論文 96-21-5-25

수퍼스칼라 마이크로프로세서용 부동 소수점 연산회로의 설계

正會員 최 병 윤*, 손 승 일**, 이 문 기**

A Design of Floating-Point Arithmetic Unit for Superscalar Microprocessor

Byeong Yoon Choi*, Seung iL Sonh**, Moon Key Lee** *Regular Members*

"이 논문은 1994년도 학술 진흥 재단 공모과제 연구비에 의해서 연구되었음"

요 약

본 논문은 파이프라인 구조의 2개의 연산 경로와 새로운 반올림 및 정규화 회로를 사용하여 덧셈과 뺄셈, 데이타 형태 변환 연산, 비교 명령 등의 15가지 부동 소수점 연산 명령을 수행하는 수퍼스칼라 마이크로프로세서용 부동 소수점 연산 회로에 대해 기술한다. 부동 소수점 연산회로는 파이프라인 구조로 된 2개의 연산 경로를 가지고 있기 때문에, 각각의 부동 소수점 연산 동작이 자신의 연산 특성에 적합한 연산 경로에 할당될 수 있으므로 고속의 동작이 가능하다. 그리고 제안한 새로운 반올림 및 정규화 기법은 반올림과 정규화 동작을 병렬적으로 수행할 수 있도록 하며, 재정규화를 위한 결과 선택 동작을 단순화 시킨다. 또한 입력 데이터에서 덧셈 결과의 상위 0의 개수를 예측하는 기법을 통해, 기존 연산 회로에서 정규화를 위해 사용하는 연산 결과의 상위 1의 감지 동작을 제거할 수 있다. 설계된 부동 소수점 연산기는 3단 파이프라인 구조를 통해 15가지 단정도와 배정도 부동 소수점 연산을 수행할 수 있고, IEEE 754 표준안을 지원할 수 있으 므로 수퍼스칼라 마이크로프로세서에 내장하기에 적합한 구조를 갖고 있다.

ABSTRACT

This paper presents a floating point arithmetic unit(FPAU) for superscalar microprocessor that executes fifteen operations such as addition, subtraction, data format converting, and compare operation using two pipelined arith-

*동의 대학교 컴퓨터 공학과
Dongeui University, Dept. of Computer Eng.
**연세대학교 전자공학과 VLSI & CAD 연구실
論文番號:96092-0314
接受日字:1996年 3月 14日

metic paths and new rounding and normalization scheme. By using two pipelined arithmetic paths, each arithmetic operation can be assigned into appropriate arithmetic path which high speed operation is possible. The proposed normalization and rounding scheme enables the FPAU to execute rounding operation in parallel with normalization and to reduce timing delay of post-normalization. And by predicting leading one position of results using input operands, leading one detection(LOD) operation to normalize results in the conventional arithmetic unit can be eliminated. Because the FPAU can execute fifteen single-precision or double-precision floating-point arithmetic operations through three-stage pipelined datapath and support IEEE standard 754, it has appropriate structure which can be integrated into superscalar microprocessor.

Ⅰ. 서 론

우주 과학, 신호처리, 통계 등과 같은 과학 계산 분야에서는 큰 수와 작은 수를 복합적으로 필요로 하므로 분수(fraction)와 지수(exponent)를 사용하는 부동 소수점 표현 방식이 널리 사용되고 있다. 이러한 데이터에 대한 표준 표현 및 연산 방식을 IEEE(Institute of Electrical and Electronic Engineer) 학회에서 1985년 제정하였다[1]. 현재 전세계의 대부분의 기업 및 연구소에서 이 표준안을 따르고 있다. 명령어의 동적인 스케쥴링에 바탕을 둔 병렬처리와 파이프라인 처리 개념을 내부 아키텍처에 하드웨어로 구현하고 있는 수퍼스칼라 마이크로프로세서를 빠른 연산 능력이 요구되는 멀티미디어 통신, 디지탈 신호처리, 그래픽과 같은 응용 분야에 사용하기 위해서는 고성능의 부동 소수점 처리 장치가 필수 불가결하다[2]. 그러나 기존의 부동 소수점 처리 장치나 수치 보조 프로세서는 하드웨어와 칩 면적을 고려하여 파이프라인 처리 구조를 사용하지 않고 마이크로프로그래밍 제어 기법을 통해 하드웨어 융통성과 호환성을 강조하는 구조를 갖고 있다[3]. 따라서 이러한 구조는 명령어 수행의 병렬성과 고성능을 요구하는 수퍼스칼라 마이크로프로세서에 내장되기 위해서는 구조 변형이 요구된다. 그리고 IEEE 표준안에서 정의한 연산 형태중에서 BCD(binary coded decimal) 데이타 처리 연산과 일부분의 수치 보조 프로세서에서 구현되는 초월 함수에 대한 연산은 하드웨어로 직접 구현하는 방안보다 단순화된 명령어를 조합하여 소프트웨어적으로 구현하는 것이 부동 소수점 연산 장치를 파이프라인 구조로 만드는데 바람직하다[4]. 본 논문에서는 IEEE 표준안의 지원과 함께 다양한 응용 분야를 고려하여 15가지 부동 소수점 연산 명령을 정의하고, 이를 효율적으로 수행하는 새로운 구조의 부동 소수점 연산기를 설계한 후 성능을 분석하였다.

본 논문의 II장에서는 FPAU의 설계 사양을 설명하고, III장에서는 2 경로 기법을 사용한 부동 소수점 덧셈/뺄셈 알고리즘 및 데이터 변환 명령어 등의 구현 알고리즘과 함께 새로운 반올림 및 정규화 알고리즘을 기술하고, IV장에서는 전체 FPAU의 하드웨어 설계를 기술하고, V장에서는 설계한 회로에 대한 모의 실험 결과 및 성능 분석에 대해 기술하고, VI장에서는 결론을 맺는다.

Ⅱ. 부동 소수점 연산기의 설계 사양

본 연구의 FPAU(Floating-Point Arithmetic Unit)에서 지원하는 명령어는 표 1과 같이 15 가지로서 IEEE 표준안을 효율적으로 지원하며, 디지탈 신호처리, 멀티미디어 통신 등의 응용을 고려하여 정의하였다. 정의된 15가지 명령과 곱셈 및 나눗셈 명령을 효율적으로 결합하면, BCD 연산 명령과 초월함수를 쉽게 구현할 수 있다. 표 1에서 아래 첨자 s는 단정도 데이터, d는 배정도 데이터, i는 정수 데이터를 의미한다.

본 부동 소수점 연산기는 부동 소수점 데이터에 대한 연산과 함께 정수와 부동 소수점 데이터간의 변환 동작 수행에 따라, 그림 1과 같이 2가지 형태의 부동 소수점 데이터와 32 비트 정수 데이터를 지원한다. 부동 소수점 형식의 수가 나타내는 값, $(FP)_{value}$은 다음과 같다.

$$(FP)_{value} = (-1)^S \, 2^E \, (1.f_1 \, f_2 \, ... \, f_{p-1}) \qquad (1)$$

표 1. 지원되는 명령
Table 1. Supported instructions

명령어	동 작
FABSs	f[rd]s ← abs(f[rs2]s)
FADDd	f[rd]d ← f[rs1]d + f[rs2]d
FADDs	f[rd]s ← f[rs1]s + f[rs2]s
FCMPd	fcc ← f[rs1]d compare f[rs2]d
FCMPs	fcc ← f[rs1]s compare f[rs2]s
FdTOi	f[rd]i ← f[rs2]d
FdTOs	f[rd]s ← f[rs2]d
FiTOd	f[rd]d ← f[rs2]i
FiTOs	f[rd]s ← f[rs2]i
FMOVs	f[rd]s ← f[rs2]s
FNEGs	f[rd]s ← − f[rs2]s
FsTOd	f[rd]d ← f[rs2]s
FsTOi	f[rd]i ← f[rs2]s
FSUBd	f[rd]d ← f[rs1]d − f[rs2]d
FSUBs	f[rd]s ← f[rs1]s − f[rs2]s

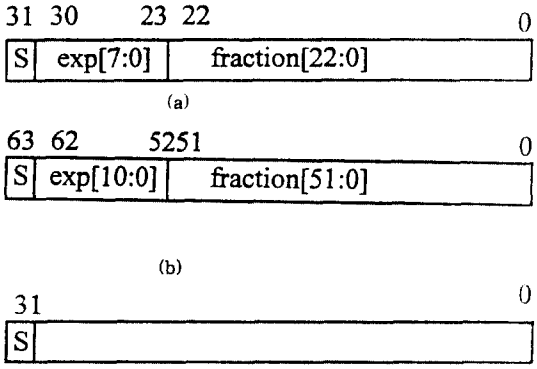

그림 1. 부동 소수점 데이터 형태와 32 비트 정수 데이터 형태
(a) 단정도 데이터
(b) 배정도 데이터
(c) 정수 데이터

Fig. 1. Floating-point data format and 32 bit integer format
(a) single-precision data
(b) double-precision data
(c) integer data

여기서 s=0 또는 1,
fi=0 또는 1
biased exponent(exp) = E + bias

부동 소수점 데이터의 경우 분수부(significand)의 맨 상위 정수값(hidden bit) 1는 컴퓨터 내부에 저장될 경우에는 제외되지만 하드웨어에서 연산시 이 값을 1로 사용하게 된다. 단, 저장된 값이 0인 경우는 예외적으로 0으로 사용한다. 본 연구에서는 확장 정밀도 데이터, 비정규화 데이터, NAN 데이터는 하드웨어로 지원치 않고 소프트 웨어로 처리하는 방식을 취했다. 그리고 IEEE 표준안에서 정의된 4가지 반올림 양식과 함께 5가지 예외처리를 지원하도록 하였다. 단, IEEE 표준안에 정의된 곱셈, 나눗셈, 제곱근 계산은 별도의 하드웨어를 사용하여 구현하는 것으로 가정하여 본 연구에서는 포함시키지 않았다.

Ⅲ. FPAU에서 구현된 명령어의 연산 알고리즘

3.1. 부동 소수점 덧셈과 뺄셈 알고리즘

부동 소수점 덧셈과 뺄셈은 지수부 차이만큼의 입력 정렬동작(우측 이동), 덧셈 또는 뺄셈, 정규화(normalization), 반올림(rounding), 재정규화(post-normalization)의 5가지 단계가 순차적으로 이루어진다. 그런데 부동 소수점 덧셈과 뺄셈의 동작 특성을 분석해 보면 크게 2가지 경로로 세분해서 처리하는 것이 가능함을 알수 있다[5]. 즉 분수부 부호와 연산 명령을 고려한 실제 분수부에 이루어지는 유효 분수부 연산(Eop:effective operation)이 덧셈(Eop=0)인 경우, 최종 결과는 정규화된 형태 또는 1 비트만큼 분수부 오버플로우가 발생한 상태로 얻어진다. 따라서 정규화 동작이 1 비트의 우측 이동 또는 이동 불필요 형태로 얻어진다. 그리고 유효 분수부 연산이 뺄셈(Eop=1)이고, 지수부 차이가 2이상인 경우 뺄셈 결과로 얻어지는 결과값은 정규화된 값 또는 1 비트만큼 분수부 언더플로우(underflow)가 발생한 형태로 생성된다. 따라서 정규화 동작시 최대 1 비트의 좌측 이동이 필요하다. 반면에 유효 분수부 연산이 뺄셈이고 지수부 차이가 0 또는 1인 경우, 뺄셈 결과값은 정규화 과정에서 2 비트 이상의 좌측 이동 동작이 필요하다. 이러한 연산 동작중 0 또는 1비트의 좌우측 이동 동작은

시프터 대신에 단순하게 2-to-1 MUX로 구현하는 것이 가능하다. 따라서 본 연구에서는 부동 소수점 덧셈과 뺄셈을 다음과 같이 2가지 경로로 나누어 처리한다.

1). SA(Shift-Add) 경로: 정규화 동작이 간단한 연산
   - 유효 연산이 덧셈
   - 유효 연산이 뺄셈이고, 지수부 차이가 2이상인 경우
2). AS(Add-Shift) 경로: 정규화 동작시 큰 좌측 이동 동작 연산
   - 유효 연산이 뺄셈이고, 지수부 차이가 0 또는 1인 경우

이와 같이 2개의 경로로 연산을 분리하면, 단일 경로와는 달리 각 경로에 하나의 시프터(shifter)만 존재하므로 빠른 연산이 가능하다. 단 기존 하드웨어에 비해 하드웨어 양이 다소 증가한다.

덧셈과 뺄셈 연산시 정규화와 반올림 동작을 효율적으로 수행하는 방안[6, 7]이 제안되고 있는데, 참고 문헌 [6] 방식의 경우 L 비트를 중심으로 상위부와 하위부로 분리하여, 정규화와 반올림 동작을 동시에 수행하는 장점이 있지만, IEEE 표준안의 4가지 방안을 적용할 경우 round-infinity 반올림 양식의 경우 결과 선택 동작이 복잡하며, 2 경로를 고려하지 않은 단점이 있다. 참고 문헌 [7]방식의 경우는 단일 경로에 대해 효율적인 정규화를 구현하고 단순하다는 장점은 있지만, 정규화 후에 반올림이 필요하며 2 경로 구조에 바람직하지 않다는 단점이 있다. 본 연구에서는 상위부와 하위부를 N 비트를 중심으로 분리하여, 모든 반올림 양식에 대해 결과값의 선택 종류를 2가지로 제한하는 방식을 제안하고, 2가지 경로에 대해 독립적인 정규화 및 반올림 기법을 적용한다.

SA 경로에 대한 덧셈과 뺄셈에 대한 결과 형태는 그림 2와 같이 3가지 형태로 나타난다. 그림 2에서 L, G, R, S 비트는 결과값이 정규화된 형태로 얻어질 경우를 기준으로 결과의 최하위 비트(lsb), 가드(guard), 라운드(round), 스티키(sticky) 비트를 나타낸다. 단, N 비트는 덧셈시 1 비트만큼의 분수부 오버플로우가 발생할 경우 반올림 위치를 나타낸다. 그림 3은 SA 경로에 대한 정규화 및 반올림 기법을 나타낸다. 정규화와 반올림 동작을 동시에 수행하기 위해 다음과 같은 동작을 수행한다. 입력 정렬기를 거친 2개의 n-비트 데이터를 반가산기(HA) 행을 통과시킨 후, N 비트를 기점으로 상위부와 하위부를 분리시켜 독립적으로 덧셈동작을 수행한다. 이 경우 상위 n-1 비트에 대해서는 하위 비트에서 넘어오는 캐리 입력값을 대비하여, 2가지 형태의 덧셈(carry vector + sum vector + 0, carry vector + sum vector + 1)을 수행한다. 이 경우, 반올림이 반영되지 않은 하위 4 비트에 대한 덧셈 결과와 반올림을 고려한 덧셈에서 단 하나의 캐리 출력만 생성되면, 2가지 경우의 캐리 출력을 OR한 값으로 반올림이 반영된 최종 결과의 상위 비트를 복합 가산기(compound adder)의 출력에서 선택할 수 있다. 최종 결과의 하위 비트는 하위 3비트에 반올림 신호(Rv, Rnv, Ruv)를 더한 결과에서 결정할 수 있다. 하위 4비트에 스티키 비트에 의한 캐리입력과 반올림에 따른 추가의 덧셈에서 하나의 캐리 출력만 생성됨을 증명하기 위해, 반가산기 행을 거친 4비트 출력의 내부 구성을 1 비트 덧셈 동작 특성(0 + 0 = 00, 1 + 1 = 10, 1 + 0 = 01, 0 + 1 = 01)을 이용하여 2가지 형식으로 분리하였다. 이때 스티키 비트가 반영된 1 비트 HA값의 출력 Cin을 더할 경우, 캐리 출력(sel_cin)이 0 또는 1이 발생할 수 있는 경우는 그룹 1이고, 캐리 출력이 항상 0인 경우는 그룹 2로 정의했다. 그런데 그룹 1은 덧셈 결과의 최상위 비트가 항상 0이므로 반올림값(Rv, Rnv, Ruv)이 더해지더라도, 반올림 과정에서 추가의 캐리 출력(sel_rnd)은 항상 0이 된다. 반면에 그룹 2는 4 비트 덧셈 결과의 상위 3비트가 모두 0 또는 1이므로 반올림 값이 더해질 경우 캐리 출력(sel_rnd)이 1이 될 수 있다. 즉 sel_cin과 sel rnd는 동시에 1이 될 수 없으므로, 반올림 결과의 상위 비트는 복합 가산기의 출력으로 결정될 수 있다. 단, 여기서 스티키 비트(S)를 포함해서 5비트 가산기를 구현하지 않은 이유는 동일 하드웨어내에 단정도와 배정도의 덧셈/뺄셈을 수행할 경우, 스티키 비트를 분리하여 1 비트 HA를 사용하여 Cin을 생성하는 것이 반올림 및 결과값 제어에 용이하기 때문이다. 그리고 sel_rnd는 반올림이 반영되지 않은 덧셈 결과의 상위 2 비트(vi_f, vi_no)에 따라 3가지 가산기의 캐리 출력 중에서 하나를 선택함으로서 결정된다. 그리고 sel_cin + sel_rnd에 따라 반올림이 반영된 상위 2비트(f_ovf, f_n)가 결정된다. 이 값을 사용하여 정규화, 반올림및 재정규화가 반영된 최종 결과 선택 동작이 이

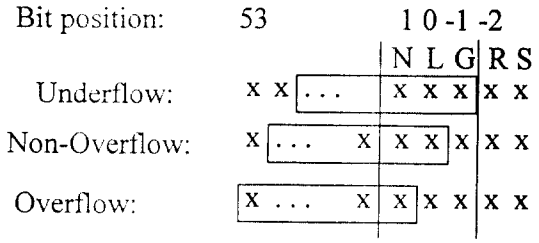

그림 2. SA 경로를 사용한 배정도 덧셈과 뺄셈에 대한 3가지 결과 형태

Fig. 2. Three result formats of double-precision addition and subtraction operation using SA arithmetic path

표 2. SA 연산 경로를 사용하는 덧셈과 뺄셈 동작에서 결과 선택 동작

Table 2. Selection operation of results in the addition and subtraction using SA arithmetic path

Eop	f_ovf	f_n	s_h	최종 결과값
0	0	X	0	sm_0[n−3:0], nv_c
			1	sm_1[n−3:0], nv_c
	1	X	0	sm_0[n−2:0], v_c
			1	sm_1[n−2:0], v_c
1	X	0	0	sm_0[n−4:0], uv
			1	sm_1[n−4:0], uv
		1	0	sm_0[n−3:0], nv_c
			1	sm_1[n_3:0], nv_c

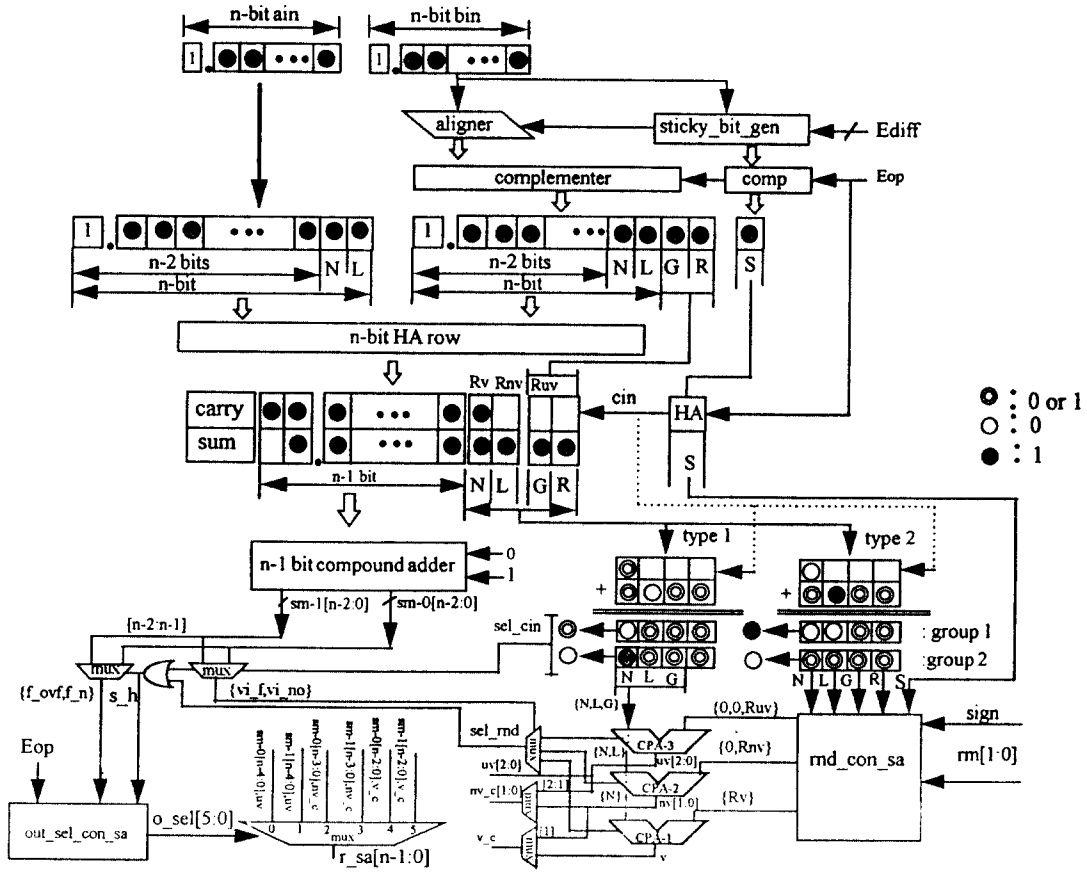

그림 3. SA 경로를 사용하는 덧셈과 뺄셈에 대한 정규화, 반올림 및 재정규화 회로

Fig. 3. Normalization, rounding, and post-normalization scheme of addition and subtraction using SA arithmetic path

루어진다. 그런데 하위 1 비트(v_c) 또는 2 비트(nv_c [1:0])를 결정할 때 고려해야 할 사항은, 반올림 동작에 의해서 1 비트만큼 결과값이 상승하는 경우 하위 비트 결과 선택시 이러한 경우를 고려해 주어야 하는데, 이를 위해 2개의 MUX를 사용하였다. 최종 결과값을 결정하는 결과 선택 제어 회로 동작은 표 2와 같다. 그리고 반올림 제어 회로는 시간 지연을 방지하기 위해 결과의 부호(sign), 4가지 반올림 양식(rm[1:0])을 사용하여, 3가지 형태의 반올림값(Rv, Rnv, Ruv)을 병렬적으로 생성한다. 그림 3에서 SA 경로에 대한 스티키 비트 생성회로(stick_bit_gen)는 Bin에서 하위 0의 개수를 결정하는 TOD(trailing one detector)에서 결정한 값과 우측 방향의 이동 거리(ediff)값에서 결정하는 방식[8, 9]을 이용하여 결정하였다. 그림 4는 AS 경로에 대한 정규화 및 반올림 기법을 나타낸다. SA 경로와 유사한 기법을 사용하지만 반올림이 발생하는 경우는, Ediff = 1로서 1 비트 우측 시프트가 있으며 정규화 동작시 왼쪽 방향으로의 이동이 없을 경우({sgn,vi} = 01)이다. 이 경우 L 비트를 기준으로 상위 비트와 하위 비트를 구분하면, SA경로와 마찬가지로 sel_cin_as와 sel_rnd_as는 동시에 1이 되지 못하므로, 반올림 결과값의 상위 비트를 2가지 종류로

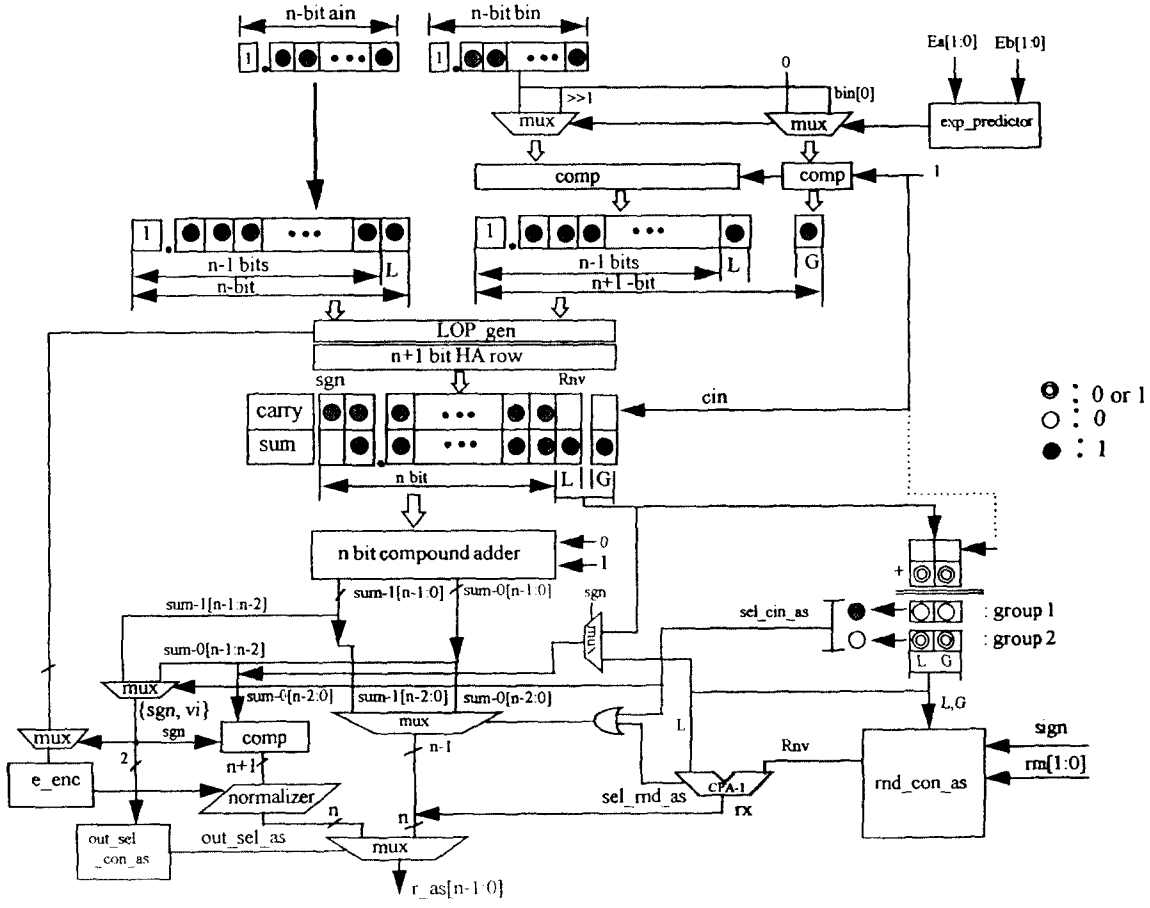

그림 4. AS 경로를 사용한 뺄셈에 대한 정규화, 반올림 및 재정규화 기법

Fig. 4. Normalization, rounding, and post-normalization scheme for subtraction using AS arithmetic path

제한할 수 있다. 그리고 뺄셈시에는 반올림에 기인한 분수부 오버플로우가 발생할 수 없으므로 결과 선택 동작이 단순하다. 그런데 AS 경로에 대한 연산 시작을 지수부에 대한 뺄셈 결과 후에 실제 지수부 차이가 0 또는 1로 결정된 후 시작하게 되면 시간 지연이 발생한다. 이러한 지연시간을 방지하기 위해, 2가지 지수부의 하위 2비트(Ea[1:0], Eb[1:0])를 사용하여, 이동할 분수부값과 이동 크기를 예측하는 지수부차 예측기(exp_predictor)[10]를 사용하였다. 올바른 지수값에 따른 경로 선택은 최종 결과값 선택 단계인 파이프라인 단계 3에서 이루어진다. 그리고 AS 경로의 경우 지수부 차이가 0인 경우 뺄셈 연산시 결과값이 음수가 되어 2의 보수 형태의 결과가 얻어질 수 있다. 그런데 부동 소숫점 데이터의 저장 형태가 부호-크기 형태이므로, 결과값에 대해 시간 지연을 야기시키는 추가의 2의 보수화 덧셈 동작이 필요하다. 이러한 문제점을 제거하기 위해 본 논문에서는 그림 5과 같이 2의 보수 뺄셈과 함께 1의 보수 뺄셈을 함께 수행한 후, 그 결과값이 양수인 경우는 2의 보수 뺄셈 결과를 취하고, 음수인 경우 1의 보수의 결과 값을 반전시켜 부호-크기 형태의 결과값으로 취하는 기법[11, 12]을 사용하였다. 그림 4의 경우 결과 값이 음수(sgn = 1)일 경우, 1의 보수 뺄셈 결과값을 복합 가산기의 출력(sum_0[n-1:0])과 복합 가산기에 참여하지 않은 L, G 비트를 선택해서 반전시킨다. 단 G 비트는 불필요하지만 정규화에 참여하므로 포함시켰다. 그리고 AS 경로를 사용한 뺄셈의 경우 최종 결과값에 대해 다수의 왼쪽 방향의 이동 동작를 수행하는 정규화 동작이 필요하다. 본 연구에서는 정규화동작이 덧셈과 뺄셈 연산이 완료된 직후에 바로 수행될 수 있도록, 뺄셈 연산중 입력 데이터에서 연산 결과의 상위 0의 개수를 예측하는 LOP(leading one predictor)회로를 사용

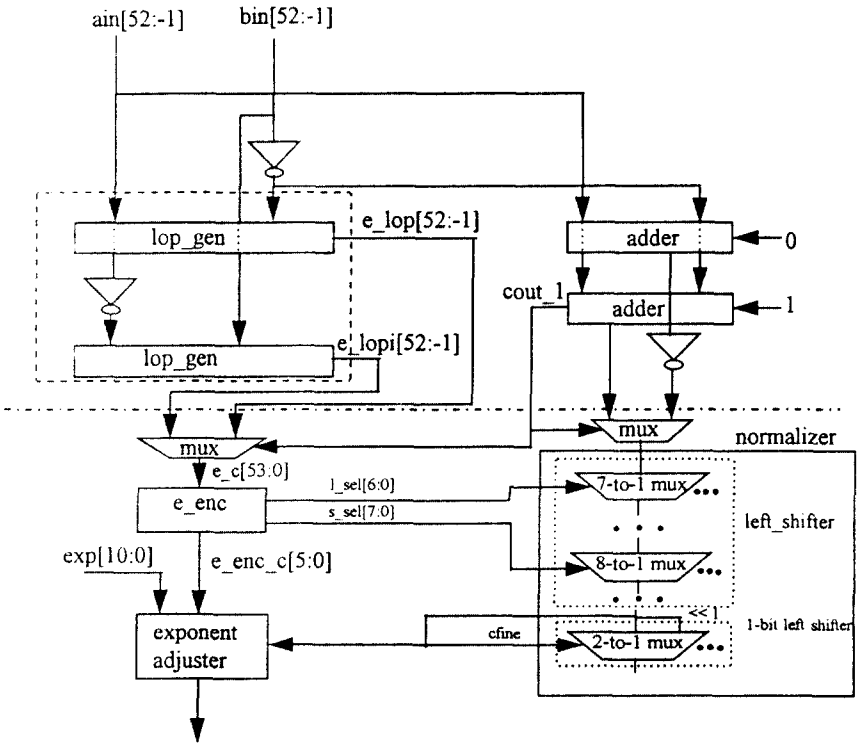

그림 5. LOP회로와 정규화 회로
Fig. 5. LOP circuit and normalizer

하였다. 단 기존의 LOP회로[13]가 복잡하기 때문에 이를 수정한 방식[14]을 이용하였다. 단 참고문헌 [14]에서 제안한 방식은 항상 결과 값이 양수일 경우에만 타당한 예측을 할 수 있는 제한이 있다. 그런데 지수값이 0인 경우 뺄셈에서 음의 결과가 생성될 수 있으므로, 본 연구에서는 상위 0의 개수를 예측하는 회로를 쌍으로 사용하여, A-B, B-A에 대한 연산 결과를 동시에 예측한 후, 연산 결과의 부호값에 따라 2가지 결과중에서 하나를 선택하는 방식을 사용하였다. 그런데 LOP에 의해 예측된 결과는 1 비트만큼의 오차가 있으므로 정규화 회로(normalizer)는 그림 5과 같이 2-단계로 구성된 왼쪽 방향의 배럴 시프트와 함께 맨 상위값(cfine)에 따라 추가의 1 비트 왼쪽 방향의 이동 동작을 제어하기 위한 하나의 멀티프렉서(MUX)로 구성된다. 정규화 회로에 사용한 왼쪽 방향 시프터(left-shifter)의 단계 1의 7-to-1 MUX은 0, 8, 16, 24, 32, 40, 48의 이동을 제어하고, 단계 2의 8-to-1 MUX는 0, 1, 2, 3, 4, 5, 6, 7단위의 이동을 제어한다. lop-gen회로는 A-B와 B-A에 대한 첫 번째 1의 위치에 대한 추정값(e_lop, e_lopi)을 생성한다. 2개의 값중에서 결과 부호에 따라 하나를 선택한후, 시프터 제어와 지수부 제어에 적합한 형태로 인코딩한 후, 정규화 동작을 제어함과 동시에 지수값에 대한 조정 동작을 수행한다. 주어진 SA와 AS 경로에 대한 정규화 및 반올림 알고리즘은 하나의 부동 소수점 데이터에 대해 적용되므로, 본 연구에서는 단정도와 배정도 연산을 동시에 지원하기 위해서 배정도 하드웨어를 구현하고, HA 열과 복합 가산기를 세분하고 중간에 MUX를 삽입한 후, 데이터 형식에 따라 적절한 제어를 통해 단정도 연산도 지원할 수 있도록 하였다.

3.2. 정수 데이터와 부동 소수점 데이터 사이의 변환 연산 알고리즘

본 연구에서 32 비트 정수 데이터를 부동 소수점 데이터로 변환하는 동작은 AS 경로를 변형하여 사용하였다. 단 정수 데이터를 배정도 부동 소수점 데이터로 변환하는 동작은 필드폭의 특성으로 반올림 동작이 발생치 않는다. 반면, 정수 데이터를 단정도 부동 소수점 데이터로 변환하는 동작은 반올림 동작이 필요하다[15]. 그림 6은 본 연구에서 사용한 정수 데이터를 부동 소수점 데이터로 변환하는 알고리즘을 나타낸다. 본 연구에서 정수를 부동 소수점 데이터로 변환하는 알고리즘은 3 단계로 구성된다.

[단계 1] 정수의 부호를 분석하여 양수이면 0 + Bin을 수행하며, 음수이면 0-Bin 동작을 수행한다. 이 과정에 LOP회로를 사용하여 정규화에 필요한 거리를 예측한다. 그리고 TOD값을 사용하여, Bin의 하위 비트에 존재하는 0의 개수를 결정한다.

[단계 2] LOP 결과(e_enc_c)를 사용하여 정규화를 수행한다. 단 정규화 과정에서 1 비트의 추가 이동조건(Cfine)이 발생되는 경우, 1 비트만큼 추가로 왼쪽 방향으로 이동한다. 그리고 지수값은 다음과 같이 결정된다.

$$\text{조정된 지수값(Exp\_c)} = (\text{Bias\_p} + 31) - \text{e\_enc\_c} - \text{Cfine} \quad (2)$$

여기서 Bias_p는 단정도인 경우 127이며, 배정도 데이터인 경우는 1023이다. 그리고 단계 1에서 결정한 TOD값과 함께 e_enc_c와 Cfine값을 사용하여, FiTOs 명령에 대한 반올림 동작을 위한 스티키 비트를 생성한다.

[단계 3] 단계 2에서 결정한 스티키 비트와 단계 2에서 구한 정규화된 데이터 값에서 Ls, Gs, Rs 값을 사용하여 4가지 반올림 모드에 대해 반올림 동작을 수행한 후 최종 결과값을 구할 수 있다.

본 연구에서는 위의 3가지 단계를 구현하기 위해, 단계 1, 2는 기존 뺄셈의 AS 경로를 일부 수정하여 적용하고, 단계 3은 추가의 하드웨어를 할당하였다. 그리고 단계 3에서 반올림이 필요한 경우는 단정도 데이터로 제한된다. 따라서 가산기(INCrement)회로의 크기가 24 비트이므로 속도 문제가 야기되지 않고 입력 데이터가 정규화된 형태로 들어오므로 반올림과 재정규화를 순차적으로 하는 방식을 사용하였다.

본 연구에서 부동 소수점 데이터를 정수 데이터로 변환하는 동작은 덧셈과 뺄셈 동작을 위한 SA 경로를 수정하여 사용하였다. 단, 이 경우 최종 결과값은 정수 데이터이므로 2의 보수 형태의 결과가 필요하다. 그런데 IEEE 표준안에는 정수 데이터에 대해서는 반올림이 정의되어 있지 않기 때문에 본 연구에서는 부동 소수점 데이터를 부호 값에 따라 2의 보수 처리 동작을 선택적으로 수행한 후, 결과값으로 32 비

트를 취하는 방식을 사용하였다. 단 32 비트로 결과 선택시 절단으로 없어지는 데이터가 존재하는 경우, 부정확한 결과를 지시하는 예외 조건(inexact-ftoi)을 생성한다. 부동 소수점 데이터를 정수 데이터로 변환하는 알고리즘은 다음과 같은 단계로 구현하였다.

[단계 1] 부동 소수점 데이터에 대한 우측 방향의 이동 거리를 식(3)과 같이 결정한다. 단, Ediff값이 0보다 작으면, 결과값은 정수로 표현하기에 너무 큰 값이 되므로 오버플로우 예외조건으로 감지한다. 단 Bias_p + 31은 사용되지 않는 Ain의 지수 필드(Ea)에 하드웨어적으로 할당한다.

$$\text{Ediff} = (\text{Bias\_p} + 31) - \text{Eb} \qquad (3)$$

[단계 2] Ediff가 양수인 경우 데이터 정렬기(aligner)를 사용하여, 우측 방향으로 이동함과 동시에 Ediff를 사용하여 스티키 비트를 생성한다.

[단계 3] 정렬된 데이터에 대해 부호값에 따라 선택적으로 반전 동작을 수행한 후, 2의 보수 개념을 사용한 덧셈을 기존 SA 경로를 사용하여 수행하고 최종 결과값으로 상위 32 비트를 취한다.

3.3. 부동 소수점 데이터에 대한 비교 명령

부동 소수점 데이터에 대한 비교 명령은 AS 경로와 함께 부호 비트, 그리고 지수값 차로 결정하였다. 즉 비교 명령인 경우 지수값 차이가 0이 아닌 경우는 지수값차이와 부호 값으로 조건 코드(Cond[3:0])를

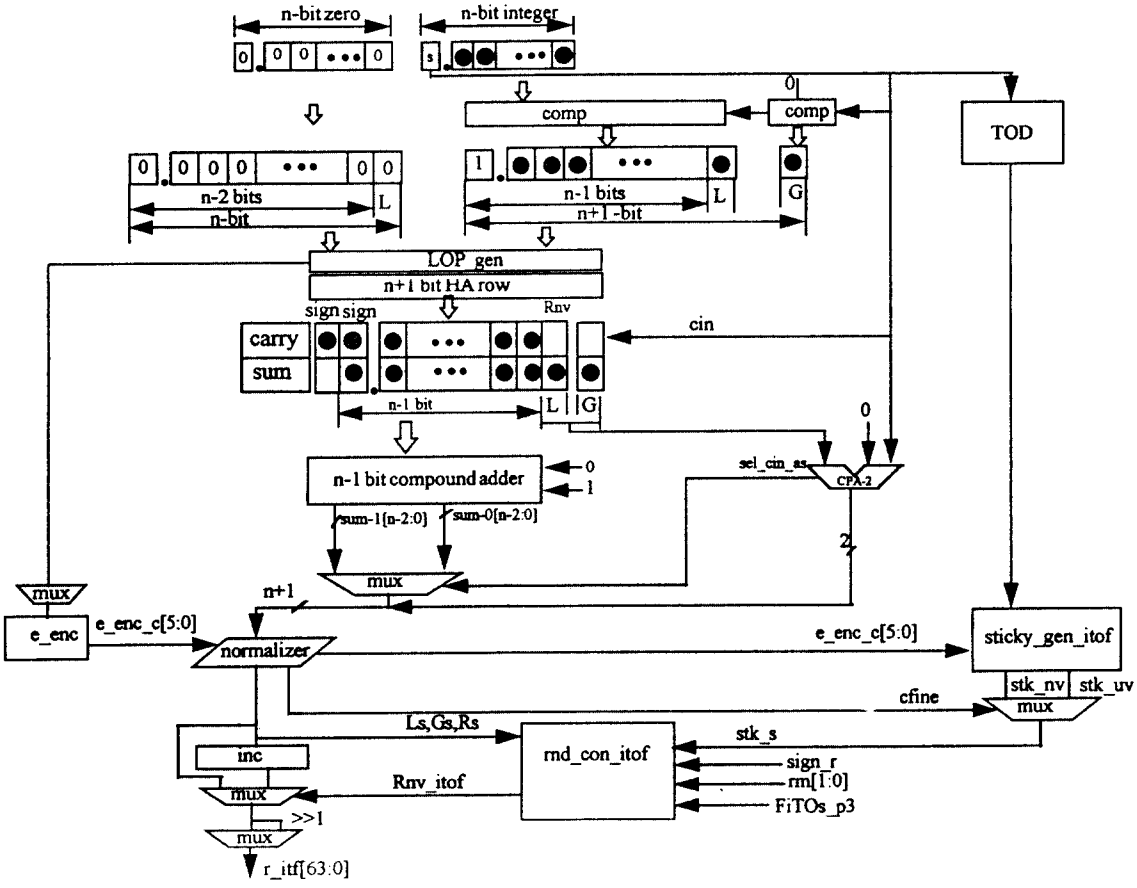

그림 6. 정수 데이터를 부동 소수점 데이터로 변환하는 연산 기법
Fig. 6. Scheme to convert integer into floating-point data

결정하고, 지수값 차이가 0인 경우는 AS 경로에서 얻어진 결과에서 분수부값의 0조건과 Fa가 Fb보다 크다는 조건을 감지하여 조건 코드를 결정한다. 단 +0와 -0는 동일(equal)조건으로 판단되어야 하므로, 입력 데이터에 대한 분석 결과를 사용한다. 그 이외에 입력 데이터의 특별한 조건(NAN) 등을 감지하여, 부순서(unordered) 비교 결과도 결정한다. 이러한 조건 코드는 파이프라인 단계 2에서 결정되지만, 1 사이클을 지연 시켜 다른 명령과 동일하게 3 단계의 파이프라인을 거치도록 하여 제어를 단순화시켰다.

3.4. 부동 소수점 데이터간 변환과 나머지 명령 구현 기법

단정도 데이터를 부동 소수점 데이터로 변환하는 명령은 AS 경로를 활용하며 지수부값만 조정하면 된다. 반면에 배정도 데이터를 단정도 데이터로 변환하는 동작은 지수부 조정과 함께 반올림 동작이 필요하다. 단 변환과정에 오버플로우 예외조건이 발생할 수 있으므로, 이에 대한 감지동작을 하드웨어로 구현하였다. 단, 반올림 과정에서 분수부 오버플로우가 야기될 수 있으므로, 빨셈을 위한 AS 경로에서 약간의 변형과 함께 반올림을 지원하기 위해, TOD 결과를 사용한 스티키 비트 생성 회로가 필요하다.

본 연구의 FPAU에서 나머지 명령(FMOVs, FABSs, FNEGs)은 기존의 AS 경로를 활용하며, 부호 값에 대한 단순한 제어로 구현된다.

Ⅳ. 하드웨어 구현

표 3는 본 FPAU에서 지원하는 15가지 명령의 2가지 경로 활용을 나타낸다. 단 ASA(Add-Shift-Add) 경로는 정수 데이터를 부동 소수점 데이터로 변환하는 동작을 구별하기 위해 표현하였으며, 실제로는 3단 파이프라인 구현시 AS 경로라 할수 있다. 레지스터 파일을 포함한 전체 부동 연산회로의 블럭도는 그림 7과 같다. III장에서 제안된 알고리즘을 기준으로 단일 파이프라인 단계에 하나의 가산기 또는 시프터를 할당하는 방식으로 데이터패스를 할당하여 동작 속도를 높이며 파이프라인 단계별로 적절한 시간 분배가 이루어 질 수 있도록 하였다. 따라서 AS경로에서 수행되는 연산은 기본적으로 정수를 부동 소수점 데이터로 변환하는 명령을 제외하고는 모두 2단계의 파이프라인으로 결과가 완료될 수 있지만, 계산된 결과를 1 사이클 지연시켜 3단계의 파이프라인 단계로 구현하였다. SA 경로를 사용하는 명령은 파이프라인 단계 1에는 시프터를 할당하고, 파이프라인 단계 2에는 복합 가산기를 할당하였다. 단계 3에는 다양한 반올림 양식을 고려하여 정규화와 반올림, 재정규화가 반영된 결과 선택을 위한 데이터 패스를 할당하였다. 단계 3의 동작이 많은 시간을 필요로 하지 않기 때문에 2개의 파이프라인 단계를 통합하는 것도 가능하지만 본 연구에서는 AS 경로가 이미 3단계의 파이프라인으로 구현되었기 때문에 마찬가지로 SA경로도 3단계의 파이프라인으로 구현하였다. 따라서 설계된 FPAU는 모든 명령을 3단계의 파이프라인을 통해 효율적으로 구현할 수 있다. 그리고 FAPU에서 지원하는 데이터가 단정도와 배정도 데이터이므로, 배정도 데이터에 대해 반복 사이클 없이 3단계로 파이프라인 동작을 지원할 수 있는 내부 구조를 갖고 있다. 그리고, 단정도 데이타에 대한 연산의 반올림을 위한 Ls, Gs, Rs, Ss 등을 생성하고 적절한 반올림 동작을 보장하기 위해, 내부 데이터 패스를 적절히 세분할 수 있도록 내부에 다수의 MUX회로를 갖고 있다. 내부 데이터 패스가 배정도 데이터에 적합한 구조이므로, 레지스터 파일에 대한 단정도 부동 소수점 데이타와 32비트 정수 데이타에 대한 연산이 동일 하드웨어상에서 구현되기 위해서는 배정도 형태로 변형이 필요하

표 3. 15가지 명령어별 경로 할당
Table 3. Path assignment for fifteen instructions

명령어	SA 경로	AS 경로	ASA 경로
FADDs, FADDd FSUBs, FSUBd	. EADD . ESUB & Ediff ≥ 2	. ESUB & Ediff = 0 or 1	
FABSs		○	
FCMPd, FCMPs		○	
FdTOi, FsTOi	○		
FdTOs, FsTOd		○	
FiTOd, FiTOs			○
FMOVs		○	
FNEGs		○	

다. 입력 데이타 변환 회로(input formatter)를 사용하여 내부 하드웨어에 적합한 형태로 입력 데이타가 변환된다. 마찬가지로 FPAU에서 연산이 완료된 후에는 최종 결과값의 바람직한 형태, 즉 배정도, 단정도 혹은 32 비트 정수 데이타 형태로 변형되어야 하므로 출력 변환 회로(out_formatter)가 사용된다. 입력 데이타 변환 회로는 레지스터 화일에서 읽은 32 비트 또는 64 비트 데이타에 대해, 54-비트 분수부 비트, 11-비트 지수 비트 및 1 비트의 지수 비트를 생성한다. 배정도 데이타의 경우, 숨겨진(hidden) 비트가 포함됨과 동시에 0의 부호 비트(sign bit)가 추가된다. 단정도 데이타의 경우 배정도 데이터와 유사하지만, 분수부의 끝에 29개의 0이 삽입된다. 32 비트 정수 데이타의 경우는 감추어진(hidden) 비트 개념이 필요없으므로, 부호(sign) 비트가 1 비트만큼 부호 확장이 됨과 동시에 부호값이 배정도 데이타의 부호 필드(Sb)에 할당된다. 레지스터 화일은 2개의 16 × 32 비트 구조의 레지스터들로 구성된다. 따라서 최대 16개의 배정도의 데이타 또는 32개의 단정도 데이타를 가질수 있다.

그림 8은 지수부에 대한 블럭도를 나타낸다. 파이프라인 단계 1의 경우 절대값의 지수차를 결정하기 위한 복합 가산기를 갖고 있다. 그리고 부동 소수점 데이터를 정수 데이터로 변환하는 명령에 대한 정렬기 이동 거리를 결정하기 위해, Ea대신에 하드웨어적으로 Bias + 31을 생성하는 MUX 회로가 존재한다. 또한 복합 가산기의 캐리 출력값 또는 외부 제어 조건에 따라 파이프라인 단계 1에서의 지수값을 결정하는 MUX 회로가 존재한다. 파이프라인 단계 2의 경우에는 AS 경로에서 수행되는 정수값을 부동 소숫점으로 변환하는 명령을 위해 Bias + 31을 선택하는 MUX회로와 LOP값에 따라 지수값을 조정하는 가산기를 갖고 있다. 그런데 이 가산기를 복합 가산기 형태로 만든 이유는 LOP에 의해 예측된 지수 조정 거

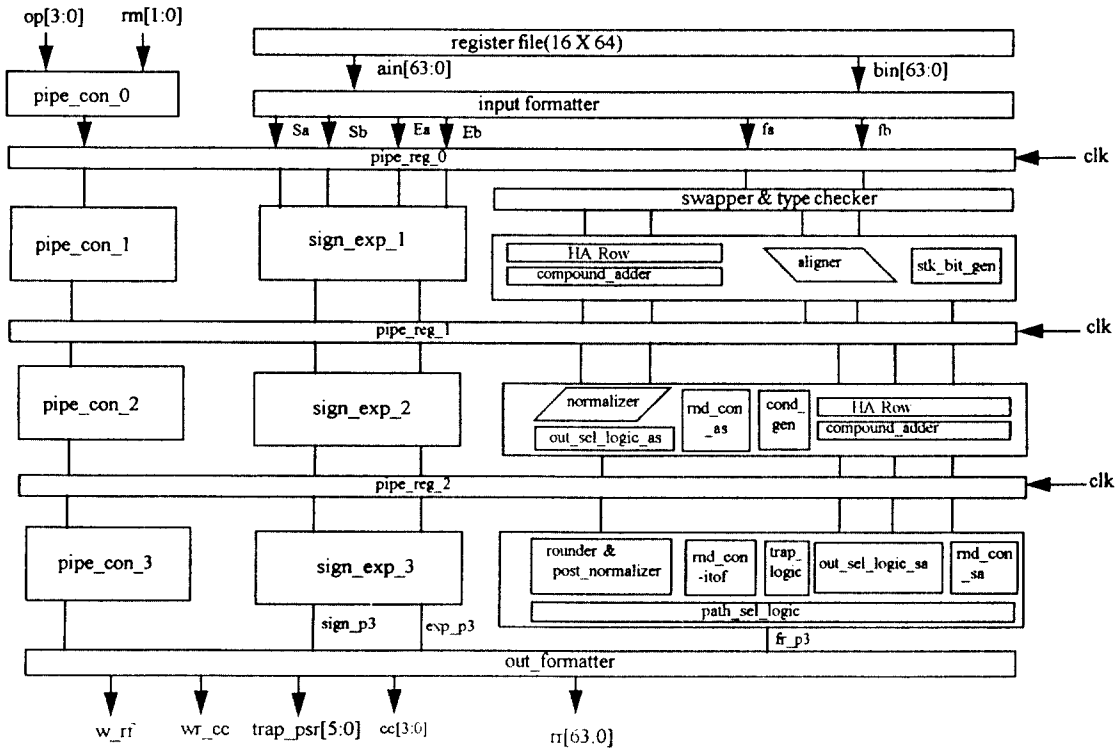

그림 7. FPAU의 전체 블록도
Fig. 7. Block diagram of FPAU

리는 1만큼 오차가 있기 때문에, 추가의 오차 조정을 위해 복합 가산기는 2가지 결과 ($A-B=A+B'+1$, $A-B-1=A+B'$)를 생성하고, 오차 신호(Cfine) 값에 따라 하나를 선택하는 방식을 취한다. 그리고 배정도 데이터와 단정도 데이터 사이의 변환은 지수 조정값(380h)을 사용한 덧셈 또는 뺄셈과 함께 반올림에 의한 분수부 오버플로우를 반영하는 회로가 필요한데, 이를 위한 가산기와 인크리멘트 회로가 존재한다. 그리고 파이프라인 단계 3의 경우 AS 경로에서의 분수부 오버플로우 조건과 함께 SA 경로에서 야기될 수 있는 1 비트 분수부 오버플로우와 1 비트 언더플로우를 대비하기 위한 인크리멘트 회로와 디크리멘트회로가 존재한다. 그리고 뺄셈 등의 동작시 지수부 값은 0이 아니지만 분수부가 0이 되어 0의 결과가 필요한 경우와 유한한값과 무한대(∞)의 값을 더하는 동작과 같이 결과값으로 무한대의 결과 생성이 필요한 경우가 있는데, 이를 위해 상수값을 하드웨어적으로 생성하는 MUX회로가 존재한다.

부호부는 결과의 부호값을 결정하는데, 파이프라인 단계 1의 경우에는 지수값차, 유효 연산(Eop) 또는

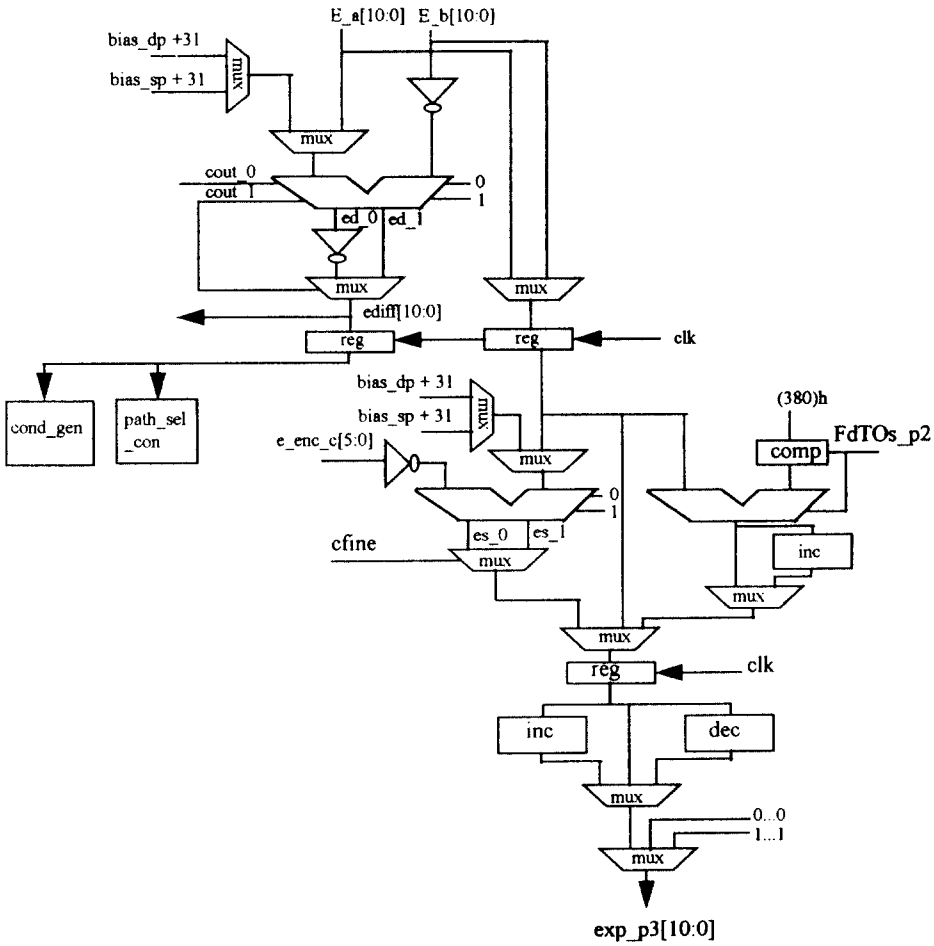

그림 8. 지수부 회로
Fig. 8. Exponent unit

표 4. 15가지 명령에 대한 아키텍쳐별 수행 특성
Table 4. Execution characteristics of architectures for fifteen instructions

명 령	SPARC FPU[3]		[2]		Proposed architecture	
	latency	throughput	latency	throughput	latency	throughput
FABSs	4	0.25	2	1	3	1
FADDd	5	0.2	3	1	3	1
FADDs	5	0.2	3	1	3	1
FCMPd	4	0.25	1	1	3	1
FCMPs	4	0.25	1	1	3	1
FdTOi	5	0.2	3	1	3	1
FdTOs	5	0.2	3	1	3	1
FiTOd	5	0.2	-	-	3	1
FiTOs	9	0.11	-	-	3	1
FMOVs	4	0.25	2	1	3	1
FNEGs	4	0.25	2	1	3	1
FsTOd	5	0.2	3	1	3	1
FsTOi	5	0.2	3	1	3	1
FSUBs	5	0.2	3	1	3	1
FSUBd	5	0.2	3	1	3	1

- : unsupported instruction
latency : numbers of clock cycle
throughput : numbers of clock cycle

연산 종류에 따라 Sa를 선택하거나 Sb를 선택한다. 그리고 FNEGs, FABSs 명령을 지원하기 위한 MUX와 XOR회로가 존재한다. 그리고 파이프라인 단계 2에는 지수값 차이가 0인 경우 뺄셈 동작 수행 결과 결과값이 음수로 나올 경우 파이프라인 단계 1에서 결정한 부호값을 반전시키는 회로가 존재한다. 그리고 파이프라인 단계 3에서는 0의 결과값에 대해 결과값에 무관하게 부호 필드를 0으로 만드는 회로가 존재한다. 제어부는 외부에서 곱셈 연산 종류(OP[3:0]), 반올림 양식(RM[1:0]), 그리고 내부에서 발생한 특별한 데이타 감지조건을 받아서, 데이타 패스에 필요한 제어 신호를 적절한 타이밍에 맞추어 발생시키는 파이프라인 구조의 데이타 정적 제어 회로이다.

V. 성능 분석과 설계 검증

본 연구에서 설계한 FPAU는 2가지 경로로 부동소수점 덧셈과 뺄셈을 비롯한 15가지 연산을 수행하므로 하드웨어가 기존 방식에 비해 다소 증가한다. 그러나 현재 반도체 기술의 발달로 집적도가 증가하게 됨에 따라 성능이 더 중시되므로 큰 문제가 되지 않는다. 그리고 설계된 회로는 표 3에 나타난 바와 같이 유효 덧셈(Eop)이 SA 경로를 사용하는데 비해 대부분의 다른 부동 소수점 연산이 AS 연산 경로를 사용하는 특성을 활용하여, 주어진 FPAU의 2가지 경로를 완전히 독립된 경로로 분리하고, 지수부와 입출력부분을 쌍으로 구성하면, 15가지 명령에 대해서도 스케쥴링을 통해 2-way 수퍼스칼라 구조를 구현할 수 있으므로 성능을 크게 향상시킬 수 있다. 표 4는 지원하는 15가지 명령에 대한 성능을 기존 프로세서와 비교한 결과를 나타낸다. 표 4에 보는 바와 같이 15가지

명령이 3단 파이프라인을 통해 수행되므로, 파이프라인 제어가 용이하다. 설계된 FPAU는 Verilog HDL 시뮬레이터[16]를 사용하여 레지스터 전달 단계, 게이트 단계로 검증을 한 후, IEEE 754 부동 소수점 표준안을 만족하는지 검증하기 위해, 미 Berkeley 대학에서 개발한 테스트 벡터(test vector)[17]를 사용하였다. 그리고 나서 설계된 회로는 전기적인 동작 특성을 구하기 위해, 미국 VLSI사의 0. 6μm CMOS 표준셀을 사용하여, 최악 전달 경로에 놓인 데이터 패스를 등가적으로 구현한후 지연 시간 특성을 분석해 보았다. TV(Timing Verifier)[18]를 통해 설계된 회로의 최악 지연 경로를 검증을 한 결과 파이프라인 단계 1의 경우 10[ns], 파이프라인 단계 2의 경우 8[ns], 그리고 파이프라인 단계 3은 6[ns]로 나타났다. 파이프라인 단계 1이 다소 큰 이유는 지수차 예측기, 분수부 지수부 선택기, 0의 값을 생성하는 MUX 등의 하드웨어에 기인한다. 따라서 본 연구에서 설계한 부동 소수점 연산회로는 대략 100Mhz의 동작 주파수를 가짐을 알수 있다. 위의 지연 특성을 고려하여 파이프라인 단계 2와 단계 3을 단일 단계로 통합하여 대략 70Mz의 동작 주파수를 갖는 2 단계 파이프라인 구조로 변형하는 것도 가능하다. 그리고 설계된 FPAU 회로는 표준 셀로 구현하게 되면, 대략 12만개의 트랜지스터로 구현될 수 있다. 본 논문에서 설계한 부동 소수점 연산회로는 다양한 형태의 명령 지원, 빠른 동작 주파수, 단정도와 배정도 데이터의 동일 하드웨어 지원 및 2개의 연산 경로를 사용한 파이프라인 구조 특성으로 수퍼 스칼라 마이크로프로세서에 내장하기에 적합한 구조를 갖고 있다.

Ⅵ. 결 론

본 논문에서는 2개의 연산 경로와 파이프라인 구조를 사용하여 고속의 동작이 가능하며, 새로운 반올림 및 정규화 기법을 사용한 부동 소수점 연산회로를 설계한 후 성능을 분석하였다. 설계된 연산회로는 대략 12만개의 트랜지스터로 구성되고 최악 조건에서 100Mhz의 동작 주파수를 갖고 있음이 확인되었다. 제안한 새로운 반올림 및 정규화 기법은 입력 데이터를 HA행과 복합 가산기를 활용하여 결과의 상위 비트와 하위 비트를 독립적으로 계산함에 따라, 반올림이 반영된 결과값의 종류를 2가지로 제한하여, 정규화와 반올림을 병렬적으로 수행하며, IEEE-754에서 정의한 4가지 반올림 모드를 시간 지연 없이 효율적으로 구현할 수 있다. 그리고 입력 데이타에서 결과값의 상위 0의 개수를 예측하는 기법을 통해 설계된 연산회로가 덧셈/뺄셈 동작이 완료된 후 바로 정규화가 가능하도록 하였다. 그리고 설계된 회로는 입력 오퍼랜드의 하위 비트에 존재하는 0의 개수를 계산하는 TOD회로와 지수값 차를 사용한 SA 경로에 대한 효율적인 스티키-비트 발생회로와 TOD값과 LOP결과를 이용한 AS 경로에 대한 스티키 비트 회로 등을 사용하여, 스티키 비트 결정이 반올림 동작의 최악 경로 상에 놓이는 것을 방지하도록 하여, 스티키 비트에 의한 시간 지연 문제를 해결하였다. 그리고 내부 아키텍처를 단정도 데이터, 배정도 데이터, 정수 데이터에 따라 적절히 MUX를 사용하여 조절하여, 동일 하드웨어상에서 하드웨어 오버헤드를 최소화하면서, 15가지 명령을 지원할 수 있다. 본 부동 소수점 연산 회로는 100 Mhz의 클럭 주파수로 최대 100 MFLOPS의 단정도 및 배정도 부동 소수점 연산 성능을 보여주며, 3단 파이프라인 구조를 갖고 있기 때문에, 다수개의 병렬 파이프라인으로 구성된 고성능의 수퍼스칼라 마이크로프로세서에서 부동 소수점 파이프라인 연산장치로 내장될 수 있다.

참 고 문 헌

1. ANSI/IEEE Standard 754-1985 for binary Floating-Point Arithmetic, IEEE Computer Society Press, Los Alamitos, Calf., 1985.
2. Nobuhiro Ide, etal, "A 320-MFLOPS CMOS Floating-Point Processing Unit for Superscalar Processors," IEEE Journal of Solid State Circuits, Vol, 28, No. 3, pp. 352-361, March 1993.
3. Cypress Semiconductor Inc, SPARC RISC User's Guide, 1990.
4. Donald C. Jackson, "RISC vs. CISC Floating Point Units," Computer Design, pp. 42, April, 1988.
5. S. Waser and M. J. Flynn, Introduction to Arithmetic for Digital System Designers, New-York:

Holts, 1982.

6. N. T. Quach, N. Takaki, and M.J. Flynn, "On fast IEEE Rounding," Tech. Rep. CSL-TR-91-459, Stanford University, March, 1991.
7. A. Katsuno, H. Takahashi, and H. Kubosawa, "A 64-bit Floating-point Processing Unit with a Horizontal Instruction Code," Proc. 1990 IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 347-350, 1990.
8. T. L. Chang and P. David Fisher, "High Speed Normalization and Rounding Circuits for VLSI Floating-Point Processor," IEEE International Conference on Circuit and Computer, pp. 512-516, 1980.
9. Vojin G. Oklobdzija, "An algorithm and Novel Design of a Leading Zero detector circuits: Comparison with Logic synthesis," IEEE Trans. on VLSI System, vol. 2, no. 1, pp. 124-128, March, 1994.
10. Bradley J. Benschneider and W. J. Bowhill, "A Pipelined 50-Mhz CMOS 64-bit Floating-Point Arithmetic Processor," IEEE J. Solid State Circuits, vol. 24, no. 5, pp. 1317-1323, October, 1989,
11. Stamatis Vassiliadis and David S. Lemon, "S/370 Sign-Magnitude Floating-Point Adder," IEEE J. Solid State Circuits, vol. 24, no. 4, pp. 1062-1070, August, 1989.
12. Hiroshige Fujii and Chikahiro, "A Floating-Point Cell Library and a 100-MFLOPS Image Signal Processor," IEEE J. of Solid State Circuits, vol. 27, no. 7, pp. 1080-1087, July, 1992.
13. E. Hokennek and R. K. Monoye, "Leading-zero anticipator(LZA) in the IBM RISC System/6000 floating-point execution unit," IBM J. Res. Develop, no. 1. pp. 71-77, 1990.
14. H. Suzuki and Y. Nakase, "Leading-One Anticipatory Logic for High-Speed Floating-Point Addition," IEEE 1995 Custom Integrated Circuit Conference, pp. 593-596, 1995.
15. J. T. Coonen, "An Implementation Guide to a Proposed Standard for Floating Point Arithmetic," IEEE Computer, pp. 68-79, vol. 13, no. 1, January 1980.
16. Donald E. Thomas and Philip Moorby, *The Verilog Hardware Description Language*, Kluwer Publisher, 1991.
17. Univ. of California, Berkeley, A compact test suite for P754 arithmetic-ver. 2.0, Computer Sci. Div., 1983.
18. VLSI Technology Inc, *TV(Timinig Verifier) software tool manual*, 1995.

최 병 윤(Byeong Yoon Choi) 정회원

1985년 2월: 연세대학교 전자공학과 졸업(공학사)
1987년 2월: 연세대학교 전자공학과 본대학원 졸업(공학 석사)
1992년 8월: 연세대학교 전자공학과 본대학원 졸업(공학 박사)
1993년 3월~현재: 동의대학교 컴퓨터공학과(조교수)
※주관심 분야: 수퍼스칼라 마이크로프로세서 설계, 신호처리 및 통신용 집적회로 설계

손 승 일(Seung IL Sonh) 정회원

1989년 2월: 연세대학교 전자공학과 졸업
1991년 8월: 연세대학교 전자공학과 졸업(공학 석사)
현재: 연세대학교 전자공학과 박사과정 재학중

이 문 기(Moon Key Lee) 정회원

1982년~현재: 연세대학교 전자공학과 교수 및 연세대학교 부설 아식 설계 공동 연구소 소장
※주관심 분야: 마이크로프로세서 설계, 디지탈 영상처리칩 설계, 설계 자동화 등

한국 통신학회 논문지 제20권 제7호, July, 1995 참조