PARTE 1:
Introducción a los Sistemas Operativos y a su relación con la Arquitectura de Computadores

Sección 3. La CPU: juego de instrucciones

La CPU (Central Processing Unit)

Es el componente fundamental de un ordenador.

Ejecuta las órdenes/instrucciones (en código máquina) de un programa.

Controla y se comunica con la memoria y la E/S.

Se compone de: unidad de control, registros, ALU y bus/es interno/s.

Características: juego de instrucciones, velocidad de procesamiento, tamaño palabra (datos y direcciones),etc.

Se suelen implementar en un chip (microprocesador) que puede contener varias CPU (“cores”) e incluir otro hardware como MMU y memoria caché.

Juego de instrucciones

El juego de instrucciones condiciona el diseño digital de la CPU (la CPU a su vez condiciona el diseño de la placa base).

Un juego más grande y más potente facilita la programación pero complica el diseño digital (y la optimización de los compiladores) y enlentece la velocidad de ejecución.

Los juegos de instrucciones más pequeños constan de... ¡una instrucción! Ej.: subleq

SUBLEQ

Sintaxis (ensamblador): `subleq a, b, c`

Semántica: resta al contenido de la posición de memoria `b` el contenido de la posición de memoria `a`. Si el resultado es menor o igual que 0 la ejecución continúa en la dirección de memoria `c`. En otro caso se ejecuta la siguiente instrucción.

`subleq a, b, c` =>

Memoria[`b`] ← Memoria[`b`] - Memoria[`a`]

Si Memoria[b] ≤ 0 ContinuarEn `c`

Si para ≤ 0 también se desea continuar en la siguiente instrucción, no se necesita especificar `c` en ensamblador.

Funcionamiento CPU-SUBLEQ

De forma resumida, la CPU de un ordenador SUBLEQ:
1- Lee la instrucción desde la memoria.
2- Carga los operandos (dos accesos a memoria).
3- Ejecuta la instrucción (resta) y actualiza la condición (¿≤0?)
4- Guarda el resultado (en memoria) y, dependiendo de si se cumplió la condición, la próxima instrucción será la siguiente o la que se encuentre en la dirección `c`.
5- Vuelve al paso 1

Como solo existe una instrucción, no tiene que identificarse de ninguna forma especial ni decodificarse.

Ejemplo SUBLEQ

Sumar al dato de la posición de memoria `b` el dato de la posición de memoria `a`. Nota: el ensamblador permite predefinir un valor en una posición de memoria y referirnos a dicha posición mediante una letra.

```
subleq z, z  ;Memoria[z]←0
subleq a, z  ;Memoria[z]←0-Memoria[a]=-Memoria[a]
subleq z, b  ;Memoria[b]←Memoria[b]-Memoria[z]=
             ;                 Memoria[b]-(-Memoria[a])
```

¿Qué hace esto?

```
subleq b, b
subleq z, z
subleq a, z
subleq z, b
```

SUBLEQ, ej. programa en ensamblador

```
;  Programa: Suma.sq
; Propósito: Suma 20 a 40
;     Autor: Javier Macías
;     Fecha: 25/09/2021
;      Obs.: La ejecución comienza por dir. 0
;            Tamaño palabra: 1 byte.
  subleq a, z
  subleq z, b
bucle:
  subleq z, z, bucle   ; “terminamos”
z: DW 0
a: DW 20
b: DW 40
```

SUBLEQ, ej. programa en cód. máquina

Etiqueta	*Dir. memoria*	Contenido	Instrucción ensamblador
	0	**10**	`subleq a, z`
	1	**9**	
	2	**3**	
	3	**9**	`subleq z, b`
	4	**11**	
	5	**6**	
`bucle`	*6*	**9**	`subleq z, z, bucle`
	7	**9**	
	8	**6**	
`z`	*9*	**0**	
`a`	*10*	**20**	
`b`	*11*	**40**	

Posible implementación CPU-SUBLEQ

Registros:

IP: Instruction Pointer. Almacena la dirección de memoria de la siguiente instrucción a ejecutar. Debe poder incrementarse en el tamaño de una instrucción y cargarse la dirección a la que saltar (multiplexor).

IR: Instruction Register. Almacena la instrucción a ejecutar.

MAR: Memory Access Register. Almacena la dirección de memoria a la cual se quiere acceder.

MBR: Memory Buffer Register. Almacena un dato para ser escrito o leído de la memoria.

SR: Status Register. Almacena el resultado de la última operación (para subleq se usa un solo bit que indica si el resultado de la resta fue menor o igual que cero; para otras CPU hay bits para indicar resultado cero, signo, acarreo, desbordamiento, interrupciones activadas, etc.).

Dos registros para almacenar los operandos (los datos recuperados de Memoria[a] y Memoria[b]) y uno para almacenar el resultado de la resta.

ALU:

Solo se precisa en este caso que tenga una operación (la resta). Se podría utilizar también para incrementar el

valor de IP (en cuyo caso también sumaría), aunque se suele dedicar un circuito específico para esto.

Unidad de control (CU):
A partir de la instrucción (en este caso, única) y, cuando procede, de condiciones internas (registro SR) o externas (en este caso no hay; en otros puede ser por ejemplo la solicitud de interrupción), genera las señales necesarias a los diversos componentes internos y externos (control multiplexores, selección operación de la ALU, puertas triestado, señales del bus del sistema, etc.) para cada suboperación/subciclo de la instrucción.

El reloj (o algún circuito basado en él) se encarga de marcar el momento de cada subciclo.
La lógica de la CU puede estar “cableada” (hecha con puertas lógicas) o microprogramada (basada en un programa almacenado en una memoria interna).

Bus:
Uno o más buses internos interconectan los distintos elementos de la CPU.

Algunas consideraciones sobre SUBLEQ

Solo una instrucción y un modo de direccionamiento es poco eficiente.

Los accesos a memoria son más lentos que los accesos a registros, por lo que interesa tener registros de propósito general en la CPU (de uno a varias decenas). Las CPU modernas suelen tener registros enteros y de coma flotante (solo trataremos aquí los enteros).

Como el bus del sistema único, solo se puede hacer un acceso a memoria (o a E/S) cada vez.

Importante disponer de “subprogramas”. Que la CPU gestione una pila facilita (y acelera) su utilización. Precisa un nuevo registro: SP (Stack Pointer).

Tipos de juegos de instrucciones

CISC: Complex instruction set computers. Muchas instrucciones, de distintos tamaños, que permiten hacer operaciones complejas (que pueden tardar uno o más ciclos de reloj) y con múltiples modos de direccionamiento. UC compleja (microprogramada). Ej.: Intel x86

RISC: Reduced instruction set computers. Generalmente menos instrucciones, de tamaño fijo y ejecución en un ciclo de reloj, operaciones sencillas. UC "sencilla" (cableada). Ej.: ARM.

Componentes de una instrucción cod.máq.

Código de operación (*opcode*). Especifica la operación (suma, lectura de un registro de memoria, etc.).

Referencia a operando(s) origen. La mayoría de instrucciones precisan uno o varios operandos origen.

Referencia al operando destino. Indica, cuando procede, dónde se guardará el resultado de una operación.

Referencia a la siguiente instrucción. Por defecto las instrucciones se ejecutan de forma consecutiva en memoria, pero es posible referenciar otra dirección, ya sea incondicionalmente o condicionada (generalmente a algún flag del SR).

Clasificación instrucciones cód. máq.

Vamos a utilizar como ejemplos de instrucciones las del Simulador que utilizaremos en el Bloque 2 de laboratorio:

- Tamaño de palabra de 16 bits, con 4 registros de propósito general (A, B, C y D). Formato memoria: big endian.

- El SR contiene los flags S (Supervisor mode), M (interrupt Mask), C (Carry), Z (Zero), F (Fault) y H (Halt).

Transferencia de datos: Mover, apilar, desapilar...

```
MOV A, 10  ;guarda en A el valor 10
MOV [B], C ;guarda en la pos. de memoria indicada
           ;en B el valor almacenado en C.
PUSH A     ;guarda el valor de A en la pila
```

Lógicas bit a bit: and, or, not...

```
AND A, 0x000F ;guarda en A el resultado A.0x000F
OR B, D       ;guarda en B el resultado B+D
```

Aritméticas: suma, resta, multiplicación, comparación... Estas operaciones pueden afectar a los flags del SR.

```
ADD A, 10 ;guarda en A el resultado de A+10
DEC B     ;decrementa el valor de B en uno
CMP A, B  ;compara A con B. Para ello, hace A-B
          ;sin almacenar el resultado pero
          ;afecta a los flags:
          ; Operand 1 == Operand 2 => C=0, Z=1
          ; Operand 1 >  Operand 2 => C=0, Z=0
          ; Operand 1 <  Operand 2 => C=1, Z=0
```

Desplazamiento y rotación: shift, rotate... Pueden ser lógicos o aritméticos (conservan signo).

```
SHL A, 1  ;desplazamiento lógico a la izquierda
          ;de 1 posición. Por ejemplo si el
          ;valor de A era 0x0808, tras ejecutarse
          ;queda en A el valor 0x1010.
```

Control de flujo: salto incondicional o condicional, llamada a rutina, retorno de rutina…

```
JZ 0x0100   ;si flag Z=1, salta a dir mem. 0x0100
JMP 0x0200  ;salto incondic. a dir mem. 0x0200
CALL 0x0300 ;salto rutina, dir mem. 0x0300. La
            ;dir. mem. instrucción debajo de CALL
            ;(dir. vuelta) se guarda en la pila
RET         ;retorno de subrutina (la dir.
            ;retorno se saca de la pila)
```

Entrada/salida: in, out… Es el caso de salida independiente (no mapeada en memoria). Solo ejecutables en modo supervisor.

```
IN 0x0006  ;el valor registro de E/S 0x0006 se
           ;guarda en el registro A (implícito)
OUT 0x0003 ;el valor del registro A se guarda en
           ;el registro de E/S 0x0003
```

Sistema: trap, halt… (algunas solo ejecutables en modo usuario o modo supervisor).

```
SVC   ;Llamada al sistema (Supervisor Call)
HLT   ;halt: “para” el sistema (se puede salir de
      ;este estado por una interrupción)
STI   ;Activa interrupciones (flag M de SR)
```

Varias: Otras instrucciones, como NOP (NO oPeration; el simulador carece de ella).

Ej. código máquina Simulador

Sea la instrucción en ensamblador del Simulador:

```
MOV B, 0xAABB
```

Ensamblada en memoria quedaría (suponemos comienza en la dirección de memoria *i*):

Dir. mem.	**Contenido(hex)**	**Comentario**
i	06	Opcode de MOV para dato inmediato a registro
i+1	01	Código registro B (A=>0, B=>1...)
i+2	AA	Byte alto del dato inmediato
i+3	BB	Byte bajo del dato inmediato

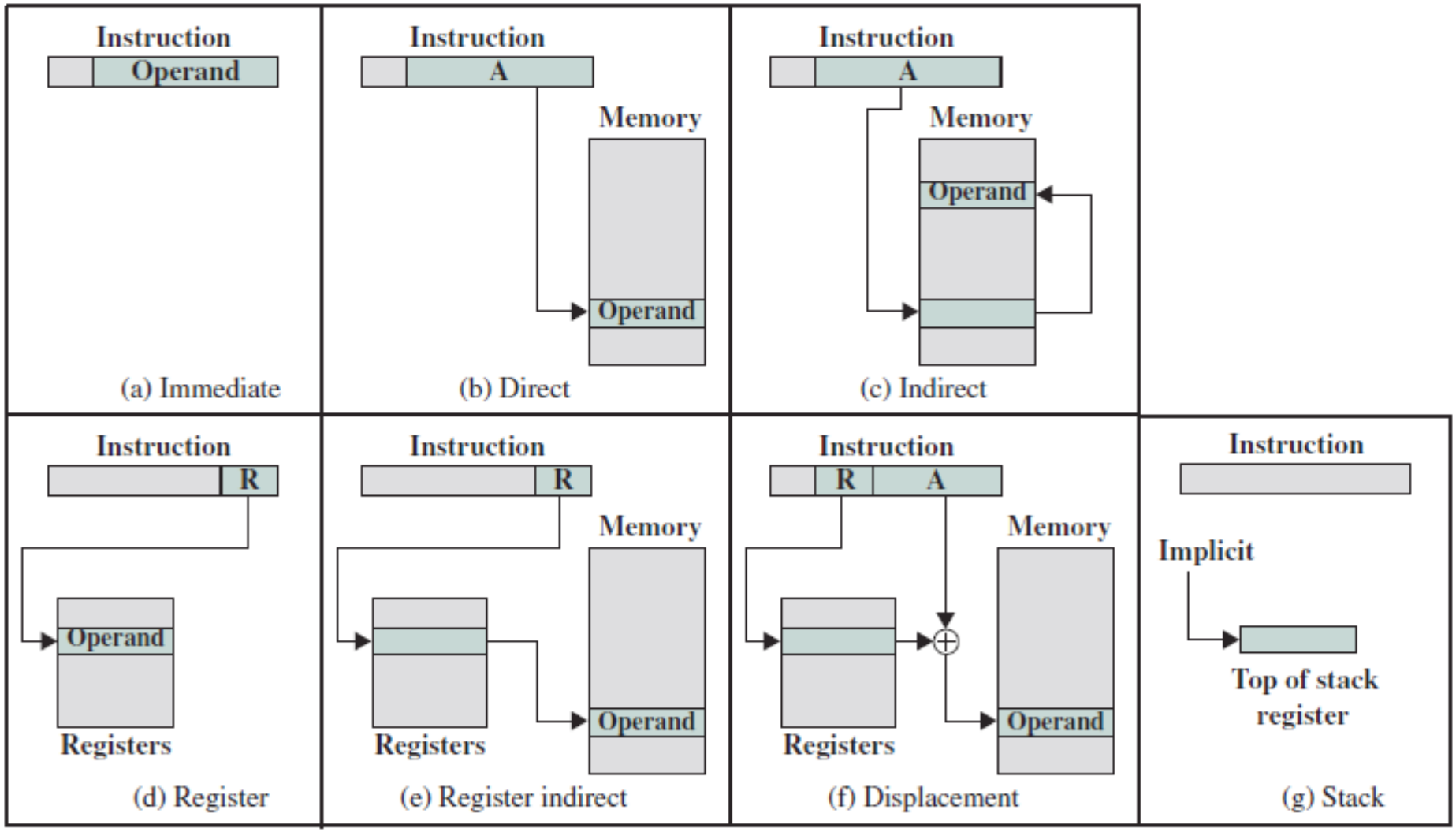
Modos de direccionamiento

Son las formas de especificar e interpretar los operandos de una instrucción.

Hay modos básico (como el mover un valor de un registro a otro) y otros que pueden ser o no implementados en un modelo/familia de CPU dada.

Los modos más complejos/potentes en procesadores CISC.

Modos de direccionamiento más comunes

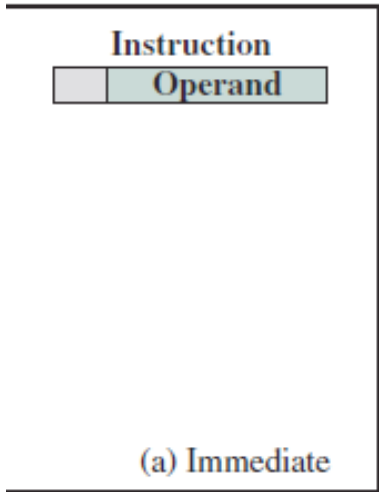


A = contents of an address field in the instruction
R = contents of an address field in the instruction that refers to a register

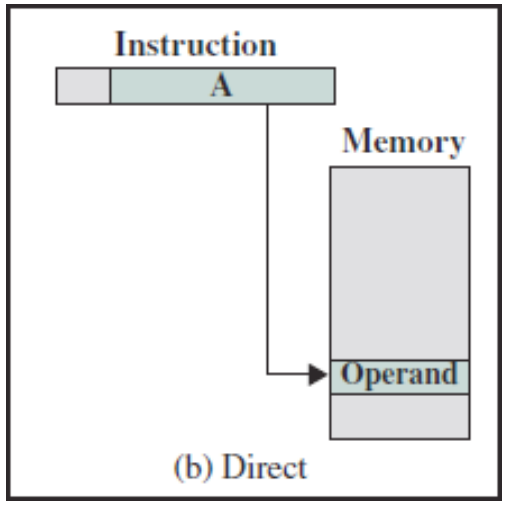
De: Computer Organization and Architecture. W. Stallings. Pearson.

Ejemplos direccionamiento del Simulador

Inmediato: `ADD B, 37`

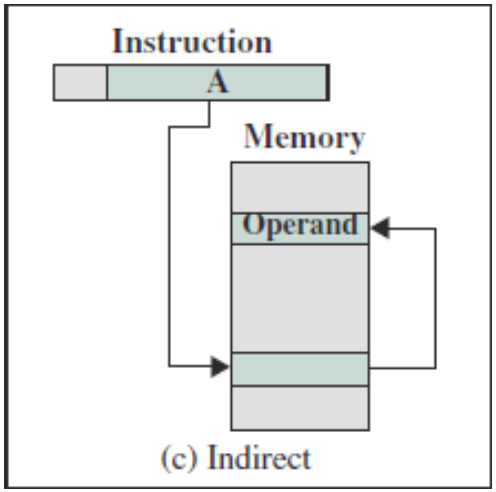


Directo: `ADD B, [37]`

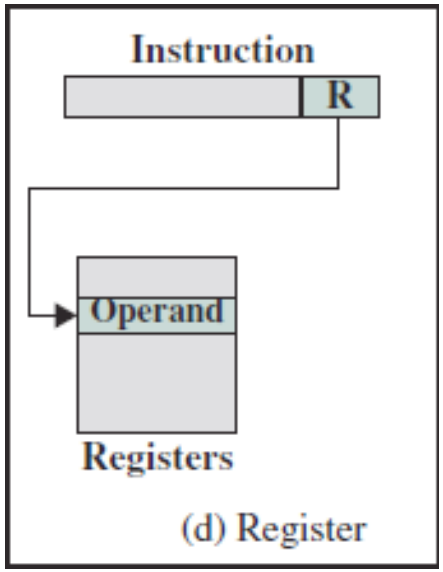


Indirecto: *No soportado.*

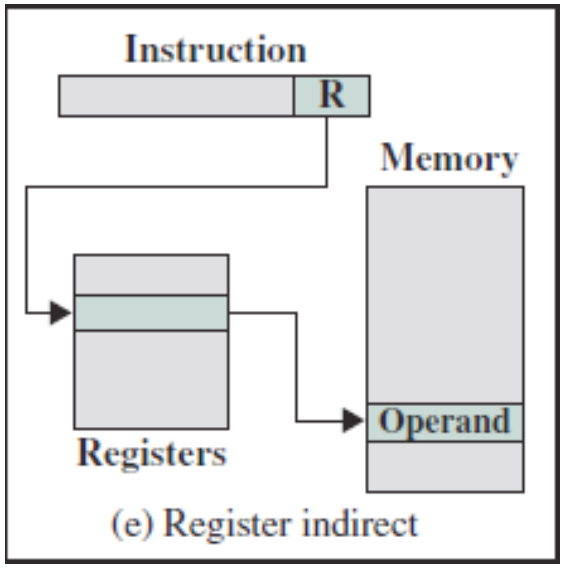
"Equivaldría" a `[[address]]`



Registro: `ADD B, C`

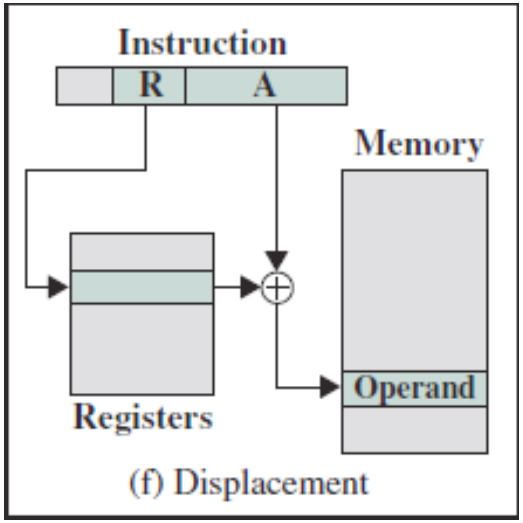


Registro indirecto: *No soportado.*
"Equivale" a `[register]`

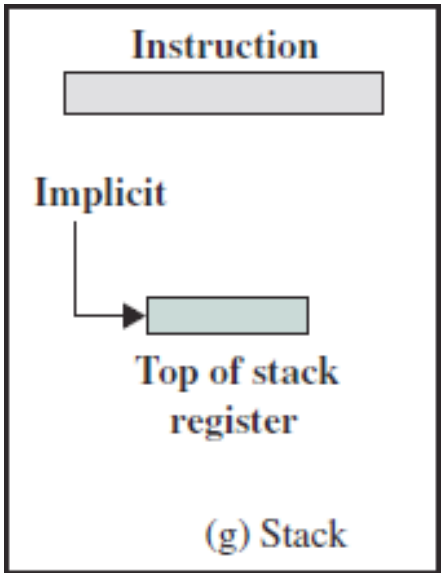


Desplazamiento: `ADD B, [C+4]`

En ensamblador se puede poner
`ADD B, [C]` *que se "traduce" a*
`ADD B, [C+0]`



Pila: POP B



Ejercicio: Para traducir a código máquina el acceso a los elementos de un registro (struct) de C, ¿que direccionamiento usarías?

Referencias:

- “Computer Organization and Architecture”. William Stallings, 2015. Pearson.
- Documentación del Simlador. Pablo Parra. https://github.com/parraman/asm-simulator/tree/master/docs
- Wikipedia.