Команды управления потоком вычислений (2)

Сложные условия

- Логические операции (И, ИЛИ, НЕ) в условиях не вычисляются
- Вместо них - несколько условных переходов
- **Порядок переходов и расположение меток определяет сложное условие**

Пример

на C:

```
if (ax>bx && ax<cx)
 ax += bx;
```

на ассемблере:

```
    cmp ax, bx
    jng endif

    cmp ax, cx
    jnl endif

    add ax, bx
endif: …
```

Пример

на C:

```
if (ax>bx || ax<cx)
 ax += bx;
```

на ассемблере:

```
    cmp ax, bx
    jg if1

    cmp ax, cx
    jl if1

    jmp endif
if1:
    add ax, bx
endif:…
```

Пример If ... then ... else

на C:

```
if (ax > bx)
   ax += bx;
else
   cx -= bx;
```

на ассемблере:

```
    cmp ax, bx
    jle else

      ; if ax > bx
    add ax, bx
    jmp endif

else:
      sub cx, bx
endif:…
```

Пример If ... then ... else

на C:

```
if (ax < bx)
   ax += bx;
else if (ax > bx)
   cx -= bx;
else
   bx = 2;
```

на ассемблере:

```
     cmp ax, bx
     jl less
     jg greater
       ; if ax = bx
     mov bx, 2
     jmp endif
less:
     add ax, bx
     jmp endif
greater:
     sub cx, bx
endif:…
```

Пример switch

на C:

```
switch(ax) {
case 0:
    ax += bx;
    break;
case 1:
case 2:
    cx -= bx;
    break;
default:
    bx = 2;
}
```

на ассемблере:

```
    tst ax, ax
    je case0
    cmp ax, 1
    je case12
    cmp ax, 2
    je case12
        ; default
    mov bx, 2
    jmp endswitch
case0:
    add ax, bx
    jmp endswitch
case12:
    sub cx, bx
endswitch:…
```

“**goto** – зло!”

- Переходы на метки затрудняют понимание кода, приводят к ошибкам
- НО - в ассемблере без них никак не обойтись!
- Особенно – переходы назад, т.к. они создают циклы

“**goto** – зло!”

Нужно:

- Минимизировать число меток
  – располагать участки кода в порядке выполнения
- Избегать переходов назад
  – исключение – циклы
- Давать меткам осмысленные имена (не номера)

Как **не** нужно делать

на C:

```
действие 1;

if (cc)
   действие 2;

действие 3;
```

на ACM:

```
        действие 1

        Jcc метка1
метка2:

        действие 3
        …

метка1:
        действие 2
        jmp метка2
```

Как нужно делать

на C:

```
действие 1;

if (cc)
   действие 2;

действие 3;
```

на ACM:

```
      действие 1

      Jncc пропустить2

      действие 2

пропустить2:
      действие 3
```

Условная пересылка данных

команда	dst	src
CMOVcc dst, src	регистр память	регистр память число

- Коды условий **cc** – те же, что и **J**cc

Пример

```
; числа в ax, bx, cx
; вычислить ax = max(ax, bx, cx)

cmp   ax, bx
cmovl ax, bx    ; ax = max(ax, bx)
cmp   ax, cx
cmovl ax, cx    ; ax = max(ax, bx, cx)
```

С использованием **jcc** получится больше кода

Пример – вычисление модуля

```
    test eax, eax
    jns skipabs
    neg eax
skipabs:
```

выполняется:
2-3 команды
1 условие

```
doneg:
    neg eax
    js doneg
```

выполняется:
2-4 команды
1-2 условия

но: код короче

Вычисление значения условия

команда	dst	
SETcc dst	регистр память } 8 бит	dst = 1, если **cc** dst = 0, иначе

- Коды **cc** – те же, что и для **J**cc**, MOV**cc

- Эквивалентно

```
mov dst, 0
cmovcc dst, 1
```

Организация циклов

команда	дополнит. условие	dst
loop dst	нет	short ptr (-128...+127)
loope dst **loopz** dst	ZF = 0	
loopne dst **loopnz** dst	ZF != 0	

- Уменьшает CX (ECX) на 1
- Если ((E)CX != 0) и (дополнит. условие)

  то переход на метку
- Не меняет флаги!

Организация циклов

- **loop** можно заменить на

```
dec (e)cx
jnz метка
```

(НО - это длиннее на 1 байт и **dec** меняет флаги!)

Организация циклов

- **loope/loopz** можно заменить на

```
cikl:   …
        …
        jnz skip
        dec (e)cx
        jnz cikl
skip:
```

Пример – вычислить 10!

- С использованием **loop** и **cx** как счетчика

```
        mov ax, 1
        mov cx, 10
for10:
        imul cx
        loop for10
```

Условный переход по значению CX

команда	расшифровка	условие
jcxz	CX zero	CX = 0
jecxz	ECX zero	ECX = 0

- Обычно используется перед циклом loop, который может выполниться 0 раз

Пример

```
; просуммировать числа от 1 до cx

        xor ax, ax
        jcxz endfor1
for1:
        add ax, cx
        loop for1
endfor1:
```

Что будет, если убрать jcxz перед циклом?

Вложенные циклы

- Проблема: счетчик внешнего цикла (CX) меняется во внутреннем цикле

- Пример: вычислить $\sum_{i=1}^{10} \sum_{j=1}^{i} j^2$

Решение 1

- Счетчики циклов в **разных регистрах**

```
    xor bx, bx; bx - результат

    mov si, 10; dx = i
fori:
    mov cx, si; cx = j
forj:
    mov ax, cx
    imul cx    ; ax = j*j
    add bx, ax; результат += j*j
    loop forj      ; конец цикла по j
    dec si         ; конец цикла по i
    jnz fori
```

внутренний цикл

Решение 2

- Использование **стека**

```
    xor bx, bx

    mov cx, 10; цикл по i
fori:
    push cx          ; сохранили cx

    ; cx уже проинициализирован!
    ; внутренний цикл

    pop cx           ; восстановили cx
    loop fori        ; конец цикла по i
```

Решение 3

- Использование **переменных**
  - Только для внешних циклов!


```
    xor bx, bx

    mov i, 10          ; начало цикла по i
fori:

    mov cx, i          ; начало цикла по j
    ; внутренний цикл

    dec i
    jnz fori           ; конец цикла по i
```