3. Arithmetic

3.2 Addition and subtraction

Integer Addition

오버플로우가 언제 발생한가?

-> 덧셈은 부호가 다른 피연사자를 연산할경우에는 오버플로우X

양수+양수=음수(result sign is 1) (오버플로우), 음수+음수-=양수(result sign is 0) (오버플로우)

-> 덧셈은 부호가 같은 피연사자를 연산할경우에는 오버플로우X

양수-음수=음수(오버플로우), 음수-양수=양수(오버플로우)

C언어는 오버플로를 무시하기 때문에 MIPS C 컴파일러는 변수의 형에 관계 없이 언제나 부호없는 버전인 addu, addiu, subu 명령어를 생성한다. 그러나 MIPS Fortran 컴파일러는 피연산자의 형에 따라서 적절한 산술 명령어 생성

MIPS에서는 오버플로를 탐지하면 exception(interrupt)을 발생.(add, addi, sub)

EPC(exception program counter)레지스터가 인터럽트가 걸린 명령어의 주소 기억

mfc0(move from system control) 명령어는 EPC를 범용레지스터에 복사하여, MIPS가 jump 명령어를 통해 인터럽트가 걸린 명령어로 되돌아갈 수 있게 해 준다.

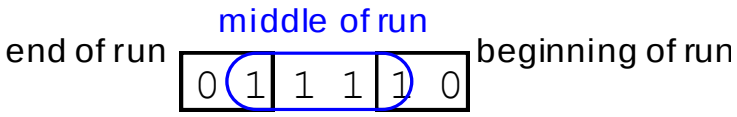
MIPS는 full word에 대해서만 정수 연산을 허용한다.

Arithmetic for Multimedia

Graphics and media processing operates on vectors of 8-bit and 16-bit data

Saturating operation -> 오버플로우 발생기 2의 보수 산술이 모듈로 연산을 하는 것과는 달리 포화는 가장 큰 양수나 음수를 결과값으로 한다. 포화 연산은 미디어 연산에 사용.

Booth's Algorithm –ppt 14-16



Current Bit	Bit to the Right	Explanation	ExampleOp	
1	0	Begins run of 1s	0001111000	sub
1	1	Middle of run of 1s	0001111000	none
0	1	End of run of 1s	0001111000	add
0	0	Middle of run of 0s	0001111000	none

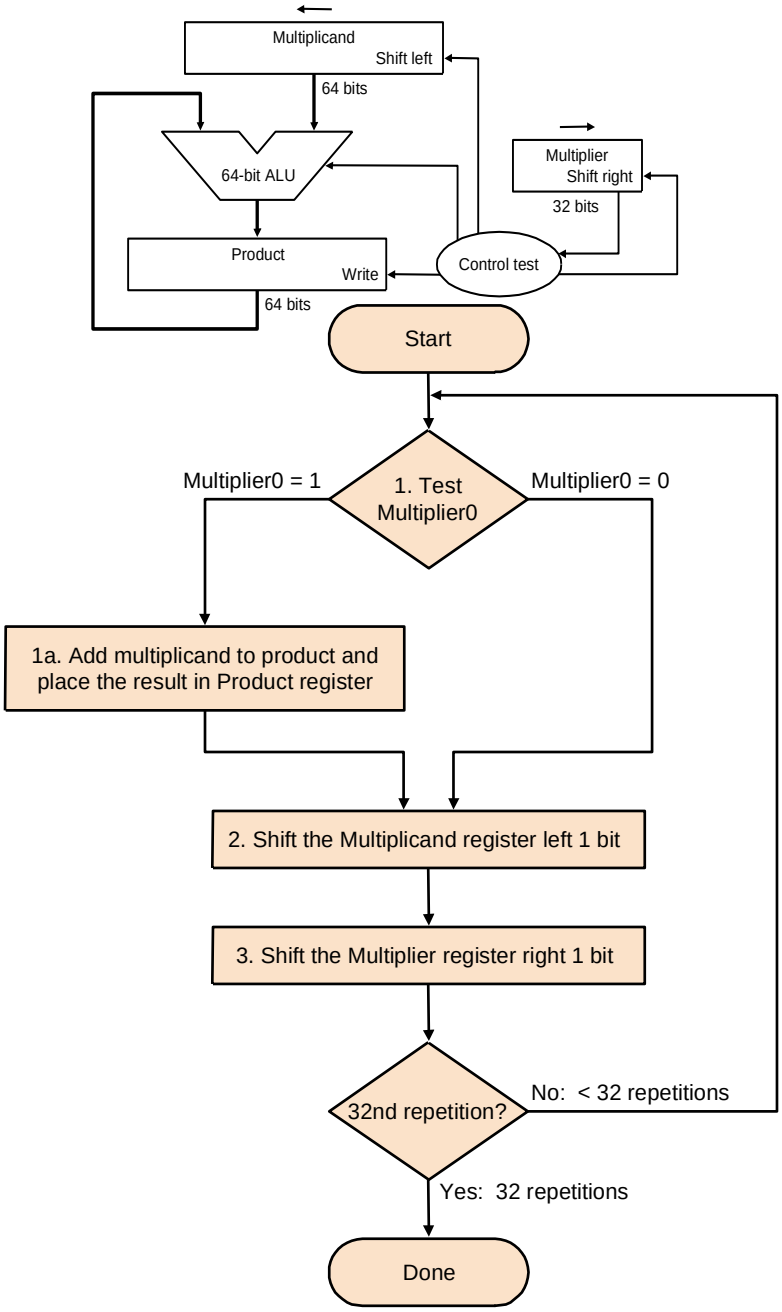
Operation	Multiplicand	Product	next?
0. initial value	0010	0000 0111 0	10 → sub
1a. P = P - m	1110	+ 1110	
		1110 0111 0	shift P (sign ext)
1b.	0010	1111 0011 1	11 → nop, shift
2.	0010	1111 1001 1	11 → nop, shift
3.	0010	1111 1100 1	01 → add
4a.	0010	+ 0010	
		0001 1100 1	shift
4b.	0010	0000 1110 0	done

Operation	Multiplicand	Product	next?
0. initial value	0010	0000 1101 0	10 → sub
1a. P = P - m	1110	+ 1110	
		1110 1101 0	shift P (sign ext)
1b.	0010	1111 0110 1	01 → add
		+ 0010	
2a.		0001 0110 1	shift P
2b.	0010	0000 1011 0	10 → sub
		+ 1110	
3a.	0010	1110 1011 0	shift
3b. 0010		1111 0101 1	11 → nop
4a		1111 0101 1	shift
4b.	0010	1111 1010 1	done

Multiplication

덧,뺏셈보다 더 많은 시간, 공간 필요

multiplicand(첫번째 피연산자), multiplier(두번째 피연산자)

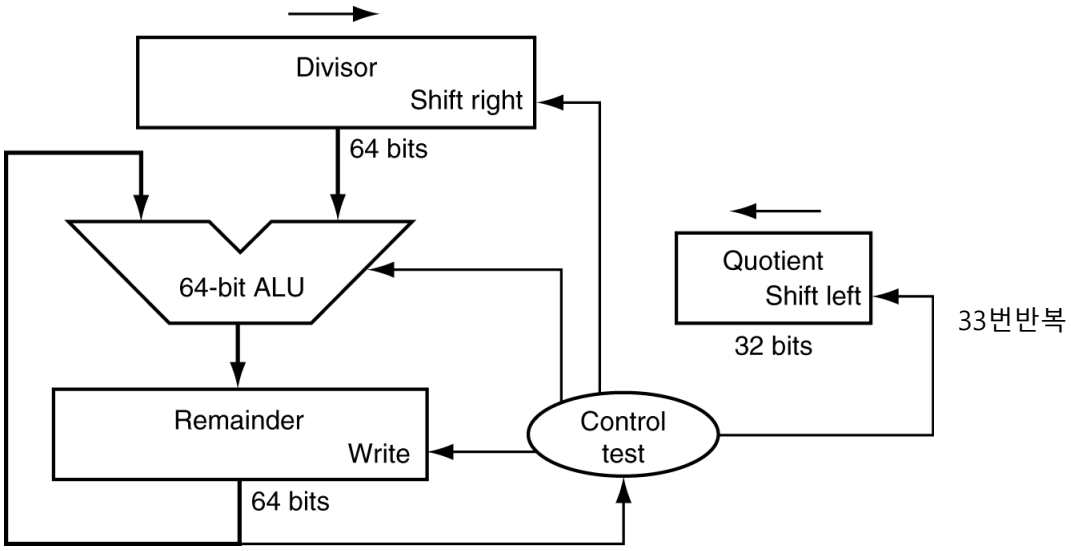


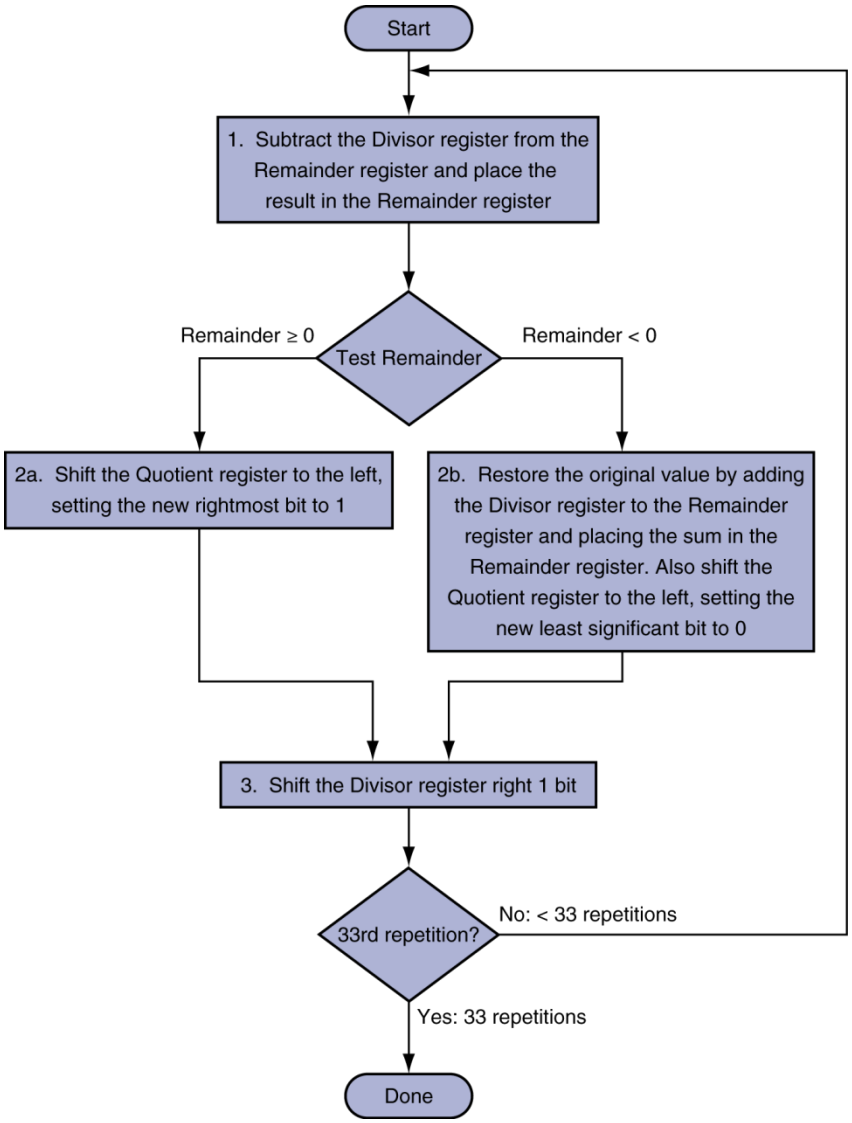
곱 레지스터는 0으로 초기화 32번반복

부호있는 수는 먼저 승수와 피승수를 양수로 변환하고 원래의 부호를 기억한다. 이 알고리즘은 부호를 계산에서 제외하고 31번 반복한후 나중에 계산해준다.

Observations on Multiply Version 1, Final Version -> p9-p12

Divison





Optimized Divider –슬라이드 21쪽, 책 페이지 187

부호있는 나눗셈 -> 두 피연사자의 부호가 다를 경우에는 몫의 부호를 음수화하고, 나머지가 0이 아니라면 그 부호는 피제수(dividend)의 부호에 따른다.

Faster Division

곰셈처럼 multipel을 사용x -> 알고리즘 다음 단계 수행하기 전에 뺄셈한 결과의 부호를 알아야 하기 때문이다. -> 그래서 각 단계에서 여러 개의 몫 비트를 예측하는 기법 사용

MIPS Division

Use **HI/LO** registers for result, Use **mfhi, mflo** to access result

->HI: 32-bit remainder, LO: 32-bit quotient

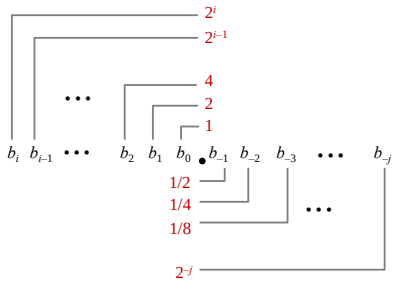
div rs, rt / *divu rs, rt* div는 부후있는 정수, divu는 부호없는 정수

MIPS의 나눗셈 명령은 오버플로를 무시하므로, 몫이 너무 커서 오버플로가 발생하는 소프트웨어로 검사해야함. 0으로 나누는것도 마찬가지이다.

3.5 부동소수점

Fraction(소수부분) Binary Numbers

Fractional Binary Numbers



Convert 13 to binary in the ordinary way.

2	13	R=1
2	6	R=0
2	3	R=1
2	1	R=1 ↑read↑
	0	

Therefore, $13_{10} = 1101_2$

Convert the decimal part in the following manner:

Read Down **INTEGER PART**

.6875 x 2 = **1**.375	1
.375 x 2 = **0**.75	0
.75 x 2 = **1**.50	1
.5 x 2 = **1**.00	1

Therefore, $.6875_{10} = .1011_2$

Hence,
$13.6875_{10} = 1101.1011_2$

Floating Point

scientific notation : 소수점의 왼쪽에는 한 자리 수만이 나타나게 하는 표현법

normalized number : 선행하는 0이 없는 부동소수점 표기법으로 나타낸 수

실수를 정규화(normalized)한 형태의 scientific notation으로 표현하면 좋은점 3가지

- 부동소수점 숫자를 포함한 자료의 교환 간단하게 함
- 부동소수점 산술 알고리즘 간단해짐
- 불필요하게 선행되는 0을 소수점 오른쪽에 있는 실제의 숫자로 바꾸기 때문에 한 워드 내에 저장할 수 있는 수의 정밀도 증가

In binary : ±1.xxxxxxx2 × 2yyyy

Floating Point Standard

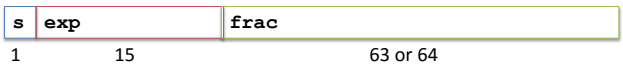
Two representations : Single precision (32-bit), Double precision (64-bit)

precision: 정밀도

- Single precision: 32 bits

- Double precision: 64 bits

- Extended precision: 80 bits (Intel only)



IEEE Floating-Point Format

S(부호)	Exponent(지수)	Fraction(소수)

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

S: 1이면 음수, 0이면 양수

E: single(8bit), double(11bit)

F: single(23bit), double(52bit)

overflow : 양수 값을 갖는 지수가 지수 부분에 표현될 수 없을 만큼 큰 상황

underflow : 음수 지수의 절댓값이 너무 커서 지수 부분에 표현될 수 없는 상황

IEEE 표준은 정규화된 이진수의 가장 앞쪽 1비트를 생략하고 표현하지 않는다.(0제외)

실제 significand(유효자리) -> single 24(23+1), double 53(52+1)

Bias => 지수를 유효자리 앞에 두면 부호가 같은 수를 비교할 때 지수가 큰 수가 작은수보다

큰 정수처럼 보임. 음수 지수의 숫자 정렬을 편하게 하기 위해서 가장 음수인 지수를 000..00(two)로 가장 양수인 지수를 11....11(two)로 표현 -> 각자 –무한대, +무한대로 표현

single precision에서는 bias 값을 127, double에서는 1023을 사용. (예를 들어 bias가 127이면 -1은 -1+127(ten) 또는 0111 1110(two)로 표현함.

단일 정밀도에서 표현가능한 가장 작은수와 큰수

작은수 : Exponent: 00000001, Fraction: 000...00 , significand = 1.0, ±1.00..00(0이23개)*2^(-126)

큰 수: exponent: 11111110, Fraction: 111...11 , significand ≈ 2.0±1.11..11(1이23개)*2^(127)

부동소수점 덧셈

9.999 × 10^1 + 1.610 × 10^–1

1) 먼저 작은 지수를 갖는 수의 소수점을 정렬시켜야 한다. 즉 작은수를 큰 지수와 일치하는 형태로 바꿔야 한다. -> **9.999 × 10^1 + 0.016 × 10^1**

2) 유효자리를 서로 더한다. -> **9.999 × 10^1 + 0.016 × 10^1 = 10.015 × 10^1**

3) 이 합은 정규화된 과학적 표기법이 아니라서 정돈해야 한다. -> **1.0015 × 10^2**

-> 한쪽이 양수고 한쪽이 음수면 결과가 선행하는 0을 가질 수 있어, 왼쪽 자리이동이 필요하다.

-> 지수가 증거하거나 감소할때마다 오ㅍㅍ버플로와 언더플로를 검사해야한다.

4) 유효자리가 네자리라고 가정했기 때문에(부호는 제외하고) 수를 자리 맞춤하여야 한다.(반올림)

-> **1.002 × 10^2** (경우에 따라서는 단계3을 다시해야하는 경우가 있다.)

지수 필드가 모두 0인 패턴은 예약되어 있으며 0의 부동소수점 표현을 위해 사용됨.

지수 필드가 모두 1인 패턴은 일반적인 부동소수점의 수의 표현을 벗어나는 값들과 상황 표현

단일 정밀도 표현방식의 최대 지수는 **127**이고 최소 지수는 **-126**이다.

부동소수점 곱셈

1.110 × 10^10 × 9.200 × 10^–5

1) 덧셈들과 달리 피연사자들의 지수를 더하기만 하면 곱의 지수를 구할 수 있다.

10+-5=5 -> 이것의 바이어스화된 지수는 5+127=132이다.

2) 유효자리의 곱셈이다. 위에서는 3개의 소수점만가지니 3개의 소수점만 생각한다.

1.110 × 9.200 = 10.212 -> 10.212 × 10^5

3) 이 곱은 비정규화되어 있으므로 알맞게 고친다.

1.0212 × 10^6

-> 이 시점에서 오버플로와 언더플로를 검사 (단일 정밀도 표현방식의 최대 지수는 **127**이고 최소 지수는 **-126**이다.)

4) 부호 부분을 제외하고 유효자리가 4자리고 가정했기 때문에 계산된 결과를 자리 맞춤하여야 한다.

1.021 × 10^6

5) 결과의 부호는 원래 피연사자의 부호에 의존하다. 부호가 같으면 양수, 다르면 음수다.

+1.021 × 10^6

MIPS의 부동소수점 명령어 -> p207,208

float f2c (float fahr) {
* return ((5.0/9.0)*(fahr - 32.0));*
} -> fahr in $f12, result in $f0, literals in global memory space

```
f2c:  lwc1   $f16, const5($gp)
      lwc1   $f18, const9($gp)
      div.s $f16, $f16, $f18
      lwc1   $f18, const32($gp)
      sub.s $f18, $f12, $f18
      mul.s $f0,   $f16, $f18
      jr     $ra
```

정확한 산술

자리올림(Rounding) -> IEEE 754는 계산하는 동안 오른편에 항상 두개의 추가 비트를 유지한다.

이를 각각 guard bit(보호비트)와 round bit(자리맞춤 비트)라고 부름.

ulp (units in the last place) -> 실제 수와 표현 가능한 수의 최하위 유효자리 비트중에 서로 다른 비트의 개수, 오버플로, 언더플로, 또는 유효하지 않은 연산의 예외가 없는 한, IEEE 754는 컴퓨터가 1/2 ulp 이내의 수를 보장한다.

	1.40	1.60	1.50	2.50	–1.50
Zero	1.00	2.00	1.00	2.00	–1.00
− ∞	1.00	1.00	1.00	2.00	–2.00
+ ∞	2.00	2.00	2.00	3.00	–1.00
Nearest Even (default)					
	1.00	2.00	2.00	2.00	–2.00

(-무한대 : 항상 자리내림, +무한대 : 항상 자리올림, nearest even: 가장 가까운 짝수로의 자리맞춤)

->마지막은 정확히 두값이 가운데 있을경우 어떻게 할지 정한다.

Nearest Even-

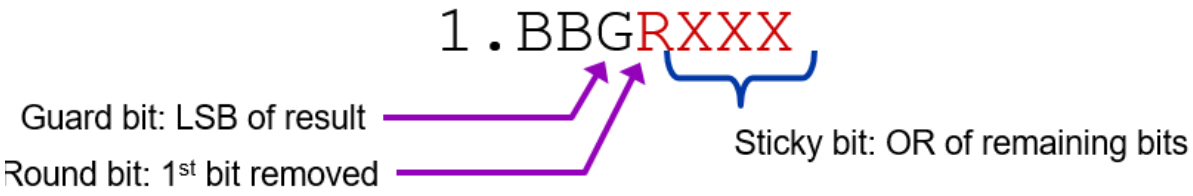
>IEEE 754표준은 계산된 결과 값이 두 값의 가운데 위치할 때, 최하위 비트가 홀수이면 더하기 1을 하고 짝수이면 잘라내라고 규정한다. 정가운데일 때 이 방법은 항상 최하위 비트를 0으로 만들어 준다.

sticky bit(점착 비트) -> guard bit와 round bit와 함께 자리맞춤에 사용되는 비트로서 자리맞춤 비트의 오른쪽에 0이 아닌 비트가 존재할 때 0이된다.

융합된 곱셈 덧셈 -> 곱셈과 덧셈을 수행하는 부동소수점 명령이지만 자리맞춤은 덧셈 수행후 한 번만 한다.

- Round to nearest 1/4 (2 bits right of binary point)

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2



오류 및 함정

오류 : 한 비트 왼쪽 자리이동 명령어가 2를 곱해 준 것과 같은 결과를 보이듯이 오른쪽 자리이동 명령어는 2로 나누어 준것과 같은 결과를 나타낸다.

-> 부호 있는 정수는 안된다.

함정 : 부동소수점의 덧셈은 결합벅칙이 성립하지 않는다.

오류 : 정수 데이터형에서 사용되는 병렬 수행 방힉은 부동소수점 데이터형에도 똑같이 적용된다.

함정 : MIPS 명령어 addiu는 16비트의 수치 필드를 부호확장하여 사용한다.

오류 : 이론 수학자만이 부동소수점 연산의 정확성에 신경쓴다.

예제 및 스스로 생각하기

p173, p176 **p181**, **p185**, p196-197, p200, **p207,208** , p215, p218