Arithmetic for computers

wnwlcks123 · 어제

CS

CS

목록 보기 2/2

컴퓨터 연산

1.서론

이 장의 목표

실수의 표현과 연산 알고리즘, 이러한 알고리즘을 수행하는 하드웨어, 그리고 이 모든 것들이 명령어 집합에 미치는 영향 등을 풀어 나가는것.

2.덧셈과 뺄셈

- 덧셈.

1. 7 + 6

0000 0111 (7)
0000 0110 (6) +

'--------------'
0000 1101 (13)

- 뺄셈.

2. 7 - 6

0000 0111 (7)
0000 0110 (6) -
'--------------'
0000 00001

덧셈과 뺄셈은 이진법으로 더하고 뺄 수 있다.

2의 보수법을 이용하여 -6을 더하는 방식으로 할 수도 있다.

0000 0111 (7)
1111 1010 (-6) +
'--------------'
0000 0001 (1)

- 오버플로우

오버플로우 : 어떤 결과가 그 비트 어떤 범위를 벗어나는 경우.

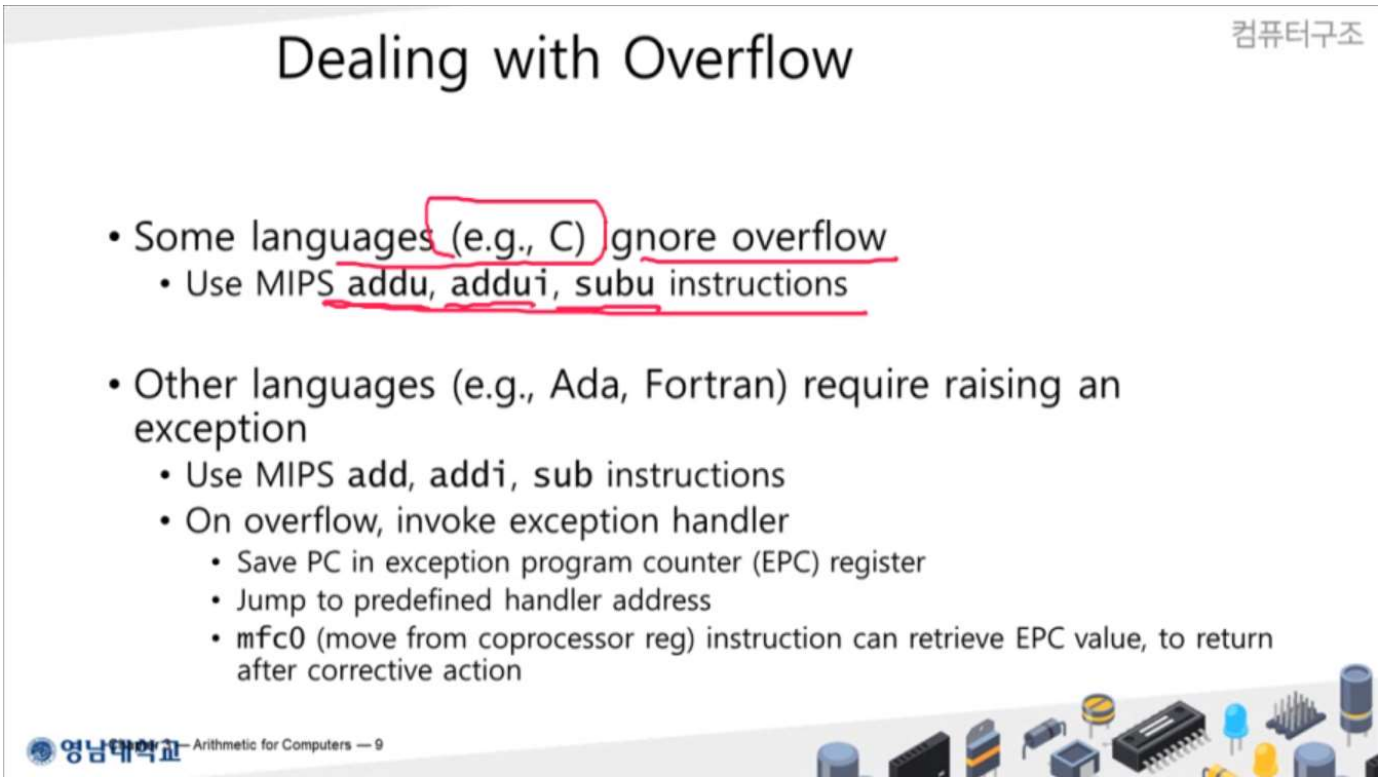
- 양수와 음수를 더할 때에는 오버 플로가 발생할 수 없다.
  ex) -10 + 4 = -6 : 피 연산자는 이미 32비트로 표현된 값이고, 합은 어느 피연산자 보다도 크지 않으므로 합 또한 32비트로 표현이 가능하다. 따라서 양수와 음수를 더할 때에는 오버플로가 발생할 수 없다.

- 뺄셈의 경우에도 똑같이 오버플로가 발생하지 않는다.
  피연산자의 부호가 같을 경우에는 오버플로가 발생할 수 없다.
  ex) c - a = c + (-a) : 두번째 피연산자의 부호를 바꾸어 더하는 방식으로 뺄셈을 처리하므로

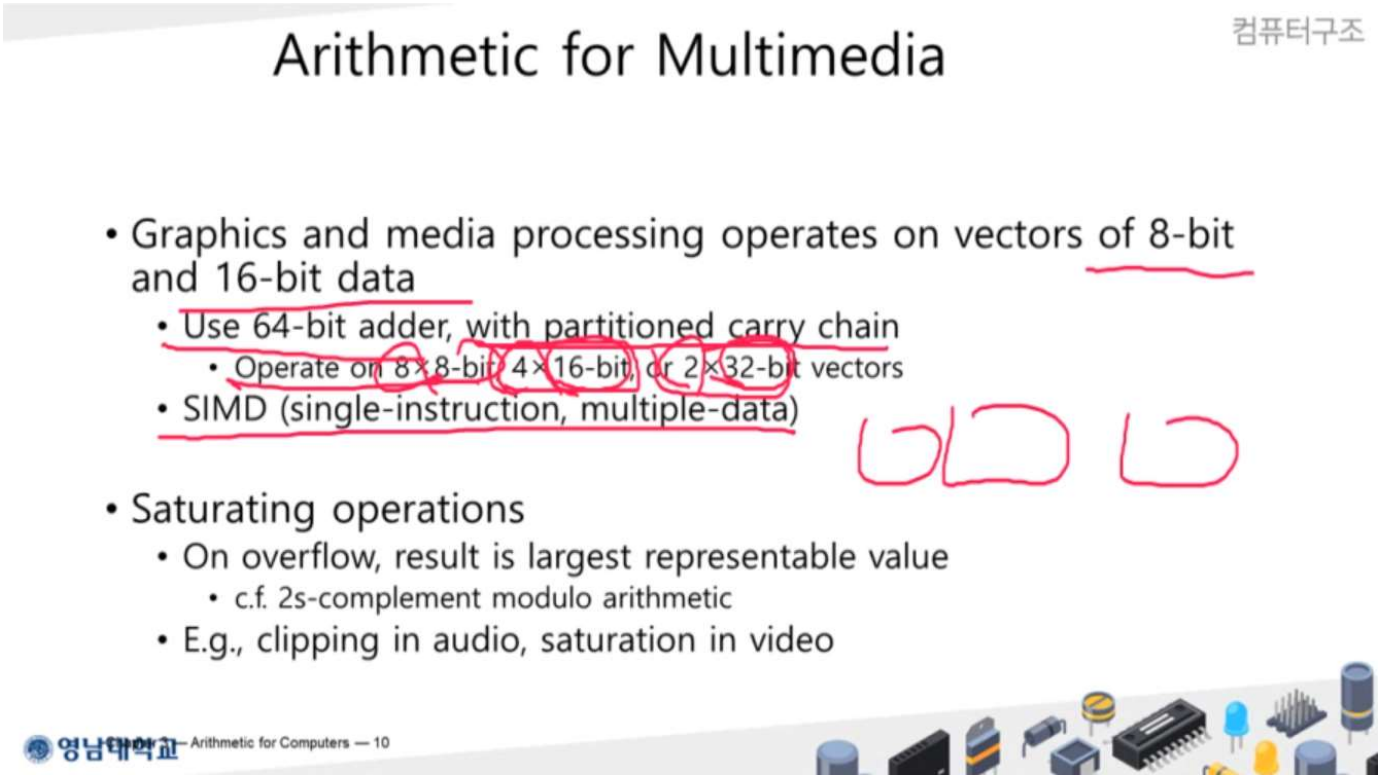
- 오버 플로우가 발생할때 : 두 양수를 더한 값이 음수가 될때 or 두 음수를 더했는데 합이 양수가 될때
  = 32비트 수 2개를 더하거나 뺀 결과를 완벽하게 표현하기 위해서는 33비트가 필요할 경우가 있다. 워드 크기가 32비트 이므로 33번째 비트는 표시할 수 없는데, 이렇게 되면 부호비트가 결과의 부호가 아니라 크기를 나타내는 비트 중 최상의 비트값으로 결정되기때문에 딱 한 비트가 부족하므로 틀릴 수 있는 것은 부호 비트 뿐이다.

```
  1111
+ 0001
------
 10000
```

첫 번째, `이진수 1111과 0001의 더하기 연산`이다. 결과 값은 `10000`으로 쉽게 계산할 수 있다. 그러나 4 bit 연산인데 결과 값은 5 bit로 **표현할 수 있는 범위를 넘겼다.** 이는 오버플로우가 발생한 것이므로 레지스터의 상태는 다음과 같이 예측할 수 있다.

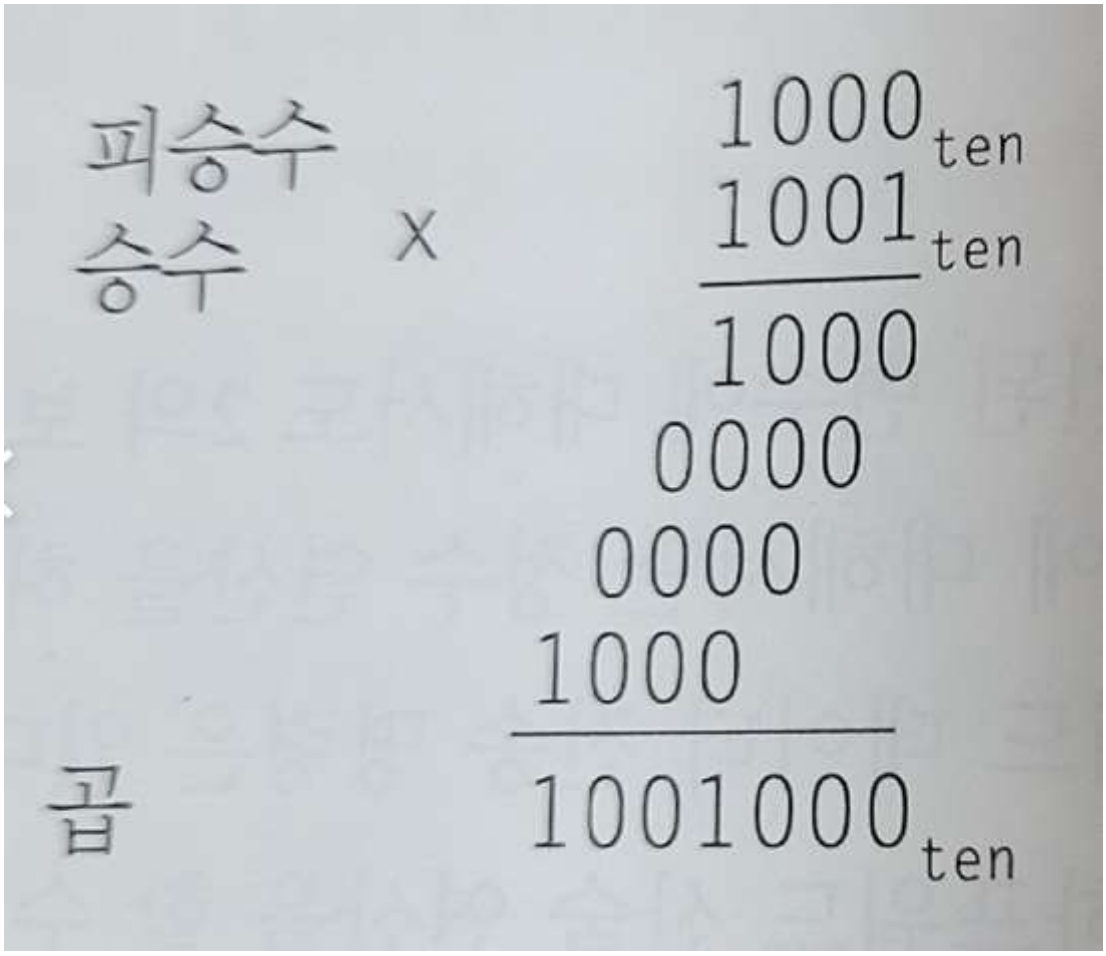


- C언어에서는 오버플로우가 발생하지않는다.
- Ada, Fortran 등등 다른언어는 exception을 발생시킨다.



- SIMD : SIMD(Single Instruction Multiple Data)는 병렬 컴퓨팅의 한 종류로, 하나의 명령어로 여러 개의 값을 동시에 계산하는 방식이다.
- Saturating operations : 오버플로우가 발생했을때 그 값을 최대의 값으로 주는것.
  ex)0xFF x 0xFF x 0xFF = 0xFF 흰색 x 흰색 x 흰색 = 흰색

곱셈

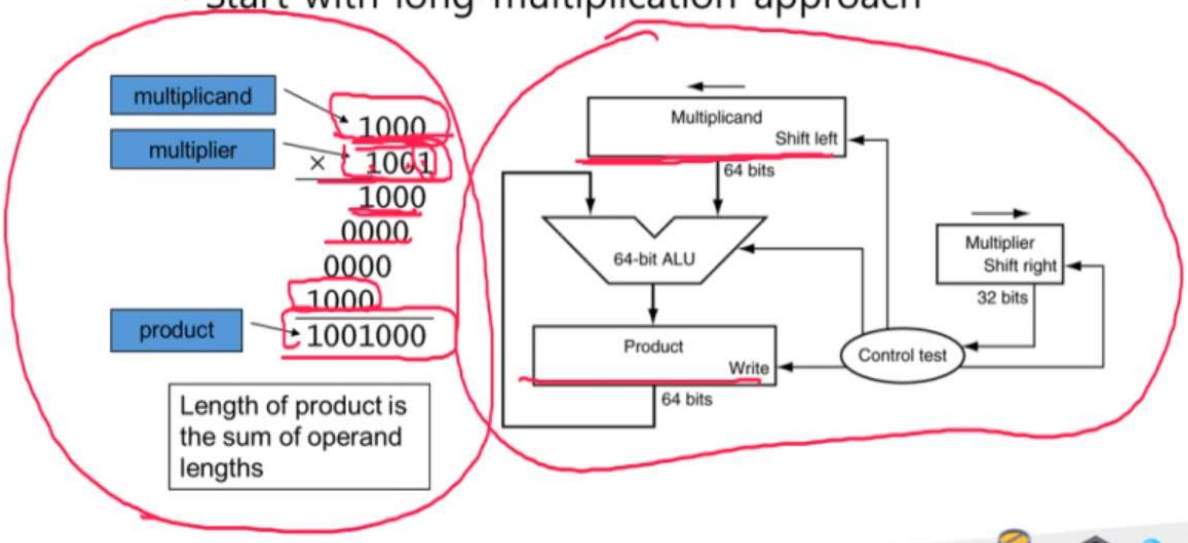


- 피승수 : 첫번째 피연산자.
- 승수 : 두번째 피연산자.

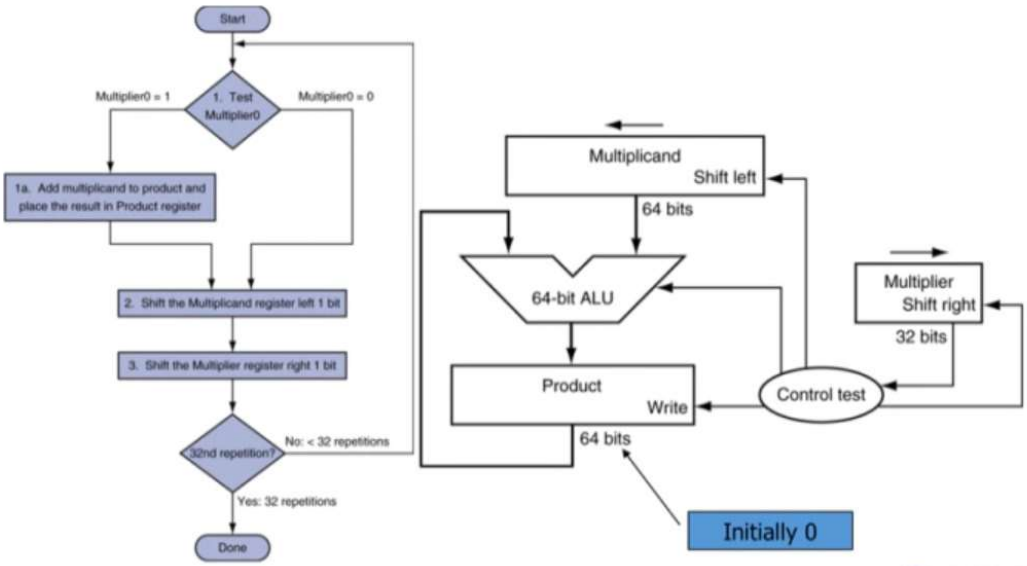
1. 승수의 자릿수가 1이면 피승수(1 X 피승수)를 해당 위치에 복사한다.
2. 승수의 자릿수가 0이면 0(0 X 피승수)을 해당 위치에 복사한다.

Multiplication

- Start with long-multiplication approach

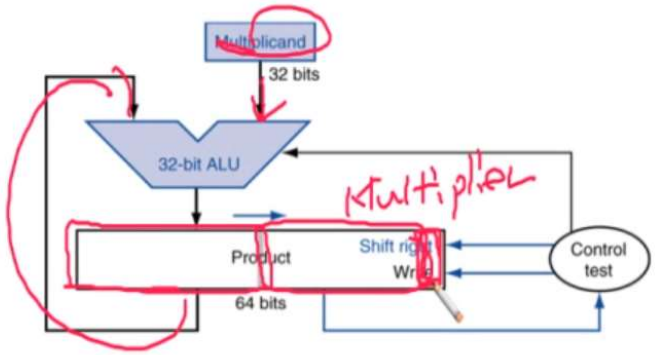


Multiplication Hardware



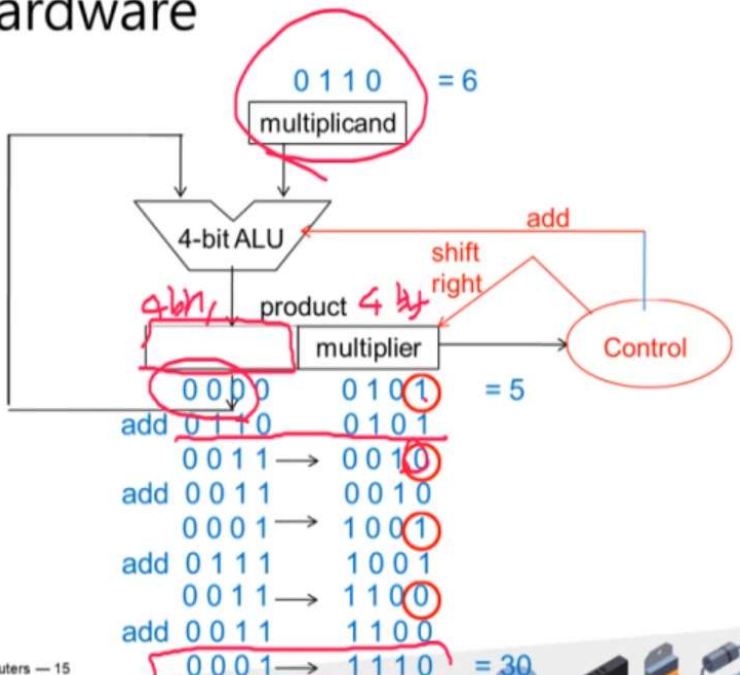
Optimized Multiplier

- Perform steps in parallel: add/shift



- One cycle per partial-product addition
  - That's ok, if frequency of multiplications is low

Add and Right Shift Multiplier Hardware

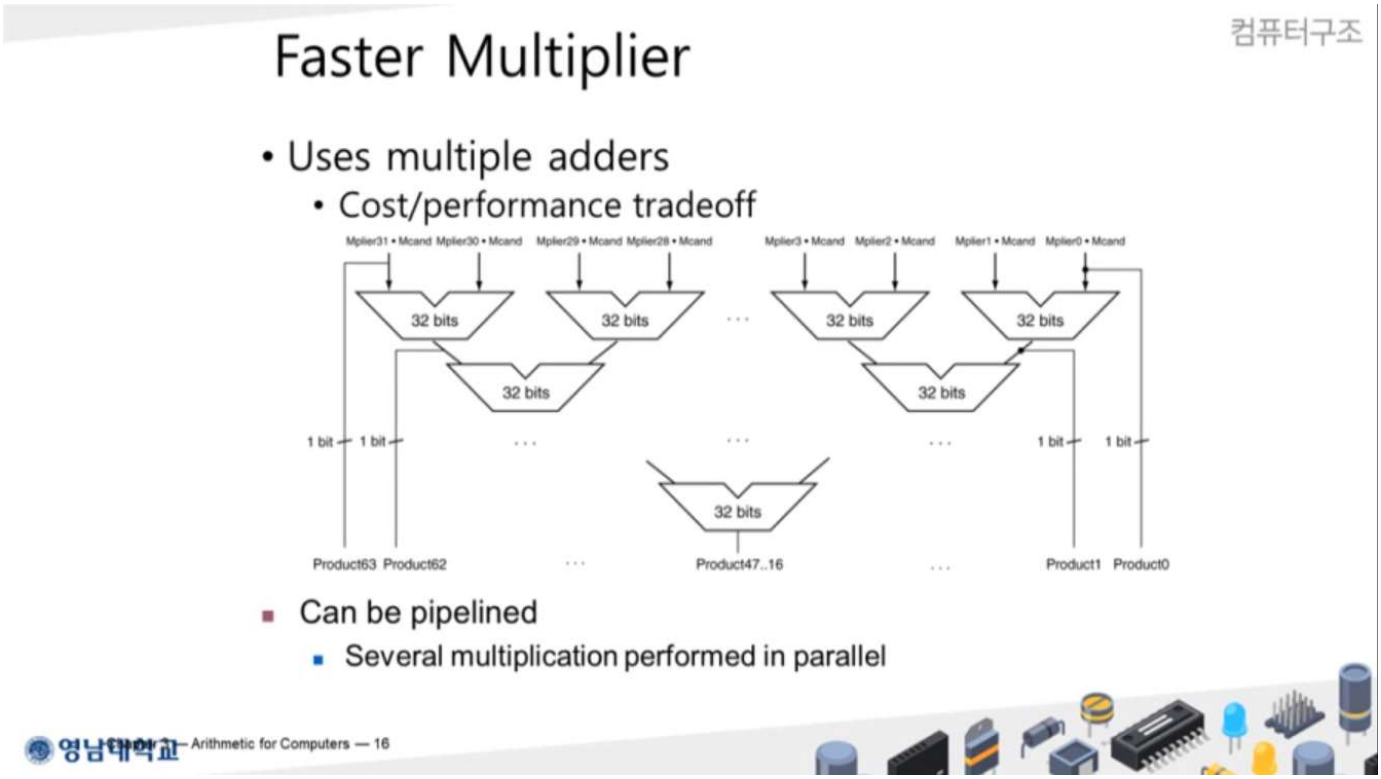


- 6과 5의 곱셈 방법
- 끝자리가 0이면 동작안하고 오른쪽으로 쉬프트, 끝자리가 1이면 더한다.

더 빠른 곱셈

- 승수의 매 비트마다 32비트 덧셈기를 하나씩 할당하면 더 빠른 곱셈이 가능하다.
- 64층의 덧셈기 스택을 만든다. (32비트 덧셈기를 다른 32비트 덧셈기에 더한다)

- 병렬 트리 구조로 만든다.



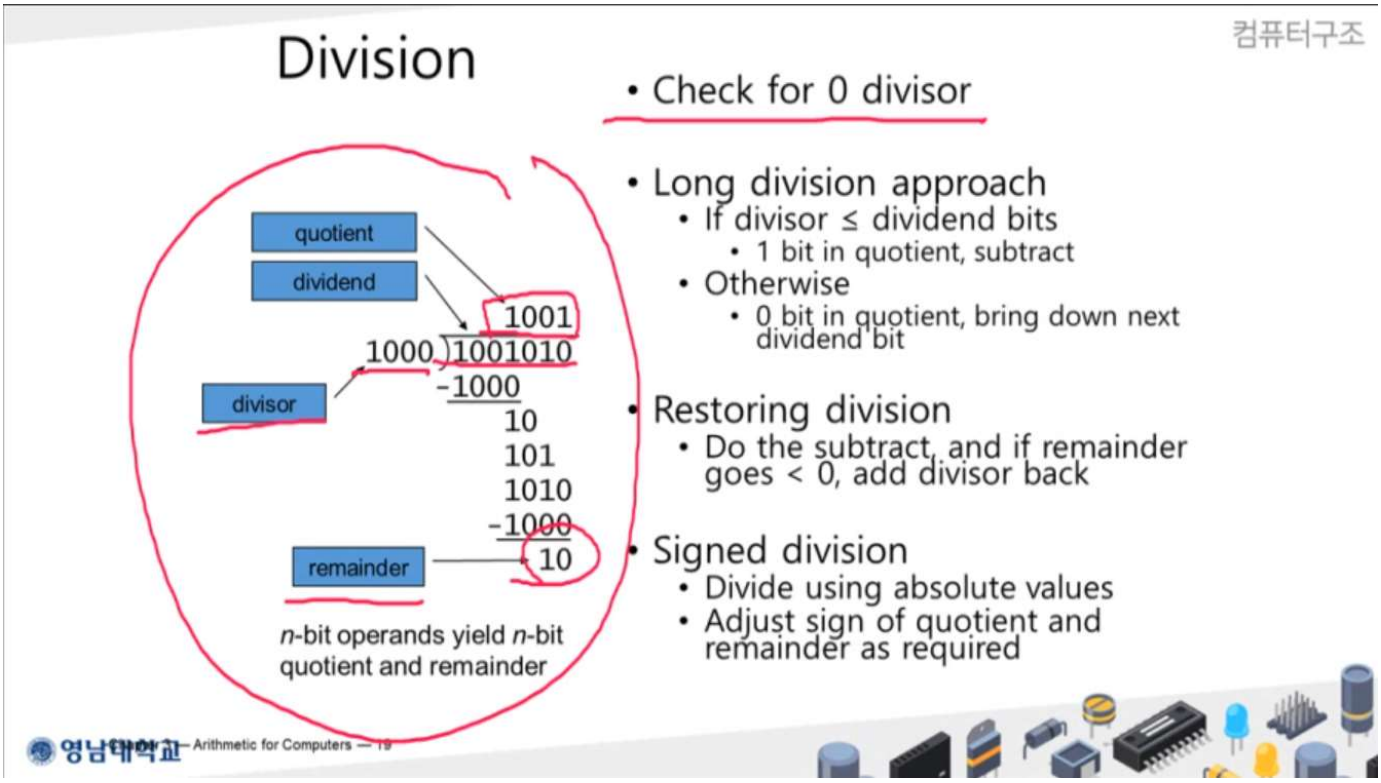
- 가격은 비싸지지만 성능은 높다.

곱셈 요약

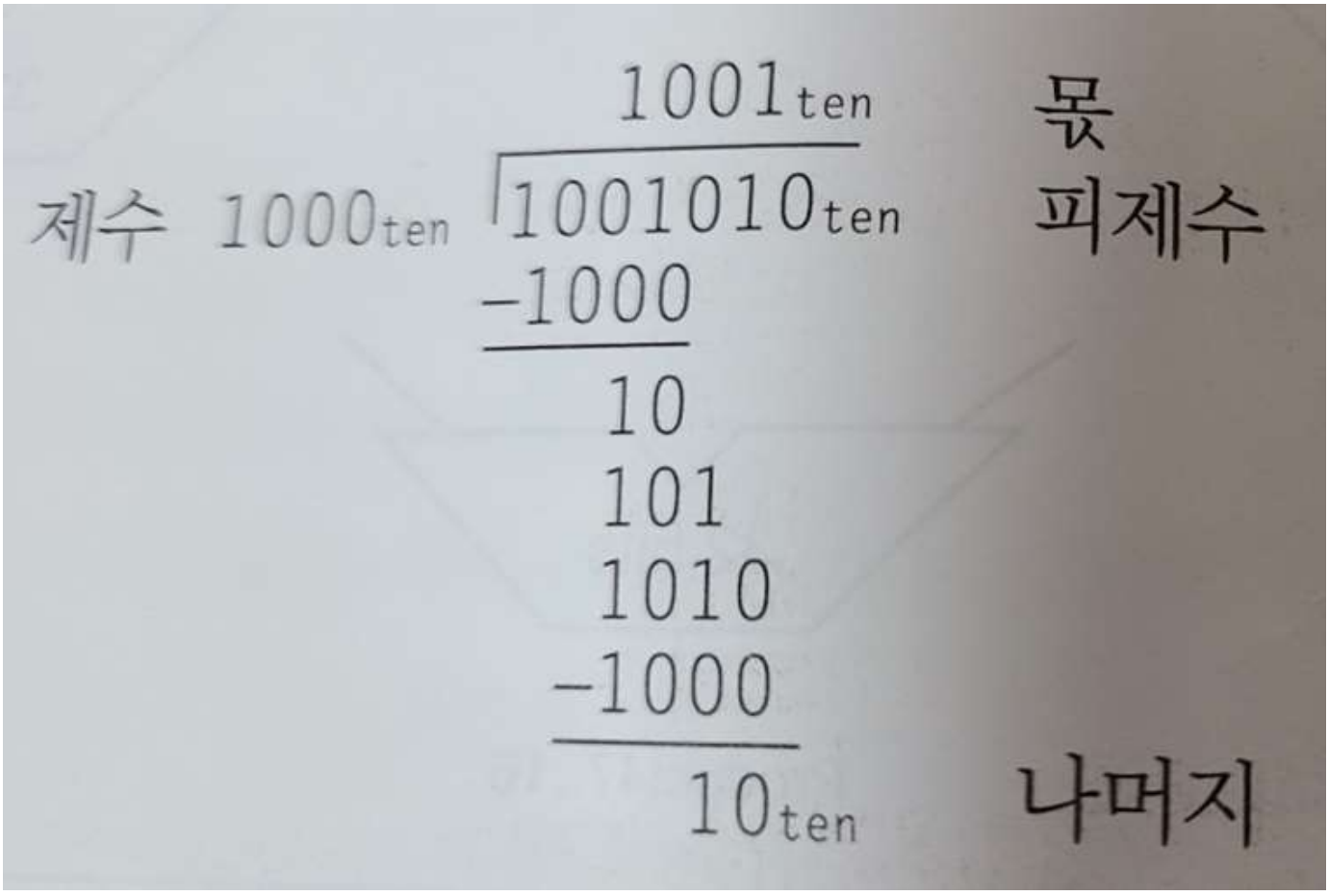
- 곱셈 하드웨어도 단순히 자리이동과 덧셈을 수행한다. 컴파일러는 2의 ''멱수 곱하기를 자리이동 명령어로 대치하기도 한다. 더 많은 하드웨어를 사용하면 덧셈을 병렬로 더 빠르게 처리할 수도 있다.

''멱수 : 멱이 되는 수. (멱수 : 거듭 제곱으로 된 수)
''거듭제곱 : 똑같은 수나 문자를 여러 번 곱한 것.

나눗셈

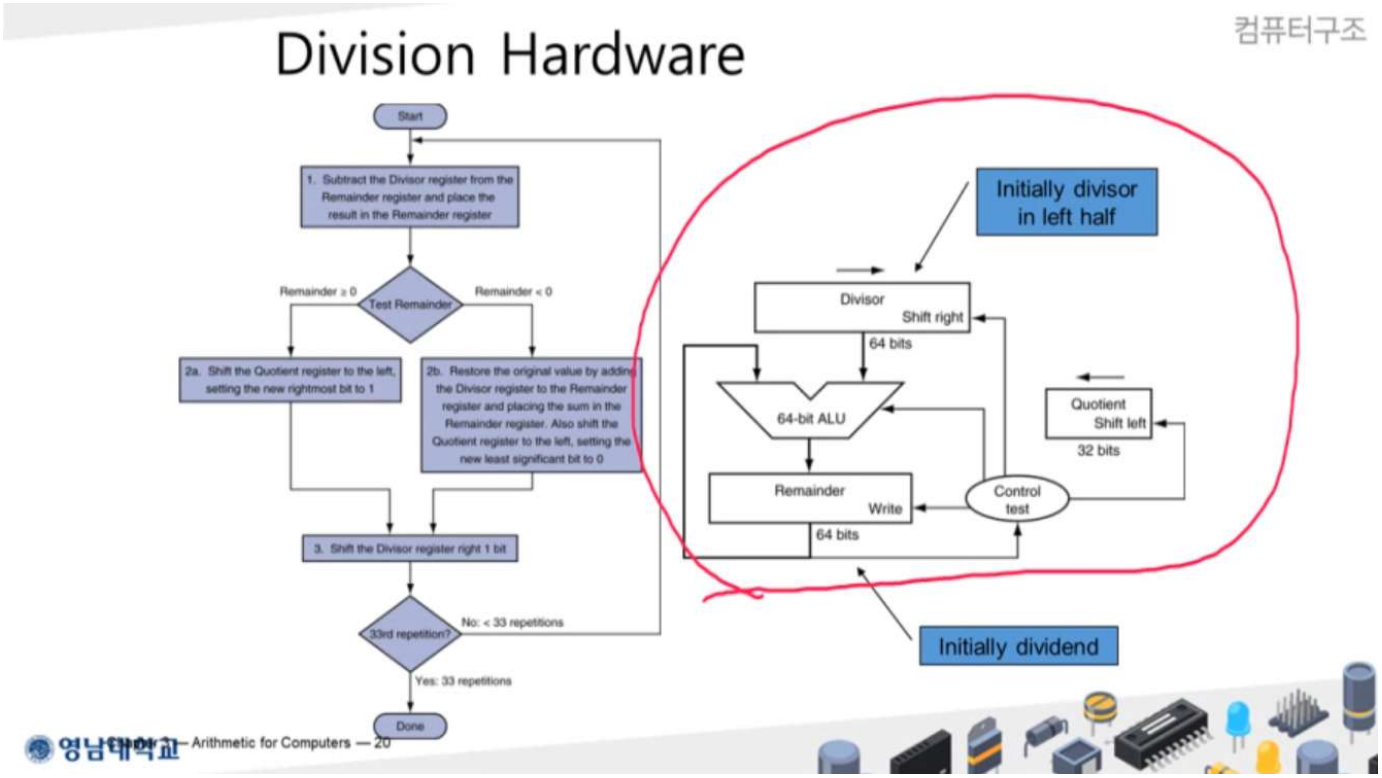


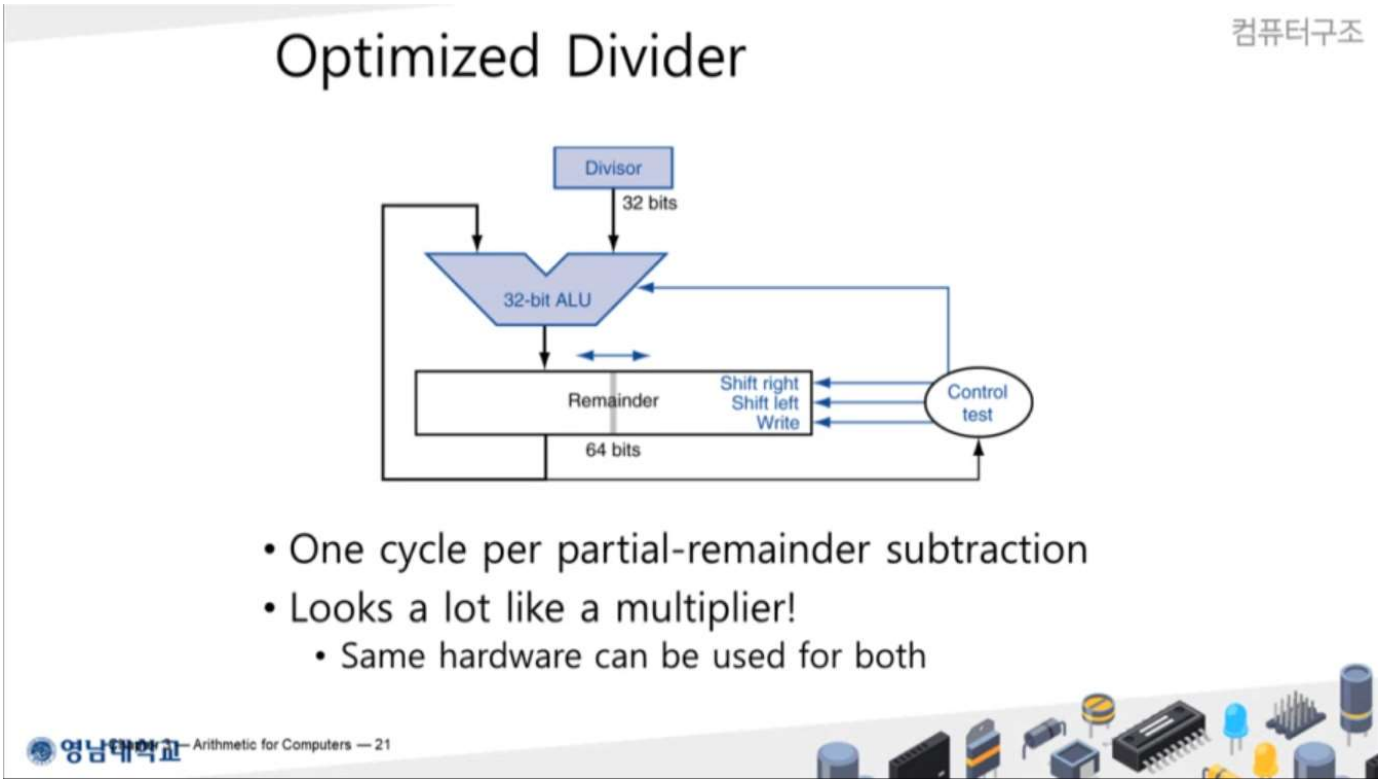
- 분모가 0인지 아닌지 먼저 체크한다.

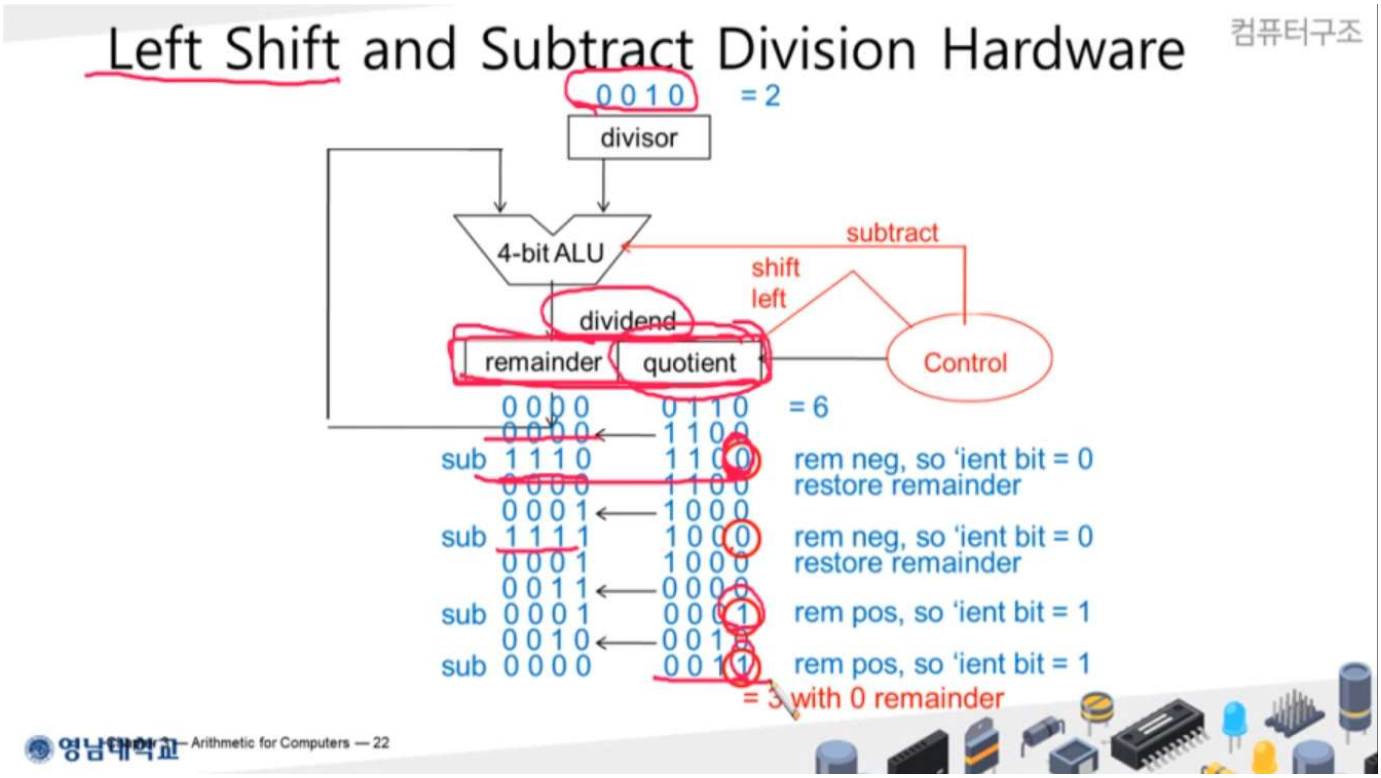


- 피제수 : 나누어 지는 수

- 제수 : 피제수를 나누는 수
- 몫 : 나눗셈의 가장 주된 결과. 제수에 곱하고 나머지를 더하면 피제수가 되는 수.
- 나머지 : 나눗셈의 두 번째 결과. 몫과 제수의 곱에 더하면 피제수가 되는 수.
- **피제수 = 몫 X 제수 + 나머지**



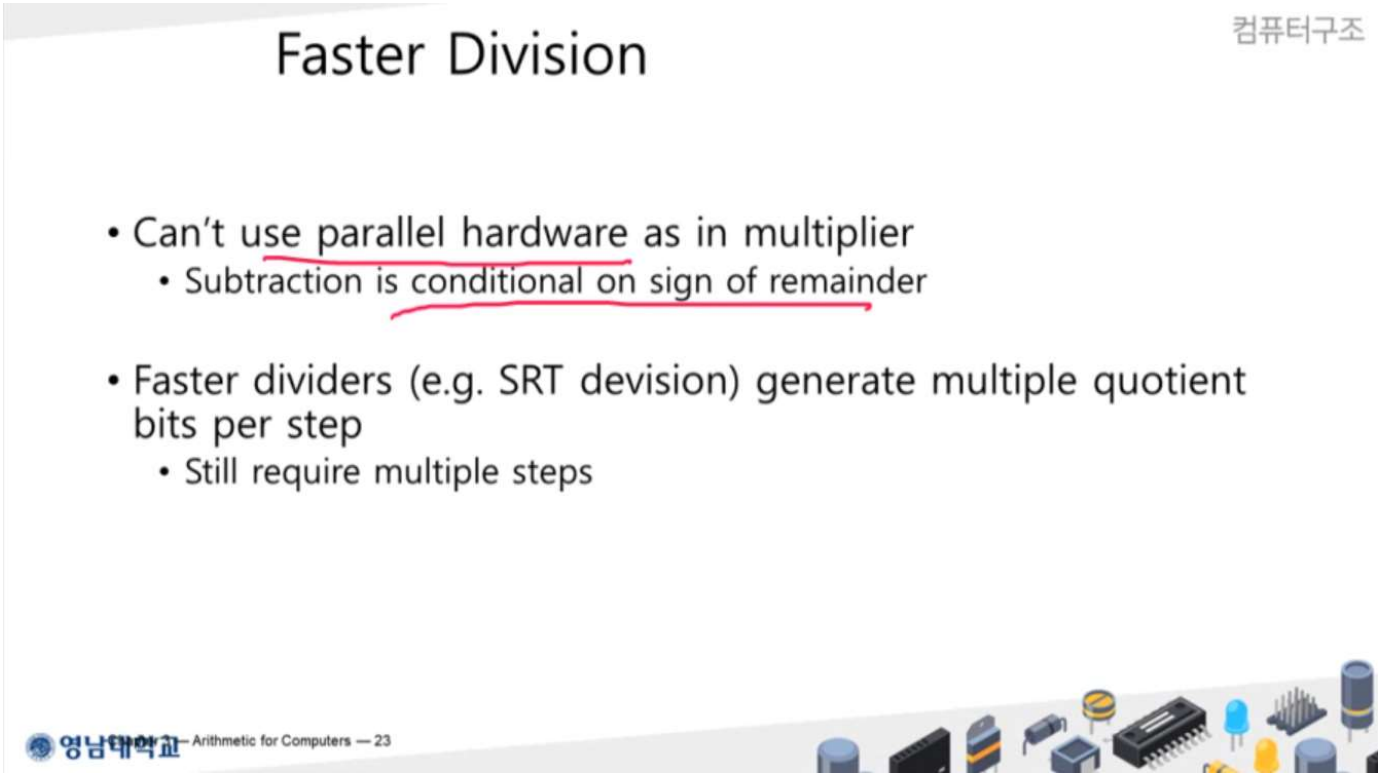




- 왼쪽으로 1비트움직이고 뺀다.
- 음수면 가장 오른쪽의 비트를 복원한다.

더 빠른 나눗셈

- SRT 나눗셈이라는 기술을 이용한다. SRT나눗셈은 여러개의 몫 비트를 예측한다. 피제수와 나머지의 상위 비트들을 이용하여 표를 찾아서 몫을 추측하고, 틀린 추측은 그 후의 단계에서 바로잡는다.



- 병렬화 할수없다 : 항상 remainder 값이 음수인지 양수인지 체크하기 때문

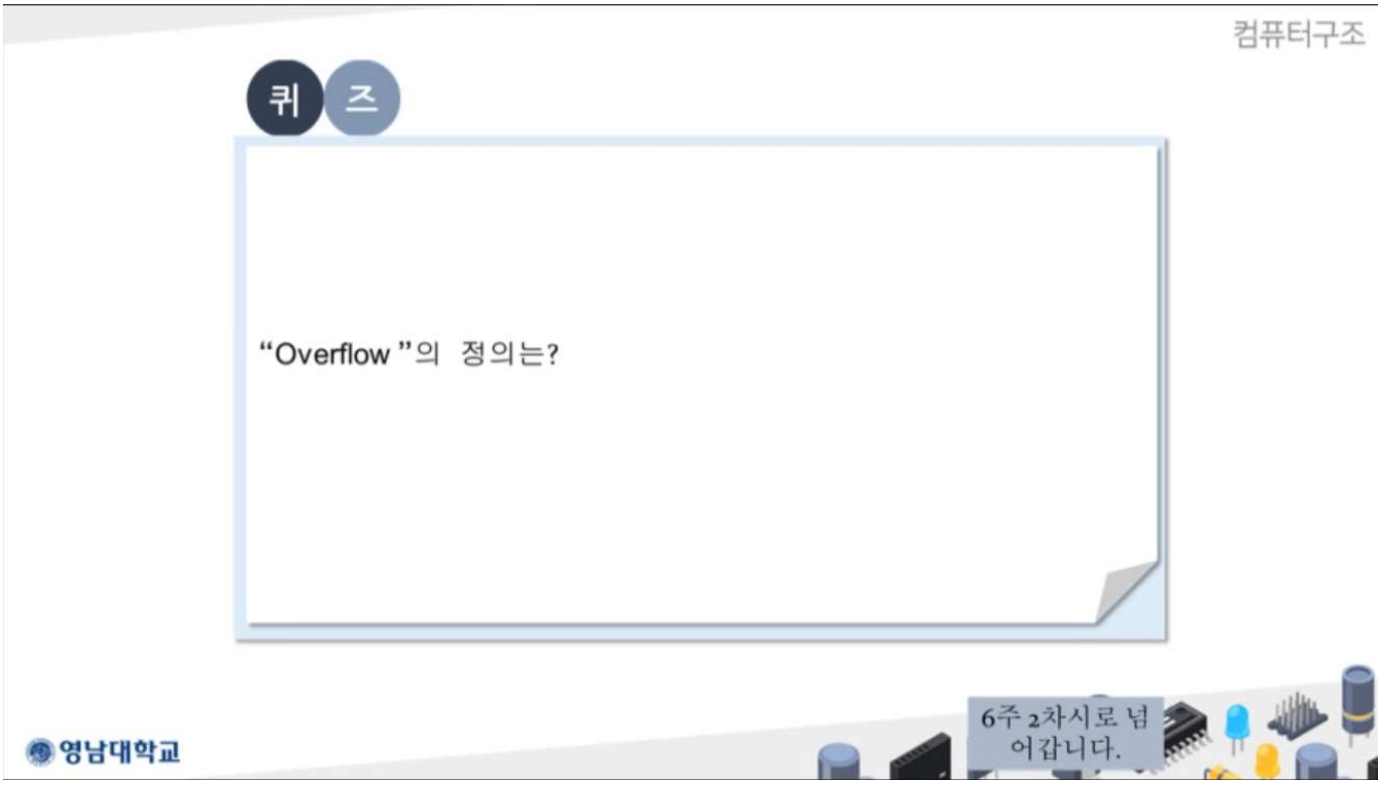
나눗셈 요약

- 여러 몫 비트를 동시에 예측하고 예측이 틀렸으면 나중에 바로잡는 방법으로 나눗셈을 빠르게 한다.

부동 소수점

- 프로그래밍 언어는 부호있는 정수와 부호없는 정수 뿐만 아니라 소수 부분을 갖는 수도 다룰 수 있어야 한다. 수학에서ㅇ는 이러한 수를 실수 라고 부른다.
- 실수의 예) 3.14159265, 2.71828m, 0.0000000000001 or 1.0 X 10^-9 등등..
- 과학적 표기법 : 소수점의 왼쪽에는 한 자릿수만이 나타나게 하는 표기법
  ex) 1.0 X 10^-9
- 실수를 정규화된 형태의 표준 과학적 표기법으로 나타내면 세가지 장점이 있다.

1. 부동 소수점 숫자를 포함한 자료의 교환을 간단하게 한다.
2. 숫자가 항상 이런 형태로 표현된다는 것을 알고 있으므로 부동 소수점 산술 알고리즘이 간단해진다.
3. 불필요하게 선행되는 0을 소수점 오른쪽에 있는 실제의 숫자로 바꾸기 때문에 한 워드 내에 저장할 수 있는 수의 정밀도를 증가시킨다.



- 오버플로우 : 어떤 결과값이 범위를 벗어나는 것.

병렬성과 컴퓨터 연산: 서브워드 병렬성

실례: x86의 SSE와 AVX

더 빠르게: 서브워드 병렬성과 행렬 곱셈

오류 및 함정

결론

역사적 고찰 및 참고문헌

자습

연습문제

도로로

필승

이전 포스트

컴퓨터 구조 - Computer Abstractions and Technology

0개의 댓글

댓글을 작성하세요

댓글 작성