x86: Práctica 2

Estructuras de control

Departamento de Ingeniería en Automática, Electrónica, Arquitectura y Redes de Computadores

Universidad de Cádiz

Índice

Introducción

Las **estructuras de control** se utilizan para **modificar el flujo de ejecución** de un programa. Su funcionamiento consiste en realizar **comparaciones** y, en función del resultado, **saltar** a una instrucción o a otra.

Índice

Realización de las comparaciones

- Para realizar comparaciones entre dos operandos se utiliza la **instrucción CMP**.
- **CMP *operando1*, *operando2***, resta al *operando1* el *operando2* y coloca las banderas del registro **EFLAGS** en función del resultado.
- El resultado de la **instrucción CMP** no se guarda. Si se quiere conservar el resultado, se puede usar en su lugar la **instrucción SUB**.

Interpretación de las comparaciones

Una vez colocadas las banderas, se pueden presentar dos casos:

- **Los operandos son sin signo:**
  - Si operando1 = operando2, entonces ZF = 1 (Zero Flag, bandera de cero) y CF = 0 (Carry Flag, bandera de acarreo)
  - Si operando1 $>$ opernado2, entonces ZF = 0 y CF = 0
  - Si operando1 $<$ operando2, entonces ZF = 0 y CF = 1
- **Los operandos son con signo:**
  - Si operando1 = operando2, entonces ZF = 1
  - Si operando1 $>$ operando2, entonces ZF = 0 y OF = SF (Overflow Flag y Sign Flag, bandera de desbordamiento y bandera de signo respectivamente)
  - Si operando1 $<$ operando2, entonces ZF = 0 y OF $\neq$ SF

Índice

Saltos

- Los saltos son instrucciones que permiten **modificar el flujo de ejecución del programa**, permitiendo que se ejecute una instrucción distinta a la inmediatamente posterior al salto.
- Todas las instrucciones de salto **reciben un único operando**. Este operando se utiliza para determinar la dirección del salto[1]. Normalmente suele utilizarse **etiquetas**.
- Debe indicarse entre el nemónico de la instrucción y el *operando*, si el salto es **hacia delante** (con el símbolo $>$) o si es **hacia detrás** (con el símbolo $<$). En el primer caso es **obligatorio**, el segundo es opcional.

[1] Para más información consultar los manuales del ensamblador y de Intel.

Tipos de saltos

Existen dos tipos de saltos:

- **Saltos incondicionales:** Estos saltos se ejecutan siempre. Se realizan mediante la **instrucción JMP**.
- **Saltos condicionales:** Estos saltos se ejecutan si se cumple una determinada condición. Existen varias instrucciones para realizar saltos condicionales en función de la condición que se evalúe.

Saltos condicionales

Algunas instrucciones que realizan saltos condicionales son:

- **JZ** salta si ZF = 1
- **JNZ** salta si ZF = 0
- **JO** salta si OF = 1
- **JNO** salta si OF = 0
- **JS** salta si SF = 1
- **JNS** salta si SF = 0
- **JC** salta si CF = 1
- **JNC** salta si CF = 0
- **JP** salta si PF = 1 (PF, Parity Flag, bandera de paridad, si es uno la paridad es par, si es cero, impar)
- **JNP** salta si PF = 0

Saltos condicionales complejos

Operandos sin signo

Dado que a veces es necesario comprobar más de una bandera a la vez, el ensamblador ofrece instrucciones que comprueban las banderas necesarias para **hacer las comparaciones en una sola instrucción.** Para los operandos sin signo, estas son:

- **JE** salta si operando1 = operando2
- **JNE** salta si operando1 $\neq$ operando2
- **JL, JNGE** salta si operando1 $<$ operando2 (Jump Less, Jump Not Greater Equal)
- **JLE, JNG** salta si operando1 $\leq$ operando2
- **JG, JNLE** salta si operando1 $>$ operando2
- **JGE, JNL** salta si operando1 $\geq$ operando2

Saltos condicionales complejos

Operandos con signo

Para los operandos con signo las instrucciones son:

- **JE** salta si operando1 = operando2
- **JNE** salta si operando1 $\neq$ operando2
- **JB, JNA**E salta si operando1 $<$ operando2 (Jump Below, Jump Not Above Equal)
- **JBE, JNA** salta si operando1 $\leq$ operando2
- **JA, JNBE** salta si operando1 $>$ operando2
- **JAE, JNB** salta si operando 1 $\geq$ operando2

Índice

Traducir estructuras if a ensamblador

Una estructura como la siguiente en C:

```
1  if (condicion) {
2      //Hacer instrucciones del bloque if.
3  }
```

Traducir estructuras if a ensamblador

Podría traducirse de la siguiente manera:

```
1      ;Codigo para establecer las baderas
2
3   ;Inicio bloque if
4      jxx end_if     ;xx es la condicion del if
5                     ;negada, puesto que saltara
6                     ;fuera del if si se cumple.
7
8      ;Instrucciones del bloque if
9
10  end_if:
11  ;Fin bloque if
12
13     ;Resto de instrucciones.
```

Índice

Traducir estructuras if-else a ensamblador

Una estructura como la siguiente en C:

```
1  if (condicion) {
2      //Hacer instrucciones del bloque if.
3  }
4  else {
5      //Hacer instrucciones del bloque else.
6  }
```

Traducir estructuras if-else a ensamblador

Podría traducirse a ensamblador de la siguiente manera:

```
1       ;Codigo para establecer las baderas
2
3       jxx if  ;xx es la condicion para hacer if.
4
5   else:
6       ;Instrucciones del bloque else
7       jmp endif
8
9   if:
10      ;Instrucciones del bloque if
11
12  endif:
13
14      ;Resto de instrucciones
```

Índice

Traducir un bucle while a ensamblador

Un bucle while, como el que a continuación se presenta en C:

```
1  while (condicion) {
2      //Cuerpo del bucle
3  }
```

Traducir un bucle while a ensamblador

Podría traducirse a ensamblador de la siguiente manera:

```
1  while:
2
3      ;Codigo que establezca las banderas
4      jxx endwhile    ;xx es la condicion negada
5      ;Cuerpo del bucle
6      jmp while   ;Vuelve al principio.
7                  ;jxx es el que termina el bucle.
8
9  endwhile:
```

Índice

Traducir un bucle do-while a ensamblador

El siguiente bucle do-while en C:

```
1  do {
2      //Cuerpo del bucle
3  } while (condicion);
```

Traducir un bucle do-while a ensamblador

Podría traducirse a ensamblador de la siguiente manera:

```
1  do:
2
3      ;Cuerpo del bucle
4      ;Codigo para establecer las banderas
5      jxx do     ;Condcion para que se repita
6
7  ;Fin del do-while
8
9      ;Resto del codigo
```

Índice

Traducir un bucle for a ensamblador

Un bucle for, como el que a continuación se presenta en C:

```
1  for (i = 10; i<10; i++) {
2      //Cuerpo del bucle
3  }
```

Traducir un bucle for a ensamblador

Podríamos traducirla a ensamblador de la siguiente manera:

```
1     mov ecx, 10      ;Metemos en ecx el numero
2                      ;de repeticiones
3  for:
4
5     ;Cuerpo del bucle
6     loop for
7
8  ;Fin del bucle for
9
10    ;Resto del programa.
```

Traducir un bucle for a ensamblador

- La **instrucción loop** decrementa en una unidad el contenido de **ECX**. Si tras el decremento el valor de **ECX** es 0, no se realiza el salto y se continúa la ejecución por la siguiente instrucción. En caso contrario salta al lugar indicado por la etiqueta.
- La instrucción loop tiene otras dos variantes: la **instrucción loope/loopz** y la **intrucción loopne/loopnz**. Estas instrucciones decrementan **ECX** y realizan el salto mientras que **ECX** $\neq 0$ y **ZF** = 1 (las primeras) o **ZF** = 0 (las segundas)
- Ninguna de estas instrucciones modifican el registro **EFLAGS**.