Computer Architecture

Network Computing Lab.

Arm assembly Lab 5

Contents

- **Lab5 : Debugger를 이용한 Matrix 곱셈 프로그램 동작 이해**
  - 5-1 : A[3, 3] x B[3, 3]=C[3,3] 곱셈 코드 이해 및 C[i,j] 계산 과정 trace
  - 5-2 : 임의의 s, u, v에서 A[s, u] x B[u, v]=C[s, v] 계산하도록 기존 코드 확장하기
  - HW9: 실습 A~B 수행후 결과를 캡쳐/설명 추가하여 보고서로 제출하세요. 실습 B에서 작성한 소스코드는 file로 제출하세요.

Lab 5 : matrix 곱셈(c코드-1/4)

Source code 분석

```c
#include <stdio.h>
#include <stdlib.h>

extern void matrix(int (*A)[], int (*B)[], int (*C)[], int index);

int main(){
        int index=3 ;
        int i = 0 ;
        int j = 0 ;
        int k = 0 ;
        int A[3][3] ;
        int B[3][3] ;
        int C[3][3] ;


        printf("This is 3 by 3 matrix version \n");

        printf("\nInput value of A : ");
        for(i=0; i < index; i++){
                for(j=0; j < index; j++){
                        A[i][j] = getchar()-'0';
                }
        }
        j=0; i=0;
        getchar();
```

```
park@ubuntu:~/CA_lab/lab5$ qemu-arm -g 8080 ./lab5
This is 3 by 3 matrix version

Input value of A :
```

행렬 A의 원소들을 입력 받는 코드입니다.
실행을 시키면 위와 같이 터미널 창에 입력하는 메시지가 나타납니다.
아래와 같이 3x3 행렬의 원소 개수만큼 입력해주세요.

```
park@ubuntu:~/CA_lab/lab5$ qemu-arm -g 8080 ./lab5
This is 3 by 3 matrix version

Input value of A : 123123123
```

Lab 5 : matrix 곱셈(c코드-2/4)

Source code 분석

```
27          printf("\nInput value of B : ");
28          for(j=0; j < index; j++){
29                  for(k=0; k < index; k++){
30                          B[j][k] = getchar()-'0';
31                          printf("%d",B[j][k]);
32                  }
33          }
34          k=0; j=0;
35
```

```
park@ubuntu:~/CA_lab/lab5$ qemu-arm -g 8080 ./lab5
This is 3 by 3 matrix version

Input value of A : 123123123

Input value of B :
```

행렬 A의 원소들을 입력시키면 동일하게
행렬 B의 원소들도 입력하라는 메시지가 뜹니다.
동일하게 원소 개수만큼 입력해주세요.

```
park@ubuntu:~/CA_lab/lab5$ qemu-arm -g 8080 ./lab5
This is 3 by 3 matrix version

Input value of A : 123123123

Input value of B : 111222333
```

Lab 5 : matrix 곱셈(c코드-3/4)

Source code 분석

```
35          printf("\nArray A \n");
36          for(i=0; i < index; i++){
37                  for(j=0; j < index; j++){
38                  printf("%d ",A[i][j]);
39                  }
40                  printf("\n");
41          }
42          i=0;
43          j=0;
44
45          printf("Array B \n");
46          for(j=0; j < index; j++){
47                  for(k=0; k < index; k++){
48                  printf("%d ",B[j][k]);
49                  }
50                  printf("\n");
51          }
52          j=0;
53          k=0;
```

```
This is 3 by 3 matrix version

Input value of A : 123123123

Input value of B : 111222333

Array A
1 2 3
1 2 3
1 2 3
Array B
1 1 1
2 2 2
3 3 3
```

입력 받은 행렬 A,B의 원소들을 출력해주는 코드입니다.
위의 터미널 창과 같이 3x3 행렬 형태를 갖추어 출력합니다.

Lab 5 : matrix 곱셈(c코드-4/4)

Source code 분석

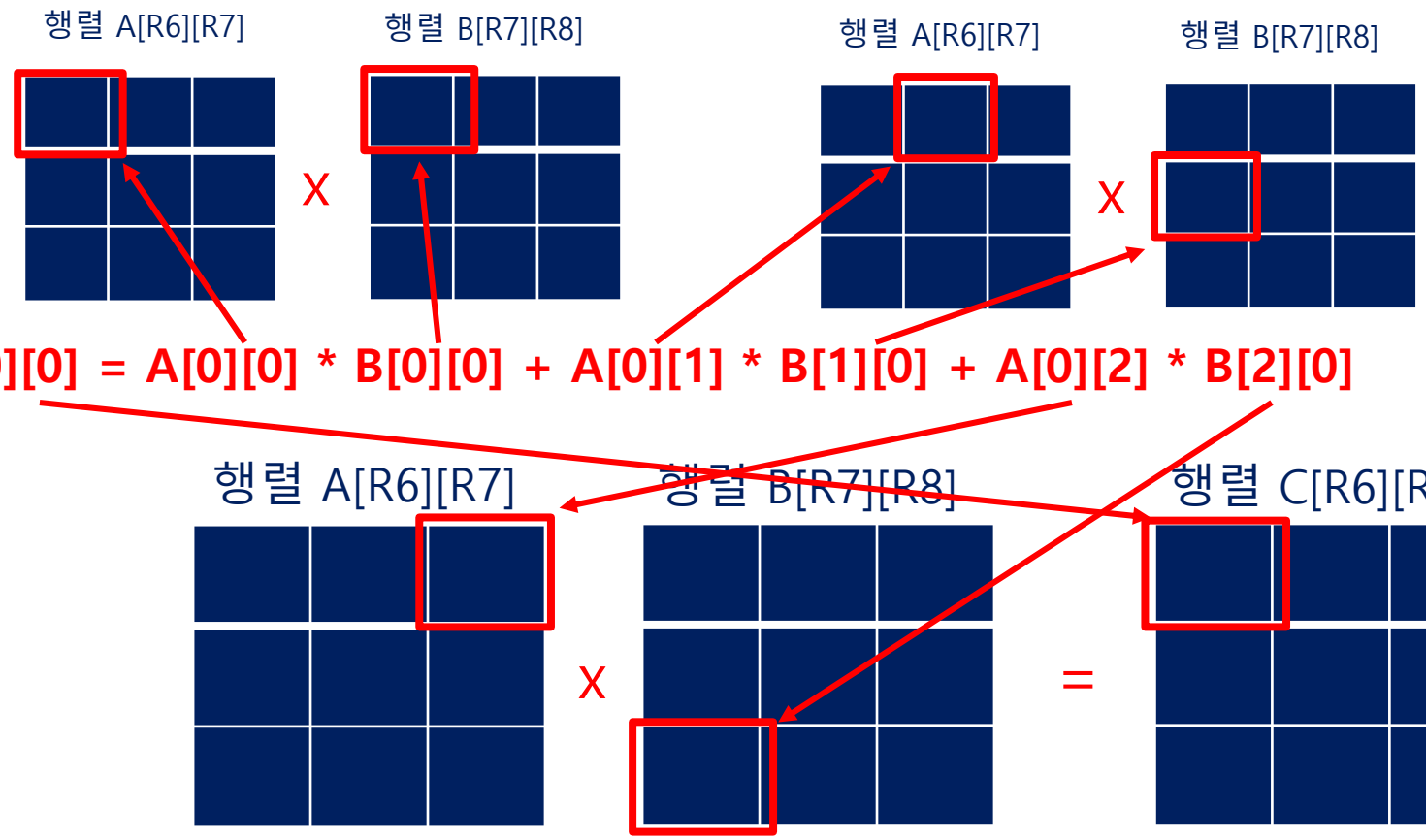
```
55          matrix( A, B, C, index);
56
57          printf("Array C after Operating : \n");
58          for(i=0; i < index; i++){
59                  for(k=0; k < index; k++){
60                  printf("%d ",C[i][k]);
61                  }
62                  printf("\n");
63          }
64
65          return 0;
66 }
```

Matrix함수는 assembly 코드로 구현되어 있습니다. 뒤에서 살펴보도록 하고, 나머지는 오른쪽에 보이는 것과 같이 연산을 한 결과 행렬이 되는 행렬 C를 터미널에 출력해줍니다.

```
This is 3 by 3 matrix version

Input value of A : 123123123

Input value of B : 111222333

Array A
1 2 3
1 2 3
1 2 3
Array B
1 1 1
2 2 2
3 3 3
Array C after Operating :
14 14 14
14 14 14
14 14 14
```

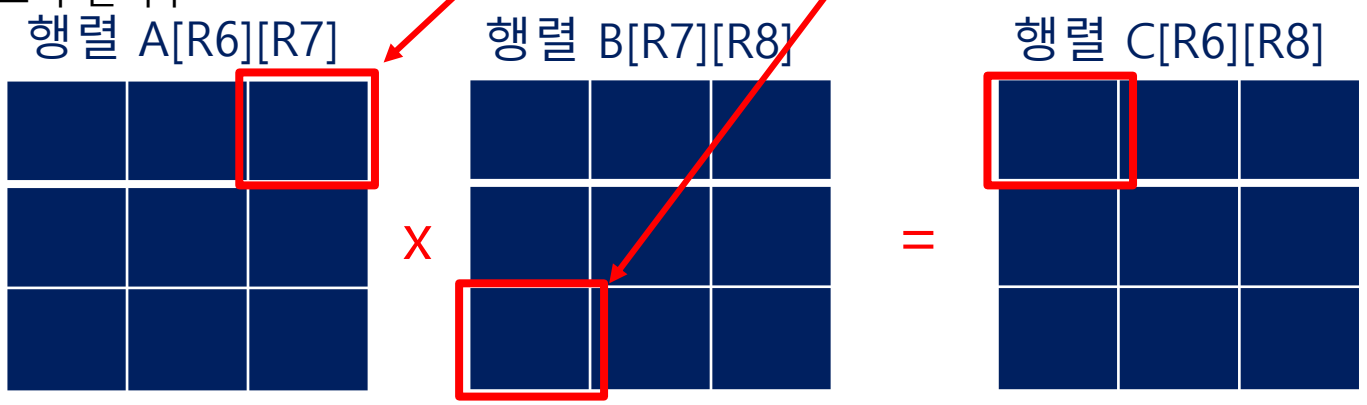
Lab 5 : matrix 곱셈 (행렬계산예제)

- 행렬 계산 예제



Lab 5 : matrix 곱셈(행렬요소의 메모리주소 계산예제)

- **A[0][2]가 저장된 메모리 주소 ?**
  - 행렬 A 시작주소(A[0][0])가 1000 이라면
  - A[0][2]가 저장된 주소는 1008. element 하나당 4번지 차지. A[0][1]이 저장된 주소는 1004
- **B[2][0]이 저장된 메모리 주소 ?**
  - 행렬 B 시작주소(B[0][0])가 2000 이라면
  - B[2][0]가 저장된 주소는 2024. B[1][0]이 저장된 주소는 2012
- **A[i][j]이 저장된 메모리 주소 ?**
  - **행렬 A 시작주소(A[0][0]) + i*3*4 + j*4**. 여기서 3은 행렬의 한 row 크기, 4는 element 당 메모리 번지수



Lab 5 : matrix 곱셈(행렬계산예제)

Network Computing Lab.

- Matrix 연산 설명, 메트릭스 A, B, C는 3 x 3 으로 가정, 메트릭스내 각 element 값은 word 크기 (32 바이트,메모리관점에서 보면 주소 4개 차지)
- C[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0] + A[0][2] * B[2][0]의 일반화 형태
- C[S][V] = A[S][0] * B[0][V] + A[S][1] * B[1][V] + A[S][2] * B[2][V] 계산은 ?

- A[s][u]의 주소 = 배열 A의 시작주소 + s*3*4 + u*4
- B[u][v]의 주소 = 배열 B의 시작주소 + u*3*4 + v*4
- 위 수식에서 4는 한 element가 차지하는 메모리 주소가 4개 차지한다는 것에서 옴

- A[s][0]의 주소를 r11에 저장. r11 주소를 이용하여 값을 r12에 갖고 온다
- B[0][v]의 주소를 r11에 저장. r11 주소를 이용하여 값을 r11에 갖고 온다
- r12 * r11 -> r9에 누적한다. (A[S][0] * B[0][V] 계산)

- A[s][1]의 주소를 r11에 저장. r11 주소를 이용하여 값을 r12에 갖고 온다
- B[1][v]의 주소를 r11에 저장. r11 주소를 이용하여 값을 r11에 갖고 온다
- r12 * r11 -> r9에 누적한다. (A[S][0] * B[0][V] + A[S][1] * B[1][V] 계산)

- A[s][2]의 주소를 r11에 저장. r11 주소를 이용하여 값을 r12에 갖고 온다
- B[2][v]의 주소를 r11에 저장. r11 주소를 이용하여 값을 r11에 갖고 온다
- r12 * r11 -> r9에 누적한다. (A[S][0] * B[0][V] + A[S][1] * B[1][V] + A[S][2] * B[2][V] 계산)

Lab 5 : matrix 곱셈(어셈 – 초기화)

Source code 분석

```
1 .text
2 .global matrix
3
4 matrix:
5         STMFD sp!, {r0-r12,lr}
6
7         MOV     r6,#0
8         MOV     r7,#0
9         MOV     r8,#0
10        MOV     r9,#0
```

Matrix함수는 도입부입니다.

R6 = s
R7 = u
R8 = v
이렇게 행렬의 인덱스 값을 의미합니다.
R9 = 행렬 계산값으로 사용, 0 으로 초기화

Lab 5 : matrix 곱셈(어셈-행렬 A 요소 및 행렬 B 요소 갖고 와서 곱하고 결과값 누적하기)

- **Source code 분석**

```
12 Loop:
13          MOV     r11, #12
14          MUL     r11,r6,r11
15          ADD     r11,r11,r7,LSL #2
16
17          ADD     r11,r11,r0
18
19          LDR     r12,[r11]
20
21          MOV     r11, #12
22          MUL     r11,r7,r11
23          ADD     r11,r11,r8,LSL #2
24          ADD     r11,r11,r1
25
26          LDR     r11,[r11]
27
28          MUL     r12,r11,r12
29          ADD     r9, r9, r12
30
31          ADD     r7,r7,#1
32          CMP     r7,r3
33          BNE     Loop
34          MOV     r7,#0
```

행렬 A의 element 주소 계산
R12에 element 값 저장

행렬 B의 element 주소 계산
R11에 element 값 저장

위에서 구한 두 값을 곱하고 r9에 누적 덧셈
결과적으로 한 element에 대한 연산 값은 r9

R7이 u값이고 r3는 index(현재 3)
행렬 크기만큼 loop

Lab 5 : matrix 곱셈(행렬 C 요소값 저장하기 및 행렬 C 다음 요소값 계산하기)

- **Source code 분석**

- 결과 값 넣을 배열 C의 element 주소 계산

- R8은 v값, r6는 s값이고 index(r3)와 비교하여 loop 수행 cmp결과 not equal이면, 행렬 C의 다음 element 계산하러 감

- 연산 끝까지 다해서 행렬 C를 다 채웠으면 reg복구하고 끝!

```
37          MOV     r11, #12
38          MUL     r11,r6,r11
39          ADD     r11,r11,r8,LSL #2
40
41          ADD     r11,r11,r2
42
43          STR     r9, [r11]
44
45          MOV     r9,#0
46
47          ADD     r8,r8,#1
48          CMP     r8, r3
49          BNE     Loop
50
51          MOV     r8,#0
52
53          ADD     r6,r6,#1
54          CMP     r6,r3
55          BNE     Loop
56
57          LDMFD sp!, {r0-r12,pc}
58
59 .end
```

Lab 5 : matrix 곱셈

- Source code 분석

```
12 Loop:
13          MOV     r11, #12
14          MUL     r11,r6,r11
15          ADD     r11,r11,r7,LSL #2
16
17          ADD     r11,r11,r0
18
19          LDR     r12,[r11]
20
21          MOV     r11, #12
22          MUL     r11,r7,r11
23          ADD     r11,r11,r8,LSL #2
24          ADD     r11,r11,r1
25
26          LDR     r11,[r11]
27
28          MUL     r12,r11,r12
29          ADD     r9, r9, r12
30
31          ADD     r7,r7,#1
32          CMP     r7,r3
33          BNE     Loop
34          MOV     r7,#0
```

```
37          MOV     r11, #12
38          MUL     r11,r6,r11
39          ADD     r11,r11,r8,LSL #2
40
41          ADD     r11,r11,r2
42
43          STR     r9, [r11]
44
45          MOV     r9,#0
46
47          ADD     r8,r8,#1
48          CMP     r8, r3
49          BNE     Loop
50
51          MOV     r8,#0
52
53          ADD     r6,r6,#1
54          CMP     r6,r3
55          BNE     Loop
56
57          LDMFD sp!, {r0-r12,pc}
58
59 .end
```

Lab 5 : matrix 곱셈

Source code 분석

```
12 Loop:
13          MOV     r11, #12
14          MUL     r11,r6,r11
15          ADD     r11,r11,r7,LSL #2
16
17          ADD     r11,r11,r0
18
19          LDR     r12,[r11]
```

행렬 A에 대한 원소의 offset을 계산하는 부분입니다. 계산된 주소 값은 r11에 있고 ldr을 이용해 r12에 원소 값을 임시 저장해 놓습니다.

Lab 5 : matrix 곱셈

- Source code 분석

```
12 Loop:
13          MOV     r11, #12
14          MUL     r11,r6,r11
15          ADD     r11,r11,r7,LSL #2
16
17          ADD     r11,r11,r0
18
19          LDR     r12,[r11]
20
21          MOV     r11, #12
22          MUL     r11,r7,r11
23          ADD     r11,r11,r8,LSL #2
24          ADD     r11,r11,r1
25
26          LDR     r11,[r11]
27
28          MUL     r12,r11,r12
29          ADD     r9, r9, r12
30
31          ADD     r7,r7,#1
32          CMP     r7,r3
33          BNE     Loop
34          MOV     r7,#0
```

```
37          MOV     r11, #12
38          MUL     r11,r6,r11
39          ADD     r11,r11,r8,LSL #2
40
41          ADD     r11,r11,r2
42
43          STR     r9, [r11]
44
45          MOV     r9,#0
46
47          ADD     r8,r8,#1
48          CMP     r8, r3
49          BNE     Loop
50
51          MOV     r8,#0
52
53          ADD     r6,r6,#1
54          CMP     r6,r3
55          BNE     Loop
56
57          LDMFD sp!, {r0-r12,pc}
58
59 .end
```

Lab 5 : matrix 곱셈

Source code 분석

```
21          MOV     r11, #12
22          MUL     r11,r7,r11
23          ADD     r11,r11,r8,LSL #2
24          ADD     r11,r11,r1
25
26          LDR     r11,[r11]
```

행렬 B에 대한 원소의 offset을 계산하는 부분입니다. 계산된 주소 값은 r11에 있고 ldr을 이용해 r11에 원소 값을 임시 저장해 놓습니다.

Lab 5 : matrix 곱셈

- Source code 분석

```
12 Loop:
13          MOV     r11, #12
14          MUL     r11,r6,r11
15          ADD     r11,r11,r7,LSL #2
16
17          ADD     r11,r11,r0
18
19          LDR     r12,[r11]
20
21          MOV     r11, #12
22          MUL     r11,r7,r11
23          ADD     r11,r11,r8,LSL #2
24          ADD     r11,r11,r1
25
26          LDR     r11,[r11]
27
28          MUL     r12,r11,r12
29          ADD     r9, r9, r12
30
31          ADD     r7,r7,#1
32          CMP     r7,r3
33          BNE     Loop
34          MOV     r7,#0
```

```
37          MOV     r11, #12
38          MUL     r11,r6,r11
39          ADD     r11,r11,r8,LSL #2
40
41          ADD     r11,r11,r2
42
43          STR     r9, [r11]
44
45          MOV     r9,#0
46
47          ADD     r8,r8,#1
48          CMP     r8, r3
49          BNE     Loop
50
51          MOV     r8,#0
52
53          ADD     r6,r6,#1
54          CMP     r6,r3
55          BNE     Loop
56
57          LDMFD sp!, {r0-r12,pc}
58
59 .end
```

Lab 5 : matrix 곱셈

Source code 분석

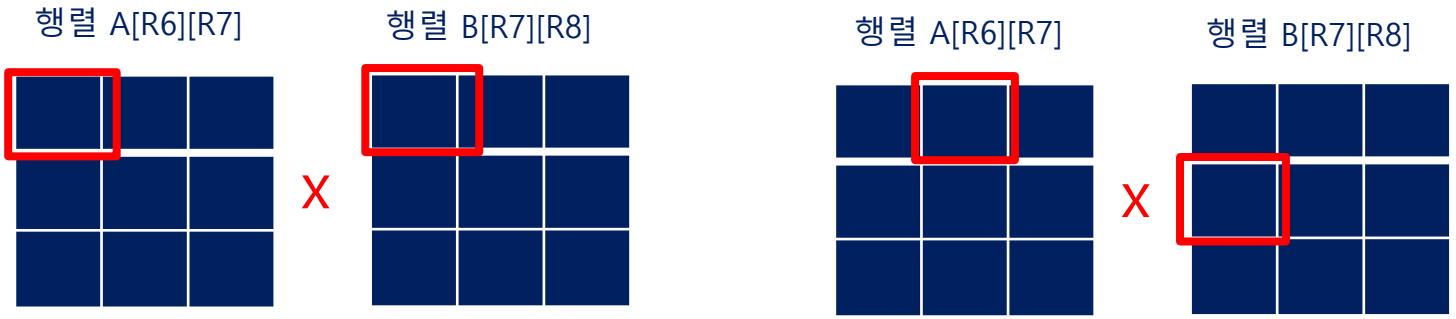
```
28          MUL     r12,r11,r12
29          ADD     r9, r9, r12
30
31          ADD     r7,r7,#1
32          CMP     r7,r3
33          BNE     Loop
34          MOV     r7,#0
```

R12에는 행렬 A의 원소와
R11에는 행렬 B의 원소가 들어가 있고
그것을 곱해서 R9에 저장해 놓습니다.

Lab 5 : matrix 곱셈

Source code 분석

행렬 C의 하나의 원소가 계산이 완료될 때까지 loop를 반복합니다. 그 값은 R9에 축적됩니다.



Lab 5-1 : matrix 곱셈 operation trace

- **실습 A : 아래 행렬 C의 빨간색 박스로 표시된 원소의 계산 과정의 일부를 trace 합니다.**
- 위 소스 코드 18 라인에 break point 주고, R6(s) = 1, R7(u) = 0, R8(v) = 2 가 되는 시점부터 trace합니다. 아래 그림상에서 1번값과 2번값이 곱해져 3번에 저장되는 과정에서, 1번 ~ 3번에 해당하는 메모리 번지 값(R11), 메모리 내용 값(1번 및 2번 읽어온 값, 3번에 저장한 값)을 캡처하시오.
- 캡쳐결과가 어느 것에 해당하는 지 표시하세요.

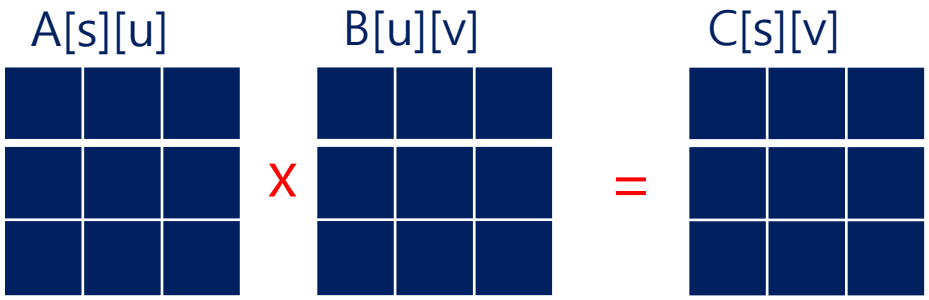
Lab 5-2 : 다른 크기의 두 matrix 곱셈 하기

- 실습 B: C 프로그램상에서 다른 크기의 matrix A[s,u], B[u,v]에서 s, u, v를 입력받고 malloc을 이용하여 matrix A, B, C[s,v]를 위한 배열 메모리를 할당한다. 두 matrix A, B의 각 원소값을 입력받아 각각 초기화한다. 매트릭스 인덱스 s, u, v 값이 저장된 3 x 1 array 1개 D를 선언한다. C 프로그램에서 Matrix A, B, C에 대한 pointer 값 3개, 매트릭스 D의 시작번지를 argument로 하여 어셈블리 함수를 호출한다. 어셈블리 코드에서는 matrix 곱셈을 수행하여 결과를 Matrix C에 저장한다. (P22, 23, 24, 27 내용대로 구현하세요)
- s, u, v 는 1보다 크고 9보다 작은 수를 사용.
- 어셈블리 코드에 주석을 추가하세요. 결과 화면을 캡처하세요.
- C 및 어셈블리 소스 코드를 파일로 제출하세요.

Lab 5 : 다른 크기의 두 matrix 곱셈하기

- matrix의 크기 s, u, v를 입력 받는 방법
- Int type 3개 크기의 array를 선언하고 각각에 s, u, v 값을 키보드 입력으로 받는다.
- 아래 예시 코드처럼 진행해보자.

```
int Array_size[3];
int idx;

printf("input values of s,u,v : ");

while( idx < 3){
        Array_size[idx] = getchar()-'0';
        idx++;
}
```



Lab 5 : 다른 크기의 두 matrix 곱셈하기

- 각 matrix 값 저장을 위한 이차원 배열은 malloc을 이용하여 동적으로 메모리를 할당받는다. 자세한 방법은 다음 페이지에서 설명함.
- 할당받은 이차원 배열 초기화를 위한 코드 및 matrix 곱셈후 최종 결과 출력은 C 코드로 작성하되 matrix element 참조 또는 지정시 포인터를 사용한다. 자세한 방법은 다다음 페이지에서 설명함.

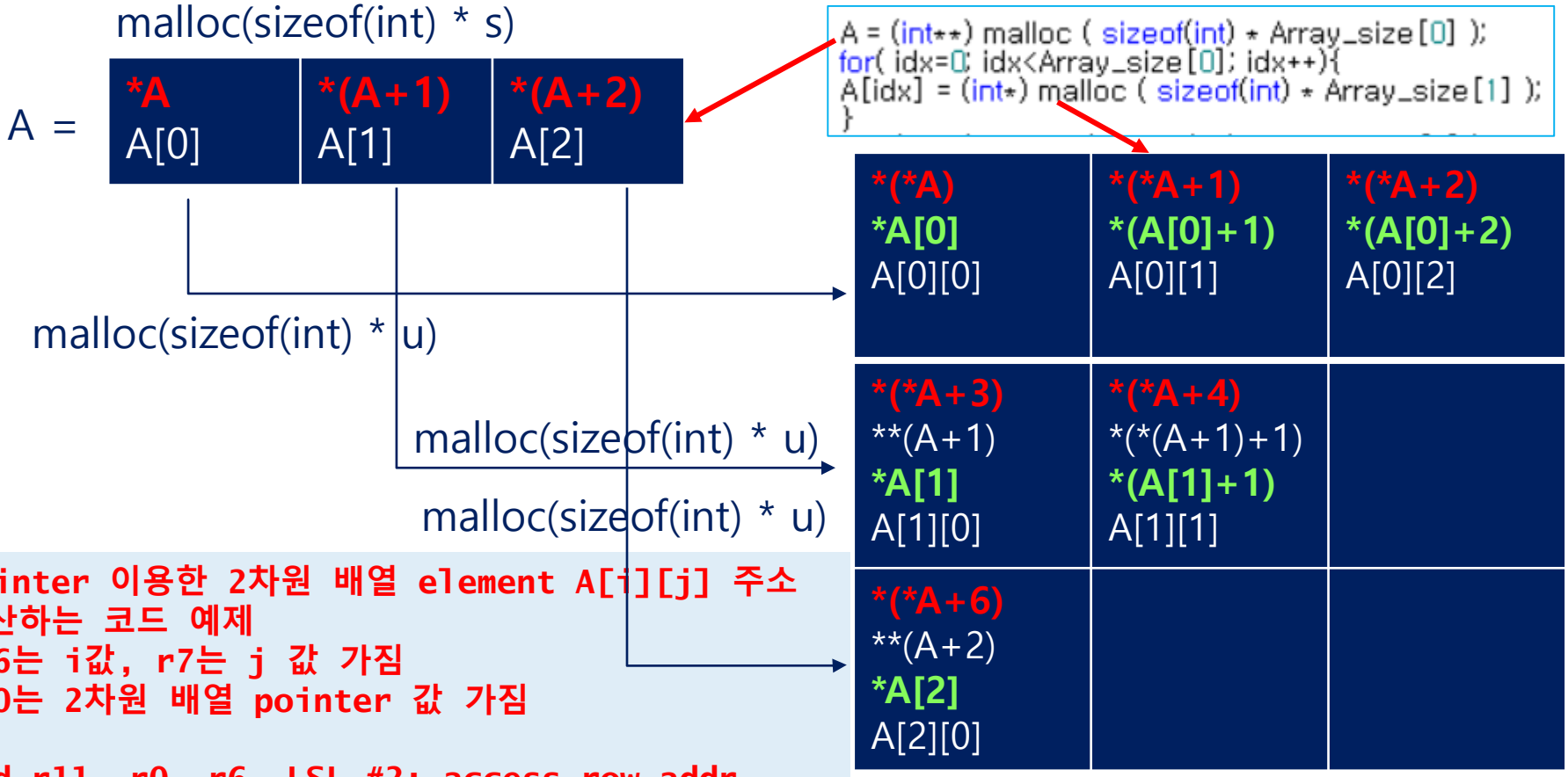
Lab 5 : 다른 크기의 두 matrix 곱셈하기

- 동적으로 입력 받은 matrix의 크기에 알맞게 malloc 함수를 사용하여 array에 메모리 공간을 할당
- malloc을 사용하기 위한 standard library를 포함한다. (#include <stdlib.h>)
- Int type 2차원 포인터를 선언한다. (int ** A)
- 아래 예시 코드는 A[s,u]를 위한 배열 메모리를 malloc 함수를 이용하여 동적으로 할당해준다. Array_size[0]은 s, Array_size[1]은 u에 해당됨. 같은 방법으로 B[u,v], C[s,v]도 메모리 할당해야 함.

```
A = (int**) malloc ( sizeof(int) * Array_size[0] );
for( idx=0; idx<Array_size[0]; idx++){
A[idx] = (int*) malloc ( sizeof(int) * Array_size[1] );
}
```

Lab 5 : 다른 크기의 두 matrix 곱셈하기

- 포인터를 이용한 2차원 배열 접근 방법(예: A[3][3])




```
Pointer 이용한 2차원 배열 element A[i][j] 주소 계산하는 코드 예제
-r6는 i값, r7는 j 값 가짐
-r0는 2차원 배열 pointer 값 가짐

add r11, r0, r6, LSL #2; access row addr
ldr r11, [r11]
add r11, r11, r7, LSL #2; access col addr
```

Lab 5 : 다른 크기의 두 matrix 곱셈하기

- **C 코드에서 A[3][3] 할당하고 어셈블리 코드에서 이차원 배열 시작주소 A를 R0로 받아왔다고 가정하자.**
- **어셈블리코드에서 A[i] [j] 값을 갖고 오기**
  - A[0][0]는 M[R0] – R0를 주소로 하여 메모리에서 읽어온다
  - A[0][1]는 M[R0+4] – R0+4 를 주소로 하여 메모리에서 읽어온다
  - A[0][2]는 M[R0+8]
  - A[1][0]는 M[R0+12]
  - A[1][1]는 M[R0+16]
  - A[1][2]는 M[R0+20]
  - A[2][0]는 M[R0+24] – R0+24 를 주소로 하여 메모리에서 읽어온다

Lab 5 : 다른 크기의 두 matrix 곱셈하기

- **Test 를 아래와 같이 수행하여 결과를 제출한다.**
- **값을 입력받아 Matrix 초기화**
  - Matrix 첫 원소 A[0][0]에 1을 넣고 element 증가할 때마다 1씩 증가한 값을 넣는다. A[0][1]=2, A[0][2]=3, ….
  - (예) A[3,3]={1,2,3,4, ….9}, A[3,4]={1,2,3,4, ….12}
- **s, u, v 값을 전부 3으로 입력하고, Matrix 배열 메모리를 할당받고, 위와 같이 초기화 한후 Matrix 곱셈 수행 후 결과 출력하기.**
- **s=3, u=4, v=5로 입력하고, Matrix 배열 메모리를 할당받고, 위와 같이 초기화 한후 Matrix 곱셈 수행후 결과 출력하기.**