나눗셈 알고리즘

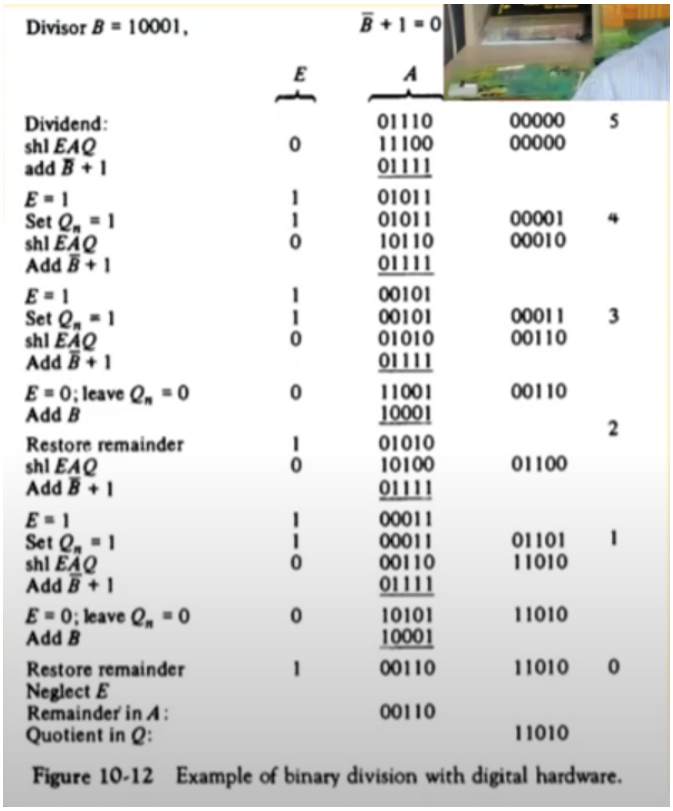
이진 나눗셈

- 하드웨어는 곱셈과 동일
  - A레지스터 Q레지스터를 합쳐서 나눗셈의 피젯수가 들어가고 동시에 결과가 들어감
  - B레지스터에는 나누기가 들어가고 그 나누기의 개수만큼 SC(Sequence counter)가 세팅된다.
- 나눗셈 오버플로우의 처리

나눗셈의 처리

Figure 10-11 Example of binary division.

Divisor:		
$B = 10001$	11010	Quotient = Q
	)0111000000	Dividend = A
	01110	5 bits of $A < B$, quotient has 5 bits
	011100	6 bits of $A \geq B$
	-10001	Shift right B and subtract; enter 1 in Q
	-010110	7 bits of remainder $\geq B$
	--10001	Shift right B and subtract; enter 1 in Q
	--001010	Remainder $< B$; enter 0 in Q; shift right B
	---010100	Remainder $\geq B$
	----10001	Shift right B and subtract; enter 1 in Q
	----000110	Remainder $< B$; enter 0 in Q
	-----00110	Final remainder



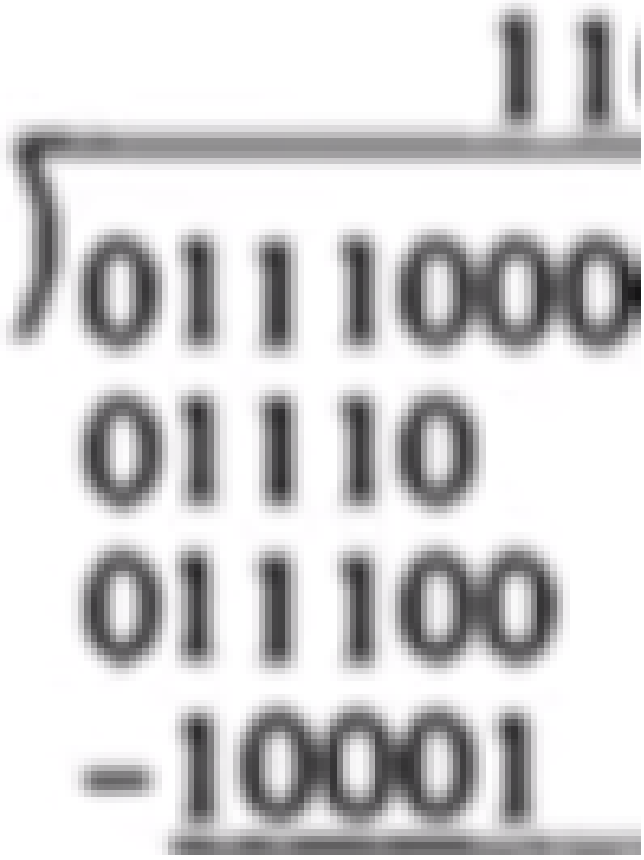
Divisor $B = 10001$, $\overline{B} + 1 = 01111$

	E	A	Q	
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		01111		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		10001		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Figure 10-12 Example of binary division with digital hardware.

나눗셈은 뭐냐?? 계속 빼는거다

한정된 비트의 레지스터를 가지고 연산하면 문제가 된다 -> 만약 나누고자하는 피젯수 젯수를 빼가는 과정 예를 들어보자
젯수가 10001 이고 피젯수가 0111000000이다.
그럼 여기서 피젯수에서 젯수를 빼나가기 시작하는데 01110에서 10001을 빼는데 이러면 뭐가 문제냐 빼는수가 더 크다. 그럼 어쩌냐??? 다음 비트까지 포함하여 빼준다. 아래 그림처럼 그래서 5비트짜리를 연산하는데 6비트짜리가 사용된다. 만약 ab가 16비트인데 a가 b보다 작으면 17비트짜리를 사용해야한다. 연산이 될리가 없다. 이런 경우가 오버플로우다.



요약
16비트를 16비트로 나누는데 나누는게 나눠지는거보다 커지면 다음자리수를 가져와야한다. 그럼 17비트가 되버리니까 연산이 불가능해서 오버플로우가 발생한다.

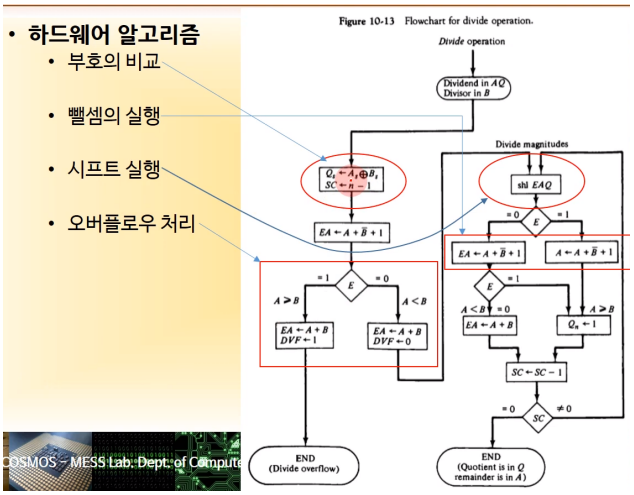
오른쪽 사진 설명

- 처음에 A레지스터보다 B 나누는값(젯수)가 더 크므로 쉬프트 시키고 빼준다. 4번째 줄이 뺀 결과값 01011이고 E는 1이된다. 그 E를 Q로 넘겨주고 다시 시프트 시킨다. 그리고 다시 빼주고 이거를 반복
- 마지막 결과(몫)는 Q에 나온다 11010
- 그리고 오버플로우가 나게 되면 제일 처음에 오버플로우 때문에 비트수가 모자라서 처리가 안된다 하면 오버플로우 인터럽트를 내줘야한다.

하드웨어 알고리즘

- 부호의 비교

- 뺄셈의 실행
- 시프트 실행
- 오버플로우 처리



- 이거도 부호를 제일 처음에 비교해줘야한다. 그리고 SC를 세팅해준다
- 그리고 A에서 B값을 빼본다. 근데 E값이 1이 나왔다 즉 오버플로우다. 그러면 DVF를 1로 해주고 처리해준다.
- E가 0이다 그럼 오버플로우가 아니니까 A가 B보다 작으니까 문제가 없다. 이제 나눗셈 시작
- 이제 시프트 실행 E가 1이면 A에서 B를 뺀거를 A에 넣어주고
- E가 0이면 A에서 B를 뺀거를 EA에 넣어준다. 요기는 또 E가 1인 경우 0인 경우가 있는데 E가 1이면 Q를 넘기고 SC를 바꾸고 SC가 0이 아니면 다시 반복 연산이 끝나면 Q레지스터에 값이 모여서 몫이 남는다 나머지는 A레지스터에 남는다

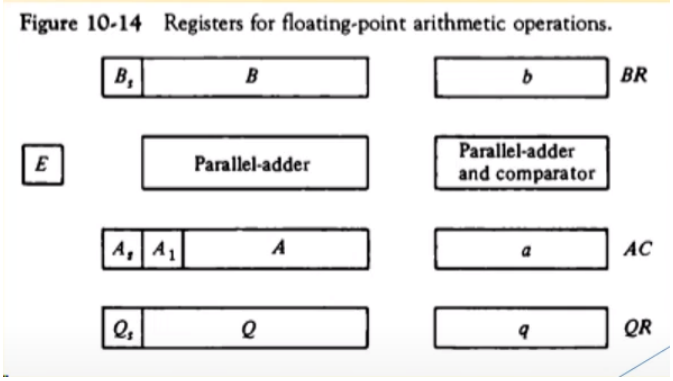
부동 소수점 산술 연산

숫자의 표현

- M * r^n r: Radix

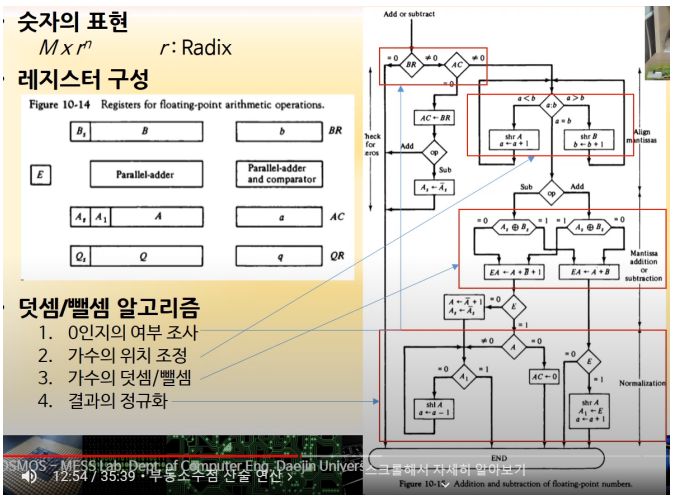
- 우리는 컴퓨터로 사용하니까 그냥 r은 2로 생각한다.
- M은 실수의 가수로 표현

레지스터 구성



- 부동 소수점을 표현하기 위해서 숫자와 부호로만 나타내는게 아니라 지수부분을 표현하는 레지스터(b)가 존재한다. 지수부분은 당연히 정수부분보다 길다

덧셈/뺄셈 알고리즘



- 덧셈에서 제일 처음하는거 지수부분이 0인지 아닌지 보는거
- 둘다 0이 아니다 그럼 a와 b의 크기 비교를 한다. 그리고 큰쪽으로 맞춰서 작은애들을 시프트 해준다. 그리고는 똑같이 덧셈뺄셈
- 대신 이거는 부동소수점이니까 마지막에 정규화를 해준다. 계산 이후에 소수점 부분만 남아야하는데 그렇지 않은 경우가 있다. 정수부분이 있으면 그

정수부분을 자리수를 밀고(시프트하고) 제일 왼쪽이 소수점이 되도록 맞춘다. 맞춘 값으로 정해지고 결과로 나온다.

곱셈 알고리즘

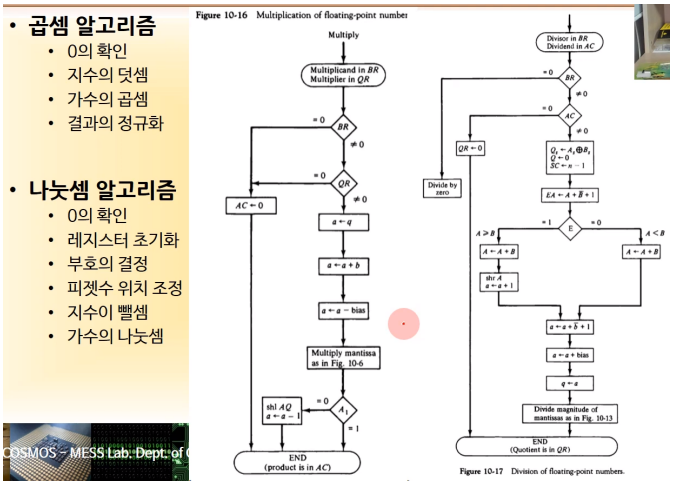
- 0의 확인
- 지수의 덧셈
- 가수의 곱셈
- 결과의 정규화
- 얘도 똑같다!! 지수의 곱셈결과가 정수부분을 넘어가면 정규화, 0.0xx 라면 맨왼쪽이 소수점이 되도록 시프트 해주고 그런 과정을 반복
- 지수부분을 더하고 mantissa를 곱해준다. 그리고 소수점이하의 값이 0이 아닐때까지 시프트해준다. 이걸 반복

나눗셈 알고리즘

- 0의 확인
- 레지스터 초기화
- 부호의 결정
- 피젯수 위치 조정
- 지수의 뺄셈
- 지수의 나눗셈
- 아래 그래프처럼 a에서 b를 빼고 a에 bias값을 더하는데

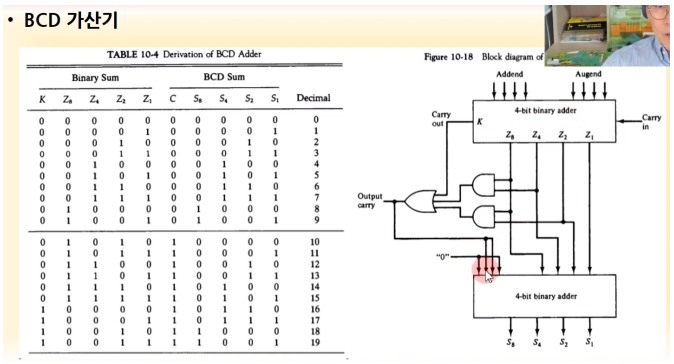
여기서 bias

bias는 기준이 되는 값을 말하는데
위에서 BR AC QR에는 부호 비트가 없다 그럼 2^(-3) 요거는 어케 표현하느냐???
그래서 bias를 사용한다 예를들어서 지수부분이 8비트라고 가정하면 표현가능한거는 2^8 = 256개 그럼 128개를 기준으로 하여서 128이 2^0 이 된다. 2^(-3)은 128에서 3을 빼줘서 125가 된다. 2^6은 128+6으로 134가 된다



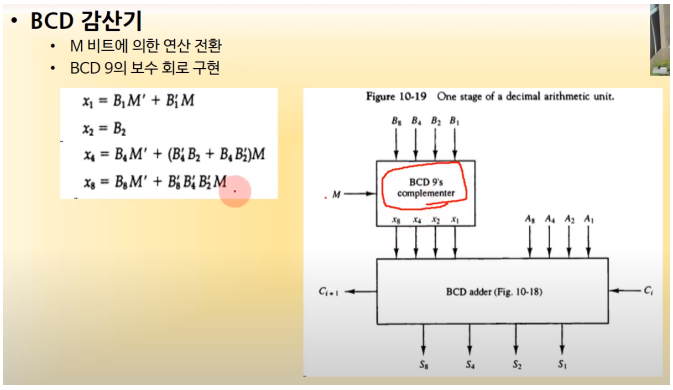
십진 산술 장치

BCD 가산기



- 4비트 binary 덧셈기에 6(0110)을 더하는거 이거 만든거는 논리회로 시간에 설명함
- 결과값이 0 ~ 9 는 그냥 두는데
- 10을 넘어간다? 그럼 BCD값은 0이되고 자리수가 1로 올라간다
- 캐리가 있다 그럼 그냥 6을 더한게 같아진다
- 캐리가 없다 근데 Z8이 1이고 Z4가 1인경우 그리고 Z8이 1 Z2가 1인경우 다 6을 더한경우란다.

BCD 감산기

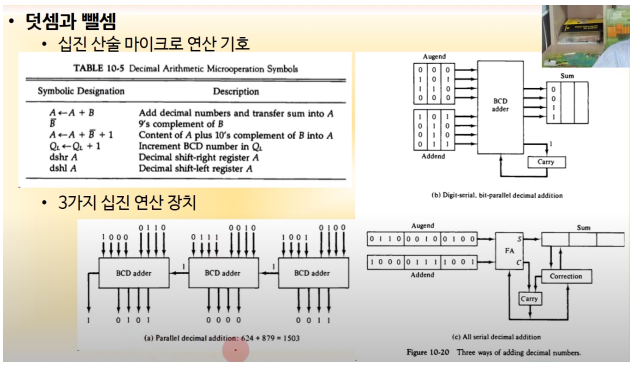


- M이 1이면 그냥 BCD에 9의 보수기를 넣으면된다.
- 9의 보수기는 왼쪽에 그림이다.

십진 산술 연산

덧셈과 뺄셈

- 십진 산술 마이크로 연산 기호



- 3가지 십진 연산 장치

> - Parrallel decimal addtion 요거는 각 자리수대로 하나씩 더하는거 캐리가 발생하면 앞단계로 넘겨준다. 임마는 한번만 연산, 빠르고 복잡
> - Digit-serial 요거는 시리얼로 하는건데 각각의 BCD값을 하나씩 자리수별로 그래서 캐리가 발생하면 다시 임마는 3번 연산
> - All serial decimal addtion 요거는 한비트 한비트 비트by비트 연산 한비트 연산해서 sum에 넣고 캐리 발생하면 넘겨주고 임마는 12번 연산, 느리고 간단

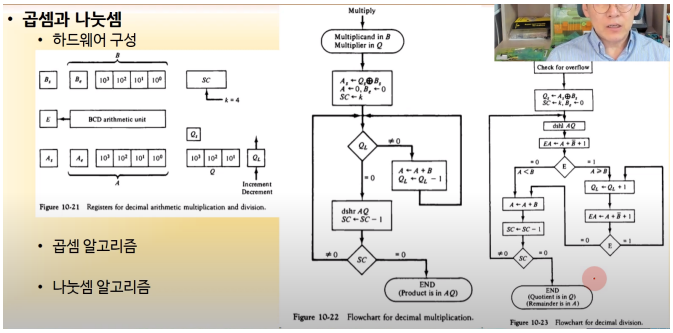


Figure 10-21 Registers for decimal arithmetic multiplication and division.

Figure 10-22 Flowchart for decimal multiplication.

Figure 10-23 Flowchart for decimal division.

- BCD 연산장치를 사용하는거 말고는 다른게 없다

기억할거는 십진 산술 연산에서는 BCD 사용한다는거