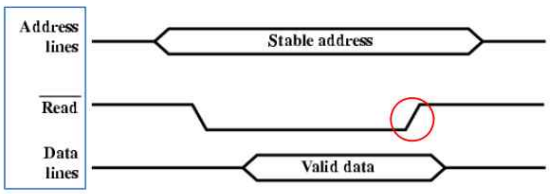
# 컴구 기말 공부

## 1. 비동기 vs 동기

### 1) 비동기(Asynchronous)

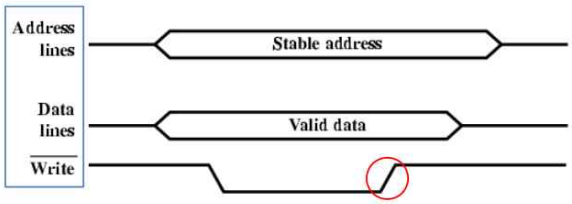
비동기 방식. 쉽다. 이벤트에 맞춰서 함. 시계 필요 없음.

#### (1) Read diagram



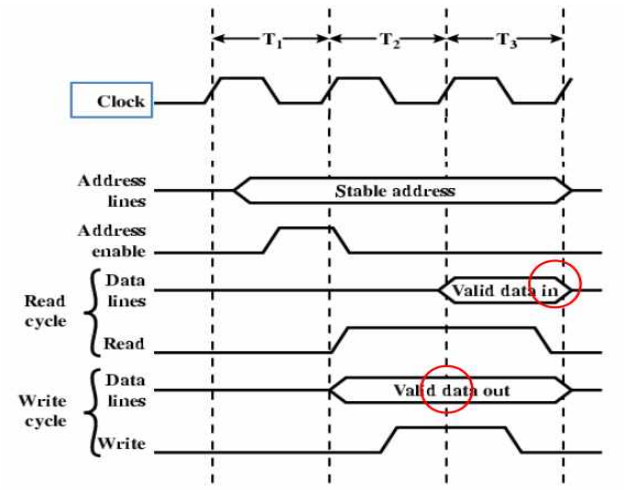

read에 대한 timing diagram chart임. address와 read는 CPU가 만듦. active low니까 내려갈 때 true. read 끝나면 다시 1 됨. rising edge에서 메모리나 다른 io 장치가 valid data를 내놓도록 약속해야 함.

#### (2) Write diagram



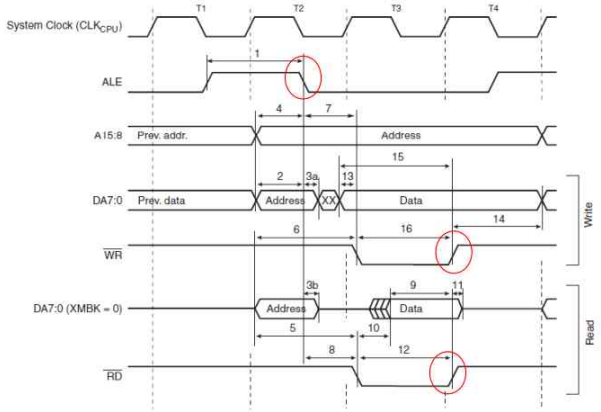

비동기 write도 똑같음. rising edge일 때 data를 들고감. interface하기 쉬움.

### 2) 동기(synchronous)




동기 방식. 시계 필요. 버리는 시간 없이 효율적이게 데이터 전송 가능.

## 2. AVR의 External Memory Timing




### 1) AVR의 address pin

avr은 address pin 16개, 즉 64K 주소 만들 수 있음. 근데, address pin이 8개 밖에 없음. data bus를 겸용으로 만들어서! DA7:0 이게 address 핀이 되다가도 data bus가 되기도 함.

### 2) ALE

address latch enable. 이게 언제 address고 언제 bus인지 가르쳐줌. ALE가 1일 때 address임.

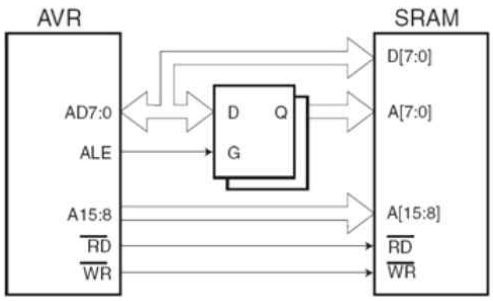
### 3) WR, RD

비동기식으로 WR, RD가 동작.

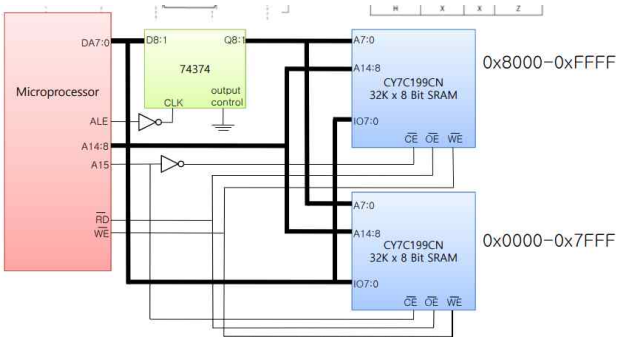
## 3. Latch

latch 소자는 데이터를 문다.

3 state : high, low, Z(high impedance)




Avr은 8개 address 선. 밑에 8개는 address, bus 분리. ALE에 latch 달아서 물고 있음.
address가 없는 때엔 그 전의 address를 latch가 계속 물고 있음. 그래서 address latch enable이라는 이름임. 왜? pin 아껴서 싸게 할려고.




avr은 falling edge에서 유효하다고 한다.
이 latch 소자는 rising edge에서 된다고 함.
그래서 초록색 latch ALE에 not이 붙은거임.

## 4. memory

### 1) internal, external

internal은 cpu 안에 들었다. chip이 cpu 의미.

### 2) rom과 ram

#### (1) rom

read only memory. 이미 새겨져 있음. power 꺼도 계속 있는 프로그램.

① **prom** : programmable. 사람이 딱 한 번 굽고, 그 후엔 못 바꿈. cd rom.

② **eprom** : erasable prom. 유리창 있고, 자외선(UV)으로 지울 수 있음.

③ **eeprom** : 밝기 상태 껐다 켰는데 기억하고 있는거. 전기적으로 지울 수 있음. 설정 같은 거. avr 有.

④ **flash memory** : 한 번에 지우기 가능. 대용량 가능. 한 블록씩 한 번에 전기적으로 지울 수 있음. avr 有. usb.

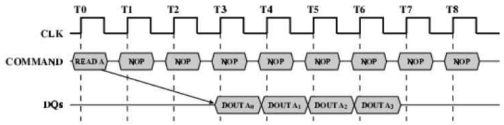
#### (2) ram

write, read 다 됨. 이름 잘못 붙임. (rom도 random access 가능). sequential access 말고도 random access 가능. volatile(;휘발성. 시간 지나면 없어짐. 전원 꺼지면 싹 없어짐.)

① **dram** : dynamic은 한 비트 만들 때 switch 1개, capacitor 1개로 만듦. cap 충전되어 있으면 1, 방전이면 0. 밀도 높아짐. 가격 낮음. 큰 메모리 필요할 때 씀. 전하 다 샌다는 문제점. 시간 지나면 data 깨짐. 몇 ms만 지나면 깨짐. 방전 막기 위해 가끔 재충전 refresh hardware 필요. 비선호됨. 충전 시간 필요. 느림.

② **sram** : static은 고정. refresh 필요 없음. flip flop (4 to 6 transistor) 이라는 digital hardware 씀. on or off 상태 유지 가능. 밀도가 낮아짐. 한 개의 칩 안에 넣을 수 있는 게 1/4로 줆. 가격 비쌈. switch 상태로만 1, 0 표시하기 때문에 빠름.

③ **sdram** : 동기식 dram. 효율 높음. access를 clock에 맞춤. clock 속도에 따라 대용량 데이터 전송. burst mode는 대용량 고속 전송; 시작 address와 clock만 주면 순서대로 쫘라락 sequential access함. random access도 되긴 하는데 느림. pc는 대부분 burst mode로 함. ddr (double data rate; clock 올라갈 때, 내려갈 때 각각 데이터 내놓게 하자) 즉, sdram은 clock 속도랑 똑같음.




## 5. memory hierarchy

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

메모리는 capacity(용량)이 많을수록, 빠를수록 좋지만, 비싸짐. 에 대한 계급 나눔.
<- 그림에서 위에가 빠른 거(비싼 거. 용량 작음), 밑에가 느린 거(싼 거. 용량 큼)!
HDD는 엄청 느림. dram에 비해 몇백 배 느림. CDrom은 (optical)은 Disk보다 몇십 배 느림. Tape는 (지금은 안씀) 엄청나게 느림.

## 6. Von Neumann Bottleneck

cpu, register 짱빠름. (요즘엔 cpu가 4GHz 정도로 clock이 엄청 빨라짐. 1ns도 안됨(0.3ns). 다른 장치들이 그 속도를 못 따라감. register는 cpu 내에 있으니까 cpu랑 속도 똑같음. 3GHz.) but, main memory == ram, dram 느림. (수백MHz)
즉, dram 속도 올려봐야 cpu 속도를 못 따라감. main memory가 느리니까 소용 없음. 본 노이만. bottleneck 병목. cpu와 메모리 사이가 병목.
이 문제를 해결하기 위해,
[방법1] main memory 속도를 늘려. sram으로.

8giga, 16giga를 달아. 그럼 비싸짐.
[방법2] 어쩔 수 없이 dram을 써야 되면, cache memory 3개 집어넣음. L1 Cache, L2 Cache라는 중간 계급(; dram보다는 훨 빠름.) 씀.

## 7. Cache

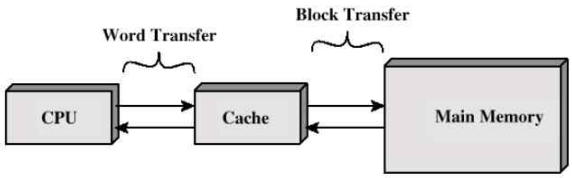
### 1) 개념

Cache는 창고, 메모리임. main memory의 어떤 부분을 cache에 통째로 베껴 놓음. cpu는 cache에서 들고 옴. cpu가 cache꺼 갖다 쓰다가 cache에 없으면 main memory꺼 갖다 씀. 그리고 cache는 또 왕창 갖다 놓음. cpu가 access할 것은 cache에 있을 확률이 큼.
avr 엔 cache란 게 없음.
**(1) cache hit** : cache에 cpu가 원하는 게 있으면 hit함. cache hit 높이려면 cache 용량이 커야 함. hit ratio. 97% 정도 될거임, 내 컴퓨터는.
**(2) cache miss** : cpu가 원하는 게 없음. 그러면 삼성은 LRU (least recently used; 가장 옛날에 사용된 것을 포기하는 방법.)

### 2) cache의 계급

**(1) 32KB L1** : havard 구조.
**(2) 256KB L2**
**(3) 8MB L3** : original von neumann 구조임. 양 多, but 속도는 상대적으로 느림. 물론 main memory보단 빠르고.

### 3) block/word transfer




**(1) block transfer** : cache와 main memory 사이는 cache miss 일어나면 그 부분을 block 별로 통째로 들고옴. burst mode로 함.
**(2) word transfer** : cpu와 cache 사이는 byte별로 random access. 이래서 동기식 dram이 위력을 발휘.

## 8. virtual memory

(<-> physical memory; 물리, 실제 메모리. dram.)

### 1) 개념

내 dram이 8giga로, 부족함. ram 20giga 필요한 프로그램 어떻게 돌릴까? HDD(;용량 매우 큼)를 마치 실제 메모리(ram)인 것처럼 쓴다. 내 컴퓨터가 8giga밖에 없지만 갑자기 32giga, 100giga 등처럼 느껴짐.
c++에서 100번지, 200번지 이렇게 지정하는 건 가상 메모리에서의 100, 200번지임. windows라는 운영체제가 가상 메모리 구동시키는 역할 함. 가상 공간을 프로그램마다 4giga 줄 수 있음.

### 2) paging

hdd에서 퍼오는 거, 즉 왔다 갔다 하는 거.

### 3) page fault

main memory에 없으면 page fault 났다. 그럼 main memory 쓰던 거 내쫓고 hdd에서 퍼와야 함. swap 해야함.
main memory가 워낙 양이 적으니까 page fault 자주 일어나면, 즉 hdd, main memory 간 swapping 계속 하면 계속 느려짐. 그러기 때문에 ram의 용량을 늘리는 게 중요하다.

## 9. IO (입출력 장치)

### 1) Programmed IO 방식

프로그램 수행하는 방법으로 입출력 하겠다. cpu가 일하는 것은 program 도는 거니까 당연한거임. cpu가 일일히 data 하나하나씩 보내는 거임. io 장치는 느리니까 전원 준비되어있나. 즉 io 장치의 상태를 확인해서 읽을지 쓸지 가르쳐줘야 함. data 보냄. 이걸 cpu는 하나씩 하나씩 함.
근데, cpu는 워낙 빠르고 io 장치들은 느림. cpu가 기다리다 끝남. cpu에게 피곤한 일...

### 2) DMA 방식

direct memory access. 직접 메모리를 읽고 쓰겠다. data 전송 시, dma가 단순히 data 읽고 쓰는 행위를 대신 함. cpu가 dma에게 명령함. dma도 당연히 bus controller임. block transfer 시 효율적. 대규모 전달할 때. cpu에게

부담 안줌.

## 10. Addressing IO

### 1) Memory-mapped IO 방식

memory 주소 공간에 io 가 지정된 거. 메모리 중 (~~)는 ram꺼, (~~)는 hdd꺼, (~~)는 io 꺼. 속도는 느리지만 메모리 access 하는 법과 똑같은 방식으로 할 수 있음. cpu는 그게 hdd꺼인지 ram꺼인지 모름. 하지만 방식이 똑같음.

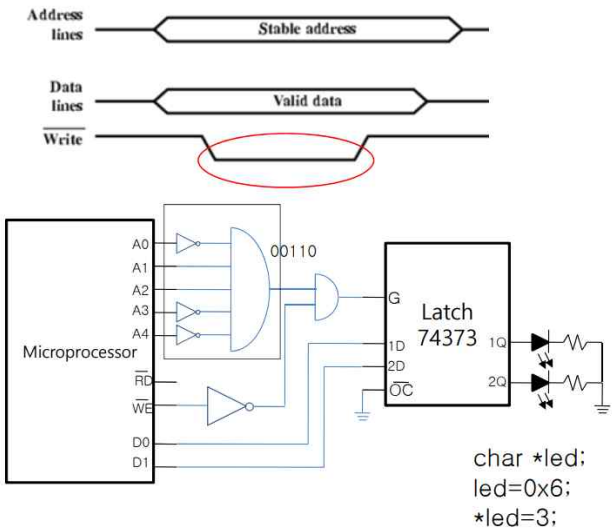
요즘 pc, 모든 스마트폰은 메모리 주소 널널해서 다 이 방식 씀.

### 2) IO-mapped IO 방식

io 주소 및 명령어(; in, out 의미하는) 가 따로 있음. 일반적으로 느림.

옛날 컴퓨터. 점점 없어지는 추세.

## 11. OUT의 예 : LED




A : address가 5bit니까 주소 32개 만들 수 있음.

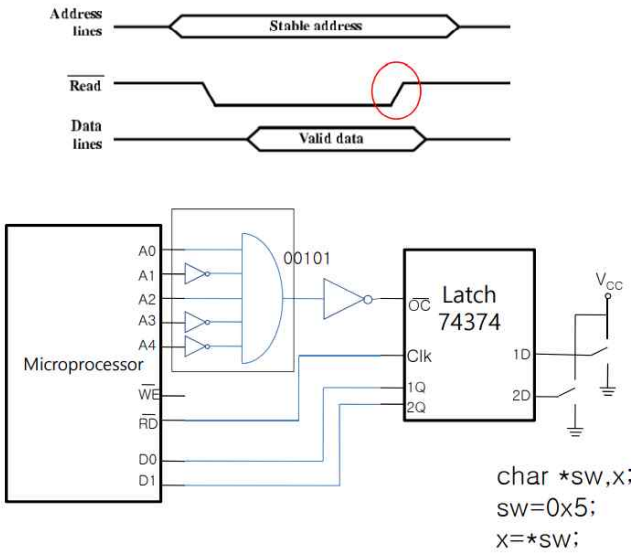
D : 2bit computer임. data line 2개라서.

전구를 두 개 달려고 함. 우선 몇 번지에 붙일까, 하는 주소를 배정해야 함. 이 LED 장치를 6번지에 하자! latch는 G라는 핀이 high 되었을 때 데이터 내보냄, 즉 한번 쓰면 그 선택을 계속 유지토록 함. 6번지니까 address line에서 00110 만듦.

동시에 true일 때 뒤 data 버스가 연결돼서 led 켜짐. read는 필요 없지. 쓰기만 하면 되니까.

코드 : char *led; led=(char*)0x6; *led=3;

## 12. IN의 예 : switch




switch로 입력. 보통 때에는 선이 끊어져 있어야 됨. 주소 가장 먼저 배정. 이 경우 5번 주소라고 하자! 이 latch 소자는 output control이라는 pin 있음. 보통은 Z상태 즉, 떨어져있는 상태.

이 소자는 입력 부분에 언제 값이 넘어오냐면 clock의 rising edge일 때임. read의 rising edge니까 다이렉트하게 연결.

코드 : char *sw, x; sw=(char*)0x5; x=*sw;

## 13. 입출력의 두 가지 방식

avr은 latch꺼까지 다 만들어놓음.

입출력이라고 하는 것은 외부의 상태, 장치와 데이터 주거니 받거니 하는거임. io 하려면 외부 장치랑 interact 해야 함. 그러려면 그쪽의 상태도 알아야 함.

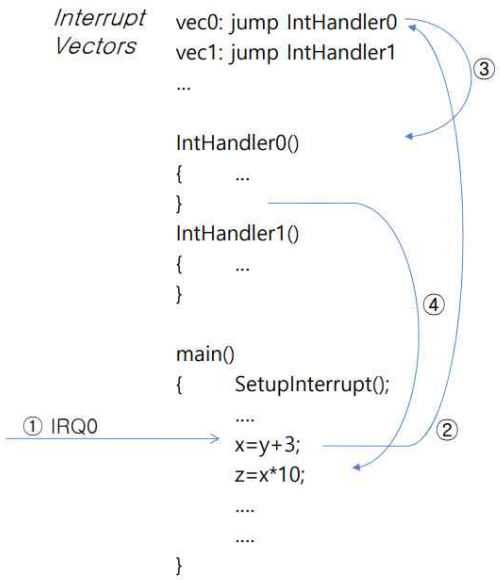
### 1) Polling 화장실

cpu가 장치에게 계속 노크하면서 물어보는 소통 방식. device 입장에선 느닷없이 외부에서 노크한 거임. polling은 loop임. 장점은 simple. 단점은 inefficient.

### 2) interrupt 전화기

device가 하던 일 멈추고 cpu에게 request 하는 것이 interrupt (= issue). cpu가 책 읽는 도중에 전화가 울리는거임. 장점은 cpu가 자기 일을 할

수 있음. 단점은 hardware; interrupt하는 핀 있어야 함.
program 짤 때에는, interrupt는 언제 외부에서 걸지 모르니까 비동기 방식으로 짜야됨.
(1) 장치가 cpu에게 interrupt request : IRQ.
(2) cpu가 응답 : interrupt acknowlege.
(3) cpu가 씹음 : mask. IMR. mask 씌워놓으면 device가 아무리 interrupt 걸어도 cpu는 안들림.
(4) interrupt vector : 주소임. 코드의 주소.
(5) interrupt service routine : cpu가 하는 일련의 순서.
(6) 외부 장치가 interrupt request 걸었음. 내가 해야될 일이 쓰여있는 곳의 주소가 interrupt vector. 즉, 전화 걸려오면 갑자기 interrupt vector로 뛰고 interrupt service routine 수행하고 다시 돌아옴.




main program에서 SetupInterrupt() 함수 부름. interrupt 다 살려놓음. ① irq0 신호 들어오면 하던 일 멈추고. ② interrupt vector 0로 감. ③ 이게 interrupt service routine임. 다 끝나면, ④ 원래로 돌아옴. 외부 pin이 함수를 부르게 하는 거임. inthandler0 끝나면 return 되니까 원래 자리로 돌아옴.

### 14. MPU microprocessor unit

반도체 발달 전엔 intel이 cpu 다 합쳐놓음. 작아서 micro processor라 함. 관습적으로 지금도 그냥 부름. cpu임.

### 15. MCU microcontroller unit

얘도 cpu임. 제어용 목적으로 만든 전용 cpu를 mcu라 부름. mcu에는 이 칩 하나로 모든 걸 해결할 수 있도록 ram rom latch 통신장치, analog 장치 달 수 있도록 하는 adc도 넣음. 반도체 기술 발달해서 가능. all in one임.
power consumption이 적어야 하는 게 중요.
avr이 이거임. 8051은 옛날 꺼.

### 16. Digital Signal Processor

digital signal processor도 mcu임. 계산을 무지하게 잘함. 곱하기, 더하기를 한 cycle에 해치우고. fast computation (parallel multiplier accumulator, hardware - controlled loop, pipelinnig, fft addressing, dma, fpu)
adc, dac 가 있는 dsp가 많음.
없어도 이걸 간단히 연결할 수 있게 만들어놓음.
TI의 320 시리즈.

### 17. debegger

in circuit emulator (ICE)는 avr을 내 컴에 연결했을 때 내 컴에 얘가 뭐 수행하고있는지 다 보이는거임. JTAG

### 18. sw 개발 툴

editor, c compiler, linker 필요. 통합된 게 visual studio, avr studio 등. avr studio 깔아야 함. 근데 이건 c compiler 없음. 설치해야 됨.

## 1. Microcontroller AVR

avr은 atmel이 만듦. 학생이 개발. 8bit com, RISC다. low power다. flash memeory 내장되어 있어서 별도의 메모리 달 필요 없다.

## 2. Basic Families

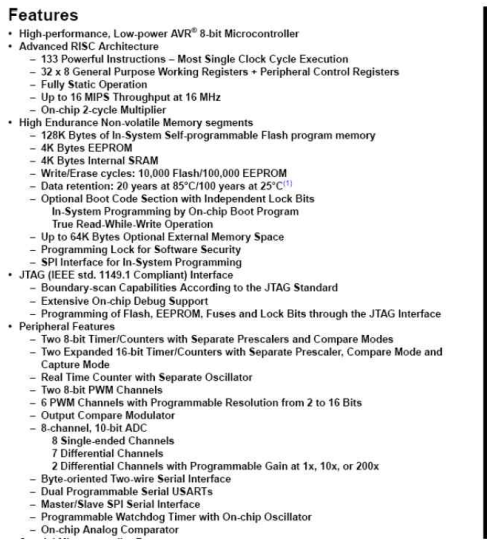
avr은 family라 불리는 게 있음. 같은 컴퓨터라서 호환되는 거 뺌. 소자들이 원하는 거 고르는 걸 line up이라 함. avr에도 여러 lineup 있음.
rom size, pin개수, 내장옵션 다름.

**1) ATiny 시리즈** : 제일 간단. 1~8kB program memory, 9~32 pin.

**2) ATmega 시리즈** : (우리 avr) 4~256kB program memory, 28~100 pin, multiply 등.

**3) ATxmega 시리즈** : DMA, DAC 있음.

## 3. AVR datasheet



**Features**

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 133 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers + Peripheral Control Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 128K Bytes of In-System Self-programmable Flash program memory
  - 4K Bytes EEPROM
  - 4K Bytes Internal SRAM
  - Write/Erase cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Up to 64K Bytes Optional External Memory Space
  - Programming Lock for Software Security
  - SPI Interface for In-System Programming
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - Two Expanded 16-bit Timer/Counters with Separate Prescaler, Compare Mode and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Two 8-bit PWM Channels
  - 6 PWM Channels with Programmable Resolution from 2 to 16 Bits
  - Output Compare Modulator
  - 8-channel, 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels
    - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Dual Programmable Serial USARTs
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with On-chip Oscillator
  - On-chip Analog Comparator

ATMEL

8-bit AVR® Microcontroller with 128K Bytes In-System Programmable Flash

ATmega128
ATmega128L

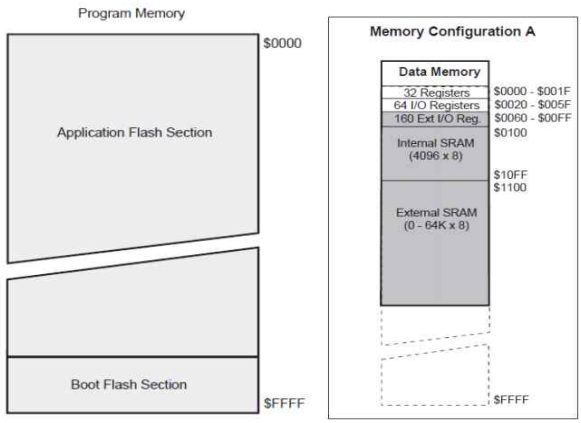
avr 데이터시트의 첫 번째 장. flash memory rom이 128k byte라서 **ATmega128**이다. L 붙은 건 3.3V, 그냥 껀 5V (우리꺼).

## 4. WinAVR 기능

1) SRAM에서의 변수 및 상수
sizeof(int) = 2 = sizeof(short)이다.
sizeof(longlong) = 64
2) PM에서의 상수, EEPROM에서의 변수
3) IO registers
4) Interrupt programming
5) Standard and special functions
ex) sei, cli, _delay_ms, eeprom_rb, ...
AVR studio - Help -> avr-libc Reference Manual

## 5. 메모리 구조

독특한 io 사용한다는 것은, programmer 입장에서는 그것과 관련된 내부의 special register에 정해진 값을 쓴다는 얘기임.




**1) rom**

rom은 program memory.를 flash memory형태로 가지고 있음. 0~64K까지. 한 주소가 2byte라서 총 128K임.

**2) ram**

data memory는 0~64K. 이미 조금 쓰고 있음. 내부 범용 레지. 내부 special regi. 확장용 regi. internal ram에 4K만큼 이미 차지함.
hexa면 앞에 $ 붙인거. 내부메모리 4K로 왠만하면 다 함. 0100 ~ 1100 까지 이걸 쓰는거임 우리는.

## 6. AVR의 special register

MCUCR (MCU control reg.)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SRE | SRW10 | SE | SM1 | SM0 | SM2 | IVSEL | IVCE | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

External mem. enable

XMCRA, XMCRB (External memory control reg.)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | SRL2 | SRL1 | SRL0 | SRW01 | SRW00 | SRW11 | – | XMCRA |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Sector limit — Wait cycles (U/D sectors)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | XMBK | – | – | – | – | XMM2 | XMM1 | XMM0 | XMCRB |
| Read/Write | R/W | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Port C/ A15:8

MCUCR의 7번째에 1 쓰면 외부 메모리.

### 7. clock control

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | XDIVEN | XDIV6 | XDIV5 | XDIV4 | XDIV3 | XDIV2 | XDIV1 | XDIV0 | XDIV |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

$$f_{CLK} = \frac{\text{Source clock}}{129 - d}$$

내가 안쓸 때, 고성능 필요 없을 땐, clock 속도 떨어뜨려서 성능 떨어뜨리고 배터리 아낌.

XDIV라는 레지스터의 7번째 비트가 0이면 동작 안함 16MHz(기본). 7번째 비트가 1이면 0~6비트으로 d값 설정.

### 8. AVR의 기본 HW

**1) memory lock bit and fuse bit**

메모리 잠궈서 남들이 못보게 함.

**2) watchdog timer**

일종의 시계로, controller에만 있음. 일반 pc cpu엔 없음. 내 cpu인데 자기가 죽었는지 살았는지 보는 거임. 정해진 시간 끝나면 cpu의 리셋 버튼 눌러서, 오동작 할 때 껐다 키는 셈.

프로그래머가 어떤 식으로 쓰냐면 10초, 5초 이런 식으로 설정하면 5초 지나기 전에 이걸 다시 5초로 만듦. 이 프로그램이 수행되는 한 이 시계는 절대 0 안감. 기계가 오동작하면, 이 프로그램 자체가 안돌으니까 watch dog timer 시계를 그 프로그램이 5초로 다시 만들 수 없음. 0이 되면 cpu를 리셋(hard reset)시킴.

**3) sleep mode**

최소한의 기능만 켜놓고 cpu가 좀 쉬는 거. 나머지는 power 다 끊어서 power 절약.

**4) boot loader**

power 딱 키면 뭔가 해야됨.

**5) jtac interface**

incircuit emlator의 국제 표준 그룹의 이름.

### 9. IO port

port는 외부로 data를 주거니 받거니 하는 항구.

**1) general digital IO**

범용으로 쓸 수 있게 한 것. avr에 많이 준비되어 있음. bidirectional(양 방향) parallel한 io port의 pin은 53개고, 각 핀들이 독립적으로 동작.

**2) PORTA~PORTF(8bit), PORTG(5bit)**

avr은 8bit cpu니까 8bit 단위가 여러 개. 즉 48bit가 입출력 가능. 근데, PORTG는 5bit만 쓸 수 있음. 3bit를 다른 목적으로 씀. 이렇게 해서 총 53bit임.

**3) Max 40mA**

보통 digital pin들은 0,1만 표시하면 돼서 전류 거의 안흐름. 상태만 보여주면 되니까. avr의 io port는 전류를 40mA 흐를 수 있게 되어있음. 엄청 큰 단위임.

**4) Optional pull-up resistors for input**

optional pull up resistor도 내장하고 있음.

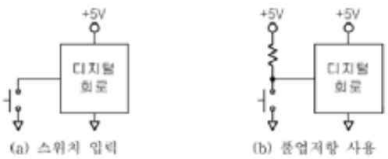
**5) DDRx, PORTx, PINx**

DDRx는 data direction register로, 0은 in, 1은 out. 독립적으로 입출력 설정 가능. PORTx는 output, PINx는 input.

```
DDRA=0xff; //porta는 전부 출력모드.
porta=0xff; //porta는 전부 1을 출력. 전부 led 켜짐.
DDRB=0x0; //전부 입력모드
while(1) PORTA=PINB; // 스위치 값을 읽어 그대로 LED에 출력 반복.
// b는 스위치, a는 led.
```

### 10. pull up / down resistor

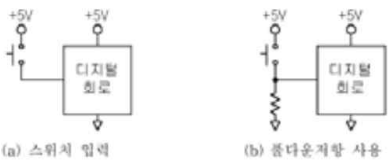
**1) pull-up 저항**




a: 스위치 누르면 0V, 떼면 ??.

b: 스위치 누르면 0V, 떼면 5V.

**2) pull-down 저항**




a: 스위치 누르면 5V, 떼면 ??.

b: 스위치 누르면 5V, 떼면 0V.

=> 이렇게 하면 초기값 정확히 부여 가능. 또 다른 목적은 모터 할려고. 외부에 추가 전원 연결해서 하는 open collector, open drain 회로에서 출력 전류를 증대시킬려고. 출력 모드에서 pull up registor 필요.

| DDxn | PORTxn | PUD (in SFIOR) | I/O | Pull-up | Comment |
|---|---|---|---|---|---|
| 0 | 0 | X | Input | No | Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low. |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

pull up resistor 필요한 사람은
SFIOR의 두번째 비트인 PUD란 비트를 설정하면 io port에 pull up 저항 살려줌.
선택적으로 pull up 저항 필요하면 입력 포트에 자기네들이 미리 준비해놓음.
그걸 1로 하면 pull up 상태임.

## 11. Alternate port function

무려 53개를 io port 니까 다리가 64개니까 이걸 다른 목적으로 쓸 수도 있게 함.

### 1) PORTA

**AD7:0, in/out, 7segment LED**

Address Data pin 0~7. 외부에 address 달거면 porta 포기하고 이걸로 하는거임.

### 2) PORTB

**in/out, timer/counter, SPI**

incircuit emelator 다운로드 할 수 있지만 isp 가능하다. 그 핀 4개를 씀. incircuit emulator 없는 사람은 PORTB0~4 이걸로 쓰는거임.

### 3) PORTC

**A15:8, 0:3 7segmentLED, 0:3 out, 4:7 in/out.**

portc는 address pin 필요한 사람은 이 portc를 포기하고 쓰는거임.

### 4) PORTD

**4:7 in/out, timer/counter, external interrupt, usart1/TWI**

내부의 시계로, interrupt로, 0~4는 외부 통신으로도 씀.

### 5) PORTE

**4:7 in/out, timer/counter, external interrupt, usart0**

우리가 interrupt 과제할 때 쓴 포트.

### 6) PORTF

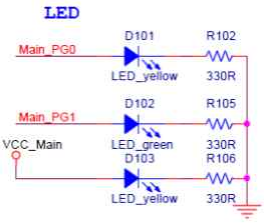
**0:3 in/out, JTAG, ADC**

### 7) PORTG

**0:1 LED, 2:3 Switch, in/out, timer/counter, external memory interface**

## 12. LED interface

### 1) PORTG의 LED




```
DDRG = 0x3; //0bit, 1bit를 출력모드
PORTG = 0x3; //0,1번째 비트에 1을 주므로 led 킴.
```

```
DDRG=0x3; //0,1번인 LED를 출력.
while(1) {PORTG=PING>>2;
_delay_ms(200); //200초 동안 이 루프 돈다.
}
```

### 2) PORTA에 LED

```
DDRA = 0xff; //모두 출력모드.
PORTA = 0xff; //모두 킴.
```

## 13. Switch Input

### 1) PORTG의 스위치

```
char sw[2];
sw[0] = PING&0x4; //PING은 G port의 input. masking을 씌움. ping에 입력한 데이터와 0000 0100을 and하면 2번 비트만 살아남음.
sw[1] = PING&0x8;

if (PING&0x4)
{ ①.... //사람이 스위치 누르면.
  while (PING&0x4);
  ②.... //사람이 스위치 떼면.
}
```

**2) PORTA에 스위치**

```
char sw;
DDRA = 0x0; //모두 입력모드.
sw = PINA;

sw&0x1? //0번비트 눌러져있는지아닌지
sw&<(1<<n)?
```

**3) 스위치에 capacitor**

capacitor가 switch에 연결되어 있는 이유는 debouncing condensor, 즉 튀는 걸 없애는.

//////////////////과제4//////////////////

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <avr/delay.h>

char number[10] = {0x3f, 0x06,..., 0x67};
//font, active high.
void main() {
char digit[4] = {3,4,7,9};
DDRC = 0x0f; // port C direction
DDRA = 0xff; // port A direction
while (1) {
PORTC = ~0x01;
PORTA = number[digit[3]];
_delay_ms(1); //1ms 지연시킴.

PORTC = ~0x02;
PORTA = number[digit[2]];
_delay_ms(1);

PORTC = ~0x04;
PORTA = number[digit[1]];
_delay_ms(1);

PORTC = ~0x08;
PORTA = number[digit[0]];
_delay_ms(1);
}}
```

## 14. 7 Segment LED

7개의 선, 한 개의 점으로 0~9, A~F도, 16진법도 표현 가능. 숫자 한 개당 포트 하나, 즉 포트 네 개 쓰는 건 아까우니까(;avr의 총 포트 6개 뿐), newtc의 꼼수는, 포트 두 개 씀.

**(1) A포트** : 숫자 데이터를 각 latch에 load. 공용. 모든 led를 portA에 연결, 시간 간격 주며 껏다켰다.

**(2) C포트** : latch 선택. enable 단자에다가 portC0 ~ portc3 달아둠. active low로.

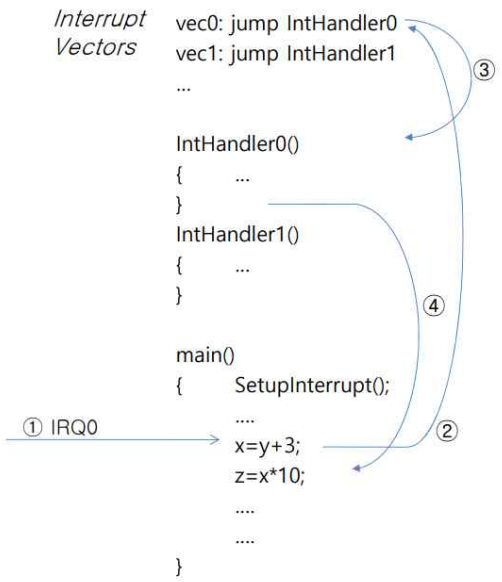
## 15. Interrupt

**1) 개념**

- polling은 화장실, interrupt는 전화기.
- 우리는 portD 0:3 말고(;얘는 외부 통신 기능용) portE 4:7을 interrupt할 때 씀.
- interrupt를 건다 = issue한다 = 외부에서 cpu에게 신호 보내는 것.
- 시계, 통신 등 이런 건 내부 interrupt. 매우 많음. 우리는 당장 timer adc 이런 거 안배웠으니까 외부 interrupt 활용해서 연습하겠다.

| | Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|---|
| Highest priority | 1 | $0000(1) | RESET | External Pin, Power-on Reset, B Watchdog Reset, and JTAG AV |
| | 2 | $0002 | INT0 | External Interrupt Request 0 |
| | 3 | $0004 | INT1 | External Interrupt Request 1 |
| | 4 | $0006 | INT2 | External Interrupt Request 2 |
| | 5 | $0008 | INT3 | External Interrupt Request 3 |
| | 6 | $000A | INT4 | External Interrupt Request 4 |
| | 7 | $000C | INT5 | External Interrupt Request 5 |
| | 8 | $000E | INT6 | External Interrupt Request 6 |
| | 9 | $0010 | INT7 | External Interrupt Request 7 |

- interrupt vector는 interrupt 개수만큼 있음. 우선순위가 빠른 거 먼저 처리. 가장 우선순위 빠른 것은 reset. $는 헥사 표시. 0x와 같은 말.
- 외부 interrupt 0~7가 그 다음 우선순위. 2word씩 건너 뜀.
- interrupt 7번 쓸거면 0x10라고 써야되는 거임. int4 쓰려면 0xA에 프로그래밍 해야되겠지.




- 그런데, 겨우 2word가지고 service routine을 길게 쓸 수 없음. 명령어 하나만 jmp ISR_INT0 이렇게 딱 써놔서 우선 벡타로 오고, interrupt service routine으로 가서 수행하고 원래자리 가겠지.
- ISR_INT0 이런게 내가 짤거. IntHandeler임.

### 2) interrupt 관련 registers

**(1) SREG** (bit 7) (호텔문)

얘를 1로 만들면 전체 interrupt를 활성화 (enable)

**(2) EMISK**(external interrupt mask register 방열쇠 8bit)

각각의 interrupt를 독립적으로 살리고 죽임.

**(3) EIFR**(external interrupt flag register 8bit)

interrupt 들어오면 깃발로 독립적으로 알려줌.

**(4) EICRA, EICRB**

레지스터 설정. 각 비트 당 2개 필요. interrupt a는 0~3, b에는 4~7.

| ISCn1 | ISCn0 | Description |
|---|---|---|
| 0 | 0 | The low level of INTn generates an interrupt request. |
| 0 | 1 | Reserved |
| 1 | 0 | The falling edge of INTn generates asynchronously an interrupt request. |
| 1 | 1 | The rising edge/Any logical change on INTn generates an interrupt request |

00이면 해당 interrupt 걸림.
01이면 reserved. 지금 안쓰지만 미래 위해 확보.
10이면 falling edge에 따라 interrupt 걸음.
11이면 rising edge에 따라 interrupt 걸음.

### 3) interrupt하는 원리

interrupt request는 외부 장치가 0~7의 pin에 edge나 logic 변화로 trigger 가할 때, interrupt flag의 0~7가 1이 되는 거임. sreg의 I-bit(7번째비트를 의미)와 해당되는 interrupt enable bit가 EIMISK에 set 되어 있으면, 즉 두 열쇠 다 열려잇으면, 그 때 MCU(CPU)가 interrupt vector로 점프한다. flag가 clear된다, flag가 0이 된다, interrupt routine이 수행 완료되면. 혹은, flag는 clear될 수 있다, 강제로, 그 해당 비트에 logical 1을 쓰면. 이 flag는 INT 0~7를 level interrupt로 설정하면 clear.

### 4) 코드

```
#include <avr/interrup.h>
ISR(INT7_vect){ //interrupt service routine
…….,,
}

int main() {
EICRB = (EICRB & 0x3f) | 0x80;
//1에서 0으로 떨어지게 할 때 되게하는 설정.
//b번의 6, 7번을 10으로 설정.
EiMSK |=0x80; //방문 엶.
EIFR=0xff; //clear함.
sei(); //현관문 엶. SERG|=0x80;랑 같음.
while(1); //wait for interrupt
}
```

//////////////과제5//////////////////////

## 16. Timer/Counter

**1) 개념**

avr은 독립적인 4개 내부 시계 있음.

시계는, digital hardware counter로, 일종의 register임. 0,1이 들어오면 그 register의 비트 하나 증가.

**(1) timer**

내부 clock(from MCU, 16MHz의 주기 신호), 빠름, 분주 가능, 동기 모드.

**(2) counter**

외부 clock, 느림, 분주불가능, 비동기모드, event counter.

**2) avr의 timer/counter**

8bit timer/counter0, 2 (이걸로 숙제)

16bit timer/counter1, 3.

- 각 시계는 독립된 prescaler, compare mode 갖고 있음.
- 0~255까지 셀 수 있음.
- single channel counter.
- clear time on compare match (auto reload) (비교해서 맞아떨어지면 timer를 0으로 클리어.)
- glitch-free, phase correct pulse width modulator (PWM; pulse width modulator. 주기의 폭 변화. base band를 고대로 주파수만 옮기는 게 modulation. am은 A sin(2pifct) 해서 크기만 변화시킴,노이즈 많음. fm은 V_FM(t) = A sin (2pi (fc + aVm(t) ) t)..... 전압 커지면 pulse 넓어짐. 주기는 고정, 한 주기당 얼마나 차지하느냐가 전력량 결정하고, 그 퍼센트를 duty cycle라 함. 쉬는 기간 없이 다 전달 되면 100퍼. timer 출력 단자의 켜져잇는 시간 꺼져잇는 시간을 duty cycle 조절해서하는거임.)
- frequency generator로 쓸 수 있음.
- 10bit의 clock prescaler (cpu가 16MHz라서, 그거보다 더 빠른 시계는 필요가 없음. 그래서 시계의 주기를 분주기로 늦춘다.)
- overflow, compare match에 대해 interrupt 걸 수 있다.
- 외부에 커패시터 달면 32kHz 달 수 있기도 함.

**3) register**

**(1) TCCRn**

control register. TCCR0~TCCR3 까지.

**(2) TCNTn**

counter

**(3) OCRn**

output compare register. 시계가 몇 초 지나면 알려주는 거.

**(4) TIFR**

timer interrupt flag register, 20초 지나면 interrupt 하는 거.

**(5) TIMSK**

timer interrupt mask register. interrupt를 살리고 죽이고.

**(6) SFIOR**

special function IO register

**4) Operation mode**

**(1) mode 0 : Normal**

control register(TCCRn)로. WGM1:0=00 설정. 외부 event or clock에 따라 0~255까지 세줌. 255가 0되는 그 순간을 0xFF, overflow interrupt를 걸어줘서 cpu한테 알려줌. flag가 올라가고 interrupt 걸림.

**(2) mode 2 : CTC** (clear time on compare match)

WGM1:0=10로 설정. 내가 정해놓은, OCRn까지 upcounter 로 셈. match interrupt를 걸어줌( OCFn=1) toggle mode(toggle은 0이면 1, 1이면 0. 현재의 상태 뒤집음. 일종의 not. 쭉 올라감. ocr값이 match 되면 toggl이 뒤집음.) 로 출력도 함.(COMn1:0 = 01) ocrn 바꾸면 pulse width 변화, 즉 PWM 가능.

**(3) mode 3 : Fast PWM**

WGM1:0=11로 설정. upcounter인데, ocrn 넣어주면 over flow interrupt, match interrupt 도 걸림. 0,2번모드 섞은거.

**(4) mode 1 : Phase correct PWN**

0~255~0~255이고, ocr 값 하고 match가 걸리면 match interrupt도 걸림. 3 2 1 0 1 2 3 에서만 overflow interrupt 일어나고 254 255 254에선 overflow interrrupt 안일어남.

### 5) operation mode 결정

#### (1) TCCR0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00 |
| Read/Write | W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

① WGM00,01 (6,3) : waveform generation mode

| Mode | WGM01[1] (CTC0) | WGM00[1] (PWM0) | Timer/Counter Mode of Operation |
|---|---|---|---|
| 0 | 0 | 0 | Normal |
| 1 | 0 | 1 | PWM, Phase Correct |
| 2 | 1 | 0 | CTC |
| 3 | 1 | 1 | Fast PWM |

00하면 0번모드 01하면 1번모드 10하면 2번모드 11하면 3번모드.

② COM01,00 : compare match output mode

01,하면 토글모드, 00 하면 nomal io port로 쓰겟다. 타이머 차단하고. mode마다 설정 다름.

③ CS02,01,00 : prescaing

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped) |
| 0 | 0 | 1 | $clk_{T0S}$ (No prescaling) |
| 0 | 1 | 0 | $clk_{T0S}/8$ |
| 0 | 1 | 1 | $clk_{T0S}/32$ |
| 1 | 0 | 0 | $clk_{T0S}/64$ |
| 1 | 0 | 1 | $clk_{T0S}/128$ |
| 1 | 1 | 0 | $clk_{T0S}/256$ |
| 1 | 1 | 1 | $clk_{T0S}/1024$ |

000이면 timer counter가 stop, disable됨. 입력 아예 차단. 001 010 011 등 각 값에 대해. 001은 direct하게 연결. 010하면 1/8니까 2MHz의 클럭임. 011하면 0.5MHz

④ FOC0 : force output compare

FOC0는 우리가 쓸 일 거의 없음. 하지 말기. cpu가 강제로 출력하고 싶을 때.

#### (2) TCCR2

① CS22, 21, 20

| CS22 | CS21 | CS20 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped) |
| 0 | 0 | 1 | $clk_{I/O}$/(No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}/8$ (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}/64$ (From prescaler) |
| 1 | 0 | 0 | $clk_{I/O}/256$ (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}/1024$ (From prescaler) |
| 1 | 1 | 0 | External clock source on T2 pin. Clock on falling edge |
| 1 | 1 | 1 | External clock source on T2 pin. Clock on rising edge |

### 6) ASSR

asynchronous status register

밑에 세 비트는 status 알려줌.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | AS0 | TCN0UB | OCR0UB | TCR0UB |
| Read/Write | R | R | R | R | R/W | R | R | R |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

① TCNT0

업데이트 되고있는지, 끝났는지 알려주는 거.

② AS0

counter mode, timer 모드 둘 다 가능.

시계로 쓴다는 건, 주기 안다는 뜻. 내부 시계 연결. clock source selection. 내부꺼 쓸까 외부꺼 쓸지 결정. 0이면 내부, 1이면 외부꺼(TOSC1에 연결된 핀). //우리는 0으로 설정할 것임. 어차피 디폴트라서 연결 필요x.

### 7) SFIOR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TSM | – | – | – | ACME | PUD | PSR0 | PSR321 | SFIOR |
| Read/Write | R/W | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

5가지 비트 쓸 수 있음. 타이머가 4개 있다 그랬지. 헤어지기 전에 시간 다 맞춰 놓는 거. 시계의 동기 맞춰주는거.

① TSM

1로 하면 네 개의 시계 동기시킴. 시작점 통일.

② PSR0

비동기 모드일 때 reset시키고 싶을 때. 16MHz 그거를 동기시킨다는 얘기임.

### 8) TIMSK

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 | TIMSK |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

①bit0 : overflow interrupt enable

②bit1 : output compare match interrupt enable

특정 타이머의 특정 interrupt 살릴까 죽일까.

### 9) TIFR

| OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 | TIFR |
|---|---|---|---|---|---|---|---|---|
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

①bit0 : overflow flag

②bit1 : output compare flag 0

10) 코드

```
#include <avr/io.h>
#include <avr/interrupt.h>
ISR(TIMER0_COMP_vect) //output compare match interrupt
{
....
}
ISR(TIMER0_OVF_vect) //timer overflow interrupt. 하고싶은 일 쓰기. 방열쇠 열기.
{
....
}
int main() {
TIMSK=.... //unmask timer interrupt
TIFR=... //clear interrupt flags

OCR0=.... //set OCR0(노멀모면 스킵)
ASSR=... //select the clock source
TCNT0=0; //clear counter
TCCR0=... //setup TImer0, timer begins!
sei(); //enable global interrupt

while(1)
{
....
}
}
```

```
#include <avr/io.h>
#include <avr/interrupt.h>

// global variables
char LEDdigit[4]={0,0,0,0};    // for LED display

// timer overflow interrupt
// 256*16 μs = 4.096 ms
ISR(TIMER0_OVF_vect)
{    static char number[10]={0x3f,0x06,0x5b,0x4f,0x66
                ,0x6d,0x7d,0x27,0x7f,0x67};
     static char index=0;
     index++;
     index&=0x3;    // 0,1,2,3
     PORTC= ~(1<<(3-index));
     PORTA= number[LEDdigit[index]];
}

void PrintLED(int num)   // binary to decimal, display
{    LEDdigit[3] = num/1000;
     num -= LEDdigit[3] *1000;
     LEDdigit[2]= num/100;
     num -= LEDdigit[2]*100;
     LEDdigit[1] = num/10;
     num -= LEDdigit[1]*10;
     LEDdigit[0] = num;
}

void main()
{    int count=0;

     DDRC=0x0f;          // 7-Seg. LED control
     DDRA=0xff;          // 7-Seg. LED data
     DDRG=0x03;          // bit 0,1 : outputs for other LEDs

     TCCR0=0b00000110;   // Normal mode, disable OC0, 1/256 (16 μs)
     TIMSK=0b00000001;   // unmask timer0 overflow interrupt
     sei();              // enable global interrupt

     while (1)
     {    if (PING&0x4)              // sw1-on?
          {    count++;
               PrintLED(count);
               while (PING&0x4);     // wait till sw1-off
          }
          if (PING&0x8)              // sw2-on?
          {    count = 0;
               PrintLED(count);
               while (PING&0x8);     // wait till sw2-off
          }
     }
}
```

7segment LED를 CPU에서 반복시키게 하지 말구 interrupt로 해서 하는 거 하자. CPU는 딴 일 하게! 정해진 시간 (4ms)가 되면 LED를 교체할거야. PrintLED 함수는 LEDdigit[i] 이거 값 바꾸는 일만 함.

//////////////과제6//////////////////////

## 17. USART

universal synchronous and asynchronous Receiver and Transmitter. 범용 동기,비동기 송수신기
컴퓨터 간 연결. 다른 컴과 통신할 수 있는 장치인 usart가 2개임. chain으로 해서 여러 컴 묶을 수 있게 하려고! pc통신. 다른 컴퓨터(server)와 데이터 주거니 받거니 com1, com2 그거임. 블루투스는 usart를 무선으로 바꾸는거.
1 giga lan : 1초에 100mega bit = 1억 bps를 전송. 1억 bit를 초당 전송.

### 1) Serial communication

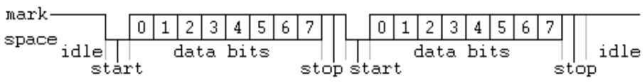
serial 직렬. data 1byte 보낼 때 한 줄로 순서대로 주루룩 보내서(직렬) 함. 장거리 통신 가능. 무선도 됨. (parallel은 전깃줄 여러 개 쓰는거. 여러 비트 동시. 현재는 안씀.)

### 2) duplex(양방향), simplex

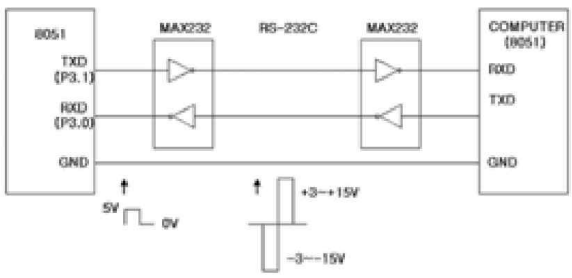
full duplex는 완벽한 양방향. 스마트폰, usart. (usart는 최소 필요한 선이 세 가닥: ground. 9핀 중 2,3,7번핀 씀. 나머진 modem, 동기식 등을 위한 것.)
half duplex는 양방향이긴 한데 동시에 양방향은 안됨. 무전기. walkie-talkie.
simplex는 애초에 단방향. 라디오 같은 거.

### 3) RS-232C 책

통신 protocol이라고도 함.




먼저 start bit를 보냄. 통신 안할 때는 high였다가 시작을 알르켜주기 위해 내려감, start bit. bit0부터 bit7까지. 끝나면 stopbit를 1로 하고 또 0으로 내림.



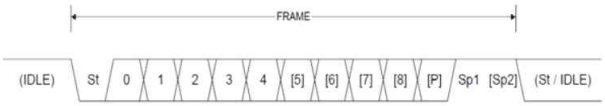

TXD transmit를 저쪽 컴엔 RXD에 연결. RXD receive가 저쪽 컴엔 TXD. 신호선을 규정된 걸로 변환해주는 MAX232 칩을 씀. avr보드에는 max232가 작게 뗌질되어있음. 0~5V 안쓰고, -3~3로 함. fax에도 이걸 설정하는 설정창 있음.

### 4) baud rate

bps안쓰고 baud rate란 단위 씀, usart에선.
baud는 8비트 외에 신호 시작인 start, stop, parity bit 이런거 까지 다 포함시켜서 초당 몇 비트인지를 의미. bps는 순수하게 데이터 비트.

### 5) com1 포트 설정


### 6) Frame format




idle : 놀고 있다.
start bit는 항상 0. 이걸로 시작. bit0부터 보냄. 1개 보낼수도 있고 8개 보낼수도 있고. 주로 8개 보내겠지.
parity bit를 선택적으로 넣거나 빼거나. (
parity엔 3가지 옵션, no, odd, even. 이걸 다 더했을 때 짝수 or 홀수일거임. odd parity는 0~7 다 더했을 때 그것이 홀수면 p = 0 해서 전체가 홀수가 되도록 함. even parity는 0~7 더했는데 1이면 p=1으로 해서 전체가 짝수 되게 함. no parity는 parity 칸이 아예 없음. 시간 아낄라고. 일반적으로 no parity임.)
stop bit는 한 프레임 끝남을 의미. high로 띄움. stop bit가 하나 or 두 개임. 주로 하나.
start~stop까지가 한 프레임.

### 7) usart의 data register

#### (1) UDR0 , UDR1

: receive/transmit data buffer

이 데이터에다가 receive/transmit data buffer 쓰면 TXBn 쓰고 (송신버퍼) 읽을 땐 RXBn 씀. (수신버퍼)

#### (2) UBRRnH,L

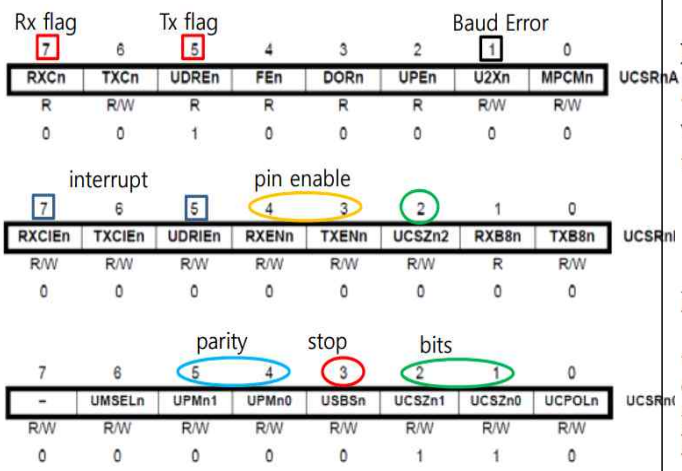
: baud rate registers

- UBRR=$f_{osc}$/(16*baud)-1 (U2X=0) or =$f_{osc}$/(8*baud)-1 (U2X=1)
- Baud : 300, 600, 1200, 2400, 4800, 9600, 14.4k, 19.2k, 38.4k, 57.6k, 76.8k, 115.2k, 230.4k, 250k, 0.5M, 1M

국제표준 중 맞출 수 있음. 우리 avr은 1M까지 됨. f_OSC 으로 해서 UBRR에 쓰면 됨. 12비트의 값을 써야되니까 2byte임, UBRR은. low byte는 UBRRnL에, high byte는 UBRRnH는 11:8. f_OSC/(16*baud)-1 값이 12bit임.

#### (3) UCSnA,B,C

: control and status register. n은 0or1.




//원 : control 값

초록 : Cregister의 1,2비트랑 B의 2 : bit 개수

파랑 : parity.

빨강 : stop bit. 1or2이니까.

노랑 : pin enable. 보통 때에는 io port로 씀. io port 안쓰고 통신 비트로 바꿀려는 거. transmit 살릴지, receive 살릴지. 리는 1로 해놔야 됨.

//네모 : status

Rx flag : cpu에게 데이터 들어오면 알려줌.

Tx flag : 내가 썼으면 여기다가 씀. 써도 되면 1. 통신은 cpu에 비해 워낙 느려서 이렇게 따로 해줌. 다 완료되면 Tx flag 듦.

Baud Error : baudrate 틀렸음을 알려줌.

interrupt는 통신 포트 당 3가지임. receive / transmit (2개) 완료, databuffer 비어있는.

### 8) 코드

함수 3 필요 : 통신포트 설정, 보내는거, 받는 거.

- **Initialize (set baud rate, frame format, enable TX and RX)**

```
#define F_CPU 16000000UL      // clock speed

void USART0_Init(long baud)
{    /* Set baud rate */
     unsigned int ubrr=
       (unsigned int)( (double)F_CPU/8/baud-1+0.5);
     UBRR0H = (unsigned char)(ubrr>>8);
     UBRR0L = (unsigned char)ubrr;

     UCSR0A |= (1<<U2X);

     /* Set frame format: 8-bit data, 1-stop bit, no
     parity, asynchronous */
     UCSR0C = 0b00000110;

     /* Enable receiver and transmitter */
     /* Disable interrupts */
     UCSR0B =0b00011000;
}
```

- **Transmit one byte**

```
void USART0_Transmit( unsigned char data )
{   // Wait for empty transmit buffer
    while ( !(UCSR0A & (1<<UDRE)) );

    // Put data into buffer and send the data
    UDR0 = data;
}
```

- **Poll the receive buffer**

```
char USART0_Poll( )
{    return (UCSR0A & (1<<RXC));
}
```

- **Receive one byte**

```
unsigned char USART0_Receive( )
{   // Wait for data to be received
    while ( !USART0_Poll() );

    // Get and return received data from buffer
    return UDR0;
}
```

PC를 serial로 연결하여 모니터를 출력, 키보드를 입력 장치로 사용할 수 있음. (Terminal)

PC에 serial port (COM1,2)가 없으면, Serial-to-USB cable을 이용할 것 (제어판에서 port number 확인)

PC에서 DOS 창의 C program과 유사

PC에서는 HyperTerminal 프로그램을 사용 (Vista에는 없음 → internet 검색) or PuTTY v0.6

**printf, scanf, getchar(), putchar()**의 함수들을 모두 사용할 수 있음. (*Wow!!!!!*)

Debugging에 매우 편리

**주의 사항** : 메모리 절약을 위해 %f (floating point)를 지원하지 않음. %f를 사용하려면,

Project → Configuration options → Custom options → Linker options 에 다음을 add

```
-Wl,-u,vfprintf -lprintf_flt -lm
-Wl,-u,vfscanf -lscanf_flt -lm
```

```
#include <stdio.h>

int USART0_putchar(char c, FILE *stream)
{     if (c=='₩n') USART0_putchar('₩r',stream);
      while ( !(UCSR0A & (1<<UDRE)) );
      UDR0 = c;
      return c;
}

int USART0_getchar(FILE *stream)
{     char c;
      while ( !(UCSR0A & (1<<RXC)) );
      c=UDR0;
      if (c=='₩r') USART0_putchar('₩n',stream);
      USART0_putchar(c, stream);       // echo
      return c;
}     // backspace 지원하지 않음.

int main()
{     FILE usart0 = FDEV_SETUP_STREAM(USAF
      stdin = stdout = &usart0;
      USART0_Init(9600);

      printf("Hello, world!₩n");
      ......
}
```

- **Receive Interrupt (overrun 방지) : AVR has two receive bu**

```
int RxIndex=0, RdIndex=0;
char RxBuffer[RXBUFSIZE];       // receive ring buffer

ISR(USART0_RX_vect)
{     RxBuffer[ RxIndex++ ] = UDR0;
      if (RxIndex == RXBUFSIZE) RxIndex=0;
}

int main( )
{     USART0_Init(9600);          ← UCSR0B =0b10011000;   (enable receive interrupt)
      sei();

      while (1)
      {       while (RxIndex != RdIndex)
              {       char c = RxBuffer[ RdIndex++ ];
                      if (RdIndex == RXBUFSIZE) RdIndex=0;
                      ......
              }
              .......
      }
}
```

### 9) 대량의 데이터를 serial 통신으로 전송,수신

- 규약 정하기

char code; short n; char data[n]; 등.

- Handshaking : 우리가 신호 보내면 저쪽에선 acknowloege 보내고. 일방적으로 보내면 확인할 방법이 없으므로. parity 방법 쓸 수도 있고. faximili는 handshaking 안함.

Example (host)

```
void Transmit(char code,short n,char *data)
{     USART0_Transmit(code);
      while (USART0_Receive()!=ACK);
      USART0_Transmit((char)(n&0xff));
      USART0_Transmit((char)(n>>8));
      for (i=0; i<n; i++) USART0_Transmit(data[i]);
      while (USART0_Receive()!=ACK);
}
void Receive(char code,short n,char *data)
{     USART0_Transmit(code);
      while (USART0_Receive()!=ACK);
      USART0_Transmit((char)(n&0xff));
      USART0_Transmit((char)(n>>8));
      for (i=0; i<n; i++) data[i]=USART0_Receive();
      USART0_Transmit(ACK);
}
```

Example (slave)

```
void Transmit(short n,char *data)
{     USART0_Transmit((char)(n&0xff));
      USART0_Transmit((char)(n>>8));
      for (i=0; i<n; i++) USART0_Receive(data[i]);
      while (USART0_Receive()!=ACK);
}
int Receive(char *data)
{     short n;
      n=(unsigned char)USART0_Receive();
      n+=(unsigned short)USART0_Receive()<<8;
      for (i=0; i<n; i++) data[i]=USART0_Receive();
      USART0_Transmit(ACK);
      return n;
}
```

## ///////////////과제4/////////////////////

```
#include  <avr/io.h>
int main() {
    unsigned char sw[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80 };
    DDRA = 0xFF;// LED는 출력 1111 1111
    DDRB = 0xFF;// 스위치는 입력 0000 0000
    DDRG = 0x03;// PORTG의 0:1 LED 출력, 2:3 스위치 입력 0011
    PORTA = 0x00; //LED가 모두 꺼지도록 초기화
    PORTG = 0x00; //LED가 모두 꺼지도록 초기화
    int state[8] = { 0 }; //led 상태 배열 선언
    int stateA = 0; //A의 상태표시 정수 선언
    int x = 0, y = 0;

    while (1) {
        for (int i = 0; i < 8; i++) {
            if (PINB & sw[i]) {//i번째 스위치가 눌러졌을 때
                if (state[i] == 0) {//led가 꺼져있는 상태라면
                    PORTA += sw[i]; }
                else {//led가 켜져있는 상태라면
                    PORTA -= sw[i]; }
            while (PINB & sw[i]); //i번째 스위치에서 손을 뗐을 때
            }
        if (PING & 0x4) {//스위치 a가 눌러졌을 때
            x = (int)PORTA;
            PORTA = 0x00; //PORTA 초기화
            PORTG = 0x00; //PORTG 초기화
            while (PING & 0X4);
            break; }  }

  while (1) {
        for (int i = 0; i < 8; i++) {//스위치 8개에 대해 반복
            if (PINB & sw[i]) { //i번째 스위치가 눌러졌을 때
                if (state[i] == 0) {//led가 꺼져있는 상태라면
                    PORTA += sw[i]; } //스위치 입력 상태 더함
                else {//led가 켜져있는 상태라면
                    PORTA -= sw[i]; } }//PORTA에 스위치 상태뺌.
            while (PINB & sw[i]); //i번째 스위치에서 손을 뗐을 때
        }

        if (PING & 0x8) { //스위치 b가 눌러졌을 때
            y = (int)PORTA;
            PORTA = 0x00; //PORTA 초기화
            PORTG = 0x00; //PORTG 초기화
            while (PING & 0X8);
            break; } }
    PORTA = (char)(x - y);
}
```

## ///////////////과제5/////////////////////

```
#include <avr/io.h> #include <avr/delay.h>
#include <avr/interrupt.h>
/* interrupt */
ISR(INT4_vect) { //E포트의 4번 스위치 10번이상누르면 초기화
            cntA++; if (cntA >= 10) cntA = 0; }
/* main 함수 */
void main() {
            DDRC = 0x0f; // 7segmentLED. latch 선택
            DDRA = 0xff; // 숫자데이터를 각 latch에 load함.
            DDRG = 0x03;
            DDRE = 0x00; //PORTE의 스위치는 입력(0)으로 선언.
            EICRB = 0xAA; //INT4~7 falling edge trigger 선언
            EIMSK |= 0xf0; //1111 0000 enable INT4~7
            EIFR = 0xff; //1111 1111 clear interuupt flag
            sei(); //enable global interrupt

            while (1) {
if (session == 0) PrintLED(cntA, cntB, cntC, cntD);
else if (session == 1) {
cntA = cntB = cntC = cntD = 0; check = 0; session = 2; }
else if (session == 2) PrintLED(cntA, cntB, cntC, cntD);

if (PING & 0x04) { // sw0이 눌리면
xnum1 = cntA; xnum2 = cntB; xnum3 = cntC; xnum4 = cntD;
//스위치가 눌려진 횟수에 따라 x값을 저장해둠.
session = 1; //세션 변경
while (PING & 0x04); // sw0가 눌러지는 동안
}

if (PING & 0x08) { // sw1이 눌리면
check++;
while (PING & 0x08); }// sw1가 눌러지는 동안

if (check == 0) {
ynum1 = cntA;ynum2 = cntB;ynum3 = cntC;ynum4 = cntD; }
//스위치가 눌려진 횟수에 따라 y값을 저장해둠.

x = xnum1 * 1000 + xnum2 * 100 + xnum3 * 10 + xnum4;
y = ynum1 * 1000 + ynum2 * 100 + ynum3 * 10 + ynum4;
if (check == 1) add(); if (check == 2) sub(); if (check == 3) mul();
if (check >= 4) dev();
} //while 괄호 닫기
} // void main() 괄호 닫기

void PrintLED(int cntA, int cntB, int cntC, int cntD) {
PORTC = ~0x01; //1111 1110
PORTA = number[cntA]; //left
_delay_ms(1);

PORTC = ~0x02; //1111 1101
PORTA = number[cntB];
_delay_ms(1);
PORTC = ~0x04; //1111 1011
PORTA = number[cntC];
if (check >= 4) PORTA |= 0x80; //1000 0000. 나눗셈이면, 점'.'을
표시
_delay_ms(1);
PORTC = ~0x08; //1111 0111
PORTA = number[cntD]; //right
_delay_ms(1);
}
```

## ///////////////과제6/////////////////////

```
/*타이머 interrupt*/
ISR(TIMER0_OVF_vect) {
static char number[10] = { 0x3f,0x06,.....0x67 }; //폰트
static char index = 0;
ndex++;
index &= 0x3; //0,1,2,3
PORTC = ~(1 << (3 - index));
PORTA = number[LEDdigit[index]];
}
ISR(TIMER2_COMP_vect) {
count++; //0.001s마다 count가 하나 증가
/*원하는 배속 설정 가능*/
if (count == 1000) { sec++; count = 0; } //1초마다 sec 하나 증가
ideal
if (sec == 60) { min++; sec = 0; }
if (min == 60) { hour++; min = 0; }
if (hour == 24) { day++; hour = 0; }
if (month == 13) { yun++; month = 1; }
if (yun == 4) { yun = 0; }
}

/*메인 함수*/
int main() {
DDRC = 0x0f;
DDRA = 0xff;
DDRG = 0x03;
TCCR0 = 0b00000110; //TIMER0 Normal mode (256 prescale)
TCCR2 = 0b00001011; //TIMER2 CTC mode (64 prescale, 4us)
OCR2 = 249; // 2번 타이머를 0~249번까지 셈. ( 250*4us =  1000us
= 0.001s )
TCNT2 = 0; // 2번 타이머를 초기화
TIMSK = 0x81; // 0번 타이머를 overflow, 2번 타이머를 CTC로 사용
하도록 unmask
sei(); // enable global interrupt

while (1) {
if (PING & 0x04) { //s2가 눌리면 다음 세션으로 넘어간다.
session++;
if (session == 5) session = 0;
while (PING & 0x04);
}
if (session == 0) { // 분 설정
if (PING & 0x08){
min++;
while (PING & 0x08);}
printTIME(hour, min);}
}

void printTIME(int hour, int min) { // 시각, 분 출력 함수
            LEDdigit[3] = hour / 10;
            LEDdigit[2] = hour % 10;
            LEDdigit[1] = min / 10;
            LEDdigit[0] = min % 10; }
```

## ///////////////과제7/////////////////////

```
#include <stdio.h> #include <avr/io.h> #include <string.h>
#define F_CPU 16000000UL
void main() {
    FILE usart0 = FDEV_SETUP_STREAM(USART0_putchar,
USART0_getchar, _FDEV_SETUP_RW);
    stdin = stdout = &usart0;
    USART0_Init(19200);
//baudrate를 내 컴퓨터와 avr 통신이 가장 빠르게 된다.
    while (1)    {
volatile float x=0;
printf("x를 입력하세요? : ");
scanf("%f", &x);
.....
    } //while문 닫기
}//main함수 닫기

void USART0_Init(int baud)
{   // set baud rate
    unsigned int ubrr = (unsigned int)( (double) F_CPU / 8 /
baud - 1 + 0.5 );
    UBRR0H = (unsigned char)(ubrr >> 8);
    UBRR0L = (unsigned char)ubrr;
    // set frame format : 8-bit data, 1-stop bit
    // no parity, asynchronous
            UCSR0A |= (1<<U2X);
    UCSR0C = 0b00000110;
    //enable receiver and transmitter
    //disable interrupts
    UCSR0B = 0b00011000;
}

int USART0_putchar(char c, FILE* stream)
{
    if (c == '\n') USART0_putchar('\r', stream);
    while (!(UCSR0A & (1 << UDRE)));
    UDR0 = c;
    return c;
}

int USART0_getchar(FILE* stream)
{
    char c;
    while (!(UCSR0A & (1 << RXC)));
    c = UDR0;
    if (c == '\r') USART0_putchar('\n', stream);
    USART0_putchar(c, stream);
    return c;
}
```