C basic language

임베디드스쿨 2기

Lv1과정

2021. 05. 14

김효창

명령어

AVRDUDE는 AVR 마이크로 컨트롤러의 ROM 및 EEPROM 컨텐츠를 다운로드/업로드/조작 등을 도와주는 유틸리티

Atmel Studio & CodevisonAVR은 Linux 환경을 지원하지 않으므로,
Linux 환경에서 Hex 파일을 AVR microcontrollers에 다운로드 하기 위해 사용

sudo avrdude -c avrisp2 -p m328p -U flash:w:blink_test.hex

sudo : 임시 관리자 권한 접근
avrdude : AVR 의 ROM 및 EEPROM 접근
-c : 프로그래머
avrisp2 : 사용 중인 프로그래머 이름
-p : 부품 part 은 atmega328p (m328p)
m328p : 사용 중인 board
-U : 〈filename〉 을 읽기/쓰기/확인
프로그램 해야 하는 메모리는 플래쉬(flash) : 쓰기(w): 파일명은 ㅇㅇ.hex
blink_test.hex : hex 파일 이름

avrdude 매뉴얼 참고..

PORTB / DDRB / PINB

13.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
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	

13.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
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	

13.4.4 PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

SREG (status Register)

6.3.1 SREG – AVR Status Register

The AVR status register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
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	

- **Bit 7 – I: Global Interrupt Enable**

The global interrupt enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the global interrupt enable register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

용어 정리

PORTB

0 → output 0 [V] (Low)
1 → output 5 [V] (High)
PORT 는 출력 값으로 사용 (PIN 은 입력 값으로 사용)

DDRB

0 → Input
1 → Output
각 Bit 의 입력 또는 출력 모드를 설정

PINB

PIN 으로부터 데이터를 입력 받는 레지스터
DDRB 가 입력으로 설정되어 있으면 High / Low 상태를 읽는다.

용어 정리

SREG (status Register)

인터럽트를 활성화하려면 global interrupt Enable 를 (1) 로 설정해야 한다.

Global Interrupt Enable → sei ( ); , SREG |= (1 << 7)
Global Interrupt Disable → cli ( ) ; , SREG |= (0 << 7)

전역 인터럽트 활성화 레지스터를 지운 경우 개별 인터럽트 활성화 설정과 관계없이 모든 인터럽트가 활성화되지 않는다.

외부 인터럽트는 입출력 포트의 기능을 사용하는 것이 아니므로 DDR 레지스터는 설정하지 않는다.

Polling

부모님이 오늘 집에 오기로 했는데 도착하기 전까지 수시로 집 밖으로 나가서 확인하는 것을 반복.

Interrupt

부모님이 오늘 집에 오기로 했는데 집 안에서 게임하다가 부모님이 도착해서 초인종을 눌렀을 때 밖으로 나가서 확인.

용어 정리

인터럽트는 스택 함수 호출과 비슷하다.

-일반적인 작업(task) 실행
-인터럽트 발생(요청)
-수행 작업 정지 (복귀할 주소 값 저장)
-인터럽트 벡터 테이블 → ISR 주소 값으로 점프
-인터럽트 서비스 루틴 실행
-ISR 종료 후 복귀
-멈췄던 작업 실행

S/W 인터럽트 : 마이크로프로세서 내부에서 발생 , 타이머 / 카운터 등
H/W 인터럽트 : 마이크로프로세서 외부에서 발생 , 스위치 / 센서 등

인터럽트 사용 시 코드에 #include 〈avr/interrupt.h〉 선언

인터럽트 서비스 루틴은 가능하면 변수 값 처리만 하도록 권장.

```
ISR ( OO_vector )
{
      인터럽트 서비스 루틴 코드 삽입
}
```

AND / OR / XOR / NOT

AND

스위치 또는 센서의 상태를 입력 처리할 때 사용
Bit 값을 0 으로 출력할 때 사용

OR

시프트 연산 또는 출력 처리할 때 사용

XOR

반전 출력할 때 사용
Bit 가 같다 = 0 , 다르다 = 1

Delay 사용 방법

Features

- High performance, low power AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single clock cycle execution
  - 32 × 8 general purpose working registers
  - Fully static operation
  - Up to 16MIPS throughput at 16MHz
  - On-chip 2-cycle multiplier
- Speed grade:
  - 0 to 8MHz at 2.7 to 5.5V (automotive temperature range: –40°C to +125°C)
  - 0 to 16MHz at 4.5 to 5.5V (automotive temperature range: –40°C to +125°C)

8.8 External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 8-4. To run the device on an external clock, the CKSEL fuses must be programmed to “0000” (see Table 8-14).

Table 8-14. Crystal Oscillator Clock Frequency

Frequency	CKSEL3..0
0 to 16MHz	0000

Delay 사용 방법

28.3 DC Characteristics

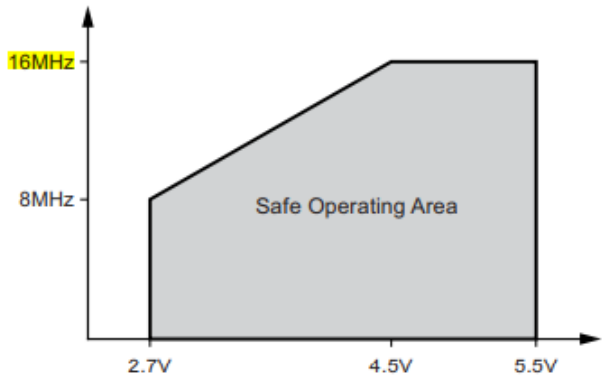
T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Parameter	Condition	Symbol	Min.	Typ.(2)	Max.	Units
Power supply current(1)	Active 4MHz, V_{CC} = 3V	I_{CC}		1.5	2.4	mA
	Active 8MHz, V_{CC} = 5V			5.2	10	mA
	Active 16MHz, V_{CC} = 5V			9.2	14	mA
	Idle 4MHz, V_{CC} = 3V			0.25	0.6	mA
	Idle 8MHz, V_{CC} = 5V			1.0	1.6	mA
	Idle 16MHz, V_{CC} = 5V			1.9	2.8	mA
Power-down mode(3)	WDT enabled, V_{CC} = 3V				44	µA
	WDT enabled, V_{CC} = 5V				66	µA
	WDT disabled, V_{CC} = 3V				40	µA
	WDT disabled, V_{CC} = 5V				60	µA

Notes:
1. Values with Section 9.10 “Minimizing Power Consumption” on page 36 enabled (0xFF).
2. Typical values at 25°C.
3. The current consumption values include input leakage current.

28.4 Speed Grades

Figure 28-1. Maximum Frequency



Delay 사용 방법

#include <util/delay.h>

F_CPU 정의되지 않으면 경고 발생.

외부 크리스탈과 동일한 값을 F_CPU 에 define 해야 한다.

define F_CPU 16000000L
: 16 MHz 사용 , L 은 자료형 long 4 Byte 의미 16 / 00 / 00 / 00

_delay_ms(2500);
: 2.5초 지연 , us 도 사용 가능

기타

while (1) ; while (1) 다음 세미콜론 필수 , 중괄호 작성 시는 세미콜론 생략.
(조건식) : 0 = 실행 중지, 1 = 계속 수행.

volatile : 인터럽트 서비스 루틴에서 사용되는 변수를 선언할 때 사용
volatile unsinged char switch;

unsigned 는 양수 , 0 만을 가진다. 0 ~ 255 사이의 값
음수를 표현하기 위해서는 unsigned 를 사용하지 않는다.

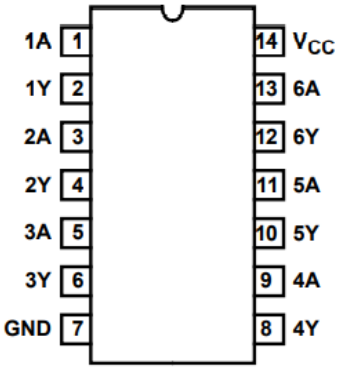
74HC14

74HC14 는 74HC04 + Schmitt Trigger 를 추가한 6 개의 게이트가 들어 있는 회로

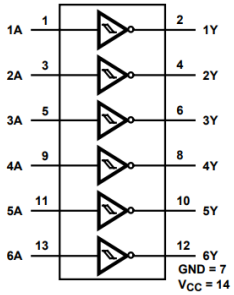
Schmitt Trigger

- 입력 값의 노이즈 때문에 출력 값이 수시로 변화한다.
- 상위 값 4V , 하위 값 3V 가정하면 4V 가 초과 되어야 High 상태가 되며 3V 미만일 때 Low 상태가 된다. (3.9 , 3.7 , 3.5 , 3.3V 등으로 감소해도 High 상태를 유지한다.)
- 상위 값과 하위 값의 차를 Hysteresis 전압이라고 부른다.
- 사인파를 구형파 상태로 만든다.

CD54HC14, CD54HCT14
(CERDIP)
CD74HC14, CD74HCT14
(PDIP, SOIC, TSSOP)
TOP VIEW



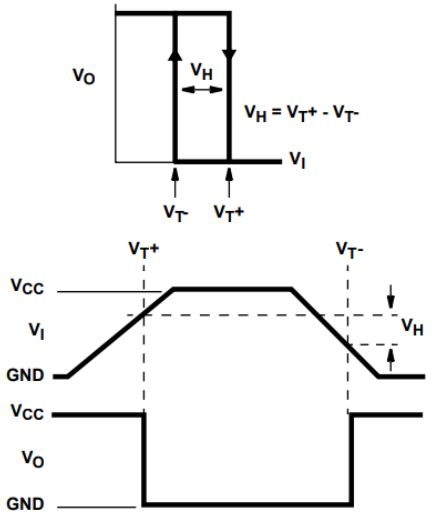
Functional Diagram



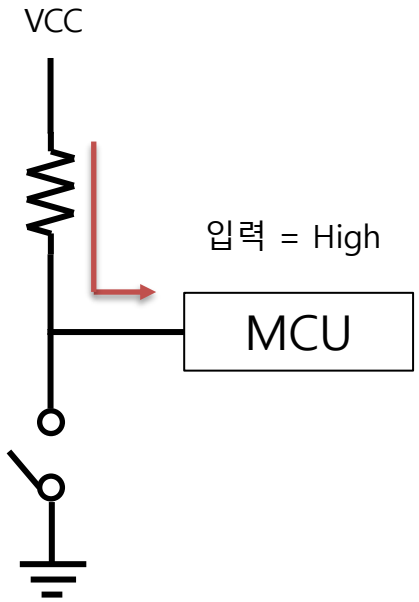
TRUTH TABLE

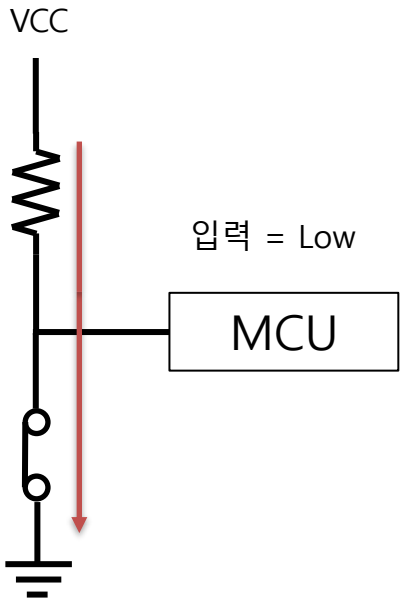
INPUT (A)	OUTPUT (Y)
L	H
H	L

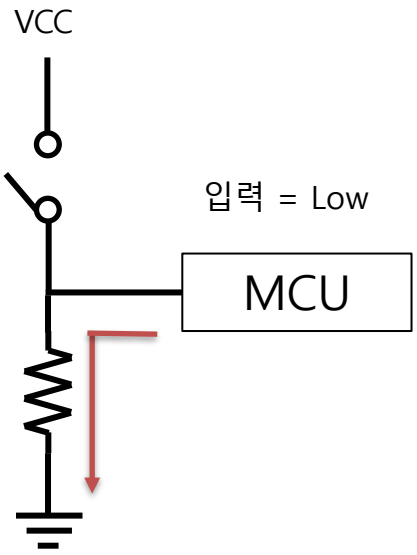
H= High Level
L= Low Level

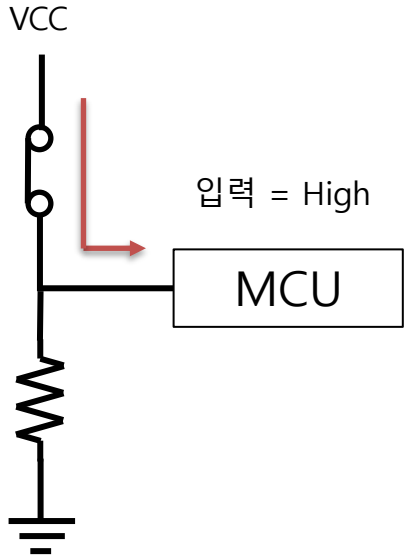


MCU 상태









Pin Change Interrupt Control Register

12.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..3 - Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

- **Bit 2 - PCIE2: Pin Change Interrupt Enable 2**

When the PCIE2 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 2 is enabled. Any change on any enabled PCINT23..16 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI2 interrupt vector. PCINT23..16 pins are enabled individually by the PCMSK2 register.

- **Bit 1 - PCIE1: Pin Change Interrupt Enable 1**

When the PCIE1 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 1 is enabled. Any change on any enabled PCINT14..8 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI1 interrupt vector. PCINT14..8 pins are enabled individually by the PCMSK1 register.

- **Bit 0 - PCIE0: Pin Change Interrupt Enable 0**

When the PCIE0 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI0 interrupt vector. PCINT7..0 pins are enabled individually by the PCMSK0 register.

Pin Change Mask Register

12.2.6 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

- **Bit 7..0 – PCINT23..16: Pin Change Enable Mask 23..16**

Each PCINT23..16-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

12.2.7 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: Reserved Bit**

This bit is an unused bit in the Atmel® ATmega328P, and will always read as zero.

- **Bit 6..0 – PCINT14..8: Pin Change Enable Mask 14..8**

Each PCINT14..8-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT14..8 is set and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT14..8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

12.2.8 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

- **Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**

Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Interrupt Vector

11.1 Interrupt Vectors in ATmega328P

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt

PCICR / PCMSK

PCICR

핀의 상태가 변화하면 인터럽트 발생. (L → H , H → L)

PCIE0 비트가 설정되고 (1) , 상태 레지스터 (SREG)의 I 비트가 설정 (1)되면 핀 변경 인터럽트 0 이 활성화된다..
활성화 된 PCINT7 ~ 0 핀의 변경은 인터럽트를 발생시킵니다.
핀 변경 인터럽트 요청의 해당 인터럽트는 PCI0 인터럽트 벡터에서 실행됩니다.
(데이터시트 상에 해당 범주를 지원한다면 가능하고 지원하지 않는다면 불가능합니다.)

PCINT 0 ~ 7 중에 신호가 변화하면 PCINT0_vector 호출
PCINT 8 ~ 14 중에 신호가 변화하면 PCINT1_vector 호출
PCINT 16 ~ 23 중에 신호가 변화하면 PCINT2_vector 호출

PCINT7 ~ 0 핀은 PCMSK0 레지스터에 의해 개별적으로 활성화된다.

PCMSK

PCINT7 ~ 0 Bit 는 해당 I / O 핀에서 핀 변경 인터럽트를 활성화할지 여부를 선택한다.
PCINT7 ~ 0 이 설정되고 PCICR의 PCIE0 비트가 설정되면 해당 I / O 핀에서 핀 변경 인터럽트가 활성화.
PCINT7 ~ 0 이 해제되면 해당 I / O 핀의 핀 변경 인터럽트가 비활성화된다.

Interrupt 설정 방법

STEP 1 : Turn on Pin Change Interrupts

PCINT를 설정하려면 PCICR 을 구성해야 한다.
모든 인코더를 PCINT23 : 16으로 라우팅 할 것이므로 PCIE2를 활성화 한다.
이를 위해 코드 라인: PCICR |= 0x04 을 포함.
여러 가지 방법으로 할 수 있지만 다른 비트는 변경하지 않고 핀은 1로 설정하는 것이 좋다.

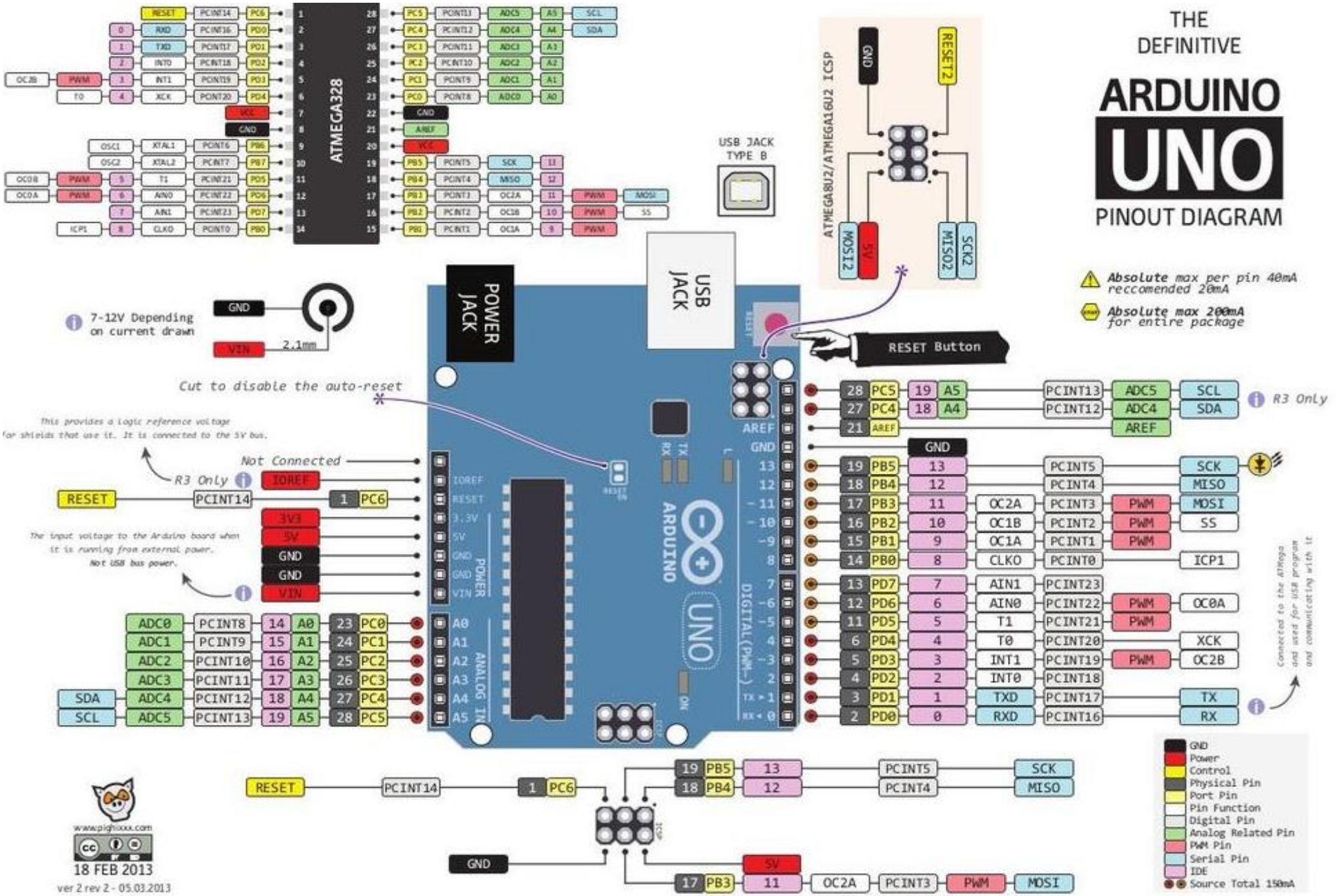
STEP 2: Pin Selection

PCI2에 매핑 된 모든 핀은 해당 ISR을 실행하지만 핀 변경 마스크 레지스터에서 마스크 된 비트를 설정 한 경우에만 해당.
PCI2에 매핑 된 핀 8개 중 6개만 사용하면 나중에 필요할 경우에도 다른 아날로그-인 핀을 사용할 수 있다.
PCINT 16:21에 대한 인코더만 배선하므로, 다음 코드 라인을 포함하면 됩니다: PCMSK2 |= 0x3F.

STEP 3: Write the ISR

ISR을 작성할 때마다 가능한 짧게 유지하고 변수를 선언할 때는 최적화되지 않도록 변수 유형을 휘발성 유형으로 만들어야 합니다.
ISR을 정의하려면 다음을 포함 : ISR(PCINT2_vect){} // PCINT16-PCINT23.

Pin Map


THE DEFINITIVE
ARDUINO UNO
PINOUT DIAGRAM
ATMEGA328
ATMEGA8U2/ATMEGA16U2 ICSP
USB JACK TYPE B
POWER JACK
USB JACK
RESET Button
7-12V Depending on current drawn
2.1mm
Cut to disable the auto-reset
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
R3 Only
Not Connected
The input voltage to the Arduino board when it is running from external power. Not USB bus power.
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
Connected to the ATMega and used for USB program and communicating with it
www.pighixxx.com
18 FEB 2013
ver 2 rev 2 - 05.03.2013
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA

인터럽트 코드

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

int main (void)
{
      DDRB &= ~( 1 << DDB5 );
      PORTB |= ( 1 << PORTB4 );

      PCICR |= ( 1 << PCIE0 )
      PCMSK0 | = ( 1 << PCINT5 )

      sei ( );
      while (1);
      return 0;
}

ISR ( PCINT0_vect )
{
      PORTB ^= 0x10;
}
```

Port clear 해당 비트는 무조건 0, 입력으로 받겠다.
OR : 덮어쓰기 , 있다 = 변함없다 , 없다 = 반영

pin change interrupt 활성화
어떤 pin 을 사용할 것인지 설정

sei ( ) 로 전체 인터럽트 활성화

ISR은 연결된 해당 Port의 Pin에서 조건이
충족 될 때마다 호출됩니다
해당 Pin 의 입력 상태(신호 레벨) 변화에 따라
ISR(PCINT0_vect) 로 즉시 이동해 함수가 실행된다.
13번 입력 핀(PCINT5)이 변화 할 때마다
12번 핀의 LED 출력을 반전한다.

인터럽트 코드

```
ISR ( PCINT0_vect )
{
      PORTB ^= 0x10;
}
```

버튼을 누를 때마다 toggle

Bit	7	6	5	4	3	2	1	0
PORTB	0	0	0	0	0	0	0	0
0x10	0	0	0	1	0	0	0	0
XOR 결과 값 (1)	0	0	0	1	0	0	0	0
0x10	0	0	0	1	0	0	0	0
XOR 결과 값 (2)	0	0	0	0	0	0	0	0

회로 측정

그림 1. VCC 와 GND 사이의 전압

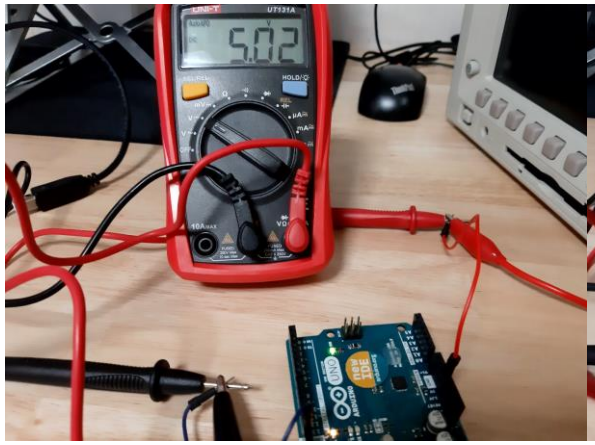
: 5.02 [V]

그림 2. VCC 와 GND 사이의 전류

: 0.6 [mA]

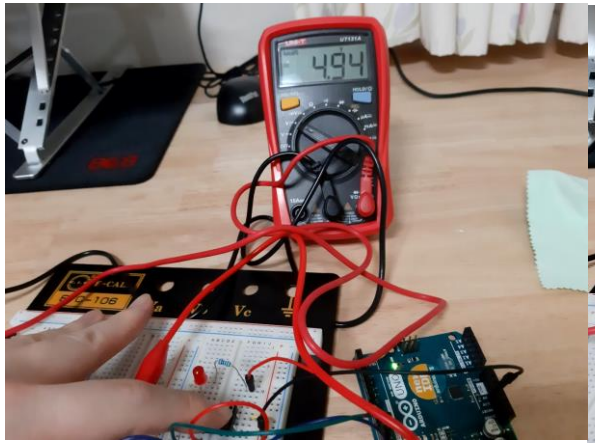
그림 3~4. 74HC14 의 Pin 2 와 GND 사이의 전압

- 스위치 ON → 4.94 [V]
- 스위치 OFF → 0.002 [V]

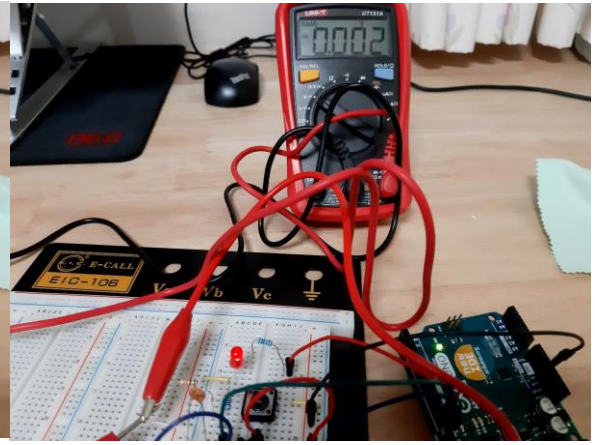


1

2



3



4

회로 측정

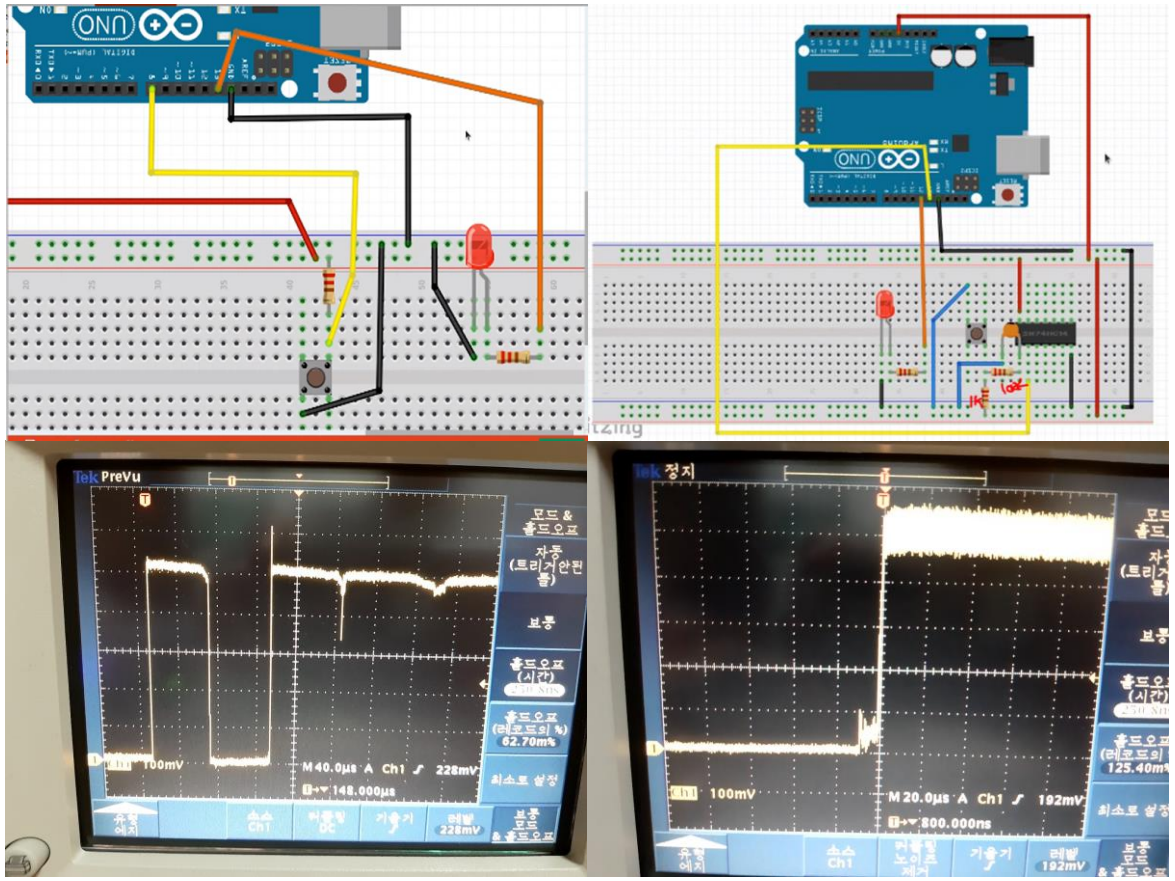
그림 1.

chattering : 스위치 접점이 닫히거나 열리는 순간에 기계적인 진동에 의해 매우 짧은 시간 안에 붙었다가 떨어지는 것을 반복하는 현상

그림 2.

RC 필터를 활용하여 파형이 정상적으로 표시된다.
C : 세라믹 콘덴서

(RC 필터 A4 용지 작성 중)



1 2

RC 필터 및 open drain 컬렉터

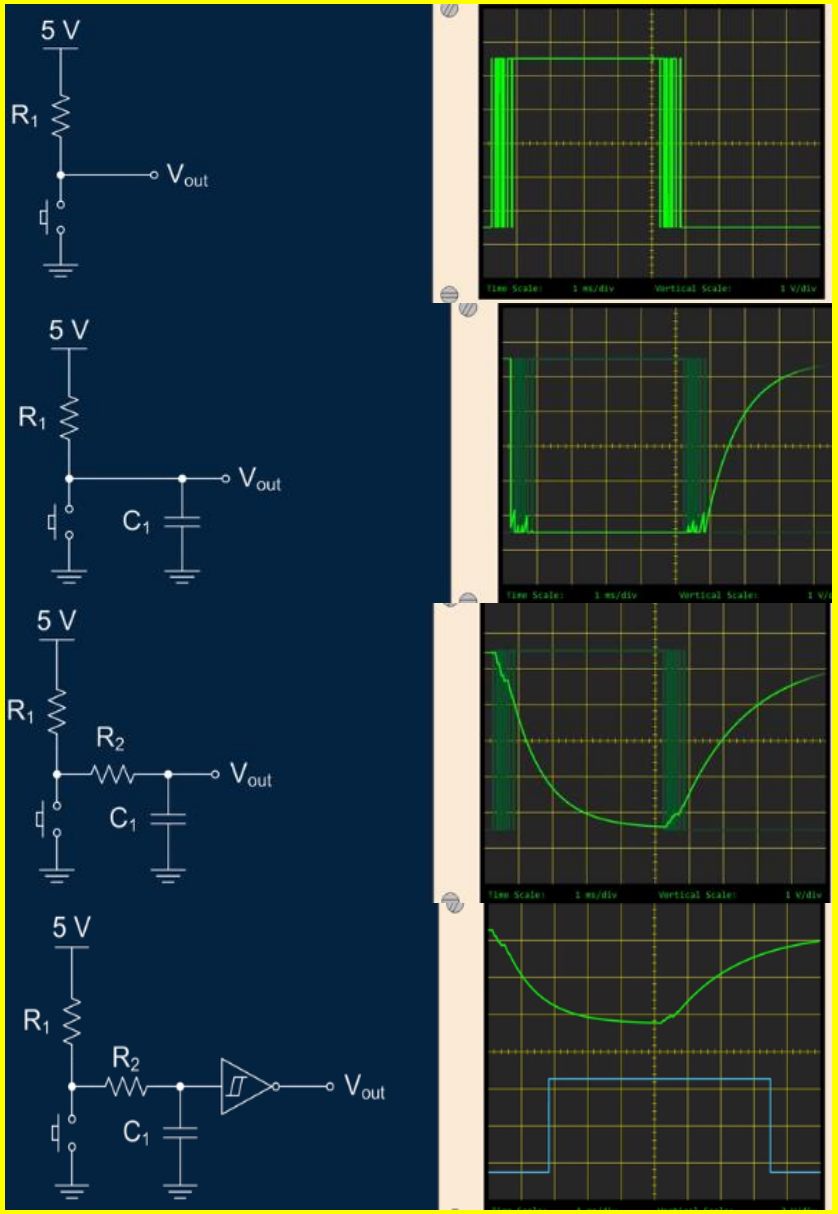
1ns 감지하며 신호에 의해 트리거
신호의 상승 에지와 하강 에지 모두 바운스를 볼 수 있다.

콘덴서 연결 시 작동하지만 초기 방전이 느리지 않다.
아직도 신호가 양방향으로 조금씩 바운스 되고 있다.

저항을 추가로 연결 시 바운스는 상승 및 하강 에지 모두 필터링 되지만 매우 느리다.
신호가 약간 반등하여 임계 값 수준으로 떨어지면 일반 CMOS 입력에서 이중 트리거가 발생할 수 있다.

해결 : 슈미트 트리거와 같은 입력 히스테리시스가
추가된 논리 장치를 사용하는 것.

상승 및 하강 에지 트리거 레벨을 분리하여 동작으로 인한 문제를 제거하고 느린 입력 edge 로 인한 초과 전류를 방지



RC 필터 및 Schmitt trigger

채터링 (바운스)

- 짧은 시간 내 (수십 uS 또는 nS) 여러 번 On, OFF 반복한다.
- LOGIC IC들은 수십 MHz 까지 동작하기 때문에 오동작이 일어난다.

슈미트 트리거

- LOGIC 에서 판단 할 수 있게 파형을 정형해주는 역할 (채터링 제거하는 역할이 아니다)
- 히스테리시스의 목적 : 노이즈 마진을 얻기 위함 , 주파수를 정확히 얻기 위해서 사용

파형 정형

전자회로에서 신호에 혼입되어 들어오는 잡음의 영향을 막기 위해
일정 값 이상일 때 출력 ON / 일정 값 미만일때 출력 OFF , 그 차이 값이 hysterisis 값.

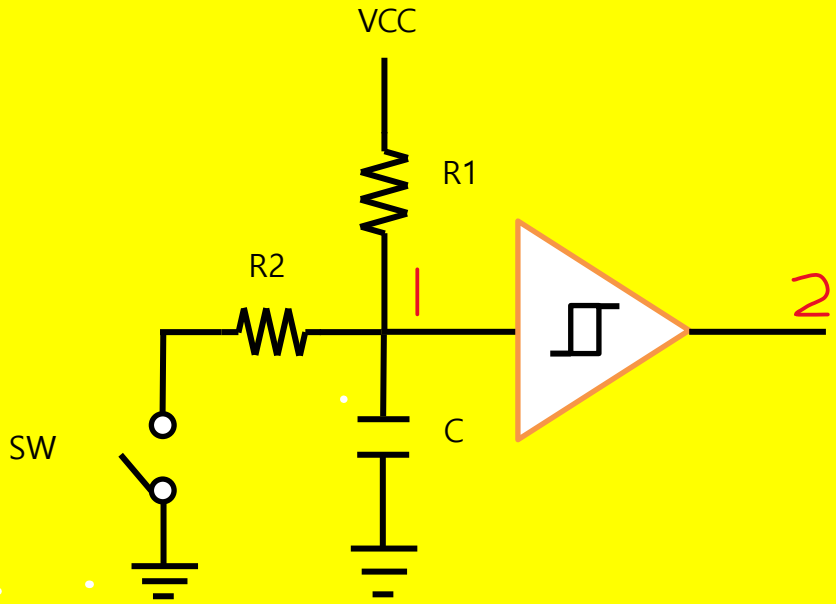
스위치와 병렬로 0.1uF ~ 1uF 정도 되는 콘덴서를 연결
채터링이 짧은 시간동안 이루어지므로 콘덴서의 충반전이 이들 시간보다 길다면 채터링을 제거할 수 있다.
상승시간이 지연되면서 짧은 파형의 채터링 부분이 없어진다.
상승시간이 둔화 되어서 신호를 받는 쪽에서 오류가 발생할 수 있어 다음 단에 슈미트 트리거를 연결하여 파형을 정형한다.
RC 회로 때문에 스위치 속도가 빠른 경우 신호가 들어오지 않을 수 있기 때문에 RS-FF 이용

RC 필터 및 Schmitt trigger

스위치에 콘덴서를 병렬로 연결하여 회로의 반응을 의도적으로 느리게 만드는 방법은 논리 값의 판단이 불분명한 영역에 신호가 존재하기 때문에 적절하지 않다.

콘덴서 연결한 신호를 슈미트 트리거에 보내면 바운스에 의한 전압이 Vt + 까지 올라가지 않으면 깨끗한 신호 출력

첫 번째 신호와 슈미트 트리거 출력을 비교하면 히스테리시스에 의한 지연이 있다.

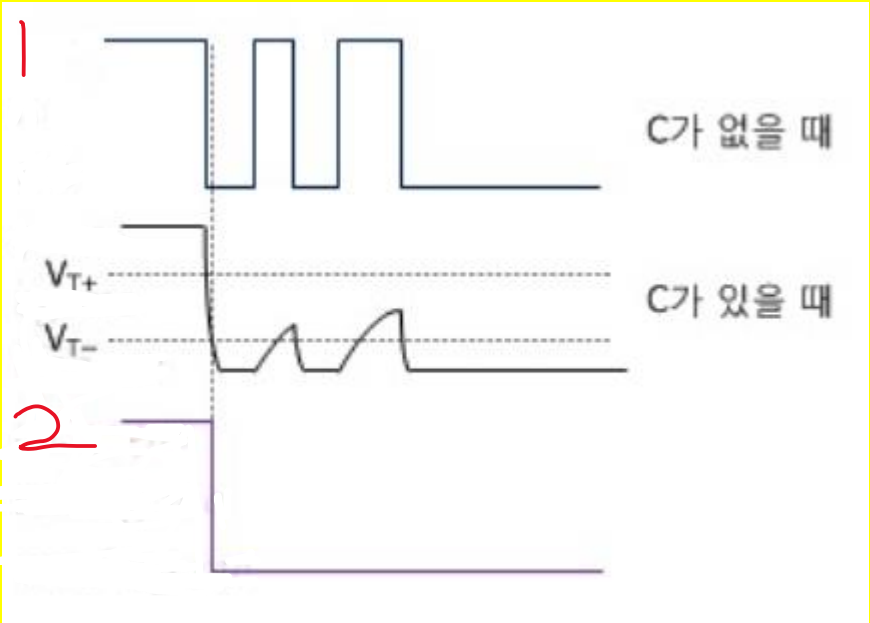


바운스 제거 회로는 시정수를 적절하게 정하는 것이 중요하다.

충전 시정수는 R1, C
방전 시정수는 R2, C

R1 〉〉 R2 의 관계가 만족해야 한다.

그렇지 않으면 스위치를 눌렀을 때의 전압이 충분히 낮아지지 않는다.
R1 , C 값이 바운스에 의해 발생하는 펄스의 주기보다는 더 크도록 설정해야 바운스 현상을 제거할 수 있다.
R1 , C 값을 필요 이상으로 크게 설정하면 회로의 반응이 너무 느려져서 스위치가 인식 못할 수 있다.
인식하더라도 시간 지연이 크다.
반면에 **시정수를 너무 작게 하면** 바운스에 의한 전압이 슈미트 트리거의 문턱 전압 이상 올라가게 되므로 슈미트 트리거를 사용하더라도 바운스를 제거하지 못한다.



RC 필터 및 Schmitt trigger

콘덴서에는 5V 전압이 충/방전 반복

충전 시 RC 경로의 시정수 KΩ (VCC) * 0.47 uF = 4.7 ms

방전 시 RC 경로의 시정수 Ω (SW) * 0.47 uF = 47us

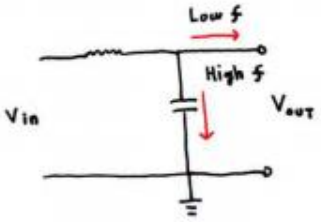
스위치 누르면 충전 시정수가 방전 시정수보다 충분히 크기때문에 콘덴서는 충전되지 못하고 방전된다.

스위치 개방하면 콘덴서는 다시 5V 로 충전되고, 충전된 콘덴서에 의해 슈미트 트리거에

High 전압이 인가되어 Low 가 출력된다.

RC 필터

Low pass Filters

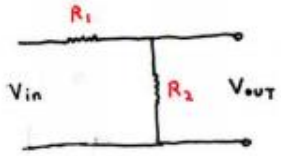


$$X_c = \frac{1}{2\pi f C}$$

$$V_{out} = \frac{X_c}{Z} \cdot V_{in}$$

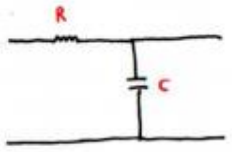
$$= \frac{X_c}{\sqrt{R^2 + (X_L - X_c)^2}} \times V_{in}$$

$f \uparrow \quad X_c \downarrow \quad V_{out} \downarrow$



$$V_{out} = \frac{R_2}{R_1 + R_2} \times V_{in}$$

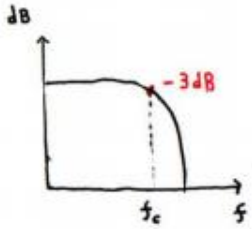
$R_2 \uparrow \quad V_{out} \uparrow$



$$f_c = \frac{1}{2\pi RC}$$

$C \uparrow \quad f \downarrow$

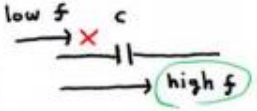
$R \uparrow \quad f \downarrow$



$V_{out} = 0.707\ V_{in}$

$$dB = 20 \log \frac{V_{out}}{V_{in}}$$

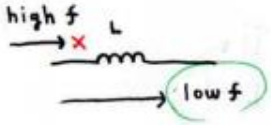
$$dB = 20 \log (0.707) = -3.01$$



$$X_c = \frac{1}{2\pi f C}$$

$f \uparrow \quad X_c \downarrow$

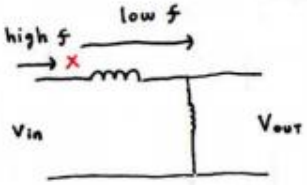
$f \downarrow \quad X_c \uparrow$



$$X_L = 2\pi f L$$

$f \uparrow \quad X_L \uparrow$

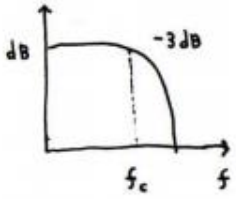
$f \downarrow \quad X_L \downarrow$



$$V_{out} = \frac{R}{\sqrt{R^2 + (X_L - X_c)^2}} \times V_{in}$$

$f \uparrow \quad X_L \uparrow \quad V_{out} \downarrow$

$f \downarrow \quad X_L \downarrow \quad V_{out} \uparrow$



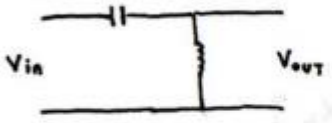
$$f_c = \frac{R}{2\pi L}$$

$R \uparrow \quad f_c \uparrow$

$L \uparrow \quad f_c \downarrow$

RC 필터

High pass Filters

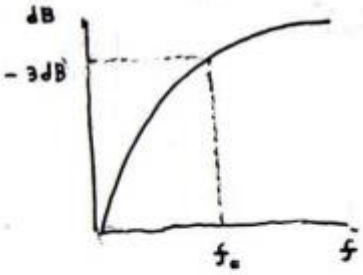


$$V_{OUT} = \frac{R}{\sqrt{R^2 + (X_L - X_C)^2}} \times V_{in}$$

$f \uparrow$ $X_C \downarrow$ $V_{OUT} \uparrow$

$f \downarrow$ $X_C \uparrow$ $V_{OUT} \downarrow$

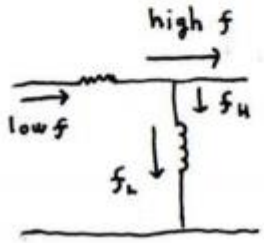
$f \infty$ $V_{OUT} \rightarrow V_{in}$

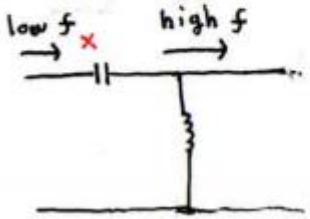


$$f_c = \frac{1}{2\pi RC}$$

$$\text{Gain (dB)} = 20 \log \left(\frac{V_{OUT}}{V_{in}} \right)$$

$V_{OUT} = 0.707\ V_{in}$





dB

상대값 , 어떤 값을 log scale 로 보여주는 것

로그 함수로 데시벨 표현

비율의 상용 로그 값을 크게 만들기 위해 10을 곱셈

dBm

절대적인 전력량을 1mW 기준으로 dB scale 로 보여주는 것

dB + dBm = ?? x10 15mW = ???

dBm + dB = dBm 15mW x10 = 150mW

dBm 으로 표현된 어떤 전력 값에

dB 로 표현된 신호 전력의 증가 / 감소 를 덧셈

전력 : 10 log $P = \frac{V^2}{R} = I^2R$

전압 : 20 log

전압은 제곱해야 전력이 되고 , 제곱을 dB 로 변환하면 (log scale)

10 x 2 = 20 을 곱셈

RC Low pass filter

Low pass filter

0 Hz 부터 차단주파수 fc 까지 저주파 신호를 통과시킨다.
fc 를 벗어나면 다른 모든 주파수 성분을 거부한다

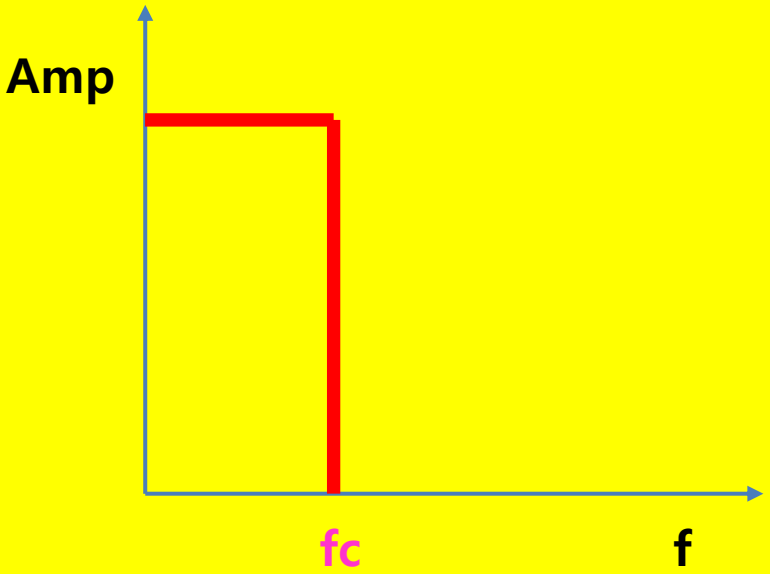
출력은 항상 입력보다 작다.

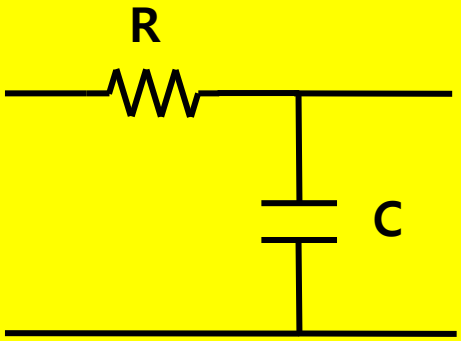
낮은 주파수에서 Xc 는 증가한다.
따라서 ' 출력 = 입력 ' 거의 동일하다
Vout = Vin

높은 주파수로 이동하면 Xc 는 감소한다.
출력 감소 , 매우 높은 주파수에서는 출력이 0 이 되는 경향이 있다.
Vout = 0

입력 신호의 저주파를 전달하여 고주파를 거부하거나 감쇠시킨다.

$$Vout = \frac{Xc}{Xc+R} \times Vin$$





RC Low pass filter

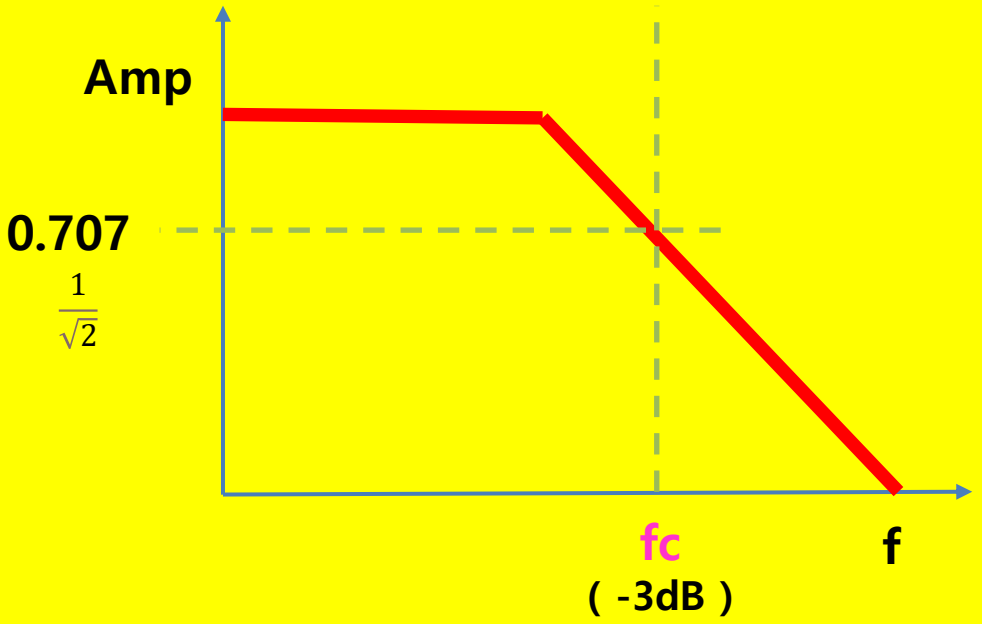
낮은 주파수에서 zero 또는 최소 감쇠 제공
높은 주파수는 감쇠 증가

(출력 = 입력 × 0.707) 의 주파수를
차단 주파수 또는 (−3 dB) 주파수

차단 주파수 이후 출력이 −20 dB / decade 속도로
감소한다.

즉, 주파수를 10 배 증가하면 출력이 10 배 감소

차단 주파수에서 출력은 입력의 1 / $\sqrt{2}$ 배,
(시스템의 이득 또는 감쇠)



$$\frac{Vout}{Vin} = \frac{Xc}{\sqrt{R^2+Xc^2}} \;,\; \frac{1}{\sqrt{2}} = \frac{\frac{1}{wC}}{\sqrt{R^2+(\frac{1}{wC})^2}} \;,\; \frac{1}{2} = \frac{(\frac{1}{wC})^2}{R^2+(\frac{1}{wC})^2}$$

$$w = \frac{1}{RC} \;,\; fc = \frac{1}{2\pi RC}$$

RC Low pass filter

입력 주파수를 높이면 신호의 출력이 감소할 뿐만 아니라 위상도 변경된다.

$\emptyset = -tan^{-1}(wCR)$

w = 0 → $\emptyset = -tan^{-1}(0) = 0°$ (출력은 입력과 동 위상)

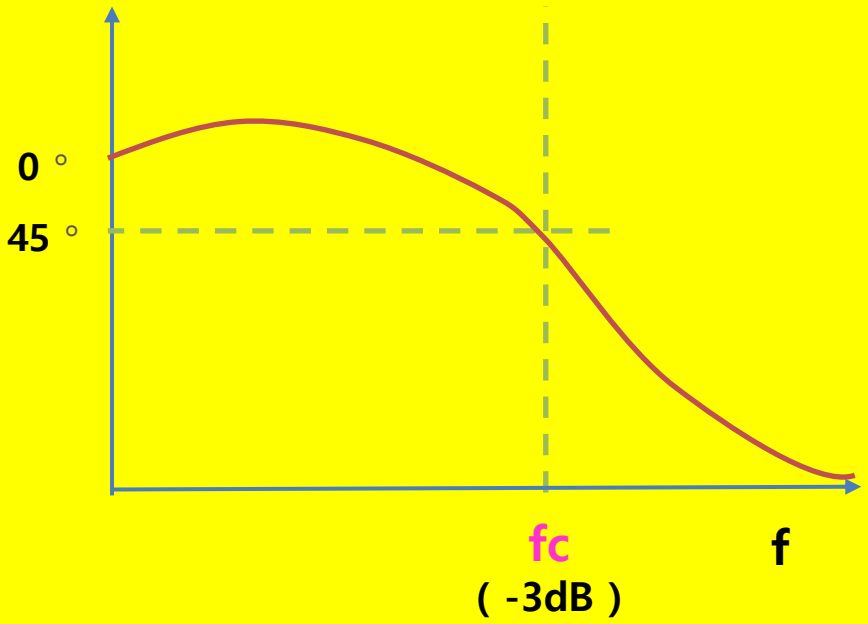
w = wc → $\emptyset = -tan^{-1}(1) = -45°$ (차단 주파수에서 w 의 값은 1/RC 와 같다)

w = ∞ → $\emptyset = -tan^{-1}(1) = -90°$

0 주파수에서는 위상 = 0 °

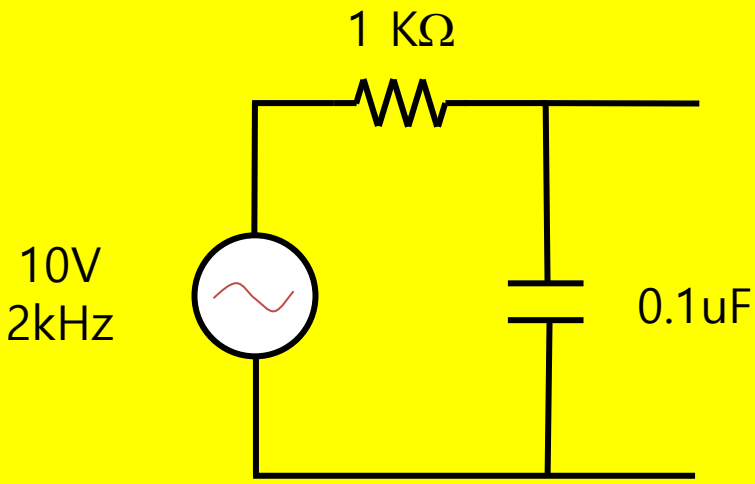
차단 주파수에서는 위상 = −45 °
출력 신호가 입력 신호보다 45 ° 지연

차단 주파수 벗어나면 위상은 −90 ° 방향으로 이동



RC Low pass filter

주어진 필터에 대해 – 3 dB 주파수 찾기
주어진 입력 신호에 대한 출력 전압 찾기



입력에는 2 kHz 의 10 [V] 사인파 신호가 적용

$fc = 1 / 2\pi RC = 1 / 2\,\pi \times 10^{3} \times 0.1 \times 10^{-6}$

R 과 C 의 값을 입력하면 차단 주파수 값 = 1.59 kHz

이제 2 kHz 주파수에서 출력 전압 값 찾기 전에
리액턴스 값 확인

$$Vout = \frac{Xc}{\sqrt{R^2+Xc^2}} \times Vin$$

$Xc = 1 / 2\pi fc = 1 / 2\pi \times 2000 \times 0.1 \times 10^{-6}$

Xc 의 값은 796 을 얻을 수 있다.

$$Vout = \frac{796}{\sqrt{R^2+796^2}} \times Vin$$

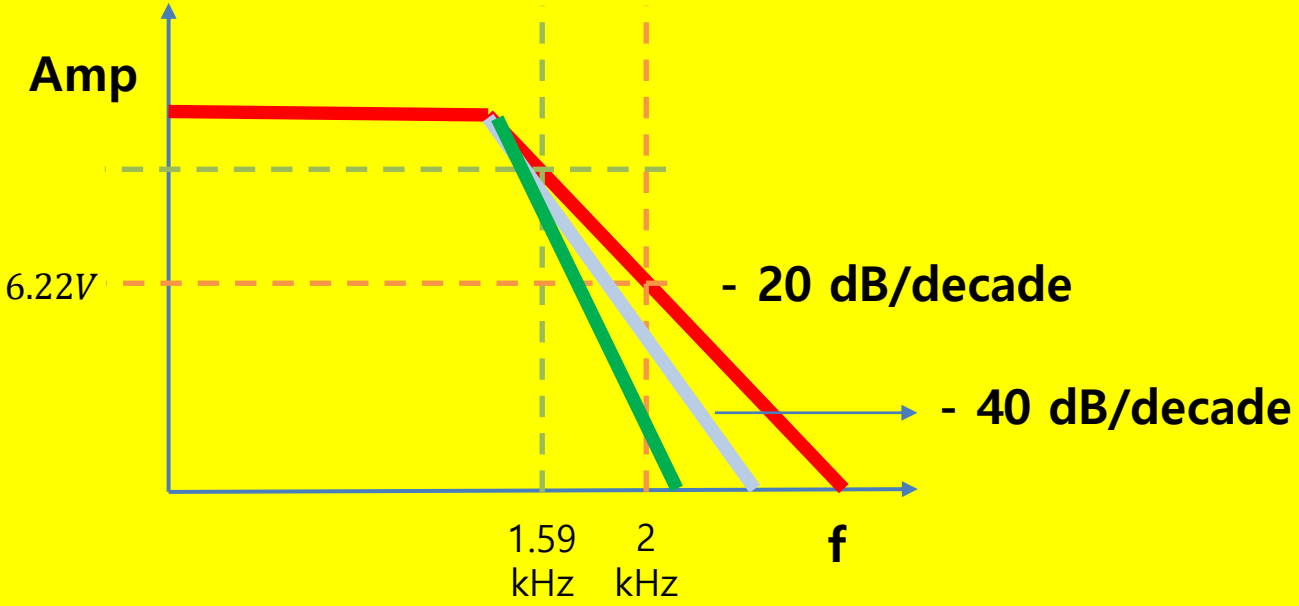
입력 측에서 2 kHz 주파수의 10V 정현파 신호를 적용하면 출력 전압은 6.22 [V]

RC Low pass filter

차단 주파수는 1.59 kHz

2 kHz 주파수의 출력 값은 6.22 [V]

n^{th} 필터의 숫자가 올라갈수록,
- 2 kHz 주파수에서 진폭이 크게 줄어든다.
- 날카로운 급격한 감쇠를 얻을 수 있다.
- n 차 필터를 사용하는 경우 감쇠는 n × −20 dB / decade



RC Low pass filter

2 차 필터의 차단 주파수는

$$fc = \frac{1}{2\pi \times \sqrt{R_1 C_1 R_2 C_2}} = \frac{1}{2\pi R_1 C_1}$$

R1 = R2 및 C1 = C2 라고 가정하면 fc 는 $1 / 2\pi R1C1$ 으로 주어질 수 있다.

더 높은 차수의 필터를 사용할수록 차단 주파수에서의 감쇠도 증가

1차 필터 : 출력 = 입력의 $1 / \sqrt{2}$ 배
2차 필터 : 출력 = 입력의 $1 / \sqrt{2} \times 1 / \sqrt{2}$ (출력 = 입력 × 0.5)

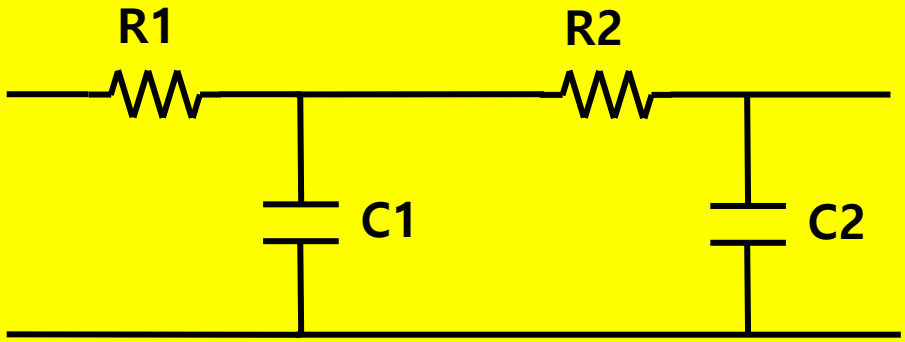
차단 주파수가 동일한 n 개의 필터를 cascade 하면

출력이 $(\frac{1}{\sqrt{2}})^n \times$ Vin의 계수만큼 감소

Low pass 필터의 2 차는 1 차까지 로드되지 않도록 주의해야 한다.

부하 효과를 최소화 하려면 R2 값이 R1 의 최소 10배 이상이어야 한다. (R2 = R1 × 10배)

또는 Active filter 사용 (활성필터는 이득 제공 뿐만 아니라, 두 단계 사이에서 버퍼 역할을 한다.)



Instruction-level Parallelism

전체 instruction 을 overlap 하는 방식
명령어 간의 상관관계를 잘 따져서 병렬처리가 가능하도록 하는 기술

(가)와 (나)의 경우 의존성이 있는 반면,
(다) 및 (라)는 어느 코드도 의존성을 갖고 있지 않다.

1 cycle 에서 실행하는 명령 : (가) (다) (라)
2 cycle 에서 실행하는 명령 : (나)

4 개의 명령어가 모두 의존성으로 묶여 있다면,
4 개 명령어를 4 cycle 에 처리할 수 있다고 하여
ILP = 4 / 4 = 1

위와 같은 경우, 4 개의 명령을 2 cycle 에 처리할 수 있으므로
ILP = 4 / 2 =2 라고 수치화 할 수 있다.
의존성이 서로 아무 것도 없는 명령들이면 ILP 값은 = 4

```c
int main(void)
{
    int x, y, a, b, c, d;
    int array[4] = { 10, 20 , 30 , 40};

    x = array[4]; // 가
    printf(" x 의 결과 : %d\n", x);
    y = x + 10; // 나
    a = b - 4; // 다
    c = d * 7; // 라

    return 0;
}
```

멀티 스레드를 통한 명시적인 병렬성이냐(TLP)
아니면 싱글 스레드에서도 명령어 의존성을 분석하여 병렬성을 구현했느냐(ILP)의 차이

컴파일러는 제한 없이 자유롭게 넓은 범위의 의존성을 검사할 수 있을 것
structual stalls / Data Hazard stalls / Control Stalls
컴퓨터 과학자들은 이런 값들을 얼마나 줄일 수 있을까 ? 고민

Instruction-level Parallelism

시스템 설계자는 항상 프로그램을 더 빠르게 실행하는 방법을 생각한다.

- 프로그램의 명령어 수에 평균 프로세서 수를 곱한 값
- 각 명령 (CPU) 을 실행하는 데 필요한 사이클에 필요한 시간을 곱한 값
- 각 프로세서 주기 (t_CLK)

실행 시간을 줄이려면 이러한 용어 중 하나 이상을 줄여야 한다.

우리는 하드웨어를 설계하여 5단계 파이프 라인 구현

- CPI_{stall} contributors
  - Data hazards
  - Control hazards: branches, exceptions
  - Memory latency: cache misses

즉시 사용하려고 할 때 취해진 분기 명령에서 발생한다.

경우에 따라 하드웨어는 필요한 작업을 완료할 수 없는 경우 파이프라인 단계의 실행을 지연시키기 위해 파이프라인에 "NOP 버블"을 도입해야 한다.

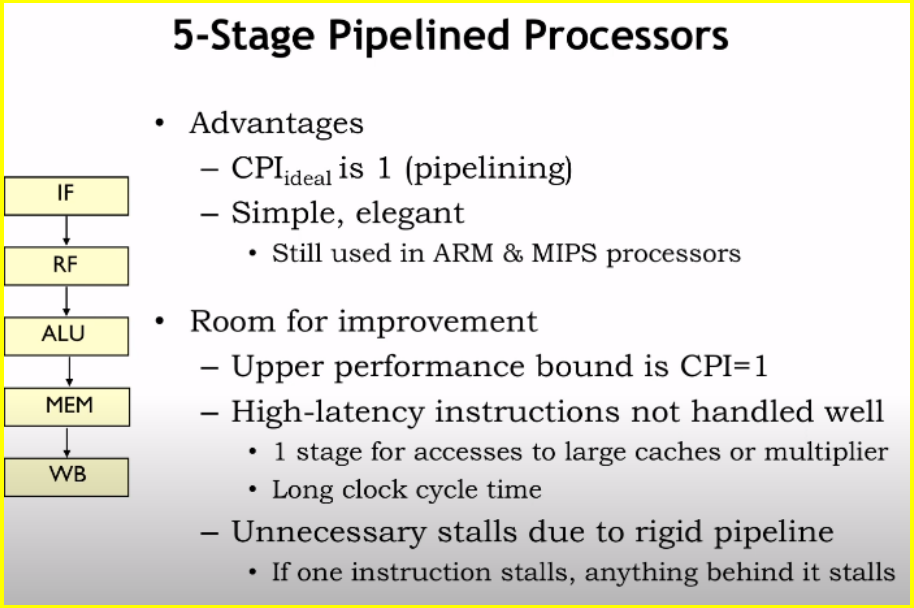
LD 명령에 의해 메모리에서 로드 된 값을 즉시 사용하려고 할 때, 그리고 메인 메모리에서 캐시 누락이 충족되기를 기다릴 때 분기 명령에 따라 발생한다.

CPI_stall은 파이프라인에 도입된 NOP에 손실된 주기

Instruction-level Parallelism

그 값은 채취한 분기의 빈도와 LD 결과의 즉각적인 사용에 따라 달라진다.

예를 들어 LD가있는 6 개의 명령어 루프가 완료하는 데 8 주기가 소요되는 경우 루프에 대한 CPI_stall은 6 개 명령어마다 2 개의 추가 사이클이 된다.



고전적인 5단계 파이프라인은 CPI_stall을 합리적으로 적당한 값으로 유지하면서 t_CLK를 상당히 줄일 수 있는 효과적인 절충안이다.

개선의 여지가 있다.

각 스테이지가 한 번에 하나의 지침에 대해 작업 중이기 때문에 CPI_ideal은 1

Instruction-level Parallelism

느린 작업은 ALU 단계에서 곱하기를 완료하거나 IF 또는 MEM 단계에서 큰 캐시에 액세스한다.
한 주기에 수행되어야 하는 모든 작업을 t_CLK가 강제로 수용하도록 한다.

캐시 누락으로 인해 LD 명령이 MEM 단계에서 지연되고,
초기 단계의 모든 명령도 LD에 의해 생성된 값에 의존하지 않을 수 있지만 지연된다.

이러한 제약을 완화하고 프로그램 실행 시간을 개선하는 데 무엇이 필요할까 ?

Improving 5-stage Pipeline Performance

- Lower t_{CLK}: deeper pipelines
  - Overlap more instructions
- Higher CPI_{ideal}: wider pipelines
  - Each pipeline stage processes multiple instructions
- Lower CPI_{stall}: out-of-order execution
  - Execute each instruction as soon as its source operands are available
- Balance conflicting goals
  - Deeper & wider pipelines ⇒ more control hazards
  - Branch prediction
- It all works because of instruction-level parallelism (ILP)

Instruction-level Parallelism

파이프 라인 단계의 수를 늘리면 클럭 사이클 시간을 줄일 수 있다.

병목 현상을 해소하기 위한 단계를 추가하고 파이프라인 단계(MEM1 및 MEM2)를 추가하여 메모리 작업을 완료하는 데 더 오랜 시간이 걸릴 수 있다.

추가 MEM 단계마다 LD 데이터 위험이 있을 때 더 많은 NOP 버블이 도입되어야 하기 때문에 CPI_stall 이 발생한다.

파이프라인이 깊어지면 프로세서가 더 많은 명령을 병렬로 실행하게 된다.

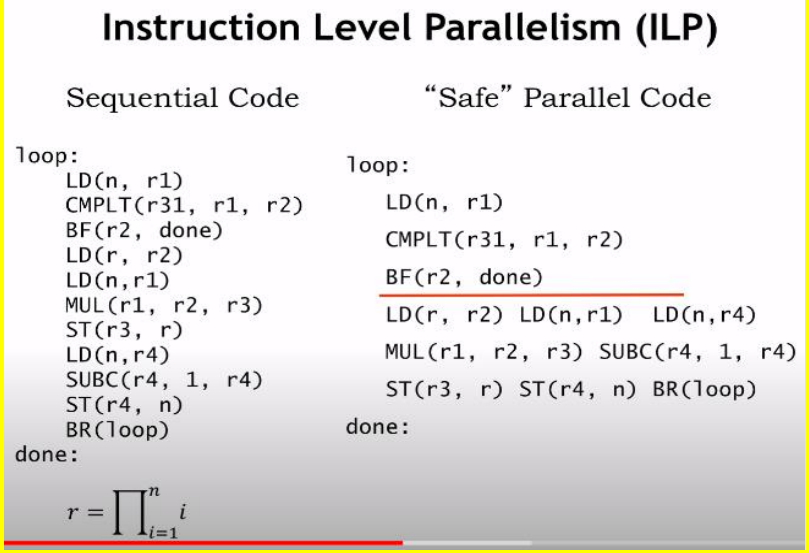
데이터 위험으로 인해 파이프라인에서 정지된 명령이 있는 경우, 실행이 계속 진행될 수 있는 다음과 같은 명령이 있을 수 있다.
파이프 라인에서 명령어가 서로 전달되도록 허용하는 것을 순서에 맞지 않는 실행이라고 한다.
실행 순서를 변경해도 프로그램에서 생성된 값에 영향을 주지 않도록 주의해야 한다.
더 많은 파이프라인 단계와 더 넓은 파이프라인 단계는 제어 위험의 폐기해야 하는 작업의 양을 증가시켜 잠재적으로 CPI_stall을 증가시킨다.
따라서 분기 결과를 예측하여 제어 위험의 수를 최소화하는 것이 중요.
광범위한 파이프라인 및 비순차적 실행을 활용할 수 있는 우리의 능력은 다음과 같은 지침을 찾는 것에 달려 있다. (PPT 39 Page 그림 참고)

이러한 속성은 집합적으로 "명령 수준 병렬화"(ILP)라고 한다.

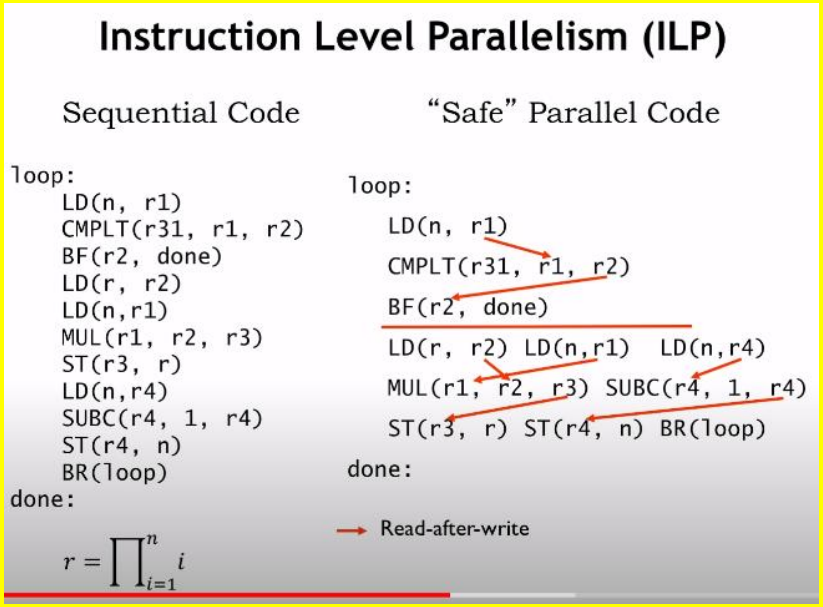
Instruction-level Parallelism

왼쪽은 최적화 되지 않은 loop.
BF 의 명령에 따르는 빨간 선을 주목.

라인 아래의 명령은 BF가 * 사용되지 않는* 경우에만 실행되어야 한다.



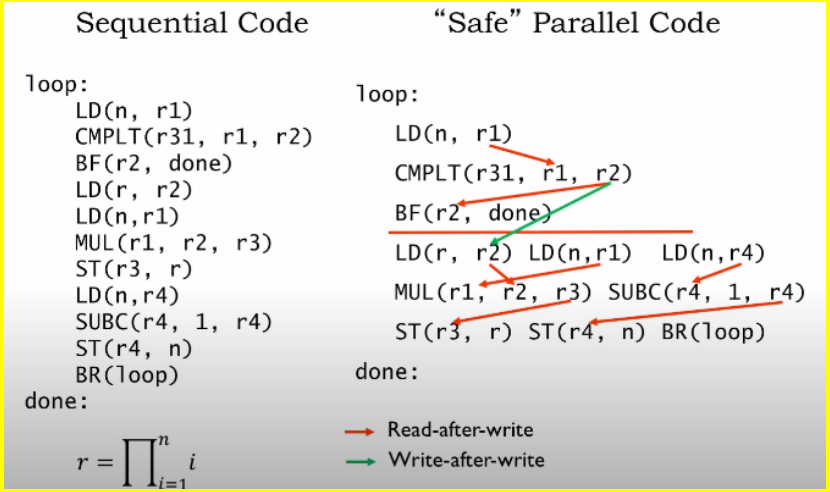
하나의 명령에 대한 피연산자 값이 이전 명령의 결과에 따라 달라지는 경우, 발생할 수 있는 잠재적 데이터 위험으로 인식한다.
5 단계 파이프 라인에서 ALU, MEM 및 WB 단계의 값을 피연산자 값이 결정되는 RF 단계로 다시 우회하여 이러한 위험을 많이 해결할 수 있었습니다.
우회는 명령이 실행 된 경우에만 작동하므로 그 결과를 우회할 수 있다.



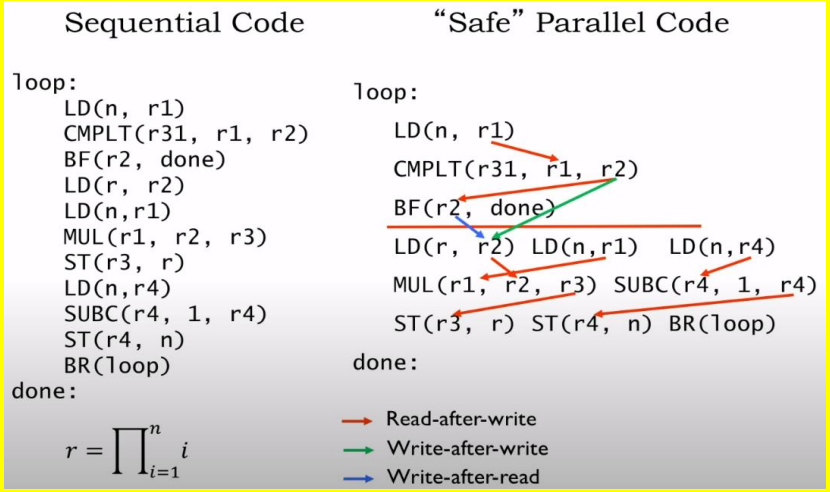
Instruction-level Parallelism

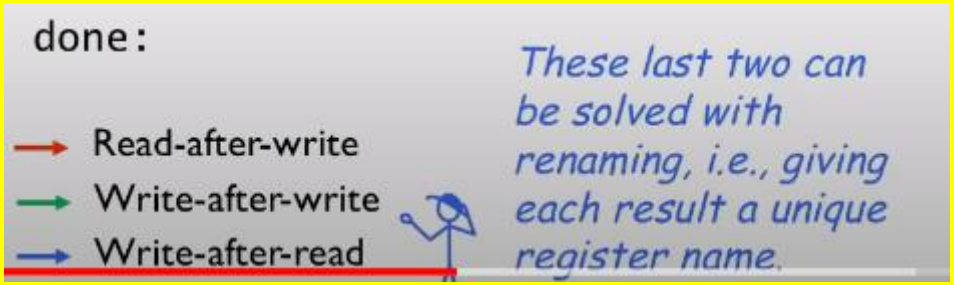
녹색 화살표는 동일한 대상 레지스터를 가진 두 명령 사이의 쓰기 후 제약 조건을 식별한다.

루프 끝의 R2에 올바른 값이 있는지 확인하기 위해 LD (r, R2) 명령어는 CMPLT 명령어의 결과가 레지스터 파일에 저장된 후 그 결과를 레지스터 파일에 결과를 저장해야 한다



파란색 화살표는 레지스터에 액세스 할 때 올바른 값이 사용되도록 하는 읽기 후 쓰기 제약 조건을 보여준다. LD (r, R2)는 BF에 대한 Ra 피연산자를 R2에서 읽은 후 R2에 저장해야 한다.





Instruction-level Parallelism

결과적으로 각 명령어 결과에 고유한 레지스터 이름을 지정하면 WAW 및 WAR 제약 조건을 제거 할 수 있다.

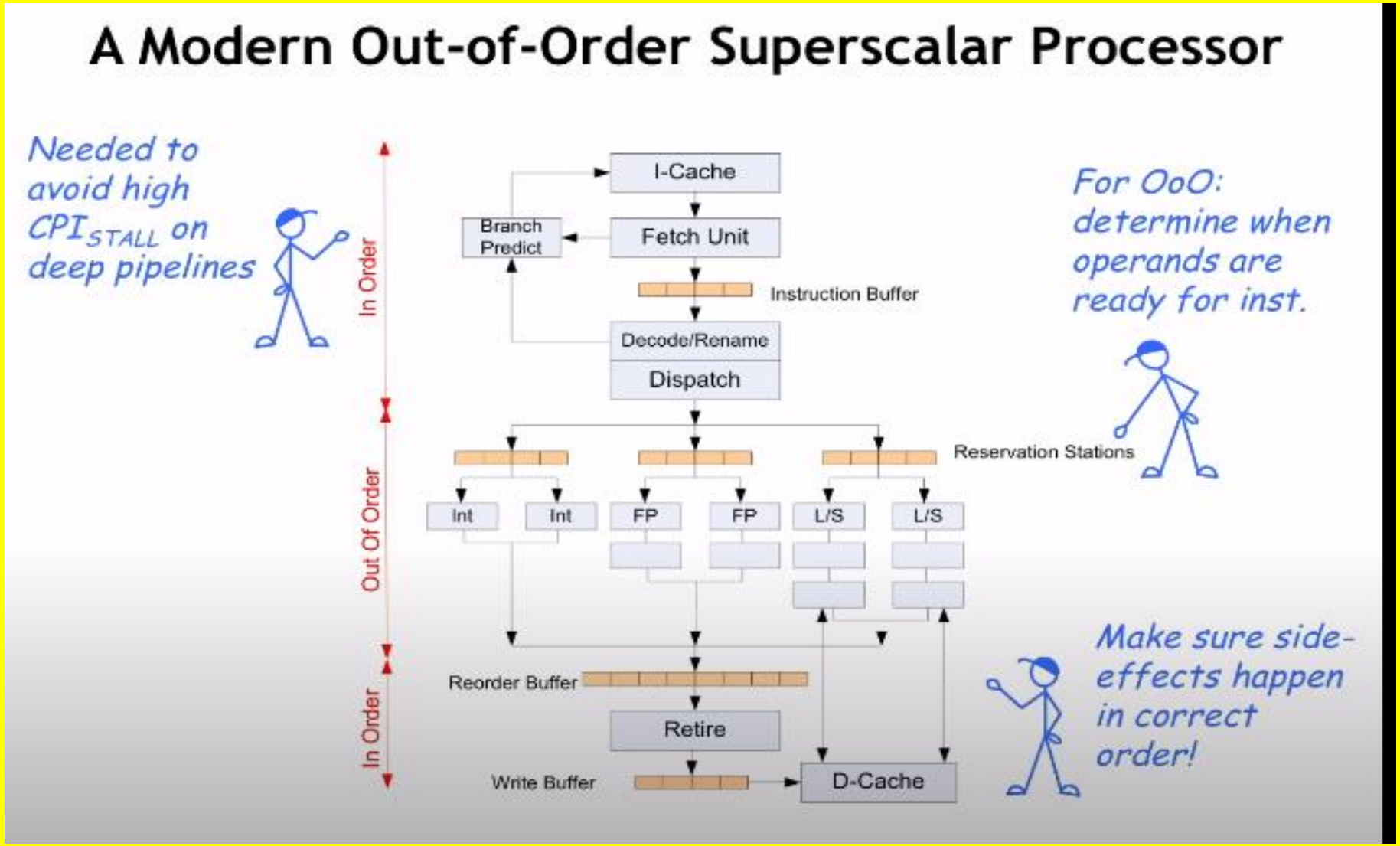
동시성을 활용하려면 N 개의 명령어를 병렬로 실행하도록 파이프 라인을 수정해야 한다.
실행 속도를 유지할 수 있다면 CPI_ideal은 최종 파이프 라인 단계를 종료 할 때 각 클록주기에서 N 개의 명령어 실행을 완료하므로 1 / N 이 된다.

N에 대해 어떤 값을 선택해야 할까?
다른 ALU 하드웨어에 의해 실행되는 명령어는 병렬, ADD 및 SHIFT 또는 정수 및 부동 소수점 연산으로 실행하기 쉽다.

주소 산술을 위한 별도의 하드웨어 (LD / ST 단위라고 함)가 있으면 LD / ST 명령어와 정수 산술 명령어를 동시에 실행할 수 있다.

기본적으로 ALU의 기능 단위 수와 레지스터 파일 및 주 메모리의 메모리 포트 수를 늘리면 여러 명령의 동시 실행을 지원하는 데 필요한 것을 갖게 될 것이다.

Instruction-level Parallelism



Instruction-level Parallelism

44 page 그림에 슈퍼스칼라 프로세서의 간단한 도표가 있다.
명령어 가져 오기 및 디코딩 핸들, 한 번에 4 개의 명령어.

실행 속도를 유지하는 능력은 분기 명령의 결과를 예측하는 능력에 크게 좌우 된다.
넓은 파이프 라인이 실제로 실행하려는 명령으로 대부분 채워지도록 한다.
좋은 분기 예측은 이전 분기의 기록을 사용해야 하며 최소한의 하드웨어에서 좋은 예측을 얻기 위해 많은 영리함을 제공했다. (위키백과에서 "분기 예측 변수"를 검색)

명령어가 피연산자로서 이전 명령의 결과를 필요로 하는 경우, dispatcher는 어떤 기능 단위가 결과를 산출할 것인지를 식별하였다.

명령은 표시된 기능 단위가 결과를 생성 할 때까지 대기열에서 대기하고 모든 피연산자 값이 알려지면 마지막으로 대기열에서 명령을 가져와 실행한다.

명령은 피연산자가 사용 가능한 즉시 다른 기능 단위에 의해 실행되기 때문에 실행 순서는 원래 프로그램과 동일하지 않을 수 있다.

효과적인 CPI는 캐시 적중률, 성공적인 분기 예측, 이용 가능한 ILP 등에 따라 다르다

Instruction-level Parallelism

파이프 라인 깊이가 증가하면 CPI_stall 및 타이밍 오버 헤드가 증가 할 수 있다.

ILP를 높이기 위해 더 많은 비 순차적 실행을 사용하는 것과 잘못 예측된 분기의 영향으로 인한 CPI_stall의 증가와 메인 메모리를 더 빨리 실행할 수 없는 것 사이에 유사한 절충이 존재한다.

낮은 t_CLK 및 추가적인 비 순차적 실행 로직으로 인한 성능 향상보다 전력 소비량이 빠르게 증가한다.

분기 예측 및 동시 실행을 추가로 개선하는 데 필요한 추가적인 복잡성은 매우 어렵게 보인다.

이러한 모든 요인은 비 순차적 슈퍼 스칼라 파이프 라인 프로세서의 성능에서 향후 실질적인 개선을 기대할 수 없음을 시사한다.

그래서 시스템 설계자들은 데이터 레벨 병렬화(DLP)와 스레드 레벨 병렬화(TLP)를 이용하는 데 관심을 돌렸다.

Goto

옵션 수가 많고 값 범위가 작을 때 효율적

goto 문을 실행하면 프로그램에서 지정된 지점으로 직접 분기가 만들어진다.

분기를 만들 프로그램 위치를 식별하려면 레이블이 필요하다.

레이블은 변수 이름과 동일하다.바로 뒤에 콜론이 와야 한다.

레이블은 분기가 작성될 명령문 바로 앞에 위치하며 goto와 동일한 기능 또는 메소드에 표시되어야 한다.

exit() 함수는 stdlib.h 선언 후 사용

exit(0) : 정상 종료
exit(1) : 에러 메시지 종료

일반 함수 내에 return 문을 사용하면 그 함수만 종료되지만,
exit() 는 C 프로그램 자체를 완전 종료한다.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int goals;
    printf("Enter the number of goals scored against India ");
    scanf("%d", &goals);
    if (goals <= 5)
        goto atnyla ;

    else
    {
        printf( "About time soccer players learnt C\n");
        printf( "and said goodbye! adieu! to soccer \n");
        exit(0); /* terminates program execution */
    }

    atnyla :
        printf("To err is human! \n");
}
```

Goto

goto 문을 실행하면 프로그램에서 지정된 지점으로 직접 분기가 만들어진다.

"goto"문은 CPU의 "분기 예측"에 어떤 영향을 주는 걸까?

goto는 분기 예측과 100 % 무관한 무조건 점프.
분기 예측은 조건 분기를 위한 것. (if while, for, 가상 및 함수 포인터 ..)

"CPU 파이프 라인"의 "명령 가져 오기"단계를 설명하고 있다. "분기 예측"는 명령어 가져 오기의 큰 부분이지만, 전체 가져 오기 단계에 대해 묻는 것 같다.

무조건 분기는 분기 예측자가 예측할 필요가 없기 때문에 분기 예측에게 문제가 되지 않는다.
분기 (조건부 및 무조건 부 모두)는 명령 가져 오기 단위의 복잡성을 증가시킨다.

기타

- ILP 미완료 (DLP , TLP 미 작성)
- BOM 리스트 미 작성
- goto 의 효율성 = 분기 패널티 미 작성

기타

- ILP 미완료 (DLP , TLP 미 작성)
- BOM 리스트 미 작성
- goto 의 효율성 = 분기 패널티 미 작성

회로 측정

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

int main (void)
{
	DDRB &= ~( 1 << DDB5 );
	PORTB |= ( 1 << PORTB4 );

	PCICR |= ( 1 << PCIE0 )
	PCMSK0 | = ( 1 << PCINT5 )

	sei ( );
	while (1);
	return 0;
}

ISR ( PCINT0_vect )
{
	PORTB ^= 0x10;
}
```

파이프라인

파이프라인이 왜 필요한지부터 알아야하지 않을까요 ? 여기서 어떻게 확장되는지 궁극적으로 무엇을 하려고 하는지 등등요

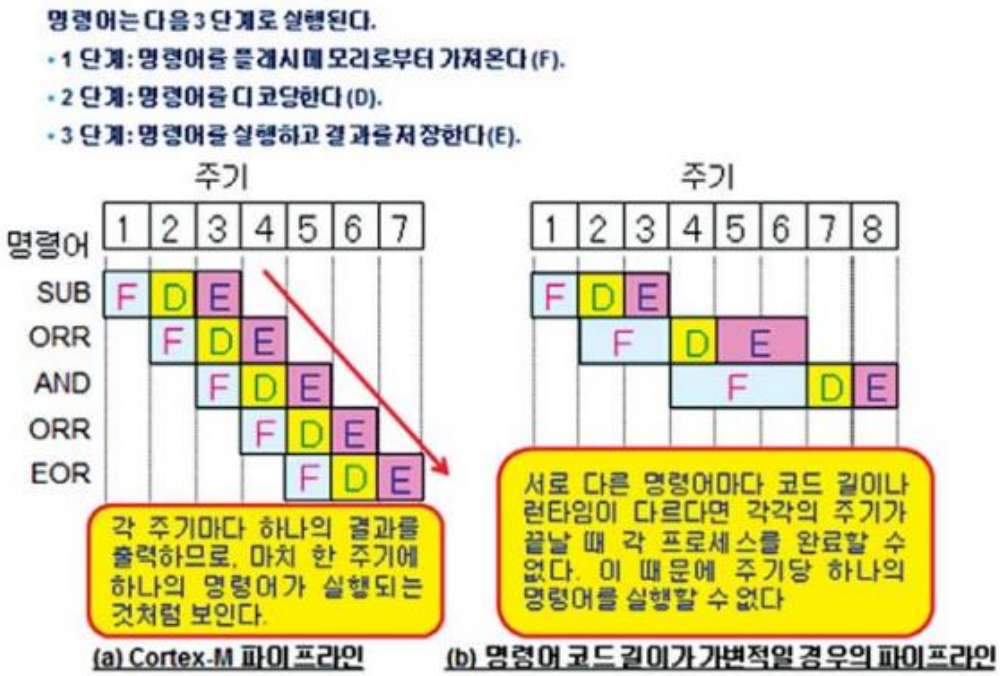
기본적인 파이프라인에 대해 조사해보도록 합니다. 추가적으로 현재 ARM 아키텍처에 적용되는 5단계 파이프라인과 9단계 파이프라인에 대해 추가적인 조사를 해보는 것입니다. 그리고 과거 프로젝트 내용 보고 프로젝트 제안서 제출하는 것 까지를 이번주 숙제 범위로 잡겠습니다.

파이프라인 장점 구현한 것.

동작 속도가 빠르다. (해석 속도가 빠르고 여러 개의 명령어를 처리하기에 적합)
파이프라인과 슈퍼스칼라 기술을 쓰면 멀티태스킹이 가능.
하드웨어를 대폭 단순화 및 효율화 시킬 수 있다. (전력 소모가 적다)
가격경쟁력 측면에서 우위를 확보한다.

C source 는 Flash 메모리에 저장한다.
Flash 에 있는 데이터를 처리하기 위해 메모리에 접근해서 읽어야 한다. (Fetch 단계)
읽어온 데이터를 기계어로 번역 (Decode 단계)
기계어로 번역한 것을 실행 (Execute 단계)

파이프라인


명령어는 다음 3 단계로 실행된다.
• 1 단계: 명령어를 플래시 메모리로부터 가져온다 (F).
• 2 단계: 명령어를 디코딩한다 (D).
• 3 단계: 명령어를 실행하고 결과를 저장한다 (E).
주기
명령어
1 2 3 4 5 6 7
SUB
ORR
AND
ORR
EOR
F D E
각 주기마다 하나의 결과를 출력하므로, 마치 한 주기에 하나의 명령어가 실행되는 것처럼 보인다.
(a) Cortex-M 파이프라인
주기
1 2 3 4 5 6 7 8
서로 다른 명령어마다 코드 길이나 런타임이 다르다면 각각의 주기가 끝날 때 각 프로세스를 완료할 수 없다. 이 때문에 주기당 하나의 명령어를 실행할 수 없다
(b) 명령어 코드 길이가 가변적일 경우의 파이프라인


FETCH
명령어 Fetch
DECODE
디코드
Reg Select
Reg Read
EXECUTE
Shift
ALU 연산
MEMORY
메모리 억세스
WRITE
Register Write

파이프라인

IF는 메모리에서 명령어를 읽어오고 PC(program counter) 혹은 IP(instruction pointer)를 다음 명령어를 지시한다.

DE는 명령어를 해독. 비트단위로 쪼개서 어떤 instruction 인지 보고 그에 해당하는 제어신호를 보내 instruction을 처리. 또한 operand로 들어온 레지스터를 read한다.

EX는 산술(덧,뺄,곱,나눗셈), 논리(AND, OR, NOT) 등을 수행. coprocessor(보조프로세서)를 이용해 캐시, MMU 등을 컨트롤하거나 SIMD, 부동소수점 연산을 수행.

MEM은 메모리 접근을 수행. Memory에 R/W하는 동작은 여기서 실행.

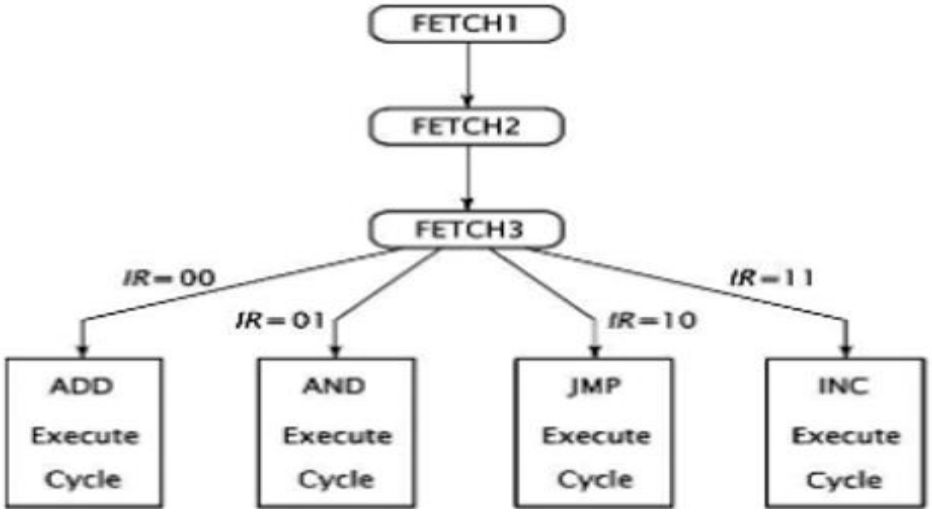
WB은 레지스터 쓰기를 수행. Mem단계에서 읽어온 메모리의 값을 레지스터에 쓰거나, EX 단계에서 add 등을 이용해 연산된 결과를 레지스터에 저장할 때 사용.

파이프라인

1. 명령어 인출 상태
FETCH1 : AR <- PC
FETCH2 : DR <- M, PC <- PC +1
FETCH3 : IR <- DR[7..6] , AR <- DR[5..0]

AC : 8비트 누산기
AR : 16비트 주소 레지스터
PC : 16비트 프로그램 카운터
DR : 8비트 데이터 레지스터
IR : 8비트 명령어 레지스터
TR : 8비트 임시 레지스터

2. 인출된 명령어를 해독하고 실행 루틴(어떤 작업을 정의한 명령어의 그룹) 결정



파이프라인

PC : Program Counter
다음에 인출할 명령어의 주소를 가지고 있는 레지스터
각 명령어가 인출된 후에는 자동으로 일정 크기(한 명령어 길이)만큼 증가
분기 (branch) 명령어가 실행되는 경우에는 목적지 주소로 갱신

AC : Accumulator
데이터를 일시적으로 저장하는 레지스터
레지스터의 크기는 CPU 가 한 번에 처리할 수 있는 데이터 비트 수 (단어 길이)

IR : Instruction Register
가장 최근에 인출된 명령어 코드가 저장되어 있는 레지스터

파이프라인 성능 저하 요인
파이프라인 하드웨어를 단순화하기 위해 모든 명령어가 stage 들을 모두 통과해야 한다.
파이프라인의 클럭은 처리 시간이 가장 오래 걸리는 stage 를 기준으로 한다.
분기 명령어가 실행되면 미리 인출하여 처리하던 명령어들이 무효된다.

파이프라인 논문

프로그램 카운터 RPC를 사용하면 명령어 캐시가 CPU에서 주소를 전송하지 않고도 연속 명령어에 액세스 할 수 있다.

분기 또는 예외가 발생한 경우에만 명령 메모리에 CPU의 주소가 필요하다.

프로세서 구현의 상대적 성능 수치는 일반적으로 MIPS (초당 수백만 명령)로 표현되며, 주기 시간 Tcycle 과 명령당 평균 사이클 수에 (CPI) 반비례한다. [average number of Cycles per Instruction]

스칼라 RISC 프로세서에 대한 CPI는 기본적으로 하나의 명령 사이클과 파이프 라인 위험으로 인한 평균 가능한 사이클 수입니다.

예를 들어, 분기 및 로드 페널티, 캐시 누락 후 지연 주기 등이 있다.

명령어 당 필요한 사이클의 수(CPI) 를 줄일 수 있다.
프로그램 흐름을 정확하게 예측함으로써 잘못 예측된 분기로 인해 발생할 수 있는 파이프라인 플러시를 최소화 한다.

기능 콜이 있을 때마다 복귀 주소가 하드웨어 스택으로 들어간다.
하드웨어가 기능 복귀 명령을 인식하면, 주소 값이 복귀 스택에서 튀어나와 예측된 복귀 주소로 사용.
정확한 복귀 주소가 확보되면 파이프라인 끝에서 해당 예측을 할 수 있다.

파이프라인 논문

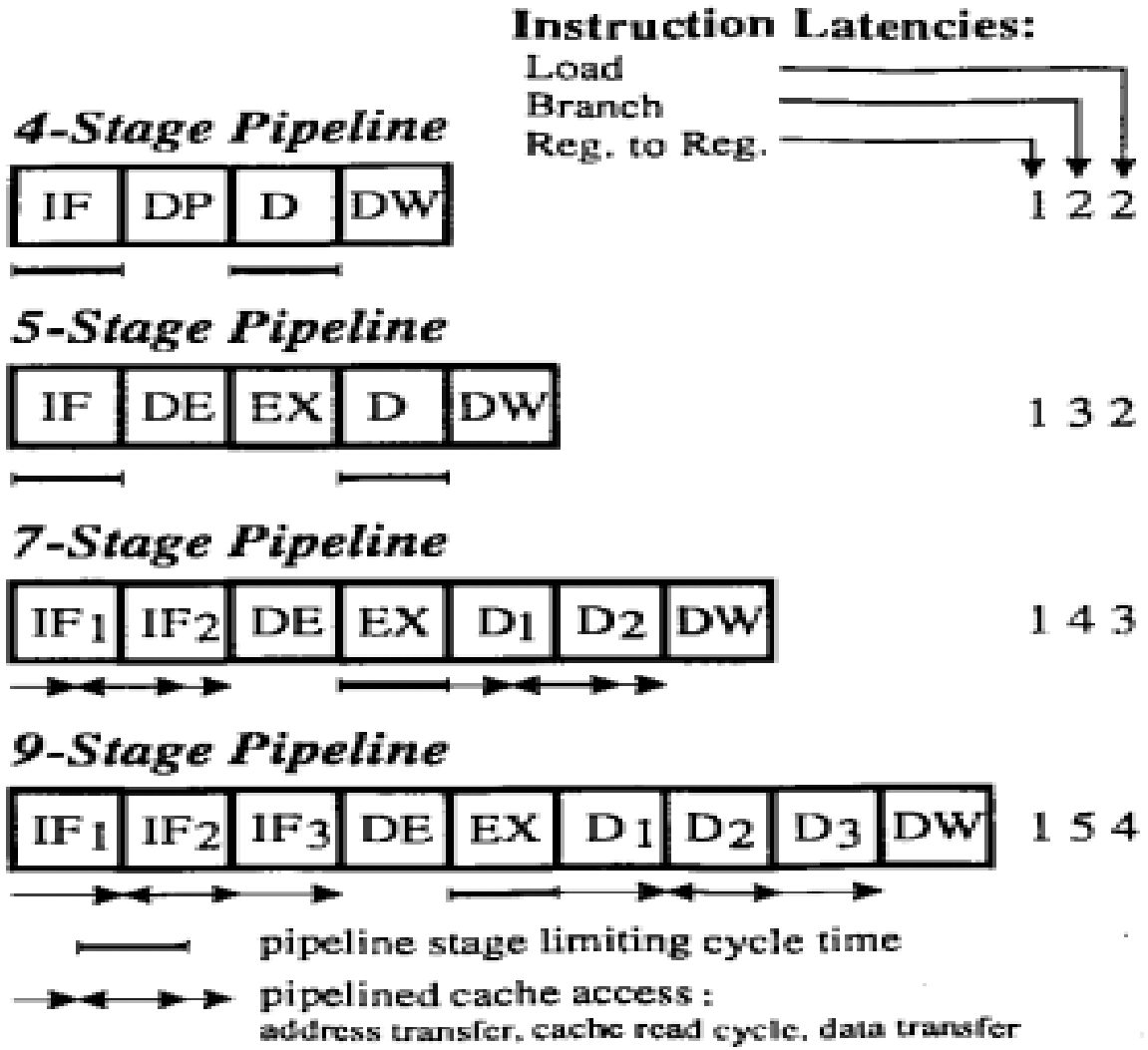


Fig. 2. Potential instruction pipeline schemes for F-RISC/I.

파이프라인 논문

그림 2 는 주기 시간과 CPI를 기준으로 평가

5단계 및 4단계 파이프라인의 주기 시간은 데이터 I/O(D) 단계에서 캐시 메모리 액세스에 의해 설정된 동일한 주기 시간으로 제한됩니다.

7단계 파이프라인은 GaAs 기술에 의해 설정된 주기 시간을 달성할 수 있으며 파이프라인 캐시 메모리 액세스도 제공하지만 주소/데이터 전송을 위한 CPU 사이클은 절반에 불과하다.

9단계 파이프라인은 주소/데이터 전송에 대해 하나의 전체 사이클을 할당하고 캐시 메모리 액세스에 대해 하나의 CPU 사이클을 할당하여 캐시 크기를 늘리고 캐시 누락에 따른 불이익을 최소화하였다.

파이프라인 논문

TABLE I
CPI CONTRIBUTIONS DUE TO BRANCH AND LOAD PENALTIES

Instruction	*Instruction frequency* [1]	*Cost 4- stage*	*Cost 7-stage*	*Cost 9-stage*
ALU	60%	1	1	1
BRANCH (not taken)	5%	1	1	1
BRANCH (taken) [2]	15%	1.368	2.673	3.673
LOAD [3]	15%	1.1	1.5	2.3
STORE	5%	1	1	1
$CPI_{intrinsic}$		1.07	1.326	1.596

1 Average dynamic instruction mix [8].

2 Cost of a taken branch depends upon the branch latency slot fill-in probability of the compiler. The compiler branch fill-in probabilities are taken from [7]. (4-fill = 0, 3-fills = 36.8%, 2-fills = 20.2%, 1-fills=16.5% and 0-fill = 26.5%).

3 Cost of a load instruction depends upon the load latency slot fill-in probability of the compiler. The compiler load latency slot fill-in probabilities are taken from [5].

파이프라인 논문

표 1
일반적인 UNIX 프로그램 집합에서 파생된 동적 명령어 조합에 대한 분기 및 로드 지연 시간의 CPI 기여도

주소/데이터 전송 주기가 길면 MCM에 더 큰 캐시를 구현할 수 있는데, 이는 사용 가능한 주소/데이터 전송 시간 내에 더 많은 SRAM 칩에 도달할 수 있기 때문이다.
그러나 주기 시간이 길수록 (4 단계 파이프 라인) 최대 명령어 실행 속도가 느려집니다.

주소 / 데이터 전송을 위해 더 많은 주기 / 단계 (7 단계 및 9 단계 파이프 라인)를 허용하면 load 및 분기 페널티가 증가한다.

7 단계 및 9 단계 파이프 라인의 1단계 캐시 크기는 주로 열 관리, 네트워크 라우팅 / 토폴로지, 할당된 주소 / 데이터 전송 시간의 세 가지 요소에 따라 달라집니다.

파이프라인 논문

TABLE II
PERFORMANCE VERSUS PIPELINE DEPTH

Pipeline Depth	T_{cycle} *(Relative)*	$CPI_{intrinsic}$	CPI_{cache_miss}	*Performance (Relative)*
4-stage	1	1.07	0.188	1
7-stage	0.625	1.326	0.202	1.317
9-stage	0.625	1.596	0.140	1.159

[1] Cache miss ratios are based on SPEC92 benchmark trace data [11].

파이프라인 논문

표 II는 세 가지 파이프라인 체계의 상대적 성능을 비교.

7 단계 파이프 라인은 4 단계 및 9 단계 파이프 라인보다 성능이 좋다.

9 단계 파이프 라인은 cache miss 페널티가 가장 낮지만 분기 /load 페널티가 크다.

4 단계 파이프 라인은 분기 / load 페널티가 가장 낮고 캐시 누락 페널티가 상당히 적지만 ,
주기 시간이 길수록 파이프라인 효율성이 저하된다.

파이프라인 논문

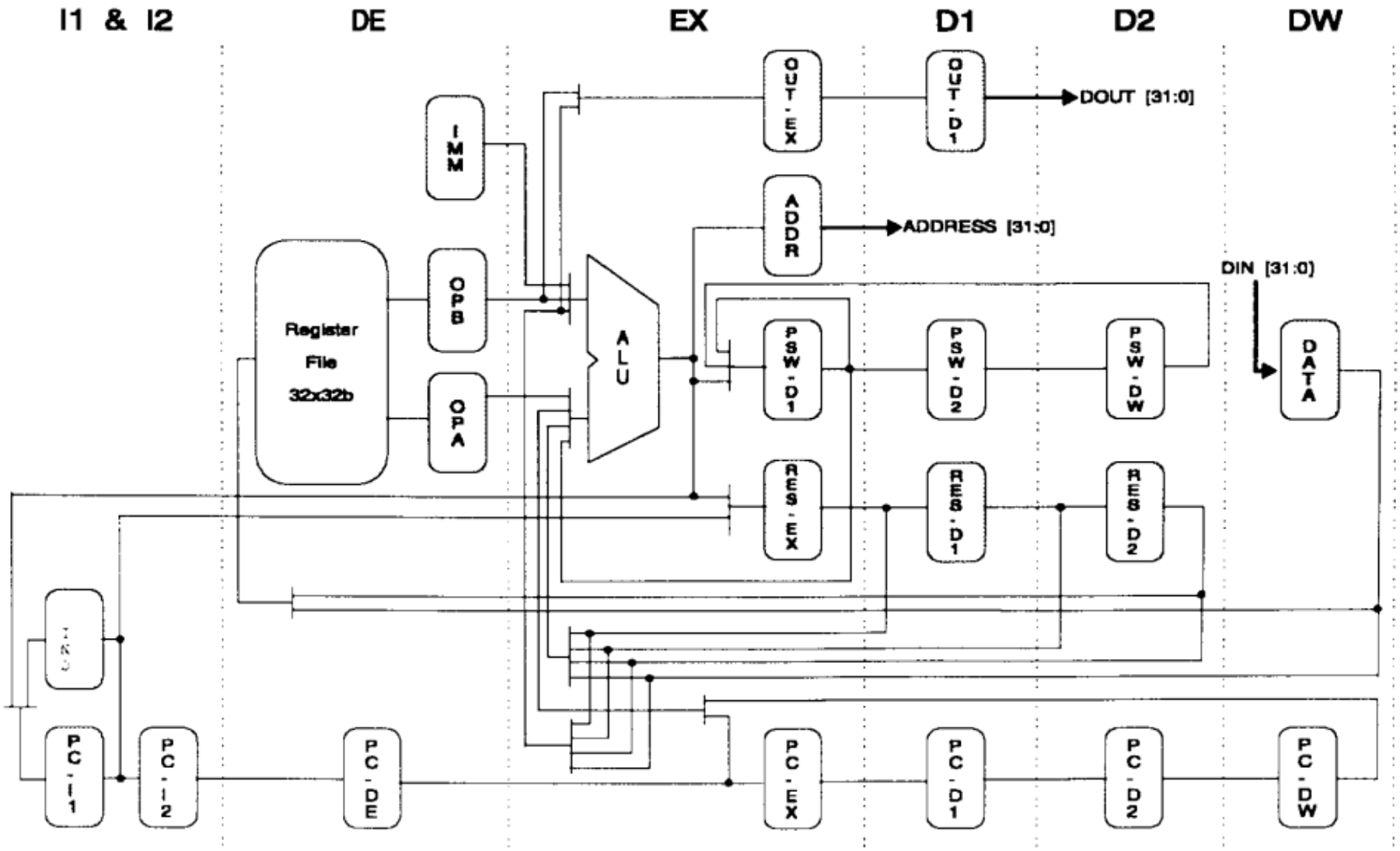


Fig. 7. FRISC/I datapath.

파이프라인 논문

TABLE III
COMPARISONS OF CRITICAL PATH DELAYS

Critical Path	*Simulated Delay*	*Measured Delay*
PC increments in I1 stage	4.21 ns	5.0 ns
Register File Read in DE stage	3.12 ns	3.36 ns
ALU Execution in EX stage	5.17 ns	6.0 ns
LSSD latch Delay	1.42 ns	1.48 ns

그림 7은 F-RISC / I의 데이터 경로

중요한 경로는 I1 단계의 PC 증가, DE 단계의 레지스터 파일 읽기, ALU 실행 및 EX 단계의 결과 feed-forward 다.
가장 긴 경로는 결과 레지스터 (RES EX)의 출력에서 시작하여 멀티플렉서와 ALU를 거쳐 RES EX의 입력에서 종료된다..
이 중요한 경로는 ADD 명령이 이전 명령의 결과를 필요로 할 때 사용

표 III 시뮬레이션 결과와 측정 사이의 비교

파이프라인 논문

분기 패널티란?

왜 존재하는가? 위험을 방지하기 위해?

파이프라인을 통해 명령을 얻는 데는 몇 번의 클럭 사이클이 필요하며, 분기 명령은 파이프라인의 중간 지점까지 분기 목표를 결정하지 못할 수 있다. 따라서 대상이 결정되고 다음 명령이 가져올 때까지 파이프라인을 정지해야 한다.

특히 대상 명령이 캐시에 없는 경우(아키텍처에 캐시가 있다고 가정) 이 작업에는 여러 주기가 걸릴 수 있다.)

분기 예측이 정확하지 않으면 ALU가 조건부 분기 또는 간접 분기의
이동 방식을 결정할 때까지 가져오기 작업은 다음에 무엇을 가져올지 알 수 없다.

따라서 ALU에서 분기가 실행될 때까지 대기.

잘못된 예측으로 인해 잘못된 경로에서 가져온/암호 해독된 명령은 무용지물이므로
분기 잘못 예측 페널티라고 합니다.

분기 예측은 일반적인 경우 숨긴다.

또 다른 용어는 "분기 지연 시간".
분기 명령을 가져올 때부터 front-end fetches가 유용한 다음 명령을 가져올 때까지의 사이클 수입니다.

명령이 분기라는 사실은 해독된 후에 알 수 있다.
실행보다 파이프라인에서 더 이전이므로 조건부 또는 간접 분기에 비해 더 작다.

무조건적인 분기에서도 분기 지연 시간이 있다.

파이프라인 논문

로드 사용 페널티란 무엇을 의미?.

로드를 시작할 때부터 데이터를 실제로 사용할 수 있을 때까지의 주기 수입니다
현대의 슈퍼스칼라 비순차 시스템에서는 프로세서가 일반적으로 이 지연 시간을 잘 숨긴다.

메모리에서 레지스터를 로드하는 명령 A가 있을 수 있다.
추가의 피연산자로 레지스터를 사용하는 후속 명령 B가 있을 수 있다.

첫 번째 명령이 완료되는 주기와 두 번째 명령이 완료되는 주기의 최소 간격은 "사용할 부하" 페널티이다.

L1 캐시(대부분)에서 발생하는 메모리 작업의 경우 이 숫자는 약 3 사이클이 될 수 있습니다. L2 히트, L3 히트 또는 메인 메모리의 경우 더 클 수 있다.

캐시 읽기 데이터를 ALU로 직접 우회하므로 메모리 읽기 데이터를 레지스터 파일에 저장한 다음 다시 읽을 필요가 없다.

파이프라인의 슈퍼스칼라(superscalar) 설계에서 가능한 모든 작업을 병렬로 수행한다, 가상 주소를 물리적 주소로 변환하고, 캐쉬 어레이로 주소를 가져오고, 태그를 비교하고, 패리티를 확인하고, 읽기 데이터를 정렬하고, 결과 데이터를 다시 가져오는 작업에는 정말 엄청난 양의 작업이 있다. ALU로요

파이프라인 논문

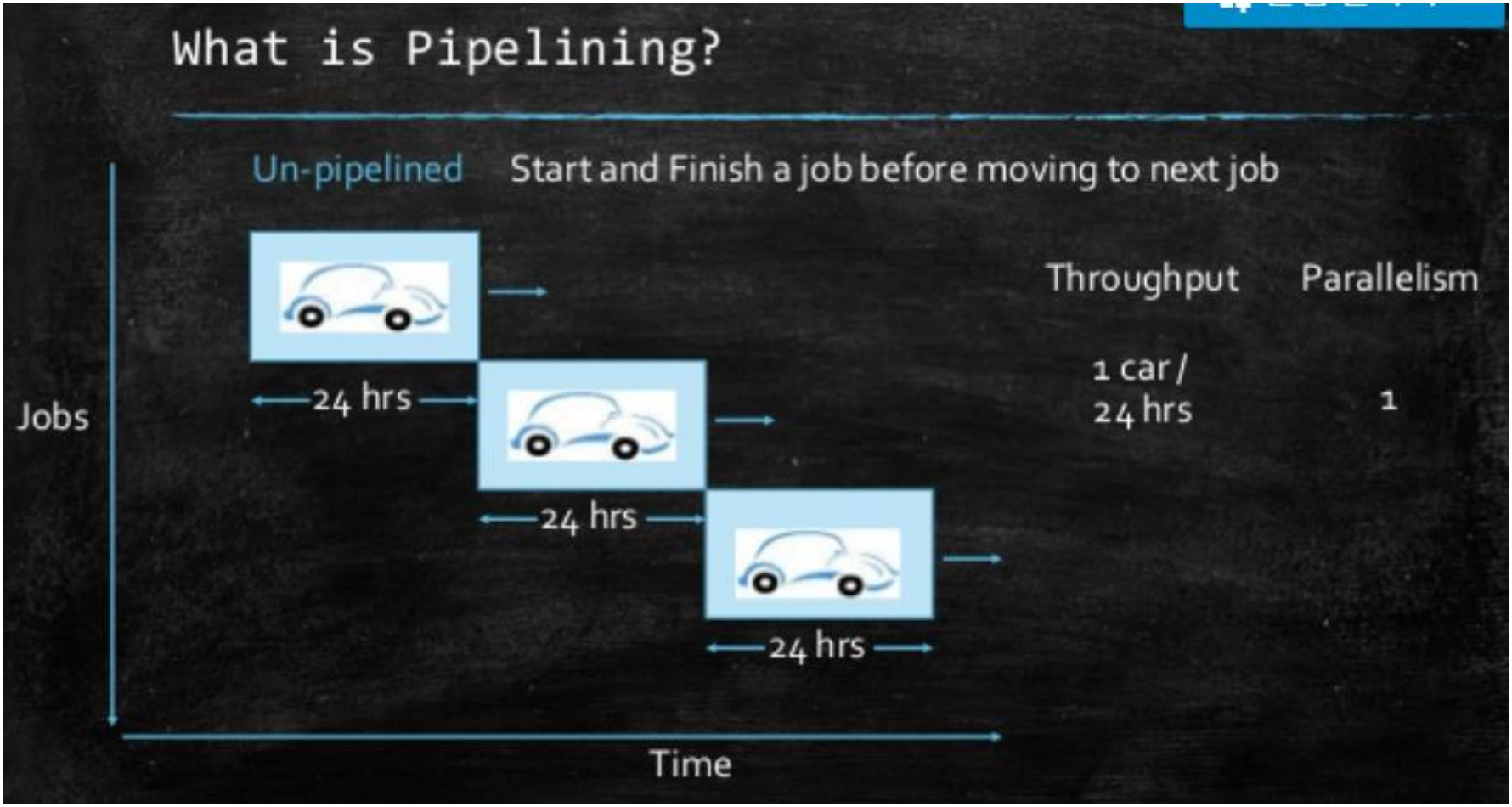
일반적인 데이터 위험은 특히 짧은 파이프라인에서 상당히 빠르게 우회할 수 있다

두 번째 명령이 실행되려면
L1 데이터 캐시(일반적으로 여러 사이클의 지연 시간) 또는 RAM/기타 캐시 레벨에서 데이터를 검색해야 합니다.

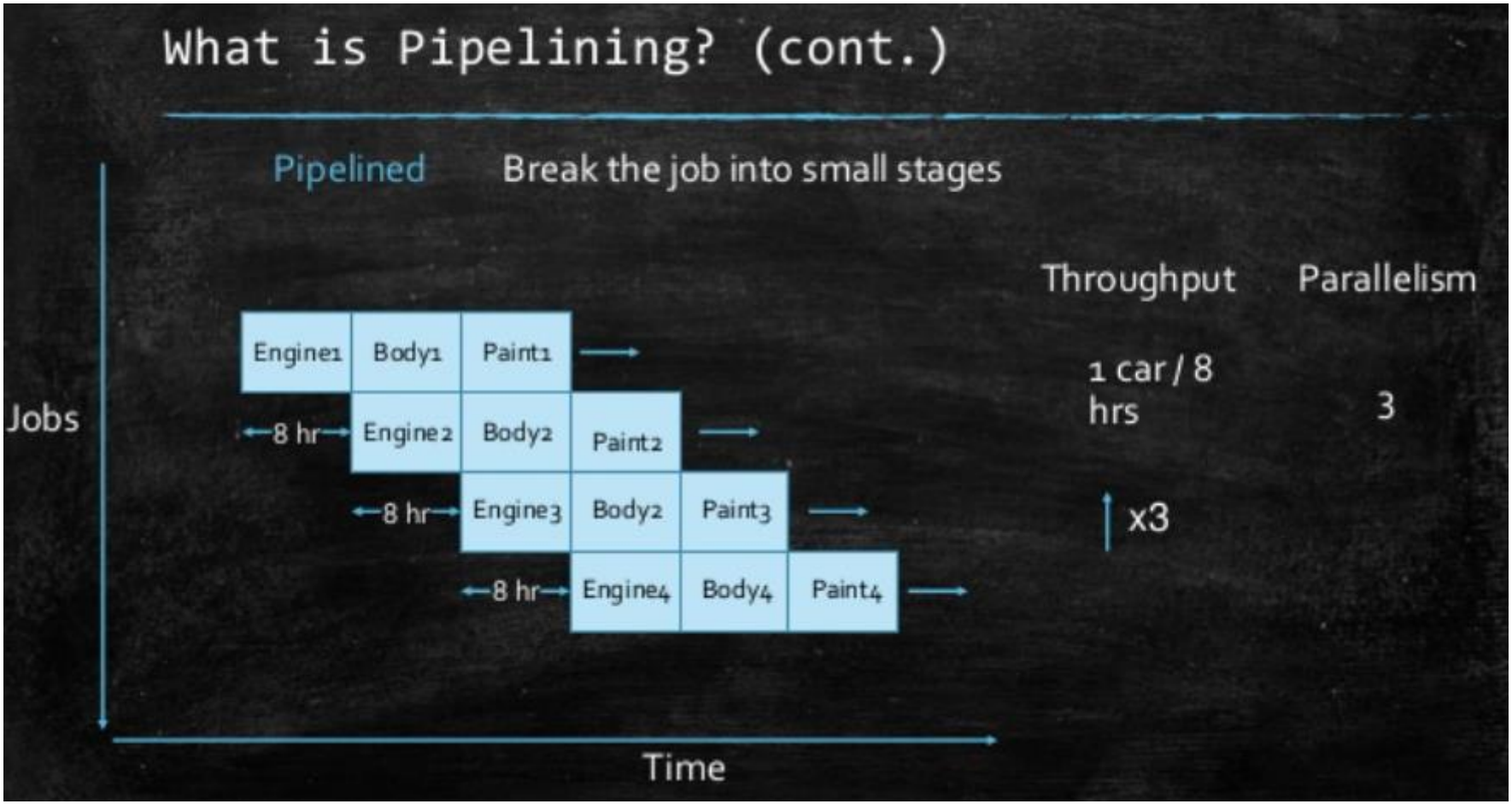
"단순한" 데이터 위험보다 훨씬 더 많은 지연이 있으며, 상당히 변동할 수 있습니다(RAM 접근은 L1 접근보다 약 2배 느림).

이것이 바로 로드 스토어의 페널티입니다.
로드 명령과 이를 피연산자로 사용하는 후속 명령 사이에 낭비되는 주기는 얼마나 됩니까?

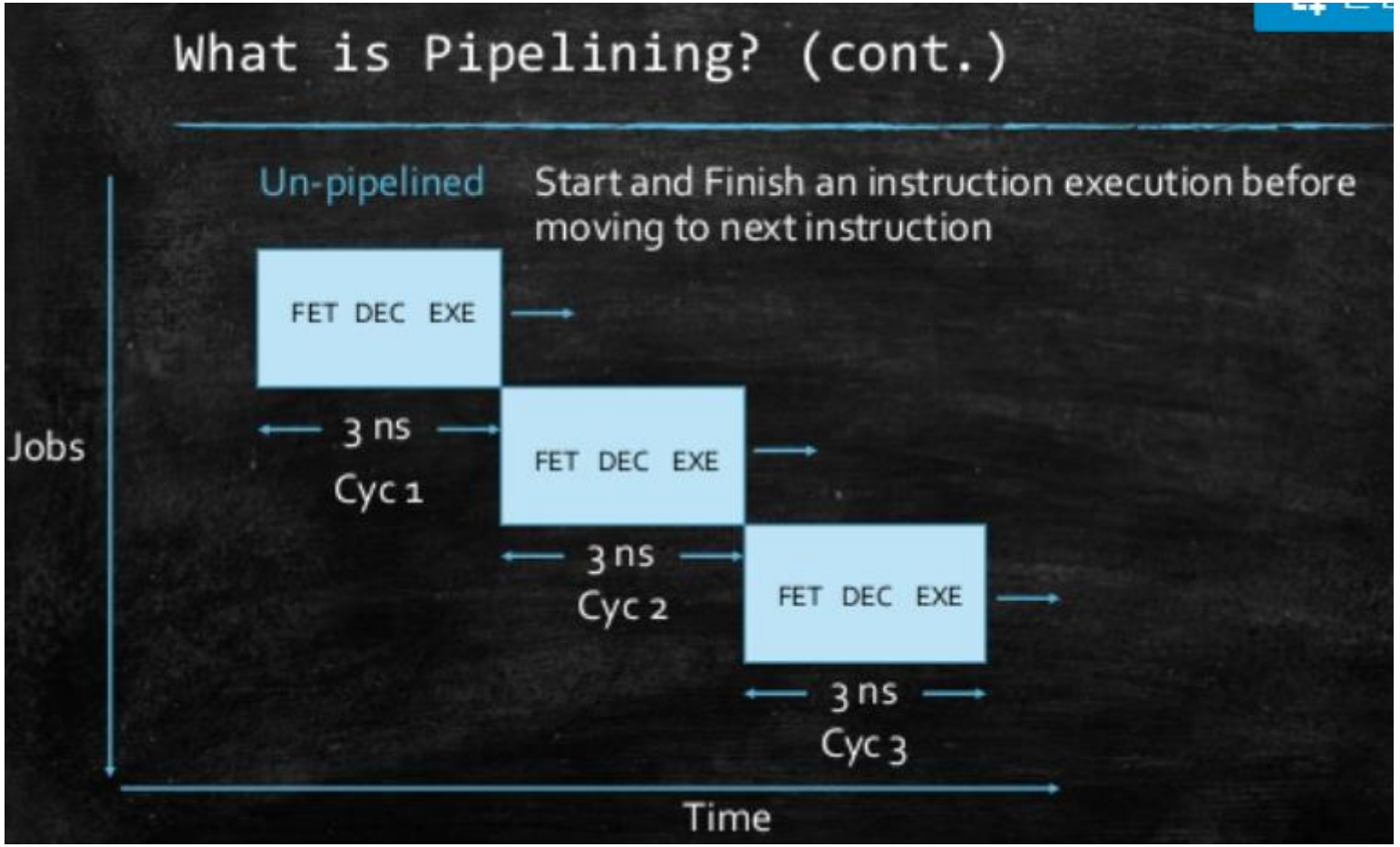
기타


What is Pipelining?
Un-pipelined
Start and Finish a job before moving to next job
Throughput
Parallelism
1 car /
24 hrs
1
24 hrs
24 hrs
24 hrs
Jobs
Time

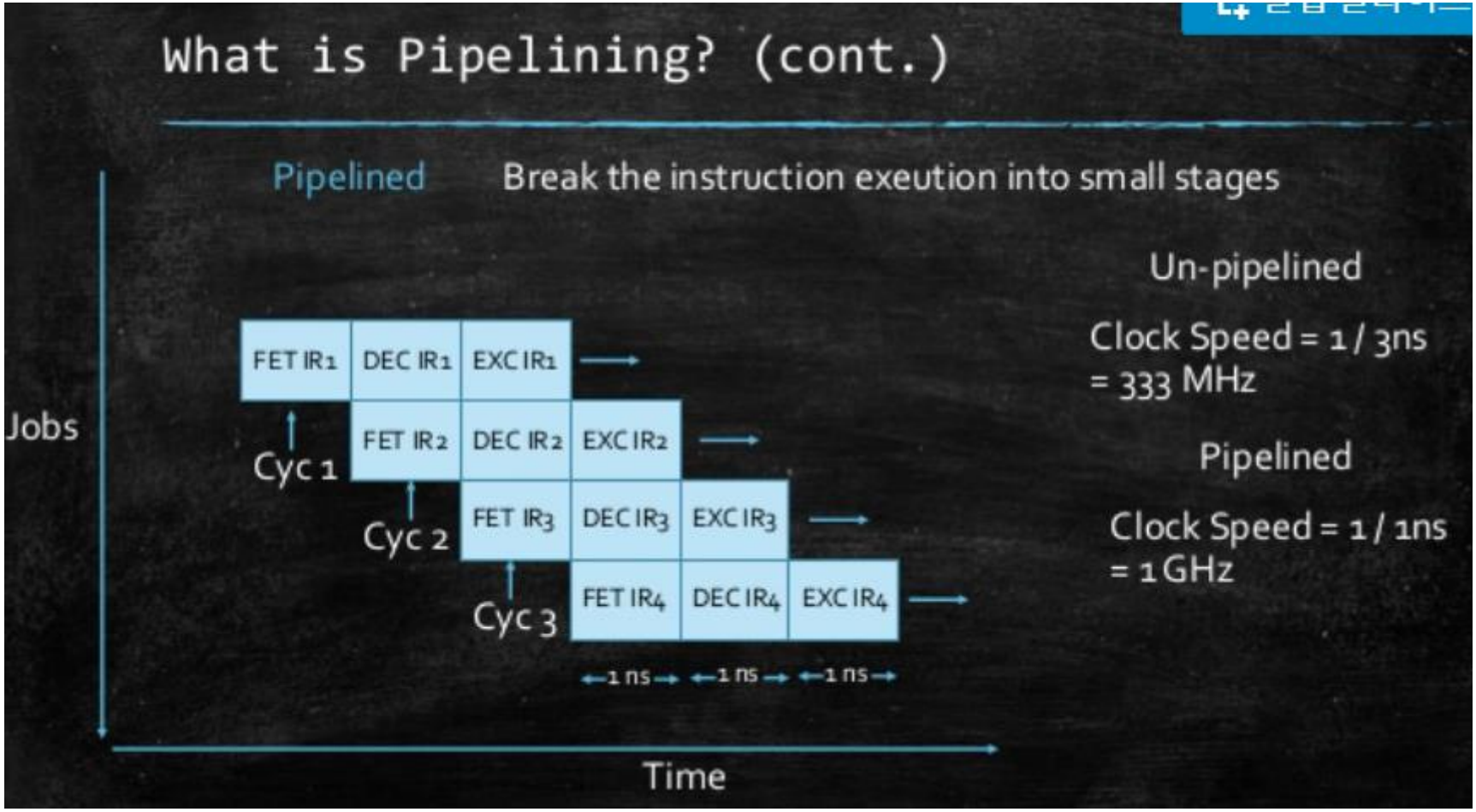
기타


What is Pipelining? (cont.)
Pipelined
Break the job into small stages
Throughput
Parallelism
1 car / 8 hrs
3
x3
Jobs
Time
Engine1
Body1
Paint1
8 hr
Engine2
Body2
Paint2
8 hr
Engine3
Body2
Paint3
8 hr
Engine4
Body4
Paint4

기타


What is Pipelining? (cont.)
Un-pipelined
Start and Finish an instruction execution before moving to next instruction
FET DEC EXE
3 ns
Cyc 1
FET DEC EXE
3 ns
Cyc 2
FET DEC EXE
3 ns
Cyc 3
Jobs
Time

기타


What is Pipelining? (cont.)
Pipelined
Break the instruction exeution into small stages
Un-pipelined
Clock Speed = 1 / 3ns
= 333 MHz
Pipelined
Clock Speed = 1 / 1ns
= 1 GHz
Jobs
Time
FET IR1
DEC IR1
EXC IR1
FET IR2
DEC IR2
EXC IR2
FET IR3
DEC IR3
EXC IR3
FET IR4
DEC IR4
EXC IR4
Cyc 1
Cyc 2
Cyc 3
1 ns
1 ns
1 ns

1. 프로젝트 소개

product name

LED 제어

Function

전원부는 5V , 12V , 24V 中 택1

약 5 ~ 10 개 정도의 LED 를 순차적으로 ON/OFF 하며 반복하는 것.

LED 의 ON 시간과 OFF 시간을 버튼을 통해 사용자가 설정.

ON / OFF 시간 및 카운터는 LCD 또는 7-segment 를 통해 숫자를 표기.

카운터 설정을 통해 반복되는 구간을 설정.

LED 구간을 정해 반복 작동하기 위한 값으로 카운터가 있다.
- 5 로 설정하면 1 ~ 5 라인만 반복.
- 10 으로 설정하면 1 ~ 10 라인만 반복

2. 프로젝트 일정 및 계획

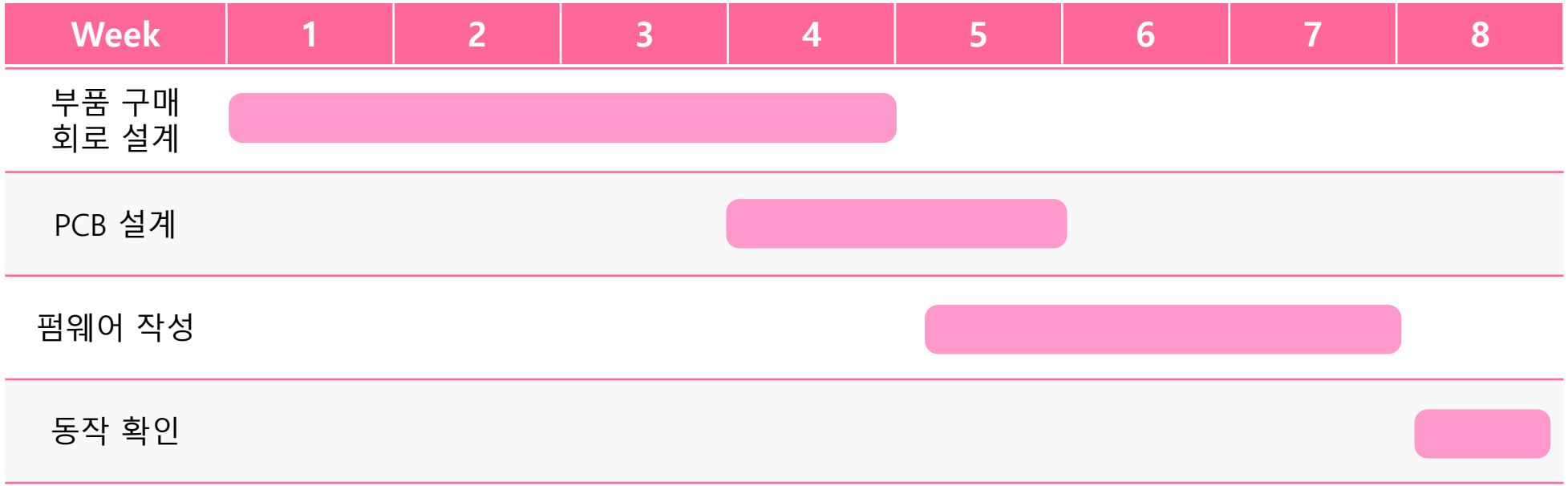
부품 구매 : 판다파츠 또는 디바이스 마트에서 부품 구매.

회로 설계 : 오실로스코프 , 함수 발생기 , 디지털 멀티미터 2개 사용.

PCB 설계 : 만능기판으로 납땜.

펌웨어 작성 : Function 을 C 코드로 구현.

동작 확인 : 장시간 동안 동작해도 문제 없는지 확인. (디버깅)



3. BOM 리스트

일련번호	품목	규격	소요량	비고
1	KEYPAD	Adafruit Industries 3845	2	
2	MCU	ATMEGA328P-PU	2	
3	MCU	ATMEGA4809-PB	3	
4	PROGRAMMER	PG164140	1	
5	IC - CMOS	CD4532	1	
6	IC - CMOS	CD4042B	1	
7	IC - CMOS	74HC85	1	
8	IC - Regulator	LM7805	1	
9	IC - CMOS	74HC02	1	
10	IC - CMOS	CD4518B	1	
11	IC - CMOS	74HC42	1	
12	IC - CMOS	CD4516B	1	
13	IC - TTL	74LS47	1	
14	Timekeeping	DS1302	1	
15	timer	NE555P	2	
16	timer	NA556N	1	
17	RS232	MAX232CPE+	1	

3. BOM 리스트

18	MOSFET - P	IRF9Z24PBF	5	
19	MOSFET- N	IRF630PBF	5	
20	TR - NPN	2N3904BU	3	
21	DIODE	1N4148	4	소신호
22	DIODE	1N5231B	3	제너
23	DIODE	1N4007	4	정류
24	콘덴서	세라믹, 전해 등등	?	
25	저항		?	
26	Crystal	Crystals WE-XTAL 16.0MHz 50ppm	1	
27	부저	KPX1205A	1	
28	LED	10 Segment Light Bar Graph LED Display -Red	1	
29	7 segment	FND507	6	
30	Switches	1825360-3	1	
31	만능기판	-	1	

End of Document