(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.[7]
G06F 15/00
G06F 11/00

(11) 공개번호 10-2005-0083877
(43) 공개일자 2005년08월26일

(21) 출원번호 10-2005-7007619
(22) 출원일자 2005년04월29일
번역문 제출일자 2005년04월29일
(86) 국제출원번호 PCT/US2003/031313
국제출원일자 2003년10월03일
(87) 국제공개번호 WO 2004/040427
국제공개일자 2004년05월13일

(30) 우선권주장 10/331,879 2002년12월31일 미국(US)
60/421,773 2002년10월29일 미국(US)
60/421,774 2002년10월29일 미국(US)
60/421,775 2002년10월29일 미국(US)

(71) 출원인 록히드 마틴 코포레이션
미국 20817 메릴랜드주 베데스다 락릿지 드라이브6801

(72) 발명자 대프 마이클 씨
미국 뉴욕 13760 앤드웰 아이본 에비뉴 1130
렛트 에릭 씨
미국 뉴욕 13760 앤드웰 캐밀롯 로드 2723

(74) 대리인 특허법인 신성

심사청구 : 없음

(54) 침입 검출 가속기

요약

네트워크에 접속된 컴퓨터 시스템 또는 그 노드 상으로의 가능한 침입 또는 공격, 또는 다른 보안 침해(security breach)를 나타낼 수 있는 문서내의 문자 스트링의 기호는 하드웨어 파서 가속기 환경 내의 하드웨어 가속기를 이용하여 고속으로 검출된다. 따라서, 이러한 보안 침해, 침입 또는 공격을 구성할 수 있는 커맨드(command)가 문서의 파싱(parsing)에 의해 실행가능하게 되기 전에 호스트 CPU에 인터럽트(interrupt) 또는 예외(exception)가 발행될 수 있다. CPU는 침입을 방지 또는 제한하도록 네트워크 제어 측정(network control measures)을 개시할 수 있다.

대표도

도 2b

색인어

보안, 침입, 인터럽트, 네트워크, 가속기

명세서

기술분야

본 발명은 일반적으로 XML™ 문서(documents)와 같은 문서의 파싱(parsing)에 관한 것으로, 특히 네트워크 노드 상의 잠재적인 침입 또는 공격을 검출하기 위한 문서 또는 네트워크 데이터 패킷의 그 밖의 논리 시퀀스를 파싱하는 것에 관한 것이다.

배경기술

최근에, 컴퓨터 간의 디지털 통신 분야와 컴퓨터를 네트워크로 링크시키는 것이 빠르게 개발되고 있고, 마찬가지로 최근 몇년 동안의 개인용 컴퓨터(PC)의 확산에 대한 많은 방식에서 개발이 이루어지고 있다. 원격 처리의 가능성 및 상호접속성에서의 이러한 증가는, 이러한 네트워크 방식 시스템에서 개별 컴퓨터의 효율적인 기능 및 성능을 매우 증가시키고 있다. 그럼에도 불구하고, 컴퓨터가 사용될 때, 개별 컴퓨터 및 시스템의 다양한 사용, 그 사용자들의 기호 및 기술의 상태는 개별 머신 및 그 오퍼레이팅 시스템의 상당한 정도의 다양한 성능 및 구성을 가져왔으며, 이것은 대체로 특히 오퍼레이팅 시스템과 프로그래밍 언어의 레벨에서 어느 정도 서로 호환되지 않는 "플랫폼"으로 집합적으로 언급된다.

플랫폼 특성의 비호환성, 및 통신 및 원격 처리 능력과 그것을 지원하는 상당한 정도의 호환성에 대한 동시 요건은 (엔티티, 속성 및 관계의 참조 시스템을 통해 다소 일반화된 모듈의 그룹으로 데이터 뿐만 아니라 애플리케이션을 모으는 개념을 수용하는) 객체 지향 프로그래밍 및 이것을 구현하는 다수의 프로그래밍 언어의 개발을 가져왔다. 이러한 언어로는, 광범위하게 사용되고, 임의의 구조 및 아키텍처의 네트워크를 통해 문서로서 전송될 수 있는 확장성 생성 언어(Extensible Markup Language™ : XML™)가 있다.

이러한 언어에서, 임의의 문자 스트링(character string)은 특수 문자 및 기타 중요한 데이터를 포함한 임의의 커맨드 또는 식별자에 대응하는데(집합적으로 제어 단어(control words)로 언급됨), 이것은 데이터 또는 동작이 스스로 식별가능하도록 하여, 그후에 "객체(objects)"로 취급될 수 있고, 이러한 관련 데이터 및 커맨드는 주어진 머신에서 요구된 처리를 지원하는데 충분한, 각각 접속된 플랫폼의 호환성을 생성하기 위해, 상이한 언어에서 상이한 애플리케이션의 적절한 포맷 및 커맨드로 변환될 수 있다. 이들 문자 스트링의 검출은 파싱(parsing: 구문분석)으로 알려진 동작에 의해 수행되는데, 이것은 문장과 같은 표현 구문을 그것의 구성 성분으로 분석하고 이를 문법적으로 기술하는 보다 전형적인 방법과 유사한 것이다.

XML™ 문서를 파싱하는 경우, 중앙 처리 장치(CPU) 실행 시간의 대부분은, 처리될 특정 XML™ 표준으로 정의된 제어 단어, 특수 문자 및 기타 중요 데이터에 대한 문서 검색을 트래버싱(traversing)하는데 소비된다. 이것은 통상적으로, 각 문자를 문의하고, 이것이 관심 스트링의 소정 세트, 예를 들면 "<command>", "<data = dataword>", "<endcommand>" 등을 포함한 스트링 세트에 속하는지를 판단하는 소프트웨어에 의해 수행된다. 목표 스트링 중 임의의 것이 검출되면, 토큰이 그 토큰의 시작과 토큰의 길이에 대응하는 문서 내의 위치에 대한 포인터와 함께 저장된다. 이러한 토큰들은 전체 문서가 파싱될 때까지 축적된다.

종래의 접근방법은 관심 스트링을 검색하기 위해 소프트웨어로 테이블-기반 유한 상태 머신(FSM)을 구현하는 것이다. 상태 테이블은 메모리 내에 위치하고, 문서 내의 특정 패턴을 검색하도록 설계된다. 상태 테이블에서 현재 상태가 기본 주소로 사용되고, 입력 문자의 ASCII 표현이 테이블에서 인덱스가 된다. 예를 들면, 상태 머신이 상태 0에 있고, 제1 입력 문자가 ASCII 값 02인 것으로 가정하면, 상태 엔트리에 대한 절대 주소는 이 기본 주소(상태 0)와 인덱스/ASCII 문자(02)의 합/접합(concatenation)이 될 것이다. FSM은 CPU를 이용하여 메모리로부터 입력 문서의 제1 문자를 인출하는 것을 시작한다. 그리고 나서, CPU는 초기화/현재 상태 및 입력 문자에 대응하는 메모리 내의 상태 테이블에서 이 절대 주소를 구성하고, 상태 테이블로부터 상태 데이터를 인출한다. 리턴된 상태 데이터에 기반하여, CPU는 상이한 경우, 현재 상태를 새로운 값으로 갱신하고(그 문자가 관심 스트링의 첫 번째 문자에 대응한다고 지시함), 상태 데이터에서 지시된 다른 동작들(예를 들면, 하나의 문자가 특수 문자이거나 또는 현재 문자가 관심 스트링의 마지막 문자인 것으로 확인된 경우, 토큰 또는 인터럽트를 발행하는 것)을 수행한다.

전술한 프로세스가 반복되어, 관심 스트링의 후속 문자들이 확인됨에 따라 상태가 변경된다. 다시 말해서, 초기 문자가 관심 스트링의 초기 문자와 같이 관심 문자인 경우에는, FSM의 상태는 새로운 상태로(예를 들면, 초기 상태 0으로부터 상태 1로) 진행될 수 있다. 문자가 관심 문자가 아닌 경우, 상태 머신은 (일반적으로) 상태 테이블 주소로부터 리턴되는 상태 테이블 엔트리에 동일한 상태(예를 들면, 상태 0)를 지정하거나 또는 상태 갱신을 명령하지 않음으로써, 동일하게 유지된

다. 이것으로 제한되지는 않지만, 가능한 동작들은 인터럽트를 설정하는 것, 토큰을 저장하는 것 및 포인터를 갱신하는 것을 포함한다. 그리고 나서 프로세스를 다음 문자에 대해 반복한다. 관심 스트링이 후속되고, FSM이 상태 0 이외의 상태(또는 관심 스트링이 아직 확인되지 않았거나 현재 후속됨을 나타내는 그 밖의 상태)인 경우, 현재 스트링과 일치하지 않지만 다른 관심 스트링의 초기 문자가 확인될 수 있다. 이러한 경우, 상태 테이블 엔트리는, 새로운 스트링이 완전히 식별되거나 또는 관심 스트링이 아닌 것으로 확인될 때까지, 이전에 후속된 스트링 부분을 지시 및 식별하고, 가능한 신규 관심 문자를 뒤따르도록 하기 위한 적절한 동작을 지시할 수 있다. 다시 말해서, 관심 스트링이 내포(nest)될 수 있고, 상태 머신이 다른 관심 스트링 등에서 관심 문자를 검출할 수 있을 것이다. 이것은 XML™ 문서를 완전히 파싱하기 위해 CPU가 XML™ 문서의 부분들을 여러 번 트래버싱하도록 요구할 수 있다.

전체 XML™ 문서 또는 다른 언어의 문서는 전술한 방식으로 문자마다(character-by-character) 파싱된다. 가능한 목표 스트링이 인식되면, 관심 문자가 완전히 식별되거나, 또는 가능한 관심 스트링과 일치하지 않는 문자를 만날 때(예를 들면, 스트링이 완전히 매칭되거나 또는 문자가 목표 스트링에서 벗어날 때)까지, FSM은 여러 상태를 통해 문자별로 단계를 밟을 수 있다. 후자의 경우, 일반적으로 초기 상태 또는 다른 목표 스트링의 초기 문자 검출에 대응하는 상태로 리턴하는 것 이외에는 다른 동작이 수행되지 않는다. 전자의 경우에는, 입력 문서 내의 시작 주소 및 토큰 길이에 따라 메모리에 토큰이 저장된다. 파싱이 완료되면, 모든 객체가 식별될 것이고, 국부적인 또는 주어진 플랫폼에 따라 처리가 시작될 수 있다.

일반적으로 다수의 관심 스트링에 대해 검색이 수행되기 때문에, 상태 테이블은 주어진 상태로부터 많은 변화(transitions)를 제공할 수 있다. 이 접근방법은 편리하게 내포형(nested) 스트링을 수용하면서, 동시에 현재 문자가 많은 목표 스트링에 대해 분석될 수 있게 한다.

전술한 설명으로부터, XML™ 문서와 같은 문서의 파싱은 많은 반복과 이러한 반복을 위해 많은 메모리 액세스를 필요로 한다는 것을 알 수 있다. 따라서, 범용 CPU의 처리 시간이 불가피하게 상당한 시간이 된다. 다수의 스트링을 처리하는 보다 큰 복잡성은 큰 상태 테이블의 생성에 있고, 실시간 패킷 처리로부터 오프라인으로 핸들링된다. 그러나, 이것은 입력 문자 데이터를 인출하고, 상태 데이터를 인출하고, 문서 내의 각 문장에 대한 상태 주소 및 여러 포인터를 갱신하기 위해 많은 수의 CPU 사이클을 필요로 한다. 따라서, 이러한 XML™ 문서와 같은 문서의 파싱은 CPU 또는 플랫폼 상의 그 밖의 처리를 완전히 선점(pre-empt)하고, 요구된 처리를 실질적으로 지연시키는 것이 일반적이다.

종래 기술에서, 프로그래밍을 통해, 범용 하드웨어가 특수 목적의 하드웨어를 에뮬레이팅할 수 있고, 구조 및 프로그램이 서로 정확하게 대응하더라도, 특수 목적의 하드웨어를 관리 및 제어하는데 걸리는 부하가 적기 때문에, 특수 목적의 데이터 처리 하드웨어가 종종 프로그램된 범용 하드웨어보다 훨씬 빠르게 동작할 수 있다는 것을 알 수 있었다. 그럼에도 불구하고, 특히 처리 속도 이득이 최저(marginal)일 경우에, 임의 처리에 필요한 하드웨어 리소스는 특수 목적의 하드웨어에 대해 엄청나게 클 수 있다. 또한, 특수 목적의 하드웨어는 반드시 기능 제한을 가지며, 또한 임의 수의 문자들의 임의 조합을 검색하는 능력을 제공하는 것과 같은 어떠한 애플리케이션에 충분한 유연성을 제공하는 것도 금지될 수 있다. 따라서, 특수 목적의 하드웨어는 실질적인 하드웨어의 효율적 사용을 제공하면서 처리 속도에서 큰 이득을 제공해야 한다. 즉, 기능적 유연성 또는 프로그램 가능성을 증가시면서 동시에 수용하기 어려운 요건들이 요구된 처리 기능에 필요하다.

이와 관련하여, XML™ 문서와 같은 문서를 파싱하는데 필요한 처리 시간량과 상호접속성에 의해 시스템 보안의 문제도 제기되고 있다. 반면에, 비교적 높은 우선순위로 극도의 처리 시간량을 요구하는 프로세스는, 시스템 또는 그 노드 상의 서비스 거부(denial-of-service: DOS) 공격(attack)의 일부 특성과 유사하고, 이러한 공격에 사용될 수 있는 툴(tool)이 될 수 있다.

DOS 공격은 가용 리소스를 고의적으로 소비하고 결국은 과부하시키는 의도로, 사소하거나 또는 기형적인 서비스 요구를 시스템으로 빈번하게 나타낼 수 있다. 적절한 구조의 하드웨어 가속기는 가용 리소스의 과부하 가능성을 감소시키거나 없앨 수 있다. 또한, 과부하시, 시스템은 종종 보안의 취약성을 드러낸다. 따라서, 과부하를 없애는 것은 중요한 보안 사항이 된다.

또한, 상태 테이블은 시스템 성능의 중대한 손상없이 보안이 어렵거나 불가능한 기본 레벨에서 CPU 커맨드를 포함할 수 있기 때문에, 파싱이 완료되기 전에, 일부 처리가 시작되거나 일부 커맨드가 실행될 수 있다. 요약하면, XML™ 파싱과 같은 프로세스에 대한 처리 시간의 감축에 의해, 보안성의 타협에 대한 가능성은 반드시 감소되지만, 이러한 파싱을 위한 처리 시간을 현저하게 감소시키기 위한 어떠한 기술도 이용가능하지 않았다.

많은 보안 시스템은 시도된 보안 침해를 초기 단계에서 검출하는 능력에 의존하고, 보안 침해는 일단 개시되었다면, 빨리 또는 프로그램된 개입을 통하여 인터럽트하는 것이 어렵거나 불가능할 수 있다. 예를 들면, 성능이 뛰어난 보안 시스템이 제안되었는데, 미국특허출원번호 제09/973,769호 및 제09/973,776호에 개시되어 있으며, 이 두 건 모두 본 출원의 양수

인에게 양도되었다. 이 출원들은 가능한 공격 또는 침입이 검출된 노드가 구분되고 나서, 필요한 경우 네트워크에 재접속하기 전에 자동으로 복구되는 것에 의하여, 매우 고속으로 수행되는 2단계의 노드간 통신을 갖는 시스템을 개시하고 있다. 따라서, 파싱의 가속도는 잠재적인 공격에 대한 응답을 초기에 지원하며, 네트워크의 적절한 제어가 파싱의 사건으로서 개시될 수 있고, 이에 따라 파싱이 현저하게 가속될 수 있는 경우보다 이른 시간에 개시될 수 있기 때문에 상기 인용된 특허출원들에 기재된 시스템에 개시된 것과 같은 시스템에 있어서 특히 잇점이 있다. 검출 경보에 응답하여 적시에 개시되는 적절한 네트워크 제어는 침입 검출에 추가하여 침입 방지를 달성할 수 있다.

발명의 상세한 설명

본 발명은, 잠재적으로 실시간 침입 검출 및 방지 작용을 위한 네트워크 전송 패킷 속도를 수용하는 속도로, 네트워크에 접속된 컴퓨터 시스템에서의 가능한 침입, 공격 또는 다른 보안 침해(security breach)의 기호의 검출을 위해 문서의 파싱을 극도의 가속도로 제공하는 하드웨어 파서 가속기(hardware parser accelerator)를 제공한다.

본 발명의 이러한 목적 및 다른 목적을 달성하기 위해서, 가능하게는 문서 파서 내에 구현되는 침입 검출 시스템이 제공되며, 이 침입 검출 시스템은 문서의 다수의 바이트를 위한 문자 버퍼, 인터럽트 또는 예외, 및 상태 테이블로부터의 다음 상태 데이터 중 적어도 하나에 액세스하기 위해 문서의 바이트 및 상태에 따라 주소지정이 가능한 상태 테이블(state table), 다음 상태 데이터를 저장하기 위한 레지스터, 상태 메모리로의 다른 주소를 형성하기 위해 문서의 후속 바이트와 레지스터의 내용을 결합하기 위한 가산기(adder), 및 인터럽트 또는 예외를 호스트 CPU와 통신하기 위한 버스를 포함한다.

도면의 간단한 설명

도1은 문서의 파싱에 이용되는 상태 테이블의 일부를 나타낸 도면.

도2a는 동시출원된 관련 가특허출원에 따른 파서 가속기의 고레벨 개략도.

도2b는 본 발명에 따른 파서 가속기의 고레벨 개략도.

도2c는 호스트 프로세서 및 메인 메모리를 구비한 본 발명의 실시예를 도시한 도면.

도3은 도2a에 도시된 바와 같은 바람직한 문자 팔레트 포맷(character palette format)을 도시한 도면.

도4a 및 도4b는 도2a에 도시된 바와 같은 본 발명의 바람직한 형태에서 상태 테이블 포맷 및 그것과 함께 이용되는 상태 테이블 제어 레지스터를 도시한 도면.

도5a는 도2a에 도시된 바와 같은 바람직한 다음 상태 팔레트 포맷을 도시한 도면.

도5b는 도2b에 도시된 바와 같은 본 발명과 이용하기 위한 바람직한 상태 테이블 엔트리 포맷을 도시한 도면.

도6은 도5에 도시된 바와 같은 바람직한 토큰 포맷(token format)을 도시한 도면.

실시예

이제, 도면들, 보다 구체적으로는 도1을 참조하면, 본 발명을 이해하는데 유용한 상태 테이블의 일부를 나타낸 도면이 도시되어 있다. 도1에 도시된 상태 테이블은 잠재적으로 XML™ 문서를 파싱하는데 유용한 상태 테이블의 매우 작은 일부일 뿐이고, 사실상 예시를 위한 것으로 의도된다는 것을 이해해야 한다. 전체 상태 테이블은 물리적으로 존재하지 않지만, 적어도 본 발명 및 도1에 도시된 형태는 공지된 소프트웨어 파서의 동작의 이해를 돕는데 이용될 수도 있고, 도1의 어떠한 부분도 본 발명에 관해서는 종래기술인 것으로 인정되지 않는다.

본 발명에 따른 가속기를 이용하여 처리될 수 있는 논리 데이터 시퀀스의 하나의 타입의 예로서 본 명세서에서는 XML™ 문서가 이용된다는 것에 주목해야 한다. 다른 논리 데이터 시퀀스 역시 공유된 서버 컴퓨터들에 의한 실행용으로 의도된 사용자 단말기 커맨드 스트링(command strings)과 같은 네트워크 데이터 패킷 내용으로부터 구성될 수 있다. 이러한 커맨드 스트링은 유해(malicious) 사용자에 의해 빈번하게 발생되어, 장기간 침입 시도의 부분으로서 공유된 서버 컴퓨터들로 전송된다. 본 발명에 따른 가속기는 많은 이러한 논리 데이터 시퀀스를 처리하는데 적절하다.

도1에 예시된 상태 테이블의 부분내의 많은 엔트리(entry)가 중복적이라는 것은 유용한 관찰이 될 것이고, 도1로 나타내어진 상태 테이블의 완전함(entirety)을 수용하기 위한 하드웨어가 요구되지 않는다는 것은 본 발명의 평가(appreciation)에 중요하다. 반대로, 본 발명이 가능하게는 전용 프로세서를 이용하여, 소프트웨어로 구현될 수 있지만, 본 발명에 따른 하드웨어 요건이 충분히 제한되고, 소프트웨어에 의한 파싱 처리 시간을 증가시키는 불이익(penalty)이 하드웨어의 임의의 가능한 경제성(economy)에 의해 정당화되지 않는다.

그러나, 침입 검출 시스템은 어떠한 타입의 디지털 파일에도 적용가능하도록 의도되고, 보안 공격이 일반적으로 행해지는 네트워크를 통한 실시간 디지털 전송을 수용할 수 있는 패킷 전송 속도로 또는 그 속도를 초과하여 특정 애플리케이션 또는 데이터 구조를 나타내는데 이용될 수 있는 텍스트 파일 또는 특정 언어에 한정되지 않는다. 따라서, 본 발명은 단지 침입 검출을 제공하기 위한 구성으로서 구현될 수 있고, 이 경우, 최저 비용으로 실질적으로 최적의 성능이 기대된다. 그러나, 신호 전송 속도로 침입 검출을 수행하는 목적은, 가능하게는 하기에 설명되는 바와 같은 대안적인 상태 테이블 메모리 구성에 의해 증대되는 다른 가속도를 제공하기 위해 어떤 동작이 생략된 파서 가속기의 특정 동작 모드로서 달성될 수 있다. 따라서, 그 관계가 실시간 고속 침입 검출을 위해 의도된 것으로서 기능하기 위해 본 발명에 필요한 것보다 복잡하더라도, 본 발명은 완전성을 위하여 그리고 본 발명에 의해 제공되는 장점의 범위의 보다 완전한 이해를 전달하기 위해 파서 가속기와 관련하여 설명될 것이다.

도1에서, 상태 테이블은 임의의 개수의 로우(row)로 분할되고, 각각의 로우는 상태에 대응하는 기준 주소를 갖는다. 기준 주소의 로우는 파싱될 문서내의 문자를 나타내는데 이용될 수 있는 코드의 개수에 대응하는 다수의 칼럼(column)으로 분할되며, 본 실시예에서는, 상태 테이블로의 인덱스로서 이용되는 문자를 위한 기본적인 8비트 바이트에 대응하는 256개의 칼럼으로 분할된다.

특히, 도1에 예시된 상태 테이블의 어느 정도로 작은 부분이 많은 단어의 검출을 지원하는지에 대한 이해를 전달함에 있어서, 도시된 상태 테이블 엔트리의 여러 양태를 주목하는 것이 도움이 될 것이다.

1. 도시된 상태 테이블에서, 상태 0에 대한 로우에 있는 단지 2개의 엔트리는 테스트되는 문자가 임의의 관심 스트링의 초기 문자와 부합하지 않을 때 초기 상태를 유지하는 "상태 0에 머무르시오" 이외의 엔트리를 포함한다. 상태 1로의 진행을 제공하는 단일 엔트리는 모든 관심 스트링이 동일 문자로 시작하는 특별한 경우에 대응한다. 다른 상태로의 진행을 제공하는 임의의 다른 문자는, 반드시 그러한 것은 아니지만 일반적으로, 상태 1 이외의 상태로 진행하지만, 다른 문자를 통해 도달될 수 있는 동일 상태에 대한 다른 기준이 예를 들어, 내포형 스트링 검출에 유용할 수 있다. {상태 0, FD}에 예시된 "상태 0에 머무르시오"라는 커맨드(예를 들면, "특별한 인터럽트")의 포함은 단일 특수 문자에 대해 검출 및 동작하는데 이용될 것이다.

2. 상태 0을 넘는 상태들에서, "상태 n에 머무르시오"라는 엔트리는 그 상태가, 예를 들면, 흔히 마주치게 되는 바와 같은, 커맨드의 수치적 인수(numerical arguments)에서 마주치게 되는 것과 같은 하나 또는 그 이상의 문자들의 잠재적으로 긴 런(run)을 통해 유지되게 한다. 하기에 상세하게 논의되는 바와 같이, 본 발명은 이러한 타입의 문자 스트링의 특별한 조정을 제공하여 개선된 가속도를 제공한다.

3. 상태 0을 넘는 상태들에서, "상태 0으로 가시오"라는 엔트리는, 얼마나 많은 부합하는 문자들이 이전에 검출되었는지에 상관없이, 임의의 관심 스트링으로부터 그 스트링을 구별해내는 문자의 검출을 나타내고, 파싱 처리를 초기/디폴트 상태로 리턴(return)하여 다른 관심 스트링의 검색을 시작한다. (이러한 이유로, "상태 0으로 가시오"라는 엔트리는 일반적으로 상태 테이블에서 월등히 가장 빈번하거나 다수인 엔트리일 것이다.) 상태 0으로의 리턴은 파싱 동작이 구별 문자가 검출되었을 때에 뒤따르게 되는 스트링을 시작하는 문자에 뒤따르는 문서내의 문자로 리턴하도록 요구할 수 있다.

4. "상태 0으로 가시오"라는 커맨드를 포함하는 엔트리는 완전한 관심 스트링의 검출의 완료를 나타낸다. 일반적으로, 이 커맨드는 그후 그 스트링을 객체로서 처리되도록 하는 토큰(token)(이 토큰의 주소 및 길이를 구비함)을 저장할 것이다. 그러나, "상태 n으로 가시오"라는 커맨드는 관심 스트링에 잠재적으로 부합할 수 있는 스트링을 계속하여 뒤따르는 동안 중간 지점에서의 동작의 개시를 제공한다.

5. 2개의 관심 스트링(예를 들면, (n-1)개의 동일한 초기 문자를 갖지만 서로 다른 n번째 문자를 갖거나, 또는 서로 다른 초기 문자를 갖는 스트링) 사이에서 검색이 갈라지는 임의의 지점에서의 모호함을 피하기 위해, 일반적으로 {상태 1, 01} 및 {상태 1, FD}에 예시된 바와 같이, 서로 다른(예를 들면, 비-연속적인) 상태로 진행할 필요가 있다. 임의의 길이 n의 스

트링의 완전한 식별은 특수 문자의 포함된 스트링 및 공통 초기 문자를 갖는 관심 스트링의 특별한 상황을 제외한 (n-1) 상태를 요구할 것이다. 이러한 이유로, 상태 테이블의 상태 및 로우의 개수는 관심 스트링의 개수가 비교적 많지 않더라도, 보통 매우 커야 한다.

7. 이전 단락과 반대로, 대부분의 상태는 하나 또는 그 이상의 고유 엔트리 및 디폴트 "상태 0으로 가시오"로 완전히 특징지어질 수 있다. 도1의 상태 테이블의 이러한 특징은 관심 스트링의 일반적인 경우에 대한 파싱 처리의 높은 정도의 하드웨어 경제성 및 상당한 가속도를 생성하기 위해 본 발명에서 이용된다.

상기에 언급된 바와 같이, 통상적으로 수행되는 바와 같은 파싱 동작에서는, 도1에 상태 0으로 도시된 소정의 디폴트/초기 상태에서 시스템을 시작하고, 그 다음, 부합하는 문자가 처리의 반복시에 발견됨에 따라 보다 높은 번호가 붙은 상태로 진행한다. 관심 스트링이 완전하게 식별되었을 때나 잠재적으로 부합(match)인 스트링에서의 중간 위치에서 특별한 동작이 특정될 때, 토큰의 저장 또는 인터럽트의 발행과 같은 동작이 수행된다. 그러나, 문서의 각각의 문자에 대한 각각의 반복에서, 문자는 CPU 메모리로부터 인출되어야 하고, 상태 테이블 엔트리는 (다시 CPU 메모리로부터) 인출되어야 하고, (예를 들어, 문서의 문자 및 상태 테이블내의 기준 주소에 대한) 다양한 포인터(pointer) 및 (예를 들어, 초기 부합된 문자 주소 및 스트링의 축적된 길이에 대한) 레지스터는 순차적인 동작으로 갱신되어야 한다. 따라서, 파싱 동작은 많은 처리 시간을 소모할 수 있음을 쉽게 알 수 있다.

본 발명에 따른 파서 가속기(100)의 고레벨의 개략적인 블록도가 도2a에 예시되어 있다. 이 기술분야의 당업자이면 알 수 있는 바와 같이, 도2a는 파싱을 수행하기 위해 본 발명에 따라 수행되는 단계들을 예시하는 흐름도로서 이해될 수 있다. 도3, 도4a, 도4b, 도5a 및 도6과 관련하여 하기에 보다 상세하게 논의되는 바와 같이, 본 발명은 비록 시간적으로 약간 빗나가더라도 본질적으로 병렬로 동작하는 다수의 하드웨어 파이프라인이 전개되도록 상태 테이블을 나타냄에 있어서 어떤 하드웨어 경제성을 활용한다. 따라서, 포인터 및 레지스터의 갱신은 다른 동작과 실질적으로 병렬로 그리고 동시에 수행될 수 있고, 메모리 액세스를 위해 요구되는 시간은, 병렬로 동작되는 보다 빠른 액세스 하드웨어와 상태 테이블 및 문서와 관련하여 CPU 메모리로부터의 사전인출(prefetching) 양쪽 모두를 통해서 많이 감소된다.

전반적인 개요로서, XML™ 문서와 같은 문서는 레지스터(112, 114)에 의해 인덱싱되는 DRAM(120)에 외부로 저장되어, 바람직하게는, 32 비트 단어에 의해 파이프라인에 대한 멀티플렉서의 역할을 하는 입력 버퍼(130)로 전송된다. 각각의 파이프라인은 문자 팔레트(140), 상태 테이블(160) 및 다음 상태 팔레트(170)의 사본(copy)을 포함하고, 각각은 상태 테이블의 부분의 압축된 형태를 수용한다. 다음 상태 팔레트(170)의 출력은 상태 테이블(160)내의 엔트리로의 주소의 다음 상태 주소 부분과 저장될 토큰값(token value)을 양쪽 모두 포함한다. 문자 팔레트(140) 및 다음 상태 팔레트(170)에서의 동작은 서로 병렬로 뿐만 아니라 (캐시로서 구현될 수 있는) 상태 테이블(160)을 형성하는 고속의 외부 DRAM으로의 간단한 메모리 액세스와 병렬로 수행될 수 있는 고속의 내부 SRAM으로의 간단한 메모리 액세스이다. 따라서, (일단 개시되면, 문서 데이터를 리프레시하고 토큰을 저장하도록 단지 예비의 CPU 메모리 동작 호출에 의해 자동으로 기능할 수 있는) 이들 하드웨어 요소를 처음에 제어하는 CPU의 비교적 소수의 클록 사이클만이 문서의 각각의 문자의 평가를 위해 요구된다. 기본적인 가속도 이득은 CPU에서의 문자 당 모든 메모리 동작 지속시간의 합계 더하기 고속 SRAM 또는 DRAM에서의 단일 자동 수행 메모리 동작의 지속시간에 대한 CPU 오버헤드의 감소이다.

본 명세서에서 "외부"로 칭해지는 메모리 구조는 요구되는 저장량 및 하드웨어 파서 가속기 및/또는 호스트 CPU로부터의 액세스에 비추어 볼 때 현재 발명자에 의해 바람직한 메모리(120, 140)의 구성을 나타내는 것으로 의도된다는 것을 이해해야 한다. 다시 말하면, 메모리의 공유 또는 적어도 호스트 CPU 뿐만 아니라 하드웨어 가속기에 의한 메모리에의 액세스를 용이하게 하기 위해 본 발명에 따른 파서 가속기의 아키텍처를 제공하는 것은, 토큰의 조정 및 어떤 다른 동작에 장점적일 수 있다. 어떠한 다른 의도된 함축적 의미도, 그리고 동기식 DRAM(SDRAM)과 같은 매우 다양한 하드웨어 대안들도, 이러한 논의에 비추어 볼 때 이 기술분야의 당업자에 의해 적절하다고 인식되지 않을 것이다.

이제, 도3 내지 도6을 참조하면, 문자 팔레트(140), 상태 테이블(160), 다음 상태 팔레트(170) 및 다음 상태 및 토큰의 포맷은 도2a의 바람직한 실시예를 지원하는 하드웨어 경제성의 예시로서 논의될 것이다. 다른 기술/포맷이 또한 이용될 수 있고, 예시된 포맷은 비록 현재 바람직하더라도 예시로서 이해해야 한다.

도3은 관심 스트링에 포함되거나 포함될 수 있는 문자에 대응하는 문자 팔레트의 바람직한 형태를 예시한다. 이 포맷은 바람직하게는 도1의 상태 테이블의 칼럼의 개수에 대응하는, 0 내지 255의 번호가 붙은 엔트리를 제공한다. ("팔레트"라는 용어는, 총체적으로 색역(gamut)이라고 칭해지는 지원되는 각각의 컬러에 대한 데이터를 포함하는 "컬러 팔레트"라는 용어와 동일한 의미로 이용된다. 팔레트의 이용은 상태 테이블의 엔트리/칼럼을 줄인다.) 예를 들어, 상태의 변화를 야기하지 않는 "널 문자(null character)"라고 칭해지는 문자는 많은 이러한 칼럼보다는 오히려 상태 테이블의 하나의 칼럼에 표현될 수 있다. 파싱을 위한 처리를 실질적으로 가속할 수 있는 144에서 출력된 널 문자를 테스트하는 것은 바람직한데, 그 이유

는 그것이 상태 테이블 액세스를 위한 추가의 메모리 동작 없이 다음 문자의 즉시 처리를 가능하게 하기 때문이다. 이 포맷은 예를 들어, (도2에서 메모리 플레인(plane)을 겹치는 것에 의해 개략적으로 예시된) 특정 문자 팔레트를 가리키는 기준 주소 레지스터(142)내의 데이터에 의해서와 같이 구성된 단일 레지스터 또는 메모리 위치에 의해 조정될 수 있다. 문서(예를 들면, XML™ 문서)로부터의 현재의 8비트 문자 즉, 외부 DRAM(120)으로부터 4 바이트 단어로서 수신되어 입력 버퍼(130)로부터 제공된 4개 중 하나는 그 다음에 상태 메모리로의 인덱스 또는 부분 포인터로서 주소를 출력하는 문자 팔레트 내의 엔트리를 주소지정한다. 따라서, 이러한 포맷의 팔레트에 제공함으로써, 도1의 기능성의 일부가 비교적 한정된 용량의 단일 레지스터의 형태로 제공될 수 있고, 이로써 그것들 다수가 상당한 하드웨어 경제성을 유지하고 상태 테이블(160) 내의 다른 것을 지원하면서 병렬로 형성 및 동작될 수 있게 한다.

도4a는 문자 팔레트와 유사하게 (예를 들어, 실질적으로 레지스터로서) 구성 또는 배열되는 바람직한 상태 테이블 포맷을 도시한다. 도3의 문자 팔레트와의 주된 차이점은, 레지스터의 길이가 원하는 문자에 대한 응답의 개수 및 관심 스트링의 개수와 길이에 의존한다는 점이다. 따라서, 경제적으로 제공될 수 있는 내부 메모리의 양이 특정 경우에 불충분하다면, CPU내의 이러한 메모리 또는 다른 외부 DRAM(가능하게는 내부 또는 외부 캐시를 구비)을 구현할 가능성을 제공하는 것이 바람직하다고 고려된다. 그럼에도 불구하고, 도1의 상태 테이블내의 매우 중복적인 엔트리가 단일 엔트리로 감소될 수 있기 때문에 상당한 하드웨어 경제성이 제공된다는 것은 자명하며, 단일 엔트리의 주소는 도3의 문자 팔레트에 따라 상기에 설명된 바와 같이 제공된 데이터에 의해 수용된다. 상태 테이블(160)의 출력은 바람직하게는, 1, 2 또는 4 비트이지만, 32 비트 만큼 많은 비트를 제공하면, 도4b와 관련하여 하기에 논의되는 바와 같이, 증가된 유연성을 제공할 수 있다. 임의의 경우, 상태 테이블의 출력은 다음 상태 팔레트(170)로의 주소 또는 포인터를 제공한다.

이제, 이 후반의 측면에 있어서의 본 발명의 완전한 특징으로서, 도4b를 참조하면, 본 발명의 바람직한 구현 특징은, 특히, 상태 테이블(160)의 32비트 출력이 제공될 경우, 추가적인 상당한 하드웨어 경제성을 가능하게 하는 상태 테이블 제어 레지스터(162)를 포함한다. 본래, 상태 테이블 제어 레지스터는 가변 길이 단어가 상태 테이블에 저장 및 상태 테이블로부터 판독될 수 있게 함으로써 상태 테이블 정보의 압축을 제공한다.

보다 구체적으로, 상태 테이블 제어 레지스터(162)는 도4a의 상태 테이블(160)내의 각각의 엔트리의 길이를 저장 및 제공한다. 도1의 어떤 상태 테이블 엔트리가 매우 중복적이기 때문에(예를 들면, "상태 0으로 가시오", "상태 n에 머무르시오"), 이들 엔트리는 상태 테이블(160)내의 단일 엔트리로 나타내어질 수 있을 뿐만 아니라, 적어도 도1에서보다 훨씬 소수의 엔트리도 가능하게는, 어떤 상태 테이블에서 편리하다고 판명될 수 있는 바와 같이, 대부분 또는 모든 중복적인 엔트리가 상태 테이블에 포함되어 있더라도 상당한 하드웨어 경제성을 산출하는 하나만큼 소수인 보다 소수의 비트로 나타내어질 수 있다. 이러한 감소의 원리는 소위 엔트로피 코딩(entropy coding)과 유사한 것으로 이 기술분야의 당업자에 의해 인식될 것이다.

이제, 도5a를 참조하면, 다음 상태 팔레트(170)의 바람직한 포맷이 논의될 것이다. 다음 상태 팔레트(170)는 상기에 논의된 문자 팔레트(140)와 매우 동일한 방식으로 구현되는 것이 바람직하다. 그러나, 상태 메모리(160)에서 처럼, 요구될 수 있는 엔트리의 개수는 선험적으로(*a priori*) 알려져 있지 않고, 개별 엔트리의 길이는 바람직하게는 훨씬 길다(예를 들면, 2개의 32비트 단어). 한편으로, 다음 상태 팔레트(170)는 (예를 들어, 다음 상태 팔레트 기준 주소 레지스터(172)를 이용하여) 캐시로서 동작될 수 있는데, 그 이유는 단지 비교적 작고 예측가능한 범위의 주소가 임의의 소정 시간에 포함될 필요가 있기 때문이다. 또한, 상태 테이블(160)의 32비트 출력이 제공된다면, 그 데이터의 일부가 다음 상태 팔레트(170)의 엔트리내의 데이터를 보충하는데 이용될 수 있고, 이로써 후반부에 보다 짧은 엔트리를 허용할 수 있거나, 띠줄(175)로 표시된 바와 같이 다음 상태 팔레트를 전체적으로 바이패스(bypass)할 수 있다.

도5a에 도시된 바와 같이, 다음 상태 팔레트(170)로부터의 하위 주소 32 비트 단어 출력은 세이브(save)될 토큰이다. 이 토큰은 16비트의 토큰값, 8비트의 토큰 플래그 및 8비트의 제어 플래그로서 형성되는 것이 바람직하고, 이들 토큰값과 토큰 플래그 양쪽 모두는 성공적인 문자 비교를 카운트함으로써 축적된 길이와 함께, 스트링의 시작부에 대한 포인터(192)에 의해 제공되는 주소로 토큰 버퍼(190)에 저장된다. 제어 플래그는 호스트 CPU에 인터럽트를 설정하거나 파서 가속기에서의 처리를 제어한다. 이들 후반부의 제어 플래그 중 하나는 상기에 언급된 바와 같이, 관심 스트링에서 발생할 수 있는 임의의 길이의 동일하거나 관련된 문자들의 스트링과 같은 상태 0 이외의 상태에서 상태의 변경을 야기하지 않는 문자에 대한 스킵 인에이블 기능(skip enable function)을 설정하는데 이용되는 것이 바람직하다. 이러한 경우에, 다음 상태 테이블 엔트리는 SRAM/SDRAM으로부터 그것을 인출하지 않고 재사용될 수 있다. 입력 버퍼 주소(112)는 추가적인 처리 없이도 증분되고, 이로써, 문자의 어떤 스트링에 대하여 파싱의 상당한 추가 가속도를 허용한다. 두번째 32 비트 단어는 다음 문자를 위한 상태 테이블로의 포인터를 형성하기 위해 문자 팔레트로부터의 인덱스 출력과 연결되도록 레지스터(180) 및 가산기(150)에 공급되는 주소 오프셋이다. 상태 0에 대응하는 초기 주소는 레지스터(182)에 의해 공급된다.

그러므로, 문자 팔레트, 단축된 형태의 상태 메모리 및 다음 상태 메모리의 사용은 통상적인 상태 메모리 동작의 기능을 별개의 스테이지(stage)로 나눈다는 것을 알 수 있으며, 별개의 스테이지 각각은 다른 동작 및 토큰의 저장과 차례로 그리고 병렬로 문서의 개별 문자에 대해 동작하는 병렬 파이프라인을 형성하기 위해 중복될 수 있는 비교적 작은 고속 메모리를 이용하여 매우 신속하게 수행될 수 있다. 따라서, 파싱 처리는 다른 문자의 처리가 개시될 수 있기 전에 이들 기능 전체를 순차적으로 수행해야 하는 전용 프로세서에 비해서도 크게 가속될 수 있다.

요약하면, 가속기는, 문자 데이터(종종 네트워크의 전송을 의미하는 패킷 데이터라고 칭해짐) 및 상태 테이블이 위치하는 호스트 CPU의 프로그램 메모리와의 액세스를 갖는다. 가속기(100)는 메모리 맵 레지스터(memory-mapped register)를 통한 메인 CPU의 제어하에 있다. 가속기는 침입 검출과 관련하여 패턴 부합 경보(pattern matching alert), 침입 이벤트 경보(intrusion event alert) 등이라고 칭해질 수 있는 예외, 경보 및 종료를 나타내기 위해 메인 CPU를 인터럽트할 수 있다. 파싱이 개시될 때, 포인터(112, 114)는 분석될 입력 버퍼(130) 데이터의 시작부 및 끝부에 설정되고, (기준 주소(182) 및 다른 제어 정보(예를 들면, 142)로 표시된 바와 같은) 이용될 상태 테이블은 가속기 내에서 설정된다.

가속기의 동작을 개시하기 위해, CPU는 가속기에 커맨드를 발행하고, CPU는 응답하여, CPU 프로그램 메모리(예를 들면, 120 또는 캐시)로부터의 데이터의 제1 32 비트 단어를 인출하고, 그것을 제1 바이트/아스키(ASCII) 문자가 선택되는 입력 버퍼(130)로 위치결정한다. 가속기는 입력 문자(즉, 도4a는 도1의 전체 상태 테이블의 단일 문자 또는 단일 칼럼에 대응함) 및 현재 상태에 대응하는 상태 정보를 인출한다. 상태 정보는 다음 상태 주소 및, CPU를 인터럽트하거나 처리를 종료하는 것과 같은 수행될 임의의 특별한 작용을 포함한다.

다음으로, 가속기는 입력 버퍼(130)로부터 분석될 다음 바이트를 선택하고, 가산기(150)에 이미 이용가능하게 될 새로운 상태 정보를 이용하여 그 처리를 반복한다. 동작 또는 토큰 정보 저장은 동시에 수행될 수 있다. 이것은 입력 단어의 모든 4개의 문자가 분석될 때까지 계속한다. 그 다음(또는 사전인출에 의한 네번째 문자의 분석과 동시에), 버퍼들(112, 114)이 비교되어, 문서 버퍼(120)의 끝에 도달하였는지 여부를 결정하고, 만약 도달하였다면, CPU에 인터럽트가 전송된다. 만약 도달하지 않았다면, 새로운 단어가 인출되고, 버퍼(112)가 갱신되며, 처리가 반복된다.

포인터와 카운터가 전용 하드웨어에 구현되어 있기 때문에, 이들 포인터와 카운터는 소프트웨어로 구현된 경우에 요구되는 바와 같이 연속적이라기 보다는, 병렬로 갱신될 수 있다. 이것은 데이터 바이트 분석 시간을, 로컬 입력 버퍼로부터 문자를 인출하고, 고속 로컬 문자 팔레트 메모리로부터 상태 테이블 주소를 발생시키고, 메모리로부터 대응하는 상태 테이블 엔트리를 인출하고, 다시 로컬 고속 메모리로부터, 다음 상태 정보를 인출하는데 걸리는 시간으로 감소시킨다. 이들 동작 중 일부는 별개의 병렬 파이프라인에서 동시에 수행될 수 있고, (다음 상태 팔레트를 통해 부분적으로 또는 전체적으로 제공된) 상태 테이블 정보에 특정된 다른 동작들은 추가의 문자의 분석이 계속되는 동안에 실행될 수 있다.

따라서, 본 발명이 작고 경제적인 양의 전용 하드웨어를 통해 파싱 처리의 상당한 가속도를 제공한다는 것을 알 수 있음은 자명하다. 파서 가속기가 CPU를 인터럽트할 수 있는 동안, 처리 동작은 파서 가속기에 대한 초기 커맨드 후에 CPU로부터 완전히 제거된다. 그러나, 다른 파싱 동작과 동시에 수행되더라도 토큰의 처리를 위해 상당한 시간이 요구되기 때문에, 상기에 설명된 바와 같이 제공되는 가속도는 특히, 보안이 어렵거나 불가능한 동작들이 파싱 처리 동안에 커맨드의 발행에 의해 개시될 수 있다는 사실에 비추어 볼 때, 가능한 침입 또는 보안 침해의 검출을 위한 최적이 아니다.

이제, 도2b를 참조하면, 상기에 설명된 바와 같은 도2a의 구성의 파싱 처리 속도보다 높게 파싱 처리 속도를 현저하게 개선시키지만, 가능한 침입 또는 보안 침해의 기호의 검출을 위한 제한된 목적에 대해서는 완전히 호환가능한 하드웨어 파서 가속기에 대한 구성이 도시되어 있다. 도2b를 도2a와 비교함으로써, 도2b의 구성은 주로 도2a의 구성의 서브-세트(sub-set)이고, 침입의 기호일 수 있는 모든 스트링을 검색할 수 있는 동일한 장점(예를 들면, 하나 이상의 표현 또는 그 부분과 부합시키는 것에 의해, 상태 테이블의 인코딩된 완전한 표현을 부합시키기 전에 패턴 부합 경보가 CPU에 발행될 수 있고, 이에 따라 응답 속도를 증가시키는 것)을 제공하지만, 이와 함께, 시스템을 보호하기 위한 인터럽트 또는 예외의 발행만이 처리 결과로서 발행될 필요가 있기 때문에 토큰 처리의 생략을 통해 추가의 가속도를 제공한다는 것이 이 기술분야의 당업자에 의해 인식될 것이다. 문서의 완전한 파싱을 위한 상기에 설명된 바와 같은 처리는 가능한 보안 침해 기호의 포함이 완료되도록 문서가 차폐(screen)된 후에 수행될 수 있다. 이 차폐 처리 동안에 토큰 처리가 생략되기 때문에, 메모리 액세스는 수적으로 훨씬 감소된다. 즉, 본 발명에 따른 침입 검출 가속도에 대하여(그리고 상기에 설명된 하드웨어 파서 가속기와 비교하여), 토큰 처리 및 문자 팔레트의 이용이 생략됨으로써, 보다 낮은 메모리 리소스 요건 및 약간의 처리 시간 감소의 결과를 야기한다. 그러나, 이러한 처리 중 많은 것이 병렬로 행해지기 때문에, 처리 속도 증가는 일반적으로 약 25% 또는 다소 작아짐으로써, 부분적으로, 메모리 속도, 로직 속도 등에 의하여 다양한 리소스에 이용된 특정 디바이스에 의존한다. 그러나, 어쩌면 속도 보다 중요한 것은, 가능한 보안 침해의 임의의 기호가 검출되어, 대응하는 커맨드 전에 발행되는 구제(remedial) 인터럽트 또는 예외가 공격의 부분으로서 CPU에 의해 실행될 수 있다는 사실이다.

도2b의 구성의 모든 기능적인 요소들은 도2a에 존재하고, 동일한 참조 부호가 대응하는 요소에 이용된다. 따라서, 본 발명에 따른 침입 검출 파서 가속기(200)는 상기에 설명된 파서 가속기와 완전히 혼환가능하고, 그 구성의 변경은 침입 검출 처리가 본질적으로 도2a의 파서 가속기의 특별한 동작 모드가 되도록 하는 프로그래밍을 통해 대부분 달성될 수 있다는 것은 명백하다.

특정하게는, 주소 레지스터(112, 114), 가산기(150) 및 상태 테이블 기준 주소 레지스터(182)와 함께, 입력 버퍼(120) 및 입력 단어 버퍼(130)는 전술된 대응하는 요소와 동일하고, 상태 테이블(160)에 액세스하기 위해 동일한 방식으로 기능한다. 대체로, 문자 팔레트 및 다음 상태 팔레트 메모리의 생략과 상태 테이블내의 데이터 및 그 내부 포맷에 있어서 차이점이 존재한다. 상태 테이블은 본질적으로 도2a의 실시예에서와 동일한 폭의 256개의 문자로 되어 있다. 검색이 수행되는 기호는 임의의 개수의 문자로 된 간단한 스트링보다 복잡할 수 있음을 이해해야 한다. Vikram Vaswani 등에 의한 "So What's a $#!%% Regular Expression Anyway?!"에서 논의되는 바와 같이, 기호는 스트링보다 복잡할 수 있는 "정규 표현(regular expressions)"으로서 보다 일반적으로 설명된다. (개발자가 영향을 미친(developer shed)) 저작권 Melonfire 2000-2002는 본 명세서에 참조로서 완전히 통합되어 있다. 따라서, 정규 표현에 대응하는 상태 테이블은 간단한 스트링에 대해서보다 훨씬 더 클 수 있다. 또한, 매우 큰 상태 테이블을 야기할 수 있는 동일한 상태 테이블을 동시에 이용하기 위하여 다중 정규 표현이 검색될 수 있다. 그러나, 사실상, 64가지 상태가 일반적으로는 충분하다. 그러나, 그렇지 않은 경우, 8비트를 넘도록 상태 테이블 엔트리를 확장하는 것이 필요하다. 따라서, 전술된 바와 같이 제공된 극도의 압축은 일반적으로 필요하지 않고, 전술된 바와 같은 상태 테이블(160)의 하드웨어는 요구되는 메모리 액세스의 수를 줄이기 위해 공격 기호의 검출에 요구되는 상태 테이블의 전체가 아닌 상당 부분을 제공할 수 있다.

도2a의 실시예에서와 같이, 상태 테이블내의 데이터의 포맷은 바이트로 표현될 수 있는 문자의 개수를 수용하기 위해 n=256 엔트리를 포함하는 것이 바람직하다. 그러나, 도2b의 실시예의 경우에, 상태 테이블로의 액세스가 단어 버퍼(130)에 버퍼링된 문자 비트로부터 직접 수행된다. 도5b에 도시된 바와 같이, 상태 테이블내의 개별 엔트리의 내용은, 관심 스트링을 정의하고 스트링이 뒤따르도록 허용하는 로딩될 상태 테이블의 다음 상태 또는 로우, 및/또는 (반드시 이러한 기호를 구성할 수 있는 스트링의 마지막 문자일 필요는 없는) 기호로서의 스트링의 인식을 지원하는 스트링의 문자에 대하여 발행될 인터럽트 또는 예외를 위한 플래그 또는 다른 코드를 단지 포함해야 한다. 다음 상태는 보통 8비트보다 작게 표현될 수 있고, 관심 스트링의 검출시 생성될 예외의 인터럽트는 단일 비트로 표현될 수 있다.

따라서, 문자는 순차적으로 테스트되고, 관심 스트링의 첫번째 문자인 소정 문자와 마주치게 될 때까지 레지스터(112, 114) 이외의 임의의 레지스터의 어떠한 갱신도 요구되지 않는다. 즉, 이러한 검출까지, 상태 조차도 변경되지 않고, 다음 상태는 레지스터(180)에서 갱신되지 않는다. 따라서, 문서는 극도의 속도로 초기 문자에 대해 차폐될 수 있다. 관심 스트링의 초기 문자가 마주치게 될 때, 다음 상태 데이터는 상태 테이블로부터 판독되고, 레지스터(180)는 갱신되고, 새로운 상태 테이블 데이터는 이미 존재하지 않는 경우 상태 메모리로 로딩되고, 다음 문자는 동일한 방식으로 처리된다. 상태 테이블 메모리는 전술된 XML™ 파서에 대하여 훨씬 작다. 이것은 상태 테이블 메모리가 도2a 또는 도2b의 다른 로직 및 요소를 갖는 칩(chip)과 함께 보드(board)에 구현되도록 허용함으로써, 가능하게는 외부 메모리를 이용한 설계에 요구되는 시간의 25%만큼 작게 처리 사이클 시간을 감소시킨다. 그러나, 가속기가 간단하게 XML™ 파서의 특별한 동작 모드로서 설계된다면, 상태 테이블은 외부 메모리에 구현될 것이고, 이러한 속도 증가는 제공되지 않을 것이다. 따라서, 이러한 경우에 상태 테이블 사이즈에 따라 교대로 이용되도록 보드와 외부 메모리 양쪽 모두에 제공하는 것이 비용 효율적일 수 있다. 그러므로, 공격 기호에 대해 문서를 차폐하기 위해 각각의 문자에는 비교적 작은 클록 사이클만이 요구된다. 관심 스트링을 식별하기 위해 충분한 개수의 문자가 처리되었을 때, 상태 테이블로부터 판독된 데이터는 시스템 보호를 위한 커맨드로서 호스트 CPU에 발행되는 인터럽트 또는 예외를 포함할 것이다.

도2a 또는 도2b에 도시된 바와 같이 구체화된 본 발명을 포함하는 시스템의 아키텍처는 본 발명의 실행에 결정적이지 않고, 도2c에 도시된 아키텍처는 도2b의 침입 검출 가속기와 관련하는 것이 바람직하다. 특정하게는, 호스트 CPU(230)와 그 메인 메모리(210)는 본 발명의 하드웨어 파서 가속기가 메인 메모리(210) 및 호스트 CPU(230)와 통신하는 버스(220)에 의해 연결된다. CPU(230)가 메인 메모리(210)와 가속기(100/200) 사이의 통신을 감시할 수 있는 동안에, 토큰은 아직 정의되거나 확립되지 않았고, 공격을 수행하기 위한 코드의 실행은 가능하지 않다. 따라서, 본 발명은 공격에 포함될 수 있는 임의의 동작이 수행되기 전에 구제 인터럽트 또는 예외의 발행을 제공한다.

전술한 설명에 비추어 볼 때, XML™ 문서와 같은 문서의 파싱 시간을 본 발명에 앞서 요구되었던 짧은 시간으로 현저하게 줄이는 하드웨어 파서 가속기의 전후관계 및 환경내에서 시도된 공격의 가능성을 나타낼 수 있는 기호에 대하여 문서의 매우 신속한 차폐를 제공한다는 것을 알 수 있다. 본 발명의 침입 검출 파서는 본 발명에 따른 파서 가속기에 비하여 어떠한 추가적인 요소 또는 하드웨어도 요구하지 않고, 임의의 침입 처리가 실행가능하게 되기 전에 인터럽트 및/또는 예외를 발행할 수 있다.

본 발명은 단일의 바람직한 실시예에 의하여 설명되었지만, 이 기술분야의 당업자이면, 본 발명이 첨부된 청구항의 기술적 사상을 벗어나지 않는 범위 내에서 변경이 가능하다는 것을 인식할 것이다.

(57) 청구의 범위

청구항 1.

문서의 다수의 바이트를 위한 문자 버퍼;

인터럽트 또는 예외, 및 상태 테이블로부터의 다음 상태 데이터 중 적어도 하나에 액세스하기 위해 문서의 바이트 및 상태에 따라 주소지정이 가능한 상태 테이블(state table);

상기 다음 상태 데이터를 저장하기 위한 레지스터;

상기 상태 메모리로의 다른 주소를 형성하기 위해 문서의 후속 바이트와 상기 레지스터의 내용을 결합하기 위한 수단; 및

상기 인터럽트 또는 예외를 호스트 CPU와 통신하기 위한 버스

를 포함하는 침입 검출 시스템.

청구항 2.

제1항에 있어서,

상기 침입 검출 시스템은 파서 내에 구현되는

침입 검출 시스템.

청구항 3.

제1항에 있어서,

상기 상태 테이블은 상기 레지스터와 상기 결합 수단 중 적어도 하나로서 동일 칩상의 메모리에 구현되는

침입 검출 시스템.

청구항 4.

제2항에 있어서,

상기 상태 테이블은 외부 메모리에 구현되는

침입 검출 시스템.

청구항 5.

제4항에 있어서,

상기 상태 테이블이 상기 외부 메모리 내의 구현을 필요로 하지 않을 때 상기 상태 테이블을 저장하기 위해 상기 레지스터 및 상기 결합 수단 중 적어도 하나로서의 동일 칩상의 메모리

를 더 포함하는 침입 검출 시스템.

청구항 6.

제1항에 있어서,

상기 상태 테이블은 네트워크 패킷 전송 속도보다 높은 속도로 액세스되는

침입 검출 시스템.

청구항 7.

제1항에 있어서,

상기 상태 테이블에 인코딩된 하나 또는 그 이상의 시퀀스의 기호를 부합시키는 입력 시퀀스의 발생의 검출에 응답하여 상기 CPU에 표시될 패턴 부합 경보를 표시하기 위한 수단 - 이것으로 인해 응답 속도가 증가됨 -

을 더 포함하는 침입 검출 시스템.

청구항 8.

제7항에 있어서,

상기 인터럽트 또는 예외에 대응하는 침입 경보가 침입 시도를 방지 또는 제한하는 침입 방지 작용을 개시하도록 상기 CPU와 통신되는

침입 검출 시스템.

청구항 9.

제1항에 있어서,

상기 상태 테이블은 네트워크 데이터 패킷 전송 속도와 실질적으로 동일한 속도로 액세스되는

침입 검출 시스템.

청구항 10.

인터럽트 또는 예외, 및 상태 테이블로부터의 다음 상태 데이터 중 적어도 하나에 액세스하기 위해 문서의 바이트 및 상태에 따라 주소지정이 가능한 상태 테이블에 액세스하는 단계;

상기 다음 상태 데이터를 저장하는 단계;

상기 상태 메모리로의 다른 주소를 형성하기 위해 문서의 후속 바이트와 상기 저장된 다음 상태 데이터를 결합하는 단계; 및

상기 인터럽트 또는 예외를 호스트 CPU와 통신하는 단계

를 포함하는 침입 검출 방법.

청구항 11.

제10항에 있어서,

상기 침입 검출 방법은 파서 내에 구현되는

침입 검출 방법.

청구항 12.

제11항에 있어서,

상기 상태 테이블은 외부 메모리에 구현되는

침입 검출 방법.

청구항 13.

제10항에 있어서,

상기 상태 테이블은 네트워크 패킷 전송 속도보다 빠른 속도로 액세스되는

침입 검출 방법.

청구항 14.

제10항에 있어서,

상기 상태 테이블에 인코딩된 하나 또는 그 이상의 시퀀스의 기호를 부합시키는 입력 시퀀스의 발생의 검출에 응답하여 상기 CPU에 표시될 패턴 부합 경보를 표시하는 단계 - 이것으로 인해 응답 속도가 증가됨 -

를 더 포함하는 침입 검출 방법.

청구항 15.

제14항에 있어서,

상기 인터럽트 또는 예외에 대응하는 침입 경보가 침입 시도를 방지 또는 제한하는 침입 방지 작용을 개시하도록 상기 CPU와 통신되는

침입 검출 방법.

청구항 16.

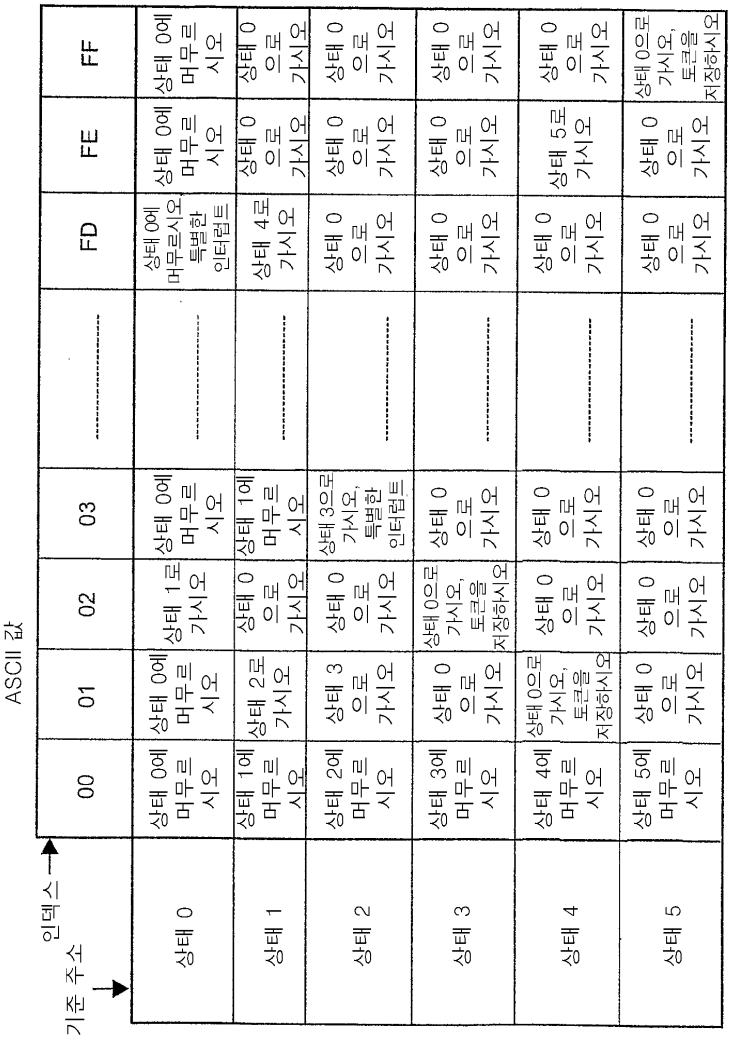
제10항에 있어서,

상기 상태 테이블은 네트워크 데이터 패킷 전송 속도와 실질적으로 동일한 속도로 액세스되는

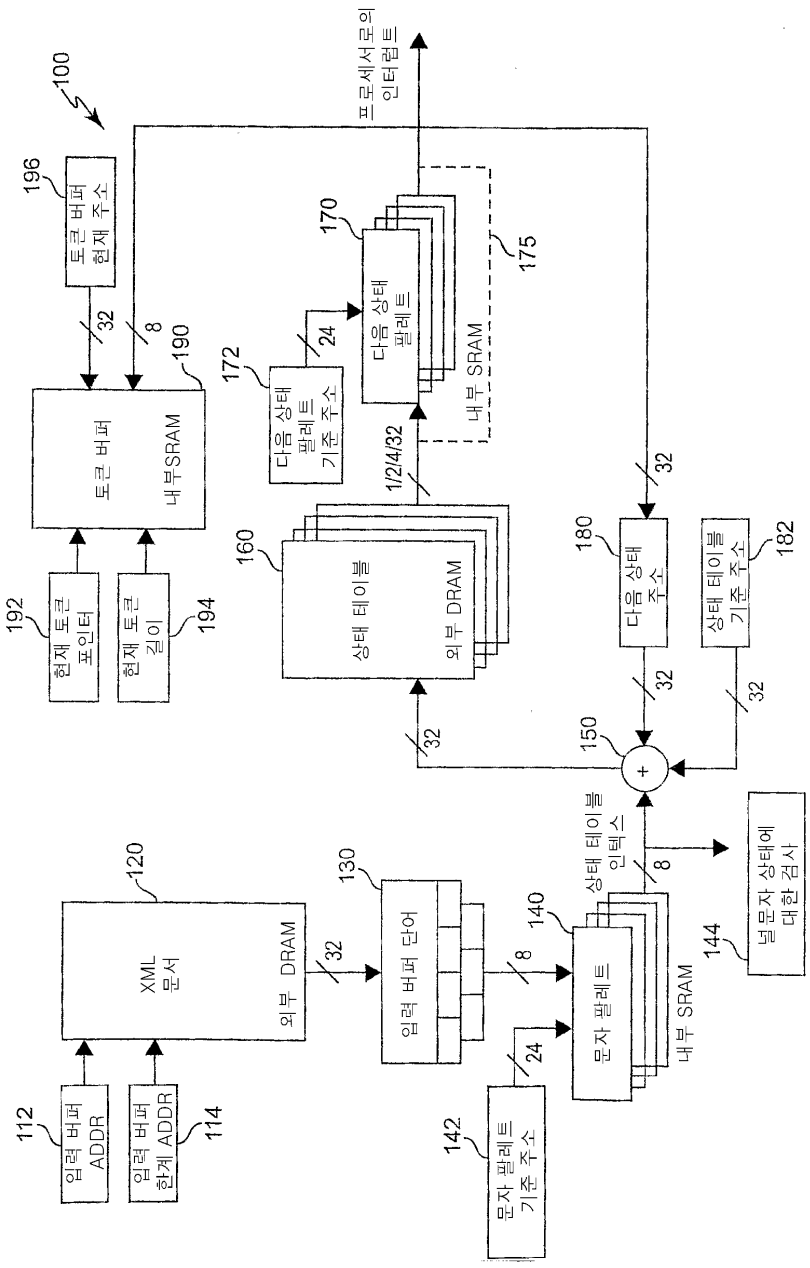
침입 검출 방법.

도면

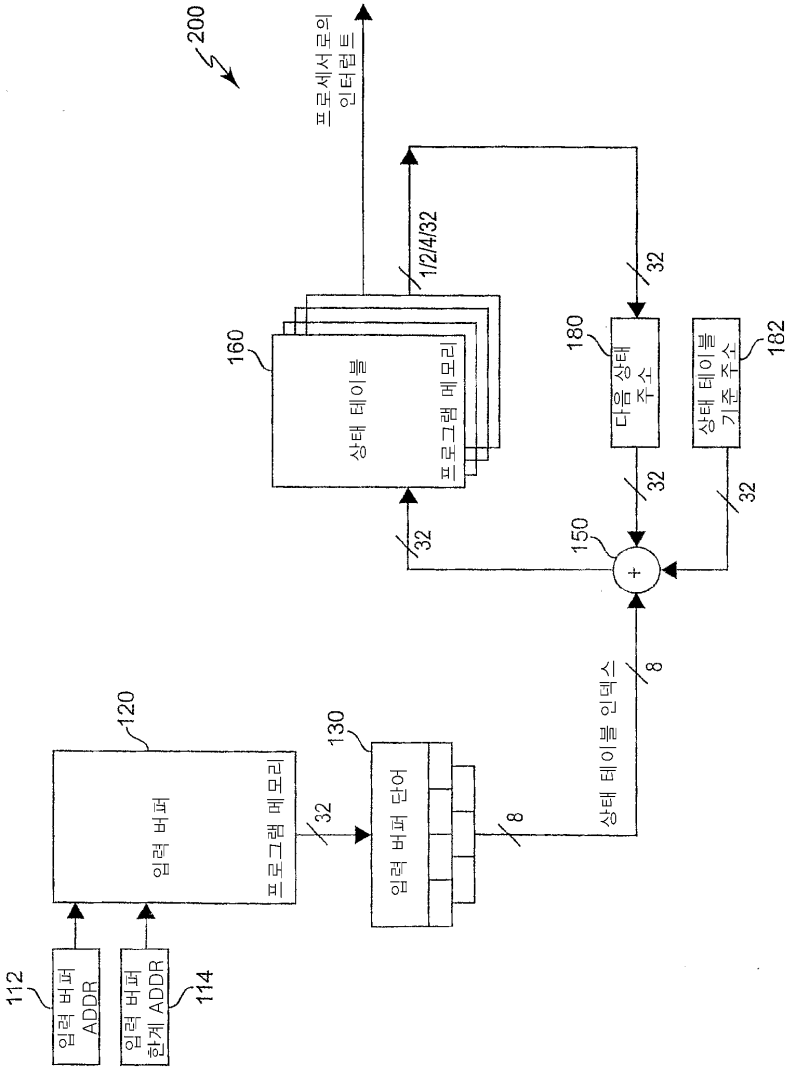
도면1



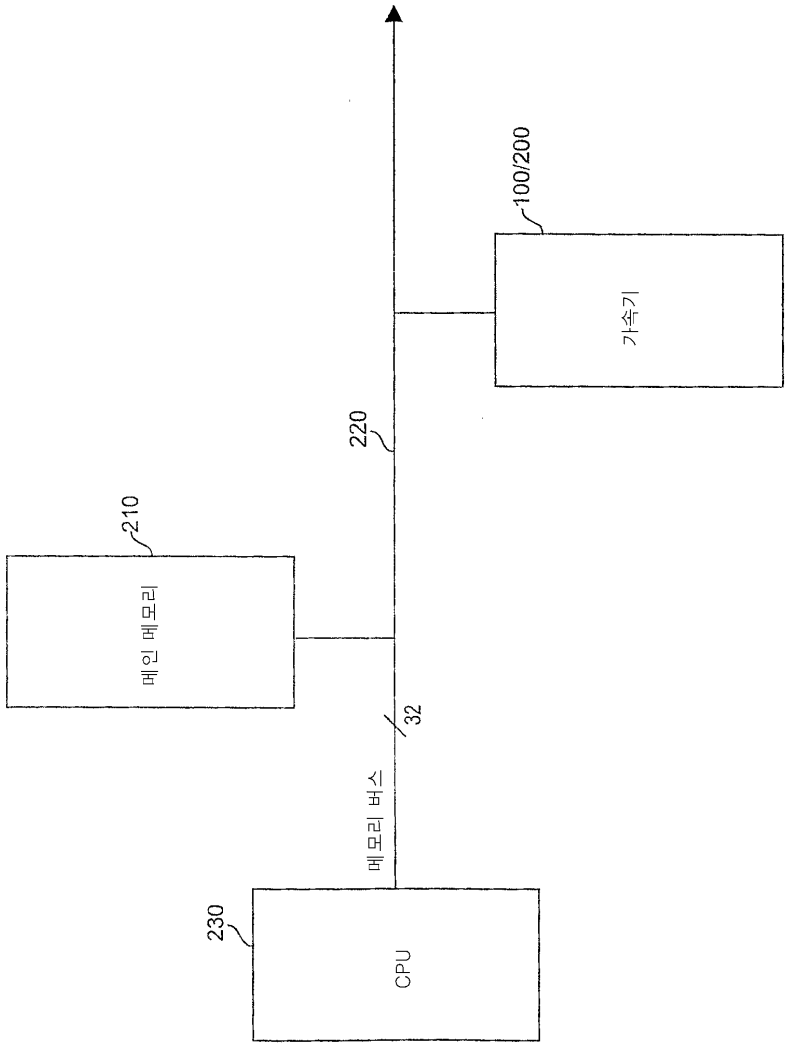
도면2a


100
프로세서로의 인터럽트
196
토큰 버퍼 현재 주소
32
8
190
토큰 버퍼
내부SRAM
192
현재 토큰 포인터
194
현재 토큰 길이
172
다음 상태 팔레트 기준 주소
24
170
다음 상태 팔레트
175
내부 SRAM
1/2/4/32
160
상태 테이블
외부 DRAM
32
180
다음 상태 주소
32
182
상태 테이블 기준 주소
32
150
+
상태 테이블 인덱스
8
120
XML 문서
외부 DRAM
32
130
입력 버퍼 단어
8
140
문자 팔레트
내부 SRAM
144
널문자 상태에 대한 검사
112
입력 버퍼 ADDR
114
입력 버퍼 한계 ADDR
142
문자 팔레트 기준 주소
24

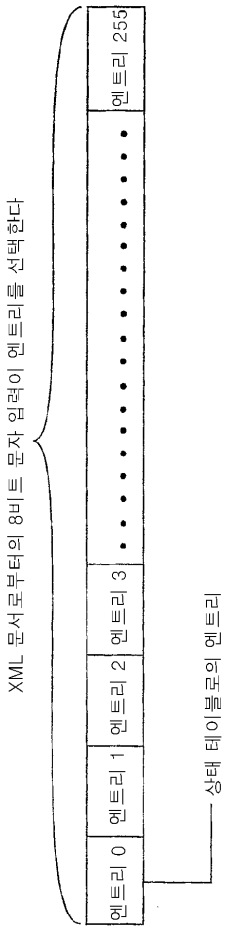
도면2b


200
프로세서로의 인터럽트
1/2/4/32
32
160
상태 테이블
프로그램 메모리
32
180
다음 상태 주소
32
182
상태 테이블 기준 주소
32
150
+
8
상태 테이블 인덱스
120
입력 버퍼
프로그램 메모리
32
130
입력 버퍼 단어
8
112
입력 버퍼 ADDR
114
입력 버퍼 한계 ADDR

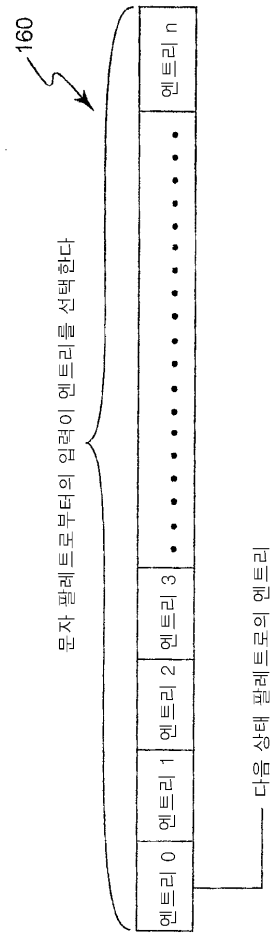
도면2c



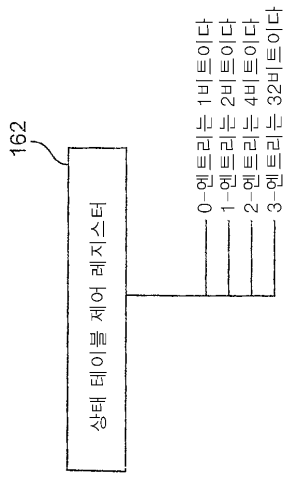
도면3


XML 문서로부터의 8비트 문자 입력이 엔트리를 선택한다
엔트리 0
엔트리 1
엔트리 2
엔트리 3
엔트리 255
상태 테이블로의 엔트리

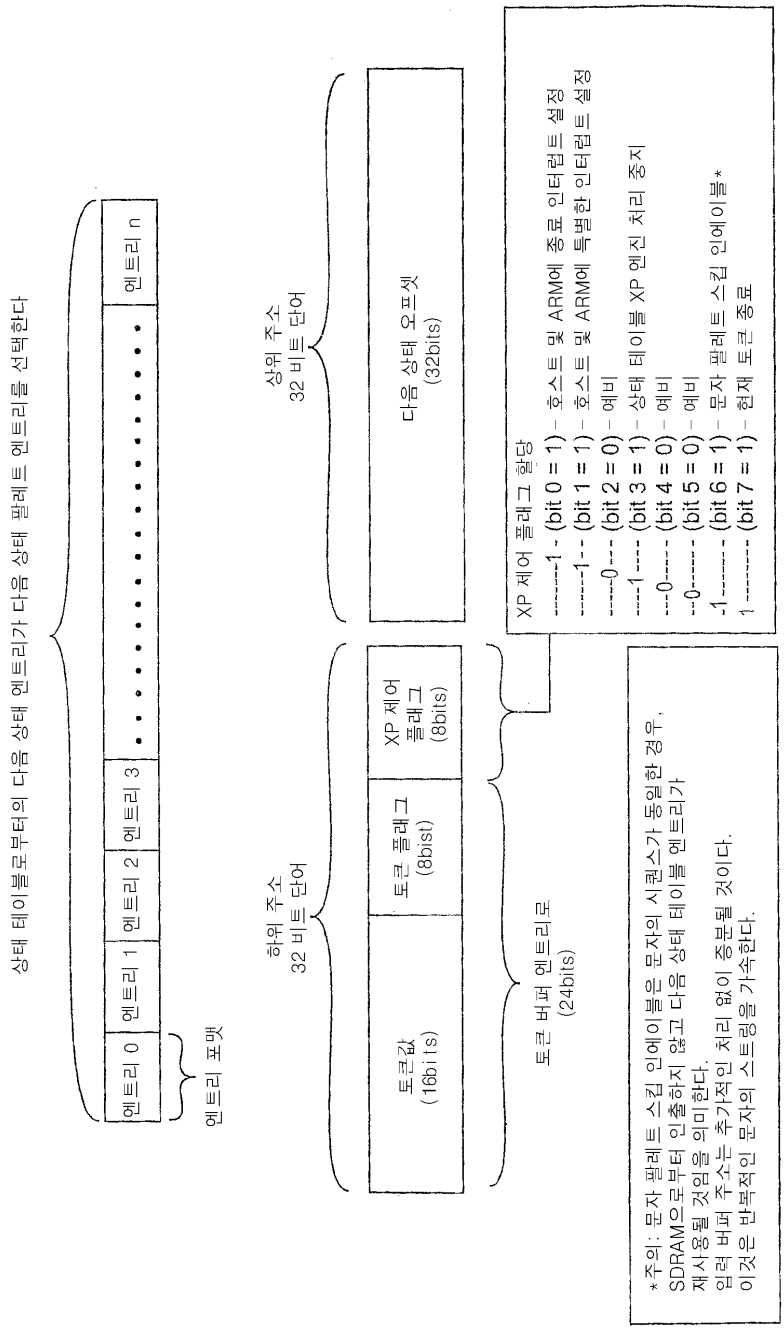
도면4a


160
문자 팔레트로부터의 입력이 엔트리를 선택한다
엔트리 0
엔트리 1
엔트리 2
엔트리 3
엔트리 n
다음 상태 팔레트로의 엔트리

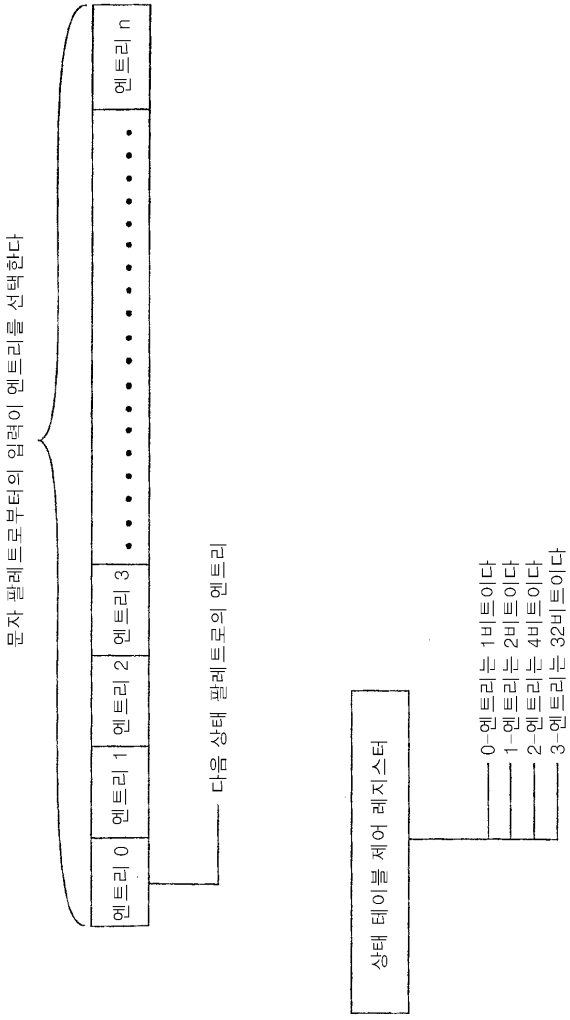
도면4b


162
상태 테이블 제어 레지스터
0-엔트리는 1비트이다
1-엔트리는 2비트이다
2-엔트리는 4비트이다
3-엔트리는 32비트이다

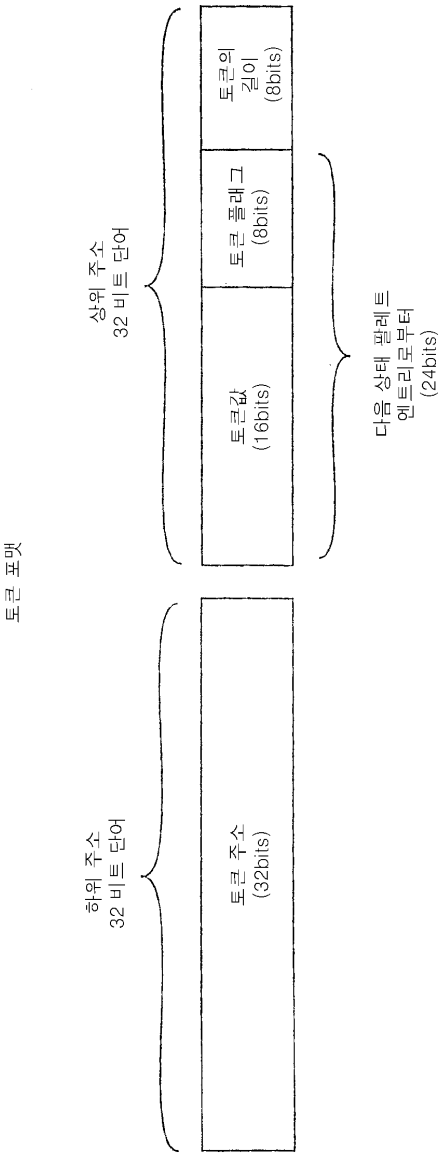
도면5a


상태 테이블로부터의 다음 상태 엔트리가 다음 상태 팔레트 엔트리를 선택한다
엔트리 0
엔트리 1
엔트리 2
엔트리 3
엔트리 n
엔트리 포맷
하위 주소
32 비트 단어
상위 주소
32 비트 단어
토큰값
(16bits)
토큰 플래그
(8bist)
XP 제어
플래그
(8bits)
다음 상태 오프셋
(32bits)
토큰 버퍼 엔트리로
(24bits)
XP 제어 플래그 할당
-------1 - (bit 0 = 1) - 호스트 및 ARM에 종료 인터럽트 설정
------1- - (bit 1 = 1) - 호스트 및 ARM에 특별한 인터럽트 설정
-----0-- - (bit 2 = 0) - 예비
----1---- - (bit 3 = 1) - 상태 테이블 XP 엔진 처리 중지
---0----- - (bit 4 = 0) - 예비
--0------ - (bit 5 = 0) - 예비
-1------- - (bit 6 = 1) - 문자 팔레트 스킵 인에이블*
1-------- - (bit 7 = 1) - 현재 토큰 종료
*주의: 문자 팔레트 스킵 인에이블은 문자의 시퀀스가 동일한 경우,
SDRAM으로부터 인출하지 않고 다음 상태 테이블 엔트리가
재사용될 것임을 의미한다.
입력 버퍼 주소는 추가적인 처리 없이 증분될 것이다.
이것은 반복적인 문자의 스트링을 가속한다.

도면5b


문자 팔레트로부터의 입력이 엔트리를 선택한다
엔트리 0
엔트리 1
엔트리 2
엔트리 3
엔트리 n
다음 상태 팔레트로의 엔트리
상태 테이블 제어 레지스터
0-엔트리는 1비트이다
1-엔트리는 2비트이다
2-엔트리는 4비트이다
3-엔트리는 32비트이다

도면6


토큰 포맷
하위 주소
32 비트 단어
토큰 주소
(32bits)
상위 주소
32 비트 단어
토큰값
(16bits)
토큰 플래그
(8bits)
토큰의
길이
(8bits)
다음 상태 팔레트
엔트리로부터
(24bits)