

Assembly – Tradução de Mecanismos de Controle

Fluxo de Execução em Assembly

- Normalmente, comandos em C e instruções de máquina são executados sequencialmente
- Algumas construções em C (mecanismos de decisão e repetição) quebram a execução sequencial
 - a sequência de operações executada depende do resultado de testes aplicados aos dados: **execução condicional**
- A linguagem de máquina provê mecanismos básicos para isso, permitindo
 - testar valores de dados (resultado de operações aritméticas e lógicas)
 - alterar o fluxo de controle do programa conforme esse resultado

Testes sobre Dados

- A CPU mantém um conjunto de bits (EFLAGS) que descreve o resultado da última operação aritmética/lógica realizada
 - ZF, SF, CF, OF
- Combinações desses bits podem ser usadas como condições para alterar o fluxo de controle
 - decisão sobre qual a próxima instrução a ser executada

Instruções de Desvio

- Podem alterar a sequência de execução (“goto”)
- O destino do desvio é indicado no código assembly por um “label simbólico” (jump direto)
 - traduzido pelo assembler + linker para um endereço de memória

L1:

```
    cmp1    %ebx, %ebx  
    jz     L2     →  condicional  
    ...  
    jmp    L1     →  incondicional
```

L2:

```
    xorl   %eax, %eax
```

Desvios Condicionais

Instrução	Sinônimo	Descrição
je Label	jz	equal/zero
jne Label	jnz	not equal/ not zero
js Label		negative
jns Label		non negative
jg Label	jnle	signed > (greater)
jge Label	jnl	signed >= (greater or equal)
j1 Label	jnge	signed < (less)
jle Label	jng	signed <= (less or equal)
ja Label	jnbe	unsigned > (above)
jae Label	jnb	unsigned >= (above or equal)
jb Label	jnae	unsigned < (below)
jbe Label	jna	unsigned <= (below or equal)

Mecanismos de controle: 'if'

- Suponha o código C:

```
if (a==b) c=d; → %eax, %ebx, %ecx, %edx  
d=a+c;
```

- Esse código pode ser traduzido da seguinte forma:

```
cmpl %eax, %ebx
```

```
jne depois_if → condição é negativa do if!
```

```
movl %edx, %ecx
```

```
depois_if:
```

```
movl %eax, %edx
```

```
addl %ecx, %edx
```

Exemplo do Laboratório

- Impressão dos números pares de um array

```
movl $S2, %ecx    /* ecx = &S2; */
loop:
movl (%ecx), %eax /* eax = *ecx; */
cmpl $0, %eax     /* eax == 0 ? */
je    after
movl %eax, %edx
andl $0x01, %edx  /* testa se par */
jnz   done
call  printnum
done:
addl $4, %ecx     /* ecx += 4; */
jmp  L1
after:
```

```
if (a % 2 == 0)
    printnum(a);
```

```
if (!t)
    goto done;
then-statement
done:
```

Traduzindo "if-else"

```
if (test-expr)
  then-statement;
else
  else-statement;
```

↓ Esquema geral ↓

```
t= test-expr;
if (!t)
  goto false;
then-statement
goto done;
false:
  else-statement
done:
```

```
d = 16;
a = 10;
if (d < a) c= a - d;
else c = d -a;
```

↓ Caso Específico ↓

```
movl $0x10, %edx
movl $0xA, %eax
cmpl %eax, %edx
jge L1
  movl %eax, %ecx
  subl %edx, %ecx
  jmp L2
L1: /* false */
  movl %edx, %ecx
  subl %eax, %ecx
L2: /* done*/
```

Traduzindo “while”

```
while (test-expr)
  Body
```

Esquema geral

```
loop:
  t= test-expr;
  if (!t)
    goto after;
  Body
  goto loop;
after:
```

```
while (a<=b ){
  ...
  a++;
}
```

Caso Específico

```
loop:
  cml %ebx, %eax
  jg after /* se a>b */
  [
    incl %eax
    jmp loop
  ]
after:
```

Exemplo do Laboratório

- Percorre array enquanto elemento != 0 (fim)

```
movl $S2, %ecx    /* ecx = &S2; */
loop:
movl (%ecx), %eax /* eax = *ecx; */
cmpl $0, %eax    /* eax == 0 ? */
je    after
movl %eax, %edx
andl $0x01, %edx /* testa se par */
jnz   done
call  printnum
done:
addl $4, %ecx    /* ecx += 4; */
jmp  L1
after:
```

```
while (*p != 0){
    a = *p;
    if (a % 2 == 0)
        printnum(a);
    p++;
}
```

```
loop:
    if (!t) goto after
    Body
    goto loop
after:
```

Traduzindo o “for”

```
for (Init; Test; Update )  
  Body
```



Versão “while”

```
Init;  
while (Test) {  
  Body ;  
  Update ;  
}
```



Versão goto

```
Init;  
loop:  
  if (!Test)  
    goto done;  
  Body ;  
  Update ;  
  goto loop;  
done:
```

Exemplo do Laboratório

- Percorrendo array de quatro elementos

```
L1:  movl  $S2, %ecx    /* ecx = &S2; */
     movl  $0, %ebx   /* ebx = 0; */
     cmpl  $4, %ebx   /* if (ebx == 4) ? */
     je    L2         /* goto L2 */
     movl  (%ecx), %eax /* eax = *ecx; */
     call  printnum   /* imprime eax */
     addl  $1, %ebx   /* ebx += 1; */
     addl  $4, %ecx   /* ecx += 4; */
     jmp   L1        /* goto L1; */
```

L2:

```
for (b = 0; b != 4; b++) {
    ...
}
```

```
Init;
loop:
    if (!Test)
        goto done;
    Body ;
    Update ;
    goto loop;
done:
```

Avaliando condições com “curto circuito”

- A condição de teste em uma construção de controle pode conter operadores lógicos
 - Exemplo: `((c>a) || ((a ==1) && (d < b)))`
- Em C e outras linguagens de alto nível, a avaliação é interrompida assim que o resultado é conhecido (“*curto circuito*”)
 - `(x || y)` se x resulta em true, não avalia y
 - `(x && y)` se x resulta em false, não avalia y
- Isto é refletido no código Assembly gerado

Exemplo de curto circuito

```
if ((a==b) || (c<d)){  
    a = c;  
}  
c = d;
```

```
    cmp %ebx, %eax  
    je L1  
    cmp %edx, %ecx  
    jge L2  
L1:  
    movl %ecx, %eax  
L2:  
    movl %edx, %ecx
```

