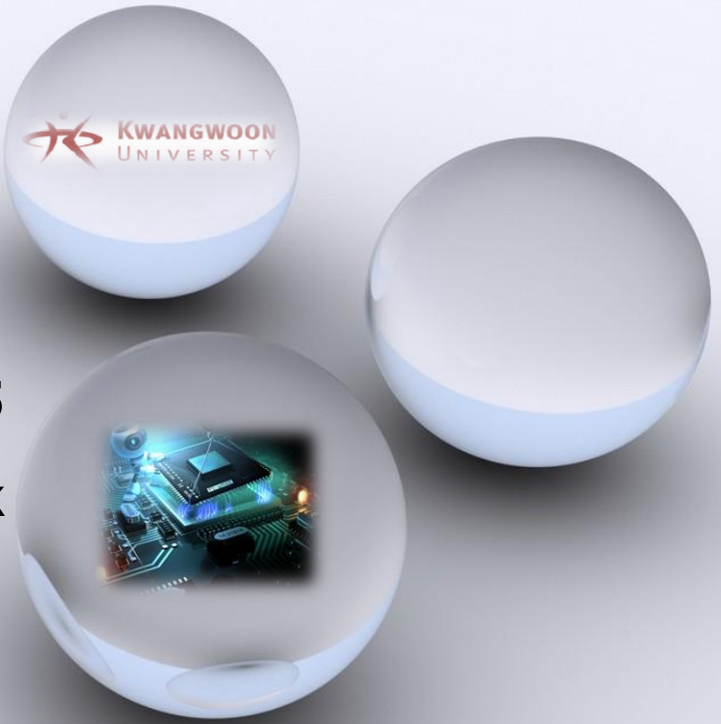
어셈블리프로그램 설계 및 실습



Lab #5

Block Data Transfer & Stack

Kwangwoon University
Embedded System Architecture Lab

학습 목표

- 레지스터와 메모리간 블록 단위의 데이터 저장/가져오기를 이해한다.
- 어셈블리어의 서브 루틴(함수) 사용 방법에 대해 익힌다.
- 프로그래밍 수준에서 스택 명령어의 필요성과 효용성에 대해 이해하고 응용한다.
- 여러 종류의 스택 저장 방식(stack type)에 대해 이해하고 각 방식에 따른 메모리 접근 방법을 실질적으로 확인해봄으로써 그 차이를 이해한다.

Contents

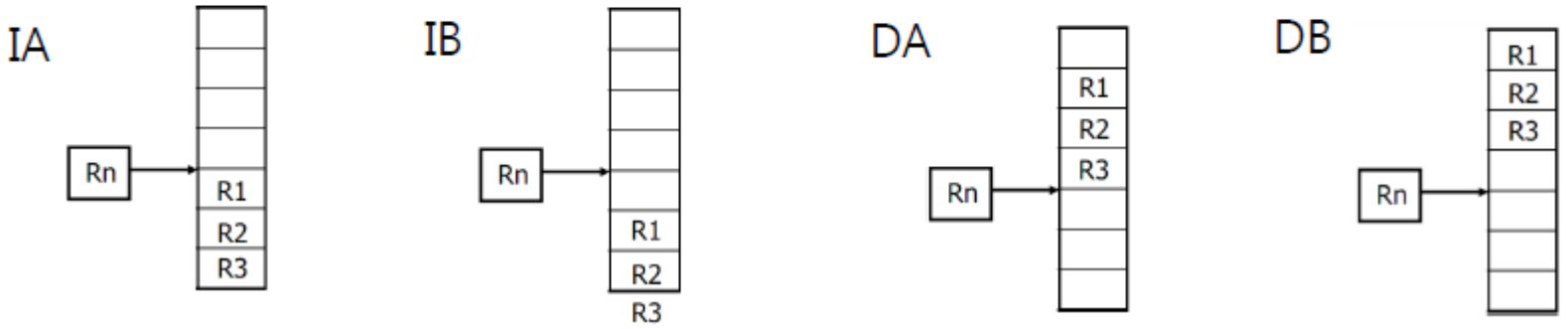
- Multiple load/store register
  - LDM/STM & addressing mode
- Stack mode
  - STM/LDM mode
- Flow control instructions
  - B group
- Problem
  - Block data transfer
  - Stack & subroutine

Multiple Load/Store Register(1/2)

- Instructions
  - LDM: Load multiple registers
  - STM: Store multiple registers

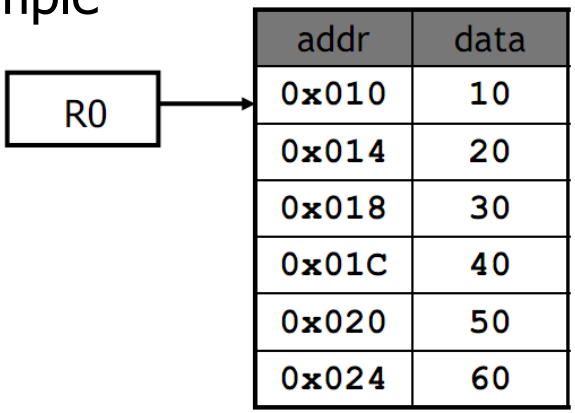
```
Syntax: <LDM|STM>{<cond>}<addressing mode> Rn{!},<registers>{^}
```

Addressing mode	Description	Start address	End address	*Rn*!
IA	increment after	*Rn*	*Rn* + 4*N* − 4	*Rn* + 4*N*
IB	increment before	*Rn* + 4	*Rn* + 4*N*	*Rn* + 4*N*
DA	decrement after	*Rn* − 4*N* + 4	*Rn*	*Rn* − 4*N*
DB	decrement before	*Rn* − 4*N*	*Rn* − 4	*Rn* − 4*N*



Multiple Load/Store Register(2/2)

- Example



NOW: R0=0x10, R1=R2=R3 = 0

- LDMIA R0, {R1, R2, R3}
- or
- LDMIA R0, {R1-R3}

→ R1: 10 R2: 20 R3: 30 R0: 0x10

- LDMIA R0!, {R1, R2, R3}
- or
- LDMIA R0!, {R1-R3}

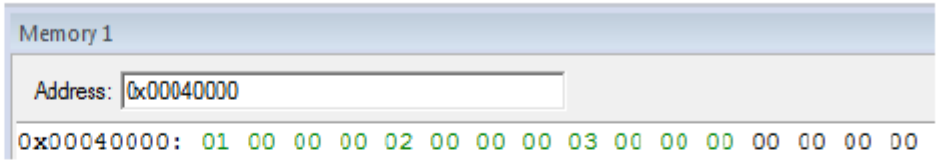
→ R1: 10 R2: 20 R3: 30 R0: 0x1C

Simple example

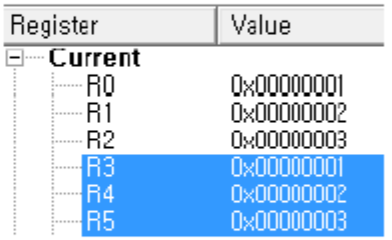
- Copy values from R0-R2 to R3-R5
  - Code size: 40 bytes, states: 19

```
    AREA    blockData, CODE, READONLY     ; Name this block of code
    ENTRY                  ; Mark first instruction to execute

start
    MOV r0, #1
    MOV r1, #2
    MOV r2, #3

    LDR     r6, TEMPADDR1   ;set start address

①  STMIA   r6!, {r0}       ; store r0 on mem
   STMIA   r6!, {r1-r2}    ; store r1-r2 on mem

②  LDR     r6, TEMPADDR1   ;set start address
   LDMIA   r6, {r3-r5}     ; load from r0-r2 to r3-r5
                           ; consequentially, block copy is executed
    MOV    pc, #0    ;end

TEMPADDR1   &  &00040000 ;stack start address
    END
```

①



②

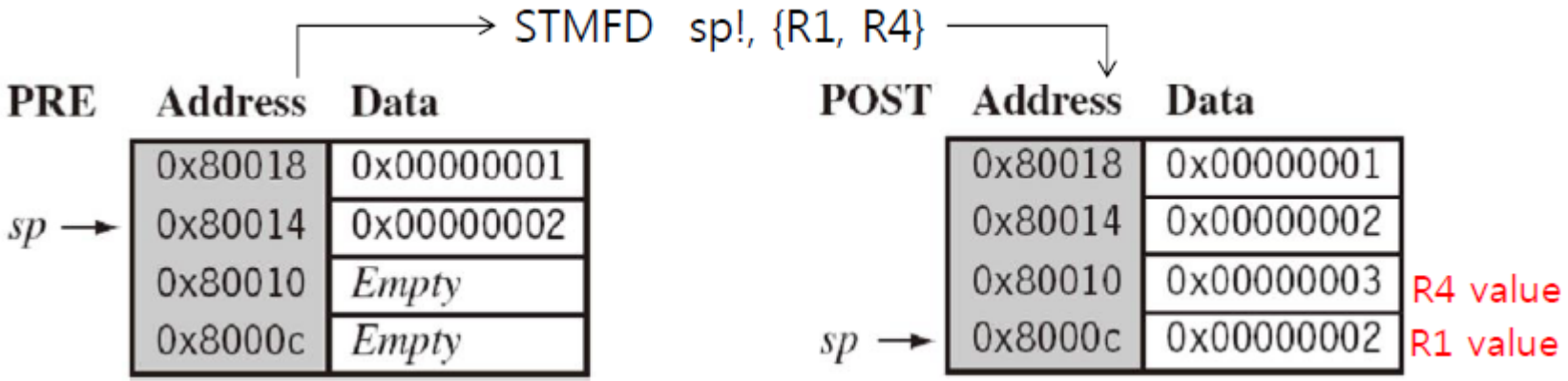


Let's correct this code(①~②) for higher performance! → *32byte / 15states*

Stack mode

- Stack

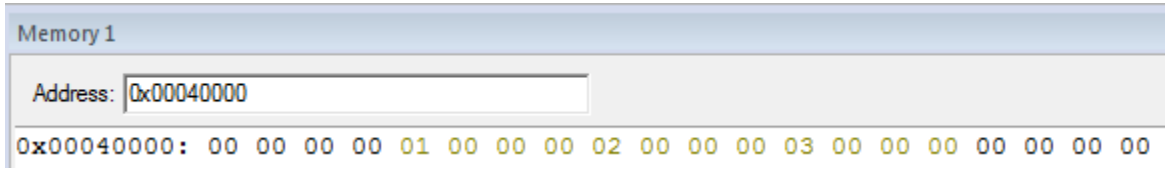
mode	POP	=LDM	PUSH	=STM
Full ascending (FA)	LDMFA	LDMDA	STMFA	STMIB
Full descending (FD)	LDMFD	LDMIA	STMFD	STMDB
Empty ascending (EA)	LDMEA	LDMDB	STMEA	STMIA
Empty descending (ED)	LDMED	LDMIB	STMED	STMDA



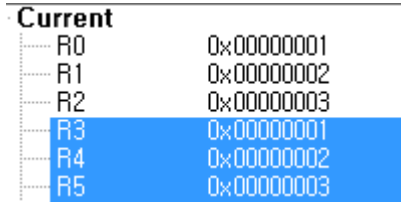
Simple example

- Copy values from R0-R2 to R3-R5
  - Code size: 36 bytes, states: 16

```
    AREA blockData, CODE, READONLY
        ENTRY
start
    MOV r0, #1
    MOV r1, #2
    MOV r2, #3

    LDR sp, TEMPADDR1

①   STMFA sp!, {r0}
    STMFA sp!, {r1-r2}

②   LDMFA sp!, {r3-r5}

    MOV pc, #0

TEMPADDR1 & &00040000
    END
```



②



Flow control instructions

- Determine the instruction to be executed next

```
Syntax: B{<cond>} label
        BL{<cond>} label
        BX{<cond>} Rm
        BLX{<cond>} label | Rm
```

B	branch	*pc* = *label* pc-relative offset within 32MB
BL	branch with link	*pc* = *label* *lr* = address of the next instruction after the BL
BX	branch exchange	*pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1
BLX	branch exchange with link	*pc* = *label*, *T* = 1 *pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1 *lr* = address of the next instruction after the BLX

- B *label*

  ...

label: ...

Branch conditions

- Conditional field {Cond}

Suffix	Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not Equal
CS/HS	C set	Higher or same (unsigned >=)
CC/LO	C clear	Lower (unsigned <)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned >)
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V different	Signed <
GT	Z clear, and N and V the same	Signed >
LE	Z set, or N and V different	Signed <=
AL	Any	Always (This suffix is normally omitted.)
NV	Researved	

Simple example

- Addition of bigger value
  - Code size: 48 bytes, states: 9-11 *(caused by branch)*

```
    AREA    branch, CODE, READONLY
        ENTRY

start
        MOV r0, #1
        MOV r1, #2

        CMP r0, r1          ;CPSR update
        BMI   Label1        ;If r0 < r1, jump Label1
        BHI   Label2        ;else If r0 > r1, jump Label2
        ADD  r2, r0, r1     ;else r2 = r0+r1
        B       Done        ;go to the end

Label1
        ADD  r2, r1, r1     ;r2 = r1+r1
        B       Done        ;go to the end

Label2
        ADD  r2, r0, r0     ;r2 = r0+r0
        B       Done        ;go to the end

Done
        MOV     pc, #0      ;end
        END
```

Exercise (1/2)

- Subroutine *(i.e., function() in C/C++ language)*
  - 다음 두 프로그램을 구현한 후 메모리에 저장된 값의 차이를 확인하라.
  - 두 프로그램의 동작은 다음과 같다.
    - R0와 R1의 숫자를 더한 후 R0에 그 값을 넣는다. 해당 동작은 함수 doadd()에서 진행한다.
    - 함수에서 돌아온 후 R0에서 R1을 뺀 값을 다시 R0에 넣는다.
    - 위의 두 단계(덧셈, 뺄셈)마다 메모리 0x00040000 번지에 결과 값을 저장한다.
  - *두 프로그램의 동작은 같지만 전혀 다른 결과를 도출하기 때문에, 퍼포먼스 비교는 의미가 없음에 유의 하시오.*

Exercise (2/2)

- Subroutine program codes

①

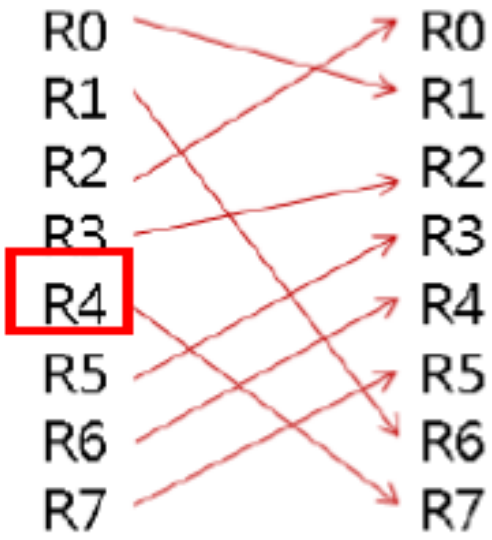
```
;-----52 bytes 26 states
   AREA    subrout, CODE, READONLY
      ENTRY

start   MOV     r0, #10          ; set up parameters
      MOV     r1, #3

      LDR r2, TEMPADDR1
      MOV lr, pc              ;store return address
      B     doadd             ;call subroutine
      SUB   r0, r0, r1        ;subtraction
      STR   r0, [r2]      ;store on mem

stop
      MOV     pc, #0

doadd
      STMFD   sp!, {r0, r1, lr}    ;PUSH into stack
      ADD     r0, r0, r1         ; addition
      STR   r0, [r2], #4         ;store on mem
      LDMFD   sp!, {r0, r1, pc}  ;POP from strack

TEMPADDR1   &  &00040000

      END
```

②

```
;-----48 bytes 25 states
    AREA subrout, CODE, READONLY
        ENTRY
start
    MOV r0, #10
    MOV r1, #3

    LDR r2, TEMPADDR1
    BL doadd
    SUB r0, r0, r1
    STR r0, [r2]

stop
    MOV pc, #0

doadd
    STMFD sp!, {r0, r1, lr}
    ADD r0, r0, r1
    STR r0, [r2], #4
    LDMFD sp!, {r0, r1, pc}

TEMPADDR1 & &00040000
    END
```

Problem (1/2)

- Problem 1
  - 아래 그림에 주어진 대로 r0로 부터 r7까지 레지스터 간의 값을 바꾸는(왼편의 값을 오른편의 레지스터에 저장시키는) 코드를 작성하라.
  - 단, 반드시 stack 혹은 블록 복사 명령어를 활용



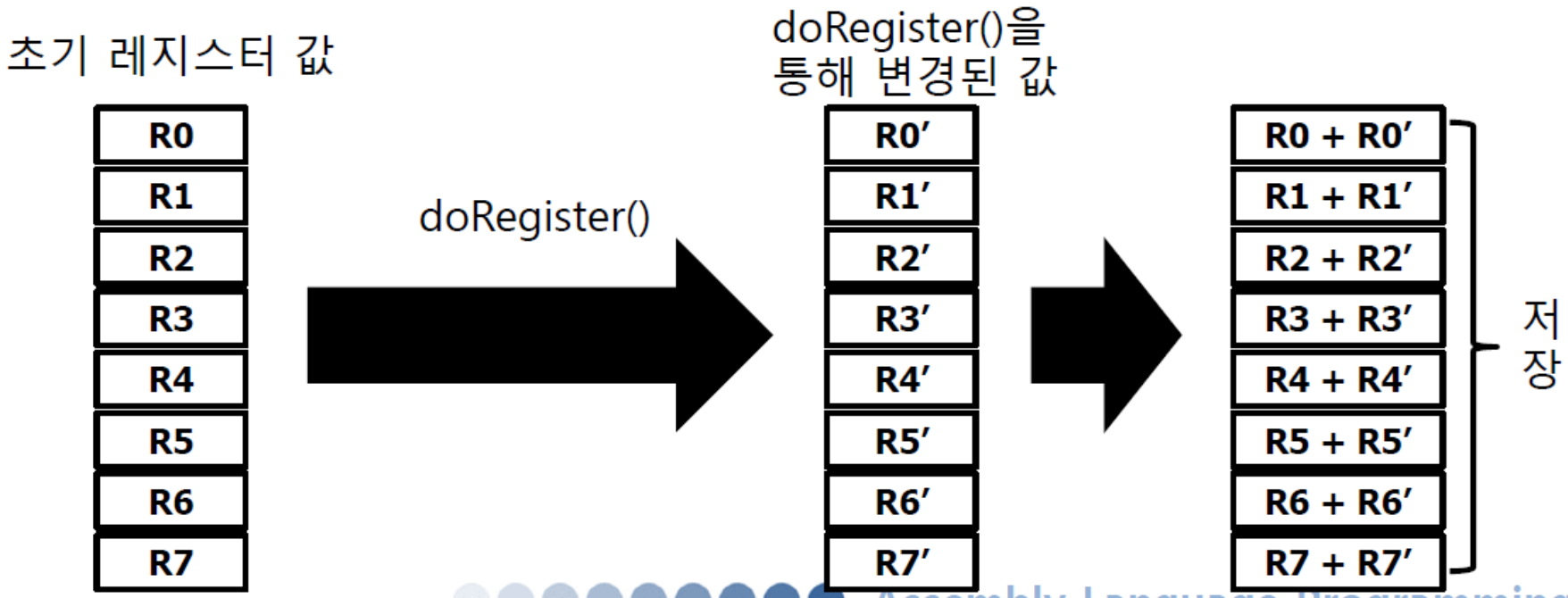
Problem (2/2)

- Problem 2
  - 다음 동작을 하는 프로그램을 작성하라
  - Register r0-r7에는 10~17의 값이 저장되어 있음
  - 함수 doRegister() 호출
    - 각 register의 값과 register의 index(예를 들어, r0의 index는 0, r1의 index는 1)를 더한 후, r0-r7의 값들의 합을 return
  - 함수 doGCD()
    - 위에서 구해진 r0-r7의 합과 160과의 최대공약수 구하기
  - 구해진 GCD와 각 register(r0-r7)들의 값 및 doRegister()를 통해 값이 변경되기 전의 값을 더한 결과 값을 0x40000번지에 저장
  - 프로그램 종료

Problem (2/2) cont.

- Problem 2에서 저장해야 할 값

- 총 9개의 값
  - 구해진 GCD
  - 각 register(r0-r7)들의 값과 doRegister()를 통해 값이 변경되기 전의 값를 더한 값



Homework

- 제출기한
  - Soft copy
    - 2019.10.11(금) 16시 29분 59초까지 u-campus에 제출
    - 압축 파일은 각 과제 파일을 모아서 제출(ini파일 반드시 포함)
    - 압축파일 형식
      - Assignment_(번호)_(학번).zip
        - ex) Assignment_4_2012722069.zip
    - 하드카피 제출 X

Q&A

Thank you