論文 90-27-8-7

RISC 프로세서 On-Chip Cache의 설계

(Design of A On-Chip Caches for RISC Processors)

洪 仁 植*, 林 寅 七*

(In Sik Hong and In Chil Lim)

要 約

본 논문에서는 RISC(reduced instruction set computer)아키텍춰 상에서 빠른 명령어 Fetch와 데이타 Read/Write를 수행하기 위한 On-Chip 명령어/데이타 캐쉬를 제안하여 RISC 프로세서의 성능향상을 도모하도록 한다.

초기의 RISC 형태에서는 HLL(high level language)프로그램의 실행중 가장 많이 사용되는 로컬 스칼라 변수들은 대형 레지스터군에 저장하여 처리하였으나, 어레이, 구조체 그리고 광역 스칼라 변수들은 컴파일러가 레지스터에 할당하기 어려웠다.[19] 그리고 명령어 엑세스시에는 매번 Pad를 거쳐야 하므로, 프로세서 수행속도를 크게 저하시켰다. 이러한 문제점들은 On-Chip 명령어/데이타 캐쉬를 메모리와 CPU사이에 삽입함으로써 해결할 수 있다. 제안한 캐쉬 메모리는 RISC 프로세서의 속도 개선과 파이프라인 아키텍춰를 효과적으로 지원하기 위하여 정보 RAM으로는 SRAM(static RAM)을 사용하며, 전체 캐쉬 블럭의 회로에는 CMOS와 Hardwired 회로기술을 사용한다. 회로와 Timing 시뮬레이션은 Apollo DN4000 워크스테이션 상에서 Mentor Graphics CAD 툴들을 사용하여 수행한다.

Abstract

This paper proposes on-chip instruction and data cache memories on RISC reduced instruction set computer) architecture which supports fast instruction fetch and data read/write, and enables RISC processor under research to obtain high performance. In the execution of HLL (high level language) programs, heavily used local scalar variables are stored in large register file, but arrays, structures, and global scalar variables are difficult for compiler to allocate registers. These problems can be solved by on-chip Instruction/Data cache. And each cycle of instruction fetch, pad delay causes the lowering of the processors's performance.

Cache memories are designed in CMOS technology and SRAM (static-RAM), that saves layout area and power dissipation, is used for instruction and data storage. To speed up and support RISC processor's piplined architecture efficiently, hardwired logic technology is used overall circuits i cache blocks.

The schematic capture and timing simulation of proposed cache memorises are performed on Apollo DN4000 workstation using Mentor Graphics CAD tools.

*正會員, 漢陽大學校 電子工學科

(Dept. of Elec. Eng., Hanyang Univ.)

接受日字 : 1990年 4月 16日

I. 서 론

반도체 기술의 발전과 함께 컴퓨터 아키텍춰의 구조적 발달이 급속도로 이루어지고 있으며, 특히 기판

(Board)이 아닌 단일 칩으로 구성되는 마이크로 프로세서는 반도체 기술의 혁신과 아키텍춰 기술의 발달로 탁상용 컴퓨터(Desktop Computer)및 워크스테이션(Workstation)으로도 수퍼 미니급의 컴퓨터가 지니는 처리능력을 보유할 수 있게 되었다.[1][3][23]

RISC는 CISC(complex instruction set computer)의 설계 개념과는 달리 출발하여 하드웨어의 단순화를 지향함으로써 하드웨어 자원의 이용효율을 극대화하고 설계 및 개발에 소요되는 비용을 줄일 수 있는 장점이 있다. 또한 RISC 아키텍춰 기법은 모든 명령어를 동일한 크기와 고정된 형식을 사용하여 대부분의 명령을 단일 싸이클로 수행시킬 수 있으며, 거의 일정한 데이타 패스를 거쳐 처리 하게된다. 따라서, RISC는 단일 싸이클 동작이 가능하게 되고, 많은 스테이지를 갖는 파이프라인 아키텍춰로 실현하여 높은 성능을 나타낼 수 있다.[11][14]

초기의 RISC 형태에서는 엑세스 빈도가 높은 로컬 스칼라 변수들을 대형 레지스터군에 저장하여 RISC의 개념에 충실하고자 하였으나, 명령어를 엑세스하기 위해서는 매번 Pad를 거쳐 주기억 장치로부터 가져와야 되었기 때문에 본래의 취지인 빠른 속도를 유지하기 어려웠다.[11][12]

이러한 단점을 보완하기 위하여 최근의 RISC 아키텍춰들은 CPU와 주기억 장치 사이에 고속의 버퍼메모리인 캐쉬(Cache)메모리를 사용하고 있다.[2] 캐쉬 메모리는 그 수행속도에서 주기억장치 엑세스 시간의 10-15% 이내에 참조가 이루어지므로 CPU는 명령어와 오퍼랜드 Load와 Store시 기다리는 시간을 크게 줄일 수 있다.[10][24] 캐쉬 메모리는 CISC 아키텍춰의 대형 시스템은 물론 상용 RISC 프로세서 및 공유 메모리를 갖는 다중 프로세서 분야에서도 활발히 연구 이용되어지고 있다.[5] 이중 대부분의 프로세서들은 캐쉬 메모리 블럭을 CPU외부의 칩(Off-Chip)으로 구성하거나 블럭의 일부를 CPU와 동일한 칩(On-Chip)상에 설계하고 있다.[1][13][23]

본 논문에서는 빠른 명령어 Fetch를 수행할 수 있도록 현재 설계중인 HyRISC 프로세서의 CPU와 동일한 칩상에 명령어 캐쉬 메모리를 설계하며, 컴파일러의 부담을 줄이기 위해 배열, 구조체와 같은 비스칼라 변수들과 광역 스칼라 변수들을 저장하는 데이타 캐쉬 메모리 또한 동일 칩상에 설계함으로써 HyRISC 프로세서의 성능을 향상시킬 수 있도록 한다.

캐쉬 블럭의 세부동작을 제어하는 제어부는 빠른 속도의 제어를 수행할 수 있도록 와이어드 조직(Wired-Logic)기술을 사용하여 설계하였다.

또한, 캐쉬 메모리의 정보 RAM 부분은 그 속도를 중요시하여 SRAM으로 구성하며 태그부는 그 비교수행 목적상 CAM(content addressable memory)셀을 이용하여 설계한다. 설계에 필요한 여러가지 사양들은 HyRISC 프로세서의 크기와 [3][6][13]논문의 DTMR(design target miss ratio)등을 참조하여 결정하며 그 동작및 속도의 측정은 Apollo DN4000 W/S상에서 Mentor Graphics CAD툴들을 이용하여 수행한다.

Ⅱ. 캐쉬 메모리 설계 배경

1. On-Chip 캐쉬의 설계

마이크로 프로세서의 성능은 어드레싱, 오퍼랜드 엑세스, 디코딩, 분기처리, 부동 소숫점 연산 등을 수행하는 속도에 의하여 평가된다. RISC 기술은 이러한 수행속도에 관점을 집중시킨 고성능의 프로세서의 설계를 지향한다. 독자적으로 설계중인 HyRISC 프로세서는 최소의 명령어 세트로 구성되어 있으며 빠른 제어를 위해 제어부는 하드와이어드 콘트롤(Hardwired-control)로 이루어져 있다. 또한 대부분의 Gate들을 와이어드 로직으로 변환하여 설계함으로써 고속의 동작을 가능케하고 있으며 CPU와 동일 칩상에 레지스터 파일이외에 명령어 캐쉬 메모리와 데이타 캐쉬 메모리를 함께 설계하여 파이프라인의 흐름이 보다 신속하고 원활히 수행될 수 있도록 하였다.

타겟 프로세서인 HyRISC의 전체 블럭 다이아그램은 그림 1과 같다.

HyRISC의 데이타패스는 하드웨어 구조를 단순화시키고 수행속도를 향상시키기 위해 많은 부분을 와

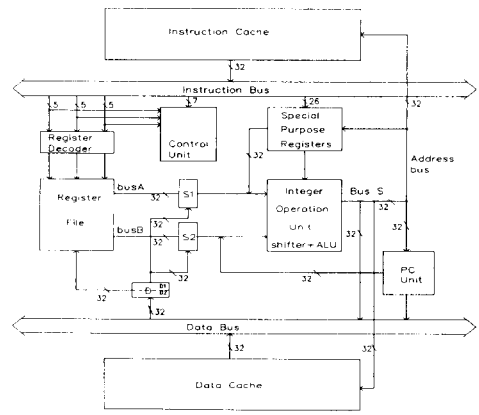


그림 1. HyRISC의 전체 블럭 다이아그램
Fig. 1. Block diagram of HyRISC.

이어드 로직을 사용하여 구성하였으며 쉬프트와 ALU 연산을 하나의 Phase에 처리 해주는 IOU(integer operation unit)를 가지고 있는 등, RISC 아키텍춰의 개념에 충실하도록 설계 되었으며 모든 명령은 거의 일정한 데이타 패스를 거쳐 처리된다.[22][23]

마이크로 코드를 사용하는 CISC 아키텍춰와는 달리 복잡한 명령어 형태를 사용하지 않는 RISC 아키텍춰에서는 캐쉬 메모리를 엑세스 할 때에도 간단한 어드레싱 방식을 사용한다. 본 논문에서는 캐쉬 메모리 엑세스시 인텐스 레지스터의 값과 옵셋(Off Set)의 값을 이용한 간단한 어드레싱 방식을 사용한다. 그리고 캐쉬 메모리의 SRAM-Cell을 제외한 나머지 부분을 단순화된 하드와이어드·로직만을 사용하여 설계함으로써 캐쉬의 국부 제어기(로컬-Controller)가 각 블럭을 빠르게 제어하도록 하고, 1파이프라인 스테이지 싸이클 내에 캐쉬 메모리 엑세스를 수행하여 파이프라인을 효과적으로 지원할 수 있도록 하였다.

2. 레지스터 파일

데이타 패스 중 레지스터 파일은 가장 속도가 빠른 메모리 엑세스 디바이스로서 높은 활용 빈도수를 갖는 로컬-스칼라 변수들을 컴파일러로 부터 할당받아 수행하는 역할을 수행한다. 그러나, 데이타 캐쉬를 같은 칩상에 구현할 경우 로컬-스칼라 변수만을 레지스터 파일에 할당하려면 캐쉬 제어부와 전체 제어부와의 상호관계와 데이타 패스와 캐쉬 메모리 사이의 부가적 경로 발생으로 인하여 하드웨어의 증가가 요구된다.

HyRISC에서는 이러한 점들을 고려, 레지스터 파일을 데이타 캐쉬와 데이타 패스사이의 또하나의 Temporal 메모리로서 설계하였으며, 로컬/광역 변수들의 구분없이 저장되도록 하였다. HyRISC의 레지스터 파일은 32비트 레지스터를 32개 가지고 있으며, 2개의 레지스터를 동시에 읽는 이중 포트(Dual Port)로 구성되어 있다. 이중 레지스터 R0는 항상 0값을 갖는 Hard-wired Zero로 되어있다. (그림2(a))[11][23]

3. 명령어 캐쉬와 데이타 캐쉬

파이프라인 아키텍춰의 데이타 패스를 고속으로 제어하기 위하여, RISC 아키텍춰의 프로세서들은 보다 간단한 명령어 형태와 빠른 명령어 Fetch를 요구한다. 이러한 요구를 만족시키기 위해서는, 즉 파이프라인의 파괴없이 빠른 동작을 수행하려면 데이타 패스와 메인 메모리 사이에 필연적으로 버퍼 메모리가 존재하여야 하며, 이를 위하여 본 논문에서는 1024 바이트 크기의 캐쉬 메모리를 설계하였다. 레지

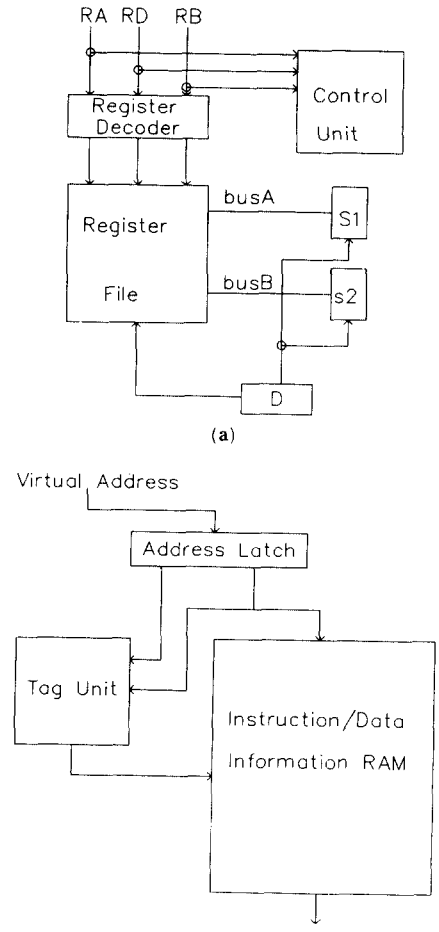


그림 2. (a) 레지스터 파일
(b) 캐쉬 메모리
Fig. 2. (a) Register file,
(b) Cache memory.

스터 파일 엑세스시, 명령어내의 수 비트를 사용하여 엑세스하게 되나 캐쉬 메모리의 경우 어드레스 계산을 통한 Full-Length 메모리 어드레스를 이용하게 된다. (그림 2(b))

Ⅲ. 캐쉬메모리의 설계를 위한 사양

1. 어드레스 Mapping

어드레스 변환과 캐쉬 메모리 엑세스의 상호관계는 캐쉬 메모리 엑세스 시간에 영향을 주게된다. 그림 3은 여러가지 어드레스 Mapping 방식을 보여준다.

캐쉬 메모리 설계를 위하여 (c)방식을 사용하며, 이 방식은 실제 어드레스를 사용하지 않고 가상 어드레

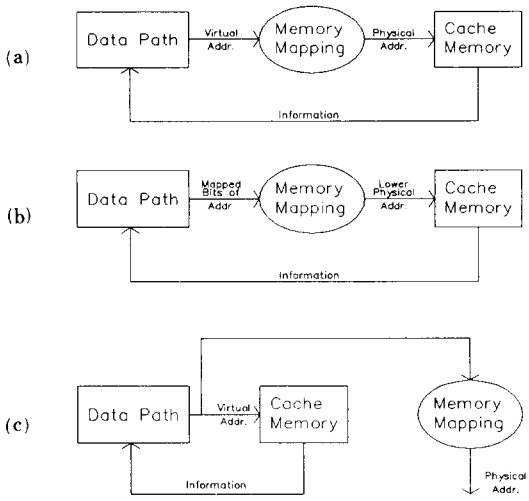


그림 3. Mapping 방식의 종류
Fig. 3. Various mapping method.

스를 사용하는 방식으로써 CPU에서 계산된 가상 어드레스가 어드레스 변환을 거치지 않고 캐쉬 메모리를 엑세스하기 위하여 직접 사용되므로 어드레스 변환으로 인한 지연 시간과 TLB(tanslation lookaside buffer)Miss로 인한 지연이 문제시 되지 않는다. 또한, 가상 메모리 어드레스를 사용하여 설계하면 그 크기를 임의대로 크게 설계할 수 있으므로 Miss Ratio를 감소시킬 수 있다.[22]

2. 캐쉬 메모리의 크기

캐쉬 메모리의 크기는 캐쉬의 Miss Ratio를 결정하는 중요한 인자(Factor)들 중의 하나로 여러가지 요인들에 의하여 제약을 받는다. 일반적으로, 캐쉬 메모리의 크기와 Miss Ratio는 반비례 관계를 가지나 그 크기가 지나치게 커지면 오히려 많은 팬 인(Fan-in)과 팬 아웃(Fan-out)의 요구, 그리고, 회로의전력 공급으로 인한 느린 상승시간 등으로 인하여 실행 속도가 저하된다. 그래서, 캐쉬 메모리의 크기는 칩(Chip)과 보드(Board)의 면적, 설계비용, 소비전력, 냉각 조건등의 측면에서 효율적인 크기이어야 하며, 여러가지 Trace-Driven 시뮬레이션을 통해 그 크기와 Set의 수, Line의 크기 등을 결정하여야 한다. 본 논문에서는 [6][13]논문의 DTMR(design target miss ratio)과 HyRISC의 전체 칩 면적등을 고려하여 64바이트의 Line Size를 갖는 1024바이트의 명령과 데이타 캐쉬를 2 Set으로 구성되는 8-way Set Associative 방식으로 설계한다. (표 1)

3. 캐쉬 메모리의 Line Size(Block Size)

캐쉬 메모리의 Line Size는 메모리 지연(Memory Latency)을 고려하여 16내지 64바이크의 크기로 설계하는 것이 적합[3]하며, HyRISC의 경우 그림4의 예제 프로그램을 통한 시뮬레이션으로 얻어진 DTMR을 참고하여 64바이트의 Line Size를 선택한다.

4. 캐쉬의 용도별 분리

캐쉬 메모리는 크게 명령어와 데이타를 함께 저장하는 Unified(Mixed)캐쉬 형태와 분리해서 사용하는 두가지 형태를 생각할 수 있는데, RISC의 파이프라인 아키텍춰 상에서는 흔히 명령어 Fetch와 데이타 Read/Write가 독립적으로 발생하므로 Unified캐쉬의 경우 엑세스 Conflict를 유발시켜 프로세서의 성능을 저하시킨다. (표 1) 또한, Conflict가 발생하지 않는 경우라도 중재 지연(Arbitration delay)을 유발할 수 있다. 그러나, 명령어와 데이타 캐쉬를 분리하여 설계할 경우 이러한 장애 요소들을 피할 수 있고, 명령어와 데이타를 독립적으로 엑세스할 수 있어 이중 포트(Dual Port)의 효과를 얻을 수 있다. HyRISC 에서는 이 두가지 캐쉬를 각각 명령어 버스와 데이타 버스를 통해 연관이 깊은 기능 블럭 근처에 위치시켜 연결함으로써 캐쉬 메모리 엑세스 시간을 감소시킬 수 있도록 한다.[6][24]

5. Associativity

메인 메모리의 어드레스를 캐쉬 메모리로 맵핑(Mapping)하는 방식으로 Direct, Set-associative, Fully-associative 등의 방식이 있는데, 이중 Set-associative 방식이 나머지 두 방식의 중간적인 구조를 가지면서 가장 널리 사용되고 있다. 이 방식은 몇개의 Line이 모여 Set을 형성하는 Set-associative 캐쉬 메모리로서 어드레스가 Set 내부로 맵핑되고 선택된 Set에 대해서 순차적인 정보검색이 이루어진다. Set-associative 캐쉬의 동작을 그림 4 에 나타내었다.

일반적인 프로그램 Trace-Driven 시뮬레이션의 결과를 보면 Associativity를 증가시킬 경우 Miss-Ratio는 감소된다. 그러나, Associativy를 증가시킨다 해도 동일한 비율로 Miss-Ratio가 감소하지는 않는다. 즉, Set-Associatity에 따른 Mjss-Ratio의 감소는 Workload, Line Size, 그리고 캐쉬 메모리에 의해 제약을 받는다. 대개의 경우 8-32K 바이트 이하의 캐쉬를 설계할 때 Direct Mapped 캐쉬보다 Set-associative 캐쉬가 더욱 좋은 성능을 나타내며 또한 Set의 수도 커다란 변수로 작용한다. HyRISC에서는 이러한 점들과 DTMR, 그리고 칩면적을 고려하여 8-

표 1. 캐쉬 메모리의 설계사양 결정을 위한 DTMR
Table 1. DTMR for decision of cache memory design spec.

Design target miss ratio

Design Target Miss Ration for Unified Caches												
Cache	Block Size 16 Bytes				Block Size 32 Bytes				Block Size 64 Bytes			
Size	8-way	4-way	2-way	1-way	8-way	4-way	2-waz	1-way	8-way	4-way	2-way	1-way
1K	0.210	0.219	0.239	0.288	0.162	0.170	0.188	0.244	0.137	0.144	0.162	0.229
2K	0.170	0.179	0.197	0.240	0.124	0.130	0.146	0.188	0.098	0.104	0.118	0.163
4K	0.120	0.126	0.140	0.172	0.082	0.087	0.097	0.126	0.059	0.063	0.072	0.099
8K	0.080	0.084	0.093	0.116	0.052	0.053	0.059	0.077	0.033	0.035	0.040	0.055
16K	0.060	0.063	0.069	0.088	0.036	0.038	0.042	0.055	0.023	0.025	0.028	0.038
32K	0.040	0.042	0.046	0.059	0.024	0.025	0.028	0.037	0.014	0.015	0.017	0.023

Design Target Miss Ratios for Instruction Cache												
Cache	Block Size 16 Bytes				Block Size 32 Bytes				Block Size 64 Bytes			
Size	8-way	4-way	2-way	1-way	8-way	4-way	2-way	1-way	8-way	4-way	2-way	1-way
1K	0.200	0.211	0.234	0.271	0.134	0.140	0.155	0.179	0.098	0.104	0.115	0.133
2K	0.150	0.159	0.179	0.210	0.098	0.103	0.117	0.139	0.068	0.072	0.082	0.097
4K	0.100	0.106	0.120	0.143	0.063	0.067	0.076	0.091	0.043	0.046	0.053	0.063
8K	0.060	0.064	0.072	0.089	0.037	0.039	0.046	0.056	0.023	0.025	0.028	0.035
16K	0.050	0.053	0.060	0.078	0.029	0.031	0.034	0.045	0.018	0.019	0.022	0.029
32K	0.030	0.033	0.038	0.046	0.017	0.018	0.021	0.027	0.010	0.011	0.011	0.016

Design Target Miss Ratios for Data Cache												
Cache	Block Size 16 Bytes				Block Size 32 Bytes				Block Size 64 Bytes			
Size	8-way	4-way	2-way	1-way	8-way	4-way	2-way	1-way	8-way	4-way	2-way	1-way
1K	0.160	0.170	0.192	0.244	0.138	0.146	0.166	0.216	0.147	0.150	0.170	0.227
2K	0.120	0.127	0.143	0.183	0.094	0.106	0.114	0.149	0.084	0.089	0.102	0.138
4K	0.100	0.106	0.117	0.148	0.070	0.075	0.084	0.109	0.054	0.058	0.067	0.090
8K	0.080	0.084	0.092	0.118	0.053	0.056	0.061	0.081	0.039	0.042	0.047	0.064
16K	0.060	0.062	0.068	0.084	0.039	0.041	0.045	0.058	0.026	0.028	0.031	0.042
32K	0.040	0.041	0.045	0.055	0.025	0.026	0.028	0.037	0.017	0.018	0.020	0.027

way Set-Associativity로 명령어와 데이타 캐쉬를 설계한다.

6. Fetch 알고리듬

이것은 언제 어떤 정보를 캐쉬 메모리에서 엑세스할 것인지 결정하는데 사용되어지는 정책(Policy)으로서 크게 두 가지로 분류된다. 필요할때마다 Line을 Fetch하는 Demand Fatch 방식과 미리 앞서서 필요하게될 정보를 추측하여 Fetch하는 Prefetch 방식이 그것이다. 이중 Prefetch 방식은 Miss-Ratio를 크게 줄일 수는 있으나 Traffic이 증가하는 단점을 가지고 있다. 본 논문에서는 Demand Fetch 방식을 사용한다.

7. 대치 정책(Replacement Policy)

캐쉬 메모리에서 Miss가 발생하여, 하나의 새로운 블럭이 캐쉬로 들어올 때, 캐쉬가 완전히 채워있는 상태(Full state)에서는 오래된 블럭중의 하나가 제거되어져야만 한다. 일반적으로 하나의 프로그램을 수행시킬 때 캐쉬 메모리는 충분히 크지 못하므로 모든 Working Set을 캐쉬내에 저장시킬 수는 없다. 그러나, 캐쉬는 Locality-of-Reference의 성질을 가지

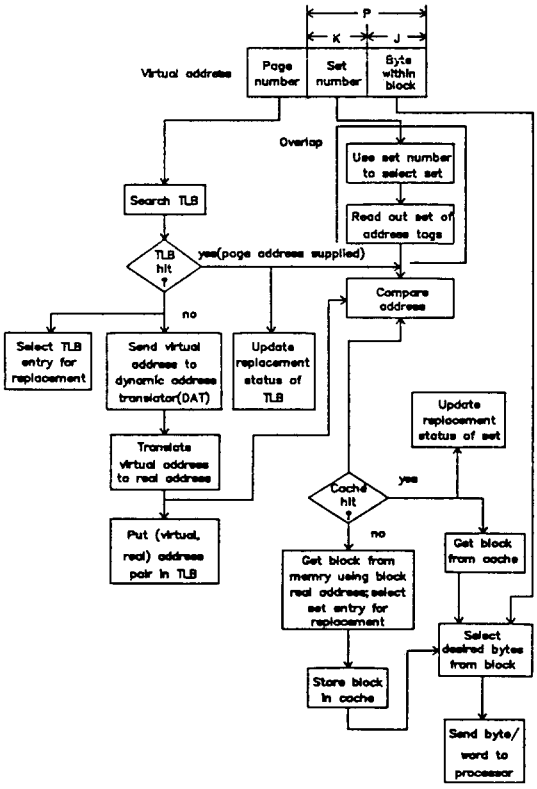


그림 4. Set-associative 캐쉬의 동작
Fig. 4. Diagram of Set-associative cache.

고 있으므로 좋은 대치 알고리듬을 만들 수 있는 단서를 제공한다. Set-Associative 캐쉬에 있어서는 각 Set들 자체내의 Block Frame들에 의해서만 수행하여진다. 그러므로 단지 고정된 공간의 대치 알고리듬들이 고려되어지고 있다. 이러한 대치 알고리듬에서는 LRU, FIFO 그리고 Random등이 있는데 LRU 정책이 가장 좋은 성능을 나타낸다. 그러나, 본 논문에서는 구현(Implementation)이 간단하고 다른 알고리듬에 비해 성능이 크게 뒤떨어지지 않는 FIFO 알고리듬을 대치정책으로 사용한다.

8. 주 기억 장치 Update Policy

CPU가 메모리에 데이타를 Write 할 때, 그 동작은 실제로 캐쉬와 메인 메모리에 여러가지 방법으로 반영되어 진다. 예를들어 캐쉬 메모리에 임의의 데이타가 Write된 후에 캐쉬 Miss로 인하여 그 Line (Block)이 대치되어 질때 메인 메모리의 내용을 Update하는 경우, 이러한 방식을 Copy-Back이라 한다. 또다른 방식으로는 데이타에 변화가 있을 때마다 캐쉬 메모리와 주기억 장치를 동시에 Update 하는 방식으로 Write-Through 방식이 있다. 본 논문에서는 불필요한 메모리 Traffic을 줄이기 위해 Copy-Back 방식을 사용한다.

Ⅳ. 캐쉬 메모리의 설계

HyRISC의 On-Chip 캐쉬 메모리는 그림 6과 같이 여러개의 블럭으로 나누어 설계하였다.

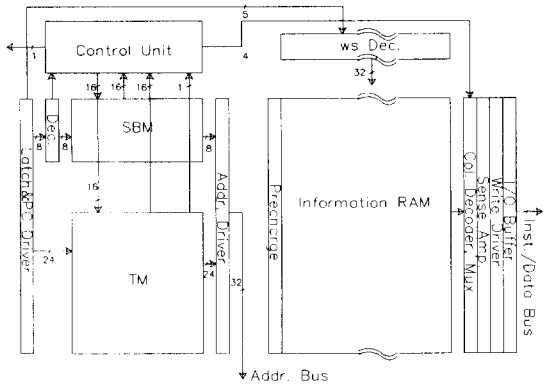


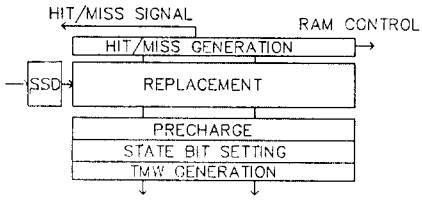
그림 5. 캐쉬 메모리의 전체 구성도
Fig. 5. Block diagram of cache memory.

1. 제어부(Control Unit)

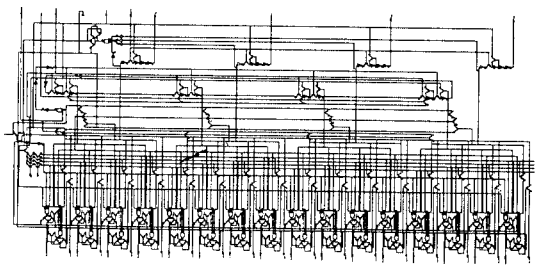
캐쉬의 제어부는 HyRISC의 전체 제어부로 부터 전체적인 동작을 지정받고 세부적인 동작은 독자적으로 처리하도록 설계하였으며, CPU에서 제공받은 어드레스로부터 Set를 선택하고 태그부에서의 HIT/MISS 신호를 발생시킨다. 그리고 각 블럭의 프리차아지(Precharge)를 수행한다. 또한 상태 비트(State Bit)를 워드(Word)의 유효성에 맞도록 세팅(Setting)시키며, 태그 대치(Tag Replacement)와 캐쉬 RAM으로 제어신호를 주어 RAM의 Read/Write를 수행한다. 제어부의 구성요소와 회로도는 그림6에 나타내었다. HIT/MISS 신호는 각 Line의 로컬 HIT/MISS 신호를 ORing하여 얻어내며, 재 충전은 4-phase 클럭 신호에 의해 제어된다. 그리고, FIFO 알고리듬의 태그 대치는 Modulo-8 카운터를 이용하여 실현하였다.

2. 태그부(Tag Unit)

태그부는 어드레스 태그를 저장하는 태그 메모리와 데이타의 상태 비트를 저장하는 상태 비트 메모리 그리고 어드레스 버스 드라이버로 구성되어 있다.



(a)



(b)

그림 6. (a) 제어부의 기능도
(b) 제어부의 회로도
Fig. 6. (a) Functional diagram of control unit,
(b) Circuit diagram of control unit.

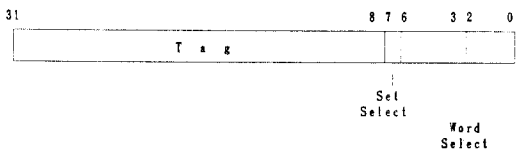
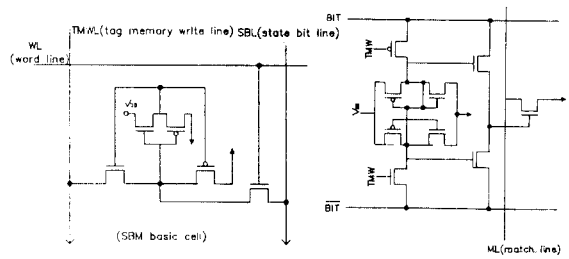


그림 7. 가상 어드레스의 구성
Fig. 7. Organization of virtual address.

HyRISC의 PCU(program counter unit)로 부터 캐쉬 메모리에 전달되는 가상 어드레스(Virtual Address)는 전체 32bit로 다음과 같은 구성을 갖는다. (그림 7)

이 중 최하위 3비트는 블럭 중의 워드 지정에 사용되며 Set선택과 워드 선택의 5비트를 사용하여 캐쉬 RAM의 Line과 Set를 선택하게 된다. 태그부에 저장되는 데이타는 384개의 어드레스 태그 메모리 비트와 256개의 상태 비트 메모리로 구성되어 있다. 이중 태그 메모리는 24개의 셀로 이루어진 16개의 어드레스 태그로 구성된 태그 어레이(Tag 어레이)와 비교 종료 어레이(Compare-Finished 어레이)로 구성된다. 셀은 써 넣은 후 비교만이 가능한 CAM (content-addressable memory)셀을 사용하였다.



(a) (b)

그림 8. (a) 상태 비트 메모리의 기본 셀
(b) 태그 메모리의 기본 셀
Fig. 8. (a) Basic cell of state bit,
(b) Basic cell of tag memory.

그림 8에 상태 비트 메모리와 태그 메모리의 기본 셀을 나타내었다.

상태 비트 메모리는 CPU가 요구하는 Word의 유효성을 저장하게 되며 WSD(word selection decoder와 상태 비트 어레이(State Bit 어레이)로 구성된다. 이중 상태 비트 어레이는 어드레스 태그마다 16개의 셀로 구성되고 WL(word line)의 값에 따라 동작한다. 태그 Miss 시에는 TMW Line을 High로 구동시켜 태그와 함께 대치된다.

3. 정보 RAM(Information RAM)

정보 RAM은 그림 5에서와 같이 Word Select

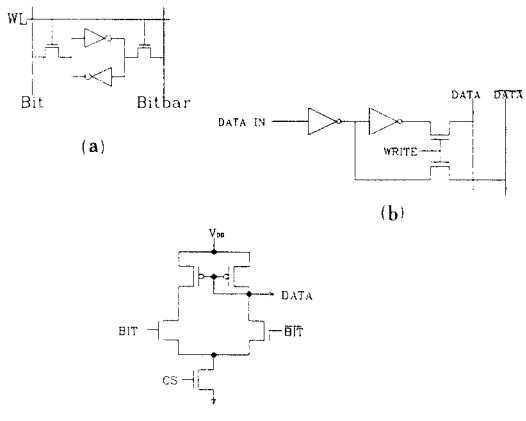


그림 9. (a) SRAM의 기본 셀
(b) Sense Amp의 기본 셀
(c) Write driver의 기본 셀
Fig. 9. (a) Basic cell of SRAM,
(b) Basic cell of sense Amp,
(c) Basic cell of write driver.

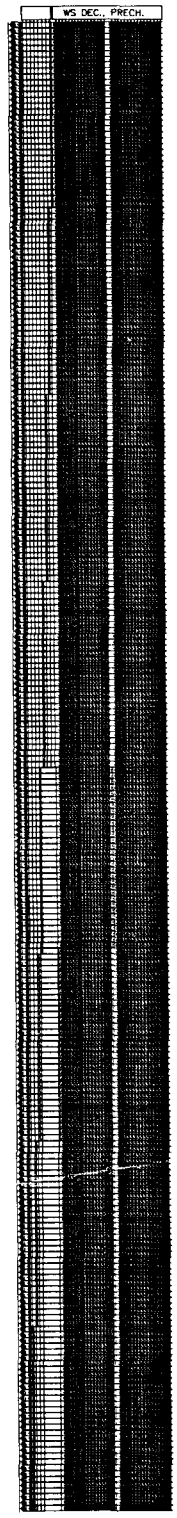


그림10. RAM 부분의 회로도
Fig. 10. Circuit diagram of RAM part.

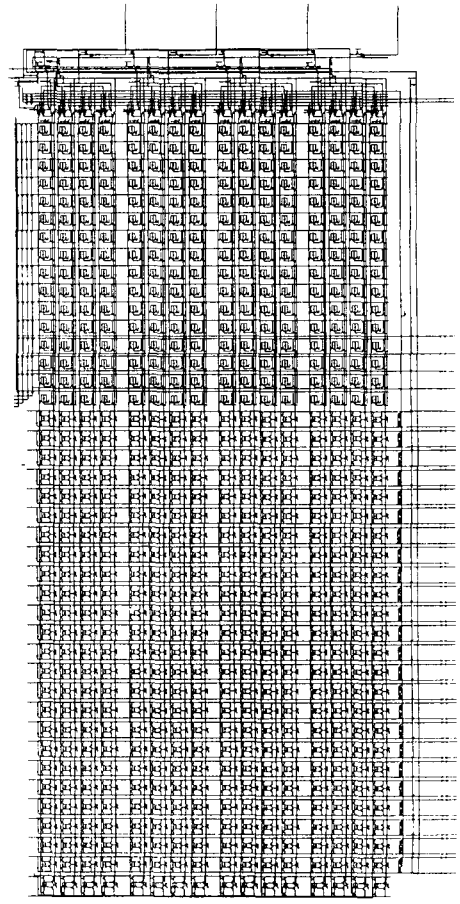


그림11. (a) 제어부와 태그부의 회로도
Fig. 11. (a) Circuit diagram of control unit and tag unit.

Decoder, RAM, MUX, SenseAmp., 그리고 Write Driver 등으로 구성된다. 이중 디코더는 가상 어드레스 중 세트 선택과 Word 선택 5비트를 가지고 CPU가 요구하는 Word Line을 선택하게 되는데, 회로는 패스 TR.만을 사용하여 빠른 동작을 하도록 하였다.

RAM 부분은 그 속도를 중요하게 생각하여 SRAM (static-RAM)으로 구성하였으며, Read/Write를 위하여 차등 Sense Amplifier와 Write Driver를 RAM의 측면에 설계하였다.

그림 9에 각 부분에 대한 기본 셀들을 표시하였다. SRAM의 Associativity와 Set에 따른 회로도는 그림 10에 나타내었다.

RAM 부분을 제외한 캐쉬 메모리 전체의 회로도는 그림11에 나타내었다.

V. 시뮬레이션 및 분석

제안한 캐쉬 메모리의 각 블럭에 대한 시뮬레이션은 Apollo 워크스테이션 상에서 Mentor Graphics CAD 툴들을 사용하여 수행하였다. Schematic Editor의 사용시 각 기본 소자, 즉 N-채널 MOS 트랜지스터, P-채널 트랜지스터 그리고 인버터의 지연시간을 2μm-Double Metal 디자인 룰을 기초로 하는 SPICE 시뮬레이션 결과를 적용시켜 각각 1ns, 1ns 그리고 1.6ns으로 하였다. 우선 현재 상용화된 프로세서들의 시스템 사이클 타임을 표 2에 나타내었다. 표 2에서 여타 프로세서들의 공정 기술로는 대부분 1.2μm 이하의 디자인 룰을 적용하고 있고 사이클 타임은 이의 적용하에서 측정된 결과치 들이다. 예를 들어 1.2μm 디자인룰에서의 패스 트랜지스터 지연은 0.5ns 이하이다.

표 2. 프로세서들의 시스템 사이클 타임
Table 2. System cycle time of various processors.

Machine	Pipeline	Cycle Time
RISC II	3	330ns (12MHz)
AM29000	4	40ns (25MHz)
SPARC	4	30ns (33MHz)
MIPS LR300 ALC	5	40ns (25MHz)
MIPS LR2000/16	5	60ns (17MHz)
MIPS-X	5	50ns (20MHz)
SPUR	6	100ns (10MHz)
MC88000	4	50ns (20MHz)
i860	4	25ns (40MHz)
HyRISC	4	60ns (17MHz)

표 2에서와 같이 HyRISC의 파이프 라인 1스테이지의 요구시간은 60ns으로 설정되어 있다. 본 논문의 캐쉬 메모리 시뮬레이션 결과를 보면 제어부와 태그부에서 태그의 비교를 끝내고 HIT/MISS 신호 발생까지 20ns, 그리고, HIT 신호 발생시 RAM 엑세스 시간이 약 20ns으로 정상적인 경우 40ns 이내에 전체 캐쉬 메모리의 엑세스를 마칠 수 있어 HyRISC의 파이프라인을 지연시키지 않고 효과적으로 지원할 수 있다.

표 3. 프로세서들의 캐쉬 메모리 엑세스 타임
Table 3. Cache memory access time of various processors.

Processor	Access Time
SPUR (TTL & CMOS RAMs)	100ns
Titan I (ECL RAMs)	45 ns
MIPS-X (CMOS RAMs)	33 ns
i860 (CMOS RAMs)	25 ns (128 bit BUS)
HyRISC (CMOS RAMs)	40 ns

표 3에 여러 프로세서들의 캐쉬 메모리 엑세스 타임을 나타내었다.

전체 캐쉬 블럭의 설계에는 약 16만개의 트랜지스터가 소요되었으며 제어부와 SBM 그리고 태그부를 레이아웃 해보았을때 약 3.3×10mm² 정도의 면적이 소요되었다. 결국 HyRISC의 데이타 패스와 On-Chip 캐쉬를 모두 하나의 칩에 구현하려면 현재의 디자인 룰로는 많은 어려움이 있을 것으로 예상된다.

만약 현재 시뮬레이션에 사용한 2μm를 보다 좋은 디자인 룰과 ECL등의 고속 회로 기술을 사용한다면 적정 레이아웃 면적 내에서 더욱 빠른 동작의 결과를 얻을 수 있을 것이다.

VI. 결 론

본 논문에서는 빠른 명령어 Fetch와 데이타 Read/Write를 수행할 수 있도록 HyRISC 프로세서의 CPU와 동일한 칩상(On-Chip)에 명령어와 데이타 캐쉬 메모리를 설계하여 HyRISC 프로세서의 성능을 향상시킬 수 있도록 하였다.

적합한 설계사양을 얻기 위해 전체 칩의 구조와크기, 그리고 여러가지 DTMR과 trace-driven 시뮬레이션의 결과를 이용하였으며, 칩의 크기와 속도 면에서의 trade-off를 고려하였다.

본 논문에서 제안한 명령어/데이타 캐쉬 메모리의 설계와 시뮬레이션은 Apollo 워크스테이션 상에서 Mentor graphics CAD 툴들을 이용하여 수행하였으며, 40ns의 캐쉬 메모리의 엑세스 타임을 얻어 한계치인 45ns내에 동작을 완료할 수 있음을 보였다.

앞으로 보다 나은 회로기술과 디자인 룰을 적용시켜 설계한다면 높은 속도를 기대할 수 있으며 HyRISC의 데이타 패스와 함께 시스템 레벨 시뮬레이션을 행하여 전체 시스템의 성능 향상을 꾀하는 것이 앞으로의 연구 과제이다.

參 考 文 獻

[1] A.J. Smith, "Cache memories," *Computing Surveys*, vol. 14, no. 3, September 1982.
[2] A.J. Smith, "Design of CPU cache memories," *Proc. IEEE TENCON, Seoul*, Korea, August, 1987.
[3] A.J. Smith, "Line size choice for CPU cache memories," *IEEE Transactions on*

Computers, vol. C-36, no. 9, Septemer 1987.

[4] Benjamin Maytal, Sorin Iacobovici, et al., "Design considerations for a general purpose microprocessor," *IEEE Computer,* vol. 22, no. 1, January 1989.

[5] D.A. Wood, Susan J. Eggers, Garth Gibon, "SPUR meory system architecture," UCB/CAS Version 2.1: July 1988

[6] "Eavluating associativity in CPU caches, "M.D. Hill, alan. J. Smith, *IEEE Transactions on Computers,* vol. 38, no. 12, Dec. 1989.

[7] Hideto Hidaka, Kazuyasu Fujishima, Yoshio Matsuda, et al AM's, "Twisted bit-line architecture," *IEEE J. of Solidstate Circuits,* vol. 24, no. 1, February 1989.

[8] Hiroshi Kadota, Jiro Miyake, Ichiro Okabayashi, et al, "A 32-bit CMOS microprocessor with on-chip cache and TLB," *IEEE J. of Solid-State Circuits* vol. SC-22, no. 5, October 1987.

[9] Jiren Yuan and Christer Svensson, "High-speed CMOS circuit technique," *IEEE J. of Solid-State Circuits,* vol. 24, no. 1, February 1989 .

[10] Kimming So and Rudolph N. Rechtesch-affen, "Cache operation by MRU change," *IEEE Transactions on computers,* vol. 37, no. 37, no. 6, June 1988.

[11] Manolis G.H. Katevenis, "Reduced instruction set computer architectures for VLSI," ACM Doctoral Dissertation Award 1984, the MIT Press.

[12] Mark Horowitz, Paul Chow, Don Stark, etl al, "MIPS-X: A 20-MIPS peak 32-bit Micr-oprocessor with on-chip cache," *IEEE J. of Solid-State Circuits,* vol. SC-22, no. 5, October, 1987.

[13] M.D. Hill, "Aspects of cache memory and instruction buffer performance," Report no. UCB/CSD87/381, November 1987.

[14] Paul Chow, Mark Horowitz, "Architectural tradeoffs in the design of MIPS-X, " ACM 1987.

[15] Robert W. Sheburne, Jr., Manolis G.H. Katevenis, et al "A 32-bit NMOS microprocessor with a large register file," *IEEE J. of Solid State Circuits,* vol. SC-19, no. 5, October 1984.

[16] Paul Chow and Mark Horowitz, "Architectural tradeoffs in the design of MIPS-X, ACM, 1987.

[17] SUBHASIS LAHA, JANAK H. PATEL, "Accurate low-cost methods for performance evaluation of cache memory systems," *IEEE Transactions on Computers,* vol. 37, no. 11, November 1988.

[18] Tekla S. Perry, "Intel's secret is out," *IEEE, SPECTRUM,* 1989. 4

[19] William Stallings, "Reduced instruction set computers," *IEEE Computer society Press.*

[20] Gerry Kane, "MIPS R2000 RISC architecture," Prentice Hall, 1987.

[21] 홍인식, 정성호, 이승호, 임인칠, "32-bit RISC 프로세서의 제어부의 설계,"대한전자공학회 회계학술 발표회, 1989년 7월.

[22] 홍인식, 양동준, 김대영, 정성호, 이승호, 임인칠, "32-bit RISC CPU의 shifter설계," '88 특정 연구 결과 발표회, 1989년 7월.

[23] 김은성, 홍인식, 정준모, 김학림, 이주상, 유광석, 정성호, 임인칠, "RISC 프로세서의 data path설계에 관한 연구," 한국전자통신연구소 최종보고서, 1988년 5월.

[24] 김은성, 박동규, 유광석, 정성호, 임인칠, "캐쉬 메모리의 performance evaluation에 관한 연구," 한국전자통신연구소 최종보고서, 1989년 4월.

著 者 紹 介

洪 仁 植 (正會員) 第27卷 第7號 參照
현재 한양대학교 전자공학과 박사과정

林 寅 七 (正會員) 第27卷 第7號 參照
현재 한양대학교 전자공학과 교수