AVR - HW3

임베디드스쿨1기

Lv1과정

2020. 09. 25

박하늘

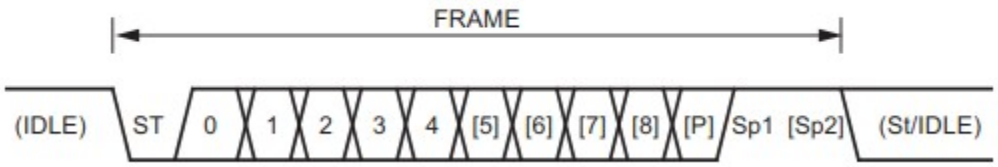
1. UART(Universal Synchronus Receiver Transmit)

1) 특징

- 1:1 시리얼 직렬 통신, RX 와 TX가 서로 교차하여 연결

- 1 바이트 정보를 1비트 단위로 나눠서 전송
- 보드- 보드, 장치- 장치 사이의 통신에 주로 사용 (RS232, RS422, RS485 전기적전송 규격)
- MCU와 PC사이의 통신에 사용
- 속도, 데이터 길이, 패리티, 정지 비트 (9600, 8, N, 1)
  예) 9600bps: 1초에 9600칸 or bit를 보낼 수 있음

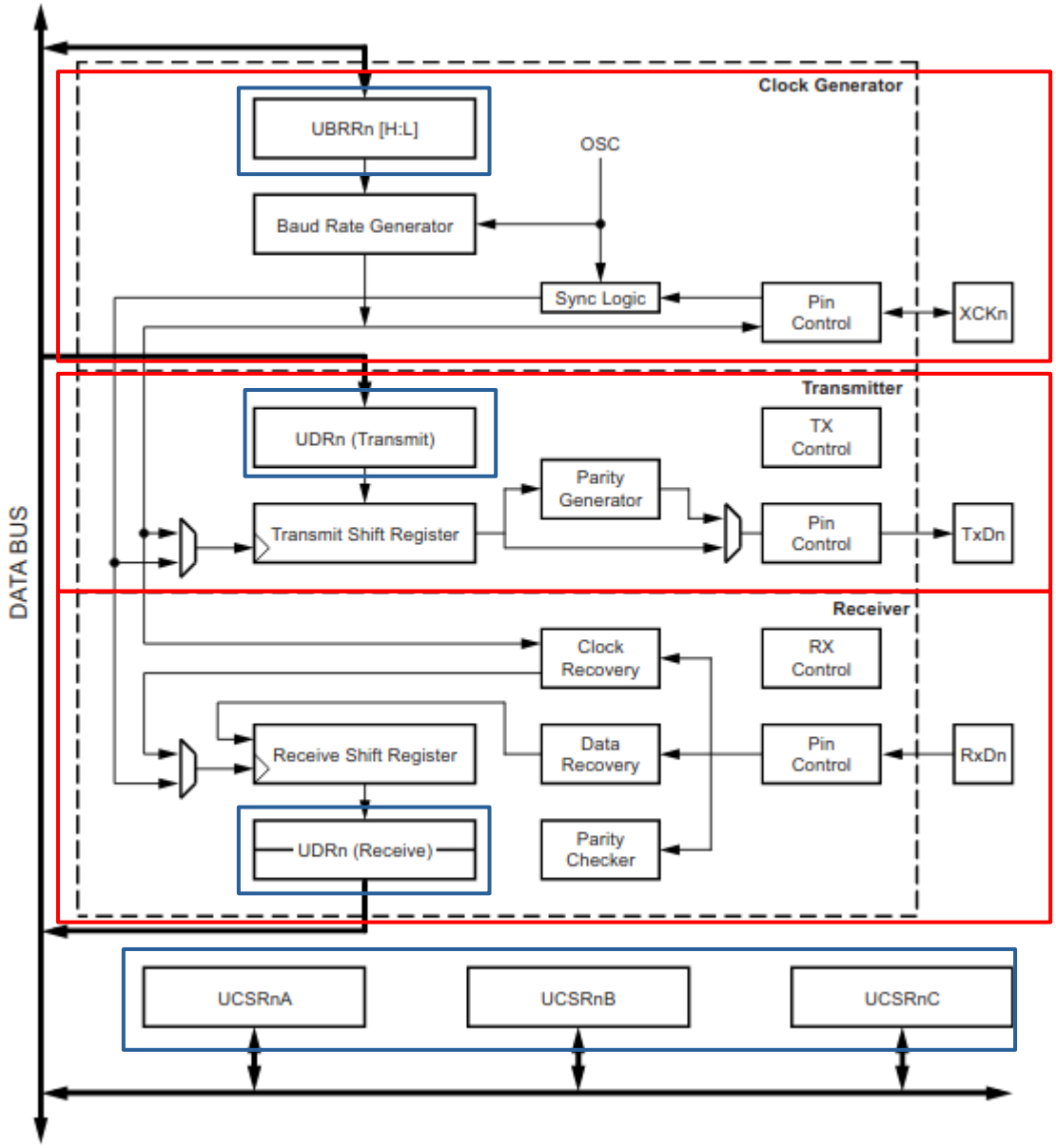
Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn).

1. UART Block Diagram

1) UART Register : UDRn, UBRRn, UCSRnA, UCSRnB, USCRnC
2) 클록 발생부, 송신부, 수신부로 나뉨



포기하면 얻는 건 아무것도 없다.

2. UART 레지스터 셋팅

1) UDR0 (USART Data Register 0)

:송신 버퍼인 TXB와 수신 버퍼와 RXB가 UDR0이라는 공통의 이름으로 사용

:UDR0 레지스터에 데이터를 쓰면 TXB에 기록, 데이터를 읽으면 RXB의 값 읽힘

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
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	

2. UART 레지스터 셋팅

2) UCSR0A (UART Control and Status Register 0A)

: UART 통신의 제어와 상태 확인을 위한 레지스터

- 2배속 설정: U2X0비트
- 송신 가능 상태 확인: UDRE0비트
- 수신 완료 상태 확인 : RXC0비트

UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

비트	이름	설명
7	RXC0	Receive Complete: 수신 버퍼(UDR0)에 읽지 않은 문자가 있을 때는 1이 되고 버퍼가 비어 있을 때는 0이 된다. UCSR0B 레지스터의 RXCIE0 비트와 함께 사용되어 수신 완료 인터럽트를 발생시킬 수 있다.
6	TXC0	Transmit Complete: 시프트 레지스터의 데이터가 송신되고 송신 버퍼(UDR0)가 비어 있을 때 1이 된다. UCSR0B 레지스터의 TXCIE0 비트와 함께 사용되어 송신 완료 인터럽트를 발생시킬 수 있다.
5	UDRE0	USART Data Register Empty: 송신 버퍼(UDR0)가 비어 있어 데이터를 받을 준비가 되어 있을 때 1이 된다. UCSR0B 레지스터의 UDRIE0 비트와 함께 사용되어 송신 데이터 레지스터 준비 완료 인터럽트를 발생시킬 수 있다.
4	FE0	Frame Error: 수신 데이터가 없는 경우 수신 값은 HIGH 상태이며 LOW 상태로 바뀌면서 데이터 프레임 수신이 시작되고 프레임의 마지막 정비 비트 수신 시에 다시 HIGH 상태로 바뀐다. 마지막에 HIGH 상태를 수신하지 못하여 프레임 수신에 오류가 발생하면 FE0은 1의 값을 가진다.
3	DOR0	Data Overrun Error: 수신 버퍼가 가득 찬 상태에서 수신 시프트 레지스터에 새로운 문자가 수신 완료되고 다시 그 다음 문자의 시작 비트가 검출되는 오버런 상황이 발생한 경우 1의 값을 가진다.
2	UPE0	USART Parity Error: 수신 버퍼에 저장된 문자 데이터에 패리티 오류가 발생한 경우 1의 값을 가진다. 패리티 비트 사용이 설정된 경우에만 사용할 수 있다.
1	U2X0	USART Double Transmission Speed: 비동기 전송 모드에서만 사용되며, 2배속 모드이면 1, 1배속 모드이면 0의 값을 가진다.
0	MPCM0	1개의 마스터 프로세서가 여러 개의 슬레이브 프로세서에 특정 어드레스를 전송함으로써 1개의 슬레이브만을 지정하여 데이터를 전송하는 멀티프로세서 통신 모드에서 1의 값을 가진다

전송버퍼가 비어 있다면 1값을 가지고 새로운 데이터를 받아서 전송할 준비 완료된 것임

수신버퍼가 비어 있다면 0으로 클리어, 1이 되면 데이터 수신 받았다는 것임

1. UART 레지스터 셋팅

3) UCSR0B (USART Control and Status Register 0B)

: UART 통신의 송신 및 수신을 가능하게 하기 위해 사용

- 송신 가능 설정 비트: RXEN0
- 수신 가능 설정 비트 : TXEN0
- 디폴트로 송수신은 금지되어 있음

```
UCSR0B |= _BV(RXEN0);                    // 수신 가능
UCSR0B |= _BV(TXEN0);                    // 송신 가능
```

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
UCSR0B=(1<<RXEN0)|(1<<TXEN0);
```

1. UART 레지스터 셋팅

4) UCSR0C (USART Control and Status Register 0C)

: 통신 방법 및 데이터 형식 지정

- UMSEL0n : 통신 모드 설정
- UPM0n : 패리티 비트 설정
- USBS0 : 정지 비트 설정 (0: 1비트, 1: 2비트)
- UCSZ0n : 데이터 비트 수 설정 (UCSR0B레지스터와 함께 사용)
- UCPOL0 : 클록 극성 설정 (동기 모드에서만 사용)

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

UMSEL01	UMSEL00	모드
0	0	비동기 USART
0	1	동기 USART
1	0	-
1	1	마스터 SPI 모드

UPM01	UPM00	패리티 비트
0	0	사용 안 함
0	1	-
1	0	짝수 패리티
1	1	홀수 패리티

UCSR0B 레지스터	UCSR0C 레지스터		데이터 비트 수
UCSZ02	UCSZ01	UCSZ00	
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	0	0	–
1	0	1	–
1	1	0	–
1	1	1	9

```
UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01); //Charecter size : 8비트

UCSR0C &= ~(1<<USBS0); //stop  bit : 1비트

UCSR0C &= ~((1<<UPM01)|(1<<UPM00)); // no parity mode
```

1. UART 레지스터 셋팅

5) UBRR0H, UBRR0L (USART Baud Rate Register)

: 8bit 레지스터 조합에 의한 가상 16bit 레지스터 (12bit로 보율 결정함)

: 보율에 따른 레지스터 계산값은 정수가 아니므로 근사값으로 설정

- 약간의 오차 발생
- 비트 데이터 확인을 위해 여러번의 샘플링 필요 (2배속 모드에서는 횟수 절반임)

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

	UBRR0 실제계산값		UBRR0 사용값	
보율	1배속	2배속	1배속	2배속
1200	832.33	1665.67	832	1666
2400	415.67	832.33	416	832
4800	207.33	415.67	207	416
9600	103.17	207.33	103	207
14400	68.44	137.89	68	138
19200	51.08	103.17	51	103
28800	33.72	68.44	34	68
38400	25.04	51.08	25	51
57600	16.36	33.72	16	34
115200	7.68	16.36	8	16

```
UBRR0H = (uint8_t)(UBRR_VALUE>>8);

UBRR0L = (uint8_t) UBRR_VALUE;
```

1. USART 통신 속도 계산

1) 시스템 클록과 원하는 Baud rate를 이용하여 UBRR(USART Baud Rate Register)를 계산한다.

BAUD = fosc / 16 * (UBRR + 1)

(참고)

BAUD Baud rate (in bits per second, bps)

fOSC System Oscillator clock frequency

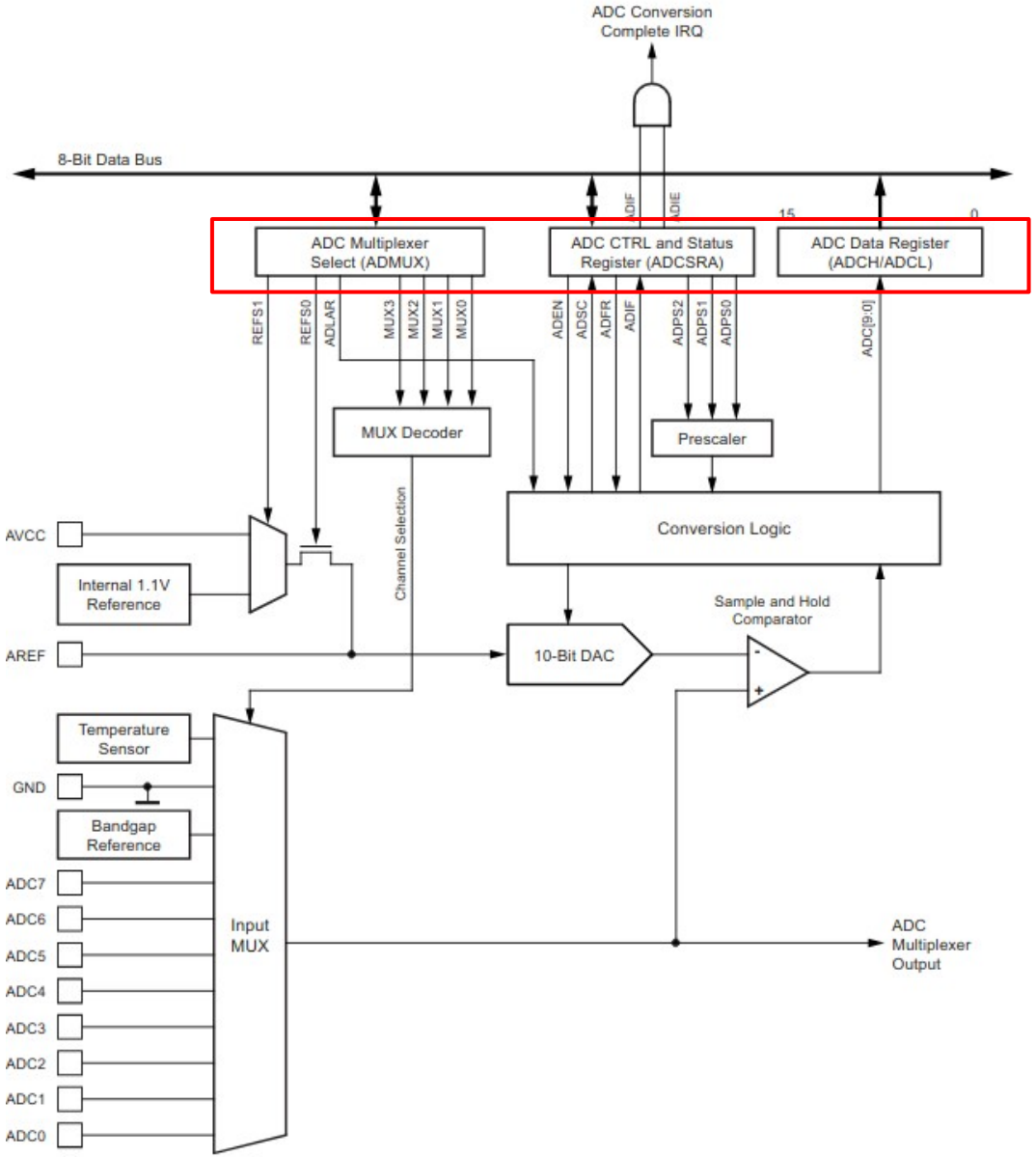
UBRR Contents of the UBRRH and UBRRL Registers, (0 – 4095)

(코드)

```
UBRR_VALUE = (((F_CPU / (9600 * 16UL))) - 1);
```

2. ADC(Analog Digital Converter) Block Schematic

1) ADC Register : ADMUX, ADCSRA, ADCH/ADCL



2. ADC 레지스터 셋팅

1) ADCSRA (ADC Control and Status Register A)

: AD 변환 상태 및 변환 과정 제어

: 분주율

- ADPSn(ADC Prescaler Selection) 비트로 설정
- 16MHz 시스템 클록을 나누는 값으로 시스템 클록 보다 낮은 속도로 ADC수행
- 50~200KHz 범위 추천

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

비트	이름	설명
7	ADEN	ADC Enable: ADC를 활성화시킨다.
6	ADSC	ADC Start Conversion: AD 변환을 시작한다.
5	ADATE	ADC Auto Trigger Enable: 선택된 트리거 신호에 의해 변환이 시작되도록 설정한다. 트리거 신호 선택은 ADCSRB 레지스터 설정에 따른다.
4	ADIF	ADC Interrupt Flag: AD 변환이 종료되었음을 알리는 플래그이다.
3	ADIE	ADC Interrupt Enable: AD 변환이 종료되었을 때 인터럽트 발생을 허용한다.
2	ADPS2	ADC Prescaler Select: ADC를 위한 분주율을 설정한다.
1	ADPS1	
0	ADPS0	

ADPS2	ADPS1	ADPS0	분주율
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

2. ADC 레지스터 셋팅

2) ADCSRB

: AD 변환을 위한 트리거 소스 선택

- 디폴트값은 프리러닝 모드, 일반적임
- ADCSRA레지스터의 ADATE비트에 따라 단일 변환모드(=0)와 프리러닝모드(=1)로 설정해 사용

(단일변환모드: 한번의 AD변환후 멈춤, 프리러닝모드: AD변환이 완료되면 다음 AD변환 자동시작)

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADTS2	ADTS1	ADTS0	트리거 소스
0	0	0	프리러닝 모드
0	0	1	아날로그 비교기
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

2. ADC 레지스터 셋팅

3) ADMUX

: 기준 전압 선택, 입력 채널 선택, ADC레지스터 정렬 방식 지정

- REFSn: 기준 전압 선택
- MUXn: 입력 채널 선택
- ADLAR: 정렬 방식(1: 왼쪽정렬, 0: 오른쪽정렬) : ADCL부터 읽음

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

REFS1	REFS0	설명
0	0	외부 AREF 핀 입력을 기준 전압으로 사용한다.
0	1	외부 AVCC 핀 입력을 기준 전압으로 사용한다.
1	0	–
1	1	내부 1.1V를 기준 전압으로 사용한다.

(AVCC는 VCC와 +-0.3V를 넘지 않아야 한다.)

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

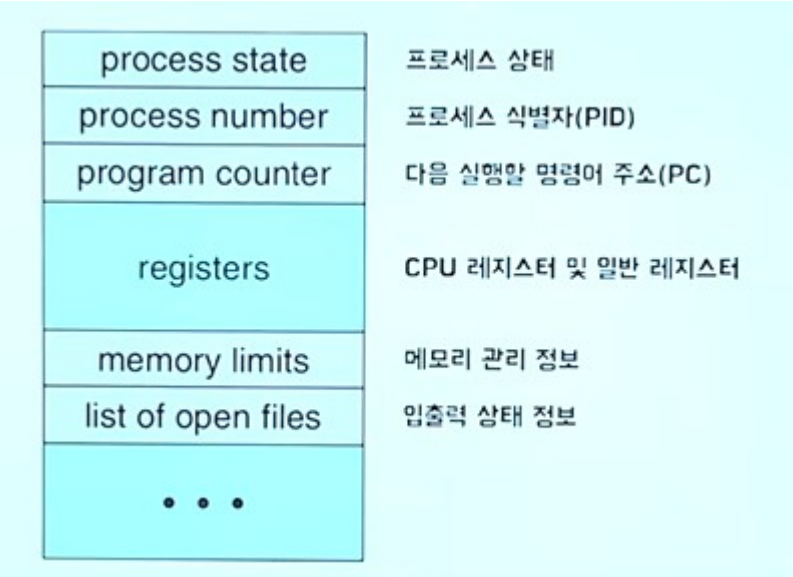
ADLAR = 1

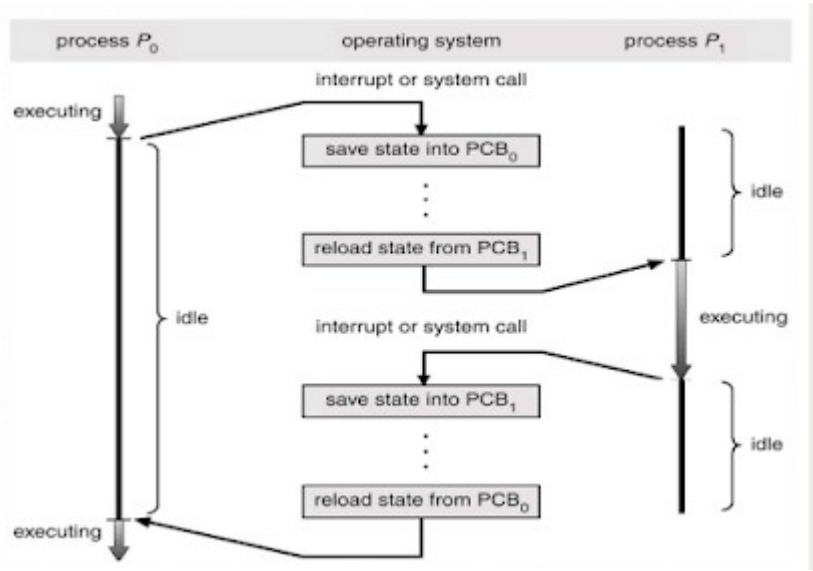
Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

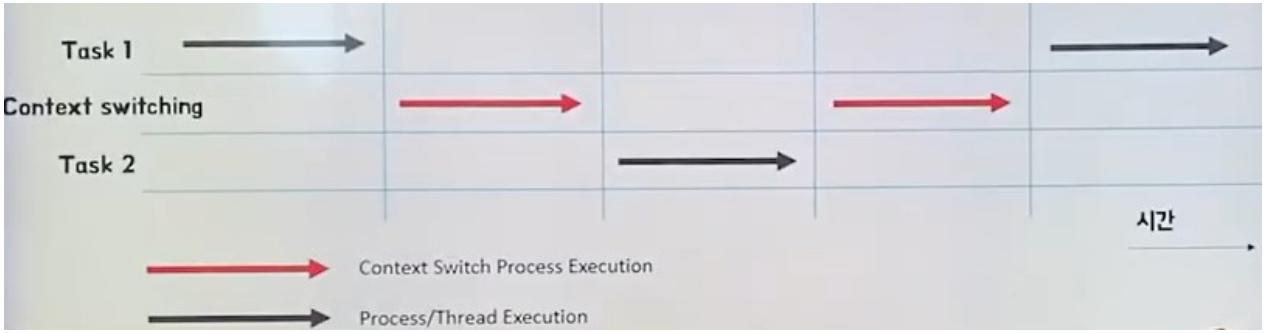
3. Context Switching

1) Context Switching이란?

: 하나의 프로세스가 CPU를 사용중인 상태에서 다른 프로세스가 CPU 를 사용하도록 하기 위해, 이전의 프로세스의 상태(문맥)을 보관하고 새로운 프로세스의 상태를 적재하는 작업을 말한다. 한 프로세스의 문맥은 그 프로세스의 PCB(Process Control Block)에 기록되어 있다. - wiki





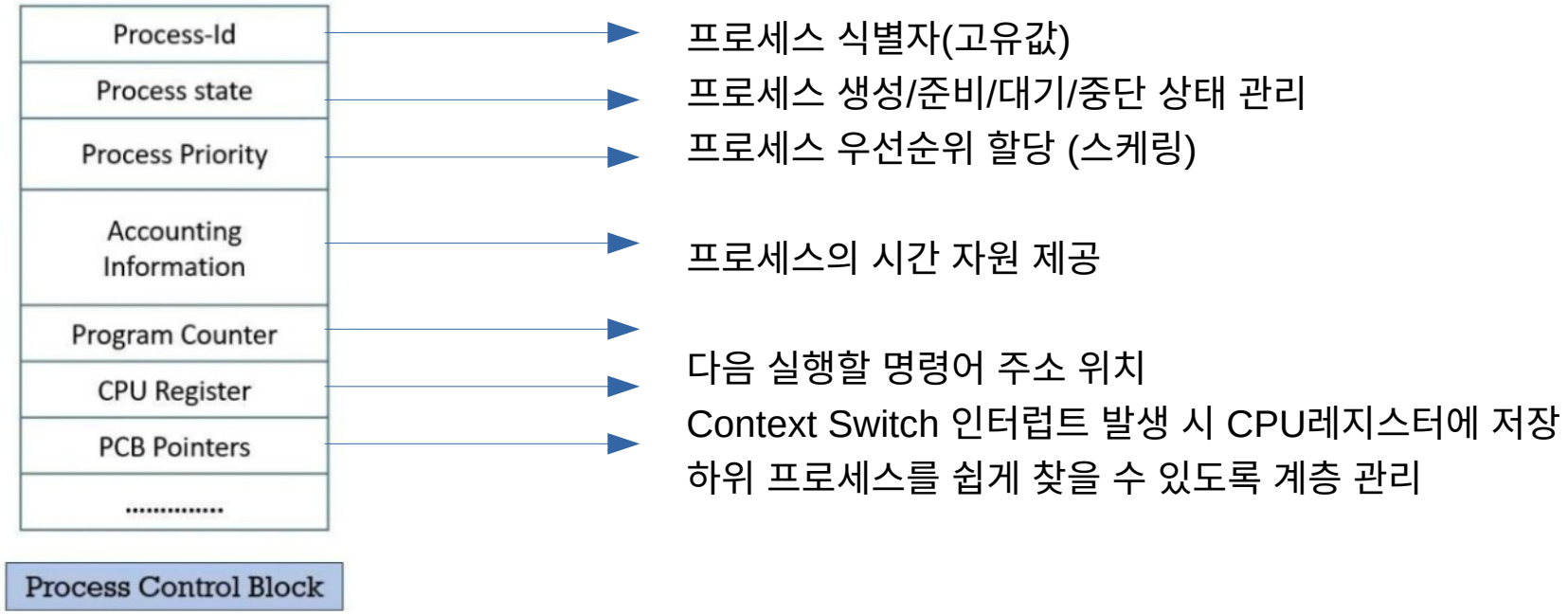


*state: CPU의 여러 레지스터들
*save와 reload사이에 interrupt를 처리
*idle상태에는 overhead발생

4. PCB

1) PCB(Process Control Block)

1. 특정한 프로세스를 관리할 필요가 있는 정보를 포함하는 운영 체제 커널의 자료 구조
2. PCB는 프로세스의 중요한 정보를 데이터 블록 형태로 관리하고 있으며 보호된 메모리 영역 안에 존재
3. 특징: 운영체제 입장에서 프로세스 상태 관리와 문맥교환(Context switch)에 필요하며 프로세스 생성시 만들어지고 주기억장치에 유지됨



5. Process

1) Program

: (실행되기 전 상태의)명령어, 코드 및 정적인 데이터의 묶음

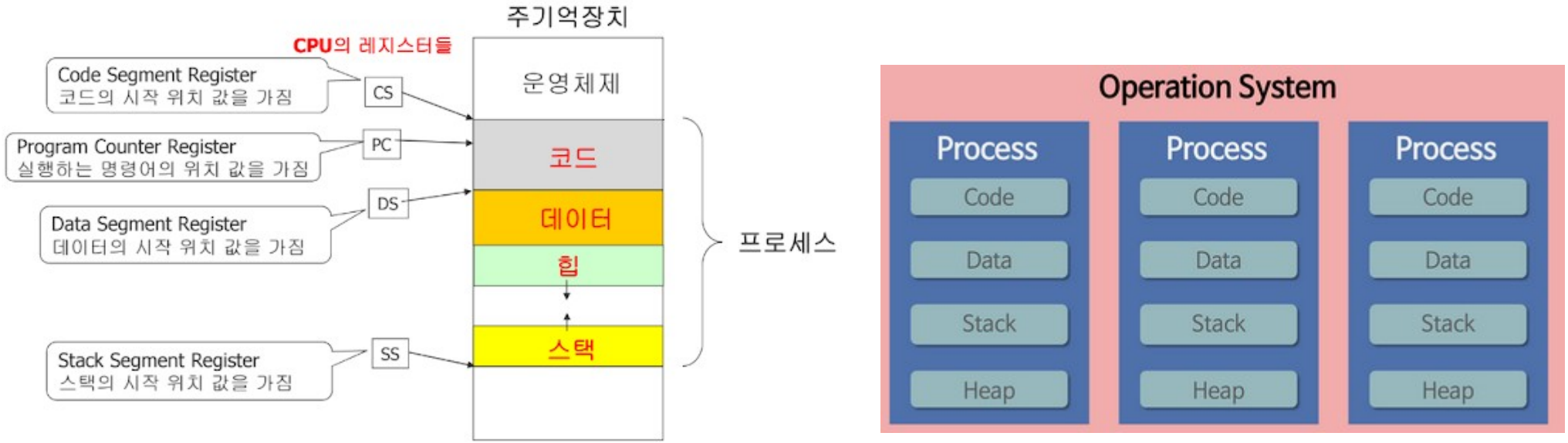
2) Process

: 실행 중인 Program

: 운영체제로부터 시스템 자원을 할당 받는 작업의 단위

→ CPU(프로세서)는 한순간에 하나의 프로세스만 실행할 수 있음

→ 운영체제가 짧은 시간 여러 프로세스를 교체하고 있어 동시에 여러개 작업 실행되는 것처럼 느껴짐

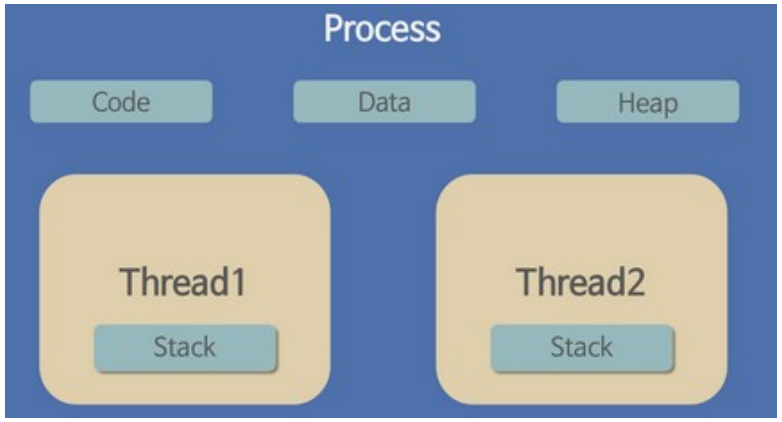


[프로세스 구조]

6. Thread & Multi Thread

4) Thread

- 프로세스 내에서 실행되는 흐름의 단위
- CPU 이용의 기본 단위
- Text, data, heap 영역을 공유
- 각 thread는 별도의 stack 영역을 가짐



5) Multi Thread

- 프로세스의 자원을 공유
- 향상된 응답성
- context switching 비용이 적음
- 자원을 공유하는 만큼 충돌을 주의 (Thread-safe) ex) web server

6) Multi Process

-하나의 작업을 여러 개의 프로세스가 처리
- 프로세스간 통신(IPC: Interprocess communication)
- Context switching 비용이 큼
- 자식 프로세스 중 하나가 문제가 생겨도 다른 프로세스에 영향이 없음 ex) Google Chrome

→ 멀티스레드를 사용하는 이유: 프로세스를 생성하여 자원을 할당하는 시스템 콜이 줄어들어 자원을 효율적으로 관리 가능 (프로세스간 context switching 시 단순히 CPU 레지스터 교체 뿐만 아니라 RAM과 CPU 사이의 캐쉬메모리에 대한 데이터까지 초기화 되므로 오버헤드가 크기 때문)

감사합니다.