

Trabajo práctico

Monocromatización de una imagen .bmp

Organización de Computadoras 2013

UNQ

Índice

I	Introducción	2
II	Arquitectura IA-32	2
1.	Instrucciones de dos operandos	3
2.	Salto	4
3.	Ejemplos	4
III	Marco teórico	5
4.	Formato BMP	5
5.	Algoritmo de monocromatización	6
IV	Enunciado	7
V	Anexos	8

6. Instalación de Linux	8
7. Instalación de herramientas necesarias	8
8. Errores comunes	8

Parte I

Introducción

El siguiente documento presenta el trabajo práctico de la materia Organización de computadoras. El mismo tiene por objetivo introducir a los alumnos un lenguaje ensamblador utilizado en la industria para que puedan desarrollar una aplicación sencilla.

En el apartado II se presentan los aspectos necesarios del lenguaje a utilizar. En III se presenta la base teórica para realizar el trabajo. Finalmente en IV se plantea la consigna que deben resolverse

Parte II

Arquitectura IA-32

IA-32 (Intel Architecture, 32-bit), conocida de manera genérica como x86, x86-32 o i386, es la arquitectura del conjunto de instrucciones del procesador de Intel comercialmente más exitosa. Prácticamente cualquier procesador actual de 32 bits trabaja con esta arquitectura.

Para este trabajo práctico deberán utilizar el lenguaje de ensamblador de la misma. A continuación se presentan las características principales de ella que son necesarias para la resolución del trabajo:

- La arquitectura cuenta con los siguientes registros de uso general (32 bits): *EAX*, *EBX*, *ECX*, *EDX*, *ESI* y *EDI*.
- El direccionamiento es a byte.
- Las direcciones tienen 32 bits.
- Posee flags de carry, overflow, negative y zero.

Entre las instrucciones que brinda la arquitectura tenemos:

1. Instrucciones de dos operandos

Sintaxis	Efecto
MOV dst,src	$\text{dst} \leftarrow \text{src}$
CMP dst, src	Realiza dst-src y modifica los flags de acuerdo al resultado
ADD dst, src	$\text{dst} \leftarrow \text{dst} + \text{src}$
SUB dst, src	$\text{dst} \leftarrow \text{dst} - \text{src}$

El primer operando es el destino y el segundo es la fuente. Los modos de direccionamiento soportados para el destino son: registro, indirecto registro o directo. En cambio para la fuente tenemos: registro, indirecto registro, directo o inmediato. Sin embargo, cuando el operador destino es indirecto o indirecto registro, el operador fuente debe ser registro o inmediato. La sintaxis para los distintos modos de direccionamiento es similar a la utilizada en la arquitectura **QARQ**.

Ejemplos

Instrucciones válidas

- MOV EAX, 0004h
- MOV EAX, EBX
- MOV EAX, [EBX]
- MOV EAX, [0004h]
- MOV [EAX], EBX
- MOV [EAX], 0004h
- MOV [0004h], 0004h
- MOV [0004h], EBX

Instrucciones inválidas

- Inválidas porque usan dos operadores en memoria
 - MOV [EAX], [0004h]: El operador fuente no puede ser directo
 - MOV [EAX], [EBX]: El operador fuente no puede ser indirecto a registro
 - MOV [0004h], [EBX]: El operador fuente no puede ser indirecto a registro
 - MOV [0004h], [0055h]: El operador fuente no puede ser directo
- Inválidas porque usan un destino inmediato
 - MOV 0005h, EBX: El operador destino no puede ser inmediato
 - MOV 0005h, [EBX]: El operador destino no puede ser inmediato
 - MOV 0005h, [0004h]: El operador destino no puede ser inmediato
 - MOV 0005h, 0004h: El operador destino no puede ser inmediato

2. Saltos

- Salto incondicional: Soporta un operando que puede estar en modo directo, indirecto a registro, registro o inmediato. Usualmente se usa una etiqueta para indicar el salto. Las etiquetas funcionan igual que en la arquitectura **QARQ**.
 - JMP dest
- Saltos condicionales: Como operando toman un desplazamiento (offset). Usualmente se usa una etiqueta para indicar el salto.
 - JE offset: Saltar por igual
 - JZ offset: Saltar por cero
 - JNE offset: Saltar por distinto
 - JNZ offset: Saltar por distinto de cero
 - JA offset: Saltar por mayor sin signo
 - JAE offset: Saltar por mayor o igual sin signo
 - JB offset: Saltar por menor sin signo
 - JBE offset: Saltar por menor o igual sin signo
 - JC offset: Saltar por carry
 - JG offset: Saltar por mayor con signo
 - JGE offset: Saltar por mayor o igual con signo
 - JL offset: Saltar por menor con signo
 - JLE offset: Saltar por menor o igual con signo
 - JN offset: Saltar por negativo
 - JO offset: Saltar por overflow

3. Ejemplos

Restar EAX con EBX y poner 1 en ECX si la resta da 0

```
        SUB EAX, EBX
        JZ poner_uno
        RET
poner_uno: MOV ECX, 0001h
        RET
```

Sumar todas las posiciones de un arreglo de enteros de 32 bits que comienza en [EAX] y tiene tamaño ECX guardando el resultado en EBX

```
MOV EBX, 0
CMP ECX, 0
```

```

ciclo:  JE fin
        ADD EBX, [EAX]
        ADD EAX, 0004h ; Sumamos 4 porque queremos movernos 32 bits hasta el próximo
                        ; número y la arquitectura direcciona a byte: 4 bytes = 32 bits
        SUB ECX, 1
        JMP ciclo
fin:    RET

```

Parte III

Marco teórico

Para este trabajo deberán trabajar con imágenes en formato BMP, así como implementar un sencillo algoritmo de monocromatización. A continuación se brinda una introducción a los conceptos que deberán conocer para la resolución del mismo.

4. Formato BMP

El formato BMP es un modo simple de guardar imágenes en una computadora. Básicamente la imagen se representa con una tabla, donde cada celda es un pixel. En el caso del BMP de *24 bits*, que es el que utilizaremos para este trabajo, los pixeles tienen tres componentes: Rojo, verde y azul (o R, G y B por sus iniciales en inglés), y cada uno de ellos puede tomar valores entre 0 y 255, es decir *8 bits* por componente. Los valores determinan con que color se ve el pixel. En la figura 1 se observan que color toma el pixel en base a sus componentes.



Figura 1: Distintos valores de RGB con sus colores

En memoria esta tabla se almacena como un arreglo de pixeles. A fin de facilitar la implementación del trabajo consideraremos que la tabla se encuentra almacenada en tres arreglos, uno que contiene todas las componentes rojas, otro con todas las verdes y otro con las azules. Esto se explica gráficamente en la figura 2.

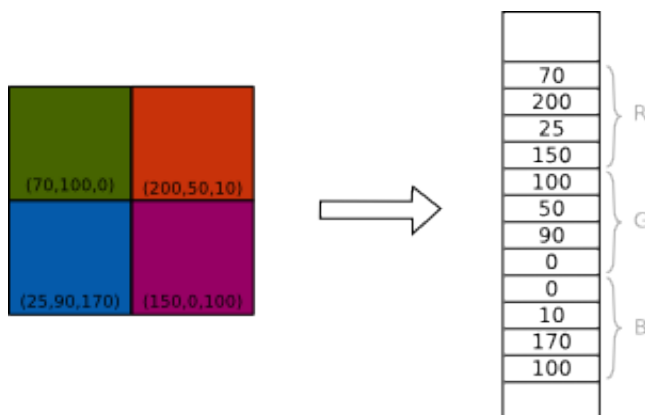


Figura 2: Cómo se organiza la tabla en la memoria

5. Algoritmo de monocromatización

Monocromatizar una imagen es pasarla a escala de grises. Los colores de esta escala se representan con píxeles que tienen todas sus componentes con el mismo valor, así por ejemplo el blanco tiene a todas en 255, el negro todas en 0, mientras que valores intermedios dan distintos grises. Para pasar una imagen a color a una en escala de grises hay que aplicar algún algoritmo que determine para cada píxel de la imagen original, en base a sus componentes, que valor entre 0 y 255 le corresponde. Ese valor es el que se le da a las componentes del píxel de salida. De esta forma como las tres tienen el mismo valor nos aseguramos que el píxel tenga un color que pertenece a la escala de grises.

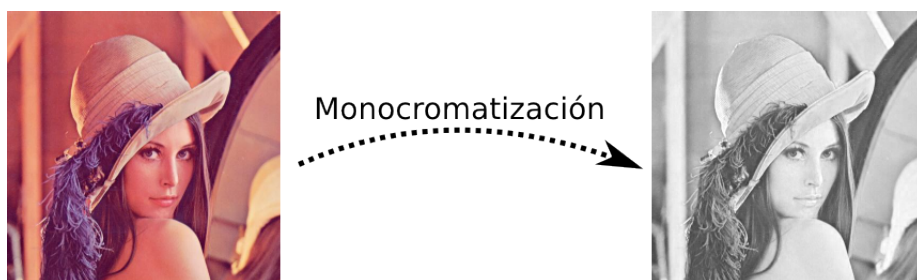


Figura 3: Imagen original y la obtenida luego de aplicar el algoritmo de monocromatización del trabajo práctico

Existen diversos algoritmos, de distinto grado de refinación y complejidad. En este trabajo práctico consideraremos uno sencillo, el algoritmo de monocromatización por máxima componente: Dado un píxel se asigna como valor de salida el máximo de los valores de sus componentes. Así por ejemplo a un píxel cuyas componentes son 0,127 y 200 el algoritmo asigna el valor 200 ya que es el máximo de las tres.

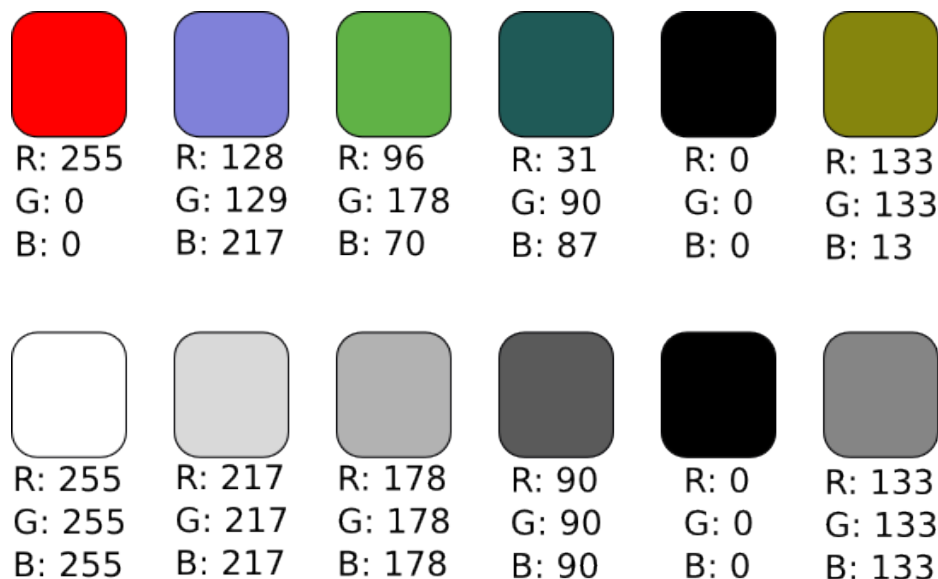


Figura 4: Distintos pixeles y el resultado de aplicar el algoritmo de monocromatización en ellos

Parte IV

Enunciado

El trabajo consiste en utilizar la arquitectura IA-32 para implementar el algoritmo de monocromatización por máxima componente, como se explicó en la sección 5. Para hacerlo deberán completar el archivo *MonocromatizarMax.asm* provisto por los docentes. Los demás archivos contienen implementaciones auxiliares realizadas por los docentes para facilitar las pruebas de la aplicación y no deben ser modificados. El trabajo práctico debe realizarse en GNU/Linux. Para llevarlo a cabo y probarlo deberán tener instalado el compilador del lenguaje *C* de GNU (*gcc*), el comando *make* y el ensamblador *nasm*. En el anexo 7 se muestra como instalar estas utilidades en una maquina con Ubuntu Linux.

La rutina a implementar recibe tres arreglos los cuales comienzan en las posiciones de memoria *red*, *green*, y *blue* representando las componentes de los pixeles de la imagen. Estas componentes se guardan en números de 32 bits. El tamaño de estos arreglos se encuentra en *tam* (son todos del mismo tamaño) y finalmente el resultado debe guardarse en un arreglo que comienza en *res*.

Para poder probar su trabajo deberán compilar y ensamblar el código. Para hacer esto, hay que pararse con una consola en el directorio donde se encuentren los archivos provistos por los docentes e invocar el comando *make*. A diferencia de los programas que escribimos en papel, el ensamblador es estricto frente a los errores sintácticos, de modo que si hay sintaxis inválida en el código se obtendrá un error como salida de *make*. Es por esta razón que es importante prestar atención a los mensajes de error. En el anexo 8 se muestran y explican algunos errores comunes que pueden obtener durante la realización del trabajo.

Luego de compilar, se genera un archivo `monocromatizador`. Para ejecutarlo deben hacer:
`./monocromatizador -i NOMBRE_ARCHIVO_ENTRADA -o NOMBRE_ARCHIVO_SALIDA`.

Parte V

Anexos

6. Instalación de Linux

Aquellos que no dispongan de una pc con Linux pueden, en el caso de no querer usar las maquinas de la facultad, instalarlo fácilmente. Incluso es posible instalarlo en una maquina virtual dentro de una pc con Windows. Hay muchos tutoriales en internet, por ejemplo:

- Para instalarlo en la pc:

<http://sliceoflinux.com/2011/04/28/instalar-ubuntu-11-04-paso-a-paso/>

- Para instalarlo en una maquina virtual:

<http://sliceoflinux.com/2009/11/05/instalar-ubuntu-dentro-de-windows-con-virtualbox/>

7. Instalación de herramientas necesarias

Para instalar las herramientas de compilación en Ubuntu, alcanza con abrir una consola e ingresar los siguientes comandos:

```
sudo apt-get install build-essential
sudo apt-get install nasm
sudo apt-get install gcc-multilib
```

Al ingresar el primer comando se solicitará el password para administrador a fin de poder realizar la instalación.

8. Errores comunes

- `monocromatizarMax.asm:38: error: comma, colon or end of line expected`: Este error indica que falta una coma o un fin de linea (un enter). En general se debe a que falta una coma entre los dos operandos de una instrucción, o a que falta un enter entre dos instrucciones que quedaron en la misma linea. El "38" indica que el error se detectó en la linea número 38 del archivo.

- `monocromatizarMax.asm:38: error: parser: instruction expected`: En la línea 38 debería estar el nombre de una instrucción pero hay otra cosa. Generalmente ocurre por un error de tipeo en el nombre de la instrucción (ejemplo `aad` en vez de `add`).
- `monocromatizarMax.asm:48: error: symbol 'NOMBRE_ETIQUETA' undefined`: Este error suele aparecer cuando se usa una etiqueta que no está definida, también puede deberse a que se ingreso erróneamente el nombre de un registro, por ejemplo `EXA` en lugar de `EAX`.
- `monocromatizarMax.asm:46: warning: byte value exceeds bounds`: Esta advertencia indica que se está usando un literal que no entra en 32 bits.
- `monocromatizarMax.asm:44: error: invalid combination of opcode and operands`: Este error ocurre cuando se usan modos de direccionamientos inválidos, por ejemplo inmediato para el destino. Puede ocurrir también si le falta el segundo operando a una instrucción de dos operandos.
- `monocromatizarMax.asm:46: error: operation size not specified`: Este error ocurre cuando se usan modos de direccionamientos inválidos, por ejemplo dos modos indirectos.
- Al ejecutar obtengo el siguiente mensaje: `Segmentation fault` o `Violación de segmento`. Esto ocurre cuando están leyendo o escribiendo una posición de memoria que no les corresponde, en general es por un error en un contador que hace que sigan escribiendo en un arreglo, pese a que ya se le acabaron las posiciones.