Arquitectura de Ordenadores

Juego de Instrucciones del Procesador Intel Pentium

Abelardo Pardo
abel@it.uc3m.es

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Ensamblador del Pentium

- Se ha presentado **un subconjunto muy reducido de instrucciones del Pentium**
- Un **programa ensamblador** se escribe con un editor que escriba los ficheros en formato ASCII.
- En este fichero se escribe **la secuencia de instrucciones a ejecutar** así como **los datos que necesita el programa**.
- Estos dos tipos de información se diferencian en el fichero mediante **dos secciones diferentes**.
- El comienzo de la **sección de datos** se denota por la palabra clave `.data`.
- El comienzo de la **sección de instrucciones** se denota por la palabra clave `.text`.
- En la sección de **código** se especifica la **primera instrucción a ejecutar** precediéndola de un nombre terminado en ':' y definiendo este nombre como global mediante la palabra clave **.globl** (o .global)
- A los nombres terminados en ':' situados a la izquierda de las instrucciones o datos se les denomina **etiquetas**.

Estructura Básica de un Programa Ensamblador


```
        .data /* Comienzo de la sección de datos */

        /* Definición de los datos a manipular por el programa */

        .text /* Comienzo de la sección de código */
        .globl start

start: /* primera instrucción */

        /* Instrucciones del programa */

        ret /* Instrucción de terminación */
```

- La primera instrucción del programa se comienza a ejecutar con **ciertos valores en los registros** y una cierta pila.
- El programa **debe dejar los valores de los registros y el estado de la pila idéntico a como estaba** antes de ejecutar la instrucción de terminación `ret`
- Esto implica que antes de utilizar un registro por primera vez **se debe guardar su contenido** para restaurarlo al terminar.
- Para esto se utiliza **la pila**.

Salvar y Restaurar el Contexto de un Programa


```
start: push %eax
       push %ebx
       push %ecx
       push %edx

       /* Instrucciones del programa que manipulan los 4 registros */

       pop %edx
       pop %ecx
       pop %ebx
       pop %eax

       ret /* Instrucción de terminación */
```

- Los registros se salvan en **cualquier orden** al principio del programa.
- Se restauran **en el orden inverso** justo antes de terminar el programa.
- Se recomienda primero escribir el programa y **al final** escribir el código para salvar y restaurar los registros.
- El registro %esp o puntero de pila debe apuntar, antes de la instrucción `ret` a **la misma posición que al comenzar el programa**.

Etiquetas

- El ensamblador permite definir datos y código que **serán almacenados en memoria** antes de su ejecución.
- Al escribir programas es preciso referirse a **posiciones de memoria** en las que están almacenados datos, o en las que hay una determinada instrucción.
- ¿Cómo se puede saber la dirección de memoria en la que **estará almacenado** un dato o una instrucción?
- El ensamblador resuelve este problema mediante la definición de **etiquetas**.
- Las etiquetas se anteponen a aquellos datos o instrucciones **que necesitan ser referenciadas**.
- **Ejemplo:**

```
msg:    .asciz "Hello World"
        ...
        push $msg
```

- Los **destinos** de las instrucciones de salto se especifican como etiquetas.

Valores de las Etiquetas

- Cuando una instrucción referencia a una etiqueta **el ensamblador** se encarga de sustituir la etiqueta por su valor.
- Para referirse a **la posición de memoria** en la que está la información con una determinada etiqueta se debe anteponer el prefijo **$** al nombre de la etiqueta.
- **Ejemplo:** push $msg
- La excepción a esta regla son **las instrucciones de salto** en las que, a pesar de referirse a una etiqueta, no es preciso incluir el prefijo.
- El nombre de una etiqueta **sin prefijo** se interpreta como **el contenido al que apunta**.
- **Ejemplo:**

```
msg:  .asciz "Hello World"
      ...
      mov $msg, %eax  /* Almacena en %eax la dirección de msg */
      mov msg, %eax   /* Almacena en el registro %eax los */
                      /* cuatro bytes de memoria almacenados a partir */
                      /* de la posición msg (0x48656C6C). */
```


El sufijo de Tamaño

- Es posible que en una instrucción **no se sepa con exactitud el tamaño de un operando**.
- **Ejemplo:** mov $-2, resultado
- ¿Cuántos bytes deben transferirse a la **posición de memoria referida por la etiqueta `resultado`**?
- El ensamblador notifica esta anomalía que se soluciona añadiendo al código de instrucción un **sufijo de tamaño**.
- Los posibles sufijos son: **'b' para byte, 'w' para 16 bits y 'l' para 32 bits**.
- Asumiendo 32 bits de tamaño de dato la instrucción corregida es: **movl $-2, resultado**.
- Este sufijo sólo es necesario cuando **ninguno de los operandos** tiene un tamaño prefijado.
- **Ejemplo:** mov $-2, %eax /* Tamaño es 32 bits: %eax = 0xFFFFFFFE */

Definición de Datos Enteros

- En la sección de datos se define una **secuencia de números enteros** mediante la palabra clave `.int`
- Los números deben ir separados por comas, y pueden escribirse en **decimal, binario (con prefijo 0b), octal (si empiezan por 0) o hexadecimal (con prefijo 0x)**.
- **Ejemplo:**

```
tabla:  .int 0b1110, 2210, 3340, 0xAA0132FF
```

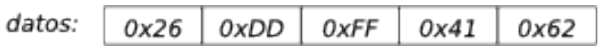
nums:	0x03	0x00	0x00	0x00
	0x04	0x00	0x00	0x00
	0x05	0x00	0x00	0x00
	0xAB	0x12	0x00	0x00
	0xAB	0x10	0x00	0x00
	0x38	0x00	0x00	0x00
	0x38	0x00	0x00	0x00
	0x15	0x00	0x00	0x00
	0xFA	0x0F	0x00	0x00

- Cada entero se almacena en **cuatro bytes consecutivos.**
- Los enteros se almacenan **uno a continuación del siguiente**.
- No se almacena **ningún dato referente al tamaño total ocupado por la secuencia**.

Definición de Bytes

- Análoga a la definición de enteros, mediante la palabra clave `.byte` se puede definir una secuencia de bytes.
- Los valores se separan **por comas**.
- Se almacenan en **posiciones consecutivas de memoria**.
- **Ejemplo:**

```
valores:  .byte 38, 0b11011101, 0xFF
```



- Si el valor no puede ser representado por un byte, el ensamblador lo **trunca** pero notifica de la anomalía.

```
          .byte 323
...
****  Warning:value 0x143 truncated to 0x43
```


Definición de Strings

- Existen **dos palabras clave** para definir una secuencia de caracters ASCII.
  - `.asciz`: Almacena el string especificado pero `termina su representación con un byte con valor 0`. Se puede detectar el **final del string**.
  - `.ascii`: Almacena **únicamente** el string especificado.
- **Ejemplo:**

```
string1: .ascii "Mens. 1"
string2: .asciz "Mens. 2"
```

- Los datos **se almacenan de forma consecutiva**.

$$\$string1 + 7 = \$string2$$

- La palabra clave `.string` es **sinónimo** de `.asciz`.
- Cualquiera de las tres variantes admite una **lista de strings separados por comas**.

Definición de Espacio en Blanco

- Se puede definir una **porción de memoria vacía**.
- La palabra clave para definir esta porción es `.space <tamaño>, <relleno>`.
- El parámetro `<tamaño>` es un número natural que expresa el tamaño **en bytes** del espacio definido.
- El parámetro `<relleno>` y la coma que le precede son opcionales, y si estan presentes indican el valor con el que rellenar los bytes de la porción de memoria. Si se omiten, se asume el valor **cero**.
- **Ejemplo:**

```
resultado:  .space 10, 0xAA
```


Aspectos de la Programación en Ensamblador

- Aspectos a tener en cuenta cuando se **escriben programas en lenguaje ensamblador**

1. El valor de los registros y el puntero de pila antes de ejecutar la última instrucción del programa **deben ser idénticos a los valores que tenían al comienzo**.
2. Debido al gran número de instrucciones disponibles y a la simplicidad de las operaciones **siempre hay más de una forma posible de realizar una operación**. Se debe tratar de elegir la más rápida.
3. Evitar las operaciones **innecesarias** (por ejemplo, salvar y restaurar todos los registros).
4. **Indentar el código**. Las etiquetas a principio de línea, las instrucciones todas a la misma altura (se tendrá en cuenta en las prácticas).
5. **Comentar** el código a nivel de bloques. No es preciso comentar cada instrucción, pero sí un conjunto de instrucciones (se tendrá **muy en cuenta** en las prácticas).

Ejemplo de Programación en Ensamblador

```
        .data
num1:   .int 0x00545604
num2:   .int 0x00A8F202
num3:   .int 0x00340141
resul1: .space 4
resul2: .space 4
resul3: .space 4
        .text
        .globl start
start:  push %eax
        mov num1, %eax
        add $10, %eax
        mov %eax, resul1
        mov num2, %eax
        add $20, %eax
        mov %eax, resul2
        mov num3, %eax
        mov %eax, resul3
        add $30, resul3
        pop %eax
        ret
```

- El programa toma los valores almacenados en `num1`, `num2`, `num3`, les suma 10, 20 y 30 respectivamente, y los almacena en `resul1`, `resul2`, `resul3`
- El espacio para el resultado se reserva mediante la palabra clave **.space**.
- No se puede realizar la suma y movimiento de dato **con una única instrucción para cada número**.
- Se utiliza el registro %eax para **realizar las operaciones**.
- El registro %eax **se salva al principio del programa y restaura al final**.
- Tanto la pila como los registros contienen al final del programa **los mismos datos que al principio**.
- El número a sumar se puede hacer **sobre el registro, o sobre memoria**.
- El programa **no escribe nada por pantalla**.
- ¿Qué pasa si suprimimos la **penúltima instrucción**?