(12) 특허협력조약에 의하여 공개된 국제출원

(19) 세계지식재산권기구
국제사무국

(43) 국제공개일
2023 년 11 월 9 일 (09.11.2023)

WIPO | PCT

(10) 국제공개번호
WO 2023/214668 A1

(51) 국제특허분류:
G06F 9/30 (2006.01) *G06F 9/38* (2006.01)

(21) 국제출원번호: PCT/KR2023/003208

(22) 국제출원일: 2023 년 3 월 9 일 (09.03.2023)

(25) 출원언어: 한국어

(26) 공개언어: 한국어

(30) 우선권정보:
10-2022-0054408 2022 년 5 월 2 일 (02.05.2022) KR

(71) 출원인: 이화여자대학교 산학협력단 (**EWHA UNIVERSITY - INDUSTRY COLLABORATION FOUNDATION**) [KR/KR]; 03760 서울특별시 서대문구 이화여대길 52, Seoul (KR).

(72) 발명자: 윤명국 (**YOON, Myung Kuk**); 06736 서울특별시 서초구 강남대로 224, 1204호, Seoul (KR).

(74) 대리인: 특허법인 무한 (**MUHANN PATENT & LAW FIRM**); 06144 서울특별시 강남구 언주로 560, 8층, Seoul (KR).

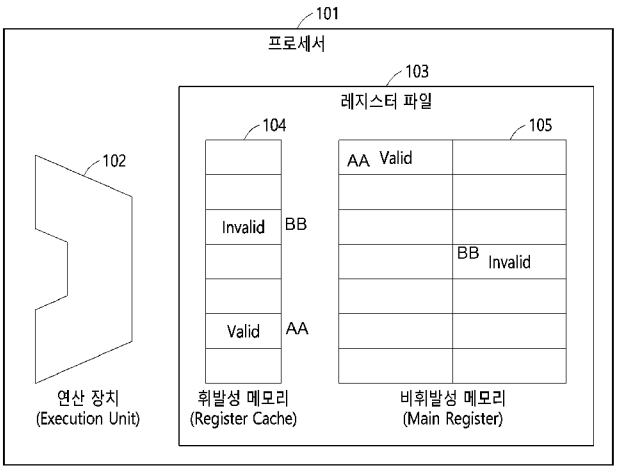
(81) 지정국 (별도의 표시가 없는 한, 가능한 모든 종류의 국내 권리의 보호를 위하여): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) 지정국 (별도의 표시가 없는 한, 가능한 모든 종류의 역내 권리의 보호를 위하여): ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), 유라시아 (AM, AZ, BY, KG, KZ, RU, TJ, TM), 유럽 (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

공개:
— 국제조사보고서와 함께 (조약 제21조(3))

(54) **Title:** INSTRUCTION EXECUTION METHOD OF PROCESSOR USING COMPILER DATA DEPENDENCY INFORMATION

(54) **발명의 명칭:** 컴파일러 데이터 의존성 정보를 이용한 프로세서의 명령어 연산 수행 방법



101 ... Processor
102 ... Calculation device (Execution Unit)
103 ... Register file
104 ... Volatile memory (Register Cache)
105 ... Non-volatile memory (Main Register)
AA ... Valid
BB ... Invalid

(57) **Abstract:** Disclosed is a method for a processor to execute instructions using compiler data dependency information. The method for a processor of a computing unit to execute instructions according to an embodiment of the present invention may comprise the steps in which the processor: reads data from a register file, which uses volatile memory and non-volatile memory together, in order to execute instructions using an execution unit; executes the instructions on the basis of the data by using the execution unit; and updates, in the volatile memory, result values derived through the execution of the instructions.

(57) **요약서:** 컴파일러 데이터 의존성 정보를 이용한 프로세서의 명령어 연산 수행 방법이 개시된다. 본 발명의 일실시예에 따른 컴퓨팅 장치의 프로세서가 수행하는 명령어 연산 수행 방법은, 프로세서가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리를 함께 사용하는 레지스터 파일로부터 읽는 단계; 상기 데이터에 기초하여 상기 연산장치를 이용해 상기 명령어 연산을 수행하는 단계; 및 상기 명령어 연산을 통해 도출된 결과 값을 상기 휘발성 메모리에 업데이트하는 단계를 포함할 수 있다.

WO 2023/214668 A1

명세서

발명의 명칭: 컴파일러 데이터 의존성 정보를 이용한 프로세서의 명령어 연산 수행 방법

기술분야

[1] 본 발명은 컴파일러 데이터 의존성 정보를 이용한 프로세서의 명령어 연산 수행 방법에 관한 것이다.

배경기술

[2] 비휘발성(Non-Volatile) 메모리는 높은 집적도(density) 및 낮은 파워 소모량(power consumption) 등의 장점을 가지고 있다. 따라서 비휘발성 메모리의 장점을 최대한 활용하기 위해, 최근 연구는 기존의 휘발성(Volatile) 메모리를 비휘발성 메모리로 대체하려 하고 있다. 특히, 비휘발성 메모리를 프로세서 내부의 저장장치인 레지스터 파일로 사용하기 위한 방법이 연구되고 있다. 그러나, 비휘발성 메모리의 특성인 쓰기 시간(write latency)이 오래 걸리고 쓰기 횟수가 제한되어 있는(write endurance) 등의 문제가 존재한다. 그러므로, 비휘발성 메모리를 레지스터 파일로 곧 바로 사용하기에는 어려움이 있다. 이러한 문제를 해결하기 위해, 기존의 휘발성 메모리와 비휘발성 메모리를 함께 사용하는 레지스터 파일 기술이 제안되었다.

발명의 상세한 설명

기술적 과제

[3] 본 발명은 컴파일러 데이터 의존성 정보를 이용하여 휘발성 메모리에 저장된 불필요한 데이터를 제거하여 휘발성 메모리에서 비휘발성 메모리로 데이터를 이동하는 연산 중 불필요한 쓰기 연산을 줄이기 위한 방법을 제공한다.

[4] 본 발명은 컴파일러가 데이터 의존성 정보를 생성하는 방법을 제공한다.

과제 해결 수단

[5] 본 발명의 일실시예에 따른 컴퓨팅 장치의 프로세서가 수행하는 명령어 연산 수행 방법에 있어서, 프로세서의 명령어 연산 수행방법은, 프로세서가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리를 함께 사용하는 레지스터 파일로부터 읽는 단계; 상기 데이터에 기초하여 상기 연산장치를 이용해 상기 명령어 연산을 수행하는 단계; 및 상기 명령어 연산을 통해 도출된 결과 값을 상기 휘발성 메모리에 업데이트할 수 있다.

[6] 상기 프로세서가 명령어 의존성 정보를 이용하여 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계; 및 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 휘발성 메모리에 저장된 데이터를 제거하는 단계를 더 포함할 수 있다.

[7] 상기 명령어 의존성 정보는, 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보일 수 있다.

[8] 상기 휘발성 메모리는, 명령어 의존성 정보에 의해 상기 명령어 연산을 수행하기 위한 상기 휘발성 메모리에 저장된 데이터가 제거된 이후 상기 명령어 연산의 결과 값이 업데이트될 수 있다.

[9] 상기 휘발성 메모리는, 정적 랜덤 액세스 메모리(Static Random Access Memory, SRAM)일 수 있다.

[10] 상기 비휘발성 메모리는, 스핀전달토크형 자기저항메모리(Spin-Transfer-Torque Magnetoresistive Random Access Memory, STT-MRAM) 또는 스핀궤도토크형 자기저항메모리(Spin-Orbit Torque Magnetoresistive Random Access Memory, SOT-MRAM)일 수 있다.

[11] 상기 레지스터 파일은, 쓰기 연산 필터를 포함하고, 상기 쓰기 연산 필터는 마지막 참조 비트 벡터 및 소스 피연산자 인덱스에 기초하여 마지막 참조 비트를 결정할 수 있다.

[12] 상기 휘발성 메모리는, 읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 거짓 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화하지 않을 수 있다.

[13] 상기 휘발성 메모리는, 읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 참 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화할 수 있다.

[14] 본 발명의 일실시예에 따른 명령어 의존성 정보를 생성하는 방법은, 컴파일러가 프로그램의 컴파일 과정에서 상기 프로그램을 실행하는 명령어 연산에 이용되는 명령어를 순서대로 확인하는 단계; 레지스터에 저장된 데이터가 상기 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계; 및 상기 레지스터에 저장된 데이터가 상기 명령어에 의해 이용된 이후에 추가로 이용되지 않는 경우, 상기 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보인 명령어 의존성 정보를 결정하는 단계를 포함할 수 있다.

[15] 상기 레지스터에 저장된 데이터가 상기 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계는, 상기 레지스터에 저장된 데이터가 상기 명령어의 후속 명령어에 의해 업데이트되는지 여부에 따라 상기 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단할 수 있다.

[16] 본 발명의 일실시예에 따른 명령어 연산 수행 방법을 수행하는 컴퓨팅 장치는 프로세서를 포함하고, 상기 프로세서는, 프로세서가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리를 함께 사용하는 레지스터 파일로부터 읽고, 상기 데이터에 기초하여 상기 연산장치를 이용해 상기 명령어 연산을 수행하고, 상기 명령어 연산을 통해 도출된 결과 값을 상기 휘발성 메모리에 업데이트할 수 있다.

[17] 상기 프로세서는, 상기 프로세서가 명령어 의존성 정보를 이용하여 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되는지 여부를 판단하고, 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 휘발성 메모리에 저장된 데이터를 제거할 수 있다.

[18] 상기 명령어 의존성 정보는, 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 데이터가 저장된 레지스터의 번호 정보를 포함할 수 있다.

[19] 상기 휘발성 메모리는, 명령어 의존성 정보에 의해 상기 명령어 연산을 수행하기 위한 상기 휘발성 메모리에 저장된 데이터가 제거된 이후 상기 명령어 연산의 결과 값이 업데이트될 수 있다.

[20] 상기 휘발성 메모리는, 정적 랜덤 액세스 메모리(Static Random Access Memory, SRAM)일 수 있다.

[21] 상기 비휘발성 메모리는, 스핀전달토크형 자기저항메모리(Spin-Transfer-Torque Magnetoresistive Random Access Memory, STT-MRAM) 또는 스핀궤도토크형 자기저항메모리(Spin-Orbit Torque Magnetoresistive Random Access Memory, SOT-MRAM)일 수 있다.

[22] 상기 레지스터 파일은, 쓰기 연산 필터를 포함하고, 상기 쓰기 연산 필터는 마지막 참조 비트 벡터 및 소스 피연산자 인덱스에 기초하여 마지막 참조 비트를 결정할 수 있다.

[23] 상기 휘발성 메모리는, 읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 거짓 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화하지 않을 수 있다.

[24] 상기 휘발성 메모리는, 읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 참 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화할 수 있다.

발명의 효과

[25] 본 발명의 일실시예에 따르면, 컴파일러 데이터 의존성 정보를 이용하여 휘발성 메모리에 저장된 불필요한 데이터를 제거하여 휘발성 메모리에서 비휘발성 메모리로 데이터를 이동하는 연산 중 불필요한 쓰기 연산을 줄일 수 있다.

[26] 본 발명의 일실시예에 따르면, 컴파일러가 데이터 의존성 정보를 생성할 수 있다.

도면의 간단한 설명

[27] 도 1은 본 발명의 일실시예에 따른 프로세서의 구성을 설명하기 위한 도면이다.

[28] 도 2는 본 발명의 일실시예에 따른 휘발성 메모리에 저장되어 있던 데이터를 비휘발성 메모리로 이동하는 연산을 설명하기 위한 도면이다.

[29] 도 3은 본 발명의 일실시예에 따른 휘발성 메모리에 저장되어 있던 데이터를 비휘발성 메모리로 이동하지 않고 명령어 연산의 결과 값을 휘발성 메모리에 저장하는 방법을 설명하기 위한 도면이다.

[30] 도 4는 본 발명의 일실시예에 따른 프로세서의 명령어 연산 수행 방법을 도시한 플로우 차트이다.

[31] 도 5는 본 발명의 일실시예에 따른 명령어 의존성 정보를 생성하는 방법을 도시한 플로우 차트이다.

[32] 도 6은 본 발명의 일실시예에 따른 레지스터 파일에서의 읽기 및 쓰기 연산을 설명하기 위한 도면이다.

[33] 도 7은 본 발명의 일실시예에 따른 STT-MRAM 레지스터 파일과 SRAM 레지스터 캐시를 포함하는 GPU를 도시하는 도면이다.

[34] 도 8은 본 발명의 일실시예에 따른 Write-Back 연산 필터와 레지스터 캐시의 동작을 설명하기 위한 도면이다.

[35] 도 9는 본 발명의 일실시예에 따른 Write-Back 연산 필터가 마지막 참조 비트를 결정하는 방법을 도시한 플로우차트이다.

[36] 도 10은 레지스터 캐시가 수행하는 캐시 라인의 무효화 방법을 설명하기 위한 플로우차트이다.

발명의 실시를 위한 최선의 형태

[37] 이하, 실시예들을 첨부된 도면을 참조하여 상세하게 설명한다. 그러나, 특허출원의 범위가 이러한 실시예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한 부재를 나타낸다.

[38] 아래 설명하는 실시예들에는 다양한 변경이 가해질 수 있다. 아래 설명하는 실시예들은 실시 형태에 대해 한정하려는 것이 아니며, 이들에 대한 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

[39] 실시예에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로, 실시예를 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수 개의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 명세서 상에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[40] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 실시예가 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가지고 있다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥 상 가지는 의미와 일치하는 의미를 가지는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[41] 또한, 첨부 도면을 참조하여 설명함에 있어, 도면 부호에 관계없이 동일한 구성요소는 동일한 참조 부호를 부여하고 이에 대한 중복되는 설명은 생략하기로 한다. 실시예를 설명함에 있어서 관련된 공지 기술에 대한 구체적인 설명이 실시예의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우 그 상세한 설명을 생략한다.

[42] 이하, 본 발명의 실시예를 첨부된 도면을 참조하여 상세하게 설명한다.

[43]

[44] 도 1은 본 발명의 일실시예에 따른 프로세서의 구성을 설명하기 위한 도면이다.

[45] 도 1을 참고하면, 프로세서(101), 연산장치(102) 및 레지스터 파일(103)이 도시된다.

[46] 프로세서(101)는 레지스터 파일 구조를 사용하는 모든 프로세서인 CPU(Central Processing Unit), GPU(Graphic Processing Unit), NPU(Neural Processing Unit), 가속기 등일 수 있다. 프로세서(101)는 연산장치(execution unit)(102) 및 레지스터 파일(103)을 포함할 수 있다.

[47] 연산장치(102)는 컴퓨터 프로그램의 지시에 따라 연산과 계산을 수행하는 프로세서의 일부이다. 즉, 연산장치(102)는 명령어 연산을 수행할 수 있다. 연산장치(102)는 레지스터 파일(103)로부터 데이터를 받아서 명령어 연산을 수행할 수 있다.

[48] 레지스터 파일(103)은 내부 RAM으로 이루어진 레지스터의 집단 또는 레지스터의 어레이를 의미한다. 여기서 내부 RAM은 휘발성 메모리인 정적 랜덤 액세스 메모리(Static Random Access Memory, SRAM)일 수 있다. 레지스터 파일(103)은 내부 데이터 메모리 및 레지스터의 역할을 모두 수행할 수 있다.

[49] 레지스터 파일(103)은 프로세서에서 에너지를 가장 많이 소모하는 구성중 하나이다. 따라서, 레지스터 파일(103)의 에너지 소비를 감소시키기 위해 스핀전달토크형 자기저항메모리(Spin-Transfer-Torque Magnetoresistive Random Access Memory, STT-MRAM) 또는 스핀궤도토크형 자기저항메모리(Spin-Orbit Torque Magnetoresistive Random Access Memory, SOT-MRAM)와 같은 새로운 비휘발성 메모리를 레지스터 파일로 사용하는 GPU 아키텍쳐가 연구되었다.

[50] 그러나, 비휘발성 메모리(105)에 대한 액세스는 시간이 더 오래 걸리는 문제가 있었다. 이 문제를 해결하기 위해, 휘발성 메모리(104)와 비휘발성 메모리(105)가 있는 계층적 레지스터 파일(hierarchical register file)가 제안되었다. 계층적 레지스터 파일은 하이브리드 레지스터 파일(hybrid register file)라고도 한다.

[51] 계층적 레지스터 파일은 STT-MRAM과 같은 비휘발성 메모리(105)로 구성된 레지스터 파일과 기능 유닛(functional unit)의 사이에 작은 SRAM에 기초한 레지스터 캐시(register cache)가 배치된다. 레지스터 캐시는 휘발성 메모리(104)로 구성될 수 있다. 레지스터 캐시는 레지스터 파일에 대한 액세스를 감소시켜 프로세서의 성능 손실을 최소화하고 에너지 효율성을 향상시킬 수 있다.

[52]

[53] 도 1의 레지스터 파일(103)은 계층적 레지스터 파일일 수 있다. 레지스터 파일(103)은 휘발성 메모리(104) 및 비휘발성 메모리(105)에 기초할 수 있다. 즉, 레지스터 파일(103)은 휘발성 메모리(104) 및 비휘발성 메모리(105)를 함께 사용할 수 있다.

[54] 휘발성 메모리(104)는 레지스터 캐시일 수 있다. 비휘발성 메모리(105)는 메인 레지스터(main register)일 수 있다. 휘발성 메모리(104)는 SRAM 일 수 있다. 비휘발성 메모리(105)는 STT-MRAM 또는 SOT-MRAM 일 수 있다. 휘발성 메모리(104) 및 비휘발성 메모리(105)는 복수의 레지스터로 이루어질 수 있다. 휘발성 메모리(104)는 비휘발성 메모리보다 적은 수의 레지스터로 이루어질 수 있다. 레지스터 파일(103)에 저장된 데이터는 연산장치(102)에 명령어 연산을 수행하기 위해 제공될 수 있다. 명령어 연산을 통해 도출된 결과 값은 레지스터 파일(103)에 업데이트될 수 있다.

[55]

[56] 도 2는 본 발명의 일실시예에 따른 휘발성 메모리에 저장되어 있던 데이터를 비휘발성 메모리로 이동하는 연산을 설명하기 위한 도면이다.

[57] 즉, 도 2는 계층적 레지스터 파일(hierarchical register file)에서 휘발성 메모리(104)에 저장되어 있던 데이터가 비휘발성 메모리(105)로 이동되는 연산인 Write-Back 연산을 설명하기 위한 도면이다.

[58] 연산장치(102)는 휘발성 메모리(104) 또는 비휘발성 메모리(105)로부터 명령어 연산을 수행하기 위해 데이터를 제공받을 수 있다. 연산장치(102)가 명령어 연산을 수행하면 결과 값이 생성될 수 있다. 계층적 레지스터 파일은 결과 값을 휘발성 메모리(104)에 먼저 저장할 수 있다.

[59] 이때, 휘발성 메모리(104)의 저장 공간이 부족한 경우, 프로세서는 휘발성 메모리(104)에 저장되어 있던 기존 데이터를 비휘발성 메모리(105)로 이동하고 휘발성 메모리(104)의 저장 공간을 확보하는 Write-Back 연산을 수행할 수 있다.

[60] 그런데, 휘발성 메모리(104)의 저장 공간은 제한적이므로, Write-Back 연산이 빈번하게 일어나게 되어 성능의 저하가 발생할 수 있다. 또한, 불필요한 데이터가 비휘발성 메모리(105)에 저장되어 비휘발성 메모리(105)의 쓰기 횟수 제한에 더 빨리 도달할 수 있다.

[61] 따라서, Write-Back 연산을 제거하여, 불필요한 데이터의 비휘발성 메모리(105)에 대한 쓰기 연산을 줄일 필요가 있다. 이하에서 Write-Back 연산을 제거한 프로세서의 명령어 연산 수행 방법 대하여 설명하겠다.

[62]

[63] 도 3은 본 발명의 일실시예에 따른 휘발성 메모리에 저장되어 있던 데이터를 비휘발성 메모리로 이동하지 않고 명령어 연산의 결과 값을 휘발성 메모리에 저장하는 방법을 설명하기 위한 도면이다.

[64] 즉, 도 3은 계층적 레지스터 파일(hierarchical register file)에서 휘발성 메모리(104)에 저장되어 있던 불필요한 데이터가 제거되어 Write-Back 연산이 제거됨을 설명하기 위한 도면이다.

[65] 연산장치(102)는 휘발성 메모리(104) 또는 비휘발성 메모리(105)로부터 명령어 연산을 수행하기 위해 데이터를 제공받을 수 있다. 연산장치(102)가 명령어 연산을 수행하면 결과 값이 생성될 수 있다. 계층적 레지스터 파일은 결과 값을 휘발성 메모리(104)에 먼저 저장할 수 있다.

[66] 이때, 휘발성 메모리(104)에서 명령어 연산을 수행하기 위해 연산장치(102)에 제공된 데이터가 이후에 추가로 이용되지 않는 경우, 프로세서는 명령어 연산을 수행하기 위해 연산장치(102)에 제공된 데이터를 휘발성 메모리(104)로부터 제거할 수 있다. 즉, 더 이상 사용되지 않는 데이터인 불필요한 데이터가 휘발성 메모리(104)에서 제거됨으로써 연산에 따른 결과 값을 저장할 수 있는 추가 공간을 확보할 수 있다.

[67] 명령어 연산을 수행하기 위해 연산장치(102)에 제공된 데이터가 이후에 추가로 이용되는지 여부는 명령어 의존성 정보를 통해 판단할 수 있다.

[68] 명령어 의존성 정보는, 명령어 연산을 수행하기 위한 데이터가 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보일 수 있다.

[69] 결국, 연산장치(102)가 명령어 연산을 수행한 결과 값은 휘발성 메모리(104)로 업데이트 되고, Write-Back 연산은 이루어지지 않는다.

[70] 따라서, 비휘발성 메모리(105)의 반복된 쓰기 연산으로 인한 성능 저하 문제가 해결될 수 있다. 비휘발성 메모리(105)의 반복된 쓰기 연산의 감소로 비휘발성 메모리(105)의 수명을 연장할 수 있다. 또한, Write-Back 연산을 수행하지 않음으로써 레지스터 파일(103)의 파워 소모량이 감소할 수 있다.

[71]

[72] 도 4는 본 발명의 일실시예에 따른 프로세서의 명령어 연산 수행 방법을 도시한 플로우 차트이다.

[73] 단계(401)에서, 프로세서(101)가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리를 함께 사용하는 레지스터 파일로부터 읽을 수 있다.

[74] 데이터가 휘발성 메모리 및 비휘발성 메모리에 나누어 저장된 경우, 프로세서는 휘발성 메모리 및 비휘발성 메모리로부터 데이터를 읽을 수 있다.

[75] 단계(402)에서, 프로세서(101)는 레지스터 파일로부터 읽은 데이터에 기초하여 연산장치를 이용해 명령어 연산을 수행할 수 있다. 프로세서의 연산장치(102)는 명령어 연산을 수행할 수 있다.

[76] 단계(403)에서, 프로세서(101)는 명령어 연산을 통해 도출된 결과 값을 휘발성 메모리에 업데이트할 수 있다. 계층적 레지스터 파일은 명령어 연산을 통해 도출된 결과 값을 항상 휘발성 메모리에 먼저 저장할 수 있다.

[77] 프로세서의 명령어 연산 수행 방법은 Write-Back 연산을 제거하기 위해, 프로세서(101)가 명령어 의존성 정보를 이용하여 명령어 연산을 수행하기 위한 데이터가 명령어 연산에 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계를 더 포함할 수 있다.

[78] 명령어 의존성 정보는 명령어 연산을 수행하기 위한 데이터가 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보일 수 있다.

[79] 또한, 프로세서의 명령어 연산 수행 방법은 Write-Back 연산을 제거하기 위해, 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 휘발성 메모리에 저장된 데이터를 제거하는 단계를 더 포함할 수 있다.

[80]

[81] 도 5는 본 발명의 일실시예에 따른 명령어 의존성 정보를 생성하는 방법을 도시한 플로우 차트이다.

[82] 단계(501)에서, 컴파일러가 프로그램의 컴파일 과정에서 프로그램을 실행하는 명령어 연산에 이용되는 명령어를 순서대로 확인할 수 있다.

[83] 즉, 컴파일러는 프로그램의 컴파일 과정에서 각 레지스터에 저장된 데이터를 마지막으로 읽는 명령어를 확인할 수 있다.

[84] 단계(502)에서, 컴파일러는 레지스터에 저장된 데이터가 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단할 수 있다.

[85] 컴파일러는 레지스터에 저장된 데이터가 현재 명령어의 후속 명령어에 의해 업데이트되는지 여부에 따라 명령어에 의해 이용된 이후에 추가로 이용되는지를 결정할 수 있다.

[86] 단계(503)에서, 컴파일러는 레지스터에 저장된 데이터가 명령어에 의해 이용된 이후에 추가로 이용되지 않는 경우, 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보인 명령어 의존성 정보를 결정할 수 있다.

[87] 구체적으로, 컴파일러는 레지스터 파일로부터 데이터를 읽는 요청이 특정 레지스터에 대하여 마지막 요청인지를 식별하기 위해 마지막 참조 비트 벡터인 LRB(Last Reference Bit) 벡터를 생성하도록 수정될 수 있다. 컴파일러는 명령어를 순서대로 확인하여 명령어가 마지막으로 특정 레지스터를 소스 피연산자(source operand)로 사용하는지 여부를 확인할 수 있다.

[88] 여기서 마지막이란, 현재 명령어에 의하여 레지스터가 이용된 이후에 더 이상 이용되지 않는 경우이다. 또는 현재 명령어가 아닌 다른 명령어에 의하여 레지스터 값이 업데이트 되지 않은 경우이다.

[89] 소스 레지스터는 명령어에 입력으로 데이터를 제공하는 레지스터일 수 있다. 소스 레지스터의 데이터는 소스 피연산자 일 수 있다.

[90] 명령어가 마지막으로 해당 레지스터에 포함된 데이터를 소스 피연산자로 사용하는 경우, 컴파일러는 마지막 참조 비트인 LRB(Last Reference Bit) 벡터를 결정

할 수 있다. LRB 벡터는 소스 피연산자가 후속 명령어에 의해 더 이상 사용되지 않는 레지스터인 데드 레지스터(dead register)의 위치를 나타내는 벡터일 수 있다. 여기서, LRB 벡터는 명령어 의존성 정보일 수 있다.

[91] LRB 벡터는 4비트 벡터이다. LRB 벡터는 [S0 S1 S2 S3]의 형태일 수 있다. 여기서, S0는 해당 명령어의 첫번째 소스 레지스터를 의미할 수 있다. S1은 해당 명령어의 두번째 소스 레지스터를 의미할 수 있다. S2는 해당 명령어의 세번째 소스 레지스터를 의미할 수 있다. S3는 해당 명령어의 네번째 소스 레지스터를 의미할 수 있다. 소스 레지스터가 해당 명령어에서 이후에 추가로 사용되지 않는 경우 1로 표시할 수 있다. 소스 레지스터가 해당 명령어 이후에 추가로 사용되는 경우 0으로 표시할 수 있다.

[92] 예를 들어, [1 0 0 0] 인 LRB 벡터는 해당 명령어의 첫번째 소스 레지스터가 다시 사용되지 않음을 의미할 수 있다. [0 1 0 0] 인 LRB 벡터는 해당 명령어의 두번째 소스 레지스터가 다시 사용되지 않음을 의미할 수 있다.

[93] 컴파일러는 해당 소스 레지스터가 마지막으로 나타나는 경우 명령어에 의해 이용된 이후에 추가로 이용되지 않는 것으로 결정할 수 있다. 컴파일러는 소스 레지스터에 저장된 데이터가 현재 명령어의 후속 명령어에 의해 업데이트되는지 여부에 따라 명령어에 의해 이용된 이후에 추가로 이용되는지를 결정할 수 있다.

[94] 예를 들어, 도 6을 참조하면, 601에서 R5는 명령어 i24의 소스 피연산자로 나타나고, 명령어 i24 이후의 모든 명령어에서 소스 피연산자로 다시는 나타나지 않는다. 따라서, 컴파일러는 R5가 명령어 i24에 의해 이용된 이후에 추가로 이용되지 않는 것으로 결정할 수 있다. 그러므로, 해당 예시에서 명령어 i24의 LRB 벡터의 첫번째 소스 피연산자의 비트인 R5가 1로 표현된다. 생성된 LRB 벡터는 해당 명령어로 인코딩 될 수 있다.

[95] 수정된 컴파일러에서 Loop/While과 같은 반복문에 의해 루프가 생성될 수 있다. 수정된 컴파일러에서 루프의 명령어는 LRB 벡터를 생성하기 위해 신중하게 처리되어야 한다. 컴파일러가 루프에서 단순히 명령어의 순서를 확인하면, 다음 루프 반복의 명령어에 의해 레지스터가 재사용될 수 있기 때문에 잘못된 LRB 벡터를 생성할 수 있다. 따라서, 컴파일러는 모든 루프의 명령어를 두 번 반복하여 LRB 벡터를 생성할 수 있다.

[96] 컴파일러가 모든 루프의 명령어를 두번 반복하면, 항상 명령어에 대한 한 쌍의 LRB 벡터가 있다. LRB가 두 벡터의 쌍에 모두 설정되어 있으면, 루프 전달 종속성(loop-carried dependency)이 없다. 루프 전달 종속성이 없으면, 레지스터는 제거될 수 있다. 여기서 루프 전달 종속성이란 처음 루프에서 생성된 결과값이 다음 루프의 입력으로 사용되는 경우이다.

[97] 컴파일러가 모든 루프의 명령어를 두번 반복한 결과로 LRB가 두 벡터의 쌍에 모두 설정되어 있으면, 해당 레지스터는 제거될 수 있다.

[98] 그러나, LRB가 하나의 벡터에만 설정되어 있으면, 해당 레지스터는 후속 명령어에 의해서 다시 사용될 수 있으므로 반복될 수 있으므로 제거되어서는 안 된다. 즉, 루프 전달 종속성이 있으므로 해당 레지스터는 제거되어서는 안 된다.

[99]

[100] 도 6은 본 발명의 일실시예에 따른 레지스터 파일에서의 읽기 및 쓰기 연산을 설명하기 위한 도면이다.

[101] 도 6은 기존의 계층적 레지스터 파일의 레지스터 캐시(602) 및 본 발명의 레지스터 파일의 레지스터 캐시(603)의 읽기 및 쓰기 동작의 예를 도시한 도면이다. 레지스터 캐시는 레지스터 파일에 포함되는 휘발성 메모리일 수 있다.

[102] 여기서, 명령어 i13 및 명령어 i26에서 계산된 결과 값인 R5 및 R7은 레지스터 캐시의 동일한 캐시 라인으로 저장된다고 가정한다. R5는 명령어 i24에서 읽은 후에 다시 사용되지 않는다고 가정한다.

[103] 기존의 계층적 레지스터 파일의 레지스터 캐시(602) 및 본 발명의 레지스터 파일의 레지스터 캐시(603)는 시간의 흐름에 따른 레지스터 캐시의 동작을 도시한다.

[104] 기존의 계층적 레지스터 파일의 레지스터 캐시(602)에서, 명령어 i13의 결과 값은 R5에 쓰여진 후에 명령어 i24가 읽는다. 몇 사이클이 지난 후, 명령어 i26은 그 결과 값인 R7을 레지스터 캐시에 업데이트하려고 시도하지만, 해당 레지스터 캐시 항목이 이미 R5를 보유하고 있어서 즉시 업데이트 할 수 없다. 따라서 R5를 유지하면서 R7을 업데이트하기 위하여, 레지스터 캐시는 R5를 레지스터 캐시에서 메인 레지스터 파일인 STT-MRAM으로 이동하는 Write-Back 연산을 수행해야 한다. Write-Back 연산이 완료된 후에, 명령어 i26은 결과값 R7을 레지스터 캐시에 업데이트할 수 있다.

[105] 반면에 본 발명의 레지스터 파일의 레지스터 캐시(603)의 연산에서, Write-Back 연산은 마지막 참조 정보(last reference information)에 의하여 제거될 수 있다. 마지막 참조 정보는 명령어 의존성 정보일 수 있다. 기존의 계층적 레지스터 파일의 레지스터 캐시(602)와 유사하게, 본 발명의 레지스터 파일의 레지스터 캐시(603)에서 명령어 i13은 먼저 결과값인 R5를 업데이트 할 수 있다. 그 다음, R5가 다음 명령에 의하여 다시 사용되지 않기 때문에, 본 발명의 레지스터 파일의 레지스터 캐시(603)는 명령어 i24가 R5를 읽은 후 R5에 해당하는 캐시 라인을 제거할 수 있다. R5가 다음 명령어에 의하여 다시 사용되지 않는다는 정보는 해당 명령어에서 데드 레지스터의 위치를 나타내는 LRB 벡터에 인코딩 될 수 있다. R5에 해당하는 캐시 라인을 제거함으로써, 레지스터 캐시에서 레지스터 파일로 데이터를 이동하는 Write-Back 연산은 더 이상 필요하지 않다. 601에서 명령어 i24의 LRB 벡터는 첫번째 소스 레지스터인 R5가 다시 사용되지 않음을 나타낸다. 마지막으로 명령어 i26은 Write-Back 연산을 기다리지 않고 결과값 R7을 즉시 레지스터 캐시에 업데이트 할 수 있다.

[106] 이하에서, 본 발명의 일실시예인 GPU 아키텍쳐를 설명하겠다.

[107]

[108] 도 7은 본 발명의 일실시예인 STT-MRAM 레지스터 파일과 SRAM 레지스터 캐시를 포함하는 GPU를 도시하는 도면이다.

[109] 도 7을 참조하면 GPU(700), 복수의 스트리밍 멀티프로세서(Streaming Multiprocessor, SM)(701) (이하, SM), 계층적 레지스터 파일(702), 레지스터 캐시(703) (이하, RC), STT-MRAM 레지스터(704), 쓰기 연산 필터(Write-Back Operation Filter, WOF)(705)(이하, WOF) 및 피연산자 수집 장치(Operand Collector)(706)이 도시 된다.

[110] GPU(700)는 복수의 SM(701)을 포함할 수 있다. SM(701)은 계층적 레지스터 파일(702)을 포함할 수 있다. 계층적 레지스터 파일(702)은 RC(703), STT-MRAM 레지스터(704), WOF(705) 및 피연산자 수집 장치(706)를 포함할 수 있다. 여기서, RC는 SRAM 레지스터 캐시로 휘발성 메모리일 수 있다. STT-MRAM 레지스터는 비휘발성 메모리일 수 있다.

[111] 이하에서 WOF가 LRB를 결정하는 방법 및 RC가 LRB에 기초하여 무효화를 수행하는 방법을 설명하겠다.

[112]

[113] 도 8은 본 발명의 일실시예에 따른 Write-Back 연산 필터(705)와 레지스터 캐시(703)의 동작을 설명하기 위한 도면이다.

[114] 마지막 참조 비트(이하 LRB)는 목표 피연산자(destination operand)에서 가장 가까운 소스 피연산자부터 나중의 소스 피연산자까지 1씩 증가하여 인덱싱 될 수 있다. 즉, LRB 벡터 [S0 S1 S2 S3]에서, S0는 목표 피연산자에서 가장 가까운 소스 피연산자 일 수 있다. S3은 목표 피연산자에서 가장 나중의 소스 피연산자 일 수 있다.

[115] 따라서, S0는 목표 레지스터로부터 가장 가까운 소스 레지스터이므로, 0으로 인덱싱 되고 해당 명령어의 첫번째 소스 레지스터를 의미할 수 있다. 마찬가지로, S1은 목표 레지스터로부터 두번째로 가까운 소스 레지스터이므로, 1로 인덱싱되고 해당 명령어의 두번째 소스 레지스터를 의미할 수 있다. S2는 목표 레지스터로부터 세번째로 가까운 소스 레지스터이므로, 2로 인덱싱되고 해당 명령어의 세번째 소스 레지스터를 의미할 수 있다. S3는 목표 레지스터로부터 가장 나중의 소스 레지스터이므로, 3으로 인덱싱되고 해당 명령어의 네번째 소스 레지스터를 의미할 수 있다. 목표 레지스터는 결과 값이 업데이트되는 레지스터를 의미할 수 있다. 목표 피연산자는 목표 레지스터에 업데이트되는 결과 값을 의미할 수 있다.

[116] 도 6의 601을 참조하면, 명령어는 [Opcode, D, S0, S1, S2, S3]으로 표현할 수 있다. D는 목표 피연산자(destination operand)를 의미할 수 있다. 결국, 명령어는 최대 4개의 소스 피연산자를 고려할 수 있다. 즉, 4 비트의 LRB 벡터를 고려할 수 있다.

[117] 예를 들어, [IADD R5, R4, R3]와 같은 IADD GPU 명령어가 있다고 가정하겠다. 해당 IADD GPU 명령어는 R4와 R3을 더한 결과값을 R5에 업데이트할 수 있다. 이때, R5는 목표 피연산자를 의미할 수 있다. R4는 S0로 첫번째 소스 레지스터를 의미할 수 있다. R3은 S1으로 두번째 소스 레지스터를 의미할 수 있다.

[118] 도 8을 참조하면 마지막 참조 비트(Last Reference Bit)를 결정하는 WOF(705) 및 캐시 라인을 무효화할 수 있는 RC(703)가 도시된다.

[119] WOF(705)는 각 커널 연산의 시작시에 LRB 벡터(802) 및 초기 프로그램 카운터(Program Counter, PC)(801)(이하, PC)를 로딩 할 수 있다. 프로그램 카운터는 프로세서 내부에 있는 레지스터 중 하나로 다음에 실행될 명령어의 주소를 가지고 있다.

[120] PC(803) 및 소스 피연산자 인덱스(Source OP. Index)(804)는 각각 64 비트(8 바이트) 및 2 비트일 수 있다. 명령어의 크기가 8 바이트(2^3)이므로, 2개의 연속한 PC의 차이는 항상 8일 수 있다. 그러므로, 소스 피연산자 인덱스가 5보다 작은 경우, WOF(705)는 초기 PC(801)를 입력 PC(803)에서 뺄 수 있다.

[121] 이때, 초기 PC(801)를 입력 PC(803)에서 뺀 출력 값은 초기 커널 명령어의 명령어 수의 8배를 의미할 수 있다. 따라서, 초기 커널 명령어의 명령어 인덱스는 초기 PC를 입력 PC에서 뺀 출력 값에서 3비트를 이동시켜 결정할 수 있다. 초기 명령어의 명령어 인덱스는 초기 PC를 입력 PC에서 뺀 출력 값에서 8을 나누는 연산을 수행하여 결정할 수 있다.

[122] 결정된 명령어 인덱스를 이용하여 WOF(705)는 LRB 벡터를 4 대 1 MUX(805)에 입력할 수 있다. WOF(705)는 소스 피연산자 인덱스(Source OP. Index)(804)를 MUX의 선택 비트로 사용할 수 있다. 예를 들어 첫번째 LRB 벡터 [0 0 1 0]이 MUX에 입력되고, 소스 피연산자 인덱스(예: 10)는 벡터의 세번째 비트를 선택할 수 있다.

[123] 마지막 참조인 4 대 1 MUX(805)의 출력 비트는 현재 들어오는 레지스터 읽기 요청의 LRB(Last Reference Bit)(806)이며 RC(703)으로 전달될 수 있다.

[124] 각 GPU 커널 코드가 상대적으로 짧기 때문에, 본 발명의 일실시예에서 설계된 WOF(705)는 1024개의 LRB 벡터(512 바이트에 해당함)를 보유할 수 있다. 1024개 이상의 명령어가 있는 경우, WOF(705)는 1025번째 명령어에서 RC(703)으로 항상 거짓(제로) 신호를 생성하고, 이 명령어는 항상 Write-Back 연산을 수행한다. 즉, 1024개 이상의 명령어가 있는 경우, WOF(705)는 2 대 1 MUX(807)가 항상 0을 출력하도록 할 수 있다. WOF(705)는 2 대 1 MUX(807)에서 출력된 결과 값인 LRB를 RC(703)으로 전달할 수 있다.

[125] RC(703)은 총 256개의 엔트리(808)를 가질 수 있다. 각 엔트리(808)는 1 비트 유효, 6비트 태그 및 1024 비트 데이터로 구성될 수 있다. 엔트리(808)에서 V는 유효(valid), T는 태그(Tag), D는 데이터(Data)를 나타낼 수 있다. RC(703)은 Warp ID(이하 WID)와 Register ID(이하 RID)의 가능한 모든 조합을 나타내기 위해, 총

14 비트(6 비트 warp, 8비트 레지스터)를 사용하여 인덱스/태그 계산기(809)를 통해 각 캐시 라인의 인덱스와 태그를 계산할 수 있다.

[126] 인덱스를 계산한 후, 인덱싱 된 캐시 라인의 태그를 계산된 태그와 비교하여 선택된 캐시 라인이 원하는 것인지 여부를 확인할 수 있다. 즉, RC(703)의 각 엔트리는 여러 Warp 및 레지스터에 의해 공유될 수 있으므로, 읽기 요청을 생성하는 Warp와 레지스터의 조합이 선택된 캐시 라인에 저장되어 있는지 여부를 확인할 수 있다.

[127] 태그 비교가 유효하고 RC에 대한 요청 타입이 읽기이면, 데이터가 피연산자 수집 장치(706)에 로딩될 수 있다.

[128] 이 경우, 데이터가 피연산자 수집 장치(706)에 로딩된 후 하나의 추가적인 단계가 수행될 수 있다.

[129] 읽기 요청은 항상 WOF(705)로부터 전달된 마지막 참조 비트(806)와 함께 RC(703)에 삽입될 수 있다. 마지막 참조 비트가 거짓(제로) 값인 경우, 캐시 라인은 곧 다시 사용될 수 있으므로, 캐시 라인의 유효한 필드는 수정되어서는 안 된다. 반면에 현재 들어오는 요청이 캐시 라인을 마지막으로 사용하는 경우, RC(703)는 데이터를 읽은 후에 해당 캐시 라인을 무효화할 수 있다. 즉, 마지막 참조 비트가 참 값인 경우 RC(703)는 해당 캐시 라인을 제거할 수 있다.

[130] 무효화 단계를 지원하기 위해, 1개의 AND 게이트(811)와 한 개의 2 대 1 MUX(810)를 추가할 수 있다. AND 게이트(811)는 WOF(705)의 히트/미스 비트(Hit/Miss bit) 및 마지막 참조 비트(806)와 연결될 수 있다.

[131] Warp 명령어가 필요로 하는 데이터는 RC(703) 또는 메인 레지스터에 저장되어 있을 수 있다. Warp 명령어가 RC(703)에 접근하였는데 연산에 필요한 데이터가 RC(703)가 아닌 메인 레지스터인 STT-MRAM 레지스터(704)에 저장되어 있는 경우 미스 비트(Miss bit)를 생성할 수 있다. Warp 명령어가 RC(703)에 접근하였는데 연산에 필요한 데이터가 RC(703)에 저장되어 있는 경우 히트 비트(Hit bit)를 생성할 수 있다.

[132] 따라서, AND 게이트(811)는 Warp 명령어의 읽기 요청이 히트하고 현재 요청이 마지막 읽기 요청인 경우 활성화될 수 있다. AND 게이트(811)의 출력이 활성화되는 경우, MUX(810)는 거짓(제로) 값을 전달하여 캐시라인을 무효화할 수 있다.

[133] Warp 명령어의 읽기 요청의 결과가 미스이거나, 현재 진행중인 요청이 마지막 읽기 요청이 아닌 경우, AND 게이트(811)는 활성화되지 않을 수 있다. 이 경우, MUX(810)는 원래 값을 보존하기 위해 캐시 라인에서 원래 유효 필드를 전달할 수 있다.

[134] 무효화 과정은 RC(703)에서 데이터를 읽은 후에 완료될 수 있으므로, 데이터를 읽는 중요한 경로에 있지 않다.

[135]

[136] 도 9는 본 발명의 일실시예에 따른 Write-Back 연산 필터가 마지막 참조 비트(Last Reference Bit)를 결정하는 방법을 도시한 플로우차트이다.

[137] 단계(901)에서, WOF(705)는 각 커널 연산의 시작시에 LRB 벡터 및 초기 프로그램 카운터(Program Counter, PC)(이하, PC)를 로딩 할 수 있다. 프로그램 카운터는 프로세서 내부에 있는 레지스터 중 하나로 다음에 실행될 명령어의 주소를 가지고 있다. PC의 크기 및 소스 피연산자 인덱스(Source OP. Index)는 각각 64 비트(8 바이트) 및 2 비트일 수 있다.

[138] 단계(902)에서, 피연산자 인덱스가 5보다 작은 경우, WOF(705)는 초기 PC를 입력 PC에서 뺄 수 있다. 명령어의 크기가 8 바이트(2^3)이므로, 2개의 연속한 PC의 차이는 항상 8일 수 있다. 이때, 계산된 출력 값은 초기 커널 명령어의 명령어 수의 8배를 의미할 수 있다.

[139] 단계(903)에서, WOF(705)는 초기 커널 명령어의 명령어 인덱스를 초기 PC를 입력 PC에서 뺀 출력 값에서 3비트를 이동시켜 결정할 수 있다. WOF(705)는 초기 커널 명령어의 명령어 인덱스를 초기 PC를 입력 PC에서 뺀 출력 값에 대해 8을 나누는 연산을 통해 결정할 수 있다.

[140] 단계(904)에서, WOF(705)는 결정된 명령어 인덱스를 이용하여 LRB 벡터를 4 대 1 MUX에 입력할 수 있다. WOF(705)는 소스 피연산자 인덱스(Source OP. Index)를 MUX의 선택 비트로 사용할 수 있다.

[141] 단계(905)에서, WOF(705)는 4 대 1 MUX의 출력 값인 출력 비트를 RC(703)으로 전달할 수 있다. 출력 비트는 현재 들어오는 레지스터 읽기 요청의 LRB일 수 있다.

[142] 각 GPU 커널 코드가 상대적으로 짧기 때문이에, 본 발명의 일실시예에서 설계된 WOF(705)는 1024개의 LRB 벡터(512 바이트에 해당함)를 보유할 수 있다. 1024개 이상의 명령어가 있는 경우, WOF(705)는 1025번째 명령어에서 RC(703)으로 항상 거짓(제로) 신호를 생성하고, 이 명령어는 항상 Write-Back 연산을 수행한다. 즉, 1024개 이상의 명령어가 있는 경우, WOF(705)는 2 대 1 MUX가 항상 0을 출력하도록 할 수 있다.

[143]

[144] 도 10은 레지스터 캐시가 수행하는 캐시 라인의 무효화 방법을 설명하기 위한 플로우 차트이다.

[145] 단계(1001)에서, RC(703)는 읽기 요청 및 WOF(705)로부터 마지막 참조 비트를 수신할 수 있다.

[146] 단계(1002)에서, RC(703)는 수신된 마지막 참조 비트가 거짓(0) 또는 참(1)인지 여부에 따라 캐시라인을 제거할 수 있다.

[147] 즉, 마지막 참조 비트가 거짓(0)인 경우, 캐시 라인은 곧 다시 사용될 수 있으므로, 캐시 라인의 유효한 필드는 수정되어서는 안 된다. 반면에 현재 들어오는 요청이 캐시 라인을 마지막으로 사용하는 경우, RC(703)는 데이터를 읽은 후에 캐시 라인을 무효화할 수 있다.

[148]

[149] 한편, 본 발명에 따른 방법은 컴퓨터에서 실행될 수 있는 프로그램으로 작성되어 마그네틱 저장매체, 광학적 판독매체, 디지털 저장매체 등 다양한 기록 매체로도 구현될 수 있다.

[150] 본 명세서에 설명된 각종 기술들의 구현들은 디지털 전자 회로조직으로, 또는 컴퓨터 하드웨어, 펌웨어, 소프트웨어로, 또는 그들의 조합들로 구현될 수 있다. 구현들은 데이터 처리 장치, 예를 들어 프로그램가능 프로세서, 컴퓨터, 또는 다수의 컴퓨터들의 동작에 의한 처리를 위해, 또는 이 동작을 제어하기 위해, 컴퓨터 프로그램 제품, 즉 정보 캐리어, 예를 들어 기계 판독가능 저장 장치(컴퓨터 판독가능 매체) 또는 전파 신호에서 유형적으로 구체화된 컴퓨터 프로그램으로서 구현될 수 있다. 상술한 컴퓨터 프로그램(들)과 같은 컴퓨터 프로그램은 컴파일된 또는 인터프리트된 언어들을 포함하는 임의의 형태의 프로그래밍 언어로 기록될 수 있고, 독립형 프로그램으로서 또는 모듈, 구성요소, 서브루틴, 또는 컴퓨팅 환경에서의 사용에 적절한 다른 유닛으로서 포함하는 임의의 형태로 전개될 수 있다. 컴퓨터 프로그램은 하나의 사이트에서 하나의 컴퓨터 또는 다수의 컴퓨터들 상에서 처리되도록 또는 다수의 사이트들에 걸쳐 분배되고 통신 네트워크에 의해 상호 연결되도록 전개될 수 있다.

[151] 컴퓨터 프로그램의 처리에 적절한 프로세서들은 예로서, 범용 및 특수 목적 마이크로프로세서들 둘 다, 및 임의의 종류의 디지털 컴퓨터의 임의의 하나 이상의 프로세서들을 포함한다. 일반적으로, 프로세서는 판독 전용 메모리 또는 랜덤 액세스 메모리 또는 둘 다로부터 명령어들 및 데이터를 수신할 것이다. 컴퓨터의 요소들은 명령어들을 실행하는 적어도 하나의 프로세서 및 명령어들 및 데이터를 저장하는 하나 이상의 메모리 장치들을 포함할 수 있다. 일반적으로, 컴퓨터는 데이터를 저장하는 하나 이상의 대량 저장 장치들, 예를 들어 자기, 자기-광 디스크들, 또는 광 디스크들을 포함할 수 있거나, 이것들로부터 데이터를 수신하거나 이것들에 데이터를 송신하거나 또는 양쪽으로 되도록 결합될 수도 있다. 컴퓨터 프로그램 명령어들 및 데이터를 구체화하는데 적절한 정보 캐리어들은 예로서 반도체 메모리 장치들, 예를 들어, 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic Media), CD-ROM(Compact Disk Read Only Memory), DVD(Digital Video Disk)와 같은 광 기록 매체(Optical Media), 플롭티컬 디스크(Floptical Disk)와 같은 자기-광 매체(Magneto-Optical Media), 롬(ROM, Read Only Memory), 램(RAM, Random Access Memory), 플래시 메모리, EPROM(Erasable Programmable ROM), EEPROM(Electrically Erasable Programmable ROM) 등을 포함한다. 프로세서 및 메모리는 특수 목적 논리 회로조직에 의해 보충되거나, 이에 포함될 수 있다.

[152] 또한, 컴퓨터 판독가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 가용매체일 수 있고, 컴퓨터 저장매체 및 전송매체를 모두 포함할 수 있다.

[153] 본 명세서는 다수의 특정한 구현물의 세부사항들을 포함하지만, 이들은 어떠한 발명이나 청구 가능한 것의 범위에 대해서도 제한적인 것으로서 이해되어서는

안되며, 오히려 특정한 발명의 특정한 실시형태에 특유할 수 있는 특징들에 대한 설명으로서 이해되어야 한다. 개별적인 실시형태의 문맥에서 본 명세서에 기술된 특정한 특징들은 단일 실시형태에서 조합하여 구현될 수도 있다. 반대로, 단일 실시형태의 문맥에서 기술한 다양한 특징들 역시 개별적으로 혹은 어떠한 적절한 하위 조합으로도 복수의 실시형태에서 구현 가능하다. 나아가, 특징들이 특정한 조합으로 동작하고 초기에 그와 같이 청구된 바와 같이 묘사될 수 있지만, 청구된 조합으로부터의 하나 이상의 특징들은 일부 경우에 그 조합으로부터 배제될 수 있으며, 그 청구된 조합은 하위 조합이나 하위 조합의 변형물로 변경될 수 있다.

[154] 마찬가지로, 특정한 순서로 도면에서 동작들을 묘사하고 있지만, 이는 바람직한 결과를 얻기 위하여 도시된 그 특정한 순서나 순차적인 순서대로 그러한 동작들을 수행하여야 한다거나 모든 도시된 동작들이 수행되어야 하는 것으로 이해되어서는 안 된다. 특정한 경우, 멀티태스킹과 병렬 프로세싱이 유리할 수 있다. 또한, 상술한 실시형태의 다양한 장치 컴포넌트의 분리는 그러한 분리를 모든 실시형태에서 요구하는 것으로 이해되어서는 안되며, 설명한 프로그램 컴포넌트와 장치들은 일반적으로 단일의 소프트웨어 제품으로 함께 통합되거나 다중 소프트웨어 제품에 패키징 될 수 있다는 점을 이해하여야 한다.

[155] 한편, 본 명세서와 도면에 개시된 본 발명의 실시 예들은 이해를 돕기 위해 특정 예를 제시한 것에 지나지 않으며, 본 발명의 범위를 한정하고자 하는 것은 아니다. 여기에 개시된 실시 예들 이외에도 본 발명의 기술적 사상에 바탕을 둔 다른 변형 예들이 실시 가능하다는 것은, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 자명한 것이다.

청구범위

[청구항 1] 컴퓨팅 장치의 프로세서가 수행하는 명령어 연산 수행 방법에 있어서,
프로세서가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리를 함께 사용하는 레지스터 파일로부터 읽는 단계;
상기 데이터에 기초하여 상기 연산장치를 이용해 상기 명령어 연산을 수행하는 단계; 및
상기 명령어 연산을 통해 도출된 결과 값을 상기 휘발성 메모리에 업데이트하는 단계
를 포함하는 프로세서의 명령어 연산 수행 방법.

[청구항 2] 제1항에 있어서,
상기 프로세서가 명령어 의존성 정보를 이용하여 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계; 및
상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 휘발성 메모리에 저장된 데이터를 제거하는 단계
를 더 포함하는 프로세서의 명령어 연산 수행 방법.

[청구항 3] 제2항에 있어서,
상기 명령어 의존성 정보는,
상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보인 프로세서의 명령어 연산 수행 방법.

[청구항 4] 제1항에 있어서,
상기 휘발성 메모리는,
명령어 의존성 정보에 의해 상기 명령어 연산을 수행하기 위한 상기 휘발성 메모리에 저장된 데이터가 제거된 이후 상기 명령어 연산의 결과 값이 업데이트되는 프로세서의 명령어 연산 수행 방법.

[청구항 5] 제1항에 있어서,
상기 휘발성 메모리는,
정적 랜덤 액세스 메모리인 프로세서의 명령어 연산 수행 방법.

[청구항 6] 제1항에 있어서,
상기 비휘발성 메모리는,
스핀전달토크형 자기저항메모리 또는 스핀궤도토크형 자기저항메모리인 프로세서의 명령어 연산 수행 방법.

[청구항 7] 제1항에 있어서,
상기 레지스터 파일은,

쓰기 연산 필터를 포함하고,
상기 쓰기 연산 필터는 마지막 참조 비트 벡터 및 소스 피연산자 인덱스에 기초하여 마지막 참조 비트를 결정하는 프로세서의 명령어 연산 수행 방법.

[청구항 8] 제7항에 있어서,
상기 휘발성 메모리는,
읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 거짓 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화하지 않는 프로세서의 명령어 연산 수행 방법.

[청구항 9] 제7항에 있어서,
상기 휘발성 메모리는,
읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 참 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화하는 프로세서의 명령어 연산 수행 방법.

[청구항 10] 컴파일러가 프로그램의 컴파일 과정에서 상기 프로그램을 실행하는 명령어 연산에 이용되는 명령어를 순서대로 확인하는 단계;
레지스터에 저장된 데이터가 상기 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계; 및
상기 레지스터에 저장된 데이터가 상기 명령어에 의해 이용된 이후에 추가로 이용되지 않는 경우, 상기 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보인 명령어 의존성 정보를 결정하는 단계
를 포함하는 명령어 의존성 정보를 생성하는 방법.

[청구항 11] 제10항에 있어서,
상기 레지스터에 저장된 데이터가 상기 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단하는 단계는,
상기 레지스터에 저장된 데이터가 상기 명령어의 후속 명령어에 의해 업데이트되는지 여부에 따라 상기 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단하는 명령어 의존성 정보를 생성하는 방법.

[청구항 12] 명령어 연산 수행 방법을 수행하는 컴퓨팅 장치는 프로세서를 포함하고,
상기 프로세서는,
프로세서가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리를 함께 사용하는 레지스터 파일로부터 읽고,
상기 데이터에 기초하여 상기 연산장치를 이용해 상기 명령어 연산을 수행하고,
상기 명령어 연산을 통해 도출된 결과 값을 상기 휘발성 메모리에 업데이트하는, 컴퓨팅 장치.

[청구항 13] 제12항에 있어서,

상기 프로세서는,
상기 프로세서가 명령어 의존성 정보를 이용하여 상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되는지 여부를 판단하고,
상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 휘발성 메모리에 저장된 데이터를 제거하는, 컴퓨팅 장치.

[청구항 14] 제13항에 있어서,
상기 명령어 의존성 정보는,
상기 명령어 연산을 수행하기 위한 데이터가 상기 명령어 연산에 이용된 이후에 추가로 이용되지 않는 경우, 상기 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보인, 컴퓨팅 장치.

[청구항 15] 제12항에 있어서,
상기 휘발성 메모리는,
명령어 의존성 정보에 의해 상기 명령어 연산을 수행하기 위한 상기 휘발성 메모리에 저장된 데이터가 제거된 이후 상기 명령어 연산의 결과 값이 업데이트되는, 컴퓨팅 장치.

[청구항 16] 제12항에 있어서,
상기 휘발성 메모리는,
정적 랜덤 액세스 메모리인, 컴퓨팅 장치.

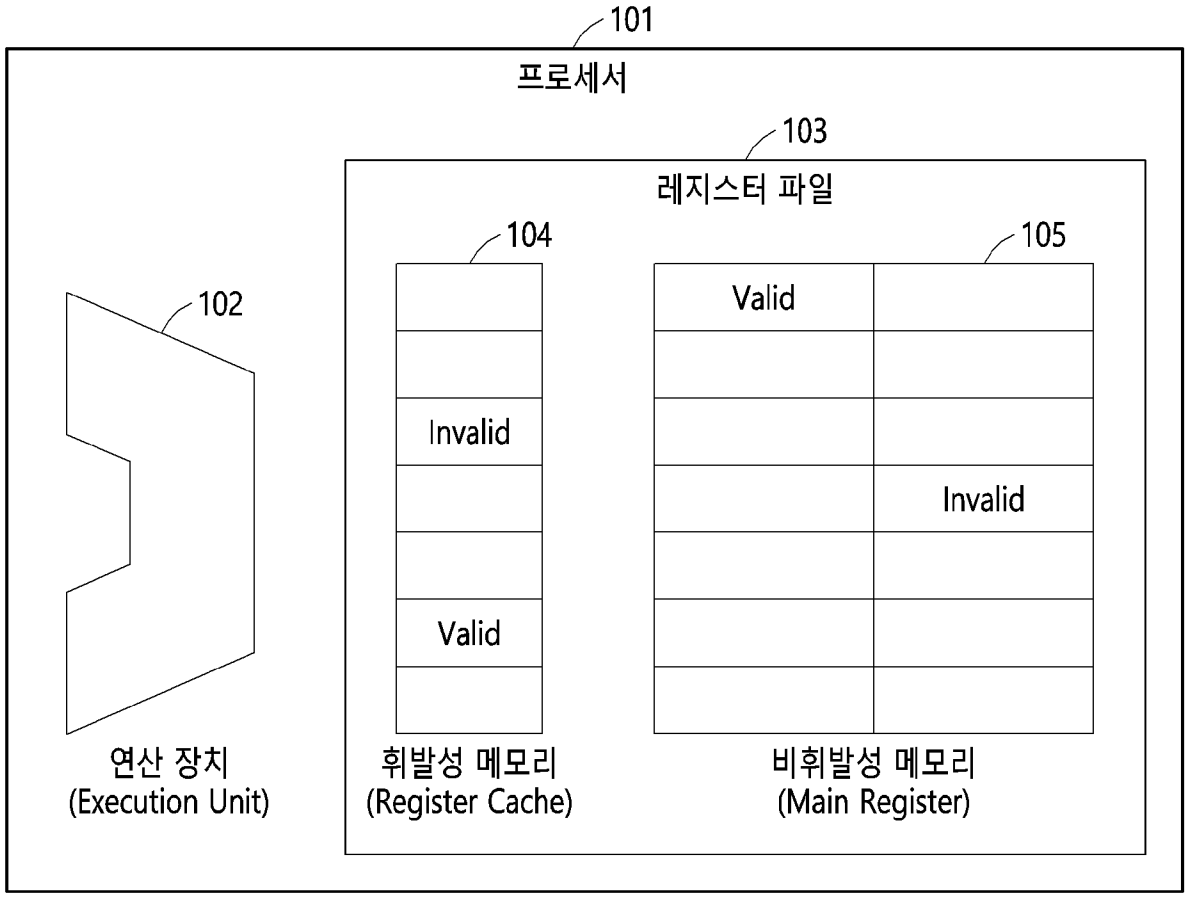
[청구항 17] 제12항에 있어서,
상기 비휘발성 메모리는,
스핀전달토크형 자기저항메모리 또는 스핀궤도토크형 자기저항메모리인, 컴퓨팅 장치.

[청구항 18] 제12항에 있어서,
상기 레지스터 파일은,
쓰기 연산 필터를 포함하고,
상기 쓰기 연산 필터는 마지막 참조 비트 벡터 및 소스 피연산자 인덱스에 기초하여 마지막 참조 비트를 결정하는 컴퓨팅 장치.

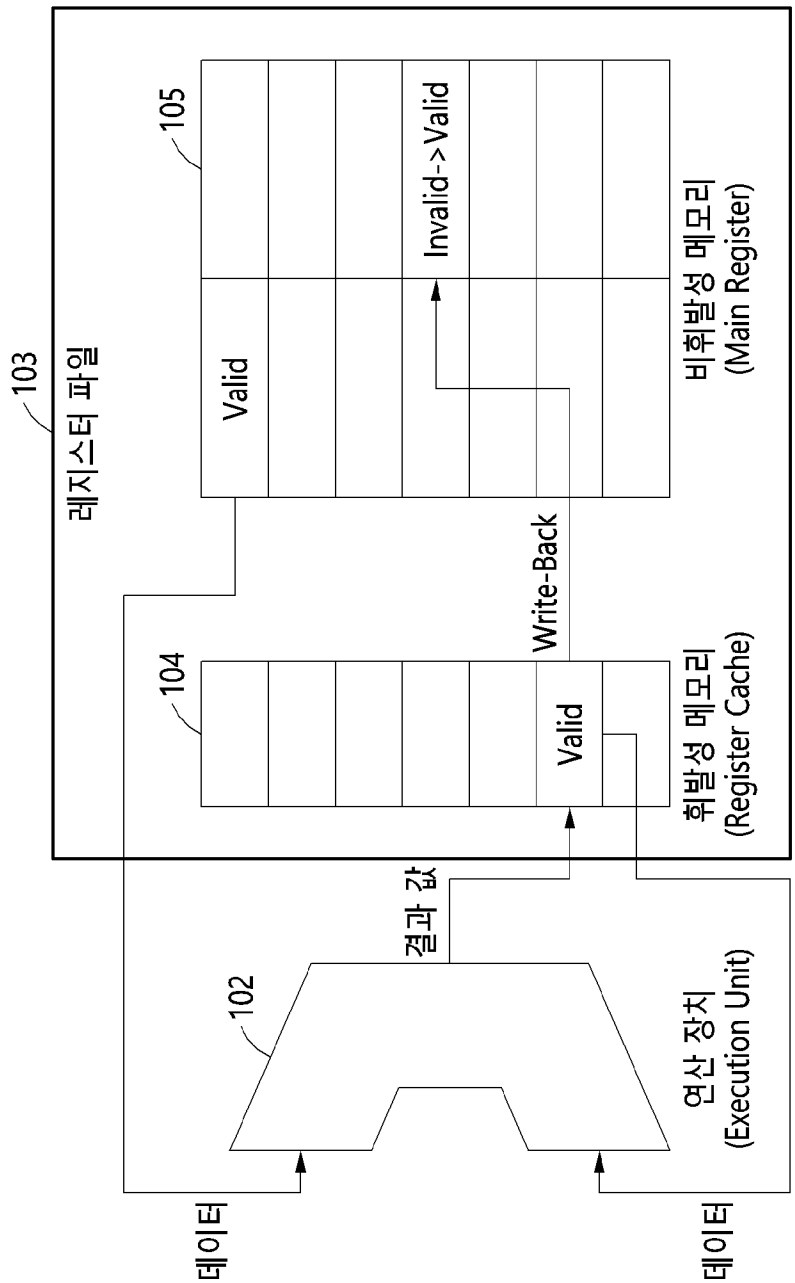
[청구항 19] 제18항에 있어서,
상기 휘발성 메모리는,
읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 거짓 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화하지 않는 컴퓨팅 장치

[청구항 20] 제18항에 있어서,
상기 휘발성 메모리는,
읽기 요청 및 상기 마지막 참조 비트에 기초하여, 마지막 참조 비트가 참 값인 경우 상기 휘발성 메모리의 캐시 라인을 무효화하는 컴퓨팅 장치.

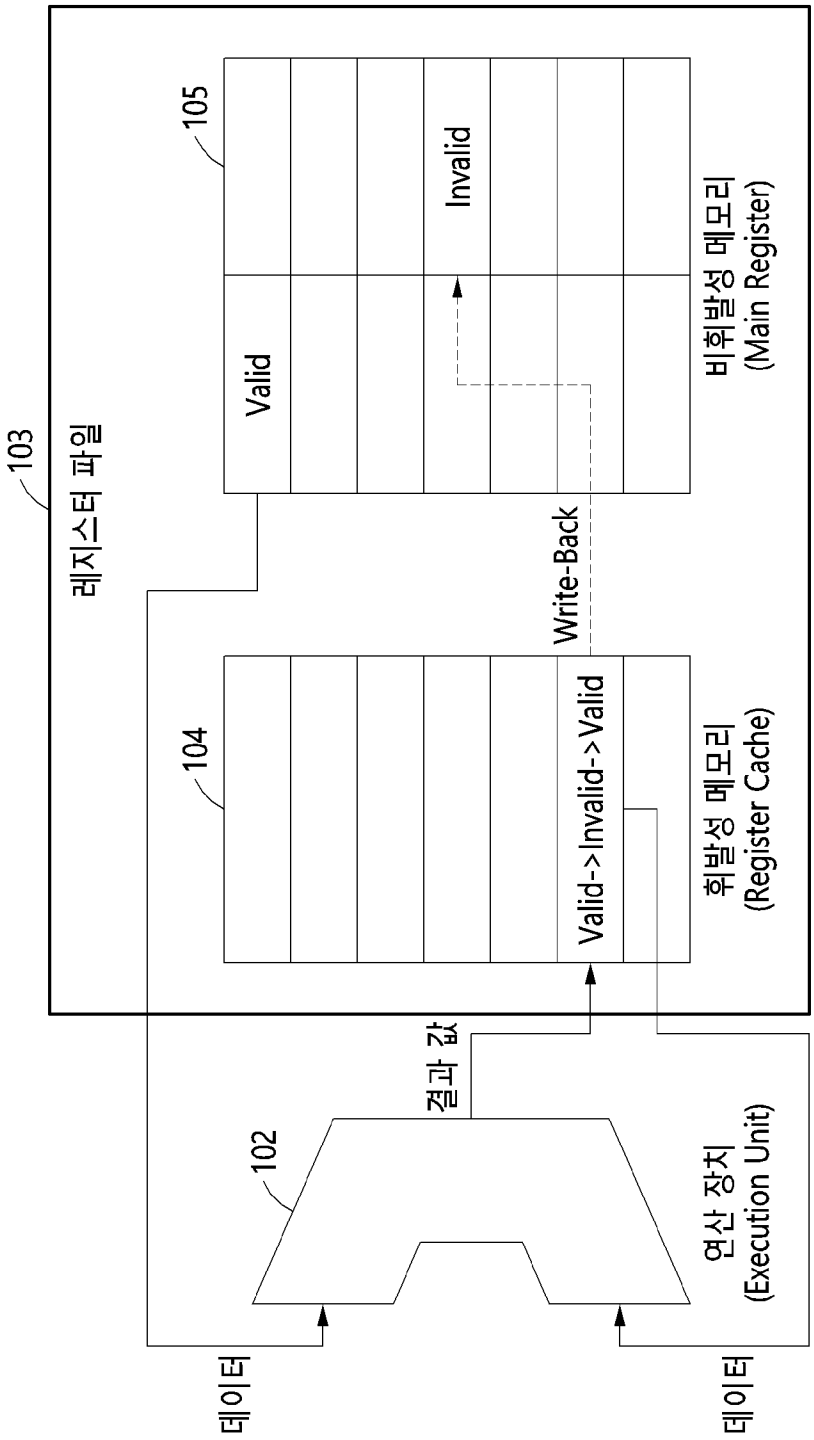
[도1]


101
프로세서
103
레지스터 파일
104
105
102
Valid
Invalid
Invalid
Valid
연산 장치
(Execution Unit)
휘발성 메모리
(Register Cache)
비휘발성 메모리
(Main Register)

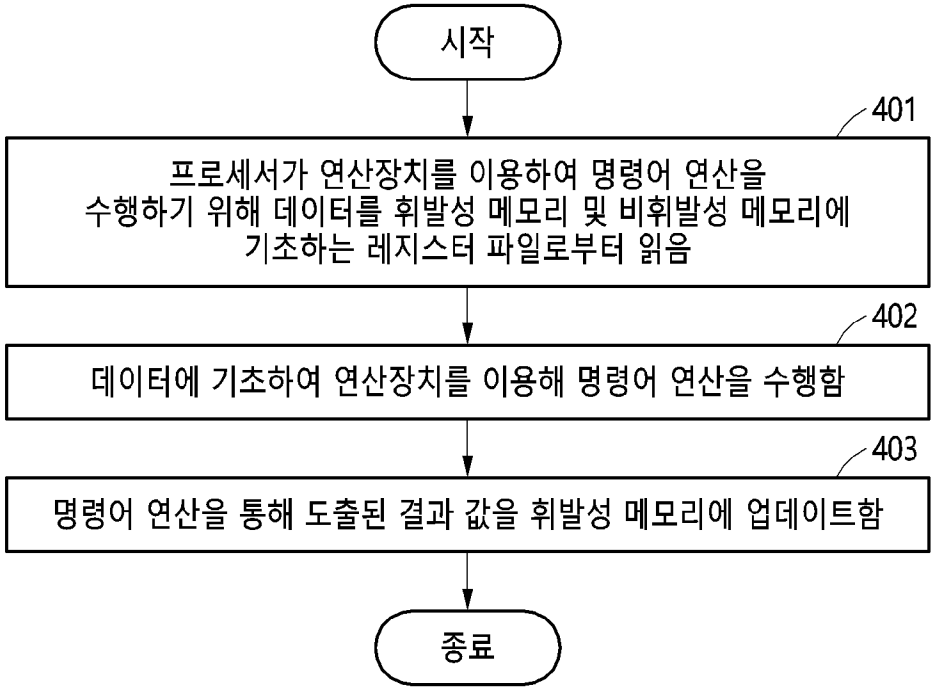
[도2]


103
레지스터 파일
105
Valid
Invalid->Valid
비휘발성 메모리
(Main Register)
Write-Back
104
Valid
휘발성 메모리
(Register Cache)
결과 값
102
연산 장치
(Execution Unit)
데이터
데이터

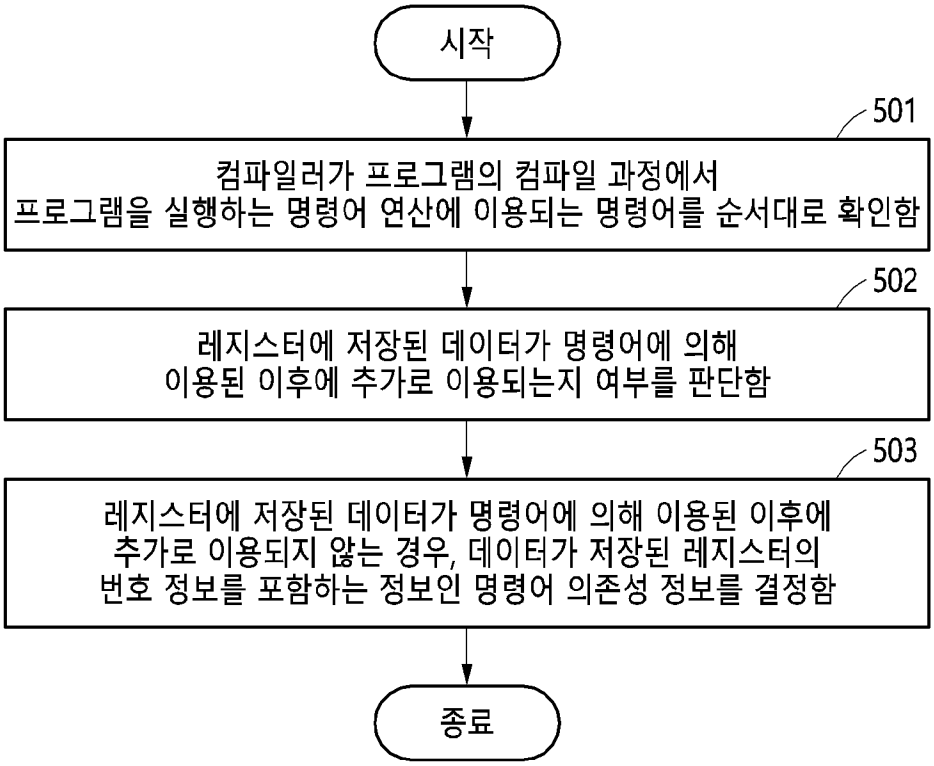
[도3]


103
레지스터 파일
105
Valid
Invalid
비휘발성 메모리
(Main Register)
Write-Back
104
Valid->Invalid->Valid
휘발성 메모리
(Register Cache)
결과 값
102
연산 장치
(Execution Unit)
데이터
데이터

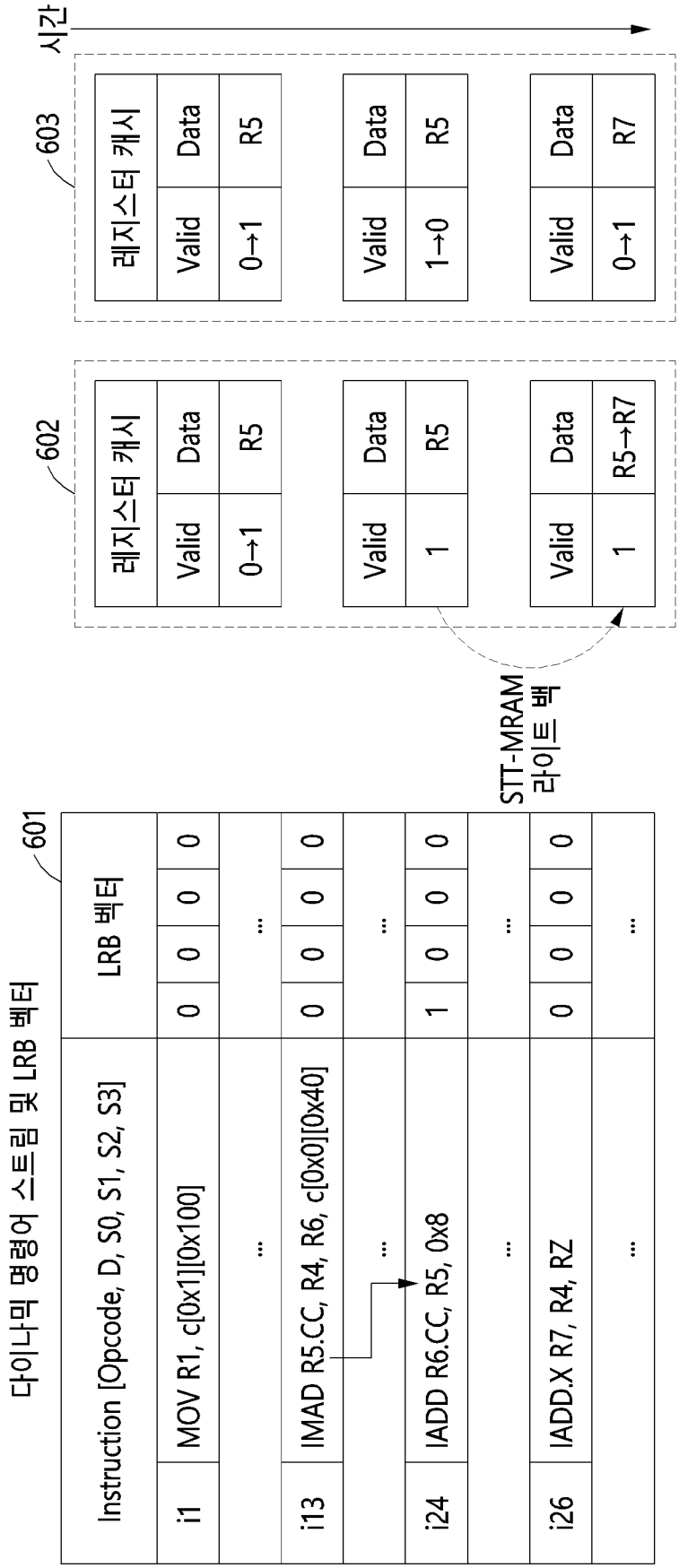
[도4]


시작
401
프로세서가 연산장치를 이용하여 명령어 연산을 수행하기 위해 데이터를 휘발성 메모리 및 비휘발성 메모리에 기초하는 레지스터 파일로부터 읽음
402
데이터에 기초하여 연산장치를 이용해 명령어 연산을 수행함
403
명령어 연산을 통해 도출된 결과 값을 휘발성 메모리에 업데이트함
종료

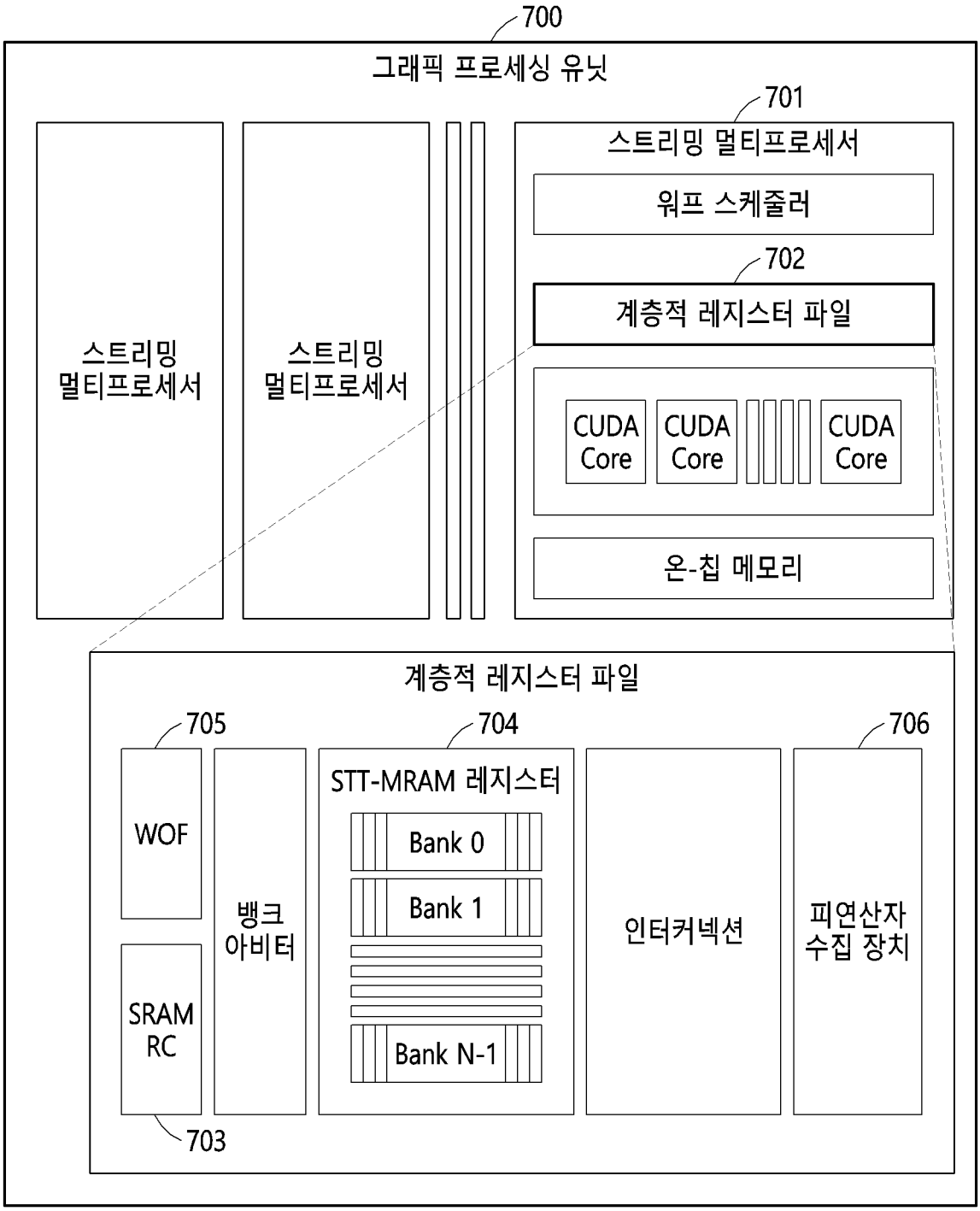
[도5]


시작
501
컴파일러가 프로그램의 컴파일 과정에서 프로그램을 실행하는 명령어 연산에 이용되는 명령어를 순서대로 확인함
502
레지스터에 저장된 데이터가 명령어에 의해 이용된 이후에 추가로 이용되는지 여부를 판단함
503
레지스터에 저장된 데이터가 명령어에 의해 이용된 이후에 추가로 이용되지 않는 경우, 데이터가 저장된 레지스터의 번호 정보를 포함하는 정보인 명령어 의존성 정보를 결정함
종료

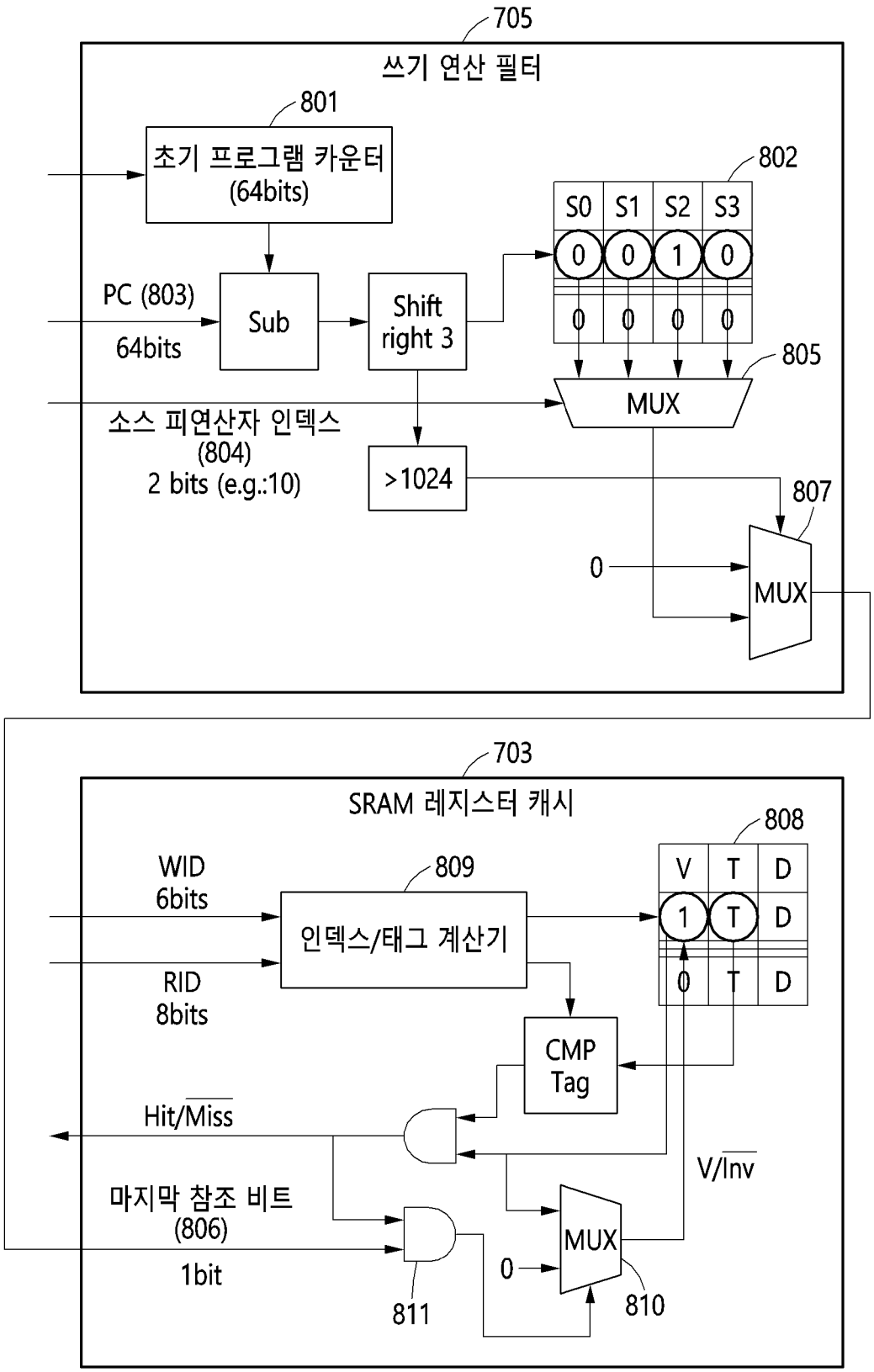
[도6]


시간
603
레지스터 캐시
Valid
Data
R5
0→1
1→0
R7
602
1
R5→R7
STT-MRAM
라이트 백
601
다이나믹 명령어 스트림 및 LRB 벡터
Instruction [Opcode, D, S0, S1, S2, S3]
LRB 벡터
i1
MOV R1, c[0x1][0x100]
i13
IMAD R5.CC, R4, R6, c[0x0][0x40]
i24
IADD R6.CC, R5, 0x8
i26
IADD.X R7, R4, RZ

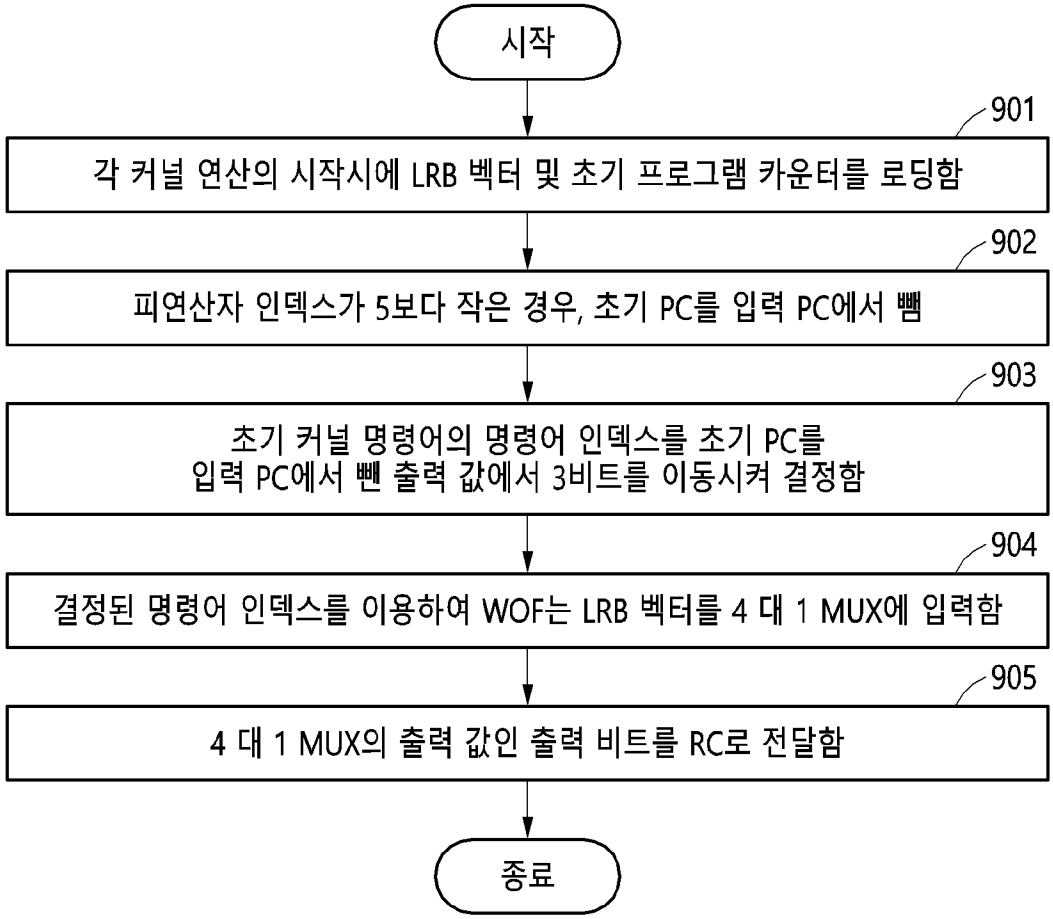
[도7]


700
그래픽 프로세싱 유닛
701
스트리밍 멀티프로세서
워프 스케줄러
702
계층적 레지스터 파일
CUDA Core
CUDA Core
CUDA Core
온-칩 메모리
스트리밍 멀티프로세서
스트리밍 멀티프로세서
계층적 레지스터 파일
705
WOF
SRAM RC
703
뱅크 아비터
704
STT-MRAM 레지스터
Bank 0
Bank 1
Bank N-1
인터커넥션
706
피연산자 수집 장치

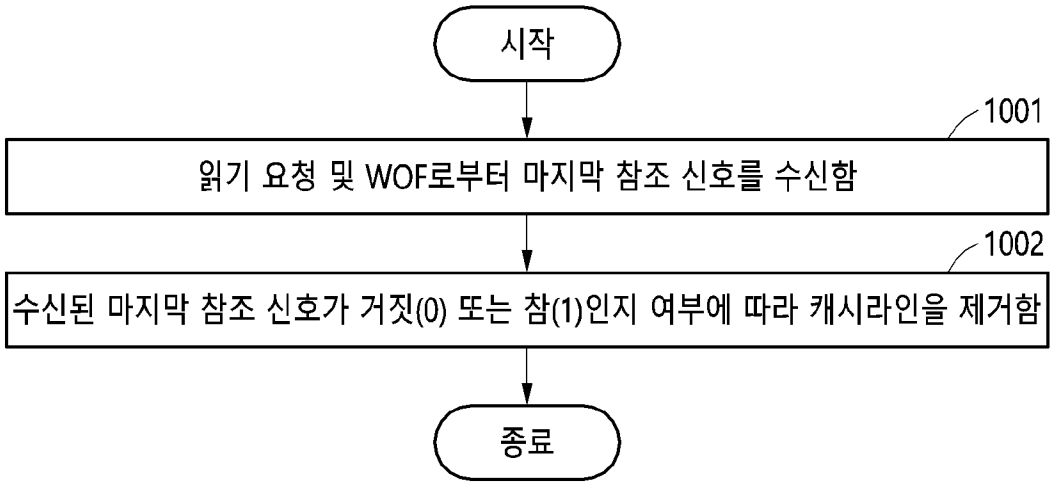
[도 8]


705
쓰기 연산 필터
801
초기 프로그램 카운터
(64bits)
802
S0 S1 S2 S3
0 0 1 0
0 0 0 0
PC (803)
64bits
Sub
Shift
right 3
805
MUX
소스 피연산자 인덱스
(804)
2 bits (e.g.:10)
>1024
807
0
MUX
703
SRAM 레지스터 캐시
808
V T D
1 T D
0 T D
WID
6bits
809
인덱스/태그 계산기
RID
8bits
CMP
Tag
Hit/Miss
V/Inv
마지막 참조 비트
(806)
1bit
MUX
0
811
810

[도9]


시작
901
각 커널 연산의 시작시에 LRB 벡터 및 초기 프로그램 카운터를 로딩함
902
피연산자 인덱스가 5보다 작은 경우, 초기 PC를 입력 PC에서 뺌
903
초기 커널 명령어의 명령어 인덱스를 초기 PC를
입력 PC에서 뺀 출력 값에서 3비트를 이동시켜 결정함
904
결정된 명령어 인덱스를 이용하여 WOF는 LRB 벡터를 4 대 1 MUX에 입력함
905
4 대 1 MUX의 출력 값인 출력 비트를 RC로 전달함
종료

[도10]


시작
1001
읽기 요청 및 WOF로부터 마지막 참조 신호를 수신함
1002
수신된 마지막 참조 신호가 거짓(0) 또는 참(1)인지 여부에 따라 캐시라인을 제거함
종료

INTERNATIONAL SEARCH REPORT

International application No.
PCT/KR2023/003208

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/30(2006.01)i; **G06F 9/38**(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/30(2006.01); G06F 12/02(2006.01)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models: IPC as above
Japanese utility models and applications for utility models: IPC as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS (KIPO internal) & keywords: 컴파일러(compiler), 메모리(memory), 명령어(instruction), 연산(operation), GPU

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	OH, Yunho et al. CASH-RF: A Compiler-Assisted Hierarchical Register File in GPUs. IEEE Embedded Systems Letters. Vol. 14, No. 4, pp. 187–190, 31 March 2022. [Retrieved on 22 May 2023]. Retrieved from <https://ieeexplore.ieee.org/abstract/document/9745582>. See abstract; and pages 187-189.	1-20
A	JEON, Won et al. Hi-End: Hierarchical, Endurance-Aware STT-MRAM-Based Register File for Energy-Efficient GPUs. IEEE Access. Vol. 8, pp. 127768-127780, 13 July 2020. [Retrieved on 22 May 2023]. Retrieved from <https://ieeexplore.ieee.org/abstract/document/9139253>. See pages 127768-127779; and figures 1-15.	1-20
A	LIU, Xiaoxiao et al. An efficient STT-RAM-based register file in GPU architectures. The 20th Asia and South Pacific Design Automation Conference. pp. 490-495, 19 January 2015. [Retrieved on 22 May 2023]. Retrieved from <https://ieeexplore.ieee.org/abstract/document/7059054>. See pages 490-495; and figures 1-10.	1-20

☑ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
01 June 2023	**01 June 2023**

Name and mailing address of the ISA/KR	Authorized officer
Korean Intellectual Property Office **Government Complex-Daejeon Building 4, 189 Cheongsa-ro, Seo-gu, Daejeon 35208**	
Facsimile No. **+82-42-481-8578**	Telephone No.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/KR2023/003208

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	ZHANG, Jie et al. FUSE: Fusing STT-MRAM into GPUs to Alleviate Off-Chip Memory Access Overheads. arXiv:1903.01776v2 [cs.AR]. pp. 1-15, 09 March 2019. [Retrieved on 22 May 2022]. Retrieved from <https://arxiv.org/pdf/1903.01776.pdf>. See pages 1-13; and figures 1-20.	1-20
A	KR 10-2021-0088683 A (ADVANCED MICRO DEVICES, INC.) 14 July 2021 (2021-07-14) See paragraphs [0016]-[0071]; claims 1-20; and figures 1-14.	1-20

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/KR2023/003208

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
KR 10-2021-0088683 A	14 July 2021	CN 113168378 A	23 July 2021
		EP 3895025 A1	20 October 2021
		JP 2022-510715 A	27 January 2022
		JP 7108141 B2	27 July 2022
		US 11232039 B2	25 January 2022
		US 2020-0183848 A1	11 June 2020
		US 2022-138107 A1	05 May 2022
		WO 2020-123343 A1	18 June 2020

국 제 조 사 보 고 서

국제출원번호
PCT/KR2023/003208

A. 발명이 속하는 기술분류(국제특허분류(IPC))

G06F 9/30(2006.01)i; **G06F 9/38**(2006.01)i

B. 조사된 분야

조사된 최소문헌(국제특허분류를 기재)

G06F 9/30(2006.01); G06F 12/02(2006.01)

조사된 기술분야에 속하는 최소문헌 이외의 문헌

한국등록실용신안공보 및 한국공개실용신안공보: 조사된 최소문헌란에 기재된 IPC
일본등록실용신안공보 및 일본공개실용신안공보: 조사된 최소문헌란에 기재된 IPC

국제조사에 이용된 전산 데이터베이스(데이터베이스의 명칭 및 검색어(해당하는 경우))

eKOMPASS(특허청 내부 검색시스템) & 키워드: 컴파일러(compiler), 메모리(memory), 명령어(instruction), 연산(operation), GPU

C. 관련 문헌

카테고리*	인용문헌명 및 관련 구절(해당하는 경우)의 기재	관련 청구항
X	YUNHO OH 등, CASH-RF: A Compiler-Assisted Hierarchical Register File in GPUs, IEEE Embedded Systems Letters, Vol. 14, No. 4, pp. 187–190, 2022.03.31 [검색일: 2023-05-22], 출처 <https://ieeexplore.ieee.org/abstract/document/9745582> 초록; 및 페이지 187-189	1-20
A	WON JEON 등, Hi-End: Hierarchical, Endurance-Aware STT-MRAM-Based Register File for Energy-Efficient GPUs, IEEE Access, Vol. 8, pp. 127768-127780, 2020.07.13 [검색일: 2023-05-22], 출처 <https://ieeexplore.ieee.org/abstract/document/9139253> 페이지 127768-127779; 및 도면 1-15	1-20
A	XIAOXIAO LIU 등, An efficient STT-RAM-based register file in GPU architectures, The 20th Asia and South Pacific Design Automation Conference, pp. 490-495, 2015.01.19 [검색일: 2023-05-22], 출처 <https://ieeexplore.ieee.org/abstract/document/7059054> 페이지 490-495; 및 도면 1-10	1-20
A	JIE ZHANG 등, FUSE: Fusing STT-MRAM into GPUs to Alleviate Off-Chip Memory Access Overheads, arXiv:1903.01776v2 [cs.AR], pp. 1-15, 2019.03.09 [검색일: 2023-05-22], 출처 <https://arxiv.org/pdf/1903.01776.pdf> 페이지 1-13; 및 도면 1-20	1-20

☑ 추가 문헌이 C(계속)에 기재되어 있습니다.

☑ 대응특허에 관한 별지를 참조하십시오.

* 인용된 문헌의 특별 카테고리:

"A" 특별히 관련이 없는 것으로 보이는 일반적인 기술수준을 정의한 문헌

"D" 본 국제출원에서 출원인이 인용한 문헌

"E" 국제출원일보다 빠른 출원일 또는 우선일을 가지나 국제출원일 이후에 공개된 선출원 또는 특허 문헌

"L" 우선권 주장에 의문을 제기하는 문헌 또는 다른 인용문헌의 공개일 또는 다른 특별한 이유(이유를 명시)를 밝히기 위하여 인용된 문헌

"O" 구두 개시, 사용, 전시 또는 기타 수단을 언급하고 있는 문헌

"P" 우선일 이후에 공개되었으나 국제출원일 이전에 공개된 문헌

"T" 국제출원일 또는 우선일 후에 공개된 문헌으로, 출원과 상충하지 않으며 발명의 기초가 되는 원리나 이론을 이해하기 위해 인용된 문헌

"X" 특별한 관련이 있는 문헌. 해당 문헌 하나만으로 청구된 발명의 신규성 또는 진보성이 없는 것으로 본다.

"Y" 특별한 관련이 있는 문헌. 해당 문헌이 하나 이상의 다른 문헌과 조합하는 경우로 그 조합이 당업자에게 자명한 경우 청구된 발명은 진보성이 없는 것으로 본다.

"&" 동일한 대응특허문헌에 속하는 문헌

국제조사의 실제 완료일	국제조사보고서 발송일
2023년06월01일(01.06.2023)	**2023년06월01일(01.06.2023)**

ISA/KR의 명칭 및 우편주소	심사관
대한민국 특허청 **(35208) 대전광역시 서구 청사로 189, 4동 (둔산동, 정부대전청사)**	**변성철**
팩스 번호 **+82-42-481-8578**	전화번호 **+82-42-481-8262**

국 제 조 사 보 고 서

국제출원번호
PCT/KR2023/003208

C. 관련 문헌

카테고리*	인용문헌명 및 관련 구절(해당하는 경우)의 기재	관련 청구항
A	KR 10-2021-0088683 A (어드벤스드 마이크로 디바이시즈, 인코포레이티드) 2021.07.14 단락 [0016]-[0071]; 청구항 1-20; 및 도면 1-14	1-20

국 제 조 사 보 고 서
대응특허에 관한 정보

국제출원번호
PCT/KR2023/003208

국제조사보고서에서 인용된 특허문헌	공개일	대응특허문헌	공개일
KR 10-2021-0088683 A	2021/07/14	CN 113168378 A	2021/07/23
		EP 3895025 A1	2021/10/20
		JP 2022-510715 A	2022/01/27
		JP 7108141 B2	2022/07/27
		US 11232039 B2	2022/01/25
		US 2020-0183848 A1	2020/06/11
		US 2022-138107 A1	2022/05/05
		WO 2020-123343 A1	2020/06/18