IMS T-800 트랜스퓨터 아키텍쳐

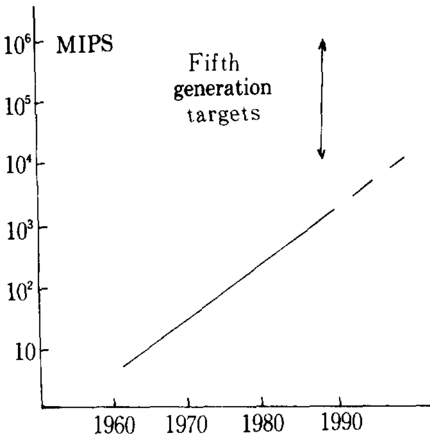
황 희 융* · 최 정 훈**
(*서울대 공대 전자계산기 공학과 교수, **석사과정)

1. 서 론

1.1 배경

과거 수십년 간 컴퓨터 시스템의 성능은 10년마다 10배씩 향상되어 왔다. (표1)

표 1 시스템 성능 발달



이것은 계속적인 집적회로 기술의 발달에 기인한 것으로 현재의 기술수준은 1개의 실리콘 웨이퍼위에 16 M 의 Static memory 또는 256개의 마이크로 프로세서를 만들 수 있다고 한다. 이런 기술의 발달은 대용량의 메모리에 다중 프로세서를 가진 병행 시스템 발달을 촉진해 고성능 슈퍼 컴퓨터가 나오게 되었다.

유럽은 이런 추세에 발맞추어 1984년 Esprit(European Strategic Program of Research and development in information Technology : 유럽 정보기술 전략 개발)에서 병렬 컴퓨터 아키텍쳐 프로젝트를 시작했고 그중 P1085의 일부로서 새로운 아키텍쳐를 가진 디바이스를 개발했다. 이의 명칭은 시스템의 빌딩 블럭으로 사용된다는 의미에서 전자회로의 빌딩블럭으로 쓰이는 Transistor와 Computer 를 결합해 Transputer라고 명명되었다.

1.2 트랜스퓨터 기본 아키텍쳐 및 패미리

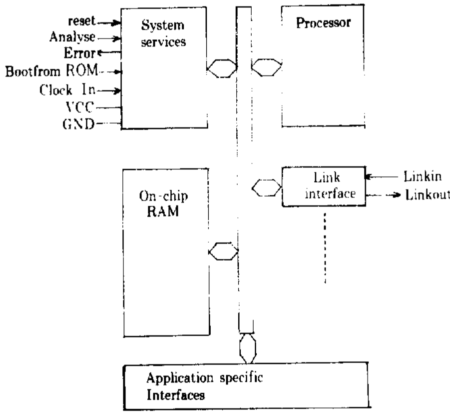
1. 2. 1 기본 아키텍쳐

일반적인 슈퍼 컴퓨터들은 다수의 프로세서와 로컬 메모리(Local memory) 들이 존재하고 이들간에 연결이 필요한데 이연결회선들은 시스템의 크기를 증가시키는 요인이 된다. 게다가 프로세서와 메모리간의 데이터 이동 시간이 중요한 성능감소 요인이 된다.

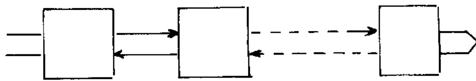
트랜스퓨터는 이들을 하나의 VLSI 칩 안에 집적시킴으로써 데이터 이동 시간을 줄이고 시스템 크기를 감소시키고자 했다. 이때 부가적인 인터페이스들도 포함하여 직접 트랜스퓨터들을 연결할 수 있게 했다. (그림1)

이들 빌딩블럭으로 경제적이고 손쉽게 큰 병행 시스템을 구축할 수 있다. VLSI 기술상 대량 생산시 디바이스의 단가를 낮출 수 있으므로 동일한 빌딩블룩들을 직접 연결해 만들어진 시스템은 매우 경제적이다.

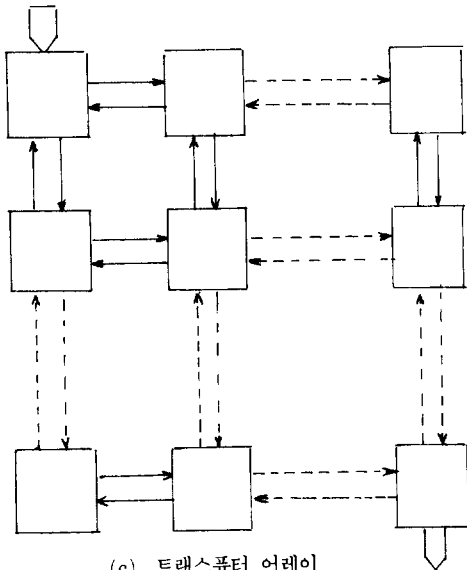
IC 에 있어서 패키지의 크기는 디바이스의 실리콘 면적보다 외부 핀(PIN)의 갯수에 의해 결정되므로, 트랜스퓨터들간의 링크는 직렬 접속(Serial connection)하도록 설계되었다. IMS C001 등의 어댑터(Adaptor)를 사용하



(a) 트랜스퓨터 블럭 다이어그램(프로세서, 메모리, 인터페이스등이 단일 칩으로 집적되어 있다)



(b) 파이프라인



(c) 트랜스퓨터 어레이

그림 1. 트랜스퓨터를 빌딩블럭으로한 시스템접속

면 공통 버스(Shared bus)를 이용할 수 있으나, 기본적으로 지점간 직접 접속함(Point-to-point connectlon)을 원칙으로 한다.(그림 1)

단일 트랜스퓨터에 의해 실행되는 프로세스도 여러개의 병행프로세스로 구성될 수 있으므로 마이크로 코드화

표 2 트랜스퓨터 시스템 성능과 트랜스퓨터 갯수

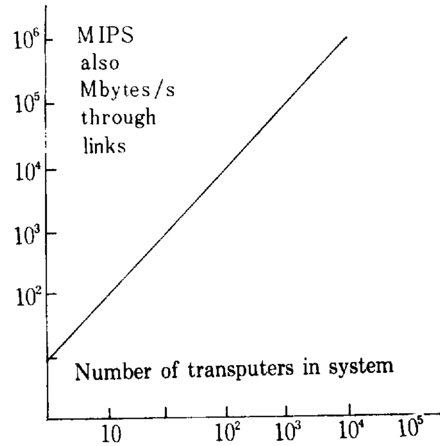


표 3 트랜스퓨터 패미리들의 부동점 연산 성능

종 류	부동점 연산 능력	비 고
IMS T414-20	10만 FLOPS이하	32비트 CPU
IMS T800-20	1.5 MFLOPS 이상	IMS T414에 FPU추가
IMS T800-30(1988 예상)	2.5 MFLOPS 이상	30 MHz clock 사용

된 스케쥴러가 2레벨의 우선 순위로 스케쥴링을 한다.

트랜스퓨터 어레이로 구성된 시스템의 전체 성능은 트랜스퓨터의 갯수(표 2), 트랜스퓨터간 통신 속도, 각 트랜스퓨터의 부동성 연산 성능(표 3)등에 좌우된다.

원래 P1085의 목적은 약 20개의 트랜스퓨터로 된 노드(Node) 하나로 강력한 워크스테이션을 구성하고, 이런 노드를 64개까지 확장했을 때 1 GFLOPS의 초고성능 슈퍼 컴퓨터를 달성하기 위한 것이었다.

1.2.2 트랜스퓨터 패미리

IMS T414(84핀칩)는 32비트 프로세서(CPU), 2K 바이트의 고속 내장(On-chip) RAM, 32비트 외부 메모리 인터페이스, 다른 트랜스퓨터와로의 통신 링크 4개(각각 2선)로 구성된다. IMS T414-20(20 MH_z)의 경우 Sequential Program에 대해 약 10 MIPS의 성능을 보인다.

IMS T800 은 IMS T414 에 부동점 연산부(FPU : Floating Point Unit)를 추가하고 4K로 On-chip memory를 증가 시켰으며 그래픽을 위한명령들이 추가되었다. CPU와 FPU연산의 오버랩(Overlap)으로 성능은 더욱향상되었으나 외부적으로는 IMS T414와 Pin-compatible하다.

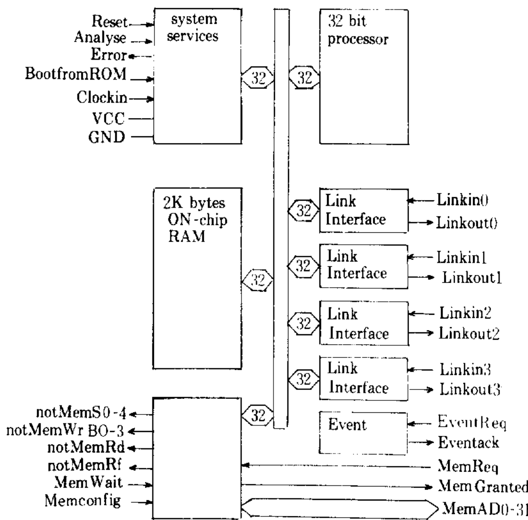


그림 2. IMS T414 출력도

IMS T212(68핀칩)는 외부 메모리 인터페이스가 16비트라는 점을 제외하면 T414와 동일하다.

IMS T212 디스크 프로세서는 16비트 CPU, RAM, ROM, 통신 링크 2개, 윈체스터 디스크 및 플로피 디스크 제어용 특수 하드웨어등으로 구성되어 있다.

이외에도 IMS C001 링크 어댑터(Link Adaptor)는 8 비트 병렬 데이터와 트랜스퓨터 1비트 직렬(Bit-serial) 데이터간에 인터페이스 기능을 수행하므로써 기존 마이크로 프로세서와 연결할 수 있도록 한다. 또한 지점간 접속대신 버스 시스템으로 연결 할 수 있도록 하는 모드도 존재한다.

IMS C002 링크 어댑터가 2M 의 DRAM 과 한개의 T414를 가진 보드를 IBM PC에 접속하는 데 사용되었다.

1.3 Occam 언어

Occam언어는 병행시스템 및 분산시스템을 프로그래밍하기 위해 개발된 언어이다. Occam은 채널을 통해 주변장치나 트랜스퓨터 상호간 통신하면서 동작하는 병행 프로세스들로 시스템을 기술하는데 적합하다. 이때 공유변수가 아닌 채널을 통해 통신하므로 공용 메모리(Shared Memory)가 없는 Loosely coupled system 의 프로그램에 적합하다. 채널을 통한 통신은 내장된 링크 인터페이스에서 하드웨어적으로 동기화를 수행하므로 소프트 웨어에 의존할 때보다 빠르다.

Occam 프로그램은 세가지 프리미티브 프로세스들로 구성된다.

V := E 수식(Expression) E를 변수(Variable) V에 치환

C ! E 수식 E 를 채널(channel) C로 출력

C ? V 채널 C 에서 변수 V 로 입력

이들 프리미티브 프로세스들을 결합하여 구조자(Construct)를 구성할 수 있다.

SEQ 성분 프로세스들을 순서대로(Sequential) 수행

PAR 성분 프로세스들을 병렬적으로(Parallel) 수행

ALT 성분 프로세스들 중 준비가 된 최초 프로세스만 수행(Alternative)

Construct자체도 프로세스이므로 다른 Construct의 성분이 될 수 있다.

이때 Indentation을 하여 프로그램 구조를 표시한다.

```
PAR
  source ? next.problem
SEQ
  compute.next.solution(this.problem.solution)
  result ! solution
```

위의 예제 프로그램은 PAR 구조자 안에 두개의 병렬 프로세스가 있다. 첫번째는 채널 source에서 변수 next.problem으로 입력하는 프리미티브 프로세스이다. 두번째는 SEQ 구조자 안에 있는 프로세스인데 이것은 다시 두개의 순서적인 프리미티브 프로세스로 구성된다. 하나는 compute.next.problem의 호출이고 두번째는 solution을 채널 result 로 출력한다.

전형적인 순차 프로그램(Sequential program)은 SEQ 구조자 안에 표현된다. IF와 WHILE 구조자가 있다. IF 구조자는 순서대로 조건을 테스트해 참이 되면 그 프로세스를 수행한다. 다음은 숫자 a, b의 대소를 비교하는 예이다.

```
IF
  a>b
    order : = gt
  a<b
    order : = lt
  TRUE
    order : =eq
```

병행 프로그램은 PAR 및 ALT 구조자 안에 표현된다. 이때 통신이 필요하므로 채널과의 입출력 프리미티브 프로세스가 포함되는 것이 보통이다. 이때 통신은 입력 프로세스와 출력 프로세스가 모두 준비 상태일때 동기화 되어 일어난다. 출력 프로세스에서 입력 프로세스로 데이터 복사가 끝나면 두 프로세스는 실행을 계속한다.

ALT 구조자안에서 여러 채널들로부터 입력을 기다린다면 다른 프로세스에 의해 최초로 동기화(해당 채널로 출력)된 채널로부터 입력을 받고 ALT 구조자 다음으로 진행한다.

Occam 버젼 1에서는 단일 데이터 타입과 1차원 배열만을 가지고 있었으나 버젼 2에서는 다수의 데이터 수형과 다차원 Array를 지원한다. 다음 예는 크기 10인 정수형 배열의 반은 채널 c에서 반은 채널 d에서 동시에 입력하고 있다.

```
[10] INT a
PAR
    c?[a FROM 0 FOR 5]
    d?[a FROM 6 FOR 5]
```

다른 구조자 중에 Replicated Construct가 있는데 이는 FOR라는 지정어를 동반한다. 다음은 Replicated SEQ의 예로서 C 나 Pascal 의 For 루프에 해당한다.

```
SEQ  i=base FOR count
       a[i] : =i
```

이는 다음의 단순한 SEQ 구조자와 동등하다.

```
SEQ
   a[base] : =base
   a[base+1] : =base+1
      .        .
      .        .
   a[base+count-1] : =base+count-1
```

2. 본 론(IMS T800 트랜스퓨터)

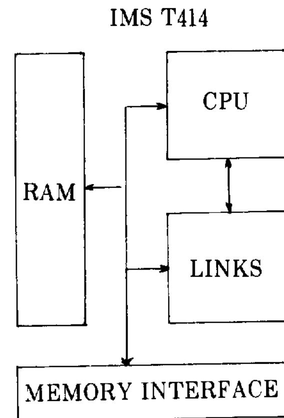
2.1 IMS T800 아키텍쳐 IMS T414

마이크로 프로세서의 부동점 연산 능력을 증진 시키는 방법으로 보통 코프로세서(Coprocessror)를 사용해왔다. 트랜스퓨터 시스템에 있어서도 개개의 부동점 연산 능력이 전체 시스템의 성능을 좌우하게 되므로 IMS T800은 T414의 부동점 연산 능력을 증진시키기 위해 개발되었다.

트랜스퓨터는 필요한 모든 것을 단일 칩안에 집적시키기를 원하므로 기존의 코프로세서는 적합치가 않았다. 그렇다고 해서 칩이 너무 많은 실리콘 면적을 차지하는 복잡한 것이 되기를 원하지도 않았다.

IMS T800은 부동점 연산용 프로세서(FPU : Floating Point Unit)를 IMS T414에 추가했으나 크기는 IMS T414에서 20%밖에 증가하지 않았다(그림 3). 이에 비해 Intel 80386용 Weitek WTL 1167 코프로세서는 3개의 칩 세트로 구성되었다.

FPU 는 마이크로 코드가 든 ROM을 포함하는데 CPU의 제어 아래 CPU의 연산과 오버랩(Overlap)되어 동작하므로써 성능을 향상 시켰다.



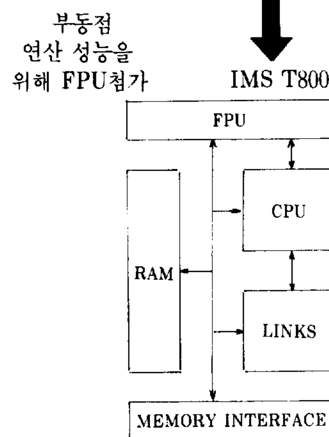


그림 3 IMS T800 블럭도

CPU 에는 A, B, C 세개의 레지스터가 하드웨어 스택을 구성하여 정수 연산 및 주소 계산에 쓰인다. CPU 로 값을 Load 하면 A 로 값이 로드되면서 이전의 A값은 B로, B값은 C로 Push 된다. CPU 값을 Store 하면 A 값이 지정된 주소로 저장되면서 이전의 B 값이 A로, C값이 B로 pop 된다. 일반적인 누산기(Accumulator)로 된 CPU는 모든 연산이 누산기를 통해야 하므로, 필요시 중간 결과를 다른 레지스터나 임시 기억 장소로 옮겨야 한다. 그러나 스택으로 구성된 CPU 는 스택 오버프로우(Overflow)가 일어났을때 이외엔 Push, Pop 에 의해 자동적으로 중간값이 저장되곤 하므로 속도가 빠르다. 게다가 피연산부(Operand)가 레지스터 스택에 저장되어 있다고 가정하므로 명령 포맷(Instruction Format)에서 피연산부가 생략될 수 있다. 따라서 명령 코드의 길이를 짧게 할수 있다.

세 레지스터 이외에도 Operand Register(피연산자 레지스터)가 있는데 명령의 Operand 를 구성하는데 쓴다. 이의 사용은 뒤에서 설명한다.

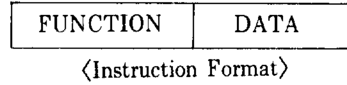
FPU에도 비슷하게 AF, BF, CF 레지스터가 하드웨어 스택을 구성하며, 부동점 연산에 사용된다. 이들에 값을 Load 하고 Store 하는 동작은 CPU 의 세 레지스터와 비슷하다. 부동점 숫자의 주소는 CPU 스택상에서 계산되고 해당 주소에서 FPU 스택으로 값을 전송한다. 이때도 CPU 의 제어하에 동작한다. CPU 스택은 주소만을 가지므로 CPU의 워드 길이는 FPU 와 무관하다. 따라서 IMS T212 처럼 16비트 CPU 에서도 동일한 FPU를 쓸 수 있다.

IMS T800 도 IMS T414 처럼 2 레벨의 우선 순위로 동작한다. 이때 낮은 레벨의 프로세스가 높은 레벨 프로세스에게 CPU를 빼앗길 때 FPU 스택의 값을 메모리에 기록할 필요가 없도록 FPU 레지스터 스택의 복사본(Duplicate) 이 존재한다. 이의 사용으로 응답시간(response time)이 2.5 마이크로 초(IMS T800-30), 3.7 마이크로 초(T800-20)로 단축되었다.

2. 1. 1 명령 인코딩(Instruction Encoding)

모든 트랜스퓨터는 똑같은 기본 명령들의 집합을 가진다. 이는 프로그램중에 가장 빈도수가 많은 연산을 compact 하게 표현할 수 있도록 선택된 동일한 형식의 명령들로 구성된다. 일반적으로 스택으로 구성된 CPU는 명령의 Operand 를 생략할 수 있으므로 명령의 길이가 짧다. 이때 피연산부는 스택을 이루는 레지스터에 들어 있는 것이 보통이다.

기본 명령의 양식은 1 바이트로 4 비트씩 두 부분으로 나뉜다.(T800의 부동점 연산 명령들은 2 바이트, 4바이트의 명령들을 추가했다.)



〈Instruction Format〉

상위 4비트는 기능 코드(Function code)이고 하위 4비트는 데이터이다. 따라서 16가지의 기능이 가능한데 LOAD, STORE, JUMP, CALL 등의 일반적인 명령들은 이렇게 1 바이트로 표현될 수 있다. 데이터가 4비트로 표현될 수 없다면 PREFIX나 NEGATIVE PREFIX 명령으로 데이터 부분의 길이를 확장할 수 있다. (그림 4)

OPERATE 명령은 DATA 부분을 Function code 의 일부로서 간주해 어떤 명령(ADD, SUBTRACT, MULTIPLY, …)을 수행할 것인지 결정하는데 쓴다. 이때 연산은 레지스터상에서 수행된다.

모든 명령은 Operand 레지스터에 하위 4 비트의 데이터부를 Load 한후, 이를 피연산자로 사용해 실행한다. 실행이 끝나면 PREFIX, NEGATIVE PREFIX 명령이외에는 Operand 레지스터를 지워 다음 명령을 준비한다.

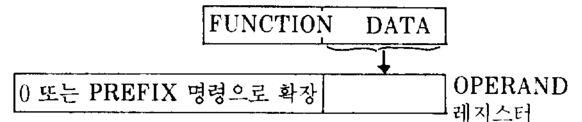


그림 4. PREFIX 명령과 오퍼랜드 레지스터.

PREFIX 명령은 한번에 4 비트씩 Operand 레지스터에 DATA 부분을 Load 한 후 4비트 쉬프트한다. 따라서 긴 데이터를 필요로 하면 먼저 PREFIX 명령을 여러번 반복해 피연산자 레지스터에 상위 비트들을 저장한 뒤 OPERATE 명령등으로 마지막 4비트를 적재(Load)하면서 실행한다.

NEGATIVE PREFIX 명령은 보수를 취해 쉬프트한다. 그러나 ADD 나 GREATER THAN 등의 대부분의 일반적인 명령들은 PREFIX 명령없이도 표현될 수 있다.

IMS T800 은 FPU 스택에의 LOAD, OPERATE, STORE 명령들이 존재한다. 또한 컬러 그래픽이나 패턴 인식등을 위한 새로운 명령들이 첨가되었다.

2.1.2 부동점 연산 명령

IMS T800 디자인 초기에 핵심적인 부동점 연산 명령 집합을 정의 했다. 이 핵심 명령(Instruction core)에는

단순한 LOAD, STORE, 산술 논리 연산 명령들이 포함된다.

성능 향상과 코드 밀도를 높이기 위해 새로운 명령들을 첨가할 것이 제안되었다.

이들을 기존 T414의 명령 세트에 추가후, 컴파일러를 만들어 실제 포트란 프로그램에 적용해 시뮬레이션을 해본 결과 다음의 명령들이 추가되었다.

FPU 스택과 트랜스퓨터 메모리 사이의 데이터 전송 명령으로 Floating-point Load 와 Floating-point Store 명령이 있다. 이 명령은 1 워드 전송 및 2 워드 전송 (Single-length & Double-length)의 두 그룹이 존재한다. 앞으로는 Double-length에 대해서만 언급하겠다.

부동점 숫자의 주소는 CPU 스택에서 계산되고, 그 주소에서 FPU로 LOAD 해 실행한다. 두개의 주소 계산용 명령이 Double-word(64 비트 실수와 정수) 엑세스 개선을 위해 추가되었다. 첫번째로 WORD SUBSCRIPT DOUBLE 인데 더블 워드값을 인덱싱하기 위한 것이다. 즉 이 명령뒤에서는 CPU의 A레지스터를 베이스 주소로 B레지스터를 Double word Offset으로 사용해 주소를 계산한다.

두번째는 DUPLICATE 로 Double word 대상의 상하위 워드의 주소를 CPU가 각각 처리해야할 필요가 있을때 사용된다.

FPU 스택의 오퍼랜드는 Length Tag(길이 표시 비트)를 가진다. 오퍼랜드가 Load되거나 계산될 때 set될 것이다. 이 태그는 부동점 연산에 필요한 명령의 갯수를 감소시킨다. 예들 들어 Floating Add Single 과 Floating Add Double 를 다 가질 필요없이 Floating Add 명령 하나로 충분하다.

Double-length 부동점수를 FPU 스택으로 Load 하는 명령에 두 가지가 있다.

Floating Load Nonlocal Double, Floating Load Indexed Double 이 그것이다. 전자는 CPU의 A 레지스터가 가리키는 주소에서 Load한다. 후자는 A레지스터값을 베이스 주소로 B 레지스터값을 Double-word Offset로 사용한다. (그림 5) 이것은 WORD SUBSCRIPT DOUBLE 명령뒤에 Floating Load Nonlocal Double 를 사용한 것과 같은 효과이다.

Floating Load Indexed 명령은 두개의 명령으로 대치 가능하다. 그러나 이의 존재는 코드 크기를 감소시킨다. Floating Load Indexed Double 명령은 2 바이트이고 동등한 두 명령은 도합 4 바이트를 차지한다. 배열의 엑세스시 이런 형태의 엑세스가 많으므로 유용하다.

그러나 floatint store 명령에는 Floating Store Nonlocal Double(또는 Single) 뿐 Floating Store Indexed 명령은 만들지 않았다. 왜냐 하면, 어떤 프로그램에서도 Load 가 Store 보다 많고, 한편 표현할 수 있는 Function의 갯수가 한정되어 있기 때문에 Load 기능을 더 최적화 한 것이다.

부동점수의 산술 연산(Add, Subtract, Multiply, Divide)을 위해 하나의 명령이 제공된다. 이 명령들은 AF, BF 레지스터상에서 수행되고 결과를 AF 레지스터에 저장하며 CF 는 BF 로 pop 시킨다. 비슷하게 Floating-Point Greater Than, Floating Point Equality 명령은 AF 와 BF 의 값을 비교한다. 단 결과는 CPU의 A 레지스터에 저장된다.

예로서 다음 Occam 프로그램은 32 비트 부동점 변수 absolute. error 가 epsilon 보다 작은지 비교해 부울 변수

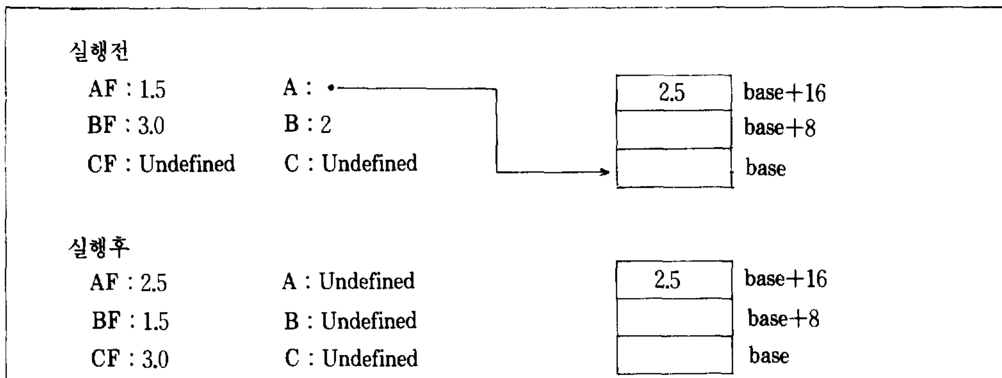


그림 5. Floating Load Indexed Double의 실행

converged 를 세트한다.

```
BOOL converged :
REAL32 absolute. error, epsilon :
SEQ
    ....
    converged : =absolute. error < epsilon
```

이를 컴파일한 코드에 Floating greater than 명령이 있다.

load local pointer	epsilon	epsilon 의 주소
floating load non-local single		FPU 로 load
load local pointer	absolute. error	absolute. error의 주소
floating load non-local single		FPU 로 load
floating greater than		CPU 스택에 결과 저장
store local	converged	converged 에 store

Load 와 Operate 를 한번에 수행하는 명령이 4개 있다. 이것은 Load indexed 명령처럼 코드 밀도(code compactness) 를 위한 것이다. load and add (또는 multiply) single (또는 double)이 그것이다. Floating-Load Add Single 은 Floating-Load Nonlocal Single 후 Floating Add를 하는 것과 같은 결과다. 나머지도 비슷하다.

덧셈과 곱셈만을 최적화 한것은 프로그램중에 이 연산을 더 많이 쓰기 때문이다.

2.2 리버모어 루프(The Livermore Loop)

리버모어 포트란 커널(Livermore Fortran Kernel)은 포트란 응용에 있어서 실제 부동점수 연산 성능을 측정하기 위해 디자인된 24개의 커널의 집합이다. 이는 리버모어 루프라는 이름으로 많이 알려져 있다.

이것이 과학계산용 프로그램의 구조에 대해 유용한 정보를 줄 수 있으므로 T800의 디자인 과정에 이들을 고려했다. 이중 Loop 7 의 포트란 코드는 다음과 같다. (그림 6)

2.2.1 스택 이용도의 최적화

CPU 와 FPU 스택의 깊이(depth)는 조심스럽게 선택되었다. 부동점 연산의 피연산자가 1차원 또는 2차원 배열일때가 많으므로 부동점 수식은 보통 주소 계산을 포함

```
DO 7 k= 1, n
  x(k)=    U(k  )+ R(Z(k)) +R*Y(k) ) +
           T*(U(k+3)+R(U(k+2)+R*U
           (k+1)  ) +
           T*(U(k+6)+R(U(k+5)+R*U
           (k+4)  )) )
7 CONTINUE
```

그림 6 리버모어 루프 7의 포트란 코드

한다. CPU 스택의 깊이는 대부분의 정수 계산과 주소 계산이 그 안에서 이루어질 수 있을 만큼 충분히 깊다. 마찬가지로 FPU 스택도 대부분의 부동점 수식 계산이 스택안에서 이루어질 수 있도록 충분히 깊다. 이때 부동점수의 주소계산은 CPU 스택상에서 행해진다.

스택 오버플로우(stack overflow)를 위한 하드웨어는 없다. 컴파일러가 수식을 검사해 스택 오버플로우가 생기지 않도록 메모리에 임시 변수를 할당한다. 계산 순서를 적절히 조정하면 임시 변수의 갯수를 최소화 할 수 있다. 이런 최적화 알고리즘이 Inmos 사의 문헌에 소개되어 있는데 이는 IMS T414의 정수 스택에 관한 알고리즘으로 IMS T800 의 CPU 스택에 적용할 수 있다.[4]

2.2.2 FPU와 CPU의 병행 연산(Concurrency)

FPU 의 부동점 연산중 CPU 는 다음에 필요한 주소 계산을 동시에 수행한다. 리버모어 루프 7 처럼 배열의 주소를 계산하고 나서 곱셈 나눗셈등 시간이 많이 걸리는 연산을 해야 할 경우, FPU 가 곱셈을 하는 도중 CPU 는 미리 다음에 필요한 주소계산을 시작할 수 있다. 이런 병행 연산은 배열 엑세스를 많이 하는 경우, 놀라운 성능 향상을 보였다.

리버모어 루프는 다음 절의 성능 평가에서 보이는 Whetstone 벤치마크(Benchmark)에서는 볼 수 없는 프로그램 구조를 가지고 있다. 특히 이들은 IMS T800내부의 병행성에 의해 좋은 성능 향상을 보일 수 있는 2차원 및 3차원 배열에의 엑세스를 포함한다.

컴파일러는 때로 오버랩을 극대화시키기 위해 주소 계산 순서를 선택할 수 있다. (그림 7)는 이런 순서의 변화를 포함해 (그림 6)의 포트란 코드를 Occam 으로 바꾼 것이다.

이제 위의 프로그램이 어떻게 IMS T800 에서 수행되는가를 수행하면서 오버랩이 일어남을 설명하자. 이 벤치

```
SEQ k= 0 FOR n
 x[k] : =     u[k]+(((r*(z[k]+(r*y[k])))+
        (t*((u[k+3]+(r*(u[k+2]+(r*u
        [k+1])))))) +
        (t*((u[k+6]+(r*(u[k+5]+(r*u
        [k+4])))))))
```

그림 7 리버모어 루프 7의 Occam 프로그램

마크에 대해 IMS T800-30 은 2.25 MFLOPS, IMS T800-30 은 1.5 MFLOPS, IMS T414-20 은 0.09 MFLOPS 의 성능을 보였다. (VAX 11/780 에서 부동점 가속기 (Floating-point accelerator)를 달고서 0.54 MFLOPS 의 성능을 보인다.)

위의 루프 7 에서 가장 먼저 계산되어야 할 부분은 z[k]+(r*y[k])이다. 이때 다음과 같은 가정을 세우자.

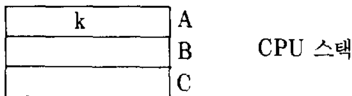
가정

r	: 부동점 변수 – Global data area에 존재
x, y, z, u	: 부동점 배열 – static.link 통해 엑세스
k	: 루프 카운트 – 프로세스의 work space 안에 존재

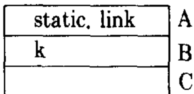
i) y[k]를 FPU 에 Load

load local k	(a)
load local static. link	(b)
load non-local pointer y	(c)
floating load indexed single	(d)

(a) 프로세스 작업공간에서 k를 CPU 스택에 load

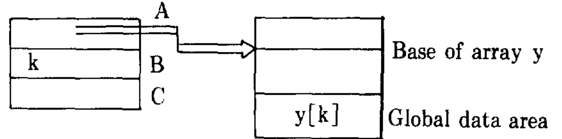


(b) 똑같이 static. link 를 load (k 값은 push 됨)



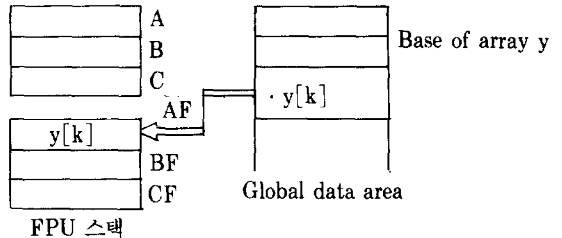
(c)현재 메모리 영역의 y번째 element 에 대한 포인터를 생성한다.

이것이 배열 y의 베이스이다. 이때 Global 데이터 영역에 y 가 들어 있으므로 Load local 이 아니라 Load non-local 명령을 쓴다.



(d)이제 A에는 배열 y의 base, B에는 첨자 k 값이 들어 있다.

y[k]값을 FPU 스택으로 load 한다.

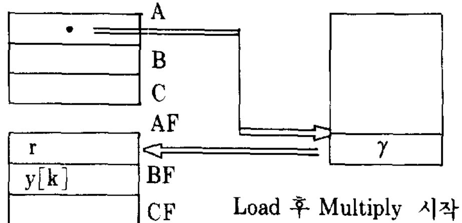


ii)r을 FPU 스택에 Load 하고 Multiply

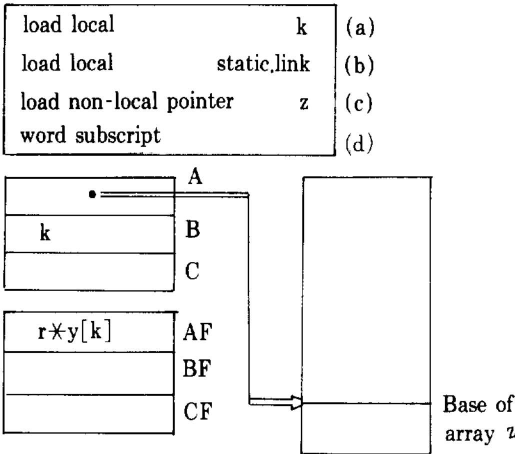
load local static. link	(a)
load non-local pointer	(b)
floating load and multiply single	(c)

(a),(b) 는 앞의 과정과 비슷하게 A에 변수 r의 주소를 생성한다. A가 가리키는 주소에서 FPU 스택으로 값을 load한 후 곱셈을 수행한다.

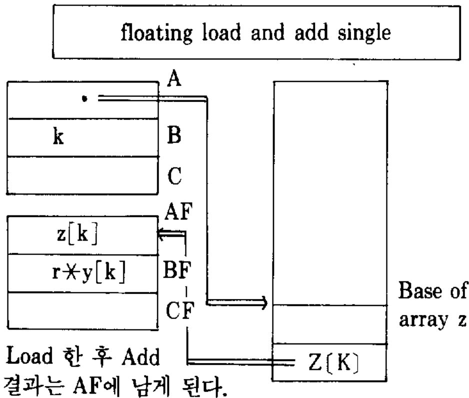
이때 r이 load되면 y[k]가 BF에 push되고 r이 AF에 들어간다. 곱셈이 끝나면 결과가 AF에 남고 CF가 BF로 pop될 것이다.



iii) 비록 FPU 스택상의 ii)의 곱셈이 여러 싸이클이 걸리더라도 그동안 CPU는 다른 명령을 수행할 수 있다. FPU 에서 곱셈이 진행되는 동안 CPU는 다음 코드들의 수행을 완료할 수 있다.



(a)-(c) 에 의해 CPU스택은 z[k]의 주소를 엑세스할 수 있다.
word subscript 명령은 뒤의 Floating-Load Add Single 명령에서 B레지스터를 offset 으로 사용하기 위한 것이다.
iv) 결국 곱셈이 끝나고 FPU의 AF에는 r*y[k]의 값이 남아 있다. 이제 z[k]를 FPU에 load하면서 덧셈을 한다. 결과인 z[k]+(r*y[k]) 값이 FPU의 AF에 남게 된다. 이때 z[k]의 load는 FPU스택에서의 곱셈이 끝난 후 동기화되어 일어 난다.



나머지 수식들의 계산도 마찬가지로 진행된다. ii)와 iii)에서 CPU와 FPU의 동작이 오버랩되는 것을 알 수 있다. 이런 오버랩은 다차원 배열의 엑세스에도 효과적이다.

IMS T800은 IMS T414의 고속 곱셈 명령 Product를 가지고 있다. 이것은 다차원 배열의 엑세스의 주소 계산에 함축되어 있는 곱셈을 위한 것이다. 이의 계산 시간은 두번째 Operand(승수)의 1값을 가진 최상위 비트 위치에 따라 다르다.

다음 Occam 프로그램은 2차원 배열의 엑세스를 포함하고 이를 컴파일한 코드는 Product 명령을 가지고 있다. (그림 8)

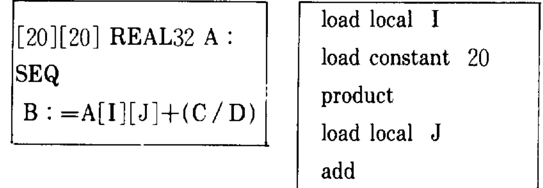


그림 8 A[I][J]의 load를 위해 A의 베이스에서 offset을 계산해야 한다.

이 경우 Product명령은 8 cycle(T800-30은 267 ns, T800-20은 400 ns)이 걸리고 전체 주소 계산은 19 cycle 이 걸린다. 이는 FPU상의 나눗셈 C/D 와 오버랩된다.

2.3 부동점 연산부 디자인(FPU design)

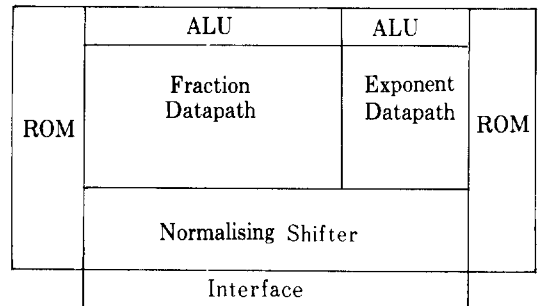


그림 9 Floating Point Unit 블럭 다이어그램

2. 3. 1 Multiplier

IMS T800과 같은 병행시스템의 구성요소를 디자인할 때는 주어진 실리콘 면적하에 성능을 극대화 시켜야 한다. 이것은 실리콘 면적을 많이 쓰어라도 성능의 극대화가 주목적인 코프로세서 디자인과는 대조적이다. (T800-20은 Intel80386에 WTL 1167 코프로세서 칩 세트(3칩)를 부착시킨 것과 맞먹는 성능을 내었다.)

코프로세서는 Flash Multiplier같은 크고 빠른 디바이스를 쓰지만 칩간의 데이터이동 시간이 필요하다. 이에 비해 트랜스퓨터는 느리지만 작은 디바이스를 CPU와 동일 칩안에 포함시키므로써 데이터 이동 시간을 줄이게 된다.

물론 벡터 처리와 같이 적은 정보로 많은 작업을 코프

2.7.2 위의 세프리미티브들을 이용한 프로그램예

이들을 프리미티브로 사용해 컬러 텍스트를 스크린에 그리는 과정이 그림 20에 나와 있다.

```
-- Draw character ch in texture F on background
texture B
PROC DrawChar (VAL INT Ch, F, B)
  SEQ
    IF
      (x + width[ch]) > screenwidth
        SEQ
          x := 0
          y := y + height
      (x + width[ch]) <= screenwidth
        SKIP
    INT Temp :
    SEQ
      Move2d(Texture[F], 0, 0, Temp, 0, 0, width[ch],
height)
      Clip2d(Font[ch], start[ch], 0, Temp, 0, 0, width
[ch], height
      Move2d(Texture[B], 0, 0, Screen, x, y, width
[ch], hoight))
      Draw2d(Temp, 0, 0, Screen, x, y, width[ch],
height)      x := x + width[ch].
```

그림 20 선택된 배경컬러 텍스트를 그리는 Occam 프로그램

현재의 X 좌표에 글자의 폭을 더한 값이 스크린 경계를 벗어 나면 안되므로 먼저 이를 테스트해 벗어날 경우 다음 라인으로 넘어간다. 이제 이 위치에 문자의 색깔이 될 texture #1을 Move2d를 써 temp에 블럭 전송한다.

다음에는 Font에 저장된 문장을 Clip2d로 블럭 전송해 글자 이외의 부분을 지운다. 이제 바탕 색깔이 될 texture #2를 Move2d로 스크린에 전송한다. 따라서 스크린에는 바탕색깔이 먼저 전송된다. 끝으로 temp를 Draw2d로 스크린에 전송해 픽셀값이 0이 아닌 글자 부분이 바탕색 위에 복사된다.

3. 결 론

IMS T800 트랜스퓨터는 병행시스템을 위한 고성능 빌딩 블럭이다. IMS T800의 디자인에서는 부동점 연산 성능 향상을 위해 코프로세서를 도입하지 않았다. 실리콘 경제성(Silicon econmics)을 고려해 IMS T800은 부동점 연산부(FPU), 중앙 처리부(CPU), 메모리 및 통신 시스템을 한 디바이스 안에 집적 시켰다. CPU와 FPU의 구조가 레지스터 스택으로 구성되고 이들의 동작이 오버랩됨으로써 성능이 향상되었다.

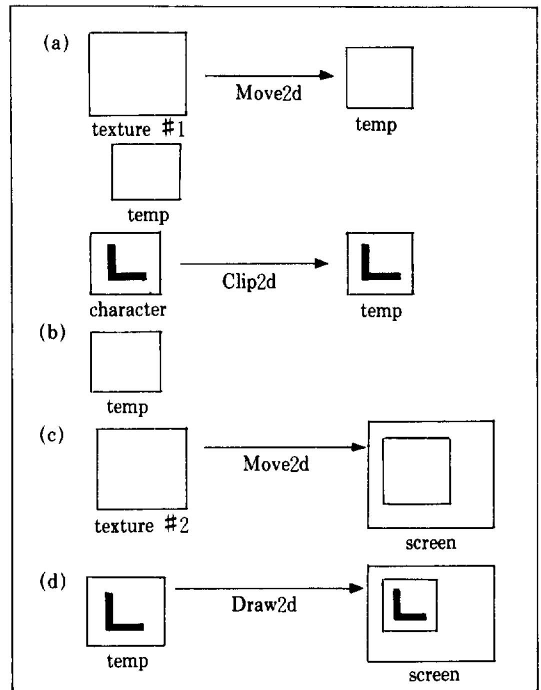


그림 21 그림 20의 실행과정

이것은 단일 칩만으로도 완전한 컴퓨터이다. 예를 들어 내장된 4 Kbytes의 메모리 만으로 외보 메모리 없이 많은 signal processing 응용에서 실용가능하다.

유럽에서 IMS T800은 슈퍼 컴퓨터를 위한 가장 강력한 기초가 되었다.

현재 에딘버러(Edinbvrgh) 대학에선 약 1000개의 트랜스퓨터로 이루어진 슈퍼컴퓨터를 개발 중인데 이것은 1988년 4월까지 1 Gbyte 메인 메모리를 가지고 동작할 것이다. 이것이 매우 큰 기계가 될 것처럼 보이겠지만 트랜스퓨터의 특징과 VLSI 기술의 발달로 1990년대 초에는 몇 입방 feet 정도의 크기로 충분히 만들 수 있다.

현재의 패키지 및 PCB 기술로도 IMS T800-20을 사용해 1입방 feet당 1.5 GFLOPS의 고성능 시스템을 만들 수 있다.

참 고 문 헌

1) Mark Homewood. David May, David Shepherd, and Roger Sherpherd. "The IMS T800 Transputer", IEEE MICRO. October, 1987
2) "The Transputer family", Inmos Ltd. June, 1986
3) "IMS T414 Transputer", Inmos Ltd. 1985
4) "The Transputer Instruction Set-A Compiler Writers' Guide", Inmos Ltd. Bristol, England. 1987
5) "The Livermore Fortran Kernel : A Computer test of the Numerical Performance Range" Tech.Report ICRL-53745, Lawrence Livermore National Laboratory, Nat'l Tech. Info. Serv., US Dept. of Commerce, Springfield, Va
6) "IMS C001 Link Adaptor", Inmos Ltd., Sep. 1985
7) "IMS C002 Link Adaptor", Inmos Ltd., Sep. 1985
8) David May, "Occam 2 Product Definition", June, 1986