Команды ассемблера

Курсовой проект "Эмулятор PDP-11"
Занятие 3 и 4

Что уже сделано

- Память RAM 64Кб, 16-битная
- функции чтения и записи байта и слова из/в память
- загрузка программы в эту память

Регистры — очень быстрая память

- Память большая, но медленная.
- Быстрая память — дороже, поэтому ее мало.
- Регистры
  - 8 штук
  - 16 бит
  - имена R0 .. R7
- В эмуляторе:

```
word reg [8];     // массив регистров, reg[ i ]
```

Первая программа — сумма 2 чисел

```
.= 1000;          ; разместим код начиная с адреса 1000
mov  #2, R0       ; R0 = 2
mov  #3, R1       ; R1 = 3
add R0, R1        ; R1 = R0 + R1
halt              ; завершение программы
```

```
0200 000c
c0
15
02
00
c1
15
03
00
01
60
00
00
```

- **Все числа восьмеричные**
- С **;** начинаются комментарии
- **.=** - псевдо команда ассемблера,
  где размещать код, который написан дальше
- По умлочанию *эталонный* эмулятор начинает выполнять программу с адреса **1000**. Наш тоже.

Рекомендуемая организация директорий

- pdp11
  - repo — директория вашего проекта
    - pdp.c, pdp.h .git .gitignore
  - pdp11em — директория с эталонным эмулятором
  - test — директория с программами на ассемблере
    - 1_sum
      - sum.txt — программа на ассемблере
      - sum.o — что грузим в эмулятор
      - sum.l — листинг (для отладки)

Установка эталонного эмулятора

- Скачать архив
  http://acm.mipt.ru/twiki/bin/view/Cintro/PDPrun
-

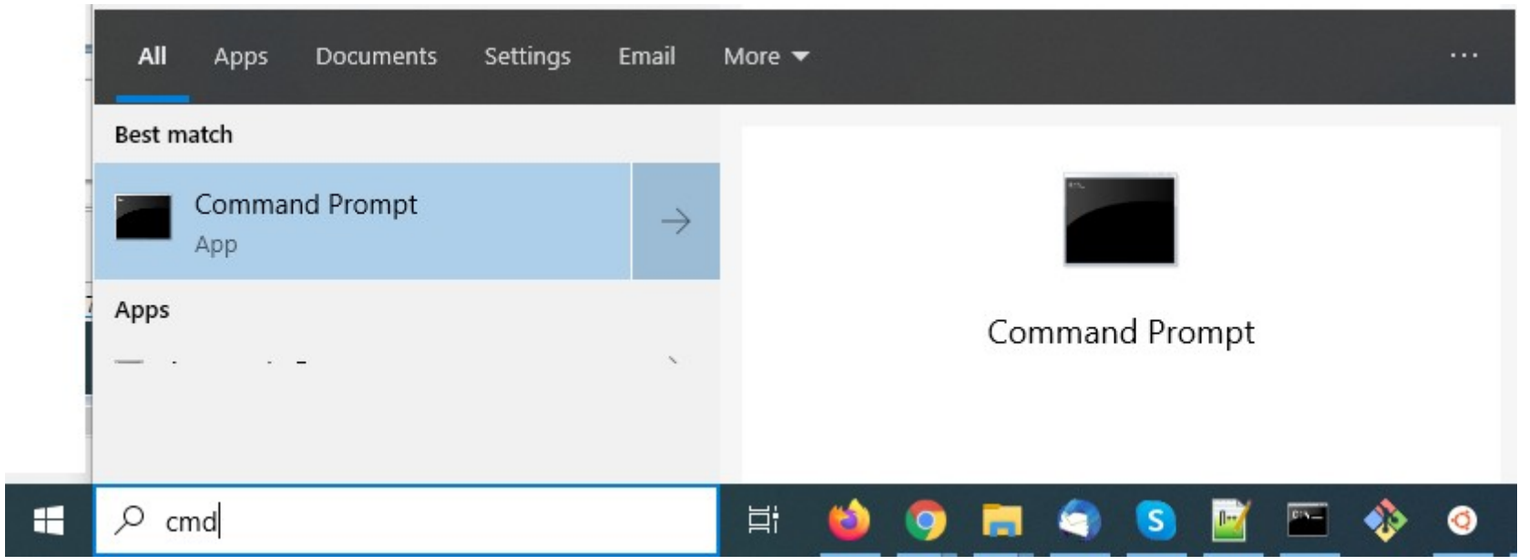
ОС	Файл
Linux	pdp11em-linux.zip
Mac	mac_exe.zip
Windows	pdp11em-win32.rar

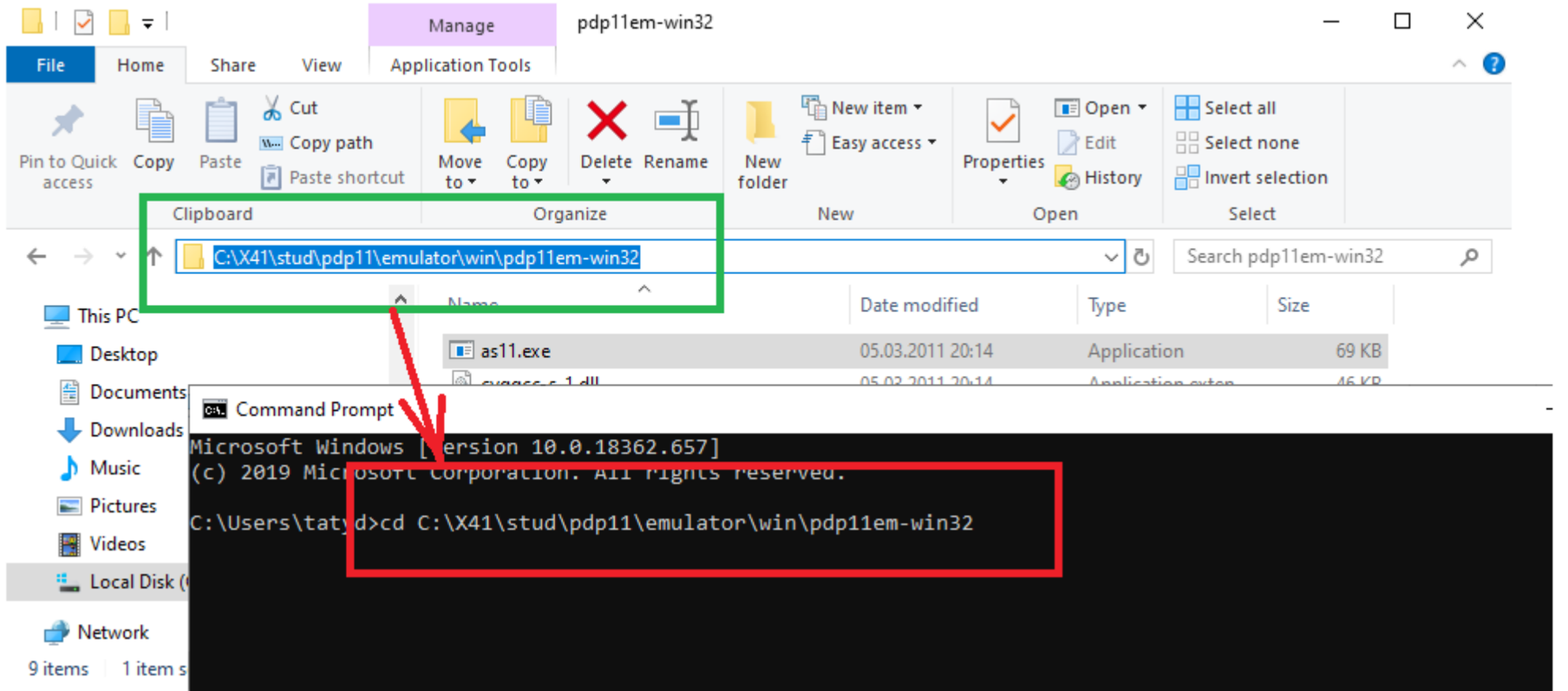
Linux — установка (Mac - аналогично)

- распаковать архив в ту директорию, где будем работать
- в терминале перейти в эту директорию
- добавить права на исполнение файлам
  chmod +x as11
  chmod +x pdp11
  Запустить
  ./as.11
- Если не корректно запускается, то
  chmod +x as11.static
  chmod +x pdp11.static
  ./as11.static (и далее запускать только со static)

Windows — установка

- Распаковать архив в директорию
- Запустить Command Prompt (cmd.exe, командную строку):
  Win + R
  cmd
- Скопировать из Проводника путь к архиву
- В командной строке набрать
  cd /D "путь к директории с файлами в кавычках"
  - /D нужен только если переходить на другой диск
  - кавычки нужны, если в пути есть пробелы
  - можно набрать путь вручную, а не копировать


All
Apps
Documents
Settings
Email
More
Best match
Command Prompt
App
Apps
Command Prompt
cmd


Manage
pdp11em-win32
File
Home
Share
View
Application Tools
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Move to
Copy to
Delete
Rename
New folder
New item
Easy access
Properties
Open
Edit
History
Select all
Select none
Invert selection
Clipboard
Organize
New
Open
Select
C:\X41\stud\pdp11\emulator\win\pdp11em-win32
Search pdp11em-win32
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk
Network
Name
Date modified
Type
Size
as11.exe
05.03.2011 20:14
Application
69 KB
9 items
Command Prompt
Microsoft Windows [Version 10.0.18362.657]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\tatyd>cd C:\X41\stud\pdp11\emulator\win\pdp11em-win32

Запуск

- Компилятор запускаем

  as11.exe — в Windows
  as11 или as11.static (что работает) — Linux, Mac

- Эмулятор запускаем
  - pdp11.exe — в Windows
    pdp11 или pdp11.static — Linux, Mac
- Дальше примеры будут для as11 и pdp11
  Подумайте, как нужно изменить примеры
  для вашего компьютера.
- **Windows все dll файлы должны быть в той же директории, что exe файлы.**

Компиляция

- Компиляция
  as11 -o sum.o **-l** sum.l **sum.txt**
- Usage: as11 [-l listing-file] [-o output-file] assembly-file
- sum.txt — файл с программой на ассемблере
- sum.o — получим, будем грузить в эмулятор
- sum.l — вспомогательный, удобно отлаживать
- опции -o (output) и -l (listing)
- В Windows лучше указывать имена
  sum.o.txt
  sum.l.txt - чтобы читать в редакторе

Файлы (octal & hex)

- **sum.txt**

```
.= 1000;          ; разместим код начиная с адреса 1000
mov  #2, R0       ; R0 = 2
mov  #3, R1       ; R1 = 3
add R0, R1        ; R1 = R0 + R1
halt              ; завершение программы
```

- **sum.l** **адрес** **машинный_код** ассемблер

```
000000:       .= 1000
001000:       mov #2, R0  ; R0 = 2
              012700
              000002
001004:       mov #3, R1  ; R1 = 3
              012701
```

sum.o

```
0200 000c
c0
15
02
00
c1
15
03
00
01
60
00
00
```

Запуск эталонного эмулятора

- Использование с ключом **-t** (трассировка)
  ./pdp11 -t sum.o
- Напечатает:

```
---------- running ----------
001000: mov     #000002,r0          [001002]=000002
001004: mov     #000003,r1          [001006]=000003
001010: add     r0,r1               R0=000002 R1=000005
001012: halt
---------- halted ----------
r0=000002 r2=000000 r4=000000 sp=000000
r1=000005 r3=000000 r5=000000 pc=001014
psw=000000: cm=k pm=k pri=0       [4]
```

Задача урока

- Наш эмулятор должен печатать похоже на эталонный эмулятор и листинг.

  адрес,
  машинный код,
  имя команды

Почему адреса только четные?
Почему после 6 идет 10?
Почему 012700 это mov?

- Напечатает:

001000 012700: **mov**
001002 000002: unknown
001004 012701: **mov**
001006 000003: unknown
001010 060001: **add**
001002 000000: **halt**

Кодирование системы команд

Простейшая система команд

- Кодируем игрушечный автомобиль.
  Память 1 битовая.
  Команды могут быть или 0, или 1.

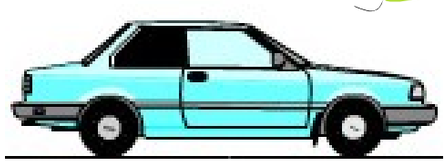
- Пример системы команд:
  0 — стоп
  1 — ехать вперед

Команда	Код	Действие
move	1	
stop	0	

1
1
1
0







Повороты

- **Команды без операндов**
  - **01 поворот направо**
  - **10 поворот налево**
- **Команда поворот 1 операнд**
  - **0 право**
  - **1 лево**
  - **1** код операции opcode
  - кодируем повороты: **10** и **11**

Команда	Код	Действие
move	11	
stop	00	
turnR	01	
turnL	10	

Команда	Код	Действие
move	01	
stop	00	
turn R	10	
turn L	11	

Больше поворотов, 3 бита

- Расширим список возможных поворотов XX:

  00: 45°

  01: 90°

  10: - 45°

  11: - 90°

Команда	Код
move	001
stop	000
turn ***a***	1***XX***

- **110** означает "**поворот** на **- 45** градусов"
- Неиспользуемые коды

  010
  011

- Можно добавить включить/выключить фары (сирена)

Разная длина операндов

- D - вкл/выкл

  0: включить (on)
  1: выключить (off)

- XX — список поворотов

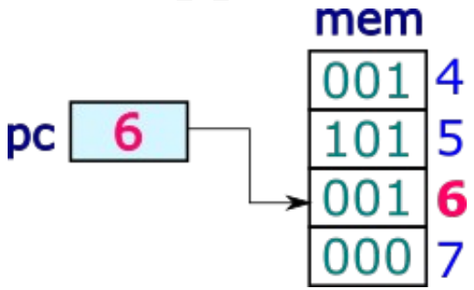
  00: 45°
  01: 90°
  10: - 45°
  11: - 90°

Команда	Код
move *on/off*	00**D**
фары *on/off*	01**D**
turn ***a***	1***XX***

- Неиспользуемых кодов нет
- opcode (код операции) и операнды
- иногда opcode называют расшифровку 00D

Где искать очередную команду программы

- Команды загружены в память машины mem[ ]
- Команды выполняются одна за другой
- **PC** (programm counter) — память,
  в которой лежит **адрес** следующей команды
  (в ассемблере x86 это ip — instruction pointer)



- цикл работы программы автомобиля:
  - **word w** = **w_read** (**pc**);
    pc += 1; // подумать, так ли для pdp-11
    разобрать слово **w** на команду и аргументы
    выполнить команду
- В pdp-11 **pc** это **R7**

Какая это команда

- прочли очередную команду с аргументами в w
  w = 011
- какая это команда?
  - обнулить аргументы
  - сравнить с кодом операции

Команда	Код	Маска
move *on/off*	00**D**	11**0**
фары *on/off*	01**D**	11**0**
turn ***a***	1***XX***	1**00**

- if ((w & 110) == 000)
  значит команда move
  else if ((w & 110) == 010)
  значит команда фары
  else if ((w & 100) == 100)
  значит команда turn

Команды PDP-11 mov, add, halt, inc

- Список команд http://acm.mipt.ru/twiki/bin/view/Cintro/PDPCommandList
- SS и DD — 6-битные аргументы
- B — 0 или 1 — команда оперирует словами или байтами

Mnemonic	Opcode	NZVC	Description	Notes
ADD s, d	**06**SSDD	****	Add	d = s + d
INCb s, d	**B052**DD	****	Increment	d = d + 1
MOVb s,d	**B1**SSDD	****	Move	d = s
HALT	**0000000**	---	Halt	
MOV s,d	**01**SSDD	****	Move word	d = s
MOVB s,d	**11**SSDD	****	Move byte	d = s

Задача урока

- В цикле от 1000 адреса до команды HALT
  - читать слово
  - печатать адрес и слово по формату %06o
  - если это команда add, mov или halt, печатать мнемонику (имя) команды
  - для прочих данных печатать unknown
  - после обнаружения команды halt
    - напечатать THE END
    - закончить выполнение программы

Цикл выполнения программы

- word reg[8]; // массив регистров R0..R7
  #define pc reg[7]
- // выполнение программы после загрузки
  void run( ) {
  // как в эталонном эмуляторе начинаем
  // выполнять программу с адреса 1000
  pc = 01000;
  while (1) {
  word w = w_read(pc);
  trace("%06o %06o : ", pc, w);
  pc += 2;
  // разбор и выполнение прочитанной команды

Этапы реализации нахождения команд

- ```
  if (w == 0) {                    // это halt ?
      trace("halt ");
      do_halt();   // написать самим
  } else {
      trace("unknown ");
  }
  trace("\n");
  ```
- Как проверить, что это mov (вспоминаем bit_1, bit_2..)
  B1SSDD — код операции по таблице
  01SSDD — **mov**   (надо определить)
  11SSDD — movb (не сейчас)
- SS и DD — по 6 бит, слово 16 бит
  ```

Этапы реализации нахождения команд

- if (w == 0) { // это halt ?
  trace("halt ");
  do_halt(); // написать самим
  } else {
  trace("unknown ");
  }
  trace("\n");
- else if ((w & 0170000) == 0010000) { // B1SSDD
  trace("mov ");
  do_mov();
  }
- дописать проверку на add

Сверните в цикл (как в задаче bit_4)

- Общие части:

```
//              mask                opcode
 if ( (w & 0170000) == 0010000) {
    trace("mov ");                  // name
    do_move();                      // do_func()
}
```

- struct Command {

```
    word mask;                  // 0170000
    word opcode;                // 0010000
    char * name;                // "mov"
    void (*do_func)(void);      // void do_mov() { .. }
} cmd [ ] = { ... };
```

Кто хочет хранить сдвиг,
смотрит на опкод команды BNE

Дополнительный бонус

- Разделите проект на несколько *.c файлов
  - работа с памятью: b_read, b_write,.. load_file
  - run (сюда будем писать функции работы с аргументами)
  - do_halt, do_mov, do_add, массив описания команд
- Подумайте, как в этом случае описать цикл по всему массиву cmd (условие окончания цикла)
  - специальная "команда" — признак конца массива (как \0 в строке)
  - unknown command

Следующее занятие

- Разбор аргументов SS и DD
- Тест сложения 2+3 работает