論文 90-27-7-4

RISC 프로세서 제어부의 설계

(Design of A RISC-Processor's Control Unit)

洪 仁 植* 林 寅 七*

(In Sik Hong and In Chil Lim)

要 約

본 논문에서는 고성능 RISC(reduced instruction set computer)형 마이크로 프로세서의 제어부 설계를 제안한다. 이 제어부는 4-스테이지 파이프라인 아키텍춰 상에서 타겟 프로세서의 전체 데이타 패스와 on-chip 명령어/데이타 캐쉬를 제어한다.

속도 개선을 위하여, 데이타 패스와 제어부 회로의 대부분은 domino-CMOS 기술과 hard-wired 기술로 설계한다.

먼저, 명령어 세트와 데이타 패스를 정의하고 데이타 패스의 제어에 필요한 모든 신호를 분석한 뒤 제어부의 명령어 디코더와 클럭 생성 논리 블럭(clock generated logic block)을 기존 다이나믹 CMOS PLA의 단점을 보완한 DCAL(dynamic CMOS array logic)기술을 적용하여 설계한다.

회로의 시뮬레이션은 Apollo W/S상에서 Mentor Graphics CAD툴 들을 사용하여 수행한다.

Abstract

This paper proposes the control unit of a 32-bit high-performance RISC type microprocessor. This control unit controls the whole data path of target processor and on chip instruction/data caches in 4-stage pipelined scheme. For the improvement of speed, large parts of data path and control unit are desiged by domino-CMOS and hard-wired circuit technology.

First, in this paper, target processor's instruction set and data path are defined, and next, all signals needed to control the data path are analyzed. The decoder of conrol unit and clock generated logic block are implemented in DCAL (Dynamic CMOS Array Logic) with modified clock scheme for the purpose of speed up and supporting RISC processor's pipelined architecture efficiently.

I. 서 론

컴퓨터 아키텍춰 및 반도체 기술의 발달은 최근 메가 프로세서 테크노로지의 단계에까지 와있으며, 이러한 워크스테이션(workstation)급의 컴퓨터는 기존의 슈퍼 미니급의 처리능력을 보유하게 되었다. 현재 컴퓨터 시스템의 개발은 RISC(reduced instruction set computer)의 응용이 크게 확대되고 있는 추세이다.[1] RISC프로세서는 하드웨어의 단순화를 지향하여 하드웨어 자원의 이용효율을 극대화 하고 디자인 사이클을 대폭줄이는 것이 특징이다. 그리고 고속의 클럭 사이클로 대부분의 명령어를 단일 사이클 내에 처리하여 저렴한 비용의 개발비로 높은 성능을

*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 3月 19日

얻을 수 있음을 보여주고 있다.[2][3] 또한 RISC는 모든 명령어가 동일한 크기를 갖고 고정된 형식(fixed format)을 사용하므로 명령어의 디코드 시간을 감소시키고 이를 처리하기 위한 데이타 패스를 단순화할 수 있으며, 그러한 특징을 이용, 고속의 동작속도와 강력한 파이프 라인(pipeline)아키텍춰를 실현할 수 있다.[2]

초기의 RISC 아키텍춰에서는 액세스 빈도가 높은 local-scalar 변수를 액세스가 매우 빠른 레지스터 화일에 저장하여 빠른 속도를 갖도록 설계하였으나 명령어(instruction)액세스의 경우 패드(pad)를 거쳐야 하므로 많은 지연을 초래하였다.[2] 또한 배열, 구조체, global-scalar 변수들은 컴파일러가 레지스터에 할당하는 것이 어려웠다.[4] 이러한 문제점은 CPU와 주기억 장치 사이에 고속의 버퍼로서 캐쉬 메모리를 사용하여 해결할 수 있으며, 최근에는 단일 레벨이 아닌 다중 레벨 캐쉬와 공유 메모리를 갖는 다중 프로세서 분야에서도 캐쉬에 대한 연구는 활발히 진행되고 있다.[3][4][5]

본 논문에서 제안하는 제어부의 타겟프로세서는 독자적인 명령어 세트(instruction set)와 고속의 메모리 액세스 블럭인 레지스터 화일, 그리고 on-chip으로 명령어 캐쉬와 데이타 캐쉬를 가지는 HyRISC로 한다.[21][23] 데이타 처리를 위한 데이타패스는 각 기능 블럭별로 나누어 설계중이며 대부분이 속도가 빠른 domino-CMOS로 설계된다. 본 논문에서는 이러한 데이타 패스와 캐쉬 메모리, 그리고 기능 블럭간의 data bus를 제어할 수 있는 제어부를 제안한다. 제어부는 OP 코드를 입력으로 받아들여 파이프라인 스테이지의 하나의 클럭 phase내에 각 데이타 패스를 제어할 수 있는 제어신호를 발생시켜야 하므로 빠른 동작속도가 요구된다.[3] 제어부의 OP 코드 디코더로써 일반적인 PLA(programmable logic array)를 사용할 경우 그 속도면에서 만족할 만한 결과를 얻기 어렵다.[2] 또한, CMOS Logic Array가 가지고 있는 누설 전류와 전하 결합에 의한 제반 문제와 race 문제를 일으킬 수 있다. HyRISC의 제어부 디코더에는 이러한 단점들을 보완하고 매우 빠른 동작속도를 갖는 DCAL 기술을 사용하며, PLA 논리 최소화 툴인 PLAMIN[18]을 이용 회로를 단순화 한다.[25]

Ⅱ. 데이타 패스

타겟 프로세서의 데이타 패스는 하드웨어 구조를 단순화 시키고 수행속도를 향상 시키기 위해 많은 부분을 하드 와이어드 로직을 사용하여 구성 하였으며 그 구성도는 그림 1과 같다. 전체구성은 정수처리를 수행하기 위해 쉬프터(Shifter)와 ALU로 구성되어 있는 IOU(integer operation unit), 그리고 변수들을 저장하기 위한 32-워드 크기의 레지스터 화일, 어드레스를 저장하기 위한 PCU(program counter unit), 명령어를 이동시키기 위한 명령어 버스, 데이타를 운반하기 위한 데이타 버스, 그리고 명령어 페치와 데이타의 Read/Write를 빠르게 수행하기 위한 명령어/데이타 캐쉬 메모리 등으로 이루어지며 우수한 성능/가격비를 얻기 위하여 RISC 아키텍춰의 개념에 충실하도록 하였으며, 모든 명령은 거의 일정한 데이타패스를 거쳐 처리된다.

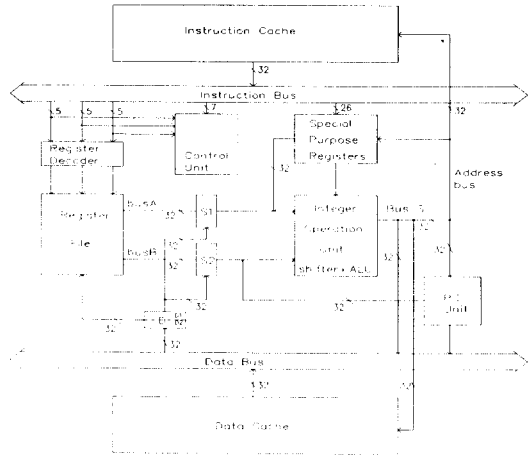


그림 1. HyRISC의 데이타 패스
Fig. 1. Data path of HyRISC.

Ⅲ. 명령어(Instruction Set)와 파이프라인(Pipeline)

하드웨어(데이타 패스)의 설계시 그에 따른 컴파일러의 설계가 가장 큰 문제로 대두되는데, HyRISC는 [2]논문의 예제프로그램 결과를 참고하여 load와 store, 논리 산술계산명령, 제어명령등 44개의 32-bit로 고정된 명령어 필드를 가지고 있으며 명령어 형태와 데이타 패스상에서의 레지스터 할당 그리고 예외(Exception) 발생시의 처리방식등을 버클리 계보의 RISC 프로세서와 유사한 형태를 갖도록 설계하여 기존의 SPARC 스테이션(SUN 4/60)의 컴파일러를 일부 수정하여 이식가능하도록 설계하였으며, SPARC 스테이션의 컴파일러에 관한 연구는 진행중에 있다. 명령어의 형태는 그 목적상 그림 2와 같이 4가지 종류로 나누었으며, 형태에 따른 명령의 종류는 표 2에 나타내었다.

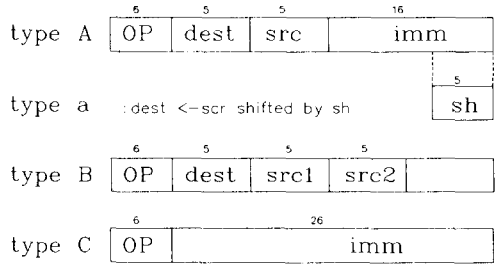


그림 2. 명령어의 형식
Fig. 2. Instruction format of HyRISC.

1. 레지스터를 이용한 연산(register to register operation)

RISC 아키텍춰에서 모든 명령은 레지스터 상에서만 수행된다. 그리고 메모리 액세스 동작은 load와 store 동작을 기본으로 한다. 이러한 레지스터 동작은 기본적으로 3주소 코드 방식을 취하며 그 형태는 다음과 같다.

DEST←SRC1 op SRC2.

2. 명령어의 정의

명령어의 op-code는 규칙성을 최대한 이용하면서 제어부에서의 디코드(decode)용이성을 고려하여 설계하였으며 표 1과 같다.

표 1. 명령어 op-code
Table 1. Op-code of instruction set.

	00xxxx	01xxxx	10xxxx	11xxxx
xx 0000	LDBS	ADDI	ADD	BEQ
xx 0001	LDBU	SUBI	ADDC	BNE
xx 0010	LDHS	LDHI	SUB	BLT
xx 0011	LDHU	ANDI	SUBC	BEG
xx 0100	LDWS	ORI	SUBR	BLO
xx 0101		XORI		BHS
xx 0110				BNV
xx 0111				BRV
xx 1000	STB	CALL	AND	BRA
xx 1001	STH	RET	OR	
xx 1010	STW	SAVEPC	XOR	
xx 1011		SAVEPSW		
xx 1100	SLLI	BACKPSW	SLL	
xx 1101	SRLI	RETI	SRL	
xx 1110	SRAI	TRAP	SRA	
xx 1111				

표 1의 명령어들은 각기 다음과 같은 기능을 수행한다. (표 2)

3. 파이프 라인(pipeline)

HyRISC는 하드웨어 자원의 효율적인 이용을 위하여 기본적으로 3-스테이지 파이프 라인으로 구성되며 각각 instruction fetch/decode, execution, memory write의 단계를 갖는다. 그러나 단일 포트를 사용하여 외부 메모리를 액세스 하는 경우, 현재 수행중인 데이타 read/write가 종료될때까지 파이프라인의 지연이 발생하게 된다. 명령어 페치의 경우, on-chip instruction 캐쉬를 내부에 두고 있으므로 명령어 페치의 지연은 대부분 보완될 수 있으나 레지스터 화일상에 존재하지 않는 데이타를 액세스 하는 경우, 즉 그러한 데이타를 데이타 캐쉬 또는 외부 메모리로 부터 가져오는 경우 파이프 라인의 지연을 최대한 이용하도록 하는 방법에는 두가지를 생각할 수 있다.

첫째는 2중으로 write가 가능한 레지스터 화일을 설계하는 것이고 두번째 방법은 dummy 스테이지를 마련해 두는 것이다. HyRISC 에서는 가능한 한 단순화된 데이타 패스를 목표로 하고 있으므로 첫번째의 부과 하드웨어의 오버헤드가 발생되는 방법을 피하고 두번째 방법을 선택하여 파이프 라인을 구성하였다. (그림3)이 경우 dummy 스테이지는 forwarding을 위한 내부참고와 캐쉬미스시 맵모리 액세스에 이용할 수 있으므로 과도한 파이프라인의 지연을 초래하지 않는다.[20][23]

1) 해저드(Hazard)의 처리.

파이프라인 아키텍춰를 갖는 시스템에서는 동시에 수행되는 명령어들 간의 자원(data) 상충에 의한 타이밍 해저드와 분기명령어를 수행하여 분기타겟을 결정하기 위한 지연시간 때문에 발생하는 시퀀싱 해저드 그리고 load와 store 명령 사이의 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 부가하게된다.[2][29]

본 논문에서는 하드웨어 인터록을 사용하지 않고 컴파일시에 명령어들의 해저드 관계를 고려하여 원래의 코드 시퀀스를 재 배열 해줌으로써 해저드를 처리하는 코드 스케줄링을 이용하였다. 소프트웨어적인 코드 스케줄링에 의한 해저드 처리는 간단하고 규칙적인 하드웨어의 특징을 살릴 수 있고 단일 VLSI 프로세서 칩 제조를 위한 중요한 요소가 되고 또,

표 2. 명령어의 정의
Table 2. Definition of instruction set.

TYPE	INSTRUCTION	OPERATION	TYPE	INSTRUCTION	OPERATION
A	LDBS	dest←M[src+imm]	B	OR	dest←src1. OR. src2
A	LDBU	dest←M[src+imm]	B	XOR	dest←src1. XOR. src2
A	LDHS	dest←M[src+imm]	B	SLL	dest←src1 shifted by sh
A	LDHU	dest←M[src+imm]	B	SRL	dest←rrc1 shifted by sh
A	LDWS	dest←M[src+imm]	B	SRA	dest←src1 shifted by sh
A	STB	M[src+imm]←dest	C	BEQ	Npc←pc1+imm
A	STH	M[src+imm]←dest	C	BNE	Npc←pc1+imm
A	STW	M[src+imm]←dest	C	BLT	Npc←pc1+imm
A	ADDI	dest←src+imm	C	BGE	Npc←pc1+imm
A	SUBIR	dest←src+imm	C	BLO	Npc←pc1+imm
A	LDHI	dest⟨31:16⟩←src+imm;	C	BHS	Npc←pc1+imm
		dest⟨31:16⟩←0	C	BNV	Npc←pc1+imm
A	ANDI		C	BRV	Npc←pc1+imm
A	ORI		C	BRA	Npc←pc1+imm
A	XORI		A	CALL	dest←pc2
a	SLLI	dest←src shifted by sh			Npc←pc1+offset
a	SRLI	dest←src shifted by sh	A	RET	Npc←src
a	SRAI	dest←src shifted by sh	A	SAVEPC	dest←Lpc
B	ADD	dest←src1+src2	A	SAVEPSW	dest←psw
B	ADDC	dest←src1+src2+carry	A	BACKPSW	psw←src
B	SUB	dest←src1−src2	A	RETI	s bit←p bit
B	SUBC	dest←src1−src2−carry			Npc←R[trap-Lpc]
B	SUBR	dest←src2−src1	A	TRAP	dest←Lpc
B	AND	dest←src1. AND. src2			Npc←trap vector

하드웨어의 감소로 인한 성능 향상 기대할 수 있다.

한편, 프로그램 실행시 분기 명령어는 전체 명령어의 25%~30%를 차지하게 되며 분기 명령은 분기 타겟이 결정될때까지 파이프 라인을 일시적으로 중단시키게 된다. HyRISC는 이를 해결하기 위해 지연 분기 방식(delayed branch)을 이용한다. 지연 분기 방식은 분기명령의 수행으로 파이프 라익의 지연이 발생하게 되는 시퀀싱 해저드가 가장 적은 하드웨어를 사용하여 해결하며 분기명령후에 중단되었던 명령을 연속적으로 수행하도록 하기 위해 컴파일시에 명령들을 재배열 하거나 지연공간에 NOP(No OPeration)을 삽입하여 파이프 라인을 동작시킨다. (그림 4)

즉 그림 4에서와 같이 분기 명령인 I1은 분기타겟이 결정될때까지 파이프라인을 일시적으로 중단시켜야 하기 때문에 I1과 무관한 I2를 페치하거나 NOP을 삽입하여 파이프라인을 진행시키다가 분기 타켓인 I3가 결정되면 계속 실행해 나간다.

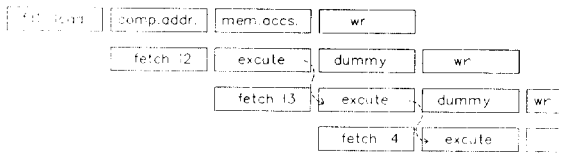


그림 3. 명령어 캐쉬를 이용한 파이프라인
Fig. 3. Pipe line using instruction cache.

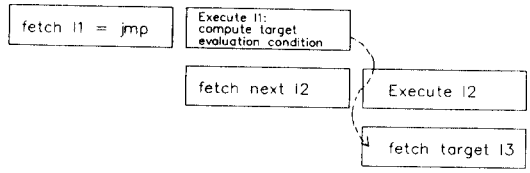


그림 4. 지연 분기 방식
Fig. 4. Delayed branch.

2) Forwarding

파이프 라인 시스템에서 데이타 상호 의존관계의 발생을 좀 더 빠르게 해결해주기 위해 스테이지간의 communication 채널을 추가로 만들어 주는 방법을 사용하는데 이러한 방법을 forwarding 또는 short circuiting이라고 하며 파이프 라인에서 데이타 의존관계의 발생시간을 감소시키기 위해 널리 사용된다. 그림 5의 예를 들어보면 I2명령어의 계산된 결과를 I3 명령수행중 사용한다면, write 스테이지가 끝날때까지 기다리지는 않고 I2의 스테이지 2에서 계산된 결과를 곧바로 bypass 시켜준다.[3][23]

HyRISC에서는 제어부내에 match detector를 부과하여 이러한 forwarding을 수행하도록 하였다. 그러나 forwarding mechanism을 수행하기 위해서는 선행조건으로 첫째, I2에서 compute된 데이타가 충분히 안정된 후에 temporary 래치로 이동시켜야 하며, 둘째, 스테이지 간의 데이타 전송에 걸리는 시간 만큼 파이프 라인 기본 사이클 시간을 늘려주어야 한다.

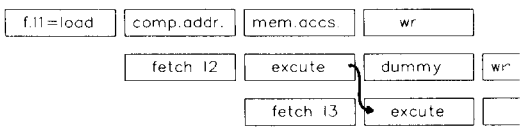


그림 5. Forwarding
Fig. 5. Forwarding.

3. 타이밍

RISC 아키텍춰에 있어서 데이타 패스의 제어는 하드와이어드로 직접 제어되므로 제어부의 제어신호 타이밍은 매우 중요한 비중을 차지한다. HyRISC의 시스템 클럭은 기본 4-phase를 사용하였으며 명령에 대한 각 phase별 시스템의 기본동작을 표3에 나타내었다. 표 3(a)에서 하나의 phase는 15nsec로 하나의 스테이지는 60nsec가 된다. 표 3(b)에는 몇가지 명령에 대한 파이프라인 스테이지별 데이타패스의 동작을 자세히 나타내었다.

Ⅳ. 제어부 (Control Unit)

제어부는 명령어를 페치(fetch)하여 각각의 명령어를 해석(decode)하여 데이타 패스가 적절히 동작하도록 제어해 주는 부분이다. CISC에 있어서 프로세서 내부에서 수행될 각 명령어에 대한 논리함수(logic function)들은 ROM에 저장되어지고 필요에 따라 꺼내서 사용되어진다. RISC에서는 명령어의 형식이 고정되므로 직접 디코드하여 제어신호를 발생시킬 수 있다. 즉, 명령신호가 직접 데이타 패스를 구동시키는 하드 와이어드 제어방식을 택하고 있다. 파이프라인에 의한 데이타 패스의 부분별 동작은 디코드 되어진 명령어를 타이밍 게이트의 입력으로 하여 특정 시간에 특정 데이타 패스를 동작 시켜주는 순차적 제어신호를 출력신호로 제공하여 준다.

1. 제어신호

HyRISC의 데이타 패스에 필요한 제어신호는 표 4와 같다.

2. 제어부의 구조

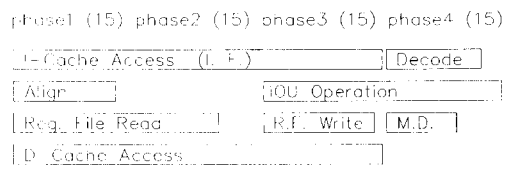
데이타 패스를 제어하는 제어부는 크게 op 코드를 디코딩하는 디코더와, 파이프 라인 제어부 그리고 디코드 되어진 신호와 시스템 클럭을 조합하여 파이프 라인에 부합되는 동작을 할 수 있도록 하는, 즉 적절한 시기에 제어신호를 데이타 패스에 보내기 위한 타이밍 회로, 그리고 internal forwarding 동작시 레지스터 비교와 그에따른 데이타 패스 제어를 위한 match detector로 이루어진다. 제어부의 전체구성은 그림 6에 나타내었다.

1) 디코더

디코더는 6-bit의 op코드와 1-bit의 interrupt bit를 입력으로 받아들여 44개의 명령어를 수행할 수 있도록 제어 신호를 발생시키며, DCAL을 이용한 PLA로 구성하였다. PLA의 구성중 OR 평면은 파이프라인의 스테이지에 따라 3부분으로 나누었으며 그 구조와 동작은 그림 7의 (b)에 나타내었다. DCAL PLA를 구동시키기 위한 클럭 생성 회로는 그림7의 (a)에 나타내었다.

그림 8(a)는 설계된 디코더를 PLA AND 평면논리 최소화 프로그램을 적용시킨 후의 함수 표현이다. PLA 논리 최소화 툴(PLAMIN[18])의 적용결과는 그림 8에 나타내었다. PLA의 제어신호 CK, CKD, CK*, CKD*는 그림 7의 (a)에서와 같이 시스템 클럭

표 3. (a) 데이타 패스의 phase별 기본동작
Table 3. (a) Basic operation of data path according to clock phase.



(a)

표 3. (b) 명령별 데이파패스의 동작

Table 3. (b) Operation of data path according to instruction.

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		ALU		Data	Load			Shifter (Align-ment)	Dest2	RF. Write	
		IMM1 (0:15)	REG. DEC	S2 <= RB	AI←IMM2	(Addition)	BAR <= efA(1:0)	Tag detec-tion		SIGN. EXT	S1 <= Dest2				Precharge
(Tag detection)		RB <= (20 : 16)	Match Detection		BI <= S2 IMM2 (Sign Extention)										
		RD <= (25 : 21)	PC1 <= NPC INC	IMM2 IMM1			SAR <= BAR			Dest2 <= EADD (31 : 0)					
				Internal forwarding		PC2 <= PC1	Latch2 <= NPC								
			Latch <= NPC												

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	BI <= SI	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)	AI←IMM2	Shifter <= S2	Shifter (align-ment)								
		RD (25 : 21)		IMM2 <= IMM1			BAR <= EADD (1 : 0)								
			PC1 <= NPC INC					Dcache <= EADD(31:2)							
				Internal forwarding		PC2<= PC1	SAR <= BAR								
			Latch <= NPC												

TRAP(type A)

STAGE 1				STAGE 2				STAGE 3				STAGE 4			
B 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			ALU			Dummy					RF. Write	
			REG. DEC	S1 <= LPC	AI <= S1										Precharge
(Tag detection)				S2⇔R0	BI <= S2			Icache <= trapvector							
		RD <= (25 : 21)	PC1 <= NPC INC		PC3 <= PC2		Dest1 <= Data (31 : 6)	NPC <= trapvector		Dest2 <= Dest1					
			Latch1 <= NPC			PC2 <= PC1									

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	SIGN. EXT	AI←IMM2	ALU	(ADD)							RF. WRITE	
		IMM1	REG. DEC		BI <= PC1										Precharge
(Tag detection)				IMM2 <= IMM1			Dest1 <= PC2	Icache <= EADD (31 : 2)							
		RD <= (25 : 21)						NPC <= EADD (31 : 2)							
			PC1 <= NPC INC							Dest2 <= Dest1					
						PC2 <= PC1									
			Latch1 <= NPC												

(b)

표 4. 데이타 패스에 필요한 제어신호
Table 4. Control signals for data path.

Function Block	Control Signals
Shifter	s1 s2 strb1 Left Right1 Rotate1 Right2 Rotate2 Right4 Rotate4 Right8 Rotate8 Right16 Rotate16 SIGN BO
Mask Gen.	strb2 LOG/ARI M[0 : 4]
IMM1	LOAD1 LOAD2
IMM2	LOAD3 LOAD4 LOAD5 LOAD
SIGN EXT	BAR[1:0] sign VDD sign VSS MSB
SAR	LOAD6 LOAD7 LOAD8
BAR	B LOAD10
ALU	P0 P1 P2 P3 K0 K1 K2 K3 R0 R1 R2 R3 Cin
REG. FILE	READ WRITE WordLA WordLB
REG. Dec.	$\phi 1$ $\phi 2$
PCU	$\phi 1$ $\phi 2$ $\phi 3$ $\phi 4$
LATCH	CS
BUS	BR BW
PSW	psr psw
COMP	ce[0:3]

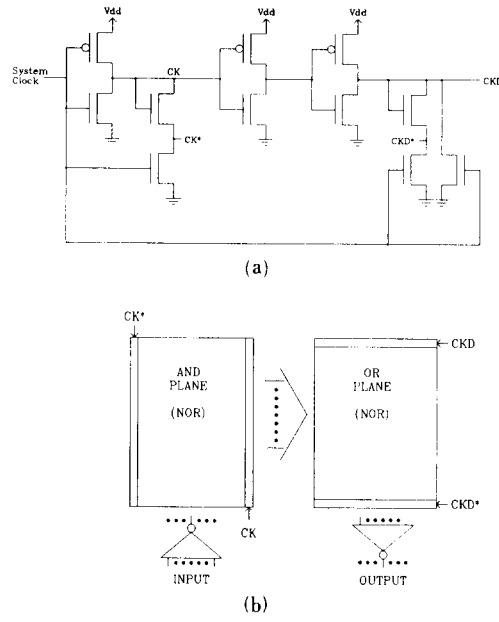


그림 7. (a) DCAL의 클럭 생성기
(b) 디코더의 동작
Fig. 7. (a) Clock generator of DCAL,
(b) Operation of decoder.

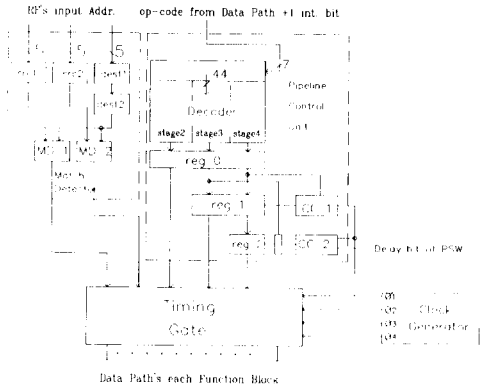


그림 6. 제어부의 구성도
Fig. 6. Block diagram of control unit.

ϕ_4를 변형하여 하나의 phase 내에 디코더 동작을 완료하게 된다.

2) DCAL 회로의 동작

DCAL PLA는 AND평면과 OR평면이 INVERT NOR-NOR-INVERT 구조를 갖는 NMOS들의 병렬 구성으로 이루어지며 이를 효과적으로 구동시키기위

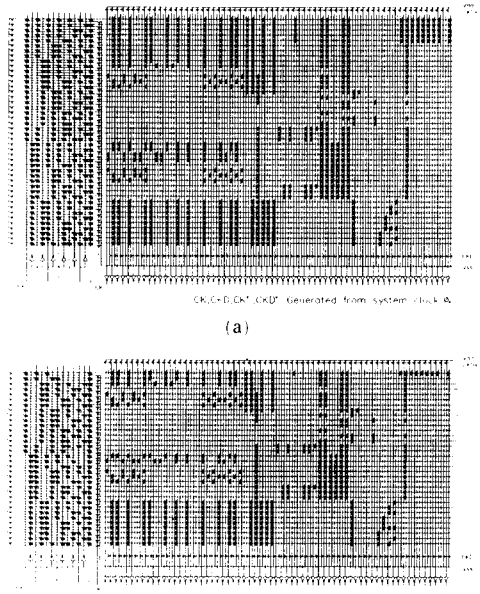


그림 8. (a) 디코더 PLA의 함수표현
(b) 디코더 PLA의 AND평면 논리 최소화
Fig. 8. (a) Functional description of decoder PLA.
(b) AND plane logic minimazation of decoder PLA.

한 내부 클럭 발생회로를 가지고 있다. (그림 7)

NOR-NOR 논리 구조의 2단 다이나믹 CMOS 들의 함수 블럭은 NMOS트랜지스터들이 병렬로 연결되므로 대규모 제어회로에 매우 적합하며 기존의 다이나믹과 스터틱 CMOS PLA 보다 훨씬 빠른 동작속도를 갖는다. 내부 클럭 발생 회로는 evaluation동작시 두개의 클럭이 일정한 지연 시간을 갖도록 하여(CKD, CKD*)다이나믹 CMOS Array 논리회로의 최대 난제인 시차문제(race problem)를 해결하고, 전압을 약간 낮춘 클럭(CK*,CKD*)을 Vdd 쪽의 pre-charge용 PMOS에 제공함으로써 누설 전류와 전하 결합 상태에 의하여 회로의 출력전압과 잡음 여유가 감소되는 문제를 해결하였다. 그러나 내부 클럭 발생기로 인한 레이아웃 면적의 증가는 전체 PLA의 크기에 비해 매우 작은 비율을 차지하므로 부가회로에 의한 면적증가의 문제를 초래하지 않는다.[25]

3) 파이프 라인 제어부

파이프 라인 제어부는 파이프 라인 스테이지에 따라 디코더 PLA를 통해 나온 제어신호를 구분하여 타이밍 회로로 보내주는 역할을 한다. 파이프 라인 제어부는 제어회로와 제어신호를 저장하는 2개의 레지스터로 이루어진다. 그 중 제어신호는 CC1(control circuit)과 CC2로써 각각 데이타 캐쉬 액세스시 사용되는 스테이지와 4-스테이지 파이프 라인의 마지막 스테이지를 제어하게된다. CC1의 경우 디코더에서 데이타 출력신호에 의해 회로가 프리차아지되고 2번째 스테이지의 phase 4에서 신호가 출력되어 디코더 출력이 타이밍 회로에 보내진다. CC2는 PSW (proseccor status word)의 지연비트(delayed bit)가 clear 될때 레지스터를 읽어 내용을 타이밍 회로에 보낸다.

4) Match Detector

HyRISC에서는 하나의 명령어를 처리하면서 한번의 forwarding만을 고려하게 되며, 이러한 하드웨어 forwarding의 가능성을 검토하는 것이 match detector이다. match detector는 페치된 명령어의 Src1, Src2와 전번 명령어 스테이지의 Dest를 비교하여 그림 6에서와 같이 레지스터 파일의 read 신호와 래치 S1 S2를 제어하여 전번 명령에서 계산 결과가 버스 A, B를 통해 직접 정수 실행부로 들어갈 수 있도록 해준다. (그림9)[6] 그림 9의 회로는 S10~S14까지의 내용이 D0~D4와 일치한다면 출력 OUT을 High로 만드는 다이나믹 XOR 회로이며, 이 OUT 신호는 D래치와 S 래치 사이의 패스를 열어주는 역할을 하게된다.

5) 타이밍 회로(timing gate)

타이밍 회로는 앞에서 설명한 디코더, 파이프라인 제어부 그리고 match detector를 통해 나온 제어신호들과 clock generator의 시스템 클럭 신호를 조합하여 적절히 데이타 버스를 제어할 수 있도록 해주며 그 회로는 그림10에 나타내었다.

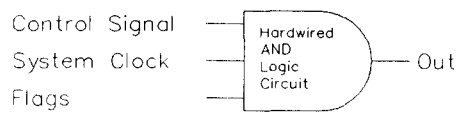


그림10. 타이밍 회로
Fig. 10. Timing circuit.

V. 성능 평가

본 논문에서 제안한 제어부를 2um-double metal 디자인 룰로 레이아웃할 것을 예상, 기본 소자에 대해 Spice 시뮬레이션을 행한 결과를 표 5에 나타내었으며 line 지연과 contact 지연은 고려하지 않았다.

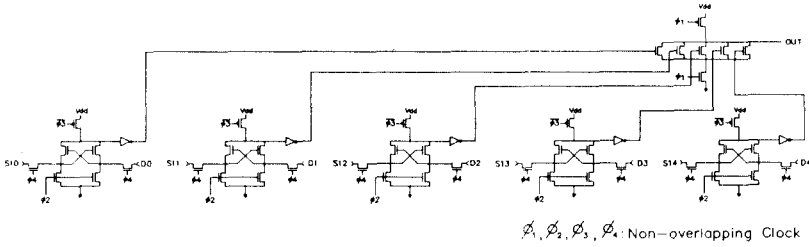


그림 9. Match detector.
Fig. 9. Match detector.

표 5. 기본 소자의 지연시간
Table 5. Delay Time of basic elements.

Gate	Delay Time	Technology
P-MOS	1ns	2um
N-MOS	1ns	
Inverter	1.7ns	Double
2-IN NAND	2.7ns	
2-IN NOR	2.0ns	metal-CMOS
5-IN NAND	5.0ns	

이와 같은 기본 소자의 지연시간을 바탕으로 전체 회로의 schematic capture와 타이밍 시뮬레이션을 Apollo 워크스테이션 DN4000상에서 Mentor Graphics CAD툴을 사용하여 수행 하였으며 얻어진 제어부의 블럭별 지연시간은 표 6과 같다.

표 6. 제어부의 각 블럭별 지연시간
Table 6. Each block's delay time of control unit.

Data Path	TR.	Delay Time
Decoder	1038	6.2ns
M.D.	47	6.4ns
P.L. Cont	265	5.4ns
Timing	270	2.7ns

표 6에서 보년 설계된 제어부는 약 11.6ns에 동작을 완료하게 된다. HyRISC는 17MHz(60ns)에서 동작 하도록 설정되어 있으며 이중 명령어 디코드와 forwarding에 필요한 detection 시간은 각각 4-phase 클럭킹을 사용하는 파이프 라인의 한 Stage에서 1-phase에 해당하는 15ns 씩이 할당 되어있다. 시뮬레이션 결과를 보면 충분한 동작속도를 갖는다. 다음의 표 7은 여타 프로세서들의 기본 사이클 타임을 HyRISC와 비교한 것이다.

Ⅵ. 결 론

본 논문에서는 독자적으로 정의된 명령어들을 바탕으로 on-chip 캐쉬를 사용하여 성능을 향상시킨 고성능 RISC 프로세서인 HyRISC를 효율적으로 제어하기 위한 제어부의 설계에 관하여 논하였다. 제어부의 설계중 명령어 디코더는 기존 CMOS 다단 논리 회로의 단점을 보완한 DCAL을 이용한 PLA로 구현하고 이에 PLAMIN을 적용하여 회로를 단순화 하였다.

표 7. 프로세서들의 기본 사이클 타임
Table 7. Basic cycle time of processors.

Machine	Pipe Line	Cycle Time
RISC II	3	330ns(12MHz)
AM29000	4	40ns (25MHz)
SPARK	4	30ns (33MHz)
MIPS LR300 ALC	5	40ns (25MHz)
MIPS LR2000/16	5	60ns (17MHz)
MIPS-X	5	50ns (20MHz)
SPUR	4	100ns(10MHz)
M88000	4	50ns (20MHz)
i860	4	25ns (40MHz)
HyRISC	4	60ns (17MHz)

디코더를 제외한 나머지 부분인 파이프 라인 제어부, match detector 그리고 타이밍 회로는 각각 명령어에 대한 데이타 패스의 동작과정을 분석하여 가장 효과적인 제어를 할 수 있도록 하드와이어드 로직을 사용하여 높은 속도와 적은 레이아웃 면적을 갖도록 설계하였다. 각 부분의 회로 레벨 시뮬레이션은 회로의 기본 소자인 pass-tr., 인버터 등에 대하여 2μm-double metal 디자인 룰 파라미터를 적용, SPICE 시뮬레이션을 행한후 그 결과 데이타를 이용 Apollo W/S상의 Mentor Graphics CAD tool 들로 수행하였다. 그 결과 HyRISC의 최소 기본 phase인 15nsec내에 안정된 동작을 마칠 수 있음을 확인하였다. 앞으로의 연구과제로는 전체적인 데이타 패스와 함께 타이밍 시뮬레이션을 수행하는 것이다.

參 考 文 獻

[1] Richard S. Piepho, William S. Wu, "A comparison of RISC architecture," *IEEE MICRO Magagine,* August 1989, pp. 51-52.
[2] Manolis G.H. Katenis, "*Reduced instruction set computer architecture for VLSI,*" MIT Press 1985.
[3] *MIPS R2000 Processor User's Manual.*
[4] Robert Waran Sherburne Jr. "Processor design tradeoffs in VLSI," Dissertion for the degree of Doctor of Philosophy of the University of california, Berkeley, 1984. 4.
[5] R.D. Simpson, P.D. Hester, "The IBM RT ROMP processor and memory management unit architecture," *IBM System Jounal* vol. 26, no. 4, pp. 346-360, 1987.
[6] David Lee, "A VLSI chip set for a Multiprocessor workstation," *IEEE SOLID-*

STATE Circuits vol. 24 no. 6, pp. 1688-1698, 12 1989.

[7] Amar Mukherjee, "*Introduction to nMOS& cMOS VLSI System Design,*" Prentice-Hall, 1986.

[8] Carver Mead, Lynn Conway, "Introduction to VLSI system," Addison Wesley Publishing Company, 1980.

[9] Inseok S. Hwang, Aron L. Fisher, "Ultra compact 32-bit CMOS address in multi-output domino logic," *IEEE Jounal of Solid State Circuits* pp. 358-369, April 1989.

[10] Les Kohn, Neal Margulis, "Introducting the intel i860 64-bit microprocessor," *IEEE MICRO Magagine,* August 1989, pp. 15-30.

[11] M. Morris Mano, "Digital logic and computer design," Prentice Hall, 1979.

[12] M. Oprebska, S. Chuquillanqui, H' Derantonia, "PLA and desgn," Design Methodologies, pp. 83-122, 1986.

[13] Nei Weste, Kamran eshraghian, "Principles of CMOS VLSI design a system perspective," Addison-Wesley, 1985.

[14] Rich Goss, Motolola Inc., "*Motorola's 88000: Integration, Performance and Applications,*" *IEEE,* 1989.

[15] Richard A. Blomseth, "A big RISC," *Masters Project final report,* University of California, Berkeley, July 1983.

[16] Sung Mo Kang, "DOMINO-CMOS barrel switch for 32-bit VLSI processor," *IEE CIRCUIT and DEVICE MAGAZINE,* pp. 3-8, May 1987.

[17] Steven M. Rubin, "*Computer Aids for VLSI Design,*" Addison-Wesley, 1987.

[18] 이재민, 고밀도 Programable Logic Array의 설계방식에 관한 연구, 한양대학교 박사학위 논문, 1986.

[19] 홍인식, 신선구, 이승호, 정성호, 임인칠, 32bit 마이크로 프로세서의 controller 설계에 관한 연구, 과학기술처 87 특정연구 결과 발표회 논문집, pp. 177-180, 1989.

[20] 김은성 파이프라인 RISC 아키텍춰의 코드 최적화에 관한 연구, 한양대학교 박사학위논문, 1988.

[21] 홍인식, 이승호, 정성호, 양동준, 김대영, 임인칠, 32-bit RISC CPU의 Shfter 설계, 과학기술처 88 특정연구 결과 발표회 논문집, pp. 377-381, 1989. 7.

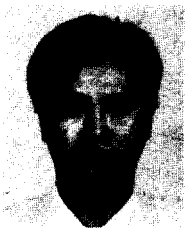
[22] Reduced Instruction set Computer에 관한 연구, 전자통신연구소 최종보고서, 1987. 3.

[23] RISC 프로세서의 Data Path 설계에 관한 연구 전자통신연구소 최종보고서, 1988. 5.

[24] 32-bit RISC CPU의 Shifter 설계에 관한 연구, 전자통신연구소 최종 보고서, 1989. 6.

[25] 한석봉, 임인칠, DYNAMIC CMOS ARRAY LOGIC의 설계, 대한전자공학회 논문지 vol. 26 no. 10, pp. 145-155, 1989. 10.

著 者 紹 介



洪 仁 植(正會員)

1962年 2月 23日生. 1986年 2月 한양대학교 전자공학과 졸업. 1988年 2月 한양대학교 대학원 전자공학과 졸업 공학 석사 학위 취득. 1988年 3月～현재 한양 대학교 대학원 전자공학과 박사과정 재학중. 주 관심분야는 RISC Processor Design, Computer Architecture. 고속 연산 장치, Layout 등임.

林 寅 七 (正會員) 第25卷 第8號 參照
현재 한양대학교 전자공학과 교수