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

Chapter 5
Memory Access Exercises ANS

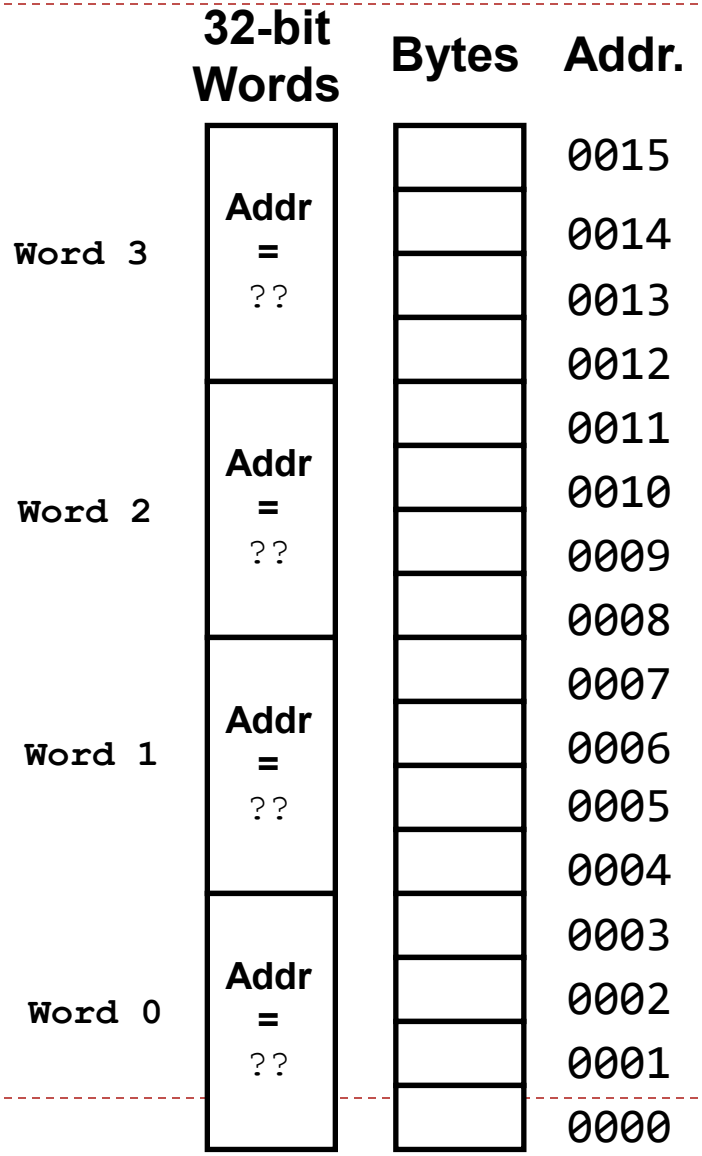
Zonghua Gu

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/

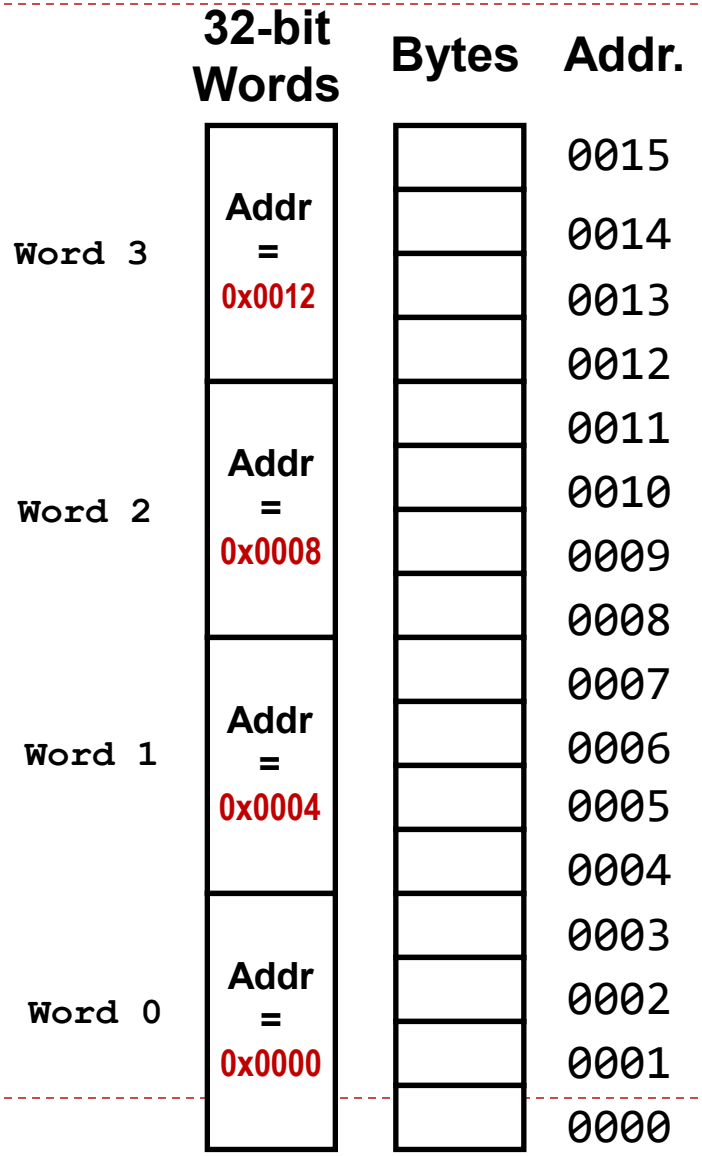
Endianness

What are the memory address of these four words?



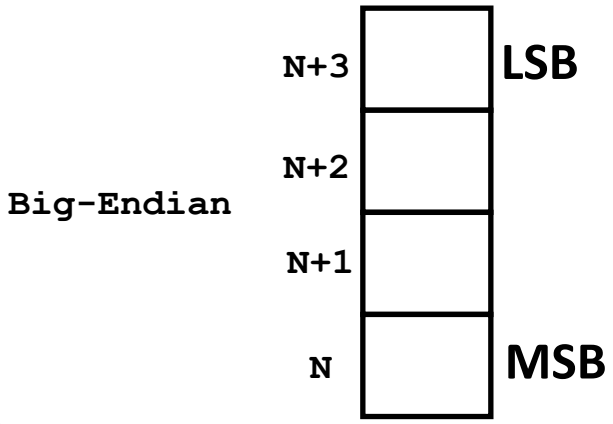
Endianness ANS

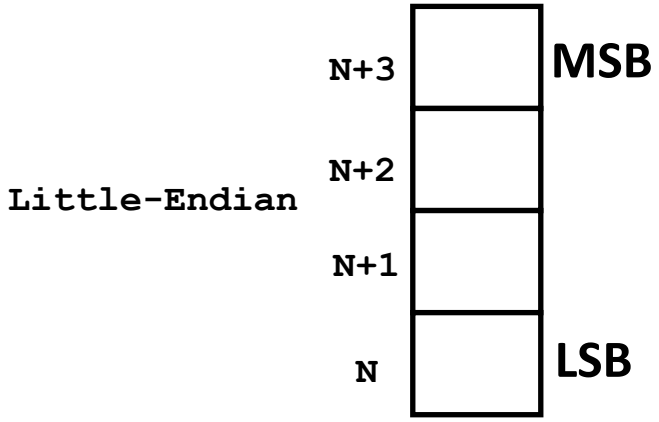
What are the memory address of these four words?
Same as the address of the lowest-address Byte
(this is true **for either Little-Endian or Big-Endian ordering**)



Endianness

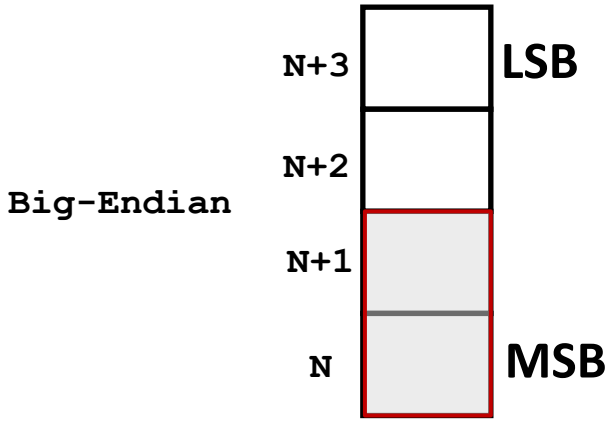
- Q: Assume Big-Endian ordering. If a 32-bit word resides at memory address N, what is the address of:
  - (a) The MSB (Most Significant Byte)
  - (b) The 16-bit half-word corresponding to the most significant half of the word
- Q: Redo the question assuming Little-Endian ordering.

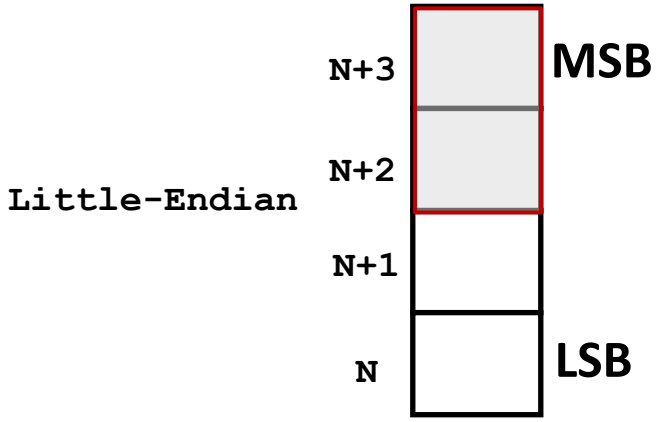




Endianness ANS

- 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)





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

Endianness ANS

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

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 store:
  - Memory content after store:
- **STR r1, [r0, #4]!**
  - Assume r1 = 0x76543210
  - r0 after store:
  - 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 store: 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 store: 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**

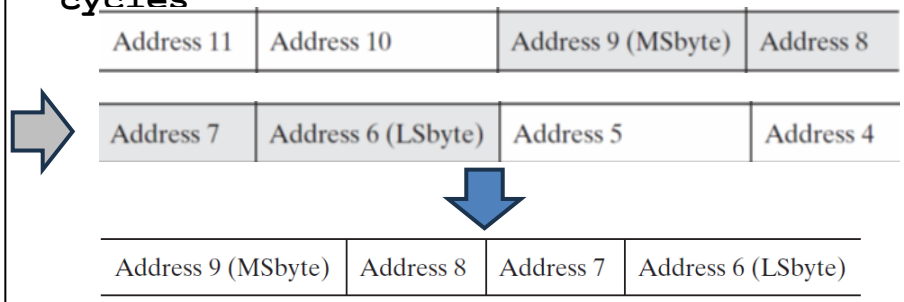
Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9	Address 8
Address 7 (MSbyte)	Address 6	Address 5	Address 4 (LSbyte)
Address 3	Address 2	Address 1	Address 0

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:



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

Data Alignment ANS

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.

Data Alignment

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?

Data Alignment ANS

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?
  - (c) How many memory cycles are required to read the word at address 102?
  - (d) 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

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

Memory Cycles ANS

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 least 2 consecutive words, and at most 3 consecutive words in memory)

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
  - (d) longs

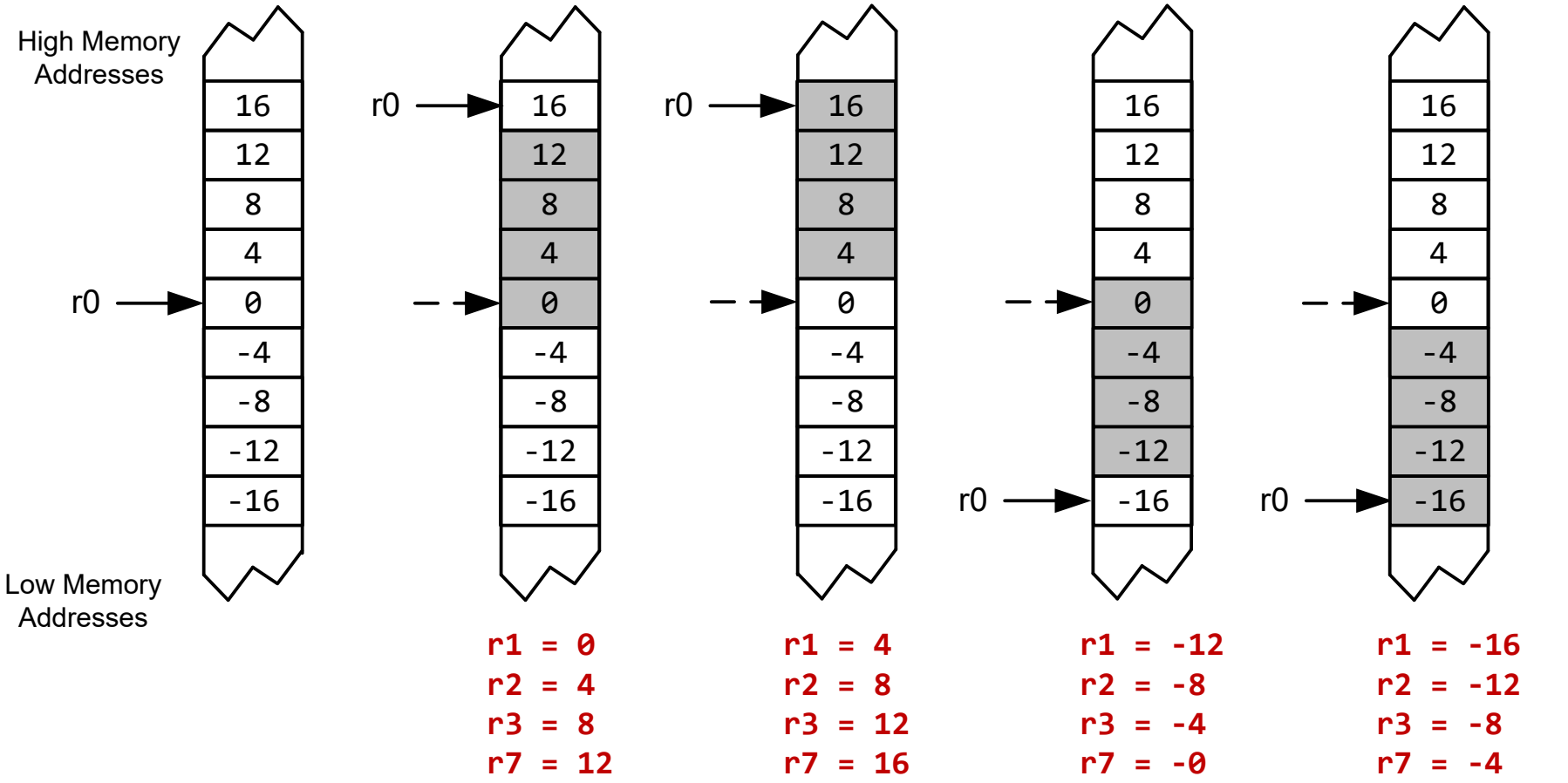
Arrays ANS

- 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
  - (d) 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
  - (d) longs: 0x12345678+8 = 0x12345680

Review

Load Multiple Registers

LDMxx r0!, {r3,r1,r7,r2}



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. (Note that r3 is incremented by 4 (1 word) each time, not 1 byte)

LDR

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

LDRB R7, [R2, #1] ANS

- After STR R5, [R2, #0] (same as STR R5, [R2]) and LDRB R7, [R2, #1] (detailed explanations not needed for exam):
  - Memory address R2 contains 0x23456789. 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
	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, #2] ANS

- ANS after LDRSH R7, [R2, #2]:
  - 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, #2] 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, #2] 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

Program Understanding 1

- Compute register and memory values at each step of this program, given initial register values and memory contents, assuming little-endian ordering. Memory addresses and contents are shown in the table. (Assume memory addresses increase from left to right, and from bottom to top in the table.)

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

Initial Register Values

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

Initial Memory Contents

0x10000200	60	1B	11	12	EE	FF	11	22	33	44	55	66	77	88	99	92
0x100001F0	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0x100001E0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

Program Understanding 1 ANS

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

- After MOVW R0, #0xAFE1, MOVT R0, #0xBADC
- R0 = 0xBADCAFE1
- After MOVT R2, #0xABCD
  - R2 = 0xABCDFFFF
- After STR R3, [R1]
  - R3 = 0x18675309 is stored at mem address R1 = 0x10000200
- After LDRSH R4, [R1, #0xC] (load signed half word from R1+0xC=0x1000020C)
  - R4 = 0xFFFF8877

Initial Register Values

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

0x10000200	09	53	67	18	EE	FF	11	22	33	44	55	66	77	88	99	92
0x100001F0	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0x100001E0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

Initial Memory Contents

Program Understanding 2

- Show all updates to registers as the assembly code shown below runs, assuming little-endian ordering. Memory addresses and contents are shown in the table. (Assume memory addresses increase from left to right, and from bottom to top in the table.) Show the NZCV flags next to the instruction if they change. Each instruction is 32 bits. Do not update PC after each instruction. R13 and R15 have initial values shown.

```
     LDR R1, =0x10000010
     LDR R2, [R1]
     LDR R3, [R1,#4]
     BL max
     SUBS R4, R2, R0  @NZCV =
     MOVW R5, #1
     LSL R6, R5, #4
     BIC R7, R2, R6
     ANDS R8, R7, R6  @NZCV =
     ROR R9, R3, #12
     REV R10, R9
     RBIT R11, R10
     ADDS R12, R3, R9 @NZCV =
     STR R12, [R1,#8]

loop B loop
     ENDP

max PROC
     CMP R2, R3       @NZCV =
     BLT second
first MOV R0, R2
      B done
second MOV R0, R3
done   BX LR
     ENDP
```

R0		R4		R8		R12	
R1		R5		R9		R13	0x100 00200
R2		R6		R10		R14	
R3		R7		R11		R15	0x000 00250

0x10000010	FF	EF	CD	AB	00	00	CD	AB	00	00	00	00

Program Understanding 2 ANS

- Step-by-step execution shown in next slide.

```
    LDR R1, =0x10000010
    LDR R2, [R1]
    LDR R3, [R1,#4]
    BL max
    SUBS R4, R2, R0  @NZCV = 0110
    MOVW R5, #1
    LSL R6, R5, #4
    BIC R7, R2, R6
    ANDS R8, R7, R6  @NZCV = 0110
    ROR R9, R3, #12
    REV R10, R9
    RBIT R11, R10
    ADDS R12, R3, R9 @NZCV = 1000
    STR R12, [R1,#8]

loop B loop
     ENDP

max PROC
     CMP R2, R3      @NZCV = 0010
     BLT second
first MOV R0, R2
```

R0	0xABCDEFFF	R4	0x00000000	R8	0x00000000	R12	0xABD7BCD0
R1	0x10000010	R5	0x00000001	R9	0x000ABCD0	R13	0x10000200
R2	0xABCDEFFF	R6	0x00000010	R10	0xD0BC0A00	R14	0x00000260
R3	0xABCD0000	R7	0xABCDEFEF	R11	0x00503D0B	R15	omitted

0x10000010	FF	EF	CD	AB	00	00	CD	AB	D0	BC	D7	AB

- LDR R1, =0x10000010 → R1 = 0x10000010 (literal load of the base address).
- LDR R2, [R1] → reads bytes FF EF CD AB at 0x10000010..13 and forms R2 = 0xABCDEFFF (little-endian).
- LDR R3, [R1,#4] → reads bytes 00 00 CD AB at 0x10000014..17 and forms R3 = 0xABCD0000 (little-endian).
- BL max → in max: CMP R2, R3 sets NZCV = 0010 since R2 > R3 (N=0, Z=0, C=1,V=0), path takes first: MOV R0, R2, then BX LR returns with R0 = 0xABCDEFFF.
- SUBS R4, R2, R0 → R4 = R2 − R0 = 0, NZCV = 0110 (N=0, Z=1, C=1,V=0) because subtracting equal values yields zero with carry set.
- MOVW R5, #1 → R5 = 0x00000001 (loads 16-bit immediate into low halfword).
- LSL R6, R5, #4 → R6 = 0x00000010 (1 shifted left by 4).
- BIC R7, R2, R6 → R7 = R2 & ~R6 = 0xABCDEFFF & 0xFFFFFFEF = 0xABCDEFEF (clears bit 4).
- ANDS R8, R7, R6 → R8 = 0xABCDEFEF & 0x10 = 0x00000000 with NZCV = 0110 (zero result with carry preserved from previous SUBS instruction).
- ROR R9, R3, #12 → rotates 0xABCD0000 right by 12 bits to R9 = 0x000ABCD0.
- REV R10, R9 → byte-reverse 00 0A BC D0 into D0 BC 0A 00, giving R10 = 0xD0BC0A00.
- RBIT R11, R10 → bit-reverse all 32 bits of 0xD0BC0A00, yielding R11 = 0x00503D0B (per architectural bit-reverse).
- ADDS R12, R3, R9 → R12 = 0xABCD0000 + 0x000ABCD0 = 0xABD7BCD0 with NZCV = 1000 (negative due to high bit, non-zero, no carry, no overflow).
- STR R12, [R1,#8] → stores 0xABD7BCD0 at 0x10000018 as bytes D0 BC D7 AB in little-endian.

R0	0xABCDEFFF	R4	0x00000000	R8	0x00000000	R12	0xABD7BCD0
R1	0x10000010	R5	0x00000001	R9	0x000ABCD0	R13	0x10000200
R2	0xABCDEFFF	R6	0x00000010	R10	0xD0BC0A00	R14	0x00000260
R3	0xABCD0000	R7	0xABCDEFEF	R11	0x00503D0B	R15	omitted

0x10000010	FF	EF	CD	AB	00	00	CD	AB	D0	BC	D7	AB

Notes on Flags

- CMP R2,R3 → NZCV = 0010 because R2 > R3 in unsigned comparison semantics used by CMP on 32-bit registers without carry-in.
- SUBS R4,R2,R0 → NZCV = 0110 because the result is zero and subtraction of equal values sets carry and clears negative and overflow.
- ANDS R8,R7,R6 → NZCV = 0110 because the logical AND produced zero, and the carry is preserved from SUBS.
- ADDS R12,R3,R9 → NZCV = 1000 because the sum has the top bit set (negative in signed sense), is non-zero, and does not generate carry or overflow for these operands.

R0	0xABCDEFFF	R4	0x00000000	R8	0x00000000	R12	0xABD7BCD0
R1	0x10000010	R5	0x00000001	R9	0x000ABCD0	R13	0x10000200
R2	0xABCDEFFF	R6	0x00000010	R10	0xD0BC0A00	R14	0x00000260
R3	0xABCD0000	R7	0xABCDEFEF	R11	0x00503D0B	R15	omitted

0x10000010	FF	EF	CD	AB	00	00	CD	AB	D0	BC	D7	AB

R13, R14, R15

- R13 is the stack pointer (SP), R14 is the link register (LR), and R15 is the program counter (PC). Table shows
- R13 (SP): points to the current top of the stack used for pushes, pops, and call frames; here it's initialized to 0x10000200 and remains the same since there is no stack operation (PUSH, POP).
- R15 (PC): Initially, PC = 0x00000250 at the first instruction "LDR R1, =0x10000010". It gets incremented by 4 after execution of each instruction without branch, hence PC = 0x0000025C at instruction "BL max". When the program finishes, PC is at the last instruction "loop B loop", and shown as "omitted" in the table.
- R14 (LR): holds the return address set by BL (address of the instruction after the branch), which is 0x0000025C + 4 = 0x00000260.

0x00000250
0x0000025C
0x00000260

```
    LDR R1, =0x10000010
    LDR R2, [R1]
    LDR R3, [R1,#4]
    BL max
    SUBS R4, R2, R0  @NZCV =
    MOVW R5, #1
    LSL R6, R5, #4
    BIC R7, R2, R6
    ANDS R8, R7, R6  @NZCV =
    ROR R9, R3, #12
    REV R10, R9
    RBIT R11, R10
    ADDS R12, R3, R9 @NZCV =
    STR R12, [R1,#8]

loop B loop
     ENDP

max PROC
     CMP R2, R3       @NZCV =
     BLT second
first MOV R0, R2
```

R0	0xABCDEFFF	R4	0x00000000	R8	0x00000000	R12	0xABD7BCD0
R1	0x10000010	R5	0x00000001	R9	0x000ABCD0	R13	0x10000200
R2	0xABCDEFFF	R6	0x00000010	R10	0xD0BC0A00	R14	0x00000260
R3	0xABCD0000	R7	0xABCDEFEF	R11	0x00503D0B	R15	omitted

0x10000010	FF	EF	CD	AB	00	00	CD	AB	D0	BC	D7	AB

6. *Assembly*

(a) Show all updates to registers as the assembly code shown below runs. Show the NZCV flags next to the instruction if they change. Do not update PC each time. You can assume that each instruction is 32 bits.

```
0x00000250      LDR R1, =0x10000010
0x00000254      LDR R2, [R1]
0x00000258      LDR R3, [R1,#4]
0x0000025C      BL max
0x00000260      SUBS R4, R2, R0     NZCV = 0110
                MOVW R5, #1
                LSL R6, R5, #4
                BIC R7, R2, R6
                ANDS R8, R7, R6     NZCV = 0110
                ROR R9, R3, #12
                REV R10, R9
                RBIT R11, R10
                ADDS R12, R3, R9    NZCV = 1000
                STR R12, [R1,#8]
loop B loop
                ENDP

max   PROC
                CMP R2, R3          NZCV = 0010
                BLT second
first           MOV R0, R2
                B done
second          MOV R0, R3
done BX LR
                ENDP
```

EC361 2020 Exam 2 Q1
https://www.youtube.com/watch?v=XkjC1avu6Ow

R0	0xABCDEFFF	R4	0x00000000	R8	0x00000000	R12	0xABD7BCD0
R1	0x10000010	R5	0x00000001	R9	0xD0ABCD0	R13	0x10000200
R2	0xABCDEFFF	R6	0x00000010	R10	0xD0BCDA00	R14	0x00000260
R3	0xABCD0000	R7	0xABCDEFEF	R11	0x0050300B	R15	0x00000250

0x10000010	FF	EF	CD	AB	00	00	CD	AB	D0	BC	D7	A8