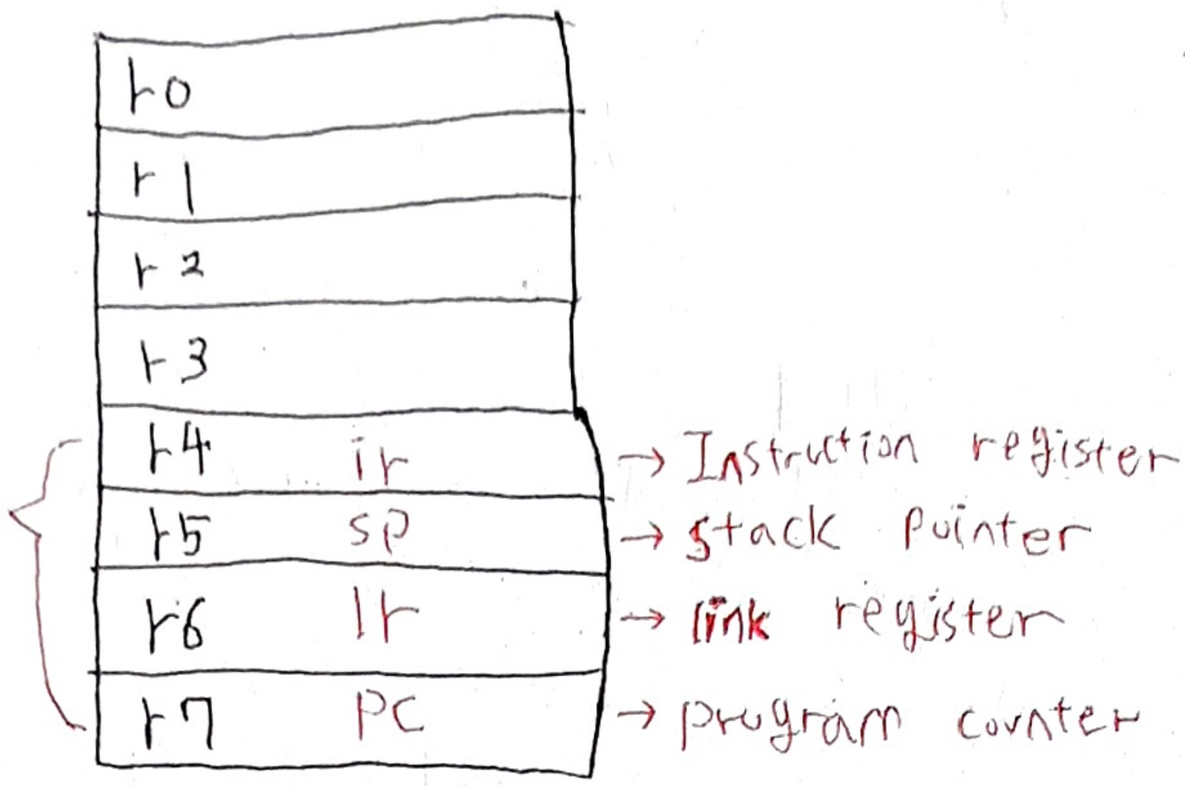
4장	컴퓨터 구조의 접근방법

-1/9 이충헌

32/64

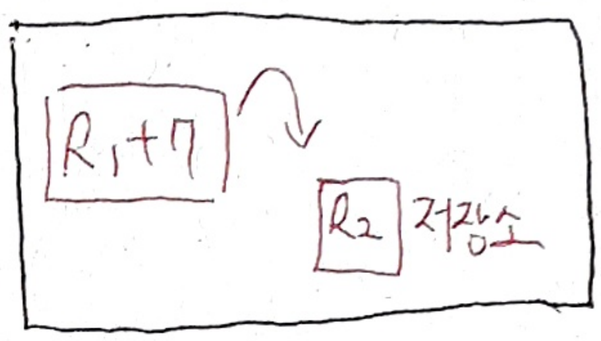
- 몇 Bit? (보통 n bit)
- 몇개 레지스터? (다다익선)
- 무슨 용도?

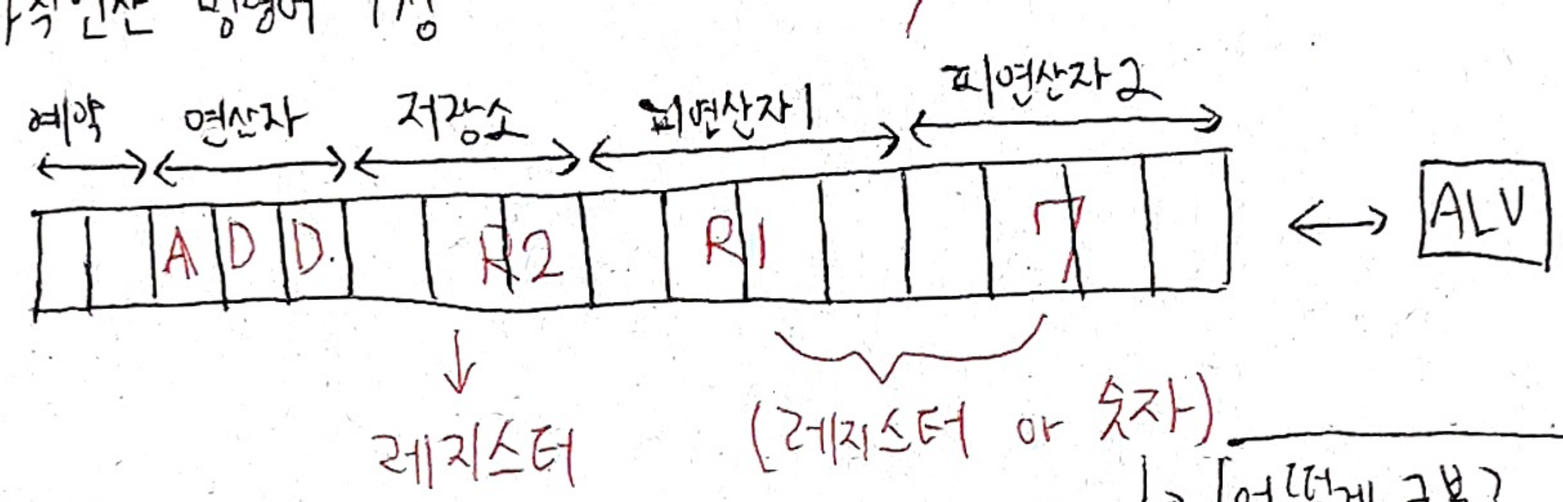
<레지스터 디자인>



<명령어 디자인>

- CISC의 간소화 RISC (고성능)
- 명령어 기본모델 = 16 비트
- 사칙연산 명령어 구성





어떻게 구별?

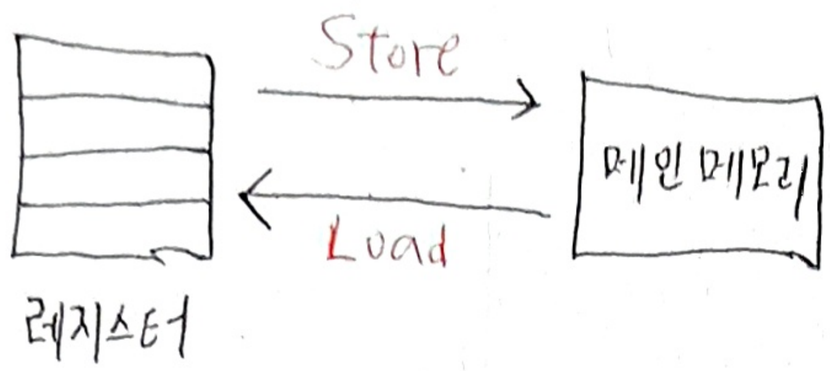
0 → 숫자

1 → 레지스터 정보

+	001
−	010
X	011
÷	100

r0	000
r1	001
r2	010
r3	011
r4	100
r5	101
r6	110
r7	111

<Load & Store 명령어의 필요성>



ex)

```
int a=10; // 0x10 번지 할당
int b=20; // 0x20 〃
int c=0; // 0x30 〃
c=a+b;
```

// (ADD) (c) (a) (b)

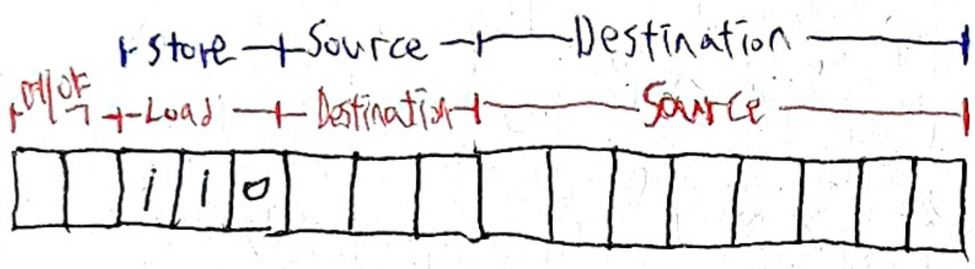
ADD → 연산결과 / c → 반드시 레지스터 / a, b → 숫자 or 레지스터

명령어의 제한

① 사칙연산의 피연산자는 숫자 or 레지스터

② 연산결과는 레지스터에 저장 -> 레지스터통해 모든 연산 진행하도록 디자인한다!

Load 명령어
Store



Load r1, 0x20

목적지 번지에 있는 값(메모리)

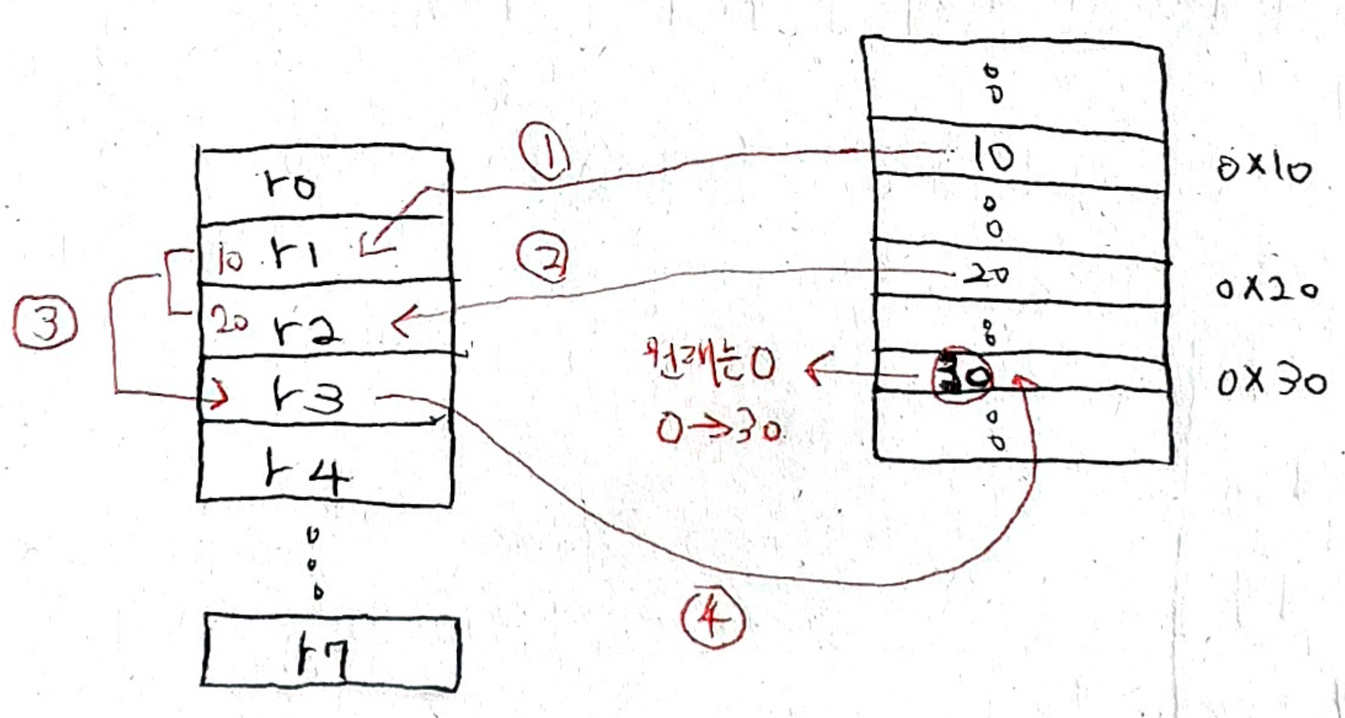
- Destination = 데이터 저장할 메모리 주소 정보
- Source = 데이터 읽어올 레지스터 정보

- Destination = 데이터 저장할 레지스터 정보
- Source = 데이터 읽어올 메모리 주소 정보

Store r1, 0x20

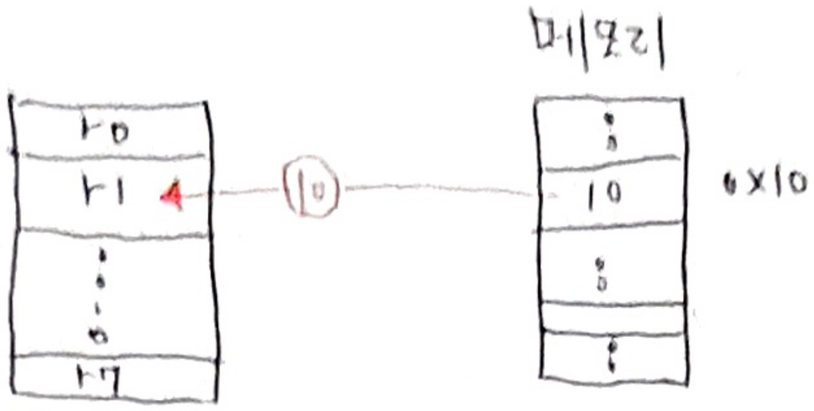
ex)

① Load r1, 0x10
② Load r2, 0x20
③ ADD r3, r1, r2
④ Store r3, 0x30



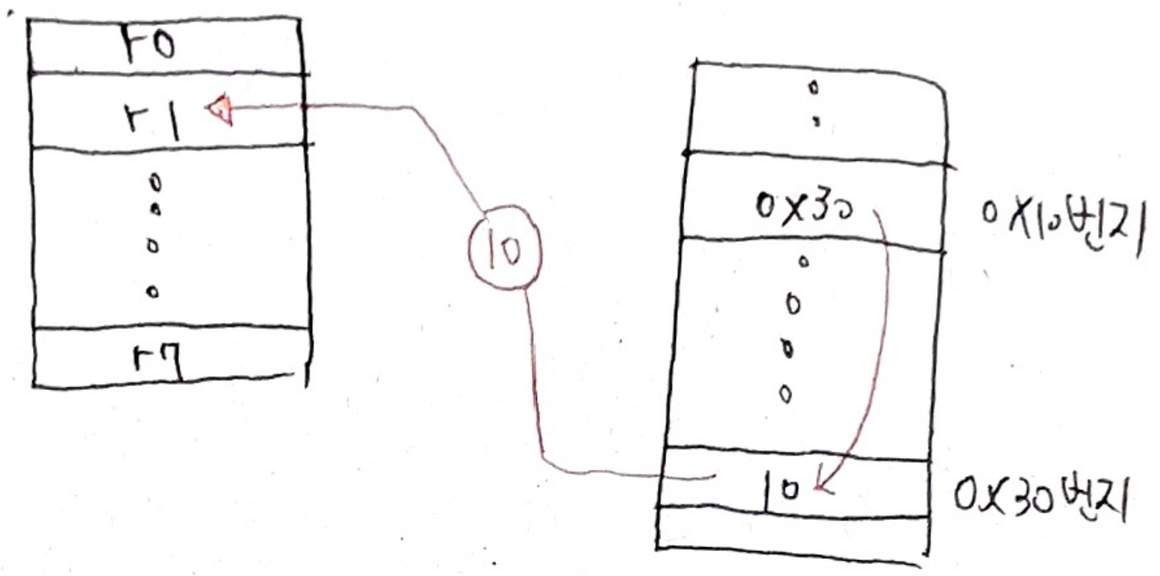
<Direct & InDirect 모드>

Direct 모드 - 명령어의 메모리 참조



<Load r1, 0x10 실행시>

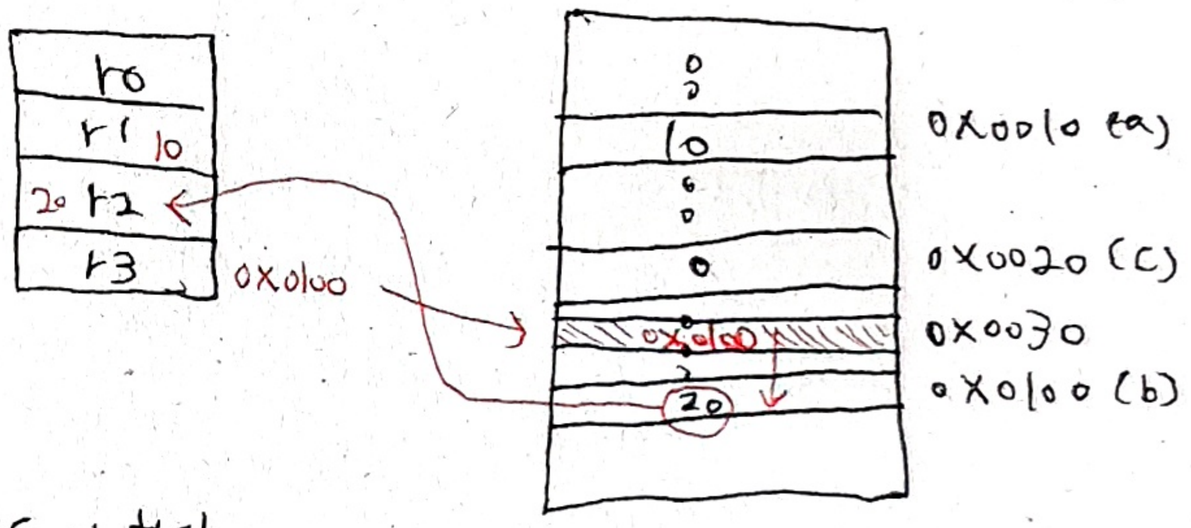
Indirect 모드 - 0x10 번지에 있는 값을 메모리로 참조



<Load r1, [0x10]>

ex)

```
int a=10; // 0x0010 번지 할당
int b=20; // 0x0100 〃
int c=0; // 0x0020 〃
c=a+b;
```



Load r1, 0x0010

실제로 0x0100번지는 표현불가 → 직접 만들어야 한다.

MUL r0, 4, 4 16

MUL r2, 4, 4 16

MUL r3, r0, r2 256 → 100번지 (16진수로)

store r3, 0x0030

Load r2, [0x0030] ← Indirect 모드, 결과적으로 r2에 20 들어감

ADD r3, r1, r2 // 30