# Universidad Autónoma de Baja California
# Ingeniero en computación


## Organización y Arquitectura De Computadoras

**Practica 5**

**Nombre Del Alumno:** López Mercado Brayan

**Matrícula:** 1280838

**Grupo:** 551

**Docente:** José Isabel García Rocha

**Fecha de entrega:** 5 de octubre del 2023

## Objetivo

Identificar los modos de direccionamiento adecuados para manejo de memoria en aplicaciones de sistemas basados en microprocesador mediante la distinción de su funcionamiento, de forma lógica y responsable.

## Desarrollo

1- **Responda los siguientes cuestionamientos sobre los programas en lenguaje ensamblador de 80386 usando NASM.**

   a) ***¿Qué es la sección .data?***
   En esta sección es donde se declaran las constantes del programa, es decir, aquellos datos que no van a cambiar durante la ejecución.

   b) ***¿Qué es la sección .bss?***
   Es la sección donde se declaran todas las variables del programa, es decir, aquellos que se van a modificar durante la ejecución del programa.

   c) ***¿Qué es la sección .text?***
   Sección donde se encuentra el código del programa y sus subrutinas.

   d) ***¿Qué es la directiva global?***
   Es la que se encarga de indicar en donde comienza la ejecución del programa como tal, un detalle a tener en cuenta con la directiva global es que no deben existir declaraciones de subrutinas dentro de ella, en caso de existir cuerpos de subrutinas en la directiva, se puede provocar una segmentación.

2- **Realice las siguientes implementaciones con el objetivo de probar como funciona NASM:**

   a) ***Implemente un ejemplo de modo de direccionamiento de registro.***
   ```
   mov eax,ebx
   ```

   b) ***Implemente un ejemplo de modo de direccionamiento inmediato.***
   ```
   mov ecx,0x1280838
   ```

   c) ***Implemente un ejemplo de modo de direccionamiento directo.***
   ```
   mov [804A008h],ebx
   ```

   d) ***Implemente un ejemplo de modo de direccionamiento indirecto.***
   ```
   mov eax,[ebx]
   ```

   e) ***Implemente un ejemplo de modo de direccionamiento base más índice.***
   ```
   mov edx,[bx+di]
   ```

***f) Implemente un ejemplo de modo de direccionamiento relativo a registro.***

```
mov edx,array[bx+8]
```

***g) Implemente un ejemplo de modo de direccionamiento relativo a base más índice.***

```
mov edx,array[bx+si]
```

***h) Implemente un ejemplo de modo de direccionamiento índice escalado.***

```
mov eax,[ebx+edi*4]
```

**3- Copie el código del listado 1 en un archivo llamado P5.asm. Complete el código agregando las instrucciones necesarias para hacer los siguientes movimientos de datos:**

***a) Utilizar el modo de direccionamiento indirecto para guardar en M su matrícula (use el valor como si fuera hexadecimal).***

```
;Guardar La Matricula en M [Indirecto]
mov ecx,0x1280838
mov ebx,M
mov [ebx],ecx
mov eax,[M]
call printHex
call newLine
```

**Figura 1: Instrucciones realizadas para guardar la matrícula en M y mostrar el resultado en la terminal.**

**Descripción:** Se guarda la matrícula en ECX, el registro EBX contiene la dirección de la variable M, el movimiento se realiza pasando el contenido de ECX (la matrícula) a EBX por medio del operador [] para guardar ECX en la dirección que contiene EBX, para verificar el resultado, se pasa el contenido de M a EAX y llamando a la función printHex.

***b) Utilizar el modo de direccionamiento indirecto para guardar 0xACF2359A como binario en EBX.***

```
; Guardar 0xACF2359A como binario en EBX [Indirecto]
mov eax,N
mov dword[eax],10101100111100100011010110011010b
mov ebx,[eax]
mov eax,ebx
mov esi,cad
call printHex
call newLine
```

**Figura 2: Instrucciones realizadas para guardar ACF2359Ah como binario en EBX y mostrar el resultado en la terminal.**

**Descripción:** Se utiliza EAX para guardar la dirección de N (solo se usa como intermediario) y luego se guardar el número binario en la dirección que contiene EAX de 32 bits, luego el contenido se pasa a EBX y luego se guarda en EAX para ser mostrado en la terminal con printHex.

***c) Utilizar el modo de direccionamiento de registro para pasar lo de BX a CX.***

```
;Pasar lo de BX a CX [De Registro]
mov cx,bx
movzx eax,cx
call printHex
call newLine
```

**Figura 3: Instrucciones realizadas para mover el contenido de CX a BX y ser mostrado en la terminal.**

**Descripción:** Se realizar un movimiento de CX a BX, la instrucción movzx se utiliza únicamente para mover el contenido de CX a EAX de manera que la mitad más significativa de EAX del inciso anterior se vuelva cero.

***d) Utilizar el modo de direccionamiento Indirecto para guardar en M un 0x524.***

```
; Guardar 524h en M [Indirecto]
mov edx,M
mov ecx,0x524
mov dword[edx],ecx
mov eax,[M]
call printHex
call newLine
```

**Figura 4: Instrucciones realizadas para guardar 0x524 en M y mostrarlo en la terminal.**

**Descripción:** Se utiliza EDX para guardar la dirección de M, ECX contiene el número que pasara a M, se realiza el movimiento de ECX a la dirección contenida en EDX (32 bits), para verificar el resultado se pasa el contenido de M a EAX para ser mostrado en la terminal por medio de printHex.

***e) Utilizar el modo de direccionamiento directo para copiar el contenido de EBX a la dirección obtenida de M.***

```
;Copiar El Contenido De Ebx a la dirección de M [Directo]
;lea eax,M
mov [804A008h],ebx
mov eax,[804A008h]
call printHex
call newLine
```

**Figura 5: Instrucciones realizadas para guardar el contenido de EBX en M y mostrarlo en la terminal.**

**Descripción:** Para obtener la dirección de M se utilizó la instrucción LEA (aunque un simple Mov eax,M hace lo mismo), luego de obtener la dirección se realiza un modo de direccionamiento directo para pasar el contenido de EBX a la dirección de M y luego se pasa el contenido de dicha dirección a EAX para ser mostrado en la terminal por medio de printHex. En caso de que pregunte de porque no hice un simple mov [M], ebx se debe a que el ensamblador de 16 bits no deja realizar esa instrucción e intento no acostumbrarme a hacerlo todo en una solo instrucción.

***f) Crear una variable llamada array de 256 bytes, esta debe de estar después de la memoria reservada para cad.***

```
cad resb 16
; Crear Variable Array
array resb 256
```

**Figura 6: Creación de la variable array de 256 bytes después de la variable cad.**

***g) Guardar con el modo de direccionamiento directo a array los siguientes valores hexadecimales de 32 bits cada uno de ellos: 0,1,2,3,4 en su correspondiente posición de memoria.***

```
; Guardar en array los valores 0,1,2,3,4 (32 bits) [Directo]
;lea eax,array
mov ecx,0x0
mov [0x0804A01C],ecx
mov ecx,0x1
mov [0x0804A020],ecx
mov ecx,0x2
mov [0x0804A024],ecx
mov ecx,0x3
mov [0x0804A028],ecx
mov ecx,0x4
mov [0x0804A02C],ecx
```

**Figura 7: Instrucciones realizadas para guardar 0,1,2,3,4 en array.**

**Descripción:** Al igual que el inciso e, se utilizó LEA para obtener la dirección de inicio de array (aunque al final solo aparece como comentario), luego se copia a ECX los valores a guardar en las direcciones del arreglo (se tiene un incremento manual por elemento), luego se guardan en las direcciones de los elementos de array (cada una incrementa en 0x4 respecto a la posición anterior) y así hasta guardar todos los elementos.

***h) Comprobar con el modo de direccionamiento base más índice escalado que estén guardados los valores anteriores en sus correspondientes valores de memoria. Apóyese de printHex para revisar si están haciendo correctamente las instrucciones.***

```
;Comprobar que se encuentre en el valor correcto [Base+Indice Escalado]
mov ebx,array
mov edi,0x0
mov eax,[ebx+edi*4]
call printHex
call newLine
mov edi,0x1
mov eax,[ebx+edi*4]
call printHex
call newLine
mov edi,0x2
mov eax,[ebx+edi*4]
call printHex
call newLine
mov edi,0x3
mov eax,[ebx+edi*4]
call printHex
call newLine
mov edi,0x4
mov eax,[ebx+edi*4]
call printHex
call newLine
```

**Figura 8: Instrucciones realizadas para verificar los contenidos de la variable array.**

**Descripción:** Se copia a EBX la dirección de inicio de array (la base) y se copia en EDI el elemento a mostrar (en la primera iteración es el cero), luego se copia a EAX el contenido en la dirección resultante y se muestra en la terminal, se repite el proceso anterior con todos los elementos, es decir, hasta acceder a la posición donde está el 0x4.

## *Resultados*

***Inciso a)***

```
01280838
```

***Inciso b)***

```
ACF2359A
```

***Inciso c)***

```
0000359A
```

***Inciso d)***

```
00000524
```

***Inciso e)***

```
ACF2359A
```

***Inciso h)***

```
00000000
00000001
00000002
00000003
00000004
```

***Terminal Con los resultados de todos los incisos***

```
brayan@Cake-Roll:~/OacAgain/p5$ ./P5
01280838
ACF2359A
0000359A
00000524
ACF2359A
00000000
00000001
00000002
00000003
00000004
```

***Link Al Código Completo***

*Practica 5: [https://github.com/BrayanLMercado/OAC_Practica5_v2.git](https://github.com/BrayanLMercado/OAC_Practica5_v2.git)*

***Conclusiones y Comentarios***

- El realizar los movimientos con los modos de direccionamiento especificados no resulto de gran complejidad, lo que resulta algo complicado de entender es cuando se está usando cada modo de direccionamiento, puesto que algunos de ellos se parecen.

- La ayudo a diferenciar con mayor facilidad cada uno de los modos de direccionamiento y a conocer las limitantes o dificultades en la implementación de cada uno.

***Dificultadas en el desarrollo***

- Una dificultad que fue difícil de identificar fue con el inciso g, el problema era que no guardaba los valores en array donde debía, el problema se resolvió al realizar los incremento con aritmética hexadecimal en lugar de tratar de hacerlo con aritmética decimal, en pocas palabras, está realizando los incrementos de la siguiente manera: 0,4,8,12,16, de esa manera nunca iba a obtener el resultado correcto en el inciso h.

***Referencias***