AVR 프로그래밍

4강

정보컴퓨터공학과 권혁동

https://youtu.be/4iVy9tvFrmE

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

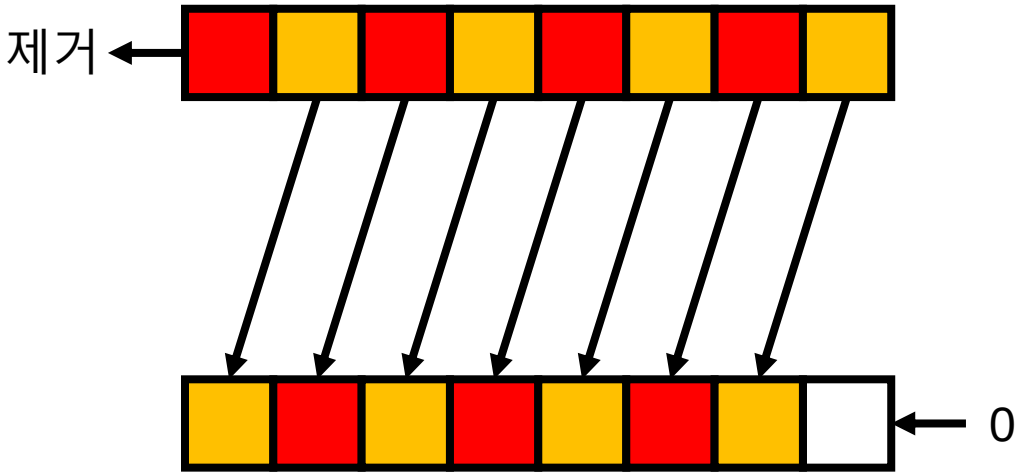
로테이션 연산 구현

- 사용할 명령어 모음

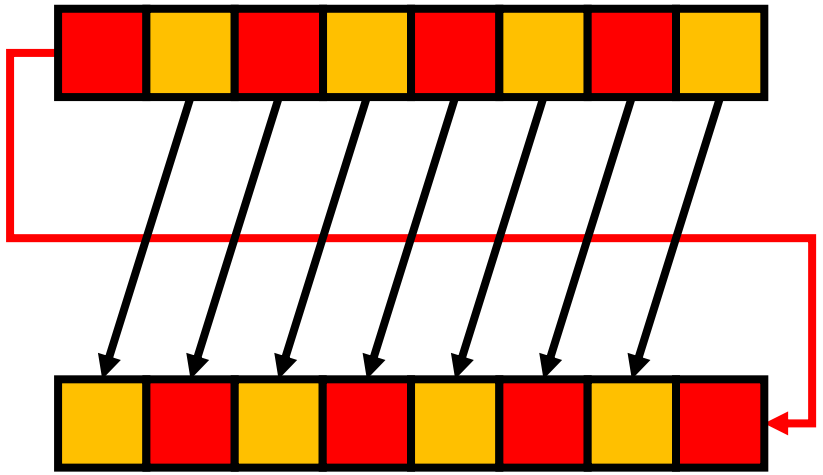
명령어	동작
ROL	좌측으로 로테이션
ROR	우측으로 로테이션
LSL	좌측으로 시프트
LSR	우측으로 시프트
BLD	레지스터의 특정 비트를 플래그로 이동
BST	플래그를 레지스터의 특정 비트로 이동
PUSH	레지스터의 값을 스택으로 이동
POP	스택의 값을 레지스터로 이동

로테이션 연산 구현

- 로테이션과 시프트
- 시프트: 비트를 이동 후, 끝 자리에 0으로 채움
- 로테이션: 비트를 이동 후, **끝 자리에 반대쪽 값**이 옴



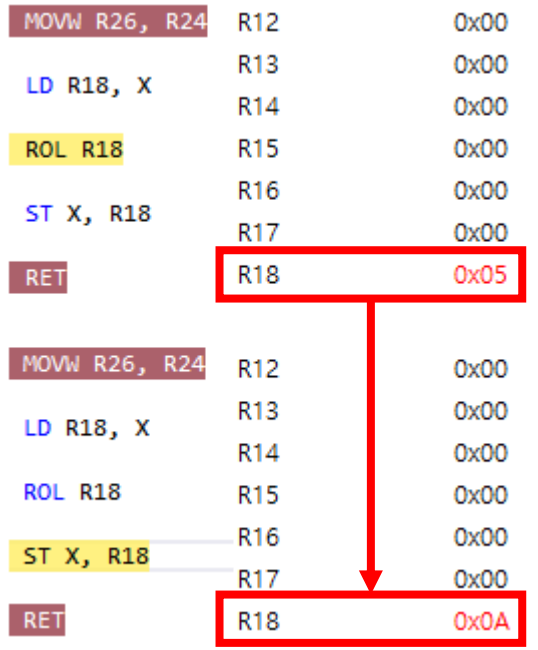
좌측 시프트



좌측 로테이션

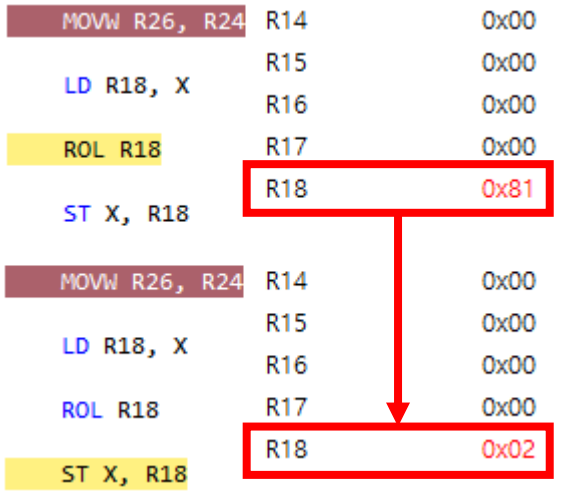
로테이션 연산 구현

- ROL 명령어 이후 모든 비트가 좌측으로 1회 로테이션
  - 좌측 비트 이동은 2 곱셈 연산과 동일
  - 우측 비트 이동은 2 나누기 연산과 동일
- ROR, LSL, LSR에서도 비슷한 결과를 확인 가능



로테이션 연산 구현

- 로테이션 연산의 맹점
- 8-bit 로테이션 시, **끝 값이 제대로 이동하지 않는 경우**가 발생
  - 시작: 0x81 (= 0b10000001)
  - 예상: 0x03 (= 0b00000011)
  - 실제: 0x02 (= 0b0000001**0**)
- ROR도 동일하게 발생
  - LSL, LSR은 로테이션이 아니므로 문제 없음



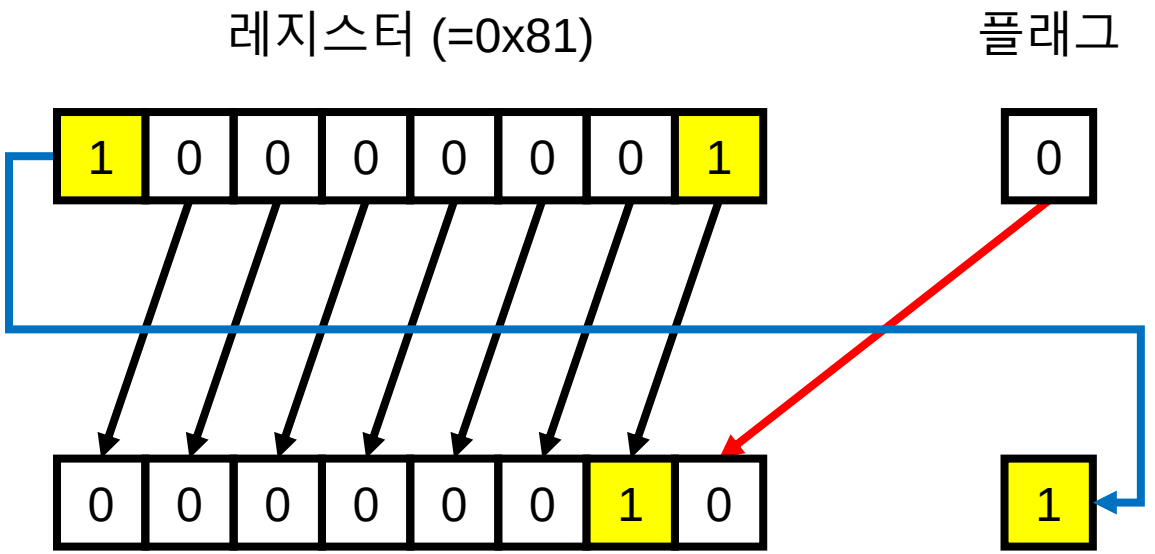
로테이션 연산 구현

- ROL의 연산 순서

1. Rd(0) ← C
2. Rd(n+1) ← Rd(n)
3. C ← Rd(7)

- ROR의 연산 순서

1. Rd(7) ← C
2. Rd(n) ← Rd(n+1)
3. C ← Rd(0)

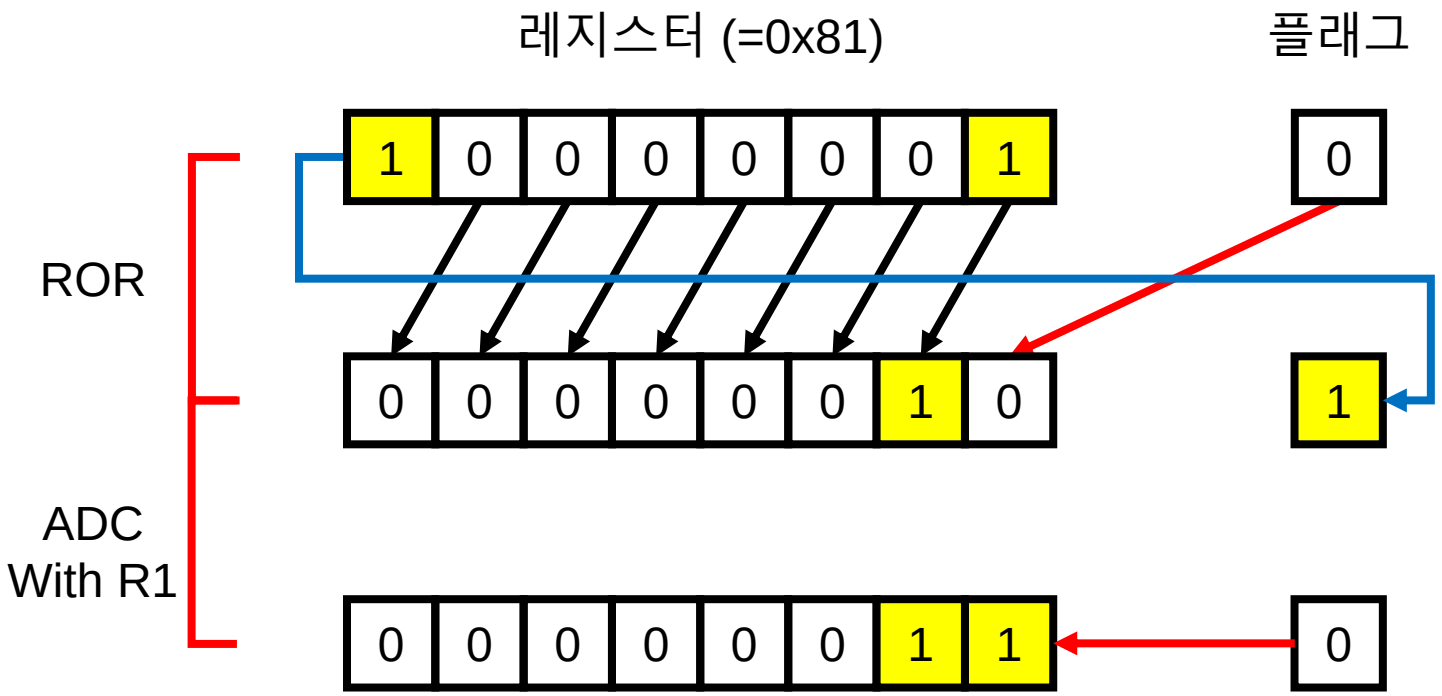


- **캐리가 가장 먼저 들어오기 때문에** 정상적인 결과물 출력이 불가능
  - SHL, SHR도 최상위 또는 최하위 비트가 소멸이 아닌 캐리 플래그로 이동

로테이션 연산 구현

- 정확한 로테이션을 위해서 ADC 또는 BST와 BLD 명령어 활용
  - ROL Rd
  - ADC Rd, R1(= ZERO)

  - BST Rd, 0
  - ROR Rd
  - BLD Rd, 7



로테이션 연산 구현

8-bit above 로테이션 구현

- ROR, ROL은 하나의 레지스터 대상으로 연산
- **따라서 16-bit, 32-bit 로테이션은**

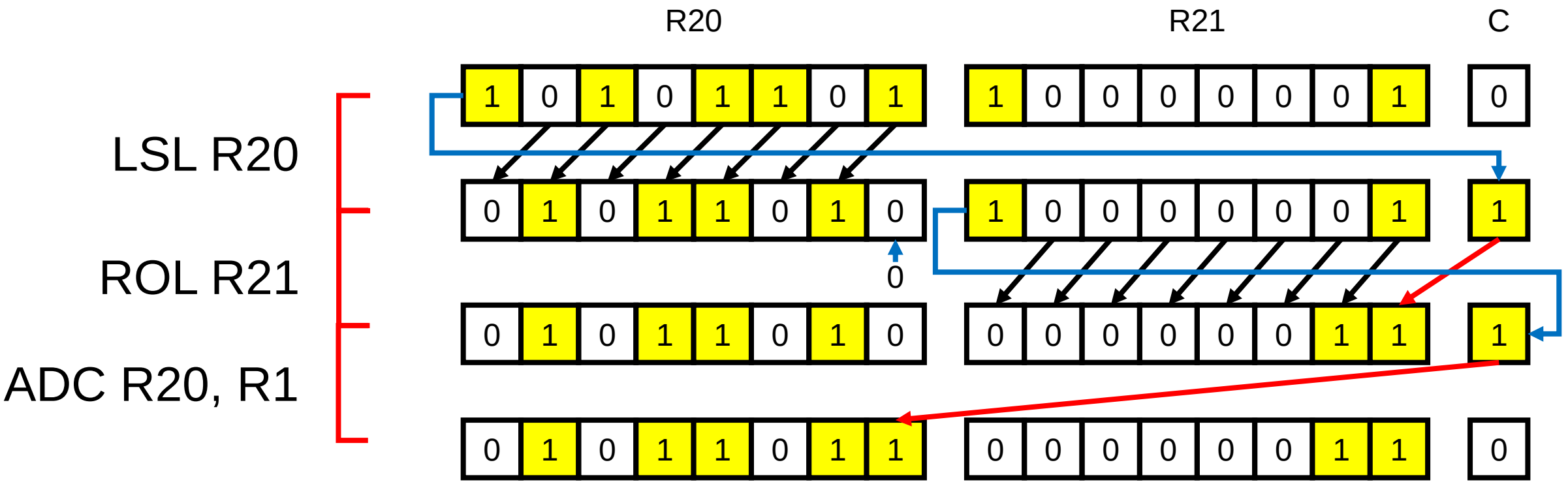
  ROR, ROL 특성과 다른 명령어를 활용

- 로테이션 횟수에 따라 각기 다른 형태로 구현

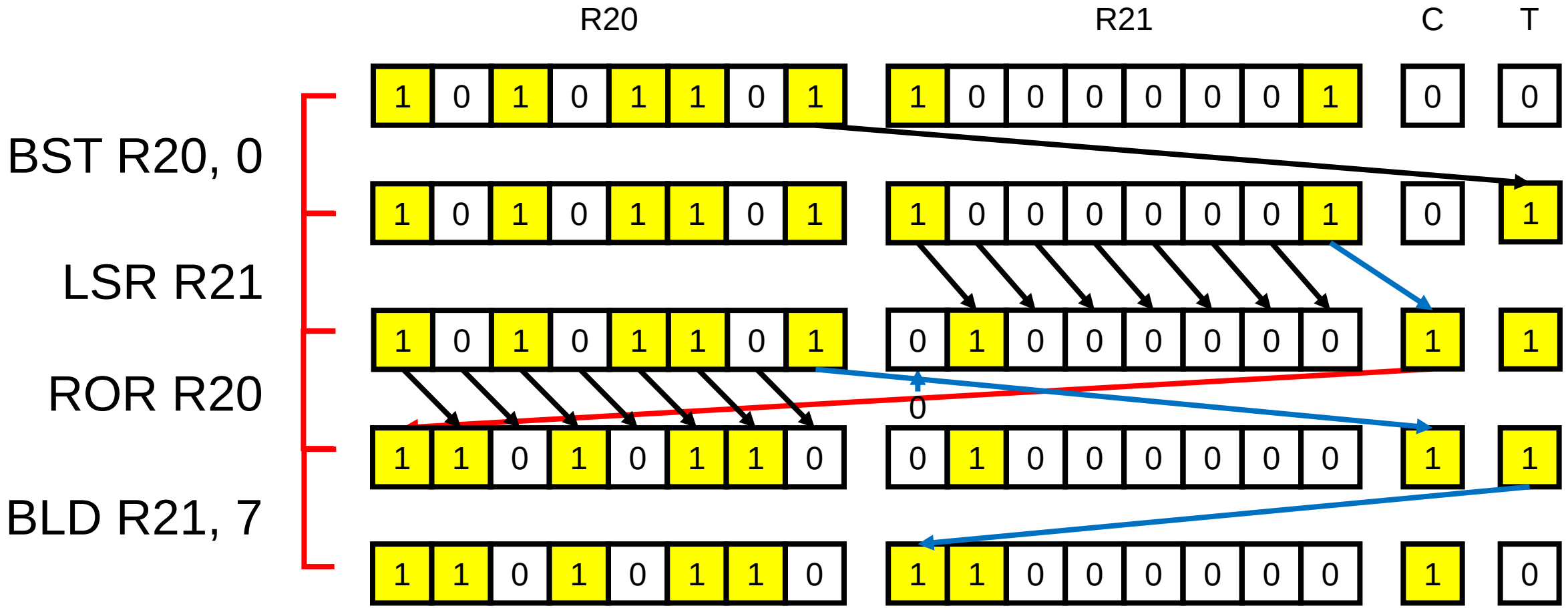
로테이션 연산 구현

	Left side	Right side
16-bit	LSL Rd1 ROL Rd2 ADC Rd1, R1(= ZERO)	BST Rd1, 0 LSR Rd2 ROR Rd1 BLD Rd2, 7
32-bit	LSL Rd1 ROL Rd2 ROL Rd3 ROL Rd4 ADC Rd1, R1(= ZERO)	BST Rd1, 0 LSR Rd4 ROR Rd3 ROR Rd2 ROR Rd1 BLD Rd4, 7

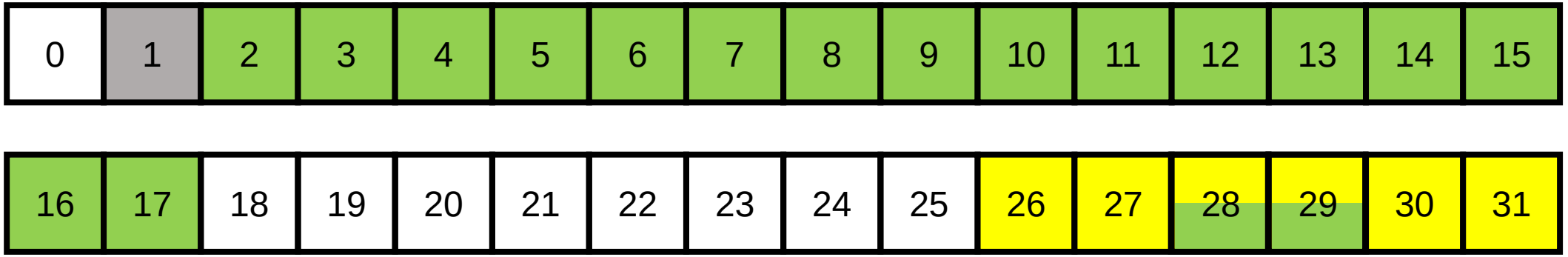
로테이션 연산 구현


R20
R21
C
LSL R20
ROL R21
ADC R20, R1
1 0 1 0 1 1 0 1
1 0 0 0 0 0 0 1
0
0 1 0 1 1 0 1 0
1 0 0 0 0 0 0 1
1
0
0 1 0 1 1 0 1 0
0 0 0 0 0 0 1 1
1
0 1 0 1 1 0 1 1
0 0 0 0 0 0 1 1
0

로테이션 연산 구현


R20
R21
C
T
BST R20, 0
LSR R21
ROR R20
BLD R21, 7
1 0 1 0 1 1 0 1
1 0 0 0 0 0 0 1
0
0
1 0 1 0 1 1 0 1
1 0 0 0 0 0 0 1
0
1
1 0 1 0 1 1 0 1
0 1 0 0 0 0 0 0
1
1
0
1 1 0 1 0 1 1 0
0 1 0 0 0 0 0 0
1
1
1 1 0 1 0 1 1 0
1 1 0 0 0 0 0 0
1
0

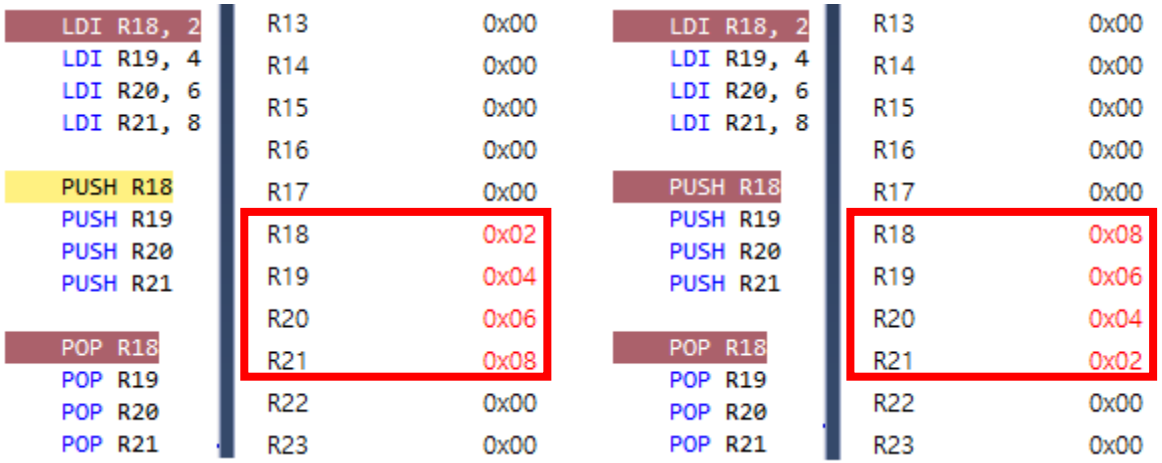
스택 활용



- R2~R17, R28, R29 레지스터를 사용할 때는 **값 보존**이 필요
  - Callee saved registers
- **스택에 값을 저장해두는 것으로 보존** 가능
- PUSH, POP 명령어 사용
  - PUSH Rd: Rd 레지스터의 값을 스택으로
  - POP Rd: 스택의 값을 Rd 레지스터로

스택 활용

- 스택은 후입선출(LIFO)구조
  - Last In First Out
- 따라서 스택에 값을 저장하고 반출할 때는 역순으로 반출

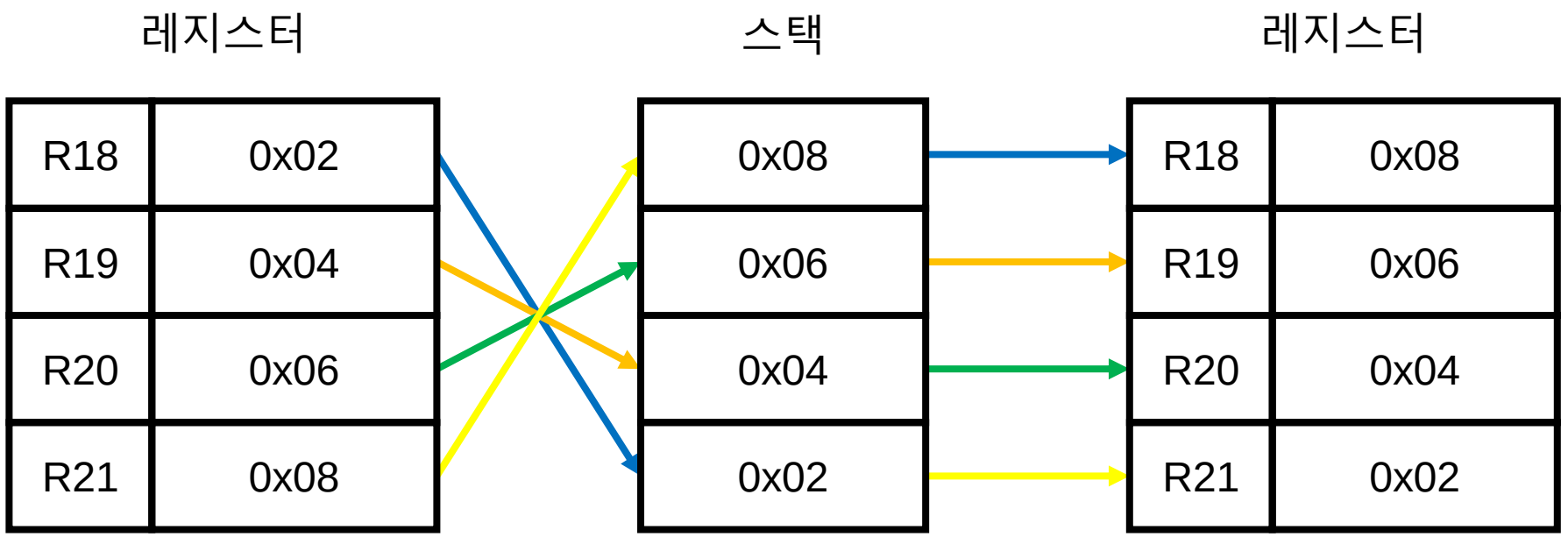


스택 활용

```
LDI R18, 2
LDI R19, 4
LDI R20, 6
LDI R21, 8

PUSH R18
PUSH R19
PUSH R20
PUSH R21

POP R18
POP R19
POP R20
POP R21
```



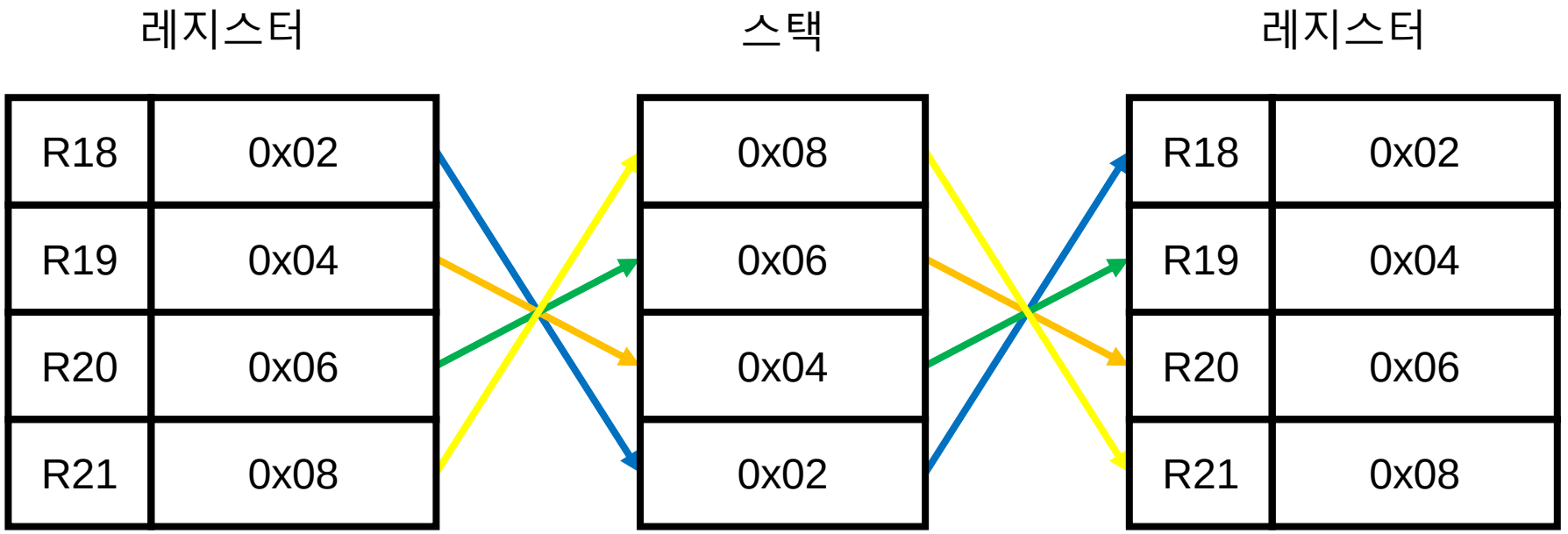
- 스택 반출을 **원래 순서**대로 한다면, 의도한 값이 **역순**으로 됨

스택 활용

```
LDI R18, 2
LDI R19, 4
LDI R20, 6
LDI R21, 8

PUSH R18
PUSH R19
PUSH R20
PUSH R21

POP R21
POP R20
POP R19
POP R18
```



- 스택 **반출을 역순**으로 한다면, 의도한 값으로 **복구됨**
  - 스택 반입을 역순으로 하고 스택 반출을 역순으로 해도 동일

Q & A