# H421 코드기반의 더하기 곱셈기법

박지훈*, 김만필*, 최인수**

# An Improved Processor of Multiplication using the Addition based on H421 code

Ji-Hoon Park*, Man-Pil Kim*, In-Soo Choi**

요 약

본 논문은 H421코드를 이용한 연산처리 과정에서 기반 되어지는 덧셈에 의한 곱셈법의 성능향상을 목적으로 한 알고리즘 및 회로 구현을 제시하므로, 유비쿼터스 환경에서 필수적인 도구로 임베디드 모형의 기초화를 예시한다.

ABSTRACT

In this paper, we propose the algorithm and circuit implementation to improve the performance of Multiplication using Addition based on H421 code. We expect that our method will be an essential element to make a embedded prototype in Ubiquitous environment.

Key-word : H421, 8421, Multiplication-Adder

## Ⅰ. 서 론

오늘의 컴퓨터는 VLSC 칩으로 집적된 중앙연산처리 프로세서가 최첨단 Itanium2 프로세서와 같은 행정사무용 정보처리(BDP)에 적합하도록 개선 발달되어져 있으므로, 일반적인 사람들에 컴퓨터의 연산처리 알고리즘과 회로에 대한 10진 수 처리에서의 실시간 과학기술 정보처리의 성능문제에 매우 둔감한 기술적 접근에 처하게 되어 있다. 여기서, 일상생활의 10진 수진법의 개념을 가지면서도 H421코드를 이용한 연산처리 과정에서 기반되어지는 덧셈에 의한 곱셈법의 성능향상을 목적으로 한 알고리즘 및 회로 구현을 제시하므로, 유비쿼터스 환경에서 필수적인 도구로 임베디드 모형의 기초화를 예시한다.

## Ⅱ. 기존 연구

### 2.1 가산기의 종류의 특성

가산기는 캐리(carry)를 전달하는 방법에 따라 직렬 구조, 트리 구조, 하이브리드 구조로 분류한다. 그러나, 이들 구조는 정보의 전달시간과 물리적인 회로면적이 각기 다르므로 하드웨어에 따라 다르게 이용된다.[2]

#### 2.1.1 RCA(Ripple Carry Adder)

Full Adder를 일렬로 연결하여 구성한 덧셈회로이며, 가산기 종류 중 가장 간단한 구조로 많은 비트수를 계산할 때 지연시간이 증가한다. 따라서, 이는 연산에 필요한 시간이 비트수에 선형적으로 증가하므로 고성능 하드웨어에서는 이용할 수가 없다.

* 강원대학교 컴퓨터과학과 박사과정(if93014@kangwon.ac.kr)
** 동원대학 컴퓨터정보과 부교수

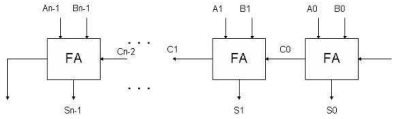


그림 1. RCA
Fig 1. RCA

그림 1은 8비트 데이터(in_a, in_b) 두개와 캐리(in_c)를 입력받아서 두수의 합을 출력(out_s)하고 캐리가 있을 경우의 출력(out_c) 회로도이다.

2.1.2 CSKA(Carry SKip Adder)

CSKA는 특정 블록의 비트 패턴 정보를 이용하여, 그 블록으로 캐리 입력이 인가되었을 경우, 블록내의 비트별 캐리 전파 없이 다음 블록으로 캐리 신호를 전파하는 캐리 전달 방식을 갖는다. 그리고, 이를 위하여 각 블록은 캐리를 전파시킬 수 있는 입력 패턴을 검출하기 위한 부가 회로가 필요하다.

따라서, CSKA는 RCA보다 향상된 가산 처리 속도를 갖고 있으나, 부가 회로의 추가로 인하여 회로면적이 증가한다.

2.1.3 Carry Look-ahead Adder(CLA)

각 전가산기(FA)는 자신의 오른쪽 전가산기에서 자리 올림이 올라올 때까지 기다리기 때문에 연산 장치의 속도를 늦게 하는 요인이 된다.

그러므로, 각 전가산기가 아랫자리에서 올라오는 자리 올림을 위하여 기다리는 시간을 줄여 주면 속도는 더 빨라지게 된다. 이는 자리 올림이 발생할 수 있는지를 미리 예상하여 알려줌으로서 아랫자리에서 자리 올림이 올라올 때까지 기다리지 않고 곧바로 각 전가산기가 동작하도록 하는 자리 올림 예견 가산기(CLA : carry lock-ahead adder)가 설계된다. [1]

이 때 자리 올림이 발생할 수 있는 조건은 가수와 피가수 모두가 1인 경우이다. 가수와 피가수가 모두 1인 경우나 둘 중 하나가 1이고 아랫자리에서 올라오는 자리 올림이 1인 경우에 모두 자리 올림이 발생하고, 아랫자리에서 자리 올림이 발생할 것을 미리 예상하여 독립적으로 윗자리의 각 전가산기에서 자리 올림 1을 전달하고, 각 전가산기는 아랫자리에서 자리 올림이 전해질 때까지 기다릴 필요 없이 독립적으로 동작할 수 있기 때문에 보다 빨리 계산할 수가 있다.

따라서, 입력 비트 수가 많아질수록 fan-in이 많아지고 많은 회로 영역을 필요로 하는 단점을 극복하기 위하여 가산기를 여러 그룹으로 나눈 후, 그 그룹을 더 작은 그룹으로 나누는 다층 레벨로 구현하는 방법이 이용된다.

2.2 가산기를 이용한 곱셈연산

곱셈연산은 가수와 피가수를 입력받아 배수만큼 덧셈 연산을 반복하는 과정이다.

따라서, 가수와 피가수 중 작은 값을 배수로 채택하여 덧셈 연산의 반복 수를 줄여 속도를 향상시키는 방법과 그림 2와 같이 계속적인 시프트와 덧셈 연산을 반복하는 방법을 사용할 수 있다.

여기서, 가수와 피가수는 2차원 배열처럼 결합된 AND 게이트에 전달되어 논리곱이 계산된 후, 그 결과가 전 가산기의 입력으로 사용되도록 한다.

즉, 곱셈에서 결과는 피승수와 승수의 유효 자리를 합한 만큼의 자리가 되므로, 피가수와 가수가 세 자리씩이면 최대 여섯 자리의 결과가 된다.

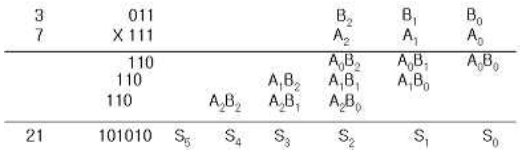


그림 2. 가산기를 이용한 곱셈연산
Fig 2. Multiplicate using the Addition

2.3 코드의 분류

데이터 표현방식으로 크게 가중치(Weighted) 코드, 비가중치(UnWeighted) 코드, 에러검출코드, 자보수 코드로 나누고 있다. 가중치와 비가중치는 각 자리수가 고유값인 가중치 유무에 따라 구별되고 에러검출 코드로는 8421 코드, 3초과 코드, 헤밍 코드 등이 이용된다.

그리고, 어떤 코드의 1의 보수를 취한 값이 10진수의 9의 보수인 자보수 코드라 부르는 3초과 코드, 2421 코드, 51111 코드 등이 있다.

2.3.1 BCD 코드

BCD 또는2진 부호화 10진수 (binary coded decimal)란 4비트의 2진수를 이용하여 1자리의 10진수를 표현한 코드 방식으로 8421코드라고도 하며, 이 코드는 각 비트의 위치에 따라 8,4,2,1의 가중치가 할당된다.

그런데, 표 1와 같이 8421 코드는 각각의 10진 숫자를 4비트의 2진 코드로 표현하는 것이므로, 10진수에서는 숫자를 표현하는 10개의 BCD 코드만 필요하게 된다.

예를 들면, 10진수 369은 다음과 같이 표시된다.

0011 0110 1001

369

BCD 표현 방식의 장점은 수의 크기에 제한이 없다는 것이며, 10진수 1자리를 추가할 경우, 새로이 4-비트만 더 부가하면 된다.

2.3.2 10진 산술장치

사용자는 10진법으로 자료를 입력하고 출력한다. 산술장치 내부에서는 입력된 10진 데이터를 2진수로 변환한 후 연산하고, 다시 10진수로 변환하여 출력한다. 계산과정이 적고 입력 및 출력이 많은 경우에는 10진수 그대로 연산하는 것이 효율적일 수가 있다.

즉, BCD 가산기는 2진수의 덧셈의 규칙에 따라 두 수를 더한 후 연산 결과 4비트의 집합의 값이 9이거나 9보다 작으면 그대로 출력하고 연산 결과 4비트의 집합의 값이 9보다 크거나 자리 올림수가 발생하면 그 결과 값에 $6(0110_2)$을 더한다.

표 1. 8421 Code
Table 1. 8421 Code

| 8 | 4 | 2 | 1 | 10진수 | 16진수 | BCD |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0000 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0001 |
| 0 | 0 | 1 | 0 | 2 | 2 | 0010 |
| 0 | 0 | 1 | 1 | 3 | 3 | 0011 |
| 0 | 1 | 0 | 0 | 4 | 4 | 0100 |
| 0 | 1 | 0 | 1 | 5 | 5 | 0101 |
| 0 | 1 | 1 | 0 | 6 | 6 | 0110 |
| 0 | 1 | 1 | 1 | 7 | 7 | 0111 |
| 1 | 0 | 0 | 0 | 8 | 8 | 1000 |
| 1 | 0 | 0 | 1 | 9 | 9 | 1001 |
| 1 | 0 | 1 | 0 | 10 | A | 1010 |
| 1 | 0 | 1 | 1 | 11 | B | 1011 |
| 1 | 1 | 0 | 0 | 12 | C | 1100 |
| 1 | 1 | 0 | 1 | 13 | D | 1101 |
| 1 | 1 | 1 | 0 | 14 | E | 1110 |
| 1 | 1 | 1 | 1 | 15 | F | 1111 |

예를 들면,

$9(1001_2) + 5(0101_2) = 1110_2$

결과 값이 $9(1001_2) < 1110_2$ 이므로,

$1110_2 + 6(0110_2) = 4(0100_2)$

으로 계산되며, Carry가 발생하여

$14(0001\ 0100_2)$

가 된다.

## Ⅲ. 제안된 곱셈연산

3.1 2421 코드

이 코드는 X421로 정해지었고, 0에서 9까지 표현하기 위해 9-4-2-1=2의 해를 가지고 2421로 표시한 H421의 명칭을 가진다.

즉, 4까지는 8421 코드와 같으나 5부터는 다르다. 이는 처음 수의 자리값(weight)이 8이 아니고 2이기 때문이다. 이 코드에 있어서도 각 자리의 수를 '0'은 '1'로, '1'은 '0'으로 바꿈으로써 9의 보수를 간단히 얻을 수 있다. 이처럼 2진수의 반전에 의해서 보수를 얻을 수 있는 코드를 자보수 코드(self complementing code)라 하고, <표 2>에서와 같이 5이상인 경우 첫 번째 비트가 1이므로 입력된 두수의 합에 대한 Carry 발생여부를 알 수 있

는 장점으로, 이 코드는 주로 전자식 탁상계산기에서 사용된다.

표 2. 2421 code
Table 2. 2421 code

| 10진수 | 2421 코드 |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 1011 |
| 6 | 1100 |
| 7 | 1101 |
| 8 | 1110 |
| 9 | 1111 |

### 3.2 가감산조합법에 의한 곱셈 연산

실 예로, 5 X 7의 경우 피승수인 5에 대한 가산연산을 7번 수행하는 것보다 10의 배수인 50을 만들기 위해 비트 연산을 수행한 후 감산 연산을 3번 수행한다면, 승수만큼 피승수 가산 연산을 수행하는 것보다 10의 배수 처리 후 감산을 수행하는 방법이 효율적이다.

즉, 10의 배수를 만드는 과정은 다음과 같다.

2421 코드는 4비트가 십진수 한자리를 나타내므로 5(1011)의 값에 Shift 4를 취하게 되면 50(1011 0000)이 만들어진다.

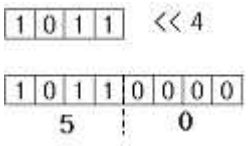


그림 3. Shift
Fig 3. Shift

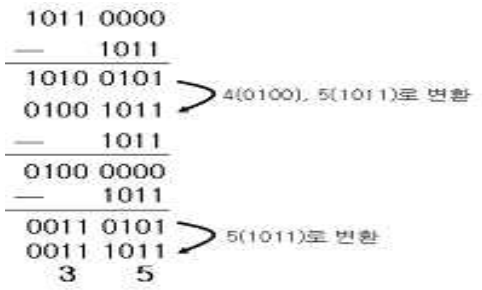


그림 4. 2421 Code 감산연산
Fig 4. The subtraction using 2421 Code

## IV. 실험 및 성능평가

BCD와 2421 코드의 곱셈연산에 대한 가산연산의 처리를 확인하기위하여 어셈블러 코드를 분석하였다. 가산연산인 ADD 명령에 의해 영향을 받는 Flag register 들은 CF(Carry Flag), PF(Parity Flag), AF(Auxiliary carry Flag), ZF(Zero Flag), SF(Sign Flag), OF(Overflow Flag)이다.

[실험 1]에서 가산연산을 수행하는 C로 작성된 프로그램을 수행하면 Compile되어진 후, Deview[5]를 이용하여 그림 5과 같이 어셈블러에서 One Step씩 디버그를 수행하였다.

ADD eax DWORD PTR_j$[ebp]문을 수행하면 하위 8비트만 대상으로 하는 PF만 1로 설정되어 있다. 결과값은 0011 1111B(3FH)가 된다. 하지만 BCD는 0000에서 1001까지 (0~9)의 값을 가지므로 BCD가 아니다. 이를 보정하기위하여 하위 비트에 6(0110)의 가산연산을 수행하여야 한다. 이와 같은 보정작업은 상위 Digit에서도 연속적으로 발생한다.

이와 같은 문제를 해결하기위해 Intel 80x86이나 모토롤라 680x0와 같은 모든 단일 CISC 마이크로프로세서 칩에는 BCD를 취급하는 명령어(DAA, DAS)가 존재한다.

이와는 대비적으로 2421 코드에서는 보정작업의 수행이 불필요하다.

2421 코드e와 BCD에서의 가산 연산을 표현하면 다음과 같다.

$$x^{(i)} = \sum_{j=0}^{i} x_j \qquad (2421\ 코드)$$

$$x^{(i)} = \sum_{j=0}^{i} 2^j x_j \qquad (BCD)$$

$x$는 각 Digit가 된다.

2421 코드는 Digit 수만큼 가산연산을 수행하는 반면에 BCD는 Digit 수의 2의 배수만큼 가산연산을 수행하게 된다.

BCD 감산연산의 경우 보정작업을 필요로 하는 경우는 다음과 같다.

1) SUB 명령어 수행 후 하위 니블이 9보다 크거나 AF=1이면 하위 니블에서 0110을 뺀다.
2) 상위 니블이 9보다 크거나 CF=1이면, 상위 니블에서 0110을 뺀다.

제시된 알고리즘에서는 곱셈연산이므로 BCD 감산연산은 제외되며 2421 코드의 첫 번째 비트를 확인한 후 0이면 가산연산만을 수행하며, 1이면 Shift 연산과 감산을 수행한다.

[실험 1] 가산연산

```
#include <stdio.h>

int main(void)
{
        int i, j;

        i = 17;
        j = 28;

        j = i+j;

        printf("%d", j);
        return 0;
}
```

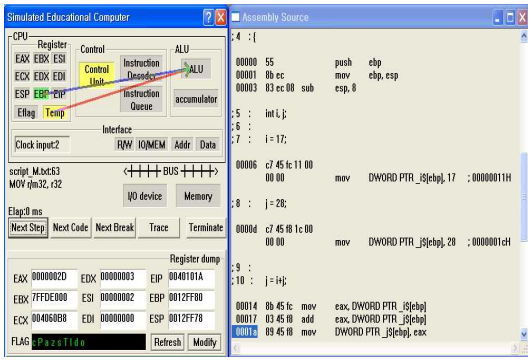


그림 5. Flag Register
Fig 5. Flag Register

제시된 알고리즘[실험 2]의 성능평가 실험 시스템의 하드웨어 사양은 Processor: 2.8GHz, Memory: 512Mbyte, 운영체제는 Linux환경의 Kernel 2.6.15이며, gcc 컴파일러를 사용한 환경에서 2421 code 기반의 가감산조합법에 대한 시뮬레이션의 비교 평가를 기존의 배수만큼 가산연산을 반복하는 방법으로 하였다.

여기서, 32bit 운영체제에서 bit 연산을 시뮬레이션 하기 위해 32bit 자료형의 배열에 저장하고, 공통적으로 사용되어진 함수로 10진수를 입력받는 int input_digit(int digit), 2진수를 생성하는 함수인 int *create_arr(int digit), 그리고 Time 측정을 위해 프로세스의 시작과 끝에 Time Stamp를 사용하여, 이 실행시간을 소요된 Time은 ( (finish - start)/CLOCKS_PER_SEC )으로 측정하였다.

[실험 2] 배수만큼 가산연산 반복
2-1. 가산기
2-1. The Addition

```
int Add( int *array1, int *array2 )
      {
      int CarryFlag;
                  /*캐리발생을 체크 */
                  int Present, temp;
                 /* 현비트의 가산결과,
                      임시저장 변수*/
                      int start, finish;
                 /* Time Stamp */
                 Add_value = (int *)malloc
      (sizeof(int) * 32);
 /* 32bit자료형 배열 생성 */
start = clock();/* Time Stamp */

// 반가산기
CarryFlag = arr1[31] & arr2[31];
Present = arr1[31] ^ arr2[31];
Add_value[31]=Present;
// 전가산기
for(i=maxlength;i>startvalue;i--){ /* 비트별 가산연산 수행 */
Present = array1[i] ^ array2[i] ^
 CarryFlag;
CarryFlag = (array1[i] & array2[i]) |
(array2[i] & CarryFlag) | (CarryFlag &
array1[i]);
Add_value[i] = Present;
}
}

TimeStamp = (double)(finish-start)
/CLOCKS_PER_SEC);
/* 수행 Time 계산 */
}
```

[실험 3] 2421 code 기반의 가감산조합법

3-1. 2421 코드 생성

3-1. Generation of 2421 Code

```
Int *create_2421(int digit){
for(i=0;i<4; i++){
 /* 배열의 초기화 */
temp[i]=0;
}
arr = (int *)malloc(sizeof(int) * 32);
      /* 32bit 자료형 배열 생성 */
switch( digit ){/* /* 2421코드 생성 */
case 0:
       '0000'
        break;
case 1:
       '0001'
        break;
        case 2:
        '0010'
        break;
case 2:
        …
case 9:
        '1111'
        }
        return array;
      }
```

3-2. Shift 연산 및 가감산 조합

3-2. Shift calculation, Combination of The Addition and The Subtraction

```
int ShiftAndSubtract(int count, int *arr1, int times){

for(count; count<maxlength; count++){/* right 4
shift수행 */
arr1[i]=arr1[count];
i++;
}
/* 2421 가중치에 의한 십진수 변환 10자리 계산 */
temp2 = (arr1[index]*2 + arr1[index]*4
+ arr1[index]*2 + arr1[index])*10;

for(i=0;i<times;i++){    /* 감산 반복수행 */
temp2 -= temp1;
}

return temp2;
}
```

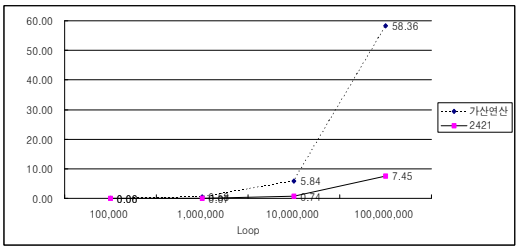


그림 252 실험 측정 결과
Fig 6. The summary of results

그림 6 는 가산연산법을 이용한 곱셈연산 수행방법과 제안된 2421코드를 기반으로 한 가감산조합법의 시간을 측정한 결과이다. 가산연산법의 경우 Loop의 횟수가 증가함에 따라 소요시간이 급격히 증가하는 반면에 2421코드의 가감산조합법의 경우 Loop의 증가에 따라 완만한 곡선을 보이는 것으로 측정되었다.

## V. 결론

유비쿼터스 환경에서의 필수적인 도구로 임베디드 제품은 저 에너지 소비와 단순회로 기능을 바탕 하여야 한다. 따라서, 10진 수 진법의 기반의 논리연산장치(ALU)의 간소화를 구현하도록 H421코드를 이용한 덧셈에 의한 곱셈법의 성능향상을 <그림 5>에서와 같이 실현하였다. 그러므로, 제시된 알고리즘과 회로 구현을 적용할 경우, HW 및 SW적으로 임베디드 제품개발에 기대 된다.

## 참고문헌

[1] 김태환, "캐리-세이브 가산기에기초한 연산 하드웨어 최적화를 위한 실질적 합성 기법", 정보과학회 논문집: 시스템 및 이론 제 28권 제 10호(2001.10)
[2] 이덕영, "정수 선형 프로그래밍을 이용한 혼합 가산기 구조의 최적 설계", 정보과학회 논문집: 시스템 및 이론 제 34권 제 8호(2007.8)
[3] 임윤재, " 휴대폰을 위한 임베디드 리눅스 경량화 기법" 한국컴퓨터종합학술대회 논문집 Vol. 33, No 1(A)
[4] 한주선, " 실시간 시스템의 DMA I/O 요구를 위한 최악 시간 분석", 정보과학회 논문집: 시스템 및 이론 제 32권 제 4호(2005.4)
[5] 성중안, “形式言語 C++ 프로그램 일러스터 : DE-View”, 강원대학교(2004)

## 저자약력

박 지 훈(Ji-Hoon Park)

2000년 강원대학교 사료생산공학과 학사
2003년 강원대학교 컴퓨터과학과 석사
2008년 강원대학교 컴퓨터과학과 박사
<관심분야> 시스템 프로그래밍, 정보보안, Motion Graphic

김 만 필(Man-Pil Kim)

2000년 강원대학교 전자계산학과 학사
2003년 강원대학교 컴퓨터과학과 석사
2007년 강원대학교 컴퓨터과학과 박사 수료
<관심분야> 시스템프로그래밍, 이미지프로세싱, 정보보안, GIS

최 인 수(In-Soo Choi)

2004년 강원대학교 컴퓨터과학과 박사
1987~1992 시스템공학연구소
1996~현재 동원대학 컴퓨터정보과 부교수
<관심분야> 유비쿼터스 컴퓨팅, 정보보안, Motion Graphic