ARQUITECTURA DE COMPUTADORAS

Unidad N° 2

- Paridad
- Direccionamiento
- Introducción al Assembler

Profesor: Fabio Bruschetti

Ver 2013-01

UP Universidad de Palermo

Códigos de paridad

- Una combinación binaria puede contener una cantidad para o impar de unos (por ende de ceros también)
- Paridad PAR = cantidad par de "unos"
- Paridad IMPAR = cantidad impar de "unos"
- Cualquier combinación binaria puede convertirse en una combinación de paridad par o impar, agregando un bit tal que se cumpla con las condiciones de paridad
- Ejemplo:
  - `0001010100 b` → Es de paridad IMPAR
  - `000101010011b` → Es de paridad PAR
- Se los utiliza para detectar errores de transmisión de datos (una cantidad par o impar de errores)
  - Si transmito una combinación de paridad PAR y recibo una de paridad IMPAR, puedo asegurar que hay **error**
  - Si transmito PAR y recibo PAR, no puedo asegurar nada, pero… la tasa de error en más de 1 bit es muy baja!!

Decimal N°	Códigos de paridad Código Z	Código W
0	0 0 0 0 0	0 0 0 0 1
1	0 0 0 1 1	0 0 0 1 0
2	0 0 1 0 1	0 0 1 0 0
3	0 0 1 1 0	0 0 1 1 1
4	0 1 0 0 1	0 1 0 0 0
5	0 1 0 1 0	0 1 0 1 1
6	0 1 1 0 0	0 1 1 0 1
7	0 1 1 1 1	0 1 1 1 0
8	1 0 0 0 1	1 0 0 0 0
9	1 0 0 1 0	1 0 0 1 1

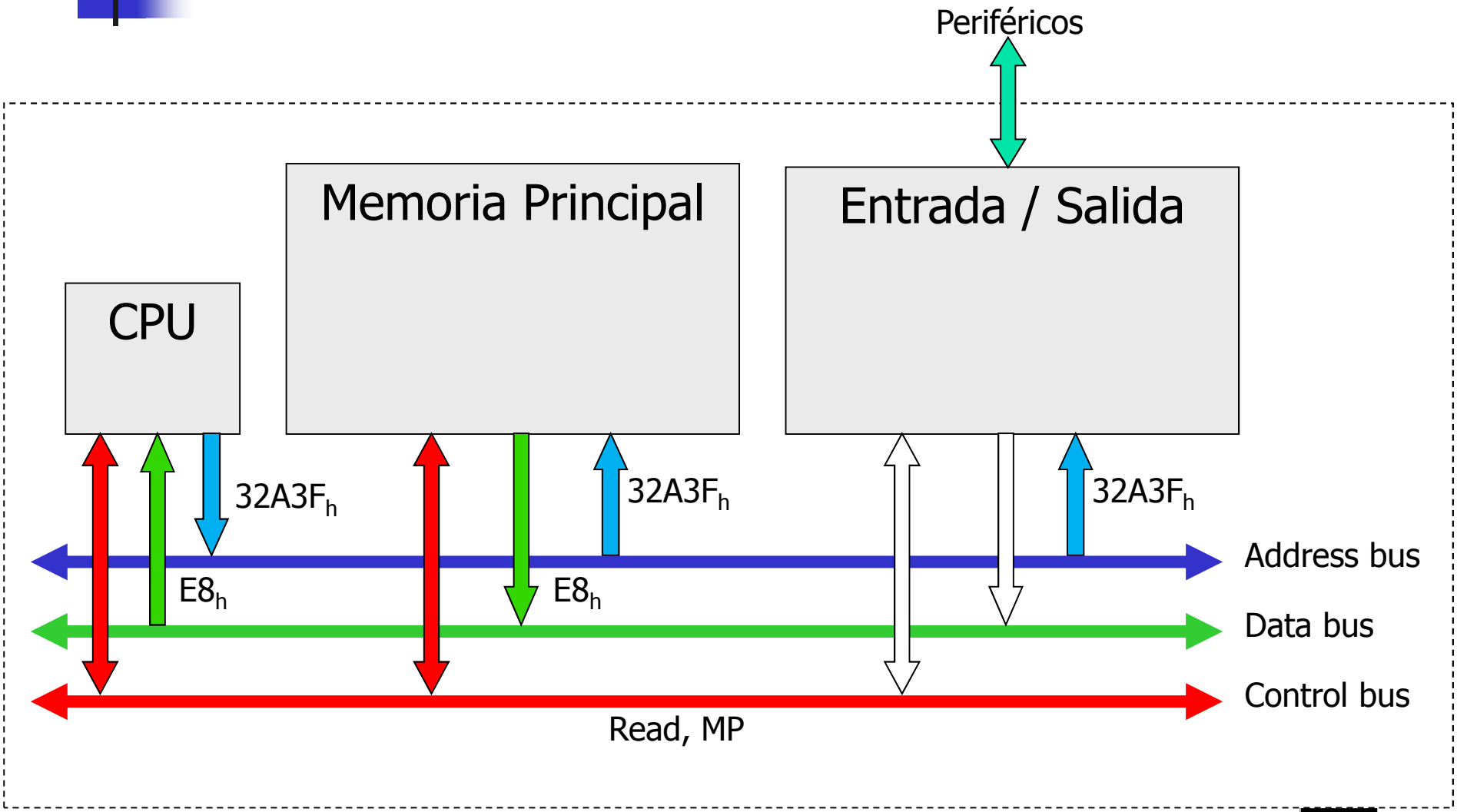
UP Universidad de Palermo

Direccionamiento

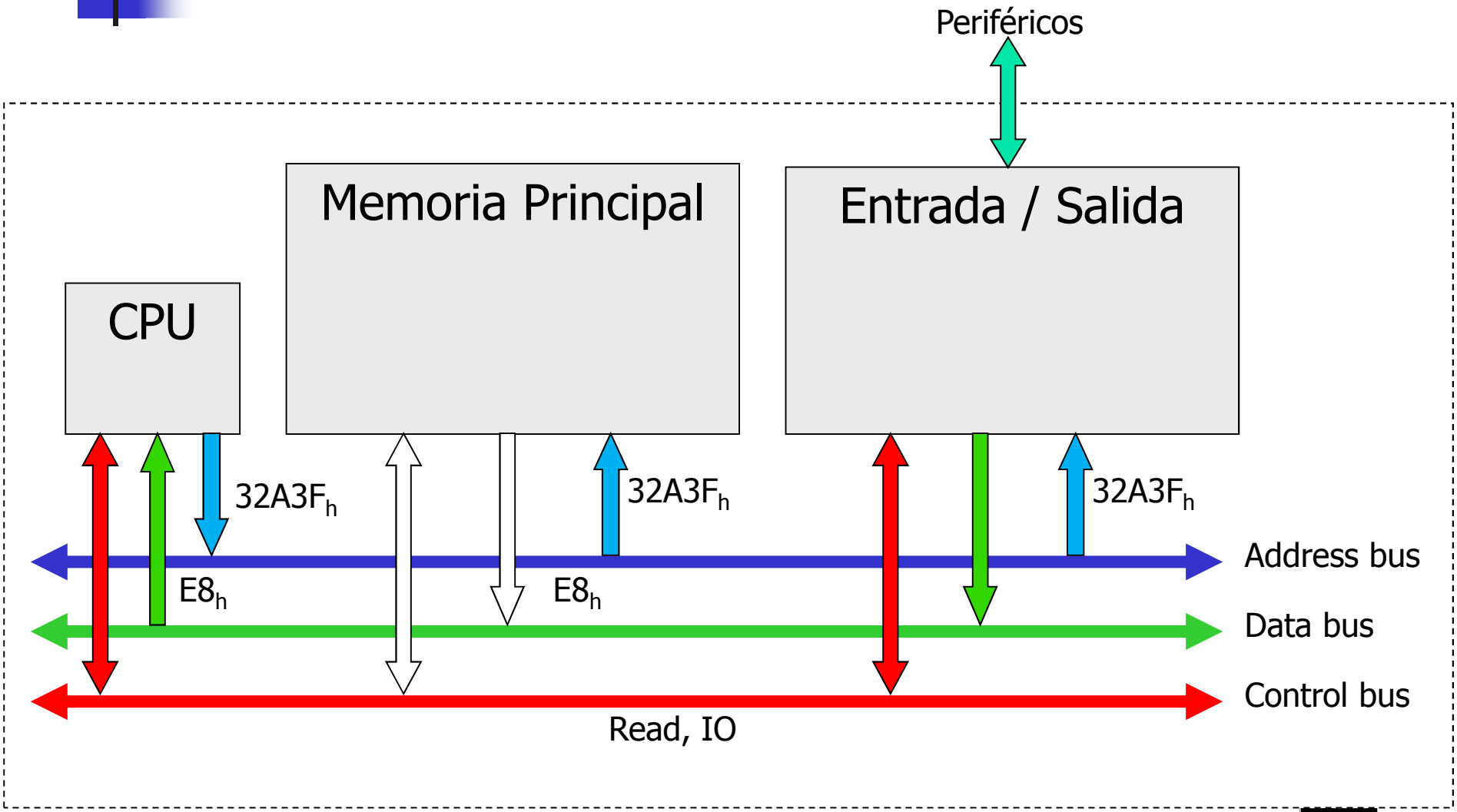
- Forma que tiene la CPU para apuntar a una dirección
- Esa dirección es un número binario
- La dirección apuntada se presenta en el bus de Direcciones (Address Bus)
- Los dispositivos (Memoria Pincipal, Periféricos) están "escuchando" (sensando) el bus de direcciones
- Cuando en el bus haya una dirección, será unívoca. A través del bus de control (Control Bus) se indicará si se refiere a una dirección de Memoria Principal o Periféricos
- En el bus de control se indicará si se requiere
  - leer (Read): el dato leído será provisto en el bus de datos (Data Bus) por el dispositivo direccionado
  - escribir (Write): el dato debe ser provisto en el bus de datos por el que direcciona al dispositivo
- La señal de CLK será la encargada de disparar la operación deseada

UP Universidad de Palermo

Direccionamiento – Lectura MP



Direccionamiento – Lectura E/S

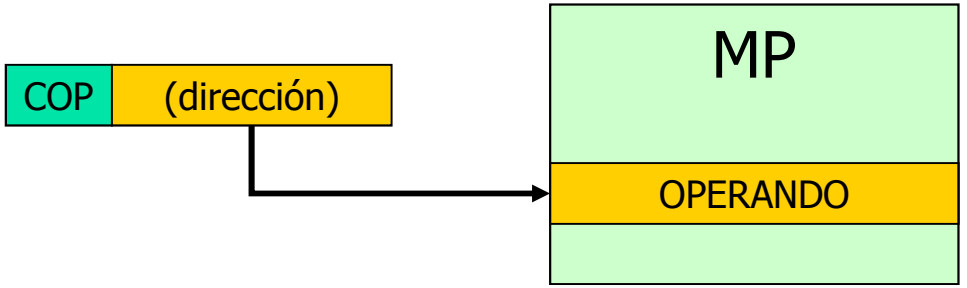


Modos de direccionamiento

- Implícito
  - No hay un operando explícito en la operación. El Código de Operación lo contiene
  - CLC (Clear Carry Flag)

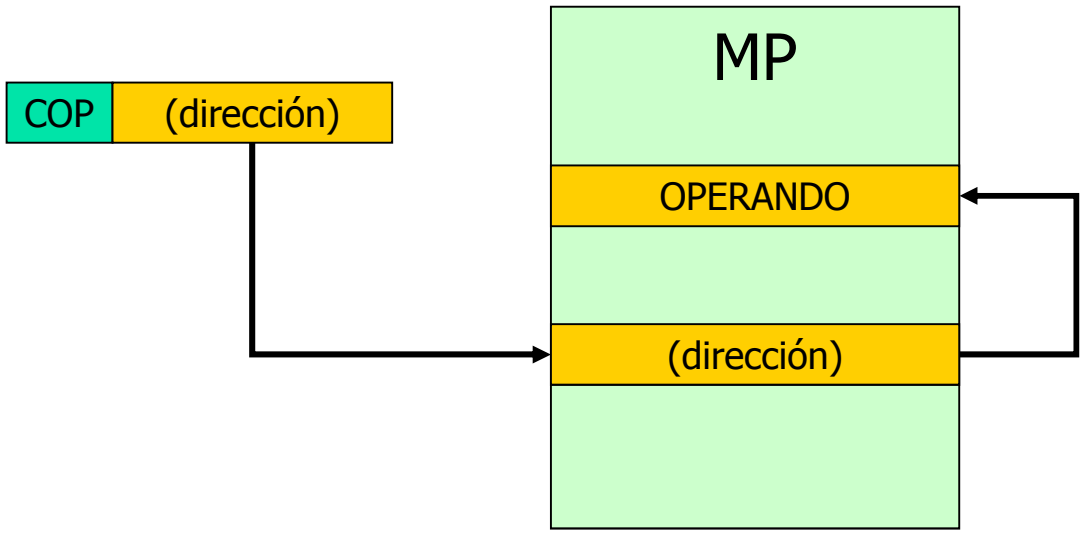
- Inmediato
  - El operando está presente en la propia instrucción
  - ADD AX, 5_h

- Directo
  - El campo de direcciones contiene la dirección efectiva del operando
  - ADD AX, [100_h]



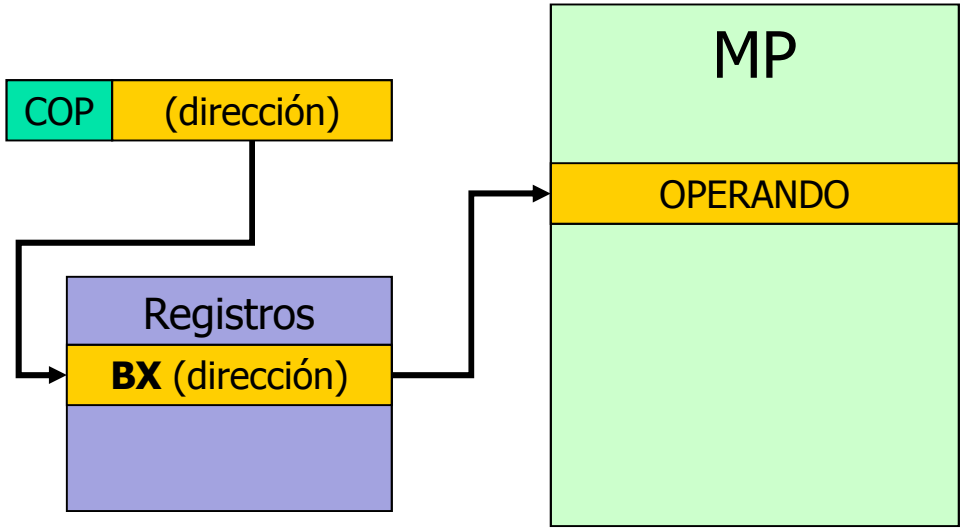
Modos de direccionamiento

- Indirecto
  - El campo de direcciones contiene la dirección que contiene la dirección efectiva del operando



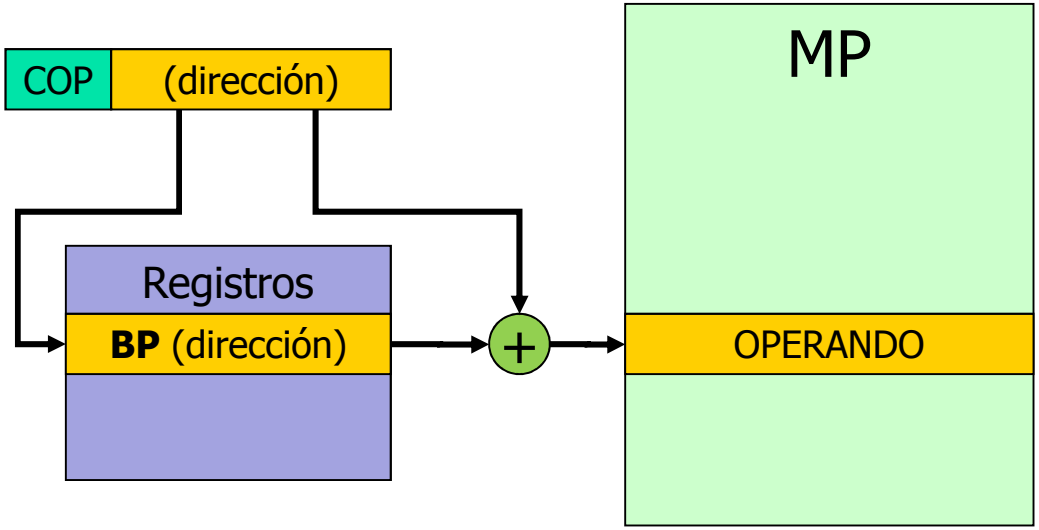
Modos de direccionamiento

- Indirecto con Registro
  - El campo de direcciones contiene el registro que contiene la dirección efectiva del operando
  - ADD AX, [BX]



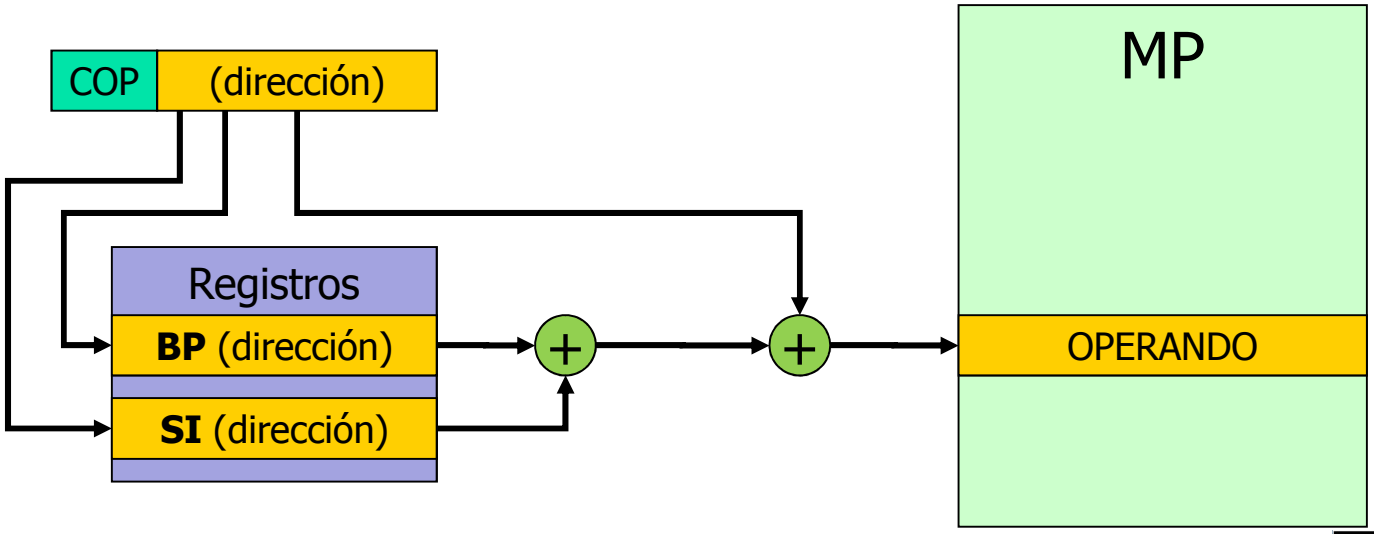
Modos de direccionamiento

- Con Desplazamiento (Relativo a la Base o Basado)
  - El campo de direcciones contiene el registro cuyo dato sumado a una constante determina la dirección efectiva del operando
  - ADD AX, [BP + 100_h]
  - BP = Base Pointer



Modos de direccionamiento

- Con Desplazamiento (Basado e Indexado)
  - El campo de direcciones contiene el registro cuyo dato sumado al contenido de otro registro (índice) más una constante determina la dirección efectiva del operando
  - ADD AX, [BP + SI + 100_h]
  - BP = Base Pointer
  - SI = Segment Index



Programación en Assembler

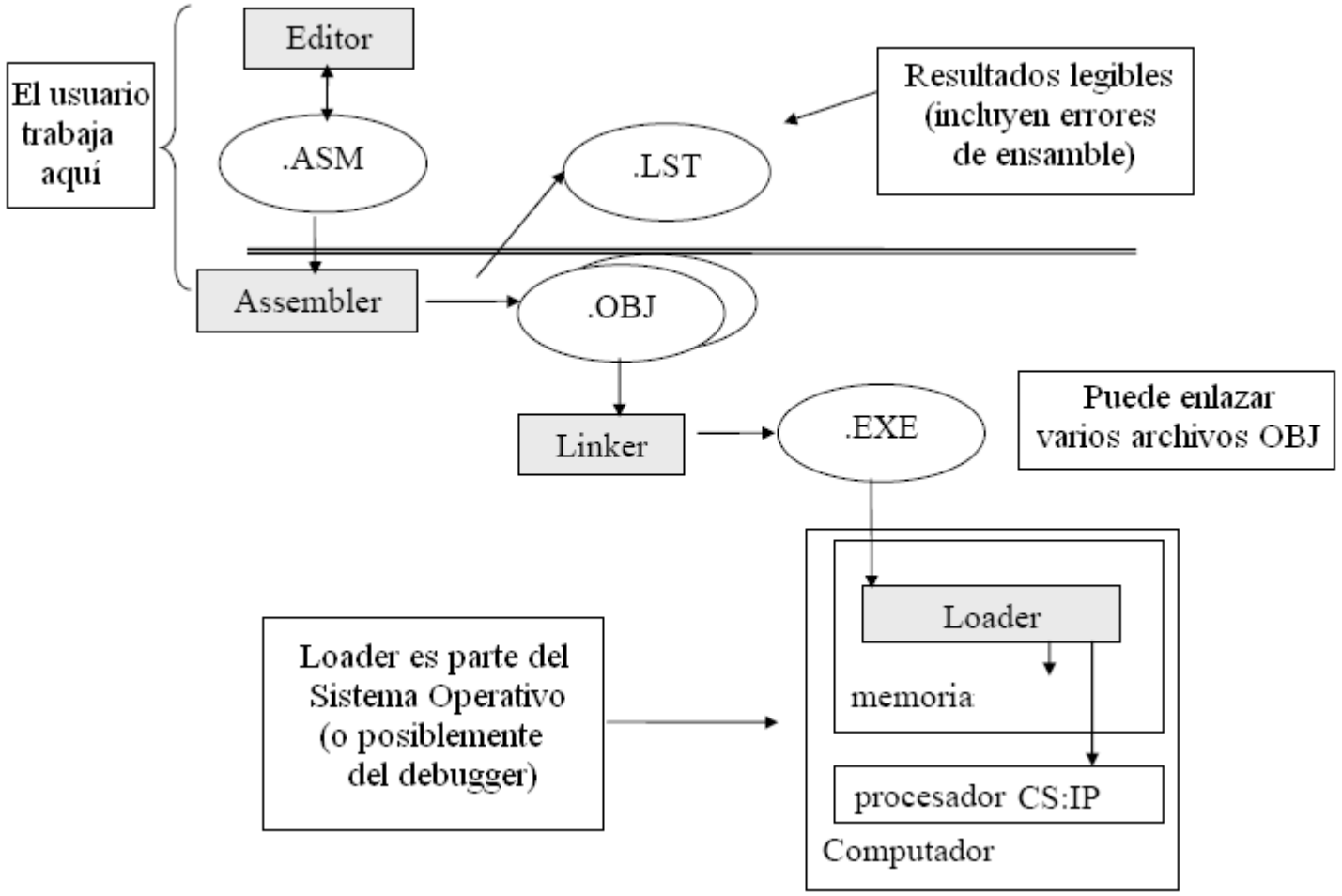
- Problema: Se debe convertir ideas (pensamiento humano) en un programa ejecutable (imagen binaria en memoria)
- El proceso de Desarrollo de Programas usa un conjunto de herramientas para escribir programas y luego convertirlos en formato binario
- Lenguaje de Programación:
  - **Sintaxis**: Conjunto de símbolos + reglas de gramática para construir sentencias usando símbolos.
  - **Semántica**: Cuál es el significado de las sentencias, o cuál es el resultado de la ejecución
  - **Lenguaje Assembler**: Un lenguaje legible que mapea uno a uno con las instrucciones de máquina (con las operaciones que son soportadas por la CPU)

Programación en Assembler

- Assembler o Ensamblador:
  - Se trata de un programa que convierte el lenguaje assembler al formato Objeto.
    - El código objeto es el programa ejecutable en formato de máquina (formato binario)
    - El código objeto puede contener Referencias No Resueltas.
- Linker o Enlazador:
  - Se trata de un programa que conviena archivos en formato objeto y genera un solo archivo ejecutable
    - Con todas las referencias resueltas
- Loader o Cargador:
  - Se trata de un programa que carga el archivo ejecutable en memoria y puede inicializar algunos registros (Ej. IP) e inicia la ejecución.
- Debugger o Depurador:
  - Similar al Loader pero controla la ejecución del programa con el objetivo de ver y modificar el estado de las variables durante la ejecución.

UP Universidad de Palermo

Programación en Assembler



Programación en Assembler

- Código Fuente
  - Es un programa escrito en lenguaje assembler o de mayor nivel que el binario y almacenado en un archivo "fuente" (.ASM)
- Código Objeto
  - Es la salida de un ensamblador o compilador
  - Es el programa ejecutable en formato binario (instrucciones de máquina) (.OBJ)
  - Referencias externas no resueltas (Linker resuelve estas referencias y genera el archivo ejecutable)
- Código Ejecutable
  - El programa ejecutable completo en formato binario con la estructura impuesta por el sistema operativo (.EXE o .COM)

UP Universidad de Palermo

Assembler – Constantes

- Valores Binarios : consisten solamente de 0's y 1's
  - Terminan con 'B'o 'b'
  - ej. 10101110b
- Valores Hexadecimales: comienzan con 0 .. 9
  - Puede incluir 0 . . 9, A .. F (a . . f)
  - Terminan con 'H' o 'h'
  - Requieren un cero antes de poner A..F como primer dígito
  - ej. 0FFH (valor hex de 8-bit)
- Valores Decimales:
  - Formato por default – no necesitan "calificador"
  - consisten de dígitos entre 0 . . 9
- String: secuencia de caracters codificados como bytes ASCII:
  - Se encierran entre comillas simples ` '
  - ej. 'Hola Ma' – 7 bytes
  - caracter: string de longitud = 1
  - D.O.S. Strings DEBEN TERMINAR SIEMPRE con '$'

UP Universidad de Palermo

Programa “Hello world!”

C:\...\DEBUG <Enter>
- e 110 "Hello, world!__$" <Enter>
- e 11C ... 0dh <Espacio> 0ah <Enter>

- a 100

CS:0100 mov ah,9	***"DOS Function call to print a message”***
CS:0102 mov dx, 110	***"DX=Pointer to message ”***
CS:0105 int 21h	***"DOS General Purpose Interrupt”***
CS:0107 mov ax,4C00h	***“DOS Function call to exit back to DOS”***
CS:010A int 21h	***"DOS General Purpose Interrupt”***

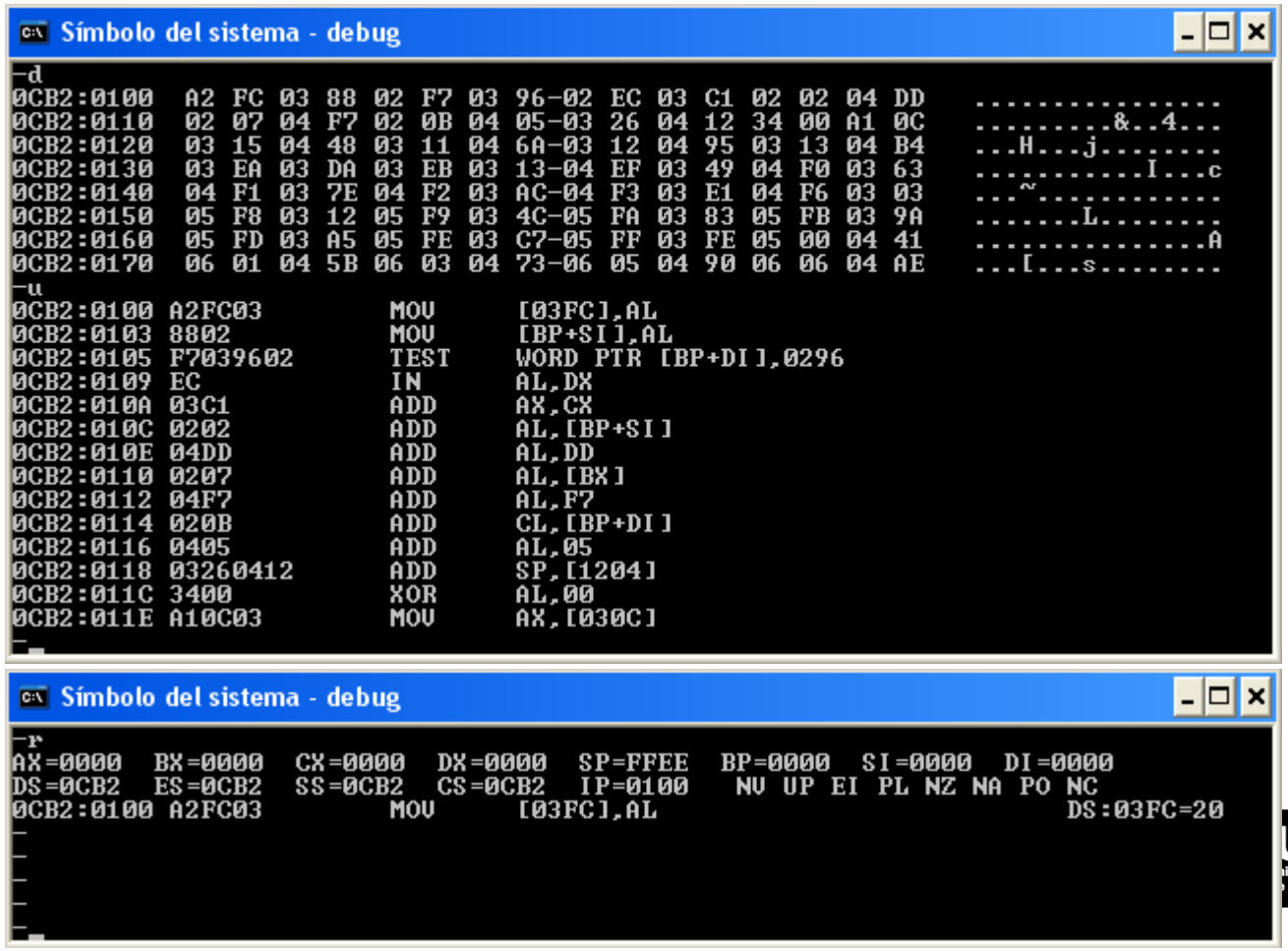
- u 100 10B <Enter> *(Para verificar el programa cargado)*
- d 110 <Enter> *(Para verificar los datos cargados)*

- n HOLA.COM <Enter> *(Nombre del programa)*
- RCX <Enter>
- 20 <Enter>
- w <Enter>

UP Universidad de Palermo

Depuración de Código

- Una forma de depuración de código se puede llevar a cabo con el comando DEBUG.EXE



Depuración de Código

- Debug
  - d muestra memoria de datos
  - a permite ingresar instrucciones assembler
  - u muestra las instrucciones en memoria
  - n nombre del archivo
  - w escribe en el archivo la cantidad de bytes
  - p procesa paso a paso las instrucciones
  - e escribe posiciones de memoria
  - t ejecución instrucción por instrucción
  - g ejecuta el programa completo (hasta encontrar INT 20)
  - r muestra el contenido de los registros
    - rip permite cambiar el contenido del registro ip
    - rcx permite cambiar el contenido del registro cx
    - rf permite cambiar el contenido de los flags
  - Como almacenar un programa
    - -n c:\temp\prg.bin
    - -rcx
      - CX 0000
      - :44
    - -w

8086/8 – Características Generales

- Procesador de 16 bits
- Bus de direcciones de 20 bits : 1 Mbyte
- Bus de datos interno de 16 bits
- Bus de datos externo de
  - 16 bits en el 8086
  - 8 bits en el 8088
- Original del IBM PC/XT
- 89 instrucciones
- No tiene coprocesador matemático

8086/8 – Tipos de Datos

- ASCII
- BCD
- Enteros sin signo
  - 8 bits 0..255
  - 16 bits 0..65535
- Enteros con signo
  - 8 bits -128..127
  - 16 bits -32768..32767
- Cadenas secuencia de bytes o palabras

8086/8 – Grupos de Instrucciones

- Transferencia de datos (14)
  - Movimiento de datos entre registros y/o memoria
- Aritméticas (20)
  - Operaciones aritméticas de enteros
- Manipulación de bits (10)
  - Operaciones lógicas
- Cadenas (5)
  - Movimiento, búsqueda y comparación de cadenas de datos
- Transferencia de programa (29)
  - Saltos, llamadas...
- Control del procesador (11)
  - Detención, depuración, etc.

UP Universidad de Palermo

8086 - Conjunto de Registros

- Registros de Propósito general de 16 bits
  - Se puede acceder a los 16 bits de una vez
  - Se puede acceder al byte (H) alto y al byte (L) bajo
  - AX (Acumulador)
  - BX (Base)
  - CX (Count)
  - DX (Data)

AH	AL
BH	BL
CH	CL
DH	DL

- Registros de Direccionamiento de Segmentos de 16 Bits
  - CS Code Segment
  - DS Data Segment
  - SS Stack Segment
  - ES Extra Segment
- Registros Punteros en Segmentos de 16 Bits
  - SP Stack Pointer
  - BP Base Pointer
- Registros Indices de 16 Bits
  - SI Source Index
  - DI Destination Index

UP Universidad de Palermo

8086 - Conjunto de Registros

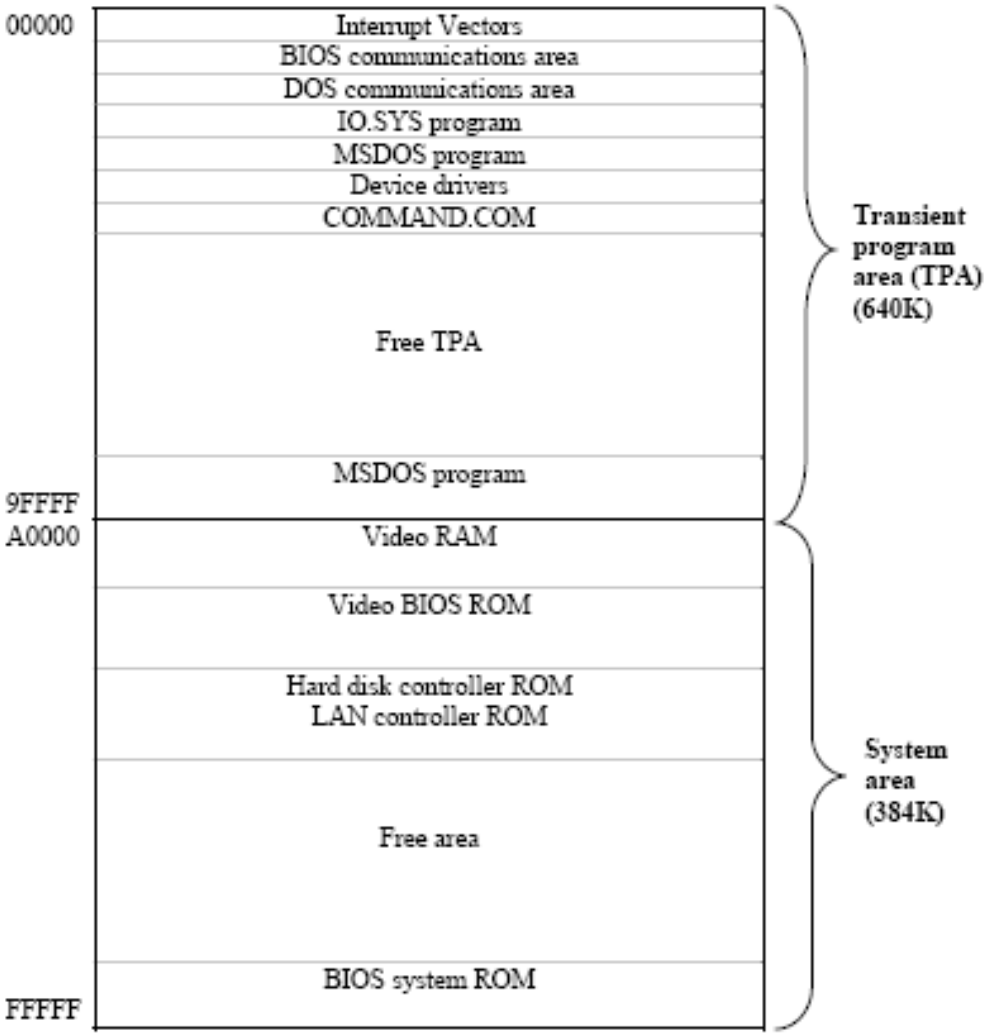
- Registros de Control y Estado de 16 bits
  - IP Instruction Pointer
  - FLAGS Registro de 16 bits
    - No se trata de un valor de 16 bits sino un colección de flags de 9 bits (seis son usados)
    - Un flag está "**Set**" cuando es igual a 1
    - Un flag está "**Clear**" cuando es igual a 0
    - Flags de Control
      - **Dirección** Usado en instrucciones sobre STRINGs para moverse hacia delante o atrás sobre el string
      - **Interrupt** Usado para habilitar o deshabilitar las interrupciones (mas adelante)
      - **Trap** Usado para habilitar o deshabilitar el trap de paso a paso

UP Universidad de Palermo

8086 - Conjunto de Registros

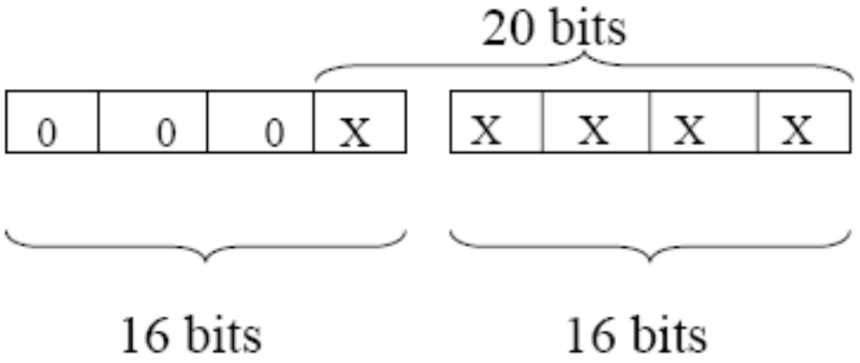
- Flags de Estado
  - Los flags son seteados o limpiados según efectos colaterales de una instrucción ejecutada
  - Parte del aprendizaje de una instrucción es conocer que flags modifica
  - Hay instrucciones que leen un flag e indican si está o no "seteado"
    - C = Acarreo: en la suma y arrastre en la resta
    - P = Paridad: (0, impar y 1, par)
    - A = Acarreo auxiliar: Indica el acarreo o arrastre entre los bits 3 y 4
    - Z = Cero: indicación de resultado en AX igual a cero
    - S = Signo: Indicación del signo del resultado. 0=positivo, 1=negativo
    - T = Trampa: Habilita la característica de depuración del procesador
    - I = Interrupción: Habilitación de interrupciones de hardware
    - D = Dirección: Selección de incremento o decremento en los índices
    - O = sobreflujo

8086 – Mapa D.O.S. de Memoria 1Mb

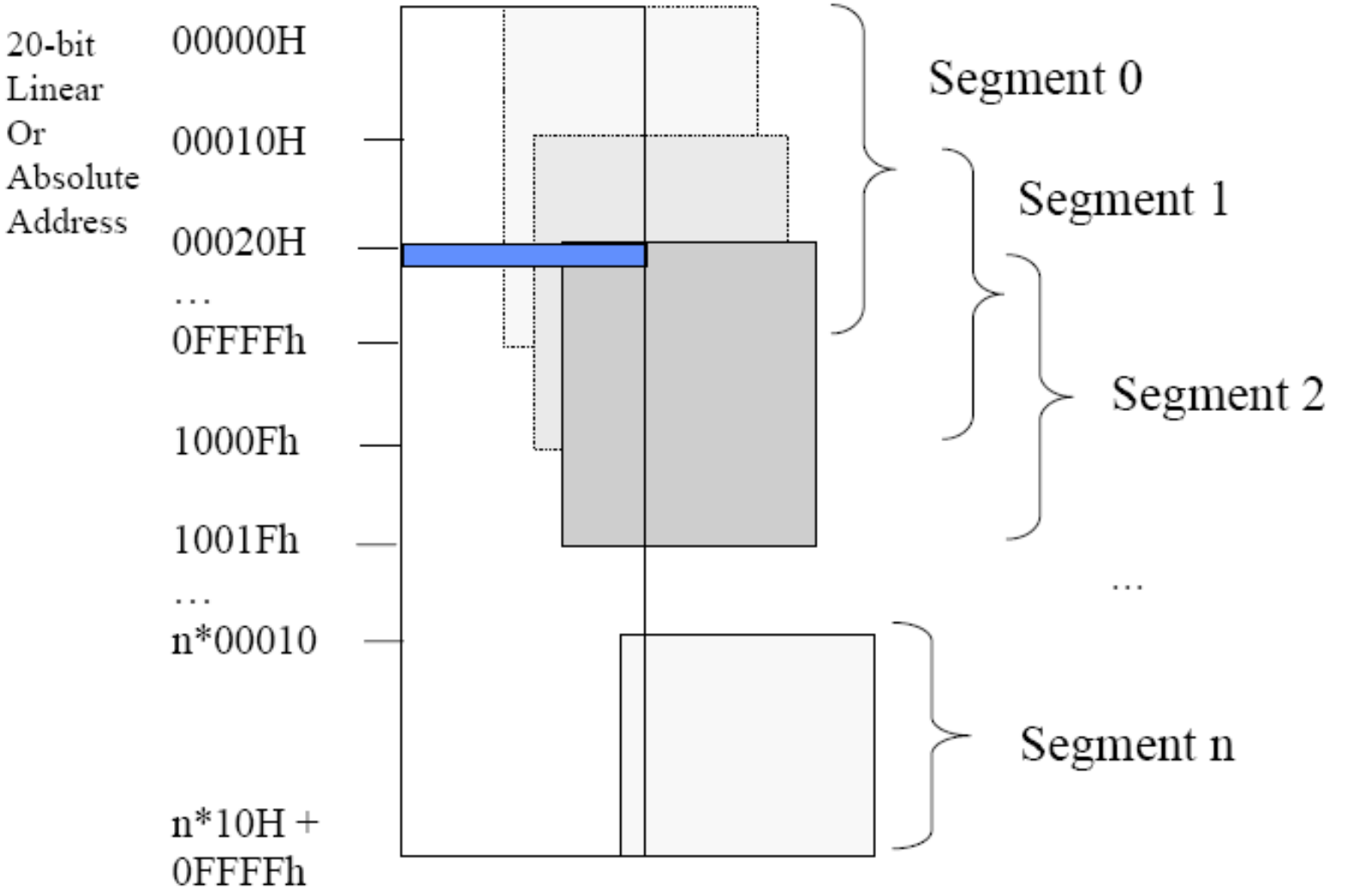


8086 – Memoria Segmentada

- Modelo de Memoria Segmentada para 8086 con 20 bits de espacio de Direccionamiento
  - Problema de Diseño de Procesador
    - ¿Como usar registros y valores de 16 bits para especificar direcciones de 20 bits?
    - Una forma: Usar dos registro "uno al lado del otro"

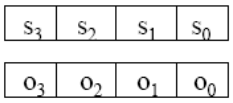


8086 – Memoria Segmentada



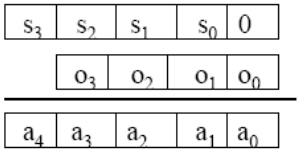
8086 – Memoria Segmentada

- A Nivel de Hardware:
  - Una dirección se coloca en el bus de direcciones de 20 bits como una dirección absoluta
- A Nivel de Programador:
  - Las direcciones NUNCA se especifican como valores de 20 bits
  - Las direcciones SIEMPRE se especifican como dos valores de 16 bits:
    - Segmento:Desplazamiento Segment:offset
- Quien hace la conversión?
  - La CPU, durante el fetch de una instrucción
    - Recordar que cada segmento comienza cada 16 bytes
    - La dirección de un segmento = Número de Segmento * 16_{10}
    - Segmento:
    - Offset:



Determinado por el nro de segmento

    - Segmento * 10h
    - Offset



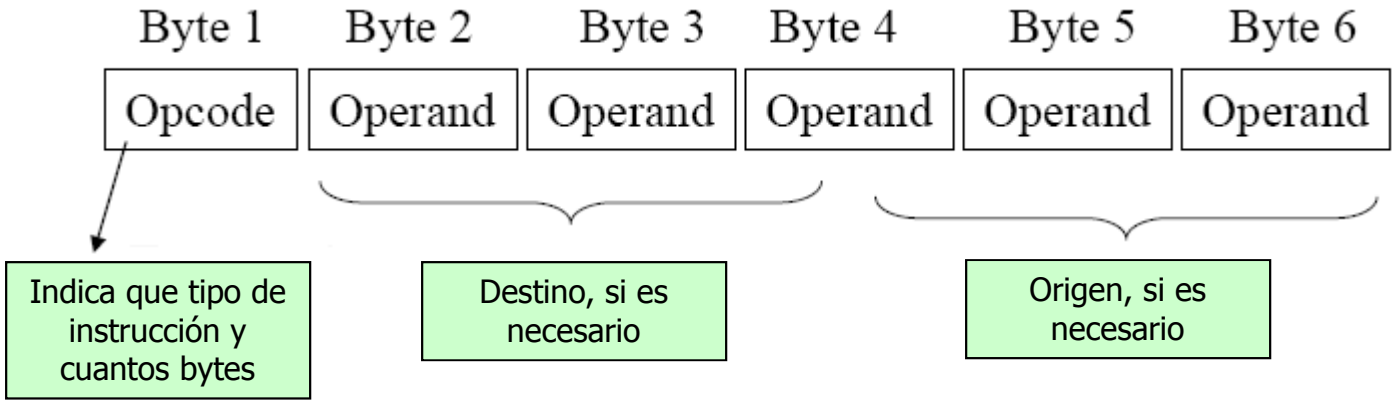
Dirección de 20 bits (Dirección abs[illegible])

8086 – Memoria Segmentada

- Ejemplo:
  - Suponamos tener el número de segmento = 6020_h y el offset 4267_h
  - Segmento * 10h 60200_h
  - Offset 4267_h
  - Dirección de 20 bits 64467_h
- Algunos de estos registros son usados por default:
  - Todos los fetchs de instrucciones: CS:IP
  - Los accesos a datos: DS:OFFSET
- Hay que tener en cuenta que los segmentos hay que inicializarlos antes de ser usados.

8086 – Codificación de Instrucciones

- En el 8086 las instrucciones es una secuencia de bytes de 1 a 6 bytes.



- # de bytes depende de # operandos
  - Nop 10010000
  - INC BX 01000001
  - ADD BX, 1 10000011 11000011 00000001 00000000

8086 – Instrucciones

- Las instrucciones de assembler son formas legibles de las instrucciones de máquina
- Se usan mnemónicos para representarlos:
  - MOV SUB ADD JMP
- Las instrucciones están compuestas por:
  - Operación Como se usan los valores de las variables
  - Operandos Que variables de estado se usarán
- Los operandos se pueden especificar de varias formas:
  - Modos simples: registro, inmediato, directo
  - Mas poderoso: Indirecto

8086 – Instrucciones

- Transferencia de Datos
  - IN = carga el acumulador desde un dispositivo de I/O
  - LAHF = carga los flags en AH
  - LEA = carga una dirección efectiva
  - LDS = carga DS y un registro de 16 bits con datos de memoria de 32 bits
  - LES = carga ES y un registro de 16 bits con datos de memoria de 32 bits
  - MOV = carga byte o palabra o doble palabra
  - OUT = saca datos del acumulador a un puerto de I/O
  - POP = recupera una palabra de la pila
  - POPF = recupera los flags de la pila
  - PUSH = almacena una pálabra en la pila
  - PUSHF = almacena los flags en la pila
  - SAHF = carga AH en los flags
  - XCHG = intercambia bytes o palabras
  - XLAT = emplea AL para entrar a una tabla de conversión

8086 – Instrucciones

- Aritméticas
  - AAA, AAD, AAM, AAS = ajuste ASCII para suma, división, producto y resta
  - ADD = suma datos entre registros o la memoria y otro registro
  - ADC = suma con acarreo
  - CBW = convierte byte a palabra
  - CMP = compara los datos
  - CWD = convierte palabra a doble palabra
  - DAA, DAS = ajuste decimal en AL para una suma/resta en BCD
  - DEC = decrementa operando en 1
  - DIV = división sin signo
  - IDIV = división con signo
  - IMUL = multiplicación con signo
  - INC = incrementa operando en 1
  - MUL = multiplicación sin signo
  - NEG = cambia el signo
  - SBB = resta con acarreo
  - SUB = resta datos entre los registros y la memoria u otro reg.

8086 – Instrucciones

- Manipulación de bits
  - AND = Y lógica
  - NOT = invertir (complemento a 1)
  - OR = O lógica
  - SAR = desplazamiento aritmético a derecha
  - SHL/SAL = desplazamiento a izquierda
  - SHR = desplazamiento lógico a derecha
  - RCL = rotación a la izquierda con acarreo
  - ROR = rotación a izquierda
  - RCR = rotación a derecha con acarreo
  - ROR = rotación a derecha
  - TEST = operación con el AND lógico pero sólo afecta banderas
  - XOR = O exclusivo

8086 – Instrucciones

- Transferencia de programa
  - CALL = llamada a subrutina
  - INT = interrupción de software
  - INT 3 = interrupción 3
  - INTO = interrupción si hay overflow
  - IRET = retorno de una rutina de interrupción
  - JA, JAE, JB, JBE = saltar si mayor, mayor o igual, menor, menor o igual
  - JE/JZ = saltar si es cero o igual
  - JG, JGE, JL, JLE = saltar si mayor, mayor o igual, menor, menor o igual
  - JMP = salto incondicional
  - JNE/JNZ = saltar si no es igual o no es cero
  - JNC, JNO, JNP, JNS = saltar si no acarreo, overflow, paridad, signo
  - JC, JO, JP, JS = saltar si acarreo, overflow, paridad, signo
  - LOOP = repite un ciclo CX veces
  - LOOPE, LOOPNE = igual a la anterior pero termina prematuramente por Z=1, 0
  - JCXZ = saltar si CX es 0
  - RET = retorno de subrutina

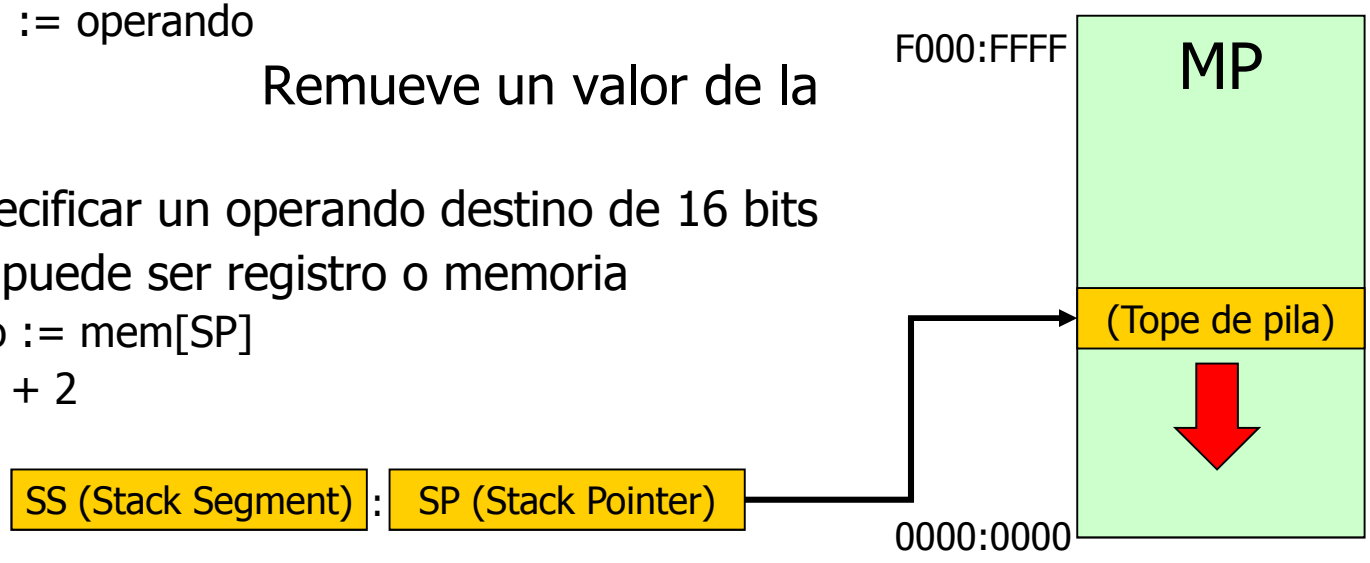
UP Universidad de Palermo

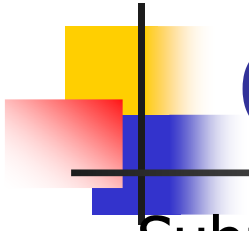
8086 – Instrucciones

- Control del Procesador
  - CLC = borrar acarreo
  - CLD = habilitar incremento automático
  - CLI = deshabilitar terminal INTR
  - CMC = complementar acarreo
  - HLT = alto hasta que se reinicialice o exista interrupción
  - NOP = no operación
  - STC = activa acarreo
  - STD = habilitar decremento automático
  - STI = habilitar interrupciones
  - WAIT = espera a que el terminal TEST=0
  - LOCK = controla el terminal LOCK

8086 – Pila (Stack)

- Stack de Intel
  - PUSH operando Agrega un nuevo valor a la pila
    - Se debe especificar un operando origen de 16 bits
    - El operando puede ser registro o memoria
    - El stack crece para abajo (hacia las dir más bajas)
      - SP := SP – 2
      - mem[SP] := operando
  - POP operando Remueve un valor de la pila
    - Se debe especificar un operando destino de 16 bits
    - El operando puede ser registro o memoria
      - operando := mem[SP]
      - SP := SP + 2





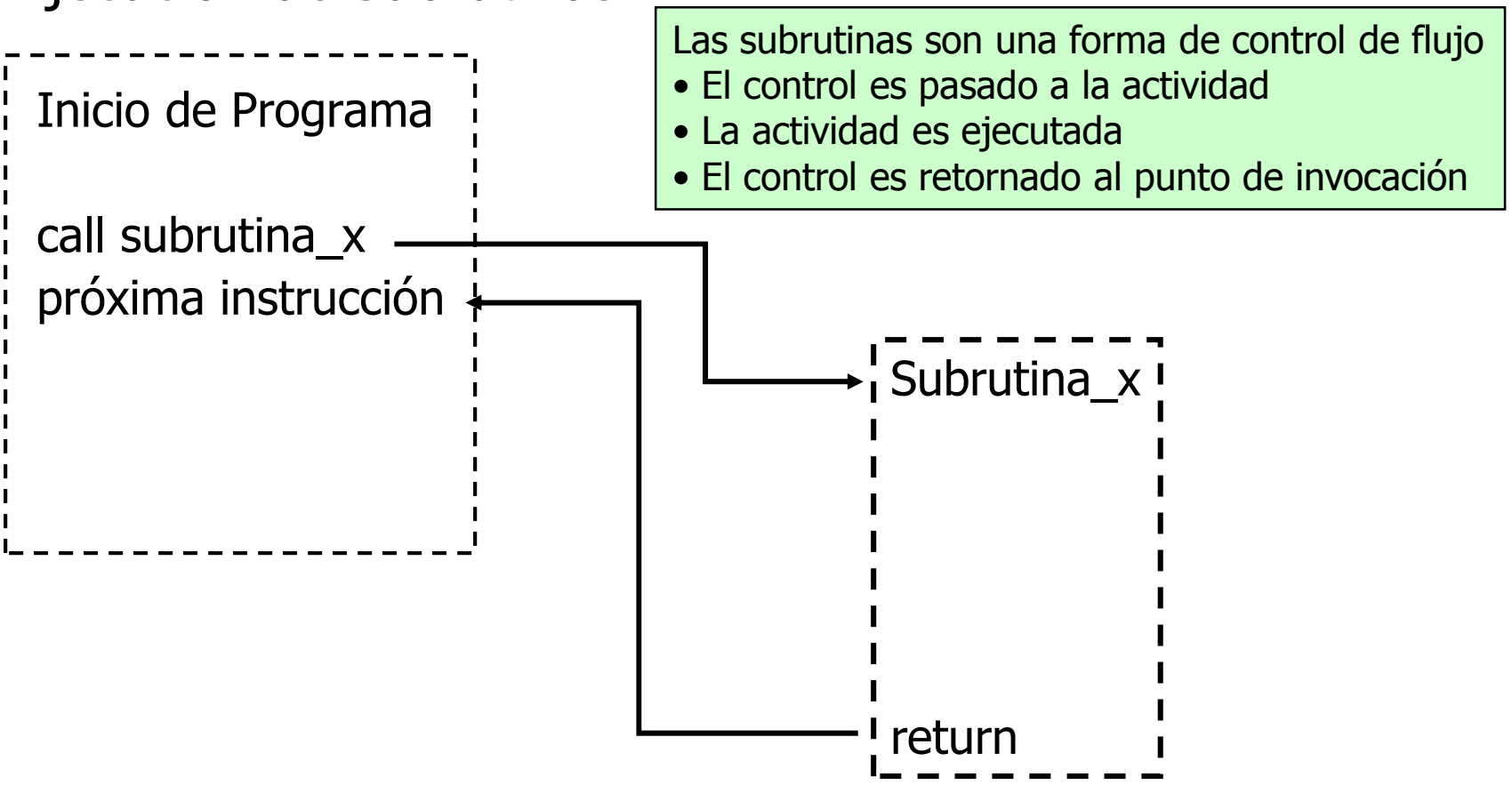
Código - Subrutinas

- Subrutinas
  - Es una secuencia de instrucciones que pueden ser llamados desde varios lugares en el programa
  - Permite realizar la misma operación pero con distintos parámetros
  - Simplifica el diseño de un programa complejo usando la aproximación divide y conquista
  - Simplifica el testeo y mantenimiento
  - Ocultamiento de información en la representación interna de variable auxiliares para resolver la operación implementada
  - En lenguajes de alto nivel se llaman FUNCIONES, PROCEDIMIENTO o METODOS
  - En lenguaje assembler se llaman SUBRUTINAS

UP Universidad de Palermo

Código - Subrutinas

- Ejecución de Subrutinas



Durante la invocación se debe salvar el punto de invocación
Durante el retorno, el punto de invocación debe restablecerse

Código - Subrutinas

- Implementación de Subrutinas a nivel de máquina
  - CALL destino ; Invoca la rutina destino
    - Semántica de la Ejecución
      - Salvar la dirección de retorno en el STACK de ejecución (runtime)
        - PUSH IP
      - Transferir el control a destino
        - JMP destino

Valor de IP posterior al fetch del CALL

  - RET ; Retorna de la subrutina
    - Semántica de la Ejecución
      - Retorna el control a la dirección salvada en el tope del stack
        - POP IP

Código - Subrutinas

- Manejo a través de Interrupciones
  - INT número ; Invoca la rutina destino
    - Se genera un stack frame distinto al de las rutinas CALL
    - Es el hardware quien escribe los siguientes valores en el stack frame
      - PUSH registro de FLAGS
      - bit IF = 0 en el registro FLAGS
      - bit TF = 0 en el registro FLAGS
      - PUSH CS
      - PUSH IP
      - CS := [0:n*4+2]
      - IP := [0:n*4]
  - IRET ; Retorna de la Interrupción
    - Saca la dirección de retorno de 32 bits (CS:IP)
    - Saca los flags