

# *Part 9*

## CHAPTER 3

### Architecture and Organization



1

These slides are provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside the class.

All downloaded copies of the slides are for personal use only.

Students must destroy these copies within 30 days after receiving the course's final assessment.

# ARM Load and Store Encoding

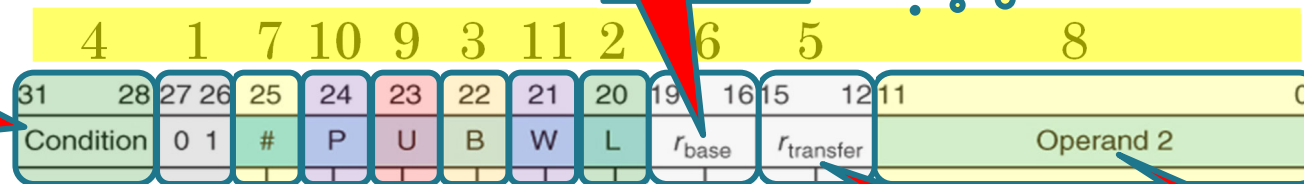
- ❑ The figure illustrate the format of the ARM's load and store instructions.

0 0 Data processing instructions

1 0 1 B / BL instructions

A suggested order of interpreting the LDR/STR fields

See Slide # 83.



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

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.

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.

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

1<sup>st</sup> operand

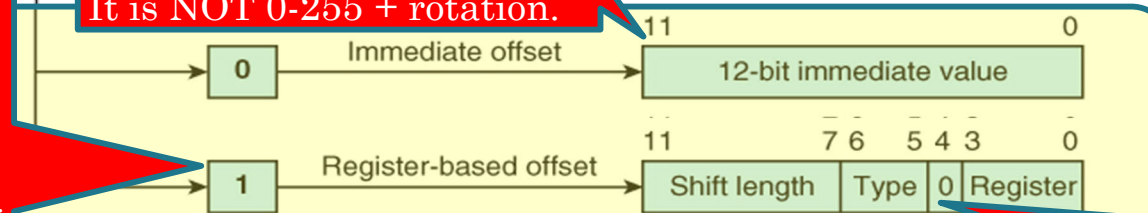
Offset

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

Review Slide # 129.

- 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



Shift type

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

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

❑ Decode the following **ARM** machine code instruction **0x57224106**



# 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 <sub>base</sub>	0010	Base register	r2 is the base (pointer) register
r <sub>transfer</sub>	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





# ARM Load and Store Encoding

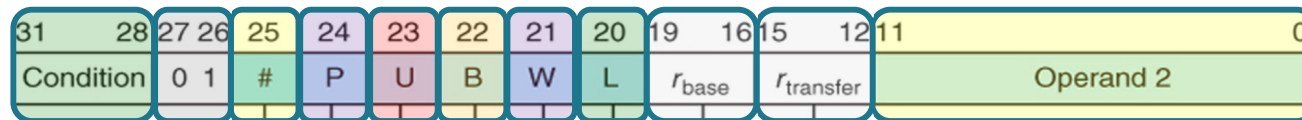
Decoding the ARM Instruction **LDR r1, [r2],r3,ASR#4**

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 <sub>base</sub>	0010	Base register	r2 is the base (pointer) register
r <sub>transfer</sub>	0001	Source/destination	r1 is the destination in this load instruction
Shift length	00100	Shift length	Shift the register 4 places
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]**



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

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.

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

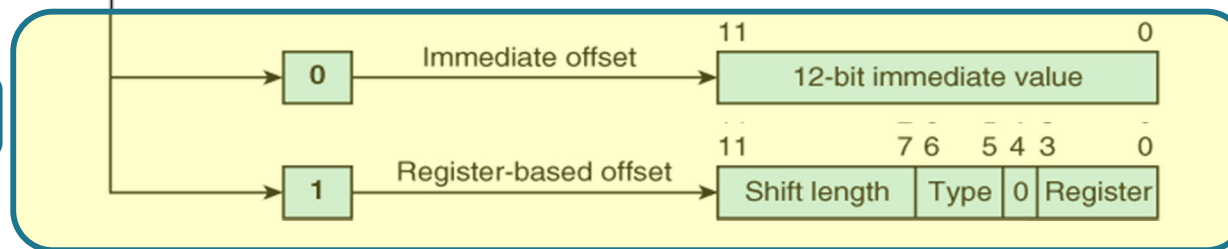
Pointer adjust (Pre/post-increment)

0 = post-index operation: use pointer then adjust  
1 = pre-index operation: adjust pointer then use pointer

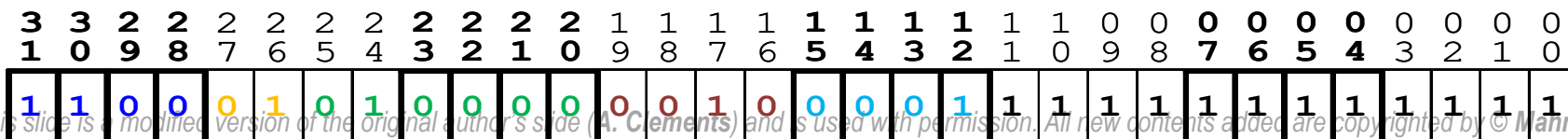
When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

U specifies subtraction or addition of an offset

**0xC5021FFF**



© Cengage Learning 2014



# ARM Load and Store Encoding

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

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 <sub>base</sub>	0010	Base register	r2 is the base (pointer) register
r <sub>transfer</sub>	0001	Source/destination	r1 is the source in this store instruction
Immediate offset	1111111111	Shift length	Offset value = 0xFFFF



# ARM Load and Store Encoding

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



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

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.

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

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

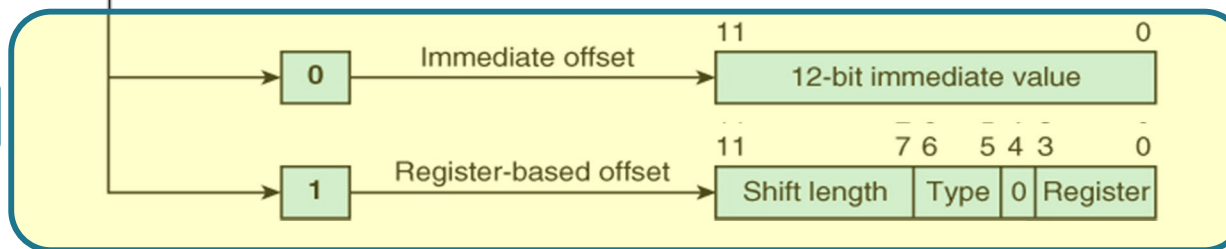
Pointer direction (Up/down)

0 = decrement pointer  
1 = increment pointer

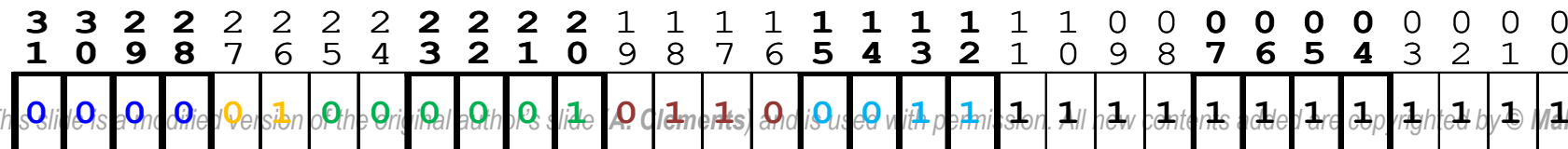
Pointer adjust (Pre/post-increment)

0 = post-index operation: use pointer then adjust  
1 = pre-index operation: adjust pointer then use pointer

**0x04163FFF**



© Cengage Learning 2014



# ARM Load and Store Encoding

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

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 <sub>base</sub>	0110	Base register	r6 is the base (pointer) register
r <sub>transfer</sub>	0011	Source/destination	r3 is the destination in this load instruction
Immediate offset	1111111111	Shift length	Offset value = 0xFFFF

## ARM Load and Store Encoding

- ❑ Encode the following **ARM** instructions.

**LDR R1 , [R2]**

**LDR R1 , [R2] , #0**

**LDR R1 , [R2 , #0]**

**LDR R1 , [R2 , #0] !**

**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

The screenshot displays the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of the processor registers. R0-R15 are all 0x00000000. CPSR is 0x000000D3. SPSR is 0x00000000.
- Disassembly Window:** Shows the disassembly of the program. Instructions include:
  - 3: ADR r2, X
  - 4: E28F2024 ADD R2, PC, #0x00000024
  - 5: E5921000 LDR R1, [R2]
  - 6: E4921000 LDR R1, [R2], #0
  - 7: E5921000 LDR R1, [R2], #0
  - 8: E5B21000 LDR R1, [R2]!
  - 9: E28F2014 ADD R2, PC, #0x00000014
  - 10: E5821000 STR R1, [R2]
  - 11: E4821000 STR R1, [R2], #0
  - 12: E5821000 STR R1, [R2], #0
  - 13: E5A21000 STR R1, [R2]!
  - 14: EAF FFFF B
  - 15: 12345678 EORNES R5, R4, #0x07800000
  - 16: 87654321 STRHIB R4, [R5, -R1, LSR #6]!
  - 17: 00000000 ANDEQ R0, R0, R0
- Source Code Window (ex1.asm):** Shows the assembly code for the program:
 

```

1 AREA various_STR_and_LDR_instructions, code, REE
2 ENTRY
3 ADR r2, X
4 LDR R1, [R2]
5 LDR R1, [R2], #0
6 LDR R1, [R2], #0
7 LDR R1, [R2], #0!
8 ADR r2, Y
9 STR R1, [R2]
10 STR R1, [R2], #0
11 STR R1, [R2], #0
12 STR R1, [R2], #0!
13 loop B loop
14 X DCD 0x12345678
15 Y DCD 0x87654321
16 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.