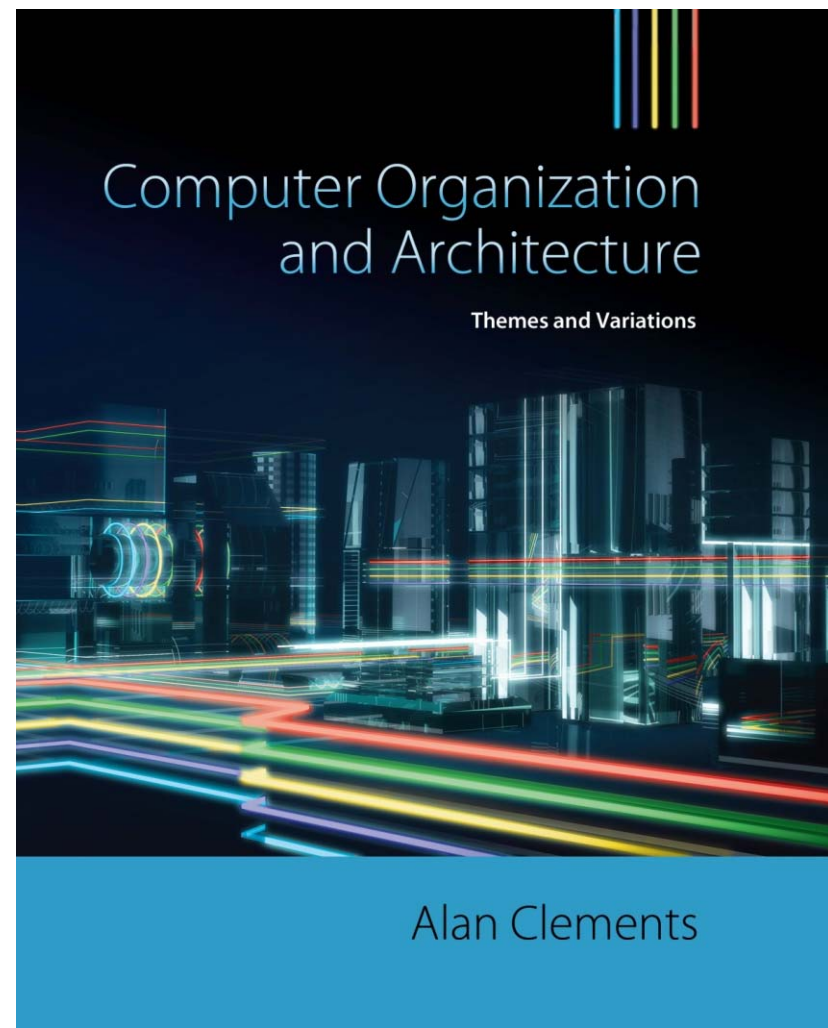


Part 0xA

CHAPTER 3

Architecture and Organization



1

These slides are being provided with permission from the copyright for in-class (CS2208B) use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.

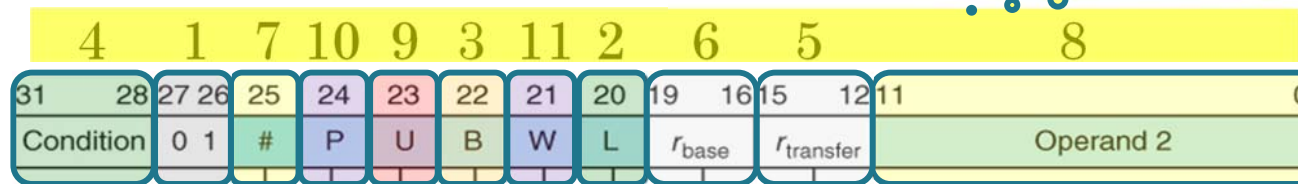
ARM Load and Store Encoding

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

0 0 Data processing instructions

1 0 1 B/BL instructions

The order of interpreting the LDR/STR fields



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 #94.

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

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

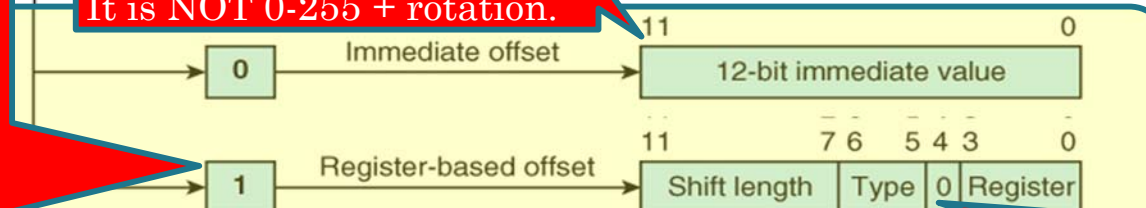
Pointer adjust (Pre/post-increment)

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

Review Slide # 128.



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 _{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

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 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)

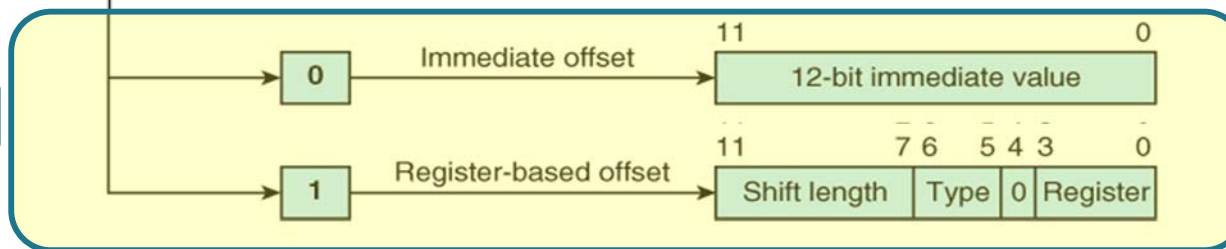
0 = post-index operation: use pointer then adjust

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

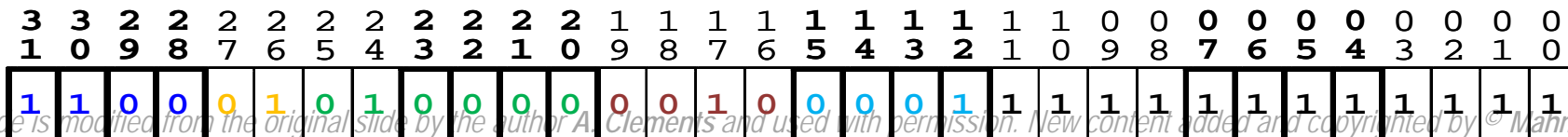
When P=0 (i.e., post-indexed addressing), the write back bit (W) is redundant and is always set to zero.

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 | Update pointer before use |
| 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 | 1111111111 | Shift length | Offset value = 0xFFFF |

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



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

This slide is modified from the original slide by the author A. Clementis and used with permission. New content added and copyrighted by M. Mahdian.

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 | Update pointer before use |
| 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 | 1111111111 | Shift length | Offset value = 0xFFF |

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 LDR 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
```

ARM Load and Store Encoding

The screenshot displays the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly Window:** Shows the disassembly of the program. Instructions include:
 - 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: DCD 0x12345678
 - 15: DCD 0x87654321
 - 16: END
- Source Window (ex1.asm):** Shows the assembly code corresponding to the disassembly. It includes a loop that stores and loads data from memory locations R2 and R2+0.

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.