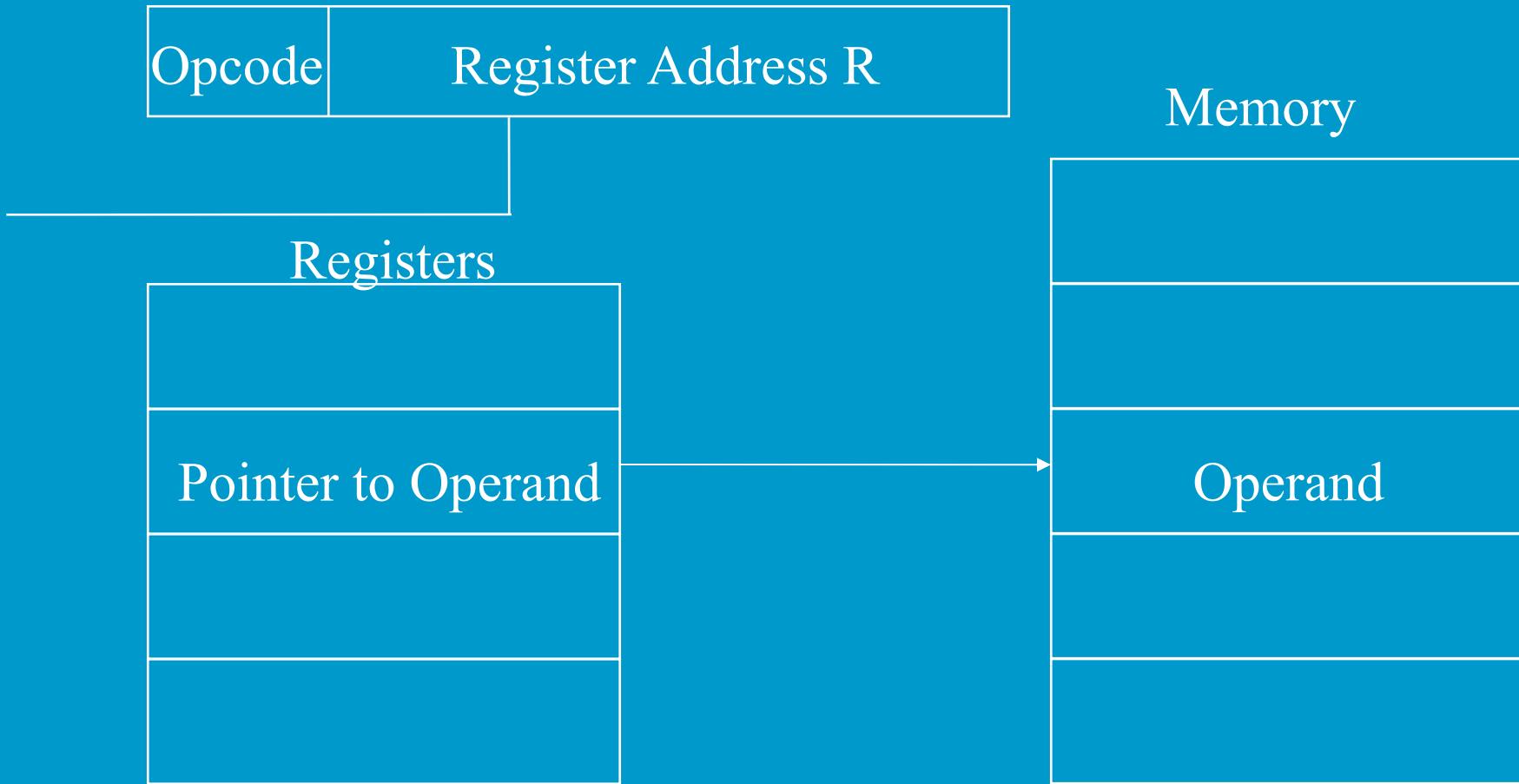
Registers



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

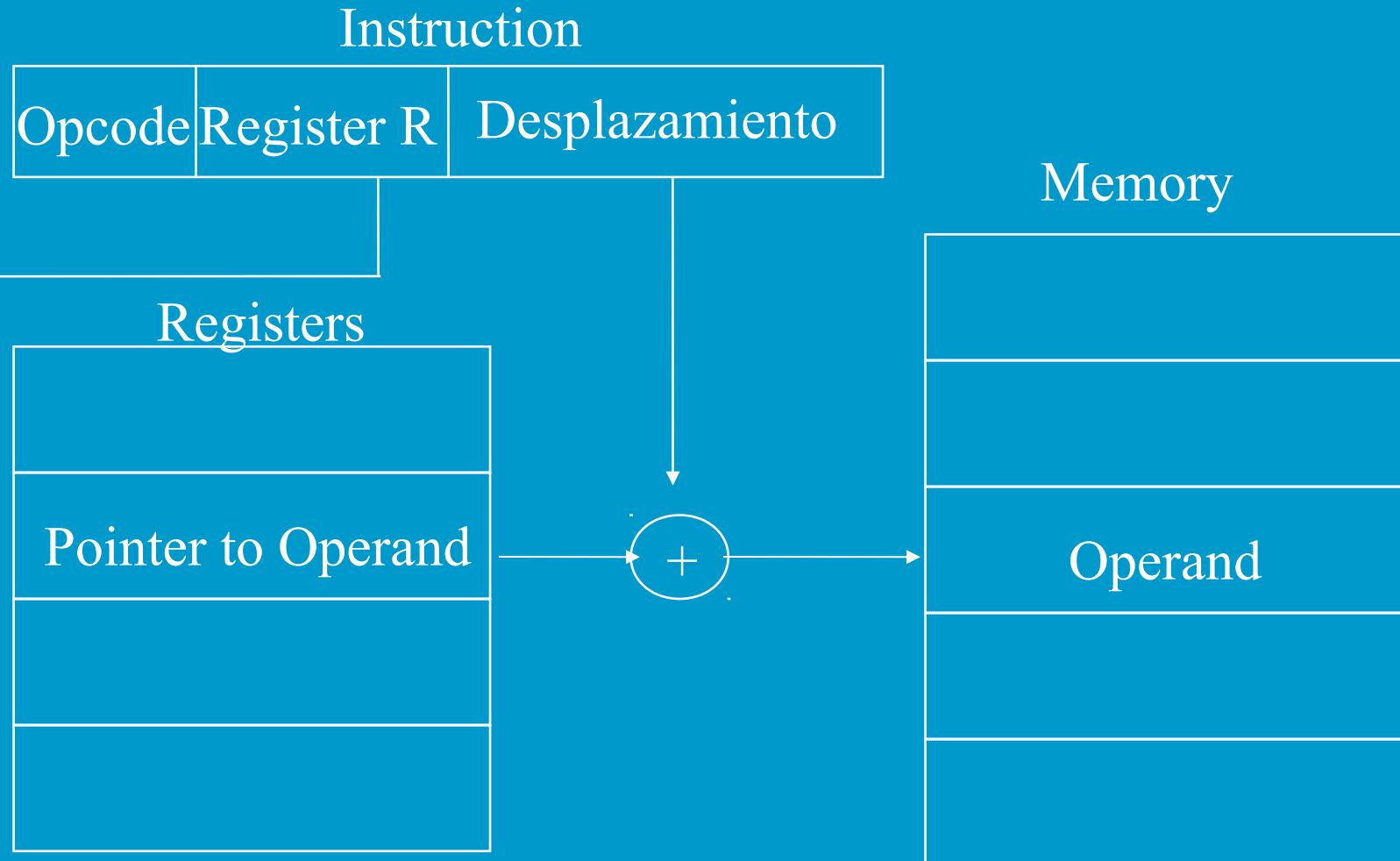
Registro Indirecto



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

Desplazamiento



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

Modo de Direccionamiento

Inmediato

Mov R, #N

R ← N

Directo

Mov R, [A]

R ← mem[A]

Register

Mov R1,R2

R1 ← R2

Register Indirect

Mov R1,[R2]

R1 ← mem[R2]

Displacement

Mov R1, [R2+D]

R1 ← mem[R2+ D]

Base-Register Displacement

Mov R1, D (R_{base})

R1 ← mem[R_{base} + D]

Indexed

Mov R1, A[R]

R1 ← mem[A+R]

Indexed Scaled

Mov R1, A[R*Scale]

R1 ← mem[A+R*Scale]

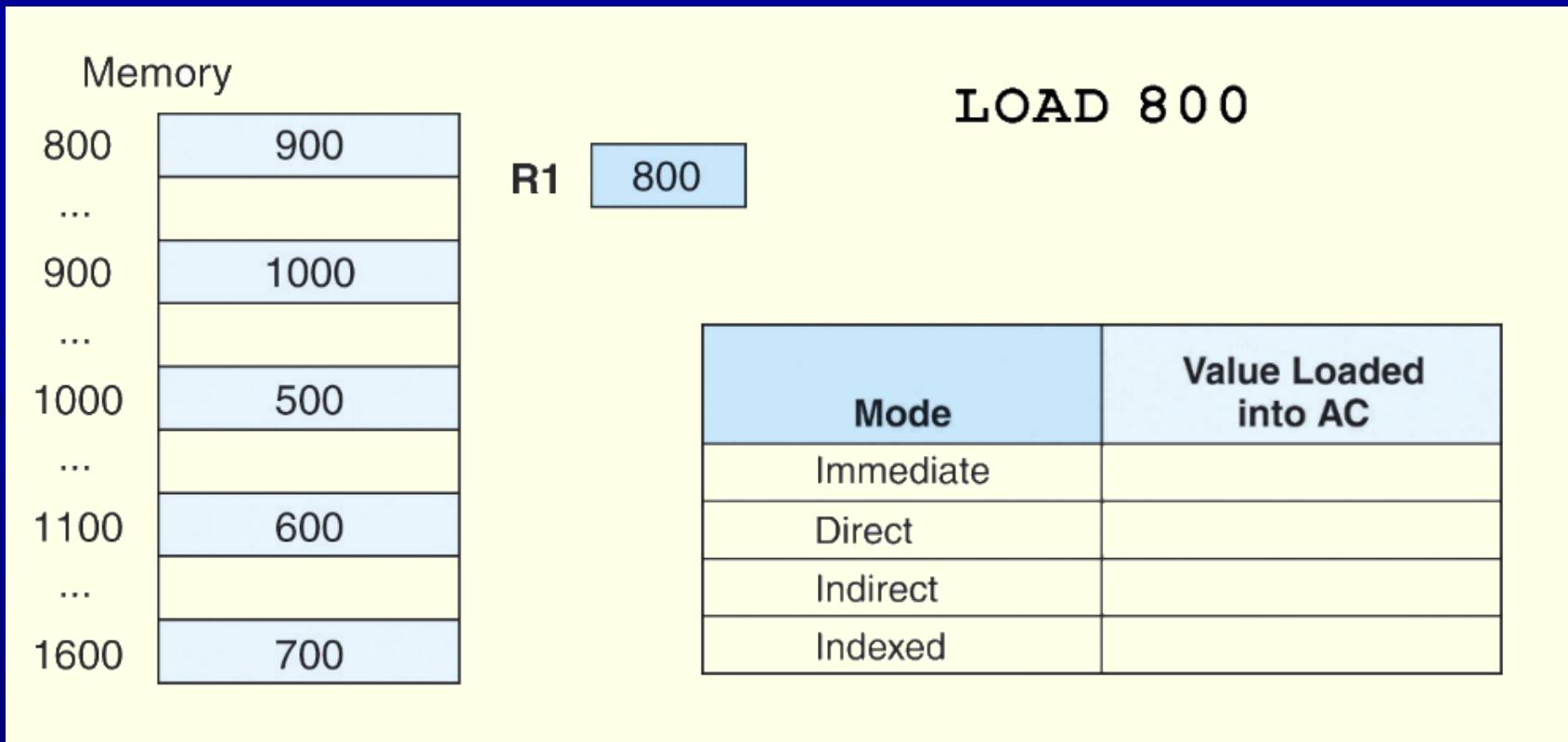
PC Relative

Jump N

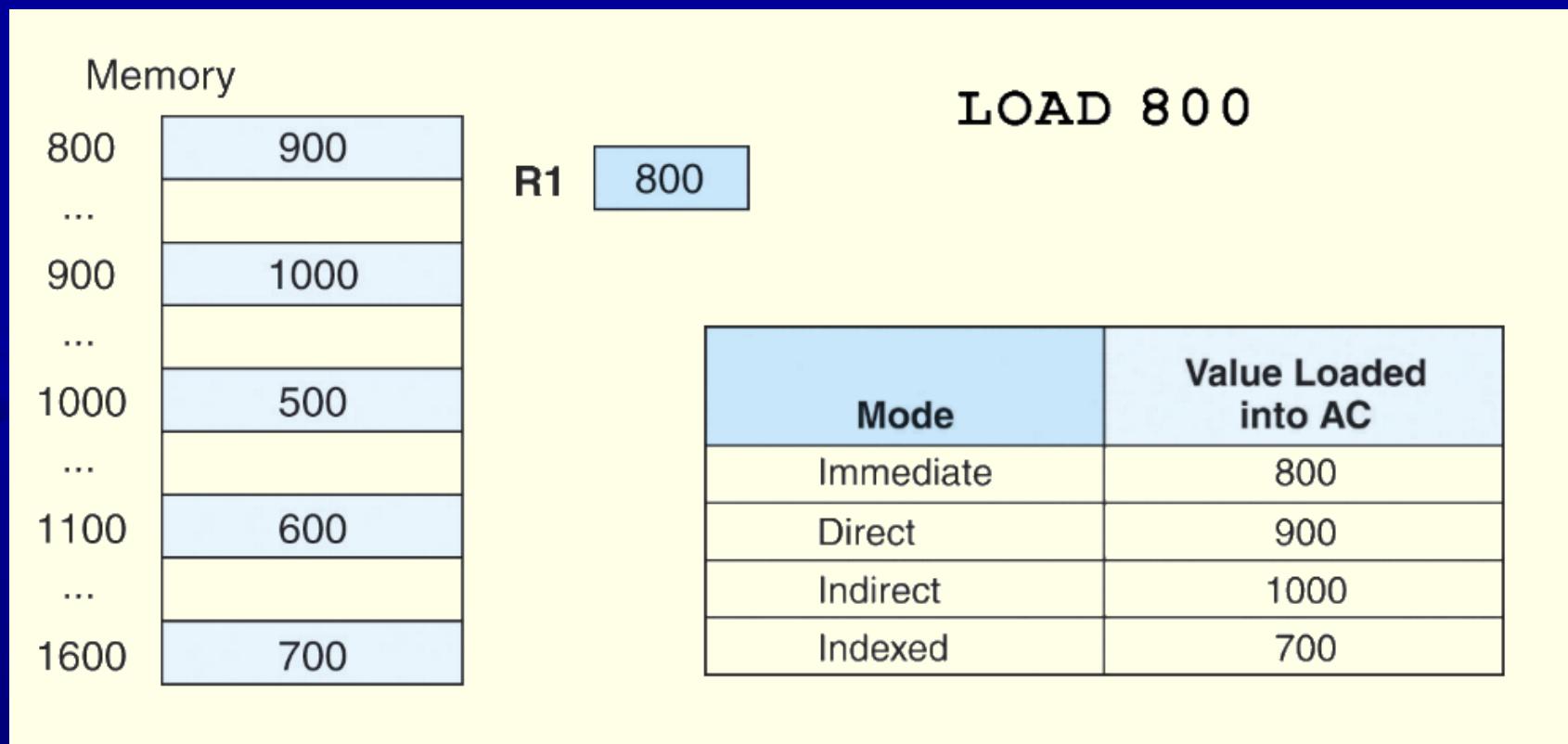
PC ← PC + N

Ejemplo

- Completar el valor de AC según el modo de direccionamiento



Ejemplo



Diseñando el 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 el ISA

Algunas métricas...

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

$$AC = AC + \text{Dato}$$

0 operandos: Stack Machines

Usan un stack

Muchos operandos (VLIW)

Paralelismo implícito en la instrucción

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	94	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