Введение в архитектуру ЭВМ

Лекция 4. Машинная арифметика. Дополнительный код. Система команд. Программа `2to10`

План лекции

- Машинная арифметика
- Дополнительный код
- Обзор системы команд
- Программа перевода числа из двоичной системы счисления в десятичную, `2to10.S`

Машинная арифметика

В каждой системе счисления есть таблица сложения. Для двоичной системы она имеет вид:

`0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0` и `1` переноса.

Пример 1.

```
  1 0 0 1 1 1 0 1
+
  0 1 0 1 0 1 0 1
-------------------------
  1 1 1 1 0 0 1 0
```

Пример 2. Получение таблиц для переводов `2 <-> 8, 16`.

```
  0 0 0 0
+
  0 0 0 1
-------------
  0 0 0 1
```

```
  +
    0 0 0 1
  -------------
    0 0 1 0
  +
    0 0 0 1
  -------------
    0 0 1 1
  +
    0 0 0 1
  -------------
    0 1 0 0
```

Продолжите самостоятельно.

Дополнительный код

В архитектуре IA-32 программист может интерпретировать байты, слова и двойные слова как беззнаковые или знаковые целые. При *беззнаковой интерпретации* все биты рассматриваются как цифры двоичного числа, что определяет следующие диапазоны:

Байт: 00 — FF **или** 0 — 255.

Слово: 0000 — FFFF **или** 0 — 65535.

Двойное слово: 00000000 — FFFFFFFF **или** 0 — 4294967295 (четыре миллиарда двести девяносто четыре миллиона девятьсот шестьдесят семь тысяч двести девяносто пять).

При *знаковой интерпретации* старший бит трактуется как индикатор знака, его значение 1 указывает на отрицательное число, 0 — на положительное, что определяет следующие диапазоны:

Байт: 0x80 — 7F **или** -128 — +127.

Слово: 0x8000 — 7FFF **или** -32768 — +32767.

Двойное слово: 80000000 — 7FFFFFFF **или** -2147483648 — +2147483648.

NBNB. **Значение отрицательного числа представляется в дополнительном коде, обеспечивавшем одинаковые алгоритмы операций сложения и вычитания для знаковых и беззнаковых целых. Операции умножения и деления для беззнаковых целых выполняются командами** mul{s}, div{s}, **для знаковых — командами** imul{s}, idiv{s} **соответственно, где** s **— суффикс размера операндов** {b|w|l}.

Алгоритм получение дополнительного кода.

1. Инвертировать биты числа.

2. К результату прибавить единицу.

Пример.

Получим представление в байте отрицательного числа -5 **из представления числа** +5.

Исходное число: 0 0 0 0 0 1 0 1

Шаг 1: 1 1 1 1 1 0 1 0

Шаг 2: +

0 0 0 0 0 0 0 1

---

1 1 1 1 1 0 1 1

Убедиться самостоятельно, что применение этого алгоритма к полученному двоичному представлению числа -5 дает число +5.

Команда `neg{b|w|l}` `операнд` **записывает в** `операнд` **его дополнительный код.**

Обзор системы команд

`MOV` `Пересылка данных`
`Move data`

Переп	Напр	Прер	Шаг	Знак	Нуль	Всп	Четн	Перенос
O	D	I	T	S	Z	A	P	C

Код	Команда	Краткое описание
88 /r	mov r/m8,r8	из регистра-байта в r/m-байт
89 /r	mov r/m16,r16	из регистра-слова в r/m-слово
89 /r	mov r/m32,r32	из регистра-двойного слова в r/m-двойное слово
8A /r	mov r8,r/m8	из r/m-байта в регистр-байт
8B /r	mov r16,r/m16	из r/m-слова в регистр-слово
8B /r	mov r32,r/m32	из r/m-двойного слова в регистр-двойное слово
8C /r	mov r/m16,Sreg	из сег.регистра в r/m-слово
8E /r	mov Sreg,r/m16	из r/m-слова в сег.регистр
A0	mov AL,moffs8	из байта (сег:пер) в AL
A1	mov AX,moffs16	из слова (сег:пер) в AX
A1	mov EAX,moffs32	из двойного слова (сег:пер) в EAX
A2	mov moffs8,AL	из AL в байт (сег:пер)
A3	mov moffs16,AX	из AX в слово (сег:пер)
A3	mov moffs32,EAX	из EAX в двойное слово (сег:пер)
B0+rb	mov reg8,imm8	из байта в команде в регистр
B8+rw	mov reg16,imm16	из слова в команде в регистр
B8+rw	mov reg32,imm32	из двойного слова в команде в регистр
C6	mov r/m8,imm8	из байта в команде в r/m-байт
C7	mov r/m16,imm16	из слова в команде в r/m-слово

C7	mov r/m32,imm32	из двойного слова в команде в r/m-двойное слово

Первый операнд получает значение второго операнда (NBNB в синтаксисе AT&T — ВТОРОЙ операнд получает значение первого).

ADD Сложение
Add

Перепо Р	Напр D	Прер I	Шаг T	Знак SP	Нуль ZP	Всп AP	Четн PP	Перенос CP

Код	Команда	Краткое описание

04 ib	ADD AL,imm8	байт в команде + AL
05 iw	ADD AX,imm16	слово в команде + AX
05 id	ADD EAX,imm32	двойное слово в команде + EAX
80 /0 ib	ADD r/m8,imm8	байт в команде + r/m-байт
81 /0 iw	ADD r/m16,imm16	слово в команде + r/m-слово
81 /0 id	ADD r/m32,imm32	двойное слово в команде + r/m-слово
82 /0 ib	ADD r/m8,imm8	байт в команде + r/m-байт
83 /0 ib	ADD r/m16,imm8	знаково-расширяемый байт в команде + r/m-слово
83 /0 ib	ADD r/m32,imm8	знаково-расширяемый байт в команде + r/m-двойное слово
00 /r	ADD r/m8,r8	регистр-байт + r/m-байт
01 /r	ADD r/m16,r16	регистр-слово + r/m-слово
01 /r	ADD r/m32,r32	регистр-слово + r/m-двойное слово
02 /r	ADD r8,r/m8	r/m-байт + регистр-байт
03 /r	ADD r16,r/m16	r/m-слово + регистр-слово
03 /r	ADD r32,r/m32	r/m-слово + регистр-двойное слово

Приемник и источник складываются. Первый операнд (NBNB в синтаксисе AT&T - ВТОРОЙ) получает значение результата, устанавливаются флаги. Если байт в команде складывается с операндом-словом или двойным словом (код команды 83 /0), то до сложения из байта в ЦП формируется операнд-слово или двойное слово, младший байт которого равен байту в команде, а биты старших байтов совпадают с его старшим битом (знаковое расширение).

Jcc Условный короткий переход
Jump if condition is met

Переп	Напр	Прер	Шаг	Знак	Нуль	Всп	Четн	Перенос
O	D	I	T	S	Z	A	P	C

Код	Команда	Условие перехода
По значению одного флага		
70 cb	JO rel8	было переполнение (OF=1)
0F 80 cw/cd	JO rel16/rel32	было переполнение (OF=1)
71 cb	JNO rel8	не было переполнения (OF=0)
0F 81 cw/cd	JNO rel16/rel32	не было переполнения (OF=0)
72 cb	JC rel8	был перенос (CF=1)
0F 82 cw/cd	JC rel16/rel32	был перенос (CF=1)
73 cb	JNC rel8	не было переноса (CF=0)
0F 83 cw/cd	JNC rel16/rel32	не было переноса (CF=0)
74 cb	JE rel8	равно (ZF=1)
	JZ rel8	результат равен 0 (ZF=1)
0F 84 cw/cd	JE rel16/rel32	равно (ZF=1)
	JZ	результат равен 0 (ZF=1)
75 cb	JNE rel8	не равно (ZF=0)
	JNZ rel8	не было нуля (ZF=0)
0F 85 cw/cd	JNE rel16/rel32	не равно (ZF=0)
	JNZ	не было нуля (ZF=0)
78 cb	JS rel8	был знак минус (SF=1)
0F 88 cw/cd	JS rel16/rel32	был знак минус (SF=1)
79 cb	JNS rel8	не было знака минус (SF=0)
0F 89 cw/cd	JNS rel16/rel32	не было знака минус (SF=0)
7A cb	JP rel8	была четность (PF=1)
	JPE rel8	количество единиц четно (PF=1)
0F 8A cw/cd	JP rel16/rel32	была четность (PF=1)
	JPE	количество единиц четно (PF=1)
7B cb	JNP rel8	не было четности (PF=0)
	JPO rel8	количество единиц НЕчетно (PF=0)
0F 8B cw/cd	JNP rel16/rel32	не было четности (PF=0)
	JPO	количество единиц НЕчетно (PF=0)
После операций над ЗНАКОВЫМИ операндами		
7C cb	JL rel8	меньше (SF<>OF)
	JNGE rel8	не “больше или равно” (SF<>OF)
0F 8C cw/cd	JL rel16/rel32	меньше (SF<>OF)
	JNGE	не “больше или равно” (SF<>OF)

7D cb	JGE rel8	больше или равно (SF=OF)
	JNL rel8	не меньше (SF=OF)
0F 8D cw/cd	JGE rel16/rel32	больше или равно (SF=OF)
	JNL	не меньше (SF=OF)
7E cb	JLE rel8	меньше или равно (ZF=1 и SF<>OF)
	JNG rel8	не больше (ZF=1 и SF<>OF)
0F 8E cw/cd	JLE rel16/rel32	меньше или равно (ZF=1 и SF<>OF)
	JNG	не больше (ZF=1 и SF<>OF)
7F cb	JG rel8	больше (ZF=0 и SF=OF)
	JNLE rel8	не “меньше или равно” (ZF=0 и SF=OF)
0F 8F cw/cd	JG rel16/rel32	больше (ZF=0 и SF=OF)
	JNLE	не “меньше или равно” (ZF=0 и SF=OF)
После операций над БЕЗзнаковыми операндами		
72 cb	JB rel8	ниже (CF=1)
	JNAE rel8	не “выше или равно” (CF=1)
	JC rel8	был перенос (CF=1)
0F 82 cw/cd	JB rel16/rel32	ниже (CF=1)
	JNAE	не “выше или равно” (CF=1)
	JC	был перенос (CF=1)
73 cb	JAE rel8	выше или равно (CF=0)
	JNB rel8	не ниже (CF=0)
	JNC rel8	не было перноса (CF=0)
0F 83 cw/cd	JAE rel16/rel32	выше или равно (CF=0)
	JNB	не ниже (CF=0)
	JNC	не было переноса (CF=0)
76 cb	JBE rel8	ниже или равно (CF=1 или ZF=1)
	JNA rel8	не выше (CF=1 или ZF=1)
0F 86 cw/cd	JBE rel16/rel32	ниже или равно (CF=1 или ZF=1)
	JNA	не выше (CF=1 или ZF=1)
77 cb	JA rel8	выше (CF=0 и ZF=0)
	JNBE rel8	не “ниже или равно” (CF=0 и ZF=0)
0F 87 cw/cd	JA rel16/rel32	выше (CF=0 и ZF=0)
	JNBE	не “ниже или равно” (CF=0 и ZF=0)
Проверка регистра (E)CX		
E3 cb	JCXZ rel8	регистр CX содержит 0
	JECXZ rel8	регистр ECX содержит 0

Программа перевода числа из двоичной системы счисления в десятичную, 2to10.S

```
.include "my-macro"  # подключение файла с макроопределениями

/*
Получение 10-х цифр двоичного числа и вывод их на терминал
 */

.data    # секция данных, распределение памяти

n:   .short 2345   # число для перевода 2->10

digits_of_n:  .ascii "□□□□□" # коды символов 10-х цифр - 5 пробелов

    .align 16# выравниваем на 16 битовую границу

length: .long 0         # количество 10 цифр в числе

ten:     .short 10     # делитель


.text # секция команд процесора

.global _start    # точка входа - глобальная метка

_start:
    movl $0, %ebx      # счетчик делений
    movw n, %ax        # готовим деление
    movl $4,%esi       # начальное значение индекса - на старший
                       # байт строки digits_of_n

nextdigit:

#        Т.к. переводим 16 битовое слово, то работаем с 16 бит рег

        movw  $0, %dx           # готовим деление в цикле
        idivw ten               # делим объединенные регистры dx:ax на 10
                                # частное в  ax, остаток в dx

#       Превращаем значение остатка в код цифры
```

```
      addw $48,%dx

#    !!! пишем код цифры остатка в байт строки 10digits-of-n
#    на который указывает  %esi

     leal digits_of_n(%esi), %ecx # демонстрация исполнительного адреса

#    команда leal вычисляет ИА и посылает его в регистр 2 операнда
#    теперь в %ecx - адрес байта куда запишется код цифры
#    команда пристутствует только для учебных целей


        movb %dl,digits_of_n(%esi)
```

Лекционный комментарий.

1. Алгоритм перевода получает коды цифр десятичного представления от младших к старшим. Т.е. их надо размещать в байты строки digits_of_n начиная с адреса digits_of_n + 4, т.е

NBNB значение %esi должно меняться в цикле от 4 до 0 (если получаем все пять цифр)!

Адрес байта digits_of_n+	0	1	2	3	4
Значение %esi	0	1	2	3	4
Цифра разряда	a4	a3	a2	a1	a0

```
#    готовим обработку след. цифры

        incl %ebx               # счетчик делений + 1
        decl %esi           # устанавливаем индекс на байт для
#                           след. кода цифры
        cmpl $0, %eax           # частное = 0 ?
        jg   nextdigit          # НЕТ, продолжаем
        movl %ebx, length       # ДА, сохраняем результат

#    Вывод десятичного представления, сис. вызов write

     movl $4, %eax              # номер сист. вызова write
        movl $1, %ebx            # 1 дескриптор стандартного вывода
        movl $digits_of_n, %ecx # адрес памяти с выводимыми символами
```

```
        movl $5, %edx           # количество байтов для вывода
        int $0x80               # выполнить системный вызов

    Finish                  # конец работы, возврат в ОС
                            # (макро из файла my-macro)

.end   # последняя строка исходного текста
```