Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
СибГУТИ

Отчёт по информатике
«Моделирование архитектуры фон Неймана на языке С»

Выполнил студент: Родионов П.А.
группы: ИВ-522
Вариант 9
Проверил ассистент кафедры ВС: Шевченко М.В.

Новосибирск 2025 г.

Цель работы

Цель работы — изучить принцип пошагового выполнения команд программой, работающей с памятью, и закрепить понимание того, как реализуется алгоритм обработки данных с помощью простейших управляющих структур.

Описание программы

Уровень реализации: «5».

Программа представляет собой **алгоритм работы с памятью**, в которой заранее записана последовательность команд.
Каждая команда задаёт определённое действие: загрузку данных, выполнение арифметической операции, вывод результата или завершение работы.

В процессе выполнения программа пошагово читает команды из памяти, выполняет их и переходит к следующей.
Для этого используется **счётчик команд(`pc`)**, который определяет текущую позицию в памяти, и **аккумулятор(`acc`)**, служащий для хранения промежуточных результатов вычислений.

Команда	Код	Описание
`STOP`	0	Останавливает выполнение
`PRINT`	1	Выводит текущее значение аккумулятора
`LOAD`	2	Загружает в аккумулятор следующее значение
`MUL`	3	Умножает аккумулятор на следующее значение
`JUMP`	4	Безусловный переход на адрес
`JZ`	5	Переход, если аккумулятор равен нулю

Организация памяти:

Массив `prog[]` содержит последовательность чисел, где каждый элемент — либо код операции, либо её аргумент.

Пошаговое выполнение (пример):

```
int prog[] = {
    2, 5, // load 5
    3, 3, // mult 3
    1,    // print acc
    4, 7, // jump 7
    2, 0, // load 0
    5, 11,// jz 11
    2, 2, // load 2
    5, 1, // jz 1
    3, 4, // mul 4
    1,    // print
    0     // stop
};
```

1. `LOAD 5` → `acc = 5`
2. `MUL 3` → `acc = 15`
3. `PRINT` → выводит 15
4. `JUMP 7` → переход на позицию 7
5. `LOAD 2` → `acc = 2`
6. `JZ 1` → так как `acc != 0`, переход не выполняется
7. `MUL 4` → `acc = 8`
8. `PRINT` → выводит 8
9. `STOP` → завершение

Таким образом, программа демонстрирует последовательное выполнение и условные переходы в рамках единой памяти.

Листинг программы

```
#include <stdio.h>
void pt(int prog[], int cmd,int acc,int pc);

int main() {

    int prog[] = {
        2, 5, // load 5
        3, 3, // mult 3
        1,    // print acc
        4, 7, // jump 7
        2, 0, // load 0
        5, 11,// jz 11
        2, 2, // load 2
        5, 1, // jz 1
        3, 4, // mul 4
        1,    // print
        0     // stop
    };

    int cmd, pc, acc;
    cmd = pc = acc = 0;

    while(pc != -1){
        cmd = prog[pc];
        switch(cmd){
            case 0: //Проверка на команду остановки
                printf("STOP\t ACC = %d\t PC = %d",acc,pc);
                pc = -1;
                break;

            case 1: // Комманда 1 - вывод регистра
                pt(prog,cmd,acc,pc);
                pc++;
                break;

            case 2: // Комманда 2 - Присвоить регистру число
                acc = prog[pc+1];//   из след ячейки памяти
                pt(prog,cmd,acc,pc);
```

```
            pc+=2;
            break;

        case 3: // Комманда 3 - Умножить регистр на число
            acc *= prog[pc+1];//  в след ячейке памяти
            pt(prog,cmd,acc,pc);
            pc +=2;
            break;

        case 4: // Комманда 4 - Безусловный переход на указанную
            pt(prog,cmd,acc,pc); //   ячейку памяти
            pc = prog[pc+1];
            break;

        case 5: // Комманда 5 - Переход на указанную ячейку
            pt(prog,cmd,acc,pc);// памяти если регистр равен 0
            if(acc == 0){pc = prog[pc+1];}
            else(pc+=2);
            break;

        default: printf("err\n");return 1; // Защита от неправильной комманды
                                          //    и бесконечного зацикливания
      }
   }
   return 0;
}

   // Блок отвечающий за вывод в терминал
   // значиний в момент выполнения комманд
void pt(int prog[],int cmd,int acc,int pc){
   printf("\n");
   printf("cmd: %d\n",cmd);
   switch(cmd){
      case 1:
         printf("\tPrint acc\n\t\tacc = %d\n",acc);
         printf("\t\tcurr pc = %d",pc);break;
      case 2:
         printf("\tLoad acc\n\t\tacc = %d\n",acc);
         printf("\t\tcurr pc = %d",pc);break;
      case 3:
```

```
            printf("\tMult acc\n\t\tacc = %d\n",acc);
            printf("\t\tcurr pc = %d",pc);break;
        case 4:
            printf("\tJump to\n\t\tto = %d\n",prog[pc+1]);
            printf("\t\tcurr pc = %d",pc);break;
        case 5:
            if(acc==0){
                printf("\tJump to if acc zero\n\t\tto = %d\n",prog[pc+1]);
            }else(printf("\tJump to if acc zero\n\t\tacc != 0\n"));
            printf("\t\tcurr pc = %d",pc);
            break;
    }
    printf("\n");
}
rintf("\tJump to if acc zero\n\t\tto = %d\n",prog[pc+1]);
            }else(printf("\tJump to if acc zero\n\t\tacc != 0\n"));
            printf("\t\tcurr pc = %d",pc);
            break;
    }
    printf("\n");
}
```

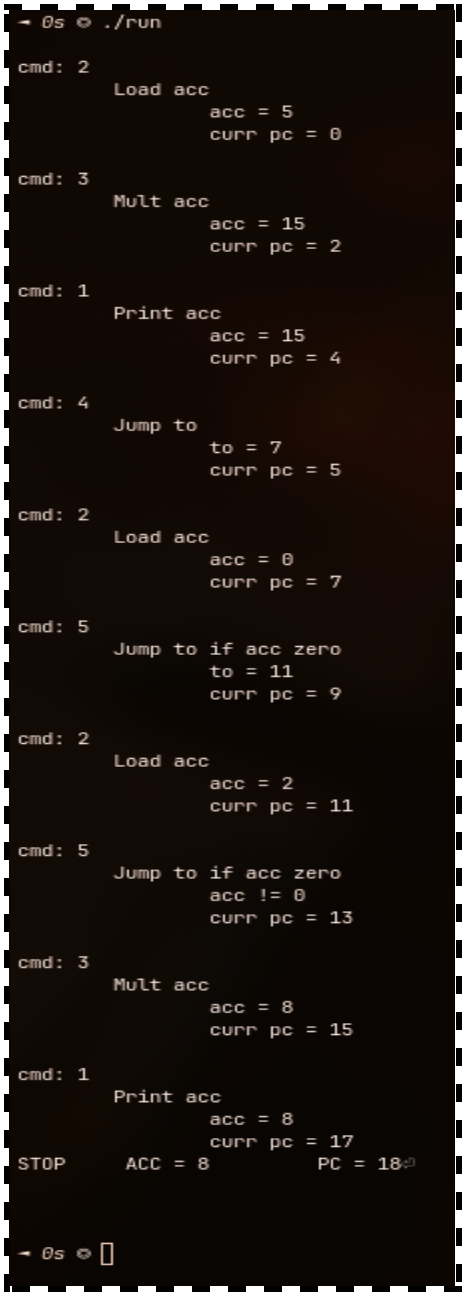
Результаты выполнения

Программа успешно выполняет последовательность команд, записанных в массиве `prog[]`.
На экране отображается текущая выполняемая команда, значение аккумулятора и позиция счётчика команд.

Во время выполнения видно, как программа:

- загружает числа в аккумулятор,
- выполняет арифметические операции,
- осуществляет переходы по адресам (в том числе условные),
- завершает работу по команде `STOP`.

Входные данные в данной программе **не требуются**, так как все значения заранее записаны в память (`prog[]`).
Переходы `JUMP` и `JZ` демонстрируют работу с адресами — **безусловный** и **условный** переход, что реализует простейшее ветвление программы.



Вывод

В ходе выполнения работы была реализована программа, демонстрирующая **алгоритм последовательного выполнения команд**, записанных в памяти. Реализация показала, как процессорная логика может быть смоделирована с помощью простых управляющих конструкций на языке C.

В работе были продемонстрированы ключевые принципы **архитектуры фон Неймана**:

- совместное хранение данных и инструкций в одной памяти,
- последовательное выполнение команд,
- использование счётчика команд и регистра (аккумулятора).

Трудности, возникшие при реализации программы:

Обработка команд с разным количеством параметров потребовала точного управления счётчиком команд, чтобы правильно переходить к следующей инструкции. Также важно было контролировать корректность данных в массиве программы, чтобы избежать выхода за границы памяти и ошибок выполнения. Для удобства отладки была реализована функция вывода состояния программы, которая помогла быстро выявлять и устранять проблемы.