CSE4100 시스템 프로그래밍
개별 프로젝트 #1
담당교수 : 서강대학교 컴퓨터공학과 김지환

1. 프로젝트 문제 및 목표

이 프로그램은 앞으로 구현하게 될 SIC/XE 머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더들을 실행하게 될 셸(shell)과 컴파일을 통해서 만들어진 object 코드가 적재 되고 실행될 **메모리공간**과 mnemonic (ADD, COMP, FLOAT, etc …)을 opcode 값으로 변환하 는 **OPCODE 테이블**과 **관련 명령어**들을 구현하는 프로그램입니다.

2. 요구사항

2.1 프로젝트 목표 설정

- 이 프로그램을 실행시키면 아래와 같이 unix shell 과 유사한 입력 프롬프트상태가 된다.

  sicsim>

- 이 상태에서 아래에 있는 명령어들을 입력할 때, 그에 해당되는 기능을 수행하여야 한다.
- 구현해야 할 사항들 (다음 페이지에 보다 자세한 설명이 나옵니다.)
  1. 셸 (sicsim>)
  2. 셸 관련 명령어들 (help, dir, quit, history)
  3. 메모리공간 (1MB 의 메모리를 할당해서 사용)
  4. 메모리공간 관련 명령어들 (dump, edit, fill, reset)
  5. opcode 테이블 (HashTable 로 만들어야 함)
  6. opcode 관련 명령어들 (opcode, opcodelist)

2.2 합성

본 프로젝트는 어셈블러, 링크, 로더들을 실행하게 될 **셸(shell)**과 컴파일을 통해서 만들어진 object 코드가 적재되고 실행될 **메모리공간**과 mnemonic(ADD, COMP, FLOAT, etc…) 을 opcode 값으로 변환하는 **OPCODE 테이블**과 **관련 명령어**들을 구현하는 것이다. 따라서 완성된 프로그램이 수행해야 하는 각각의 기능을 구현하기 위해 필요한 자료구조와 알 고리즘을 구상한 후, 전체적인 프로그램을 설계한다.

2.3 제작 / 2.4 시험 / 2.5 평가

1) Shell 관련 명령어

① sicsim> **help**

- 아래와 같이 Shell 에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력해준다.

~~h[elp]~~
~~d[ir]~~
~~q[uit]~~
~~hi[story]~~
du[mp] [start, end]
e[dit] address, value
f[ill] start, end, value
reset
opcode mnemonic
~~opcodelist~~ → Hash function

② sicsim> **d[ir]**

- 현재 디렉터리에 있는 파일들을 출력한다.
- system call 을 이용하여 구현하지 않도록 해야 합니다.
  즉, system(..) 이나 exec(…) 와 같은 함수는 사용을 금합니다.
- dirent.h, sys/stat.h 를 참조합니다.
- .과 ..는 포함되어도, 되어있지 않아도 관계없습니다

ex) sicsim> dir

Desktop/	Work/	dead.letter	mail/
Mail/	a.out*	lngabi/	

Dir 의 결과를 출력할 때 실행 파일은 파일 이름 옆에 '*' 표시를, 디렉터리는 '/' 표시를 해야 합니다.

③ sicsim> **q[uit]**

- sicsim 을 종료한다.

④ sicsim> hi[story]

- 아래와 같이 현재까지 사용한 명령어들을 순서대로 번호와 함께 보여준다. 가장 최근 사용한 명령어가 리스트의 하단에 오도록 한다.

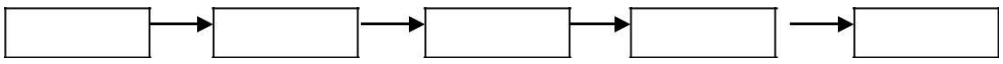
ex) sicsim> history

```
1     dump
2     dump 14, 37
3     e 14, E3
      …중간 생략…
908   reset
909   d
910   history
```

수행한 모든 명령어는 History 에 계속 추가되어야 합니다. 만약 history 가 비어 있다면 아무것도 출력하지 않고 다시 입력 프롬프트로 돌아옵니다.
(일반적으로 history 를 친다면 그것도 명령어로 가정하므로, 빈 경우는 없습니다.)

잘못된 명령어를 입력하는 경우, 수행하지 않고 history 에도 남기지 않습니다

History 명령은 아래의 그림과 같이 **반드시 linked list 의 형태**로 구현이 되어야 합니다.
(자료구조 책에 나오니 참고 바람, linked list 가 아닐 경우 감점대상)



2) 메모리 관련 명령어

- 앞으로 구현하게 될 assembler, linker& loader 를 통해서 만들어진 object 파일을 올려서 실행하게 될 Shell 내의 메모리 공간에 관련된 명령어들로써 이 Shell 에서는 사이즈가 1MByte(16 X 65536)인 가상의 메모리 공간을 구현하여야 한다.

- **아래에 나와있는 숫자는 모두 16 진수입니다.**

① sicsim> du[mp] [start, end]

- 할당되어 있는 메모리의 내용을 아래와 같은 형식으로 출력시켜 준다.

```
00000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; ......... 01
00020 32 33 34 35 36 37 38 39 20 20 20 20 20 2D 3D 2B ; 23456789 -=+
00030 5B 5D 7B 7D 20 20 20 20 20 20 20 54 68 69 73 20 ; []{} This
00040 69 73 20 73 61 6D 70 6C 65 20 50 72 6F 67 72 61 ; is sample Progra
00050 6D 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E ; m..............
00060 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E ; ................
00070 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E ; ................
```

- 가장 왼쪽 칼럼은 출력하는 메모리 주소를 나타낸다. 주소는 5 자리로 고정하고 16 진수로 출력할 것. **16 진수 표현 시 알파벳은 대문자로 표시**한다.

- 가운데 칼럼은 메모리 내용을 16 진수 형태로 보여준다. 역시 16 진수 표현 시 알파벳은 대문자로 표시할 것.

- 가장 오른쪽 칼럼은 메모리 내용을 byte 별로 대응하는 ASCII code 형태로 보여준다. ASCII code 로 출력해야 할 범위는 16 진수로 20 ~ 7E 까지 이며 그 이 외의 값은 '.' 으로 출력한다.

- 자세한 ASCII code 는 교재의 appendix 또는 인터넷을 참고할 것.

- **dump**

  기본적으로 10 라인이 출력된다. (한 라인은 메모리의 16 개 바이트로 구성)
  dump 의 실행으로 출력된 마지막 address 는 내부에 저장하고 있다.
  다시 dump 를 실행시키면 마지막 (address + 1) 번지부터 출력된다.
  dump 명령어가 처음 시작될 때는 0 번지부터 출력된다
  dump 가 출력할 시 boundary check 를 하여 exception error 처리한다.

- **dump start**

  start 번지부터 10 라인(160 개)을 출력.

  주소를 넘어간 경우 주소의 끝 (0xFFFFF)까지 출력.

- **dump start, end**

  start 부터 end 번지까지의 내용을 출력.

  Start 주소가 end 주소보다 작은 값이 들어온 경우, 에러 처리.

  Ex> dump 37, 4

ex) sicsim> dump 4, 37

```
00000             00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; .........     01
00020 32 33 34 35 36 37 38 39 20 20 20 20 20 2D 3D 2B ; 23456789     -=+
00030 5B 5D 7B 7D 20 20 20 20                         ; []{}    ........
```

참고) dump 시 메모리 주소 0x04 부터 0x37 까지 보여주므로 메모리 0x00 0x01 0x02 0x03 번지의 내용은 16 진수로 나타나지 않는다.
그러나 ASCII code 컬럼에서는 나타나지 않는 부분도 역시 '.' 로 출력하도록 한다.

주의) 프로그램은 어떠한 경우에도 segmentation fault 발생 후 종료되면 안됩니다. 예를 들어 dump 에서 주소 값이 허용범위 밖이 지정되면, 에러 메시지만 출력하고 다음 단계로 넘어가야 합니다. 이후 얘기할 모든 명령어에도 동일하게 적용되는 사항입니다. 만약 명령어를 테스트 하는 도중 segmentation fault 가 발생되는 경우 해당 명령어는 구현하지 않은 것으로 처리합니다.

② sicsim> **e[dit] address, value**

- 메모리의 address 번지의 값을 value 에 지정된 값으로 변경한다.

ex) sicsim> dump 4, 37

```
00000             00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; .........     01
00020 32 33 34 35 36 37 38 39 20 20 20 20 20 2D 3D 2B ; 23456789     -=+
00030 5B 5D 7B 7D 20 20 20 20                         ; []{}    ........
```

sicsim> **e 4, 6D**

sicsim> du 4, 37

```
00000             6D 00 00 00 00 00 00 00 00 00 00 00 ; ....m...........
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; .........     01
00020 32 33 34 35 36 37 38 39 20 20 20 20 20 2D 3D 2B ; 23456789     -=+
00030 5B 5D 7B 7D 20 20 20 20                         ; []{}    ........
```

③ sicsim> f[ill] start, end, value

- 메모리의 start 번지부터 end 번지까지의 값을 value 에 지정된 값으로 변경한 다.

ex) sicsim> du 4, 37

```
00000             00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; .........     01
00020 32 33 34 35 36 37 38 39 20 20 20 20 20 2D 3D 2B ; 23456789     -=+
00030 5B 5D 7B 7D 20 20 20 20                         ; []{}    ........
```

sicsim> f 24, 34, 2A

sicsim> du 4, 37

```
00000             00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; .........     01
00020 32 33 34 35 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A ; 2345**********
00030 2A 2A 2A 2A 2A 20 20 20                         ; *****   ........
```

④ sicsim> reset

- 메모리 전체를 전부 0 으로 변경시킨다.

ex) sicsim> du

```
00000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 20 20 20 20 20 30 31 ; .........     01
00020 32 33 34 35 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A ; 2345**********
00030 2A 2A 2A 2A 2A 20 20 20 20 20 20 54 68 69 73 20 ; *****      This
      .....
```

sicsim> reset

sicsim> du 0

```
00000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
      ....
```

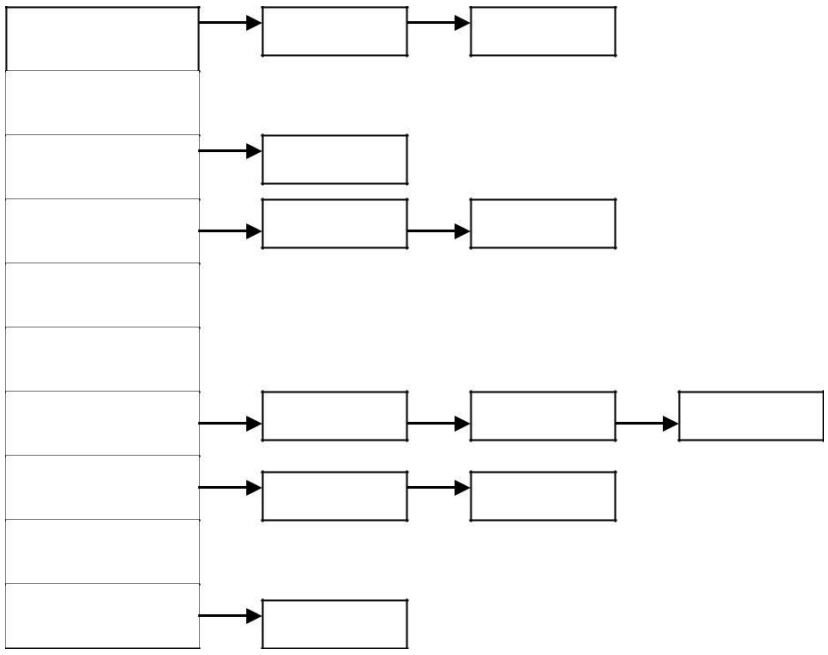
3) OPCODE TABLE 관련 명령어

- sic source 코드를 어셈블러를 통해서 object 코드로 변환시키기 위해서는 주어진 명령어(mnemonic)를 해당하는 opcode 로 변환하는 작업이 필요합니다.(교재 Appendix A(495p~498p) 참고)

1) **opcode.txt** 파일은 Appendix A 에 있는 내용과 동일하며 다음과 같은 형식으로 되 어있음. 제공하는 파일을 사용할 것. (instruction name 은 모두 대문자)

18	ADD	3/4
58	ADDF	3/4
40	AND	2
…	…	…
DC	WD	3/4

2) sicsim 프로그램을 실행시킬 때 opcode.txt 파일의 내용을 읽어 들여서 아래와 같은 모양의 Hash Table 을 만들어야 합니다. **Hash Table 의 사이즈는 20** 이며, Hash Function 등의 다른 사항들은 각자가 디자인 할 것.



- Hash Table 의 사이즈에 비해서 테이블에 넣어야 할 mnemonic 의 개수가 훨씬

시스템 프로그래밍 프로젝트 #1

많기 때문에 충돌이 생기는 mnemonic 들은 위의 그림과 같이 **반드시 linked list 의 형태**로 구현이 되어야 합니다.

- (자료구조 책에 나오니 참고 바람, linked list 가 아닐 경우 감점대상)

① sicsim> opcode mnemonic

- 명령어를 입력하면 해당하는 opcode 를 출력한다.

```
sicsim> opcode ADD
opcode is 18
simsim> opcode LDB
opcode is 68
```

② sicsim> opcodelist

- opcode Hash Table 의 내용을 아래와 같은 형식으로 출력합니다. (아래는 단지 참고사항이며 Hash Function 에 따라서 나오는 값들의 순서는 다를 수 있습니다.)

```
0 : [ADD,18] -> [JEQ,30]
1 : [STS,7C] -> [LDS,6C] -> [JEQ,30]
                ...
19 : [LPS,D0]
```

operation character를 모두 더해서 20으로 나누자!

주의) **아래와 같은 사항에 유의하여 프로그램을 작성하기 바랍니다.**

- Shell 명령어는 모두 소문자로만 인식합니다.
- 인자로 사용되는 숫자는 모두 16 진수입니다.
  16 진수 입력 시에는 알파벳 대소문자 모두 사용 가능해야 합니다.
- 잘못된 명령어나 필요한 인자를 지정하지 않은 경우, 또는 범위를 벗어나는 인자에 대해 적절한 에러처리를 할 수 있어야 합니다.
- 명령어의 탭이나 띄어쓰기 등에 대한 처리는 다음과 같습니다.
  1. dump AA, AB (O)
  2. dump AA , AB (O)
  3. dump AA, AB (O)
  4. dump AA,AB (O)
  5. dump AA AB (X)
  6. dumpAA, AB (X)

  위와 같이 ,(comma)로 인자를 구분하여 구현하시면 됩니다.

3. 기타

3.1 환경 구성

Linux (gcc) : 반드시 gcc 만을 이용해서 프로그램 하십시오. 다른 프로그램으로 작성했을 시에는 0 점 처리합니다. (도스 및 윈도우)

참고) 컴파일 시, make 파일에 gcc -Wall 옵션을 사용하여 warning 을 철저히 확인하시기 바랍니다.
(Warning 발생시 감점 처리함.)

3.2 팀 구성

개별 프로젝트입니다.

3.3 수행기간: 3 월 25 일(월) 23:59 까지

3.4 제출물 (5 가지 파일 중 하나라도 없는 경우에는 0 점 처리함)

1) 프로그램 소스
2) Makefile
3) 프로그램 다큐멘테이션 리포트: 소스 및 프로그램의 구현방법을 설명한 Document. 반드시 예제 파일에(프로그램_도큐멘테이션_리포트_예제.doc) 준해서 작성할 것 (제출하지 않거나 엉터리로 작성할 경우 최대 30% 감점).
4) 프로그램의 컴파일 방법 및 실행방법에 대한 간단한 내용을 적은 README 파일
5) opcode.txt

3.5 제출 방법

sp 학번_proj1 이름의 디렉터리를 만들고, 이 디렉터리에 소스파일, makefile, 도큐먼트, readme 파일, opcode.txt 파일들을 넣어서 **디렉터리를 tar 로 압축하여 한 파일로 만든 후** 메일로 보내시기 바랍니다. (바이너리파일 및 코어파일을 제외한 소스파일만을 넣을 것)

ex) sp20191234_proj1/

README	-> 컴파일 방법 및 실행방법에 대한 간단한 내용을 적은 파일
Document.doc	-> (또는 Document.docx)
20191234.c	-> 소스 파일이 여러 개인 경우 main 함수가 있는 파일의 이름을 학번.c 로 합니다.
20191234.h	-> 최소 한 개 이상의 헤더 파일. 하나인 경우 학번.h

Makefile -> 실행파일은 20191234.out 처럼 학번.out 이름으로 고정할 것.
opcode.txt

Tar 파일로 묶을 때 -z 옵션을 사용하지 않습니다.
tar 파일의 이름은 다음과 같이 지정합니다.

sp<학번>_proj1.tar

ex) sp20191234_proj1.tar

제출 주소 : 2019sp02@gmail.com
메일제목 형식 : [SP_proj#1]_학번_이름
(예: [SP proj#1]_20191234_홍길동)

주의사항

메일로 첨부할 파일이 잘 작성되었는지 확인하고 보내시기 바랍니다.

+ 제출형식(메일제목, tar file 이름 형식, 내용물)이 잘못되었을 시, 감점 10%
+ 중복으로 메일 보낼 시, 1회에 추가로 5%씩 감점 + 제출 시간이 늦춰질 시, 감점

24 시간(1 일) 이내 10%감점
2 일 이내 20%감점
3 일 이내 30% 감점
4 일 이내 40% 감점
5 일 이내 50% 감점 그 이후는 100% 감점

3.6 Source code 관련

Segmentation fault

실행 불가 시 : 0 점
명령 수행 시 : 그 부분점수 0 점

Warning

1 점 감점

Average case

기본 예제 파일 수행

Boundary case

기본 예제 파일 수행 이외에 더 많은 것을 수행
1 건당 5 점 가산

주석

주석이 없거나, 알아볼 수 없는 경우 감점 시키겠습니다.
타인이 알아볼 수 있는 형태로 주석을 달아주십시오.

테스트 방법

[실행파일명]
예) # ./20191234.out

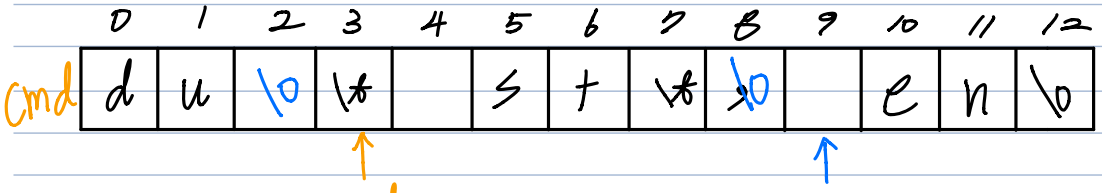
위와 같이 테스트 할 예정이니 착오 없으시길 바랍니다.
실행파일명은 “학번.out”이라고 하시면 됩니다.

3.7 프로젝트에 대한 질문사항은 eclass 질문게시판을 이용해 주세요. 중복 질문이 덜 하도록, 제목에 질문의 요지를 포함해 주세요.

*** 본 프로젝트는 시간이 많이 소요됩니다. 반드시 일찍 시작해서 프로젝트 수행 시 나타나는 질문을 미리 해결해야 프로젝트를 잘 마치실 수 있습니다. 한 주 전에 프로젝트 마감을 목표로 진행하는 것이 중요합니다!

∫ Input
└ Input in form → correct input → history 추가
                ↳ wrong " → continue

① Input을 받아
② format에 맞게 정리하고




```
cmd = strtok(NULL, " ,");
if (cmd != NULL) {
    char start[10]
    strcpy(start, removeSpace(start));

    char hexStart;
    int res = sscanf(start, "%x", &hexStart);
    if(res != 1) {
        error;
        break;
    }
}
else type=1
```

dump
dump start
dump start, end

edit address, value

fill start, end, value

```
cmd = strtok(NULL, "\0");
res = sscanf(cmd, "%x, %x, %x", &start, &end, &value);
switch(res) {
    case 0:
    case 1:
    case 2:
    case 3:
}
```

★ White space 만 들어온 경우 ; continue

i) cmd 만 필요한 경우

→ 이상한 옵션 붙은 경우 error

ii) cmd param 하나

→ param 없는 경우 error

→ param 두개 이상인 경우 error

string 으로서 2byte 의 string 이 들어온다.

3B

int hex;

sscanf(params, "%X", &hex);

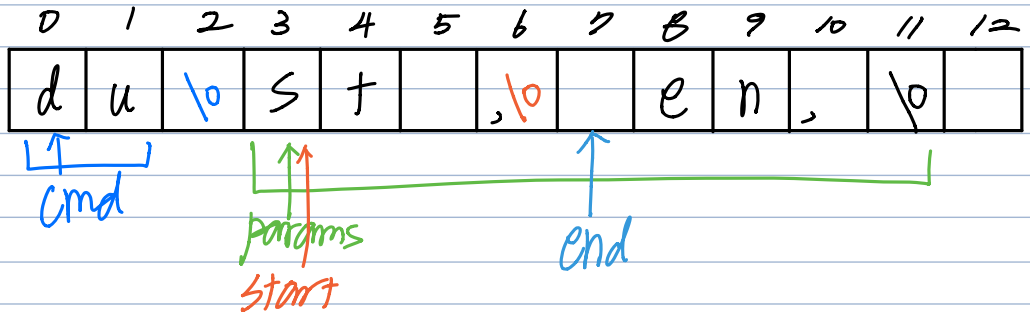
Possible Cases

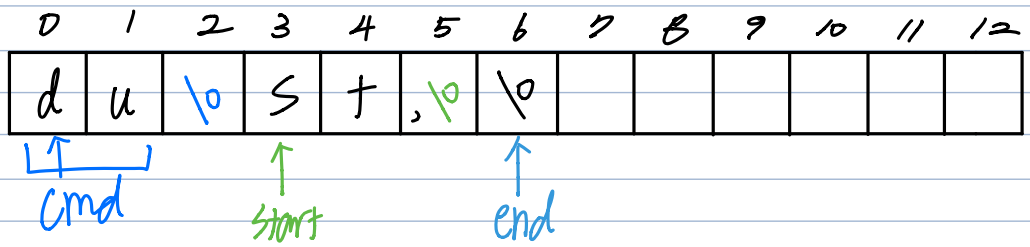
du [s e]

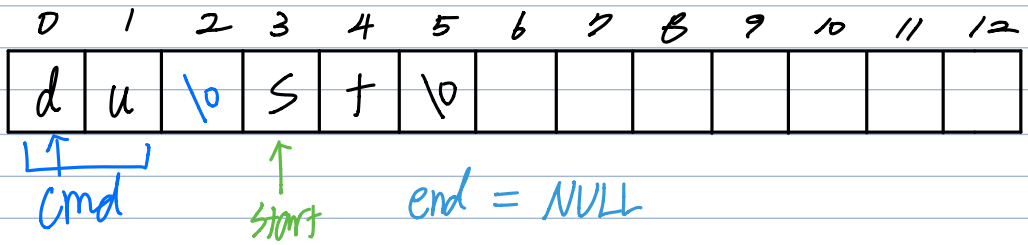
✓ du du s du s, e

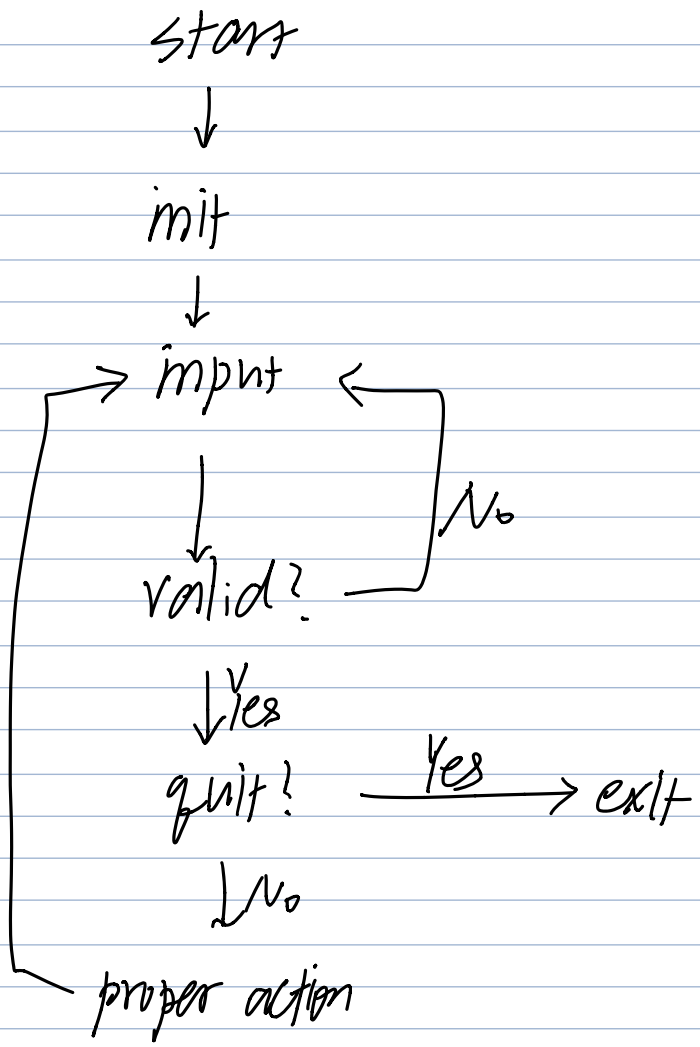
du s,

du s, e,








start
init
input
valid?
No
Yes
quit?
Yes
exit
No
proper action