

# Языки ассемблера и двоичный код

Лекция №4  
АКОС 2019-2020

# Стадии трансляции

- Препроцессинг текста; на выходе - текст
- Из исходного текста - Abstract Syntax Tree и таблицы символов; формат - внутренний для реализации компиляторов
- Из AST получаем:
  - либо код на языке ассемблера (текст)
  - либо двоичный код

# Языки ассемблера

- Последовательность команд - линейная
- Нет вложенных конструкций
- Не бывает вычислимых выражений (точнее, это *syntax sugar*)
- Текст распознается регулярным выражением, а не контекстно-свободной грамматикой

# Языки ассемблера

- Из AST получаем:
  - либо код на языке ассемблера (текст)
  - либо двоичный код
- Из текста программы можно выделить отдельные фрагменты, которые кодируют бинарные инструкции
- Последовательность бинарных инструкций позволяет восстановить текст на ASM

# Кодирование команд

- RISC: одна команда - это машинное слово
  - зная начало кода, можно извлечь любую инструкцию
- CISC: одна команда - это произвольное количество байт
  - нужно прочитать и декодировать несколько байт, чтобы найти следующую команду

# Команды и регистры

- Процессор, в общем случае, может выполнять некоторые действия только над значениями регистров
- Регистр - не просто быстрая ячейка памяти, а ячейка, к которой можно обратиться напрямую
- Доступ памяти - достаточно сложная операция
- Но: некоторые процессоры (x86) имеют разные способы адресации, позволяющие адресовать память

# Кодирование команд (AVR)

## ADD – Add without Carry

### Description

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

- (i) (i)  $Rd \leftarrow Rd + Rr$

Syntax:

Operands:

Program Counter:

- (i) ADD Rd,Rr

$0 \leq d \leq 31, 0 \leq r \leq 31$

$PC \leftarrow PC + 1$

16-bit Opcode:

0000	11rd	dddd	rrrr
------	------	------	------

# Кодирование команд (AVR)

## LDI – Load Immediate

### Description

Loads an 8-bit constant directly to register 16 to 31.

Operation:

(i)  $Rd \leftarrow K$

Syntax:

(i) LDI Rd,K

Operands:

$16 \leq d \leq 31, 0 \leq K \leq 255$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------



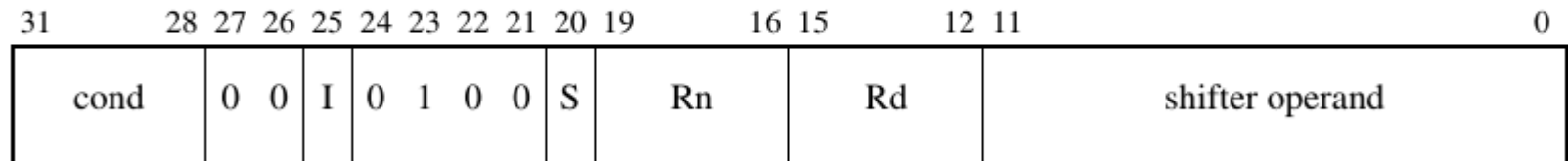
# Что нужно закодировать

(или что делает ассемблер)

- Какая именно команда
- С какими регистрами она работает
- Константы:
  - могут встречаться в численных операциях
  - адреса (метки) в инструкциях перехода

# Кодирование ARM-32

## ADD



- **cond** - условие выполнения команды
- **I** - флаг, определяющий, что закодировано в **shifter\_operand**
- **S** - флаг, определяющий, нужно ли обновлять регистр статуса
- **Rn, Rd** - первый аргумент операции и куда записать результат

Примеры:

ADD **r0, r1, r2** //  $r0 \leftarrow r1 + r2$

ADD**S** **r0, r1, r2** //  $r0 \leftarrow r1 + r2$ , обновить флаги Z,V,C,N

ADD**EQ** **r0, r1, r2** // Если Z, то  $r0 \leftarrow r1 + r2$

ADD **r0, r1, r2, lsl #3** //  $r0 \leftarrow r1 + (r2 \ll 3)$  [5bit shift; 2bit type; 0; 4bit reg]

ADD **r0, r1, #0xFF** //  $r0 \leftarrow r1 + 0b0000'1111'1111$

ADD **r0, r1, #0x200** //  $r0 \leftarrow r1 + (0b0000'0010 \text{ ROR } 0b1100)$

# Ключевая проблема

Нельзя впихать невпихуемое

- Длина команды - 32 бит
- Часть из них заняты OPcode и номерами регистров

# Циклический сдвиг констант (ARM)

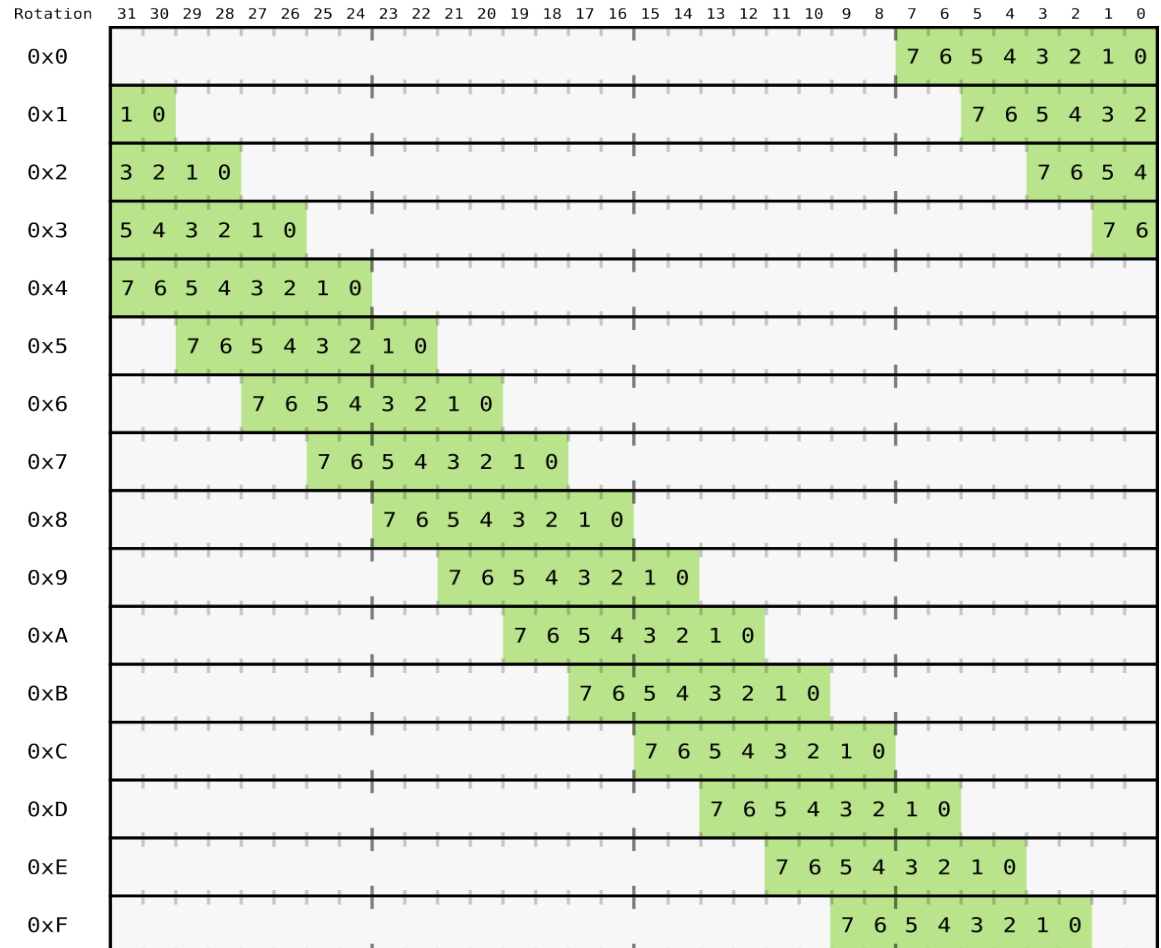
- Под константы зарезервировано 12 бит
- 8 бит - значение
- 4 бит - сдвиг со вращением

# Циклический сдвиг констант (ARM)

```

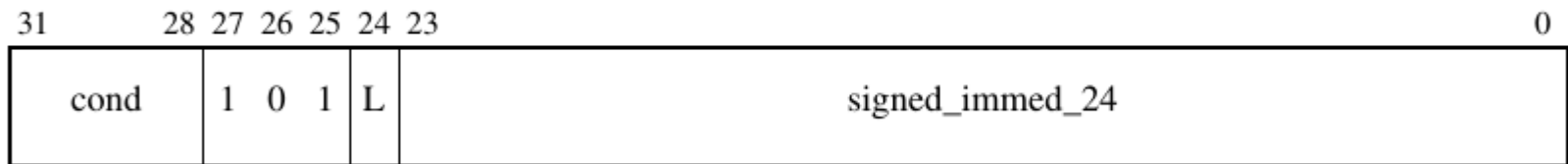
255 = 0b11111111 ROR 0
256 = 0b00000001 ROR 24
512 = 0b00000010 ROR 24
1024 = 0b00000100 ROR 24
1024 = 0b00000001 ROR 22

```



# Кодирование ARM-32

**B, BL**



$\pm 2^{23}$  - это  $\pm 8\text{Mб}$

# Переходы на метки

- В пределах +/- 8Мб - нет проблем
- Прыжок на далекую функцию - через "трамплин":
  - одна из причин использования PLT

*Нарисовать на доске: как устроено размещение в памяти*

# Procedure Linkage Table

```
function@plt:  
    add    ip, pc, #0  
    add    ip, ip, #OFFSET_TO_TABLE_BEGIN  
    ldr    pc, [ip, #OFFSET_TO_FUNCTION_INDEX]
```



# Procedure Linkage Table

- Решает проблему "длинных" прыжков
- Позволяет размещать часть кода в заранее не известном месте адресного пространства

*Показать вывод objdump -d*

# Про кодирование команд

- Расширения ARM:
  - Thumb -- 16 битные инструкции
  - Jazelle -- декодирование байткода Java

# Про кодирование команд

- Регистровые машины выполнения
- Стековые машины выполнения

# Байткод java

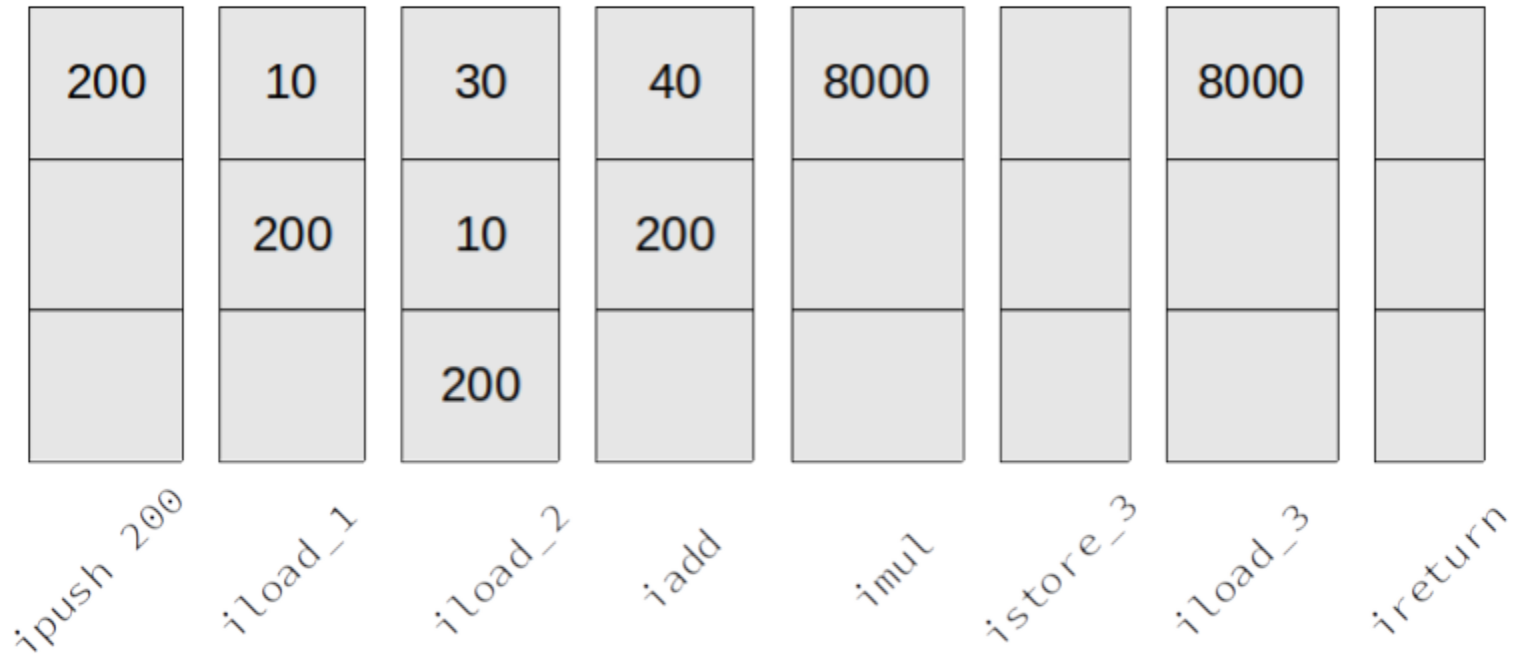
```
.....
2: int someFunc(int a, int b){
3:     int c = 200*(a+b)
4:     return c;
5: }
.....

int someFunc(int, int);
  descriptor: (II)I
  flags:
  Code:
    stack=3, locals=4,
  args_size=3
    0: sipush 200
    3: iload_1
    4: iload_2
    5: iadd
    6: imul
    7: istore_3
    8: iload_3
    9: ireturn
  lineNumberTable:
    line 3: 0
    line 4: 8
```

`int c = 200*(a+b)`

Пусть  $a = 10$ ,  $b = 30$

```
0: sipush 200
3: iload_1
4: iload_2
5: iadd
6: imul
7: istore_3
8: iload_3
9: ireturn
```



0	1	2	3
this	a	b	c

# Уровни абстракции

- Высокоуровневый язык
- Intermediate Language:
  - bytecode (Java, CLI или PyPy)
  - LLVM - абстракция от ассемблеров разных архитектур
- [Язык ассемблера]
- Двоичный код

# Зачем понимать кодирование команд

**Хочу денег**

Закончился 30-дневный пробный период  
использования Великой Программы.

Для продолжения использования необходимо  
ввести серийный номер:

—  —  —

# Уголовный кодекс РФ, глава 21 «Преступления против собственности»

## *Статья 159.6. Мошенничество в сфере компьютерной информации*

*1. Мошенничество в сфере компьютерной информации, то есть хищение чужого имущества или приобретение права на чужое имущество путем ввода, удаления, блокирования, модификации компьютерной информации либо иного вмешательства в функционирование средств хранения, обработки или передачи компьютерной информации или информационно-телекоммуникационных сетей, - наказывается штрафом в размере до ста двадцати тысяч рублей или в размере заработной платы или иного дохода осужденного за период до одного года, либо обязательными работами на срок до трехсот шестидесяти часов, либо исправительными работами на срок до одного года, либо ограничением свободы на срок до двух лет, либо принудительными работами на срок до двух лет, либо арестом на срок до четырех месяцев.*



# Как ломают

```
void TrialEndDialog_OkPressed()  
{  
    const char * sn =  
Dialog_GetSerialNumber();  
    if (CheckSerialNo(sn)) {  
        ContinueLaunch();  
    }  
    else {  
        ShowErrorMessage();  
        ExitProgram();  
    }  
}
```

# Как ломают

```
. . . . .
01: callq   CheckSerialNo    ;; вызов [bool CheckSerialNo]
02: testb   $1, %al          ;; сравнение результата с 1
03: je      $05              ;; если OK, то переход к 05
04: jmp     $08              ;; переход к 08
05: movb    $0, %al          ;; очистка регистра AL
06: callq   ContinueLaunch   ;; вызов [bool ContinueLaunch]
07: jmp     $0C              ;; переход к строке после if {}
08: movb    $0, %al          ;; очистка регистра AL
09: callq   ShowErrorMessage  ;; вызов [void ShowErrorMessage]
0A: movb    $0, %al          ;; очистка регистра AL
0B: callq   ExitProgram       ;; вызов [void ExitProgram]
. . . . .
```

# Как ломают

```
. . . . .
01: callq    CheckSerialNo    ;; вызов [bool CheckSerialNo]
02: testb    $1, %al          ;; сравнение результата с 1
03: je jmp    $05            ;; если OK, то переход к 05
04: jmp      $08              ;; переход к 08
05: movb $0, %al              ;; очистка регистра AL
06: callq    ContinueLaunch   ;; вызов [bool ContinueLaunch]
07: jmp      $0C              ;; переход к строке после if {}
08: movb $0, %al              ;; очистка регистра AL
09: callq    ShowErrorMessage  ;; вызов [void ShowErrorMessage]
0A: movb $0, %al              ;; очистка регистра AL
0B: callq    ExitProgram       ;; вызов [void ExitProgram]
. . . . .
```

