SPRING 2016
CMPE 364

Microprocessor Based Design

Dr. Ryan Riley
(Slides adapted from Dr. Mohamed Al-Meer)

Memory Access Instructions

ARM Architecture

Lecture Expectations

- Expected to achieve:
  - Know about **Memory model** and **Endians**
  - Know about **Addressing Modes**
  - Know **Single Register Load-Store** Instructions.
  - Know about **Loading Constants**
  - Know about **Multiple Registers Load-Store** Instructions
  - Know about The **Stack**.
  - Test Yourself and **solve problems**

Single Register Load Store Instructions - LDR

- Loads 32-bit word from memory to a register.
- Destination is DST
- Second operand is the addressing mode.
- Syntax

  LDR DST, <addressing mode>

Single Register Load Store Instructions - LDRH

- Load a register with a **Unsigned Half-Word** or 16-bit specified by addressing mode into destination register.
- Only the lower 16-bit part of DST is affected.
- Upper 16-bit of DST = 0.
- Syntax

  LDRH DST, <addressing mode>

Single Register Load Store Instructions - LDRH

- Example
  - Determine R3, R5, and R9 if R3 = 0xA6529123, R5 = 0x00000003,and R9 = 0x1000001F?
  - **LDRH R3, [R9, +R5]**
- Solution
  - EA = **0x10000022**
  - R3 = **0x0000F4E2**.
  - R5 and R9 unchanged.
  - **Memory location**
  - **With contents**

1000001C	20	**10000024**	FA
1000001D	2A	10000025	1E
1000001E	51	10000026	1A
1000001F	49	10000027	80
10000020	DA	**10000028**	8B
10000021	01	10000029	31
10000022	**E2**	1000002A	80
10000023	**F4**	1000002B	2D

Single Register Load Store Instructions - LDRSH

- Load a register with a **Signed Half-Word** or 16-bit specified by addressing mode into destination register.
- Only the lower 16-bit part of DST is affected.
- Upper 16-bit of DST = **Signed Extended**.
- Syntax

  LDRSH DST, <addressing mode>

Single Register Load Store Instructions - LDRSH

- Example
  - Determine R0, R4, if R0 = 0xA9D16250, R4 = 0x**37500052**?
  - **LDRSH R0, [R4], #-6**
- Solution
  - R0 = **0xFFFF9382**.
  - R4 = **0x3750004C**.

37500048	92	**37500050**	EF
37500049	16	37500051	01
3750004A	33	**37500052**	**82**
3750004B	20	**37500053**	**93**
3750004C	86	**37500054**	AB
3750004D	AB	37500055	69
3750004E	95	37500056	13
3750004F	DA	37500057	F6

Single Register Load Store Instructions - LDRB

- Load a register with a **Unsigned Byte** or 8-bit specified by addressing mode into destination register.
- Only the lower 8-bit part of DST is affected.
- Upper 16-bit of DST = **0's**.
- Syntax

  LDRB DST, <addressing mode>

Single Register Load Store Instructions - LDRB

- Example
  - Determine R0, R1, and R6 if R0 = 0xEA6C9421, R6 = 0x3FFFFF25, and R1 = 0x00000007?
  - **LDRB R0, [R6,+R1, LSL #6]!**
- Solution
  - R0 = **0x00000064**.
  - R6 = **0x400000E5**.
  - R4 unchanged.

400000DC	FE	**400000E4**	52
400000DD	8C	**400000E5**	**64**
400000DE	41	400000E6	87
400000DF	11	400000E7	9E
400000E0	82	**400000E8**	A4
400000E1	CA	400000E9	73
400000E2	90	400000EA	20
400000E3	1F	400000EB	15

Single Register Load Store Instructions - LDRSB

- Load a register with a **Signed Byte** or 8-bit specified by addressing mode into destination register.
- Only the lower 8-bit part of DST is affected.
- Upper 16-bit of DST = **Signed Extended**.
- Syntax

  LDRB DST, <addressing mode>

Single Register Load Store Instructions - LDRSB

- Example
  - Determine R7 if R8 = **0x00CD0049**?
  - **LDRSB R7, [R8]**
- Solution
  - R7 = **0x00000065**.
  - R8 unchanged.

00CD0040	65	**00CD0048**	2A
00CD0041	C2	**00CD0049**	**65**
00CD0042	92	00CD004A	9C
00CD0043	3A	00CD004B	71
00CD0044	FE	**00CD004C**	12
00CD0045	A9	00CD004D	FA
00CD0046	32	00CD004E	64
00CD0047	04	00CD004F	10

Single Register Load Store Instructions - STR

- Store Word stores the word in source register SRC at address specified by addressing mode.
- Syntax

  STR SRC, <addressing mode>

Single Register Load Store Instructions - STR

- Example
  - Indicate changes in memory after next instruction if R1 = **0xEA18DC04** and **R4** = **0x09E00BFC**?
  - **STR R1, [R4]**
- Solution
  - R1 = **unchanged**.
  - R4 = unchanged.

09E00BF8	52	**09E00C00**	FE
09E00BF9	F1	09E00C01	A7
09E00BFA	69	09E00C02	33
09E00BFB	10	09E00C03	01
09E00BFC	**04**	**09E00C04**	15
09E00BFD	**DC**	09E00C05	C2
09E00BFE	**18**	09E00C06	A5
09E00BFF	**EA**	09E00C07	3A

Single Register Load Store Instructions - STRH

- Store Half Word stores lower 16-bitin source register SRC at address specified by addressing mode.
- Syntax

  STRH SRC, <addressing mode>

Single Register Load Store Instructions - STRH

- Example
  - Indicate changes in memory after next instruction if R0 = **0xE9CA425E** and R1 = 0xA6C92F86, and R5 = 000000BC?
  - **STRH R0, [R1, -R5, LSL #6]!**
- Solution
  - EA = R1 – R5 << 6
  - EA = A6C92F85 – 000002F0 * 64
  - **EA = 0xA6C90086**

A6C90080	XX	**A6C90088**	XX
A6C90081	XX	A6C90089	XX
A6C90082	XX	A6C9008A	XX
A6C90083	XX	A6C9008B	XX
A6C90084	XX	**A6C9008C**	XX
A6C90085	XX	A6C9008D	XX
A6C90086	**5E**	A6C9008E	XX
A6C90087	**42**	A6C9008F	XX

Single Register Load Store Instructions - STRB

- Store Byte stores lower 8-bitin source register SRC at address specified by addressing mode.
- Syntax

  STRB SRC, <addressing mode>

Single Register Load Store Instructions - STRB

- Example
  - Indicate changes in memory after next instruction if R5 = **0x644E4EC5** and R0 = **0x7DB0532E**?
  - **STRB R0, [R5], #-2**
- Solution
  - **EA = R5 = 0x644E4EC5**
  - **R0 = unchanged**
  - **R5 = 0x644E4EC3**

644E4EBC	XX	**644E4EC4**	XX
644E4EBD	XX	**644E4EC5**	**2E**
644E4EBE	XX	644E4EC6	XX
644E4EBF	XX	644E4EC7	XX
644E4EC0	XX	**644E4EC8**	XX
644E4EC1	XX	644E4EC9	XX
644E4EC2	XX	644E4ECA	XX
644E4EC3	XX	644E4ECB	XX

Loading Constants

- You might have noticed that there is no ARM instruction to move a 32-bit constant into a register.
  - Why? Since ARM instructions are 32 bits in size, they obviously cannot specify a general 32-bit constant
- Only 8-bit data can be accommodated
- Pseudo instruction looks like an assembly language but cannot be mapped to a single native instruction.
  - Must be translated to multiple native ones

Loading Constants

- To aid programming there are two **pseudoinstructions** to move a 32-bit value into a register.
- Development tools allows **pseudoinstructions**.

```
Syntax: LDR Rd, =constant
        ADR Rd, label
```

LDR	load constant pseudoinstruction	*Rd* = 32-bit constant
ADR	load address pseudoinstruction	*Rd* = 32-bit relative address

Loading Constants

- **Syntax**

  LDR DST, =immediate_data

- Example

  MOV R6, #-6

- -6 = FFFF FFFA which be accommodated in a single instruction. So the solution will be as a following:
  - Cannot be represented in 8-bit value.
  - Use of MVN with sign extension.

  MVN R6, #5

Loading Constants

- Example
- Load a constant 0xEC017404 into R4 using Pseudoinstruction?
- Solution
  - LDR R4, = 0xEC017404

Multiple Register Load-store Instruction

- ARM has instructions to transfer data between multiple registers and memory.
- It is a special type of Load-Store instructions (LDM and STM)
- Multiple-register transfer instructions are more efficient from single-register transfers for
  - Moving blocks of data around memory and
  - Saving and restoring context and stacks
- LDM and STM are useful for copying memory block from address to another address.

Multiple Register Load-store Instruction

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

LDM	load multiple registers	{Rd}*N <- mem32[start address + 4*N] optional Rn updated
STM	save multiple registers	{Rd}*N -> mem32[start address + 4*N] optional Rn updated

Addressing mode for load-store multiple instructions.

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 Register Load-store Instruction - LDM

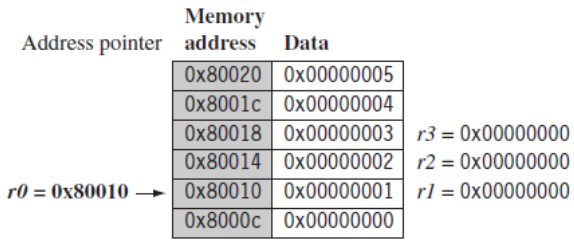
- Load Multiple Register instructions copies words from memory to multiple registers
- **Syntax**

  LDM<address_mode> SRC, {register list}
- SRC has a reference to block of memory. Must be even aligned (divisible by 4).
- Register lists may be comma delimited list, a range of registers, or both.
- Order of registers is irrelevant. Lowest numbered register is loaded with from memory with lowest address.

Multiple Register Load-store Instruction - LDM

- **Example**

  LDMIA r0!, {r1-r3}



POST r0 = 0x0008001c

r1 = 0x00000001

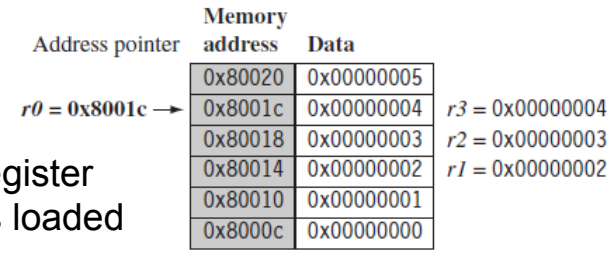
r2 = 0x00000002

r3 = 0x00000003

Multiple Register Load-store Instruction - LDM

- **Example**
- Now replace the **LDMIA** instruction with a load multiple and increment before **LDMIB** instruction and use the same PRE conditions.

LDMIB r0!, {r1-r3}



- The first word pointed to by register *r0* is ignored and register *r1* is loaded from the next memory location as shown in next Figure.

Multiple Register Load-store Instruction - STM

- Store Multiple Register Instruction (STM) copies words from registers specified in register list to memory
- SRC has a reference to block of memory. Must be even aligned (divisible by 4).
- Register lists may be comma delimited list, a range of registers, or both.
- Order of registers is irrelevant. Lowest numbered register is stored to memory with lowest address.
- **Syntax**

STM<address_mode> SRC, {register list}

Multiple Register Load-store Instruction - STM

- **Example**

  r5 = 0xC0640024

  STMIA r5, {r1, r6-r8, r2}

PRE	r1 = 0x00009000	**POS**	r1 > 0xC0640024
	r2 = 0x0000000		r2 > 0xC0640028
	r6 = 0x00000008		r6 > 0xC064002C
	r7 = 0x00000007		r7 > 0xC0640030
	r8 = 0x00000008		r8 > 0xC0640034

Multiple Register Load-store Addressing Modes

- **Increment After (IA)**
  - Accesses the first word of the block in SRC.
  - After each access from memory, Effective Address is Incremented by 4.

    SRC + 4 * (n-1) n = number of registers in list
  - ! Indicates auto indexing after SRC, so
  - SRC is updated with

    SRC + 4 * n.

Multiple Register Load-store Addressing Modes

- **Increment After (IA)**
- **Example**
- Indicated changes in memory and registers after the execution of next instruction, R0 = 0x91762103, R1 = 0x1545BAC1, R4 = 0xE64C0098, R3 = 0xE917E8F4.

STMIA R4, {R1, R3, R0}

- R4 unchanged(NOT Auto indexing)
- R0, R1, R3 are saved in mem location
- 0xE64C0098, and unchanged

POS r0 > 0xE64C0098
r1 > 0xE64C009C
r3 > 0xE64C00A0

Multiple Register Load-store Addressing Modes

- **Increment Before (IB)**
- Adds 4 to SRC before accessing the first location.
- Next locations are accessed by incrementing EA by 4 before accessing the next one.
- Ending address is:

  SRC + 4 *n

- n = number of registers in list
- ! Indicates auto indexing after SRC

Multiple Register Load-store Addressing Modes

- **Increment Before (IB)**
- **Example**

  STMIB R9!, {R10, R2-R0}
- If **R9 = 0x000080F8**?
- **R9** changes to **0x00008108**.
- R0, R1, R2, R10 NOT changed.

PRO r0 > 0x000080FC
r1 > 0x00008100
r2 > 0x00008104
r10 >0x00008108

Multiple Register Load-store Addressing Modes

- **Decrement After (DA)**
  - Uses SRC as EA for first memory access and decrement EA by 4 **after** each memory access.
  - Address range is from SRC to SRC – 4 * n, where n number of registers in the list.
  - ! Indicates auto indexing after the SRC, which will update SRC by SRC – 4 * n

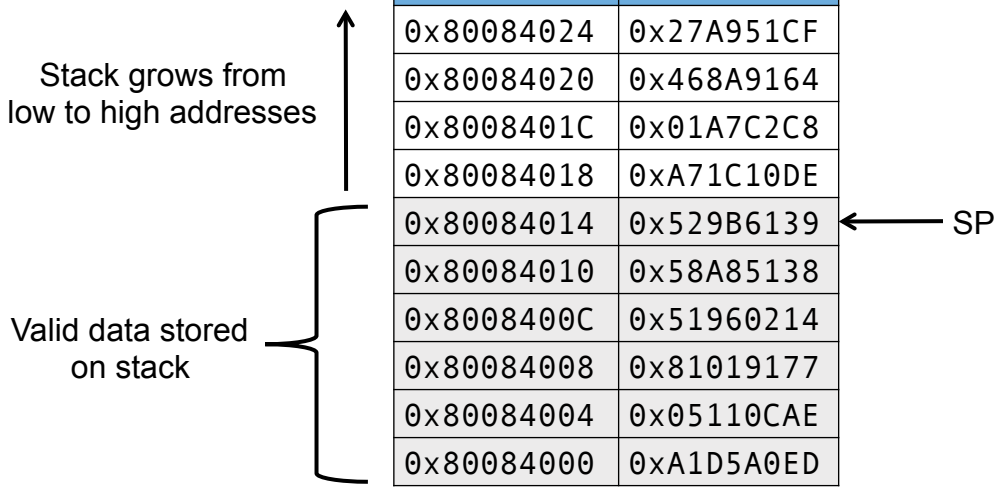
Multiple Register Load-store Addressing Modes

- **Decrement Before (DB)**
  - Uses SRC as EA for first memory access and decrement EA by 4 **before** each memory access.
  - Starting address is SRC to SRC – 4 , and ending address is SRC - 4*n, where n number of registers in list.
  - ! Indicates auto indexing after the SRC, which will update SRC by SRC – 4 * n (ending address).

The Stack

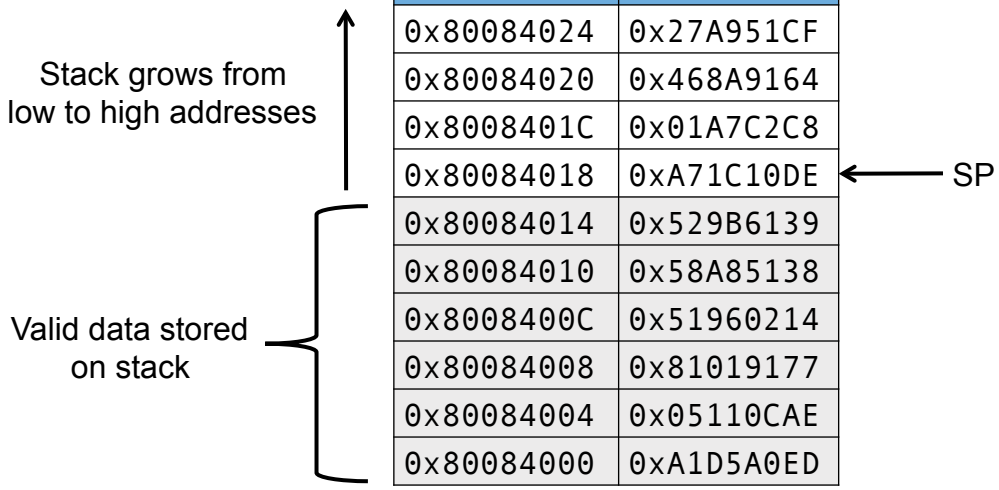
- The ARM architecture uses the load-store multiple instructions to carry out stack operations
- The *pop* operation (removing data from a stack) uses a load multiple instruction.
- the *push* operation (placing data onto the stack) uses a store multiple instruction.
- When using a stack you have to decide whether the stack will **grow up** or **down** in memory.
  - A stack is either ***ascending* (A) or *descending* (D)**.
- You also need to decide if the stack point (SP) will point to the next empty space or the last full space

Full Ascending Stack



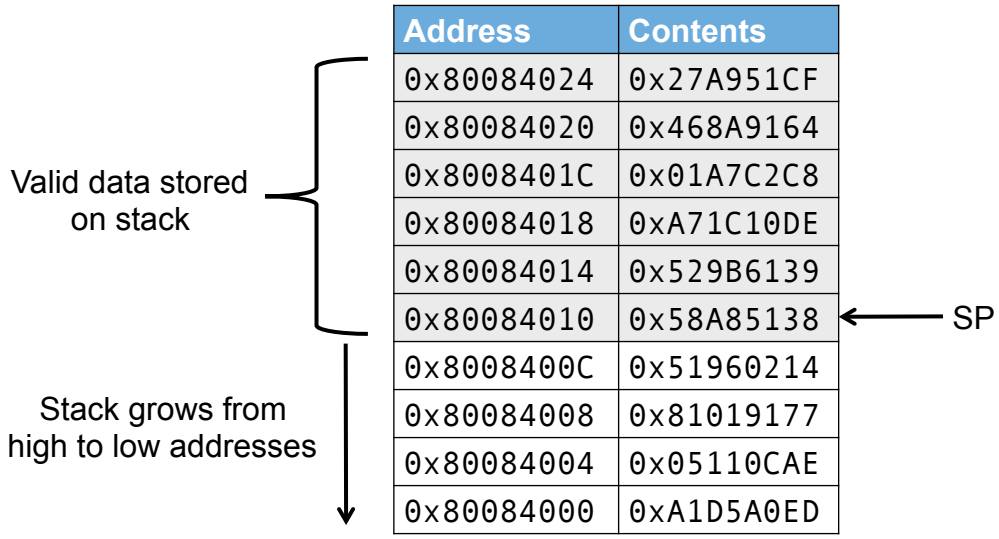
Address	Contents
0x80084024	0x27A951CF
0x80084020	0x468A9164
0x8008401C	0x01A7C2C8
0x80084018	0xA71C10DE
0x80084014	0x529B6139
0x80084010	0x58A85138
0x8008400C	0x51960214
0x80084008	0x81019177
0x80084004	0x05110CAE
0x80084000	0xA1D5A0ED

Empty Ascending Stack

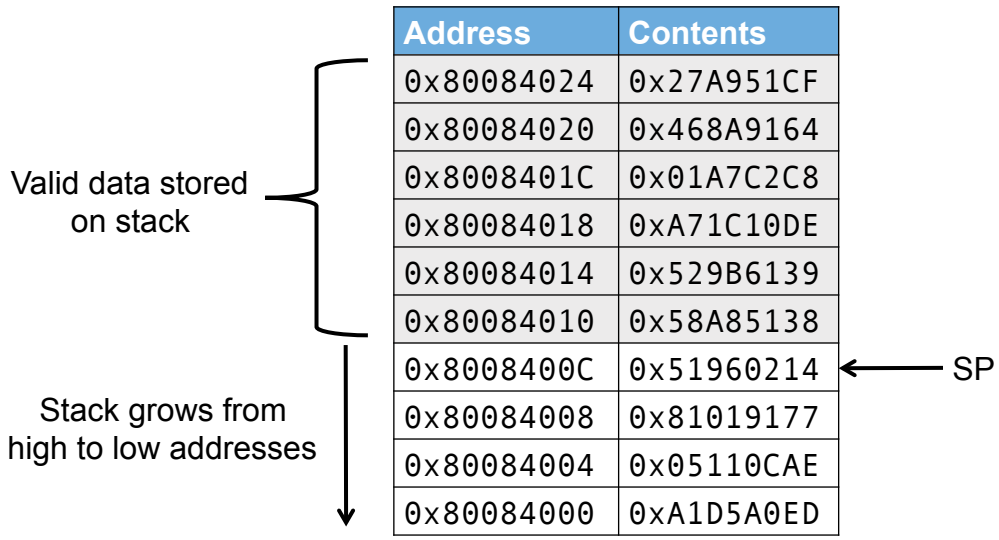


Address	Contents
0x80084024	0x27A951CF
0x80084020	0x468A9164
0x8008401C	0x01A7C2C8
0x80084018	0xA71C10DE
0x80084014	0x529B6139
0x80084010	0x58A85138
0x8008400C	0x51960214
0x80084008	0x81019177
0x80084004	0x05110CAE
0x80084000	0xA1D5A0ED

Full Descending Stack



Empty Descending Stack



Multiple Register Load-store Stack

Addressing methods for stack operations.

Addressing mode	Description	Pop	= LDM	Push	= STM
FA	full ascending	LDMFA	LDMDA	STMFA	STMIB
FD	full descending	LDMFD	LDMIA	STMFD	STMDB
EA	empty ascending	LDMEA	LDMDB	STMEA	STMIA
ED	empty descending	LDMED	LDMIB	STMED	STMDA

The Stack - FD

- **EXAMPLE**
  - **LDMFD SP!, {R0, R5, R2}**
  - SP = 0xFFFFFE00
  - Data loaded to r0, r2, and r5 from full descending stack
  - R0 = mem32(0xFFFFFE00)
  - R2 = mem32(0xFFFFFE04)
  - R5 = mem32(0xFFFFFE08)
  - SP = mem32(0xFFFFFE0C)

The Stack - FD

- **EXAMPLE**
  - **STMFD SP, {R1, R4}**
  - SP = 0xFFF008040, R1 = 0x01506273, R4 = 0xAFFE2401

  - Data from r1 and r4 pushed to full descending stack
  - mem32(0xFFF00803C) = R4
  - mem32(0xFFF008038) = R1
  - SP = UNCHANGED.

Multiple Register Load-store Stack - ED

- **EXAMPLE**
  - **LDMED SP, {R7 – R5}**
  - SP = 0xFFFFF05C
  - TOS Empty
  - Data when popped SP incremented before data popped out
  - R5 = mem32(0xFFFFF060)
  - R6 = mem32(0xFFFFF064)
  - R7 = mem32(0xFFFFF068)

Multiple Register Load-store Stack - ED

- **EXAMPLE**
  - **STMED SP!, {R8, R14}**
  - **SP = 0x800EBFF8**, R8 = ?, R14 = ?.

  - mem32(**0x800EBFF8**) = R14
  - mem32(**0x800EBFF4**) = R8
  - SP = **0x800EBFF0**

Multiple Register Load-store Stack - FA

- **EXAMPLE**
  - **LDMFA SP!, {R2, R9}**
  - **SP = 0x8000040E4**

  - mem32(**0x8000040E4**) -> R9
  - mem32(**0x8000040E0**) -> R2
  - SP = **0x8000040DC** = TOS

Multiple Register Load-store Stack - FA

- **EXAMPLE**
  - **STMFA SP, {R0, R12, R9}**
  - **SP = 0x800C0808**?

  - mem32(**0x800C080C**) <- **R0**
  - mem32(**0x800C0810**) <- **R9**
  - mem32(**0x800C0814**) <- **R12**
  - SP = UNCHANGED.

Multiple Register Load-store Stack - EA

- **EXAMPLE**
  - **LDMEA SP, {R1-R3, R0}**
  - **SP = 0x00040A88**

  - **R3** = mem32(**0x00040A84**)
  - **R2** = mem32(**0x00040A80**)
  - **R1** = mem32(**0x00040A7C**)
  - **R0** = mem32(**0x00040A78**)
  - **SP** = UNCHANGED.

Multiple Register Load-store Stack - EA

- **EXAMPLE**
  - **STMEA SP!, {R12, R14}**
  - **SP = 0x00000E00**

  - mem32(**0x00000E00**) <- R12
  - mem32(**0x00000E04**) <- R14
  - **SP = 0x00000E08**.

Swap Instruction

- Swaps a **Byte** or a **Word** between a register an a memory.
- Atomic Operation
  - Reads and writes from to memory in the same cycle.
- Syntax

  SWP Rd, Rm, Rn
- What happens?

  temp = mem32[rn]

  mem32[rn] = rm

  rd = temp

Swap Instruction

- Swaps a **Byte** or a **Word** between a register an a memory.
- Atomic Operation
  - Reads and writes from to memory in the same cycle. Used to implement things like semaphores
- Syntax

SWP Rd, Rm, Rn	**SWPB Rd, Rm, Rn**
• What happens?	
temp = mem32[rn]	temp = mem8[rn]
mem32[rn] = rm	mem8[rn] = rm
rd = temp	rd = temp