http://dx.doi.org/10.3745/KIPSTA.2011.18A.6.265

16비트 명령어 기반 프로세서를 위한 페어 레지스터 할당 알고리즘

이 호 균[†] · 김 선 욱[††] · 한 영 선[†††]

요 약

다양한 영역에서32비트 명령어 기반 마이크로프로세서의 사용이 일반화되고 있지만, 임베디드 시스템 환경에서는 여전히 16비트 명령어 기반 프로세서가 널리 사용되고 있다. 인텔 8086, 80286 및 모토로라 68000, 그리고 에이디칩스의 AE32000과 같은 프로세서들이 그 대표적인 예이다. 그러나, 16비트 명령어들은 32비트 명령어보다 그 크기로 인해 상대적으로 낮은 표현력을 가지고 있어 동일한 기능을 구현하는데 32비트 명령어 기반 프로세서에 비해 많은 명령어를 수행해야 한다는 문제점을 가지고 있다. 실행 명령어 수는 프로세서의 실행 성능과 밀접한 관련을 가지므로 16비트 명령어셋의 표현력을 향상시켜 성능 저하 문제를 해결할 필요성이 있다. 본 논문에서는 기존의 그래프 컬러링 기반 레지스터 할당(Graph-coloring based Register Allocation) 알고리즘을 보완한 페어 레지스터 할당(Pair Register Allocation) 알고리즘을 제안하고, 이를 통한 성능 분석 결과 및 추후 연구 방향을 제시하고자 한다.

키워드 : 레지스터 할당 알고리즘, 마이크로프로세서, 명령어셋 구조

Pair Register Allocation Algorithm for 16-bit Instruction Set Architecture (ISA) Processor

Hokyoon Lee[†] · Seon Wook Kim[††] · Youngsun Han[†††]

ABSTRACT

Even though 32-bit ISA based microprocessors are widely used more and more, 16-bit ISA based processors are still being frequently employed for embedded systems. Intel 8086, 80286, Motorola 68000, and ADChips AE32000 are the representatives of the 16-bit ISA based processors. However, due to less expressiveness of the 16-bit ISA from its narrow bit width, we need to execute more 16-bit instructions for the same implementation compared to 32-bit instructions. Because the number of executed instructions is a very important factor in performance, we have to resolve the problem by improving the expressiveness of the 16-bit ISA. In this paper, we propose a new pair register allocation algorithm to enhance an original graph-coloring based register allocation algorithm. Also, we explain about both the performance result and further research directions.

Keywords : Register Allocation Algorithm, Microprocessor, Instruction Set Architecture(ISA)

1. 서 론

일반가전, 자동차 전자장비, 휴대용 통신 단말기, 그리고 태블릿 PC까지 임베디드 시스템 시장이 급격히 확대되고 다기능화되면서 해당 시스템들이 채용하는 마이크로프로세서가 다양화, 고성능화 되는 추세이다. 이런 흐름에 따라 오늘날의 임베디드 시스템에서는 32 비트 명령어 기반의 프로세서의 사용이 주류를 이루고 일부 제품에 한해서는 64비트 명령어 기반 프로세서 혹은 멀티 코어 프로세서를 사용하는 경우도 발생하고 있다. 하지만, 저사양 임베디드 시스템에는 여전히 8 또는16 비트 명령어 기반의 프로세서가 널리 사용되고 있는 상황이다.

인텔 8086[1], 80286[2] 및 모토로라 68000[3], 그리고 에이디칩스 AE32000[4]에서 사용된 16비트 명령어들은 32 비트 명령어들과 비교하여 프로그램 코드의 크기를 줄일 수 있다는 장점이 있으나 32 비트 명령어들에 비해 작은 명령어 크기로 인해 명령어 표현이 제한된다는 단점이 있다. 예를 들면, 16비트 명령어는 한 명령어에서 사용할 수 있는

※ 본 논문은 2011년도 경일대학교 신임교원정착연구비 지원에 의하여 수행되었음.
† 준 회 원 : 고려대학교 전자전기공학과 석박사통합과정
†† 종신회원 : 고려대학교 전기전자전파공학부 교수
††† 정 회 원 : 경일대학교 전자공학과 전임강사(교신저자)
논문접수 : 2011년 11월 9일
수 정 일 : 1차 2011년 12월 8일
심사완료 : 2011년 12월 8일

범용 레지스터의 개수가 최대 2개로 제한되어 있어 3개 이상의 범용 레지스터를 사용할 수 있는 32비트 명령어에 비해 제한된 표현력을 가진다. 이런 이유로 제한된 표현력을 보상하는 16비트 명령어셋에 특화된 레지스터 할당 알고리즘에 대한 연구의 필요성이 대두된다.

레지스터 할당 알고리즘은 제한된 수의 CPU 레지스터를 프로그램의 변수에 할당하는 알고리즘을 의미하며 레지스터 할당 기능은 컴파일러의 핵심 기능 중 하나이다. 대표적인 레지스터 할당 알고리즘에는 선형 스캔 방식[5]과 그래프 컬러링 방식[6]이 있다.

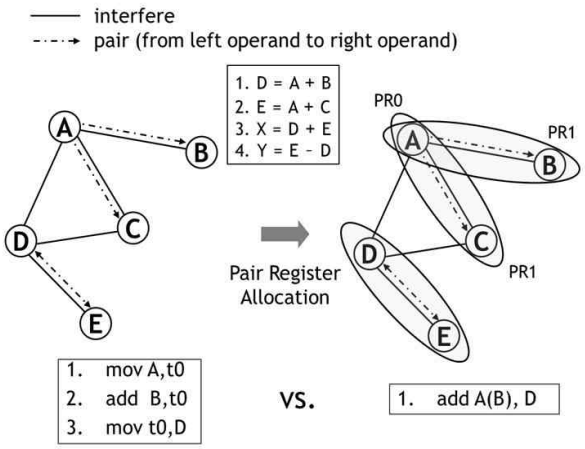
본 논문에서는 그래프 컬러링 기반의 레지스터 할당 방식[6]을 확장하여 16비트 명령어셋을 이용하여 3개 이상의 범용 레지스터를 지정할 수 있는 페어 레지스터 할당 알고리즘(Pair Register Allocation)을 제안하고, 임베디드 시스템의 성능 평가에 사용되는 EEMBC 벤치마크 프로그램[7]을 이용하여 해당 알고리즘의 성능을 분석하고 향후 연구 방향을 제시하고자 한다.

논문의 구성은 다음과 같다. 2장에서는 그래프 컬러링 기반 레지스터 할당 알고리즘에 대해 간단히 설명하고, 이를 확장한 페어 레지스터 알고리즘을 자세히 다룬다. 3장에서는 실제 GNU C 컴파일러 환경[8]에 페어 레지스터 알고리즘을 적용하는 과정을 설명한다. 4 장에서는 해당 알고리즘의 성능을 분석하고, 5장에서는 해당 연구의 향후 진행 방향에 대해 간략하게 설명한다. 마지막으로 6장에서 결론을 맺는다.

2. 페어 레지스터 할당 알고리즘 (Pair Register Allocation Algorithm)

본 장에서는 페어 레지스터 할당 알고리즘의 기본 개념을 소개하고 기존 그래프 컬러링 기반 레지스터 할당 알고리즘[6]으로부터의 확장에 대해 설명한다.

2.1 기본 개념

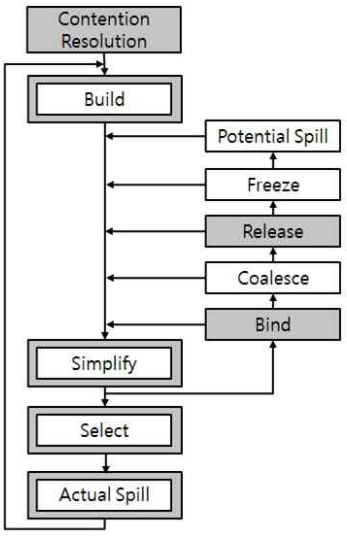


(그림 1) 페어 레지스터 할당 기본 개념도

(그림 1)은 페어 레지스터 할당의 기본 개념도를 보여준다. 해당 그림은 A, B, C, D, E의 다섯 프로그램 변수에 페어 레지스터 할당 알고리즘을 통해 물리 레지스터 PR0와 PR1을 할당하여 최적화하는 것을 보여준다. (그림 1)의 왼쪽 그래프는 그래프 컬러링 기반 레지스터 할당 알고리즘에서 사용되는 간섭 그래프(Interference Graph)를 보여준다. 직선으로 표현된 간선은 두 변수의 간섭 여부를 나타낸다. 특정 시점에서 두 변수 모두가 물리 레지스터 할당을 요구하여 동일한 물리 레지스터를 사용할 수 없을 때 해당 두 변수는 간섭한다고 표현한다. 이와 함께 본 논문에서 새롭게 제안한 페어 레지스터 알고리즘 적용을 위해 화살표가 있는 점선으로 페어 관계를 나타내도록 확장되었다. 페어 관계는 그림 중앙에 위치한 프로그램 코드 1번 D = A + B의 변수 A와 B 사이의 관계를 의미한다. 또한, 프로그램 코드 2번은 (A, C), 코드 3, 4번은 (D, E)가 페어 관계가 있다. 다시 말해, 정적 단일 할당(SSA: Static Single Assignment) 형식[9]으로 표현된 명령어의 두 피연산자 사이의 관계를 나타낸다. 페어 레지스터 할당 알고리즘을 통해 이들 두 페어 변수에 짝으로 묶인 물리 레지스터를 할당한다. 이를 통해 화살표의 시작점에 할당된 하나의 물리 레지스터 주소 PR0 만으로도 짝으로 묶인 두 개의 물리 레지스터PR0와 PR1 모두에 접근 할 수 있어 해당 16비트 기반 명령어셋의 표현력을 향상시킨다. 결과적으로 기존 레지스터 할당 알고리즘 사용시 프로그램 코드 1번 D = A + B을 표현하기 위해 사용되던 세 개의 명령어를 하나로 줄이는 최적화가 가능하다. 해당 페어 레지스터 할당 알고리즘은 모든 16비트 명령어 기반 프로세서 환경에 적용 가능하다.

2.2 알고리즘 확장

(그림 2)는 일반적인 그래프 컬러링 기반 레지스터 할당 알고리즘[6]을 확장한 페어 레지스터 할당 알고리즘의 흐름도를 보여준다. 회색 사각형은 새로 추가된 단계를 의미하

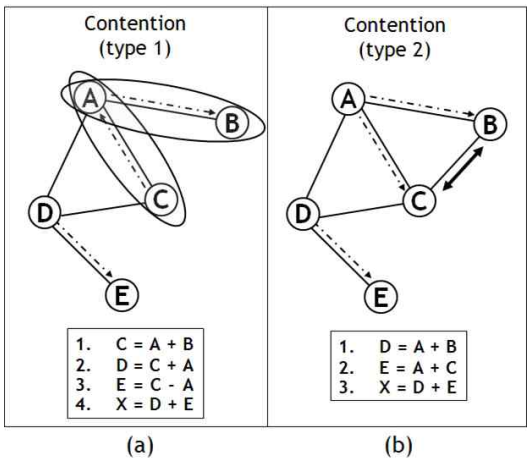


(그림 2) 페어 레지스터 할당 알고리즘 흐름도

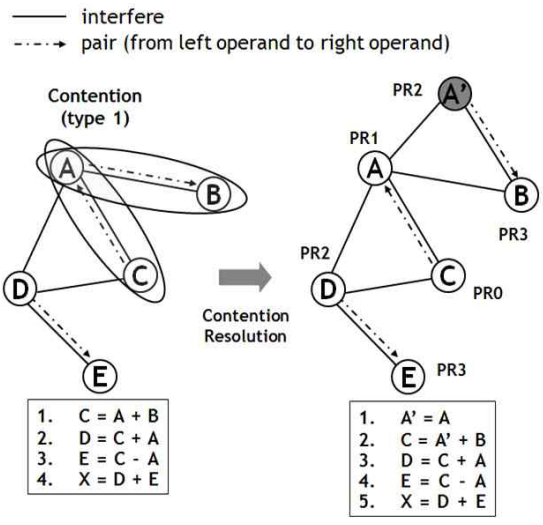
며, 회색 사각형 안의 흰색 사각형은 기존 단계를 확장한 것을 의미한다. 마지막으로 흰색 사각형은 변경이 없는 기존 단계를 나타낸다.

2.2.1 경쟁 해결 (Contention Resolution)

(그림 3)은 페어 레지스터 할당 시 발생 가능한 두 종류의 경쟁(Contention) 상황을 보여준다. 첫 번째 경쟁 상황은 (그림 3(a))의 간섭 그래프에서 프로그램 변수 A가 서로 다른 두 페어 (C->A)와 (A->B)에 속하면서 발생한다. 2. 1절에서 설명된 것처럼 페어 관계는 방향성을 가지므로 변수 A에 동일한 물리 레지스터를 할당할 수 없게 되며, 두 페어 사이에 경쟁 관계가 형성되어 페어 레지스터 할당을 수행할 없게 된다. (그림 3(b))가 보여주는 두 번째 경쟁 상황은 두 개의 서로 다른 페어 (A->B)와 (A->C) 사이에서 발생한다. 이 경우는 첫 번째 경쟁 상황과 달리 프로그램 변수 B와 C 사이에 존재하는 간섭(Interference) 관계로 인해 변수 B와 C에 동일한 페어 물리 레지스터를 할당할 수 없어 발생하는 경쟁 상황이다.



(그림 3) 페어 레지스터 할당 시 경쟁 상황



(그림 4) 경쟁 상황 1의 해결 과정

본 논문에서는 프로그램 변수의 생존 시간 분리(Lifetime splitting) 기법[10]을 통해 해당 경쟁 상황을 제거하여 페어 레지스터를 할당 할 수 있는 경쟁 해결 기법을 제시한다. (그림 4)는 생존 시간 분리를 통해 경쟁 상황 1을 해결하는 방법을 표현한 것이다. 프로그램 코드 1번 A' = A를 추가하여 변수 C, A, B 사이의 연속적인 페어 관계(C->A->B)를 독립적인 두 페어 (C->A)와 (A'->B)로 분리하여 경쟁 관계를 제거한다. 경쟁 상황 2도 이와 같은 방법으로 해결 가능하다. 이와 같은 경쟁 해결 기법을 위해 추가된 하나의 이동(Move) 명령은 페어 레지스터 할당을 통해 얻어진 명령어 하나의 이득과 동일하므로 추가적인 불이익은 발생하지 않는다.

2.2.2 묶음 (Bind)

묶음(Bind) 단계는 간섭 그래프(Interference Graph) 상에서 페어 관계에 있는 서로 다른 두 프로그램 변수의 차수(Degree)가 모두 K-1 보다 작거나 같으면 이들에게 페어 레지스터를 할당할 수 있다고 판단하여 단순화(Simplify) 하는 과정을 말한다. 단, 이때 K는 할당 가능한 물리 레지스터의 개수이다.

2.2.3 해제 (Release)

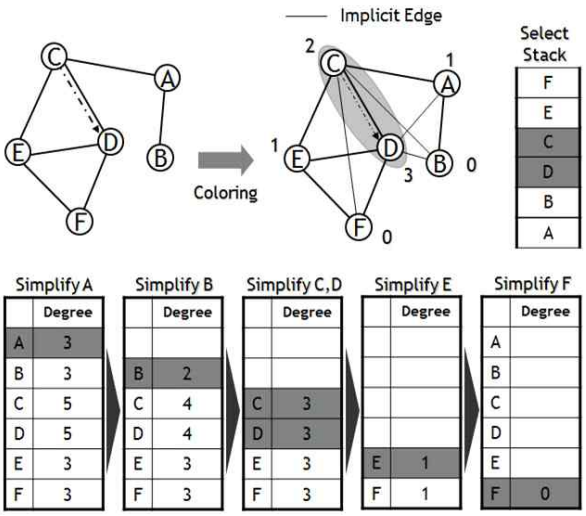
해제(Release) 단계는 페어 관계에 있는 두 프로그램 변수가 묶음(Bind) 단계를 수행할 수 없을 때, 해당 두 변수 사이의 페어 관계를 제거하여 개별적으로 단순화(Simplify) 되도록 하는 과정이다. 이 단계를 통해 불필요한 페어 레지스터 할당으로 인한 물리 레지스터 부족 문제를 회피할 수 있다.

2.2.4 Build/Simplify/Select/Actual Spill

페어 레지스터 할당을 위해 새로 추가된 경쟁 해결(Contention Resolution), 묶음(Bind), 해제(Release) 단계와 달리 다음의 네 단계 Build, Simplify, Select, 그리고 Actual Spill 단계는 기존 그래프 컬러링 기반 레지스터 할당 알고리즘에서 차용하여 페어 레지스터 할당에 맞게 확장하였다. 이와 더불어 Potential Spill, Freeze, 그리고 Coalesce 단계는 수정 없이 적용 가능 하다[6].

2.3 알고리즘 적용 예

(그림 5)는 할당 가능한 물리 레지스터의 개수(K)가 4일 때, 페어 레지스터 할당 알고리즘을 적용하여 물리 레지스터를 할당한 예를 보여준다. 그림 상단 왼쪽의 그래프는 간단한 간섭 그래프(Interference Graph) 예를 보여준다. 그림의 상단 오른쪽의 그래프는 총 5차례의 단순화 과정을 거쳐 물리 레지스터를 할당한 결과를 보여준다. 변수 C, D에는 물리 레지스터 2, 3이 할당되고 나머지 변수들에는 물리 레지스터 0, 1이 할당되었다. 특히, 세 번째 단순화 과정은 페어 관계인 변수 C, D에 짝으로 묶인 물리 레지스터 2, 3을 할당하는 묶음(Bind) 단계를 보여준다. 페어 관계가



(그림 5) 페어 레지스터 할당 예 (K=4일 때)

없는 나머지 변수에는 차수가 K-1 보다 작거나 같을 때 단순화되는 일반적인 단순화 과정이 적용된다. 이때, 내포된 간선(Implicit Edge)은 페어 관계에 속한 변수 C, D와 실질적인 간섭 관계가 없는 변수들 사이에 가상으로 추가된 간섭 관계를 의미한다. 이를 통해 페어 관계에 있는 변수 C, D는 변수 A, B, E, F와는 다른 물리 레지스터를 할당 받게 된다.

3. 구 현

본 장에서는 페어 레지스터 할당 알고리즘의 구현에 대해 설명한다. 페어 레지스터 할당 알고리즘은 실제 구현 방식에 따라 프로그램 함수 시작 시 특정한 물리 레지스터를 페어 레지스터로 특성화시켜 사용하는 전역 지정(Global Specification) 방식과 몇 개의 명령어에만 물리 레지스터를 정하여 사용하는 지역 지정(Local Specification) 방식으로 나뉜다.

3.1 전역 지정 방식

(그림 6)은 프로그램 함수 시작 시에 pair 명령어를 이용하여 특정 물리 레지스터를 페어 레지스터로 지정하여 사용한 예를 보여준다. 해당 예에서 보듯이, pair %r4, %r9 명령어를 통해 물리 레지스터 r4와 r9이 함수 내에서 페어 관계임을 표시한다. 이와 같이 한 함수 내에서 일부 물리 레지스터를 특성화시켜 페어 레지스터 할당에 사용함으로써 이후에 추가적인 지정 과정이 불필요하다는 장점이 있다. 반면에 물리 레지스터를 특성화 시켜 사용함으로써 페어 관계에 있지 않은 프로그램 변수들에 대해서는 할당 가능한 전체 물리 레지스터의 개수가 줄어드는 것과 같은 부작용이 발생하게 된다. 이로 인해 전역 지정 방식 적용 시에는 물리 레지스터를 할당할 수 없는 변수들이 과도하게 발생하여 오히려 전체 프로그램이 성능이 악화되는 문제를 발생시킨다.

(그림 6) 전역 지정 방식 예

(그림 7) 지역 지정 방식 예

3.2 지역 지정 방식

(그림 7)에서 보듯이 지역 지정 방식은 전역 지정 방식과 달리 페어 레지스터를 채용한 명령어가 실행되기 직전에 pset (pair set) 명령어를 사용하여 페어 레지스터를 설정하는 방식이다. 함수 시작 부분에서 지정된 페어 레지스터들이 함수 종료까지 전역으로 유지되는 전역 지정 방식과 달리 지역 지정 방식에서는 함수 실행 도중에 pset 명령어를 이용하여 페어 레지스터 설정을 변경할 수 있다. (그림 7)의 예에서 보듯이 첫 번째 pset 명령어에 의해 물리 레지스터 r4와 r9가 페어 레지스터로 설정되고, 두 번째 pset 명령어에 의해 물리 레지스터 r9 대신 r4와 r5가 페어 레지스터로 설정된다. 페어 레지스터 설정이 지역적으로 변경되는 것이다. 이런 이유로 전역 지정 방식에서처럼 특정 물리 레지스터가 페어 레지스터로 특성화되어 페어 관계에 있지 않은 프로그램 변수에 할당 가능한 전체 물리 레지스터가 부족해지는 부작용을 제거하였다.

그러나, 지역 지정 방식의 경우 페어 레지스터를 채용한 명령어 마다 pset 명령어가 추가 되기 때문에, 전체 실행 명령어 수는 페어 레지스터의 명령어의 수만큼 증가한다. 그렇기 때문에 본 논문에서는 프로세서 구조의 간단한 수정을 통해 명령어 프리페치(Instruction Prefetch) [11] 기법을 적용함으로써 이와 같은 오버헤드를 없애는 방법을 제안한다. pset 명령어 프리페치 기법은 페어 레지스터를 채용한 명령어의 피연산자 페어 정보를 미리 알 수 있게 하여, pset 명령어로 인한 추가 명령어 실행을 숨길 수 있다.

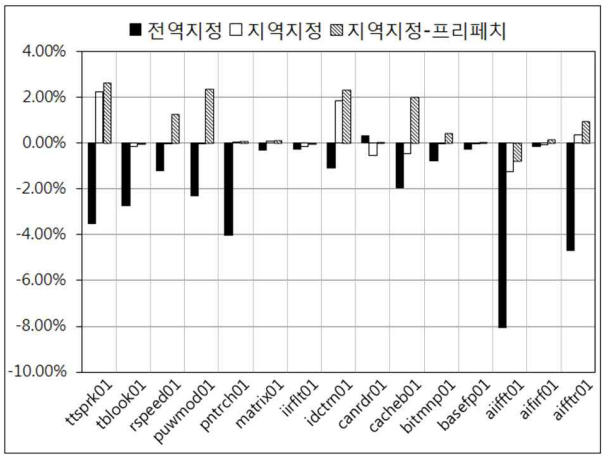
4. 성능 분석

본 장에서는 페어 레지스터 할당 알고리즘의 성능 분석을 위한 실험 환경과 분석 결과에 대해 설명한다.

4.1 실험 환경

본 페어 레지스터 알고리즘은 EISC 구조의 16비트 명령어 기반 프로세서인 AE32000 환경[4]에 적용됐다. 해당 실험 환경에서 사용된 컴파일러는 GCC 3.4.5[8]를 기반으로 하며 -fnew-ra 옵션을 통해 선형 스캔 방식이 아닌 그래프 컬러링 기반 레지스터 할당 알고리즘을 사용하도록 설정되었으며, 페어 레지스터 할당 알고리즘 지원을 위해 확장되었다. AE32000 프로세서[4]는 명령어 페치 단계와 디코드 단계 사이에 프리페치 된 명령어가 저장 되어 있는 8-엔트리의 명령어 큐(instruction queue)가 있으며, pset 명령어를 디코드 단계 이전에 판단 할 수 있도록 확장되었다.

또한 페어 레지스터 할당 알고리즘 적용을 통한 성능 향상은 EEMBC벤치마크 프로그램[7]을 사용하여 측정한 실행 명령어 수의 변화를 기반으로 평가하였다.



(그림 8) 페어 레지스터 지정 방식에 따른 실행 명령어 수의 감소 비율

4.2 실험 결과

(그림 8)은 전역 지정 방식과 지역 지정 방식, 그리고 마지막으로 명령어 프리페치 [11] 기법을 통해 지역 지정 방식에서 pset 명령어로 인한 성능 저하를 제거한 결과를 기존 그래프 컬러링 기반 레지스터 할당 방식의 전체 실행 명령어 수와 비교하여 감소되는 비율을 표현하였다. 그림에서 보듯이 전역 지정 방식의 경우, 할당 가능한 물리 레지스터 수 감소로 인한 부작용 때문에 오히려 전체적으로 실행 명령어 수가 증가하는 문제가 발생하였다. 이를 해결하기 위해 제안된 지역 지정 방식을 적용한 결과, 전역 지정 방식에 비해 평균 약 2.8%의 성능 향상이 있음을 확인할 수 있다. 이와 더불어 명령어 프리페치 기법을 통해 pset 명령어 수행 지연을 제거함으로써 ttsprk01의 경우 그래프 컬러링 기반 레지스터 할당 알고리즘에 비해 최대 2.3%로의 성능 향상을 얻음을 확인하였다. 이는 페어 레지스터 할당을 통해 mov 명령어가 제거됨으로써 얻어진 성능 향상으로 반복문 내에서 삭제된 mov 명령어의 비중이 크다. 불행하게도 성능 향상이 크지 않은 경우는 대부분 2.2.1절에서 설명한 경쟁 관계로 인해 페어 레지스터 할당을 통해 획득하는 이득이 상쇄되었기 때문이다.

또한, (그림 7)에서 보듯이 지역 지정 방식의 페어 레지스터 할당 알고리즘은 페어 관계에 있는 명령어 앞에 pset 명령어를 추가 하지만, mov 명령어를 제거하기 때문에 컴파일된 실행 파일의 코드 크기(code size)는 거의 변화가 없었다.

5. 향후 연구

본 연구를 수행하는 과정에서 페어 관계에 있는 프로그램 변수와 그렇지 않은 변수 간의 합치(Coalesce)가 원활히 이루어지지 않아 불필요한 이동(Move) 명령어가 발생하는 문제를 확인하였다. 추후에 묶음(Bind) 단계를 수정하여 이들 변수 간의 합치를 가능토록 한다면 성능 향상을 얻을 수 있을 것이다.

6. 결 론

본 논문에서 제안한 페어 레지스터 할당 알고리즘은 16비트 명령어셋의 표현력을 향상시켜 16 비트 명령어 기반 프로세서의 성능 향상을 위해 제안된 것이다. 본 논문에서 소개된 성능 분석 결과를 토대로 해당 알고리즘의 우수성 및 향후 추가 연구의 필요성을 확인할 수 있었다. 해당 알고리즘을 EISC 기반의 16비트 프로세서인 AE32000 환경에 적용하여 코드 크기에 영향을 주지 않으면서도 최대 2.3%의 성능 향상을 얻었다. 이와 더불어 향후 연구 방향 제시를 통해 추가적인 성능 향상 가능성을 보여주었다.

참 고 문 헌

[1] S.P. Morse, W.B. Pohlman and B.W. Ravenel. “The Intel 8086 Microprocessor: a 16-bit Evolution of the 8080”, IEEE Computer, Vol.11, pp.18-27, 1978.

[2] D.A. Patterson. “A performance evaluation of the Intel 80286”, ACM SIGARCH Comput. Archit. News, Vol.10, pp.16-18, 1982.

[3] E. Stritter and T. Gunter. “Microsystems a Microprocessor Architecture for a Changing World: The Motorola 68000”, IEEE Computer, Vol.12, pp.43-52, 1979.

[4] H. Kim, D. Jung, H. Jung, Y. Choi, J. Han, B. Min, and H. Oh. “AE32000B: a Fully Synthesizable 32-Bit Embedded Microprocessor Core”, ETRI Journal, Vol.25, pp.337-344, 2003.

[5] M. Poletto and V. Sarkar. “Linear Scan Register Allocation”. ACM Trans. Program. Lang. Syst., Vol.21, pp.895-913, 1999.

[6] G.J. Chaitin, “Register Allocation and Spilling via Graph Coloring”. In Proc. 1892 SIGPLAN Symp. on Compiler construction, pp.98-105, 1982.

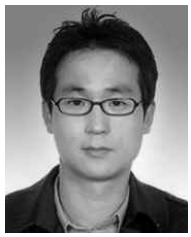
[7] EEMBC. Characterization of the EEMBC Benchmark Suite. http://eembc.org/benchmark/characterization.pdf.
[8] GNU Compiler Collection, http://gcc.gnu.org
[9] M.M. Brandis and H. Mossenbock. “Single-pass generation of static single-assignment form for structured languages”. ACM Trans. Program. Lang. Syst., Vol.16, pp.1684-1698, 1994.
[10] C. Wimmer and H. Mossenbock. “Optimized interval splitting in a linear scan register allocator”. In Proc. 1st ACM/USENIX int. conf. Virtual execution environments, pp.132-141, 2005.
[11] L. Spracklen, Y. Chou, and S. G. Abraham. “Effective Instruction Prefetching in Chip Multiprocessors”. In Proc. 11th Int. Symp. HPCA, pp.225-236, 2005.

이 호 균

e-mail : hokyoon79@korea.ac.kr
2006년 고려대학교 전기전자전파공학부 (학사)
2006년～현 재 고려대학교 전자전기공학과 석박사통합과정
관심분야 : 컴파일러, 임베디드 시스템 등

김 선 욱

e-mail : seon@korea.ac.kr
1988년 고려대학교 전자전산공학과(학사)
1990년 Ohio State Univ. 전기공학과 (공학석사)
2001년 미국 퍼듀대학 전기컴퓨터공학과 (공학박사)
2002년～2005년 고려대학교 전기전자전파공학부 조교수
2005년～2009년 고려대학교 전기전자전파공학부 부교수
2009년～현 재 고려대학교 전기전자전파공학부 교수
관심분야 : 컴파일러, 프로세서 및 SoC 디자인, 병렬처리, 성능평가 등



한 영 선

e-mail : youngsun@kiu.ac.kr
2003년 고려대학교 전기전자전파공학부 (학사)
2009년 고려대학교 전자컴퓨터 공학과 (공학박사)
2009년～2010년 삼성전자 시스템 LSI 책임연구원
2011년～현 재 경일대학교 전자공학과 전임강사
관심분야 : 컴파일러, 임베디드 시스템, 컴퓨터 구조