

# 어셈블리프로그램 설계및실습 보고서

## 5차 실습과제

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화요일 6, 7

학 번: 2019202050

성 명: 이강현

제 출 일: 2022.10.31(월)

## 1. Problem Statement

레지스터와 메모리간에 블록단위로 데이터를 이동하는 것에 대한 이해를 기른다. 어셈블리에서 서브루틴의 사용법을 익힌다. 어셈블리에서 스택의 필요성과 스택이 어떻게 사용되고 실행되는지 관련 예제를 통해 이해하고 직접 활용해본다.

## 2. Design

### <problem1>

첫번째 문제는 10!의 값을 스택과 재귀함수를 활용하여 구현하는 것이다.

따라서 곱할 값 r1과 결과값을 저장할 r2의 초기값을 1로 세팅해두고 sp(stack pointer)에는 40000의 주소값을 저장해두고 시작한다.

그 이후 factorial이라는 함수로 서브루틴을 진행하기위해 이동하고 해당 함수내에서 r1을 1씩 증가시켜가며 r2에 곱해준다. 탈출조건은 r1이 10과 같아질때로 설정해두었기 때문에 함수가 함수내에서 탈출조건을 만족하기 전까지는 계속 함수를 호출한다. 결국 r1이 10과 같아졌을 때는 r2에 10!값이 저장되어 있게 되고 Endline이라는 함수로 이동하게 된다. 이 함수에서는 sp값에 해당하는 메모리에 r2의 값을 저장하고 프로그램을 종료한다.

### <problem2>

두번째 문제는 레지스터의 값을 서로 바꾸는데 스택 혹은 블록복사 명령어를 활용하여 구현하는 것이다. 우선 초기세팅은 r0에 0, r1에 1, r2에 2.....

r7에 7을 넣어둔 상태로 세팅해두었다. 그 후 r0부터 순차적으로 저장되어야하는 레지스터값을 메모리에 저장하였고 블록단위가 필요하다면 활용하였다. 그 후 메모리에 있는 값들을 블록단위로 한번에 LDMFA명령어를 통해 가져와 r0부터 r7까지의 레지스터에 값들을 넣어주고 프로그램을 종료한다.

### <problem3>

세번째 문제는 r0에는 10, r1에는 11..... r7에는 17이 저장되어 있는 초기세팅을 주고 이를 함수를 호출하여 값들을 변경하고 최종값을 메모리에 저장하는 흐름으로 진행된다. 우선 레지스터에 문제에서 제공된 값들을 저장한 후 doRegister함수를 BL명령어로 실행한다. doRegister함수는 각각의 register 값과 register의 인덱스 값을 더해 다시 레지스터에 저장하고 모든 레지스터의 값들을 더한 후 그 값을 return하는 함수이다. 따라서 ADD명령어를 통해 구현하였고 값을 구한 후

MOV pc,lr 명령을 통해 BL명령어를 사용한 바로 다음줄로 이동한다. 그 후 B 명령어를 통해 doGCD함수를 실행하는데 해당 함수는 doRegister에서 return된 값과 160의 최대공약수를 구하는 함수이다. 최대공약수를 구하는 법은 유클리드 호제법을 활용하였고 구한 최대공약수에 r4레지스터의 초기값을 더해주고 메모리에 저장해야하므로 메모리에 카피해둔 레지스터값들을 스택포인터를 활용한 블록단위명령어를 통해 레지스터에 저장한 후 r4를 최대공약수와 더해준다. 이를 sp값에 저장하고 프로그램을 종료한다.

### 3. Conclusion

#### <problem1>

Current	R0	0x00000000
	R1	0x00000003
	R2	0x00000006
	R3	0x00000000
	R4	0x00000000

Current	R0	0x00000000
	R1	0x00000004
	R2	0x00000018
	R3	0x00000000
	R4	0x00000000
	R5	0x00000000

Register	Value
Current	
R0	0x00000000
R1	0x00000005
R2	0x00000078
R3	0x00000000
R4	0x00000000
R5	0x00000000

Register	Value
Current	
R0	0x00000000
R1	0x0000000A
R2	0x00375F00
R3	0x00000000
R4	0x00000000
R5	0x00000000

왼쪽 위부터 오른쪽순서로 r1이 1씩 증가해서 r2에 곱해지고 있고 최종적으로 r1이 10이 되었을 때 아래와 같이 메모리에 저장되는 것을 확인할 수 있다.

Memory 1	Internal
Address: 0x40000	PC \$ 0x00000028
0x00040000: 00 5F 37 00 00 00 00	Mode Supervisor
0x00040009: 00 00 00 00 00 00 00	States 81
0x00040012: 00 00 00 00 00 00 00	Sec 0.00000000

코드의 state 수는 81이다.

Register	Value
<b>Current</b>	
R0	0x00000000
R1	0x00000001
R2	0x00000002
R3	0x00000003
R4	0x00000004
R5	0x00000005
R6	0x00000006
<b>R7</b>	<b>0x00000007</b>
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00040000
R14 (LR)	0x00000000
<b>R15 (PC)</b>	<b>0x00000024</b>
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	

## 초기세팅

메모리에 문제에서

The screenshot displays the Keil MDK-ARM IDE interface. On the left, the **Registers** window shows the current state of the processor registers. The **Current** tab is active, listing registers R0 through R15, CPSR, and SPSR. Register R15 (PC) is highlighted in blue, showing its value as 0x00000040. Below the registers, the **Supervisor** section is expanded, showing options like User/System, Fast Interrupt, Interrupt, Abort, Undefined, and Internal. The **Internal** section is also expanded, showing the PC register with a value of 0x00000040.

On the right, the **Disassembly** window shows the assembly code for the selected memory address. The code is organized into three sections: **problem1.s**, **problem2.s**, and **problem3.s**. The **problem2.s** section is currently selected, showing the following instructions:

```

2      ENTRY
3      start
4      LDR sp,tempaddr
5      MOV r0,#0
6      MOV r1,#1
7      MOV r2,#2
8      MOV r3,#3
9      MOV r4,#4
10     MOV r5,#5
11     MOV r6,#6
12     MOV r7,#7
13
14     STR r2,[sp],#4
15     STR r0,[sp],#4
16     STR r3,[sp],#4
17     STMIA sp!,{r5-r7} ;block da
18     STR r1,[sp],#4
19     STR r4,[sp]
20     LDMFA sp,{r0-r7} ;block dat
21     MOV pc,#0
22

```

```

Internal
  PC $      0x00000040
  Mode      Supervisor
  States     35
  Sec        0.00000000

```

state수는 35이다.

### <problem3>

The screenshot shows the ARM Disassembler interface. On the left, the 'Registers' window displays the current state of registers R0 through R15, CPSR, and SPSR. R15 (PC) is highlighted with the value 0x00000028. On the right, the 'Disassembly' window shows the assembly code for 'problem3.s'. The code includes instructions like MOV, BL, and B, with comments explaining their functions. The 'doRegister' function is also visible, showing a loop that increments registers R0 through R7 by 1.

<초기세팅>

The screenshot shows the 'Memory' window with the address 0x40000. The memory contents are displayed in hexadecimal, showing a sequence of zeros and some non-zero values (0A, 0B, 0C, 0D, 0E, 0F) at specific addresses, indicating the initial state of memory before the program execution.

doRegister함수에서 레지스터값을 변경하기 전 미리 메모리에 저장해둠

Register	Value
<b>Current</b>	
R0	0x0000000A
R1	0x0000000C
R2	0x0000000E
R3	0x00000010
R4	0x00000012
R5	0x00000014
R6	0x00000016
R7	0x00000018

<인덱스 값을 각각 더해준 모습>

Register	Value
<b>Current</b>	
R0	0x0000000A
R1	0x0000000C
R2	0x0000000E
R3	0x00000010
R4	0x00000012
R5	0x00000014
R6	0x00000016
R7	0x00000018
R8	0x00000088

<이후 모든 레지스터의 값들을 더해 r8에 저장>

R10	0x00000000	13	MOV r7,#17
R11	0x00000000	14	BL doRegister ;branch with link
R12	0x00000000	15	B doGCD ;branch
R13 (SP)	0x00040000	16	
R14 (LR)	0x0000002C	17	doRegister
R15 (PC)	0x0000002C	18	STMFA sp,{r0-r7} ;copy register data
CPSR	0x000000D3	19	ADD r0,r0,#0
SPSR	0x00000000	20	ADD r1,r1,#1
User/System		21	ADD r2,r2,#2
Fast Interrupt		22	ADD r3,r3,#3
Interrupt		23	ADD r4,r4,#4
Supervisor		24	ADD r5,r5,#5
Abort		25	ADD r6,r6,#6
Undefined		26	ADD r7,r7,#7 ;register = register + 1
Internal		27	
PC \$	0x0000002C	28	ADD r8,r8,r0
Mode	Supervisor	29	ADD r8,r8,r1
States	43	30	ADD r8,r8,r2
Sec	0.00000000	31	ADD r8,r8,r3

BL명령어로 저장해둔 곳에 doRegister함수를 마치고 돌아온 모습

R8	0x00000008	35	ADD r8,r8,r7 ;r8: register's sum
R9	0x00000008	36	MOV pc,lr; end subroutine
R10	0x00000000	37	
R11	0x00000000	38	doGCD
R12	0x00000000	39	CMP r8,r9
R13 (SP)	0x00040000	40	SUBGT r8,r8,r9 ;if r8>r9,r8=r8-r9
R14 (LR)	0x0000002C	41	SUBLT r9,r9,r8 ;if r8<r9,r9=r9-r8
R15 (PC)	0x00000088	42	
CPSR	0x600000D3	43	BNE doGCD ; not equal? loop
SPSR	0x00000000	44	BEQ Endline ; equal? go endline

R8,R9이 서로 값이 같아지자 doGCD함수를 탈출하는 모습

Register	Value	46:	LDMED sp,{r0-r7}
R0	0x0000000A	47:	ADD r8,r8,r4
R1	0x0000000B	48:	STR r8,tempaddr2 ; save
R2	0x0000000C		
R3	0x0000000D		
R4	0x0000000E		
R5	0x0000000F		
R6	0x00000010		
R7	0x00000011		
R8	0x00000008	32	ADD r8,r8,r4
R9	0x00000008	33	ADD r8,r8,r5
R10	0x00000000	34	ADD r8,r8,r6
R11	0x00000000	35	ADD r8,r8,r7 ;r8: register's sum
R12	0x00000000	36	MOV pc,lr; end subroutine
R13 (SP)	0x00040000	37	
R14 (LR)	0x0000002C	38	doGCD
R15 (PC)	0x00000090	39	CMP r8,r9
CPSR	0x600000D3	40	SUBGT r8,r8,r9 ;if r8>r9,r8=r8-r9
SPSR	0x00000000	41	SUBLT r9,r9,r8 ;if r8<r9,r9=r9-r8
User/System		42	
Fast Interrupt		43	BNE doGCD ; not equal? loop
Interrupt		44	BEQ Endline ; equal? go endline
Supervisor		45	Endline
Abort		46	LDMED sp,{r0-r7}
Undefined		47	ADD r8,r8,r4
		48	STR r8,tempaddr2 ; save
		49	MOV r8,#0

초기 r4값을 연산한 결과와 더해주기 위해 메모리에서 블록단위로 가져오는 모습

R7	0x00000011	Memory 1
R8	0x00000016	
R9	0x00000008	
		Address: 0x40000
		0x00040000: 16 00 00 00
		0x00040017: 00 0F 00 00

R8에 저장하고 STMFA를 통해 값을 저장하고 LDMED를 통해 값을 가져왔으므로 40000번지는 비어있고 sp또한 40000번지를 가리키고 있으므로 sp가 가리키는

곳에 0x16을 저장한다.

```
Internal
  PC $ 0x00000000
  Mode Supervisor
  States 118
  Sec 0.00000000
```

state수는 118이 나왔다.

#### 4. Consideration

이번 실습에서는 스택과 블록단위 데이터 이동, 재귀함수의 사용 등 다양한 개념을 학습하였다. FA,FD,EA,ED,IA,IB,DA,DB가 블록단위명령어 뒤에서 어떻게 작동하는지를 코드를 구현해보며 쉽게 이해할 수 있었다. 또한 서브루틴을 직접 구현해보면서 어셈블리 언어도 다른 high level 언어와 같이 함수를 구현할 수 있다는 것을 깨달았다. 또한 state수를 통해 블록단위 명령어,서브루틴등의 작업이 어느 정도의 state를 요구하는지를 알 수 있었다.

또한 BL명령어를 사용해보면서 일반적으로 사용해본 B명령어와는 다르게 다시 순차적으로 실행하던 부분으로 돌아갈 수 있다는 점이 앞으로 코드를 구현할 때 더 유용하게 사용할 수 있을 것 같다는 생각이 들었다.

#### 5. Reference

이형근/어셈블리프로그램 설계 및 실습/광운대학교(컴퓨터정보공학부)/2022