Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

Chapter 5
Memory Access
Exercises ANS

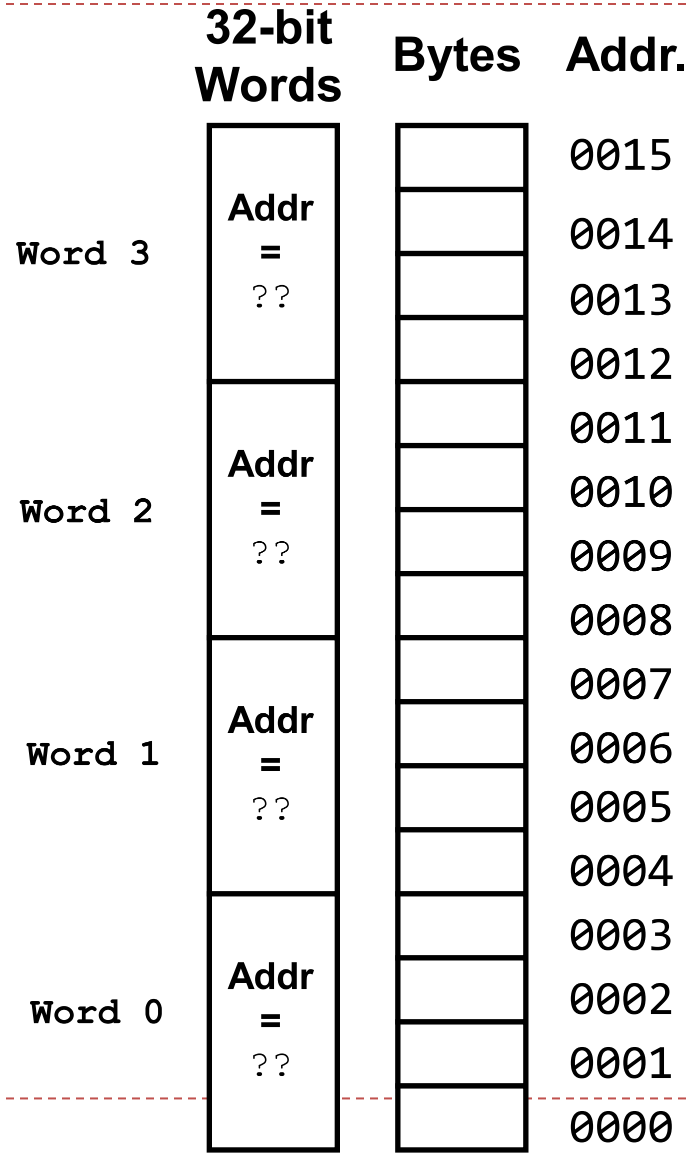
Zonghua Gu

Fall 2025

Acknowledgement: Lecture slides based on Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, University of Maine https://web.eece.maine.edu/~zhu/book/

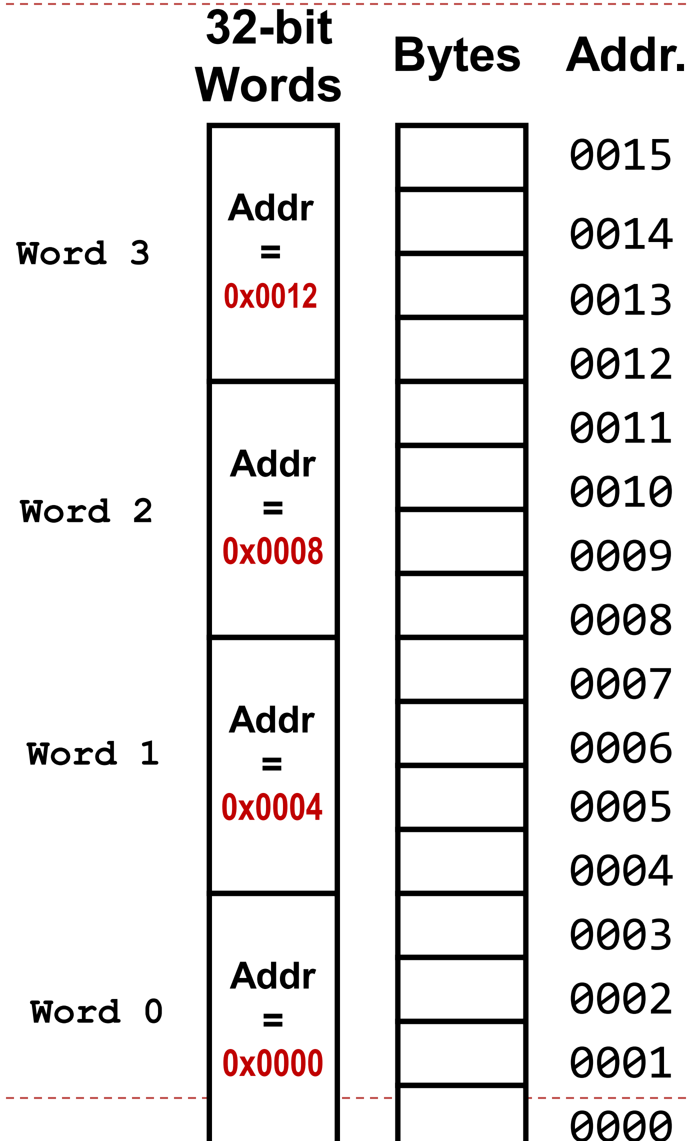
Question: Endianness

What are the memory address of these four words?



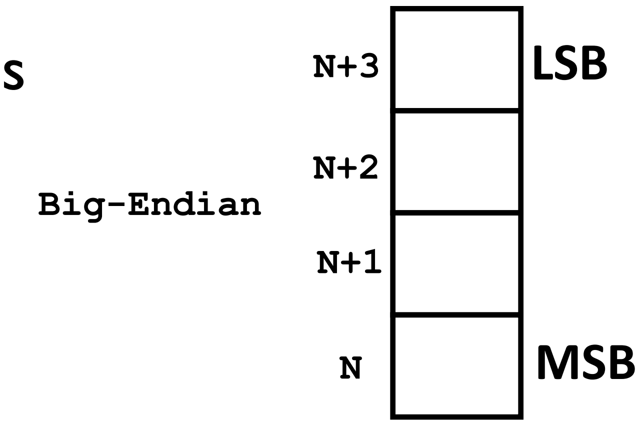
Answer: Endianness

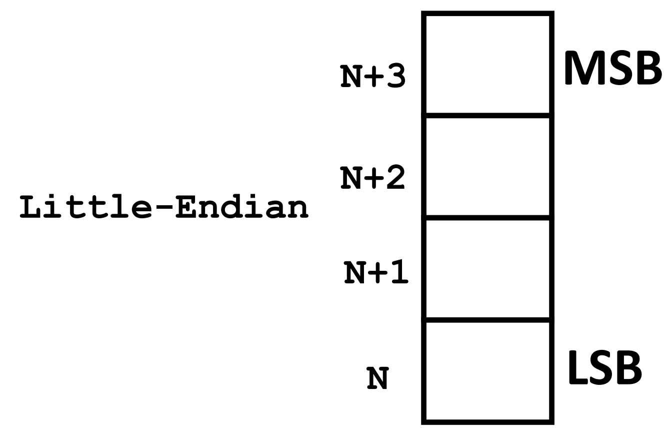
What are the memory address of these four words?
Same as the address of the lowest-address Byte
(this is true **`for either Little-Endian or Big-Endian ordering)`**



Question: Endianness

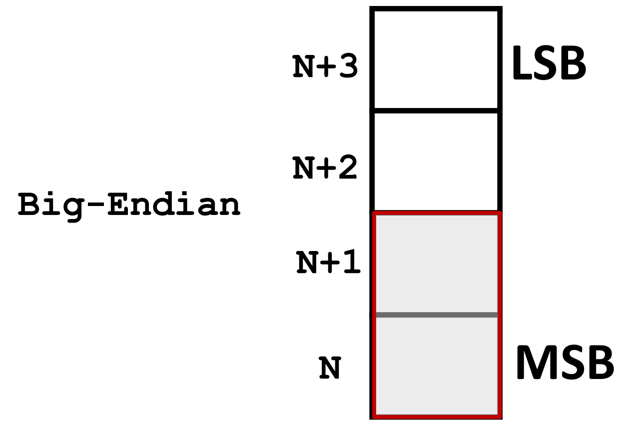
- Q: Assume Big-Endian ordering. If a 32-bit word resides at memory address N, what is the address of:
  - (a) The MSB (Most Significant Byte)
  - (b) The 16-bit half-word corresponding to the most significant half of the word
- Q: Redo the question assuming Little-Endian ordering.



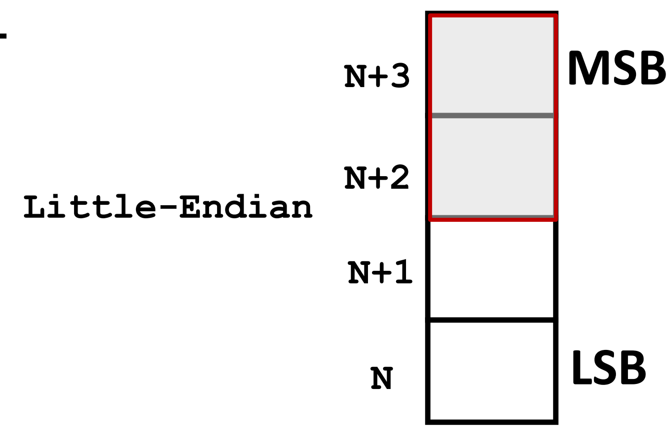


Answer: Endianness

- A: With Big-Endian ordering:
  - (a) Address of MSB: N
  - (b) Address of 16-bit half-word corresponding to the most significant half of the word: N (the half-word has address range of [N, N+1], so its address is N)



- With Little-Endian ordering:
  - (a) Address of MSB: N+3
  - (b) Address of 16-bit half-word corresponding to the most significant half of the word: N+2 (the half-word has address range of [N+2, N+3], so its address is N+2)



Question: Endianness

The word stored at address 0x20008000 with Big-Endian ordering is

?

The word stored at address 0x20008000 with Little-Endian ordering is

?

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Answer: Endianness

The word stored at address 0x20008000 with Big-Endian ordering is

0xEE8C90A7

The word stored at address 0x20008000 with Little-Endian ordering is

0xA7908CEE

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Endianness only specifies byte order, not bit order in a byte!

Endianness

```
LDR r11, [r0]
; r0 = 0x20008000
```

r11 before load

0x12345678

r11 after load w/ Big-Endian ordering

r11 after load w/ Little-Endian ordering

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Endianness ANS

```
LDR r11, [r0]
; r0 = 0x20008000
```

r11 before load

0x12345678

r11 after load w/ Big-Endian ordering

0xEE8C90A7

r11 after load w/ Little-Endian ordering

0xA7908CEE

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Endianness ANS

```
LDR r11, [r0]
; r0 = 0x20008000
```

r11 before load

0x12345678

r11 after load w/ Big-Endian ordering

0xEE8C90A7

r11 after load w/ Little-Endian ordering

0xA7908CEE

Endianness

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

- Assume little endian for the following questions: r0 = 0x20008000
- **LDRH r1, [r0]**
  - r1 after load:
- **LDSB r1, [r0]**
  - r1 after load:
- **STR r1, [r0], #4**
  - Assume r1 = 0x76543210
  - r0 after store:
  - Memory content after store:
- **STR r1, [r0, #4]**
  - Assume r1 = 0x76543210
  - r0 after load:
  - Memory content after store:
- **STR r1, [r0, #4]!**
  - Assume r1 = 0x76543210
  - r0 after load:
  - Memory content after store:

Endianness ANS

- Assume little endian for the following questions: r0 = 0x20008000
- **LDRH r1, [r0]**
  - r1 after load: 0x00008CEE
- **LDSB r1, [r0]**
  - r1 after load: 0xFFFFFFEE
- **STR r1, [r0], #4**
  - Assume r1 = 0x76543210
  - r0 after store: 0x20008004
    - Post-index. Store at old r0, then r0 = r0 + 4.
  - Memory content after store:

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Memory Address	Memory Data
0x20008007	
0x20008006	
0x20008005	
0x20008004	
0x20008003	0x76
0x20008002	0x54
0x20008001	0x32
0x20008000	0x10

Endianness ANS

- **STR r1, [r0, #4]**
  - Assume r1 = 0x76543210
  - r0 after load: 0x20008000
    - Pre-index. Store at r0 + 4; r0 unchanged.
  - Memory content after store:

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Memory Address	Memory Data
0x20008007	0x76
0x20008006	0x54
0x20008005	0x32
0x20008004	0x10
0x20008003	
0x20008002	
0x20008001	
0x20008000	

Endianness ANS

- **STR r1, [r0, #4]!**
  - r0 := r0 + 4; store at new r0.
  - Assume r1 = 0x76543210
  - r0 after load: 0x20008004
    - Pre-index with update. r0 = r0 + 4; store at new r0.
  - Memory content after store:

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Memory Address	Memory Data
0x20008007	0x76
0x20008006	0x54
0x20008005	0x32
0x20008004	0x10
0x20008003	
0x20008002	
0x20008001	
0x20008000	

Review

Data Alignment

- **Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide**
- **Consider 16 bytes of memory (addresses 0 to 15) arranged as four 32-bit words (4**

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9	Address 8
Address 7 (MSbyte)	Address 6	Address 5	Address 4 (LSbyte)
Address 3	Address 2	Address 1	Address 0

Well-aligned: each word begins on a mod-4 address, which can be read in a single memory cycle

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9 (MSbyte)	Address 8
Address 7	Address 6 (LSbyte)	Address 5	Address 4
Address 3	Address 2	Address 1	Address 0

Ill-aligned: a word begins on address 6, not a mod-4 address, which can be read in 2 memory cycles

```
The first read cycle would retrieve 4 bytes
from addresses 4 through 7; of these, the
bytes from addresses 4 and 5
are discarded, and those from addresses 6
and 7 are moved to the far right;
The second read cycle retrieves 4 bytes from
addresses 8 through 11; the bytes from
addresses 10 and 11 are  discarded, and
those from addresses 8 and 9 are moved to
the far left;
Finally, the two halves are combined to form
the desired 32-bit operand:
```

		Address 7	Address 6 (LSbyte)

Address 9 (MSbyte)	Address 8		

Address 9 (MSbyte)	Address 8	Address 7	Address 6 (LSbyte)

Question: Data Alignment

- Q: Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide. Consider 16 bytes of memory (addresses 0 to 15) arranged as four 32-bit words (4 bytes each). How many memory cycles are required to read each of the following from memory?
  - (a) A 2-Byte operand read from decimal address 5
  - (b) A 2-Byte operand read from decimal address 15
  - (c) A 4-Byte operand read from decimal address 10
  - (d) A 4-Byte operand read from decimal address 20

Answer: Data Align

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9	Address 8
Address 7	Address 6	Address 5	Address 4
Address 3	Address 2	Address 1	Address 0

- Q: Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide. How many memory cycles are required to read each of the following from memory?
  - (a) A 2-Byte operand read from decimal address 5
  - (b) A 2-Byte operand read from decimal address 15
  - (c) A 4-Byte operand read from decimal address 10
  - (d) A 4-Byte operand read from decimal address 20
- A: (a) The operand contains memory content in address range [5,6]. It can be read in 1 memory cycle; the memory controller returns a word in address range [4,7]. The operand can be obtained via 1-Byte offset addressing into the word.
- (b) The operand contains memory content in address range [15,16]. It can be read in 2 memory cycles; the memory controller returns 2 words in address ranges [12,15] and [16, 19], which can be combined to return a word in address range [14,17]. The operand can be obtained via 1-Byte offset addressing into the word.
- (c) The operand contains memory content in address range [10,13]. It can be read in 2 memory cycles; the memory controller returns 2 words in address ranges [8,11] and [12, 15], which can be combined to return a word with address range [10,13].
- (d) The operand contains memory content in address range [20,23]. Since 20%4=0, it is well-aligned, and can be read in 1 memory cycle.

Question: Data Align

Address 111	Address 110	Address 109	Address 108
Address 107	Address 106	Address 105	Address 104
Address 103	Address 102	Address 101	Address 100
Address 99	Address 98	Address 97	Address 96

- Q: Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide. Consider 16 bytes of memory (addresses 0 to 15) arranged as four 32-bit words (4 bytes each).
  - (a) What is the address of MSB of the word at address 102, , assuming Little-Endian ordering?
  - (b) What is the address of LSB of the word at address 102, , assuming Little-Endian ordering?
  - (b) How many memory cycles are required to read the word at address 102?
  - (c) How many memory cycles are required to read the half word at address 102?

Answer: Data Align

Address 111	Address 110	Address 109	Address 108
Address 107	Address 106	Address 105	Address 104
Address 103	Address 102	Address 101	Address 100
Address 99	Address 98	Address 97	Address 96

- Q: Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide. Consider 16 bytes of memory (addresses 0 to 15) arranged as four 32-bit words (4 bytes each).
  - (a) What is the address of MSB of the word at address 102, , assuming Little-Endian ordering?
  - (b) What is the address of LSB of the word at address 102, , assuming Little-Endian ordering?
  - (b) How many memory cycles are required to read the word at address 102?
  - (c) How many memory cycles are required to read the half word at address 102?
- A:
  - (a) MSB of the word at address 102 is 105
  - (b) LSB of the word at address 102 is 102
  - (c) 2 cycles.
  - (d) 1cycle.

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9	Address 8
Address 7	Address 6	Address 5	Address 4
Address 3	Address 2	Address 1	Address 0

Answer: Memory Cycles

- Q: Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide.
  - It takes _____ memory cycle(s) to read a Byte from memory
  - It takes _____ memory cycle(s) to read a half-word from memory
  - It takes _____ memory cycle(s) to read a word from memory
  - It takes _____ memory cycle(s) to read a double word from memory

Answer: Memory Cycles

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9	Address 8
Address 7	Address 6	Address 5	Address 4
Address 3	Address 2	Address 1	Address 0

- Q: Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide.
  - It takes _____ memory cycle(s) to read a Byte from memory
  - It takes _____ memory cycle(s) to read a half-word from memory
  - It takes _____ memory cycle(s) to read a word from memory
  - It takes _____ memory cycle(s) to read a double word from memory
- A:
  - It takes __1__ memory cycle(s) to read a Byte from memory
  - It takes __1 or 2__ memory cycle(s) to read a half-word from memory
  - It takes __1 or 2__ memory cycle(s) to read a word from memory
  - It takes __2 or 3__ memory cycle(s) to read a double word from memory (a double word may span at most 3 consecutive words in memory)

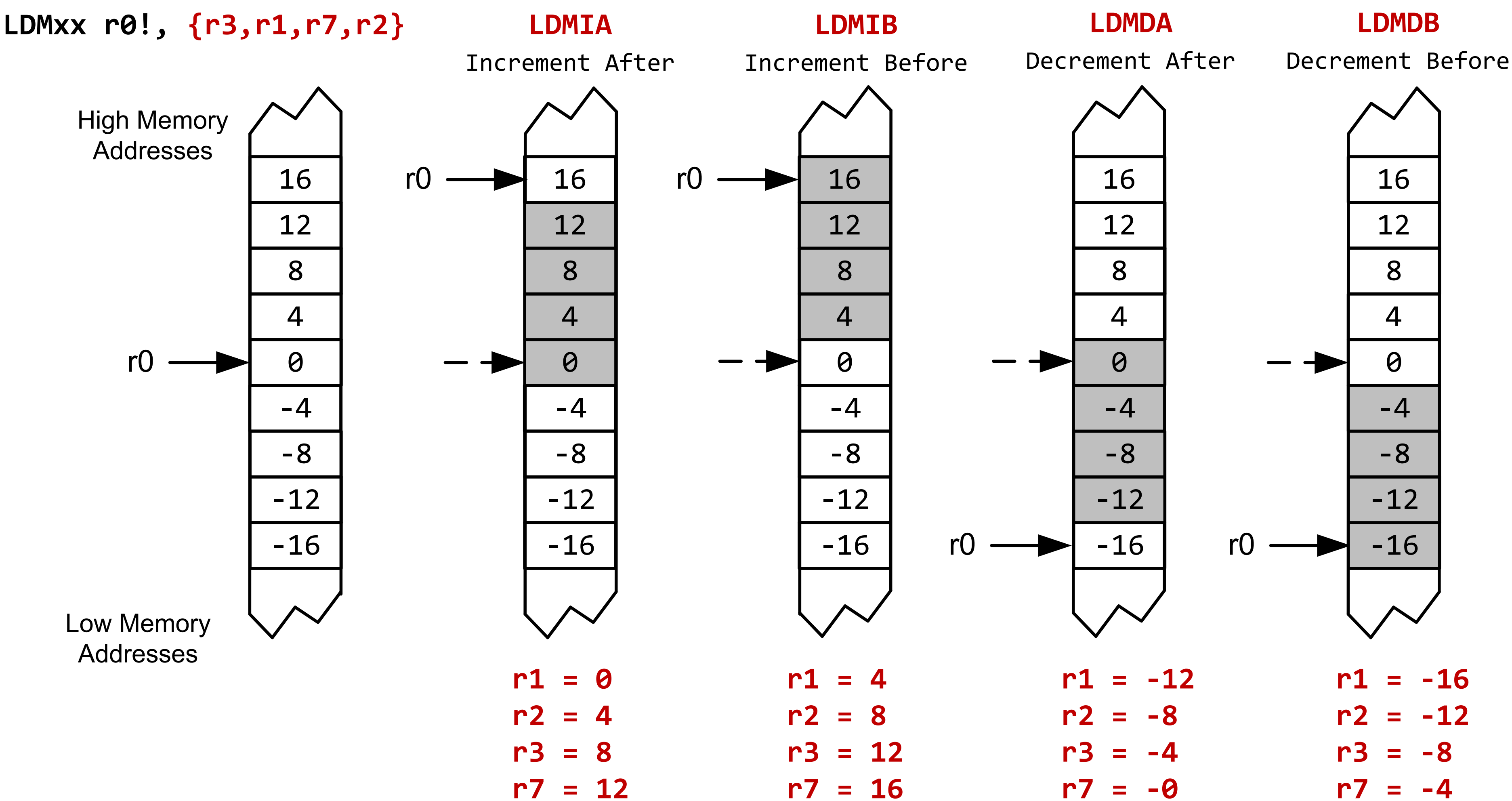
Question: Arrays

- Q: If the first element of a one-dimensional array x[] is stored at memory address 0x12345678, what is address of the second element if the array x[] contains
  - (a) chars
  - (b) shorts
  - (c) ints
  - (c) longs

Answer: Arrays

- Q: If the first element of a one-dimensional array x[] is stored at memory address 0x12345678, what is address of the second element if the array x[] contains
  - (a) chars
  - (b) shorts
  - (c) ints
  - (c) longs
- A: x[1]'s address is x's address plus the data type size in Bytes
  - (a) chars: 0x12345678+1= 0x12345679
  - (b) shorts: 0x12345678+2= 0x1234567A
  - (c) ints: 0x12345678+4= 0x1234567C
  - (c) longs: 0x12345678+8= 0x12345680

Load Multiple Registers



LDM

- Assume that memory and registers r0 through r3 appear as follows. Suppose r3 = 0x8000. Describe the memory and register contents after executing each instruction (individually, not sequentially):
  - LDMIA r3!, {r0, r1, r2}
  - Or LDMIB r3!, {r2, r1, r0}
  - Or LDMIB r3!, {r1, r2, r0}

	Memory Address	Memory Data
	0x8010	0x00000001
	0x800c	0xFEEDDEAF
	0x8008	0x00008888
	0x8004	0x12340000
r3 ⇨	0x8000	0xBABE0000

LDM ANS

- Assume that memory and registers r0 through r3 appear as follows. Suppose r3 = 0x8000. Describe the memory and register contents after executing each instruction (individually, not sequentially):
  - LDMIA r3!, {r0, r1, r2}
  - Or LDMIB r3!, {r2, r1, r0}
- ANS:
  - After LDMIA r3!, {r0, r1, r2}
  - r0 = 0xBABE0000 (loaded from 0x8000)
  - r1 = 0x12340000 (loaded from 0x8004)
  - r2 = 0x00008888 (loaded from 0x8008)
  - r3 = 0x800C (auto-incremented)
  - Or after LDMIB r3!, {r2, r1, r0}
  - r0 = 0x12340000 (loaded from 0x8004)
  - r1 = 0x00008888 (loaded from 0x8008)
  - r2 = 0xFEEDDEAF (loaded from 0x800c)
  - r3 = 0x800C (auto-incremented)
  - Or after LDMIB r3!, {r1, r2, r0}
  - r0 = 0x12340000 (loaded from 0x8004)
  - r1 = 0x00008888 (loaded from 0x8008)
  - r2 = 0xFEEDDEAF (loaded from 0x800c)
  - r3 = 0x800C (auto-incremented)

	Memory Address	Memory Data
	0x8010	0x00000001
	0x800c	0xFEEDDEAF
	0x8008	0x00008888
	0x8004	0x12340000
r3 ⇨	0x8000	0xBABE0000

The order in which registers are listed does not matter. For STM/LDM, the lowest-numbered register is stored/loaded at the lowest memory address.

LDR

- Suppose R2 and R5 hold the values 8 and 0x23456789 After following code runs on a Big-Endian system, what value is in R7? How about in a little-endian system?
  - STR R5, [R2, #0]
  - LDRB R7, [R2, #1]
  - LDRSH R7, [R2, #1]
  - LDRSH R7, [R2, #2]

LDRB R7, [R2, #1] ANS

- ANS after LDRB R7, [R2, #1] (detailed explanations not needed for exam):
  - STR stores a 32-bit register value to memory at base-plus-immediate without changing the base, and LDRB loads a single byte and zero-extends to 32 bits, so endianness only affects which byte resides at offset +1.
  - R2 holds 8 (base address), and R5 holds 0x23456789; first the store writes that 32-bit word to memory at address R2+0, and then a byte load reads one byte from address R2+1 into R7.
  - Big-endian: the word 0x23456789 is laid out in memory as bytes 23 45 67 89 at addresses A, A+1, A+2, A+3 respectively, so LDRB R7,[R2,#1] reads 0x45 and zero-extends it to R7 = 0x00000045.
  - Little-endian: the same word is laid out as 89 67 45 23 at addresses A, A+1, A+2, A+3 respectively, so LDRB R7,[R2,#1] reads 0x67 and zero-extends it to R7 = 0x00000067.

	Memory Address	Memory Data
	0x0000000B	0x89
	0x0000000A	0x67
	0x00000009	0x45
R2 ⇨	0x00000008	0x23

	Memory Address	Memory Data
	0x0000000B	0x23
	0x0000000A	0x45
	0x00000009	0x67
R2 ⇨	0x00000008	0x89

LDRSH R7, [R2, #1] ANS

- ANS after LDRSH R7, [R2, #1]:
  - Big-endian: the word 0x23456789 is laid out in memory as bytes 23 45 67 89 at addresses A, A+1, A+2, A+3 respectively, so LDRSH R7, [R2, #1] reads 0x4567 and sign-extends it to R7 = 0x00004567. (Sign bit is 0 for 0x4567)
  - Little-endian: the same word is laid out as 89 67 45 23 at addresses A, A+1, A+2, A+3 respectively, so LDRSH R7, [R2, #1] reads 0x4567 and sign-extends it to R7 = 0x00004567. (Sign bit is 0 for 0x4567)

	Memory Address	Memory Data		Memory Address	Memory Data
	0x0000000B	0x89		0x0000000B	0x23
	0x0000000A	0x67		0x0000000A	0x45
	0x00000009	0x45		0x00000009	0x67
R2 ⇨	0x00000008	0x23	R2 ⇨	0x00000008	0x89

LDRSH R7, [R2, #2] ANS

- ANS after LDRSH R7, [R2, #1]:
  - Big-endian: the word 0x23456789 is laid out in memory as bytes 23 45 67 89 at addresses A, A+1, A+2, A+3 respectively, so LDRSH R7, [R2, #1] reads 0x6789 and sign-extends it to R7 = 0x00006789.
  - Little-endian: the same word is laid out as 89 67 45 23 at addresses A, A+1, A+2, A+3 respectively, so LDRSH R7, [R2, #1] reads 0x2345 and sign-extends it to R7 = 0x00002345.

	Memory Address	Memory Data
	0x0000000B	0x89
	0x0000000A	0x67
	0x00000009	0x45
R2	0x00000008	0x23

	Memory Address	Memory Data
	0x0000000B	0x23
	0x0000000A	0x45
	0x00000009	0x67
R2	0x00000008	0x89

E0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
F0	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
200	AA	1B	11	12	EE	FF	11	22	33	44	55	66	77	88	99	AA

0x00000000
0x10000200
0x0000FFFF
0x18675309
0x00000000
0x00000000
...
0x10000200

```
MOVW R0, #0xAFE1
MOVT R0, #0xBADC
MOVT R2, #0xABCD
STR R3, [R1]
LDRSH R4, [R1, #0xC]
PUSH {R1,R3}
```

Compute register and memory values at each step of this program

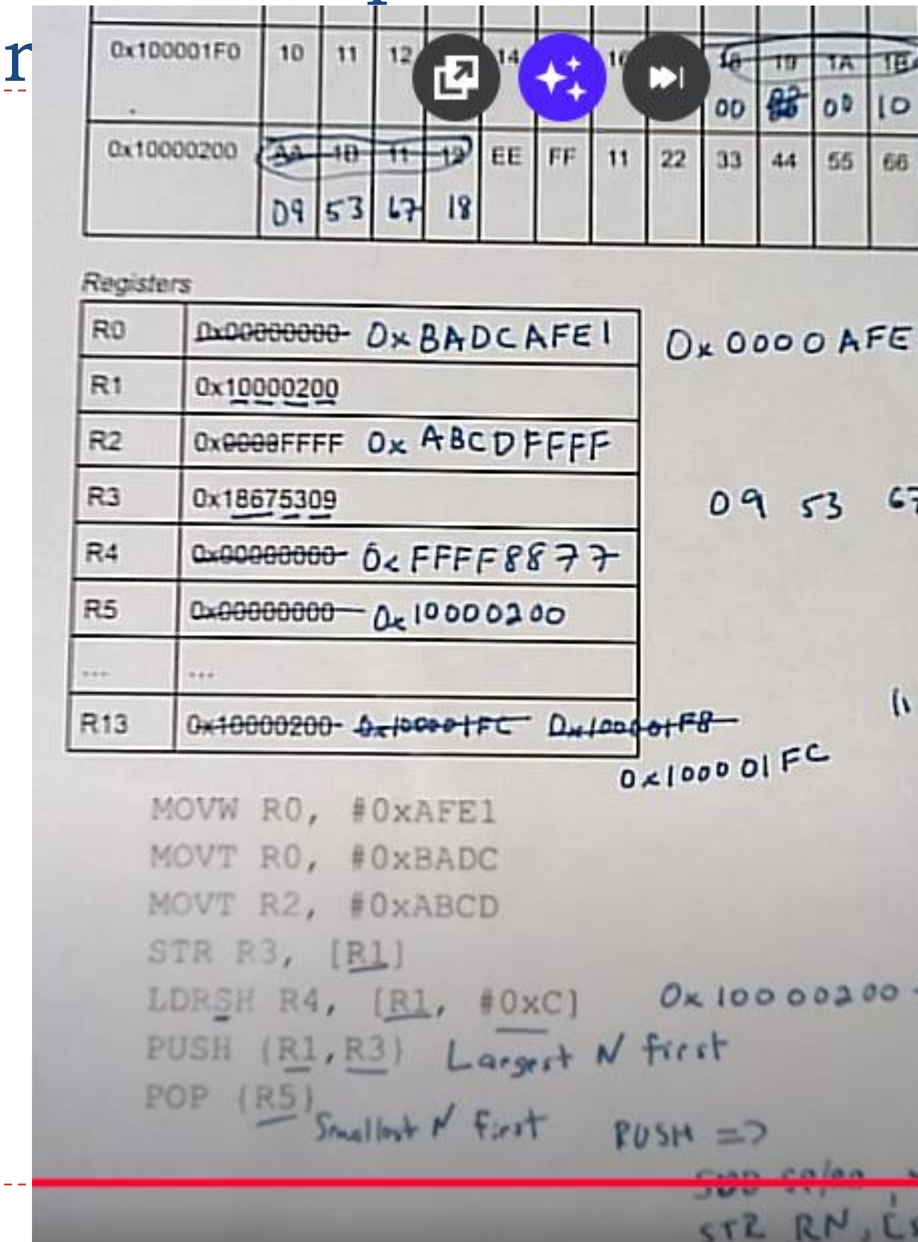
0F	1F	AA
0E	1E	99
0D	1D	88
0C	1C	77
0B	1B	66
0A	1A	55
09	19	44
08	18	33
07	17	22
06	16	11
05	15	FF
04	14	EE
03	13	12
02	12	11
01	11	1B
00	10	AA
0x100001E0	0x100001F0	0x10000200
Mem Address	Mem Address	Mem Address

- MOVW R0, #0xAFE1
- MOVT R0, #0xBADC
- MOVT R2, #0xABCD
- STR R3, [R1]
- LDRSH R4, (R1, #0xC)
- PUSH (R1, R3)
- POP (R5)

R0	0x00000000
R1	0x10000200
R2	0x0000FFFF
R3	0x18675309
R4	0x00000000
R5	0x00000000
…	
R13	0x10000200

Compute register values at each step of this pr

- After
  - MOVW R0, #0xAFE1
  - MOVT R0, #0xBADC
- We have R0=0xBADCAFE1
- After MOVT R2, #0xABCD
- We have R2=
- STR R3, [R1]
- LDRSH R4, (R1, #0xC)
- PUSH (R1, R3)
- POP (R5)



- MOVW R0, #0xAFE1
- MOVT R0, #0xBADC
- MOVT R2, #0xABCD
- STR R3, [R1]
- LDRSH R4, (R1, #0xC)
- PUSH (R1, R3)
- POP (R5)