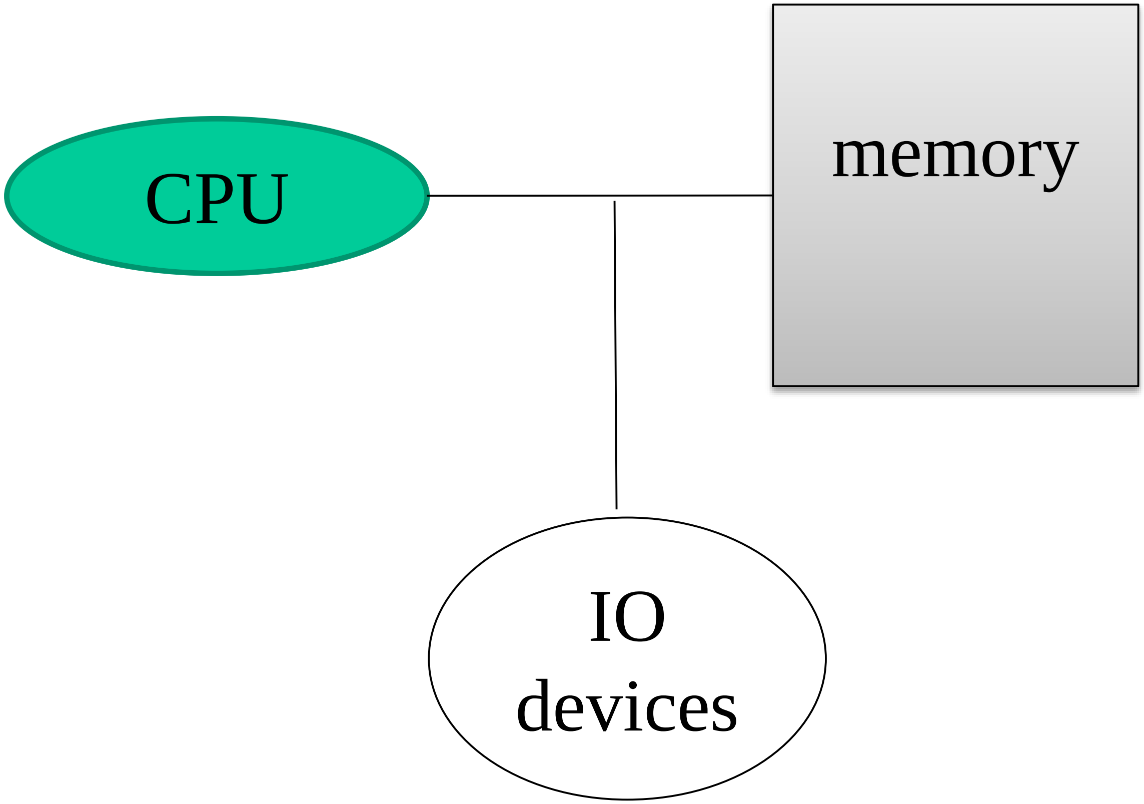
Input/Output



Standard Input/output

- Use library functions such as scanf and printf

- Example in C language

```
int x;

printf("  input = ");
scanf("%d", &x);
printf("== output == %d\n", x);
```

Std. Input/output assembly program

```
        .section ".data"
fmt1: .asciz "== output == %d \n"
fmt2: .asciz "   input = "
fmt4: .asciz "%d"
        .align 4
    x: .word 0

       .section ".text"
       .global scanf, printf, main

main:   save  %sp, -96, %sp
```

```
set fmt2, %o0
 call  printf nop
 set fmt4, %o0
 set x, %o1
 call scanf
  nop
  set x, %o0
  ld [%o0], %l1
  set  fmt1, %o0
  mov  %l1, %o1
  call  printf
  nop
  mov   1, %g1
  ta    0
```

```
fmt0: .asciz "%s"
fmt1: .asciz "%s\n"
        .align 4
        .global main, scanf, printf
main:   save %sp, -240, %sp
        set fmt0, %o0
        add %fp, -144, %o1
        call scanf          ! scanf("%s", buf)
        nop
        set fmt1, %o0
        add   %fp, -144, %o1
        call printf         ! printf("%s\n", buf)
        nop
        mov 1, %g1
        ta 0
```

Input/Output programming

- Programing I/O directly

  ① program control

  ② interrupt

  ③ DMA (direct memory access)

- Programming I/O using OS service

  ✓ OS provides abstraction for device access

  - ➢ IO device sharing
  - ➢ can avoid low level control of device

Memory Mapped I/O

- Addresses assigned to device registers
  - ✓ Some memory addresses are mapped to I/O devices

    (e.g.: 0xfff00000:0xffffe000)
  - ✓ No I/O instructions: use load/store instructions
  - ✓ CPU issues an address for I/O devices to the memory bus => memory system ignores but device controller catches

- Comparison: Isolated IO
  - ✓ Two different address space exist for memory and I/O
  - ✓ I/O specific instructions

Character Devices

- RS232: bit serial data transmission
- Output: stb

```
mov      “a”, %o0
set      0xffff0000, %o1
stb      %o0, [%o1]
```

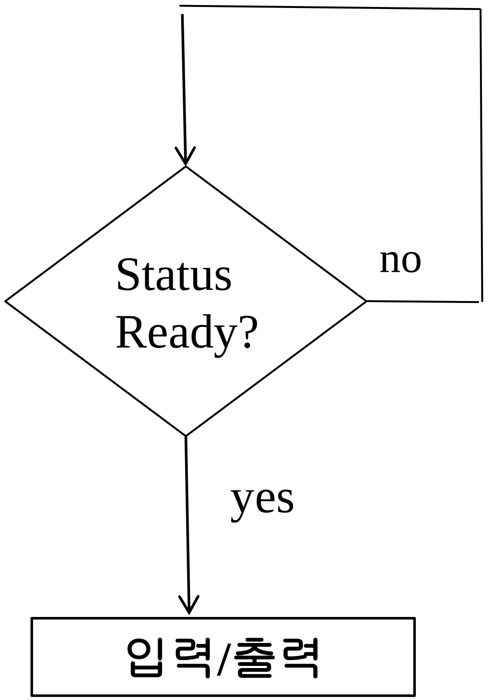
- Input: ldub

```
set      0xffff0008, %o1
ldub       [%o1], %o0
```

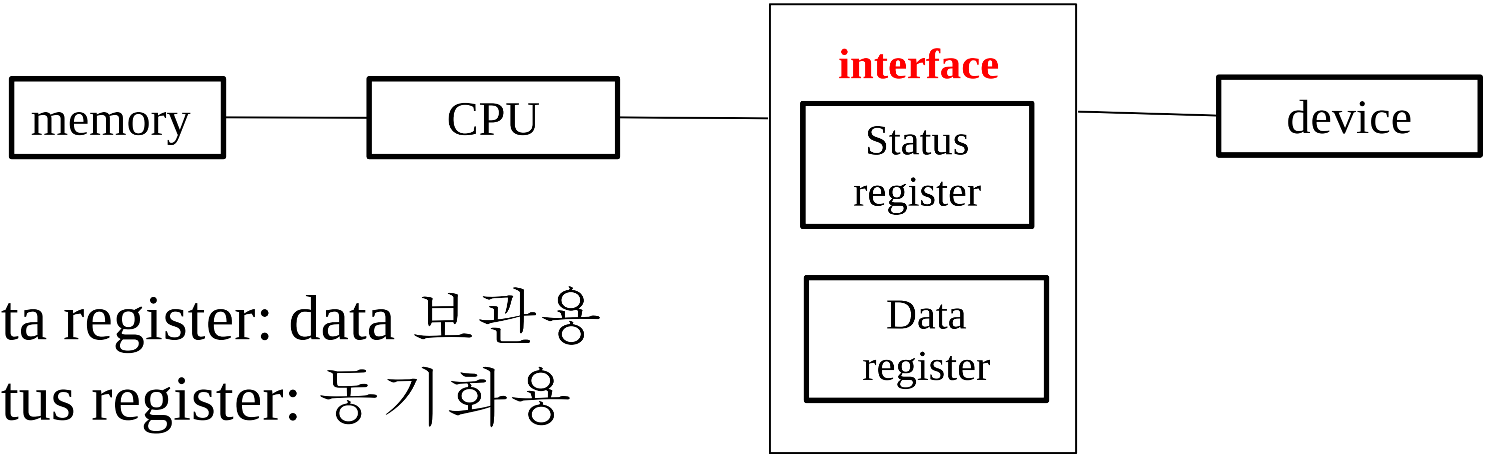
Programmed I/O

- 상태 레지스터 (status register)를 통한 동기화
- ready bit / error bit / interrupt bit
- Busy waiting: device ready 될 때까지 상태 레지스터를 프로세서가 계속 검사

- 입출력 과정



- 관련 하드웨어



✓ data register: data 보관용
✓ status register: 동기화용

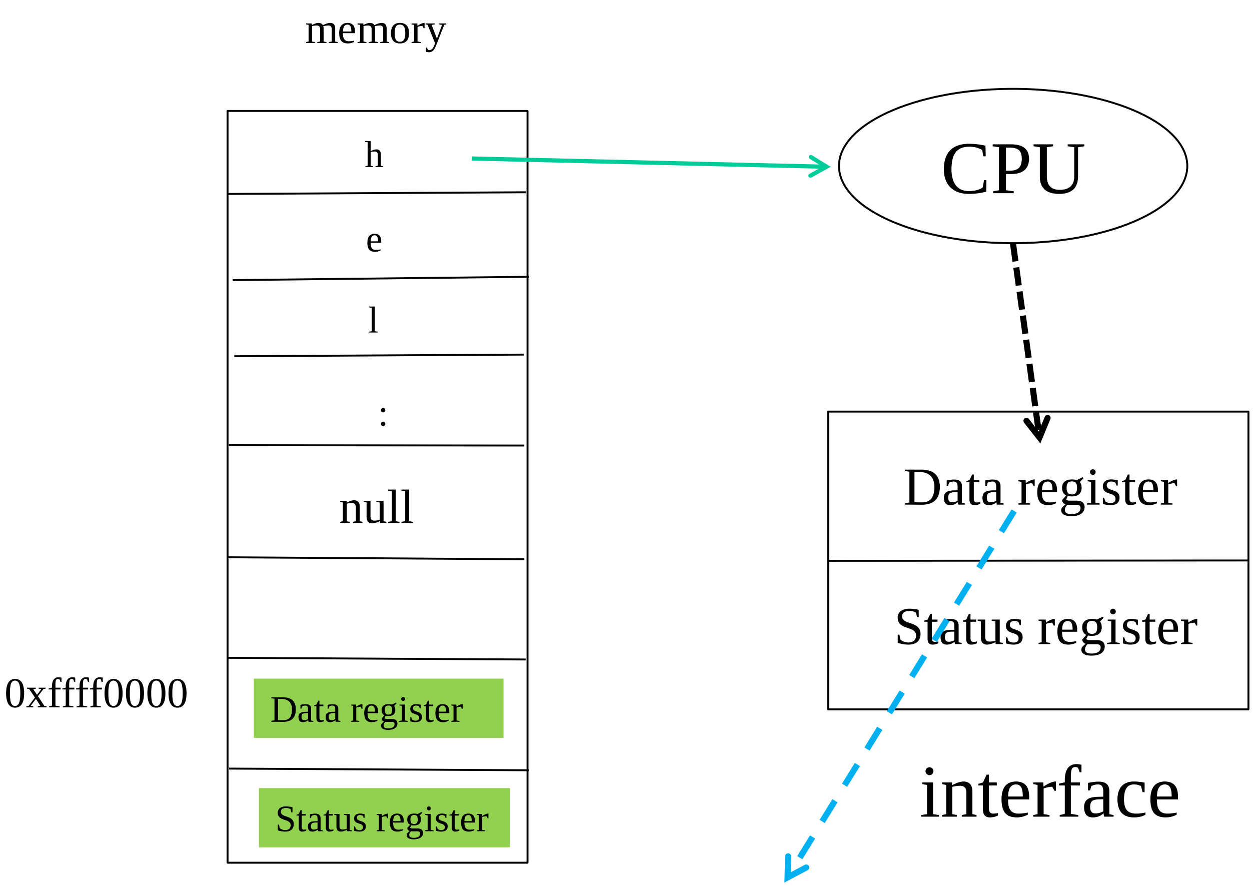
- 상태 레지스터: ready bit, error bit, interrupt bit등

7	6	5	4	3	2	1	0
ready	error	interrupt					

✓ if ready = 1 then 준비 ok

programmed I/O의 예

- "hello, world＼n"을 crt로 출력



```
! structure crt
       data = 0
       status = 4
! crt address
      crt = 0xffff0000
! status register bits
      crt_ready = 0x80
      crt_error =  0x40
      crt_intr  =  0x20
      crt_reset =  0x1
! define register
      ! l2 = crt base register
      ! l3 = pointer to string
      ! l4 = address of pointer
      ! l5 = data
      ! l6 = status
```

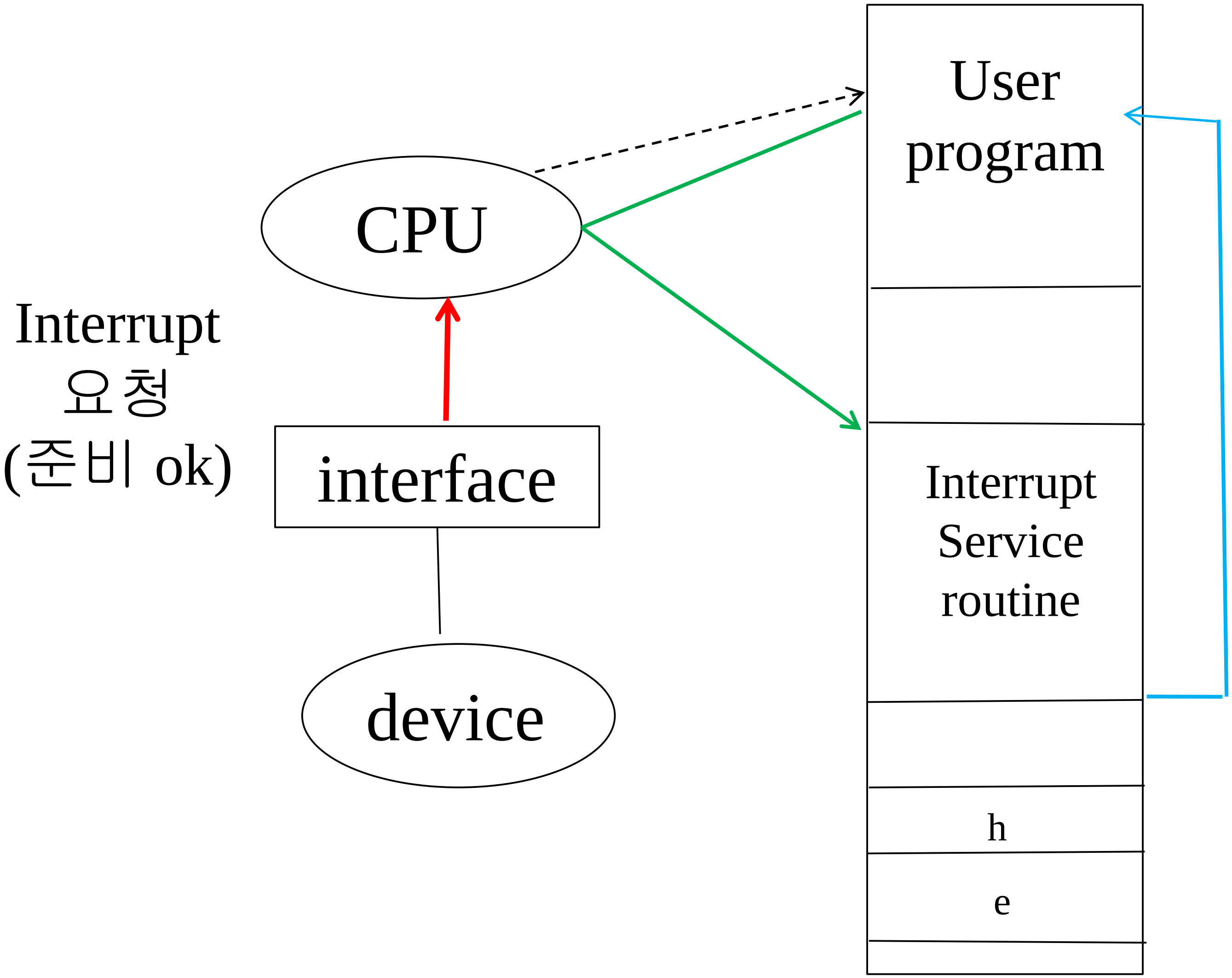
```
 .global   hello, ptr_m

hello:
       .asciz     "hello, world \ n"

     .section ".data"
ptr_m:
       .word hello  ! string pointer
 .section ".text"
       .align 4
       .global  main
 main:  save %sp, -96, %sp

       set ptr_m, %l4  ! address of string pointer
       ld  [%l4], %l3   ! pointer to string
       set crt, %l2     ! addr. crt device struct(32-bits)
```

```
        mov crt_reset, %l6 ! clear error & intr.(8-bits)
        stb  %l6, [%l2+status] ! and thus reset ready bit
        ldub [%l3], %l5    ! output first character
        stb %l5, [%l2+data]
next:   inc  %l3          ! increment pointer
        ldub [%l3], %l5  ! load byte of data
        tst %l5           ! check to see if end string
        be done           ! null character의 경우 branch
        ldub [%l2+ status], %l6 ! load status
wait:  btst  crt_ready, %l6    ! device ready?    and 연산하여 cc를 set
        be    wait             ! ready =0 이면 btst 결과가 0
        ldub [%l2+status], %l6
        ba  next        ! ready-bit = 1 이면 받을 준비 OK
        stb  %l5, [%l2+data] ! output next character
done:  mov  crt_reset, %l6  ! reset device
        stb %l6, [%l2+status] ! clear error & intr.
```

1000 0000
?000 0000
를 and 연산

Interrupt-driven I/O

- Busy waiting 제거
- 입출력 장치가 준비되면, 프로세서를 인터럽트(interrupt request signal)
- 인터럽트 발생
  - ✓ 프로세서가 일하던 상태를 저장하고
  - ✓ 입출력 장치 서비스(interrupt service routine)
  - ✓ 서비스 후 인터럽트 직전의 상태 회복



Interrupt
요청
(준비 ok)

CPU

interface

device

User
program

Interrupt
Service
routine

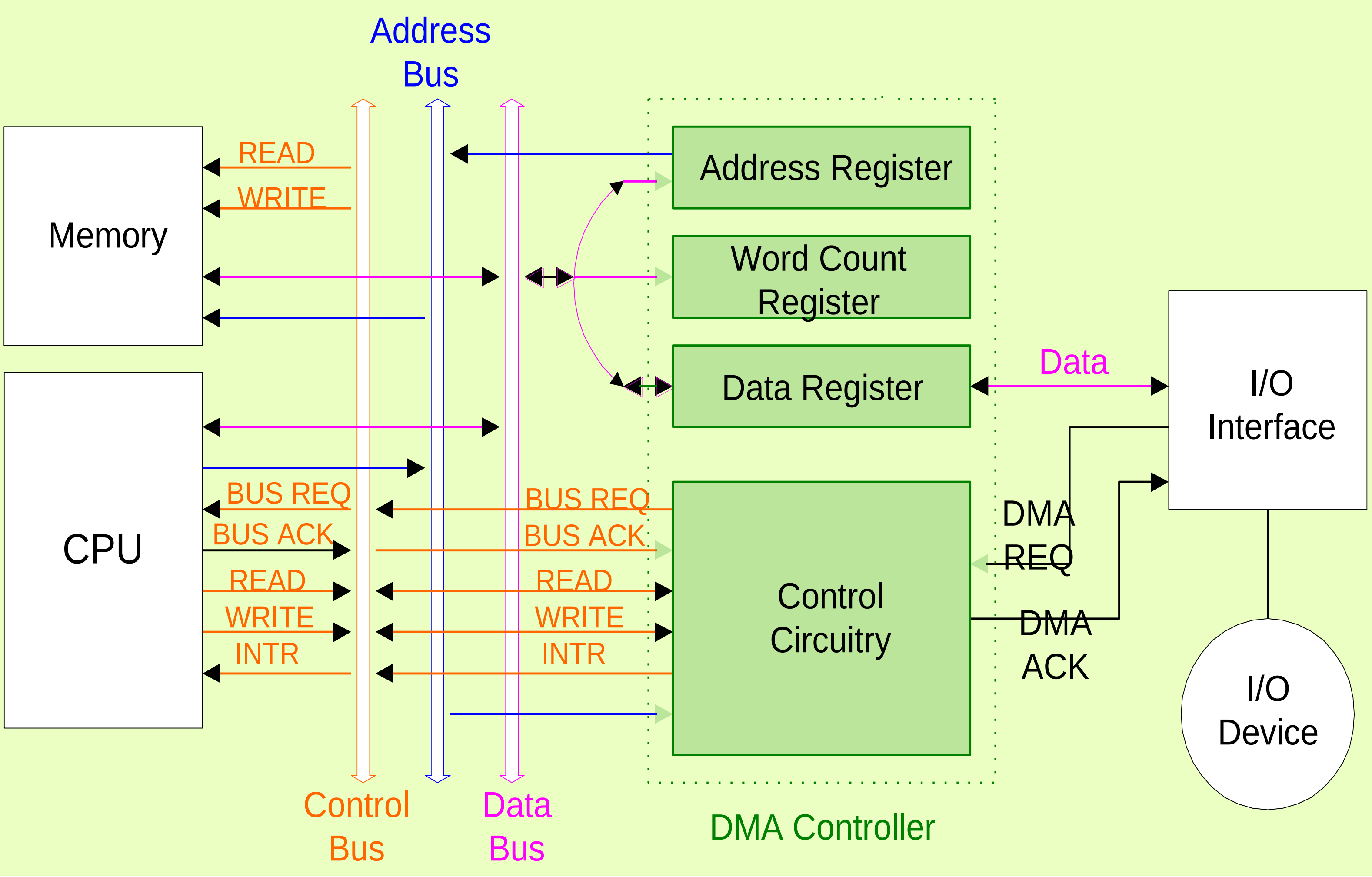
h

e

Block Devices

- Interrupt overhead
  - ✓ 상태 저장과 회복
  - ✓ service routine 실행
- 블록 단위의 데이타 전송(high rate device)
- DMA (Direct Memory Access)
  - ✓ CPU의 도움 없이 입출력 장치가 직접 메모리 접근
  - ✓ DMA controller
    - ➢ starting address와 item count만 갖고 동작 가능

DMA Controller



System I/O

- OS가 입출력 담당: trap, system calls
  - ✓ mov 1, %g1

    ta 0 ! No delay slot
- User mode vs. Supervisor mode
- device = file
  - – 장치 접근 절차와 화일 접근 절차 동일
  - – open → access(read/write) → close
  - – 오류 발생시 CC의 C 비트가 지정
    - ➢ file does not exist, access mode 부적절
  - – fd: 화일 디스크립터, 양의 정수, 열린 화일

트랩 서비스

- ta 명령
  - ✓ %g1에 원하는 서비스 종류 지정

%g1	요청 서비스
3	read
4	write
5	open
6	close
8	create

  - ✓ 필요한 인자는 out 레지스터로 전달
  - ✓ 결과는 %o0로 반환

UNIX I/O

```
int n_read = read(int fd, char *buf, int n);
int n_written = write(int fd, char *buf, int n);
int fd = open(char *name, int flags, int perms);
int fd = creat(char *name, int perms);
close(fd);
```

fd: file descriptor

buf: character buffer

n: 읽거나 기록될 문자수

name: 화일 이름(path name)

flags: read/write/both etc., UNIX man page 참조

perms: 화일 접근 권한

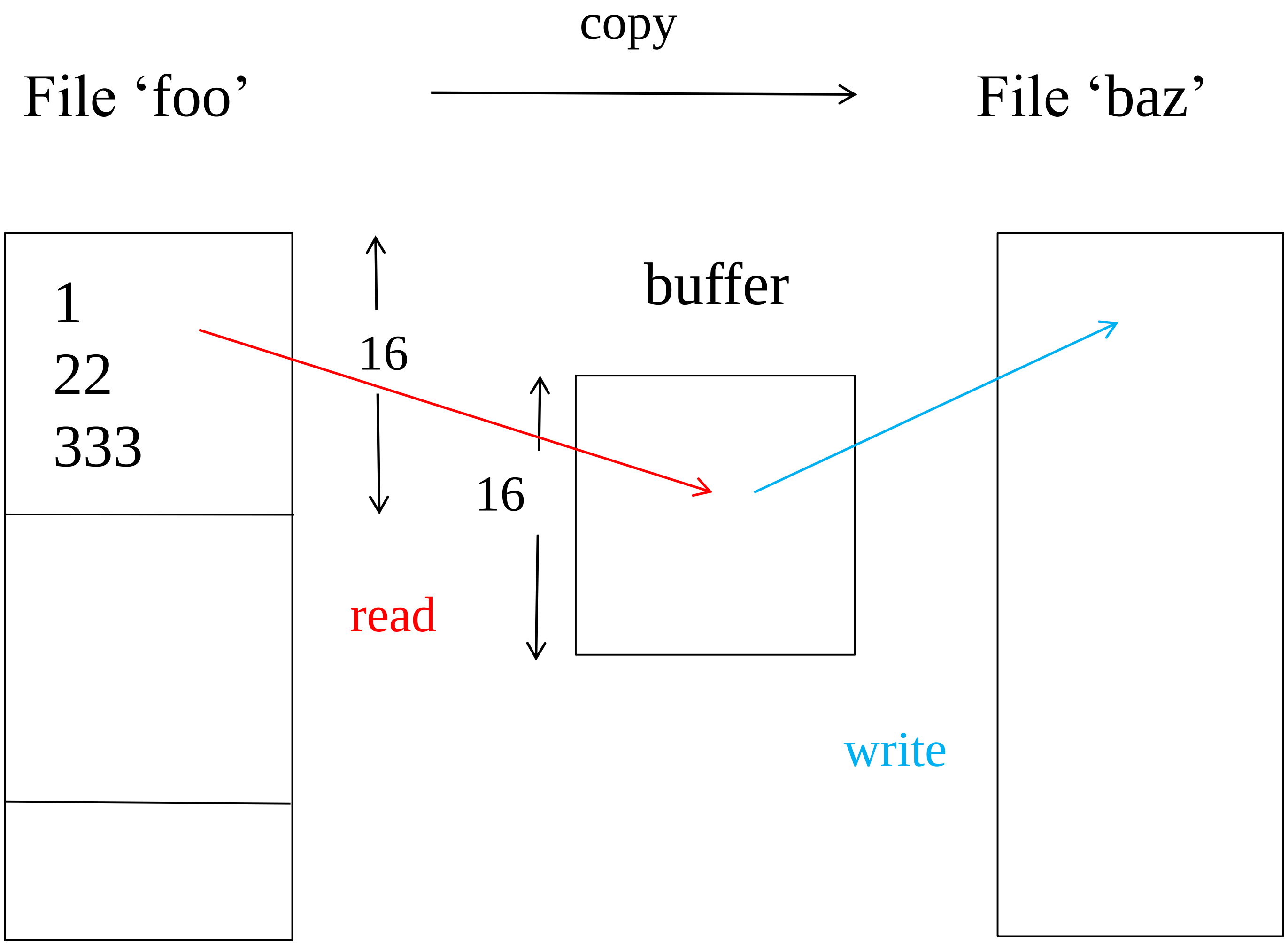
입출력 예: C

화일 ‘foo’를 ‘baz’로 복사

```
#define PERMS 0666
#define BUFSZ  256
main() {
      int ff, ft;
      int n;
      char buf[BUFSZ];

      if ((ff = open("foo", 0, 0)) == -1) exit(0);
      if ((ft = creat("baz", PERMS)) == -1) exit(0);
      while ((n = read(ff,buf, BUFSZ)) > 0)
          if (write(ft, buf, n) != n) exit(0);
}
```

< n의 값>

- ✓ read 성공시: 읽은 문자수
- ✓ eof : 0
- ✓ error : -1 */



copy

File ‘foo’

File ‘baz’

1

22

333

16

buffer

16

read

write

어셈블리 프로그램

```
/* Assembly program */

        OPEN = 5                 !trap definitions
        CREAT = 8
        READ = 3
        WRITE = 4
        O_RDONLY = 0             ! defined in <fcntl.h>

str1:   .asciz  "foo"
str2:   .asciz  "baz"

        .align 4
                                 !%ff_r = %l0
                                 !%ft_r = %l1
                                 !%n_r = %l2
```

```
                !buffer size  :  16
                !local variables :  register 사용
        buf = -16                                 !read/write buffer

        .global main
main:   save        %sp, -112, %sp

        set   str1, %o0
        clr   %o1                                 !open file to read
        clr   %o2                                 !mode
        mov   OPEN, %g1                           !open file for reading
        ta    0
        bcc   open_ok
        mov   %o0, %l0                            !read file descriptor
        clr   %g1                                 !error, exit
        ta    0
```

```
cmp  %o0, 0
bge   open_ok
mov  %o0, %l0
```

```
open_ok:
        set   str2, %o0
        mov   0666, %o1           !file access permissions
        mov   CREAT, %g1          !create file
        ta    0
        bcc   creat_ok
        mov   %o0, %l1            !write file descriptor
        clr   %g1                 !error, exit
        ta    0
creat_ok:
        ba    write_ok            !test
        mov   %l0, %o0            !read file descriptor
read_ok:
        add   %fp, buf, %o1       !buffer pointer
        mov   %l2, %o2            !number bytes to write
        mov   WRITE, %g1          !write
        ta    0
        cmp   %o0, %l2              !check number written in %o0
        be    write_ok
        mov   %l0, %o0              !read file descriptor
        clr   %g1
        ta    0                     !error, exit
write_ok:
        add   %fp, buf, %o1         !pointer to buffer
        mov   16, %o2               !max chars to read
        mov   READ, %g1             !read
        ta    0                     ! %o0: 읽은 문자 개수
        addcc %o0, 0, %l2           !check if any chars read
        bg    read_ok
        mov   %l1, %o0              !write file descriptor
        be    all_done              ! eof
        clr   %g1
        ta    0                     !error, exit
all_done:
        ret
        restore
```