論文 90-27-7-5

RISC프로세서 프로그램 카운터 부(PCU)의 설계

(The Design of A Program Counter Unit 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)형 고성능 프로세서의 파이프라인 아키텍춰상에서 명령어 어드레스를 기억장치(명령어 캐쉬)에 효율적으로 제공할 수 있는 프로그램 카운터 부를 제안한다.

프로그램 카운터 부는 RISC프로세서에 요구되는 빠른 동작 속도에 부합되도록 명령의 어드레스를 계속적으로 계산하여야 하므로 외부 기능 블럭(ALU 또는 전용 Adder)을 이용하는 방식보다 자체 인크리먼터(Incerementor)를 이용하는 것이 그 속도면에서 효율적이다.[5][8]

제안한 프로그램 카운터 부는 캐리 전파 지연(Carry Propagation Delay)을 갖지 않는 빠른 속도의 +4(Byte Address) 연산 기능 인크리먼터를 갖도록 설계하며 동작에 대한 정확한 사양을 얻기 위하여 타겟 프로세서의 전체 데이타 패스에 대한 각 명령별 동작을 파이프라인 스테이지의 Phase별로 분석하여 설계한다.

전체 프로그램 카운터 부의 회로에는 저소비 전력으로 적은 레이아웃 면적에 빠른 동작 속도를 갖는 CMOS와 Wired Logic 회로기술을 사용하며, 회로의 시뮬레이션은 Apollo W/S 상에서 Mentor Graphics CAD 툴들을 사용하여 수행한다.

Abstract

This paper proposes a program counter unit(PCU)on the pipelined architecture of RISC(Reduced Instruction Set Computer) type high performance processors, PCU is used for supplying instruction addresses to memory units (Instruction Cache) efficiently.

A RISC processor's PCU has to compute the instruction address within required intervals continnously. So, using the method of self-generated incrementor, is more efficient than the conventional one's using ALU or private adder.

The proposed PCU is designed to have the fast +4 (Byte Address) operation incrementor that has no carry propagation delay. Design specifications are taken by analyzing the whole data path operation of target processor's default and exceptional mode instructions. CMOS and wired logic circuit technologies are used in PCU for the fast operation which has small layout area and power dissipation

The schematic capture and logic, timing simulation of proposed PCU are performed on Apollo W/S 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 5月 14日

I. 서 론

컴퓨터 아키텍춰의 구조 및 반도체 기술의 발달은 기판(Board)이 아닌 단일칩으로 구성되는 메가프로

세서의 단계에 까지 와 있으며, 탁상용 컴퓨터(Desktop Computer) 및 워크스테이션(Workstation)으로도 슈퍼 미니급 컴퓨터가 보유하는 처리능력을 가능케 하고 있다.[5][12]

최근 성능 향상 및 설계 비용 감소를 꾀하는 경향의 아키텍춰로서 인정받고 있는 RISC는 모든 명령어에 동일한 크기와 고정된 형식(Fixed Format)을 사용하여 대부분의 명령이 일정한 데이타 패스를 거쳐 처리되어진다. 따라서, RISC는 단일 싸이클 동작이 가능하며, 많은 스테이지를 갖는 강력한 파이프라인 아키텍춰로 실현하여 높은 성능을 나타낼 수 있다.[5][7]

이러한 고성능의 파이프라인을 유지하기 위하여는 정상동작(Default)시 매 스테이지 마다 페치될 명령어의 어드레스(Address)를 지속적으로 계산해내고 하나의 머신 싸이클내에서 수행되어지는 명령들의 어드레스를 모두 저장하여, 예외(Exception) 발생시 레지스터 파일(Register File)에 대피 시켰다가 예외처리 후 파이프라인을 복구/수행 하는데 필요한 일련의 체인으로 구성되는 다기능 프로그램 카운터 부(PCU)의 지원이 필수적이다.[4][5]

기존의 PCU에서는 다음에 수행될 명령어의 어드레스를 계산하기 위하여 ALU를 사용하거나 전용 덧셈기(Adder)등의 외부 기능 블럭을 사용하여 왔다. 그러나, 최근 상용화된 RISC 칩들의 경우와 같이 10nsec이내에 어드레스 인크리먼트 동작을 완료해야 될 경우 자체 인크리먼터의 사용은 필수적이다.

본 논문에서는 타겟 프로세서인 HyRISC[11][14]의 데이타 패스 동작과 일치하는 프로그램 카운터 부를 제안하며 그 중 핵심부분인 인크리먼터 부에는 캐리 전파 지연을 유발하지 않아 빠른 속도로 동작할 수 있는 회로 방식을 제안하고 설계한다. 설계 사양을 얻기 위하여 각 명령어에 대한 타겟 프로세서의 데이타 패스 동작을 파이프라인 스테이지의 Phase 별로 분석하며, 설계후 회로 레벨 시뮬레이션을 수행하여 논리와 타이밍에 있어 타겟 프로세서의 동작과 부합됨을 입증한다.

Ⅱ. 설계 배경

본 논문의 타겟 프로세서인 HyRISC는 고유의 명령어 세트를 갖는 4-스테이지 파이프라인 구조의 프로세서로써 데이타 패스는 그림 1과 같은 구조를 갖는다.

하드웨어(데이타 패스)의 설계시 그에 따른 컴파일러의 설계가 가장 큰 문제로 대두하게 되는데, Hy-

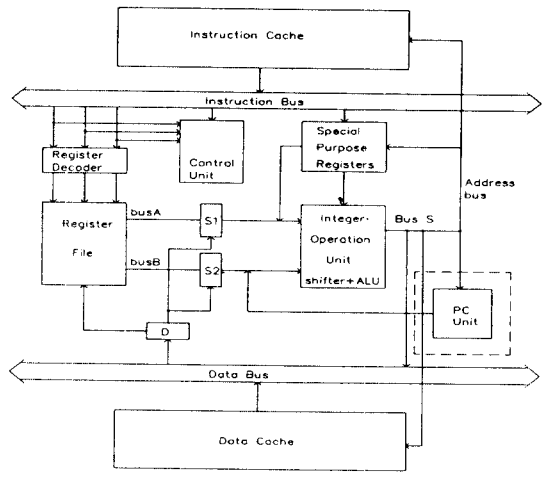


그림 1. HyRISC의 전체 블럭 다이아그램
Fig. 1. Block diagram of HyRISC.

RISC는 명령어의 형태와 데이타 패스상에서 레지스터 할당 그리고 예외(Exception) 발생시의 처리 방식들을 버클리 계보의 RISC 프로세서와 유사한 형태를 갖도록 설계하여 SPARC 스테이션(SUN 4/60)의 컴파일러를 일부 수정하여 이식가능 하도록 설계하였으며, SPARC스테이션 컴파일러에 관한연구는 진행 중에 있다.

수행되는 명령들은 모두 단일 머신 싸이클에 수행되어 빠른 속도의 처리가 가능하며, 그 종류는 메모리 엑세스를 위한 Load와 Store명령과, 논리적/산술적 계산을 하기 위한 명령으로 크게 나눌 수 있다.

데이타 패스 중 OP-코드를 입력으로 받아 기능 블럭의 기능을 통제하는 제어부는 명령어의 고정된 형식을 이용 하드와이어 콘트롤(Hardwired Control)로 실현되어 있으며 빠른 동작을 위하여 기능 블럭들의 대부분은 논리 게이트들이 하드와이어 로직(Hardwired-Logic) 기법으로 설계되어 있다.

그리고, 빠른 명령어 페치(Fetch)를 수행하기 위하여 데이타 패스와 동일한 칩(On-Chip) 상에 명령어 캐쉬 메모리를 가지고 있어 레지스터 파일 만을 사용하여 명령어 페치시 매번 Pad를 거쳐야 하는 초기 형태 프로세서[5]의 단점을 보완하고, 데이타 캐쉬 또한 동일 칩상에 설계하여 배열, 구조체와 같은 비 스칼라 변수들과 광역 스칼라 변수들을 저장할 수 있어 컴파일러의 부담을 줄일 수 있다. 그외에 데이타 캐쉬와 데이타 패스 사이에 고속의 버퍼 메모리로서 32-워드 크기의 레지스터 파일을 가지고 있으며 그 중 R0는 하드와이어드 제로로 되어 있다.

데이타를 처리하는 정수 처리부(Integer Operation Unit)는 5단 배럴 쉬프터와 그에 연결된 고속의 ALU로 구성되며 원활한 동작을 위해 한번의 동작으로 쉬프터와 ALU가 동작하도록 설계되어 있다.

1. 파이프 라인

HyRISC는 하드웨어 자원의 효율적인 이용을 위하여 4-스테이지 파이프라인으로 구성되며 그림 2와 같다.

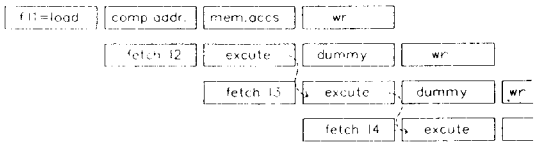


그림 2. 명령어 캐쉬를 이용한 4-스테이지 파이프라인
Fig. 2. 4-stage pipeline using instruction cache.

이는 각각 명령어 페치/디코드, 실행, 메모리 액세스, 메모리 Write의 단계이다.

2. 해저드(Hazard)의 처리

파이프 라인 아키텍춰를 갖는 시스템에서는 동시에 수행되는 명령어들 간의 자원 (Data) 상충에 의한 타이밍 해저드와 분기명령어를 수행하여 분기 타겟을 결정하기 위한 지연시간 때문에 발생하는 시퀀싱 해저드 그리고 Load와 Store 명령사이의 메모리 액세스 시간의 차이로 인한 구조적 해저드 등으로 이상적인 파이프 라인의 동작을 기대하기 어렵다. 파이프라인 아키텍춰에서 인터록 메카니즘의 설계는 복잡할뿐만 아니라 고성능 프로세서에 대한 과도한 하드웨어 오버헤드를 부과하게 된다.[5][12]

HyRISC에서는 하드웨어 인터록을 사용하지 않고 컴파일시에 명령어들의 해저드 관계를 고려하여 원래의 코드 시퀀스를 배열 해줌으로써 해저드를 처리하는 코드 스케쥴링을 이용하는 것을 전제로 하였다. 소프트웨어적인 코드 스케쥴링에 의한 해저드 처리는 간단하고 규칙적인 하드웨어의 특징을 살릴 수 있고 단일 VLSI 프로세서 칩 제조를 위한 중요한 요소가 되고 또, 하드웨어의 감소로 인한 성능 향상을 기대할 수 있다.

한편, 프로그램 실행시 분기 명령어는 전체 명령어의 25%~30%를 차지하게 되며 분기 명령은 분기 타겟이 결정될때까지 파이프 라인을 일시적으로 중단 시키게 된다. HyRISC는 이를 해결하기 위해 지연 분기 방식(Delayed Branch)을 이용한다. 지연분기 방식은 분기명령의 수행으로 파이프라인의 지연이 발생하게 되는 시퀀싱 해저드가 가장 적은 하드웨어를 사용하여 해결하며 분기명령후에 중단 되었던 명령을 연속적으로 수행하도록 하기 위해 컴파일시에 명령들을 재배열 하거나 지연공간에 NOP(No OPeration)을 삽입하여 파이프라인을 동작시킨다. (그림 3)[5][9][12]

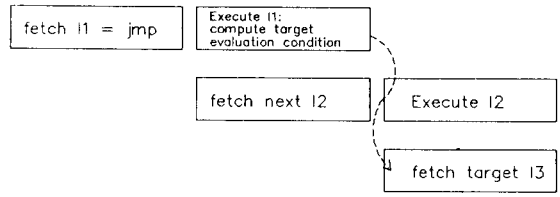


그림 3. 지연 분기 방식
Fig. 3. Delayed branch.

3. 명령어 세트(Instruction Set)

프로그램 카운터 부(PCU)에서 명령어 캐쉬(Instruction Cache)로 전달된 가상 어드레스(Virtual Address)로 부터 데이타 패스의 I-버스(Instruction Bus)에 액세스되는 명령어는 모두 45가지로 32-비트의 고정된 명령어 필드(Field)를 가지고 있으며 그 형태에 따라 4가지로 나누어 진다. 이 모든 명령은 레지스터 상에서만 수행되며 메모리 액세스 동작은 Load와 Store동작을 기본으로 한다. 명령어들의 형식과 동작은 표 1과 같다.

Ⅲ. 프로그램 카운터 부(PCU)의 구조

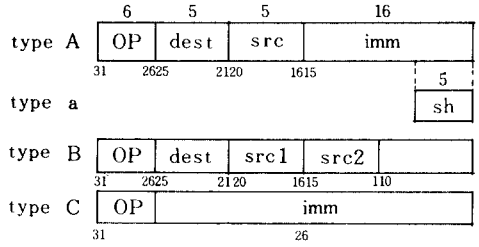
프로그램 카운터 부는 정상적인 동작시 다음번 메모리 사이클에 페치(Fetch)할 명령어의 어드레스를 연속적으로 제공하게 되며, 이러한 기본동작 외에는 프로그램 제어(Control) 명령의 리턴(Return) 되는 어드레스를 저장하기 위하여, 레지스터로 이동시키거나 레지스터와 ALU에서 계산된 어드레스를 가지고 새로운 명령을 수행하기 위한 어드레스를 제공하는 역할을 한다.

HyRISC의 프로그램 카운터 부는 4-스테이지파이프라인의 동작을 수행하기 위해 4개의 연속적인 고리(Chain) 구조로 되어 있으며 그 기본 구조는 그림 4에 나타내었다.

표 1. 명령어의 형식과 동작
Table 1. Format and operation of instruction set.

TYPE	INSTRUCTION	OPERATION
A	LDBS	dest<−M[src+imm]
A	LDBU	dest<−M[src+imm]
A	LDHS	dest<−M[src+mm]
A	LDHU	dest<−M[src+imm]
A	LDWS	dest<−M[src+imm]
A	STB	M[src+imm]<−dest
A	STH	M[src+imm]<−dest
A	STW	M[src+imm]<−dest
A	ADDI	dest<−src+imm
A	SUBIR	dest<−src+imm
A	LDHI	dest<31:16> <−src+imm;
		dest⟨31:16⟩<−0
A	ANDI	
A	ORI	
A	XORI	
a	SLLI	dest<−src shifted by sh
a	SRLI	dest<−src shifted by sh
a	SRAI	dest<−src shifted by sh
B	ADD	dest<−src1+src2
B	ADDC	dest<−src1+src2+carry
B	SUB	dest<−src1−src2
B	SUBC	dest<−src1−src2−carry
B	SUBR	dest<−src2−src1
B	AND	dest<−src1. AND. src2

TYPE	INSTRUCTION	OPERATION
B	OR	dest<−src. ORsrc2
B	XOR	dest<−src1. XOR. src2
B	SLL	dest<−src1 shifted by sh
B	SRL	dest<−src1 shifted by sh
B	SRA	dest<−src1 shifted by sh
C	BEQ	Npc<−pc1+imm
C	BNE	Npc<−pc1+imm
C	BLT	Npc<−pc1+imm
C	BGE	Npc<−pc1+imm
C	BLO	Npc<−pc1+imm
C	BHS	Npc<−pc1+imm
C	BNV	Npc<−pc1+imm
C	BRV	Npc<−pc1+imm
C	BRA	Npc<−pc1+imm
A	CALL	dest<−pc2
		Npc<−pc1+offset
A	RET	Npc<−src
A	SAVEPC	dest<−Lpc
A	SAVEPSW	dest<psw
A	BACKPSW	psw<−src
A	RETI	s bit<−p bit
		Npc<−R[trap-Lpc]
A	TRAP	dest<−Lpc
		Npc<−trap vector
A	JUMPPC	Npc<−R[SAVEPC]



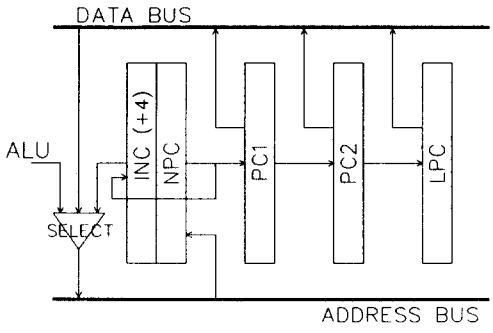


그림 4. 프로그램 카운터 부의 기본 구조
Fig. 4. Block diagram of PCU.

1. 기본 동작 및 동작 조건

그림 4 에서와 같이 프로그램 카운터는 현재 명령어 사이클(Instruction Cycle) 동안 수행되는 명령어의 어드레스를 갖는 PC1 레지스터와 파이프라인 상에서 최근에 수행 되었거나 수행되어질 명령어의 어드레스를 저장하는 LPC와 PC2로 구성되며, 현재 사이클 동안 페치(Fetch)되는 명령어의 어드레스를 갖는 NPC 레지스터의 내용이 PC1으로 전달됨과 동시에 + 4 (Byte Address)연산을 수행하는 인크리먼터(INC)가 NPC에 병렬로 연결되는 구조를 갖는다. 즉, 프로그램 카운터 부의 레지스터들은 파이프라인 아키텍춰 구조의 데이타 패스에서 수행되어 지

는 1-머신 사이클 단위의 명령어 어드레스들을 체인(Chain) 구조로 유지하게 된다.

프로그램 카운터 부의 BUS와 연결된 레지스터 입출력 채널은 각 레지스터의 내용을 그림 1의 데이타 패스에서와 같이 어드레스 버스 또는 데이타 버스를 통해 명령어 캐쉬와 D(Destination)레지스터로 보내거나 IOU(Integer Operation Unit)로 새로운 어드레스의 계산을 위해 전달하는데 사용 되어진다.

각 입출력 채널에 사용되어진 래치(Latch)는 그림 4에 표시된 바와 같이 각 파이프라인 스테이지의 4 Phase중 하나의 Phase에 동작하게 된다. 특히 인크리먼터(INC) 부분에서는 NPC의 값이 PC1으로 전달됨과 동시에 NPC의 어드레스 값에 4(Byte Address)를 더하여 하나의 Phase로 설정된 15nsec 이내에 NPC에 재 저장할 수 있도록 고속의 연산회로로 구성하여, 종래의 ALU나 Adder를 이용한 방식에서 발생되는 시스템 클럭 증가의 문제를 해결 하였다.

HyRISC는 하나의 스테이지가 60nsec의 동작속도를 갖는 아키텍춰의 프로세서로 설계하였으며 파이프라인의 동작과 그에 따른 프로그램 카운터 부의 기본 동작을 그림 5에 나타 내었다.

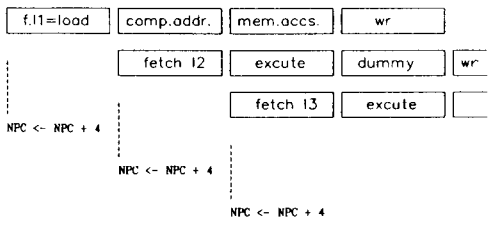


그림 5. 파이프라인과 프로그램 카운터 부의 동작
Fig. 5. Pipeline and basic operation of PCU.

그림에서와 같이 프로그램 카운터 부의 인크리먼터는 파이프라인의 매 스테이지마다 +4의 연산을 수행하게 되는데 그 동작 시기는 네번째 Phase에서 수행 되도록 하며(그림 6 참조), NPC 레지스터에 데이타가 충분히 안정되는 시간을 고려하여 NPC에 새로운 값을 Load할 때는 바로 다음 스테이지가 아닌 두 번째 스테이지에서 수행하게 하였다.(표 3, 4 참조)

2. 수행되는 명령어 세트(Instruction Set)

(1) 동작 개요

전체 명령어 세트중 프로그램 카운터 부의 기본 동작과 관련되는 명령을 제외한 명령 세트들을 표 2에 나타내었다.

표 2. 프로그램 카운터 부에서의 예외동작 명령
Table 2. Exceptional instruction set on PCU.

Control Transfer Inst.

*Program Control

CALL dest<-pc2
Npc<-pc1+offset

RET Npc<-src

SAVEPC dest<-Lpc

*System Control(Previleged Instruction)

SAVEPSW dest<-psw

BACKPSW psw<-src

RETI if i of psw is on
s bit<-p bit
Npc<-R[trap_Lpc]

Trap TRAP dest
dest<-Lpc
Npc<-trap vector

Hardware Interrupt
INTR dest, #V
dest<-Lpc
Npc<-#V
disable interrupt

(#V : hardware generated interrupt vector)

Jumppc Npc<-R[SAVEPC]

JUMP and BRANCH Inst.

BEQ, BNE, BLT, BGE, BLO, BHS, BNV, BRV, BRA
op-code, src1, src2, imm Npc<-pc1+imm

표 2의 명령어 세트들에 관한 동작은 타이밍과 관련하여 2.2절에서 자세히 다루기로 한다. 이상의 동작을 만족하기 위한 프로그램 카운터 내에서의 데이타 패스를 고려하면 그림 6과 같은 Phase별 동작을 보이게 된다.

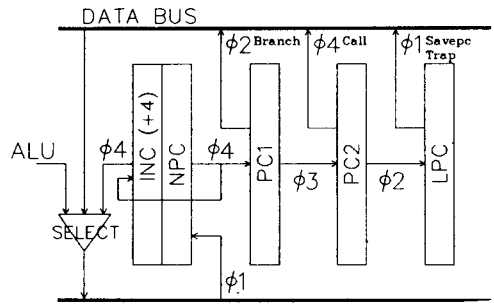


그림 6. 프로그램 카운터 내부 래치의 동작
Fig. 6. Operation of internal latch of PCU.

각 래치(Latch)에 표시된 Phase는 해당 파이프라인상에서의 동작 시기를 나타낸다.

(2) 동작 타이밍과 파이프 라인

먼저 정상 상태에서의 기본 동작(Default Operation)은 NPC와 INC에서 계속적으로 +4(Byte Address) 연산을 수행하여 명령어 캐쉬에 전달하게 되고 PC1은 현재 수행되는 명령어의 번지를 저장하게 되고 나머지 LPC는 인터럽트나 트랩이 발생할 경우에 대비하여 현재 머신 사이클에서 수행되는 각 파이프라인의 프로그램 카운트를 저장하게 된다. 가장 대표적인 예인 로드(Load)와 스토아(Store) 명령의 Phase에 따른 각 스테이지별 동작을 표3에 나타내었다.

정상 상태 이외의 동작시에는 대개 현재 수행중인 명령어들의 어드레스를 대피시킨 후 새로운 동작 상태로 만들어 주게 되는데 표2에 표시한 명령들이 바로 그것들이다. 이 명령들 중 대표적인 것 몇가지의 Phase에 따른 스테이지별 동작을 표4에 나타내었다.

이러한 명령들로서 실제 파이프라인 상에서 예외 상황(Exceptional States)이 발생할 경우를 예로 들어 그림7에 나타내 보았다.

그림에서 처음 Branch 명령이 페치(Fetch)되어 실행될 때 (a지점) 이미 두번째 명령은 디코드를 완료한 상태가 되며, Branch 명령으로 부터 발생된 세번째 명령이 페치 되어진다. 이때·두번째 명령에서 예외(Exception) 상황(캐쉬 미스, 하드웨어 인터럽트 등)이 발생한다면 b지점에서 감지 되어지고 이때 세번째 명령은 이미 디코드가 완료된다. 이 경우 두번째와 세번째 명령은 데이타 패스 상에서 삭제(Flush)

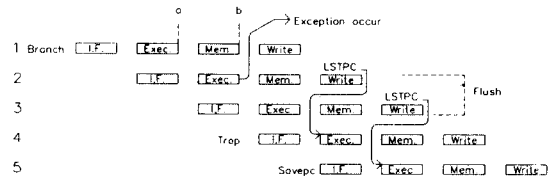


그림 7. 예외 상황의 처리
Fig. 7. Exception handling.

표 3. 정상 상태의 데이타 패스 동작
Table 3. Data path operation of deafult states.

Load Byte Signed(type A)

STAGE 1				STAGE 2				STAGE 3				STAGE 4			
ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
Instruction		Fetch	Inst. DEC	RF. read S2 <= RB	A1←IMM2	ALU		Data	Load			Shifter (Align-ment)		RF. Write	
		IMM1 (0 : 15)	REG. DEC			Addition	BAR <= efA (1:0)	Tag detec-tion		SIGN, EXT	S1 <= Dest2		Dest2		Precharge
(Tag detection)		RB <= (20 : 16)	Match Detection		B1 <= S2 IMM2 (Sign Extention)		SAR <= BAR			Dest2 <= EADD (31 : 0)					
		RD <= (25 : 21)	PC1 <= NPC INC	IMM2 <= IMM1			Latch2 <= NPC								
			Latch <= NPC	Internal Forwarding		PC2 <= PC1									

Store Byte

STAGE 1				STAGE 2				STAGE 3				STAGE 4			
ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
Instruction		Fetch	Inst DEC	REG. FILE	B1<=S1	ALU	(ADD)	Data	Cache	(store)					
		IMM1 (0 : 15)	REG. DEC	S2 <= D(31 : 0)				Tag detec-tion							
(Tag detection)		RB (20 : 16)	Match Detection	S2 <= RD(31 : 0)	A1←IMM2	Shifter <=S2	Shifter (align-ment)								
		RD (25 : 21)	PC1 <= NPC INC	IMM2 <= IMM1			BAR <= EADD (1 : 0)	Dcache <= EADD (31 : 2)							
			Latch <= NPC	Internel Forwarding		PC2 <= PC1	SAR <= BAR								

표 4. 정상 상태 이외의 명령에 따른 데이타 패스 동작
Table 4. Data path operation of exceptional states.

TRAP (type A)

	STAGE 1			STAGE 2				STAGE 3				STAGE 4			
	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
	Instruction		Fetch	Inst. DEC REG. DEC			ALU		Dummy						RF. Write	Precharge
	(Tag detection)		RD <= (25 : 21)	PC1 <= NPC INC Latch1 <= NPC	S1 <= LPC S2⇔RO	AI <= S1 BI <= S2 PC3 <= PC2	PC2 <= PC1	Dest1 <= Data (31 : 6)	Icache <= trapvector NPC <= trapvector		Dest2 <= Dest1					

Callr (type A)

	STAGE 1			STAGE 2				STAGE 3				STAGE 4			
	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
	Instruction		Fetch	Inst. DEC REG. DEC			ALU	(ADD)							RF. WRITE	Precharge
	(Tag detection)		RD <= (25 : 21)	PC1 <= NPC INC Latch1 <= NPC	SIGN EXT IMM2 <= IMM1	AI←IMM2 BI <= PC1	PC2 <= PC1	Dest1 <= PC2	Icache <= EADD (31 : 2) NPC <= EADD(31:2)		Dest2 <= Dest1					

Jumppc (type A)

	STAGE 1			STAGE 2				STAGE 3				STAGE 4			
	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
	Instruction		Fetch	Inst. DEC REG. DEC	RF. read		ALU									
	(Tag detection)		RA RB <= RO	PC1 <= NPC INC	S1 <= RA S2 <= RB	AI <= S1 BI <= S2		BAR <= efA (1 : 0)	NPC <= DATA Icache <= data							

Save PC (type A)

	STAGE 1			STAGE 2				STAGE 3				STAGE 4			
	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
	Instruction		Fetch	Inst. DEC REG. DEC	S2 <= RO										RF. Write	Precharge
	(Tag detection)		RO RD <= (25 : 21)	PC1 <= NPC INC Latch1 <= NPC	S1 <= LPC	AI <= S1 BI <= S2	PC2 <= PC1	Dest1 <= Data			Dest2 <= Dest1 !					

되어져야 하며 예외 처리(Exception Handling)후 다시 수행되어지기 위해서는 그 명령어의 어드레스를 저장하여야 한다. 그러므로 네번째와 다섯번째에 그림과 같이 Trap과 Savepc 명령을 사용하여 두번째와 세번째 명령의 네번째 스테이지에서 LPC에 저장되어 있을 어드레스 값을 각기 지정된 Destination에 저장시키게 된다. 이와같은 동작을 위해서는 그림 6에서와 같은 Phase별 래치의 동작이 요구된다.

Branch 명령은 PC1의 내용과 IMM을 더한후 INC를 거치지 않고 NPC에 결과를 저장함으로써 완료된다.

Ⅳ. 회로 설계(Circuit Design)

1. 기본 레지스터 셀(Basic Register Cell)의 설계

프로그램 카운터 부에서 NPC, PC1, PC2 그리고 LPC에 사용되는 기본 레지스터 셀은 데이타 패스의 레지스터 파일(Register file)에서와 같이 순환루프를 갖는 인버터(Inverter)쌍으로 구성하였고 그림 6의 구조를 만족시키기 위해 그림 8과 같이 패스 트랜지스터 (Pass Tr.)를 배열하였다.

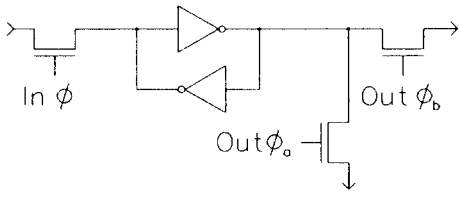


그림 8. 기본 레지스터 셀
Fig. 8. Basic register cell.

이중, 데이타 버스(Data Bus)에 연결되는 패스에는 버스 드라이버(Bus Driver)를 접속시켜 설계하였다. 이제, NPC에서 ϕ_a로 인하여 인크리먼터(INC)로 전달되는 내용을 가지고 +4(Byte Address) 연산동작을 수행하는 인크리먼트 부에 대한 기본 설계를 생각해 보자.

2. 인크리먼터의 설계

고속의 데이타 패스 동작을 수행하는 RISC 타입의 머신에서는 파이프라인 구조의 아키텍춰를 가지고 있으며 파이프라인의 매 스테이지마다 프로그램 카운트를 증가시켜야 한다. 또한 파이프라인상에서의 예외처리(Exception Handling)를 위해서는 각 스테이지의 일정 Phase 내에 동작을 완료하여야 한다. (표3, 4 참조)

HyRISC는 4-스테이지 파이프라인 구조로써 하나의 스테이지 당 15nsec의 길이를 갖는 4-Phase 클럭킹으로 구동된다. 이 경우 하나의 Phase에 해당되는 15nsec 이내에 인크리먼트 동작과 레지스터에 안정된 상태로 저장하기까지의 동작을 완료하여야 한다. 기존의 ALU를 사용하거나 전용 가산기(Adder)를 사용할 경우 제한 시간내의 동작이 어려우므로, 여타 RISC프로세서에서는 특수형태의 가산기를 사용하여 처리하거나[4] 전용 하드웨어로써 처리하고 있다.[10]

데이타 패스 동작에 적합한 속도를 얻기위해 NPC와 직접 연결되어 순환루프를 갖는 자리 이동형(Bit Displacement) 전용 하드웨어를 제안하여 프로그램 카운터 부의 설계에 사용하였으며, +4 동작의 기본 개요는 다음과 같다.

- 하위 2^0, 2^1자리 비트의 내용은 변함없이 전달한다.
- 2^2자리 비트의 내용을 무조건 반전시켜 전달한다.
- NPC의 2^2자리 비트의 내용이 0이였다면 2^3자리 이상의 비트들은 변화없이 전달한다.
- NPC의 2^2자리 비트의 내용이 1이였다면 2^3자리

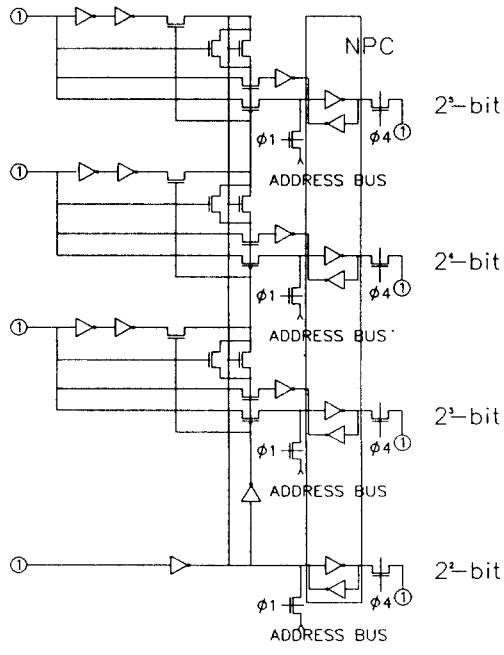


그림 9. 인크리먼트 부의 기본 회로도
Fig. 9. Basic circuit diagram of INC.

이상의 비트들중 처음 만나는 0 비트까지만 반전시켜 전달하고 나머지 상위 비트들은 변화없이 전달한다.

3. 전체 회로

프로그램 카운터 부의 동작 Phase와 전체회로도는 그림10과 같다.

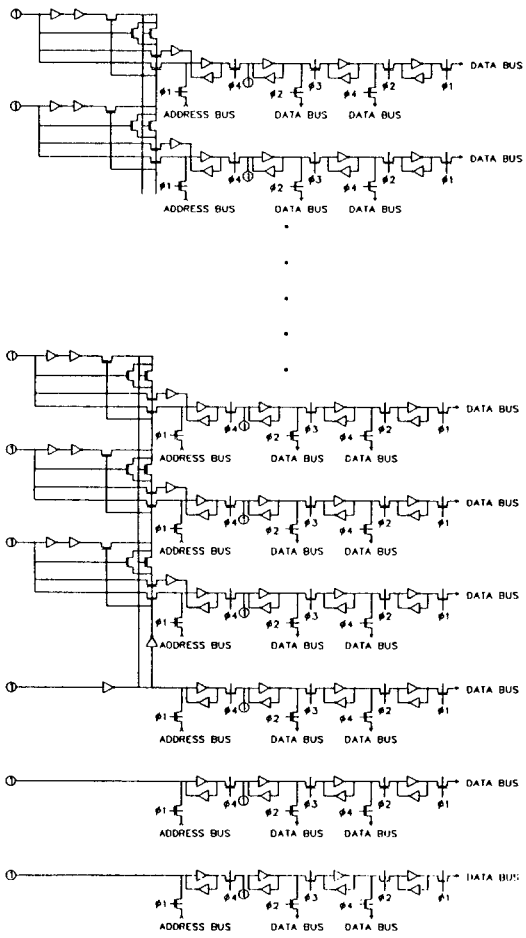


그림10. 프로그램 카운터 부의 회로도
Fig. 10. Circut diagram of PCU.

V. 성능 평가

제안한 프로그램 카운터 부(PCU)에 대한 회로 시뮬레이션은 Apollo 워크스테이션 상에서 Mentor Graphics CAD툴들을 이용하여 수행하였다. 이때 각 기본 소자의 지연 시간을 2μm-Double Metal 디자인룰의 파라미터를 기본으로 하여 얻은 SPICE시뮬레이션 결과를 적용하였다.[14] 즉, n-채널 트랜지스터, p-채널 트랜지스터, 그리고 기본 인버터의 지연시간을 1nsec, 1nsec, 1.6nsec로 하였으며 배선 지연은 고려하지 않았다.

HyRISC는 스테이지 당 60nsec(17Mhz)의 길이를 갖는 4-스테이지 파이프라인 구조로 설계되어 있으며 각 스테이지는 4-Phase의 기본 클럭킹을 사용한다. 그러므로, 하나의 Phase는 15nsec의 길이를 갖고 프로그램 카운터 부는 이 기간내에 동작을 완료하여야 한다.

이러한 동작조건을 만족하려면 어드레스 카운트 증가시 덧셈에서 유발되는 캐리 전파(Carry Propagation)의 문제를 해결해야 한다. 제안한 프로그램 카운터부의 인크리먼터는 그림 9 에서와 같이 캐리를 전파하는 방식이 아니고 2^2자리 비트의 값과 NPC로부터 순환되어 오는 값들과의 Wired-Logic 값으로 결정되는 캐리전파지연을 갖지 않는 방식이다.

본 논문의 프로그램 카운터 부의 최대 지연 시간은 NPC에서 출발한 데이타가 인크리먼터(INC)를 거쳐 다시 NPC에 저장되어 안정화 되는데 까지의 기간이며 레지스터의 안정화 기간을 1nsec로 하였을 때 시뮬레이션 결과는 8.8nsec의 지연을 갖는다. 이러한 결과는 HyRISC의 파이프라인을 지연시키지 않고 효과적으로 지원할 수 있다.

Ⅳ. 결 론

본 논문에서는 HyRISC 프로세서의 파이프라인 상에서 정상동작(Default)과 예외동작(Exception) 발생시 명령어 캐쉬에 지속적으로 어드레스를 제공할 수 있도록 자체 +4 연산 인크리먼터를 가지고 있는 프로그램 카운터 부를 제안하고 설계하였다.

정확한 동작에 대한 설계 사양을 얻기 위하여 정상동작과 예외동작에 대한 전체 데이타 패스의 동작을 각 스테이지의 Phase별로 분석하였으며 설계와 시뮬레이션은 Apollo DN4000 워크스테이션 상에서 Mentor Graphics CAD툴들을 이용하여 수행하였다. 시뮬레이션 결과 8.8nsec의 동작시간을 얻어 한계치인 15nsec 이내에 동작을 완료할 수 있음을 보였다.

앞으로의 연구과제로는 명령어 캐쉬와 연결하여 캐쉬 미스시의 동작에 대한 정확한 시뮬레이션과 그 밖의 예외동작시에 대한 연구가 요구된다.

參 考 文 獻

[1] Amar Mukherijee, "Introduction to nMOS&cMOS VLSI system design," Prentice-Hall, 1986.

[2] Carver Mead, Lynn Conway, "Introduction to VLSI system," Addison-Wesley Publishing Company, 1980.
[3] Inseok S. Hwang, Aaron L. Fisher, "Ultra compact 32-bit CMOS adders in multi-output domino logic," *IEEE Journal of Solid State Circuits* pp. 358-369, April 1989.
[4] Joh L. Hennessy, Mark A. Horowitz, "An overview of the MIPS-X-MP project," CSL, Stanford Univ. April. 1986.
[5] Manolis G. H. Katevenis, "Reduced instruction set computer architecture for VLSI," ACM Doctoral Dissertation Award 1984, The MIT Press.
[6] M. Morris Mano, "Digital logic and computer design," Prentice Hall, 1979.
[7] MIPS R2000 Processor User's Manual.
[8] Neil Weste, Kamran Eshraghian, "Principles of CMOS VLSI design A systems perspective, Addison-Wesley, 1985.
[9] Robert Warn Sherburne Jr. "Processor design tradeoffs in VLSI," Dissertion for the degree of Doctor of Philosophy of the Univercity of California, Berkeley, 1984. 4.
[10] Richard A. Blomseth, "A Big RISC," Masters Project Final Report, University of California, Berkeley, July 1983.
[11] 홍인식, 선선구, 이승호, 정성호, 임인칠, 32 bit 마이크로 프로세서의 controller 설계에 관한 연구, 과학기술처 87 특정연구 결과 발표회 논문집, pp177-180, 1989.
[12] 김은성 파이프라인 RISC 아키텍춰의 코드 최적화에 관한 연구, 한양 대학교 박사학위논문, 1988.
[13] Reduced Instruction Set Computer에 관한 연구, 전자통신연구소 최종보고서, 1987. 3.
[14] RISC 프로세서의 Data Path설계에 관한 연구, 전자통신연구소 최종 보고서, 1988. 5.

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25卷 第8號 參照
현재 한양대학교 전자공학과 교수