Системное программирование

Лекция 3

Работа с битами. Команды сдвига.

Логические команды.

Адресное пространство. Способы адресации.

Организация сравнения.

Циклы. Функции.

Код команды и количество тактов выполнения.

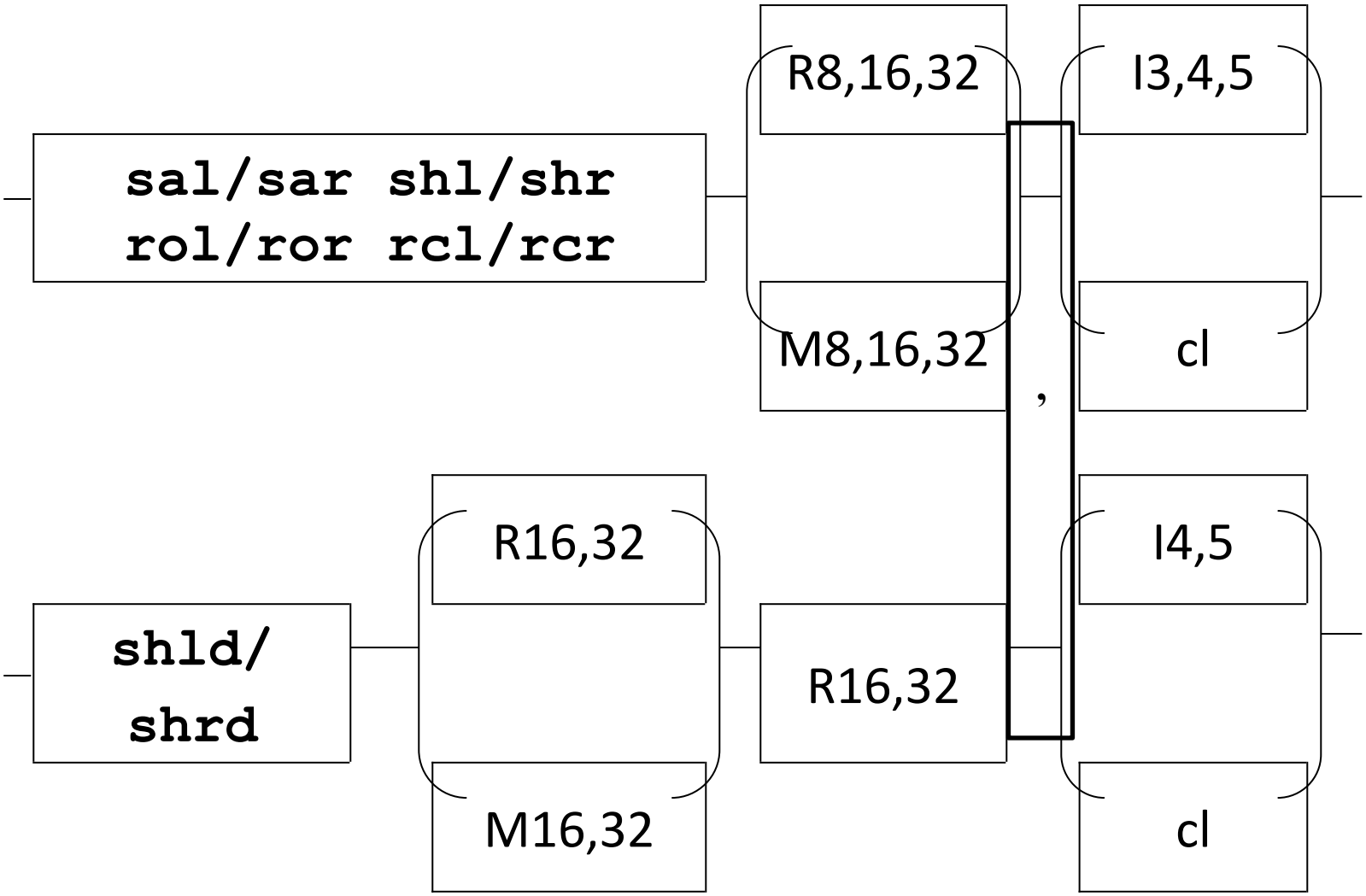
Команды работы с битам

- **sal**
- **sar**
- **shr**
- **shl**

- **rcl**
- **rcr**

Команды сдвига

Команды работы с битами



Использование команд сдвига

Команды работы с битами

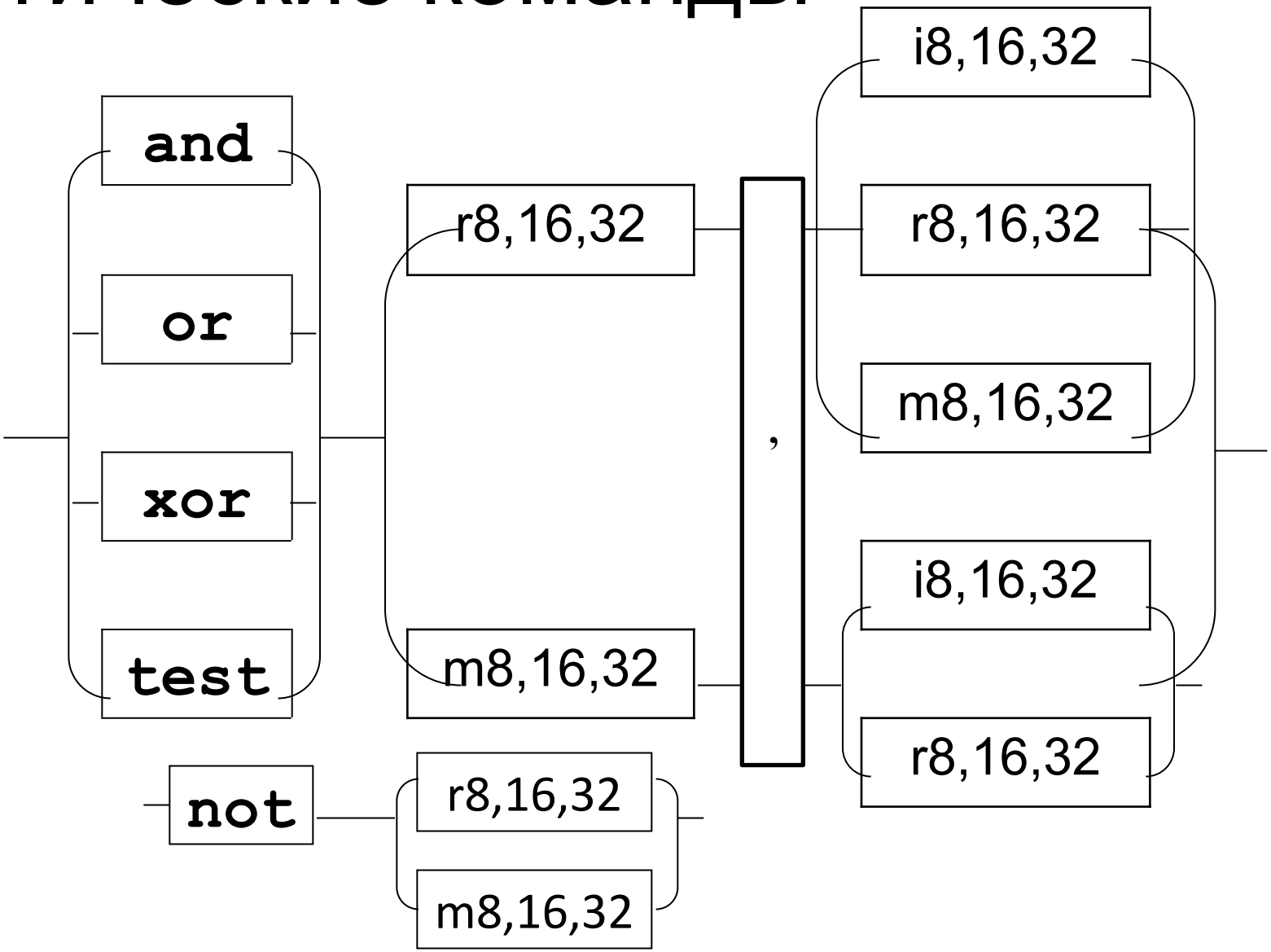
- «Логический» сдвиг
  - **shl** …; ***cf*** ← op ← 0
  - **shr** …; 0 → op → ***cf***
  - «переворот» байт
  - умножение/деление
- В приёмник
  - **shld** …;
  - **shrd** …;
  - источник → приёмник
  - последний бит → ***cf***
- Циклические
  - **rol** …
  - **ror** …
  - дублирование в ***cf***
  - **rcl** …
  - **rcr** …
  - замыкание через ***cf***
- Арифметические
  - **sal** …; тоже что **shl**
  - **sar** …;
  - знак → мантисса → ***cf***

Поразрядные логические команды

- `and`
- `or`
- `xor`

- `not`

Поразрядные логические команды



Использование логических команд

- Сброс значения регистра

```
xor EAX, EAX
```

Примеры использования команд

Команды работы с битами

Логические команды

Пересылка данных

```
CDQ
```

```
NOT EAX
```

```
NEG EAX
```

Примеры использования команд

Команды работы с битами

Логические команды

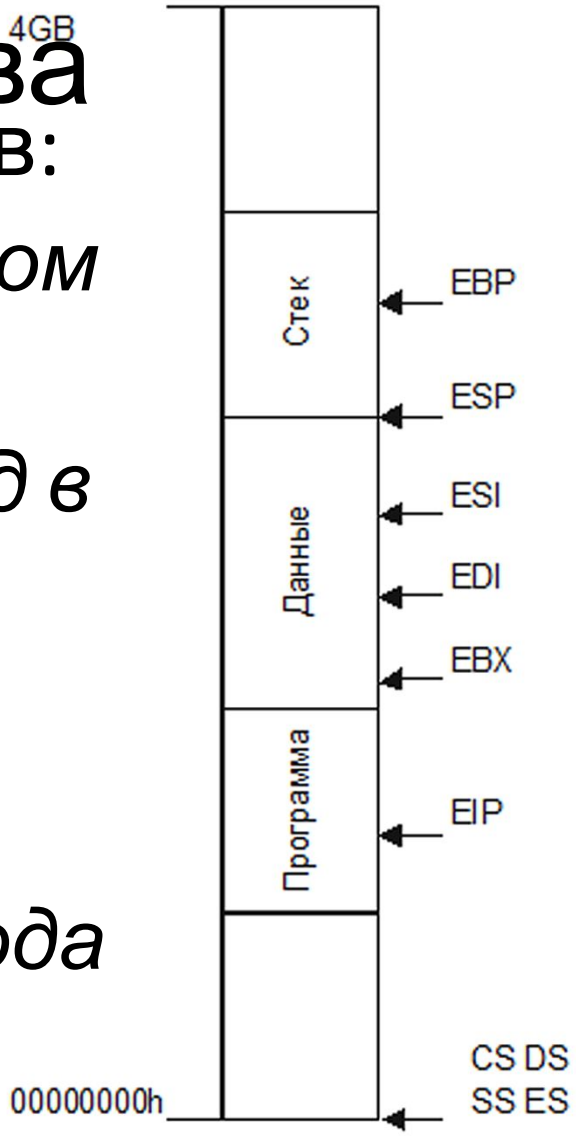
Пересылка данных

- Быстрое умножение
- Вычисление абсолютного значения числа
  (если a<0, то a=-a)
- Определения минимума из двух чисел
  (если b<a, то a=b)
- Выбор из двух чисел по условию
  (если a<>0 то a=б, иначе a=c)

Модель адресного пространства

Способы задания операндов:

- *неявно на микропрограммном уровне*
- *непосредственный операнд в самой команде*
- *указание регистра*
- *указание памяти*
- *указание порта ввода/вывода*



Flat модель защищенного режима

Способы адресации

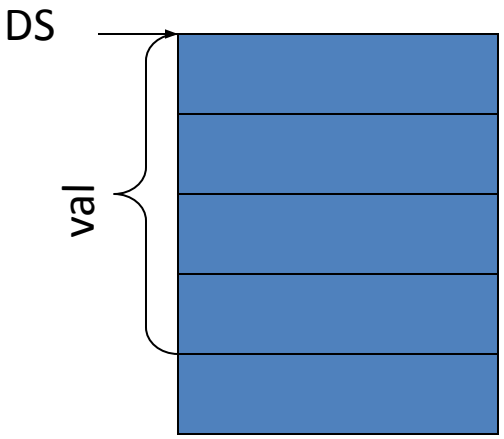
Прямая

Косвенная

Прямая адресация

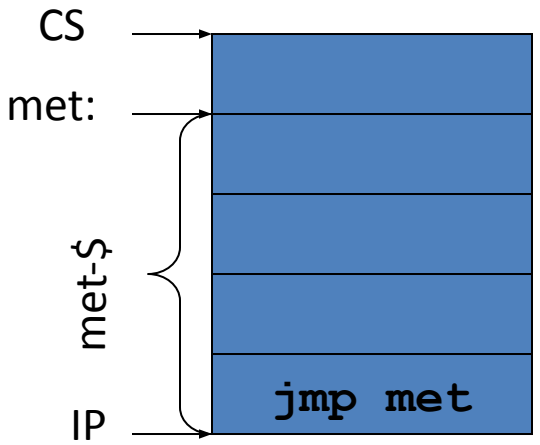
- Абсолютная

  `mov ebx,val`



- Относительная

  `jmp met`



Косвенная адресация

– адресация с помощью заключенных в квадратные скобки регистров, содержащих адрес памяти

Директива переопределения типа **`ptr`**

– применяется для переопределения или уточнения типа метки или переменной, определяемых выражением.

Тип может принимать одно из следующих значений:

`byte, word, dword, qword, tbyte, near, far.`

- **`mov ebx, dword ptr mem[ecx*4+eax]`**

Косвенная базовая адресация

– **регистровая адресация**

эффективный адрес операнда может находиться в любом из регистров общего назначения, кроме ***esp*** и ***ebp***

Пример,

`mov ax,[ecx]`

команда помещает в регистр **`ax`** содержимое слова по адресу из сегмента данных со смещением, хранящимся в регистре **`ecx`**.

- Так как содержимое регистра легко изменить в ходе работы программы, данный способ адресации позволяет динамически назначить адрес операнда для некоторой машинной команды.
- Используется для организации циклических вычислений и для работы с различными структурами данных типа таблиц или массивов.

Косвенная базовая адресация со смещением

– **регистровая адресация со смещением**

является дополнением предыдущего и предназначен для доступа к данным с известным смещением относительно некоторого базового адреса

Пример

`mov ax,[edx+3h]`

команда помещает в регистр **`ax`** слова из области памяти по адресу: содержимое **`edx + 3h`**

`mov ax,mas[dx]`

команда пересылает в регистр **`ax`** слово по адресу: содержимое **`dx`** плюс значение идентификатора **`mas`**, равное смещению этого идентификатора относительно начала сегмента.

- Используется для доступа к элементам структур данных, когда смещение элементов известно заранее, на стадии разработки программы, а базовый (начальный) адрес структуры должен вычисляться динамически, на стадии выполнения программы.

Индексная адресация

похожа на косвенную базовую адресацию со смещением. Для формирования эффективного адреса используется один из регистров общего назначения. Но индексная адресация связана с возможностью так называемого масштабирования содержимого индексного регистра.

Пример

```
mov ax,mas[si*2]
```

команда помещает в регистр **ax** слово по адресу: значение идентификатора **mas** плюс значение индексного регистра **si** масштабируемое в 2 раза.

- Используется для организации циклических вычислений и для работы с массивами при условии , что размер элементов массива составляет 1, 2, 4 или 8 байт

Базово -индексная адресация и базово-индексная со смещением

Эффективный адрес формируется как сумма трех составляющих:

- содержимого базового регистра
- содержимого индексного регистра с масштабированием
- значение поля смещения в команде

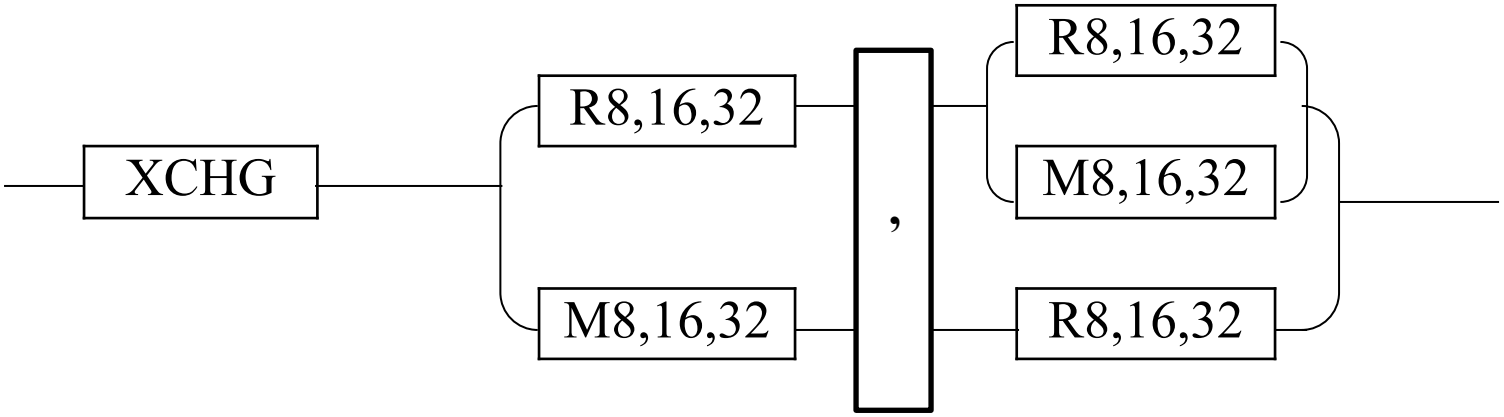
Пример

```
mov eax,[esi][edx]
mov eax,[esi+5][edx]
add ax,array[esi*4][ebx]
```

- Масштабирование допускается использовать для любых регистров общего назначения.

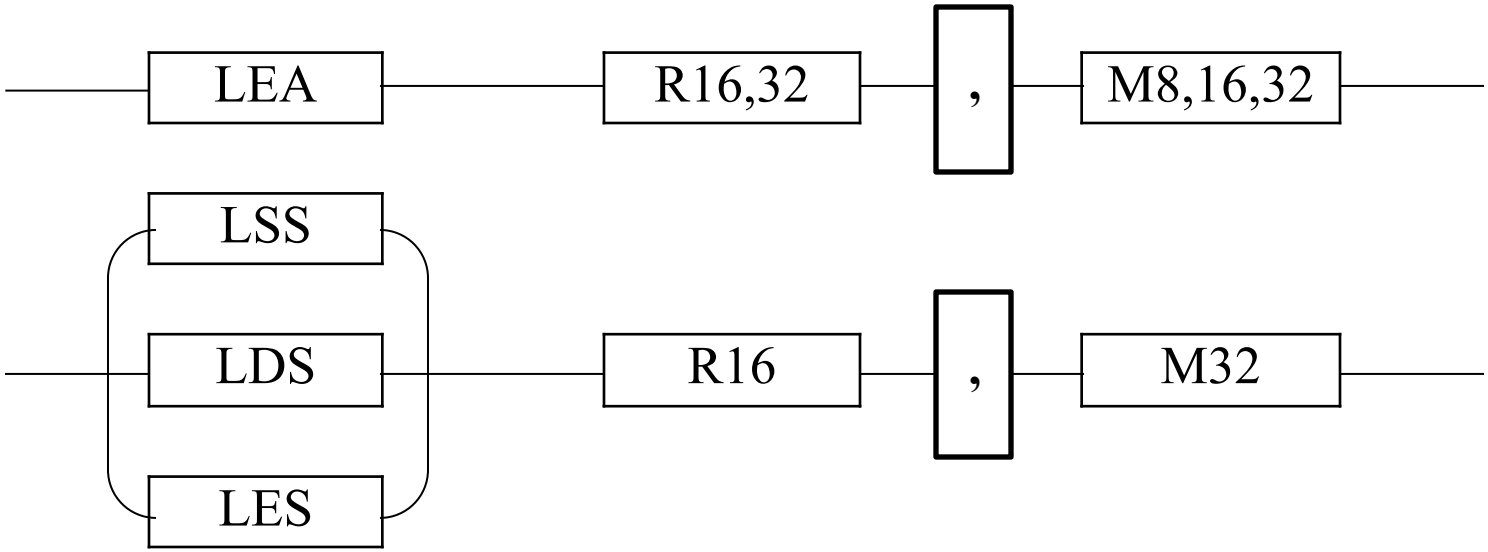
Инструкция XCHG

Пересылка данных



Инструкции загрузки адреса

Пересылка данных



Команды передачи управления

Передача управления

Безусловная	Взаимодействие с процедурами	Условные переходы			Циклы
JMP	**CALL**	**JL=JNGE**	**JC**	**JNC**	**LOOP**
	RET	**JLE=JNG**	**JP**	**JNP**	**LOOPE**
		JG=JNLE	**JZ**	**JNZ**	**LOOPZ**
	INT	**JGE=JNL**	**JS**	**JNS**	**LOOPNE**
	IRET	**JB=JNAE**	**JO**	**JNO**	**LOOPNZ**
		JBE=JNA			
		JA=JNBE	**JCXZ**		
		JAE=JNB	**JECXZ**		

Условные переходы

Передача управления

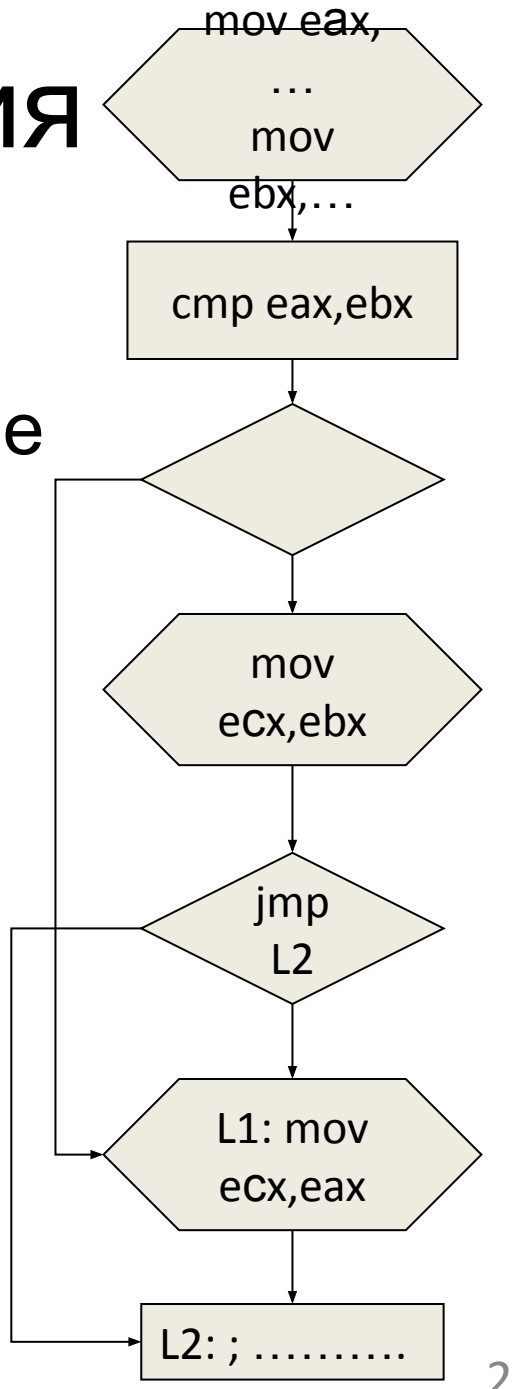
- `J?? <op> ;` много вариантов
- По результатам сравнения
  - **E**qual, **N**ot **E**qual
  - **G**reater, **L**ess, **G**reater or **E**qual, **L**ess or **E**qual (со знаком)
  - **A**bove, **B**elow, **A**bove or **E**qual, **B**elow or **E**qual (без знака)
  - Примеры: `JL` - если `SF != OF`, `JB` - если `CF=1`
- По состоянию одного флага
  - [**N**ot] flag `{Z|S|C|O|P}F` set to `1`»
  - `JZ, JNZ,…,JP,JNP`
- По состоянию счётчика
  - `JECXZ` – обход цикла для реализации «предусловия»

Команда сравнения

- `CMP op1,op2`

  «безрезультатное» сравнение

```
Пример:
a=…;
b=…;
if (a   b) c=a;
else c=b;
```

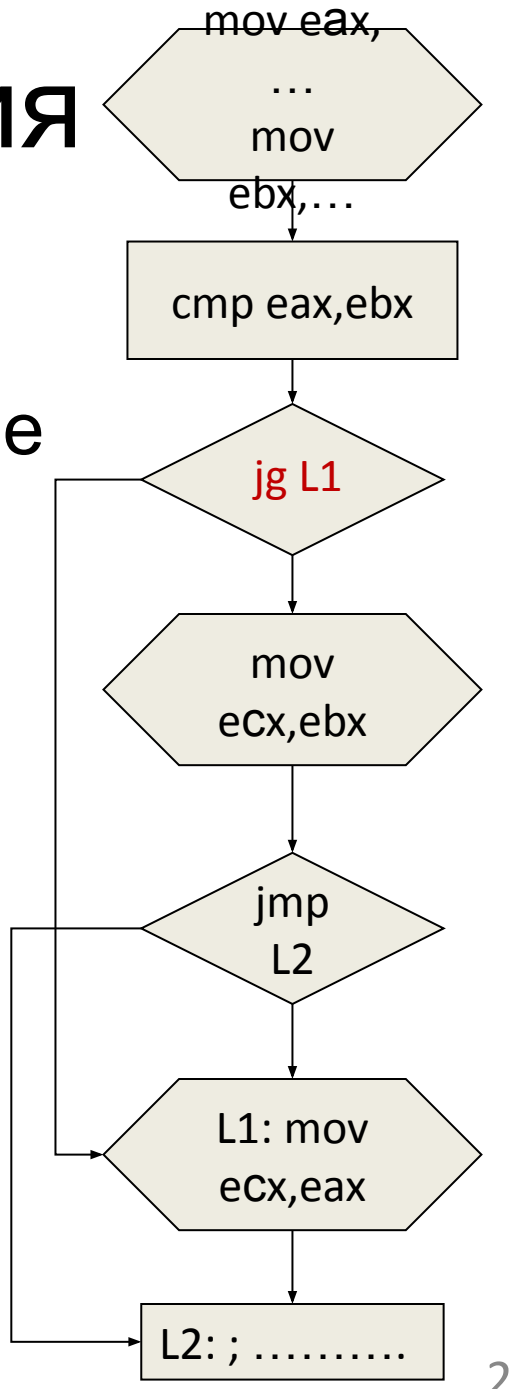


Команда сравнения

- `CMP op1,op2`

  «безрезультатное» сравнение

```
Пример:
a=…;
b=…;
if (a > b) c=a;
else c=b;
```



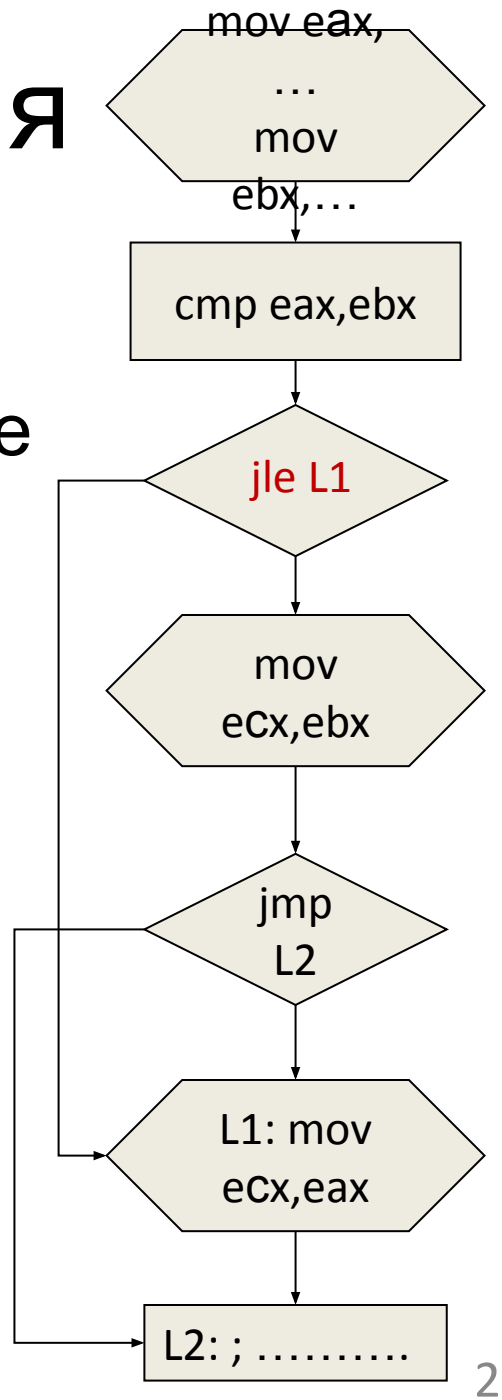
Команда сравнения

- `CMP op1,op2`

  «безрезультатное» сравнение

  Пример:

```
a=…;
b=…;
if (a <= b) c=a;
else c=b;
```



Циклы

LOOP* <op>;

LOOP: `if (! ECX)goto <метка>.`

счётчик цикла в `ECX`,

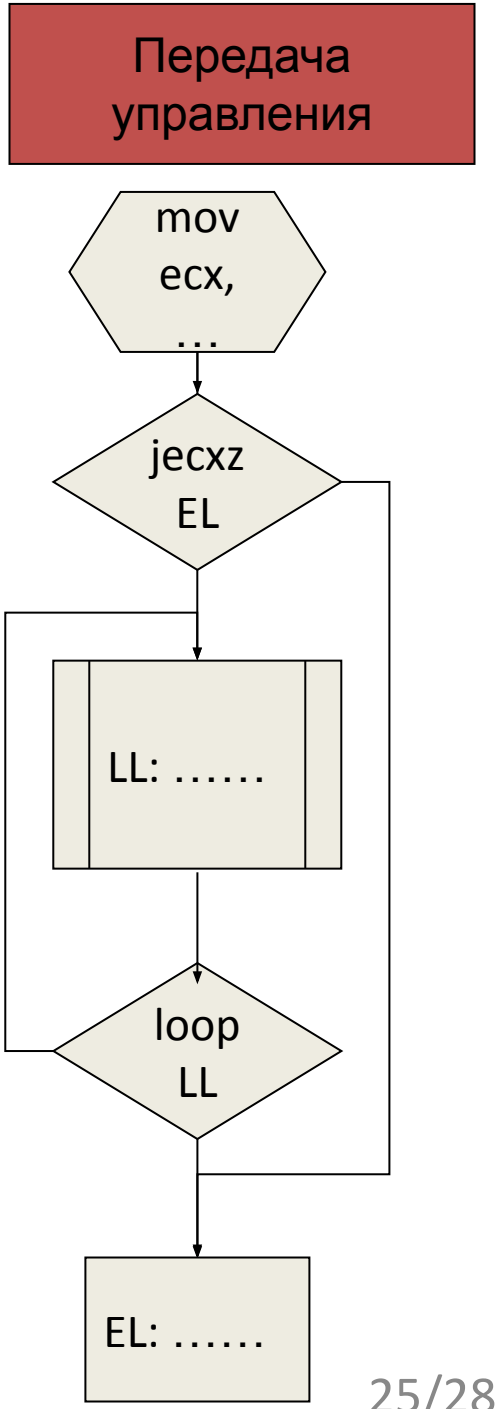
LOOPE/LOOPZ: Поиск отличного

`ecx!=0`

`if(! ECX || ZF)goto <метка>`

LOOPNE/LOOPNZ: Поиск требуемого

`if(! ECX || !ZF)goto <метка>`



Функции: логика работы

Передача управления

```
CALL Calc
```
Логика: PUSH EIP

EIP = EIP+ смещение к процедуре Calc

```
Calc PROC
        PUSH EBP
    Тело_процедуры
        POP EBP
      RET
Calc ENDP
```
Логика: POP EIP

Код команды

Структура машинной команды процессора

Число байт	Компонент команды
0 или 1	Префикс команды (для строковых команд)
0 или 1	Префикс изменения размера адреса
0 или 1	Префикс изменения размера операнда
0 или 1	Префикс замены сегмента
1 или 2	Код операции
0 или 1	Байт MRM - (mod,reg,r/m)
0 или 1	Байт SIB - (scale,index,base)
0,1,2 или 4	Поле для задания адреса
0,1,2 или 4	Непосредственный операнд

Примеры кода команды

```
mov EBX,ECX;
89CB
mov BX,CX
6689CB
mov ECX,6[EBX+EDI*4]
8B4CBB06
```

Количество тактов выполнения команды

- **Latency** — число тактов, необходимое инструкции для того, чтобы следующая зависимая инструкция могла начать использовать результат работы этой инструкции.
- **Throghput** — число тактов, необходимое для того, чтобы следующая независимая по данным инструкция с тем же кодом могла начать выполняться

Команда	Latency	Throghput
ADD/AND/CMP/SUB/TEST mem, reg;	1	1
BTC/BTR/BTS mem, reg	11	10
BSF/BSR reg, reg	16	15
IMUL/MUL EAX, r32	6	5
IDIV r/m32;	57	56
JCC; JMP reg;	1	1
JCXZ; JECXZ;	4	1
JMP mem;	2	1
LEA reg, mem	1	1
MOV reg, mem;	1	1
NEG/NOT mem	10	9
NEG/NOT reg; NOP	1	0.5
POP mem	3	2
PUSH mem	2	1
POP reg; PUSH reg	1	1
POPA ; POPAD	9	8

Команда	Latency	Throghput
RCL mem, 1; RCL reg, 1	1	1
RCL mem, CL;	14	13
RCL reg, CL;	14	14
RCR mem, 1	7	6
RCR reg, 1	5	4
RCR mem, CL; RCR reg, CL;	12	11
ROL; ROR; SAL; SAR; SHL; SHR	1	1
SETcc	1	1
SHLD r32, r32; SHRD r32, r32	2	1
SHLD m32, r32; SHRD m32, r32	4	3
CALL mem	2	2
CALL reg	1	1
RET	79	