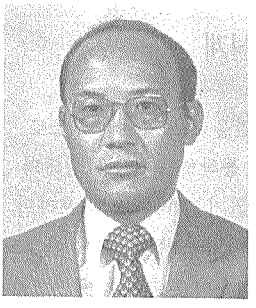
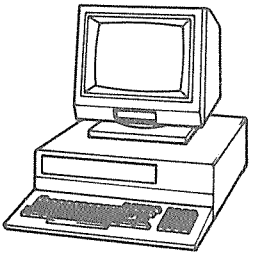


吳 吉 祿
韓國電子通信研究所
컴퓨터기술연구단장

HARP의 파이프라인 設計



Ⅰ. 서 론

파이프라인은 하나의 연산과정(computational process)을 여러 부분으로 나누고 각 연산 과정을 여러 다른 하드웨어에 할당하여 동시에 수행시키는 것을 의미한다. 이는 처리 속도를 높일뿐만 아니라 유한한 자원을 쉬지않고 사용할 수 있으므로 생산성을 높일 수 있는 장점을 가지고 있다.

파이프라인을 사용한 컴퓨터 구조는 1960년대 이래 수퍼 컴퓨터 시스템의 구현 과정에서 사용되어 왔다. 그러나, 근래에는 반도체 기술의 급격한 발전으로 마이크로 프로세서를 이용하여 수퍼미니급 컴퓨터의 성능을 얻을 수 있게 되었으며, 이들의 대부분은 RISC(Reduced Instruction Set Computer) 형태의 프로세서로 한 사이클(cycle)에 한 명령어의 수행이 가능하도록 파이프라인 컴퓨터 구조를 가지고 있다.

1980년대 이후에 만들어진 프로세서 중에서 CPU 외부에 캐쉬(cache)를 가지고 있는 대부분의 프로세서들은 3 또는 4 단계의 파이프라인 구조를 가지고 있다. Berkeley 대학에서 설계된 RISC Ⅱ는 기본적으로 명령어 페치, 명령어 수행 및 결과 저장의 3단계 파이프라인을 가지고 있으며[1], AM29000은 명령어 페치, 명령어 해석, 명령어 수행, 그리고 결과 저장의 4단계의 파이프라인을 가지고 있다.[2] MIPS의 경우 명령어 페치, 명령어 해석, 오퍼랜드, 페치, 명령어 수행, 그리고 결과 저장의 5단계로 되어 있으나, Code Compaction을 사용하여 2단계 마다 1개의 명령어가 수행된다.[3] 이들은 1클럭 사이클을 1파이프라인 단계로 사용하고 있으며 한클럭 사이클은 대부분 메모리 액세스 시간으로 결정된다. 그러나, 메모리를 액세스하는 시간이 CPU 내부의 한 단계에서 소요되는 시간보다 크기 때문에 명령어 페치 단계에서 병목 현상이 발생하는 단점이 있다. 액세스 시간이 빠른 메모리 소자를 사용하면 이를 부분적으로 제거할 수 있는 반면에 비용이 증가된다.

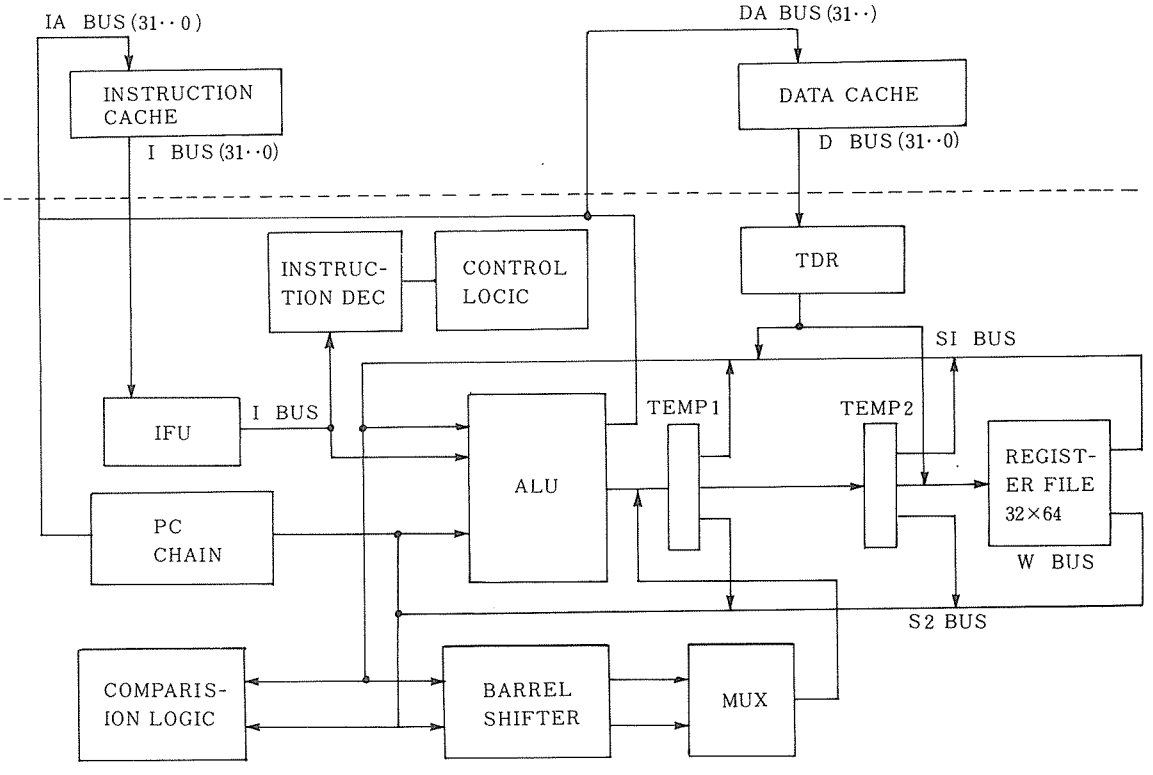
파이프라인으로 동작하는 프로세서는 명령어와 동시 수행으로 인하여 Timing Hazard와 Sequencing Hazard가 발생하게 된다. 대부분의 프로세서에서는 이러한 문제를 해결하기 위하여 Bypassing Logic을 사용하여 Internal Dataforwarding 방식을 이용하고, 분기(branch)의 경우에는 지연분기(delayed branch)와 예측분기(squashing branch) 중의 한가지 방법을 사용하고 있다.[1][4] 그러나 조건부분기(conditional branch)에서는 만족할 만한 결과를 얻지 못하고 있다.

HARP에서는[5] 이러한 단점들을 보완할 수 있도록 중첩된 메모리 액세스 방법을 사용하여 파이프라인을 설계하였다. 그리고 Codereorganizer의[6] 지원으로 성능을 향상시킬 수 있도록 파이프라인을 프로세서 구조 레벨에서 설계하였다.

본 논문의 Ⅱ장에서는 파이프라인의 설계에 관하여 설명하고 Ⅲ장에서는 파이프라인 동작으로 인하여 발생하는 자원 충돌을 해결하는 방법을 기술한다. 그리고 Ⅳ장에서는 명령어 수행도중 파이프라인 상에서 발생하는 익셉션 처리방법에 대하여 기술하며, Ⅴ장에서는 결론을 기술한다.

Ⅱ. 파이프라인 설계

Flynn은 명령어 세트만을 프로세서의 구조로 정의했으나[7], 파이프라인으로 동작하는 RISC형태의 프로세서에서는 효율을 증가시키기 위하여 파이프라인, MMU (Memory Management Unit) 등 구현에 관계되는 중요한 부분을 구조에 포함하여 프로세서 구조를 설계하



TEMP1 : TEMPorary register1
TEMP2 : TEMPorary register2
IFU : Instruction Fetch Unit
TDR : Temporary Data Register
SI BUS : Source 1BUS
W BUS : Write BUS

I BUS : Instruction BUS
IA BUS : Instruction Address Bus
D BUS : Data BUS
DA BUS : Data Address BUS
S2 BUS : Source2 BUS

〈그림 1〉 HARP의 마이크로 구조 블럭 다이어그램

는 것이 바람직하다.

프로세서 구현과 밀접한 관계가 있는 파이프라인을 설계하는데 중요한 요소는 크게 3가지로 생각할 수 있다. 첫번째는 파이프라인을 몇 단계로 구성할 것인가 하는 문제이다. 파이프라인 단계의 수를 줄이면 익셉션 처리 등에 편리하지만, 각 단계의 실행 시간이 길어지므로 단위 시간 당 명령어를 수행할 수 있는 양이 줄어들어 성능이 저하된다. 두번째는 각 단계에서 소요되는 시간을 결정하는 문제이다. 이것은 프로세서의 성능에 직접적인 영향을 미치는 것으로, 대부분의 다른 프로세서에서는 메모리 액세스 시간을 기준으로 파이프라인을 설계하여 프로세서의 성능이 메모리 액세스 시간에 의해 결정되었다. 세번째는 칩 구현의 난이도가 파이프라인 구조에 의해 많은 부분이 좌우되므로 각 파이프라인 단계에서 수행할 범위를 적당하게 나누어야 한다.

1. 파이프라인 단계 및 동작

HARP는 CPU, 명령어 캐쉬(I-HACAM)와 데이타 캐쉬(D-HACAM)의 3개의 칩으로 구성된다[5]. 〈그림 1〉은 HARP의 마이크로 구조의 블럭 다이어 그램으로 나타낸 것으로 제어 부분은 생략하였다.

HARP에서는 대부분의 명령어들이 수행되는데 5클럭 사이클이 소요되며, 파이프라인을 이용하므로 한 사이클 마다 하나의 명령어가 수행된다. 파이프라인은 IF(Instruction Fetch), ID(Instruction Decode), EX(EXecute), MA(Memory Acess) 및 WB(Write Back)의 5단계로 구성되며, 각 단계는 한 클럭 사이클이 소요된다. 그리고 한 클럭 사이클은 다시 PH1(PHase 1)과 PH2(PHase 2)로 나누어진다. 〈그림 2〉는 파이프라인의 단계와 각 단계에서의 동작을 요약한 것이다.

IF : PC에서 명령어의 어드레스를 I-HACAM으로 보낸다.
PC를 증가시킨다.
ID : 명령어를 IR에 저장한다.
명령어를 해석한다.
EX : 레지스터 파일을 액세스하고 오퍼랜드를 연산 유니트로 보낸다.
연산을 수행한다.
MA : Load/store, 분기 명령어의 경우 어드레스를 D-HACAM과 I-HACAM으로 각각 보낸다.
로직 연산의 결과를 임시 레지스터에 저장한다.
WB : Store의 경우 데이타를 메모리로 보낸다.
수행된 결과를 레지스터 파일에 저장한다

〈그림 2〉 파이프라인 단계와 기능

IF 단계는 프로세서가 수행할 명령어를 캐쉬(I-HACAM)로부터 가져오는 단계이다. PH1시간 동안에는 CPU에서 명령어 어드레스 버스를 통하여 명령어 어드레스를 명령어 캐쉬로 보낸다. 이때 CPU에서는 명령어 캐쉬로 명령어 페치를 요구하는 IREQ(Instruction REQuest) 신호를 동시에 보낸다. PH2 시간 동안에는 PC를 증가시켜 다음에 수행될 명령어 어드레스를 보낼 준비를 한다.

ID 단계는 메모리로부터 명령어를 받아 디코딩하는 단계이다. PH1 동안에는 명령어 캐쉬에서 보낸 데이타를 받아 IFU(Instruction Fetch Unit) 내에 있는 IR(Instruction Register) 또는 IFB(Instruction Fetch Buffer)에 저장한다. 캐쉬성공(cache hit)이면 작업을 계속 수행하며, 캐쉬실패(cache miss)이면 CPU는 수행하던 작업을 모두 중지하고 캐쉬가 필요한 명령어를 가져올 때까지 기다린다. 그러나 TLB(Translation Lookaside Buffer) 실패이면 익셉션이 수행된다. PH2 동안에는 명령어 레지스터에 있는 명령어를 해석하여 명령어 종류를 판별하고, 제어로직을 명령어의 종류에 따라 제어한다.

EX단계는 연산을 수행하는 단계이다. PH1 시간 동안에는 판별된 Opcode에 의해 EX단계의 PH2 단계에서 사용될 오퍼랜드를 레지스터파일이나 TEMP1 또는 TEMP2로부터 가져와 연산 유니트(ALU, comparision logic, barrel shifter)의 입력 래치(latch)에 저장한다. PH2 동안에는 명령어의 종류에 따라 연산을 수행하고 결과를 출력 래치에 저장한다.

MA 단계는 데이타 캐쉬를 액세스하는 Load/store명령어를 위해 존재한다. PH1 동안에 Load/store 명령어인 경우에는 데이타 어드레스를 데이타 캐쉬로 보내고, 분기 명령어에서는 어드레스를 명령어 캐쉬로 보냄과 동시에 이 값을 BTB(Branch Target address Blffer)에 보내 저장한다. 이외의 명령어에서는 출력 래쉬에 있는 값을 TEMP1에 저장한다. 분기의 경우에는 PH2 동안에 BTB에 있는 어드레스를 1증가 시킨다.

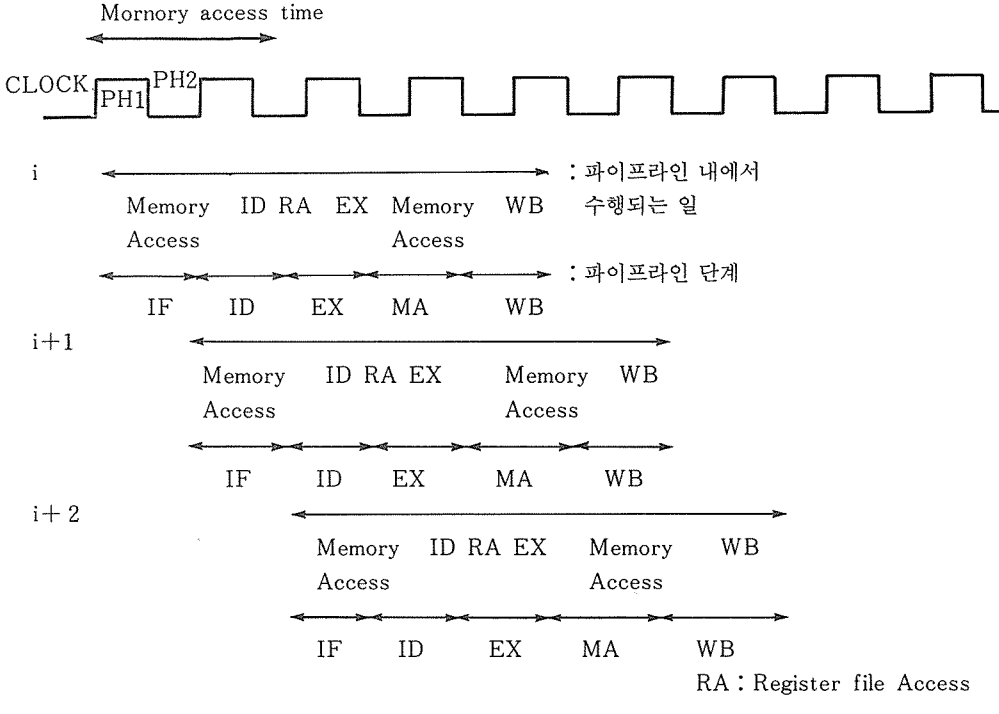
WB 단계는 명령어의 수행을 마치고 연산된 결과를 레지스터 파일에 저장하는 단계이다. PH1시간에 Load

명령어의 경우에는 데이타를 캐쉬로부터 받아 데이타 임시 레지스터(TDR)에 저장한다. 이 데이타가 다른 명령어의 EX단계에서 사용되면 연산 유니트의 입력 래취에 저장된다. Store 명령어에서는 메모리에 저장된 데이타를 캐쉬로 보낸다. 캐쉬 실패가 발생하면 프로세서는 수행을 중지하고 기다리며, TLB실패 등이 발생하면 익셉션을 처리한다. 분기 명령어의 경우에는 BTB의 내용을 명령어 캐쉬로 보냄과 동시에 PC에 기록한다. 이외의 명령어에서는 TEMP1에 있는 내용을 TEMP2로 이동한다. PH2 동안에는 TEMP2에 있는 값을 레지스터 파일에 저장한다.

2. 명령어 페치 방법

파이프라인으로 동작하는 프로세서에서 큰 장점 중의 하나는 1사이클에 1개의 명령어가 수행되는 것으로, 이를 위해서는 1사이클에 1개의 명령어가 CPU에 공급되어야 한다. 그러나 내부에 명령어 캐쉬가 없는 대부분의 프로세서에서는 외부 명령어 캐쉬로부터 명령어를 가져오는데 너무 긴 시간이 소요된다. 메모리 액세스에 소요되는 시간은 CPU에서 어드레스를 보내는 시간, 메모리 칩 내에서 어드레스를 받아 필요한 데이타를 찾는 시간과 메모리에서 CPU로 데이타를 보내는 시간으로 나누어질 수 있다. 그런데 CMOS(Complementary Metal Oxide Semiconductor)로 프로세서를 구현할 경우 부하 구동 능력(load driving capability) 이 약하기 때문에 두 칩 사이에서 데이타의 전달 시간이 길어지게 된다. 그러므로 명령어를 프로세서에 공급할 수 있는 용량이 프로세서의 처리능력에 미치지 못하여 병목현상을 일으킬 수도 있다. 그리고 HARP에서는 캐쉬의 크기를 증가시키기 위해 HACAM 수를 늘릴 가능성이 있으므로 HACAM를 선택하는데 시간이 더 필요하게 된다.

HARP에서는 이러한 단점을 보완하기 위하여 〈그림 3〉과 같이 메모리 액세스 시간을 CPU에서 어드레스를 보내 HACAM을 선택하는 시간과 HACAM에서 데이타를 찾아 CPU로 보내는 시간으로 나누었으며, 기존의 프로세서보다 단위시간에 더 많은 명령어를 CPU에 공급할 수 있도록 중첩된 메모리 액세스 방법을 사용하였다. HARP는 어드레스와 데이타 버스가 분리되어 있으므로 중첩된 메모리 액세스로 인한 버스 충돌은 발



〈그림 3〉 중첩된 메모리 액세스 방법

생하지 않는다. 따라서 다른 프로세서의 명령어 페치에서 발생하는 병목현상을 줄일 수 있다. 그리고 범용의 메모리 소자를 캐쉬로 사용한 프로세서에서는 중첩된 메모리 액세스 방법을 사용하기 위하여 약간의 하드웨어가 필요하나, HARP에서는 HACAM이 커스텀 칩으로 구현되므로 중첩된 메모리 액세스를 구현하는데 어려움이 따르지 않는다.

Ⅲ. 파이프라인의 성능 개선

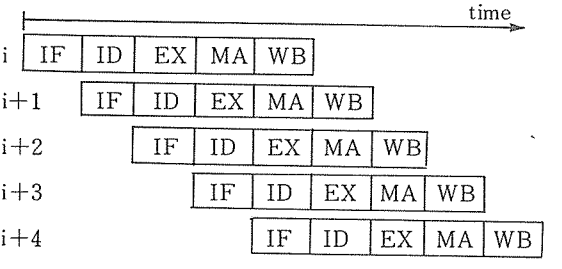
파이프라인으로 동작하는 프로세서는 명령어의 동시 수행으로 Timing Hazard의 Sequencing Hazard가 발생하게 된다[6]. HARP에서는 이러한 자원 충돌을 해결하기 위하여 하드웨어와 소프트웨어를 동시에 사용한다.

1. Timing Hazard의 해결 방법

메모리를 액세스할 때 발생하는 충돌과 오퍼랜드를 가져오는 데 걸리는 지연 시간은 프로세서의 성능을 결정하는 중요한 요소이며, 이는 버스 구조와 밀접한 관계를 가지고 있다. 어떤 프로세서에서는 메모리 충돌을 피하기 위하여 지연 시간을 두어 명령어를 수행하는데[8], 이러한 경우는 버스를 공유하기 때문에 발생하는 것으로 프로세서의 성능을 저하시킨다. 그리고 캐쉬가 명령어와 데이타에 공통으로 사용될 경우 명령어와 데이터가 충돌이 생기면 시분할을 하여야 한다. 따라서 프로세서는 명령어 수행을 중지하고 오퍼랜드가 모두 도착할 때까지 기다려야 하므로 프로세서의 성능이 심각하게 저하된다.

이러한 단점을 제거하기 위하여 HARP에서는 명령어 캐쉬와 데이타 캐쉬를 분리하여 사용하고 있으며, CPU는 하바드 버스 구조, 즉 명령어, 명령어 어드레스, 데이타 및 데이타 어드레스 버스를 가지고 있다. 따라서 데이타의 경우에도 데이타의 상호관련(data dependency)이 없으면 지연시간 없이 Load Store 명령어를 연속으로 사용할 수 있으므로 Code-Reorganizer의 부담을 줄일 수 있으며, 익셉션이나 콘텍스트 스위치 등에서 성능을 향상시킬 수 있다.

데이타의 상호관련 때문에 발생하는 명령어 수행의 지연은 현재 수행되고 있는 명령어가 이전 명령어의 수행결과를 이용하여야 하나 이전 명령어의 수행이 끝나지 않아 수행결과를 이용할 수 없을 경우에 발생하게 된다. 〈그림 4〉에서 명령어 i가 Load 명령어이면, 그 명령어의 수행 결과는 명령어 i의 WB 단계를 지나서 이용될 수 있다. 즉, Load 명령어 수행 결과를 이용할 수 있는 첫번째 명령어는 1÷3이다. 따라서 Load 명령어의 결과를 ALU 연산에 이용하기 위해서는 2명령어의 지연이 생기게 된다.



〈그림 4〉 명령어 수행 순서

HARP에서는 데이타 상호관련 때문에 발생하는 지연구간을 줄이기 위하여 Source1, Source 2 레지스터에 Internal Data Forwarding Logic을 사용한다. Load 명령어의 결과는 WB의 PH1 끝에서 CPU에 도달하므로 이 데이타를 레지스터 파일에 저장하지 않고 직접 EX 단계의 PH2에서 이용한다. 이를 위해서는 약간의 하드웨어 로직이 필요하며, 이를 이용하여 1개의 지연구간을 줄일 수 있다. 그리고, 나머지 1개의 지연구간에는 간단하게 NOP(No OPeration)을 사용하여 채울 수도 있으나, Code-Reorganizer가 Load 명령어 전후에서 데이타 상호 관련이 없는 명령어를 선택하여 사용함으로써 NOP의 사용빈도를 최소화할 수 있으며, 이 구간을 중첩 구간이라 부른다.

HARP는 레지스터 파일을 액세스할 때 충돌을 방지할 수 있도록 칩 면적을 고려하여 2Read, 1Write 포트(port)로 구성된 레지스터 파일을 가지고 있다. 따라서 동시에 레지스터 파일에 2Read, 1Write를 할 수 있고, 액세스 시간을 줄일 수 있도록 포트에 대해 각각의 디코딩 로직을 사용하여 디코드 단계가 Critical Path가 되지 않도록 한다.

2. 분기처리

파이프라인으로 동작하는 프로세서에서 가장 다루기 어렵고 프로세서의 성능을 저하시키는 부분이 분기와 익

셉션이다. 익셉션은 Ⅳ장에서 언급하기로 하고 여기서는 분기 처리에 관하여 기술 한다.

〈그림 4〉에서 i가 분기 명령어이면 분기할 어드레스는 EX 단계의 끝에서 얻어지며, MA 단계에서 분기 어드레스가 I-HACAM으로 보내진다. 따라서 i명령어 다음에 있는 i+1, i+2 두 개의 명령어는 수행되지 않아야 한다. 그러나 이 구간에 있는 명령어를 Flushing 시키는 것은 프로세서의 효율을 감소시키므로 파이프라인을 이용한 프로세서에서는 대부분 지연분기를 사용한다. 즉, 데이타 상호관련이 없는 명령어를 기본 블럭 내에서 찾아 수행 순서를 바꿔 이 구간에 수행함으로써 프로세서가 쉬는 시간을 가능한 한 줄인다. 그러나 이 구간에 들어갈 수 있는 명령어를 찾을 수 있는 확률이 첫번째 구간에서 약 70%이고[9], 두번째 구간에서는 더 줄어들므로 MIPS-X의 경우에는 예측분기를 사용한다[7]. 이 경우 분기가 성공하면 이 구간에 있는 명령어는 모두 수행되므로 쉬는 시간이 없으나 분기가 실패하면 2개의 명령어를 Flushing 시켜야 한다.

HARP에서는 이러한 단점을 보완하기 위하여 지연분기와 예측분기를 동시에 동적으로 사용하여 NOP 의 사용빈도를 최소화한다.

즉, 분기 명령어 바로 다음에 오는 1개의 명령어는 지역구간에 사용되며, 그 다음에 오는 1개의 명령어는 예측구간에 사용된다. 지연구간에는 분기 명령어 앞에 있는 명령어 중에서 분기 명령어와 데이타 관련성이 없는 명령어로 수행순서를 바꿔 분기 명령어 다음에 오게 한다. 예측구간에는 분기할 어드레스에 있는 명령어가 들어가게 된다. 그런데 지연구간에 들어갈 수 있는 명령어를 찾을 수 있는 확률이 약 70%에 불과하므로 조건부 명령어 포맷에 Delayed Bit를[6] 두어 지연구간에 사용될 수 있는 명령어가 없으면 Code-Reorganizer가 지연구간을 예측구간으로 바꿔 NOP의 사용을 줄이도록 한다. 이러한 방법은 기존의 프로세서 보다 성능을 향상시킬 수 있다. 〈그림 5〉는 분기 명령어가 있을 경우 Code-Reorganizer가 작업 순서를 변화시키 [illegible] 나타낸 것이다.

HARP에서는 분기 조건이 맞아 분기할 경우에는 쉬는 파이프 단계가 없으며, 분기하지 않을때에는 예측구간에 있는 명령어는 NOP으로 처리된다.

무조건 분기 명령어의 경우에는 반드시 분기를 하므로, 분기 어드레서 명령어를 가져오는 동안 분기 명령어

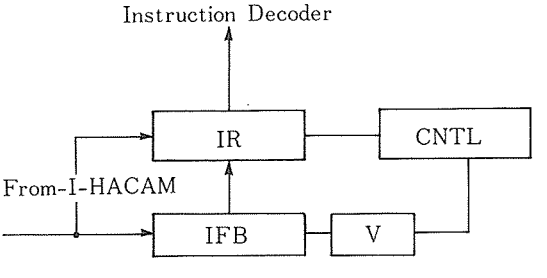
origigiinal code	roorganized code
100 inst, A	inst. A
inst. B	100 inst. B
inst. -C	inst. C
inst. D	inst. D
.	.
.	.
.	.
inst. E	inst. E
inst. F	inst. G
inst. G	inst. BR 100
inst. B R 100	inst. F → 지연구간
inst. H	inst. A ← 예측구간
inst. I	inst. H
	inst. I

〈그림 5〉 분기 처리방법

다음의 2개의 슬롯에 있는 명령어는 반드시 수행되며, 예측구간만 존재한다. 이러한 명령어에는 Delayed Bit이 필요하지 않으며 쉬는 시간없이 분기 명령어를 수행할 수 있다.

3. Instruction Fetch Unit(IFU)

HARP에서는 분기가 발생했을 경우 파이프라인이 쉬지않고 계속 동작할 수 있도록 지연분기와 예측분기를 동시에 사용한다. 따라서 분기가 실패하면 예측구간에 있는 1개 혹은 2개의 명령어가 Flushing되어야 한다. HARP에는 분기 명령어 다음의 예측구간에 있는 명령어의 Flushing를 가능한 한 방지할 수 있도록 〈그림 6〉과 같은 IFU가 있다.

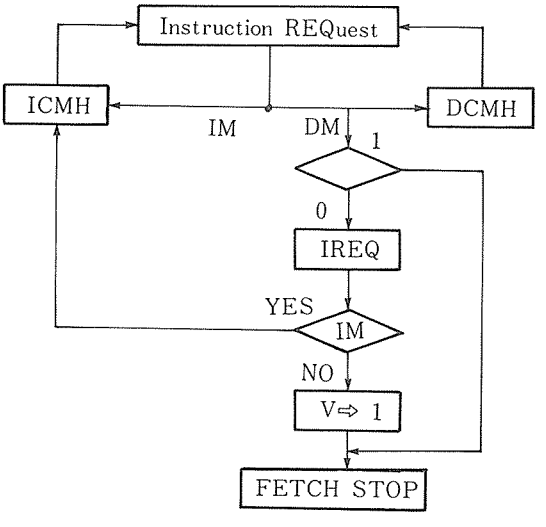


〈그림 6〉 Instruction Fetch Unit

IFU는 1워드 명령어를 저장할 수 있는 IFB(Instruction Fetch Buffer), IFB의 상태를 나타내는 Valid Bit(V), IR 및 IFU를 제어하는 로직으로 구성된다. CNTL블럭은 Valid Bit와 분기 조건의 연산결과를 이용하여 명령어 순서를 재배치한다. 프로그램이 처음 시

작되었을 때는 IFB는 비어있다. 이 경우 IFB를 채우기 위하여 프로세서의 수행을 중지하는 것은 효과적이지 못하므로 IFB가 비어 있는 상태에서는 I-HACAM으로부터 들어온 명령어는 IFB를 거치지 않고 직접 IR로 이동되어 수행되도록 CNTL회로를 설계하였다.

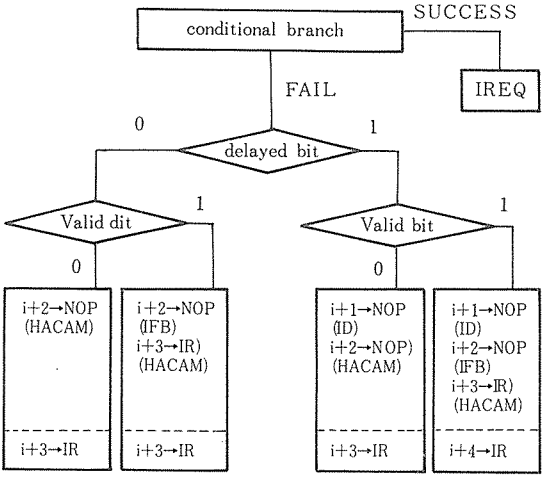
〈그림 7〉은 IFB를 채우는 과정을 나타낸 것이다. 즉, IFB는 데이타 캐쉬실패가 발생하여 오퍼랜드를 기다리는 동안 명령어가 수행되지 않을 때 채워지게 되며, Valid Bit이 1로 변환된다. 이때는 CPU에서 IREQ 신호를 보내지 않으므로 버퍼에 빈자리가 있을 때까지 Prefetch를 중단한다. 프로세서가 수행을 다시 개시하면 IFB에 있는 명령어가 IR로 이동되고 I-HACAM에서 Prefetch된 명령어는 IFB에 저장된다.



IM : Instruction cache Miss
DM : Data cache Miss
ICHM : Instruction Cache Miss Handler
DCMH : Data cache Miss Handler
V : Valid bit
IREQ : Instruction REQuest

〈그림 7〉 명령어 Prefetch방법

조건부 분기를 수행하는 경우 ID 단계에서 Delayed Bit와 Valid Bit를 검사한다. 분기 명령어 다음에 있는 명령어는 Delayed Bit가 0이면 지연구간에 존재하며 Delayed Bit가 1이면 예측구간에 존재한다. EX 단계에서 분기가 성공하면 Code-Reorganizer에서 배열된 순서대로 수행을 계속한다. 그러나 분기가 실패하면〈그림 8〉에서와 같이 Valid Bit가 1일 경우 IFB에는 예측구간에 있는 명령어가 존재하므로 이 명령어를 IR로 보내지 않고 Fulshing시킴과 동시에 I-HACAM에서 들어오는 명령어를 직접 IR로 보내 프로세서가 1사이클 쉬는 것을 방지할 수 있다.



〈그림 8〉 IFB 제어도

Katevenis에 의하면 프로그램에서 Load/Store 명령어가 약 40%, 분기가 약 30%이고 이 중에서 조건부 분기가 약 45%인 것으로 알려져 있다.[1] 그리고 8Kbyte 크기를 가진 2-Way Set Associative Unified Data Cache with 16 Byte Line에 대하여 약 10%의 캐쉬실패가 발생하는 것으로 알려져 있다[10]. HARP에서 D-HACAM은 8Kbyte 크기를 가지고 2-Way Set Associative Unified Data Cache With 16Byte Line으로 구성된다. 그러므로 49%의 Load/Store 명령어 중에서 10%의 데이타 캐쉬 실패를 예상할 수 있으므로 이때 IFB가 채워지게 되며, 이 확률은 약 4%로 예상된다. 따라서 지연분기, 완측분기 및 IFU를 사용하는 HARP에서는 분기가 실패할 경우 예측구간에 있는 2개의 명령어가 Flushing 되는 것을 방치할 수 있으므로 예측분기만을 사용하는 분기처리 방법보다 성능을 개선할 수 있다.

Ⅳ. 익셉션 처리 방법

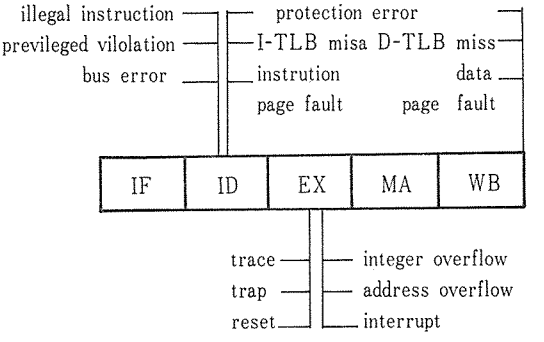
익셉션은 CPU 내부의 비정상적인 상태나 외부의 요

구(external events), 또는 특정 명령어들로 인하여 현재 수행되고 있는 프로세스의 프로그램을 정상적으로 수행할 수 없는 상태를 말한다. 익셉션이 발생하면 프로세서는 현재 수행하고 있는 프로세스의 수행을 일단 멈추고 어떤 특정한 순서에 의해 그 익셉션 요인을 처리하고, 진행이 중단된 프로세스를 다시 정상적으로 수행시키는 일련의 작업을 수행하여야 한다.

익셉션을 처리하는 방법은 Instruction Continuation 방법과 Instruction Restart 방법이 있다. Instruction Continuation 방법은 MC68010과 MC68020에서 사용하고 있으며[11][12], 익셉션이 발생하면 그 당시 프로세서의 상태를 그대로 Micro-Instruction Level에서 보전하고, 그 요인에 대한 처리를 완료한 후 보전되었던 프로세서의 상태를 복원하여 계속 수행하는 방법이다. 이 방법은 익셉션이 발생했을 때의 프로세서의 상태를 정확하게 보전해야 하므로 보전될 정보의 양이 대단히 많다. Instruction Restart 방법은 Machine Instruction Level에서 프로세서의 상태를 보전했다가 다시 저장된 명령어를 수행하는 방법으로 첫번째 방법에 비해 프로세서의 상태에 대하여 저장할 정보의 양이 적으며 덜 복잡하다. 최근에 발표된 RISC 형태의 대부분의 프로세서들은 (MIPS, AM29000 등) 후자의 방법을 사용하고 있다. 그러나, Instruction Restart 방법은 Machine Instruction의 재수행을 위해 프로세서의 상태를 익셉션이 발생하기 전의 상태로 복원하여야 한다.

HARP에서는 5단계의 파이프라인을 통해 한 사이클에 한 명령어를 수행하여 동시에 5개의 명령어를 처리할 수 있다. 따라서, 익셉션이 발생했을 때 파이프라인 각 단계의 상태를 포함하여 프로세서의 완벽한 상태 보전이 어려우며, 익셉션의 처리가 복잡하다. 이를 피하기 위해 HARP에서는 Instruction Restart 방법을 사용하여 익셉션을 처리하며, 명령어의 재수행을 위해 익셉션이 발생한 명령어 이전의 상대는 파이프라인의 마지막 단계에서 명령어의 수행 결과를 처리하도록 함으로써 보전한다.

〈그림 9〉는 파이프라인의 각 단계에서 발생할 수 있는 모든 익셉션을 나타내며, 이는 익셉션이 발생했음을 CPU에서 아는 시점을 기준으로 하였다. HARP에서 익셉션은 그들의 종류에 따라 눠르게 처리할 수 있으나, 기본적으로 두 부류로 나누어 처리된다. 그 하나는 익셉션 처리 후 익셉션이 발생한 명령어를 다시 수행하는



〈그림 9〉 파이프라인 단계와 익셉션

부류로, ID 그리고 WB 단계에서 발생하는 익셉션이다. 다른 하나는 EX 단계에서 발생하는 익셉션으로 처리후 익셉션이 발생한 명령어를 다시 수행하지 않고, 그 다음 명령어부터 다시 수행하는 부류이다.

가. ID와 WB 단계에서 발생하는 익셉션 처리

ID와 WB 단계에서 익셉션이 발생하면, 현재 파이프라인 각 단계에서 수행되고 있던 명령어들의 수행은 중단되고 익셉션이 처리된다. 그리고 멈추었던 프로세스의 재수행은 수행이 중단된 명령어들을 다시 시작함으로써 이루어지며, 아래와 같은 방법으로 익셉션은 처리된다.

1) 현재 수행하고 있는 프로세스의 수행과 PC Chain의 Shifting을 일단 멈추고, 파이프라인을 동결시킨다.
2) Vector Number를 결정한다. 이때 Vector Number는 익셉션의 요인에 따라 내부 로직이나 외부의 로직에 의해 생성되며, Priority를 나타낸다. Priority는 Vector Number가 낮을수록 높으며, Vector Number와 익셉션의 종류는 〈表 1〉과 같다. 단, Trace는 가장 낮은 Priority를 갖는다.
3) SSW (System Status Word)의 Trap Enable 비트 TE가 Reset되어 있거나, PSW의 익셉션 레벨 필드 EL의 값과 Vector Number를 비교하여 낮은 Priority의 익셉션을 Pending시키고 일시 중단된 프로세스의 수행을 계속한다. 그러나, SSW의 TE가 Set되어 있고, 높은 Priority의 익셉션이면 4)~7)과 같이 처리된다.
4) SSW와 PSW (Program Status Word)를 각각

〈표 1〉 Vector number와 익셉션

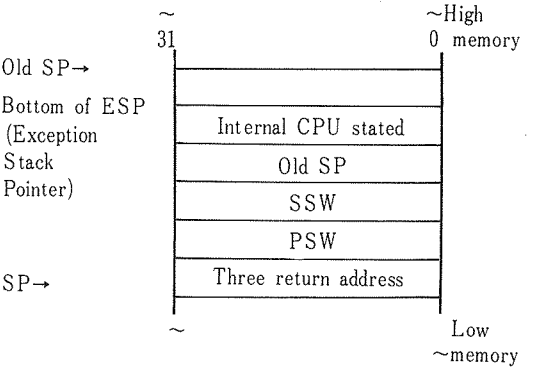
vector number	description
0	reset
1	bus error
2	D-TLB miss
3	data page fault
4	I-TLB miss
5	instruction page fault
6	protection error
7	
:	reserved
15	
16	address overflow
17	integer overflow
18	illegal instruction
19	privilege violation
20	trace
:	reserved
32	TRAP 0
:	:
63	TRAP 31
:	:
128	reserved
:	:
143	prioritized interrupt

SSW′(익셉션 SSW)와 PSW′(익셉션 PSW)에 복사하고, SSW의 User 비트 U를 Reset 시켜 Supervisor Mode로 변환한다. 또한 SSW의 TE를 Clear시켜 Restart에 필요한 정보를 익셉션 스택 프레임에 저장할 동안 다른 익셉션이 발생하지 못하게 한다.

5) PSW의 익셉션 레벨 필드 EL에 Vector Number를 Write시키고, Trace 비트 T를 Clear시켜 익셉션 처리 루틴이 Tracing없이 수행되도록 한다.

6) 익셉션 Vector Number에 Vector Entry Size를 곱하여 익셉션 처리 루틴의 초기 어드레스를 얻고 그곳으로 제어를 넘겨 익셉션 처리 루틴을 시작한다. 이때 파이프라인에서 수행되고 있던 명령어들을 Squashing 시킨다. 익셉션 처리루틴은 SSW′와 PSW를 저장하는 일로부터 시작되며, 또 PC Chain으로부터 Return Address로 사용될 세 명령어의 PC 값도 "PUSHSR"명령에 의해 익셉션 스택 프레임에 저장된다. 이들 세 명령어의 PC 값은 EX, MA, 그리고 WB 단계에서 수행되고 있던 명령어들의 PC값들로, PC Chain의 PC3, PC4와 PC5의 값이다. 익셉션 스택 프레임에 저장된 정보는 〈그림 10〉과 같다.

7) PC 체인의 Shifting을 Enable 시키고, SSW의 TE 비트를 Set시켜 다른 익셉션의 발생을 허용하고, 본격적인 익셉션 처리를 수행한다.



〈그림 10〉 익셉션 스택 프레임

익셉션의 처리가 모두 끝나면 익셉션 처리를 위하여 중단되었던 프로세스의 수행을 다시 시작하여야 한다. HARP에서는 이를 위한 명령어를 따로 두지 않았으며, 익셉션 처리 루틴 내에서 일련의 명령어들을 수행함으로써 저장되었던 정보를 복원하고 중단되었던 프로세스가 다시 수행되도록 한다. 일련의 작업은 다음과 같다.

1) Pending된 익셉션을 조사하고 있으면 그 중 가장 높은 Priority를 갖는 익셉션을 위에서 기술한 방법으로 처리한다.

2) Pending된 익셉션이 없으면, PC 체인의 Shifting을 멈추고 SSW의 TE를 Clear시켜 저장된 정보를 복원할 동안 다른 익셉션을 허용하지 않는다.

3) 저장된 세 개의 Return Address를 "POPSR"명령을 이용해 RARs(Return Address Register)에 저장시키고, PSW와 SSW를 PSW′와 SSW′에 각각 저장한다.

4) RTU 명령어를 세번 수행함으로써 중단되었던 프로세스는 수행이 다시 시작되며, 이때 PSW′와 SSW′를 PSW와 SSW에 각각 복원한다.

나. EX 단계에서 발생하는 익셉션 처리

EX 단계에서 발생한 익셉션은, 익셉션이 발생한 명

령어 이전의 MA와 WB 단계에서 수행되고 있던 명령어들의 수행을 완료한 후 처리된다. 이때 또 다른 익셉션이 MA와 WB 단계의 명령어들을 수행하면서 발생할 수 있으며, 다른 익셉션이 발생하면 전에 발생한 익셉션은 무시되고 나중에 발생한 익셉션을 "가" 항에서 기술한 방법으로 처리한다. 다른 익셉션의 발생이 없으면 아래와 같이 처리된다.

1) MA와 WB 단계에 있는 명령어의 수행을 완료한다. 이때 다른 익셉션이 발생하면 그것을 "가" 의 1)~3)에서 기술한 방법으로 처리 한다.
2) 현재 수행되고 있는 프로세서의 수행과 PC Chain의 Shifting을 멈춘다.
3) Vector Number를 결정한다.
4) 나머지 처리 방법은 "가"항의 4)~7)에서 기술한 것과 동일하다.

V. 결 론

파이프라인은 프로세서의 구조설계와 반도체의 구현 모두에 아주 밀접하 관계를 가지고 있다. 한국전자통신연구소에서 개발하고 있는 RIRC 형태의 32비트 프로세서인 HARP에서는 구조 설계에서부터 파이프라인을 고려하여 짧은 시간에 반도체 구현이 가능하도록 명령어 세트를 설계하였다.

HARP의 파이프라인은 기존의 프로세서에서 명령어 페치시 발생하는 단점을 보완할 수 있도록 중첩된 메모리 액세스 방법을 사용하며, 이는 IF, ID, EX, MA 및 WB의 5단계로 구성된다. 또한 파이프라인 동작 시에 발생하는 자원충돌을 해결하기 위하여 지연분기와 예측분기를 동시에 동적으로 사용하며, Code-Reorganizer로도 해결할 수 없는 조건부 분기의 예측구간에 대해서는 IFU를 사용하여 NOP이 최소로 사용되도록 성능을 개선하였다. 그러나 지연구간에 들어갈 수 있는 명령어가 역어 지연구간이 예측구간으로 사용되고 분기가 실패할 경우에는 예측구간만을 사용한 경우와 같이 2개의 명령어를 Flushing시켜야 한다.

명령어 수행시 파이프라인 상에서 발생하는 익셉션에 대해서는 익셉션 처리의 복잡성을 피하기 위하여 Instruction Restart 방법을 사용하며, ID와 WB에서 발생하는 익셉션과 EX단계에서 발생한는 익셉션으로 나누어서 처리한다.

〈參 考 文 獻〉

1. Manolis G. H. Katevenis, Reduced Instruction Set Computers for VLSI, The MIT Press, 1984.
2. AMP, AM29000 User's Manual, 1987.
3. Steben A. Przybylski, et al., Organization and VLSI Implementation of MIPS, Stanford University TR84-259, April 1984.
4. Mark Horowitz, Paul Chow, et al., "MIPS-X : A 20-MIPS Peak, 32-bit Microprocessor -or with On-Chip Cache," IEEE Journal of Solid-state Circuits, Vol, SC-22, No. 5, Oct. 1987, pp. 790-799.
5. 김강철, 박종원, 이장선, 이만재, "HARP (High performance Architecture for Risc-type Processor) 의 구조 설계," 전자통신, 제10권, 제3호, 1988. 10.
6. 강익태, 김강철, HARP code reorganizer, 한국전자통신연구소 Technical Memo, 1988. 3.
7. Flynn, M. J. "Some Computer Organization and Their Effectiveness," IEEE Trans. on Computer, C-21, No. 9, Sept, 1972, pp. 948-960.
8. Sun Microsystems, Inc., A RISC Tutorial, 1987.
9. Scott Mcfarling, John Hennessy, "Reducing the Cost of Branches," The 13th Annual International Symposium on Computer Arch., 1986, pp. 396-403-
10. Fairchild, CLIPPER 32-Bit Microprocessor User's Manual, Prentice Hall, 1987.
11. MacGregor, D., Mothersole, D., Moyer, B., "The Motorola MC68020," IEEE Micro, Vol. 4, 4, No. 4, Aug. 1984, pp. 101-118.
12. MacGregor, D., Mothersole, D. S., "Virtual Memory and the MC 68020," IEEE Micro, V Vol. 3, No. 3, June 1983, pp. 24-38.