연구용 CAD툴에 의한 소형 MPU의 설계 및 파이프라인화의 고찰

이 수 정† 박 도 순†† 송 낙 운†††

요 약

본 논문에서는 연구용 VHDL 및 CAD 툴을 사용하여 톱다운 설계방식에 의하여 소형 마이크로프로세서(MPU;microprocessor unit)의 설계를 수행한다. 이를 위하여 기본 MPU와 이의 파이프라인화 구조를 제안한다. 설계목표와 명령어, 아키텍처가 결정되면, 이를 우선 C언어로 모의실험하여 동작을 확인하며, 다음 VHDL 모의실험의 경우, 주어진 입력에 대하여 내부 레지스터의 내용을 점검하여 동작을 확인한다. 다음에, 이를 연구용 CAD 툴에 의해 완전주문형(full-custom)/반주문형(semi-custom) 설계방식에 의해 레이아웃을 수행하며 관련 모의실험을 수행한다. 이어 성능개선을 위하여 제안한 파이프라인 구조를 모의실험을 통하여 타당성을 확인하며 아울러 관련 문제점 및 향후 연구방향에 관해 논한다. 결론적으로, 본 논문를 통하여 MPU의 설계방법을 정립하였으며, 아울러 성능개선을 위한 아키텍쳐의 설계변화가 가능하였다.

Investigation of Small MPU Design and its Pipelining by Research CAD Tools

Soo Jung Lee†, Do Soon Park†† and Nag Un Song†††

ABSTRACT

In this paper, design of small microprocessor unit is implemented using research purpose VHDL and CAD tools by top-down design method. For this, original basic MPU and its pipelining architectures are suggested. Once, design target, instruction sets, architecture are decided, the operation is confirmed by C language simulation, and then the operation is confirmed by checking internal register contents for given inputs in the case of VHDL simulation. Then, design layouts are made by full/semi-custom design methods by research CAD tools and related simulation is implemented. The feasibility of suggested pipelined structure for performance improvement is confirmed by simulation, and related problems and future research directions are discussed. In conclusion, the MPU design methodology is set up and the design change of architecture is possible by this paper.

1. 서 론

VLSI 회로가 ASIC화 되면서 개발시간의 단축을 위한 톱다운 설계 및 합성(top-down design & synthesis)을 위한 상하위단계 시스템 설계를 통합체계화하는 CAD 환경, 즉, 실리콘 컴파일러에 관한 연구가 활발히 이루어지고 있다. 한편, 이 CAD 환경은 HLS(high level synthesis:상위단계합성)와 LLS(low level synthesis:하위단계합성)로 나누는 것이 일반적이다. 이중, HLS의 경우, 상위단계 모의실험과 아키텍처합성 등으로 이루어지며, 이의 톱다운 설계합성을 위하여, AHPL을 포함한 각종 HDL(hardware description language) 관련 언어가 등장하게 되었다.

이 가운데 가장 대표적인 언어인 VHDL은 1981년경 DoD에 의하여 VHSIC의 일환으로 시작된 이래 1987년 IEEE에 의해 하드웨어 기술을 위한 표준언어로 채택되는 등, 현재 많은 기업들과 학계 등에서 이를 통한 연구를 활발하게 진행

† 정 회 원 : 홍익대학교 전자공학과
†† 정 회 원 : 홍익대학교 컴퓨터공학과 교수
††† 정 회 원 : 홍익대학교 전자공학과 교수
논문접수 : 1994년 8월22일, 심사완료 : 1994년 10월31일

하고 있으며 이에 관한 연구는 크게 모의실험과 합성의 두 방향으로 진행되고 있다. 이의 설계관련 연구로서는 ASIC 설계를 위한 모듈 라이브러리 설계[1, 2, 3], ASIC 시스템에의 응용에 관한 연구가 있고[4, 5], CAD 관련 연구로서는 VHDL로부터 흐름도표 등의 변환을 통한 합성 및 설계환경 구축에 관한 연구[6, 7]가 널리 이루어지고 있으며, 국내에서도 이에 관한 연구가 활발하게 이루어지고 있다[8, 9]. 한편 이러한 HDL 언어 이외에도, HLS의 아키텍처합성 방법론(즉, 데이타경로와 제어경로합성 등)에 관한 많은 연구가 이루어지고 있다[10]. 다음으로, LLS의 경우 주로 로직합성 아랫단계(레이아웃 포함)를 의미하게 되는데 비교적 다양하고 튼튼한 연구용 CAD 툴이 널리 보급되어 쓰여지고 있다.

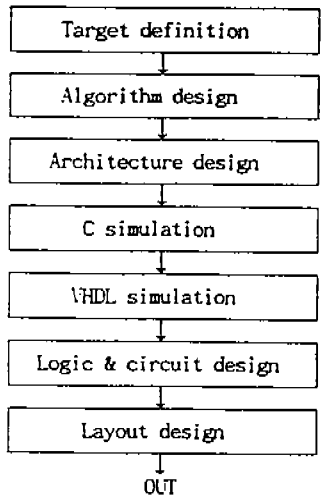
자동화 설계환경구축을 통한 디지탈시스템의 ASIC 설계에 따른 종합적인 이해를 위한 대표적인 예로는 마이크로프로세서를 들 수가 있으며, HDL에 의한 관련 설계구현에 관해 많은 교육적인 연구가 있어왔다[11, 12]. 이들의 자동화설계에 관한 연구도 활발하여, CMU 대학의 Walkeret al.[13]은 그들의 SAW 프로젝트에서 HDL에 속하는 ISP 언어를 사용하며 VT 개념을 도입하여 대형 디지탈시스템의 톱다운설계를 구현하였으며, 동 대학의 Birmingham et al.[14]는 기존의 부품을 기초로 한 소형 컴퓨터의 지식베이스 합성툴인 MICON 프로젝트를 수행하였다. 아울러 RISC 컴퓨터의 보편화에 따라 다양한 내부구조(pipelining, cache, FPU 등)를 채택하게 되었으며[15, 16, 17, 18], 이중 파이프라이닝의 최적화에 관한 많은 연구가 있었다.

그러나 이러한 많은 연구가 각 팀 고유 CAD 툴에 의한 연구로 혹은 고가의 CAD 툴에 의한 일부의 관련 분야의 연구로 국한된 감이 없지 않으며, 특히 연구용 설계툴을 이용한 디지탈시스템의 톱다운 설계를 위한 국내의 연구는 미약한 편이다. 일반적으로 연구용 CAD툴은 소프트웨어간의 상호 연계성 미비로 일관된 통합체계면에서 매우 취약하므로, 어느 정도 확실한 연계관계가 있는 소프트웨어 이외에는 설계자의 끊임없는 간여 및 상호연계 노력이 필요하게 된다. 이를 위하여 본 연구에서는 상용툴보다 성능면에서는 다소 미흡하나, 가격면에서 저렴한 일반적인 연구용 CAD 설계환경의 구축을 통하여 기본 MPU를 설계하며 톱다운 자동화 설계의 가능성과 문제점을 점검한다. MPU의 설계목표는 그 복잡도의 고려정도 및 개발방법에 따라 다양한 접근이 가능하나, 먼저 간단한 기본구조를 가정한 후 이를 상위단계의 언어에 의한 모의실험을 수행하며, 이의 성능개선을 위한 파이프라인 구조를 제안하여 모의실험과 관련 문제점을 검토한다. 특히, 이 과정을 통하여 C 언어에 의한 모의실험으로 상위단계 MPU 동작과, 이에 이은 VHDL 언어 모의실험에 의한 MPU 하드웨어 동작을 검증, 확인하였으며, 아울러 레이아웃을 포함한 하위단계 설계를 통하여, 설계단계상 상호연계의 문제점 등을 검토한다.

2. 기본 MPU의 구성 및 파이프라인화

2.1 설계환경의 구성

본 논문에서 채택한 마이크로프로세서의 설계과정의 흐름도를 다음 (그림 1)에 보였다.



(그림 1) 설계 흐름도
(Fig. 1) Design flowchart

이러한 톱다운 설계방식에 의하여 본 연구에서는 기본 MPU 및 성능 개선을 위한 파이프라인 구조의 MPU의 명령어 흐름과 내부 아키텍쳐를 결정하여 이의 모의실험과 레이아웃을 설계하였다.

본 설계의 CAD 환경은 주로 연구교육용 툴로 구성하였다. C언어는 표준 ANSI-C를 사용하였으며, VHDL 언어로서는, 1989년초 Pittsburgh 대학에서 연구교육용으로 발표된 Pittsburgh-VHDL 버전 2.216[19]을 사용하였다. 대체로 이 툴은 연구 및 교육용 버전으로서 VHDL 표준에서 제공하는 신택스(syntax)를 완벽하게 지원하지는 않지만, 간단한 하드웨어는 충분히 설계가 가능하다. 로직합성(LS:logic synthesis)과 레이아웃수행은 주로 Berkeley 툴(OCTTOOL, MAGIC 등)을 사용하였으며 반주문형 설계에는 MSU(Mississippi State University) 표준셀을 이용하였다.

2.2 기본 MPU의 구성

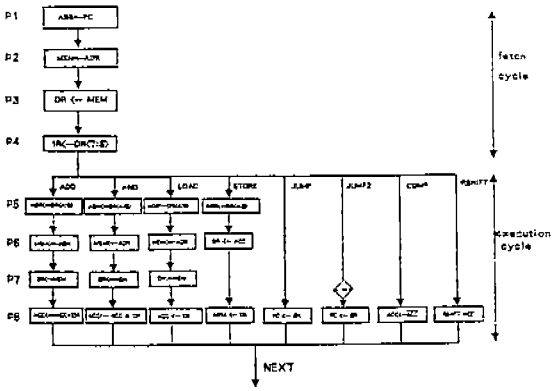
본 논문에서는 Hayes의 참고문헌[20]을 기초로 한 기본 구조를 택하였다. 이는 8비트 RISC CPU 구조로, 8개의 명령어를 채택하였으며, 각 명령어의 연산부호(opcode)를 나타내면 다음 〈표 1〉과 같다.

〈표 1〉 명령어의 연산부호 포맷

〈Table 1〉 OPCODE format of instructions

명령어	RSHIFT	LOAD	STORE	ADD
연산부호	000	001	010	011
명령어	AND	JUMP	JUMPZ	COMP
연산부호	100	101	110	111

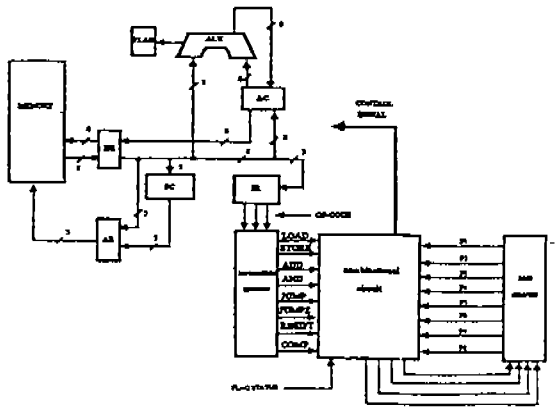
명령어 포맷은 3비트 연산부호, 5비트 피연산자(operand)의 단일 바이트의 고정포맷을 사용하였고, 데이타 포맷은 이진수로 작성하였으며, 문제의 단순화를 위하여 어드레싱(addressing)은 디렉트 모우드(direct mode)만을 가정하였다. 각 명령어 수행의 기본 흐름도를 (그림 2)에 보였다.



(그림 2) 명령어의 수행 흐름도

(Fig. 2) Flowchart of instructions

타이밍의 기본 머신주기(machine cycle)는 페치(fetch)와 엑시큐션(execution)으로 나누어져 있고 각각 4사이클로 구성되어 있으며, 이들은 직렬연속적으로 수행된다. 설계하고자 하는 MPU 구조는 ALU(arithmetic logic unit), DR (data register), AR(address register), IR(instruction register), ACC(accumulator), PC(program counter), RC(ring counter), FR(flag register), CON(control part)등으로 되어 있으며, 이중 사용자가 억세스 가능한 레지스터는 ACC뿐이다. (그림 3)에 이의 내부구조를 보였다.



(그림 3) 기본 MPU 구조

(Fig. 3) Basic MPU architecture

ALU의 기능은 덧셈과 로직연산, 보수로직(complement logic) 연산, 우측이동(right shift) 연산 등으로 되어 있으며, FR에는 제로(zero) 플래그만을 두었다.

2.3 성능 개선의 고찰

앞에서의 MPU의 성능을 개선하기 위하여 (그림 1)의 흐름도에 설계의 제한 및 개선점을 포함하는 과정을 위한 피드백 경로를 설정할 수가 있다. 일단 설계의 측면에서는 크게 (그림 1)의 흐름도에서 가장 상위단계인 내부구조의 개선과 하위단계에 해당하는 모듈셀의 생성으로 고려할 수가 있다. 또한 관련 CAD 소프트웨어의 측면에서는 일반적으로 각 설계과정에 걸쳐 데이타 생성 및 처리에 관여하게 되나, 이를크게 합성(synthesis)과 모듈셀 생성(module-cell generation)관련 소프트웨어로 나눌 수가 있으며 전자의 경우는 주로 상위단계에 관계된 컴파일레이션, HLS, 매핑관련 소프트웨어 등으로 나눌 수가 있고, 후자는 주로 하위단계합성(즉, 로직합성, 레이아웃생성 등) 소프트웨어로 볼 수 있다. 따라서 CAD의 분야에서는 성능의 개선을 위하여 이러한 CAD 소프트웨어의 알고리즘의 개선과 이들간의 효율적인 연계운용 등이 중요하다. 본 논문에서는 이중 설계의 측면에서 상위단계인 내부구조 개선의 한 방법인 파이프라인화에 주력하였다.

일반적으로 파이프라인의 구현에는 명령어 단계, ALU 등 데이타 경로 관련 하드웨어 단계의 변형을 검토하여야 하나, 본 논문에서는 명령어 단계의 파이프라인화만을 고려하였다. 명령어의 파이프라인화에서 얻어지는 성능향상은 파이프라인 단(stage)의 요소를 가지고 증가한다. 일반적으로 컴퓨터성능의 향상에 대한 기본적인 접근은 수행시간의 감소를 유도하는 것이다. 이 수행시간은 식 (1)과 같이 정의된다.

$$\text{수행시간} = \text{명령어의 갯수(Ic)} * \text{T(클록주기 시간)} * \text{CPI(clock per instruction)} \quad (1)$$

식 (1)에서 우변의 어느 변수를 강조하는 가에 따라 그 아키텍쳐의 특성이 결정되는데 이러한 맥락에서 파이프라인 기술의 도입은 CPI가 감소하는 효과를 가져온다고 할 수 있다. 그러므로 이를 개선하기 위한 명령어의 병렬성(parallelism) 증가에 관한 많은 연구가 수행되어 왔으며 이에 의한 여러가지 알고리즘들이 사용되어 왔다. 명령어의 병렬성을 방해하는 것은 이들 간의 의존성(dependency)이므로, 이들 장애(hazard)를 효과적으로 제거하거나 극복하는 것이 중요하다. 한 예로 이들 명령어의 의존성은 명령어의 거리가 멀수록 감소하는 경향을 가지므로, 명령어의 수행순서(프로그램의 순차적인(sequential)특성)를 적절하게 교환함으로써 병렬성의 향상을 가져온다. 이 방법을 명령어 스케쥴링(instruction scheduling)이라 하고, 이는 정적(static) 스케쥴링과 동적(dynamic) 스케쥴링으로 나누며, 주로 컴파일러 단계와 하드웨어 단계에서 이의 최적화를 통하여 해결한다. 이상에서와 같이 명령어 단계의 설계에서 파이프라인 성능을 개선하기 위해 데이타 관련 스케쥴링 방법 이외에도, 제어관련의 분기예측(branch prediction) 등 많은 방법들이 있다[15, 16, 17, 18].

2.4 파이프라인 구조의 결정

기본적인 MPU의 아키텍쳐를 바탕으로 이를 효과적으로 수행하는 파이프라인 구조(즉, 주기와 단의 수)를 결정해야 하는데 이는 명령어의 흐름도에서처럼 각 단사이에 걸리는 시간과 이를 수행하는 하드웨어의 숫자 등에 의해 정의되는 비용함수에 의해 결정이 된다. 일반적으로 이를 위하여 앞서의 명령어 흐름도에서 각 제어동작의 수행시간을 바탕으로 이들이 중복되지 않게 진행되면서 내부구조의 가동이 최대한 골고루 균등하게 수행되도록 해야한다. 또한 수행시간의 분배중에 시간상 병목현상이 이루어지는 경우에 임계경로상에 위치하는 하드웨어를 보강하는 형태로, 속도비용과 하드웨어 비용의 절충(tradeoff)으로 구조를 결정한다. 한편, 주기의 시간슬롯을 늘리

는 경우, 해당슬롯에서의 세부동작(micro-operation)을 위상의 차이를 두어 수행하게 함으로써 동일 하드웨어를 쓰는 경우의 조정등을 이루어 가동도를 높일 수가 있게 된다. 아울러, 각 단의 해당 시간슬롯 초기에 이러한 조정을 위한 점검시간을 할당하였다. 그러나 이러한 방법으로도 목표가 이루어지지 않는 경우에는 그 상위단계의 설계로 올라가 시도하는 등의 조정을 행해야 한다.

본 논문에서는 그림 2의 흐름에 의한 파이프라이닝화 문제를 시도하였다. 물론 이는 단순한 구조를 택한 관계로, 참고문헌[15, 16, 17]의 보편적인 RISC 접근방식과는 차이가 있기는 하나, (그림 3)의 기본적인 구조를 최대한 이용하여 변형하는데 역점을 두었다. 우선, (그림 2)의 흐름도상의 결리는 시간을 고려하여 단위시간 슬롯을 결정하는데, 정확한 시간의 추출을 위하여는 레이아웃을 통한 해당 모듈의 모든 시간소요 정도를 고려하여야 하나 일단은 이를 단순화하여 중요한 시간소요만을 고려하였으며, 단일 포트 메모리(single-port memory)를 채택함에 따라서 메모리의 데이타 입출시간이 가장 고려의 대상이 되었다. 이제 데이타 흐름(IF:instruction decode, ID : instruction decode, OF:operand fetch, EXE: execute)에 따라서 이를 단수 2-4에 걸쳐 시간슬롯을 조정 할당하며 관련 하드웨어를 정리하여 이들로부터 해석하고자 하는 구조를 채택해야 하는데, 2단의 경우, 속도면에서 뒤지며 메모리 페치시에 충돌에 대한 고려, 위상차 조정 또는 하드웨어의 추가 등이 이루어져야 한다. 4단의 경우에는 속도는 개선이 되나 단수의 증가에 따른 하드웨어가 증가하며 메모리 페치시에도 고려가 이루어져야 한다. ID단에서의 ID와 OF에 걸리는 시간이 서로 연속적인 관계에 있으며 ID에서 일단 조건 점검(condition check;CC)을 행한 후 이의 결과에 따라 해당 제어신호가 발생되게 됨으로 OF가 행해지는 명령어와는 동작상에 얼마간의 중복수행이 일어나게 되며, 해당 데이타 페치 등이 일어난 후에 이를 저장하는 장소의 중복문제가 생기므로, 이를 완화(relax)하는 구조를 채택하여 관련 모의실험을 수행하였다. 한편 파이프라인 구조의 결정에 정확히는 이들에 대한 면적 등 하드웨어의 비용도 포함시켜야 하나, 여기에서는 간략화한 속도개선 측면에서만 이를 고려하였다. 일단 이와같이 3단 파이프라인 구조를 결정한 후, 각 단에서의 하드웨어 균등화가 이루어진 후에는 이들간의 하드웨어 공유 및 각종 장애 문제, 즉, 구조 장애(structural hazard), 제어 장애(control hazard), 데이타 장애(data hazard)

〈표2〉 파이프라인 MPU의 세부동작 도표

(Table 2)Micro-operations in pipelined MPU

STAGE	ADD, AND	LOAD	COMP RSHIFT	JUMP	JUMPZ	STORE
IF	MAR<—PC MBR<—M(MAR) PC<—PC+1	MAR<—PC MBR<—M(MAR) PC<—PC+1	MAR<—PC MBR<—M(MAR) PC<—PC+1	MAR<—PC MBR<—M(MAR) PC<—PC+1	MAR<—PC MBR<—M(MAR) PC<—PC+1	MAR<—PC MBR<—M(MAR) PC<—PC+1
ID	DECODING MDR<—M[OPERAND]	DECODING MDR<—M[OPERAND]	DECODING	DECODING	DECODING	DECODING MDR<—AC
EXE	**AC<—AC+MDR** **AC<—AC and MDR**	**AC<—MDR**	**AC<—rshift AC** **AC<—not AC**	PC<—MBR2_OUT	**AC = 0?** PC<—MBR2_OUT	M(MAR)<—MDR

M 메모리를 억세스하는 단(음영이 있는 부분)

M ACC를 억세스하는 단(진한 글자)

능을 해결하기 위한 타이밍을 고려해야 한다. 특히, 이는 명령어의 각 수행단에서 전 명령어의 선행수행단의 조건이 점검되어 이에 따른 제어신호를 발생하여야 한다. 따라서 이를 위하여 동일 IF, ID+OF, EXE의 수행에서 전단의 조건점검의 인식을 위한 지연시간을 포함하였다.

이러한 조건들을 고려하여 기본 MPU의 구조를 파이프라인화된 MPU로 내부 아키텍쳐를 변형하였으며, 이경우 명령어 흐름의 각 파이프라인 단에서 발생하는 RTL 단계의 세부동작은 (표 2)와 같다.

즉, 파이프라인 수행을 3개의 단(stage), 즉, IF, ID, EXE로 구성하며, 이를 위하여 IQ(instruction queue)형태의 명령어 버퍼 레지스터와 데이타 레지스터를 포함하는 3단 파이프라인 구조를 도입하여, DR을 MBR(memory buffer register)과 MDR(memory data register)로 나누어 담당하게 하여 MBR에는 메모리에서 페치한 명령어를 저장하고, MDR에는 메모리에서 페치한 피연산자를 저장한다. 특히, MBR은 IQ와 같이 3개의 명령어를 순차적으로 저장할 수 있도록 했다.

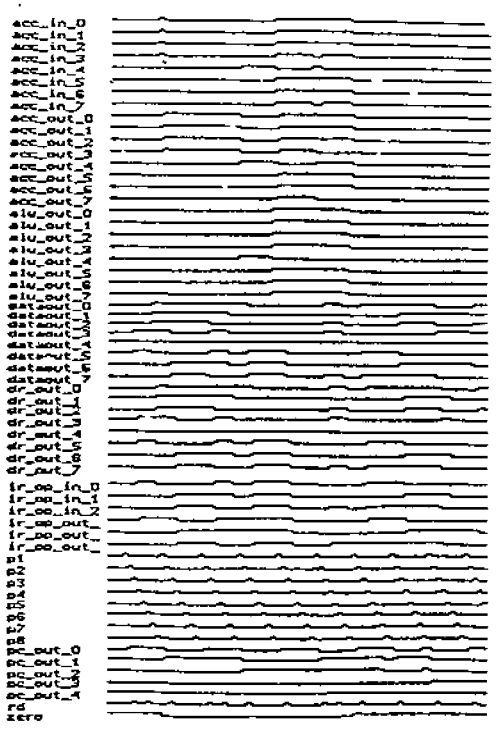
3. 설계 및 모의실험

3.1 기본 MPU의 설계

3.1.1 모의실험

본 절에서는 2.2절에서 제시한 기본 MPU구조를 C언어로 기술하여 모의실험을 수행하였다. MPU에서 명령어 수행방법은 각 명령어마다 페치와 엑시큐션 주기를 연속적으로 수행하여 그 명령어가 완전히 수행된 후, 다음 명령어로 제어가 넘어간다. 이때, 명령어의 수행 순서는 분기문(JUMP, JUMPZ)이 없는 한, 주 메모리에 들어온 순서대로 순차적으로 수행한다. 하드웨어로 MPU를 설계할 때 세부동작들의 동작시점을 결정하는 것과 데이타 경로상에서 발생하는 타이밍 문제가 대두되겠지만, C 언어는 하드웨어에 독립인 언어이기 때문에 클록과 지연문제는 고려하지 않고, 논리적인 흐름만을 고려하여 모의실험을 수행하였다. 또한, MPU내에 있는 레지스터는 프로그램내의 변수로 선언하여, 한 명령어의 주기가 끝날 때마다, 그 변수의 내용을 표시하였다. 이것은 실제 MPU에서 해당 명령어를 수행한 후에 변경된 레지스터의 내용과 같은 의미를 갖는다. C언어에 의한 모의실험 결과는 다음의 VHDL 언어에 의한 모의실험 결과와 시간의 지연정도를 제외하고는 대체로 일치하였다.

다음에는 앞의 기본 MPU 구조를 VHDL 언어에 의하여 모의실험을 수행한다. 클록주기의 발생은 링카운터가 하고, 각 주기에서 필요한 신호가 발생되도록 프로그램을 작성하였다. 여기서 발생되는 신호들은 클록의 지속기간에서만 사용하고 에지(edge)에서의 발생은 사용하지 않았으며, 해당 클록의 주기는 200ns, 듀티사이클(duty cycle)은 0.5로 하였으며, 메모리 억세스(memory access) 시간은 50ns로 놓았다. 제어부분의 설계



(그림 4) 기본 MPU의 VHDL 모의해석 결과
(Fig. 4) VHDL simulation results of basic MPU

는 하드와이어드(hard-wired) 방식을 택하였으며 (그림 2)의 흐름도에 의하여 구성하여 이를 행위 모델링(behavioral modeling)을 이용하여 RTL 시뮬레이션을 수행하였고 통합구조에 각 성분(ALU, 레지스터 등)을 프로세스(process) 문으로 기술하였다. 이와 같이 주어진 각 명령어의 번호를 가지고 수행 순서를 1 3 6 7 0 4 5 6 2로 하는 테스트 벡터를 구성하여 내부 데이타의 변화를 점검하였는데, 이의 모의실험 결과를 (그림 4)에 보였으며 각 레지스터의 내용은 예상한 바와 동일하였다.

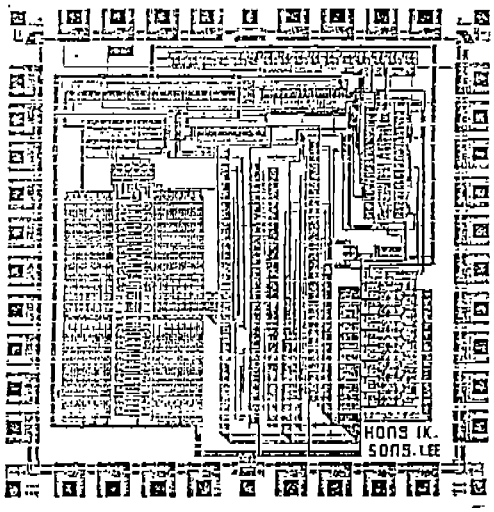
모의실험시에 고찰사항은 모듈 인터페이스 간의 지연을 고려하여 올바른 데이타 전달이 이루어지게 하는 것이었다. 본 논문의 경우, 블록(프로세스) 연산시에 지연이 어느 정도 생긴다고 가정을 하여 행위기술을 하였기 때문에 이와 같은 상황이 발생하는데, 물론 이들도 블록 모듈간에 즉각적인 데이타 전달이 생긴다고 가정하면 무시할 수가 있겠으나, 실제의 경우 항상 이들간에 시간지연이 존재함으로 이의 고려시에는 주의를 기울여야 했다. 이는 본 시뮬레이터가 사건피구동(event-driven) 형이기 때문에 생기는 것으로서 이들 문제는 대체적으로 일정한 값의 지연(delay)이나 속성(attribute)을 사용하여 해결하였는데 이를 요약하면 아래와 같다.

첫째로, 임의의 신호 할당(signal assignment)이 있을 때 이것이 프로세스문 안에 있을 때가 프로세스 문 밖에 있을 때보다도 1 클록주기 지연 출력되어, 이러한 지연된 신호의 출력 때문에 모의실험이 정확하게 이루어지지 않은 경우, 이를 해결하기 위해 해당 신호 할당을 프로세스문 밖에서 기술하여 타이밍 문제를 해결하였다. 그러나, 이 방법은 앞으로 설명할 두 방법으로의 해결이 곤란한 경우에만 사용하였다. 둘째로는 타이밍 문제 해결을 위해 가장 많이 사용한 방법으로, 프로그램을 작성할 때 명백한 시간 지연을 표현하는 것이다. 이상을 요약하면, 첫번째가 델타지연(delta delay), 두번째가 관성지연(inertial delay)에 해당된다. 셋째로는, 사용한 툴에서 제공하는 속성인 상승(rising), 하강(falling)을 이용하여, 이벤트 선택의 폭을 '0', '1'에서 '0', '1', rising, falling으로 확장하여 사용하였다. 이상과 같이 타이밍의 충돌 문제를 해결하였으나 부득이한 경우 타이밍 매치(timing match)를 위하여 지연성분을 삽입하여 이를 해결하였다.

3.1.2 하위단계 설계

다음에 기본 MPU를 UC, Berkeley의 CAD 툴인 OCTTOOL을 사용하여 반주문형 방식으로 설계하였으며, 메모리 등 일부 모듈블록은 MAGIC 툴을 사용하여 완전주문형 방식에 의해 설계하여 통합하였다. 일단 이의 설계를 메모리와 기타로 나눈다. 메모리는 대체로 아날로그(analog) 성격이 강하므로 SPICE로 모의실험을 수행하고, 이를 근거로 MAGIC 툴로 레이아웃을 하였다. 전체 메모리 구조는 각 셀 블록이 4개가 있고 어드레스 디코더와 데이타 버스, SA(sense amplifier) 등으로 이루어져 있다. 각 셀 블록은 8개의 워드선을 갖는 64셀로 이루어졌으며, 이중 두 개의 셀블록이 하나의 SA를 공유하도록 하였고 SA의 출력에 데이타 버스가 연결되도록 하였다. 메모리 제어 신호로는 wr(write), rd(read)만을 사용하였다. 메모리 설계 후, 전체 MPU 설계를 할때 모듈셀중 AR, DR, ACC, IR, FR, PC, CON 등은 OCTTOOL로 반주문형 설계를 하였다. 반주문형 설계를 한 모듈 중, 조합 회로(CON, PC)는 로직단계의 HDL인 BDS로 구현하였는데, 이때 BDSYN, MIS2, WOLFE를 이용하여 합성과 배치배선(P&R : placement & routing)을 수행하였으며, 이에 대해 순서회로는(IR, AR, DR, ACC, FR) MSU 표준셀 라이브러리로 레지스터에 관한 네트리스트를 만들어서 BDNET에 의하여 테크놀로지 매핑을 수행한 후 WOLFE를 이용하여 배치배선을 수행하였으며, 이렇게 설계한 표준셀로 마크로셀을 설계하였다. 한편, 이에 대해 전술한 메모리 및 ALU, PC 등은 MAGIC으로 완전주문형 설계를 하였고, 이중 ALU는 다기능을 갖는 ADDER를 사용하여 제어신호(add_en, comp_en, r_shi_acc, and_en)에 의해서 출력을 선택하

도록 하였다. 설계의 통합은 MAGIC을 사용하였는데 이 과정에서 각 모듈들의 연결선 캐패시턴스가 문제될 수 있으므로 이를 최소화되도록 하였다. 현재의 설계환경으로는 통합된 MPU의 전체 모의실험은 곤란하기 때문에 각 모듈을 각각 모의실험하여 그 결과를 얻고 이를 바탕으로 전체를 통합 설계하였다. 최종적으로 이를 통합하여 레이아웃한 결과를 (그림 5)에 보였다. 레이아웃에는 MAGIC에서 제공하는 2um CMOS(double metal single poly) 설계 법칙과 데이타 파일을 사용하였으며 칩 크기는 (1516 um X 1464.5 um)이다.

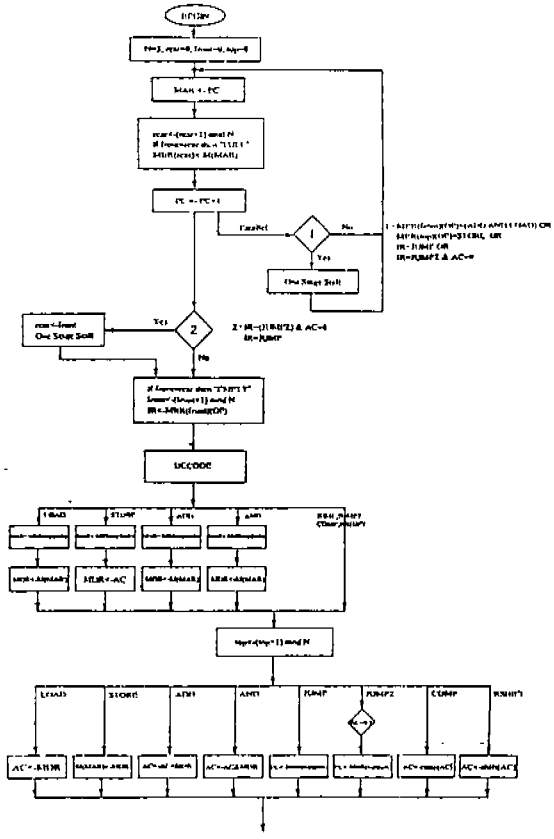


(그림 5) 기본 MPU의 레이아웃도
(Fig. 5) Layout of basic MPU

하위단계 합성 및 레이아웃시에, 각 설계시스템 공히 앞서의 VHDL 모의실험시와 레이아웃 모의실험시 시간지연 등 타이밍이 달라지는 경우가 흔히 발생하게 된다. 이를 VHDL 모의실험 후 일단 하위단계로 요구하는 경우, 로직합성(매핑포함) 등 하위단계 합성에서는 관련 모듈셀 생성기의 관련 성능개선을 위한 back-annotation에 의한 조정내지는 가변적인 셀합성을 통하여 대처할 수가 있게 되나, 이것이 곤란한 경우엔 다시 상위단계 및 VHDL 모의실험에서의 시간조정에 의해 매칭이 가능하게 된다. 현재의 본 설계환경의 여건상으로는 후자의 접근이 보다 용이하여 이를 통하여 지연시간에 관한 조정을 하였다. 따라서 이의 시간절약을 위하여서는 MPU의 각 모듈 및 메모리 설계에서의 관련 내부구조 및 지연시간 등 관련 변수의 데이타베이스화가 필요하다.

3.2 파이프라인 MPU의 모의실험

여기에서는 〈표 2〉의 파이프라인 MPU를 이를 C언어로 모의실험을 수행하였다. 2.4절에서 제안한 아키텍쳐에 의하여 다음에는 각 단의 앞끝 부분에서 조건 점검을 하여 메모리 충돌(memory



(그림 6) 파이프라인 MPU의 C 모의실험시의 명령어 흐름도
(Fig. 6) Instruction flowchart in C simulation of pipelined MPU

conflict)과 제어 장애의 경우를 점검한다. 여기서는 메모리 포트가 한 개일 경우를 가정했으므로 IF단에서 명령어를 페치하는 것과 ID단에서 피연산자를 페치하는 것, 즉, 명령어(ADD, AND, LOAD 등)은 동시에 수행할 수 없게되어 다음 명령어의 단을 일단 멈추게(stall; 스톨)하여 메모리 충돌을 해결하였다. 제어 장애는 제어가 분기되는 지점에서 발생한다. 즉, 무조건 분기하는 JUMP 명령과 JUMPZ 명령(단, ACC의 값이 0일 때)을 만나게 되면, 그 다음 명령어의 단을 일단 멈추고, MBR내에 남아 있는 무효(invalid)명령어를 지운다. 특히, JUMP 명령은 ID단에서 감지될 수 있으므로, EX 단이 진행되는 동안 다른 명령들이 속해있는 단은 무조건 일단 멈추게 하여 불필요한 수행을 방지했다. 이를 종합하여 C 모의실험을 위한 명령어 동작의 변형된 흐름도를 (그림 6)에 보였다.

파이프라인의 특성상, 여러 단이 병렬로 수행되어야 할 경우가 많이 생기게 되는데, 이때에는 프로그램 내부에서는 IF, ID, EXE 단의 순으로 순차적 처리를 하지만, 레지스터의 내용은 동시에 수행했을 때처럼 나타나도록 설계하였다. 또한, 각 단에서의 조건 점검을 위해 rear(IF), front(ID), top(EXE) 등 3개의 포인터를 두어 각 단에 해당하는 MBR내의 명령어를 참조하여 메모리 충돌 장애 및 제어 장애가 어느 단에서 발생하는지 알아내도록 했다. 이에 대한 모의실험 결과는 시간의 지연정도를 제외하고는 다음 VHDL 결과와 동일하였다.

다음에는 파이프라인 MPU 구조의 VHDL 모의실험을 수행하였다. 여기서는 C언어로 먼저 모의실험한 결과를 바탕으로 기본구조가 같도록 하였는데, 각 파이프라인 단에서 발생하는 RTL 단계의 세부동작은 앞의 〈표 2〉와 같다. VHDL 모델링에서는 행위기술이 프로세스문 간의 동시 발생(concurrent)인 동작으로 기술되며 프로세스문 안에서 모든 신호할당을 사용하므로 타이밍 문제를 상당히 주의하였다. VHDL 코드상에서의 파이프라인의 기본 구조는 명령흐름을 가지는 레지스터와 각각의 레지스터에 연관된 장애 검사부분과 제어 로직부분으로 이루어진다. 일반적으로 장애는 동적으로 검사가 이루어지므로 각 레지스터간을 체크하는 방법이 주로 타이밍문제의 근간을 이룬다고 말할 수 있다. 앞절에서 언급한 바와 같이 본 MPU의 파이프라인 구조를 VHDL로 모의실험할 시에 생기는 장애 중, 구조 장애의 경우 주로 자원의 부족(예로 메모리 충돌)에 의하여 생기고, 관련 명령(ADD, AND, LOAD 등)에 발생하며, 제어 장애의 경우 JUMP 관련 명령에서 일어난다. 데이타 장애는 존재하기는 하나 미미하므로 전술한 두 장애를 주로 검토한다. 많은 장애 해결방법이 제시되어 있지만 여기서는 다음과 같은 방법을 채택하였으며, 전반적인 기법은 C언어시의 모의실험과 동일하게 하였다. 즉, 다음과 같은 장애 검사조건에 따라 장애가 발생하게 되면 해당 파이프라인단에 _busy (EXE_busy, ID_busy) 신호를 발생시킨다. 이때 장애검사는 우선적으로 전 명령어의 선행 단의 조건검사를 통하여 각 명령어에 따라 제어신호를 발생하게 되는데 이를 요약하면 다음과 같다.

* 장애점검 및 해결

1. ID 단에서 고려되는 스톨(stall) 조건

(1) 현재 EXE단에서 JUMP 명령어이거나 JUMPZ이 채택(taken)일 경우(EXE_busy)에는 ID 단을 플러시(flush)시킨다.

(2) 현재 EXE단에서 STORE 명령어가 수행중이고 현 ID 단에서 메모리를 억세스(access)하는 명령어의 경우에는 ID 단을 스톨시킨다.

2. IF단에서 고려되는 스톨 조건

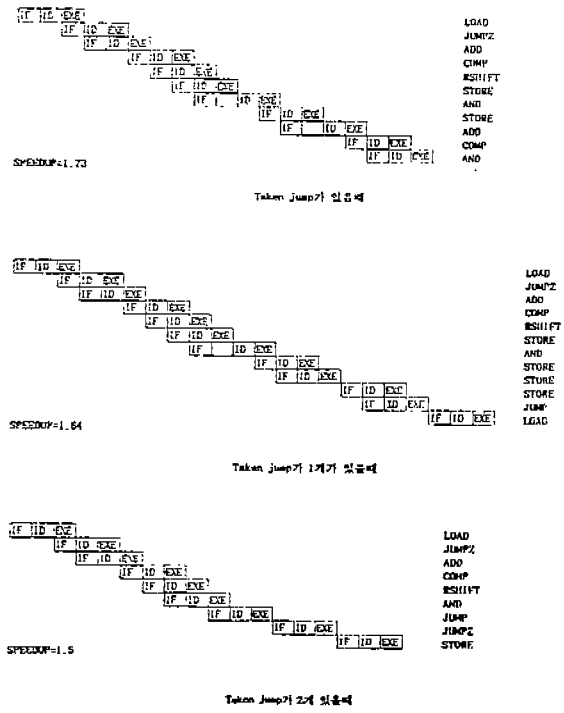
(1) 현재 EXE단에서 JUMP명령어이거나 JUMPZ이 채택(taken)일 경우(EXE_busy)에는 IF 단을 플러시시킨다.

(2) 현재 ID단에서 메모리를 사용하는 명령어가 있으면(ID_busy) 페치를 수행하지 않는다.

앞에서 제어 장애에 의한 스톨의 단 수는 2단

이고 메모리 충돌이 일어나는 구조 장애의 경우에는 앞선 명령어가 우선 순위를 갖도록 한 단을 스톨하였다. 이제 앞서에서와 같은 조건 점검(스톨 조건 검사 등)의 결과에 의해 해당 제어신호를 발생하게 된다. 이를 위하여 우선 페치, 디코드, 엑시큐트 단의 명령어를 나타내는 포인터(pointer)를 각각, 페치 명령 포인터(fetch instruction pointer; FIP), 디코드 명령 포인터(decode instruction pointer; DIP), 엑시큐션 명령 포인터(execution instruction pointer; XIP)로 하여 각 포인터들을 각각 다른 룰록, 즉, FIP는 in0_clock, DIP는 in1_clock, XIP는 in2_clock에 의해 동작한다. 즉, 앞에서와 같은 장애 검사 조건에 따라 해당단에 _busy 신호를 발생시킨다. 이에 의하여 장애 조건 점검이 끝나면, 지금까지의 각각의 포인터를 입력으로 하고 각 명령어에 따라 곧 실질적인 제어신호, 즉, IF는 in0_clock_delay, ID는 in1_clock_delay, EXE는 in2_clock_delay를 발생하게 되어 해당 단의 세부동작을 수행한다.

이와 같이 파이프라인 구조의 MPU를 VHDL 언어로 통합설계한 후 최종적으로 RTL 단계의 모의실험을 수행하였다. 모의실험을 수행할때 일단 데이타경로의 지연은 매우 작다고 가정하였고 제어부분에서는 애프터 구문(after clause; inertial delay를 의미)을 표시하여 시간조절이 되도록 하였다. 그런데 모의실험 수행시 VHDL 시뮬레이터 자체가 연구용 툴인 관계로 앞의 파이프라인 모의실험시에 다음과 같은 문제가 발생하였다. 우선 시뮬레이터가 가지는 상태(state)는 '0', '1', 'u'(undefined)의 3개이며, 여기서 시뮬레이터는 u를 '0'와 같은 상태로 보고 모의실험을 한다. 따라서 명령어 흐름을 가지는 각각의 포인터의 입력에 처음에 미확정 상태(undefined state)가 들어오면 이를 '0' 상태로 보고 시뮬레이션을 수행하여 그 출력이 '0'가 되어 '000'를 갖는 연산부호가 되는 문제가 발생하였다. 그래서 이를 해결하기 위해서 연산부호에 한 비트를 추가하여 '000'은 불법적인(illegal) 연산부호로 판정하여 언급한 문제가 발생이 되지 않도록 조정하였다. 이와 같이 주어진 각 명령어의 번호를 가지고 장애가 발생하는 경우의 수행 순서를 갖는 테스트 벡터를 구성하여 내부 데이타의 변화를 점검하여 명령수행을 확인하였다. 다음 몇가지 임의의 프로그램을 수행하여 나온 결과의 예를 (그림 7)에 나타내었다.



(그림 7) 파이프라인 MPU의 VHDL 모의실험결과 예
(Fig. 7) VHDL simulation result examples in pipelining

이들은 전절의 C언어에 의한 모의실험의 결과와 동일하였으며, 제반 예들이 장애 문제에 의하여 처리속도가 비교적 낮았다. 이는 주로 구조장애 및 제어 장애의 영향때문이다. 특히 구조장애 중, 메모리 이용 빈도가 높게 나타나게 되어 이에 의한 성능저하가 크다는 것을 알 수 있다.

이는 본 연구가 일반적인 RISC의 구조를 지극히 단순화한 구조이기 때문에 기본적으로 이들의 파이프라이닝에서의 기법을 그대로 적용하기는

곤란하였으며, 예로 데이타 저장 경로의 제약으로 인하여 스톨이 빈번히 발생하여 이것은 스케쥴링에도 제한된 운신만을 초래하였다.

4. 개선된 파이프라인 MPU의 고찰

본 장에서는 앞절의 문제를 개선하기 위한 방법을 제시한다. 첫째, 구조장애를 개선하기 위하여는 메모리에 걸리는 입출력 부담을 줄이는 것이 성능 향상의 가장 중요한 관건이 된다고 볼 수 있으므로 2개의 메모리(명령어, 데이타)구조를 도입하여, IF단에서는 명령어 메모리, ID와 EXE단에서의 수행은 데이타 메모리를 사용한다. 둘째, 제어 장애의 경우, JUMP 관련 명령어에서의 조건 및 주소계산을 ID 단에서 수행한다. 이때 조건 점검의 계산에 필요한 기능은 기존 ALU에 포함시키도록 하나, 필요시 계산모듈을 추가한다. 이와 같은 구조개선을 통하여 상당한 속도의 개선을 얻을 수가 있으나, 이 경우에도 해결해야 할 각종 장애가 발생하며 대표적인 예는 다음과 같다.

1. ID 단에서의 장애
   (1) 선행 명령어의 EXE단에서 AC를 사용하고 있을 때, JUMPZ, STORE 명령어의 ID단이 수행된다.
   (2) 선행 명령어의 EXE 단에서 MDR 혹은 데이타 메모리를 사용하고 있을 때, ID단에 MDR 혹은 데이타 메모리를 사용하는 명령어가 수행된다.
2. IF 단에서의 장애
   (1) 선행 명령어(제어관련 명령)의 ID 단에 PC가 사용되고 있을 때, 다음 명령어의 IF가 수행된다.

아울러 이들을 해결하는 간단한 방법은 앞에서와 같이 스톨을 삽입하는 방법이지만, 먼저 다음과 같은 방법, 즉 충돌이 일어나는 시간주기를 반으로 나눠, 해당 모듈 동작을 write, read(혹은 read, write)로 이원화하여 한 주기내에서 동작이 가능한가를 검사하여 이를 우선적으로 채택하는데, 특히 이는 장애 문제 1(2)에 유효하였다. 이상과 같은 방법에 의하여 (그림 7)의 모의실험을 수행할 경우, 속도는 약 2((그림 7)에서는 약 1.6)로 개선되었다. 이상에서의 속도개선은 주로 듀얼 메모리(dual memory)에 의한 구조의 개선에 의한 효과가 컸으며, 다음으로는 제어 명령의 신속한 수행에도 기인하였다. 이러한 속도개선을 위하여는 상응하는 하드웨어가 늘어나게 되며, 아울러 주어진 시간주기에서 해당 모듈의 시간할당을 위한 회로설계를 정확히 하여야 한다. 이를 더욱 확장, 개선하며 설계하는 경우의 고려해야 할 점은 아래와 같다. 첫째, 앞의 2.4절에서 언급하였듯이 파이프라인 주기의 결정에는 IF, ID, EXE로 이루어진 각 단에서의 내부적인 임계경로에 따른 최악지연(worst-case delay)을 점검하여 한 주기 내에 끝나도록 하여야 한다. 더욱 CC(condition check)에 걸리는 시간도 이의 초기 부분에 고려되어야 하며 가능한 빠른 시간에 점검이 이루어지도록 설계해야 한다. 본 절에서는 CC에 이은 디코드 제어(decode control) 신호를 각 해당 단에서 발생하게 하여 파이프라인화에 따른 제어신호 체계를 분산화하여 시간부담을 줄였다. 본 구조에서는 메모리 억세스 시간과 ALU에서의 억세스 시간이 가장 중요하며 앞의 기준(criteria)을 만족해야 한다. CC시간의 경우 일반적인 랜덤 로직(random logic) 회로로 구현을 하게 되며 이어 관련 제어신호를 발생하게 된다. 따라서 실질적인 지연 등 타이밍 시간분배를 회로 및 레이아웃설계 기법에 의하여 점검하여야 한다. 이때 수행될 로직합성과 레이아웃 수행시의 VHDL에서와의 지연 등의 타이밍 조정 방법, 전반적인 레이아웃 수행방법은 3.1절의 방법과 동일하나, 파이프라이닝의 경우 타이밍 조정 문제 및 관련 회로 설계가 보다 중요하다. 아울러 개발시간의 관점에서, 성능관련 변수화를 고려한 모듈셀의 생성에 관한 기법을 향후 강화하여야 한다.

둘째, 이상과 같이 MPU의 성능개선을 위한 내부구조의 변형도 일단은 C 언어와 VHDL 언어를

사용하여 상위단계에서 모의실험하여 설계가능함이 확인되었다. 아울러 각 계산단에서의 내부자원의 균등화 등의 변화 및 본 단계의 데이타경로(ALU 등)의 파이프라인화 등으로 엑시큐션의 속도개선을 함으로도 목표설계가 미흡한 경우에는 상위단계의 설계로 올라가 내부구조를 바꾸는 등의 조정을 수행해야 하는데, 이는 광의의 의미에서 시스템 아키텍처의 최적화설계로 볼 수 있다.

셋째, 앞에서와 같은 국지적인 메모리(local memory)의 확장은 임시적인(temporary) 레지스터의 다양화를 초래하게 되어 일반적인 RISC의 구조로 변형되어 가는 결과를 만들게 된다. 단, 이 경우 이들간의 데이타흐름을 효율적으로 경영하는 개념이 중요하게 되리라 본다. 또한 일반적인 4 내지 5 단의 RISC구조의 형태, 즉, 데이타메모리 출입(MEM), 연산결과의 저장(WB) 등에 관련된 성능을 위한 검토를 하여야 한다. 아울러 본 연구에서는 엑시큐션시의 시간을 단순 규격화하였으나, 부동소숫점 연산 등을 고려하여 이의 할당시간의 연장을 위한 고려를 해야 한다. 이 경우 물론 내부적인 스케줄링 등의 유연성도 고려되어야 한다.

5. 결 론

본 논문에서는 연구교육용으로 개발된 CAD 툴(VHDL, OCTTOOL 등)을 이용하여 소형 마이크로프로세서 설계에 적용하여 보았다. 이 경우 8비트 8명령어 RISC형태의 기본구조를 택하여 설계를 수행하였으며, 다음에는 성능 개선을 위하여 하드웨어 비용의 최적화 관점에서 구조를 검토하여 명령어 대기를 위한 버퍼 레지스터의 구조 및 각 단에서의 장애 해결을 위한 조건점검 관련 기능의 구조를 채택한 파이프라인 구조를 채택하여 이를 모의실험하였다.

먼저 기본 MPU를 C언어로 모의실험하여 정상적인 동작을 확인한 후 VHDL의 모의실험을 수행하였다. VHDL의 경우, 전체 함수를 하나의 구조로 통합하여 구현하였으며 각 요소는 프로세스문으로 묘사하였다. 이때 동작은 클록주기의 수행에 따라 순서대로 배열된 명령을 수행해 가며, CPU 내부의 레지스터의 상태를 점검, 확인하고 내부 레지스터 등의 데이타를 점검한 결과 제대로 동작함을 알 수 있었다. 한편 이의 레이아웃은 완전주문형/반주문형 설계방법을 혼합 사용하였는데, 이 경우 특히 메모리의 설계에 상당히 노력을 기울여야 했으며, 전체 MPU 시스템 통합은 MAGIC 상에서 진행하였다. 이때 각 모듈별로 모의실험을 수행하였고, 통합 시뮬레이션은 VHDL 모의실험으로 대체하였다. 다음으로 파이프라인 MPU의 경우 C와 VHDL 언어에 의한 모의실험을 시도하여 제대로 동작함과 성능개선을 확인하였으며, 이 과정중에 병목현상에 해당하는 메모리 구조 개선의 확장 등 개선점을 제시하여 모의실험을 통하여 성능개선을 확인하였다. 결론적으로, 본 연구를 통하여 비교적 저렴한 연구용 CAD 툴에 의한 MPU 설계 및 이의 성능 개선을 위한 파이프라인화의 가능성을 확인하였다. 특히 C 언어와 VHDL언어에 의한 MPU 모의실험의 방법을 정립하였으며, 레이아웃을 포함한 하위단계의 설계와의 연계를 통한 타이밍 조정 등을 통하여 보다 현실성있는 타이밍 설계 및 구조 개선이 가능함을 확인하였다. 각 단계의 모듈 설계에서 VHDL 모의실험시와 레이아웃 모의실험시 타이밍이 좀 달라지는 경우가 발생하였는데, 이는 VHDL 모의실험시, 또는 모듈셀 생성기의 설계시에 가변적인 셀합성을 통하여 조정 설계할 수 있게 된다. 한편 본 연구에서의 파이프라인 구조가 현재 보편화되어 있는 RISC 등의 파이프라인 구조와는 아키텍처의 단순화로 인한 문제점이 있기는 하나, 일단 제반동작의 모의실험 및 문제점 등을 확인하여 설계방법을 정립하였으므로, 현재 이의 명령어 및 구조를 보다 현실화하는 작업과 이 경우에서의 제반 문제, 즉, 명령어의 의존성, 분기 예측, 예외(exception) 등의 개선에 관한 연구를 진행중이며, 아울러 FPU 등 연산로직 단계의 파이프라인화, 캐쉬를 포함한 메모리경영

(memory management) 등에 관한 연구를 진행중이다. 궁극적으로는 다양한 성능의 MPU의 RISC (superscalar, superpipeline, vector 구조 등 포함) 및 VSP(video signal processor), DSP(digital signal processor)의 아키텍쳐 설계 연구를 진행중에 있다.

아울러 본 교육적인 CAD툴에 의한 궁극적인 자동화 설계 환경 구축을 위하여 보강하여야 할 점은 우선, CAD의 측면에서는 첫째, 가변적인 모듈셀 합성기를 보강 정립하여야 하며, 기본셀의 확장 및 이의 로직 및 RTL 단계의 라이브러리 매핑과 연계하는 문제를 해결하여야 한다. 즉, 모듈셀변수의 확장측면에서 이들을 구체화해야 하며 이에 따른 라이브러리 확장을 셀 생성기와 매핑관련 소프트웨어를 개발해야 한다. 둘째, 상위단계의 설계에 필요한 컴파일레이션을 수행하며 이는 VHDL에 의한 BDS 변환, VHDL 합성을 위한 중간형태에로의 변환(control data flow graph;CDFG 등), VHDL의 행위기술로부터 구조기술로의 변환 등 컴파일레이션 소프트웨어 개발을 포함해야 한다. 셋째, 상위 단계 합성을 위하여 분할, 스케쥴링과 하드웨어 할당 등 최적화 설계방법을 개발 정립하여야 하며, 이의 시스템 각 수준의 최적설계 정립을 위한 성능 관련 변수의 비용 변수화를 알고리즘, 아키텍쳐 측면에서 검토하여 매핑 등을 위한 최적화 알고리즘에 적응토록 해야 한다. 이와 더불어 파이프라이닝 구조 해석시의 제어 경로 설계의 최적화에 관한 보강이 이루어져야 한다. 다음으로 설계의 측면에서는 파이프라인시의 타이밍 제어 부분을 포함한 MPU의 전체 레이아웃을 완수하며 아울러 당면한 문제들, 즉, 혼합 모우드 해석을 포함한 대형회로의 모의해석, 지연매칭, 애널록 메모리 등을 포함한 모듈셀의 설계 및 모듈셀 생성기의 특성변수화를 포함한 보강 등의 문제를 통합 설계환경의 구축차원에서 검토해야 한다.

참 고 문 헌

[1] Z. Navabi, "Tutorial on use of VHDL for description of digital systems," Proc. ASIC conf., T12(1-8), Sept. 1991.

[2] ITD VHDL standard component library, MSU 1990.

[3] Z. Navabi et al., "Templates for synthesis from VHDL," Proc. ASIC conf., P-16(1-4), Sept. 1990.

[4] S. S. Leung et al., ASIC system design with VHDL: A paradigm, KAP, 1989.

[5] W. Chen et al., "An 8x8 DCT model using VHDL," Proc. IEEE ASIC conf., pp. 339-344, Sept. 1992.

[6] D. Gajski et al., "VHDL synthesis using structured modeling," Proc. DAC conf., paper 37.3, pp. 606-609, June 1989.

[7] A. C-H Wu et al., "An efficient multi-view design model for real time interactive synthesis," Proc. ICCAD conf., pp. 328-331, Nov. 1992.

[8] 최기영, KVUG workshop(tutorial), 1992.

[9] 조중휘, KVUG workshop(tutorial), 1992.

[10] D. Gajski, High-level Synthesis, KAP, 1992.

[11] Z. Navabi, VHDL, ch. 9, MH, 1993.

[12] H. Kanbara, "KUE-chip," Proc. IEEE ASIC seminar, P10-4(1-4), Sept. 1990.

[13] R. A. Walker, D. E. Thomas, "Design representation and transformation in the system archtect's workbench," Proc. ICCAD proceed, pp. 166-169, Nov. 1987.

[14] W. P. Birmingham, A. P. Gupta, D. P. Siewiorek, "The MICON system for computer design," Proc. DAC conf., pp. 135-140, June 1989.

[15] J. Hennessy, D. A. Patterson, Computer architecture : A quantitative approach, Morgan Kaufmann Pub. Inc., 1990.

[16] D. A. Patterson, J. Hennessy, Computer Organization & Design, Morgan Kaufmann

publishers, 1994.

[17] K. Hwang, Advanced Computer Architecture, MH, 1993.

[18] P. M. Kogge, The Architecture of Pipelined Computers, MH, 1981.

[19] S. Levitan, Pittsburgh VHDL tool, self communication.

[20] J. P. Hayes, Computer Architecture and Organization, MH, 1988.

송 낙 운

1975년 서울대 전자과 졸
1986년 Univ. Texas Austin (PhD)
1986년~89년 금성반도체
1989년 홍익대 전자공학과 교수
관심분야 : VLSI시스템 자동화 설계

박 도 순

1978년 서울대 전자과 졸
1980년 과학원 이학석사
1988년 고려대 전산과 PhD
1983년 홍익대 컴퓨터공학교 교수
관심분야 : Advanced 컴퓨터 구조, CAD

이 수 정

1992년 홍익대 전자과 졸
1993년 홍익대 대학원전자과
관심분야 : 컴퓨터 architecture 설계, ASIC design