論 文

IDL을 이용한 16-비트 SIP의 설계와 시뮬레이션에 관한 연구

正會員 朴 斗 烈* 正會員 李 鐘 憲**

A Study on the Design and Simulation of 16-bit SIP by using IDL

Doo Youl PARK* Jong Heon LEE** *Regular Members*

要 約 본 논문에서는 16-비트의 소형명령의 프로세서를 설계할때 IDL로서의 APL를 사용하고 있다. 그것은 다른 HDL들이 갖지못하는 하드웨어의 구조를 표현하고 기술하는 것이 가능했다.

여기서는 프로세서를 설계할때 전체적인 시스템을 모듈별로 분리하여 설계하였기 때문에, 직접 디코딩 방법을 선택하였다. 설계된 각 모듈들은 실험체제를 통하여 입력된 12-비트의 제어워드에 따라 실행되며, 그 실험체제는 기호화된 명령어들로 구성된다.

여기서, 2진코드를 사용하여 SIP의 명령코드를 세팅함으로써, 명령형식과 어셈블러 명령을 구성했고, 실험체제를 통하여 제시된 명령어 세트를 입력함으로써 SIP의 동작을 확인했다.

제시된 SIP에서는 입력하는 프로그램이 기호화된 언어이기 때문에 설계자나 사용자가 시스템의 동작을 쉽게 이해할 수 있을 것이다. 특히, SIP내에서 유니트함수를 임의로 정의할 수 있기 때문에 유니트함수의 사용에 제한을 받지않고 다양하고 쉽게 호출할 수 있을 것이다.

ABSTRACT In this paper, We use the APL as IDL when simulation a 16-bit SIP. It was possible for IDL to represent and describe a structure of a H/W which other HDL have not.

Because We partitioned whole system to various modules when desingning processor, We adopted a direct decoding method. A designed each modules are executed according to 12-bit control word was inputed through experimental framework, Which were composed to symbolized instructions.

In here, By setting instruction codes of the SIP using binary code, We composed instruction format and assembler instruction, and verified the SIP behaviour that try to implement by entering a presented instruction set through experimental framework.

In a presented SIP, Because inputing program are a symbolized language, Designer and user will easily understand behaviour of system. Especially, Because we can immediatly specify a uint function within SIP, We wil use variously and easily the library cell.

*東州女子專門大學 電子計算科
Dept. of Computer Science Dong Ju Women's Junior College
**東亞大學校 電子工學科 教授
Dept. of Elec. Eng. Dong-A Univ.
論文番號 : 1990-05(接受1989. 8. 30)

I. 서 론

근래에 컴퓨터 하드웨어(Computer Hardware)를 가지는 디지탈 시스템이나 MSI, LSI, VLSI와 같은 집적회로를 설계하고 시뮬레이션(Simulation)하기 위해서 여러가지의 하드웨어 기술언어(H/W Description Language; HDL)[1-6]들이 연구되어 왔으며, 또 응용되고 사용[7-28]되어왔다. 더우기 고급언어(High Level Language; HLL)를 이용하여 그런 디지탈 시스템을 설계하고 기술하여 시뮬레이션하는 기법들도 연구[29-33]되어왔다. 일반적으로 이들 HDL들을 이용하여 시스템을 기술하는데는 각부의 동작을 알고리듬(Algorithmic), 레지스터 전송(Register Transfer), 메모리 스윗치(Memory Switch), 스윗칭 회로(Switching Circuit), 인스트럭션적(Instructional), 그리고 아날로그 회로(Analog Circuit) 등의 레벨로 규정하고 있다.[33-35] 여기서 이 6가지의 레벨에서 하드웨어를 기술하는 HDL들을 여러가지 설계프로그램과 설계자가 사용하는 이들 설계도구들 사이에서 상호간의 공통매체로 사용되며, 시스템을 소부분으로 분리하여 기술할때 소부분의 입출력을 위한 언어로 사용되며, 설계결과의 확인, 성능의 평가, 오류의 테스트세트의 생성및 논리설계의 프로그램에 적용되어 왔다.

이런 하드웨어 기술언어들은 구조를 기술하기보다는 주로 정확한 동작을 기술하는 데에 도움을 주는 것[34-36]이라고 보여지며, CDL(Computer Description Language)은 시스템의 시뮬레이션에 필요한 마이크로 프로그래밍의 생성에 적합하다고는 할 수 있으나 이들은 논리적 다이아그램의 레벨의 기술을 만들때 번역기의 스트링(String) 출력을 사용하기가 어렵다고 보았다. 또 ISP (Instructional Set Processor)는 논리생성(Logic Generation)레벨을 표현하지 못하는 점이 있다고 여겨진다[33-34]. 그리고 AHPL은 직접회로망에서 마스크의 개발을 위해 완전하게 자동화된 설계처리[37-38]를 하고자 할때 아주 간편하면서도 편리한 입력도구가 되고 있다. 그러나 구성한 시스템을 동작시키기 위한 시뮬레이션을 할때 통신부(Communication Section)에 숫자만 입력되며, 시스템을 구조적으로 기술하여 프로그램하기도 어렵다고 하였다[33-34]. 위에서 설명한 바에 따른 HDL들은 하드웨어 시스템의 구조를 기술하기 위한 프로그램을 기계적으로 작성하기가 어렵다고 볼 수 있을 것이며, 시뮬레이션을 하기위한 실험체제의 실현을 기호화하기가 어려운 점이 있다고 보여진다.

본 논문에서는 컴퓨터나 디지탈회로를 설계하고 기술하여, 시뮬레이션하기 위해 시스템의 구조적 기술을 가능[29]하도록 해주고 있고, 기호화된 언어에 의한 실험체제의 실현이 용이한 IDL(Interactive Design language)[29-32]로 APL[39-41]을 사용하고 있다. 역시 이 언어는 논리게이트나 플립플롭의 동작에 대응하는 논리연산자들을 이미 포함하고 있거나, 간단히 정의하여 사용할 수 있기 때문에 논리생성 레벨을 정확하게 표현할 수 있을 것으로 보았고 동작확인을 위해서 실현된 명령을 본 연구에서 제시한 실험체제(Experimental Framework)를 거쳐서 입력시킴으로써 입력프로그램의 작성을 용이하게 하고 스트링(String)출력을 쉽게 얻을 수 있을 것으로 지목했다. 또 그 언어는 대화용 언어(Interactive Language)이므로 시스템과 기호화된 명령어와 실험체제를 실현하기 위한 프로그램을 작성하기가 아주 용이하므로 시스템을 간결하고 상세하게 기술할 수가 있을 것이며, 표현되는 기능들을 나타내는 연산자와 그것의 전송기호가 하드웨어 구조및 동작에 대한 논리 표현식과 대응성을 갖게 하였으므로 시뮬레이션 역시 간편하게 할 수 있을 것이라고 내다 보았다.

종래의 HDL들로 구성한 시스템[8,10,38]이나, 컴파일러 언어로 구성한 시스템[30,31]들의 동작을 확인하고, 결과를 얻기 위한 입력자료가 숫이(2,8,10,16 진수)뿐이었다. 반면에 여기서 저자는 IDL로 기호화한 언어를 고안하여 실험체제를 구축하였고 그 실험체제를 거쳐서 언어를 입력하도록 하여서 입력되는 자료가 시스템의 무슨 동작을 위한 입력자료인지를 쉽게 알 수 있도록

시도하였으며, 또 입력된 프로그램의 결과를 프로그램 카운터 마다 2진스트링으로 출력하도록 시도하여 동작확인하기가 용이하도록 하였다. 역시 여기서는 설계자가 유니트함수를 임의로 정의해 두고 호출할 수 있게하여 유니트함수의 사용에 제한을 받지않고 그들을 다양하고, 효과적으로 사용할 수있게 하려고 시도하였다.

이어서 저자는 시스템설계자가 응용하고자 하는 분야에 맞게끔 적합하게 설계할 수 있는 소형명령을 갖는 프로세서(Small Instructional Processor; SIP)를 제어순차(Control SEquence)에 따라 동작하도록 설계하여 논리게이트와 플립플롭에 근거를 둔 스위칭 회로레벨에서 기술하고, 기술한 시스템에 고안한 언어를 입력하여 시뮬레이션하고 그것의 동작과 결과를 고찰하였다. 먼저 설계에 있어서 대상의 전체적인 블록도를 구축하고, 그것을 동작별로 소부분의 시스템으로 분리하여 회로를 설계한후 IDL로 각 회로의 구조를 논리적인 함수로 기술하고, 시뮬레이션하도록 하겠다.

본론에서는 SIP의 전체적인 동작의 흐름을 위한 블록도를 구축하여 시스템의 동작을 해석토록 하였고 3장에서는 프로세서의 각 부분별 회로를 도시하여 논리적인 함수에 대한 함수식을 기술하였고, 이어서 부분별 회로를 동작시키는 동작 코드를 기호화하고 명령형식을 설정하여 프로세서의 동작을 시험하기 위한 실험체제를 제시하여 실현하도록 하였다. 이어서 4장에서는 IDL을 이용한 프로세서의 설계알고리듬(Design Algorithm)을 기술하여 시뮬레이션하고 5장에서는 동작코드를 기호화한 명령을 이용하여 구성한 프로세서를 실행시키기 위한 프로그램을 실험체제를 통하여 입력하는 예를 들겠다.

끝으로 본 연구에서 시스템을 설계하기위해 사용한 IDL의 특징을 설명할 것이며, HDL이나 고급언어로 설계한 시스템과 본 연구에서 제시한 실험체제를 거쳐서 입력자료를 넣어줌으로써 시뮬레이션된 SIP를 비교하여 그것의 유용성을 밝히겠다.

Ⅱ. 16-비트의 소형명령의 프로세서의 구성과 동작의 흐름

본 연구에서 제시하는 프로세서는 16-비트이며, 그것의 전체적인 블록의 구성은 〈Fig. 2-1〉에 주어진다. 여기서는 레지스터, 레지스터선택기, ALU, 자리이동, 레지스터, 누산기, 디코더, 그리고 상태비트 레지스터로써 구성하였다. 그림에서 보여주는 4개의 16비트 범용레지스터(General Register)는 처리장치에서 처리되는 데이타가 기억되어있는 레지스터이며 출발지(Source)레지스터이다. 이 레지스터의 출발동작은 멀티플렉서 그룹A를 위해 필요되는 1개 혹은 2개를 선택하는 레지스터 선택회로에 의해 수행된다. 이 회로에서 선택단자의 제어코드 S0,S1,S2,S3는 한번의 처리를 위해 제공되는 제어워드(Control Word)중의 4-비트이다.

선택기에 의해 선택된 1개 혹은 2개의 16비트 데이타는 각각 A-BUS와 B-BUS에 연결되어

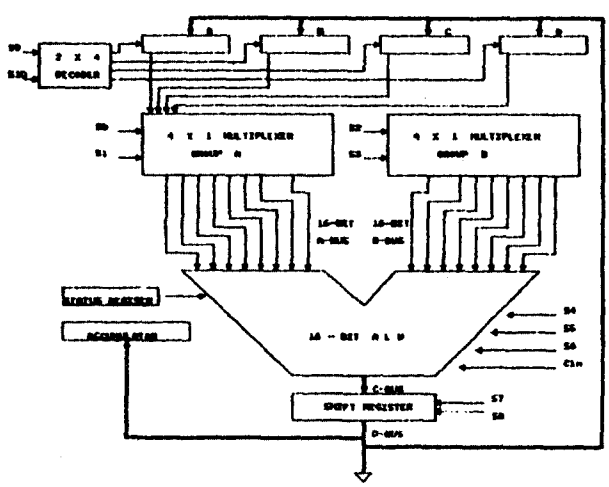

Fig.2-1 16-BIT Processor Block Diagram

산술및 논리연산장치(Arithmetic and Logic Operation Unit; ALU)에 입력되어 처리된다. 이때 ALU에서 처리하는 기능은 S4,S5,S6,Cin의 선택코드의 비트값에 따라 산술연산 8개와

논리연산 4개로 분류되며 산술연산과 논리연산의 구분은 S4의 값에 따라 분류된다. 여기서 Cin은 산술연산에서 필요로 되는 초기의 입력케리(Carry)이며 논리연산을 행할 때에는 무관조건(Redundancy)으로 처리된다. 따라서 여기서 구성한 ALU는 전가산기를 사용하여 산술연산과 논리연산을 겸한 처리장치로서 사용된다.

이 ALU에서 처리된 결과는 C-BUS에 실려 16-비트 자리이동 레지스터(Shift Register)에 인가된다. 이 16-비트 자리이동레지스터는 병렬로더가 가능하며 양방향(좌우측)자리이동이 가능한 레지스터이다. 자리이동 레지스터에서의 기능은 4가지가 있으며 그 기능은 선택코드 S7,S8에 따라 결정된다. 자리이동 레지스터에서 나온 데이터는 D-BUS에 실려 누산기(Accumulator: AC)로 가고 또 제어워드의 값에 따라 목적지 레지스터로 전송된다. 이때 목적지레지스터의 선택은 4개의 레지스터중 1개에 로더신호를 주기 위한 2X4 디코더(Decoder)에서 결정한다. 즉 이 디코더의 선택코드 S9,S10의 값에 의해 4개의 출력중 어느 하나가 1이되며 이때의 1신호가 로더신호로서 목적지 레지스터를 선택하게 된다.

상태비트 레지스터(Status Bit Register)는 4-비트 레지스터로서 오버플로비트(Overflow Flag Bit), 제로비트(Zero Flag Bit), 부호비트(Sign Flag Bit), 캐리비트(Carry Flag Bit)로 구성된다. 오버플로비트 V는 ALU에서 처리된 16비트의 데이타중 15번째 비트를 처리할 때 발생한 캐리 C16과 16번째 비트를 처리할 때 발생한 캐리 C17를 E-OR(Exclusive-OR)하여 그 결과에 따라 V플레그 비트가 세트 리세트되고, 제로플래그비트 Z는 ALU에서 처리한 16-비트의 데이타가 전부 0인지, 아니면 단 한 비트라도 1이 있는지에 따라 세트 리세트된다. 부호플래그비트 S는 처리된 16비트의 데이타중 최상위비트(Most Significant bit; MSB)값이 1인지 0인지에 따라 세트 리세트되고, 마지막으로 캐리플래그비트 C는 최종캐리 C17로 볼수 있으므로 C17의 비트값에 따라 그 플레그가 세트 리세트된다.

Ⅲ. SMALL INSTRUCTIONAL PROCESSOR의 조합회로및 순차회로의 구성

이 장에서는 프로세서의 각 구성회로의 설계와 논리함수식을 유도한다. 여기서 모든 순차회로는 D-플립플롭으로 구성하고있다.

Ⅰ. 16-비트 범용레지터

레지스터 로더신호는 〈Fig.2-1〉의 2×4 디코더의 출력중 1이 되는 값을 각각의 레지스터의 로더신호로 사용한다. 설계된 회로는 4비트로서 〈Fig.3-1〉에 보여 주었다.

〈Fig.3-1〉의 회로에서 4개의 레지스터회로 A,B,C,D 에 대한 입출력의 상태를 표시하는 상태방정식(State Equation)을 i번째 비트에 대해 유도해 보면 식(3-1)처럼 기술된다. 아래의 식에서 사용되는 연산자는 APL의 연산자이며, 그들의 설명은 부록에 보여진다.

$$Ai(t+1) \rightarrow (\sim Sl)\ \triangle Ai(t)\ \triangle DBi\ \triangle Sl \quad (3\text{-}1)$$

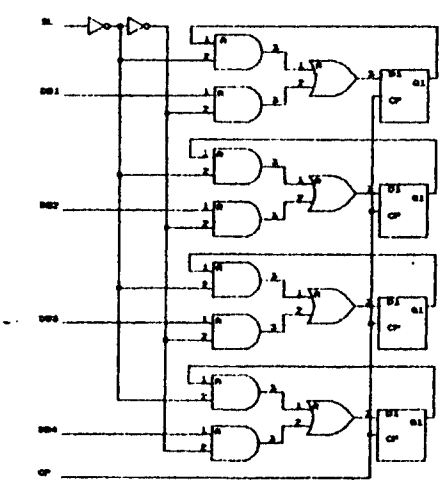

Fig.3-1 4-bit Parallel Load Register Using D-FF

여기서 Ai(t+1)은 i번째 D-FF의 다음 상태(Next-State)값이고 Ai(t)는 i번째 D-FF의 과거에 기억되어 있던 값이다. 또 SL은 디코더에서 만들어진 레지스터 로더신호이며, DBi는 16-비트 D-BUS의 i번째 비트값이다.

Ⅱ. 출발지 레지스터 선택기.

프로세서내의 4개의 레지스터중 하나를 처리하기 위해서는 먼저 레지스터를 선택하여 버스에 연결해야 하는데, 그것을 행하기 위해서 4X1 멀티플렉서가 필요하고 레지스터는 16-비트이므로 이런 멀티플렉서가 16개 필요하게된다. 이 동작을 위한 이 레지스터 선택기는 〈Fig.3-2〉에 보여진다.

레지스터 선택기로서의 멀티플렉서에서 선택코드는 전체 블록도에서 보여주었듯이 S0,S1과 S2,S3이다. 이 선택코드의 값에 따라서 4개의 레지스터중 하나를 선택하게 된다. 〈Fig. 3-2〉로

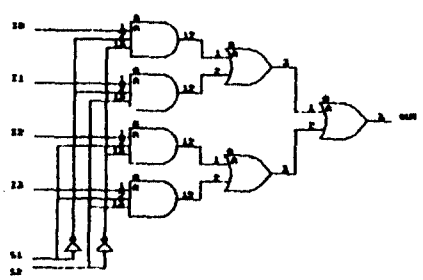

Fig.3-2 Source Register Selecter

부터 이 레지스틱 선택기의 논리함수식을 유도해 보면 식(3-2)와 식(3-3)과 같이 기술된다.

$$Y1 \leftarrow I0 \triangle (\sim S1) \triangle (\sim S0) \nabla I1 \triangle (\sim S1) \triangle S0 \nabla I2 \triangle S1 \triangle (\sim S0) \nabla I3 \triangle S1 \triangle S0 \quad (3\text{-}2)$$

$$Y2 \leftarrow I0 \triangle (\sim S3)(\sim S2) \nabla I1 \triangle (\sim S3) \triangle S2 \nabla I2 \triangle S3 \triangle (\sim S2) \nabla I3 \triangle S3 \triangle S2 \quad (3\text{-}3)$$

여기서 Y1은 멀티플렉서 그룹A에서, 그리고 Y2는 멀티플렉서 그룹B에서 선택된 것이다.

Ⅲ. 16-비트 산출및 논리연산장치(ALU)의 설계

여기서는 ALU를 병렬가산기(Parallel Adder)를 이용하여 산술연산과 논리연산을 겸하는 회로로 실현하고 있다. 먼저 산술연산에 대한 기본회로의 입력구성을 연산형태에 따라 8가지로 분류할 수 있다. 분류된 기능은〈Fig. 3-3〉에 보여진다.

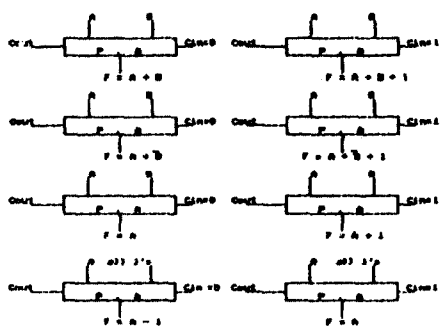

Fig.3-3 Operation of obtaining by controling inputs of parallel Adder

〈Fig.3-3〉을 선택코드 S5, S6으로 제어하여 조합회로를 실현하면 〈Fig.3-4〉처럼 실현된다.

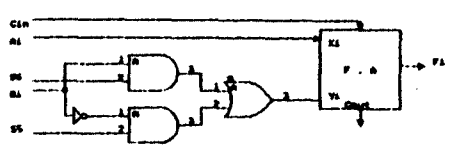

Fig.3-4 Implementation of 1-bit Arithmetic Unit using control code S5,S6

〈Fig. 3-4〉의 회로를 논리함수식으로 표현하면 식(3-4)로 기술된다.

$$Xi \leftarrow Ai \quad (3\text{-}4\text{-}1)$$

$$Yi \leftarrow Bi \triangle S6 \nabla (\sim Bi) \triangle S5 \quad (3\text{-}4\text{-}2)$$

여기서 Xi와 Yi는 1-비트 연산회로의 전가산기의 두 입력이다. 식(3-4)로 부터 〈Fig. 3-4〉의 회로기능표를 제어코드 S5,S6의 값에 따라 표시한 것은 〈Table 3-1〉에 보여진다.

Table 3-1 Function selection of control code S5,S6 and Cin for ⟨Fig. 3-4⟩

기능선택 S5 S6 Cin	Y	출 력	기 능
0 0 0	0	F=A	A를 전송하라
0 0 1	0	F=A+1	A를 증가시켜라
0 1 0	B	F=A+B	A에 B를 더하라
0 1 1	B	F=A+B+1	A+1에 B를 더하라
1 0 0	B	F=A+$\bar{B}$	A에 B의 1의 보수를 더하라
1 0 1	B	F=A+$\bar{B}$+1	A에 B의 2의 보수를 더하라
1 1 0	1	F=A−1	A를 감소시켜라
1 1 1	1	F=A	A를 전송하라

여기서 논리연산기능 4가지(OR,AND,NOT,EOR) 기능은 전가산기에서 입력캐리를 0으로 두고 제어코드 S4를 모두 1로 둠으로써 산술연산회로로 부터 얻어진다. 먼저 전가산기에서 합을 구하는 식(3-5)

Fi←Xi ⊕Yi ⊕Ci (3-5)

에서 Ci=0이면

Fi←Xi⊕Yi (3-6)

이 되어 논리연산 EOR가 얻어진다. 또 OR연산은 산술연산에서 S5,S6=00이면 A를 ALU의 출력으로 전송하는 것이다. 여기서 Ai를 Ai▽Bi로 대치함으로써

Fi←Ai▽Bi (3-7)

이 되고 S5,S6=10이면

Fi←Ai ⊕(∼Bi)=Ai△Bi▽(∼Ai)△(∼Bi)

=Ai ⊙ Bi (3-8-1)

이 된다. 여기서는 Ai를 Ai▽$\bar{Bi}$로 대치하면

Fi←Ai△Bi (3-8-2)

이 되어 AND연산이 되고 5,S6=11이면

Fi←Ai⊕1=∼Ai (3-9)

가 되어 NOT연산이 된다. 따라서 식(3-4-9)로부터 산술연산및 논리연산을 겸한 회로의 입력 논리함수식을 다음 식처럼 기술해낼 수가 있다.

Xi←Ai▽S4△(∼S5)△(∼S6)△Bi

▽S4△S5△(∼S6)△(∼Bi) (3-10-1)

Yi←S6△Bi▽S5△(∼Bi) (3-10-2)

여기서 Ai와 Bi는 연산에 사용되는 두개의 데이타이고 Xi와 Yi는 ALU에서 1-비트 연산을 위한 전가산기의 i 번째 입력이며 Zi는 i-번째 입력캐리로 된다. 식(3-10)으로 부터 1-비트 전가산기의 입력논리회로를 실현하면⟨Fig. 3-5⟩로 구성된다.

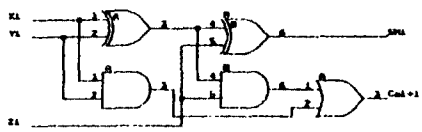

Fig.3-5 Implementation of 1 bit full-Adder by control code S4,S5,S6, and Cin

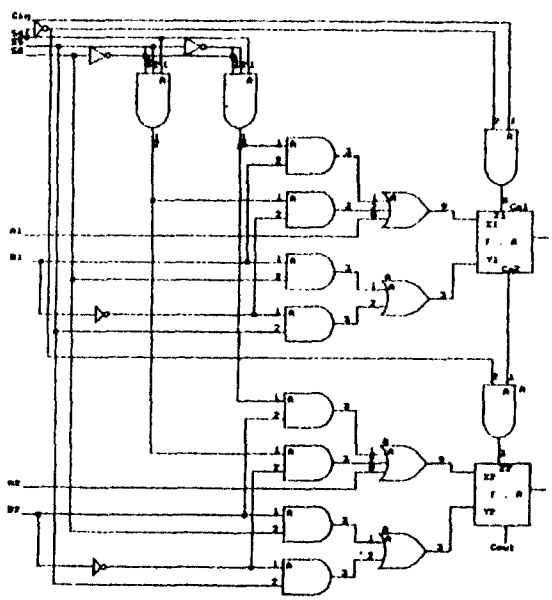

Fig. 3-6 Implementation of ALU using Full-Adder

⟨Fig. 3-5⟩로부터 i-번째 비트의 ALU출력 Fi와 출력캐리는 식(3-11)로 기술된다.

Fi←Xi⊕Yi⊕Ci (3-11-1)

Ci+1←(Xi⊕Yi)△Zi▽Xi△Yi (3-11-2)

위의 식(3-11)은 ALU의 1-비트만을 기술하는 식이며 이것을 전체 16-비트로 실현하면 〈Fig. 3-6〉으로 된다. 〈Fig. 3-6〉으로 실현된 16-비트 ALU에서의 12가지 연산들이 〈Table 3-2〉에 정리되어있다. 각각의 기능은 S4,S5,S6 and Cin에 의해서 선택되며, 선택된 기능에 의한 처리결과는 ALU의 출력F로 전송된다.

〈Table 3-2〉 Function of ALU by Control code S4,S5,S6 and Cin

선택코드 S4 S5 S6 Cin	출 력	기 능
0 0 0 0	F=A	TRANSFER A
0 0 0 1	F=A+1	INCREMENT A
0 0 1 0	F=A+B	ADDITION A, B
0 0 1 1	F=A+B+1	ADDTION WITH CARRY
0 1 0 0	F=A+$\overline{B}$+1	SUBTRACT WITH CARRY
0 1 0 1	F=A+$\overline{B}$	SUBTRACT
0 1 1 0	F=A-1	DECREMENT
0 1 1 1	F=A	TRANSFER A
1 0 0 X	F=A▽B	OR OPERATION
1 0 1 X	F=A⊕B	E-OR OPERATION
1 1 0 X	F=A△B	AND OPERATION
1 1 1 X	F=$\overline{A}$	NOT OPERATION

Ⅳ. 상태 레지스터(Status Register)

이 프로세서에서 처리되고 있는 데이타는 부호가 있는 수들이기 때문에 처리된 결과에서 그 값이 음수인가, 양수인가 또는 차의 값이 0인지, 아닌지, 오버플로가 발생했는지 하지 않았는지, 또는 캐리가 발생했는지 안했는지 하는 등의 상태를 나타내는 상태레지스터(Status Register)가 있다. 이 상태레지스터는 V,Z,S,C로 표시되며 각각은 아래의 4가지 조건에 따라 세트-리세트된다.

가) 상태 비트C는 캐리비트로서 ALU의 출력캐리가 1이면 세트되고 0이면 리세트된다.

나) 캐리 비트S는 부호비트로서 ALU의 출력의 최상위비트가 1이면 세트되고 0이면 리세트된다.

다) 제로 비트Z는 제로비트로서 ALU의 출력16비트 모두가 0이면 세트되고 단1비트라도 1이 있으면 리세트된다.

라) 비트V는 오버플로비트로서 C16과 C17을 배타논리합(EOR)한 결과가 1이면 세트되고 0이면 리세트된다.

위의 4가지 조건을 조합하여 논리회로로 실현하면 〈Fig. 3-7〉처럼 실현된다.

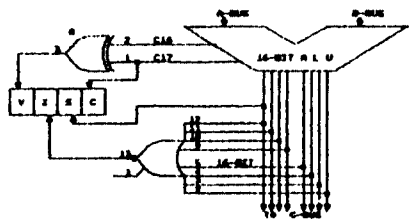

Fig. 3-7 Implementation of Status Register

5) 자리이동 레지스터(Shifte Register)

ALU에서의 출력은 16-비트의 C-BUS에 실려 자리이동 레지스터로 전송된다. 이 자리이동 레지스터는 4가지의 기능을 갖게되며 이 4가지 기능을 실행하도록 하기 위해서는 병렬로더가 가능한 양방향자리이동 레지스터로 실현되어야 한다. 여기서, 4가지의 기능은 제어코드 S7,S8에 따라 구분되는데, 이때 S7S8=(00)이면 입력데이타를 변화없이 출력으로 전송하고, S7S8=(01)이면 입력 데이타를 우측으로, S7S8=(10)이면 입력 데이타를 좌측으로 각각 1-비트 자리이동하여 전송하고, S7S8=(11)이면 출력측에 아무런 변화도 주지 못한다. 이 동작으로부터 16-비트 양방향 자리이동 레지스터를 실현하면 아래의 〈Fig. 3-8〉로 실현된다.

6) 목적지 레지스터 선택기

자리이동 레지스터의 출력은 D-BUS에 실려 목적지 레지스터 A,B,C,D중 어느 하나에 전송되게 되어있는데, 이때 목적지 레지스터를 선정하는 회로가 디코더이다. 여기서 4개의 레지스터중 하나를 선택하여 로더신호를 주어야 하기 때문에 2×4디코더로 실현되어야 한다. 이를 위한 2개의 입력은 제어코드 S9,S10이 되며 이 코드의 값에 따라 레지스터가 선택된다. 즉, S9S10=(00)이면

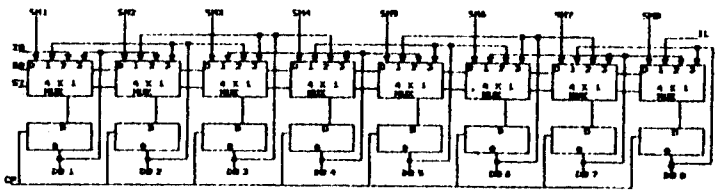

Fig. 3-8 Implementation of Bidirectional Shift Register

목적지 레지스터로서 A를, S9S10=(01)이면 B를, S9S19=(10)이면 C를, S9S10=(11)이면 D를 선택한다.

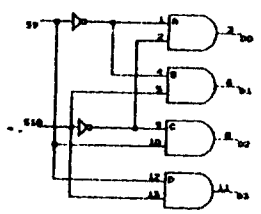

Fig.3-9 Destination Register Selecter

이 레지스터 선택기는 〈Fig. 3-9〉에 보여진다. 예를 들면 S9,S10=00이면 D0=1이 되며 이것이 로더신호가 되어 레지스터A를 로더하게 되고 D-BUS의 데이타는 레지스터A로 전송된다.

7) 12-비트의 소형명령의 코드세트를 이용한 실험체제의 실현.

여기서는 실험체제를 실현하기 위해 명령코드를 세트하여 실현하는 과정을 보여준다. 지금까지 설명한 부분별 회로를 동작시켜주는 각 선택코드들을 동작의 순서에 따라 조합하여 프로세서가 1개의 명령에 의해 1번 동작하는 제어워드(Control Word)를 구성해 보면

Multiplexer group A	; S0 S1
Multiplexer group B	; S2 S3
ALU Function Select	; S4 S5 S6 Cin
Shift Function Select	; S7 S8
2 X 4 Decoder	; S9 S10

으로부터 제어워드 CW는 S0,S1,S2,S3,S4,S5,S6,Cin,S7,S8,S9,S10의 12-비트로 표시된다. 여기서 S0,S1과 S2,S3는 출발지 레지스터를 선택하는 것이므로 오퍼란드로 간주되며 또 S9,S10도 목적지 레지스터를 선택하는 것이므로 역시 오퍼란드로 간주된다. 따라서 실제적인 명령의동작코드는 ALU의 기능 선택을 위한 제어코드S4,S5,S6,Cin과 자리이동 레지스터의 기능 선택을 위한 제어코드 S7,S8을 조합하여 6-비트로 표시하였다. 이 6-비트의 동작코드에 대해 기호화된 명령은 〈Table. 3-3〉에 보여진다.

Table.3-3. Mnemonic Instruction by control code S4,S5,S6,Cin,S7,S8

S4 S5 S6	Cin	S7 S8	OPERATION CODE	HEX-CODE	INSTRUCT-ION
0 0 0	0	0 0	00 0000	0 0	TRA
0 0 0	0	0 1	00 0001	0 1	SHR
0 0 0	0	1 0	00 0010	0 2	SHL
0 0 0	1	0 0	00 0100	0 4	INC
0 0 1	0	0 0	00 1000	0 8	ADD
0 0 1	1	0 0	00 1100	0 C	ADC
0 1 0	0	0 0	01 0000	1 0	SBC
0 1 0	1	0 0	01 0100	1 4	SUB
0 1 1	0	0 0	01 1000	1 8	DEC
1 0 0	0	0 0	10 0000	2 0	LOR
1 0 1	0	0 0	10 1000	2 8	EOR
1 1 0	0	0 0	11 0000	3 0	AND
1 1 1	0	0 0	11 1000	3 8	NOT

〈Table. 3-3〉의 명령코드의 세팅은 APL언어에 의하여 시뮬레이션하였다. 이렇게 하여 구성한 기호화된 명령은 한개 혹은 두개의 출발지 레지스터의 데이타를 갖고와서 처리하고 그 다음 결과를 1개의 목적지 레지스터로 전송하게 되는데 이때 출발지 레지스터를 지정하는 코드와

목적지 레지스터를 지정하는 코드와 또 동작코드가 모두 조합되어 완전한 명령형식을 이룬다. 이 명령형식은 〈Fig. 3-10〉에 보여진다.

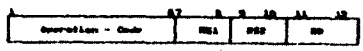

Fig.3-10. Instruction Format

〈Fig. 3-10〉에서 RS1,RS2와 RD는 레지스터를 지정하는 정보의 비트로서 레지스터A,B,C,D를 지정한다. 따라서 이것을 완전한 명령으로 표시하면 OP-CODE,RS1,RS2,RD이다. 예를 들어 RA레지스터의 내용과 RB레지스터의 내용을 더하여 RC레지스터에 전송하는 명령을 실행하기 위한 프로그램은(ADD,RA,RB,RC)로 작성된다. 이 명령을 2진코드로 표시하면 (0010 000 0110)으로 조합된다. 여기서 각 제어코드를 조합하는 것은 APL의 Catenation동작에 의해 수행된다. 여기서 구성한 명령어 코드는〈Fig. 3-10〉에서 보여지듯이 12-비트의 폭을 가지고 있다. 이 기호화된 명령어 코드는 프로그램 작성 순서에 따라 12비트의 ROM에 차례로 기억되며 작성한 프로그램을 실행시킬때는 구성한 컴퓨터 프로세서에 작성한 프로그램을 입력하여 프로그램 카운터의 순서에 따라 차례로 처리된다. 여기서 명령어 코드중 오퍼란드도 ROM에 기억되도록 하였으며 명령에 사용되는 실제 자료는 레지스터들로부터 명령에 코드의 선택코드 비트에 의해 선택되도록 하였다. 물론 이때의 처리한 결과를 보관하는 목적지 레지스터의 선택도 역시 명령어 코드 비트에 포함되어 있다. 그리고 명령어 코드 세트를 기술하여 실험체제를 실현하는 프로그램을 부록에 보여진다.

Ⅳ. IDL에 의한 시뮬레이션

16-비트 프로세서의 시뮬레이션을 위한 알고리듬은 아래의 순서에 따라 실현된다. 여기서 IDL로서 사용되고 있는 APL은 문헌[39,40,41] 에 보여진다.

과정1) 먼저 각 레지스터 A,B,C,D의 초기치를 아래처럼 기술하여 부여한다.

```
IMA←LDA INITIAL-DATA
A←16ρIMA←0
A←16ρ ((16ρ2)γDATA)
```

이것은 A레지스터에 초기치를 부여하는 것으로 다른 3개의 레지스터에 대해서도 동일한 방법으로 초기치를 부여하여 이 초기치를 주는 유니트 함수는 구성한 하드웨어 프로세서에 입력하는 프로그램으로 부터 호출되어 사용된다.

과정2) 실행시키고자 하는 명령의 제어워드를 프로그램 카운터가 지정하는 ROM의 번지로부터 읽혀져 와서 명령 레지스터 IR로 로드되고 이것을 12-비트의 CW를 통하여 마이크로 프로세서에 입력한다. 이것은 아래 처럼 기술될 수 있다.

```
ROMP ◇ CW← IR

SO ←CW[7]◇S1←CW[8]◇S2←CW[9]◇S3
←CW[10]◇S4←CW[1]◇S5←CW[2]
S6←CW[3]◇CA1←CW[4]◇S7←CW[5]◇S8
←CW[6]◇S9←CW[11]◇S10←CW[12]

→((△/CW)=1)/HALT
```

여기서 ROMP는 프로그램 카운터가 지정하는 ROM의 번지의 명령을 IR로 로드하는 동작을 위한 유니트 함수이다. 그리고 두번째 프로그램은 CW의 12-비트를 각각의 선택 코드에 할당하는 것이며 CA1은 ALU의 초기캐리이다. 세번째 프로그램은 HLT명령을 실행하는 것으로 HLT 명령의 제어워드 12-비트 모두가 1이므로 모든 비트를 AND하여 1이면 프로램의 실행을 중지하게 되고 그렇지 않으면 하드웨어 프로세서를 계속 동작시키게 된다.

과정3) 각 레지스터에 기억된 16비트값은 제어워드의 비트값에 따라 선택되기 위하여 각각 치환된다. 즉

Ai←A[i], Bi←B[i], Ci←C[i], Di←D[i]로 되어 4×1 멀티플렉서 그룹1과 2에 연결된다.

과정4) 멀티플렉서 그룹1과 그룹2에 연결된 4개의 레지스터들이 A-BUS에 실리기 위해서는 과정1)에서 받아들여진 S0,S1값에 따라 결정되며 B-BUS에 실리기 위해서는 S2,S3값에 따라 결정된다. 따라서 제어워드중 S0,S1,S2,S3와 식(3-2)와 식(3-3)으로부터 각각 아래 처럼 표시된다.

MAi←(Ai△(~S0)△(~S1)▽(Bi△(~S0)△S1)▽(Ci△S0△(~S1)▽(Di△S0△S1)

MBi←(Ai△(~S0)△(~S3)▽(Bi△(~S2)△S3)▽(Ci△S2△(~S3)▽(Di△S2△S3)

여기서 MAi는 A-BUS에 연결되어 AC로 들어가고MBi는 B-BUS에 연결된다.

과정5) 제어워드의 값에 의존하는 모든 연산은 AC와 B-BUS에 실려온것을 처리한다. 따라서 과정4)로 부터의 데이타는 ALU로 들어와서 제어코드S4,S5,S6,CA1(CW[5 6 7 8])의 비트값에 의해 연산의 종류가 구분되어 처리된다. 앞장에서 설명했듯이 산술연산및 논리연산은 한 회로내에서 처리되고 있다. 식(3-4), 식(3-10), 식(3-11)로 부터 ALU의 출력은 아래처럼 얻어진다.

Xi←ACi▽(S4△(~S5)△(~S6)△BSi)▽(S4△S5△(~S6)△(~BSi))
Yi←(S6△BSi)▽(S5▽(~BSi)
Zi←(~S4)△CAi

여기서 CAi는 AC의 i번째 비트값이고 BSi는 B-BUS의 i번째 비트값이다. 이 프로그램으로부터 i번째 비트의 연산결과값은

SMI←Xi⊕Yi⊕Zi

이 되고 이때 발생하는 캐리CAi+1은

CAi←(Xi△Yi)▽(Xi⊕Yi)△Zi

가 된다. 이처럼 ALU에서 처리된 결과는 상태레지스터를 세트시키고 C-BUS에 실려 자리이동 레지스터로 들어가게 된다.

과정 6) 상태 레지스터는 아래의 프로그램처럼 동작하게된다.

COUT←CA17◇SIGN←SM1◇ZERO ←~(▽/SM) ◇OVF←CA16⊕CA17
STAT←4ρST←OVF, ZERO, SIGN, COUT

과정7) C-BUS에 실려와 ALU에서 처리된 결과는 자리이동 레지스터의 제어코드 S7,S8에 의하여 4가지의 기능을 한다. 이 처리는〈Fig. 3-8〉로부터 아래 처럼 처리된다.

MOUTi←(SMi△(~S7)△(~S8)▽((SMi-1)△(~S7)△S8)▽((SMi+1)△S7△(~S8)

여기서, 처리된 결과의 Mout는 D BUS에 실려 목적지 레지스터로 향하게된다.

과정 8) 자리이동 레지스터에서 처리되어 D-BUS에 실린 데이타는〈Fig. 3-1〉로 구성된 4개의 16 비트 레지스터중 하나에 로더되는데 이때의 목적지 레지스터를 선정하는 것은〈Fig. 3-9〉의 2×4 디코더회로의 선택입력 S9,S10이다. 여기서 D0=1이면 레지스타A가 선택되며, D1=1이면 레지스터B가, D2=1이면 레지스터C가, D3=1이면 레지스터D가 선택된다. 아래 프로그램은 디코더된 S9,S10의 출력이 목적지 레지스터에 로더되어 연결되는 것을 보여준다.

DAFi←(Ai△(~DC1)▽(Fi△DC1)
DBFi←(Bi△(~DC2)▽(Fi△DC2)
DCFi←(Ci△(~DC3)▽(Fi△DC3)
DDFi←(Di△(~DC4)▽(Fi△DC4)

마지막으로 HLT명령이 아닐 경우는 프로그램 카운터의 내용을 1만큼 증가시켜 다음 명령을 ROM으로 부터 읽어와야 하므로 시스템의 첫동작으로 돌아가게 된다. 여기서 프로그램 카운터

의 내용을 증가시키는 유니트함수는 INCP로 주어진다. 위의 8가지의 과정에 의해 처리된 전체적인 16-비트 소형명령의 프로세서를 기술하여 시뮬레이션하였다.

V. 실험체제를 통한 프로세서의 동작의 예

먼저 입력하고자 하는 제어워드는 CW라는 변수를 통하여 프로세서에 2진코드로 입력되어 각 장치에 필요로 되는 제어코드로 분리되어 처리된다.

아래에서는 필요로하는 마이크로 동작(Micro-Operation)을 위한 제어워드를 넣어 프로세서가 동작하는 예들을 보여준다.

1) A←A+B

이것은 (레지스터 A의 내용과 레지스터 B의 내용을 더하여 레지스터 A에 전송하라.)하는 마이크로 동작이다. 이 동작을 위해서는 각 부분의 동작에 대해 아래와 같은 제어워드를 넣어야 한다.

1) MUX Group A에서 레지스터 A를 선택; S0,S1=0 0
2) MUX Group B에서 레지스터 B를 선택; S2,S3=0 1
3) ALU에서 (+)연산을 위해; S4,S5,S6, Cin= 0 0 1 0
4) Shifter에서 자리이동없이 전송을 위해; S7 ,S8=0 0
5) Decoder에서 A에 Load신호를 주기 위해; S9,S10=0 0

따라서 이 예에 대한 제어워드는 CW에

MUX A	MUX B	ALU
SO,S1	S2,S3	S4,S5,S6,Cin
0 0	0 1	0 0 1 0
SHIFTER		DECODER
S7,S8		S9,S10
0 0		0 0

을 (ADD,RA,RB,RC)로 기호화한 언어를 구성한 프로세서에 입력하여 넣어줌으로써 요구하는 동작을 하게된다.

2) C←A-D

이것은 (레지스터 A에서 레지스터D를 빼서 레지스터 C에 전송하라.)하는 마이크로 동작이다. 이동작을 위한 제어워드를 만들어 보자.

1) MUX Group A에서 레지스터 A를 선택; S0,S1=0 0
2) MUX Group B에서 레지스터 D를 선택; S2,S3=1 1
3) ALU에서 2의 부수에 의한 빼기연산을 위해; S4,S5,S6, Cin=0 1 0 1
4) SHIFTER에서 자리이동없이 전송을 위해; S7,S8=0 0
5) DECODER에서 C에 Load신호를 주기위해; S9,S10=1 0

이 동작을 위한 프로세서의 제어워드는

MUX A	MUX B	ALU
SO,S1	S2,S3	S4,S5,S6,Cin
0 0	1 1	0 1 0 1
SHIFTER		DECODER
S7,S8		S9,S10
0 0		0 0

로 표현되므로 (SUB,RA,RD,RC)를 입력하면 요구하는 마이크로 동작을 행하게 한다. 위의 동작들을 기본으로 하여 프로그램을 작성한 예는 아래에 보여진다. 이 예들에서 COMPROC은 본 연구에서 실현한 16-비트 소형명령의 프로세서이다.

3) 프로그램의 작성예

```
1. PROGIN1
2. STT←1 ◇DP←1
3. PC←PCDTB DP
4. '···· PROGRAM IN ····'
5. LDA 40
6. LDB 30
```

```
7. LDC 32
8. LDD 23
9. ROM [DP;]←AND,RA,RB,RC
10. ROM[DP+1;]←LOR,RC,RD,RA
11. ROM[DP+2;]←HLT
12. '---PROCESSOR CALL---'
13. COMPROC

1. PROGIN2
2. STT←1 ◇DP←1
3. PC←PCDTB DP
4. '--- PROGRAM IN ---'
5. LDA 100
6. LDB 200
7. LDC 250
8. LDD 300
9. ROM [DP;]←AND,RA,RB,RC
10. ROM[DP+1;]←LOR,RC,RD,RA
11. ROM[DP+2;]←HLT
12. '---PROCESSOR CALL---'
13. COMPROC
```

Ⅵ. 결 론

본 논문에서는 프로세서를 시뮬레이션할때 APL을 사용하였고, 이 APL은 다른 하드웨어 기술 언어들이 갖지 못하는 구조의 표현과 기술이 가능하도록 하여서 조합회로와 순차회로의 함수적인 수준(Functional Level)에서 실제회로의 구조와 대응성있게 시뮬레이션할 수 있었다. 특히 APL은 벡타(Vector)나 매트릭스(Matrix), 그리고 스칼라(Scalar)량을 기억하고 처리하기가 간편하였으므로 프로세서의 레지스터나 그것의 소자인 플립-플롭의 구조나 동작을 표현하기가 간결하였다. 역시 선언된 매트릭스의 원소를 선택적으로 처리할 수가 있어서 프로세서의 제어워드를 각 부분 회로에 분리하여 인가시켜 동작시킬 수가 있었다.

본 논문에서 제시한 소형 명령을 갖는 16비트의 프로세서는 IDL로서의 APL을 이용하여 그 구조와 동작을 시뮬레이션하였으며 각 부분 회로는 멀티플렉서 그룹 2개, ALU, 자리이동 레지스터, 디코더, 그리고 범용레지스터 그룹으로 분류하였으며 ALU의 설계시 산술연산과 논리연산을 겸하도록 하였기 때문에 그 프로세서의 동작에 대한 표현과 구조에 대한 기술이 간단해졌다.

특히 하드웨어를 구성할때 사용되는 논리게이트나, 플립플롭들이 이미 IDL속에 포함되어 있거나, 간단히 정의되어 사용될 수 있게 하였기에 논리생성레벨을 정확히 표현할 수 있다. 그리고 각각의 부분회로는 여기서 제시한 실험체제를 통하여 시스템에 인가된 12-비트 제어워드의 각 선택 코드에 의해 동작하도록 하였으며, 한번의 제어워드의 순차에 의해 프로세서가 동작을 할때 각회로의 중간 동작과정을 설계자가 시뮬레이션하는 방법에 따라 쉽게 변화시킬 수 있도록 하였고, 쉽게 식별할 수 있도록 하였으며, 스트링 출력을 얻기가 쉬웠으므로 시스템의 구조에 의한 동작의 이해가 양호하였다.

그리고 구성된 컴퓨터 시스템에 입력하는 명령과 자료가 숫자뿐만 아니라 기호화하여 제시한 언어를 사용하도록 하였기 때문에, 그리고 설계자나 사용자가 요구하는 동작을 위해 기호화하여 세팅된 명령을 실현한 실험체제를 거쳐서 일반적인 어셈블러(Assembler)언어로 프로그램하는 것과 유사하게 입력할 수 있게 하였기 때문에 역시 구성한 프로세서의 동작과 구조를 쉽게 확인할 수 있었다. 여기서는 설계자가 유닛트 함수를 필요로 할때 임의로 즉시에 만들어 사용하였기에 시스템을 기술할때 유닛트 함수의 기능과 수에 제한을 받지않고 다양하고 효과적으로 사용할 수 있었다.

REFERENCE

1. F.J.Hill "Introducing AHPL", Computer, Vol. 7 No. 12, pp 28-30, 1974.
2. Y.CHU "Introducing CDL", Computer, Vol.7 No.12, pp31-33, 1974.
3. D.L.Dietmeyer "Introducing DDL", Computer, Vol.7

No.12, pp 34-38, 1974.

4. D.Siewiork "Introducing ISP", Computer Vol.7 No. 12, pp39-41, 1974.
5. D.Siewiork "Introducing PMS", Computer Vol.7 No. 12, pp42-44, 1974.
6. J.Aylor, R.Waxman and C.Scarratt "VHDL: Feature Description and Analysis", IEEE Design & Test of Computer, 1986, April
7. G.R.Peterson, Cliff D'Souza and F.G Hill "AHPL: A Language for Function Level Design", First Annual Peonix Conference on Computers and Communications, pp 48-53, 1982, May.
8. Z.Navabi "Generation Gate Level Two Phase Dynamic MOS Logic from Ahpl", Microprocessor and Microprogram(Netherlands), pp 89-94, 1985, Sept.
9. C.K.Alexander and R,Z.Makki "Digital System Design Using Structured Ahpl", IEEE Southeastcon '83 Conference Proceedings, pp 63-67, 1983, April.
10. G.N.Reddy, D.E.Simpson and H.A.Jones "8095 Simulatior Using Ahpl", Proceedings of the Ismm symposium Mini and Microcomputers and Their Applications Mini 89, pp.119-123. 1987 July
11. J.A.Convingtion and S.G.Shiva Modular Hardware Synthesis using an Ahpl" IEEE Southeastcon 198 1 Conference Proceedings, No.4, 1981, April.
12. A.M.Shah and S.G.Shiva "H/W Systhes is from DDL" IEEE Southeastcon 1981 Conference Proceedings, pp 517-521, 1981, April.
13. N.Kawato, T.Saito and T.Uehara "Design and Verification of Karge Scale Computers by using DDL", Fujitsu SCI and TECH. J(Japan), Vol.15 No. 2, pp1-19, 1979, June.
14. S.G.Shiva "On The selection and Development of an HDL", Proceedings of The Seventeenth Annual Hawaii National Conference on the system sciences 1984, Vol.1 No.6, pp 120-128.
15. .Roesner "A H/W Desing Language for Logic Simulation and Systhesis in VLSI", Proceedings of Vlsi and Computers First International conference on Computer Technology, Systems and and Applications Compeuro 87, pp 311-314 1987, May.
16. M.Morgan "Model-an HDL for IC Design", Proceedings of the Third Silicon Design Conference, pp 83-87, 1986, July.
17. F.Meshkinpour and M.D.Ercegovac "A Functional Language for Description and Design of Digital Systems: Sequential Constructs", 22ND ACM / IEEE Design Automation Conference Proceedings 1985 CAT, pp 238-244, 1985, JUNE.
18. K.W.LI, J.R.Amstrong and J.G.Tront "An Ahl Simulation of the Effects of Single Event Upsets on Microprocessor Porgram flow", IEEE Transaction on Nuclear Science, Vol.Ns-31 No.6, pp 1139-1144 , 1984, DEC.
19. Y.Hollander "Using an RTL Simulatior to Simplify Vlsi Desing", VLSI DES. (USA), Vol.4 No.5, pp 60-66, 1983.
20. A.Brish, R.Keinan and Y.Ravid "A Smart System That compiles RTL Models From Shcemetics", VLSI Syst. DES.(USA), Vol.9 No.2, pp 35-36, FEB, 19 88.
21. B.Milne "Synthsize Your Asics from RTL Behavioral Blocks", Electron DES.(USA), Vol.36 No.13, June 1988.
22. Y.Nakamura and K.Oguri "An RTL Logic Design aid Parrallel Control Vlsi Prcessor", VLSI '87 VLSI Design of Digital System Proceedings of The ifitp TC10 / WG105 International conference of Very Large Scale 1988, pp 35-38.
23. S.J.Piatz "HSL: A Comprehensive Hardware Design Language", Proceedings of the Seventeenth Hawaii International conference on System Sciences 1984, Vol.2, pp 137-139, Jan. 1984.
24. A.K.Burston "Logic Systehsis for a Computer Hardware Design Language", Aust. Comput. J(Australia), Vol.16 No.4,pp 130-139, Nov. 1984.
25. J.Miles "Ella-NOT Just a H/W Design and Description Language", Proceedings og the Third Silicon Design Conference, pp 65-76, 1986.
26. A.Sugimoto and M.Fukushima "Vega: A Visual Modeling Language for Digital Circuit Design", Proceedings of IEEE International conference on Computer Design 'Vlsi in Computers ICCD '84, pp 807-812, Oct. 1984.
27. T.Sasaki, N.Koike, K.Ohmori and K.Tomota "Hal: A Block Level H/W Logic Simulatior", 20th Design Automation Conference IEEE, pp 150-156, 1983.
28. Y.James and O.Fong "Microprocessor Modeling for Logic Simulation", IEEE Test Conference, 1981.
29. L.I.Maissel and DL.L.Ostapko "Interactive Design

Language: A Unified Approach to H/W Simulation, Synthesis and Documentaion", ACM IEEE 19th Design Automation Conference Proceedings 1982, pp 193-201, Jun. 1982.
30. D.Thomas "RTL Simulation Makes A Comback complex Vlsi", Comput. DES.(USA) Vol.25 No.3, pp 63-67, Feb. 1986.
31. "Functional Simulation Using Logic Programming", IBM Tech. Disclosure Bull(USA), Vol.28 No.2, pp 665-667, Jul. 1985.
32. K.Hosono "Developement of a Personalized Information Retrieval System by Using Apl Programming Language", Communication Information, Proceedings of the 43ND Asia Annual Meeting, Vol.17, pp 77-79, Oct. 1980.
33. M.R.Barbacci "A Comparisor. of Register Transfer Languages for Describing Computers and Digital systems", IEEE Trans. On Comput., Vol.C-24 No. 2, pp 137-150, Feb. 1975.
34. S.G.Shiva "Computer Hardware Description Language A Tutorial", Proceedings of the IEEE, Vol.67 No. 12, pp 1605-1615, Dec. 1979.
35. G.J.Lipovski "H/W Description Languages: Voice From the of Babel", Computer Vol.10. No.6, Jun. 1977.
36. R.Davis and H.Shrobe "Representing Structure and Behavour of Digital H/W", IEEE Trans. Comput., pp 75-82, Oct. 1983.
37. R.E.Swanson "Exetensions of AHPL and Optimization of the AHPL Compile for MSI/LSI Design", PH.D.Dissertation, Univ. of Arizona, Mar. 1978.
38. M.Masud "Modular Implementation of a Digital H/W Design Automation System", PH.D.Dissertation, Univ. of Arizona, May 1981.
39. H.Jellerman and A.Smith "APL/360 Programming and Application", McGraw Hill Newyork, 1976.
40. L.Gilman and A.J.rose "APL: An Interactive Apptoach", John Wiley and Son, NewYork, 1976.
41. K.E.Ilverson "A Programming Language", John Wiley, Newyork, 1962.

부록

APL 연산자

←: Transfer	→: Branch
□: Input/Output	π: Comment
△: Logical and	ρ: Array Decaration
▽: Logical or	γ: Decode
~: Logical Not	◇: Connect
⊕: Logical Exclusive or	/: Summation
' : Catenate	⊙: Equivalence

OPCODE

TRA←6ρTRA←000000
SHR←6ρSHR←000001
SHL←6ρSHL←000010
INC←6ρINC←000100
ADD←6ρADD←001000
ADC←6ρADC←001100
SBC←6ρSBC←010000
SUB←6ρSUB←010100
DEC←6ρDEC←011000
LOR←6ρTRA←100000
EOR←6ρEOR←101000
AND←6ρAND←110000
NOT←6ρNOT←111000

朴 斗 烈(Doo Youl PARK) 正會員
1956年 1 月30日生
1980年 2 月：東亞大學校 電子工學科 卒業
1982年 2 月：東亞大學校 電子工學科 碩士學位 取得(工學碩士)
1985年 2 月：東亞大學校 電子工學科 博士課程 修了
1980年～1983年 9 月：東亞大學校 電子工學科 助教
1985年 3 月～現在：東洲女子專門大學 電子計算科 助教授

李 鍾 憲(Jong Heon LEE) 正會員
1933年12月25日生
東亞大學校 物理學科 卒業
東亞大學校 電子工學科 教授(現在)
87年～'89年：電子工學會 釜山－慶南 支部長