Семинар #1. Машина Тьюринга. Оценки сложности.

Определение. *Алфавит Σ – конечное множество символов.*

Определение. *Слово w – конечная последовательность символов.*

Определение. *Язык L – множество слов в алфавите Σ.*

Определение. *k-ленточная машина Тьюринга (MT^k) – кортеж $\langle \Sigma, Q, q_0, Q_F, \pi \rangle$.*

- *Q – конечное множество состояний,*
- *$Q_F \subset Q$ – множество заключительных состояний,*
- *$q_0 \in Q$ – начальное состояние,*
- *$\pi : Q \times \Sigma^k \to Q \times (\Sigma \times (L, R, H))^k$ – программа.*

Определение. *Конфигурация МТ $\langle q, S, P \rangle$:*

- *$q \in Q$ – текущее состояние,*
- *S – состояние ленты,*
- *P – позиции головок.*

Определение. *Протокол – последовательность пройденных конфигураций.*

Определение. *Словарная функция $\phi_M : \Sigma^* \to \Sigma^*$.*

Определение. *Временная сложность в худшем случае:*

$$T_M(n) = max\{t_M(w) \ . \ \|w\| \leq n \wedge w \in D(\phi_M)\}$$

Определение. *Временная сложность в среднем:*

$$T_M(n) = \sum_{\|w\|=n} t_M(w) p(n, w)$$

где, $p(n, w)$ – вероятность появления w среди всех входов длины n.

Пример работы. Запись в табличном и графовом виде.

Задача: написать программу для одноленточной МТ в алфавите $\{0, 1, \#\}$, которая инвертирует двоичную строку. Например, входные данные #0011101010# следует преобразовать в #1100010101#.

Запись МТ в табличном виде:

$$START: \quad \begin{array}{l} \# \to \#R \ INVERT \\ 0 \to \#H \ ERROR \\ 1 \to \#H \ ERROR \end{array} \qquad INVERT: \quad \begin{array}{l} \# \to \#H \ STOP \\ 0 \to 1R \ INVERT \\ 1 \to 0R \ INVERT \end{array}$$

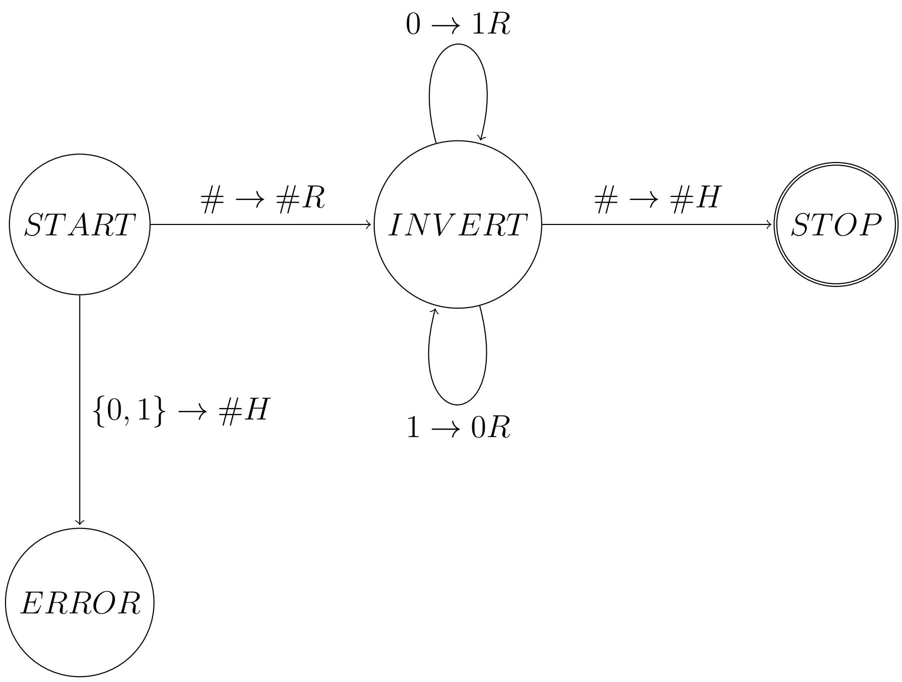


Рис. 1: Запись МТ в графовом виде

Инкремент

Задача: написать программу для одноленточной МТ в алфавите $\{0, 1, \#\}$, которая к двоичному числу, записанному "слева направо" (т.е. наименее значимый бит – Least Significant Bit, LSB – слева) без лидирующих нулей, прибавляет единицу. Примеры:

$$\#1\# \Rightarrow \#01\#$$

$$\#001\# \Rightarrow \#101\#$$

$$\#1001\# \Rightarrow \#0101\#$$

Оценить сложность в худшем. Оценить сложность в среднем.

Подсказки

Разбить множество всех входов фиксированной длины I_n на классы $I_n^0, I_n^1, ..., I_n^n$, где

$$I_n^i = \{w \in I.\ w = \#\ \underbrace{\underbrace{1...1}_{i \text{ единиц}}\ 0\ \underbrace{*...*}_{\text{произвольные символы}}}_{\text{всего } n \text{ цифр}}\ \#\}$$

Найти $T_i(w)$ для $w \in I_n^i$. Вычислить $|I_n^i|$.

$$S_1(t) = t + t^2 + ... + t^n \Rightarrow tS_1(t) - S_1(t) = t^{n+1} - t \Rightarrow ...$$
$$S_2(t) = t + 2t^2 + ... + nt^n \Rightarrow tS_2(t) - S_2(t) = ...$$

Упражнение. *Написать программу для одноленточной МТ в алфавите* $\{0, 1, \#\}$*, которая разворачивает данную строку, состоящую из нулей и единиц. Примеры:*

$$\#1\# \Rightarrow \#1\#$$

$$\#001\# \Rightarrow \#100\#$$

$$\#1101\# \Rightarrow \#1011\#$$

Упражнение. *Написать программу для одноленточной МТ в алфавите* $\{0, 1, \#\}$*, которая проверяет, является ли заданная строка правильной записью двоичного числа (LSB – слева).*

Прибавление константы

Задача: написать программу для одноленточной МТ в алфавите $\{0, 1, \#\}$, которая к двоичному числу, записанному слева направо без лидирующих нулей, прибавляет тройку. Обобщить на константу k.

Упражнение. *Написать программу для одноленточной МТ в алфавите* $\{0, 1, ..., m-1, \#\}$*, которая к* m*-ричному числу, записанному слева направо без лидирующих нулей, прибавляет константу* k*.*

Упражнение. *Перевести двоичное число* x *в* 16*-ричную систему счисления.* $\Sigma = \{0, 1, ..., 9, A, B, C, D, E, F, \#\}$

Дополнительный блок: строковые алгоритмы

Задача: написать программу для одноленточной МТ в алфавите $\{a, b, \#\}$, которая распознает язык $L = \{a^n b^n \mid n \geq 0\}$.

Оценить сложность в худшем. Ускорить программу в k раз (асимптотически), увеличив число состояний, но не изменяя алфавит.

Упражнение. *Задача: написать программу для одноленточной МТ в алфавите* $\{a, b, \#\}$*, которая проверяет, есть ли во входной строке подстрока abaababa (шаблон). Предложить процедуру построения оптимальной программы поиска для любого заранее известного шаблона.*

Упражнение. *Задача: написать программу для одноленточной МТ в алфавите* $\{a, b, \#\}$*, которая проверяет, есть ли в строке ababababbbbaaaabababaabaabbba (база данных) подстрока, записанная на входной ленте. Предложить процедуру построения оптимальной программы для любой заранее известной БД.*

Семинар #2. Многоленточная МТ.

Пример работы

Задача: инвертировать обе ленты 2-ленточной МТ в алфавите $\Sigma = \{0, 1, \#\}$.

Краткая запись программы: $X, Y \in \{0, 1, \#\}$, $x, y \in \{0, 1\}$.

$$
\begin{array}{lrrrll}
START: & & & & & \\
 & \# & \# \rightarrow & \#R & \#R & LOOP \\
 & X & Y \rightarrow & XH & YH & ERROR \\
LOOP: & & & & & \\
 & x & y \rightarrow & [(x+1)\ mod\ 2]R & [(y+1)\ mod\ 2]R & LOOP \\
 & x & \# \rightarrow & [(x+1)\ mod\ 2]R & \#H & LOOP \\
 & \# & y \rightarrow & \#H & [(y+1)\ mod\ 2]R & LOOP \\
 & \# & \# \rightarrow & \#H & \#H & STOP
\end{array}
$$

Сложение

Задача: написать программу для 3-ленточной МТ в алфавите $\{0, 1, \#\}$, которая складывает двоичные числа (LSB слева), записанные на первой и второй ленте, а результат записывает на третью ленту.

Оценить сложность в худшем.

Упражнение. *Решить аналогичную задачу для 2-ленточной МТ. Ответ записать на вторую ленту.*

Умножение

Задача: написать программу для 3-ленточной МТ в алфавите $\{0, 1, \#\}$, которая умножает двоичные числа, записанные на первой и второй ленте, а результат записывает на третью ленту.

Оценить, как зависит сложность алгоритма от:

- Формата записи числа (LSB слева vs. LSB справа);
- Порядка записи чисел (большее число на первой ленте vs. на второй ленте);
- Выбранного алгоритма (столбиком vs. крестьянский способ умножения).

Упражнение. *Написать программу для МТ и проверить ее в эмуляторе.*

Семинары #3-4. Моделирование

Определение. *Модель вычислений. Вычислитель. Допустимые входы.*

Определение. *Модель вычислений M_1 можно моделировать моделью вычислений M_2, если для любой машины A в M_1 можно построить машину B в M_2 такую, что:*

$$\phi_A = \phi_B \text{ при подходящих интерпретациях,}$$

$$T_B(n) = O(p(T_A(n))) \text{ для некоторого полинома } p.$$

Определение. *Неотрицательные функции $f_1(n)$, $f_2(n)$* ***полиномиально связаны*** *тогда и только тогда, когда $\exists p_1(n), p_2(n)$ – полиномы, такие что:*

$$\forall n \; f_1(n) \leq p_1(f_2(n)) \text{ и } f_2(n) \leq p_2(f_1(n))$$

Определение. *Как решать задачи на моделирование:*

- *Зафиксировать модель вычислений M_1 в языке Σ_1, модель вычислений M_2 в языке Σ_2.*
- *Указать функцию $code : \Sigma_1^* \to \Sigma_2^*$. Проанализировать сложность.*
- *Указать функцию $decode : \Sigma_2^* \to \Sigma_1^*$. Проанализировать сложность.*
- *Указать алгоритм, который по вычислителю $A \in M_1$ построит вычислитель $B \in M_2$.*
- *Обосновать $\phi_A = \phi_B$. При пошаговом моделировании удобно строить индуктивное доказательство. Важно: не забыть о граничных случаях – аварийное завершение работы и зацикливание вычислителя.*
- *Сравнить $T_A(n)$ и $T_B(n)$, $|A|$ и $|B|$.*

Моделирование б*о*льшего алфавита

Задача: промоделировать 1-ленточную МТ с алфавитом $\Sigma_1 = \{0, 1, \#\}$, на 1-ленточной МТ с алфавитом $\Sigma_2 = \{0, 1\}$. Рассматриваются МТ с лентой, бесконечной вправо.

Идея решения: заменить букву из Σ_1 на пару букв из Σ_2.

- Определим функцию кодирования *code*:

  $$w = a_1...a_n \to b_1^0 b_1^1 ... b_n^0 b_n^1$$

  где $a_i \in \Sigma_1$ переходит в $b_i^0 b_i^1, b_i^j \in \Sigma_2$ согласно правилу

  $$\# \to 00, 0 \to 01, 1 \to 11$$

- Определим функцию декодирования как функцию, обратную *code*, полагая, что 10 переходит в символ 0.

  Упражнение. *Доказать, что функцию декодирования можно доопределить произвольным образом.*

- Зафиксируем МТ $A = \langle \Sigma_1, Q^A, q_0^A, Q_F^A, \pi \rangle$. Пусть

$$q : x \to yR\ q' \text{ где } q, q' \in Q^A, x, y \in \Sigma_1$$

$$code(x) = x_1 x_0, code(y) = y_1 y_0, \text{ где } x_1, x_0, y_1, y_0 \in \Sigma_2$$

  В таком случае МТ $B = \langle \Sigma_2, Q^B, q_0^B, Q_F^B, \pi \rangle$ должна содержать следующие команды:

$$q : x_1 \to x_1 R\ q^{x_1 \hat{?}}$$

$$q^{x_1 \hat{?}} : x_0 \to y_0 L\ q^{\hat{x_1} y_0}$$

$$q^{\hat{x_1} y_0} : x_1 \to y_1 R\ q^{y_1 \hat{y_0}}$$

$$q^{y_1 \hat{y_0}} : y_0 \to y_0 R\ q'$$

  Упражнение. *Указать, как трансформируются команды*

$$q : x \to yL\ q'$$

$$q : x \to yH\ q'$$

- Предположим, что на ленте МТ A записано слово w, текущее состояние q, указатель находится на позиции k и будет выполнена команда перехода t, изменяющая слово на w', сдвигающая указатель на позицию k' и переводящая МТ в состояние q'. Предположим, что МТ B была получена из A по алгоритму, указанному выше. Пусть на ленте B записано слово $v = code(w)$, текущее состояние q, указатель находится на позиции $n = 2 * k$ и будут выполнены команды перехода t_1, t_2, t_3, t_4, изменяющие слово на v', сдвигающие указатель на позицию n' и переводящие МТ в состояние $\hat{q}$.

  Упражнение. *Доказать, что* $decode(v') = w', n' = 2 * k', \hat{q} = q'$. *Рассмотреть случай* $k' = -1$.

  Упражнение. *Доказать, что* $\phi_A = \phi_B$. *Рассмотреть случай зацикливания* A.

- По построению

$$T_B(w) \leq 4T_A(w)$$

$$|Q_B| \leq 4|Q_A|$$

Упражнение. *Решить обратную задачу: промоделировать 1-ленточную МТ с алфавитом* $\Sigma_1 = \{0, 1\}$, *на 1-ленточной МТ с алфавитом* $\Sigma_2 = \{0, 1, 2, \#\}$. *Можно ли ускорить произвольную программу?*

Задачи для моделирования

- Однонапраленная vs. двунаправленная лента.
- 1 лента vs. k лент.
- Σ_1 vs. Σ_2 для $|\Sigma_1| = n, |\Sigma_2| = m, n > m$.
- Один указатель на текущий символ vs. k указателей.

Семинар #5. Конечные автоматы.

Определение. *Детерминированный конечный автомат (Deterministic Finite State Machine) / Недетерминированный конечный автомат (Nondeterministic Finite Automaton) – (X, Q, q_0, Q_F, π):*

- $X = \{a_1, ..., a_n\}$ *– конечный алфавит,*
- Q *– конечное множество состояний,*
- $q_0 \in Q$ *– начальное состояние,*
- $Q_F \in Q$ *– множество заключительных состояний,*
- $\pi : Q \times X \to Q \;/\; \pi : Q \times X \to 2^Q$ *– функция перехода.*

Определение. *Регулярное выражение над* $X = \{a_1, ..., a_k\}$*:*

- $\emptyset, \epsilon, a_i$ *– регулярные выражения;*
- *если* α, β *– регулярные выражения, то*
  - $\alpha\beta$ *– конкатенация,*
  - $\alpha|\beta$ *– альтернатива,*
  - $(\alpha)^*$ *– итерация;*
- *других нет.*

Определение. *Основные теоремы:*

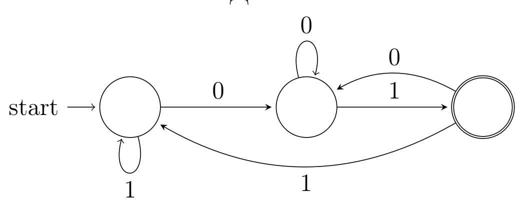
- *Любой язык, распознаваемый НКА, может быть распознан некоторым ДКА.*
- *Любой язык, распознаваемый ДКА, соответствует языку, задаваемому некоторым регулярным выражением.*
- *Любой язык, задаваемый регулярным выражением, может быть распознан некоторым НКА.*

Пример

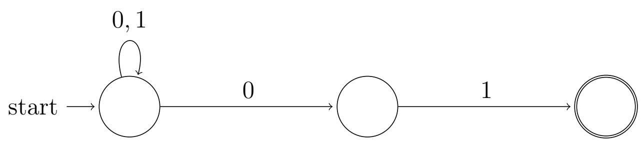
Язык $L = \{\text{слово в алфавите } \Sigma = \{0, 1\}, \text{ которое оканчивается на } 01\}$.

Регулярное выражение - $(0|1)^*01$.

ДКА



НКА



Табличный метод проверки на эквивалентность ДКА

См. лекции.

Минимизация ДКА методом грубейшего разбиения

[А.Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979]

Страницы 181 - 187 (пункт 4.13 “Разбиение”).

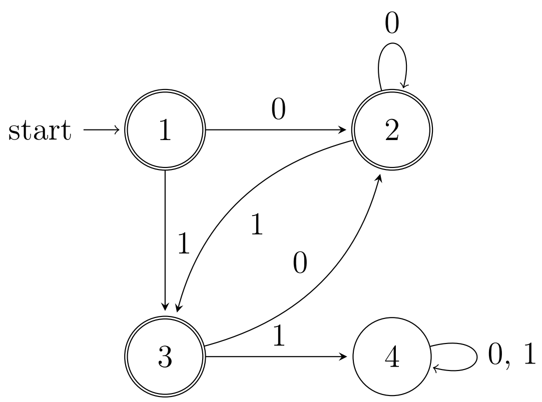
Проверка эквивалентности ДКА алгоритмом Объединить-Найти (Union-Find)

[А.Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979]

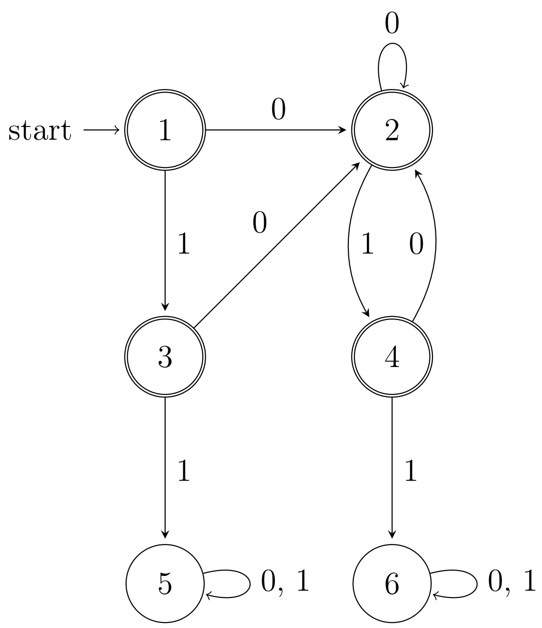
Страницы 165 - 168 (приложение 3 “Эквивалентность конечных автоматов”).

Автоматы для разбора на семинаре

I



II



Семинар #6. RAM машина.

Команда	Аргумент	Символьная запись	Команда	Аргумент	Символьная запись
0	0	Halt 0	13	13	Mult 13
1	1	Load =1	14	14	Mult *14
2	2	Load 2	15	15	Div =15
3	3	Load *3	16	16	Div 16
4	4	Store 4	17	17	Div *17
5	5	Store *5	18	18	Read 18
6	6	Add =6	19	19	Read *19
7	7	Add 7	20	20	Write =20
8	8	Add *8	21	21	Write 21
9	9	Sub =9	22	22	Write *22
10	10	Sub 10	23	23	Jump 23
11	11	Sub *11	24	24	JZero 24
12	12	Mult =12	25	25	JGTZ 25

Определение. *Словарная функция.*

Определение. *Равномерный весовой критерий.*

Определение. *Логарифмический весовой критерий.*

Пример программы

```
          Read 1                ; R1 = N, R2 = 0, R0 = 0

LOOP:     Load 2                ; R0 = R2
          Mult 0                ; R0 = R0 * R0
          Sub 1                 ; R0 = R0 - R1
          JZero FINISHED        ; if R0 == 0 goto FINISHED
          JGTZ FIN_SUB          ; if R0 > 0  goto FIN_SUB
          Load 2                ; R0 = R2
          Add =1                ; R0 = R0 + 1
          Store 2               ; R2 = R0
          JUMP LOOP             ; goto LOOP

FIN_SUB:  Load 2
          Sub =1
          Store 2               ; R2 = R2 - 1

FINISHED: Load =2               ; R0 = 2
          Write *0              ; print (R_(R_0)) ~ print(R_2) ~ print(R2)
          Halt 0                ; stop
```

Алгоритм Евклида: пошаговое кодирование

Псевдокод

```
a <- read()
b <- read()

while (a != b) {
  if (a > b) {



    a = a - b

  } else {

    b = b - a

  }

}


write(a)
```

Циклы -> метки

```
a <- read()
b <- read()
LOOP:
  if (a == b) goto END
  if (a > b) {



    a = a - b

  } else {

    b = b - a

  }
  goto LOOP

END:

  write(a)
```

Условия -> метки

```
a <- read()
b <- read()
LOOP:
  if (a == b) goto END
  if (a > b) goto SUB_A
  goto SUB_B

SUB_A:
  a = a - b
  goto AFTER_IF:

SUB_B:
  b = b - a
  goto AFTER_IF

AFTER_IF:
  goto LOOP

END:
  write(a)
```

Удаление лишних переходов

```
a <- read()
b <- read()
LOOP:

  if (a == b) goto END
  if (a > b) goto SUB_A

  b = b - a

  goto LOOP

SUB_A:
  a = a - b
  goto LOOP

END:
  write(a)
```

Распределение регистров

```
R1 <- read()
R2 <- read()
LOOP:
  R0 = R1 - R2
  if (R0 == 0) goto END
  if (R0 > 0) goto SUB_A

  R2 = R2 - R1

  goto LOOP

SUB_A:
  R1 = R1 - R2
  goto LOOP

END:
  write(R1)
```

Оптимизация общих выражений

```
R1 <- read()
R2 <- read()
LOOP:
  R0 = R1 - R2
  if (R0 == 0) goto END
  if (R0 > 0) goto SUB_A

  R0 = -R0
  R2 = R0
  goto LOOP

SUB_A:
  R1 = R0
  goto LOOP

END:
  write(R1)
```

RAM ассемблер			Машинный код	
1.		Read 1	18	1
2.		Read 2	18	2
3.	LOOP:	Load 1	2	1
4.		Sub 2	10	2
5.		JZero END	24	12
6.		JGTZ SUB_A	25	10
7.		MULT =-1	12	-1
8.		Store 2	4	2
9.		Jump LOOP	23	3
10.	SUB_A:	Store 1	4	1
11.		Jump LOOP	23	3
12.	END:	Write 1	21	1
13.		Halt 0	0	0

Упражнение. *Провести анализ сложности алгоритма при логарифмическом весовом критерии.*

Косвенная адресация

Задача: написать программу для RAM-машины, которая вычисляет скалярное произведение двух непустых векторов.

Вход: $n \geq 1$, $a_1, ..., a_n, b_1, ..., b_n$

Выход: $\sum_{i=1}^{n} a_i b_i$

Проанализировать сложность программы, используя равномерный и логарифмический весовой критерий.

Реализация стека

Задача: написать программу для RAM-машины, которая выводит вектор целых чисел в обратном порядке. Признак конца вектора – число 0.

Вход: $a_1, ..., a_n, 0$, где $a_i \neq 0$.

Выход: $a_n, a_{n-1}, ..., a_1$

Использовать псевдопроцедуры $push, pop$.

Семинар #7. RAM машина как ассемблер ЯВУ: вызов функций

Функция высшего порядка map

Задача: написать программу для RAM-машины, которая применяет некоторую целочисленную функцию f к массиву чисел.

Вход: $n > 0, a_1, ..., a_n$.

Выход: $f(a_1), ..., f(a_n)$.

Обобщить на функции от нескольких параметров:

Вход: $n > 0, 0 < k < 5, a_1^1, a_1^2, ..., a_1^k, ..., a_n^1, a_n^2, ..., a_n^k$.

Выход: $f(a_1^1, a_1^2, ..., a_1^k), ..., f(a_n^1, a_n^2, ..., a_n^k)$.

Соглашение о вызовах и адрес возврата

Переписать программу из предыдущего пункта, используя следующий контракт:

- В регистре R_1 передается номер инструкции, на которую следует перейти после выполнения f (ret-address).
- В регистрах $R_2, ..., R_5$ передаются параметры для f (param-passing regs). После выполнения f регистры должны содержать то же самое значение (non-volatile regs).
- Регистры $R_6, ..., R_{10}$ после выполнения f могут произвольным образом измениться (volatile regs).
- В регистре R_0 должен быть сохранен результат вычисления функции.

Реализовать "псевдоинструкцию" *call*.

Упражнение. *Реализовать транслятор "расширенного" RAM-ассемблера, использующего инструкцию call, в обычный ассемблер. Использовать любой язык программирования. Возможно ли таким образом реализовать рекурсивные вызовы?*

Дополнительный блок: арифметические выражения

Упражнение. *Дана строка в обратной польской записи, в которой встречаются только положительные целые числа и коды операций:*

$$+ \to -1$$

$$- \to -2$$

$$* \to -3$$

$$/ \to -4$$

Вычислить данное выражение либо зациклиться, если происходит недопустимая операция (деление на ноль).

Упражнение. *Усложнение предыдущей задачи: в строке встречаются переменные*

$$x \to -5$$

$$y \to -6$$

$$z \to -7$$

и задана таблица их значений в виде пар $\langle$ код переменной, значение переменной $\rangle$.

Семинары #8-9. RASP машина. Самомодифицирующийся код. JIT компиляция. Эмуляция косвенности. Загрузчик. Квайн.

Пример самомодифицирующейся программы

```
START: Load  =0
       Store  9
       Load  =0
       Store 10
       Jump  START
```

Переход на произвольную инструкцию

```
LABEL: Load x           ; R0 = x
       Store [LABEL + 5] ; --|
       JUMP Label        ; <-|  JUMP x
```

Упражнение. *Объяснить, как реализовать рекурсивный вызов процедур на RASP машине. Использовать концепцию стекового кадра и стека вызовов*[1]*.*

Упражнение. *Реализовать транслятор "расширенного" RASP-ассемблера, использующего инструкцию call, в обычный ассемблер. Поддержать рекурсивные вычисления, в том числе косвенную рекурсию. Использовать любой язык программирования.*

Генерация программы в памяти во время исполнения

Задача: написать *процедуру* для RASP-машины, которая принимает аргументом число $n > 0$ и печатает числа от 1 до n включительно.

Упражнение. *Оценить алгоритмическую сложность программы при равномерном весовом критерии.*

```
void printer(int n) {
  i = 0
LOOP:
  write (i)
  if (n == 0) goto END
  i = i + 1
  n = n - 1
  goto LOOP
END:
  return
}
```

Loop unrolling

Оценить алгоритмическую сложность программ при равномерном весовом критерии.

[1] https://en.wikipedia.org/wiki/Call_stack

```
void printer_2_naive(int n) {
  i = 0
LOOP:
  write (i)
  if (n == 0) goto END
  i = i + 1
  n = n - 1

  write (i)
  if (n == 0) goto END
  i = i + 1
  n = n - 1

  goto LOOP
END:
  return
}
```

```
void printer_2_preheader(int n) {
  i = 0

  mod = n % 2
  if (mod == 0) goto LOOP
  write (i)
  i = i + 1
  n = n - 1

LOOP:
  write (i)
  i = i + 1
  write (i)
  i = i + 1
  n = n - 2
  if (n == 0) goto END
  goto LOOP
END:
  return
}
```

Упражнение. *Написать RASP-программу, которая для заданного k генерирует программу $printer\_k\_preheader$. Оценить сложность программы генерации и сложность сгенерированной программы.*

Упражнение. *Проанализировать эффективность оптимизации "Раскрутка цикла" – найти формулы для оценки сложности при равномерном весовом критерии и для затрат памяти. Какие стратегии раскрутки кажутся Вам наиболее эффективными? Привести контрпримеры. Можно ли использовать информацию времени выполнения (a.k.a. execution profile), чтобы принять оптимальное решение?*

Загрузчик программ. Отличие от интерпретатора.

Задача: написать RASP-программу, которая считывает с входной ленты поток чисел длины N, являющийся допустимой RASP-программой, располагает "входную программу" в регистрах $R_1, R_2, ..., R_N$, а затем передает ей управление, выполнив $JUMP$ на регистр R_1.

Вопросы для самостоятельного разбора.

Итоговая контрольная работа содержит задачи на материал данного раздела.

Пример RASP программы, выводящей свой собственный код.

https://en.wikipedia.org/wiki/Quine_(computing)

Эмуляция косвенности (операций со звездочкой).

[А.Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979]

Страницы 26 - 31 (пункт 1.4 "Модель с хранимой программой").