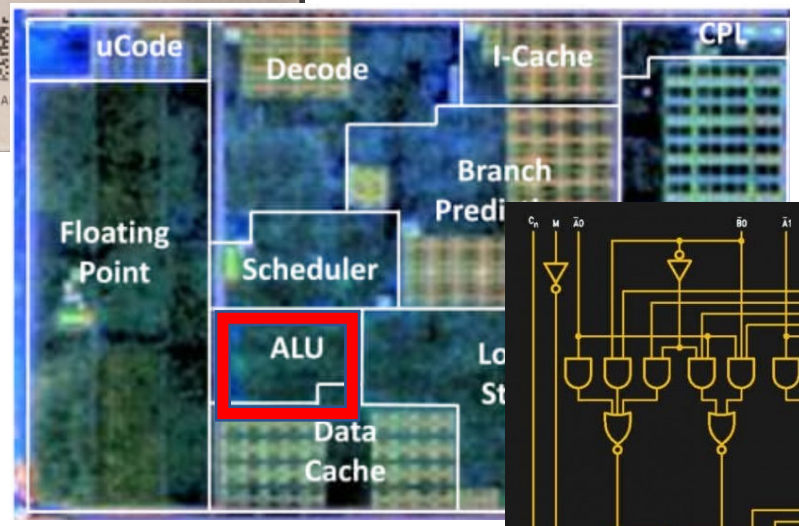
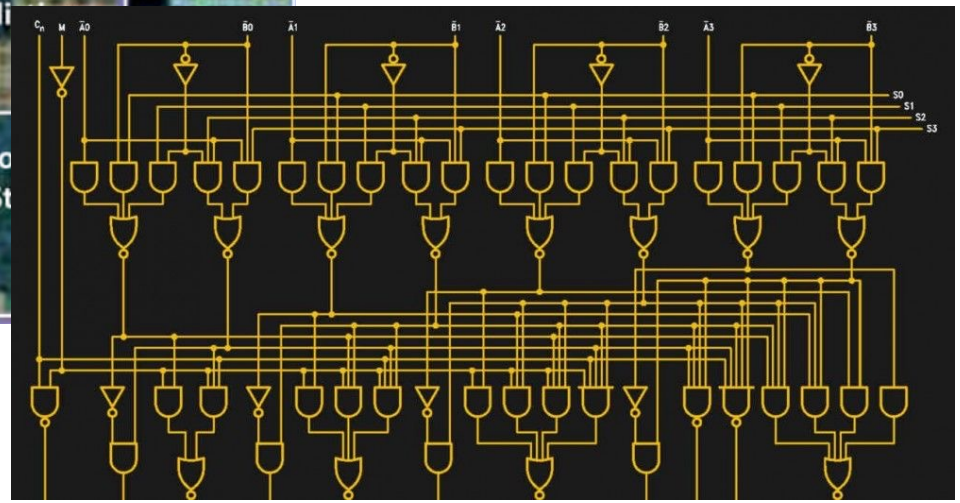
STEVE
CAPSTONE DESIGN


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DESIGN
STEVE
Topic & Purpose
Plan
Exit
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


1
TOPIC AND PURPOSE

TOPIC AND PURPOSE





ALU (Arithmetic Logic Unit: 산술 논리 장치)

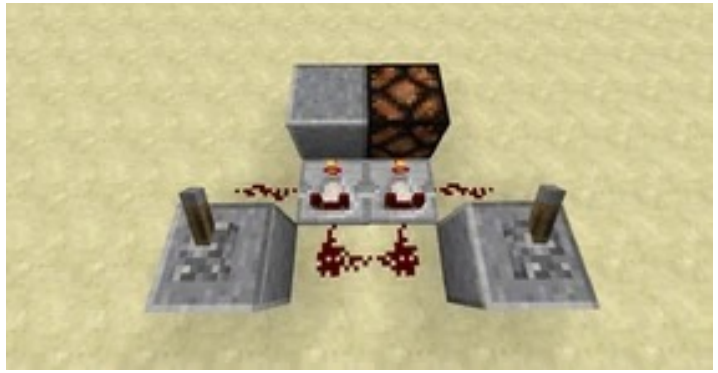
CPU 내에서 산술연산(+, − , *, /)와 논리연산(&&, ||, <<, >>) 계산을 수행하는 디지털 회로

1 TOPIC AND PURPOSE

ALU (Arithmetic Logic Unit: 산술 논리 장치)

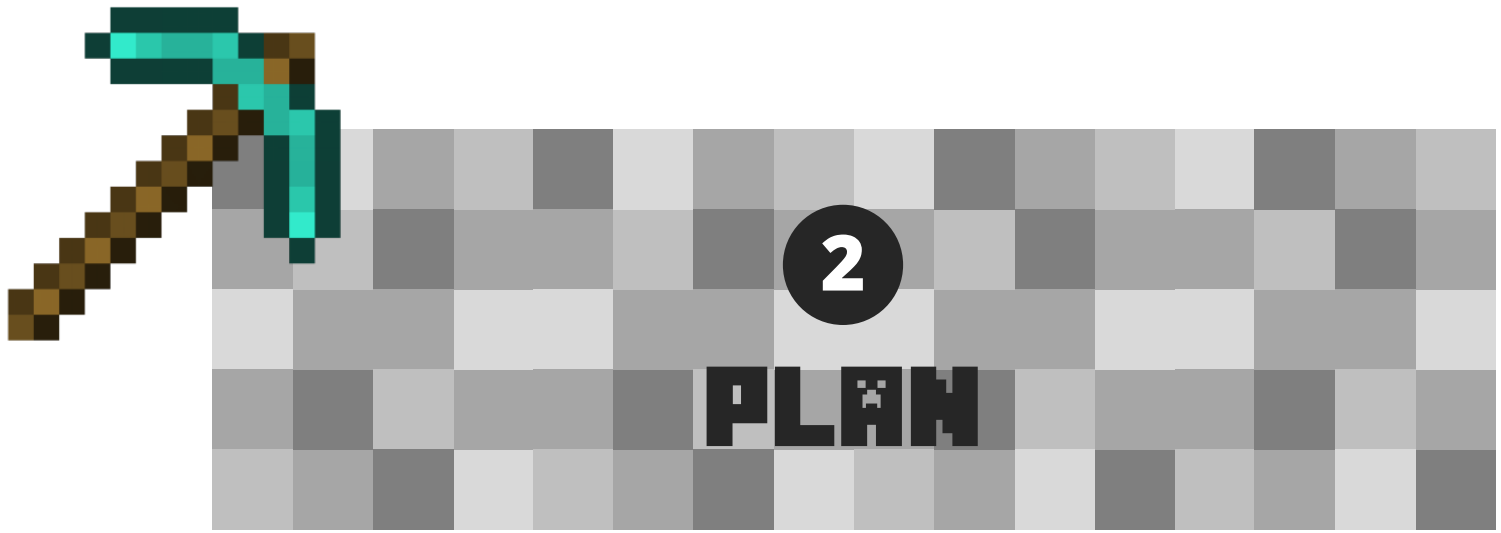
CPU 내에서 산술연산(+, − , *, /)와 논리연산(&&, ||, <<, >>) 계산을 수행하는 디지털 회로

XOR GATE

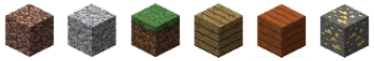


MINECRAFT 를 이용하여 실제 연산을 수행하는 계산기를 만들고,

실제 구동 원리를 이해한다.

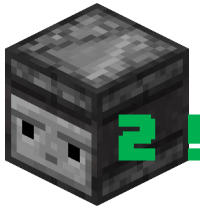

2
PLAN

PLAN



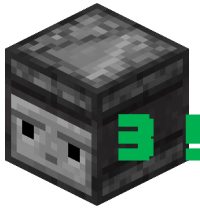
1 STEP

계산기에 사용되는
실제 논리 회로와
이를 마인크래프트
내에서 표현하는
방법(레드스톤공학)
을 익힌다.



2 STEP

논리 회로 조합을 이용한,
가산기·감산기·곱셈기
·비교기를 구현한다.

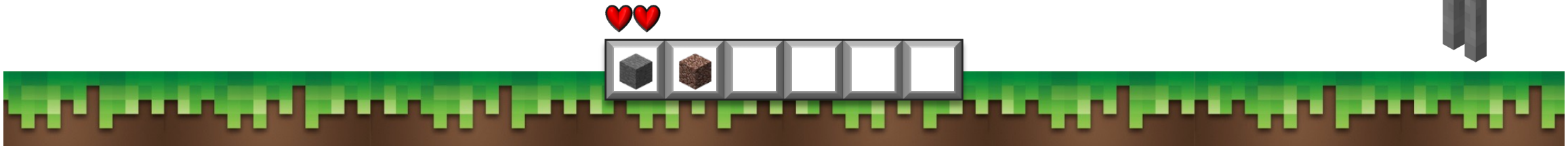


3 STEP

논리 회로 조합을 이용한,
10x2 인코더,
2x10 디코더를 구현한다.

4 STEP

디 코 더 를 통 해 나 온
결과값을 출력할 수 있는
디스플레이를 구현한다.

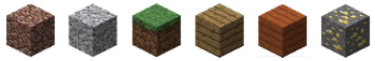


TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재현
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DESIGN
마인크래프트 소개
마인크래프트 레드 스톤 공학
논리회로 구현
Exit
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

1 마인크래프트란?

마인크래프트란?



Mojang(모장) 스튜디오에서
2009년 처음 발매한
샌드박스 형식의 비디오 게임

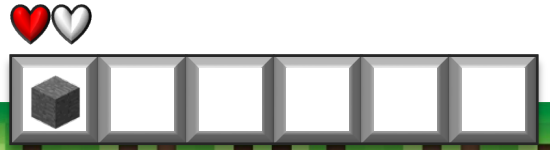
Enderman
김병수

Wither
김상혁

Creeper
한동휘

Skeleton
김재헌

Piglin
최정훈

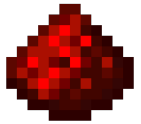


2 RED STONE

Red stone

회로구현에 필요한 주요 아이템

입력(Input)



레드스톤 가루 (RedStone Dust) : 전기를 공급하는 전선 역할

레드스톤 블록

지속적인 전원 공급

레버

전원 공급 On/Off

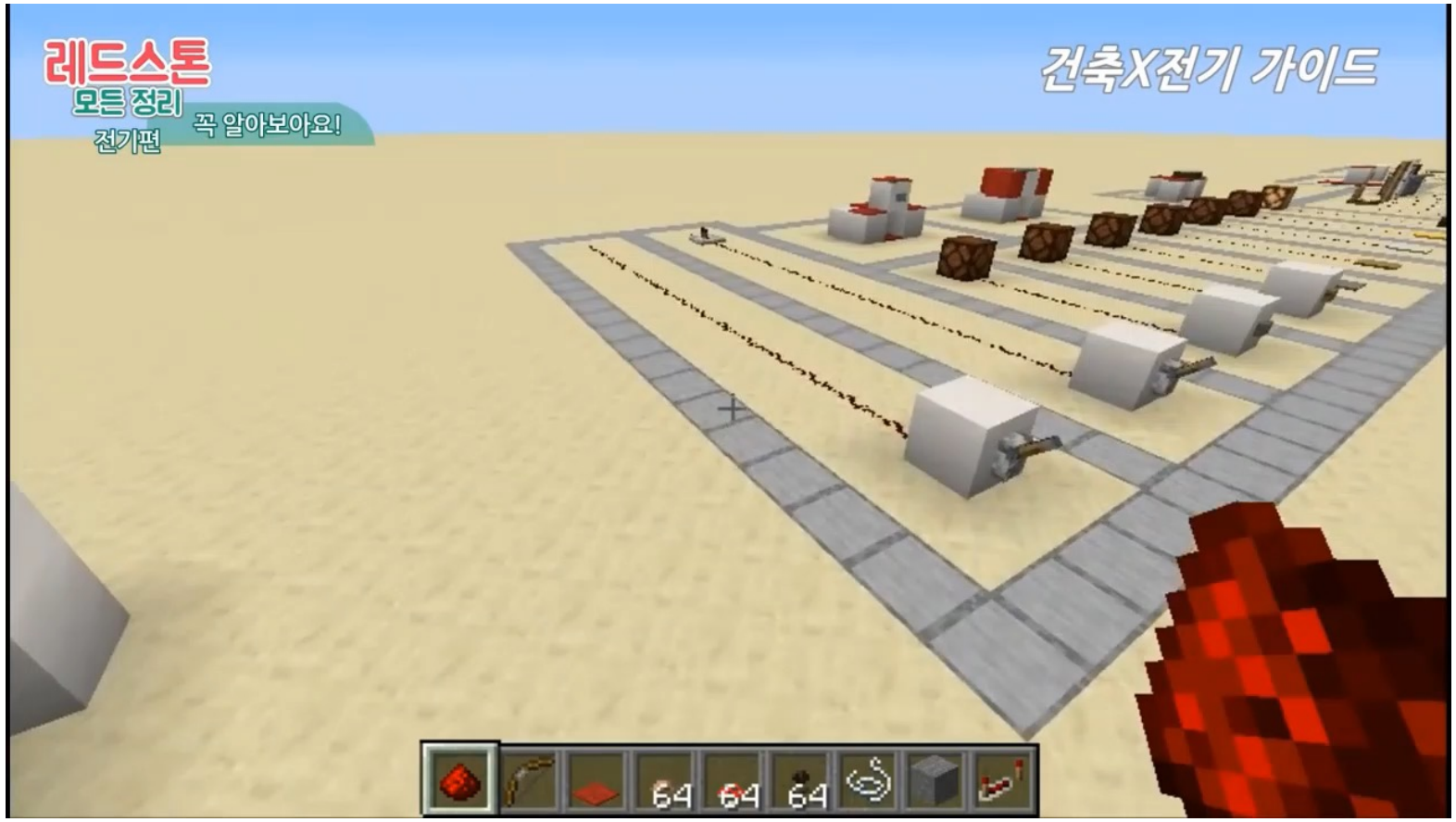
관측기

블록 업데이트가 발생되면 그것을 감지하여 관측기의 뒷면으로 신호를 방출.

레드스톤 횃불

전원 공급

2 Red stone



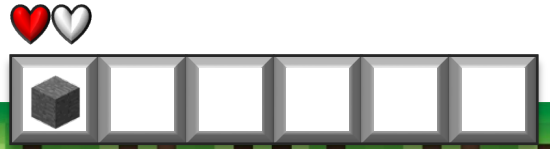
출처 : 유튜브 채널 건축X전기 가이드

회로구현에 필요한 주요 아이템

출력(output)

피스톤
전력이 공급되면,
블록을 한 칸 민다.

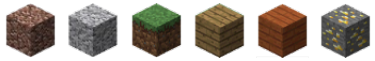
레드스톤 조명
전력이 공급되면,
빛을 낸다.



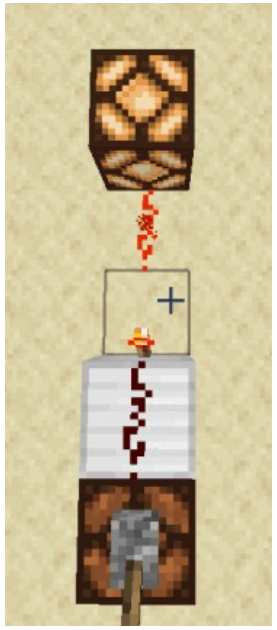
3

RED STONE 회로

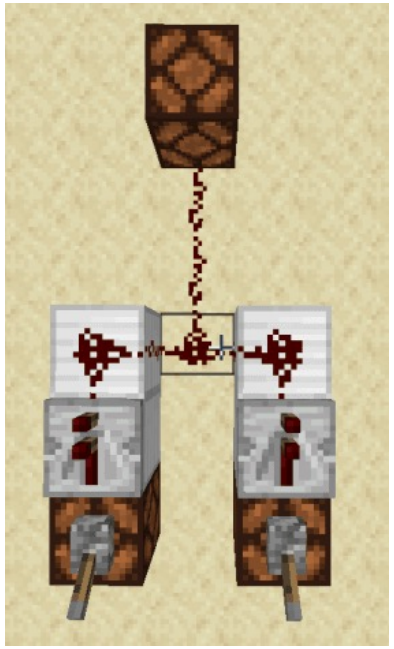
Logic gate



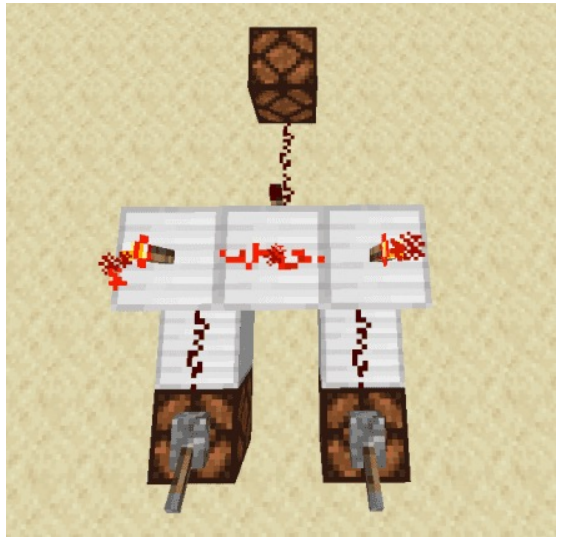
NOT



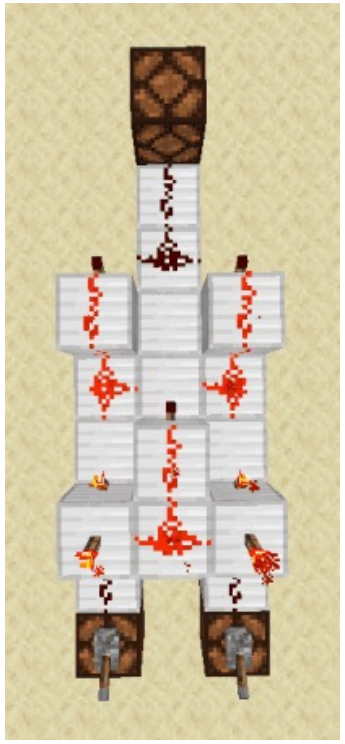
OR

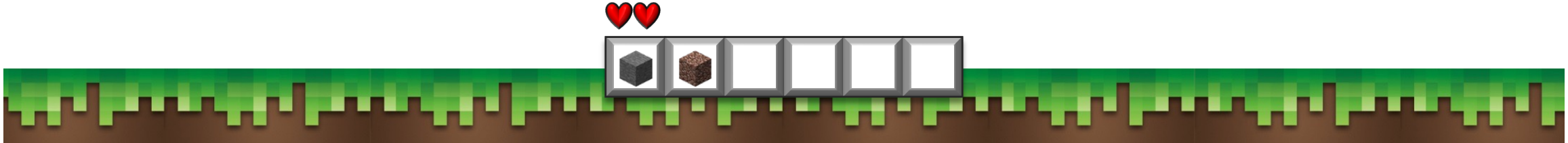


AND

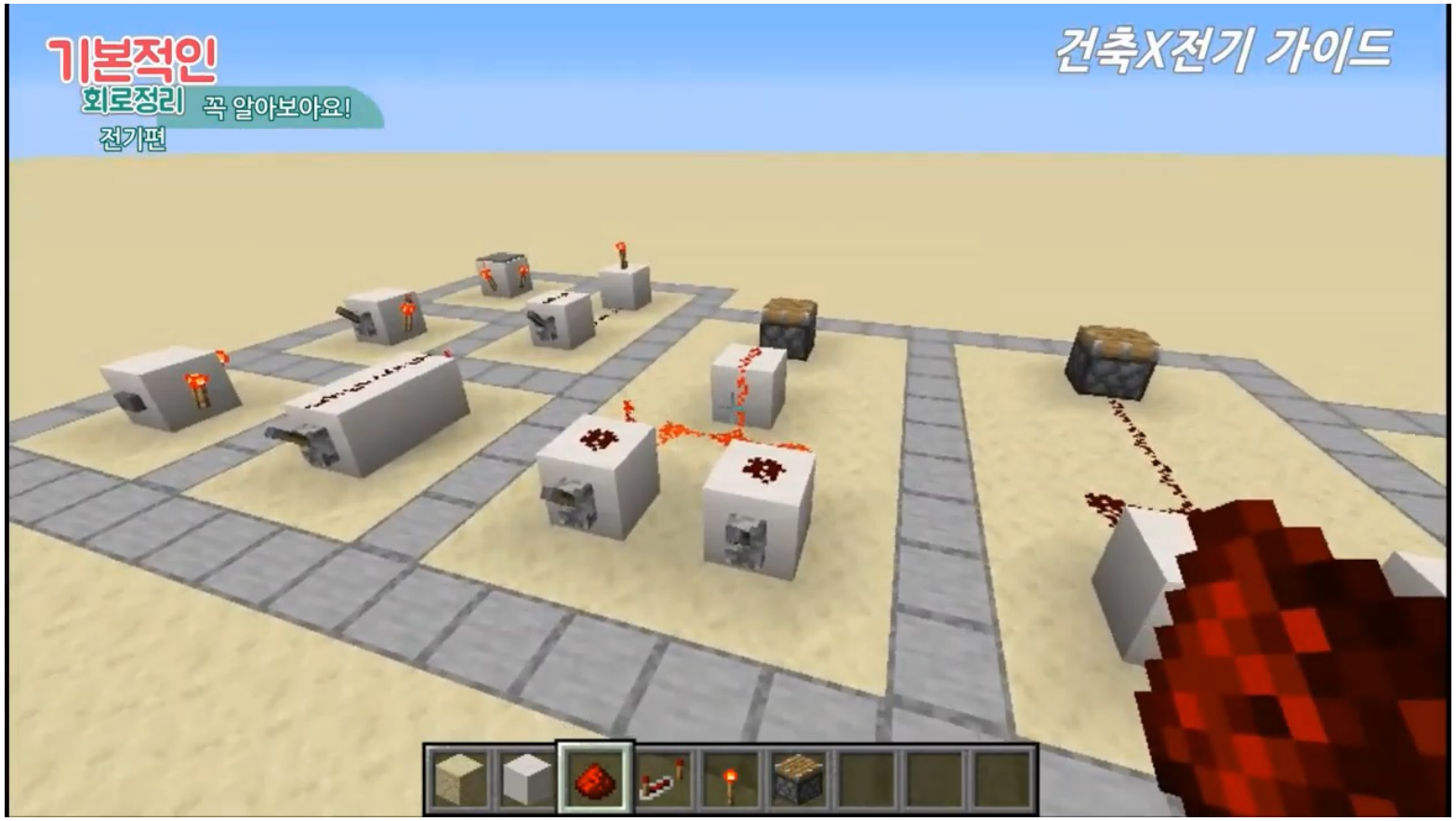


XOR





Logic gate



출처 : 유튜브 채널 건축X전기 가이드

Logic gate

출처 : 유튜브 채널 SpeedyStyle

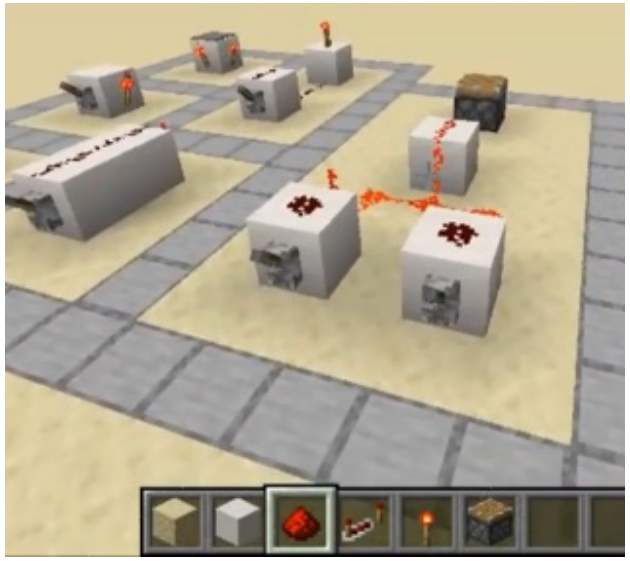
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재현
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DESIGN
지난주 내용 복습
앞으로의 계획
Exit
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

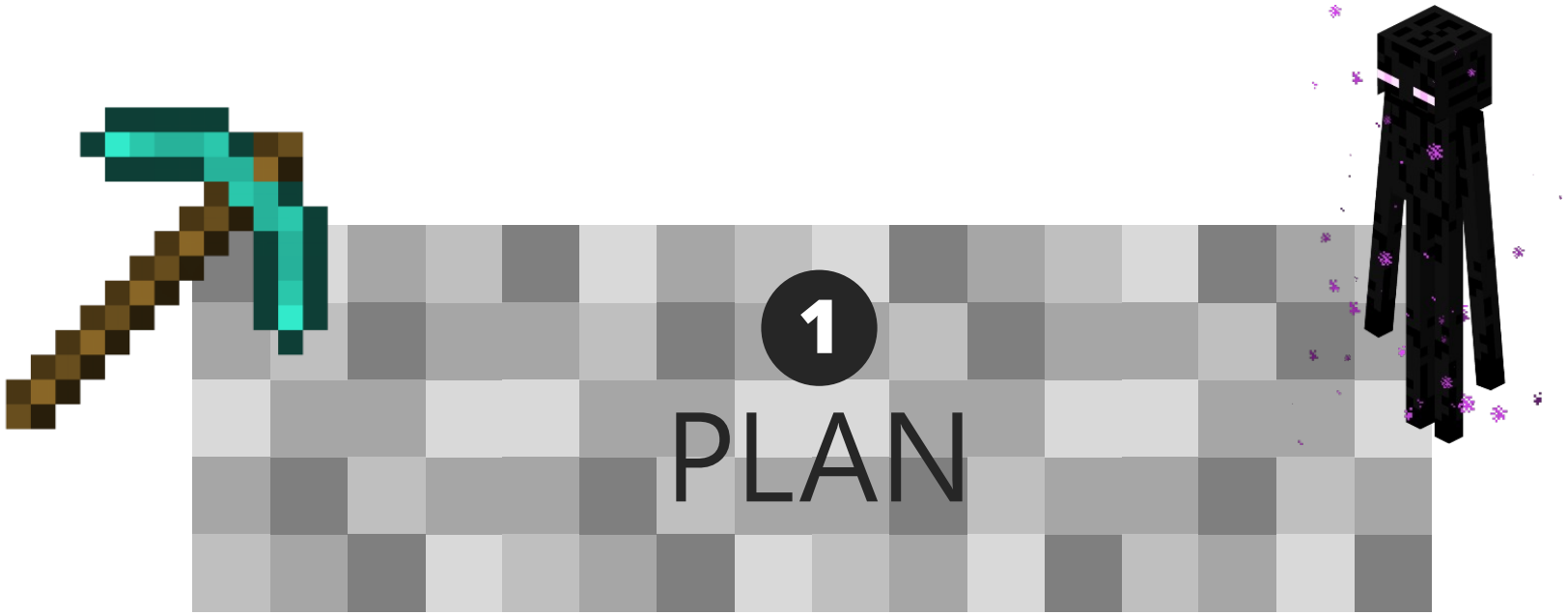
저번주에는?

아이템

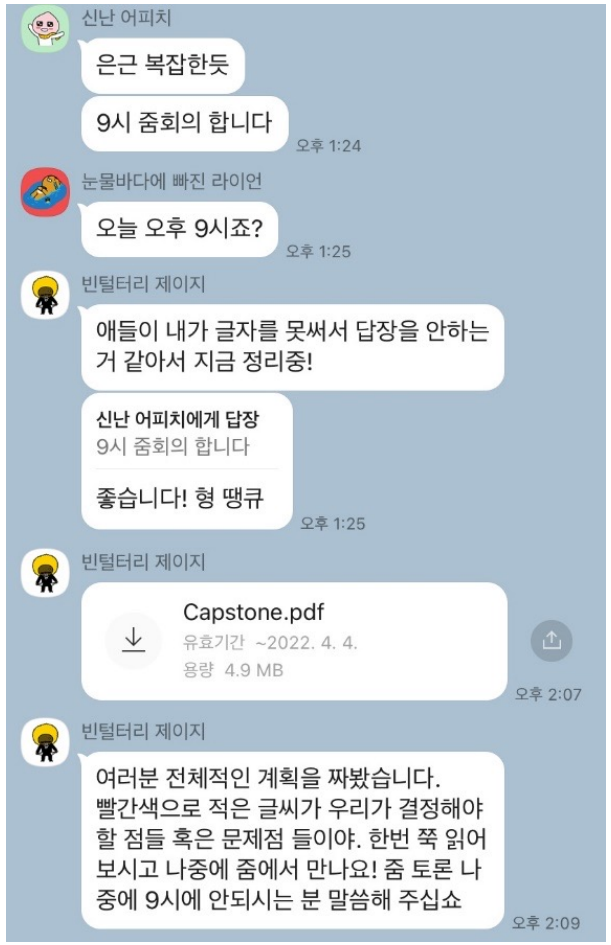
논리 회로



최종 목표


1
PLAN

2 PLAN



3월 21일 오후 9시
4주차 회의

15주 개괄적인 계획 수립

프로젝트 시작 전 결정해야할 사안

PLAN

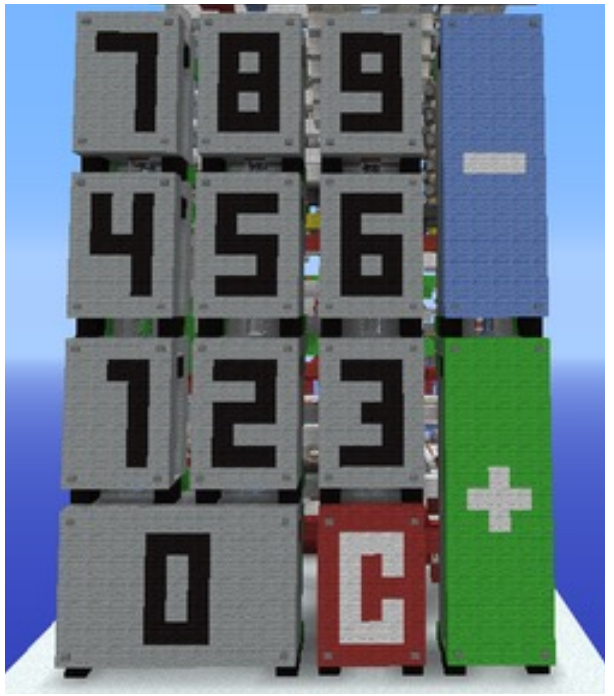
~중간고사(~8주차) :
회로 공부
계산기(8bit)

입력-(부호화)-연산-(복호화)-출력

1. 입력과 출력 : 2진법 OR 10진법? > 부호화 & 복호화 필요
2. 첫번째 숫자를 입력하고 이후 과정을 진행하기 위한 저장소
3. 부호화와 복호화 방법

입력

0~9버튼 + 자리 변환 버튼 + 입력버튼 + 연산자별 버튼(+,-,*,/ 예정) + 연산



PLAN

연산 : 가산기, 감산기, 곱셈기, 나눗셈기

가산기

감산기 : 가산기 + 2의 보수

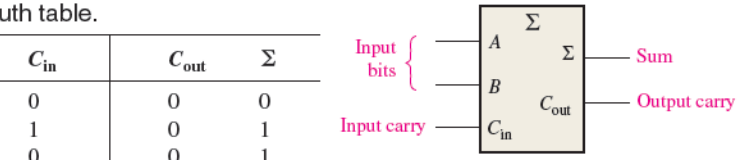
곱셈기 : 덧셈 연산 + 자릿수

나눗셈기

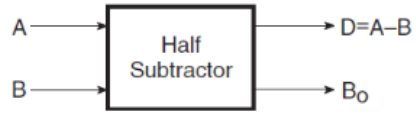
Full-adder truth table.

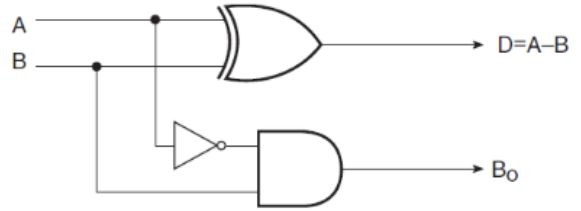
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

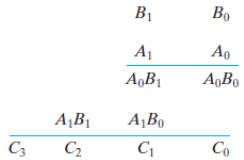
C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

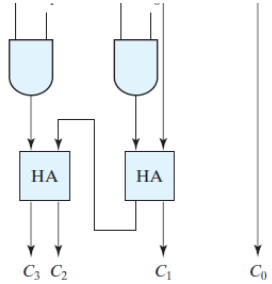


A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

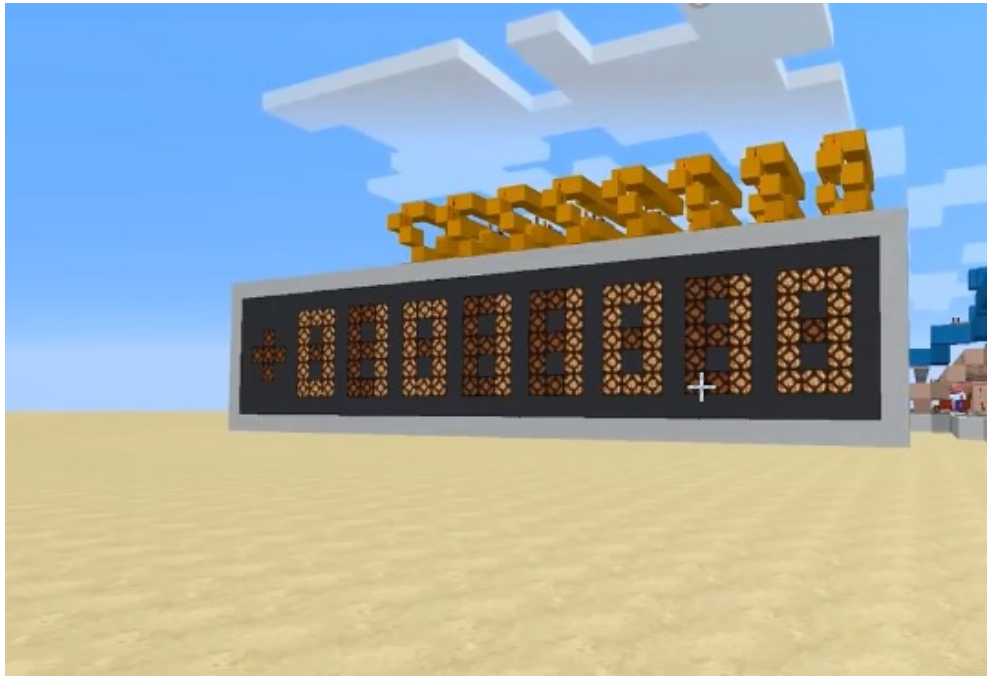








출력 : 디스플레이



Logic gate

~중간고사(~8주차) :
회로 공부
전원 각 주차에 맞는 이론 각자 공부 후 마인크래프트에서 실습 /
발표는 순차적으로(인당 3번)

4주차 : 계획 수립 및 발표 — 김병수

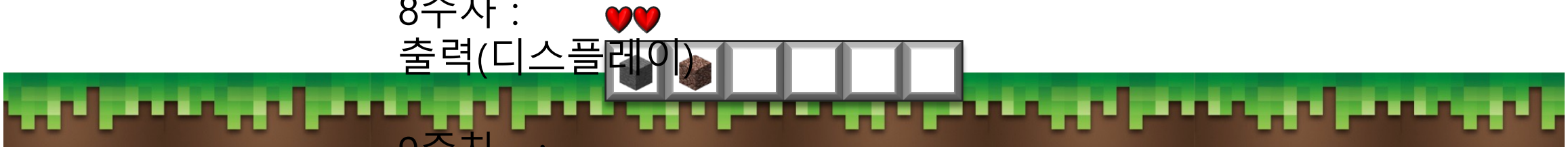
5주차 : 입력 구현 — 최정훈/한동희

6주차 : 가산기/감산기 — 김상혁

7주차 : 곱셈기/나눗셈기 — 김재헌

8주차 : 출력(디스플레이) — 최정훈/한동희

9주차 : 추후배정



TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재현
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DEGIGN
수체계
논리회로
전체 회로도
입력과 출력
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

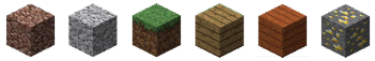

1
목차

목차

1

수체계

수체계



①여러가지 수체계

10진수(Decimal number)

8진수, 16진수, n진수

양자컴퓨터

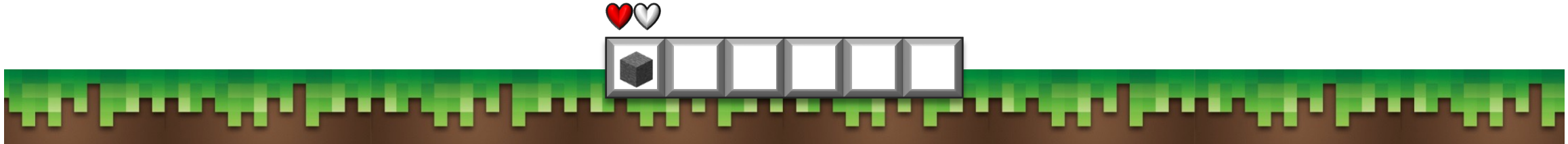
②2진법 수체계

UNSIGNED CHAR

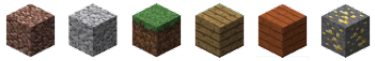
SIGNED CHAR

BCD

Grey code



수체계



16진수, n진수

8진수는 **2진수 3bit 정보**를 group화 한다
16진수는 **2진수 4bit 정보**를 group화 한다

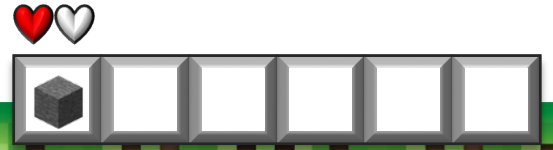
효율!!

2진법	10진법	16진법
0	0	0
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9

1010	10	A
1011	11	B
1111	15	F
10000	16	10
10001	17	11
10010	18	12

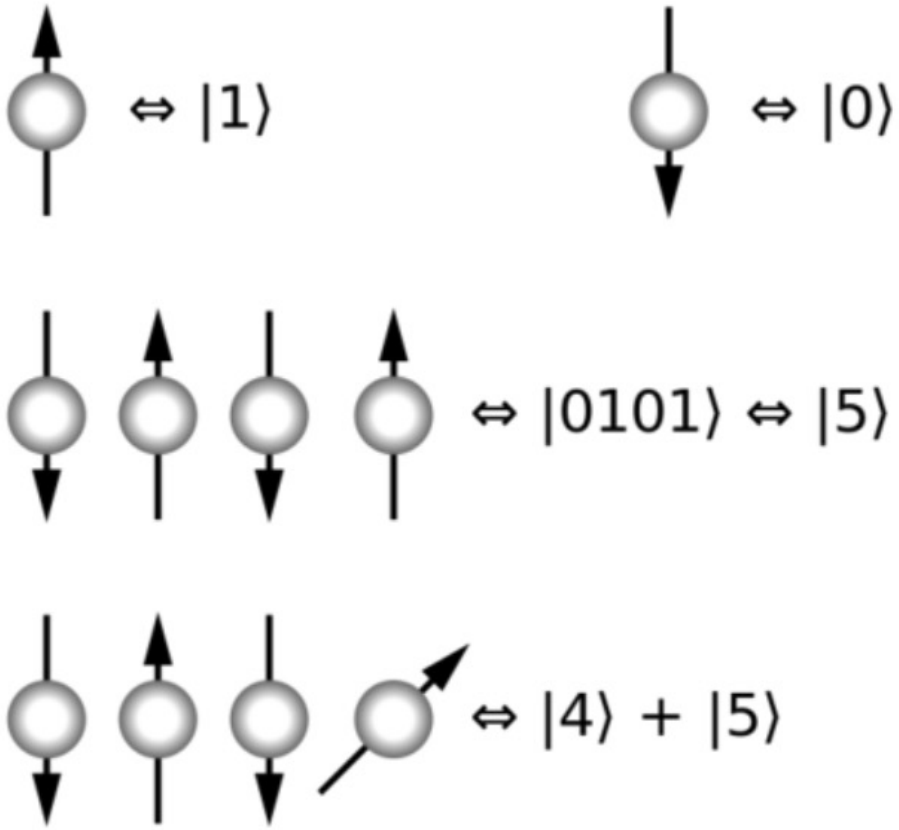
$$A_nA_{n-1}\cdots A_1A_0.A_{-1}\cdots A_{-m}$$

$$= A_nR^n + A_{n-1}R^{n-1} + \cdots + A_1R^1 + A_0R^0 + A_{-1}R^{-1} + \cdots + A_{-m}R^{-m}$$



수체계

양자컴퓨터

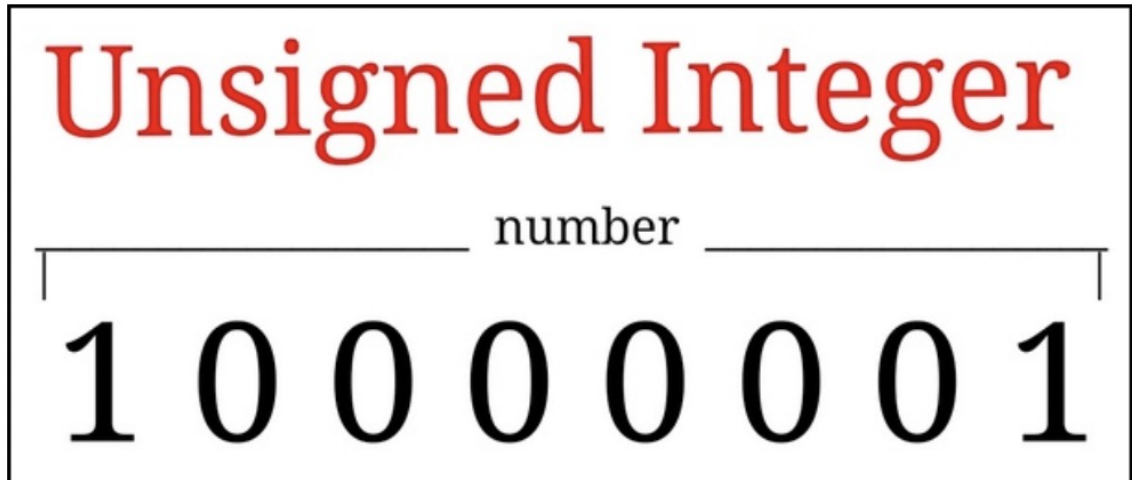


- 양자역학적 물리현상을 활용
- 큐비트라는 일종의 수체계
- 상태의 중첩으로 여러 연산을 동시적으로 처리가능

수체계는 너무나 중요한 개념!!

2진수 수체계

UNSIGNED CHAR(부호가 없는 8비트)



- 2진법의 기본표기
- 8비트에서, 0~255까지 숫자표현

2진수 표현 체계

2진수 수체계

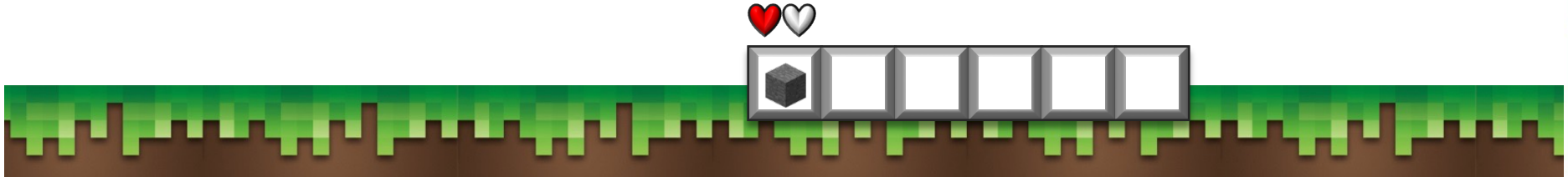
SIGNED CHAR (부호가 있는 8비트)

7+(-7)

=0111 + 1001 = 1 0000

장점: 덧셈회로만으로 뺄셈도 수행가능

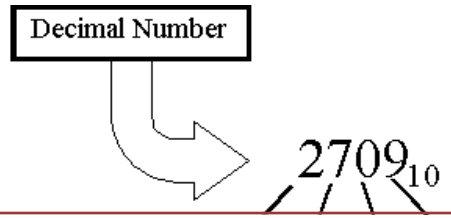
2의 보수	
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1



2진수 표현 체계

2진수 수체계

BCD (Binary-coded decin



십진수 한자

13 = 0001 0011

이 특성이 회로를 최적하는데 매우 중요

10진수	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

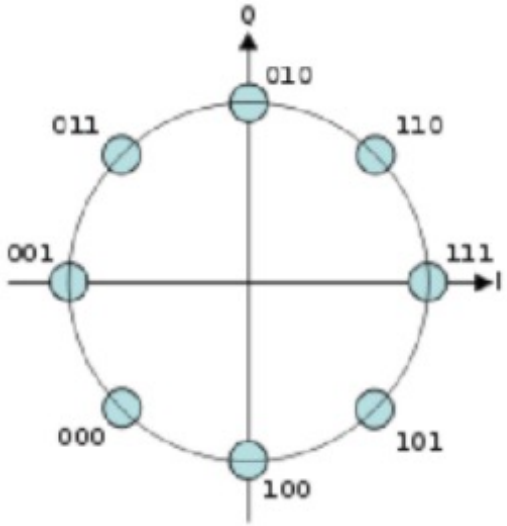
수체계

2진수 수체계

10진수	Binary number	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Grey code

디지털 통신에 사용
에러가 발생해도 1bit만 틀리기 때문에 **오류가 적다**.



수체계는 너무나 중요한 개념!!

2 논리회로

1 수체계

①논리게이트

AND게이트

OR게이트

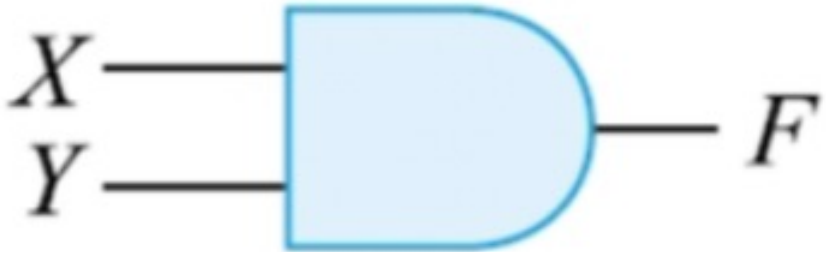
XOR게이트

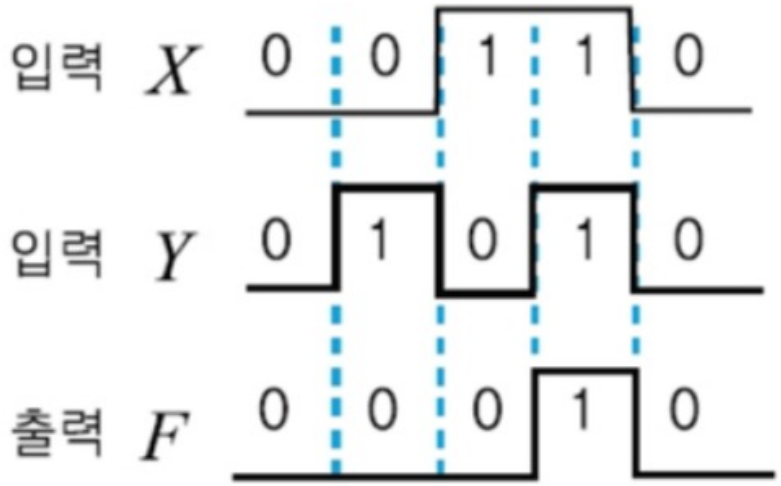
②불대수

③카르노맵

①논리게이트

AND게이트



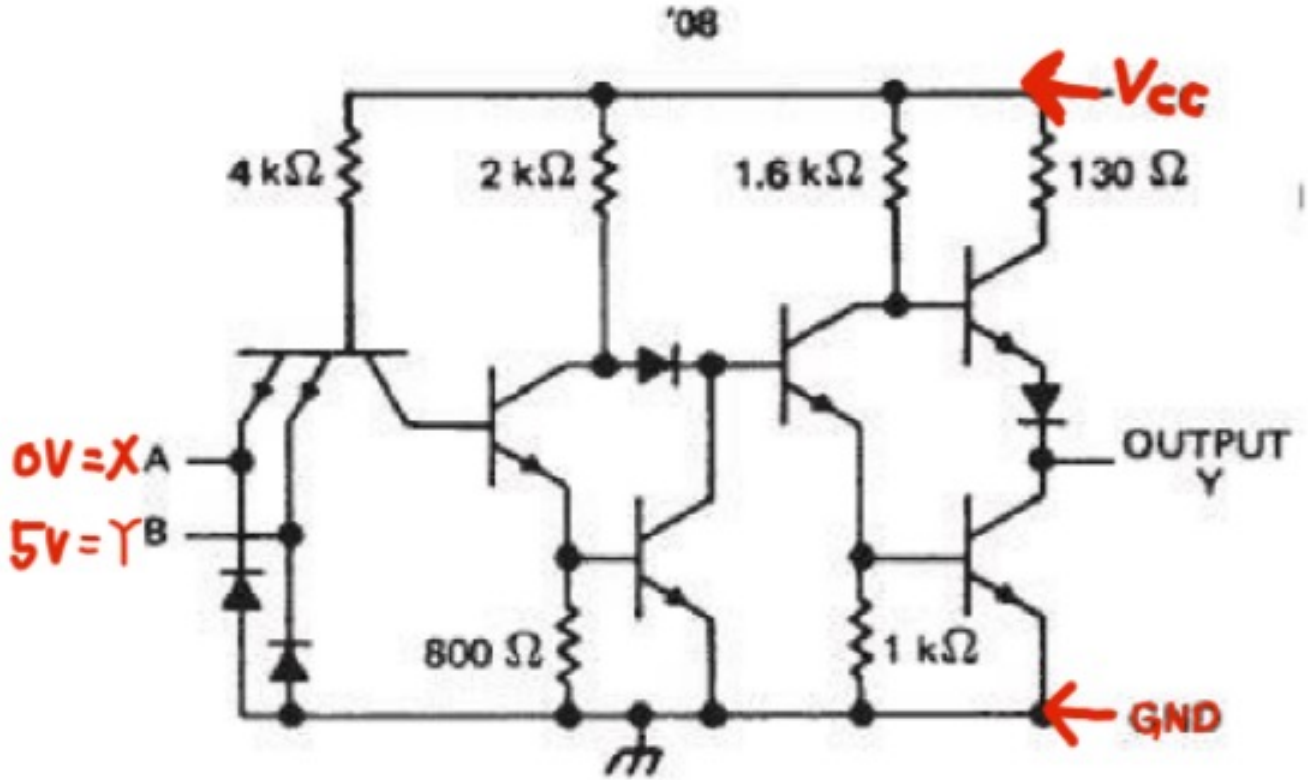


X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1

$F = XY$

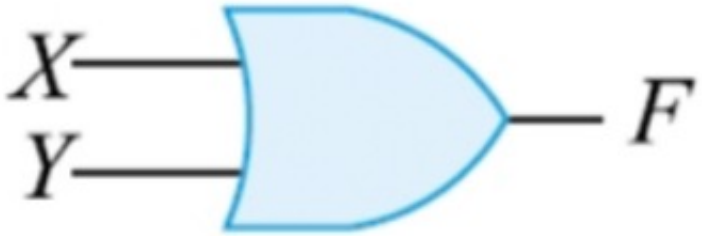
①논리게이트

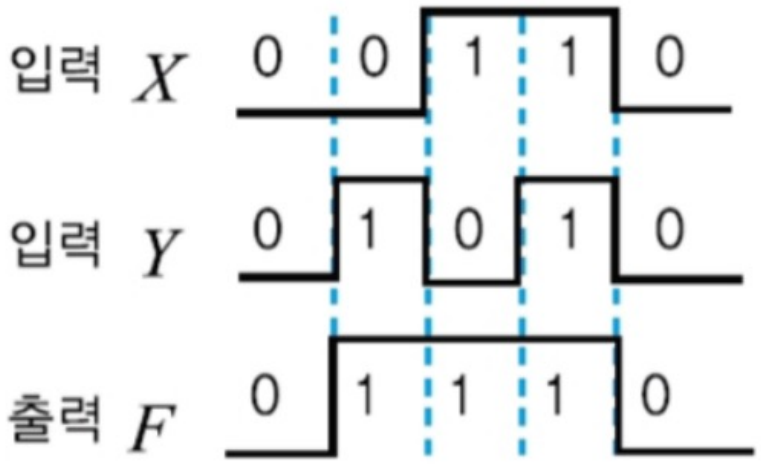
AND게이트



①논리게이트

OR게이트



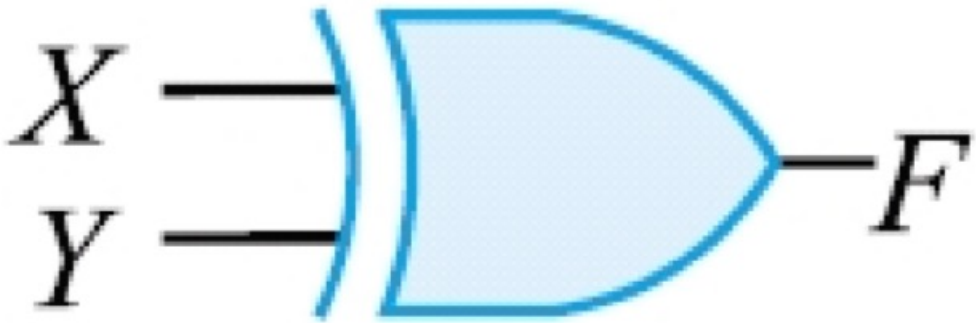


X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

$$F = X + Y$$

①논리게이트

XOR게이트



X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

$$F = X \oplus Y$$

②불대수

AND: F=XY

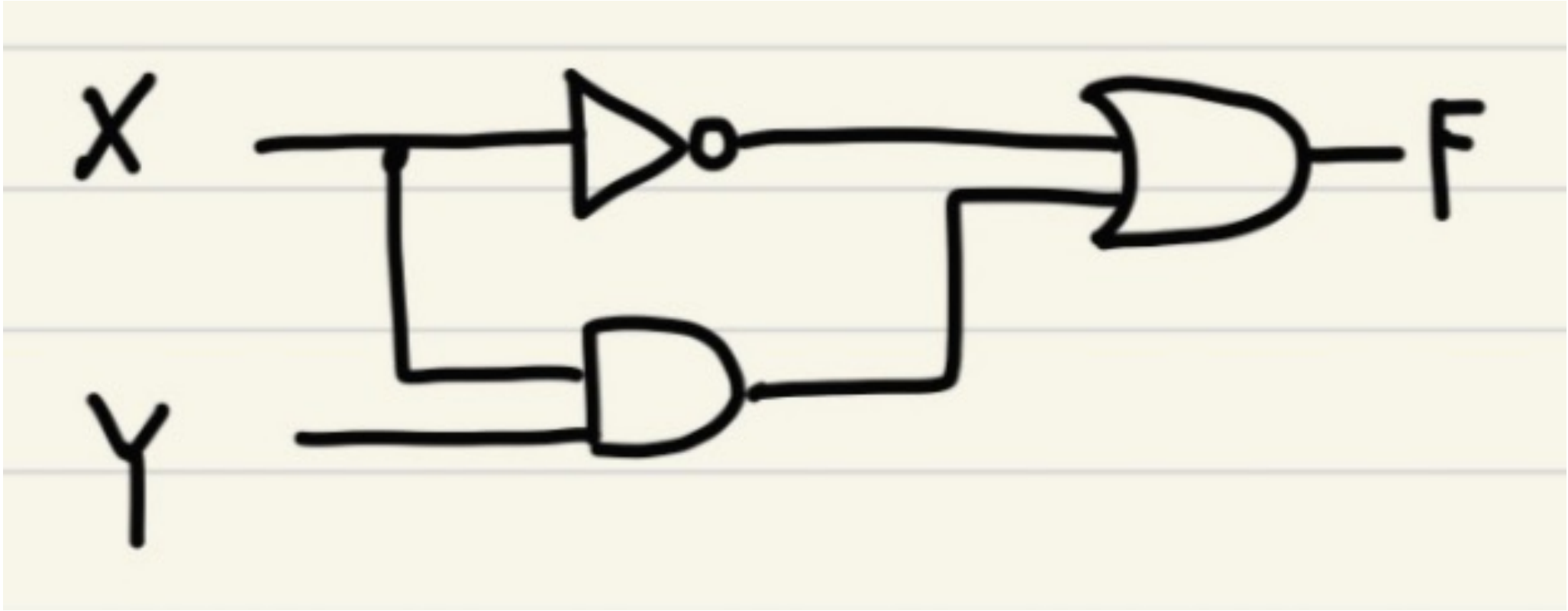
OR: F=X+Y

수학법칙을 만족

[1] $X + 0 =$ [3] $X + 1 =$ [5] $X + X =$ [7] $X + \bar{X} =$ [9] $\bar{\bar{X}} =$	[2] $X \cdot 1 =$ [4] $X \cdot 0 =$ [6] $X \cdot X =$ [8] $X \cdot \bar{X} =$	
[10] $X + Y = Y + X$	[11] $XY = YX$	Commutative law
[12] $X + (Y + Z) = (X + Y) + Z$	[13] $X(YZ) = (XY)Z$	Associative law
[14] $X(Y + Z) = XY + XZ$	**[15]** $X + YZ = (X + Y)(X + Z)$	Distributive law
[16] $\overline{X + Y} = \bar{X}\,\bar{Y}$	[17] $\overline{XY} = \bar{X} + \bar{Y}$	DeMorgan's law

②불대수

$$F = \overline{X} + XY$$

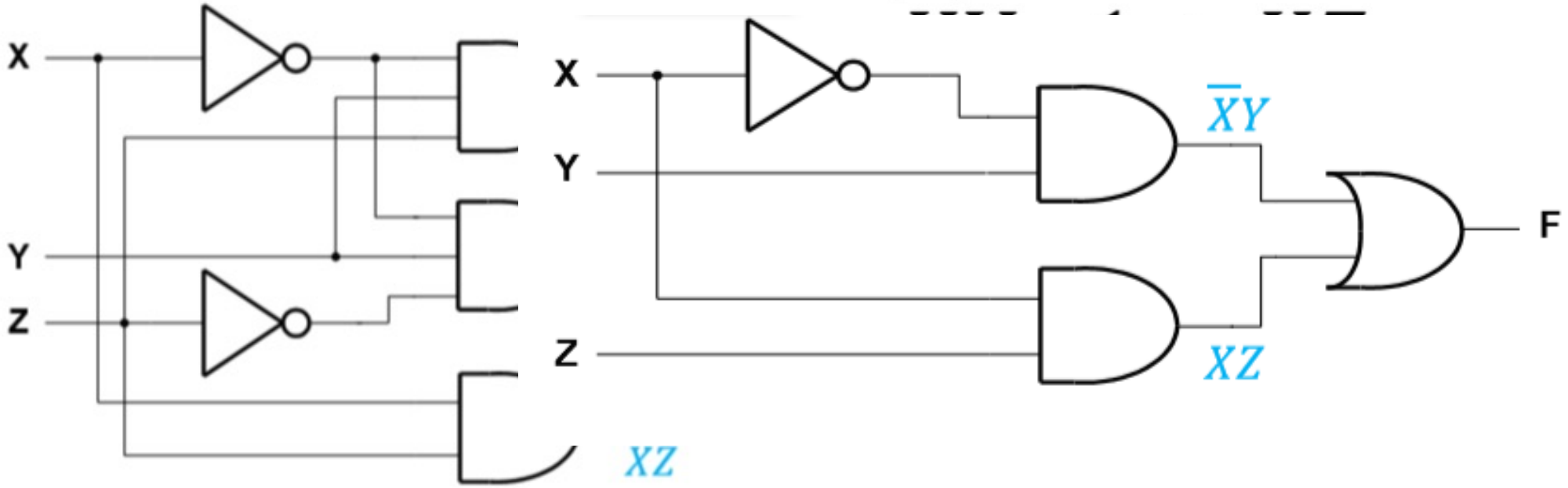


$$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$$

$$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$$

$$= \overline{X}Y(Z + \overline{Z}) + XZ$$

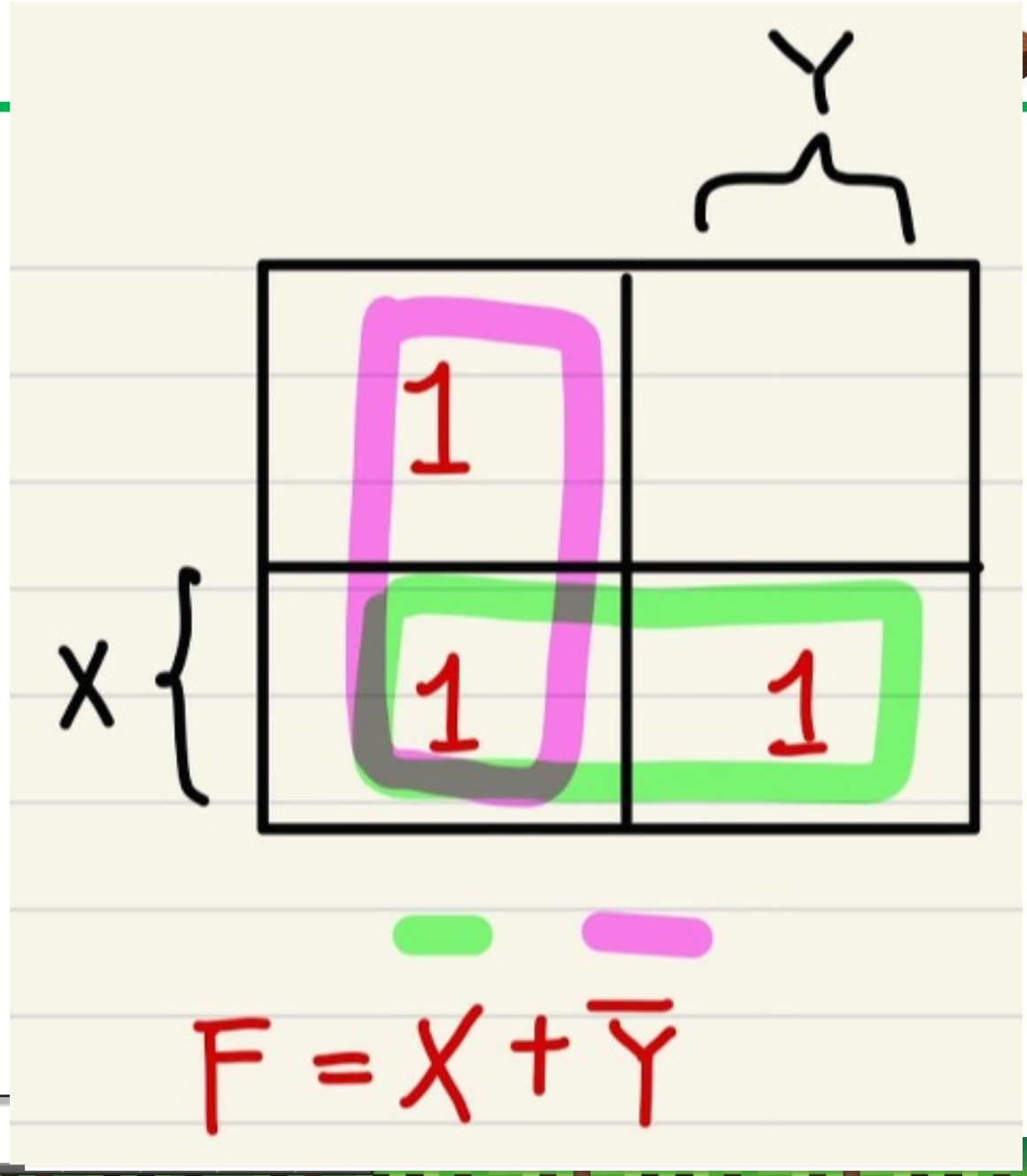
Circuit




M1

③카르노맵

X	Y	F
0	0	1
0	1	0
1	0	1
1	1	1



진리표>불대수 >최적화

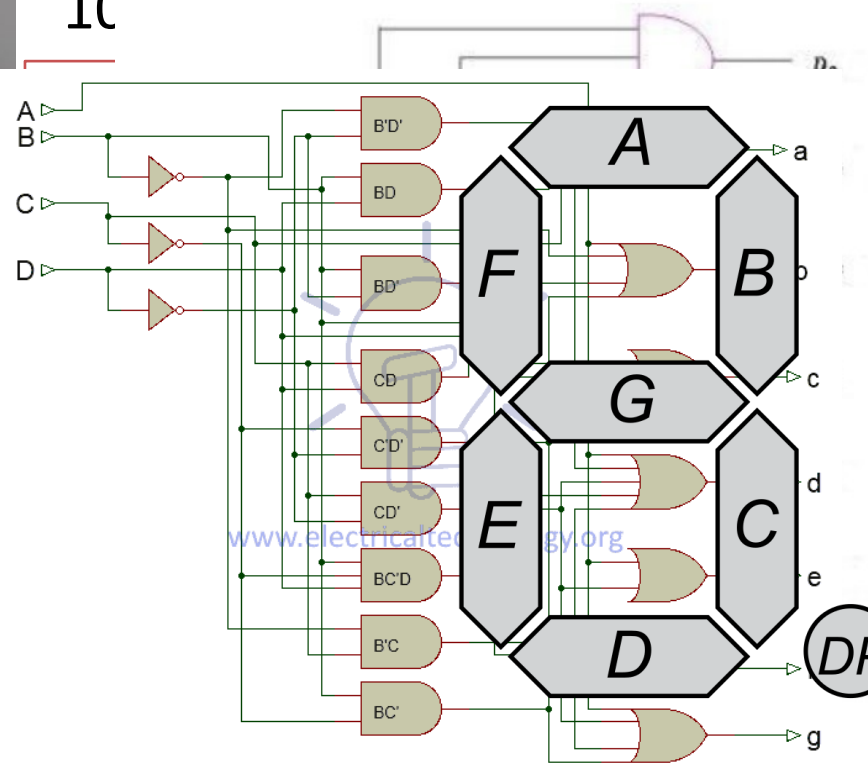
2

전체회로도

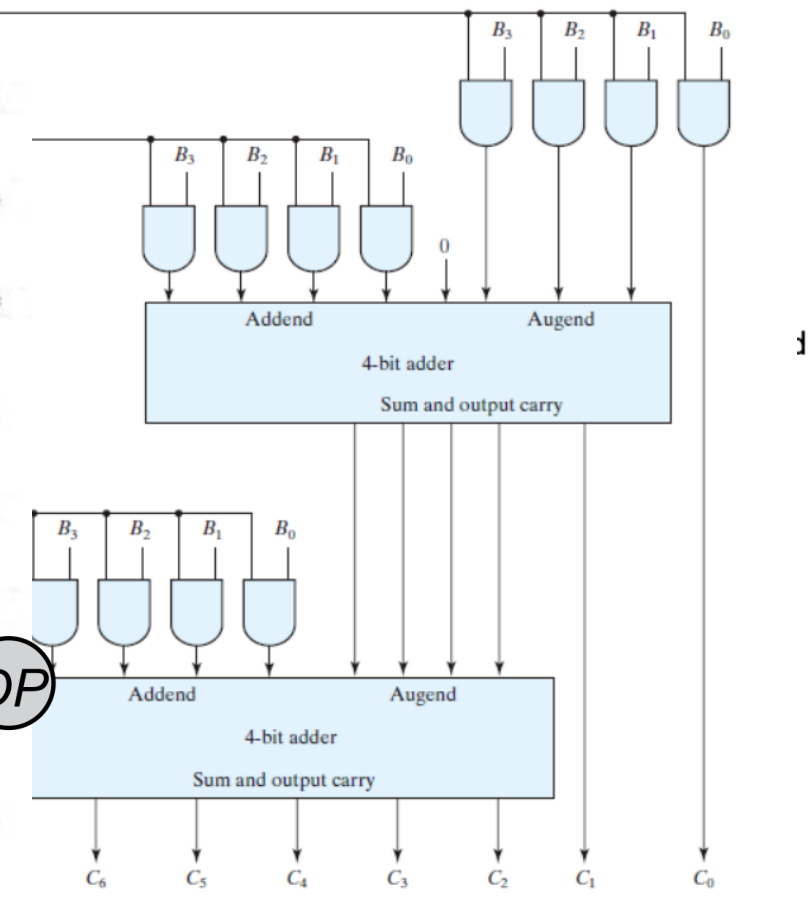
흐름도

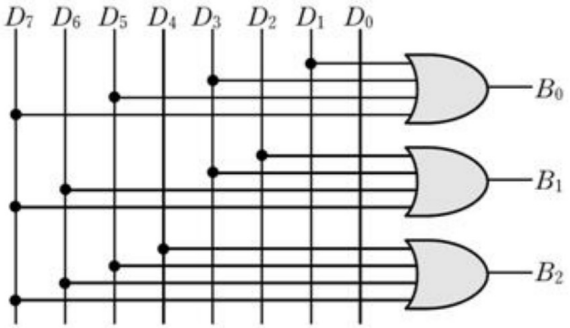
입력 ➡ 연산자 ➡ 출력 ➡ 결과

10



Schematic of BCD to 7-Segment Decoder





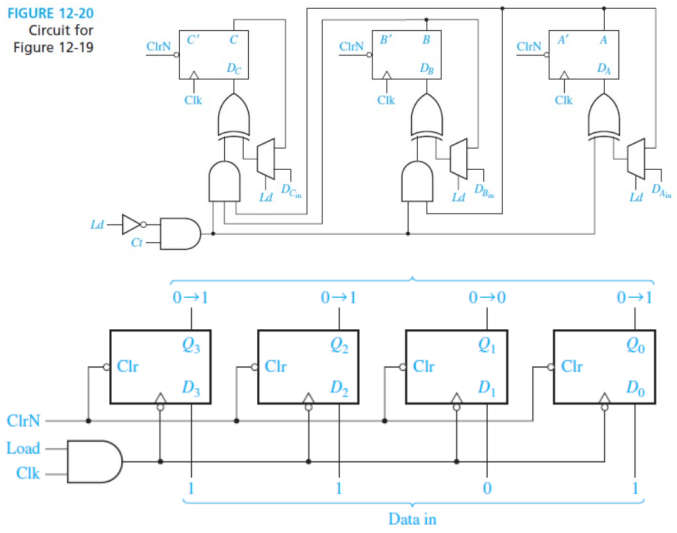
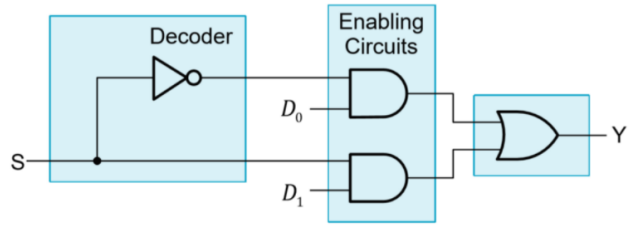
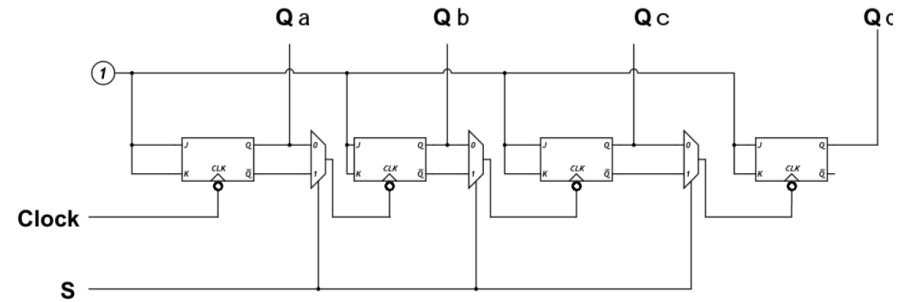


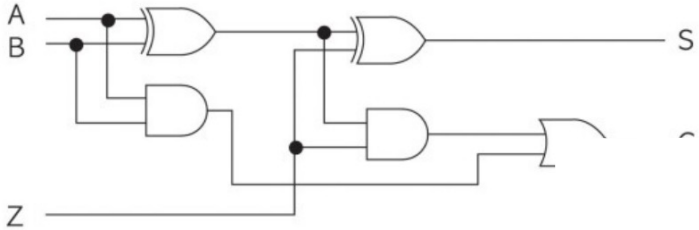
FIGURE 12-20
Circuit for
Figure 12-19

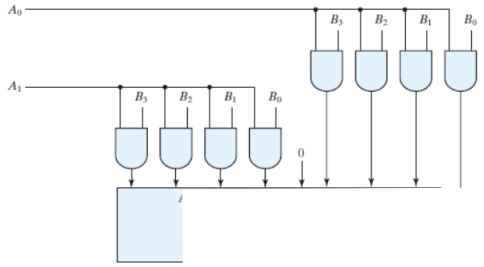
(a) Using gated clock

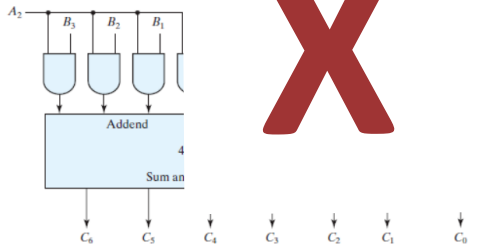


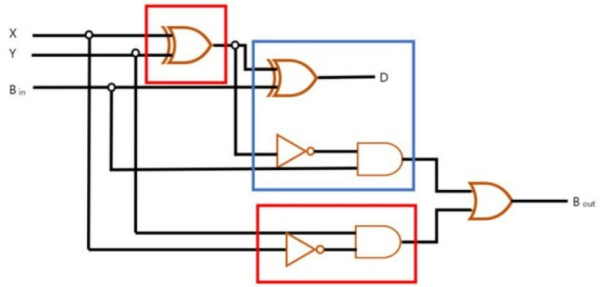
$$Y = \overline{S} \cdot D_0 + S \cdot D_1$$





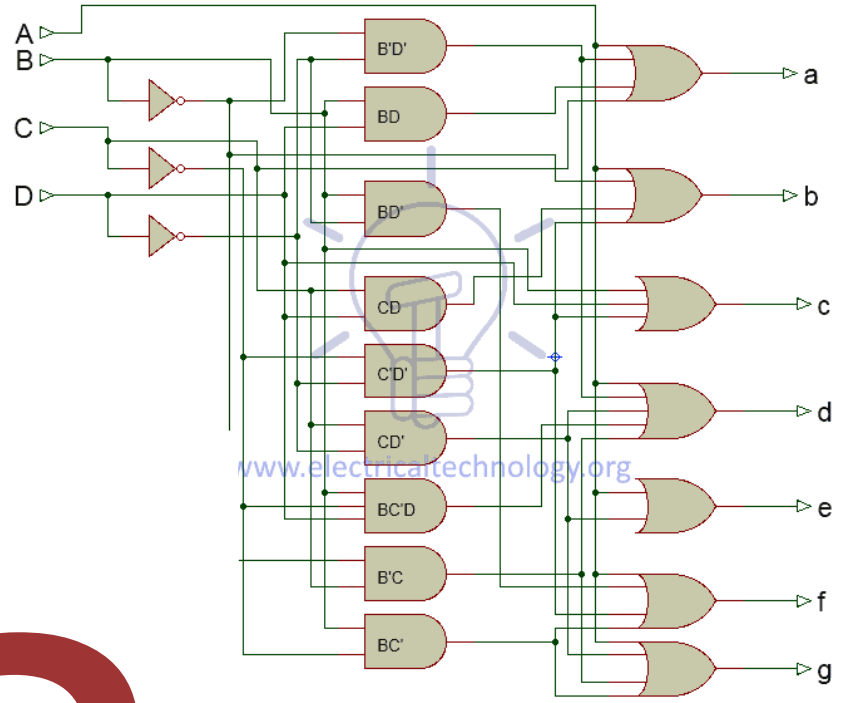


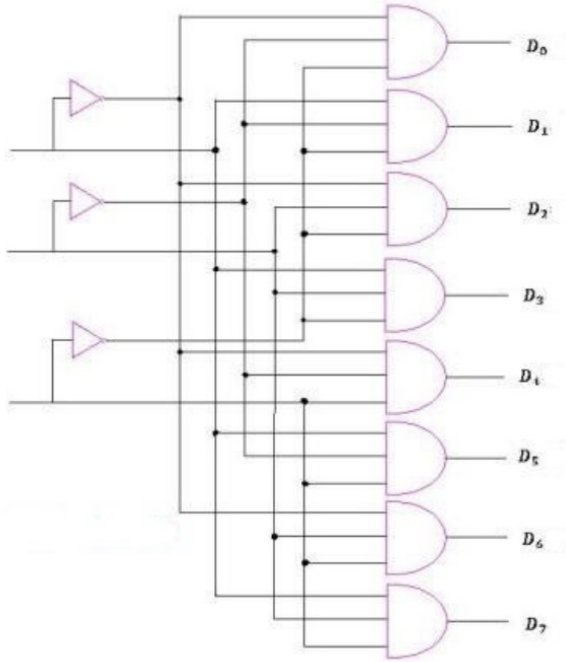




X

8



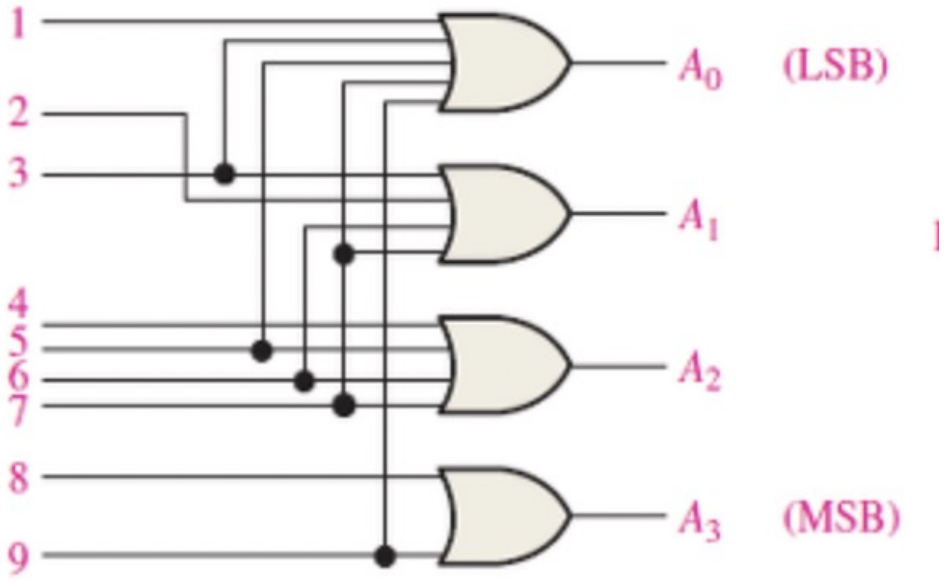


4

입력부터 출력까지

입력(Input)

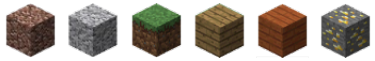
인코더



OR 게이트 4개 이용

10진수를 2진법으로 변환

입력부터 출력까지



플립플롭

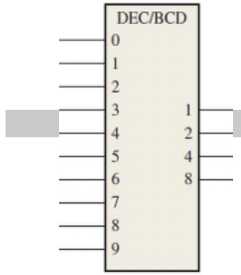
레지스터

카운터

입력(Input)

정수 35 입력

3 입력



0 0 0 0 0 0 1 1

(3)

곱셈기(x10)

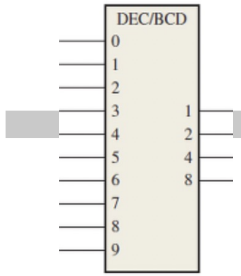
0 0 0 1 1 1 1 0

(30)

input1

가산기

5 입력



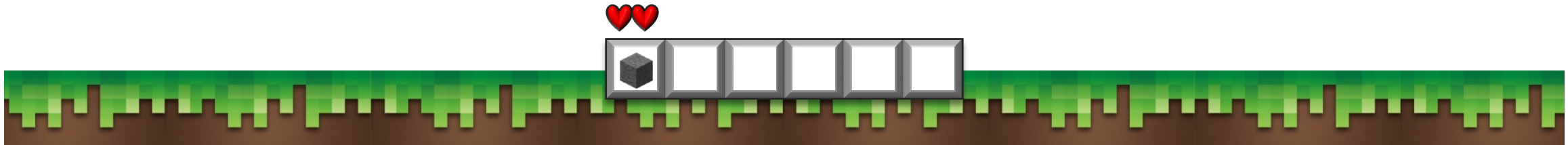
0 0 0 0 0 1 0 1

(5)

input2

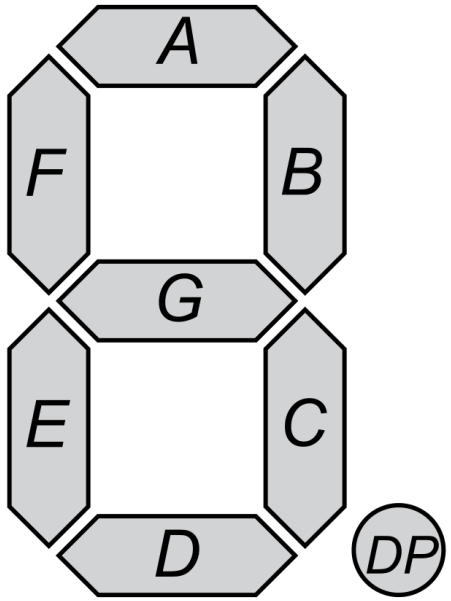
0 0 1 0 0 0 1 1

(35)



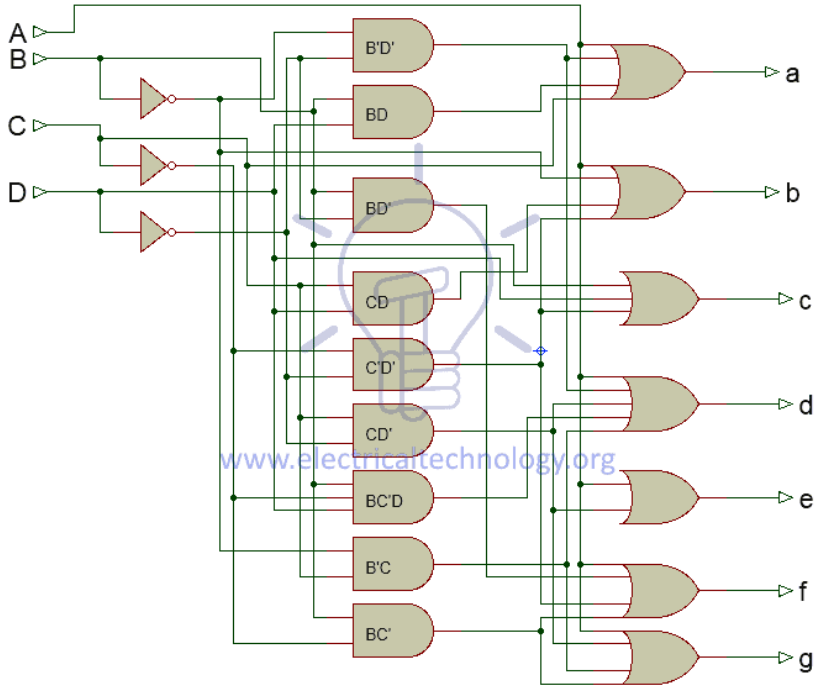
출력(output)

7-Segment Display



2진수를 10진수로 디스플레이에 나타낸다.

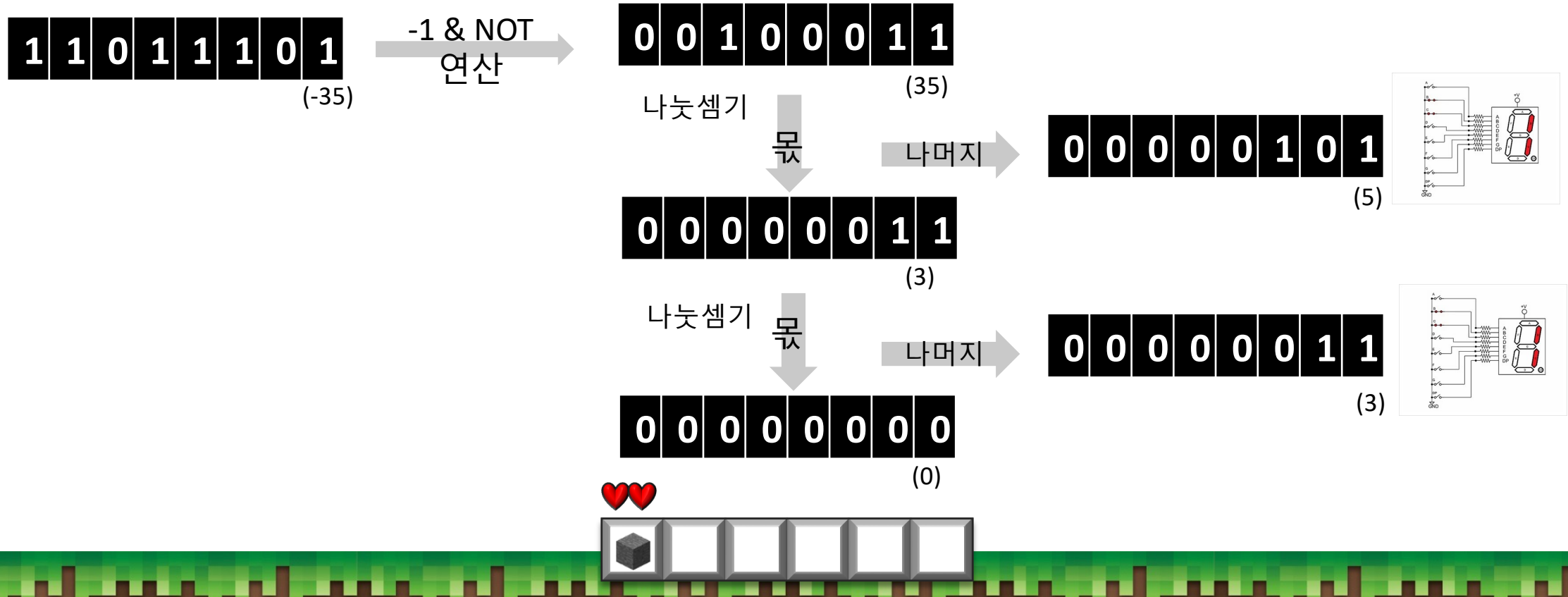
카르노 맵핑 활용하는 전형적인 예

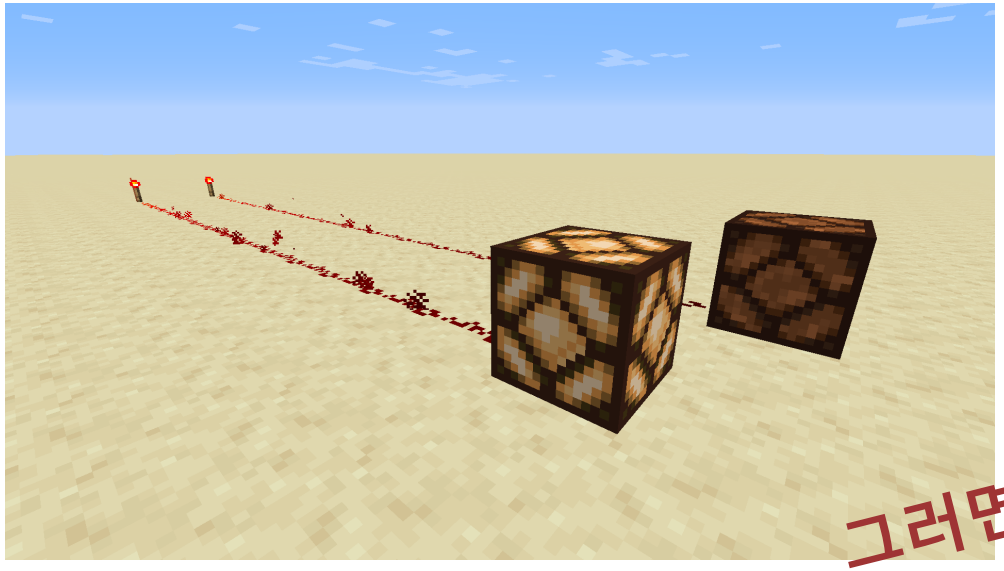


Schematic of BCD to 7-Segment Decoder

출력(output)

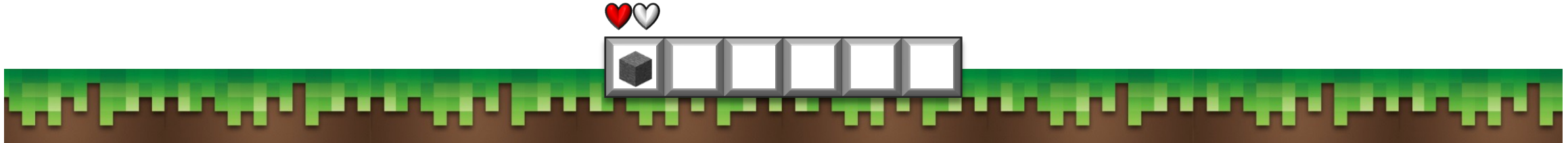
2진법으로 표현된 숫자를 다시
10진수로 자릿수 별로 뽑아내기 위한 **나눗셈 연산**





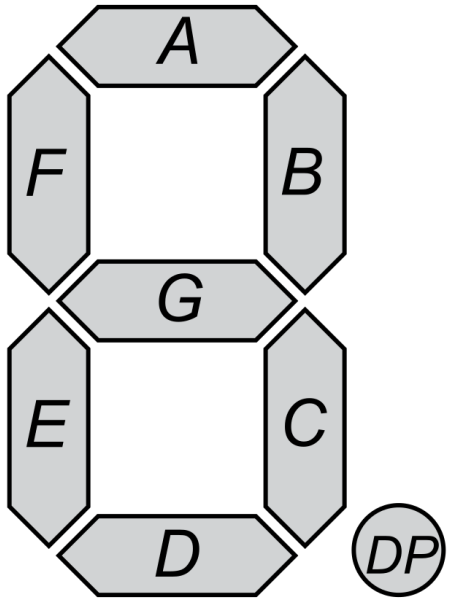
8개의 비트(1 byte) 크기의 정수를 사용하기로 결정!

그러면, 어떻게 숫자를 표현하는 것이 좋을까?



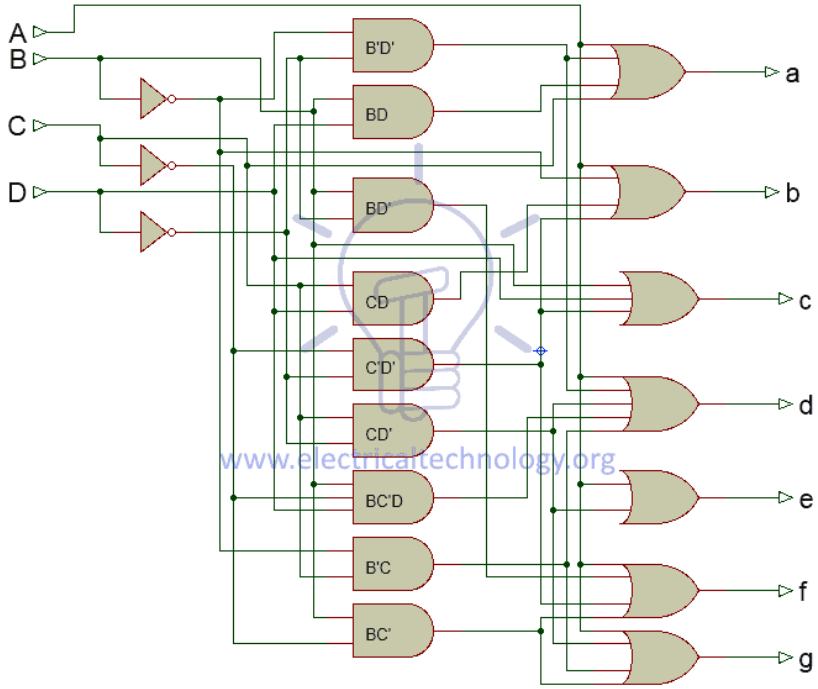
출력(output)

7-Segment Display



2진수를 10진수로 디스플레이에 나타낸다.

카르노 맵핑 활용하는 전형적인 예



Schematic of BCD to 7-Segment Decoder

Truth table

10진수	BCD				Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10~15					X	X	X	X

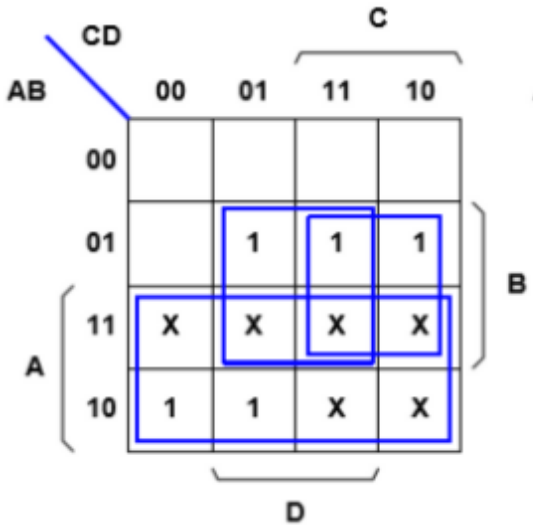
$$W = A + BC + BD = A + B(C + D)$$

$$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D} = \overline{B}(C + D) + B\overline{C}\,\overline{D}$$

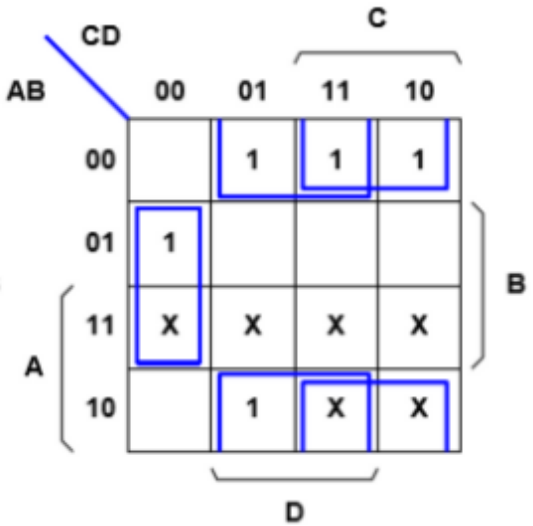
$$Y = CD + \overline{C}\,\overline{D} = \overline{C \oplus D}$$

$$Z = \overline{D}$$

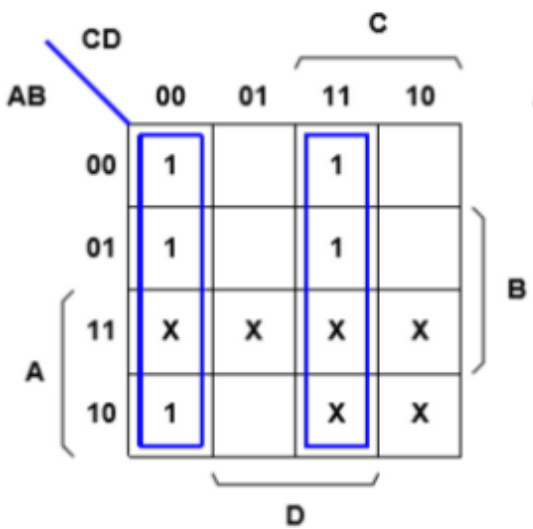
K-map



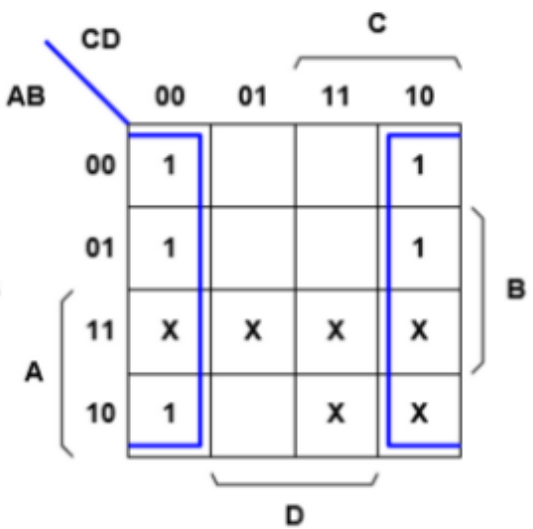
$$W = A + BC + BD$$



$$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D}$$



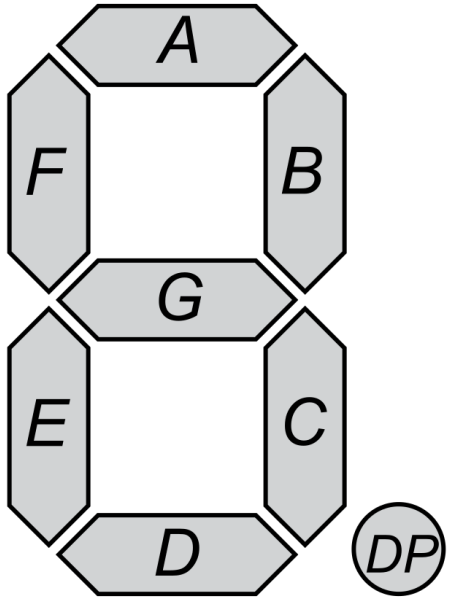
$$Y = CD + \overline{C}\,\overline{D}$$



$$Z = \overline{D}$$

출력(output)

7-Segment Display



UNSIGNED CHAR(부호가 없는 8비트)

숫자가 커지면 회로가 복잡
복잡한 나눗셈기

SIGNED CHAR (부호가 있는 8비트)

감산기(뺄셈)가 필요없지만
보수 변경이 필요
나눗셈기

BCD (Binary-coded decimal)

입출력 회로 간단화

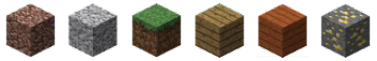
실제로 BCD 회로는 전자 회로와
마이크로프로세서에서 복잡한 나눗셈 회로를 피하기 위해 많이 사용된다.

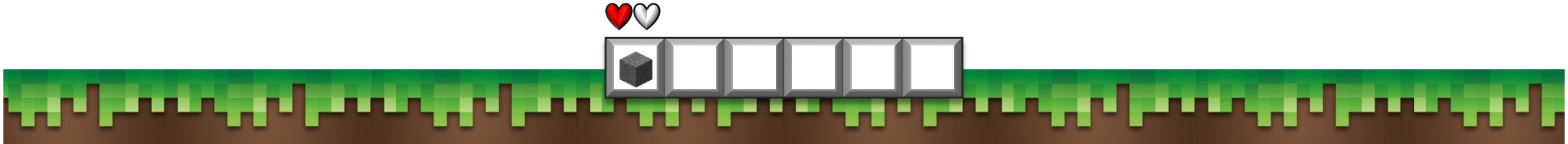
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DEGIGN
가산기
감산기
보수
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


1
목차

목차

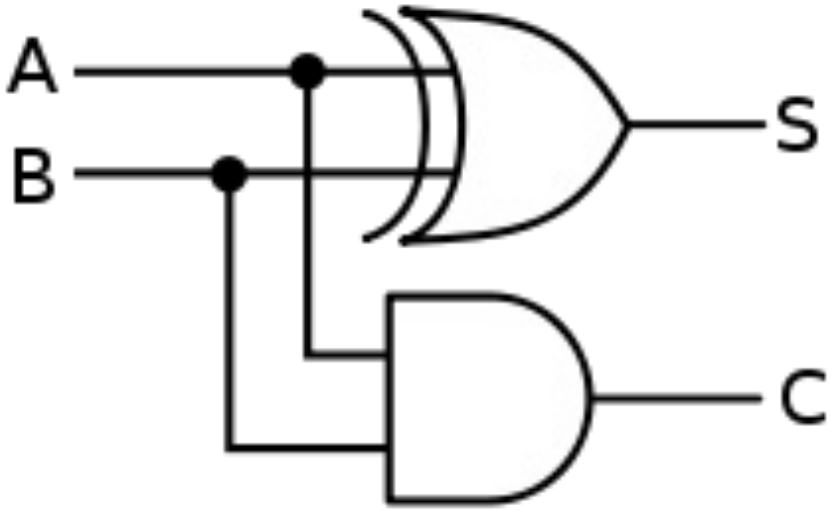





1
가산기

가산기

① 반가산기(Half-Adder)

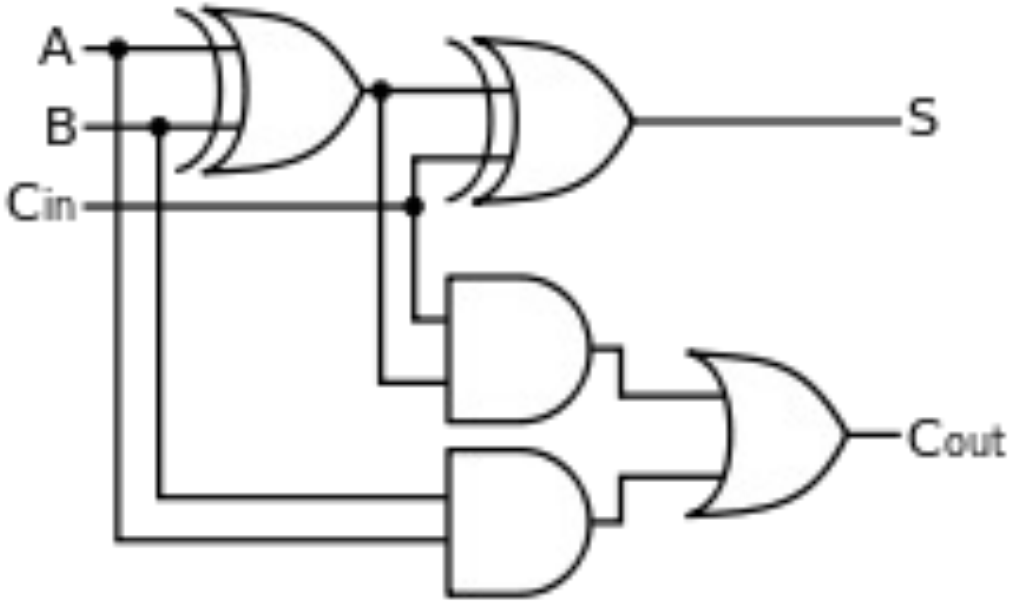


- A : 피연산수
- B : 연산수
- C : 올림수
- S : 합

입력		출력	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

가산기

② 전가산기(Full-Adder)

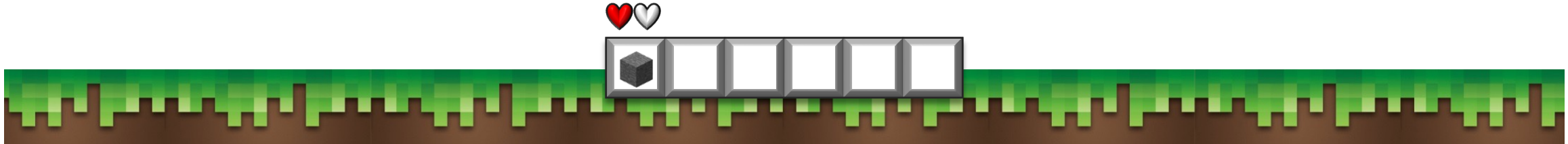


입력			출력	
A	B	C	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- C_{in} : 하위 비트에서 발생한 입력올림수

③ 동영상 시청

https://youtu.be/7a8yTYs84hM

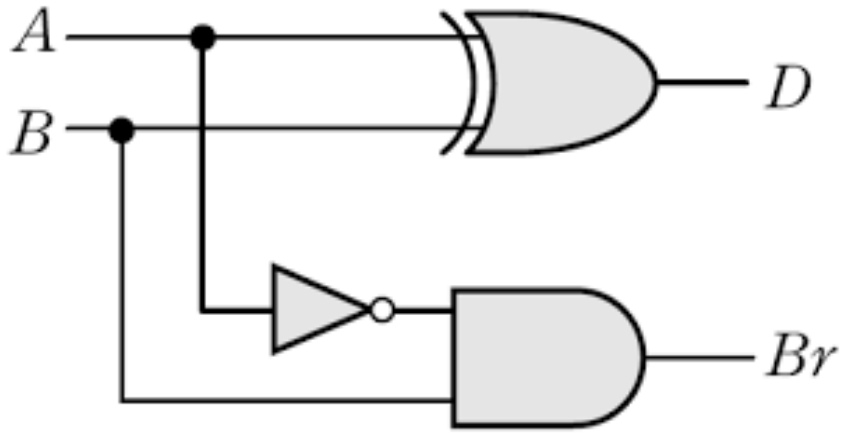


2

감산기

2 감산기

①반감산기(Half-Subtractor)

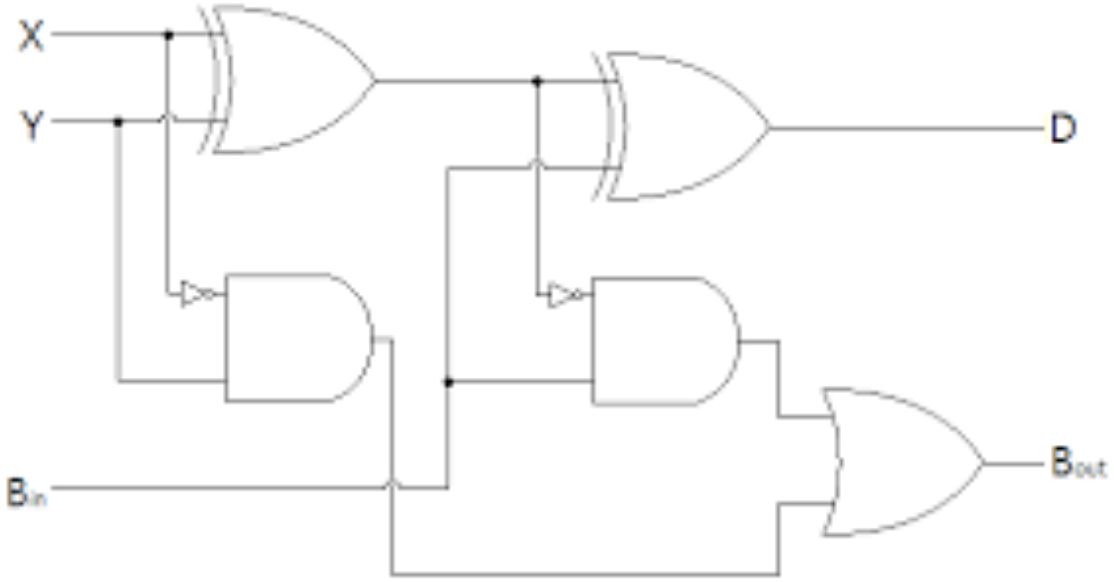


- A : 피연산수
- B : 연산수
- Br : 빌림수
- D : 차

입력		출력	
A	B	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

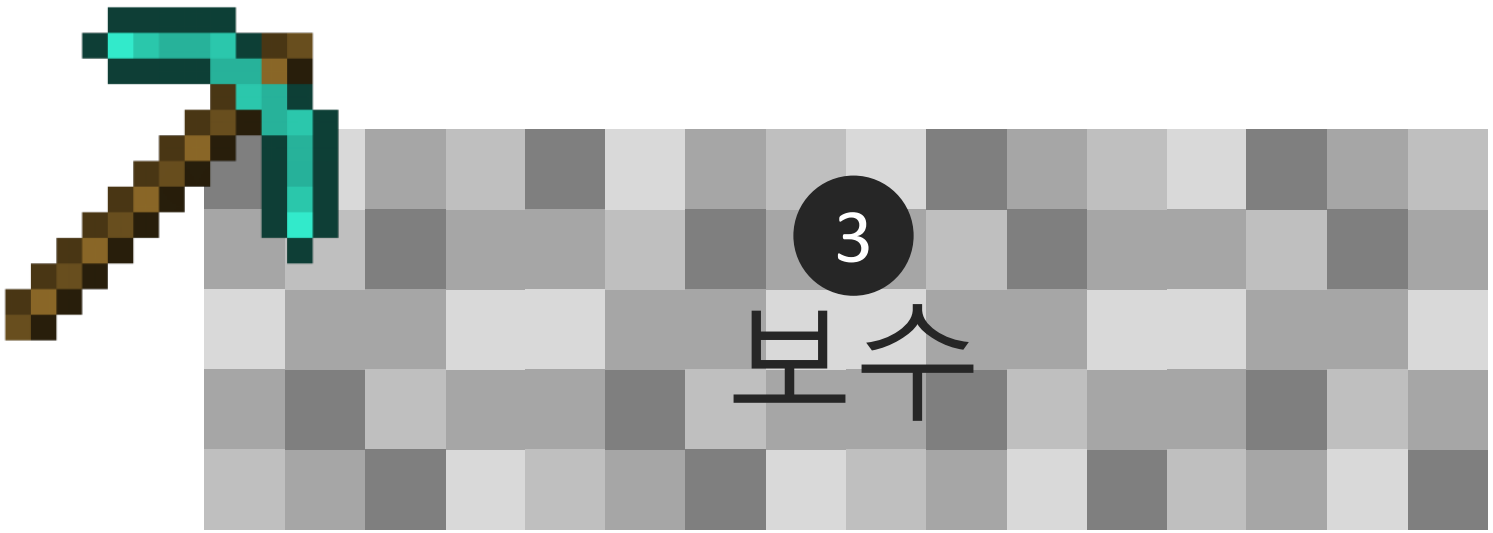
감산기

②전감산기(Full-Subtractor)



- C : 빌려준 빌림수

입력			출력	
A	B	C	Br	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1


3
보수

3 보수

①보수 (complement)

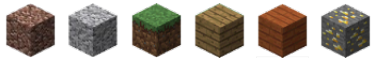
- 각 자리의 숫자가 일정한 값이 되도록 보충해주는 수

Ex) 10진법에서 12.87의 9에 대한 보수는 87.12 → 합이 99.99
10진법에서 12.87의 10에 대한 보수는 87.13 → 합이 100.00

- 2진법에서도 보수를 적용할 수 있다.

Ex) 2진법에서 $(1011)_2$의 1에 대한 보수는 $(0010)_2$ → 합이 $(1111)_2$
2진법에서 $(1011)_2$의 2에 대한 보수는 $(0101)_2$ → 합이 $(10000)_2$

보수



②부호 절댓값 (sign-magnitude)

0000 : +0	1000 : -0
0001 : +1	1001 : -1
0010 : +2	1010 : -2
0011 : +3	1011 : -3
0100 : +4	1100 : -4
0101 : +5	1101 : -5
0110 : +6	1110 : -6
0111 : +7	1111 : -7

- 부호가 같은 두 수의 덧셈이 제대로 작용

Ex) 3 + 2 = $(0011)_2$ + $(0010)_2$= $(0101)_2$= 5

(-3) + (-2) = $(1011)_2$+ $(1010)_2$= $(1101)_2$= -5

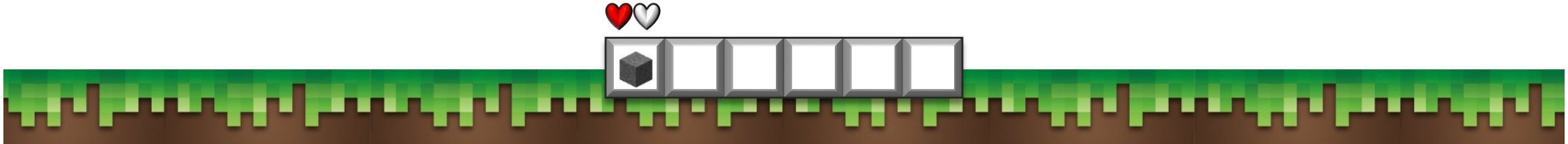
- 음수와 양수의 덧셈이 제대로 작용되지 않음

Ex) 3 + (-2) = $(0010)_2$ + $(1010)_2$ = $(1100)_2$= -4

→ **FALSE**

3 - 2 = $(0010)_2$ - $(1010)_2$ = $(0001)_2$ = 1

→ **TRUE**



보수

③ 1의 보수 (1's complement)

0000 : +0	1111 : -0
0001 : +1	1110 : -1
0010 : +2	1101 : -2
0011 : +3	1100 : -3
0100 : +4	1011 : -4
0101 : +5	1010 : -5
0110 : +6	1001 : -6
0111 : +7	1000 : -7

- 부호와 절댓값이 다른 두 수에 대한 덧셈도 가능

Ex) 3 + (-4) = $(0011)_2$ + $(1011)_2$ = $(1110)_2$= -1

- 부호가 다르고 절댓값이 같은 두 수에 대한 덧셈에서 오류 발생

Ex) 3 + (-3) = $(0011)_2$+ $(1100)_2$

=$(1111)_2$= -0 > $(0000)_2$= +0

→ **FALSE**

- 캐리가 발생하면 최하위 비트에 1을 더해야 한다.

Ex) 3 + (-2) = $(0011)_2$ + $(1101)_2$

= $(0000)_2$ + $(0001)_2$ = $(0001)_2$ = 1

보수

④ 2의 보수 (2's complement)

0000 : +0	0000 : - 0
0001 : +1	1111 : -1
0010 : +2	1110 : -2
0011 : +3	1101 : -3
0100 : +4	1100 : -4
0101 : +5	1011 : -5
0110 : +6	1010 : -6
0111 : +7	1001 : -7

- -0 = +0 이기 때문에 결과가 0인 계산에 대한 오류가 사라짐

Ex) 3 + (-3) = $(0011)_2$+ $(1101)_2$

$=(0000)_2$= 0

→ **TRUE**

- 캐리가 발생해도 최하위 비트에 1을 더하지 않아도 된다.

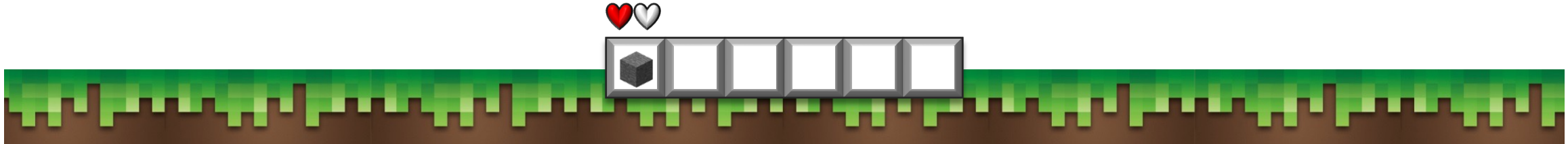
Ex) 7+ (-3) = $(0111)_2$ + $(1101)_2$= $(0100)_2 = 4$

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DEGIGN
곱셈기
나눗셈기
활동 사진
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


0
목차

목차



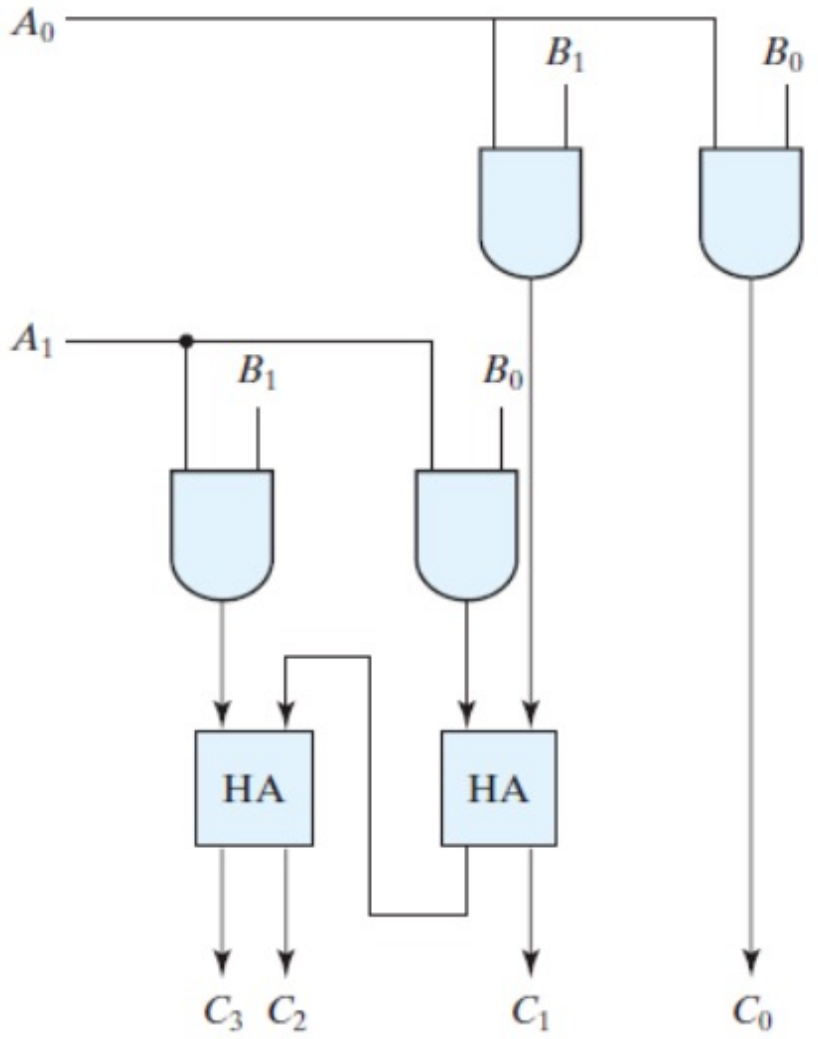
1

곱셈기

곱셈기(multiplier)

① 2진 곱셈기(multiplier)

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



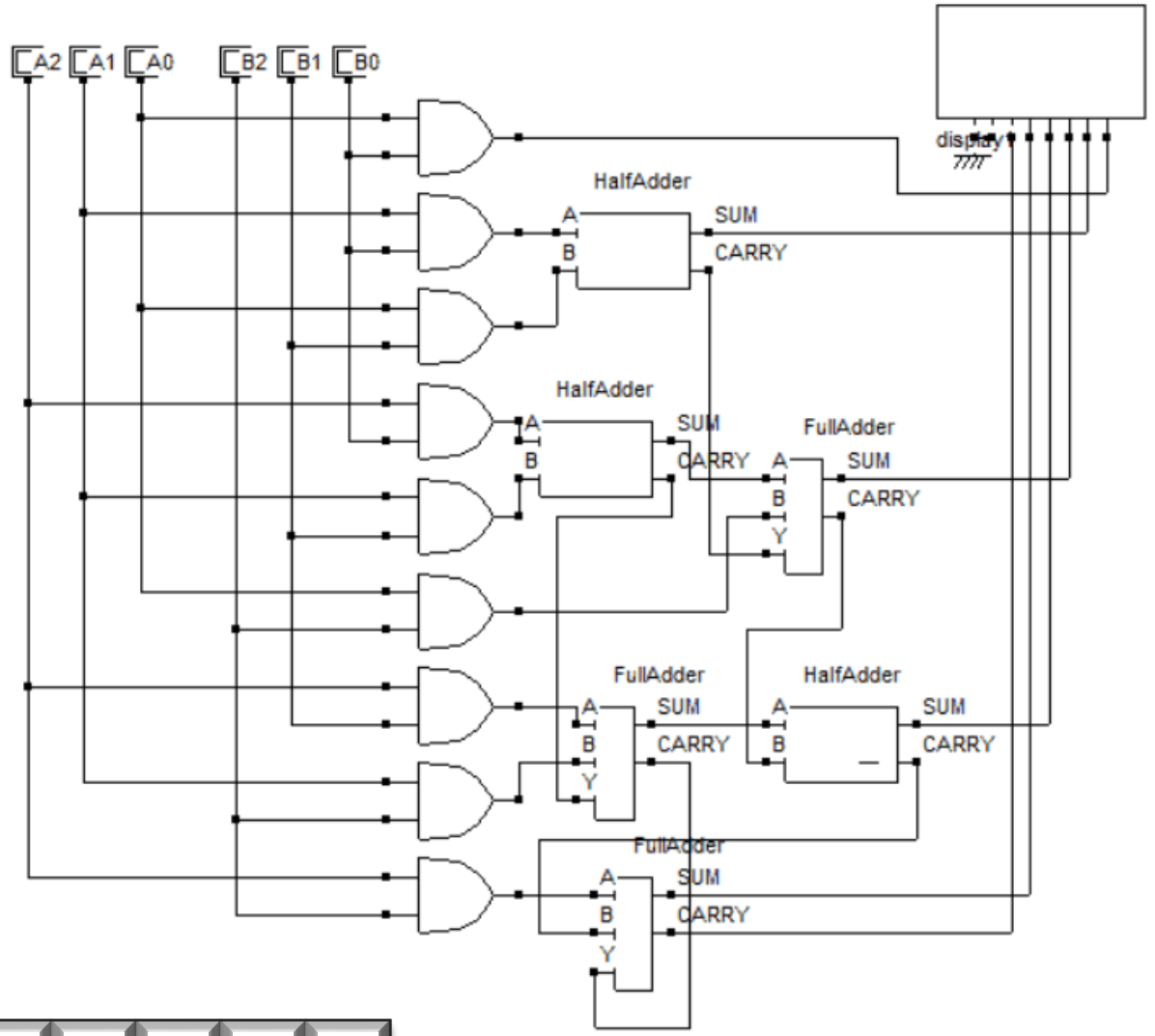
Ex. 10 X 11 = ?

곱셈기(multiplier)

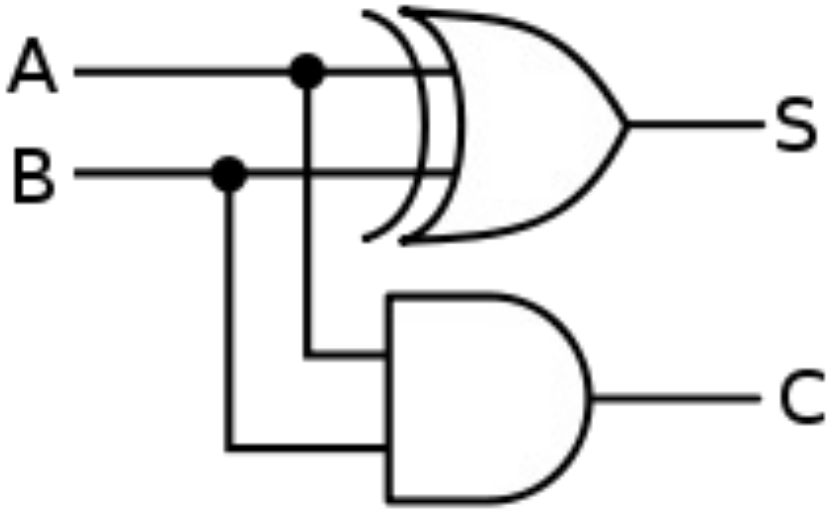
② 응용1 - 3비트 곱셈기(3-bit multiplier)

		A2	A1	A0
		B2	B1	B0
		A2B0	A1B0	A0B0
	A2B1	A1B1	A0B1	X
A2B2	A1B2	A0B2	X	X
A2B2+C+C	A2B1+A1B2 +A2B2+C+C	A1B1+C+ A0B2+A2B0	A0B1+A1B0	A0B0

=> 비트가 늘어날수록
전가산기도 필요한 이유?

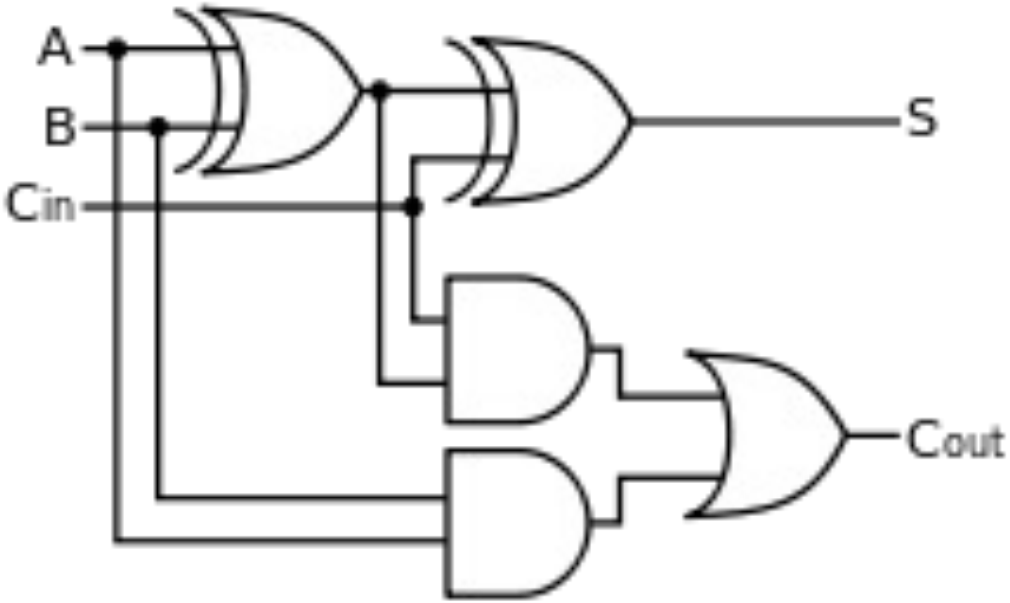


곱셈기(multiplier)



반가산기(Half Adder)

입력		출력	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



전가산기(Full Adder)

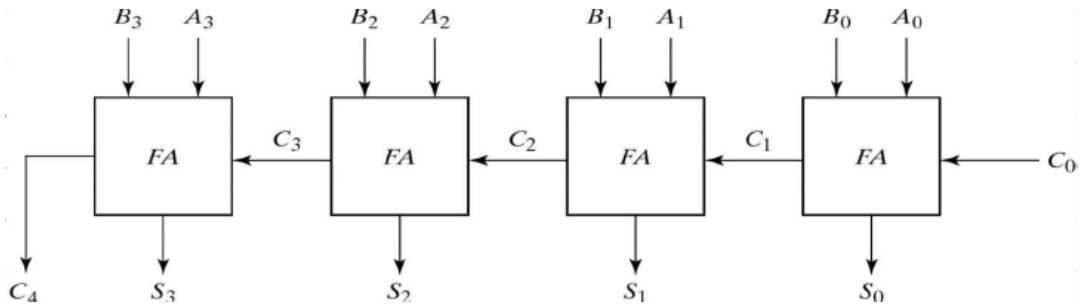
입력			출력	
A	B	C	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

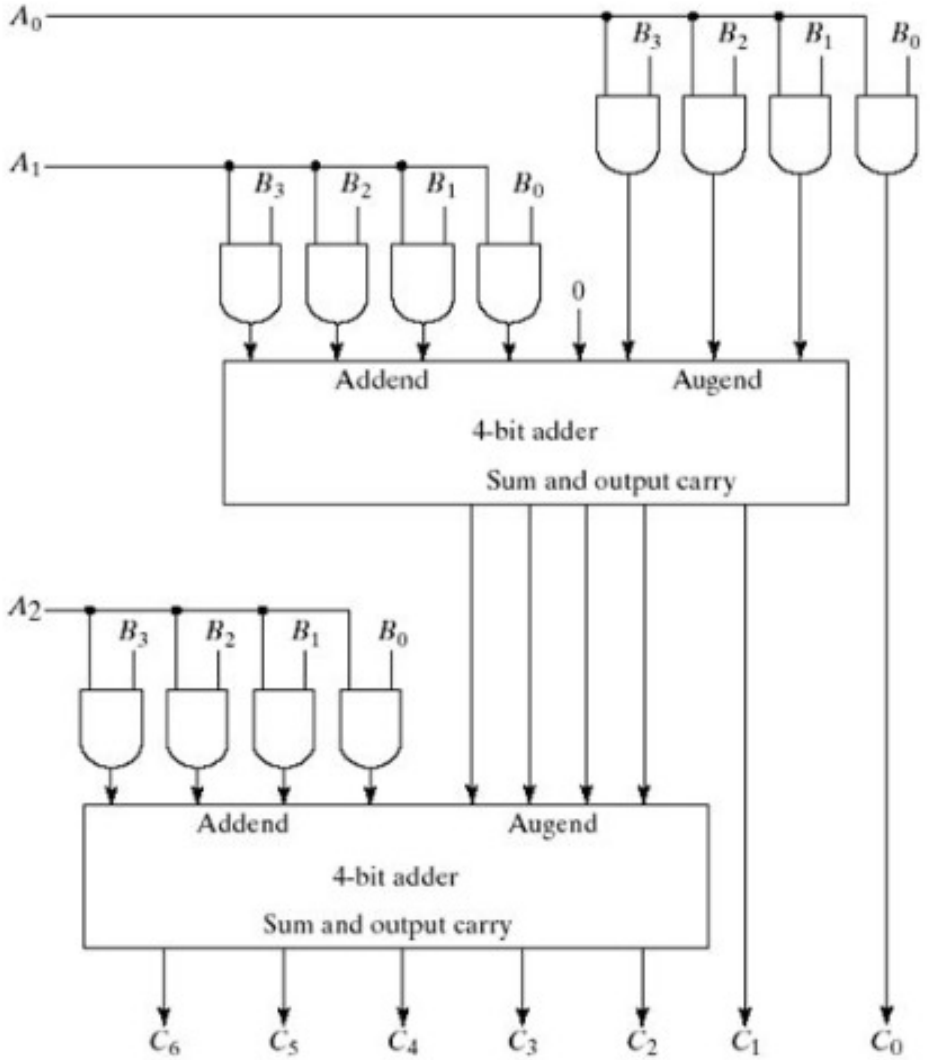
곱셈기(multiplier)

③ 응용2 - K비트 X J비트 곱셈기

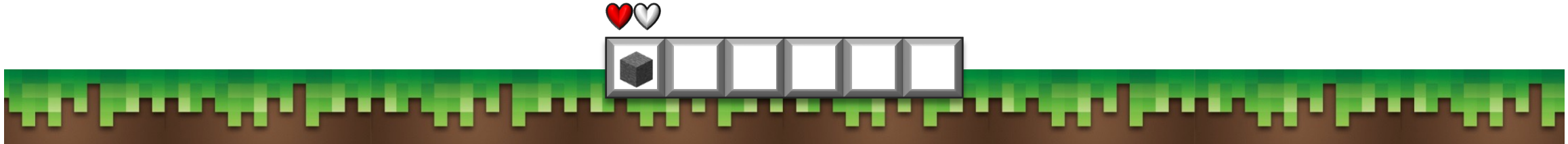
(K비트) X (J비트)의 곱셈?

⇒ (K X J)개의 AND gate
⇒ (J-1)개의 K비트의 adder





BCD multiplier로의 응용?

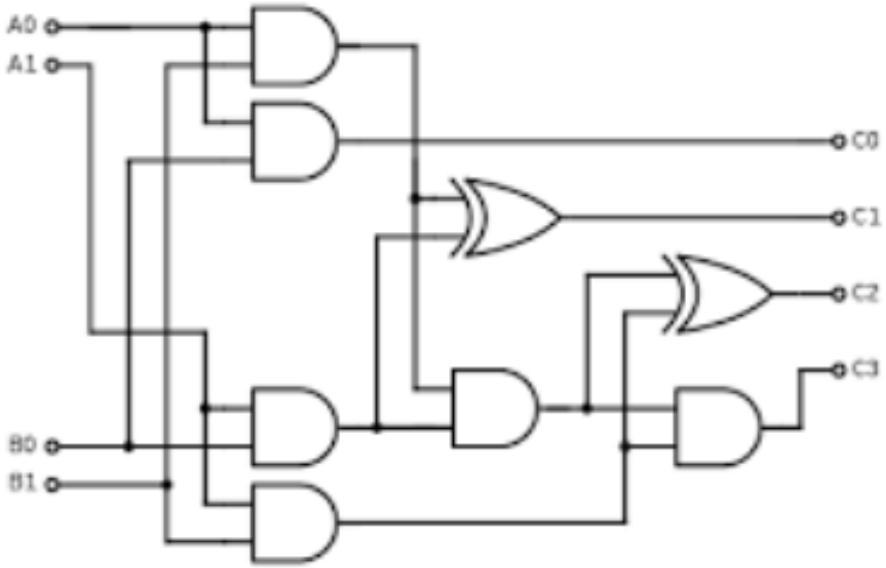


2

나눗셈기

나눗셈기(divider)

①감산기을 이용한 division



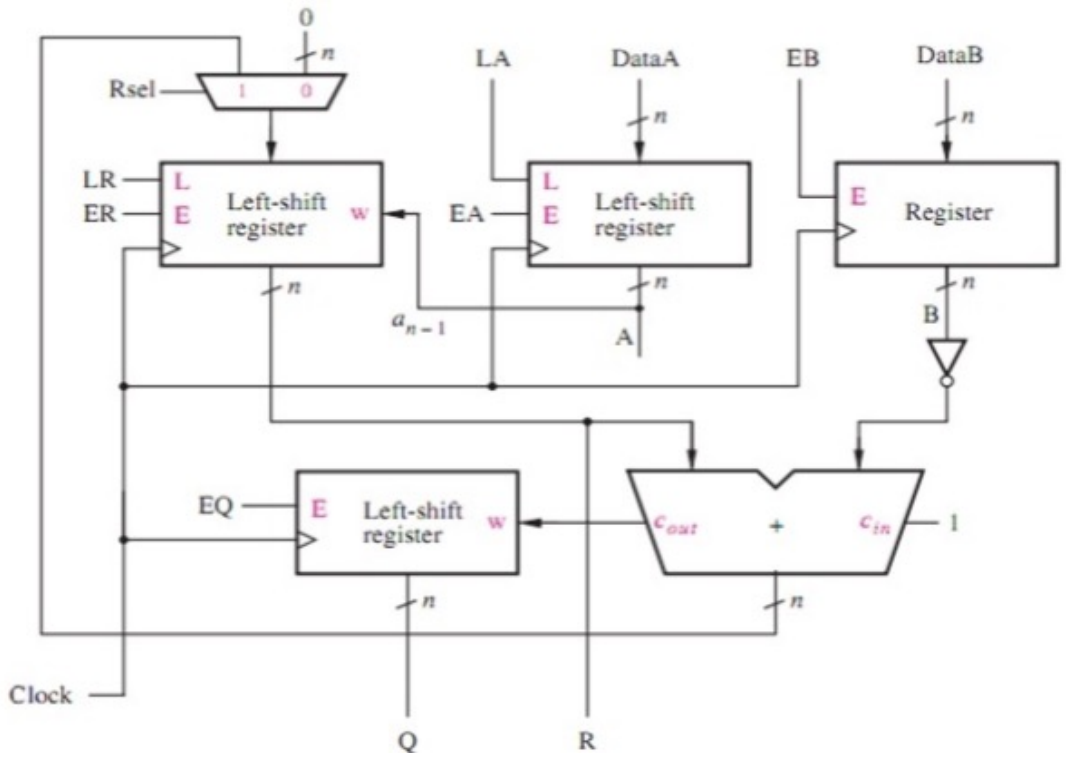
Two Bit Binary Division

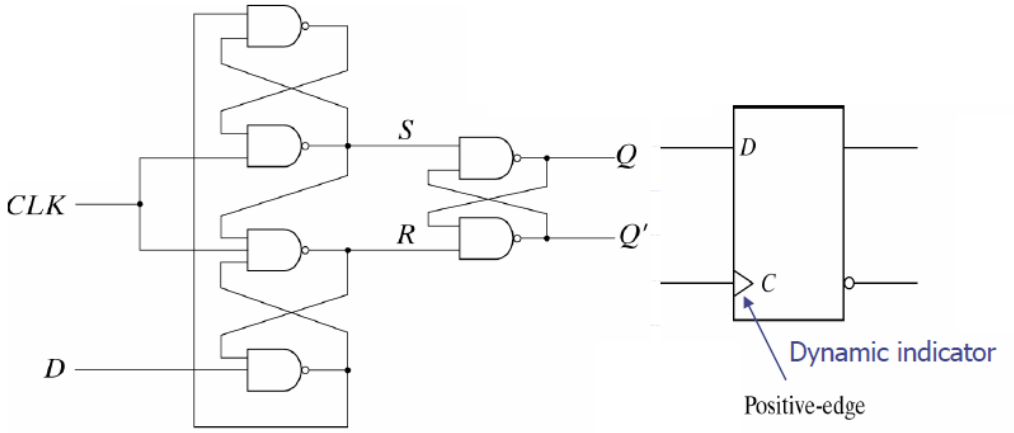
	000111
110	101010
	0
	10
	-0
	101
	-0
	1010
	-110
	1001
	-110
	0110
	0110
	0

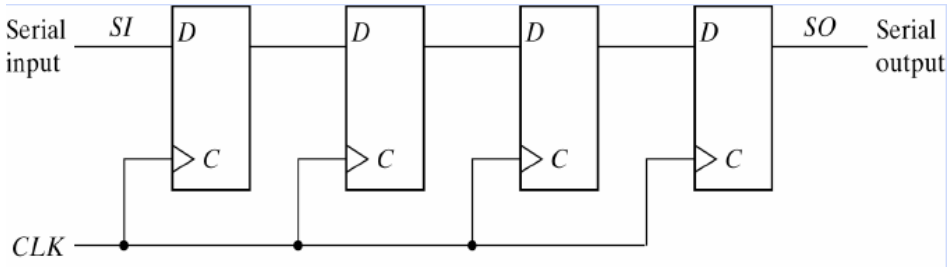
Binary Division

나눗셈기(divider)

② 레지스터를 이용한 division

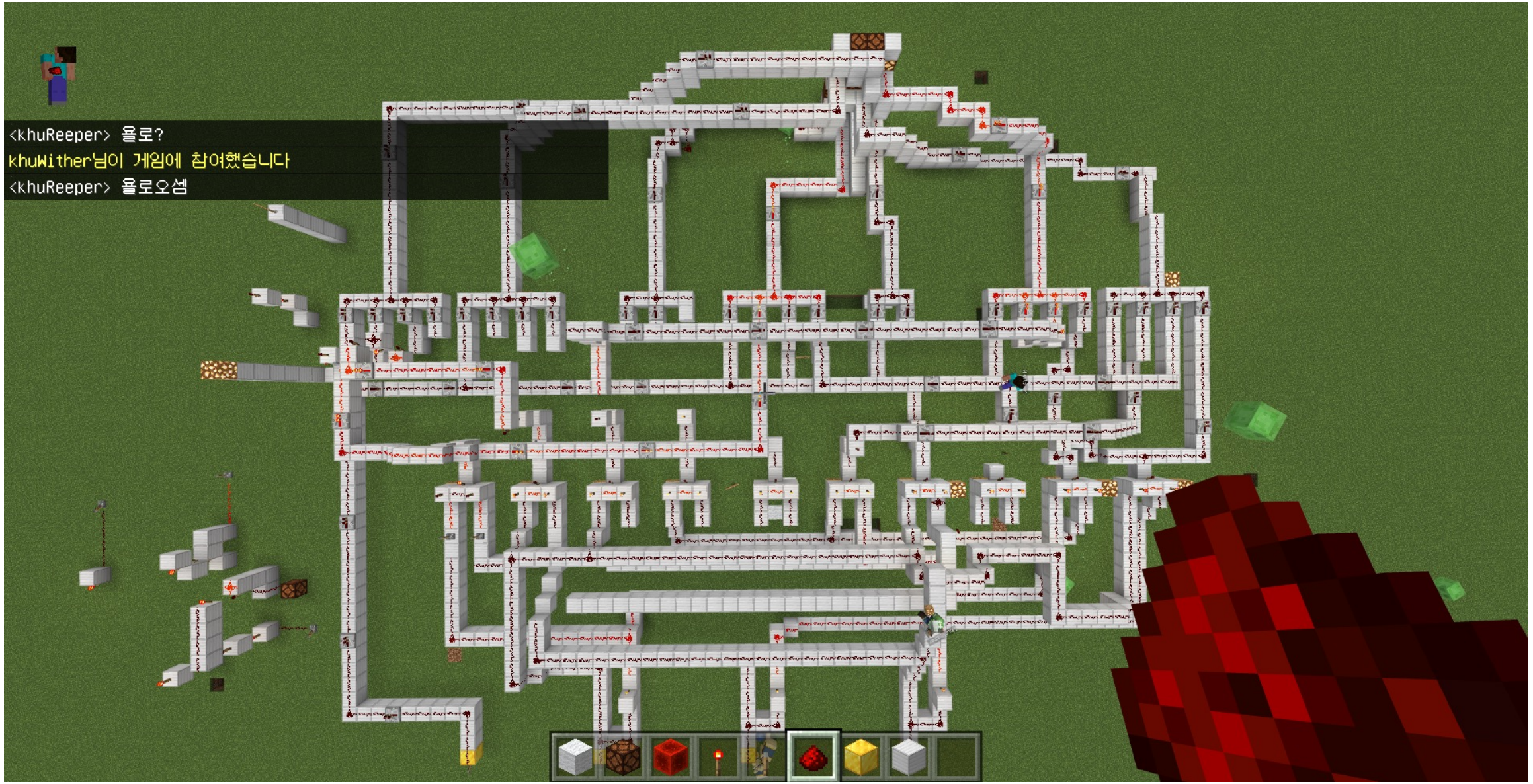






3

활동사진


<khuReeper> 욜로?
khuWither님이 게임에 참여했습니다
<khuReeper> 욜로오셈

Coming up..

1. 계산기 회로도 컴퓨터로 제작
2. 마인크래프트로 실제 제작 설명

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !

MINECRAFT

CAPSTONE DEGIGN

전체적인 회로 설계 1

Preview

이번주 건축 실황

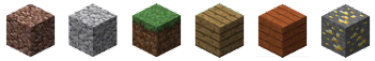
종료

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

1

전체적인 회로 설계 1

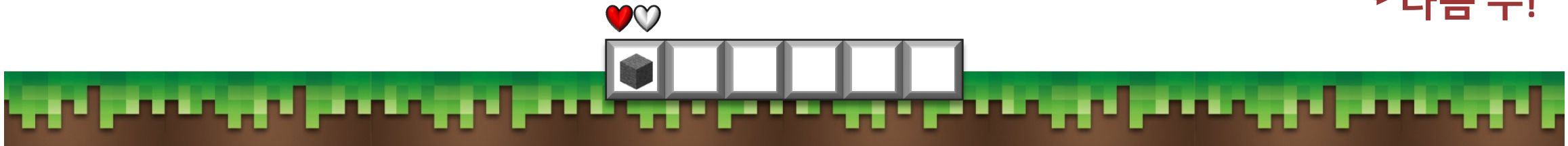
전체적인 회로 설계 1



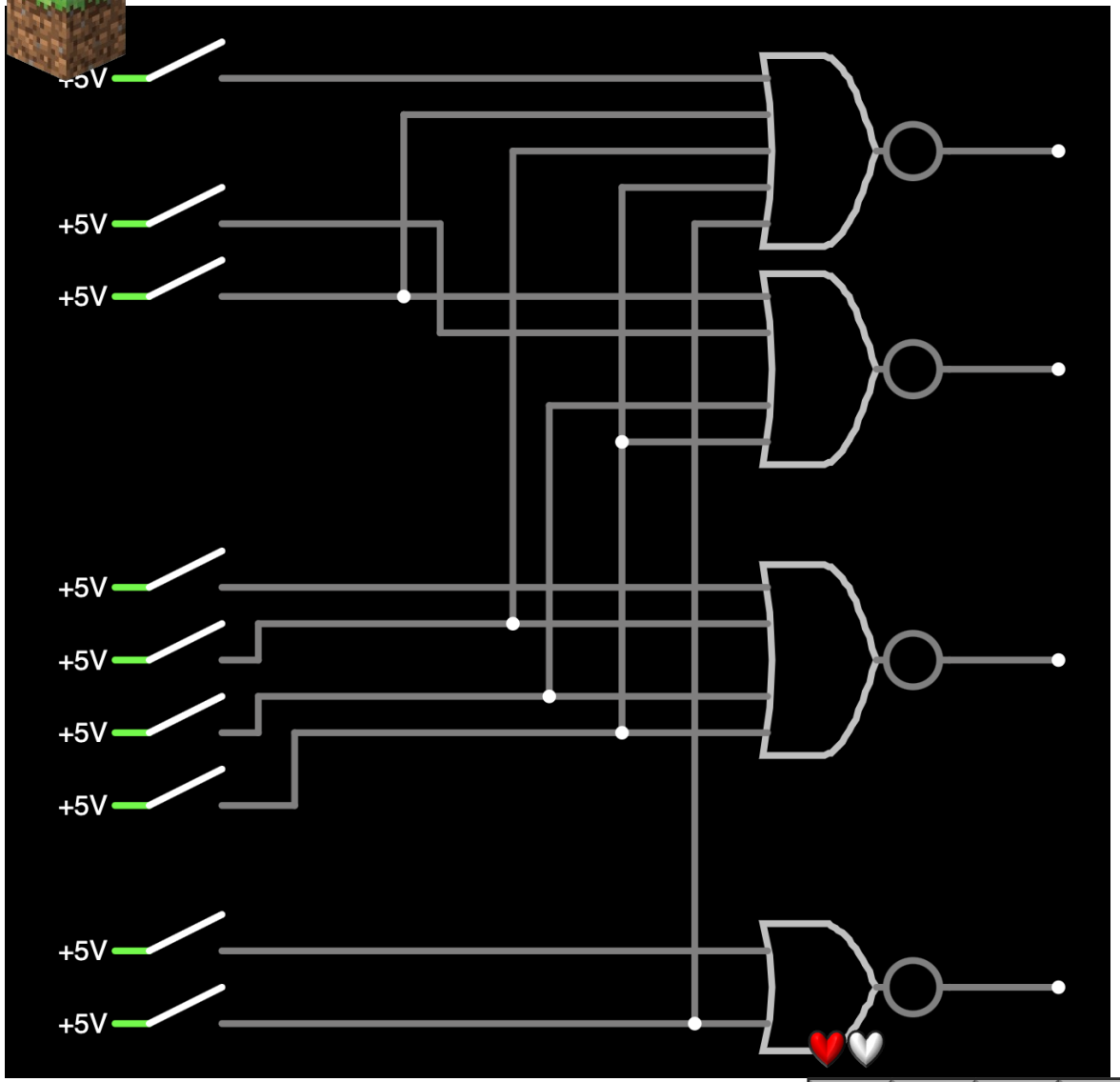
Circuit Simulator를 이용한 **계산기 설계** https://www.falstad.com/circuit/circuitjs.html

1. 인코더
2. 레지스터 1
3. 레지스터 2
4. 디 멀티플렉서
5. 가산기
6. 감산기
7. 곱셈기
8. 7-Segment

다음 주!



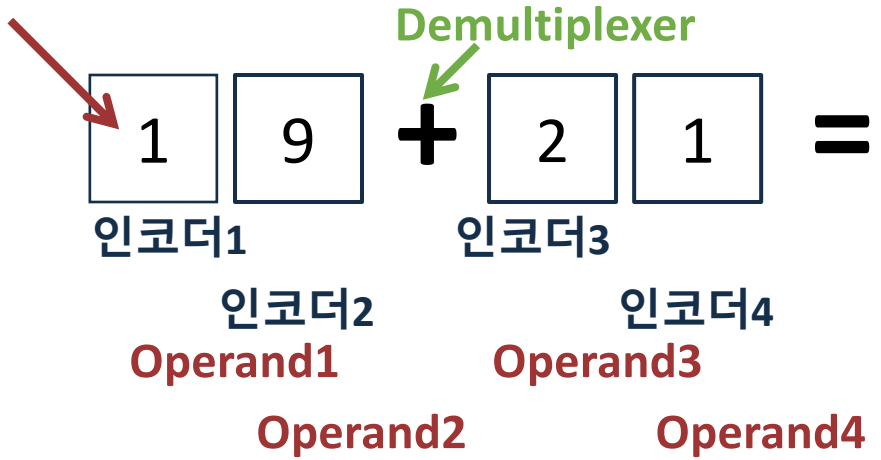
1 전체적인 회로 설계 1



4bit 인코더

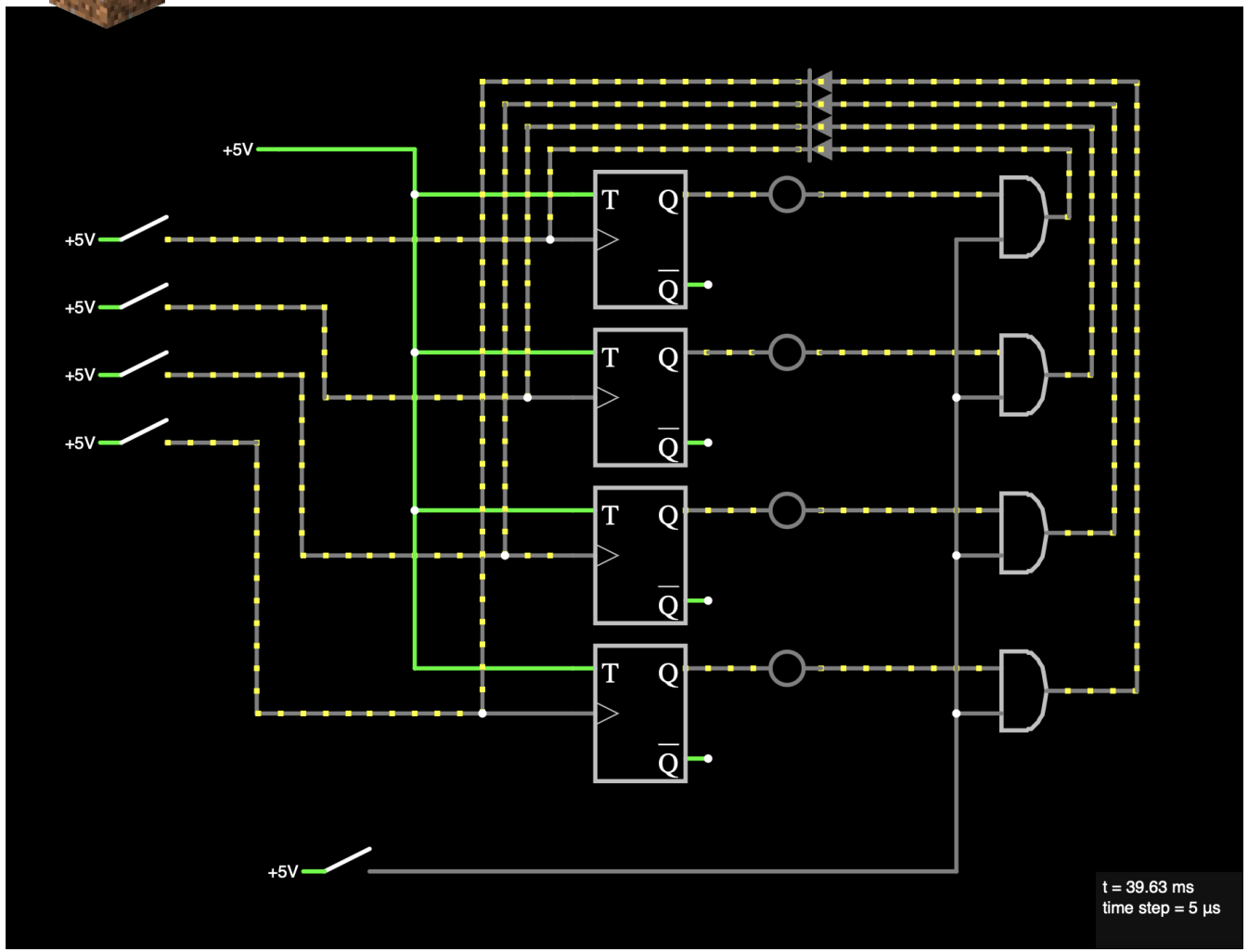
0-9 입력을 **2진수로 변환**

계산기의 가장 **초입**



전체적인 회로 설계 1

Operand1



십진수 한자리 저장하는 **레지스터**

인코더를 통해 변환된 숫자를 **저장**

Clear버튼은 입력값을 **초기화**하는 역할

레지스터는 **플립플롭의 집합**!

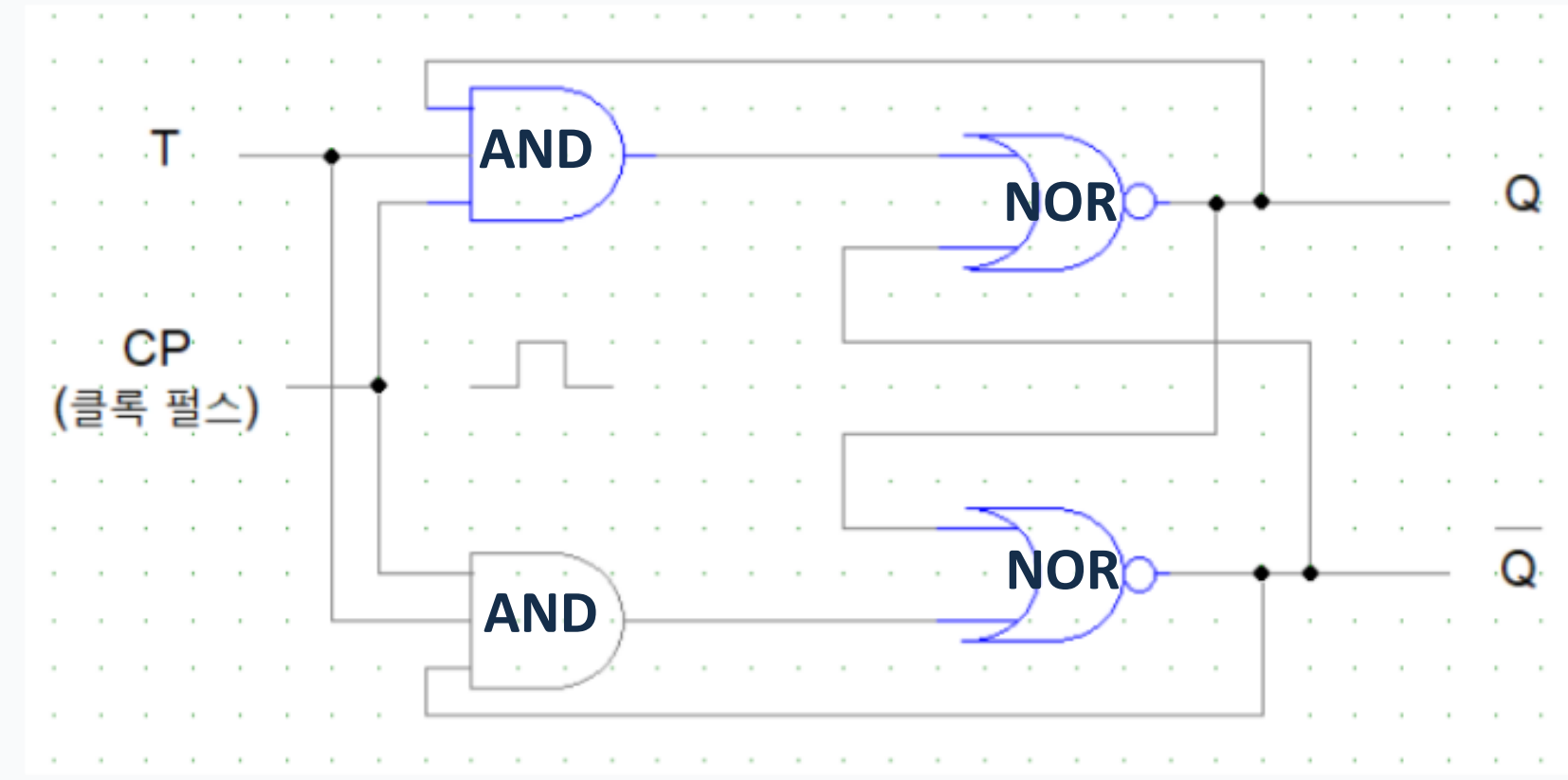
T플립플롭

T	Q(t+1)
0	상태유지 (그대로 출력)
1	반전

정보를 **저장!**

정보를 **초기화!**

T플립플롭



T플립플롭

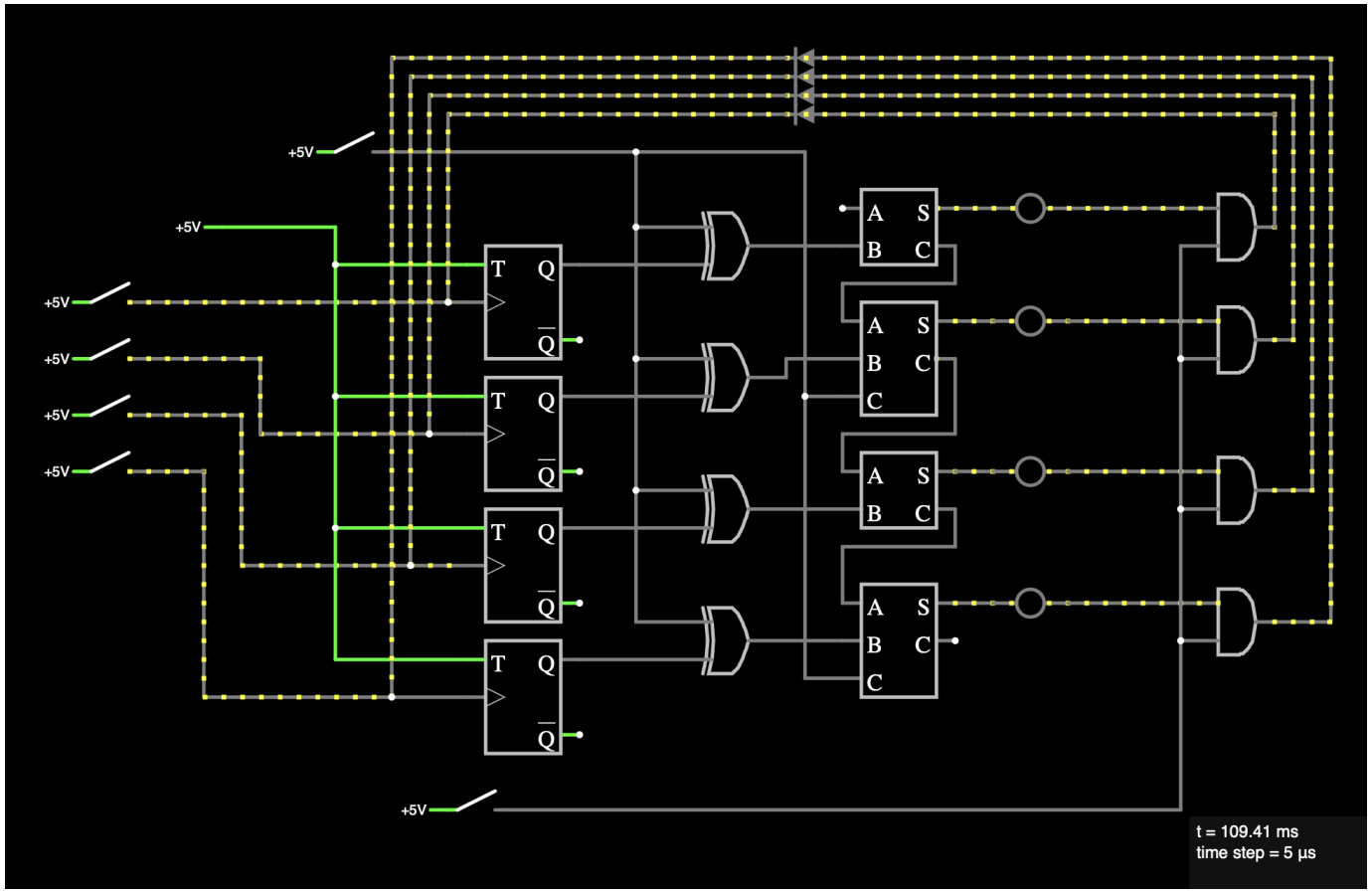
T	Q(t+1)
0	상태유지 (그대로 출력)
1	반전

NOR

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

전체적인 회로 설계 1

Operand2



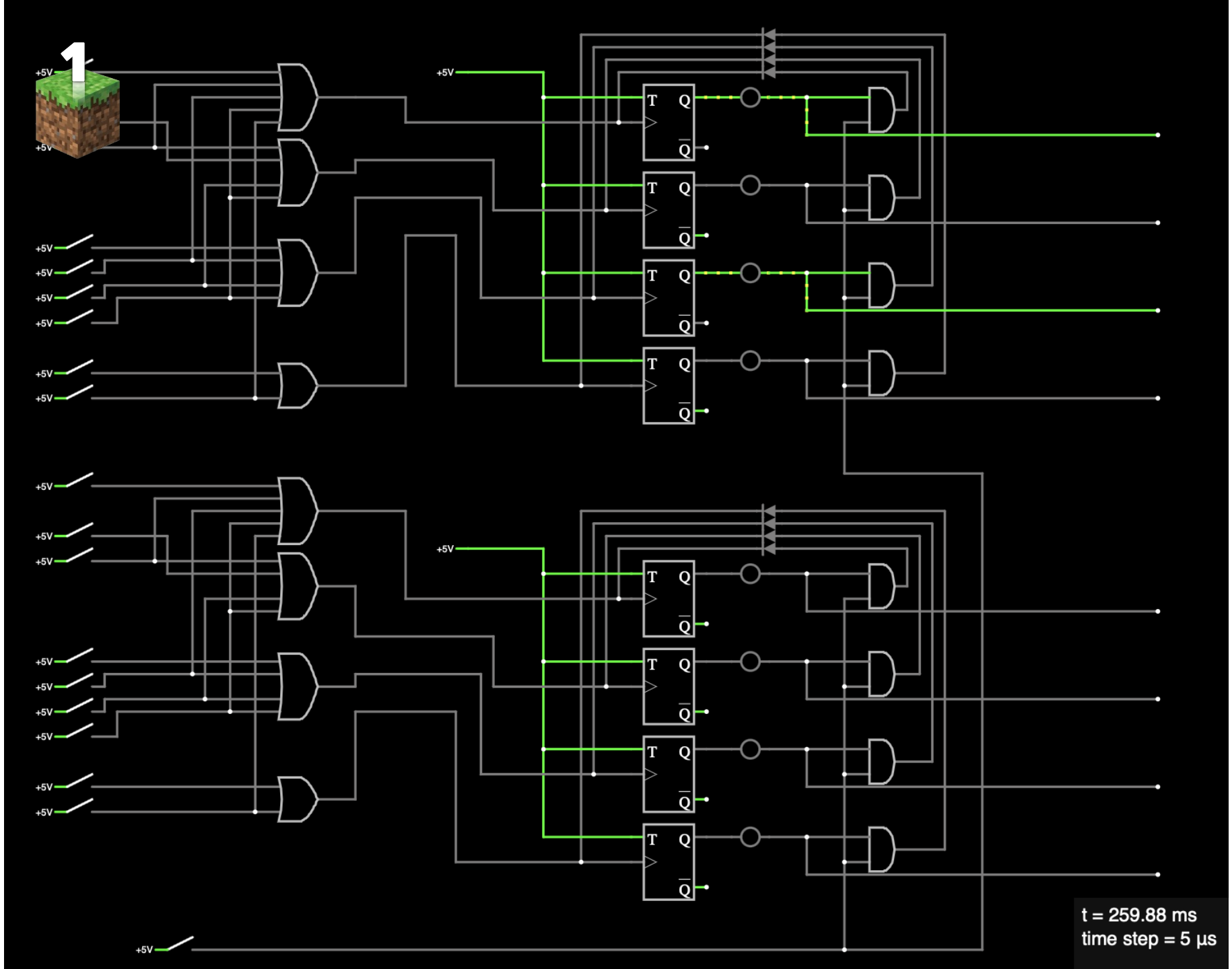
-Operand1 + (보수기능추가)

-보수란 **음수자체를 포함**하는 수!

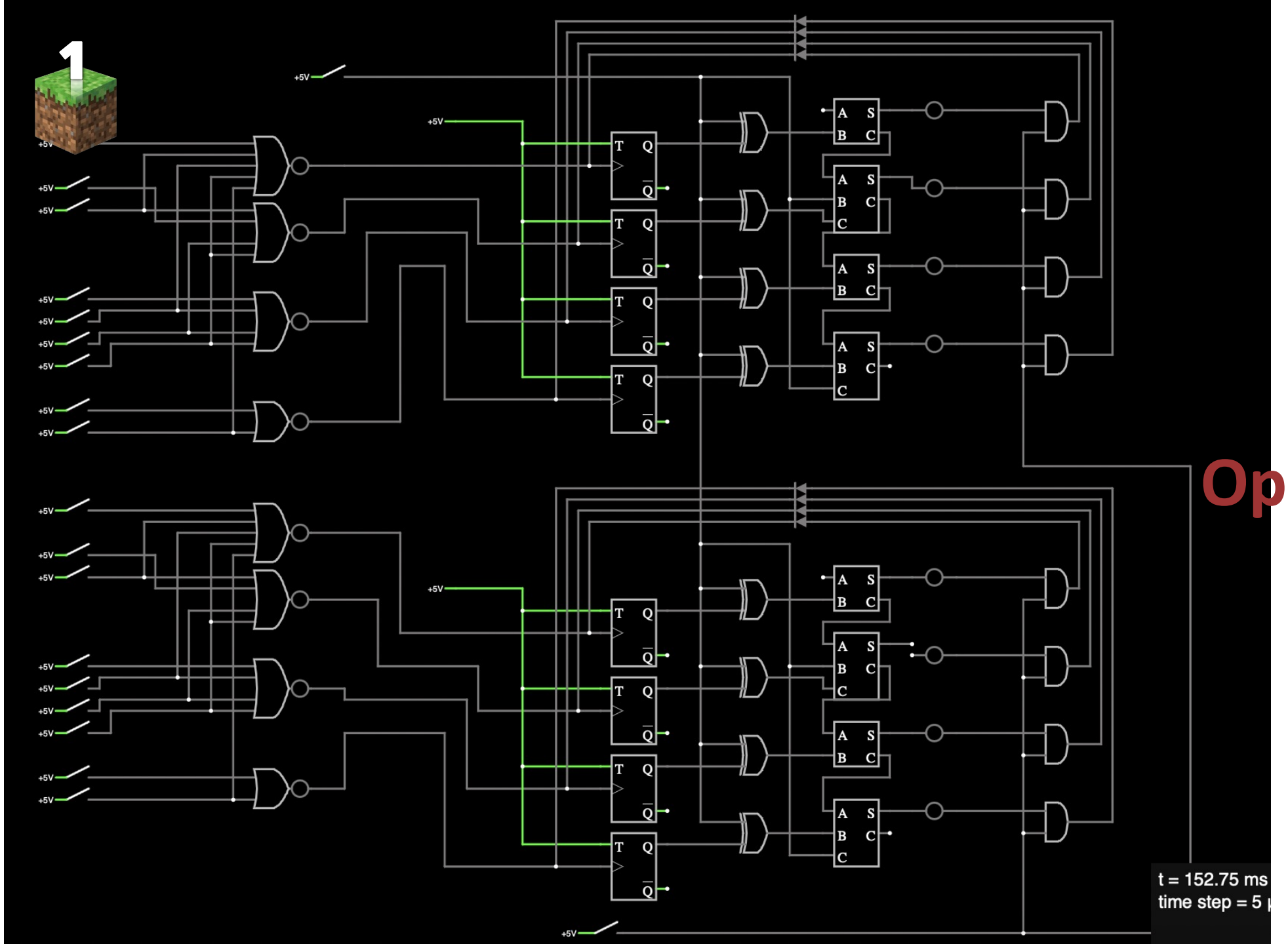
-가산기를 통해 **뺄셈 수행** 가능!

-보수화: **NOT+0001**

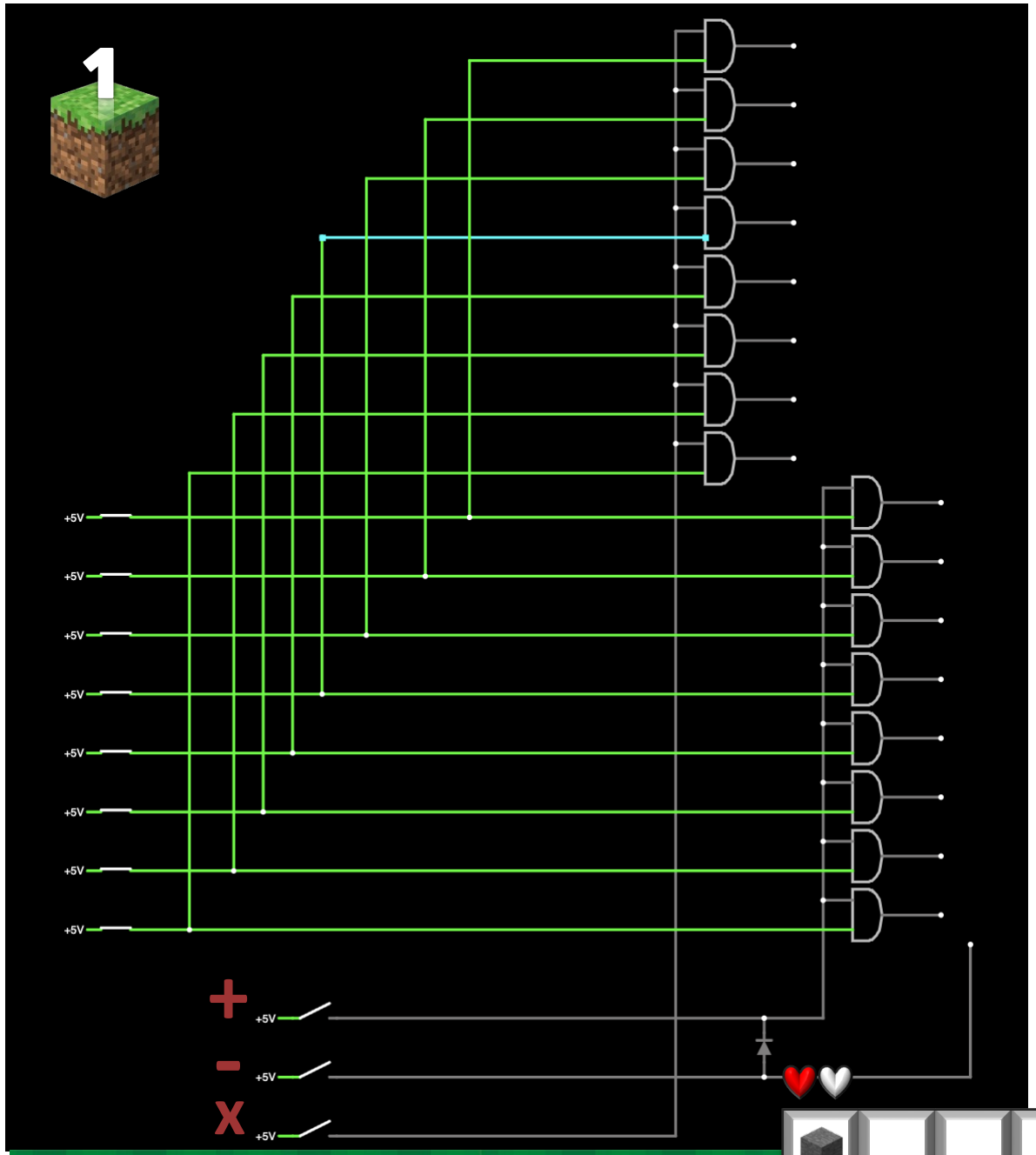
-마인크래프트 건설상황상 operator에 넣을 수도



Operand1



Operand2



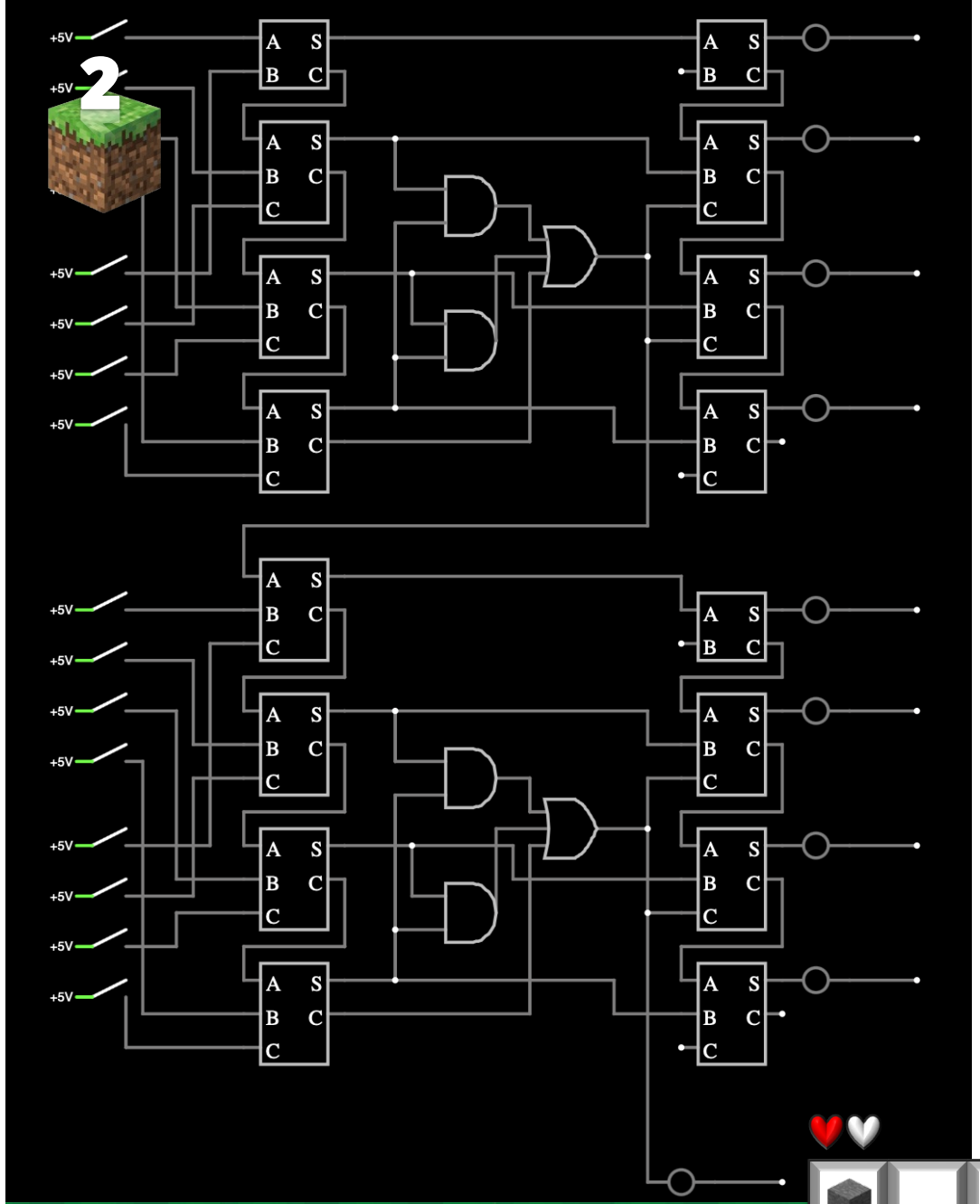
디멀티플렉서(Demultiplexer)

입력된 Operand1, Operand2는 레지스터에 저장되어, 디멀티플렉서를 거쳐서 연산자로 들어간다.

더하기 or 빼기 or 곱하기
연산자(Operator) 를 골라주는 역할

덧셈과 뺄셈은 동시 수행 연산기를 사용
(보수변환必)


2
Preview



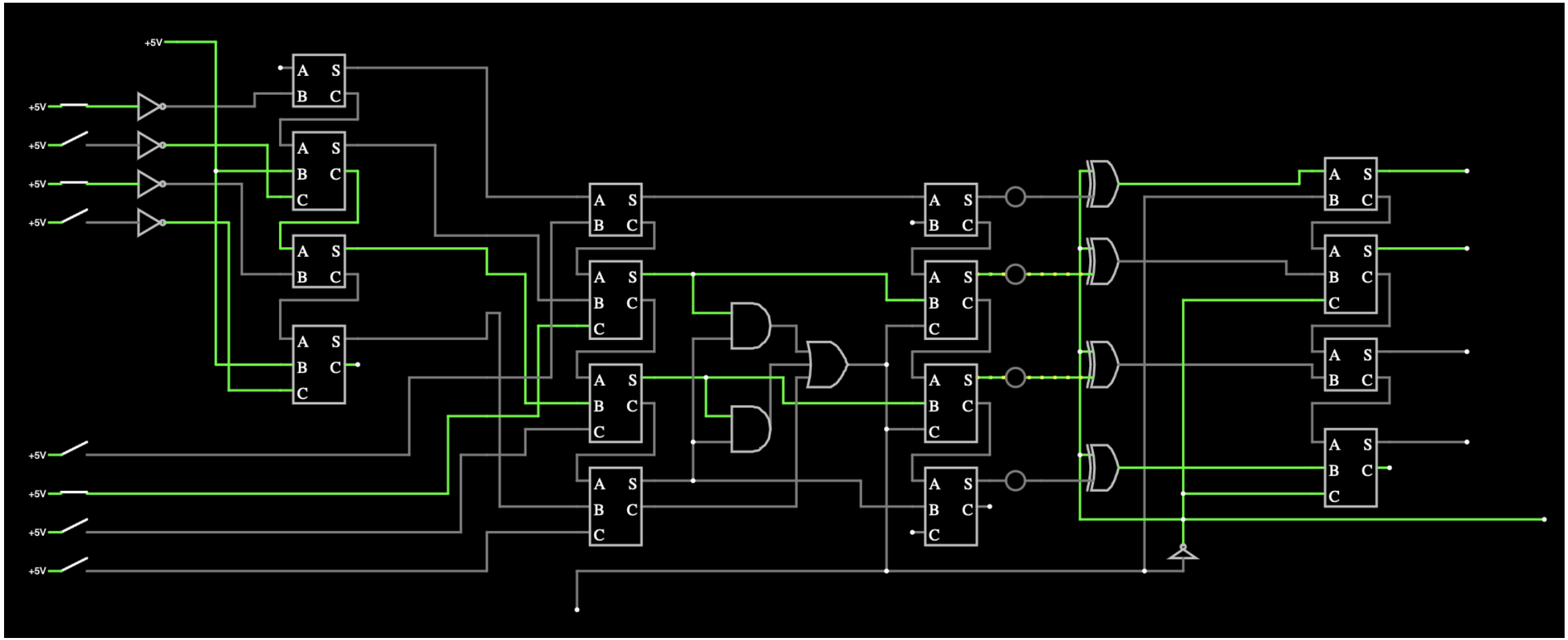
BCD 가산기(BCD Adder)

Operand1 + Operand2 결과를 출력

일반적인 2진법 가산기에
추가적인 AND, OR 게이트를 통해
1010(10), 1011(11), 1100(12), 1101(13), 1110(14),
1111(15) 인 경우, Carry 발생

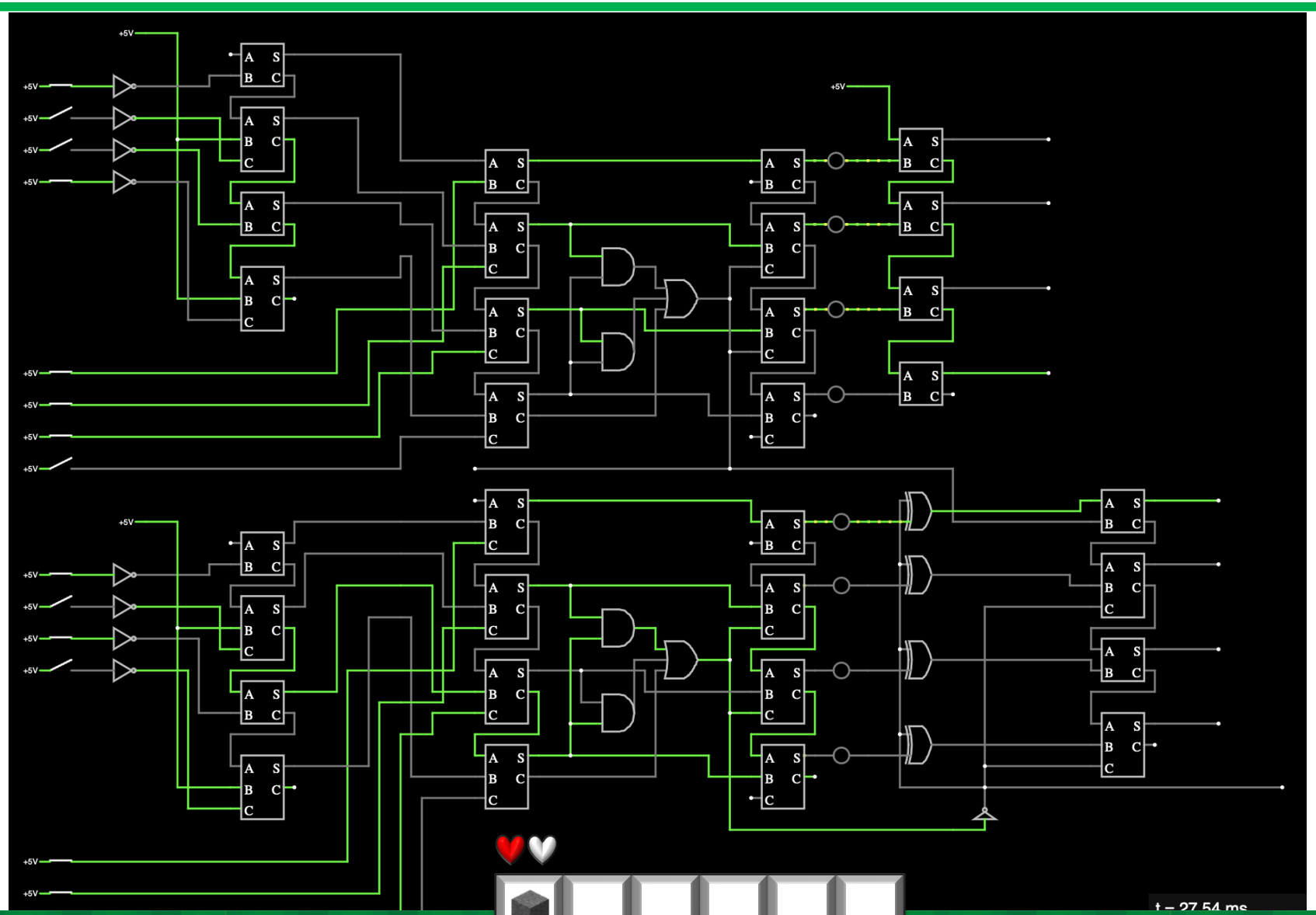
Preview

BCD 감산기(BCD Subtrator)

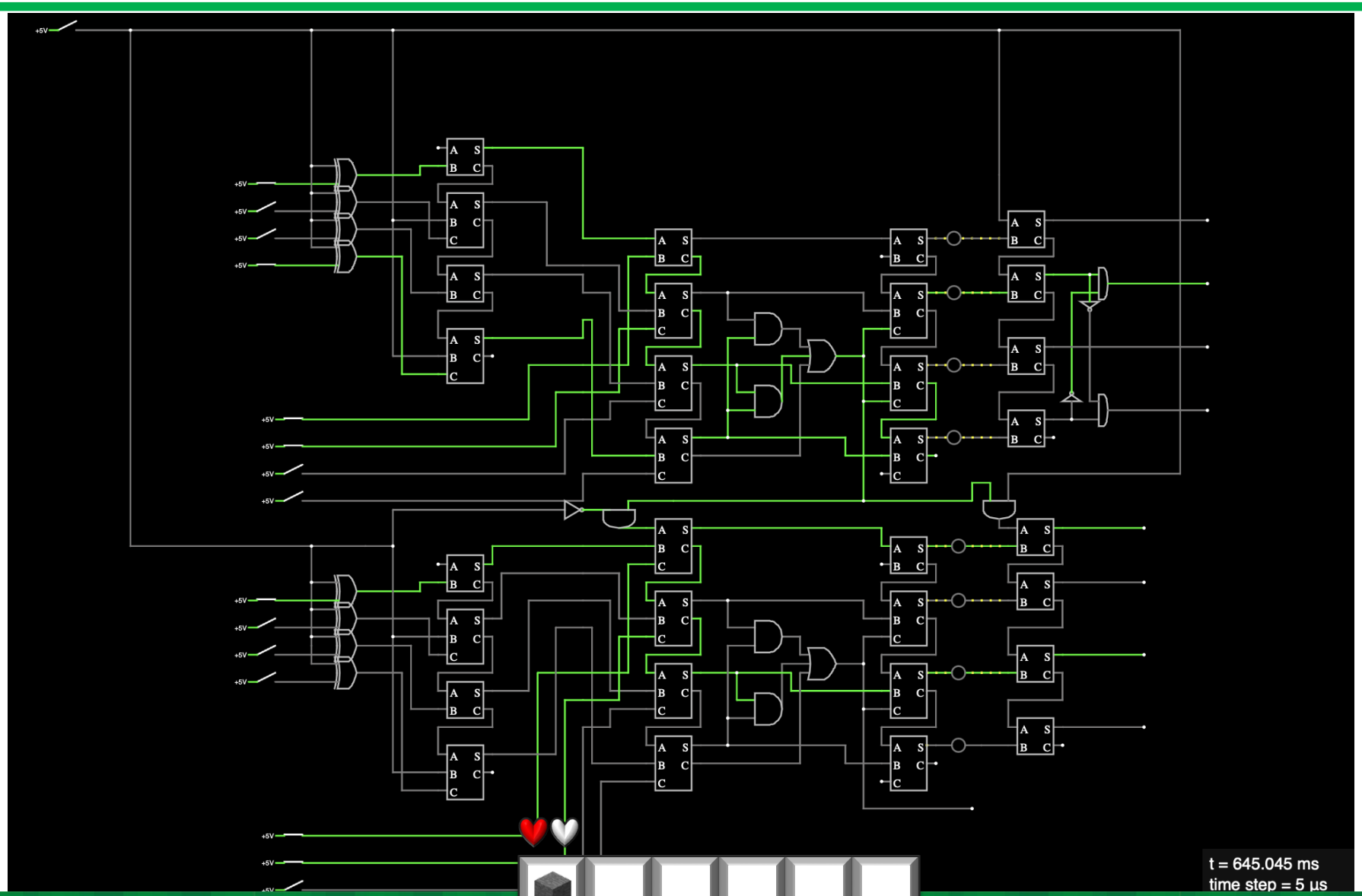


Preview

BCD 감산기(BCD Subtrator)



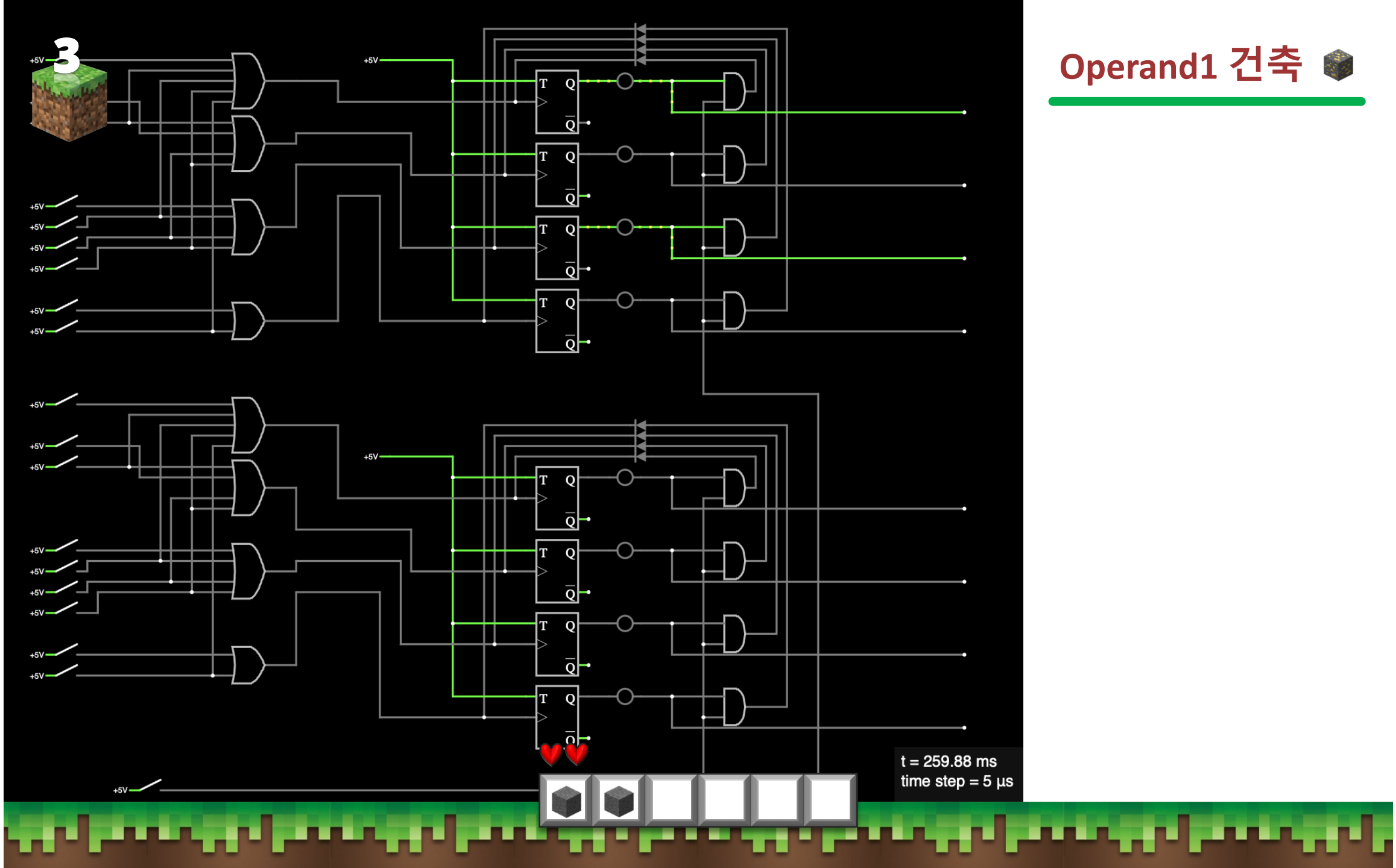
BCD 가산 & 감산 동시 수행 연산기

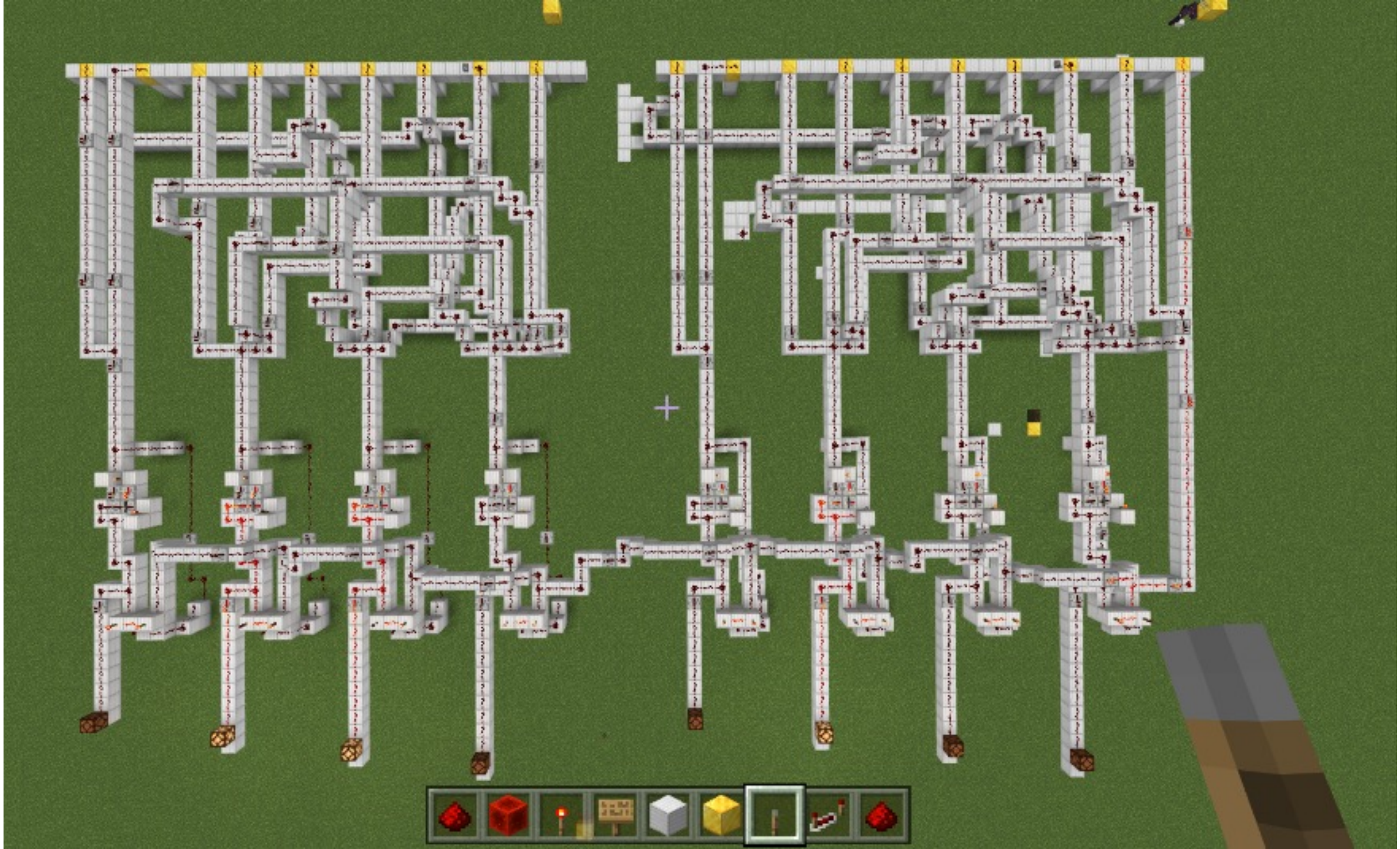


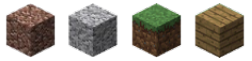
3

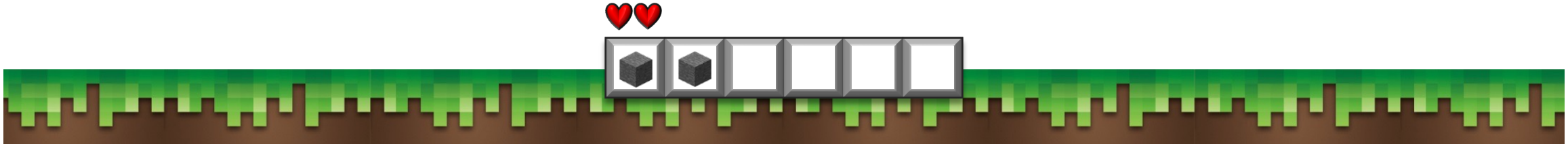
이번주 건축실황

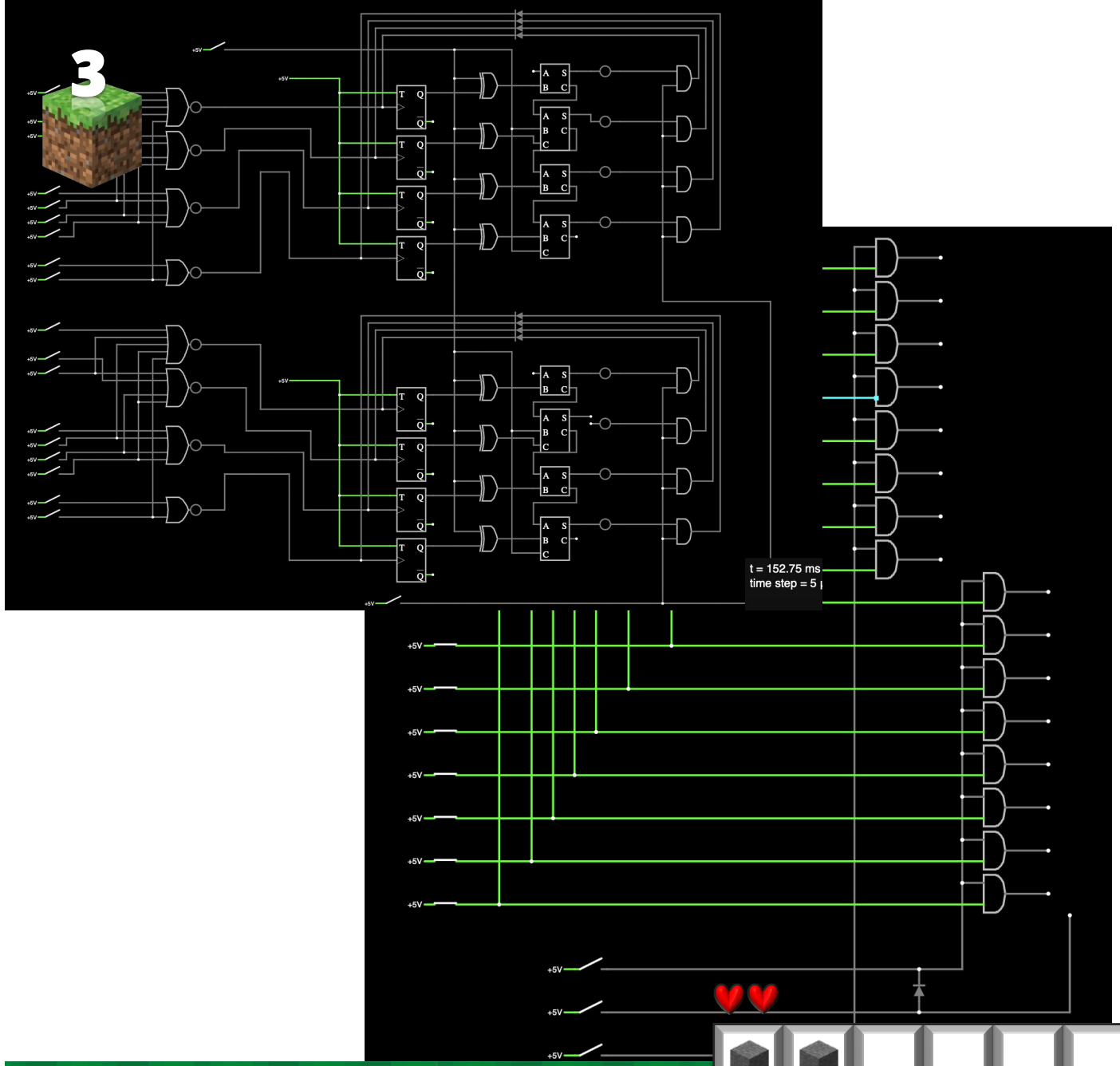
Operand1 건축











다음주 목표

7-Segment 회로설계
동시연산수행기 최적화
디멀티플렉서 최적화

Operand2 건설
디멀티플렉서 건설

Thank you

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !

MINECRAFT

CAPSTONE DEGIGN

전체적인 회로 설계 2

Preview

이번주 건축 실황

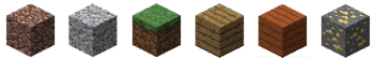
종료

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

1

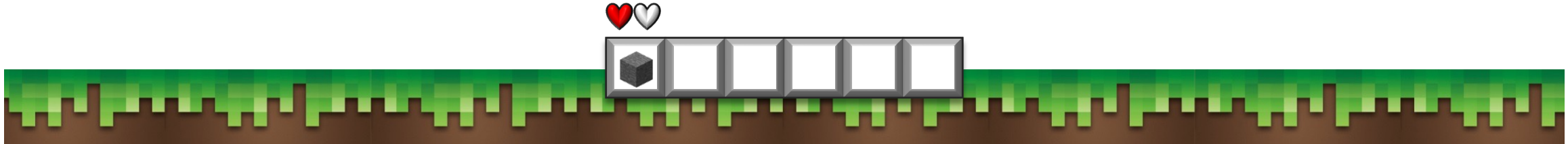
전체적인 회로 설계 2

전체적인 회로 설계 2

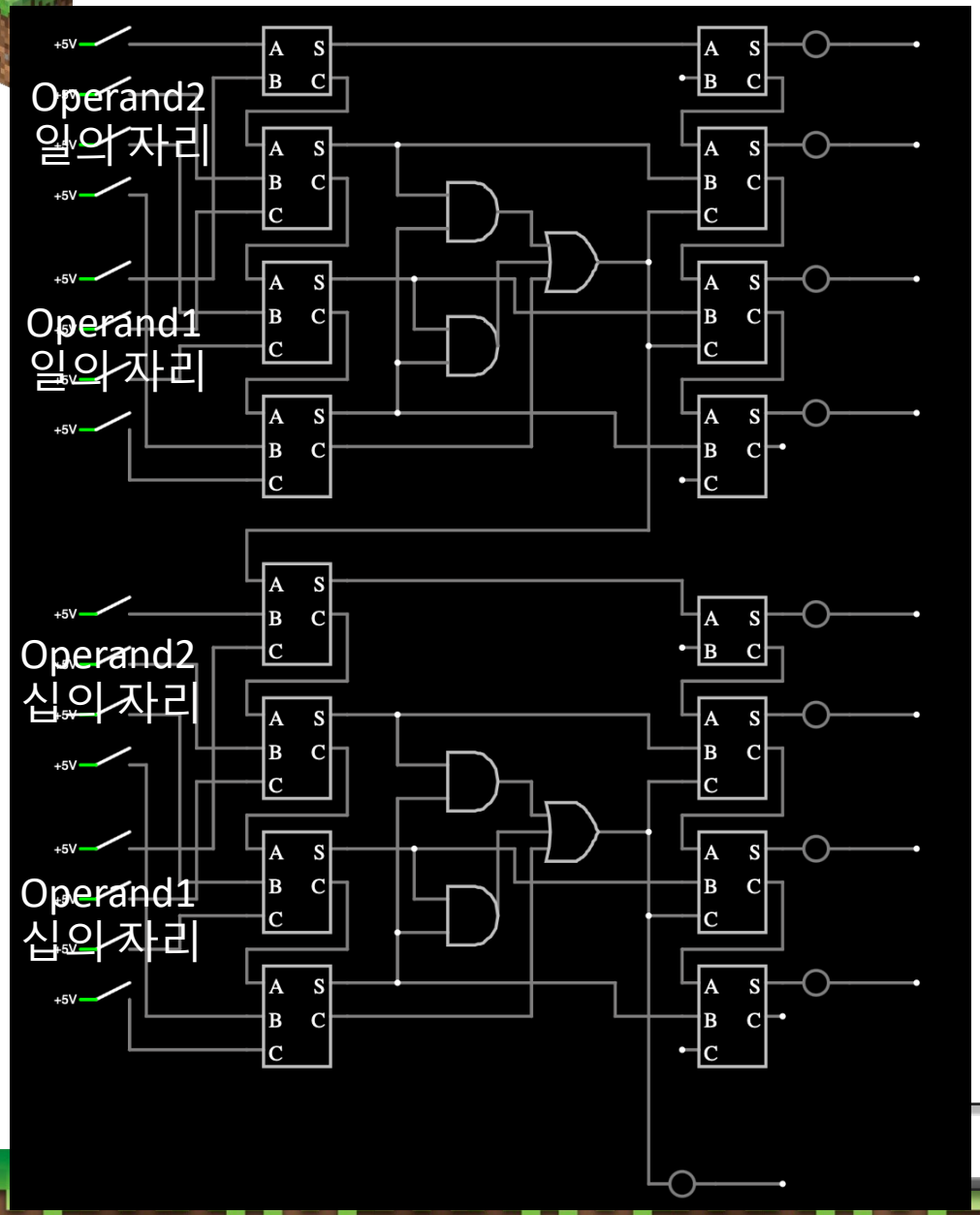


Circuit Simulator를 이용한 **계산기 설계** https://www.falstad.com/circuit/circuitjs.html

1. 인코더
2. 레지스터 1
3. 레지스터 2
4. 디 멀티플렉서
5. **가산기**
6. **감산기**
7. 곱셈기
8. 7-Segment



전체적인 회로 설계 2



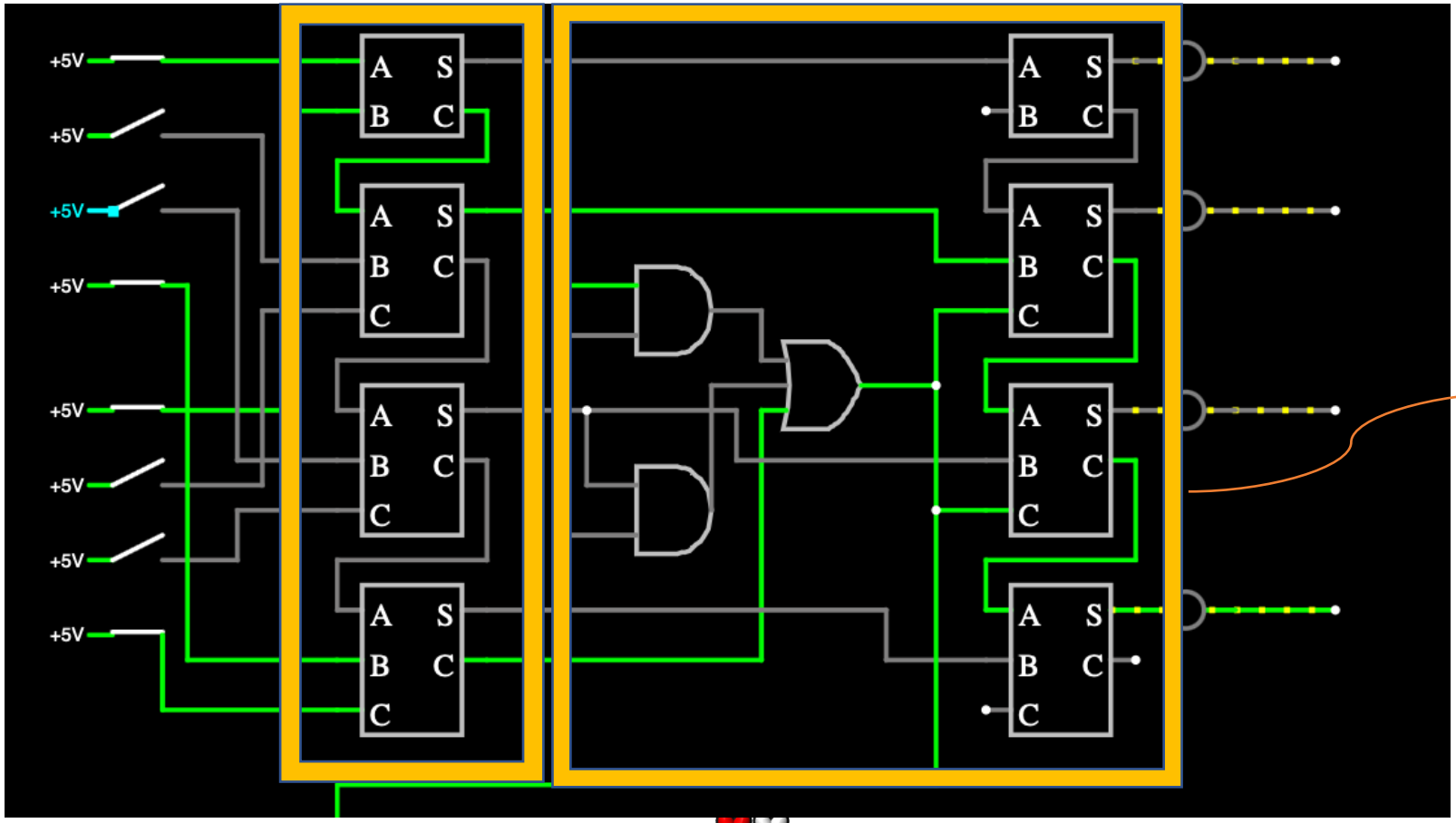
BCD 가산기(BCD Adder)

Operand1 + Operand2 결과를 출력
(두 자리) + (두 자리)

전체적인 회로 설계 2

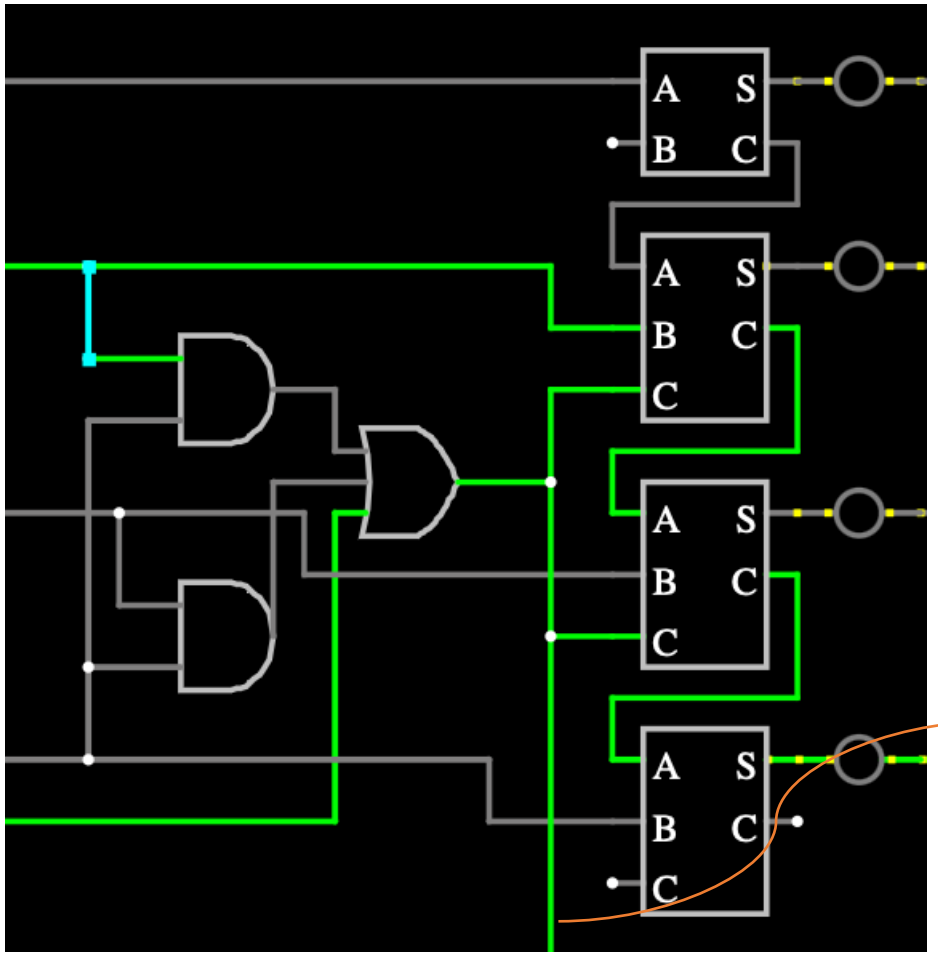
일반적인 2진법 가산기

BCD 가산기(4bit)
자세히 알아보자



BCD 가산을 위한
추가적인 회로

전체적인 회로 설계 2



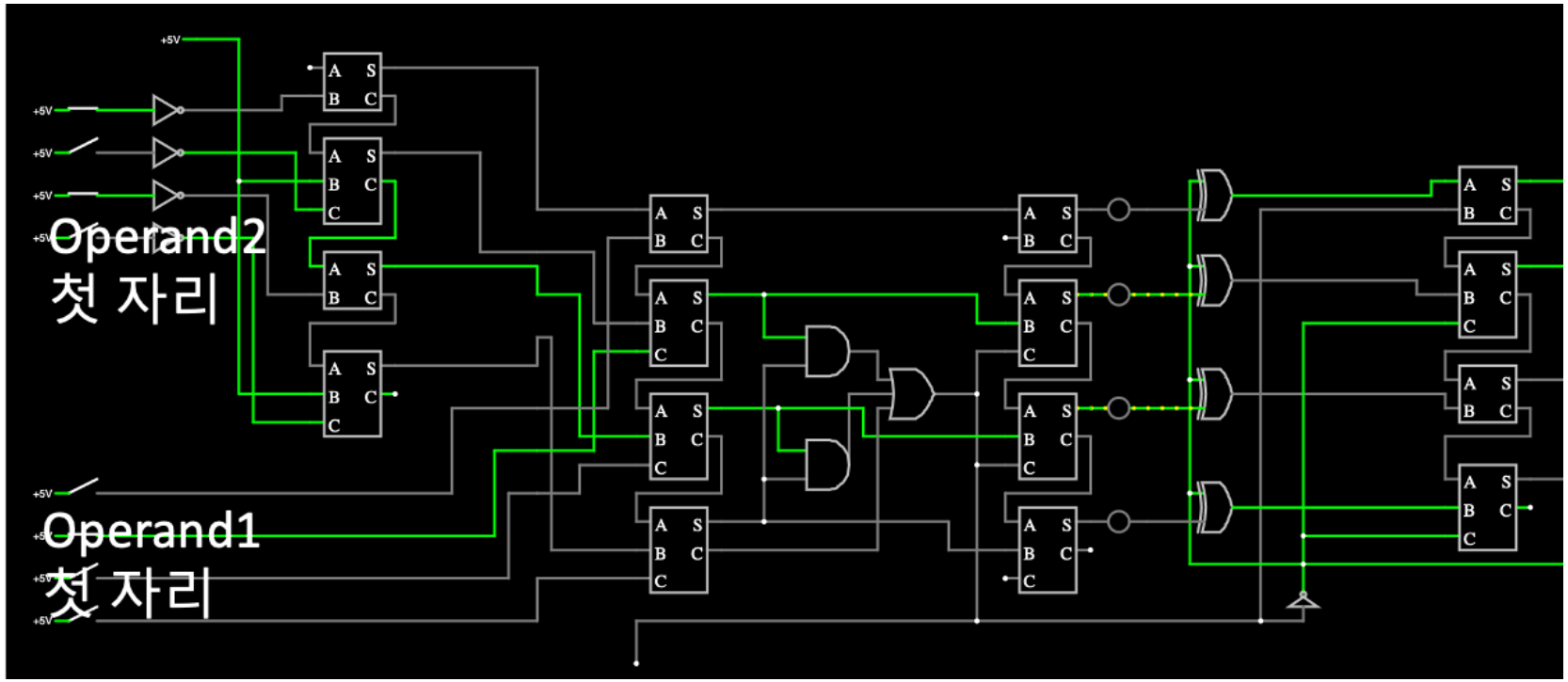
일반적인 2진법 가산기의 출력이 아래인 경우

1010(10) += 0110(6) -> (0)0000
1011(11) += 0110(6) -> (1)0001
1100(12) += 0110(6) -> (1)0010
1101(13) += 0110(6) -> (1)0011
1110(14) += 0110(6) -> (1)0100
1111(15) += 0110(6) -> (1)0101

※ 괄호 속 캐리는 회로에서 무시

& 십의 자리에 +1 캐리 수행

전체적인 회로 설계 2

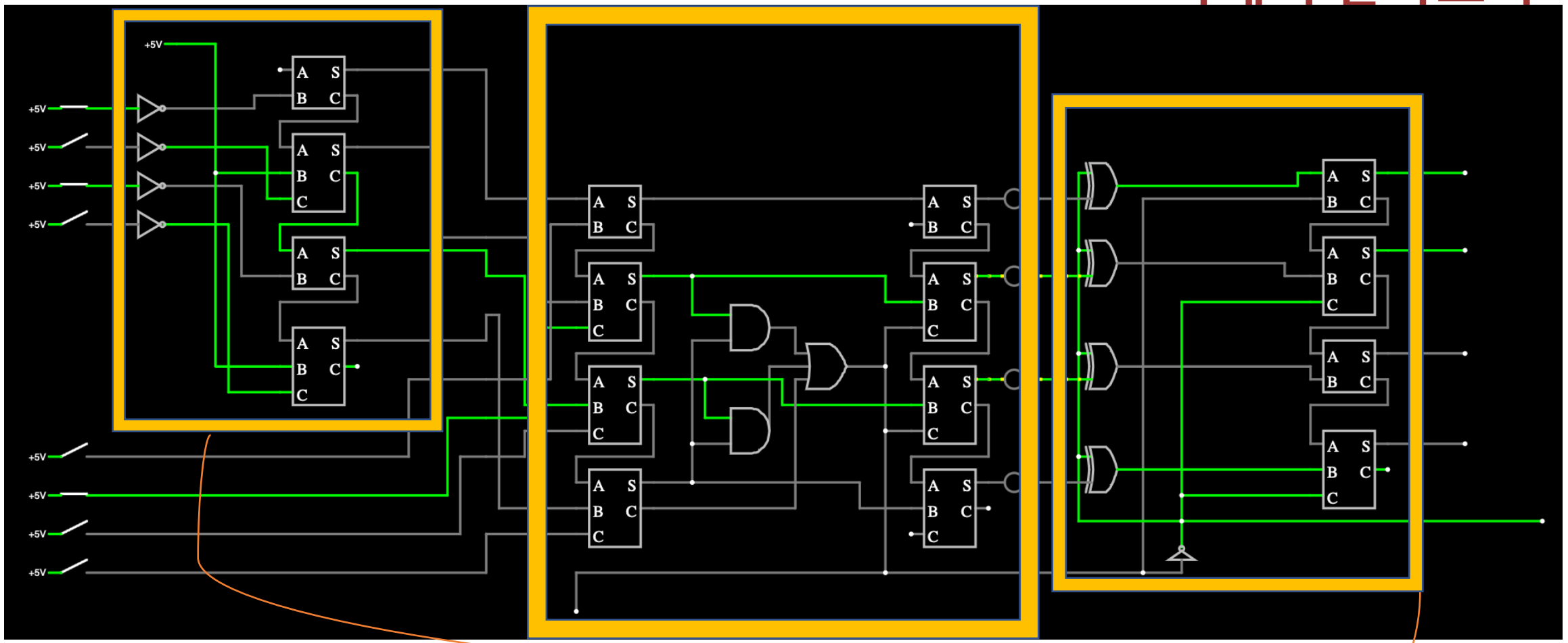


BCD 감산기(4bit)

Operand1 – Operand2
결과를 출력

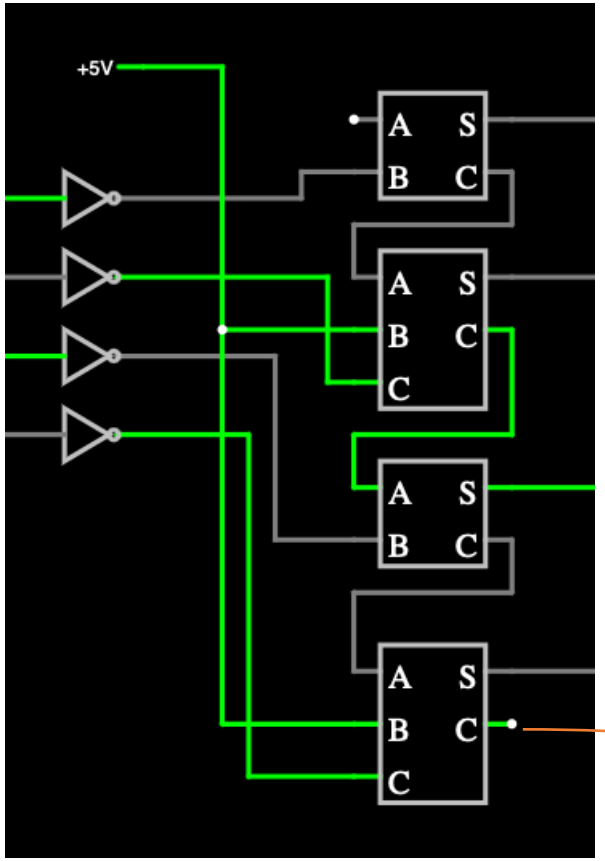
전체적인 회로 설계 2

BCD 감산기(4bit) 자세히 알아보자



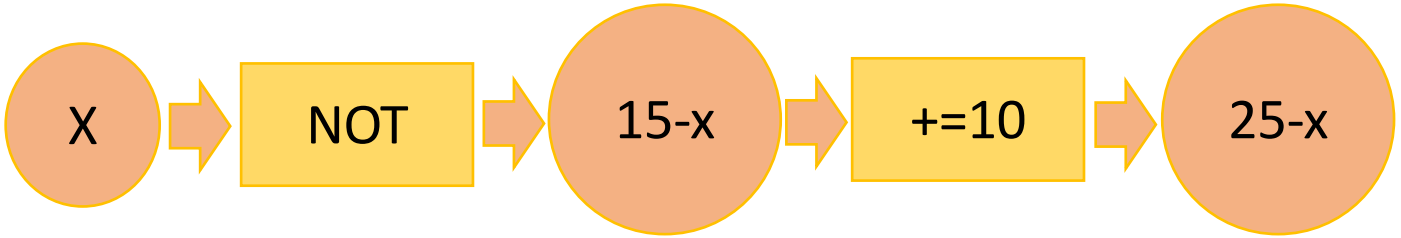
전체적인 회로 설계 2

BCD 감산기(4bit) part 1



Operand1에 대한 9의 보수처리
NOT gate + 1010(10)

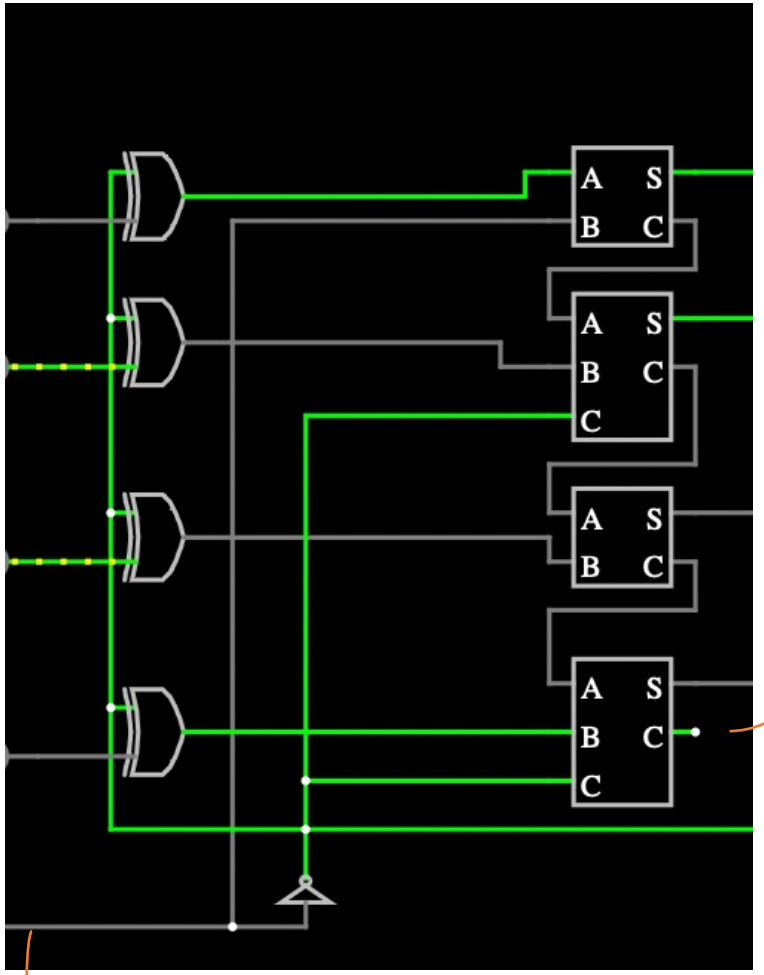
비트에 대한 NOT 연산은 1의 보수 처리를 의미
입력을 “X” 라고 할 경우,



Carry(2^4)을 무시할 시, -=16
9-X == 9’s 보수

전체적인 회로 설계 2

BCD 감산기(4bit) part 2



BCD 가산기 Carry

Operand1을 Y, Operand2를 X라고 할 때,
(9-X) + Y가 10보다 작은 경우(BCD 가산기 Carry=0),
9's 보수[(9-X) + Y]

(9-X) + Y가 10이거나 큰 경우 (BCD 가산기 Carry=1),
(9-X) + Y +=1

BCD에서 Carry 무시는 -=10을 의미

즉,
Y>=X 인 경우, return Y-X

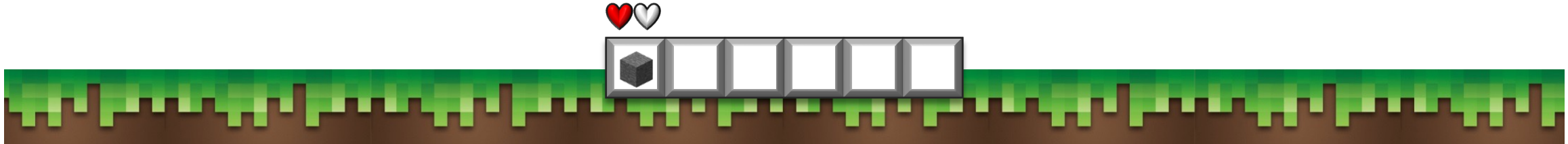
Y<X 인경우, return X-Y & (-)sign ON

BCD 감산기(8bit)

2자리수 이상의 감산 연산을 위해서는 4bit 감산기에서
Part 2의 일부를 변형해 빌림이 가능하도록 회로를 수정해야 함

EX)
77– 59=18 로 일의 자리가 X-Y(or Y-X)가 아님
일의 자리가 Operand1보다 Operand2가 더 큰 경우,
십의 자리가 X-Y(or Y-X)가 아닌 X-Y-1(or Y-X-1)이 되어야 함

->
일의 자리 연산을 (9-X)+Y+1으로 변형
일의 자리에서 Carry 발생 시(Y>=X)에만,
십의 자리 연산에서 +=1 수행

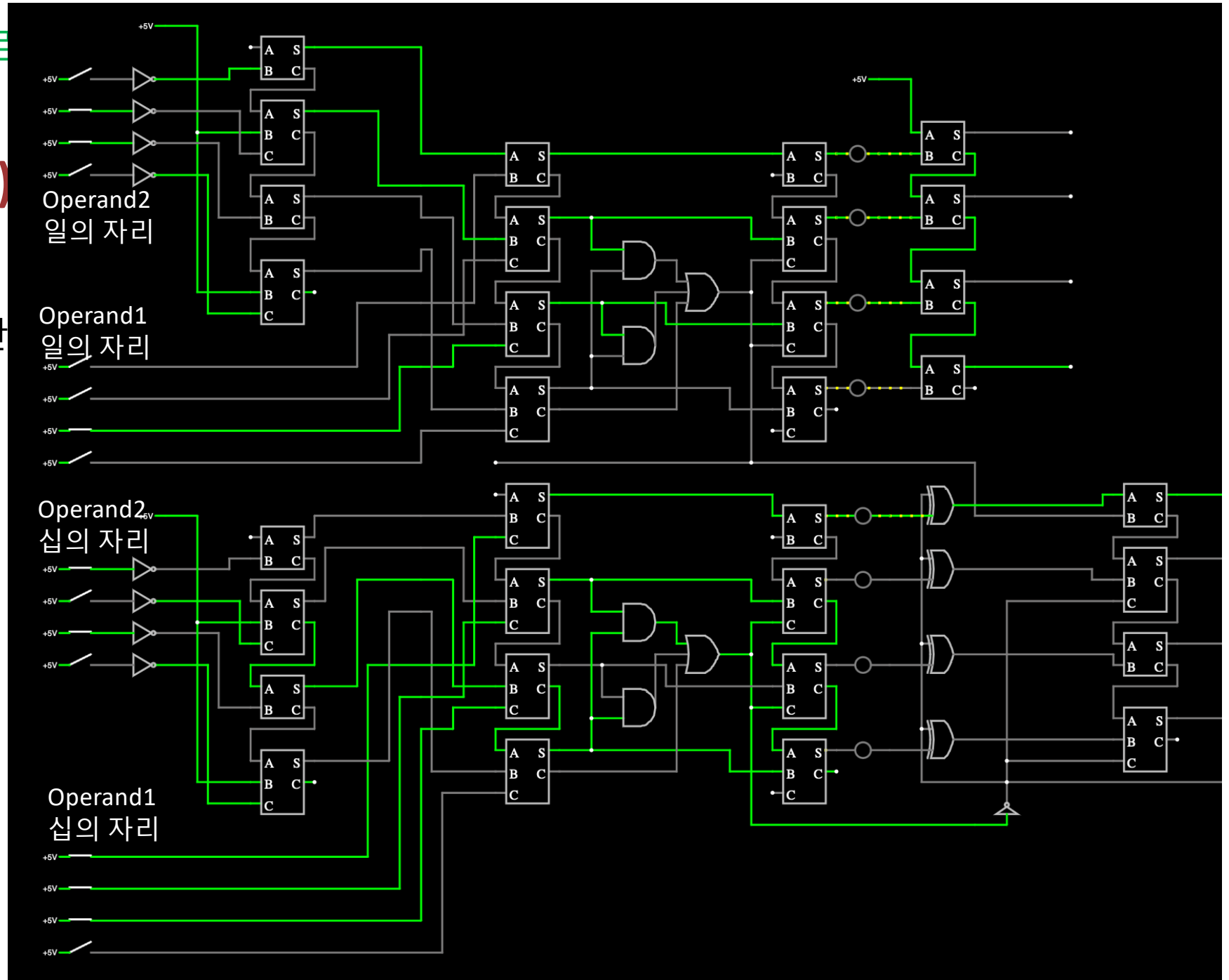


전체적인 회로

BCD 감산기(8bit)

문제점)
(Operand1> Operand2)
인 경우에 대해서만 연산
수행 가능

-> 그냥 안고 가기로 했습니다.ㅜ

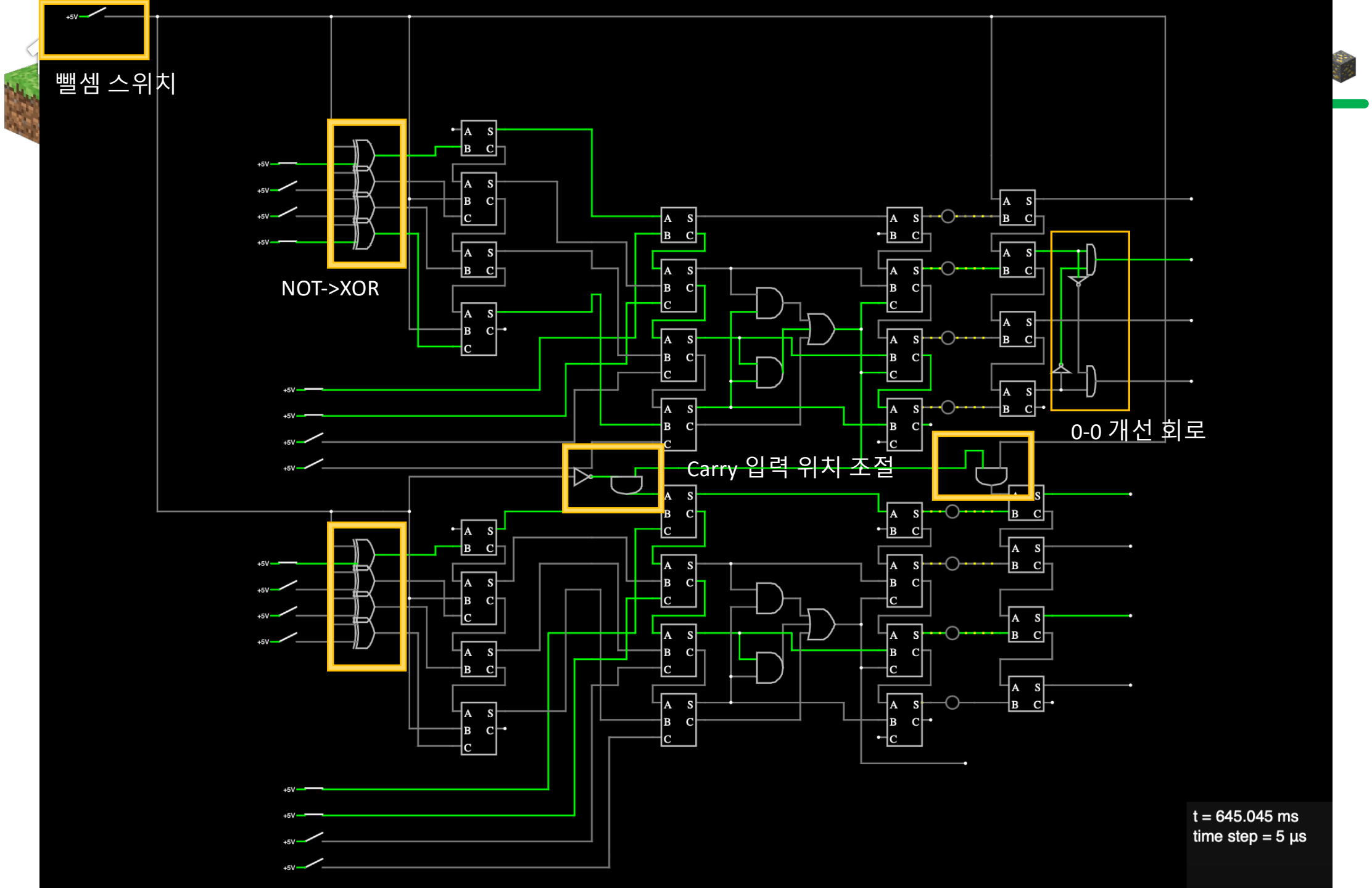


BCD 감&가산기(8bit)

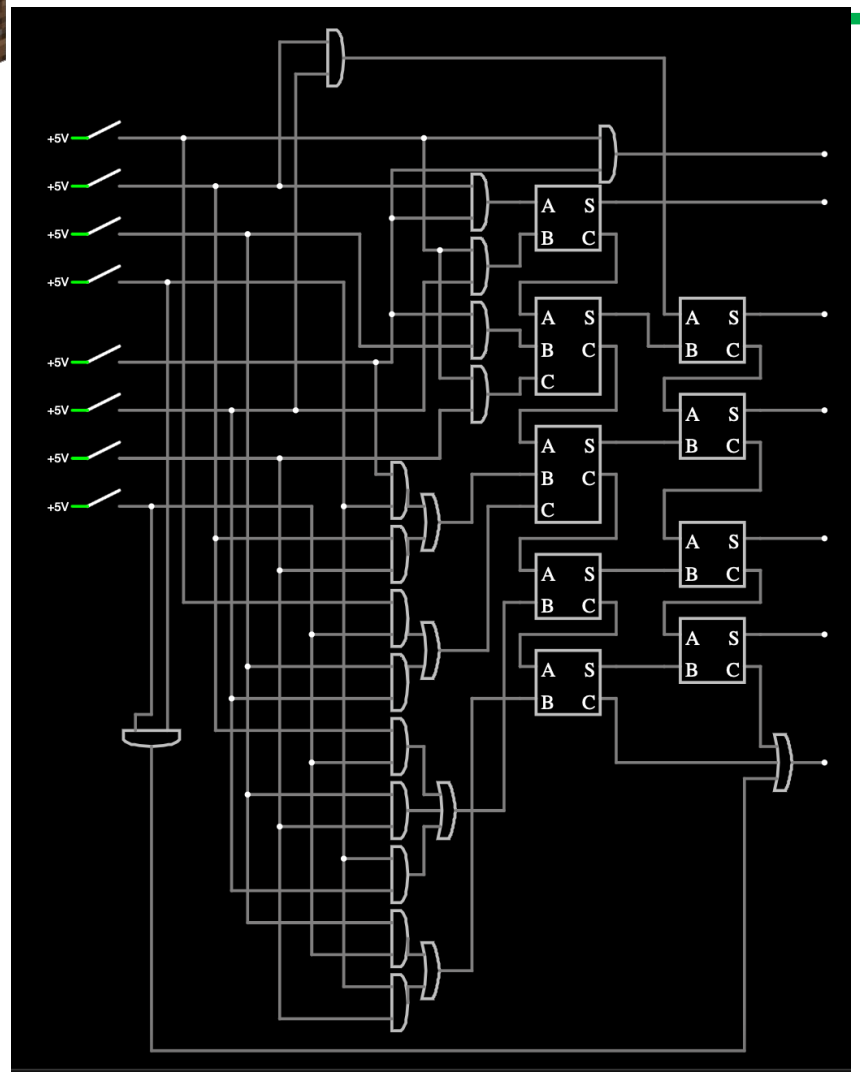
기존 목표대로 가산기와 감산기를 동시에 수행하기 위한 회로
공통으로 사용하는 BCD 가산기를 제외한 회로는 빼기 스위치가 켜진 경우에만 동작

1) 감산기는 가산기 연산에 앞 뒤로 보수 처리를 위한 NOT과 +1010(10) 연산이 포함된다.
-> NOT을 XOR로 변경해서 뺄셈 스위치가 켜진 경우만 NOT 연산 수행
-> 뺄셈 스위치가 꺼진 경우에는 +0000(0) 연산을 스위치가 켜진 경우 +1010(10) 연산을 수행

2) 십의 자리에서 감산기와 가산기 캐리 입력 부분이 다름
-> NOT을 이용해 뺄셈 스위치 0, 1에 따라 다르게 캐리가 입력되도록 수정


빨셈 스위치
NOT->XOR
Carry 입력 위치 조절
0-0 개선 회로
t = 645.045 ms
time step = 5 µs


2
Preview



BCD 곱셈기

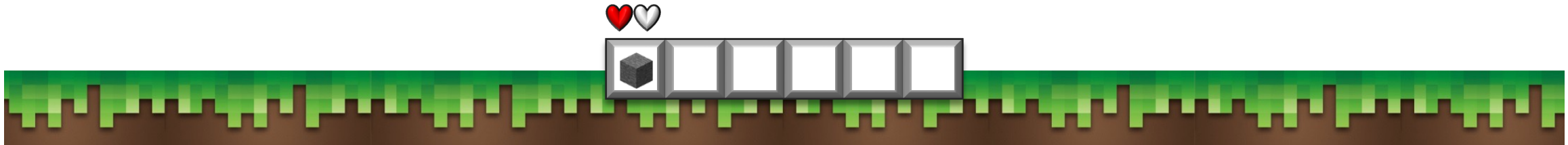
Operand1 X Operand2에 대한 연산 수행

곱셈기는 자리수에 따라 회로가 2^n으로 증가
+ 2자리 x 2자리는 최대 4자리 숫자까지 발생

회로를 구상해보고 결정할 예정

BCD 8bit 곱셈기 회로 구상

7 Segment 회로



3

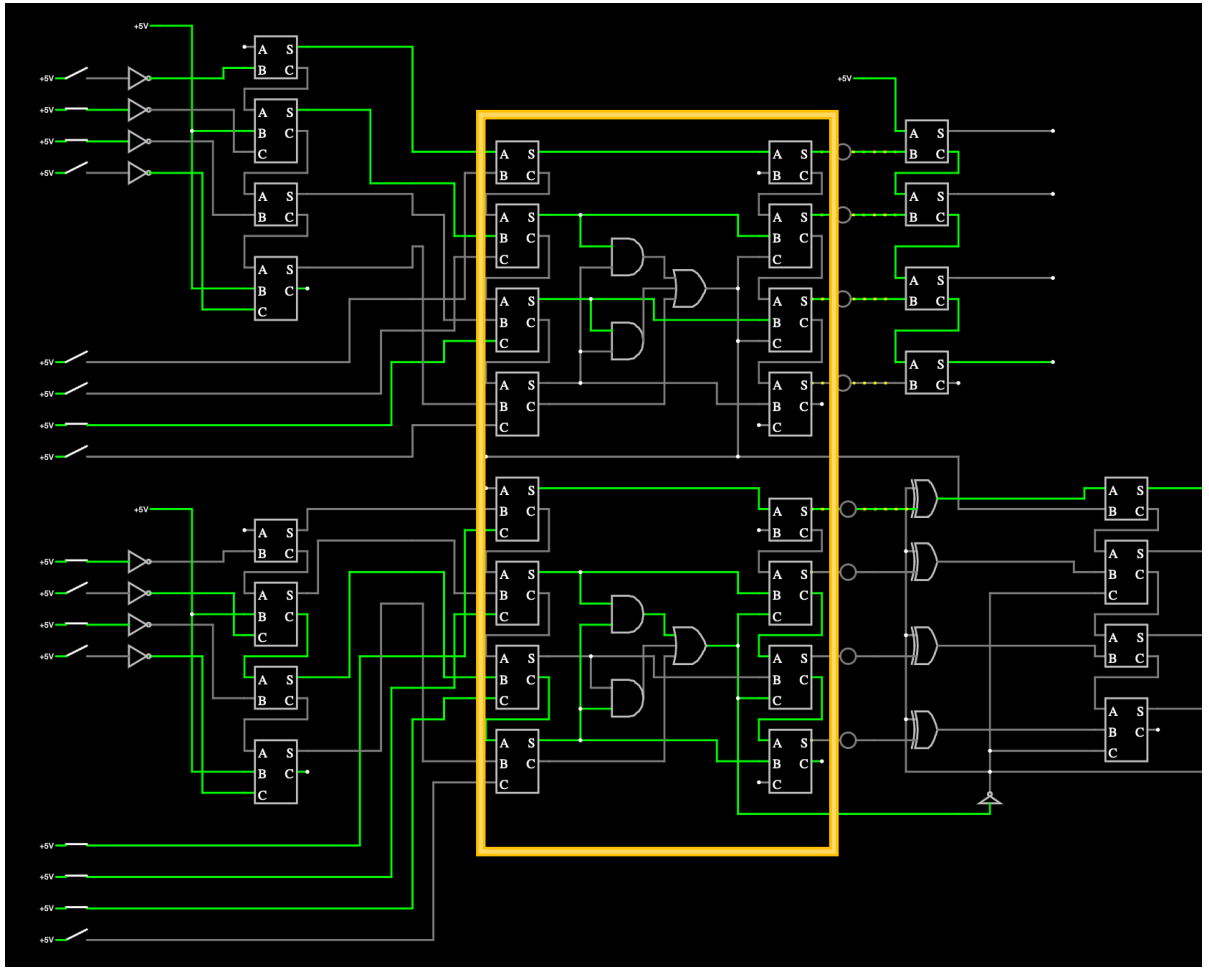
이번주 건축실황

금주 건축 상황

OPERAND1(1층)
OPERAND 2(2층)

DE-MULTIPLEXER:
우측 곱셈기, 좌측 감&가산기

금주 건축 상황



다음주 목표

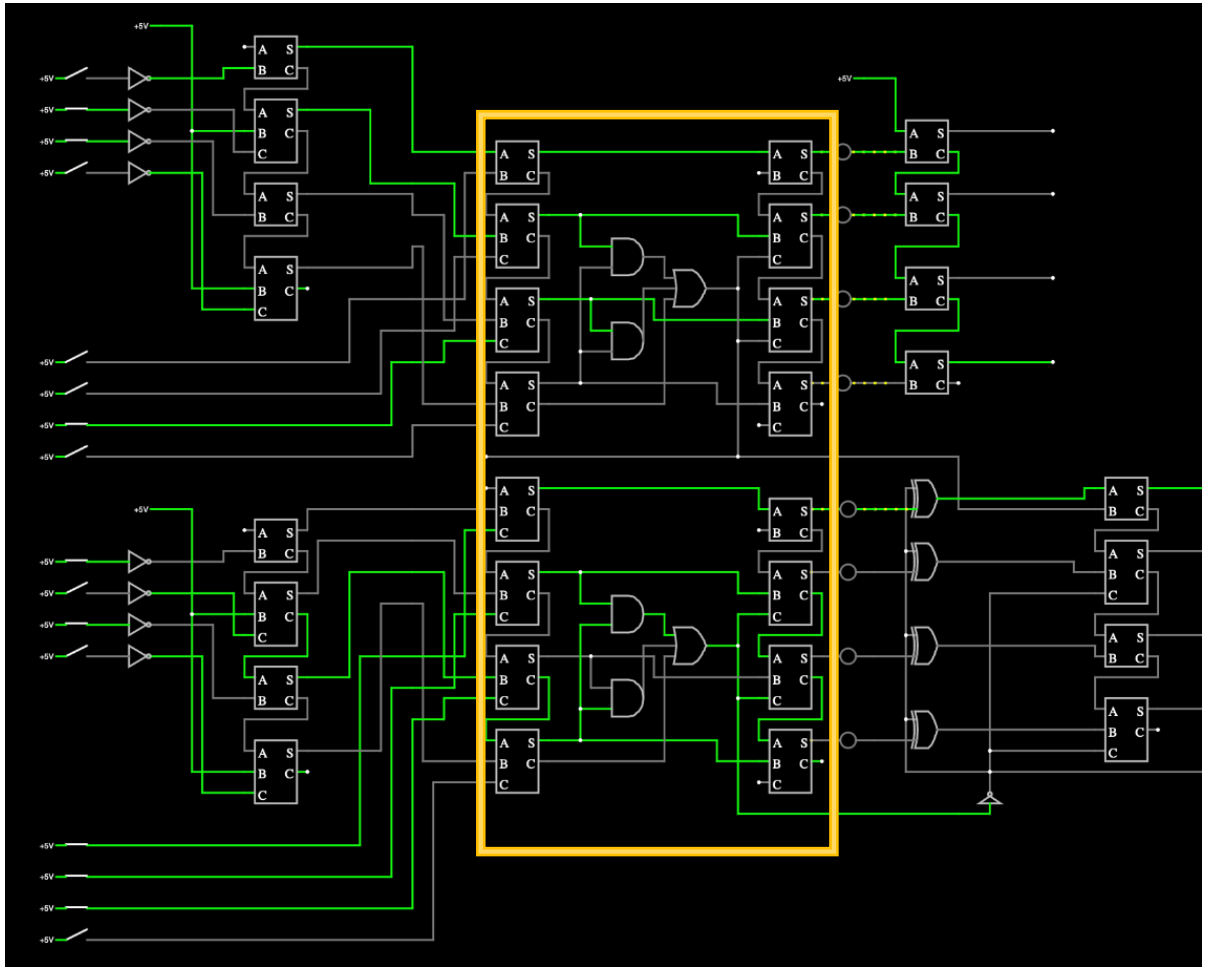
7-Segment 회로 설계
8bit 곱셈기 회로 설계 및 계획

마인크래프트 상에서
BCD 감&가산기의 “가산기” 구현


치: 92, -26, 19
khuEnderman
khuCreeper
khuSkeleTon
khuWither
Thank you
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DEGIGN
Binary Multiplier
BCD Multiplier
이번주 건축 실황
종료
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

Preview



다음주 목표

<회로>

7-Segment 회로 설계

8bit 곱셈기 회로 설계 및 계획

<마인크래프트>

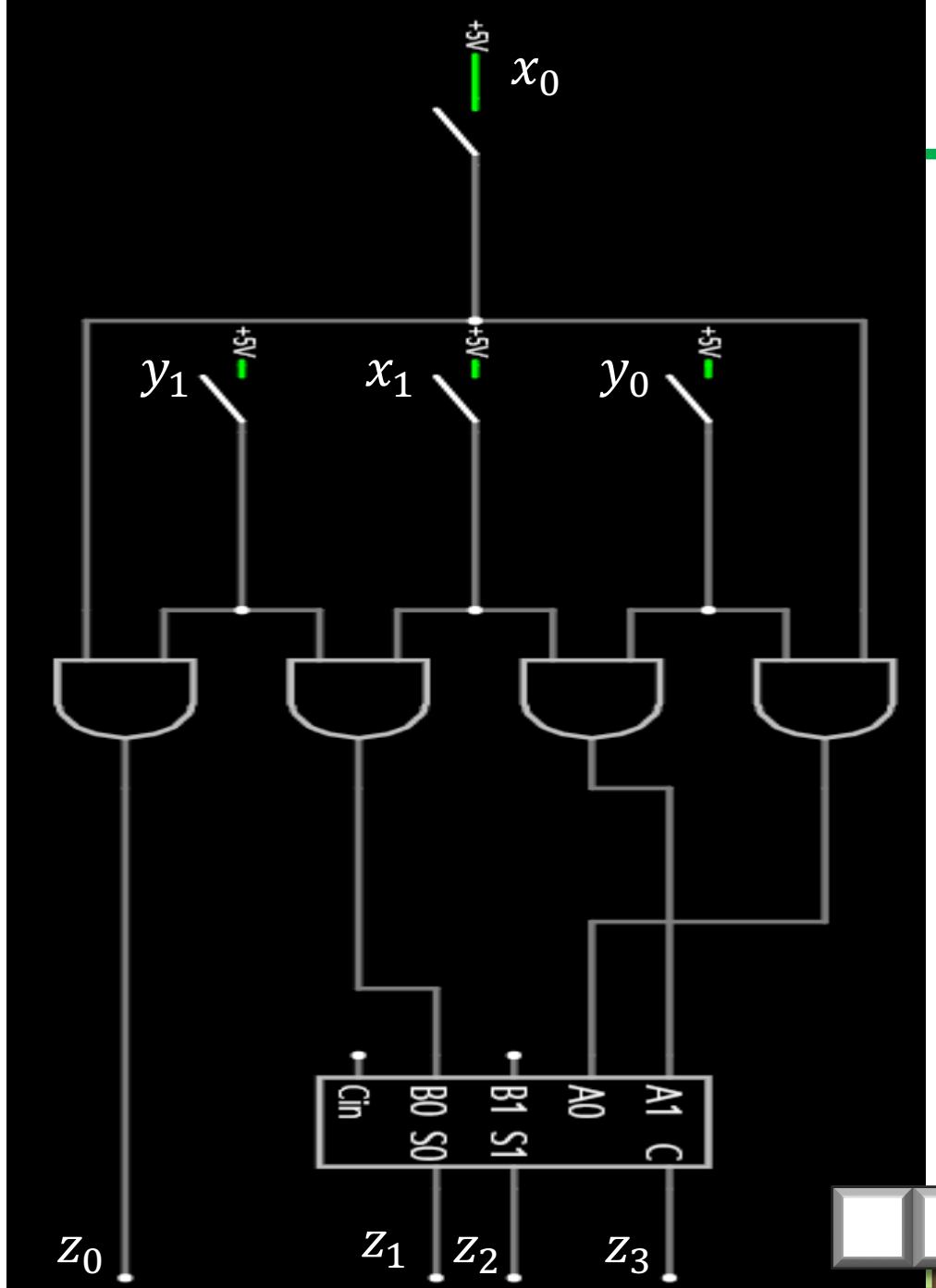
BCD 감가산기 구현

1 Binary Multiplier

2진법에서의 곱셈 (4 bits)

Multiplicand	1011	<- Operand 1
Multiplier	x 0110	<- Operand 2
	0000	
	10110	
	101100	
	0000000	
Product	1000010	

ex) 11 × 6 = 66



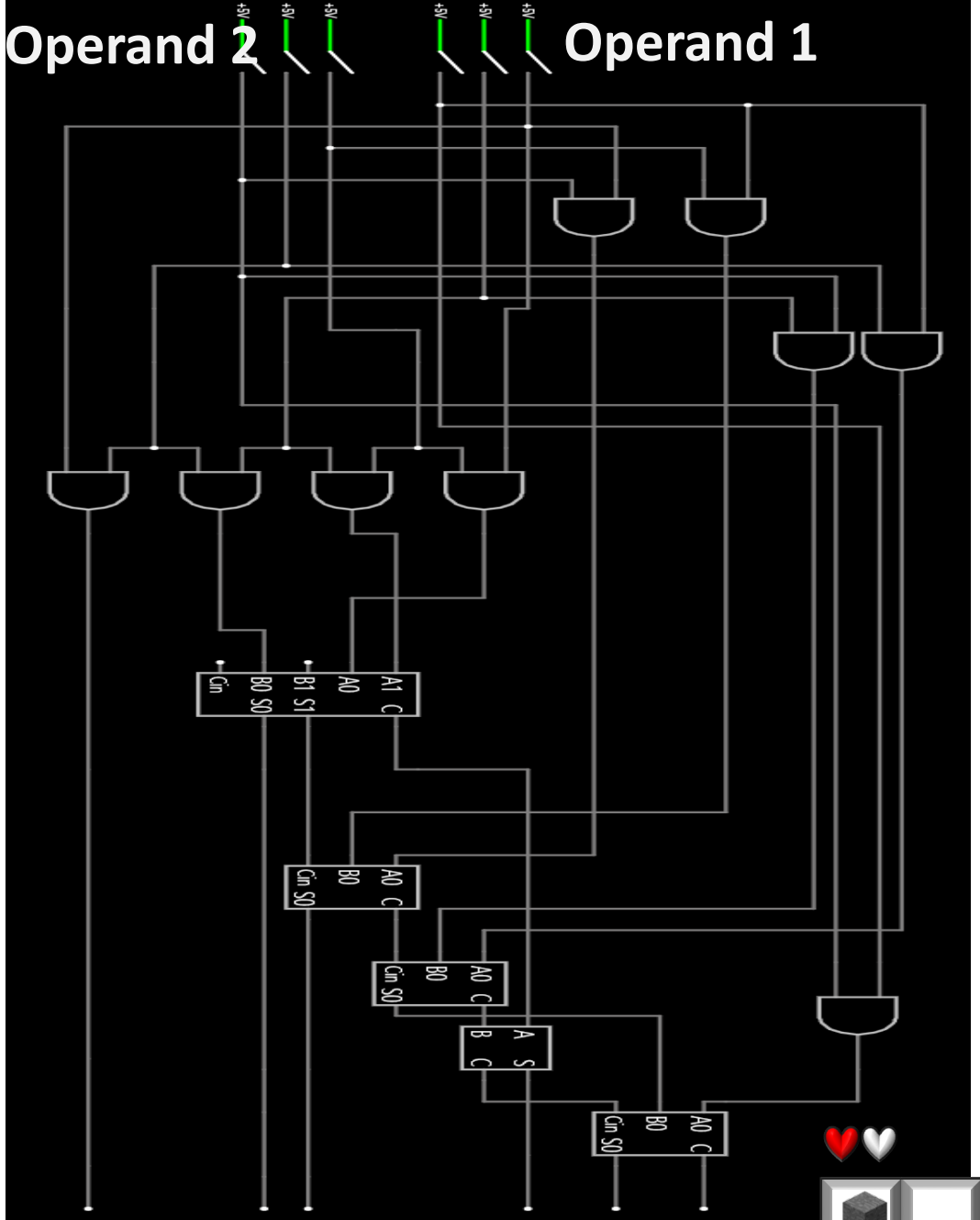
2bit X 2bit Binary Multiplier

INPUT : 2bits 값 2개 (x, y)
x, y : 0 ~ 3의 정수

OUTPUT : 4 bits 값

$2^3 = 8 < 3X3 = 9 < 2^4 = 16$

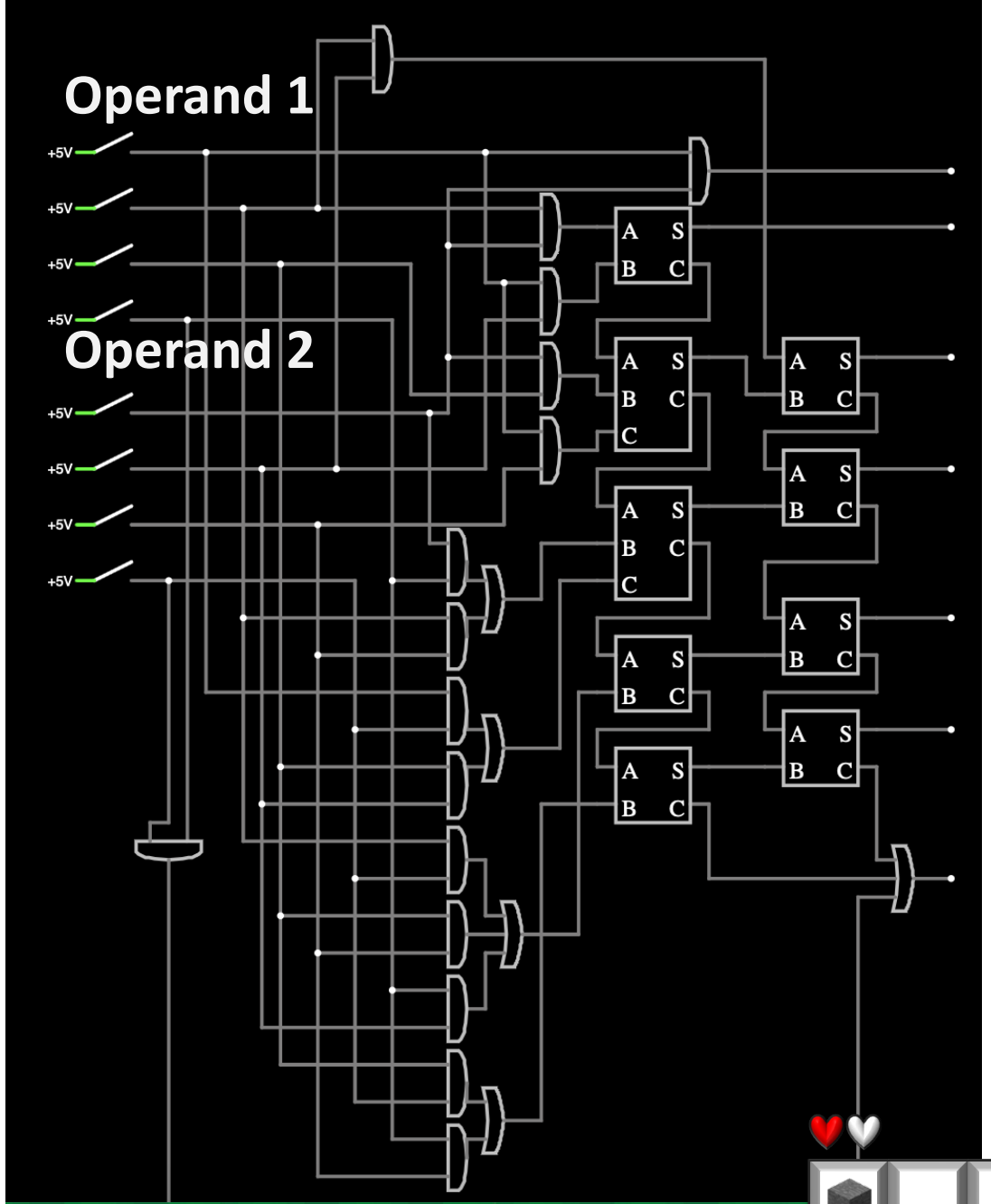

2
BCD Multiplier



3bit X 3bit BCD multiplier

Operand1 X Operand2에 대한
연산 수행

3Bits X 3Bits는 최대 6Bits까지의
결과 발생



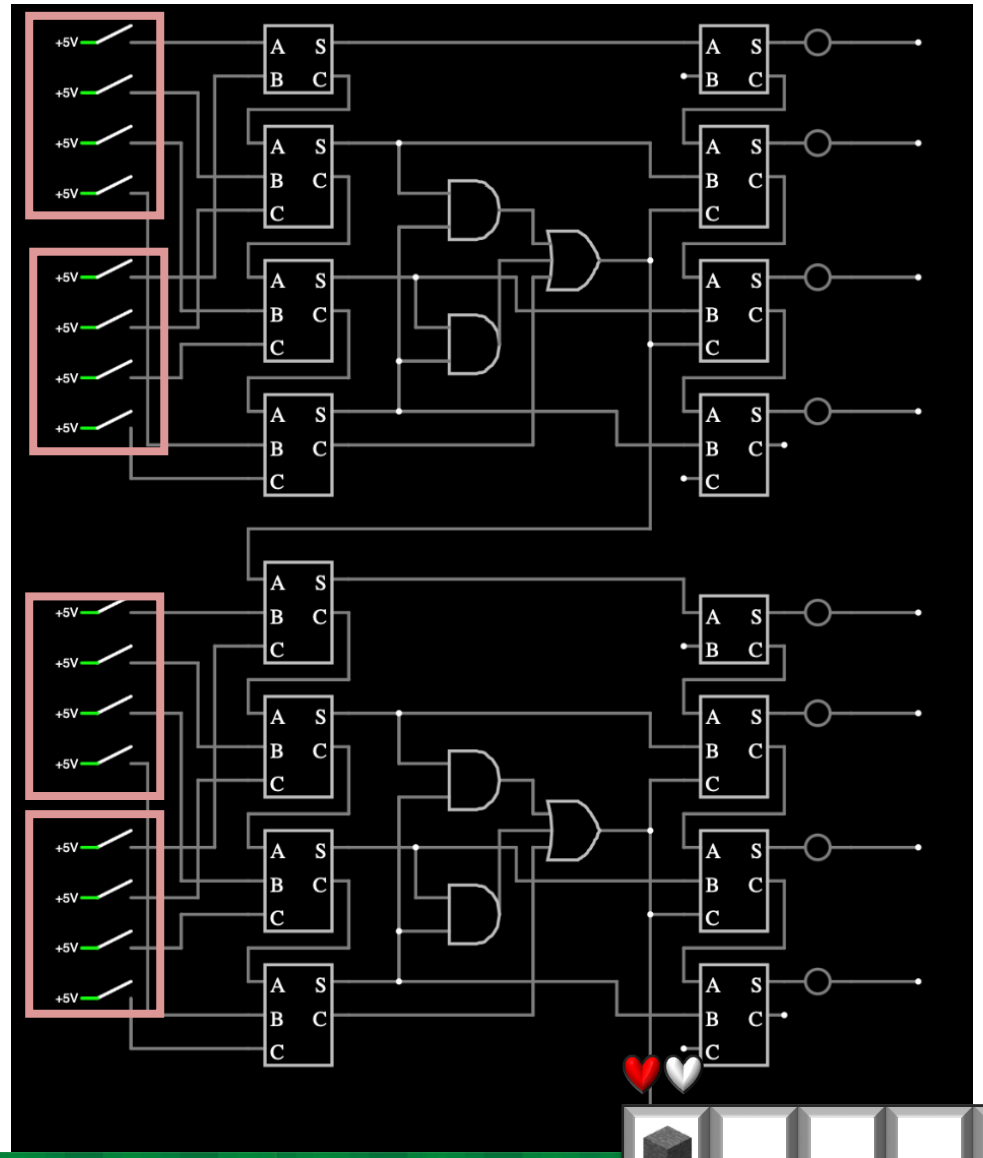
4bit X 4bit BCD multiplier

Operand1 X Operand2에 대한 연산 수행

4bit X 4bit는 최대 7bit 까지의 결과 발생

1.곱하는 비트수가 높아질수록 회로가 매우 복잡해짐
2.결과값이 2진수이기 때문에 bcd코드로 바꾸어주는 변환기(Converter)가 필요함

곱셈기



8 bit X 8bit case

ex) 11 X 11 = 1 + 10 + 10 + 100

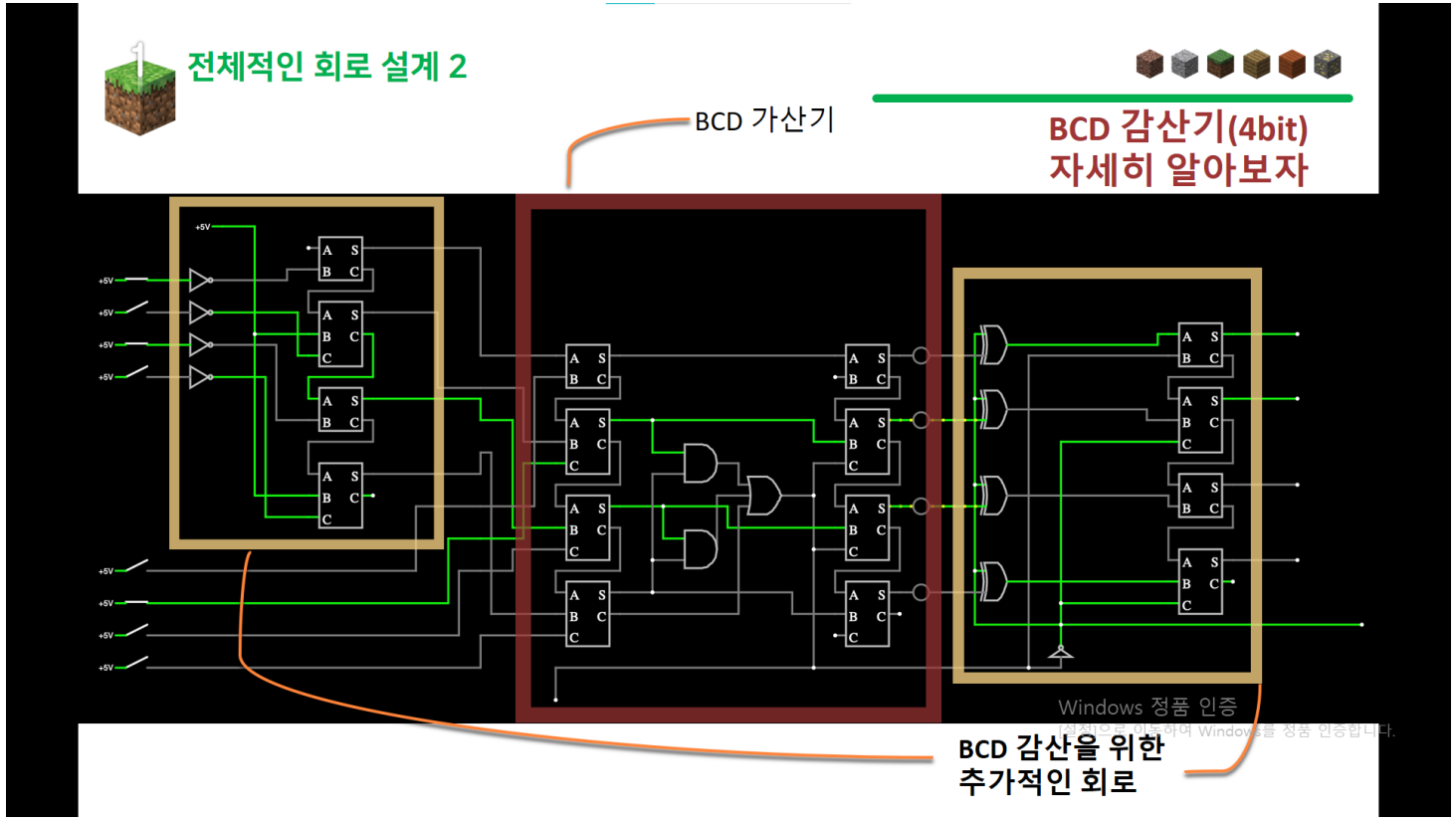
Operand1 X Operand2 (일의자리) X (일의자리)
Operand1' X Operand2 (십의자리) X (일의자리)
Operand1 X Operand2' (일의자리) X (십의자리)
Operand1' X Operand2' (십의자리) X (십의자리)

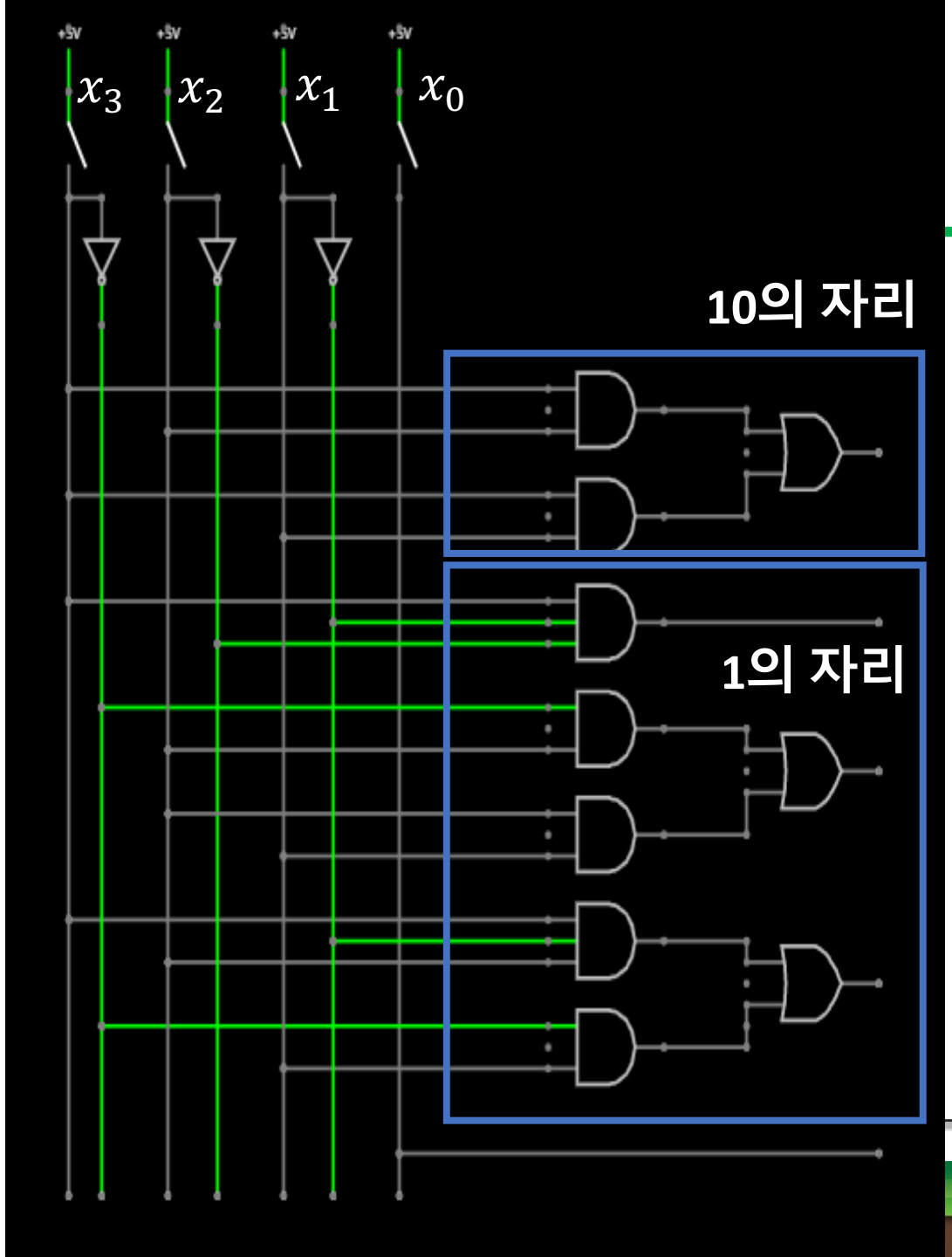
즉 4bit X 4bit BCD multiplier X4

+

BCD 가산기(자릿수 덧셈)

5/12일 BCD 감가산기 발표 中





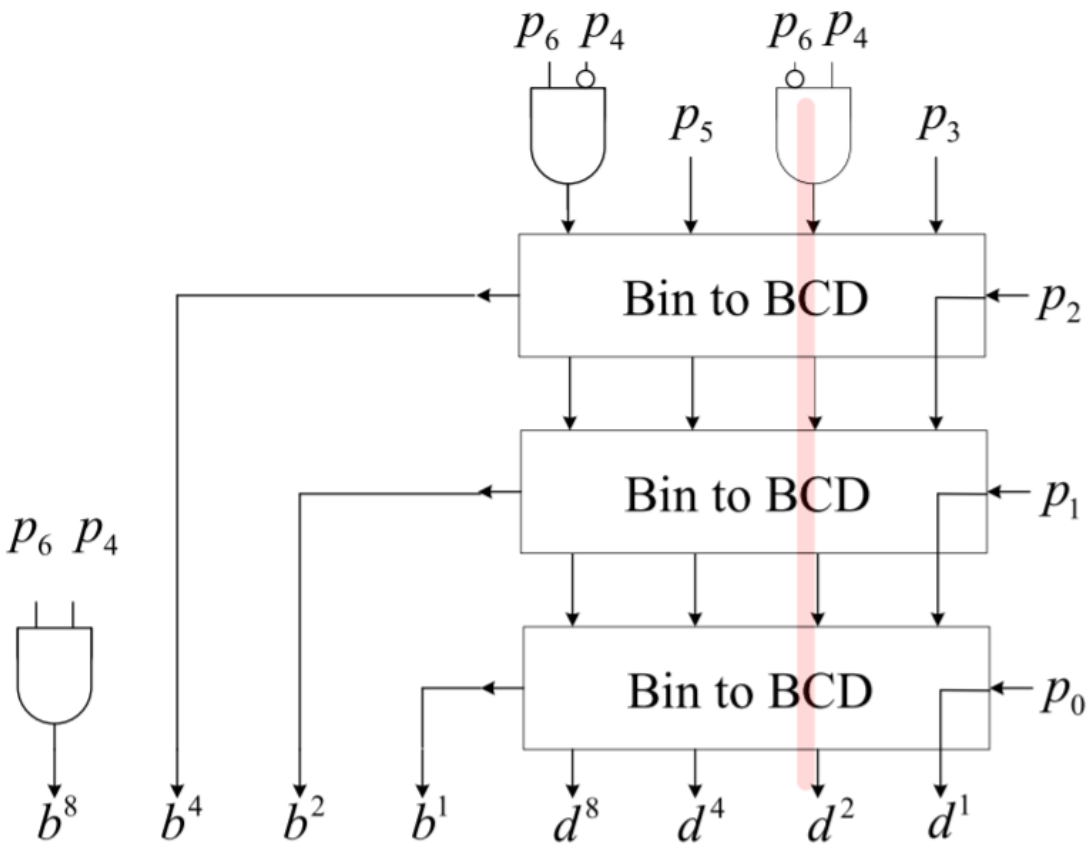
Binary to BCD Converter

어떻게 2진수를 BCD로 바꿀 수 있을까?

-> Binary to BCD Converter

옆의 회로는 4bit BCD Converter

입력값이 커질수록 회로는 더 복잡해짐



7bit Binary to BCD Converter

Converter 내부에 Converter가 3개 포함

(한 자리 수) X (한 자리 수) ≤ 81 < 128 = 2^7

8bit X 8bit 곱셈기

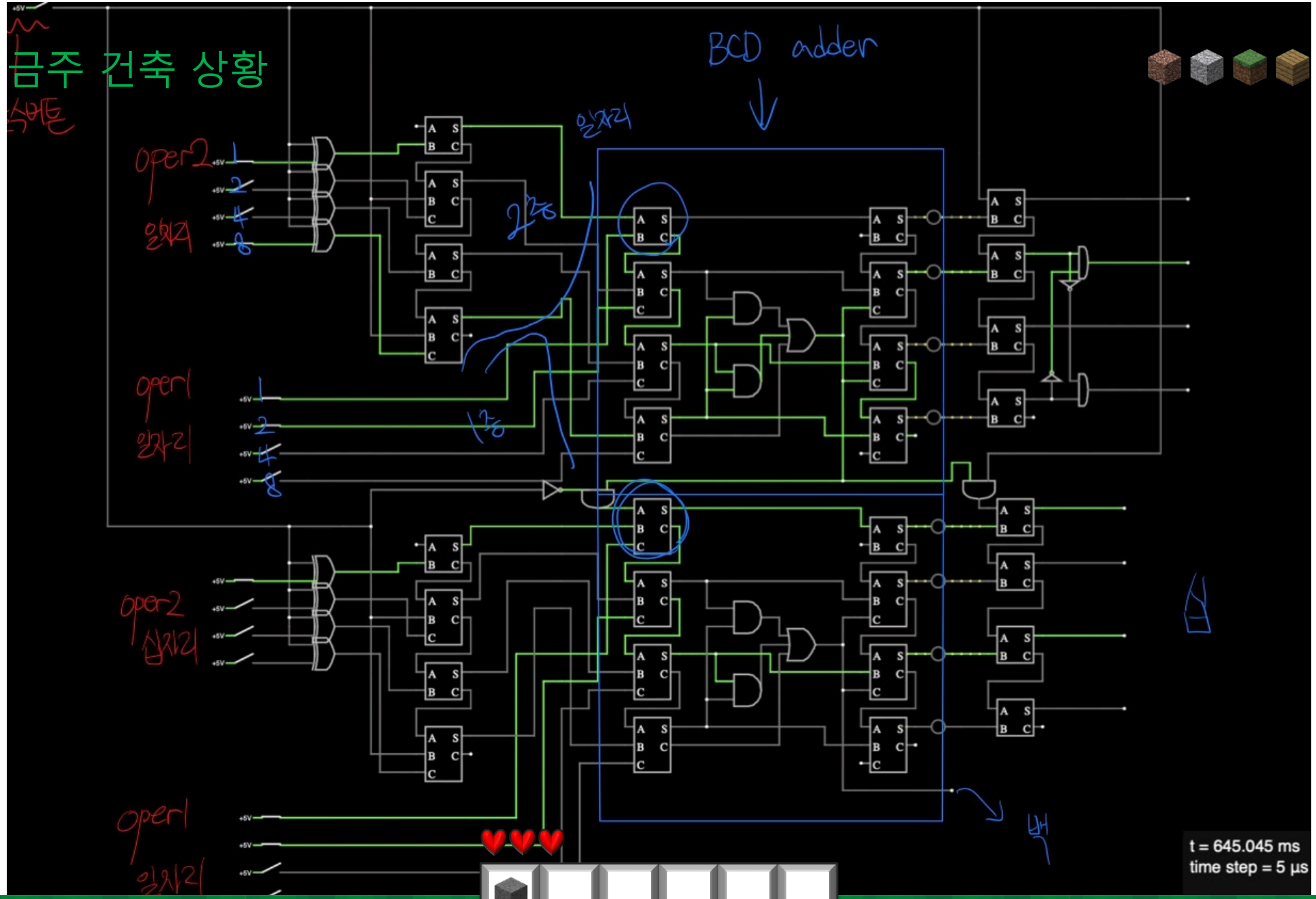
4bit X 4bit bcd 곱셈기 X 4 +
7bit BCD Converter X 4 +
BCD 가산기들(자릿수 덧셈)

= 구현하기 현실적으로 어려움...

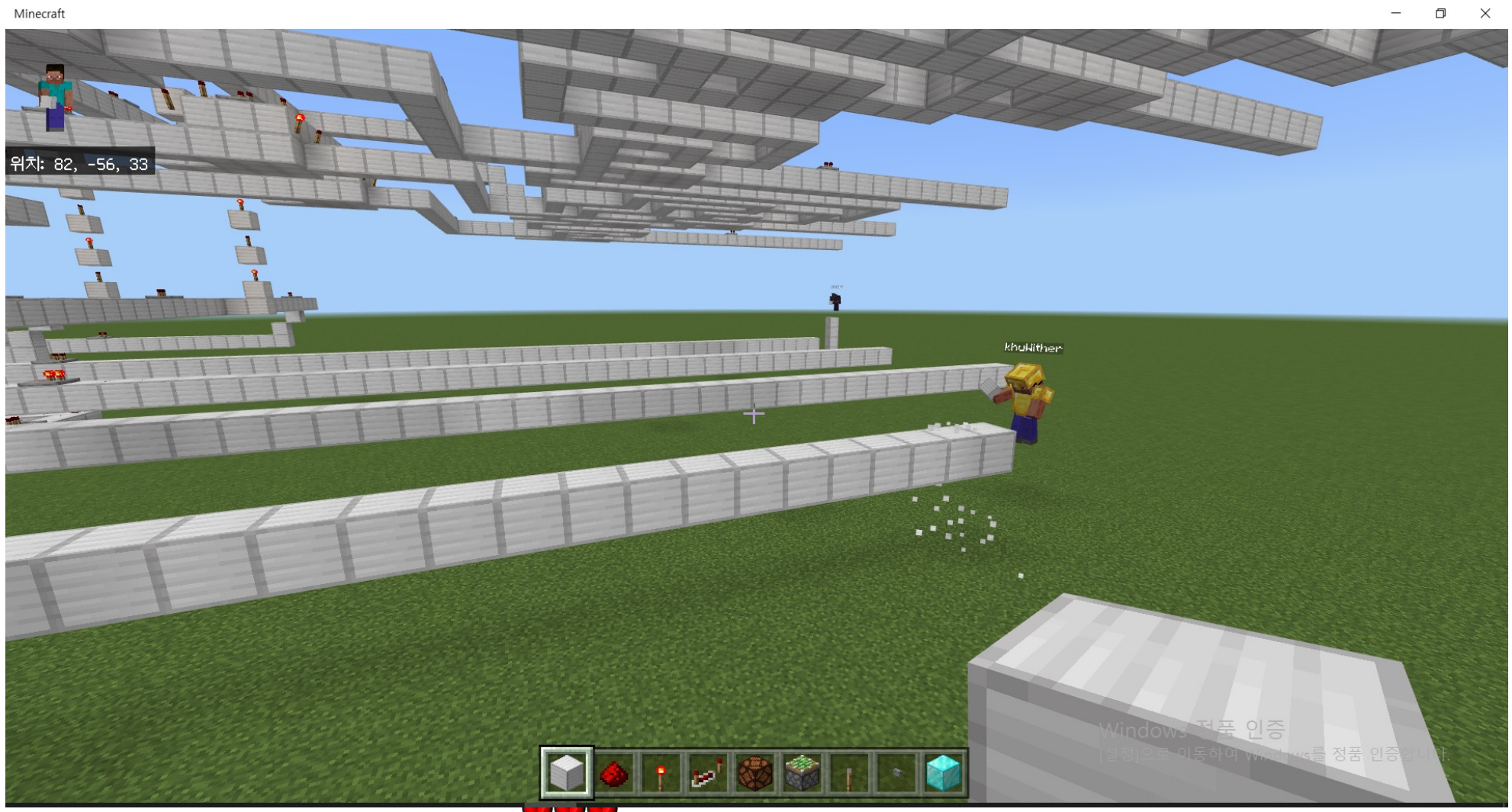
2

이번주 건축실황

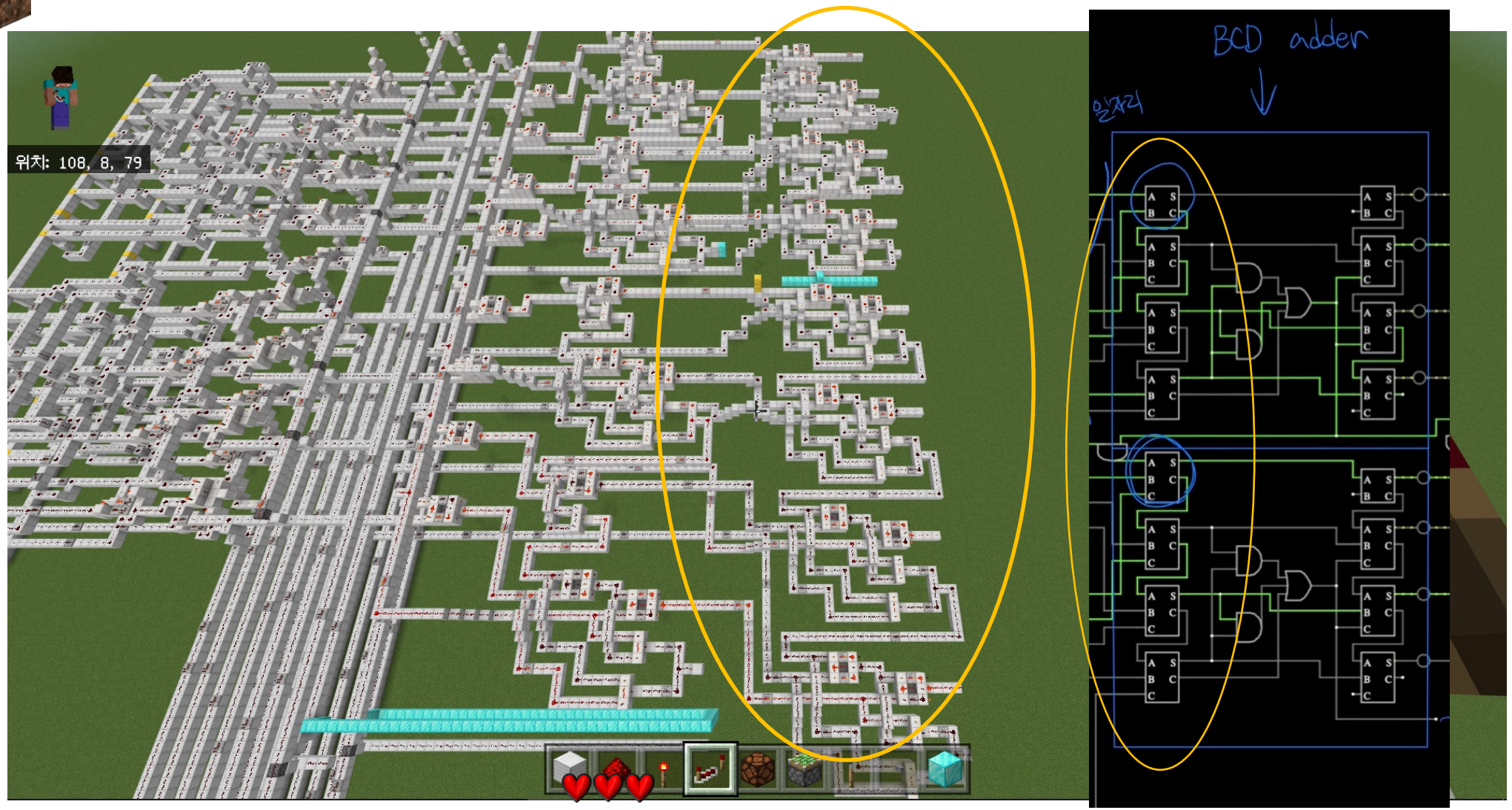
금주 건축 상황



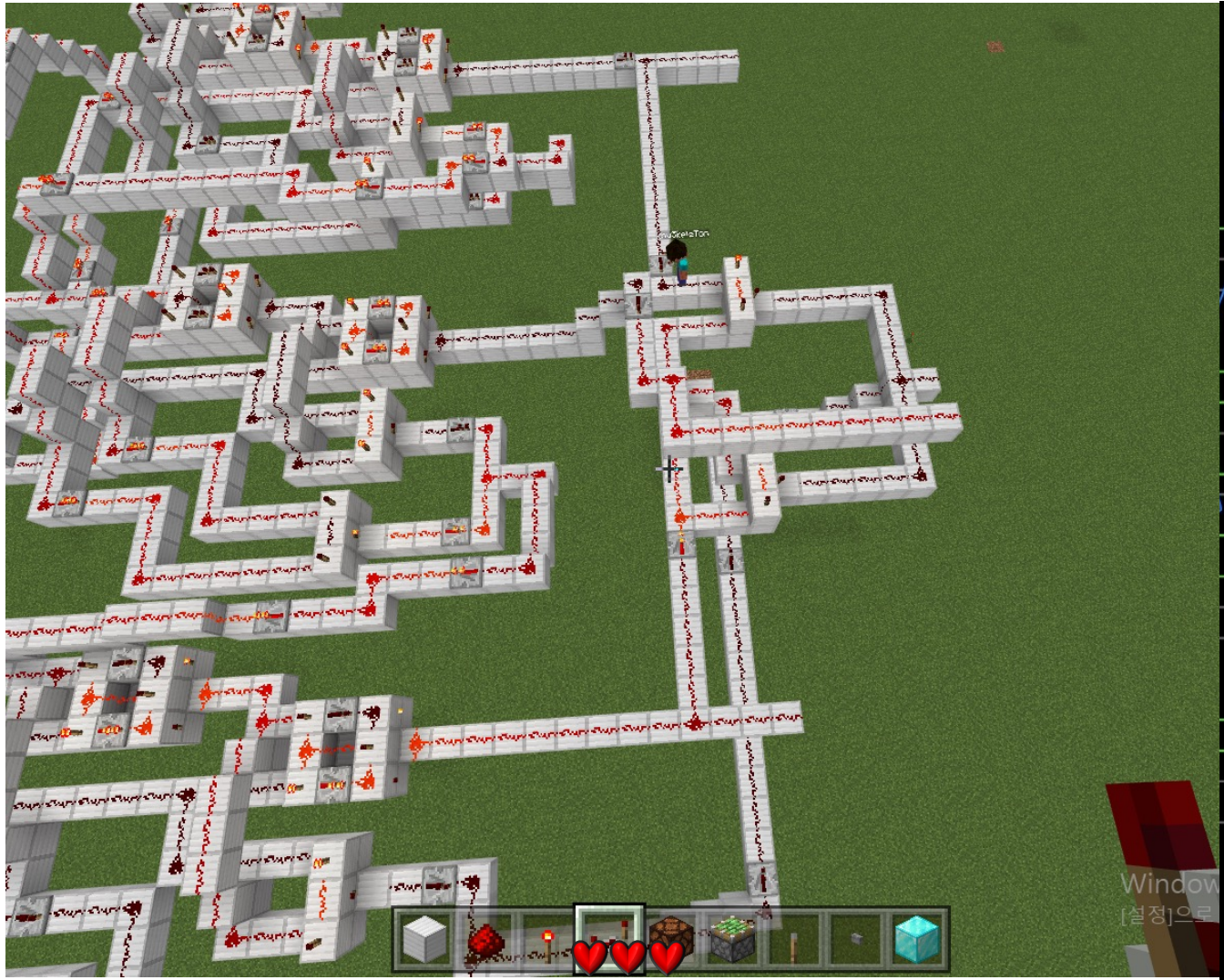
금주 건축 상황

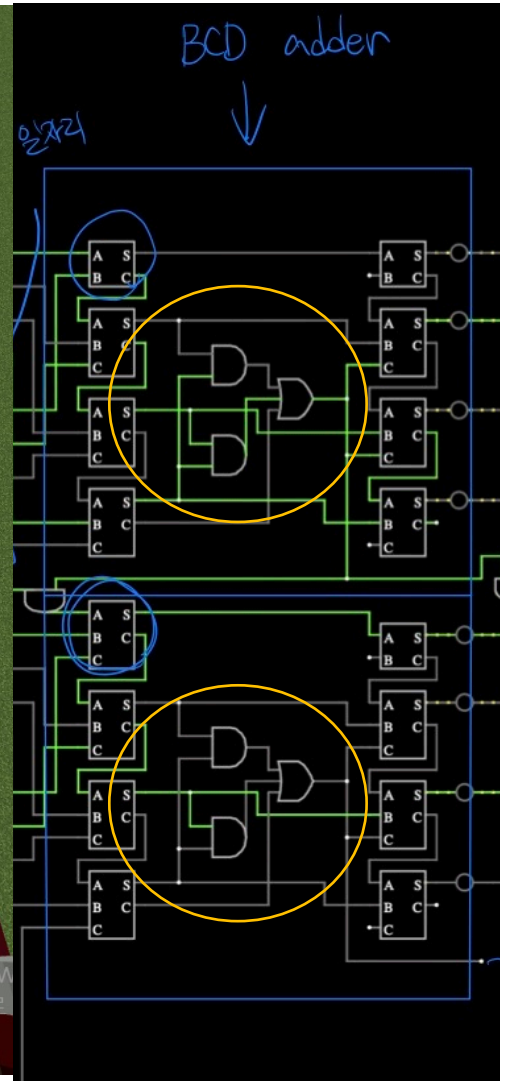


금주 건축 상황

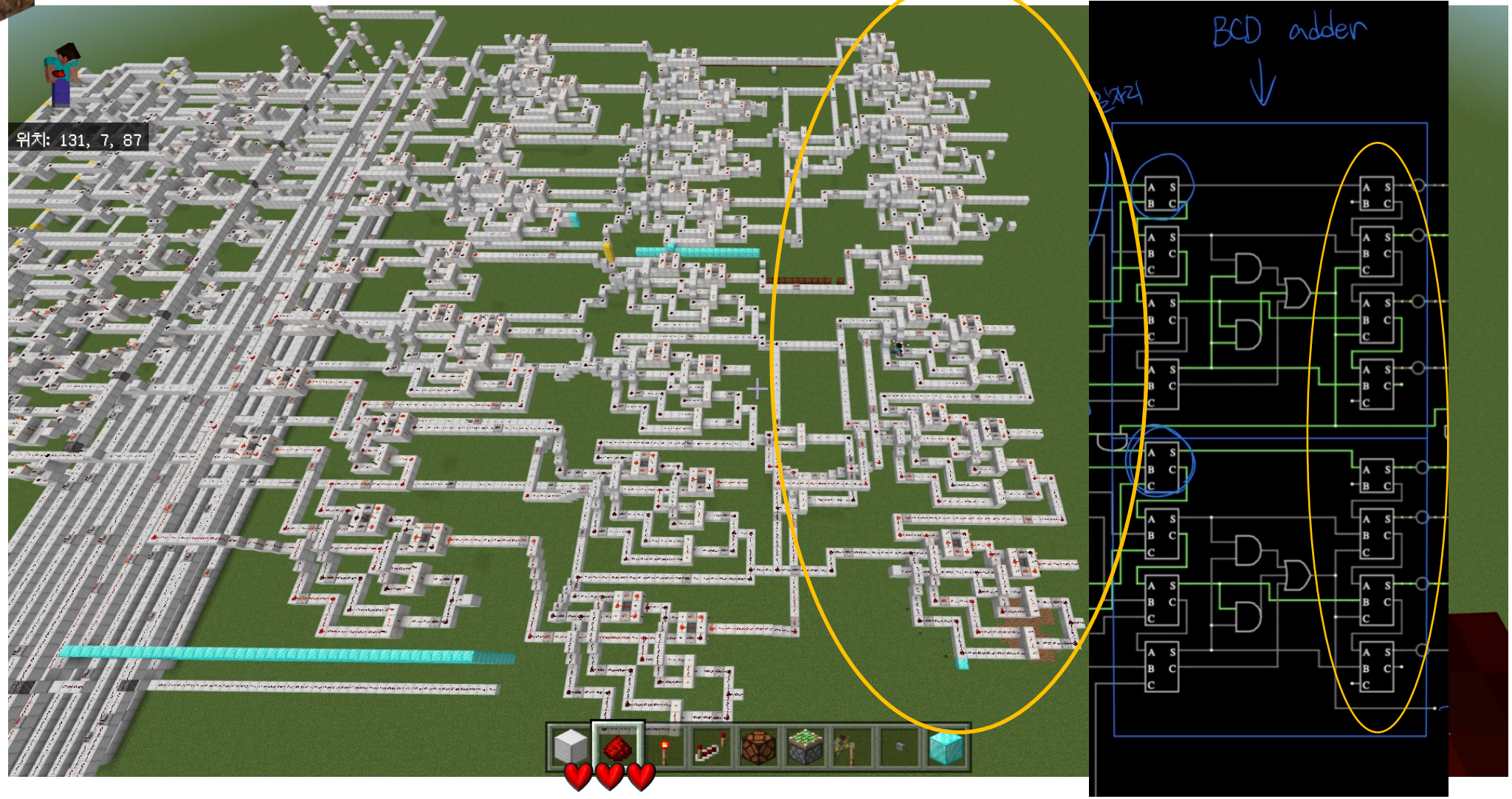


금주 건축 상황

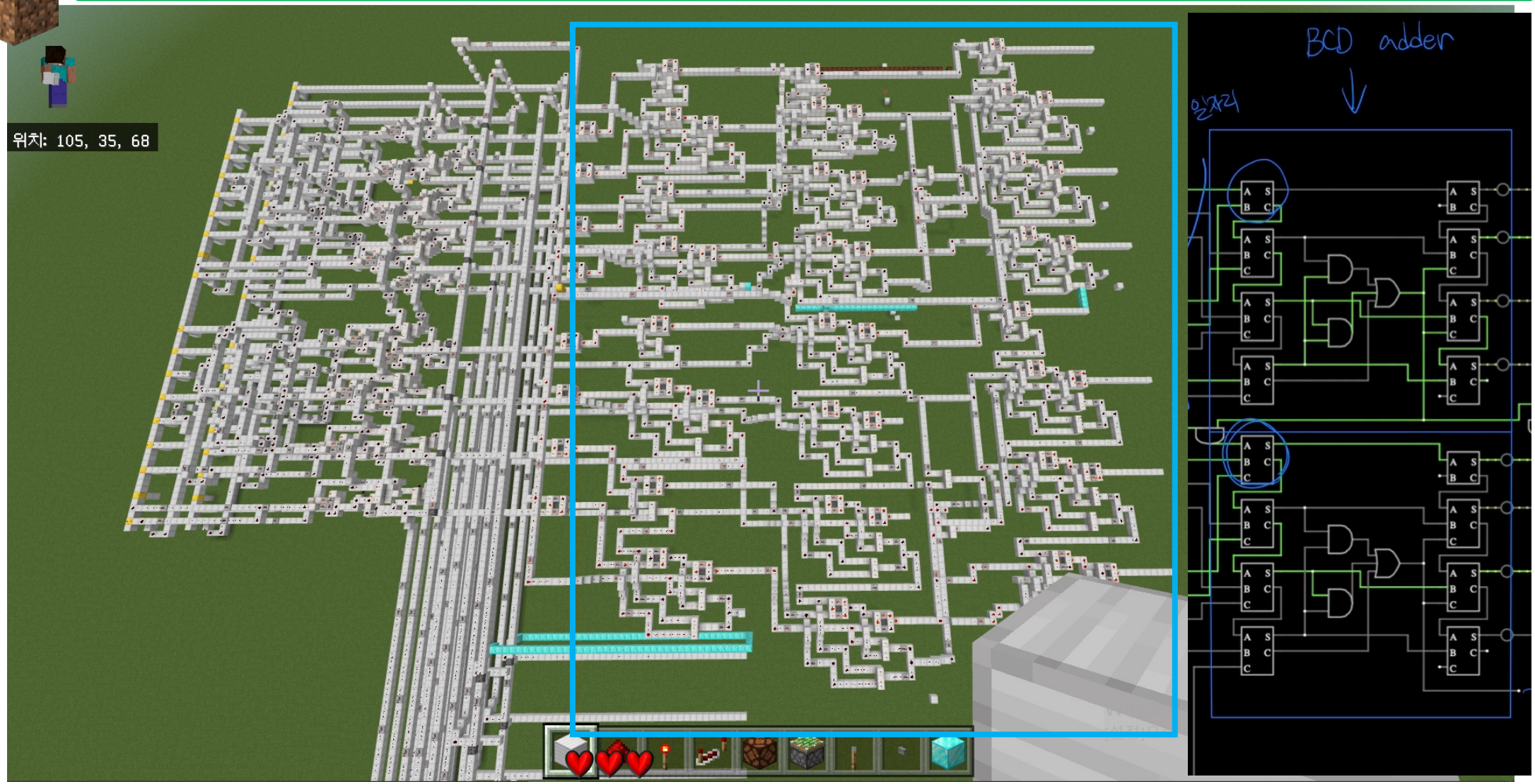




3 금주 건축 상황



3 금주 건축 상황



금주 건축 상황

다음주 목표

“7 Segment”


khuEnderman
치: 92, -26, 19
Thank you
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !

MINECRAFT

CAPSTONE DEGIGN

7-segment display

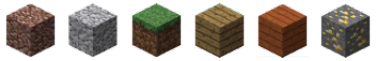
현재까지의 진행상황 및 향후 계획

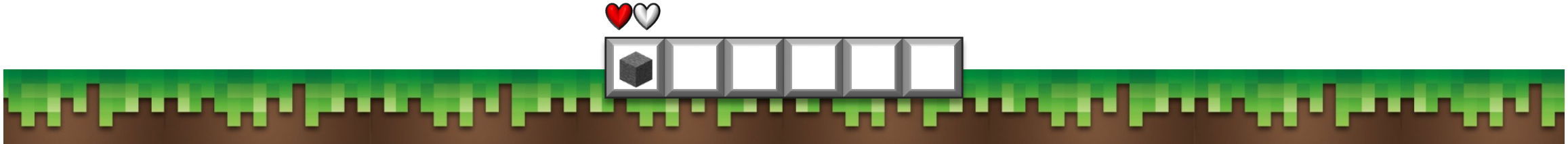
활동 사진

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


0
목차

목차





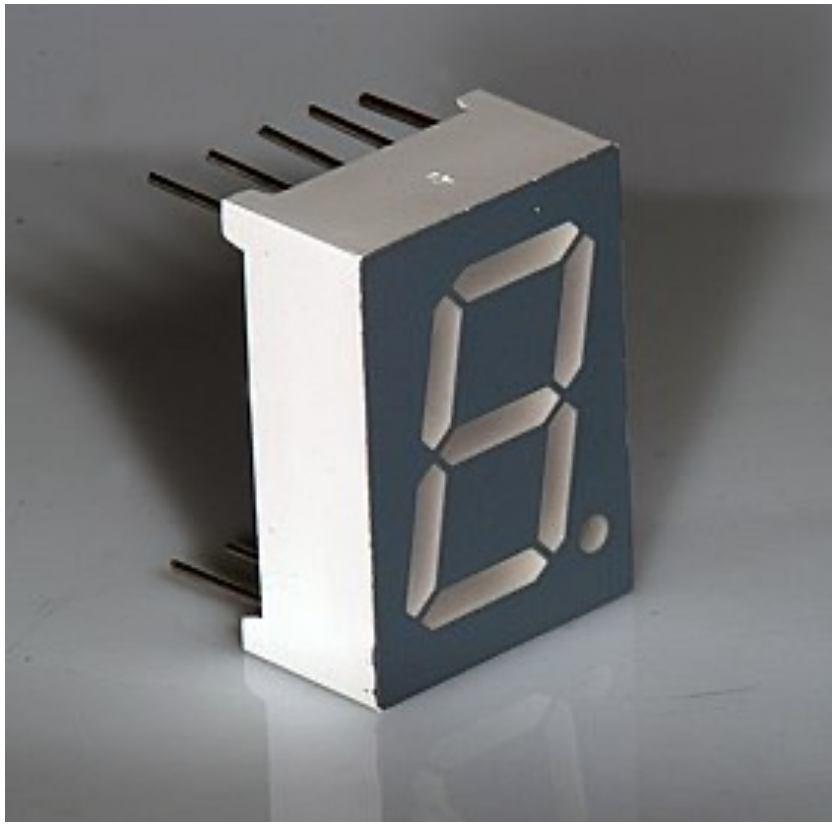
1

7-segment display

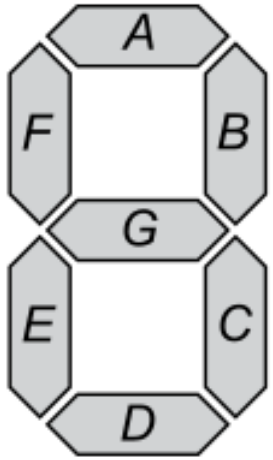
7-segment display

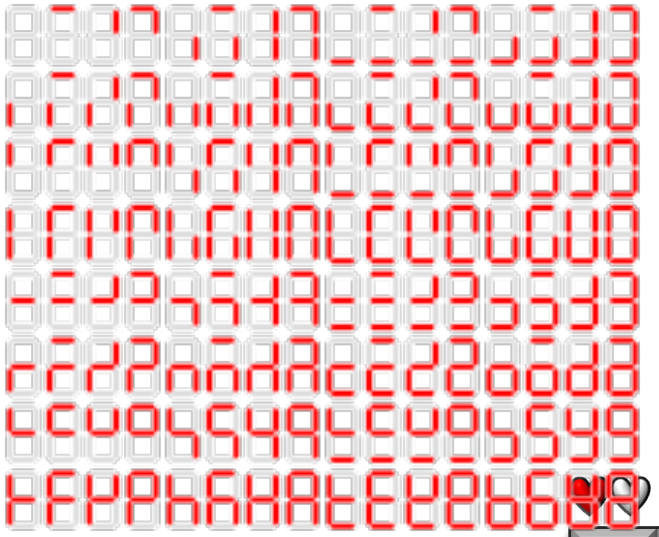
- 7개의 발광소자를 선택하여 조합하여 표시 장치를 구성

- 0 ~ 9까지의 숫자의 디스플레이 표시 가능



7-segment display

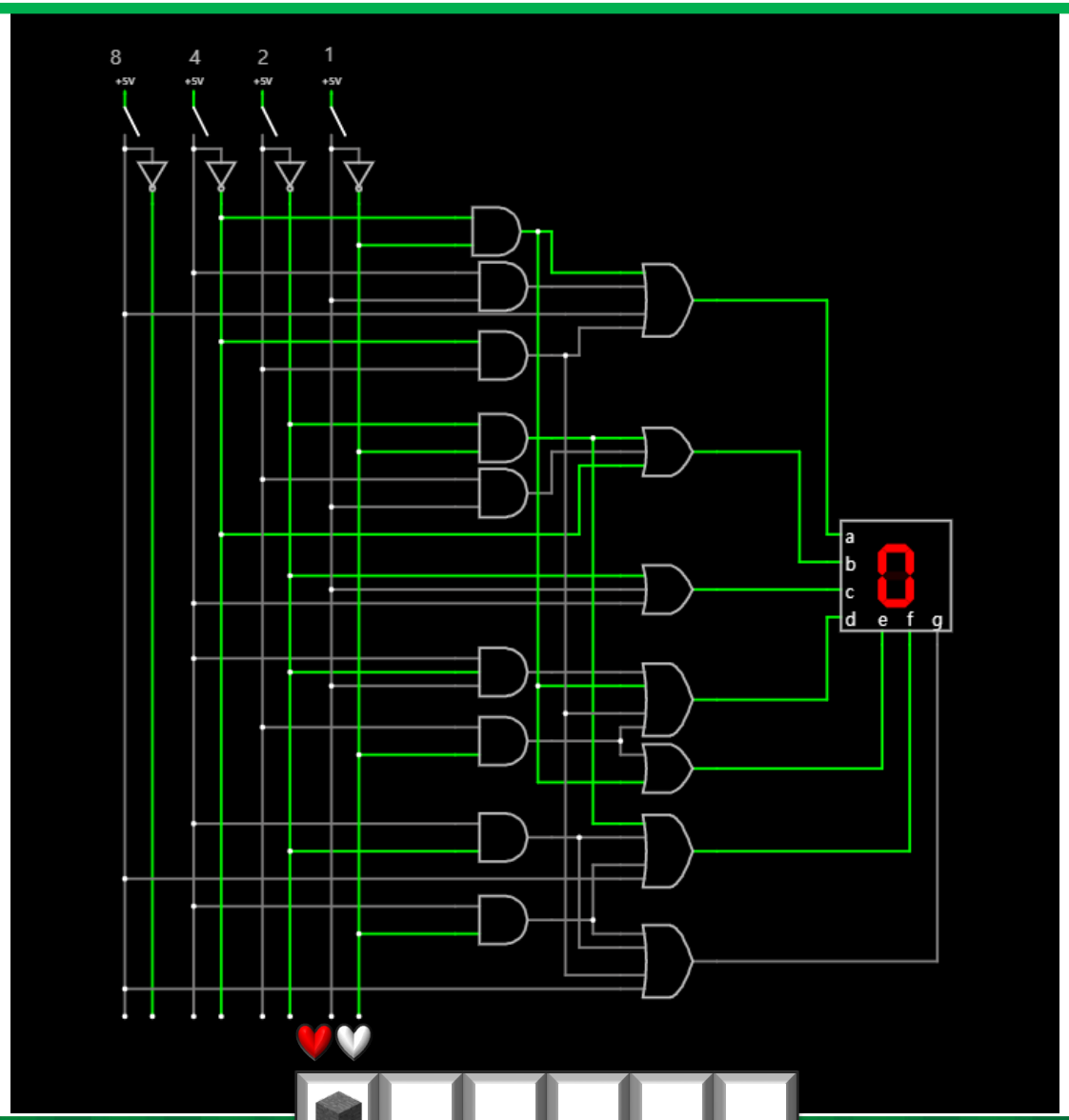




Digit	Display	p	a	b	c	d	e	f	g
0	0		on	on	on	on	on	on	
1	1			on	on				
2	2		on	on		on	on		on
3	3		on	on	on	on			on
4	4			on	on			on	on
5	5		on		on	on		on	on
6	6		on		on	on	on	on	on
7	7		on	on	on				
8	8		on	on	on	on	on	on	on
9	9		on	on	on	on		on	on

7-segment display

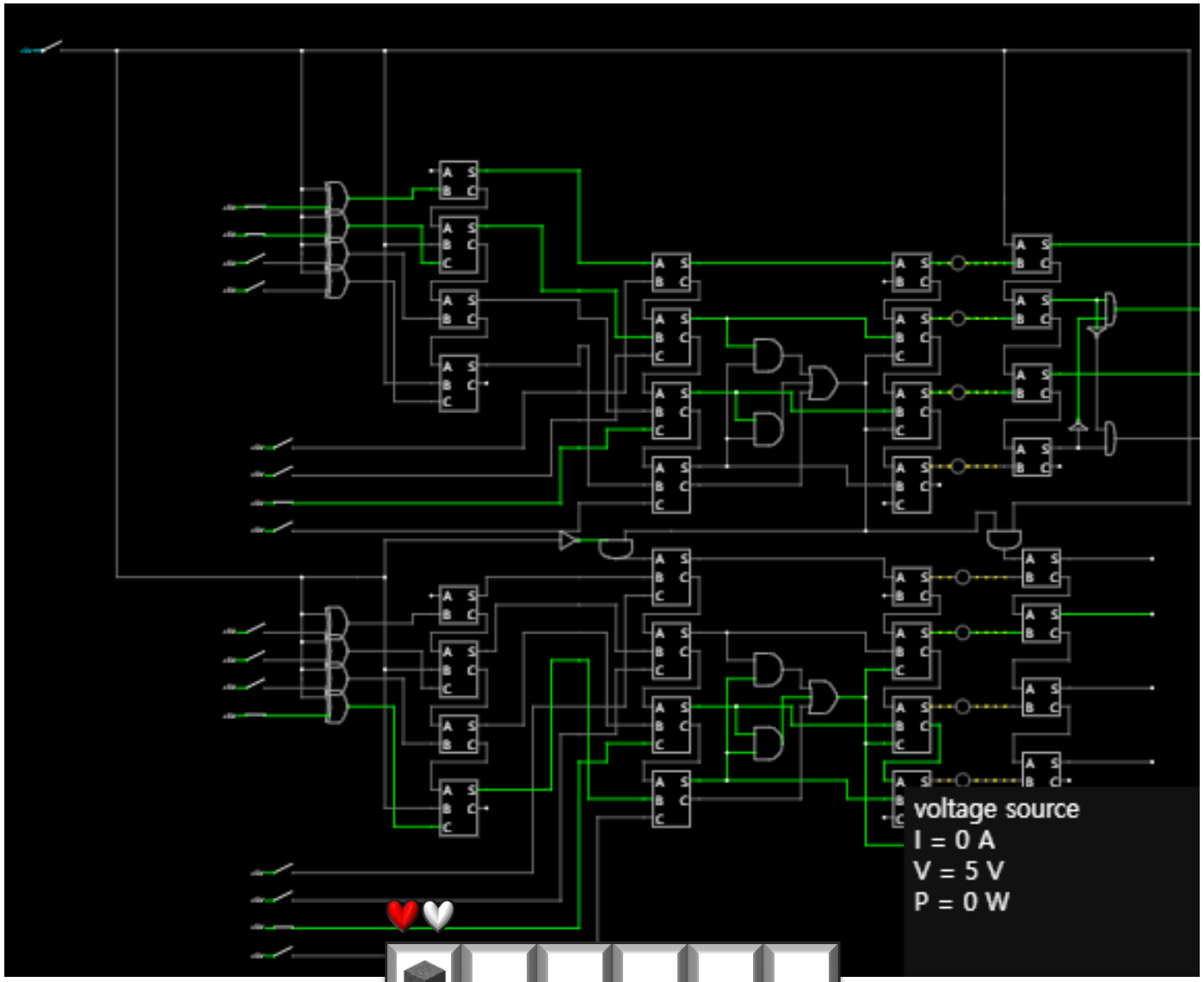
7-segment display



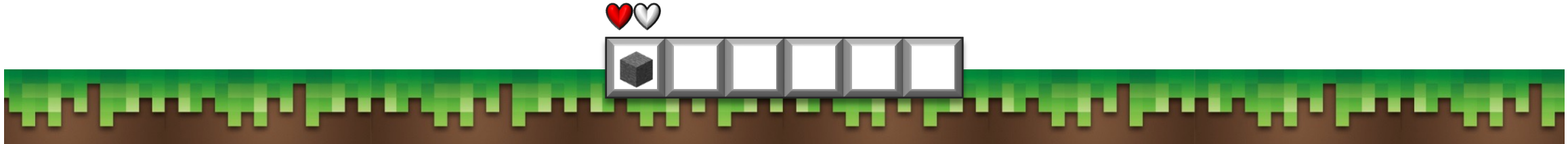
2

현재까지의 진행상황 및 향후 계획

현재까지의 진행상황 및 향후계획

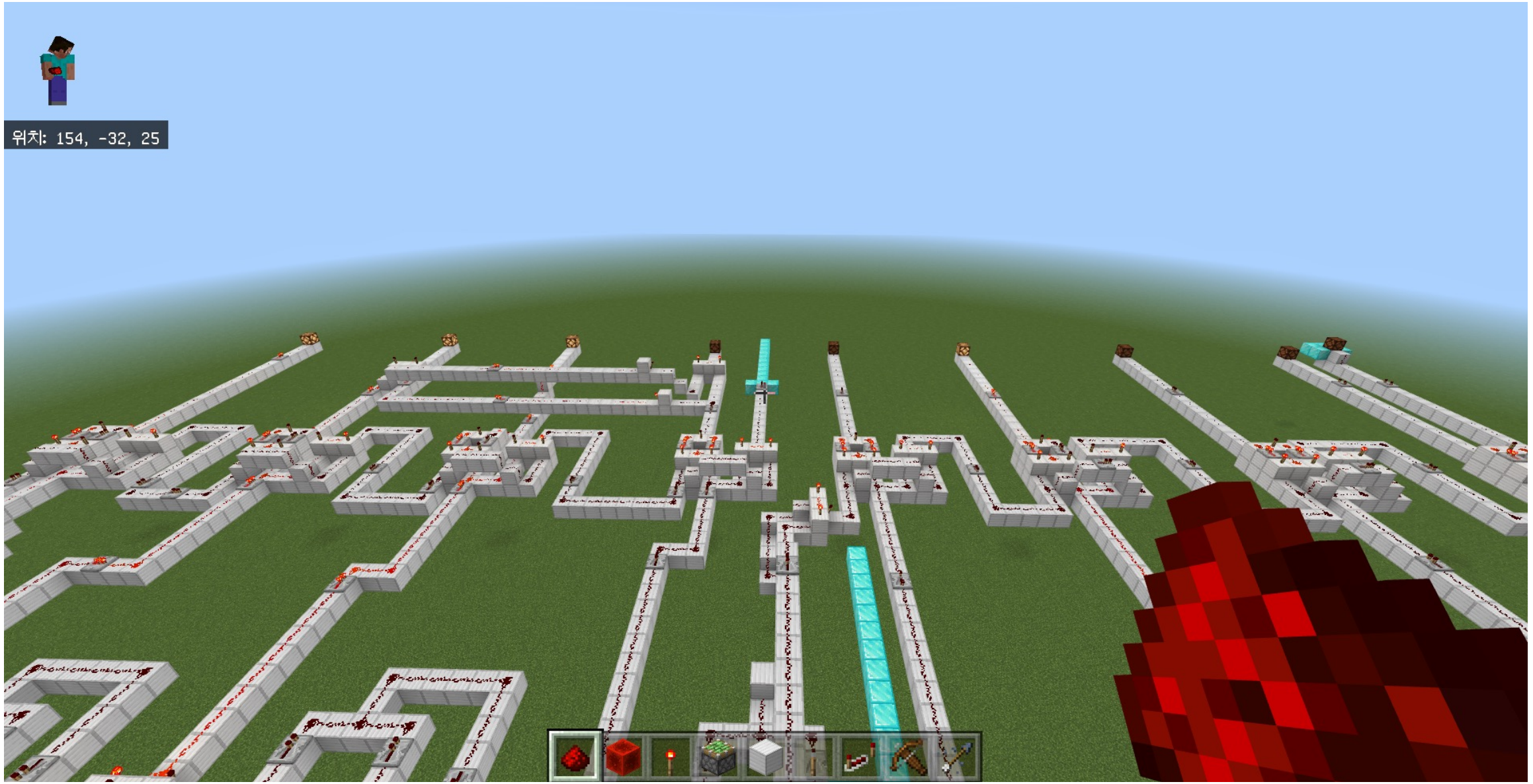


향후계획?



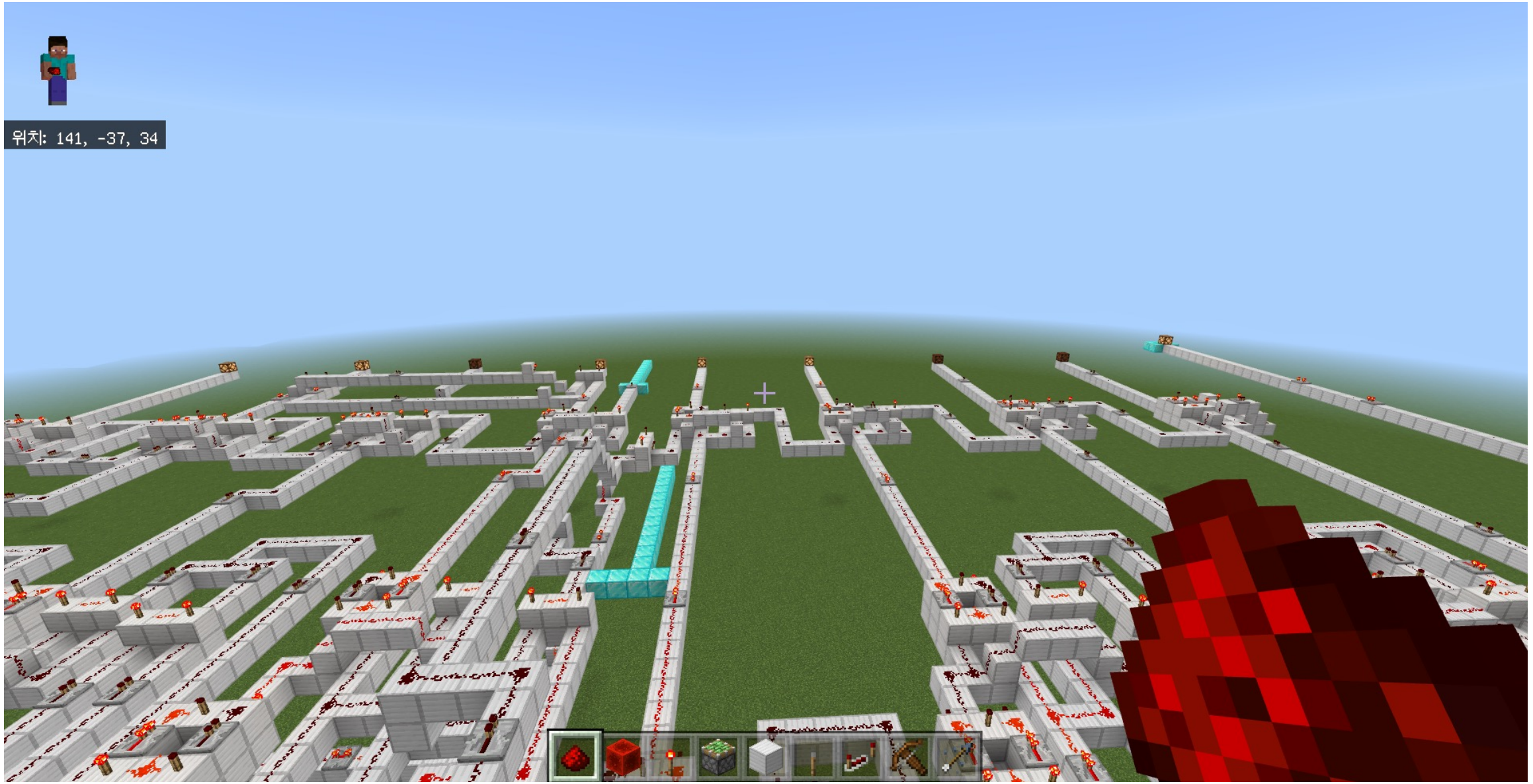
3

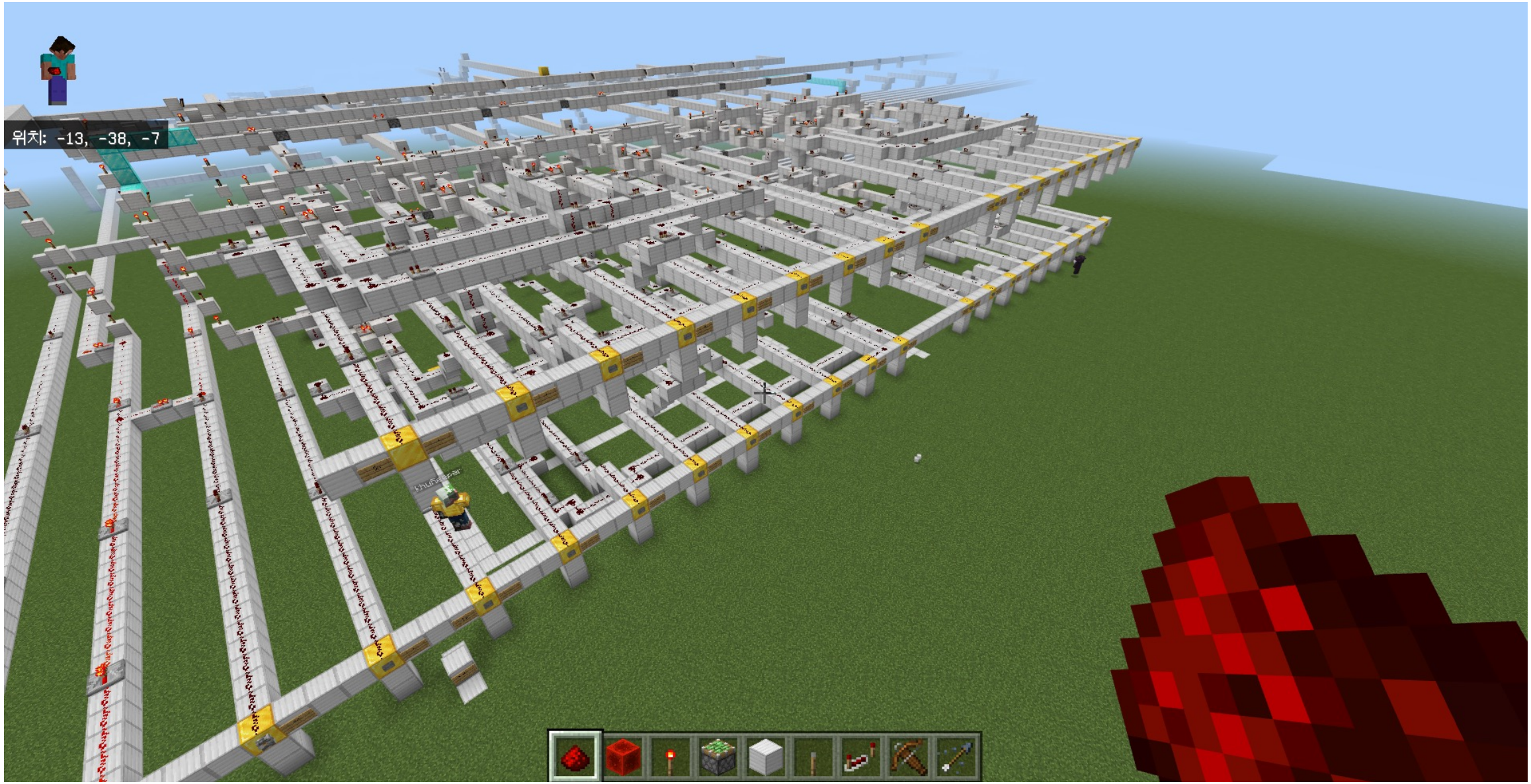
활동사진


위치: 154, -32, 25


위치: 146, -48, 65
khuWither
khuCreeper


위치: 140, -47, 79
khuPigLin
khuWither


위치: 141, -37, 34

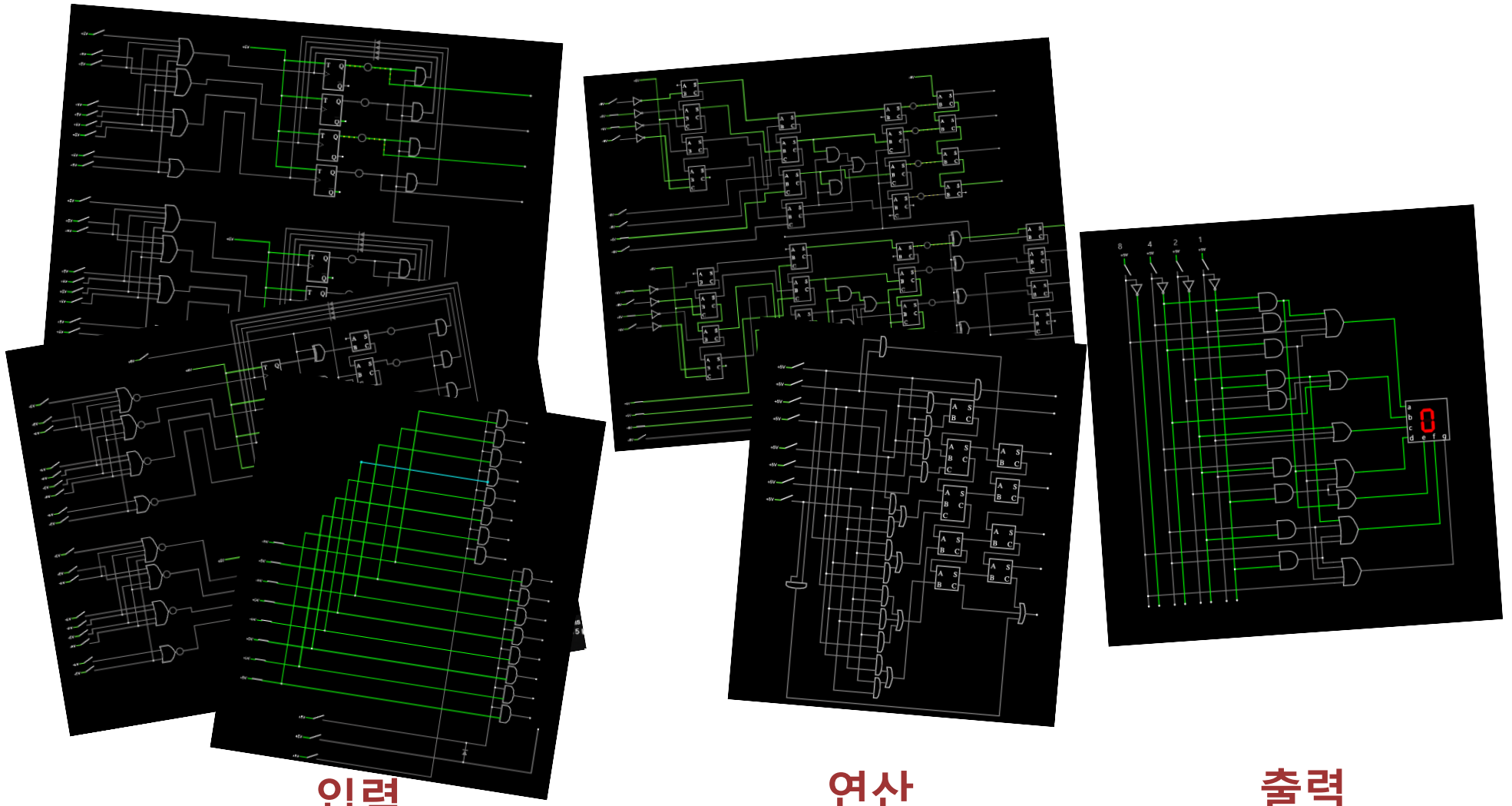

위치: -13, -38, -7

Thank you

ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


마 인 크 래 프 트 로 계 산 기 를 만 들 어 보 자 !
MINECRAFT
CAPSTONE DEGIGN
진짜 CPU에 대해 알아보자
다음주
종료
TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘

지난 4주



입력 연산 출력

1

진짜 CPU에 대해 알아보자

대략적인 구조와 사칙연산 방법

Intel Pentium 4 (2000)

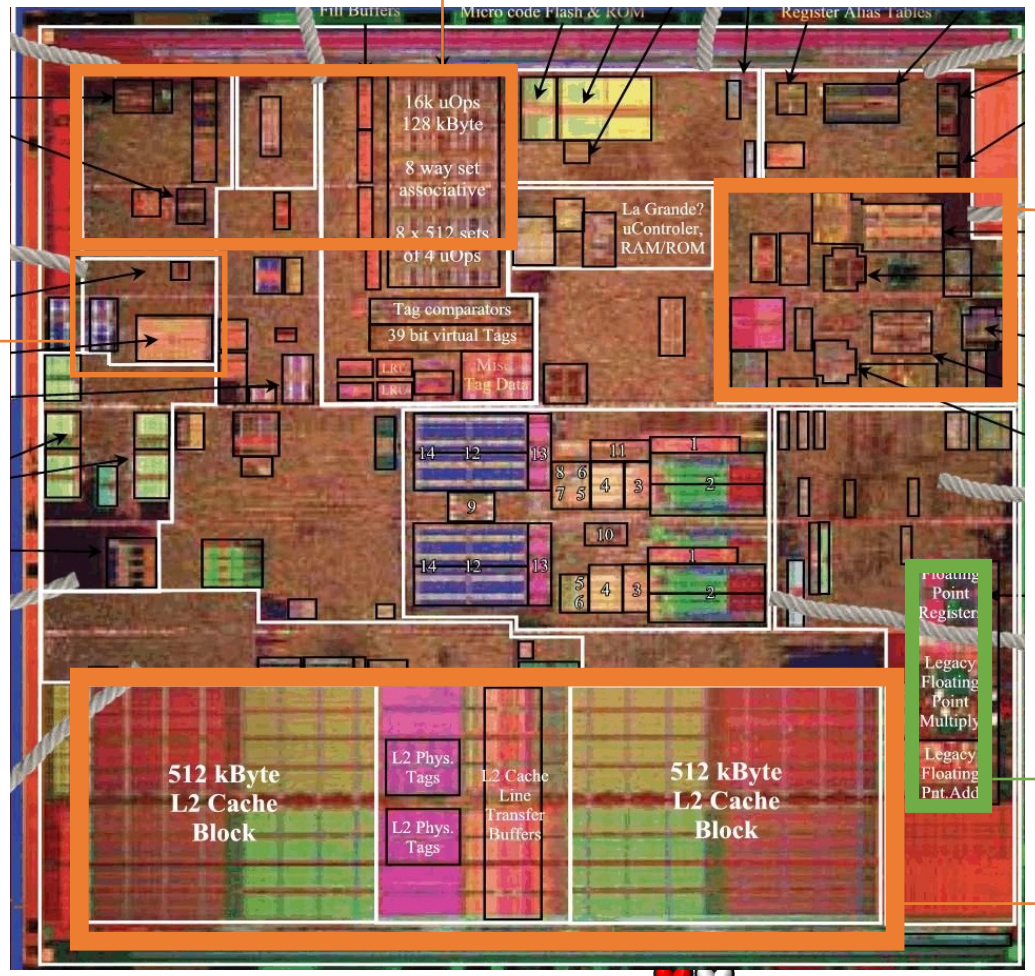
인텔에서 출시한,
싱글 코어 90nm 공정 CPU

AMD Ryzen9 5900X (2020)
12코어(24스레드) 7nm 공정

진짜 CPU에 대해 알아보자

L1 Cache & Cache Table

Intel Pentium 4 Core



Instruction Decoder: 명령어 해독

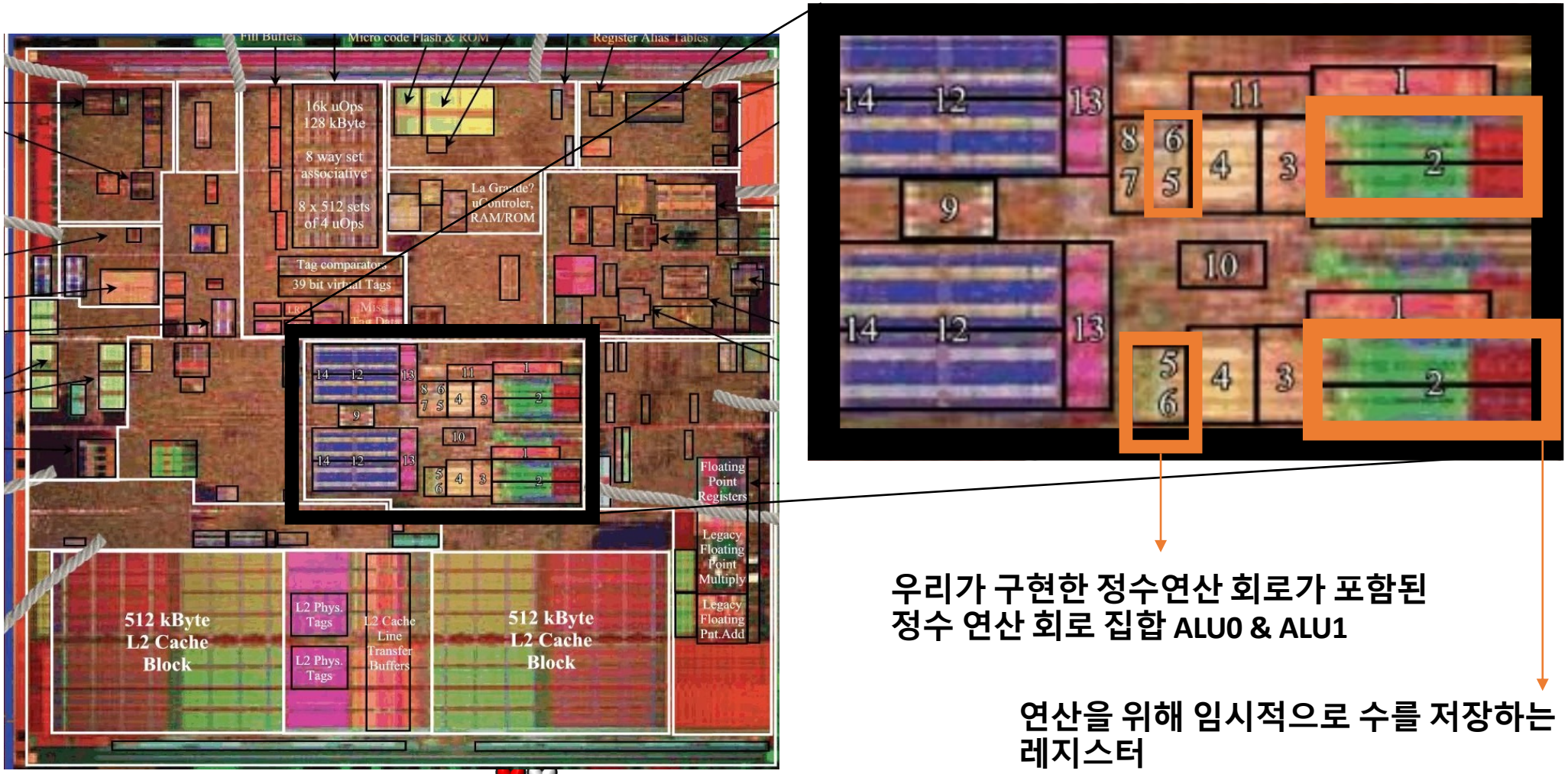
Scheduler 를 위한 회로

FLU: 부동 소수점 연산
& FLU Register

L2 Cache

진짜 CPU에 대해 알아보자

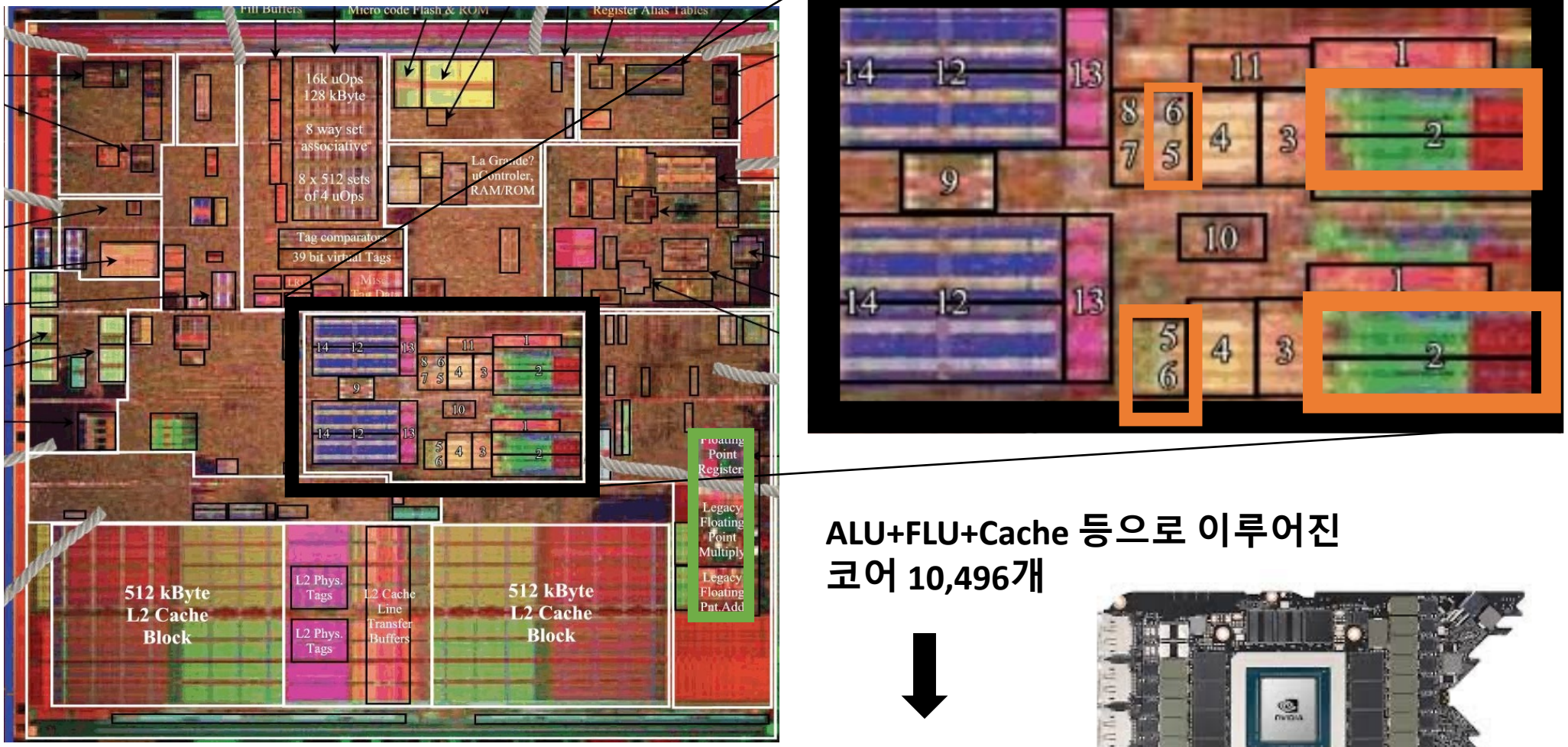
Intel Pentium 4 Core



우리가 구현한 정수연산 회로가 포함된
정수 연산 회로 집합 ALU0 & ALU1

연산을 위해 임시적으로 수를 저장하는
레지스터

진짜 CPU에 대해 알아보자

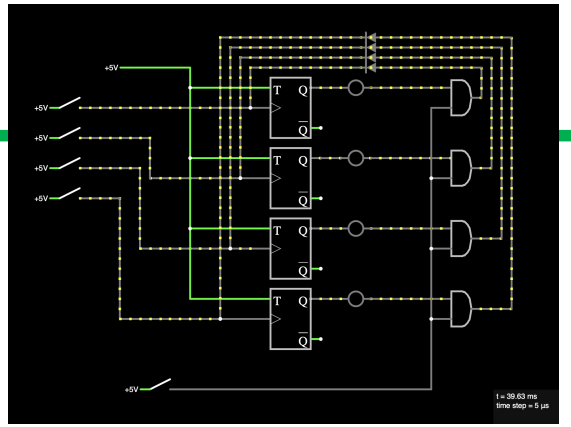


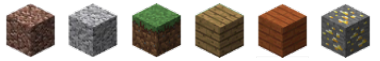
ALU+FLU+Cache 등으로 이루어진 코어 10,496개

딥러닝, 게임 그래픽 등 연산들을 병렬로 빠르게 수행

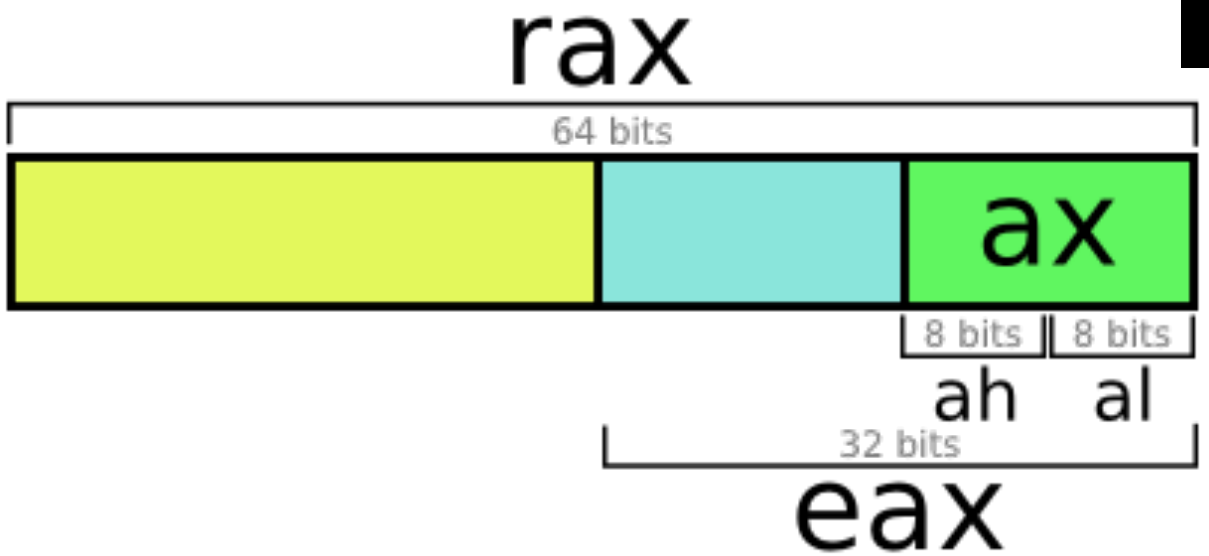
NVIDIA GeForce RTX 3090 (2020.09)

진짜 CPU에 대해 알아보자





1. Register

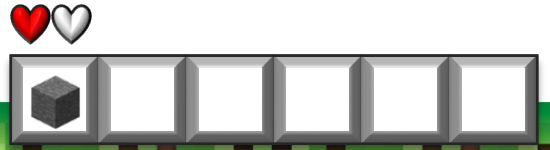


Operand로 입력되는 숫자를 표시하기 위한, Flip-Flop 집합

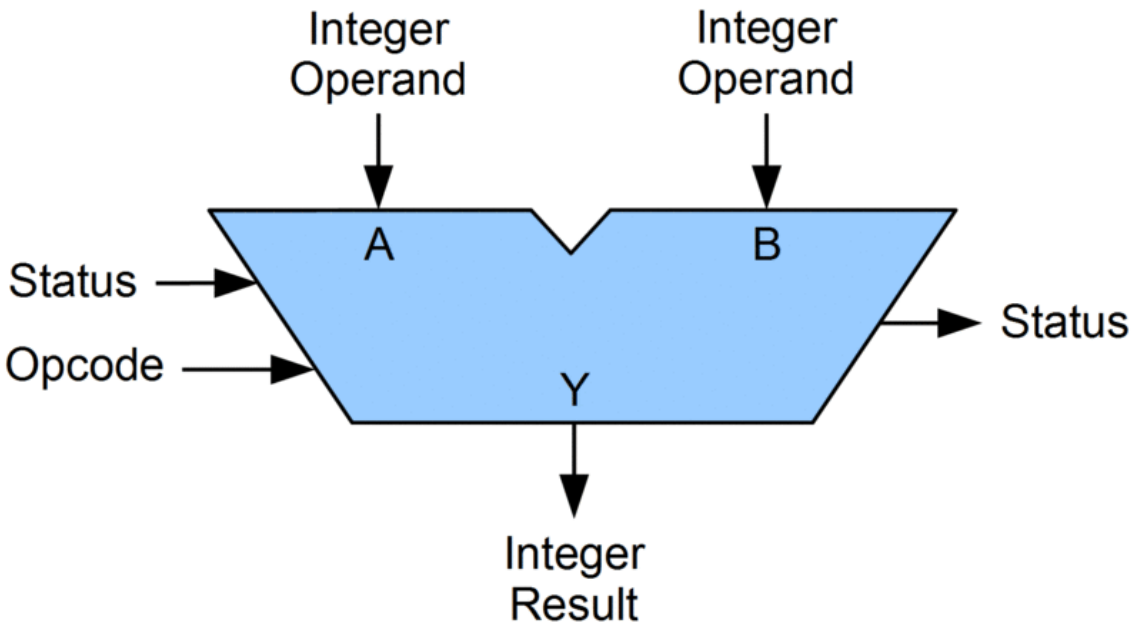
레지스터의 비트의 크기는 컴퓨터의 아키텍처(x86:32bit, x64:64bit)에 따라 결정

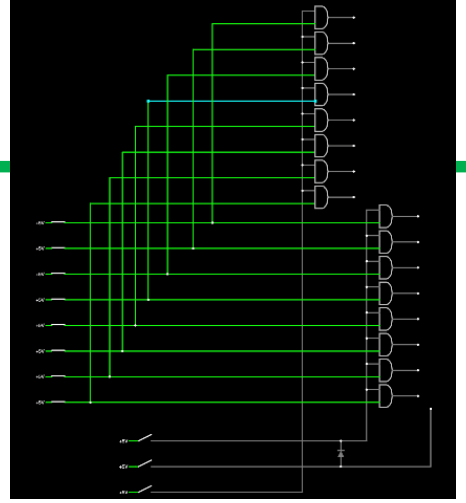
CPU가 사용하는 X64 아키텍처는 정수 표현을 위해 16개의 64bit 레지스터 사용

ARMv8은 31개의 64bit 레지스터 사용



2. ALU





컴퓨터에 포함된 수많은 연산자 중
사용할 연산자를 선택할 수 있는

“멀티 플렉서” + “연산 회로”

덧셈, 뺄셈
논리 연산(AND, OR, XOR ,OR)
시프트 연산 은 필수적으로 포함!

**한 개 이상의 수(혹은 레지스터)를 입력 후, 연산자에 해당하는 “Opcode”를 입력해 주면,
해당 연산에 해당하는 결과를 출력**

OPCODE

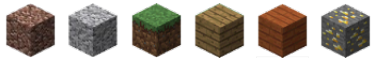
ADD Eb Gb 00	ADD Ev Gv 01	ADD Gb Eb 02	ADD Gv Ev 03	ADD AL Ib 04	ADD eAX Iv 05	PUSH ES 06	POP ES 07	OR Eb Gb 08	OR Ev Gv 09	OR Gb Eb 0A	OR Gv Ev 0B	OR AL Ib 0C	OR eAX Iv 0D	PUSH CS 0E	TWOBYTE 0F
ADC Eb Gb 10	ADC Ev Gv 11	ADC Gb Eb 12	ADC Gv Ev 13	ADC AL Ib 14	ADC eAX Iv 15	PUSH SS 16	POP SS 17	SBB Eb Gb 18	SBB Ev Gv 19	SBB Gb Eb 1A	SBB Gv Ev 1B	SBB AL Ib 1C	SBB eAX Iv 1D	PUSH DS 1E	POP DS 1F
AND Eb Gb 20	AND Ev Gv 21	AND Gb Eb 22	AND Gv Ev 23	AND AL Ib 24	AND eAX Iv 25	ES: 26	DAA 27	SUB Eb Gb 28	SUB Ev Gv 29	SUB Gb Eb 2A	SUB Gv Ev 2B	SUB AL Ib 2C	SUB eAX Iv 2D	CS: 2E	DAS 2F
XOR Eb Gb 30	XOR Ev Gv 31	XOR Gb Eb 32	XOR Gv Ev 33	XOR AL Ib 34	XOR eAX Iv 35	SS: 36	AAA 37	CMP Eb Gb 38	CMP Ev Gv 39	CMP Gb Eb 3A	CMP Gv Ev 3B	CMP AL Ib	CMP eAX Iv	DS:	AAS
INC eAX 40	INC eCX 41	INC eDX 42	INC eBX 43	INC eSP 44	INC eBP 45	INC eSI 46	INC eDI 47	DEC eAX 48	DEC eCX 49	DEC eDX 4A	DEC eBX 4B				
PUSH eAX 50	PUSH eCX 51	PUSH eDX 52	PUSH eBX 53	PUSH eSP 54	PUSH eBP 55	PUSH eSI 56	PUSH eDI 57	POP eAX 58	POP eCX 59	POP eDX 5A	POP eBX 5B				
PUSHA 60	POPA 61	BOUND Gv Ma 62	ARPL Ew Gw 63	FS: 64	GS: 65	OPSIZE: 66	ADSIZE: 67	PUSH Iv 68	IMUL Gv Ev Iv 69	PUSH Ib 6A	IMUL Gv E 6B				
JO Jb 70	JNO Jb 71	JB Jb 72	JNB Jb 73	JZ Jb 74	JNZ Jb 75	JBE Jb 76	JA Jb 77	JS Jb 78	JNS Jb 79	JP Jb 7A	JN Jb 7B				
ADD Eb Ib 80	ADD Ev Iv 81	SUB Eb Ib 82	SUB Ev Ib 83	TEST Eb Gb 84	TEST Ev Gv 85	XCHG Eb Gb 86	XCHG Ev Gv 87	MOV Eb Gb 88	MOV Ev Gv 89	MOV Gb Eb 8A	MO Gv E 8B				
NOP 90	XCHG eAX eCX 91	XCHG eAX eDX 92	XCHG eAX eBX 93	XCHG eAX eSP 94	XCHG eAX eBP 95	XCHG eAX eSI 96	XCHG eAX eDI 97	CBW 98	CWD 99	CALL Ap 9A	WA 9B				

ADD -- Add

Opcode	Instruction	Clocks	Description
04 ib	ADD AL,imm8	2	Add immediate byte to AL
05 iw	ADD AX,imm16	2	Add immediate word to AX
05 id	ADD EAX,imm32	2	Add immediate dword to EAX
80 /0 ib	ADD r/m8,imm8	2/7	Add immediate byte to r/m byte
81 /0 iw	ADD r/m16,imm16	2/7	Add immediate word to r/m word
81 /0 id	ADD r/m32,imm32	2/7	Add immediate dword to r/m dword
83 /0 ib	ADD r/m16,imm8	2/7	Add sign-extended immediate byte to r/m word
83 /0 ib	ADD r/m32,imm8	2/7	Add sign-extended immediate byte to r/m dword
00 /r	ADD r/m8,r8	2/7	Add byte register to r/m byte
01 /r	ADD r/m16,r16	2/7	Add word register to r/m word
01 /r	ADD r/m32,r32	2/7	Add dword register to r/m dword
02 /r	ADD r8,r/m8	2/6	Add r/m byte to byte register
03 /r	ADD r16,r/m16	2/6	Add r/m word to word register
03 /r	ADD r32,r/m32	2/6	Add r/m dword to dword register

동일한 add 연산을 크기, operand 종류에 따라 9개의 회로로 구분

진짜 CPU에 대해 알아보자



add_sub.c

```c
int add1(int num){
  return num+=0b1111; // +=15
}

unsigned int add2(unsigned int num){
  return num+=0b1111; // +=15
}

int sub1(int num){
  return num-=0b1111; // -=15
}

unsigned int sub2(unsigned int num){
  return num-=0b1111;   // -=15
}
```

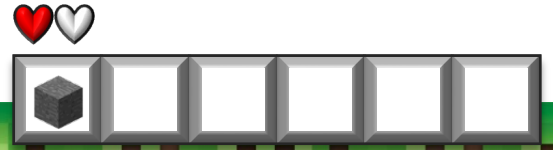
mult_div.c

```c
int div1(int num) {
    return num / 0b1010;   // 10
}

unsigned int div2(unsigned int num) {
    return num / 0b1000;   // 8
}

int mult1(int num) {
    return num *= 0b1010;    // 10
}

unsigned int mult2(unsigned int num) {
    return num *= 0b1000;    // 8
}
```



add

Giovanni tmp.s

X86 AT&T

```
1   .file   "tmp.c"
2   .text
3   .globl   _add1
4   .def   _add1;   .scl   2;   .type   32;   .endef
5  _add1:
6  LFB0:
7   .cfi_startproc
8   pushl   %ebp
9   .cfi_def_cfa_offset 8
10  .cfi_offset 5, -8
11  movl   %esp, %ebp
12  .cfi_def_cfa_register 5
13  addl   $15, 8(%ebp)
14  movl   8(%ebp), %eax
15  popl   %ebp
16  .cfi_restore 5
17  .cfi_def_cfa 4, 4
18  ret
19  .cfi_endproc
20 LFE0:
21  .globl   _add2
22  .def   _add2;   .scl   2;   .type   32;   .endef
23 _add2:
24 LFB1:
25  .cfi_startproc
26  pushl   %ebp
27  .cfi_def_cfa_offset 8
28  .cfi_offset 5, -8
29  movl   %esp, %ebp
30  .cfi_def_cfa_register 5
31  addl   $15, 8(%ebp)
32  movl   8(%ebp), %eax
33  popl   %ebp
34  .cfi_restore 5
35  .cfi_def_cfa 4, 4
36  ret
37  .cfi_endproc
38 LFE1:
39  .globl   _sub1
40  .def   _sub1;   .scl   2;   .type   32;   .endef
```

(%ebp)+8 에 매개변수 num 저장되어 있음

num+=15를 레지스터 eax에 저장(반환 값)

Bonita tmp.s

ARM

```
1   .section   __TEXT,__text,regular,pure_instructions
2   .build_version macos, 12, 0 sdk_version 12, 3
3   .globl   _add1                          ; -- Begin function add1
4   .p2align   2
5  _add1:                                   ; @add1
6   .cfi_startproc
7  ; %bb.0:
8   sub sp, sp, #16
9   .cfi_def_cfa_offset 16
10  str w0, [sp, #12]
11  ldr w8, [sp, #12]
12  add w0, w8, #15
13  str w0, [sp, #12]
14  add sp, sp, #16
15  ret
16  .cfi_endproc
17                                          ; -- End function
18  .globl   _add2                          ; -- Begin function add2
19  .p2align   2
20 _add2:                                   ; @add2
21  .cfi_startproc
22 ; %bb.0:
23  sub sp, sp, #16
24  .cfi_def_cfa_offset 16
25  str w0, [sp, #12]
26  ldr w8, [sp, #12]
27  add w0, w8, #15
28  str w0, [sp, #12]
29  add sp, sp, #16
30  ret
31  .cfi_endproc
32                                          ; -- End function
33  .globl   _sub1                          ; -- Begin function sub1
34  .p2align   2
```

_add1(signed), _add2(unsigned) 동작이 동일!

sub

X86 AT&T

```
41  _sub1:
42  LFB2:
43      .cfi_startproc
44      pushl   %ebp
45      .cfi_def_cfa_offset 8
46      .cfi_offset 5, -8
47      movl    %esp, %ebp
48      .cfi_def_cfa_register 5
49      subl    $15, 8(%ebp)
50      movl    8(%ebp), %eax
51      popl    %ebp
52      .cfi_restore 5
53      .cfi_def_cfa 4, 4
54      ret
55      .cfi_endproc
56  LFE2:
57      .globl   _sub2
58      .def    _sub2;    .scl    2;    .type    32;    .endef
59  _sub2:
60  LFB3:
61      .cfi_startproc
62      pushl   %ebp
63      .cfi_def_cfa_offset 8
64      .cfi_offset 5, -8
65      movl    %esp, %ebp
66      .cfi_def_cfa_register 5
67      subl    $15, 8(%ebp)
68      movl    8(%ebp), %eax
69      popl    %ebp
70      .cfi_restore 5
71      .cfi_def_cfa 4, 4
72      ret
73      .cfi_endproc
74  LFE3:
75      .ident   "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

ARM

```
35  _sub1:                                  ; @sub1
36    .cfi_startproc
37  ; %bb.0:
38    sub sp, sp, #16
39    .cfi_def_cfa_offset 16
40    str w0, [sp, #12]
41    ldr w8, [sp, #12]
42    subs  w0, w8, #15
43    str w0, [sp, #12]
44    add sp, sp, #16
45    ret
46    .cfi_endproc
47                                          ; -- End function
48    .globl  _sub2                           ; -- Begin function sub2
49    .p2align  2
50  _sub2:                                  ; @sub2
51    .cfi_startproc
52  ; %bb.0:
53    sub sp, sp, #16
54    .cfi_def_cfa_offset 16
55    str w0, [sp, #12]
56    ldr w8, [sp, #12]
57    subs  w0, w8, #15
58    str w0, [sp, #12]
59    add sp, sp, #16
60    ret
61    .cfi_endproc
62                                          ; -- End function
```

_sub1(signed), _sub2(unsigned) 동작이 동일!

Multi & Divide –O1 & -O3

-O1(default 최적화)

Bonita tmp.s -O1

ARM

```
1   .section  __TEXT,__text,regular,pure_instructions
2   .build_version macos, 12, 0 sdk_version 12, 3
3   .globl  _div1                           ; -- Begin function div1
4   .p2align  2
5 _div1:                                    ; @div1
6   .cfi_startproc
7 ; %bb.0:
8   sub sp, sp, #16
9   .cfi_def_cfa_offset 16
10  str w0, [sp, #12]
11  ldr w8, [sp, #12]
12  mov w9, #10
13  sdiv w0, w8, w9
14  add sp, sp, #16
15  ret
16  .cfi_endproc
17
18  .globl  _div2
19  .p2align  2
20 _div2:
21  .cfi_startproc
22 ; %bb.0:
23  sub sp, sp, #16
24  .cfi_def_cfa_offset 16
25  str w0, [sp, #12]
26  ldr w8, [sp, #12]
27  mov w9, #8
28  udiv w0, w8, w9
29  add sp, sp, #16
30  ret
31  .cfi_endproc
32
33  .globl  _mult1
34  .p2align  2
35 _mult1:                                  ; @mult1
36  .cfi_startproc
37 ; %bb.0:
38  sub sp, sp, #16
39  .cfi_def_cfa_offset 16
40  str w0, [sp, #12]
41  ldr w8, [sp, #12]
42  mov w9, #10
43  mul w0, w8, w9
44  str w0, [sp, #12]
45  add sp, sp, #16
46  ret
47  .cfi_endproc
48                                          ; -- End function
49  .globl  _mult2                          ; -- Begin function mult2
50  .p2align  2
51 _mult2:                                  ; @mult2
52  .cfi_startproc
53 ; %bb.0:
54  sub sp, sp, #16
55  .cfi_def_cfa_offset 16
56  str w0, [sp, #12]
57  ldr w8, [sp, #12]
58  lsl w0, w8, #3
59  str w0, [sp, #12]
60  add sp, sp, #16
61  ret
62  .cfi_endproc
63                                          ; -- End function
```

-O3(컴파일러 지원 최대 최적화)

Bonita tmp.s -O3

ARM

```
1   .section  __TEXT,__text,regular,pure_instructions
2   .build_version macos, 12, 0 sdk_version 12, 3
3   .globl  _div1                           ; -- Begin function div1
4   .p2align  2
5 _div1:                                    ; @div1
6   .cfi_startproc
7 ; %bb.0:
8   mov w8, #26215
9   movk  w8, #26214, lsl #16
10  smull x8, w0, w8
11  lsr x9, x8, #63
12  asr x8, x8, #34
13  add w0, w8, w9
14  ret
15  .cfi_endproc
16                                          ; -- End function
17  .globl  _div2                           ; -- Begin function div2
18  .p2align  2
19 _div2:                                   ; @div2
20  .cfi_startproc
21 ; %bb.0:
22  lsr w0, w0, #3
23  ret
24  .cfi_endproc
25                                          ; -- End function
26  .globl  _mult1                          ; -- Begin function mult1
27  .p2align  2
28 _mult1:                                  ; @mult1
29  .cfi_startproc
30 ; %bb.0:
31  add w8, w0, w0, lsl #2
32  lsl w0, w8, #1
33  ret
34  .cfi_endproc
35                                          ; -- End function
36  .globl  _mult2                          ; -- Begin function mult2
37  .p2align  2
38 _mult2:                                  ; @mult2
39  .cfi_startproc
40 ; %bb.0:
41  lsl w0, w0, #3
42  ret
43  .cfi_endproc
44                                          ; -- End function
```

div & mul 연산이 없음

Multi –O3

X86 AT&T

```
48  _mult1:
49  LFB2:
50      .cfi_startproc
51      pushl   %ebp
52      .cfi_def_cfa_offset 8
53      .cfi_offset 5, -8
54      movl    %esp, %ebp
55      .cfi_def_cfa_register 5
56      movl    8(%ebp), %edx
57      movl    %edx, %eax
58      sall    $2, %eax
59      addl    %edx, %eax
60      addl    %eax, %eax
61      movl    %eax, 8(%ebp)
62      movl    8(%ebp), %eax
63      popl    %ebp
64      .cfi_restore 5
65      .cfi_def_cfa 4, 4
66      ret
67      .cfi_endproc
68  LFE2:
69      .globl   _mult2
70      .def    _mult2;   .scl    2;    .type    32;    .endef
71  _mult2:
72  LFB3:
73      .cfi_startproc
74      pushl   %ebp
75      .cfi_def_cfa_offset 8
76      .cfi_offset 5, -8
77      movl    %esp, %ebp
78      .cfi_def_cfa_register 5
79      sall    $3, 8(%ebp)
80      movl    8(%ebp), %eax
81      popl    %ebp
82      .cfi_restore 5
83      .cfi_def_cfa 4, 4
84      ret
85      .cfi_endproc
86  LFE3:
87      .ident   "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

```
9   int mult1(int num) {
10      return num *= 0b1010;     // 10
11  }
12
13  unsigned int mult2(unsigned int num) {
14      return num *= 0b1000;     // 8
15  }
```

eax=num;
eax<<=2; // 2진수 체계에서 left shift n는 곱하기 2^n를 의미
eax+=num; // eax==(2^2*num)+num=5*num
eax+=eax; // eax==(5*num)+(5*num)=10*num

eax<<=3; // eax==(num^3)=8*num

연산 비용이 높은 div 연산 대신
시프트 연산과 더하기 연산을 이용

Divide –O3

피연산자가 2의 거듭제곱인 경우

X86 AT&T

```
    .file   "tmp.c"
    .text
    .globl  _div1
    .def    _div1;  .scl    2;  .type   32;  .endef
_div1:
LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    movl    8(%ebp), %ecx
    movl    $1717986919, %edx
    movl    %ecx, %eax
    imull   %edx
    sarl    $2, %edx
    movl    %ecx, %eax
    sarl    $31, %eax
    subl    %eax, %edx
    movl    %edx, %eax
    popl    %ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE0:
    .globl  _div2
    .def    _div2;  .scl    2;  .type   32;  .endef
_div2:
LFB1:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    movl    8(%ebp), %eax
    shrl    $3, %eax
    popl    %ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE1:
    .globl  _mult1
    .def    _mult1;  .scl    2;  .type   32;  .endef
```

?!

```
int div1(int num) {
    return num / 0b1010;    // 10
}

unsigned int div2(unsigned int num) {
    return num / 0b1000;    // 8
}
```

shrl: logical shift

-> 왼쪽 끝 비트는 0으로 채움(for unsigned)

sarl: arithmethic shift

-> 왼쪽 끝 비트는 부호비트로 채움(for signed)

eax>>=3; // eax==(num/2^3)=num/8

Divide –O3

피연산자가 2의 거듭제곱이 아닌 경우

X86 AT&T

```
_div1:
LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    movl    8(%ebp), %ecx
    movl    $1717986919, %edx   1)
    movl    %ecx, %eax          2)
    imull   %edx
    sarl    $2, %edx            3)
    movl    %ecx, %eax
    sarl    $31, %eax           4)
    subl    %eax, %edx
    movl    %edx, %eax          5)
    popl    %ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE0:
    .globl   _div2
    .def    _div2;    .scl    2;    .type    32;    .endef
```

```
1  #include <iostream>
2  #include <bitset>
3
4  int main(void){   // num/=10(unsigned)
5    unsigned int edx=1717986919;    // 2^32*(4/10)
6
7    unsigned int num=45;
8    unsigned int eax;
9    long unsigned edx_eax;
10
11   // 1
12   std::bitset<32> bit_32(edx);
13   std::cout<<"\nmovl $1717986919, %edx"<<std::endl;
14   std::cout<<"edx== "<<edx<<": "<<bit_32<<'\n'<<std::endl;
15
16   // 2
17   eax=num;
18   edx_eax=(unsigned long)eax*(unsigned long)edx;
19   std::bitset<64> bit_64(edx_eax);
20   std::cout<<"imull %edx"<<std::endl;
21   std::cout<<"edx:eax=eax*edx==\n"<<edx_eax<<": "<<bit_64<<'\n'<<std::endl;
22
23   // 3
24   edx=(unsigned int)(edx_eax>>(32+2));
25   bit_32=edx;
26   std::cout<<"sarl $2, %edx"<<std::endl;
27   std::cout<<"edx=eax>>2== "<<edx<<": "<<bit_32<<'\n'<<std::endl;
28
29   // 4
30   eax>>=31;
31   bit_32=eax;
32   std::cout<<"movl %ecx, %eax\nsarl $31, %eax"<<std::endl;
33   std::cout<<"eax>>=31== "<<eax<<": "<<bit_32<<'\n'<<std::endl;
34
35   // 5
36   edx-=eax;
37   bit_32=edx;
38   std::cout<<"subl %eax, %edx"<<std::endl;
39   std::cout<<"edx-=eax== "<<edx<<": "<<bit_32<<'\n'<<std::endl;
40
41   return 0;
42 }
```

Divide –O3

피연산자가 2의 거듭제곱이 아닌 경우

```
_div1:
LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    movl    8(%ebp), %ecx
    movl    $1717986919, %edx
    movl    %ecx, %eax
    imull   %edx
    sarl    $2, %edx
    movl    %ecx, %eax
    sarl    $31, %eax
    subl    %eax, %edx
    movl    %edx, %eax
    popl    %ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE0:
    .globl  _div2
    .def    _div2;  .scl    2;  .type   32;  .endef
```

X86 AT&T

```
1  #include <iostream>
2  #include <bitset>
3
4  int main(void){   // num/=10(unsigned)
5    unsigned int edx=1717986919;    // 2^32*(4/10)
6
7    unsigned int num=45;
8    unsigned int eax;
9    long unsigned edx_eax;
```

```
movl $1717986919, %edx
edx== 1717986919: 01100110011001100110011001100111

imull %edx
edx:eax=eax*edx==
77309411355: 0000000000000000000000001001000000000000000000000000011011

sarl $2, %edx
edx=eax>>2== 4: 00000000000000000000000000000100

movl %ecx, %eax
sarl $31, %eax
eax>>=31== 0: 00000000000000000000000000000000

subl %eax, %edx
edx-=eax== 4: 00000000000000000000000000000100
```

45//10=4

```
32   std::cout<<"movl %ecx, %eax\nsarl $31, %eax"<<std::endl;
33   std::cout<<"eax>>=31== "<<eax<<": "<<bit_32<<'\n'<<std::endl;
34
35   // 5
36   edx-=eax;
37   bit_32=edx;
38   std::cout<<"subl %eax, %edx"<<std::endl;
39   std::cout<<"edx-=eax== "<<edx<<": "<<bit_32<<'\n'<<std::endl;
40
41   return 0;
42 }
```

Divide –O3

피연산자가 2의 거듭제곱이 아닌 경우

X86 AT&T

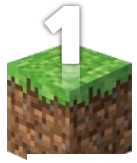
```
_div1:
LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    movl    8(%ebp), %ecx
    movl    $1717986919, %edx
    movl    %ecx, %eax
    imull   %edx
    sarl    $2, %edx
    movl    %ecx, %eax
    sarl    $31, %eax
    subl    %eax, %edx
    movl    %edx, %eax
    popl    %ebp
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE0:
    .globl  _div2
    .def    _div2;  .scl    2;  .type   32;  .endef
```

1717986918==2^32*(4/10)
gcc 컴파일러가 계산을 빨리 하도록 미리 계산

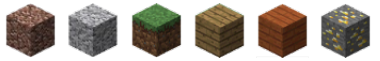
2^32*(4/10)*num>>2
최하위 32비트 값을 무시하게 되면

(num)/10 값을 얻음

num의 부호비트를 뽑아서,
서로 뺄셈 == 보수 처리



진짜 CPU에 대해 알아보자



Optimizer and Non-Optimizer

X86 AT&T

```
1  #include <iostream>
2  #include <chrono>
3
4  unsigned int div10(unsigned int num){
5    return num/=10;
6  }
7
8  unsigned int div10_O(unsigned int num){
9    static long int magic_num=1717986918l;
10   return (unsigned int)((magic_num*num)>>34);
11 }
12
13 int main(void){
14   unsigned int result;
15   std::chrono::system_clock::time_point start, end;
16
17   // #1
18   start=std::chrono::system_clock::now();
19   for(int i=0; i<10000000; ++i){
20     div10(350);
21   }
22   end=std::chrono::system_clock::now();
23   std::chrono::nanoseconds time=end-start;
24   std::cout<<"COST TIME1: "<<time.count()<<"[ns]"<<std::endl;
25
26   // #2
27   start=std::chrono::system_clock::now();
28   for(int i=0; i<10000000; ++i){
29     div10_O(350);
30   }
31   end=std::chrono::system_clock::now();
32   time=end-start;
33   std::cout<<"COST TIME2: "<<time.count()<<"[ns]"<<std::endl;
34
35   return 0;
36 }
```

나눗셈 회로를 통한 연산

나눗셈 회로 대신,
곱셈 회로+시프트 회로를 이용한 연산

```
COST TIME1: 40550000[ns]
COST TIME2: 25858000[ns]
```

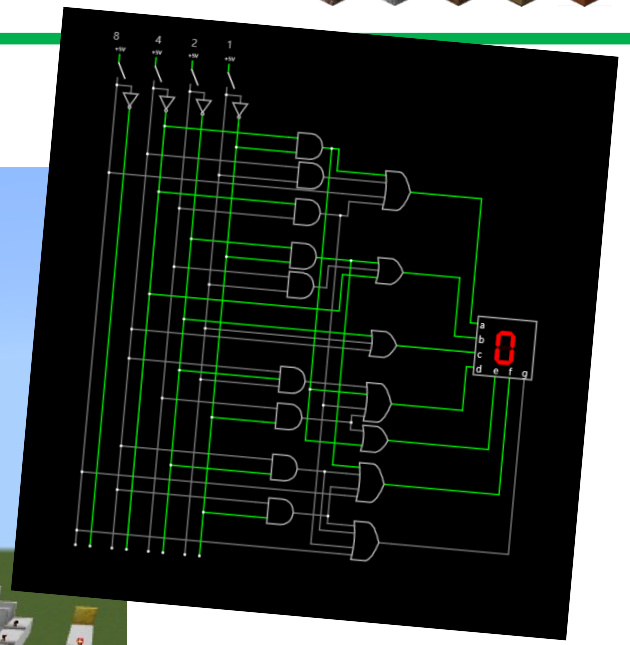
대략 2배의 시간이 소요된다.

컴퓨터는 최대한 나눗셈을 사용하지 않는 방향으로 연산을 수행!

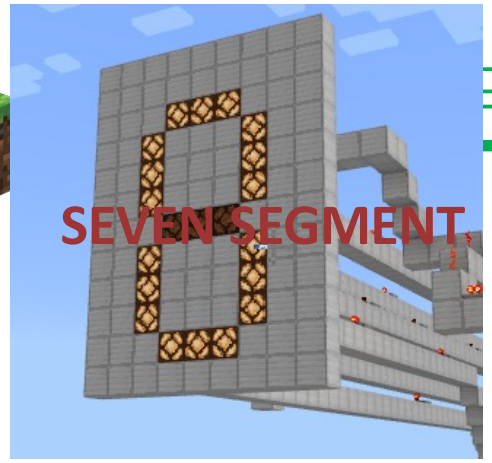
2

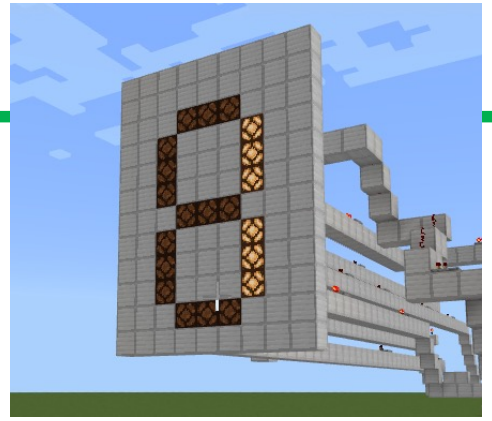
이번주 마인크래프트

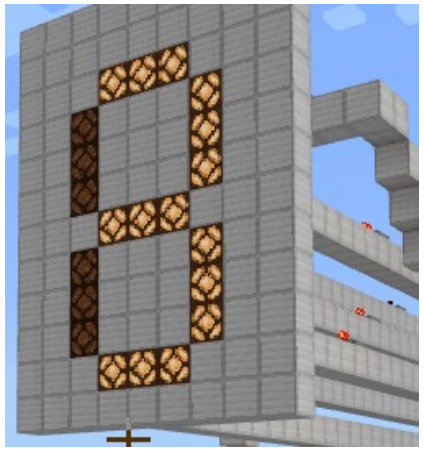
SEVEN SEGMENT

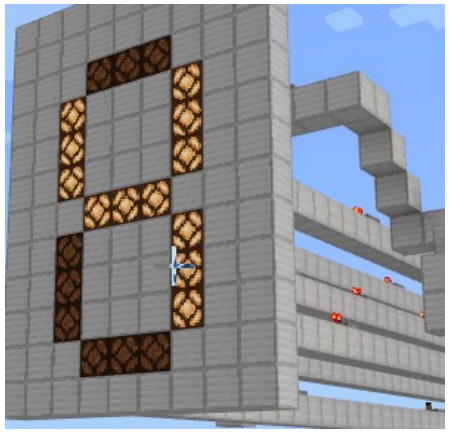


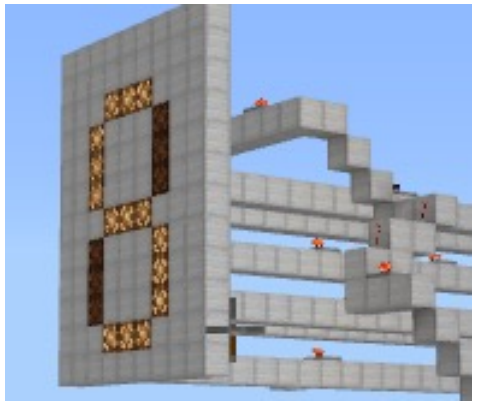
SEVEN SEGMENT

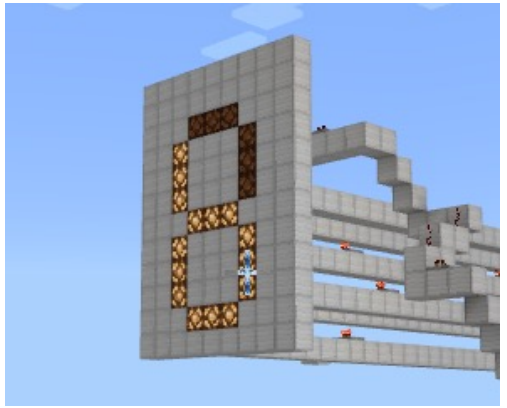


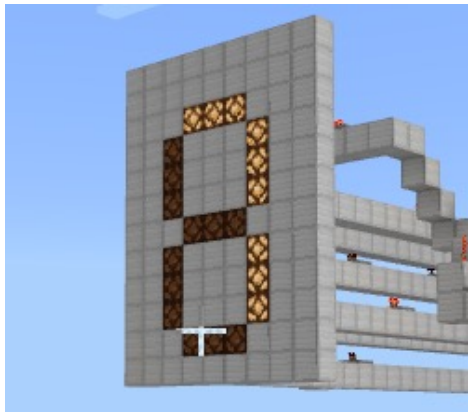


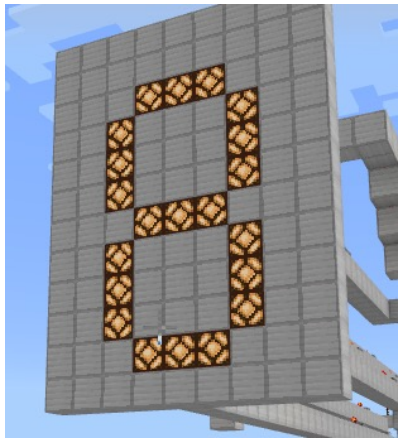


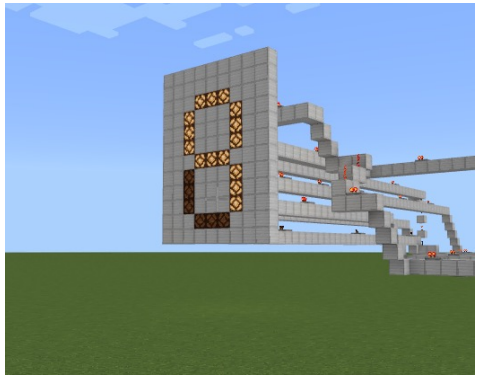


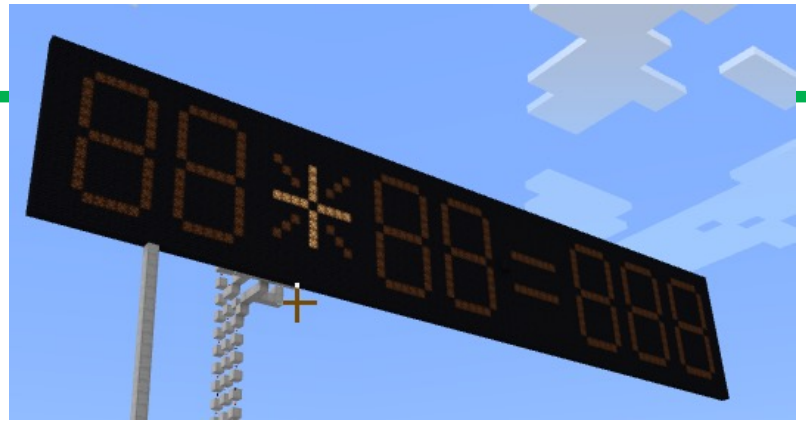


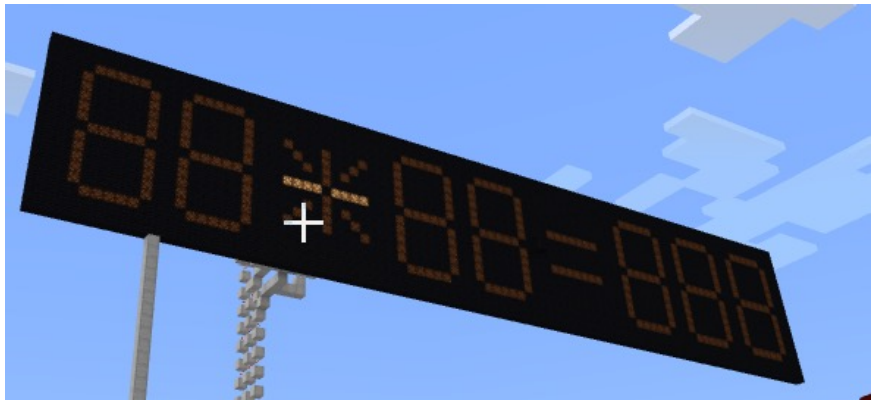












다음 주!

전광판에

Operand1, Operand2, Result 연결해서

8bit 덧셈+뺄셈 계산기 완성


치: 38, 2, 22
Thank you
TEAM.STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘


MINECRAFT
A circle-free environment!
khuReeper
스티브조의 시작
컨텐츠 제작
배운 것들
탈의실
©Mojang AB

1 스티브조의 시작


TAX+
TAX−
MRC
M−
M+
%
÷
×
−
+
=
·
0
1
2
3
4
5
6
7
8
9

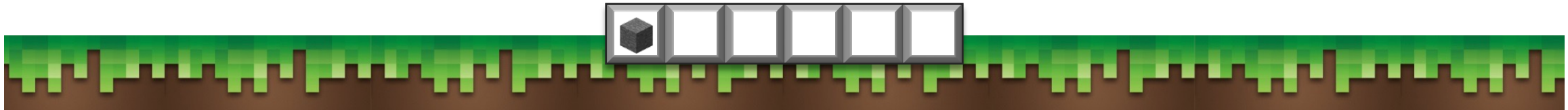
2

협동심제고 컨텐츠 제작


MINECRAFT


Pr

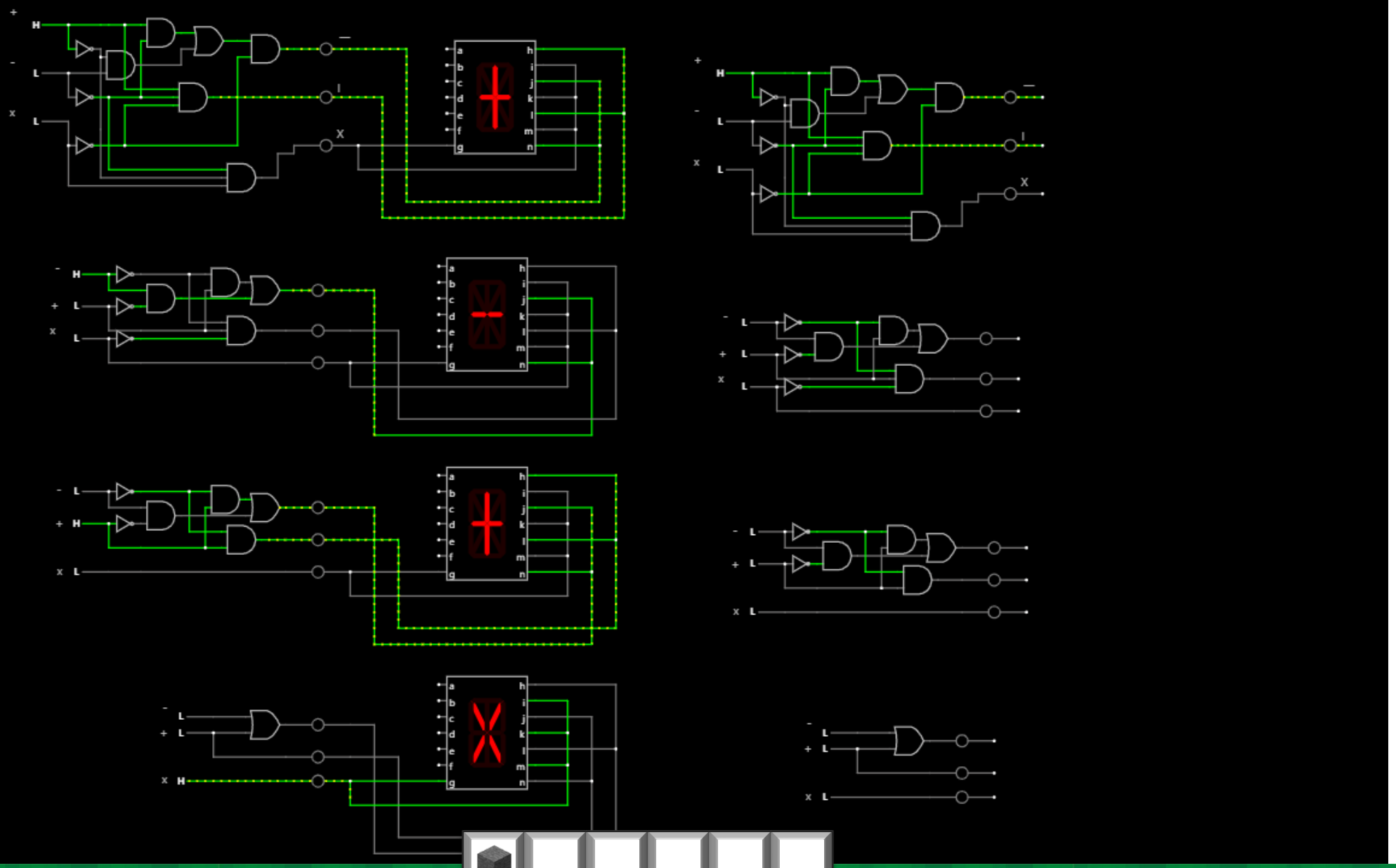

Ps



3

우리가 배운 것들

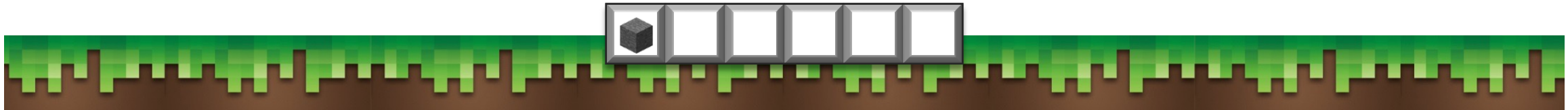
14-segment (+,-,x)




위치: 18, -9, 110
88x88=8888


Pr


Ps



경희대학교 미래인재센터
OFFICE OF CAREER DEVELOPMENT
KYUNG HEE UNIVERSITY

프로그램 참가 확인서

소속 이과대학
성명 한동휘
프로그램 2022학년도 미디어 리터러시 교육 – 프리미어 기초
운영기간 **2022.5.24. 19:00~22:00**

상기 인은 경희대학교 미래인재센터에서 주관한 <미디어 리터러시 교육 – 프리미어 기초> 프로그램에 참여하였음을 확인합니다.

2022년 5월 27일

확인자 : 박 주 아 (서명 또는 인)
연락처 : 미래인재센터 오픈랩 02-961-0243

경희대학교 미래인재센터

경희대학교 미래인재센터
OFFICE OF CAREER DEVELOPMENT
KYUNG HEE UNIVERSITY

프로그램 참가 확인서

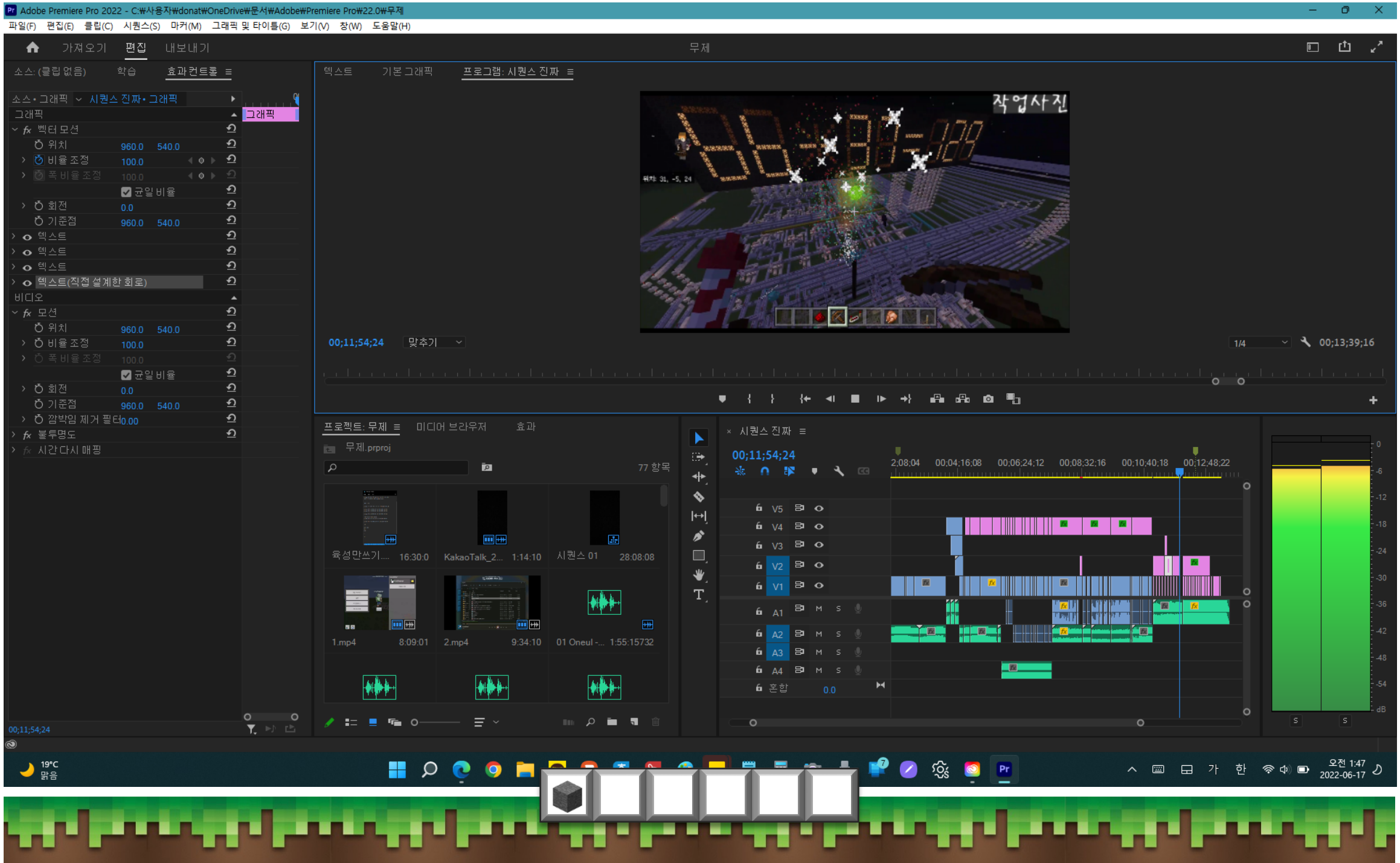
소속 이과대학
성명 한동휘
프로그램 2022학년도 미디어 리터러시 교육 – 포토샵
운영기간 **2022.5.31. 19:00~22:00**

상기 인은 경희대학교 미래인재센터에서 주관한 <미디어 리터러시 교육 – 포토샵 기초> 프로그램에 참여하였음을 확인합니다.

2022년 6월 3일

확인자 : 박 주 아 (서명 또는 인)
연락처 : 미래인재센터 오픈랩 02-961-0243

경희대학교 미래인재센터


Adobe Premiere Pro 2022 - C:₩사용자₩donat₩OneDrive₩문서₩Adobe₩Premiere Pro₩22.0₩무제
파일(F) 편집(E) 클립(C) 시퀀스(S) 마커(M) 그래픽 및 타이틀(G) 보기(V) 창(W) 도움말(H)
가져오기 편집 내보내기
무제
소스: (클립 없음)
학습
효과 컨트롤
소스 • 그래픽 시퀀스 진짜 • 그래픽
그래픽
벡터 모션
위치 960.0 540.0
비율 조정 100.0
균일 비율
회전 0.0
기준점 960.0 540.0
텍스트
텍스트
텍스트
텍스트(직접 설계한 회로)
비디오
모션
위치 960.0 540.0
비율 조정 100.0
균일 비율
회전 0.0
기준점 960.0 540.0
깜박임 제거 필터 0.00
불투명도
시간 다시 매핑
텍스트
기본 그래픽
프로그램: 시퀀스 진짜
작업사진
00;11;54;24
맞추기
1/4
00;13;39;16
프로젝트: 무제
미디어 브라우저
효과
무제.prproj
77 항목
육성만쓰기.... 16:30:0
KakaoTalk_2... 1:14:10
시퀀스 01 28:08:08
1.mp4 8:09:01
2.mp4 9:34:10
01 Oneul -... 1:55:15732
시퀀스 진짜
00;11;54;24
2;08:04 00;04;16;08 00;06;24;12 00;08;32;16 00;10;40;18 00;12;48;22
V5 V4 V3 V2 V1
A1 A2 A3 A4
혼합 0.0
19°C 맑음
오전 1:47 2022-06-17


181+49=28


위치: 68, 3, 58
khuFigLin
khuEnderman
khuSkeleTon


4
마지막화

동영상 ㅎ

Thank you

TEAM STEVE
ENDERMAN 김병수
WITHER 김상혁
SKELETON 김재헌
PIGLIN 최정훈
CREEPER 한동휘