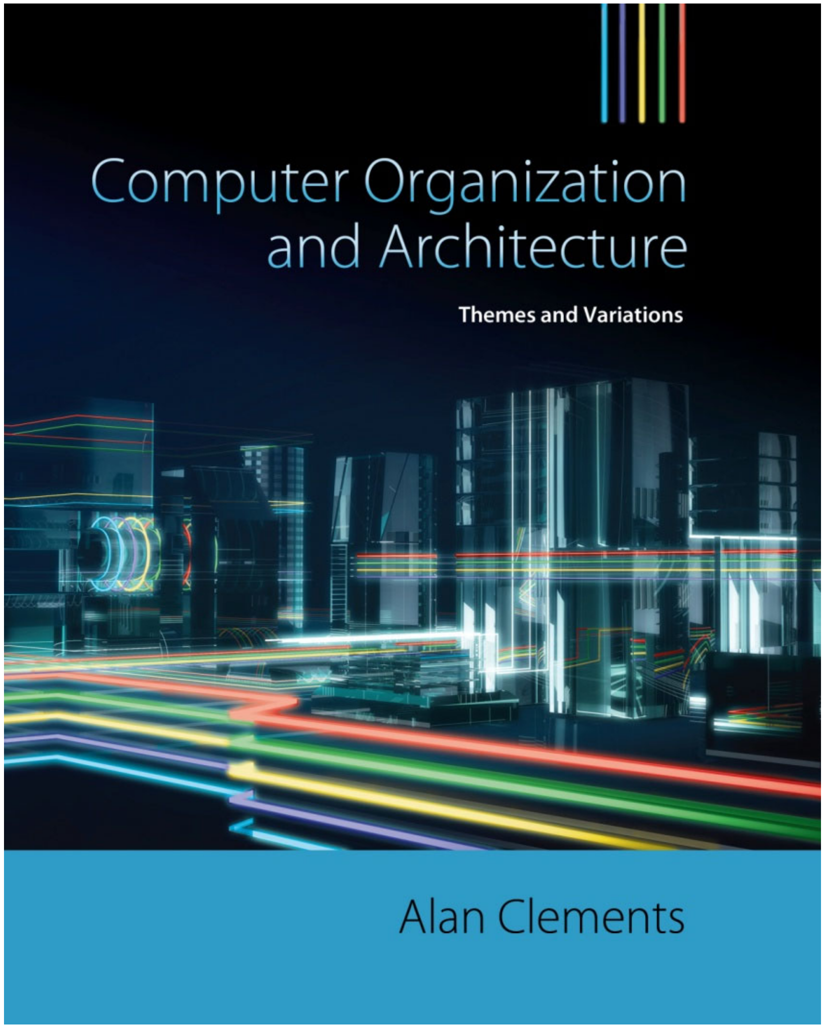
Part 9

CHAPTER 3

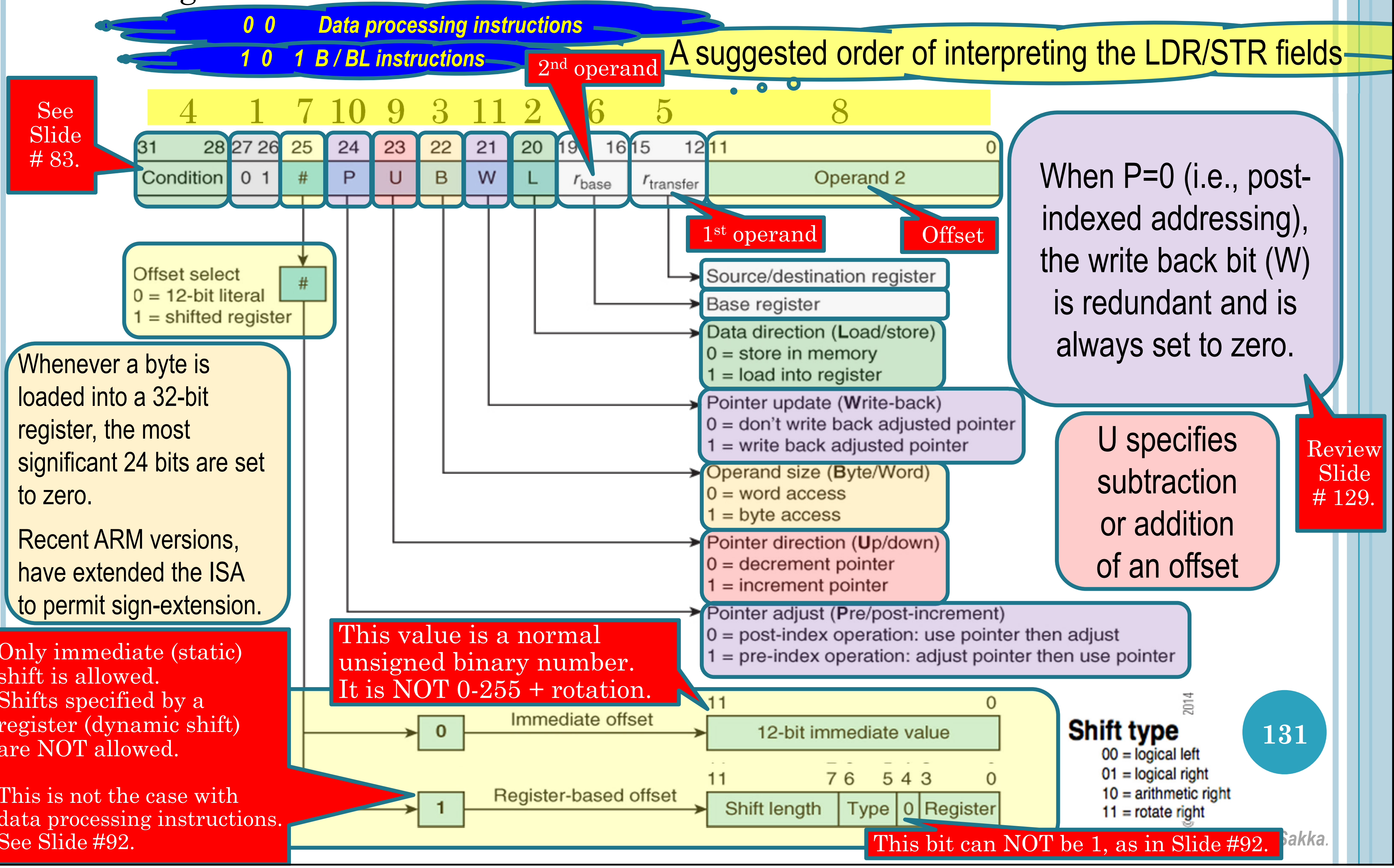
Architecture and Organization



CENGAGE Learning

ARM Load and Store Encoding

- The figure illustrate the format of the ARM's load and store instructions.



0 0 Data processing instructions

1 0 1 B / BL instructions

2nd operand

A suggested order of interpreting the LDR/STR fields

4 1 7 10 9 3 11 2 6 5 8

See Slide # 83.

31 28	27 26	25	24	23	22	21	20	19 16	15 12	11 0
Condition	0 1	#	P	U	B	W	L	r_{base}	$r_{transfer}$	Operand 2

1st operand

Offset

Offset select
0 = 12-bit literal
1 = shifted register — #

Source/destination register

Base register

Data direction (Load/store)
0 = store in memory
1 = load into register

Pointer update (Write-back)
0 = don't write back adjusted pointer
1 = write back adjusted pointer

Operand size (Byte/Word)
0 = word access
1 = byte access

Pointer direction (Up/down)
0 = decrement pointer
1 = increment pointer

Pointer adjust (Pre/post-increment)
0 = post-index operation: use pointer then adjust
1 = pre-index operation: adjust pointer then use pointer

When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

Review Slide # 129.

U specifies subtraction or addition of an offset

Whenever a byte is loaded into a 32-bit register, the most significant 24 bits are set to zero.

Recent ARM versions, have extended the ISA to permit sign-extension.

This value is a normal unsigned binary number. It is NOT 0-255 + rotation.

0 — Immediate offset — 11 0: 12-bit immediate value

1 — Register-based offset — 11 7 6 5 4 3 0: Shift length | Type | 0 | Register

Only immediate (static) shift is allowed.
Shifts specified by a register (dynamic shift) are NOT allowed.

This is not the case with data processing instructions. See Slide #92.

Shift type
00 = logical left
01 = logical right
10 = arithmetic right
11 = rotate right

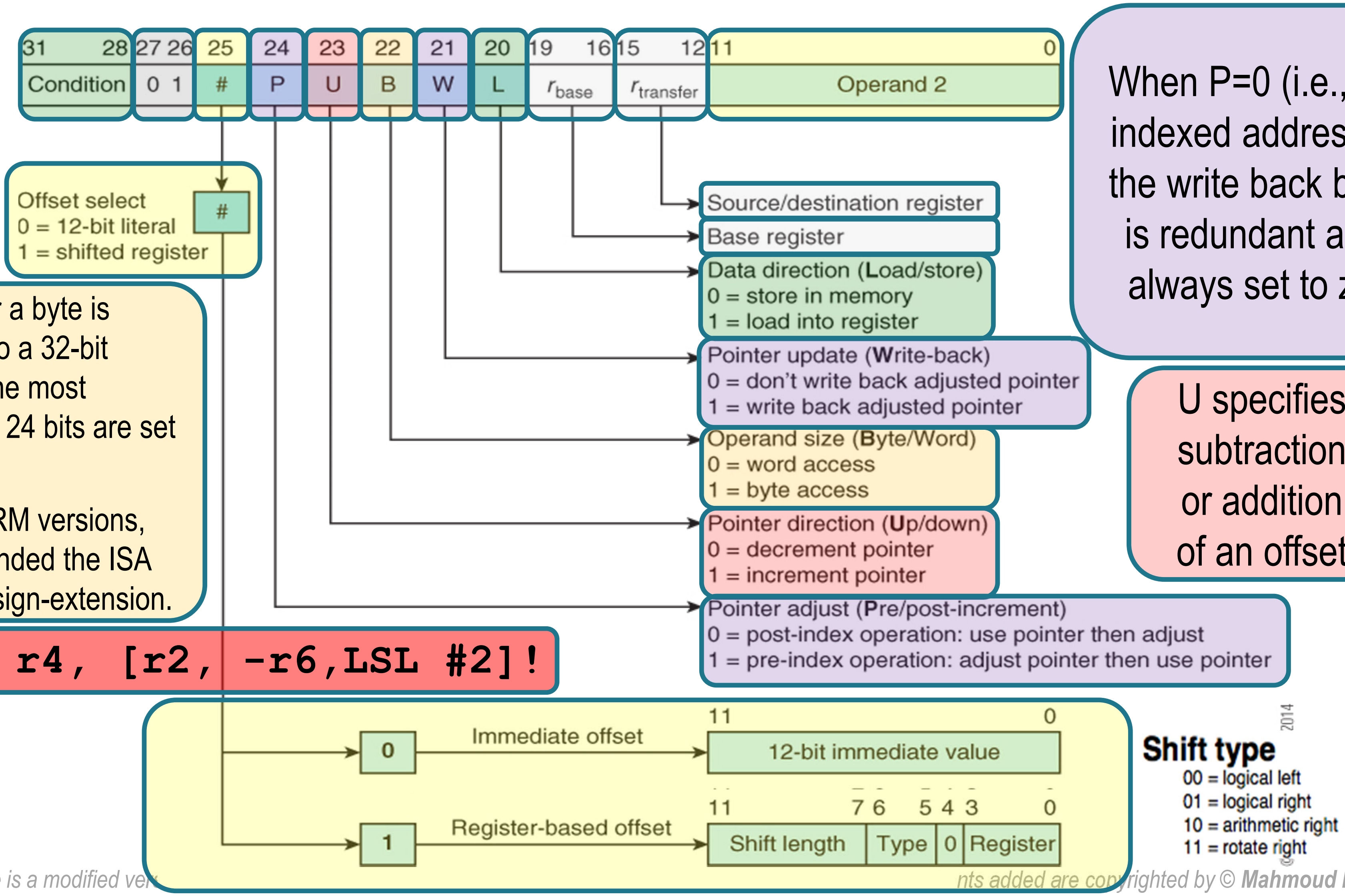
2014

This bit can NOT be 1, as in Slide #92.

ARM Load and Store Encoding

- Decode the following ARM machine code instruction 0x57224106

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	1	0	1	0	1	1	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0



31 28	27 26	25	24	23	22	21	20	19 16	15 12	11 0
Condition	0 1	#	P	U	B	W	L	r_{base}	$r_{transfer}$	Operand 2

Offset select #
0 = 12-bit literal
1 = shifted register

Source/destination register

Base register

Data direction (**L**oad/store)
0 = store in memory
1 = load into register

Pointer update (**W**rite-back)
0 = don't write back adjusted pointer
1 = write back adjusted pointer

Operand size (**B**yte/Word)
0 = word access
1 = byte access

Pointer direction (**U**p/down)
0 = decrement pointer
1 = increment pointer

Pointer adjust (**P**re/post-increment)
0 = post-index operation: use pointer then adjust
1 = pre-index operation: adjust pointer then use pointer

Shift type
00 = logical left
01 = logical right
10 = arithmetic right
11 = rotate right

When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

U specifies subtraction or addition of an offset

Whenever a byte is loaded into a 32-bit register, the most significant 24 bits are set to zero.

Recent ARM versions, have extended the ISA to permit sign-extension.

```
STRPL r4, [r2, -r6,LSL #2]!
```

ARM Load and Store Encoding

Decoding the ARM Instruction **STRPL r4,[r2,-r6,LSL#2]!**

Field Name	Value	Action	Interpretation
Condition	0101	PL	Execute on positive
OP-code	01		Defines load/store instruction
#	1	Operand 2 format	Operand is a shifted register
P	1	Pre/post adjust	Adjust pointer before using
U	0	Pointer direction	Decrement pointer
B	0	Byte/word	This is a word access
W	1	Pointer write back	Update pointer after use
L	0	Load/store	Store data in memory
r_{base}	0010	Base register	r2 is the base (pointer) register
$r_{transfer}$	0100	Source/destination	r4 is the source in this store instruction
Shift length	00010	Shift length	Shift the register 2 places
Shift type	00	Logical shift left	Logical shift left the offset in r6
Op-code	0		
Shift register	0110	Specified register to be shifted	r6 is shifted twice

Operand 2

STRPL r4,[r2,-r6,LSL#2]!

0 1 0 1 | 0 1 1 1 | 0 0 1 0 | 0 0 1 0 | 0 1 0 0 | 0 0 0 1 | 0 0 0 0 | 0 1 1 0 |

0 x E 6 D 2 1 2 4 3

1 1 1 0 | 0 1 | 1 0 1 1 0 1 | 0 0 1 0 | 0 0 0 1 | 0 0 1 0 0 | 1 0 | 0 | 0 0 1 1

AL   S/T   reg base   post line byte   no write back   LDR   R2   R1   #4   ASR   R3.

LDRB R1, R2, ASR R3, #4

LDRB R1, [R2], R3, ASR#4..

↑

post-index

[R2, R3, ASR#4]

pre-index.

STRGT R1, [R2, #-0xFFF].

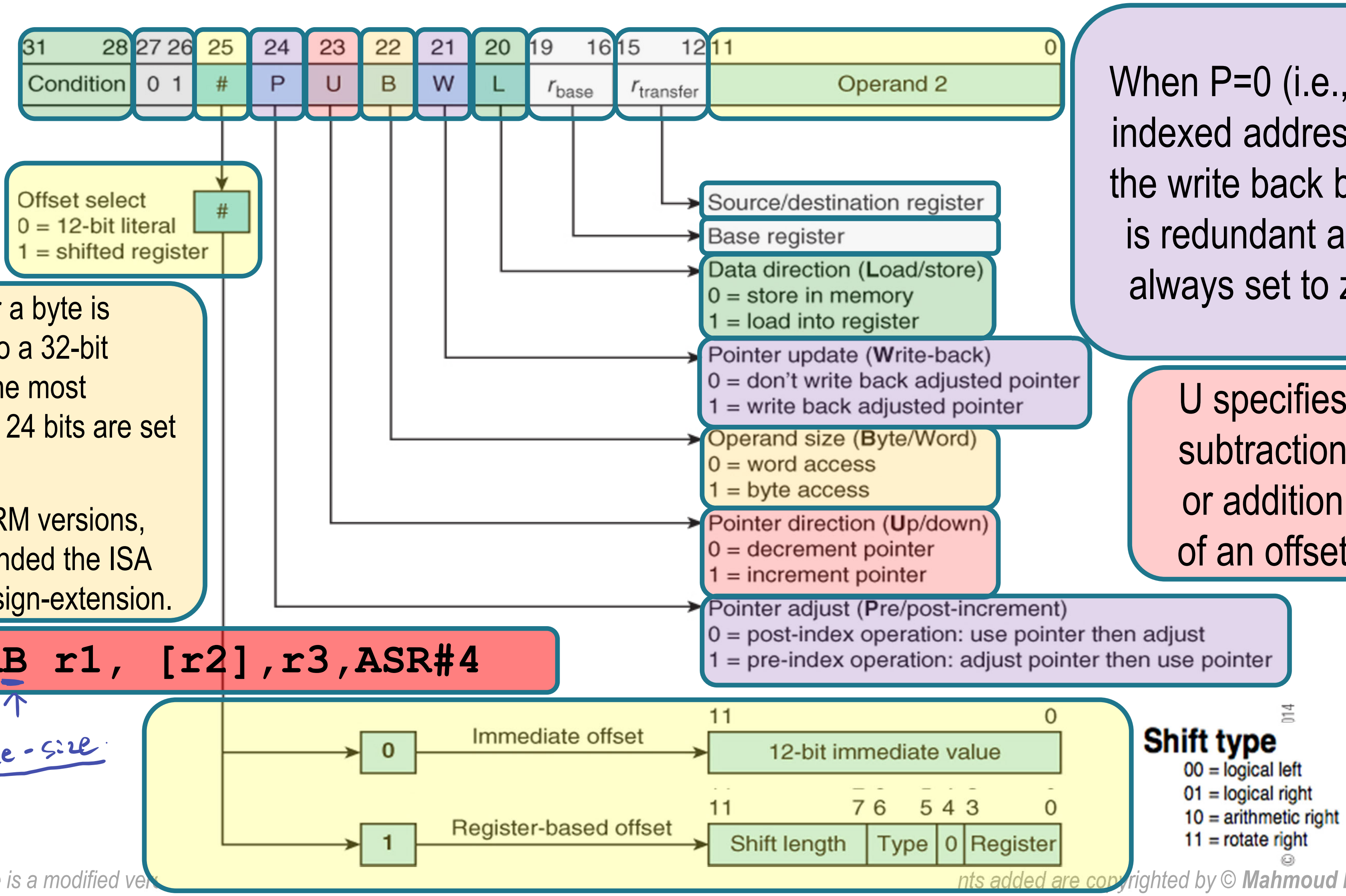
1 1 0 0 | 0 1 0 1 | 0 0 0 0 | 0 0 1 0 | 0 0 0 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1

C 5 0 2 1 F F F.

ARM Load and Store Encoding

- Decode the following **ARM** machine code instruction **0xE6D21243**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1



When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

U specifies subtraction or addition of an offset

Whenever a byte is loaded into a 32-bit register, the most significant 24 bits are set to zero.

Recent ARM versions, have extended the ISA to permit sign-extension.

`LDRB r1, [r2],r3,ASR#4`

byte - size

ARM Load and Store Encoding

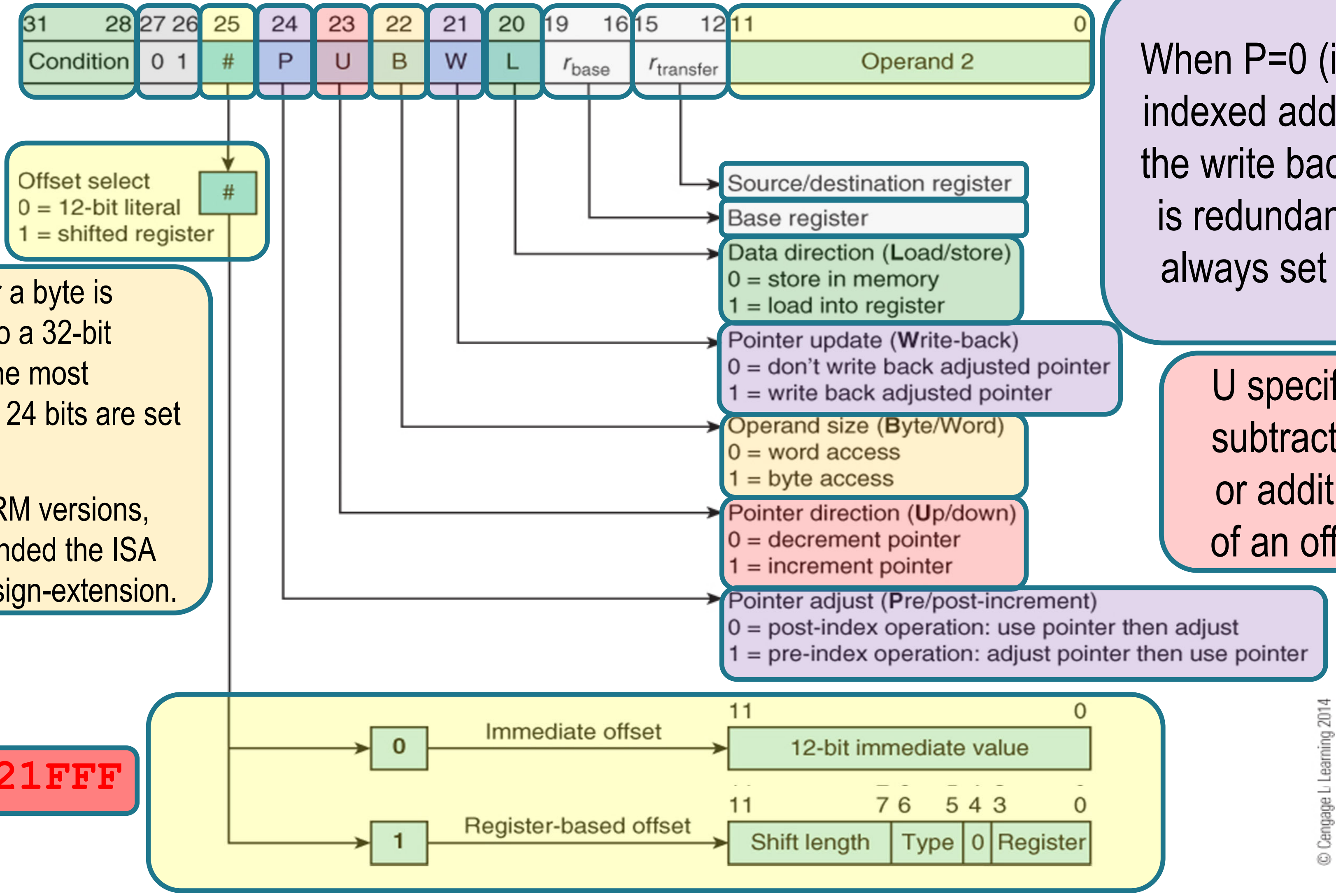
Decoding the ARM Instruction**LDR r1, [r2],r3,ASR#4**

Field Name	Value	Action	Interpretation
Condition	1110	AL	Always (default)
OP-code	01		Defines load/store instruction
#	1	Operand 2 format	Operand is a shifted register
P	0	Pre/post adjust	Adjust pointer after using
U	1	Pointer direction	Increment pointer
B	0	Byte/word	This is a word access
W	0	Pointer write back	As P=0, W is redundant and always=0
L	1	Load/store	Load data from memory
r_{base}	0010	Base register	r2 is the base (pointer)register
$r_{transfer}$	0001	Source/destination	r1 is the destination in this loadinstruction
Shift length	00100	Shift length	Shift the register 4places
Shift type	10	Arithmetic shift right	Arithmetic shift right the offset in r3
Op-code	0		
Shift register	0011	Specified register to be shifted	r3 is shifted four times

Operand 2

ARM Load and Store Encoding

- ❑ Encode the following **ARM** instruction **STRGT r1,[r2,#-0xFFF]**



When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

U specifies subtraction or addition of an offset

Whenever a byte is loaded into a 32-bit register, the most significant 24 bits are set to zero.

Recent ARM versions, have extended the ISA to permit sign-extension.

0xC5021FFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

ARM Load and Store Encoding

Decoding the ARM Instruction **STRGT r1,[r2,#-0xFFF]**

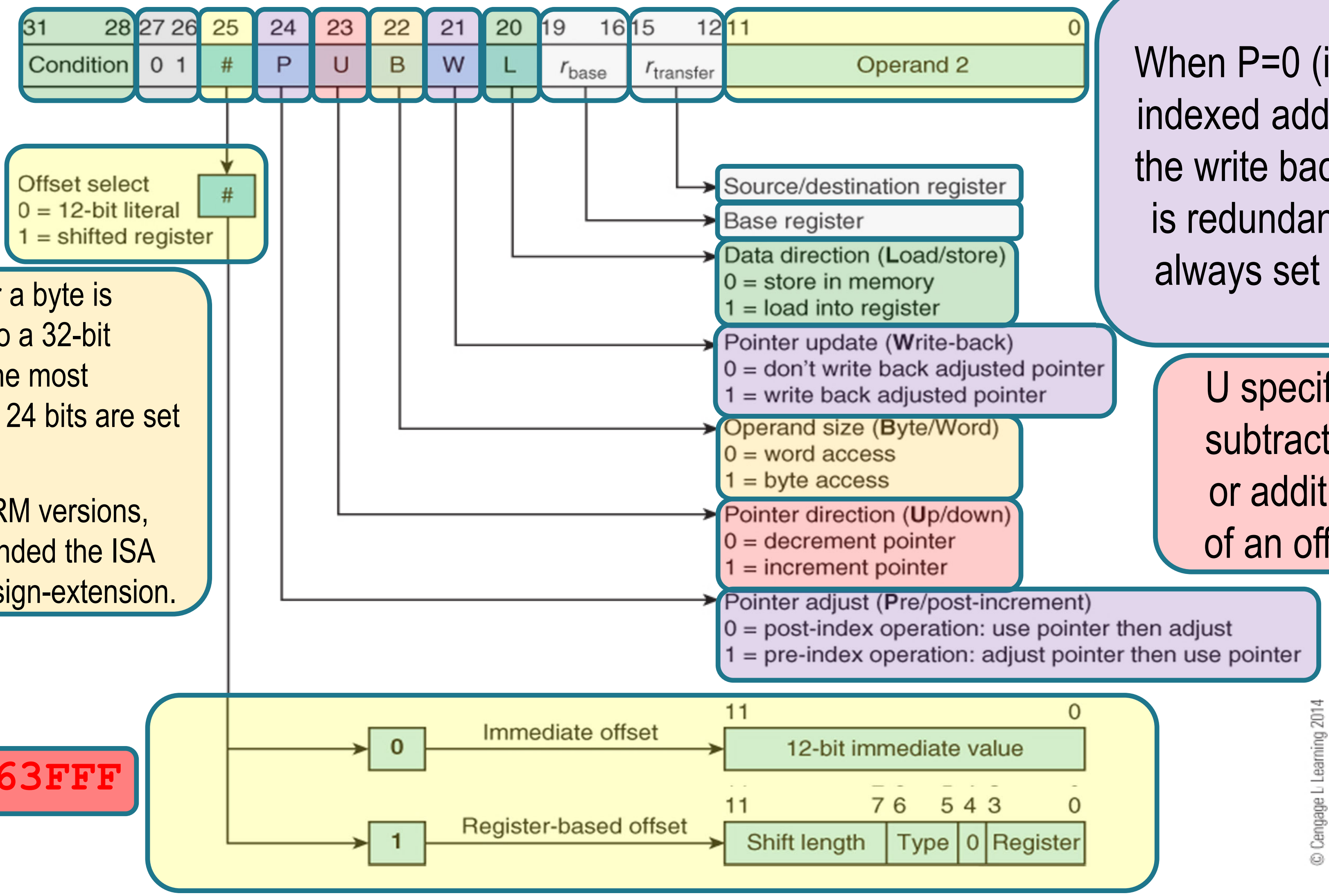
Field Name	Value	Action	Interpretation
Condition	1100	GT	Execute on greater than
OP-code	01		Defines load/store instruction
#	0	Operand 2 format	Operand is immediate
P	1	Pre/post adjust	Adjust pointer before using
U	0	Pointer direction	Decrement pointer
B	0	Byte/word	This is a word access
W	0	Pointer write back	Do not write back the adjusted pointer
L	0	Load/store	Store data in memory
r_{base}	0010	Base register	r2 is the base (pointer) register
$r_{transfer}$	0001	Source/destination	r1 is the source in this store instruction
Immediate offset	111111111111	Shift length	Offset value = 0xFFF

C 5 0 2 1 F F F

1100 01 0 1 0 0 0 0 0010 0001 111111111111

ARM Load and Store Encoding

- Encode the following **ARM** instruction **LDREQ r3,[r6],#-0xFFF**



Offset select
0 = 12-bit literal
1 = shifted register

Source/destination register

Base register

Data direction (Load/store)
0 = store in memory
1 = load into register

Pointer update (Write-back)
0 = don't write back adjusted pointer
1 = write back adjusted pointer

Operand size (Byte/Word)
0 = word access
1 = byte access

Pointer direction (Up/down)
0 = decrement pointer
1 = increment pointer

Pointer adjust (Pre/post-increment)
0 = post-index operation: use pointer then adjust
1 = pre-index operation: adjust pointer then use pointer

© Cengage L Learning 2014

When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

U specifies subtraction or addition of an offset

Whenever a byte is loaded into a 32-bit register, the most significant 24 bits are set to zero.

Recent ARM versions, have extended the ISA to permit sign-extension.

0x04163FFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

ARM Load and Store Encoding

Decoding the ARM Instruction **LDREQ r3,[r6],#-0xFFF**

Field Name	Value	Action	Interpretation
Condition	0000	EQ	Execute on equal
OP-code	01		Defines load/store instruction
#	0	Operand 2 format	Operand is immediate
P	0	Pre/post adjust	Adjust pointer after using
U	0	Pointer direction	Decrement pointer
B	0	Byte/word	This is a word access
W	0	Pointer write back	Write back adjusted pointer
L	1	Load/store	Load data from memory
r_{base}	0110	Base register	r6 is the base (pointer) register
$r_{transfer}$	0011	Source/destination	r3 is the destination in this load instruction
Immediate offset	111111111111	Shift length	Offset value = 0xFFF

ARM Load and Store Encoding

□ Encode the following **ARM** instructions.

Run on uVision later.

```
LDR R1,[R2]           => LSL #0.
LDR R1,[R2],#0          P=0  w=0
LDR R1,[R2,#0]          P=1  w=0
LDR R1,[R2,#0]!         P=1  w=1

STR R1,[R2]
STR R1,[R2],#0
STR R1,[R2,#0]
STR R1,[R2,#0]!
```

□ Is there any *effective* difference between the 4 LDR instructions?

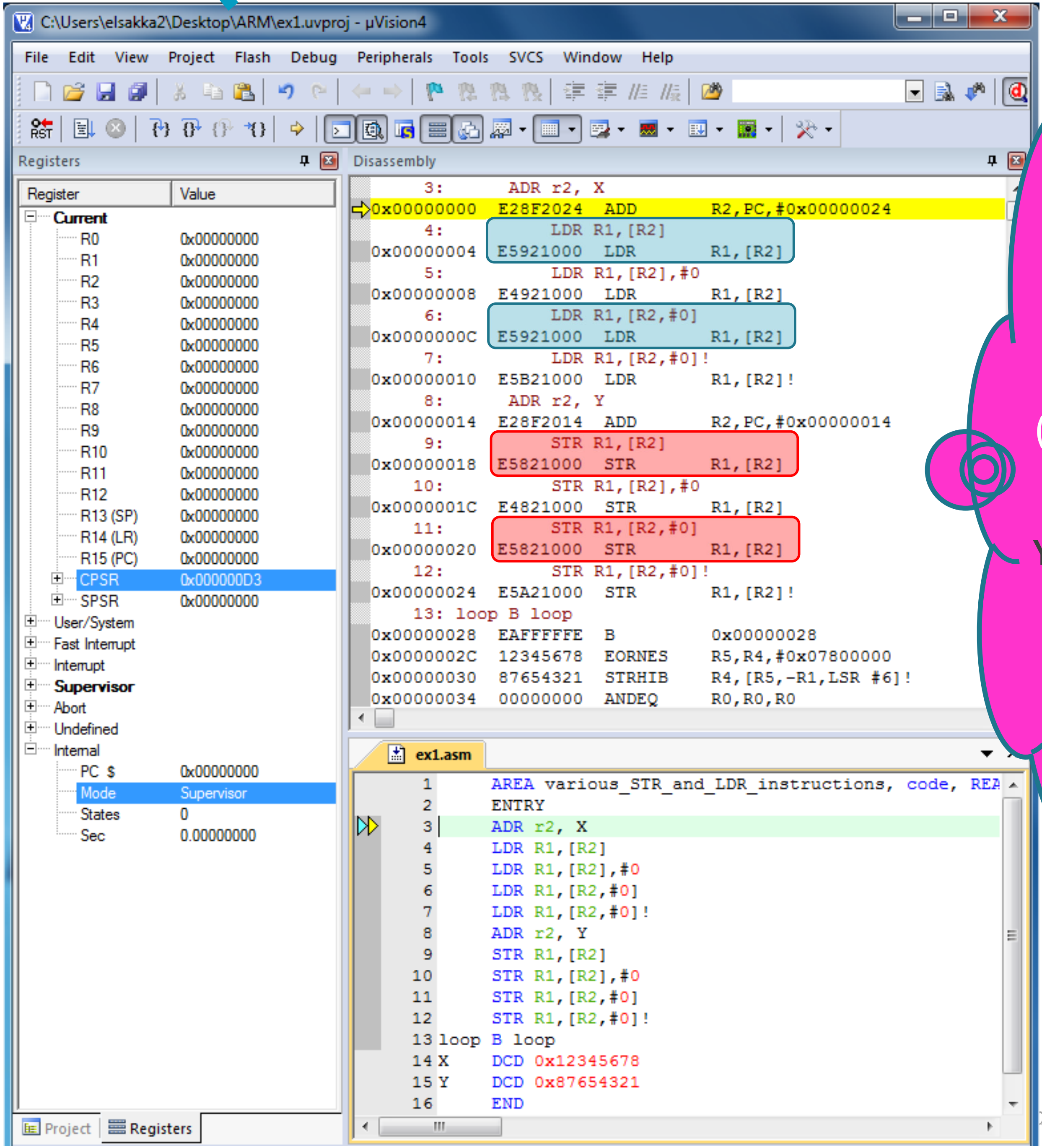
□ Is there any *effective* difference between the 4 STR instructions?

ARM Load and Store Encoding

```
     AREA various_STR_and_LDR_instructions, code, READONLY
     ENTRY
     ADR r2, X
     LDR R1,[R2]
     LDR R1,[R2],#0
     LDR R1,[R2,#0]
     LDR R1,[R2,#0]!
     ADR r2, Y
     STR R1,[R2]
     STR R1,[R2],#0
     STR R1,[R2,#0]
     STR R1,[R2,#0]!
loop B loop
X    DCD 0x12345678
Y    DCD 0x87654321
     END
```

31–28	27 26	25	24	23	22	21	20	19–16	15–12	11–0
Condition	0 1	#	P	U	B	W	L	r_{base}	$r_{transfer}$	Operand 2

ARM Load and Store Encoding


C:\Users\elsakka2\Desktop\ARM\ex1.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
ex1.asm
Project
Registers

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000000
Mode	Supervisor
States	0
Sec	0.00000000

```
       3:      ADR r2, X
0x00000000  E28F2024  ADD        R2,PC,#0x00000024
       4:         LDR R1,[R2]
0x00000004  E5921000  LDR        R1,[R2]
       5:         LDR R1,[R2],#0
0x00000008  E4921000  LDR        R1,[R2]
       6:         LDR R1,[R2,#0]
0x0000000C  E5921000  LDR        R1,[R2]
       7:         LDR R1,[R2,#0]!
0x00000010  E5B21000  LDR        R1,[R2]!
       8:      ADR r2, Y
0x00000014  E28F2014  ADD        R2,PC,#0x00000014
       9:         STR R1,[R2]
0x00000018  E5821000  STR        R1,[R2]
      10:         STR R1,[R2],#0
0x0000001C  E4821000  STR        R1,[R2]
      11:         STR R1,[R2,#0]
0x00000020  E5821000  STR        R1,[R2]
      12:         STR R1,[R2,#0]!
0x00000024  E5A21000  STR        R1,[R2]!
      13: loop B loop
0x00000028  EAFFFFFE  B          0x00000028
0x0000002C  12345678  EORNES     R5,R4,#0x07800000
0x00000030  87654321  STRHIB     R4,[R5,-R1,LSR #6]!
0x00000034  00000000  ANDEQ      R0,R0,R0
```

```
     AREA various_STR_and_LDR_instructions, code, REA
     ENTRY
     ADR r2, X
     LDR R1,[R2]
     LDR R1,[R2],#0
     LDR R1,[R2,#0]
     LDR R1,[R2,#0]!
     ADR r2, Y
     STR R1,[R2]
     STR R1,[R2],#0
     STR R1,[R2,#0]
     STR R1,[R2,#0]!
loop B loop
X    DCD 0x12345678
Y    DCD 0x87654321
     END
```

To test this program, you need to change the permission of memory-locations from 0x30 to 0x33 (i.e., the location of Y) to make it read/write.

You also need to open a memory window to see the effect of the STR instructions.