



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2009-0032721
(43) 공개일자 2009년04월01일

(51) Int. Cl.

G06F 9/44 (2006.01) G06F 9/30 (2006.01)

(21) 출원번호 10-2007-0098184

(22) 출원일자 2007년09월28일

심사청구일자 2007년09월28일

(71) 출원인

고려대학교 산학협력단

서울 성북구 안암동5가1 고려대학교 내

(72) 발명자

추지호

서울 성북구 삼선동5가 174번지 302호

한영선

서울 성북구 안암동5가 159-2

김선욱

경기 남양주시 도농동 부영2차 214동 2003호

(74) 대리인

전중학

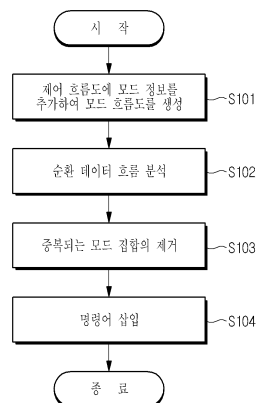
전체 청구항 수 : 총 13 항

(54) 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어최적화 방법

(57) 요약

본 발명은 복수의 명령어들을 실행하여 데이터를 처리하는 명령어 처리기에서, 상기 각 명령어의 기능을 확장시키는 모드 집합 명령어를 최적화시키는 방법에 관한 것으로, 특히 상기 복수의 명령어들을 처리하는 제어 흐름도의 각 노드에 상기 모드 집합 명령어에 대한 모드 정보를 추가하여 모드 흐름도를 생성하는 단계; 상기 생성된 모드 흐름도를 구성하는 상기 각 노드에 대해 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계; 및 상기 각 노드에 대해 상기 입력 모드 상태 및 출력 모드 상태의 변경 여부에 따라 해당 모드 집합 명령어를 삽입하는 단계;를 포함하는 것을 특징으로 한다.

대표도 - 도1



이 발명을 지원한 국가연구개발사업

과제고유번호	R0602842
부처명	(재)서울특별시 시정개발연구원
연구사업명	전략산업혁신클러스터육성지원
연구과제명	나노 SoC 산업육성을 위한 산학협력 혁신 클러스터2차년도(테라스케일 SoC 설계 위한 나노
IP DB구축)	
주관기관	광운대학교
연구기간	2006년 12월 01일 ~ 2007년 11월 30일

특허청구의 범위

청구항 1

복수의 명령어들을 실행하여 데이터를 처리하는 명령어 처리기에서, 상기 각 명령어의 기능을 확장시키는 모드 집합 명령어를 최적화시키는 방법에 있어서,

상기 복수의 명령어들을 처리하는 제어 흐름도의 각 노드에 상기 모드 집합 명령어에 대한 모드 정보를 추가하여 모드 흐름도를 생성하는 단계;

상기 생성된 모드 흐름도를 구성하는 상기 각 노드에 대해 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계; 및

상기 각 노드에 대해 상기 입력 모드 상태 및 출력 모드 상태의 변경 여부에 따라 해당 모드 집합 명령어를 삽입하는 단계;를 포함하는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 2

제1항에 있어서, 상기 명령어 처리기는,

디지털 신호 처리기인 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 3

제1항에 있어서, 상기 모드 흐름도의 상기 각 노드는,

하나의 명령어를 나타내는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 4

제1항에 있어서, 상기 순환 데이터 흐름 분석은,

상기 입력 모드 상태 및 출력 모드 상태의 변화가 없을 때까지 반복해서 수행하는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 5

제1항에 있어서, 상기 입력 모드 상태는,

상기 각 노드로 입력되는 모든 선행자의 출력 모드 상태들의 교집합으로 산출되는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 6

제1항에 있어서, 상기 출력 모드 상태는,

상기 해당 노드에서의 입력 모드 상태와 소멸되는 모드 상태의 차를 생성되는 모드 상태와 합집합하여 산출하는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 7

제6항에 있어서, 상기 생성되는 모드 상태가 비어 있을 경우,

상기 소멸되는 모드 상태는 공집합으로 정의되는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 8

제6항에 있어서, 상기 생성되는 모드 상태가 비어 있지 않을 경우,

상기 소멸되는 모드 상태는 상기 입력 모드 상태와 같게 되는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 9

제1항에 있어서, 상기 순환 데이터 흐름 분석은,

각 순환 주기마다 상기 각 노드의 상기 입력 모드 상태 및 출력 모드 상태가 임시 공간에 저장되는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 10

제1항에 있어서, 상기 모드 집합 명령어는,

부호 확장 모드인 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 11

제1항에 있어서, 상기 모드 집합 명령어는,

결과물 이동 모드인 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 12

제1항에 있어서, 상기 방법은,

상기 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계에서,

상기 모드 흐름도 상에서 역방향으로의 진행이 존재할 경우, 중복되는 모드 집합을 제거하는 단계;를 더 포함하는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

청구항 13

제12항에 있어서, 상기 중복되는 모드 집합의 제거 방법은,

상기 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계에서, 상기 역방향으로 진행하여 입력되는 모드 정보 중 입력 모드 상태를 산출하기 위한 교집합 연산 과정에서 소실된 정보를 반영하여 구현하는 것을 특징으로 하는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법.

명세서

발명의 상세한 설명

기술 분야

- <1> 본 발명은 명령어 처리기에 관한 것으로, 보다 상세하게는 모드 집합을 사용하는 명령어 처리기에서 모드 명령어를 최적화하는 방법에 관한 것이다.

배경 기술

- <2> 디지털 신호 처리기(Digital Signal Processor; 이하, 'DSP'라 한다.)는 멀티미디어 데이터를 매우 빠르게 처리할 수 있고, 파워 소모도 적도록 개발되었기 때문에 멀티미디어 장비의 확산과 더불어 임베디드(embedded) 시스템에 널리 사용되고 있는 프로세서이다. 하지만, 신호 처리를 실시간으로 하기 위해서 DSP에서는 매우 복잡한 명령어 집합을 제공한다. 이러한 복잡성은 컴파일러가 좋은 코드를 생성하는데 방해가 된다.
- <3> 한편, 상기 DSP에서 모드 집합(mode set)은 상기 명령어에 확장된 기능성을 제공한다. 따라서, 하나의 명령어는 상기 모드 집합에 따라서 다른 기능을 수행할 수가 있게 된다.
- <4> 상기 모드 집합의 예로서 부호 확장 모드(Sign eXtension Mode; 이하, 'SXM' 이라 한다.) 또는 결과물 이동 모드(product shift mode) 등이 있다. 이러한 모드 집합은 DSP 아키텍처에서 명령어 집합을 최소화하기 위해 사용된다. 즉, 하나의 명령어에 대해서 프로세서는 모드 집합의 모드 상태에 따라 다른 기능을 수행할 수가 있게 된

다.

- <5> 널리 사용되는 모드 중 하나인 상기 SXM의 경우, SXM의 상태 값에 따라 산술 연산 명령어가 부호 연산(sign operation)을 수행하거나 무부호 연산(unsigned operation)을 수행한다. 다음 알고리즘은 'Zaram compactDSP16 아키텍처'의 명령어 중 하나를 나타내고 있다.
- <6> SUB ACC, LOC << ShiftCode
- <7> If SXM=0,
- <8> [ACC]-unsigned[addresslocation]×2^{ShiftCode}
- <9> If SXM=1,
- <10> [ACC]-signed[addresslocation]×2^{ShiftCode}
- <11> 상기 알고리즘에서 만약 SXM 이 설정되어 있지 않으면 SUB 명령어는 무부호 연산을 수행하고, 설정되어 있으면 부호 연산을 수행한다. 이때, 컴파일러는 SXM 명령어를 산술 연산이 일어날 때마다 삽입할 수도 있지만, 지난 SXM 상태가 현재 명령어가 원하는 SXM 상태와 같다면, 현재 명령어를 위해 SXM 명령어를 삽입할 필요가 없게 된다.
- <12> 최근에는 이러한 모드 명령어를 데이터 흐름 그래프(Dataflow Analysis Graph; DAG)를 스케줄링함으로써 최소화 하는 방법이 제안되었다. 그러나 이러한 DAG를 이용한 최적화 방법을 사용하기 위해서는 그래프 자체가 변환되므로 더욱 복잡한 알고리즘이 적용되어야 하는 문제점이 있었다.
- <13> 따라서, 제어 흐름도(control flow graph; CFG)를 수정하지 않고서도 데이터 흐름 분석을 통해 모드 명령어를 효과적으로 최적화할 수 있는 방법이 요구되고 있는 실정이다.

발명의 내용

해결 하고자하는 과제

- <14> 본 발명의 목적은 제어 흐름도를 수정하지 않고서도 디지털 신호 처리기와 같은 명령어 처리기에서 사용되는 모드 명령어를 최적화할 수 있는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법을 제공함에 있다.
- <15> 또한, 본 발명의 목적은 데이터 흐름 분석을 통해 디지털 신호 처리기의 명령어 집합에서 사용되는 모드 명령어를 최적화할 수 있는 모드 집합을 사용하는 명령어 처리기에서의 모드 명령어 최적화 방법을 제공함에 있다.

과제 해결수단

- <16> 상기한 목적을 달성하기 위한 본 발명의 방법은, 복수의 명령어들을 실행하여 데이터를 처리하는 명령어 처리기에서의 모드 명령어 최적화 방법에서, 상기 각 명령어의 기능을 확장시키는 모드 집합 명령어를 최적화시키는 방법에 있어서, 상기 복수의 명령어들을 처리하는 제어 흐름도의 각 노드에 상기 모드 집합 명령어에 대한 모드 정보를 추가하여 모드 흐름도를 생성하는 단계; 상기 생성된 모드 흐름도를 구성하는 상기 각 노드에 대해 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계; 및 상기 각 노드에 대해 상기 입력 모드 상태 및 출력 모드 상태의 변경 여부에 따라 해당 모드 집합 명령어를 삽입하는 단계;를 포함하는 것을 특징으로 한다.
- <17> 상기 모드 흐름도의 상기 각 노드는 하나의 명령어를 나타내는 것을 특징으로 한다.
- <18> 상기 순환 데이터 흐름 분석은 상기 입력 모드 상태 및 출력 모드 상태의 변화가 없을 때까지 반복해서 수행하는 것을 특징으로 한다.
- <19> 상기 입력 모드 상태는 상기 각 노드로 입력되는 모든 선행자의 출력 모드 상태들의 교집합으로 산출되는 것을 특징으로 하며, 상기 출력 모드 상태는 상기 해당 노드에서의 입력 모드 상태와 소멸되는 모드 상태의 차를 생성되는 모드 상태와 합집합하여 산출하는 것을 특징으로 한다.
- <20> 상기 생성되는 모드 상태가 비어 있을 경우 상기 소멸되는 모드 상태는 공집합으로 정의되는 것을 특징으로 하며, 상기 생성되는 모드 상태가 비어 있지 않을 경우 상기 소멸되는 모드 상태는 상기 입력 모드 상태와 같게

되는 것을 특징으로 한다.

- <21> 상기 순환 데이터 흐름 분석은 각 순환 주기마다 상기 각 노드의 상기 입력 모드 상태 및 출력 모드 상태가 임시 공간에 저장되는 것을 특징으로 한다.
- <22> 상기 모드 집합 명령어는 부호 확장 모드 또는 결과물 이동 모드인 것을 특징으로 한다.
- <23> 상기 방법은 상기 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계에서 상기 모드 흐름도 상에서 역방향으로의 진행이 존재할 경우, 중복되는 모드 집합을 제거하는 단계;를 더 포함하는 것을 특징으로 한다.
- <24> 상기 중복되는 모드 집합의 제거 방법은, 상기 입력 모드 상태 및 출력 모드 상태를 순환 데이터 흐름 분석에 의해 산출하는 단계에서, 상기 역방향으로 진행하여 입력되는 모드 정보 중 입력 모드 상태를 산출하기 위한 교집합 연산 과정에서 소실된 정보를 반영하여 구현하는 것을 특징으로 한다.

효 과

- <25> 본 발명에 따르면, 제어 흐름도에 기반을 둔 모드 명령어 최적화를 위해 모드 흐름도를 생성하여 집합 명령어에 의해 모드 상태를 분석함으로써 제어 흐름도를 수정하지 않고서도 모드 집합 명령어의 수를 최소로 할 수 있게 되는 장점이 있다.

발명의 실시를 위한 구체적인 내용

- <26> 본 발명은 DSP와 같은 모드 집합을 사용하는 명령어 처리기에서의 명령어 집합에서 제어 흐름도(Control flow graph; CFG)에 기반을 둔 모드 셋 최적화를 위한 데이터 흐름 분석 방법을 제안한다.
- <27> 상기 본 발명에 따라 코드 생성과 다른 최적화를 위해 사용되는 제어 흐름도(control flow graph)를 수정할 필요가 없이 데이터 흐름 분석을 통해 이러한 최적화를 매우 간단하게 구현할 수 있게 된다.
- <28> 이하 설명에서는 먼저 본 발명에 따른 모드 최적화를 위한 데이터 흐름 분석 방법을 설명하며, 다음으로 제안된 알고리즘을 SXM 명령어에 적용시킨 실시 예를 설명하기로 한다.
- <29> 이하 본 발명의 바람직한 실시 예에 따른 상세한 설명을 첨부된 도면들을 참조하여 설명한다. 하기에 본 발명을 설명함에 있어 관련된 공지 기능 또는 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략할 것이다.
- <30> 한편, 본 발명은 모드 집합을 사용하는 어떠한 명령어 처리기에도 적용이 가능하며, 후술하는 설명에서는 상기 명령어 처리기 중에서 대표적인 디지털 신호 처리기(즉, DSP)를 예로 들어 설명하기로 한다.
- <31> 도 1은 본 발명에 따른 디지털 신호 처리기의 명령어 집합에서 모드 명령어 최적화 절차를 나타낸 흐름도이다. 상기 도 1을 참조하면, 본 발명에 따른 모드 명령어 최적화 절차로서 먼저 제어 흐름도에 모드 정보를 추가하여 모드 흐름도(Mode Flow Graph; MFG)를 생성(S101)하게 된다.
- <32> 그런 다음, 상기 생성된 모드 흐름도에 대해 순환 데이터 흐름 분석(S102)을 수행하게 된다.
- <33> 추가적으로 본 발명의 실시 예에 따라 중복되는 모드 집합을 제거(S103)한 후, 모드 집합 명령어를 삽입(S104)함으로써 모드 명령어 최적화 절차가 완료된다.
- <34> 이하, 상술한 각 단계의 절차를 보다 상세히 설명하기로 한다.
- <35> <제1 단계 : 모드 흐름도 생성>
- <36> 본 발명에 따른 모드 명령어 최적화를 위해 생성하는 모드 흐름도(Mode Flow Graph)는 제어 흐름도(Control flow graph)와 유사하지만 다음과 같은 차이점이 있다.
- <37> 1. 제어 흐름도에서 하나의 노드(node)가 하나의 베이직 블록(basic block)을 나타내는 것과는 다르게, 모드 흐름도에서 하나의 노드(node)는 하나의 명령어를 나타낸다.
- <38> 2. 모드 흐름도의 각각의 노드에는 추가적인 정보가 들어가게 되는데 이를 모드 정보(mode information)라 한다. 상기 모드 정보는 모드 집합 명령어당 최대 하나씩 정의된다. 만약 노드가 어떤 모드 집합도 가지지 않는다면, 그 노드는 'Ø' 상태를 가진다. 모드 하나에 N+1 개의 모드 상태가 가능하고, 각각의 모드는 M_i

로 표현된다. ($i=0,1,2,\dots,N-1$)

- <39> 3. 모드 정보는 제어 흐름에 따라 전달된다.
- <40> 도 2는 본 발명의 실시 예에 따라 생성되는 나타낸 모드 흐름도이다. 상기 도 2를 참조하면, 본 발명에 따라 생성되는 모드 흐름도는, DSP의 각 명령어를 노드로 하여, 상기 각 명령어의 실행 흐름에 따라 제어 흐름도와 유사한 형태로 생성된다. 예컨대, 1번 노드부터 10번 노드까지 화살표가 가리키는 방향으로 명령어가 순차적으로 실행된다. 이때, 상술한 바와 같이 상기 각 노드에는 모드 정보(예컨대, $\emptyset$, M_0 , M_1 등)가 삽입된다.
- <41> 상기 도 2를 참조하면, 제1 노드의 모드 정보는 ' $\emptyset$ '이고, 제2 노드의 모드 정보는 ' M_0 '이다. 따라서, 제1 노드의 명령어가 실행된 후, 제2 노드의 명령어가 실행될 때에는 모드 정보가 ' M_0 '로 변환되어야 한다. 이에 따라, 상기 제2 노드에는 ' M_0 '로 모드 정보를 변경하는 모드 집합 명령어가 삽입되어야 한다.
- <42> 동일한 방법으로서, 제5 노드, 제7 노드, 제8 노드, 제9 노드 등에도 모드 집합 명령어가 삽입되어야 한다. 상기 각 노드와 같이 모드 집합 명령어가 삽입되는 5개의 노드는 도 2에서 음영으로 표시된 노드이다.
- <43> 한편, 이와 같이 최초 생성된 모드 흐름도에서는 모드 집합 명령어가 변경되는 각 노드마다 모드 집합 명령어를 삽입하여 최대 5개의 모드 집합 명령어가 삽입(worst case)되어야 하나, 후술하는 데이터 흐름 분석(3개의 모드 집합 명령어 삽입) 및 중복되는 모드 집합 제거 방법(2개의 모드 집합 명령어 삽입)에 의해 모드 집합 명령어를 최소한으로 줄여 최적화할 수가 있게 된다.
- <44> <제2 단계 : 순환 데이터 흐름 분석>
- <45> 본 발명에서는 순환 데이터 흐름 분석을 위해 두 개의 집합 ' $In[n]$ ' 및 ' $Out[n]$ '을 다음과 같이 정의한다.
- <46> (1) $In[n] = \{\text{노드 } n \text{으로 들어가는 모드 상태}\}$
- <47> (2) $Out[n] = \{\text{노드 } n \text{의 상속자(successor)에게 전달되는 모드 상태}\}$
- <48> 또한, 모드 집합 명령어의 실행이 최소가 되게 하기 위해서 각각의 노드 n 에서의 상기 In 과 Out 집합을 하기 <수학식 1> 및 <수학식 2>와 같이 정의한다.

수학식 1

<49>
$$Out[n] = Gen[n] \cup (In[n] - Kill[n])$$

수학식 2

<50>
$$In[n] = \bigcap_{q \in PRED[n]} Out[q]$$

- <51> 상기 <수학식 2>를 참조하면, $In[n]$ 은 노드 n 의 모든 선행자의 Out 의 교집합으로 정의된다. 그리고 상기 <수학식 1>을 참조하면, $Out[n]$ 은 $Gen[n]$, $In[n]$, 그리고 $Kill[n]$ 에 의해 정의된다.
- <52> 이때, 상기 $Gen[n]$ 은 노드 n 에서 생성되는 모드 상태($\emptyset$ 또는 M_i)이고, $Kill[n]$ 은 노드 n 에서 소멸되는 모드 정보이다. 만약 $Gen[n]$ 이 비어있지 않다면 ($Gen \neq \emptyset$), $Kill[n]$ 은 $In[n]$ 과 같고, 만약 비어 있으면 $Kill[n]$ 은 $\emptyset$ 로 정의된다.
- <53> 상기 <수학식 1>에서와 같이 데이터 흐름 방정식의 $Out[n]$ 을 계산하기 위해서는 합집합과 차집합이 사용된다. 만약 노드 n 에 어떤 모드 집합 M_i 가 있으면, 그 모드 정보는 노드 n 의 상속자(successor)에게 전달되어야 한다. 그렇지 않을 경우에는, 노드 n 에 들어오는 모드 상태 정보가 노드 n 의 상속자에게 전달될 필요가 있다. 이를 바탕으로, 상기 $Out[n]$ 은 $Kill[n]$ 을 적절히 조절하여 계산되게 된다. 따라서, $Gen[n]$, $In[n]$, $Kill[n]$ 및 $Out[n]$ 의 조합은 하기 <표 1>과 같다.

표 1

<54>

State of n	In[n]	Kill[n]	Gen[n]	Out[n]
$\emptyset$	M_i	$\emptyset$	$\emptyset$	M_i
M_i	M_i	M_i	M_i	M_i
M_i	M_j	M_j	M_i	M_i
M_i	$\emptyset$	$\emptyset$	M_i	M_i
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

<55>

이때, 상기 <표 1>에서 i와 j는 같지 않다.

<56>

한편, 본 발명에서 제안하고 있는 모드 최적화를 위한 데이터 흐름 분석의 전체 과정은 다음과 같다.

<57>

for each n

<58>

In[n] = $\emptyset$

<59>

Out[n] = $\emptyset$

<60>

for each n

<61>

In'[n] = In[n]

<62>

Out'[n] = Out[n]

<63>

Out[n] = Gen[n] $\cup$ (In[n]-Kill[n])

<64>

In[n] = $\bigcap_{q \in \text{PRED}[n]} \text{Out}[q]$

<65>

until In'[n] = In[n] & Out'[n] = Out[n]

<66>

상기 알고리즘을 참조하면, 먼저 초기화 단계에서 각 노드의 In 과 Out 집합은 $\emptyset$ 로 초기화된다. 데이터 흐름 분석은 역방향으로 이루어지며, 각 순환 주기마다 각 노드의 In과 Out은 임시 공간에 저장된다. 이렇게 저장된 정보는 순환 종료의 시기를 결정하는데 사용된다. 즉, 상기 분석은 모든 노드의 In과 Out에 변화가 없을 때까지 계속된다. 한편, 제안하고 있는 알고리즘의 복잡도(complexity)는 $O(n^4)$ 이고, 여기서 n은 모드 흐름도(MFG)의 노드 개수이다. 상기 복잡도는 생존성 분석(Liveness analysis)에서의 복잡도와 같게 된다.

<67>

상기 알고리즘의 과정을 실제로 적용시킨 예가 도 3 및 <표 2>에 제시된다.

<68>

즉, 상기 도 2에 도시된 모드 흐름도에 상술한 본 발명에 따른 데이터 흐름 분석 알고리즘을 적용하면 하기 <표 2>와 같이 된다.

표 2

	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5	
Node	Out	In	Out	In	Out	In	Out	In	Out	In
1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
2	M ₀	∅	M ₀	∅	M ₀	∅	M ₀	∅	M ₀	∅
3	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
4	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
5	M ₁	∅	M ₁	∅	M ₁	∅	M ₁	∅	M ₁	∅
6	∅	∅	∅	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁
7	M ₁	∅	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁
8	M ₀	∅	M ₀	∅	M ₀	∅	M ₀	M ₁	M ₀	M ₁
9	M ₁	∅	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁	M ₁
10	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

<69>

<70>

상기 <표 2>를 참조하면, 제10 노드부터 제1 노드까지 상술한 알고리즘에 의해 각 노드의 In과 Out을 산출하여 임시 공간에 저장한다. 이와 같은 절차를 반복해서 수행하게 되면, 모든 노드에 대한 In 및 Out 값이 변하지 않는 때가 발생한다. 즉, 상기 <표 2>에서는 5번째 반복 연산(Iteration 5)의 결과가 4번째 반복 연산(Iteration 4)의 결과와 같음을 알 수 있다. 따라서, 상기 5번째 반복 연산에서 알고리즘을 종료하게 된다.

<71>

한편, 상기 5번째 반복 연산의 최종 결과에서 In과 Out의 모드가 변경된 노드를 살펴보면, 제2 노드, 제5 노드, 제8 노드가 됨을 알 수 있다. 따라서, 하기 <표 3>에서와 같이 모드가 변경되는 각 노드에 모드를 변경시키는 모드 집합 명령어를 삽입하게 된다.

표 3

<72>

Node n	Out[n]	In[n]	Insertion
1	∅	∅	
2	M ₀	∅	Instruction for M ₀
3	∅	∅	
4	∅	∅	
5	M ₁	∅	Instruction for M ₁
6	M ₁	M ₁	
7	M ₁	M ₁	
8	M ₀	M ₁	Instruction for M ₀
9	M ₁	M ₁	
10	∅	∅	

<73>

상기 <표 3>을 참조하면, 총 3개의 모드 집합 명령어가 필요하게 되며, 이는 도 3에 음영으로 표시된 부분이다. 이를 상술한 도 2(즉, 순환 데이터 흐름 분석 방법 적용 전)와 비교하면, 모드 집합 명령어가 5개에서 3개로 줄어든 것을 알 수 있다.

<74>

<제3 단계: 중복되는 모드 집합 제거>

<75> 모드 흐름도 상에서 역방향으로의 진행은 역방향으로 진행되어 입력되는 새로운 노드와의 교집합 연산으로 인해 예기치 못한 결과가 발생할 수 있다. 따라서, 상기와 같은 상황을 고려하여 본 발명의 실시 예에 따라 모드 명령어 최적화를 위해 중복되는 모드 집합의 제거가 수행될 수 있다.

<76> 이를 도 4를 참조하여 상세히 설명하면 다음과 같다.

<77> 도 4는 본 발명의 실시 예에 따른 역방향 진행으로 인해 발생하는 중복 모드 집합을 나타낸 모드 흐름도이다. 상기 도 4를 참조하면, 본 발명에 따라 제안된 분석을 실행하면 노드 2에서 $\emptyset$ 와 M_i 의 In 과 Out 을 얻게 된다. 즉, 상기 도 4에 알고리즘을 적용하면 하기 <표 4>와 같이 된다.

표 4

Node	Iteration 1		Iteration 2	
	Out	In	Out	In
1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2	M_0	$\emptyset$	M_0	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

<79> 그러나 상기 도 4에 따르면 최적의 결과를 얻기 위해서는 모든 노드의 In 과 Out 집합은 전부 $\emptyset$ 이거나 M_i 이어야 한다는 것을 알 수 있다.

<80> 이러한 예상하지 못한 결과는 노드 1과 노드 4의 In 집합을 계산할 때 생기는 교집합 때문에 발생한다. 이 현상은 모드 흐름도에서 역방향으로의 진행이 있을 때만 발생하며, 역방향 진행에 관계한 두 노드가 모두 $\emptyset$ 이어야 한다. 이러한 예상치 않은 결과는 교집합 연산만을 통해서는 최적화된 된 결과를 얻을 수 없음을 말해준다.

<81> 따라서, 상기 문제를 해결하기 위해서, 본 발명의 실시 예에 따라 교집합을 수행하는 과정에서 어떠한 모드 정보가 소실되었는지를 $\emptyset$ 에 아래 첨자 형태로 정보를 추가하였다. 만약, 상기 도 4의 제1 노드에서와 같이 모드 M_i 와 $\emptyset$ 가 교집합을 수행하게 되면 결과물은 $\emptyset$ 가 아닌 $\emptyset_{mi}$ 가 되게 된다. 이때, $\emptyset$ 의 아래 첨자는 단지 $\emptyset$ 가 숨겨진 정보를 가지고 있음을 의미한다.

<82> 한편, 상기와 같이 추가되는 아래 첨자는 분석 과정에서 집합 연산에 대해서는 어떠한 영향도 주어서는 안 되고, 오직 다른 아래 첨자에만 영향을 주어야 한다. 다시 말하면, 아래 첨자를 제외한 모든 연산은 상술한 제2 단계에서의 집합 연산과 동일하게 동작한다. 아울러, 상기 연산 과정에서 $\emptyset$ 의 아래 첨자는 전달되거나 생성되거나 소멸될 수 있다. 모드 i 와 j ($i \neq j$)의 아래 첨자를 포함한 집합 연산의 관계는 하기 <수학식 3>과 같다.

수학식 3

$$\begin{aligned} \emptyset - \emptyset_{M_i} &= \emptyset \\ \emptyset_{M_i} - \emptyset &= \emptyset_{M_i} \\ \emptyset_{M_i} - \emptyset_{M_j} &= \emptyset_{M_i} \\ \emptyset_{M_i} \cup \emptyset &= \emptyset_{M_i} \text{ (Transmit)} \\ \emptyset_{M_i} \cup \{M_i\} &= \{M_i\} \\ \{M_i\} \cap \emptyset &= \emptyset_{M_i} \text{ (Create)} \\ \{M_i\} \cap \emptyset_{M_i} &= \emptyset_{M_i} \\ \{M_i\} \cap \emptyset_{M_j} &= \emptyset \\ \emptyset_{M_i} \cap \emptyset_{M_j} &= \emptyset \text{ (Eliminate)} \\ \emptyset_{M_i} \cap \emptyset_{M_i} &= \emptyset_{M_i} \\ \emptyset_{M_i} \cap \emptyset &= \emptyset \end{aligned}$$

<83>

<84>

분석의 모든 단계가 끝나면 $\emptyset_{M_i}$ 는 M_i 로 간주할 수 있다. 왜냐하면, M_i 의 정보는 분석 중에 숨겨졌지만 분석이 끝난 단계에서도 영향을 주고 있기 때문이다.

<85>

따라서, 상기 도 4를 분석한 <표 4>를 새로 추가된 아래 첨자에 의해 다시 분석하면 하기 <표 5>와 같이 된다.

표 5

Node	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5		Modified Information	
	Out	In	Out	In	Out	In	Out	In	Out	In	Out	In
1	$\emptyset$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	M_0	M_0
2	M_0	$\emptyset$	M_0	$\emptyset$	M_0	$\emptyset_{M_0}$	M_0	$\emptyset_{M_0}$	M_0	$\emptyset_{M_0}$	M_0	M_0
3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	M_0	M_0
4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	M_0	M_0

<86>

<87>

상기 <표 5>의 결과를 보면 노드 2의 In 은 $\emptyset_{M_0}$ 정보를 가지고 있다. 이것은 $\emptyset$ 와 M_0 의 교집합 연산이 있었음을 의미하며 숨겨진 M_0 정보가 노드 2로 들어가고 있음을 말해준다. 따라서, 모든 노드의 In 과 Out은 M_0 로 생각될 수 있다 즉, <표 4>에서는 제2 노드에 모드 집합 명령어가 삽입되어야 하지만, 상술한 바와 같이 아래 첨자 연산에 의해 중복되는 모드 집합 제거가 이루어지면 <표 5>에서와 같이 제2 노드에서도 모드 집합 명령어가 삽입될 필요가 없어지게 된다.

<88>

따라서, 상술한 <표 2>를 중복되는 모드 집합 제거 방법에 의해 다시 산출하면 하기 <표 6>과 같이 된다.

표 6

	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5		Modified Information	
Node	Out	In	Out	In	Out	In	Out	In	Out	In	Out	In
1	$\emptyset$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	M_0	M_0
2	M_0	$\emptyset$	M_0	$\emptyset$	M_0	$\emptyset_{M_0}$	M_0	$\emptyset_{M_0}$	M_0	$\emptyset_{M_0}$	M_0	M_0
3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	M_0	M_0
4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	$\emptyset_{M_0}$	M_0	M_0
5	M_1	$\emptyset$	M_1	$\emptyset$	M_1	$\emptyset$	M_1	$\emptyset_{M_0}$	M_1	$\emptyset_{M_0}$	M_1	M_0
6	$\emptyset$	$\emptyset$	$\emptyset$	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1
7	M_1	$\emptyset$	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1
8	M_0	$\emptyset$	M_0	$\emptyset$	M_0	$\emptyset$	M_0	M_1	M_0	M_1	M_0	M_1
9	M_1	$\emptyset$	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1
10	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

<89>

<90>

상기 <표 6>을 참조하면, 제5 노드 및 제8 노드에서만 모드 명령어 변화가 발생하므로 2개의 모드 집합 명령어 만으로도 각 노드에서의 모드 집합을 만족시킬 수가 있게 된다. 따라서, 이를 모드 흐름도에 적용하면 도 5와 같이 나타낼 수 있다.

<91>

<제4 단계: 명령어 삽입(Instruction Insertion)>

<92>

모드 집합 명령어는 모드 상태를 정하기 위해서 사용된다. 그리고 모드 최적화의 목표는 실행되는 모드 집합 명령어의 수를 최소화하는데 있다. 따라서, 모드 명령어가 어디에 삽입이 되어야 하는지 정확하게 알기 위해서는 각 노드의 In과 Out 집합을 비교해 볼 필요가 있다. 만약 $Out[n] \neq In[n]$ 이라면, $Out[n]$ 을 위해서 모드 집합 명령어를 삽입해야 한다. 하기 <표 7>은 상기 <표 6>에서의 결과에 대한 명령어 삽입을 나타내고 있다.

표 7

<93>

Node n	Out[n]	In[n]	Insertion
1	M_0	M_0	
2	M_0	M_0	
3	M_0	M_0	
4	M_0	M_0	
5	M_1	M_0	Instruction for M_1
6	M_1	M_1	
7	M_1	M_1	
8	M_0	M_1	Instruction for M_0
9	M_1	M_1	
10	$\emptyset$	$\emptyset$	

<94>

상기 <표 7>을 참조하면, 제5 노드 및 제8 노드에서만 모드 상태 변경이 발생하므로, 2번의 모드 집합 명령어만 이 필요하게 된다. 따라서, 동일한 명령어 흐름에서 최초 5개의 모드 집합 명령어를 사용하던 것이 본 발명에 따른 순환 데이터 흐름 분석을 통해 3개의 모드 집합 명령어만을 사용하게 되며, 추가적으로 중복되는 모드 집합 제거 방법에 의해 2개의 모드 집합 명령어만을 사용해서 구현할 수가 있게 된다.

<95> 이를 정리하면 하기 <표 8>과 같이 된다.

표 8

<96>

	Worst case	순환 데이터 분석	중복되는 모드 제거
명령어 삽입 지점	2, 5, 7, 8, 9	2, 5, 8	5, 8
정적 명령어 개수	5	3	2
worst case 대비 감소 비율(%)	0	40	60

실시예

<97>

이하, 상술한 본 발명을 SXM 명령어에 적용시킨 실시 예를 설명하기로 한다. 또한, 후술하는 실시 예에서 실행되는 명령어의 수와 코드 크기의 측면에서 성능 테스트를 비교 분석하기로 한다.

<98>

<SXM을 위한 모드 흐름도>

<99>

부호 확장 모드(Sign eXtesnsion Mode)에서는 오직 설정과 해제 두 개만이 존재한다. 따라서 SXM은 다음과 같은 세 가지 상태를 가질 수 있다.

<100>

(1) M_0 : SXM 에서 SXM_UNSET 상태를 필요로 하는 노드

<101>

(2) M_1 : SXM 에서 SXM_SET 상태를 필요로 하는 노드

<102>

(3) $\emptyset$: SXM 의 상태를 필요로 하지 않는 노드

<103>

설명의 편의를 위해 상기 M_0 를 U(Unset)라고 표기하고, M_1 을 S(Set)라고 표기하기로 한다. 이를 모드 흐름도로 나타내면 도 6과 같이 나타낼 수 있다.

<104>

한편, 상술한 바와 같이 중복되는 모드 집합 제거를 위해 두 개의 아래 첨자 $\emptyset_U$ 와 $\emptyset_S$ 가 필요하다.

<105>

<SXM을 위한 데이터 흐름 분석>

<106>

분석을 위해서는 각각의 노드 n의 Gen 과 Kill 집합을 정확히 할 필요가 있다. 이는 하기 <표 9>와 같이 정의될 수 있다.

표 9

<107>

State	Kill[n]	Gen[n]
$\emptyset$	$\emptyset$	$\emptyset$
U	S	U
S	U	S

<108>

또한, In 과 Out의 관계는 상술한 <수학식 1> 및 <수학식 2>에 의해 하기 <표 10>과 같이 산출될 수 있다.

표 10

<109>

State of n	In[n]	Kill[n]	Gen[n]	Out[n]
$\emptyset$	U	$\emptyset$	$\emptyset$	U
U	U	S	U	U
S	U	U	S	S
$\emptyset$	S	$\emptyset$	$\emptyset$	S
U	S	S	U	U
S	S	U	S	S
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
U	$\emptyset$	S	U	U
S	$\emptyset$	U	S	S

<110> 상기 <표 10>을 참조하여, 상기 도 6의 모드 흐름도를 순환 데이터 흐름 분석하면, 하기 <표 11>과 같이 된다.

표 11

Node	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5		Iteration 6		Modified Information	
	Out	In	Out	In	Out	In	Out	In	Out	In	Out	In	Out	In
1	o	U	U	U	U	U	U	U	U	U	U	U	U	U
2	S	o	S	o	S	U	S	U	S	U	S	U	S	U
3	U	o	U	o	U	U	U	U	U	U	U	U	U	U
4	S	o	S	o	S	o	S	o	S	o	S	o	S	o
5	o	o	o	o _S	o _S	o _S	o _S	o _S	o _S	o _S	o _S	o _S	S	S
6	S	o	S	o	S	o	S	o _S	S	o _S	S	o _S	S	S
7	o	o	o	o	o	o	o	o _S	o _S	o _S	o _S	o _S	S	S
8	o	o	o	o _S	o _S	o _S	o _S	o _S	o _S	o _S	o _S	o _S	S	S
9	U	o	U	o	U	o	U	o _S	U	o _S	U	o _S	U	S

<111>

<112> <SXM을 위한 명령어 삽입>

<113> 상기 SXM을 위한 모드 집합 명령어는 두 개가 있다. SXM_UNSET은 모드를 해제하기 위한 명령어이고, SXM_SET은 모드를 설정하기 위한 명령어이다. 상기 <표 11>의 분석의 결과를 통해서, 하기 <표 12>과 같이 언제 모드 명령어 삽입이 필요한지를 알 수가 있게 된다.

표 12

Node n	Out [n]	In[n]	Insertion
1	U	U	
2	S	U	SXM_SET
3	U	U	
4	S	∅	SXM_SET
5	S	S	
6	S	S	
7	S	S	
8	S	S	
9	U	S	SXM_UNSET

<115> 따라서, 도 6의 제2 노드 및 제4 노드에 'SXM_SET'이라는 모드 명령어가 삽입되고, 제9 노드에 'SXM_UNSET'이라는 모드 명령어가 삽입된다.

<116> <실험 결과>

<117> 본 발명에 따른 상기 SXM 최적화의 영향을 측정하기 위해서 GCC기반의 'Zaram DSP16 컴파일러'에 최적화 알고리즘을 적용하였다. 또한 'EEMBC(Embedded Microprocessor Benchmark Consortium) 벤치마크'를 이용하여 최적화 알고리즘을 평가하였다. 이때, 'Automotive 패키지'에 있는 5개의 벤치마크를 테스트하였는데, 각각의 상세 정보는 하기 <표 13>과 같다.

표 13

Code name	Category	Description
repeed01_lite	Automotive	Calculate the load speed
aifirt01_lite	Automotive	Finite impulse response filter
a2time01_lite	Automotive	Angle to Time Conversion
bitmnp01_lite	Automotive	Bit manipulation
iirflt01_lite	Automotive	Infinite impulse response filter

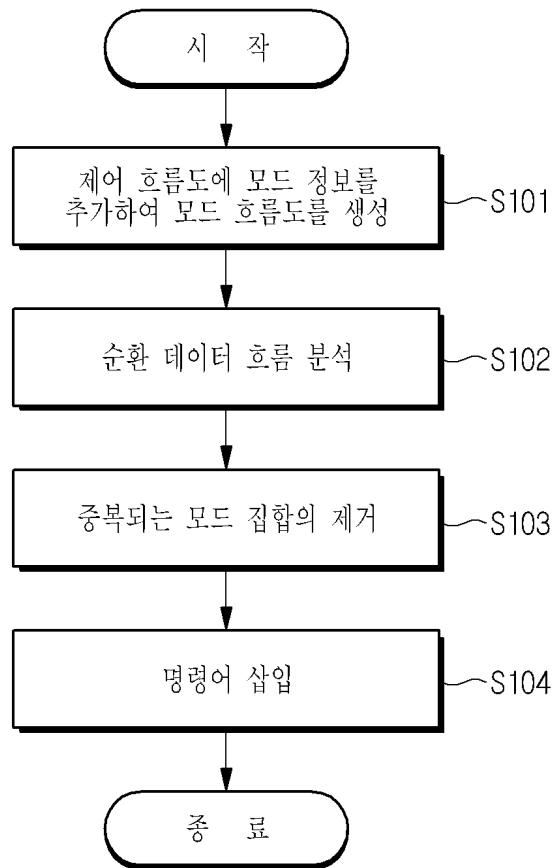
- <119> 한편, 본 발명에 따른 SXM 최적화의 구현은 수백 라인의 코드만을 추가함으로써 구현이 가능하며, 다른 최적화 알고리즘과 비교하였을 때, 매우 적은 노력으로도 구현이 가능하게 된다. 또한, SXM 최적화의 효과는 도 7 및 도 8에 나타나 있다.
- <120> 상기 도 7은 본 발명의 실시 예에 따른 실행 시간에 SXM 명령어의 감소 비율을 나타낸 그래프이고, 상기 도 8은 본 발명의 실시 예에 따른 실행 시간에 SXM 명령어 코드의 감소 비율을 나타낸 그래프이다.
- <121> 상기 도 7을 참조하면, 본 발명에 따라 명령어의 수가 평균 50%에서 최대 93%까지 감소한 것을 볼 수 있다. 또한, 상기 도 8을 참조하면, 본 발명에 따라 실행 코드 파일 크기가 약 2% 정도 줄어들었다.
- <122> 한편, 상기 실험에서, 'aifirt01_lite' 벤치마크는 다른 벤치마크와 다른 결과를 보여주고 있다. 즉, SXM 명령어의 감소 비율이 매우 높게 나왔는데, 이것은 'aifirt01_lite' 벤치마크가 다른 벤치마크에 비해 루프 내에서 CALL 패턴의 수가 적기 때문이다. 분석 과정 중에서 CALL 패턴을 만나면 모든 SXM 정보가 소실된다. 따라서 CALL 패턴 뒤에는 SXM 정보를 초기화시켜 주기 위해서 SXM 명령어를 삽입하게 된다. 따라서 코드를 생성할 때, CALL 패턴을 줄일 수 있다면 보다 나은 성능 향상을 얻을 수 있을 것이다.
- <123> 또 다른 포인트는 상기 도 7에 나타나듯이 반 이상의 SXM 명령어 실행시간에 성능을 낭비하고 있다는 점이다. 즉, 상기 도 8에 도시된 바와 같이 실제 SXM 명령어 코드의 감소는 적다고 하더라도, 상기 도 7에 도시된 바와 같이 SXM 최적화의 효과는 크게 나타나게 된다. 이것은 다수의 중복된 SXM 명령어가 루프 내에 존재하기 때문이다.
- <124> DSP는 상술한 바와 같이 스트리밍 데이터를 빠르게 처리할 수 있기 때문에 임베디드 시스템에 널리 사용되고 있다. 하지만, 복잡한 명령어 집합은 컴파일러가 보다 나은 코드를 생성하는데 장애물이 되고 있다. 또한, 모드 집합 명령어는 DSP에서 명령어 집합을 줄이기 위해 쓰이는 명령어이다. 모드 집합에 따라 같은 명령어도 다르게 동작할 수 있다.
- <125> 따라서, 본 발명에서 제안하는 중복된 모드 집합 명령어를 줄일 수 있는 코드 최적화 기법에 따라, 기존의 모드 명령어 최소화 기법과는 다르게 모드 흐름도(MFG)를 생성하고, 집합 명령어에 의해 모드 상태를 분석함으로써, 실행 시간에 모드 집합 명령어의 수가 최소가 됨을 보장하게 된다.
- <126> 또한, 상기 실시 예에서와 같이 본 발명에 따른 알고리즘을 'Zaram DSP16 아키텍처'의 부호 확장 모드(SXM)에 적용할 경우, 이를 통해 실행 시간에 평균 50%, 최고 93%의 모드 집합 명령어를 줄이고, 2%의 실행 코드 크기를 줄일 수 있음을 알 수 있다.
- <127> 한편, 본 발명의 실시 예에서는 구체적인 실시 예에 관해 설명하였으나, 본 발명의 범위에서 벗어나지 않는 한도 내에서 여러 가지 변형이 가능함은 물론이다. 그러므로 본 발명의 범위는 설명된 실시 예에 국한되어 정해져서는 안되며 후술하는 특허 청구의 범위뿐만 아니라 이 특허 청구의 범위와 균등한 것들에 의해 정해져야 한다.

도면의 간단한 설명

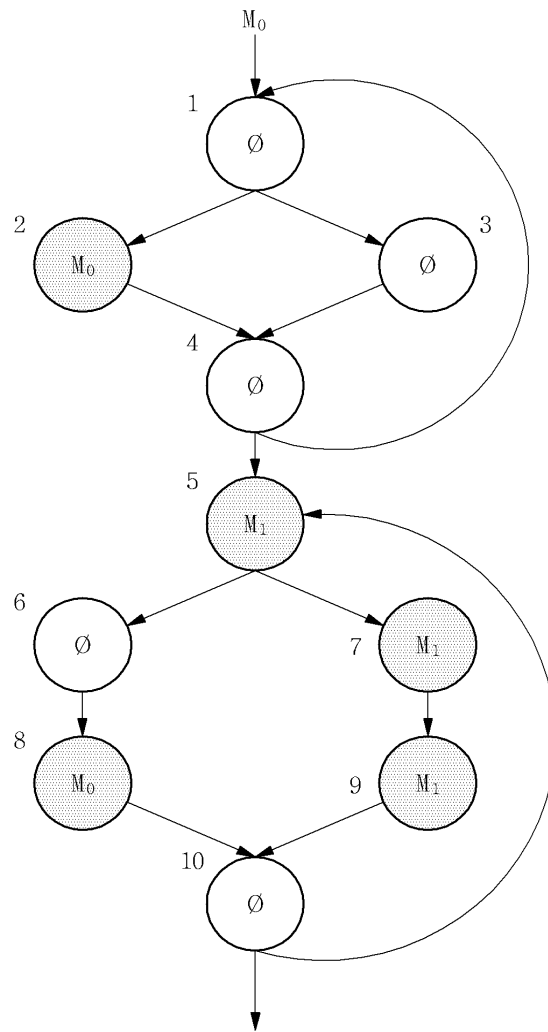
- <128> 도 1은 본 발명에 따른 디지털 신호 처리기의 명령어 집합에서 모드 명령어 최적화 절차를 나타낸 흐름도.
- <129> 도 2는 본 발명의 실시 예에 따라 생성되는 나타낸 모드 흐름도.
- <130> 도 3은 본 발명의 실시 예에 따른 순환 데이터 분석을 이용한 모드 명령어 삽입 지점을 나타낸 모드 흐름도.
- <131> 도 4는 본 발명의 실시 예에 따른 역방향 진행으로 인해 발생하는 중복 모드 집합을 나타낸 모드 흐름도.
- <132> 도 5는 본 발명의 실시 예에 따른 중복 모드를 제거한 모드 흐름도.
- <133> 도 6은 본 발명을 SXM에 적용한 모드 흐름도.
- <134> 도 7은 본 발명의 실시 예에 따른 실행 시간에 SXM 명령어의 감소 비율을 나타낸 그래프.
- <135> 도 8은 본 발명의 실시 예에 따른 실행 시간에 SXM 명령어 코드의 감소 비율을 나타낸 그래프.

도면

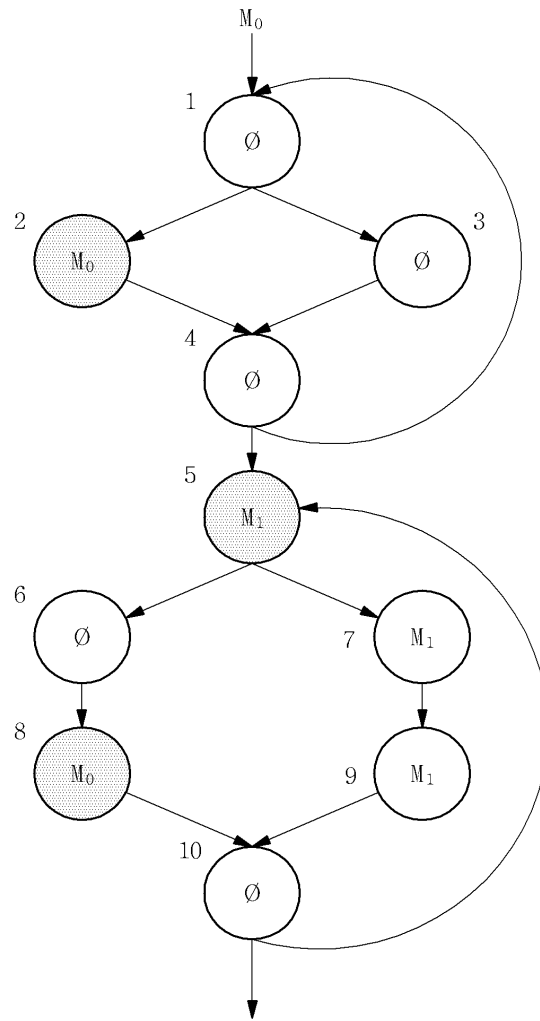
도면1



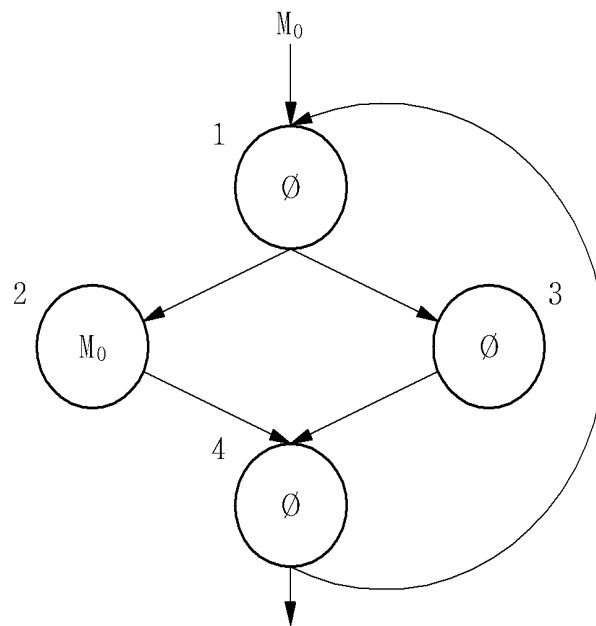
도면2



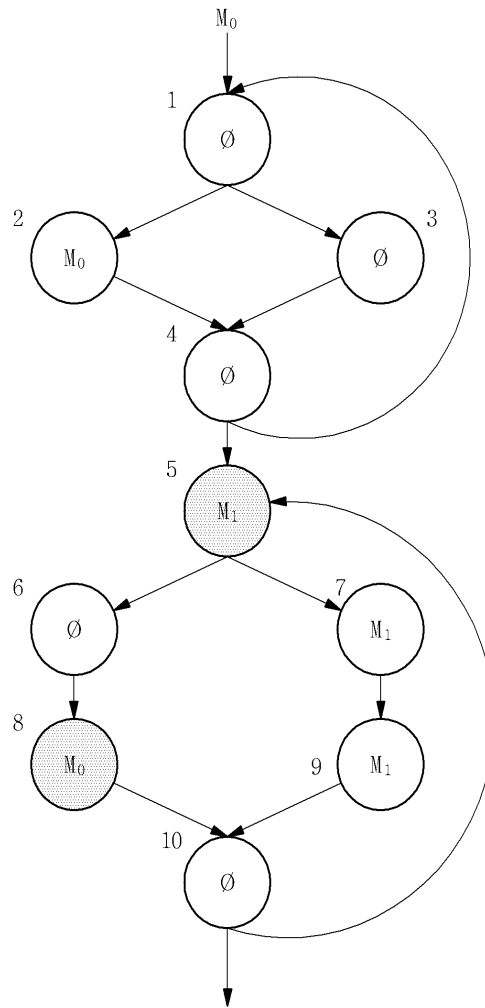
도면3



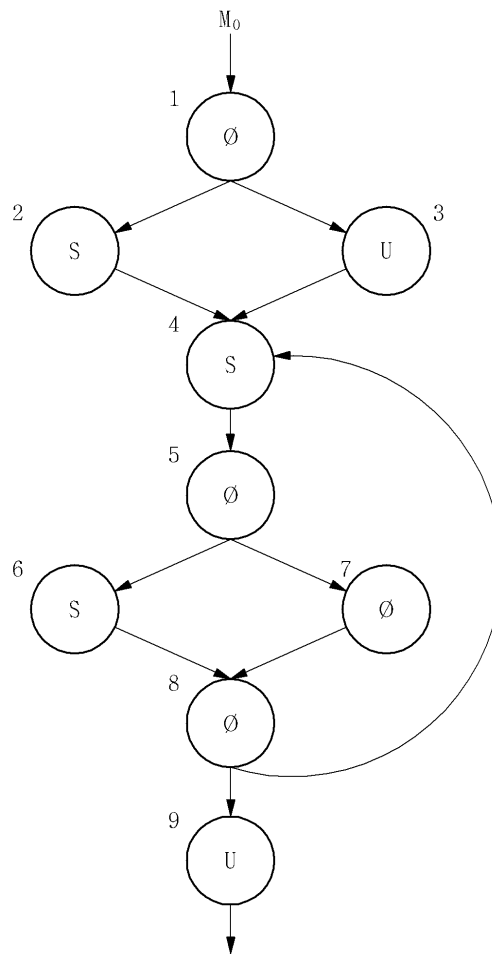
도면4



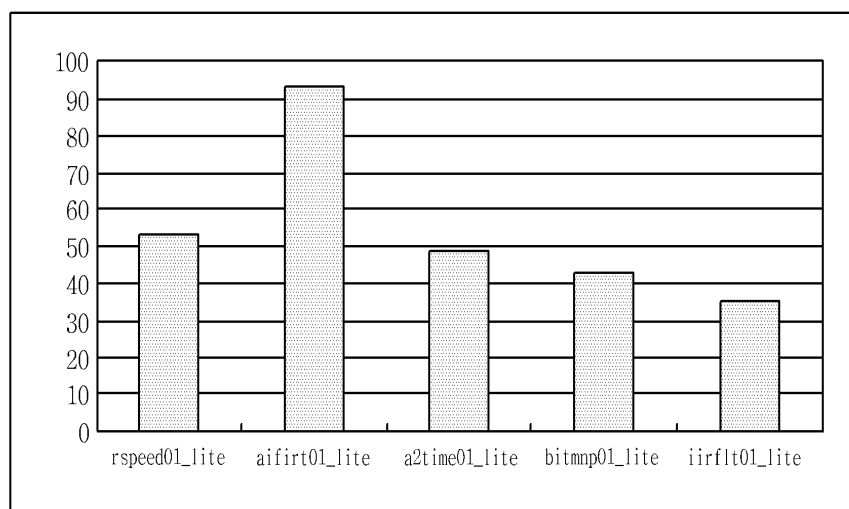
도면5



도면6



도면7



도면8

