Journal of The Korea Society of Computer and Information
Vol. 18, No. 7, July 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.7.001

Thumb-2 명령어 집합 구조의 병렬 분기 명령어 확장

김 대 환*

Parallel Branch Instruction Extension for Thumb-2 Instruction Set Architecture

Dae-Hwan Kim*

요 약

본 논문에서는 Thumb-2 명령어 집합 구조의 성능을 개선하기 위하여 분기 명령어와 사용 빈도가 높은 명령어를 동시에 실행하는 병렬 분기 명령어 집합을 제시한다. 제시된 기법에서는 16비트 분기 명령어와 사용 빈도가 높은 16비트 LOAD, ADD, MOV, STORE, SUB 명령어를 각각 결합하는 새로운 32비트 명령어를 도입한다. 새로운 명령어의 인코딩 공간을 제공하기 위해 사용 빈도가 낮은 기존 명령어의 레지스터 필드에 사용되는 비트 수를 줄이고 이를 통해 절약된 비트들을 이용하여 병렬 분기 명령어를 인코딩한다. 실험 결과, 제시된 방법은 코드 크기를 증가시키지 않고 전통적인 방식과 비교하여 평균 8.0%의 성능을 향상시킨다.

▸ Keywords : 명령어 집합 설계, 병렬 분기 명령어, 임베디드 프로세서, Thumb-2, ARM

Abstract

In this paper, the parallel branch instruction is proposed which executes a branch instruction and the frequently used instruction simultaneously to improve the performance of Thumb-2 instruction set architecture. In the proposed approach, new 32-bit parallel branch instructions are introduced which combine 16-bit branch instruction with each of the frequently used 16-bit LOAD, ADD, MOV, STORE, and SUB instructions, respectively. To provide the encoding space of the new instructions, the register field in less frequently executed instructions is reduced, and the new instructions are encoded by using the saved bits. Experiments show that the proposed approach improves performance by an average of 8.0% when compared to the conventional approach.

▸ Keywords : Instruction set design, Parallel branch instruction, Embedded processor, Thumb-2, ARM

•제1저자 : 김대환 •교신저자 : 김대환
•투고일 : 2013. 3. 27, 심사일 : 2013. 6. 23, 게재확정일 : 2013. 7. 14.
* 수원과학대학교 컴퓨터정보과 (Dept. of Computer Information, Suwon Science College)

I. 서 론

ARM 프로세서는 가장 널리 사용되는 32비트 임베디드 프로세서로 스마트폰, 태블릿, 가전 기기, 자동차 등의 다양한 제품에 탑재된다. 2012년 판매된 약 87억 개의 칩이 ARM 구조에 기반을 두고 있으며 스마트폰 프로세서 시장의 95%가 넘는 시장 점유율을 차지한다[1].

상용화된 가장 최신 ARM 구조는 ARMv7 (버전 7)이며 이 구조를 바탕으로 하는 프로세서는 스마트폰과 고성능 시스템을 대상으로 하는 Cortex-A (Application), 실시간 시스템을 위한 Cortex-R (Real-time), 저비용 임베디드 시스템을 위한 Cortex-M (Microcontroller)의 세 가지 유형이 있다. 2012년에 라이선스(license)된 전체 ARM 프로세서 중에서 ARMv7 구조는 전체의 77%(Cortex-M이 37%, Cortex-A가 32%, Cortex-R이 8%)를 차지할 정도로 널리 사용된다[1].

ARMv7 구조에서는 Thumb-2가 핵심 명령어 집합(Instruction Set)이다[2]. Thumb-2 명령어 집합은 기존 16비트 Thumb 명령어들[3]과 32비트 ARM 명령어들을 하나의 명령어 집합으로 통합하여 Thumb과 ARM 두 모드 사이의 모드 전환 오버헤드(overhead)를 제거한다. 추가로, 개선된 비트 처리 명령어와 같은 효율적인 명령어를 새롭게 도입한다. 이를 통해 Thumb-2는 기존 16비트 Thumb의 높은 코드 밀도를 제공하면서도 32비트 ARM 코드의 성능을 제공한다. ARM 모드에서는 명령어의 길이가 32비트, Thumb 모드에서는 16비트로 고정적인데 비해 Thumb-2 명령어는 16비트 명령어와 32비트 명령어가 혼재한다. 그림 1은 세 구조에서 이러한 명령어 흐름의 예를 보여준다.

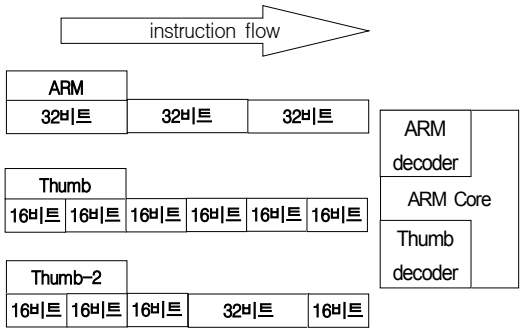


그림 1. ARM, Thumb, Thumb-2 구조에서의 명령어 흐름 예

Fig. 1. Instruction flow example in ARM, Thumb, Thumb-2 architectures

프로세서의 성능을 개선하는 다양한 연구가 진행되어 오고 있다. 대표적인 기법이 여러 개의 실행 유닛(execution unit)을 두어 명령어 수준 병렬성(instruction level parallelism)을 향상시키는 슈퍼스칼라(superscalar)[4]와 VLIW (Very Long Instruction Word)[4], [5]로 두 방법 모두 서로 독립적인 여러 명령어들을 병렬적으로 실행한다. 이러한 병렬 실행 가능한 명령어들을 슈퍼스칼라와 VLIW에서는 각각 하드웨어와 컴파일러가 결정한다. 하지만, 두 구조의 병렬 실행은 다양한 하드웨어 장치를 필요로 한다. 한 번에 여러 개의 명령어를 불러올 수 있는 명령어 인출 유닛(fetch unit), 여러 개의 명령어를 이슈(issue)하는 회로, 독립적인 명령어들을 동시에 실행시킬 수 있는 여러 개의 실행 유닛이 필요하며 슈퍼스칼라에서는 독립적인 명령어들을 판단할 수 있는 회로가 추가로 요구된다. VLIW에서의 명령어 길이는 최대 병렬 실행이 가능한 명령어 그룹들을 모두 포함하기에 지원되는 구조만큼의 병렬성을 컴파일러가 찾지 못하는 경우에는 프로그램의 코드 크기가 증가하는 단점이 있다.

본 논문에서는 프로그램의 코드 크기를 증가시키지 않고 하드웨어 추가를 최소화하면서도 명령어의 병렬성 수준을 향상시키고자 한다. 이를 위해 실행 빈도가 높으면서도 병렬 처리 구현 복잡도가 낮은 분기(branch) 명령어와 다른 명령어를 동시에 실행하는 병렬 분기 명령어를 도입한다. 분기 명령어는 전체 프로그램의 실행 사이클의 15%-25%의 비율을 차지할 정도로 실행 빈도가 높으면서[4] 대부분의 최신 구조에서는 분기 명령어에 의한 제어 의존성(control dependence)을 줄이기 위해 분기 대상 주소를 실행 단계가 아닌 디코더 단계에서 계산하기 때문에 다른 명령어와 병렬 실행을 지원하기 위한 추가적인 하드웨어의 복잡도가 낮다.

본 논문에서는 병렬 분기 명령어를 Thumb-2 명령어 집합 구조에 적용한 PABEX (PArallel Branch EXtension)이라 불리는 새로운 명령어 집합 구조를 제시하는데 이 구조에서는 16비트 분기 명령어와 ADD, MOV, LOAD 등 빈번하게 사용하는 16비트 명령어를 병렬 실행하는 새로운 32비트 명령어를 도입한다. 새로운 명령어의 인코딩 공간을 제공하기 위해 사용 빈도가 낮은 명령어의 레지스터 필드에 사용되는 비트 수를 줄이고 이를 통해 절약된 비트들을 이용하여 새로운 병렬 분기 명령어를 인코딩한다. 제시된 기법을 적용하면 기존의 Thumb-2 명령어 집합을 사용할 때와 비교하여 응용 프로그램의 실행 속도가 평균적으로 8.0% 향상되는 의미 있는 결과를 얻는다.

본 논문의 구성은 다음과 같다. II장에서는 관련 연구를 기술하고, III장에서는 병렬 분기 명령어를 제시한다. IV장에서

는 제시된 기법의 성능을 평가하며, 마지막으로 V장에서 결론을 맺는다.

II. 관련 연구

명령어 집합을 확장함으로써 프로세서의 성능을 개선하는 다양한 연구들이 진행되어 왔다[6-16]. 대표적인 임베디드 프로세서인 ARM의 32비트 ARM 명령어 집합과 16비트 Thumb 명령어 집합 구조를 개선하기 위해 다양한 기법들이 제시되어 왔다[6-15]. 일부의 기법은 16비트 Thumb 명령어 집합을 개선한다[6-9]. [6]에서는 16비트 Thumb 명령어 집합 구조를 개선하기 위하여 주소 지정 방식(addressing mode)을 확장한다. 데이터 리스트(data list)에 대한 접근 속도를 향상시키기 위하여 크기가 조정된 레지스터 오프셋 주소 지정 방식(scaled register offset addressing mode)과 사후 인덱스 주소 지정 방식(post-indexed addressing mode)을 메모리 참조 명령어인 로드와 저장 명령어에 도입한다. 실험 결과, 제시된 방법은 16비트 Thumb 명령어 집합의 성능을 평균 8.5%의 향상시킨다. [7], [8]과 같은 기법들은 사용 가능한 레지스터의 수를 증가시킴으로써 성능을 개선한다. 반면에 [9]과 같은 기법에서는 코드 크기를 더욱 감소시키기 위해 분할된 레지스터 확장(partitioned register extension)을 제시한다.

다른 기법들은 32비트 ARM 명령어 집합 구조 개선에 초점을 맞춘다[10-15]. [10]에서는 네트워크, 멀티미디어 처리 등의 영역에서 요구되는 워드보다 작은 단위의 데이터를 처리할 수 있는 비트 부분 확장(bit section extension) 기법을 도입한다. [11]에서는 사용 빈도가 낮은 조건 필드(conditional field) 인코딩 비트들을 레지스터 기술에 대신 할당함으로써 가용 레지스터의 수를 증가시킨다.

ARM DSP 명령어 확장[12]은 신호 처리 응용 프로그램의 실행 속도를 향상시키기 위해 DSP 명령어를 ARM 명령어 집합에 추가한다. 이 명령어 집합은 ARM926EJ-S, ARM946E-S, ARM966E-S 등의 다양한 프로세서에서 지원되며 Audio 응용 프로그램의 경우 70%까지 속도를 개선한다.

ARMv6 구조[13]은 SIMD (Single Instruction Multiple Data) 명령어 집합을 지원한다. 이 명령어 집합에서는 두 개의 16비트 또는 네 개의 8비트 산술 연산 등을 동시에 실행할 수 있는 등 여러 개의 데이터에 대해 동일 연산을 병렬적으로 수행할 수 있다. 이 집합은 ARM11 프로세서 등에 포함되었으며 오디오 및 비디오 코덱, 그래픽, 이미지, 음성 처리 등의 프로그램에 유용하게 적용된다. ARMv7 구조는 개선된 SIMD 집합인 NEON[14]을 지원한다. 이 구조는 비디오, 오디오 코덱 응용 프로그램의 속도를 네 배 정도 개선하며 Cortex-A9 등의 프로세서에 채택된다.

ARM 구조 개발팀은 병렬성(parallelism)을 향상시키기 위해 다양한 기법들을 제시한다[15]. 그 기법들은 멀티프로세싱 (multiprocessing), 스레드 단위 병렬성 (thread-level parallelism), 가변 실행 시간(variable execution time), 서브워드 병렬성(subword parallelism), 유사 DSP 연산(DSP-like operations) 등을 포함한다.

2011년에 제시된 ARMv8 구조는 기존 명령어 집합에 64비트 응용프로그램에서 활용 가능한 A64라는 명령어 집합을 추가하여 64비트 연산과 확장된 가상 주소 방식(virtual addressing)을 지원한다. A64의 각 명령어는 피연산자와 상수 값을 위한 연속적인 비트 필드를 가지는 구조를 가진다. 이를 통해 ARMv8 구조에서는 디코딩이 단순화되며 독립적인 디코딩 기능은 고급 분기 예측을 용이하게 한다. 이 구조는 성능 향상을 위해 31개의 64비트 범용 레지스터를 추가하고 LDM/STM (load/store multiple) 명령어를 제거하고 구현 복잡도 대비 상대적인 이득이 작은 조건부 명령어를 감소시킴으로써 실질적인 명령어 집합을 단순화하여 프로세서 메모리 및 시스템 효율을 향상시킨다. 2014년에 64비트를 지원하는 차세대 ARMv8 구조 기반의 Cortex-A57과 Cortex-A53제품이 출시될 예정이다.

III. 명령어 집합 설계

1. 제시된 병렬 분기 명령어

본 논문에서는 분기 명령어와 사용 빈도가 높은 명령어를 동시에 실행하는 병렬 분기 명령어를 제시한다. 병렬 분기 명령어는 16비트 분기 명령어와 사용 빈도가 높은 16비트 명령어를 결합하는 32비트 명령어이다. 제한된 인코딩 공간 때문에 PABEX는 분기 명령어와 결합하여 병렬 실행할 명령어 중 사용 빈도가 높은 명령어를 선정하다. Thumb-2에서 분기 명령어와 의존 관계가 없어서 병렬 실행이 가능한 명령어의 빈도를 측정한다. 사용 빈도가 높은 명령어는 프로그램마다 상이할 수 있으므로 명령어 빈도 측정을 위한 프로그램은 널리 사용되는 대표적인 임베디드 벤치마크인 MiBench, CPU 벤치마크인 SPEC2006, 멀티미디어 벤치마크인

MediaBench에서 상호 보완적으로 선정한다. 상세한 내용은 IV장의 표 3을 참고하기 바란다. 이 벤치마크에서 명령어들의 빈도를 측정한다. 분기 명령어와 데이터 의존 관계가 없어서 분기 명령어와 병렬 실행이 가능한 명령어는 LOAD, ADD, MOV, STORE, SUB의 순으로 빈도가 높으며 각 빈도는 각각 전체 실행 사이클 수의 3.8%, 1.6%, 1.3%, 1.0%, 0.3%이다. 상세한 성능 평가는 IV장의 표 5를 참고하기 바란다.

표 1. PABEX 병렬 분기 명령어 조합
Table 1. PABEX parallel branch instruction combination

분기 명령어	분기 명령어와 결합 가능 명령어
BEQ	ADD, SUB, MOV, LDR, LDRH, LDRSH, LDRB, LDRSB, STR, STRH, STRB
BNE	
BCS	
BCC	
BMI	
BPL	
BVS	
BVC	
BHI	
BLS	
BGE	
BLT	
BGT	
BLE	
B	

표 1은 PABEX에서 제시되는 병렬 분기 명령어를 보여준다. 분기 명령어와 병렬 실행이 지원되는 명령어는 ADD, SUB, MOV 명령어와 LOAD 및 STORE 명령어에 해당되는 LDR (Load word), LDRH (Load halfword), LDRSH (Load signed halfword), LDRB (Load byte), LDRSB (Load signed byte), STR (Store word), STRH (Store halfword), STRB (Store byte) 이다. 이 명령어들은 BEQ, BNE 등 ARM 구조에서 지원하는 15개의 조건 분기 명령어와 결합 가능하다. 병렬 분기 명령어의 연상기호(mnemonic)은 두 명령어가 병렬 실행된다는 점을 기술하기 위해 VLIW 구조에서처럼 '||' 기호로 표현한다. 예를 들어, Thumb-2에서의 'ADD R1, R2, R3' 명령어와 'BEQ offset'를 병렬 처리하는 32비트 명령어는 PABEX에서 'ADD R1, R2, R3 || BEQ offset'으로 표현된다.

그림 2는 제시된 병렬 분기 명령어의 예를 보여준다. 그림 2 (a)와 (b)는 각각 기존 Thumb-2 명령어 집합의 코드와 제시된 병렬 분기 명령어 코드를 보여준다. 그림 2 (a)의 ADD와 BLT 명령어는 서로 의존 관계가 없어서 그림 2 (b)와 같이 하나의 병렬 분기 명령어로 결합할 수 있다. 그림 2 (a)에서는 ADD 명령어의 수행이 끝난 후에 BLT 명령어가 실행되지만 병렬 분기 명령어에서는 두 개의 명령어가 동시에 수행되어 실행 사이클이 한 사이클(cycle) 절약된다.

```
ADD   R1, R2, R3
CMP   R4, R5
BLT   |loop1|
```

(a) C 코드

```
CMP   R4, R5
ADD   R1, R2, R3 || BLT   |loop1|
```

(b) C 코드

그림 2. 병렬 분기 명령어 예
Fig. 2. Parallel branch instruction example

Thumb-2의 명령어 인코딩 공간은 대부분 사용되기 때문에 제시되는 새로운 명령어를 인코딩하기 위한 공간은 거의 없다. 따라서 병렬 분기 명령어의 인코딩 공간을 생성하기 위하여, 제시된 방식에서는 두 개의 명령어들에서 레지스터 필드에 사용되는 비트 수를 감소시킨다. 이러한 변경은 사용 빈도가 낮은 명령어에 국한되기에 프로그래머에게 부담을 주지는 않으리라 기대된다. 표 2는 PABEX에서 제한되는 Thumb-2 명령어와 제한 사항을 보여준다. 자주 사용되지 않으면서도 새롭게 추가되는 병렬 분기 명령어를 내장할 수 있는 충분한 피연산자 비트 수를 가지는 두 명령어가 선택된다. 해당 명령어는 LDMIA (Load multiple)과 STMIA (Store multiple)이다. LDMIA와 STMIA 명령어는 각각 메모리와 범용 레지스터들 사이에 여러 개의 데이터를 로드하고 저장한다. 제시된 방법은 각 명령어에서 레지스터 리스트 필드를 한 비트 감소시켜 이를 이용한다. LDMIA와 STMIA 명령어에서 레지스터 R7을 전송 가능한 리스트에서 제외하여서 해당 레지스터 리스트를 R0~R7에서 R0~R6로 축소한다. R7에 사용하는 비트는 축소된 Thumb-2 명령어(R7 비트=0)와 새로운 PABEX 명령어(R7 비트=1)를 구분하는 데에 사용된다. 이러한 축소에 의한 성능 저하는 해당 명령어들의 사용 빈도가 낮기에 미미하다. LDMIA, STMIA 명령어는 총 실행 사이클의 단지 0.0001%만을 차지한다. 상세한 성능 평가는 IV장의 표 7을 참고하기 바란다. 또한, Thumb-2 명령어 집합 구조는 컴파일러를 대상으로 설계되어서 이미 직교적(orthogonal)이지 않다는 사실에 유의하자 [2], [3].

표 2. 제한되는 Thumb-2 명령어
Table 2. Reduced Thumb-2 instruction

명령어	설명	제한사항
LDMIA Rn!, 〈register_list〉	Rn 값을 시작 주소로 메모리 블록을 레지스터로 로드	register list를 R0 ~ R7 에서 R0 ~ R6 으로 제한
STMIA Rn!, 〈register_list〉	레지스터들을 Rn 주소의 메모리에 저장	

제시된 병렬 분기 명령어는 기존의 VLIW (Very Long Instruction Word) 머신의 명령어 집합과 개념적으로는 유사성을 가지나 병렬 분기 명령어는 기존의 16비트 명령어 두 개를 결합한 32비트 명령어이므로 코드 크기를 증가시키지 않는다. 제시된 기법은 기존의 지연 분기(delayed branch) 명령어[4]와도 구분되다. 제시된 기법이나 지연 분기 두 가지 방법 모두 제어 의존성을 감소시켜 성능을 향상시키려고 하는 점에서는 유사하지만 실행 방식에서는 커다란 차이가 있다. 지연 분기 명령어 방식에서는 분기 명령어와 분기 명령어의 다음 주소에 위치하는 명령어가 파이프라인 방식으로 동작하며 따라서 다음 주소의 명령어는 분가 명령어의 다음 사이클에 실행된다. 반면에 제시된 기법에서는 병렬 분기 명령어를 구성하는 분기 연산과 사용 빈도가 높은 연산을 동시에 병렬적으로 실행한다. 또, 지연 분기 명령어는 슈퍼스칼라 머신에서는 다수의 명령어들이 지연 슬롯에서 실행되어야하기 때문에 하드웨어 구현이 복잡해지는 단점이 있어서 사용되지 않는 반면 제시된 명령어 구조는 특정 구조에 의존적이지 않다.

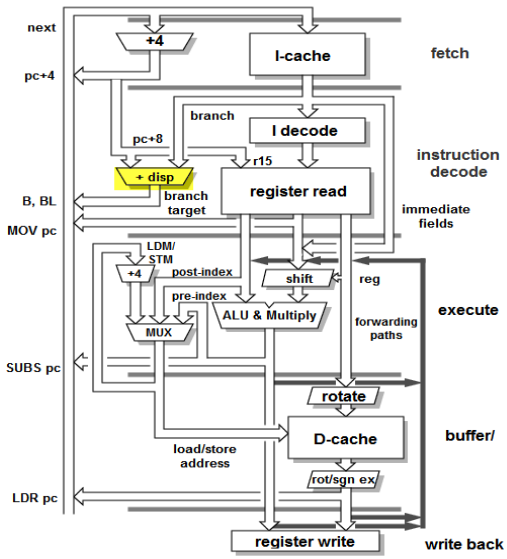


그림 3. 분기 전용 덧셈기가 있는 StrongARM의 데이터 경로
Fig. 3. StrongARM data path which has a dedicated branch adder

병렬 분기 명령어 지원하기 위한 하드웨어 장치를 고려해 보자. 분기 대상 주소는 보통 현재 프로그램 수행 위치를 저장하는 레지스터인 PC (Program Counter) 값과 분기 명령어의 오프셋의 합으로 결정되며 이 계산을 수행하기 위해서는 덧셈기(adder)가 필요하다. 이를 위해 ARM7TDMI와 같은 일부의 ARM 프로세서는 ALU (Arithmetic Logic Unit)를 이용하며 StrongARM과 같은 일부의 프로세서는 전용 분기 덧셈기를 가진다. 전용 덧셈기를 사용하는 경우 주소 계산은 레지스터 읽기 단계와 병렬적으로 동작 가능하다. 분기 명령어의 분기 주소는 종종 디코더 단계에서 미리 계산하여 분기가 발생하는 경우의 분기 비용(penalty)를 감소시키게 된다. 그림 3은 StrongARM의 데이터 경로(data path)를 보여준다. 명령어 디코더 단계에서 분기 명령어의 필드 값으로 주어지는 오프셋 값을 사용하여 분기 주소 계산을 수행한다. 분기 명령어 실행 유닛은 분기 대상 주소를 결정하기 위한 덧셈기와 PSR (Program Status Register)에서 조건 값을 읽어서 분기 명령어의 조건과 비교하고 프로그램 수행 위치를 지정하는 하드웨어로 구성된다. 분기 전용 덧셈기가 있는 ARM 구조에서는 분기 여부 결정을 위한 조건 비교기만 추가하면 분기 명령어와 다른 명령어를 병렬적으로 실행할 수 있게 된다.

2. PABEX 명령어 인코딩

Thumb-2에서는 16비트 명령어와 32비트 명령어가 함께 존재하며 모든 명령어에 대해 16비트 정렬한다. 32비트 명령어는 두 개의 하프워드(halfword), hw1과 hw2로 구성되고 hw1이 하위 주소 번지에 위치한다. 그림 4는 32비트 명령어가 A번지~A+3번지의 바이트 주소에 위치하는 경우의 바이트와 하프워드 순서를 보여준다. Hw1에 A번지 바이트, A+1번지 바이트가 위치하고 A+2, A+3 번지의 바이트는 hw2에 위치한다. 16비트 Thumb-2 명령어인 경우는 hw1만이 존재한다.

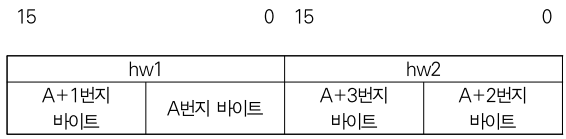
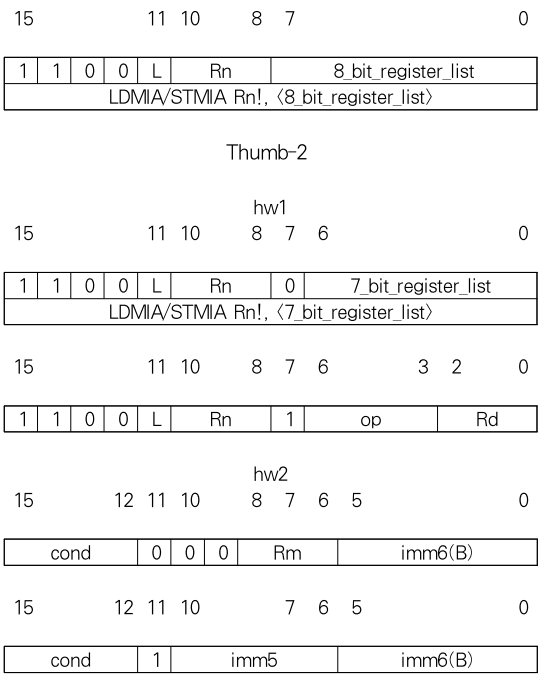


그림 4. 32비트 Thumb-2 명령어 형식
Fig. 4. 32-bit Thumb-2 instruction format

그림 5는 PABEX에서 제시된 병렬 분기 명령어를 인코딩하기 위해 Thumb-2의 16비트 LDMIA (L 비트=1)와

STMIA (L 비트=0) 명령어 형식을 변경한 것이다. 제시된 기법에서는 두 명령어에서 전송 가능한 레지스터 리스트를 R0~R7에서 R0~R6로 축소하여 R7에 해당하는 비트(hw1의 비트 7)을 절약한다. 절약된 비트를 이용하여 병렬 분기 명령어를 도입한다. 이 비트는 LDMIA과 STMIA 명령어에 대해서는 0, 새로운 PABEX 명령어에 대해서는 1의 값을 가진다. LDMIA, STMIA 명령어는 16비트 명령어인데 비해 PABEX 명령어는 32비트 명령어임에 유의하자.

Hw1의 비트 0부터 2, 비트 8부터 10는 각각 두 개의 레지스터, Rd와 Rn을 인코딩한다. 비트 0에서 2는 병렬 분기 명령어의 목적지 레지스터를 인코딩하고 비트 8부터 10은 로드, 저장 명령어의 경우 베이스 레지스터를, 데이터 처리 명령어의 경우 첫 번째 피연산자 레지스터를 인코딩한다. 제한된 인코딩 공간 때문에 두 레지스터의 범위는 16비트 Thumb 명령어와 마찬가지로 R0~R7로 제한된다.



op	bit 11	명령어	피연산자 형식
0000	0	STRB	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
0001	0	LDRB	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
0010	0	STRH	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
0011	0	LDRH	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
0100	0	STR	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
0101	0	LDR	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
0110	0	LDRH	Rt,[Rn,Rm, LSL #1]
	1	-	-
0111	0	LDR	Rt,[Rn,Rm, LSL #2]
	1	-	-
1000	X	-	-
1001	0	LDRSB	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
1010	0	MOV	Rd,Rm
	1		Rd, #imm
1011	0	LDRSH	Rt,[Rn,Rm]
	1		Rt,[Rn,#imm]
1100	X	-	-
1101	0	SUB	Rd,Rn,Rm
	1		Rd,Rn,#imm
1110	0	ADD	Rd,Rn,Rm
	1		Rd,Rn,#imm
1111	0	ADD	Rd,Rn,Rm LSL #2
	1	-	-

PABEX

그림 5. Thumb-2 LDMIA/STMIA 명령어 형식과 이 형식을 이용한 PABEX 명령어 형식
Fig. 5. Thumb-2 LDMIA/STMIA instruction format and its corresponding PABEX instruction format

Hw1의 비트 3에서 6은 분기 명령어와 결합할 명령어의 연산코드(op)를 기술한다. 연산코드는 병렬 분기 명령어에서 지원되는 ADD, SUB, MOV, LDR (Load word), LDRH (Load halfword), LDRSH (Load signed halfword), LDRB (Load byte), LDRSB (Load signed byte), STR (Store word), STRH (Store halfword), STRB (Store byte) 명령어를 지시한다. 로드와 저장 명령어에 대해 오프셋 주소 지정 방식(offset addressing mode)을 지원하는데 이 방식은 유효 주소(effective address)를 기준 레지스터(base register)의 값에 오프셋 값을 더한 값으로 계산한다. 오프셋 값은 즉시(immediate), 레지스터(register) 중 하나로 명시될 수 있으며 LDR와 LDRH에 대해서는 각각 32비트 워드 배열 및 16비트 하프워드 배열의 요소를 접근하는 데에 유용한 2비트 및 1비트 왼쪽 시프트된 레지스터 오프셋 주

소 지정 방식을 추가로 제공한다. ADD, SUB 명령어는 두 번째 피연산자로 레지스터와 즉시값(immediate)를 지원하며 주소 계산에 널리 사용되는 ADD 명령어에 대해서는 두 번째 피연산자로 2비트 왼쪽으로 시프트된 레지스터 형식을 추가로 지원한다.

Hw2는 분기 명령어의 조건과 오프셋을 기술한다. 비트 12에서 비트 15는 EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE, AL의 15가지 조건을 기술한다. 비트 0에서 비트 5는 분기 명령어의 오프셋을 지정한다. 비트 6에서 비트 10은 비트 11의 값에 따라 다르게 해석된다. 비트 11이 1이면 비트 6에서 비트 10은 5비트의 즉시값으로 해석된다. 비트 11이 0이면 비트 6에서 비트 8은 레지스터 번호로 해석되며 로드, 저장 명령어에서는 오프셋 레지스터, 데이터 처리 명령어의 경우 두 번째 피연산자 레지스터로 해석된다.

IV. 성능 평가

제시된 방법의 효율성을 측정하기 위하여 Qemu ARM 시뮬레이터[17]에서 Cortex-M3 프로세서[18]를 대상으로 성능을 평가한다. 시뮬레이터는 새로운 PABEX 명령어 집합을 지원하기 위해 확장된다. 각 프로그램에 대해 ARM Realview 컴파일러가 Cortex-M3 프로세서 명령어 집합인 Thumb-2 명령어 집합 어셈블리 코드를 생성하고 생성된 어셈블리 중에서 PABEX의 병렬 분기 명령어로 변경될 수 있는 코드 패턴을 기본 블록(basic block)에서 찾아내어 변경한다. 컴파일러의 명령어 재배치(instruction scheduling)에 의한 영향을 제거하여 Thumb-2 코드와 동일한 조건에서 성능을 비교하기 위하여 제시된 기법에서는 추가적인 명령어 재배치를 수행하지 않는다. ARM Realview 컴파일러가 생성한 코드를 대상으로 제시된 기법을 적용하기 전인 Thumb-2 코드와 적용 후인 PABEX 코드의 성능을 Qemu ARM 시뮬레이터로 측정한다.

성능 평가를 위해 사용한 벤치마크 프로그램은 임베디드 벤치마크인 MiBench, CPU 벤치마크인 SPEC2006, 멀티미디어 벤치마크인 MediaBench에서 상호 보완적으로 선정한다. MiBench에서 이미지 코덱인 jpeg, 음성 코덱인 adpcm, 데이터 암호화 프로그램인 sha를. SPEC2006에서 데이터 압축 프로그램인 164(gzip)과 mcf를 선정한다. MediaBench에서는 동영상 압축 코덱인 mpeg과 데이터 암호화 및 인증 프로그램인 pegwit을 선정한다. Jpeg과 adpcm은 MiBench와 MediaBench 모두에 포함된다. 표 3은 사용된 벤치마크 프로그램에 대한 기술한다.

표 3. 벤치마크 프로그램
Table 3. Benchmark program

벤치마크	설명
jpeg	이미지 압축 표준
mpeg	동영상 압축 표준
164	GNU gzip 데이터 압축
sha	보안 해시 알고리즘
adpcm	음성 인코딩 및 디코딩
pegwit	공개키 암호화 및 인증
mcf	차량 스케줄링 조합 최적화 문제

표 4는 Thumb-2 명령어 집합 구조와 비교하여 제시된 기법인 PABEX의 성능을 보여준다. 이 표에서는 일곱 개의 각 벤치마크 프로그램에 대하여 Thumb-2의 실행 사이클과 PABEX의 실행 사이클의 수, 그리고 PABEX 사이클 수와 Thumb-2 사이클의 비율을 보여준다. 벤치마크 프로그램에 대해 평균 8.0%의 성능이 향상되며 jpeg, mpeg, 164, sha, adpcm, pegwit, mcf에서 평균적으로 각각 5.7%, 6.1%, 8.4%, 8.4%, 5.4%, 12.3%, 10.1% 성능이 개선된다. 성능 향상의 범위는 5.4%에서 12.3%이다. Pegwit 프로그램의 경우 가장 크게 성능(12.3%)이 향상된다. 이는 프로그램이 루프 내에서 병렬 분기 명령어로 변환될 수 있는 분기 명령어의 수가 많기 때문이다.

표 4. Thumb-2 대비 PABEX의 효율
Table 4. Efficiency of PABEX compared to Thumb-2

벤치마크	Thumb-2에서의 총 실행 사이클 수	PABEX에서의 총 실행 사이클 수	PABEX / Thumb-2
jpeg	11,937,548	11,258,544	94.3
mpeg	57,170,864	53,679,136	93.9
164	9,948,213	9,116,250	91.6
sha	14,061,131	12,881,624	91.6
adpcm	16,663,431	15,771,579	94.6
pegwit	50,034,843	43,861,701	87.7
mcf	117,378,155	105,511,838	89.9
평균	-	-	92.0

표 5는 병렬 분기 명령어에 포함할 명령어를 선정하기 위해 Thumb-2의 16비트 명령어 중 분기 명령어와 결합 가능한 명령어의 빈도를 측정한 결과를 보여준다. 표 3의 벤치마

표 5. 분기 명령어와 결합 가능한 16비트 명령어의 등장 빈장 빈도
Table 5. Frequency of the 16-bit instruction that can be combined with a branch instruction

벤치마크	Thumb-2에서의 총 실행 사이클 수	LOAD		ADD		MOV		STORE		SUB	
		사이클 수	비율 (%)	사이클 수	비율 (%)	사이클 수	비율 (%)	사이클 수	비율 (%)	사이클 수	비율 (%)
jpeg	11,937,548	209,680	1.8	129,040	1.1	122,032	1.0	209,956	1.8	8,296	0.1
mpeg	57,170,864	973,226	1.7	609,813	1.1	137,403	0.2	1,714,817	3.0	56,469	0.1
164	9,948,213	558,166	5.6	133,225	1.3	18,905	0.2	118,905	1.2	2,762	0.0
sha	14,061,131	2	0.0	779,681	5.5	399,783	2.8	41	0.0	0	0.0
adpcm	16,663,431	442,561	2.7	0	0.0	308,416	1.9	1	0.0	140,874	0.8
pegwit	50,034,843	5,413,229	10.8	362,544	0.7	147,574	0.3	102,670	0.2	147,125	0.3
mcf	117,378,155	4,916,701	4.2	1,763,529	1.5	3,166,814	2.7	744,080	0.6	1,275,193	1.1
평균	-	-	3.8	-	1.6	-	1.3	-	1.0	-	0.3

크 프로그램에 대하여 ARM Realview 컴파일러 툴 체인을 사용하여 Thumb-2 어셈블리 코드를 생성하고 분기 명령어와 병렬 실행 가능한 명령어를 기본 블록에서 찾아서 ARM Realview 시뮬레이터로 동적 실행 빈도를 측정한다. 분기 명령어와 병렬 처리 가능한 명령어 중 빈도가 높은 명령어는 LOAD, ADD, MOV, STORE, SUB의 순이며 등장 확률은 각각 3.8%, 1.6%, 1.3%, 1.0%, 0.3%이다. 분기 명령어와 결합 가능한 LOAD 명령어의 빈도 범위는 전체 실행 사이클의 0.0%에서 10.8%이다. ADD의 범위는 0.0%에서 5.5%, MOV는 0.2%에서 2.8%, STORE는 0.0%에서 3.0%이며 SUB 명령어는 0.0%에서 1.1%사이의 범위에 존재한다. 제시된 기법의 평균 성능 향상인 8.0%는 분기 명령어와 병렬 실행 가능한 LOAD, ADD, MOV, STORE, SUB 명령어의 평균 비율의 총합과 동일하다.

표 6은 분기 명령어의 등장 빈도를 보여준다. 제시된 병렬 분기 명령어의 길이는 32비트로 16비트 분기 명령어와 사용 빈도가 높은 명령어를 결합한다. 따라서 본 실험에서는 16비트 분기 명령어의 빈도를 측정한다. 분기 명령어의 평균 등장 빈도는 12.3%이며 빈도의 범위는 7.0%에서 22.1%이다. mpeg에서 등장 빈도가 7.0%로 가장 낮으며 adpcm에서의 등장 빈도가 22.1%로 가장 높다.

표 6. 16비트 분기 명령어의 빈도수
Table 6. Frequency of the 16-bit branch instruction

벤치마크	총 실행 사이클 수	분기 명령어 실행 사이클 수	분기 명령어 비율 (%)
jpeg	11,937,548	973,814	8.2
mpeg	57,170,864	4,012,893	7.0
164	9,948,213	910,449	9.2
sha	14,061,131	1,328,701	9.4
adpcm	16,663,431	3,681,360	22.1
pegwit	50,034,843	6,347,532	12.7
mcf	117,378,155	20,328,544	17.3
평균	-	-	12.3

표 7. R7을 사용하는 LDMIA와 STMIA 명령어의 빈도
Table 7. Frequency of the LDMIA and STMIA instructions that use register R7

벤치마크	총 실행 사이클 수	LDMIA		STMIA	
		사이클 수	비율 (%)	사이클 수	비율 (%)
jpeg	11,937,548	99	0.0008	99	0.0008
mpeg	57,170,864	0	0.0000	0	0.0000
164	9,948,213	0	0.0000	0	0.0000
sha	14,061,131	0	0.0000	0	0.0000
adpcm	16,663,431	0	0.0000	0	0.0000
pegwit	50,034,843	0	0.0000	0	0.0000
mcf	117,378,155	2	0.0000	0	0.0000
평균	-	-	0.0001	-	0.0001

표 7은 제시된 기법에서 병렬 분기 명령어를 위한 인코딩 공간을 제공하기 위해 축소된 두 명령어의 실행 빈도를 보여준다. LDMIA와 STMIA 명령어 중 R7을 전송 레지스터로 포함하는 명령어의 동적 실행 사이클을 측정한다. 각 프로그램에 대하여 ARM Realview 컴파일러 툴 체인이 생성한 Thumb-2 이진 코드를 대상으로 ARM Realview 시뮬레이터를 사용하여 총 동적 실행 사이클과 두 명령어의 사이클을 각각 측정한다. R7을 전송 레지스터로 포함하는 두 명령어는 평균적으로 총 실행 사이클의 0.0001%만을 차지한다. Jpeg의 경우 두 명령어는 각각 99번 실행되어 전체 실행 사이클의 0.0008%를 차지한다. Mcf의 경우 LDMIA 명령어가 2번 실행된다. 나머지 벤치마크 프로그램에서는 두 명령어는 전혀 등장하지 않는다.

Thumb-2 명령 집합 대비 PABEX의 압축 효율을 고려해 보자. PABEX에서는 Thumb-2의 16비트 분기 명령어와 다른 16비트 명령어 두 개를 결합하여 하나의 32비트 명령어로 대체하므로 코드 크기가 그대로 유지되며 증가하지 않는다.

V. 결 론

본 연구에서는 프로세서의 성능을 개선하기 위해 분기 명령어와 사용 빈도가 높은 명령어를 하나로 결합하여 병렬 수행하는 병렬 분기 명령어를 제시한다. 제시된 기법을 Thumb-2 명령어 집합 구조의 Cortex-M3에서 실험 결과 제시된 기법은 코드 크기를 증가시키지 않으면서도 평균 8.0%의 성능을 향상시킨다. 제시된 방식은 ARM사의 최신 구조인 Thumb-2 명령어 집합 구조를 채택하는 다양한 프로세서들에 적용 가능하다. 향후에는 제시된 기법을 Cortex-A15와 같은 Thumb-2 기반의 최신 멀티코어 프로세서 등에 적용하고 성능을 평가하여 멀티코어 프로세서에 최적화된 병렬 분기 명령어를 제안하려고 한다.

참고문헌

〔1〕 Advanced RISC Machines Ltd., "ARM Annual Report & Accounts 2012," Advanced RISC Machines Ltd., 2012.

〔2〕 R. Phelan, "Improving ARM Code Density and Performance," Technical report, Advanced RISC Machines Ltd.. June 2003.

〔3〕 S. Segars, K. Clarke, and L. Goudge, "Embedded control problems, Thumb, and the ARM7TDMI," IEEE Micro, Vol. 15, No. 5, pp. 22-30, Oct. 1995.

〔4〕 J. L. Hennessy, and D. A. Patterson, "Computer Architecture - A Quantitative Approach (5. ed.)," Morgan Kaufmann, pp. 148-261, 2011.

〔5〕 J. A. Fisher, P. Faraboschi, and C. Young, "Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools," Elsevier Morgan Kauffman, pp. 45-398, 2005.

〔6〕 D. -H. Kim, "AMEX: Extending Addressing Mode of 16-bit Thumb Instruction Set Architecture," Journal of The Korea Society of Computer and Information, Vol. 17, No. 11, pp. 1-10, 2012.

〔7〕 A. Krishnaswamy and R. Gupta, "Efficient Use of Invisible Registers in Thumb Code," In Proc. of the 38th IEEE/ACM International Symposium on Microarchitecture, pp. 30-42, Nov 2005.

〔8〕 A. Krishnaswamy, and R. Gupta, "Dynamic coalescing for 16-bit instructions," ACM Transaction on Embedded Computing System, Vol. 4, No. 1, pp. 3-37, Feb. 2005.

〔9〕 Y. -J. Kwon, X. Ma, and H. J. Lee, "PARE: instruction set architecture for efficient code size reduction," IEE Electronics Letters, Vol. 35, No. 24, pp. 2098-2099, Nov. 1999.

〔10〕 B. Li, and R. Gupta, "Bit Section Instruction Set Extension of ARM for Embedded Applications," In Proc. of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), pp. 69-78, Grenoble, France, 2002.

〔11〕 H.-H. Chiang, H.-J. Cheng, and Y.-S. Hwang, "Doubling the Number of Registers on ARM Processors," In Proc. of the 16th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-16), pp. 1-8, Feb. 2012.

〔12〕 Hedley Francis, "ARM DSP-Enhanced Extensions," ARM Ltd., 2001.

〔13〕 J. Rokov, and D. Ing, "ARM Architecture and

Multimedia Applications," RIZ-Transmitters Co., 2010.
〔14〕 ARM Ltd. "Introducing NEON™ Development Article," 2009.
〔15〕 J. Goodacre, and A. N. Sloss, "Parallelism and the ARM instruction set architecture," Computer, Vol. 38, No. 7, pp. 42-50, 2005.
〔16〕 D. -H. Kim, "The Compressed Instruction Set Architecture for the OpenRISC Processor," Journal of The Korea Society of Computer and Information, Vol. 17, No. 10, pp. 11-23, 2012.
〔17〕 F. Bellard, "QEMU, a fast and portable dynamic translator," In Proc. of the Int. Conf. on USENIX Annual Technical Conference, Berkeley, CA, USA, pp. 41-41, 2005.
〔18〕 ARM, "Cortex-M3 technical reference manual," http://www.arm.com, 2010.

저 자 소 개

김 대 환
1993: 서울대학교
계산통계학과 이학사.
1995: 서울대학교
전산과학전공 이학석사.
2010: 서울대학교
전기컴퓨터공학부 공학박사
현 재: 수원과학대학교
컴퓨터정보과 교수
관심분야: 컴파일러, 컴퓨터 구조,
임베디드 시스템,
모바일컴퓨팅
Email : kimdh@ssc.ac.kr